EECS 361
Computer Architecture
Lecture 2 - Performance

Prof. Gokhan Memik

memik@eecs.northwestern.edu

Course slides developed in part by Profs. Hardavellas, Hoe, Falsafi, Martin, Roth, Lipasti, Goldstein, Mowry

Four Lessons from the Previous Class

Importance and properties of ISA

- See the notes

Moore's Law

- Number of transistors doubles every two years
  - Largely due to increase in density

Memory Wall

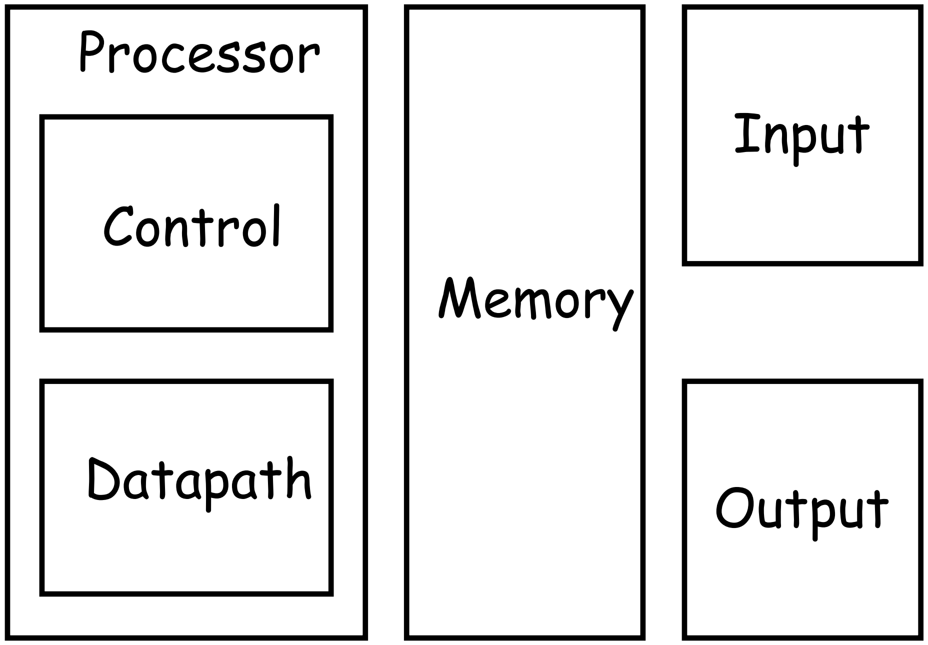
- Performance of different components do not scale with the same rates
  - Memory is becoming relatively slower every generation

Five components of a computer

- Datapath, Control, Memory, Input, Output

Five Major Components of a Computer

Busses & controllers to connect
processor, memory, IO devices



Today's Lecture

Performance Concepts

- Response Time
- Throughput

Performance Evaluation

- Benchmarks

Announcements

Processor Design Metrics

- Cycle Time
- Cycles per Instruction

Amdahl's Law

- Speedup what is important

Critical Path

Performance Concepts

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

Our goal: understand cost & performance implications of architectural choices

Two Notions of “Performance”

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

Which has higher performance?

Execution time (response time, latency, …)

- Time to do a task

Throughput (bandwidth, …)

- Tasks per unit of time

Response time and throughput are not parallel

Definitions

Performance is typically in units-per-second

- higher is better

If we are primarily concerned with response time

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

"X is n times faster than Y" means

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

Example

- Time of Concorde vs. Boeing 747?
  - Concorde is
    1350 mph / 610 mph = 2.2 times faster than Boeing 747
    = 6.5 hours / 3 hours

- Throughput of Concorde vs. Boeing 747 ?
  - Concorde is 178,200 pmph / 286,700 pmph = 0.62 "times faster"
  - Boeing is 286,700 pmph / 178,200 pmph = 1.60 "times faster"

- Boeing is 1.6 times ("60%") faster in terms of throughput
- Concorde is 2.2 times ("120%") faster in terms of flying time

We will focus primarily on execution time for a single job

Lots of instructions in a program ➔ Instruction throughput important!

Benchmarks

Evaluation Tools

Benchmarks, traces and mixes

- Macrobenchmarks and suites
- Microbenchmarks
- Traces

MOVE	39%
BR	20%
LOAD	20%
STORE	10%
ALU	11%

```
LD 5EA3
ST 31FF
....
LD 1EA2
....
```

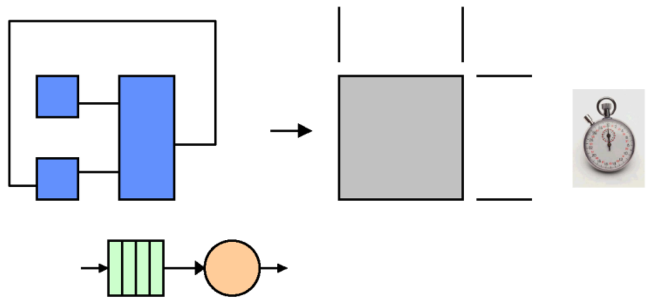
Workloads

Simulation at many levels

- ISA, microarchitecture, RTL, gate circuit
- Trade fidelity for simulation speed (Levels of abstraction)

