CS425
Computer Systems Architecture

Fall 2018

Metrics

Previous Lecture

- CPU Evolution
- What is Computer Architecture?

Outline

- Measurements and metrics:
  - Performance, Cost, Dependability, Power
- Guidelines and principles in the design of computers
- CPU Performance

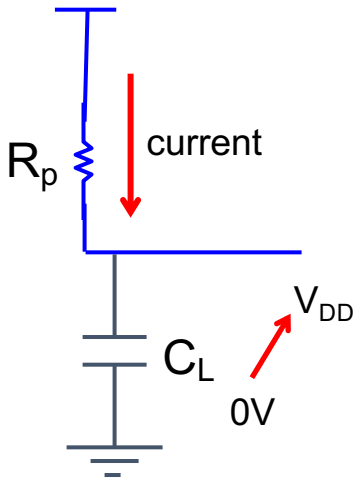
Major Design Challenges

- Power
- CPU time
- Memory latency/bandwidth
- Storage latency/bandwidth
- Transactions per second
- Intercommunication
- Dependability

Everything Looks a Little Different

Power Consumption

Charge external capacitance



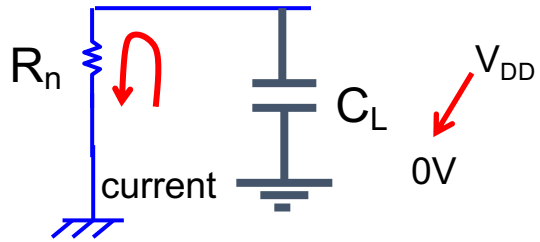
$Q = C_L V_{DD}$

$E_{dynamic} = Q V_{DD} = C_L V_{DD}^2$

½ E_d thermal energy on R_P
½ E_d stored on C_L

(since $E_{CL} = ½ C_L V_{DD}^2$)

Discharge external capacitance



½ $E_{dynamic}$ stored on C_L
becomes thermal energy on R_N

$P_{dynamic} = ½ C_L V_{DD}^2$ frequency

Power Equations

$$Power_{dynamic} = \frac{1}{2} \times Capacitive\ load \times Voltage^2 \times Frequency$$

$$Energy_{dynamic} = Capacitive\ load \times Voltage^2$$

$$Power_{static} = Current_{static} \times Voltage$$

- Power due to switching more transistors increases
- Static power due to leakage current increasing

