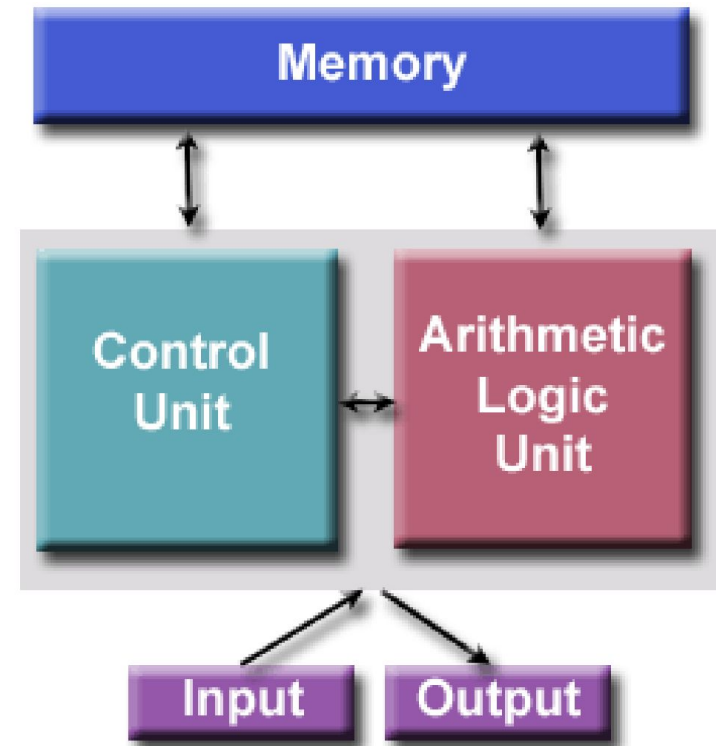


Von Neumann architecture

- The Von Neumann architecture consists of a single, shared **memory** for programs and data, a single bus for **memory** access, an arithmetic unit, and a program control unit. The Von Neumann processor operates fetching and execution cycles serially.
- **Disadvantage** of Von Neumann architecture: shared memory for instructions and data with one data bus and one address bus between processor and memory. Instructions and data have to be fetched in sequential order (known as the Von Neumann Bottleneck), limiting the operation bandwidth.



Great Ideas in Computer Architecture

Application of the following great ideas has accounted for much of the tremendous growth in computing capabilities over the past 50 years.

- Design for Moore's law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

Design for Moore's Law

Gordon Moore, one of the founders of Intel made a prediction in 1965 that integrated circuit resources would double every 18–24 months.

This prediction has held approximately true for the past 50 years. It is now known as Moore's Law.

Moore's Law

1965; Gordon Moore – co-founder of Intel

Observed number of transistors that could be put on a single chip was doubling every year

Consequences of Moore's law:

The pace slowed to a doubling every 18 months in the 1970's but has sustained that rate ever since

The cost of computer logic and memory circuitry has fallen at a dramatic rate

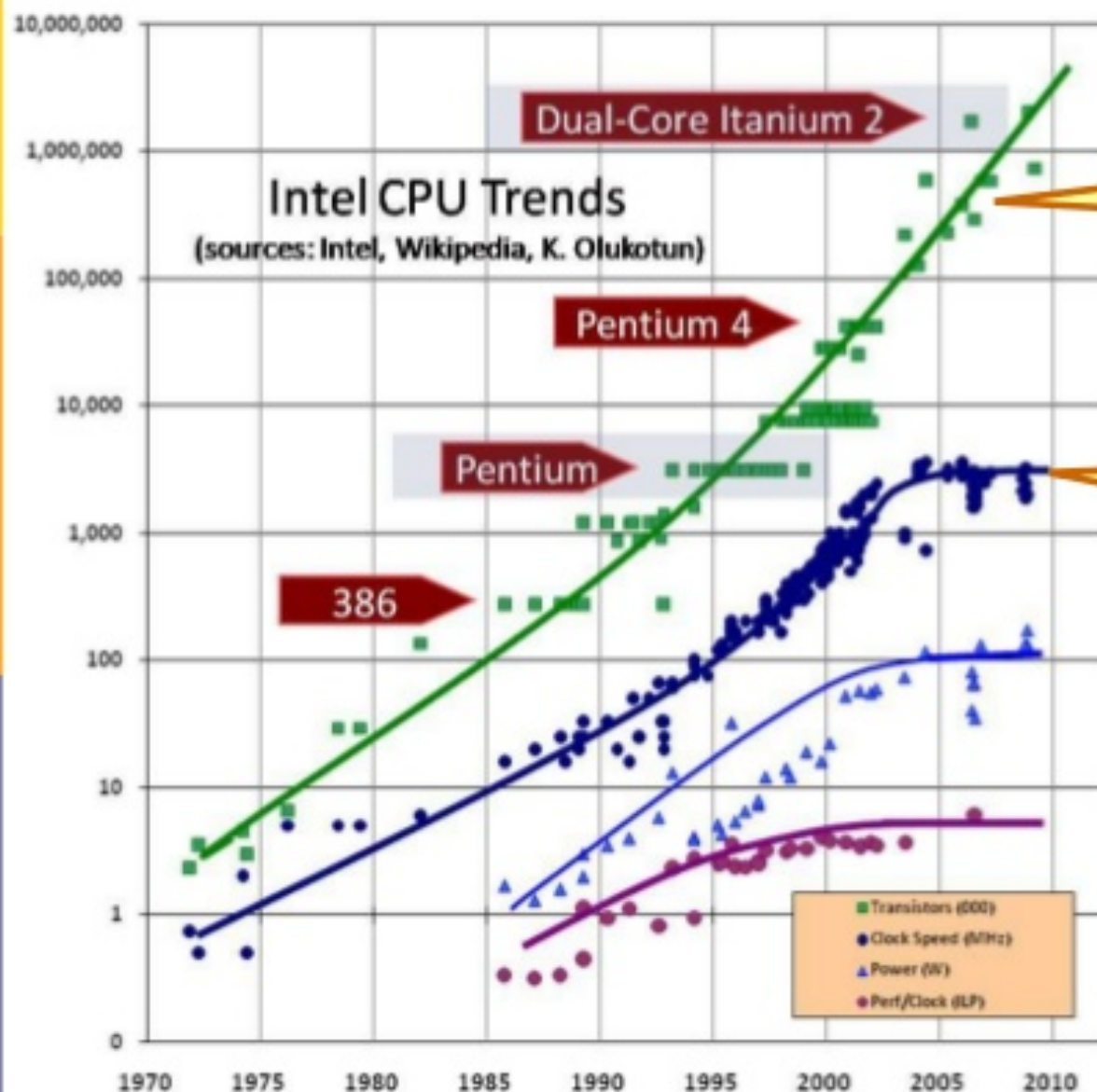
The electrical path length is shortened, increasing operating speed

Computer becomes smaller and is more convenient to use in a variety of environments

Reduction in power and cooling requirements

Fewer interchip connections

Moore's Law – Today's Status



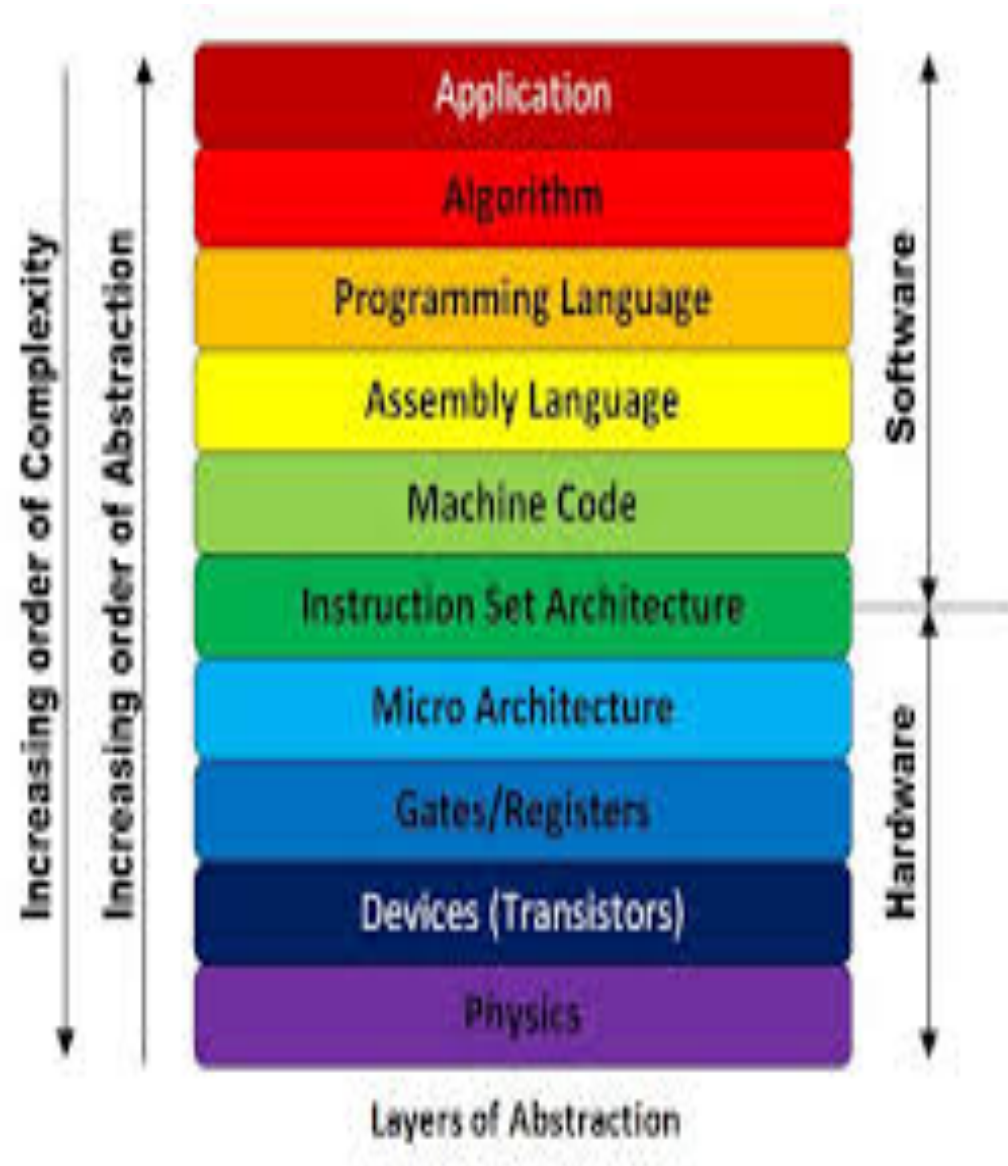
Transistor
count still
rising

Clock speed
flattening
sharply

Moore's Law – No of
transistors on a chip
tends to double about
every 2 years

Use Abstraction to Simplify Design

- Abstraction uses multiple levels with each level hiding the details of levels below it. For example:
- The instruction set of a processor hides the details of the activities involved in executing an instruction.
- High-level languages hide the details of the sequence of instructions need to accomplish a task.
- Operating systems hide the details involved in handling input and output devices.

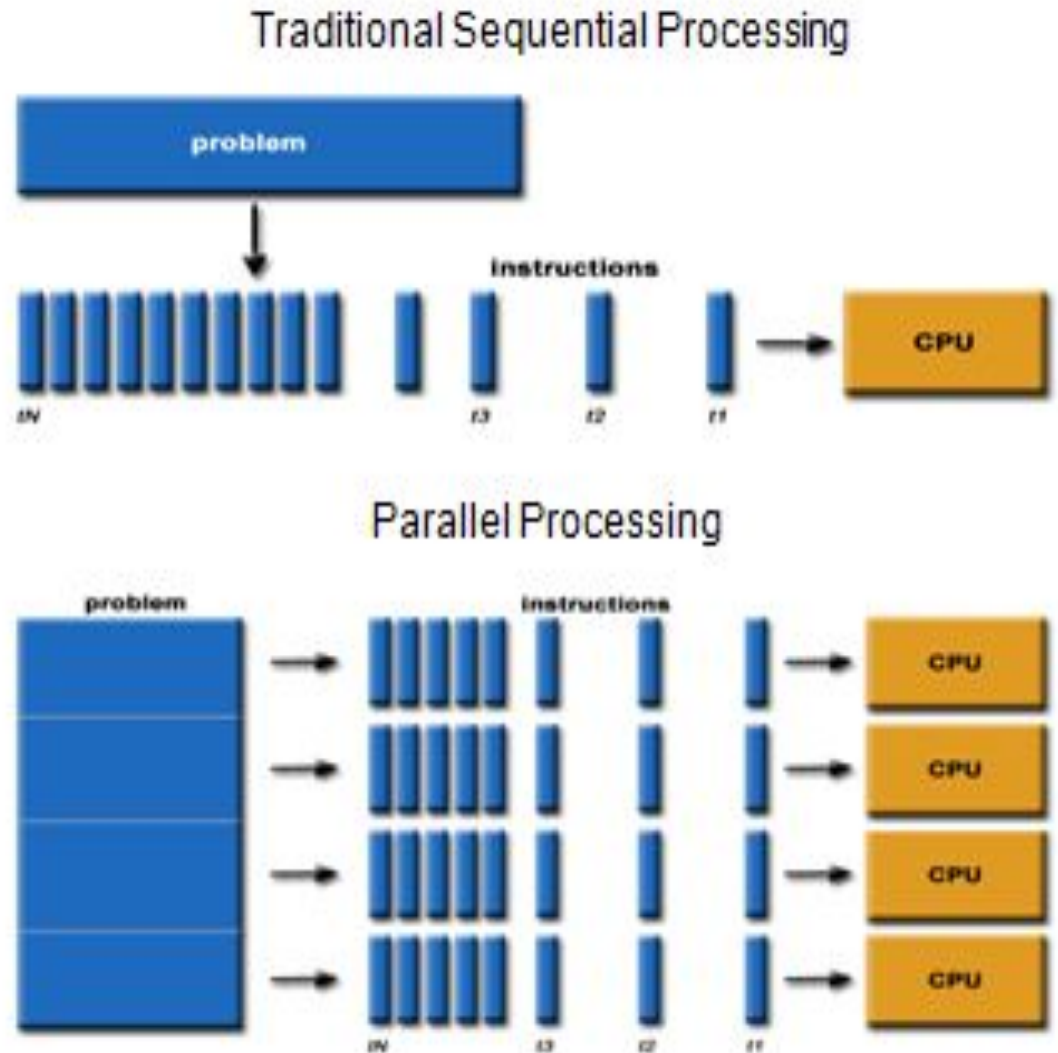


Make the Common Case Fast

- The most significant improvements in computer performance come from improvements to the common case — areas where the current design is spending the most time.
- This idea is sometimes called Amdahl's law, though it is preferable to use that term to refer to a mathematical law for analyzing improvements. The mathematical law also is closely related to the law of diminishing returns.

Doing different parts of a task in parallel accomplishes the task in less time than doing them sequentially. A processor engages in several activities in the execution of an instruction. It runs faster if it can do these activities in parallel.

Performance via Parallelism



Performance via Pipelining

This idea is an extension of the idea of parallelism. It is essentially handling the activities involved in instruction execution as an assembly line. As soon as the first activity of an instruction is done you move it to the second activity and start the first activity of a new instruction. This results in executing more instructions per unit time compared to waiting for all activities of the first instruction to complete before starting the second instruction.

Performance via Prediction

- A conditional branch is a type of instruction that determines the next instruction to be executed based on a condition test. Conditional branches are essential for implementing high-level language if statements and loops.
- Unfortunately, conditional branches interfere with the smooth operation of a pipeline — the processor does not know where to fetch the next instruction until after the condition has been tested.
- Many modern processors reduce the impact of branches with speculative execution: make an informed guess about the outcome of the condition test and start executing the indicated instruction. Performance is improved if the guesses are reasonably accurate and the penalty of wrong guesses is not too severe.

Hierarchy of Memories

- The principle of locality states that memory that has been accessed recently is likely to be accessed again in the near future. That is, accessing recently accessed data is a common case for memory accesses. To make this common case faster you need a cache — a small high-speed memory designed to hold recently accessed data.
- Modern processors use as many as 3 levels of caches. This is motivated by the large difference in speed between processors and memory.

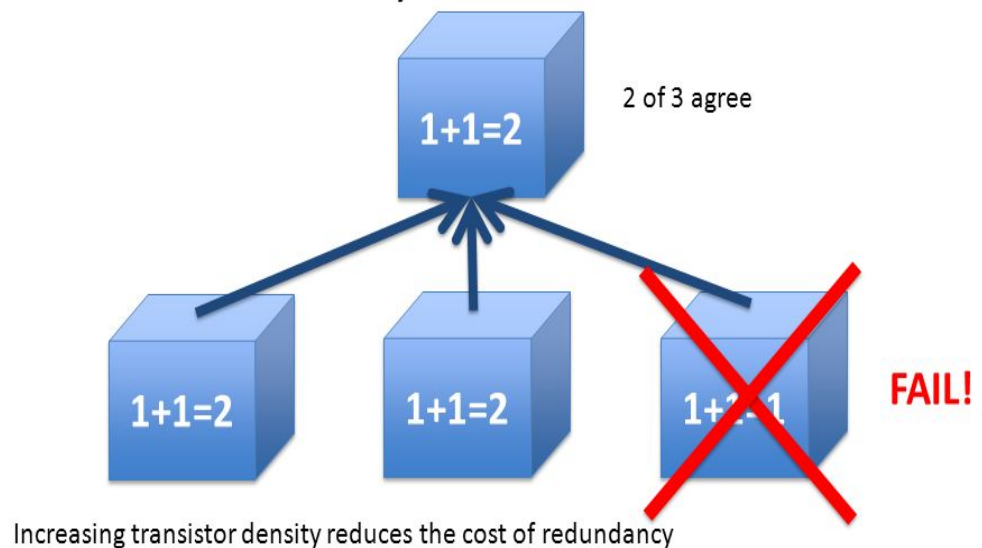
Dependability via Redundancy

One of the most important ideas in data storage is the Redundant Array of Inexpensive Disks (RAID) concept. In most versions of RAID, data is stored redundantly on multiple disks. The redundancy insures that if one disk fails the data can be recovered from other disks.

