RAD Lab

The Origins of Software

RAD Lab

Disclaimers

- I wasn’t there when it happened
- Not an exhaustive survey of computing *history*
- Ideas and terms we’ll use to describe these events are applied in retrospect
- Your mileage may vary
- Organization
  - Key intellectual concepts
  - Influential people & artifacts (hard to separate!)
  - Wider impact — commercial, social, intellectual

RAD Lab

What is software?

RAD Lab

What is software?

- Software is information
- Software is a machine

- symbolic representation of some task to be performed by a physical device

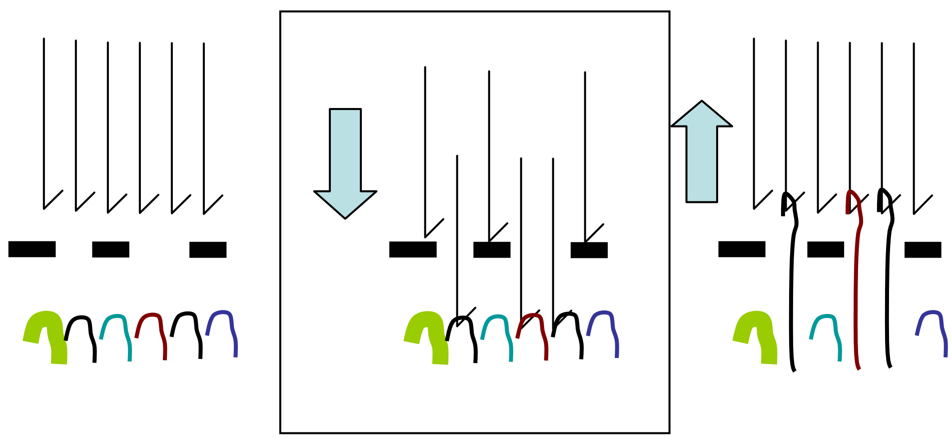
- ...implies a vocabulary—but what are the elements of the vocabulary?

RAD Lab

Jacquard loom (1804)

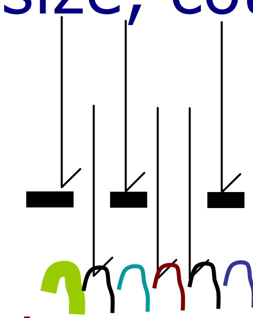
- Different threads attached to different spools
- Hooks drop down, “catch” and pull thread thru hole in card
- No hole in card => hook is blocked and no weave occurs

Edge-on schematic view of card:



3 aspects of software

- *Logical structure:* the pattern of holes in the card "describe" what the finished textile looks like
- *Representation:* if we knew the card size, could encode a weave as a binary string
  - *Here is* 010110
  - Why would we need to know card dimensions?

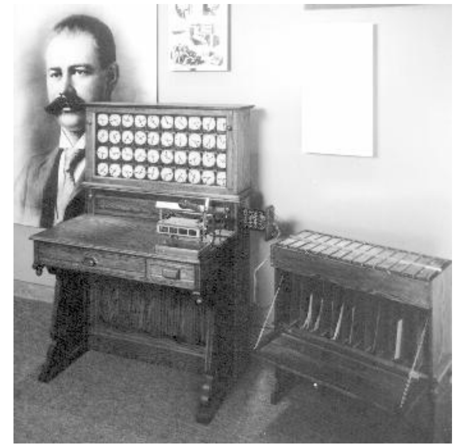


- *Relationship to structure of physical device*
  - Positioning of holes == positioning of weaving hooks
  - Speed of feeding cards == speed of moving shuttle
  - *Card is useless without knowing machine geometry, how different thread spools are ordered, etc.*
  - Analogy: records/record players, CD's/cd players...

Evolution of software loosened these associations.

