history of programming, part I

thursday 3 august 2000

lecture overview:

- John von Neumann (1945)
- intervening history
- John Backus (1977)
- lambda calculus
- John McCarthy (1979)

is everyone named "John"???

John von Neumann

- Hungarian (János Neumann), (1903-1957)
- "First Draft of a Report on the EDVAC", technical report, 1945.
- ENIAC: Electronic Numerical Integrator Computer
  - developed at Moore School of Electrical Engineering, University of Pennsylvania
- EDVAC: Electronic Discrete-Variable Automatic Computer
- spawned a generation of successor machines: ILLIAC, JOHNIAC, SILLIAC, CSIRAC...

von Neumann: background

- two groups worked on design of EDVAC:
  - Eckert and Mauchly: memory
  - von Neumann, Goldstine and Burks: logic, architecture
- von Neumann's "First Draft":
  - published in June 1945
  - initially 24 copies
  - later, hundreds circulated
  - killed all possibility of patenting EDVAC design
- conflicting interests:
  - Eckert & Mauchly: commercial
  - von Neumann, et.al: scientific
- ensuing dispute:
  - Eckert & Mauchly: "von Neumann is a glory hog..."
  - von Neumann: "E & M are selfish and venal..."

von Neumann: background, cont.

- Spring 1946:
  - everyone leaves Moore School
  - E & M form Electronic Control Co., later, Eckert-Mauchly Computer Corp.
  - von Neumann, Goldstine and Burks went to Institute for Advanced Study (IAS), Princeton University
- Summer 1946:
  - "Moore School Lectures"
  - lectures by all principals
  - attended by scientists and engineers from all over North America and the U.K.
  - together with EDVAC report, a major catalyst for computer industry
  - established tradition of conferences for disseminating computer science research

von Neumann: background, aftermath

- Summer 1947:
- – E & M file ENIAC patent application
- – others not invited
- – they claim original architecture and electronics for arithmetic and control
- – von Neumann's attitude: *ENIAC and EDVAC were funded by the public, so they should be owned by the public.*
- – nevertheless, both von Neumann and E & M attempted to file (conflicting) patents for EDVAC
- – application was immediately dismissed because of von Neumann's report (prior publication means that the information is public domain)

von Neumann: EDVAC

- automatic computing system

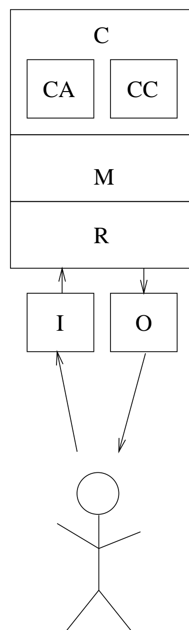
- carries out series of *instructions*

- must include:
- – initial and boundary values of dependent variables
- – values of fixed parameters... "constants"
- – tables of fixed functions

- *encoded* on some medium:
- – e.g., punch cards, teletype tape, magnetic tape
- – results get recorded on this medium as well

- must compute without "intelligent human intervention"

von Neumann: EDVAC architecture



components:
- C = central component
- – CA = arithmetic
- – CC = control
- M = memory
- R = recording medium
- I = input
- O = output

von Neumann: memory functions

- short-term memory (M)

- logical instructions (R)

- function tables (R)

- initial conditions and parameters (R)

- intermediate results (R)

- sorting and statistics (R)

von Neumann: relays

• could be:
– wheels (mechanical)
– electric circuits (i.e., telegraph relays)
– combination of wheels and circuits
– vacuum tubes

• exist in 2 (or more) discrete *states*
– also called "equilibria"
– parallel to neurons in animal brains, which have two states: quiescent and excited

⇒ binary (base 2) system
– M, CC, CA: use binary
– R uses decimal
– I and O have binary ⇔ decimal converters

• timing
– may be regulated externally
→ "asychronous"
– or internally, by a fixed clock
→ "sychronous"

von Neumann: argument for sequential architecture

• "telescoping" requires (non-redundant) duplication of processing elements
– form of parallelization
– e.g., addition:
1. add all pairs of digits at once
2. add all first order carries
3. add all second order carries
4. etc.
– necessary in pre-electronic calculators

