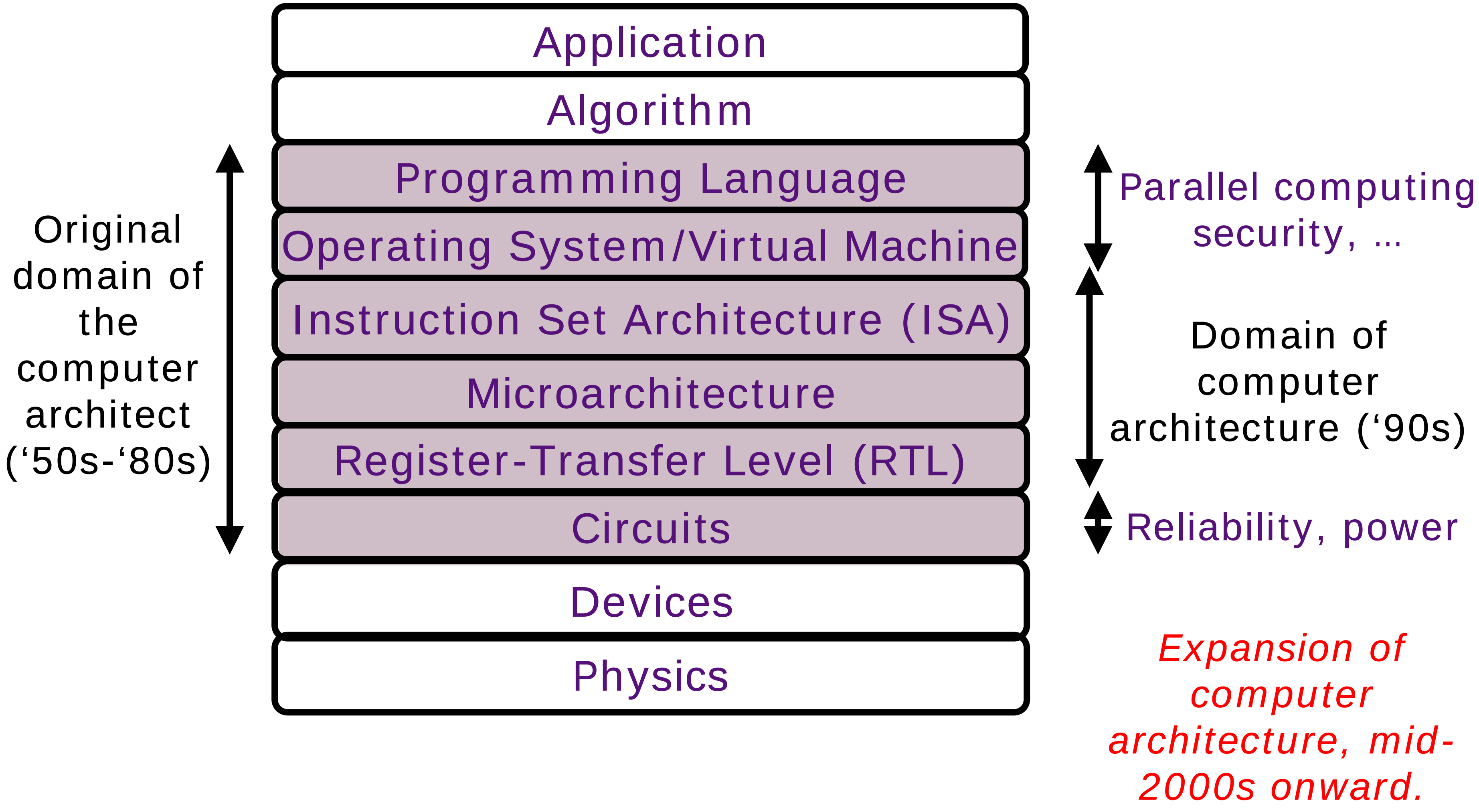
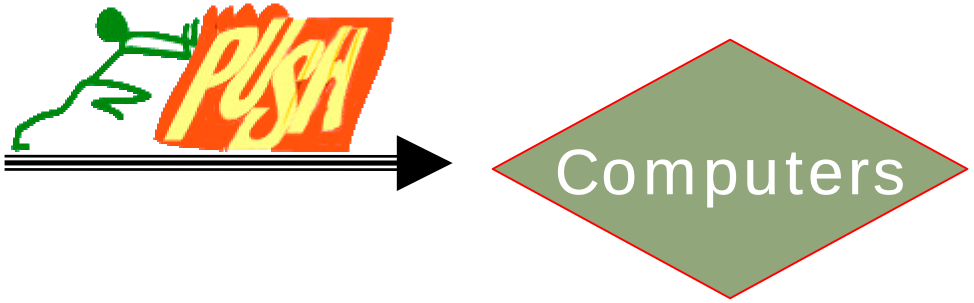
Computer Architecture is the *design of the abstraction layers*


