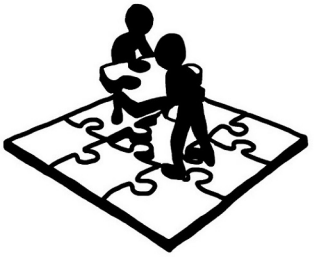
Program Programming Programmer

밑바닥부터 만드는 컴퓨팅 시스템

불 논리부터 컴퓨터 아키텍처, 운영체제까지

노암 니산·시몬 쇼켄 지음 | 김진홍 옮김



The Elements of Computing Systems

Building a Modern Computer from First Principles

Program
Programming
Programmer

10

The Elements of Computing Systems

THE ELEMENTS OF COMPUTING SYSTEMS:
Building a Modern Computer from First Principles
by Noam Nisan and Shimon Schocken

밑바닥부터 만드는 컴퓨팅 시스템
불 논리부터 컴퓨터 아키텍처, 운영체제까지

초판 PDF 1.0 2019년 4월 29일 **지은이** 노암 니산, 시몬 쇼켄 **옮긴이** 김진홍 **펴낸이** 한기성 **펴낸곳** 인사이트 **편집** 백주옥 **등록번호** 제10-2313호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 연남로5길 19-5 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** http://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-245-8 이 책의 정오표는 http://blog.insightbook.co.kr에서 확인하실 수 있습니다.

The Elements of Computing Systems

밑바닥부터 만드는 컴퓨팅 시스템

불 논리부터 컴퓨터 아키텍처, 운영체제까지

노암 니산·시몬 쇼켄 지음 | 김진홍 옮김

인사이트 insight

단순한 것이 더 아름답다는 교훈을 주신 부모님께.

차례

옮긴이의 글

1970년대에 IBM에서 처음 개인용 컴퓨터(PC)가 판매된 이후로, 50년이 채 못 되어 이제 컴퓨터는 우리 생활에서 떼어놓을 수 없을 정도로 널리 쓰이고 있습니다. 처음에 컴퓨터는 업무용 기기로 사용되었지만, 가정용 PC와 노트북 컴퓨터를 거쳐 이제는 거의 모든 사람이 들고 다니는 휴대전화에 이르기까지 다양한 형태로 보급되었습니다. 그뿐 아니라 이제 자동차, 냉장고, TV, 손목시계 등 우리가 일상에서 쓰는 수많은 물건에, 단순한 계산 모듈로서가 아닌 프로그래밍 가능한 컴퓨터가 탑재되고 있습니다. 정말로 이제는 우리가 과연 컴퓨터 없이 어떤 일이라도 할 수 있을까에 대한 의문이 들 정도로, 컴퓨터는 수많은 곳에서 당연하게 쓰이고 있습니다.

이렇듯 컴퓨터가 과거보다 훨씬 우리에게 친숙하고 중요한 도구가 되었지만, 역설적으로 우리가 컴퓨터의 작동 원리를 이해하기는 더 어려워졌습니다. 일반인에게 컴퓨터는 너무나 복잡하고 정교하고 거대한 시스템이라, 대학교에서 컴퓨터 과학 관련 전공을 하지 않고서는 내부 구조가 어떤지, 어떤 원리로 동작하는지 거의 알 방법이 없습니다. 물론 컴퓨터 하드웨어 및 소프트웨어 설계자가 굳이 사용자가 컴퓨터의 작동 원리를 잘 알지 않아도 쉽게 쓸 수 있도록 복잡한 부분은 최대한 감추고 보이는 부분은 간단하게 만든 덕도 있겠지요. 하지만 컴퓨터 내부가 궁금한 사람들에게도 컴퓨터의 전체 구조와 작동 원리를 배울 수단이나 자료가 마땅치 않은 것도 사실입니다. 그런 이유로 심지어, 이 책의 저자들이 지적했듯이 컴퓨터 과학을 전공한 학생이

나 현업에서 프로그래밍을 활용하는 사람들도 컴퓨터의 작동 원리를 종합적으로 이해하지는 못한 경우가 많습니다. 나무는 봐도 숲은 보지 못한다는 것이죠.

