Computer Architecture's Changing Definition

1950s Computer Architecture

- Computer Arithmetic

1960s

- Operating system support, especially memory management

1970s to mid 1980s Computer Architecture

- Instruction Set Design, especially ISA appropriate for compilers
- Vector processing and shared memory multiprocessors

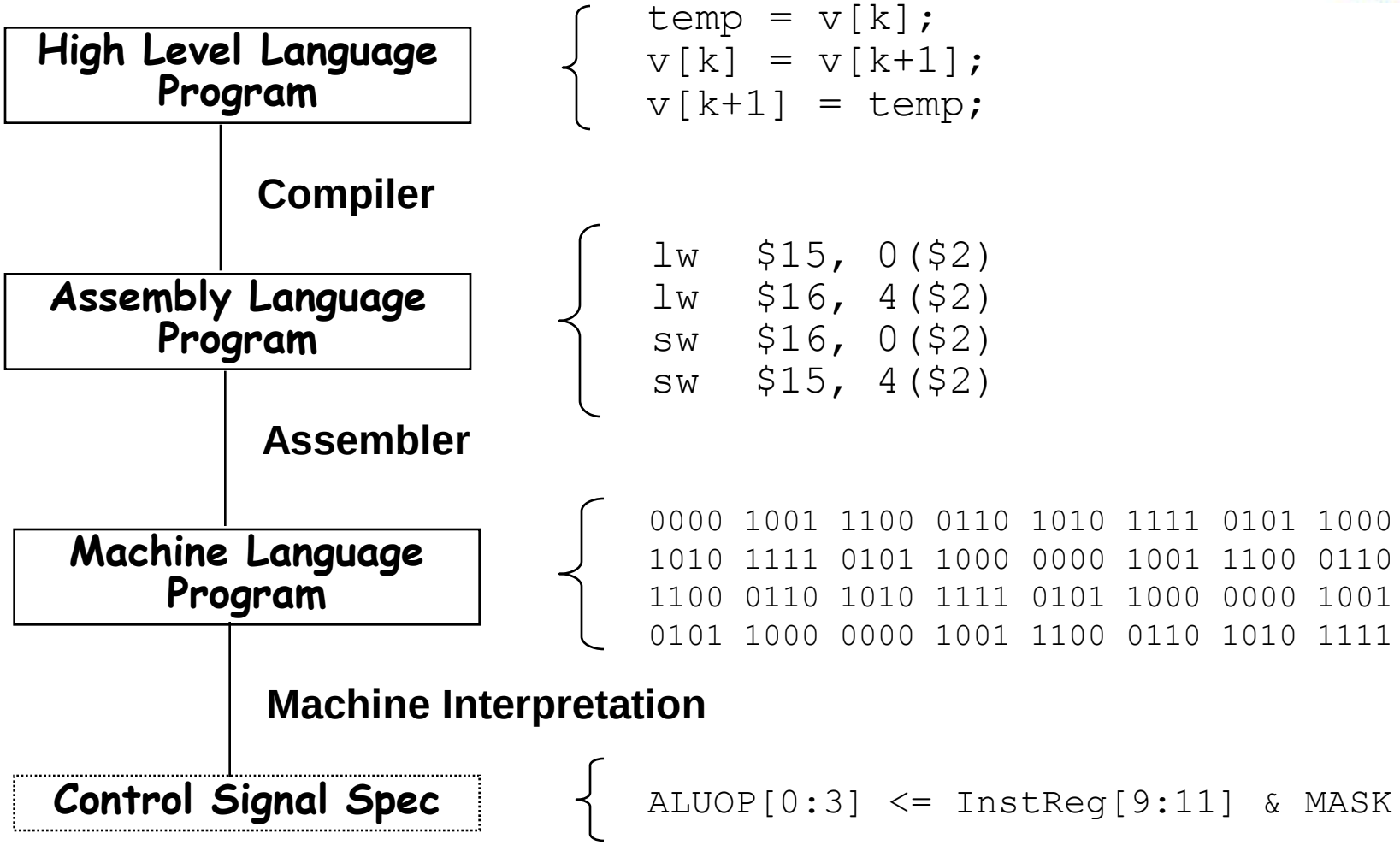
1990s Computer Architecture

- Design of CPU, memory system, I/O system, Multi-processors, Networks
- Design for VLSI