Application
Algorithm
Programming Language
Operating System/Virtual Machine
Instruction Set Architecture (ISA)
Microarchitecture
Register-Transfer Level (RTL)
Circuits
Devices
Physics
Original domain of the computer architect ('50s-'80s)
Parallel computing, security, …
Domain of computer architecture ('90s)
Reliability, power
Expansion of computer architecture, mid-2000s onward.

Technology is the dominant factor in computer design

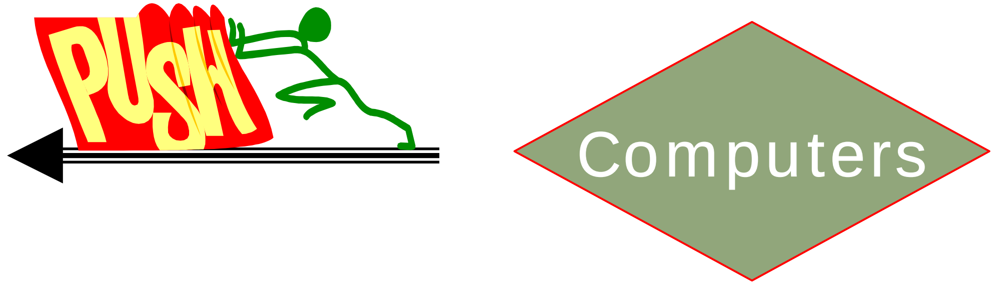
Technology

Transistors
Integrated circuits
VLSI (initially)
Flash memories, ...



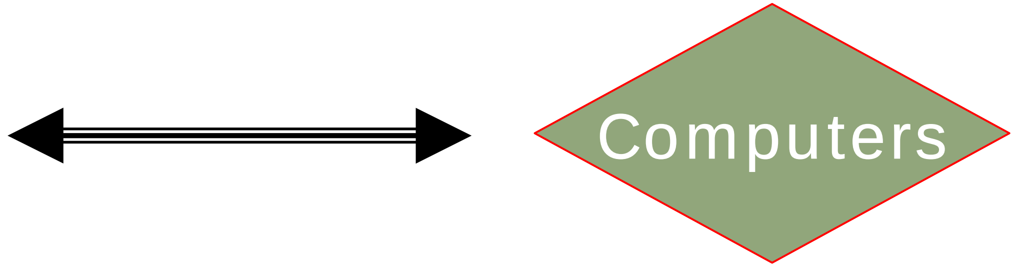
Technology

Core memories
Magnetic tapes
Disks



Technology

ROMs, RAMs
VLSI
Packaging
Low Power

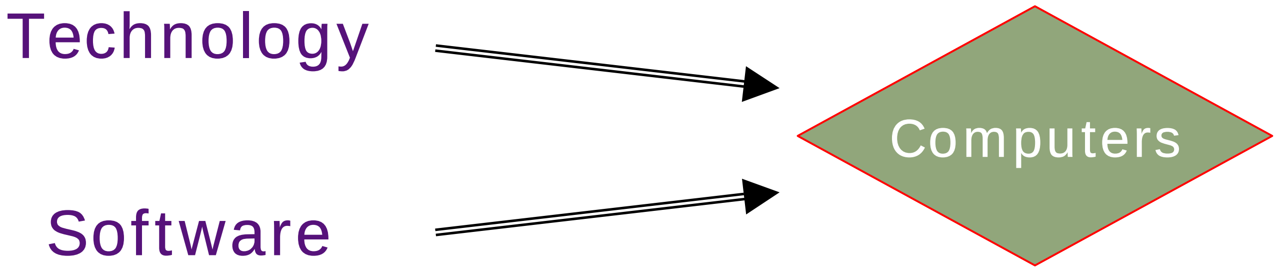


But Software...

