CS352 - Compilers: Principles and Practice
Homework 2 - When Stack Frames Attack
Due: 2004 November 30, beginning of lecture

1. Consider this simple C program:

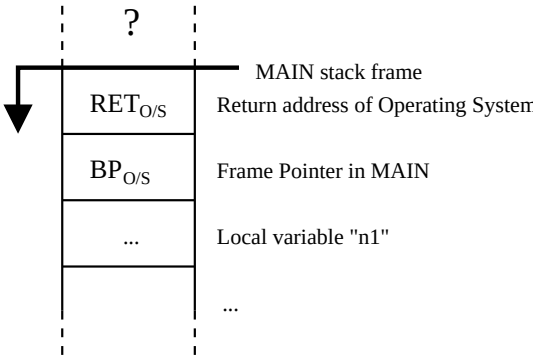
```
int fibonacci(int *n2, int *n1, int x)
{
  /* printf("fib(%d,%d,%d)\n", *n2, *n1, x); */
  int n0 = *n1 + *n2;
  *n2 = *n1;
  *n1 = n0;
  if (x <= 3) return n0;
  else return fibonacci(n2, n1, --x);
}

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.cs.purdue.edu/homes/brylow/cs352/Fall2004/Homework/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:



Fill in the rest. The Pentium architecture manuals at

`lore:/homes/cs352/Docs/Pentium-Reference-{1,2}.pdf`

may help you identify unfamiliar opcodes.

2. The source code from problem 1 can be downloaded from

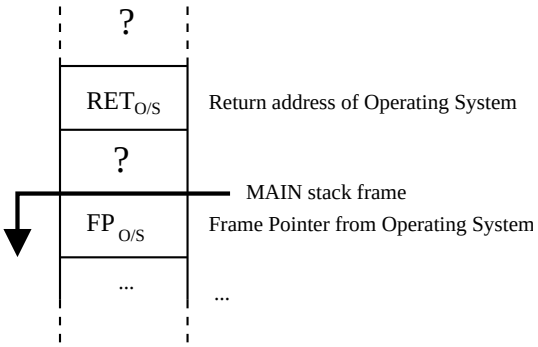
   `http://www.cs.purdue.edu/homes/brylow/cs352/Fall2004/Homework/hw2-p1.c`

   Login to Seven.cs.purdue.edu and run gcc to generate the PowerPC assembly language for hw2-p1.c.

   ```
   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:



   Fill in the rest. The PowerPC architecture manual at

   `lore:/homes/cs352/Docs/PowerPC-Reference.pdf`

   may help you identify unfamiliar opcodes.

3. (a) By convention, the PowerPC machine prefers to pass function arguments in designated 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 PowerPC machine to *spill* function arguments into the stack frame. Verify your example with the PowerPC compiler. Where does gcc put these arguments on the stack in relation to your callee's stack frame?

   (c) Each PowerPC routine begins with a line like, "`stwu 1, -x(1)`", where x is the size of the stack frame to allocate. What formula does gcc seem to be using to calculate x?