Name _________________________
Signature_______________________

cs30x_____

CSE 30
Fall 2002
Midterm Exam

1. Number Systems ___________________  (15 points)

2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)

3. Branching ___________________  (20 points)

4. Bit Operations / C Runtime Environment ___________________  (17 points)

5. Parameter Passing and Return Values (Structures) ___________________  (12 points)

6. Local Variables, The Stack and Return Values ___________________  (15 points)

7. Load/Store/Memory ___________________  (9 points)

SubTotal ___________________  (100 points)

Extra Credit ___________________  (5 points)

Total ___________________
1. Number Systems

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

<table>
<thead>
<tr>
<th>System</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>___________________________</td>
</tr>
<tr>
<td>Octal</td>
<td>0__________________________</td>
</tr>
<tr>
<td>Decimal</td>
<td>___________________________</td>
</tr>
</tbody>
</table>

Convert $+242_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

<table>
<thead>
<tr>
<th>System</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign-magnitude</td>
<td>0x_________________________</td>
</tr>
<tr>
<td>1’s complement</td>
<td>0x_________________________</td>
</tr>
<tr>
<td>2’s complement</td>
<td>0x_________________________</td>
</tr>
</tbody>
</table>

Convert $-439_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

<table>
<thead>
<tr>
<th>System</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign-magnitude</td>
<td>0x_________________________</td>
</tr>
<tr>
<td>1’s complement</td>
<td>0x_________________________</td>
</tr>
<tr>
<td>2’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)

\[
\begin{array}{ccc}
11010011 & +10101111 & \text{--------} \\
10010101 & +01111001 & \text{--------} \\
00111000 & +01010111 & \text{--------}
\end{array}
\]

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (20 points)

```c
for ( x = 55; x <= 420; x += 5 ) {
    y = x;
    if ( x <= 99 ) {
        y = x - 15;
        --x;
    } else
        x = y + 5;
}
```

SPARC assembly

```c
! x mapped to $l4
! y mapped to $l2
```
4. Bit Operations / C Runtime Environment

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

```c
set  0xC6943B7A, %10
sll  %10, 9, %10

Value in %l0 is 0x_______________________________  (2 points)

set  0xC6943B7A, %10
sra  %10, 13, %10

Value in %l0 is 0x_______________________________  (2 points)

set  0xC6943B7A, %10
set  __________, %l1
xor  %l0, %l1, %l0  ! Value in %l0 is now 0xCAFEBABE

Value set in %l1 must be this bit pattern 0x_______________________________  (3 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)

<table>
<thead>
<tr>
<th>Low memory</th>
<th>_________________</th>
<th>_________________</th>
<th>_________________</th>
<th>_________________</th>
<th>High memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Parameter Passing and Return Values (Structures)

Write the equivalent **unoptimized** SPARC assembly language instructions to perform the following C code fragment. You can assume just this one local variable. (12 points)

<table>
<thead>
<tr>
<th>C</th>
<th>SPARC assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Function Prototype */</td>
<td></td>
</tr>
<tr>
<td>short foo( short, long, char );</td>
<td></td>
</tr>
<tr>
<td>/* ... Other code ... */</td>
<td></td>
</tr>
<tr>
<td>/* Assume this local variable is declared appropriately and is the only local var. */</td>
<td></td>
</tr>
<tr>
<td>struct fubar {</td>
<td></td>
</tr>
<tr>
<td>char a;</td>
<td></td>
</tr>
<tr>
<td>char b;</td>
<td></td>
</tr>
<tr>
<td>short c;</td>
<td></td>
</tr>
<tr>
<td>long d;</td>
<td></td>
</tr>
<tr>
<td>short e;</td>
<td></td>
</tr>
<tr>
<td>int f;</td>
<td></td>
</tr>
<tr>
<td>} fb;  /* Local variable fb */</td>
<td></td>
</tr>
<tr>
<td>/* ... Other code ... */</td>
<td></td>
</tr>
<tr>
<td>/* Write the code for just this function call saving the return value appropriately */</td>
<td></td>
</tr>
<tr>
<td>fb.e = foo( fb.c, fb.d, fb.b );</td>
<td></td>
</tr>
</tbody>
</table>
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[3];
    y = local_stack_var2[1];                  /* statement 1 */
    local_stack_var1 = local_stack_var2 + 2;  /* statement 2 */
    *local_stack_var1 = 420024;               /* statement 3 */
    --local_stack_var1;                       /* statement 4 */
    return ( x + local_stack_var2[2] );       /* 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. (15 points)

**SPARC assembly**

```assembly
.global fubar
.section ".text"
fubar: /* Your unoptimized code goes below this point */
```

What gets printed in the following program? (9 points)

.load/store/memory

.global main

.section ".data"
fmt: .asciz "0x%x\n" ! prints value as hex 0XXXXXXXX

c: .byte 0x44

.align 2
s: .half 0x5678

.align 4
i1: .word 0x23456789
i2: .word 0x23456789
i3: .word 0x23456789

x: .word 0

.section ".text"
main:
save %sp, -96, %sp
set i1, %l0
set s, %l1
ldsh [%l1], %l1
sth %l1, [%l0]
set fmt, %o0
ld [%l0], %o1
call printf _______________
nop
set i2, %l0
set c, %l1
ldub [%l1], %l1
stb %l1, [%l0+2]
set fmt, %o0
ld [%l0], %o1
call printf _______________
nop
set i3, %l0
ldsh [%l0+2], %l1
sth %l1, [%l0]
set fmt, %o0
ld [%l0], %o1
call printf _______________
ret
restore
Extra Credit (5 points)

Translate the following C code fragment into optimized SPARC Assembly. Some optimizations are better/more complicated than others and are therefore worth more points.

```c
for ( b = a; b >= 0; --b ) ! a mapped to %l2
{ ! b mapped to %l4
  a = b % 16; /* Other code that may use a and b */
}
```
Scratch Paper