Computer Labs:
C Function Call Conventions

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

October 13, 2011

Mixed C and Assembly Programming

- The name of a function or a global variable in assembly is the C name prefixed with an **underscore**
  - But this is not enough, even for a function as simple as `set_timer2_freq`
- **Functions in assembly that are supposed to be called from C, must adhere to the C function call convention used by the compiler.**

C Function Call Convention

- The parameters of a C function are passed in the processor stack
  - All parameters are pushed onto the stack before calling the function
  - The stack is adjusted appropriately after returning from the function
- The function parameters are pushed onto the stack in **reverse** order:
  - The first parameter will be on top of the stack just before the call
- The return value is put in AL, AX or EAX, depending on its size

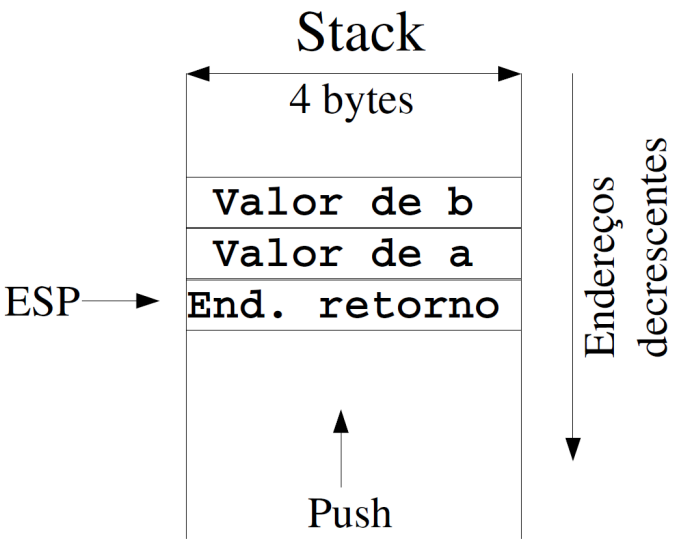
gcc and Processor Registers

In an assemby function that is called by a C function

- You can change `EAX`, `ECX` and `EDX`
- You must preserve `EBX`, `ESI`, `EDI` and `EBP`

All other registers must not be touched.

C Function Invocation




```
int foo(int a, int b);

static int a=5, b=7, c;

void bar() {
    c = foo(a,b);
}
```

gcc -S -O0 bar.c

```
   ...
   subl $24, %esp
   movl b, %edx
   movl a, %eax
   movl %edx, 4(%esp)
   movl %eax, (%esp)
   call foo
   movl %eax, c
   ...
```

```
foo:
   ...
   ret
```

Function Parameter Access

```
int foo(int a, int b) {
    return a + b;
}
```

gcc -S -O0 foo.c

```
foo:
  pushl %ebp
  movl %esp, %ebp
  movl 12(%ebp), %eax
  movl 8(%ebp), %edx
  leal (%edx,%eax), %eax
  popl %ebp
  ret
```

Question What are

```
pushl %ebp
movl  %esp, %ebp
```

for?

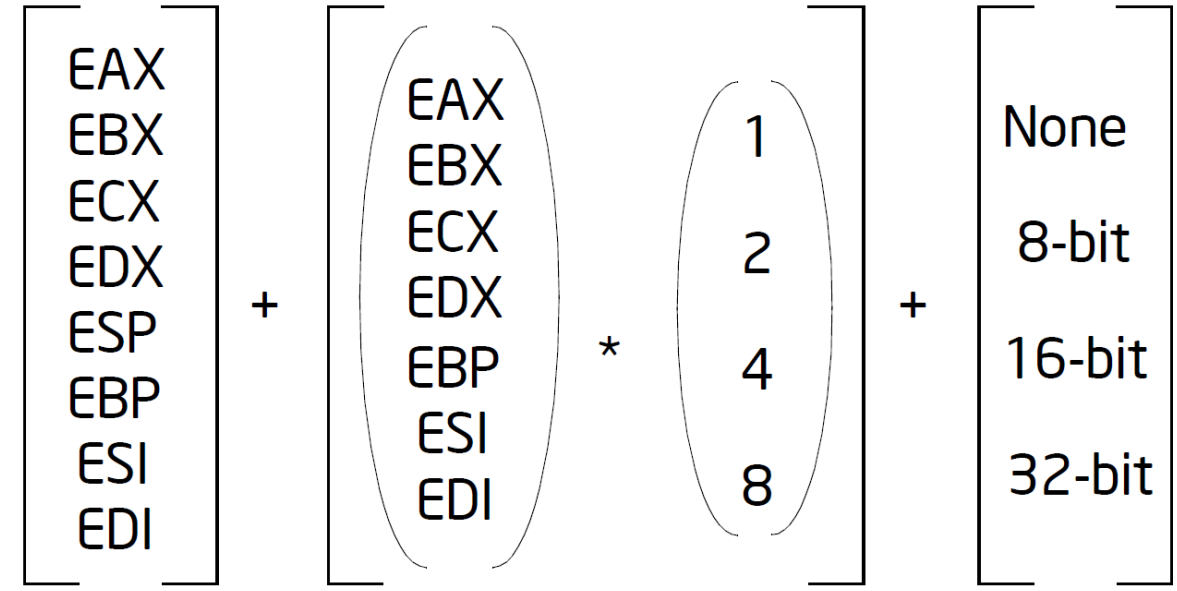
Answer The `EBP` is used to access the parameters

