* Conversion of ASCII to Hexadecimal :-

```
ASSUME CS:CODE , DS: DATA
  DATA SEGMENT
  ASCII-NO DB 36
HEX-NO DB DUP(0)
  DATA ENDS .
  CODE SEGMENT
  START:
        MOV AX , DATA
        MOV DS , AX
      MOV SI , OFFSET ASCII-NO
        MOV BL , [SI]
        SUB BL , 30H
        CMP BL , 0A
          JC GO
        SUB BL , 07 .
        GO:
            MOV HEX-NO , BL .
            MOV AH , 4CH
              INT 21H
            CODE ENDS
              END START .
```

ASCII	HEX
39 - 9	
~~A - 41~~	
41 - A	65 - decimal
42 - 0B	
43 - 0C	
44 - 0D	
45 - 0E	
46 - 0F	
47 - 10	

Gurukul Page No.:
Date: / /

* Conversion of Hex-No to ASCII

→
```
ASSUME CS:CODE , DS:DATA
   DATA SEGMENT
HEX-NO DB 05H
ASCII-NO DB 01DUP(0)
DATA ENDS.
CODE SEGMENT
START:
        MOV AX, DATA
        MOV DS, AX
      MOV SI , OFFSET HEX-NO
        MOV AL, [SI]
        CMP AL, 0A
        JC GO
      ADD AL, 37
GO:   ADD AL, 30
        MOV ASCII-NO, AL
        MOV AH, 4CH
          INT 21H
          CODE ENDS
           END START.
```

* Write an ALP using procedure for performing the operations $Z = (A+B) * (C+D)$

→

```
ASSUME CS:CODE , DS:DATA
DATA SEGMENT
 A EQU 01
 B EQU 02
 C EQU 03
 D EQU 04
SUM DB 01 DUP(0)
 Z DW 01 DUP(0)
DATA ENDS.
CODE SEGMENT
 START:
        MOV AX, DATA
        MOV DS, AX
        MOV AL, A
        MOV BL, B
    Call add-byte.
     MOV SUM, AL
        MOV AL, C
        MOV BL, D.
    call add-byte
     MUL SUM  MOV Z, AX  JMP Exit
add-byte PROC
     ADD AL, BL
        RET
add-byte ENDP
Exit: MOV AH, 4CH
        INT 21H
CODE ENDS
  END START.
```

Gurukul
Date: / /

* Write an ALP using procedure for performing following operation:-

$Z = (A * B) + (C * D)$

→

```
ASSUME CS:CODE , DS:DATA
  DATA SEGMENT
A   EQU 01
B   EQU 02
C   EQU 03
D   EQU 04
MULTIPLICATION  DW 01 DUP(0)
  Z  DW 01 DUP(0)
DATA ENDS
CODE SEGMENT
START:
      MOV AX , DATA
      MOV DS , AX
      MOV AL , A
       MOV BL , B
  call MUL-byte.
   MOV MULTIPLICATION , AX
      MOV AL , C
       MOV BL , D.
  call MUL-byte.
      ADD AX , MULTIPLICATION.
       MOV Z , AX
      jmp exit
MUL-byte proc
      MUL BL
      RET
      ENDP.
```

```
Exit: MOV AH,4CH
      INT 21H
CODE ENDS
  END START.
```

* Write an ALP for 1's complement:-

→

```
ASSUME CS:CODE , DS:DATA
   DATA SEGMENT
NUM DB 25H
ONESCOM DB 01 DUP(0)
DATA ENDS.
CODE SEGMENT
START:
        MOV AX,DATA
        MOV DS,AX
        MOV AL,NUM
          NOT AL
      MOV ONESCOM,AL
        MOV AH,4CH
          INT 21H
        CODE ENDS
          END START.
```

Gurukul Page No.:
Date: / /

* WAP for factorial of given no. by using procedure:-

→
```
ASSUME CS:CODE , DS : DATA
  DATA SEGMENT
NUM DB 04H
FACTORIAL DB 0L DUP(0)
  DATA ENDS
  CODE SEGMENT
START:
      MOV AX, DATA
       MOV DS, AX
       Call FACTORIAL
     MOV FACTORIAL , AL

     FACTORIAL PROC
     MOV AL, NUM
      MOV CL, AL
        DEC CL
BACK:
       MUL CL
        LOOP BACK
       RET
      ENDP
      MOV AH, 4CH
       INT 21H
      CODE ENDS
      END START
```

Gurukul Page No.:
Date: / /

Assembly Language programming Tools:-

1] Assembler:-

It is a programming tool used to convert an Assembly language program statements into machine level language program.

The different Assemblers for 8086 families are
(i) TASM - (Turbo Assembler)
(ii) MASM - (Microsoft MACRO Assembler)
(iii) NASM - (Netwide Assembler)

The ALP assembly language program is in the form of Mnemonics (english like words).

To make programming easier, programmers write programs in assembly language.

Then Assembler translated this program into machine level language program.

ALP program file saved with filename.asm extension.

Ex: ADDITION.ASM.

2] Editor :-

It is a program which allows user to write source program, and edit ALP program statements.

By using an editor we can create, edit, save, copy and make modification in the source file.

The file created by the editor is an ASCII format & is known as source file.

ex: Edit FILENAME.ASM
ex: Edit ADDITION.ASM

3] Linker :-

It is a programing tool used to convert object code / program into executable code / program.

~~It requires~~ A linker is a program which links smaller program together to form a large program.

It is also used to link subroutine with main program.

It links object code (machine code) program.

Gurukul Page No.: Date: / /

When ALP program is error free, the next step is to link the object modules (small program).

It is used to combines assembled modules into one executable module.

It generates .exe module.

TLINK FILENAME.OBJ

ex:

TLINK ADDITION.OBJ

4] Debugger:-

It is a programming tool used for finding / locating and correcting errors.

It allows user to test and debug programs.

The program can be tested in a single step.

It allows the execution of .exe programs in single step mode.

F7 key is used to test program in single step in TASM.

Syntax:

TD FILENAME.EXE

or

TD FILENAME

ex: TD ADDITION.EXE

or

TD ADDITION.

