1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (18 points)
4. Bit Operations ___________________ (10 points)
5. Recursion/SPARC Assembly ___________________ (10 points)
6. Local Variables, The Stack and Return Values ___________________ (20 points)
7. SPARC Subroutines and Calling Convention ___________________ (14 points)
8. Floating Point ___________________ (18 points)
9. Linkage, Scope, Lifetime, Data ___________________ (31 points)
10. Machine Instructions ___________________ (20 points)
11. I/O & Virtual Memory ___________________ (11 points)
12. Miscellaneous ___________________ (32 points)

SubTotal ___________________ (211 points)
Extra Credit ___________________ (10 points)
Total ___________________
1. Number Systems

Convert $E_{DA}16$ (2’s complement, 16-bit word) to the following. (6 points)

binary _______________________________________

octal ________________________________

decimal ______________________________

Convert $+474_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude _______________________________________

1’s complement_____________________________________

2’s complement_____________________________________

Convert $-293_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

sign-magnitude _______________________________________

1’s complement_____________________________________

2’s complement_____________________________________
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)

\[
\begin{array}{ccc}
10111001 & +01010110 & 01000101 \\
+01010110 & +01010111 & +10111011 \\
\hline
\end{array}
\]

\[
\begin{array}{cccccc}
\hline
|   |   |   |   |   |   |   |   |   |   |   |   |
\hline
\end{array}
\]

3. Branching

Write the C statements to perform the following SPARC assembly statements. Do not optimize. (18 points)

\[
\begin{array}{l}
\text{SPARC assembly} \\
\text{C} \\
\hline
\text{mov} \ 41, \ %10 \\
\text{ba} \ \text{L4} \\
\text{nop} \\
\text{L1:} \\
\text{StatementA} \\
\text{cmp} \ %10, \ 9 \\
\text{bg} \ \text{L2} \\
\text{nop} \\
\text{StatementB} \\
\text{ba} \ \text{L3} \\
\text{nop} \\
\text{L2:} \\
\text{dec} \ %10 \\
\text{StatementC} \\
\text{L3:} \\
\text{add} \ %10, \ -5, \ %10 \\
\text{L4:} \\
\text{cmp} \ %10, \ -8 \\
\text{bg} \ \text{L1} \\
\text{nop} \\
\end{array}
\]

/* %10 mapped to local variable x */
4. Bit Operations

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

set 0xAB7E9F43, %l0
set 0x78CDCD78, %l1
or %10, %l1, %l0

Value in %l0 is ________________________________  (2 points)

set 0xAB7E9F43, %l0
sra %l0, 9, %l0

Value in %l0 is ________________________________  (2 points)

set 0xAB7E9F43, %l0
sll %l0, 11, %l0

Value in %l0 is ________________________________  (2 points)

set 0xAB7E9F43, %l0
set 0x91817161, %l1
bclr %l1, %l0

Value in %l0 is ________________________________  (2 points)

set 0xAB7E9F43, %l0
set 0x928276B5, %l1
btoq %l1, %l0

Value in %l0 is ________________________________  (2 points)
5. Recursion/SPARC Assembly

Given **main.s** and **fubar.s**, what gets printed when executed? (10 points)

```assembly
/* main.s */
.global main

.section ".rodata"
.align 4
code: .word 0x43216153, 0x21764567, 0x61336E4A, 0x306F2020, 0x4C3E5220, 0x20756543
.word 0x6C760065, 0x6900734C, 0x00000030

.section ".text"
main:
  save %sp, -92 & -8, %sp
  set code, %o0
  mov 1, %o1
  call fubar
  nop
  ret

restore

/* fubar.s */

.global fubar

.section ".rodata"
fmt: .asciz "%c"

.section ".text"
fubar:
  save %sp, -(92 + 1) & -8, %sp
  inc %i1
  cmp %i0, %g0
  be end
  nop
  ldub [%i0 + %i1], %l0
  cmp %l0, %g0
  be end
  nop
  stb %l0, [%fp - 1]
  add %i1, 2, %o0
  mov %i0, %o0
  call fubar
  nop
  set fmt, %o0
  ldub [%fp - 1], %o1
  call printf
  nop

end:
  ret
restore
```
6. Local Variables, The Stack, and Return Values

Here is a C function that allocates a couple local variables, performs some assignments and returns a value. Don’t worry about any local variables not being initialized before being used. Just do a direct translation. Draw lines.

```c
int
fubar( int a, int b )
{
    short local_stack_var1;
    int *local_stack_var2;
    struct foo {
        int s1;
        unsigned short s2[3];
        char s3;
        int s4;
    } local_stack_var3;

    local_stack_var3.s3 = 'E'; /* Use the ASCII value, not 'E' */
    local_stack_var1 = local_stack_var3.s2[2];
    local_stack_var3.s1 = *++local_stack_var2;

    return ( local_stack_var3.s4 - a );
}
```

Now write the equivalent full unoptimized SPARC assembly language module to perform the equivalent. You must allocate all local variables on the stack. No short cuts. Treat each statement independently. (20 points)
7. SPARC Subroutines and Calling Convention (2 point each)

In most architectures, some of the Stack Frame is built by the calling function and some of the Stack Frame is built by the called function. The same is generally true of the SPARC architecture. Fill in the blanks for the general model of Stack Frame creation/function call convention.