Other metrics

- Area, clock frequency, power, cost, …



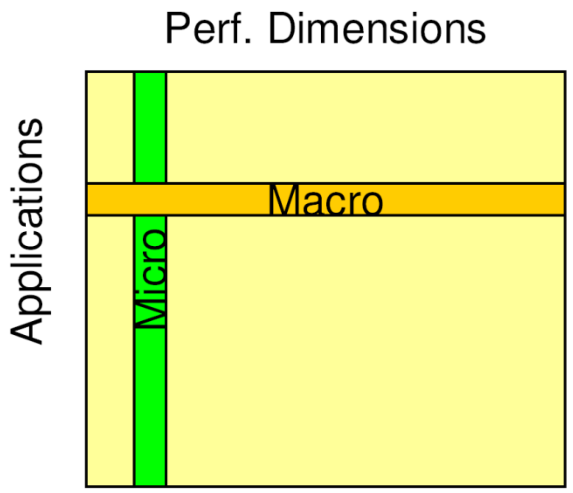
Analysis

- Queuing theory, back-of-the-envelope
- Rules of thumb, basic laws and principles

Benchmarks

Microbenchmarks

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



Macrobenchmarks

- 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 vs. sell machines/papers?
- Inventions that help real programs may not help benchmark

Popular Benchmark Suites

Desktop

- SPEC CPU - CPU intensive, integer & floating-point applications
  - SPEC CPU2006
- SPECviewperf, SPECapc - Graphics benchmarks
- SysMark, Winstone, Winbench

Embedded

- EEMBC - Collection of kernels from 6 application areas
- MediaBench, MiBench
- Bioperf, CommBench, MineBench, NetBench – Specific domains

Servers

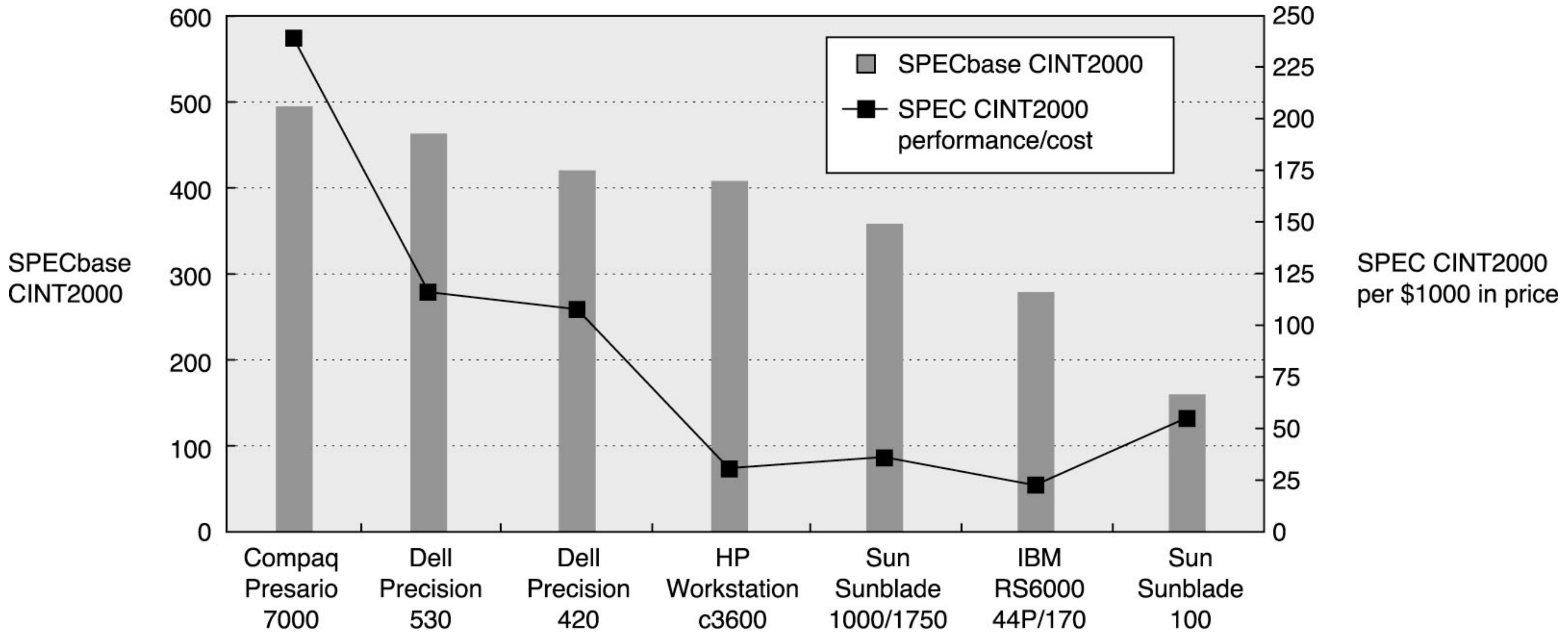
- SPECweb, SPECfs
- TPC-C - Transaction processing system; TPC-H, TPC-R - Decision support system; TPC-W - Transactional web; TPC-E – 3-tier e-commerce system

