ECE C61
Computer Architecture
Lecture 2 – performance

Prof. Alok N. Choudhary

choudhar@ece.northwestern.edu

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 often are in opposition

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

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

Example

- Time of Concorde vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 times faster

    = 6.5 hours / 3 hours

- Throughput of Concorde vs. Boeing 747 ?
  - Concord 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
- Concord 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 thruput important!

Benchmarks

Evaluation Tools

Benchmarks, traces and mixes

- Macrobenchmarks and suites
- Microbenchmarks
- Traces

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

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

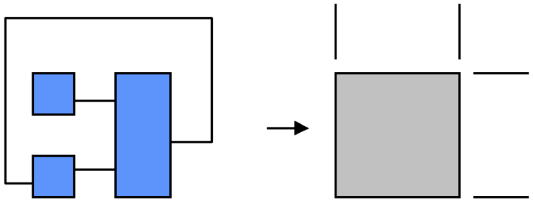
Workloads

Simulation at many levels

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

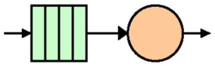
Other metrics

- Area, clock frequency, power, cost, …



Analysis

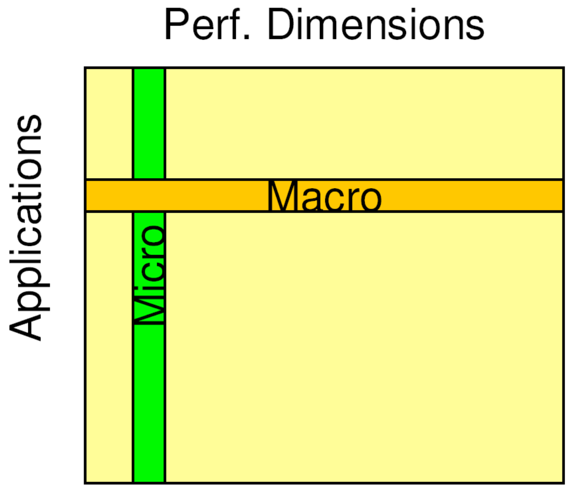
- Queuing theory, back-of-the-envelope
- Rules of thumb, basic laws and principles



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

Popular Benchmark Suites

Desktop

- SPEC CPU2000 - CPU intensive, integer & floating-point applications
- SPECviewperf, SPECapc - Graphics benchmarks
- SysMark, Winstone, Winbench

Embedded

- EEMBC - Collection of kernels from 6 application areas
- Dhrystone - Old synthetic benchmark

Servers

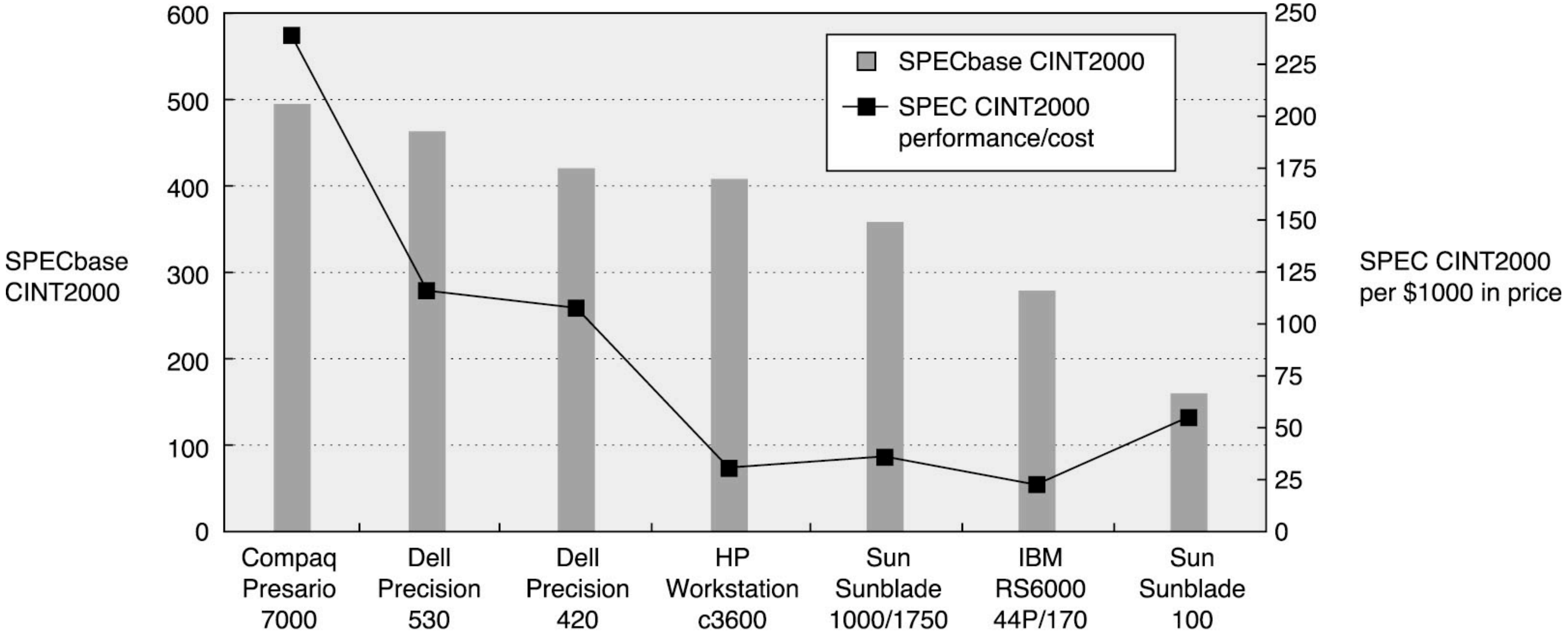
- SPECweb, SPECfs
- TPC-C - Transaction processing system
- TPC-H, TPC-R - Decision support system
- TPC-W - Transactional web benchmark

