Lab2 & Ch2_review_2

Yuanqing Miao

Lab2

Set a breakpoint for the instruction at line 13 of the assembler source code.

```
loop:
      lw     $t6, 28($sp)    # i
      mul    $t7, $t6, $t6   # i * i
      lw     $t8, 24($sp)    # sum
      addu   $t9, $t8, $t7   # sum + i*i
```

1. For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
sll $t0, $s0, 2                # $t0 = f * 4
add $t0, $s6, $t0         # $t0 = &A[f]
sll $t1, $s1, 2                # $t1 = g * 4
add $t1, $s7, $t1         # $t1 = &B[g]
lw $s0, 0($t0)                # $s0 = A[f]
addi $t2, $t0, 4            # $t2 = address of A[f+1]
lw $t0, 0($t2)                # $t0 = A[f+1]
add $t0, $t0, $s0         # $t0 = A[f+1] + A[f]
sw $t0, 0($t1)               # B[g] = t0
```

```
B[g] = A[f + 1] + A[f];
```

2. The table below shows 32-bit values of an array stored in memory.

Address	Data
24	2
28	4
32	3
36	6
40	1

For the memory locations in the table above, the following C code is sorting the data from lowest to highest, placing the lowest value in the smallest memory location shown in the figure. The data shown represents the C variable called Array, which is an array of type int, and that the first number in the array shown is the first element in the array. Assume that this particular machine is a byte-addressable machine and a word consists of four bytes.

```
temp = Array[0];
temp2 = Array[1];
Array[0] = Array[4];
Array[1] = temp;
Array[4] = Array[3];
Array[3] = temp2;
```

a. For the memory locations in the table above, write MIPS code to sort the data from lowest to highest, placing the lowest value in the smallest memory location. Assume the base address of Array is stored in register $s6.

```
temp = Array[0];
temp2 = Array[1];
Array[0] = Array[4];
Array[1] = temp;
Array[4] = Array[3];
Array[3] = temp2;
```

```
lw $t0, 0($s6)
lw $t1, 4($s6)
lw $t2, 16($s6)
sw $t2, 0($s6)
sw $t0, 4($s6)
lw $t0, 12($s6)
sw $t0, 16($s6)
sw $t1, 12($s6)
```

3. Translate function f into MIPS assembly language. If you need to use registers `$t0` through `$t7`, use the lower-numbered registers first. Assume the function declaration for `func` is `"int func(int a, int b);"`. The code for function f is as follows:

```
int f(int a, int b, int c, int d){
   return func(func(a, b), c + d);
}
```

```
f:
    addi $sp, $sp, -12          # Allocate stack space
    sw $ra, 8($sp)              # Save return address
    sw $s1, 4($sp)              # Save register $s1
    sw $s0, 0($sp)              # Save register $s0

    move $s1, $a2               # Store c in $s1
    move $s0, $a3               # Store d in $s0

    jal func                    # Call func(a, b)

    move $a0, $v0               # Move return value of func(a, b) to $a0
    add $a1, $s0, $s1           # Compute c + d and store in $a1

    jal func                    # Call func(func(a, b), c + d)

    lw $ra, 8($sp)              # Restore return address
    lw $s1, 4($sp)              # Restore $s1
    lw $s0, 0($sp)              # Restore $s0
    addi $sp, $sp, 12           # Deallocate stack space
    jr $ra                      # Return
```

Register Group	Purpose
`$a0 - $a3`	Function arguments
`$v0, $v1`	Return values
`$t0 - $t9`	Temporary values (not preserved)
`$s0 - $s7`	Saved values (must be preserved)
`$sp, $fp`	Stack and frame pointer
`$ra`	Return address

4 Right before your function f from problem 3 returns, what do we know about contents of registers `$s3`, `$ra`, and `$sp`? Keep in mind that we know what the entire function `f` looks like, but for function `func` we only know its declaration.

```
Register $ra is equal to the return address in the caller
function, registers $sp and $s3 have the same values they had
when function f was called.
```