Parallel Computers

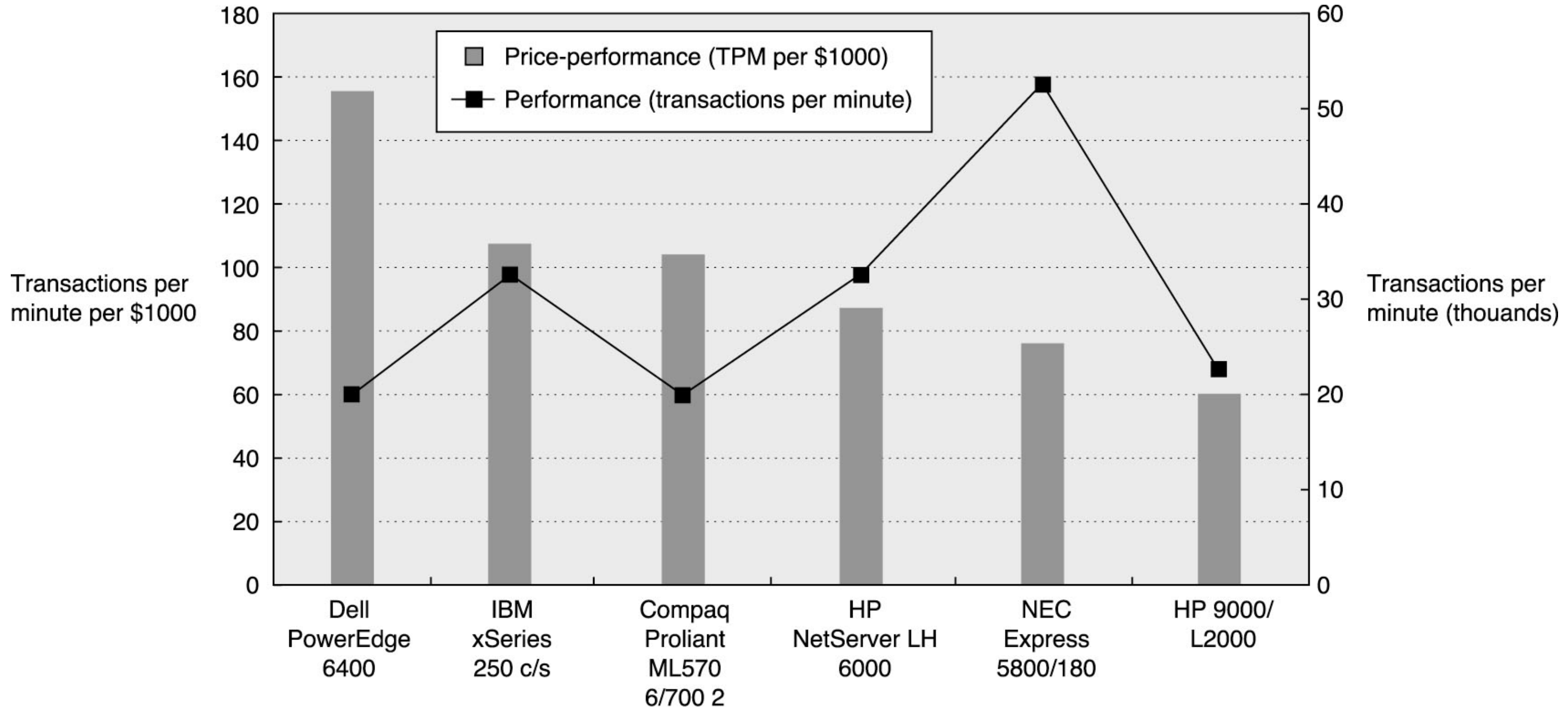
- SPLASH - Scientific applications & kernels

Most markets have specific benchmarks for design and marketing.

SPEC CINT2000



TPC-C



Price-performance (TPM per $1000)
Performance (transactions per minute)
Transactions per minute per $1000
Transactions per minute (thouands)
180
160
140
120
100
80
60
40
20
0
60
50
40
30
20
10
0
Dell PowerEdge 6400
IBM xSeries 250 c/s
Compaq Proliant ML570 6/700 2
HP NetServer LH 6000
NEC Express 5800/180
HP 9000/ L2000

Programs to Evaluate Processor Performance

(Toy) Benchmarks

- 10’s – 100’s lines of code
- e.g.,: sieve, puzzle, quicksort

Synthetic Benchmarks

- attempt to match average frequencies of real workloads
- e.g., Whetstone, dhrystone

Kernels

- Time critical excerpts

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	• less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

