Performance

Anne Bracy

CS 3410

Computer Science

Cornell University

These slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

Performance

Complex question

- How fast is the processor?
- How fast your application runs?
- How quickly does it respond to you?
- How fast can you process a big batch of jobs?
- How much power does your machine use?

Performance: Latency vs. Throughput

Latency (execution time): time to finish a fixed task

Throughput (bandwidth): # of tasks in fixed time

- Different: exploit parallelism for throughput, not latency (e.g., bread)
- Often contradictory (latency vs. throughput)
  - Will see many examples of this
- Use definition of performance that matches your goals
  - Scientific program: latency; web server: throughput?

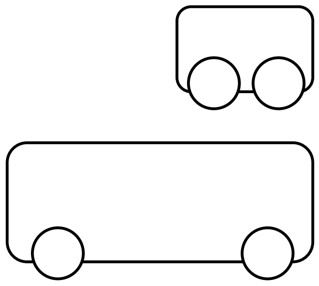
iClicker Question #1: Car vs. Bus

Car: speed = 60 miles/hour, capacity = 5

Bus: speed = 20 miles/hour, capacity = 60

Task: transport passengers 10 miles

	Latency (min)	**Throughput (PPH)**
Car		
Bus		



2 CLICKER QUESTIONS (Throughput)

A.	10
B.	15
C.	20
D.	60
E.	120

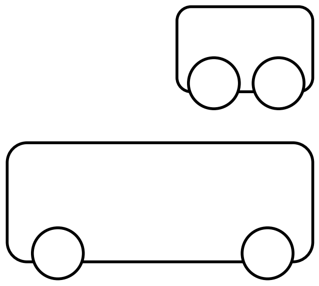
iClicker Question #1: Car vs. Bus

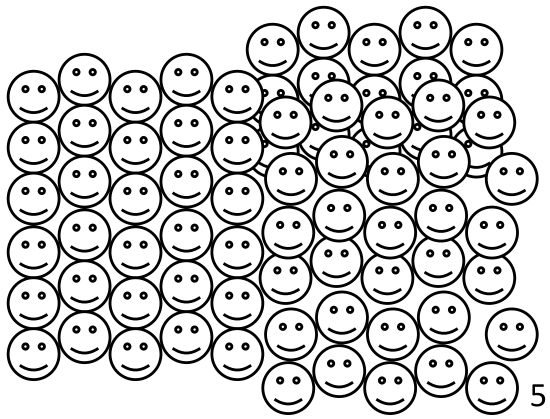
Car: speed = 60 miles/hour, capacity = 5

