CSE 30
Spring 2001
Midterm Exam

1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (13 points)
4. Bit Operations ___________________ (16 points)
5. Parameter Passing and Return Values ___________________ (10 points)
6. Local Variables, The Stack and Return Values ___________________ (14 points)

SubTotal ___________________ (80 points)
Extra Credit ___________________ (4 points)
Total ___________________
1. Number Systems

Convert $\text{FADE}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

- decimal: __________________________
- octal: 0__________________________
- binary: __________________________

Convert $-221_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- sign-magnitude: 0x_______________________________
- 1’s complement: 0x_____________________________
- 2’s complement: 0x_____________________________

Convert $+420_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- sign-magnitude: 0x_______________________________
- 1’s complement: 0x_____________________________
- 2’s complement: 0x_____________________________
2. Binary Addition/Condition Code Bits/Overflow Detection

Indicate what the condition code bits are when adding the following 8-bit 2’s complement numbers. (12 points)

\[
\begin{array}{ccc}
01010101 & +10101011 & \text{---------} \\
+01010100 & +00111011 & \text{---------} \\
10111001 & +10010110 & \text{---------}
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
N & Z & V & C \\
\hline
| & | & | & |
\hline
\end{array}
\]

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (9 points)

```
C

i = 44; ! i mapped to %l0
while ( i > 25 ) {
    statement1;
    statement2;
    i = i - 3;
}
```

For the following instruction sequence, mark with an X the conditional branch instruction which would transfer control to loop if used in place of ba. (0-4 points: +1 for each correct; -1 for each incorrect; 0 if all marked)

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>bl</th>
<th>bne</th>
<th>bpos</th>
<th>bvc</th>
<th>ble</th>
<th>bge</th>
<th>be</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td>2</td>
<td>%10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addcc</td>
<td>%10, 5, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Bit Operations

What is the value of %l0 after each statement is executed? Express your answers in hexadecimal.

set 0xB9463CAD, %l0
sra %l0, 10, %l0

Value in %l0 is 0x_____________________________________ (2 points)

set 0xB9463CAD, %l0
sll %l0, 11, %l0

Value in %l0 is 0x_____________________________________ (2 points)

set 0xB9463CAD, %l0
set 0x43CDDC65, %l1
xor %l0, %l1, %l0

Value in %l0 is 0x_____________________________________ (2 points)

void maskPassPhrase( unsigned long keys[], char passPhrase[], unsigned long mask[]);

In PA2, you wrote maskPassPhrase.s which performed an exclusive-or of keys[0] and the first 4 chars of passPhrase and stored the result in mask[0], and then performed an exclusive-or of keys[1] and the second 4 chars of passPhrase and stored the result in mask[1]. Write the body of maskPassPhrase.s in a way that requires only two loads, two exclusive-ors, and a single store instruction for a total of 5 instructions (not counting the save, ret, restore). Use only registers %l0-%l3 and %i0-%i2. Assume keys, passPhrase, and mask are all aligned on an evenly divisible by 8 byte boundary (this is a hint!). (10 points)

.global maskPassPhrase
.section " .text"
maskPassPhrase:
save %sp, -96, %sp

___________________________________________________________

___________________________________________________________

___________________________________________________________

___________________________________________________________

___________________________________________________________

ret
restore
5. Parameter Passing and Return Values

Write the equivalent unoptimized SPARC assembly language instructions to perform the following C code fragment. You can assume just this one local variable. (10 points)

C

/* Assume this local variable is declared appropriately */

struct fubar {
    int a;
    short b;
    char c;
    int d;
} fb; /* Local variable fb */

/* ... Other code ... */

/*
Write the code for just this function call saving the return value appropriately */

fb.b = foo(fb.a, fb.c);

SPARC assembly
6. Local Variables, The Stack, and Return Values

Here is a C function that doesn’t do much but allocate local variables, perform statements, and returns a value:

```c
int fubar( int x, int y ) {
    long *local_stack_var1;
    long    local_stack_var2[7];

    local_stack_var1 = &local_stack_var2[2]; /* statement 1 */
    *local_stack_var1 = 420420;               /* statement 2 */
    x = *local_stack_var1;                    /* statement 3 */
    local_stack_var1++;                       /* statement 4 */
    return ( local_stack_var2[5] - y );       /* statement 5 */
}
```

Now write the equivalent unoptimized SPARC assembly language instructions to perform the equivalent. You must allocate all local variables on the Stack. Perform each instruction literally. No short-cuts. Draw a line between groups of instructions to indicate which instructions are associated with each C statement. (14 points)

```sparc
.global fubar

.section " .text"

fubar: /* Your unoptimized code goes below this point */
```
Extra Credit

Optimize the following SPARC assembly language code fragment to eliminate any delay slots. (4 points)

<table>
<thead>
<tr>
<th>Unoptimized Version</th>
<th>Optimized Version</th>
</tr>
</thead>
</table>

/* some code above here */

    ba   test
    nop

loop:
    inc  %l1
    add  %l1, %l0, %l0

test:
    cmp  %l1, 53
    bge  loop
    nop

/*
 * other code that may
 * access %l0 or %l1
 */

Note: Only these instructions listed as part of the code fragment can be used to fill any delay slots.
Scratch Paper
Scratch Paper