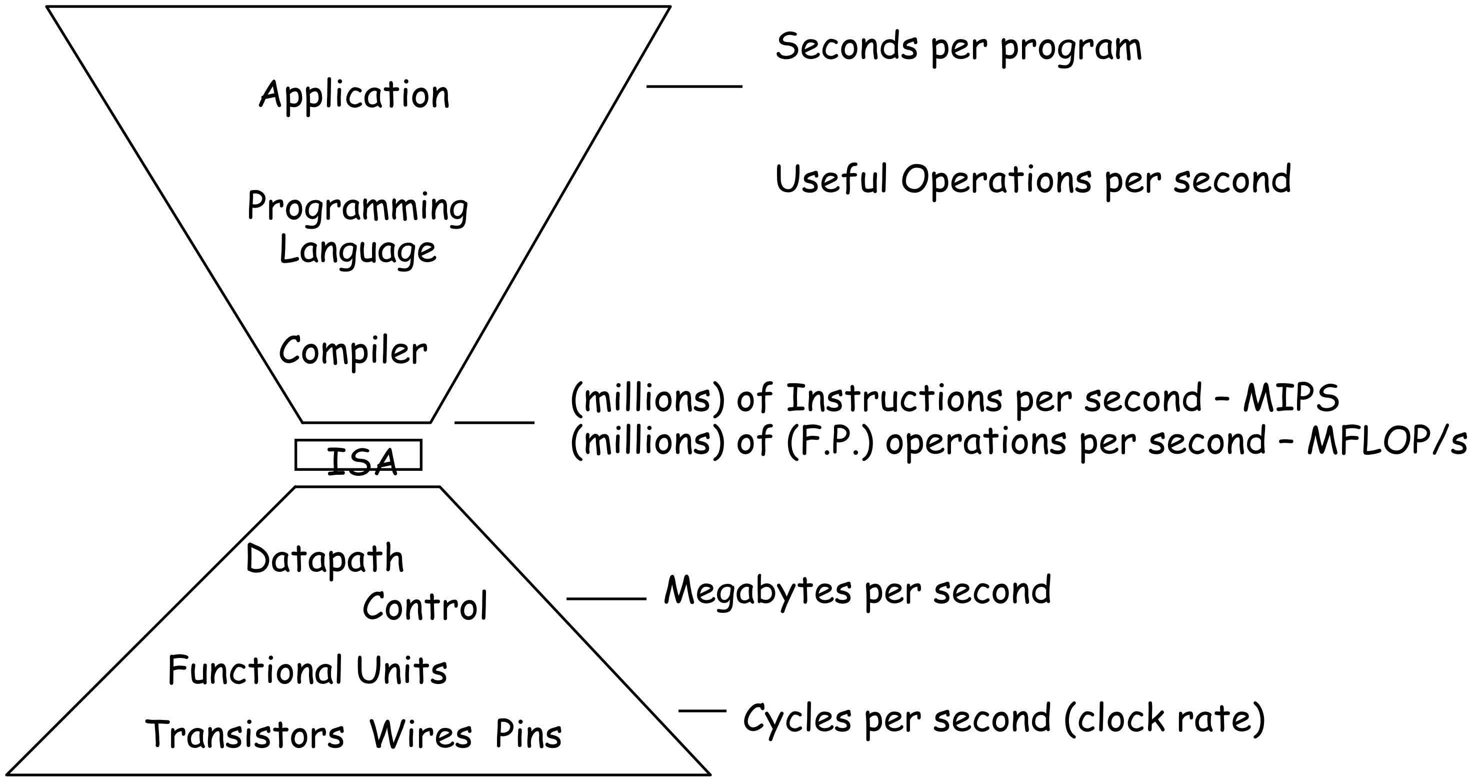
Announcements

Next lecture

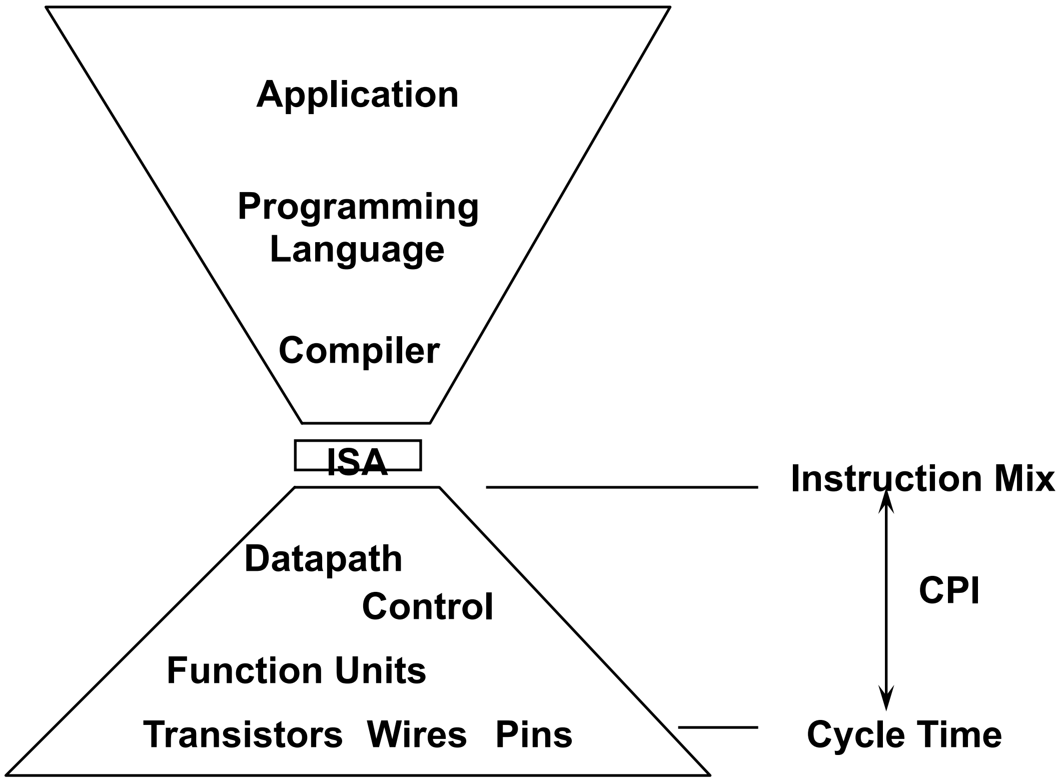
- Instruction Set Architecture

Processor Design Metrics

Metrics of Performance

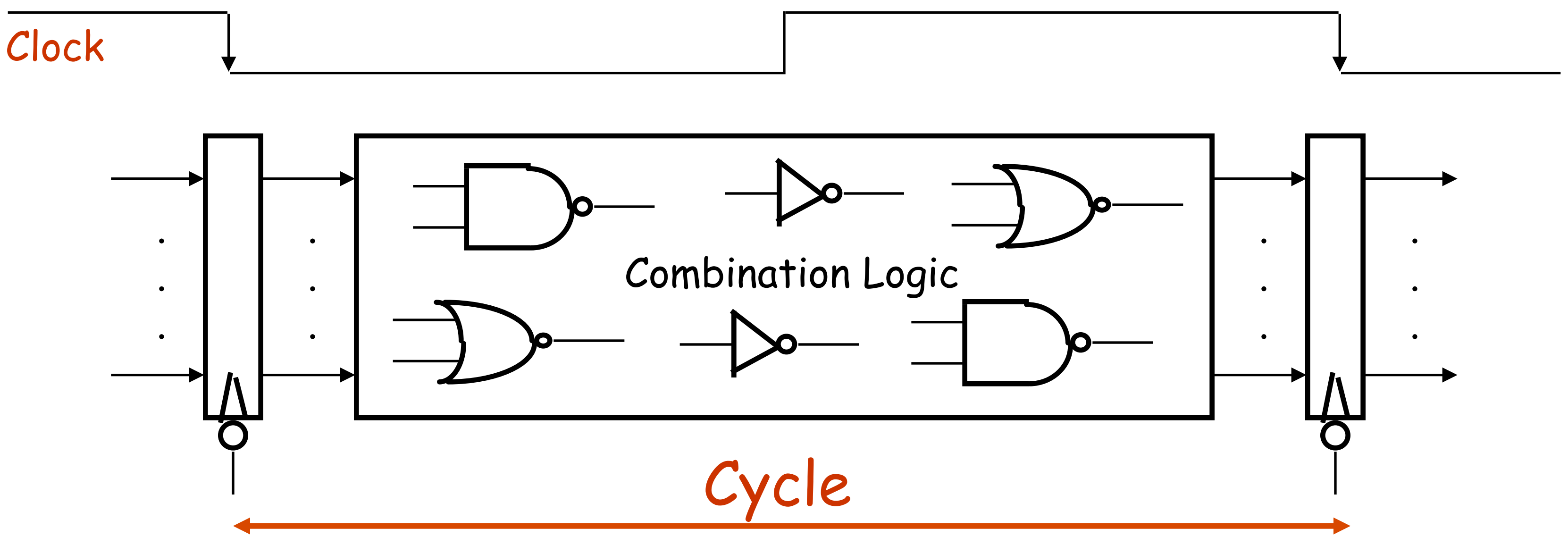

Application
Programming Language
Compiler
ISA
Datapath
Control
Functional Units
Transistors Wires Pins
Seconds per program
Useful Operations per second
(millions) of Instructions per second – MIPS
(millions) of (F.P.) operations per second – MFLOP/s
Megabytes per second
Cycles per second (clock rate)

Organizational Trade-offs



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

Processor Cycles



Most contemporary computers have fixed, repeating clock cycles

CPU Performance

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

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

	IC	CPI	Clock Cycle
Program	√		
Compiler	√	(√)	
Instruction Set	√	√	
Organization		√	√
Technology			√

IC = instruction count
CPI = cycles per instruction

Cycles Per Instruction (Throughput)

"Cycles per Instruction"

$$\text{CPI} = \frac{\text{Cycles}}{\text{Instruction Count}} = \frac{\text{CPU Time} \times \text{Clock Rate}}{\text{Instruction Count}}$$

