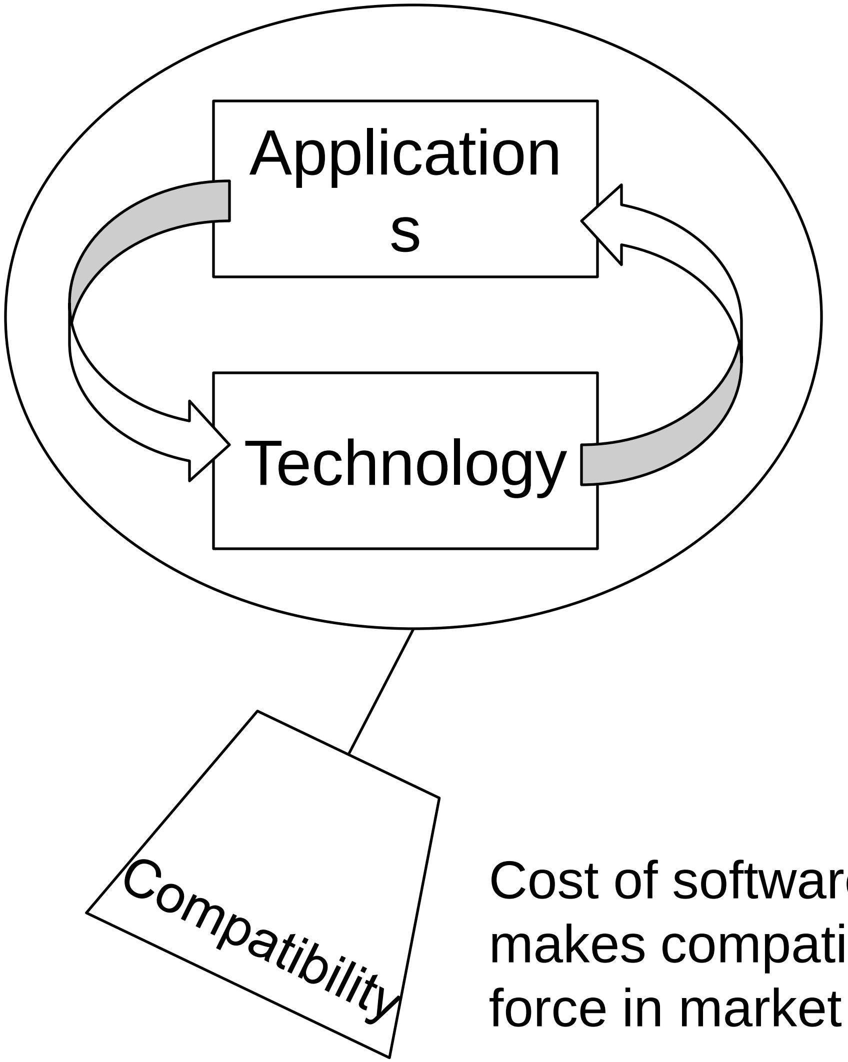
Measuring & Reporting Performance

Architecture continually changing

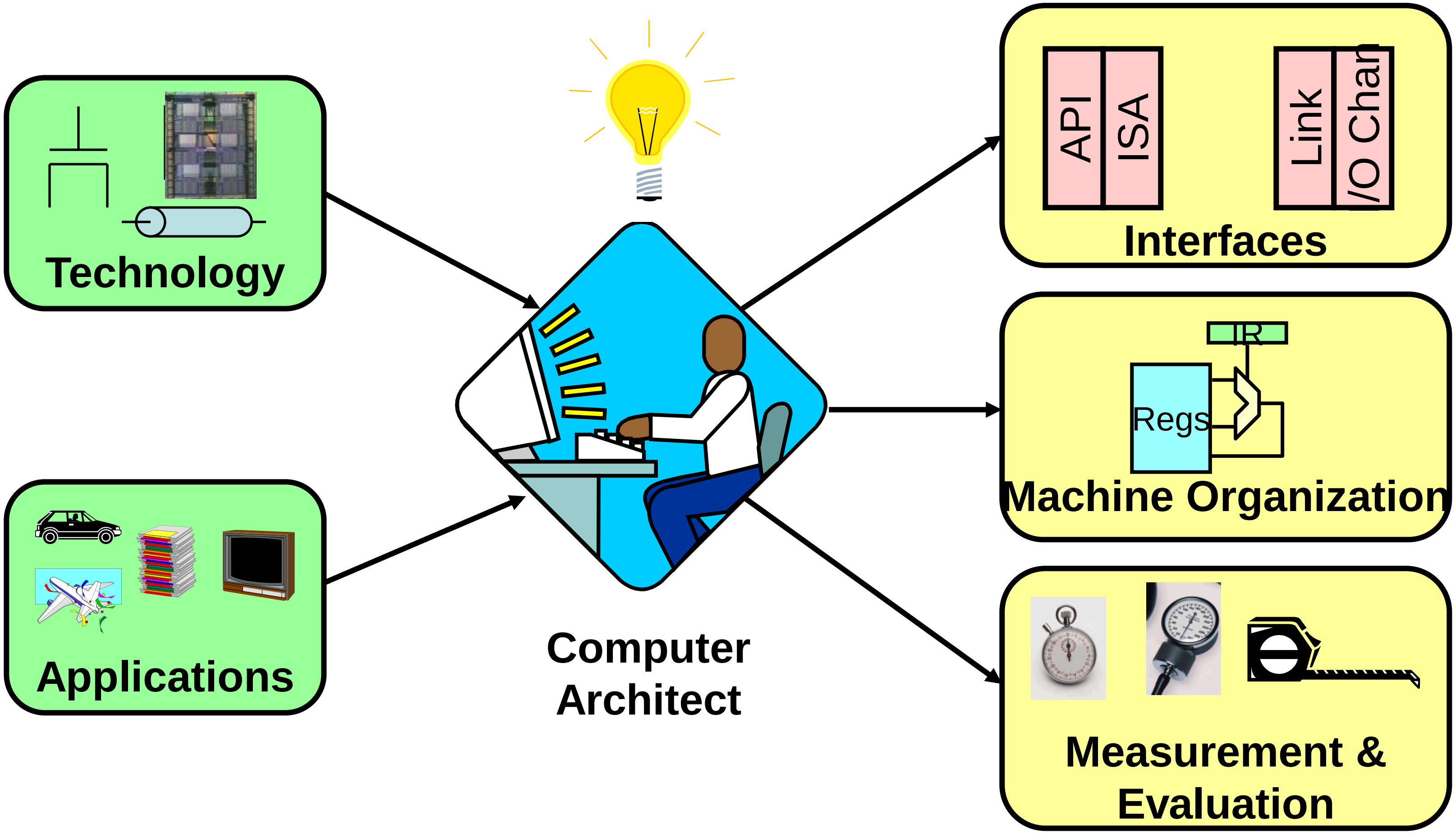
Applications suggest how to improve technology, provide revenue to fund development



