Software Security

Module 2 - C Call Stack & Pointers

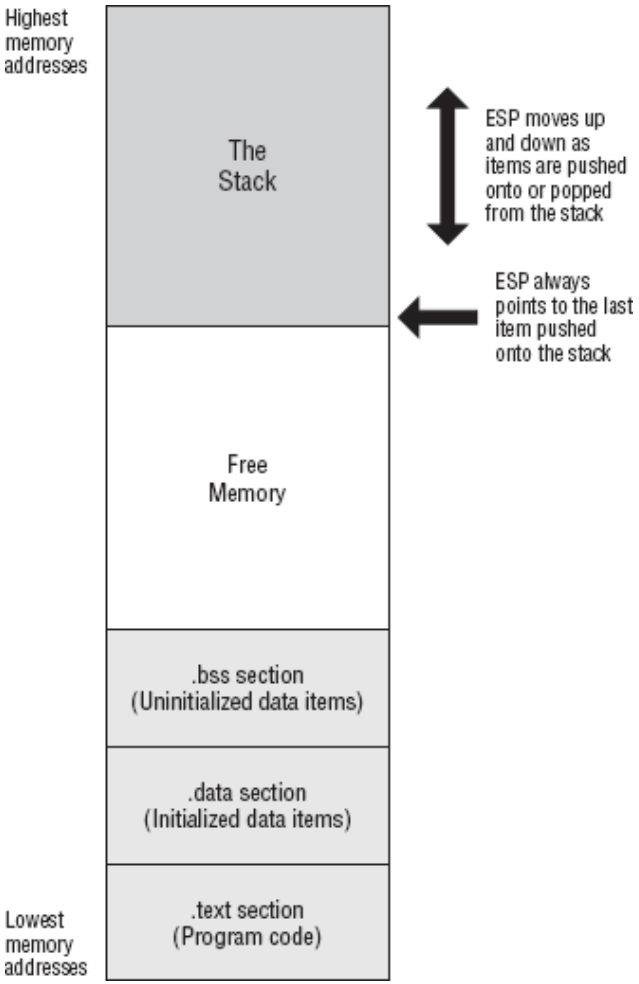
CMPE279
Software Security Technologies
San Jose State University

MEMORY LAYOUT

layout.c

```c

#include<stdio.h>
#include<malloc.h>

int glb_uninit;          /* Part of BSS Segment -- global uninitialized variable, at runtime i
int glb_init = 10;       /* Part of DATA Segment -- global initialized variable */

void  foo(void)
{
    static int num = 0;         /* stack frame count */
    int autovar;         /* automatic variable/Local variable */
    int *ptr_foo = (int*)malloc(sizeof(int));
    if (++num == 4)             /* Creating four stack frames */
        return;
    printf("Stack frame number %d: address of autovar: %p\n", num, & autovar);
    printf("Address of heap allocated inside foo() %p\n",ptr_foo);
    foo();                      /* function call */
}

int main()
{
    char *p, *b, *nb;
    int *ptr_main = (int*)malloc(sizeof(int));
    printf("Text Segment:\n");
    printf("Address of main: %p\n", main);
    printf("Address of func foo: %p\n",foo);
    printf("Stack Locations:\n");
    foo();
    printf("Data Segment:\n");
    printf("Address of glb_init: %p\n", & glb_init);
    printf("BSS Segment:\n");
    printf("Address of glb_uninit: %p\n", & glb_uninit);
    printf("Heap Segment:\n");
    printf("Address of heap allocated inside main() %p\n",ptr_main);

    return 0;
}

```

Line: 1 Column: 1 C Tab Size: 4



Group: all

eax	-1073744556
ecx	0xbffff4d0
edx	0x1
ebx	0xb7fc9ff4
esp	0xbffff490
ebp	0xbffff4b8
esi	0x8048570
edi	0x8048370
eip	0x8048497
eflags	0x200286
cs	0x73
ss	0x7b
ds	0x7b
es	0x7b
fs	0x0
gs	0x33

layout.c - Source Window

File Run View Control Preferences Help

Find:

layout.c | main | SOURCE

```
-     9 {
     10          static int num = 0;        /* stack frame count */
     11          int autovar;          /* automatic variable/Local variab
-    12          int *ptr_foo = (int*)malloc(sizeof(int));
-    13          if (++num == 4)            /* Creating four stack frame
     14          return;
-    15          printf("Stack frame number %d: address of autovar: %p\
-    16          printf("Address of heap allocated inside foo() %p\n",pt
-    17          foo();                     /* function call */
-    18 }
     19
     20 int main()
-    21 {
     22          char *p, *b, *nb;
■    23          int *ptr_main = (int*)malloc(sizeof(int));
-    24          printf("Text Segment:\n");
-    25          printf("Address of main: %p\n", main);
```

Program stopped at line 23 | 8048497 | 23

seed@seed-desktop: ~/CMPE279/modules/module2/stack

File Edit View Terminal Help

```
seed@seed-desktop:~/CMPE279/modules/module2/stack$ ./layout.out
Text Segment:
Address of main: 0x8048486
Address of func foo: 0x8048424
Stack Locations:
Stack frame number 1: address of autovar: 0xbffff4d4
Address of heap allocated inside foo() 0x804b018
Stack frame number 2: address of autovar: 0xbffff4a4
Address of heap allocated inside foo() 0x804b028
Stack frame number 3: address of autovar: 0xbffff474
Address of heap allocated inside foo() 0x804b038
Data Segment:
Address of glb_init: 0x804a01c
BSS Segment:
Address of glb_uninit: 0x804a02c
Heap Segment:
Address of heap allocated inside main() 0x804b008
seed@seed-desktop:~/CMPE279/modules/module2/stack$
```

*** not to scale ***

layout.c

```c
#include<stdio.h>
#include<malloc.h>

int glb_uninit;
int glb_init = 10;

void  foo(void)
{
    static int num = 0;
    int autovar;
    int *ptr_foo = (int*)malloc(sizeof(int));
    if (++num == 4)
        return;
    printf("Stack frame number %d: address of autovar: %p\n", num, & autovar);
    printf("Address of heap allocated inside foo() %p\n",ptr_foo);
    foo();
}

int main()
{
    char *p, *b, *nb;
    int *ptr_main = (int*)malloc(sizeof(int));
    printf("Text Segment:\n");
    printf("Address of main: %p\n", main);
    printf("Address of func foo: %p\n",foo);
    printf("Stack Locations:\n");
    foo();
    printf("Data Segment:\n");
    printf("Address of glb_init: %p\n", & glb_init);
    printf("BSS Segment:\n");
    printf("Address of glb_uninit: %p\n", & glb_uninit);
    printf("Heap Segment:\n");
    printf("Address of heap allocated inside main() %p\n",ptr_main);

    return 0;
}
```

REGISTER	CONTENTS
EIP	0x8048497
ESI	0x8048570
EDI	0x8048570
EFLAGS	
EAX	
EBX	
ECX	
EDX	
ESP	0xbffff490
EBP	0xbffff5b8

	ADDR	MEMORY
STACK		
	0xbffff4df	/** autovar #1 **/
	0xbffff4a4	/** autovar #2 **/
	0xbffff474	/** autovar #3 **/
UNUSED MEMORY		
HEAP	0x804b028	/** ptr_foo #2 **/
	0x804b018	/** ptr_foo #1 **/
	0x804b008	/** ptr_main **/
BSS		
	0x804a02c	/** glb_uninit **/
DATA		
	0x804a01c	/** glb_init **/
TEXT		
	0x8048497	/** main() **/
	0x8048424	/** foo() **/

EIP points to somewhere in Code Segment

seed@seed-desktop: ~/CMPE279/modules/module2/stack

File Edit View Terminal Help

```
seed@seed-desktop:~/CMPE279/modules/module2/stack$ ./layout.out
Text Segment:
Address of main: 0x8048486
Address of func foo: 0x8048424
Stack Locations:
Stack frame number 1: address of autovar: 0xbffff4d4
Address of heap allocated inside foo() 0x804b018
Stack frame number 2: address of autovar: 0xbffff4a4
Address of heap allocated inside foo() 0x804b028
Stack frame number 3: address of autovar: 0xbffff474
Address of heap allocated inside foo() 0x804b038
Data Segment:
Address of glb_init: 0x804a01c
BSS Segment:
Address of glb_uninit: 0x804a02c
Heap Segment:
Address of heap allocated inside main() 0x804b008
seed@seed-desktop:~/CMPE279/modules/module2/stack$
```

CDECL

C Calling Convention

CDECL / GCC/VSC++

C Calling Conventions

(note: callee = called procedure)

Caller saves: EAX, ECX, EDX

Callee saves: EBX, ESP, EBP, ESI & EDI

Caller pass parameters on stack from right to left. Example: Func(A, B, C). Push C, then B, and A.