RAD Lab

Software is how you tell the device what to do

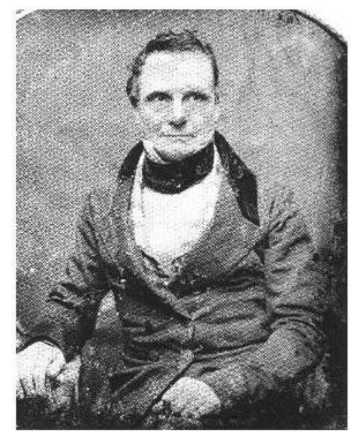


- A self-contained representation of “instructions” for a machine designed to follow them
- pre-ENIAC: special purpose devices, “software” mirrors physical organization
  - Jacquard loom, 1850 Hollerith Census machine, mechanical calculators
- c. ENIAC: concept of *logical organization* of device begins to predominate

- post-ENIAC: *assembly language*
  - physical configuration ▪ invisible to programmer
  - but assembly language constructs still mirror hardware organization
- Fully modern software: largely *independent* of hardware
  - Quasi-human-readable representation
  - Rely on *compilers* and *interpreters* to bridge gap to assembly language

Babbage, Lovelace & the Analytical Engine (~1837)

- Precursor: Difference Engine
  - Computes polynomials (for ballistics calculations) using "method of differences", which requires no multiplying or dividing
  - First Gov't (military) grant for computer research, budget overrun, unfinished project
  - Essentially a fixed-purpose calculator
- Analytical Engine: programmable calculator
  - "Instruction cards" and "variable cards"
  - "Mill" (CPU) and "store" (memory)
  - Instructions: Load, Store, arithmetic ops, conditional, forward/backward jump (skip forward/backward in card reader), subroutine—all elements of modern computers
  - Never built until



RAD Lab

Ada Lovelace, the first programmer (1815-1852)

- Brilliant and mathematically precocious daughter of (divorced) Lord Byron
  - attended society "salons" due to her social status as the "Countess of Lovelace"
  - became Babbage's protégé after becoming fascinated with Difference Engine at his salon
- One of the few who understood AE's potential
  - Devised Analytical Engine procedure for computing Bernoulli numbers
  - Likely the world's first computer program
- Recognized the possibility for *symbolic computation* at a time when few even understood what that meant
  - (It means AI, graphics, MP3 playback, text processing, Web search, ...)
- Reward: an ill-regarded language named for her

RAD Lab

The Legacy of the Calculator

- Dates back thru Zuse, Babbage, Pascal, etc.
  - 1645: Blaise Pascal constructs first true mechanical calculator
  - Reward: an ill-regarded language named for him
- Military has always been driving force
  - Solving ballistics equations requires evaluating nontrivial polynomials, taking square roots, etc.
- "Differential analyzers" and other *analog electromechanical* calculators were current trend
  - Based on physical properties of capacitive and inductive electrical elements
- Idea of a digital computer bucked that trend
- "Software" = plan for doing a complex calcuation

RAD Lab

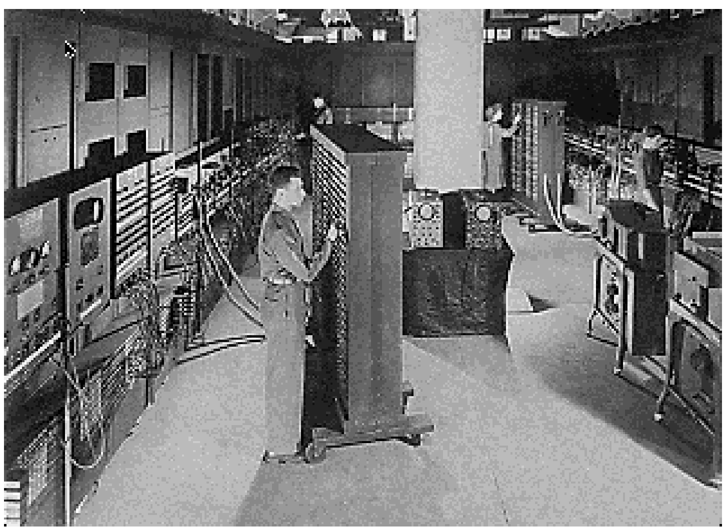
ENIAC (1945)

