

Call-de-stack

Version 1.1
Shakthi Kannan
shakthimaan.com
shakthimaan at gmail dot com
February 2008
GNU Free Documentation License

Dedication

In memory of Stack Overflow.

Environment

gcc

x86

Compilation

```
as --gstabs main.s -o main.o
```

```
ld main.o -o main
```

Execution

```
./main
```

OR

```
gdb main
```

Legend

804806f movl %ebp, %esp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
-> 8048071: popl %ebp
8048072: ret
```

Stack pointer

Instruction pointer

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0xbfc8e674
esi	0x0
edi	0x0
eip	0x8048071

Completed instruction

Value changed by completed instruction

Comment

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

Examples

- 1. Function call, without any arguments**
- 2. Function call, with an argument**
- 3. Recursive function**

Function call, without any arguments

```
.section .data

.section .text

.globl _start
_start:
    nop
    nop
    call foo
    movl $1, %eax
    int $0x80

.type foo, @function
foo:
    pushl %ebp
    movl %esp, %ebp

    nop

    movl %ebp, %esp
    popl %ebp
    ret
```

Output of "objdump -d main"

```
main:      file format elf32-i386
```

Disassembly of section .text:

```
08048054 <_start>:
```

```
8048054: 90                nop
8048055: 90                nop
8048056: e8 07 00 00 00   call 8048062 <foo>
804805b: b8 01 00 00 00   mov  $0x1,%eax
8048060: cd 80            int  $0x80
```

```
08048062 <foo>:
```

```
8048062: 55                push %ebp
8048063: 89 e5            mov  %esp,%ebp
8048065: 90                nop
8048066: 89 ec            mov  %ebp,%esp
8048068: 5d                pop  %ebp
8048069: c3                ret
```

8048054 nop

Assembly

```
.section .text
.globl _start
_start:
8048054:      nop
-> 8048055:      nop
8048056:      call foo
804805b:      movl $1, %eax
8048060:      int $0x80

.type foo, @function
foo:
8048062:      pushl %ebp
8048063:      movl %esp, %ebp
8048065:      nop
8048066:      movl %ebp, %esp
8048068:      popl %ebp
8048069:      ret
```

Registers

```
eax      0x0
ecx      0x0
edx      0x0
ebx      0x0
esp      0xbffa79a0
ebp      0x0
esi      0x0
edi      0x0
eip      0x8048055
```



Next IP

Stack

```
0xbffa79ac: 0xbffa7b3e
0xbffa79a8: 0x00000000
0xbffa79a4: 0xbffa7b1c
-> 0xbffa79a0: 0x00000001
0xbffa799c: 0x00000000
0xbffa7998: 0x00000000
0xbffa7994: 0x00000000
0xbffa7990: 0x00000000
```

8048055 nop

Assembly

```
.section .text
.globl _start
_start:
8048054:      nop
8048055:      nop
-> 8048056:      call foo
804805b:      movl $1, %eax
8048060:      int $0x80

.type foo, @function
foo:
8048062:      pushl %ebp
8048063:      movl %esp, %ebp
8048065:      nop
8048066:      movl %ebp, %esp
8048068:      popl %ebp
8048069:      ret
```

Registers

```
eax      0x0
ecx      0x0
edx      0x0
ebx      0x0
esp      0xbffa79a0
ebp      0x0
esi      0x0
edi      0x0
eip      0x8048056
```



Next IP

Stack

```
0xbffa79ac:  0xbffa7b3e
0xbffa79a8:  0x00000000
0xbffa79a4:  0xbffa7b1c
-> 0xbffa79a0:  0x00000001
0xbffa799c:  0x00000000
0xbffa7998:  0x00000000
0xbffa7994:  0x00000000
0xbffa7990:  0x00000000
```


8048056 call foo

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
-> 8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa799c
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048062

Next IP

call
pushes
return
address
to stack

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
0xbffa79a0:	0x00000001
-> 0xbffa799c:	0x0804805b
0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

Stack grows
downward

8048062 pushl %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
-> 8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

->

**Save base pointer
to stack**

Registers

