Program
Programming
Programmer

13

The Elements of Computing Systems 2/E

THE ELEMENTS OF COMPUTING SYSTEMS, 2/E
by Noam Nisan, Shimon Schocken

밑바닥부터 만드는 컴퓨팅 시스템 2판

불 논리부터 컴퓨터 아키텍처, 운영체제까지

전자책 1쇄 발행 2023년 5월 10일 **지은이** 노암 니산, 시몬 쇼켄 **옮긴이** 김진홍 **펴낸이** 한기성 **펴낸곳** ㈜도서출판인사이트
편집 백주옥 **등록번호** 제2002-000049호 **등록일자** 2002년 2월 19일 **주소** 서울시 마포구 연남로5길 19-5 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** https://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-412-4

The Elements of Computing Systems 2/E

밑바닥부터 만드는 컴퓨팅 시스템 2판

불 논리부터 컴퓨터 아키텍처, 운영체제까지

노암 니산·시몬 쇼켄 지음 | 김진홍 옮김

단순한 것이 더 아름답다는 교훈을 주신 부모님께

차례

옮긴이의 글

이 훌륭한 책의 1판을 번역한 지 얼마 되지 않은 것 같은데, 꽤 오랜만에 2판을 번역하게 되었습니다. 번역을 하다 보니, 2판이기는 하지만 1판과 주제와 구성만 유사하고 세세한 표현들은 거의 전부 새로 썼다고 해도 과언이 아닐 정도로 많은 내용이 바뀌거나 추가되었다는 사실을 알 수 있었습니다. 1판도 이미 매우 좋은 책이었는데 이렇게까지 열심히 개정을 하다니, 컴퓨터 개념을 알리고자 하는 저자들의 열정과 노력이 한층 더 마음에 와 닿는 느낌이었습니다.

저자들이 책에서도 여러 번 강조하는 이야기지만, 컴퓨터에 관한 어떤 개념을 배우고 이해하는 데 가장 좋은 방법 중 하나는, 그 개념에 관한 무언가를 밑바닥부터 직접 만들어 보는 것입니다. 하드웨어도, 소프트웨어도 그렇죠. 이 책의 장점은 컴퓨터의 한 부분에 국한된 것이 아니라, 가장 기초적인 하드웨어에서부터 OS를 거쳐 테트리스 같은 꽤 복잡한 프로그램까지, 정말로 처음부터 끝까지 다 만들어 본다는 점에 있습니다. 현대 컴퓨터 시스템의 복잡성을 생각해 보았을 때, 이렇게 나무가 아닌 숲을 조망하는 관점의 책은 흔치 않을 것 같습니다. 그러면서도 나무에 해당하는 핵심적인 개념들을 놓치지 않고 설명하고 있습니다. 따라서 컴퓨터를 전공하지 않은 사람은 물론이고, 전공한 사람도 컴퓨터 시스템의 전체적인 그림을 파악하기 위해서라면 이 책을 읽어보는 것은 매우 좋은 선택이 될 것 같습니다.

1판에 이어 2판을 번역하면서, 개인적 시간을 내서 nand2tetris.org 홈페이지에 있는 영문 자료들도 번역해서 소개하면 좋았을 것 같다는 아쉬움이

조금 들었습니다. 변명 아닌 변명(?)을 하자면, 다행히 최근에 웹 브라우저에 있는 부가기능 중 자동 번역 기능이 매우 좋아진 덕분에 언어의 장벽이 크게 낮아져 예전보다는 무리 없이 읽을 수 있지 않을까 생각합니다. 책에서 다룬 내용들, 다른 사람들이 만든 프로젝트, Q&A 게시판 등이 있으니 책과 같이 읽어 보기를 추천합니다.

끝으로, 제가 번역한 2판 번역서도 컴퓨터를 이해하고자 하는 독자 여러분들의 여정에 조금이라도 도움이 되길 바랍니다. 감사합니다.

2023년 3월
옮긴이 김진홍

지은이의 글

듣기만 하면 잊어버리기 마련이고, 보기만 하면 기억할 수만 있을 뿐이고, 행해야 이해할 수 있다.[1]

공자(기원전 551~479)

21세기의 깨우친 사람이라면 BANG, 즉 비트Bit, 원자Atom, 뉴런Neuron, 유전자Genes의 핵심 개념을 잘 알아야 한다는 이야기가 있다. 비록 과학이 그 기본 작동 체계를 밝혀내는 데 크게 성공하기는 했지만, 원자, 뉴런, 유전자가 어떻게 동작하는지 완전히 파악하지 못할 가능성도 상당하다. 하지만 비트와 컴퓨터 시스템은 다행히도 예외다. 현대적인 컴퓨터가 굉장히 복잡하기는 하지만 어떻게 동작하는지, 어떻게 만들어지는지는 완전히 이해할 수 있다. 따라서 BANG을 경외의 눈으로 바라보더라도, 적어도 이 네 가지 개념 중 하나가 인간의 이해 아래 온전히 있다는 생각을 하면 즐거워진다.

실제로 컴퓨터 역사의 초창기에는 컴퓨터에 호기심이 많은 사람이라면 누구든지 컴퓨터가 어떻게 작동하는지 전반적으로 이해할 수 있었다. 예전에는 하드웨어와 소프트웨어 사이의 상호작용이 단순했기에 컴퓨터의 작동 원리를 논리정연하게 그려볼 수 있었다. 하지만 현대에 들어 컴퓨터 기술이 점점 더 복잡해지면서 그 원리를 명료하게 이해하기 어려워졌다. 이제 컴퓨터

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.

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 세분화되는 것을 피할 수 없게 되었다.

이 책을 쓴 이유는 컴퓨터 과학과 학생들 중에 나무만 보고 숲을 보지 못하는 학생들이 많다는 생각에서다. 보통 학생들은 프로그래밍, 이론, 엔지니어링에 대한 과목들을 연이어 들어야 하다 보니 잠시 멈춰 서서 전체적인 그림의 아름다움을 감상할 여유가 없다. 이 전체적인 그림은 하드웨어, 소프트웨어 및 응용프로그램 시스템이, 추상화, 인터페이스, 규약 기반 구현들이 그물처럼 긴밀하게 연결되어 있는 그림이다.

많은 학생과 전문가가 이렇게 복잡하고 규모가 큰 시스템을 실제로 다뤄보지 못해서 컴퓨터 안에서 어떤 일이 일어나는지 잘 모른다는 느낌을 받는다. 컴퓨터는 21세기의 가장 중요한 기기인데도 이런 느낌을 받는다는 건 좋지 않은 일이다.

