컴퓨터

계산하는 사람의 작업대에서, 세계를 연결하는 기반시설까지

기준일 2026년 9월 15일

컴퓨터는 표현된 정보를 정해진 절차에 따라 처리하는 장치다. 오늘날 널리 쓰이는 디지털 컴퓨터는 숫자뿐 아니라 글·소리·영상·명령을 비트로 나타내고, 저장된 프로그램을 실행해 그 표현을 바꾼다. 이 책은 개인용 제품을 넘어 계산 도구의 전사, 전자식 컴퓨터의 성립, 하드웨어와 소프트웨어의 원리, 인터넷과 세계 산업, 사람의 일과 환경에 미친 변화까지 다룬다. 중심 질문은 하나다. **작은 물리 장치들이 어떻게 같은 절차를 정확히 반복하고, 그 반복이 어떻게 사회 전체의 일을 바꾸었는가?** [1] [2]

주

1 B. Jack Copeland, 「The Modern History of Computing」, https://plato.stanford.edu/entries/computing-history/ . 위치: 도입부, Babbage, Analog Computers, Electromechanical versus Electronic Computation, The Manchester Machine, ENIAC and EDVAC.

2 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

1장. 컴퓨터의 경계는 어디에 있는가

계산, 표현, 절차

'계산'은 산술보다 넓은 뜻으로 쓰인다. 두 수의 합을 구하는 일과 사전에서 이름을 찾는 일은 모두 입력을 표현하고 규칙을 적용해 결과를 얻는다. 사진을 밝게 만드는 프로그램은 픽셀의 수치를 바꾸고, 문서 검색 프로그램은 글자의 배열을 비교한다. 컴퓨터가 숫자를 다룬다는 사실은 숫자로만 된 대상을 다룬다는 뜻이 아니다. 현실의 무엇을 어떤 부호로 바꿀지, 그 부호에 어떤 연산을 적용할지를 정하면 계산의 대상이 넓어진다. [1] [2]

프로그램은 그 절차를 장치가 실행할 수 있게 적은 것이다. 입력을 읽고, 상태를 기억하고, 조건을 비교하고, 다음에 할 일을 선택하는 기능이 연결되면 사람이 매번 버튼을 누르지 않아도 여러 단계의 일을 수행할 수 있다. 계산기는 덧셈을 잘하지만, 보통 사용자가 연산 순서를 정한다. 프로그램으로 반복과 분기를 지정하는 컴퓨터에서는 순서를 정하는 일의 상당 부분까지 기계에 맡긴다. 경계에 놓인 프로그램식 계산기도 있으므로, 제품의 이름보다는 무엇을 실행하고 어떻게 지시하는지를 보는 편이 정확하다. [3]

서로 다른 분류 기준

기계식·전기기계식·전자식은 **무엇으로 구현했는가**를 가른다. 기계식 계산기는 톱니바퀴의 움직임을 이용하고, 전기기계식 장치는 전자석으로 움직이는 릴레이 접점을 이용하며, 전자식 회로는 진공관이나 반도체 소자로 신호를 제어한다. 아날로그·디지털은 **양을 어떻게 표현하는가**를 가른다. 이 두 기준은 독립적이다. 배비지의 톱니바퀴는 열 개의 자리 중 하나로 숫자를 나타냈으므로 기계식이면서 디지털이었다. [4]

이 책에서 아날로그 계산은 전압이나 회전각처럼 연속적으로 변하는 물리량을 문제의 양에 대응시켜 계산하는 방식을 뜻한다. 예를 들어 전압을 속도에 대응시키고 적분 회로의 출력을 위치로 해석할 수 있다. 디지털 계산은 구분되는 기호를 배열해 수를 표현하고 그 배열을 규칙에 따라 바꾼다. 아날로그 장치는 문제를 물리 관계로 직접 옮기는 데 강점이 있고, 디지털 장치는 프로그램 변경·복제·정밀도 확장에 유리하다. 디지털이라는 말이 무한 정확도를 뜻하지는 않는다. [4] [5]

범용·전용은 **어떤 일을 하도록 쓰이는가**의 구분이다. 범용 CPU를 넣은 세탁기 제어 장치가 실제로는 정해진 세탁 동작만 수행할 수 있다. 반대로 아날로그 미분해석기는 연결을 바꿔 여러 미분방정식을 풀 수 있었다. '디지털=범용, 아날로그=전용'이라고 짝지으면 이런 경우를 설명할 수 없다. 저장 프로그램 여부는 또 다른 기준이다. 프로그램을 메모리에 명령의 형태로 넣는지, 천공테이프나 배선으로 순서를 지정하는지를 묻는다. [4] [3]

PC·서버·슈퍼컴퓨터·임베디드 컴퓨터는 주로 사용 방식과 설계 목표를 가리킨다. 개인용 컴퓨터는 한 사람의 상호작용을, 서버는 다른 프로그램이나 이용자에게 서비스를 제공하는 역할을, 슈퍼컴퓨터는 큰 계산을 가능한 한 빨리 끝내는 것을 중시한다. 임베디드 컴퓨터는 더 큰 기계 속에서 특정 기능을 맡는다. 이들은 일렬로 세운 성능 등급이 아니다. 작은 컴퓨터도 서버가 될 수 있고, 슈퍼컴퓨터는 여러 서버와 가속기를 연결해 만들 수 있다. [6] [7]

주

1 Steve Ward, 「The Digital Abstraction」, https://computationstructures.org/notes/digitalabstraction/notes.html . 위치: §5.1–5.4, 이산 표현·조합 장치·잡음 여유. 이진수 조합과 덧셈은 정의에서 도출한 설명 예시.

2 Unicode Consortium, 「What is Unicode?」, https://www.unicode.org/standard/WhatIsUnicode.html . 위치: Characters Before Unicode, Unicode Characters.

3 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

4 B. Jack Copeland, 「The Modern History of Computing」, https://plato.stanford.edu/entries/computing-history/ . 위치: 도입부, Babbage, Analog Computers, Electromechanical versus Electronic Computation, The Manchester Machine, ENIAC and EDVAC.

5 David Goldberg, 「What Every Computer Scientist Should Know About Floating-Point Arithmetic」, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html . 위치: Rounding Error, Floating-point Formats, Cancellation. 십진수 0.1의 이진 표현과 반올림·상쇄의 설명.

6 Computer History Museum, 「Computers」, https://www.computerhistory.org/timeline/computers/ . 위치: 1937 Stibitz, 1941 Z3, 1944 Colossus·Mark I, 1946 ENIAC, 1948 Baby, 1949 EDSAC, 1953 트랜지스터 컴퓨터, 1960 PDP-1, 1966 HP 2116A, 1971 4004, 1975–1977 개인용 컴퓨터, 1981 IBM PC, 1984 Macintosh, 1987 ARM, 1991 PowerBook 및 2000년대 모바일·내장형 항목.

7 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「Distributed Systems」, https://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf . 위치: 장 도입, §48.1–48.5, 실패·신뢰성·재전송·RPC. 송금은 재시도와 중복 처리 문제의 설명 예시.

2장. 계산은 먼저 사람의 일이었다

표를 만드는 노동과 오류의 비용

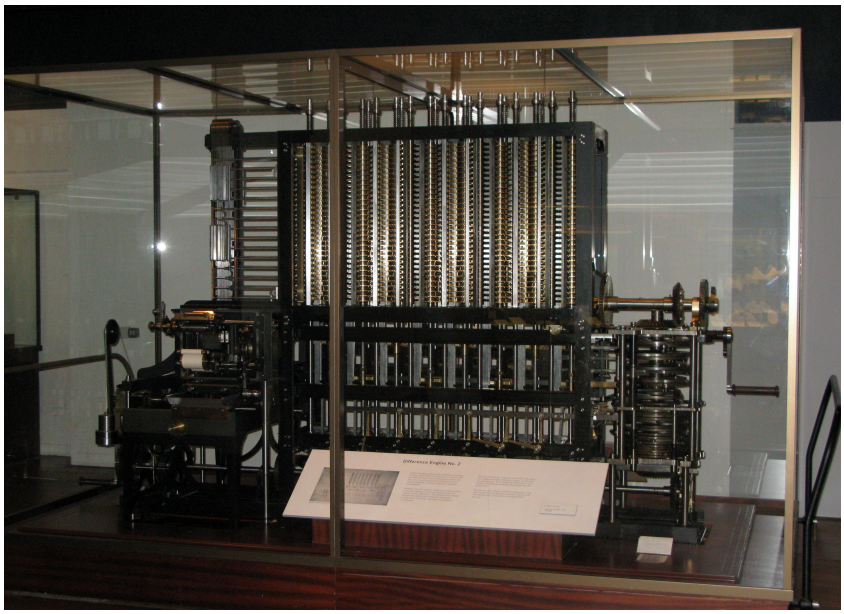
전자식 기계가 등장하기 전 영어의 *computer*는 계산하는 사람을 가리켰다. 상업·정부·연구기관은 정해진 계산법을 반복하는 직원을 고용했다. 항해와 천문학, 측량, 보험 업무에 쓰는 수학표는 계산의 결과를 인쇄해 다시 사용하는 장치였다. 사용자는 표를 찾아 복잡한 연산을 줄일 수 있었지만, 표를 만드는 사람은 많은 값을 계산하고 다른 사람이 그 결과를 옮겨 적고 조판해야 했다. 한 단계의 오류가 다음 이용자에게 복제될 수 있었다. [1] [2]

주판과 계산자는 이런 노동의 일부를 돕는다. 주판은 수의 상태를 눈앞에 보존하고 자리올림을 손으로 수행하게 한다. 계산자는 로그의 성질을 길이에 옮겨 곱셈을 눈금의 덧셈으로 바꾼다. 도구가 물리 관계를 이용하더라도 문제를 나누고 연산 순서를 결정하는 역할은 사람에게 남을 수 있다. 계산의 역사를 이해하려면 장치가 대신한 일이 기억인지, 산술인지, 순서 제어인지 구별해야 한다. [1]

배비지가 19세기 초 차분기관에 기대한 것은 수학 표 생산 과정의 자동화였다. 차분법을 쓰면 일정한 간격의 다항식 값을 덧셈의 반복으로 구할 수 있다. 예를 들어 제곱수 1, 4, 9, 16의 차이는 3, 5, 7이고 그 차이의 차이는 언제나 2다. 마지막 제곱수에 다음 홀수를 더하고 홀수에는 2를 더하는 절차를 기계화하면 제곱표를 이어 만들 수 있다. 계산한 수를 곧바로 인쇄하는 기능은 답을 다시 옮겨 적는 단계의 오류까지 줄이려는 설계였다. [2]

그러나 설계가 곧 가동하는 기계는 아니었다. 배비지는 차분기관의 일부를 만들었지만 자신의 대형 설계를 완성하지 못했다. 분석기관은 더 야심찼다. 숫자를 보관하는 저장부와 계산하는 '밀', 천공카드에 의한 제어, 조건에 따른 분기를 갖추려 했다. 러브레이스는 이런 기계가 숫자 사이의 관계만이 아니라 기호로 표현할 수 있는 다른 관계에도 쓰일 가능성을 논했다. 기계 제작, 프로그램의 개념, 비수치 응용에 대한 통찰은 여기서 서로 만난다. [1] [2]

런던 과학박물관이 배비지의 차분기관 2호 설계를 바탕으로 만든 후대의 기계는 계산부를 1991년에, 인쇄부를 2002년에 완성했다. 이 실물은 설계가 작동할 수 있음을 보여 주지만, 19세기에 그 완성품이 사용됐다는 증거는 아니다. 설계도·당시의 부품·후대 재현품을 구분해야 하는 이유다. [2]



런던 과학박물관이 배비지 설계로 후대에 제작한 차분기관 2호. 계산부 1991년·인쇄부 2002년 완성. 톱니바퀴와 덧셈의 반복으로 수학 표를 계산하는 기계식 디지털 장치이며 배비지 생전의 완성 원품은 아니다. 사진 Geni / Wikimedia Commons, CC BY-SA 4.0.

Geni (Wikimedia Commons User:geni) · CC BY-SA 4.0 International (one of the offered licenses selected) — https://creativecommons.org/licenses/by-sa/4.0/ · https://commons.wikimedia.org/wiki/File:Babbage_Difference_Engine.jpg · Science Museum, London · 후대 제작 재현품. 배비지 생전 완성 원품이 아님. Science Museum이 1847–1849년 차분기관 2호 설계에 따라 제작, 계산부 1991년·인쇄부 2002년 완성.