Great Idea #6:

Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail

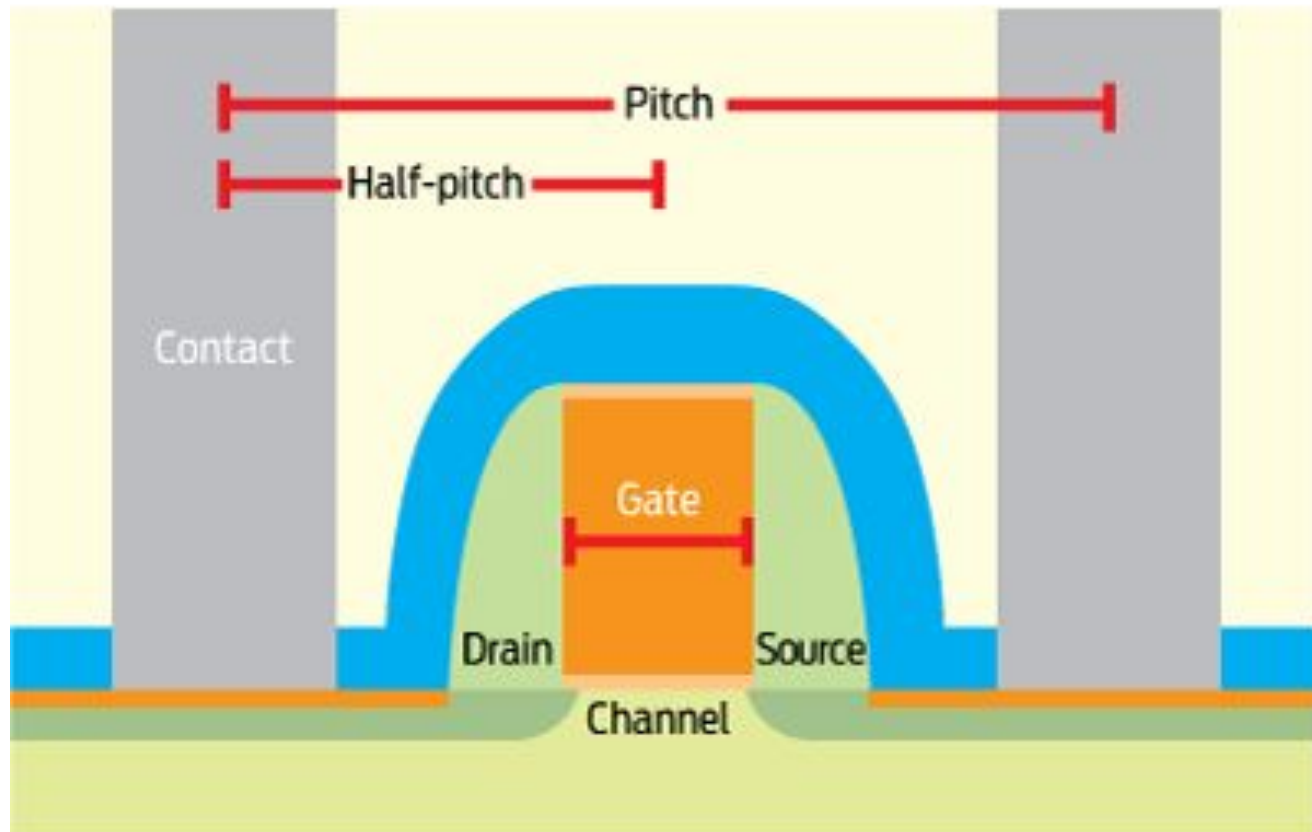


Nanometre

- A nanometer is a unit of measurement used to measure length. One nanometer is one billionth of a meter, so nanometers are certainly not used to measure long distances. Instead, they serve to measure extremely small objects, such as atomic structures or [transistors](#). A nanometer is a unit of measurement used to measure length. One nanometer is one billionth of a meter, so nanometers are certainly not used to measure long distances. Instead, they serve to measure extremely small objects, such as atomic structures or transistors found in modern [CPUs](#).
- A single nanometer is one million times smaller than a millimeter. If you take one thousandth of a millimeter, you have one micrometer, or a single [micron](#). If you divide that micron by 1,000, you have a nanometer. Needless to say, a nanometer is extremely small.
- Since [integrated circuits](#) Since integrated circuits, such as computer [processors](#), contain microscopic components, nanometers are useful for measuring their size. In fact, different eras of processors are defined in nanometers, in which the number defines the distance between transistors and other components within the CPU. The smaller the number, the more transistors that can be placed within the same area, allowing for faster, more efficient processor designs.
- Intel's processor line, for example, has included chips based on the

Nanometer

- The nanometer number on memory chips generally refers to the smallest "half-pitch" between identical features on the chip



Understanding Performance

- Algorithm
 - Determines number of operations executed
- Programming language, compiler, architecture
 - Determine number of machine instructions executed per operation
- Processor and memory system
 - Determine how fast instructions are executed
- I/O system (including OS)
 - Determines how fast I/O operations are executed

Performance of Computer Systems

Many different factors to take into account when determining performance:

- Technology
 - circuit speed (clock, MHz)
 - processor technology (how many transistors on a chip)
- Organization
 - type of processor (ILP)
 - configuration of the memory hierarchy
 - type of I/O devices
 - number of processors in the system
- Software
 - quality of the compilers
 - organization & quality of OS, databases, etc.

Metrics that Measure Performance

Execution time: time to execute one program from beginning to end

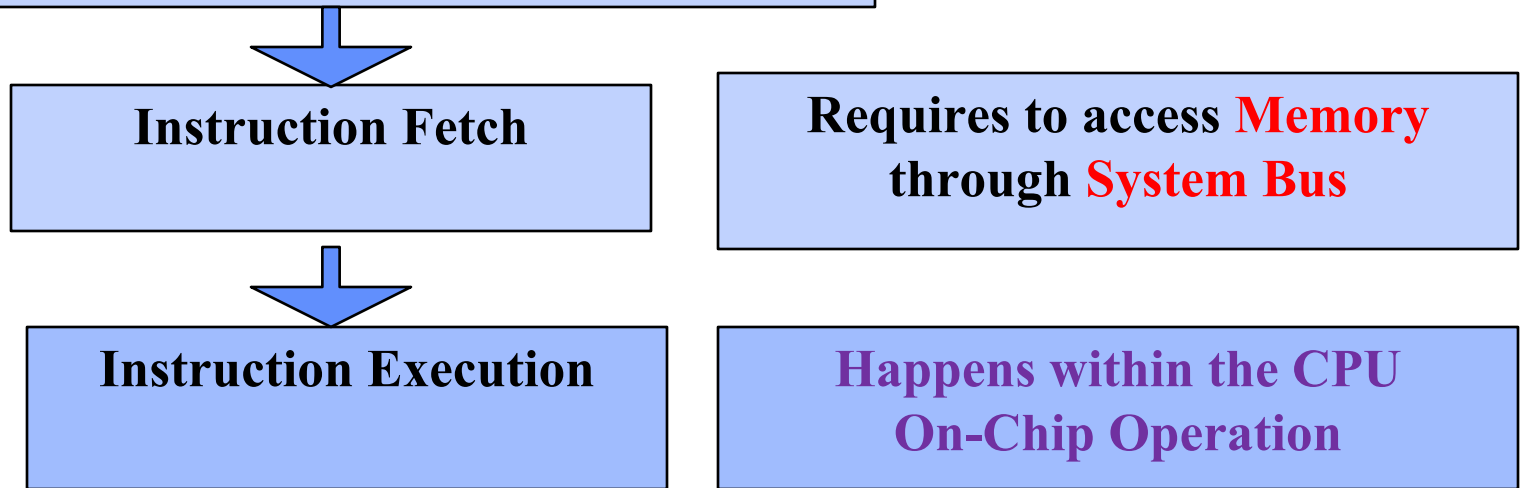
Performance = $1 / (\text{Execution Time})$

Throughput: total amount of work completed in a given time

- transactions (database) or packets (web servers) / second
- an indication of how well hardware resources are being used
- good metrics for chip designers or managers of computer systems

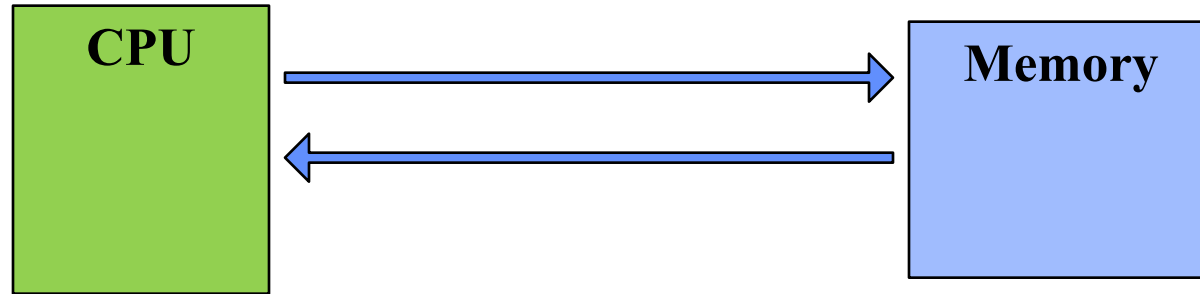
CPU Speed Vs Computer Performance

Processing of a single instruction



Instruction fetch: Read machine code of Instruction from **Memory** and store in a register within CPU/Microprocessor

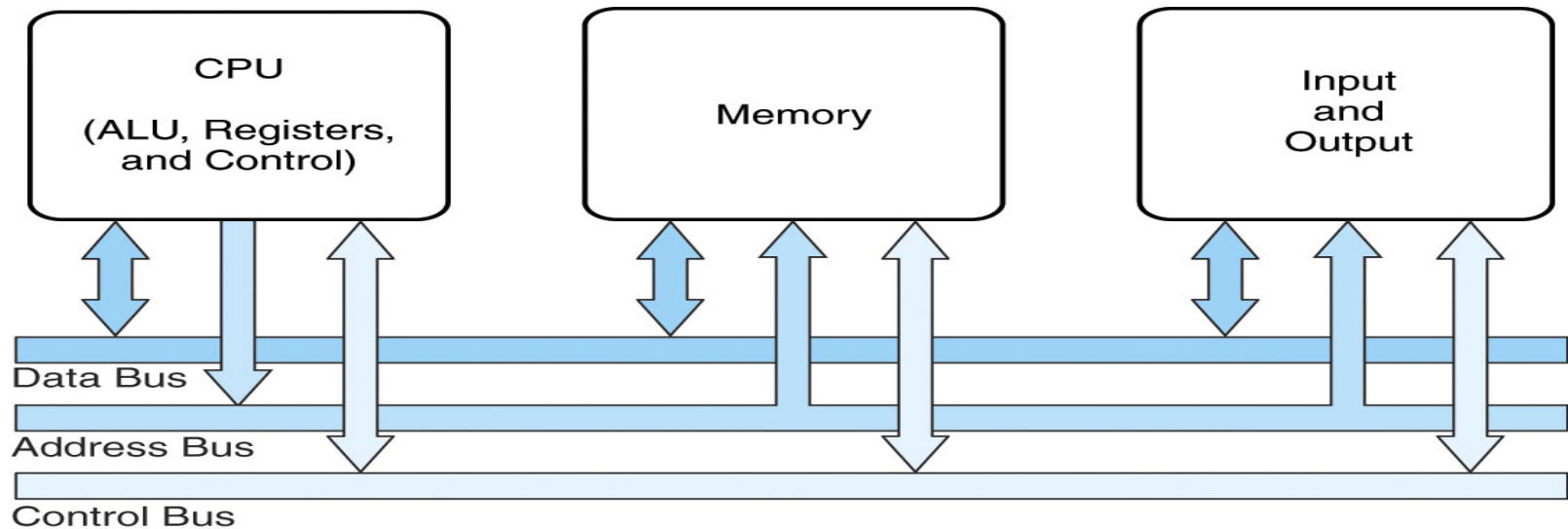
Instruction Execution: Decode instructions and perform ALU/data transfer operations etc.



Both **BUS speed** and **speed of memory device** are much **slower** than the speed of CPU/Microprocessor.

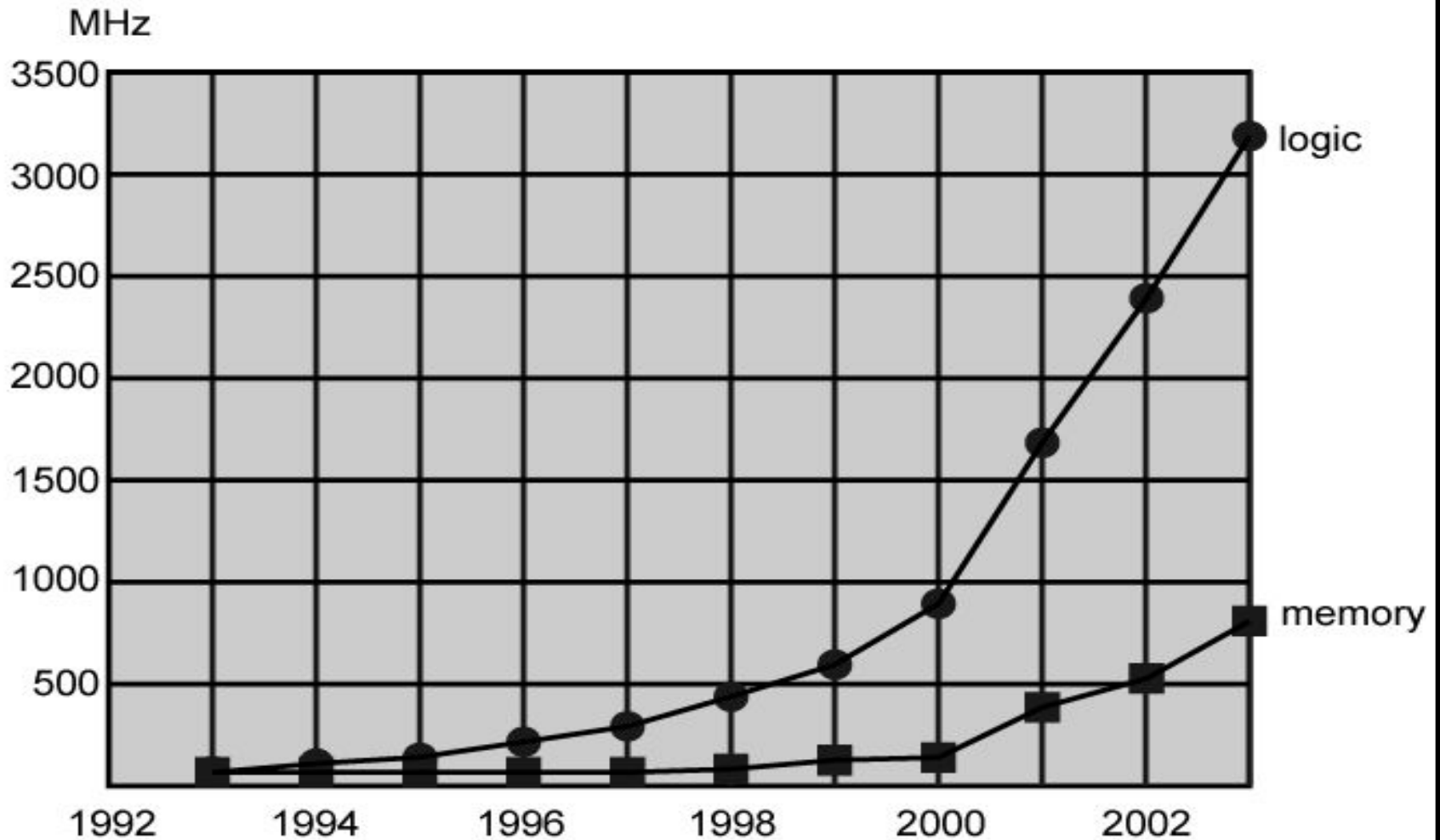
During Instruction Fetch, the CPU/Microprocessor has to **WAIT** for the Machine code to be transferred from memory to CPU through system bus.

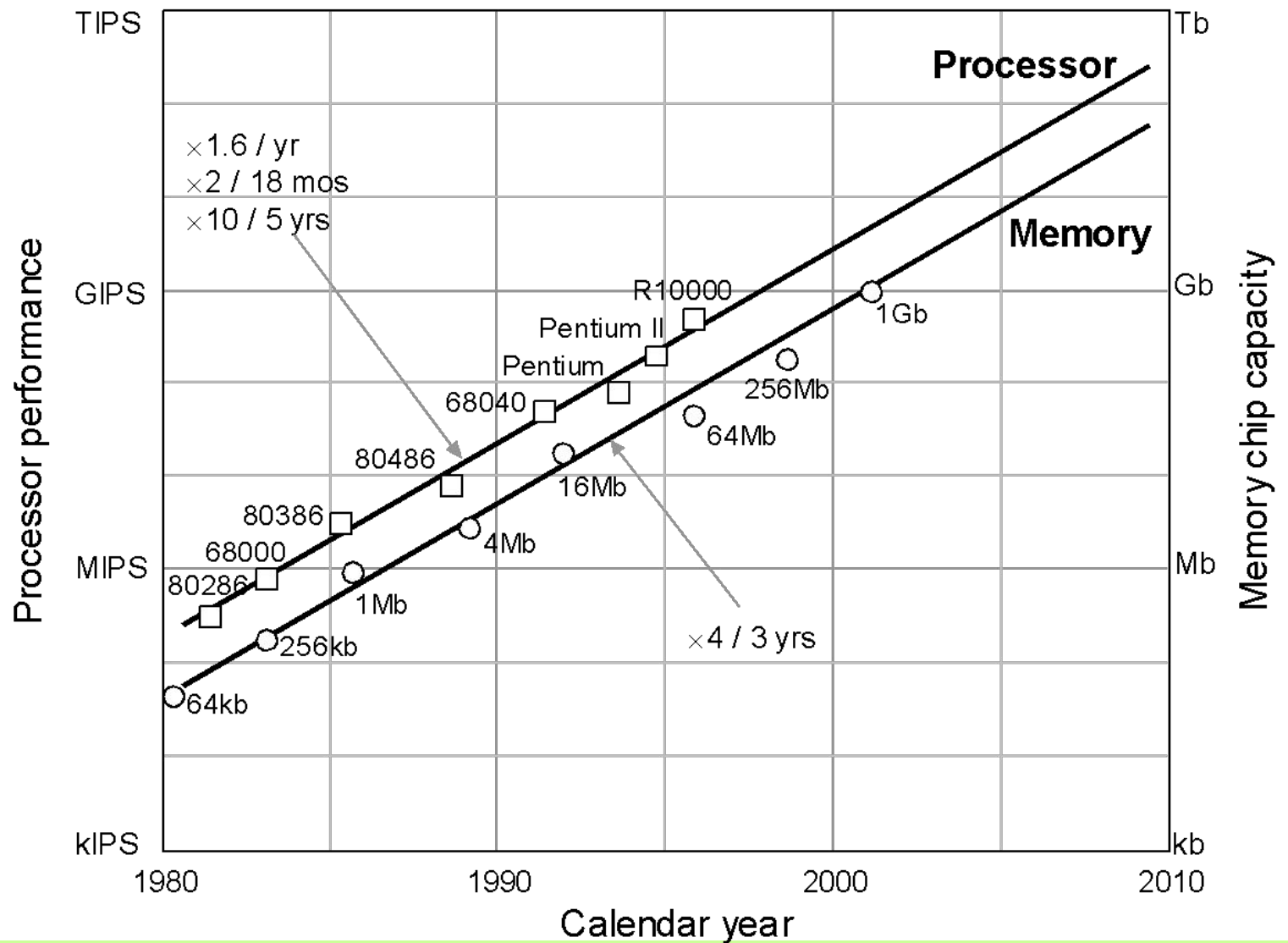
Instruction Execution: mostly **ON-CHIP** operation, requires much less time than Instruction fetch.



- The interface between processor and main memory is the most crucial pathway in the entire computer because it is responsible for carrying a constant flow of program instructions and data between memory chips and the processor.
- If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state, and valuable **processing time is lost**.