Callee sets up new stack frame: push old frame pointer (EBP), adjusts EBP and pushes local variables on the stack.

Callee's return value will be in EAX (if <= 32-bits). If >32-bits, high bits in EDX.

Caller cleans up the stack.*Callee do not pop from Stack! Callers will pop or change SP offset upon return.*

cdecl [edit]

The **cdecl** (which stands for **C declaration**) is a calling convention that originates from the C programming language and is used by many C compilers for the x86 architecture.[1] In cdecl, subroutine arguments are passed on the stack. Integer values and memory addresses are returned in the EAX register, floating point values in the ST0 x87 register. Registers EAX, ECX, and EDX are caller-saved, and the rest are callee-saved. The x87 floating point registers ST0 to ST7 must be empty (popped or freed) when calling a new function, and ST1 to ST7 must be empty on exiting a function.

In context of the C programming language, function arguments are pushed on the stack in the reverse order. In GNU/Linux, GCC sets the *de facto* standard for calling conventions. Since GCC version 4.5, the stack must be aligned to a 16-byte boundary when calling a function (previous versions only required a 4-byte alignment.)[*citation needed*]

Consider the following C source code snippet:

```
int callee(int, int, int);

int caller(void)
{
    int ret;

    ret = callee(1, 2, 3);
    ret += 5;
    return ret;
}
```

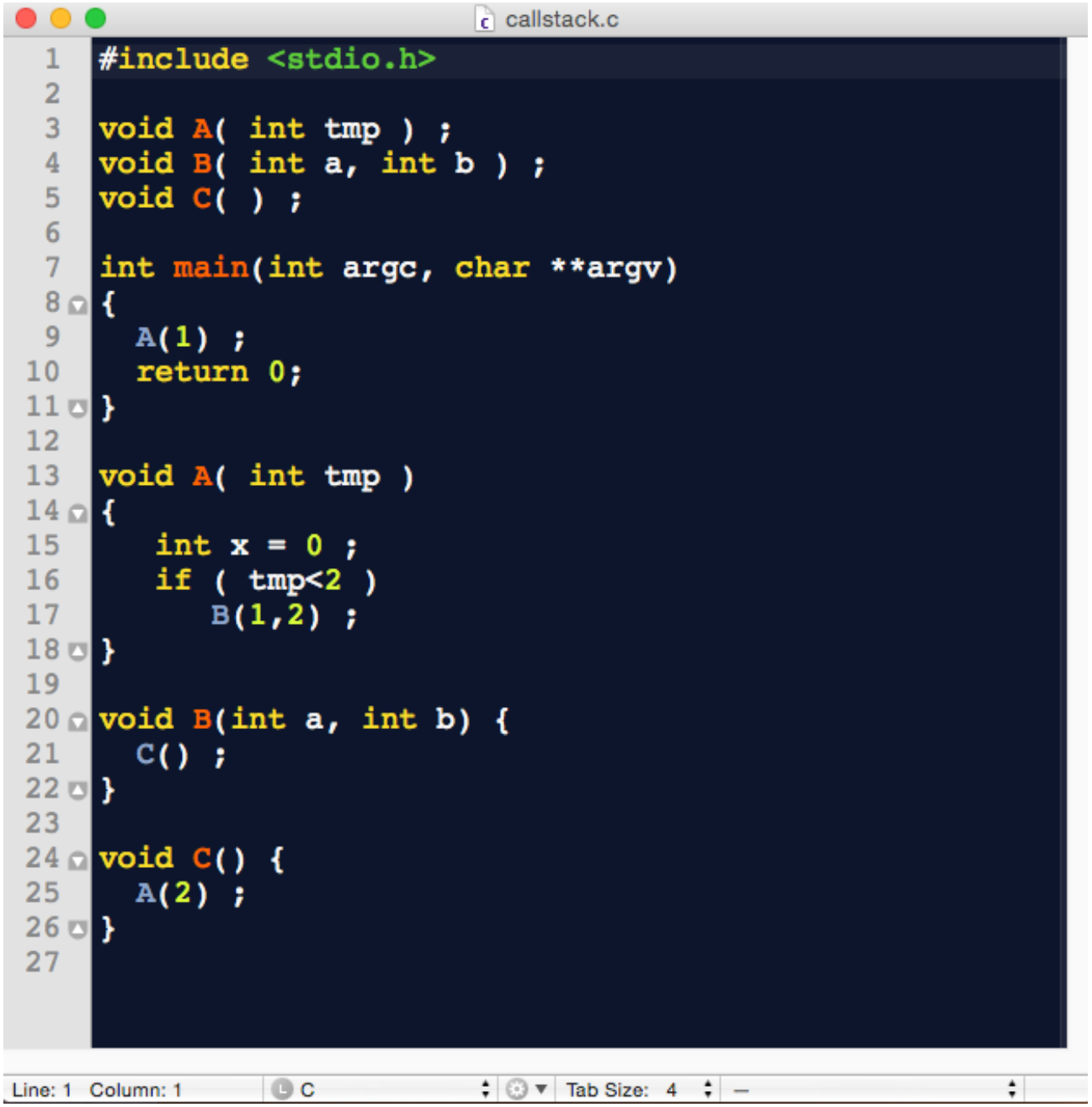
On x86, it will produce the following assembly code (Intel syntax):

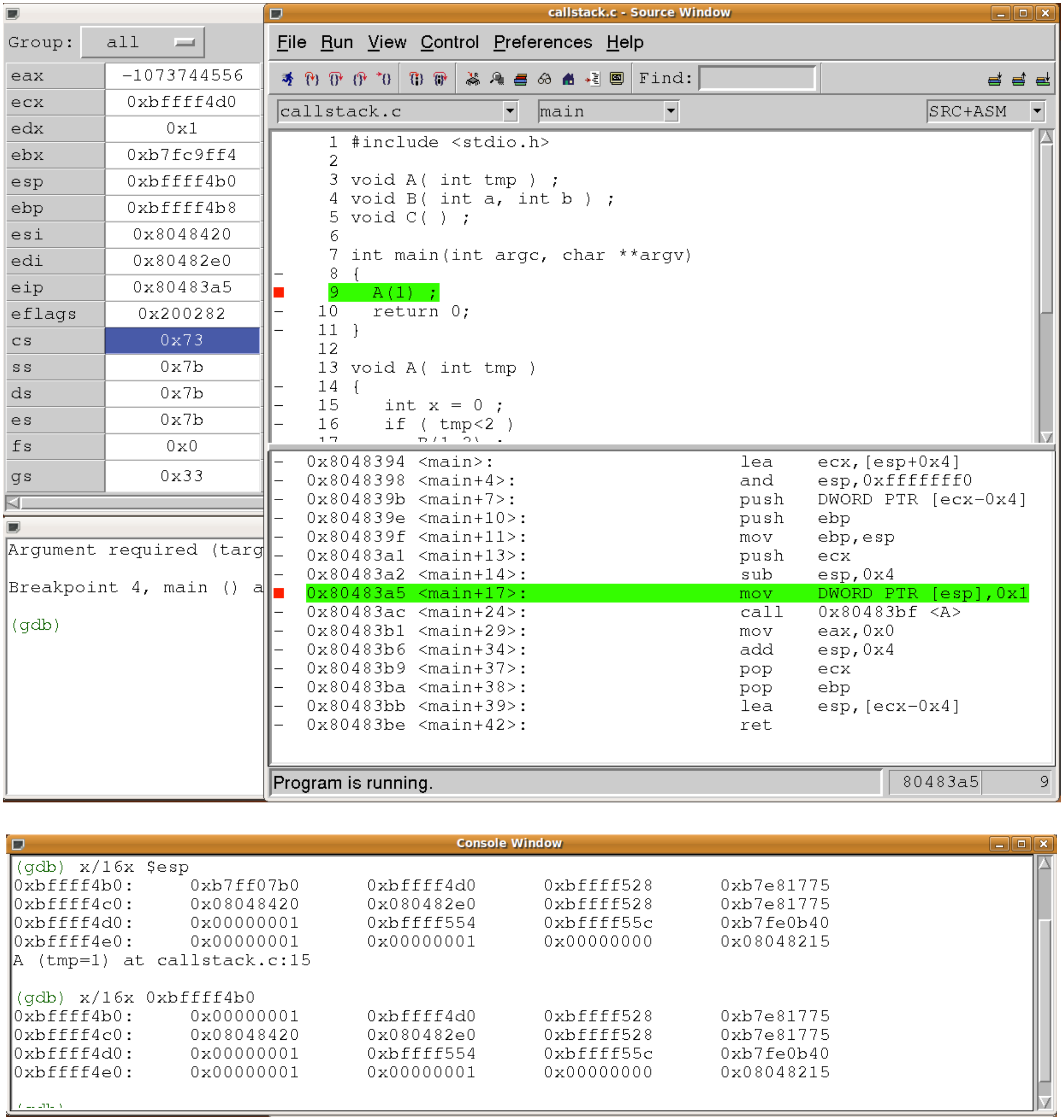
```
caller:
    push    ebp
    mov     ebp, esp
    push    3
    push    2
    push    1
    call    callee
    add     esp, 12
    add     eax, 5
    pop     ebp
    ret
```

