CSE 30
Fall 2006
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 0xF939 (2’s complement, 16-bit word) to the following. (6 points)

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

Convert -328 to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

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

Convert +477 to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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)

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

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

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Branching (20 points)
Translate the C function below into the equivalent SPARC Assembly code. Just perform a direct translation –
no optimizations. The assembly function is started for you. **Do not gotos – only use standard looping and conditional statements.**

**C**

```c
int fubar( int uno, int duo )
{
    if ( uno > duo )
    {
        while ( duo < 57 )
        {
            uno = duo - 10;
            ++duo;
        }
    }
    else
    {
        duo = uno + duo;
    }

    return duo - uno;
}
```

**SPARC ASSEMBLY**

```assembly
.globa1 fubar
.secti0n ".text"

fubar:
    save %sp, -96, %sp
```
4. Bit Operations / C Runtime Environment

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

set 0x9D9E8765, %l0
sra %l0, 9, %l0

Value in %l0 is 0x_________________________ (2 points)

set 0x9D9E8765, %l0
sll %l0, 14, %l0

Value in %l0 is 0x_________________________ (2 points)

set 0x9D9E8765, %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 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>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
\text{C} & \\
/* \text{Function Prototype} */ \hfill \text{SPARC assembly} \\
\text{int} & \text{ foo( unsigned short, int, char );} \\
/* \ldots \text{Other code} \ldots */ \\
/* \text{Assume this local variable} \\
\text{is declared appropriately} \\
\text{and is the only local var.} */ \\
\text{struct fubar} \{ \\
& \text{int a[2];} \\
& \text{char } b; \\
& \text{unsigned short c;} \\
\} \text{ fb; } /* \text{Local variable fb} */ \\
/* \ldots \text{Other code} \ldots */ \\
\end{align*}
\]

\[
\begin{align*}
/* \text{Write the code for just this} \\
\text{function call saving the} \\
\text{return value appropriately} */ \\
\text{fb.a[0] = foo( fb.c, fb.a[1], fb.b );} \\
\end{align*}
\]

Put your SPARC Assembly code in the box below.
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
.global fubar
.sect .text
fubar: /* Your unoptimized code goes below this point */
```
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 0xEE

.align 2
s: .half 0x8ABC

.align 4
i1: .word 0x24681357
i2: .word 0x24681357
i3: .word 0x24681357
x: .word 0

.section ".text"
main:
save  %sp, -96, %sp

set   i1, %l0
set   c, %l1
ldsb  [%l1], %l1
sth   %l1, [%l0]
set   fmt, %o0
ld    [%l0], %o1
call  printf _________________________________
nop

set   i2, %l0

set   x, %l1
ldub  [%l0+1], %l2
stb   %l2, [%l1+2]
ldsh  [%l0+2], %l2
stb   %l2, [%l1]
mov   %l1, %l0

set   fmt, %o0
ld    [%l0], %o1
call  printf _________________________________
nop

set   i3, %l0

set   s, %l1
ldub  [%l1], %l2
stb   %l2, [%l0+1]
ldub  [%l1+1], %l2
stb   %l2, [%l0+2]

set   fmt, %o0
ld    [%l0], %o1
call  printf _________________________________
nop

ret
restore
```
**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.

**Unoptimized SPARC Assembly**

```assembly
/* Other code you cannot use */

baz:
    save %sp, -96, %sp
    mov %i0, %l0
    set 5678, %l1
    cmp %l1, %i1
    bge L1
    nop
L2:
    sll %i0, 3, %l1
    cmp %l0, %l1
    be L3
    nop
    add %l0, %l1, %l0
    ba L4
    nop
L3:
    sub %i0, %i1, %l0
    xor %l0, 0xFF, %l0
L4:
    cmp %l1, %i1
    bl L2
    nop
L1:
/* Other code you cannot use */
```

**Optimized SPARC Assembly**

```assembly
/* Other code you cannot use */

baz:
    mov %i0, %l0
    set 5678, %l1
    cmp %l1, %i1
    bge L1
    nop
L2:
    sll %i0, 3, %l1
    cmp %l0, %l1
    be L3
    nop
    add %l0, %l1, %l0
    ba L4
    nop
L3:
    sub %i0, %i1, %l0
    xor %l0, 0xFF, %l0
L4:
    cmp %l1, %i1
    bl L2
    nop
L1:
/* Other code you cannot use */
```