CIS341 Homework 3.

SUID: 967159332 Student Name: Davon Grant

1. (2.24) Suppose the program counter (PC) is set to 0x2000 0000. Is it possible to use the jump (j) instruction to set the PC to the address as 0x4000 0000? Is it possible to use the branch-on-equal (beq) instruction to set the PC to this same address? (10 points)

It is not possible because the address is greater than 28 bits. It is not possible to branch because it is not an 18 bit address

2. (2.27) Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers $s0, $s1, $t0, and $t1 respectively. Also assume that register $s2 holds the base address of the array D. (20 points)

```
for(i = 0; i < a; i++)
    for(j = 0; j < b; j++)
        D[4*j] = i + j;
```

```
        add  $t0, $0, $0
L1      slt  $t2, $t0, $s0
        beq  $t2, $0, Exit
        add  $t1, $0, $0
L2      slt  $t2, $t1, $s1
        beq  $t2, $0, L3
        add  $t2, $t0, $t1
        sll  $t4, $t1, 4
        add  $t3, $t4, $s2
        sw   $t2, 0($t3)
        addi $t1, $t1, 1
        L2
L3      addi $t0, $t0, 1
```

3. (2.28) How many MIPS instructions does it take to implement the C code from the problem 2? If the variables a and b are initialized to 10 and 1, and all elements of D are initially 0, what is the total number of MIPS instructions that is executed to complete the loop? (10 pts)

L1

Inner loop: $3 + 4b + 4b$

Outer loop: $3 + 4a + 4(\text{Inner})$

$3 + 10(4 + (3 + 8))$

153

4. (2.31) Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function? (30 points)

```
int fib( int n ) {
    if( n == 0 )
        return 0;
    else if( n == 1 )
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

```
fib: addi $sp, $sp, -12
     sw   $a0, 8($sp)
     sw   $ra, 4($sp)
     sw   $s0, 0($sp)
     slti $t0, $a0, 2
     beq  $t0, $0, else
     add  $v0, $a0, $0
     j    exit
else: addi $a0, $a0, -1
     jal  fib
     add  $s0, $v0, $0
     addi $a0, $a0, -1
     jal  fib
     add  $v0, $v0, $s0
exit: lw  $a0, 8($sp)
     lw   $ra, 4($sp)
     lw   $s0, 0($sp)
     addi $sp, $sp, 12
     jr   $ra
```

5. (2.33) For each function call (Problem 4), show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7ffffffc, and follow the register conventions as specified in Figure 2.11. (10 pts)

0x7ffffff0 0x7fffffa8 0x7fffff90 0x7ffffff0
0x7ffffffe4 0x7fffffac 0x7fffffa8 0x7ffffffc
0x7fffffd8 0x7fffffa0 0x7fffffc0
0x7ffffffcc 0x7fffffa8 0x7fffffbc
0x7ffffffc0 0x7fffffa8 0x7fffffc0
0x7ffffffb4 0x7fffffac 0x7ffffffb4

6. (2.39) Write the MIPS assembly code that creates the 32-bit constant 0010 0000 0000 0001 0100 1001 0010 0100(two) and stores that value to register $t1. (10 pts)

```
lui  $t0, 0x2001
ori  $t0, 0x4924
add  $t1, $t0, $0
```

7. (2.46) Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instruction is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdown: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions. (10 pts / 5 pts each)

A. Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

It is not a good design
Choice because it increases execution
time

B. Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?