우리는 컴퓨터의 작동 원리를 이해하는 가장 좋은 방법은 바로 밑바닥부터 컴퓨터를 구성해 보는 거라 생각했다. 이 생각은 다음 생각으로 이어졌다. 단순하지만 충분히 강력한 컴퓨터 시스템을 설계하자. 그리고 밑바닥 하드웨어 플랫폼에서 소프트웨어 계층까지 학생들이 직접 만들게 하자. 그리고 하는 김에 제대로 하자. 왜냐면 기본 원리를 바탕으로 범용 컴퓨터를 만드는 일은 매우 큰 작업이기 때문이다.

그래서 우리는 컴퓨터를 직접 만들어 보면서, 대규모 하드웨어 및 소프트웨어 개발 프로젝트를 효과적으로 계획하는 방법도 가르치는 독창적인 교육법을 창안했다. 그뿐만 아니라 우리는 단계적인 접근법을 통해, 몇 개의 기초 구성 블록에서 대단히 복잡하고 유용한 시스템이 구성되는 방식을 보여주려 했다. 또한 기본 원리를 주의 깊게 생각하고 모듈식으로 설계해 보면서 매우 복잡하고 유용한 시스템을 만들어 보는 성취감을 주려고 노력했다.

이 노력의 결과물이 'Nand to Tetris(Nand에서 테트리스까지)'라는 이름의 프로젝트가 되었다. 이 프로젝트는 Nand라는 가장 기초적인 논리 게이트에서 시작해서 12개의 서브 프로젝트를 수행하고 나면, 테트리스 게임뿐 아니라 여러분이 떠올릴 수 있는 어떤 프로그램도 실행할 수 있는 범용 컴퓨터를 완성하게 되는 실습 프로그램이다. 컴퓨터 시스템을 직접 설계하고 만들고 다시 설계하고 다시 만드는 과정을 통해 이 책을 썼고, 다른 학습자들도 똑같이 따라할 수 있도록 그 과정을 설명했다. 또한 nand2tetris.org 웹사이트를 개설해서, 이 과정을 배우거나 가르치고자 하는 사람이라면 누구나 모든 프로젝트 자료와 소프트웨어 도구를 사용할 수 있도록 모두 공개했다.

기쁘게도 이 프로젝트에 대한 반응은 뜨거웠다. 최근 전 세계 수많은 대학, 고등학교 코딩 부트 캠프, 온라인 플랫폼, 해커 클럽에서 Nand to Tetris 과정을 강의하고 있다. 이 책과 온라인 과정은 매우 인기가 높아졌으며, 고등학생부터 구글 엔지니어에 이르기까지 수천 명의 학습자가 Nand to Tetris를 최고의 교육 경험으로 꼽는 후기를 꾸준히 올리고 있다. "내가 만들 수 없는 것은 이해할 수도 없다"고 리처드 파인만Richard Feynman이 말했다. Nand to Tetris는 결국 창작을 통해 이해하는 방법이다. 분명히 사람들은 이 창작자 정신을 열정적으로 받아들이고 있다.

이 책의 초판이 출간된 이후로 우리는 수많은 질문과 의견 및 제안을 받았다. 그 피드백에 맞춰 온라인 자료를 고쳐 나가다 보니 Nand to Tetris의 웹 버전과 책 버전에 격차가 생기게 되었다. 그리고 책의 여러 부분을 좀더 명확하게 서술하고 구성도 더 낫게 바꿀 필요도 느꼈다. 그래서 이 수술을 최대한 미루다가 소매를 걷어붙이고 2판을 쓰기로 했고 드디어 이 책이 나오게 되었다. 이 머리말의 뒷부분에서는 2판의 내용을 설명하고, 초판과의 차이점을 말하는 것으로 마무리 지으려한다.

범위

이 책은 하드웨어 및 소프트웨어를 순서대로 만들어 보면서 컴퓨터 과학의 주요 지식들을 설명한다. 특히 다음과 같은 주제들은 실습을 통해 학습하도록 짜여 있다.

- 하드웨어: 불 산술Boolean arithmetic, 조합 논리combination logic, 순차 논리sequential logic, 논리 게이트의 설계 및 구현, 멀티플렉서multiplexor, 플립-플롭flip-flop, 레지스터register, 램RAM 유닛, 카운터counter, 하드웨어 기술 언어Hardware Description Language, HDL, 칩 시뮬레이션과 검증 및 테스트.
- 아키텍처: ALU/CPU 설계 및 구현, 클록 및 사이클, 주소 지정 방법addressing mode, 인출fetch 및 실행execute 논리, 명령어 집합, 메모리 매핑 입출력I/O.
- 저수준 언어low-level language: 단순 기계어(2진 및 기호) 설계 및 구현, 명령어 집합들, 어셈블리 프로그래밍, 어셈블러.
- 가상 머신Virtual machine: 스택 기반 오토마타stack-based automata, 스택 산술, 함수 호출 및 반환, 재귀 처리, 단순 VM 언어 설계 및 구현.
- 고수준 언어high-level language: 오브젝트 기반이자 자바스러운 언어 설계 및 구현: 추상 데이터 타입, 클래스, 생성자, 메서드, 범위 지정 규칙scoping rules, 구문 및 의미론syntax and semantics, 참조reference
- 컴파일러: 어휘 분석lexical analysis, 파싱parsing, 기호 테이블symbol table, 코드 생성code generation, 배열 및 객체 구현, 2단계 컴파일
- 프로그래밍: 어셈블러, 가상 머신 및 컴파일러 구현 및 API 제공. 구현에 쓰이는 프로그래밍 언어 제한 없음
- 운영체제: 메모리 관리, 수학 라이브러리, I/O 드라이버, 문자열 처리, 문자 출력, 그래픽 출력, 고수준 언어 지원 기능의 설계 및 구현
- 데이터 구조 및 알고리즘: 스택stack, 해시 테이블hash table, 리스트, 트리, 산술 알고리즘, 기하 알고리즘, 실행 시간 고려
- 소프트웨어 공학: 모듈식 설계, 인터페이스/구현 패러다임, API 설계 및

문서화, 단위 테스트unit testing, 사전 테스트 계획, 품질 보증, 대규모 프로그래밍

Nand to Tetris의 독특한 점은 이 모든 주제가, 현대적인 컴퓨터 시스템을 밑바닥부터 구축한다는 명확하고 중요한 목표에 따라 긴밀하게 연결된다는 점이다. 사실 이 주제들이 선택된 이유도, 고수준의 객체 기반 언어로 작성된 프로그램을 실행하는 다목적 컴퓨터 시스템을 구축하는 데 필요한 최소한의 주제들이었기 때문이다. 이제 살펴보겠지만, 이 주제들은 응용 컴퓨터 과학에서 가장 아름다운 개념들뿐만 아니라 기초적인 개념과 기법들을 포함하고 있다.