The caller cleans the stack after the function call returns.

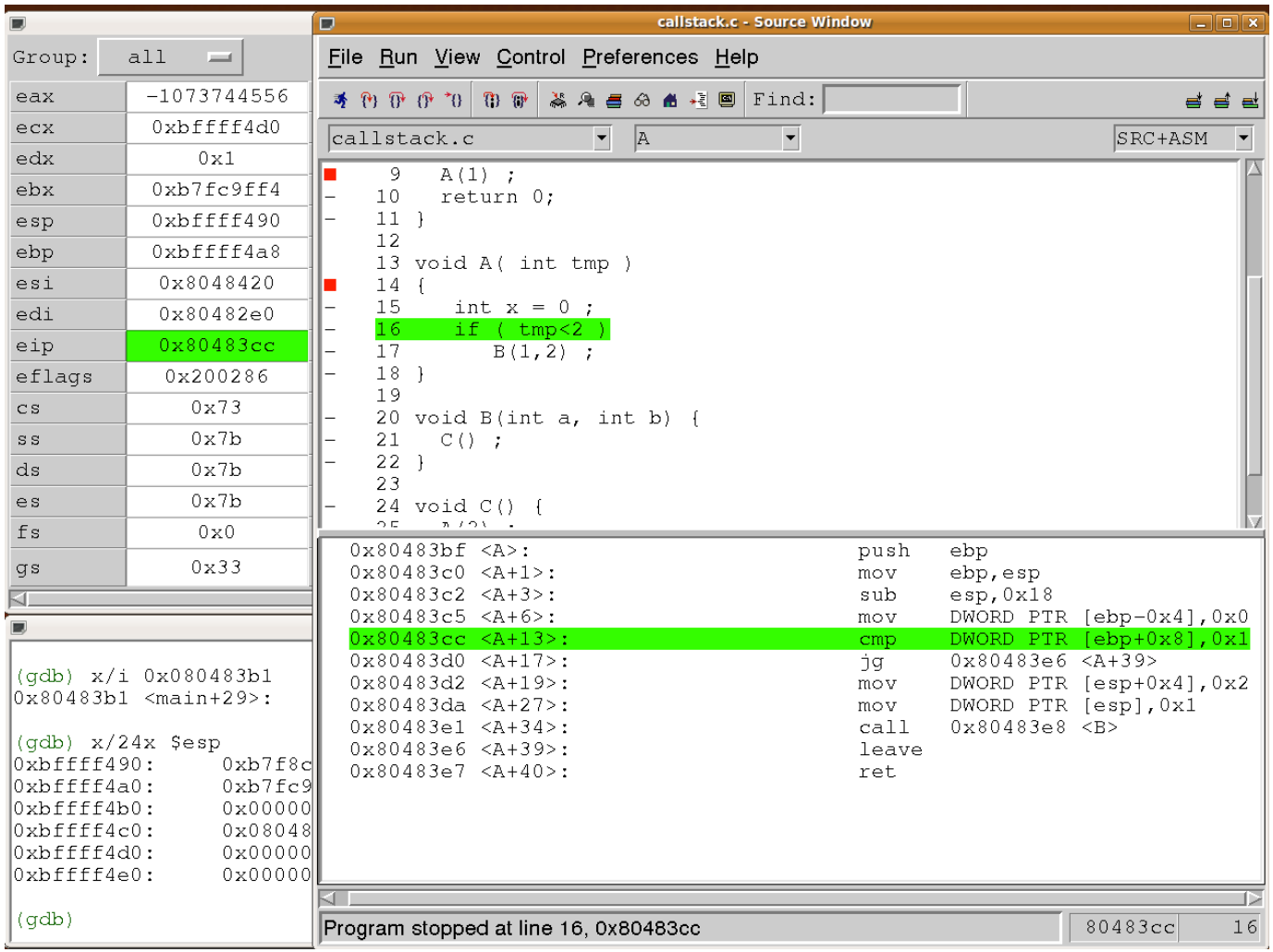
REF: http://en.wikipedia.org/wiki/X86_calling_conventions

C CALL STACK

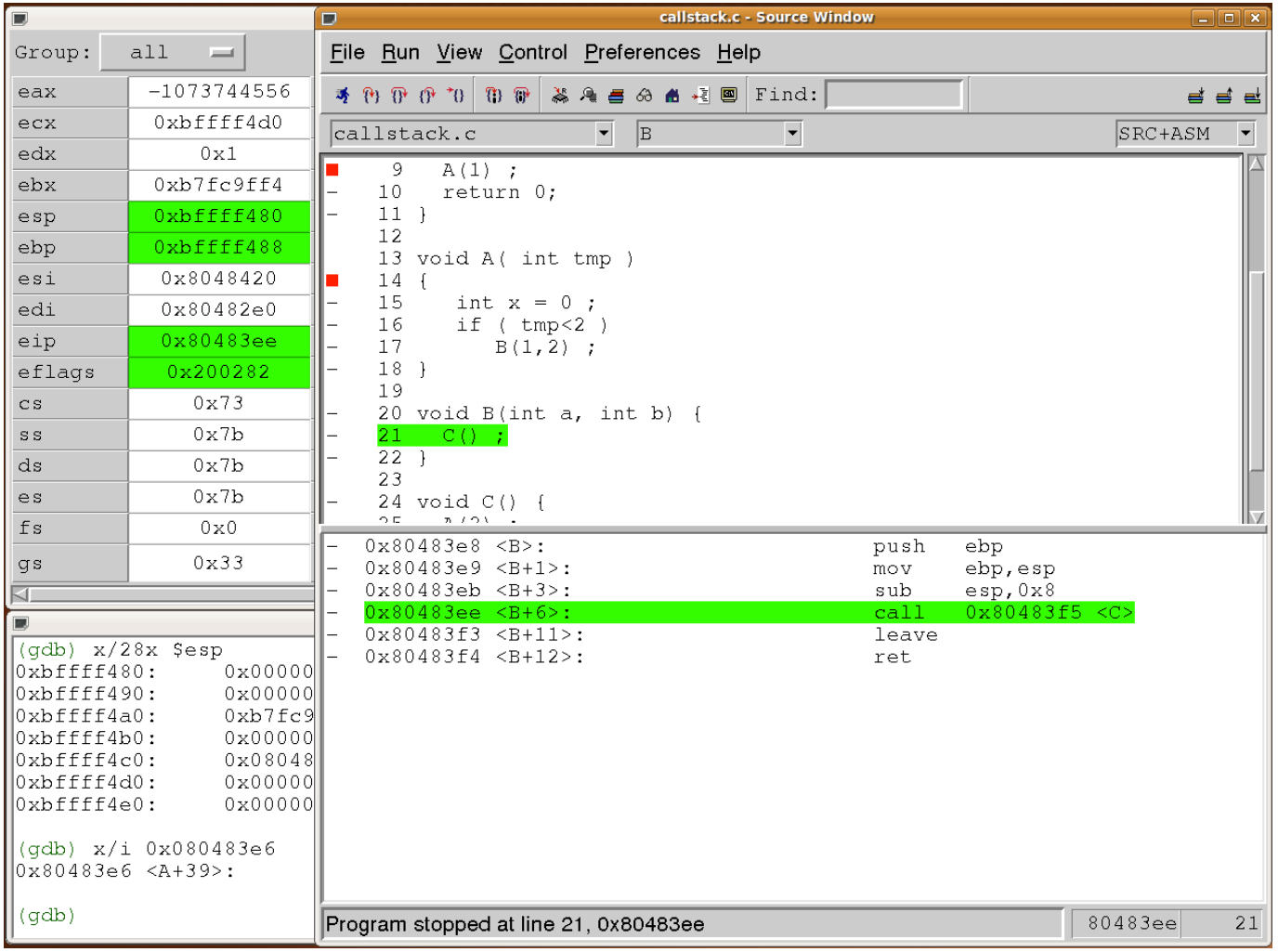

callstack.c
1 #include <stdio.h>
2
3 void A(int tmp) ;
4 void B(int a, int b) ;
5 void C( ) ;
6
7 int main(int argc, char **argv)
8 {
9 A(1) ;
10 return 0;
11 }
12
13 void A(int tmp)
14 {
15 int x = 0 ;
16 if (tmp<2)
17 B(1,2) ;
18 }
19
20 void B(int a, int b) {
21 C() ;
22 }
23
24 void C() {
25 A(2) ;
26 }
27
Line: 1 Column: 1
C
Tab Size: 4

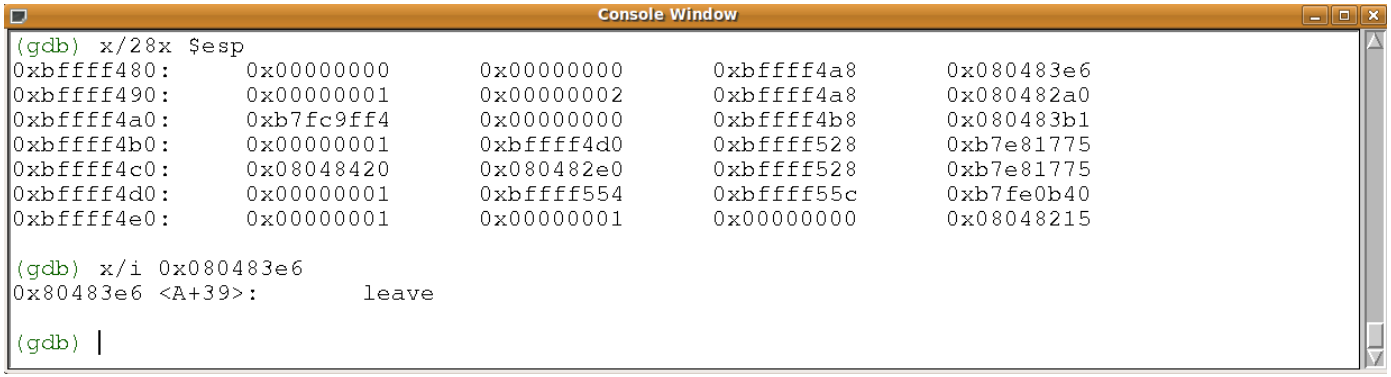


	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001

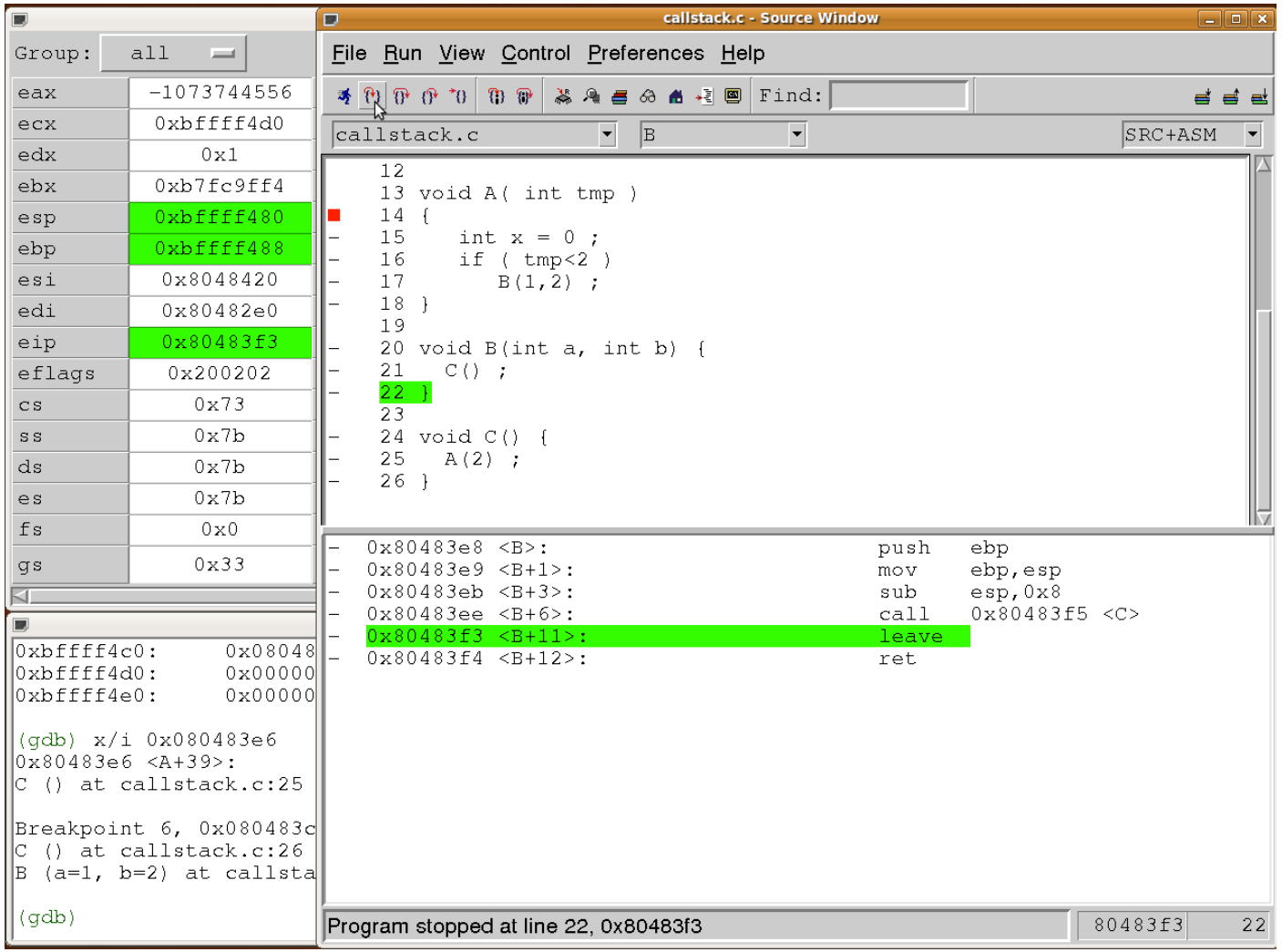


	ADDR	MEMORY
main: ebp	0xbfffff4b8	0xbfffff528
	0xbfffff4b4	0xbfffff4d0
main: esp	0xbfffff4b0	0x00000001
	0xbfffff4ac	0x080483b1
A: ebp	0xbfffff4a8	0xbfffff4b8
ebp - 4	0xbfffff4a4	0x00000000
ebp - 8	0xbfffff4a0	
ebp - 12	0xbfffff49c	
ebp - 16	0xbfffff498	
ebp - 20	0xbfffff494	
A: esp	0xbfffff490	