- The compiler can generate code with immediate offset values, without having to worry with changes to the `ESP`
  - Note that functions may call other functions.

Question Where is the `ADD` instruction?

Answer `gcc` uses the address computation machinery to perform some integer arithmetic via the `leal` instruction

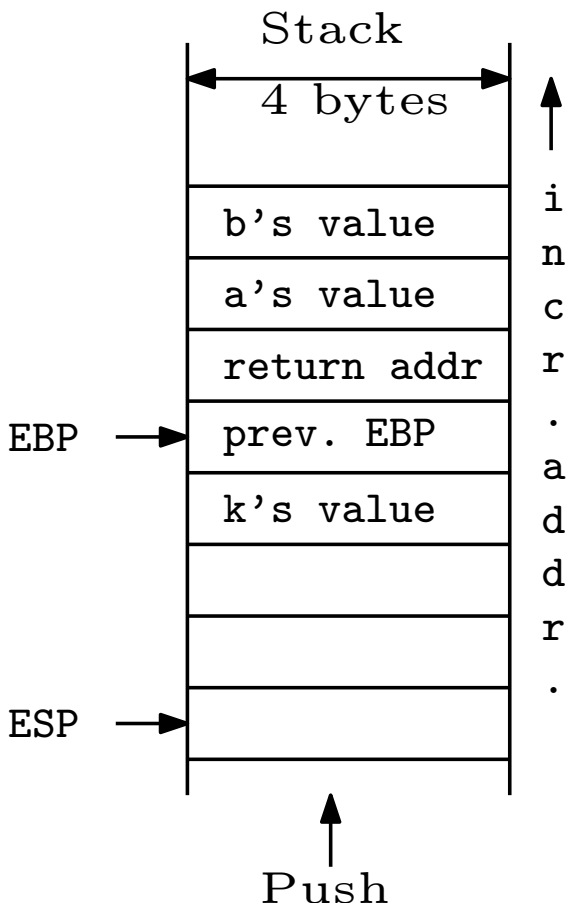
(LEA and Integer Arithmetic)



Offset = Base + (Index * Scale) + Displacement

- `gcc` first loads the `EDX` and `EAX` registers with the parameter values;
- Then issues a `leal` instruction, which assumes that the contents of `EDX` is an address, and the `EAX` is an index, computes the effective address and loads it to the `EAX`
- The net result is that the code computes the sum of the two parameters

Functions with Local Variables




```
int foo(int a, int b) {
    int k;
    k = a + b;
    return k;
}
```

gcc -S -O0 foo.c

```
foo:
  pushl %ebp                ; save previous ebp
  movl  %esp, %ebp          ; initialize ebp
  subl  $16, %esp
; reserves 16 bytes for local variables
  movl  12(%ebp), %eax      ; eax <- b
  movl  8(%ebp), %edx       ; edx <- a
  leal  (%edx,%eax), %eax   ; eax <- a + b
  movl  %eax, -4(%ebp)      ; k <- a+b
  movl  -4(%ebp), %eax      ; eax <- k
  leave
; leave is equ. to movl %ebp,%esp + popl %ebp
  ret
```

Further Reading

- Calling Conventions chapter, of the X86 Disassembly Wikibook
- IA-32 SW Developer's Manual, Vol. 1, Ch.6: Procedure Calls, Interrupts and Exceptions
- *João Cardoso e Miguel Pimenta Monteiro*, Programas com funções escritas em C e Assembly