CSE 30
Winter 2008
Midterm Exam

1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (21 points)
4. Bit Operations / C Runtime Environment ___________________ (17 points)
5. Parameter Passing and Return Values (Stack Variables) ___________________ (12 points)
6. Local Variables, The Stack and Return Values ___________________ (15 points)
7. Load/Store/Memory ___________________ (9 points)

SubTotal ___________________ (101 points)
Extra Credit ___________________ (7 points)

Total ___________________
1. Number Systems

Convert \(0xFE4A\) (2’s complement, 16-bit word) to the following. (6 points)

- **binary**
  - ___________________________ (straight base conversion)

- **octal**
  - 0__________________________ (straight base conversion)

- **decimal**
  - ___________________________ (convert to signed decimal)

Convert -317 to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- **sign-magnitude**
  - 0x________________________

- **1’s complement**
  - 0x________________________

- **2’s complement**
  - 0x________________________

Convert +488 to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 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)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11010111</td>
<td>11000101</td>
<td>01111111</td>
</tr>
<tr>
<td>+00001001</td>
<td>+10111001</td>
<td>+10000001</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. **Branching** (21 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 <= 249 )
{
    while ( b <= 37 )
    {
        b = a * 13;
    }
} else {
    b = b + a;
}
```

```
SPARC ASSEMBLY
! a is mapped to %l2
! b is mapped to %l5

if ( a <= 249 )
{
    while ( b <= 37 )
    {
        b = a * 13;
    }
} else {
    b = b + a;
}
```
4. Bit Operations / C Runtime Environment

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

- `set 0xABCD5678, %l0`
- `sra %l0, 11, %l0`

Value in \( \%l0 \) is \( 0x\)___________________________ (2 points)

- `set 0xABCD5678, %l0`
- `sll %l0, 13, %l0`

Value in \( \%l0 \) is \( 0x\)___________________________ (2 points)

- `set 0xABCD5678, %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 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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>high memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Parameter Passing and Return Values (Local Stack Variables)

Write the equivalent unoptimized SPARC assembly language instructions to perform the following C code fragment. All local variables must be allocated on the run time stack. (12 points)

C

/* Function Prototype */
short foo( short, int, char );
/* ... Other code ... */

/* Local stack variables */
short a;
char b[5];
short c;
int d[3];
/* ... Other code ... */

C

Put your SPARC Assembly code in the box below.

/*
Write the code for just this
function call saving the
return value appropriately
*/
c = foo( a, d[2], b[1] );

SPARC assembly
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 ) {
    int *local_stack_var1;
    int  local_stack_var2[3];
    local_stack_var2[0] = y; /* statement 1 */
    local_stack_var1 = &local_stack_var2[1]; /* statement 2 */
    *local_stack_var1 = 54321; /* statement 3 */
    x = *local_stack_var1++; /* statement 4 */
    return ( x - 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. (15 points)

```
SPARC assembly
.globl  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 0x0000000X

c: .byte 0xEE
.align 2
s: .half 0x89AB
.align 4
i1: .word 0x87654321
i2: .word 0x87654321
i3: .word 0x87654321
x: .word 0

.section ".text"
main:
  save  %sp, -96, %sp
  set   i1, %l0
  set   s, %l1
  ldhb  [%l1], %l2
  sth   %l2, [%l0+2]
  set   fmt, %o0
  ld    [%l0], %o1
  call  printf
  nop

  set   i2, %l0
  set   c, %l1
  ldsb  [%l1], %l2
  sth   %l2, [%l0]
  ldsb  [%l0+3], %l3
  stb   %l3, [%l0+2]
  set   fmt, %o0
  ld    [%l0], %o1
  call  printf
  nop

  set   i3, %l0
  set   x, %l1
  ldsb  [%l0], %l2
  sth   %l2, [%l1+2]
  ldsb  [%l0+3], %l3
  stb   %l3, [%l1]
  ldsb  [%l0+2], %l4
  stb   %l4, [%l1+1]
  mov   %l1, %l0
  set   fmt, %o0
  ld    [%l0], %o1
  call  printf
  nop

  ret
  restore
```
Extra Credit (7 points)

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

```c
#include <stdio.h>

int main()
{
    char a[] = "CSE30";
    char *p = a;

    printf( "%c", *p++ );  ______
    printf( "%c", ++*p );  ______
    printf( "%c", *++p + 1 ); ______
    p++;
    printf( "%c", *p = *p + 3); ______
    printf( "%c", --*p++ ); ______
    printf( "%d", p - a ); ______
    printf( "%s", a );  _____________________
    return 0;
}
```

A portion of the Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</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</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>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>
Scratch Paper