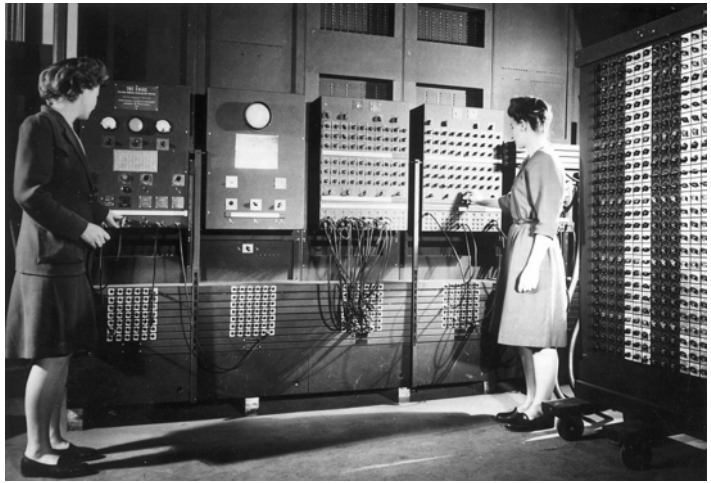


Lecture 20

A Brief History of Computers & Programming Languages

See "The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution" *by Walter Isaacson.*

IT in the News

- Scientists have been developing AI technology to identify disease from images
  - X-Rays of the lungs, C.A.T. scans of the brain, etc.
- Goal to improve the efficiency and cost of the health care system
- However, fear that hospitals and insurance companies can use adversarial attacks to maximize profits
  - Changing small numbers of pixels can trick the AI into a misdiagnosis
  - Insurers can learn the algorithms, use them for profit

Source: https://www.nytimes.com/2019/03/21/science/health-medicine-artificial-intelligence.html?action=click&module=MoreInSection&pgtype=Article&region=Footer&contentCollection=Asia

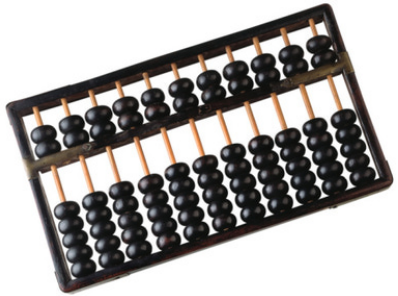
In the beginning… (1/11)

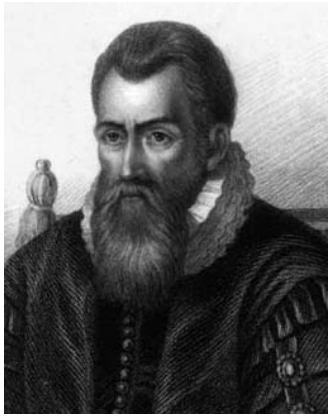
- Calculating and Calculus come from Latin word *calculus* (“a pebble or stone used for counting”), diminutive of calx (“limestone”)
- Crucial inventions are the positional numbering system, zero[1], and early algorithms such as multiplication and division
  - *Algorithm* is named after 8th century Persian mathematician and astronomer Al-Khwarizmi who was trained in Indian and Greek sources
  - *Algebra* is derived from *al-jabr*, one of the operations he used for solving quadratic equations

[1] Zero appears to have been invented in India, by Babylonians, Mayans, and Chinese. Carbon dating of an ancient Indian document, the Bakhshali manuscript, has recently placed the first written occurrence of the number zero in the third or fourth century A.D., about 500 years earlier than previously believed. (https://www.nytimes.com/2017/10/07/opinion/sunday/who-invented-zero.html)

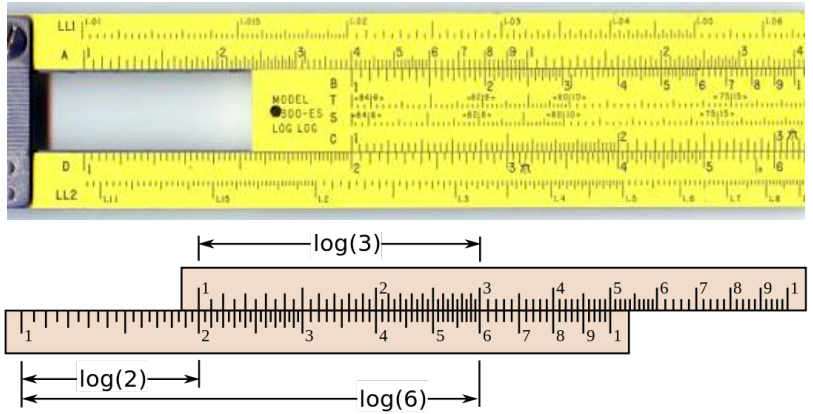
In the beginning… (2/11)

- Suanpan/Soroban/Abacus and other counting devices in multiple civilizations centuries AD! (and still in use!)
- Tables, logarithms and slide rules
  - **1614**: Napier publishes logarithms
  - **1620s**: first slide rules based on logs, effectively analog computing





Napier



In the beginning… (3/11)

- Long history of mechanical clocks, orreries, elaborate music boxes and instruments, up to and including today's player pianos (listen to Gershwin)
  - Mechanized music museum in Utrecht, NL – Museum Speelklok (player clock, like player piano)
    - Museum dedicated to mechanical music making
    - Pirates of the Caribbean theme song
    - Westworld Opening Credits