Parallel Computers

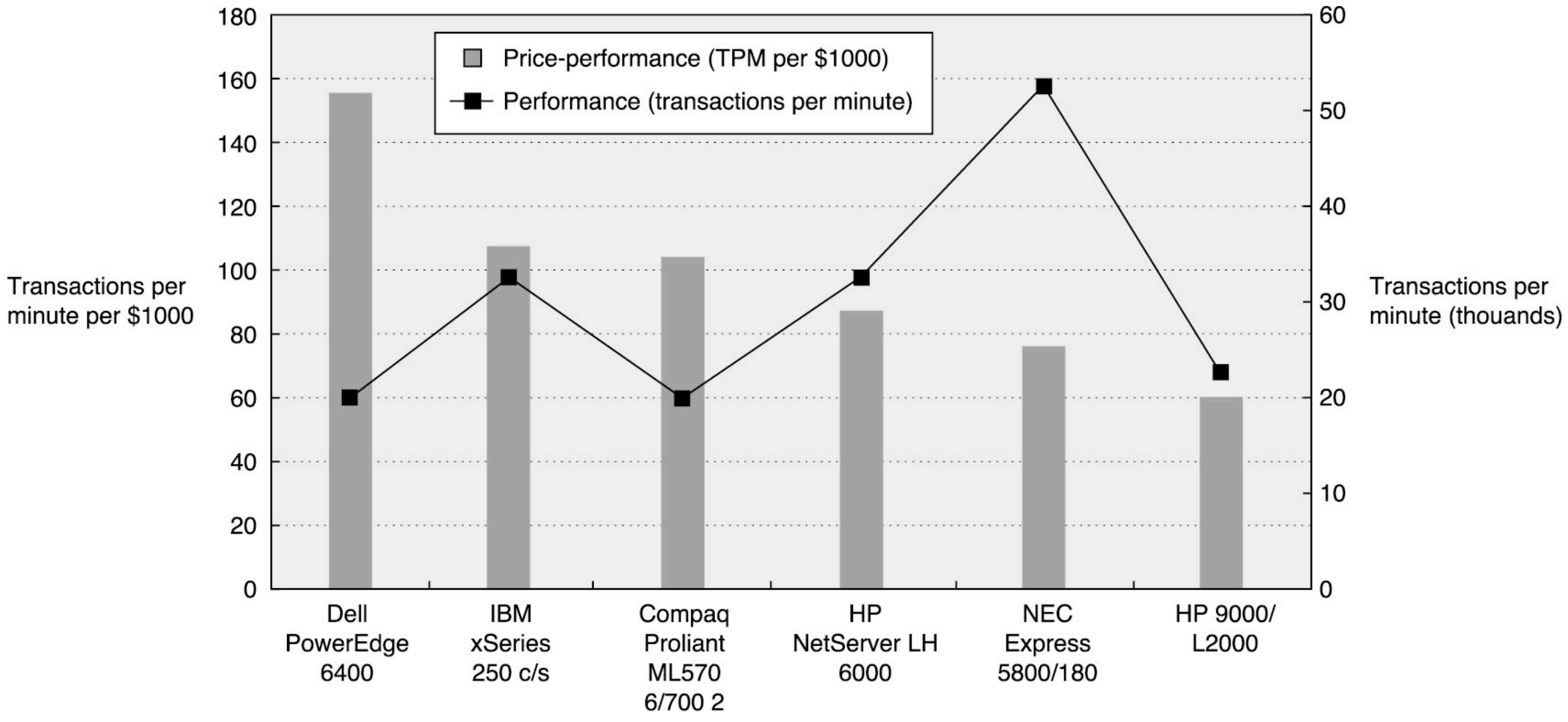
- SPLASH - Scientific applications & kernels

Most markets have specific benchmarks for design and marketing.

