# Module 3: Processor Design. Performance

# 3.3 Performance

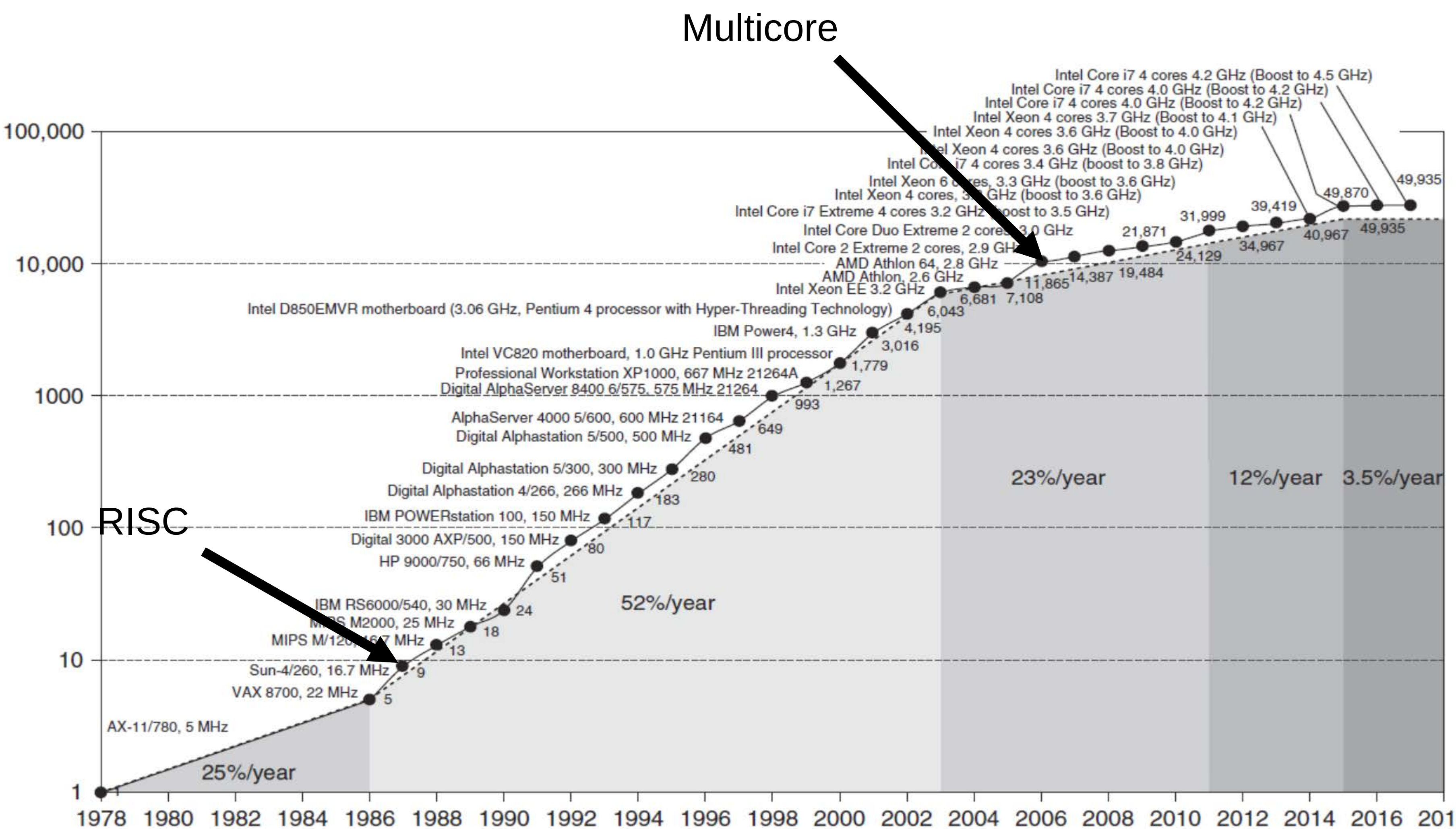
1. Performance in processors
2. Performance figures
3. Amdahl law

- **References**
  - **D. A. Patterson and J.L. Hennessy, Section 1.6**
  - Hennessy Patterson, Chapter 1 or Apendix A

# Defining performance

- Performance = speed.
- Understanding how to measure it and the limitations of performance measures is important.
- What do we mean when we say that a computer has better performance tan another?
  - **Response time**: time between the start and end of the task.
  - **Throughput**: total amount of work done in a given time.
- We will be primarily concerned with response time. Even so, we can measure:
  - **Wall clock time**
  - **CPU time**: it only counts the cycles when instructions from the task are being executed.
- We will be measuring CPU time (seconds per program)
- Which program?

