CS232 Midterm Exam 1
February 18, 2002

Name: Michael Corleone

- This exam has 6 pages, including this cover and the cheat sheet on the next page.
- There are three questions worth a total of 100 points.
- You have only 50 minutes, so budget your time!
- No written references or calculators are allowed.
- To make sure you receive full credit, please write clearly and show your work.
- We will not answer questions regarding course material.

Question	Maximum	Your Score
1	30	
2	40	
3	30	
Total	100	

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		Meaning
Arithmetic	add	$t0, $t1, $t2	$t0 = $t1 + $t2
	sub	$t0, $t1, $t2	$t0 = $t1 - $t2
	addi	$t0, $t1, 100	$t0 = $t1 + 100
	mul	$t0, $t1, $t2	$t0 = $t1 * $t2
	move	$t0, $t1	$t0 = $t1
	li	$t0, 100	$t0 = 100
Data Transfer	lw	$t0, 100($t1)	$t0 = Mem[100 + $t1]
	sw	$t0, 100($t1)	Mem[100 + $t1] = $t1
Branch	beq	$t0, $t1, Label	if ($t0 == $t1) go to Label
	bne	$t0, $t1, Label	if ($t0 != $t1) go to Label
	bge	$t0, $t1, Label	if ($t0 >= $t1) go to Label
	bgt	$t0, $t1, Label	if ($t0 > $t1) go to Label
	ble	$t0, $t1, Label	if ($t0 <= $t1) go to Label
	blt	$t0, $t1, Label	if ($t0 < $t1) go to Label
Set	slt	$t0, $t1, $t2	if ($t1 < $t2) then $t0 = 1; else $t0 = 0
	slti	$t0, $t1, 100	if ($t1 < 100) then $t0 = 1; else $t0 = 0
Jump	j	Label	go to Label
	jr	$ra	go to address in $ra
	jal	Label	$ra = PC + 4; go to Label

The second source operand of *sub*, *mul*, and all the 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

Performance

Formula for computing the CPU time of a program P running on a machine X:

$$\text{CPU time}_{X,P} = \text{Number of instructions executed}_P \times \text{CPI}_{X,P} \times \text{Clock cycle time}_X$$

CPI is the average number of clock cycles per instruction, or:

$$\text{CPI} = \text{Number of cycles needed} / \text{Number of instructions executed}$$

Question 1: Understanding MIPS programs (30 points)

```
Ostrich:
      bge    $a1, $a2, Duck
      mul    $t1, $a1, 4
      add    $t1, $a0, $t1
      mul    $t2, $a2, 4
      add    $t2, $a0, $t2
      lw     $t3, 0($t1)
      lw     $t4, 0($t2)
      sw     $t3, 0($t2)
      sw     $t4, 0($t1)
      addi   $a1, $a1, 1
      sub    $a2, $a2, 1
      j      Ostrich
Duck:
      jr     $ra
```

Part (a)
Translate *Ostrich* into a high-level language like C or Java. Be sure to describe the number and types of any arguments and return values. We will not deduct points for syntax errors *unless* they are significant enough to alter the meaning of your code. (25 points)

You won't know the exact types of the operands, but you can make some pretty good guesses. The lw and sw instructions indicate array elements that are one word long each, so we might have an array of integers or floating-point numbers, for instance. The addi and sub indicate that $a1 and $a2 are probably integers.

Given this, there are several possible solutions, ranging from very literal translations of the code above to more refined ones. Here are two sample answers.

```
void Ostrich(int a0[], int a1, int a2)
{
    int t3, t4;
    while (a1 < a2) {
        t3 = a0[a1];
        t4 = a0[a2];
        a0[a2] = t3;
        a0[a1] = t4;
        a1++;
        a2--;
    }
    return;
}
```

```
void Ostrich(int a[], int i, int j)
{
    int temp;
    for (; i < j; i++, j--) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
    return;
}
```

Part (b)
Describe briefly, in English, what this function does. (5 points)

It reverses the contents of an array, from index a1 to index a2 inclusively.

Question 2: Performance (40 points)

Let's study the performance of this function. Assume that we have a 500MHz processor with a clock cycle time of 2ns. The table below shows CPIs for some various classes of MIPS instructions. (You can assume that pseudoinstructions are also accounted for in this table.)

Instruction class	CPI
mul	5
branches and jumps	3
data transfers	3
all others	1

Part (a)

What is the CPU time of this function in nanoseconds, assuming that the loop is executed 100 times? In other words, the "bge" test fails 100 times and then succeeds on the 101st test. (10 points)

```
Ostrich:
      bge   $a1, $a2, Duck    3
      mul   $t1, $a1, 4       5
      add   $t1, $a0, $t1     1
      mul   $t2, $a2, 4       5
      add   $t2, $a0, $t2     1
      lw    $t3, 0($t1)       3
      lw    $t4, 0($t2)       3
      sw    $t3, 0($t2)       3
      sw    $t4, 0($t1)       3
      addi  $a1, $a1, 1       1
      sub   $a2, $a2, 1       1
      j     Ostrich           3
Duck:
      jr    $ra               3
```

The number of cycles needed for each instruction in the program is shown above. If you add these up, you should find that each loop iteration requires 32 cycles, so 100 iterations needs 3200 cycles. You should also add in time for the 101st execution of "bge" (3 cycles) and the return (3 cycles). So the entire function needs 3206 cycles, or 6412 ns.

