# Evolution of the Computer

- refs: [O'H&Bryant, ch 1,3.1]; [Null&Lobur, ch 1]; [Tanembaum, Ch 1,2]; related web links
- history
- computer technology & and von Neumann architecture
- architecture
  - central processing unit (CPU)
  - arithmetic logical unit (ALU)
  - registers, memory
- instruction sets: complex vs 'reduced'
- modern CPU architecture

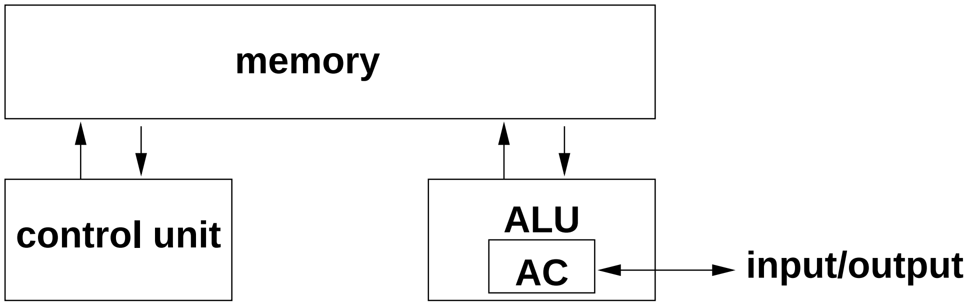
# Computer History

- 1642: Pascal built add/subtract machine
- 1672: Leibniz built add/sub/mul/div machine
- 1822: Babbage built a difference machine with punch card output.

  Later designs a general purpose analytical engine (1000 words of 50 decimal digits store), programmable in simple assembly language
- 1936: Zuse (Germany): automatic calculation machines using electromagnetic relays
- 1943: Colossus (UK, Alan Turing)
- 1946: ENIAC (Electronic Numerical Integrator and Computer)
- 1949: EDSAC (von Neumann architecture)
- 1960: PDP-1 (Digital)
- 1974: Intel 8080 (general purpose 8-bit microprocessor)
- 1981: IBM Personal Computer (PC)
- 1984: Apple Macintosh (Window system, mouse)

# Computer Technology

- First Generation (1945 – 1955): vacuum tubes and electromagnetic relays
- Second Generation (1955 – 1965): transistors
- Third Generation (1965 – 1980): integrated circuits
- Fourth Generation (1980 – ?): very large scale integration (VLSI)
- Moore's Law: # transistors on a chip doubles every 18 months
  - has other formulations; can this be sustained?
- von Neumann architecture
  - original architecture: bottleneck from the single accumulator register (AC)

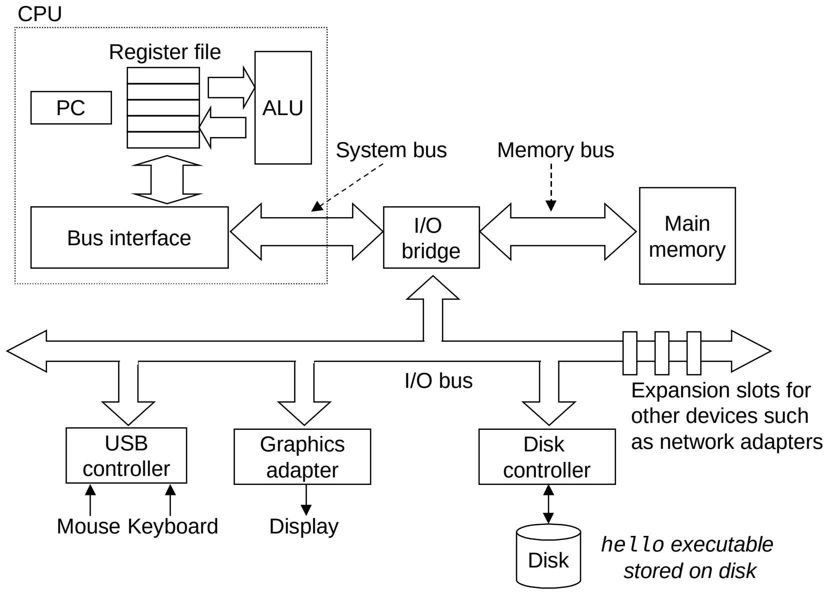



  - can add more registers to ALU, but overall CPU-memory bottleneck remains

# Computer Architecture

- CPU (central processing unit): executes programs stored in memory
- bus: communication links between the main components of a computer (many parallel wires to transmit a word or more)
- main memory: contains programs and data; volatile but fast (Random Access Memory - RAM)
- secondary storage: used to permanently store data and programs (hard disc, flash card, etc). Slower than RAM, but larger
- input/output (I/O) devices: exchange information with other computers or people (e.g. monitor, keyboard, CD-ROM, printer, network)

