Contents

1 Assembly language and computer architecture 1

1 Basic computer architecture 2

1.1 The core architecture 2

1.1.1 Model of computation 2

1.1.2 Von Neumann architecture 3

1.2 Evolution 4

1.2.1 Drawbacks of von Neumann architecture 4

1.2.2 Intel 64 architecture 5

1.2.3 Architecture extensions 5

1.3 Registers 6

1.3.1 General purpose registers 7

1.3.2 Other registers 9

1.3.3 System registers 10

1.4 Protection rings 11

1.5 Hardware stack 12

1.6 Summary 14

2 Assembly language 15

2.1 Setting up the environment 15

2.2 Writing “Hello world” 16

2.2.1 Basic input and output 16

2.2.2 Program structure 17

2.2.3 Basic instructions 18

2.3 Example: output register contents 20

2.4 Function calls 23

2.5 Working with data 26

2.5.1 Endianness 26

2.5.2 Strings 27

2.5.3 Constant precomputation 28

2.5.4 Pointers and different addressing types 28

2.6 Example: calculating string length 30

2.7 Assignment: Input-output library 32


<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8 Summary</td>
<td>34</td>
</tr>
<tr>
<td><strong>3 Legacy</strong></td>
<td></td>
</tr>
<tr>
<td>3.1 Real mode</td>
<td>36</td>
</tr>
<tr>
<td>3.2 Protected mode</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Minimal segmentation in long mode</td>
<td>40</td>
</tr>
<tr>
<td>3.4 Accessing parts of registers</td>
<td></td>
</tr>
<tr>
<td>3.4.1 An unexpected behavior</td>
<td></td>
</tr>
<tr>
<td>3.4.2 CISC and RISC</td>
<td></td>
</tr>
<tr>
<td>3.4.3 Explanation</td>
<td></td>
</tr>
<tr>
<td>3.5 Summary</td>
<td></td>
</tr>
<tr>
<td><strong>4 Virtual memory</strong></td>
<td></td>
</tr>
<tr>
<td>4.1 Caching</td>
<td></td>
</tr>
<tr>
<td>4.2 Motivation</td>
<td></td>
</tr>
<tr>
<td>4.3 Address spaces</td>
<td></td>
</tr>
<tr>
<td>4.4 Features</td>
<td></td>
</tr>
<tr>
<td>4.5 Example: accessing forbidden address</td>
<td></td>
</tr>
<tr>
<td>4.6 Efficiency</td>
<td></td>
</tr>
<tr>
<td>4.7 Implementation</td>
<td></td>
</tr>
<tr>
<td>4.7.1 Virtual address structure</td>
<td></td>
</tr>
<tr>
<td>4.7.2 Address translation in depth</td>
<td></td>
</tr>
<tr>
<td>4.7.3 Page sizes</td>
<td></td>
</tr>
<tr>
<td>4.8 Memory mapping</td>
<td></td>
</tr>
<tr>
<td>4.9 Example: mapping file into memory</td>
<td></td>
</tr>
<tr>
<td>4.10 Summary</td>
<td></td>
</tr>
<tr>
<td><strong>5 Compilation pipeline</strong></td>
<td></td>
</tr>
<tr>
<td>5.1 Preprocessor</td>
<td></td>
</tr>
<tr>
<td>5.1.1 Simple substitutions</td>
<td></td>
</tr>
<tr>
<td>5.1.2 Substitutions with arguments</td>
<td></td>
</tr>
<tr>
<td>5.1.3 Simple conditional substitution</td>
<td></td>
</tr>
<tr>
<td>5.1.4 Conditioning on definition</td>
<td></td>
</tr>
<tr>
<td>5.1.5 Conditioning on text identity</td>
<td></td>
</tr>
<tr>
<td>5.1.6 Conditioning on argument type</td>
<td></td>
</tr>
<tr>
<td>5.1.7 Evaluation order: define, xdefine, assign</td>
<td></td>
</tr>
<tr>
<td>5.1.8 Repetition</td>
<td></td>
</tr>
<tr>
<td>5.1.9 Example: computing prime numbers</td>
<td></td>
</tr>
<tr>
<td>5.1.10 Labels inside macros</td>
<td></td>
</tr>
<tr>
<td>5.1.11 Conclusion</td>
<td></td>
</tr>
<tr>
<td>5.2 Translation</td>
<td></td>
</tr>
<tr>
<td>5.3 Linking</td>
<td></td>
</tr>
<tr>
<td>5.3.1 Executable and Linkable Format (ELF)</td>
<td></td>
</tr>
<tr>
<td>5.3.2 Relocatable object files</td>
<td></td>
</tr>
<tr>
<td>5.3.3 Executable object files</td>
<td></td>
</tr>
<tr>
<td>5.3.4 Dynamic libraries</td>
<td></td>
</tr>
<tr>
<td>5.3.5 Loader</td>
<td></td>
</tr>
<tr>
<td>5.4 Assignment: Dictionary</td>
<td></td>
</tr>
<tr>
<td>5.5 Summary</td>
<td></td>
</tr>
<tr>
<td><strong>6 Interrupts and system calls</strong></td>
<td></td>
</tr>
<tr>
<td>6.1 Input and output</td>
<td></td>
</tr>
<tr>
<td>6.1.1 TR register and Task State Segment</td>
<td></td>
</tr>
<tr>
<td>6.2 Interrupts</td>
<td></td>
</tr>
<tr>
<td>6.3 System calls</td>
<td></td>
</tr>
</tbody>
</table>