인구와 거래를 세는 기계

수학표와 별개로 행정·상업은 많은 사람과 거래의 기록을 분류하고 합산해야 했다. 허먼 홀러리스의 천공카드 시스템은 1890년 미국 인구조사에 쓰였다. 직원이 조사 결과를 카드의 구멍 위치로 옮기면, 판독기의 금속 핀이 구멍을 통과해 전기 회로를 닫고 해당 계수기를 움직였다. 분류기는 카드의 속성에 따라 넣을 칸을 알려 주었다. 계산 이전에 자료를 일정한 형식으로 부호화하고, 같은 기록을 다른 기준으로 다시 집계할 수 있게 한 것이다. [3] [4]

이 방식은 범용 전자컴퓨터가 아니어도 행정의 규모를 키울 수 있음을 보여 준다. IBM의 기업사에 따르면 천공카드는 사회보장 행정과 공공요금 청구, 도서 관리 등에 쓰였다. 기록 양식과 판독기, 분류기, 집계기, 직원의 작업 절차가 하나의 체계를 이루었다. 뒤의 기업 전산은 이런 기록과 업무 흐름을 이어받으면서 자기테이프와 전자식 처리로 매체와 속도를 바꾸었다. 컴퓨터의 수요에는 과학과 군사 계산뿐 아니라 이미 축적된 사무 자동화의 시장도 있었다. [3]

아날로그라는 다른 경로

모든 문제를 자릿수 계산으로 풀 필요는 없었다. 조석 예측기는 주기적인 성분들을 기계적으로 합성해 물높이의 변화를 나타냈다. 미분해석기는 적분을 수행하는 회전 장치들을 연결해 미분방정식의 해를 만들었다. 바퀴와 원판 사이의 접촉 위치가 바뀌면 출력 회전량이 달라지는 관계를 수학적 적분에 대응시켰다. 1931년 MIT의 배니버 부시가 완성한 미분해석기는 이런 아날로그 계산의 중요한 구현이었다. [1]

장점은 여러 부분이 동시에 움직이면서 해를 얻는다는 데 있었다. 반면 새 문제를 장치의 연결 관계로 바꾸고 기계를 설정하는 작업에는 숙련이 필요했다. 정밀도를 높이려면 가공·측정·회로 안정성의 조건도 강화해야 했다. 디지털 계산에서는 더 많은 자릿수와 연산을 써서 정밀도를 높일 수 있지만, 아날로그 계산은 물리량 자체의 작은 차이를 구분해야 한다. 전자식 디지털 컴퓨터가 성장하면서 이 차이가 재사용성과 경제성에 큰 영향을 주었다. [1]

릴레이에서 진공관으로

전화 교환에 쓰이던 릴레이는 전기 신호로 스위치를 움직일 수 있게 했다. 조지 스티비츠의 계산기와 콘라트 추제의 기계는 이런 부품으로 연산과 제어를 구성했다. 추제의 Z3는 1941년에 작동한 이진 전기기계식 컴퓨터였고, 부동소수점 연산을 수행했다. 릴레이를 이용한 자동화는 사람이 매 단계 계산하는 것보다 절차를 길게 연결할 수 있게 했지만, 접점이 움직이는 시간과 마모가 속도와 신뢰성에 제약을 주었다. [5] [1]

진공관은 기계 접점의 이동 없이 전류를 제어했다. 전쟁기의 암호 분석과 탄도 계산은 빠른 반복 연산에 큰 수요와 자금을 제공했다. 영국의 콜로서스는 로렌츠 암호 분석에 필요한 논리 연산과 횟수 세기를 전자식으로 수행했고 1944년 가동됐다. 미국의 ENIAC은 더 넓은 수치 문제를 처리하는 전자식 기계로 1946년 공개됐다. 둘 다 전자식이라는 공통점이 있지만 목적과 프로그래밍 방법이 달랐다. 콜로서스를 에니그마 해독기와 동일시하거나, ENIAC의 공개 시점을 컴퓨터 일반의 탄생으로 삼으면 서로 다른 개발 경로가 사라진다. [5]

ENIAC의 빠른 산술을 실제 문제의 해답으로 바꾸려면 사람이 계산 순서를 설계하고 장치를 설정해야 했다. 베티 홀버턴, 진 바틱, 캐슬린 매클널티 등 여섯 여성 프로그래머의 작업은 그 연결을 맡았다. 회로를 만든 사람만으로 컴퓨터의 역사를 설명하면, 문제를 실행 가능한 절차로 옮긴 노동을 놓친다. 초기 컴퓨터는 인간 계산을 즉시 지워 버린 기계가 아니라 계산·설정·검산·운용의 업무를 재편한 기계였다. [6]

명령도 기억하게 만들다

배선을 바꾸어 프로그램을 설정하면 새 문제를 준비하는 시간이 길어진다. 명령을 수와 같은 부호로 만들어 메모리에 저장하면 기계는 다음 명령을 읽고, 실행하고, 조건에 따라 다른 명령으로 이동할 수 있다. 프로그램을 입력하는 작업과 그 프로그램을 실행하는 작업이 분리되고, 프로그램을 복사하거나 다른 프로그램으로 변환하는 길도 열린다. 이 변화가 저장 프로그램 방식의 핵심이다. [7] [1]

1945년 존 폰 노이만이 작성한 『EDVAC 보고서 초안』은 산술부·제어부·기억부·입출력을 연결한 저장 프로그램 구상을 널리 알렸다. 폰 노이만은 1944년 ENIAC 연구진에 합류했고, 초안은 에커트와 모클리 등을 포함한 연구진의 논의를 문서화했다. 코플랜드의 기술사 서술은 에커트가 그보다 앞서 명령과 데이터를 고속 기억장치에 함께 넣을 필요성을 이해했다고 설명한다. 문서 작성과 개념의 전파에 대한 폰 노이만의 역할을 인정하는 것과 구조 전체의 발명을 한 사람에게 귀속하는 것은 구분해야 한다. [1]

같은 시기 영국에서는 튜링의 ACE 설계와 맨체스터의 전자식 기억장치 연구가 진행됐다. 1948년 6월 21일 맨체스터의 '베이비'는 기억장치에 저장된 프로그램을 실행했다. 1949년 케임브리지의 EDSAC은 실제 연구자에게 정기적인 계산 서비스를 제공하는 기계로 이어졌다. 실험실에서 원리를 보이는 것과 다른 사람의 일을 안정적으로 받아 처리하는 것은 다른 단계였다. [5] [1]

'최초'라는 질문의 조건

어느 컴퓨터가 최초였는지에 관한 논의는 기준을 밝힐 때 의미가 생긴다. 제작된 기계인가 설계인가, 디지털인가 아날로그인가, 전자식인가 릴레이식인가, 임무를 바꿀 수 있는가, 명령을 내부 기억장치에 넣는가에 따라 후보가 달라진다. 아타나소프–베리 컴퓨터(ABC)는 연립방정식을 위한 전자식 계산, Z3는 자동 프로그램 제어, 콜로서스는 전자식 암호 분석, ENIAC은 범용 전자식 수치 계산, 베이비는 전자식 저장 프로그램 실행의 역사에서 각각 자리를 갖는다. 한 이름으로 모든 성취를 대표시키기보다 무엇이 가능해졌는지를 구분하는 것이 기술사의 설명력을 높인다. [1] [5]

주

1 B. Jack Copeland, 「The Modern History of Computing」, https://plato.stanford.edu/entries/computing-history/ . 위치: 도입부, Babbage, Analog Computers, Electromechanical versus Electronic Computation, The Manchester Machine, ENIAC and EDVAC.

2 Science Museum, 「Charles Babbage’s Difference Engines and the Science Museum」, https://www.sciencemuseum.org.uk/objects-and-stories/charles-babbages-difference-engines-and-science-museum . 위치: 수학표의 오류와 자동 인쇄, Difference Engine No.2의 1847–1849년 설계 및 1991·2002년 제작. 제곱수 차분은 원리를 설명하기 위한 계산 예시.

3 IBM. 「The punched card」. https://www.ibm.com/history/punched-card . 위치: The evolution of punched cards, The Social Security Administration.

4 William D. Nordhaus. 「The Progress of Computing」. Cowles Foundation Discussion Paper 1324, 2001. https://cowles.yale.edu/sites/default/files/2022-08/d1324.pdf . 위치: 본문 pp.4–6(홀러리스·성능연결), p.16(비용가정), pp.17–19(해석과추정), p.35 Table 4의 1998 prices 열.

5 Computer History Museum, 「Computers」, https://www.computerhistory.org/timeline/computers/ . 위치: 1937 Stibitz, 1941 Z3, 1944 Colossus·Mark I, 1946 ENIAC, 1948 Baby, 1949 EDSAC, 1953 트랜지스터 컴퓨터, 1960 PDP-1, 1966 HP 2116A, 1971 4004, 1975–1977 개인용 컴퓨터, 1981 IBM PC, 1984 Macintosh, 1987 ARM, 1991 PowerBook 및 2000년대 모바일·내장형 항목.

6 ENIAC Programmers Project, https://eniacprogrammers.org/ . 위치: Proving Ground, Hear Their Voices; 여섯 프로그래머의 이름·역할과 구술사 수집 소개.

7 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

3장. 큰 기계가 일상의 장치가 되기까지

소자와 배선을 함께 제조하다

진공관을 사용하는 컴퓨터는 많은 전력과 공간, 냉각과 유지보수를 필요로 했다. 트랜지스터는 이 조건을 바꾸었다. 작고 전력 소모가 적은 전자 소자를 이용하면 같은 기능을 더 좁은 공간에 넣고 더 많은 회로를 구성할 수 있었다. 1950년대에는 트랜지스터를 사용하는 컴퓨터가 만들어졌고, 1960년대 이후에는 회로의 여러 요소를 반도체 위에 함께 만드는 집적회로가 컴퓨터의 제작 방식을 바꾸었다. [1] [2]

회로의 여러 소자와 연결을 반도체 칩에 함께 집적하면 개별 부품을 조립하고 배선하는 부담이 줄어든다. 같은 설계를 반복 생산할 수 있으므로 설계 비용을 더 많은 제품에 나누어 부담할 수 있다. 그러나 소자를 작게 만드는 일만으로 충분하지 않았다. 반도체 표면의 상태를 안정시키고 공정 편차를 관리하는 제조 기술이 필요했다. 벨 연구소의 모하메드 아탈라와 강대원이 개발한 MOS 트랜지스터는 실리콘과 산화막의 계면 문제를 다루면서 전계효과 소자의 길을 열었다. 강대원의 기여는 한국 출신 인물이라는 표지만이 아니라 오늘날 집적회로의 바탕이 된 소자 연구 속에서 이해해야 한다. [3]

1971년에 소개된 인텔 4004는 CPU 기능을 칩에 집적한 중요한 사례다. 일본 계산기 회사 비지컴의 요구, 테드 호프와 스탠리 메이저의 구조 구상, 페데리코 파진과 시마 마사토시의 구현 작업이 결합했다. 계산기의 각 기능을 모두 별도 회로로 만드는 대신 프로그램으로 동작을 정하는 처리기를 사용하면 다른 작업에 재사용할 수 있다. 마이크로프로세서의 역사는 미국 기업의 제품 출시인 동시에 일본의 수요와 국제적인 설계 협업의 역사다. [4] [2]

호환성의 경제

기업이 컴퓨터를 바꿀 때 지출하는 것은 하드웨어 값만이 아니다. 기존 프로그램을 다시 쓰고, 데이터를 옮기고, 직원을 교육하고, 주변 장치를 연결하는 비용도 생긴다. 1964년 IBM System/360은 서로 다른 규모의 기계가 공통된 소프트웨어 구조를 따르는 제품군을 제시했다. 이용자는 더 큰 시스템으로 옮겨 가면서 프로그램 투자의 일부를 보존할 수 있었다. 호환성은 기술적 인터페이스인 동시에 구매 위험을 줄이고 시장을 묶는 경제적 장치였다. [5]

이와 함께 더 작고 운용 부담이 낮은 미니컴퓨터가 실험실과 공장으로 들어갔다. 거대한 전산실을 공동으로 이용하는 것과 장비 곁에 컴퓨터를 놓고 즉시 제어하는 것은 서로 다른 작업 방식이다. PDP 계열과 HP의 계측용 컴퓨터는 기관 전체가 아니라 부서와 실험팀이 직접 계산 자원을 사용할 수 있는 길을 넓혔다. 미니컴퓨터의 의의는 오늘날 기준으로 작았다는 데보다 컴퓨터를 소유하고 사용하는 조직의 단위가 달라졌다는 데 있다. [2]

대형 기계도 사라지지 않았다. 은행의 거래, 항공권 예약, 기업의 기록 처리는 많은 사용자와 장치를 연결하면서 데이터의 일관성과 운용 연속성을 요구했다. 메인프레임과 슈퍼컴퓨터를 단순히 '큰 컴퓨터'로 묶을 수 없는 이유다. 전자는 대규모 업무와 입출력, 호환성과 신뢰성의 가치를 중시해 발전했고, 후자는 과학·공학 계산의 처리량을 극대화하는 방향으로 발전했다. 두 영역은 기술을 공유하지만 같은 지표로 평가되지 않는다. [5] [2]

개인용 컴퓨터는 쓸 일이 생기면서 퍼졌다

1970년대의 마이크로프로세서는 취미가와 소규모 업체도 컴퓨터를 만들 수 있게 했다. 초기 개인용 기계에서는 사용자가 BASIC으로 프로그램을 입력하거나 책과 잡지의 예제를 따라 하는 일이 흔했다. 그러나 대중화에는 하드웨어를 조립하지 않고 사용할 수 있는 완제품, 저장장치, 익숙한 입출력, 유용한 프로그램이 함께 필요했다. 1977년의 Apple II·TRS-80·PET 같은 기계는 이런 묶음을 가정과 학교에 제공했다. [2] [6]

스프레드시트는 이 변화를 잘 보여 준다. 종이 계산표에서는 가정 하나를 바꾸면 관련된 칸들을 사람이 다시 계산해야 한다. 1979년 VisiCalc는 셀의 관계를 수식으로 저장하고 입력 변화에 맞춰 결과를 다시 계산했다. 컴퓨터를 잘 모르는 업무

담당자도 매출이나 비용 가정을 바꿔 결과를 비교할 수 있었다. 프로그램이 구매 이유가 되고, 늘어난 이용자가 다시 프로그램 시장을 키우는 순환이 생겼다. [6]

1981년 IBM PC와 이후 호환 기종의 확산은 하드웨어·운영체제·응용프로그램·주변기기를 큰 시장에서 결합하도록 했다. 그래픽 사용자 인터페이스는 명령을 모두 외우는 대신 화면의 대상과 메뉴를 조작하게 했고, 문서 편집과 출판에서는 화면의 결과와 인쇄물을 가까이 연결했다. PC의 확산을 특정 운영체제 하나의 등장으로 설명하기보다 부품 생산, 호환성, 소프트웨어, 유통, 사용 경험이 서로의 가치를 높인 과정으로 보아야 한다. [2] [6]

이동과 내장

휴대용 컴퓨터에는 연산 성능 외에 배터리·화면·무게·발열이라는 제약이 있다. 스마트폰은 여기에 무선 통신과 카메라, 위치·동작 센서를 결합했다. 여러 기능을 하나의 시스템온칩에 모으면 연결 거리와 공간을 줄일 수 있지만, 무선 송수신과 화면까지 포함한 전체 에너지 예산을 맞춰야 한다. 같은 처리량이라도 책상 위 장치와 주머니 속 장치가 다른 설계를 택하는 이유다. [2] [7]

임베디드 컴퓨터는 화면에 드러나지 않을 수 있다. 자동차나 계측기, 공장의 제어기에서는 센서를 읽고 정해진 시간 안에 출력을 내는 일이 중요하다. 에어컨의 온도 제어를 생각하면 계산이 빠르기만 해서는 부족하다. 센서 값이 믿을 만해야 하고, 제어 주기를 놓치지 않아야 하며, 이상 상태에서 안전하게 동작해야 한다. 작은 CPU·메모리·입출력을 함께 넣은 마이크로컨트롤러는 이런 목적에 맞는다. 컴퓨터의 보급은 사람들이 컴퓨터라고 부르는 물건의 판매만으로 측정되지 않는다. [4] [8]

주

1 B. Jack Copeland, 「The Modern History of Computing」, https://plato.stanford.edu/entries/computing-history/ . 위치: 도입부, Babbage, Analog Computers, Electromechanical versus Electronic Computation, The Manchester Machine, ENIAC and EDVAC.