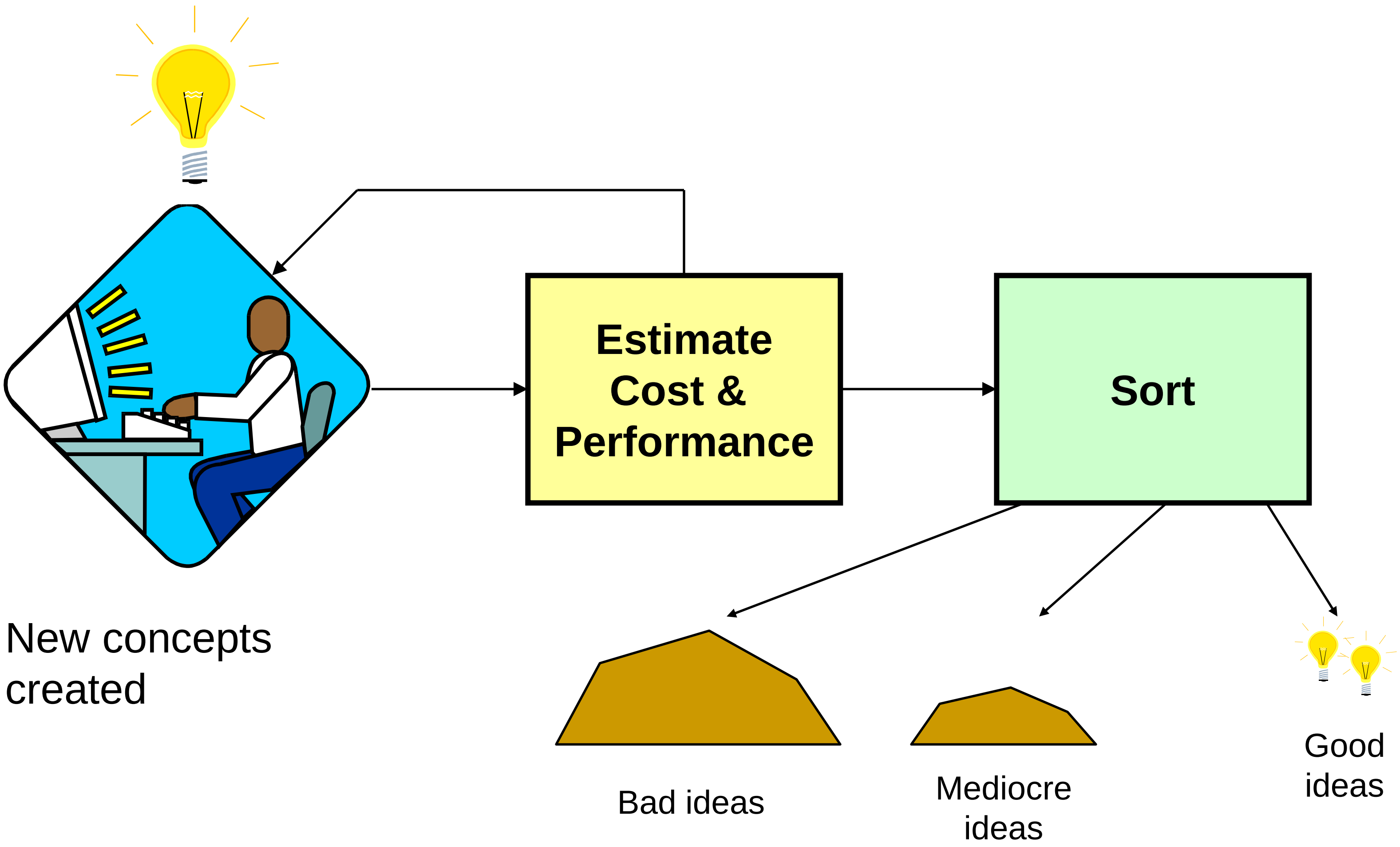
Improved technologies make new applications possible

Cost of software development makes compatibility a major force in market

Review: What is Computer Architecture?



The Architecture Process



What is Performance?

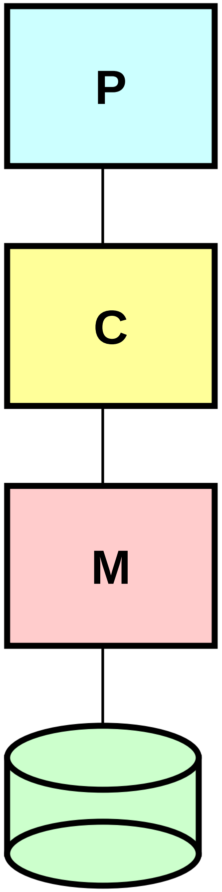
- How can we make intelligent choices about computers?
- Why is some computer hardware performs better at some programs, but performs less at other programs?
- How do we measure the performance of a computer?
- What factors are hardware related? software related?
- How does machine's instruction set affect performance?
- Understanding performance is key to understanding underlying organizational motivation

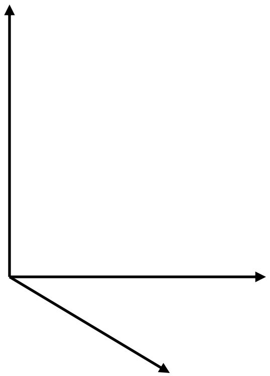
Measuring performance

- **We need measures**
  - **Comparison of machine properties**
  - **Comparison of software properties (compilers)**
- **Purpose**
  - **Making purchase decisions**
  - **Development of new architectures**
- **Is a single measure sufficient?**
  - **A machine with 600 MHz clock cycle is faster than 500 MHz clock cycle!?**
  - **Why do we still have mainframes?**

Performance Measurement and Evaluation

- Many dimensions to computer performance
  - CPU execution time
    - by instruction or sequence
      - floating point
      - integer
      - branch performance
  - Cache bandwidth
  - Main memory bandwidth
  - I/O performance
    - bandwidth
    - seeks
    - pixels or polygons per second
- Relative importance depends on applications