As people write programs and use computers, our understanding of *programming* and *program behavior* improves.

This has profound though slower impact on computer architecture

Modern architects cannot avoid paying attention to software and compilation issues.



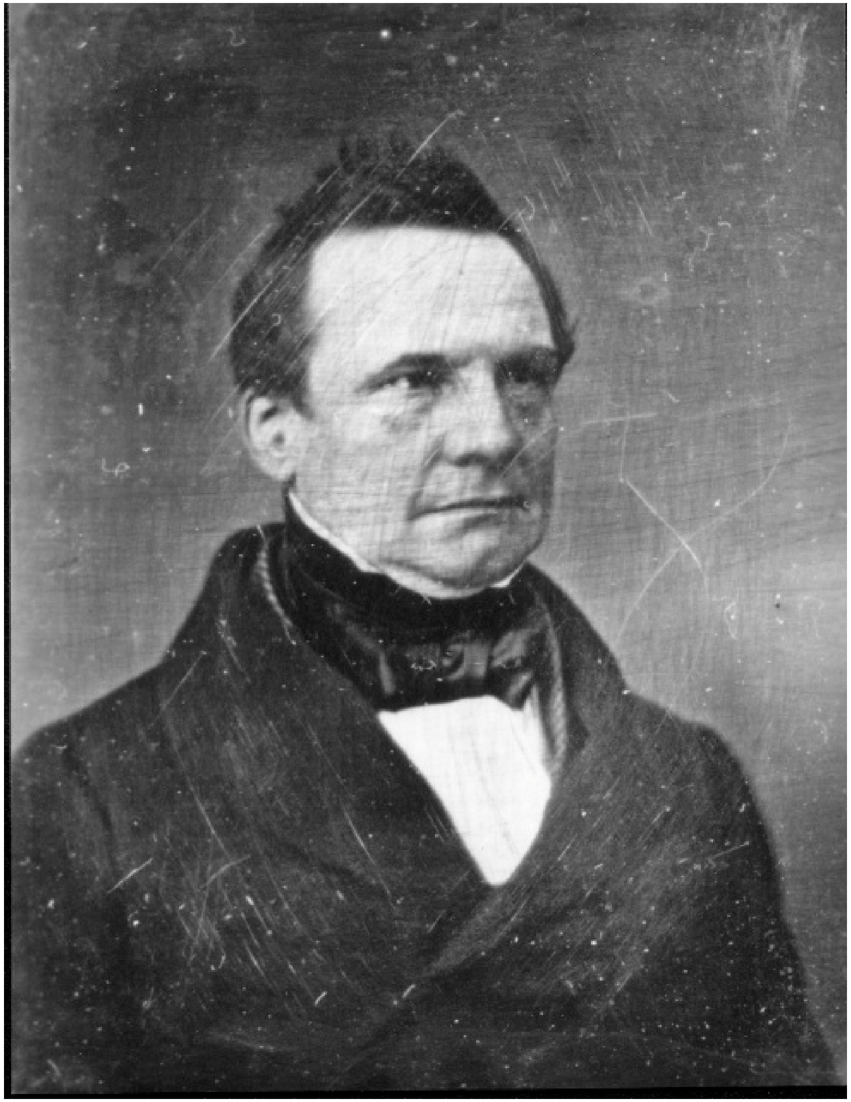
Architecture is Engineering Design under constraints

Factors to consider:

- Performance of whole system on target applications
  - Average case & worst case
- Cost of manufacturing chips and supporting system
- Power to run system
  - Peak power & energy per operation
- Reliability of system
  - Soft errors & hard errors
- Cost to design chips (engineers, computers, CAD tools)
  - Becoming a limiting factor in many situations, fewer unique chips can be justified
- Cost to develop applications and system software
  - Often the dominant constraint for any programmable device

At different points in history, and for different applications at the same point in time, the relative balance of these factors can result in widely varying architectural choices

Charles Babbage 1791-1871
Lucasian Professor of Mathematics, Cambridge University, 1827-1839



The first programmer
Ada Byron *aka* “Lady Lovelace” 1815-52

Ada’s tutor was Babbage himself!

