

Using x86 Programming Tools

Computer Architecture Exploitation and Security

Table of Contents

[Background Reading 2](#_Toc43806404)

[Important Information 2](#_Toc43806405)

[Introduction 2](#_Toc43806406)

[Problem 1 2](#_Toc43806407)

[Problem 2 3](#_Toc43806408)

[Problem 3 4](#_Toc43806409)

[Problem 4 5](#_Toc43806410)

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**:

|  |  |  |
| --- | --- | --- |
| Single-Precision Floating Point | 5 |  |
| Double-Precision Floating Point | 5 |  |
| Compiler and Assembler | 18 |  |
| **BONUS -** Linear Address Translation | 10 |  |
| TOTAL MARK | 28 + 10 |  |

Computer Architecture Exploitation and Security

Using x86 Programming Tools

This lab focuses on the following objectives:

* Linear address Translation
* Describe programming tools.
* Use programming tools.
* Explain the function of assemblers, linkers, dissemblers and debuggers.
* Write, compile and execute simple assembly code.
* Use a debugger to single-step through simple assembly code.

Background Reading

GNU C Compiler (Instructor assigned reading)

GNU C Assembler, Linker (Instructor assigned reading)

GNU GDB (Instructor assigned reading)

# Introduction

When the processor is executing tasks, it will from time to time require data which is stored in memory (RAM). The address of that data is presented to the processor. The processor must request that data, the Memory Management Unit (MMU) handles those requests. The MMU is the hardware that converts virtual address (linear address) to physical address. MMU does not know the exact location of the data but can translate the address given to it to determine where the data is stored in RAM (physical address).

# Problem 1 Single-Precision Floating Point \_\_\_/5

Express the following numbers in single-precision floating point format, **use hexadecimal notation throughout**:

**(I put both binary and hexadecimal notation)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Decimal** | **Sign** | **Exponent** | **Fraction** | **Complete 32-bit number** |
| -0.275 | 1  0x1 | 01111101  0x7D | 00011001100110011001101  0xCCCCD | 10111110100011001100110011001101  0xBE8CCCCD |
| 2.25 | 0  0x0 | 10000000  0x80 | 00100000000000000000000  0x100000 | 01000000000100000000000000000000  0x40100000 |
| -0.0275 | 1  0x1 | 01111001  0x79 | 11000010100011110101110  0x6147AE | 10111100111000010100011110101110  0xBCE147AE |

# Problem 2 Double-Precision Floating Point \_\_\_/5

Express the following numbers in double-precision floating point format. Use hexadecimal notation throughout.

Note: The decimal numbers are the same as in the previous problem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Decimal** | **Sign** | **Exponent** | **Fraction** | **Complete 64-bit number** |
| -120.54 | 1  0x1 | 10000000101  0x405 | 1110001000101000111101011100001010001111010111000011  0xE228F5C28F5C3 | 1100000001011110001000101000111101011100001010001111010111000011  0xC05E228F5C28F5C3 |
| 2.825 | 0  0x0 | 10000000000  0x400 | 0110100110011001100110011001100110011001100110011010  0x699999999999A | 0100000000000110100110011001100110011001100110011001100110011010  0x400699999999999A |
| -0.575 | 1  0x1 | 01111111110  0x3FE | 0010011001100110011001100110011001100110011001100110  0x2666666666666 | 1011111111100010011001100110011001100110011001100110011001100110  0xBFE2666666666666 |

# Problem 3 – Compiler \_\_\_/18

1. Write a “**C program**” that prints a small message on the terminal call it **message.c**

***The message could be anything tasteful like a riddle or a song lyric.***

1. **(3 marks)** Use gcc manual or any other resource and research the following gcc options. Provide a brief description or purpose of the option
   1. S

**It will stop before the assembly stage, the output is in the form of an assembler code file.**

* 1. O

**Optimizes the code, the compiler tries to reduce code size and execution time.**

* 1. c

**Compile or assemble the source files, but do not link.**

* 1. v

**Print the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper**

* 1. Wa

**Pass an argument as an option to the assembler**

* 1. adhln

**a: turn on listings, d: omit debugging directives, n: omit forms processing, h: include high-level source, l: include assembly**

1. Use the **C program** you created in **step 1** and the appropriate **gcc** option(s) to generate the **Assembly Code**. The newly created assembly code should be called **mesg.s**
2. Use any editor to open the mesg.s file. **Observe** the content of the file paying attention to the different sections created. Identify the section(s) that contains the string message your created in Step 1.
3. **(1 mark)** Redo step 3 and generate an optimized version of **mesg.s**, by adding the **-O2** **(capital O)** option during compilation. Open the newly created mesg.s file and compare with previous result without optimization. Describe any difference(s) you noticed? **HINT**: Look for any difference(s) related to the Observation made in Step 4.