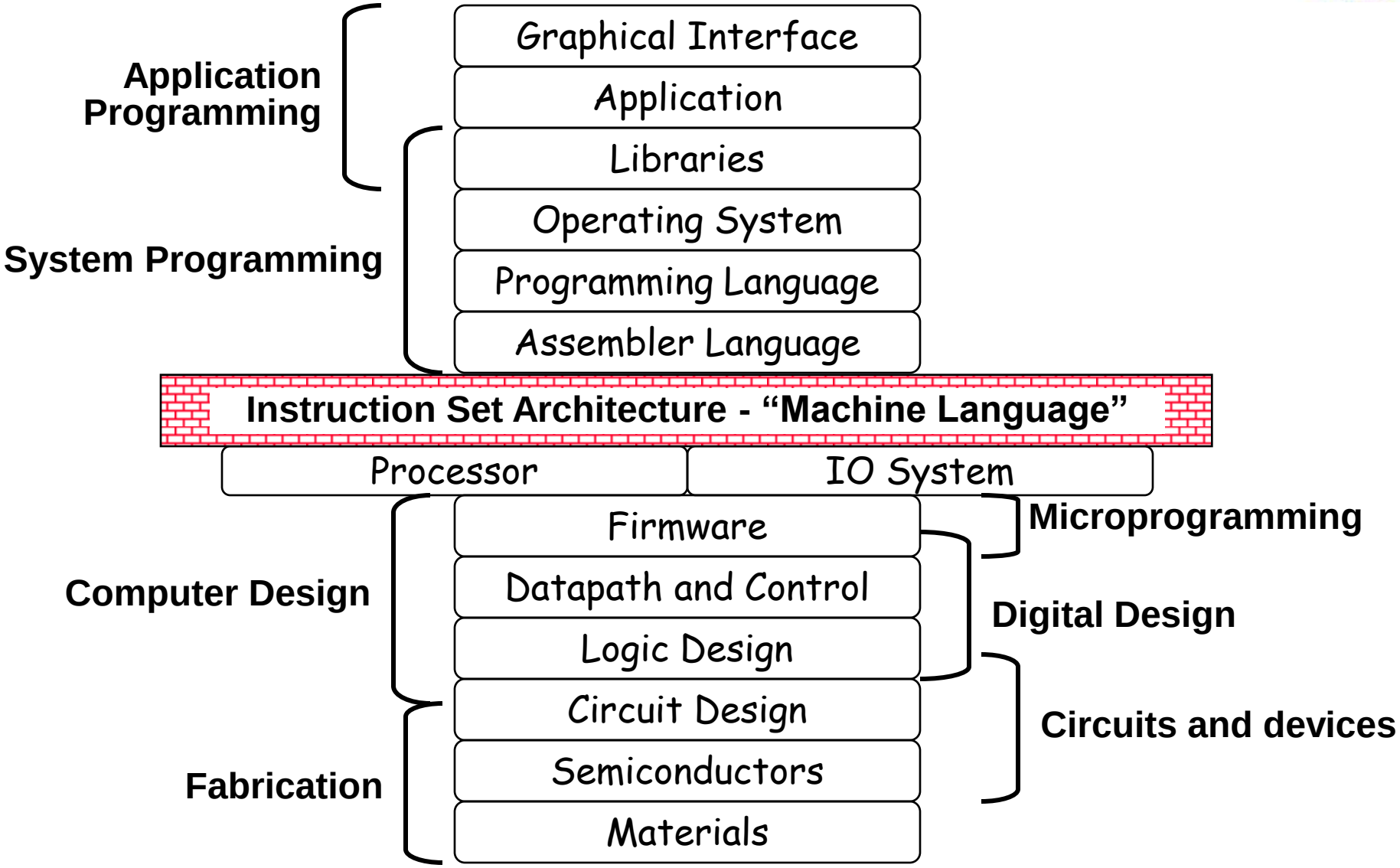
2000s Computer Architecture:

- Special purpose architectures, Functionally reconfigurable, Special considerations for low power/mobile processing, highly parallel structures