• non-redundant duplication leads to:
– slower computation (slowest element constrains speed)
– less reliable computation (increased probability of failure)

von Neumann: representation of numbers

• for operations, numbers → CA

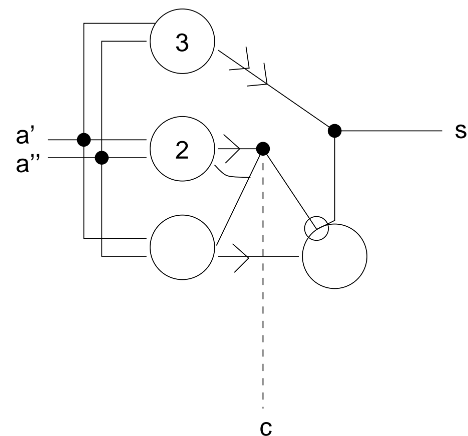
• how to represent digits?
– relays: 0 = absence of stimulus
1 = presence of stimulus

• how to represent multi-digit numbers?
(1) multliple relays
or
(2) single relay, pulsed over time

"a number is represented by a line, which emits during 30 successive periods t the stimuli corresponding to its 30 binary digits"

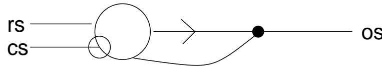
von Neumann: adder network

a' + a'' = s

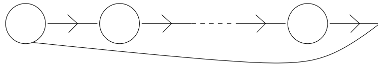


von Neumann: memory through delay lines

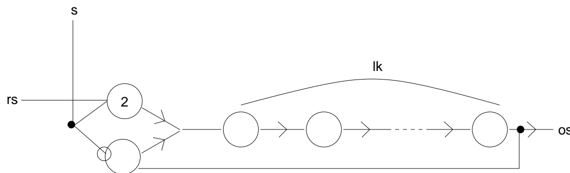
- memory device: an element which stimulates itself will hold a stimulus forever
- m1 = memory device that remembers one stimulus, or one binary digit



- mk = memory device that remembers k stimuli, or k binary digits
  ⇒ delay organ



- with terminal equipment ⇒ “perfect memory organ”



intervening history: England

• first stored-program computers produced in England
– Manchester University: Max Newman, later Alan Turing

• F.C.Williams designed CRT memory
– electrodes for reading memory
– faster than delay lines
– reliability problems (as with delay lines)
– prototype operational mid 1948
– no I/O other than using front panel

• Maurice Wilkes designed EDSAC
– Electronic Delay Storage Automatic Calculator
– Cambridge University
– paid his own way to Moore School lectures:
I recognized this at once as the real thing
– goal: build up lab with emphasis on computer usage and *software* (as opposed to hardware)
– simple design, early completion
– running mid 1949 (¡3 years after M.S.lectures)
– world’s first practical stored-program machine
⋄ 32 mercury delay lines
⋄ 3000 vacuum tubes
⋄ 30,000 watts

• after the 1940’s, England slowed down, and US took over advances

intervening history: USA

• E & M’s Electronic Control Co. (ECC)
– goal: electronic data processing, not scientific computation
– contract with US Census Bureau

• UNIVAC = UNIversal Automatic Computer (1946)
– estimated cost US$300K
– design similar to EDVAC
– mercury delay line main memory
– mass storage on magnetic tape (“magtape”): completely new idea
– also new:
⋄ digital use of mag tape
⋄ metallic mag tape
⋄ tape-card converters

• BINAC = BINary Automatic Computer (1947)
– contract with Northrop for production
– smaller “airborne” computer to guide missiles
– estimated cost US$100K
– 2 central processors with different designs, attempt to detect faults
⋄ 2 mercury delay lines
⋄ 700 vacuum tubes
⋄ 13,000 watts
– first American stored-program machine, 1949
– final cost: US$278K

intervening history: USA, cont.

• ECC sold 40% to American Totalisator,
⇒ EMCC (1948)
– backed by H.Strauss (VP)
– US$500K cash infusion
– allowed completion of BINAC and continued development of UNIVAC
– contracts for 6 UNIVACS were drawn
– 1949: Strauss killed; financial backing withdrawn
– EMCC offered to IBM
⇒ declined!