SPEC CINT2000



tpC


180
160
140
120
100
80
60
40
20
0
60
50
40
30
20
10
0
Transactions per minute per $1000
Transactions per minute (thouands)
▨ Price-performance (TPM per $1000)
■ Performance (transactions per minute)
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

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

Programs to Evaluate Processor Performance

(Toy) Benchmarks

- 10-100 line
- e.g.,: sieve, puzzle, quicksort

Synthetic Benchmarks

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

Kernels

- Time critical excerpts

Announcements

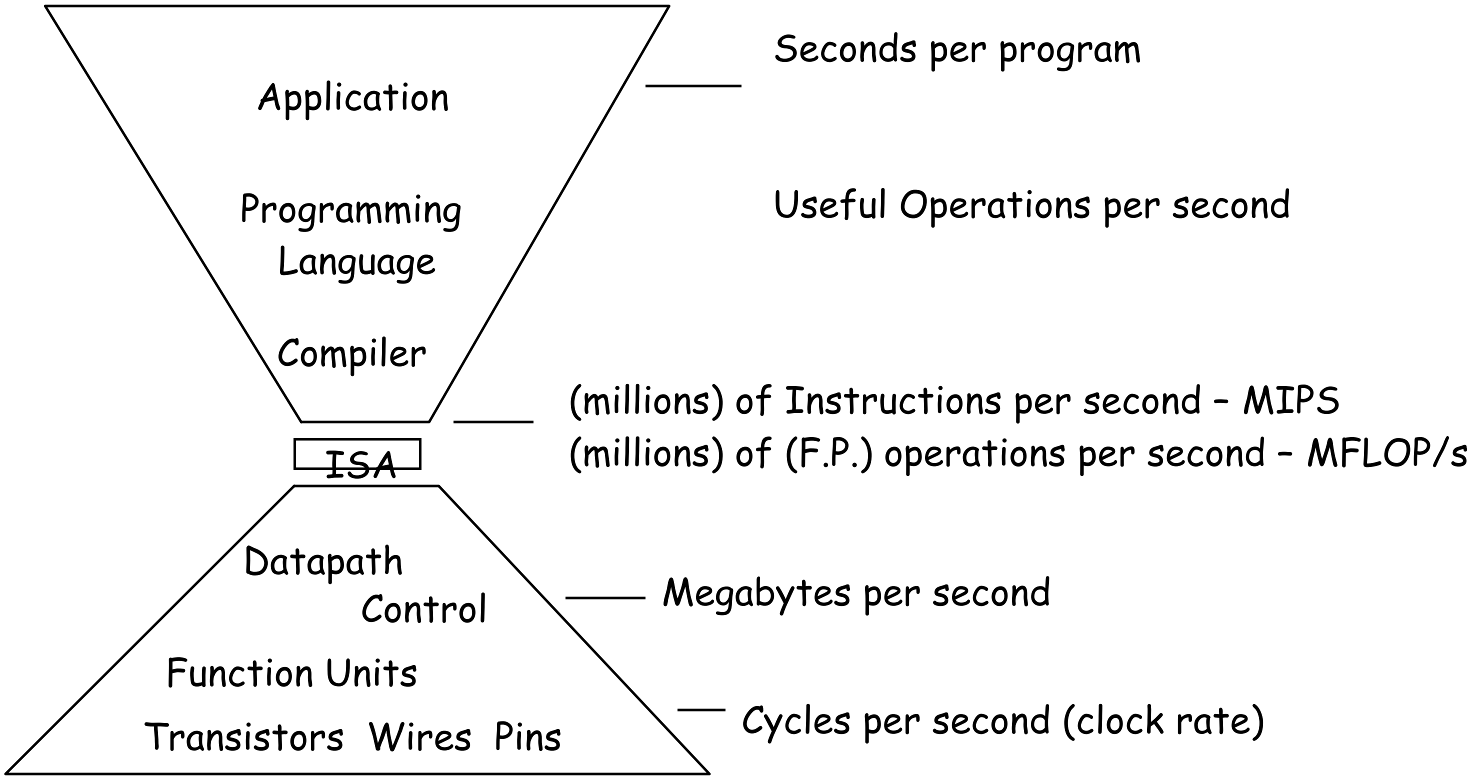
Website http://www.ece.northwestern.edu/~kcoloma/ece361

