

What's in a binary?

Simon Kågström

Consultant

<https://github.com/SimonKagstrom/emilpro>

February 20, 2025



- Part I: Motivation
- Part II: What does the disassembly writer need?
- Part III: Why is this easier now than 10 years ago?

Part I: Motivation



Motivation

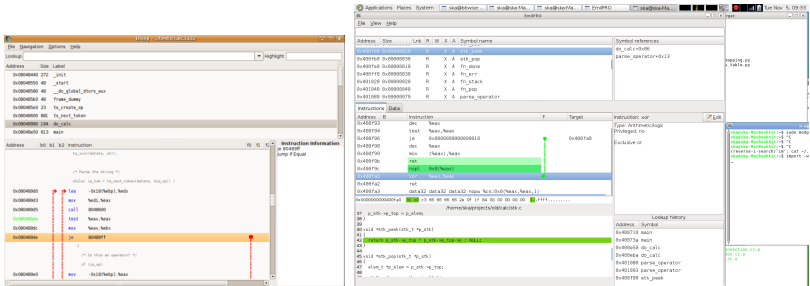
- C/C++ with inline assembly
- Systems and programming environments where a debugger wasn't available

Example

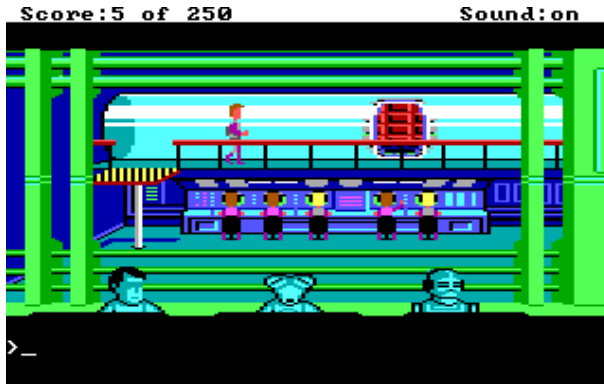
```
#define _syscall1(type,name,atype,a) type name(atype a) {
    unsigned long out;
    __asm__ volatile (
        ".set push\n.set noreorder\n"
        ".short 0xfefe\n"
        ".short %1\n"
        ".pushsection .cibylstrtab, \"aS\"\n"
        "1: .asciz \"" #name "\"\n"
        ".popsection\n"
        ".long 1b\n"
        ".set\tpop\n"
        "move %[out], $2\n"
        : [out]="=d" (out)
        : "r"(a)
        : "memory", "$2"
    );
    return (type) out;
}
```

Backstory

- Objdump output is cumbersome to navigate through
- I wanted a graphical application that allows easier navigation



Part II: What is needed for a disassembler?



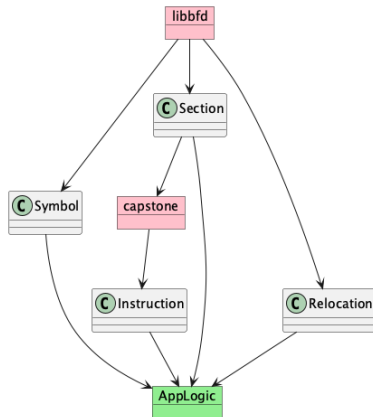
Binary formats

- **Linux/FreeBSD** etc: ELF
- **MacOS**: Mach-O
- **Windows**: PE

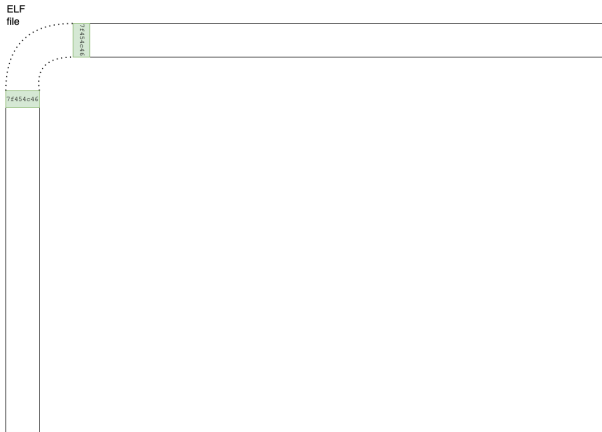
How to write a disassembler?

I did not write everything from scratch!

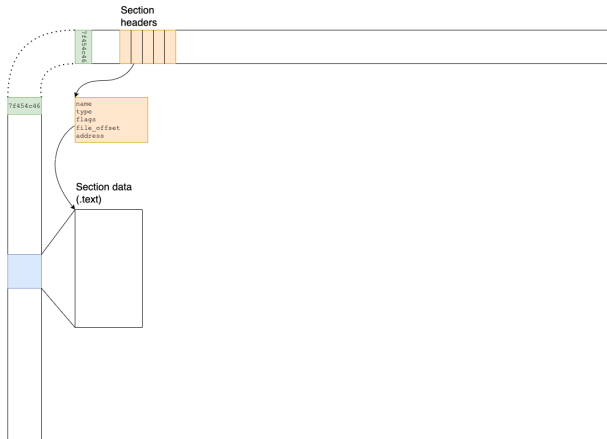
- **Qt**: GUI framework
- **libbfd**: Part of binutils, used for reading binary files
- **capstone**: Disassembler library



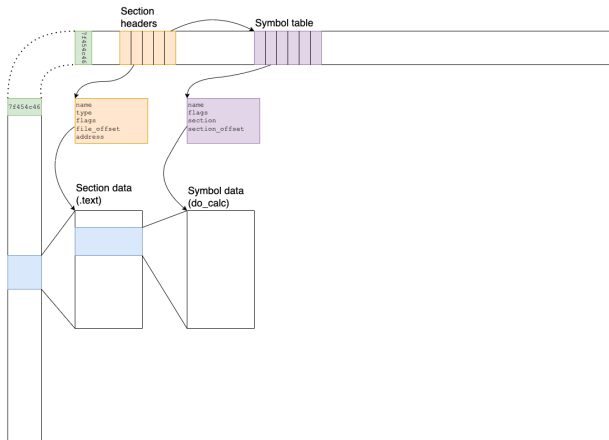
Loading a binary



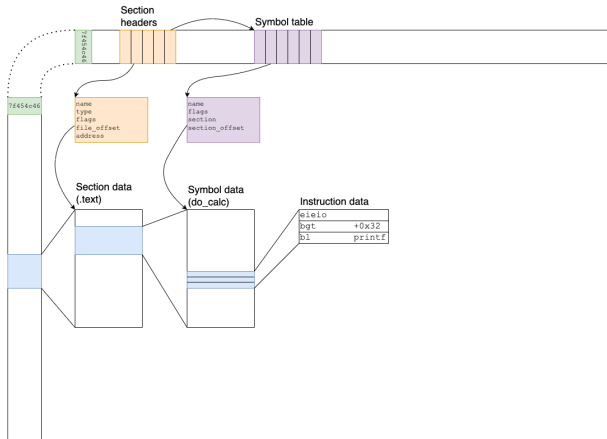
Loading a binary, sections



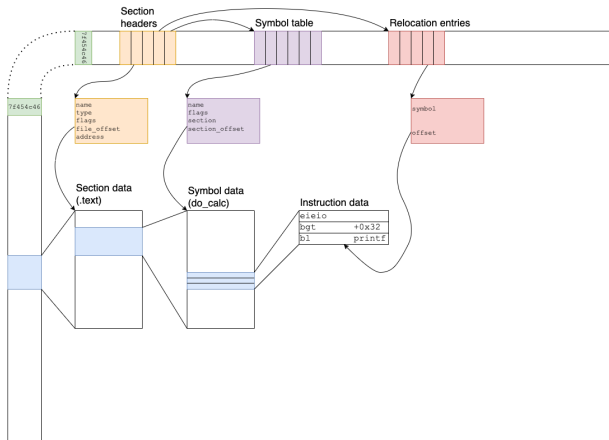
Loading a binary, symbols



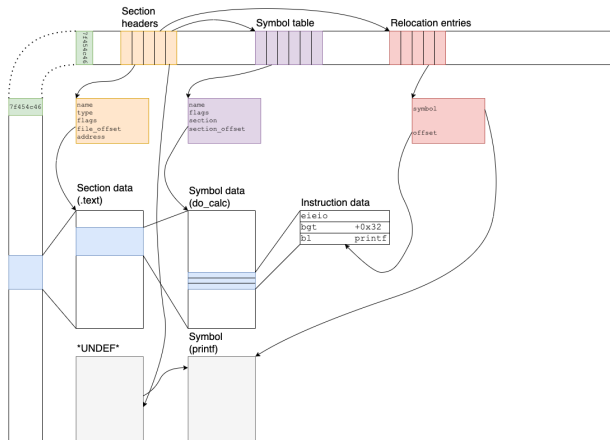
Loading a binary, instructions



Loading a binary, relocations



Loading a binary, relocations



Part III: Why is this easier now than 10 years ago?



Bad design decisions in the original project

- feature creep: core functionality sketchy, work on irrelevant features

The screenshot displays the EmuPRO debugger interface. The top status bar shows the user 'ska@ska-fra...' and the target 'EmuPRO'. The main window is divided into several panes:

- Assembly View:** Shows a list of instructions with their addresses, sizes, and disassembled forms. The instruction at address 0x400f9f is highlighted in blue.

Address	Size	Lnk	R	W	X	A	Symbol name
0x400f9f	0x00000030	R	X	A	stk_peak		
0x400f9b	0x00000030	R	X	A	stk_pop		
0x400f98	0x00000018	R	X	A	trn_dome		
0x400f95	0x00000030	R	X	A	trn_err		
0x400f92	0x00000020	R	X	A	trn_stack		
0x400f8f	0x00000040	R	X	A	trn_pop		
0x400f8b	0x00000078	R	X	A	parse_operator		
- Symbol references:** Lists symbols used by the code, including 'do_calc@x86', 'parse_operator@x13', and 'hopping.py'.

Symbol name
do_calc@x86
parse_operator@x13
hopping.py
hopping.py
- Instructions:** A detailed view of the selected instruction at address 0x400f9f, showing its fields: Address, Instruction, F, Target, and Instruction.x86.

Address	Instruction	F	Target	Instruction.x86
0x400f9f	dec %eax			Type: Arithmetic/logic