Gurukul Page No.: Date: / /

Directives:-

Directive is a statement to give direction to the assembler to perform the task of assembly process.

An assembler supports directives to define data, to define MACRO, etc.

ALP consist of 2 types of statements i.e instructions & directives.

Rules for symbols, variables & constants:-

symbols consist of following characters:-

(i) Upper case & lower case alphabets. (A to Z, a to z)

(ii) Digits (0-9)

(iii) Special characters ($, &, @, _)

* A variable can use the following characters:

(i) A to Z, a to z, 0-9, @, & _)

* A digit cannot be used as the 1st character for variable.

* $ & & are not used in a variable - because they are used for other purpose.

★ Division of 16 bit by 8 bit no.

→

```
ASSUME CS: CODE , DS: DATA
 DATA SEGMENT
DIVIDENT DW 1000H.
 DIVISOR  DB 05H.
 QUOTIENT DB 01 DUP(0)
 REMAINDER .DB 01DUP(0).
DATA ENDS
 CODE SEGMENT.
 START:
        MOV AX, DATA
         MOV DS, AX.
       MOV AX, DIVIDENT.
       MOV BL, DIVISOR.
         DIV BL.
       MOV QUOTIENT, AL
        MOV , REAMAINDER, AH.
        MOV AH, 4CH
          INT 21H.
       CODE ENDS
          END START.
```

Gurukul | Page No. : | Date: / /

* Division of 32 bit by 16 bit number :-

→

```
ASSUME CS:CODE , DS:DATA
 DATA SEGMENT
DIVIDENTL DW 4444H
DIVIDENTH DW 4444H
 DIVISOR DW 2222H.
 QUOTIENT DW 0DUP(0)
 REMAINDER DW 0DUP(0)
DATA ENDS.
CODE SEGMENT
 START:
      MOV AX, DATA
       MOV DS, AX
     MOV DX, DIVIDENTL
     MOV AX, DIVIDENTH
     MOV BX, DIVISOR
     DIV BX
   MOV QUOTIENT, AX
    MOV REMAINDER, DX.
    MOV AH, 4CH
      INT 21H
      CODE ENDS.
      END START.
```

☆ 8 bit multiplication

→

```
ASSUME CS:CODE ,DS:DATA
 DATA SEGMENT
NUM1. DB  08H
NUM2  DB  04H.
RESULT DW  01 DUP(0)
DATA ENDS
CODE SEGMENT
 START:
        MOV AX,DATA
         MOV DS,AX
        MOV AL,@NUM1
        MOV BL,NUM2
         MUL BL
        MOV RESULT,AX
        MOV AH,4CH
         INT 21H
         CODE ENDS
          END START.
```

Directives :-
The variable must begin with Alphabet or an underscore (_).

Their is no destinction between an upper case & lower case letter.

A constant value may be binary, decimal or hexadecimal number.
To distinguish them symbols, B, D, H are used at the end of a binary, decimal and hexadecimal no. respectively.

A no. without identification symbol is taken as decimal number.

A hexadecimal no. with its first digit between A to F, must begin with zero (0); otherwise it will taken as a symbol.

ex: A hexadecimal no. A5B6 must be written as 0A5B6H.

Directives :-

1) **DB: (Define byte) :-**

This directive informs to the assembler that given variable and its value is of type byte.

It reserve 8 bit for given variables value.

If their are multiple values define (array) then 8 bit (1 byte) memory space reserved for each value.

General format :-

NAME OF VARIABLE DB value(s)

ex:

```
NUM1 DB 15H.
NUMLIST DB 01H, 02H, 03H, 04H, 05H.
RESULT DB 5DUP(0).
```

NUMLIST DB 05H, 09H, 4'DUP(7), 06H, 0AH :-

This directive informs to the assembler to reserve 8 bytes of consecutive memory space / location i.e 05H, 09H, 07H, 07H, 07H, 07H, 06H, 0AH.

4 DUP(7) tells assembler to duplicates 7 value 4 times.

2] DW :- (Define Word) :-

This directive informs to the assembler that given variable & its value is of type word. (16 bits)

It occupies 2 bytes of memory space. It defines word type variable.

If there are multiple values, 2 bytes (16 bits) of memory space are reserved for each value.

* General format :-

NAME of VARIABLE DW value / values.

ex:

NUM1 DW 1234H.
NUMLIST DW 1234H, 432LH, 2345H
SUM DW ? / SUM DW 0 DUP(0).

* NUMLIST DW 1234H, 12H, 4892H, 68H

In above example. for each value it allocates 2 bytes.

Gurukul Page No.: Date: / /

The value which have only two digits, will be placed in the first bytes of memory space & second byte will contain zeros(0's) i-e 1234H, 0012H, 4892H, 0068H.

ex: NUMBER DW '25$' or NUMBER DW "25$"

Here value is string. hence it take ASCII code of 2 i.e 32.

& ASCII code of 5 is 35.

Hence, two bytes of memory space contain 3235 H

* DD:-(Define Double word)

This directive informs to the assembler that The variable is of type double word (4 byte type of variable).

It occupies 4 byte of memory. if their are multiple values, 4 bytes of memory are reserved for each value.

Syntax:

variable DD initialvalue(values)

ex:

NUM1 DD 12341234H.

4] DQ: (Define quadword) :-

The directive DQ defines a quadword type variable.

It occupies 8 bytes of memory. (quad word type variable).

If there are multiple values, 8 bytes of memory space are reserved for each value.

syntax:

NAME of variable DQ initial value (values)

ex: :- NUM1 DQ 1234 1234 1234 1234 H.

5] DT: (Define Tenbyte) :-

The directive DT defines a variable of type 10 bytes.

It can be used to define single or multiple ten (10) byte variable.

syntax:

variable DT value (values)

ex: NUM1 DT 1234 1234 1234 1234 1234 H

6] DUP :-

DUPlicate operator allocates no. of multiple occurences of a value.

It allocates space for variables in memory location depends upon the type of directive.

ex: ARRAY DB 40 DUP(0)
RESULT DW 0L DUP(0).
QUOTIENT DB 0L DUP (?)

