Assignment No: 3

Aim: Write X86/64 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:
(a) HEX to BCD b) BCD to HEX (c) EXIT.
Display proper strings to prompt the user while accepting the input and displaying the result. (Wherever necessary, use 64-bit registers)

Prerequisites:
Concept of number system

Learning Objectives:
Understand the implementation of stack for its conversion of number

Theory:
Hexadecimal number system: It is base 16 system. So there 16 numerals are required as 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
Binary has base 2 and hexadecimal has base 16(2^4). So 4 bits in binary represents one hexadecimal no.
E.g. Binary: 1011 1010 1000 0111 its equivalent hexadecimal is Hexadecimal B A 8 7

Binary coded decimal Number:
It consist of 0 to 9 decimal digits and are encoded as 4 bits in binary e.g. binary 12 is represented as 1100 but in BCD it is represented as 0001 0010
BCD no are categorised as packed BCD number and Unpacked BCD number
Packed BCD representation of 12 :0001 0010
Unpacked BCD representation of 12 :0000 0001 0000 0010

Conversion from decimal to BCD
Suppose decimal 523 numbers is converted into BCD then first converted into binary, make group of 4 bits and find its equivalent BCD no. 101 010 011 is converted into BCD as 0001 0101 0011 i.e. 153. If number is greater than 9 then we have to add 7 in it to get equivalent BCD.

Instruction Used:
1. PUSH:-Push word onto stack.
2. POP:-Pop word off stack.
3. DIV:-Divide byte/word
4. XOR: - Exclusive or byte/word
5. JA/JNBE:-Jump if above/not below or equal
6. JB/JNAE:-Jump if below /not above or equal
7. JG/JNLE:-Jump if /not less nor equal
8. JL/JNGE:-Jump if less /not greater nor equal
9. INT:-It allows ISR to be activated by programmer & external H\W device

Algorithm:
HEX to BCD.

1. Define variable in .data and .bss section
2. Display message on screen ENTER 4-DIGIT HEX NO.
3. Accept BCD NO from user.

4. Transfer 10(0AH) as a divisor in one of the register.
5. Divide the no by 0AH
6. PUSH reminder on stack
7. Increment Count _1.
8. Repeat Till quotient of BCD NO is not zero go to step 5.
9. Pop the content of Reminder in one of the register.
10. Display result by calling display procedure.
11. Decrement Count _1, till Count is not zero repeat step 9 else go to step 12.
12. Stop

BCD to HEX

1. Define variable in .data and .bss section
2. Display message on screen ENTER 5-DIGIT BCD NO.
3. Initialize result variable as zero and COUNT_1 as 5
4. Initialise index register to point at single digit .
5. Accept that number into one register
6. Multiply Result variable by 0AH & add accepted BCD digit into Result variable.
7. Decrement COUNT_1 and Increment pointer register to get the next digit
8. Repeat steps 5 to 7 till COUNT_1 becomes zero
9. Display result by calling display procedure
10. Stop.

Procedure for accept numbers: (ASCII to HEX)

1. Read a single character/digit from keyboard using function 0AH of INT 21H
2. Convert ASCII to HEX as per following:
    a. Compare its ASCII with 30H if No is less than 0 (i.e case of -ve no given) then go to step f else go to step c.
    b. Compare its ASCII with 39H if No is greater than 9 (i.e case of character A F given)then go to step f else go to step c .
    c. Store the resultant bit in NUM Variable.
    d. Check whether four digits (16-bit number) or two digits (8-bit number) are read; if yes then go to display procedure else go to step 1 for next bit
    e. Till counter is zero go to accept procedure.
    f. Display massage I/P is invalid BCD number & go to step 10.
3. End of accept procedure.

Procedure for display Result: (HEX to ASCII)

1. Compare 4 bits (one digit) of number with 9
2. If it is <= 9 then go to step 4 else go to step 3
3. Add 07 to that number
4. Add 30 to it
5. Display character on screen using function 02 of INT 21H
6. Return to main routine
7. End of display procedure.