**There is a newly added line under the string:**

**.section .text.startup,"ax",@progbits**

1. **(1 mark)** Use Linux man pages and get a general sense of how the **as** program works. What is the purpose of **as**?

**Its assembles the output of GCC for use by the linker ld**

1. **(3 marks)** Use **as** to convert the assembly code created in Step 5 into an object file and call it **mesg.o**
2. Why shouldn’t the .o file be read with a text editor?

**Because it contains binary data not ASCII and the text editor can’t recognize it**

1. Compare the size of the .o file and the .s file. What is the difference between the sizes, is that significant?

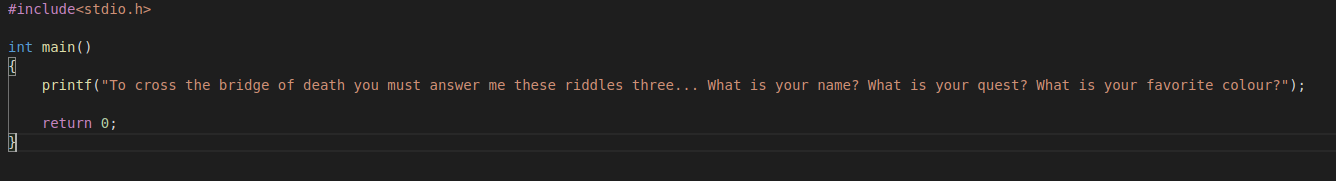
**The .o file is twice as big as the .s file, which is pretty significant**

