/\*

\* Engr220L - Lab 8

\* Date: <TODO>

\* Author: Jacob Bennett <TODO>

\*

\* This assembly code defines the BinarySearch() function

\* as explained below. It is intended for use in Engr220L-#8

\* with the associated Lab 8 main.c file.

\*

\*/

/\*

C-STYLE DECLARATION:

int BinarySearch(int\* arrayToSearchIn, int valueBeingSearchedFor,

unsigned int startIndex, unsigned int endIndex);

REGISTERS:

<TODO: list the registers you use within the function here

including a description of each. Hint - relate these

directly to the C-Code variables, parameters, etc.

A possible solution could use 7 registers. >

CASES:

<TODO: list the recursive and base-case options here. This will

help relate the assembly and C-code. Hint - take these

directly from the algorithm explained in the C-Code and

explain how they will operate in terms of the registers instead. >

\*/

.global BinarySearch

BinarySearch:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* push to stack \*/

/\* <TODO: list the registers that need to be protected on the stack

in a comment, then push their values onto the stack.

Hint - A possible solution could require 5 registers to

be protected, including ra. Lab7 slide deck shows an example. > \*/

subi sp, sp, 12

stw ra, 0(sp)

stw r16, 4(sp)

stw r17, 8(sp)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* calculate local variables \*/

/\* <TODO: setup the values of the registers in such a way that the

following code can check which of the cases we are currently in.

Hint - this corresponds to the C-Code local variable assignments. > \*/

Top\_of\_loop:

sub r20, r7, r6 /\* r16 = midIndex = start + (end - start)/2 \*/

srli r20, r20, 1

add r16, r6, r20

slli r21, r16, 2 /\* calculate offset \*/

add r22, r21, r4

ldw r17, 0(r22) /\* r17 = midValue = arrayToSearchIn[midIndex]; \*/

Check\_Case\_equal:

bne r5, r17, Check\_not\_found /\* branch if not equal \*/

Case\_equal:

sub r2, r2, r2 /\* resetting r2 to 0 \*/

addi r2, r16, 0

br Start\_Return

Check\_not\_found:

bne r6, r7, Check\_smaller

Case\_not\_found:

sub r2, r2, r2 /\* resetting r2 to 0 \*/

subi r2, r0, 1 /\* return r2 as -1 not found \*/

br Start\_Return

Check\_smaller:

bgt r5, r17, Check\_bigger

sub r7, r7, r7

subi r7, r16, 1 /\* endindex = midindex - 1 \*/

call BinarySearch

br Start\_Return

Check\_bigger:

addi r6, r16, 1 /\* startIndex = midindex + 1 \*/

call BinarySearch

br Start\_Return

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* check cases \*/

/\* <TODO: check which case we are in and take the appropriate action.

Hint - use conditional branches to "skip around" cases that we are not in.

Hint - the action we take in each case is to set the return value and

then proceed to the end without checking for any other cases. > \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* pop from stack \*/

/\* \*/

Start\_Return:

ldw ra, 0(sp)

ldw r16, 4(sp)

ldw r17, 8(sp)

addi sp, sp, 12

/\* <TODO: list the registers that need to be protected on the stack

in a comment, then pop their values off the stack.

Hint - A possible solution could require 5 registers to

be protected, including ra. Lab7 slide deck shows an example.

Hint - this needs to happen in all cases, don't "skip around" it. > \*/

ret