$$\text{CPU time} = \text{Cycle Time} \times \text{Cycles} = \text{Cycle Time} \times \sum_{j=1}^{n} CPI_j \times I_j$$

"Instruction Frequency"

$$\text{CPI} = \frac{\text{Cycles}}{\text{Instruction Count}} = \frac{\sum_{j=1}^{n} CPI_j \times I_j}{\text{Instruction Count}} = \sum_{j=1}^{n} CPI_j \times F_j$$

$$\text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

Example

Typical Mix

Op	Freq	Cycles	CPI
ALU	50%	1	.5
Load	20%	5	1.0
Store	10%	3	.3
Branch	20%	2	.4
			2.2

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
- Total CPI 2.2 → 1.6
- Relative performance is 2.2 / 1.6 = 1.38

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

Principal Design Metrics: CPI and Cycle Time (if IC is fixed)

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

$$\text{Performance} = \frac{1}{\text{CPI} \times \text{Cycle Time}}$$

$$\text{Performance} = \frac{1}{\dfrac{\text{Cycles}}{\text{Instruction}} \times \dfrac{\text{Seconds}}{\text{Cycle}}} = \frac{\text{Instructions}}{\text{Seconds}}$$

Summary: 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:

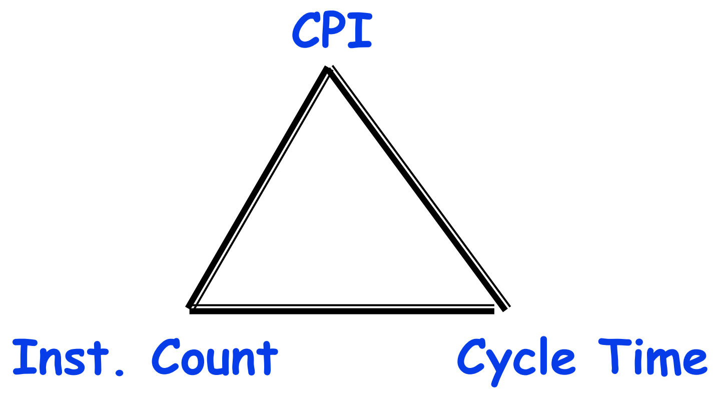
- How many bytes does the program occupy in memory?

