MPL Assignment – 4

Name: Dheeraj Gonchigar

Roll No.: 21129

Problem definition:

Write a switch case driven X86/64 ALP to perform 64 bit hexadecimal arithmetic operations (+, -, *, /) using suitable macros. Define procedure for each operation

Learning objective: Students will be able to:

- Get familiar with the arithmetic operations and implement them
- Get familiar with the ASCII concepts
- Understand the concept on macros and procedure and use the to implement following program
- Implement the function for Hexadecimal to Ascii conversion
- Write the program
- Run it in the terminal

Learning Outcome: Students will be able to:

- Get familiar with the arithmetic operations and implement them
- Get familiar with the ASCII concepts
- Understand the concept on macros and procedure and use the to implement following program
- Implement the function for Hexadecimal to Ascii conversion
- Write the program
- Run it in the terminal

Concepts related Theory:

Instructions used:

1. **ADD/SUB:** The ADD and SUB instructions are used for performing simple addition/subtraction of binary data in byte, word and doubleword size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.
   Syntax: ADD/SUB destination, source

   The ADD/SUB instruction can take place between –

   - Register to register
   - Memory to register
   - Register to memory
   - Register to constant data
   - Memory to constant data

However, like other instructions, memory-to-memory operations are not possible using ADD/SUB instructions. An ADD or SUB operation sets or clears the overflow and carry flags.

2. **MUL/IMUL:** There are two instructions for multiplying binary data. The MUL (Multiply) instruction handles unsigned data and the IMUL (Integer Multiply) handles signed data. Both instructions affect the Carry and Overflow flag.
   Syntax: MUL/IMUL multiplier
   Multiplicand in both cases will be in an accumulator, depending upon the size of the multiplicand and the multiplier and the generated product is also stored in two registers depending upon the size of the operands.

3. **DIV/IDIV:** The division operation generates two elements - a quotient and a remainder. In case of multiplication, overflow does not occur because double-length registers are used to keep the product. However, in case of division, overflow may occur. The processor generates an interrupt if overflow occurs.

   The DIV (Divide) instruction is used for unsigned data and the IDIV (Integer Divide) is used for signed data.

   Syntax: DIV/IDIV divisor

4. **Jbe:** Jumps to the destination label mentioned in the instruction if the result of previous instruction (generally compare) causes either the CF or ZF to have value equal to 1, else no action is taken.
   Syntax: jbe label
   Eg. jbe l2

5. **Jnz:** Jumps to the destination label mentioned in the instruction if the ZF is 0, else no action is taken.
   Syntax: jnz label
   Eg. Jnz l1

Algorithm:

1. Start
2. Display the menu for user and take choice from user
   I. Addition
   II. Subtraction
   III. Multiplication
   IV. Division
   V. Exit
3. Define the macros for reading input and displaying the output
4. If choice=1:
   Call Addition procedure

Else if choice=2:
    Call Subtraction procedure
Else if choice=3:
    Call Multiplication procedure
Else if choice=4:
    Call Division procedure
Else:
    Call exit procedure

Algorithm for addition procedure:

1. Move the numbers in the registers rax and rbx using mov command
2. Add the following using add rax,rbx command
3. Call the hex to ascii procedure for conversion from hexadecimal to ascii
4. Display the result and return from the procedure

Algorithm for Subtraction procedure:

1. Move the numbers in the registers rax and rbx using mov command
2. Subtract the following using sub rax,rbx command
3. Call the hex to ascii procedure for conversion from hexadecimal to ascii
4. Display the result and return from the procedure

Algorithm for multiply procedure:

1. Move the numbers in the registers rax and rbx using mov command
2. Add the following using add rax,rbx command
3. Do multiplication using mul instruction
4. Mov the rax part of result to r9 register
5. Empty the rax register using xor instruction and move the rdx part of the result to rax register
6. Call hex to ascii procedure for conversion from hexadecimal to ascii
7. Clear the rax register using xor instruction
8. Mov the result stored in r9 to rax register
9. Display the result and return from the procedure

Algorithm for division procedure:

1. Move the numbers in the registers rax and rbx using mov command
2. Add the following using add rax,rbx command
3. Call the hex to ascii procedure for conversion from hexadecimal to ascii
4. Display the result and return from the procedure

Algorithm for addition procedure:

1. Move the numbers in the registers rax and ebx using mov command respectively
2. Clear the rdx register using xor instruction
3. Perform division using the div instruction

4. Move the rdx part of result to r9 register
5. Call the hex to ascii procedure for conversion from hexadecimal to ascii
6. Clear the rax register using xor instruction
7. Move content of r9 to rax register
8. Call the hex to ascii procedure for conversion from hexadecimal to ascii
9. Display the result and return from the procedure

Test cases:

Sr. No.	Input	Output	Pass/Fail
1	0000000000000003 0000000000000005	Addition: 0000000000000008 Subtraction: 0000000000000002 Product: 00000000000000000000000000000000F Division: Quotient: 0000000000000001 Remainder: 0000000000000002	Pass
2	0000000000000007 0000000000000002	Addition: 0000000000000009 Subtraction: 0000000000000005 Multiplication: 0000000000000000000000000000000E Division: Quotient: 0000000000000003 Remainder: 0000000000000001	Pass

Conclusion/Analysis:

We have successfully taken a string as input from the user and displayed its length in the hexadecimal form.