Devanshu Surana
PC-23, Batch - C1
1032210755

MALoT Lab Assignment 8
Aim: Write X86/64 ALP to perform BCD to Hex and Hex to BCD conversion.

Theory:
Algorithm for BCD to HEX
1. Initialize product to 0
2. Accept the BCD no.
3. Extract first digit.
4. Convert it to Hex.
5. Multiply the product by 0AD.
6. Add the extracted digit to it.
7. Increment the pointer.
8. Repeat step 3 to 5 till you read "0a".
9. Display the product.

- Algorithm for Hex to BCD.
1. Accept the Hex no.
2. Pack and save it in ax.
3. Initialize dx = 0
4. Divide by 0Ah
5. Push remainder on stack.
6. Compare quotient with zero
7. If not go to step 3 else go to step 8.
8. Pop the digits from stack and display.

- Explain MUL and DIV instructions as well as Push and Pop instructions.

→ - MUL (Multiply) and DIV (Divide) instructions in assembly language are used to perform multiplication and division operations resp. on binary numbers. 1. →
- MUL instructions takes two operands and multiplication is done. Storing the result in register. The size of the two operands and results depends on the instructions used.
- DIV instruction takes two operands, a divident and a divisor and performs integer division, storing quotient in a specified register and the remainder in another register.
- Push and Pop instructions are used to manipulate the stack. The stack is a last-in-first-out (LIFO) data structure that is used to store temporary data in a program.
- Push instruction takes a value and pushes it into the top of the stack.
- Pop instruction pop a value of the top of the stack and stores it in a register.

2. →

- Input and Output :-
  (Hex to BCD)
  Input: 1A
  Output: 26

  (BCD to Hex)
  Input: 26
  Output: 001A

- Conclusion: The implementation of the conversion of BCD to Hex and Hex to BCD in a assembly language demonstrates the power and versatility of assembly language

programming, allowing for practice and efficient manipapulation of binary data

FAQ's

1 What are packed and unpacked numbers?

→ Packed numbers are stored in a compact format, taking up less memory space than unpacked numbers. This type of storage is commonly used for storing multiple smaller values in a single memory location, and is optimized for efficiency.

Unpacked numbers are stored in a format that takes up more memory space but provides more precision and accuracy. This type of storage is typically used for large or complex data structures where accuracy is more important than memory efficiency.

Eg: 98 is separated as 09 and 08. So we can say 10011000 [98] is packed and 00001001 [09] and 00001000 [08] are unpacked.

2. What is the necessity to convert from unpacked to packed?

→ Following are a few reasons:

1. Memory Efficiency: Packed numbers take up less memory space than unpacked numbers, so converting to packed numbers can help free up memory resources, which can be important in resource constraint system such as embedded systems.
2. Performance: By converting to packed numbers we can achieve improved processing speeds and reduced processing times, which can be important for performance critical applications.
3. Data Compression: - can help reduce the amount of data stored and improve storage efficiency.

Page No.
Date

3. What are assembler directives? Give example.

→ Assembler directives are directions to assembler to take some action or change a setting. They do not represent instructions and are not translated into machine code. Help automate assembly process and to improve program readability.

Examples are: ORG (origin), EQU (equate) and DS.B (define space for a byte).

Name: Devanshu Surana **Roll No.:** 23
Panel: C
Batch: C1

MAIoT Assignment 08

CODE:
HEX to BCD: %macro operat 4

mov rax,%1 mov rdi,%2 mov rsi,%3 mov rdx,%4 syscall

%endmacro

```
section .data
msg1 db "Enter HEX number:",10 msg1len equ $-msg1
msg2 db "BCD Equivalent number:",10 msglen equ $-msg2

section .bss a resb 1

num resb 5

section .code global _start

_start:
operat 1,1,msg1,msg1len operat 0,0,num,5
mov rsi,num
mov rbp,00
mov ax,00h
again:
mov bl,byte[rsi]
cmp bl,0Ah
jbe htop
cmp bl, 39h
jbe sub30h
sub bl,07h
sub30h:
sub bl, 30h
rol ax,4
add al,bl
inc rsi
jmp again
```