교육과정

Nand to Tetris 과정은 보통 학부생 및 대학원생 모두에게 해당하며, 독학하는 사람들에게 매우 인기가 있다. 이 과정은 일반적인 컴퓨터 과학 교육과정을 '위에서 아래까지' 다루고 있으므로, 교육과정 중 어느 시점에서나 가르칠 수 있다. 자연스러운 시점 두 가지는, 프로그래밍을 배우기 전 입문 과정으로 CS-2나, 교육과정 마지막 즈음에 종합하는 의미로 CS-99 정도가 알맞을 것이다. 전자는 응용 컴퓨터 과학을 미리 살펴보는 시스템 중심의 개론 강의가 될 것이고, 후자는 이전 교육 과정들의 공백을 채우는 프로젝트 기반의 총론 강의가 될 것이다.

인기 있는 강의 방식 중 하나는 전통적인 컴퓨터 아키텍처 강의와 컴파일 강의에서 핵심 주제만 뽑아서 하나로 합친 강의다. Nand to Tetris 과정은 'Nand to Tetris' 외에도, '컴퓨터 시스템의 요소', '디지털 시스템 구축', '컴퓨터 구조', '컴퓨터를 만들어 봅시다' 등 강의 목표에 따라 다양한 강의명으로 진행되고 있다.

이 책에서 다루는 프로젝트들은 상위 개념부터 시작해서 각각 6개 장, 6개 프로젝트로 이루어진 'I부: 하드웨어'와 'II부: 소프트웨어'까지 매우 모듈화된

방식으로 서술되어 있다. 모두 다 경험해 볼 것을 권하지만, 각각의 부를 따로 배워도 아무 문제없다. 책에 나온 프로젝트는 주제를 어떤 걸 선택하고 학습 속도를 어느 정도로 잡는지에 따라, 한 학기당 6~7주 정도 걸리는 두 학기 분량의 과정으로 나눌 수 있다.

이 책은 필요한 내용을 모두 담고 있다. 즉, 이 책에서 설명하는 하드웨어 및 소프트웨어를 만드는 데 필요한 모든 지식은 각 장의 프로젝트에 담겨 있다. 'I부: 하드웨어'에서는 어떤 선행지식도 필요하지 않으며, 프로젝트 1~6은 어떤 학생이나 독학자도 쉽게 접근할 수 있다. 'II부: 소프트웨어'의 프로젝트 7~12는 (어떤 종류의 고수준 언어든) 프로그래밍 지식이 필요하다.

Nand to Tetris 과정은 컴퓨터 공학 전공에 국한되지는 않는다. 따라서 컴퓨터 과학 전공자 외에도, 하드웨어 아키텍처, 운영체제, 컴파일, 소프트웨어 공학에 대한 지식을 하나의 과정으로 배우고 싶은 사람들이라면 어떤 전공의 학습자에게도 적합하다. 다시 한 번 말하지만, 필요한 선행지식은 오로지 프로그래밍(II부에 해당)뿐이다. 실제로 Nand to Tetris를 배우는 학생들 중에 컴퓨터 과학 개론을 듣고 나서 복수전공은 하지 않고 컴퓨터 과학을 더 배우고 싶어 하는 비전공생들이 많다. 또한 학생들 중에는 기술이 어떻게 작동하는지 '더 깊게' 이해해서, 더 나은 고급 프로그래머가 되고자 하는 소프트웨어 개발자들도 많다.

하드웨어 및 소프트웨어 산업에서 개발자가 급격하게 부족해지면서, 집중적이고 간결한 응용 컴퓨터 과학 프로그램에 대한 수요가 늘어나고 있다. 그에 따라 학습자가 학위 전체를 따지 않고도 취업 시장에 대비할 수 있도록 코딩 부트 캠프나 온라인 강의 모음 형식의 교육이 이뤄지고 있다. 그런 학습 프로그램은 적어도 프로그래밍, 알고리즘 및 시스템에 대한 실무 지식을 가르쳐야 할 것이다. Nand to Tetris는 하나의 강의 틀 안에서 시스템 요소를 다룰 수 있도록 특화되어 있다. 나아가 Nand to Tetris 프로젝트들은 다른 과정에서 배우는 알고리즘 및 프로그래밍 지식들의 대부분을 하나로 통합하고 실제로 수행하는 매력적인 수단이 된다.

자료

책에 설명된 하드웨어 및 소프트웨어 시스템을 구축하는 데 필요한 모든 도구는 Nand to Tetris 소프트웨어 모음에서 무료로 제공된다. 이 소프트웨어 모음에는 책에 나오는 하드웨어 시뮬레이터, CPU 에뮬레이터, VM 에뮬레이터(이상 모두 오픈 소스), 튜토리얼 및 실행 가능한 어셈블러, 가상 머신, 컴파일러, 운영체제를 모두 담고 있다. 또한 nand2tetris.org 웹사이트에는 200여 개의 테스트 프로그램과 스크립트 등 모든 프로젝트 자료가 올라와 있어서, 12개 프로젝트를 단계적으로 개발하고 단위 테스트를 할 수 있다. 소프트웨어 도구 및 프로젝트 자료는 윈도우, 리눅스, 맥OS가 실행되는 컴퓨터라면 그대로 활용할 수 있다.

구성

'I부: 하드웨어'는 1~6장으로 되어 있다. 1장은 불 대수를 소개하고 나서, 기초 Nand 게이트에서 시작해서 그 위에 다른 기초 논리 게이트를 만드는 방법을 설명한다. 2장은 조합 논리를 소개하고 가산기 집합을 거쳐 ALU를 구성한다. 3장은 순차 논리를 설명하고 레지스터와 메모리 디바이스를 거쳐 RAM을 구축한다. 4장은 저수준 프로그래밍에 대해서 이야기하고 기호 및 2진 형식의 기계어를 정의한다. 5장은 1~3장에서 구성한 칩들을 통합해서, 4장에서 만든 기계어로 작성된 프로그램을 실행할 수 있는 하드웨어 아키텍처를 만든다. 6장은 저수준 프로그램 번역을 설명하고 어셈블러 구성으로 막을 내린다.

'II부: 소프트웨어'는 7~12장으로 되어 있으며, 컴퓨터 과학 개론 수준의 프로그래밍 배경 지식이 필요하다(프로그래밍 언어는 무관). 7~8장은 스택 기반 오토마타를 소개하고 JVM 같은 가상 머신을 구축하는 방법을 설명한다. 9장은 자바와 유사한 객체 기반 고수준 언어를 설명한다. 10~11장에서는 파싱과 코드 생성 알고리즘을 다루고, 2단계 컴파일러를 구성한다. 12장은

다양한 메모리 관리와 대수 및 기하학적 알고리즘을 소개하고, 이를 실행하는 운영체제를 구현하는 방법을 설명한다. 이 OS는 II부에서 구현한 고수준 언어와 I부에서 만든 하드웨어 플랫폼 사이의 거리를 좁힐 수 있도록 설계되어 있다.