* Program SEGMENT :-

The segment register must initialized in a program before they are used for code, data & stack.

The segment directive for code, DATA & stack may be placed in any order in program.

① CODE or .CODE :- (CODE-HERE):

This directive identifies the start of code

The code segment identified (addressed by) CS & IP register.

Gurukul | Page No. :
Date: / /

(ii) .DATA or .DATA :- [DATA-HERE]

The DATA directive identifies the start of data segment. where variables are declared.

The data segment is addressed by DS & SI or DI register.

(iii) STACK or .STACK [STACK-HERE]

STACK directive sets the space for the use of program which may be of any size upto 64kB.

It is addressed by SS & SP or BP register.

(iv) ENDS:-

This directive informs to the assembler that end of segment.

The name of segment is used as the prefix of the ENDS directive.

Syntax:

SEGMENTNAME ENDS.

ex: DATA ENDS.
CODE ENDS
EXTRA ENDS
STACK ENDS.

ex: DATA SEGMENT

DATA ENDS.

(v) END :-

This directive informs to the assemblee that the end of program module.

It terminates the assembly language program.

This is used after the last statement of a program module.

It ignores the instructions (statements) after the END directive.

Therefore, the programmee should use END directive at the every end of program module.

ex: Syntax:

END LABEL.

ex:

END START.

vi] **ENDM :- (End of MACRO) :-**

This directive is used to inform to the assemblee that this is the END of MACRO.

ex:

```
PRODUCT MACRO
=============
=============
=============
ENDM.
```

Here, PRODUCT is the name of MACRO.

ASCII
41 – A
⋮
5A – Z

61 – a
⋮
7A – z

Gurukul Page No.:
Date: / /

* Write an ALP to convert upper case to lower case (string).

→
```
ASSUME CS:CODE ; DS:DATA
DATA SEGMENT
UPPERCASE DB 'A'
LOWERCASE DB 0LDUP(0)

DATA ENDS
CODE SEGMENT
START:
        MOV AX , DATA
        MOV DS , AX
       MOV AL , UPPERCASE
        ADD AL 120
       MOV LOWERCASE , AL
        MOV AH , 4CH
          INT 21H
          CODE ENDS
          END START.
```

* ASSUME :-

ASSUME directive is used to tell the assemblee the name of the logical segment. it should use for the specific segment.

In an ALP, each segment is given a name by the programmer.

ex: ASSUME CS:CODE , DS:DATA

The above directive tells the assemblee that the name of code segment is code. & name of the data segment is DATA.

The CODE contains the machine code of the instruction (instructions of a program).
and DATA contains data of the program which is being executed.

In ALP programmer gives a name to each segment is given below:-

* CODE SEGMENT :-
* CODE OF CODE_SEGMENT OF MY-CODE

* DATA SEGMENT :-

DATA OF DATA-SEG OF MY-DATA.

* **EQU :- (Equate) :-**

It is used to give a name to certain value or symbol.

The equal sign directives creates an absolute symbol for assigning the numeric expression to a name.

ex:

```
NUM1 EQU 07H.
```

It tells the assembler to insert the value 07H every times that it finds a name NUM1 in program statement.

Gurukul Page No.:
Date: / /

* Write an ALP to convert lower case to upper case :-

→

```
ASSUME CS:CODE , DS:DATA
DATA SEGMENT
LOWERCASE DB 'a'
UPPERCASE DB 01 DUP(0)
DATA ENDS.
CODE SEGMENT
START:
        MOV AX , DATA
        MOV DS , AX
        MOV AL , LOWERCASE
        SUB AL , 20
        MOV UPPERCASE , AL
        MOV AH , 4CH
        INT 21H
        CODE ENDS
        END START.
```

Gurukul Page No.: Date: / /

* EVEN :-

This directive informs the assembler to increment the content of location counter to the next memory address if it already not at an EVEN address.

It is used to both code segment & data segment.

The content of location counter holds address of the memory location assigned to an instruction or variable or constant.

When it is used in a data segment, the location counter is simply incremented to EVEN memory address, if it is already not on an even address.

When it is used in a code segment, the location counter is incremented to the next even memory address, if it is already not on an even address.

The 8086 µp takes 1 bus cycle to read a word from an even memory address, but it takes 2 bus cycles to read a word from an odd address.
Therefore, a series of consecutive memory words can be read much more quickly if their are at even memory addresses.

ex: NUMBER DB 25H
EVEN
DATA-ARRAY DW 50 DUP(0).

In above example variable named data-array start at even memory address.
Hence, it require only one bus cycle to read each element of the variable DATA-ARRAY.

Hence, total read time is much less.

★ EXTERN (External) Directive:-

This directive informs to assembler that the names, procedures & labels following this directive has already will define in some other assembly language module program.

The names, procedures & labels declared as external in one program module must be declared public using PUBLIC directive in the program module in which they have been define.

ex:

```
PRM1 SEGMENT
PUBLIC TEMP-CONTROL
  PRM1 ENDS
PRM2 SEGMENT
EXTERN TEMP-CONTROL :FAR
  PRM2 ENDS.
```

In above example PRM is program module.

general format:-

EXTERN Variablename : TYPE of variable

Gurukul | Page No.: | Date: / /

* LENGTH :- (length) :-

It is an operator to determine no. of elements in a data attempt such as an array or a string.
Variables are defined as follows:-

ex:

```
DATA SEGMENT
 ARRAY DW 10 DUP(0)
 NUM-LIST DB 10 DUP(0)
 DATA ENDS.
```

* The length operator can be used as follows:-

(i) MOV CX, LENGTH ARRAY
(ii) MOV CX, LENGTH NUM_LIST.

In above examples length of ARRAY & NUM_LIST both are 10.
So, ~~10 is moved~~ MOV no. 10 in CX register.

* OFFSET :- (offset) :-
It is an operator to determine the offset of a variable or procedure. with respect to the base of the segment. which contains the named variable or procedure.

The operator can be used to load register with the offset of a variable.