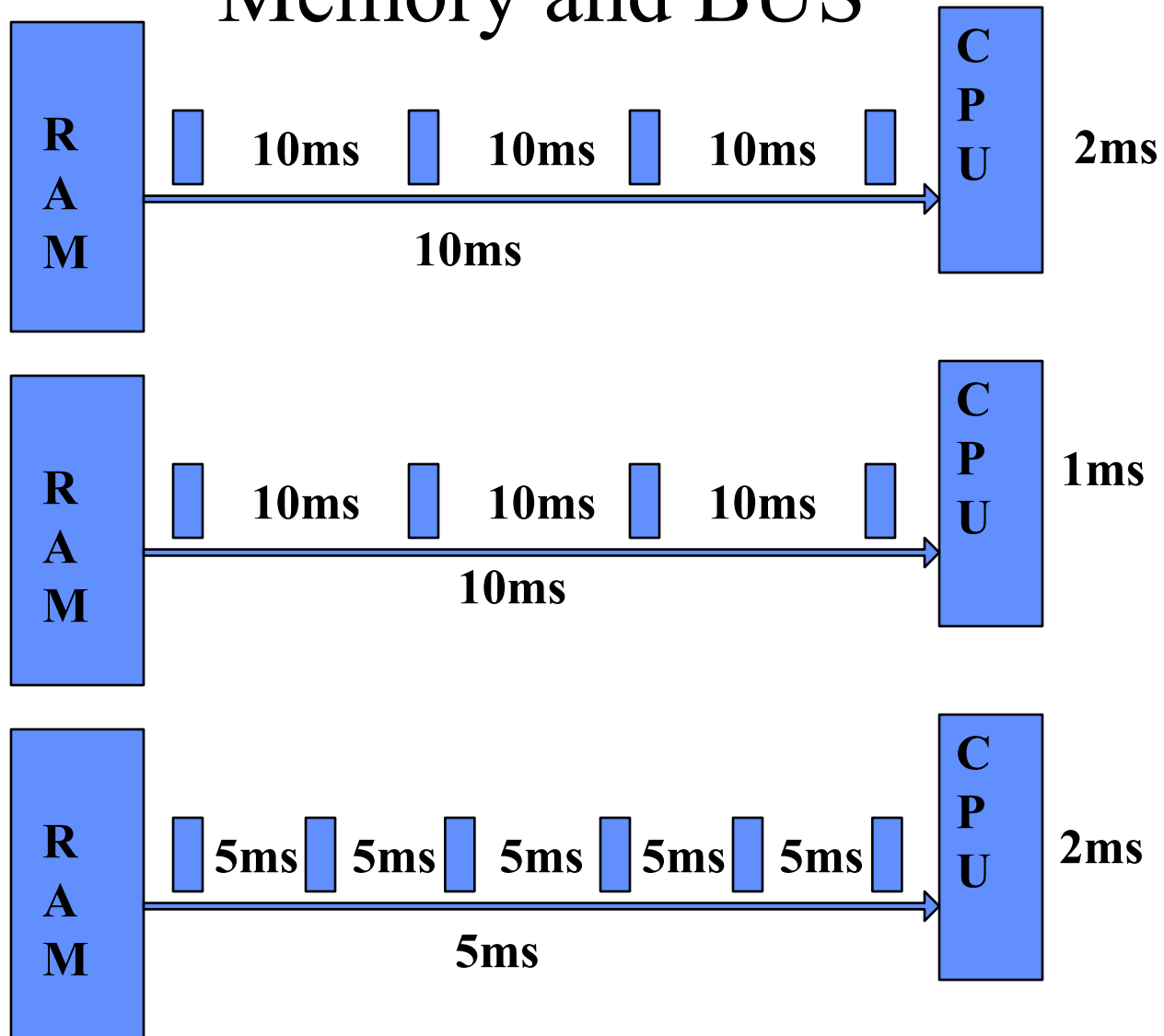
- Processor speed increased
- Memory capacity increased
- **Memory speed lags behind processor speed**



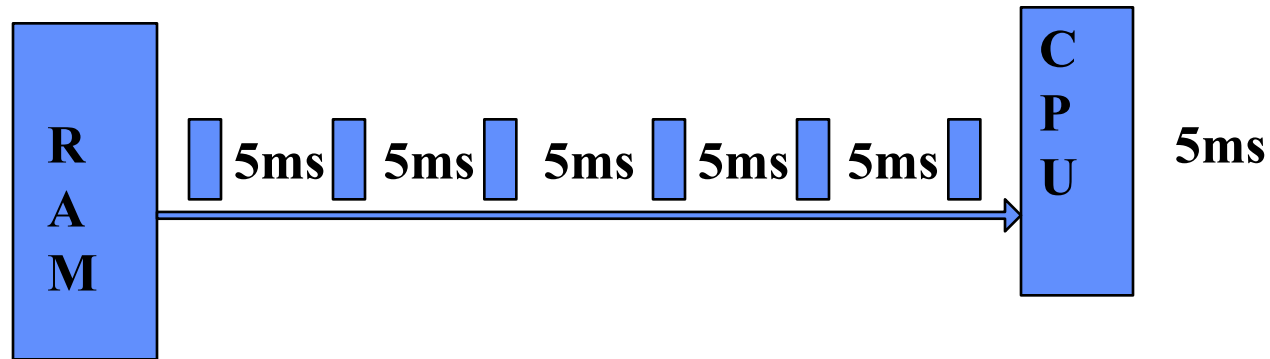


Trends in processor performance and RAM memory chip capacity (Moore's law).

Some models: Processor speed vs Memory and BUS



Processor Speed or Utilization?



The raw speed of the microprocessor will not achieve its potential unless it is fed a constant stream of work to do in the form of computer instructions.

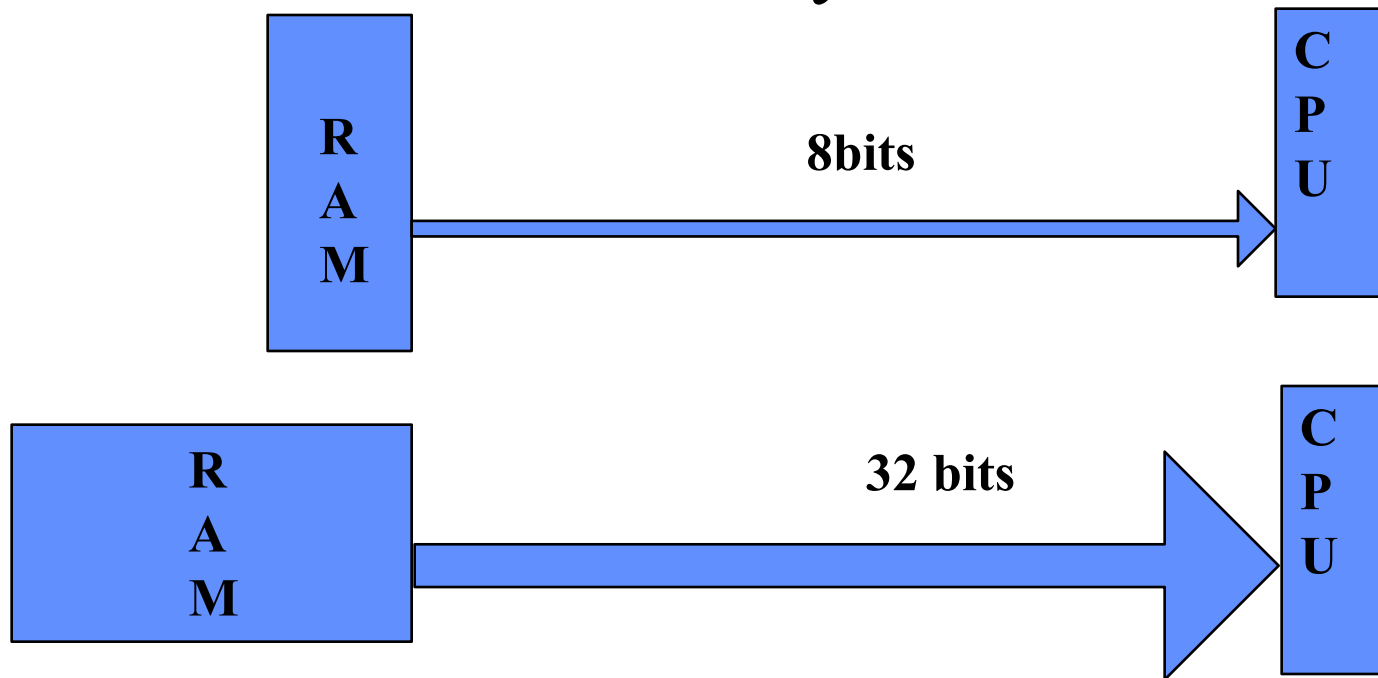
Solution: Either Memory and BUS speed should be same as Processor speed...consequence...**EXPENSIVE!**

Techniques for maintaining a continuous stream of Instructions to the Microprocessor/CPU.

Solution: Improved architecture

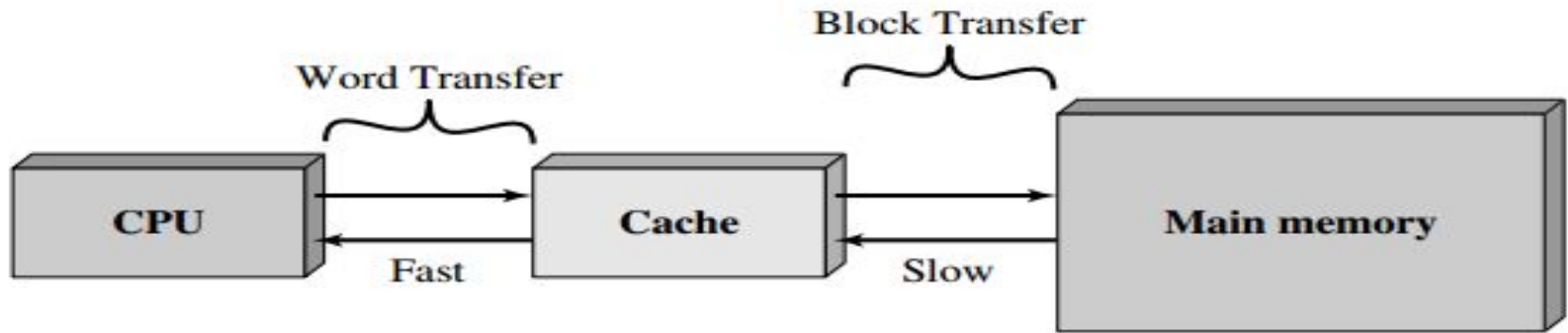
- Making RAM wider
- Making data bus wider

To allow processor to read more data and instruction in one bus cycle

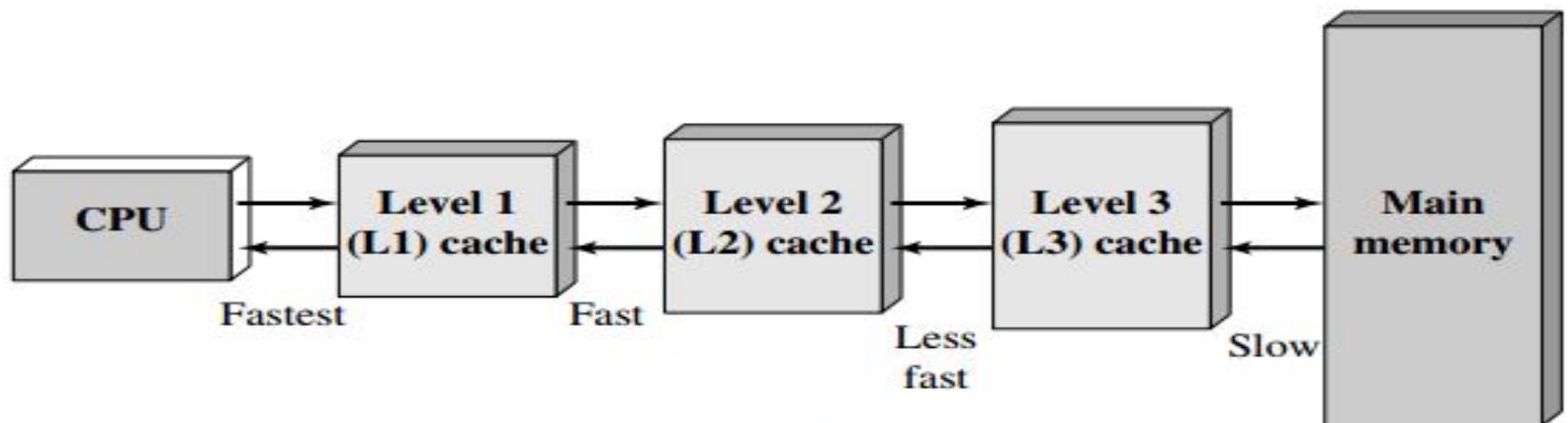


Solution: Cache memory

Reduce the frequency of memory access by incorporating cache memory.



(a) Single cache



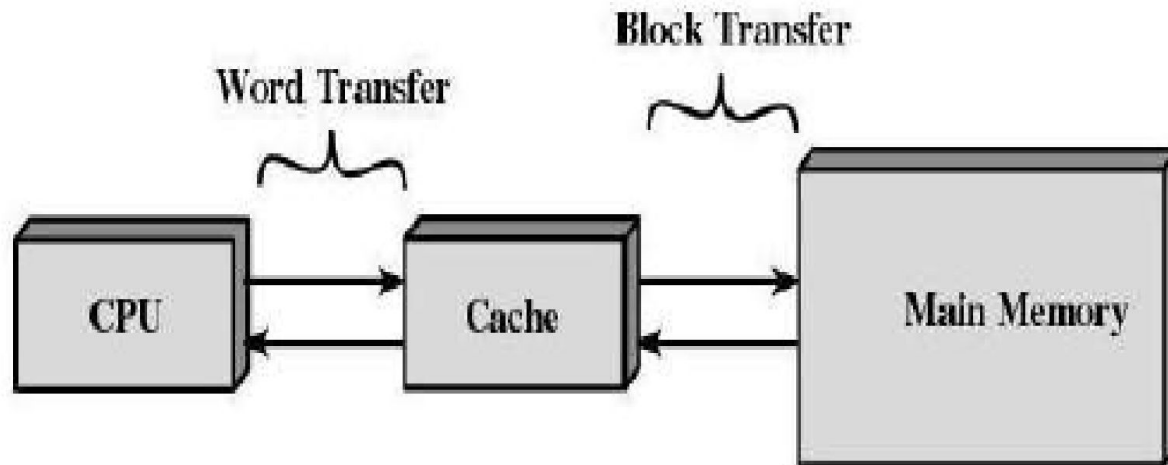
(b) Three-level cache organization

Example

- A program contains 1000 Instructions (machine) each of 1 byte length. Processor-A does not contain any cache memory and it can read 1 byte instruction in a single read operation.
- How many memory access would require to run the program? **Answer.....?**

Example

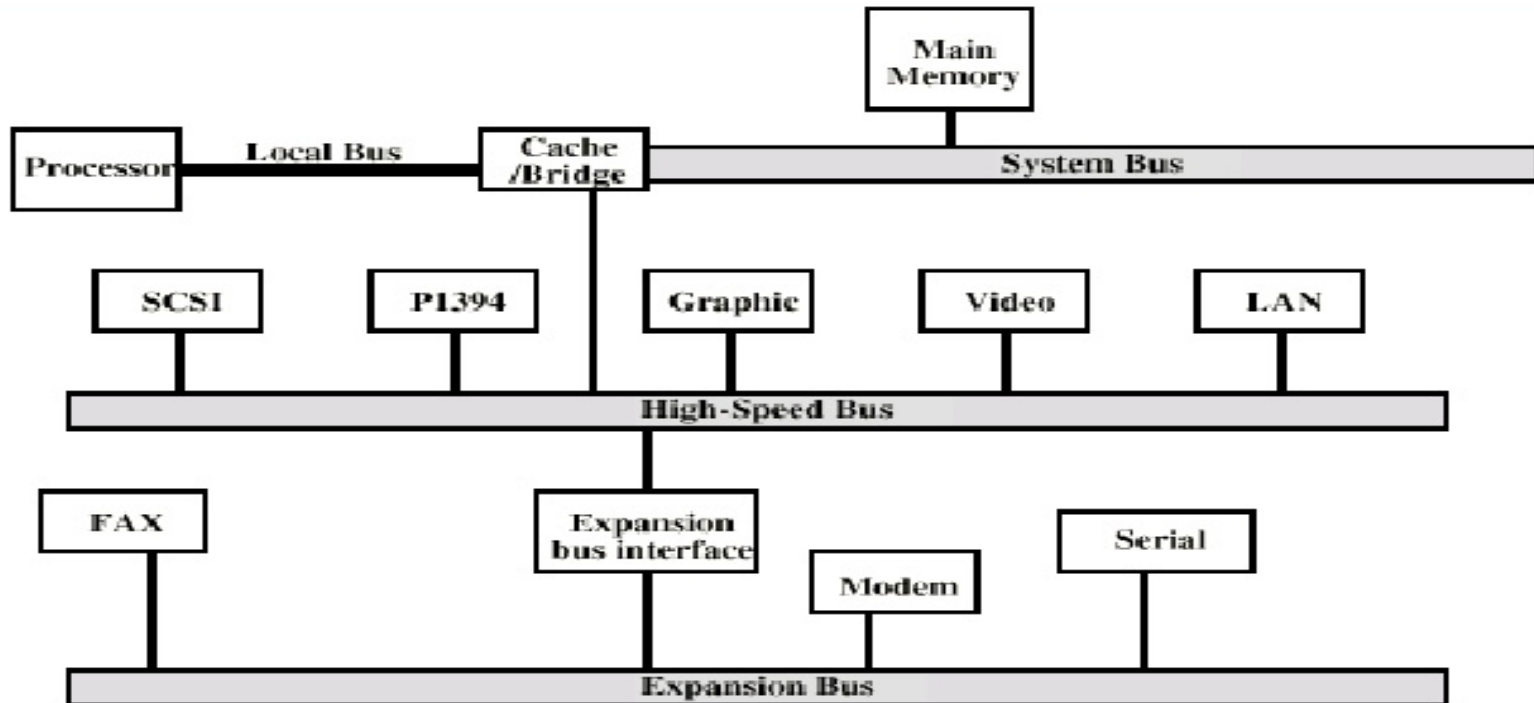
- A program contains 1000 Instructions (machine) each of 1 byte length. Level-1 cache is used as shown below with a hit ratio of 90%.
- How many memory access would require to run the program? **Answer.....?**



Solution: High-speed buses

Increase the interconnect bandwidth between processors and memory.

High Performance Bus



List the approaches to achieving increased processor speed?

- **Increase hardware speed of processor**
 - **Shrinking logic gate size**
 - **More gates, packed more tightly, increasing clock rate**
 - **Propagation time for signals reduced**
- **Increase size and speed of cache memory**
- **Change processor organization and architecture**
 - **Increase effective speed of execution**

Approaches to achieving increased processor speed

- **Increase the hardware speed of the processor.** This increase is fundamentally due to shrinking the size of the logic gates on the processor chip, so that more gates can be packed together more tightly and to increasing the clock rate. With gates closer together, the propagation time for signals is significantly reduced, enabling a speeding up of the processor. An increase in clock rate means that individual operations are executed more rapidly.
- **Increase the size and speed of caches** that are interposed between the processor and main memory. In particular, by dedicating a portion of the processor chip itself to the cache, cache access times drop significantly.
- **Make changes to the processor organization and architecture** that increase the effective speed of instruction execution. Typically, this involves using parallelism in one form or another.