Bus: speed = 20 miles/hour, capacity = 60

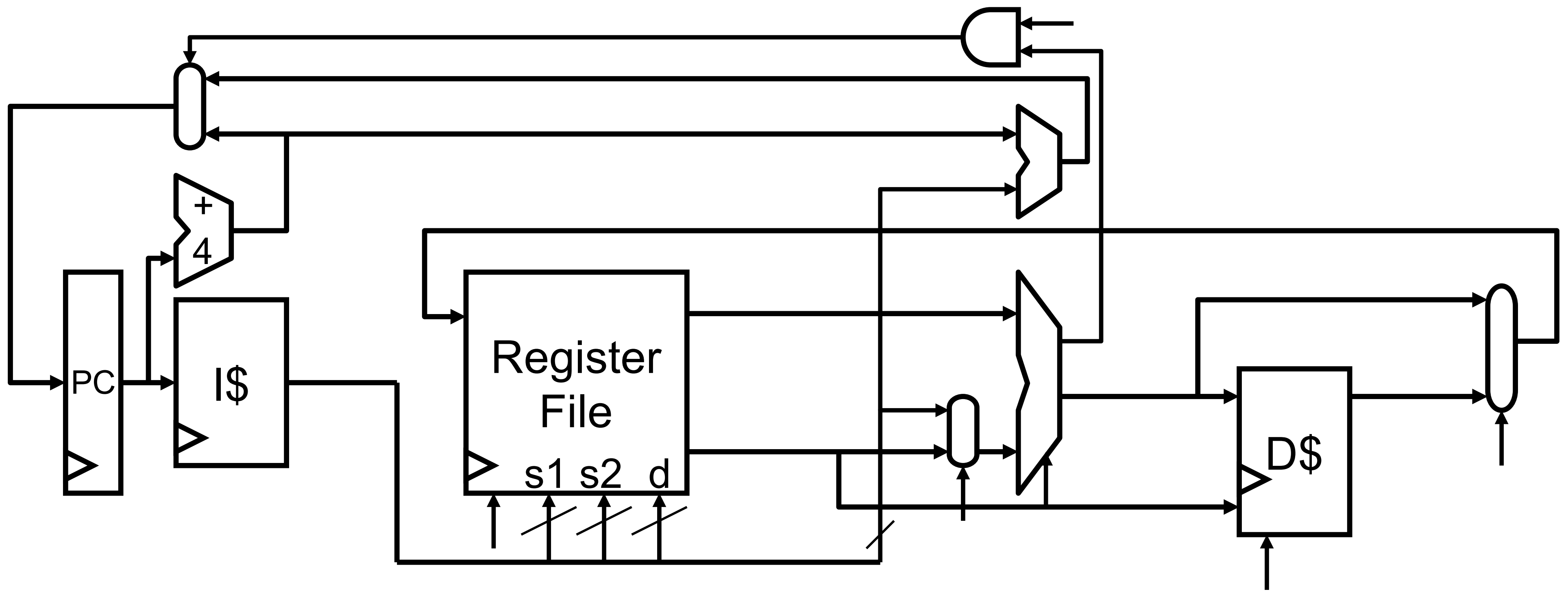
Task: transport passengers 10 miles

	Latency (min)	Throughput (PPH)
Car	**10 min**	**15 PPH**
Bus	**30 min**	**60 PPH**





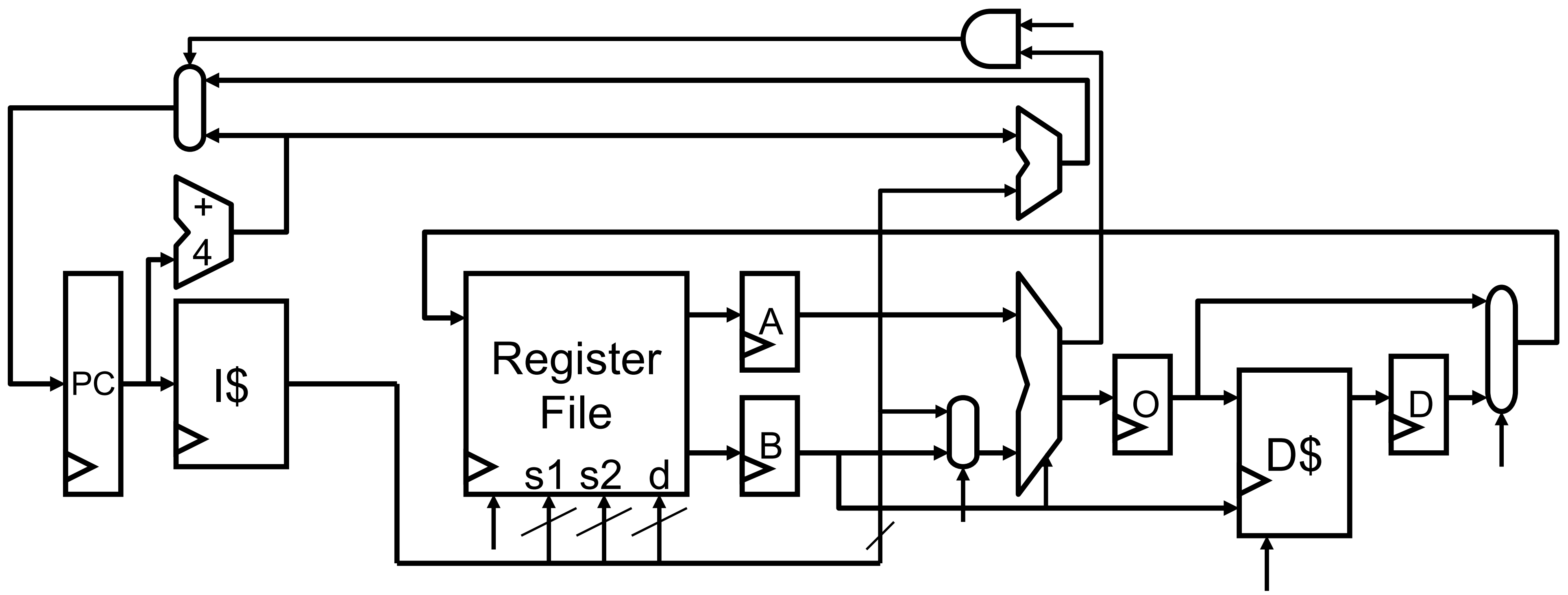
Review: Single-Cycle Datapath



Single-cycle datapath: true “atomic” fetch/execute loop
Fetch, decode, execute one instruction/cycle

+ Low CPI (see later slides): 1 by definition
– Long clock period: to accommodate slowest instruction
  (PC → I$ → RF → ALU → D$ → RF)