Babbage’s Influence

- Babbage’s ideas had great influence later primarily because of
  - *Luigi Menabrea,* who published notes of Babbage’s lectures in Italy
  - *Lady Lovelace,* who translated Menabrea’s notes in English and thoroughly expanded them.
    “... Analytic Engine weaves *algebraic patterns*....”

- In the early twentieth century - the focus shifted to analog computers but
  - *Harvard Mark I built in 1944 is very close in spirit to the Analytic Engine.*

Harvard Mark I

- Built in 1944 in IBM Endicott laboratories
  - Howard Aiken – Professor of Physics at Harvard
  - Essentially mechanical but had some electro-magnetically controlled relays and gears
  - Weighed *5 tons* and had *750,000* components
  - A synchronizing clock that beat every *0.015* seconds

Performance:

0.3	seconds for addition
6	seconds for multiplication
1	minute for a sine calculation

Broke down once a week!

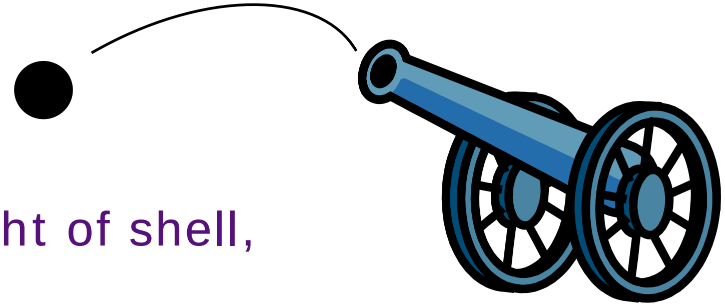
Electronic Numerical Integrator and Computer (ENIAC)

- Designed and built by Eckert and Mauchly at the University of Pennsylvania during 1943-45
- The first, completely electronic, operational, general-purpose analytical calculator!
  - 30 tons, 72 square meters, 200KW
- Performance
  - Read in 120 cards per minute
  - Addition took 200 μs, Division 6 ms
  - 1000 times faster than Mark I
- Not very reliable!

WW-2 Effort

Application: Ballistic calculations

angle = f (location, tail wind, cross wind,
air density, temperature, weight of shell,
propellant charge, ...)



Electronic Discrete Variable Automatic Computer (EDVAC)

- ENIAC’s programming system was external
  - Sequences of instructions were executed independently of the results of the calculation
  - Human intervention required to take instructions “out of order”
- EDVAC was designed by Eckert, Mauchly and von Neumann in 1944 to solve this problem
  - Solution was the *stored program computer*

    $\Rightarrow$ *“program can be manipulated as data”*
- *First Draft of a report on EDVAC* was published in 1945, but just had von Neumann’s signature!
  - Without a doubt the most influential paper in computer architecture

Stored Program Computer

Program = A sequence of instructions

How to control instruction sequencing?

manual control	calculators
automatic control	
external (paper tape)	Harvard Mark I , 1944 Zuse's Z1, WW2
internal	
plug board	ENIAC 1946
read-only memory	ENIAC 1948
read-write memory	EDVAC 1947 *(concept)*

– The same storage can be used to store program and data

The Spread of Ideas

ENIAC & EDVAC had immediate impact

brilliant engineering: Eckert & Mauchley
lucid paper: Burks, Goldstein & von Neumann

IAS	Princeton	46-52	Bigelow
EDSAC	Cambridge	46-50	Wilkes
MANIAC	Los Alamos	49-52	Metropolis
JOHNIAC	Rand	50-53	
ILLIAC	Illinois	49-52	
	Argonne	49-53	
SWAC	UCLA-NBS		

UNIVAC - the first commercial computer, 1951

Alan Turing's direct influence on these developments is often debated by historians.

And then there was IBM 701

IBM 701 -- 30 machines were sold in 1953-54

IBM 650 -- a cheaper, drum based machine,
more than 120 were sold in 1954
and there were orders for 750 more!

