Full Name:____

Andrew ID:____

CS 15-347, Spring 1997

Final Exam

May 13, 1997

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name and Andrew ID on the cover sheet.
- 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 100 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. ________	6. ________	11. ________
2. ________	7. ________	12. ________
3. ________	8. ________	13. ________
4. ________	9. ________	
5. ________	10. ________	
________	________	________
total ________		

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.

Data Sizes and Alignments

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

C Type	Bytes
`short`	2
`int`	4
Pointer	4
`char`	1
`float`	4
`double`	8

All data must be aligned.

Full Name:____

Andrew ID:____

Problem 1. (10 points):

The fundamental things apply, as time goes by.

In Homework 1 you wrote a function called $ftime(P, E_{max})$ that estimates the running time of a function P with a *relative error* of at most E_{max}. An interesting question that arose on the course newsgroup is whether we can use $ftime$ to produce an estimate with a bounded *absolute error*. In other words, can we somehow choose a value of parameter E_{max} to pass to $ftime$ that bounds the absolute error of its estimate? It turns out that we can indeed choose a value for E_{max} that bounds the absolute error, so long as we have a single measurement that estimates the running time of P.

Let P be a function with an actual running time of t_a seconds. Let t_f be the estimated running time in seconds returned by $ftime(P, E_{max})$. And let ϕ be the system clock frequency in units of clock cycles/second. The goal is to determine the maximal value of E_{max} to pass to $ftime$ such that the absolute error of its estimate is at most 1 clock cycle. In other words, we want to choose an E_{max} such that

$$\phi|t_f - t_a| \leq 1. \tag{1}$$

The solution can be derived in a series of steps, starting with the following observation: If you call $ftime$ with a parameter of E_{max}, then from the definition of $ftime$ you know that the following inequality holds:

$$\frac{|t_f - t_a|}{t_a} \leq E_{max} \tag{2}$$

A. Starting from Equation (2), use simple algebra to derive the upper bound on the absolute error, in units of clock cycles, and in terms of ϕ, t_a, and E_{max}:

$$\phi|t_f - t_a| \leq \tag{3}$$

B. Suppose that you use $get\_etime()$ to obtain a single measured time of t_m seconds for P:

```
start = get_etime();
P();
tm = get_etime() - start;
```

The rhs of Equation (3) is based on the unknown value of t_a. Find an upper bound to this rhs in terms of ϕ, t_m, Δ (the timer interval), and E_{max}:

$$\phi|t_f - t_a| \leq \tag{4}$$

C. Noticing that Equation (4) has the same lhs as Equation (1) (which is our original goal), derive the maximal value for E_{max} which guarantees that the absolute error is at most 1 clock cycle. Your answer should be expressed in terms of ϕ, t_m, and Δ:

$$E_{max} =$$

D. What are the units in your answer to part C?

Full Name:_______________________________________

Andrew ID:_______________________________________

For the next problem, consider the following C loop, which computes the inner product of two n-element floating-point vectors a and b:

```
for (i=0; i<n; i++) {
   sum += (*a++) * (*b++);
}
```

gcc -O translates this C code into the following MIPS assembly code:

```
# on entry: r4=&a, r5=&b, r6=n, r3=i=0, f4=sum=0.0
L1:     lwc1 $f0,0(r4)      #f0 = *a
L2:     lwc1  $f2,0(r5)     #f2 = *b
NOP:    nop
MS:     mul.s $f0,$f0,$f2   #f0 = (*a) * (*b)
AS:     add.s $f4,$f4,$f0   #sum += (*a) * (*b)
A1:     addiu r5,r5,4       #b++
A2:     addiu r3,r3,1       #i++
SLT:    slt   r2,r3,r6      #loop while i<n
BNE:    BNE   r2,r0,L1
A3:     addiu r4,r4,4       #a++
```

Assume that a MIPS floating point add requires 2 cycles and that a MIPS floating point multiply requires 5 cycles. Assume that the pipeline stalls if there is a dependent operation in the load delay slot of the lwc1 instruction. Also, assume that this code is being executed by a real MIPS machine, so there is no stalling or cancelling involved in handling a branch instruction.

Problem 2. (8 points):

Speed freaks only need apply.