이 책은 위와 같은 문제의식을 바탕으로 탄생했습니다. 컴퓨터의 역사에 관심이 조금 있는 분들은, 현재 아이폰으로 유명한 애플사가 1976년 스티브 잡스와 스티브 워즈니악이 차고에서 직접 설계하고 손으로 만든 애플 컴퓨터에서 시작했다는 일화를 들어보신 적이 있을 겁니다. 이 책은 독자들을 그때 그 시절, 혼자서 하드웨어부터 소프트웨어까지 모두 만들 수 있었던 시절로 데려갑니다. 이 책은 컴퓨터의 가장 기초 부품이 되는 게이트 칩에서 시작해서, RAM, CPU 등의 핵심 칩들을 구성하고, 프로그래밍이 가능하도록 고수준 언어를 기계어로 번역하는 프로그램을 만들고, 컴퓨터를 구동시킬 운영체제와 그 위에서 실행시킬 응용프로그램을 작성하는 방법에 이르기까지 단계적으로 하나하나 설명합니다. 이 과정을 모두 성공적으로 거치고 나면 짠! 하고 직접 만든 나만의 컴퓨터가 하나 생기는 것이죠.

이렇게 컴퓨터를 만들 때 거치는 단계들은 원래라면 각각 컴퓨터 과학 전공과목 하나에 해당하는 내용입니다. 물론 컴퓨터 구성에 필요한 가장 기초적인 내용만 설명하기는 하지만, 구슬이 서 말이라도 꿰어야 보배랬다고, 이렇게 처음부터 끝까지 만들어 보는 경험이 따로따로 내용을 배우기보다 훨씬 더 쉽고 직관적으로 와닿는 것 같습니다. 게다가 배우는 과정이 훨씬 더 재밌습니다. 각 분야를 따로 공부하려면 너무 양이 많고 복잡해서 막막하게 느껴지는데, 책 한 권 분량의 실습을 통해 전체적인 컴퓨터의 구조를 조망할 수 있으니 흥미를 잃지 않고 배울 수 있었습니다. 실습에 필요한 자료와 도구들이 웹사이트에 올라와 있어, 약간의 컴퓨터 지식과 프로그래밍 지식이 있으면 쉽게 따라 해 볼 수 있었다는 점도 장점입니다.

이 책은 컴퓨터 구조에 관심이 있는 일반인뿐 아니라 프로그래머, 컴퓨터 과학을 전공한 사람들에게도 도움이 되는 책입니다. 이 책의 설명 방식은 모르던 내용을 새로 배우는 데도, 알던 지식을 종합적으로 이해하는 데도 효과

적이라 생각합니다. 이 책이 저에게 도움이 되었던 만큼, 다른 분들도 이 책을 읽고 많은 것을 얻어가실 수 있으면 좋겠습니다.

끝으로 좋은 책을 저술해 준 이 책의 저자들과 이 책을 번역할 기회를 주고 오랜 번역 기간 동안 여러모로 도와주신 인사이트 출판사 관계자 여러분께 감사드립니다.

지은이의 글

듣기만 하면 잊어버리게 마련이고, 보면 기억만 할 수 있을 뿐이고, 행해야 이해할 수 있다.[1]

— 공자(기원전 551~479)

옛날 옛적, 모든 컴퓨터 전문가는 컴퓨터의 원리를 전반적으로 이해하고 있었다. 예전에는 하드웨어, 소프트웨어, 컴파일러 및 운영체제 사이의 상호작용이 단순했기에 컴퓨터의 작동 방식을 논리 정연하게 그려볼 수 있었다. 하지만 현대에 들어 컴퓨터 기술이 점점 더 복잡해지면서 그 원리를 명료하게 이해하기 어려워졌다. 이제 컴퓨터 과학의 가장 기본적인 개념이나 기법은, 여러 단계의 모호한 인터페이스 및 구현 안에 숨겨져 보이지 않는다. 이런 복잡함 탓에 컴퓨터 과학 교육과정은 전체 분야에서 일부만을 다루는 과목들로 세분화 되는 것을 피할 수 없게 되었다.

우리는 컴퓨터 과학 전공생들이 나무만 보고 숲을 놓치는 경우가 많다는 생각에 이 책을 썼다. 보통 학생들은 추상화, 인터페이스, 실제 구현을 통해 긴밀하게 연결된 하드웨어와 소프트웨어 시스템의 전체 그림을 조망할 여유도 없이 프로그래밍과 이론, 엔지니어링 과목들을 연달아 수강한다. 많은 학생과 전문가가 이런 복잡하고 규모가 큰 시스템을 실제로 다뤄보지 못해서 컴퓨터 안에서 어떤 일이 일어나는지 잘 모른다는 느낌을 받는다.

1 (옮긴이) 이 말은 실제로 공자가 한 적이 없는 말이다. 그보다는 순자의 유효편(儒效篇)에서 나온 不聞不若聞之, 聞之不若見之, 見之不若知之, 知之不若行之; 學至於行之而止矣(듣지 않는 것은 듣는 것만 못하고, 듣는 것은 보는 것만 못하며, 보는 것은 아는 것만 못하고, 아는 것은 행하는 것만 못하다. 배움이란 행하는 데에 이르러야 완성되는 것이다)에서 따온 말로 보인다.

우리는 컴퓨터 작동 원리를 이해하는 가장 좋은 방법은 바로 밑바닥부터 컴퓨터를 구성해 보는 거라 생각했다. 이 생각은 다음 생각으로 이어졌다. 단순하지만 충분히 강력한 컴퓨터 시스템을 설계하자. 밑바닥 기본 논리 게이트부터 시작해서 하드웨어 플랫폼 및 소프트웨어 시스템을 학생들이 직접 만들게 하자. 그리고 하는 김에 제대로 하자. 왜냐면 기본 원리들을 바탕으로 범용 컴퓨터를 만드는 일은 아주 큰 작업이기 때문이다. 그래서 우리는 실습을 통해 컴퓨터를 직접 구성하면서도 효과적으로 대규모 하드웨어 및 소프트웨어 개발 프로젝트를 계획하는 방법을 설명하는 독창적인 교육법을 창안했다. 또한 우리는 단계적인 접근법을 통해, 몇 개의 기초 구성 블록에서 대단히 복잡하고 유용한 시스템이 구성되는 방식을 보여주려 했다.

범위

이 책은 하드웨어 및 소프트웨어를 순서대로 만들어 보면서 컴퓨터 과학의 주요 지식들을 설명한다. 이로서 다른 컴퓨터 과학 과목에서 배우는 이론적, 응용적 기술이 실제로는 어떻게 사용되는지를 보여준다. 특히 다음과 같은 주제들을 다룬다.

- **하드웨어:** 논리 게이트, 불 산술Boolean arithmetic, 멀티플렉서multiplexor, 플립플롭flip-flop, 레지스터register, 램RAM 유닛, 카운터counter, 하드웨어 기술 언어 Hardware Description Language, HDL, 칩 시뮬레이션 및 테스트
- **아키텍처:** ALU/CPU 설계 및 구현, 기계 코드, 어셈블리assembly 언어 프로그래밍, 주소 지정 방법addressing mode, 메모리 매핑 입력/출력I/O
- **운영체제:** 메모리 관리, 수학 라이브러리, 기본 I/O 드라이버, 스크린 관리, 파일 I/O, 고수준 언어 지원
- **프로그래밍 언어:** 객체 기반 설계 및 프로그래밍, 추상 데이터 유형, 범위 지정 규칙scoping rules, 구문 및 의미syntax and semantics, 참조reference
- **컴파일러:** 어휘 분석lexical analysis, 하향식 파싱top-down parsing, 기호 테이블

symbol table, 가상 스택 기반 머신virtual stack-based machine, 코드 생성code generation, 배열 및 객체 구현

- **데이터 구조 및 알고리즘:** 스택stack, 해시 테이블hash table, 리스트, 재귀recursion, 산술 알고리즘, 기하학적 알고리즘, 실행시간 고려
- **소프트웨어 공학:** 모듈식 설계, 인터페이스/구현 패러다임, API 설계 및 문서화, 사전 테스트 계획, 대규모 프로그래밍, 품질 보증

이 주제들의 목적은 매우 명확하다. 바로 밑바닥부터 컴퓨터를 구축하는 것이다. 사실 우리가 이 주제들을 선택한 이유도 컴퓨터를 만들기 위함이다. 이 책은 온전히 동작하는 컴퓨터 시스템을 만드는 데 꼭 필요한 주제만을 다루고 있다. 그 결과, 응용 컴퓨터 과학의 기본적인 개념들이 여기에 많이 포함된다.