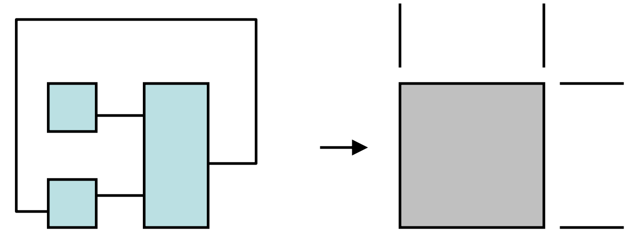


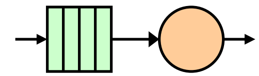


Evaluation Tools

- Benchmarks, traces, & mixes
  - macrobenchmarks & suites
    - application execution time
  - microbenchmarks
    - measure one aspect of performance
  - traces
    - replay recorded accesses
      - cache, branch, register
- Simulation at many levels
  - ISA, cycle accurate, RTL, gate, circuit
    - trade fidelity for simulation rate
- Area and delay estimation
- Analysis
  - e.g., queuing theory
  - Fundamentals Laws

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





Benchmarks and Benchmarking

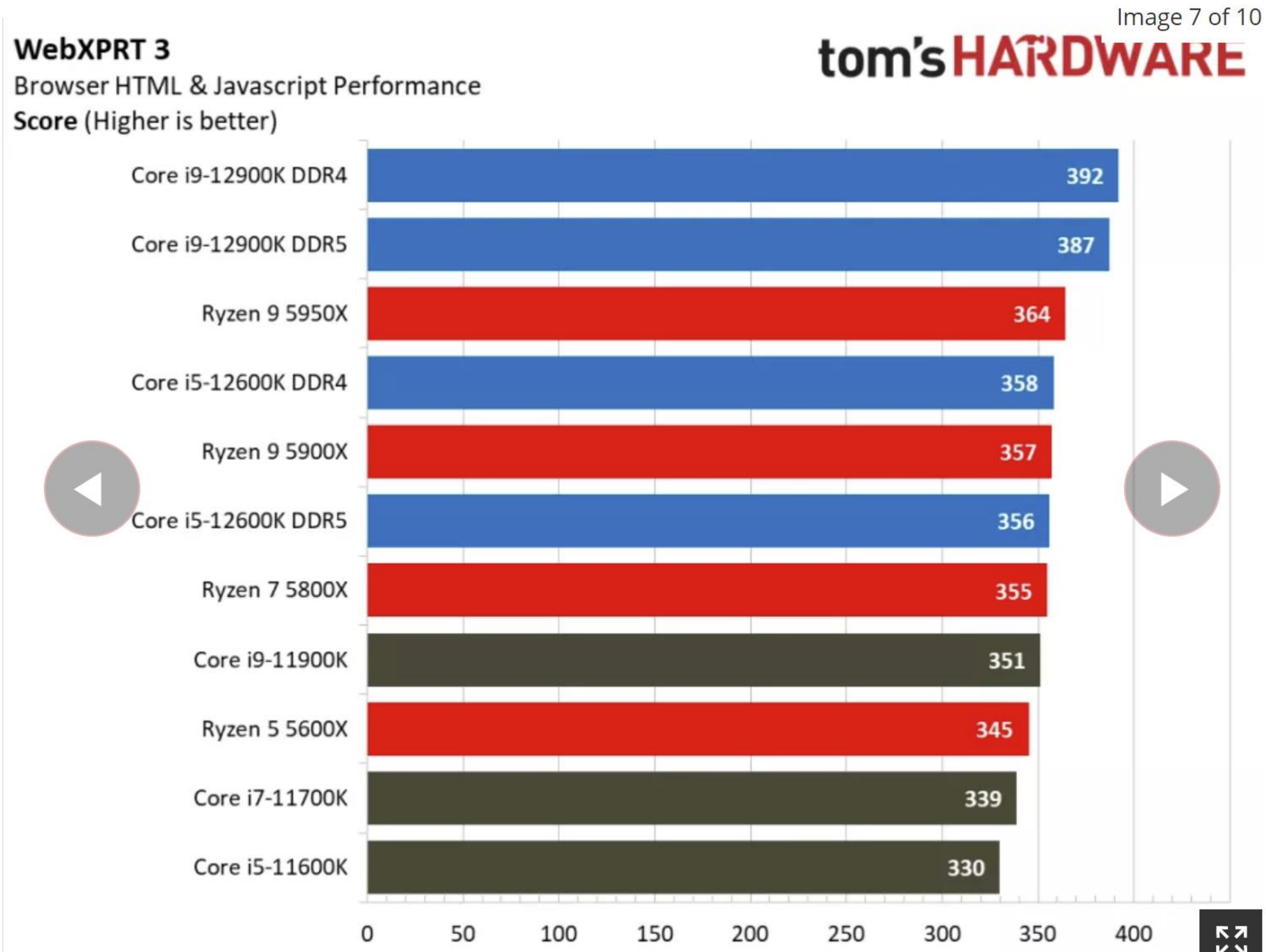
- It is a test that measures the performance of a system or subsystem on a well-defined task or set of task.
- A method of comparing the performance of different computer architecture.
- Or a method of comparing the performance of different software\
- Some Warnings about Benchmarks
  - Benchmarks measure the whole *system*
    - application
    - compiler
    - operating system
    - architecture
    - implementation
  - Popular benchmarks typically reflect yesterday's programms
    - computers need to be designed for tomorrow's programs
  - Benchmark timings often very sensitive to
    - alignment in cache
    - location of data on disk
    - values of data
  - Benchmarks can lead to *inbreeding* or positive feedback
    - if you make an operation fast (slow) it will be used more (less) often
      - so you make it faster (slower)
        - and it gets used even more (less)
          - » and so on…

