CSE 378 Autumn 2007 **Final Exam**

Machine Organization & Assembly Language

Write your answers on these pages. Additional pages may be attached (with staple) if necessary. Please ensure that your answers are legible. Please show your work. Write your name at the top of each page.

Problem	Points
1	/ 20
2	/ 10
3	/ 15
4	/ 5
5	/ 5
6	/ 10
7	/ 10
8	/ 15
9	/ 10
TOTAL	/ 100

1. [20 points] **MIPS Assembly Programming**

   Write two assembly functions: `ROT7` that takes a null-terminated string of ASCII characters pointed at by `$a0` and rotates each character by 7 in place and `UNROT7`, which also takes a null-terminated string of ASCII characters pointed at by `$a0` and *unrotates* each character by 7 in place. You must implement the body of `LOOP` in the template below.

   Here are some example inputs and outputs:

   The string
   `ABCDEFGH` becomes
   `HIJKLMNO` when `ROT7` is applied to it.

   And the string
   `ABCDEFGH` becomes
   `TUVWXYZA` when `UNROT7` is applied to it.

   You may assume that the strings passed to your functions will contain only UPPER CASE LETTERS.

```
ROT7:
    addi $s0, $0, 7     #Set arg for leaf function
    j ROT               #Jump to leaf function
UNROT7:
    addi $s0, $0, 19    #Set arg for leaf function
    j ROT               #Jump to leaf function
ROT:
    addi $s2, $0, 26
LOOP:
    # constants you may find useful
    addi $v0, $0, 65    # ASCII character code for 'A'
    addi $v1, $0, 90    # ASCII character code for 'Z'
```

```
DONE:
    jr $ra              #Return
```

2. [10 points] **Pointers**.

```
int areEqual(char a, char b, char c, char d) {
  if ( a == b && b == c && c == d )
    return 1;
  else
    return 0;
}

...
// call site
char a, b, c, d;
int result = areEqual( a, b, c, d );
...
```

Lynn Ucks Hacker, C programmer extraordinaire, heard from a friend that using pointers to pass arguments during a function call (i.e., pass-by-reference) is faster than passing the value itself. Ms. Hacker is considering rewriting the above function `areEqual`, which is called very frequently in her program, to use pointers instead of passing `char`'s directly.

(a) Rewrite the code above to implement this new pointer-based interface for `areEqual`. Be sure to change the function as well as its call site.

(b) Assuming that this code runs on a 32-bit architecture, is this a good idea? Why or why not?

3. [15 points] **Performance**

Instruction Type	CPI
load/store	2
ALU	1
branch	1.5

Application	IPC	% Loads/Stores	% ALU ops	% Branches
A	.8			
B	.667			

(a) Given the IPC for each application *A* and *B*, and the CPI costs for each instruction type (in the tables above), give a percentage breakdown for each of the instruction types that constitute that application (Note: there are multiple correct answers). Ensure that your percentages sum to 100% for each application. Hint: $\text{CPI} = \frac{1}{\text{IPC}}$.

(b) Supposing that the cost of ALU operations is now completely free (i.e., CPI = 0 for ALU ops), what is the speedup for each application? Feel free to leave your answers as fractions.

4. [5 points] **Interrupts**

   Why is it a good idea to use interrupts, instead of polling, for disk-based I/O?

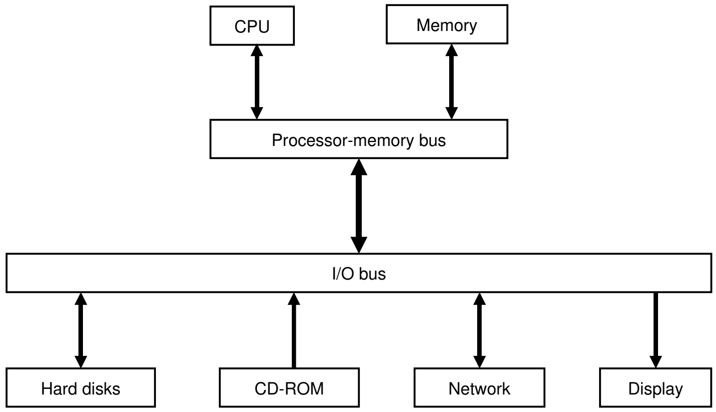
5. [5 points] **Compiler Optimization**

   Why might a compiler perform the following optimization?

```
// code before...
for ( j = 0; j < 20; j++)
  for ( i = 0; i < 200; i++)
    x[i][j] = x[i][j] + 1;

// ...code after
for ( i = 0; i < 200; i++)
  for ( j = 0; j < 20; j++)
    x[i][j] = x[i][j] + 1;
```

6. [10 points] **I/O**



(a) Explain why the buses in the figure above are split into a hierarchy.

(b) In modern PC's the display device interface is also on the Processor-Memory bus. Why is this?

(c) A CPU and memory share a 32-bit bus running at 200MHz. The memory needs 55ns to access a 128-bit value from one address. What is the effective bandwidth?

7. [10 points] **Caching**

The following C Program is run (with no optimization) on a processor with a cache that has 8-word (32-byte) blocks, and the cache holds a total of 256 bytes of data. A C `int` is 1 word in size. You **must** give your answers *as fractions*, but you can leave them unreduced.

```
int  i, j, c, stepsize, array[512];

for ( i = 0 ;  i < 100 ;  i++) {
  for ( j = 0 ;  j < 512 ; j=j + stepsize ) {
    c += array [j] + 17;
  }
}
```

(a) If we consider only the cache activity generated by references to the array, what is the miss rate when the cache is direct mapped and `stepsize = 256`?

(b) Again considering only the cache activity generated by references to the array, what is the miss rate when the cache is direct mapped but we change `stepsize` to `255`?

(c) Again considering only the cache activity generated by references to the array, what is the miss rate when the cache is *two-way set associative* and `stepsize = 256`?

8. [15 points] **Mystery Cache**

   Given the following sequence of memory requests and their responses from a mystery cache M, give a possible block size (in bytes), associativity, and total size (in bytes) for M. All addresses are *byte* addresses.

Address	Outcome
0	miss
1	hit
2	miss
3	hit
4	miss
2	hit
0	miss

9. [10 points] **Virtual Memory**

   Assume a hierarchical page table of two levels. Pages in this system are 4KB in size, and page table entries are 4B each. Assume there is exactly one 2nd-level page table P in the system, and P occupies exactly one page of physical memory. If exactly half of P's entries are valid, how many bytes of memory in our virtual address space actually reside in the physical memory? Do *not* include the space occupied by the page tables themselves in your answer.