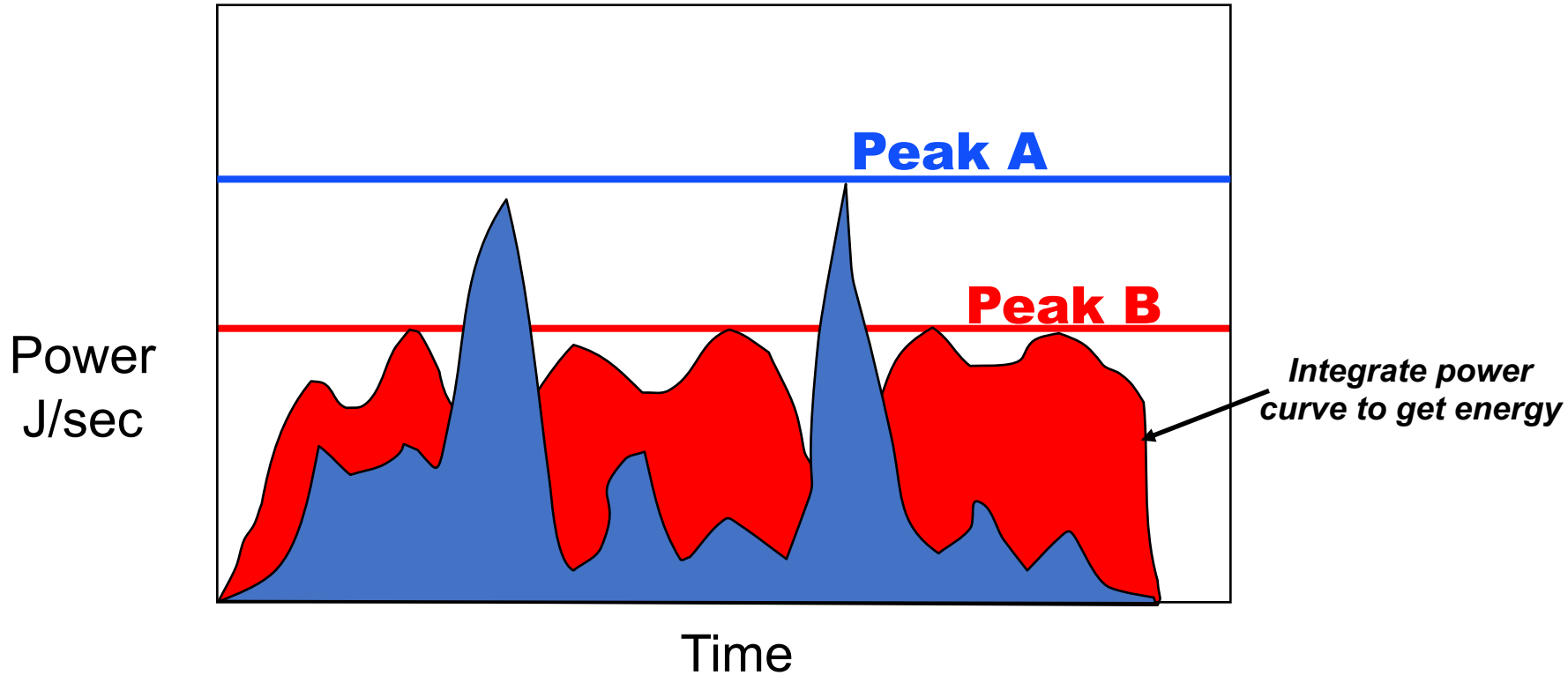
Power and Energy

- Energy to complete operation (Joules)
  - Corresponds approximately to battery life
  - (Battery energy capacity actually depends on rate of discharge)

- Peak power dissipation (Watts = Joules/second)
  - Affects packaging (power and ground pins, thermal design)

- d_i/d_t, peak change in supply current (Amps/second)
  - Affects power supply noise (power and ground pins, decoupling capacitors)

Peak Power versus Lower Energy



- System **A** has higher peak power, but lower total energy
- System **B** has lower peak power, but higher total energy

Measuring Reliability (Dependability)

Reliability equations

$$MTTF = \text{Mean Time To Failure}$$

$$FIT = \text{Failures In Time (per billion hours)} = \frac{10^9}{MTTF}$$

$$MTTR = \text{Mean Time to Repair} \quad (MTBF = MTTF + MTTR)$$

$$\text{Module availability} = \frac{MTTF}{MTTF + MTTR}$$

$$FIT_{system} = \sum_{i=1}^{\#components} FIT_i$$

MTTF = 1,000,000 hours → FIT = ?

Comparing Design Alternatives

Design X is n times faster than design Y

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- Wall-clock time: time to complete a task
- CPU time: time CPU is busy
- Workload: Mixture of programs (including OS) on a system
- Kernels: Common, important functions in applications
- Microbenchmarks: Synthetic programs trying to:
  - Isolate components and measure performance
  - Imitate workloads of real world in a controlled setting

Benchmark Suites

Desktop (SPEC = Standard Performance Evaluation Corporation, 12 INT, 17 FP, 1980)

- SPECCPU (revised every few years)
- Real programs measuring processor-memory activity

Multi-core desktop/server

- SPECOMP, SPECMPI (scientific), SPECapc (graphics)
- Focus on parallelism, synchronization, communication

Client/Server

- SPECjbb, SPECjms, SPECjvm, SPECsfs, SPECmail SPECrate, SPECWeb …
- Measuring throughput (how many tasks per unit of time)
- Measuring latency (how quickly does client get response)

Embedded systems

- EEMBC, MiBench
- Measuring performance, throughput, latency

The weakness of one benchmark is covered by the other benchmarks

Summarizing performance

Arithmetic mean of wall-clock time

- Biased by long-running programs
- May rank designs in non-intuitive ways:
  - Machine A: Program $P_1 \rightarrow$ 1000 secs., $P_2 \rightarrow$ 1 secs.
  - Machine B: Program $P_1 \rightarrow$ 800 secs., $P_2 \rightarrow$ 100 secs.
  - What if machine runs P_2 most of the time?

Means

- Total time ignores program contribution to total workload
- Arithmetic mean biased by long programs
- Weighted arithmetic mean a better choice?
- How do we calculate weights?

