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)

```
void nvidia (int array[], int len, int* p_max, int* p_min) {
  int max = 0;                // $t1
  int min = 0x3fffffff;       // $t2
  for (int i = 0; i < len ; ++ i) {     // i = $t0
      int temp = array[i];    // $t3
      if (temp > max)
        max = temp;
      if (temp < min)
        min = temp;
  }
  (*p_max) = max;
  (*p_min) = min;
}
```

The key things to realize are the following:

- $a1 is compared to $t0, suggesting it's an integer.

- $t0 is then used to calculate the offset of $a0, which suggests a0 is an array

- $t3 is loaded from $a0, and is compared to $t2, which suggests it's an integer

- $t1 and $t2 are stored to addresses of $a2 and $a3; since $a2 and $a3 are used directly as addresses, they must be pointers.

- The function does not put anything into $v0, thus should be ***void****.*

Part (b)

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

The function traverses an array of length $a1 finding the maximum and minimum elements and returning them through two pointers.

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!!!

```
recursive_bit_count:
          beq       $a0, $0, done       # if (input == 0)

          sub       $sp, $sp, 8         #save regs
          and       $t0, $a0, 1         # else and t0 = input&&1
          sw        $t0, 0($sp)
          sw        $ra, 4($sp)

          srl       $a0, $a0, 1         # a0 = a0 >> 1
          jal       recursive_bit_count          # the recursive call
          lw        $t0, 0($sp)         # restore caller saved reg
          add       $v0, $v0, $t0       # compute the return val

          lw        $ra, 4($sp)         # load and restore sp
          add       $sp, $sp, 8
          jr        $ra

done:
          li        $v0, 0              # base case
          jr        $ra
```

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)

Abstraction separates interface from implementation.

ISAs are an example of abstraction because they describe the interface to hardware, without details of the implementation. This idea enables binary compatibility across generations of hardware.

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)

*As the bits in memory are not tagged with any type information the manner that bits are interpreted is determined by the instruction with which they are accessed. Bits are interpreted as integers when loaded with an integer load (*e.g., *lw), as a floating point number if loaded with a a floating point load (*e.g., *l.s) and as instruction if jumped to using a control flow instruction (*e.g., *j or bne).*

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)

__ 3___ 0 01010011 11010011101010010101001 # positive, negative exponent, higher mantissa
__ 2___ 0 01010011 00101100010101101010110 # positive, negative exponent, lower mantissa
__ 1___ 1 11010011 00101100010101101010110 # negative (must be lowest)
__ 4___ 0 11010011 11010011101010010101001 # positive, positive exponent (must be highest)

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*