COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
14:332:331
Rutgers University
Fall 2016

Homework 2 Solution

1. Consider the following MIPS loop:

```
LOOP: slt $t2, $0, $t1
        beq $t2, $0, DONE
        subi $t1, $t1, 1
        addi $s2, $s2, 2
j LOOP
DONE:
```

a) Assume that the register $t1 is initialized to the value 10. What is the value in register $s2 assuming $s2 is initially zero?
b) For each of the loop above, write the equivalent C code routine. Assume that the registers $s1, $s2, $t1, and $t2 are integers A, B, i, and temp, respectively
c) For the loop written in MIPS assembly above, assume that the register $t1 is initialized to the value N. How many MIPS instructions are executed?

Solution:

a) $t2 is set to 1 when $t1 will be equal to 0. In each iteration, $t1 is subtracted by 1, so there will be 10 iterations. In every iteration, $s2 is incremented by 2, thus $s2 will be 20 at the end of the loop.
b)

```
while(i>0){
     i--;
    B = B + 2;
}
```

c) Each iteration contains 5 instructions. N iterations have 5*N instructions and the first two instructions need to be executed to jump out of the loop. So the answer is 5*N+2.

2.

a) Suppose that the current value of PC is 0x00004000. Can we use a single jump instruction to go to PC=0x20014924 ?(if yes, write the jump instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)

Solution JUMP address is found by concatenating 4 MSB bits of PC+4, 26 bit immediate value in JUMP, and "00". So from PC=0x00004000, the farthest address that we can jump to is: 0000 1111 1111 1111 1111 1111 1111 1100 which is 0x0FFFFFFC. So we cannot directly jump to PC=0x20014924. We can use a combination of instructions to do this jump. The following is one of these combinations:

```
lui $t0, 0x2001
ori $t0, $t0, 0x4924
jr $t0
```

Another option can be as follows:

PC: 0x00004000 J 0b1111 1111 1111 1111 1111 1111 11 # new pc=0x0FFFFFFC
PC: 0x0FFFFFFC J 0b1111 1111 1111 1111 1111 1111 11# new pc=0x1FFFFFFC
PC: 0x1FFFFFFC J 0b0000 0000 0001 0100 1001 0010 01# new pc=0x20014924

b) Suppose that the current value of PC is 0x00004000. Can we use a single branch instruction to go to PC=0x20014924? (if yes, write the branch instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)

Solution

Branch address is found by PC+4+A where A is found by sign extending the branch address (which is 16 bits 2's compliment number) and make a 30 bit value and then adding "00" as the LSB bits to make 32 bit value. So the farthest address we can go forward by a branch is PC_new= PC(of branch instruction)+4+ "0000 0000 0000 0001 1111 1111 1111 1100"=PC+4+0x0001FFFC , which is 0x00024000. So we cannot go directly from PC=0x00004000 to PC=0x20014924. We should use a combination of instructions to perform this branch. One such combination is as below:

PC: 0x00004000 beq $0, $0, 0b0111 1111 1111 1111 # new pc=0x00024000
PC: 0x00024000 J 0b1111 1111 1111 1111 1111 1111 11# new pc=0x0FFFFFFC
PC: 0x0FFFFFFC J 0b1111 1111 1111 1111 1111 1111 11# new pc=0x1FFFFFFC
PC: 0x1FFFFFFC J 0b0000 0000 0001 0100 1001 0010 01# new pc=0x20014924

c) Suppose that the current value of PC is 0x1FFFF000. Can we use a single branch instruction to go to PC=0x20014924? (if yes, write the branch instruction and show the value of the immediate field in Hex. If not, use a combinations of instructions to do so and show the immediate values in Hex)

Solution With a single branch, the farthest place we can go is the PC+4+0x0001FFFC, which is 0x2001F000. The target PC (0x20014924) is in this range. The immediate value in HEX is 0x5648. Thus,

beq $0,$0,0x5648

3. Compile the assembly code for the following C code.

```
int func (int a, int b, int c){
if (a<=c)
   return 4;
else if (a<b)
   return 8
else
   return a+c
}
```

Solution Caller: jal func //$a0 holds a, $a1 holds b and $a3 holds c.

```
func:       ble   $a0, $a3, Label1 // branch less than or equal
            blt   $a0, $a1, Label2
            j Label3
Label1:     addi  $v0, $zero, 4
            jr $ra
Label2:     addi  $v0, $zero, 8
            jr $ra
Label3:     add $v0, $a0, $a3
            jr $ra
```

4. Compile the assembly code for the following C code.

```
int f1 (int m, int n){
        return f2(4*n+m);
}
```

Solution Caller: jal f1 //$a0 holds m, $a1 holds n.

```
f1:         addi $sp, $sp,-4
            sw $ra, 0($sp)
            sll $t0,$a1,2 // 4*n
            add $t0,$t0,$a0 //4*n+m

            )
            or   $v0, $zero, $t0
            jal f2
            lw $ra, 0($sp)
            addi $sp, $sp, 4
            jr $ra

f2:         …
```

5. Compile the assembly code for the following C code.

```
int f3 (int n){
   if (n>20)
      return 0;
   else if (n<=1)
       return 1;
   else return (4*f3(n-2)+2)
}
```

Solution Caller: jal f3 //$a0 holds n

```
f3:         ori $t0,$zero,21
            slt $t1, $a0, $t0    // Reset $t1 if n>=21 which is equivalent to n>20
            ori $t0,$zero,1
            slt  $t2,$t0, $a0      // Reset $t2 if n<=1
            beq $t1,$zero, Label1
            beq $t2,$zero, Label2
else:       addi $sp, $sp, -8 //Save in the stack the $ra and $a0
            sw $ra, 0($sp)
            sw $a0, 4($sp)
```

```
            addi $a0,$a0,-2 // reduce input by 2
            jal f3          // call f3 again
            sll $v0,$v0,2   // $v0 has f3(n-2). Create 4*f3(n-2)
            addi $v0,$v0,2 // Add 2. $v0 = 4*f3(n-2)+2
            lw $ra, 0($sp)  //pop $ra from the stack
            lw $a0, 4($sp)  //pop $a0 from the stack
            addi $sp, $sp, 8 //reduce the stack size
            jr $ra          //return
Label1:     or $v0, $zero, $zero
            jr $ra
Label2:     ori $v0, $zero, 1
            jr $ra
```