Users stopped building their own machines.

Why was IBM late getting into computers?

IBM was making too much money!
Even without computers, IBM revenues
were doubling every 4 to 5 years in 40's
and 50's.

Software Developments

up to 1955 Libraries of numerical routines

- Floating point operations
- Transcendental functions
- Matrix manipulation, equation solvers, . . .

1955-60 *High level Languages* - Fortran 1956

Operating Systems -

- Assemblers, Loaders, Linkers, Compilers
- Accounting programs to keep track of usage and charges

Machines required *experienced operators*

$\Rightarrow$ Most users could not be expected to understand these programs, much less write them

$\Rightarrow$ Machines had to be sold with a lot of resident software

Linear Equation Solver
John Atanasoff, Iowa State University

1930's:
- Atanasoff built the Linear Equation Solver.
- It had 300 tubes!

Application:
- Linear and Integral differential equations

Background:
- Vannevar Bush's Differential Analyzer
  --- *an analog computer*

Technology:
- Tubes and Electromechanical relays

Atanasoff decided that the correct mode of computation was by electronic digital means.

Computers in mid 50's

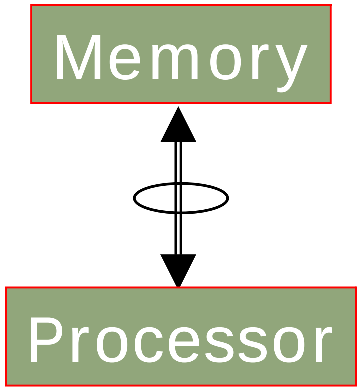
- Hardware was expensive
- Stores were small (1000 words)

  ⇒ No resident system-software!
- Memory access time was 10 to 50 times slower than the processor cycle

  ⇒ Instruction execution time was totally dominated by the *memory reference time*.
- The *ability to design complex control circuits* to execute an instruction was the central design concern as opposed to *the speed* of decoding or an ALU operation
- Programmer's view of the machine was inseparable from the actual hardware implementation

Processor-Memory Bottleneck: Early Solutions

- Fast local storage in the processor
  - 8-16 registers as opposed to one accumulator
- Indexing capability
  - to reduce book keeping instructions
- Complex instructions
  - to reduce instruction fetches
- Compact instructions
  - implicit address bits for operands, to reduce instruction fetches



Processor State

The information held in the processor at the end of an instruction to provide the processing context for the next instruction.

Program Counter, Accumulator, . . .

Programmer visible state of the processor (and memory) plays a central role in computer organization for both hardware and software:

- *Software* must make efficient use of it
- If the processing of an instruction can be interrupted then the *hardware* must save and restore the state in a transparent manner

Programmer's machine model is a contract between the hardware and software

Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Defines set of programmer visible state
  - Defines instruction format (bit encoding)
  - Defines instruction semantics
  - Examples: *MIPS, Alpha, x86, IBM 360, VAX, ARM, JVM*

- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z900 (c. 2001)
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4 (c. 2000), AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC*
  - MIPS implementations: *R2000, R4000, R10000, ...*
  - JVM: *HotSpot, PicoJava, ARM Jazelle, ...*

