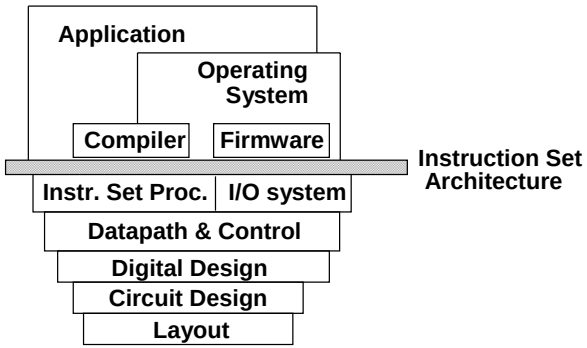
L2: Background Parts 1 & 2 – Historical Context & Performance

Outline

- Background 1: Brief Historical Context
  - Technological development
  - Milestones
- Background 2: Performance
  - Definitions of performance
    - Latency and Throughput
  - Components of processor performance:
    - Instruction count, CPI, clock period
  - Improving performance and limits on performance
    - Amdahl's Law

What is "Computer Architecture"?



- Coordination of many levels of abstraction
- Under a rapidly changing set of forces
- Design, Measurement, and Evaluation

Course Content

Computer Architecture and Engineering

Instruction Set Design	Computer Organization
Interfaces	Hardware Components
Compiler/System View	Logic Designer's View
-"Building Architect"	-"Construction Engineer"

Background 1: Historical Context

Computing Prehistory

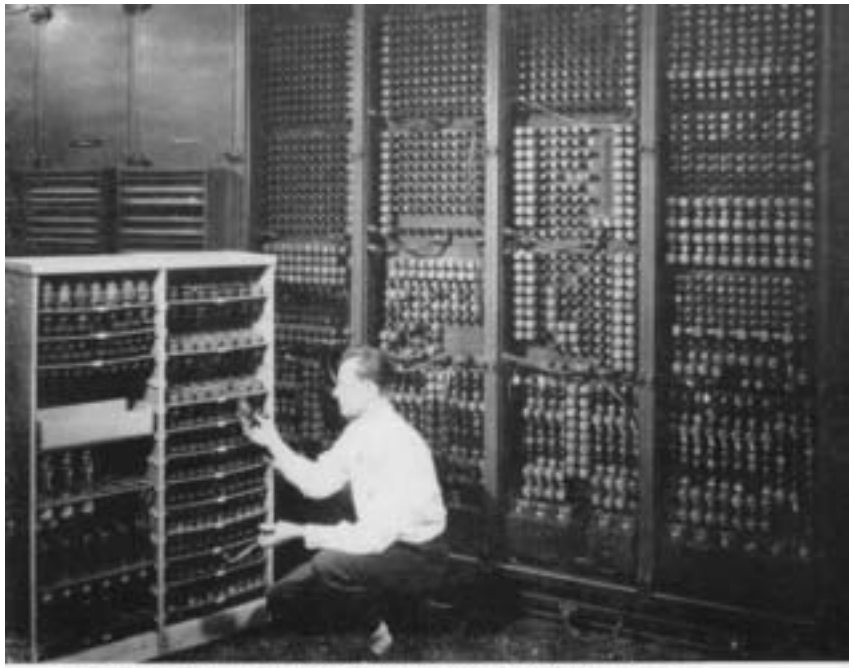
- Many influences from many cultures
- Modern computers have theoretical underpinnings in work of
  - George Boole – algebra
  - Ada Lovelace – instructing machines
  - …(many)
- Modern computers arose from practical experiments and earlier machines
  - The abacus
  - Charles Babbage's difference engine

ENIAC – dawn of electronic computation

- Electronic Numerical Integrator And Computer
- Eckert and Mauchly
- University of Pennsylvania
- Trajectory tables for weapons
- Started 1943
- Finished 1946
  - Too late for war effort
  - But early enough to be used in the design of the Hydrogen bomb
- Used until 1955
- But was it really the first?