# Processor Performance growth




From: section 1.1 of Hennessy 5th ed.

# Processor performance

- **How many cycles takes a processor to execute this program**
  - **It depends on the processor design**: for instance, for the multicycle MIPS

**lw r1, 0(r0) → 5**

**lw r2, 4(r0) → 5**

**add r3, r1, r2 → 4**

**beq r3, r5, 1 → 4**

**sub r3, r3, r5 → 4**

**sw r3, 8(r0) → 4**

- **And how much time is that?**
  - It depends on the frequency

# Performance measurements

To **compare different processors** we need to establish a performance figure

- **Two magnitudes of interest:**
  - **Execution time:** time to execute a task
  - **Throughput**: number of tasks executed in one second

The user determines which is more interesting

- **Standarized patterns or benchmarks**. These are programs used by the community as patterns to evaluate the processor performance. There are different approaches to obtain this benchmarks, the most extended are:
  - Using kernels from real programs: SPEC
  - Synthetic programs: TPC

# Performance Measurements: Execution time

- Execution time:
  - **Response time:** time to complete a task (time perceived by the user).
  - ***CPU time:*** time to execute a program, removing the time needed for I/O operations and the time used executing other programs. Composed of:
    - ***User CPU time:*** time used by the CPU executing instructions of the user program
    - ***System CPU time:*** time devoted to execute O.S. code in the context of the program, used to obtain services from the O.S. or to perform any O.S. related task.
  - The Unix ***time*** utility allows to run a program and measure these values. An example output could be: 90.7u 12.9s 2:39 65%, where:
    - User CPU time = 90.7 s
    - System CPU time = 12.9 s
    - CPU time = 90.7 s + 12.9 s = 103.6 s
    - (Example) Response time = 2 minutes 39 s =159 s
    - CPU time = 65% Response time = 159 s*0.65 = 103.6 s
    - Time devoted to I/O or waiting for other tasks: 35% Response time = 159 s *0.35 = 55.6 s

# Performance Measurements: Execution time

Time (seconds) = Number of clock cycles * Clock cycles/second

Number of clock cycles =
Number of instructions * Cycles/instruction

$T = N * CPI * T_C$

Depends on:
ISA
Compiler

Depends on:
ISA
Design/Micro-Architecture

Depends on:
Design/Micro-Architecture
Technology

- **CPI =** clock **C**ycles **P**er **I**nstruction
  - One instruction needs several cycles to complete execution
  - Different instructions may need different number of cycles
  - **CPI** Is the average of cycles needed by the instructions

# Performance measurements: Execution Time

## CPI computation

– By definition CPI is: $$CPI = \frac{Numberofcycles}{Numberofinstructions}$$

– Depending on the architecture different expressions are of interest

- For Multicycle Design

$$CPI = \frac{\sum_{i=1}^{n} CPI_i N_i}{N} = \sum_{i=1}^{n} CPI_i \frac{N_i}{N}$$

  - Where:
  - Ni = number of i-type instructions, with $N = \sum_{i=1}^{n} N_i$
  - CPIi = number of cycles for type-i instructions
- For Pipelined design:

$$CPI = \frac{N + stall\,cycles + fill/drain\,cycles}{N} = 1 + \frac{stalls}{N} + \frac{fill/drain}{N}$$

  - Usually N is much larger than the number of fill/drain cycles and the last term can be neglected.
  - To improve performance we have to reduce the number of stalls

# Example: CPI computation

– By definition CPI is:

$$CPI = \frac{Numberofcycles}{Numberofinstructions}$$

$$\frac{16}{6}$$

– Depending on the architecture different expressions are of interest

- For Pipelined design:

$$CPI = \frac{N + stallcycles + fill/draincycles}{N} = 1 + \frac{stalls}{N} + \frac{fill/drain}{N}$$

$$\frac{6 + (1+3+2) + 4}{6}$$

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | LD F0,0(R1) | IF | ID | E | M | WB | | | | | | | | | | | |
| 2 | LD F2,8(R1) | | IF | ID | E | M | WB | | | | | | | | | | |
| 3 | ADDD F4,F0,F2 | | | IF | Dp | ID | A1 | A2 | A3 | A4 | M | WB | | | | | |
| 4 | SUBD F6,F0,F2 | | | | Fp | IF | Dp | Dp | Dp | ID | A1 | A2 | A3 | A4 | M | WB | |
| 5 | ADDI R1,R1,#8 | | | | | | Fp | Fp | Fp | IF | ID | E | M | WB | | | |
| 6 | SD F6,0(R2) | | | | | | | | | | IF | Dp | Dp | ID | E | M | WB |

