Homework Assignment: Ch2

Group Information	
Number:	Name/email:
Number:	Name/email:
Number:	Name/email:

Author: Mark McKenney

Organization: CS286, Department of Computer Science, SIUE

Overview

The purpose of this assignment is to gain familiarity with MIPS assembly language programming. For this assignment, you will create a MIPS implementation of a bubble sort algorithm. The program must run in the MIPS simulator and print out the sorted array.

Simulator

We will use the MARS simulator found here:
https://courses.missouristate.edu/KenVollmar/MARS/download.htm

Other MIPS simulators (notably SPIM and its Javascript implementation JSPIM) out there. However, our assignemnt requires random number generation which is provided by the MARS simulator, and not the others. MARS is also a bit more intuitive. You solution must run in the MARS simulator with no errors and it must print out the sorted numbers.

To invoke the simulator on a linux machine, use the flag to execute a JAR file: *java -jar Mars4_5.jar*

Algorithm

You may implement any sort routine you like. Bubble sort is the simplest to implement, so I suggest that one. A C implementaion of bubble sort is as follows:

Create MIPS assembly code for the following C function. It is an insertion sort program.

```
#include <stdio.h>
#include <stdlib.h>

int main(){

        int array [5000];

        for( int i = 0; i < 5000; i++ )
                array[i] = rand()%100;

        for(int i = 0; i < 4999; i++)
                for( int j =0; j < 4999-i; j++ )
                        if( array[j+1] < array[j] ){
```

```
                                                int tmp = array[j];
                                                array[j] = array[j+1];
                                                array[j+1] = tmp;
                                        }
                for( int i = 0; i < 5000; i++ )
                        printf( "%i\n", array[i]);
        }
```

MIPS Implementation Notes

I am providing a basic file to get you started. You should be able to paste the file into the MARS simulator and run it. a couple of notes:

syscall

Printing output and generating random numbers are donw via system calls. To execute a system call, you must put the correct arguments into the *$v0*, *$a0*, and *$a1* registers. once those are in the correct place, you simply call *syscall.* The system call basically just calls a function, and chooses which function to execute (print, get a random number, etc) via the arguments. The full list of system calls and their arguments are found here:

https://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html

We will use the following system calls:

- code 1 to print integers (put the integer to print in $a0)
- code 4 to print a string (a newline character in our case)
- code 42 to get a random number

The following code shows how to get a random number, then print it

```
li $a0, 0                # set up random number system call.
                         # use generator 0
li $a1, 100              # max random number is 100
li $v0, 42               # syscall 42 is random number in a range.
                         # Result goes in $a0
syscall
li $v0, 1                # set up to print int with syscall number 1
syscall
```

Allocating Arrays

To allocate an array, we will simply create space on the stack and use that. Remember that the stack starts at a high address, and grows downwards towards lower addresses. Therefore, to reserve space for 5 integers on the stack, you simply subtact 20 (4 bytes per integer times 5 integers) from the stack pointer *$sp.* you may then use that space to load and store numbers to. Remember, it is up to you if you want element 0 of your array to be at the higher address of the space, or the lower address!

Labels

To create functions, or points in the code for you to use a branch or jump targets, you must create a label. A label is simply some text followed by a colon. The following block demonstrates.

```
beq $t1, $t0, TARGET
addi $t1, $t1, 5
TARGET:
addi $t0, $t0, 2
```

To make a function, create a label where the function starts, and then *jump and link* to that function:

```
# the function
FUNC1:
addi $t0, $a1, 10
jr $ra


#call the function
jal FUNC1
```

Assignment

You must create a MIPS program that runs in the MARS simulator. The program must do a sort (bubble sort is probably the easiest) and it must sort an array of 30 integers. The array must be allocated ON THE STACK, as shown above. The program must print the sorted numbers at the end.

Basic Example Program

The following is a basic example program that allocates an array of 2 integers on the stack, puts random numbers in it, and prints it out.

```
.data
NEWLINE: .asciiz "\n"

.text
.globl main     #required

main:           #required

addi $sp, $sp, -8  # make space for 2 ints, 8 bytes on the stack

li $a0, 0          #set up random number system call.  use generator 0
li $a1, 100        # max random number is 100
li $v0, 42         # syscall 42 is random number in a range

syscall         # random number is now in $a0
```

```
sw $a0, 0($sp)  # store the number in array location [0]
syscall         # random number is now in $a0
sw $a0, 4($sp)  # store the number in array location [1]
                # NOTE I am making the array atart at a lower address
                # and go UP in memory!

#now print out the numbers
li $v0, 1               #set up to print int
lw $a0, 0( $sp )        #get the int to print
syscall
li $v0, 4               #set up to print a string
la $a0, NEWLINE         #la is LOAD ADDRESS.  newline is declared
                        #at the start of the program
syscall

li $v0, 1               #set up to print int
lw $a0, 4( $sp )        #get the int to print
syscall
li $v0, 4               #set up to print a string
la $a0, NEWLINE         #la is LOAD ADDRESS.  newline is declared
                        #at the start of the program
syscall

li $v0, 10              # last 2 lines are required to make program exit
syscall
```