ENIAC - details

- Decimal (not binary)
- 20 accumulators of 10 digits
- Programmed manually by setting switches and plugging and unplugging cables
- 18,000 vacuum tubes
- 30 tons
- 15,000 square feet
- 140 kW power consumption
- 5,000 additions per second

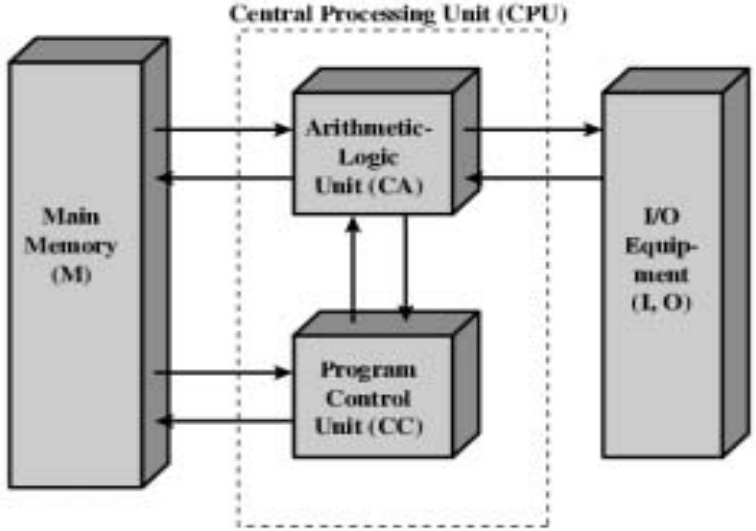


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

von Neumann/Turing

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies
  - IAS begun 1946 & completed 1952
  - Became prototype for all subsequent "general purpose" computers

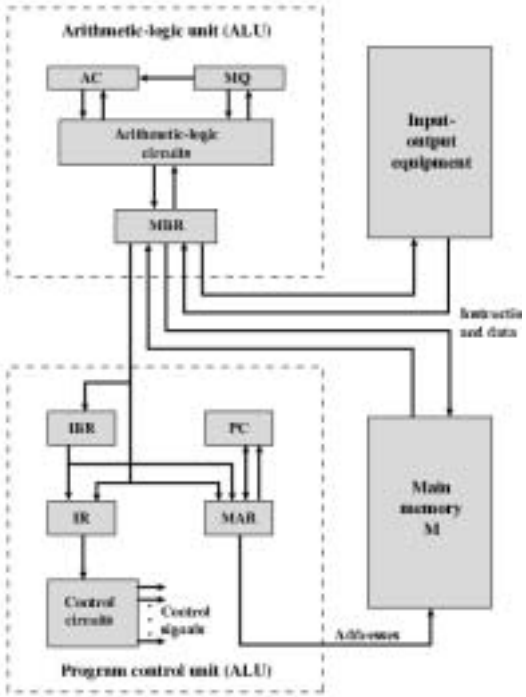
Structure of von Neumann machine



IAS - details

- 1000 x 40 bit words
  - Used binary number system
  - 2 x 20 bit instructions - 8-bit opcode; 12-bit address/operand
- Set of registers (storage in CPU)
  - Memory Buffer Register
  - Memory Address Register
  - Instruction Register
  - Instruction Buffer Register - stores right hand instruction
  - Program Counter - pointer to next instruction pair
  - Accumulator
  - Multiplier Quotient - least significant word of mult/div

Structure of IAS – detail



Commercial Computers

- 1947 - Eckert-Mauchly Computer Corporation
- UNIVAC I (Universal Automatic Computer)
  - Scientific & commercial applications e.g., matrix algebra, statistical problems, premium billings for life insurance companies, and logistical problems
- US Bureau of Census 1950 calculations
- Became part of Sperry-Rand Corporation
- Late 1950s - UNIVAC II
  - Faster
  - More memory


Univac I, Number 1 as it appeared after being built at 3747 Ridge Avenue, in Philadelphia, 1951. (Photo courtesy of Bernie Victor.)