# Performance figures: MIPS

**MIPS** (Millions of Instructions Per Second)

$$MIPS = \frac{N}{Execution\,time\,10^6} = \frac{1}{CPI \times 10^6 \times T_C} = \frac{F_C}{CPI \times 10^6}$$

$$Execution\,time = \frac{N}{MIPS \times 10^6}$$

- **Depends on the ISA**, which makes it difficult to use to compare two architectures with different ISA
- Differs from one program to the other on the same computer
- Can change inversely to the performance

# Performance figures: MFLOPS

**MFLOPS** (Millons of Floating Point Operations per second)

$$\text{MFLOPS}=\frac{Number\ of\ floating\ point\ operations}{Execution\ time \times 10^6}$$

- There are fast (e.g. addition) and slow (e.g. division) floating poing operations, thus it can be a not so useful figure

- Some have used normalized MFLOPS that give different weights to different floating point operations.

- Execution time (in seconds)= Number of cycles / Frequency (cycles per second)

# Amdahl's Law

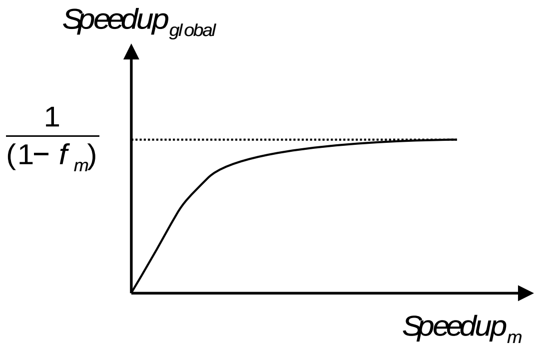
## ***Speedup***: Amdahl's law

***The speedup (improvement in speed) obtained from some modification (m) to the architecture or code is limmited by the fraction of time when it is applicable***

$$Speedup_m = \frac{Time\ without\ modification\ m}{Time\ with\ modification\ m}$$

$$Speedup_{global} = \frac{T_{orig}}{T_m} = \frac{T_{orig}}{(1-f_m)*T_{orig} + f_m \frac{T_{orig}}{speedup_m}} = \frac{1}{(1-f_m) + \frac{f_m}{speedup_m}}$$

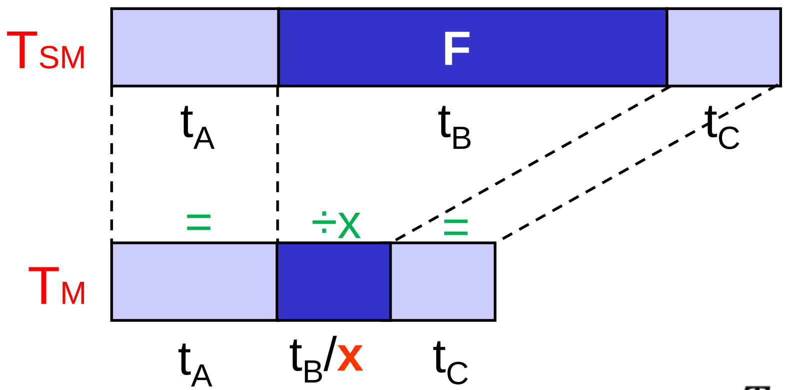
$$\lim_{speedup_m \to \infty} Speedup_{global} = \frac{1}{(1-f_m)}$$



# Amdahl's Law

## ***Speedup***: Amdahl's law

*The speedup (improvement in speed) obtained from some modification (m) to the architecture or code is limmited by the fraction of time when it is applicable*




$$T_{SM} = t_A + t_B + t_C \qquad F = \frac{t_B}{T_{SM}}$$

$$T_M = t_A + t_C + \frac{t_B}{x} =$$

$$= T_{SM}\,(1-F) + T_{SM}\frac{F}{x} = T_{SM}\left[(1-F) + \frac{F}{x}\right]$$

## ***Efficiency***

$$\mathbf{E} = \frac{Speedup}{x} = \frac{\dfrac{1}{(1-F) + \dfrac{F}{x}}}{x} = \frac{1}{x(1-F) + F} = \frac{1}{x + F(1-x)}$$

# Amdahl's Law

## Example

- In one computer system we replace the disc by other 10 times faster.
- The disc is used 40% of the execution time.
- What is the global speedup obtained?

$$Speedup_m = 10 \quad f_m = 0.4$$

$$Speedup_{global} = \frac{1}{(1-0.4)+\frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

Pretty close to the maximum!! looking for a faster disc might not be worth the cost

$$\lim_{Speedup_m \to \infty} Speedup_{global} = \frac{1}{(1-0.4)} = \frac{1}{0.6} = 1.666$$

We cannot go better than this if our disc usage only affects the 40% of the time