Full Name:______________________________________
Andrew ID:______________________________________

CS 15-347, Spring 1996

Final Exam

May 8, 1996

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 120 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. ________	16. ________
2. ________	7. ________	12. ________	17. ________
3. ________	8. ________	13. ________	
4. ________	9. ________	14. ________	
5. ________	10. ________	15. ________	
Total.________			

Problem 1. (9 points):

Suppose you need to ship an enormous amount of scientific data across the country, and that the two possible methods of getting the data there are:

- Sending it over a 100 Megabytes/hour network connection. Since you're sending over a lot of data, the latency in this case is negligible.
- Packing a copy of the data—which you happen to have on magnetic tapes—and driving out with it in your trusty old station wagon. After delivering the tapes, you must drive back. Furthermore, you may need to take multiple trips: your station wagon can only carry a terabyte (that's 10^{12} bytes) at a time. (Did we mention you have an enormous amout of data?)

It's about 5,000 km (by car) to the data's destination, and you can travel an average of about 100 km/hour if you and your cohorts take turns driving without risking getting a speeding ticket.
Assume that all you care about is getting all the data to its destination as quickly as possible, and so you need not count the time required for the final return trip.

A. Assuming you're going to take a lot of trips, what's the bandwidth of the station wagon?

B. What's the smallest amount of data you'd need to be sending in order for the station wagon to be the fastest way of getting all of it to its destination?

C. Now, suppose that Montana has just abolished all of its speed limits, and that your station wagon has a rocket engine strapped to its roof rack. Driving through Montana used to take 10% of your trip time each way, but now you can drive through Montana 10 times as fast. By what factor does the bandwidth of the station wagon improve?

Problem 2. (5 points):

A disk of radius r consists of a hole of radius h followed by t tracks, where t is proportional to $r - h$. The number of bits per track is constant and is determined by the number of bits that can be stored on the innermost track, which is proportional to the circumference of the innermost track. Given r, find the value of h that maximizes the bit capacity of the entire disk. (Hint: formulate the solution as a continuous problem.)

Problem 3. (10 points):

Consider a timed C loop of the form:

```
start = get_etime();
for (i=0; i<n; i++) {
    ...;
}
tm = (get_etime() - start) * 1000.0; /* measured time in ms */
```

If the timer is updated at a rate of 100 ticks/second, what value of `tm` is required to ensure 95% accuracy. Here, accuracy is defined as $1 - \frac{|t_m - t_a|}{t_a}$, where t_m is the measured time in milliseconds (variable `tm` in the code above), t_a is the actual running time in milliseconds, and $|x|$ denotes the absolute value of x. Assume that the only source of error in the measurement is due to the timer resolution.

Problem 4. (5 points):

Consider a program that wants to transfer n MBytes from memory to disk. There are two approaches: (1) Using programmed I/O, the program can transfer data in a tight loop from memory to disk (via the register file) at a rate of 1 MByte/sec. (2) The program can use a DMA controller attached to the bus, which requires an initial setup time of 125 ms, after which it transfers data from memory to disk at a rate of 2 MBytes/sec. What is the crossover point for (1) and (2) as a function of n? In other words, for what value of n are the transfer times for the two approaches the same?

MIPS Code Optimization

The following series of problems concern efforts to optimize C code for running on a MIPS architecture.

You are given the following C program to compute the difference between successive elements of an array `src`, and store the results in an array `dest`.

```
void adiff1(int src[], int dest[], int num_ele)
{
  int i;
  dest[0] = src[0];
  for (i = 1; i < num_ele; i++)
    dest[i] = src[i] - src[i-1];
}
```

You have been assigned the task of improving the performance of this code. Within minutes, you generate 3 variations on the original procedure. The first builds on the lesson you learned in Lab 1, that you should introduce temporary variables to avoid extra memory references:

```
void adiff2(int src[], int dest[], int num_ele)
{
  int i;
  int prev, curr;
  dest[0] = prev = src[0];
  for (i = 1; i < num_ele; i++) {
    curr = src[i];
    dest[i] = curr - prev;
    prev = curr;
  }
}
```

The second is based on replacing array code with pointer code:

```
void adiff3(int *src, int *dest, int num_ele)
{
  int i;
  int prev, curr;
  *dest = prev = *src;
  for (i = 1; i < num_ele; i++) {
    curr = *(src+i);
    *(dest+i) = curr - prev;
    prev = curr;
  }
}
```