2 Computer History Museum, 「Computers」, https://www.computerhistory.org/timeline/computers/ . 위치: 1937 Stibitz, 1941 Z3, 1944 Colossus·Mark I, 1946 ENIAC, 1948 Baby, 1949 EDSAC, 1953 트랜지스터 컴퓨터, 1960 PDP-1, 1966 HP 2116A, 1971 4004, 1975–1977 개인용 컴퓨터, 1981 IBM PC, 1984 Macintosh, 1987 ARM, 1991 PowerBook 및 2000년대 모바일·내장형 항목.

3 Computer History Museum, 「1960: Metal Oxide Semiconductor (MOS) Transistor Demonstrated」, https://www.computerhistory.org/siliconengine/metal-oxide-semiconductor-mos-transistor-demonstrated/ . 위치: Atalla·Kahng의 1959년 제작, 산화막과 표면 상태, 집적회로 가능성을 다룬 본문.

4 Computer History Museum, 「1971: Microprocessor Integrates CPU Function onto a Single Chip」, https://www.computerhistory.org/siliconengine/microprocessor-integrates-cpu-function-onto-a-single-chip/ . 위치: MCS-4·4004 설계자, 주변 기능과 microcontroller 구분. 비지컴 수요는 같은 박물관 「Computers」 1971년 4004 항목.

5 IBM, 「System/360」, https://www.ibm.com/history/system-360 . 위치: 1964년 4월 7일 발표, software-compatible architecture, growth without reprogramming, System/360 spawns new markets. 호환성은 CHM 「Computers」 1964 System/360 항목과도 대조.

6 Computer History Museum, 「Software & Languages」, https://www.computerhistory.org/timeline/software-languages/ . 위치: 1952 A-0, 1957 FORTRAN·FLOW-MATIC, 1960 COBOL, 1961 CTSS, 1963 Sketchpad, 1964 BASIC, 1967 LOGO, 1969 UNIX, 1972 C, 1979 VisiCalc, 1981 MS-DOS, 1983 GNU, 1985 PageMaker, 1991 Linux, 2010 Stuxnet, 2014 Heartbleed.

7 Steve Ward, 「CMOS」, https://computationstructures.org/notes/cmos/notes.html . 위치: MOSFET, NFETs and PFETs, Pullup and Pulldown Circuits, CMOS inverter, 전력과 전압의 관계.

8 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「I/O Devices」, https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf . 위치: §36.2–36.6, 장치 모델·폴링·인터럽트·DMA·장치 드라이버.

4장. 비트에서 논리회로까지

0과 1은 물리 상태를 읽는 약속이다

비트는 두 대안을 구별하는 단위다. 전자회로에서는 보통 전압의 낮은 범위와 높은 범위를 0과 1에 대응시킨다. 실제 전압은 연속적으로 변하지만, 회로는 일정한 범위 안의 값을 같은 논리값으로 읽는다. 출력이 다음 회로에 전달될 때 약간의 잡음이 생겨도 결과가 바뀌지 않도록 여유를 두는 것이 디지털 추상화의 핵심이다. '전류가 흐르면 1, 안 흐르면 0'이라는 설명만으로는 전압의 허용 범위와 잡음 여유를 설명할 수 없다. [1]

비트가 n개면 가능한 배열은 2^n개다. 여덟 비트인 한 바이트에는 256개의 서로 다른 배열이 있다. 그러나 배열의 의미는 정해져 있지 않다. 01000001을 수로 읽으면 65이고, 특정 문자 부호화에서는 A를 나타낸다. 같은 바이트가 색의 일부나 명령어의 일부가 될 수도 있다. 프로그램과 자료 형식은 비트의 배열을 해석하는 약속이다. [1] [2]

이 약속은 세계화의 조건이기도 했다. 초기 문자 부호는 서로 다른 언어와 기계에서 같은 숫자에 다른 글자를 배정할 수 있었다. 유니코드는 문자에 공통된 코드값을 부여해 이런 충돌을 줄인다. 한글과 아랍 문자와 수학 기호를 같은 문서에서 다루려면 글자를 부호화하는 것뿐 아니라 입력·조합·글꼴·표시도 맞아야 한다. 문자 처리는 '이미 컴퓨터가 생긴 뒤 추가한 장식'이 아니라 사람들이 자기 언어로 계산 자원을 쓰게 만드는 기반이다. [2]

논리 연산과 상태

AND는 두 입력이 모두 1일 때 1을 내고, OR는 적어도 하나가 1일 때 1을 내며, NOT은 값을 뒤집는다. XOR는 두 입력이 다를 때 1을 낸다. 한 자리 이진 덧셈에서는 XOR가 합의 낮은 자리를, AND가 다음 자리로 넘길 올림을 만든다. 1과 1을 더한 결과는 이진수 10이다. 앞자리에서 넘어온 올림까지 처리하도록 회로를 확장하고 여러 자리를 연결하면 큰 수의 덧셈을 할 수 있다. [1] [3]

계산을 이어 가려면 결과를 보관해야 한다. 입력만으로 출력이 정해지는 조합회로와 달리 순차회로는 이전 상태가 다음 동작에 영향을 준다. 레지스터가 현재 값을 보관하고 다음 순간 새 값을 받아들이면, '지금까지의 합'과 '다음에 더할 수'를 유지하면서 반복 계산을 할 수 있다. 일반적인 동기식 회로는 클록의 정해진 시점에 상태를 바꾼다. 회로가 안정되기 전에 다음 값을 읽지 않도록 지연 시간을 고려해야 한다. [3] [1]

트랜지스터가 스위치를 구현하는 방식

MOS 트랜지스터에서는 게이트에 가한 전압이 소스와 드레인 사이의 전도 상태를 바꾼다. CMOS 반전기는 상보적인 소자를 이용해 입력이 낮을 때 출력을 높은 쪽으로, 입력이 높을 때 출력을 낮은 쪽으로 연결한다. 여러 소자의 연결 관계를 바꾸면 NAND나 NOR 등 다른 논리 기능을 구현할 수 있다. 논리식은 원하는 관계를 나타내고, 트랜지스터 회로는 그 관계를 실제 전압으로 만든다. [4]

트랜지스터 하나가 언제나 비트 하나를 저장하는 것은 아니다. 소자는 논리 게이트, 증폭기, 저장 셀, 연결 제어 등 여러 곳에서 사용된다. SRAM의 한 비트를 안정적으로 유지하는 셀에는 여러 트랜지스터가 필요하고, DRAM은 전하를 저장하는 소자와 접근 소자를 이용한다. 칩의 트랜지스터 수를 곧 메모리의 비트 수나 성능으로 읽을 수 없는 이유다. [5]

상태가 바뀔 때 배선과 소자의 정전용량을 충전하고 방전해야 하므로 에너지가 든다. 단순화한 CMOS 모델에서 동적 전력은 스위칭 빈도, 정전용량, 전압의 제곱에 비례한다. 실제 칩에는 누설 전력과 다른 손실도 있다. 소자를 더 많이 넣고 클록을 올리는 설계는 전력과 열의 부담을 함께 만든다. 집적도가 높아지는 것과 모든 소자를 언제나 최고 속도로 사용할 수 있는 것은 같은 일이 아니다. [4]

주

1 Steve Ward, 「The Digital Abstraction」, https://computationstructures.org/notes/digitalabstraction/notes.html . 위치: §5.1–5.4, 이산 표현·조합 장치·잡음 여유. 이진수 조합과 덧셈은 정의에서 도출한 설명 예시.

2 Unicode Consortium, 「What is Unicode?」, https://www.unicode.org/standard/WhatIsUnicode.html . 위치: Characters Before Unicode, Unicode Characters.

3 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

4 Steve Ward, 「CMOS」, https://computationstructures.org/notes/cmos/notes.html . 위치: MOSFET, NFETs and PFETs, Pullup and Pulldown Circuits, CMOS inverter, 전력과 전압의 관계.

5 Bryant·O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2판 6장, https://csapp.cs.cmu.edu/2e/ch6-preview.pdf . 위치: 6장 도입과 §6.1.1 Random Access Memory; 계층의 속도·용량·비용, 지역성, SRAM·DRAM.

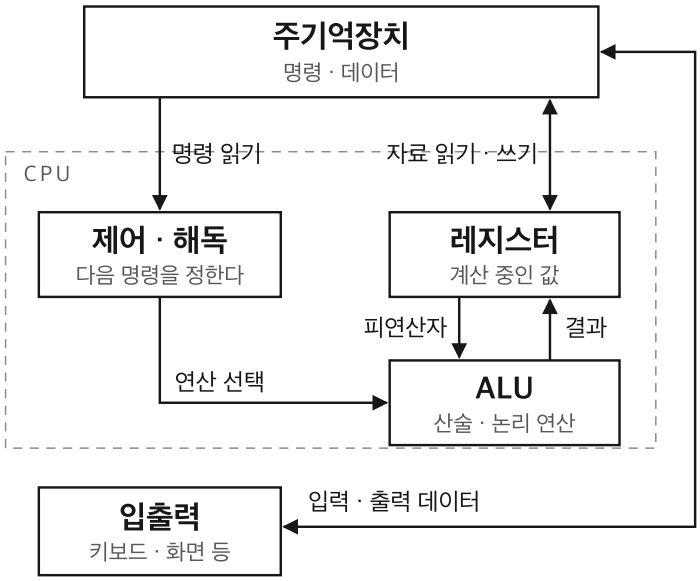
5장. CPU가 혼자서 컴퓨터를 움직이지는 않는다

명령어 집합과 실행

CPU가 실행하는 명령에는 덧셈·논리 연산·비교·메모리 읽기와 쓰기·분기 등이 있다. 명령어 집합 구조인 ISA는 소프트웨어가 하드웨어에 기대할 수 있는 약속이다. 어떤 레지스터가 보이고 어떤 명령이 어떤 결과를 만드는지 규정한다. 같은 ISA를 구현한 CPU라도 파이프라인과 캐시, 실행 장치의 수는 다를 수 있다. 이 분리 덕분에 소프트웨어의 호환성을 유지하면서 내부 설계를 바꿀 수 있다. [1]

가장 단순한 실행 모델에서 CPU는 프로그램 카운터가 가리키는 명령을 가져와 해독하고, 필요한 값을 읽어 연산한 뒤 결과를 기록한다. 다음 명령으로 이동하다가 분기를 만나면 조건에 따라 다른 주소로 간다. 1부터 100까지 더하는 프로그램이라면 합을 0으로 시작하고, 현재 수를 더하고, 수를 증가시키고, 끝에 도달했는지 검사하는 명령들이 반복된다. 덧셈 회로의 재사용에 상태와 제어 흐름이 더해진 것이다. [1]

실제 고성능 CPU는 이 단계를 겹쳐 수행한다. 파이프라인은 여러 명령이 서로 다른 단계를 통과하도록 하고, 독립적인 명령은 여러 실행 장치에서 진행할 수 있다. 분기 결과를 예측하거나 준비된 연산을 먼저 처리하는 방법도 쓰인다. 그러나 내부 실행 순서를 바꿔도 프로그램에 약속한 결과와 메모리 규칙은 지켜야 한다. 클록이 높다는 숫자만으로 서로 다른 CPU의 프로그램 실행 시간을 비교할 수 없는 이유도 여기에 있다. [2] [3]



저장프로그램 방식에서 명령과 자료가 도는 경로. 실제 칩의 배선도가 아니라 이 절이 설명하는 실행 모델을 단순화한 기능도다.
근거: Steve Ward, Instruction Set Architectures (https://computationstructures.org/notes/isas/notes.html) — 본문 ISA 각주의 출처.

기억장치는 계층을 이룬다

CPU 가까이의 레지스터와 캐시는 빠르지만 용량이 제한된다. 주기억장치는 더 많은 실행 상태를 보관하고, SSD나 하드디스크 같은 저장장치는 전원을 꺼도 파일을 남긴다. 속도·용량·비용·지속성의 차이 때문에 한 종류의 저장장치가 모든 역할을 맡지 않는다. 컴퓨터는 필요한 자료를 먼 곳에서 가까운 곳으로 가져오며 일한다. [4]

캐시의 효과는 지역성에 기대고 있다. 최근 사용한 값을 곧 다시 쓰는 시간적 지역성과, 가까운 주소들을 이어서 읽는 공간적 지역성이다. 큰 표의 값을 차례로 읽으면 한 번 가져온 데이터 덩어리의 많은 부분을 사용할 수 있다. 여기저기 흩어진 값을

읽으면 계산량이 같아도 자료를 기다리는 시간이 늘 수 있다. 메모리의 용량이 충분하다는 사실만으로 대역폭과 지연의 문제가 사라지지 않는다. [4]

SRAM은 전원이 공급되는 동안 회로의 안정된 상태로 값을 유지하고, DRAM은 누설되는 전하를 재생해야 한다. 플래시 메모리는 전원을 꺼도 상태를 남기지만 쓰기와 지우기의 방식, 내구성에 다른 제약이 있다. 저장장치가 '반도체'라는 공통점만으로 실행 중인 RAM과 파일용 SSD가 같은 자원이 되지는 않는다. 프로그램의 작업 공간과 장기 보존 공간을 구별해야 컴퓨터의 지연과 데이터 손실을 이해할 수 있다. [4] [5]

입출력과 운영체제의 협력

키보드·화면·네트워크·저장장치는 CPU와 다른 속도로 움직인다. 장치가 준비됐는지 계속 묻는 폴링은 간단하지만 CPU 시간을 소비한다. 인터럽트는 장치가 주의를 요구할 때 CPU에 알리고, 운영체제가 해당 처리를 하도록 한다. 큰 데이터 전송에는 DMA를 이용해 CPU가 바이트마다 복사하지 않고 장치와 메모리 사이에서 전송하게 할 수 있다. CPU는 계산을 하면서 다른 장치가 일을 끝내기를 기다릴 수 있다. [6]

계산기 프로그램에 '5+3'을 입력하는 장면을 따라가 보자. 입력장치의 사건이 운영체제와 응용프로그램에 전달된다. 프로그램은 글자 '5'를 계산에 쓰는 수로 바꾸고, 덧셈 동작을 선택한다. 이 단순화한 실행 예에서 CPU는 관련 명령을 수행하고 결과 8을 레지스터나 메모리에 보관한다. 프로그램은 이를 다시 표시할 글자로 바꾸고 그래픽 출력 경로가 화면을 갱신한다. 이 과정은 입력·표현·연산·출력의 연결이며, 어느 한 단계도 나머지 전체를 대신하지 않는다. [1] [6]

화면에 8이 보이는 것과 파일에 안전하게 저장된 것은 별개의 사건이다. 저장 요청이 메모리의 버퍼에만 반영됐는지, 실제 저장장치까지 도달했는지에 따라 전원 차단 뒤의 결과가 달라진다. 컴퓨터의 정확성은 산술 결과만이 아니라 어떤 결과를 언제까지, 어떤 실패에도 보존할 것인지라는 약속까지 포함한다. [3]

주

1 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

2 Steve Ward, 「Performance Measures」, https://computationstructures.org/notes/performance/notes.html . 위치: §11.1 Latency versus Throughput 및 파이프라인 설명.