htop:
mov dx,00 mov bx,0ah div bx

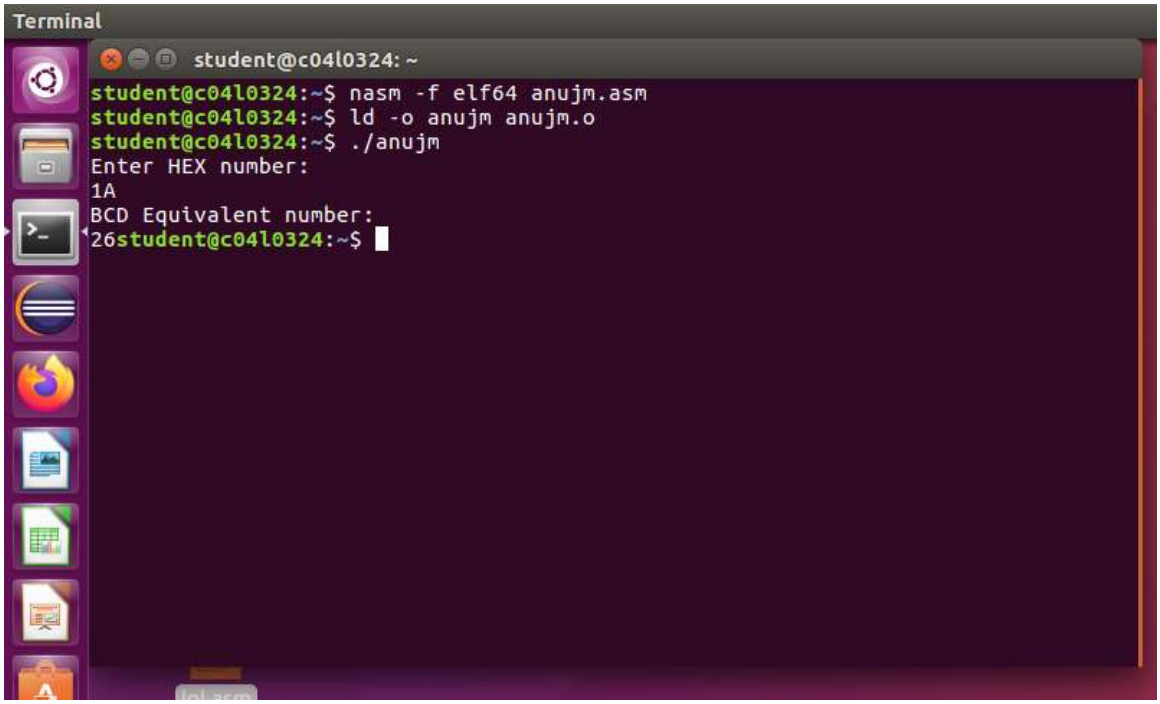
push dx
inc rbp
cmp eax,00
jnz htop
operat 1,1,msg2,msglen

prnt: pop dx

nxt1:
add dx,30h mov [a],dl operat 1,1,a,1 dec rbp
jnz prnt

exit:
mov rax,60 mov rdx,0 Syscall

OUTPUT:



BCD to HEX:

section .data
msg db "Enter the BCD number : ",10 msglen equ $-msg
msg2 db "The hex conversion is : " msg2len equ $-msg2

```
section .bss bcdnum resb 10 temp resb 1

%macro rw 4 mov rax,%1

mov rdi,%2 mov rsi,%3 mov rdx,%4 syscall %endmacro

section .text
global _start
_start:
rw 1,1,msg,msglen rw 0,0,bcdnum,10
rw 1,1,msg2,msg2len mov rsi,bcdnum

mov cx,0Ah mov bx,0 mov ax,0 again:

mov bl,[rsi] cmp bl,0Ah je done
sub bl,30h mul cx

add ax,bx inc rsi jmp again done:

mov bp,4

up: rol ax,4 mov bx,ax and ax,000Fh cmp al,09

jbe down1 add al,07h

down1: add al,30h mov byte[temp],al rw 1,1,temp,1 mov ax,bx

dec bp
jnz up
rw 60,0,0,0
```

OUTPUT: