<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Number Systems</td>
<td>15</td>
</tr>
<tr>
<td>2. Binary Addition/Condition Code Bits/Overflow Detection</td>
<td>12</td>
</tr>
<tr>
<td>3. Branching</td>
<td>22</td>
</tr>
<tr>
<td>4. Bit Operations</td>
<td>13</td>
</tr>
<tr>
<td>5. Recursion/SPARC Assembly</td>
<td>10</td>
</tr>
<tr>
<td>6. Local Variables, The Stack and Return Values</td>
<td>24</td>
</tr>
<tr>
<td>7. SPARC Leaf Subroutines</td>
<td>23</td>
</tr>
<tr>
<td>8. Floating Point</td>
<td>12</td>
</tr>
<tr>
<td>9. Machine Instructions</td>
<td>20</td>
</tr>
<tr>
<td>10. Linkage, Scope, Lifetime, Data</td>
<td>32</td>
</tr>
<tr>
<td>11. Load/Store/Memory</td>
<td>9</td>
</tr>
<tr>
<td>12. Miscellaneous</td>
<td>32</td>
</tr>
</tbody>
</table>

SubTotal: 224 points

Extra Credit: 10 points

Total: 234 points
1. Number Systems

Convert $\text{FB4}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

- binary __________________________
- octal __________________________
- decimal ________________________

Convert $-576_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- sign-magnitude __________________________
- 1’s complement _______________________
- 2’s complement _______________________

Convert $329_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 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)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01010101</td>
<td>+10101011</td>
<td></td>
<td></td>
<td>11010110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+11010100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Branching

Write the SPARC assembly statements to perform the following C statements. Do not optimize. (22 points)

```c
int x; // map x to %l1
int y; // map y to %l5

for ( y = 15; y <= 75; y = y + 3) {
    y = y % 5;
    if ( x <= y )
        x = bar( y );
}
```
4. Bit Operations

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

set 0xFADED420, %l0
set 0x420FADED, %l1
or %l0, %l1, %l0

Value in %l0 is _________________________________ (2 points)

set 0xFADED420, %l0
srl %l0, 13, %l0

Value in %l0 is _________________________________ (2 points)

set 0xFADED420, %l0
sra %l0, 9, %l0

Value in %l0 is _________________________________ (2 points)

set 0xFADED420, %l0
set 0x????????, %l1
btog %l1, %l0 ! Value in %l0 is now 0xFEEDBabe

Value set in %l1 must be this bit pattern _________________________________ (3 points)

set 0xFADED420, %l0
set 0x420FADED, %l1
and %l0, %l1, %l0

Value in %l0 is _________________________________ (2 points)

set 0xFADED420, %l0
sll %l0, 7, %l0

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

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

```
.globl main
/* main.s */

.main:
    save   %sp, -92 & -8, %sp
    set    736952856, %o0
    call   recurse
    nop
    ret
    restore

.globl recurse
/* recurse.s */

.recurse:
    save   %sp, -(92 + 8) & -8, %sp
    clr    %l0
    skip:
    st     %g0, [%fp - 4]
    st     %g0, [%fp - 8]
    mov    %i0, %o0
    mov    10, %o1
    call   .rem
    nop
    mov    %o0, %o0
    st     %o0, [%fp - 4]
    mov    %i0, %o0
    mov    10, %o1
    call   .div
    nop
    mov    %o0, %i0
    cmp    %i0, %g0
    be     no_go
    nop
    ld     [%fp - 4], %l0
    cmp    %l0, 5
    bl     skip
    nop
    mov    %i0, %o0
    call   recurse
    nop
    st     %o0, [%fp - 8]
    no_go:
    ld     [%fp - 4], %o0
    ld     [%fp - 8], %o1
    add    %o0, %o1, %o0
    st     %o0, [%fp - 8]
    set    fmt, %o0
    ld     [%fp - 8], %o1
    call   printf
    nop
    ld     [%fp - 8], %i0
    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
long fubar( short x, long y ) {
    char *local_stack_var1;
    struct foo {
        short s1[3];
        short s2;
        char s3;
        long s4;
    } local_stack_var2;
    int local_stack_var3;

    local_stack_var3 = local_stack_var2.s1[2] + 9; /* 1 */
    local_stack_var2.s2 = x; /* 2 */
    local_stack_var1 = &local_stack_var2.s3; /* 3 */
    return ( y + local_stack_var2.s4 ); /* 4 */
}
```

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. (24 points)
7. SPARC Leaf Subroutines

Write a full unoptimized leaf SPARC assembly function translation of the following C function to determine how many even numbered bits in the parameter value are set. Return the number of even numbered bits that are set to 1. The most significant bit is numbered bit 31 (odd); the least significant bit is numbered bit 0 (even). **Be sure to state which registers you are using for the various local variables and parameters.** (23 pts)