IBM – a "late" entrant

- Punched-card processing equipment
- 1953 - the 701
  - IBM's first stored program computer
  - Scientific calculations
- 1955 - the 702
  - Business applications
- Lead to 700/7000 series

Transistors – 2nd Generation electronic computer

- Replaced vacuum tubes
- Smaller
- Cheaper
- Less heat dissipation
- Solid State device
- Made from Silicon (Sand)
- Invented 1947 at Bell Labs
- William Shockley et al.
- Completely transistorized computers by late '50s

Transistor Based Computers

- Second generation machines
- NCR & RCA produced small transistor machines
- IBM 7000 followed
- DEC - 1957
  - Produced PDP-1

Microelectronics

- Literally - "small electronics"
- The computer's gates, memory cells, and interconnections can be fabricated on a semiconductor, e.g. silicon wafer
- Start of integrated circuit design – 3rd generation

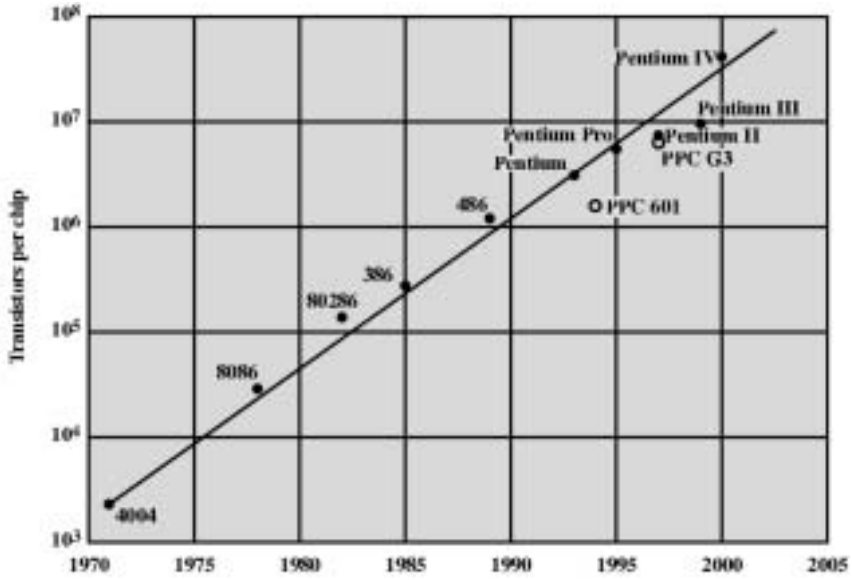
Computer generations

1. Vacuum tube: 1946-1957
2. Transistor: 1958-1964
3. Integrated Circuit (IC): 1965+
   - Small scale integration (SSI): 1965 on
     - Up to 100 devices on a chip
   - Medium scale integration (MSI): to 1971
     - 100-3,000 devices on a chip
   - Large scale integration (LSI): 1971-1977
     - 3,000 - 100,000 devices on a chip
   - Very large scale integration (VLSI): 1978 to date
     - 100,000 - 100,000,000 devices on a chip
   - Ultra large scale integration (ULSI): Now +
     - Over 100,000,000 devices on a chip

Moore's Law

- Relates to the increased density of components on chip
- Discovered by Gordon Moore (cofounder of Intel)
- Number of transistors on a chip will double every year
- Since 1970's development has slowed a little
  - Number of transistors doubles every 18 months
- Cost of a chip has remained almost unchanged
- Higher packing density means shorter electrical paths, giving higher performance
- Fewer interconnections increases reliability
- Reduced power and cooling requirements
- Smaller size gives increased flexibility

Growth in CPU Transistor Count



Computer generations

1. Vacuum tube: 1946-1957
2. Transistor: 1958-1964
3. Integrated Circuit (IC): 1965+
   - Small scale integration (SSI): 1965 on
     - Up to 100 devices on a chip
   - Medium scale integration (MSI): to 1971
     - 100-3,000 devices on a chip
   - Large scale integration (LSI): 1971-1977
     - 3,000 - 100,000 devices on a chip
   - Very large scale integration (VLSI): 1978 to date
     - 100,000 - 100,000,000 devices on a chip
   - Ultra large scale integration (ULSI): Now +
     - Over 100,000,000 devices on a chip