Problems with Increase clock speed and logic gate density in Processor

- **Power** density increases with density of logic gates and clock speed as a result **more heat dissipates**
- **May cause Delay in signal transmission within CPU due to increase in Resistance and Capacitance (RC Delay)**
 - wire interconnects become thinner as logic gate density increases, as a result resistance increases
 - Wires get closer increases capacitance

List the techniques used to build modern processors to speed up its operations

Pipelining

Processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously

Branch prediction

Processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next

Data flow analysis

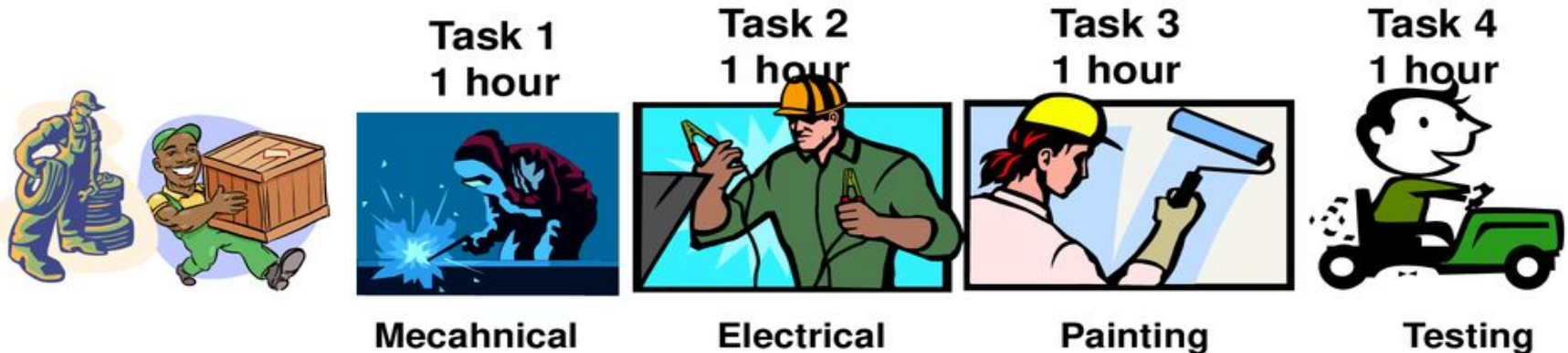
Processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions

Speculative execution

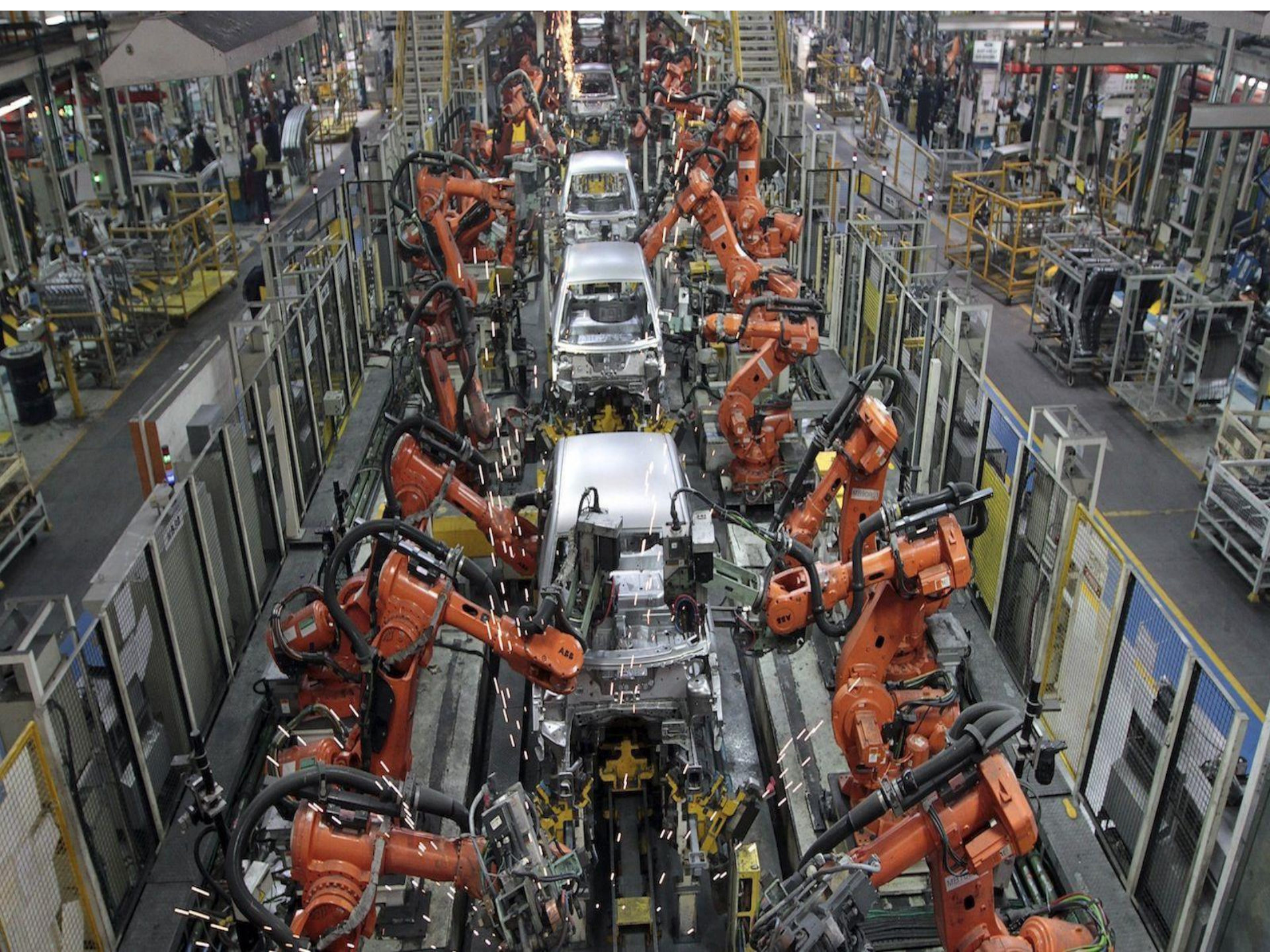
Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations, keeping execution engines as busy as possible

Pipelining example

Automobile Assembly Line



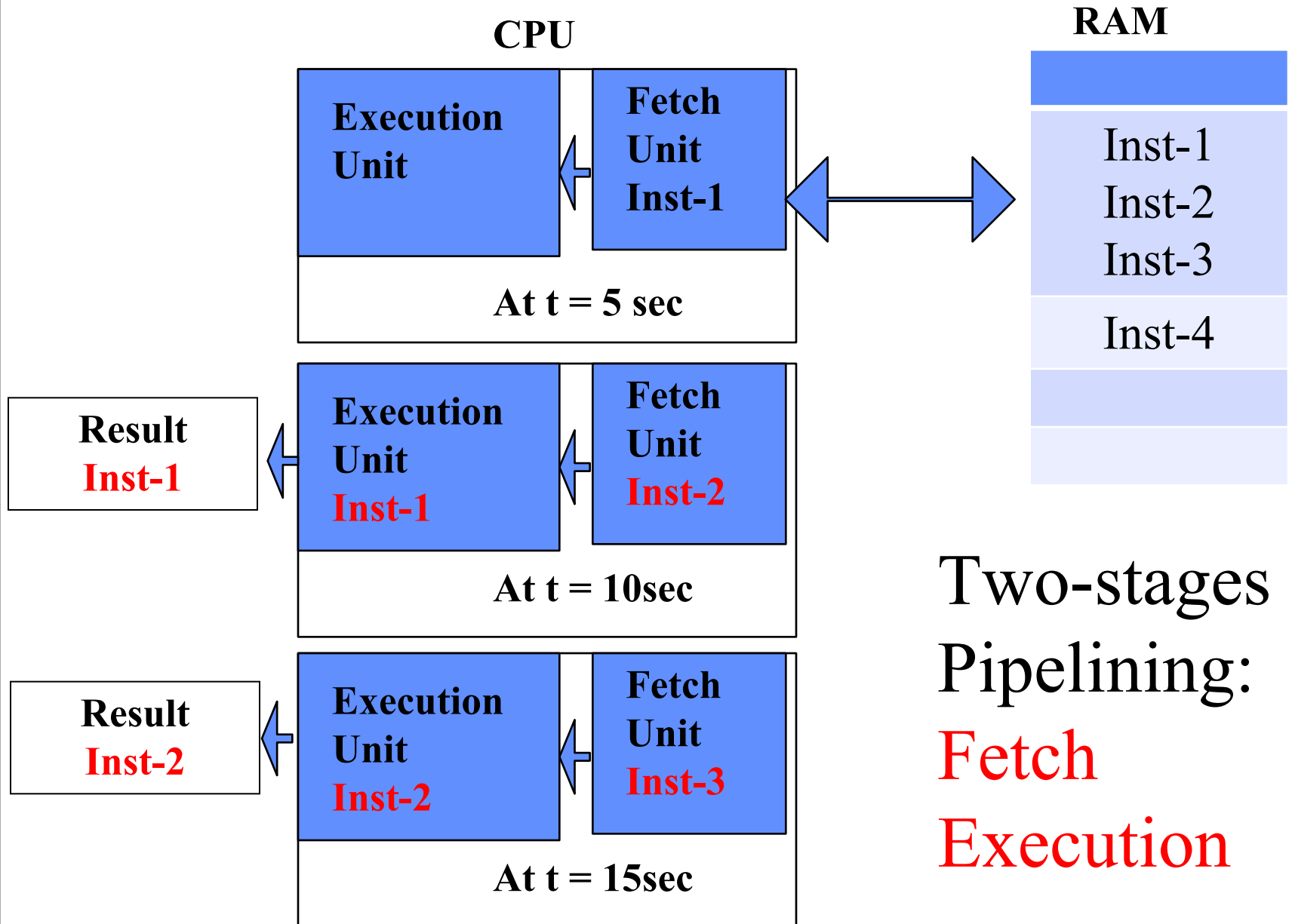
**First car assembled in 4 hours (pipeline latency)
thereafter 1 car per hour
21 cars on first day, thereafter 24 cars per day
717 cars per month
8,637 cars per year**





- A **pipeline** works much as an assembly line in a manufacturing plant enabling different stages of execution of different instructions to occur at the same time along the pipeline.

Two-stages Pipelining



Five-stage pipelining

Five-stage processing
of an instruction

Fetch Instruction from
Memory

Decode Instruction

Calculate data address and
read data from memory

Execute the operation

Write the result in register

Five-stages Hardware
Within processor

Fetch Unit (FU)

Decode Unit (DU)

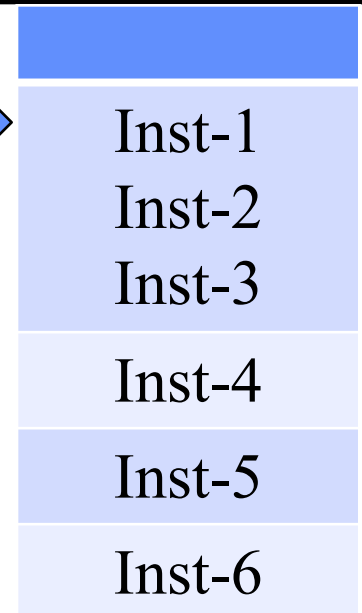
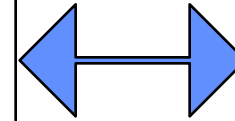
Memory Data (MD)

Execute Unit (EU)

Write Unit (WU)



CPU at $t = 0$



RAM



CPU at $t = 5\text{sec}$



CPU at $t = 10\text{sec}$



CPU at $t = 15\text{sec}$



CPU at $t = 20\text{sec}$

Five-stages
Pipelining

Five-stage pipelining

Five-stage processing
of an instruction

Five-stages Hardware
Within processor

Fetch Instruction from
Memory (**IF**)

Decode Instruction (**ID**)

Calculate data address and read
data from memory (**MEM**)

Execute the operation (**EX**)

Write the result in register (**WR**)

Fetch Unit (FU)

Decode Unit (DU)

Memory Data (MD)

Execute Unit (EU)

Write Unit (WU)

Pipelining: Time steps

Clock cycles progress left to right

Instructions progress top to bottom

Time at which each instruction is present in each pipeline stage is shown by labelling appropriate cell with pipeline name

Information from one instruction to any successor must always move from left to right

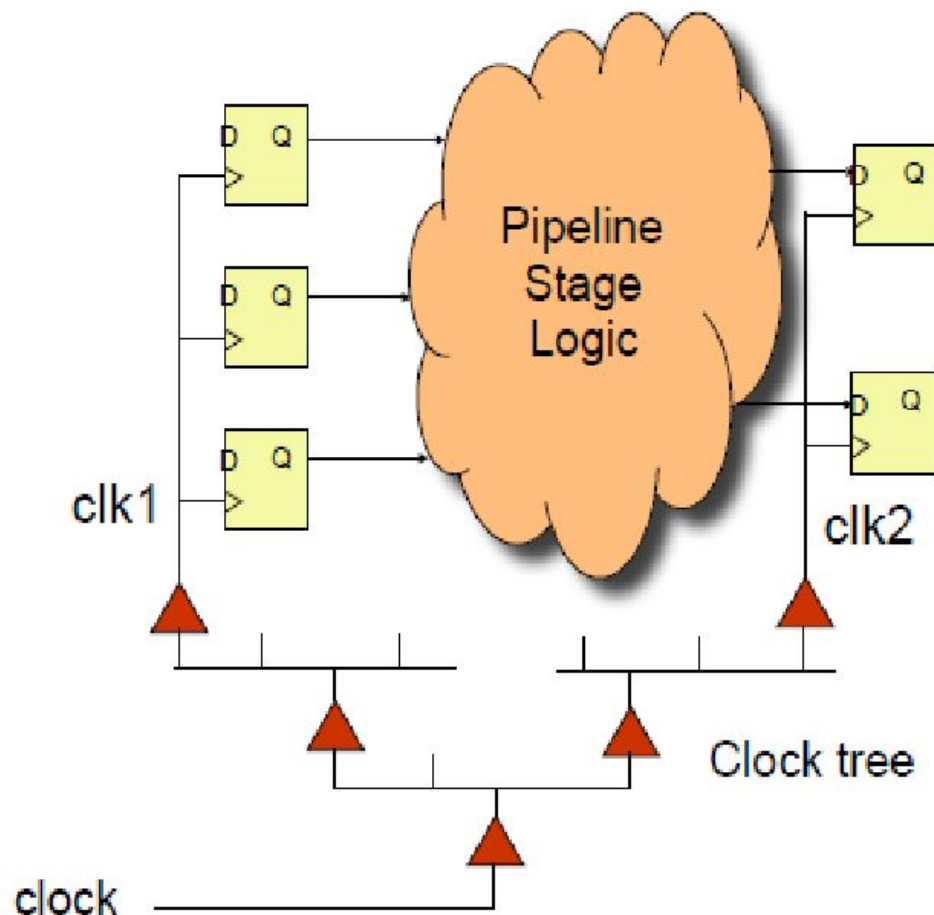
Instruction\ cycle	1	2	3	4	5	6	7	8
Instruction-1	IF	ID	MEM	EX	WR			
Instruction-2		IF	ID	MEM	EX	WR		
Instruction-3			IF	ID	MEM	EX		
Instruction-4				IF	ID	MEM		
Instruction-5					IF	ID		
Instruction-6						IF		
Instruction-7								
Instruction-8								

Clock cycles

[illegible]

Implement Issues

- Each pipeline stage is a combinational logic network
 - Registered inputs and outputs
 - Longest circuit delay through all stages determines clock period



Ideally, all delays through every pipeline stage are identical

In practice this is hard to achieve

Pipeline Hazards


- Hazards are pipeline events that restrict the pipeline flow
- They occur in circumstances where two or more activities cannot proceed in parallel
- There are three types of hazard:
 - Structural Hazards
 - Arise from resource conflicts, when a set of actions have to be performed sequentially because there is not sufficient resource to operate in parallel
 - Data Hazards
 - Occur when one instruction depends on the result of a previous instruction, and that result is not yet available. These hazards are exposed by the overlapped execution of instructions in a pipeline
 - Control Hazards
 - These arise from the pipelining of branch instructions, and other activities that change the PC.

Structural Hazards

- Multi-cycle operation
- Memory or register file port restriction

Example structural hazard caused by having only one memory port

Instruction\ cycle	1	2	3	4	5	6	7	8
Instruction-1	IF	ID	MEM	EX	WR			
Instruction-2		IF	ID	MEM	EX	WR		
Instruction-3			IF	ID	MEM	EX		
Instruction-4				IF	ID	MEM		
Instruction-5					IF	ID		
Instruction-6						IF		
Instruction-7								
Instruction-8								

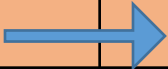


Effect is to STALL instruction 4, delaying its entry to IF by one cycle

Data Hazards

- Overlapped execution of Instructions means information may be required before it is available

Instruction\ cycle	1	2	3	4	5	6	7	8
ADD R1, R2, R3	IF	ID	MEM	EX	WR			
SUB R4, R1, R5		IF	ID	MEM	EX	WR		
AND R6, R1, R7			IF	ID	MEM	EX		
OR R8, R1, R9				IF	ID	MEM		
XOR R10, R1, R11					IF	ID		

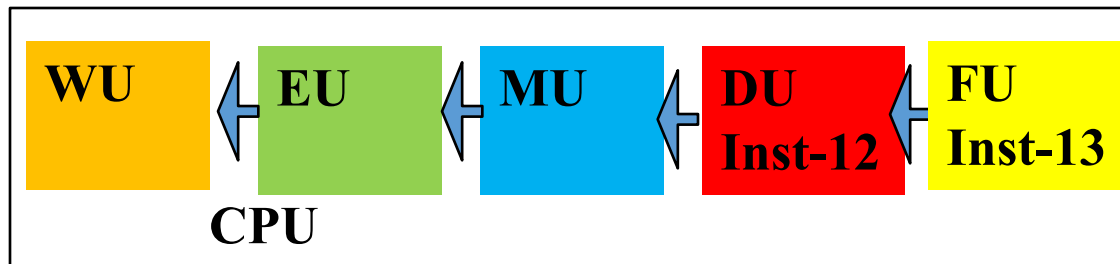
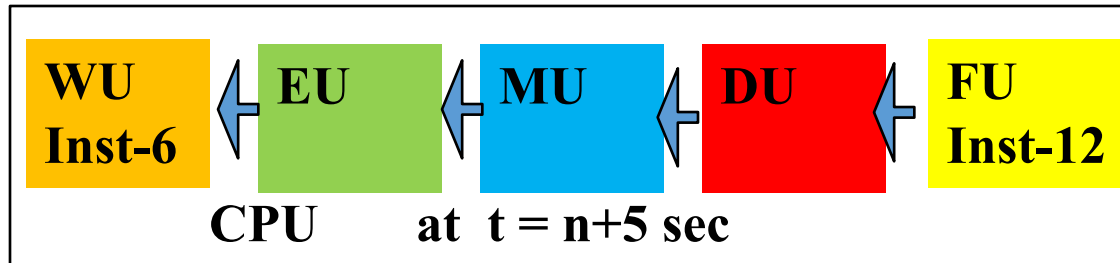
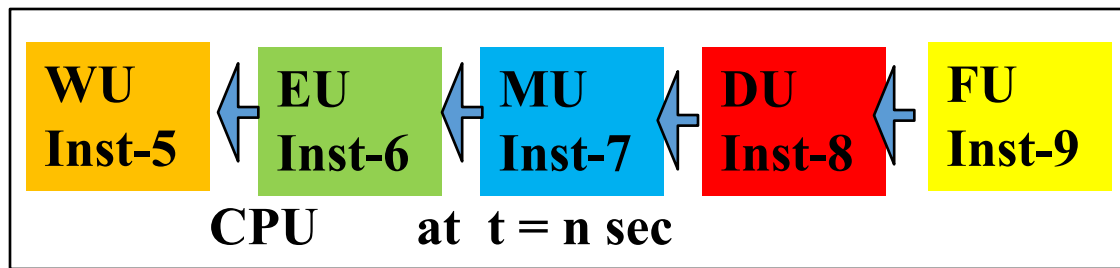


- SUB instruction must wait until R1 has been written to register file
- All subsequent instructions are similarly delayed

Control Hazards

- When a branch is executed, **Program Counter** is not affected until the branch instruction reaches the EU stage.
- By this time 3 instructions have been fetched from the fall-through path.

Instruction\cycle	1	2	3	4	5	6	7	8
SUB R1, R3, R3	IF	ID	MEM	EX	WR			
BEQZ R1, label		IF	ID	MEM	EX	WR		
OR R8, R4, R9			IF	ID	MEM	Clear Instructions in FU, DU, MU, EU		
MOV R2, R5				IF	ID			
DIV R10, R7					IF			
label: XOR R2, R3, R11						IF	ID	MEM
SUB R5, R3, R1							IF	ID
ADD M1, R6, R8								IF



Inst-6
conditional
branch

If condition
true: next
Instruction-12

If condition
False: next
Instruction-7



Example Problem

Assume that Instruction-3 is conditional branch. If the condition is TRUE, program control jumps to instruction-8. Show the pipelining stages assuming that the condition is evaluated TRUE.