교육과정

이 책의 대상 독자는 컴퓨터 과학 및 기타 공학 분야를 전공하는 학부생 및 대학원생이다. 이 책은 일반적인 컴퓨터 과학 교육과정을 '위에서 아래까지' 다루고 있으므로, 교육과정 중 어느 시점에서나 가르칠 수 있다. 보통은 프로그래밍을 배운 직후의 'CS-2'나, 교육과정 마지막 즈음의 종합용 강의인 'CS-199' 정도에 강의하는 것이 자연스러울 것이다.[2] 아마도 전자는 시스템 중심의 컴퓨터 과학 개론이, 후자는 프로젝트 중심으로 시스템을 구성해 보는 통합적인 강의가 될 것이다. 강의 이름으로는 '컴퓨터 과학의 구조적 개론'이나, '컴퓨터 시스템의 요소', '디지털 시스템 구축', '컴퓨터 구축 워크숍', '컴퓨터를 만들어 봅시다' 정도가 적당할 것이다. 이 책은 세부 주제를 선택하는 방식이나 강의 속도에 따라 1학기 또는 2학기 분량으로 가르칠 수 있다.

2 (옮긴이) 미국 대학들이 보통 강의 번호를 붙이는 방식에 따르면, CS-2에서 CS는 컴퓨터 과학(Computer Science), 맨 앞 번호 2는 강의의 난이도 또는 해당 학년을 뜻한다. 따라서 이 말은 컴퓨터 과학과 1학년 말~2학년을 위한 강의로 적당하다는 뜻이다.

이 책에서 다루는 내용은 모두 설명되어 있으므로, 어떤 종류의 언어든 프로그래밍 지식만 있으면 된다. 따라서 컴퓨터 과학 전공자 외에도, 컴퓨터를 잘 아는 사람 중에 하드웨어 아키텍처, 운영체제 및 최신 소프트웨어 공학에 대한 지식을 한꺼번에 배우고 싶어하는 사람들에게도 적합하다. 이 책과 함께 제공되는 웹사이트로 독학할 수도 있으며, 프로그래밍 수업을 들은 공학 및 과학 분야의 학생들에게도 알맞은 교과서가 될 것이다.

구성

서론에서는 이 책의 접근법과, 핵심 하드웨어 및 소프트웨어 추상화에 대해 미리 살펴본다. 서론은 1~12장의 준비과정이며, 각 장은 핵심 하드웨어 및 소프트웨어 추상화, 구현 방법, 그리고 실제로 만들고 테스트해 보는 프로젝트들을 설명한다. 처음 5개 장은 간단한 현대식 컴퓨터 하드웨어 플랫폼 구축에 중점을 둔다. 나머지 7개 장에서는 일반적인 소프트웨어 계층구조를 설계하고 구현하는 방법과, 객체 기반 프로그래밍 언어 및 간단한 운영체제 구축법에 대해 설명한다. 그림 P.1에 전체 계획이 묘사되어 있다.

그림 P.1 책에서 다루는 과정 지도. 동그라미 안에 해당하는 장 번호가 쓰여 있다.