Computer generations

1. Vacuum tube: 1946-1957
2. Transistor: 1958-1964
3. Integrated Circuit (IC): 1965+
   - Small scale integration (SSI): 1965 on
     - Up to 100 devices on a chip
   - Medium scale integration (MSI): to 1971
     - 100-3,000 devices on a chip
   - Large scale integration (LSI): 1971-1977
     - 3,000 - 100,000 devices on a chip
   - Very large scale integration (VLSI): 1978 to date
     - 100,000 - 100,000,000 devices on a chip
   - Ultra large scale integration (ULSI): Now +
     - Over 100,000,000 devices on a chip
4. ??? (Choose from Quantum, Nanotech, Biomolecular,…)

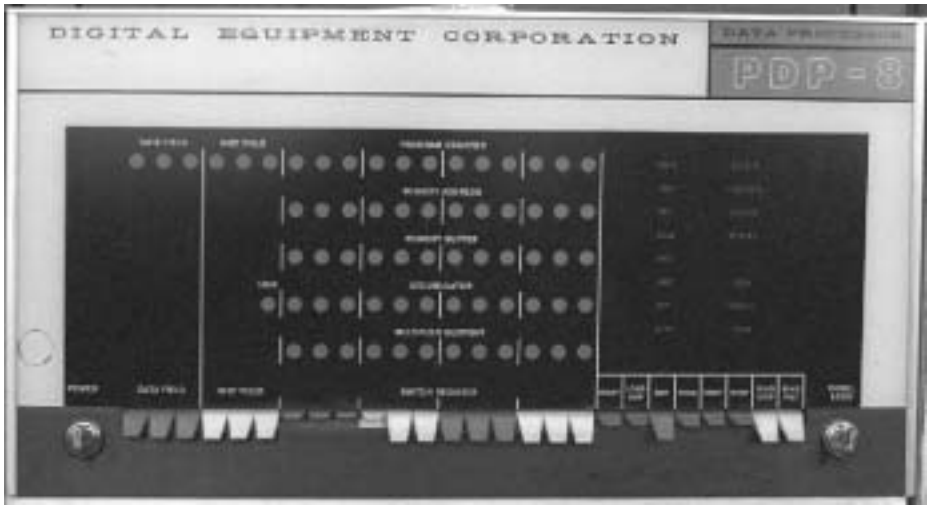
IBM 360 series – thrust IBM to forefront

- 1964 (birth of the modern era of computer design?)
- Replaced (& not compatible with) 7000 series
- First planned "family" of computers
  - Similar or identical instruction sets
  - Similar or identical O/S
  - Increasing speed
  - Increasing number of I/O ports (i.e. more terminals)
  - Increased memory size
  - Increased cost
- Multiplexed switch structure (costly)

DEC PDP-8

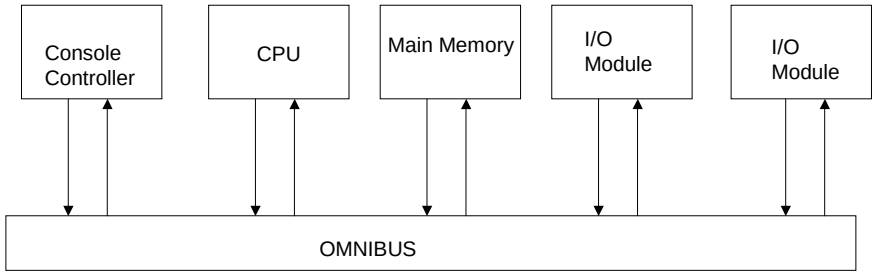
- Also 1964!
- First minicomputer (after miniskirt!)
- Did not need air conditioned room
- Small enough to sit on a lab bench
- $16,000
  - $100k+ for IBM 360
- Leant itself to embedded applications & resale by third parties
- Bus replaces switched interconnect