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = W(2) = 50

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	500.50	55.00	20.00

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = 0.909 W(2) = 0.091

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	91.91	18.19	20.00

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = 0.999 W(2) = 0.001

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	2.00	10.09	20.00

Summarizing performance (cont.)

Measuring against a reference computer

$$SPEC_{ratio_A} = \frac{Execution\ time_{reference}}{Execution\ time_A} = Performance_A / Performance_{reference}$$

$$n = \frac{SPEC_{ratio_A}}{SPEC_{ratio_B}} = \frac{\frac{Execution\ time_{reference}}{Execution\ time_A}}{\frac{Execution\ time_{reference}}{Execution\ time_B}} = \frac{Execution\ time_B}{Execution\ time_A} = \frac{Performance_A}{Performance_B}$$

Using ratios

- Ratios against reference machine are independent of running time of programs

Summarizing performance (cont.)

Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n} \text{SPEC}_{\text{ratio}}(i)}$$

$$\frac{\textit{Geometric mean}_A}{\textit{Geometric mean}_B} = \textit{Geometric mean}\left(\frac{A}{B}\right)$$

Used by SPEC98, SPEC92, SPEC95, …, SPEC2006

Pros and cons of geometric means

Pros

- Consistent rankings, independent of program frequencies
- Not influenced by peculiarities of any single machine

Cons

- Geometric mean does not predict execution time
  - Sensitivity to benchmark vs. machine remains
  - Encourages machine tuning for specific benchmarks
  - Benchmarks can not be touched, but compilers can!
- Any “averaging” metric loses information

Qualitative principles of design

Taking advantage of parallelism

- Use pipelining to overlap instructions
- Use multiple execution units
- Use multiple cores
- Use multiple processors to increase throughput (system level: scalability)

Locality (spatial and temporal locality)

- Programs reuse instructions and data
- 90-10 rule
  - 90% of execution time spent running 10% of instructions
- Programs access data in nearby addresses (spatial)

Qualitative principles of design (cont.)

Make the common case fast

- Trade-off's in design (e.g. performance vs. power/area)
- Provide efficient design for the common case
- Amdahl's Law

Example:
First optimize instruction fetch and decode unit instead of multiplier

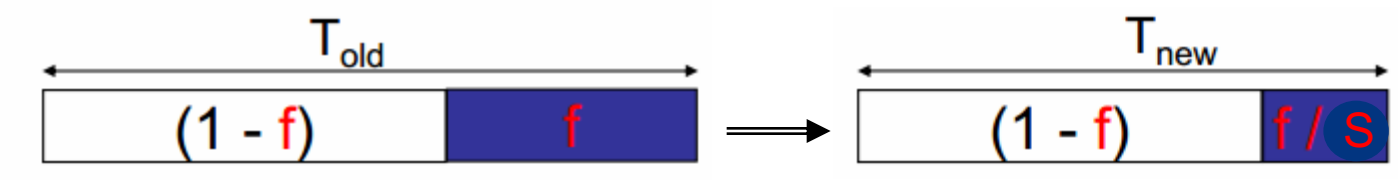
Amdahl's Law

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

$$execution\ time_{new} = execution\ time_{old} \times \left((1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}} \right)$$

$$speedup_{overall} = \frac{execution\ time_{old}}{execution\ time_{new}} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}}} \Rightarrow$$

Upper Limit: $speedup_{overall} \rightarrow \frac{1}{1 - fraction_{enhanced}}$



Amdahl's Law example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

Processor Performance

CPU time

$$\text{CPU time} = \text{CPU clock cycles} \times \text{Clock cycle time}$$

$$CPI = \frac{\text{CPU clock cycles}}{\text{instruction count}} \Rightarrow$$

$$\text{CPU time} = \text{instruction count} \times CPI \times \text{cycle time} \Rightarrow$$

$$\text{CPU time} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instructions}} \times \frac{\text{seconds}}{\text{clock cycles}}$$

Cycles Per Instruction (CPI)

"Average Cycles per Instruction"

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

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

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

"Instruction Frequency"

Example: Calculating CPI bottom up

Run benchmark and collect workload characterization (simulate, machine counters, or sampling)

Base Machine (Reg / Reg)

Op	Freq	CPI_i	$F*CPI_i$	(% Time)
ALU	50%	1	0.5	(33%)
Load	20%	2	0.4	(27%)
Store	10%	2	0.2	(13%)
Branch	20%	2	0.4	(27%)
			1.5	

Typical Mix of instruction types in program

Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction if it is shown to add 1% performance improvement on reasonable benchmarks.