```
eax    0x0
ecx    0x0
edx    0x0
ebx    0x0
esp    0xbffa7998
ebp    0x0
esi    0x0
edi    0x0
eip    0x8048063
```

Next IP

Stack

```
0xbffa79ac: 0xbffa7b3e
0xbffa79a8: 0x00000000
0xbffa79a4: 0xbffa7b1c
0xbffa79a0: 0x00000001
0xbffa799c: 0x0804805b
-> 0xbffa7998: 0x00000000
0xbffa7994: 0x00000000
0xbffa7990: 0x00000000
```

8048063 movl %esp, %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
-> 8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa7998
ebp	0xbffa7998
esi	0x0
edi	0x0
eip	0x8048065

Use base
pointer
as
reference

Next IP

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
0xbffa79a0:	0x00000001
0xbffa799c:	0x0804805b
-> 0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

8048065 nop

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
-> 8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

Registers

```
eax    0x0
ecx    0x0
edx    0x0
ebx    0x0
esp    0xbffa7998
ebp    0xbffa7998
esi    0x0
edi    0x0
eip    0x8048066
```



Next IP

Stack

```
0xbffa79ac: 0xbffa7b3e
0xbffa79a8: 0x00000000
0xbffa79a4: 0xbffa7b1c
0xbffa79a0: 0x00000001
0xbffa799c: 0x0804805b
-> 0xbffa7998: 0x00000000
0xbffa7994: 0x00000000
0xbffa7990: 0x00000000
```

8048066 movl %ebp, %esp

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
-> 8048068:    popl %ebp
8048069:    ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa7998
ebp	0xbffa7998
esi	0x0
edi	0x0
eip	0x8048068



Next IP

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
0xbffa79a0:	0x00000001
0xbffa799c:	0x0804805b
-> 0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

8048068 popl %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
-> 8048069:    ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa799c
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048069

Restore
old base
pointer

Next IP

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
0xbffa79a0:	0x00000001
-> 0xbffa799c:	0x0804805b
0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

8048069 ret

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
-> 804805b:    movl $1, %eax
8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa79a0
ebp	0x0
esi	0x0
edi	0x0
eip	0x804805b



ret
restores
saved
return
address
to ip

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
-> 0xbffa79a0:	0x00000001
0xbffa799c:	0x0804805b
0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

804805b movl \$1, %eax

Assembly

```
.section .text
.globl _start
_start:
8048054:    nop
8048055:    nop
8048056:    call foo
804805b:    movl $1, %eax
-> 8048060:    int $0x80

.type foo, @function
foo:
8048062:    pushl %ebp
8048063:    movl %esp, %ebp
8048065:    nop
8048066:    movl %ebp, %esp
8048068:    popl %ebp
8048069:    ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbffa79a0
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048060

Next IP

Stack

0xbffa79ac:	0xbffa7b3e
0xbffa79a8:	0x00000000
0xbffa79a4:	0xbffa7b1c
-> 0xbffa79a0:	0x00000001
0xbffa799c:	0x0804805b
0xbffa7998:	0x00000000
0xbffa7994:	0x00000000
0xbffa7990:	0x00000000

**Program
exited
normally.**

Function call, with an argument

```
.section .data
.section .text
.globl _start
_start:
    nop
    nop
    push $2
    call foo
    add $4, %esp
    movl $1, %eax
    int $0x80

.type foo, @function
foo:
    pushl %ebp
    movl %esp, %ebp

    movl 8(%ebp), %eax
    xorl %eax, %eax

    movl %ebp, %esp
    popl %ebp
    ret
```

Output of "objdump -d main"

main: file format elf32-i386

Disassembly of section .text:

```
08048054 <_start>:
8048054: 90                nop
8048055: 90                nop
8048056: 6a 02            push  $0x2
8048058: e8 0a 00 00 00  call  8048067 <foo>
804805d: 83 c4 04        add   $0x4,%esp
8048060: b8 01 00 00 00  mov   $0x1,%eax
8048065: cd 80            int  $0x80

08048067 <foo>:
8048067: 55                push  %ebp
8048068: 89 e5            mov   %esp,%ebp
804806a: 8b 45 08        mov   0x8(%ebp),%eax
804806d: 31 c0            xor  %eax,%eax
804806f: 89 ec            mov   %ebp,%esp
8048071: 5d                pop  %ebp
8048072: c3                ret
```

8048054 nop

Assembly

```
.section .text
.globl _start
_start:
-> 8048054: nop
-> 8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e680
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048055



Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
-> 0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000000
0xbfc8e678:	0x00000000
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048055 nop

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
-> 8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e680
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048056



Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
-> 0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000000
0xbfc8e678:	0x00000000
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048056 pushl \$0x2

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
-> 8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Next IP

Push
argument
to stack

Stack grows
downward

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e67c
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048058

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
-> 0xbfc8e67c:	0x00000002
0xbfc8e678:	0x00000000
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048058 call foo

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
-> 8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Next IP

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e678
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048067

**call
pushes
return
address
to stack**

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
-> 0xbfc8e678:	0x0804805d
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048067 pushl %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
-> 8048067: pushl %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048068

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

Save base pointer
to stack

8048068 movl %esp, %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
-> 804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0xbfc8e674
esi	0x0
edi	0x0
eip	0x804806a

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

Use base
pointer as
reference

804806a movl 8(%ebp), %eax

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
-> 804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0xbfc8e674
esi	0x0
edi	0x0
eip	0x804806d

Use the
argument

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

804806d xorl %eax, %eax

an operation
on the
argument

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
-> 804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0xbfc8e674
esi	0x0
edi	0x0
eip	0x804806f

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

804806f movl %ebp, %esp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
-> 8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e674
ebp	0xbfc8e674
esi	0x0
edi	0x0
eip	0x8048071



Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
-> 0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048071 popl %ebp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
-> 8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e678
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048072

Restore
old base
pointer

Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
-> 0xbfc8e678:	0x0804805d
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048072 ret

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
-> 804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e67c
ebp	0x0
esi	0x0
edi	0x0
eip	0x804805d

**ret
restores
saved
return
address
to ip**

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
0xbfc8e680:	0x00000001
-> 0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

804805d addl \$0x4, %esp

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
-> 8048060: movl $0x1, %eax
8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Next IP

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e680
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048060

restore
stack
pointer for
pushed
argument

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
-> 0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

8048060 movl \$0x1, %eax

Assembly

```
.section .text
.globl _start
_start:
8048054: nop
8048055: nop
8048056: pushl $0x2
8048058: call foo
804805d: addl $0x4, %esp
8048060: movl $0x1, %eax
-> 8048065: int $0x80

.type foo, @function
foo:
8048067: pushl %ebp
8048068: movl %esp, %ebp
804806a: movl 8(%ebp), %eax
804806d: xorl %eax, %eax
804806f: movl %ebp, %esp
8048071: popl %ebp
8048072: ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbfc8e680
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048065



Next IP

Stack

0xbfc8e68c:	0xbfc8fb38
0xbfc8e688:	0x00000000
0xbfc8e684:	0xbfc8fb12
-> 0xbfc8e680:	0x00000001
0xbfc8e67c:	0x00000002
0xbfc8e678:	0x0804805d
0xbfc8e674:	0x00000000
0xbfc8e670:	0x00000000

**Program
exited
normally.**

Recursive function - factorial

```
.section .data
.section .text
.globl _start
_start:
    nop
    nop
    push $3

    call factorial

    addl $4, %esp
    movl %eax, %ebx

    movl $1, %eax
    int $0x80
```

```
.type factorial, @function
factorial:
    pushl %ebp
    movl %esp, %ebp

    movl 8(%ebp), %eax
    cmpl $1, %eax
    je end_factorial

    decl %eax
    pushl %eax

    call factorial

    movl 8(%ebp), %ebx
    imull %ebx, %eax

end_factorial:
    movl %ebp, %esp
    popl %ebp
    ret
```