1. What program can be used to view this type of file. **Name at least 2 program(s)**

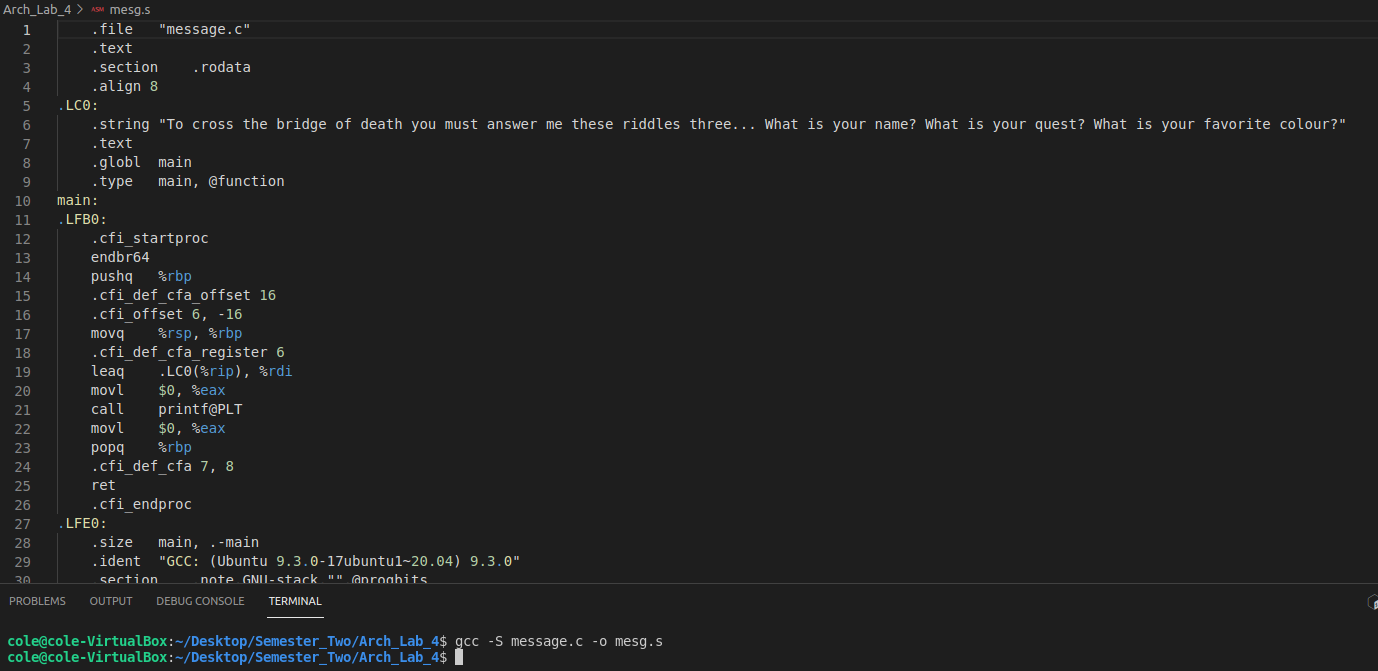
**UNIX/GCC Object File**

**Apple II File**

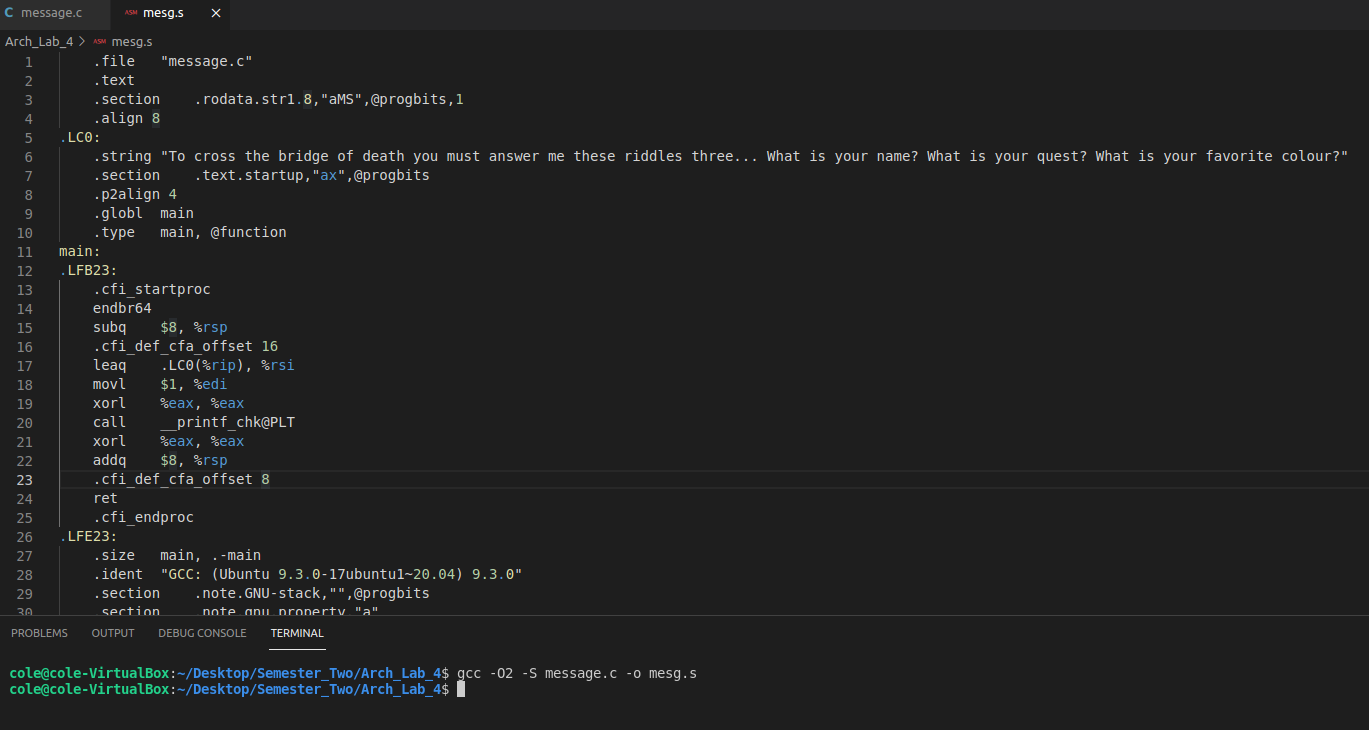
1. Use the man pages to learn about the linker **ld**. Generate an executable program from the .o file you previously created.
2. **(5 marks)** Provide screen captures that demo the following
   1. C code created