3 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「Introduction to Operating Systems」, https://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf . 위치: §2.1–2.4, CPU·메모리 가상화, 동시성, 영속성. 잔액 동시 갱신은 공유 상태 문제의 설명 예시.

4 Bryant·O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2판 6장, https://csapp.cs.cmu.edu/2e/ch6-preview.pdf . 위치: 6장 도입과 §6.1.1 Random Access Memory; 계층의 속도·용량·비용, 지역성, SRAM·DRAM.

5 Computer History Museum, 「Memory & Storage」, https://www.computerhistory.org/timeline/memory-storage/ . 위치: 반도체 메모리·DRAM·플래시 저장 관련 항목.

6 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「I/O Devices」, https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf . 위치: §36.2–36.6, 장치 모델·폴링·인터럽트·DMA·장치 드라이버.

6장. 프로그램을 만드는 프로그램

알고리즘이 바꾸는 계산량

같은 문제를 푸는 절차에도 차이가 크다. 이름이 정렬되지 않은 명단에서는 찾는 이름이 나올 때까지 하나씩 비교할 수 있다. 정렬된 명단에서는 가운데 이름을 보고 앞 절반과 뒤 절반 중 하나를 버린 뒤 같은 과정을 반복한다. 이것이 이진 탐색이다. 1,000개의 항목을 하나씩 찾으면 최악에 1,000번 확인하지만 이진 탐색은 최대 약 10번의 비교 단계로 범위를 좁힐 수 있다. [1]

이 차이는 단순히 더 빠른 컴퓨터를 사는 것과 다르다. 항목 수가 두 배가 되면 순차 탐색의 최악 비용은 두 배가 되지만, 이진 탐색의 비교 단계는 하나 정도 늘어난다. 대신 명단이 미리 정렬돼 있어야 하고 가운데 항목에 효율적으로 접근할 수 있어야 한다. 자료를 정리하는 비용과 조회 횟수를 함께 보아야 한다. 알고리즘과 자료 구조는 하드웨어 성능을 어떻게 유효한 작업으로 바꿀지 결정한다. [1]

언어가 사람의 작업 단위를 바꾸다

기계어로 프로그램을 적으면 연산과 주소의 세부를 사람이 직접 관리해야 한다. 어셈블리어는 숫자 명령에 이름을 붙여 읽기 쉽게 만들지만 여전히 특정 ISA에 가깝다. 고급 언어는 변수·반복·함수·자료 구조를 통해 사람이 문제를 표현하는 단위를 높인다. 컴파일러는 이런 표현을 다른 실행 표현으로 번역하는 프로그램이다. 저장 프로그램 컴퓨터는 프로그램을 데이터처럼 다룰 수 있기 때문에 프로그램을 변환하는 프로그램도 실행할 수 있다. [2] [3]

1950년대 FORTRAN의 중요한 과제는 과학자가 수식을 더 편리하게 쓰면서도 손으로 작성한 기계어에 가까운 효율을 얻는 것이었다. 그레이스 호퍼의 A-0와 FLOW-MATIC으로 이어진 작업, COBOL의 업무용 표현은 프로그램 작성에 들어가는 사람의 시간을 줄이려는 다른 경로였다. 언어의 발전은 표현을 편하게 한 역사이면서, 부족한 기계 자원을 아껴야 한다는 요구와 비싼 개발 노동을 줄여야 한다는 요구를 조정한 역사다. [4]

컴파일러의 일을 `y = x + 3`이라는 식으로 살펴보자. 변수 x의 값을 기억장치에 두는 단순한 기계라면, 번역 결과는 'x를 레지스터에 읽기 → 3 더하기 → 결과를 y의 위치에 쓰기'라는 명령열이 될 수 있다. x가 이미 레지스터에 있으면 첫 읽기를 생략할 수 있고, x가 항상 5라는 사실을 알면 식 전체를 상수 8로 바꿀 수도 있다. 앞의 명령열은 교육용 ISA에 맞춰 구성한 예시다. 핵심은 한 줄을 한 명령으로 바꾸는 것이 아니라, 언어가 약속한 결과를 유지하면서 대상 기계에서 실행할 표현을 만드는 데 있다. [5] [2]

컴파일과 해석은 한 실행 환경에서 결합될 수 있다. 소스에서 중간 코드를 만들고 가상 머신이 이를 실행하거나, 실행 중 일부를 기계어로 바꿀 수 있다. 그래서 언어의 이름만으로 실행 속도를 판정하기보다 어떤 변환을 언제 수행하고 그 비용을 얼마나 반복 부담하는지를 보아야 한다. [3]

언어는 오류를 줄이는 도구이기도 하다. 타입은 어떤 값을 어떤 연산에 사용할 수 있는지를 제한하고, 메모리 관리 규칙은 더 이상 유효하지 않은 주소를 읽는 문제를 줄이려 한다. 이런 안전장치는 구현 비용과 실행 비용을 동반할 수 있다. 언어마다 성능·제어권·안전성·표현력의 우선순위가 다른 까닭이다. 문법이 간단한 것과 큰 시스템을 안전하게 만들기 쉬운 것은 같은 성질이 아니다. [2]

운영체제는 자원을 나누고 실패를 다룬다

운영체제는 화면의 생김새보다 깊은 층에 있다. 프로그램마다 CPU와 메모리를 사용할 기회를 주고, 장치 접근을 조정하고, 파일을 관리한다. 프로세스는 실행 중인 프로그램의 상태를 나타내며, 운영체제는 레지스터와 주소 공간 등을 관리해 여러 실행을 분리한다. 가상 메모리는 각 프로그램에 독립된 주소 공간을 제공하면서 실제 물리 메모리를 배분하는 데 쓰인다. [6]

시분할은 비싼 컴퓨터를 여러 사람이 상호작용하며 이용하려는 요구에서 발전했다. 한 프로그램이 느린 입출력을 기다리는 동안 다른 프로그램을 실행하면 기계를 더 잘 활용할 수 있다. 1960년대 CTSS와 이후의 유닉스는 단말을 통한 상호작용, 파일과

프로그램 도구의 결합을 넓혔다. 유닉스가 C로 다시 작성되면서 다른 하드웨어로 옮기는 일이 수월해진 과정은 언어와 운영체제의 변화가 서로를 밀어 준 사례다. [4] [6]

동시에 진행되는 일은 정확성을 어렵게 한다. 잔액 100을 두 작업이 함께 읽고 각각 10을 더한 뒤 110을 쓰면, 두 번 입금했는데도 120이 되지 않는다. 각 작업이 '전부 반영되거나 전부 취소된다'는 원자성을 지켰어도 이런 유실 갱신은 생길 수 있다. **원자성은 한 트랜잭션의 변경을 묶는 속성이고, 격리는 서로 다른 트랜잭션이 간섭하는 방식을 통제하는 속성이다.** [7] [8]

PostgreSQL의 Read Committed 방식에서 두 작업이 각각 `UPDATE accounts SET balance = balance + 10`을 실행하면 같은 행을 바꾸는 두 번째 작업은 첫 번째 작업을 기다린 뒤 갱신된 잔액에 10을 더한다. 둘 다 커밋하면 120이 된다. 반면 두 프로그램이 앞서 읽은 100을 바탕으로 계산한 상수 110을 각각 쓰면 기다리더라도 값은 110으로 남을 수 있다. 읽기부터 쓰기까지 행 잠금을 유지하거나, 적절한 격리 수준과 충돌 시 재시도를 사용하는 이유다. Serializable 수준은 커밋된 트랜잭션들의 결과가 어떤 순차 실행과 같도록 보장하며, 이를 위해 충돌한 트랜잭션을 중단시킬 수 있다. [8]

파일 저장에는 중단 뒤 구조를 복구하는 문제도 있다. 파일에 내용을 추가하면 데이터 블록뿐 아니라 파일 크기, 블록 위치, 사용 중인 공간의 기록도 바뀐다. 일부만 저장된 순간 전원이 꺼지면 이 기록들이 서로 맞지 않을 수 있다. 저널링은 변경할 내용을 먼저 로그에 기록하고, 그 기록이 안전하게 저장된 뒤 커밋 표시를 남기고, 이후 원래 위치에 변경을 적용한다. 재시작 때 완료된 로그는 다시 적용하고 미완료 기록은 건너뛰어 일관된 상태를 복구한다. OSTEP의 데이터 저널링 예시는 이 세 단계와 쓰기 순서의 필요성을 설명한다. 메타데이터만 저널링하는 방식은 파일 내용 전체까지 같은 보장을 하지 않으므로 무엇을 로그에 남기는지도 중요하다. [9]

소프트웨어의 경제와 유지보수

소프트웨어는 같은 기능을 반복해 배포할 수 있지만, 만들고 유지하는 비용이 사라지지는 않는다. 운영체제·라이브러리·응용프로그램이 서로 의존하고, 데이터 형식과 사용자의 업무가 오랜 시간 쌓인다. 새 버전이 더 좋은 기능을 갖췄더라도 기존 파일이나 장치, 업무와 호환되지 않으면 전환 비용이 커진다. 그래서 오래된 소프트웨어가 단순한 관성 때문에만 남는 것은 아니다. [4] [10]

공개된 소스와 재사용을 허용하는 라이선스는 다른 개발자가 프로그램을 검토하고 수정하고 배포하는 길을 연다. GNU와 리눅스의 전개는 도구와 커널, 여러 공동체의 작업이 결합해 넓은 하드웨어에서 사용되는 환경을 만든 사례다. 동시에 널리 공유하는 작은 라이브러리의 결함이 많은 시스템으로 퍼질 수도 있다. 소프트웨어를 공동으로 쓴다는 이득에는 누가 변경을 검토하고 보안 업데이트를 유지할지라는 과제가 따라온다. [4] [11]

주

1 University of Canterbury, *Computer Science Field Guide*, 「Searching」, https://www.csfieldguide.org.nz/en/chapters/algorithms/searching/ . 위치: §2.2.1 Linear search, §2.2.2 Binary search; 1,000개 항목 예시와 정렬 전제.

2 Steve Ward, 「Compiled Languages」, https://computationstructures.org/notes/compiled/notes.html . 위치: 장 도입, §17.1 Programming Language Features, §17.1.1 Dangling References. 하드웨어 중심 표현·고급언어·타입과 메모리 관리의 설명.

3 University of Canterbury, *Computer Science Field Guide*, 「How does the computer process your program?」, https://www.csfieldguide.org.nz/en/chapters/programming-languages/how-does-the-computer-process-your-program/ . 위치: §3.4, 컴파일러·인터프리터와 혼합 구현.

4 Computer History Museum, 「Software & Languages」, https://www.computerhistory.org/timeline/software-languages/ . 위치: 1952 A-0, 1957 FORTRAN·FLOW-MATIC, 1960 COBOL, 1961 CTSS, 1963 Sketchpad, 1964 BASIC, 1967 LOGO, 1969 UNIX, 1972 C, 1979 VisiCalc, 1981 MS-DOS, 1983 GNU, 1985 PageMaker, 1991 Linux, 2010 Stuxnet, 2014 Heartbleed.

5 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

6 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「Introduction to Operating Systems」, https://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf . 위치: §2.1–2.4, CPU·메모리 가상화, 동시성, 영속성. 잔액 동시 갱신은 공유 상태 문제의 설명 예시.

7 PostgreSQL Global Development Group, 「Transactions」, https://www.postgresql.org/docs/current/tutorial-transactions.html . 위치: 트랜잭션의 all-or-nothing·동시 실행에 대한 가시성·완료 후 지속성 설명. 은행 송금 예시.

8 PostgreSQL Global Development Group. 「Transaction Isolation」, PostgreSQL 18. https://www.postgresql.org/docs/18/transaction-iso.html . 위치: §13.2.1 Read Committed, §13.2.3 Serializable.

9 Arpaci-Dusseau·Arpaci-Dusseau. 「Crash Consistency: FSCK and Journaling」, OSTEP 42장. https://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf . 위치: §42.1, §42.3 Data Journaling·Recovery·Metadata Journaling.

10 IBM, 「System/360」, https://www.ibm.com/history/system-360 . 위치: 1964년 4월 7일 발표, software-compatible architecture, growth without reprogramming, System/360 spawns new markets. 호환성은 CHM 「Computers」 1964 System/360 항목과도 대조.

11 NIST, *Cybersecurity Framework 2.0*, https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf . 위치: §1–2 및 Appendix A, GOVERN·IDENTIFY·PROTECT·DETECT·RESPOND·RECOVER와 공급망·권한·데이터 보호·복구 항목.

