CSE 30
Summer 2000
Midterm Exam

1. Number Systems __________________________________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ____________ (12 points)
3. Branching ____________________________________________ (13 points)
4. Bit Operations ________________________________________ (10 points)
5. Parameter Passing and Return Values ________________________ (12 points)
6. Local Variables, The Stack, and Return Values ________________ (18 points)

SubTotal ______________________________________________ (80 points)
Extra Credit ____________________________________________ (4 points)
Total __________________________________________________
1. Number Systems

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

decimal ____________________________

octal 0____________________________

binary ____________________________

Convert $-17_{10}$ 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 $+142_{10}$ 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)

\[
\begin{array}{ccc}
01011101 +00101011 & 01010100 +10111011 & 10111001 +10010110 \\
--------- & --------- & ---------
\end{array}
\]

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

3. Branching

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

\[
\text{C}_1
\]

\[
\text{SPARC assembly}
\]

\[
\text{for} (\ i = 5; \ i \leq 15; \ ++i) \ \{ \text{statement1;} \text{statement2;} \}
\]

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>be</th>
<th>bne</th>
<th>bl</th>
<th>bleu</th>
<th>ble</th>
<th>bge</th>
<th>bneg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov 2, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subcc %10, 5, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td></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.**

\[
\begin{align*}
\text{set} & \quad 0xBEADFDAD, \%l0 \\
\text{set} & \quad 0xB9653, \%l1 \\
\text{or} & \quad \%l0, \%l1, \%l0 \\
\text{Value in } \%l0 \text{ is } & \quad 0x\underline{\ldots} \quad \text{(2 points)}
\end{align*}
\]

\[
\begin{align*}
\text{set} & \quad 0xBEADFDAD, \%l0 \\
\text{sll} & \quad \%l0, 11, \%l0 \\
\text{Value in } \%l0 \text{ is } & \quad 0x\underline{\ldots} \quad \text{(2 points)}
\end{align*}
\]

\[
\begin{align*}
\text{set} & \quad 0xBEADFDAD, \%l0 \\
\text{sra} & \quad \%l0, 5, \%l0 \\
\text{Value in } \%l0 \text{ is } & \quad 0x\underline{\ldots} \quad \text{(2 points)}
\end{align*}
\]

\[
\begin{align*}
\text{set} & \quad 0xBEADFDAD, \%l0 \\
\text{set} & \quad 0xB65, \%l1 \\
\text{xor} & \quad \%l0, \%l1, \%l0 \\
\text{Value in } \%l0 \text{ is } & \quad 0x\underline{\ldots} \quad \text{(2 points)}
\end{align*}
\]

Write the **single** SPARC assembly instruction to **multiply** the value in %l0 by 1024 without the use of .mul. Store the result back in %l0. (2 points)

\[
\underline{\ldots}
\]
5. Parameter Passing and Return Values

Write the equivalent unoptimized SPARC assembly language instructions to perform the following C code fragment. (10 points)

C

```c
x = foo( 172, -9876 );
```

SPARC assembly

```asm
/* x is mapped to %10 */
```

Now optimize your answer to eliminate any delay slots. (2 points)

Optimized version of above SPARC assembly

```asm
```


6. Local Variables, The Stack, and Return Values

Here is a C function that doesn’t do much but allocate local variables, perform some assignments, and returns the difference of the params:

```c
int foo( int i, int j ) {
    short local_stack_var1;
    int local_stack_var2;
    local_stack_var1 = 6;
    local_stack_var2 = 10;
    i = local_stack_var1;
    return ( i - j );
}
```

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. (18 points)

```sparc
.global foo
.section " .text"

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

Optimize the following SPARC assembly language code fragment. There may be many ways of optimizing this code fragment, but some optimizations may be better than others. The better optimizations eliminate as many cycles as possible. The better optimizations get more points. (4 points)

<table>
<thead>
<tr>
<th>Unoptimized Version</th>
<th>Optimized Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* some code above here */</td>
<td></td>
</tr>
<tr>
<td>mov %10, %o0</td>
<td></td>
</tr>
<tr>
<td>mov 16, %o1</td>
<td></td>
</tr>
<tr>
<td>call .mul</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>mov %o0, %l1</td>
<td></td>
</tr>
<tr>
<td>add %fp, array_offset, %l2</td>
<td></td>
</tr>
<tr>
<td>ldup [%l2 + %l1], %l3</td>
<td></td>
</tr>
<tr>
<td>cmp %l3, 'n'</td>
<td></td>
</tr>
<tr>
<td>be no_inc</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>inc %l0</td>
<td></td>
</tr>
<tr>
<td>no_inc:</td>
<td></td>
</tr>
<tr>
<td>/* other code */</td>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper