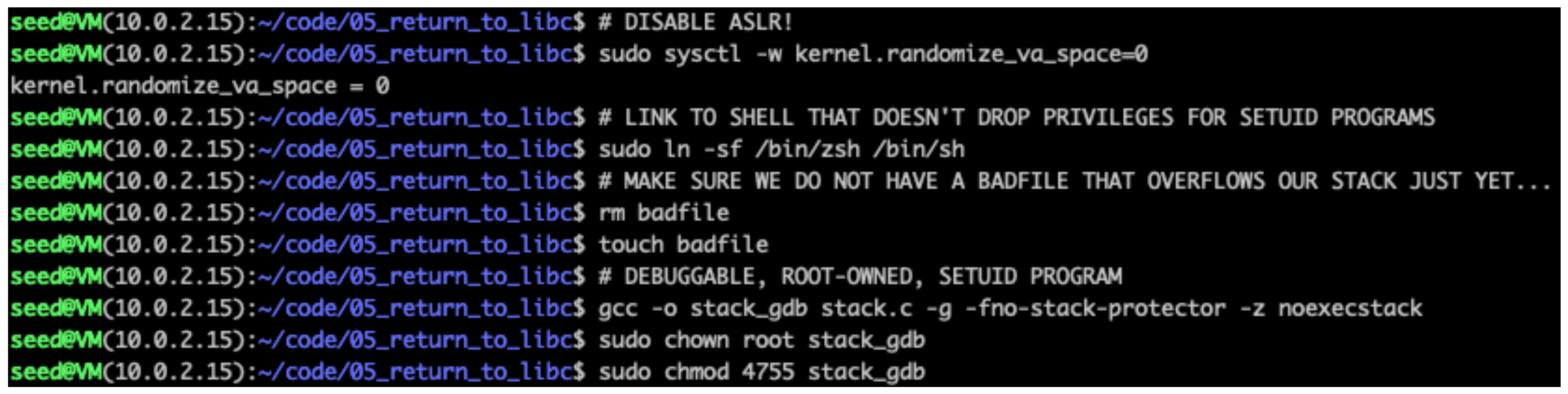
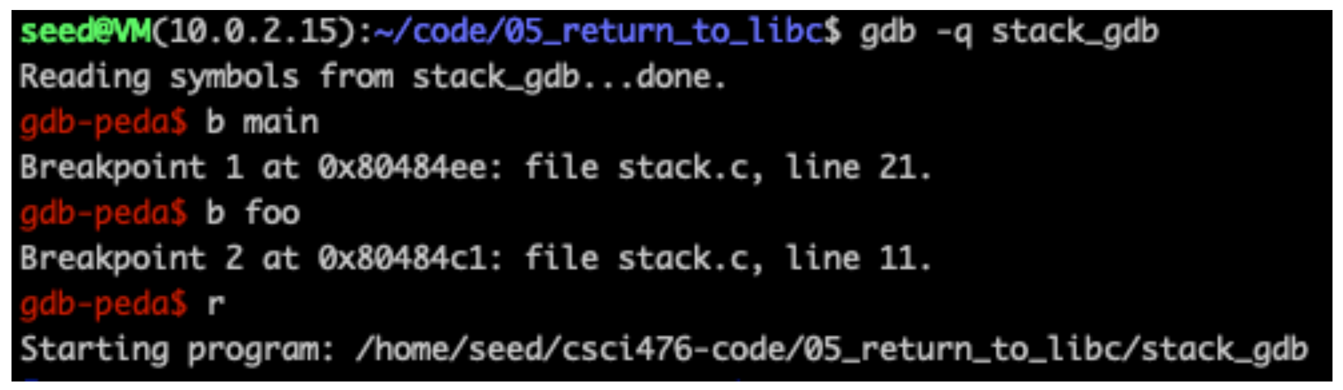
GDB Session: See How foo()'s stack frame is built:



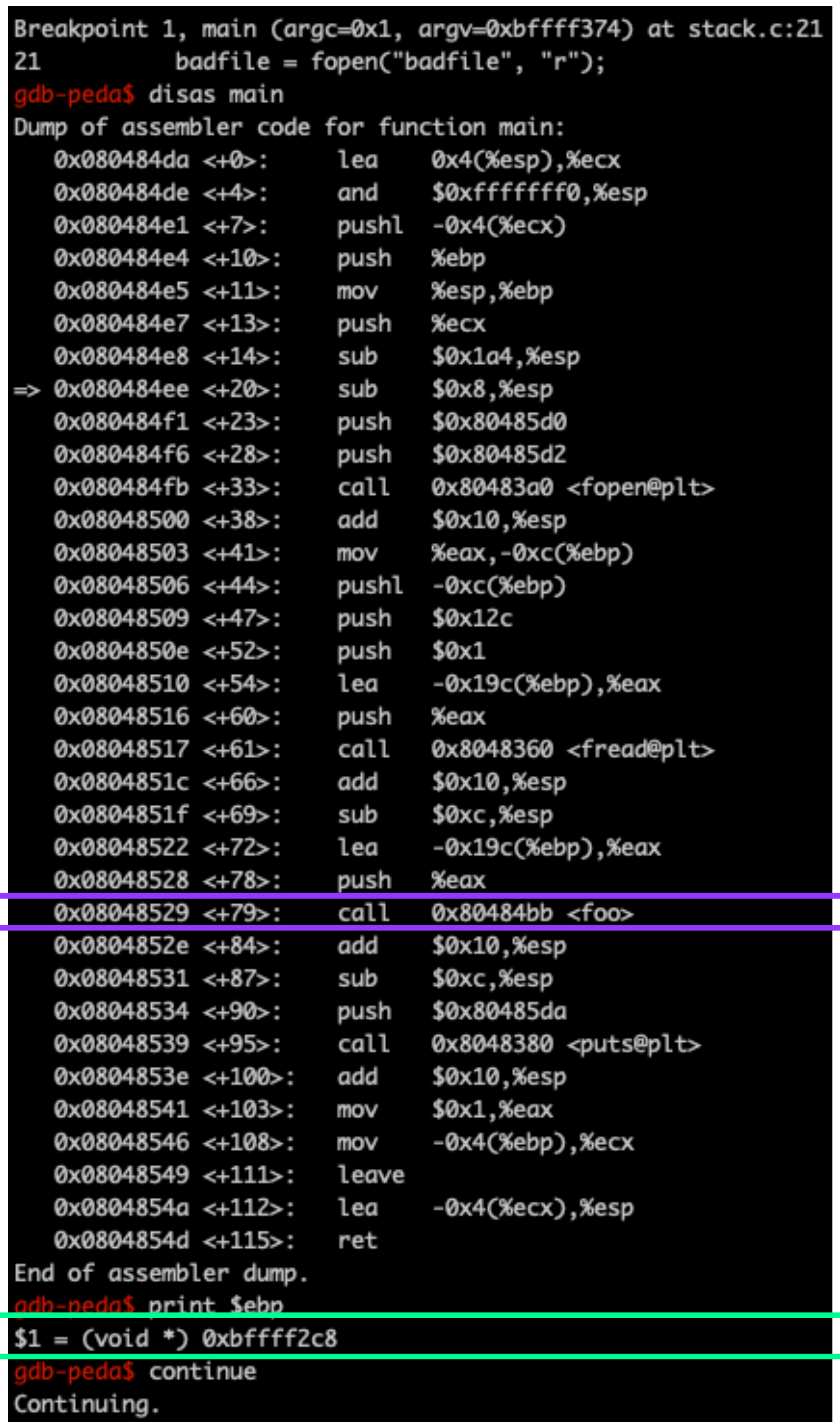
```
seed@VM(10.0.2.15):~/code/05_return_to_libc$ # DISABLE ASLR!
seed@VM(10.0.2.15):~/code/05_return_to_libc$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
seed@VM(10.0.2.15):~/code/05_return_to_libc$ # LINK TO SHELL THAT DOESN'T DROP PRIVILEGES FOR SETUID PROGRAMS
seed@VM(10.0.2.15):~/code/05_return_to_libc$ sudo ln -sf /bin/zsh /bin/sh
seed@VM(10.0.2.15):~/code/05_return_to_libc$ # MAKE SURE WE DO NOT HAVE A BADFILE THAT OVERFLOWS OUR STACK JUST YET...
seed@VM(10.0.2.15):~/code/05_return_to_libc$ rm badfile
seed@VM(10.0.2.15):~/code/05_return_to_libc$ touch badfile
seed@VM(10.0.2.15):~/code/05_return_to_libc$ # DEBUGGABLE, ROOT-OWNED, SETUID PROGRAM
seed@VM(10.0.2.15):~/code/05_return_to_libc$ gcc -o stack_gdb stack.c -g -fno-stack-protector -z noexecstack
seed@VM(10.0.2.15):~/code/05_return_to_libc$ sudo chown root stack_gdb
seed@VM(10.0.2.15):~/code/05_return_to_libc$ sudo chmod 4755 stack_gdb
```

Setup



```
seed@VM(10.0.2.15):~/code/05_return_to_libc$ gdb -q stack_gdb
Reading symbols from stack_gdb...done.
gdb-peda$ b main
Breakpoint 1 at 0x80484ee: file stack.c, line 21.
gdb-peda$ b foo
Breakpoint 2 at 0x80484c1: file stack.c, line 11.
gdb-peda$ r
Starting program: /home/seed/csci476-code/05_return_to_libc/stack_gdb
```

Start gdb.
Set **break points** at **foo** and **main**.
Run!



```
Breakpoint 1, main (argc=0x1, argv=0xbffff374) at stack.c:21
21          badfile = fopen("badfile", "r");
gdb-peda$ disas main
Dump of assembler code for function main:
   0x080484da <+0>:     lea    0x4(%esp),%ecx
   0x080484de <+4>:     and    $0xfffffff0,%esp
   0x080484e1 <+7>:     pushl  -0x4(%ecx)
   0x080484e4 <+10>:    push   %ebp
   0x080484e5 <+11>:    mov    %esp,%ebp
   0x080484e7 <+13>:    push   %ecx
   0x080484e8 <+14>:    sub    $0x1a4,%esp
=> 0x080484ee <+20>:    sub    $0x8,%esp
   0x080484f1 <+23>:    push   $0x80485d0
   0x080484f6 <+28>:    push   $0x80485d2
   0x080484fb <+33>:    call   0x80483a0 <fopen@plt>
   0x08048500 <+38>:    add    $0x10,%esp
   0x08048503 <+41>:    mov    %eax,-0xc(%ebp)
   0x08048506 <+44>:    pushl  -0xc(%ebp)
   0x08048509 <+47>:    push   $0x12c
   0x0804850e <+52>:    push   $0x1
   0x08048510 <+54>:    lea    -0x19c(%ebp),%eax
   0x08048516 <+60>:    push   %eax
   0x08048517 <+61>:    call   0x8048360 <fread@plt>
   0x0804851c <+66>:    add    $0x10,%esp
   0x0804851f <+69>:    sub    $0xc,%esp
   0x08048522 <+72>:    lea    -0x19c(%ebp),%eax
   0x08048528 <+78>:    push   %eax
   0x08048529 <+79>:    call   0x80484bb <foo>
   0x0804852e <+84>:    add    $0x10,%esp
   0x08048531 <+87>:    sub    $0xc,%esp
   0x08048534 <+90>:    push   $0x80485da
   0x08048539 <+95>:    call   0x8048380 <puts@plt>
   0x0804853e <+100>:   add    $0x10,%esp
   0x08048541 <+103>:   mov    $0x1,%eax
   0x08048546 <+108>:   mov    -0x4(%ebp),%ecx
   0x08048549 <+111>:   leave
   0x0804854a <+112>:   lea    -0x4(%ecx),%esp
   0x0804854d <+115>:   ret
End of assembler dump.
gdb-peda$ print $ebp
$1 = (void *) 0xbffff2c8
gdb-peda$ continue
Continuing.
```

Use **disas**(semble) **FUNC** to show assembly code for function FUNC inline

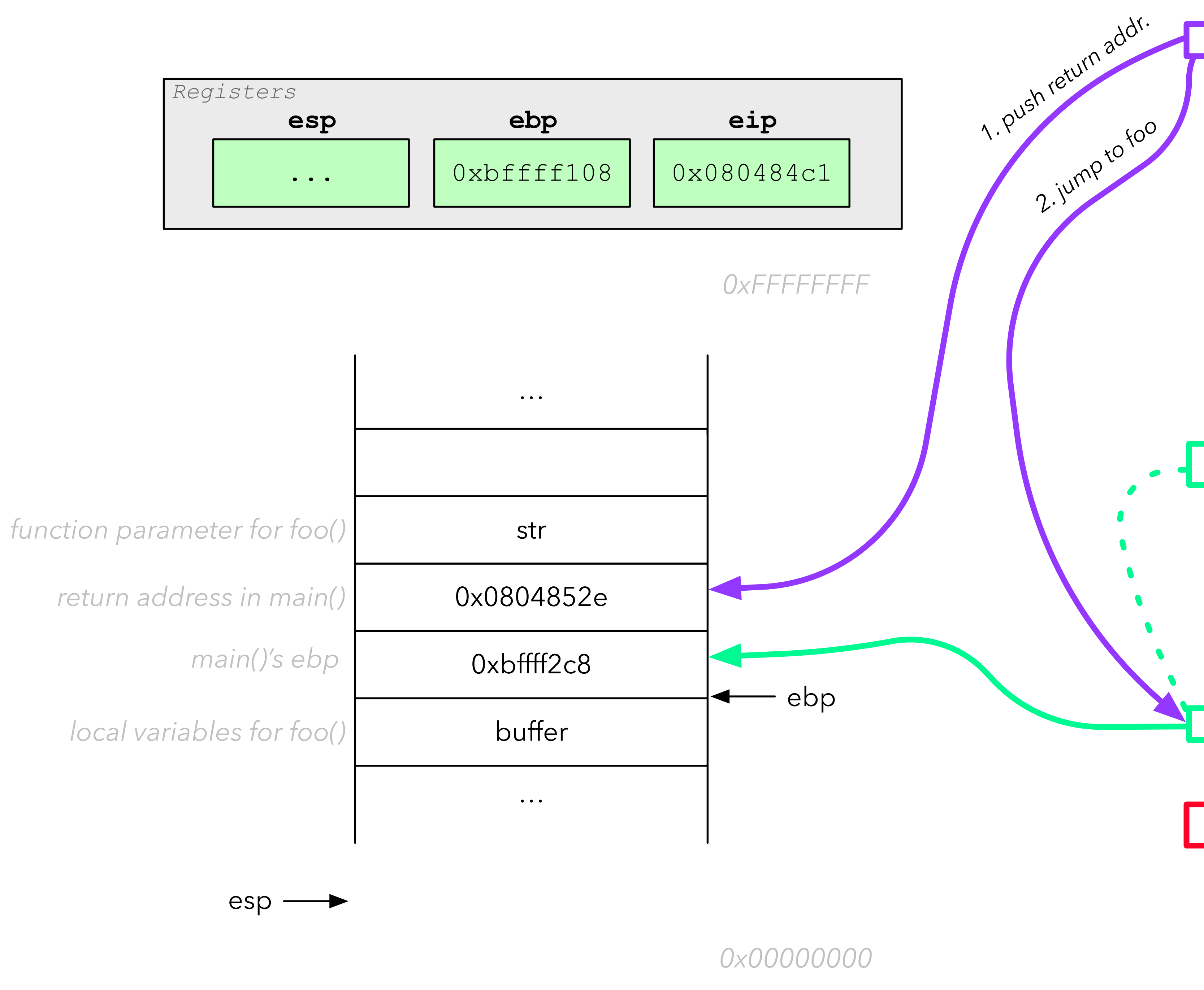
"**=>**" shows the current value of the instruction pointer **eip**
(this is simply where gdb has currently paused execution)