Levels of Representation



Levels of Abstraction



This Course Focuses on General Purpose Processors

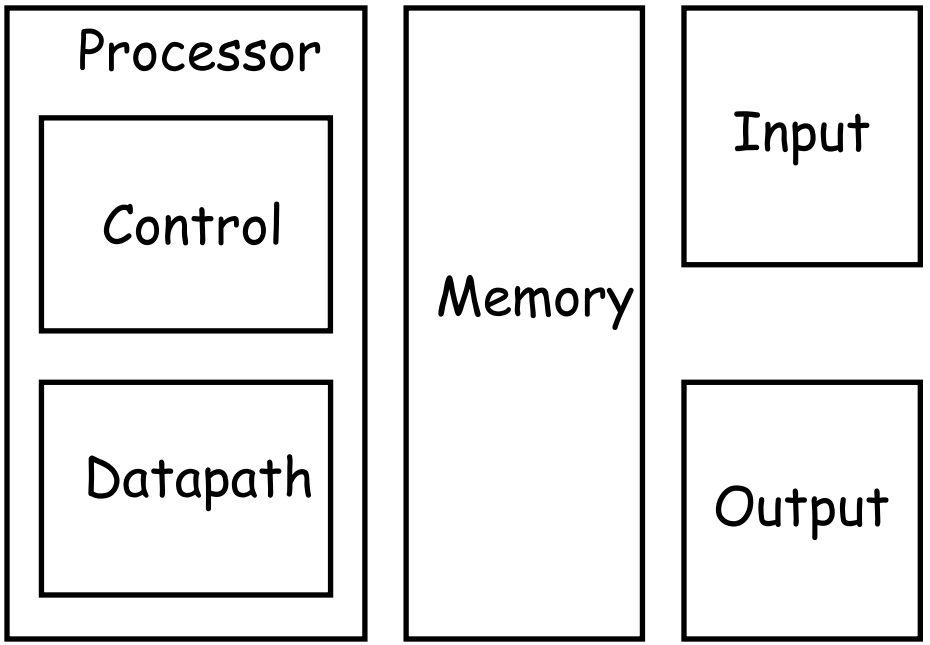
A general-purpose computer system

- Uses a programmable processor
- Can run “any” application
- Potentially optimized for some class of applications
- Common names: CPU, DSP, NPU, microcontroller, microprocessor

Unified main memory

- For both programs & data
- Von Neumann computer

Busses & controllers to connect processor, memory, IO devices



MIT Whirlwind, 1951

Computers are pervasive – servers, standalone PCs, network processors, embedded processors, …

Metrics of Efficiency - Examples

Desktop computing

- Examples: PCs, workstations
- Metrics: performance (latency), cost, time to market

Server computing

- Examples: web servers, transaction servers, file servers
- Metrics: performance (throughput), reliability, scalability

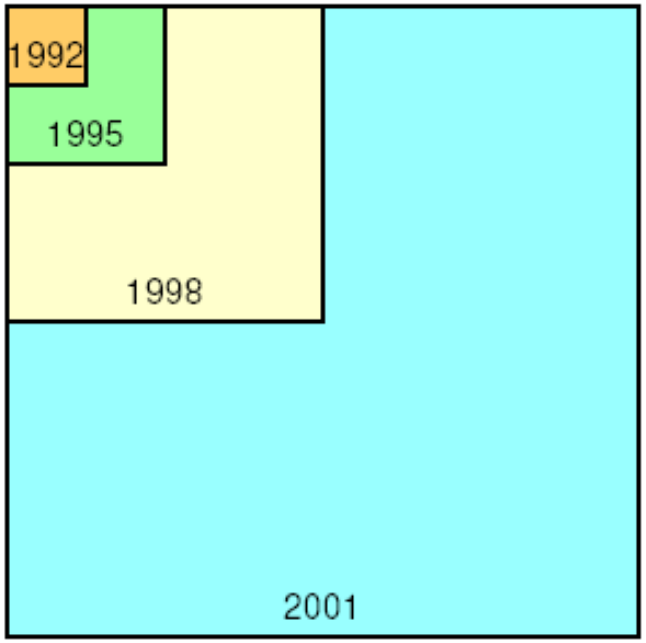
Embedded computing

- Examples: microwave, printer, cell phone, video console
- Metrics: performance (real-time), cost, power consumption, complexity