이 책은 추상화abstraction를 먼저 설명한 다음 구현implementation하는 방식으로 서술되어 있다. 각 장은 그 장에 관련된 개념 및 일반적인 하드웨어 또는 소프트웨어 시스템을 설명하는 '도입' 절로 시작한다. 그 다음은 '명세' 절로 시스템 추상화, 즉 어떤 방식으로든 제공될 다양한 서비스를 설명한다. '무엇'을 할지 이야기한 다음에는, '구현' 절에서 '어떻게' 그 추상화를 구현할지 설명한다. 그 다음은 '프로젝트' 절로, 그 장에서 설명하는 시스템을 구축하고 단위 테스트하는 데 필요한 소프트웨어 도구 및 테스트 자료, 단계별 지침을 이야기한다. 마지막 '정리' 절에서는 그 장에서 자세히 다루진 않았지만 주목할 만한 내용들을 다시 짚어본다.

프로젝트

이 책에서 설명한 컴퓨터 시스템은 진짜로 만들 수 있고, 실제로 작동한다! 이 책은 소매를 걷어붙이고 기꺼이 컴퓨터를 밑바닥부터 만들어 보려는 적극적인 독자를 대상으로 한다. 여러분이 시간과 노력을 들여 차근차근 만들어 본다면, 단순히 관련 지식을 읽어 볼 때보다 비교할 수 없을 만큼 컴퓨터에 대한 이해도가 높아질 것이다.

프로젝트 1, 2, 3, 5에서 만든 하드웨어는 단순한 하드웨어 기술 언어Hardware Description Language, HDL로 구현되고, 제공된 소프트웨어 기반 하드웨어 시뮬레이터로 시뮬레이션되는데, 이 방식은 실제 업계에서 하드웨어 설계자들이 쓰는 방법이다. 프로젝트 6, 7, 8, 10, 11(어셈블러, 가상 머신 I + II, 컴파일러 I + II)은 어떤 프로그래밍 언어로 작성해도 괜찮다. 프로젝트 4는 앞선 프로젝트에서 구현한 어셈블리 언어로 작성되며, 프로젝트 9와 12(간단한 컴퓨터 게임과 기본 운영체제)는 잭Jack이라는 자바와 유사한 고수준 언어로

작성되며, 그 컴파일러는 10, 11장에서 만든다.

이 책의 프로젝트는 모두 12개다. 각 프로젝트는 일반적인 대학 강의 기준으로 한 주 분량의 숙제에 해당한다. 각 프로젝트는 완전히 독립적이라, 원하는 순서대로 수행할 수(또는 건너 뛸 수) 있다. Nand to Tetris를 전부 다 경험하려면 모든 프로젝트를 순서대로 수행해야 하지만, 여러 선택지 중 하나일 뿐이다.

한 학기 과정에서 이렇게 많은 내용을 다룰 수 있을까? 답은 '그렇다'이다. 백문이 불여일견으로, 실제 150개 이상의 대학에서 한 학기 과정으로 Nand to Tetris를 가르치고 있다. 학생 만족도는 매우 높으며, Nand to Tetris 온라인 강좌도 전체 온라인 강좌 중에 늘 상위권을 차지한다. 학생들이 이 방법론에 반응하는 이유는 바로 '집중'이다. 우리는 너무 쉬운 경우를 제외하고는 최적화는 신경 쓰지 않고 다른 더 구체적인 강의들이 담당하도록 미뤄 두었다. 또한 학생들이 입력의 오류를 걱정하지 않도록 했다. 따라서 예외 및 오류를 처리하는 코드를 작성할 필요가 없어지고, 소프트웨어 프로젝트에 훨씬 더 집중할 수 있게 된다. 물론 잘못된 입력을 처리하는 일은 매우 중요하지만, 프로그래밍 집중 과목이나 소프트웨어 설계 과목에서 오류 처리 기법을 배울 수 있을 거라 생각했다.

2판에서 달라진 점

Nand to Tetris는 전에도 두 주제로 구성되기는 했지만, 2판에서는 이 구성을 더 명확히 했다. 이제 이 책은 'I부: 하드웨어'와 'II부: 소프트웨어'로 별개의 독립적인 부분으로 나뉜다. 각 부는 6개 장, 6개 프로젝트로 구성되며, 각 부가 시작될 때마다 도입부에 해당하는 내용을 새로 작성했다. 각각의 부는 독립적이라는 점이 중요하다. 따라서 2판의 구조는 한 학기 과정 외에도 반 학기 과정으로도 가르칠 수 있다.

2판에서는 '도입' 장을 두 개 새로 추가한 것 외에도, 부록이 새로 네 개 추

가되었다. 이 부록들은 학습자들의 요청에 따라, 초판에서 여러 장에 흩어져 있던 다양한 기술적 주제들을 묶어 놓은 것이다. 또 다른 부록에서는 어떤 불 함수라도 Nand 연산으로 구현할 수 있음을 증명해서, 하드웨어 구성 프로젝트의 이론적인 기초를 다졌다. 그 외에도 새로운 절, 그림, 예제 들이 많이 추가되었다.

모든 장과 프로젝트 자료는, Nand to Tetris의 주제에 따라 추상화와 구현을 분리하는 데 중점을 두어 다시 쓰였다. 우리는 Nand to Tetris Q&A 포럼에 수년간 올라온 수천 개의 질문들과 관련된 예제와 설명들을 추가하는 데 특별히 공을 들였다.

감사의 글

이 책의 소프트웨어 도구들은 히브리 대학교Hebrew University와 IDC 헤르츨리야IDC Herzliya의 학생들이 개발했다. 두 명의 주요 소프트웨어 설계자는 야론 우크라이니츠Yaron Ukrainitz와 얀나이 곤차로프스키Yannai Gonczarowski이고, 개발자로는 이프타치 아밋Iftach Amit, 아사프 개드Assaf Gad, 갤 카첸들러Gal Katzhendler, 하다르 로젠시어Hadar Rosen-Sior와 니어 로젠Nir Rozen이 참여했다. 오렌 바라네스Oren Baranes, 오렌 코헨Oren Cohen, 조너선 그로스Jonanthan Gross, 골란 파라시Golan Parashi, 우리 자이라Uri Zeira는 도구의 다른 부분을 개발했다. 이 학생 개발자들과 함께 일하는 것은 큰 즐거움이었으며, 이들을 가르칠 수 있었던 것을 자랑스럽게 생각한다.