• IBM (International Business Machines)
– focus on commercial importance of computers
– not scientific/defense work:
“Evolution, not revolution”
– worked on retro-fitting electromechanical calculators with electronics

• Rand
– purchased EMCC in 1950
– also bought Engineering Research Associates
– two computer divisions fought for resources
– hired Grace Hopper as head of programming
– delivered first UNIVAC to Census Bureau, early 1951

IBM realizes their mistake!

intervening history: IBM

- accelerates existing computer efforts

- IBM 701
– their first scientific computer ("defense calculator")
– delivered in 1953
– Williams tube memory, 2K words

- IBM 702
– Williams tube memory, 10K words
– first IBM commercial computer
– announced 1953, delivered 1955

- IBM 650
– "magnetic drum computer", 20K words
– delivered 1955
– much cheaper than 700 series; 2000 delivered
– *supplied to universities at a 60% discount*

- development of magnetic core memory (1953)
– at MIT and RCA
– MIT won patent, royalties worth $$$

- UNIVAC outsold IBM 700 series 30:24 until Aug 1955
  by 1963, IBM held 70% market share

intervening history: IBM vs UNIVAC

- UNIVAC / Remington Rand
– infighting between:
  E & M division (Philadelphia)
  ◇ UNIVAC I & II
  ERA division (Minneapolis)
  ◇ UNIVAC 1101, scientific computers
– poor marketing
– poor support
– UNIVAC II slow to market

- IBM
– history of excellent marketing and support, although mediocre technology
– leasing strategy made machines accessible to medium-sized companies
– installed base of card calculator users
– card/printer peripherals superior

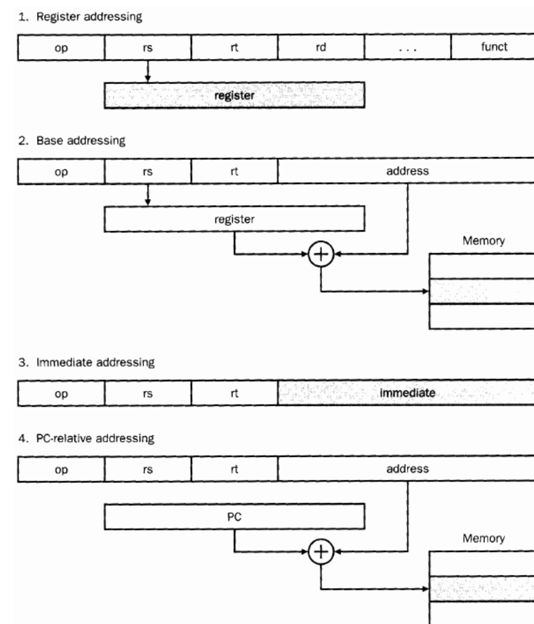
intervening history: development of machine languages

- first machine languages
– Babbage's analytical engine:
  punch card locations indicating
  ◇ operations, e.g., +, -, *, /, move, read/write
  ◇ registers, e.g., 1, 2, 3, ...
  ◇ numbers (constants)

- ENIAC
– plugs for cables
– 2-digit location codes for operations wired into function tables

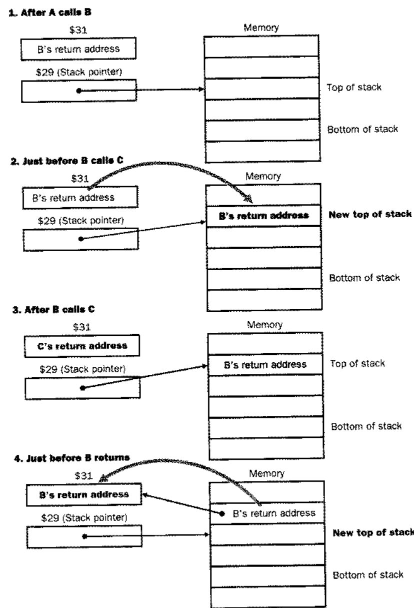
- UNIVAC (and after)
– more "ordinary" machine languages
– strings of numerical digits
  ◇ divided into words

intervening history: machine language, word definitions



(MIPS architecture)