Video Link

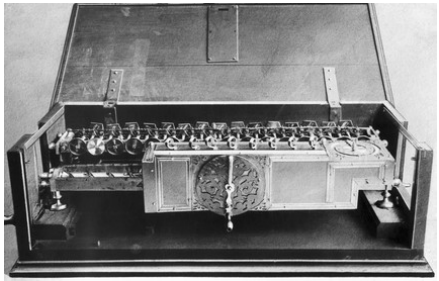
In the beginning… (4/11)

- Led to mechanical devices based on cogs on wheels and carry mechanisms
  - **1645**: Blaise Pascal creates the Pascaline adding machine – up to 8 dials
  - **1672**: Leibniz Calculating Machine (“stepped reckoner” could add, subtract, multiply, and divide)

Blaise Pascal

Pascaline

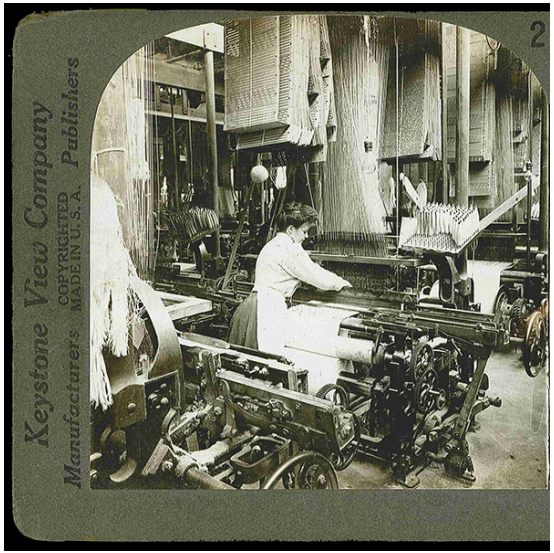
Leibniz

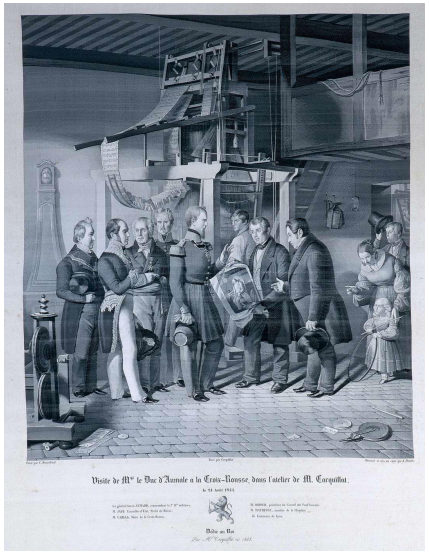


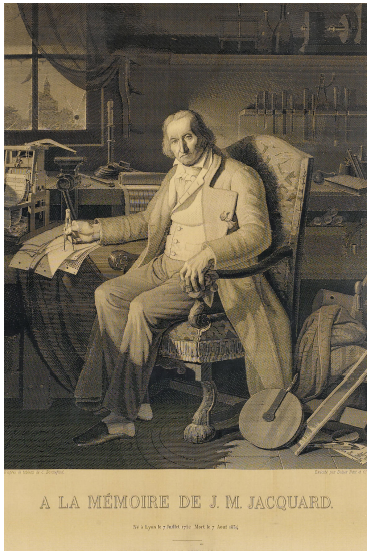
Calculating Machine

The Jacquard Loom

- **1725**: Perforated paper roll to control the patterns woven on a loom by Bouchon
- **1728**: Falcon laced punch cards together by string
- **1801**: Jacquard gets award for pattern loom driven by punched cards

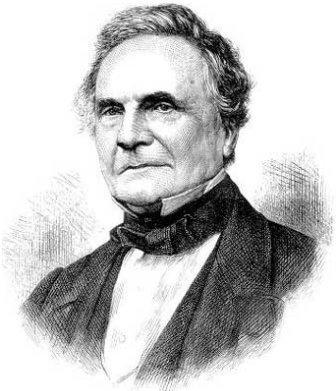






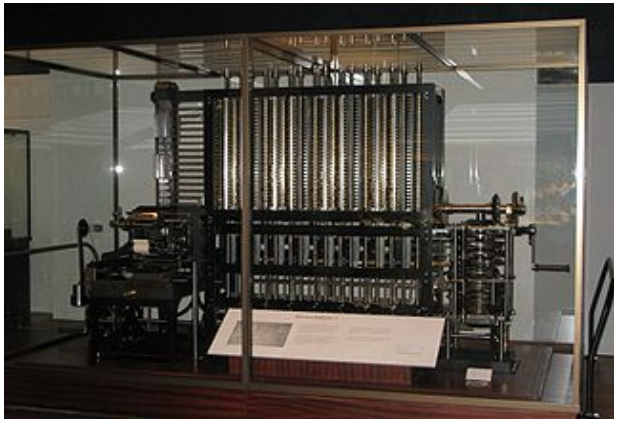
In the beginning… (5/11)

- **1822**: Eccentric British inventor Charles Babbage proposed idea of mechanical calculation to compute polynomials (for ballistics, longitude tables): The Difference Engine
  - this machine was designed but not built
  - the name derives from its method of divided differences



Charles Babbage

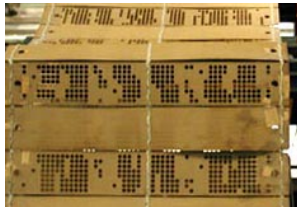
A modern implementation of the difference engine was finally completed by the London Science Museum in 2002



In the beginning… (6/11)