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?
- How "lean" a clock is practical?

Best Metric:
Time to execute the program!

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



Marketing Metrics

```
MIPS    = Million instructions per second
        = Instruction Count / Time * 10^6
        = Clock Rate / CPI * 10^6
```

- machines with different instruction sets ?
- programs with different instruction mixes ?
- dynamic frequency of instructions
- uncorrelated with performance

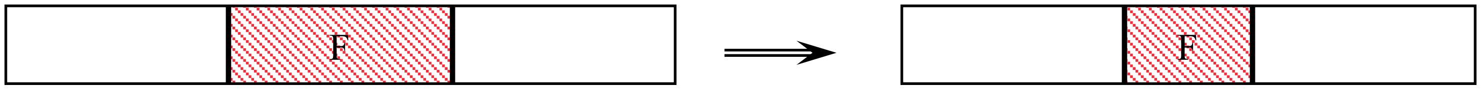
```
MFLOPS  = Million floating point operations per second
        = FP Operations / Time * 10^6
```

- machine dependent
- often not where time is spent

Amdahl's "Law": Make the Common Case Fast

Speedup due to enhancement X:

$$\text{Speedup(X)} = \frac{\text{ExecTime without X}}{\text{ExecTime with X}} = \frac{\text{Performance with X}}{\text{Performance without X}}$$



Suppose that enhancement X accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected then,

$$\text{ExecTime(with X)} = \left((1-F)+\frac{F}{S}\right) \times \text{ExecTime(without X)}$$

$$\text{Speedup(with X)} = \frac{\text{ExecTime(without X)}}{\left((1-F)+\frac{F}{S}\right) \times \text{ExecTime(without X)}}$$

$$Speedup = \frac{ExecTime_{old}}{ExecTime_{new}} = \frac{1}{\frac{Fraction_{enhanced}}{Speedup_{enhanced}} + (1 - Fraction_{enhanced})}$$

Performance improvement is limited by how much the improved feature is used

$\downarrow$

Invest resources where time is spent.

Summary

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

Time is the measure of computer performance!

Good products are created when we have:

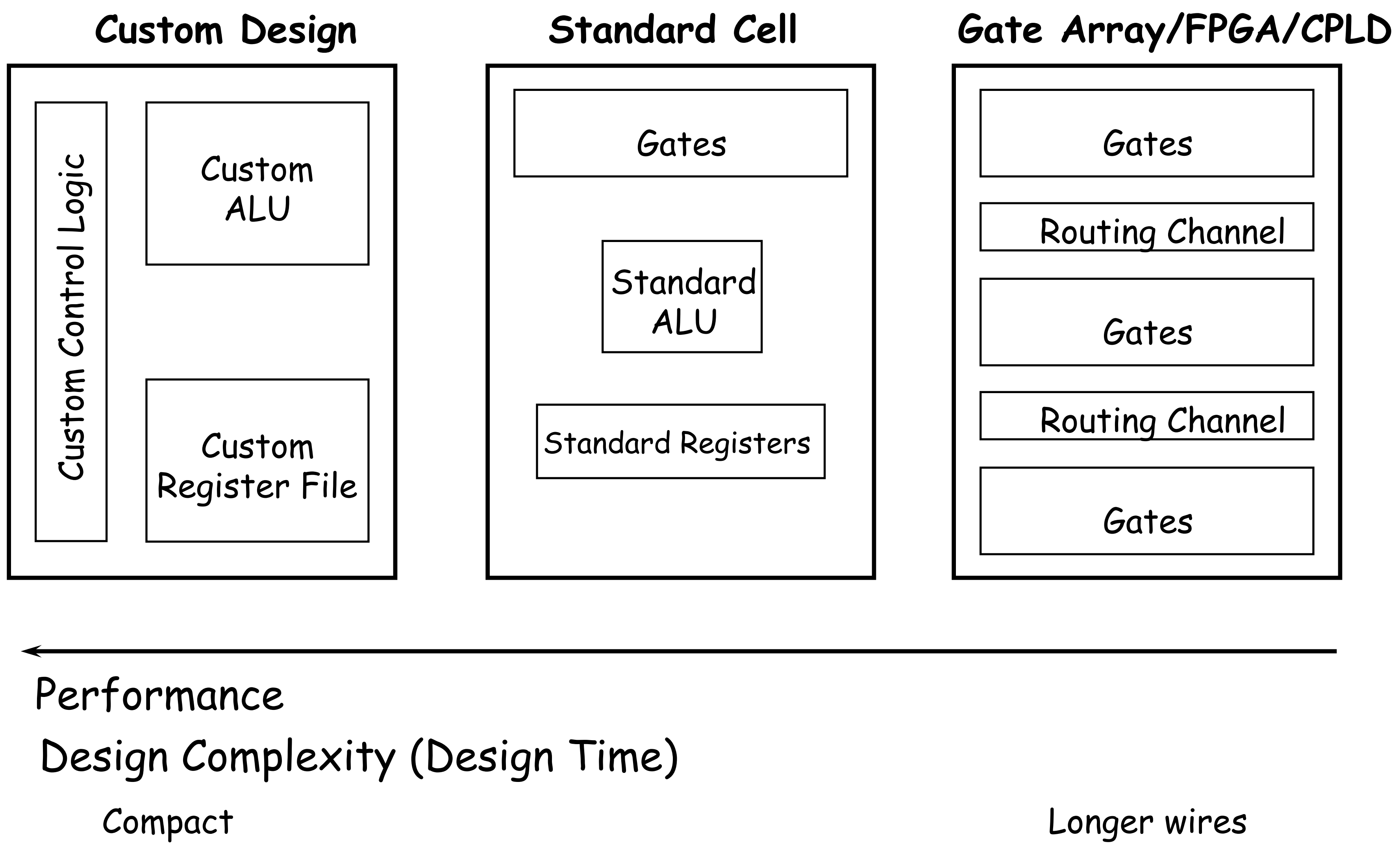
- Good benchmarks
- Good ways to summarize performance