또한 이 책을 쓰기 전 강의에서 조교를 담당해 줬던 무아위야 아카시Muawyah Akash, 필립 헨드릭스Philip Hendrix, 에이탄 리프스히트스Eytan Lifshitz, 란 나복Ran Navok, 데이비드 라비노비츠David Rabinowitz에게 감사드린다. 탈 아히투브Tal Achituv, 용 바코스Yong Bakos, 탈리 구트먼Tali Gutman, 마이클 슈뢰더Michael Schröder는 강의 자료에 여러모로 큰 도움을 주었고, 아뤼에 슈날Aryeh Schnall, 토마스츠 론스키Tomasz Różański, 루돌프 아담코비츠Rudolf Adamkovič는 편집에 관해

세심한 조언을 해 주었다. 루돌프의 의견은 특히 큰 깨우침을 줬고 이에 대해 매우 감사하게 생각한다.

Nand to Tetris에는 전 세계의 많은 사람이 참여했기에 한 명씩 각각에게 모두 감사를 건네기는 어렵다. 하지만 예외가 한 명 있다. 콜로라도 출신의 소프트웨어 및 펌웨어 엔지니어인 마크 암브러스트Mark Armbrust는 Nand to Tetris 학습자들의 수호 천사였다. 마크는 전 세계 Q&A 포럼 관리를 자발적으로 맡아서 수많은 질문에 인내심을 가지고 정중한 답변을 남겨 주었다. 마크는 정답을 말하는 대신에 학습자들이 스스로 해 보고 답을 찾아가도록 유도했다. 그래서 마크는 전 세계 수많은 학습자에게 존경과 찬사를 받았다. 마크는 10년 이상 Nand to Tetris의 최전선에서 일하면서 2,607개의 글을 쓰고, 수십 개의 버그를 발견했으며, 수정 스크립트를 작성하고 내용을 수정했다. 마크는 이 모든 일을 일상적인 업무 시간 외에 수행하면서 Nand to Tetris 커뮤니티의 기둥이 되었으며, 커뮤니티는 그의 제2의 고향이 되었다. 그는 몇 달 동안 심장병으로 투병하다 2019년 3월에 사망했다. 입원하는 동안 마크는 Nand to Tetris 학생들에게 매일 수백 통의 편지를 받았다. 전 세계 청년들은 마크의 무한한 헌신에 감사하고, 자신들이 마크에게 받은 영향을 서로 나눴다.

최근 몇 년 동안 컴퓨터 공학 교육은 개인의 성장과 경제적 지위 상승의 강력한 원동력이 되었다. 돌아보면 일찍부터 모든 교육자료를 자유롭게 사용할 수 있도록 오픈 소스로 공개하기로 결정한 것이 다행이라 생각한다. 한마디로 Nand to Tetris 과정은 아무런 제한 없이 누구라도 배우거나 가르칠 수 있다. 비영리적이기만 하다면 웹사이트에 와서 원하는 자료를 마음대로 가져가도 된다. 이 정책으로 Nand to Tetris는 고품질의 컴퓨터 과학 교육을 자유롭고 공평하게 보급하는 간편한 수단이 되었다. 그 결과 사람들의 끊임없는 참여를 바탕으로 Nand to Tetris는 거대한 교육 생태계가 되었다. 이를 실현할 수 있도록 도와 주신 전 세계의 많은 분께 감사드린다.

하드웨어
Hardware

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

마르셀 프루스트(Marcel Proust, 1871~1922)

이 책은 발견하는 여행이다. 이제 곧 독자들은 다음 세 가지를 배울 참이다. 컴퓨터가 어떻게 작동하는지, 복잡한 문제들을 어떻게 다루기 좋은 개별 요소로 쪼개는지, 그리고 대규모 하드웨어 및 소프트웨어 시스템을 어떻게 개발하는지에 대해서. 이 과정은 실제 작동하는 완전한 컴퓨터 시스템을 밑바닥부터 직접 만들어 보는 여정이 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 도입 부분에서는 앞으로 우리에게 놓인 발견, 진리, 경험들을 스케치해 보려 한다.

안녕, 밑바닥 세상아

프로그래밍 과목을 들어 본 적이 있다면, 아마도 앞부분에서 다음과 같은 프로그램을 접한 적이 있을 것이다. 본 적이 없더라도 이 프로그램이 `Hello World`를 화면에 출력하고 멈추는 프로그램이라고 추측할 수 있을 것이다. 이

프로그램은 자바와 유사한, 간단한 고수준 언어high-level language인 잭Jack으로 작성되었다.

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

Hello World 같은 프로그램은 겉으로만 간단해 보일 뿐이다. 이런 프로그램이 컴퓨터에서 실제로 작동하는 데 무엇이 필요한지 생각해 본 적이 있는가? 한번 그 속을 들여다 보자. 우선 이 프로그램은 그저 텍스트 파일에 저장된 일반 문자열일뿐임에 주목하자. 이 추상화는 기계어로 작성된 명령어만 이해하는 컴퓨터에게는 완전히 뜻 모를 수수께끼일뿐이다. 그러므로 컴퓨터가 이해할 수 있도록 먼저 고수준 코드의 문자열을 분석해서 프로그램이 수행하려는 작업의 의미를 찾아낸 후에, 대상 컴퓨터의 기계어로 그 의미를 다시 쓴 저수준 코드를 생성해야 한다. 이 정교한 번역 과정을 컴파일compilation이라고 하며, 실행 가능한 기계어 명령어들이 그 결과로 나오게 된다.

물론 기계어도 미리 약속된 2진 코드로 구성된 추상화 개념이다. 따라서 이 추상화를 명확히 하려면 하드웨어 아키텍처hardware architecture를 반드시 구현해야 한다. 이 아키텍처는 레지스터, 메모리 장치, ALUArithmetic Logic Unit 같은 칩들로 구성된다. 그리고 이 하드웨어들은 모두 저수준의 기초 논리 게이트elementary logic gate로 만들어진다. 또한 기초 논리 게이트들은 Nand나 Nor 같은 기본 게이트들로 구성할 수 있다. 이 기본 게이트들은 계층 구조의 가장 하단에 위치하지만 그 자체도 보통 트랜지스터로 구현된 몇 개의 스위치 장치switching device로 이루어진다. 그리고 트랜지스터들은? 잠깐, 여기서 더 깊이 들어가진 않겠다. 여기부터는 컴퓨터 과학이 아니라 물리학의 영역이기 때문이다.

