# Name: Shrinivas Hatyalikar

# DIv: CS-B

# Roll No: 24

**Aim :-** Write 64 bit ALP to convert 4-digit Hex number into its equivalent BCD number and  Make your program user friendly to accept the choice from user for:

Display proper strings to prompt the user while accepting the input and displaying the result. (Use of 64-bit registers is expected)

**Apparatus :**

* Core 2 duo/i3/i5/i7 - 64bit processor
* OS – ubuntu 32bit/64bit OS
* Assembler used –nasm (the netwide assembler)
* Editor Used – gedit

**Theory :**

**STACK Operations:**

The stack follows the LIFO (Last In First Out) principle. The stack can be used, for example, to pass parameters to functions. T here are three instructions that can be used for interaction with the stack: Pop, Push and Exch.

**PUSH:-**

The PUSH instruction increments the stack pointer and stores the value of the specified byte operand at the internal RAM address indirectly referenced by the stack pointer. No flags are affected by this instruction.

**Operation**

**PUSH**

**SP = SP + 1**

**SP) = (direct)**

**POP:-**

The POP instruction reads a byte from the address indirectly referenced by the SP register. The value read is stored at the specified address and the stack pointer is decremented. No flags are affected by this instruction.

**Operation**

**POP**

**(direct) = (SP)**

**SP = SP - 1**

**Algorithm :- Hexadecimal to BCD conversion**

1. Declare

Section .data of proper string messages.

Section .bss of proper variables.

Declare various macros e.g. print, read & exit macros.

Section .text as starting point of code segment.

2. Prompt messages on screen & accept choice from user.

3. According to choice, accept 4 digit hex no. from user using accept\_16 procedure.

4. Load 16-bit number in AX.

5. Load RBX as 10 (BCD i.e. base 10 conversion)

6. Clear contents of RDX (for division it is used. So clear previous remainder)

7. Divide number by 10.

    i.e. RDX:RAX / RBX = RDX = Remainder, RAX = Quotient ( number / 10 )

8. Store the remainder on stack. So that it will be popped in reverse order as BCD no.

9. Increment digitcount.

10. Repeat steps 6 to 8 till quotient becomes zero. i.e. until RAX = 00

11. Print Hex to BCD Conversion message on screen.

12. Pop the content of stack in DX. (Previously stored remainders).

13. Convert remainder from digit to character for display purpose. The remainders are in the range 0-9. So add 30h to convert digit to character.

14. Display the digit on screen.

15. Decrement digitcount.

16. Repeat steps 12-15 until digitcount becomes zero.

17. Stop.

**Code:**

section .data

nline db 10,10

nline\_len equ $-nline

ano db 10," Assignment no :4",

db 10,"------------------------------------------------------------",

db 10," Assignment Name:Conversion From HEX to BCD.",

db 10,"----------------------------------------------------------",10

ano\_len equ $-ano

menu db 10,"Hex To BCD.",

menu\_len equ $-menu

hmsg db 10,"Enter 4 digit Hex Number::"

hmsg\_len equ $-hmsg

ebmsg db 10,"The Equivalent BCD Number is::"

ebmsg\_len equ $-ebmsg

emsg db 10,"INVALID NUMBER INPUT",10

emsg\_len equ $-emsg

;------------------------------------------------------------------------------

section .bss

buf resB 6

char\_ans resB 4

ans resW 1

;-----------------------------------------------------------------------------

%macro Print 2

MOV RAX,1

MOV RDI,1

MOV RSI,%1

MOV RDX,%2

syscall

%endmacro

%macro Read 2

MOV RAX,0

MOV RDI,0

MOV RSI,%1

MOV RDX,%2

syscall

%endmacro

%macro Exit 0

Print nline,nline\_len

MOV RAX,60

MOV RDI,0

syscall

%endmacro

;---------------------------------------------------------------

section .text

global \_start

\_start:

Print ano,ano\_len

MENU: Print menu,menu\_len

call HEX\_BCD

Exit

;-----------------------------------------------------------------

HEX\_BCD:

Print hmsg,hmsg\_len

call Accept\_16 ;accept 4 digit hex number

mov ax,bx ;mov hex number in ax

mov bx,10 ;for divide hex number by 10

xor bp,bp ;counter

back: xor dx,dx ;as dx each time contains remainder

div bx ;divide ax by 10 ax=Q,dx=R

push dx ;push dx on stack as it is bcd

inc bp ;inc counter by 1

cmp ax,0 ;compare whether Q is 0 if 0 means number get over

jne back ;mov to conversion of quotient

Print ebmsg,ebmsg\_len

back1: pop dx ;pop last digit pushed on stack

add dl,30h ;add 30 to digit to make them decimal

mov [char\_ans],dl ;print individual digit

Print char\_ans,1

dec bp

jnz back1 ;mov to next digit

RET

;-------------------------------------------------------------------

Accept\_16: ;Ascii(character) to hex number input

Read buf,5

MOV RCX,4

MOV RSI,buf

XOR BX,BX

next\_byte:

SHL BX,4

MOV AL,[RSI]

CMP AL,'0'

JB error

CMP AL,'9'

JBE sub30

CMP AL,'A'

JB error

CMP AL,'F'

JBE sub37

CMP AL,'a'

JB error

CMP AL,'f'

JBE sub57

error:

Print emsg,emsg\_len

Exit

sub57: SUB AL,20H

sub37: SUB AL,07H

sub30: SUB AL,30H

ADD BX,AX

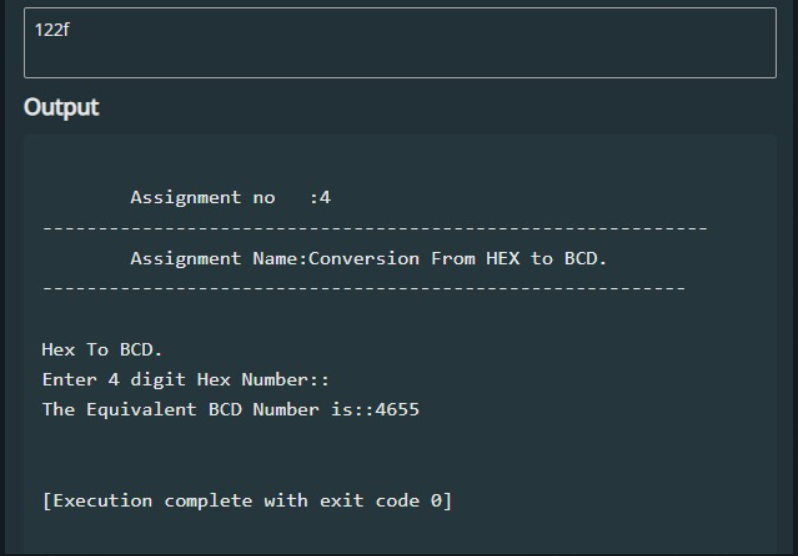
INC RSI

DEC RCX

JNZ next\_byte

RET

**Output:**

****

**Conclusion:**

The assembly program for converting a hexadecimal number to its decimal equivalent has been successfully implemented. The program takes a 4-digit hexadecimal number as input and uses a loop to iterate through each digit. It converts each digit to its decimal equivalent and adds it to a running total. The resulting decimal number is then outputted to the user. This program exemplifies the flexibility and efficiency of assembly language, enabling direct manipulation of data at a low level and optimal utilization of system resources. Furthermore, it highlights the significance of comprehending different number systems and their respective conversions. In summary, this program is a valuable tool for programmers and computer scientists who work with hexadecimal and decimal numbers. It can also be adapted to support other number systems such as binary or octal with some modifications.