CSE 30321 – Computer Architecture I – Fall 2011
Homework 04 – MIPS Procedure Calls – 60 points
Assigned: September 20, 2011 **– Due:** September 27, 2011

Problem 1: (5 points)

Translate the following sequence of ARM assembly instructions to MIPS assembly instructions.

```
        MOV     R7, #10         # initialize loop counter to 10
Loop:   ADD     R6, R6, R5      # add R5 and R6, store the result back in R6
        SUBiS   R7, R7, 1       # decrement the loop counter
        BNE     Loop            # if condition code is FALSE, repeat the loop
```

Notes / suggestions / things to think about:

- You may want to refer back to HW 01, P. 05 and/or HW 02, P. 02 for a refresher on ARM syntax
- This problem has been included to re-enforce an important idea that has been brought up in class several times. Namely, when writing assembly code for *different* instruction set architectures (ISAs), you cannot "mix and match" instructions from one ISA with that of another.
    - For example, in the provided code, an instruction like "beq $0, $0, next" would not appear.
    - A good analogy is the syntax required to implement a for loop in Matlab code or C code. Conceptually, the loops could iterate through an array of data in exactly the same way. However, the syntax required for the loop to execute correctly within the Matlab environment or to compile (in the case of the C-code) is different. Similarly, in both the ARM and MIPS ISAs, there is a way to compare the contents of a register to 0 and branch accordingly, but the assembly syntax (and thus how the processor hardware performs the operation) is different.

Problem 2: (10 points)

The C-code for an *array-based swap procedure* and the corresponding MIPS assembly appears below:

C-code:

```
void swap(int v[], int k, int j) {
        int temp;
        temp = v[k];
        v[k] = v[j];
        v[j] = temp;
}
```

MIPS assembly:

```
swap:
        sll     $t0, $a1, 2
        add     $t0, $t0, $a0
        lw      $t2, 0($t0)
        sll     $t1, $a2, 2
        add     $t1, $t1, $a0
        lw      $t3, 0($t1)
        sw      $t3, 0($t0)
        sw      $t2, 0($t1)
        jr      $ra
```

Part A (6 points):

The C-code for a *pointer-based swap procedure* appears below. Write the MIPS assembly.

C-code:

```
void swap(int *p) {
        int temp;
        temp = *p;
        *p = *(p+1);
        *(p+1) = temp;
}
```

Notes:

- Write your code as efficiently as possible.
- You should assume that $s registers, the $ra register, etc. DO NOT need to be saved.

Part B (4 points):

Which version is faster – the array-based version or the pointer-based version? By how much?

Problem 3: (30 points)

Two C-based versions of the function *find* are provided below – an *array-based version* and a *pointer-based version*. You should assume that (i) find is a leaf procedure and (ii) no $s registers, the return address register, etc. need to be saved to the stack. Note that the array *index* is returned.

Array-based version:

```
int find(int a[], int n, int x) {
      int i;
      for(i=0; i!=n; i++) {
            if(a[i] == x) {
                  return i;
            }
      }
      return -1;
}
```

Pointer-based version:

```
int find(int *a, int n, int x) {
      int *p;
      for(p=a; p!=a+n; p++) {
            if(*p==x) {
                  return p-a;
            }
      }
      return -1;
}
```

Part A (15 points):

Translate the array-based version into MIPS assembly.

Part B (15 points):

Translate the pointer-based version into MIPS assembly.

Problem 4: (15 points)

Assume that main() calls a leaf procedure *function* – which takes 5 arguments.

Part A (5 points):

Following the MIPS procedure call conventions, write the MIPS assembly instructions to callee save the first three $sx registers (i.e. $s0, $s1, and $s2). As part of your answer, explain where these instructions would go – i.e. where in main, where in *function*, etc. Remember the stack starts at higher memory addresses and grows "down" toward lower memory addresses

Part B (5 points):

Assume that main needs to use $t4 and $t5 after the call to *function*.

i. According to the MIPS convention, where are these registers saved?
ii. Write the MIPS assembly to RESTORE these registers.
iii. Where would these assembly instructions go?

Part C (5 points):

How would the 5 arguments be passed to *function*? A text description is sufficient – i.e no MIPS assembly need be written.