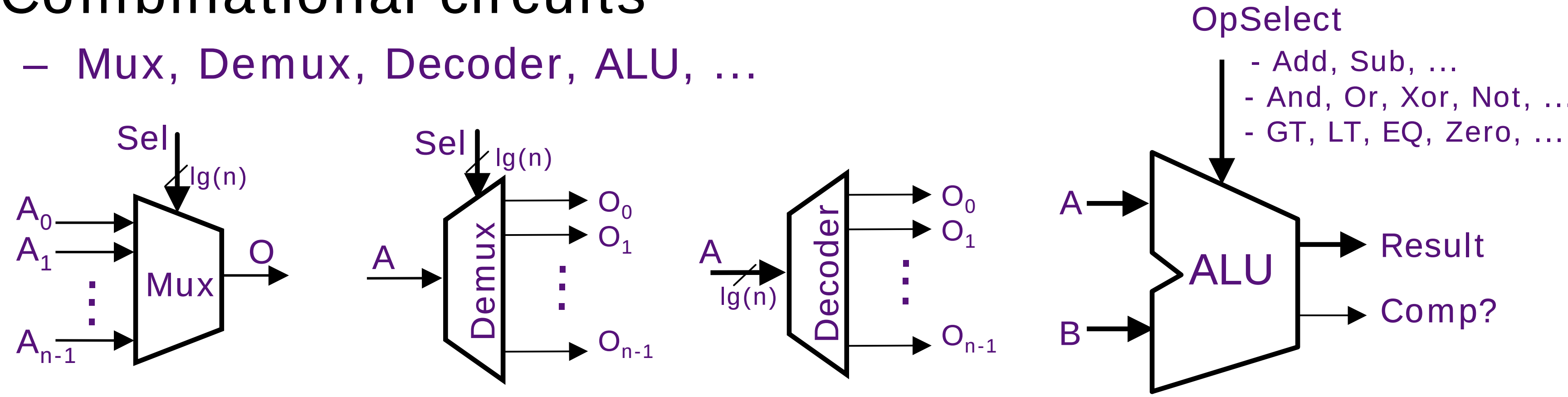
Processor Performance

$$\boxed{\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}}$$

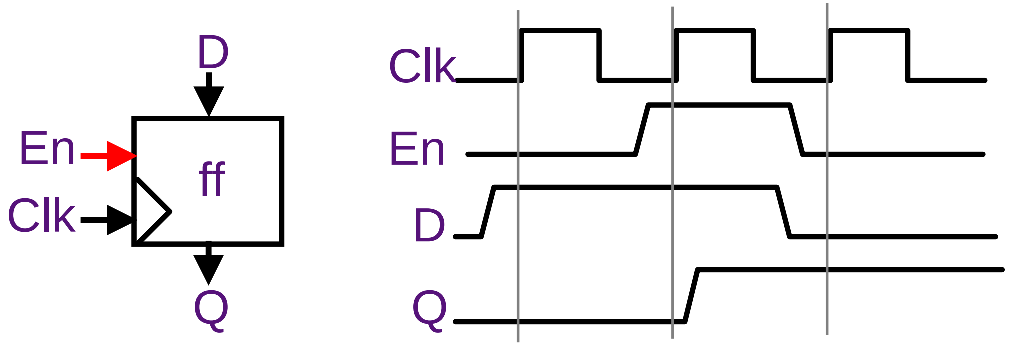
- – Instructions per program depends on source code, compiler technology and ISA
- – Cycles per instructions (CPI) depends upon the ISA and the microarchitecture
- – Time per cycle depends upon the microarchitecture and the base technology

Hardware Elements

- Combinational circuits
  - Mux, Demux, Decoder, ALU, ...