8048054 nop

Assembly

```
      _start:
8048054:  nop
-> 8048055:  nop
8048056:  pushl $0x3
8048058:  call factorial
804805d:  addl $0x4, %esp
9048060:  movl %eax, %ebx
8048062:  movl $0x1, %eax
8048067:  int $0x80

      factorial:
8048069:  pushl %ebp
804806a:  movl %esp, %ebp
804806c:  movl 8(%ebp), %eax
804806f:  cmpl $1, %eax
8048072:  je end_factorial
8048074:  decl %eax
8048075:  pushl %eax
8048076:  call factorial
804807b:  movl 8(%ebp), %ebx
804807e:  imull %ebx, %eax
      end_factorial:
8048081:  movl %ebp, %esp
8048083:  popl %ebp
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879260
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048055



Next IP

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
-> 0xbf879260:	0x00000001
0xbf87925c:	0x00000000
0xbf879258:	0x00000000
0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048055 nop

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
-> 8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879260
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048056

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
-> 0xbf879260:	0x00000001
0xbf87925c:	0x00000000
0xbf879258:	0x00000000
0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

Next IP

8048056 pushl \$3

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
-> 8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87925c
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048058

**Calculate
factorial of
3**

**Stack
grows
downward**

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
-> 0xbf87925c:	0x00000003
0xbf879258:	0x00000000
0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048058 call factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879258
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048069

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
-> 0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

Next IP

call
pushes
return
address
to stack

8048069 pushl %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0x0
esi	0x0
edi	0x0
eip	0x804806a

Save base
pointer to
stack

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806a movl %esp, %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
-> 804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x0
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x804806c

Use base pointer as reference

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806c movl 8(%ebp), %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
-> 804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x3
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x804806f

Use the argument

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806f `cmpl $1, %eax`

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
-> 8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x3
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048072

1 != 3

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048072 je end_factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
-> 8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x3
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048074

Flag not set

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048074 decl %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
-> 8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Next IP

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048075

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000000
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048075 pushl %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
-> 8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879250
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048076

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
-> 0xbf879250:	0x00000002
0xbf87924c:	0x00000000
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

Next IP

8048076 call factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87924c
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048069

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
-> 0xbf87924c:	0x0804807b
0xbf879248:	0x00000000
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

Next IP

call
pushes
return
address
to stack

8048069 pushl %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x804806a

Save base pointer to stack

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806a movl %esp, %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
-> 804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x804806c

Use base pointer as reference

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806c movl 8(%ebp), %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
-> 804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x804806f

Use the argument

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

804806f `cmpl $1, %eax`

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
-> 8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048072

1 != 2

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048072 je end_factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
-> 8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048074

Flag not set

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048074 decl %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
-> 8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048075

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000000
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

Next IP

8048075 pushl %eax

Assembly

```
      _start:
8048054:  nop
8048055:  nop
8048056:  pushl $3
8048058:  call factorial
804805d:  addl $0x4, %esp
9048060:  movl %eax, %ebx
8048062:  movl $0x1, %eax
8048067:  int $0x80

      factorial:
8048069:  pushl %ebp
804806a:  movl %esp, %ebp
804806c:  movl 8(%ebp), %eax
804806f:  cmpl $1, %eax
8048072:  je end_factorial
8048074:  decl %eax
8048075:  pushl %eax
-> 8048076:  call factorial
804807b:  movl 8(%ebp), %ebx
804807e:  imull %ebx, %eax
      end_factorial:
8048081:  movl %ebp, %esp
8048083:  popl %ebp
8048084:  ret
```

Next IP

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879244
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048076

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
-> 0xbf879244:	0x00000001
0xbf879240:	0x00000000
0xbf87923c:	0x00000000

8048076 call factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879240
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048069

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
-> 0xbf879240:	0x0804807b
0xbf87923c:	0x00000000

Next IP

call
pushes
return
address
to stack

8048069 pushl %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
-> 8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x804806a

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

Next IP

Save
base
pointer
to stack

804806a movl %esp, %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
-> 804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf87923c
esi	0x0
edi	0x0
eip	0x804806c

Use base pointer as reference

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

804806c movl 8(%ebp), %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
-> 804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf87923c
esi	0x0
edi	0x0
eip	0x804806f

Use the argument

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

804806f `cmpl $1, %eax`

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
-> 8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf87923c
esi	0x0
edi	0x0
eip	0x8048072

1 == 1

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

8048072 je end_factorial

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

```
      end_factorial:  
-> 8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Next IP

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf87923c
esi	0x0
edi	0x0
eip	0x8048081

