

Using x86 Assembly-Disassembly

Computer Architecture Exploitation and Security

Table of Contents

[Background Reading 1](#_Toc46175487)

[Important Information 1](#_Toc46175488)

[Introduction 2](#_Toc46175489)

[Problem 1 2](#_Toc46175490)

[Problem 2 4](#_Toc46175491)

[Problem 3 6](#_Toc46175492)

problem 4………………………………………………………………………………………………...10

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| C and Assembly code conversion with scan() and loops | 15 |  |
| Convert assembly code into C | 10 |  |
| Convert assembly code into C | 10 |  |
| Convert assembly code into C | 10 |  |
| TOTAL MARK | 45 |  |

Computer Architecture Exploitation and Security

Using x86 Programming Tools

This lab focuses on the following objectives:

* Decompile the assembly code.
* Find errors if present and fix them

Background Reading

GNU C Compiler (Instructor assigned reading)

GNU C Assembler, Linker (Instructor assigned reading)

GNU GDB (Instructor assigned reading)

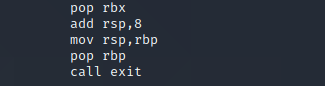
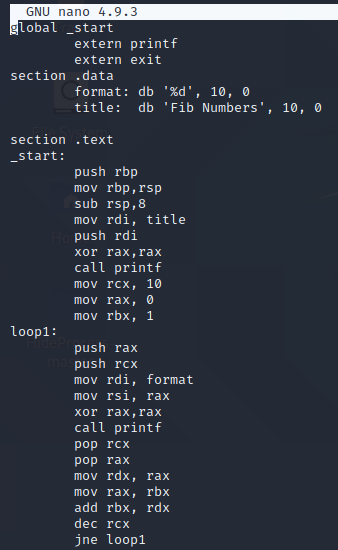
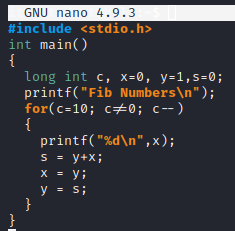
# Important Information

# Introduction

In this lab, you will write code for the x86 processor architecture using virtual machines running the Linux operating system. Ideally, you will be running Linux x86\_32 system but your x86\_64 will work with the appropriate libraries installed. You should already have the GNU C compiler, assembler, linker and the GNU Make utility installed. Additionally, you will use the GNU debugger, GDB, to execute and single-step through simple programs. You will observe how the code functions and then reverse the assembly code to create the C equivalent. Find any errors in the code and resolve it.

**HINT**: Think about how you can test the code after you have completed it to ensure that your theories are correct.

# Problem 1 C and Assembly code \_\_\_15



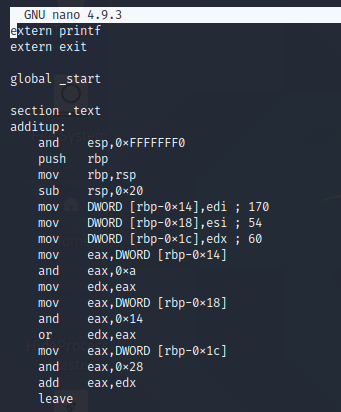
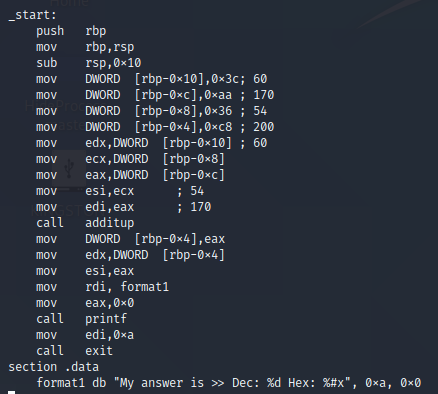
1. Run and analyze the following C code and the respective assembly code.
   1. Generate the assembly code
   2. Generate the executable
   3. Run the program to see the output
   4. Use GDB to Analyze the program
2. (5 marks) In the assembly code comment the lines identifying the following:
   1. Prolog and Epilog
   2. The registers that store the **counter** and variables **x, y**
   3. The registers that store the parameters passed to the printf() function **within the loop**
   4. The instruction that performs **x = y**
   5. The instructions that control the loop?
   6. Attach the screenshots along with the commented assembly code
3. (10 marks) Modify the C code and respective assembly code to perform the following:
   1. Ask user for number to calculate Fibonacci Series
   2. Use scanf() function to read input from STDIN.
   3. Use section .bss with a variable that will store the input read from STDIN
   4. Replace the for loop with a while loop
   5. Submit a screenshot of the modified C code and the respective assembly code as well as the results after execution.