Next lecture

- Instruction Set Architecture

Processor Design Metrics

Metrics of Performance



Application

Programming Language

Compiler

ISA

Datapath

Control

Function Units

Transistors Wires Pins

Seconds per program

Useful Operations per second

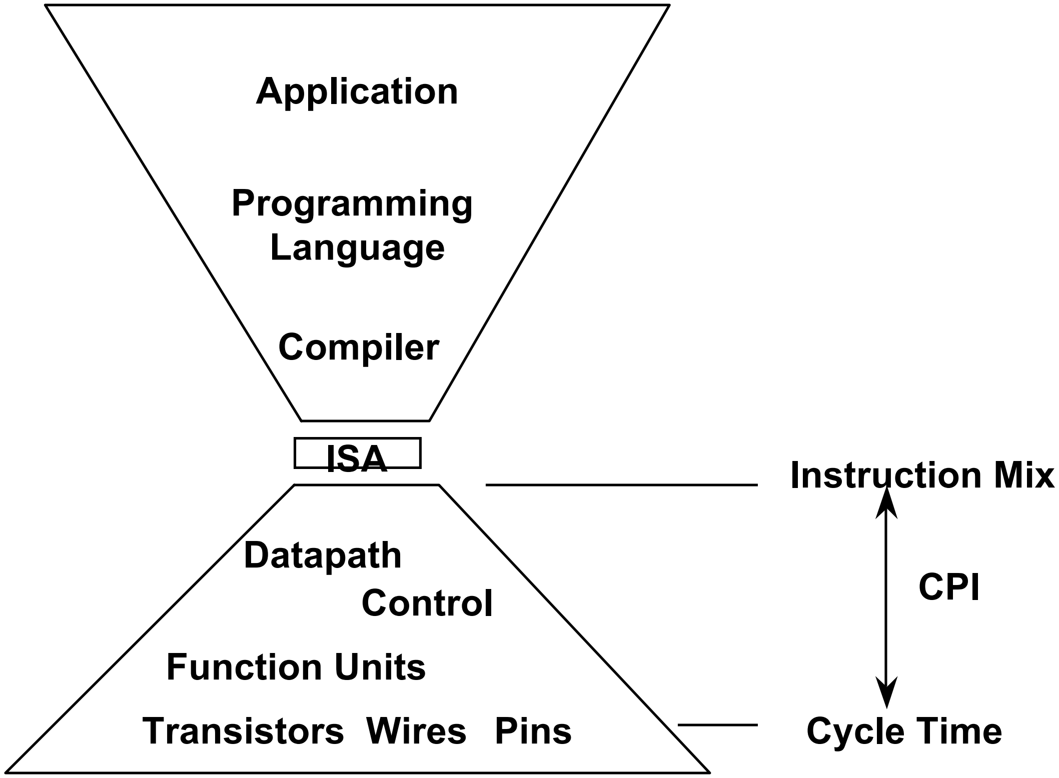
(millions) of Instructions per second – MIPS

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

Megabytes per second

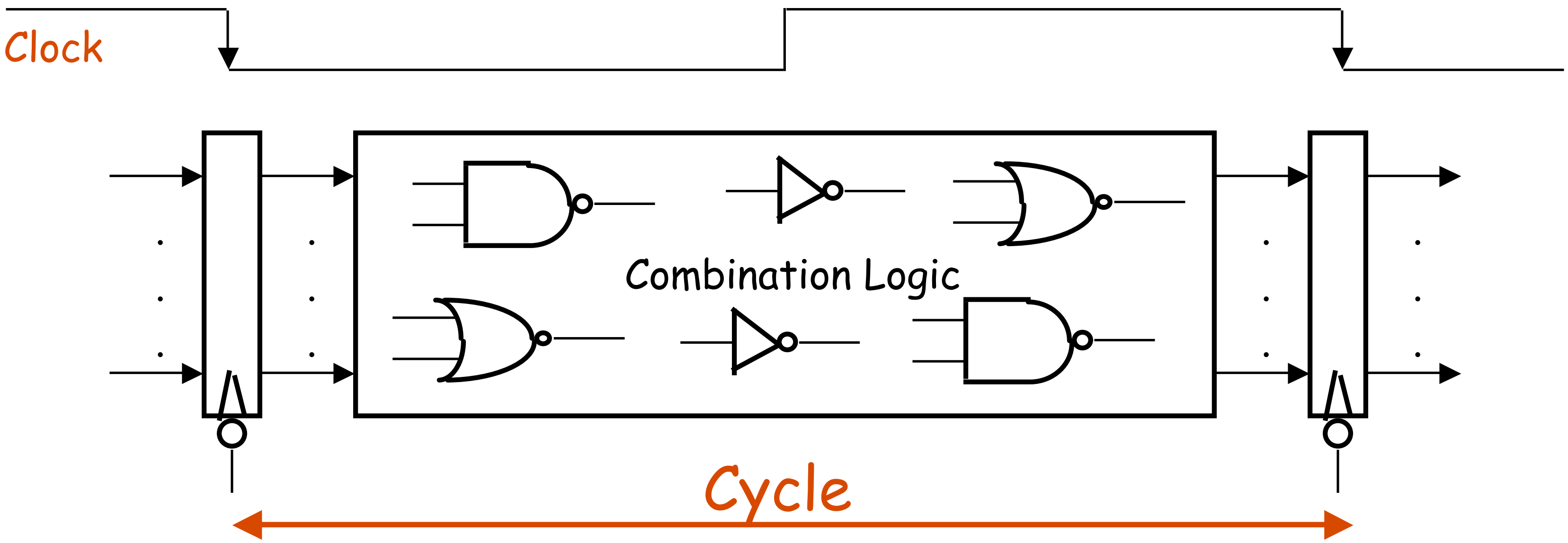
Cycles per second (clock rate)