The third uses every trick you know, including unrolling the loop to generate two elements on each iteration, and eliminating the index variable `i`. Don't worry about correctness—this code has been tested and compared to `adiff2`:

```
void adiff4(int *src, int *dest, int num_ele)
{
  int ping, pong;                    /* Successive elements of src */
  int *src_end = src+(num_ele-1);   /* Stopping point */
  *dest++ = ping = *src++;
  while (src < src_end) {
    pong = *src;
    *dest = pong - ping;             /* update dest[i]   */
    ping = *(src+1);
    *(dest+1) = ping - pong;         /* update dest[i+1] */
    src += 2;                        /* i+= 2 */
    dest += 2;
  }
  if (src == src_end)                /* Extra update if num_ele even */
    *dest = *src - ping;
}
```

Running GCC on these procedures, with `-O` and `-S` switches set, you get 3 different assembly code versions. Only the code corresponding to the inner loops is shown below, and we use our notation of showing delayed branches in upper case:

```
adiff1:
        ....    # setup
$L1:            # Main Loop
        sll     $2,$7,2
        addu    $4,$2,$5
        addu    $2,$2,$8
        lw      $3,0($2)       # Load #1
        lw      $2,-4($2)      # Load #2
        addu    $7,$7,1
        subu    $3,$3,$2
        slt     $2,$7,$6
        BNE     $2,$0,$L1
        sw      $3,0($4)
        ....    # finish

adiff2:
        ....    # setup
$L1:            # Main Loop
        sll     $3,$7,2
        addu    $2,$3,$9
        lw      $4,0($2)       # Load #3
        addu    $3,$3,$5
        addu    $7,$7,1
        subu    $2,$4,$8
        sw      $2,0($3)
        slt     $2,$7,$6
        BNE     $2,$0,$L1
        move    $8,$4
$L2:
        ....    # finish

adiff4:
        ....    # setup
$L1:            # Main Loop
        lw      $3,0($4)       # Load #4
        subu    $2,$3,$6
        sw      $2,0($5)
        lw      $6,4($4)       # Load #5
        addu    $4,$4,8
        subu    $2,$6,$3
        sw      $2,4($5)
        sltu    $2,$4,$7
        BNE     $2,$0,$L1
        addu    $5,$5,8
        ....    # finish
```

Problem 5. (5 points):

Describe a condition under which procedures `adiff1` and `adiff2` would produce different results:

[For this reason, GCC is unable to generate the more optimized code of `adiff2` when given the source code of `adiff1`].

Problem 6. (5 points):

Conventional wisdom holds that pointer code is more efficient than array code. Explain, therefore, why compiling procedures `adiff2` and `adiff3` yields the exact same assembly code.

Problem 7. (5 points):

The MIPS assembler must insert NOP's to eliminate hazards following load instructions. For each of the 5 `lw` instructions shown in the assembly code, indicate whether or not a `nop` should be inserted after it. Fill in the table with "Y" for yes, or "N" for no. Entries left blank will be marked as incorrect.

Load Instruction	NOP required (Y/N)
# 1	
# 2	
# 3	
# 4	
# 5	

Problem 8. (6 points):

Assume the entire array `src` is held in cache. On a standard MIPS machine, each of the instructions used in the three procedures (including the NOP's inserted) would require 1 clock cycle.
Indicate the approximate number of cycles to generate each element of the destination array, counting only the code within the inner loop (i.e., ignore the overhead cost of setting up and completing the loop).

`adiff1`:

`adiff2`:

`adiff3`:

Problem 9. (12 points):

Consider the following two ways of representing signed integers:

Two's complement

Sign-Magnitude in which the most significant bit indicates whether or not the number is negative, and the remaining bits give its absolute value. The number 0 always has its sign bit set to 0.

Suppose we have a machine with a 4-bit word size that normally uses a two's complement representation for signed numbers. We want to write code to perform a "less-than" test for both of the formats above. Here are three candidates. Note that each declares its arguments as unsigned, but we assume there would be no format conversion before the code is executed.

```
int comp1(unsigned x, unsigned y)
{
  return (x < y);  /* Unsigned compare */
}

int comp2(unsigned x, unsigned y)
{
  int ix = x;
  int iy = y;
  return (ix < iy);  /* Signed compare */
}

int comp3(unsigned x, unsigned y)
{
  unsigned sx = x >> 7;
  unsigned sy = y >> 7;
  return (sx && !sy) || (x < y);
}
```

Fill in the following table indicating the 4-bit patterns to represent the indicated numeric values using the two formats.

Value	Two's Comp.	Sign-Mag.
-7		
-3		
0		
$+5$		
$+7$		

Fill in the following table indicating which of the three procedures would reliably serve as a comparison function for each of the two formats. Fill each entry as "OK" if it would work for all possible arguments, or with a pair of arguments for which the procedure would produce the incorrect results. Choose your arguments from the values in the table above

	Two's Comp.	Sign-Mag.
`comp1`		
`comp2`		
`comp3`		

Pipelined Processors

For each of the following problems, you are to draw the pipeline chart.