Conclusion: Thus, We have successfully learned about number system and conversions between 2 number system.

Assignment No:4

Aim:Write X86/64 ALP to perform multiplication of two 8-bit hexadecimal numbers. Use successive addition and add and shift method. (use of 64-bit registers is expected)

Prerequisites:

Concept of multiplication method

Learning objectives:

Understanding shift & add method and successive addition method.

New Concept

Implementation of Shift and rotate instructions.

Theory

Multiplying unsigned numbers

Multiplying unsigned numbers in binary is quite easy. Recall that with 4 bit numbers we can represent numbers from 0 to 15. Multiplication can be performed done exactly as with decimal numbers, except binary system have only two digits (0 and 1). 0*1=0, and 1*1=1 (this is the same as a logical "and").

Successive Addition Method:

In successive addition, we have to add first number by second number of time to obtain the multiplication.

i.e. Suppose we have to find m*n then m*n=(m+m+m é ..n times) and if n=m=4 then the result is 8 bits.

Decimal	Binary
10 x6 60	1010 x0110 0000 1010 1010 +0000 0111100

In this case the result was 7 bit, which can be extended to 8 bits by appending a 0 at the left. When multiplying larger numbers, the result will be 8 bits, with the leftmost set to 1,

Multiplying signed numbers

There are many methods to multiply 2's complement numbers. The easiest is to simply find the magnitude of the two multiplicands, multiply these together, and then use the original sign bits to determine the sign of the result. If the multiplicands had the same sign, the result must be positive, if they had different signs, the result is negative. Multiplication by zero is a special case (the result is always zero, with no sign bit).

Multiplication and division can be performed on signed or unsigned numbers. For unsigned numbers, MUL and DIV instructions are used, while for signed numbers IMUL and IDIV are used.

Sequential Shift/Add-Method

In this method multiplier is shifted to right if carry is set or LSB is 1 then add multiplicand in result and the shift multiplicand by one bit

If carry is not set then no addition is done only shifting (multiplicand) operation is done.

Algorithm:

1. Declare macro ñdispmsgò with 2 arguments.
2. Move 01H into rax , system call number for display.
3. Move 01H into rdi , handle for screen.
4. Move first argument to macro into rsi , variable to be displayed.
5. Move second argument into rdx , size of variable.
6. Declare macro ñacceptò with 2 arguments.
7. Move 00H into rax , system call number for reading from keyboard.
8. Move 00H into rdi , handle to screen.
9. Move first argument into rsi , variable to be displayed.
10. Make syscall.
11. End macro.
12. Declare data segment.
13. Declare variable ñmsgò as string.
14. Declare variable ñmsg_lenò and store the length of ñmsgò variable in it.
15. Declare varaible ñresò as a string.
16. Declare variable ñres_lenò and store ñresò length in it.
17. Declare variable ñchoiceò as string storing menu related messages.
18. Declare variable ñchoice_lenò and store size of ñchoiceò variable in it.
19. Declare extra segment.
20. Declare ñnumò variable of 3 bytes size.
21. Declare ñnum1ò variable of 1 byte size.
22. Declare ñresultò variable of 4 byte size.
23. Declare ñchoò variable of 2 byte size.
24. Declare code segment.
25. Define entry point label ñ_startò.
26. XOR register rax with itself to make it zero.
27. XOR register rbx with itself to make it zero.
28. XOR register rcx with itself to make it zero.
29. XOR register rdx with itself to make it zero.
30. Move 00H into ñresultò variable.
31. Move 00H into ñnumò variable.
32. Move 00H into ñnum1ò variable.
33. Call ñdispmsgò macro and display ñchoiceò variable.
34. Call macro ñacceptò and read ñchoò from keyboard.
35. Compare ñchoò with 31H.
36. Jump if equal to label ñbò.
37. Jump to label ñexitò.
38. Mark label ñbò.
39. Call procedure ñSucce_additionò.
40. Jump to entry point label ñ_startò.
41. Mark label ñexitò.
42. Move 60H into rax.
43. Move 00H into rdi.
44. Make system call ñsyscallò.