*Draft May 6, 2017 – not typeset nor copyedited yet*
7 Models of computation

7.1 Finite state machines
   7.1.1 Definition
   7.1.2 Example: bits parity
   7.1.3 Implementation in assembly language
   7.1.4 Practical value
   7.1.5 Regular Expressions

7.2 Forth-machine
   7.2.1 Architecture
   7.2.2 Tracing an exemplary Forth program
   7.2.3 Dictionary
   7.2.4 How words are implemented
   7.2.5 Compiler

7.3 Assignment: Forth compiler and interpreter
   7.3.1 Static dictionary, interpreter
   7.3.2 Compilation
   7.3.3 Forth with bootstrap

7.4 Summary

II The C programming language

8 Basics

8.1 Introduction

8.2 Program structure
   8.2.1 Data types

8.3 Control flow
   8.3.1 if
   8.3.2 while
   8.3.3 for
   8.3.4 goto
   8.3.5 switch
   8.3.6 Example: divisor
   8.3.7 Example: is it a Fibonacci number?

8.4 Statements and expressions
   8.4.1 Statement types
   8.4.2 Building expressions

8.5 Functions

8.6 Preprocessor

8.7 Summary

9 Type system

9.1 Basic type system of C
   9.1.1 Numeric types
   9.1.2 Type casting
   9.1.3 Boolean type
   9.1.4 Implicit conversions
   9.1.5 Pointers
   9.1.6 Arrays
   9.1.7 Arrays as function arguments

9.2 Summary
CONTENTS

9.1.8 Designated initializers in arrays .............................................. 140
9.1.9 Type aliases ................................................................. 140
9.1.10 The main function revisited .............................................. 142
9.1.11 Operator sizeof .......................................................... 142
9.1.12 Const types ............................................................... 143
9.1.13 Strings ..................................................................... 145
9.1.14 Functional types ......................................................... 145
9.1.15 Coding well ............................................................. 147
9.1.16 Assignment: scalar product ............................................ 151
9.1.17 Assignment: prime number checker .................................... 152

9.2 Tagged types ........................................................................... 152
  9.2.1 Structures ........................................................................ 152
  9.2.2 Unions ............................................................................ 154
  9.2.3 Anonymous structures and unions ...................................... 155
  9.2.4Enumerations .................................................................... 156

9.3 Data types in programming languages ..................................... 156
  9.3.1 Kinds of typing .................................................................. 157
  9.3.2 Polymorphism .................................................................. 159

9.4 Polymorphism in C .................................................................. 160
  9.4.1 Parametric polymorphism .................................................. 160
  9.4.2 Inclusion .......................................................................... 162
  9.4.3 Overloading ...................................................................... 162
  9.4.4 Coercions ........................................................................ 163