0x400f9f	test %eax,%eax		0x400fa0	Privileged: no
0x400f96	je 0x0000000000000010			Exclusive or
0x400f98	dec %eax			
0x400f99	mov [%eax],%eax			
0x400f9b	ret			
0x400f9c	popl %eax			
0x400f9d	movl %eax,%eax			
0x400fa2	ret			
0x400fa3	data32 data32 data32 nowp %s:0x0(%eax,%eax,1)			
- Stack Trace:** Shows the current state of the stack, including the return address and the current instruction pointer.

Address	Symbol
0x400710	main
0x40073a	main
0x400e50	do_calc
0x400e50	do_calc
0x400800	parse_operator
0x400f93	parse_operator
0x400f90	stk_peak

The bottom pane shows the assembly code for the 'main' function, with the instruction 'return p_stk-tp; p_stk-tp_top = p_stk-tp_top + 1;' highlighted in green.

Bad design decisions in the original project

- feature creep: core functionality sketchy, work on irrelevant features
- libbfd for disassembly: the multiarch-dev issue

The screenshot displays the GDB interface with the following components:

- Symbol table:** Lists symbols with addresses, sizes, and types. For example, `0x489f93 0x00000000 R X A stk_peek`.
- Symbol references:** Shows references to `do_calc+0x86` and `parse_operator+0x13`.
- Instructions:** Displays assembly instructions with addresses. For example, `0x489f93: dec %eax`, `0x489f94: test %eax,%eax`, `0x489f95: je 0x0000000000000010`, `0x489f96: dec %eax`, `0x489f97: mov (%eax),%eax`, `0x489f98: ret`, `0x489f99: popl 0x0(%eax)`, `0x489fa0: xor %eax,%eax`, `0x489fa1: ret`, `0x489fa2: data32 data32 data32 nopw %cs:0x0(%eax,%eax,1)`, `0x489fa3: data32 data32 data32 nopw %cs:0x0(%eax,%eax,1)`.
- Data:** Shows a memory dump starting with `0x0000000000000000`.
- Lookup history:** Lists symbols looked up, including `0x489f10 main`, `0x489f3a do_calc`, `0x489f3a do_calc`, `0x489f3a parse_operator`, `0x489f3a parse_operator`, and `0x489f3a stk_peek`.

