

UNIVERSITY OF THE PHILIPPINES VISAYAS
COLLEGE OF ARTS AND SCIENCES
DIVISION OF PHYSICAL SCIENCES AND MATHEMATICS

CMSC 131

Introduction to Computer Organization and Machine Level Computing
A.Y. 2022 - 2023

Assignment Guide

Prepared by:

Jayvee B. Castañeda
Instructor

ACADEMIC INTEGRITY

As a student of the University of the Philippines, I pledge to act ethically and uphold the value of honor and excellence. I understand that suspected misconduct on given assignments/examinations will be reported to the appropriate office and if established, will result in disciplinary action in accordance with University rules, policies and procedures. I may work with others only to the extent allowed by the Instructor.

Laboratory Exercise #3

Reading

- Read [Section 2.3 of Paul Carter's PC Assembly Book](#)

Practice Exercise:

- Assemble the assembly code (**prime.asm**). This will create an object file (**prime.o**) for math.asm.

nasm -f elf prime.asm

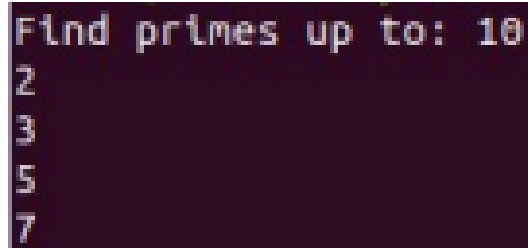
- Compile and link the assembly code with the C program (**driver.c**). In our machine, we will be using 32-bit registers thus we specify “-m32”.

```
gcc -m32 -o prime driver.c prime.o asm_io.o
```

- Execute the assembly code.

```
./prime
```

The code should show the following:

A terminal window with a dark background and light-colored text. The text reads: "Find primes up to: 10" followed by a list of prime numbers: 2, 3, 5, and 7, each on a new line.

```
Find primes up to: 10
2
3
5
7
```

- Analyze the assembly code (first.asm). Reflective questions:

What does the program do?

How do the different control structures such as “cmp” and “jmp” instructions differ from each other?

Problem #3.

In pursuing his new hobby of painting, Heinz Ketchup Doofenshmirtz finds an odd set of paints. He wants to create an invention that can sort out the paints by color, size, and code, if possible. He calls this the “Painter-ist-or-ism-inator”. However, the invention seemed to malfunction and is only spitting out two integers on its monitor. Then, a painting with the words “*I’ll see ‘em!’*” was produced. Heinz Ketchup figured out that he needs to find the “*I’ll see ‘em!’*” of these numbers to figure out the code to fix the “Painter-ist-or-ism-inator”.

- Write an assembly program that finds the least common multiple (LCM) of two integers.

Note:

The least common multiple of two integers a and b , also known as LCM of a and b , is the least positive integer that is divisible by both the two integers.

- Design a high level programming logic for the problem above and translate it to assembly language. Use control structures (*cmp*, *jmp*, etc.) to solve the problem.

- The output of your program should be something like this:

Output #1:

```
Enter two integers: 2 10
The least common multiple of 2 and 10 is 10.
```

Output #2:

```
Enter two integers: 3 5
The least common multiple of 3 and 5 is 15.
```

Output #3:

```
Enter two integers: 17 4
The least common multiple of 17 and 4 is 68.
```

- Please use the TEST CASES below to verify if your code produces the correct output.

Integers	LCM
0 1	None
1 2	2
2 10	10
3 5	15
17 4	68
5 13	65
6 8	24
15 25	75
20 30	60
100 30	300
18 27	54
24 36	72
11 20	220
12 12	12
75 50	150
250 5	250
70 90	630
56 32	224

- A good programming practice is to *write comments on important line of codes* for readability and documentation.
- Save your program in a file called *SurnameFirstLetterOfFirstName_lab3.asm* in camel case. For instance, if your surname is “Juan Dela Cruz”, submit it as follows:

DelaCruzJ_lab3.asm

- Take a screen recording of your working code and make sure to **record a video explaining each line of your code** as well as showing the correct output of your code. Use screen recorder application in Ubuntu (<https://itsfoss.com/best-linux-screen-recorders/>) or Windows (<https://atomisystems.com/screencasting/record-screen-windows-10/>)

Submission Requirements:

1. Program Code (‘.asm’ file)
2. Screen Recorded Defense Video

DEADLINE: October 20, 2022, 11:59 PM

Rubric for Programming Exercises				
Program (50 pts)	Excellent	Good	Fair	Poor
<i>Program Execution</i>	Program executes correctly with no syntax or runtime errors (9-10)	Program executes with minor (easily fixed) error (4-8)	Program executes with a major (not easily fixed) error (2-3)	Program does not execute (0-1)
<i>Correct Output</i>	Program displays correct output with no errors (9- 10)	Output has minor errors (6-8)	Output has multiple errors (3-5)	Output is incorrect (0- 2)
<i>Design of Output</i>	Program displays more than expected (7-8)	Program displays minimally expected output (5-6)	Program does not display the required output (3-4)	Output is poorly designed (0-2)
<i>Design of Logic</i>	Program is logically well-designed (9-10)	Program has slight logic errors that do not significantly affect the results (6-8)	Program has significant logic errors (3-5)	Program is incorrect (0-2)
<i>Standards</i>	Program is stylistically well designed (6-7)	Few inappropriate design choices (i.e., poor variable names, improper indentation) (4-5)	Several inappropriate design choices (i.e., poor variable names, improper indentation) (2-3)	Program is poorly written (0-1)
<i>Documentation</i>	Program is well documented (5)	Missing one required comment (4)	Missing two or more required comments (2- 3)	Most or all documentation missing (0-1)