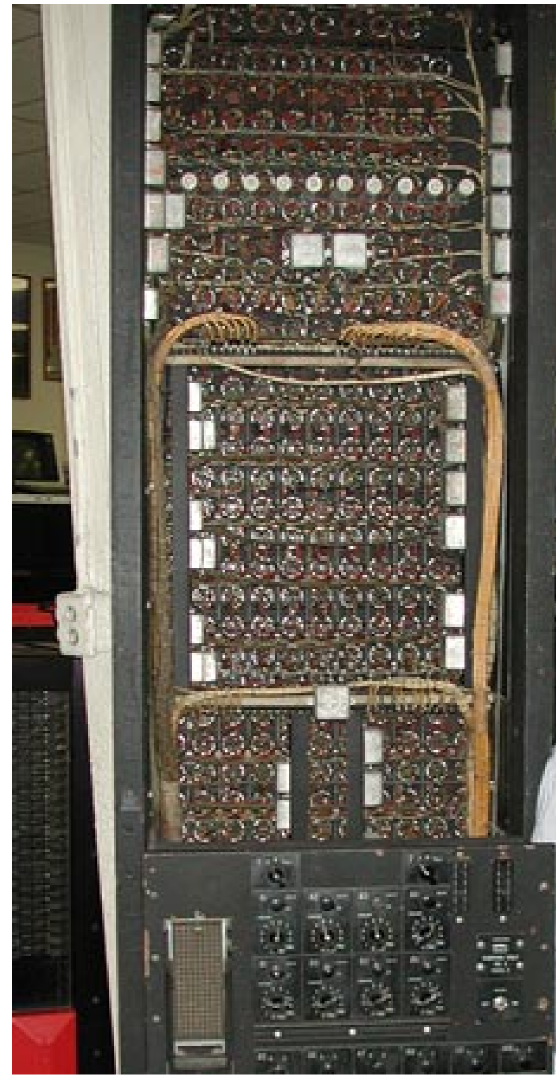
- *Electronical Numerical Integrator And Calculator* built for US military at Moore School of Engineering, UPenn
- Electronic vacuum-tube reimplementation of sequenceable calculator
  - Functional units: multiplier, divider, square root
  - 20 Accumulators, each can hold 10-digit 10's-complement number (about 4.3 bytes, so <100 bytes total)
  - Constant transmitter (from dials or punch cards)
  - Cycling unit (clock)
- in terms of *programmability*, arguably less flexible than the Analytical Engine!

RAD Lab

ENIAC: built 1937-1945, decommissioned 1955

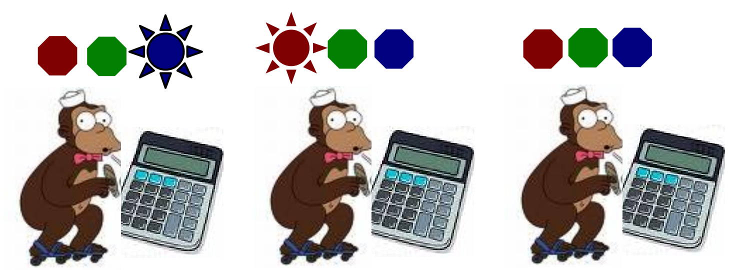
- 42 panels, each 9′x2′x1′, ~200 tons
- Housed in rare forced-air-cooled building
- 19,000+ vacuum tubes, 1500 relays
- 3,000 input switches
- Manual cabling - setup could take days
- Addition cycle 0.2ms (5 KHz), 1000x faster than Differential Analyzer