[illegible]

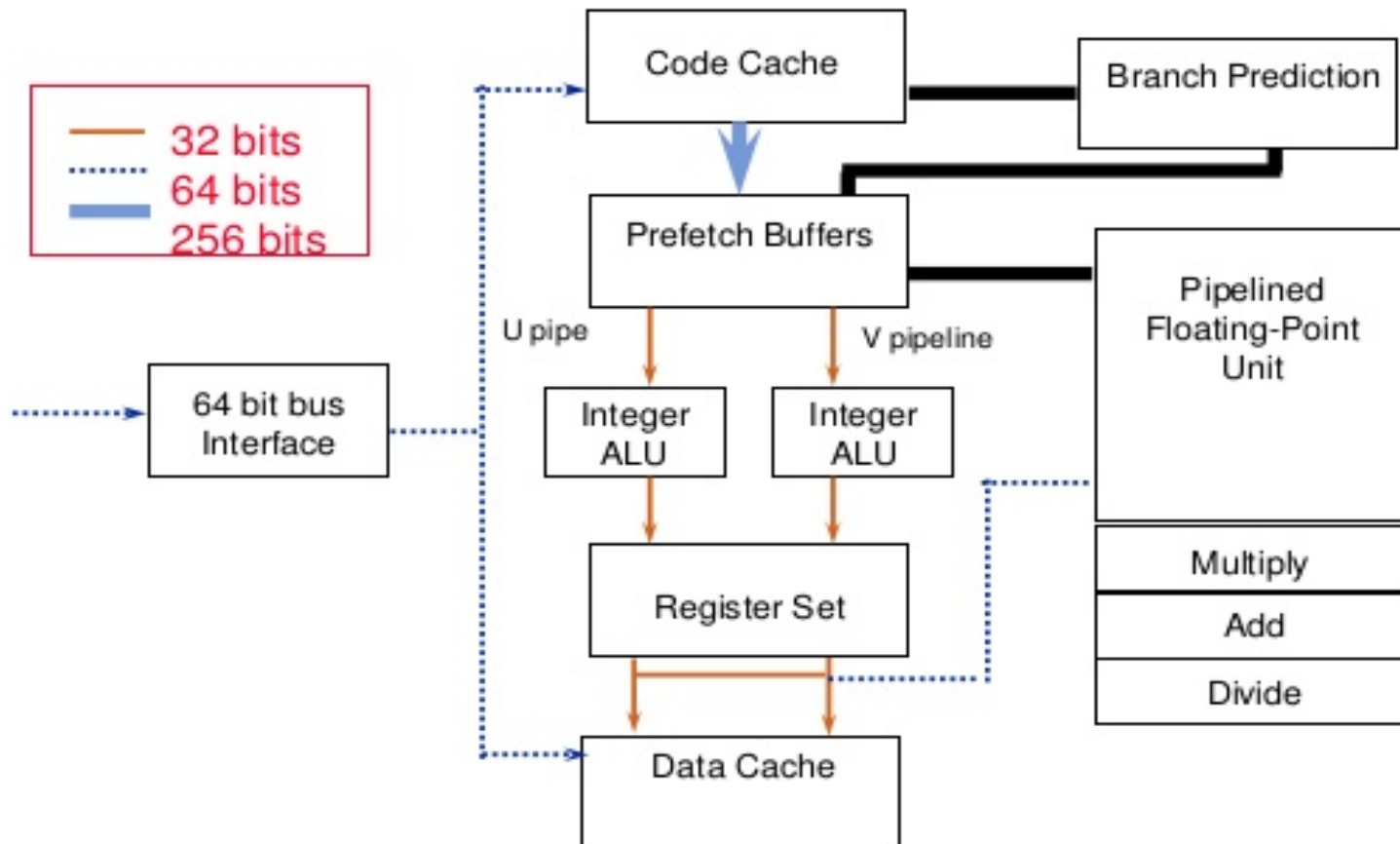
Example Problem

Assume that Instruction-4 is conditional branch. If the condition is TRUE, program control jumps to instruction-9. Show the pipelining stages assuming that the condition is evaluated TRUE.

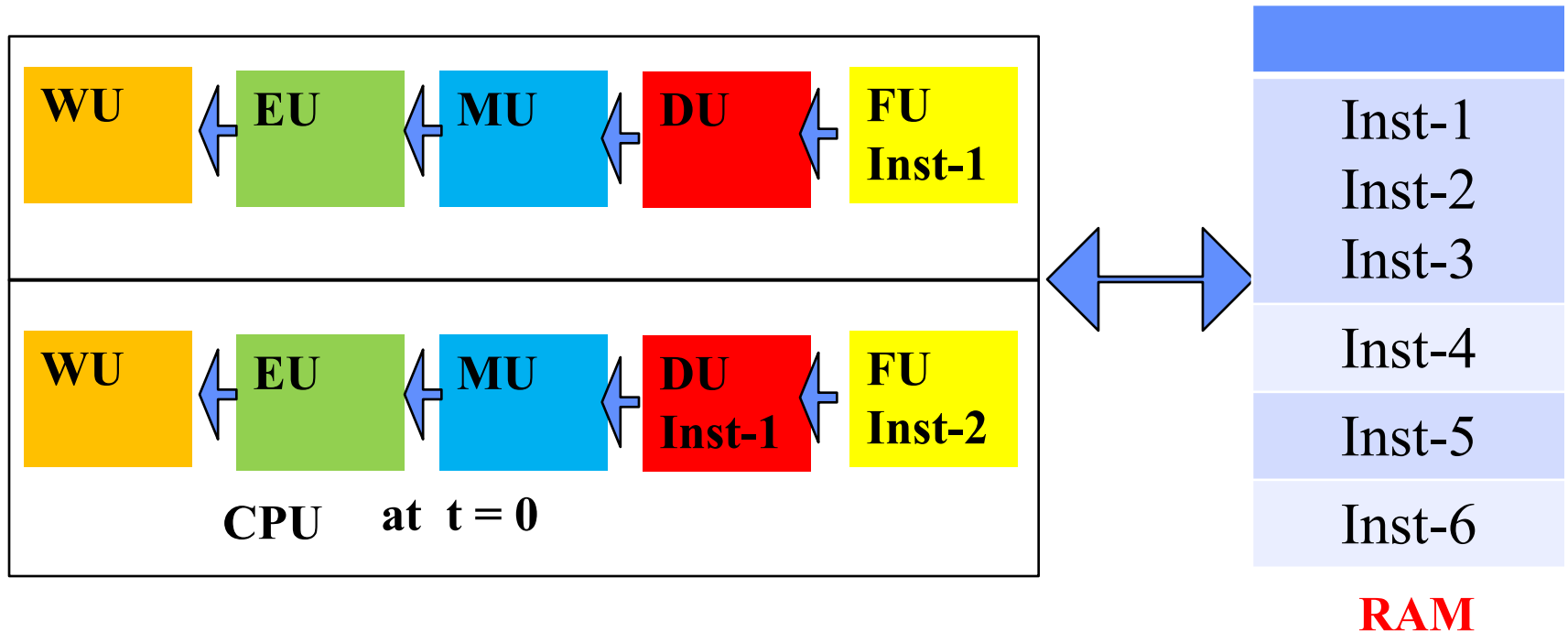
[illegible]

Branch Prediction: Pentium

PENTIUM™ PROCESSOR ARCHITECTURE

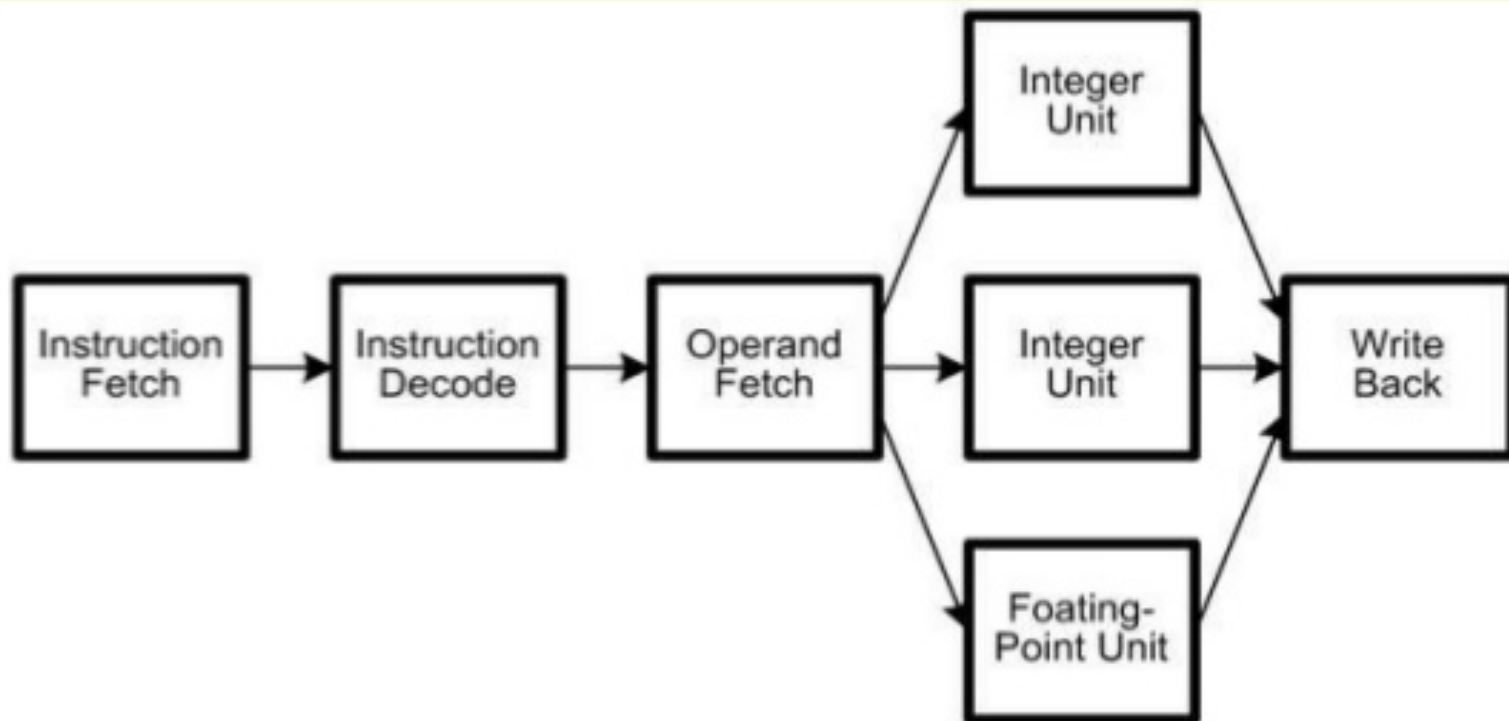


Superscalar Processor



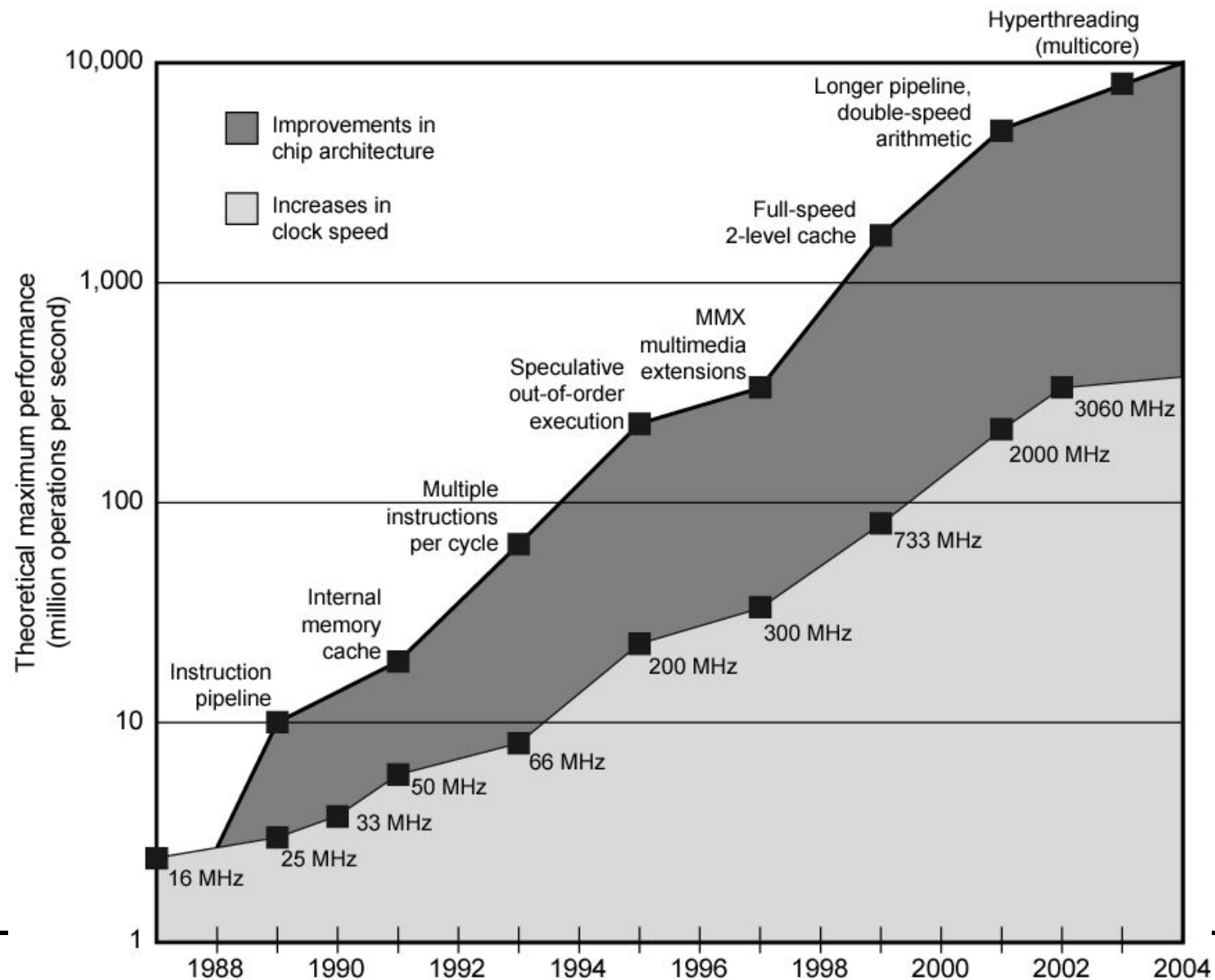
A **superscalar** approach in essence allows multiple pipelines within a single processor so that instructions that do not depend on one another can be executed in parallel.

Superscalar Operational Block Diagram



A Superscalar Processor with 3 Functional Units

Intel Microprocessor Performance



Following techniques to design and improve processor as well as system performance are believed to reach the limit and there is hardly any scope to improve/explore these further.

- Cache memory
- Pipelining
- Superscalar

New Approach – Multiple Cores

- However, simply relying on increasing clock rate for increased performance runs into the power dissipation problem.
- The faster the clock rate, the greater the amount of power to be dissipated.

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

- With all of these difficulties in mind, designers have turned to a fundamentally new approach to improving performance: placing **multiple processors on the same chip**, with a large shared cache.
- The use of multiple processors on the same chip, also referred to as multiple cores, or multicore, provides the potential to increase performance without increasing the clock rate.

New Approach – Multiple Cores

Multiple processors on single chip with
Large shared cache

- Within a processor, increase in performance **proportional to square root of increase in design complexity**

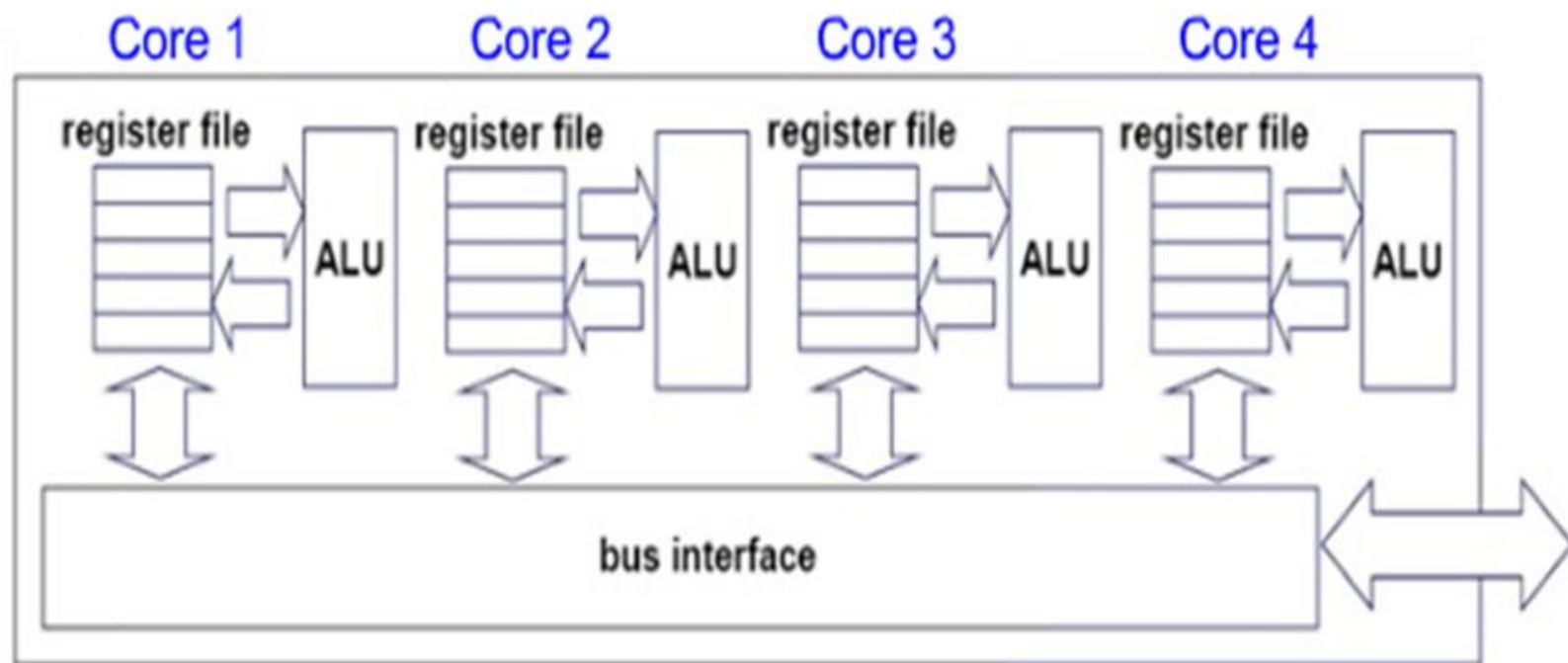
$$performance \propto \sqrt{complexity}$$