Convert procedure

1. XOR rbx with rbx to initialize it to 0.
2. XOR rcx with rcx to initialize it to 0.
3. XOR rax with rax to initialize it to 0.
4. Move 02H to rcx register.
5. Move the address of ñnumò variable into rsi.
6. Mark label ñup1ò.
7. Rotate left bl by 4 bits.
8. Move value at memory location pointed by rsi into al.
9. Compare al with 39H.
10. Jump if greater to ñp1ò.
11. Subtract 30H from al.
12. Jump to ñp2ò.
13. Mark label ñp1ò
14. Subtract 37H from al.
15. Mark label ñp2ò.
16. Add al into bl.
17. Increment rsi.
18. Loop the statements from label ñup1ò till ecx becomes zero.
19. Return from procedure.

Display procedure

1. Move 04H into rcx.
2. Move address of ñresultò variable into rdi.
3. Mark label ñdup1ò.
4. Rotate left bx register by 4 bits.
5. Move bl into al.
6. And al with 0FH.
7. Compare al with 09H.
8. Compare al with 09H.
9. Jump if greater to label ñp3ò.
10. Add 30H into al.
11. Jump to label ñp4ò.
12. Mark label ñp3ò.
13. Add 37H into al.
14.Mark label ñp4ò.
15. Move al into memory location pointed by rdi.
16. Increment rdi.
17. Loop the statements starting from label ñdup1ò till ecx becomes zero.
18. Return from procedure.

Succe_addition procedure

1. Call ñdispmsgò macro and display string msg.
2. Call ñacceptò macro and read ñnumò variable from keyboard.
3. Call ñconvertò procedure.
4. Move bl into memory location for ñnum1ò variable.

5. Call ñdispmsgò macro and display ñmsgò string.
6. Call ñacceptò macro and read ñnumò variable from keyboard.
7. Call ñconvertò procedure.
8. XOR rcx with rcx to make it zero.
9. XOR rax with rax to make it zero.
10. Move num1 into rax.
11. Mark label ñrepetò.
12. Add rax into rcx.
13. Decrement bl register.
14. Jump if not zero to label ñrepetò.
15. Move rcx into ñresultò variable.
16. Call ñdispmsgò macro and display ñresò message.
17. Move ñresultò variable into rbx register.
18. Call display procedure.
19. Return from procedure.

Add_shift procedure

1. Call macro ñdispmsgò and display message ñmsgò.
2. Call ñacceptò macro and read ñnumò value from keyboard.
3. Call ñconvertò procedure.
4. Move bl into variable ñnum1ò.
5. Call macro ñdispmsgò and display ñmsgò variable.
6. Call ñacceptò macro and read ñnumò value from keyboard.
7. Call procedure convert.
8. Move bl into ñnumò variable.
9. XOR rbx with rbx to make it zero.
10. XOR rcx with rcx to make it zero.
11. XOR rdx with rdx to make it zero.
12. XOR rax with rax to make it zero.
13. Move 08H into dl.
14. Move value of variable ñnum1ò into al.
15. Move value of ñnumò into bl.
16. Mark label ñp11ò.
17. Shift right the value of register bx by 01 bit.
18. Jump if no carry to label ñpò.
19. Add value of ax into cx.
20. Mark label ñpò.
21. Shift left value in register ax by 01 bit.
22. Decrement dl.
23. Jump if not zero to label ñp11ò.
24. Move value in register rcx into variable result.
25. Call macro ñdispmsgò and display ñresò variable.
26. Move ñresultò variable into rbx.
27. Call procedure ñdisplayò
28. Return from procedure

Conclusion: Thus, we have successfully studied multiplication of two nos.