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

$$CPUtime = \frac{Seconds}{\Pr ogram} = \frac{Cycles}{\Pr ogram} \cdot \frac{Seconds}{Cycle}$$

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

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

Cycles Per Instruction (Throughput)

"Cycles per Instruction"

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

= Cycles / Instruction Count

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

"Instruction Frequency"

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

Principal Design Metrics: CPI and Cycle Time

$$Performance = \frac{1}{ExecutionTime}$$

$$Performance = \frac{1}{CPI \times CycleTime}$$

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

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

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?

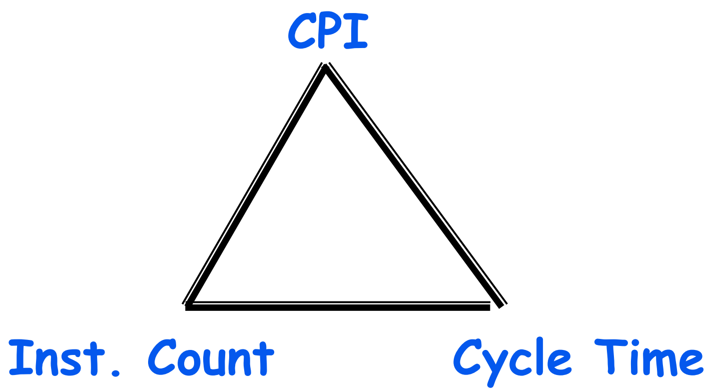
Static Metrics:

- 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!

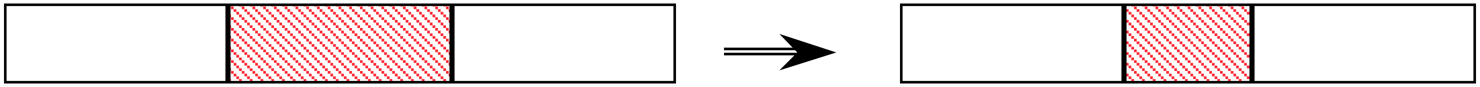


NOTE: Depends on instructions set, processor organization, and compilation techniques.

Amdahl's “Law”: Make the Common Case Fast

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



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

ExTime(with E) = ((1-F) + F/S) X ExTime(without E)

Performance improvement is limited by how much the improved feature is used → Invest resources where time is spent.

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

Marketing Metrics

MIPS = 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

MFLOP/s= FP Operations / Time * 10^6

- machine dependent
- often not 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 created when have:

