Name _________________________
Signature_______________________
cs30x_______

CSE 30
Winter 2002
Midterm Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (17 points)
4. Bit Operations / C Runtime Environment ___________________  (16 points)
5. Parameter Passing and Return Values (Structures) ___________________  (10 points)
6. Local Variables, The Stack, and Return Values ___________________  (14 points)

SubTotal ___________________  (82 points)
Extra Credit ___________________  (4 points)

Total ___________________
1. Number Systems

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

<table>
<thead>
<tr>
<th>Base</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimal</td>
<td>__________________________</td>
</tr>
<tr>
<td>octal</td>
<td>0________________________</td>
</tr>
<tr>
<td>binary</td>
<td>__________________________</td>
</tr>
</tbody>
</table>

Convert $+514_{10}$ to the following (assume 16-bit word). \textbf{Express answers in hexadecimal.} (3 points)

<table>
<thead>
<tr>
<th>Base</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign-magnitude</td>
<td>0x________________________</td>
</tr>
<tr>
<td>1(^\circ)s complement</td>
<td>0x________________________</td>
</tr>
<tr>
<td>2(^\circ)s complement</td>
<td>0x________________________</td>
</tr>
</tbody>
</table>

Convert $-347_{10}$ to the following (assume 16-bit word). \textbf{Express answers in hexadecimal.} (6 points)

<table>
<thead>
<tr>
<th>Base</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign-magnitude</td>
<td>0x________________________</td>
</tr>
<tr>
<td>1(^\circ)s complement</td>
<td>0x________________________</td>
</tr>
<tr>
<td>2(^\circ)s complement</td>
<td>0x________________________</td>
</tr>
</tbody>
</table>
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)

\[
\begin{array}{c}
01010101 \\
+10101011 \\
\hline
\end{array}
\quad
\begin{array}{c}
11010101 \\
+00111001 \\
\hline
\end{array}
\quad
\begin{array}{c}
10111001 \\
+10010110 \\
\hline
\end{array}
\]

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

3. Branching (15 points)

Fill in the SPARC assembly instructions to perform the following statements. **Do not optimize**, (17 points)

\[
\begin{array}{c}
\text{C} \\
\text{SPARC assembly} \\
\end{array}
\]

\[
\text{int } x = 41; \quad ! x \text{ mapped to } %10
\]

\[
\text{do } \{
\text{statementA;}
\text{if ( } x <= 9 \text{ ) } \{
\quad --x;
\quad \text{statementB;}
\} \text{ else }
\quad \text{statementC;}
\quad x += -5;
\} \text{ while ( } x < -8 \text{ );}
\]
4. Bit Operations / C Runtime Environment

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

```
set 0xAB46C59E, %l0
sll %l0, 13, %l0
```

Value in %l0 is 0x______________________________ (2 points)

```
set 0xAB46C59E, %l0
sra %l0, 9, %l0
```

Value in %l0 is 0x______________________________ (2 points)

```
set 0xAB46C59E, %l0
set 0xCDCAB072, %l1
and %l0, %l1, %l0
```

Value in %l0 is 0x______________________________ (2 points)

Fill in the names of the 5 areas of the C Runtime Environment as layed out by most Unix operating systems. Then state what parts of a C program are in each area. (10 points)

low memory

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

high memory

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

fb.d = foo( fb.a, fb.b );
```
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[6];
    long *local_stack_var2;
    local_stack_var2 = local_stack_var1 + 2; /* statement 1 */
    x = local_stack_var1[4]; /* statement 2 */
    *local_stack_var2 = 8675309; /* statement 3 */
    local_stack_var2--; /* statement 4 */
    return ( y + local_stack_var1[1] ); /* 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 assembly
.global fubar
.section " .text"

fubar: /* Your unoptimized code goes below this point */
```
Extra Credit (4 points)
In PA2 we performed 64-bit rotates across two 32-bit registers. We can perform other operations (like addition) in a similar manner. Fill in the blanks lines to perform a 64-bit addition across two 32-bit registers to implement adding the 64 bits starting at address a to the 64 bits starting at address b and store the results in the 64 bits starting at address c. You can ignore any overflow; do not optimize. The high-order 32 bits are denoted in the comments with a 0, as in a0 or b0 or c0 while the low-order 32 bits are denoted in the comments with a 1, as in a1 or b1 or c1.

```plaintext
.global main

.section ".data"
fmt: .asciz "%x %x\n"
.align 4
a: .word 0x12345678, 0xFFFFFFFF
b: .word 0,1
c: .word 0,0

.section ".text"
main:
save %sp, -96, %sp ! a -> %16
________________ ! a0 -> %10 (upper 32 bits)
________________ ! a1 -> %11 (lower 32 bits)
________________ ! b -> %17
________________ ! b0 -> %12 (upper 32 bits)
________________ ! b1 -> %13 (lower 32 bits)
________________ ! Add lower 32 bits -> %15
________________ ! If there was no Carry from this add,
________________ !   skip adding Carry to upper 32 bits
________________ ! Add Carry (1) to upper 32 bits of a0
no_carry:
________________ ! Add upper 32 bits -> %14
________________ ! c -> %16
________________ ! %14 -> c0 (upper 32 bits)
________________ ! %15 -> c1 (lower 32 bits)
ret
restore
```
Scratch Paper
Scratch Paper