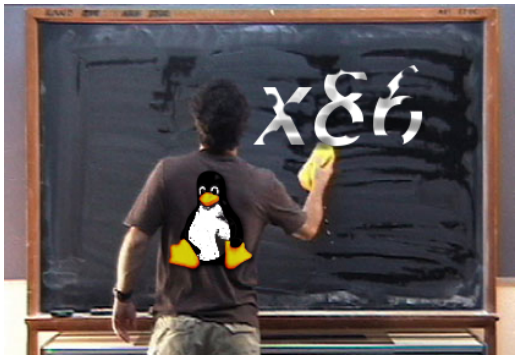
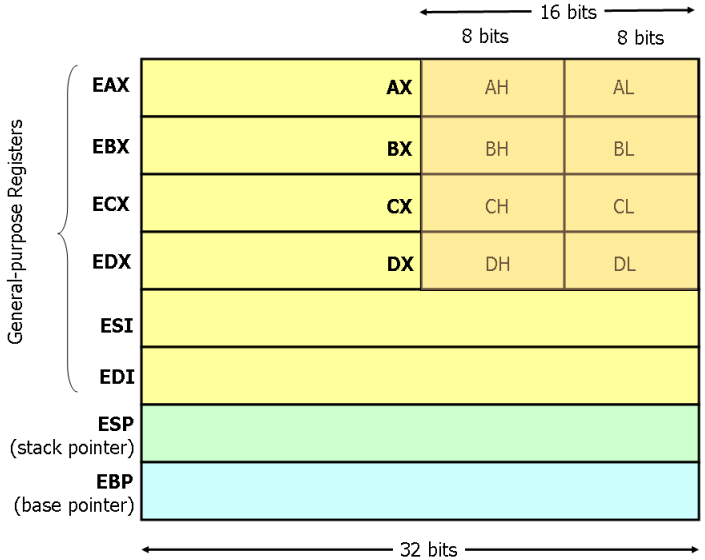
x86 Assembly Programming under Linux





Data Movement Instructions

mov — Move
push — Push stack
pop — Pop stack
lea — Load effective address

Control Flow Instructions

jmp — Jump
cmp — Compare
j*condition* — Conditional Jump
call, **ret** — Subroutine call and return

Arithmetic and Logic Instructions

add — Integer Addition
sub — Integer Subtraction
inc, **dec** — Increment, Decrement
mul — Integer Multiplication
div — Integer Division
and, **or**, **xo**r — Bitwise Logical And, Or and Xor
not — Bitwise Logical Not
neg — Negate
shl, **shr** — Shift Left, Shift Right

min.s

```
.section .data              # start of data section
a:      .long 42            # variable a
b:      .long 53            # variable b
m:      .long 0             # variable m

.section .text              # start of text section
.globl _start               # _start is a global symbol
                            # specifying the program start
_start:
        movl a, %eax
        movl b, %ebx
        cmpl %ebx, %eax     # compare a with b
        jle if              # if (a <= b)
        jmp else
if:     movl %eax, m        #     m = a
        jmp endif           # else
else:   movl %ebx, m        #     m = b
endif:  movl m, %ebx
        movl $1, %eax
        int $0x80           # exit(m)
```

Assemble:

```
as -o min.o min.s
```

Link:

```
ld -o min min.o
```

Execute:

```
./min
```

Print the result:

```
echo $?
```


min.c

```
int min(int a, int b) {
    int m;
    if (a < b)
        m = a;
    else
        m = b;
    return m;
}
```


Generate assembly code for the C code:

```
gcc –S -m32 min.c
```

This generates `min.s`

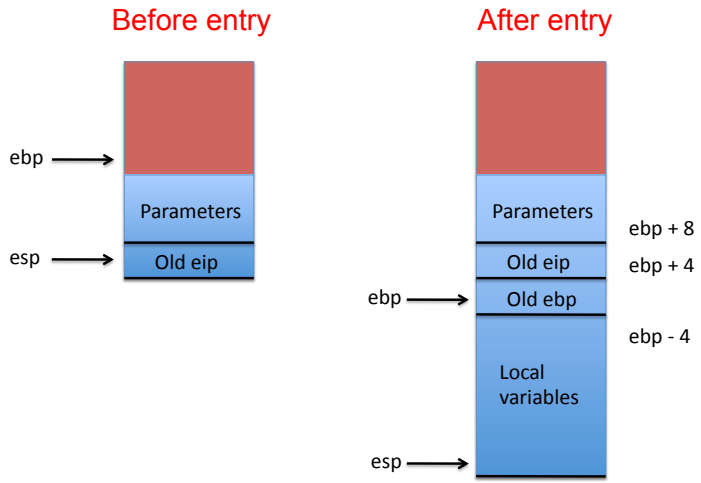
Show the assembly code:

```
cat min.s
```

or

```
gedit min.s
```


Stack frame



Call
min(2, 3)

```
pushl $3
pushl $2
call  min
addl $8, %esp
```



```
        .file   "min.c"
        .text
        .globl  min
        .type   min, @function
min:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        cmpl    12(%ebp), %eax
        jge     .L2
        movl    8(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L3
.L2:
        movl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
.L3:
        movl    -4(%ebp), %eax
        leave
        ret
```

Debugging information has been left out

equivalent to

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


Generate optimized assembly code:

```
gcc -O -S -m32 min.c
```

Show the assembly code:

```
cat min.s
```


Optimized code

```
        .file   "min.c"
        .text
        .globl min
        .type   min, @function
min:
        movl    4(%ebp), %eax
        movl    8(%ebp), %edx
        cmpl    %edx, %eax
        cmovle  %edx, %eax
        ret
```

More about x86 assembly programming

The book

> “Programming from the Ground Up”
> by Jonathan Bartlett

may be downloaded via the webpage of the course