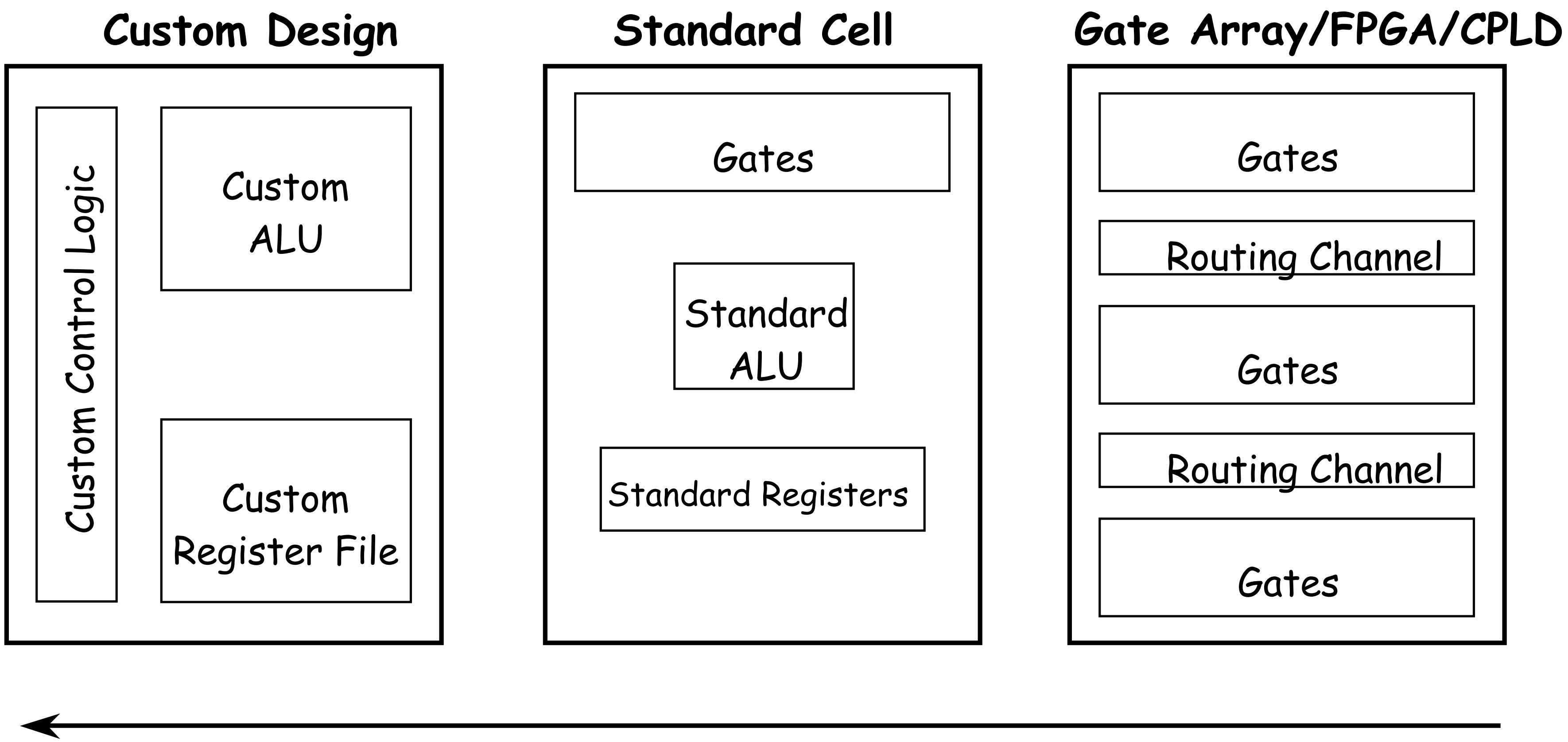
- Good benchmarks
- Good ways to summarize performance

If not good benchmarks and summary, then choice 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



