Full Name:_____

Andrew ID:_____

CS 15-347, Spring 1996

Exam 1

February 20, 1996

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew ID *on every page indicated.*
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 50 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like.

Do not write below this line

1. _____	3. _____	5. _____	7. _____
2. _____	4. _____	6. _____	8. _____

Useful Information

MIPS Notation Conventions

In all of the MIPS code shown in this exam, we will follow the notation conventions used in the lecture notes, especially regarding the distinction between delayed and normal branches. That is:

- Any jump or branch instruction written with UPPER CASE letters is a *delayed* branch. The instruction after it in the program will be executed regardless of whether or not the branch is taken.
- Any jump or branch instruction written with lower case letters is a *conventional* branch. The instruction after it in the program will not be executed if the branch is taken.
- All assembler directives, e.g., `.frame` are omitted.
- In some cases, line numbers have been added as references. These are not part of the actual assembly code.

Data Sizes and Alignments

You may assume the following sizes (in bytes) for different data types:

C Type	Bytes
`int`	4
Pointer	4
`char`	1
`float`	4
`double`	8

All data must be aligned.

The following problem concerns the implementation of the following recursive C function on a MIPS processor

```
/* Version of factorial that stores result in designated location */
void pfact(int n, int *result)
{
  int recval;
  if (n <= 1)
    *result = 1;
  else {
    pfact(n-1, &recval);
    *result = recval * n;
  }
}
```

The following is the code generated by GCC for a MIPS processor, modified to use our convention for describing branch instruction types.

```
 1       pfact:
 2               subu    $sp,$sp,40
 3               sw      $31,32($sp)
 4               sw      $17,28($sp)
 5               sw      $16,24($sp)
 6               move    $16,$4
 7               slt     $2,$16,2
 8               BEQ     $2,$0,$L2
 9               move    $17,$5
10               J       $L4
11               li      $2,0x00000001
12      $L2:
13               subu    $4,$16,1
14               JAL     pfact
15               addu    $5,$sp,16
16               lw      $2,16($sp)
17               mult    $16,$2
18               mflo    $2
19      $L4:
20               sw      $2,0($17)
21               lw      $31,32($sp)
22               lw      $17,28($sp)
23               lw      $16,24($sp)
24               addu    $sp,$sp,40
25               j       $31
```

Full Name:_____

Andrew ID:_____

Problem 1. (6 points):

Answer the following questions about this code:

A. How many bytes are allocated for the stack frame?

B. How many of these bytes are actually used in a normal call to `pfact`?

C. Which callee-save registers are used and what program values are stored in them?

D. Where is local variable `recval` stored?

E. Which line of assembly code computes `&recval`?

F. Which line (or lines) of assembly code update(s) `*result`?

Full Name:_____

Andrew ID:_____

Problem 2. (4 points):

Consider a C program that spends 60% of the time reading from disk (I/O time) and 40% of its time computing (compute time).

A. What is the maximum overall speedup that can be achieved by improving only the compute time?

B. What is the overall speedup achieved by improving the compute time by a factor of 2?

C. What is the maximum overall speedup that can be achieved by improving only the I/O time?

D. What speedup of the I/O time is required to achieve an overall speedup of 2?

Full Name:_____

Andrew ID:_____

Problem 3. (6 points):

Suppose that simulations of two competing machine designs reveal the following mix of machine language instructions for a typical compiled C program:

	Machine 1 (20 MHz clock rate)		Machine 2 (50 MHz clock rate)	
Instruction type	CPI	frequency	CPI	frequency
A	1	80%	2	40%
B	2	20%	4	60%

Recall that a program's running timing on machine k is related to three basic system performance measures by the equation $T_k = I_k \times CPI_k \times P_k$ where T_k is running time of the program, I_k is the number of machine language instructions, CPI_k is the average clock cycles per instruction, and P_k is the clock period. Given the instruction mix above, determine the following:

A. $P_1 =$ _____ nanoseconds/clock cycle.

B. $P_2 =$ _____ nanoseconds/clock cycle.

C. $CPI_1 =$ _____ clock cycles/instruction.

D. $CPI_2 =$ _____ clock cycles/instruction.

E. If $I_1 = I_2$, then $T_2 =$ _____ $\times T_1$. (Round your answer to two decimal places.)

F. If $T_1 = T_2$ and $I_1 = 1000$, then $I_2 =$ _____ . (Round your answer to the nearest integer.)

Full Name:_____

Andrew ID:_____

Problem 4. (6 points):

Consider the following set of benchmark results expressed as times:

Benchmark	Millions of floating point ops.	Computer 1 time (secs)	Computer 2 time (secs)	Computer 3 time (secs)
Program 1	100	1	10	20
Program 2	100	1000	100	20
Total		1001	110	40

The following table expresses these benchmark results as rates (MFLOPS) and summarizes overall performance (also in MFLOPS) using either the arithmetic mean (AM), the geometric mean (GM), or the harmonic mean (HM). For each of the last three rows, circle the mean that is computed in that row and indicate with a "yes" or "no" whether the results in that row accurately predict total running time:

Benchmark	Machine 1 rate (MFLOPS)	Machine 2 rate (MFLOPS)	Machine 3 rate (MFLOPS)	
Program 1	100.0	10.0	5.0	
Program 2	0.1	1.0	5.0	
AM GM HM	3.2	3.2	5.0	predicts running time? _____
AM GM HM	0.2	1.8	5.0	predicts running time? _____
AM GM HM	50.1	5.5	5.0	predicts running time? _____

Full Name:_____

Andrew ID:_____

Problem 5. (5 points):

The following problem concerns the following type and data declarations.

```
typedef struct {
  int type;
  int arr[8];
} arr_t;

arr_t *X;
int i;
```

Circle the letters of the following MIPS assembly code that put the value of `X[2].arr[i]` into `$8` assuming `i` is stored in register `$4` and `X` is in register `$5`. Put an X through the ones that do not. For credit either a circle or an X must be present.

