

Homework #2 Problems

Quiz in Recitations on Monday, October 26 & Wednesday, October 28

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

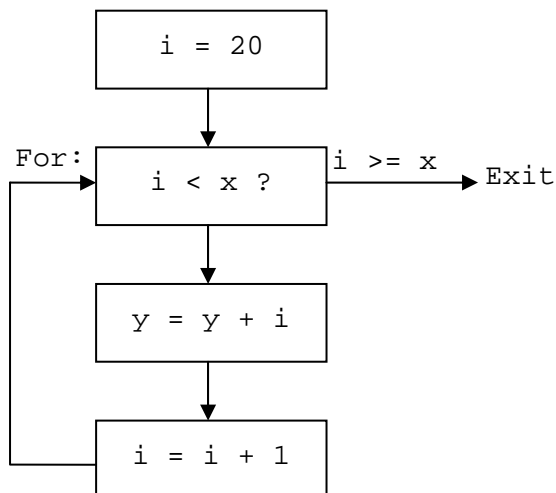
1. Assignment statements: One variable on left-hand side in Java; one variable (register) is destination in MIPS.
2. Assignment statements: Any number of variables on right-hand side in Java; 1 or 2 (registers) source in MIPS.
3. Comments: /* . . . */ in Java; // to end of line in Java; and # to end 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.

ANSWER: All are true

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;
```

ANSWER:



```

For:      addi $t0, $zero, 20      # $t0 holds i, i = 20
          bge $t0, $s0, Exit   # Assume x in $s0, i >= x?
          add $s1, $s1, $t0    # $s1 holds y, y = y + i
          addi $t0, $t0, 1     # i = i + 1
          j For                # jump to label For
Exit:

```

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

```

ANSWER:

```

loop:     add $t0, $zero, $zero      # initialize running sum $t0 = 0
          beq $a1, $zero, finish    # finished when $a1 is 0
          add $t0, $t0, $a0         # compute running sum of $a0
          sub $a1, $a1, 1           # compute this $a1 times

```

```

j loop
finish:   addi $t0, $t0, 100      # add 100 to a * b
          add $v0, $t0, $zero    # return a * b + 100

```

The program computes $a * b + 100$.

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?

ANSWER:

The fragment of Java code is

```

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

```

with a and b arrays of words at base addresses $\$a0$ and $\$a1$, respectively. First initialize i to 0 with i kept in $\$t0$:

```

add $t0, $zero, $zero      # Temp reg $t0 = 0

```

Assume that $\$s0$ holds the address of c (if $\$s0$ is assumed to hold the value of c , omit the following instruction):

```

lw $t1, 0($s0)            # Temp reg $t1 = c

```

Now the loop body accesses array elements, performs the computation, and tests for termination:

```

Loop:   add $t2, $a1, $t0   # Temp reg $t2 = address of b[i]
        lw $t3, 0($t2)     # Temp reg $t3 = b[i]
        add $t4, $t3, $t1  # Temp reg $t4 = b[i] + c

```

This `add` instruction would be `add $t4, $t3, $s0` if it were assumed that $\$s0$ holds the value of c .

Continuing the loop:

```

add $t5, $a0, $t0         # Temp reg $t5 = address of a[i]
sw $t4, 0($t5)            # a[i] = b[i] + c
addi $t0, $t0, 4          # $t0 = $t0 + 4, i.e., i = i + 1
slti $t6, $t0, 401        # $t6 = 1 if $t0 < 401, i.e., i <= 100
bne $t6, $zero, Loop     # go to Loop if i <= 100

```

The number of instructions executed is $2 + 101 \cdot 8 = 810$. The number of data references made is $1 + 101 \cdot 2 = 203$.

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 not 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.)

ANSWER:

```

loop:   addi $v0, $zero, -1 # Initialize to avoid counting zero word
        lw $v1, 0($a0)     # Read next word from source
        addi $v0, $v0, 1   # Increment count words copied
        sw $v1, 0($a1)    # Write to destination

```

```

    addi $a0, $a0, 4      # Advance pointer to next source
    addi $a1, $a1, 4      # Advance pointer to next destination
    bne $v1, $zero, loop # Loop if word copied != zero

```

Bug 1: Count (\$v0) is initialized to zero, not -1 to avoid counting zero word.

Bug 2: Count (\$v0) is not incremented.

Bug 3: Loops if word copied is equal to zero rather than not equal.

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`.

ANSWER:

The base address of `x`, in binary, is `0000 0000 0011 0000 1101 0100 0000 0000` which implies that we must use `lui`:

```

    lui $t0, 0x30          # load the address of x into $t0
    ori $t0, $t0, 0xD800
    lw $t1, 8($t0)        # $t1 = Memory[4*2 + $t0]
    add $t1, $t1, $t3     # $t1 = x[2] + a
    sw $t1, 28($t0)      # x[7] = x[2] + a

```

Problem 7: The MIPS translation of the Java segment

```

    while (save[i] == k)
        i += 1;

```

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`)?