9.5 Summary .................................................................................. 163

10 Code structure ........................................................................... 165
  10.1 Declarations and definitions .................................................. 165
    10.1.1 Function declarations ..................................................... 166
    10.1.2 Structure declarations .................................................... 167
  10.2 Accessing code from other files ............................................ 168
    10.2.1 Functions from other files .............................................. 168
    10.2.2 Data in other files ........................................................ 169
    10.2.3 Header files .................................................................. 170
  10.3 Standard library .................................................................... 172

10.4 Preprocessor .......................................................................... 174
  10.4.1 Include Guard ............................................................... 175
  10.4.2 Why is preprocessor evil? ................................................. 177

10.5 Example: sum of a dynamic array .......................................... 178

10.6 Assignment: linked list .......................................................... 180

10.7 The static keyword ................................................................ 181

10.8 Linkage .................................................................................. 182

10.9 Conclusion .............................................................................. 182

11 Memory ..................................................................................... 184
  11.1 Pointers revisited ............................................................... 184
    11.1.1 Why do we need pointers? ............................................. 184
    11.1.2 Pointer arithmetic ......................................................... 185
    11.1.3 The void* type ............................................................. 186
    11.1.4 NULL ........................................................................... 186
    11.1.5 A word on strcmp ......................................................... 187
    11.1.6 Function pointers ......................................................... 188
  11.2 Memory model ...................................................................... 189

11.2.1 Memory allocation .......................................................... 190

11.3 Arrays and pointers .............................................................. 191
13 Good code practices

13.1 Making choices .................................................. 222
13.2 Code elements ....................................................... 223
  13.2.1 General naming ............................................. 223
  13.2.2 File structure ................................................. 223
  13.2.3 Types .......................................................... 224
  13.2.4 Variables ....................................................... 225
  13.2.5 On global variables .......................................... 226
  13.2.6 Functions ....................................................... 226
13.3 Files and documentation ........................................ 226
13.4 Encapsulation ...................................................... 228
13.5 Immutability ....................................................... 231
13.6 Assertions ........................................................ 231
13.7 Error handling ..................................................... 232
13.8 On memory allocation ........................................... 234
13.9 On flexibility ....................................................... 234
13.10 Assignment: image rotation .................................... 236
  13.10.1 BMP file format ............................................. 236
  13.10.2 Architecture ................................................. 237
13.11 Assignment: custom memory allocator ....................... 239
13.12 Summary .......................................................... 241

12 Syntax, semantics and pragmatics 204

12.1 What is a programming language? .............................. 204
12.2 Syntax and formal grammars .................................... 205
  12.2.1 Example: natural numbers ................................ 206
  12.2.2 Example: simple arithmetics ............................... 207
  12.2.3 Recursive descent ............................................ 207
  12.2.4 Example: arithmetics with priorities .................... 210
  12.2.5 Example: simple imperative language ................... 211
  12.2.6 Chomsky hierarchy .......................................... 211
  12.2.7 Abstract syntax tree ........................................ 212
  12.2.8 Lexical analysis .............................................. 213
  12.2.9 Summary on parsing ......................................... 213
12.3 Semantics .......................................................... 213
  12.3.1 Undefined behavior ......................................... 213
  12.3.2 Unspecified behavior ....................................... 215
  12.3.3 Implementation-defined behavior ......................... 215
  12.3.4 Sequence points ............................................. 215
12.4 Pragmatics ........................................................ 216
  12.4.1 Alignment ....................................................... 216
  12.4.2 Data structure padding ..................................... 217
12.5 Alignment in C11 .................................................. 220
12.6 Summary .......................................................... 221

CONTENTS CONTENTS
## III Between C and assembly

### 14 Translation details

- 14.1 Function calling sequence
  - 14.1.1 XMM registers
  - 14.1.2 Calling convention
  - 14.1.3 Example: simple function and its stack
  - 14.1.4 Red zone
  - 14.1.5 Variable number of arguments
  - 14.1.6 vprintf and friends
- 14.2 volatile
  - 14.2.1 Lazy memory allocation
  - 14.2.2 Generated code
- 14.3 Non-local jumps – setjmp
  - 14.3.1 volatile and setjmp
- 14.4 inline
- 14.5 restrict
- 14.6 Strict aliasing
- 14.7 Security issues
  - 14.7.1 Stack buffer overrun
  - 14.7.2 return-to-libc
  - 14.7.3 Format output vulnerabilities
- 14.8 Protection mechanisms
  - 14.8.1 Security cookie
  - 14.8.2 Address space layout randomization
  - 14.8.3 DEP
- 14.9 Summary