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

- Branches are taken when the branch instruction reaches the MEM stage. When a branch is taken, the two instructions following the instruction in the delay slot are cancelled.
- 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.

Where forwarding happens, draw an arrow from the stage where the result is generated to the stage where the result is used. Where stalling happens, circle the stage that is stalled. When an instruction is cancelled, draw an X over the stage in which it is cancelled.

Problem 10. (7 points):

```
addiu $4, $4, 1
lw $1, 0($4)
lw $3, 0($1)
sw $3, 0($5)
```

Instruction	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
addiu	IF	ID	EX	MEM	WB						
ld		IF	ID	EX	MEM						
ld			IF	ID	EX						
sw				IF	ID						

Problem 11. (8 points):

In this problem, assume that the branch is taken.

```
        addiu $5, $5, 1
        BNE $5, $6, $L2  # Delayed branch
        addiu $5, $5, 1
        addiu $5, $5, 2
        addiu $5, $5, 4
        addiu $5, $5, 8
$L2:
        addiu $5, $5, 16
```

Instruction	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
addiu $5, $5, 1	IF	ID	EX	MEM	WB						
bne $5, $6, $L2		IF	ID	EX	MEM						
addiu $5, $5, 1			IF	ID	EX						
addiu $5, $5, 2				IF	ID						
addiu $5, $5, 4					IF						
addiu $5, $5, 8											
addiu $5, $5,16											

Problem 12. (12 points):

Consider a hypothetical version of the MIPS architecture with the following cache characteristics:

- The cache has 16 lines.
- The cache has a line size of 4 bytes.
- The cache is write-through.

Consider the following alternative policies for the associativity and replacement policy of the cache:

A. Directly mapped

B. Fully Associative, Least-Recently-Used Replacement Policy. Invalid cache lines are used before valid cache lines.

Assume you run the following program on the machine, where the cache is empty before each loop begins:

```
void loop(int *A, int cnt, int stride)
{
  register int i,j;
  for(i = 0; i < 2; i++)
    for(j = 0; j < cnt; j++)
      A[j * stride] *= 2;
}
```

For each of the following values of `cnt` and `stride`, you are to identify what types of cache hits and misses there will be for the loads in the code for each of the above types of cache. (Do not worry about the stores.) Identify the totals of each of the following types of accesses:

Hit Valid data is found in the cache.

Compulsory Miss A miss happens because the data has never been in the cache.

Other Miss Another type of cache miss happens.

A. `cnt` = 16, `stride` = 2.

	Directly Mapped	Fully Assoc. LRU
Hits		
Compulsory Misses		
Other Misses		
Total Reads	32	32

B. `cnt` = 17, `stride` = 1.

	Directly Mapped	Fully Assoc. LRU
Hits		
Compulsory Misses		
Other Misses		
Total Reads	32	32

fixPackets Revisited

On Recitation Assignment 2, the problem given was to optimize the following procedure:

```
struct DataPacketStruct
{
  char header[4];
  unsigned int data[16];
};

typedef struct DataPacketStruct DataPacket;

void fixPacketsOriginal(DataPacket* fixed, DataPacket* fouled, int n)
{
  int i;
  for (i = 0; i < n; i++)
    {
      fixed[i] = fouled[i];
      fixed[i].data[0] = fouled[i].data[1];
      fixed[i].data[1] = fouled[i].data[0];
      fixed[i].data[10] *= 10;
    }
}
```

To copy large structures, gcc uses the library function `memcpy()`. Calling `memcpy(dest, src, n)` copies `n` bytes from `src` to `dest` as efficiently as possible; it's often implemented as a hand-optimized loop.

So, the function really looks like this:

```
void fixPacketsOriginal(DataPacket* fixed, DataPacket* fouled, int n)
{
  int i;
  for (i = 0; i < n; i++)
    {
      memcpy(&fixed[i], &fouled[i], sizeof(DataPacket));
      fixed[i].data[0] = fouled[i].data[1];
      fixed[i].data[1] = fouled[i].data[0];
      fixed[i].data[10] *= 10;
    }
}
```

In one of Ralph's attempts to speed up the `fixPackets` function, he moved the memcpy out of the for loop thus:

```
void fixPackets(DataPacket* fixed, DataPacket* fouled, int n)
{
  int i;
  memcpy(fixed, fouled, n * sizeof(DataPacket));
  for (i = 0; i < n; i++)
    {
      fixed[i].data[0] = fouled[i].data[1];
      fixed[i].data[1] = fouled[i].data[0];
      fixed[i].data[10] *= 10;
    }
}
```

Problem 13. (5 points):

When Ralph timed this code for n = 100, `fixPackets` was much faster than `fixPacketsOriginal`. When he timed this code for n = 1000, `fixPackets` was much slower than `fixPacketsOriginal`. Explain qualitatively why `fixPackets` is faster than `fixPacketsOriginal` for some values of n and slower for other values.
[**Hint**: Calculate the sizes of the arrays for n = 100 and n = 1000.]