Processor Performance

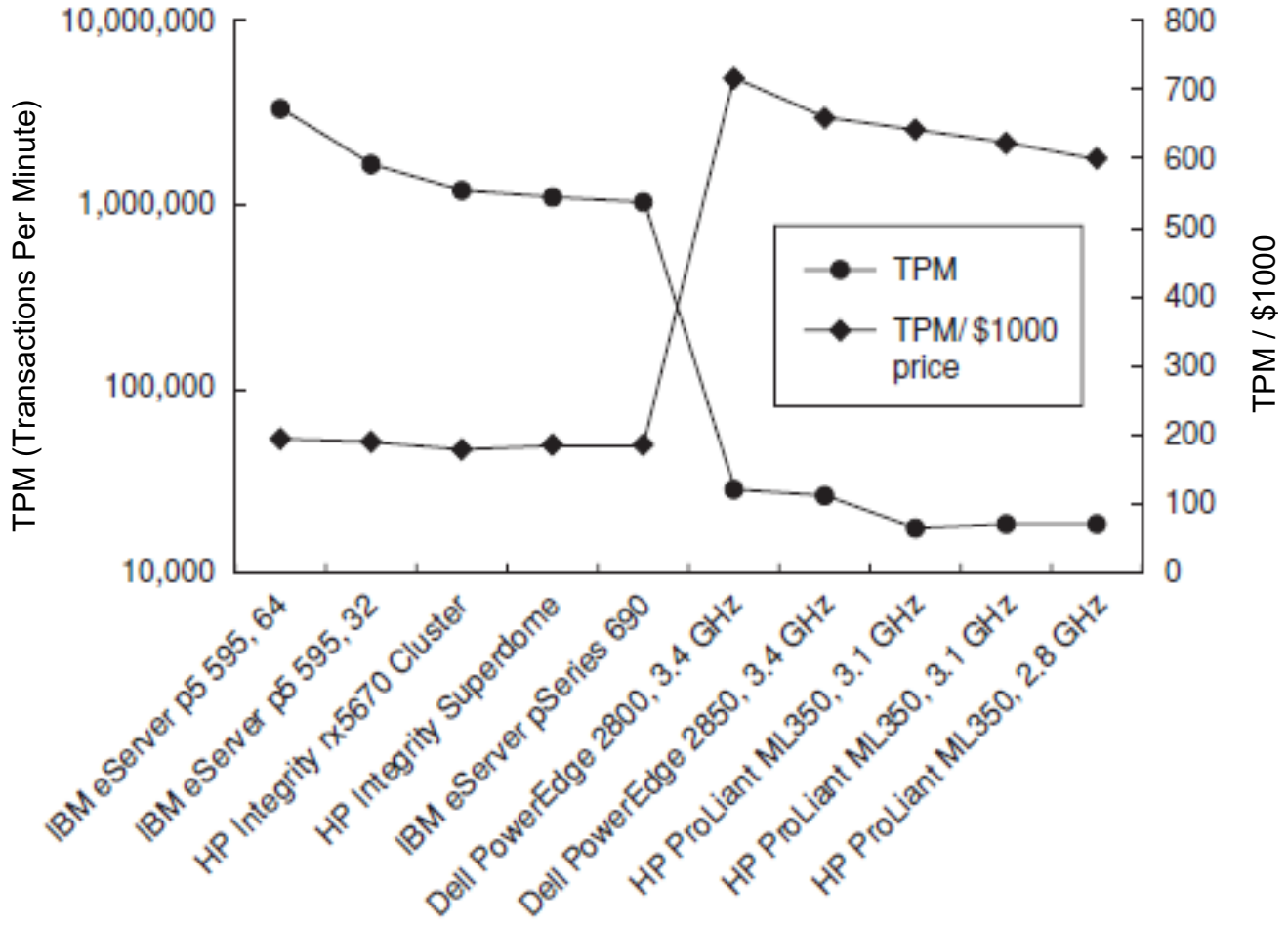
$$CPU\ time = instruction\ count \times CPI \times cycle\ time$$

How can CA help?

