1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (17 points)
4. Bit Operations / C Runtime Environment ___________________ (16 points)
5. Parameter Passing and Return Values (Structures) ___________________ (10 points)
6. Local Variables, The Stack, and Return Values ___________________ (14 points)
7. Load/Store/Memory ___________________ (9 points)

SubTotal ___________________ (93 points)

Extra Credit ___________________ (4.5 points)

Total ___________________
1. Number Systems

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

binary  

octal  0

decimal

Convert $+424_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude 0x______________________________

1’s complement 0x______________________________

2’s complement 0x______________________________

Convert $-483_{10}$ 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. (12 points)

\[
\begin{array}{ccc}
11010111 & +10101011 & \text{---------} \\
11010101 & +00111001 & \text{---------} \\
00111001 & +01010110 & \text{---------}
\end{array}
\]

\[
\begin{array}{ccccc}
| N | Z | V | C |
\end{array}
\]

\[
\begin{array}{ccccc}
| N | Z | V | C |
\end{array}
\]

\[
\begin{array}{ccccc}
| N | Z | V | C |
\end{array}
\]

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (17 points)

```
C

for ( i = 11; i < 420; i += 3 ) { ! i mapped to %l4
  statementA;
  if ( i <= 99 ) {
    statementB;
    --i;
  } else
    statementC;
}
```
4. Bit Operations / C Runtime Environment

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

```
set 0xE95C64BA, %l0
sll %l0, 14, %l0
```

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

```
set 0xE95C64BA, %l0
sra %l0, 11, %l0
```

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

```
set 0xE95C64BA, %l0
set 0xCDCAB072, %l1
xor %l0, %l1, %l0
```

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

Fill in the names of the 5 areas of the C Runtime Environment as layed out by most Unix operating systems. Then state what parts of a C program are in each area. (10 points)

```
low 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. (10 points)

```
C

/* Function Prototype */
short foo( char, short );

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

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

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

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

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

    fb.a = foo( 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 ) {
    long *local_stack_var1;
    long  local_stack_var2[4];

    local_stack_var1 = local_stack_var2 + 2;  /* statement 1 */
    y = local_stack_var2[1];                  /* statement 2 */
    *local_stack_var1 = 9035768;              /* statement 3 */
    --local_stack_var1;                       /* statement 4 */
    return ( x + local_stack_var2[3] );       /* 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. (14 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"         ! hex 0xXXXXXXXX

c: .byte 0x55

.align 2
s: .half 0x1234

.align 4
i: .word 0x34567890

x: .word 0

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

set  i, %l0

set  c, %l1
ldub [%l1], %l1
stb  %l1, [%l0+3]

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

set  s, %l1
ldsh [%l1], %l1
sth  %l1, [%l0]

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

ldsh [%l0+2], %l1
sth  %l1, [%l0]

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

ret
restore
```
Extra Credit (4.5 points)

Take your answer to problem #3. Branching (page 3) and optimize it here. Some optimizations are better/more complicated than others and are therefore worth more points. Assume the "Statements" use variable i.
Scratch Paper