* The variables can be defined as follows:-

ex: MOV SI, OFFSET ARRAY.
~~MOV AX, [BX+SI]~~
MOV SI, OFFSET NUM-LIST.

* PTR (Pointer):-
It is an operator to indicate the type (BYTE, WORD, DWORD) of a variable or label.

It specifies the exact size of operand.

ex: MUL BYTE PTR [SI]

(i) It specifies that multiplier is a byte operand.

(ii) The operator PTR is also used to override the declared type of variable
for example:-
NUMBER DW 1234H.

Now, if we write the instruction.
MOV AL, BYTE PTR-NUMBER, only the lower byte of the variable NUMBER will move to the register AL (34).

Gurukul Page No.: Date: / /

* SIZE (size):-
It is an operator to calculate the number of bytes allocated to the data attempt.

It differes from the LENGTH operator. as LENGTH operator calculates the no. of elements in data attempt.
Variables are defined as follows:-

ex:

```
DATA SEGMENT
NUMBERS DB 10 DUP(0)
ARRAY DW 10 DUP(00)
DATA ENDS
```

The size of operator can be used as follows:-
(i) MOV CX, SIZE NUMBERS.
(ii) MOV CX, SIZE ARRAY.

In example (i) no. 10 is moves to register CX, as 10 bytes are allocated to the variable numbers.

In exg → (ii) no. 20 is moved to register CX as 20 bytes are allocated to the variable ARRAY.

* **Procedure:-**

Procedure is a group of instructions in a large program can be written seperately from the main program.

This subprogram is called as procedure.

Procedure is a set of statements that can be processed seperately (Independently) from the main program.

for defining procedure. PROC & ENDP directives are used.

PROC indicates the begining of procedure. & ENDP directive indicates the END of procedure.

It must be defined in code segment only.

Advantages:-

1] Large programs can be divide into smaller modules.

2] It Reduce the size of program.

3] It is easy to debug the program.

4] It reduce work load & development time.

Gurukul Page No.
Date: / /

5] It reuse many times in a same program or in another program.

Disadvantages:-

1] Extra code is required to integrate procedure. i.e CALL & RET instruction.

General syntax:

Procedure-name PROC

Procedure statements.

Procedure-name ENDP.

ex: ADDITION PROC

Procedure statements.

ADDITION ENDP.

* MACRO :-

Small sequences of codes of the same pattern repeated frequently at different places which perform the same operation on the different data of the same data type are called MACRO.

It is also called open sub-routine.

When called, the code written within MACRO are executed automatically.

MACRO's should be used when it has few program statements.

This simplifies the programming process.
For MACRO's that you want to include in your program you must first define them.

The assembler directive 'MACRO indicates beginning of the MACRO & ENDM directive indicates end of the MACRO.

Syntax:

Name of MACRO MACRO [Arg1, Arg2, ... Argn]

Program statements

MACRO Name ENDM

ex: * ALP for to find the square of a number.

→
```
SQUARE MACRO NUM
    MOV AL, NUM
    MUL AL
    MOV BX, AX
ENDM.
DATA SEGMENT
  N DB 05H
  R DB 01 DUP(0)
DATA ENDS.
CODE SEGMENT
START:
        MOV AX, DATA
        MOV DS, AX
        SQUARE N ; call macro square.
        MOV R, BX
        MOV AH, 4CH
        INT 21H
        CODE ENDS
        END START.
```

Gurukul Page No.: Date: / /

* Write a program for addition of two numbers using MACRO ADD-num.

→
```
ADD-NUM MACRO no1, no2, result
   MOV AL, no1
   MOV BL, no2
   ADD AL, BL
   MOV result, AL
   ADD-NUM ENDM
DATA SEGMENT
   NUM1 DB 25H
   NUM2 DB 45H
   RES DB 01 DUP(0)
   DATA ENDS
CODE SEGMENT
START:
      MOV AX, DATA
      MOV DS, AX
   ADD-NUM NUM1, NUM2, RES
      MOV AH, 4CH
      INT 21H
      CODE ENDS
         END START.
```

* Using MACRO's, write ALP to solve $P = x^2 + y$

→

```
SQR-NUM MACRO NOL, SQE
    MOV AL, NOL
    MUL AL
    MOV SQE, AX
    ENDM
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
    X  DB 04H
    Y  DB 05H
    SX DB 01 DUP(0)
    SY DB 01 DUP(0)
    P  DW 01 DUP(0)
DATA ENDS.
CODE SEGMENT
START:
    MOV AX, DATA
    MOV DS, AX
SQR-NUM MACRO X, SX
SQR-NUM MACRO Y, SY
    MOV AX, SX
    ADD AX, SY
    MOV P, AX
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END START.
```

Gurukul Page No. : Date: / /

☆ Re-entrant procedure :-

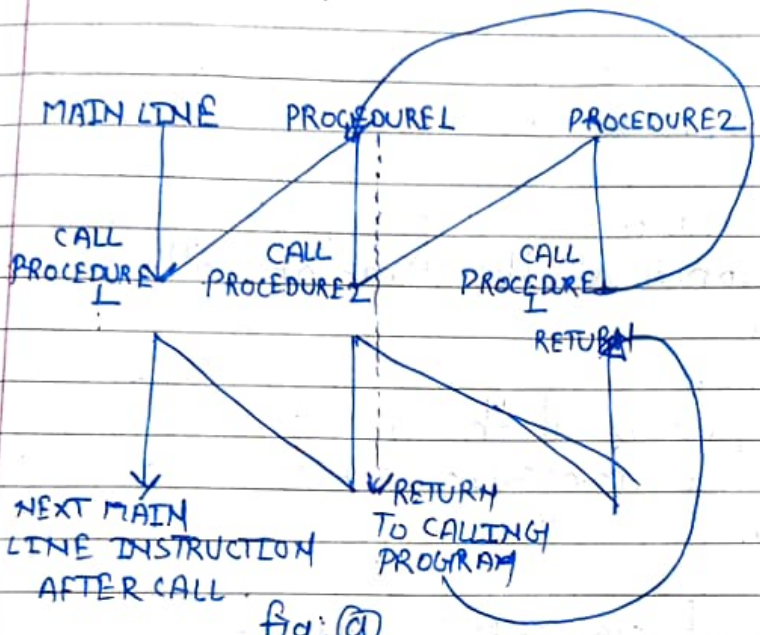


fig: (a)

In some situation it may happen that (i) procedure1 is called from main program.

