NAME:_____

CS3411 Test #1, 6 September 2007. You have 1h15min (one class period). Answer all questions. The number of points (out of 100) each problem is worth is shown in (parentheses). Be as concise as you can while still answering the question.

1. (2) What is a transistor?

   An electrical on/off switch
2. (2) What do we call the 32-bit value that we store for nested procedures that points to the instruction after the **jal** instruction that called a procedure, so that we can go back to the calling procedure?

   return address
3. (2) What do we call the register where the value from problem #2 is stored when we execute a **jal**? (Write it as it appears in MIPS code).

   $ra
4. (2) Where do we store that register (from problem #3) when we have nested procedures so that we can restore it before returning with a **jr** instruction (hint: it's not the heap)?

   the stack
5. (2) What register keeps track of the control flow and is used to calculate the 32-bit value for problem #2? (Write what we call it, not MIPS code).

   program counter

6. (10) Write what type of encoding ("?-Type") is used to encode each of the following MIPS instructions:

add $t0 $t1 $t2	**R**
j SomeLabel	**J**
jal SomeLabel	**J**
addi $t0 $t0 1	**I**
slti $t8 $t7 'c'	**I**
sw $t0 8($t1)	**I**
sll $t6 $t8 5	**R**
bneq $t5 $t7 SomeLabel	**I**
ori $t6 $t6 4	**I**
addu $t0 $t1 $t2	**R**

7. (5) If I put together a single half adder and 31 full adders to create a 32-bit adder, what would I call this? Why do circuit designers often use more complex adders than this?

   Ripple Carrry Adder, the circuit delay is too long.

8. (10) For each of these addressing modes, give an example MIPS instruction that uses that type of addressing, and for those that have limitations on what addresses in a 32-bit address space they can address, state precisely what those limitations are.

   Register addressing

   add $t0 $t1 $2

   Base or displacement addressing

   lw $t0 0($t1)

   Immediate addressing

   addi $t0 $t1 55

   PC-relative addressing

   bne $t0 $t1 SomeLabel – can only branch about 2^15 words or 2^17 bytes in either direction.

   Pseudo-direct addressing

   jal SomeProcedure – limited to a 256 MB block of memory

9. (10) Write down, for each transformation, what compiler optimization has occurred (examples are in MIPS assembly or C pseudocode of MIPS assembly):

Before optimization	After optimization	What?

for (i=0;i<4;i++) {
x[i] = y[4- i]
}
	for (i=0;i<4;i+=2) {
	x[i] = y[4 – i];
	x[i + 1] = y[3 – i];
	}
		Loop Unrolling

addi $s3 $zero 32 multu $s2 $s3 mflo $s1	sll $s1 $s2 5	*Strength Reduction*
i = 7 * 8 * 100;	i = 5600;	*Constant Folding (or Constant Propagation)*

for (i=0;i<100;i++) {
z = w * 4;
x[i] = y[i] * z;
}
	z = w * 4;
	for (i=0;i<100;i++) {
	x[i] = y[i] * z;
	}
		Code Motion

add $t1 $t2 $t3 beq $zero $zero LL: sub $t5 $t6 $t7 LL: sll $t1 $t1 4	add $t1 $t2 $t3 sll $t1 $t1 4	*Dead Code Elimination*

10. (10) What is going to happen in each of these examples (underflow, overflow, something else)? How will we know (an exception, or do we need to check something in particular)? Assume that this is just pseudocode and we're actually writing these in MIPS and doing the checks afterward or handling thrown exceptions.

a. float f; int i;

```
for (i = 0; i < 1000; i++)
    f *= 0.5;
```

Assuming I had initialized f, e.g. f = 1.0, then underflow, we'll get f == 0.0, no exception.

b. unsigned int i = 0x80000000;

```
i = i + i;
```

Overflow, have to check both operands before the add. (No exception)

c. unsigned int i = 0x80000000;

```
i = i * 2;
```

Overflow, have to use mfhi and check for non-zero after the multiply. (No exception)

d. float f = 100.0; float g = 0.0; float h;

```
h = f / g;
```

Divide by zero, h == inf. (No exception)

e. float f = 0.0; float g = 0.0; float h;

```
h = f / g;
```

Divide by zero, h == NaN. (No exception)

11. (30) Annotate the following MIPS code with comments about what is going on. Then write a single high-level sentence in English explaining what the code does. Then write what the output of the program is.

```
.data
        s1:     .asciiz "Hello123."
        s2:     .asciiz "Bla876blabla!"
        s3:     .asciiz "55xyz66op88 333:/#7"
        s:      .word 0
                .word 0
                .word 0
                .word 0
        Output: .space 100

.text
.align 2
main:
        la      $t0 s1
        la      $t1 s
        sw      $t0 0($t1)
        la      $t0 s2
        sw      $t0 4($t1)
        la      $t0 s3
        sw      $t0 8($t1)

        la      $t3 Output

        addi    $t1 $zero 0

Outer:
        sll     $t7 $t1 2
        la      $t8 s
        add     $t7 $t7 $t8
        lw      $t0 0($t7)


Inner:
        lb      $t8 0($t0)
```

```
        slti    $t5 $t8 '0'
        slti    $t6 $t8 ':'     # ':' comes after '9'
        sub     $t6 $t6 $t5
        beq     $t6 $zero There
        sb      $t8 0($t3)
        addi    $t3 $t3 1

There:
        addi    $t0 $t0 1
        bne     $t8 $zero Inner

        addi    $t1 $t1 1
        sll     $t7 $t1 2
        la      $t8 s
        add     $t7 $t7 $t8
        lw      $t8 0($t7)
        bne     $t8 $zero Outer

        addi    $t8 $zero 10    # 10 == new line
        sb      $t8 0($t3)
        sb      $zero 1($t3)

        la      $a0 Output
        li      $v0 4
        syscall                 # write_string(Output)
        jr      $ra
```

10 points (number on top) if you basically knew what a pointer was and demonstrated that by annotating la, etc. 10 points (middle number) if you knew basic ops like add and sll, 10 points if you understood at a high level what was going on, even if you didn't get the output right.

12. (10) Count the total number of basic blocks in the program from problem #11. How many basic blocks are there?

Eight. Remember that we're considering syscall to be the end of a basic block.

13. (5) In problem #11, the assembler is going to put instructions in memory in not quite the same order as the assembly code has them. Why?

Delayed branch

(Your answer for #13 should end about here, no need to write an essay because the rest of the page is blank. In fact, if you know the two-word name for it, just write those two words.)