Flag is set

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

8048081 movl %ebp, %esp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
-> 8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf87923c
ebp	0xbf87923c
esi	0x0
edi	0x0
eip	0x8048083

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
-> 0xbf87923c:	0xbf879248

Next IP

8048083 popl %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

```
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
-> 8048084:  ret
```

Next IP

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879240
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048084

Restore
old base
pointer

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
-> 0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048084 ret

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
-> 804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x0
esp	0xbf879244
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x804807b

**ret
restores
saved
return
address
to ip**

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
-> 0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

804807b movl 8(%ebp), %ebx

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
-> 804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x2
esp	0xbf879244
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x804807e

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
-> 0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

804807e imull %ebx, %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
-> 8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Next IP

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x2
esp	0xbf879244
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048081

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
-> 0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048081 movl %ebp, %esp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
-> 8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x2
esp	0xbf879248
ebp	0xbf879248
esi	0x0
edi	0x0
eip	0x8048083

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
-> 0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048083 pop %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

```
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
-> 8048084:  ret
```



Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x2
esp	0xbf87924c
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048084



Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
-> 0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048084 ret

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
-> 804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x2
esp	0xbf879250
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x804807b

**ret
restores
saved
return
address
to ip**

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
-> 0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

804807b movl 8(%ebp), %ebx

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
-> 804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x2
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf879250
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x804807e

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
-> 0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

804807e imull %ebx, %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
-> 8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Next IP

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf879250
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048081

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
-> 0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048081 movl %ebp, %esp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
-> 8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf879254
ebp	0xbf879254
esi	0x0
edi	0x0
eip	0x8048083

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
-> 0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

8048083 popl %ebp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
-> 8048084:  ret
```

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf879258
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048084

Restore
old base
pointer

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
0xbf87925c:	0x00000003
-> 0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

8048084 ret

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
-> 804805d:  addl $0x4, %esp  
9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

Next IP

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf87925c
ebp	0x0
esi	0x0
edi	0x0
eip	0x804805d

**ret
restores
saved
return
address
to ip**

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
0xbf879260:	0x00000001
-> 0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

804805d addl \$0x4, %esp

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
-> 9048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

factorial:

```
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax
```

end_factorial:

```
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x3
esp	0xbf879260
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048060

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
-> 0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

8048060 movl %eax, %ebx

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
8048060:  movl %eax, %ebx  
-> 8048062:  movl $0x1, %eax  
8048067:  int $0x80
```

```
      factorial:  
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x6
ecx	0x0
edx	0x0
ebx	0x6
esp	0xbf879260
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048062

Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
-> 0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

8048062 movl \$0x1, %eax

Assembly

```
      _start:  
8048054:  nop  
8048055:  nop  
8048056:  pushl $3  
8048058:  call factorial  
804805d:  addl $0x4, %esp  
8048060:  movl %eax, %ebx  
8048062:  movl $0x1, %eax  
-> 8048067:  int $0x80
```

```
      factorial:  
8048069:  pushl %ebp  
804806a:  movl %esp, %ebp  
804806c:  movl 8(%ebp), %eax  
804806f:  cmpl $1, %eax  
8048072:  je end_factorial  
8048074:  decl %eax  
8048075:  pushl %eax  
8048076:  call factorial  
804807b:  movl 8(%ebp), %ebx  
804807e:  imull %ebx, %eax  
      end_factorial:  
8048081:  movl %ebp, %esp  
8048083:  popl %ebp  
8048084:  ret
```

Registers

eax	0x1
ecx	0x0
edx	0x0
ebx	0x6
esp	0xbf879260
ebp	0x0
esi	0x0
edi	0x0
eip	0x8048067