Moore's Law

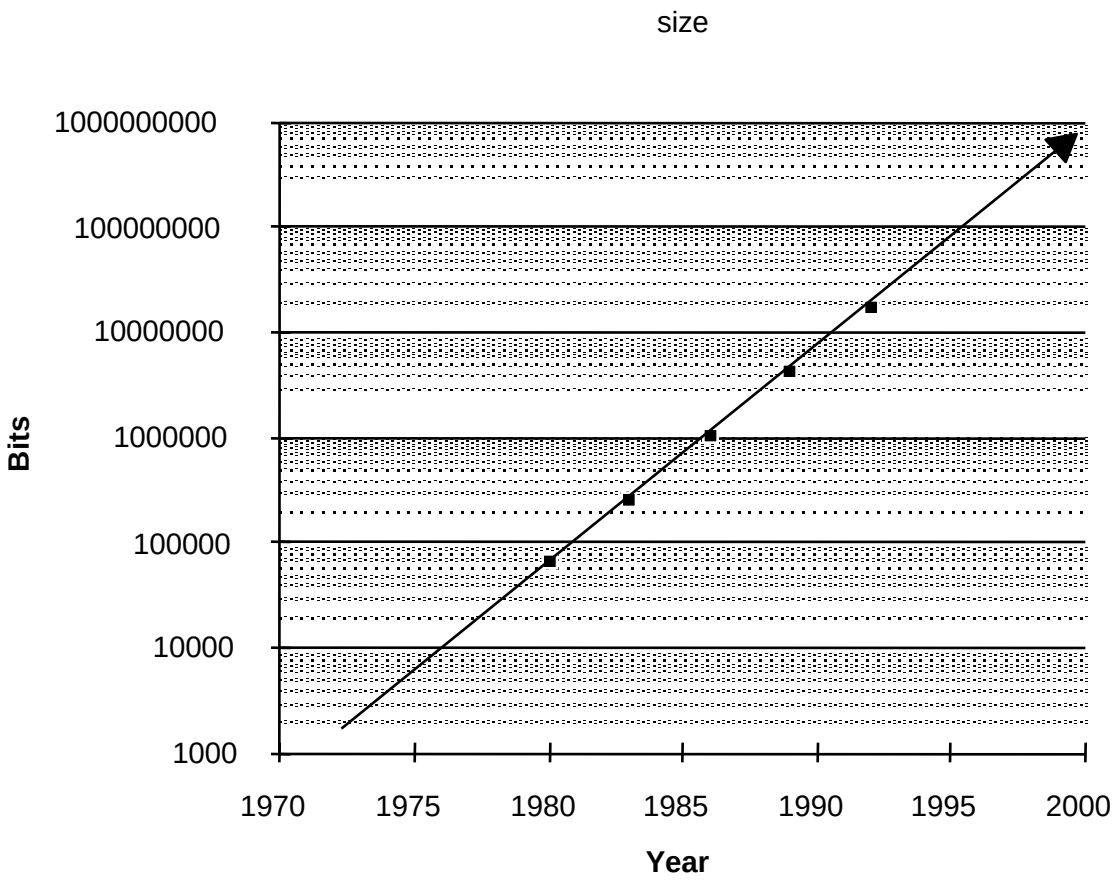
Moore's "Law" - The observation made in 1965 by Gordon Moore, co-founder of Intel, that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented. Moore predicted that this trend would continue for the foreseeable future.

In subsequent years, the pace slowed down a bit, but ***data density has doubled approximately every 18 months***, and this is the current definition of Moore's Law, which Moore himself has blessed. Most experts, including Moore himself, expect Moore's Law to hold for at least another two decades.



64x more devices since 1992
4x faster devices

DRAM Evolution



Year	Capacity	Access
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1996	64 Mb	120 ns
1999	256 Mb	100 ns
2002	1Gb	80 ns

Performance Perspectives

Purchasing perspective

- Given a collection of machines, which has the
  - Best performance ?
  - Least cost ?
  - Best performance / cost ?

Design perspective

- Faced with design options, which has the
  - Best performance improvement ?
  - Least cost ?
  - Best performance / cost ?

Both require

- basis for comparison
- metric for evaluation

Definitions

Performance is typically in units-per-second

- bigger is better

If we are primarily concerned with response time

- performance = $\frac{1}{\text{execution\_time}}$

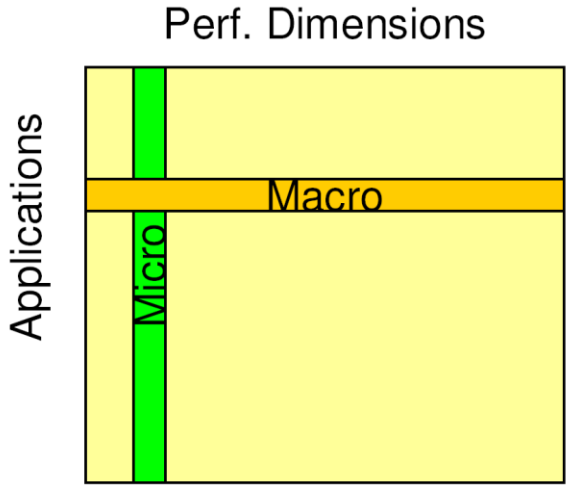
" X is n times faster than Y" means

$$\frac{ExecutionTime_y}{ExecutionTime_x} = \frac{Performance_x}{Performance_y} = n$$

