CSC 3410 Assignment 2

Getting Started with Simple Math and Strings

Skills you will learn

- Using various Unix tools to build your programs
- Using instructions to do simple math and work with strings
- Using instructions to make Linux system calls
- Using directive to define data

Things you will need

- A text editor (provide by the VM - VS code, Gedit, VIM, ...)
- The Nasm assembler (provided by the VM)
- The ld linker (provided by the VM)

Description

For this assignment, you are going to write your first assembler programs by writing some simple code to use the ADD, MOV, SUB, IMUL, and IDIV instructions. Each program below is to be written and tested on the virtual machine given in class and must compile using the nasm assembler and must link using the ld linker. For example, to compile program 1, you will enter the following commands:

```
nasm -g -f elf -F dwarf -o p1.o p1.asm
ld p1.o -m elf_i386 -o p1
```

Do not use any macros or libraries for this assignment.

The programs are described below:

1. Write an assembler language program called p1.asm that prompts the user for two single digit numbers. Once the user has entered the two numbers, the program will add them together, via the `add` instruction, and print the results. Note that when the numbers are entered, they will be in string form and you will have to convert them into numbers. To convert a single digit number represented as a string into an actual number, you must subtract 48 from it (which is the ASCII code for the character '0'). For example, '5' - '0' is 5, and '1' - '0' is 1. You will also need to convert the answer back to a character representation so that it prints correctly. To turn a single digit number into a character representation, add a 48 to it. For example, 5 + '0' is '5', and 1 + '0' is '1'. The algorithm for this first program is below:

    1. prompt the user for a single digit number
    2. read the number from the keyboard

3. prompt the user for the second single digit number
4. read the second number from the keyboard
5. convert the first number represented as a string to a number by subtracting 48 from it (if the first number is in al, then all you have to do is: `sub al, '0'`)
6. convert the second number represented as a string to a number by subtracting 48 from it
7. add the first number to the second number and save the answer
8. convert the answer from a number to a string by adding 48 to it
9. print "The answer is: "
10. print the saved answer

You can assume the user will only enter a 1-digit number, so you do not have to do any input checking for this assignment. Also, you may assume the addition of the two numbers will result in a single digit number (of course, this assumption is not valid outside of this assignment, but we have not covered enough to handle this situation, yet). A screenshot for the program is as follows:

```
The Adding Program
Please enter a single digit number: 2
Please enter a single digit number: 1
The answer is: 3
```

2. Write a program called p2.asm that is similar to program 1 but, instead of adding the numbers, the program multiplies the number using IMUL. Use the form of IMUL that takes one argument. You should first zero out AX. Then, you will load the first number (after converting it to a number, of course) into al, and call IMUL with an operand that references the memory containing the second number, for example:
   `imul [second_number]`
   The answer will be stored in AX, but you only have to worry about printing what is in AL (you can assume that the results are a single digit number for printing purposes, just like in problem 1).

   Example output:

```
The Multiplying Program
Please enter a single digit number: 2
Please enter a single digit number: 3
The answer is: 6
```

3. Write a program called p3.asm that is similar to programs 1 and 2, except that this program uses IDIV to divide the first number by the second number. IDIV is similar to IMUL. However, the quotient will be stored in AL, and the remainder will be stored in AH. Your program should print "The quotient is: " followed by the quotient on a single line, and "The remainder is: " followed by the remainder on a single line.

   Example output:

```
The Dividing Program
Please enter a single digit number: 6
Please enter a single digit number: 2
The quotient is: 3
The remainder is: 0
```

4. Write a program called p4.asm that prompts the user for a two-character string. Once the string is entered, swap the characters in the string and then print the result. You must swap the characters in the string (in other words, swap the bytes that are stored in memory) and then print the modified string to get credit for this program. Allocate the memory for the string in a single directive in the. bss section, such as:

```
two_char_string: resb 3         ; added an extra character
to put a newline into when printing
```

   Example output:

```
The Swapping Program
Please enter a two character string: We
The answer is: eW
```

Turn In

Create a directory called <your_email>_hw1 and put all of the programs in it, where <your_email> is your TnTech email id (**do not** including the @tntech.edu). Make sure you name the programs p1.asm, p2.asm, and so forth. You will send your entire project's directory. You can create your tarball with the following command:

```
tar -czf <your_email>_hw1.tar.gz <your_email>_hw1/
```

from the top-level directory (one up from your project directory). Submit the tarball via iLearn.

Rubric

Each error found in a programming assignment results in a deduction of points from a starting grade. The starting grade for your assignment is determined by stepping down from 100 points in increments of 5, starting at 100. The conditions for a steps down are outlined below.

Condition	Steps down
Major feature missing or incomplete	6
Minor feature missing or incomplete	3
One or more major bugs	2
One or more minor bugs	1
Inefficiency, poor structure, poor error checking ...	0
Nothing significant submitted	20

If nothing significant is submitted for the assignment, then the resulting grade is 0 (20 steps down is a base grade of 0). The grader determines what is "major" and "minor". For example, a major feature is typically a focus of the assignment, whereas a minor features might be some error checking that is explicitly mentioned as a requirement in the assignment. Typically, a major bug causes the program to crash or a feature to fail in some cases, and needs extra code to correct, whereas a minor bug causes the program to misbehave in some cases, but is easily correctable (such as correcting an off-by-one error by reducing the range variable by one).

Below are some example of applying the grading scheme to a program:

Example 1: The student submits a program with 1 major feature missing, 1 major bug, and two minor bugs.
Steps down = 6 + 2 + 1 = 9 , so 100 - (5*9) is 100-45 = 55

Example 2: The student submits a program with 1 minor feature missing and 2 minor bugs
Steps down = 3 + 1 = 4, so 100 - 5*4 is 100 - 20 = 80