A. How many clock cycles does one iteration of the above inner product loop require? (Including the instruction in the branch delay slot).

B. Fill in the table below with the instruction labels corresponding to an optimal schedule for this loop. Note that there is not a unique optimal schedule. You are only allowed to reorder instructions within one one basic block (i.e., no loop unrolling or software pipelining), and to possibly eliminate the nop instruction. You may not change any operands, and your schedule preserve the original program semantics.

Num	Label
1	`L1`
2	`L2`
3	
4	
5	
6	
7	
8	
9	
10	

C. How many clock cycles does one iteration of your optimized loop in part B require.

Full Name:____

Andrew ID:____

Problem 3. (6 points):

Ethernet mediums are not so rare.

This question explores the tradeoff between minimum packet size and maximum wire length in an Ethernet. Assume that the network fabric consists of a single copper wire, with no repeaters or bridges that introduce delay. And assume a propagation speed through copper of 20 cm/ns.

A. If the minimum packet size in a 100 Mbit/sec (i.e., 100×10^6 bits/sec) Ethernet is 64 bytes, then what is the maximum wire length (in meters)? (Hint: compute the required propagation delay τ for a 64-byte packet and then use this to determine the appropriate wire length.)

B. If the maximum wire length in a 100 Mbit/sec Ethernet is 1024 meters, then what is the minimum packet size (in bytes)?

The next problem concerns the following scenario:

You are charged with modifying part of a large C program, and you come across the following code:

```
typedef struct {
  int left;
  a_struct a[CNT];
  int right;
} b_struct;


void test(unsigned i, b_struct *bp)
{
  int n = bp->left + bp->right;
  a_struct *ap = &bp->a[i];
  ap->x[3] = n;
}
```

Unfortunately, the '`.h`' file defining the compile-time constant `CNT` and the structure `a_struct` are in files for which you do not have access privileges. Fortunately, you have access to a '`.o`' version of your code. You are able to disassemble this with the `dis` program. Here is the disassembly of the procedure `test`, modified to use our branch notation:

```
    lw      r3,0(r5)
    lw      r2,224(r5)
    nop
    addu    r3,r3,r2
    sll     r2,r4,1
    addu    r2,r2,r4
    sll     r2,r2,3
    addiu   r2,r2,8
    addu    r5,r5,r2
    JR      r31
    sw      r3,20(r5)
```

Full Name:______________________________________

Andrew ID:______________________________________

Problem 4. (12 points):

Why doesn't CMU have a Reverse Engineering Department?

Using your reverse engineering skills, deduce the following:

A. The size of the structure `a_struct`.

B. The alignment requirements for `a_struct`.

C. The value of `CNT`.

D. A diagram of the complete layout for `a_struct`. [You may not know the names of all of the fields for this structure, but you can determine their types.] Mark off the different fields and indicate as much information as you can. Clearly indicate the end of the structure by crossing off any unused bytes. The diagram allows a maximum of 36 bytes.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following problem concerns the components of the memory hierarchy and the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Virtual addresses are 14 bits wide.
- Physical addresses are 12 bits wide.
- The page size is 64 bytes.
- The TLB is 4-way set associative with 16 total entries.
- The cache is direct mapped, with a 4 byte line size and 16 total lines.

In the following tables, all numbers are given in hexadecimal. The contents of the TLB are as follows.

TLB												
Set Index	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	21	0	09	0D	1	00	13	0	07	02	1
1	03	17	1	02	13	0	04	21	0	0A	11	1
2	02	66	1	08	8F	0	06	FF	0	03	2D	1
3	07	29	0	03	0D	1	0A	F4	1	02	1B	0

The page table for the first 16 pages and the cache are as follows:

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
000	28	1	008	13	1
001	2B	0	009	17	1
002	33	1	00A	09	0
003	02	1	00B	1A	0
004	2A	0	00C	27	0
005	16	1	00D	2D	1
006	04	0	00E	11	1
007	26	0	00F	0D	1

Cache						
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11
1	15	0	55	59	0B	41
2	1B	1	00	02	04	08
3	36	0	84	06	B2	9C
4	82	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	91	0	A2	37	68	10
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	46	92	88	FE
A	24	1	93	15	DA	3B
B	0B	0	71	49	10	05
C	12	0	82	03	02	12
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	20	07	12	19

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 5. (4 points):

It's a good thing they make hardware that does this for you.

A. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag

13	12	11	10	9	8	7	6	5	4	3	2	1	0

B. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPO	The physical page offset
PPN	The physical page number
CO	The byte offset within the cache line
CI	The cache index
CT	The cache tag

11	10	9	8	7	6	5	4	3	2	1	0

For the following 2 problems, you are to show how the memory system translates a virtual address into a physical address and accesses the cache. For the given virtual address, indicate the TLB entry accessed, the physical address, and the cache byte value returned. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs. Carry each translation as far as possible, i.e. until there is insufficient data to complete the translation. Mark '–' through entries that cannot be correctly determined.

The following table of hex values will come in handy:

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Full Name:____
Andrew ID:____

Problem 6. (6 points):

Are we having fun yet?

Virtual address: `027C`

A. Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0

B. Address translation

Parameter	Value
VPN	
TLB Index	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	

C. Physical address format

11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte returned	

Full Name:_________________________________

Andrew ID:_________________________________

Problem 7. (6 points):

You guys never give up do you?

Virtual address: `03D4`

A. Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0

B. Address translation

Parameter	Value
VPN	
TLB Index	
TLB Tag	
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	

C. Physical address format

11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	
Cache Index	
Cache Tag	
Cache Hit? (Y/N)	
Cache Byte returned	

Full Name:______________________________________

Andrew ID:______________________________________

Problem 8. (12 points):

Hmmm...Are we using a Pentium or a Pentium II?

A string of 32 bits can be interpreted as a number in many different ways. For this problem, we will look at the three most common number formats:

Unsigned Only nonnegative integers are encoded.

Two's Complement Signed integers are encoded in two's complement form.

Floating Point Single-precision floating point values are encoded according to the IEEE standard. That is, the 32-bit word is split into a 1 bit sign, an 8-bit exponent, and a 23-bit significand. The exponent bias is 127. [**Important Note:** The lecture notes incorrectly list the size of the exponent field as being 7 bits.]

In the following table, each row shows the hexadecimal representation of a 32-bit quantity. Fill in the three numeric interpretations for each row. For each entry, give an exact values expressed as an arithmetic expression, possibly including exponents. Examples of valid entries include: 255×2^{16}, 65,536, and $2^{17} - 1$., Examples of invalid entries include: $111111100000000000000000_2$, and $\approx 3.14 \times 10^{16}$, or $2^{114} - \epsilon$. You may also use the special values *NaN* and ∞.

Hexadecimal	Unsigned	Two's Comp.	Floating Pt.
00000000			
FF800000			
0000002A			
80000000			
FFFFFFFF			

We wish to implement a data structure for representing a set of points in a plane, where each point is represented by a pair of integers encoding the X and Y coordinates. Assume there is a limit of 1024 (`0x400`) points in a set.

Consider the following two alternatives in organizing this data structure:

```
/* Separate */
typedef struct {
  int x[1024];
  int y[1024];
} separate_set;
#define X(pset,i) pset->x[i]
#define Y(pset,i) pset->y[i]
```

```
/* Paired */
typedef struct {
struct { int x, y } pt[1024];

} paired_set;
#define X(pset,i) pset->pt[i].x
#define Y(pset,i) pset->pt[i].y
```

We will consider the access patterns represented by the following three loops, where each loop is run on a point set that is not initially in the cache.

```
/* Loop A */
for (i=0; i<1024; i++)
{
  int x = X(ps,i);
  int y = Y(ps,i);
  ...
}
```

```
/* Loop B */
for (i=0; i<1024; i++)
{
  int x = X(ps,i);
  ....
}
```

```
/* Loop C */
for (i=0; i<1024; i += 3)
{
  int x = X(ps,i);
  int y = Y(ps,i);
 ....
}
```

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 9. (3 points):

So, what's the point?

We plan to run this program on a machine having a cache with a line size of 16 bytes (4 words). The cache is sufficiently large that the effects of conflict misses can be ignored.