For example, \( \text{countEvenBitsSet}( 0x55555555 ) \) will return 16
\( \text{countEvenBitsSet}( 0x504554A5 ) \) will return 10

\[
\begin{align*}
\text{C} \\
\text{leaf SPARC Assembly Subroutine}
\end{align*}
\]

```c
int countEvenBitsSet( unsigned int value ) {
    int i;
    int cnt = 0;
    unsigned int mask = 0x40000000;

    for ( i = 0; i < 16; ++i ) {
        if ( (value & mask) != 0 )
            ++cnt;
        mask = mask >> 2;
    }

    return cnt;
}
```
8. Floating Point

Convert 124.625₁₀ (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 0xC3446000 (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal __________________________________ (6 points)

9. 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).

std %i₂, [%o₃ + %l₆] __________________________________ (5 points)

subcc %i₂, %l₂, %o₅ __________________________________ (5 points)

Translate the following SPARC machine code instructions into SPARC assembly instructions.

0xD43D001B __________________________________ (5 points)

0x9A18FFF6 __________________________________ (5 points)
For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (32 points — 1 point each)

```c
int a = 0; ______________
static int b = 911; ______________
static int c; ______________
int d; ______________
int foo( int e ) {          ____________ (foo) ______________ (e)
    int f = 420; ______________
    static double g = 4.20; ______________
    int (*h)(int) = foo; ______________ (h) ______________ (where h is pointing)
    static int *i;
    i = (int *) malloc( b ); ______________ (where i is pointing)
    ...
}
```

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().

a _______ b _______ c _______ d _______ e _______ f _______ g _______ h _______ i _______ foo _______

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.

a _______ b _______ c _______ d _______ e _______ f _______ g _______ h _______ i _______ foo _______
11. Load/Store/Memory

What gets printed in the following program? (9 points)

```assembly
.global main

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

.c:
    .byte   0x44
    .align 2

.s:
    .half   0xBEAD
    .align 4

.i1:   .word   0xAB88CDEF
.i2:   .word   0xAB88CDEF
.i3:   .word   0xAB88CDEF
.x:    .word   0x00009999

.section ".text"
main:
    save    %sp, -96, %sp
    set     i1, %l0
    set     s, %l1
    lduh    [%l1], %l1
    st      %l1, [%l0]
    stb     %l1, [%l0+1]
    set     fmt, %o0
    ld      [%l0], %o1
    call    printf ________________________
    nop

    set     i2, %l0
    set     c, %l1
    ldsb    [%l1], %l1
    sth    %l1, [%l0+2]
    stb    %l1, [%l0]
    set     fmt, %o0
    ld    [%l0], %o1
    call   printf ________________________
    nop

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

    ret
    restore
```
12. Miscellaneous

Put the following in the correct order/sequence using the numbers to the left of each word:

1. executable (.exe/a.out)  4. loader  7. program execution
2. C preprocessor  5. assembler  8. C source code
3. linker/linkage editor  6. compiler

_____ → _____ → _____ → _____ → _____ → _____ → _____ → _____ → _____

Is SPARC a CISC or RISC architecture (Circle correct answer)? (1 pt)

What does SPARC stand for? (1 pt)

What is the instructor’s middle name as given by the class? (1 pt)

What does BSS stand for? (1 pt)

Given the following program, order the printf() lines so that the values that are printed when run on a Sun SPARC Unix system are displayed from smallest value to largest value. (2 points each)

```c
void foo( int, int ); /* Function Prototype */

int a = 911;

int main( int argc, char *argv[] ) {
    int b = 911;
    int c = 405;

    foo( c, argc );

    /* 1 */ (void) printf( "a --> %p\n", &a );
    /* 2 */ (void) printf( "argc --> %p\n", &argc );
    /* 3 */ (void) printf( "foo --> %p\n", foo );
    /* 4 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 5 */ (void) printf( "b --> %p\n", &b );
    /* 6 */ (void) printf( "c --> %p\n", &c );
}

void foo( int d, int e ) {

    int f = a;
    static int g;

    /* 7 */ (void) printf( "g --> %p\n", &g );
    /* 8 */ (void) printf( "e --> %p\n", &e );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "d --> %p\n", &d );
}
```

prints

smallest value

largest value
Extra Credit
What does the following SPARC assembly language program output?

```
.global main

.section ".rodata"
fmt: .asciz "%c"
.align 4
foo: .word 0x00005343, 0x7523723C, 0x664A2061, 0x4E766161, 0x6B3A652D, 0x64290021, 0x00007353

.section ".text"
main:
save %sp, -96, %sp
set foo, %l0
clr %l1
mov 3, %l2
clr %l3
mov %l2, %l4
inc %l1
ba test
nop

loop:
set fmt, %o0
ldub [%l0+%l4], %o1
call printf, 2
nop
inc %l1
mov %l1, %o0
mov 2, %o1
call .mul
nop

mov %o0, %l2
add %l3, %l2, %l4
inc %l4
test:
ldub [%l0+%l4], %o1
tst %o1
bne loop
nop
set fmt, %o0
mov 0x0A, %o1
call printf, 2
nop
ret
restore
```

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