Name _________________________
Student ID __________________ Signature____________________
cs30x_____

CSE 30
Fall 2003
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 $\text{FA9A}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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 $321_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

<table>
<thead>
<tr>
<th>Representation</th>
<th>16-Bit Value</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 $-422_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (6 points)

<table>
<thead>
<tr>
<th>Representation</th>
<th>16-Bit Value</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)

\[
\begin{array}{ccc}
11010111 & +10101001 & 01111011 + 00010100 \\
+ & + & + \\
\hline
\end{array}
\]

<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
Write the SPARC assembly instructions to complete the following. **Do not optimize nops.** (20 points)

```
/* Return index of n in a; -1 if not found. */
int findIt( int n, int a[], int size ) {
    int i;

    for ( i = 0; i < size; ++i ) {
        if ( *a == n )
            return i;
        ++a;
    }

    return -1;
}
```
4. Bit Operations / C Runtime Environment

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

set 0xAB4DE95C, %l0
sra %l0, 7, %l0

Value in %l0 is 0x_________________________________________ (2 points)

set 0xAB4DE95C, %l0
sll %l0, 10, %l0

Value in %l0 is 0x_________________________________________ (2 points)

set 0xAB4DE95C, %l0
set 0x?????????, %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 the SPARC architecture. 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. (12 points)

```
C

/* Function Prototype */
short foo( char, long, char );

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

/* Assume this local variable fb
   is declared appropriately
   and is the only local variable. */

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

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

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

fb.d = foo( fb.b, fb.c, fb.e );
```

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

```c
int fubar( int a, int b ) {
    char  local_stack_var1[4];
    char *local_stack_var2;

    local_stack_var2 = local_stack_var1 + 1;    /* statement 1 */
    *local_stack_var2 = '5';                     /* statement 2 */
    --local_stack_var2;                          /* statement 3 */
    local_stack_var1[2] = b + '0';               /* statement 4 */
    return ( a + (local_stack_var1[3] - '0') );  /* 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 */

! You may use the fact that ’0’ = 0x30 in the ASCII char. set
7. Load/Store/Memory
What gets printed in the following program? (9 points)

.global main

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

c: .byte 0xBB
.align 2
s: .half 0x87AC
.align 4
i1: .word 0x12345678
i2: .word 0x12345678
i3: .word 0x12345678
x: .word 0

.section ".text"
main:
save %sp, -96, %sp
set i1, %l0
set c, %l1
ldsb [%l1], %l1
sth %l1, [%l0+2]
set fmt, %o0
ld [%l0], %o1
call printf _________________________________
nop
set i2, %l0
set s, %l1
ldub [%l1+1], %l1
stb %l1, [%l0+1]
set fmt, %o0
ld [%l0], %o1
call printf _________________________________
nop
set i3, %l0
set x, %l1
ldub [%l0+3], %l2
stb %l2, [%l1]
ldsh [%l0], %l2
sth %l2, [%l1+2]
mov %l1, %l0
set fmt, %o0
ld [%l0], %o1
call printf _________________________________
nop
ret
restore
**Extra Credit** (5 points)
Consider the following function.

```c
void enigma( int n )
{
    if ( (n & 0x1) != 0 )
        printf( "Yes" );
    else
        printf( "No" );
}
```

What gets printed if the function is called as `enigma( 37 )`? (2 pts)

Optimize your solution to #3 **Branching** (the `findIt()` function) to eliminate nops. (3 pts)
Scratch Paper