(ii) Procedure2 is called from procedure1.

(iii) & procedure1 is again called from procedure2

In this situation program execution flow reenters in the procedure1.

These types of procedures are called re-entrant procedures.

The RET instruction at the end of procedure 1 returns to procedure 2.

The RET instruction at the end of proc 2 will return execution to proc 1.

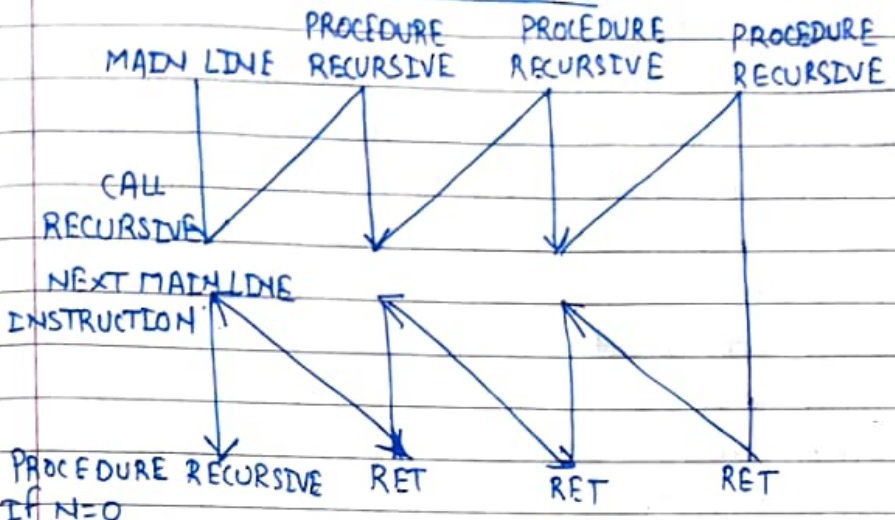
Proc 1 will again executed from where it had stopped at the time of calling procedure 2.

RET instruction at the end of this will return the program execution to main program.

The flow of program execution for re-entrant procedure is shown in fig (a)

Gurukul Page No.:
Date: / /

☆ Recursive Procedure :-



A recursive procedure is a procedure which calls itself.

Recursive procedure are used to work with complex data structures called trees.

If the procedure is called with N (recursion depth) = 3 then the N is decremented by 1. after each procedure call & the procedure is called until N=0.

Fig- shows recursive procedure.

Gurukul Page No. :
Date: / /

* Write an ALP for comparing two strings of 10 byte each:-

→
```
ASSUME CS:CODE , DS:DATA , ES:EXTRA
DATA SEGMENT
STL DB "TESTING PG"
ST2 DB "TESTING PG"
  L DB 0AH
  R DB ?
DATA ENDS.
CODE SEGMENT
START:
        MOV AX, DATA
        MOV DS, AX
        MOV ES, AX
        MOV CL, L
        CLD
    MOV SI, OFFSET STL.
    MOV DI, OFFSET ST2
     REPE CMPSB
      JNZ NEQ
     MOV R, 'E'
     JMP LAST
   NEQ:
       MOV R, 'N'
   LAST:
       MOV AH, 4CH
           CODE ENDS
           END START.
```

'$' - It is used to terminate the string.
'0D' → Being cursor to next line. (line feed character)
'0A' → Being cursor to next position (column wise).
(carriage feed character).

---

* WAP to display the string on the CRT screen of a micro-computer:-

→
```
ASSUME CS:CODE , DS:DATA
DATA SEGMENT
MESSAGE DB "MICROPROCESSOR", "$"
DATA ENDS.
CODE SEGMENT
START:
        MOV AX , DATA
        MOV DS , AX
        MOV AH , 09H
MOV DX , OFFSET MESSAGE | LEA DX , MESSAGE
        INT 21H
      MOV AH , 4CH
        INT 21H
      CODE ENDS
        END START.
```

* Display string by using MACRO :-

→

```
ASSUME CS:CODE , DS:DATA
DISP MACRO MSG1
     PUSH AX
     PUSH DX
     MOV AH, 09H
     MOV DX, OFFSET MSG1
     INT 21H
     POP DX
     POP AX
     ENDM
DATA SEGMENT
STR DB ODH, OAH, "MICROPROCESSOR", "$"
DATA ENDS
CODE SEGMENT.
START: MOV AX, DATA
       MOV DS, AX
       DISP STR
       MOV AH, 4CH
       INT 21H
       CODE ENDS
       END START.
```

Gurukul Page No.: Date: / /

* Write a program to display the string & take input from keyboard :-

→
```
ASSUME CS:CODE , DS: DATA
& DATA SEGMENT
STR1 DB 'ENTER STRING' , '$'
STR2 DB 50 DUP($)
STR3 DB 0DH , 0AH , '$'
  DATA ENDS.
CODE SEGMENT
START: MOV AX , DATA
  MOV DS , AX
  LEA DX , STR1
  INT 21H
```
} For displaying enter string.
```
  MOV AH , 0AH
  LEA DX , STR2
  INT 21H
```
} For taking ilp from keyboard.
```
  MOV AH , 09H
  LEA DX , STR3
  INT 21H
```
} For displaying newline.
```
  MOV AH , 09H
  LEA DX , STR2+2
  INT 21H.
```
} For displaying what you have entered.
```
  MOV AH , 4CH
  INT 21H
CODE ENDS
  END START.
```