이 책은 **추상화**abstraction를 먼저 설명한 다음 **구현**implementation하는 방식으로 서술되어 있다. 각 장은 그 장에 관련된 개념 및 일반적인 하드웨어 또는 소프트웨어 시스템을 설명하는 **배경** 절로 시작한다. 그다음 절은 **명세** 절로, 시스템 추상계층이 제공하는 명확한 기능을 제시한다. **명세** 절에서 **무엇**을 이야기한 다음, **구현** 절에서는 **어떻게** 그 추상화 계층을 구현하는지 설명한다. 그다음 절은 **정리**로, 그 장에서 기억할 만한 내용들을 다시 짚어본다. 각 장의 마지막은 앞에서 설명한 시스템을 실제로 구현하는 단계별 방법 및 테스트에 필요한 자료와 소프트웨어 도구를 소개하는 **프로젝트** 절로 마무리된다.

프로젝트

이 책에서 설명한 컴퓨터 시스템은 **진짜**로 만들 수 있다. 그리고 실제로 작동한다! 시간과 노력을 들여 차근차근 컴퓨터를 만들어 본 독자는, 단순히 관련 지식을 읽어 봤을 때보다 비교할 수 없을 만큼 컴퓨터 시스템에 대한 이해도가 높아질 것이다. 그래서 이 책은 소매를 걷어붙이고 기꺼이 컴퓨터를 밑바닥부터 구축해 볼 적극적인 독자를 대상으로 한다.

각 장은 독립적인 하드웨어나 소프트웨어 개발 프로젝트에 대한 설명이 담겨있다. 컴퓨터 플랫폼을 구성하는 앞부분 네 개의 프로젝트는, 단순한 하드웨어 기술 언어Hardware Description Language, HDL를 사용하여 구축되며 이 책과 함께 제공하는 하드웨어 시뮬레이터에서 시뮬레이션된다. 이어지는 다섯 개 소프트웨어 관련 프로젝트(어셈블러, 가상 머신 I 및 II, 컴파일러 I 및 II)는 어떤 종류든 최신 프로그래밍 언어를 이용해서 작성할 수 있다. 뒷부분의 세 프로젝트(저수준 프로그래밍, 고수준 프로그래밍 및 운영체제)는 앞선 프로젝트에서 구현한 어셈블리 언어와 고수준 언어로 작성된다.

프로젝트 팁 이 책의 프로젝트는 모두 12개다. 각 프로젝트는 일반적인 대학 강의 기준으로 한 주 분량의 숙제에 해당한다. 각 프로젝트는 완전히 독립적

이라, 원하는 순서대로 수행할 수(또는 건너 뛸 수) 있다. 물론 '완벽하게 경험'하려면 순서에 맞춰 프로젝트를 모두 수행해야 하지만, 여러 선택지 중 하나일 뿐이다.

이 책을 토대로 강의할 때 우리는 두 가지 점에서 타협한다. 첫째는 너무 뻔한 경우를 제외하고 최적화는 다른 더 자세한 강의로 미뤄둔다는 점이다. 둘째는 미리 검증된 테스트 파일(소스 프로그램)을 제공해서, 학생들이 번역기들(어셈블러, VM 구현 및 컴파일러)을 개발할 때 입력 파일이 잘못되었는지 신경 쓰지 않아도 되게 해주는 것이다. 따라서 예외 및 오류를 처리하는 코드를 작성할 필요가 없으므로 프로젝트가 훨씬 쉬워진다. 물론 잘못된 입력을 처리하는 일은 매우 중요하지만, 프로그래밍 및 소프트웨어 설계 과목 같은 데서 이 기법을 배울 수 있을 것이다.

소프트웨어

이 책의 웹사이트(*www.idc.ac.il/tecs*)에는 이 책에 설명된 하드웨어 및 소프트웨어 시스템을 구축하는 데 필요한 모든 도구와 자료가 있다. 여기에는 하드웨어 시뮬레이터, CPU 에뮬레이터, VM 에뮬레이터, 실행 가능한 어셈블러, 가상 머신, 컴파일러 및 운영체제가 포함된다. 또한 이 웹사이트에는 200여 개의 테스트 프로그램과 테스트 스크립트가 있어서, 12개 프로젝트 각각에 대해 단계별 개발과 단위 테스트가 가능하다. 제공된 모든 소프트웨어 도구 및 프로젝트 자료는 윈도우나 리눅스가 설치된 컴퓨터에서 그대로 사용 가능하다.