	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001
ret: main	0xbffff4ac	0x080483b1
A: ebp	0xbffff4a8	0xbffff4b8
ebp - 4	0xbffff4a4	0x00000000
ebp - 8	0xbffff4a0	
ebp - 12	0xbffff49c	
ebp - 16	0xbffff498	
ebp - 20	0xbffff494	0x00000002
A: esp	0xbffff490	0x00000001
ret: A	0xbffff48c	0x080483e6
B: ebp	0xbffff488	0xbffff4a8
	0xbffff484	0x00000000
B: esp	0xbffff480	0x00000000



	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001
ret: main	0xbffff4ac	0x080483b1
A: ebp	0xbffff4a8	0xbffff4b8
ebp - 4	0xbffff4a4	0x00000000
ebp - 8	0xbffff4a0	
ebp - 12	0xbffff49c	
ebp - 16	0xbffff498	
ebp - 20	0xbffff494	0x00000002
A: esp	0xbffff490	0x00000001
ret: A	0xbffff48c	0x080483e6
B: ebp	0xbffff488	0xbffff4a8
	0xbffff484	0x00000000
B: esp	0xbffff480	0x00000000

x86 Stack Frame Instructions

Enter and Leave [edit]

enter arg

Creates a stack frame with the specified amount of space allocated on the stack.

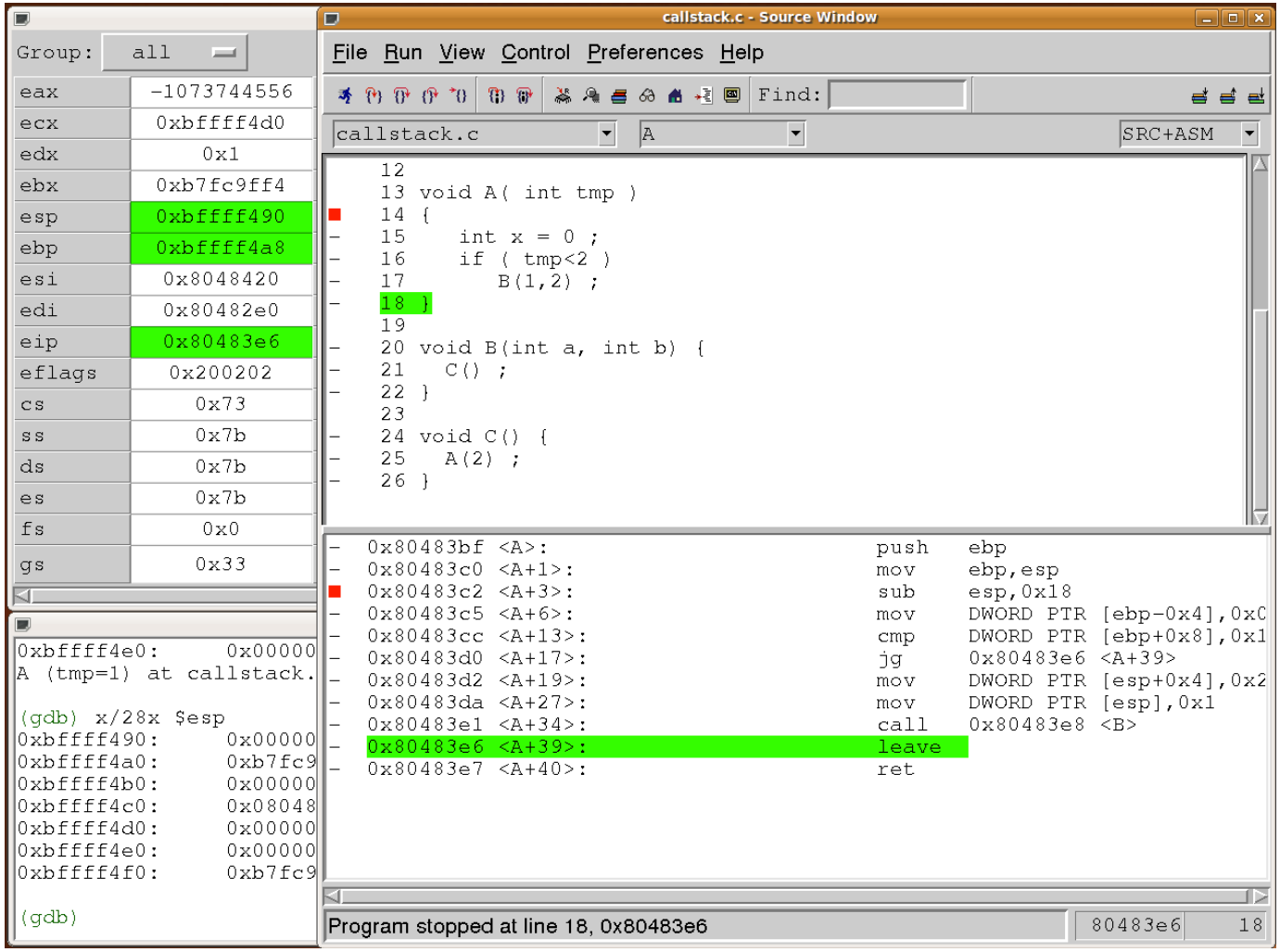
leave

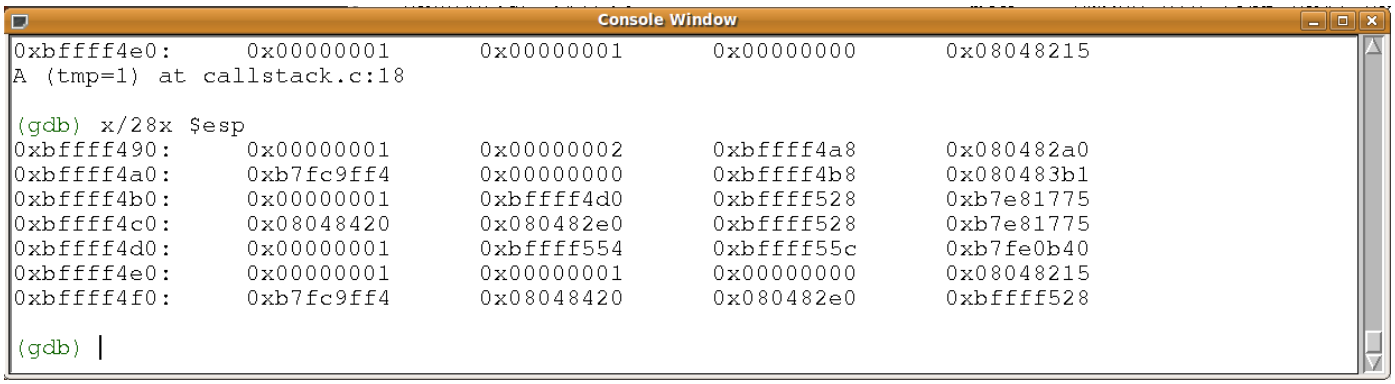
destroys the current stack frame, and restores the previous frame. Using Intel syntax this is equivalent to:

```
mov esp, ebp
pop ebp
```