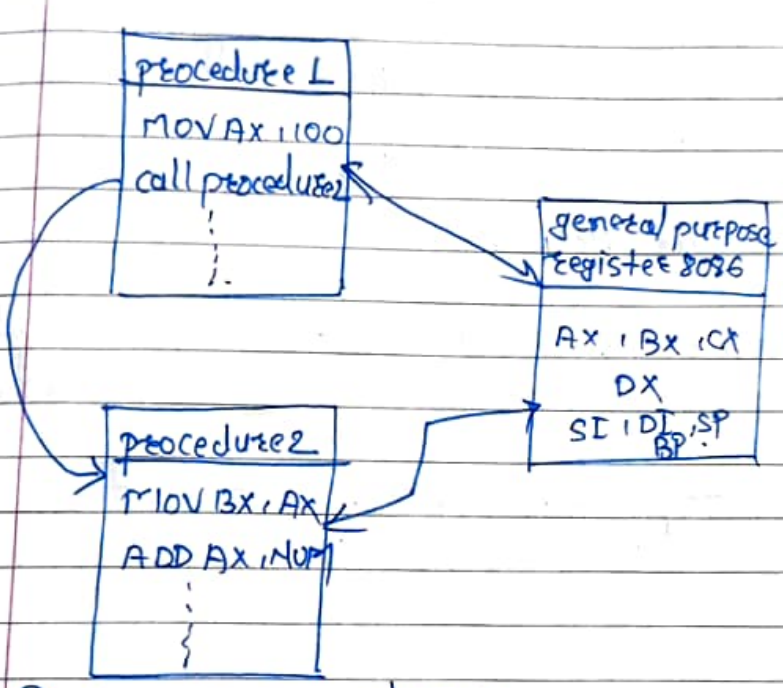
Gurukul Page No.
Date: / /

* Write a program for concatination of two strings :-

→
```
ASSUME CS:CODE, DS: DATA , ES: EXTRA
DATA SEGMENT
STR1 DB "CM", "$"
STR2 DB "401", "$"
DATA ENDS.
CODE SEGMENT
START:
        MOV AX, DATA
        MOV DS, AX
        MOV ES, AX
        CLD
MOV SI, OFFSET STR1
MOV DI, OFFSET STR2
    ADD DI, 02  } Adding the length of destination
    MOV CX, 02  } string to DI.
REP MOVSB  } This will concat 2 strings.
MOV AH, 4CH
INT 21H
CODE ENDS
END START.
```

two

* Passing Parameters :- The process of sending data / parameter to a procedure while calling proce- in the main program is known as passing parameters
* Passing parameters through registers to procedure



** Passing Parameters :-
Parameters can be passed between procedures in any of three ways.
(i) Through general purpose register.
(ii) In an Argument list.
(iii) on the stack.

① Through gen register :-

The processor does not save the state of general purpose register.

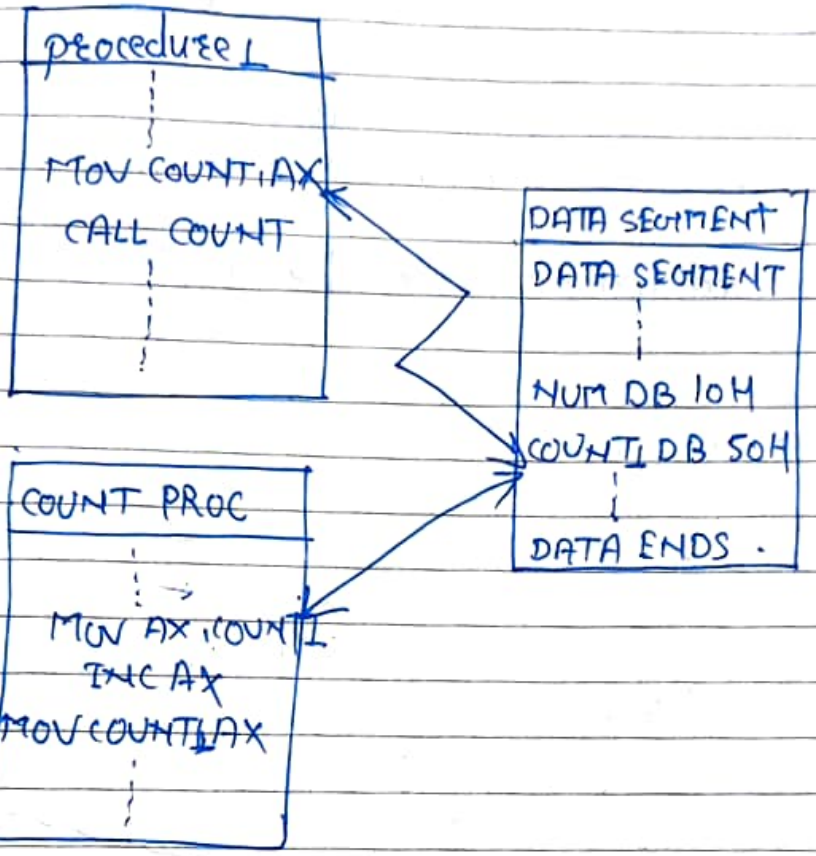
The process of sending data / parameter to a procedure using general purpose register like AX, BX, CX, DX while calling the procedure in the main program is known as passing parameters through the register to the procedure.

The data tobe processed is stored in registers & these registers accessed by the procedure.

* In above example AX register passed the data to procedure 2.