- Proposed combining mechanical calculation with idea of feeding instructions via punched cards in the style of music boxes and the Jacquard Loom, thus designing the first (mechanical) computer: the Analytical Engine
  - first had to invent tools for the precise machining required, but the Analytical Engine was never completed
  - however, the “architecture” is strikingly similar to essence of modern computers: driven by instructions, has arithmetic unit, memory, input/output

Jacquard Loom



Punch cards on a Jacquard Loom

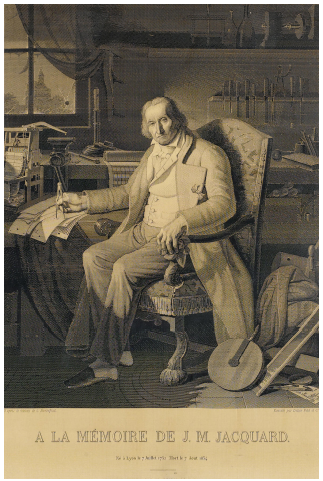
Babbage’s son built a small part of his analytic engine in 1910, and the Science Museum has begun the process of building a complete version

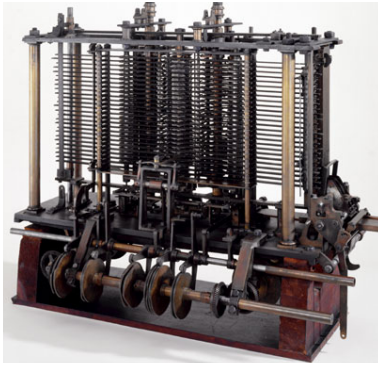
In the beginning… (7/11)

- **1845**: John Clark, cousin of founders of the Clarks' shoes empire, built the Eureka, a machine that produced "polished line[s] of Latin poetry"
  - one of the forerunners of the programmable computer
  - machine that generated Latin hexameter verse
    - hexameter is the meter of ancient epics, of the poets Ovid and Virgil
    - the strict rules of Latin hexameter make it similar to following a mathematical formula
  - 26 million possible permutations
    - *"If we had it running continuously, it would take 74 years for it to do its full tour before it started repeating itself."*

In the beginning… (8/11)

- **~1845**: Augusta Ada Lovelace (Lord Byron's mathematician, poet and philosopher daughter), writes program to calculate Bernoulli numbers for the Analytical Engine
  - first known computer program and programmer!
  - *"machine does exactly what you tell it to do"*
    - in other words, if you can tell it *what* to do, it will *do it* faithfully – the essence of computers & s/w!
  - *"the Analytical Engine weaves algebraic patterns the way the Jacquard Loom weaves flowers and leaves"*
  - Ada programming language named in her honor was a DOD-sponsored language
    - for writing programs using software engineering principles, including Abstract Data Types



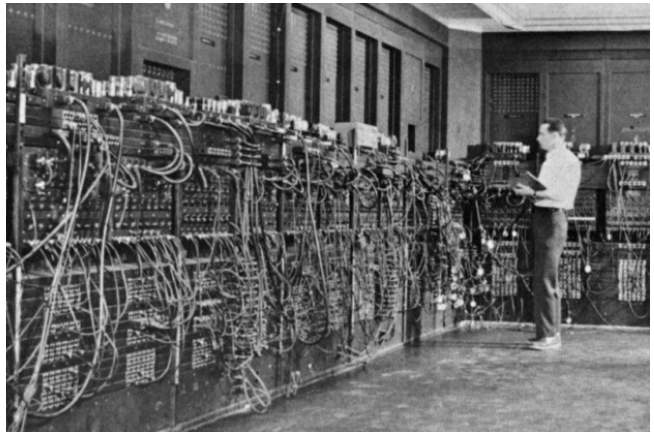


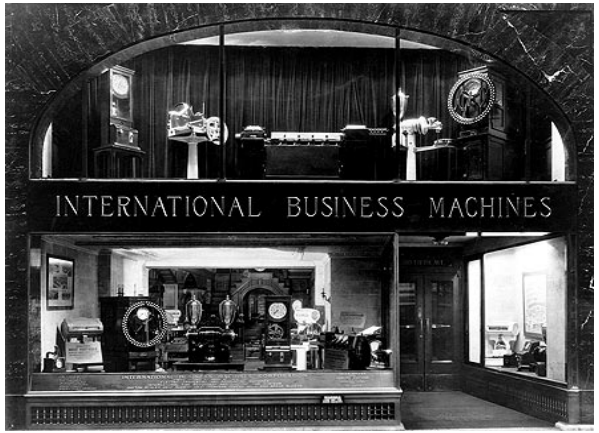
A piece of the analytic engine (photo from the Science Museum of London)

IBM Pavilion visitors view computer operating Jacquard loom during the 1968 World's Fair known as HemisFair '68 in San Antonio. Weaver/Programmer: Janice Lowry

In the beginning… (9/11)

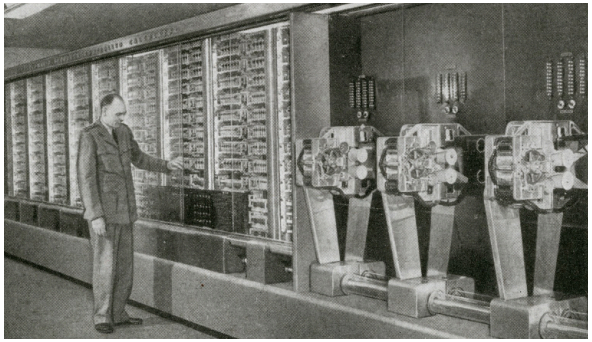
- **1890** Census bureau used Hollerith's tabulator fed by punch-cards
- **1900s**: Specialists programmed "business machines" (note IBM's original name) by actually changing hardware's wiring
  - advanced models used plug boards - try to debug that!