Benchmarks

Micro-benchmarks

- Measure one performance dimension
  - Cache bandwidth
  - Memory bandwidth
  - Procedure call overhead
- Insight into the underlying performance factors
- Not a good predictor of application performance



Macro-benchmarks

- Application execution time
  - Measures overall performance, but on just one application
  - Need application suite

Why Do Benchmarks?

How we evaluate differences

- Different systems
- Changes to a single system

Provide a target

- Benchmarks should represent large class of important programs
- Improving benchmark performance should help many programs

For better or worse, benchmarks shape a field

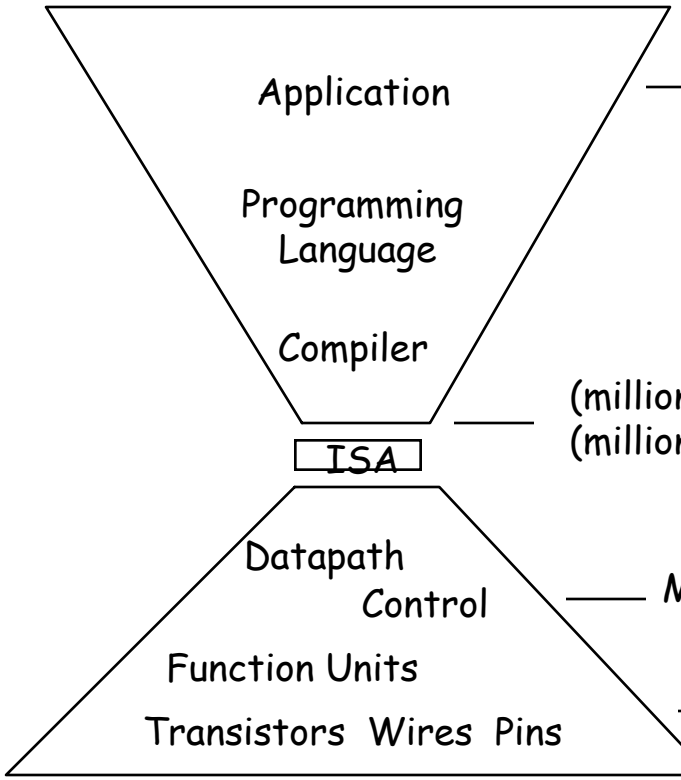
Good ones accelerate progress

- good target for development

Bad benchmarks hurt progress

- help real programs v. sell machines/papers?
- Inventions that help real programs don’t help benchmark