감사의 글

이 책에서 제공되는 모든 소프트웨어는 이스라엘의 새로운 대학인, 헤르츨리야 학제 간 연구 센터 내 에피 아라지 컴퓨터 과학과Efi Arazi School of Computer Science of the Interdisciplinary Center Herzliya 학생들이 개발했다. 수석 소프트웨어 아

키텍트는 야론 우크라이니츠Yaron Ukrainitz, 개발자는 이프타치 아밋Iftach Amit, 니어 로젠Nir Rozen, 아사프 개드Assaf Gad 및 하다르 로젠시어Hadar Rosen-Sior였다. 이 학생 개발자들과 함께 일하는 것은 큰 즐거움이었으며, 이들을 가르칠 수 있었던 건 우리에게는 행운이고 자랑거리였다. 또한 우리는 이 책을 쓰게 된 계기가 되었던 강의에서 조교를 맡고 도움을 줬던 무아위야 아카시Muawyah Akash, 데이비드 라비노비츠David Rabinowitz, 란 나복Ran Navok 및 야론 우크라이니츠Yaron Ukrainitz에게 감사드린다. 그리고 대니 시드너Danny Seidner 박사의 훌륭한 지도하에 프로젝트에 참여했던 조너선 그로스Jonathan Gross, 오렌 바라네스Oren Baranes와, 잭 언어Jack Language 통합 개발환경을 설계해 준 우리 자이라Uri Zeira, 오렌 코헨Oren Cohen, 그리고 오픈 소스 문제에 대해 조언을 해준 탤 아키투브Tal Achituv, 책을 세심하게 읽고 꼼꼼하게 조언해 준 아라이 슈넬Aryeh Schnall에게 감사의 말을 전한다.

정규 업무를 하면서 책을 쓰는 일은 쉽지 않았기에, 이 어려운 시기에 대신 일을 봐줬던 에피 아라지 컴퓨터 과학과 행정 담당인 에스티 로멤Esti Romem에게 감사하고 싶다. 마지막으로, 우리는 이 책의 초기 버전에 있던 오류들을 인내하며 버그 리포트를 통해 책 내용을 갈고 닦는 데 도움을 준 수많은 학생들에게 빚을 졌다. 그 과정에서, 학생들이 "실수는 발견으로 통하는 문이다"라는 제임스 조이스James Joyce의 교훈을 배웠길 바란다.