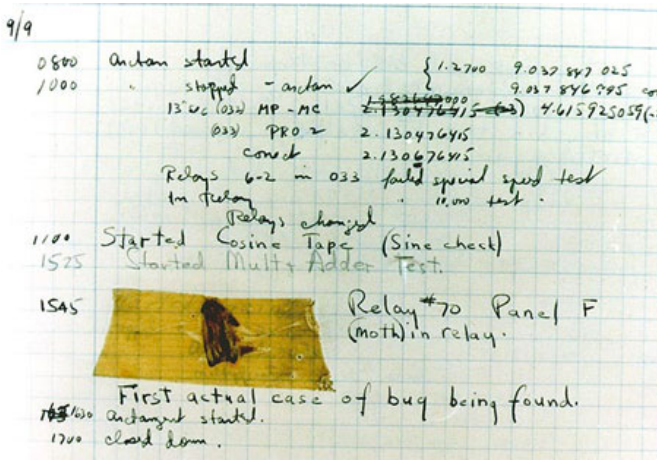




In the beginning… (10/11)

- **1930s**: The IBM **Automatic Sequence Controlled Calculator** (**ASCC**), called **Mark I** was designed by Harvard University's staff
  - Howard Aiken presented the first design in November 1937
  - it served as a general purpose electro-mechanical (relay) computer that was used in the war effort during the last part of World War II – ballistic and bomb calculations
  - Grace Hopper credited with finding a real bug





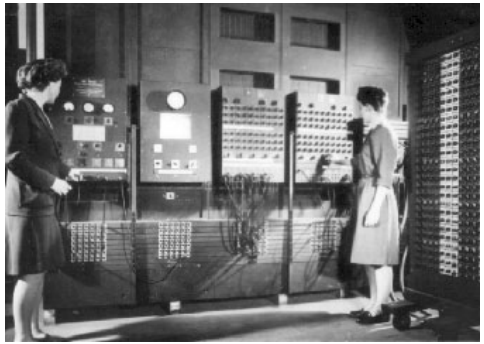
First actual case of bug being found.
See: "The moth in the relay"

In the beginning… (11/11)

- **1946**: J. Presper Eckert and John Mauchly at University of Pennsylvania built Electronic Numerical Integrator and Computer (ENIAC)
  - first electronic general purpose “automatic” computer, used for ballistics and bomb calculations
  - 18,000 vacuum tubes, MTBF of 20 minutes! Still programmed by wiring
  - 5,000 adds/ subtracts/sec, 400 multiplies/sec, 35 divisions or square roots/sec (10 digit numbers)
  - men wanted to build hardware, left less prestigious programming to the all-female corps of programmers, called “computers”, a 19th century term
    - Article: The Forgotten Female Programmers Who Created Modern Tech

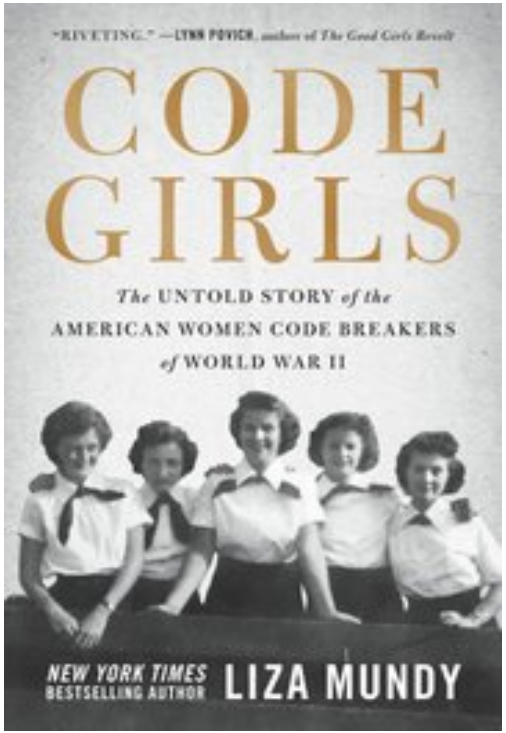
J. Presper Eckert

John Mauchly



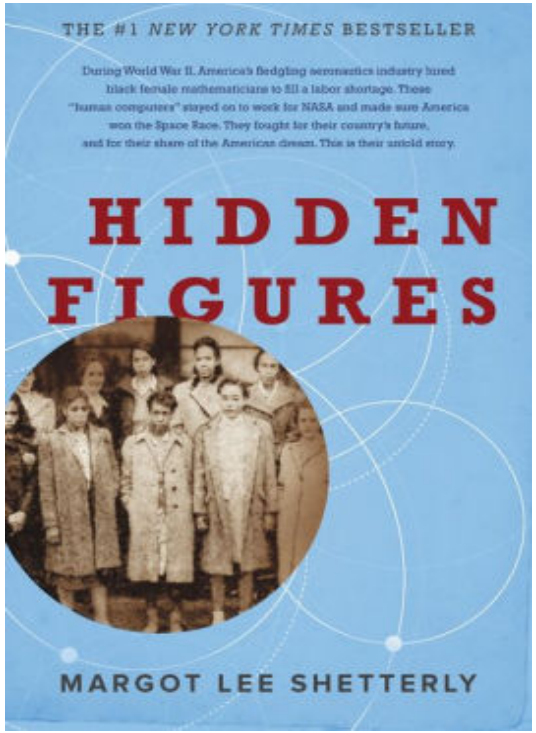
Belatedly Recognized Female Pioneers (1/2)