독자들은 이렇게 생각할 수 있다. "내 컴퓨터에서 프로그램을 컴파일하고

실행하는 건 훨씬 쉬운데. 그냥 이 아이콘을 클릭하거나 저 명령어를 입력하기만 하면 되잖아!" 정말로, 현대 컴퓨터 시스템은 꼭대기만 보이는 거대한 빙산 같다. 이런 지식은 컴퓨터 시스템에 대한 개략적이고 피상적인 이해일 뿐이다. 하지만 만약 이 물밑 세계를 탐험하고 싶다는 생각이 들었다면, 당신은 행운아다! 그 아래 세상은 컴퓨터 과학에서 가장 아름다운 부분들로 이루어진 매혹적인 세상이다. 이 물밑 세계에 대한 이해도가, 복잡한 하드웨어 및 소프트웨어 기술을 창조할 줄 아는 수준 높은 개발자와 그저 단순한 프로그래머를 가른다. 이런 기술들이 어떻게 작동하는지 뼛속 깊이 이해하는 가장 좋은 방법은 밑바닥부터 온전한 컴퓨터 시스템을 만들어 보는 것이다.

Nand에서 테트리스까지

컴퓨터 시스템을 밑바닥부터 만든다고 하면, 구체적으로 어떤 컴퓨터를 만들어야 할까? 나중에 알게 되겠지만, PC, 스마트폰, 서버 등 모든 범용 컴퓨터는 Nand to Tetris 기기다. 첫째로 모든 컴퓨터는 기본적으로 기초 논리 게이트를 기본으로 하며, Nand는 산업에서 가장 널리 사용되는 게이트다(Nand 게이트가 정확히 무엇인지는 1장에서 설명할 예정이다). 둘째로 모든 범용 컴퓨터는 테트리스 게임 외에도 여러분의 상상력을 자극하는 어떤 프로그램도 실행할 수 있도록 프로그래밍할 수 있다. 따라서 Nand나 테트리스에 특별한 건 없다. 이 책을 앞으로 독자 여러분이 경험하게 될 마법 같은 여행으로 바꿔 줄 단어는 바로 Nand to Tetris의 'to'라는 단어다. 그 여행은 단순한 스위치 장치 부품 더미에서, 텍스트, 그래픽, 애니메이션, 음악, 비디오, 분석, 시뮬레이션, 인공지능 등, 범용 컴퓨터에서 우리가 기대할 수 있는 모든 기능으로 우리를 안내할 것이다. 따라서 세상 모든 컴퓨터 시스템이 따르는 개념이나 기술에 뿌리를 두고 있는 이상, 우리가 어떤 하드웨어 플랫폼이나 소프트웨어 계층을 만들지는 그다지 중요하지 않다.

그림 1.1은 Nand to Tetris 길잡이 지도의 주요 이정표를 보여 준다. 그림의 맨 아래 계층에서 시작해서 모든 범용 컴퓨터는 ALUArithmetic Logic Unit(산술

논리 장치)와 RAMRandom Access Memory(임의 접근 메모리)를 포함하는 아키텍처를 갖는다. 모든 ALU와 RAM 장치는 기초 논리 게이트로 구성된다. 그리고 놀라우면서도 다행스럽게, 모든 논리 게이트는 Nand 게이트만으로 만들 수 있다. 이제 소프트웨어 계층을 살펴보면, 모든 고수준 언어는 고수준 코드를 기계어 수준의 명령어로 바꾸는 번역기들(컴파일러/인터프리터, 가상 머신, 어셈블러)에 의존한다. 어떤 고수준 언어는 컴파일 방식보단 인터프리터 방식을 쓰고, 또 어떤 언어는 가상 머신을 사용하지 않거나 하지만, 큰 그림은 본질적으로 같다. 다음 그림은 모든 컴퓨터는 본질적으로 동등하다는, 처치-튜링 추측Church-Turing conjecture이라는 기초 컴퓨터 과학 원리를 나타낸 것이다.

이 그림은 이 책의 접근법이 일반적임을 강조하는 그림이다. 이 책에서 접하게 될 문제, 통찰, 팁, 기법, 기술 및 용어들은 실제 하드웨어 및 소프트웨어 엔지니어들이 접하는 것과 완전히 같다. 그런 점에서 Nand to Tetris는 일

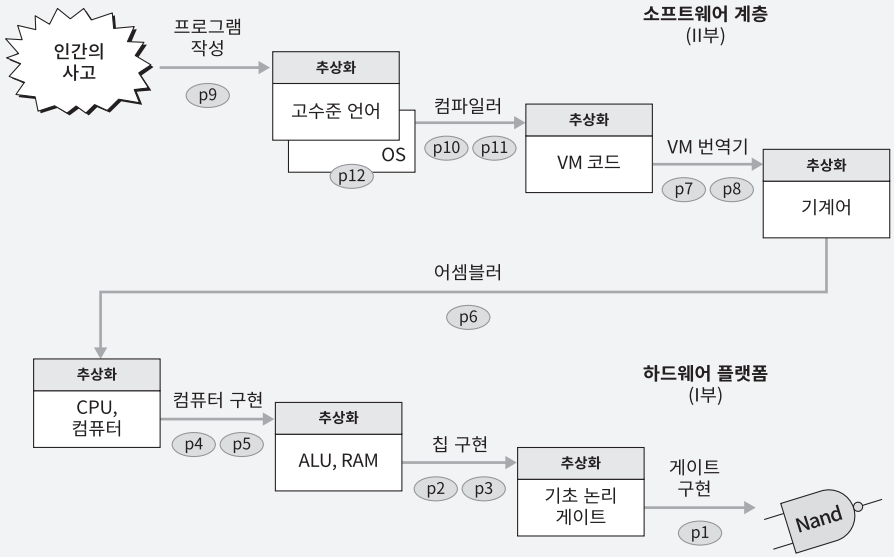

그림 I.1 하드웨어 플랫폼과 소프트웨어 계층으로 구성된 일반적인 컴퓨터 시스템의 주요 모듈들. 각 모듈에는 추상화 뷰(abstract view, 또는 모듈의 인터페이스)와 구현이 하나씩 있다. 오른쪽으로 이어지는 화살표는 각각의 모듈이 그 아래 단계의 추상화 구성 블록을 사용해서 구현되었음을 나타낸다. 각각의 원은 Nand to Tetris의 각 장과 프로젝트들을 뜻하며, 모두 합쳐서 12개의 프로젝트가 있다.

종의 입문 단계다. 이 여정을 마치면 핵심 컴퓨터 전문가가 되기 위한 기반을 훌륭히 닦을 수 있을 것이다.

그렇다면 Nand to Tetris에서 구체적으로 어떤 하드웨어 플랫폼과 고수준 언어를 만들어야 할까? 산업적으로 널리 쓰이는 컴퓨터 모델을 만들고 유명한 고수준 언어에 대한 컴파일러를 작성하는 방법도 있었을 것이다. 하지만 세 가지 이유 때문에 그 길을 선택하지 않았다. 첫째, 컴퓨터 모델은 왔다가 사라지고, 한때 인기 있던 프로그래밍 언어도 새로운 언어에 자리를 내준다. 따라서 특정 하드웨어/소프트웨어 구성에만 매이고 싶지 않았다. 둘째, 실제 사용되는 컴퓨터와 언어들은 교육적 가치는 별로 없지만, 구현하는 데 한세월이 걸리는 세부 기능이 너무 많다. 마지막으로, 제어하기 쉽고, 이해하기 쉽고, 확장하기 쉬운 하드웨어와 소프트웨어 플랫폼이 필요했다. 이 세 가지 고려사항에 따라 I부의 핵Hack 컴퓨터 플랫폼과, II부의 잭Jack 고수준 언어를 만들게 되었다.