Gurukul | Page No.: | Date: / /

2] Passing parameters in an Argument list :-



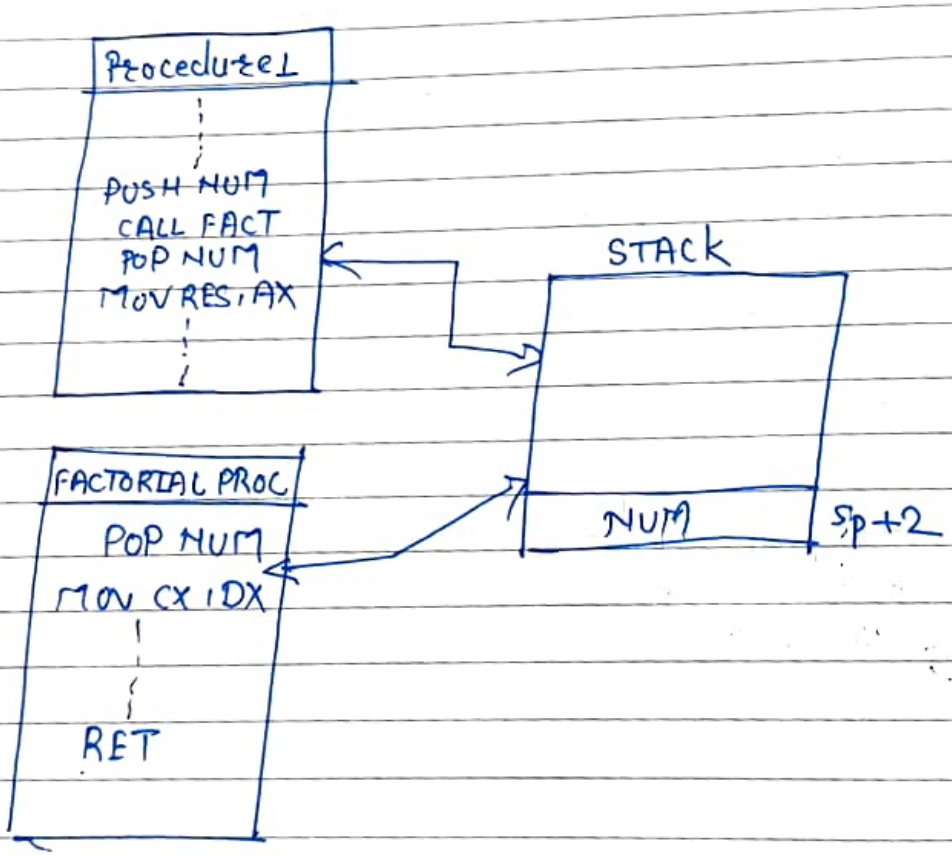
Process of sending data to a procedure using variables while calling the procedure in the main program is known as passing parameters in an argument.

A method of passing a larger no. of parameters (or a data structures) to called procedure is to place the parameters in an argument list in one of the data segments in memory.

A pointer to the argument can then be passed to the called procedure through a general purpose register or the stack.

Parameters can also be passed back to the calling procedure in this same manner.

3] Passing parameters on the stack:-



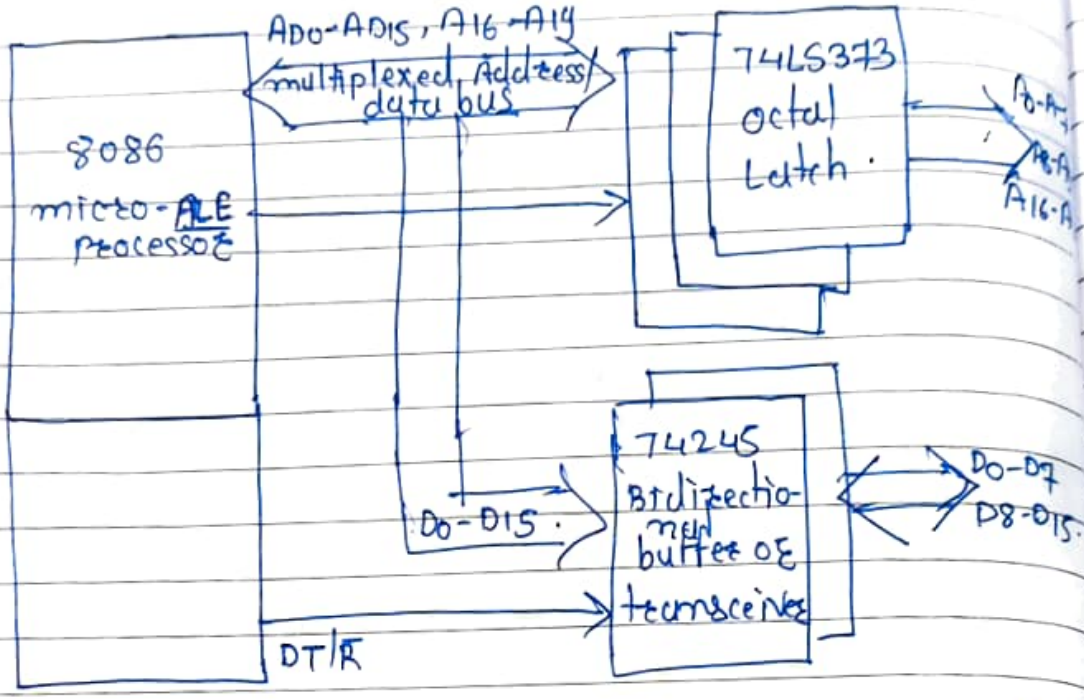
Gurukul | Page No.: | Date: / /

Process of sending data to a procedure using stack memory. While calling the procedure in the main program is known as passing parameters using stack memory to procedure.

The data is to be processed is stored (PUSH) in the stack memory locations. & these memory locations are accessed in the procedure to process the data.

Gurukul Page No.
Date: / /

* Demultiplexing of Address & data bus :-



In 8085 data & address bus are multiplexed. i.e $AD_0 - AD_7$.

In 8086 $A_0 - A_{15}$ address lines multiplexed with data line $D_0 - D_{15}$.

In 1st clock cycle (TL) address / data bus is used for to carry address. & in remaining clock cycle (T_2, T_3, T_4) acts as a data bus.

The ALE signal is applied to octal latch to seperate address & data bus.

Gurukul Page No.:
Date: / /

In 1st clock cycle ALE remains high. When it is high, (ALE=1) address is latched in octal latch & we get address lines as a output.

In 8086 there are total 20 address lines are used. Hence 1st octal latchs output A0-A7. & o/p of 2nd octal latch A8-A15. & o/p of 3rd octal latch ~~A16-A19~~ only 4 address lines i.e A16-A19.

When ALE is low (ALE=0) address/data bus is used for to carry data. and the data is transmitted or received by using transceiver or bidirectional buffer.

Fig. shows Demultiplexing address & data bus.