[O'H&Bryant, fig 1-4]

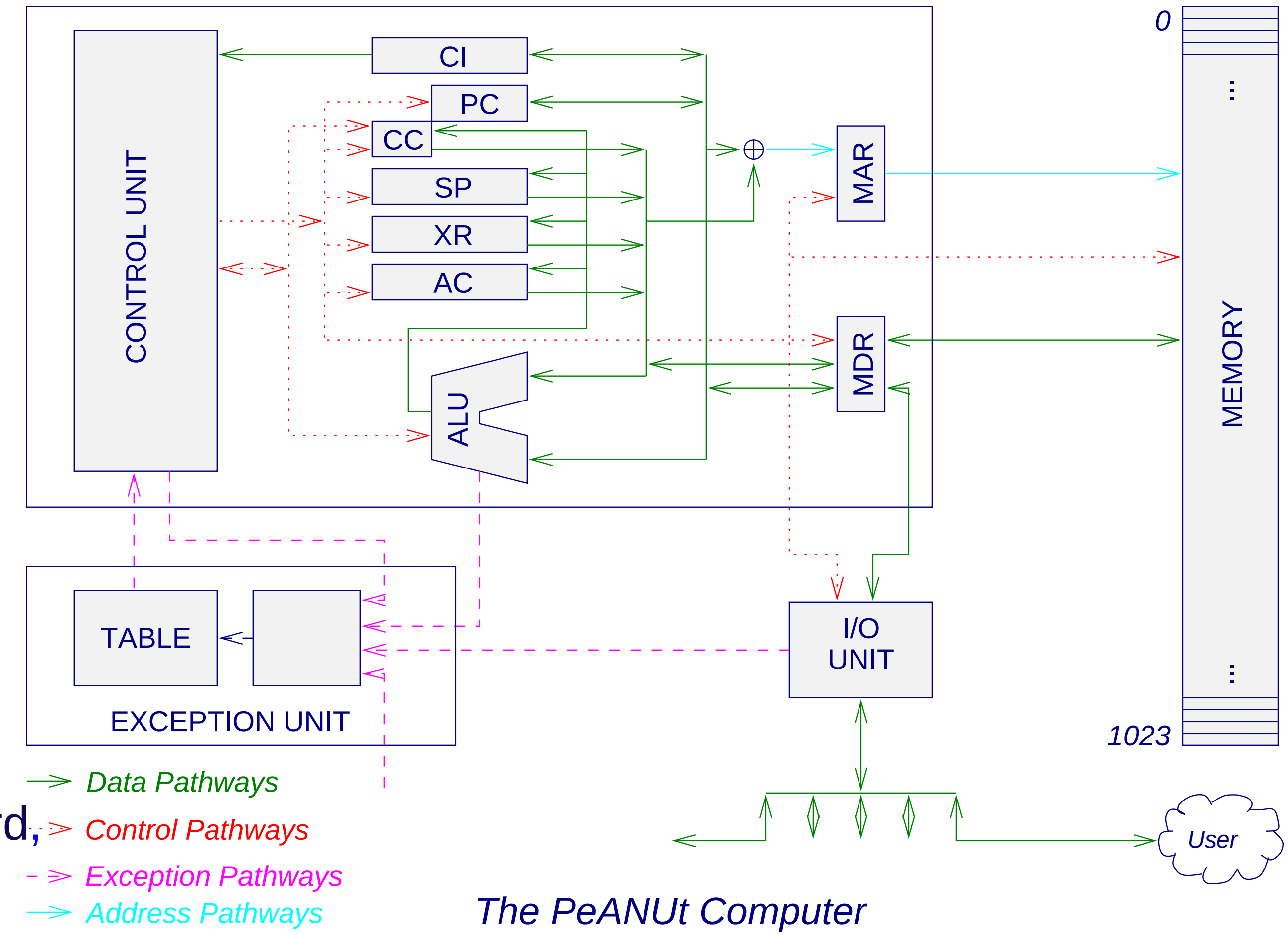


# The Central Processing Unit (CPU)

- control unit: responsible for fetching instructions from memory and initial decoding
- arithmetic logic unit (ALU): performs operations such as addition, multiplication, logical AND, logical OR, etc
  - the control unit determines which operation; also the inputs and outputs (registers and/or data)
- registers: small, very fast memory located in the CPU, used to store control information and temporary results
  - program counter (PC): stores address in memory of next instruction to be fetched and executed
  - instruction register (IR): holds current instruction
  - status register (SR): status after operation
    - e.g. results of last (numeric) comparison, overflow, division-by-zero, etc.
  - other registers (e.g. accumulator, index register, data register, etc)

# CPU Architecture

- e.g. architecture of the PeANUt microprocessor
- the data path indicates the flow of information
- communication events in a typical computer ([O'H&Bryant, figs 1.5-1.7]): read from keyboard, display string, load executable program



*The PeANUt Computer*

## Executing Machine Instructions

- instructions (of a program) are stored in memory at consecutive addresses
  - usually executed in sequence, with the control unit fetching the instruction using the address from the program counter (PC)
- this repeated process (fetch-decode-execute cycle) is central to CPU operation:

  1. fetch instruction from memory at address PC into instruction register (IR)
  2. update PC to point to the next instruction
  3. determine the type of instruction in the IR (decode)
  4. if it refers to data in memory, determine the address
  5. fetch the data (if any, into the memory data register)
  6. execute the instruction
  7. handle exceptions
  8. store the results into registers (or memory)
  9. go to step 1

# Executing Machine Instructions: Example

registers:

PC:
IR:
MAR:
MDR:

AC:

main memory

| address: | instruction: |
|---|---|
| ... | ... |
| 1A | `load` #$3_{10}$ |
| 1B | `add 2E` |
| 1C | `store 2E` |
| ... | ... |
| 2E | $4_{10}$ |

result at address $2E_{16}$ is $7_{10}$

