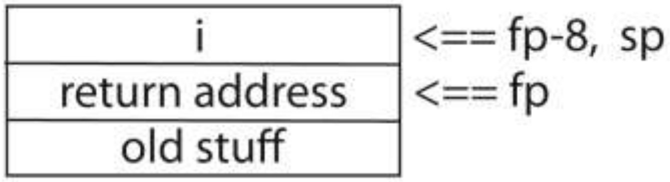
Example: The following program is an assembly language version of

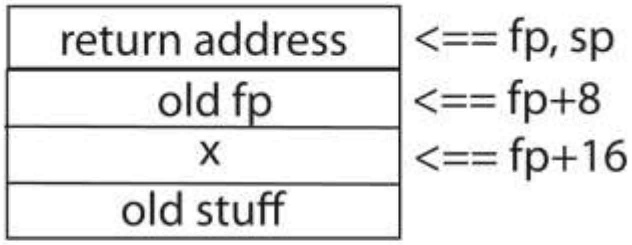
```
int f(int x) {
        return 2*x;
}

void main(  void ) {
        int i;
        i = 0;
        while (i < 10 ) {
                write( i );
                write( f(i) );
                writeln( );
                i = i + 1;
        }
}
```

This is hand-generated code, not code generated by my BPL compiler (which is much less efficient). You should be able to follow the code instruction by instruction. Note that this uses %rsp as the stack pointero and %rbx as the frame pointer. The following diagrams show the stack during the calls to main( ) and f( ):



The frame for main( )



The frame for f(x)

.

```
.section   .rodata
.WriteIntString: .string "%d "
.WritelnString: .string "\n"
.text
.globl main

f:
        movq %rsp, %rbx                # set up the frame pointer
        movq 16(%rbx), %rax            # argument value
        imul $2, %eax                  # performing multiplication
        ret                            # return from the function

main:
        movq %rsp, %rbx                # set up the frame pointer
        sub $8, %rsp                   # allocate local variable i
        movl $0, %eax                  # putting value into ac
        movl %eax, -8(%rbx)            # assign to i
.L0:
        cmpl $10, -8(%rbx)             # compare i and 10
        jge .L1                        # if i >= 10 leave the loop
        movl -8(%rbx), %esi            # value to print  (arg2 for the call)
        movq $.WriteIntString, %rdi
        movl $0, %eax                  # clear the return value
        call printf                    # call the C-lib printf function
        push -8(%rbx)                  # pushing argument for the call to f
        push %rbx                      # pushing the frame pointer
        call f                         # calling the function
        pop %rbx                       # retrieving the frame pointer
        add $8, %rsp                   # removing args from the stack
        movl %eax, %esi                # value to print (arg2 for the call)
        movq $.WriteIntString, %rdi
        movl $0, %eax                  # clear the return value
        call printf                    # call the C-lib printf function
        movq $.WritelnString, %rdi
        movl $0, %eax                  # clear the return value
        call printf                    # call the C-lib printf function
        movl -8(%rbx), %eax            # value of i
        addl $1, %eax                  # performing addition
        movl %eax, -8(%rbx)            # assign
        jmp .L0                        # WHILE: jump back to top
.L1:
         add $8, %rsp                  # deallocate local variables
         ret                           # return from the function
```