

Computer Organization and Architecture

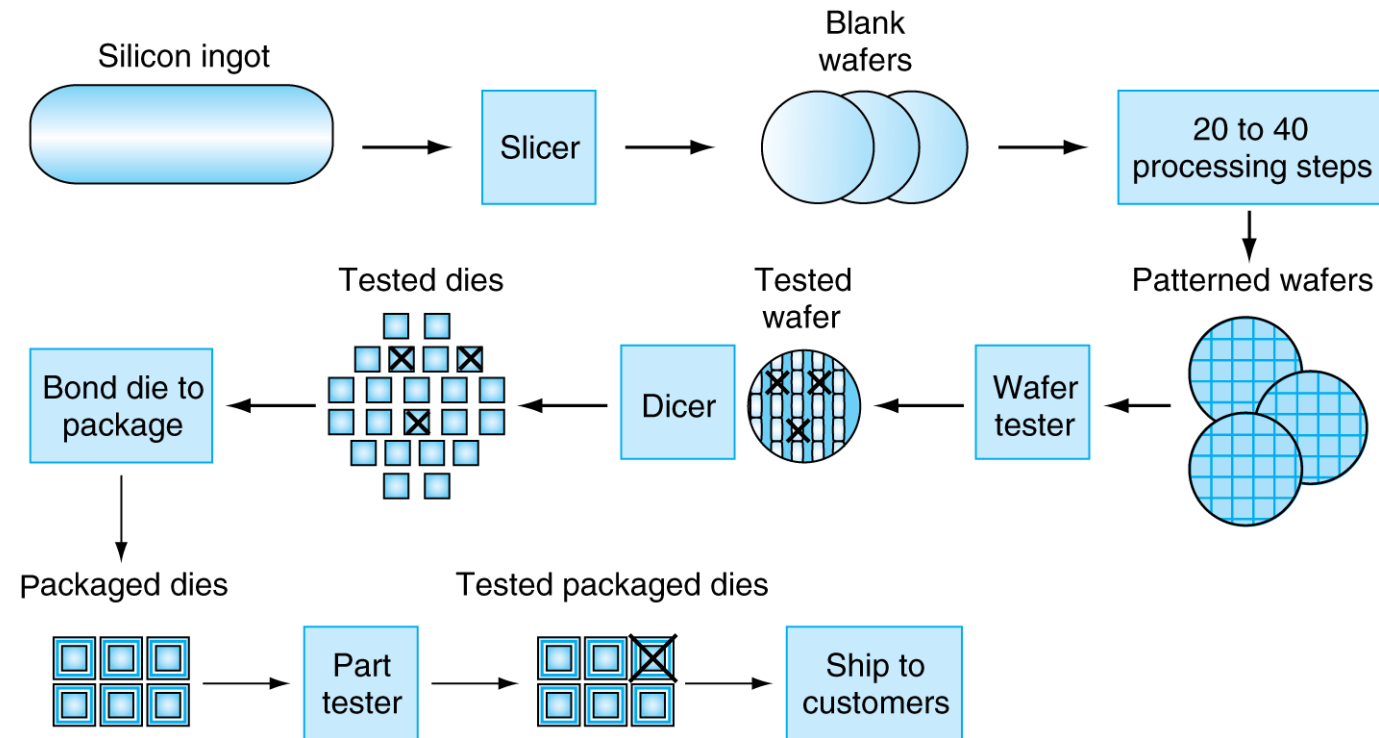
COSC 2425

Lecture – 3

Aug 29th, 2022

Acknowledgement: Slides from Edgar Gabriel & Kevin Long

Manufacturing ICs



Integrated Circuit Cost

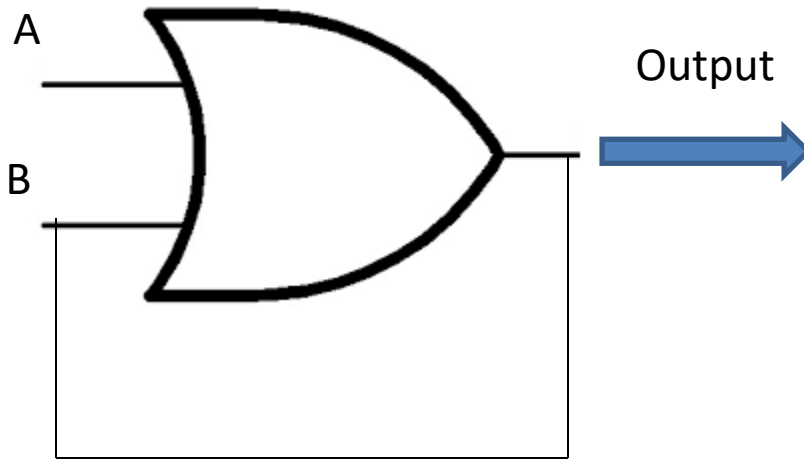
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

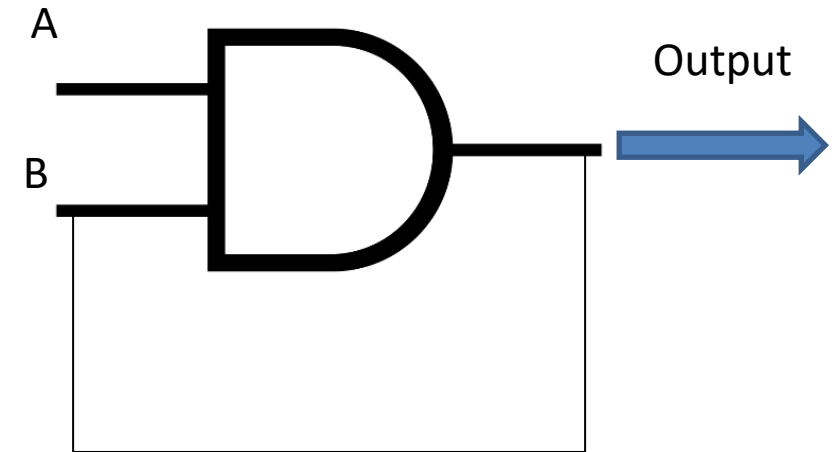
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
 - Wafer cost and area are fixed
 - Defect rate determined by manufacturing process
 - Die area determined by architecture and circuit design