### 15 Shared objects and code models

- 15.1 Dynamic loading
- 15.2 Relocations and PIC
- 15.3 Example: dynamic library in C
- 15.4 GOT and PLT
  - 15.4.1 Accessing external variables
  - 15.4.2 Calling external functions
  - 15.4.3 PLT example
- 15.5 Preloading
- 15.6 Symbol addressing summary
- 15.7 Examples
  - 15.7.1 Calling a function
  - 15.7.2 On various dynamic linkers
  - 15.7.3 Accessing an external variable
  - 15.7.4 Complete assembly example
  - 15.7.5 Mixing C and assembly
- 15.8 Which objects are linked?
- 15.9 Optimizations
- 15.10 Code models
  - 15.10.1 Small code model (no PIC)
  - 15.10.2 Large code model (no PIC)
  - 15.10.3 Medium code model (no PIC)
  - 15.10.4 Small PIC code model
  - 15.10.5 Large PIC code model
  - 15.10.6 Medium PIC code model
- 15.11 Summary
## 16 Performance

16.1 Optimizations

- 16.1.1 Myth about fast languages
- 16.1.2 General advice
- 16.1.3 Omit stack frame pointer
- 16.1.4 Tail recursion
- 16.1.5 Common Subexpressions Elimination
- 16.1.6 Constant propagation
- 16.1.7 (Named) return value optimization
- 16.1.8 Influence of branch prediction
- 16.1.9 Influence of execution units
- 16.1.10 Grouping reads and writes in code

16.2 Caching

- 16.2.1 How to use cache effectively?
- 16.2.2 Prefetching
- 16.2.3 Example: binary search with prefetching
- 16.2.4 Bypassing cache
- 16.2.5 Example: matrix initialization

16.3 SIMD instruction class

16.4 SSE and AVX extensions

- 16.4.1 Assignment: sepia filter

16.5 Summary

## 17 Multithreading

17.1 Processes and threads

17.2 What makes multithreading hard?

17.3 Execution order

17.4 Strong and weak memory models

17.5 Reordering example

17.6 What volatile is and is not

17.7 Memory barriers

17.8 Introduction to pthreads

- 17.8.1 When to use multithreading
- 17.8.2 Creating threads
- 17.8.3 Managing threads
- 17.8.4 Example: distributed factorization
- 17.8.5 Mutexes
- 17.8.6 Deadlocks
- 17.8.7 Livelocks
- 17.8.8 Condition Variables
- 17.8.9 Spinlocks

17.9 Semaphores

17.10 How strong is Intel 64?

17.11 What is lock-free programming?

17.12 C11 memory model

- 17.12.1 Overview
- 17.12.2 Atomics
- 17.12.3 Memory orderings in C11
- 17.12.4 Operations

17.13 Summary
IV Appendices

Appendix A. Using gdb 366

Appendix B. Using Make 374

Appendix C. System calls 379
  read .................................................................................................................. 379
  write .................................................................................................................. 380
  open ................................................................................................................... 381
  close ................................................................................................................... 382
  mmap .................................................................................................................. 382
  munmap ............................................................................................................. 383
  exit ..................................................................................................................... 384

Appendix D. Performance tests information 385

Bibliography 388
Introduction

This book aims to help you developing a consistent vision of the domain of low-level programming. We want to enable a careful reader to:

- Freely write in assembly language.
- Understand the programming model of Intel 64.
- Write maintainable and robust code in C11.
- Understand the compilation process and decipher assembly listings.
- Debug errors in compiled assembly code.
- Use appropriate models of computation to greatly reduce program complexity.
- Write performance-critical code.

