Technical University of Cluj-Napoca
Computer Science Department

Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 1: Introduction

https://mihai.utcluj.ro/

Course Objectives

- The lecture classes are **mandatory!**
- Provide the students with the necessary information
  - Understand: ISA, micro-architectures, CPU design methods, memory hierarchy, CPU performance improvement
  - Specification, design and implement CPUs, micro-architectures, data-paths and control units
  - To understand the new tendencies in computer architectures
- Prerequisites: Logic design, Digital System Design, VHDL Prog.

- 2C + 2L – 14 weeks
- **Assessment:**
  - Written examination: E
  - Lab activity: L
  - Homework: H

$$Lab = \frac{L + H}{2}$$

$$Grade = 0.5 \cdot E + 0.5 \cdot Lab$$

Pass Condition

$$E \geq 4.5$$

$$Lab \geq 5.0$$

Course Content

- Introduction
- High Level Synthesis – HLS
- Instruction Set Architecture – ISA
- CPU Design – Single Cycle
- ALU Design
- CPU Design – Multi-Cycle
- CPU Design – Pipeline
- Advanced Pipelining – Static and Dynamic Scheduling of Execution
- Branch Prediction
- Superscalar Architectures
- Memory Hierarchy
- Modern CPU Architectures

Laboratory Objectives

- The laboratory classes and homework are **mandatory!**
- Teach students to operate with the concepts presented during the lectures
- Develop practical skills in machine language programming, design and implementation of micro-architectures using RTL and VHDL
- Design with Xilinx Development Tools and FPGA boards
- Design synthesizable VHDL hardware components → FPGA
- MIPS assembly language, running simple programs on the designed CPU
- Design and implementation (VHDL) of MIPS micro-architectures and testing on FPGA boards
- The homework helps students in improving their problem solving abilities

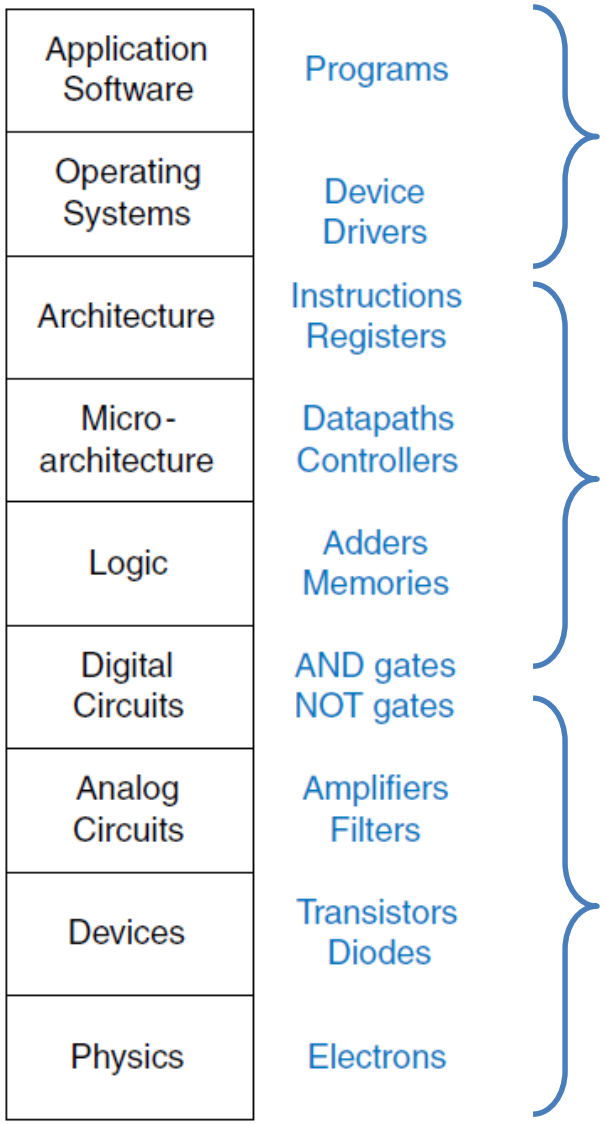
Laboratory Content

- Introduction to Xilinx ISE / **VIVADO** Design Suite
- **VHDL** programming
- Combinational Circuits
- Sequential Circuits
- Memories
- Single Cycle CPU Design
- Pipeline CPU Design
- UART Interface
- I/O Communication
- CPU testing
- CPU presentation