Remembering 1's and 0's

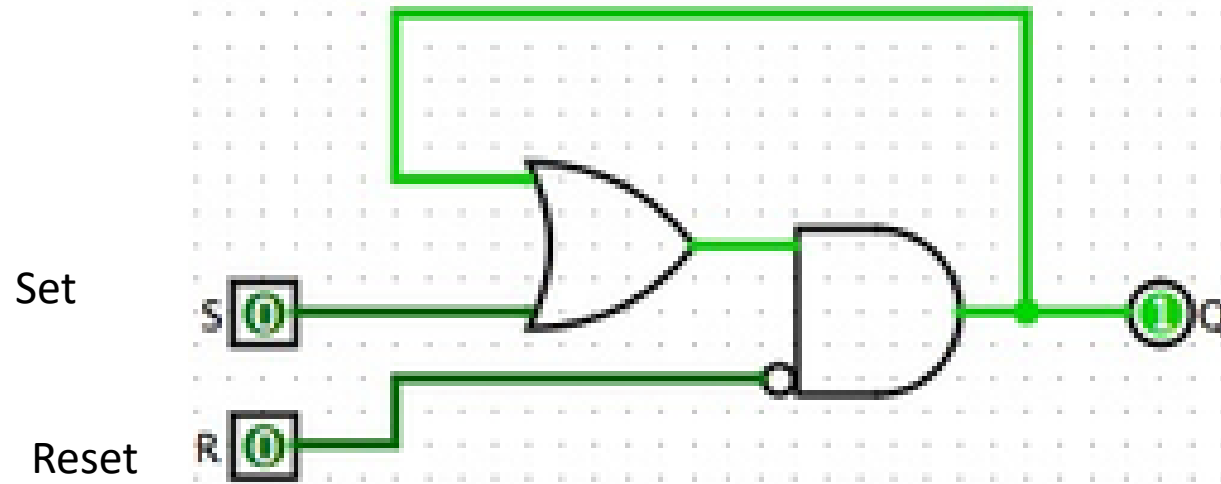


A	B	Output
0	0	0
1	0	1
0	X	1
1	X	1



A	B	Output
1	1	1
0	1	0
1	X	0
0	X	0

Combine Both: SR And-OR **Latch**



S	R	Output
1	0	1
0	0	1
0	1	0
0	0	0

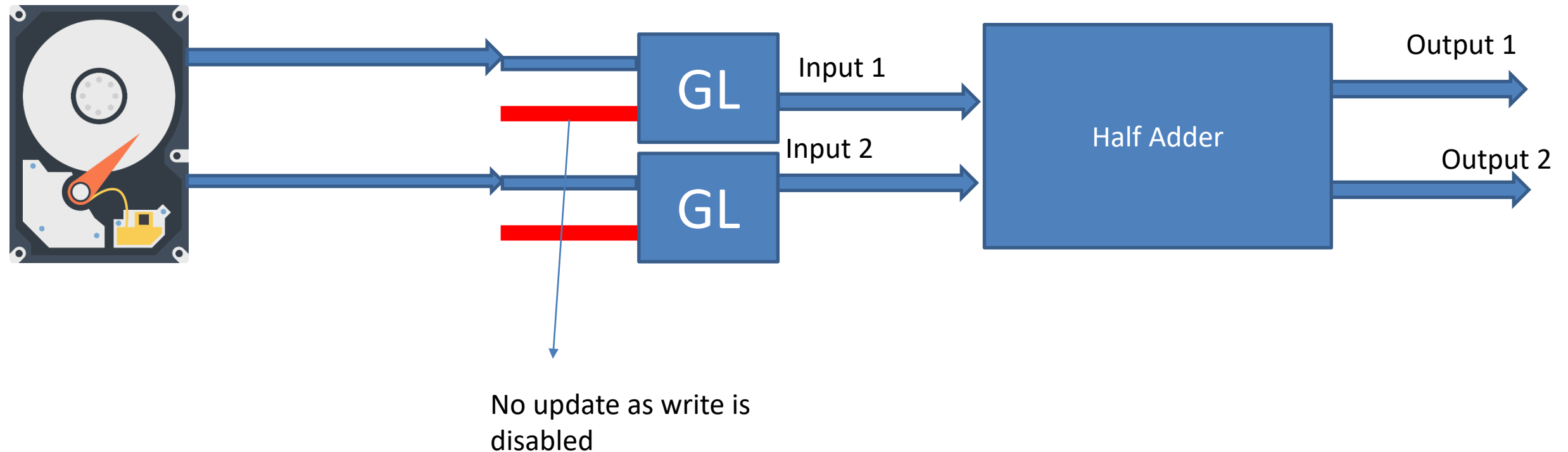
Gated Latch



I	W	O
0	0	0
1	0	0 (No update)
0	1	0 (Same as I)
1	1	1 (Same as I)
0	0	1 (No update)
1	0	1 (No update)

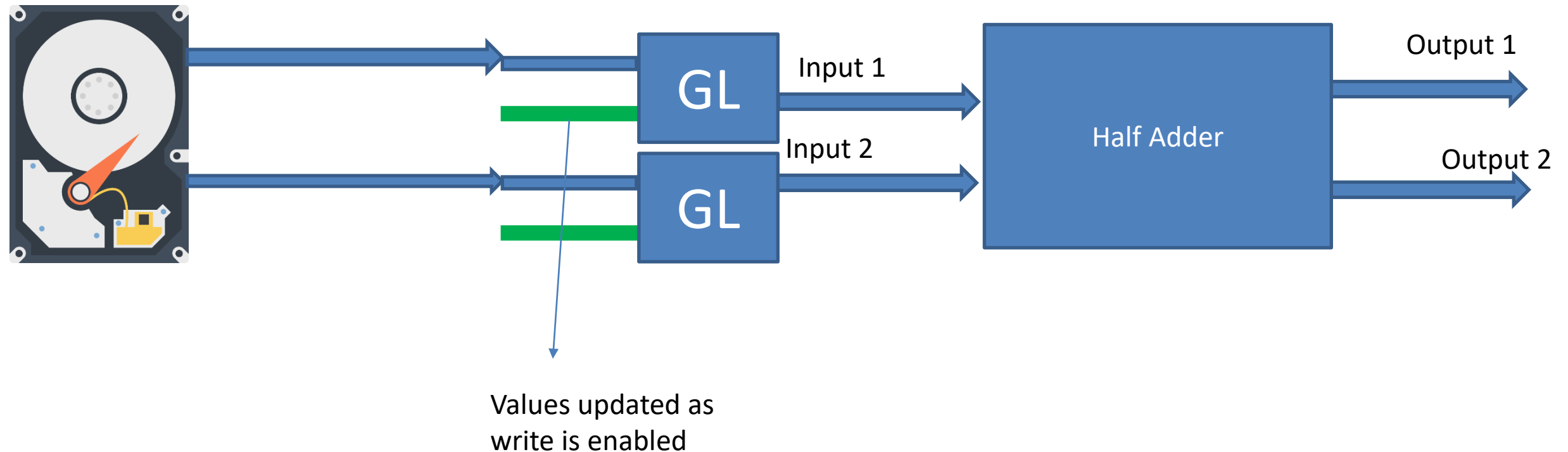
Half-Adder with manual input

Stored in memory (E.g. HDD)