SPEC: System Performance Evaluation Cooperative

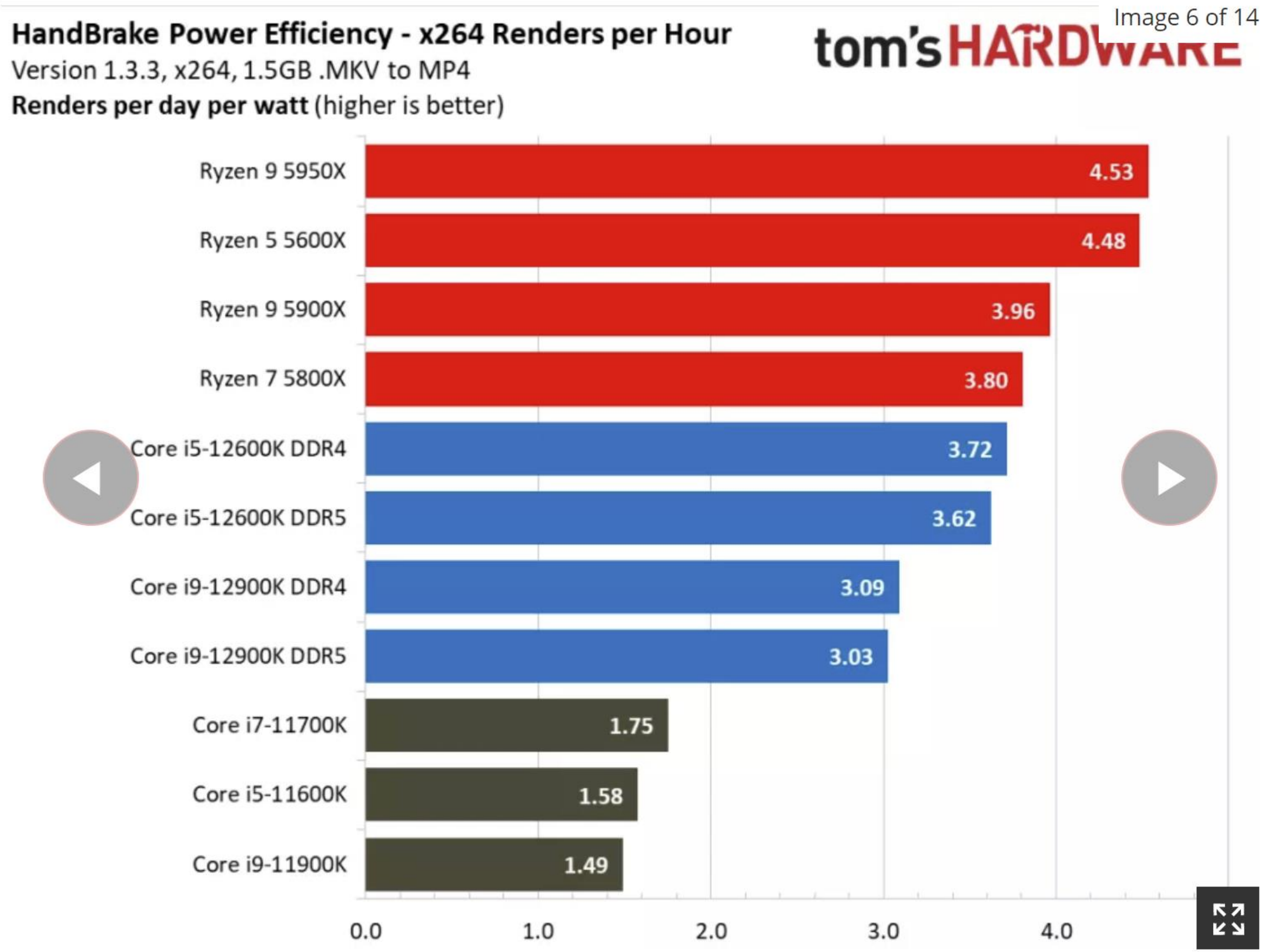
The most popular and industry-standard set of CPU benchmarks.

- SPECmarks, 1989:
  - **10 programs yielding a single number ("SPECmarks").**
- SPEC92, 1992:
  - **SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs).**
- SPEC95, 1995:
  - **SPECint95 (8 integer programs):**
    - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
  - **SPECfp95 (10 floating-point intensive programs):**
    - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
  - **Performance relative to a Sun SuperSpark I (50 MHz) which is given a score of SPECint95 = SPECfp95 = 1**
- SPEC CPU2000, 1999:
  - **CINT2000 (11 integer programs). CFP2000 (14 floating-point intensive programs)**
  - **Performance relative to a Sun Ultra5_10 (300 MHz) which is given a score of SPECint2000 = SPECfp2000 = 100**

Application Performance: Intel Core i9-12900K vs Ryzen 9 5950X and Ryzen 9 5900X



Power Consumption, Efficiency, and Cooling: Intel Core i9-12900K vs Ryzen 9 5950X and Ryzen 9 5900X



(Image credit: Tom's Hardware)

Architectural Performance Laws and Rules of Thumb

- Measurement and Evaluation
  - Architecture is an iterative process:
    - Searching the space of possible designs
    - Make selections
    - Evaluate the selections made
  - Good measurement tools are required to accurately evaluate the selection.
- Measurement Tools
  - Benchmarks, Traces, Mixes
  - Cost, delay, area, power estimation
  - Simulation (many levels)
    - ISA, RTL, Gate, Circuit
  - Queuing Theory
  - **Rules of Thumb**
  - **Fundamental Laws**

Time as a Measure of Performance

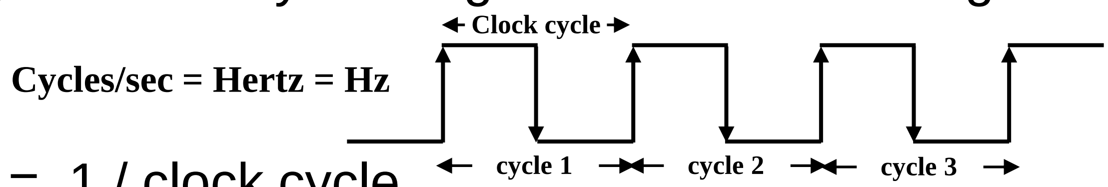
- **Response Time**
  - Time between start and completion of a task
  - As observed and measured by the end user
  - Called also **Wall-Clock Time** or **Elapsed Time**
  - Response Time = CPU Time + Waiting Time (I/O, scheduling, etc.)
- **CPU Execution Time**
  - Time spent executing the program instructions
  - **CPU time = User CPU time + Kernel CPU time**
  - Can be measured in seconds. msec, µsec, etc.
  - Can be related to the **number of CPU clock cycles**
- **Our focus: user CPU time**
  - **Time spent executing the lines of code that are "in" our program**

Throughput as a Performance Metric

- **Throughput = Total work done per unit of time**
  - Tasks per hour
  - Transactions per minute
- Decreasing the execution time improves throughput
  - Example: using a faster version of a processor
  - Less time to run a task $\Rightarrow$ more tasks can be executed per unit of time
- Parallel hardware improves throughput and response time
  - By increasing the number of processors in a multiprocessor
  - More tasks can be executed in parallel
  - Execution time of individual sequential tasks is not changed
  - Less waiting time in queues reduces (improves) response time