New: Multi-Cycle Datapath



Multi-cycle datapath: attacks slow clock
Fetch, decode, execute one insn over multiple cycles

Allows insns to take different number of cycles (main point)
±Opposite of single-cycle: short clock period, high CPI

Single- vs. Multi-cycle Performance

Single-cycle

- Clock period = 50ns, CPI = 1
- Performance = **50ns/insn**

Multi-cycle: opposite performance split

+ Shorter clock period
– Higher CPI

Example

- branch: 20% (**3** cycles), load: 20% (**5** cycles), ALU: 60% (**4** cycle)
- Clock period = **11ns**, CPI = (20%*3)+(20%*5)+(60%*4) = 4
  - Why is clock period 11ns and not 10ns?
- Performance = **44ns/insn**

Aside: CISC makes perfect sense in multi-cycle datapath

Processor Performance Equation

Program runtime:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

Instructions per program: "dynamic instruction count"

- Runtime count of instructions executed by the program
- Determined by program, compiler, ISA

Cycles per instruction: "CPI" (typical range: 2 to 0.5)

- How many *cycles* does an instruction take to execute?
- Determined by program, compiler, ISA, micro-architecture

Seconds per cycle: clock period, length of each cycle

- Inverse metric: cycles/second (Hertz) or cycles/ns (Ghz)
- Determined by micro-architecture, technology parameters

For lower latency (=better performance) minimize all three

- Difficult: ***often pull against one another***

Cycles per Instruction (CPI)

CPI: Cycle/instruction for **on average**

- **IPC** = 1/CPI
  - Used more frequently than CPI
  - Favored because "bigger is better", but harder to compute with
- Different instructions have different cycle costs
  - E.g., "add" typically takes 1 cycle, "divide" takes >10 cycles
- Depends on relative instruction frequencies

CPI example

- Program has equal ratio: integer, memory, floating point
- Cycles per insn type: integer = 1, memory = 2, FP = 3
- What is the CPI? (33% * 1) + (33% * 2) + (33% * 3) = 2
- **Caveat**: this sort of calculation ignores many effects
  - Back-of-the-envelope arguments only

iClicker Question #2: CPI

Assume a processor with instruction frequencies and costs

- Integer ALU: 50%, 1 cycle
- Load: 20%, 5 cycle
- Store: 10%, 1 cycle
- Branch: 20%, 2 cycle

Which change would improve performance more?

A: “Branch prediction” to reduce branch cost to 1 cycle?

B: “Cache” to reduce load cost to 3 cycles?

Compute CPI

	INT	LD	ST	BR	CPI
Base					
A					
B					

A. A better
B. B better
C. C equal
D. D can’t say

iClicker Question #2: CPI

Assume a processor with instruction frequencies and costs

- Integer ALU: 50%, 1 cycle
- Load: 20%, 5 cycle
- Store: 10%, 1 cycle
- Branch: 20%, 2 cycle

Which change would improve performance more?

A: “Branch prediction” to reduce branch cost to 1 cycle?

B: “Cache” to reduce load cost to 3 cycles?

Compute CPI

	INT	**LD**	**ST**	**BR**	**CPI**	
Base	0.5 x 1	0.2 x 5	0.1 x 1	0.2 x 2	**2.0**	
A	0.5 x 1	0.2 x 5	0.1 x 1	**0.2 x 1**	**1.8**	
B	0.5 x 1	**0.2 x 3**	0.1 x 1	0.2 x 2	**1.6**	(**winner**)

Mhz (MegaHertz) and Ghz (GigaHertz)

1 Hertz = 1 cycle/second
1 Ghz = 1 cycle/nanosecond, 1 Ghz = 1000 Mhz

General public (mostly) ignores CPI

- Equates clock frequency with performance!

Which processor would you buy?

- Processor A: CPI = 2, clock = 5 GHz
- Processor B: CPI = 1, clock = 3 GHz
- Probably A, but B is faster (assuming same ISA/compiler)

Classic example

- 800 MHz PentiumIII faster than 1 GHz Pentium4!
- Example: Core i7 faster clock-per-clock than Core 2
- Same ISA and compiler!

Meta-point: danger of partial performance metrics!