A. Assume a data structure of type `separate_set` is allocated with a starting address of `0x100`. In the figure below, indicating the first 12 points in the data structure, draw rectangles around groups of these 24 words indicating the partitioning into cache blocks. Label each block with its starting address.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	...
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	...

B. Do the same for a data structure of type `paired_set` starting at address `0x100`:

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	...
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	...

C. Fill in the following table, giving the approximate *miss rate* that will occur while accessing the data structure elements for each of the the three loops, and both of the possible organizations.

Case	Separate	Paired
Loop A		
Loop B		
Loop C		

Full Name:_______________________________________

Andrew ID:_______________________________________

For the following problem, you are to draw the pipeline chart (yet again).

Assume that the architecture of this problem is the pMIPS architecture from Lab 2, with the following properties:

- The processor handles hazards by forwarding results whenever possible.
- There is no delay slot after loads. The instruction following a load may use the result of that load. The processor stalls to allow this only if necessary.

Please fill in the appropriate boxes with the letters F,D,X,M or W. Where forwarding happens, draw an arrow from the stage where the result is generated to the stage where the result is used. Where a stage gets stalled, just fill in the same letter twice.

Problem 10. (8 points):

More pipeline mania.

Here is a hazardous piece of code:

```
li     $4, 16
li     $5, 19
addu   $4,$4,$5
lw     $7,0($4)
addu   $8,$7,$4
sw     $8,0($8)
```

To get you started, we have already provided the first few timesteps:

Instr	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
li $4,16	F	D	X	M								
li $5,19		F	D	X								
addu $4,$4,$5			F	D								
lw $7,0($4)				F								
addu $8,$7,$4												
sw $8,0($8)												

Full Name:______________________________________
Andrew ID:______________________________________

Problem 11. (6 points):

Went off to Congress and served a spell . . .

Consider a concurrent system containing two processes executing the following programs:

```
 Process A                              Process B
(A1)   ta = x;                        (B1)   tb = y;
(A2)    y = ta;                       (B2)    x = tb;
```

In these programs, `x` and `y` are shared variables, while `ta` and `tb` are private. Initially `x` is 1, and `y` is 0. Assume these processes are executed on a parallel computer that guarantees *sequential consistency*, as described in Lecture 27. That is, any set of concurrently-executing processes has an effect consistent with some interleaving of the individual program steps, such that the steps for each individual process occur in program order.

For each of the following outcomes, give some interleaving of the steps `A1`, `A2`, `B1`, and `B2`, such that this outcome would occur. If there is no such interleaving, label the outcome as "sequentially inconsistent."

A. `x == 0` and `y == 0`

B. `x == 0` and `y == 1`

C. `x == 1` and `y == 0`

D. `x == 1` and `y == 1`

Full Name:_______________________________________

Andrew ID:_______________________________________

Problem 12. (4 points):

Why do we say "Aloha" when we really mean goodbye?

Consider the following C data type definitions:

```
typedef struct LLStruct* LinkListPtr;

typedef struct LLStruct {
  LinkListPtr next;
  double data;
} LinkList;

typedef union {
  char hello;
  int conigewa;
  LinkList  niHow;
} AlohaUnion;
```

A. How many bytes are allocated for a variable of type `LinkList`?

B. What alignment is required for a variable of type `LinkList`?

C. How many bytes are allocated for a variable of type `AlohaUnion`?

D. What alignment is required for a variable of type `AlohaUnion`?

Full Name:______________________________________

Andrew ID:______________________________________

Problem 13. (8 points):

Ce n'est pas une pipe.

Consider the following C loop with four code sections labelled `Ai` to `Di` in the loop body:

```
for (i = 0; i < n; i++) {
      Ai
      Bi
      Ci
      Di
}
```

The following is the same loop transformed by a buggy compiler based on 4 stage software pipelining:

```
/* Prolog */
A0
B0
A1
C0
B1
A2

/* main loop */
for (i = min; i < max; i+=stride) {
      Di-3
      Ci-2
      Bi-1
      Ai
}

/* Epilog */
 ... some missing code sections
 Cn-1
 Dn-1
```

A. What are the correct values for `min`, `max`, and `stride`?

B. What are the missing code sections in the epilog?