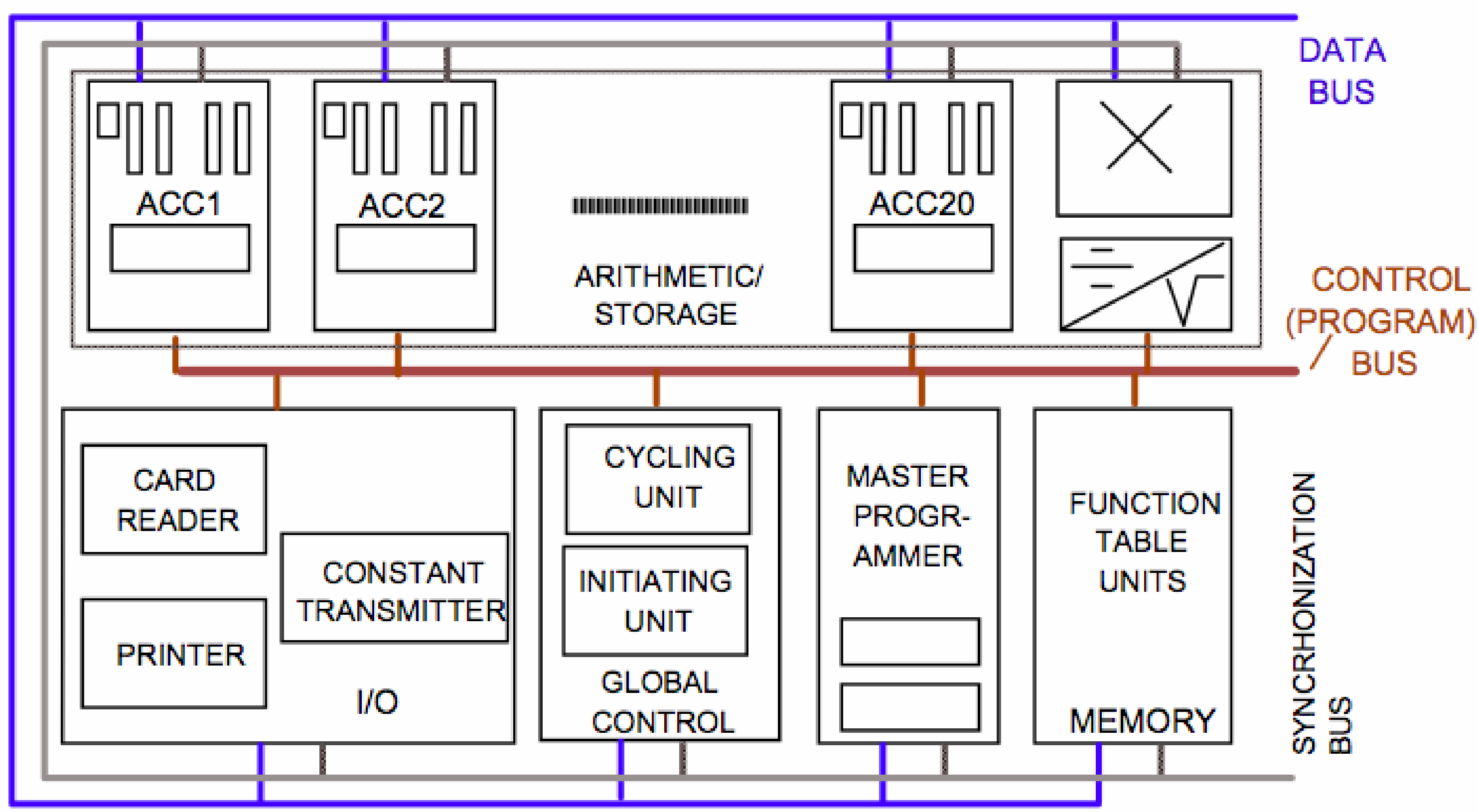
What would ENIAC "software" be like?

- Consider 3 trained monkeys with calculators
  - 2 can only add/subtract; 3rd can also multiply, $\div$, $\sqrt{\ }$
- Each monkey looks at a colored lamp to tell him what to do:
  - Show his calculator screen to another specific monkey
  - Add number shown to him to number on his screen
  - Replace his number with number written on blackboard
- Goal: compute $x = (-b + \sqrt{b^2 - 4ac})/2a$
- Deliverable: step-by-step list of lamp-lightings and what goes on blackboard at each step



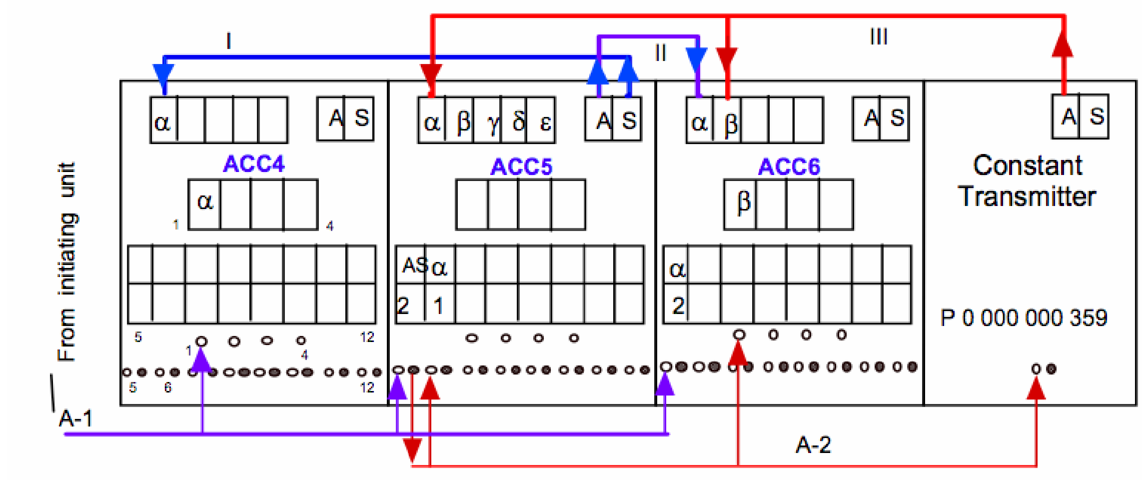
"Programming" ENIAC

- Data bus is not really a bus—just a cable tray
- True parallelism: VLIW From Hell