If no good benchmarks and performance summary, then the choice is between improving product for real programs vs. improving product to get more sales → sales almost always wins

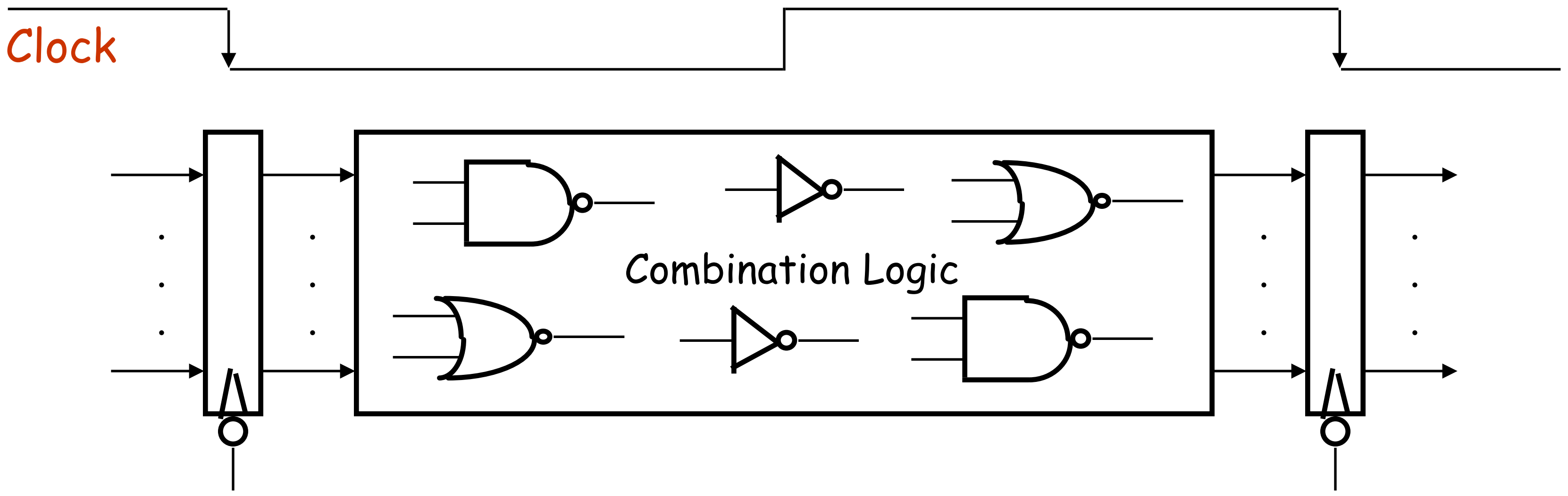
Remember Amdahl's Law: Speedup is limited by unimproved part of program

