Project 5: Assembly Language Programming

**Due** May 2 by 11:59pm | **Points** 50 | **Submitting** a file upload | **File Types** s | **Available** Apr 18 at 12am - May 2 at 11:59pm 15 days

This project will help you gain familiarity with writing X86-64 Assembly Language Code. We will be writing the function To\_Upper to change the case of a string so that all letters are in upper case.

**Getting Started**

This project has an extremely difficult learning curve for many students, especially if this is your first experience writing assembly code. However, once you figure out the basics of assembly language programming, it’s not difficult and doesn’t take long to finish. However, getting started can be extremely frustrating.  You will need to invest several hours learning the basics, but then only a few minutes writing the code.  Make sure you are caught up with lectures. Then read sections 3.3 through 3.7 of the book.

* Section 3.7 Covers function calls – you will need this information first to write the prologue and epilogue (to set up and clean up the stack)
* Section 3.6 Covers control flow – you will need this for the while loop and conditional statements
* Section 3.5 Covers Arithmetic operations. You will need this to increment your index variable or pointer, change the case of the letter and write the relational operations necessary for the conditionals
* Section 3.4 Covers moving data to/from registers and memory.
* Section 3.3 Covers Data formats – each line of assembly needs to specify how large the data is and which section of the register should be used.

**Project Specification**

Complete the function To\_Upper. A template file is provided as well as a driver. Please rename the template file to “To\_Upper.s” and do all of your work there. This function should traverse a string, and if a letter is lower case, it should change the case to upper case. One driver is provided. We recommend you edit the driver to test additional strings that include not only lower case but upper case letters as well as symbols and numbers.

Our goal is to learn to write correct human-readable assembly code. We are not trying to optimize our code, and the style guide will ask you to intentionally do some things designed to improve your chance of writing correct code at the expense of performance.

The learning goal for this project is to learn to write assembly code. The compiler also produces assembly code. In order to encourage you to do your own work and not just write C code and let the compiler do the project for you, we are going to impose some restrictions.

* **Write the C version of your assembly program** at the top of the file in the indicated section as a comment.
* **Add a table with memory addresses** and corresponding C variable names in the comment provided. Replace the example table.
* Do not use any assembler directives, including call frame information (.cfi\_) statements. (Keep the four provided lines at the top of the template file.)
* Use meaningful labels. Do not use gcc label names (i.e. .L1, .L2, .L3….)
* Follow the style guide below.
* **Use the %rax register for the base address of the string**, use %rdx for the index (if you use array index notation – if you use pointers, you may not need this), and use the %cl register for characters.
  + Example: char c = str[i]
  + Use %rax for str
  + Use %rdx for index, i. (Not needed if you use pointers instead of array index notation)
  + Use %cl for individual character, c
* Assembly code is very difficult to read. Add a comment to every line. Remember inline comments use the # symbol in assembly.
* **Write your own code. Don’t rely on gcc to do this project for you**.
* You are expected to complete this project on the CSL machines.
* Your function should not print anything – comment debugging code.

**Hints**

* Remember letters are represented in ascii. Lower case letter ‘a’ is 97 and upper case letter ‘A’ is 65. This means that to change a lower case letter to an upper case letter you will need to subtract 32 from the value.
* Strings in C are terminated by the ‘\0’ character this value is ascii code 0.
* Passing an array of characters to a function is done by passing a pointer to the first element of the array. The address of this character will be passed in register %rdi.
* Our driver has hard coded the size of the character array. We may test your code with larger arrays (longer strings)

**Files**

* [To\_Upper\_Template.s](https://canvas.wisc.edu/courses/280031/files/25733571?wrap=1)  Rename this this template file to "To\_Upper.s"
* [driver.c](https://canvas.wisc.edu/courses/280031/files/25733568?wrap=1)

If you run the code as provided, the driver will print the original string; then the function has some code you may find helpful for debugging that will print one character. Then the driver will print the string again.

Hint: The provided code that prints a character to help you debug your function will overwrite the registers used to pass parameters, including the register where we pass the address of the string. Remember to store this in memory.

**Compiling and Running on the CSL Machines**

gcc To\_Upper.s driver.c -Wall  
./a.out

**Style Guide for Writing Correct Human Readable Assembly Code**

* Include the C version in a comment
* Include a diagram of the memory layout
* Add comments (most lines will need comments to remind someone else reading your code what variable you are using)
* Break your code into blocks and use white space to keep the blocks organized
* Write the prologue and epilogue together – if you change one, then change the other at the same time.
* Refresh the value in a register every time you use it. If you expect a particular register to have a value left over from a previous command but then edit your code, you may overwrite that register and break sections that had been working.
* Store the value back into memory every time you finish a computation, even if you need it immediately for the next computation.
* Align the op codes, operands, and comments into three columns.
* Indent the opcodes by four spaces, do not indent labels or comments
* Use registers consistently. For example, %rax for base of array, %rdx for the index of an array, %cl for characters
* Get out the table of registers (Section 3.4) of the book. Remember some of them are caller saved, callee saved, or used to pass parameters (also callee saved).
* Use blank lines to divide your code into meaningful blocks
* Use meaningful labels
  + For statement: CONDITION, TRUE, FALSE, END
  + Do while: JUMP\_TO\_CONDITION, BODY, CONDITION, END
* Write your code in the smallest testable pieces possible
* Either use the provided debugging code or gdb to test.

**Useful gdb commands:**

* Start / run
* Stepi / nexti (avoid stepping into printf and putchar)
* Layout asm
* Layout regs
* Info registers
* Break \*0x555555555ABC
* Help x
* x/dw $rbp – 0x14
* x/xg $rbp – 0x0C
* quit