CSE 30
Fall 2013
Midterm Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (22 points)
4. Bit Operations ___________________  (19 points)
5. Parameter Passing and Return Values (Stack Variables) ___________________  (16 points)
6. Local Variables, The Stack and Return Values ___________________  (16 points)
7. Load/Store/Memory ___________________  (11 points)

SubTotal ___________________  (111 points)

Extra Credit ( >7% ) ___________________  (8 points)

Total ___________________
1. Number Systems

Convert 0xFB99 (2’s complement, 16-bit word) to the following. (6 points)

<table>
<thead>
<tr>
<th>Base</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>[Conversion]</td>
</tr>
<tr>
<td>Octal</td>
<td>0 [Conversion]</td>
</tr>
<tr>
<td>Decimal</td>
<td>[Conversion to signed]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign-magnitude</td>
<td>0x</td>
</tr>
<tr>
<td>1’s complement</td>
<td>0x</td>
</tr>
<tr>
<td>2’s complement</td>
<td>0x</td>
</tr>
</tbody>
</table>

Convert +448 to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign-magnitude</td>
<td>0x</td>
</tr>
<tr>
<td>1’s complement</td>
<td>0x</td>
</tr>
<tr>
<td>2’s complement</td>
<td>0x</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)

```
11010111 + 10101001 00100001 + 01001001 10000000 + 10000000
----------- ----------- ----------- ---------------
N  Z  V  C  N  Z  V  C  N  Z  V  C
```

<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 (22 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. */

if ( (a > b) || (a < 22) )
{
    a = b + 44;
}
else
{
    --b;
    do
    {
        a = a % b;
    } while ( a >= 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 0xDEADBEEF, %l0
  sra %l0, 9, %l0
  Value in %l0 is 0x__________________________ (2 points)

- set 0xDEADBEEF, %l0
  sll %l0, 11, %l0
  Value in %l0 is 0x__________________________ (2 points)

- set 0xDEADBEEF, %l0
  set 0x?????????, %l1
  xor %l0, %l1, %l0
  ! Value in %l0 is now 0xCAFEBABE
  Value set in %l1 must be this bit pattern 0x__________________________ (3 points)

Fill in the blanks to complete the SPARC assembly instructions to perform a 64-bit rotate right of one bit across two 32-bit registers using the comments as a guide. (1 point each)

/* 64-bit Rotate Right one bit across two regs: Upper 32 bits in %l0; Lower 32 bits in %l1 */

BIT_31_MASK = 0x______________________    ! Mask for bit 31
BIT_0_MASK  = 0x______________________    ! Mask for bit 0

/* Grab the bit from the upper 32 bits that will be rotated into the lower 32 bits */
________ %l0, ___________MASK, %l2    ! Which mask to use?
/* Shift this bit to correct bit position for insertion later into the lower 32 bits */
________ %l2, 31, %l2
/* Grab the bit from the lower 32 bits that will be rotated into the upper 32 bits */
________ %l1, ___________MASK, %l3    ! Which mask to use?
/* Shift this bit to correct bit position for insertion later into the upper 32 bits */
________ %l3, 31, %l3
/* Shift upper 32 bits to make room for the bit being rotating in */
________ %l0, 1, %l0
/* Shift lower 32 bits to make room for the bit being rotating in */
________ %l1, 1, %l1
/* Insert bit from lower 32 bits being rotated into the upper 32 bits */
________ %l0, %l3, %l0
/* Insert bit from upper 32 bits being rotated into the lower 32 bits */
________ %l1, %l2, %l1
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)

```
C

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

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

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

struct fubar {
    char a[5];
    short b;
    unsigned short c;
    int d;
} fb; /* Local variable fb */

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

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

fb.a[3] = foo( fb.d, fb.b, fb.c );
```

Put your SPARC Assembly code in the box below.

Using the Rt-Lt Rule, write the C variable definition for the variable named fubar that is a pointer to a function that takes a single argument of type pointer to int and returns a pointer to a double. (4 pts)
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 ) {
    int *local_stack_var1;
    int local_stack_var2[6];

    *local_stack_var1++ = a;                  /* statement 1 */
    local_stack_var1 = &local_stack_var2[2]; /* statement 2 */
    b = local_stack_var2[4];                 /* statement 3 */
    *local_stack_var1 = 9990999;             /* statement 4 */
    return ( a + local_stack_var2[1] );      /* 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
.globa l fubar
.se ction  ".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)

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

.c: .byte  0xDD
    .align 2

.s: .half 0xCAFE
    .align 4

.il: .word 0x12345678
.ii: .word 0x12345678
.iii: .word 0x12345678
.x:  .word 0xAAAA0000

.section ".text"
main:
    save  %sp, -96, %sp
    set   x, %l1
    set   s, %l2
    lduh  [%l1], %l2
    stb   %l2, [%l0+1]   _____________________  Hex value in word labeled x
    srl   %l2, 8, %l2
    stb   %l2, [%l0+2]
    set   fmt, %o0
    ld    [%l0], %o1
    call  printf
    nop   (same as output of this printf)

    set   il, %l0
    set   c, %l1
    lduh  [%l1], %l2
    sth   %l2, [%l0+2]
    stb   %l2, [%l0+1]
    set   fmt, %o0
    ld    [%l0], %o1
    call  printf
    nop   (same as output of this printf)

    set   i2, %l0
    set   i3, %l1
    ld    [%l1], %l2
    sth   %l2, [%l0]
    sra   %l2, 8, %l2
    sth   %l2, [%l0]
    set   fmt, %o0
    ld    [%l0], %o1
    call  printf
    nop   (same as output of this printf)

    ret
    restore
```
**Extra Credit** (8 points)

What gets printed at each printf() statement given the following C program? (8 pts)

```c
#include <stdio.h>

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

    printf( "%c", p[2] + 1 );
    printf( "%c", *(a+4) = *p );
    printf( "%c", *p++ );
    printf( "%c", *p-- );
    printf( "%c", +++p = *(a+2) + 2 );
    printf( "%c", *(p+2) );
    printf( "%d", ++p - a );
    printf( "\n%s\n", a );

    return 0;
}
```

---

**A portion of the C Operator Precedence Table**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ post-fix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- post-fix decrement</td>
<td></td>
</tr>
<tr>
<td>[] array element</td>
<td></td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of size of type/object (type) type cast</td>
<td></td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>

---

**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 NF</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 _</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 F</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