PDP-8



DEC – PDP-8 Bus Structure

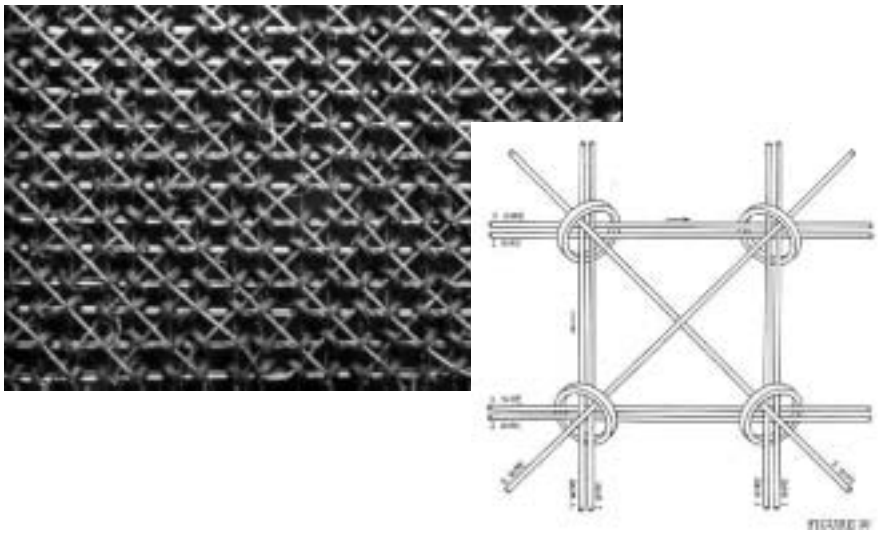
- 96 signal paths for control, address, and data arbitrated by the CPU



Semiconductor Memory

- 1970
- Fairchild
- Size of a single core
  - i.e. 1 bit of magnetic core storage
- Held 256 bits
- Non-destructive read
- Much faster than core
- Initially more expensive than core
- Capacity approximately doubles each year
- Became cheaper than core in ‘74

Core memory



Intel – an original Silicon Valley start-up

- 1971 - 4004
  - First microprocessor
  - All CPU components on a single chip
  - 4 bit
- Followed in 1972 by 8008
  - 8 bit
  - Both designed for specific applications
- 1974 - 8080
  - Intel’s first general purpose microprocessor
- 1979 – 8086
  - Ancestor of Pentium

Technology ⇒ dramatic change

- Processor
  - Logic capacity: about 30% per year
  - Clock rate: about 20% per year
- Memory
  - DRAM capacity: about 60% per year (4x every 3 years)
  - Memory speed: about 10% per year
  - Cost per bit: about 25% per year
- Disk
  - Capacity: about 60% per year

Speeding it up

- Pipelining
- On-chip cache
- Branch prediction
- On-chip L1 & L2 cache
- Data flow analysis
- Speculative execution
- Superscalar architecture
- Massively parallel machines

Background 2: Performance

How do we quantify improvements? How do we compare performance?

- Two principle measures:
  1. Response time/execution time/elapsed time/latency
     - How long does it take a job to run?
     - How long does it take to execute a task?
     - How long does it take to return a query?
  2. Throughput
     - How many jobs can be run at once?
     - What is the average execution rate?
     - How much work is done?
- Examples:
  - Upgrading a machine with a new processor
  - Adding a new machine to the lab

Response time /execution time

- Elapsed time = finish time – start time
  - Counts everything (disk and memory access, I/O etc)
  - Not so good for purpose of comparison $\Rightarrow$ depends upon machine load, location of data
- CPU time
  - Doesn't count I/O time or time spent running other programs by the CPU
  - Can be broken up into system time, and user time
  - But system time includes waits & OS overheads
- Our focus: user CPU time
  - Time spent executing the lines of code that are "in" our program

Unix **`time`** command

- `% time find . - name myfile`
  `90.7u 12.9s 2:39 65%`
  - 90.7u —— user CPU time in seconds
  - 12.9s —— system CPU time in seconds
  - 2:39 —— response time
  - (90.7 + 12.9)/(2*60+39) = 65%
  - 65% of the elapsed time was used by the CPU executing the program – referred to as “utilization”

What is “performance”?

- Performance is in units of things executed per second
  - Higher values intuitively better
- If we are primarily concerned with response time, the performance of machine X is
  performance(X) = 1/execution_time(X)
- “X is n times faster than Y" means
  - *performance(X)/performance(Y) = n*
  - *execution_time(Y)/execution_time(X) = n*

Aspects of CPU performance

- What are the factors that determine CPU performance?