CPU Performance Evaluation

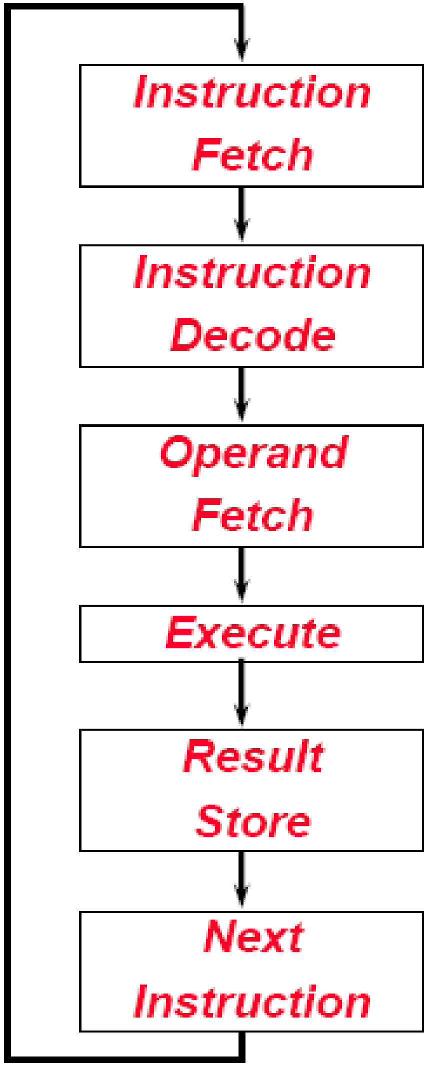
- Most computers run synchronously utilizing a CPU clock running at a constant clock rate:



  where: Clock rate = 1 / clock cycle

- The CPU clock rate depends on the specific CPU organization (design) and hardware implementation technology (VLSI) used
- A computer machine (ISA) 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.
- Thus a single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI).
  – Average CPI of a program: The average CPI of all instructions executed in the program on a given CPU design.

Generic CPU Machine Instruction Execution Steps



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use (if required)

Determine successor instruction

Computer Performance Measures: Program Execution Time

- For <u>a specific program</u> compiled to run on <u>a specific machine (CPU)</u> "A", has the following parameters:
  - The total executed instruction count of the program. $\boxed{I}$
  - The average number of cycles per instruction (average CPI). $\boxed{CPI}$
  - Clock cycle of machine "A" $\boxed{C}$
- 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:

$$\boxed{\textbf{Performance}_A = 1 \;/\; \textbf{Execution Time}_A}$$

How to compare performance of different machines?

What factors affect performance? How to improve performance?

Comparing Computer Performance Using Execution Time

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

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

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

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

- Example: **(i.e Speedup is ratio of performance, no units)**

For a given program:

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

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

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

$= 10 / 1 = 10$

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.

CPU Execution Time: The CPU Equation

- A program is comprised of <u>a number of instructions executed , I</u>
  - 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 <u>C = 1/clock rate</u>
  - 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

Executed

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

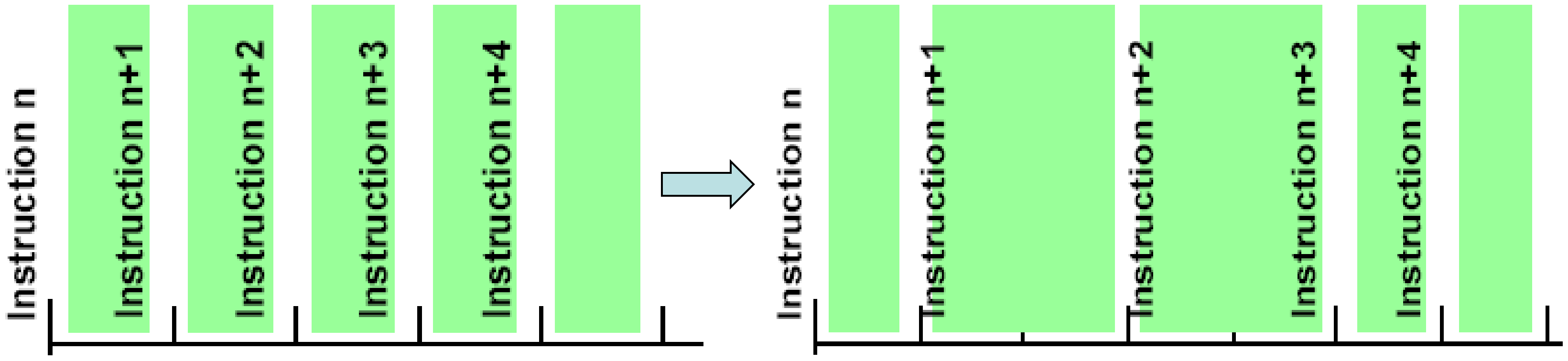
$$T = I \times CPI \times C$$

Execution Time per program in seconds | **Number of instructions executed** | **Average CPI for program** | **CPU Clock Cycle**

(This equation is commonly known as the <u>CPU performance equation</u>)

Instruction = cycle?