Custom Design

Custom Control Logic

Custom ALU

Custom Register File

Standard Cell

Gates

Standard ALU

Standard Registers

Gate Array/FPGA/CPLD

Gates

Routing Channel

Gates

Routing Channel

Gates

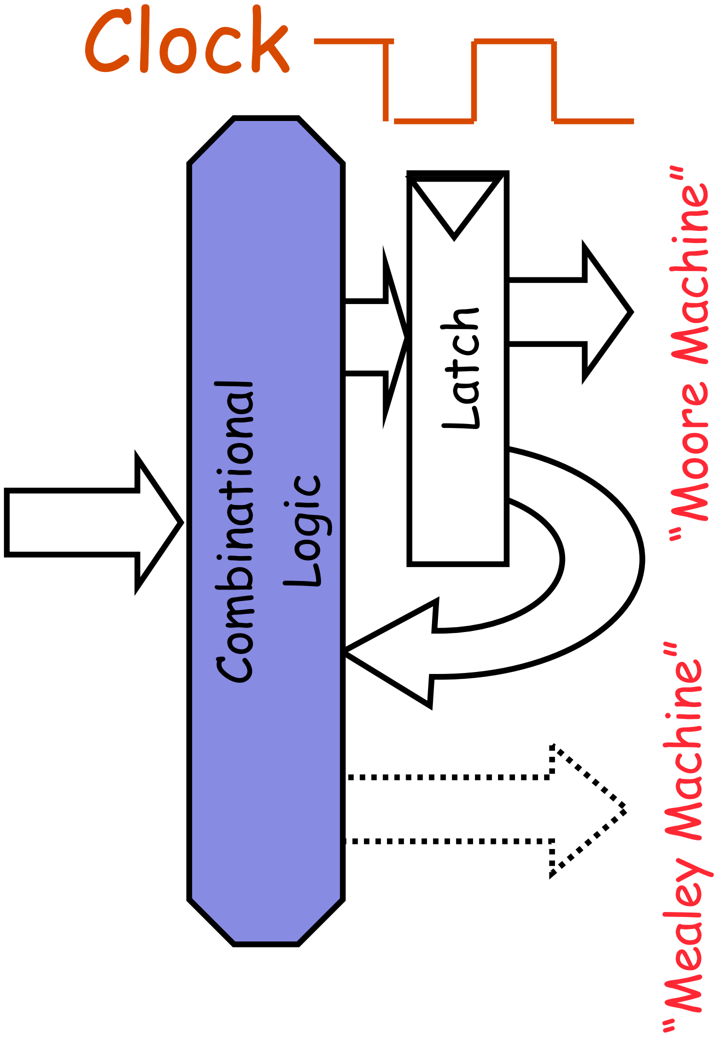
Performance

Design Complexity (Design Time)