Metrics of Processor Performance



Seconds per program

Useful Operations per second

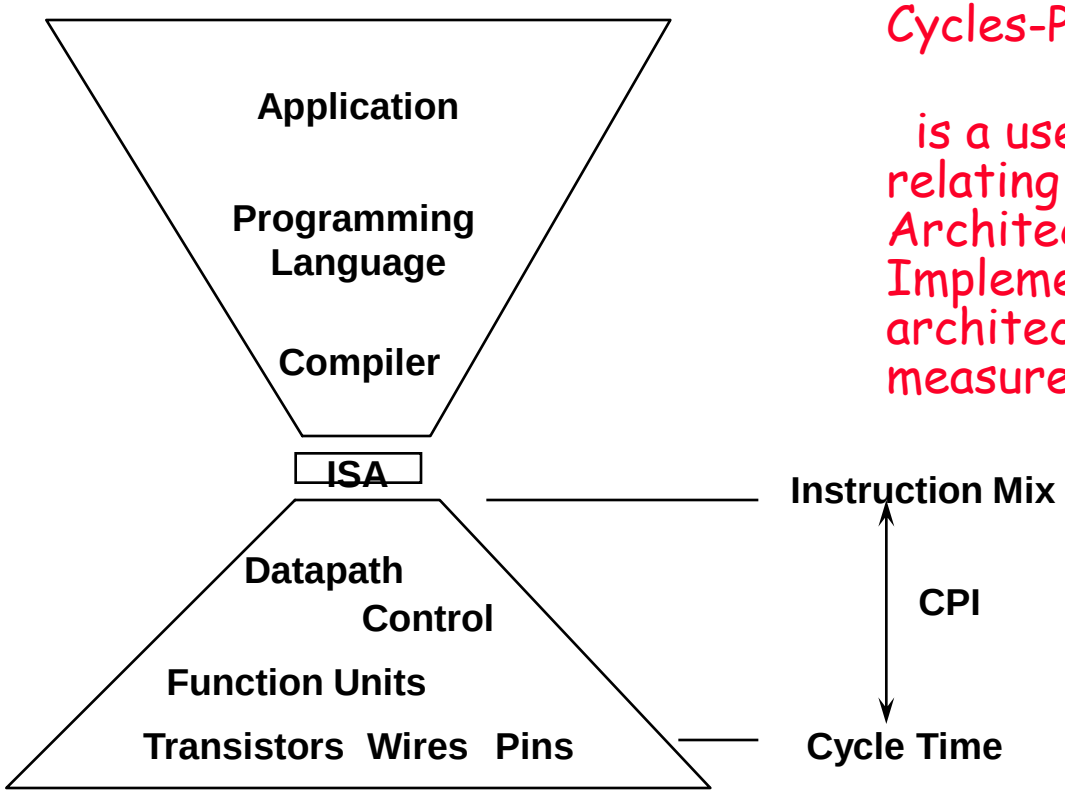
(millions) of Instructions per second – MIPS
(millions) of (F.P.) operations per second – MFLOP/s

Megabytes per second

Cycles per second (clock rate)

Instructions perform read/write/load/store operations in a series of clock cycles.

Organizational Trade-offs

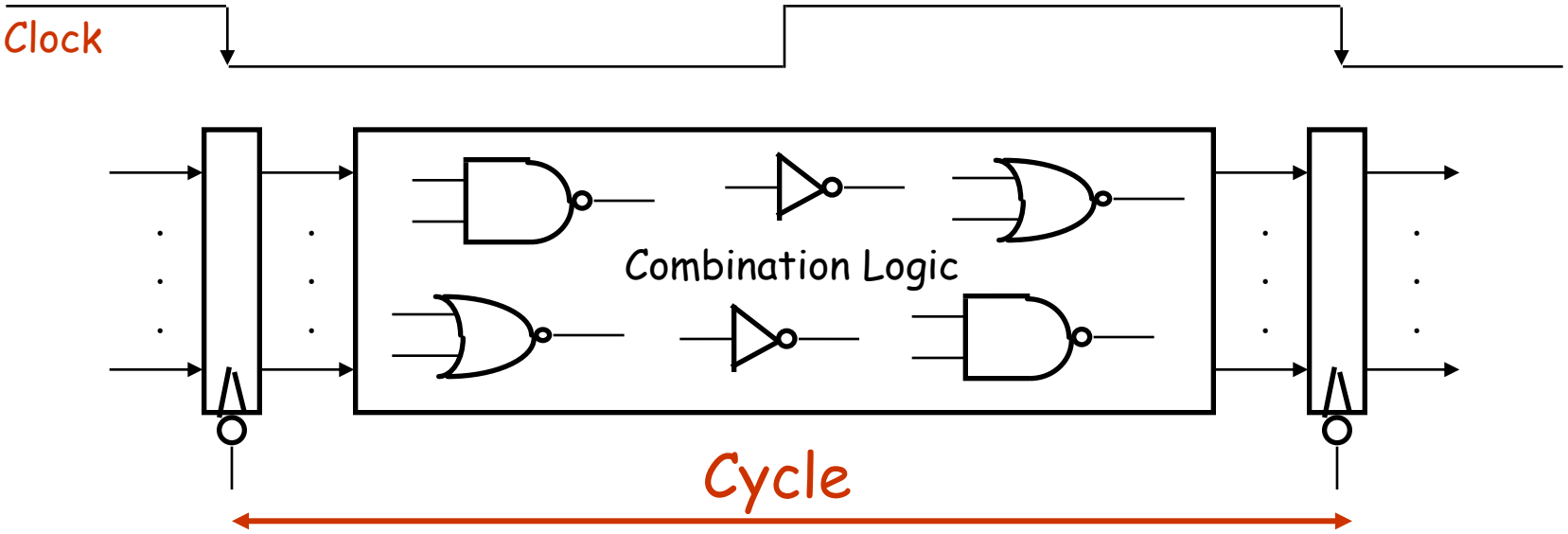


Cycles-Per-Instruction (CPI)

is a useful design measure relating the Instruction Set Architecture with the Implementation of that architecture, and the program measured

e.g. Can we handle more than one instruction in one clock cycle?

