CS232 Exam 1
February 14, 2007

Name: _____

Section: 10am noon 2pm 4pm (circle one)

- This exam has 6 pages, including this cover.
- There are three questions, worth a total of 100 points.
- The last two pages are a summary of the MIPS instruction set, calling convention, and hexadecimal notation, 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	20	
3	40	
Total	100	

Question 1: Write a recursive MIPS function (40 points)

Here is a recursive function that multiplies two unsigned integers and returns the result as an unsigned integer. Translate `multiply` 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!!!

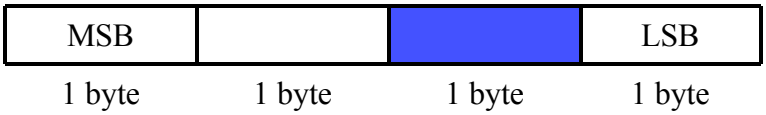
```
unsigned int
multiply(unsigned int x, unsigned int y) {
  if (y == 1) {
     return x;
  }

  unsigned temp = mul(x<<1, y>>1);
  if ((y & 1) == 1) {
     temp = temp + x;
  }
  return temp;
}
```

```
multiply:
    bne    $a1, 1, not_base_case   # check for the base case
    mov    $v0, $a0                # return x
    jr     $ra

not_base_case:
    sub    $sp, $sp, 12            # save x and y before shifting them
    sw     $ra, 0($sp)
    sw     $a0, 4($sp)
    sw     $a1, 8($sp)

    sll    $a0, $a0, 1             # x << 1
    srl    $a1, $a1, 1             # y >> 1
    jal    multiply                # temp = multiply(...)

    lw     $a1, 8($sp)             # restore original y
    and    $a1, $a1, 1             # y & 1
    bne    $a1, 1, skip_if         # skip if (y & 1) != 1

    lw     $a0, 4($sp)             # restore original x
    add    $v0, $v0, $a0           # temp += x

skip_if:
    lw     $ra, 0($sp)             # restore $ra
    add    $sp, $sp, 12            # restore stack (same amount)
    jr     $ra                     # return temp
```

Question 2: Concepts (20 points)

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

Part a) Write the body of a function that returns the second least significant byte (shown shaded in the picture) of a 32-bit integer using bit-wise logical operations and shifts. No control flow (e.g., loops/ifs) are needed. (10 points)




```
char
get_2nd_byte(int x) {

    return (x >> 8) & 0xff;


}
```

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

An abstraction layer is a way of hiding the implementation details of a particular set of functionality by providing an interface that separates the functionality from how it is implemented. An instruction set architecture (ISA) is the interface that abstracts the processor's funcationality from how it is implemented.

Part c) What is the "assembler temporary"? What is the motivation for it? (5 points)

The assembler temporary ($at) is a register that, by convention, is not used by assembly programmers or compilers, so that it can be used as a temporary register for implementing pseudo-instructions that are implemented as multiple real assembly instructions. For example, blt $a0, $a1, Label is translated into the following:

```
slt     $at, $a0, $a1       // $at = 1 if $a0 < $a1
bne     $at, $0, Label      // Branch if $at != 0
```

Question 3: Understanding MIPS programs (40 points)

```
knight:     sub     $t0, $a1, 1
pawn:       blt     $t0, $0, rook
            mul     $t1, $t0, 4
            add     $t1, $t1, $a0
            lw      $v0, 0($t1)
            blt     $v0, $a2, bishop
            sub     $t0, $t0, 1
            j       pawn
rook:       li      $v0, 0
bishop:     jr      $ra
```

Part (a)

Translate the function `knight` 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)

```
int
find_backwards(int arr[], int len, int min) {
   for (int i = len - 1 ; i >= 0 ; i = i - 1) {
       if (arr[i] < min) {
          return arr[i];
       }
   }
   return 0;
}
```

Part (b)

Describe briefly, in English, what this function does. *(Note: It has nothing to do with chess.)* (10 points)

The function walks backwards through an array of integers (potentially starting at the end), looking for an integer below the 3rd argument. The first one found is returned; if one is not found, the value 0 is returned.

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
		$t0 = $t1 + $t2
		$t0 = $t1 – $t2
		$t0 = $t1 + 100
		$t0 = $t1 x $t2

Logical	and $t0, $t1, $t2 or $t0, $t1, $t2 sll $t0, $t1, $t2 srl $t0, $t1, $t2 sra $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) $t0 = $t1 >> $t2 (Shift Right Arithmetic)
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*

Hexadecimal Notation

C and languages with a similar syntax (such as C++, C# and Java) prefix hexadecimal numerals with "0x", e.g. "0x5A3". The leading "0" is used so that the parser can simply recognize a number, and the "x" stands for hexadecimal.

Hex	Bin	Dec
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15