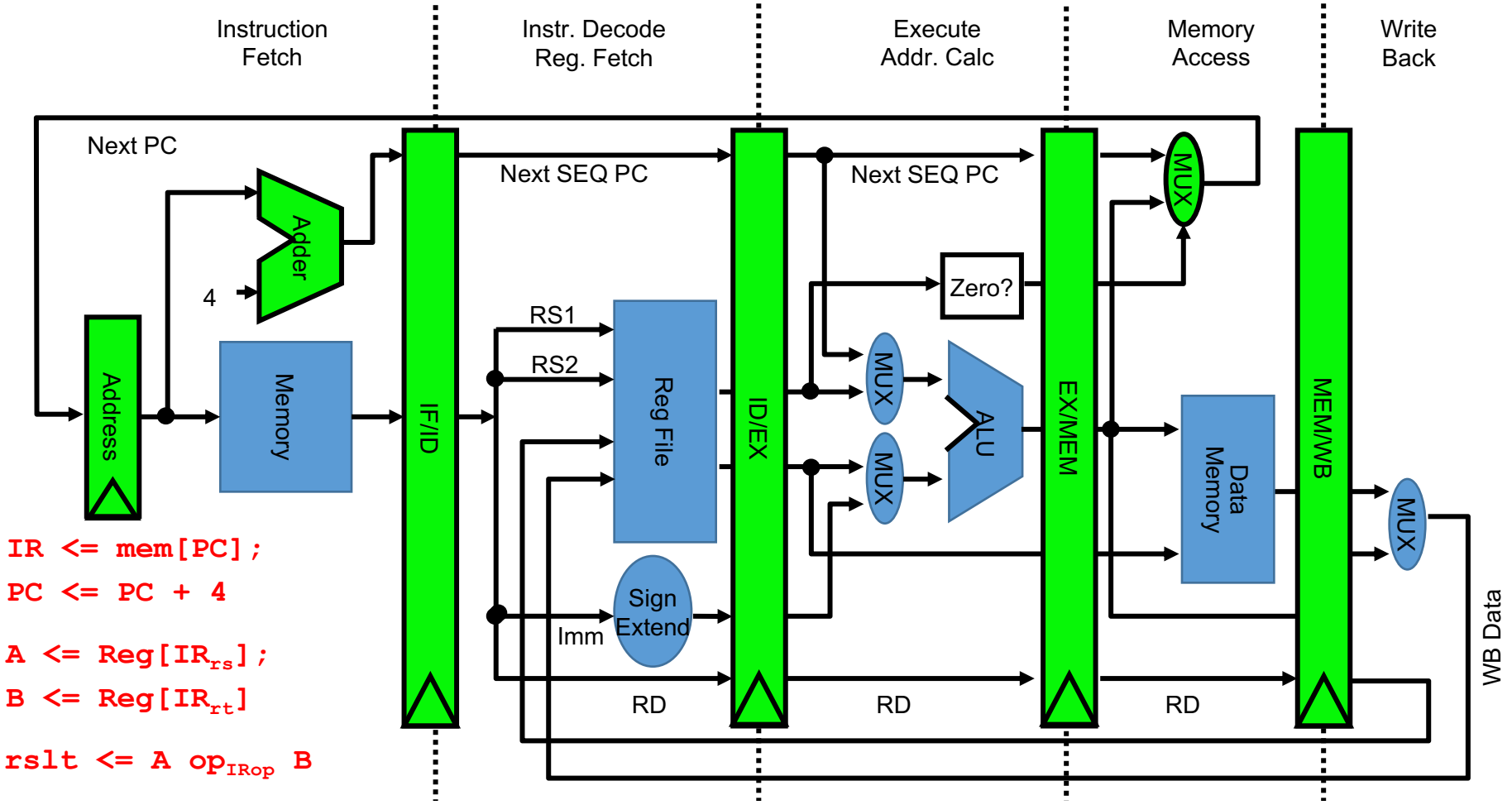
- Technology has been providing faster clock speeds
  - Main performance factor for almost 20 years
  - Trend seems to reverse
  - Limitations due to power consumption, reliability
- Architecture can pack more computing power in same area
- Architecture can improve CPI
- Algorithms and compilers can reduce instruction count

Price / Performance

benchmark for online transaction processing (OLTP) is TPC-C



Next Lecture : Pipelining




```
IR <= mem[PC];
PC <= PC + 4

A <= Reg[IR_rs];
B <= Reg[IR_rt]

rslt <= A op_IRop B

WB <= rslt

Reg[IR_rd] <= WB
```

- Data stationary control
  - local decode for each instruction phase / pipeline stage