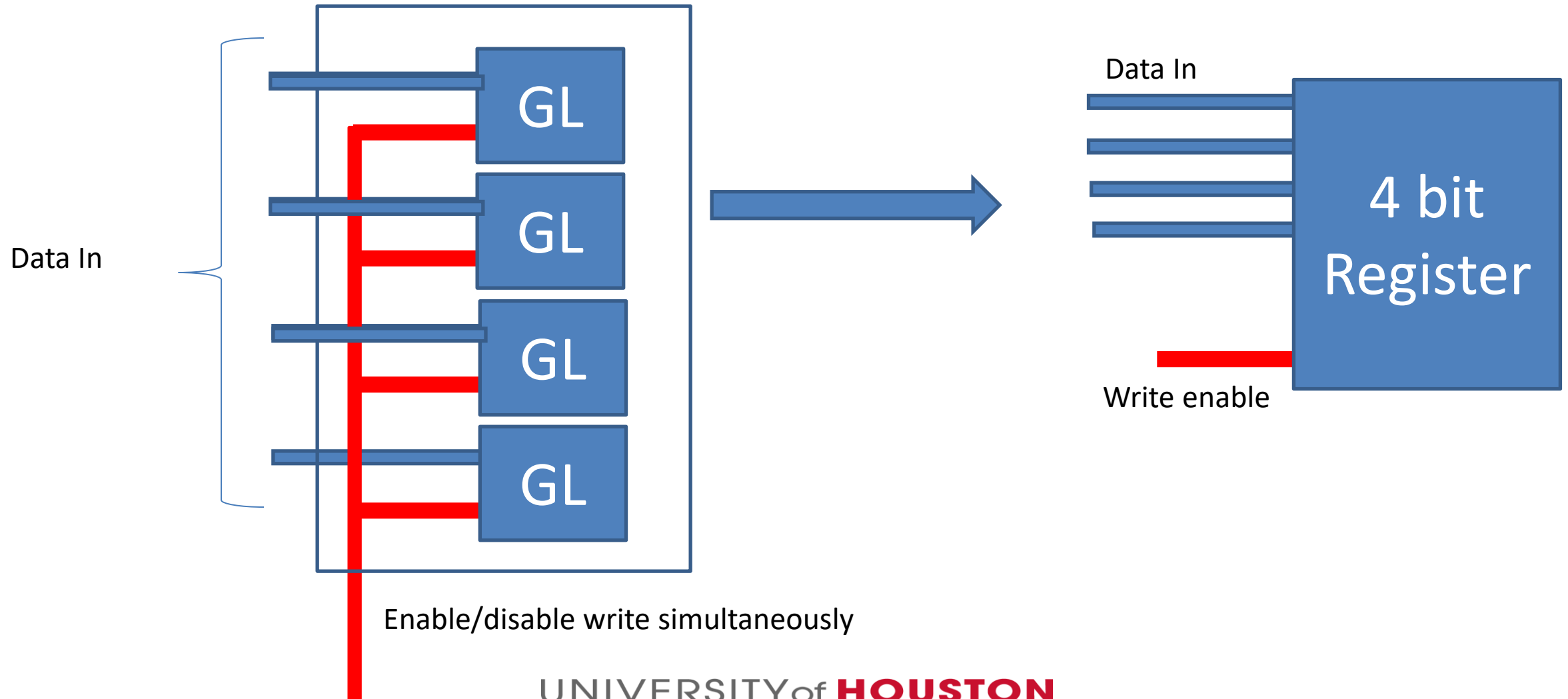


Half-Adder with manual input

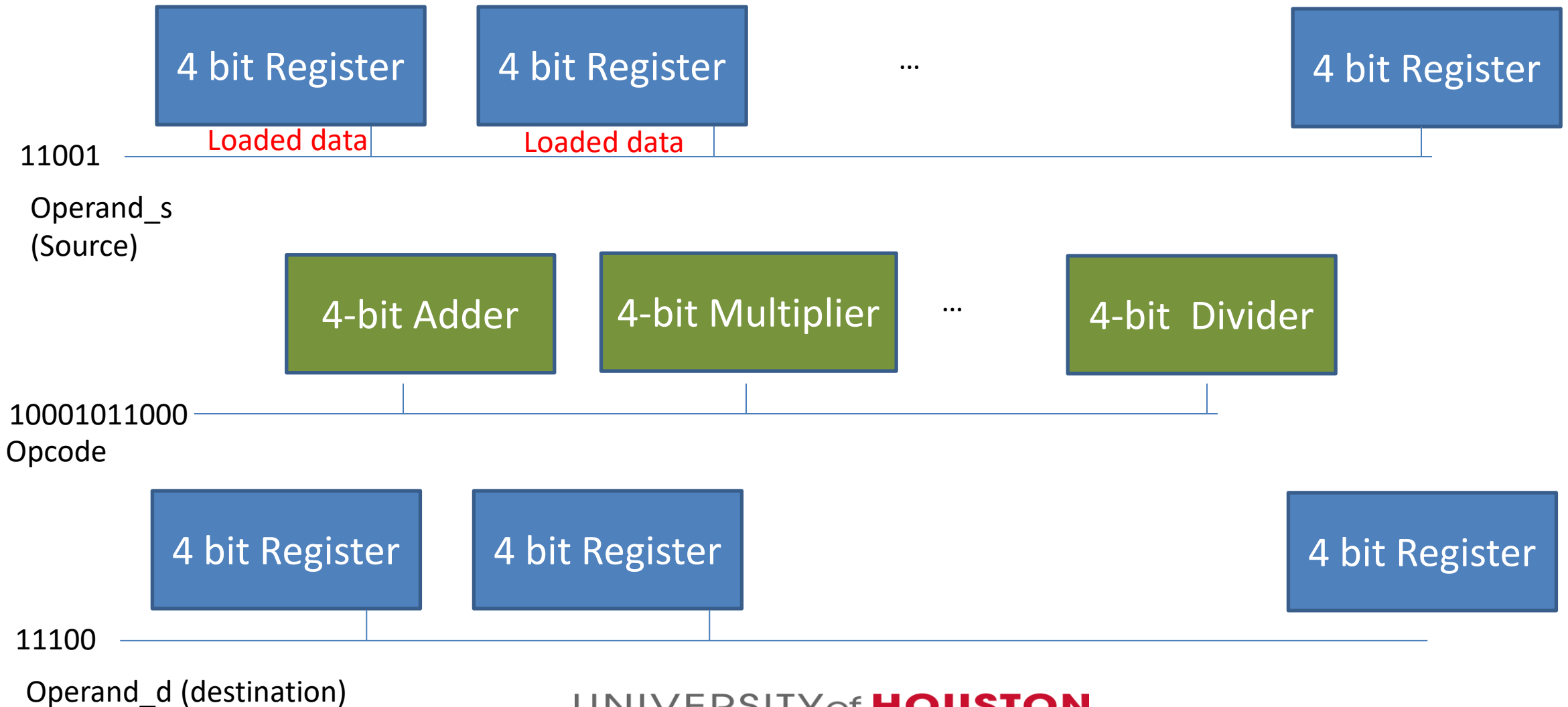
Stored in memory (E.g. HDD)



Register (4 bit) – A group of latches



Instruction



Instruction Example

Opcode	Operand_s1	Operand_s2	Operand_d
10001011000	11001	11010	11100

Instruction : 10001011000 11001 11010 11100

Instructions are represented in binary form. Stored in memory.
The only language a computer understands.
Byte code, machine code, ...

Levels of Program Code

- High-level language
 - Level of abstraction closer to problem domain
 - Provides for productivity and portability
- Assembly language
 - Textual representation of instructions
- Hardware representation
 - Binary digits (bits)
 - Encoded instructions and data

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for ARMv8)

```
swap:
    LSL  X10, X1,3
    ADD  X10, X0,X10
    LDUR X9, [X10,0]
    LDUR X11,[X10,8]
    STUR X11,[X10,0]
    STUR X9, [X10,8]
    BR   X10
```

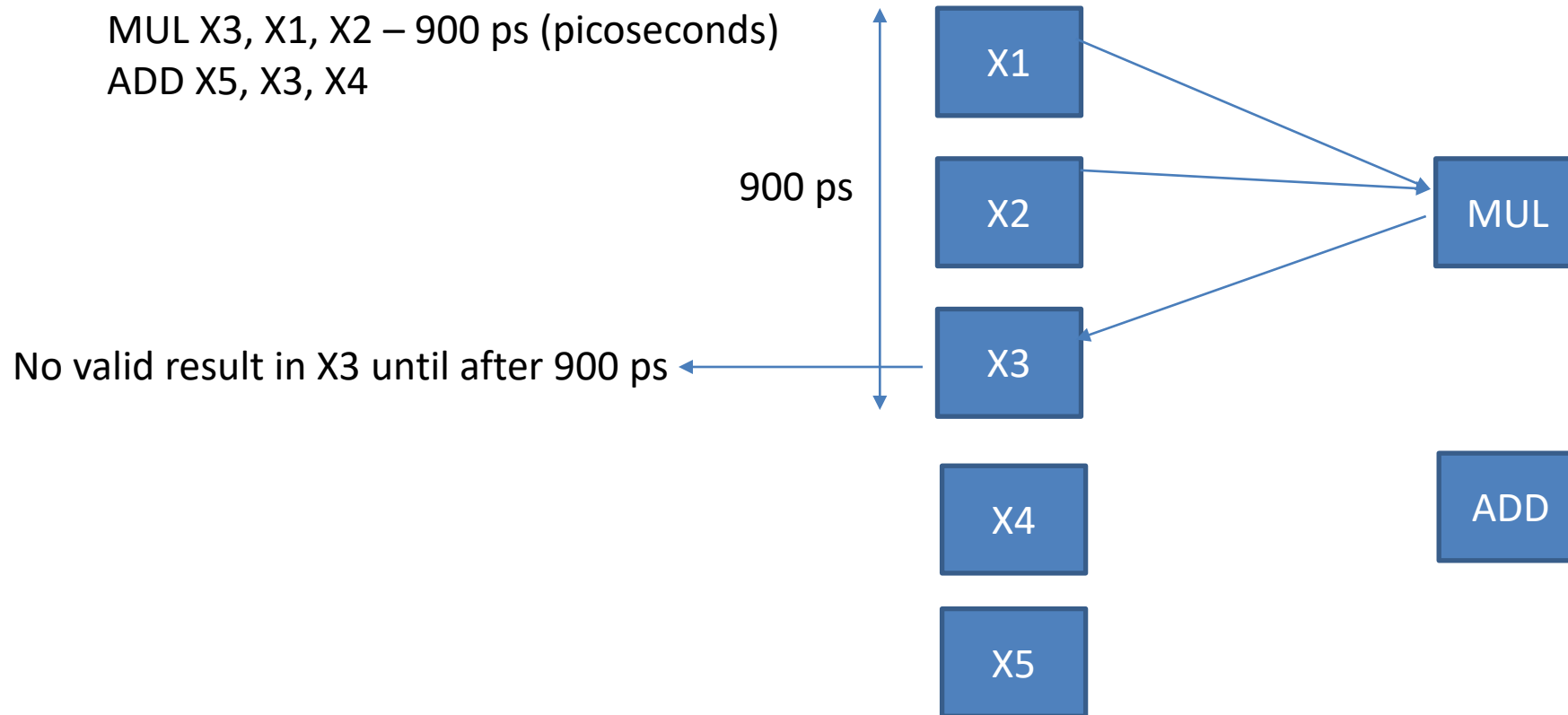
Assembler

Binary machine
language
program
(for ARMv8)

```
000000001010001000000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
100011100001001000000000000000100
101011100001001000000000000000000
101011011110001000000000000000100
00000011111000000000000000001000
```

Execution in Sequence

- CPU executes instructions in sequence.