- “Code Girls: The Untold Story of the American Women Code Breakers of World War II” by Liza Mundy
- During World War II, a group of women were chosen for their abilities and math
- They decoded several Japanese messages which enabled the US to sink military ships and stop enemy attacks
- Among them, was Ann Caracristi who became the first female NSA director in 1980
- The Women Who Helped America Crack Axis Code



Belatedly Recognized Female Pioneers (2/2)

- “Hidden Figures: The True Story of Four Black Women and the Space Race” by Margot Lee Shetterly, also made into a movie in 2016
- Tells the story of the black women who worked in NASA, in segregated Virginia and contributed to the successful launch of the first men in space
- Katherine Goble made accurate calculations which allowed the Friendship 7 shuttle to launch successfully
- Dorothy Vaughan taught herself FORTRAN so she and her co-workers could operate the new IBM machines, and was named supervisor
- Mary Jackson obtained her engineering degree and was hired as an engineer at NASA

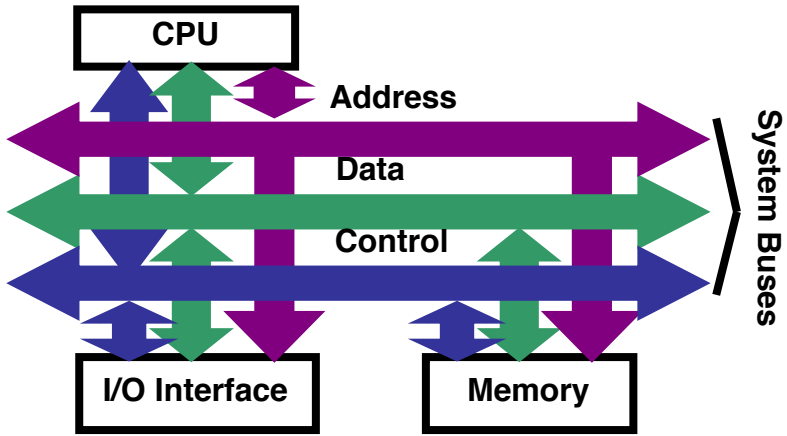


Stored Program Computer Architecture (1/2)

- **1945**: John von Neumann introduces seminal concept of “stored program”
  - “it’s bits all the way down…” for both data and instructions
  - program can be stored in main memory and even treated as data to be operated on–paved the way for modern computers

Stored Program Computer Architecture (2/2)

- Simple instruction execution loop
  - use instruction register to fetch instruction stored at that address in memory, increment instruction counter, decode and execute instruction
  - instruction typically is <op-code> <address>
  - instruction may update memory – even modify stored program itself, e.g., for loops. Unsafe: now we use h/w looping w/ index registers
- von Neumann was a polymath: worked on atom and hydrogen bombs, co-invented “game theory”, computational biology, etc.



Simplified Processor Architecture

CS15 Final Project Othello uses the minimax algorithm!

Moore's Law and the Shrinking Computer (1/2)

- Moore's (1) "Law": an observation that over the history of computing hardware, number of transistors in a dense integrated circuit doubles approximately every two years
  - In < 6 years it increases an order of magnitude!
- Smaller feature size, greater density means shorter paths, faster signal propagation in microprocessors
- We benefit not just from microminiaturization of the CPU but also from great electromechanical engineering of peripheral devices (e.g., disk controllers, disks – 40MB was a big disk in the 60s!) My graphics group's dual Meta-4 mini had 32KB, 1MB disk, ran a complete time-sharing system

(1) Gordon Moore was the co-founder of Intel and the co-inventor of the integrated circuit, which led to microprocessors, etc.

Mainframe Machine Room

IBM System 360 /65 with 2314 disk unit;
8x25MB = 200 Mbytes (late 1960's)

1100 x IBM 2316 29Mbyte disks in one 32
Gbyte microSD card

Moore's Law and the Shrinking Computer (2/2)

- IBM z13 microprocessor (6-8 cores @5 Ghz) has about same computing power as 5,000 football fields worth of Brown's IBM /360 mod 50s (.14MIPS) 50 years ago–mainframes still selling!
- A Look at Mainframe History as IBM System/360 Turns 50, COBOL Turns 55
- But are Silicon chips hitting a limit? Transistors with a single atom?
- What's next: biological computers? Quantum computers?
  - qubits store multiple states (superposition, like fundamental and harmonics in sound), allowing parallel storage and computation–big engineering problem to make them (cryogenic)
  - factorization for large N shown to be doable with quantum computers–would wreak havoc with en/decryption
  - making a full QC, let alone a general purpose QC using non-semi-conductor solutions, still very much in research stage

Computers get faster, but do they get more “powerful”?

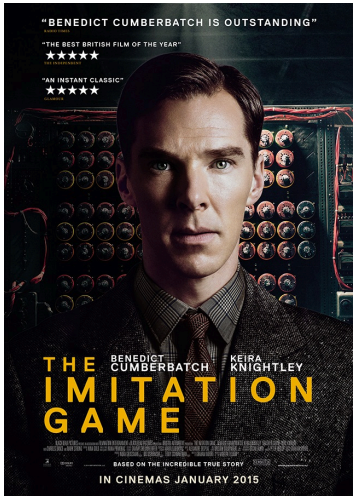
- Computer is the only Universal Machine!
- Yet theoretically only need 6 instructions for ANY algorithm!
  - load accumulator (high-speed register) from memory address
  - store accumulator to memory address
  - subtract contents of memory address from accumulator, store result in accumulator
  - jump to memory address (instruction) if accumulator < 0 (“conditional jump”)
  - read to accumulator from external input device
  - write from accumulator to external output device
- You can build
  - add by subtracting negative numbers
  - divide by repeated subtract, multiply by repeated add
  - if-then-else and loops with conditional jump
  - output to printer by write into special memory location

