Exercise Assignment 05: Conditionals, bitwise operations, floating point…

Due Monday October 24st, 2016

**Learning Objections:**

* Use bitwise operations to avoid branching
* Observe floating point operation in assembly

**Notes:**

This week we have 2 exercises dealing with more advanced topics in assembly language.

Please document your code well, using both headers, and in-line comments.

**Tasks:**

Create ***lastname\_firstname\_A5*** folder, and put your exercises inside this folder. Start a **makefile**, where you can add each exercise as you work along.

Warmup and Review: Go over the code samples from “05 tutorial lessons5” document (D2L🡪 Contents🡪 Lecture). Make sure you understand them ☺

1. Avoiding Conditional Branches. Modern processors use “speculative execution” to use parallel processors to execute multiple instructions at once. Processors use fancy techniques to predict if a branch will be taken or not. This can be costly and wasteful of CPU if a branch is not taken.

One way to address this is to avoid conditional branches where possible.

**Trick**: The **SETzz** register command sets the value of a byte register to zero or one based on flags. For example:

**SETG BL** (sets BL=1 if “greater than”)

**SETL BL** (sets BL=1 if “less than”)

**SETZ BL** (sets BL=1 if “zero”)

Based on the value in BL, there are now clever ways to calculate flow without branching.

<https://docs.oracle.com/cd/E19120-01/open.solaris/817-5477/eoizi/index.html>

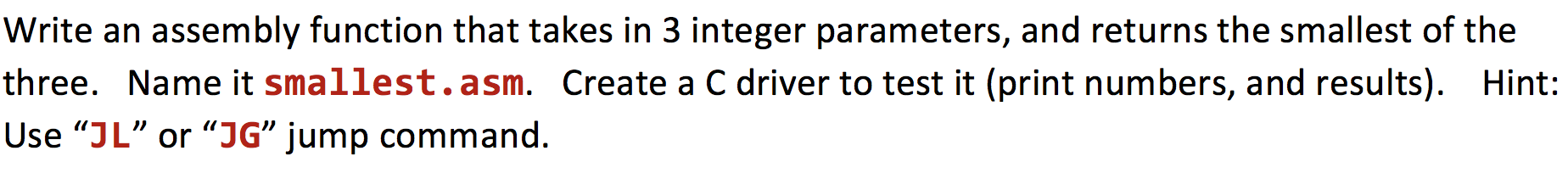
Consider the code ***max.asm*** (attached in D2L) that prints larger of two numbers.

(taken from page 54-55 <http://pacman128.github.io/static/pcasm-book.pdf> )

Study this code and try to understand it.

Now, write an assembly function that takes in 2 input parameters, and returns the “smallest”, that can be called from C driver. Your code must “avoid conditional branching” ☺ Name it “***smaller.asm***”. Provide a driver to test it out.

Then use the snippet for comparing two numbers (create a module or function) to re-do Question #4 from previous week, except do not use JL or JG. Instead, use bitwise tricks. Name it ***smallest\_nojump.asm*** For reference, last week’s question was:



Hint:

* + Use the snippets from ***max.asm*** as reference, and create a function/subroutine to that you can then call from your Question #4 code.
  + Use **SETL** (instead of **SETG**)

1. The Floating Point Unit (FPU) has 8 general purpose 80-bit registers st(0), st(1)…st(7) used for 32-bit x-87 code. (FYI There is another 64-bit coprocessor stack that evolved later.) All the arithmetic is done using these registers.

To give you a flavor of how it works here are a few instructions.

|  |  |
| --- | --- |
| **FLD src** | Push source operand (a float) onto the FPU register stack  <http://x86.renejeschke.de/html/file_module_x86_id_100.html> |
| **FILD src** | Push source operand (an integer) onto FPU register stack  <https://cs.fit.edu/~mmahoney/cse3101/float.html> |
| **FLDZ** | Push zero onto the FPU register stack |
| **FADD (no operand)** | Add the contents of st0 to st1, result in st0  <http://x86.renejeschke.de/html/file_module_x86_id_81.html> |
| **FSTP dest** | Copy the value of st0 to the destination (store floating point value)  <http://x86.renejeschke.de/html/file_module_x86_id_117.html> |
| **FDIV src** | Divide contents of st0 by contents of a memory location (cannot be immediate), result in st0  <http://x86.renejeschke.de/html/file_module_x86_id_91.html> |

Consider floating point program ***sumfloat.asm*** Study it. Now, modify the program:

* + Create an array of say 5 floating point numbers
  + Calculates the average of the numbers
  + Name it ***avefloat.asm***

Hint:

* + Declare your array in the Data section (“*table dd*”)
  + Declare a constant (N) in the data section (“*N equ ($-table)/4*”) to represent the size of the array that you have
  + Since the constant N is an integer (ie, 2 and not 2.0), and FDIV takes as operand a memory location, before you divide, you must:
    - First move N to a memory location (say, TEMP)
    - Then use FILD to load **integer** onto FPU stack st0 location

**Trivia:** Did you know that the famous Pentium FDIV bug in 1994 (discovered after production release) caused intel $500 million loss associated with fix and replacement of flawed processors. <https://en.wikipedia.org/wiki/Pentium_FDIV_bug> It was so famous that many jokes were made about intel <http://www.columbia.edu/~sss31/rainbow/pentium.jokes.html>

**Marking:**

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