Aspects of CPU performance

- Evaluated based on the CPU time of a certain program

$$\text{cpu\_time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- CPU time
  - Time in seconds the CPU spends executing a program
  - Comprised of three key components
    - instruction count = instructions/program
    - CPI = cycles/instruction
    - clock period (or cycle time)= seconds/cycle
      - clock frequency = 1/(clock period)

Aspects of CPU performance

- The various levels of computer design affect each aspect differently

Design level	Instr. count	CPI	Clock period
program	X		
compiler	X	X	
instr. set	X	X	
organization		X	X
technology			X

CPI – average cycles per instruction

$$cpu\_time = clock\_period * \sum_{i=1}^{n} CPI_i * I_i$$

Where n = the number of instructions in the ISA
I_i = instruction count for instruction i
CPI_i = number of clock cycles for instruction i

$$\begin{aligned} CPI &= (cpu\_time * clock\_frequency) / instr\_count \\ &= (cpu\_time / clock\_period) / instr\_count \\ &= cycles / instr\_count \\ &= \sum_{i=1}^{n} CPI_i * \frac{I_i}{instr\_count} \end{aligned}$$

Example: Comparing code segments

- A compiler writer must decide between two code sequences for a machine given following facts:

Instruction Class	CPI for this inst. class
A	1
B	2
C	3

- The two code sequences require the following instruction counts:

Code sequence	A	B	C
1	2	1	2
2	4	1	1

Example continued

- Which code sequence executes the most instructions?

Code sequence	A	B	C
1	2	1	2
2	4	1	1

Example continued

- Which code sequence is faster?

Instruction Class	CPI for this inst. class
A	1
B	2
C	3

Code sequence	A	B	C
1	2	1	2
2	4	1	1

- CPU clock cycles$_1$ = 2x1 + 1x2 + 2x3 = 10 cycles
- CPU clock cycles$_2$ = 4x1 + 1x2 + 1x3 = 9 cycles

Example continued

- What is the CPI for each sequence?
- CPI = (CPU clock cycles)/(Instruction count)
- CPI$_1$ = 10/5 = 2
- CPI$_2$ = 9/6 = 1.5

- This example shows the danger of using only one factor (instruction count) to assess performance – all three components must be considered when comparing machines on the basis of execution time

Improving performance

- Given an instruction set, CPU performance can be improved by
  - Increases in the clock rate
  - Improvements in processor organization that lower the CPI
  - Compiler enhancements that lower the instruction count or generate instructions with a lower CPI
- Improvements in one aspect may affect other aspects e.g. compiler that replaces sequence 2 with 1 in previous example reduces the instruction count only to increase CPI

Speedup

- Speedup due to enhancement *E*:

$$speedup(E) = \frac{ex\_time(\ )}{ex\_time(E)} = \frac{performance(E)}{performance(\ )}$$

Amdahl's Law

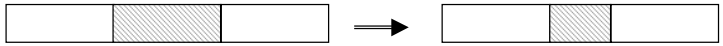
"The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used"

Amdahl's Law

- Suppose that enhancement *E* accelerates a fraction *F* of the task by a factor *S* and the remainder of the task is unaffected. Then

$$ex\_time(E) = ((1-F) + F/S) \times ex\_time(\ )$$

$$\text{ex time after improv} = \frac{\text{ex time affected by improv}}{\text{amount of improv}} + \text{ex time unaffected}$$



- Speedup is limited by the amount that the improved feature is used

$$speedup(E) = \frac{1}{(1-F) + F/S}$$

Example: Amdahl's Law

- Suppose a program runs in 100 seconds on a machine, with multiply operations responsible for 80 seconds of this time. How much does the speed of the multiplication operation have to be improved for the program to run five times faster?

$$\frac{100}{5} = \frac{80}{x} + 20$$

$$x \to \infty$$

Next week:

- Tutes & Labs commence
- Quiz Wednesday!
- Friday's lecture: Ch. 3 Instruction Set Architecture