CSE 30  
Winter 2007  
Midterm Exam

1. Number Systems  ___________________  (12 points)
2. Binary Addition/Condition Code Bits/Overflow Detection  ___________________  (8 points)
3. Branching  ___________________  (10 points)
4. Bit Operations / C Runtime Environment  ___________________  (7 points)
5. Parameter Passing and Return Values (Structures)  ___________________  (12 points)
6. Local Variables, The Stack, and Return Values  ___________________  (10 points)
7. Load/Store/Memory  ___________________  (3 points)

SubTotal  ___________________  (62 points)

Extra Credit  ___________________  (3 points)

Total  ___________________
1. Number Systems

Convert 0xFA1B (2’s complement, 16-bit word) to the following. (6 points)

binary ________________________________

octal 0______________________________

decimal ______________________________

Convert -337 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. (8 points)

\[
\begin{array}{c}
11000101 +00111011 \\
\hline
\end{array}
\quad
\begin{array}{c}
01111111 +00000001 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C \\
\hline
 & & & \\
\hline
 & & & \\
\end{array}
\quad
\begin{array}{cccc}
N & Z & V & C \\
\hline
 & & & \\
\hline
 & & & \\
\end{array}
\]
3. **Branching** (10 points)

Translate the SPARC Assembly function below into the equivalent C code. Just perform a direct translation. The C function definition is started for you indicating the names of formal parameters and local variables (and their location on the stack). **Do not gotos – use only standard looping and conditional statements.**

<table>
<thead>
<tr>
<th><strong>SPARC ASSEMBLY</strong></th>
<th><strong>C</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>.section &quot;.text&quot;</td>
<td>int baz( int a, int b )</td>
</tr>
<tr>
<td>baz:</td>
<td>}</td>
</tr>
<tr>
<td>save %sp, -96, %sp</td>
<td></td>
</tr>
<tr>
<td>cmp %i0, %i1</td>
<td></td>
</tr>
<tr>
<td>bge L1</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>sub %i0, 5, %i0</td>
<td></td>
</tr>
<tr>
<td>ba L2</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>L1:</td>
<td></td>
</tr>
<tr>
<td>sub %i1, 10, %i1</td>
<td></td>
</tr>
<tr>
<td>L2:</td>
<td></td>
</tr>
<tr>
<td>add %i0, %i1, %i0</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
<tr>
<td>restore</td>
<td></td>
</tr>
</tbody>
</table>


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

1. set 0xACDCFACE, %l0
   sra %l0, 7, %l0
   Value in %l0 is 0x__________________________ (2 points)

2. set 0xACDCFACE, %l0
   sll %l0, 7, %l0
   Value in %l0 is 0x__________________________ (2 points)

3. set 0xACDCFACE, %l0
   set 0x?????????, %l1
   xor %l0, %l1, %l0
   ! Value in %l0 is now 0xFACEACDC
   Value set in %l1 must be this bit pattern 0x__________________________ (3 points)
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 */
char foo( char, unsigned short, int );
/* ... Other code ... */
/* Assume this local variable is declared appropriately and is the only local var. */
struct fubar {
    char a;
    unsigned short b;
    char c[3];
    int d;
} 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.b, fb.d );
```

SPARC assembly

```
/* Function Prototype */
char foo( char, unsigned short, int );
/* ... Other code ... */
/* Assume this local variable is declared appropriately and is the only local var. */
struct fubar {
    char a;
    unsigned short b;
    char c[3];
    int d;
} 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.b, fb.d );
```
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[2];
    int *local_stack_var2;

    x = *local_stack_var2++;       /* statement 1 */
    local_stack_var2 = &local_stack_var1[1];   /* statement 2 */
    return ( y - local_stack_var1[0] );        /* statement 3 */
}
```

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. (10 points)

```
SPARC assembly
.global fubar
.section " .text"

fubar: /* Your unoptimized code goes below this point */
```
7. Load/Store/Memory
What gets printed in the following program? (3 points)

```assembly
.global main

.section ".data"
fmt: .asciz "0x%X\n"  ! prints value as hex  0x00000000X
c: .byte 0xDD
.s: .half 0x89AB
.i1: .word 0x87654321
.i2: .word 0x87654321
.i3: .word 0x87654321
.x: .word 0

.section ".text"
main:
save  %sp, -96, %sp
set   i3, %l0
set   x, %l1
ldsh  [%l0], %l2
sth   %l2, [%l1+2]
ldub  [%l0+2], %l2
stb   %l2, [%l1]
ldub  [%l0+3], %l2
stb   %l2, [%l1+1]
mov   %l1, %l0
set   fmt, %o0
ld    [%l0], %o1
call  printf
nop
ret
restore
```
Extra Credit (3 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>baz:</td>
<td></td>
</tr>
<tr>
<td>save %sp, -96, %sp</td>
<td></td>
</tr>
<tr>
<td>cmp %i0, %i1</td>
<td></td>
</tr>
<tr>
<td>bge L1</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>sub %i0, 5, %i0</td>
<td></td>
</tr>
<tr>
<td>ba L2</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>L1:</td>
<td></td>
</tr>
<tr>
<td>sub %i1, 10, %i1</td>
<td></td>
</tr>
<tr>
<td>L2:</td>
<td></td>
</tr>
<tr>
<td>add %i0, %i1, %i0</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
<tr>
<td>restore</td>
<td></td>
</tr>
<tr>
<td>/* Other code you cannot use */</td>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper
Scratch Paper