C++20 and onwards

- The previous implementation was done just around the C++11 introduction, but used C++03
- Now C++23, so much better!

C++03

```
for (InstructionList_t::const_iterator it = instructions.begin();  
     it != instructions.end();  
     ++it) {
```

C++23

```
for (const auto& insn : m_instructions)  
{
```

- The conan package manager
- The address sanitizer

Conan

```
[requires]
doctest/2.4.11
trompeloeil/48
fmt/11.0.2
capstone/5.0.1
etl/20.39.4
libiberty/9.1.0
```

```
[generators]
CMakeDeps
CMakeToolchain
```

```
[layout]
cmake_layout
```

- I use Github copilot
- Very helpful with Qt development

```

begin{frame}{Static executables}
% Sections, symbols, relocations
% Example from embedded system
% Load sections into memory
% Entry point
% Disassembler, compiler, linker, loader
% Relocation entries
% Why are they needed?
% - when a function is called, the linker doesn't know where it is
% - the relocation entry tells the linker to fix the call site
% - the linker will then fix the call site to point to the function
% - the function is in the shared library
% - the linker will also fix the function to point to the shared library
% - the shared library is loaded into memory
% - the function is called
% - the function is executed
% - the function returns
% - the function is called again
% - the function is executed again
% - the function returns again
% - the function is called a third time
% - the function is executed a third time
% - the function returns a third time
% - the function is called a fourth time
% - the function is executed a fourth time
% - the function returns a fourth time
% - the function is called a fifth time
% - the function is executed a fifth time
% - the function returns a fifth time
% - the function is called a sixth time
% - the function is executed a sixth time
% - the function returns a sixth time
% - the function is called a seventh time
% - the function is executed a seventh time
% - the function returns a seventh time
% - the function is called an eighth time
% - the function is executed an eighth time
% - the function returns an eighth time
% - the function is called a ninth time
% - the function is executed a ninth time
% - the function returns a ninth time
% - the function is called a tenth time
% - the function is executed a tenth time
% - the function returns a tenth time
% - the function is called an eleventh time
% - the function is executed an eleventh time

```

Questions and comments!



Images from

<http://www.falselogic.net/LetsPlay/SpaceQuest.html>

Ian Lance Taylor's linker series is the source of parts of this talk

Actors and objects

Actors

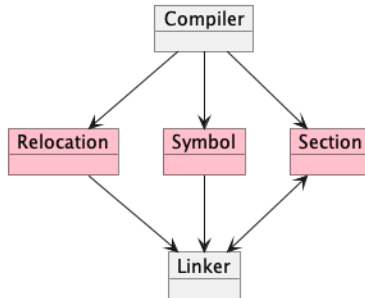
- **Compiler**
- **Linker**
- **Loader**
- **Disassembler**

Objects

- **Instructions:** The actual code
- **Sections:** Text, data, debug info etc
- **Symbols:** Functions/methods, variables, ...
- **Relocations:** Call sites for later resolving

Producing a binary

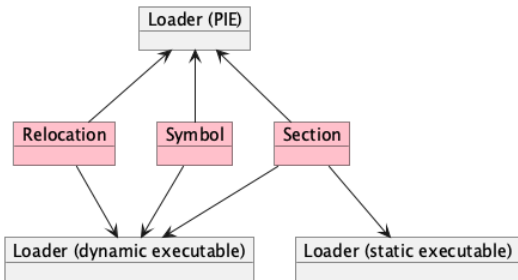
The compiler produces symbols, relocations plus data and text sections



Loading a binary

Different categories of binaries are handled differently:

- Execute from direct-mapped flash (embedded systems)
- Static executables
- Dynamic executables
- PIEs (Position-Independent Executables)



Static executables

Dynamic executables

- If a non-local function is called, an undefined symbol is added
- The compiler adds a relocation entry for the call site
- When linking, the linker will resolve these symbols
- Different types depending on instruction