# Instruction sets

- the collection of all instructions of the CPU available to the programmer is called instruction set
  - usually between 20 and 300 instructions
- typical instructions include:
  - load, store (data from/to memory)
  - add, subtract, multiply, divide (for integer and floating-point numbers)
  - and, or, not, xor (logical operators)
  - jumps (e.g. go to) and branch (e.g. if A then go to...)
  - call, return (for function calls)

# Complex Instruction Set Computer (CISC)

- has a large number of instructions, including many highly specialised ones, e.g.
  - supports many different addressing modes
  - supporting binary and (formerly) decimal arithmetic operations
  - support for generic loop and if-else operations, as well as specialized instructions for `for` loops, function calls
- 70s to early 90s processors, incl. Intel x86, Motorola 68k
- instructions require a lot of decoding (often done via microprogramming), and may take a long time (many clock cycles) to execute

# Reduced Instruction Set Computer (RISC)

- a smaller instruction set, with well-chosen instructions to support the most common operations (e.g. calls)
- load/store architecture: load/store is the only way to access memory
- fewer simpler instructions means less decoding and results in quicker execution (often one clock cycle!)
- complex and/or uncommon operations have to be written as sequence of instructions
  - e.g. integer multiplication in early RISC processors
  - but quicker execution makes it worthwhile
- idea developed in the 80s, and RISC dominated the high-end workstation market in the 90s
- includes SPARC (Sun), PowerPC (Macintosh); MIPS, Alpha

# Modern CPU Architectures

- more transistors on a chip enables complex RISC (hybrids like Pentium)
- on-chip cache memory
- pipelining: overlapping of fetch-decode-execute cycle
- superscalar architectures (on-chip parallelism):
  - execute different types of instructions (load/store, branch, integer, f.p.) simultaneously
  - can also have multiple ALUs and floating-point units (FPUs)
- this architecture has however 'hit the wall'! (can't usefully extend these techniques)