CS232 Exam 1
February 25, 2004

Name: ______________________________

- This exam has 5 pages, including this cover.
- There are three questions, worth a total of 100 points.
- The last page is a summary of the MIPS instruction set, which you may remove for your convenience.
- No other written references or calculators are allowed.
- Write clearly and show your work. State any assumptions you make.
- You have 50 minutes. Budget your time, and good luck!

Question	Maximum	Your Score
1	40	
2	45	
3	15	
Total	100	

Question 1: Understanding MIPS programs (40 points)

```
nvidia:      li      $t0,     0
             li      $t1,     0
             li      $t2,     0x3fffffff
ibm:         bge     $t0,     $a1,     sun
             mul     $t3,     $t0,     4
             add     $t3,     $a0,     $t3
             lw      $t3,     0($t3)
             ble     $t3,     $t1,     hp
             move    $t1,     $t3
hp:          bge     $t3,     $t2,     intel
             move    $t2,     $t3
intel:       addi    $t0,     $t0,     1
             j       ibm
sun:         sw      $t1,     0($a2)
             sw      $t2,     0($a3)
             jr      $ra
```

Part (a)

Translate the function `nvidia` above into a high-level language like C or Java. Your function header should list the types of any arguments and return values. Also, your code should be as concise as possible, without any gotos or pointer arithmetic. We will not deduct points for syntax errors unless they are significant enough to alter the meaning of your code. (30 points)

Part (b)

Describe briefly, in English, what this function does. (10 points)

Question 2: Write a recursive MIPS function (45 points)

In class, we demonstrated a `bit_count` function which counted the number of set bits (*e.g.,* 1's) in the two's complement representation of a 32-bit integer. Here is a recursive implementation of the same function that takes 1 argument (the 32-bit integer) and has one return value (the number of set bits).

```
int
recursive_bit_count(int input_word) {
  if (input_word == 0)
    return 0;
  int lowest_bit = input_word & 1;
  return lowest_bit + recursive_bit_count(input_word >> 1);
}
```

Translate `recursive_bit_count` into a **recursive** MIPS assembly language function; iterative versions (*i.e.,* those with loops) will not receive full credit. You will not be graded on the efficiency of your code, but you must follow all MIPS conventions. Comment your code!!!

Question 3: Concepts (15 points)

Write a short answer to the following questions. For full credit, answers should not be longer than **two sentences**.

Part a) What is abstraction? How does it relate to instruction set architectures (ISAs)? (5 points)

Part b) Explain how a machine knows whether to interpret a group of bits as an integer, a floating point number, or an instruction. (5 points)

Part c) Order the following (single precision) floating point numbers in increasing value from **1** to **4**, where **1** is the smallest number (*i.e.,* the most negative) and **4** is the largest number (*i.e.,* the most positive). (5 points)

______	0	01010011	11010011101010010101001
______	0	01010011	00101100010101101010110
______	1	11010011	00101100010101101010110
______	0	11010011	11010011101010010101001

s	e	f

$$(1 - 2s) * (1 + f) * 2^{e-bias}$$

MIPS instructions

These are some of the most common MIPS instructions and pseudo-instructions, and should be all you need. However, you are free to use *any* valid MIPS instructions or pseudo-instruction in your programs.

Category	Example Instruction	Meaning
Arithmetic	add $t0, $t1, $t2 sub $t0, $t1, $t2 addi $t0, $t1, 100 mul $t0, $t1, $t2 div $t0, $t1, $t2	$t0 = $t1 + $t2 $t0 = $t1 – $t2 $t0 = $t1 + 100 $t0 = $t1 x $t2 $t0 = $t1 / $t2
Logical	and $t0, $t1, $t2 or $t0, $t1, $t2 sll $t0, $t1, $t2 srl $t0, $t1, $t2	$t0 = $t1 & $t2 (Logical AND) $t0 = $t1 | $t2 (Logical OR) $t0 = $t1 << $t2 (Shift Left Logical) $t0 = $t1 >> $t2 (Shift Right Logical)
Register Setting	move $t0, $t1 li $t0, 100	$t0 = $t1 $t0 = 100
Data Transfer	lw $t0, 100($t1) lb $t0, 100($t1) sw $t0, 100($t1) sb $t0, 100($t1)	$t0 = Mem[100 + $t1] 4 bytes $t0 = Mem[100 + $t1] 1 byte Mem[100 + $t1] = $t0 4 bytes Mem[100 + $t1] = $t0 1 byte

Branch	beq $t0, $t1, Label
	bne $t0, $t1, Label
	bge $t0, $t1, Label
	bgt $t0, $t1, Label
	ble $t0, $t1, Label
	blt $t0, $t1, Label
		if ($t0 = $t1) go to Label
		if ($t0 ≠ $t1) go to Label
		if ($t0 ≥ $t1) go to Label
		if ($t0 > $t1) go to Label
		if ($t0 ≤ $t1) go to Label
		if ($t0 < $t1) go to Label

Set	slt $t0, $t1, $t2
	slti $t0, $t1, 100
		if ($t1 < $t2) then $t0 = 1 else $t0 = 0
		if ($t1 < 100) then $t0 = 1 else $t0 = 0

Jump	j Label jr $ra jal Label	go to Label go to address in $ra $ra = PC + 4; go to Label

The second source operand of the arithmetic, logical, and branch instructions may be a constant.

Register Conventions
The *caller* is responsible for saving any of the following registers that it needs, before invoking a function.

$t0-$t9 $a0-$a3 $v0-$v1

The *callee* is responsible for saving and restoring any of the following registers that it uses.

$s0-$s7 $ra

Pointers in C:
Declarartion: either *char *char_ptr* -or- *char char_array[]* for *char c*
Dereference: *c = c_array[i]* -or- *c =*c_pointer*
Take address of: *c_pointer = &c*