1. Number Systems

2. Binary Addition/Condition Code Bits/Overflow Detection

3. Branching

4. Bit Operations / C Runtime Environment

5. Parameter Passing and Return Values (Stack Variables)

6. Local Variables, The Stack, and Return Values

7. Load/Store/Memory

SubTotal

Extra Credit

Total
1. Number Systems

Convert \( \texttt{0xFB73} \) (2’s complement, 16-bit word) to the following. (6 points)

<table>
<thead>
<tr>
<th>binary</th>
<th>octal</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Convert -341 to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

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

Convert +418 to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

<table>
<thead>
<tr>
<th>sign-magnitude</th>
<th>1’s complement</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<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)

\[
\begin{array}{c}
01100001 \\
+11001001 \\
---------
\end{array} \quad \begin{array}{c}
11010111 \\
+10101001 \\
---------
\end{array} \quad \begin{array}{c}
01011011 \\
+00100101 \\
---------
\end{array}
\]

<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>
3. Branching (20 points)
Translate the C code below into the equivalent unoptimized SPARC Assembly code. Just perform a direct translation – no optimizations. Use the local register mappings for the variables in assembly as specified.

```c
/* Assume variables a and b have been properly declared as ints. */
do {
    if ( a == b )
    {
        b = b / a;
    }
    else if ( b < 42 )
    {
        b = a + 37;
    }
} while( b > 88 );
```

```sparc
! a is mapped to %l3
! b is mapped to %l6
```
4. Bit Operations / C Runtime Environment

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

set 0x1234CAFE, %l0
sra %l0, 6, %l0

Value in %l0 is 0x_______________________________________  (2 points)

set 0x1234CAFE, %l0
sll %l0, 10, %l0

Value in %l0 is 0x_______________________________________  (2 points)

set 0x1234CAFE, %l0
set 0x????????, %l1
xor %l0, %l1, %l0  ! Value in %l0 is now OxCAFE1234

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 laid out by the SPARC architecture. Then state what parts of a C program are in each area. (10 points)

<table>
<thead>
<tr>
<th>low memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>high memory</th>
</tr>
</thead>
</table>

|            |
|            |
|            |
|            |
|            |
|            |
|            |

5. Parameter Passing and Return Values (Local Stack Struct Variable)

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)

\[
\text{C} \\
/* Function Prototype */ \\
short foo( char, unsigned short, int ); \\
/* ... Other code ... */ \\
/* Assume this local variable is declared appropriately and is the only local var. */ \\
struct fubar { \\
    short a; \\
    int b; \\
    char c[3]; \\
    unsigned short d[2]; \\
} fb; /* Local variable fb */ \\
/* ... Other code ... */ \\

SPARC assembly \\
/* */ \\
Write the code for just this function call, saving the return value appropriately */ \\
fb.a = foo( fb.c[0], fb.d[0], fb.b ); \\
/* ... Other code ... */ \\

Using the Rt-Lt Rule, write the C variable definition for the variable named \texttt{bar} that is a pointer to a function that takes a single argument of type pointer to float and returns a pointer to a long. (4 pts)
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 ) {
    int *local_stack_var1;
    int local_stack_var2[4];

    a = *local_stack_var1++; /* statement 1 */
    local_stack_var1 = &local_stack_var2[3]; /* statement 2 */
    *local_stack_var1 = 7777777; /* statement 3 */
    local_stack_var2[0] = b; /* statement 4 */
    return ( ++b + local_stack_var2[2] ); /* 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. (16 points)

```assembly
.global fubar
.section ".text"

fubar: /* Your unoptimized code goes below this point */
```
7. **Load/Store/Memory** Specify the hex values requested after those lines have been fully executed. (11 points)

```assembly
.global main

.section ".data"
fmt: .asciz "0x%08X\n" ! prints value as hex 0xXXXXXXXX

c: .byte 0xAA
.s: .half 0xABCD
.i1: .word 0x23456789
.i2: .word 0x23456789
.i3: .word 0x23456789
.x: .word 0x66660000

.section ".text"
main:
  save %sp, -96, %sp
  set x, %l0
  set s, %l1
  lduh [%l1], %l2
  stb %l2, [%l0+3]
  srl %l2, 4, %l2
  stb %l2, [%l0]
  set fmt, %o0
  ld [%l0], %o1
  call printf
  nop
  set i1, %l0
  set c, %l1
  ldsb [%l1], %l2
  sth %l2, [%l0+2]
  stb %l2, [%l0]
  set fmt, %o0
  ld [%l0], %o1
  call printf
  nop
  set i2, %l0
  set i3, %l1
  ld [%l1], %l2
  sth %l2, [%l0]
  sra %l2, 16, %l2
  sth %l2, [%l0+2]
  set fmt, %o0
  ld [%l0], %o1
  call printf
  nop
  ret
  restore
```
Extra Credit
What gets printed at each printf() statement given the following C program? (8 pts)

```
#include <stdio.h>

int main()
{
    char a[] = "FEB13";
    char *p = a;

    printf("%c", *p++ );  ______
    printf("%c", *++p );  ______
    printf("%c", *(p+2) = p[1] );  ______
    printf("%c", *(p+1) = p[-1] );  ______
    printf("%c", *p++ = *a + 2 );  ______
    printf("%c", --*p++ );  ______
    printf("%d", ++p - a );  ______
    printf("\n%s\n", a );  _______________________

    return 0;
}
```

A portion of the C Operator Precedence Table

```
Operator Associativity
++ postfix increment L to R
-- postfix decrement R to L
[] array element function call
------------
* indirection R to L
++ prefix increment
-- prefix decrement
sizeof size of type/object
(type) type cast
------------
* multiplication L to R
/ division
% modulus
------------
+ addition L to R
- subtraction

= assignment R to L
```

Hexadecimal - Character

```
| 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 DBL| 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 EM | 1F US |
| 20 SP | 21 ! | 22 " | 23 \ | 24 $ | 25 % | 26 & | 27 ' |
| 28 ( | 29 ) | 30 * | 31 + | 32 , | 33 - | 34 . | 35 / |
| 36 0 | 37 1 | 38 2 | 39 3 | 40 4 | 41 5 | 42 6 | 43 7 |
| 44 8 | 45 9 | 46 A | 47 B | 48 C | 49 D | 50 E | 51 F |
| 52 G | 53 H | 54 I | 55 J | 56 K | 57 L | 58 M | 59 N |
| 60 O | 61 P | 62 Q | 63 R | 64 S | 65 T | 66 U | 67 V |
| 68 W | 69 X | 6A Y | 6B Z | 6C a | 6D b | 6E c | 6F d |
| 70 e | 71 f | 72 g | 73 h | 74 i | 75 j | 76 k | 77 l |
| 78 m | 79 n | 7A o | 7B p | 7C q | 7D r | 7E s | 7F t |
```
Scratch Paper