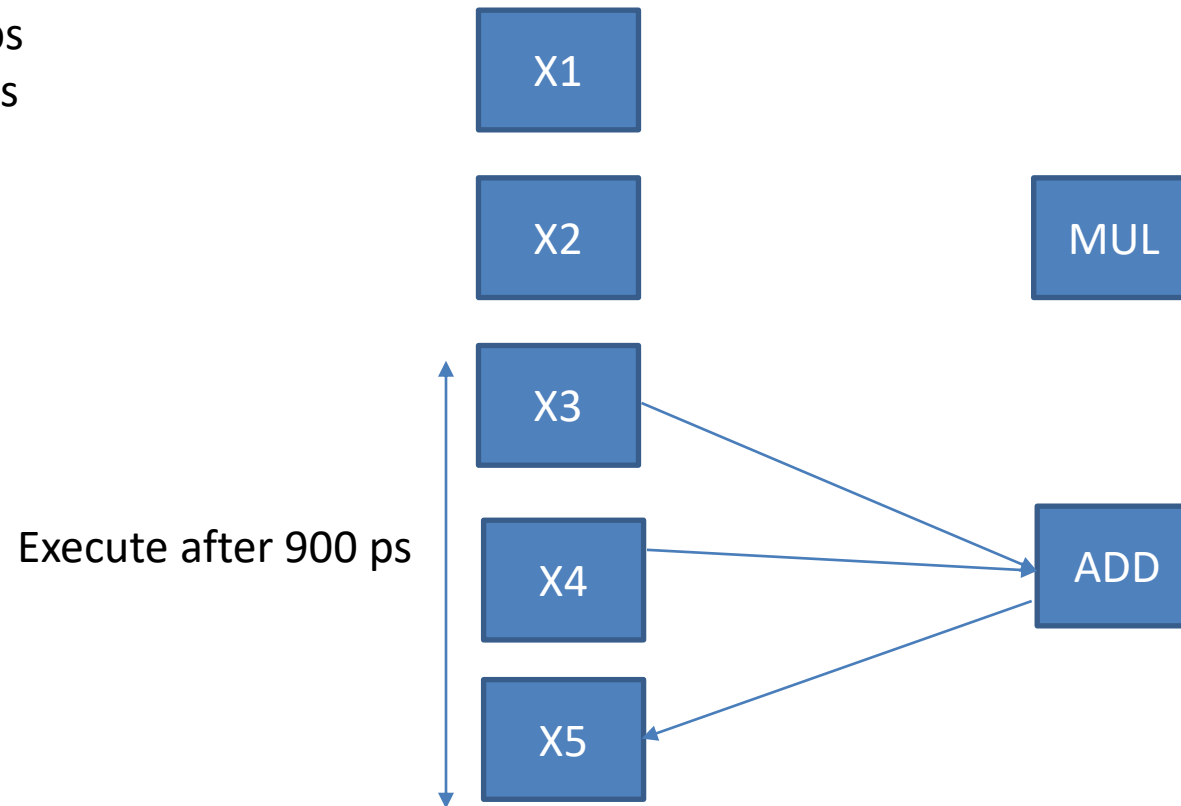


Execution in Sequence

- CPU executes instructions in sequence.

MUL X3, X1, X2 – 900 ps

ADD X5, X3, X4 – 200 ps



Execution in Sequence

- CPU executes instructions in sequence.

Operation	Time (ps)
Add	200
Mul	900
Div (Max)	1200

- Hundreds of instructions.
 - Too complicated to compute how much to wait.
- Choose the largest values as the **clock period** for all instructions.
- All instructions are executed for that period of time.

CPU Clocking

- Clock period: duration of a clock cycle
 - e.g., $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate in Hertz): cycles per second

$$\begin{aligned} &= \frac{1}{(250 \times 10^{-12})} \text{Hz} = \left(\frac{1000}{250} \right) * 10^9 \text{Hz} = 4 * 10^9 \text{Hz} \\ &= 4 * 10^6 \text{KHz} = 4 * 10^3 \text{MHz} = \mathbf{4 \text{ GHz}} \end{aligned}$$

Clock Cycles Per Instruction (CPI)

- All take **at-least** one clock cycle.
- Some instructions can take more than one clock cycle.
- If the instruction set has only **three** instructions

Operation	Clock Cycles
Add	1
Mul	2
Div	12

$$\text{Average Clock Cycle Per Instruction (CPI)} = \frac{1+2+12}{3} = 5$$

Defining Performance

- Which airplane has the best performance?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time
 - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
 - Replacing the processor with a faster version?
 - Adding more processors?
- We'll focus on response time for now...

Relative Performance

- Define Performance = $1/\text{Execution Time}$
- “X is n time faster than Y” or
- “Speedup of X over Y “ is

$$\begin{aligned} & \text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n \end{aligned}$$

- Example: time taken to run a program
 - 10s on A, 15s on B
 - $\text{Execution Time}_B / \text{Execution Time}_A$
 $= 15\text{s} / 10\text{s} = 1.5$
 - So A is 1.5 times faster than B
 - Speedup of A over B is 1.5



Measuring Execution Time

- Elapsed time
 - Total response time, including all aspects
 - Processing, I/O, OS overhead, idle time
 - Determines system performance
- CPU time
 - Time spent processing a given job
 - Discounts I/O time, other jobs' shares
 - Comprises user CPU time and system CPU time
 - Different programs are affected differently by CPU and system performance

CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
 - Reducing number of clock cycles
 - Increasing clock rate
 - Hardware designer must often trade off clock rate against cycle count

MUL X3, X1, X2 → 1200ps →  600ps X  2 Clock cycles

Instruction Count and CPI

$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$

$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$

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

- Instruction Count for a program
 - Determined by program, ISA and compiler
- Average cycles per instruction
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
 - Algorithm: affects IC, possibly CPI
 - Programming language: affects IC, CPI
 - Compiler: affects IC, CPI
 - Instruction set architecture: affects IC, CPI, T_c

Energy Consumed by Transistor

Dominant technology is CMOS (complementary metal oxide semiconductor).

The primary source for energy consumption is called dynamic energy

0 → 1 → 0

1 → 0 → 1

$$\text{Energy} \propto \text{Capacitive load} \times \text{Voltage}^2$$

Energy stored transistor

Energy Consumed by Transistor

Dominant technology is CMOS (complementary metal oxide semiconductor).

The primary source for energy consumption is called dynamic energy

$0 \rightarrow 1 \rightarrow 0$

$1 \rightarrow 0 \rightarrow 1$

$$\text{Energy} \propto \text{Capacitive load} \times \text{Voltage}^2$$

Energy consumed by transistor to switch states.