There are two kinds of technical books: ones used as a reference and ones used to learn. This book is, without doubt, of second kind. It is pretty dense on purpose, and in order to successfully digest the information we highly suggest continuous reading. To quickly memorize new knowledge you should try to connect it with the knowledges you are already familiar with. That is why we tried to base the explanation of each topic on top of the previous ones whenever possible.

This book is written for programming students, intermediate-to-advanced programmers and low-level programming enthusiasts. The prerequisites are a basic understanding of binary and hexadecimal systems and a basic knowledge of Unix commands.

Questions and answers

Throughout this book you will encounter numerous questions. Most of them are meant to make you think again about what you have just learned, but some of them encourage you to do an additional research, pointing to the relevant keywords.

We propose the answers to these questions in our GitHub page, which also hosts all listings and starting code for assignments, updates and other goodies.


There you can also find several preconfigured virtual machines with Debian Linux installed, with and without GUI, which allows you to start practicing right away without spending time to set up your system. More information can be found in Section 2.1.

We start with the very simple core ideas of what a computer is, explaining concepts of model of computation and computer architecture. We expand the core model with extensions until it become adequate enough to describe a modern processor as a programmer sees it. From the Chapter 2 onward we start programming in the real assembly language for Intel 64 without resorting to older 16-bit architectures, that are often taught for historical reasons. It allows us to see the interactions between applications and operating system through the system calls interface and the
specific architecture details such as endianness. After a brief overview of legacy architecture features, some of which are still in use, we study virtual memory in great details and illustrate its usage with the help of procfs and examples of using mmap system call in assembly. Then we dive into the process of compilation, overviewing preprocessing, static and dynamic linking. After exploring interrupts and system calls mechanisms in greater details, we finish the first part with a chapter about different models of computations, studying examples of finite state machines, stack machines and implementing a fully functional compiler of Forth language in pure assembly.

The second part is dedicated to the C language. We start from the language overview, building a core understanding of its model of computation necessary to start writing programs. In the next chapter we study the type system of C and illustrate different kinds of typing, ending with the talk about polymorphism and providing exemplary implementations for different kinds of polymorphism in C. Then we study the ways of correctly structuring the program by splitting it into multiple files and also viewing the effect it produces on the linking process. The next chapter is dedicated to the memory management, input and output. After that, we elaborate the three facets of each language: syntax, semantics and pragmatics, and concentrate on the first and the third ones. We see, how the language propositions are transformed into abstract syntax trees, what is the difference between undefined and unspecified behavior in C and what effect does the language pragmatics have on the assembly code produced by the compiler. In the end of the second part, we dedicate a chapter to the good code practices to give an idea of how the code should be written depending on specific requirements to it. The sequence of the assignments for this part is ended by the rotation of a bitmap file and a custom memory allocator.

The final part is a bridge between the two previous ones. It dives into the translation details such as calling conventions and stack frames and advanced C language features, requiring a certain understanding of assembly, such as volatile and restrict keywords. We provide an overview of several classic low-level bugs such as stack buffer overflow, which can be exploited to induce an unwanted behavior in the program. The next chapter tells about shared objects in great details and studies them on assembly level, providing minimal working examples of shared libraries written in C and assembly. Then, a relatively rare topic of code models is discussed. The following chapter studies the optimizations modern compilers are capable of and how that knowledge can be used to produce readable and fast code. We also give an overview of performance amplifying techniques such as specialized assembly instructions usage and cache usage optimization. This is followed by an assignment where you will implement a sepi filter for an image using specialized SSE instructions and measure its performance. The last chapter introduces multithreading via pthreads library usage, memory models and reorderings, which anyone doing multithreaded programming should be aware of, and elaborates the need for memory barriers.

The appendices include short tutorials on gdb (debugger), make (automated build system), a table of most frequently used system calls for reference and system information to make performance tests given throughout the book easier to reproduce. They should be read when necessary, but we recommend you getting used to gdb right when you start assembly programming in Chapter 2.

Most illustrations were produced using VSVG library aimed to produce complex interactive vector graphics, written by Alexey Velikiy (http://www.corpglory.com). The sources for the library and book illustrations are available at VSVG Github page: https://github.com/corpglory/vsvg.

We hope that you find this book useful and wish you an enjoyable read!
Part IV

Appendices
Bibliography