Tradeoffs in Power/Complexity of Instruction

- Tradeoffs
  - complexity of instruction (how much it does)
  - speed of instruction execution
  - size of instruction set (and can compiler take advantage of them)
- Today's computers
  - Complex Instruction Set Computer (CISC) > 500
    - started with IBM mainframes in 50s and 60s, now "Intel architecture" dominates
  - Reduced Instruction Set Computers (RISC) 100–300 (simpler but faster instructions)
    - major innovation and important in 80s and 90s
  - Intel architecture has adapted best ideas from RISC
  - ARM architecture also RISC; used in phones, tablets, etc. Arduino boards for IoT
  - emphasis today is on "multi-core" (multiple CPUs per chip) and low-power designs
  - GPUs (Graphics Processing Units) are even more powerful. For games, but also for data crunching, e.g., scientific simulation (weather prediction, protein folding, machine learning…) – increasingly use GPU clusters for heavy duty computation
    - program GPUs with "shader programming" in CS123, Intro to Graphics

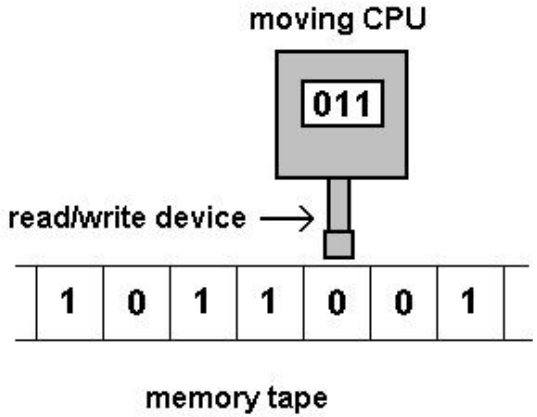
Turing, Computability (1/3)

- Alan Turing (1912–1954): logician, mathematician, cryptanalyst, first computer scientist, theoretical biologist
- Designed code breaking machine to crack the WWII German Enigma cypher



Turing, Computability (2/3)

- Formalized notions of algorithm, computer and mechanized computing in an era that was very concerned with what was computable and what was not, mechanized mathematics, e.g., undecidability, halting problem, etc.…also started AI, Turing Test
- Turing Machine as the simplest possible *conceptual* device to execute an arbitrary algorithm: device with a writable tape and a read/write head; the logic is in a table



Turing, Computability (3/3)

- Table contains the “program” of “instructions” as a “state machine”–if in state *i* and read 1, do action *x*, then go to a next state; if read 0, do action *y*, go to a next state. Simple actions:
    1) move the r/w head one square left or right
    2) read/write current cell (empty or tape alphabet)
    3) does not have to halt
- Universal Turing Machine that could simulate any other TM by simulating its table.
    - proof that one could build a universal “programmable” computer
    - MIT’s AI pioneer Marvin Minsky devised a 43-state UTM!
- Turing committed suicide after being prosecuted and “treated” (chemically castrated) for being gay
    - PM Gordon Brown publicly apologized in 2009, Queen Elizabeth granted a posthumous pardon in 2013

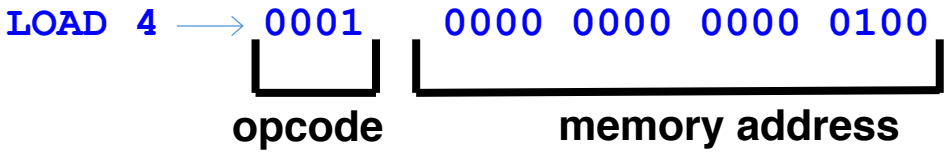
Marvin Minsky

Born: August 9, 1927

Died: January 24, 2016

First, Numeric Machine Language, Then Came Assembly Language (1/2)

- **1949**: John Mauchly develops Short Order Code
  - first assembly language
  - provided vehicle for higher-level languages to be developed
- Symbolic code that is 1:1 with machine code



  - load accumulator with contents stored at address 4
  - program translates to machine code via table lookup of opcode, decimal to binary conversion algorithm
  - assembler early example of treating a program as data!

First, Numeric Machine Language, Then Came Assembly Language (2/2)

- Must be defined for each processor
  - hard-wired for particular processor's architecture
  - generated by compilers for higher-level languages
- Modern processors are very complicated
  - so writing at assembly language level takes real skill (learn it in CS33!)
  - compilers can optimize code globally for high-level languages, using sophisticated computation graphs
  - programmers generally optimize code only locally
- Still used today when speed and size count
  - embedded computers, device drivers, games
  - programmer must understand hardware well to use effectively
  - increasingly, C is used as a "machine-independent" assembly language (CS33)

High-Level Languages

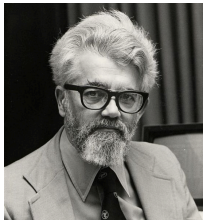
- Attempt to make programming more intuitive
  - closer to programmer's concepts (high-level)
  - further from machine's concepts (low-level)
- Symbolic code that is 1:N with machine code
  - one high-level instruction may become tens or hundreds of machine code instructions
- Most importantly, machine independent
  - avoided vendor lock-in
  - depended on compiler to translate high-level constructs to computer's machine code
  - thus allows one source program to be used on many target architectures
- Still trying to make languages higher level
  - Java guarantees single compilation, same execution on multiple machines via byte codes: write once, run everywhere
  - compile to byte code virtual machine; computer will have virtual machine interpreter

High-Level Languages: Important Dates (1/2)

- **1957**: John Backus et al. at IBM develop FORTRAN language and compiler
  - FORmula TRANslator
  - still used today, mostly for scientific computing, highly optimized for number crunching
- **1959**: Committee on Data System Languages develops COBOL
  - led by Rear Admiral ***Grace Hopper***, one of first modern programmers (Grace Hopper Celebration of Women in Computing – from ~200 in 2001 to ~ 20,000 in 2018!)
  - Common Business Oriented Language, “English-like,” support for data records
  - still tons of legacy code in banks, insurance companies, retail… (Y2K!)