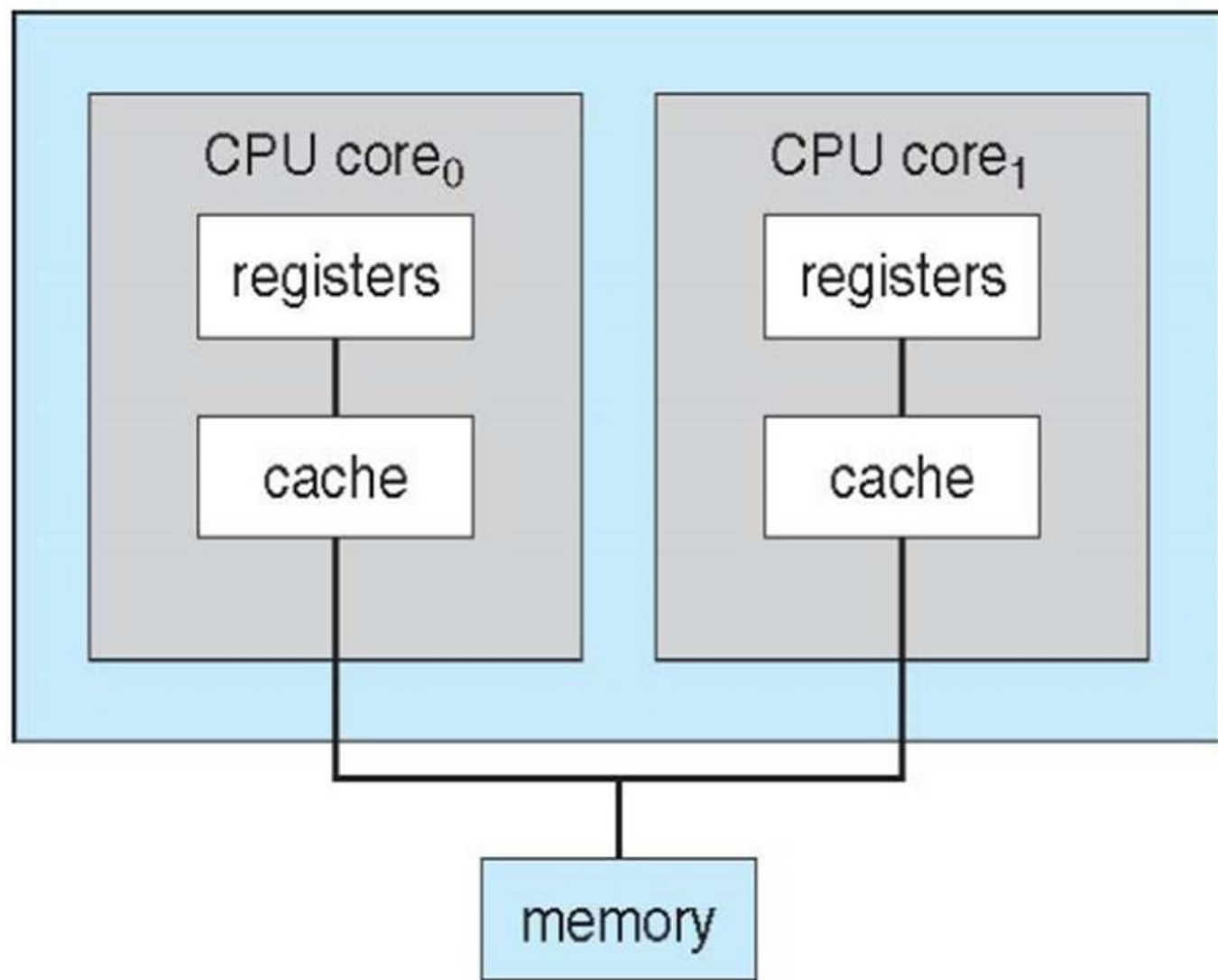
- If software can use multiple processors, **doubling number of processors almost doubles performance**

$$performance \propto no\ of\ cores$$

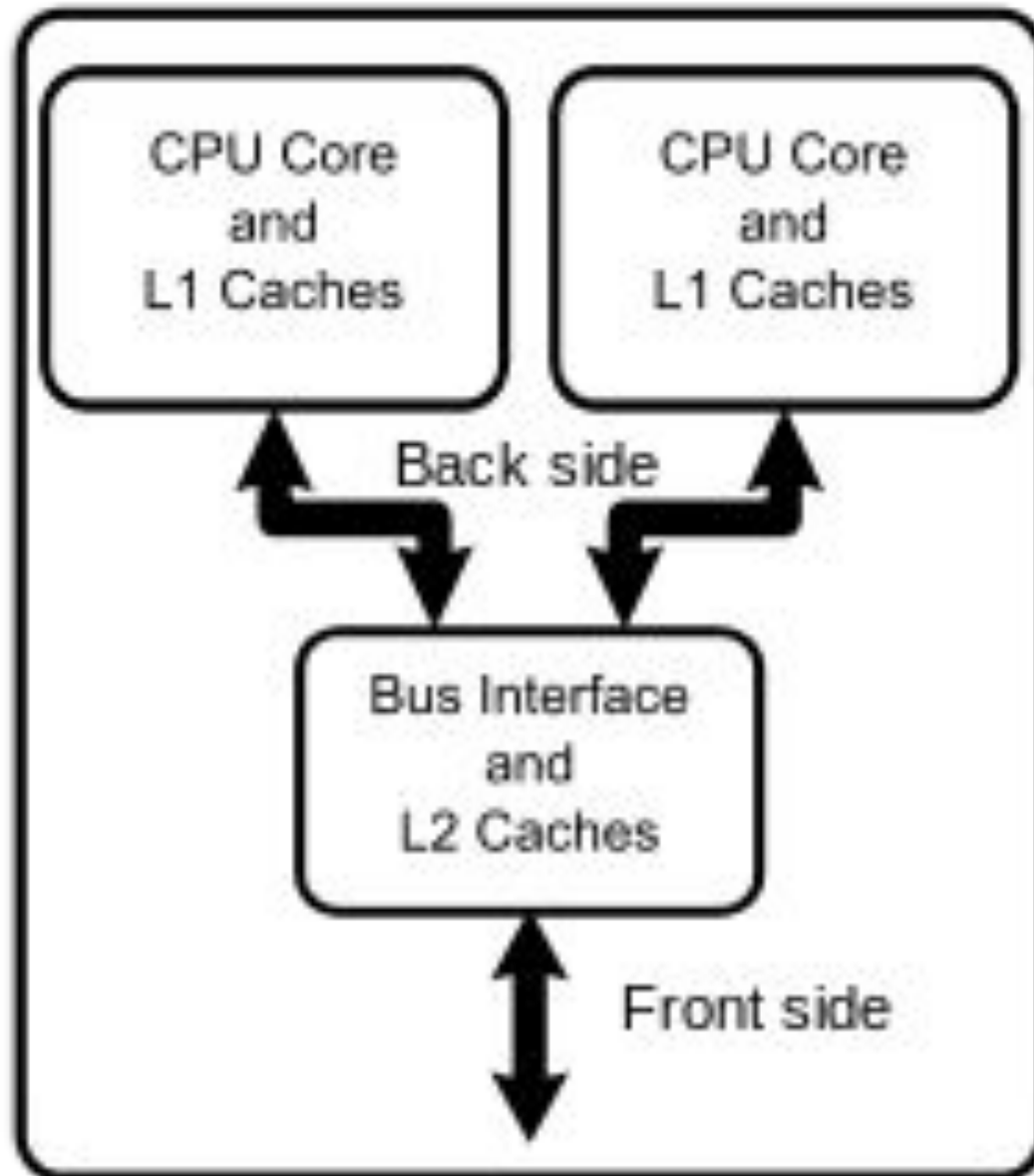
- Studies indicate that, within a processor, **the increase in performance is roughly proportional to the square root of the increase in complexity**. But if the software can support the effective use of multiple processors, **then doubling the number of processors almost doubles performance**. Thus, the strategy is to use two simpler processors on the chip rather than one more complex processor.
- In addition, with two processors, larger caches are justified. This is important because **the power consumption of memory logic on a chip is much less than that of processing logic**. In coming years, we can expect that most new processor chips will have multiple processors.

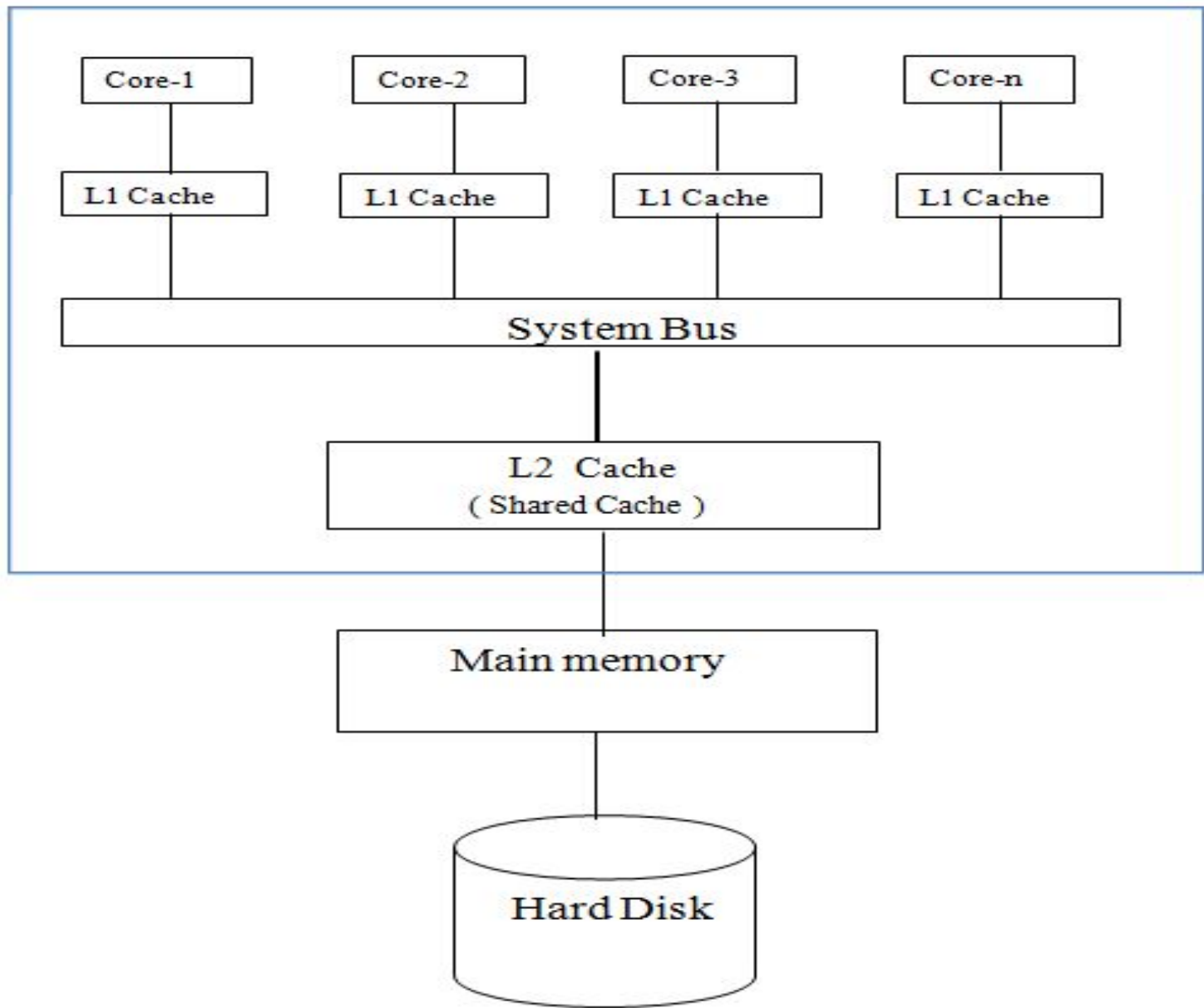
Multi-core architecture





Multicore processor

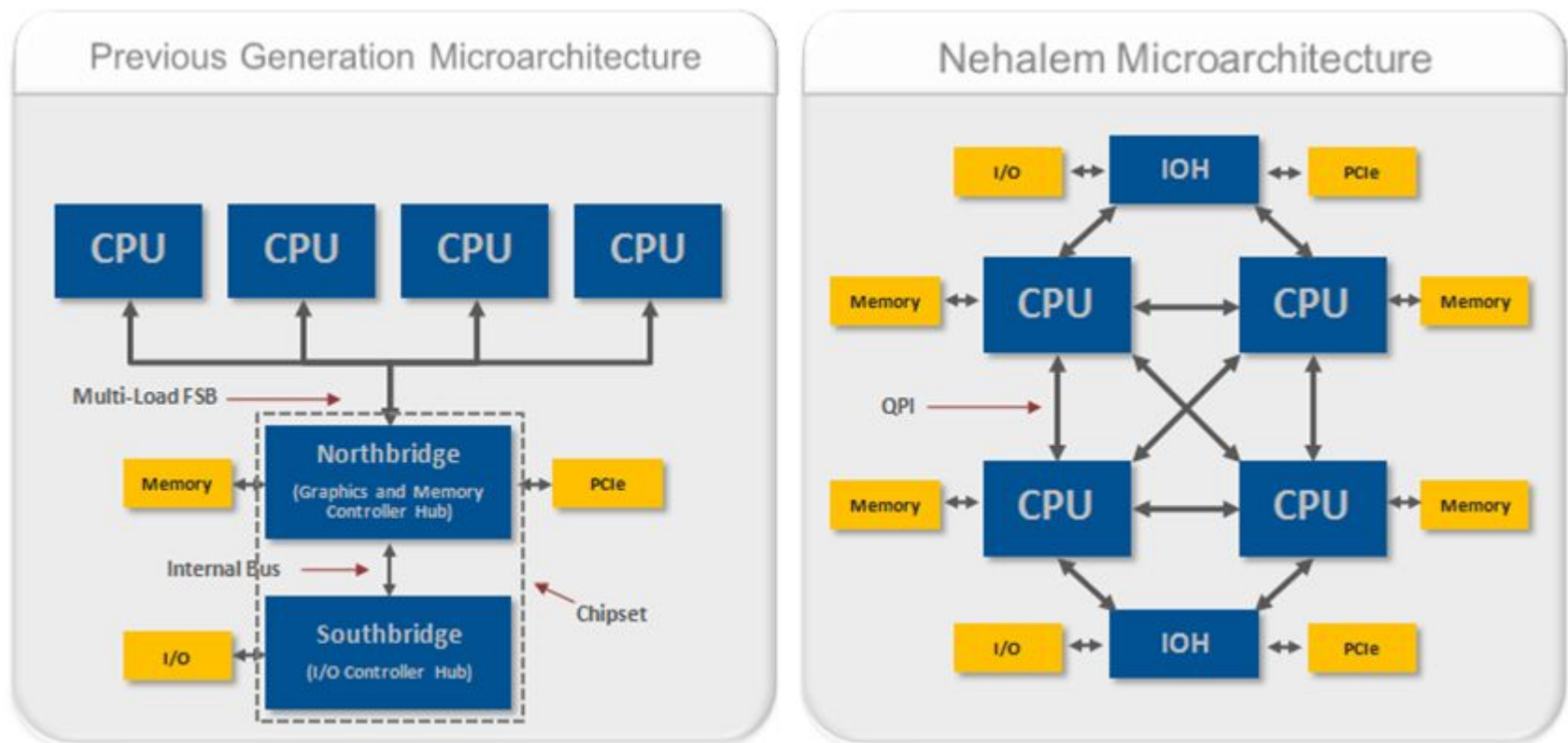




Multicore processor

<http://www.ni.com/white-paper/11266/en/>

Core i3 processors have two cores, Core i5 CPUs have four and Core i7 models also have four. Some Core i7 Extreme processors have six or eight cores. Generally speaking, we find that most applications can't take full advantage of six or eight cores.



Core i9 processor

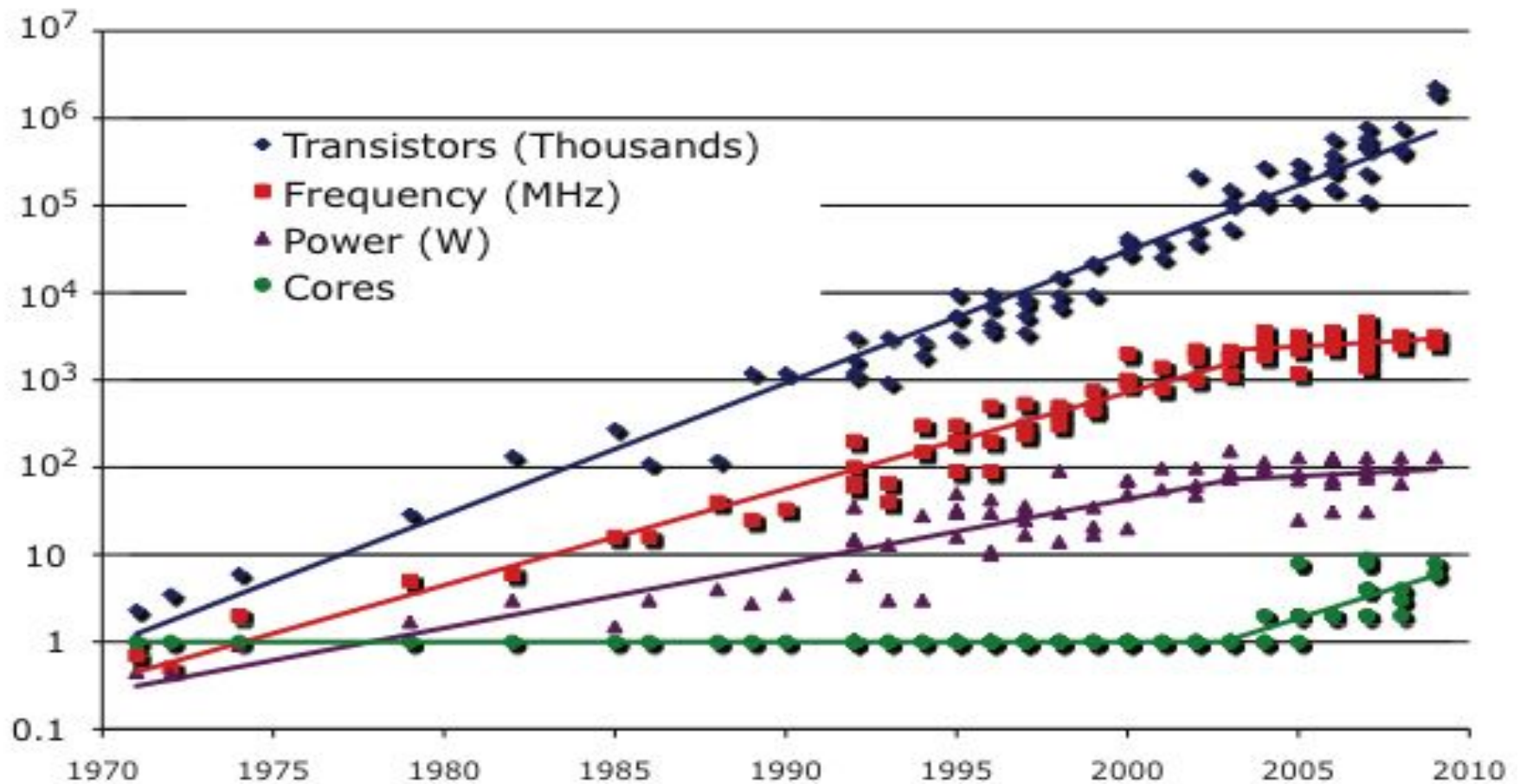
A family of 64-bit x86 CPUs with up to 18 cores from Intel. Designed for high-performance computing and gaming.

