CSEE 3827: Homework 4 — Due: 4/14/10

- *To turnin this homework, create and submit a gzipped tarball called your_uni.tar.gz. Unpacking this tarball should create a directory named with your UNI containing three files: squares.s, priority_encode.s, bubble_sort.s.*
- *The source files for this assignment are available here: http://www.cs.columbia.edu/ martha/courses/3827/sp10/hw/templates.tar.gz.*
- *A brief tutorial on the MIPS simulator, XSpim, is available here: http://www.cs.columbia.edu/ martha/-courses/3827/sp10/hw/xspim.pdf.*

1. **squares.s** The purpose of this problem is to have you write a simple program and get you accustomed to using xspim.

```
################################################################
#
# Write a little program to print the first ten squares.
# The output should be:
#       1
#       4
#       9
#       ..
#       100
#
# Note that a print function has been provided.  You call it
# by placing the integer to be printed in $a0 and then jal print'
#
# You should again use this template, implementing your code in the
# main body where indicated.
#
################################################################

.data
newline: .asciiz "\n"

.text
.globl main

main:
##
##  Implement this function
##
  li $v0, 10            # exit
  syscall

print:
  li $v0, 1
  syscall
  li $v0, 4
  la $a0, newline
  syscall
  jr $ra
```

2. **priority_encode.s** Using the template below, implement (yet another) priority encoder. See the specification at the top of the file.

```
################################################################
#
# Implement the priority_encode() function, which will take a
# single 32-bit argument in $a0 and return the position of the
# most-significant 1 in $v0.
#
# You should use this template, implementing priority_encode
# function where indicated.  Your function should be able to
# correctly handle any 32-bit input value. (I.e., do not write
# a function that handles only the test inputs called from main)
#
################################################################

.data
newline: .asciiz "\n"

.text
.globl main

main:
  addi $a0, $zero, 1         # priority_encode(1) -> 0
  jal priority_encode
  move $a0, $v0
  jal print

  addi $a0, $zero, 10        # priority_encode(1010) -> 3
  jal priority_encode
  move $a0, $v0
  jal print

  addi $a0, $zero, 100       # priority_encode(1100100) -> 6
  jal priority_encode
  move $a0, $v0
  jal print

  li $v0, 10                 # exit
  syscall

priority_encode:

##
##  Write your code here
##

print:
  li $v0, 1
  syscall
  li $v0, 4
  la $a0, newline
  syscall
  jr $ra
```

3. **bubble_sort.s** Finally, using this third template, implement a function that will bubble sort a string. The specification is again at the top of the file. You may find the sorting example in 2.13 of P&H instructive, but be aware of the differences between the example and this problem.

```
#############################################################
#
# Implement the bubble_sort() function which takes a pointer to
# a string in $a0 and applies the bubble sort algorithm
# (http://en.wikipedia.org/wiki/Bubble_sort) to sort the
# characters of the string in place (i.e., using the same memory).
#
# The correct output on the string
# "Bubble sort the letters in this string." is
# "       .Bbbeeeeghhiiillnnorrrsssttttttu"
#
# You should again use this template, implementing bubble_sort
# function where indicated.  Your function should be able to
# correctly handle any input string, not just the one shown here.
#
#############################################################

.data
string: .asciiz "Bubble sort the letters in this string."

.text
.globl main

main:
  la $a0, string
  jal bubble_sort
  la $a0, string
  jal print

  li $v0, 10                    # exit
  syscall

bubble_sort:
##
##  Implement this function
##

print:
  li $v0, 4
  la $a0, string
  syscall
  jr $ra
```