* 1. msg.s and command used to generate it



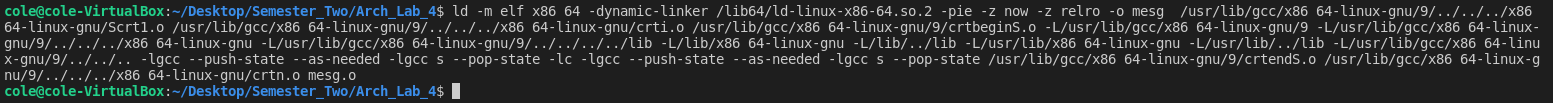
* 1. optimized mesg.s and command used to generated



* 1. msg.o and command used to generat it



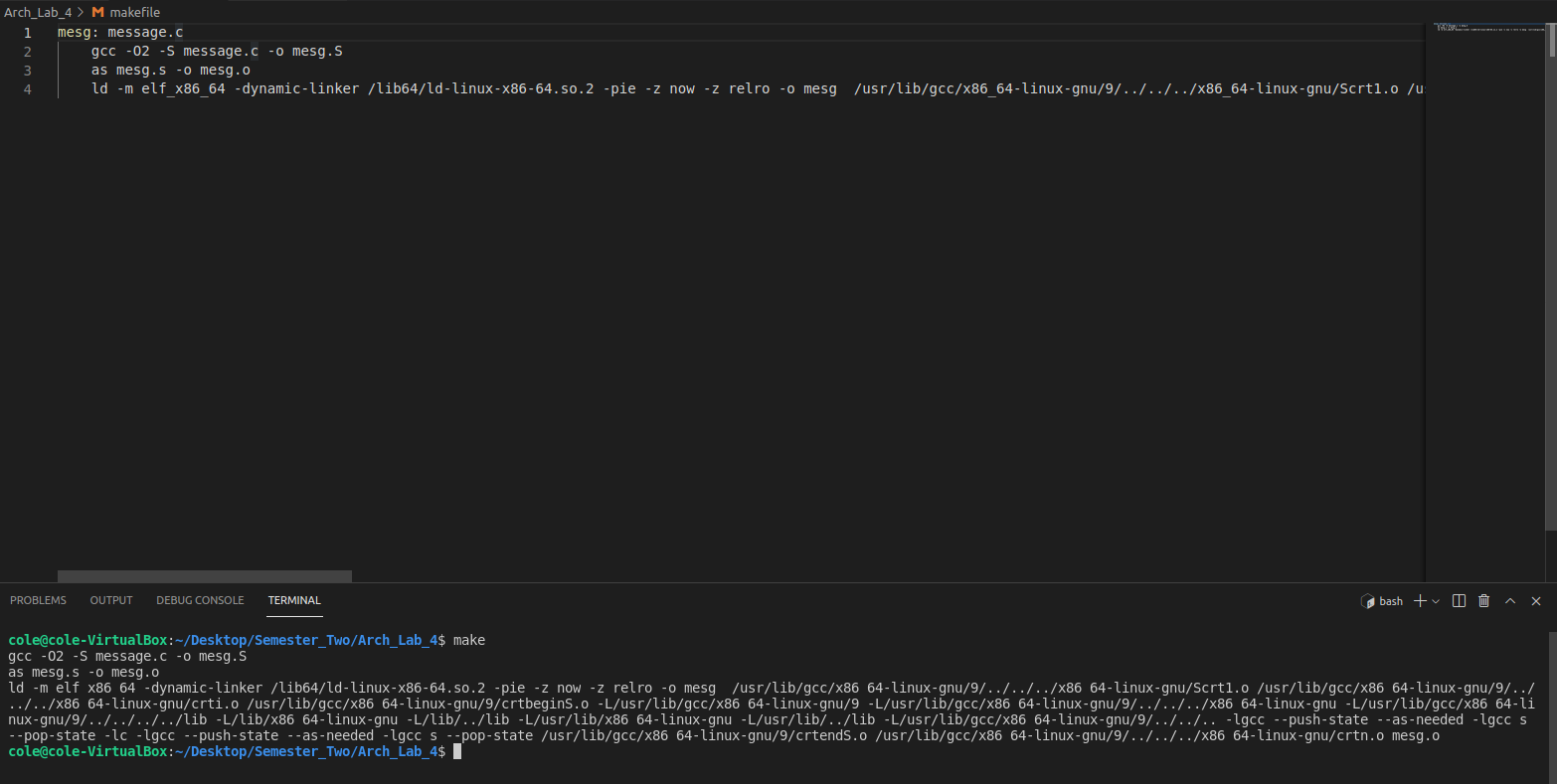
* 1. executable and command used to generated it



1. **(1 mark)** Use Linux man to learn basic of make utility. What is the purpose of make utility?

