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 $FBCE_{16}$ (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>0</td>
<td></td>
</tr>
</tbody>
</table>

Convert $494_{10}$ 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 $-415_{10}$ 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)

```
10111011
+01010100
-----
N Z V C
| | | | |
```

```
01010110
+10101010
-----
N Z V C
| | | | |
```

```
01111011
+00010100
-----
N Z V C
| | | | |
```
3. Branching (20 points)
There are a few different ways to write checkRange() from PA1. Write checkRange with these restrictions:

The only instructions you can use are: `save, cmp, nop, bl, bg, ba, restore, mov, ret, .section, .global`

The following instructions can be used only once: `save, bl, bg, restore, ret, .section, .global`

The other instructions can be used more than once: `cmp, nop, ba, mov`

The only registers you can use are: `%i0, %i1, %i2, %sp`

No optimizations allowed (no filling delay slots with anything other than nops)

/* int checkRange( long theValue, long minRange, long maxRange ); */
/* Return value of 0 means No, theValue was not in the range; 1 otherwise. */

| Unoptimized version (17 pts) | Optimized version (filling nops) (3 pts) |
4. Bit Operations / C Runtime Environment

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

```c
set  0x867B5309, %l0
sll  %l0, 7, %l0
```

Value in %l0 is **0x___________________________** (2 points)

```c
set  0x867B5309, %l0
sra  %l0, 14, %l0
```

Value in %l0 is **0x___________________________** (2 points)

```c
set  0x867B5309, %l0
set  0x?????????, %l1
xor  %l0, %l1, %l0       ! Value in %l0 is now 0xCafeFEeD
```

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 layed 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 fb
   is declared appropriately
   and is the only local variable. */

struct fubar {
    char  a;
    char  b[4];
    long  c;
    char  d;
    short e;
    int   f;
    float g;
} fb;   /* Local variable fb */

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

/

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

fb.d = foo( fb.b[2], fb.c, fb.e );

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( int a, int b ) {
    short  local_stack_var1[5];
    short *local_stack_var2;

    local_stack_var1[3] = a + 777;               /* statement 1 */
    local_stack_var2 = &local_stack_var1[4];     /* statement 2 */
    ++local_stack_var2;                          /* statement 3 */
    *(local_stack_var2 + 2) = 333;               /* statement 4 */
    return ( b + (local_stack_var1[3] - 111) );  /* 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**

```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)

```assembly
.global main

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

c: .byte 0xAA

.align 2
s: .half 0x87CE

.align 4
i1: .word 0x98765432
i2: .word 0x98765432
i3: .word 0x98765432
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
ldub [%l1], %l1
stb %l1, [%l0+3]

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

set i3, %l0

set x, %l1
ldub [%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 the following function in SPARC Assembly to initialize the elements of an int array with the supplied value. **Unoptimized version on the left side and optimized version on the right side.**

```c
void
initArray( int a[], int size, int value )
{
    int i;

    for ( i = 0; i < size; ++i )
        a[i] = value;
}
```

**Unoptimized version here**

```
```
Scratch Paper