Enough x86 assembler to be dangerous

Charles Bailey

Bloomberg LP

Developer Experience Engineering London

Enough x86 assembler to be *very* dangerous

Charles Bailey

Bloomberg LP

Developer Experience Engineering London

Why dangerous?

- Assembly language isn't object code
- Object code isn't directly executed by “silicon”
- There is not substitute for measuring

What about writing assembly code?

- There are many pitfalls...
- ... in summary, don't.

So why learn an assembly language?

- Understand the next abstraction down
- Observe the impact of your code choices
- It's fun

Just enough “Architecture” to get by

Architecture summary

- Registers
- Virtual address space
- Stack

Registers

- Fast to access
- Local to each CPU core

General purpose registers

```
eax ebx ecx edx
esi edi esp ebp
```

Special purpose registers

`eip / rip`
`eflags / rflags`

Virtual address space

- Data mapped in from the executable
- Dynamically allocated data
- The stack

Stack

- Specific to each thread
- Combined call stack and data stack
- Grows downward from higher memory addresses

Show me assembly code!

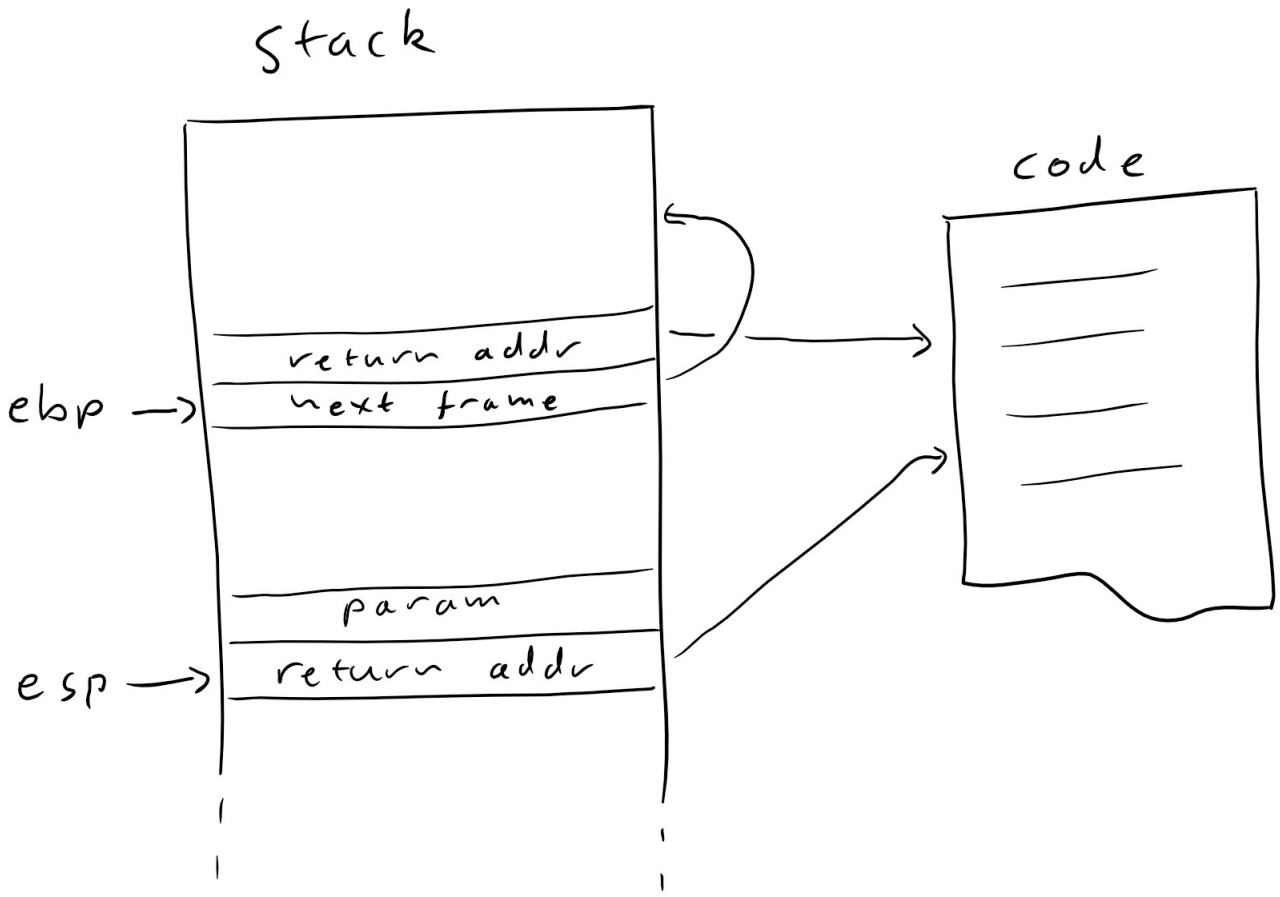
- `objdump -d -Mintel <object-file>`
- `-S -masm=intel` to gcc or clang
- `disassemble` in gdb
  (after `set disassembly-flavor intel`)
- Open the Debug > Windows > Disassembly window in Visual Studio

A simple example

```
long add(long a, long b) { return a + b; }

g++ -m32 -O0 -masm=intel -fno-exceptions\
 -fno-asynchronous-unwind-tables -fno-pic

_Z3addll:
    push    ebp
    mov     ebp, esp
    mov     edx, DWORD PTR [ebp+8]
    mov     eax, DWORD PTR [ebp+12]
    add     eax, edx
    pop     ebp
    ret
```


stack
code
return addr
ebp
next frame
param
esp
return addr

Demonstration class

```
struct Foo {
    int data;
    Foo();
    Foo(const Foo&);
    Foo(Foo&&);
    ~Foo();

    Foo& operator+=(const Foo&);
};
```

Candidates for operator+

```
Foo op_plus1(const Foo& a, const Foo& b) {
    Foo r(a);
    r += b;
    return r;
}

Foo op_plus2(const Foo& a, const Foo& b) {
    return Foo(a) += b;
}
```

operator+ (candidate one)

```
_Z8op_plus1RK3FooS1_:
    push    ebx
    sub     esp, 16
    mov     ebx, DWORD PTR [esp+24]
    push    DWORD PTR [esp+28]
    push    ebx
    call    _ZN3FooC1ERKS_          ; copy constructor
    pop     eax
    pop     edx
    push    DWORD PTR [esp+32]
    push    ebx
    call    _ZN3FoopLERKS_          ; operator+=
    add     esp, 24
    mov     eax, ebx
    pop     ebx
    ret     4
```

operator+ (candidate two)

```
; much the same code until we call operator+= ,
; [...]
    push    eax
    push    esi
    call    _ZN3FooC1ERKS_          ; copy constructor
    mov     DWORD PTR [esp], ebx
    call    _ZN3FooD1Ev             ; destructor
    lea     esp, [ebp-8]
    mov     eax, esi
; [...]
```

Questions?

References

Stackoverflows x86 tag wiki
https://stackoverflow.com/tags/x86/info

Intel developer's manual
http://www.intel.com/content/www/us/en/processors/a
software-developer-manuals.html

x86-* ABI documentation https://github.com/hjl-tools/x86-psABI