**Login name __________________** **Quiz 4** **Name ________________________**

# CSE 131

**Signature ____________________** **Winter 2009** **Student ID ____________________**

**1.** Project II Code Gen – Phase II.3:
What is the output of the following Reduced-C program:

```
function : int foo( int & x, int y )
{
  int z;

  x = y + 5;
  y = x + 5;
  z = x + y;

  cout << x << endl;
  cout << y << endl;
  cout << z << endl;

  return x;
}

function : int main( )
{
  int a = 5;
  int b = 10;
  int c;

  c = foo( a, b );

  cout << a << endl;
  cout << b << endl;
  cout << c << endl;

  return 0;
}
```

| Output |
|---|
| |

Assume variables a, b, and c in main() are allocated space in main()'s stack frame at memory locations

```
a   %fp-4
b   %fp-8
c   %fp-12
```

Write the SPARC assembly instructions for the line

```
c = foo( a, b );
```

You can assume all the initializations of the local variables have been performed. Just write the code to pass the actual arguments a and b to the function foo() and store the return value in c.

Assume the formal parameters x and y are allocated space in foo()'s stack frame at memory locations

```
x   %fp+68
y   %fp+72
```

Write the SPARC assembly instructions for the line

```
return x;
```

(over)

**2.** Pick one of the following letters to answer the questions below related to most calling conventions.

A) Caller B) Callee

_____ Allocates space for actual arguments

_____ Retrieves return value from return value location

_____ Allocates space for local variables

_____ Copies actual arguments into argument space

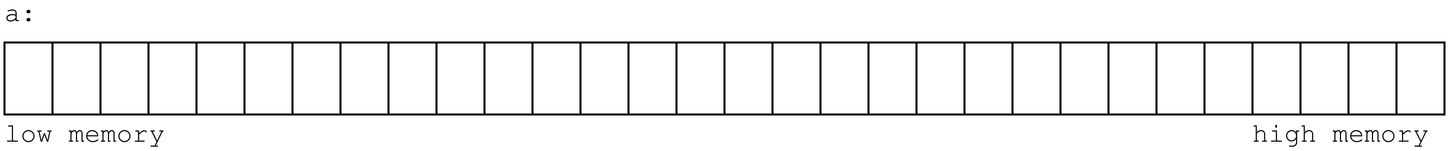
_____ Saves %pc into the return address location

_____ Retrieves saved return address for return

_____ Performs initialization of local variables

_____ Saves registers in callee-save scheme

**3.** Given the following C array declaration `short a[4][3];` mark with an **A** the memory location(s) where we would find `a[3][1]`

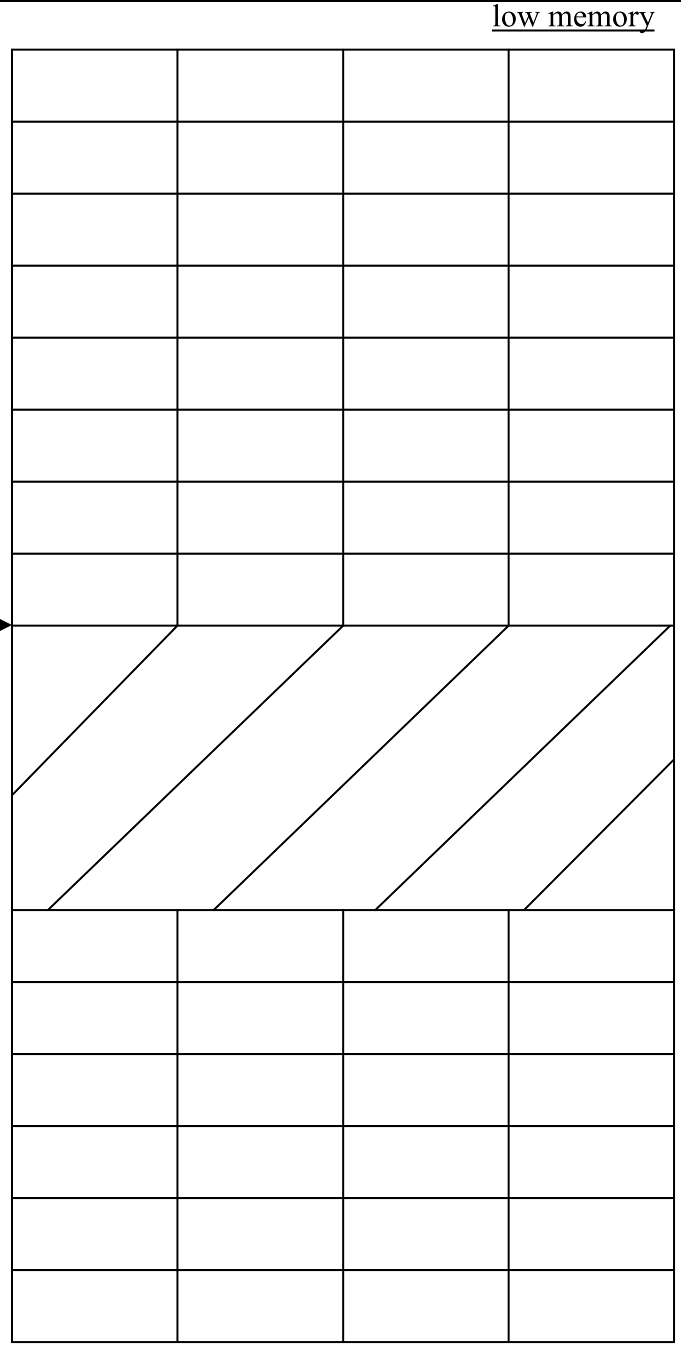


Each box represents a byte in memory.

**4.** Given the following C function definition

```
void foo( int a, int b, int c )
{
    int    d;
    short  e;
    char   f[3];
    double g;
    int    h;

    /* function body */
}
```



Show the **SPARC** memory layout of the stack frame for foo() taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate local variable and parameter name. For example, if variable or parameter name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the variable is an array, use the name followed by the index number. For example, some number of `p[0]`s, `p[1]`s, `p[2]`s, etc. Place an `X` in any bytes of padding. Use the Sun C compiler model. Do not allocate unneeded padding similar to how gcc puts extra padding between local variables. There may be more memory slots than needed, so do not feel like you have to fill them all.