The __________ function calls the instruction to set the program counter with the saved return address.

The __________ function deallocates the space of the Stack used for the arguments

The __________ function allocates space for and possibly saves values of registers that it may use.

The __________ function allocates space on the Stack for the arguments being passed in the function call.

The __________ function places the return value in the architecture-specific return value area.

The __________ function calls the instruction to save the current value of the program counter to be used as the return address.

The __________ function allocates space on the Stack for local variables and performs any initialization.

8. Floating Point

Convert $97.375_{10}$ (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

binary fixed-point ________________ (2 points)

IEEE floating-point ________________ (4 points)

Convert $0xC3D25000$ (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal ________________ (6 points)

Complete the following initializations of the parts of a single-precision IEEE floating-point representation: (2 pts each)

```c
void classifyFloatParts( float f ) {
    unsigned int bits = *(unsigned int *) &f; /* bits is the bit pattern of f */
    unsigned int sign = ________________; /* sign bit */
    unsigned int exp = ________________; /* exponent bits */
    unsigned int frac = ________________; /* mantissa/fraction bits */
    ...
}
```
For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (31 points — 1 point each)

```c
int a;
int b = 4;
static int c = 20;
static int d;
int foo( int e ) {
    float f = 4.20;
    static int g = 420;
    int h;
    int *i;
    i = (int *) malloc( g );
}
...
```

Fill in the letter corresponding to the correct scoping/visibility for each of the variables:
- A) Global across all modules/functions linked with this source file.
- B) Global just to this source file.
- C) Local to function foo().

<table>
<thead>
<tr>
<th>Variable</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>foo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill in the letter corresponding to the correct lifetime for each of the variables:
- A) Exists from the time the program is loaded to the point when the program terminates.
- B) Exists from the time function foo() is called to the point when foo() returns.

<table>
<thead>
<tr>
<th>Variable</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>foo</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10. Machine Instructions

Translate the following instructions into SPARC machine code. Use hexadecimal values for your answers. If an instruction is a branch, specify the number of instructions away for the target (vs. a Label).

\[
\text{add } %i3, %i5, %o1 \quad \text{______________________________} \quad (5 \text{ points})
\]

\[
\text{ld } [\%o1], %i4 \quad \text{______________________________} \quad (5 \text{ points})
\]

11. I/O & Virtual Memory

Which combination of two of the following is slowest/most inefficient for large data transfers.

- Programmed I/O
- Interrupt-driven ___________ / ___________ (2 points)
- DMA
- CPU blocked (Polling)

Paging and swapping refer to parts of or entire program images being moved back and forth between what 2 storage areas/hierarchies? (2 points)

_________________________ and  _________________________

A ________ __________ occurs because a page is needed but was not found to be in main memory. (1 points)

The _____________ translates ______________ to ______________ addresses. (3 points)

The _____________ caches these recently translated addresses. (1 points)

This type of I/O uses regular instructions to perform I/O and not any special instructions. (1 points)

______________________________

A ____________________ cache ensures the data in the cache is the same as what is in main memory. (1 points)
12. Miscellaneous

Circle T for true; Circle F for false; +2 if correct, 0 if incorrect or blank, -1 if both T and F are circled:

T or F One way to generally speed up string compares on a Big-Endian machine is to use word compares instead of byte by byte compares. (2 points)

T or F With virtual memory, more than one process using the same virtual address space can reside in physical memory at the same time. (2 points)

T or F Paging will free up physical memory faster than swapping. (2 points)

T or F The hidden bit in IEEE floating-point representation allows us to represent larger values. (2 points)

T or F When combining segmentation with paging, paging is done within segments. (2 points)

T or F Static linked binaries are usually smaller than dynamic linked binaries. (2 points)

T or F Doubling the clock rate on a specific CPU will mean all instructions will execute twice as fast. (2 points)

T or F The resulting Text segment of a program compiled on a CISC system is generally larger than that of the same program compiled on a RISC system. (2 points)

Write this modified `classifyFloatParts()` in SPARC assembly. Allocate local variables on the Stack. (16 points)

```c
unsigned int foo( float *ptr ); /* Function prototype for foo() */

void classifyFloatParts( float f1, float f2 ) {
    unsigned int bits1 = foo( &f1 ); /* Local Stack variable */
    unsigned int bits2 = foo( &f2 ); /* Local Stack variable */
}
```
Extra Credit