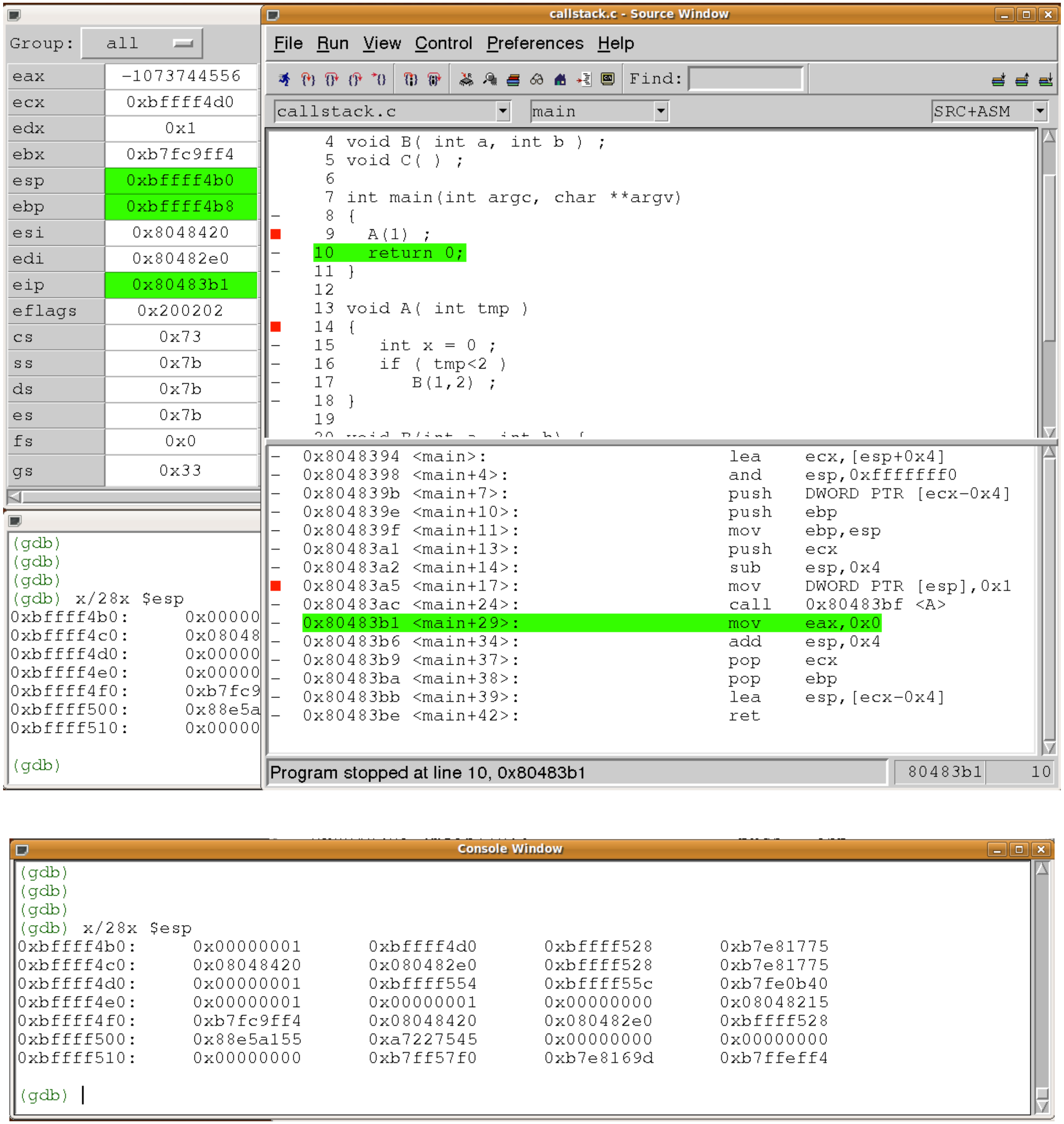
This will set `EBP` and `ESP` to their respective value before the function prologue began therefore reversing any modification to the stack that took place during the prologue.

http://en.wikibooks.org/wiki/X86_Assembly/Control_Flow#Function_Calls





	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001
ret: main	0xbffff4ac	0x080483b1
A: ebp	0xbffff4a8	0xbffff4b8
ebp - 4	0xbffff4a4	0x00000000
ebp - 8	0xbffff4a0	
ebp - 12	0xbffff49c	
ebp - 16	0xbffff498	
ebp - 20	0xbffff494	0x00000002
A: esp	0xbffff490	0x00000001



	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001

A() Calling B(1,2)

callstack.c

```c
#include <stdio.h>

void A( int tmp ) ;
void B( int a, int b ) ;
void C( ) ;

int main(int argc, char **argv)
{
  A(1) ;
  return 0;
}

void A( int tmp )
{
   int x = 0 ;
   if ( tmp<2 )
      B(1,2) ;
}

void B(int a, int b) {
  C() ;
}

void C() {
  A(2) ;
}
```

Line: 1 Column: 1 C Tab Size: 4

Prev EBP		
	2nd argument	EBP + 12
Prev ESP	1st argument	EBP + 8
	Return Address	EBP + 4
EBP	Previous EBP	EBP + 0
	Local Var #1	EBP - 4
ESP	Local Var #2	EBP + 8

	ADDR	MEMORY
main: ebp	0xbffff4b8	Main's Stack Frame
	0xbffff4b4	
main: esp	0xbffff4b0	
ret: main	0xbffff4ac	Return Address to Main
A: ebp	0xbffff4a8	Main's (Old) EBP
	0xbffff4a4	
	0xbffff4a0	
	0xbffff49c	
	0xbffff498	
	0xbffff494	2nd Argument to B()
A: esp	0xbffff490	1st Argument to B()
ret: A	0xbffff48c	Return Address to A
B: ebp	0xbffff488	A's (Old) EBP
	0xbffff484	0x00000000
B: esp	0xbffff480	0x00000000

	ADDR	MEMORY
main: ebp	0xbffff4b8	0xbffff528
	0xbffff4b4	0xbffff4d0
main: esp	0xbffff4b0	0x00000001
ret: main	0xbffff4ac	0x080483b1
A: ebp	0xbffff4a8	0xbffff4b8
ebp - 4	0xbffff4a4	0x00000000
ebp - 8	0xbffff4a0	
ebp - 12	0xbffff49c	
ebp - 16	0xbffff498	
ebp - 20	0xbffff494	0x00000002
A: esp	0xbffff490	0x00000001
ret: A	0xbffff48c	0x080483e6
B: ebp	0xbffff488	0xbffff4a8
	0xbffff484	0x00000000
B: esp	0xbffff480	0x00000000

STRINGS

Array Concepts

Size

Number of bytes allocated to the array (same as `sizeof(array)`).

Count

Number of elements in the array (same as the Visual Studio 2010 `_countof(array)`).

Length

Number of characters before null terminator.

Target size (Tsize)

Same as `sizeof(array)`.

Bound

The number of elements in the array.

Lo

The address of the first element of the array.

Hi

The address of the last element of the array.

TooFar

The address of the one-too-far element of the array, the element just past the Hi element.

String Concepts

Null-terminated

At or before Hi, the null terminator is present.

Length

Number of characters prior to the null terminator.

Strings are also “Array of Chars”

Improperly Bounded String Copies

gets_1.c

```c
#include <stdio.h>
#include <stdlib.h>


void get_y_or_n(void) {
    char response[8];
    puts("Continue? [y] n: ");
    gets(response);
    if (response[0] == 'n')
      exit(0);
    return;
}

int main() {
  get_y_or_n() ;
  return 0 ;
}

```

Line: 1 Column: 1 C Tab Size: 4

gets_2.c

```c
#include <stdio.h>
#include <stdlib.h>

char *gets(char *dest) {
    int c = getchar();
    char *p = dest;
    while (c != EOF && c != '\n') {
      *p++ = c;
      c = getchar();
    }
    *p = '\0';
    return dest;
}

int main() {
  char buffer[8] ;
  gets(buffer) ;
  return 0 ;
}

```

Line: 1 Column: 1 C Tab Size: 4

gets() doesn't limit number of chars to read from input

Un-Bounded Copies

Null-Termination Errors

null_term.c

```c

#include <string.h>

int main(void) {
  char a[16];
  char b[16];
  char c[16];
  strncpy(a, "0123456789abcdef", sizeof(a));
  strncpy(b, "0123456789abcdef", sizeof(b));
  strcpy(c, a);
 }