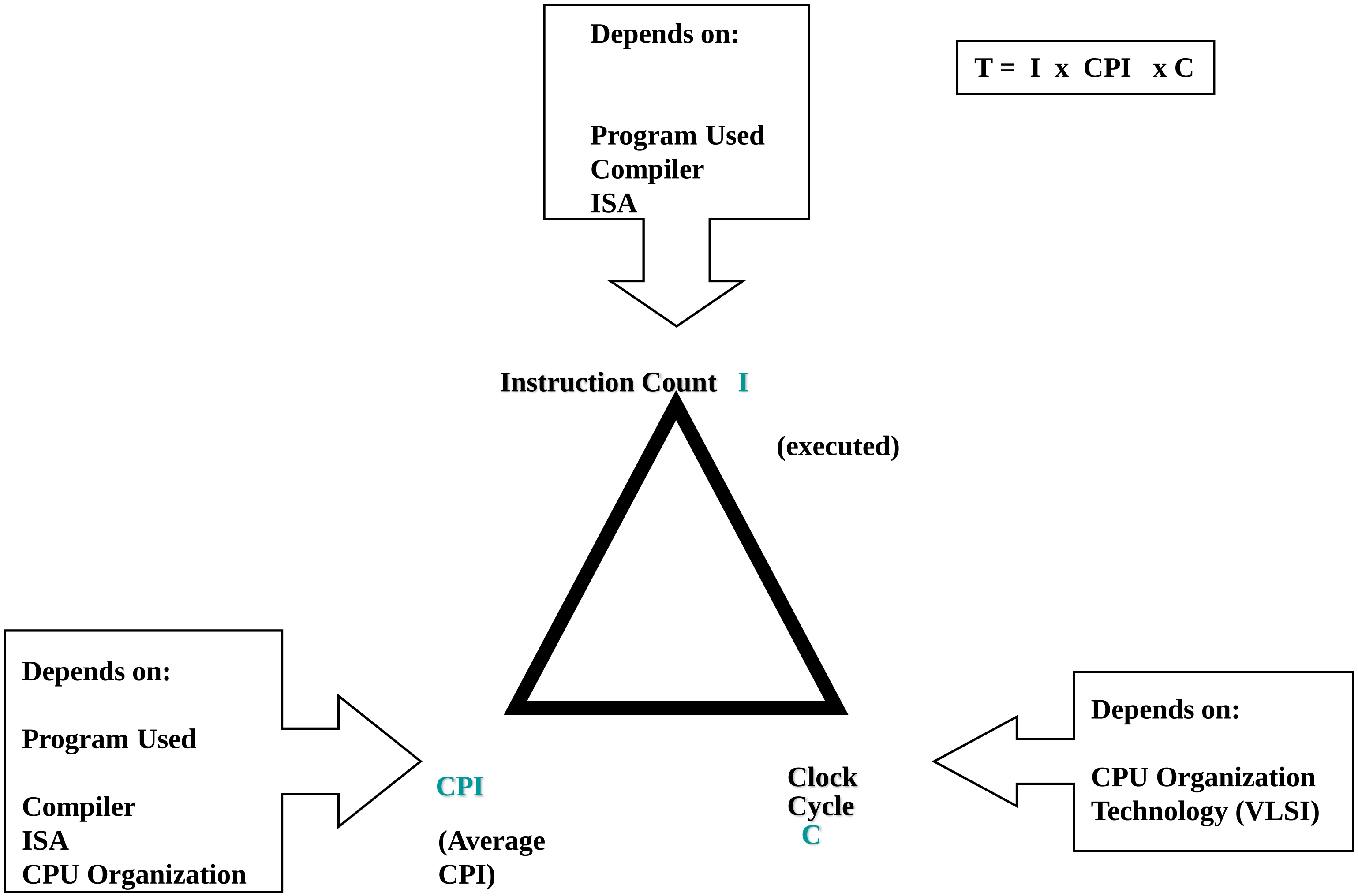
- **Is the number of cycles identical with the number of instructions?**
  - **No!**
    - **The number of cycles depends on the implementation of the operations in hardware**
      - **The number differs for each processor**
  - **Why?**
    - **Operations take different time**
      - **Multiplication takes longer than addition**
      - **Floating point operations take longer than integer operations**
    - **The access time to a register is much shorter than to memory location**



Aspects of CPU Execution Time

CPU Time = Instruction count x CPI x Clock cycle

T = I x CPI x C



Factors Affecting CPU Performance

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

	Instruction Count I	CPI	Clock Cycle C
Program	X	X	
Compiler	X	X	
Instruction Set Architecture (ISA)	X	X	
Organization (CPU Design)		X	X
Technology (VLSI)			X

CPU Execution Time: Example

- A Program is running on a specific machine (CPU) with the following parameters:
  - Total executed instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz. (clock cycle = 5×10^{-9} seconds)
- What is the execution time for this program:

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

CPU time = Instruction count x CPI x Clock cycle

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= .125 seconds

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

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.
- 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?

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

Speedup = (10,000,000 x 2.5 x 5×10^{-9}) / (9,500,000 x 3 x 3.33×10^{-9})

= .125 / .095 = 1.32

or 32 % faster after changes.

Clock Cycle = 1/ Clock Rate

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

Instruction Types & CPI

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

i = 1, 2, …. n

C_i = Count of instructions of type_i executed

CPI_i = Cycles per instruction for type_i

Then:

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

Where:

$$CPU\,clockcycles = \sum_{i=1}^{n} \left(CPI_i \times C_i \right)$$

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 x 1 + 1 x 2 + 2 x 3 = 10 cycles

  CPI for sequence 1 = clock cycles / instruction count

  = 10 /5 = 2

- CPU cycles for sequence 2 = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles

  CPI for sequence 2 = 9 / 6 = 1.5

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

CPI = CPU Cycles / I

Instruction Frequency & CPI

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

i = 1, 2, …. n

C_i = Count of instructions of type_i

CPI_i = Average cycles per instruction of type_i

F_i = Frequency or fraction of instruction type_i executed

= C_i/ total executed instruction count = C_i/ I

Then:

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

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

Instruction Type Frequency & CPI: A RISC Example

Program Profile or Executed Instructions Mix

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

Base Machine (Reg / Reg)

Given

Op	Freq, F_i	CPI_i	$CPI_i \times F_i$	% Time
ALU	50%	1	.5	23% = .5/2.2
Load	20%	5	1.0	45% = 1/2.2
Store	10%	3	.3	14% = .3/2.2
Branch	20%	2	.4	18% = .4/2.2

Typical Mix

Sum = 2.2

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

CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2
= .5 + 1 + .3 + .4

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:

MIPS Rating = Instruction count / (Execution Time x 10^6)
= Instruction count / (CPU clocks x Cycle time x 10^6)
= (Instruction count x Clock rate) / (Instruction count x CPI x 10^6)
= Clock rate / (CPI x 10^6)

- 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/CPUs 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.

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 <u>only valid</u> to compare the performance of different CPUs <u>provided that the following conditions are satisfied</u>:

  1 <u>The same program is used</u>

  (actually this applies to all performance metrics)

  2 <u>The same ISA is used</u>

  3 <u>The same compiler is used</u>

  $\Rightarrow$ (Thus the resulting programs used to run on the CPUs and obtain the MIPS rating are <u>identical</u> at the machine code level including the <u>same instruction count</u>)

Wrong!!!

- 3 significant problems with using MIPS:
  - Problem 1:
    - MIPS is instruction set dependent.
    - (And different computer brands usually have different instruction sets)
  - Problem 2:
    - MIPS varies between programs on the same computer
  - Problem 3:
    - MIPS can vary inversely to performance!
- Let’s look at an examples of why MIPS doesn’t work…

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)

MIPS = Clock rate / (CPI x 10^6) = 100 MHz / (CPI x 10^6)

CPI = CPU execution cycles / Instructions count

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

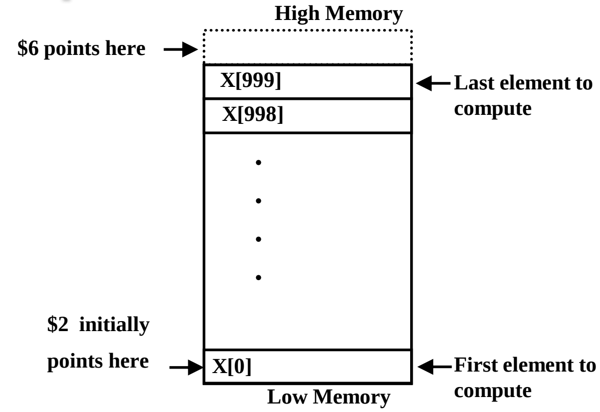
CPU time = Instruction count x CPI / Clock rate