$0 \rightarrow 1$

$1 \rightarrow 0$

$$\text{Energy} \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2$$

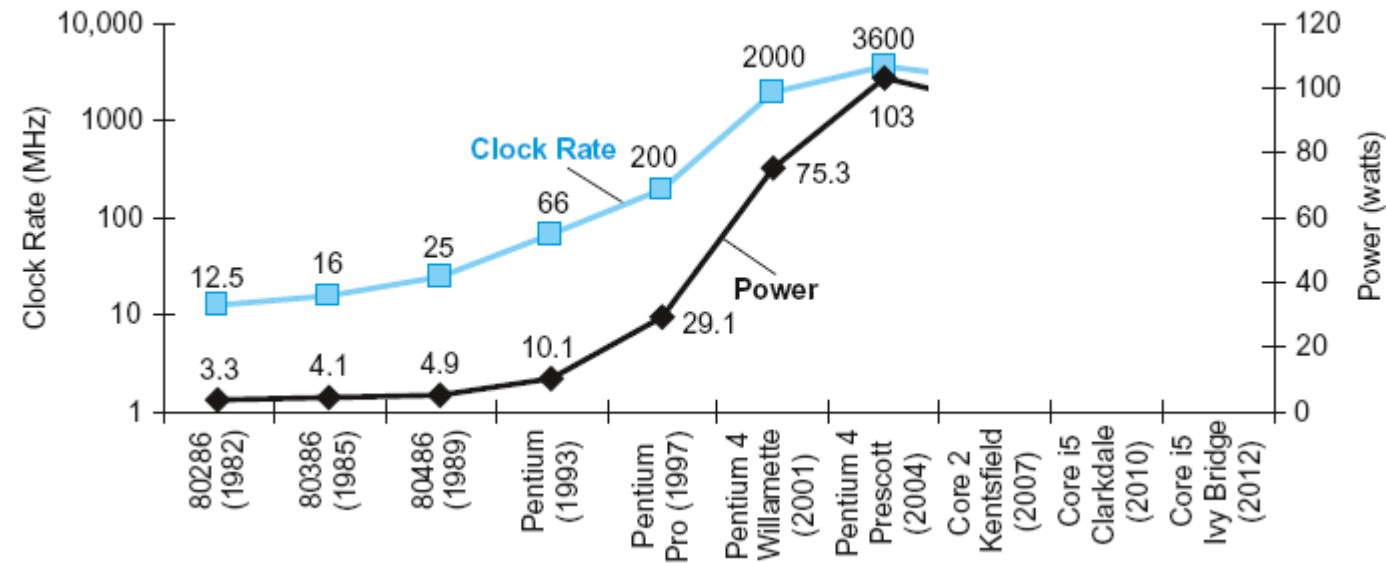
Power required per transistor

- Depends on number of transitions, frequency of the cpu

$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$$

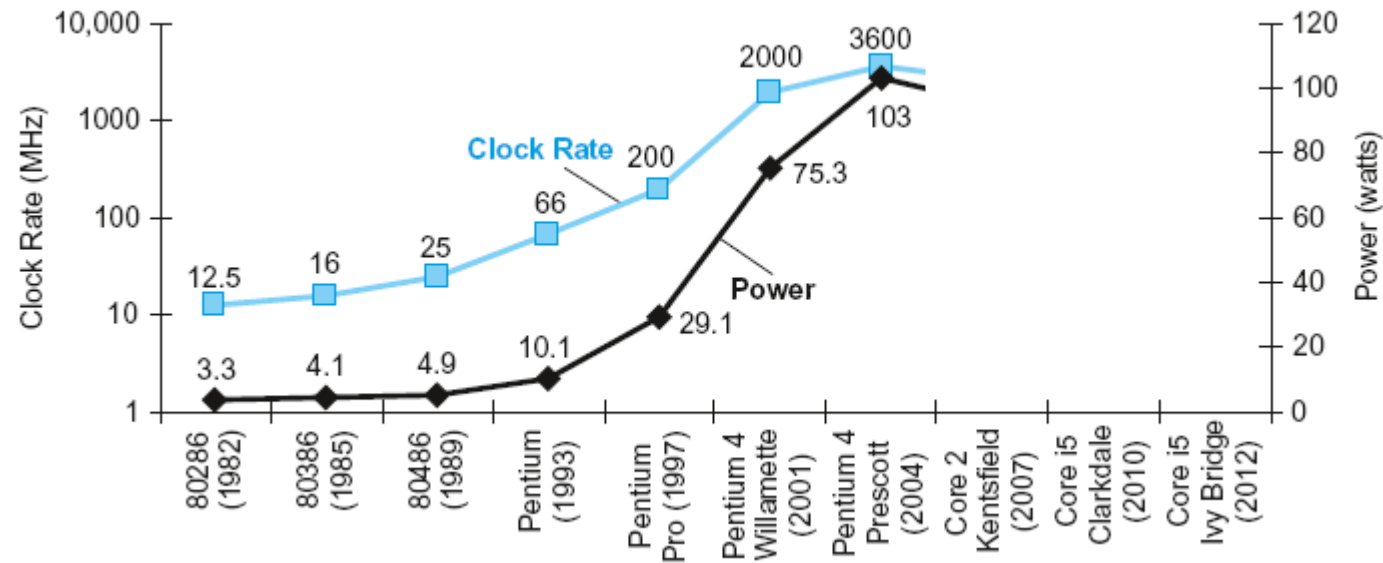
- Capacitive load depends on
 - Number of transistors connected to output
 - Technology (defines capacitance of wires, and transistors)

