Login name __________________

Quiz 4
CSE 131
Spring 2008

Name ________________________

Signature ____________________

Student ID ___________________

1. Project II Code Gen – Phase I.1:
Fill in the SPARC Assembly code that might be generated by a compiler this quarter for the following Reduced-C program:

```
bool b;

function : int main() {

 /* code that may change b */

 cout << b << endl;

 return 45;
}
```

```
        .section ".rodata"
        .align 4
TRUE:   .asciz "true"
FALSE:  .asciz "false"
ENDL:   .asciz "\n"

        .section ".bss"

        .align ________
b:      .skip 4

        .section __________
        .align 4
        .global main
main:
        set SAVE.main, %g1

        _______ %sp, %g1, %sp

    /* Code that may change b */

        set b, %l0

        _______ [%l0], %l0

        _______ %l0, %g0

        _______ .L1
        nop

        set TRUE, %o0

        _______ .L2
        nop
.L1:
        set FALSE, %o0
.L2:
        call _________
        nop

        set ENDL, _______
        call printf

        ________

        set 45, _________
        ret

        ___________

SAVE.main = -(92 + 0) & -8
```

2. Pick one of the following letters to answer the questions below.

A) Pre-Call B) Function Prologue
C) Function Epilogue D) Post-Return

_____ Where parameter space is allocated

_____ Performs initialization of local variables

_____ Store return value in %i0 in SPARC subroutine

_____ Restores callee-save registers

_____ Where local variable space is allocated

_____ Retrieve return value from %o0 in SPARC subroutine

_____ Retrieves saved return address

_____ Where local variable space is deallocated

_____ Saves caller-save registers

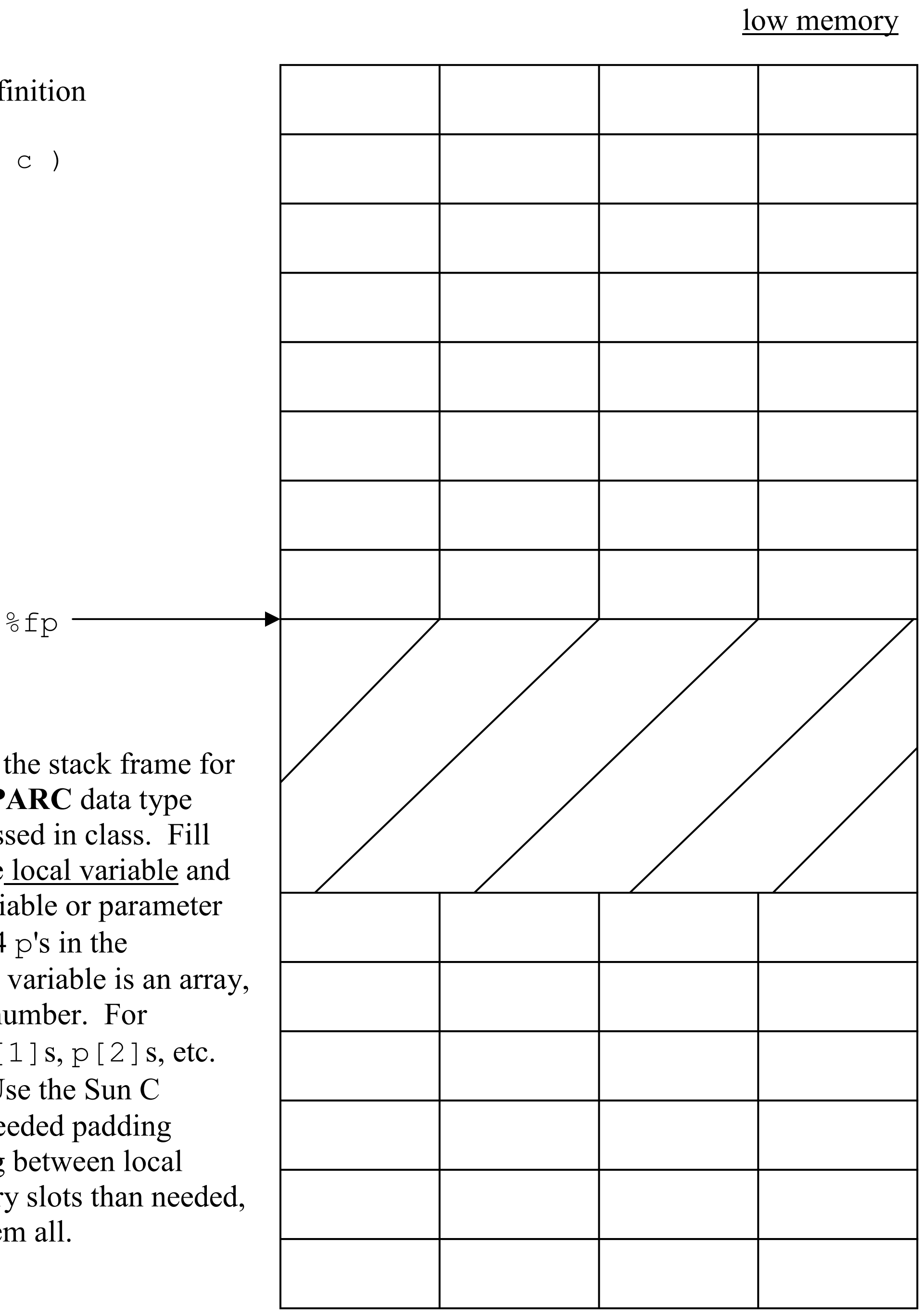
_____ Saves the return address

_____ Where parameter space is deallocated

3. Given the following C function definition

```
void foo( int a, int b, int c )
{
    short  d;
    int    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.