Date: / /

- Write an ALP to find out number of positive and number of negative numbers from given series of signed numbers (8 bit)

```
ASSUME CS: CODE, DS: DATA
 DATA SEGMENT
LIST DB 83H, 64H, 10H, 1BH, 20H
 POSITIVE DB 1 DUP(0)
NEGATIVE DB 1 DUP(0)
 DATA ENDS
CODE SEGMENT
START: MOV Ax, DATA
       MOV DS, Ax
       XOR Ax, Ax
       MOV CL, 05H
       MOV SI, OFFSET LIST
  GO: SHL/ROL [SI] 01H
       JC N
       INC AL
       JMP NEXT
   N: INC BL
Next: INC SI
       DEC CL
       JNZ GO
      MOV POSITIVE, AL
      MOV NEGATIVE, BL
      MOV AH, 4CH
       INT 21H
      CODE ENDS
      END START
```

(8 bit)

- Write an ALP to count number of even number and number of odd numbers in given array

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 01H, 02H, 03H, 04H, 05H
EVEN NUMBER DB 01 DUP(0)
ODD NUMBER DB 01 DUP(0)
RESULT DB 01 DUP(0)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       XOR AX, AX
       MOV CL, 05H
       MOV AL, 00H
       MOV BL, 00H
       MOV SI, OFFSET LIST
GO: SHR [SI], 01H
       JC DOWN
       INC AL
UP: INC SI
       DEC CL
       JNZ GO
DOWN: INC BL
       JMP UP
       MOV EVEN NUMBER, AL
       MOV ODD NUMBER, BL
       MOV AH, 4CH
       INT 21H
       CODE ENDS
       END START
```

- Difference between minimum mode and maximum mode.

Sr. No	Minimum mode	Maximum mode
1.	MN/$\overline{MX}$ pin is connected to Vcc i.e. MN/$\overline{MX}$ = 1.	MN/$\overline{MX}$ pin is connected to gro MN/$\overline{MX}$ = 0
2.	control system M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ is available on 8086 directly.	control system M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ is available direct in 8086
3.	Single processor configuration in minimum mode system	Multiprocessor configuration in maximum mode system.
4.	In this mode, no separate bus controller is required	Seperate bus controller (8288) is required in maximum mode.
5.	control signal such as $\overline{IOR}$, $\overline{IOW}$, $\overline{MEMW}$, $\overline{MEMR}$ can be generated using control signals M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ which are available on 8086 directly	control signal such as $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$ and $\overline{AIOWC}$ are generated by bus controller 82
6.	ALE, $\overline{DEN}$, DT/$\overline{R}$ and $\overline{INTA}$ signals are directly available	ALE, $\overline{DEN}$, DT/$\overline{R}$ and $\overline{INTA}$ sign are not directly av able and generate

7.	HOLD and HLDA signals are available to interface another master in system such as DMA controller	by bus controller 8288 $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$ signals are available to interface another master in system such as DMA controller
8.	Status of the instruction queue is not available	Status of the instruction queue is available on pin QS_0 and QS_1.

- Write an ALP to count number of zeros in given number.

```
ASSUME CS:CODE , DS:DATA
DATA SEGMENT
NUM DW 2345H
RESULT DB 1 DUP(0)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV BL, 00H
       MOV AX, NUM
       MOV CL, 10H
UP:    SAL AX, 01H / ROL AX, 01H
       JNC GO
       JMP Next
GO:    INC BL
Next:  DEC CL   } LOOP UP
       JNZ UP
       MOV RESULT, BL
```

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

- Write an ALP to ~~exchange~~ transfer of 2 blocks of data from array1, array2.

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H
LIST2 DB 8 DUP (0)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV CL, 08H
       MOV SI OFFSET LIST1 / LEA SI, LIST1
       MOV DI OFFSET LIST2 / LEA DI, LIST2
BACK : MOV AL, [SI]
       MOV [DI], AL
       INC SI
       INC DI
       DEC CL
       JNC back
       MOV LIST2 [DI]
       MOV AH, 4CH
       INT 21H
       CODE ENDS
       END START
```

- Write an ALP multiply the content of Bx by 6 using shift instruction.

```
ASSUME CS:CODE, DS:DATA
 DATA SEGMENT
MULTIPLICAND DW 6060H
 RESULT DW 1 DUP(0)
 DATA ENDS
 CODE SEGMENT
 START: MOV AX,DATA
        MOV DS,AX
        MOV BX, MULTIPLICANT
        SAL BX,02H
        MOV RESULT,BX
        MOV AH,4CH
        INT 21H
 CODE ENDS
 END START.
```

- Difference between Near procedure and Far procedure

Sr.NO	Near procedure	Far procedure
1.	A near procedure refers to a procedure which is in the same code segment from that the call instruction	A far refers to a procedure which is in the different code segment from that of the call instruction.
2.	It is also called intra-segment procedure	It is called inter-segment procedure call.

3	A near procedure call replaces the old IP with new IP	A far procedure call replaces the old CS:IP pairs with new CS:IP
4	The value of old IP is Pushed on the stack SP=SP-2; save IP on Stack (address of procedure)	The value of old CS:IP pairs are pushed on the stack. SP=SP-2; save CS on stack SP=SP-2; Save IP (new offset address of called procedure)
5	Less stack locations are required	More stack locations are required
6	Example: call Delay	Example: call FARPTR Delay

Name :

Gurukul | Page No. :
Date : / /

- Write an ALP to count number of ones in given number.

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM DB 24 H
RESULT DB 1 DUP(0)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV BL, 00H
       MOV AL, NUM
       MOV CL, 10H
UP:    SAL AL, 01H
       JC GO
       JMP NEXT
GO:    INC BL
NEXT:  DEC CL
       JNZ UP
       MOV RESULT, BL
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START.
```

- Write an ALP to count number of ones in given number.

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM DB 24 H
RESULT DB 1 DUP(0)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV BL, 00H
       MOV AL, NUM
       MOV CL, 10H
UP:    SAL AL, 01H
       JC GO
       JMP NEXT
GO:    INC BL
NEXT:  DEC CL
       JNZ UP
       MOV RESULT, BL
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START.
```