Power Trends



$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$

Power Trends

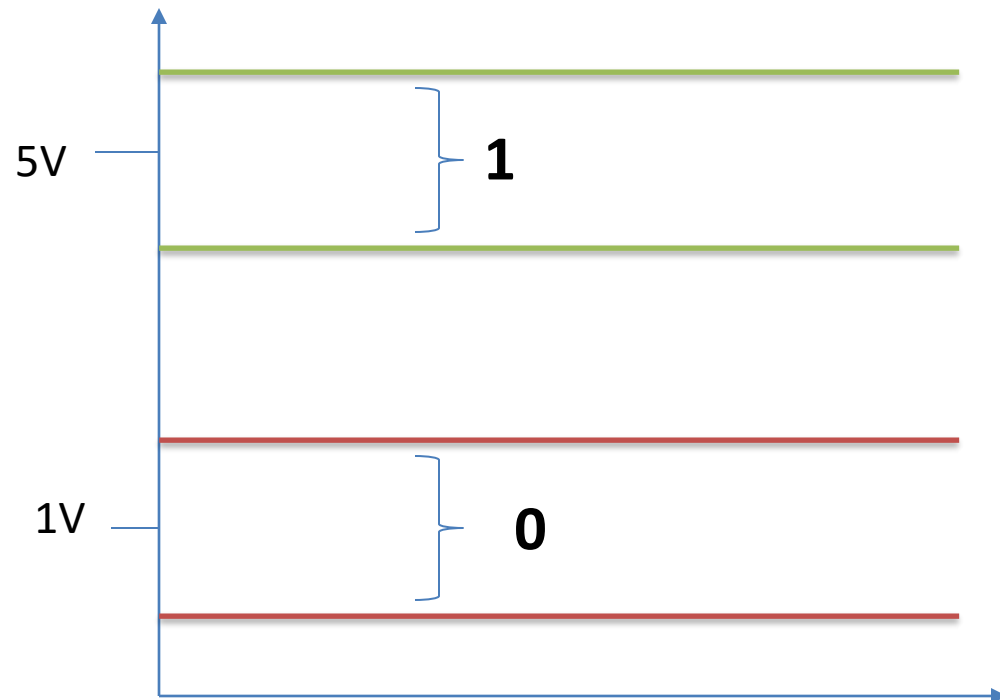


$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$$

×30

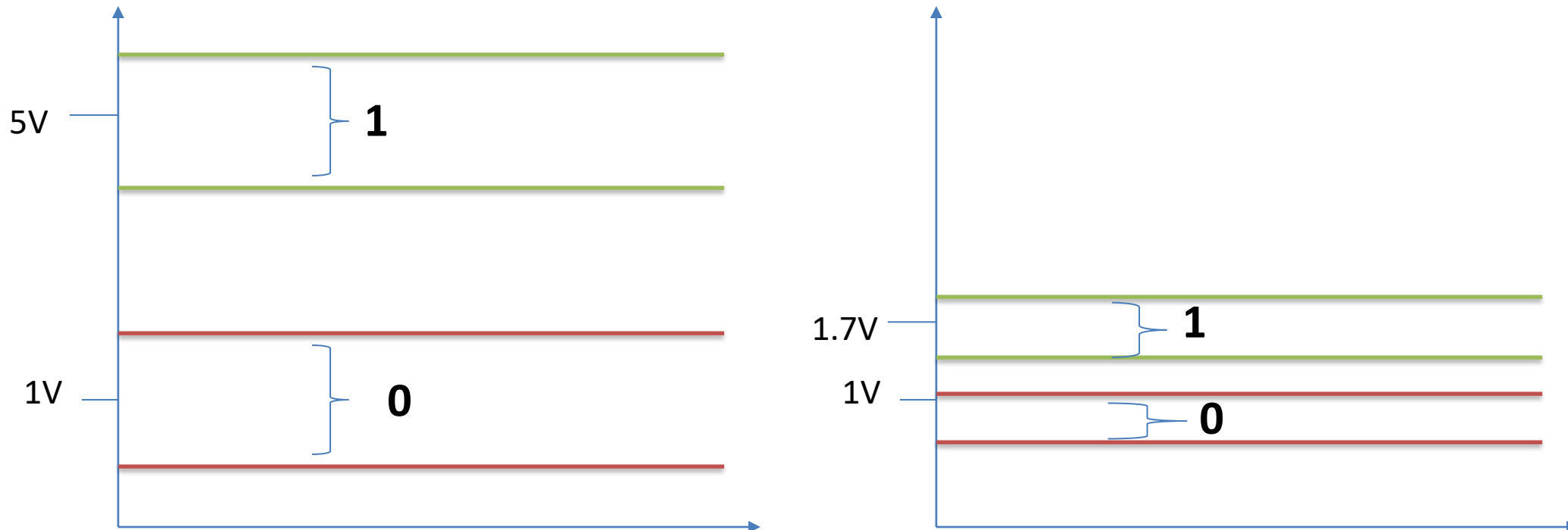
×1000

Reducing Voltage



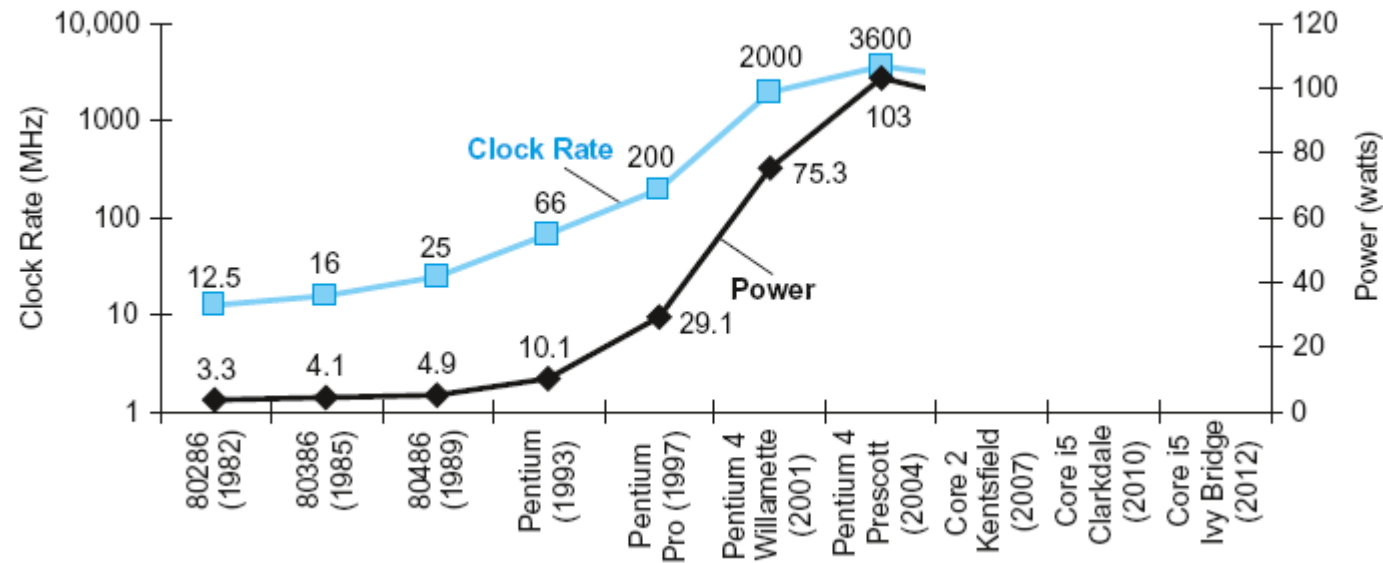
Reducing Voltage

Reduces overall power consumptions



Improvements in technology allowed for lesser fluctuations and lowering the voltage

Power Trends



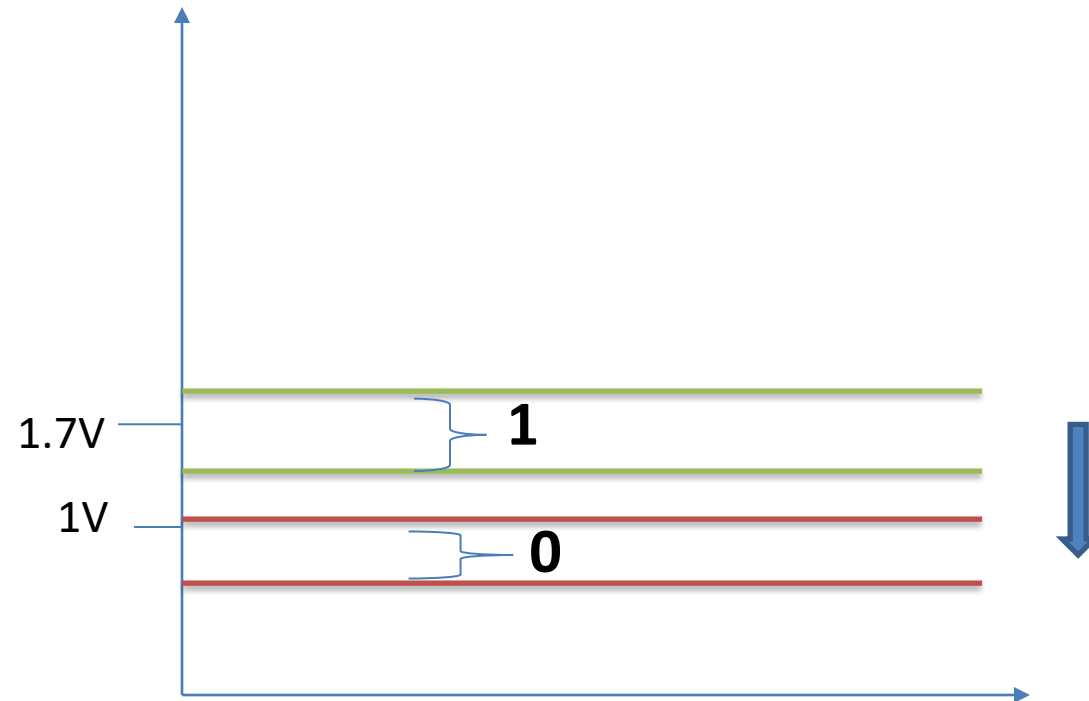
$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$$

×30

5V → 1V

×1000

Further Reducing Voltage → Leaky Transistors



Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it can adjust voltage so that it can reduce voltage 15% compared to processor B, which results in a 15% shrink in frequency. What is the impact on dynamic power?

Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it can adjust voltage so that it can reduce voltage 15% compared to processor B, which results in a 15% shrink in frequency. What is the impact on dynamic power?