일반적으로 컴퓨터 시스템을 설명할 때는, 높은 수준의 추상화가 어떻게 더 단순한 추상화로 구현되는지 하향식top-down으로 설명한다. 예를 들어 어떤 컴퓨터 아키텍처에서 실행되는 2진 명령어가, 어떻게 더 작은 코드들로 쪼개져서 아키텍처의 전선을 통해 전달되고 최종적으로 어떻게 저수준 ALU와 RAM 칩을 조작하게 되는지 설명할 수 있을 것이다. 또는 상향식bottom-up으로, ALU와 RAM 칩이 어떻게 잘 모여서 그 작은 코드들을 실행하는 2진 기계 명령어군을 구성하는지 설명하는 방법도 있을 것이다. 하향식과 상향식 접근법은 둘 다 가치가 있으며, 만들려는 시스템에 대한 서로 다른 관점을 보여 준다.

그림 I.1에서 화살표 방향은 하향식 접근법을 나타내고 있다. 모듈들 사이에는 고수준 모듈에서 저수준 모듈로 향하는 방향을 가리키는 화살표가 있다. 이 화살표의 의미는 명확하다. 고수준 모듈이 그 아래 수준의 추상화 구성 블록을 사용해서 구현된다는 뜻이다. 예를 들어 고수준 프로그램은 각 고수준 명령문을 추상화 VM 명령 집합으로 번역하는 식으로 구현된다. 그리고

각 VM 명령은 추상화 기계어 명령어로 번역되는 식이다. 추상화와 구현의 구분은 시스템 설계에서 핵심적인 역할을 하며, 이제 곧 논의할 것이다.

추상화와 구현

독자들은 완전한 컴퓨터 시스템을 기초 논리 게이트만 이용해 밑바닥부터 구성하는 것이 어떻게 가능한지 궁금할지도 모르겠다. 엄청난 작업이지 않은가! 그래서 우리는 이 복잡한 프로젝트를 모듈로 쪼개서, 각 모듈별로 한 개 장씩 따로 할애해서 개별 프로젝트로 만들어 나갈 예정이다. 그러면 어떻게 이 모듈들을 따로 설명하고 구성할지가 궁금할 것이다. 당연히 이 모듈들은 모두 서로 연관되어 있다! 책 전반에 걸쳐 설명하겠지만, 좋은 모듈 디자인이란 시스템의 나머지 부분을 신경 쓰지 않아도 되도록 모듈을 독립적으로 만드는 것이다. 사실 시스템이 잘 설계되었다면 이 모듈들을 어떤 순서로 만들든, 팀으로 일한다면 동시에 만들어도 상관없다.

복잡한 시스템을 다루기 좋은 모듈로 '분할과 정복'하는 인지 능력은, 각 모듈의 추상화abstraction와 구현implementation을 식별하는 사고방식에 의해 강화된다. 컴퓨터 과학에서 이 용어는 구체적인 의미로 쓰인다. 즉, 추상화는 모듈이 '무엇을 하는지'를, 구현은 '어떻게 하는지'를 가리킨다. 이 구분 개념을 기본으로, 이제 시스템 공학에서 가장 중요한 규칙이 나온다. 바로 어떤 모듈이든 구성 블록으로 사용할 때는 모듈의 추상화에만 집중하고, 상세 구현은 완전히 무시해야 한다는 점이다.

예를 들어, '컴퓨터 아키텍처' 수준부터 시작하는 그림 I.1의 맨 아래 계층에 주목해 보자. 그림에서 볼 수 있듯, 이 아키텍처는 임의 접근 메모리Random Access Memory, RAM를 포함한 그 아래 수준의 구성 블록들 몇 가지를 활용하여 구현된다. RAM은 놀라운 장치다. RAM은 수십억 개의 레지스터로 구성되어 있지만, 그중 어떤 레지스터라도 거의 즉각적으로 직접 접근할 수 있다. 그림 I.1은 컴퓨터 설계자가 이 직접 접근 장치가 실제로 어떻게 구현되는지는 전

혀 신경 쓰지 않고 추상적으로만 사용할 것임을 말해 준다. 직접 접근 RAM 마법을 '어떻게' 구현했는지 나타내는 모든 기술적 탁월함이나 드라마 같은 정보는, RAM을 실질적으로 활용하는 맥락과는 무관하므로 완전히 무시해야 한다.

그림 I.1에서 한 단계 아래로 내려가면, 이제 RAM 칩을 구현해야 하는 입장이 되었다. 어떻게 해야 할까? 오른쪽 화살표를 따라가면 그보다 더 아래 수준의 기초 논리 게이트 및 칩들로 RAM이 구현되었음을 알 수 있다. 특히 RAM 저장 및 직접 접근 기능은 각각 레지스터와 멀티플렉서를 이용해서 구현된다. 그리고 다시 한 번 추상화-구현 원리가 적용된다. 우리는 이 칩들을 추상화 구성 블록으로 사용하고, 인터페이스에만 중점을 두고 구현에 대해서는 전혀 신경 쓰지 않을 것이다. 이런 식으로 Nand 수준까지 내려간다.

요약하면, 저수준 하드웨어나 소프트웨어 모듈을 사용해서 구현할 때는 이 모듈들을 블랙박스 기성품으로 생각해야 한다. 구현에 필요한 건 오로지 모듈이 '무엇'을 하는지 설명하는 인터페이스 문서뿐이다. 모듈 인터페이스가 말하는 것을 '어떻게' 수행하는지에 대해서는 신경 쓰지 말아야 한다. 이 추상화-구현 패러다임을 통해 개발자는 매우 큰 시스템을 잘 정의된 모듈로 쪼개서 오류 처리와 수정 및 구현 작업을 수행하기 쉽게 만들 수 있으며, 시스템의 복잡함을 줄이고 온전한 상태를 유지할 수 있게 된다. 이 패러다임은 하드웨어 및 소프트웨어 구축 프로젝트에서 가장 중요한 설계 원리다.