Processor Cycles

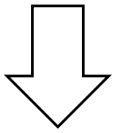


Most computers have fixed, repeating clock cycles

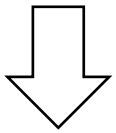
CPU Performance

$$CPUtime = \frac{Seconds}{Program} = \frac{Cycles}{Program} \cdot \frac{Seconds}{Cycle}$$

$$CPUtime = \frac{Instructions}{Program} \cdot \frac{Cycles}{Instruction} \cdot \frac{Seconds}{Cycle}$$



CPI



Clock rate

Cycles Per Instruction (Throughput)

"Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

Ij is the count of instruction-j in the total instruction set

"Instruction Frequency"

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

Example

Typical Mix

Op	Freq	Cycles	CPI
ALU	50%	1	0.5
Load	20%	5	1.0
Store	10%	3	0.3
Branch	20%	2	0.4
			2.2

Remember:

CPI is F x #cycles

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

- Load → 20% x 2 cycles = .4 0.5 + **0.4** + 0.3 + 0.4
- Total CPI 2.2 → 1.6
- Relative performance is 2.2 / 1.6 = 1.38 faster

How does this compare with reducing the branch instruction to 1 cycle?

- Branch → 20% x 1 cycle = .2
- Total CPI 2.2 → 2.0
- Relative performance is 2.2 / 2.0 = 1.1

Evaluating Instruction Sets and Implementation

Design-time metrics:

- Can it be implemented, in how long, at what cost?
- Can it be programmed? Ease of compilation?

Static Metrics:

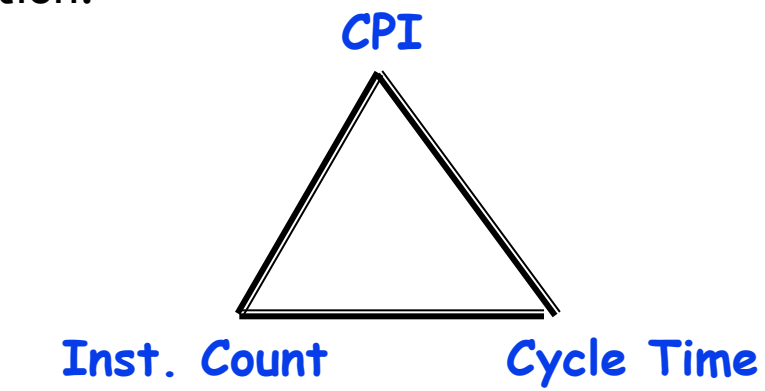
- How many bytes does the program occupy in memory?

Dynamic Metrics:

- How many instructions are executed?
- How many bytes does the processor fetch to execute the program?
- How many clocks are required per instruction?

