CSE 30  
Fall 2005  
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 \(0xFA21\) (2’s complement, 16-bit word) to the following. (6 points)

- **binary**
  

- **octal**

- **decimal**

Convert -326 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 +435 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)

\[
\begin{array}{ccc}
01010111 & +00101001 & \hline
11010110 & +10111001 & \hline
10111011 & +00010100 & \hline
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C \\
\hline
 & | & | & | \\
\hline
 & | & | & | \\
\hline
 & | & | & | \\
\end{array}
\]
3. **Branching** (20 points)
Write the SPARC assembly instructions to complete the following. **Do not optimize nops**, (20 points)

```c
int
fubar( int value, int decAmt, int *head, int *tail )
{
    int count = 0;

    while ( head != tail )
    {
        count++;
        *head++ = value; /* *head = value; head++; */
        value = value - decAmt;
    }

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

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

- `set 0x98ACED76, %l0`
- `sra %l0, 13, %l0`
  
  Value in %l0 is 0x______________________________ (2 points)

- `set 0x98ACED76, %l0`
- `sll %l0, 11, %l0`
  
  Value in %l0 is 0x______________________________ (2 points)

- `set 0x98ACED76, %l0`
- `set 0x?????????, %l1`
- `xor %l0, %l1, %l0` ! **Value in %l0 is now OxCAFEBABE**
  
  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 laid out by the SPARC architecture. Then state what parts of a C program are in each area. (10 points)

<table>
<thead>
<tr>
<th>low memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<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)

```
C

/* Function Prototype */

short foo( char, unsigned short, char );

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

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

struct fubar {
    char           a;
    int            b;
    short          c[3];
    char           d[3];
    unsigned short e;
} fb;   /* Local variable fb */

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

/

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

fb.c[1] = foo( fb.a, fb.e, fb.d[2] );
```

SPARC assembly
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( short a, char b ) {
    int *local_stack_var1;
    int local_stack_var2[4];

    local_stack_var2[3] = b - 123; /* statement 1 */
    local_stack_var1 = &local_stack_var2[2]; /* statement 2 */
    local_stack_var1++; /* statement 3 */
    *(local_stack_var1 + 2) = 321; /* statement 4 */
    return ( a + (local_stack_var2[1] + 420) ); /* 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
.globl fubar
.sect .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%X\n"    ! prints value as hex 0xXXXXXXXX
 c:      .byte 0xDD
 .align 2
 s:      .half 0x2345
 .align 4
 i1:     .word 0x9876ACED
 i2:     .word 0x9876ACED
 i3:     .word 0x9876ACED
 x:      .word 0

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

Optimize the following SPARC Assembly code fragment. You can assume there are other instructions above and below this code fragment, but only optimize using the instructions given in this code fragment. Some optimizations may be worth more than others.

<table>
<thead>
<tr>
<th>Unoptimized SPARC Assembly</th>
<th>Optimized SPARC Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Other code you cannot use */</td>
<td></td>
</tr>
<tr>
<td>cmp %i2, %l5</td>
<td></td>
</tr>
<tr>
<td>bg end_loop</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td></td>
</tr>
<tr>
<td>add %i2, %i0, %i2</td>
<td></td>
</tr>
<tr>
<td>mov 32, %o0</td>
<td></td>
</tr>
<tr>
<td>mov %i2, %o1</td>
<td></td>
</tr>
<tr>
<td>call .mul</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>mov %o0, %i2</td>
<td></td>
</tr>
<tr>
<td>cmp %i2, %l5</td>
<td></td>
</tr>
<tr>
<td>ble loop</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>end_loop:</td>
<td></td>
</tr>
<tr>
<td>inc %i2</td>
<td></td>
</tr>
<tr>
<td>/* Other code you cannot use */</td>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper
Scratch Paper