# x86 Calling Conventions on 32bit Linux

**Overview** Calling conventions describe the interface of the called code:

- the order in which atomic (scalar) parameters, or individual parts of a complex parameter, are allocated
- how parameters are passed (pushed on the stack, placed in registers, or mix of both)
- which registers may be used by the callee without first being saved (i.e. pushed)
- how the task of setting up for and restoring the stack after a function call is divided between the caller and the callee

**Register usage** The following is a summary of x86 registers.

| type | name | usage |
|---|---|---|
| general | `%eax` | return value |
| | `%edx` | dividened register |
| | `%ecx` | counter register value |
| | `%ebx`, `%esi`, `%edi` | local register variable |
| | `%esp` | stack pointer |
| | `%ebp` | frame pointer (optional) |
| FP | `%sp(0)` | FP stack top, return value |
| | `%sp(0)` | FP next to stack top |
| | … | |
| | `%sp(7)` | FP stack bottom |

The following are 32-bit Linux register usage. Note that this is different from that of 32-bit Windows.

| | |
|---|---|
| **scratch registers** | `%eax`, `%ecx`, `%edx`, `%st(0)`–`%st(7)`, `%xmm(0)`–`%xmm(7)`, `%ymm(0)`–`%ymm(7)` |
| **callee-save registers** | `%ebx`, `%esi`, `%edi`, `%ebp` |
| **argument registers** | none |
| **registers for return** | `%eax`, `%st(0)`, `%xmm(0)`, `%ymm(0)` |

**cdecl calling convention** The **cdecl** calling convention is used by many C systems (incl. GCC) for the x86 architecture.

- Function **arguments** are passed on the stack in a right-to-left order.
- Function **return values** are returned in the `%eax` register (except for floating point values, which are returned in the x87 register `%st(0)`)
- The registers, `%eax`, `%ecx`, and `%edx` do not need to be preserved, while others do.
- **caller** is responsible for **stack cleanup**

```
int foo(int, int, int);
int a, b, c, x;
x = foo(a, b, c);
```

```
push c;      ; arg3
push b;      ; arg2
push a;      ; arg1
call foo;
add esp, 12  ; pop funargs (a, b, c) from stack
mov x, eax   ; fetch return value
```

| position | contents | frame |
|---|---|---|
| `4n+8(%ebp)` | argument *n* | previous |
| | … | |
| `8(%ebp)` | argument 0 | |
| `4(%ebp)` | return address | current |
| `0(%ebp)` | previous `%ebp` (optional) 0 | |
| `-4(%ebp)` | unspecified | |
| | … | |
| `0(%esp)` | | |

In case, to force cdecl calling convention, we can add `_cdecl` modifier:

```
void _cdecl foo(int, int);
```

**Functions returning scalars or no value** A function that returns an integral or pointer value places its result in register `%eax`. A floating-point return value appears on the top of the x87 register stack. *The caller is responsible for removing the value from the x87 stack, even if it does not use it.*

- **function prologue**:

```
prologue:
  pushl %ebp       ; save frame pointer
  movl  %esp, %ebp ; set new frame pointer
  subl  $80, %esp  ; allocate stack space
  pushl %edi       ; save local register
  pushl %esi       ; save local register
  pushl %ebx       ; save local register
```

- **call** instruction: *pushes the address of the next instruction (**the return address**) onto the stack.*
- **function epilogue**: restores the state for the caller

```
  movl  %edi, %eax ; set up return value
epilogue:
  popl  %ebx       ; restore local register
  popl  %esi       ; restore local register
  popl  %edi       ; restore local register
  leave            ; restore frame pointer
  ret              ; pop return address
```

- **ret** instruction: *pops the address off the stack and effectively continues execution at the next instruction after the **call** instruction*