$$Power \propto 0.5 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$$

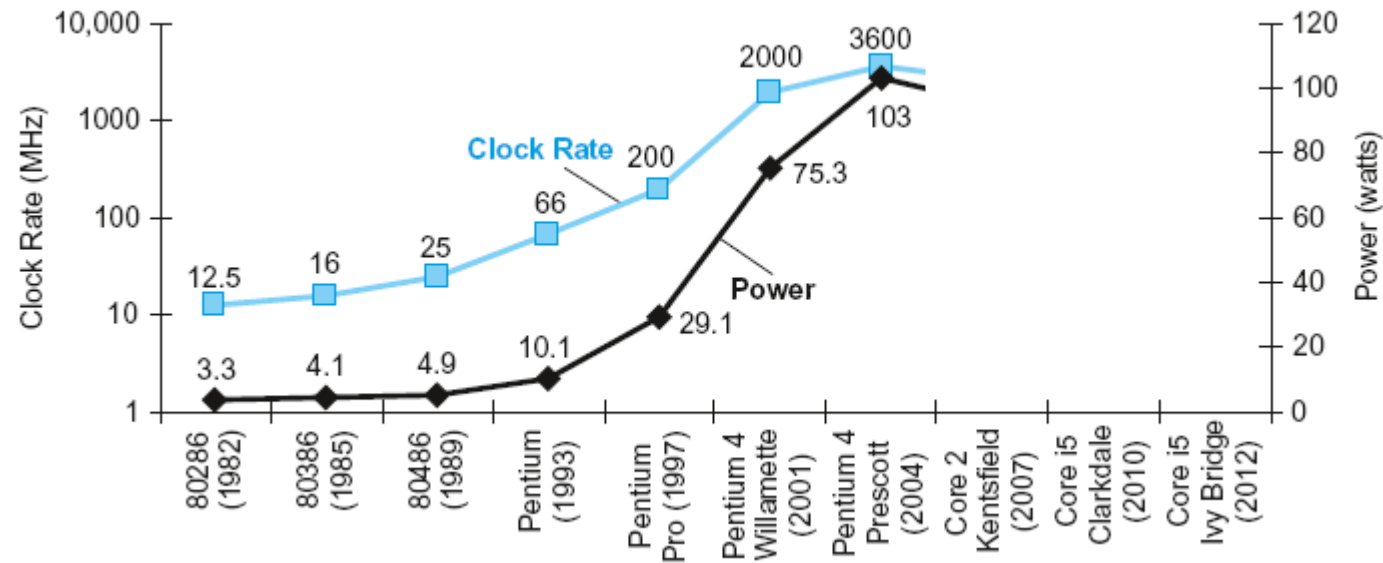
$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = \frac{\langle \text{Capacitive load} \times 0.85 \rangle \times \langle \text{Voltage} \times 0.85 \rangle^2 \times \langle \text{Frequency switched} \times 0.85 \rangle}{\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}}$$

Thus the power ratio is

$$0.85^4 = 0.52$$

Hence, the new processor uses about half the power of the old processor.

Power Wall

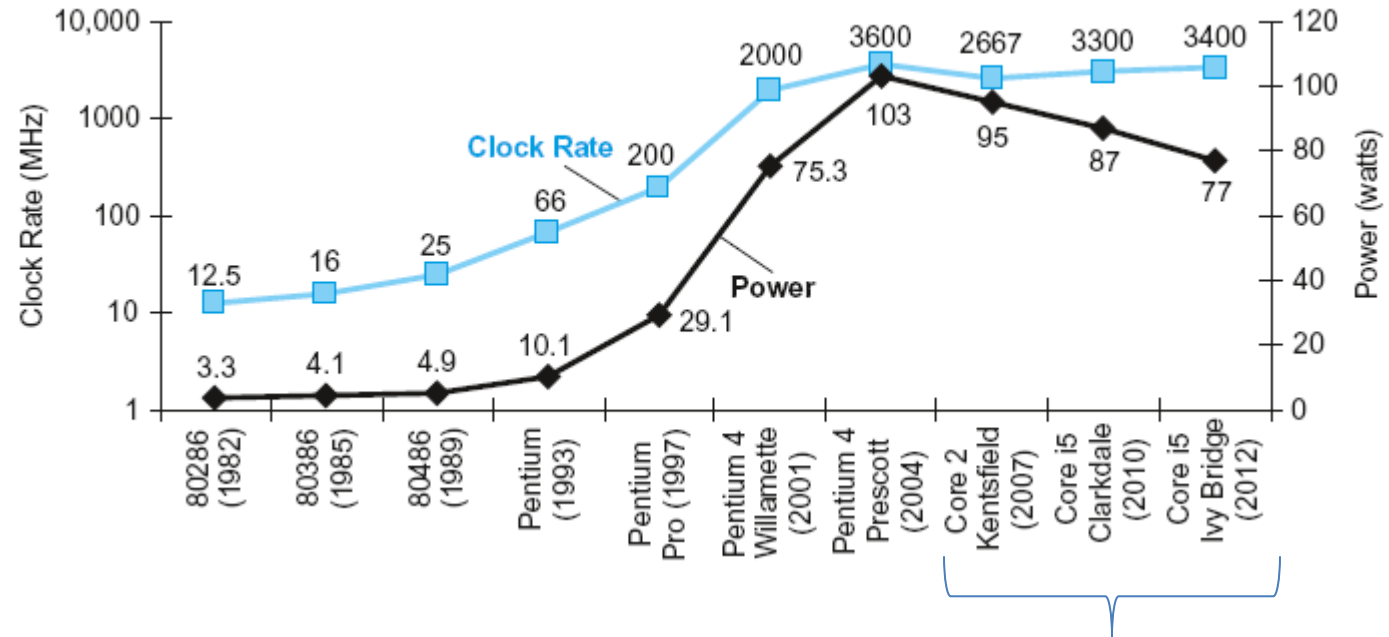


To address the power problem

1. Larger cooling devices.
2. Set CPU to idle when not in use.

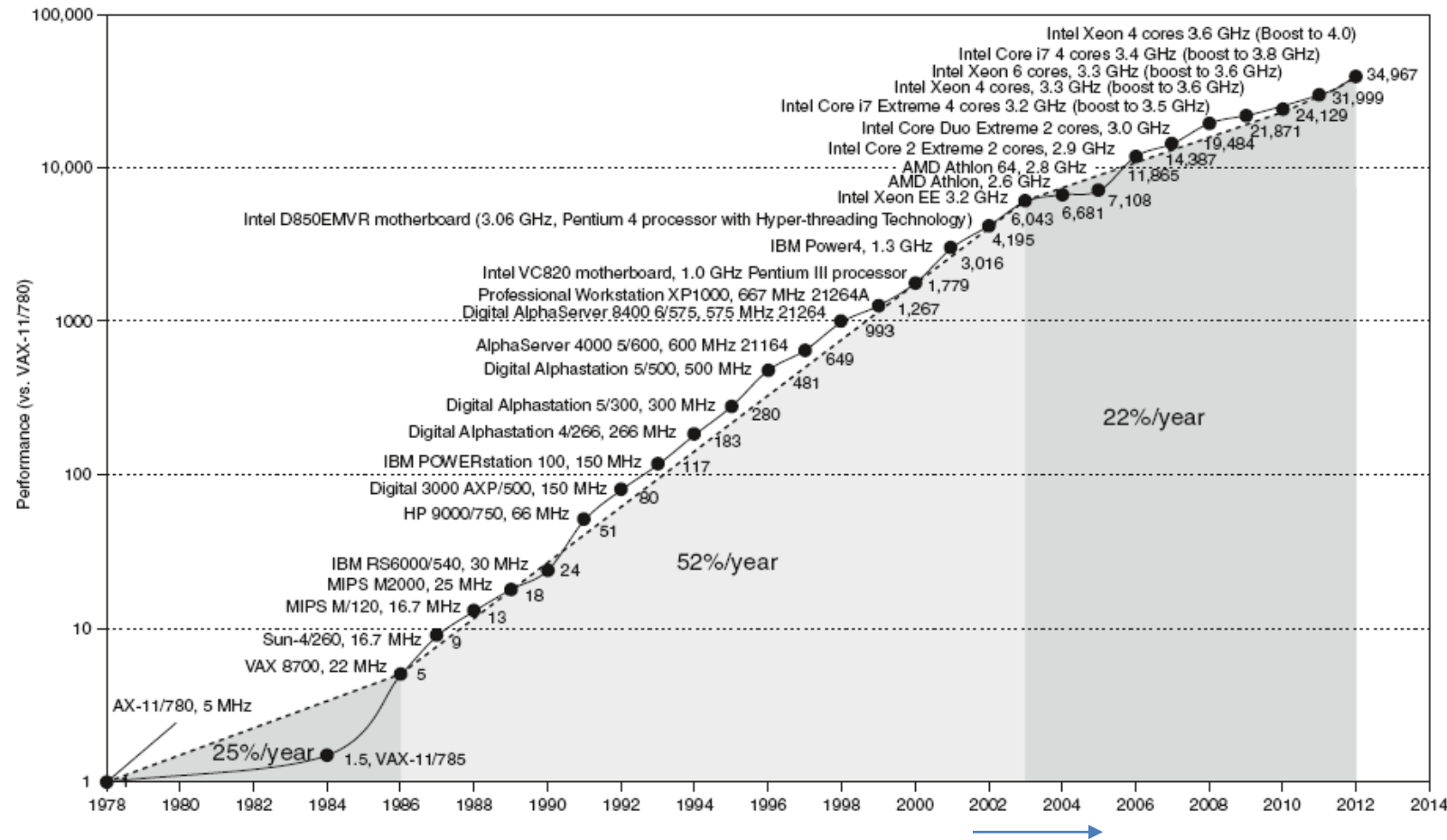
More advanced cooling mechanism can be expensive
Computer designers hit a **Power Wall**.

