cs61c, Summer 1999
Midterm #1
Professor Clancy

Problem #1 (4 points) [10 minutes]

Let us define a new MIPS instruction similar to the SBN (Subtract and Branch if Negative) instruction from your first homework as follows:

```
        SBN   reg1,reg2,reg3,label

Where 'reg1', 'reg2', and 'reg3' are registers and 'label' is an address.
Its function is identical to that of the following pseudocode:

        reg1 = reg2 - reg3
        if (reg1 < 0) branch to label
```

You have learned of three different instruction formats used by the MIPS processor: R-, I- and J- format. Since this new instruction doesn't fit well into any of these three, let us define a new format, the Y-Format, as follows to use with this instruction:

reg2	reg3	reg1	Opcode	Label
4 bits	4 bits	4 bits	6 bits	16 bits

List three completely distinct reasons why the Y-Format instruction, as defined, would not work correctly on the MIPS processor we've been studying.

1.

2.

3.

Problem #2 (8 points) [20 minutes]

Each part below specifies an instruction that could be defined to be a pseudoinstruction for a particular MIPS simulator. The effect of each instruction is described so there will be no confusion. For each, specify whether it can be translated into a single TAL instruction by circling YES (it can be translated into a single TAL instruction) or NO (it can not be translated into a single TAL instruction). Also for each write out the shortest sequence of TAL instructions that could be used to simulate the given instruction. (Note: This sequence should be only one instruction for all 'YES' answers and more than one instruction for all 'NO' answers.)

```
a.      move    reg1, reg2
effect: The value in 'reg2' is copied into 'reg1'.

Yes    No
```

```
b.      sbn    reg1,reg2,label
```

Effect: 'reg1' gets the value (reg1-reg2). If this value is negative, branch to the given 'label'.

```
Yes    No
```

```
c.      j    AnotherFunc
```

Note: The address of this jump instruction is 0x4400000C. The address of the label 'AnotherFunc' is 0x7E0080A4. Effect: The normal effects of a jump instruction.

```
Yes    No
```

```
d.      neg    reg1,reg2
Effect: 'reg1' gets set to the negative of the value in 'reg2'.
```

```
Yes    No
```

```
e.      pow2    reg1,reg2,immed
Note: 'immed' represents an immediate. Assume 0 <== immed <= 31.
```

Effect 'reg1' gets the value (reg2 * 2^immed). That is, 'reg1' gets the value of 'reg2' multiplied by 2 to the power of the immediate.

```
Yes    No
```

```
f.      slti    reg1,reg2,0xFAFFFF
Effect: The normal effects of an slti instruction.
```

```
Yes    No
```

```
g.      clear    reg1
Effect: 'reg1' gets set to 0
```

```
Yes    No
```

Problem #3 (6 points) [15 minutes]

Follow the instructions below CONSECUTIVELY, and after each step write your value in HEXADECIMAL into the space provided. (Don't bother preceding it with "0x".) Also, this will be a lot easier if you pay attention to the hints along the way. In case you do any work in this space, please be sure to clearly mark which is your final answer for each step.

Step 1: Pick any 32-bit number containing 19 ones and 13 zeros. (Hint: It should contain eight hex digits, no more, no less.)

Step 2: Multiply it by 256. (Hint: Do not turn your number into the decimal equivalent. You can do this while the number is still in hex form. If necessary though, feel free to convert to binary.)

Step 3: Add the decimal value 10 to the number from step 2. (Hint: Again, you can easily do this while the number is in hex form.)

Step 4: AND the value from step 3 with the hex value 0xFC00 0FFF.

Step 5: Take the one's complement of the value from step 4. (Hint: This means invert all the bits.)

Step 6: OR the value from step 5 with the hex value 0x FF00FF00.

Step 7: take the one's complement of the value from step 6. (Hint: See hint in step 5.)

Step 8: Right shift by 2. (Hint: If you have to ask whether you should shift right logical (srl) or shift right arithmetic (sra), then you've done something wrong, so redo the previous steps.)

Step 9: Assuming the value from step 8 is in register $t0, what will register $t0 contain after execution of the instruction:

```
        lui     $t0, 0xFADE
```

Final question: Regardless of the chosen value for step 1, the value after step 9 is fixed. What is the earliest step for which this is true? That is, what is the earliest step after which no semblance remains of the value chosen in step 1?

Problem #4 (6 points) [15 minutes]

Assume 'array' is a ten element array of ints whose address is stored in $t0. Also assume variable x is $t1 and y is $t2. For each part, pick the MIPS assembly code which has the same functionality as the given C code.

```
a.      x=array[3];

A - lw  $t1, 3($t0)

B - lw  $t0, 3($t1)

C - lw  $t1, 12($t0)

D - lw  $t1, 12($t0)

E - lb  $t1, 12($t0)


b.      x=*x;
```

```
A - lw  $t0, 0($t1)

B - add $t1, $t1, $0

C - lw  $t1, 0($t1)

D - mult $t1, $t1
    mflo $t1

E - lw  $t1, 0($t0)
```

c. x=array[x];

```
A -  sll $t1, 3($t0)
     add $t1, $t0, $t1
     lw  $t1, 0($t1)

B -  lw $t1, $t1($t0)

C - sll $t1, 3($t0)
    add $t1, $t0, $t1
    lb  $t1, 0($t1)

D - sll $t1, 3($t0)
    add $t1, $t0, $t1
    lw  $t1, 0($t0)

E - none of the above
```

d. x=y%128;

```
A - addi   $1, $0, 128
    div    $t1, $1

B - addiu  $1, $0, 128
    div    $t1, $1

C - xor $t1, $t2, 0x7f

D - ori $t1, $t2, 0x7f

E - andi $t1, $t2, 0x7f
```

e. x*=x;

```
A - lw  $t0, 0($t1)

B - add $t1, $t1, $0
```

```
C - lw  $t1, 0($t1)

D - mult  $t1, $t1
    mflo  $t1

E - lw  $t1, 0($t0)


f.      x=y*128

A - mult $t1, $t2, 128

B - sll $t1, $t2, 128

C - sll $t1, $t2, 0x7f

D - andi $t1, $t2, 0x7f

E - sll $t1, $t2, 7
```

Problem #5 (12 points) [30 minutes]

The following pattern of numbers is called Pascal's triangle.

ROW \ COLUMN	1	2	3	4	5
1	1				
2	1	1			
3	1	2	1		
4	1	3	3	1	
5	1	4	6	4	1

The numbers along the two edges of the triangle are all 1 (one), and each number in the interior of the triangle is the sum of the number above it and the number above and to the left of it.

Write a RECURSIVE MIPS procedure that calculates a specific element of Pascal's triangle.

Your procedure should be named "pascal" and should take two arguments: the first being the (integer) row number of the element to calculate and the second being the (integer) column number. It should return the calculated element also an integer, as the return value. You must follow register conventions as well as standard procedure calling conventions for full credit on this question (in other words, make no assumptions about the calling procedure). You may assume the inputs are each greater than or equal to 1.

Here is a C function which accomplishes the desired task. You are advised to stay as close to this function as possible in order to simplify your own life (and ours).

```
int pascal (int row, int column) {
        if (column == 1) return 1;
        else if (row == column) return 1;
        else {
```

```
                return (pascal (row-1, column) + pascal (row-1, column-1));
        }
}
```

Important:
1. Solutions that are not recursive will not get any credit.
2. You must stick to all register conventions and procedure calling conventions
3. No pseudoinstructions. TAL only
4. You must write comments. Code that is not adequately commented will be penalized.

Write your procedure on the next page.