"Programming" ENIAC, #2

- Ex: compute (a–b), (b+359), (c+2b+359)



1. A4←add 10's comp. of A5 (A4=a-b)
   A6←A5 on α (A6=c+b)
2. A6←A5 on α, since A5 RepCount=2 (A6+=b)
3. A5←359 on β connector (A5+=359)
   A6←359 on β connector (A6+=359)
4. End state: A4=a-b, A5=b+359, A6=c+2b+359

ENIAC's contributions & significance

- Noteworthy...
  - True parallel addition (think: the VLIW From Hell)
  - Historical origin of "accumulator"—reflects calculator legacy
- Easy to see leap to data-bus-based architecture with true microcode
  - ENIAC: connect specific inputs to outputs with hardwired cables; different for each problem to be solved
  - true data bus: output from any unit available to all; operation being performed selects which one reads
- MIT Whirlwind computer and Mark I calculator
  - punch cards would be used to do this selection: holes in cards route outputs to inputs and select operations
  - *hole patterns on card can be interpreted as data*

RAD Lab

Ramifications of machine language concept

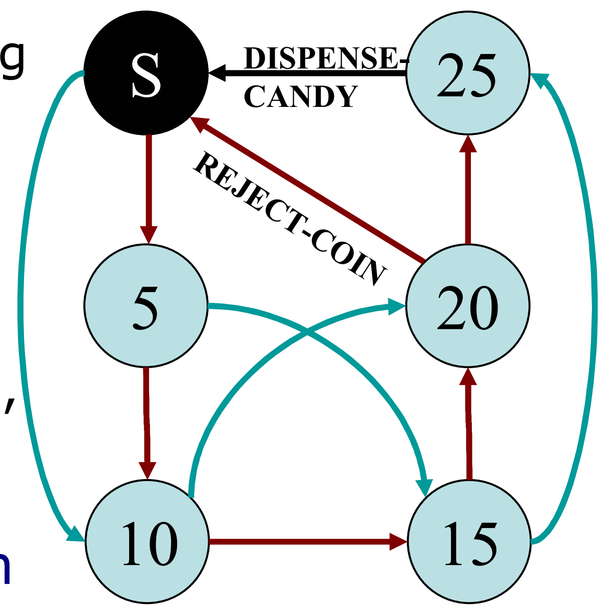
Deep insight: *Programs Are Data*

- Sequences of 1's and 0's to activate machine elements <=> binary representations of numbers
- **Practical implication:** program to be executed can be *stored* in the same medium as the data on which it operates ("stored-program computer")
  - John Von Neumann unfairly credited with idea
- **Non-obvious but deep implication:** program *itself* can be operated on like data

RAD Lab

Alan Turing: A formal model of computation (1936)

- *Turing machine:* "essence" of computing
- Easy-to-understand version: *Finite state machine*
  - Machine's next behavior depends only on *current state* and *current inputs.*
  - Example: 6-state FSM for 25-cent vending machine that takes **nickels** and **dimes**
- Slightly harder to understand version:
  - Infinite paper tape divided into cells holding one symbol each
  - **Head** examines one cell at a time and can move left/right
  - Table of instructions: "If in state X, and symbol under tape is ∑ , erase/write a symbol [on the tape], move Left (or Right), and enter state Y."
- Machine definition is a finite-length list of tuples <*X, ∑, Write, Move, Next-state*> that can be represented numerically



RAD Lab

Implications: Computability Theory and Universality

- Programs as data: can subject them to formal manipulation and analysis
- Famous result: *Universality, Turing completeness*
  - *Given* a description (transcribed to “paper tape”) of a particular turing machine *M*,
  - *one can construct* a “universal” Turing machine *UTM* that can read that tape and *behave exactly as M would.*
- *Practical importance:* physical computer with properties of a UTM is *just as powerful* (in a theoretical sense) as any other computer
- A deep and revolutionary result we now take for granted!
  - Practical result: *compilers and interpreters*
  - Practical result*: emulators and simulators* (eg Apple ~1997)

