1. Consider this simple C program:

```c
int fibonacci(int *n2, int *n1, int x)
{
    /* printf("fib(%d,%d,%d)\n", *n2, *n1, x); */
    int n0 = *n1 + *n2;
    *n2 = *n1;
    *n1 = n0;
    if (x <= 3) return n0;
    else return fibonacci(n2, n1, --x);
}

int main()
{
    int n1 = 1;
    int n2 = 1;
    printf("%d\n", fibonacci(&n2, &n1, 4));
}
```

The Intel x86 assembly language for this program can be found at http://www.mscs.mu.edu/~brylow/cosc170/Spring2009/Homework/HW2/hw2-p1.s

Draw the stack of activation records for this program, showing the final contents of each location when the stack is at its largest point. Label each word in the stack that you can identify. Some locations in the stack may be undefined or unlabeled.

The first three words of the stack will look like this:

```
?  MAIN stack frame
   RET_OS
   BP_OS
   ...
```

Fill in the rest. The Pentium architecture manuals at http://www.mscs.mu.edu/~brylow/cosc065/Fall2008/IA32-OpcodeRef-2{A,B}.pdf may help you identify unfamiliar opcodes.
2. The source code from problem 1 can be downloaded from
http://www.mscs.mu.edu/~brylow/cosc170/Spring2009/Homework/HW2/hw2-p1.c
Login to Morbius and run the MIPS cross-compiler to generate MIPS assembly lan-
guage for hw2-p1.c.

/usr/local/project/mipsel-dev/bin/mipsel-gcc -S hw2-p1.c

Draw the stack of activation records for this program, showing the final contents of
each location when the stack is at its largest point. Label each word in the stack that
you can identify. Some locations in the stack may be undefined or unlabeled.

The first three words of the stack will look like this:

```
<table>
<thead>
<tr>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN stack frame</td>
</tr>
<tr>
<td>RA_GS</td>
</tr>
<tr>
<td>Return address of Operating System</td>
</tr>
<tr>
<td>FP_GS</td>
</tr>
<tr>
<td>Frame Pointer from Operating System</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```

Fill in the rest.

3. (a) By convention, the MIPS machine prefers to pass function arguments in design-
nated argument registers. Why does the program in question 2 store the values
of fibonacci’s arguments on the stack during execution?

(b) Give an example of a C program that requires the MIPS machine to spill function
arguments into the stack frame. Verify your example with the MIPS compiler.
Where does gcc put these arguments on the stack in relation to your callee’s stack
frame?

(c) Each MIPS routine begins with a line like, “addiu sp, sp, -x”, where x is the
size of the stack frame to allocate. What formula does gcc seem to be using to
calculate x?