What does the following SPARC assembly language program output?

```
.global main

.section ".rodata"
fmt:    .asciz  "%c"
.align 4
foo:    .word   0x43005321, 0x4567336E, 0x306F204C, 0x52207565, 0x6C766569, 0x734C0000

.section ".text"

main:
  save  %sp, -92 & -8, %sp
  mov   1, %l1
  clr   %l2
  set   foo, %l0
  ba    test
  nop
loop:
  set   fmt, %o0
  ldup  [%l0+%l2], %o1
  call  printf, 2
  nop
  mov   %l1, %o0
  mov   2, %o1
  call  .mul
  nop
  mov   %o0, %l2
  inc   %l1
test:
  ldup  [%l0+%l2], %o1
  tst   %o1
  bne   loop
  nop
  set   fmt, %o0
  mov   0x0A, %o1
  call  printf, 2
  nop
  ret
  restore
```

```
main:
  save  %sp, -92 & -8, %sp
  mov   1, %l1
  clr   %l2
  set   foo, %l0
  ba    test
  nop
loop:
  set   fmt, %o0
  ldup  [%l0+%l2], %o1
  call  printf, 2
  nop
  mov   %l1, %o0
  mov   2, %o1
  call  .mul
  nop
  mov   %o0, %l2
  inc   %l1

  test:
  ldup  [%l0+%l2], %o1
  tst   %o1
  bne   loop
  nop
  set   fmt, %o0
  mov   0x0A, %o1
  call  printf, 2
  nop
  ret
  restore
```

Output ________________________
(2 points)

Now optimize the code to get the same result with the fewest cycles. Some optimizations are better than others. You may not be able to eliminate all nops. Go for the fewest machine cycles assuming memory accesses are several more cycles than other non-memory access instructions. You cannot change the overall algorithm. (8 points)
Hexadecimal - Character

<table>
<thead>
<tr>
<th>00 NUL</th>
<th>01 SOH</th>
<th>02 STX</th>
<th>03 ETX</th>
<th>04 EOT</th>
<th>05 ENQ</th>
<th>06 ACK</th>
<th>07 BEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>08 BS</td>
<td>09 HT</td>
<td>0A NL</td>
<td>0B VT</td>
<td>0C NP</td>
<td>0D CR</td>
<td>0E SO</td>
<td>0F SI</td>
</tr>
<tr>
<td>10 DLE</td>
<td>11 DC1</td>
<td>12 DC2</td>
<td>13 DC3</td>
<td>14 DC4</td>
<td>15 NAK</td>
<td>16 SYN</td>
<td>17 ETB</td>
</tr>
<tr>
<td>18 CAN</td>
<td>19 EM</td>
<td>1A SUB</td>
<td>1B ESC</td>
<td>1C FS</td>
<td>1D GS</td>
<td>1E RS</td>
<td>1F US</td>
</tr>
<tr>
<td>20 SP</td>
<td>21 !</td>
<td>22 &quot;</td>
<td>23 #</td>
<td>24 $</td>
<td>25 %</td>
<td>26 &amp;</td>
<td>27 '</td>
</tr>
<tr>
<td>28 (</td>
<td>29 )</td>
<td>2A *</td>
<td>2B +</td>
<td>2C ,</td>
<td>2D -</td>
<td>2E .</td>
<td>2F /</td>
</tr>
<tr>
<td>30 0</td>
<td>31 1</td>
<td>32 2</td>
<td>33 3</td>
<td>34 4</td>
<td>35 5</td>
<td>36 6</td>
<td>37 7</td>
</tr>
<tr>
<td>38 8</td>
<td>39 9</td>
<td>3A :</td>
<td>3B ;</td>
<td>3C &lt;</td>
<td>3D =</td>
<td>3E &gt;</td>
<td>3F ?</td>
</tr>
<tr>
<td>40 0</td>
<td>41 A</td>
<td>42 B</td>
<td>43 C</td>
<td>44 D</td>
<td>45 E</td>
<td>46 F</td>
<td>47 G</td>
</tr>
<tr>
<td>48 H</td>
<td>49 I</td>
<td>4A J</td>
<td>4B K</td>
<td>4C L</td>
<td>4D M</td>
<td>4E N</td>
<td>4F O</td>
</tr>
<tr>
<td>50 P</td>
<td>51 Q</td>
<td>52 R</td>
<td>53 S</td>
<td>54 T</td>
<td>55 U</td>
<td>56 V</td>
<td>57 W</td>
</tr>
<tr>
<td>58 X</td>
<td>59 Y</td>
<td>5A Z</td>
<td>5B [</td>
<td>5C \</td>
<td>5D ]</td>
<td>5E ^</td>
<td>5F _</td>
</tr>
<tr>
<td>60 `</td>
<td>61 a</td>
<td>62 b</td>
<td>63 c</td>
<td>64 d</td>
<td>65 e</td>
<td>66 f</td>
<td>67 g</td>
</tr>
<tr>
<td>68 h</td>
<td>69 i</td>
<td>6A j</td>
<td>6B k</td>
<td>6C l</td>
<td>6D m</td>
<td>6E n</td>
<td>6F o</td>
</tr>
<tr>
<td>70 p</td>
<td>71 q</td>
<td>72 r</td>
<td>73 s</td>
<td>74 t</td>
<td>75 u</td>
<td>76 v</td>
<td>77 w</td>
</tr>
<tr>
<td>78 x</td>
<td>79 y</td>
<td>7A z</td>
<td>7B {</td>
<td>7C</td>
<td></td>
<td>7D }</td>
<td>7E ~</td>
</tr>
</tbody>
</table>
Scratch Paper