Processor	No. of Cores & Clock
i9-7900X (June 2017)	10
i9-7980XE (Sept 2017)	18
<u>i9-9990XE</u> (January 2018)	14

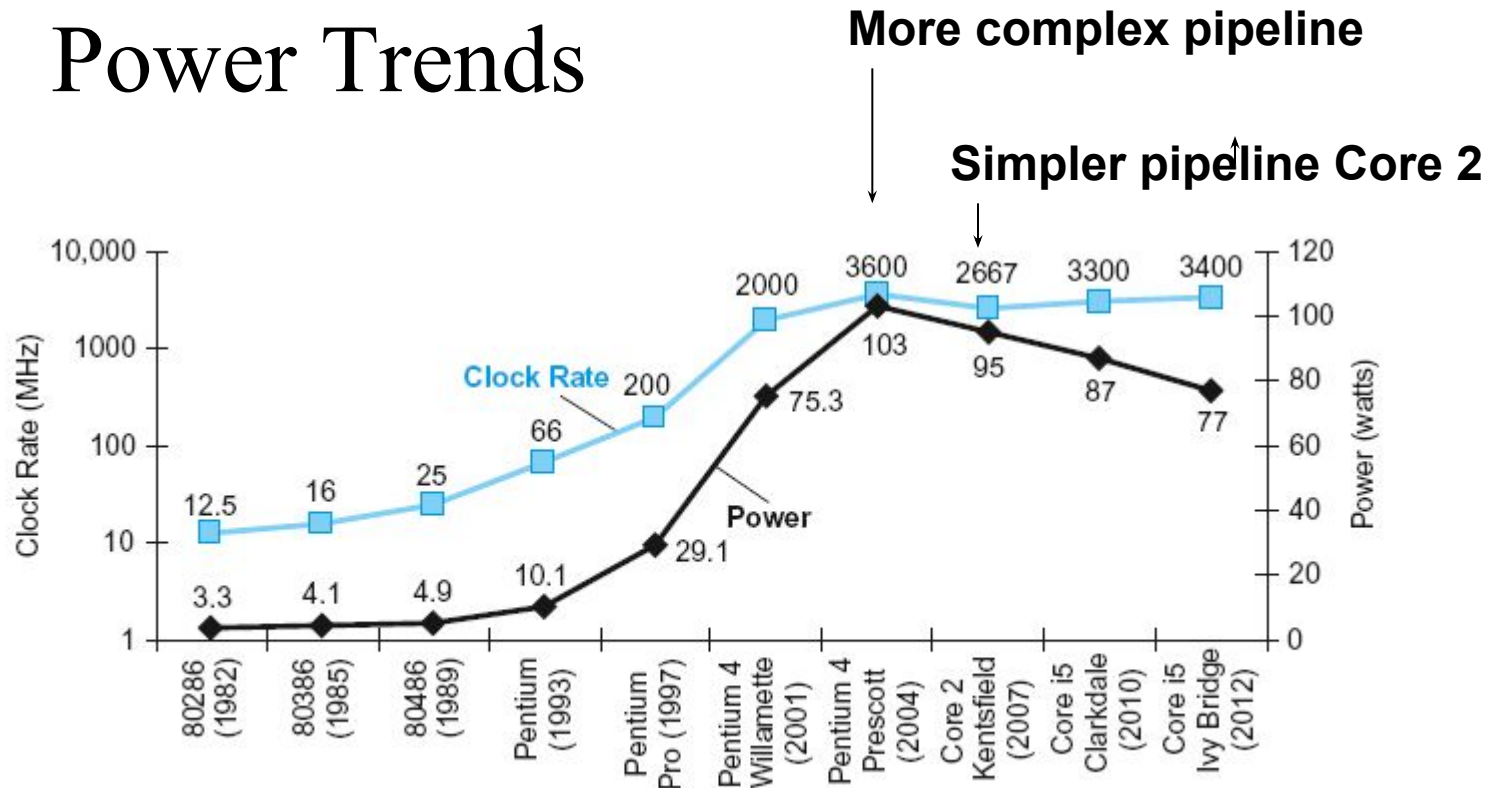




Processor Trends



Power Trends



- In CMOS IC technology

CMOS primary energy consumption is dynamic energy, switch on->off; off->on controlled by the clock freq.

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

Dynamic Power

5V → 1V

×1000

Reducing Power

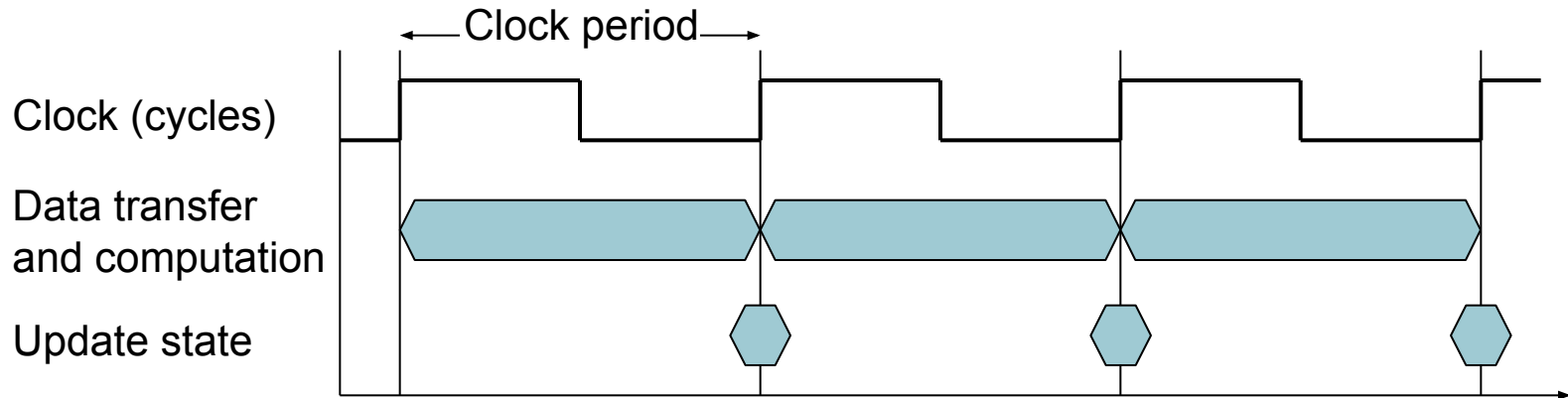
- Suppose a new CPU has
 - 85% of capacitive load of old CPU
 - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- **The power wall**
 - We can't reduce voltage further
 - We can't remove more heat
- **How else can we improve performance?**

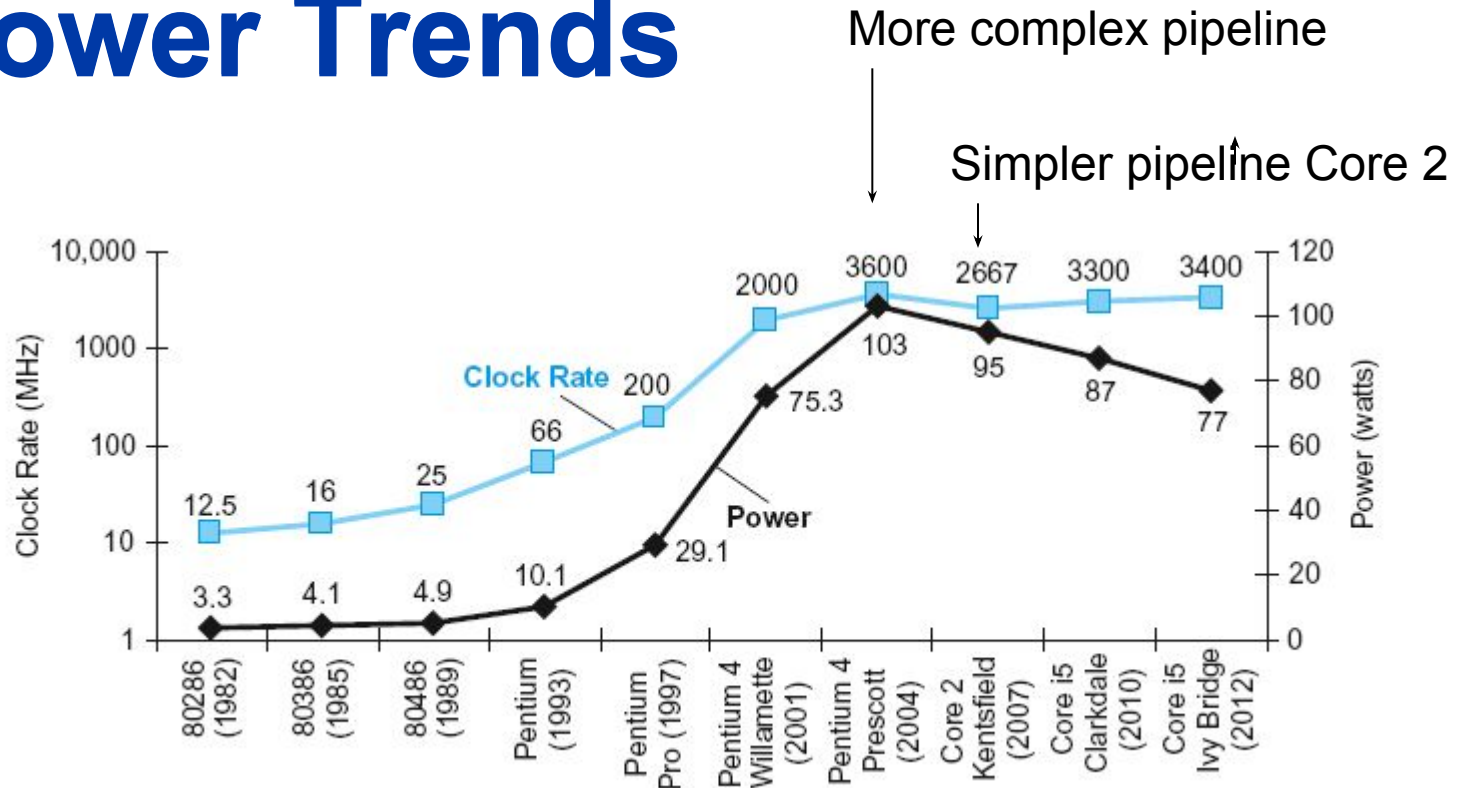
CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- 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): cycles per second
 - e.g., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

Power Trends



■ In CMOS IC technology

CMOS primary energy consumption is dynamic energy, switch on->off; off->on controlled by the clock freq.

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

Dynamic Power

×30

5V → 1V

×1000

Reducing Power

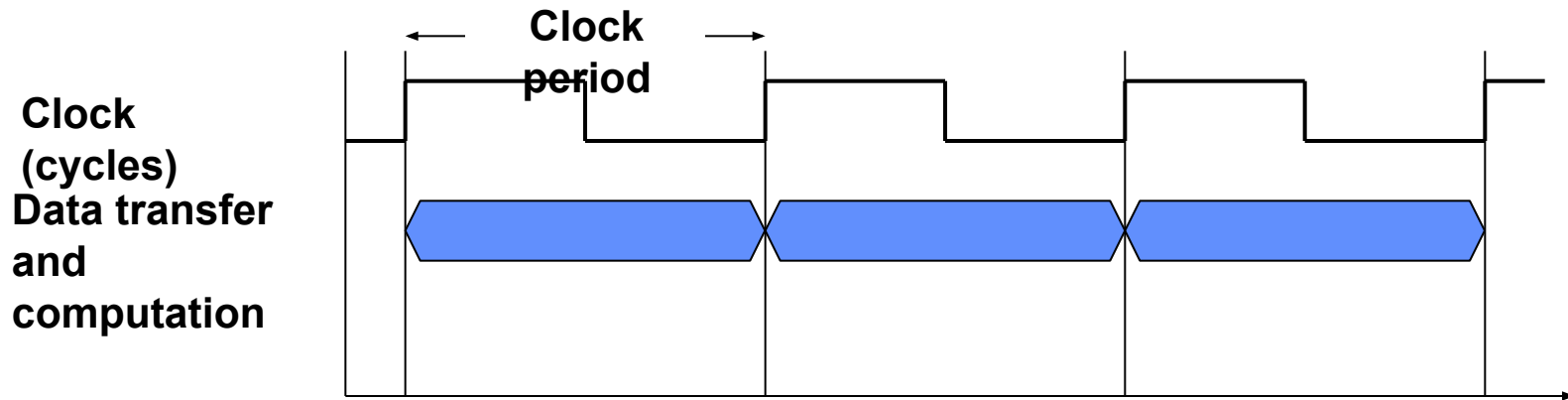
- Suppose a new CPU has
 - 85% of capacitive load of old CPU
 - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- The power wall
 - We can't reduce voltage further
 - We can't remove more heat
- How else can we improve performance?

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- 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): cycles per second
 - e.g., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

The CPU clock rate depends on the specific CPU organization (design) and hardware implementation technology (VLSI) used.

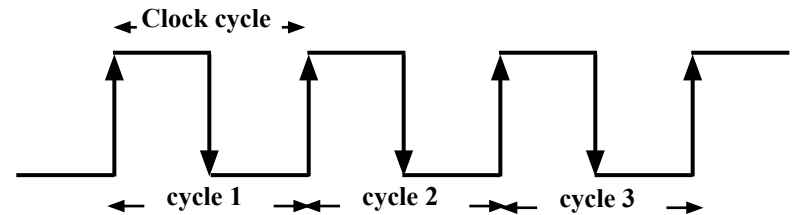
Micro operations

- A computer machine instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the instruction and the exact CPU organization (Design).
 - A micro operation is an elementary hardware operation that can be performed during one CPU clock cycle.
 - This corresponds to one micro-instruction in microprogrammed CPUs.
 - Examples: register operations: shift, load, clear, increment, ALU operations: add , subtract, etc.

Cycles Per Instruction (CPI)

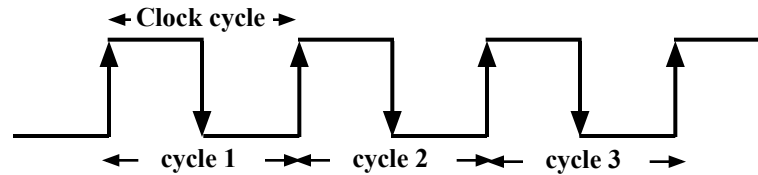
Clock rate (*Frequency*), $f = 1 / \text{clock cycle}$

Clock Cycle, $C = 1 / f$



- A single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI).

Example: Cycles Per Instruction (CPI)



CPU Clock rate, $f = 1\text{MHz}$; Clock Cycle, $C = 1$ micro second

- If CPU takes **1 micro second** to complete an Instruction, then **CPI = 1** for that Instruction.
- If CPU takes **3 micro seconds** to complete another Instruction, then **CPI = 3** for that Instruction.

If CPU Clock rate, $f = 2\text{MHz}$; Clock Cycle, $C = 0.5$ micro second

- If CPU takes **1 micro second** to complete an Instruction, then **CPI = 2** for that Instruction.
- If CPU takes **3 micro seconds** to complete another Instruction, then **CPI = 6** for that Instruction.

Average CPI

For a given program executed on a given CPU

Avg CPI = $\frac{\text{Total CPU Clock cycles for all instruction of program}}{\text{Instructions count in the program}}$

Instructions count in the program

Example: A program contains following instructions. CPI for different types of instructions are indicated. Calculate the Average CPI of the CPU for the program.

Instruction type	Instruction Count	CPI
ALU	500	2
Load	200	4
Store	200	4
Branch	100	6

Answer: Avg CPI = $(500 \times 2 + 200 \times 4 + 200 \times 4 + 100 \times 6) / 1000$

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C	
CPI for class	1	2	3	
IC in sequence 1	2	1	2	2+1+2=5 inst.
IC in sequence 2	4	1	1	4+1+1=6 inst.

- Sequence 1: IC = 5

- Clock Cycles
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$
 $= 10$
- Avg. CPI = $10/5 = 2.0$

- Sequence 2: IC = 6

- Clock Cycles
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$
 $= 9$
- Avg. CPI = $9/6 = 1.5$

CPU Execution Time: The CPU Equation

- A program is comprised of a number of instructions executed , I
 - Measured in: instructions/program
- The average instruction executed takes a number of *cycles per instruction (CPI)* to be completed.
 - Measured in: cycles/instruction, CPI
- CPU has a fixed clock cycle time $C = 1/\text{clock rate}$
 - Measured in: seconds/cycle
- CPU execution time is the product of the above three parameters as follows:

Or Instructions Per Cycle (IPC):
 $IPC = 1/CPI$

$$C = 1/f$$

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

$$T = I \times CPI \times C$$

execution Time
per program in seconds

Number of
instructions executed

Average CPI for program

CPU Clock Cycle

(This equation is commonly known as the CPU performance equation)

Computer Performance Measures: Program Execution Time

- For a specific program compiled to run on a specific machine (CPU) “A”, has the following parameters:
 - The total executed instruction count of the program. I
 - The average number of cycles per instruction (average CPI). CPI
 - Clock cycle of machine “A” C
- How can one measure the performance of this machine (CPU) running this program?
 - Intuitively the machine (or CPU) is said to be faster or has better performance running this program if the total execution time is shorter.
 - Thus the inverse of the total measured program execution time is a possible performance measure or metric:

$$Performance_A = 1 / Execution Time_A$$

Comparing Computer Performance Using Execution Time

- To compare the performance of two machines (or CPUs) “A”, “B” running a given specific program:

$$\text{Performance}_A = 1 / \text{Execution Time}_A$$

$$\text{Performance}_B = 1 / \text{Execution Time}_B$$

- Machine A is n times faster than machine B means (or slower? if $n < 1$) :

$$\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}$$

(i.e Speedup is ratio of performance, no units)

- Example:

For a given program:

Execution time on machine A: $\text{Execution}_A = 1$ second

Execution time on machine B: $\text{Execution}_B = 10$ seconds

$$\begin{aligned} \text{Speedup} &= \text{Performance}_A / \text{Performance}_B = \text{Execution Time}_B / \text{Execution Time}_A \\ &= 10 / 1 = 10 \end{aligned}$$

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program.