Power Wall



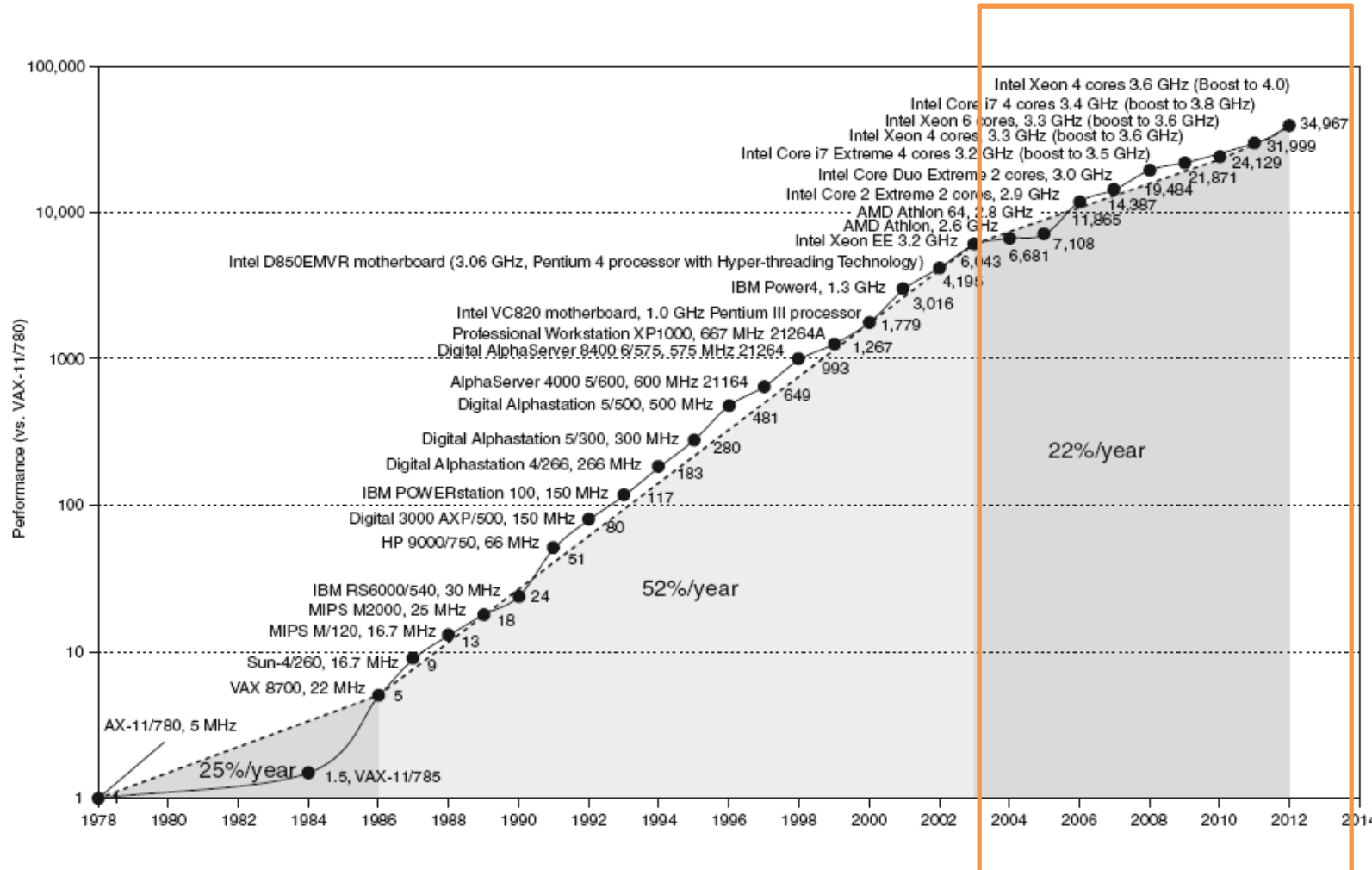
Multiple processors
per chip

Response time



Reduced from 1.5 per year to 1.2 per year

Response time



- Instead of reducing response time, focused on increasing throughput.
- 2006 and later computers shipped with multiple processes (per chip)

Multiprocessors

- Multicore microprocessors
 - More than one processor per chip
- Requires explicitly parallel programming
 - Compare with instruction level parallelism
 - Hardware executes multiple instructions at once
 - Hidden from the programmer
 - Hard to do
 - Programming for performance
 - Load balancing
 - Optimizing communication and synchronization
 - Scheduling, load balancing, synchronization, communication (overhead)

CPU Benchmarking

- A common reference to specify the performance of a processor
- Define a set of common programs (workload).
- Run on two candidate computers and compare execution time.

SPEC CPU Benchmark (Textbook)

- Programs used to measure performance
 - Supposedly typical of actual workload
- System Performance Evaluation Corp (SPEC)
 - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2006
 - Elapsed time to execute a selection of programs
 - Negligible I/O, so focuses on CPU performance

SPEC CPU Benchmark (**Updated**)

- Programs used to measure performance
 - Supposedly typical of actual workload
- **Standard** Performance Evaluation Corp (SPEC)
 - Develops benchmarks for CPU, I/O, Web, ...
- **SPEC CPU2017**
 - Elapsed time to execute a selection of programs
 - Negligible I/O, so focuses on CPU performance

SPEC CPU2017

SPEC CPU 2017 includes four suites that focus on different types of compute intensive performance:

Short Tag	Suite	Contents	Metrics	How many copies? What do Higher Scores Mean?
intspeed	SPECspeed® 2017 Integer	10 integer benchmarks	SPECspeed®2017_int_base SPECspeed®2017_int_peak SPECspeed®2017_int_energy_base SPECspeed®2017_int_energy_peak	SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.
fpspeed	SPECspeed®2017 Floating Point	10 floating point benchmarks	SPECspeed®2017_fp_base SPECspeed®2017_fp_peak SPECspeed®2017_fp_energy_base SPECspeed®2017_fp_energy_peak	
intrate	SPECrate® 2017 Integer	10 integer benchmarks	SPECrate®2017_int_base SPECrate®2017_int_peak SPECrate®2017_int_energy_base SPECrate®2017_int_energy_peak	SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more <i>throughput</i> (work per unit of time).
fprate	SPECrate® 2017 Floating Point	13 floating point benchmarks	SPECrate®2017_fp_base SPECrate®2017_fp_peak SPECrate®2017_fp_energy_base SPECrate®2017_fp_energy_peak	

The "Short Tag" is the canonical abbreviation for use with `runcpu`, where context is defined by the tools. In a published document, context may not be clear. To avoid ambiguity in published documents, the Suite Name or the Metrics should be spelled as shown above.

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1



Reference computer: Sun Microsystems server, the Sun Fire V490 with 2100 MHz UltraSPARC-IV+ chips

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1

$$SPECratio = \frac{Refernce\ Time}{Execution\ Tlme}$$

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 ⁹	CPI	Clock cycle time (seconds x 10 ⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Performance Summary

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}_i}$$

SPEC CPU Benchmark

- Programs used to measure performance
 - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
 - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2017
 - Elapsed time to execute a selection of programs
 - Negligible I/O, so focuses on CPU performance
 - Normalize relative to reference machine
 - Summarize as geometric mean of performance ratios
 - **CINT2006 (integer) and CFP2006 (floating-point) summary**

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}_i}$$

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

Pitfall: Amdahl's Law

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example:
- Takes a total of 100s
- multiply accounts for 80s (of the 100s)
 - How much improvement in multiply performance to get 5x overall?

Pitfall: Amdahl's Law

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example:
- Takes a total of 100s
- multiply accounts for 80s (of the 100s)
 - How much improvement in multiply performance to get 5x overall?

$$(5 \text{ times faster}) 100/5 = 20 = \frac{80}{n} + 20 \quad \text{■ Can't be done!}$$

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
 - Doesn't account for
 - Differences in ISAs between computers
 - Differences in complexity between instructions

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

- CPI varies between programs on a given CPU