```

Line: 1 Column: 1 C Tab Size: 4

POINTERS

datatype_sizes.c

```c
#include <stdio.h>

int main() {
    printf("The 'int' data type is\t\t %d bytes\n", sizeof(int));
    printf("The 'unsigned int' data type is\t %d bytes\n", sizeof(unsigned int));
    printf("The 'short int' data type is\t %d bytes\n", sizeof(short int));
    printf("The 'long int' data type is\t %d bytes\n", sizeof(long int));
    printf("The 'long long int' data type is %d bytes\n", sizeof(long long int));
    printf("The 'float' data type is\t %d bytes\n", sizeof(float));
    printf("The 'char' data type is\t\t %d bytes\n", sizeof(char));

    printf( "sizeof(int*) = %lu\n", sizeof(int*)) ;
    printf( "sizeof(int)  = %lu\n", sizeof(int)) ;
    printf( "sizeof(int*)==sizeof(int) = %d\n", sizeof(int*)==sizeof(int)) ;
}
```

Line: 14 Column: 5 C Tab Size: 4 main

```
seed@seed-desktop: ~/CMPE27
File  Edit  View  Terminal  Help
seed@seed-desktop:~/CMPE279/modules/module
The 'int' data type is           4 bytes
The 'unsigned int' data type is  4 bytes
The 'short int' data type is     2 bytes
The 'long int' data type is      4 bytes
The 'long long int' data type is 8 bytes
The 'float' data type is         4 bytes
The 'char' data type is          1 bytes
sizeof(int*) = 4
sizeof(int)  = 4
sizeof(int*)==sizeof(int) = 1
seed@seed-desktop:~/CMPE279/modules/module
```

```
dragon:hacking pnguyen$ ./a.out
The 'int' data type is           4 bytes
The 'unsigned int' data type is  4 bytes
The 'short int' data type is     2 bytes
The 'long int' data type is      8 bytes
The 'long long int' data type is 8 bytes
The 'float' data type is         4 bytes
The 'char' data type is          1 bytes
sizeof(int*) = 8
sizeof(int)  = 4
sizeof(int*)==sizeof(int) = 0
dragon:hacking pnguyen$
```

http://en.wikibooks.org/wiki/C_Programming/C_Reference/limits.h

pointer.c

```c
#include <stdio.h>
#include <string.h>

int main() {
   char str_a[20];  // a 20 element character array
   char *pointer;   // a pointer, meant for a character array
   char *pointer2;  // and yet another one

   strcpy(str_a, "Hello World\n");
   pointer = str_a; // set the first pointer to the start of the array
   printf(pointer);

   pointer2 = pointer + 2; // set the second one 2 bytes further in
   printf(pointer2);       // print it
   strcpy(pointer2, "y you guys!\n"); // copy into that spot
   printf(pointer);        // print again
}
```

Line: 1 Column: 1 C Tab Size: 4

seed@seed-desktop: ~/CMPE279/modules/module2/hacking

File Edit View Terminal Help

```
seed@seed-desktop:~/CMPE279/modules/module2/hacking$ ./pointer.out
Hello World
llo World
Hey you guys!
seed@seed-desktop:~/CMPE279/modules/module2/hacking$
```

addressof2.c

```c
#include <stdio.h>

int main() {
   int int_var = 5;
   int *int_ptr;

   int_ptr = &int_var; // put the address of int_var into int_ptr

   printf("int_ptr = 0x%08x\n", int_ptr);
   printf("&int_ptr = 0x%08x\n", &int_ptr);
   printf("*int_ptr = 0x%08x\n\n", *int_ptr);

   printf("int_var is located at 0x%08x and contains %d\n", &int_var, int_var);
   printf("int_ptr is located at 0x%08x, contains 0x%08x, and points to %d\n\n",
      &int_ptr, int_ptr, *int_ptr);
}
```