Question 2 continued

Part (b)
Our code is somewhat slow, since expensive multiplication operations are executed repeatedly. Rewrite the function to eliminate both multiplications from the main body of the loop. (20 points)

```
Ostrich:
      bge    $a1, $a2, Duck
      mul    $t1, $a1, 4
      add    $t1, $a0, $t1
      mul    $t2, $a2, 4
      add    $t2, $a0, $t2
      lw     $t3, 0($t1)
      lw     $t4, 0($t2)
      sw     $t3, 0($t2)
      sw     $t4, 0($t1)
      addi   $a1, $a1, 1
      sub    $a2, $a2, 1
      j      Ostrich
Duck:
      jr     $ra
```

```
Ostrich:
      mul    $t1, $a1, 4         5
      add    $t1, $t1, $a0       1
      mul    $t2, $a2, 4         5
      add    $t2, $t2, $a0       1
Loop:
      bge    $t1, $t2, Duck      3
      lw     $t3, 0($t1)         3
      lw     $t4, 0($t2)         3
      sw     $t3, 0($t2)         3
      sw     $t4, 0($t1)         3
      addi   $t1, $t1, 4         1
      sub    $t2, $t2, 4         1
      j      Ostrich             3
Duck:
      jr     $ra                 3
```

The original intention of this question was to have you move the multiplications outside the loop body, and to add or subtract four from $a1 and $a2 within the loop body. However, the question didn't specifically say what to do, and we got a range of answers. The solution to the right and above was what we were hoping for. (It's annotated with cycle times for each of the instructions, for use in Part (c) below.)

Part (c)
What is the CPU time in nanoseconds for 100 loop iterations of your revised function? (10 points)

Instruction class	CPI
mul	5
branches and jumps	3
data transfers	3
all others	1

Your answer here will depend upon what you wrote for Part (b) above. In our solution, each loop iteration now requires just 20 cycles, and 100 iterations requires 2000 cycles. Adding in the arithmetic initializations and the final "bge" and "jr," the whole function works out to 2018 cycles and 4036 ns. That's a speedup of about 1.6 times over the original code!

Question 3: Writing a nested function (30 points)

Show how to translate the pseudocode below for *SelectionSort* into a MIPS assembly language function. This algorithm first finds the largest element in A[0]..A[n] and moves it to position n, then finds the largest element in A[0]..A[n-1] and puts that in position n-1, and so forth.

- You will not be graded on the efficiency of your code, but you *must* follow all MIPS conventions.
- Assume that you already have a MIPS function *MaxIndex*, which takes two arguments A and n, and returns the index of the largest element of A[0]..A[n]. The arguments and return values are passed in registers $a0, $a1 and $v0 respectively.
- Remember that integers are 32-bit, or 4-byte, MIPS quantities.

```
SelectionSort(A, n)
    for i = n downto 1
        m = MaxIndex(A, i)          // Find the largest element in A[0]..A[i]
        exchange A[i] and A[m]      // Move it to position i
```

There's lots of different ways to do this one. You should have handled all of the register saving and restoring properly; you can use the $s or $t registers or both, but you'll have to save them since this function is both a caller and a callee. If you use the $s registers, then you'll have to save them at the beginning of your function and restore them at the end. If you use $ts instead, you'll have to save them before the call to MaxIndex and restore them afterwards.

The solution on the next page uses $s0-$s3 as temporary registers. This means we only have to save and restore them once, at the beginning and end of the function. Since these are callee-saved registers, we can safely assume that MaxIndex will not alter them. We're also using $a0 throughout the function to show you how to handle caller-saved registers. Here it's possible that $a0 (or any other caller-saved registers) is modified by MaxIndex, so we need to save and restore its value before and after the jal instruction.

Question 3 continued

```
SelectionSort:
      sub     $sp, $sp, 24
      sw      $ra, 0($sp)
      sw      $s0, 4($sp)
      sw      $s1, 8($sp)
      sw      $s2, 12($sp)
      sw      $s3, 16($sp)
      sw      $a0, 20($sp)          # Caller-saved, but value never changes

      move    $s0, $a1              # $s0 = i
      mul     $s1, $s0, 4
      add     $s1, $a0, $s1         # Compute &A[i]

_SortLoop:
      blt     $s0, 1, _SortExit     # Stop when i < 1
      lw      $a0, 20($sp)
      move    $a1, $s0
      jal     MaxIndex              # Call MaxIndex

      lw      $a0, 20($sp)
      mul     $v0, $v0, 4
      add     $v0, $v0, $a0         # Find &A[m]
      lw      $s2, 0($v0)           # Swap A[m] and A[i]
      lw      $s3, 0($s1)
      sw      $s2, 0($s1)
      sw      $s3, 0($v0)

      sub     $s0, $s0, 1           # i = i - 1
      sub     $s1, $s1, 4           # Keep track of &A[i]
      j       _SortLoop

_SortExit:
      lw      $ra, 0($sp)
      lw      $s0, 4($sp)
      lw      $s1, 8($sp)
      lw      $s2, 12($sp)
      lw      $s3, 16($sp)
      addi    $sp, $sp, 24
      jr      $ra
```