Critical Path

Range of Design Styles



Clocking Methodology

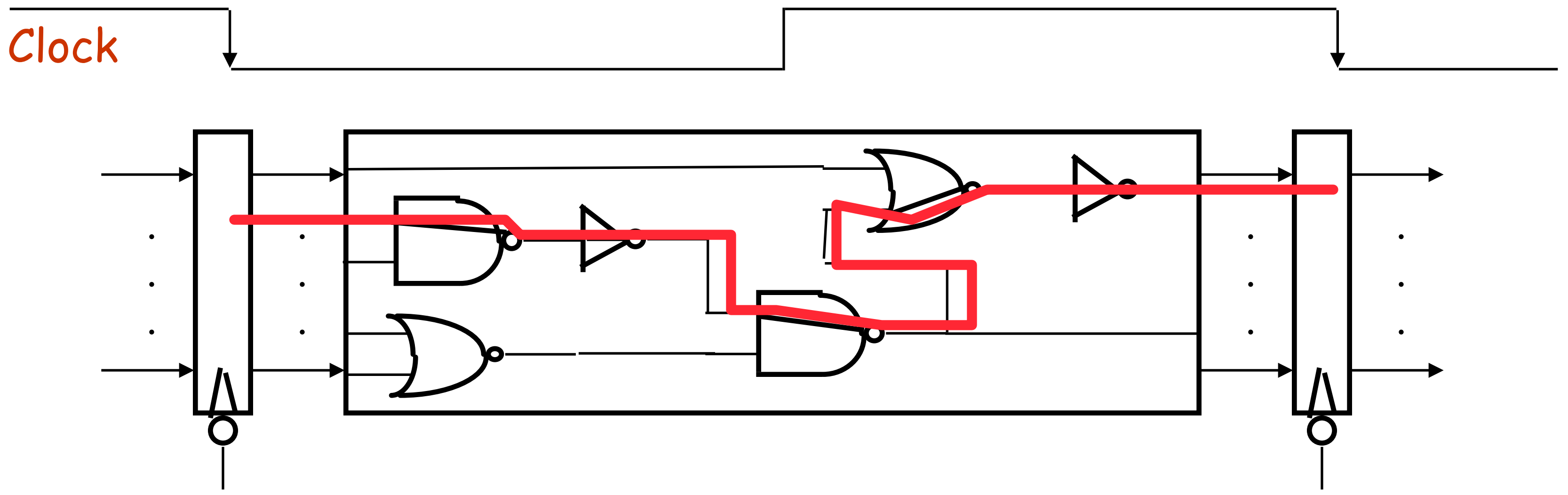


All storage elements are clocked by the same clock edge (but there may be clock skews)

The combination logic block's:

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

Critical Path & Cycle Time

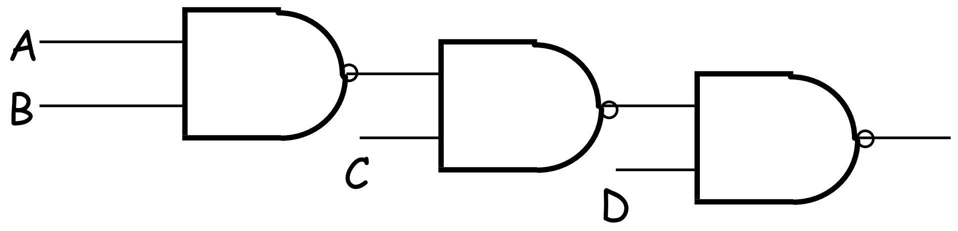


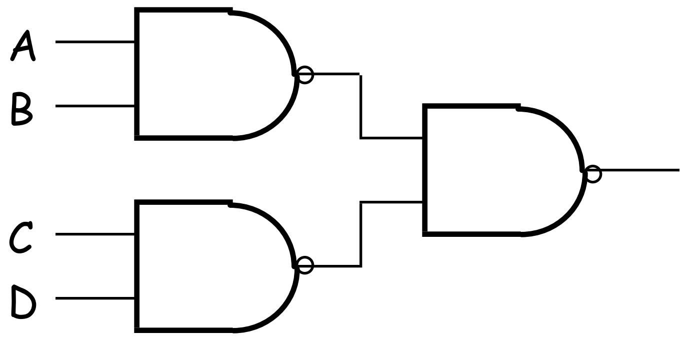
Critical path: the slowest path between any two storage devices

Cycle time is a function of the critical path

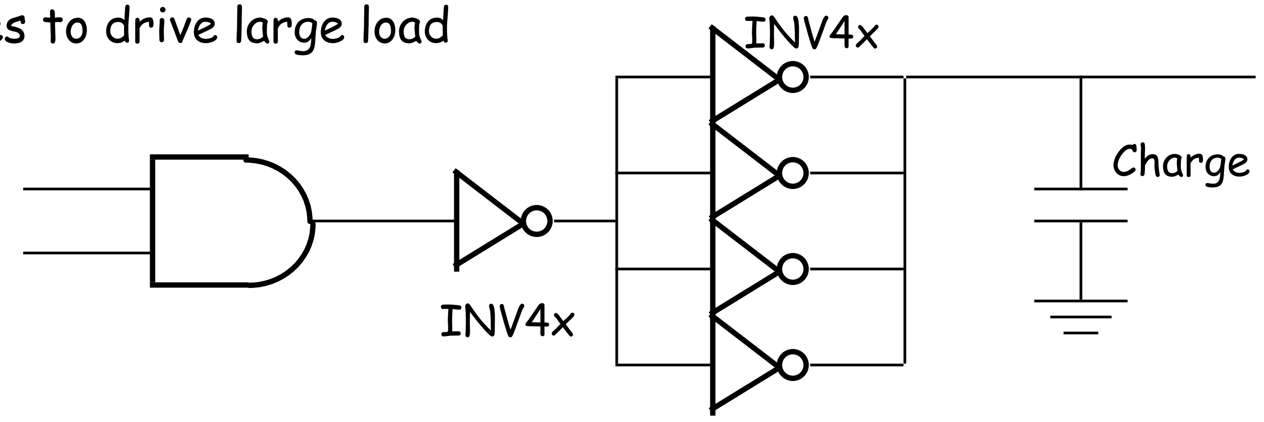
Tricks to Reduce Cycle Time

Reduce the number of gate levels





- Pay attention to loading
  - One gate driving many gates is a bad idea
  - Avoid using a small gate to drive a long wire
- Use multiple stages to drive large load
- Revise design



Summary

Performance Concepts

- Response Time
- Throughput

Performance Evaluation

- Benchmarks

Processor Design Metrics

- Cycle Time
- Cycles per Instruction

Amdahl's Law

- Speed up what is important
- **Make the common case fast, and the rare case correct**

Critical Path

Keywords

- Benchmarks, CPI – IC – Cycle Time, Amdahl's Law, Critical Path