Relative Performance

- Define Performance = 1/Execution Time
- “X is n time faster than Y”

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

- **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

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}\end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

CPU Execution Time: Example

- A Program is running on a specific machine (CPU) with the following parameters:

I

 - Total executed instruction count: 10,000,000 instructions
 - Average CPI for the program: 2.5 cycles/instruction.
 - CPU clock rate: 200 MHz. (clock cycle = $C = 5 \times 10^{-9}$ seconds)
i.e 5 nanoseconds
- What is the execution time for this program:

CPU time	=	<u>Seconds</u>	=	<u>Instructions</u>	<u>x</u>	<u>Cycles</u>	<u>x</u>	<u>Seconds</u>
		Program		Program		Instruction		Cycle

$$\begin{aligned}\text{CPU time} &= \text{Instruction count} \times \text{CPI} \times \text{Clock cycle} \\ &= 10,000,000 \times 2.5 \times 1 / \text{clock rate} \\ &= 10,000,000 \times 2.5 \times 5 \times 10^{-9} \\ &= 0.125 \text{ seconds}\end{aligned}$$

$T = I \times \text{CPI} \times C$

Aspects of CPU Execution Time

CPU Time = Instruction count executed x CPI x Clock cycle

$$T = I \times CPI \times C$$

Depends on:
Program Used
Compiler
ISA

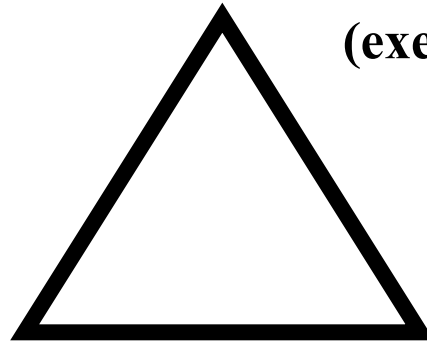
Instruction Count I
(executed)

Depends on:
Program Used
Compiler
ISA
CPU Organization

CPI
(Average
CPI)

**Clock
Cycle C**

Depends on:
CPU Organization
Technology (VLSI)



Factors Affecting CPU Performance

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

T	=	I	x	CPI	x	C
		Instruction Count		Cycles per Instruction		Clock Rate (1/C)
Program						
Compiler						
Instruction Set Architecture (ISA)						
Organization (CPU Design)						
Technology (VLSI)						

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

	instr. count	CPI	clock rate
Program	X	X	
Compiler	X	X	
Instr. Set Arch.	X	X	
Organization		X	X
Technology			X

Performance Comparison: Example

- From the previous example: A Program is running on a specific machine (CPU) with the following parameters:
 - Total executed instruction count, I: 10,000,000 instructions
 - Average CPI for the program: 2.5 cycles/instruction.
 - CPU clock rate: 200 MHz. Thus: $C = 1/(200 \times 10^6) = 5 \times 10^{-9}$ seconds
- Using the same program with these changes:
 - A new compiler used: New executed instruction count, I: 9,500,000
New CPI: 3.0
 - Faster CPU implementation: New clock rate = 300 MHz
- What is the speedup with the changes? Thus: $C = 1/(300 \times 10^6) = 3.33 \times 10^{-9}$ seconds

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{Clock cycle}_{\text{old}}}{I_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{Clock Cycle}_{\text{new}}}$$

$$\begin{aligned}\text{Speedup} &= (10,000,000 \times 2.5 \times 5 \times 10^{-9}) / (9,500,000 \times 3 \times 3.33 \times 10^{-9}) \\ &= .125 / .095 = 1.32 \\ &\text{or } 32 \% \text{ faster after changes.}\end{aligned}$$

Instruction Types & CPI

- Given a program with n types or classes of instructions executed on a given CPU with the following characteristics:

C_i = Count of instructions of type _{i} executed

CPI_i = Cycles per instruction for type _{i} $i = 1, 2, \dots, n$

Then:

Depends on CPU Design

$$CPI = \text{CPU Clock Cycles} / \text{Instruction Count } I$$

i.e average or effective CPI

Executed

Where:

$$CPU \text{ clock cycles} = \sum_{i=1}^n (CPI_i \times C_i)$$

$$\text{Executed Instruction Count } I = \sum C_i$$

Instruction Types & CPI: An Example

- An instruction set has three instruction classes:

Instruction class	CPI
A	1
B	2
C	3

For a specific CPU design

- Two code sequences have the following instruction counts:

Instruction counts for instruction class			
Code Sequence	A	B	C
1	2	1	2
2	4	1	1

- CPU cycles for sequence 1 = $2 \times 1 + 1 \times 2 + 2 \times 3 = 10$ cycles

CPI for sequence 1 = clock cycles / instruction count

i.e average or effective CPI

$$= 10 / 5 = 2$$

- CPU cycles for sequence 2 = $4 \times 1 + 1 \times 2 + 1 \times 3 = 9$ cycles

CPI for sequence 2 = $9 / 6 = 1.5$

$$CPU \text{ clock cycles} = \sum_{i=1}^n (CPI_i \times C_i)$$

$$CPI = CPU \text{ Cycles} / I$$

Instruction Frequency & CPI

- Given a program with n types or classes of instructions with the following characteristics:

C_i = Count of instructions of type _{i} executed

$i = 1, 2, \dots, n$

CPI_i = Average cycles per instruction of type _{i}

F_i = Frequency or fraction of instruction type _{i} executed

= C_i / total executed instruction

Where: Executed Instruction Count $I = \sum C_i$

Then:

$$CPI = \sum_{i=1}^n (CPI_i \times F_i)$$

i.e average or effective CPI

Fraction of total execution time for instructions of type $i = \frac{CPI_i \times F_i}{CPI}$

$$T = I \times CPI \times C$$

Instruction Type Frequency & CPI: A RISC Example

Program Profile or Executed Instructions Mix

Base Machine (Reg / Reg)

Depends on CPU Design

$$\frac{CPI_i \times F_i}{CPI}$$

% Time

Given

Op	Freq, F_i	CPI_i	$CPI_i \times F_i$
ALU	50%	1	.5
Load	20%	5	1.0
Store	10%	3	.3
Branch	20%	2	.4

$$23\% = .5/2.2$$

$$45\% = 1/2.2$$

$$14\% = .3/2.2$$

$$18\% = .4/2.2$$

Typical Mix

Sum = 2.2

$$CPI = \sum_{i=1}^n (CPI_i \times F_i)$$

i.e average or effective CPI

$$CPI = .5 \times 1 + .2 \times 5 + .1 \times 3 + .2 \times 2 = 2.2$$

$$= .5 + 1 + .3 + .4$$

$$T = I \times CPI \times C$$

Computer Performance Measures :

MIPS (Million Instructions Per Second) Rating

- For a specific program running on a specific CPU the MIPS rating is a measure of how many millions of instructions are executed per second:

$$\begin{aligned}\text{MIPS Rating} &= \text{Instruction count} / (\text{Execution Time} \times 10^6) \\ &= \text{Instruction count} / (\text{CPU clocks} \times \text{Cycle time} \times 10^6) \\ &= (\text{Instruction count} \times \text{Clock rate}) / (\text{Instruction count} \times \text{CPI} \times 10^6) \\ &= \text{Clock rate} / (\text{CPI} \times 10^6)\end{aligned}$$

- Major problem with MIPS rating: As shown above the MIPS rating does not account for the count of instructions executed (I).
 - A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.
- In addition the MIPS rating:
 - Does not account for the instruction set architecture (ISA) used.
 - Thus it cannot be used to compare computers/CPU's with different instruction sets.
 - Easy to abuse: Program used to get the MIPS rating is often omitted.
 - Often the Peak MIPS rating is provided for a given CPU which is obtained using a program comprised entirely of instructions with the lowest CPI for the given CPU design which does not represent real programs.

$$T = I \times \text{CPI} \times C$$

Computer Performance Measures :

MIPS (Million Instructions Per Second) Rating

- **Under what conditions can the MIPS rating be used to compare performance of different CPUs?**
- **The MIPS rating is only valid to compare the performance of different CPUs provided that the following conditions are satisfied:**
 - 1 **The same program is used**
(actually this applies to all performance metrics)
 - 2 **The same ISA is used**
 - 3 **The same compiler is used**

⇒ (Thus the resulting programs used to run on the CPUs and obtain the MIPS rating are identical at the machine code level including the same instruction count) (binary)

Compiler Variations, MIPS & Performance:

An Example

- For a machine (CPU) with instruction classes:

Instruction class	CPI
A	1
B	2
C	3

- For a given high-level language program, two compilers produced the following executed instruction counts:

Instruction counts (in millions) for each instruction class			
Code from:	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

- The machine is assumed to run at a clock rate of 100 MHz.

Compiler Variations, MIPS & Performance: An Example (Continued)

$$\text{MIPS} = \text{Clock rate} / (\text{CPI} \times 10^6) = 100 \text{ MHz} / (\text{CPI} \times 10^6)$$

$$\text{CPI} = \text{CPU execution cycles} / \text{Instructions count}$$

$$\text{CPU clock cycles} = \sum_{i=1}^n (CPI_i \times C_i)$$

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{Clock rate}$$

- For compiler 1:
 - $\text{CPI}_1 = (5 \times 1 + 1 \times 2 + 1 \times 3) / (5 + 1 + 1) = 10 / 7 = 1.43$
 - $\text{MIPS Rating}_1 = 100 / (1.428 \times 10^6) = 70.0 \text{ MIPS}$
 - $\text{CPU time}_1 = ((5 + 1 + 1) \times 10^6 \times 1.43) / (100 \times 10^6) = 0.10 \text{ seconds}$
- For compiler 2:
 - $\text{CPI}_2 = (10 \times 1 + 1 \times 2 + 1 \times 3) / (10 + 1 + 1) = 15 / 12 = 1.25$
 - $\text{MIPS Rating}_2 = 100 / (1.25 \times 10^6) = 80.0 \text{ MIPS}$
 - $\text{CPU time}_2 = ((10 + 1 + 1) \times 10^6 \times 1.25) / (100 \times 10^6) = 0.15 \text{ seconds}$

MIPS rating indicates that compiler 2 is better
while in reality the code produced by compiler 1 is faster

Computer Performance Measures :

MFLOPS (Million FLOating-Point Operations Per Second)

- A floating-point operation is an addition, subtraction, multiplication, or division operation applied to numbers represented by a single or a double precision floating-point representation.
- MFLOPS, for a specific program running on a specific computer, is a measure of millions of floating point-operation (megaflops) per second:

$$\text{MFLOPS} = \text{Number of floating-point operations} / (\text{Execution time} \times 10^6)$$

- MFLOPS rating is a better comparison measure between different machines (applies even if ISAs are different) than the MIPS rating.
 - Applicable even if ISAs are different
- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no floating-point operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.
 - Peak MFLOPS rating for a CPU: Obtained using a program comprised entirely of the simplest floating point instructions (with the lowest CPI) for the given CPU design which does not represent real floating point programs.

Quantitative Principles of Computer Design

- **Amdahl's Law:**

The performance gain from improving some portion of a computer is calculated by:

— i.e using some enhancement

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for the entire task without using the enhancement}}$$

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

Here: Task = Program

Recall: Performance = 1 / Execution Time

Performance Enhancement Calculations:

Amdahl's Law

- The performance enhancement possible due to a given design improvement is limited by the amount that the improved feature is used
- Amdahl's Law:

Performance improvement or speedup due to enhancement E:

$$\text{Speedup}(E) = \frac{\text{Execution Time without E}}{\text{Execution Time with E}} = \frac{\text{Performance with E}}{\text{Performance without E}}$$

- Suppose that enhancement E accelerates a fraction F of the execution time by a factor S and the remainder of the time is unaffected then:

$$\text{Execution Time with E} = ((1-F) + F/S) \times \text{Execution Time without E}$$

Hence speedup is given by:

$$\text{Speedup}(E) = \frac{\text{Execution Time without E}}{((1 - F) + F/S) \times \text{Execution Time without E}} = \frac{1}{(1 - F) + F/S}$$

F (Fraction of execution time enhanced) refers to original execution time before the enhancement is applied

original

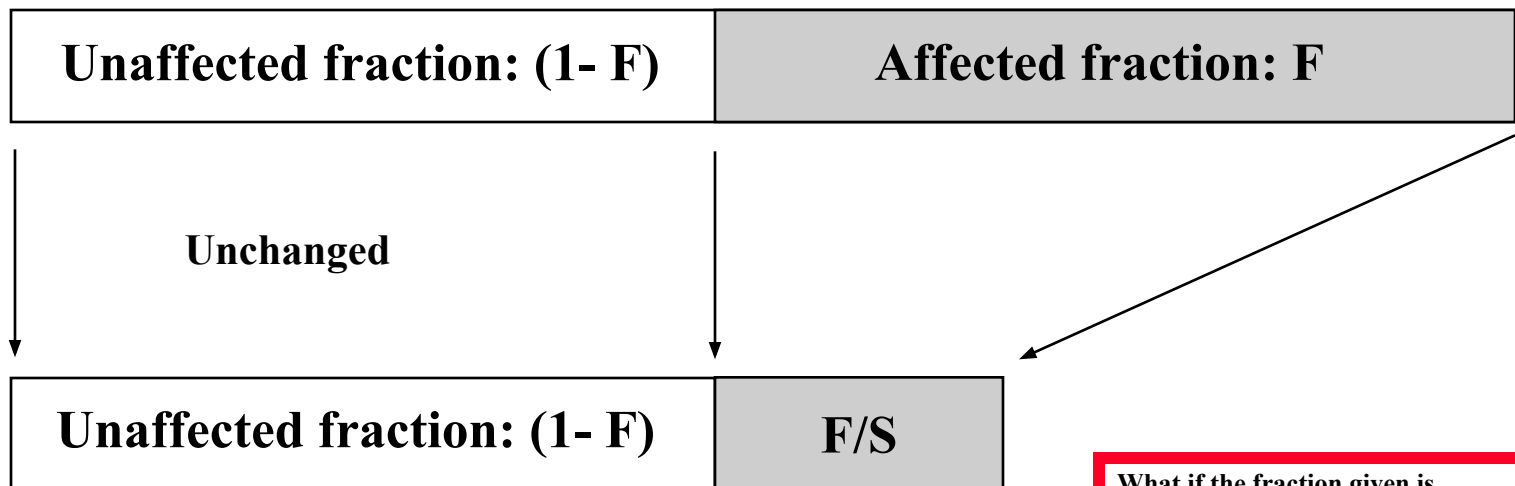
Pictorial Depiction of Amdahl's Law

Enhancement E accelerates fraction F of original execution time by a factor of S

Before:

Execution Time without enhancement E: (Before enhancement is applied)

- shown normalized to $1 = (1-F) + F = 1$



After:

Execution Time with enhancement E:

What if the fraction given is
after the enhancement has been applied?
How would you solve the problem?
(i.e find expression for speedup)

$$\text{Speedup}(E) = \frac{\text{Execution Time without enhancement E}}{\text{Execution Time with enhancement E}} = \frac{1}{(1 - F) + F/S}$$

Performance Enhancement Example

- For the RISC machine with the following instruction mix given earlier:

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

CPI = 2.2

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Fraction enhanced = $F = 45\%$ or $.45$

Unaffected fraction = $1 - F = 100\% - 45\% = 55\%$ or $.55$

Factor of enhancement = $S = 5/2 = 2.5$

Using Amdahl's Law:

$$\text{Speedup}(E) = \frac{1}{(1 - F) + F/S} = \frac{1}{.55 + .45/2.5} = 1.37$$

An Alternative Solution Using CPU Equation

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

$$\text{CPI} = 2.2$$

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Old CPI = 2.2

New CPI of load is now 2 instead of 5

$$\text{New CPI} = .5 \times 1 + .2 \times 2 + .1 \times 3 + .2 \times 2 = 1.6$$

$$\begin{aligned} \text{Speedup}(E) &= \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\cancel{\text{Instruction count}} \times \text{old CPI} \times \cancel{\text{clock cycle}}}{\cancel{\text{Instruction count}} \times \text{new CPI} \times \cancel{\text{clock cycle}}} \\ &= \frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37 \end{aligned}$$

Which is the same speedup obtained from Amdahl's Law in the first solution.

$$T = I \times \text{CPI} \times C$$

Performance Enhancement Example

- A program runs in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program four times faster?

$$\text{Desired speedup} = 4 = \frac{100}{\text{Execution Time with enhancement}}$$

→ Execution time with enhancement = $100/4 = 25$ seconds

$$25 \text{ seconds} = (100 - 80 \text{ seconds}) + 80 \text{ seconds} / S$$

$$25 \text{ seconds} = 20 \text{ seconds} + 80 \text{ seconds} / S$$

→ $5 = 80 \text{ seconds} / S$

→ $S = 80/5 = 16$

Alternatively, it can also be solved by finding enhanced fraction of execution time:

$$F = 80/100 = .8$$

and then solving Amdahl's speedup equation for desired enhancement factor S

$$\text{Speedup}(E) = \frac{1}{(1 - F) + F/S} = 4 = \frac{1}{(1 - .8) + .8/S} = \frac{1}{.2 + .8/s}$$

Hence multiplication should be 16 times

Solving for S gives S= 16

faster to get an overall speedup of 4.

Performance Enhancement Example

- For the previous example with a program running in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program five times faster?

$$\text{Desired speedup} = 5 = \frac{100}{\text{Execution Time with enhancement}}$$

$$\rightarrow \text{Execution time with enhancement} = 100/5 = 20 \text{ seconds}$$

$$20 \text{ seconds} = (100 - 80 \text{ seconds}) + 80 \text{ seconds} / s$$

$$20 \text{ seconds} = 20 \text{ seconds} + 80 \text{ seconds} / s$$

$$\rightarrow 0 = 80 \text{ seconds} / s$$

No amount of multiplication speed improvement can achieve this.

Extending Amdahl's Law To Multiple Enhancements

n enhancements each affecting a different portion of execution time

- Suppose that enhancement E_i accelerates a fraction F_i of the original execution time by a factor S_i and the remainder of the time is unaffected then:

$i = 1, 2, \dots, n$

$$\text{Speedup} = \frac{\text{Original Execution Time}}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i} \right) \times \text{Original Execution Time}}$$

Unaffected fraction \nearrow

$$\text{Speedup} = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i} \right)}$$

What if the fractions given are after the enhancements were applied?
How would you solve the problem?
(i.e find expression for speedup)

Note: All fractions F_i refer to original execution time before the enhancements are applied.

Amdahl's Law With Multiple Enhancements: Example

- Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

$$\text{Speedup}_1 = S_1 = 10 \quad \text{Percentage}_1 = F_1 = 20\%$$

$$\text{Speedup}_2 = S_2 = 15 \quad \text{Percentage}_1 = F_2 = 15\%$$

$$\text{Speedup}_3 = S_3 = 30 \quad \text{Percentage}_1 = F_3 = 10\%$$

- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.
- What is the resulting overall speedup?

$$\text{Speedup} = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i} \right)}$$

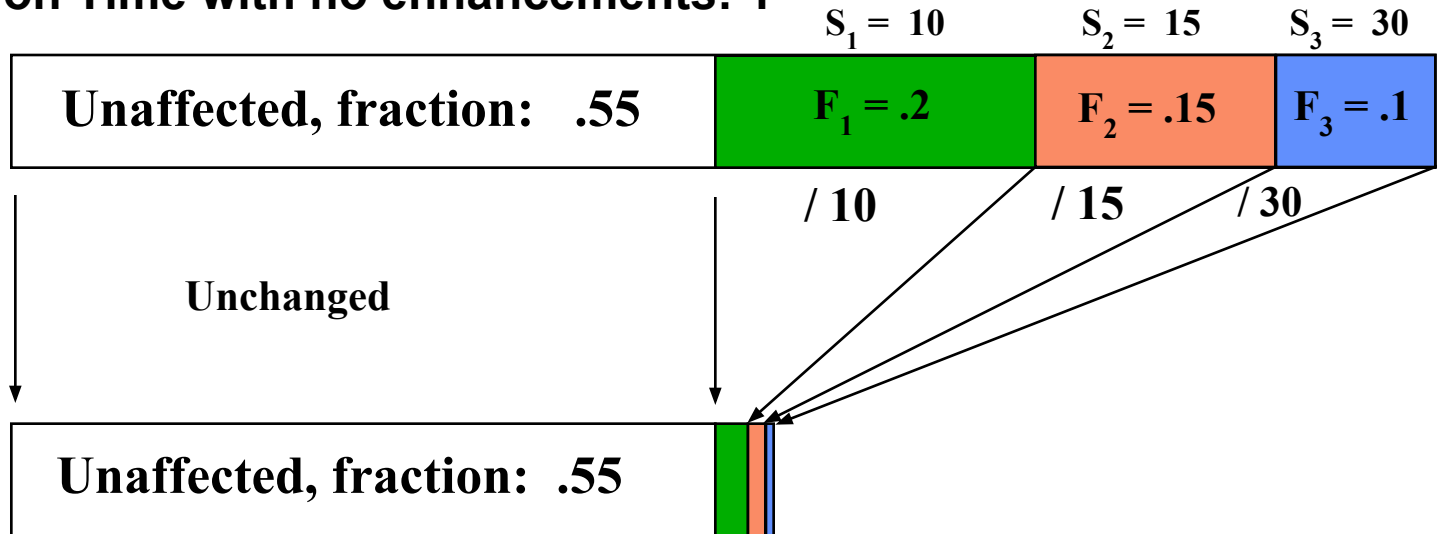
$$\begin{aligned} \text{Speedup} &= 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30] \\ &= 1 / [\quad .55 \quad + \quad .0333 \quad] \\ &= 1 / .5833 = 1.71 \end{aligned}$$

Pictorial Depiction of Example

Before:

Execution Time with no enhancements: 1

i.e normalized to 1



After:

Execution Time with enhancements: $.55 + .02 + .01 + .00333 = .5833$

Speedup = $1 / .5833 = 1.71$

What if the fractions given are after the enhancements were applied?
How would you solve the problem?

Note: All fractions F_i refer to original execution time.

Pitfall: Amdahl's Law

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

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

- Example: multiply accounts for 80s/100s
 - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20 \quad \quad \quad \blacksquare \text{ Can't be done!}$$

- Corollary: make the common case fast

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
 - At 100% load: 258W
 - At 50% load: 170W (66%)
 - At 10% load: 121W (47%)
- Google data center
 - Mostly operates at 10% – 50% load
 - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

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

Concluding Remarks

- Cost/performance is improving
 - Due to underlying technology development
- Hierarchical layers of abstraction
 - In both hardware and software
- Instruction set architecture
 - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
 - Use parallelism to improve performance