1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (13 points)
4. Bit Operations ___________________ (18 points)
5. Parameter Passing and Return Values ___________________ (10 points)
6. Local Variables, The Stack, and Return Values ___________________ (18 points)

SubTotal ___________________ (86 points)
Extra Credit ___________________ (4 points)
Total ___________________
1. Number Systems

Convert $\text{DEAD}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

<table>
<thead>
<tr>
<th>Number System</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>____________________</td>
</tr>
<tr>
<td>Octal</td>
<td>0 ____________________</td>
</tr>
<tr>
<td>Binary</td>
<td>____________________</td>
</tr>
</tbody>
</table>

Convert $-151_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

<table>
<thead>
<tr>
<th>Number System</th>
<th>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 $+146_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

<table>
<thead>
<tr>
<th>Number System</th>
<th>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)

\[
\begin{array}{ccc}
01011101 & +00101011 & = 01010000 \\
01010100 & +10111011 & = 11000011 \\
10111001 & +10010110 & = 11001111 \\
\end{array}
\]

\[
\begin{array}{cccc|cccc|cccc}
| & | & | & | & | & | & | & | & | & |
\end{array}
\]

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (9 points)

```sparc
C
i = 5; ! i mapped to %l0
do {
    statement1;
    statement2;
    ++i;
} while ( i <= 15 );
```

For the following instruction sequence, mark with an X the conditional branch instruction which would transfer control to loop if used in place of ba. (0-4 points: +1 for each correct; -1 for each incorrect; 0 if all marked)

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>bl</th>
<th>bne</th>
<th>bpos</th>
<th>bcs</th>
<th>ble</th>
<th>bge</th>
<th>be</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Bit Operations

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

set 0xB6127834, %l0
sll %l0, 7, %l0

Value in %l0 is 0x___________________________  (2 points)

set 0xB6127834, %l0
sra %l0, 5, %l0

Value in %l0 is 0x___________________________  (2 points)

set 0xB6127834, %l0
set 0x13579B65, %l1
xor %l0, %l1, %l0

Value in %l0 is 0x___________________________  (2 points)

In PA2, the shift operation across two 32-bit registers required us to make a copy of the bit that was being shifted out of the one register and put that bit in the appropriate bit position of the other register. Write the SPARC assembly instructions to do this piece of shift.s assuming bank0 is in %l0 and bank1 is in %l1 and you are shifting left. This operation can be done using only bitwise and shift instructions. Use %l5 for your bitmask. Use %l6 to hold a copy of the bit you need to put in the appropriate position of the other register. You may not use other registers. The only instructions you can use are bitwise and shift instructions. You do not need to deal with the shift count. (12 points)

__________________________________________! Set the bitmask into %l5
loop:

__________________________________________! Save the bit about to be shifted out into %l6

__________________________________________

__________________________________________

__________________________________________

dec %i1                                       ! Decrement the shift count

test:
5. Parameter Passing and Return Values

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

/* Assume this local variable is declared appropriately */

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

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

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

fb.c = foo( fb.b, fb.a );
```

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 some assignments, and returns the difference of the params:

```c
int foo( int i, int j ) {
    int local_stack_var1[3];
    int *local_stack_var2;
    local_stack_var2 = &local_stack_var1[1];
    *local_stack_var2 = 10;
    j = *local_stack_var2++;
    return ( i + local_stack_var1[2] );
}
```

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. (18 points)

```
SPARC assembly
.global foo
.section " .text"
foo: /* Your unoptimized code goes below this point */
```
Extra Credit

Optimize the following SPARC assembly language code fragment to eliminate any delay slots. (4 points)

Unoptimized Version

/* some code above here */

loop:
   addcc %l0, %l1, %l1
   be end
   nop
   inc %l0

test:
   cmp %l0, 15
   ble loop
   nop

del: 
   clr %o0

/*
 * other code that may
 * access %l0 or %l1
 */

Optimized Version
Scratch Paper