ANSWER:

The Java loop is

```

    while (save[i] == k)
        i += 1;

```

with `i` and `k` corresponding to registers `$s3` and `$s5` and the base of the array `save` in `$s6`. The assembly code given in the example is

Code before

```

Loop:    sll $t1, $s3, 2 # Temp reg $t1 = 4 * i
         add $t1, $t1, $s6 # $t1 = address of save [i]
         lw $t0, 0($t1) # Temp reg $t0 = save[i]
         bne $t0, $s5, Exit # go to Exit if save[i] != k
         addi $s3, $s3, 1 # i = i + 1
         j Loop # go to Loop

Exit:

```

Number of instructions executed if `save[i + m]` does not equal `k` for `m = 10` and does equal `k` for `0 <= m <= 9` is 10. `6 + 4 = 64`, which corresponds to 10 complete iterations of the loop plus a final pass that goes to `Exit` at the `bne` instruction before updating `i`. Straightforward rewriting to use at most one branch or jump in the loop yields

Code after

```

Loop:    sll $t1, $s3, 2 # Temp reg $t1 = 4 * i
         add $t1, $t1, $s6 # $t1 = address of save [i]
         lw $t0, 0($t1) # Temp reg $t0 = save[i]
         bne $t0, $s5, Exit # go to Exit if save[i] != k
         addi $s3, $s3, 1 # i = i + 1
         sll $t1, $s3, 2 # Temp reg $t1 = 4 * i
         add $t1, $t1, $s6 # $t1 = address of save [i]

```

```

    lw $t0, 0($t1) # Temp reg $t0 = save[i]
    beq $t0, $s5, Loop # go to Loop if save[i] = k

```

Exit:

The number of instructions executed by this new form of the loop is $4 + 10 \cdot 5 = 54$. If $\$t1$ is incremented by 4 inside the loop, then further saving in the loop body is possible.

Code after further improvement

```

    sll $t1, $s3, 2 # Temp reg $t1 = 4 * i
    add $t1, $t1, $s6 # $t1 = address of save[i]
    lw $t0, 0($t1) # Temp reg $t0 = save[i]
    bne $t0, $s5, Exit # go to Exit if save[i] != k
Loop:
    addi $s3, $s3, 1 # i = i + 1
    add $t1, $t1, 4 # $t1 = address of save[i]
    lw $t0, 0($t1) # Temp reg $t0 = save[i]
    beq $t0, $s5, Loop # go to Loop if save[i] = k
Exit:

```

The number of instructions executed is now $4 + 10 \cdot 4 = 44$.

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 $\$fp$.

ANSWER:

```

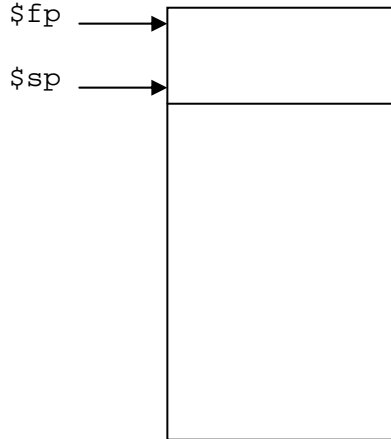
    add $s1, $zero, $zero      # sum = 0
    add $s0, $zero, $zero      # i = 0
    addi $t0, $zero, 9         # max iterations is 9
    la $s2, array              # $s2 points to elements of array
loop:
    sll $t1, $s0, 2            # $t1 = i * 4
    add $t2, $s2, $t1          # $t2 = address of array[i]
    lw $a0, 0($t2)             # pass array[i]
    sll $t3, $t1, 2            # $t3 = address of array[i+1]
    lw $a1, 0($t3)             # pass array[i+1]
    jal sub                    # call sub(array[i+1], array[i])
    sw $v0, 0($t2)             # array[i] = sub(array[i+1], array[i])
    lw $t4, 0($t2)             # load value of array[i]
    add $s1, $s1, $t4          # sum = sum + array[i]
    addi $s0, $s0, 1           # i++
    bne $s0, $t0, loop         # loop if i<9
    la $a0, STR                 # load string to print: STR
    li $v0, 4                   # load string type for syscall
    syscall
    la $a0, $s1                 # load value of sum
    li $v0, 1                   # load integer type for syscall
    syscall
    la $a0, ENDL                # load string to print: ENDL
    li $v0, 4                   # load string type for syscall
    syscall

```

```

        li $v0, 10          # load exit type for syscall
        syscall
sub:    sub $v0, $a0, $a1   # return a-b
        jr $ra            # return
.data
array: .word 0,1,2,3,4,5,6,7,8,9
STR:   .asciiz "Sum is "
ENDL:  .asciiz "\n"

```



Before sub, during sub, and after sub (sub is a simple function, which doesn't change any registers. Therefore there is no need to push/pop anything from the stack.

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        #

```

ANSWER:

The new instruction treats \$s0 and \$s1 as a register pair and performs a shift, wherein the least significant bit of \$s0 becomes the most significant bit of \$s1 and both \$s0 and \$s1 are shifted right one place.