Exercise Assignment 02

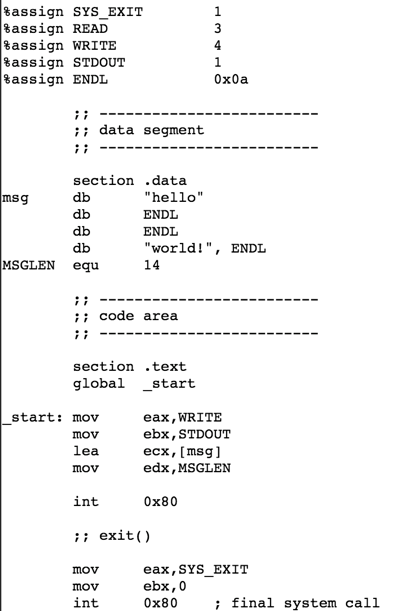
Due September 25, 2016 Sunday 11:59pm (D2L)

|  |  |  |
| --- | --- | --- |
| Name: | Student Number: | Set: |

**Learning Objectives:**

* To write/compile/link/run assembly code, based on today’s tutorial (see tutorial note)
* Understand instruction set and data organization in a program
* Familiarize with system calls to perform tasks such as output and exit
* Experiment with the flow of compile, link and run of: 32-bit, 64-bit programs, other C functions and drivers.

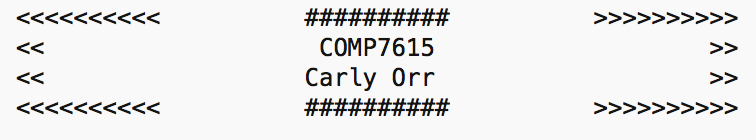
Algorithms Workbench



1. Warm up. Type in the code for both 32-bit and 64-bit version of the hello world program on the right, and provide comments for each line. (ie, two files: greet32.asm, and greet64.asm). In the header section of the code, include how to run, compile, link.

greet.asm

1. Write an assembly language program (name.asm) to display a banner, with your name  (similar to the one below). Try to be efficient: ie, see if you can use “as less data space” as possible, for storing the arrow characters, or the “space” character; and use “as few instructions as possible” for writing out the characters. The banner should be “symmetrical”.



1. Functions work in a similar way to high level languages. Passing parameters can be done using registers or stack. Today we’ll use registers, where you can load eax, ebx, ecx, edx before the function is called.

Write an assembly language program (addnums.asm) that has a function called “sumup” to add 3 numbers. A function precedes with label, and ends with ret. In the main function (global main), call sumup a few times with different values in registers/parameters. For example:

|  |
| --- |
| ….  asm\_main:  mov ebx, 3  mov ecx, 5  mov edx 6  call sumup  mov ebx 15  mov ecx 20  mov edx 30  call sumup  sumup:  ….  ret |

1. Write an assembly language function that will add 3 numbers: 10, 20, 30. Provide a C driver to call this function.
2. Elmo has a home-made computer with the following instruction set.

*Elmo’s Super-Computer Instruction Set*

|  |  |  |
| --- | --- | --- |
| Opcode | Mnemonic | Description |
| 0000 | NOP | No operation |
| 0001 | LOA | Load memory  A |
| 0010 | SUB | Subtract |
| 0011 | ADD | Add with A, result  A |
| 0100 | STR | Store A  memory |
| 0101 | OUT | Output Aconsole |
| 0110 | JMP | Jump to address |
| 0111 | LDI | Load immediate  A |
| 1000 | JC | Jump if carry flag |
| 1111 | HLT | Halt execution |

Elmo sketched on paper a program to add two numbers that are stored in memory.

|  |  |  |
| --- | --- | --- |
|  | Address | Instruction |
| Program section | 0 | LOA [4] |
| 1 | ADD [5] |
| 2 | OUT |
| 3 | HLT |
| Data section | 4 | 14 |
|  | 5 | 28 |

Assuming 4-bit address bus, and 8-bit data bus (ie, 8-bit computer), what will the loaded program (in RAM) look like? Fill in the blanks below:

|  |  |
| --- | --- |
| Address | Data |
| 0000 | 0001 0100 |
| 0001 |  |
| 0010 |  |
| 0011 |  |
| 0100 |  |
| 0101 |  |

Submission D2L

Save answers in file “lastname\_firstname\_0x.docx”

Create a makefile to organize the above programming exercises.

Provide **all files & code** (and any readme files if needed) in zipped folder “lastname\_firstname”.

Marking:

|  |  |
| --- | --- |
|  | Max |
| Completeness | 2 |
| Correctness | 2 |
| Files are well organized | 2 |
| Code is well commented and documented | 2 |
| Proper file naming conventions | 2 |