

Homework #2 Problems**Quiz in Recitations on Monday, October 26 & Wednesday, October 28**

Problem 1: Java versus MIPS: Which statements (if any) are true?

1. Assignment statements: One variable on left-hand side in Java; one variable (register) is destination in MIPS.
2. Assignment statements: Any number of variables on right-hand side in Java; 1 or 2 (registers) source in MIPS.
3. Comments: /* . . . */ and // to end of line in Java; and # to end of line in MIPS
4. Variables: declared in Java; no declaration in MIPS.
5. Types: Associated with declaration in Java (normally); associated with instruction (operator) in MIPS.

Problem 2: Construct a control flow graph (like the one shown in Fig. 2.9 in your textbook) for the following Java code. Write the corresponding MIPS instructions.

```
for (i = 20; i < x; i++)
    y = y + i;
```

Problem 3: Add comments to the following MIPS code and describe what it computes. Assume that \$a0 and \$a1 are used for the input and both contain integers a and b, respectively. Assume that \$v0 is used for the output.

```

loop:      add $t0, $zero, $zero
          beq $a1, $zero, finish
          add $t0, $t0, $a0
          sub $a1, $a1, 1
          j loop
finish:    addi $t0, $t0, 100
          add $v0, $t0, $zero
```

Problem 4: Consider the following fragment of Java code:

```
for (i = 0; i <= 100; i++)
    a[i] = b[i] + c;
```

Assume that a and b are arrays of words and the base address of a is in \$a0 and the base address of b is in \$a1. Register \$t0 is associated with variable i and register \$s0 with c. Write the code for MIPS. How many instructions are executed during the running of this code? How many memory data references will be made during execution?

Problem 5: The following program tries to copy words from the address of register \$a0 to the address in register \$a1., counting the number of words copied in register \$v0. The program stops counting when it finds a word equal to 0. You don't have to preserve the contents of registers \$v1, \$a0, and \$a1. This terminating word should be copied, but not counted.

```

loop:      addi $v0, $zero, 0 # Initialize count
          lw $v1, 0($a0) # Read next word from source
          sw $v1, 0($a1) # Write to the destination
          addi $a0, $a0, 4 # Advance pointer to next source
          addi $a1, $a1, 4 # Advance pointer to next destination
          beq $v1, $zero, loop # Loop if word copied != zero
```

There are multiple bugs in this MIPS program; fix them. How many bugs are there? (Easiest way to find bugs is to write the MIPS program and run it in the SPIM simulator.)

Problem 6: Show the single MIPS instruction or minimal sequence of instructions for this Java statement:

```
x[7] = x[2] + a;
```

Assume that a corresponds to register \$t3 and the array x has a base address of 3,200,000₁₀.

Problem 7: The MIPS translation of the Java segment

```
while (save[i] == k)
    i++;
```

on pages 107-108 of your textbook uses both a conditional branch and an unconditional jump each time through the loop. Only poor compilers would produce code with this loop overhead. Rewrite the assembly code so that it uses at most one branch or jump each time through the loop. How many instructions are executed before and after the optimization if the number of iterations of the loop is 10 (i.e., `save[i + 10*j]` do not equal `k` and `save[i], ..., save[i+9*j]` equal `k`)?

Problem 8: Implement the following Java code in MIPS.

```
int sub (int a, int b) { return a-b; }
int myMethod () {
    int i = 0;
    int sum = 0;
    int array[] = {0,1,2,3,4,5,6,7,8,9};
    for (i = 1; i < 10; i++) {
        array[i-1] = sub(array[i], array[i-1]);
        sum += array[i-1];
    }
    System.out.println("Sum is " + sum);
    return 0;
}
```

Be sure to handle the stack and the frame pointers appropriately. Draw the status of the stack before calling `sub` and during the function call. Indicate the names of registers and variable stored on the stack and mark the location of `$sp` and `$fp`.

Problem 9: The following MIPS instruction sequence could be used to implement a new instruction that has two register operands. Give the instruction a name and describe what it does. Note that register `$t0` is being used as a temporary.

```
srl $s1, $s1, 1      #
sll $t0, $s0, 31     # These four instructions accomplish
srl $s0, $s0, 1      # "new $s0 $s1"
or $s1, $s1, $t0     #
```