Function Call

The C function call is translated into assembly language using "call" assembly instruction. When the call instruction is executed, it pushes the address of the next instruction on the stack and then passes the control to the first instruction of the function which is called. The C return statement (there is an implicit return statement at the end of C void function) is translated using "ret" assembly instruction. The ret does the reverse of call instruction. It pops the stack to get the address and jumps to the address. So this is very important to make sure the top of the stack contains the correct return address when executing the return instruction. This is the responsibility of the compiler to generate the bug free code so that it will not mess-up with the return address.
Lets us see an example of how gcc generate code for function call

C code:

```
void fun()
{
   int x = 0;
   x++;
}
int main()
{
   fun();
}
```

Generated assembly code

```
.text
.globl fun
fun:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   movl    $0, -4(%ebp)
   addl    $1, -4(%ebp)
   leave
   ret
.globl main
main:
   pushl   %ebp
   movl    %esp, %ebp
   call    fun
   popl    %ebp
   ret
```

Comments on the generated code:

```
# .text segment starts
   .text

# export the function fun
.globl fun

# start of the function fun
fun:
   pushl   %ebp
   movl    %esp, %ebp
   subl    $16, %esp
   movl    $0, -4(%ebp)
   addl    $1, -4(%ebp)
   leave
   ret

# export the function main
.globl main

# main function start here
main:
   pushl   %ebp
   movl    %esp, %ebp

# main function calls fun
   call    fun

# main function return. it will restore the ebp register then return
   popl    %ebp
   ret
```

‹ Function Pointer (/cin/functionpointer.html)
up (/cin/cin.html)
Parameter Passing › (/cin/parameter.html)