7장. 연결된 컴퓨터는 새로운 실패를 만난다

패킷과 네트워크의 연결

초기 컴퓨터 네트워크의 중요한 동기는 비싼 계산 자원과 데이터를 원격에서 공유하는 것이었다. 데이터 통신은 전화 대화처럼 일정한 흐름이 계속 이어지기보다 요청과 응답이 몰렸다가 쉬는 경우가 많다. 패킷 교환은 데이터를 작은 단위로 나눠 통신 경로를 함께 사용하게 한다. 한 사용자가 침묵하는 동안 다른 사용자가 같은 기반시설을 이용할 수 있다. [1]

ARPANET은 1969년 연구기관의 컴퓨터들을 연결하기 시작했다. 이후의 과제는 같은 네트워크 안의 기계를 잇는 것을 넘어 서로 다른 네트워크를 연결하는 것이었다. 영국 NPL의 패킷 연구, 미국의 ARPA 연구, 대학과 연구소의 구현들이 이 과정에 참여했다. TCP/IP는 각 네트워크의 내부를 모두 같은 방식으로 바꾸지 않아도 상호 통신할 수 있는 공통 규칙을 제공했다. 인터넷은 한 대의 초대형 컴퓨터가 아니라 서로 독립적으로 관리되는 네트워크들의 연결이다. [1]

IP는 목적지로 패킷을 전달하고, TCP는 손실과 순서 뒤바뀜을 다루면서 응용프로그램에 순서 있는 바이트 흐름을 제공한다. 이름을 주소와 연결하는 DNS는 이름 공간을 나누어 관리하게 한다. 웹과 전자우편은 이런 기반 위에서 동작하는 응용이다. 인터넷과 웹을 같은 말로 쓰면 기반 통신과 그 위의 서비스가 구분되지 않는다. [2] [1]

공개된 규격은 다른 조직이 각자 구현한 장치를 연결하도록 했다. RFC 문서, 운영체제에 포함된 네트워크 구현, 연구망에 대한 공적 투자와 상용 서비스의 확장은 함께 작용했다. 연결되는 이용자가 늘수록 같은 네트워크에 참여할 가치가 커지고, 그 가치는 다시 프로그램과 사업의 투자를 불렀다. 네트워크의 확산에는 기술의 우수성뿐 아니라 문서 접근성, 운영 주체의 협력, 연결 비용을 누가 부담하는지가 중요했다. [1]

주소와 링크로 묶인 문서

웹은 네트워크 위에 문서와 자원을 연결하는 규칙을 만들었다. URL은 자원을 가리키는 주소이고, HTTP는 브라우저가 그 자원을 요청하고 서버가 응답하는 통신 규칙이며, HTML은 문서의 구조와 다른 자원으로 가는 링크를 표현한다. 사용자가 링크를 누르면 그 주소의 자원을 요청하고, 받은 문서에 포함된 그림이나 다른 자원도 추가로 읽어 화면을 구성한다. 웹의 통일된 사용 경험 아래에는 여러 서버와 형식이 협력한다. [3]

팀 버너스리는 1989년 CERN에서 서로 다른 컴퓨터와 자료 체계에 흩어진 정보를 연결하려고 하이퍼텍스트 기반 시스템을 제안했다. 모든 자료를 하나의 데이터베이스로 옮기기보다 기존 정보를 연결하고, 다른 시스템으로도 읽게 하는 것이 중요했다. W3C의 역사 기록은 1993년 CERN이 웹 기술 사용에 사용료를 받지 않겠다고 발표한 일과 여러 운영체제용 Mosaic 브라우저의 확산을 함께 기록한다. 공개된 연결 규칙과 이용하기 쉬운 브라우저가 만난 것은 지식과 서비스를 조직 경계 밖으로 배포하는 비용을 낮추는 경로가 됐다. [4] [5]

분산의 이득과 모호함

작업과 데이터를 여러 컴퓨터에 나누면 처리량을 늘리고, 사용자 가까이에 서비스를 두고, 한 기계의 장애에 대비할 수 있다. 그러나 메시지가 도착하지 않았을 때 원인을 바로 알 수는 없다. 상대 기계가 멈췄는지, 네트워크가 끊겼는지, 응답만 늦는지 구분하기 어렵다. 한 대의 컴퓨터 안에서 당연하게 여긴 메모리 접근과 함수 호출의 가정을 그대로 원격 호출에 적용하면 이런 차이를 놓친다. [6]

송금 요청을 보냈는데 응답을 받지 못한 경우를 생각해 보자. 서버가 요청을 받기 전에 고장 났을 수도 있고, 송금을 끝낸 뒤 응답이 사라졌을 수도 있다. 무조건 재시도하면 중복 송금이 될 수 있다. 요청 식별자를 기록해 같은 요청의 재실행을 구분하고, 처리 결과를 다시 조회할 수 있게 하는 등 응용 수준의 설계가 필요하다. 통신 계층이 바이트를 전달했다는 보장과 업무가 한 번 처리됐다는 보장은 다르다. [6] [2]

복제본을 여러 곳에 두는 것도 단순한 복사보다 어렵다. 주 기계가 결과를 외부에 알렸는데 백업에는 그 결과를 재현할 정보가 없다면, 주 기계의 고장 뒤에 이미 확인해 준 작업을 잃을 수 있다. MIT의 가상 머신 내결함성 강의는 결과를 보내기 전에 백업이 앞선 실행 기록을 받았다고 확인하도록 하는 사례를 설명한다. 이런 확인은 복구 가능성을 높이지만 외부 응답에 기다림을 더한다. 복제는 장애 대비와 지연을 함께 다루는 설계다. [7]

클라우드는 소유와 운영의 방식을 바꾼다

NIST의 정의에서 클라우드는 공유된 계산 자원을 네트워크로 필요할 때 확보하고 해제하는 모델이다. 주문형 셀프서비스, 폭넓은 네트워크 접근, 자원 풀링, 신속한 탄력성, 사용량 측정이 주요 특징이다. 서버를 원격에 둔 것만으로 모든 조건을 충족하는 것은 아니다. [8]

이용자는 서버·저장장치 같은 기반 자원을 빌릴 수도 있고, 프로그램을 올릴 실행 환경을 이용할 수도 있으며, 완성된 응용서비스를 사용할 수도 있다. 각각 IaaS·PaaS·SaaS라 부르는 구분은 누가 어느 층을 운영하는지를 드러낸다. 물리 서버 관리가 제공자에게 넘어가도 이용자 계정·데이터·응용 설정에 관한 책임이 자동으로 없어지지는 않는다. [8] [9]

짧은 기간에만 큰 계산이 필요한 연구팀을 생각하면 탄력성의 효용이 분명해진다. 필요할 때 자원을 늘리고 작업이 끝나면 해제할 수 있다. 다만 사용량을 측정해 서비스 비용을 정하는 모델에서는 실행 시간과 저장 용량, 어떤 서비스를 이용하는지가 계속 중요하다. NIST의 서비스 구분에 따라 이용자가 관리할 수 있는 층도 달라진다. 완성된 응용을 이용하는 것과 운영체제까지 직접 관리하는 것을 같은 운영 부담으로 비교할 수는 없다. [8]

주

1 Leiner 외, 「A Brief History of the Internet」, https://www.internetsociety.org/internet/history-internet/brief-history-internet/ . 위치: Origins of the Internet, The Initial Internetting Concepts, Proving the Ideas, Transition to Widespread Infrastructure, The Role of Documentation. 1997년 참여자 회고로 서술된 초기 역사.

2 W. Eddy 편, *RFC 9293*, https://www.rfc-editor.org/rfc/rfc9293.html . 위치: §2.2, TCP의 reliable, in-order, byte-stream service.

3 MDN. 「How the web works」. https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works . 위치: Clients and servers, DNS·HTTP·component files, 페이지 요청과조립.

4 Tim Berners-Lee. 「Information Management: A Proposal」, 1989. https://www.w3.org/History/1989/proposal.html . 위치: Linked information systems, A solution: Hypertext, Requirements.

5 W3C. 「A Little History of the World Wide Web」. https://www.w3.org/History.html . 위치: 1989 제안, 1993-04-30 CERN 사용료면제 및 Mosaic 항목.

6 Arpaci-Dusseau·Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 「Distributed Systems」, https://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf . 위치: 장 도입, §48.1–48.5, 실패·신뢰성·재전송·RPC. 송금은 재시도와 중복 처리 문제의 설명 예시.

7 MIT PDOS, 「VMware FT」 강의노트, https://pdos.csail.mit.edu/6.824/notes/l-vm-ft.txt . 위치: Output rule, backup acknowledgement·backup lag 설명. 특정 주/백업 구조의 사례이며 모든 분산 시스템의 동일한 구현을 뜻하지 않음.

8 Mell·Grance, *The NIST Definition of Cloud Computing*, SP 800-145, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf . 위치: 본문 pp.2–3, Essential Characteristics·Service Models·Deployment Models.

9 NIST, *Cybersecurity Framework 2.0*, https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf . 위치: §1–2 및 Appendix A, GOVERN·IDENTIFY·PROTECT·DETECT·RESPOND·RECOVER와 공급망·권한·데이터 보호·복구 항목.

8장. 더 빠른 계산은 어떻게 만들어지는가

지연, 처리량, 병렬성

한 작업이 끝나는 데 걸리는 시간은 지연이고, 일정 시간에 끝내는 작업의 양은 처리량이다. 사용자 요청 한 건의 빠른 응답과 하루 동안 처리할 요청 수의 최대화는 서로 다른 목표가 될 수 있다. CPU는 개별 작업의 빠른 반응과 복잡한 제어를 위해 많은 회로를 쓰고, GPU와 가속기는 많은 비슷한 연산을 함께 처리하는 데 자원을 집중할 수 있다. [1] [2]

병렬화에는 독립적으로 진행할 수 있는 일이 있어야 한다. 한 작업의 절반만 병렬화할 수 있고 나머지 절반은 반드시 순서대로 해야 한다면, 병렬 부분을 아무리 빠르게 해도 전체 속도 향상은 두 배를 넘지 못한다. 이는 통신·동기화 비용을 무시한 상한이다. 코어를 늘리는 것보다 순차 부분과 데이터 이동을 줄이는 일이 더 중요할 수 있다. [3]

GPU가 그래픽에서 강한 이유는 많은 픽셀과 정점에 비슷한 연산을 적용할 수 있기 때문이다. 과학 계산과 신경망에서도 벡터·행렬 연산이 반복되면 이 구조를 활용할 수 있다. 반면 작업마다 분기 경로가 크게 다르거나 작은 데이터에 짧은 계산만 필요하면 연산 장치를 충분히 활용하지 못할 수 있다. GPU의 최대 연산량과 실제 프로그램의 가속률은 다르다. [3] [2]

연산보다 데이터가 비쌀 때

학습 기반 프로그램에서는 사람이 모든 입력에 대한 답의 규칙을 직접 적는 대신, 학습 알고리즘이 자료를 이용해 모델의 가중치를 조정한다. 학습된 모델에 새 입력을 넣어 출력을 얻는 단계가 추론이다. 학습 규칙을 구현한 프로그램, 가중치를 정하는 데 쓰인 데이터, 그 결과인 모델 매개변수를 구분해야 한다. 2017년 TPU 논문은 학습과 추론에 필요한 수치 형식과 응답시간 조건을 구별하고, 추론을 대상으로 TPU의 성능과 전력을 평가했다. [2]

신경망의 한 층은 입력 벡터와 가중치 행렬을 곱하고 결과를 변환하는 형태로 나타낼 수 있다. 같은 가중치를 여러 입력에 재사용하거나 중간 결과를 가까운 메모리에 유지하면 매 연산마다 외부 메모리에서 데이터를 가져오는 일을 줄일 수 있다. 행렬 곱 전용 배열은 곱셈·누산을 규칙적으로 이어서 데이터가 계산 장치 사이를 흐르게 한다. [2]

구글 연구진이 2017년 발표한 초기 TPU 연구는 이런 설계 선택의 구체적인 사례다. 이 칩은 신경망 추론을 위해 8비트 정수 곱셈·누산 배열과 큰 온칩 메모리를 사용했다. 연구진은 평균 처리량뿐 아니라 사용자 요청의 응답시간 제약을 중요하게 다뤘다. 같은 연산 수라도 오래 묶어 처리해야 효율이 나오는 장치는 빠른 응답이 필요한 서비스에서 충분히 활용되지 않을 수 있었다. 특정 연구의 가속 배율을 모든 신경망과 모든 세대의 장치에 옮길 수 없는 이유다. [2]

정밀도를 낮추면 같은 면적과 전력에서 더 많은 연산을 수행할 수 있다. 그러나 수를 더 거칠게 표현해도 작업의 정확도가 유지돼야 한다. 음성 인식의 오차, 과학 계산의 수치 안정성, 금융 계산의 반올림 요구는 서로 다르다. '초당 연산 수'는 어떤 형식의 연산인지, 얼마의 데이터 이동이 필요한지, 어떤 정확도를 얻는지와 함께 읽어야 의미가 있다. [2] [4]

무어의 법칙이 말하는 것

무어가 1965년에 제시한 전망의 대상은 부품당 비용이 낮아지는 집적 수준에서 **칩 하나에 들어가는 부품 수**였다. 1959~1964년 자료를 바탕으로 대략 12개월마다 두 배가 되는 추세를 1975년까지 연장했다. 1975년에는 새 자료와 회로·공정 발전을 고려해 앞으로의 증가 간격을 약 2년으로 수정했다. 이 관찰은 산업의 연구개발 목표로도 작용했지만 물리적 자연법칙은 아니다. 집적 부품 수가 두 배가 된다고 특정 프로그램의 실행 시간이나 클록 속도가 같은 비율로 바뀌는 것은 아니다. [5]

소자가 작아지고 많아지더라도 전압과 발열, 배선, 메모리, 제조 비용의 제약은 남는다. 그래서 성능 개선은 더 많은 코어, 특화된 연산기, 더 가까운 메모리, 더 나은 알고리즘과 소프트웨어를 함께 요구한다. 무엇을 만들 수 있는가와 무엇을 실제 작업에 경제적으로 사용할 수 있는가 사이에는 여러 층의 설계가 있다. [6] [7] [2]

주

1 Steve Ward, 「Performance Measures」, https://computationstructures.org/notes/performance/notes.html . 위치: §11.1 Latency versus Throughput 및 파이프라인 설명.

2 Jouppi 외, 「In-Datacenter Performance Analysis of a Tensor Processing Unit」, https://arxiv.org/pdf/1704.04760 . 위치: 초록·§1–2, 추론과 응답시간 요구, 곱셈·누산 배열, 온칩 메모리, 정밀도와 데이터 이동.

3 Lawrence Livermore National Laboratory, 「Introduction to Parallel Computing Tutorial」, https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial . 위치: Amdahl's Law, Partitioning, SIMD, Communication overhead. 순차 부분 절반의 2배 상한은 통신 비용을 무시한 법칙의 계산 예시.

4 David Goldberg, 「What Every Computer Scientist Should Know About Floating-Point Arithmetic」, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html . 위치: Rounding Error, Floating-point Formats, Cancellation. 십진수 0.1의 이진 표현과 반올림·상쇄의 설명.

5 Computer History Museum, 「Moore's Law Predicts the Future of Integrated Circuits」, https://www.computerhistory.org/siliconengine/moores-law-predicts-the-future-of-integrated-circuits/ . 위치: 1965년의 연간 배증 전망과 1975년의 2년 수정.

6 Steve Ward, 「CMOS」, https://computationstructures.org/notes/cmos/notes.html . 위치: MOSFET, NFETs and PFETs, Pullup and Pulldown Circuits, CMOS inverter, 전력과 전압의 관계.

7 Bryant·O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2판 6장, https://csapp.cs.cmu.edu/2e/ch6-preview.pdf . 위치: 6장 도입과 §6.1.1 Random Access Memory; 계층의 속도·용량·비용, 지역성, SRAM·DRAM.