# Problem 2 Convert Assembly into C cod­e \_\_\_\_10

1. You are provided with the assembly code on the page below. You will need to take the code and convert it to it’s C language equivalent.
2. There are errors in the code. It could be extra instructions or missing instructions. Use the debugger for troubleshooting. Provide a report of the errors you found.
3. Types of errors include problems with how the stack pointer or stack frame is being manipulated. Possibly the values being placed on the stack are in the wrong order.
4. The program should print (***It is more important for you understand the flow of the program and less so on getting the exact output***): My answer is >> Dec: 70 Hex: 0x46
5. Submit a screenshot of the C code as well as the output of the program after execution.
6. **DO NOT** hard code the output string.

**HINT**: It may be helpful to draw a stack diagram to get a layout of the values and confirm that the values are as expected.

**Note**: There are tools available to convert images with text in it, to get a decent text document representation that saves some typing. If you use such a program **confirm** that the output is correct.



# Problem 3 Convert Assembly code to C \_\_\_\_10

1. You are provided with the assembly code on the following page. You will need to take the code and convert it to it’s C language equivalent.
2. There are a few problems with the code that you will need to debug to resolve. You will report on what those errors are (eg **extra or unnecessary code added or missing code**) and which function the errors were found in.
   1. The program should print the following output:

Entered Main

Entering function1

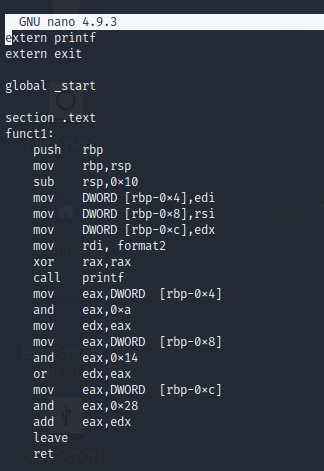
First value: **yyyyy** + second val: **xxxxx** = **zzzzz**

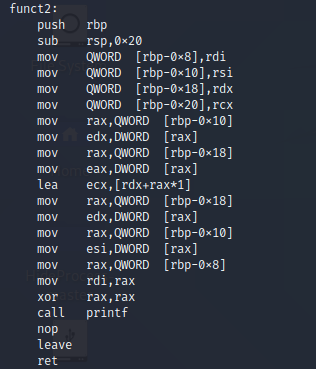
Exiting program

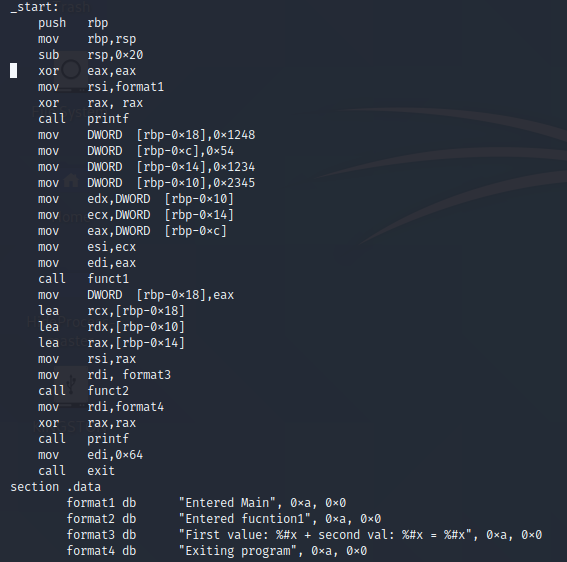
The values xxxxx, yyyyy and zzzzz are not the actual values. They are placeholders.

1. You will submit the following:
   1. An explanation of what you did to make the program work.
   2. A screenshot of the C code as well as the output of the program after execution.

**Note**: There are tools available to convert images with text in it, to get a decent text document representation that saves some typing. If you use such a program **confirm** that the output is correct.







# Problem 4 Convert Assembly code to C \_\_\_/10

1. You are provided with the assembly code below. You will need to take the code and convert it to its C language equivalent.
2. Take screenshots and **submit your fixed code** along with the C code equivalent.