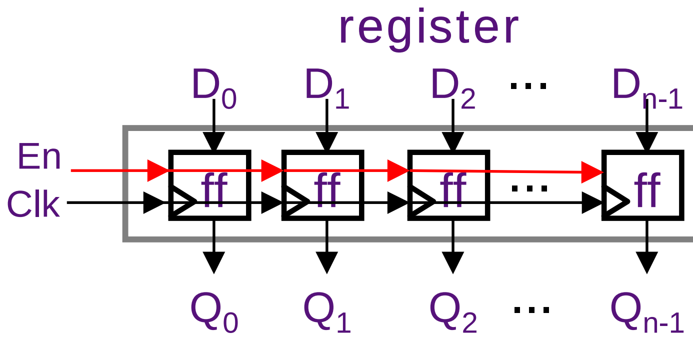


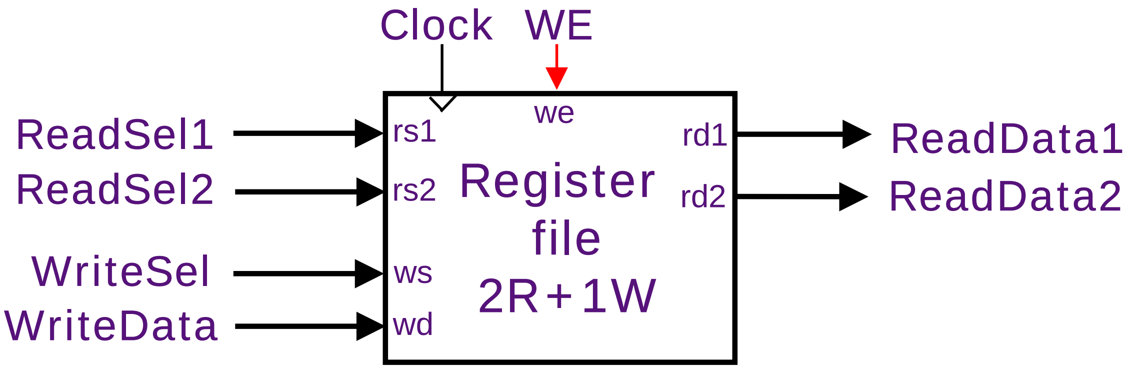
- Synchronous state elements
  - Flipflop, Register, Register file, SRAM, DRAM



Edge-triggered: Data is sampled at the rising edge

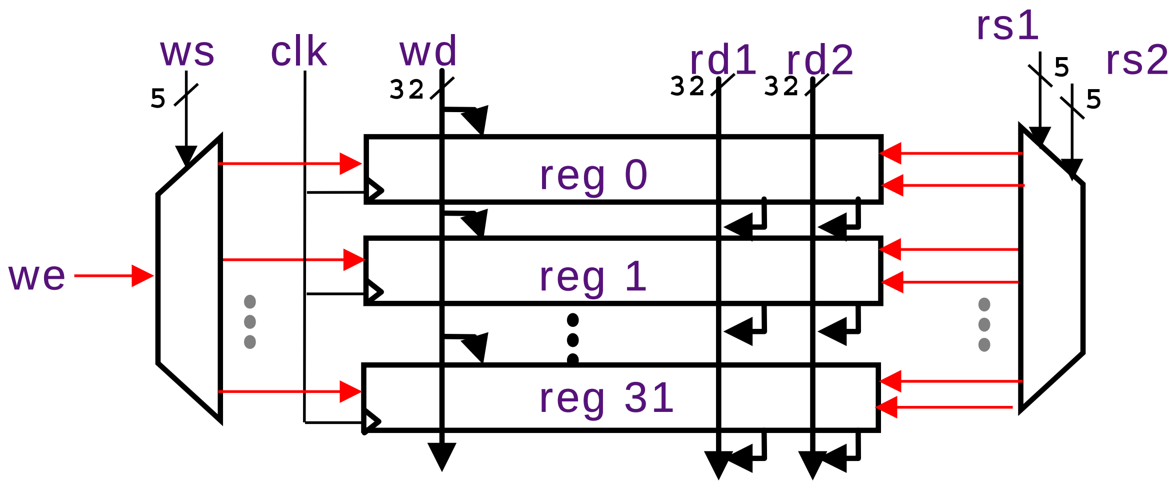
Register Files





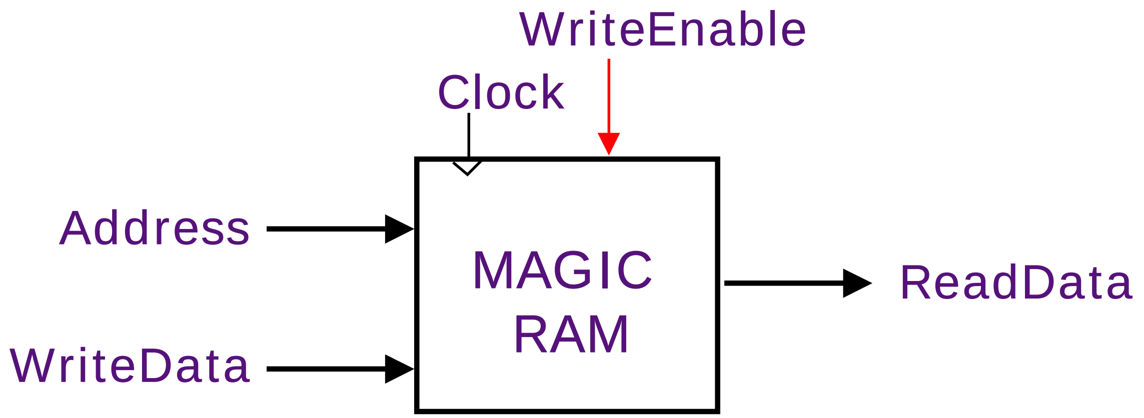
- No timing issues in reading a selected register