9장. 세계 산업과 보급의 경제

컴퓨터 한 대를 만드는 여러 산업

완제품의 상표만으로 생산 지리를 설명하기 어렵다. 프로세서의 구조와 회로를 설계하는 일, 웨이퍼 위에 회로를 제조하는 일, 메모리를 만드는 일, 칩을 포장하고 검사하는 일, 기판과 화면·전원·케이스를 조립하는 일이 분리될 수 있다. 설계가 같아도 제조 공정과 수율, 패키징, 공급 가능한 메모리가 제품의 비용과 생산량에 영향을 준다. [1] [2] [3]

TSMC는 1987년 설립 때부터 고객의 칩을 제조하고 자체 완제품 칩과 경쟁하지 않는 파운드리 모델을 내세웠다. 이는 설계 회사가 자기 공장을 갖추지 않고도 제조 역량을 이용할 수 있는 구조다. 반대로 공장을 운영하는 회사는 여러 고객의 수요를 모아 대규모 설비를 활용한다. 분업은 진입 장벽을 낮추는 부분과 특정 제조 역량에 대한 의존을 높이는 부분을 동시에 갖는다. [1]

네덜란드 ASML의 노광장비는 빛으로 회로 패턴을 웨이퍼에 옮기는 제조 단계에 관여한다. 한국 삼성전자의 반도체 사업에는 DRAM·낸드 등 메모리, 시스템 반도체 설계, 파운드리가 함께 있다. 특히 대량의 데이터를 연산 장치에 공급해야 하는 시스템에서는 메모리의 용량과 대역폭이 설계의 핵심 조건이 된다. 한국의 메모리 산업은 완성 PC의 브랜드 순위와 다른 층에서 컴퓨팅 산업에 참여한다. [2] [3] [4]

이 분업에서는 한 단계의 부족이 다른 단계의 생산능력을 묶을 수 있다. 설계 파일이 있어도 제조·패키징·검사가 모두 가능해야 출하할 수 있고, 칩을 확보해도 전력과 통신 기반시설이 없으면 데이터센터를 가동할 수 없다. 공급망의 회복력은 어느 한 나라가 '컴퓨터를 만들 수 있다'는 문장보다, 무엇을 다른 곳에서 받아야 하고 대체에 얼마나 시간이 필요한지를 묻는 데서 드러난다. [1] [2] [3]

값이 내려가도 모두에게 같은 기회가 오지는 않는다

계산 비용을 긴 시간에 걸쳐 비교하려면 '컴퓨터 한 대'의 가격 대신 어떤 일을 얼마나 처리하는지를 맞춰야 한다. 경제학자 윌리엄 노드하우스는 2001년 연구에서 107개 계산 장치의 자료를 모아 덧셈·기계 명령·후대 벤치마크를 '표준화 연산'이라는 연결 지표로 환산했다. 다음 표는 이 연구의 회귀 추정에서 얻은 **1998년 불변 달러당 계산 능력의 연평균 증가율**이다. 기기 소매가격의 하락률이나 모든 응용의 속도 향상률이 아니다. [5]

기간	불변 달러당 계산 능력의 연평균 증가율
1940–1950	75.7%
1950–1960	52.9%
1960–1970	21.0%
1970–1980	36.6%
1980–1990	88.4%
1990–2001	83.5%

이 수치는 노드하우스의 단일 역사적 재구성이다. 자본 비용에는 연 실질 이자율 10%, 감가상각률 30%, 연 2,000시간 이용이라는 가정을 사용하고 일부 운영비도 포함했다. 서로 다른 시대의 연산과 장치를 연결하므로 단기간의 정밀한 비교보다 비용 개선이 장기적으로 얼마나 컸는지를 보는 데 적합하다. 소프트웨어 품질·화면·통신·사용 편의성까지 하나의 완전한 지표로 포괄하지 못하며, 2001년 이후를 외삽하지 않는다. 그래도 같은 예산으로 가능한 계산이 크게 늘면서 이전에는 경제적이지 않던 용도가 열렸다는 역사를 설명하는 데 도움이 된다. [5]

보급을 볼 때 인터넷 이용률은 중요한 지표지만 컴퓨터 소유율과 같지는 않다. 한 사람이 여러 기기를 쓰기도 하고, 한 기기를 여럿이 공유하기도 한다. 연결돼 있어도 접속 요금·속도·기술·언어·접근성 때문에 이용할 수 있는 서비스가 다르다. 장치를

판매한 수와 사람들이 실제로 계산 자원을 사용할 수 있는 정도를 구분해야 한다. [6] [7]

보급의 결과는 계산 성능의 개선과 별도로 측정해야 한다. 아래 ITU의 2024년 추계는 인구 중 인터넷을 이용하는 사람을 세며 컴퓨터 설치 대수를 세지 않는다. 고소득 국가와 저소득 국가 사이의 격차는 세계적 기술 확산이 곧 같은 이용 기회로 이어지지 않았음을 보여 준다. [6]

지표	대상·기준연도	ITU가 제시한 값
인터넷 이용 인구 비율	세계, 2024년 추계	68%
인터넷 이용 인구	세계, 2024년 추계	약 55억 명
인터넷 비이용 인구	세계, 2024년 추계	약 26억 명
인터넷 이용 인구 비율	고소득 국가, 2024년 추계	93%
인터넷 이용 인구 비율	저소득 국가, 2024년 추계	27%

이 표의 국가 소득군은 해당 ITU 자료가 사용한 세계은행 분류를 따른다. 보급의 차이는 공급 산업의 위치와도 다르다. 반도체를 수출하는 경제와 그 나라 모든 주민이 안정적인 기기와 통신, 교육 기회를 갖는다는 주장은 별도의 측정이 필요하다. [6]

주

1 TSMC, 「Company Profile」, https://www.tsmc.com/english/aboutTSMC/company_profile . 위치: 1987년 설립과 dedicated foundry business model, 제조시설·지원 사무소 구분.

2 ASML, 「About ASML」, https://www.asml.com/en/company/about-asml . 위치: lithography systems, 빛으로 회로 패턴을 인쇄하는 기능과 네덜란드 본사.

3 Samsung Semiconductor, 「Business Area」, https://semiconductor.samsung.com/about-us/business-area/ . 위치: Memory, System LSI, Foundry.

4 Bryant·O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2판 6장, https://csapp.cs.cmu.edu/2e/ch6-preview.pdf . 위치: 6장 도입과 §6.1.1 Random Access Memory; 계층의 속도·용량·비용, 지역성, SRAM·DRAM.

5 William D. Nordhaus. 「The Progress of Computing」. Cowles Foundation Discussion Paper 1324, 2001. https://cowles.yale.edu/sites/default/files/2022-08/d1324.pdf . 위치: 본문 pp.4–6(홀러리스·성능연결), p.16(비용가정), pp.17–19(해석과추정), p.35 Table 4의 1998 prices 열.

6 ITU, *Facts and Figures 2024*, 「Internet use」, https://www.itu.int/itu-d/reports/statistics/2024/11/10/ff24-internet-use/ . 위치: Internet use continues to grow 본문, 주1–3. 2024년 추계와 세계은행 소득군 분류; 해당 판의 수치로 인용.

7 UNESCO, *Global Education Monitoring Report 2023*, https://www.unesco.org/gem-report/en/publication/technology . 위치: 보고서 소개의 다섯 경로와 세 가지 체계적 조건—access to technology, governance regulation, teacher preparation.

10장. 사람의 일, 문화, 책임

일자리는 작업들의 묶음이다

컴퓨터가 수행하기 쉬운 일은 반복적이라는 이유만으로 결정되지 않는다. 규칙과 입력을 명확히 부호화할 수 있는지가 중요하다. 스프레드시트는 반복 계산을 줄이고 사무용 시스템은 기록의 검색과 갱신을 자동화한다. 그러나 고객의 사정을 판단하고, 잘못 정의된 문제를 다시 묻고, 예외를 책임지는 작업이 같은 직무에 함께 들어 있을 수 있다. [1]

경제학자 데이비드 오터는 자동화가 일부 노동을 대체하면서 다른 노동을 보완하고, 생산을 늘려 노동 수요에도 영향을 준다고 설명한다. 이 관점에서는 '기술이 직업을 없애는가'라는 질문을 어떤 작업이 줄고 어떤 작업의 가치가 높아지는가로 나누어 본다. 그의 2015년 논의는 정형화해 명시할 수 있는 작업의 대체와 문제 해결·적응·창의성의 보완을 함께 강조하며, 기술 변화가 일자리의 종류와 임금을 바꾼다고 지적한다. [1]

사무 자동화의 구체적인 변화는 일괄 처리와 온라인 거래의 차이에서도 보인다. 천공카드를 모아 한꺼번에 처리하면 고객이 질문하는 시점과 기록이 갱신되는 시점 사이에 간격이 생긴다. CHM이 설명하는 CICS는 단말을 통해 고객 정보와 거래를 온라인으로 처리하도록 했고, 공공요금 업무에서 금융·보험 등으로 확산됐다. 직원은 다음 처리 묶음을 기다리는 대신 기록을 조회하고 거래 결과를 이용해 고객과 상호작용할 수 있었다. 자동화의 효과는 연산량보다 업무가 진행되는 시점과 순서를 바꾸는 데서도 나타난다. [2]

교육은 기기를 놓는 일보다 크다

BASIC과 LOGO, 이후의 교육용 프로그래밍 환경은 컴퓨터를 정답을 받아 보는 기계뿐 아니라 절차를 만들고 시험하는 도구로 사용하게 했다. 화면에 움직임을 만들려면 목표를 작은 단계로 나누고, 예상과 다른 결과가 나오면 규칙을 고쳐야 한다. 이런 활동은 컴퓨터 과학의 일부를 배울 수 있는 방법이지만 기기를 지급하는 것만으로 자동으로 이루어지지 않는다. [2]

UNESCO의 2023년 교육 보고서는 기술의 잠재력이 실현되려면 접근성, 거버넌스와 규제, 교사 준비라는 조건이 필요하다고 강조한다. 학생마다 연결과 기기의 품질이 다르고, 교사는 도구를 수업 목표와 연결할 시간과 지원이 필요하다. 학습 과정에서 생긴 데이터가 누구에게 수집되고 어떻게 사용되는지도 교육의 조건이다. 따라서 '컴퓨터를 많이 보급했다'와 '학습이 개선됐다'는 다른 성과 지표다. [3]

기록과 창작의 재료가 바뀌다

문서·사진·음악을 같은 디지털 환경에서 수정하고 복사하고 전달할 수 있게 되면서 창작의 작업 과정이 바뀌었다. 탁상출판은 글과 그림의 배치를 화면에서 조정하고 인쇄하는 일을 가까이 연결했다. 컴퓨터 그래픽스와 CAD는 설계 대안을 바꾸고 결과를 비교하는 비용을 줄였다. 변화의 핵심은 컴퓨터가 예술적 판단을 대신한다는 데보다 수정 가능한 표현과 도구가 창작 과정에 들어왔다는 데 있다. [2]

디지털 보존에서는 파일의 바이트와 그 파일을 읽는 환경을 함께 관리해야 한다. Digital Preservation Coalition의 지침은 형식과 버전을 식별하고 보존해야 할 특성을 정한 뒤 이전 전략을 고르도록 한다. 마이그레이션은 오래된 파일을 새 형식으로 변환하지만 모양이나 기능, 메타데이터가 달라질 수 있어 결과 검증이 필요하다. 에뮬레이션은 옛 실행 환경을 재현해 기존 프로그램을 계속 이용하려는 접근이다. 이 지침이 드는 항공우주 CAD 사례에서는 변환한 설계가 동일한지 검증하는 부담 때문에 기존 소프트웨어 환경을 유지하는 편이 유리했다. 보존은 파일을 남기는 일과 그 의미·기능을 다시 사용할 수 있게 하는 일을 연결한다. [4]

인증·권한·암호화와 복구

컴퓨터 보안에는 비밀이 새지 않게 하는 일, 기록이 멋대로 바뀌지 않게 하는 일, 필요할 때 서비스를 사용할 수 있게 하는 일이 포함된다. 프로그램 결함뿐 아니라 계정 권한, 설정, 공급받은 소프트웨어, 운영 절차가 모두 관여한다. 같은 통신 기술이 협업을 가능하게 하면서 공격자가 멀리서 접근할 길도 만들고, 같은 프로그램의 대량 배포가 편익과 결함을 함께 확산시킬 수 있다. [5]

기술적 통제도 목적이 다르다. 인증은 로그인한 사람이 누구인지 확인하고, 권한 검사는 그 사람이 특정 계좌나 파일에 어떤 동작을 할 수 있는지 판정한다. 서버는 요청마다 해당 자원의 권한을 확인해야 하며, 화면에서 버튼을 숨기는 것만으로 접근을 통제할 수 없다. HTTPS의 TLS는 통신 중인 데이터의 기밀성과 무결성을 보호하고 보통 서버의 신원을 인증한다. 로그인에 성공한 이용자의 모든 요청을 허용하거나, 암호화됐다는 이유로 목적지 자체를 신뢰하면 서로 다른 보장을 혼동하게 된다. [6] [7] [8]

피싱은 이 차이를 악용한다. 공격자는 익숙한 기관을 흉내 낸 메시지와 링크로 이용자를 가짜 로그인 화면에 유도해 비밀번호를 받는다. 가짜 사이트와 이용자 사이의 연결이 암호화돼 있어도 비밀번호가 공격자에게 전달되는 문제는 남는다. CISA가 예상하지 못한 요청을 별도로 확보한 연락처로 확인하고, 메시지의 링크 대신 알려진 사이트로 직접 이동하라고 권하는 이유다. [9] [8]

NIST의 사이버보안 프레임워크 2.0은 이런 보호 조치에 거버넌스·식별·탐지·대응·복구를 연결한다. 예를 들어 탈취된 계정의 권한을 제한하고 이상 접근을 발견하고 세션을 폐기한 뒤 피해 데이터를 복구하는 업무에는 기술과 책임자가 모두 필요하다. 보호에 실패한 뒤 무엇을 할지까지 정해야 서비스의 위험을 줄일 수 있다. [5]

컴퓨터가 물리 장치를 제어하면 데이터 오류가 현실의 위험으로 이어질 수 있다. 2010년 알려진 스턱스넷은 산업 제어와 악성코드가 연결될 수 있음을 보여 주었다. 반대로 모든 고장을 공격으로 볼 수도 없다. 전원·열·부품·소프트웨어·사람의 설정이 서로 다른 방식으로 실패한다. 중요한 시스템에서는 정상 동작뿐 아니라 고장과 오작동이 어떤 결과를 만드는지까지 설계해야 한다. [2] [5]

주

1 David H. Autor, 「Why Are There Still So Many Jobs?」, https://www.aeaweb.org/articles?id=10.1257/jep.29.3.3 . 위치: 공개 초록; routine, codifiable tasks의 대체와 문제 해결·적응·창의성의 보완.