RAD Lab — Grace Murray Hopper, the Mark I "compiler", and subroutines (1944)

- Navy officer (eventually rear adm.) & math professor, visiting Prof. Howard Aiken's Harvard Computation Lab
  - Mark I & III computers developed for US Military
  - "Programming" == punch a row of 24 holes in paper tape to represent one machine instruction
  - First *automatic* computer, but not stored-program
- Hopper's insight: keep library of tapes of commonly-used "subtasks" (eg square root)
  - But each time used, have to change argument values, what to do with the result, etc.
  - Idea: a *program* to automatically compile paper tape of complete procedure, "splicing in" subtasks as needed
  - Modern (re)birth of the *subroutine* concept; would be absent from original FORTRAN!
- Eventually became A-0 "compiler" for Univac 1 (1952; photo c.1962)

RAD Lab

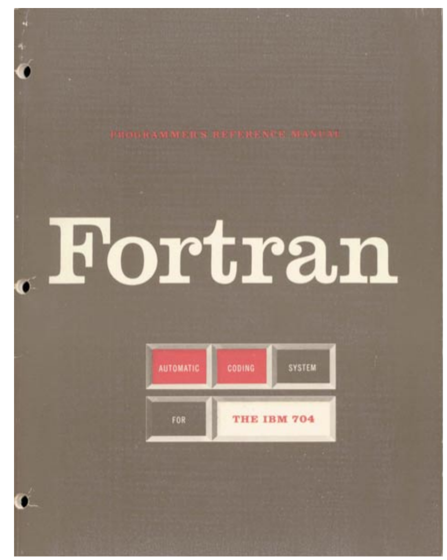
Laning & Zierler: MIT Whirlwind "Interpretive Program" (1954)

- Input: algebraic expressions punched onto cards
- Output: machine-language program to do the computation
  - Where to put intermediate results
  - How to "schedule" computation of intermediate results
  - This would've been ENIAC's assembler, if it had one!
- Probably the first *assembler*
  - Origin: "assembling" a deck of cards from subroutines, constants, etc.
  - Vocabulary of what to do is still tied to machine hardware
  - But "housekeeping" tasks managed automatically
  - Likely forerunner of modern compilers
- "source" and "object" code not stored in same memory—"programming" still seen as separate from "computing"
- First complaints by "real programmers" that compiler-generated code is much worse than hand-tuned assembly

John Backus, FORTRAN, and the IBM 704 (1957)

```
PROGRAM HYPOTENUSE
REAL X,Y,Z, T1
PRINT *,"ENTER X and Y VALUES:"
READ *, X,Y
IF (X.EQ.0  .OR.  Y.EQ.0) THEN
     PRINT *, "X,Y MUST BE NON-ZERO"
  ELSE
     T1 = X**2 + Y**2
     Z = T1**0.5
     PRINT *, "HYPOTENUSE IS:", Z
  END IF
END
```

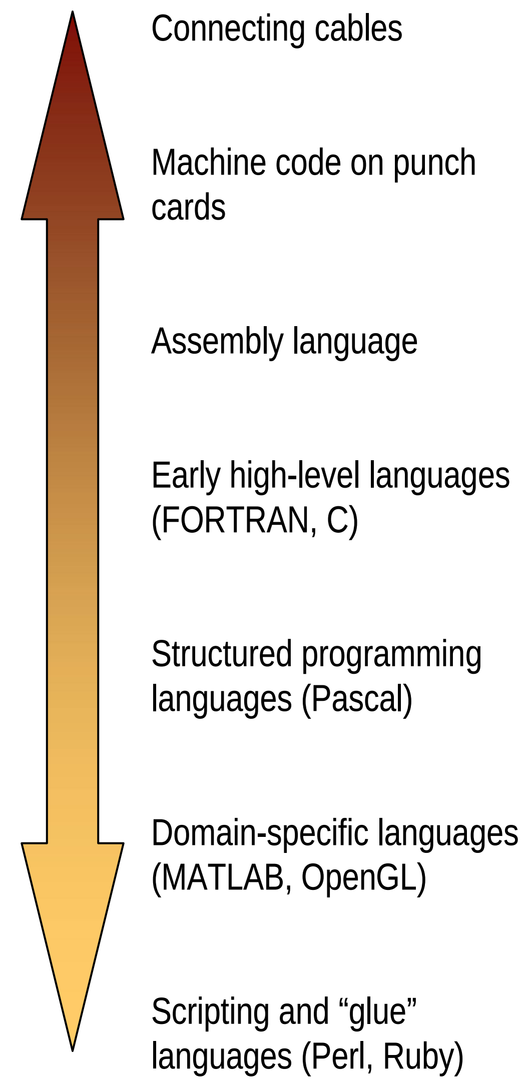
- Resembles algebra, hides physical implementation
  - *Immediate* hit
  - Industry realization: users want to do work, not futz with artifacts
  - Not clear if this has sunk in
- Developed by IBM for use on its pioneering 704 computer
  - Among first to have *floating point* hardware
  - Computer, language & compiler co-designed by John Backus to exploit this => *fast*
- Compiler is itself a machine code program on cards!