Register File Implementation



- Register files with a large number of ports are difficult to design
  - Almost all Alpha instructions have exactly 2 register source operands
  - *Intel's Itanium, GPR File has 128 registers with 8 read ports and 4 write ports!!!*

A Simple Memory Model



Reads and writes are always completed in one cycle

- a Read can be done any time (i.e. combinational)
- a Write is performed at the rising clock edge if it is enabled

$\Rightarrow$ *the write address and data must be stable at the clock edge*

Instruction Execution

Execution of an instruction involves

1. instruction fetch
2. Decode
3. Register fetch
4. ALU operation
5. memory operation (optional)
6. write back

and the computation of the address of the *next instruction*

Instruction Sets

- what is an instruction set?
- what is a *good* instruction set?
- the forces that shape instruction sets
- aspects of instruction sets
- instruction set examples
- RISC vs. CISC

Instruction Sets

"Instruction set architecture is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct (timing independent) program for that machine"

–IBM introducing 360 (1964)

an instruction set specifies a processor's *functionality*

- what operations it supports
- what storage mechanisms it has & how they are accessed
- programmer/compiler use to communicate programs to processor

What Makes a Good Instruction Set?

implementability

- supports a (performance/cost) *range* of implementations
  - implies support for *high performance* implementations

programmability

- easy to express programs

backward/forward/upward compatibility

- implementability & programmability *across generations*
  - e.g., x86 generations: 8086, 286, 386, 486, Pentium, Pentium II, Pentium III, Pentium 4...

Implementability

low performance implementation

- easy, trap to software to emulate complex instructions

high performance implementation

- more difficult
- components: pipelining, parallelism, dynamic scheduling?
  - avoid *artificial* sequential dependences
  - deterministic execution latencies simplify scheduling
  - avoid instructions with *multiple* long latency components
  - avoid not-easily-interruptable instructions

Programmability

programmability timeline

- –1975: most code was hand-assembled
- 1975–1985: most code was compiled
  - but people thought that hand assembled code was superior
- 1985–: most code was compiled
  - and compiled code was at least as good as hand-assembly
- big shift in what “programmability” meant

–1980: Human Programmability

focus: instruction sets that were easy for *humans* to program

- ISA sematically close to high-level language (HLL)
  - closing the "semantic gap"
- semantically heavy (CISC-like) instructions
  - automatic saves/restores on procedure calls
  - VAX insque
- people thought computers may execute HLL directly
  - never materialized
- one problem with this approach: multiple HLLs
  - "semantic clash": not exactly the semantics you want

1980–: Compiler Programmability

focus: instruction sets that are easy for compilers to compile to

- primitive instructions from which solutions are synthesized
  - Wulf: primitives not solutions
  - hard for compiler to tell if complex instruction fits situation
- regularity: do things the same way, consistently
  - “principle of least astonishment” (true even for hand-assembly)
  - one vs. all (either one way for all things, or one way for each thing)
- orthogonality, composability
  - all combinations of operation, data type, addressing mode possible
- few modes/obvious choices
  - compilers do giant case analysis, don’t add more cases

Today’s Semantic Gap

popular argument: today’s ISAs are targeted to one HLL, it just so happens that this HLL (C) is very low-level (assembly++)

- would ISAs be different if Java was dominant?
  - more object oriented?
  - GC support?
  - support for bounds-checking?
  - security support?

Compatibilities: Upward/Forward/Backward

basic tenet: make sure all written software works

- business reality: software cost greater than hardware cost
  - intel first company to realize this

thinking about compatibility ahead of time is hard

- temptation: use ISA gadget for 5% performance gain
- frequent outcome: must continue to support gadget
  - even if gain disappears or turns into loss!!
- e.g.'s: register windows, delayed branches

forward compatibility

- reserve trap hooks to emulate future ISA extensions