2 Computer History Museum, 「Software & Languages」, https://www.computerhistory.org/timeline/software-languages/ . 위치: 1952 A-0, 1957 FORTRAN·FLOW-MATIC, 1960 COBOL, 1961 CTSS, 1963 Sketchpad, 1964 BASIC, 1967 LOGO, 1969 UNIX, 1972 C, 1979 VisiCalc, 1981 MS-DOS, 1983 GNU, 1985 PageMaker, 1991 Linux, 2010 Stuxnet, 2014 Heartbleed.

3 UNESCO, *Global Education Monitoring Report 2023*, https://www.unesco.org/gem-report/en/publication/technology . 위치: 보고서 소개의 다섯 경로와 세 가지 체계적 조건—access to technology, governance regulation, teacher preparation.

4 Digital Preservation Coalition. 「File formats and standards」, Digital Preservation Handbook. https://www.dpconline.org/handbook/technical-solutions-and-tools/file-formats-and-standards . 위치: 포맷 식별·migration 위험과 CAD의 emulation 사례.

5 NIST, *Cybersecurity Framework 2.0*, https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf . 위치: §1–2 및 Appendix A, GOVERN·IDENTIFY·PROTECT·DETECT·RESPOND·RECOVER와 공급망·권한·데이터 보호·복구 항목.

6 MDN. 「Authentication」. https://developer.mozilla.org/en-US/docs/Web/Security/Authentication . 위치: 인증 정의와 세션 관리.

7 OWASP. 「Authorization Cheat Sheet」. https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html . 위치: 인증과권한의구별, Least Privilege, Deny by Default, Validate the Permissions on Every Request.

8 MDN. 「Transport Layer Security」. https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Transport_Layer_Security . 위치: 도입부 Encryption·Integrity·Authentication, Server authentication.

9 CISA. 「Recognize and Report Phishing」. https://www.cisa.gov/secure-our-world/recognize-and-report-phishing . 위치: 가짜 로그인·수상한 요청과 별도 연락처 확인.

11장. 정확한 기계에도 한계가 있다

표현의 한계와 수치 오차

고정된 비트 수로 나타낼 수 있는 정수의 범위는 유한하다. 부호 없는 네 비트는 0부터 15까지 나타낼 수 있다. 15에 1을 더한 결과를 네 비트만 남기는 규칙으로 계산하면 0이 된다. 이것은 덧셈의 수학이 바뀐 것이 아니라 결과를 저장하는 범위와 규칙이 다른 것이다. 프로그램은 범위 초과를 검사하거나 더 넓은 표현을 선택해야 한다. [1] [2]

부동소수점은 유효숫자와 지수를 나누어 매우 큰 수와 작은 수를 제한된 비트로 나타낸다. 그 대신 많은 실수를 정확하게 저장할 수 없다. 십진수 0.1도 이진수에서는 무한 반복이므로 유한한 이진 부동소수점에서는 근삿값으로 저장된다. 연산 결과를 다시 표현 가능한 수로 맞추는 반올림이 뒤따른다. [3]

가까운 두 근삿값을 빼면 공통된 앞자리들이 사라지고 작은 오차가 결과의 큰 부분을 차지할 수 있다. 계산 순서를 바꾸면 반올림이 일어나는 지점이 달라져 결과도 달라질 수 있다. 수치해석은 이런 현상을 고려해 안정적인 계산법을 찾는다. 더 많은 자릿수를 쓰는 것은 한 방법이지만, 문제 자체가 입력의 작은 변화에 민감하면 입력의 정확도와 모형의 조건도 함께 보아야 한다. [3]

과학 계산에서는 검증의 질문도 나뉜다. NASA의 전산유체역학 안내에서 verification은 방정식과 계산법이 프로그램에 올바르게 구현됐는지, validation은 결과가 관측된 물리 현상을 적절히 나타내는지를 묻는다. 격자를 세분했을 때 해가 수렴하는지 확인하는 일과 풍동 실험 결과에 맞는지 비교하는 일은 목적이 다르다. 수치 오차를 줄여도 물리 모형의 가정이 부적절하면 현실 예측은 어긋날 수 있다. [4]

계산할 수 없는 문제

튜링의 계산 모델은 간단한 기호 조작을 정해진 절차로 반복하는 것이 무엇을 뜻하는지 수학적으로 다룬다. 범용 기계는 다른 기계의 기술을 입력받아 그 계산을 모사할 수 있다. 이 통찰은 하나의 장치가 프로그램에 따라 다른 일을 하는 범용성의 바탕을 설명하지만, 모든 질문에 답하는 기계를 보장하지 않는다. [5]

정지문제는 임의의 프로그램과 입력을 받아 그 프로그램이 언젠가 멈출지 항상 맞게 판정하는 절차가 있는지를 묻는다. 있다고 가정하면, 그 판정기가 '멈춘다'고 예측할 때는 무한 반복하고 '멈추지 않는다'고 예측할 때는 멈추는 프로그램을 만들 수 있다. 이 프로그램이 자기 자신에 대한 예측을 사용하게 하면 어느 답도 맞을 수 없다. 따라서 모든 경우에 통하는 일반적인 판정 절차는 존재하지 않는다. [5]

이는 특정 프로그램의 종료를 분석할 수 없다는 뜻이 아니다. 반복 횟수가 정해져 있거나 어떤 값이 매번 감소한다는 사실을 증명하면 종료를 보일 수 있다. 불가능한 것은 모든 프로그램과 입력을 예외 없이 판정하는 만능 도구다. 하드웨어가 빨라져도 이 논리적 장벽은 없어지지 않는다. [5]

너무 오래 걸리는 문제

계산 가능하더라도 현실적인 시간과 메모리 안에서 풀 수 있는지는 별개의 문제다. 입력 크기 n에 비례하는 절차, n^2에 비례하는 절차, 2^n에 비례하는 절차는 입력이 커질 때 전혀 다르게 성장한다. 초당 처리량을 두 배로 높여도 지수적으로 늘어나는 전체 후보를 모두 시험하는 비용은 빠르게 커진다. [6]

복잡도 이론의 P와 NP는 이런 자원 문제를 정교하게 구분한다. 거칠게 말해 P는 다항 시간에 풀 수 있는 결정 문제들이고, NP는 '그렇다'는 답의 증거를 다항 시간에 확인할 수 있는 문제들이다. P와 NP가 같은지는 미해결이다. NP 문제라는 이유만으로 모든 실제 사례가 어렵다거나, 어떤 알고리즘도 빠를 수 없다는 결론을 내릴 수는 없다. 최악의 경우, 입력 구조, 근사해 허용 여부가 중요하다. [6]

실제 NP 문제로, 여러 작업을 두 장비에 나누어 각각 정해진 시간 안에 끝낼 수 있는지 묻는 경우를 생각할 수 있다. 각 작업을 어느 장비에 넣을지를 0 또는 1로 표시하면 용량 제한은 정수 부등식이 된다. 누군가 배치를 제시하면 작업 시간이 제한을 넘는지 더해서 검사할 수 있다. 그러나 맞는 배치를 처음부터 찾는 일은 같은 검산 절차를 한 번 실행하는 것으로 해결되지 않는다. 입력이 커질 때 모든 조합을 확인하는 방법의 비용은 급증한다. 교재의 0/1 정수계획 예시는 이처럼 '해의 증거를 빨리 검사함'과 '해를 빨리 찾아냄'을 구별한다. [6]

실무에서는 가능한 답의 범위를 줄이고, 문제의 특수 구조를 이용하고, 근사나 확률적 방법을 택하고, 일부 보장을 양보하기도 한다. 어느 선택이 적절한지는 문제의 목적에 달렸다. 항공기 안전 검증과 영화의 한 프레임을 만드는 작업이 같은 오차와 실패를 허용하지 않는 것처럼, '빠른 답'의 의미도 과제마다 다르다. [6] [3]

물리적 기계의 한계

회로 전력의 제약보다 더 근본적인 질문은 정보 하나를 지우는 데 최소한의 열이 필요한가다. 처음에 0과 1이 같은 확률로 가능한 비트를 어느 값이었든 0으로 초기화하면, 출력만 보고 원래 값을 복원할 수 없다. 이처럼 두 가능성을 하나로 합치는 동작은 논리적으로 비가역적이다. [7]

온도 T의 열적 환경에서 이런 한 비트를 지울 때 란다우어 한계는 평균적으로 환경에 내보내야 할 열을 **k_B T ln 2**로 준다. k_B는 볼츠만 상수, T는 절대온도이며 식의 단위는 줄(J)이다. 이는 처음부터 값이 확실한 비트를 그대로 두는 경우가 아니라 한 비트의 불확실성을 제거하는 조건이다. 열역학적으로 되돌릴 수 있는 한계에 가깝게, 충분히 느리게 소거할 때 이 최소값에 접근할 수 있고 빠른 실제 과정에는 추가 소산이 생길 수 있다. [7]

2012년 베뤼와 동료들은 이중 우물 퍼텐셜에 갇힌 단일 입자로 한 비트의 기억을 만들고 두 우물에 있을 초기 확률을 각각 절반으로 준비했다. 장벽을 낮추고 한쪽으로 기울여 입자를 모은 뒤 장벽을 되돌리는 소거 과정에서, 충분히 긴 주기일수록 평균 방출 열이 한계에 접근함을 보고했다. 이 식은 모든 논리 게이트의 실제 소비전력을 계산하는 법칙이 아니다. 논리적 정보 손실의 최소 열과 CMOS의 충전·방전·누설 및 주변장치의 에너지 소비는 서로 다른 수준의 설명이다. [7]

주

1 Steve Ward, 「The Digital Abstraction」, https://computationstructures.org/notes/digitalabstraction/notes.html . 위치: §5.1–5.4, 이산 표현·조합 장치·잡음 여유. 이진수 조합과 덧셈은 정의에서 도출한 설명 예시.

2 Steve Ward, 「Instruction Set Architectures」, https://computationstructures.org/notes/isas/notes.html . 위치: CPU Datapath, CPU Control, Instruction Set Architecture as an Abstraction. 본문의 가상 계산 절차는 이 교육용 실행 모델을 일반화한 설명.

3 David Goldberg, 「What Every Computer Scientist Should Know About Floating-Point Arithmetic」, https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html . 위치: Rounding Error, Floating-point Formats, Cancellation. 십진수 0.1의 이진 표현과 반올림·상쇄의 설명.

4 NASA Glenn Research Center. 「Overview of Verification and Validation」. https://www.grc.nasa.gov/www/wind/valid/tutorial/overview.html . 위치: verification/validation 정의, 격자수렴 및 실험 비교.

5 Sedgewick·Wayne, 「Computability」, https://introcs.cs.princeton.edu/java/54computability/ . 위치: §5.4, Turing machines와 The halting problem is unsolvable.

6 Sedgewick·Wayne, 「Intractability」, https://introcs.cs.princeton.edu/java/55intractability/ . 위치: §5.5, Computational complexity, Polynomial-time, NP, P, NP-completeness.

7 Bérut 외, 「Experimental verification of Landauer's principle linking information and thermodynamics」, Nature 483(2012), pp.187–189. https://www.nature.com/articles/nature10872 ; 대학 제공 논문 사본 https://www.physik.uni-kl.de/eggert/papers/raoul.pdf . 위치: p.187의 kTln2, 같은 확률의 초기상태와 소거 프로토콜, pp.188–189의 긴 주기 한계.

12장. 다음 컴퓨터를 판단하는 기준

더 가까운 메모리, 더 특화된 연산

앞으로의 설계에서 유력한 방향 중 하나는 계산과 데이터를 가까이 놓는 것이다. 가속기와 고대역폭 메모리의 결합, 칩 내부와 칩 사이 연결의 개선, 메모리에서 직접 연산하는 방식은 모두 데이터 이동의 부담을 줄이려 한다. 다만 어느 방향이 유리한지는 연산의 형태와 재사용 정도, 장치의 비용과 전력, 소프트웨어가 활용할 수 있는 정도에 달려 있다. [1] [2] [3]

아날로그 인메모리 계산은 이 방향을 다른 물리 방식으로 구현한다. 메모리 소자의 전도도를 가중치에 대응시키고 입력 신호에 따른 전류를 합치면 행렬·벡터 곱의 일부를 직접 수행할 수 있다. 2023년 암브로조와 동료들은 상변화 메모리를 이용한 칩으로 음성 인식 과제를 수행한 결과를 발표했다. 작은 키워드 인식 과제에서는 소프트웨어와 동등한 정확도를, 더 큰 음성 전사 과제에서는 그에 가까운 정확도를 보고했다. [3]

이 연구의 핵심은 숫자 하나의 최대 연산 효율보다 무엇까지 실제로 구현했는가에 있다. 논문은 최종 제품에 필요한 보조 디지털 연산과 데이터 준비의 일부를 칩 밖에서 수행했다고 밝힌다. 가중치를 쓰는 비용과 소자의 내구성, 아날로그 오차, 입력과 출력의 변환도 남는다. 이 비용을 포함한 전체 작업에서 정확도와 에너지 이점이 유지된다면 적용 범위가 넓어질 수 있다. 반대로 보정과 데이터 이동이 이득을 상쇄하면 유리한 과제가 제한될 것이다. [3]

양자는 모든 계산의 지름길이 아니다

양자 상태의 각 성분에는 확률 자체가 아니라 부호와 위상을 가진 **확률진폭**이 붙고, 측정 확률은 그 진폭의 절댓값을 제곱해 얻는다. 여러 계산 경로가 같은 결과로 모일 때 진폭을 더하므로, 같은 위상은 강화하고 반대 위상은 상쇄할 수 있다. 이것이 계산에 쓰이는 간섭이다. 확률처럼 양수만 더하는 것과 달라지는 지점이다. [4]