RAD Lab

Terminology: Low Level, High Level

- By 1957, modern languages had begun to evolve
  - 1937: ENIAC programming is physical reconfiguration
  - 1950: Whirlwind programming converts algebra equations to machine instructions
  - 1957: FORTRAN expresses *task to be done* with no reference to physical machine
- Next big revolutions:
  - technology: integrated circuits
  - *research & business models* resulting from "unbundling" of software



RAD Lab

Fred Brooks, IBM System/360, and compatibility

- "Architecture": IBM's new term for 360 approach
  - *Assembly language* used by programmers reflected only *logical* machine organization
  - *Microcode* (different for each model) implemented assembly instruction in terms of physical circuits
  - Input & output circuitry standardized "channel" circuitry
  - Result: Buy any 360 model, upgrade later, your programs and I/O peripherals will still work!
- First step in the total decoupling of HW & SW
  - Intel/Microsoft strategy ~30 years later
- Fred Brooks (principal architect of OS/360): first "hard lessons" from a gargantuan software project, *The Mythical Man-Month*

RAD Lab

Ken Olsen, Digital Equipment Corp., and the PDP-8 (1965)

- DEC:First startup to recruit new college grads (MIT)
- Many **important firsts** of PDP series (esp. PDP-8)**:**
  - First minicomputer: size, packaging, cost (~$120K), and use model—users, not operators
  - [Geek] First commercial DMA: fast I/O at fraction of IBM price
  - [Geek] First use of indirect addressing & paging to extend address space while keeping native instruction size small
- First open API's
  - to compete with IBM, DEC encouraged its customers and prospects to learn about, modify, and play with their system
  - Simple architecture—could be quickly understood by an assembly-language programmer
  - Trivia: used for first computer-controlled lighting (*A Chorus Line,* 1975) and BART info displays (1972)
- No real engineering breakthrough, but a massive cultural shift..."a hacker-friendly computer"

Ken Thompson, Dennis Ritchie, Brian Kernighan: Unix & C (1971)

- Unix: a "simple" operating system originally developed for PDP-7 (the Ford Escort of minicomputers)
  - name alludes to MIT MULTICS, pioneering "timeshare" system
  - 1st ed. 1971; for text processing of patent documents with *roff*
- C: a compact and modest programming language
  - Provides high-level language constructs (looping, subroutines, simple data structures, etc.)
  - but doesn't hide machine-level structures
- Most of Unix rewritten in C ~1973: *first source portable OS*
  - Berkeley Software Distribution (BSD) ~1975: AT&T-contested parts rewritten from scratch, ported to VAX, available free
  - Unix+C+VAX (PDP-8 successor) swept research community
  - 1982: Sun decides to base workstation business on Unix
- source portability and C compiler now taken for granted (*gcc*)
  - Linux: widest open-source manifestation of this trajectory

RAD Lab

Gates, Allen, Roberts, the MITS Altair, and Micro-Soft [sic] BASIC

- MITS Altair - first "hobbyist" computer kit, offered in *Popular Electronics* for $395, **sold like crazy**
  - But you couldn't do anything with it: no I/O devices, programming was all in binary (Intel 8080)
- Gates & Allen saw an opportunity: BASIC language
  - created in 1964 at Dartmouth for teaching programming
  - Gates & Allen founded "Micro-Soft" and created a version of BASIC for the Altair
  - Later licensed BASIC for TRS-80, Apple II, and many others
- Big loser: Gary Kildall, inventor of CP/M
  - Turned down IBM; Microsoft got contract, bought QDOS for $175K, repackaged as MS-DOS
  - Kildall thought people would pay more for a better product
  - Windows (direct descendant of QDOS) now runs 90+% PC's
- Would be repeated with Apple's Macintosh & John Scully