High-Level Languages: Important Dates (1/2)

- **1959**: John McCarthy develops LISP
  - LISt Processing
  - seen as slow, so primarily used only for “AI” projects
  - Scheme is a modern Lisp-like “functional programming” language
  - Python (CS16) can be seen as language with LISP functionality, but with modern syntax (http://norvig.com/python-lisp.html)



- **1960**: ALGOL 60 standard published
  - ALGOrithm Language
  - basis of most popular languages today

- **1964**: John Kemeny and Thomas Kurtz at Dartmouth develop BASIC
  - Beginners All-purpose Symbolic Instruction Code
  - simple language, meant to be used by beginners and non-professionals, efficient on microcomputers
  - was popularized by Microsoft’s Visual BASIC, now being replaced by JavaScript

Structured Programming (1/2)

- **1968**: Edsger Dijkstra writes landmark note: “GoTo Statement Considered Harmful”
  - GoTo, an unconditional branch without a return, leads to spaghetti code
  - no predictability, FoC can go anywhere in program, can’t be understood by programmer or compiler

- New languages would have constructs for common one-in-one-out flows of control for controlled branching–the return from the branch is prescribed
  - `if/else-if` and `switch` statements
  - `while` and `for` loops
  - gives sequential, predictable order to code, only controlled branches allowed
  - allows better code optimization

Structured Programming (2/2)

- Brown's AM101, AM40, CS11 (CS15 precursors) switched to new structured programming style using only 1-in, 1-out branching in late 60's
  - taught PL/I, then PL/C, a student-oriented subset
  - even taught "structured assembly" based on positive experiences
  - switched to Pascal as a more modern language
  - then to Object Pascal, an OOP extension; then to Java
  - see -- we have a rich history of experimentation!

Next Generation High-Level Procedural Languages

- Emphasize task decomposition, no bundling of data and procedures in “objects”
- **1964**: Researchers at IBM develop PL/I, an omnibus language
  - Programming Language I
  - designed to synthesize best features of FORTRAN, COBOL, and Algol 60
  - failed as attempt to be the one general purpose programming language
- **1970**: Niklaus Wirth develops Pascal
  - named for Blaise Pascal, designed to be an educational language
- **1972**: Dennis Ritchie at Bell Labs develops C (also learned in CS33)
  - predecessor named B
  - often called portable assembly language
  - surpassed COBOL as most popular language

Object-Oriented Programming Languages (1/3)

Even **OOPLs are Relatively Old!**

- **1967**: Ole-Johan Dahl and Kristen Nygaard at Norwegian Computing Centre develop Simula, SIMUlation Language and first OO programming language, classes
- **1972**: Alan Kay, Adele Goldberg, et al. at Xerox PARC develop Smalltalk and the Windows metaphor/GUI
- **1972**: Barbara Liskov at MIT develops CLU, with focus on ADTs (next slide)

Aside: Barbara Liskov's Talk at Brown 11/06/14

- Biography:
  - member of the National Academy of Engineering and the National Academy of Sciences, the National Academy of Inventors.
  - ACM Turing Award (the Nobel prize of CS), IEEE Von Neumann medal, Brown honorary degree
- The Power of Abstraction
  - abstraction is at the center of much work in Computer Science
  - finding the right interface for a system as well as finding an effective design for a system implementation
  - furthermore, abstraction is the basis for program construction, allowing programs to be built in a modular fashion.
- What I learned from her talk
  - ADTs need to describe the behavior, not just the method signatures, return types, errors: "pragmatics"
  - Java and other OOPLs can only provide support for enforcing that subtypes can do what supertypes can–they can't enforce the idea that subtypes should also exhibit the same behavior
  - CS15 has de-emphasized inheritance, pushing interfaces and composition

Object-Oriented Programming Languages (2/3)

- **1980**: US Department of Defense develops Ada to combat plethora of languages whose code doesn't interoperate
  - ADT's, Objects, Concurrency…
  - like PL/I, an omnibus, complex language
  - defense contractors had trouble finding qualified staff to write in it
- **1983**: Bjarne Stroustrup develops C++
  - OO extensions to popular C language–named C++ as a play on the ++ operator (one better than C!)

