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 0xFBA7 (2’s complement, 16-bit word) to the following. (6 points)

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Convert +366 to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

<table>
<thead>
<tr>
<th></th>
<th>Sign-magnitude</th>
<th>1’s complement</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Convert -495 to the following (assume 16-bit word). Express answers in hexadecimal. (6 points)

<table>
<thead>
<tr>
<th></th>
<th>Sign-magnitude</th>
<th>1’s complement</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>10111011</td>
<td>01010111</td>
<td>11010110</td>
</tr>
<tr>
<td>+10010100</td>
<td>+00101001</td>
<td>+00111001</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>--------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<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>
</tr>
</tbody>
</table>

2
3. **Branching** (20 points)
Write the SPARC assembly instructions to complete the following. **Do not optimize nops**. (20 points)

```c
int findMax( int a[], int n ) {
    int *end = a + n;
    int max = a[0];

    ++a;
    while ( a < end ) {
        if ( *a > max )
            max = *a;
        ++a;
    }

    return max;
}
```

```sparc
.global findMax
.section ".text"

findMax:
    save %sp, -96, %sp
    /* Complete the rest of this function starting here */
    ! end mapped to $11
    ! max 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 0xABCDEF19, %l0
sll %l0, 9, %l0
Value in %l0 is 0x___________________________ (2 points)

set 0xABCDEF19, %l0
sra %l0, 13, %l0
Value in %l0 is 0x___________________________ (2 points)

set 0xABCDEF19, %l0
set 0x????????, %l1
xor %l0, %l1, %l0
Value set in %l1 must be this bit pattern 0x___________________________ (3 points)

! Value in %l0 is now 0xCAFEEFEED
Value in %l0 is now 0xCAFEEFEED

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)

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. (12 points)

```c
/* Function Prototype */

char foo( char, long, short );

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

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

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

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

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

fb.e[0] = foo( fb.a, fb.c, fb.f );
```
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( char a, int b ) {
  unsigned short *local_stack_var1;
  unsigned short  local_stack_var2[7];

  --local_stack_var1;                          /* statement 1 */
  *(local_stack_var1 + 3) = 222;               /* statement 2 */
  local_stack_var2[5] = a + 444;               /* statement 3 */
  local_stack_var1 = &local_stack_var2[4];     /* statement 4 */
  return ( a + (local_stack_var2[1] + 999) );  /* 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 assembly
  .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)

.global main

.section ".data"
fmt:   .asciz "0x%x\n" ! prints value as hex 0xXXXXXXXX

.c:
   .byte 0xEE
   .align 2

.s:
   .half 0x9897
   .align 4

.i1:
   .word 0xA9876543

.i2:
   .word 0xA9876543

.i3:
   .word 0xA9876543

.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
    ldcb [%l1], %l1
    stb %l1, [%l0+3]
    set fmt, %o0
    ld [%l0], %o1
    call printf
    nop

    set i3, %l0
    set x, %l1
    ldcb [%l0+1], %l2
    stb %l2, [%l1]
    ldsh [%l0], %l2
    sth %l2, [%l1+2]
    mov %l1, %l0
    set fmt, %o0
    ld [%l0], %o1
    call printf
    nop

    ret
    restore
Extra Credit (5 points)
Write a function in SPARC Assembly that takes two unsigned ints and interleaves the lower two bytes into a full-length unsigned integer and returns that interleaved value. For example,

word1: 0x 00 00 AC 4D
word2: 0x 00 00 73 B1

interleaves the lower two bytes to produce

result: 0x AC 73 4D B1

You may use only the following assembly instructions: \texttt{sll}, \texttt{srl}, \texttt{or} (along with \texttt{save}, \texttt{ret}, \texttt{restore}). (You should be able to do this with 4 \texttt{sll}, 4 \texttt{srl}, and 3 \texttt{or} instructions.)

Here is the function prototype for this function:

\begin{verbatim}
unsigned int interleave_bytes( unsigned int word1, unsigned int word2 );
\end{verbatim}
Scratch Paper
Scratch Paper