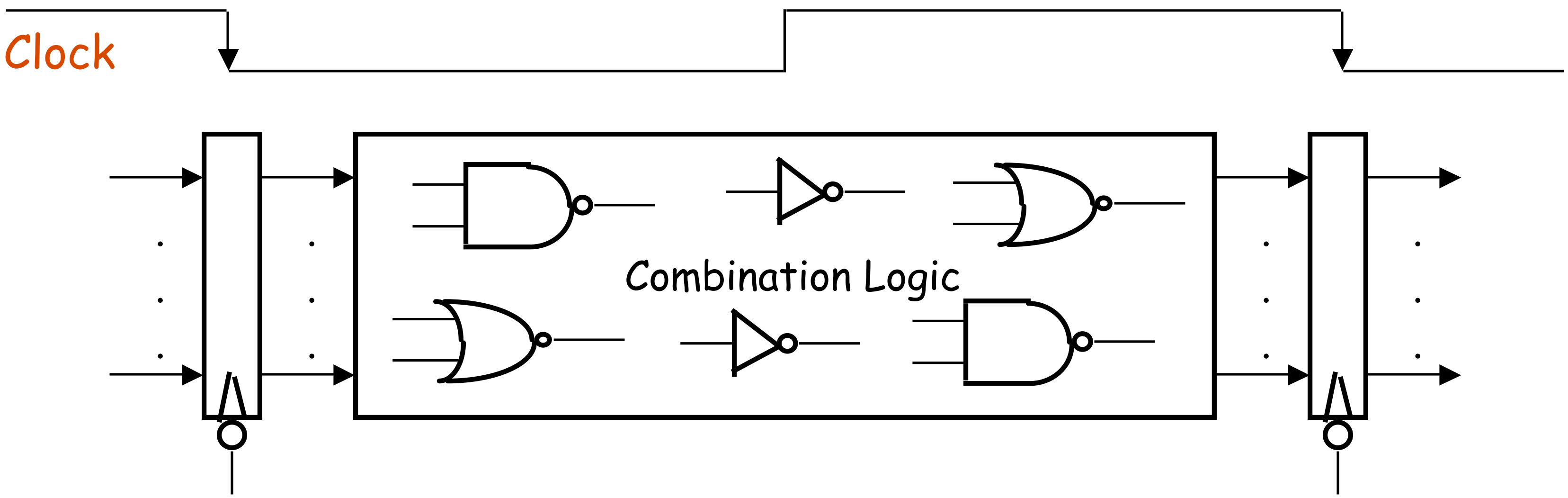
Compact

Longer wires

Implementation as Combinational Logic + Latch



Clocking Methodology

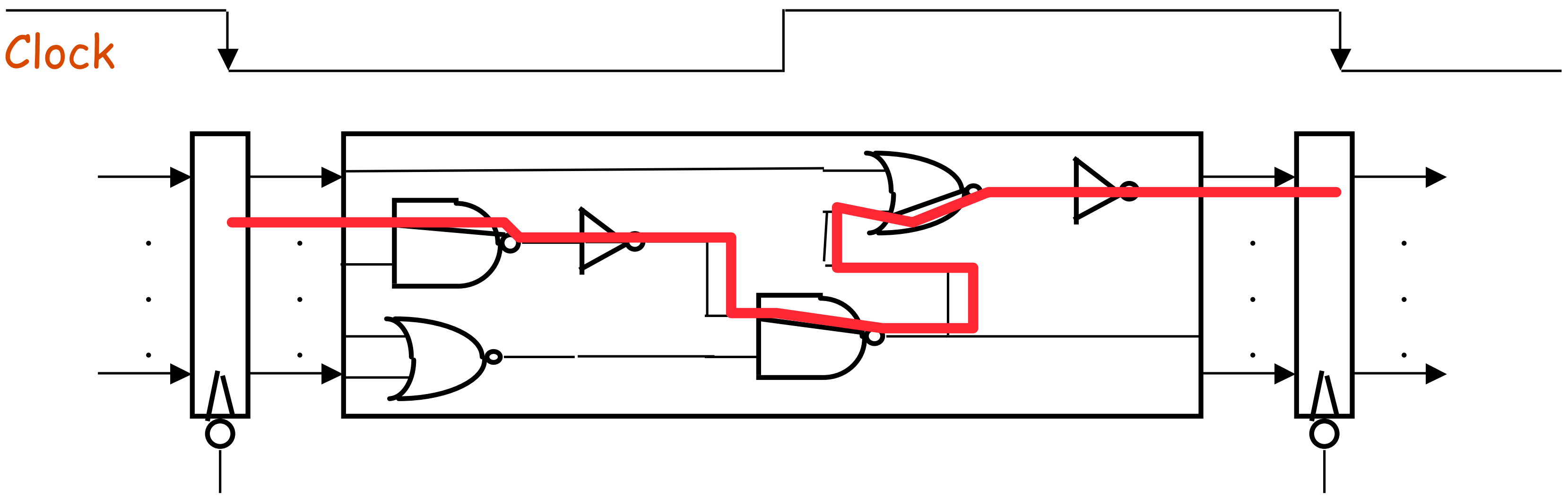


All storage elements are clocked by the same clock edge (but there may be clock skews)

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

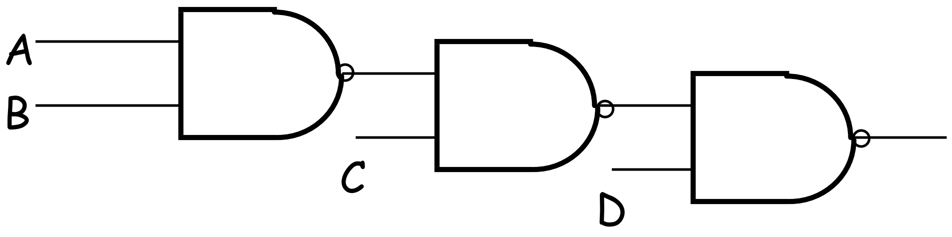


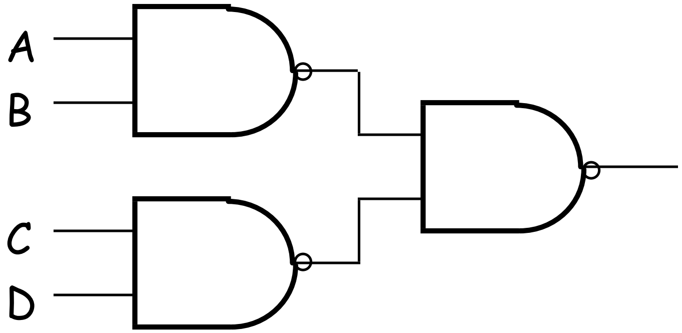
Critical path: the slowest path between any two storage devices

Cycle time is a function of the critical path

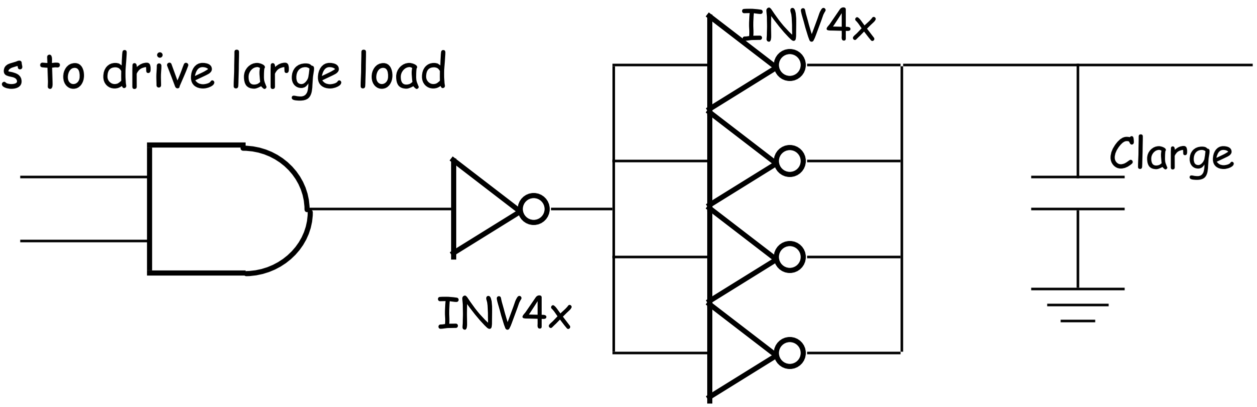
Tricks to Reduce Cycle Time

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

- Speedup what is important

Critical Path