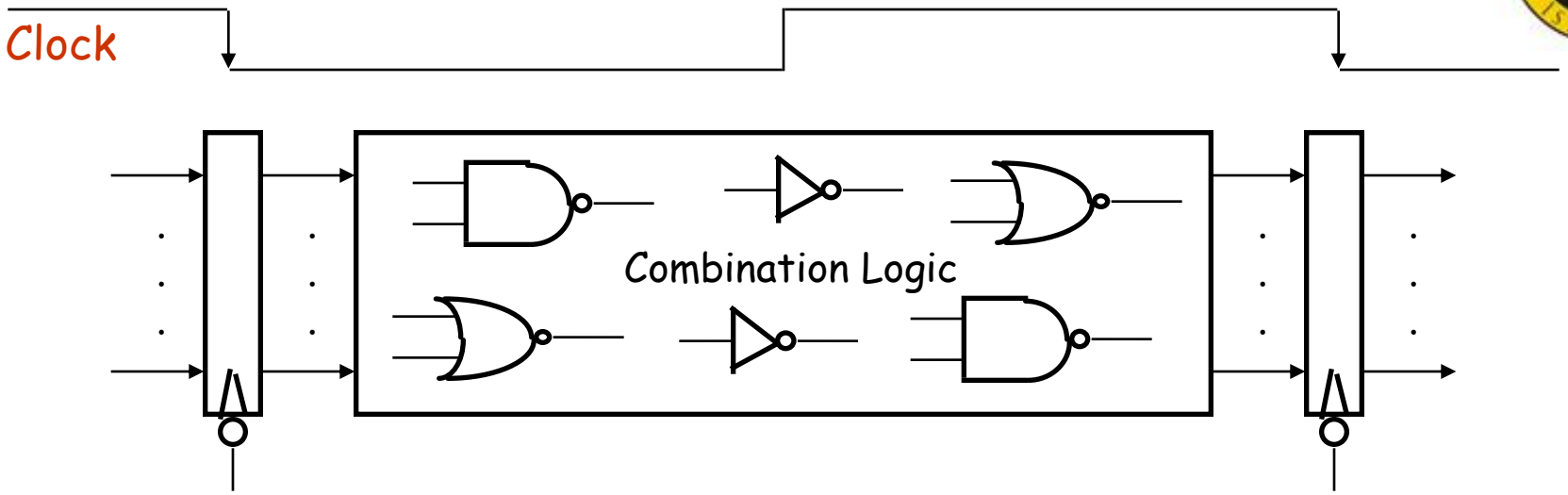
Best Metric:

Time to execute the program!



NOTE: Depends on instructions set, processor organization, and compilation techniques.

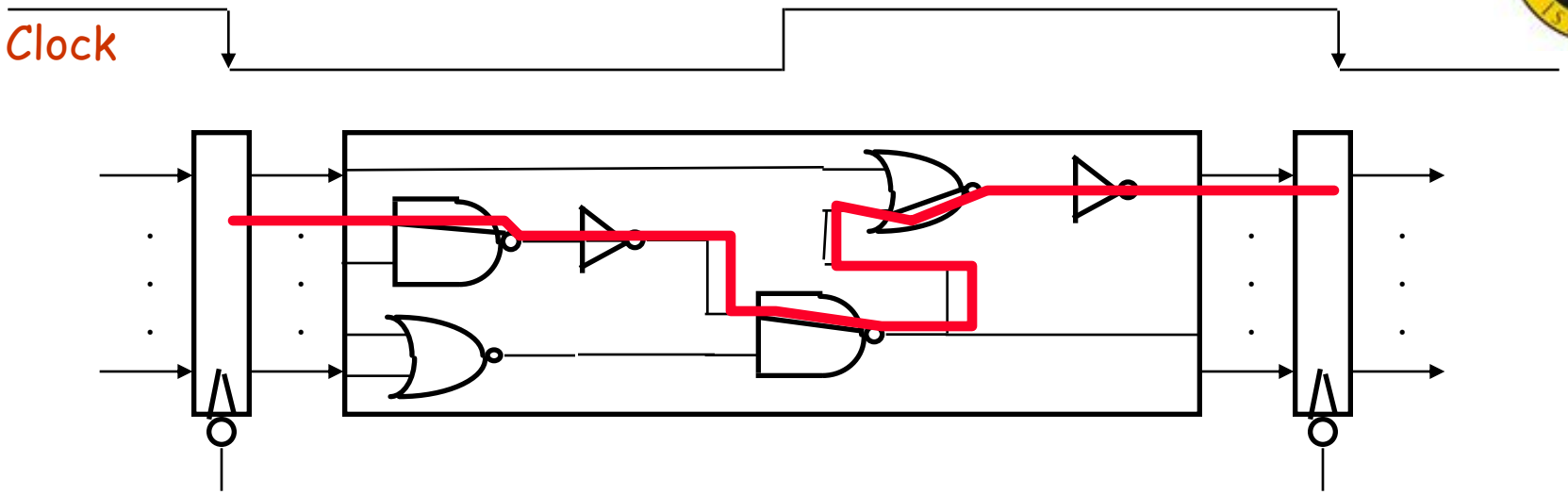
Clocking Methodology



All storage elements are clocked by the same clock edge

- Inputs are updated at each clock tick
- All outputs MUST be stable before the next clock tick

Critical Path & Cycle Time



Instructions follows various paths in these logic gates...

Critical path: the slowest path between any two storage devices

Cycle time is a function of the critical path