intervening history: machine language, memory

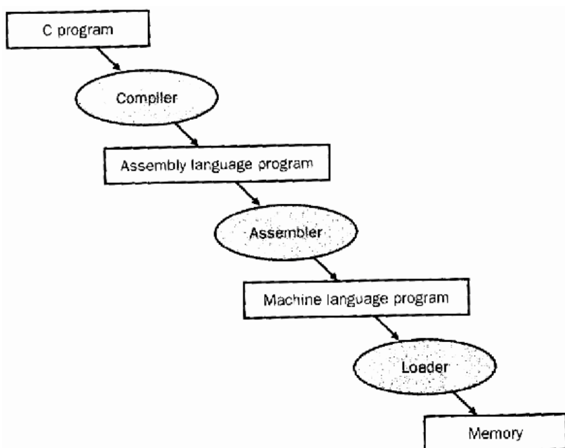


(MIPS architecture)

intervening history: first assembler

- EDSAC computer, Cambridge
- – first programming in reality, except ENIAC
  (unlike famous historical vaporware: Ada Lovelace's imagined program, Turing's paper-driven chess algorithm)
- – Wilkes recognized time/effort of "debugging"
- – Wilkes & Wheeler: subroutine library
  - ◇ goal: software re-use
- – Wheeler developed:
  - ◇ loader
  - ◇ relocating loader
  - ◇ assembler
  - ◇ subroutines

intervening history: machine languages, on to higher level languages



intervening history: first high-level languages

- Short Code
- – for BINAC and UNIVAC (1949)
- – by Mauchly and Schmitt
- – more primitive than assembler
- – features:
  - ◇ variables
  - ◇ assignment
  - ◇ basic mathematical functions
  - ◇ line numbers
  - ◇ output
- – used 2-character codes, 6 per instruction
- – not widely used

intervening history: FORTRAN to COBOL

- developed by John Backus, et.al at IBM, 1954-7
  - programming+debugging >50% computer costs
  - so...specify procedures in mathematical language and use computer to translate to machine language

first law of software: programmers will always produce 10 lines of code (+/- e) per day (debugged and documented)

  - FORTRAN I (1957)... forgot subroutines!
  - FORTRAN II (1959)... included subroutines

- Grace Hopper
  - organized conferences on "automatic programming" (1954 and 1956)
  - accelerated acceptance of high-level languages
  - first "real" compilers
  - sparked move from Mathematic to Flow-matic
    - ◇ use English-like words
    - ◇ separate data definitions from procedure definitions
    - ◇ self-documenting code

programs ought to be intelligible to their users!

- COBOL
  - initiated at Pentagon meeting, May 1959
  - users: government, manufacturers
  - COBOL = COmmon Business Oriented Language
  - first instance of a group of competitors producing a co-operative computer standard (later ISO, ANSI, OSF, etc)

John Backus

- led IBM team to develop FORTRAN (1957)
- contributed to design of ALGOL-60
- invented Backus-Naur Form (BNF) notation for language specification
- "Can programming be liberated frm the von Neumann style? A functional style and its algebra of programs" (1977)
  - scathing criticism of von Neumann style languages
  - description of a (new) functional approach to programming

Backus: criticism overview

- primitive word-at-a-time style of programming
- close coupling of semantics to state transitions
- division of programming into a world of expressions and a world of statements
- inability to effectively use powerful combining forms for building new programs from existing ones
- lack of useful mathematical properties for reasoning about programs

Backus: models for computing systems

- criteria for models
  - foundations
  - history sensitivity
  - type of semantics
  - clarity and conceptual usefulness of programs
- classification of models
  - simple operational models
  - applicative models
  - von Neumann models

"We shall be concerned only with applicative and von Neumann models." (in this paper)

Backus: on von Neumann

- the von Neumann computer
– the "intellectual parent" of conventional programming languages
– contains three parts:
  - ◇ central processing unit (CPU)
  - ◇ store
  - ◇ connecting tube
  - ⇒ *"the von Neumann bottleneck"*

"Not only is this tube a literal bottleneck for the data traffic of a problem, but, more importantly, it is an intellectual bottleneck that has kept us tied to word-at-a-time thinking..."