- **For compiler 1:**
  - CPI_1 = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.43
  - MIPS $Rating_1$ = 100 / (1.428 x 10^6) = 70.0 MIPS
  - CPU $time_1$ = ((5 + 1 + 1) x 10^6 x 1.43) / (100 x 10^6) = 0.10 seconds

- **For compiler 2:**
  - CPI_2 = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25
  - MIPS $Rating_2$ = 100 / (1.25 x 10^6) = 80.0 MIPS
  - CPU $time_2$ = ((10 + 1 + 1) x 10^6 x 1.25) / (100 x 10^6) = 0.15 seconds

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

MIPS (The ISA not the metric) Loop Performance Example



For the loop:

for (i=0; i<1000; i=i+1){
    x[i] = x[i] + s; }

MIPS assembly code is given by:

```
        lw      $3,   8($1)       ; load s in $3
        addi    $6,   $2,   4000  ; $6 = address of last element + 4
loop:   lw      $4,   0($2)       ; load x[i] in $4
        add     $5,   $4, $3      ; $5 has x[i] + s
        sw      $5,   0($2)       ; store computed x[i]
        addi    $2,   $2,   4     ; increment $2 to point to next x[ ] element
        bne     $6,   $2,   loop  ; last loop iteration reached?
```

The MIPS code is executed on a specific CPU that runs at 500 MHz (clock cycle = 2ns = 2×10^{-9} seconds) with following instruction type CPIs :

Instruction type	CPI
ALU	4
Load	5
Store	7
Branch	3

For this MIPS code running on this CPU find:

1- Fraction of total instructions executed for each instruction type
2- Total number of CPU cycles
3- Average CPI
4- Fraction of total execution time for each instructions type
5- Execution time
6- MIPS rating , peak MIPS rating for this CPU

X[ ] array of words in memory, base address in $2 ,
s a constant word value in memory, address in $1

MIPS (The ISA) Loop Performance Example (continued)

- The code has 2 instructions before the loop and 5 instructions in the body of the loop which iterates 1000 times,
- Thus: Total instructions executed, I = 5x1000 + 2 = 5002 instructions

1 Number of instructions executed/fraction F_i for each instruction type:

- ALU instructions = 1 + 2x1000 = 2001 CPI_{ALU} = 4 $Fraction_{ALU}$ = F_{ALU} = 2001/5002 = 0.4 = 40%
- Load instructions = 1 + 1x1000 = 1001 CPI_{Load} = 5 $Fraction_{Load}$ = F_{Load} = 1001/5002= 0.2 = 20%
- Store instructions = 1000 CPI_{Store} = 7 $Fraction_{Store}$ = F_{Store} = 1000/5002 = 0.2 = 20%
- Branch instructions = 1000 CPI_{Branch} = 3 $Fraction_{Branch}$= F_{Branch} = 1000/5002= 0.2 = 20%

2 $$CPU\ clock\ cycles = \sum_{i=1}^{n} \left(CPI_i \times C_i \right)$$

= 2001x4 + 1001x5 + 1000x7 + 1000x3 = 23009 cycles

Instruction type	CPI
ALU	4
Load	5
Store	7
Branch	3

3 Average CPI = CPU clock cycles / I = 23009/5002 = 4.6

4 Fraction of execution time for each instruction type:

- Fraction of time for ALU instructions = CPI_{ALU} x F_{ALU} / CPI= 4x0.4/4.6 = 0.348 = 34.8%
- Fraction of time for load instructions = CPI_{load} x F_{load} / CPI= 5x0.2/4.6 = 0.217 = 21.7%
- Fraction of time for store instructions = CPI_{store} x F_{store} / CPI= 7x0.2/4.6 = 0.304 = 30.4%
- Fraction of time for branch instructions = CPI_{branch} x F_{branch} / CPI= 3x0.2/4.6 = 0.13 = 13%

5 Execution time = I x CPI x C = CPU cycles x C = 23009 x 2×10^{-9} =

= 4.6×10^{-5} seconds = 0.046 msec = 46 usec

6 MIPS rating = Clock rate / (CPI x 10^6) = 500 / 4.6 = 108.7 MIPS

- The CPU achieves its peak MIPS rating when executing a program that only has instructions of the type with the lowest CPI. In this case branches with CPI_{Branch} = 3
- Peak MIPS rating = Clock rate / (CPI_{Branch} x 10^6) = 500/3 = 166.67 MIPS

Quantitative Principles of Computer Design

- Amdahl's Law:
  - The performance gain from improving some portion of a computer is calculated by:

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

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

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:

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

Hence speedup is given by:

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

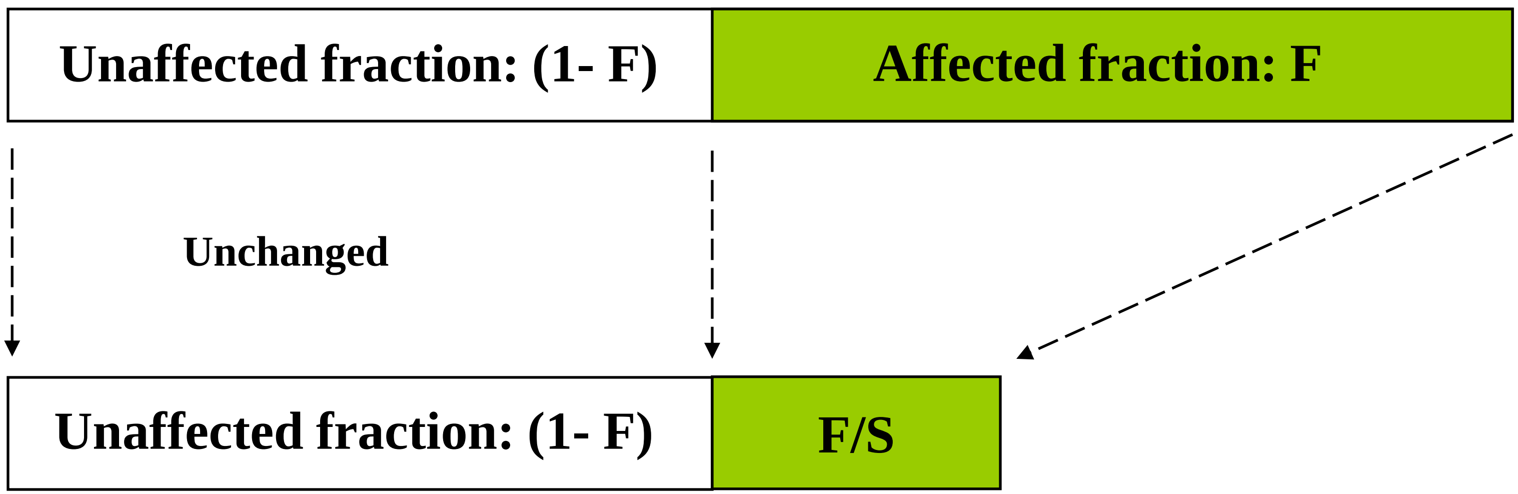
F (Fraction of execution time enhanced) refers to original execution time before the enhancement is applied