한 큐비트에 적용하는 아다마르 연산 H는 $|0\rangle$을 $(|0\rangle+|1\rangle)/\sqrt{2}$, $|1\rangle$을 $(|0\rangle-|1\rangle)/\sqrt{2}$라는 중첩으로 바꾼다. 처음 $|0\rangle$에 H를 두 번 적용하면 $|1\rangle$로 끝나는 두 경로의 진폭은 +1/2와 −1/2여서 상쇄되고, $|0\rangle$의 진폭은 더해져 1이 된다. 중간 상태를 측정하지 않았다면 최종 측정에서는 0이 나온다. 중간에서 0이나 1을 측정해 버리면 이 간섭에 필요한 관계가 사라진다. 양자 알고리즘은 이런 진폭의 변환을 조직해 필요한 정보가 드러나게 한다. [4]

구체적인 응용 후보로는 큰 정수의 소인수분해와 양자계의 모사가 있다. NIST가 소개하는 쇼어의 알고리즘은 큰 수를 소인수로 분해하는 양자 계산법이다. 두 소수를 곱하는 것은 쉬워도 그 곱에서 소수를 되찾는 일은 큰 입력에서 어려워진다는 비대칭을 이용하는 RSA 계열의 보안에 중요한 의미를 갖는다. 그러나 이런 알고리즘을 큰 문제에 실행하려면 오랜 계산 동안 오류를 억제해야 한다. 작은 장치에서 계산 원리를 보이는 것과 현실의 암호 크기에 도달하는 것은 다른 공학적 과제다. [5]

분자나 물질의 양자 상태를 모사하는 일도 대상 자체의 양자적 구조를 계산에 활용하려는 경로다. 초전도 회로·이온·중성 원자 등 여러 구현에서 상태를 제어하고 읽는 방법을 연구하지만, 외부 잡음과 불완전한 조작이 계산을 흐트러뜨린다. 입력 준비와 측정까지 포함해 같은 정확도의 고전적 계산과 비교해야 유용성을 판단할 수 있다. [5]

판단 기준은 큐비트의 개수만이 아니다. 어떤 문제를 어떤 정확도로 풀었는지, 전체 과정이 얼마나 걸렸는지, 같은 문제의 고전적 최선 방법과 어떻게 비교되는지를 보아야 한다. 오류 억제와 규모 확장이 동시에 가능해지고 유리한 응용이 확인되면 고전 컴퓨터를 보완하는 계산 자원으로 자리 잡을 수 있다. 비교 대상의 알고리즘이 개선돼 이점이 사라진다면 해당 과제의 우위 주장은 다시 평가해야 한다. [5]

환경은 설계 바깥의 문제가 아니다

컴퓨터는 제조부터 사용과 폐기까지 자원을 소비한다. 칩의 전력 효율이 좋아져도 사용하는 기기와 계산이 늘면 전체 부담은 다른 방향으로 움직일 수 있다. 유지보수와 수리 가능성, 소프트웨어 지원 기간, 재사용과 회수 체계는 장치의 유효 수명에 영향을 준다. 새 칩의 효율만으로 컴퓨팅의 환경 효과를 판단할 수 없는 이유다. [6] [7]

ITU와 UNITAR의 『Global E-waste Monitor 2024』는 2022년 세계 전기·전자폐기물이 약 6,200만 톤 발생했고, 그중 정식 수거·재활용으로 문서화된 비율이 22.3%라고 보고했다. 이 수치는 컴퓨터만의 폐기물이 아니라 플러그나 배터리를 사용하는 전기·전자제품 전체를 포함한다. 또한 문서화되지 않은 나머지가 모두 같은 경로로 처리됐다는 뜻도 아니다. 컴퓨터의 환경 문제를 더 넓은 전자제품 생산·수리·폐기 체계 속에 놓게 하는 지표다. [6]

다음 컴퓨터를 평가할 때에는 더 빠른가만큼 같은 일을 얼마나 적은 에너지와 자원으로 수행하는가, 얼마나 오래 안전하게 쓸 수 있는가, 필요한 사람에게 접근 가능한가를 물어야 한다. 계산의 역사에서 큰 변화는 회로 하나의 성능만으로 일어나지 않았다. 표현과 절차, 제작과 소프트웨어, 제도와 사용이 연결될 때 새로운 일이 가능해졌다. 미래의 변화도 그 연결이 실제로 작동하는지를 통해 판단할 수 있다. [7] [8] [9] [6]

주

1 Bryant·O'Hallaron, *Computer Systems: A Programmer's Perspective*, 2판 6장, https://csapp.cs.cmu.edu/2e/ch6-preview.pdf . 위치: 6장 도입과 §6.1.1 Random Access Memory; 계층의 속도·용량·비용, 지역성, SRAM·DRAM.

2 Jouppi 외, 「In-Datacenter Performance Analysis of a Tensor Processing Unit」, https://arxiv.org/pdf/1704.04760 . 위치: 초록·§1–2, 추론과 응답시간 요구, 곱셈·누산 배열, 온칩 메모리, 정밀도와 데이터 이동.

3 Ambrogio 외, 「An analog-AI chip for energy-efficient speech recognition and transcription」, https://www.nature.com/articles/s41586-023-06337-5 . 위치: Abstract, Main, Chip architecture; 본문 중 보조 디지털 코어와 데이터 준비를 칩 외부에서 처리했다는 한정 및 메모리 기록의 내구성·에너지 조건.

4 John Watrous. 「Quantum information」, IBM Quantum Learning. https://quantum.cloud.ibm.com/learning/en/courses/basics-of-quantum-information/single-systems/quantum-information . 위치: Quantum state vectors, Measurements, Unitary operations의 Hadamard와 H^2 예시.

5 NIST, 「Quantum Computing Explained」, https://www.nist.gov/quantum-information-science/quantum-computing-explained . 위치: 중첩과 측정, 활용, 오류와 큐비트 구현, Current quantum computers are being used mainly… 대목.

6 ITU·UNITAR, *The Global E-waste Monitor 2024*, https://ewastemonitor.info/the-global-e-waste-monitor-2024/ . 위치: 보고서 공식 발표의 2022년 62 million tonnes·22.3% 및 수리·수명·관리 기반시설 요인. 컴퓨터 외 전기·전자제품을 포함하는 범위.

7 Steve Ward, 「CMOS」, https://computationstructures.org/notes/cmos/notes.html . 위치: MOSFET, NFETs and PFETs, Pullup and Pulldown Circuits, CMOS inverter, 전력과 전압의 관계.

8 Mell·Grance, *The NIST Definition of Cloud Computing*, SP 800-145, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf . 위치: 본문 pp.2–3, Essential Characteristics·Service Models·Deployment Models.

9 UNESCO, *Global Education Monitoring Report 2023*, https://www.unesco.org/gem-report/en/publication/technology . 위치: 보고서 소개의 다섯 경로와 세 가지 체계적 조건—access to technology, governance regulation, teacher preparation.

참고문헌

역사와 보급

- B. Jack Copeland. 「The Modern History of Computing」. *Stanford Encyclopedia of Philosophy*, 2000; 2006년 개정. https://plato.stanford.edu/entries/computing-history/
- Computer History Museum. *Timeline of Computer History*: 「Computers」, 「Software & Languages」, 「Memory & Storage」. https://www.computerhistory.org/timeline/computers/ · https://www.computerhistory.org/timeline/software-languages/ · https://www.computerhistory.org/timeline/memory-storage/
- Science Museum. 「Charles Babbage's Difference Engines and the Science Museum」. https://www.sciencemuseum.org.uk/objects-and-stories/charles-babbages-difference-engines-and-science-museum
- ENIAC Programmers Project. 프로젝트 소개와 구술사 수집 소개. https://eniacprogrammers.org/
- Computer History Museum. *The Silicon Engine*: 「Metal Oxide Semiconductor (MOS) Transistor Demonstrated」, 「Microprocessor Integrates CPU Function onto a Single Chip」, 「Moore's Law Predicts the Future of Integrated Circuits」. https://www.computerhistory.org/siliconengine/metal-oxide-semiconductor-mos-transistor-demonstrated/ · https://www.computerhistory.org/siliconengine/microprocessor-integrates-cpu-function-onto-a-single-chip/ · https://www.computerhistory.org/siliconengine/moores-law-predicts-the-future-of-integrated-circuits/
- IBM. 「System/360」. https://www.ibm.com/history/system-360

그림 출처

- Geni. 「Babbage Difference Engine」. Wikimedia Commons. 런던 과학박물관이 후대 제작한 차분기관 2호 사진. https://commons.wikimedia.org/wiki/File:Babbage_Difference_Engine.jpg . 사진 라이선스: CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/). 원사진을 내용 변경 없이 사용.

원리와 소프트웨어

- Steve Ward. *Computation Structures*. MIT 교육자료. 「The Digital Abstraction」, 「CMOS」, 「Instruction Set Architectures」, 「Performance Measures」, 「Compiled Languages」. https://computationstructures.org/notes/digitalabstraction/notes.html · https://computationstructures.org/notes/cmos/notes.html · https://computationstructures.org/notes/isas/notes.html · https://computationstructures.org/notes/performance/notes.html · https://computationstructures.org/notes/compiled/notes.html
- Randal E. Bryant·David R. O'Hallaron. *Computer Systems: A Programmer's Perspective*, 2판, 6장 미리보기. https://csapp.cs.cmu.edu/2e/ch6-preview.pdf
- Remzi H. Arpaci-Dusseau·Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. 「Introduction to Operating Systems」, 「I/O Devices」, 「Distributed Systems」. https://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf · https://pages.cs.wisc.edu/~remzi/OSTEP/file-devices.pdf · https://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf
- University of Canterbury, Computer Science Education Research Group. *Computer Science Field Guide*. 「Searching」, 「How does the computer process your program?」. https://www.csfieldguide.org.nz/en/chapters/algorithms/searching/ · https://www.csfieldguide.org.nz/en/chapters/programming-languages/how-does-the-computer-process-your-program/

- Unicode Consortium. 「What is Unicode?」. https://www.unicode.org/standard/WhatIsUnicode.html
- David Goldberg. 「What Every Computer Scientist Should Know About Floating-Point Arithmetic」. *ACM Computing Surveys*, 1991; Oracle 허가 재판. https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
- Robert Sedgewick·Kevin Wayne. *Introduction to Programming in Java*. §5.4 「Computability」, §5.5 「Intractability」. https://introcs.cs.princeton.edu/java/54computability/ · https://introcs.cs.princeton.edu/java/55intractability/

연결, 산업과 사회

- Barry M. Leiner 외. 「A Brief History of the Internet」. Internet Society, 1997. https://www.internetsociety.org/internet/history-internet/brief-history-internet/
- W. Eddy 편. *RFC 9293: Transmission Control Protocol*. IETF, 2022. https://www.rfc-editor.org/rfc/rfc9293.html
- Lawrence Livermore National Laboratory. 「Introduction to Parallel Computing Tutorial」. https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
- PostgreSQL Global Development Group. 「Transactions」. https://www.postgresql.org/docs/current/tutorial-transactions.html
- MIT PDOS. 「VMware FT」 강의노트. https://pdos.csail.mit.edu/6.824/notes/l-vm-ft.txt
- Peter Mell·Timothy Grance. *The NIST Definition of Cloud Computing*, SP 800-145, 2011. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
- TSMC. 「Company Profile」. https://www.tsmc.com/english/aboutTSMC/company_profile
- ASML. 「About ASML」. https://www.asml.com/en/company/about-asml
- Samsung Semiconductor. 「Business Area」. https://semiconductor.samsung.com/about-us/business-area/
- ITU. *Facts and Figures 2024*: 「Internet use」. https://www.itu.int/itu-d/reports/statistics/2024/11/10/ff24-internet-use/
- David H. Autor. 「Why Are There Still So Many Jobs? The History and Future of Workplace Automation」. *Journal of Economic Perspectives* 29(3), 2015, pp.3–30. https://www.aeaweb.org/articles?id=10.1257/jep.29.3.3
- UNESCO. *Global Education Monitoring Report 2023: Technology in education—A tool on whose terms?* https://www.unesco.org/gem-report/en/publication/technology
- NIST. *The NIST Cybersecurity Framework (CSF) 2.0*, 2024. https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.29.pdf
- ITU·UNITAR. *The Global E-waste Monitor 2024*. https://ewastemonitor.info/the-global-e-waste-monitor-2024/

연구 전선과 물리적 한계

- Norman P. Jouppi 외. 「In-Datacenter Performance Analysis of a Tensor Processing Unit」. ISCA, 2017. https://arxiv.org/pdf/1704.04760

- Stefano Ambrogio 외. 「An analog-AI chip for energy-efficient speech recognition and transcription」. *Nature* 620, 2023, pp.768–775. https://www.nature.com/articles/s41586-023-06337-5
- Antoine Bérut 외. 「Experimental verification of Landauer's principle linking information and thermodynamics」. *Nature* 483, 2012, pp.187–189. https://www.nature.com/articles/nature10872
- NIST. 「Quantum Computing Explained」. https://www.nist.gov/quantum-information-science/quantum-computing-explained

보강 참고문헌

- IBM. 「The punched card」. https://www.ibm.com/history/punched-card
- William D. Nordhaus. 「The Progress of Computing」. Cowles Foundation Discussion Paper 1324, 2001. https://cowles.yale.edu/sites/default/files/2022-08/d1324.pdf
- PostgreSQL Global Development Group. 「Transaction Isolation」, PostgreSQL 18. https://www.postgresql.org/docs/18/transaction-iso.html
- Arpaci-Dusseau·Arpaci-Dusseau. 「Crash Consistency: FSCK and Journaling」, OSTEP 42장. https://pages.cs.wisc.edu/~remzi/OSTEP/file-journaling.pdf
- Digital Preservation Coalition. 「File formats and standards」, Digital Preservation Handbook. https://www.dpconline.org/handbook/technical-solutions-and-tools/file-formats-and-standards
- MDN. 「Authentication」. https://developer.mozilla.org/en-US/docs/Web/Security/Authentication
- OWASP. 「Authorization Cheat Sheet」. https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- MDN. 「Transport Layer Security」. https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Transport_Layer_Security
- CISA. 「Recognize and Report Phishing」. https://www.cisa.gov/secure-our-world/recognize-and-report-phishing
- NASA Glenn Research Center. 「Overview of Verification and Validation」. https://www.grc.nasa.gov/www/wind/valid/tutorial/overview.html
- John Watrous. 「Quantum information」, IBM Quantum Learning. https://quantum.cloud.ibm.com/learning/en/courses/basics-of-quantum-information/single-systems/quantum-information
- Tim Berners-Lee. 「Information Management: A Proposal」, 1989. https://www.w3.org/History/1989/proposal.html
- W3C. 「A Little History of the World Wide Web」. https://www.w3.org/History.html
- MDN. 「How the web works」. https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works