Backus: evil assignment statements

- at the center of typical programs

- splits programming into two worlds:

(1) the right side of assignment statements ⇒ the world of expressions
  – orderly

(2) the world of statements
  – disorderly

- alternative: applicative programs
– no assignment statements
– but also, no history, no storage

Backus: inner product example

- von Neumann language

```
c = 0;
for ( i=1; i<n; i++ ) {
  c += a[i] * b[i];
}
```

- functional language

```
(defun inner-product( a,b )
  (cond (nil a)
        0
        (+ (car(a)
            car(b)
            inner-product( cdr(a),cdr(b) )))))
```

(pseudo LISP...)

Backus: inner product example, in defense of the functional language

- operates only on its arguments

- is hierarchical

- is static and non-repetitive

- operates on whole conceptual units (not words)

- incorporates no data

- is general

Backus: language parts

- framework: overall rules of the system
- changeable parts: existence is anticipated by the framework, but particular behavior is not specified by it

"Suppose a language had a small framework which could accommodate a great variety of powerful features entirely as changeable parts. Then such a framework could support many different features and styles without being changed itself. In contrast to this pleasant possibility, von Neumann languages always seem to have an immense framework and very limited changeable parts."

⇒ functional programming languages

Backus: on functional programs

- *"the important point to remember is that no part of a function definition is a result itself. Instead, each part is a function that must be applied to an argument to obtain a result.*
- function f is a program
  object x is the contents of the store
  $f : x$ is the contents of the store after f is activated with x in the store
- BUT functional programs are not history-sensitive – no program can alter the library of programs

Backus: Applicative State Transition (AST) systems

- class of history-sensitive computing systems
- loosely coupled state-transition semantics in which a state transition occurs only once in a major computation
- simply structured state and simple transition rules
- depends heavily on underlying applicative system both to provide language of the system and to describe its state transitions
- structure:

(1) applicative subsystem
(2) state D, that is a set of definitions of (1)
(3) set of transition rules that describe how inputs are transformed into outputs and how state D changes

Backus

if nothing else, read the summary (p.639-640)

a really brief peek at lambda calculus

- founded by Alonzo Church (1932-33)
- the original applicative language
- $\lambda x.t(x)$ is the function f that assigns to the argument a the value $t(a)$
- simple examples:

identity: $\lambda x.x$, so $\lambda x.xa = a$

truth values: $\lambda xy.x =$ true, $\lambda xy.y =$ false

- after that, it quickly gets very complex...

John McCarthy

- inventor of LISP
- MIT, then Stanford University
- "History of Lisp" (1979)
- referenced by Backus (as one of the good guys)
- work began in the 1950's at IBM as FLPL (FORTRAN List Processing Language)
- Lisp is the second oldest programming language in widespread use today

McCarthy: characterization of Lisp

- computes with symbolic expressions rather than numbers
- represents symbolic expressions and other information by list structure in the memory of a computer
- represents information in external media mostly by multi-level lists and sometimes by s-expressions
- contains a small set of selector and constructor operations expressed as functions
- allows composition of functions as a tool for forming more complex functions
- uses conditional expressions for getting branching into function definitions
- uses recursive conditional expressions as a sufficient tool for buildling computable functions
- uses λ-expressions for naming functions
- represents Lisp programs as Lisp data
- allows conditional expression interpretation of Boolean connectives
- contains $eval$ function that serves both as a formal definition of the language and as an interpreter
- provides garbage collection for erasure

quiz

(1) Name 2 of the fathers and mothers of computing who lived prior to the 20th century, state approximately when they lived, and in one concise sentence (each), describe their contribution.

(3) A Turing machine M is defined as follows: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$. In general terms, what are Q, Γ, δ, q_0, B and F?
(e.g. Σ is the alphabet of input symbols)

(3) Is the following an example of a von Neumann language or an applicative language?

```
(defun factorial (a)
  (cond (= a 0)
        1
        (* a factorial(- (a 1)))))
```

for next lecture:

- read Codd (1970) *A Relational Model of Data for Large Shared Data Banks* (in green book)

- read Parnas (1972) *On the Criteria to Be Used in Decomposing Systems into Modules* (in green book)

the assignment will be distributed at the next lecture and will be due on 31 August.