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:

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

Example of Amdahl's Law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$$\mathbf{ExTime_{new} = ExTime_{old} \times (0.9 + .1/2) = 0.95 \times ExTime_{old}}$$

$$\mathbf{Speedup_{overall} = \frac{1}{0.95} = 1.053}$$

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%

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 = .5 x 1 + .2 x 2 + .1 x 3 + .2 x 2 = 1.6

$$\text{Speedup(E)} = \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\cancel{\text{Instruction count}} \times \text{old CPI} \times \cancel{\text{clock cycle}}}{\cancel{\text{Instruction count}} \times \text{new CPI} \times \cancel{\text{clock cycle}}}$$

$$= \frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37$$

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

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?

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

$\rightarrow$ Execution time with enhancement = 100/4 = 25 seconds

25 seconds = (100 - 80 seconds) + 80 seconds / S

25 seconds = 20 seconds + 80 seconds / S

$\rightarrow$ 5 = 80 seconds / S

$\rightarrow$ S = 80/5 = 16

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

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

Solving for S gives S= 16

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

and then solving Amdahl's speedup equation for desired enhancement factor S Hence multiplication should be 16 times 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?

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

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

20 seconds = (100 - 80 seconds) + 80 seconds / s

20 seconds = 20 seconds + 80 seconds / s

$\rightarrow$ 0 = 80 seconds / s

No amount of multiplication speed improvement can achieve this.

Extending Amdahl's Law To Multiple Enhancements

- 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:

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

Unaffected fraction

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

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:

  Speedup$_1$ = S_1 = 10 Percentage$_1$ = F_1 = 20%
  Speedup$_2$ = S_2 = 15 Percentage$_1$ = F_2 = 15%
  Speedup$_3$ = S_3 = 30 Percentage$_1$ = F_3 = 10%

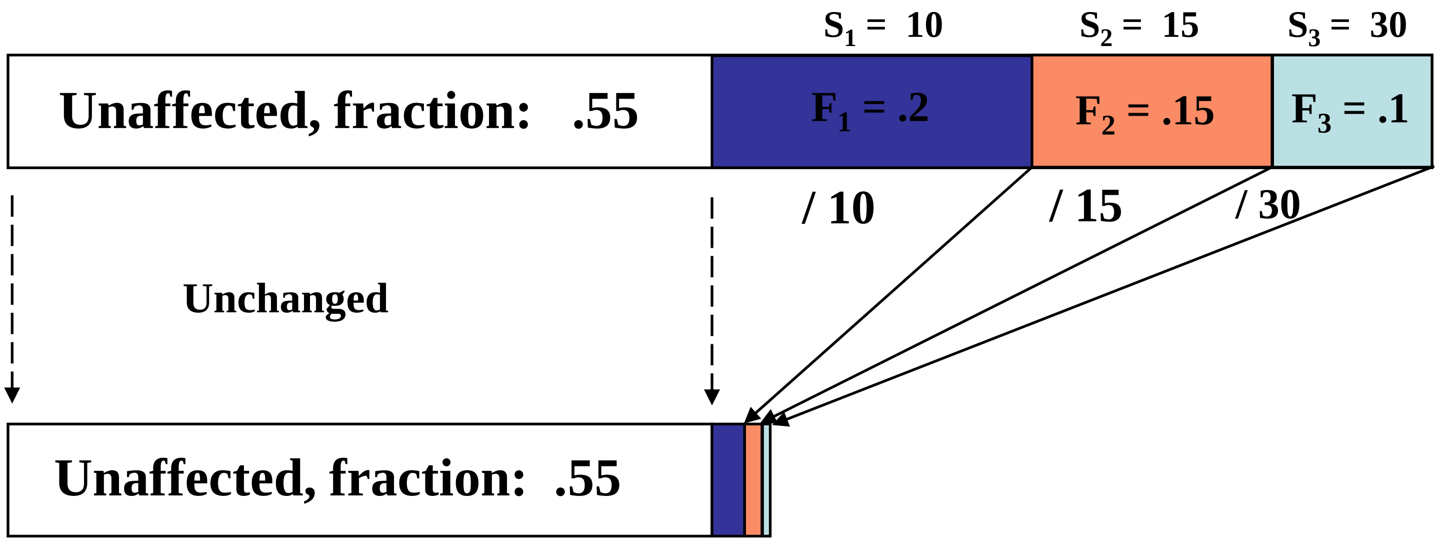
- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.
- What is the resulting overall speedup?

$$\boldsymbol{Speedup} = \frac{1}{\left((1 - \sum_i \boldsymbol{F}_i) + \sum_i \frac{\boldsymbol{F}_i}{\boldsymbol{S}_i}\right)}$$

- Speedup = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30)]
  = 1 / [.55 + .0333]
  = 1 / .5833 = 1.71

Pictorial Depiction of Example

Before:
Execution Time with no enhancements: 1



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

Speedup = 1 / .5833 = 1.71

Note: All fractions refer to original execution time.

Example

You are going to enhance a machine and there are two types of possible improvements: either

- make multiply instructions run 4 times faster, or
- make memory access instructions run two times faster than before.

You repeatedly run a program that takes 100 seconds to execute (on the original machine) and find that of this time 25% is used for multiplication, 50% for memory access instructions, and 25% for other tasks.

Example

1. What will the speedup be if you improve both multiplication and memory access?
2. Assume the program you run has 10 billions instructions and runs on the machine that has a clock rate of 1GHz. Calculate the CPI for this machine. Assume further that the CPI for multiplication instructions is 20 cycles and the CPI for memory access instructions is 6 cycles. Compute the CPI for all other instructions.
3. What is the CPI for the improved machine when improvements on both multiplication and memory access instructions are made?