RAD Lab

Impact: software as information vs. as machine

- Unbundling of software and backward compatibility
  - Unheard-of before IBM S/360; impractical before PDP-8
  - Result: customer investment is mostly software: licensing, training, support organization, etc.
  - The entire business model of Intel/Microsoft
- Breaking away from the "priesthood" model: BSD+VAX
  - Before DEC & BSD, IBM owned the software/computer industry
  - today, >2/3 of Web servers rely on Open Source software, the spiritual descendant of PDP-8/BSD Unix
- Moore's Law (computer speeds double every 18 months) makes very-high-level languages affordable
  - Compilers no longer slow
  - Interpreters no longer slow
  - Languages can focus on being easier to learn: each language element does a lot more computing work
  - Everyday examples: Excel macros, MATLAB, Visual Basic

Impact: Software as functionality (vs. hardware)

- What kind of intellectual property is software?
  - Source code is like a book → copyright
  - Software directs the operation of a machine → patent
  - Software can be tweaked and incrementally modified → derivative work
- If I develop a new algorithm...
  - it's patentable if I implement it directly in silicon (ENIAC-style)
  - it's copyrightable if I publish the source code
  - it's a mess if I claim its "look & feel" is protectable
  - what if it implements a "business method", like Amazon 1-click™©® purchasing?
- Has spawned a whole subfield of innovation-stifling litigation

Impact: software as abstract representation

- Turing's formalisms made it meaningful to talk about *computer science* as distinct from *electrical engineering, programming, etc.*
  - Design of domain-specific languages
  - Design of programming methodologies
  - Computer language engineering: building the programs that analyze, compile, and optimize other programs
- Formal methods for proving things about programs
  - Programs are abstract descriptions of computation; what can we prove about those descriptions?
  - Famous Turing result: the *halting problem* and undecidability
  - Lots of work in verification, protocol checking, bug finding
- Critical question: *what is actually being verified?*
  - the gap between software-as-abstraction and software-as-machine has always been with us, and probably always will be

RAD Lab

Impact: source portability

- Source-portability taken for granted
  - Increased leverage of programmers everywhere
  - BSD Unix and later GNU/FSF made it affordable (free)
  - *gcc* now taken for granted on any new architecture
- Interpreters and source-portability
  - ut interpreters too slow for “production” software?
  - Moore’s Law fixed all that
  - Perl, Python, PHP, etc. now common for web sites
- Software virtual machines, eg Java
  - Interpreter + just-in-time compiling
  - Software VM exposes machine-level and OS-level concepts (threads, scheduling, I/O primitives, etc.) normally hidden by high-level languages
  - VM “bytecode” is itself interpreted/compiled

RAD Lab

Impact: viruses

- Software has become overwhelmingly complex
  - Windows NT: ~60 million lines of source
  - Beyond the ability of any individual to fully grok
- *Software is not hardware*
  - Programmers tend to have an abstract state machine in mind (Turing) when designing software
  - But the system on which it runs has many "physically legal" states that don't correspond to any programmer-anticipated state
- Annoying result: bug
- Dangerous result: bug == security hole
  - Like a Murphy's Law—any bug that *can* be exploited as a security hole, *will* be, and at the worst possible time and by evil people

RAD Lab

Conclusion

- Separation of hardware and software may be the most important intellectual bifurcation of 20th c.
- Concepts go far beyond digital computers!
  1. Software as information that can be operated on, analyzed, etc.
  2. Software as an abstract description of how a machine should do a procedure
  3. Relationship between the physical machines and the representation(s) of its "software"
- Now replace "software" with "DNA" and "machine" with "biological system"
  - The last 50 years witnessed a profound revolution from the development of ideas of computer software
  - Both positive and negative impacts
  - Will the next 50 be the same for "biological

RAD Lab

For more...

- Computer Museum Visible Storage, Mountain View, CA
- Computer Museum Online Timeline—www.computerhistory.org
- Analytical Engine simulator: *www.fourmilab.ch/babbage*
- ENIAC online simulator (Google it)
- Turing Machine online simulators (ditto)
- the Hello World archive
- New Hacker's Dictionary (online a/k/a The Jargon File)
  - Esp. "The story of Mel, a *real programmer*" for insights into mentality of machine vs. assembly vs. compilers