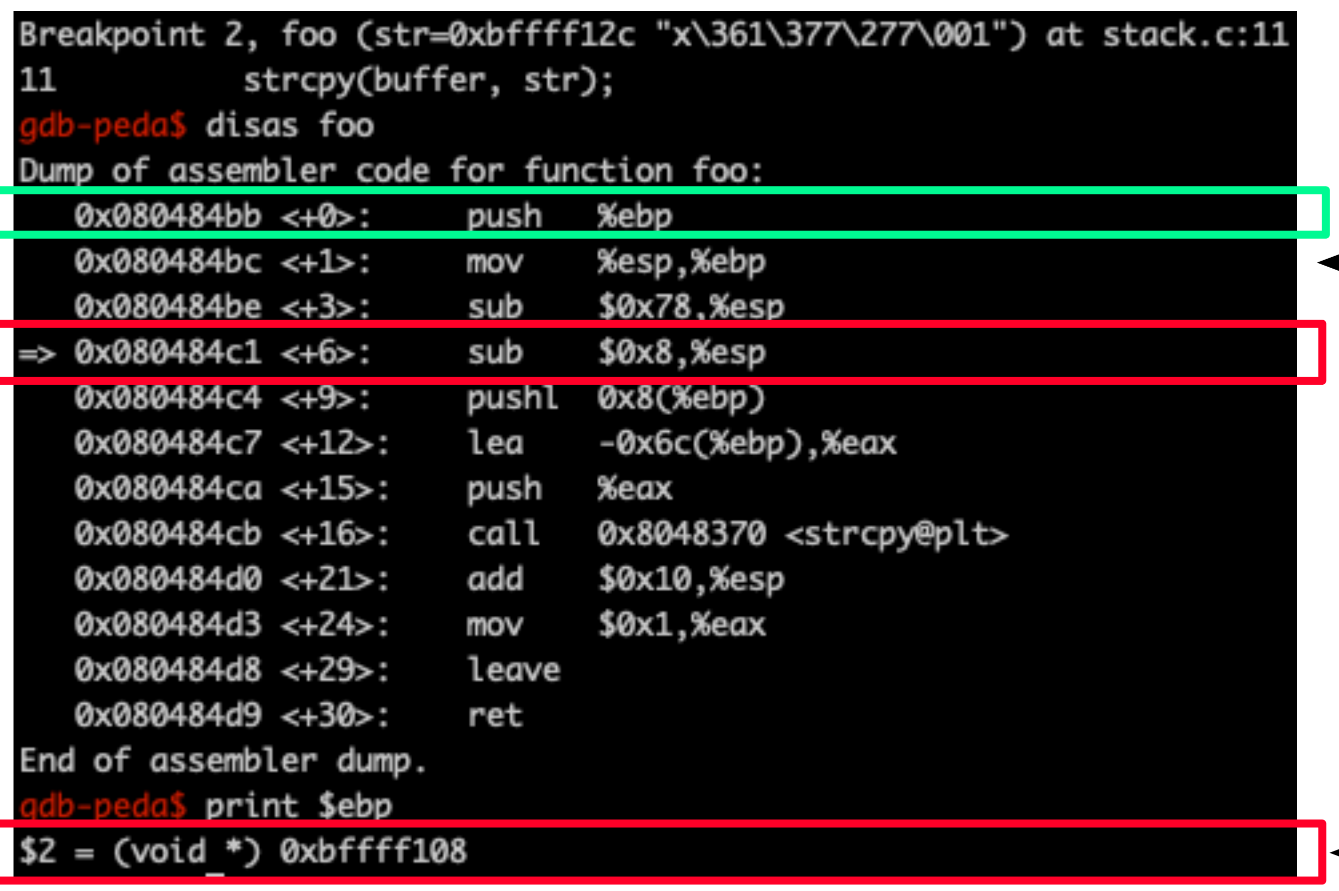
Also note:
instruction pointer (eip) == program counter (pc)

push the **address of str** onto the stack (stored in eax; the only argument for foo())
call = push **return address**, then jump to **call address** (foo() function)
To pick up execution in main() after returning from foo(),
the address just after call is pushed onto the stack as the return address

NOTE: this is **ebp** in **main()**

[Step 1]
foo()'s stack frame in stack.c
(AFTER function prologue)





```
Breakpoint 2, foo (str=0xbffff12c "x\361\377\277\001") at stack.c:11
11          strcpy(buffer, str);
gdb-peda$ disas foo
Dump of assembler code for function foo:
   0x080484bb <+0>:     push   %ebp
   0x080484bc <+1>:     mov    %esp,%ebp
   0x080484be <+3>:     sub    $0x78,%esp
=> 0x080484c1 <+6>:     sub    $0x8,%esp
   0x080484c4 <+9>:     pushl  0x8(%ebp)
   0x080484c7 <+12>:    lea    -0x6c(%ebp),%eax
   0x080484ca <+15>:    push   %eax
   0x080484cb <+16>:    call   0x8048370 <strcpy@plt>
   0x080484d0 <+21>:    add    $0x10,%esp
   0x080484d3 <+24>:    mov    $0x1,%eax
   0x080484d8 <+29>:    leave
   0x080484d9 <+30>:    ret
End of assembler dump.
gdb-peda$ print $ebp
$2 = (void *) 0xbffff108
```

After pusing **prev. ebp**, this call frame's **ebp is set to esp** (current position of stack pointer)

Note the address of the next instruction we will execute
now that execution has jumped to **foo()**.

This address is stored in **eip**.

NOTE: this is **ebp** in **foo()**