말할 필요도 없이 이 이야기의 모든 것은 복잡한 모듈식 설계 기술에 달려 있다. 문제를 잘 정의된 모듈로 구분하고, 그 사이에 뚜렷한 인터페이스를 만들고, 각각을 독립적으로 구현하기 좋은 크기로 나누고, 단위 테스트 프로그램으로 테스트할 수 있게 만드는 인간의 기술 말이다. 실제로 모듈식 설계는 응용 컴퓨터 과학에서 밥 먹듯 하는 가장 기본적인 방식이다. 시스템 설계자들은 모두 일상적으로 추상화(때로는 모듈 또는 인터페이스라고도 함)를 정의한 다음 구현하거나, 다른 사람에게 구현을 맡긴다. 추상화는 보통 층층히 쌓여서 점점 더 고수준의 기능을 수행하도록 설계된다. 시스템 설계자가 모듈

들을 잘 설계한다면 구현은 물 흐르듯 진행될 것이다. 만약 설계가 엉성하다면 구현도 실패할 것이다.

모듈식 설계는 다른 잘 설계된 추상화들을 보거나 구현하면서 체득되는 기술이다. 그리고 그것이 Nand to Tetris에서 여러분이 경험하게 될 내용이다. 독자들은 수백 가지 하드웨어 및 소프트웨어 추상화 기능의 우아함을 감상하는 법을 배우게 될 것이다. 그리고 이 책은 추상화들을 한 번에 한 단계씩 구현하면서 점점 더 큰 기능들을 구현하는 길로 안내할 것이다. 이 여행을 한 장씩 계속하면서, 여러분의 노력으로 컴퓨터 시스템이 점차 모양을 갖춰 나가는 모습을 돌아보는 일은 신나는 경험이 될 것이다.

방법론

Nand에서 테트리스로 가는 여정은 하드웨어 플랫폼과 소프트웨어 계층을 구현하는 일로 이뤄진다. 하드웨어 플랫폼은 I부에서 만드는 약 30개의 논리 게이트 및 칩들을 기반으로 한다. 최상위 컴퓨터 아키텍처를 포함한 이 게이트와 칩들은 모두 하드웨어 기술 언어Hardware Description Language, HDL로 구현된다. HDL은 부록 2에 문서화되어 있으며, 1시간 정도면 배울 수 있다. 작성한 HDL 프로그램이 올바르게 동작하는지는 독자의 PC에서 실행되는 소프트웨어 기반 하드웨어 시뮬레이터로 테스트하게 된다. 이 방식은 하드웨어 엔지니어들이 실제로 일하는 방식과 똑같다. 엔지니어들도 소프트웨어 기반 시뮬레이터로 칩을 구현하고 테스트한다. 그리고 엔지니어들은 칩 시뮬레이션 성능이 잘 나오면 그 사양(HDL 프로그램)을 칩 제조 회사에 보낸다. 거기서 HDL 프로그램을 최적화한 후에, 칩을 제조하는 로봇 팔에 입력되어 실리콘으로 된 하드웨어가 만들어진다.

Nand에서 테트리스로 가는 여정 중 II부에서는, 어셈블러, 가상 머신, 컴파일로 된 소프트웨어 계층을 만들게 된다. 이 프로그램들은 고수준 언어 중 어느 것이라도 택해서 구현할 수 있다. 추가로 잭 언어를 이용해 기본적인 운

영체제도 만들 것이다.

이 야심찬 프로젝트들을 어떻게 책 한 권 분량의 강의에서 개발할 수 있을지 궁금할지도 모르겠다. 사실 모듈식 설계 외에 우리의 비법은 설계의 불확실성을 최소한으로 줄이는 것이다. 각 프로젝트마다 자세한 API, 뼈대가 되는 프로그램, 테스트 스크립트, 단계별 구현 지침 등으로 발판을 세세하게 깔아 놓았다.

프로젝트 1~12를 완성하는 데 필요한 소프트웨어 도구들은 모두 Nand to Tetris 소프트웨어 모음에 들어 있으며, *www.nand2tetris.org*에서 무료로 다운로드할 수 있다. 이 모음에는 하드웨어 시뮬레이터, CPU 에뮬레이터, VM 에뮬레이터, 실행 가능한 하드웨어 칩, 어셈블러, 컴파일러, OS가 들어 있다. 소프트웨어 모음을 PC에 다운로드하고 나면 이 모든 도구를 손쉽게 사용할 수 있을 것이다.

앞으로 가야 할 길

Nand에서 테트리스로 가는 여정에는 12개의 하드웨어 및 소프트웨어 구현 프로젝트가 있다. 이 프로젝트들의 개발 방향이나 책의 목차는 상향식 방식을 따른다. 우리는 기초 논리 게이트부터 시작해서 객체 기반의 고수준 프로그래밍 언어 개발로 향할 것이다. 동시에 프로젝트 내의 개발 방식은 하향식이다. 특히 하드웨어나 소프트웨어 모듈을 처음 소개할 때는 항상 그 모듈이 무엇을 하도록 설계되었는지, 그리고 왜 필요한지에 대해 추상적인 설명으로 시작할 것이다. 모듈의 추상화 개념(그 자체로도 깊은 세계)을 이해하고 나면, 그보다 아래 수준의 추상화 구성 블록들을 이용해서 해당 모듈을 구현하게 된다.

그래서 이 기나긴 여정에서 I부의 원대한 계획은 다음과 같다. 1장에서는 Nand 논리 게이트 하나에서 시작해서, And, Or, Xor 등과 같이 일반적으로 사용되는 기초적인 논리 게이트들을 만들 것이다. 2장과 3장에서는 이 구성

블록들로 산술 논리 장치Arithmetic Logic Unit, ALU와 메모리 장치를 각각 만들 것이다. 4장에서는 하드웨어 구현은 잠시 멈추고 기호 및 2진 형식의 저수준 기계어를 소개한다. 5장에서는 앞에서 만든 ALU와 메모리 장치로 중앙 처리 장치Central Processing Unit, CPU와 임의 접근 메모리RAM를 구현할 것이다. 이 장치들은 하드웨어 플랫폼으로 통합되어서, 4장에서 도입한 기계어로 작성된 프로그램을 실행할 수 있게 된다. 5장에서는 어셈블러를 설명하고 구현할 예정인데, 어셈블러는 기호 방식 기계어로 작성된 저수준 프로그램을 실행 가능한 2진 코드로 번역하는 프로그램이다. 이 작업으로 하드웨어 플랫폼 구성이 끝난다. 이 플랫폼은 II부의 출발점이 되어서, 앞으로 가상 머신, 컴파일러, 운영체제로 이뤄진 최신 소프트웨어 계층을 갖춘 플랫폼으로 확장해 나갈 것이다.

독자들이 대략적이나마 앞으로 나올 내용들을 이해했길 바라며, 새로운 지식을 향한 이 긴 여행을 기꺼이 시작했으면 한다. 자, 이제 준비되었다면 카운트다운을 해보자. 1, 0, 땅!