노암 니산Noam Nisan

시몬 쇼켄Shimon Schocken

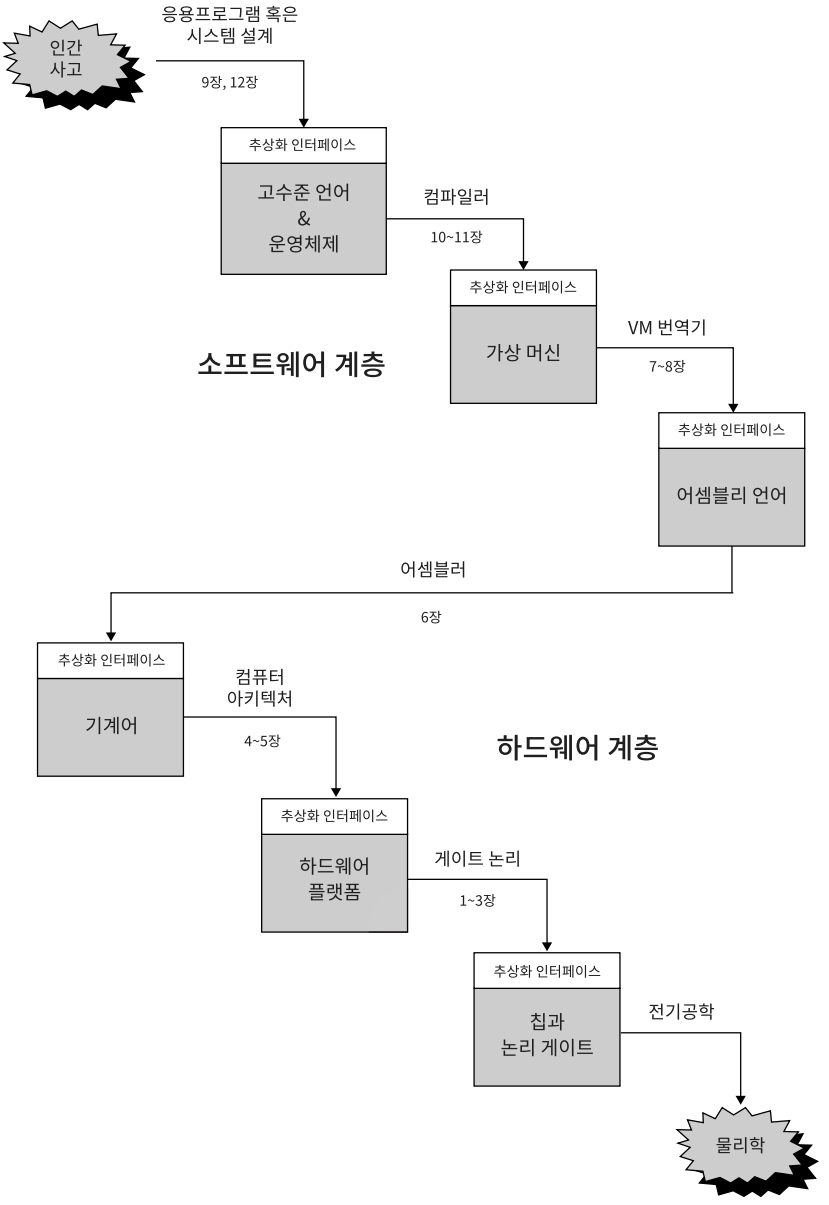


그림 I.1 전형적인 컴퓨터 시스템의 설계에 포함되는 주요 추상화 계층. 각 단계는 그 아래 단계의 추상화된 서비스 및 구성 블록을 통해 구현된다.

서론

안녕, 밑바닥 세상아

여행에서 진짜 발견이란, 새로운 장소를 가는 것뿐 아니라 새로운 시각을 갖는 것이다.

— 마르셀 프루스트(Marcel Proust, 1871~1922)

이 책은 발견하는 여행이다. 이제 막 독자들은 세 가지를 배울 참이다. 컴퓨터가 어떻게 작동하는지, 복잡한 문제들을 어떻게 다루기 좋은 개별 요소로 쪼개는지, 그리고 대규모 하드웨어 및 소프트웨어 시스템을 어떻게 개발하는지다. 이 과정은 실제 작동하는 완전한 컴퓨터 시스템을 밑바닥부터 직접 만들어 내는 작업이 될 것이다. 그리고 이 실습 과정을 통해 컴퓨터 그 자체보다 훨씬 더 중요하고 일반적인 지식을 함께 얻게 될 것이다. 심리학자 칼 로저스Carl Rogers는 "행동에 유의미하게 영향을 미치는 학습방법은, 경험을 통해 스스로 발견하고 스스로 적응하며 진리를 완전히 이해하는 방법뿐이다"라고 말했다. 이 장에서는 앞으로 우리에게 놓인 발견, 진리, 경험들을 스케치해 보려 한다.

위쪽 세상

프로그래밍 과목을 들어 본 적이 있다면, 아마도 시작 부분에서 다음 쪽의 코드와 같은 프로그램을 접했을 것이다. 다음 프로그램은 **잭**Jack이라는, 오브젝

트 기반의 간단한 고수준 언어high-level language로 작성되었다.

```
// 101 프로그래밍 과목의 첫 예시
class Main {
  function void main() {
    do Output.printString("Hello World");
    do Output.println(); // 새 라인
    return;
  }
}
```

"Hello World" 같은 프로그램은 겉으로만 간단해 보일 뿐이다. 이런 프로그램이 컴퓨터에서 **실제로 작동**하는 데 무엇이 필요한지 생각해 본 적이 있는가? 한번 그 속을 들여다 보자. 우선 이 프로그램은 그저 텍스트 파일에 저장된 문자 덩어리일 뿐임에 주목하자. 그러므로 컴퓨터를 이해할 수 있으려면 먼저 텍스트를 분석해서 그 의미를 찾아낸 후에, 저수준 언어로 다시 표현해야 한다. **컴파일**compilation이라는 이 정교한 번역 과정을 거쳐도 그 결과물은 여전히 기계 수준 코드로 된 텍스트 파일이다.