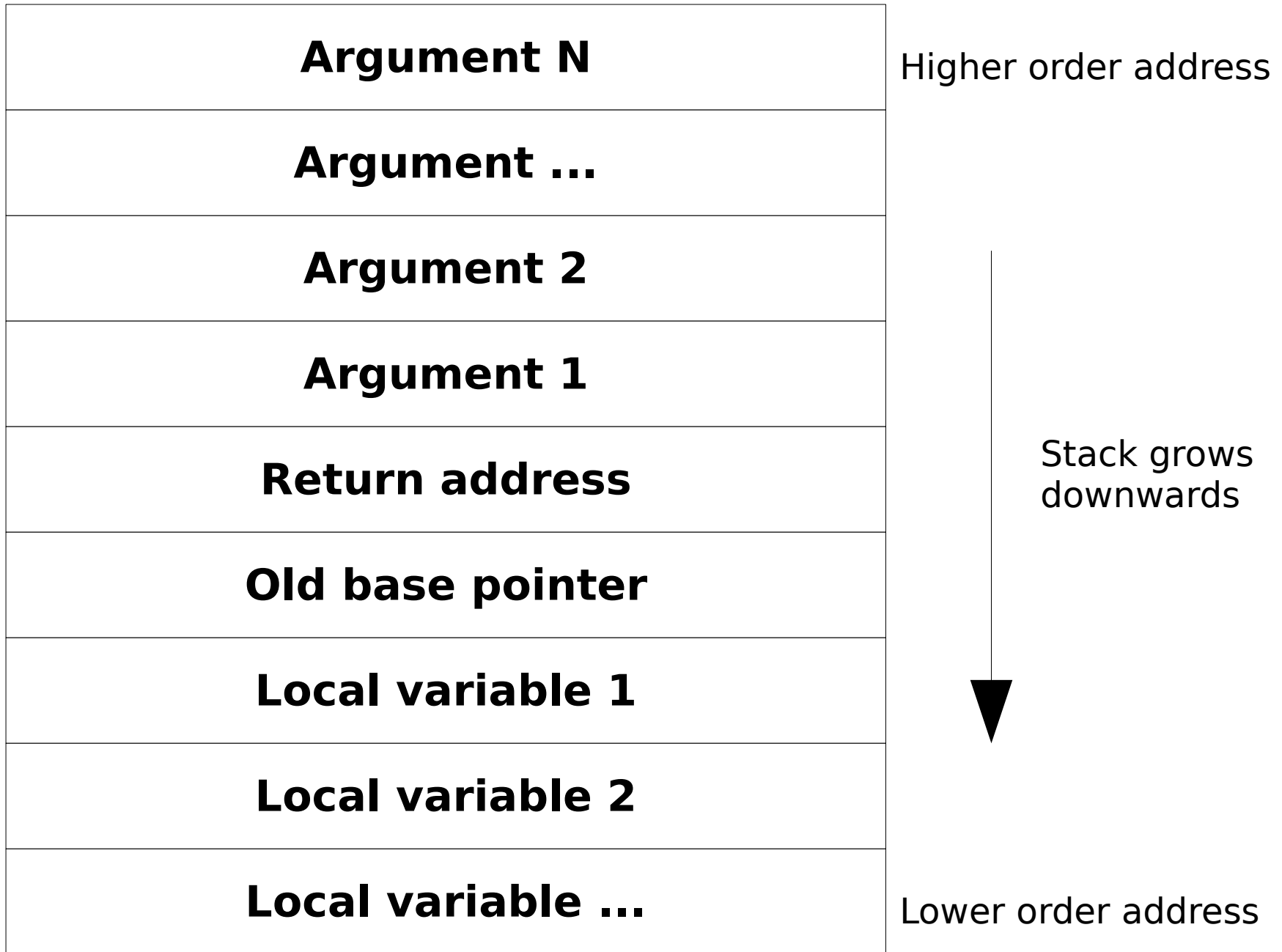
Stack

0xbf87926c:	0xbf879b36
0xbf879268:	0x00000000
0xbf879264:	0xbf879b0f
-> 0xbf879260:	0x00000001
0xbf87925c:	0x00000003
0xbf879258:	0x0804805d
0xbf879254:	0x00000000
0xbf879250:	0x00000002
0xbf87924c:	0x0804807b
0xbf879248:	0xbf879254
0xbf879244:	0x00000001
0xbf879240:	0x0804807b
0xbf87923c:	0xbf879248

Next IP

**Program
exited
normally.**

Stack Summary



References

- Richard Blum. 2005. *Professional Assembly Language*. Wrox.
- Jonathan Bartlett. 2003. *Programming from the Ground Up*.