CS232 Midterm Exam 1
October 1, 2003

Name:_____

Section:_____

- This exam has 7 pages, including this cover and the cheat sheet at the end.
- 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	10	
2	30	
3	30	
4	30	
Total	100	

Question 1: Concepts (10 points)

Write short answers 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 is the concept of separating interface from implementation. An ISA abstracts the specification of what computation should be performed from how it is performed, allowing binary code compatibility across a family of implementations that implement the same ISA.

Part (b) Motorola unveils a new processor design the M-232 that runs at 10GHz thanks to a sleek new ISA. What additional information do you need to know how much faster (or slower) the M-232 is than a 3GHz Pentium4 when running the Photoshop benchmark? (5 points)

The performance formula for CPU time has three components: number of instructions executed, number of cycles per instruction (CPI) and clock cycle time. We're given the clock cycle time (the inverse of number of instructions per second), so to calculate how well a given program will execute, we need to know CPI for the Photoshop benchmark program and the number of instructions required to run Photoshop benchmark on both machines.

Do not write in the shaded area.

Question 2: Understanding MIPS Programs (30 points)

```
origami:
          li      $t0, 0
          li      $v0, 0
pear:     mul     $t1, $t0, 4
          add     $t1, $a0, $t1
          lw      $t1, 0($t1)
          blt     $t1, $0, pizza
          mul     $t2, $v0, 4
          add     $t2, $a1, $t2
          sw      $t1, 0($t2)
          add     $v0, $v0, 1
pizza:    add     $t0, $t0, 1
          blt     $t0, $a2, pear
          jr      $ra
```

Part (a)
Translate the function `origami` above into a high-level language like C or Java. You should include a header that lists the types of any arguments and return values. Also, your code should be as concise as possible, without any gotos or pointers. We will not deduct points for syntax errors unless they are significant enough to alter the meaning of your code. (20 points)

```
int origami(int A[], int B[], int a2) {
     // array A[] starts at a0 and array B[] starts at a1

     int t0 = 0;
     int v0 = 0;

     do {
          int t1 = A[t0];
          if ( t1 >= 0) {
               B[v0] = A[t0];
               v0++;
          }
          t0++;
     }while (t0 < a2);

     return v0;
}
```

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

This function inspects first a2 elements of array A and copies them into array B if they are non-negative. The return value is the number of copies.

Question 3: Write a recursive function (30 points)

Here is a function `pow` that takes two arguments (`n` and `m`, both 32-bit numbers) and returns n^m (*i.e.*, n raised to the m^{th} power). This function assumes that m is greater than or equal to one.

```
int
pow(int n, int m) {
   if (m == 1)
      return n;
   return n * pow(n, m-1);
}
```

Translate this into a MIPS assembly language function. Argument registers $a0 and $a1 will correspond to *n* and *m*, and the return value should be placed in $v0 as usual. You will not be graded on the efficiency of your code, but you *must* follow all MIPS conventions. Comment your code!!!

```
pow:    bne   $a1, 1, rec     # if m == 1, return a0
        move  $v0, $a0        # base case: set return value to n
        j     $ra             # jump back to the caller

rec:
        addi  $sp, $sp, -8    # adjust the stack to store $ra
        sw    $ra, 0($sp)     # save $ra
        sw    $a0, 4($sp)     # save $a0
        addi  $a1, $a1, -1    # recursive case: m = m - 1
        jal   pow             # call pow with n and m - 1
        lw    $a0, 4($sp)     # restore $a0
        mult  $v0, $v0, $a0   # multiply the result by n
        lw    $ra, 0 ($sp)    # restore the return address of pow
        addi  $sp, $sp, 8     # readjust the stack pointer $sp
        j     $ra             # jump back to the caller
```

Notes:

- **Since pow does not manipulate $a0, it is OK (but perhaps bad style) not to preserve its value in this program.**
- **Contents of $a1 need not be saved, because $a1 is not used by pow after the recursive call.**

Question 4: Performance (30 points)

Here is part of the Unix man page for the `bzero`() function.

```
NAME
     bzero - write zeroes to a byte string

SYNOPSIS
     void
     bzero(void *b, size_t len);

DESCRIPTION
     The bzero() function writes len zero bytes to the
     string b.  If len is zero, bzero() does nothing.
```

Here is a MIPS implementation that clears bytes one by one:

```
bzero:
        beq     $a1, $zero, end
loop:   sb      $zero, 0($a0)
        add     $a0, $a0, 1
        sub     $a1, $a1, 1
        bne     $a1, $zero, loop
end:    jr      $ra
```

Part (a)
Given a 2GHz processor (cycle time .5ns) with the CPI's in the table, what would be the exact CPU time of this function if it were clearing a string of length 25 bytes? (15 points)

Instruction type	CPI
Arithmetic (add, sub)	3
Memory (load, store)	5
Control (branches and jumps)	4

First, assign CPI values to each instruction:

```
bzero:
        beq     $a1, $zero, end     (4)
loop:   sb      $zero, 0($a0)       (5)
        add     $a0, $a0, 1         (3)
        sub     $a1, $a1, 1         (3)
        bne     $a1, $zero, loop    (4)
end:    jr      $ra                 (4)
```

Next, calculate number of cycles. Since loop is executed 25 times, the total number of cycles is:

4 + 25 * (5 + 3 + 3 + 4) + 4 = 8 + 25 * 15 = 383 cycles

Given .5 ns per cycle, the total CPU time for this function is:

383 * .5 ns = 191.5 ns

Part (b)
Compute the average CPI for this function. You can leave your answer as a fraction. (5 points)

CPI = number of cycles needed / number of instructions executed

Number of cycles, calculated above is 383.
Number of instructions is 2 + 25 * 4 = 102.

CPI = 383/102

Part (c)
If we could reduce the CPI of one instruction class (arithmetic, memory, or control) by a single cycle, which would most benefit the `bzero` code? (5 points)

Regardless of the type of instruction we select, the savings is one cycle per instruction, so we need to select the instruction class that is most often used in the function (more precisely, in the loop). The loop consists of two arithmetic instructions, one memory instruction and one control instruction. To best improve this code, we should reduce arithmetic instructions by one cycle.

Part (d)
When zeroing out large memory regions it is more efficient to use `sw` (store word) rather than `sb` (store byte) because we can do four times as much work in a single instruction. But `sw` alone can only handle strings with lengths divisible by 4. Below is a version of the code that only uses `sb` to handle the last few bytes for strings whose length is not evenly divisible by 4.

```
bzero:
        blt     $a1, 4, bytes
l1:     sw      $zero, 4($a0)
        add     $a0, $a0, 4
        sub     $a1, $a1, 4
        bge     $a1, 4, l1
bytes:  beq     $a1, $zero, end
l2:     sb      $zero, 0($a0)
        add     $a0, $a0, 1
        sub     $a1, $a1, 1
        bne     $a1, $zero, l2
end:    jr      $ra
```

Give an example of a case when the new code will be slower than the original code. (5 points)

This is not a trick question. For 0 – 3 bytes, the new code needs to perform one extra branch, compared to the old code, so the new code will be slower for strings 0 – 3 bytes long.