물론 기계어도 미리 약속된 2진 코드로 구성된 추상화 개념이다. 따라서 기계어를 실현하려면 **하드웨어 아키텍처**hardware architecture가 반드시 필요하다. 이 아키텍처는 레지스터, 메모리 장치, ALUArithmetic Logic Unit 같은 **칩**chip들로 구현된다. 이런 하드웨어들은 모두 **기초 논리 게이트**elementary logic gate를 집적해서 만들어진다. 또한 기초 논리 게이트들은 Nand나 Nor 같은 기본 게이트들로 구성할 수 있다. 기본 게이트들은 보통 트랜지스터로 구현된 몇 개의 **스위치 장치**로 이루어진다. 그리고 각 트랜지스터들은? 잠깐, 여기서 더 깊이 들어가진 않겠다. 여기부터는 컴퓨터 과학이 아니라 물리학의 영역이기 때문이다.

독자는 이렇게 생각할 수 있다. "내 컴퓨터에서 프로그램을 컴파일하고 실행하는 건 훨씬 쉬운데. 그냥 아이콘 몇 개 클릭하거나 명령어 몇 개 입력하기만 하면 되잖아!" 정말로, 현대 컴퓨터 시스템은 꼭대기만 보이는 거대한

빙산 같다. 이런 지식은 컴퓨터 시스템에 대한 개략적이고 피상적인 이해일 뿐이다. 만약 여러분에게 이 물 밑 세계를 탐험하고 싶다는 생각이 들었다면, 여러분은 행운아다! 그 아래 세상은 컴퓨터 과학에서 가장 아름다운 부분들로 이루어진 매혹적인 세상이다. 이 수면 아래 세계를 잘 이해하는지가, 응용 프로그램뿐 아니라 복잡한 하드웨어 및 소프트웨어 기술을 창조할 줄 아는 수준 높은 개발자와, 그저 단순한 프로그래머를 가른다. 밑바닥부터 온전한 컴퓨터 시스템을 만들어 보는 것이, 이런 기술들이 어떻게 작동하는지 뼛속 깊이 이해하는 가장 좋은 방법일 것이다.

추상화

온전한 컴퓨터 시스템을 기초 논리게이트만 이용해 독자 혼자서 밑바닥부터 구성하는 것이 가능할지 궁금할지도 모르겠다. 컴퓨터는 엄청나게 복잡한 시스템이지 않는가! 그래서 우리는 이 복잡한 프로젝트를 **모듈**로 쪼개서, 각 모듈별로 한 개 장씩 따로 할애해 다루려 한다. 그러면 어떻게 이 모듈들을 별개로 설명하고 구성할지가 궁금할 것이다. 당연히 이 모듈들은 모두 서로 연관되어 있다! 책 전반에 걸쳐 설명하겠지만, 좋은 모듈 설계는 서로 독립적으로 만드는 것이다. 실제로 이 모듈들은 아무렇게나 원하는 순서대로 만들어도 상관없다!

추상화라는, 사람에게만 있는 고유한 능력 덕에 이런 모듈식 접근은 효과가 좋다. 추상화는 일반적으로 어떤 개체의 본질을 포착하여 간결한 방식으로 구별해 내는 정신적 활동을 뜻하며, 예술이나 과학의 핵심이 된다. 컴퓨터 과학에서는 추상화 개념을 '개체가 어떻게 동작하는지'보다는 '개체가 무엇을 하는지'로 매우 구체적으로 규정한다. 이런 방식은 어떤 개체의 기능을 활용하는 데만 관심을 둔다. 여러 작업이나 정교한 설계, 기타 정보나 이야깃거리들은 사용자와 무관하다는 이유로 사용자가 볼 수 없게 개체의 구현 속에 숨겨진다. 전문가의 기본적인 업무는 이런 추상화를 표현하고, 구현하고, 사용