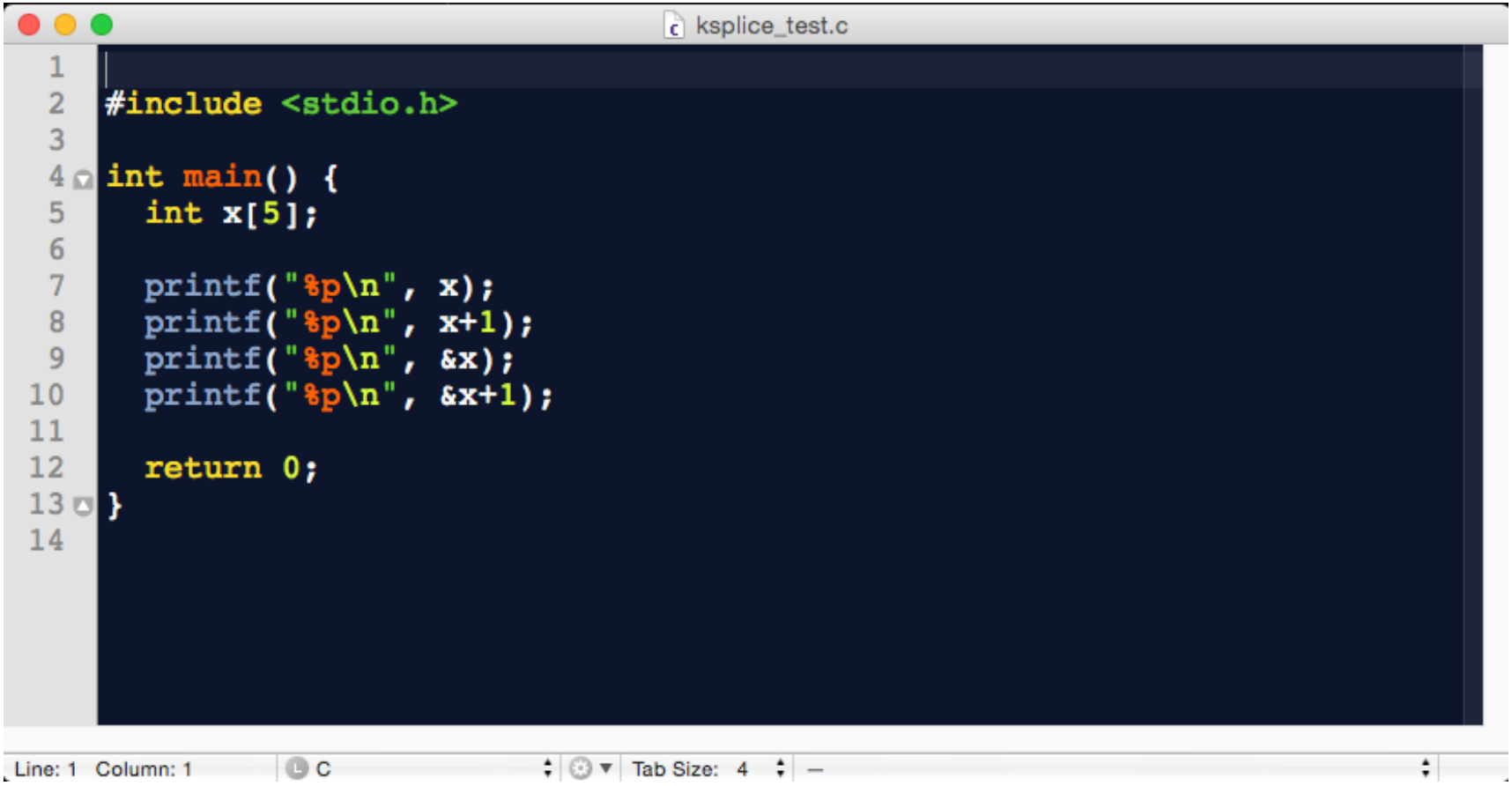
Line: 1 Column: 1 C Tab Size: 4

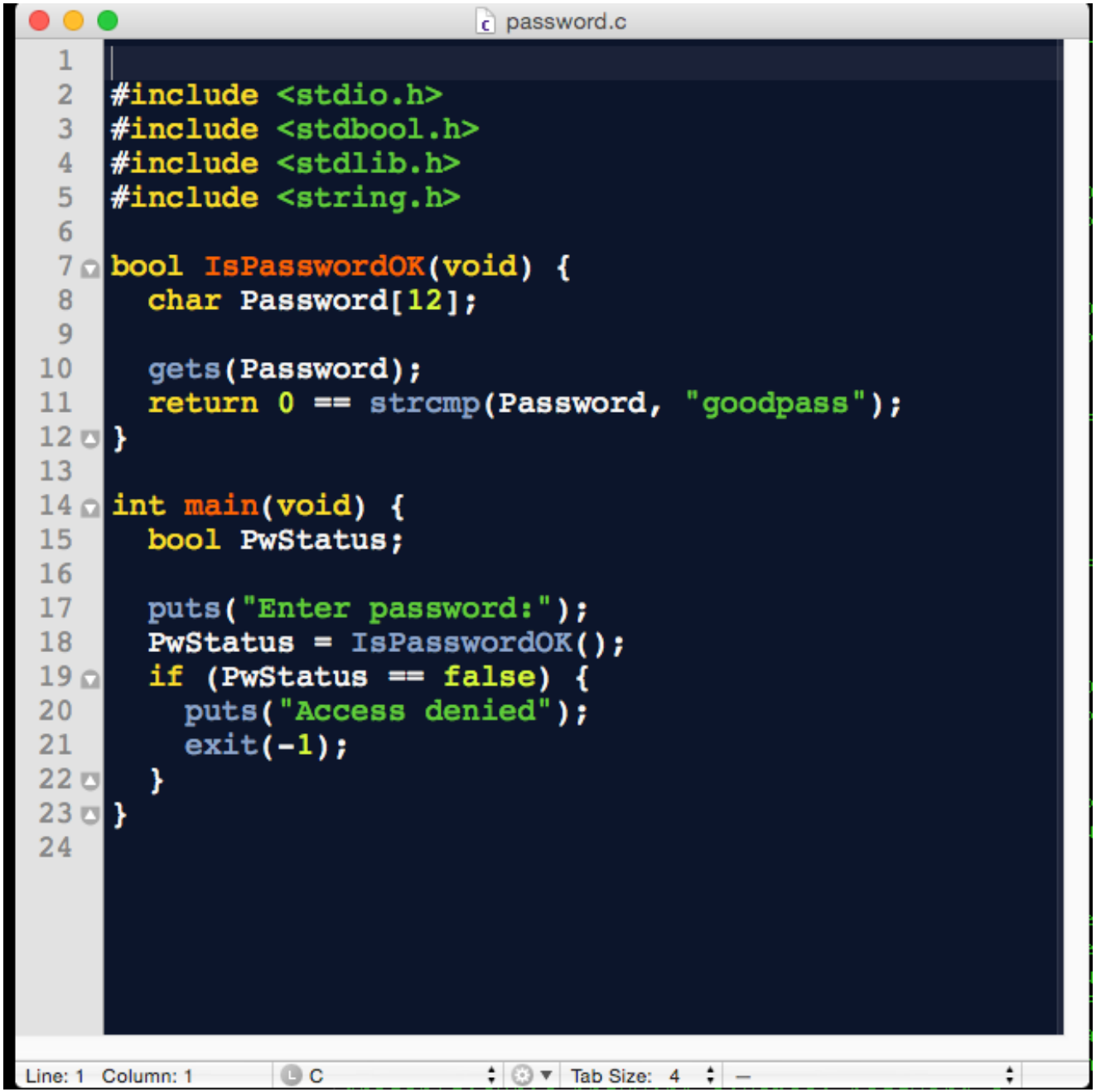
seed@seed-desktop: ~/CMPE279/modules/module2/hacking

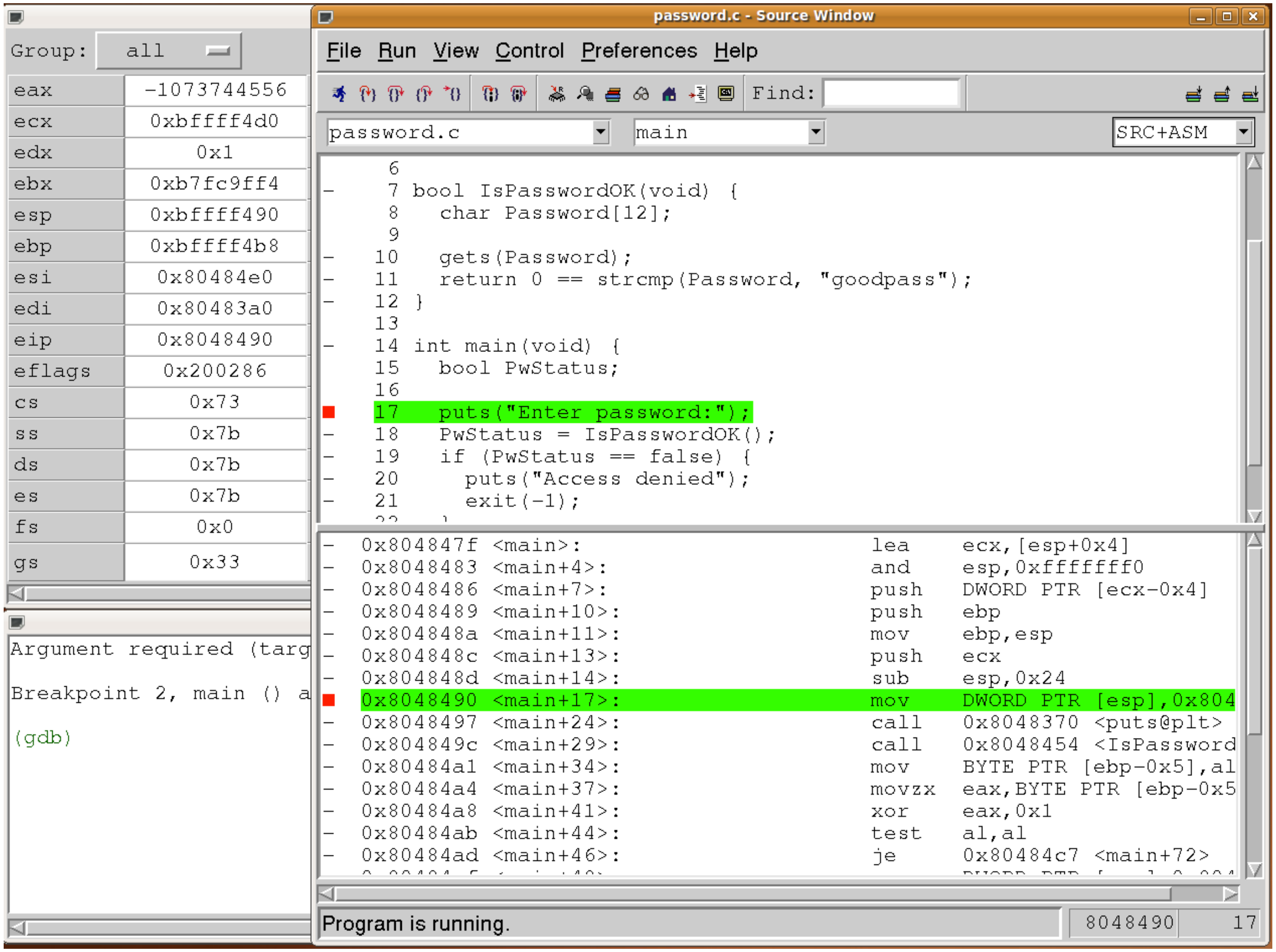
File Edit View Terminal Help

```
seed@seed-desktop:~/CMPE279/modules/module2/hacking$ ./addressof2.out
int_ptr = 0xbffff4f0
&int_ptr = 0xbffff4ec
*int_ptr = 0x00000005

int_var is located at 0xbffff4f0 and contains 5
int_ptr is located at 0xbffff4ec, contains 0xbffff4f0, and points to 5

seed@seed-desktop:~/CMPE279/modules/module2/hacking$
```

Explain the Output of the Following…




password.c
1
2 #include <stdio.h>
3 #include <stdbool.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 bool IsPasswordOK(void) {
8 char Password[12];
9
10 gets(Password);
11 return 0 == strcmp(Password, "goodpass");
12 }
13
14 int main(void) {
15 bool PwStatus;
16
17 puts("Enter password:");
18 PwStatus = IsPasswordOK();
19 if (PwStatus == false) {
20 puts("Access denied");
21 exit(-1);
22 }
23 }
24
Line: 1 Column: 1
C
Tab Size: 4



	ADDR	MEMORY
main: ebp		
main: esp		