**The make utility will determine automatically which pieces of a large program need to be recompiled and issue the commands to recompile them.**

1. **(4 marks )** Create a makefile file with basic rules to bquild the executable for message.c and provide the screen capture for. Your makefile should have all the steps of the compilation process you learnt about in class. You screen capture should have the following:
   1. makefile
   2. results after running **make**



# Bonus - Linear Address Translation (paging) \_\_\_10

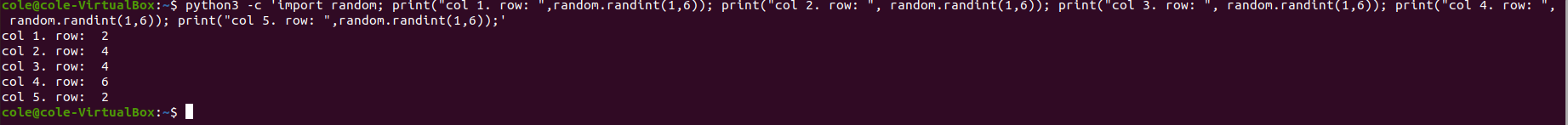
## For details of Linear Address Translation read Intel Manual Volume-3A 4.3 32-bit Paging Use the table from page 2908 of the Intel Manual for the figure “4-2 Linear-Address Translation to a 4Kbyte Page using 32-Bit Paging” required to calculate the physical address below.

1. Use the following python3 command line statement to randomly choose your values required to perform the Linear address translation.

python3 -c 'import random; print("col 1. row: ",random.randint(1,6)); print("col 2. row: ", random.randint(1,6)); print("col 3. row: ", random.randint(1,6)); print("col 4. row: ", random.randint(1,6)); print("col 5. row: ",random.randint(1,6));'



The output should be 5 columns col1, col2, col3, col4, col5 with respective row. The row can be different or the same for each column.

1. Attach the screen capture that demos the results after running python commands
2. Use the previous results to select from the following table the address for:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Linear Address | Page Directory Entry | CR3 | Page Table Entry | Page |
| 1 | 0xCC82F3C2 | 0x5C876420 | 0x36AF78D7 | 0xD0127A5B | 0x1FAA065B |
| 2 | 0xE8667836 | 0xF1916205 | 0xDF0C625F | 0x4E3BC98B | 0xF7175364 |
| 3 | 0xD30BE93F | 0xB4134CFA | 0x67BF0E26 | 0x1564EE26 | 0xFE77B7D |
| 4 | 0xFA8B2CFA | 0xDF38D489 | 0x439A0E0A | 0x698BDBA9 | 0x2A32AE71 |
| 5 | 0x4F9A5195 | 0x7431A25D | 0xC6FCC02B | 0xA1B518C2 | 0x5BFAE708 |
| 6 | 0xDC21C09D | 0x77100F48 | 0x10BAEC4B | 0x1E469881 | 0xDC240E18 |

1. Linear Address:
2. Page Directory Entry:
3. CR3:
4. Page Table Entry:
5. Page :
6. On the table highlight (color or circle) the selected address for each column
7. Use the selected addresses to perform the linear address translation and find the respective physical address
8. Attach a screen capture that demo the process to obtain the physical address. The results and calculations has to be presented in a clear and organized manner. It the results are not clear the grade will be 0 for the question.

Typically, the upper 20 bits of CR3 become the page directory base register (PDBR), which stores the physical address of the first page directory entry  
  
- **https://en.wikipedia.org/wiki/Control\_register#CR3**

“The base physical address of the paging-structure hierarchy is contained in control register CR3. The entries in the paging structures determine the physical address of the base of a page frame, access rights and memory management information”.

* **Intel Manual Section 2.1.5 Memory Management p.2862**