MIPS (performance metric, not the ISA)

(Micro) architects often ignore dynamic instruction count

- Typically have one ISA, one compiler → treat it as fixed

CPU performance equation becomes

Latency: $$\frac{\text{seconds}}{\text{insn}} = \frac{\text{cycles}}{\text{insn}} \times \frac{\text{seconds}}{\text{cycle}}$$

Throughput: $$\frac{\text{insn}}{\text{seconds}} = \frac{\text{insn}}{\text{cycles}} \times \frac{\text{cycles}}{\text{second}}$$

MIPS (millions of instructions per second)

- **Cycles / second**: clock frequency (in MHz)
- Ex: CPI = 2, clock = 500 MHz → 0.5 * 500 MHz = 250 MIPS

Pitfall: may vary inversely with actual performance

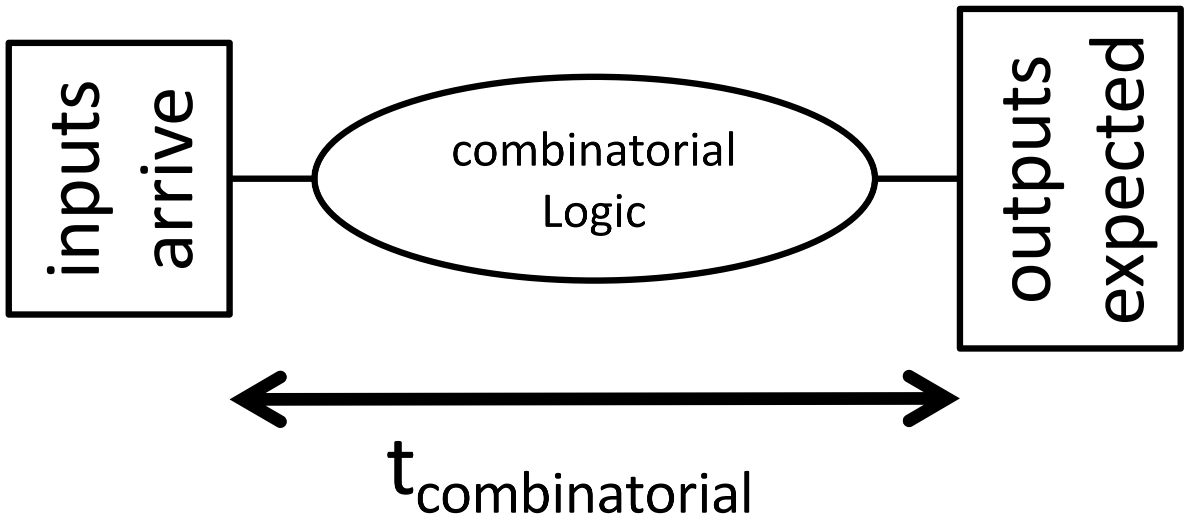
– Compiler removes insns, program faster, but lower MIPS
– Work per instruction varies (multiply vs. add, FP vs. integer)

How to make the computer faster?

Decrease latency

Critical Path

- Longest path determining the minimum time needed for an operation
- Determines minimum length of clock cycle
  i.e. determines maximum clock frequency



iClicker Question #3

Goal: Make Multi-Cycle @ 30 MHz CPU (15MIPS) run 2x faster by making arithmetic instructions faster

Instruction mix (for P):

- 25% load/store, CPI = 3
- 60% arithmetic, CPI = 2
- 15% branches, CPI = 1

A. 2.0	A. 1.0
B. 2.1	B. 1.3
C. 2.2	C. 1.4
D. 2.3	D. 1.5
E. 2.4	E. 2.0

(1) What is CPI?

Goal: Make processor run 2x faster (30→ 15 MIPS)
Try: Arithmetic 2 → 1? **(2)**
(2→X what would x have to be?)

iClicker Question #3

Goal: Make Multi-Cycle @ 30 MHz CPU (15MIPS) run 2x faster by making arithmetic instructions faster

Instruction mix (for P):

- 25% load/store, CPI = 3
- 60% arithmetic, CPI = 2
- 15% branches, CPI = 1

What is CPI? $= 0.25 \times 3 + 0.6 \times 2 + 0.15 \times 1 = 2.1$

Goal: Make processor run 2x faster (30→ 15 MIPS)

Try: Arithmetic 2 → 1? $= 0.75 + 0.6 \times \underline{\mathbf{1}} + 0.15 = 1.5$

(2→X what would x have to be?)

$1.05 = 0.75 + 0.6x + 0.15$ → $x = 0.25$ (yikes!)

Amdahl's Law

Amdahl's Law

Execution time after improvement =

$$\frac{\text{execution time affected by improvement}}{\text{amount of improvement}} + \text{execution time unaffected}$$

Or: Speedup is limited by popularity of improved feature

Corollary: **build a balanced system**

- Don't optimize 1% to the detriment of other 99%
- Don't over-engineer capabilities that cannot be utilized

Caveat: Law of diminishing returns