Object-Oriented Programming Languages (3/3)

- **1995**: James Gosling et al. at Sun Microsystems develop Java, a cleaned-up, smaller dialect of C++
  - meant to be internet and embedded device programming language
  - provide facilities for better reuse and safety
  - some professionals avoid it because it is seen as inefficient (use C++ or C instead)
  - Microsoft's C# is a powerful Java-ish competitor; also Python, Ruby-on-Rails
  - JavaScript is NOT Java, and is only partially an OOP

Important note: OOP is one of multiple programming paradigms, **not a panacea**. Procedural and **functional programming**, and special purpose languages like MATLAB and MATHEMATICA, are tools with their own applicability, and anyone developing s/w needs to be multi-paradigm, multi-lingual

Who “owns” APIs? (1/2)

- **Oracle vs. Google reuse of APIs and Java code**
- **November 7, 2014: Computer Scientists Ask Supreme Court to Rule APIs Can’t Be Copyrighted**
  - The Electronic Frontier Foundation (EFF) filed a brief with the Supreme Court, arguing on behalf of 77 computer scientists that the justices should review a disastrous appellate court decision finding that application programming interfaces (APIs) are copyrightable
  - That decision, handed down by the U.S. Court of Appeals for the Federal Circuit in May, up-ended decades of settled legal precedent and industry practice
  - Signatories to the amicus brief include five Turing Award winners, four National Medal of Technology winners, and numerous fellows of the Association for Computing Machinery, IEEE, and the American Academy of Arts and Sciences
    - the list also includes designers of computer systems and programming languages such as AppleScript, AWK, C++, Haskell, IBM S/360, Java, JavaScript, Lotus 1-2-3, MS-DOS, Python, Scala, SmallTalk, TCP/IP, Unix, and Wiki. Avd also signed

Who “owns” APIs? (2/2)

- **June 29th 2015**: Supreme Court refuses to rule on Court of Appeals ruling upholding Oracle’s ownership of Java API’s; the suit for copyright infringement against Google is ongoing, with Google using “fair use” doctrine
  - May 2016: Jury ruled in Google’s favor, using “fair use” doctrine
  - March 2018: After Oracle filed an appeal, Federal Appeals Court overturned the jury and said Google’s use of Java was not “fair use.” Case is back to trial court to determine damages. Google petitioned the entire Court to rehear the case, but The Federal Circuit denied Google’s petition.
  - January 2019: Google filed another petition asking the Supreme Court to review Federal Circuit Decisions.

- Question: even if you can copyright an API, is it enforceable?

Software Engineering (1/4)

- **1968**: NATO Science Committee addresses “software crisis”
  - hardware progressing rapidly, but not software
  - software development seen mostly as craft with too much trial-and-error
  - too little has changed – e.g., ACA website debacle! (and RI’s multiple failed roll-outs: DMV, DHS SNAP, …)
  - coins term software engineering
- **1975**: Frederick Brooks writes landmark book “The Mythical Man-Month”
  - says “no silver bullet,” software is inherently complex – most complex man-made systems
  - complexity can be ameliorated but cannot be cured by higher-level languages
  - adding people to project delays it (“9 women can’t make a baby in a month”)

Software Engineering (2/4)

- **1990**s: Les Hatton develops “30-5-1” rule
  - from study of real commercial programs
  - discovered 30 bugs per 1000 lines untested code on average, then only 5 in well-tested code, and 1 bug still remaining after code in production
  - rule held regardless of language, probably still true today!
  - all commercial s/w has day-one bugs, and for non-life-threatening s/w, we tolerate it
  - under the guise of “early availability” vendors let the user community debug
  - unacceptable for mission-critical s/w used in nuclear reactors, weapons, vehicles, medical apparatus, EFT and other banking apps, IRS s/w, etc.

Software Engineering (3/4)

- Sophisticated development and testing methodologies
  - CS17 and CS19 teach students to write tests that inform the implementation rather than write tests that are tailored to the implementation
  - goal is to cover both general and edge cases
  - formal verification (proving h/w and s/w correct) is in the ascendency again

Software Engineering (4/4)

- Libraries of reusable components
  - companies offer well-tested common components
  - “plug-n-play” frameworks to connect trusted catalogue parts
  - OOP/D is a good paradigm to make this goal feasible–works well for GUI widgets (aka controls), as in JavaFX, and large-scale components (e.g., “Enterprise JavaBeans”, QT Framework used in CS123)
- CS32: modern software engineering using Java!
- **Note**: new languages and software engineering technologies (frameworks, IDEs…) still hot subjects, both in industry and in academia
  - e.g. Apple’s Swift, positioned as successor to C and Objective-C, and Google’s Dart, designed to make common problems in app development easier to catch

Implications of Information Technology

- Computing/IT history isn't just about all the great strides we've made, and the fact that we're still in the dawn of this technological revolution!
- Growth was driven by techno-optimism and the disruption paradigm, focus of Schumpeter's "creative destruction" economic theory of the effects of innovation
  - Steve Jobs: personal computing
  - Bill Gates: information at your fingertips
  - Mark Zuckerberg: connecting the world
- Now we recognize that there can be unforeseen, harmful consequences, disruption
  - job displacement because of automation (eliminating and creating new jobs, upskilling, …)
  - hacking of personal and corporate data
  - integrity of the voting process
    - influencing elections via fake news; disturbingly easy to hack voting machines,…
- Instead of bringing us together via affinity groups and virtual communities, these technologies have…
  - created bubbles, hardening positions, amplifying the echo chamber, fueling conspiracy theories
  - spread of alternate facts and the post-truth world
  - fueled anti-ethnic/religious biases, "nationalism"
  - Cambridge Analytica, etc.

Be techno-optimists, help improve our world, but think about unintended consequences

Announcements

- Tetris Early Deadline is **Tomorrow**!
  - On-time deadline: Sunday, November 17
  - Late deadline: Tuesday, November 19
- No sections next week!
- Explanation of final projects will be upcoming Tuesday. Make sure you come to lecture!
- Handouts for Final Projects will be released on Saturday the 16th
  - There are five options –take a look through them and start thinking about which one you want to do.
  - Help Sessions for each project will be held during Thursday's class.
- Final Project Hours will start next Monday