Problem 14. (6 points):

In this problem you will determine the properties of a processor's cache, given the performance of the processor on a test program.
You know the following about the program and processor:

- The processer runs at 100MHz.
- The program loops over an array of size SIZE and reads in 4 bytes of data at a stride of STRIDE bytes.
- The program has (essentially) no overhead instructions.
- every loop is executed 1000000 times.

The following data and plot show the results of the test program:

Size	Stride				
	64	32	16	8	4
4096	400000	400000	400000	400000	400000
8192	400000	400000	400000	400000	400000
16384	400000	400000	400000	400000	400000
32768	400000	400000	400000	400000	400000
65536	36364	36364	36364	66667	114286
131072	36364	36364	36364	66667	114286

Based on this data, determine the following cache parameters:

A. The size of the cache in bytes:

B. The line size of the cache in bytes:

C. The miss penalty for the cache in cycles:

Problem 15. (5 points):

Suppose you are examining a memory architecture with the following features:

- The memory is byte addressable.
- Virtual addresses are 14 bits wide.
- Physical addresses are 12 bits wide.
- The page size is 256 bytes.
- The TLB is 8-way set associative with 32 total entries.
- The cache is direct mapped and physically addressed and physically tagged, with a 16 byte line size and 8 total lines.

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:

VPN The virtual page number

VPO The virtual page offset

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:

PPN The physical page number

PPO The physical page offset

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

Memory Hierarchy

The following series of problems concern a hierarchical memory structure with the following features:

- 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 32 total entries.
- The cache is direct mapped, **virtually addressed and virtually tagged**, with a 8 byte line size and 8 total lines.

Note that this memory structure is **different** in several respects from the memory structure in the previous problem!
In the following tables, all numbers are given in hexadecimal.

TLB

Index	Member 0			Member 1			Member 2			Member 3		
	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	1F	00	1	0B	03	0	06	26	1	05	23	1
1	18	11	1	01	0B	0	01	03	1	1D	17	0
2	1C	08	1	07	09	1	0C	07	0	02	10	1
3	08	1A	1	08	1E	0	09	20	1	12	18	0
4	02	17	1	12	01	0	00	04	0	0F	15	1
5	03	00	0	0F	28	1	1B	19	1	19	0A	1
6	00	16	0	01	16	1	15	05	1	12	11	0
7	1E	0E	1	18	1F	1	00	24	0	03	05	0

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

Cache

Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0	04	0	49	20	73	6E	6F	72	74	20
1	02	1	74	68	65	20	6E	6F	73	65
2	0A	1	2C	20	4C	75	63	69	66	65
3	1B	0	72	21	20	20	42	61	6E	61
4	21	0	6E	61	21	20	20	42	61	6E
5	11	1	61	6E	61	21	20	20	46	6C
6	07	0	79	69	6E	67	20	70	75	72
7	18	1	70	6C	65	20	68	69	70	70

Page Table (Partial)

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
000	0B	1	004	04	0	008	22	1	00C	1D	1
001	27	1	005	07	1	009	03	1	00D	21	0
002	14	1	006	1C	1	00A	2F	1	00E	16	1
003	02	1	007	04	1	00B	1D	0	00F	20	0

For the following problems, you are to show how the processor accesses the cache and translates virtual addresses into physical addresses. For the given virtual address, indicate the cache entry accessed, the cache byte value returned, the TLB entry accessed, and the physical address computed. 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. (Note that you should carry through as many steps as possible on each problem, even if a real processor wouldn't necessarily need to. For example, you must carry through the physical address computation if possible, even if the virtual address accessed results in a cache hit.)

Problem 16. (10 points):

Virtual address `0132`

Virtual address format:

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

VPN: bits 13–6; VPO: bits 5–0

TLBT: bits 13–8; TLBI: bits 7–6

CT: bits 13–6; CI: bits 5–3; CO: bits 2–0

A. Cache reference

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

B. Address translation

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

C. Physical address

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

Problem 17. (10 points):

Virtual address `008D`

Virtual address format:

<table>
<tr><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">VPN</td><td colspan="6">VPO</td></tr>
<tr><td colspan="6">TLBT</td><td colspan="2">TLBI</td><td colspan="6"></td></tr>
<tr><td colspan="8">CT</td><td colspan="3">CI</td><td colspan="3">CO</td></tr>
</table>

A. Cache reference

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

B. Address translation

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

C. Physical address

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