ASSIGNMENT NO. 3

AIM :- Write 64 bit ALP to convert 4-digit Hex number into its equivalent BCD number and 5-digit BCD number into its equivalent HEX number. Make your program user friendly to accept the choice from user for:

1. HEX to BCD
2. BCD to HEX
3. EXIT

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.

ALGORITHM:- BCD to Hexadecimal 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 5 digit BCD no. from user using read macro. The number is accepted in character format.
4. Point at the start of number by RSI.
5. Clear contents of RAX. Final answer will be stored in RAX.
6. Load RBX as 10 (BCD number i.e. base 10, so multiply with 10)
7. Load counter CX as 5. (5 digit no.)
8. Clear contents of RDX (for multiplication it is used. So clear previous answer)
9. Multiply RAX by 10.
    i.e. RAX * RBX = RDX:RAX
    previous digit * 10 = ans
10. Clear contents of RDX (for multiplication it is used. So clear previous answer)

11. Load current digit in DL from RSI.
12. Convert the contents of DL from character to digit by subtracting 30h.
13. Add the current digit in final answer.
14. Increment source pointer.
15. Repeat steps 8 to 14 till CX becomes zero.
16. Store answer from AX to ans variable.
17. Print BCD to HEX ConversiIon message on screen.
18. Store answer from ans variable to AX again.
19. Call display_16 procedure to display final 16-bit hex no. on screen.
20. Stop

CONCLUSION:

_____

_____

_____