A B C D E

Code A:

```
addi   $9,$5,72
lw     $9,4($9)
sll    $4,$4,2
lw     $8,0($9)
```

Code B:

```
addi   $9,$5,72
add    $9,$9,$4
addi   $8,$9,1
```

Code C:

```
addi   $9,$5,72
sll    $4,$4,2
add    $9,$9,$4
addi   $9,$9,4
lw     $8,0($9)
```

Code D:

```
lw     $9,0($5)
addi   $9,$9,72
sll    $4,$4,2
add    $9,$9,$4
addi   $9,$9,4
lw     $8,0($9)
```

Code E:

```
addi   $9,$5,72
add    $9,$9,$4
addi   $9,$9,1
lw     $8,($9)
```

Full Name:_____

Andrew ID:_____

Problem 6. (7 points):

Shown below is code generated by GCC for a MIPS processor, modified to use our convention for describing instruction types. The code was generated from the entire body of a simple C function. You don't know what the code of that C function actually was, but you do know the declaration:

```
int func(int* x, int n);
```

The code for `func` is shown below:

```
func:
        subu    $sp,$sp,8
        move    $6,$0           #hint: concentrate on the code in the
        BLEZ    $5,$L3          #inner loop before worrying about the
        move    $3,$0           #special case in which this BLEZ is taken
$L5:
        lw      $2,0($4)
        addu    $6,$6,1
        addu    $3,$3,$2
        slt     $2,$6,$5
        BNE     $2,$0,$L5
        addu    $4,$4,4
$L3:
        move    $2,$3
        addu    $sp,$sp,8
        J       $31
        nop
```

A. Give a one-sentence English description of this function.

B. "Reverse-engineer" this function by writing a short function in C that always produces the same result as the code above does. Note that the body of this function doesn't need to be more than five or six lines long!

Full Name:_____

Andrew ID:_____

Problem 7. (8 points):

The C function below has been compiled into MIPS assembly by a somewhat crude compiler that isn't very good at filling delay slots:

```
int foo(int x, int y)
{
  if (x > 0)
    return bar(y)+x;
  else
    return bar(y-x);
}
```

An annotated version of the resulting MIPS code is shown below:

```
1               subu $sp, $sp, 24       #allocate stack frame
2               sw $16, 16($sp)         #save callee save $16
3               sw $31, 20($sp)         #save return address
4               move $16, $4            #copy x to $16
5               move $4, $5             #copy y to $4 (clobber old x)
6               BGTZ $16, $L2           #if x > 0, branch to $L2
7               nop
8               subu $4, $4, $16        #calculate (y-x)
9               JAL bar                 #jump to bar with (y-x) as arg
10              nop
11              J $L4                   #jump to end with bar(y-x) as ret val
12              nop
13      $L2:
14              JAL bar                 #jump to bar with y as argument
15              nop
16              addu $2, $2, $16        #return value = bar(y)+x
17      $L4:
18              lw $31, 20($sp)         #restore return address
19              lw $16, 16($sp)         #restore callee save $16
20              addu $sp, $sp, 24       #deallocate stack frame
21              J $31                   #we're outta here
22              nop
```

For each of the following lines with `nops` in delay slots, indicate which (useful) instructions could safely be moved into the delay slot without changing the behavior of the function. Circle the line numbers of the instructions that could be moved into the slot, and cross out the ones that can't. (When working on filling a given delay slot, assume that the only change that may be made from the original code is the movement of one instruction—unaltered—into the delay slot.)

```
A.  Line 7:    3   4   5   8
B.  Line 10:   3   4   5   8
C.  Line 14:   8  16  18  19
D.  Line 21:  16  18  19  20
```

Full Name:_____

Andrew ID:_____

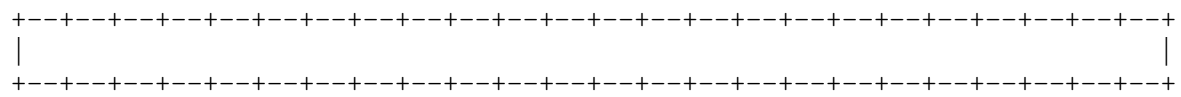
Problem 8. (8 points):

Consider the following data type definitions:

```
typedef struct {
  float f[2];
  struct {
    int i;
    char c;
  } structs[2];
} struct1;
```

```
typedef struct {
  double d;
  int is[2];
  char cs[2];
} struct2;
```

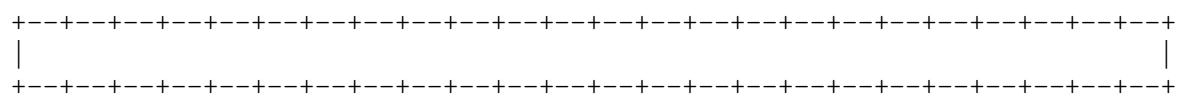
A. Using the template below (allowing a maximum of 24 bytes), indicate the allocation of data for a structure of type `struct1`. Mark off and label the areas for each individual element (there are 6 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment). Clearly indicate the right hand boundary of the data structure.



B. How many bytes are allocated for each element of type `struct1`?

C. What alignment is required for an element of type `struct1`?

D. Label the template below for data type `struct2` (5 individual elements).



E. How many bytes are allocated for each element of type `struct2`?

F. What alignment is required for an element of type `struct2`?