CSE 30
Winter 2003
Midterm Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (22 points)
4. Bit Operations / C Runtime Environment ___________________  (17 points)
5. Parameter Passing and Return Values (Structures) ___________________  (10 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 $\text{EDBA}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Value</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 $+434_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- **Sign-magnitude:** $0x$__________________________
- **1’s complement:** $0x$__________________________
- **2’s complement:** $0x$__________________________

Convert $-295_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- **Sign-magnitude:** $0x$__________________________
- **1’s complement:** $0x$__________________________
- **2’s complement:** $0x$__________________________
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)

\[
\begin{array}{cccc}
10111001 & +10010110 & = 11101111 & N \ Z \ V \ C = 1 \ 0 \ 1 \ 0 \\
01010000 & +00111011 & = 10101011 & N \ Z \ V \ C = 0 \ 0 \ 0 \ 1 \\
01011011 & +10101011 & = 11110110 & N \ Z \ V \ C = 0 \ 0 \ 1 \ 1 \\
\end{array}
\]

3. Branching
Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (22 points)
Do not change the algorithm. Just a direct translation.

```c
for ( x = 5555; x >= 420; x -= 75 ) {
    if ( x < 939 ) {
        y = x - 15;
        --x;
    } else {
        x = y % 59;
    }
}
```

```sparc
! x mapped to %13
% % % % %

! y mapped to %15
% % % % %
```
4. Bit Operations / C Runtime Environment

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

set 0xCABAFEBD, %l0
sra %l0, 9, %l0

Value in %l0 is 0x_______________________________ (2 points)

set 0xCABAFEBD, %l0
sll %l0, 13, %l0

Value in %l0 is 0x_______________________________ (2 points)

set 0xCABAFEBD, %l0
set __________, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now 0xDEADBEEF

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)

| low memory |
| __________________________ |
| __________________________ |
| __________________________ |
| __________________________ |
| __________________________ |

| high memory |
| __________________________ |
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. (10 points)

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

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

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

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

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

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

fb.a = foo( fb.f, fb.d, 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 ) {
    int local_stack_var1[4];
    int *local_stack_var2;
    local_stack_var1[0] = y;                  /* statement 1 */
    local_stack_var2 = local_stack_var1 + 3;  /* statement 2 */
    *local_stack_var2 = 411911;               /* statement 3 */
    local_stack_var2++;
    return ( x + local_stack_var1[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
.global  fubar
.section  " .text"
fubar:    /* Your unoptimized code goes below this point */
```
7. Load/Store/Memory

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

```assembly
.global main

.section ".data"
fmt: .asciz "0x%\n"       ! hex 0xXXXXXXX

.c:      .byte 0xFF
.s:      .half 0xABCD
.i1:     .word 0x23456789
.i2:     .word 0x23456789
.i3:     .word 0x23456789

.section ".text"
main:
save    %sp, -96, %sp
set     i1, %l0
set     s, %l1
ldsh    [%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]
set     fmt, %o0
ld      [%l0], %o1
call    printf _________________________________
nop
set     i3, %l0
ldsh    [%l0], %l1
sth    %l1, [%l0+2]
set     fmt, %o0
ld      [%l0], %o1
call    printf _________________________________

ret
restore
```
Extra Credit (5 points)

Take the SPARC assembly code you wrote for #3 Branching (page 3) and optimize the code to eliminate nop instructions. Do not change the algorithm.
Scratch Paper