Bibliography

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 3th edition, ed. Morgan–Kaufmann, 2005
2. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013
3. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011
4. D. M. Harris, S. L. Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann, San Francisco, 2007
5. D. A. Patterson, J. L. Hennessy, “ORGANIZAREA SI PROIECTAREA CALCULATOARELOR. INTERFATA HARDWARE/SOFTWARE”, Editura ALL, Romania, ISBN: 973-684-444-7
6. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
7. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.
8. World Wide Web …

Levels of abstraction of a computing system



Applications that run on a computer

Digital Circuits, Logic Gates, Register Transfer Level (RTL), Micro-Architecture

OUR FOCUS IS HERE

Electronic Circuits and Devices

Basic Concepts

- Architecture – the interface between a user and an object
- Computer Architecture
  - Instruction Set Architecture (ISA)
  - Computer Organization – micro-architecture

- ISA: the interface between Hardware and low-level Software
- Micro-architecture: components and connections between them
  - Registers, ALU, Memory, Shifters, Logic Units, …
- The same ISA can have different organizations:
  - MIPS single-cycle, multi-cycle, pipeline
- A specific architecture can be implemented by different micro-architectures with different price/performance/power constraints
- ISA Examples: IA-32, IA-64, MIPS, SPARC, ARM, etc.

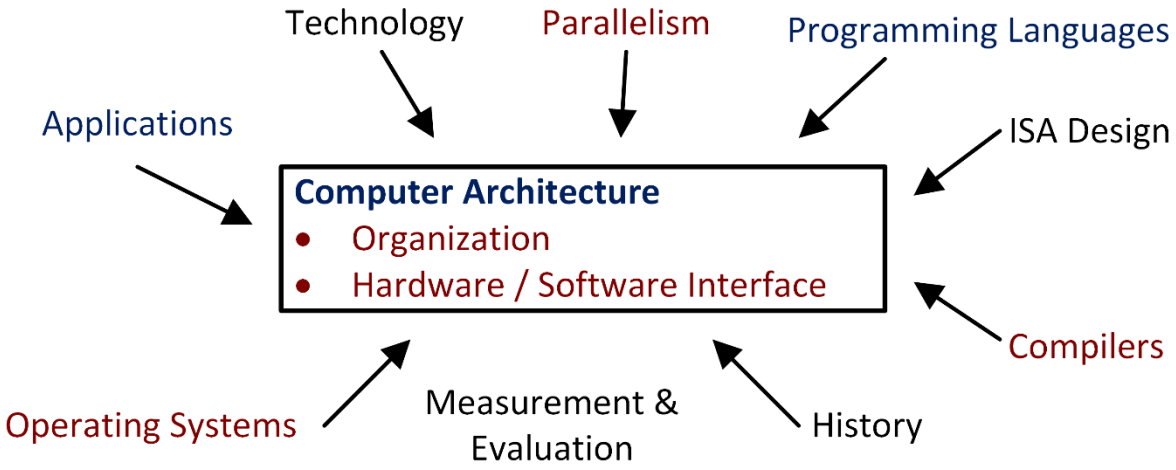
Basic Concepts

- Recommendations to Manage Complexity
  - Hierarchy – dividing the system into modules and sub-modules, until the pieces are easy to understand
  - Modularity – the modules must have well defined functions and interfaces, in order to be easily integrated
  - Regularity – uniformity among modules → reusable modules, in order to reduce the number of modules that must be designed

- A computer architect designs a computer that must fulfill
  - Functional requirements
  - Price/Power/Performance/Availability constraints

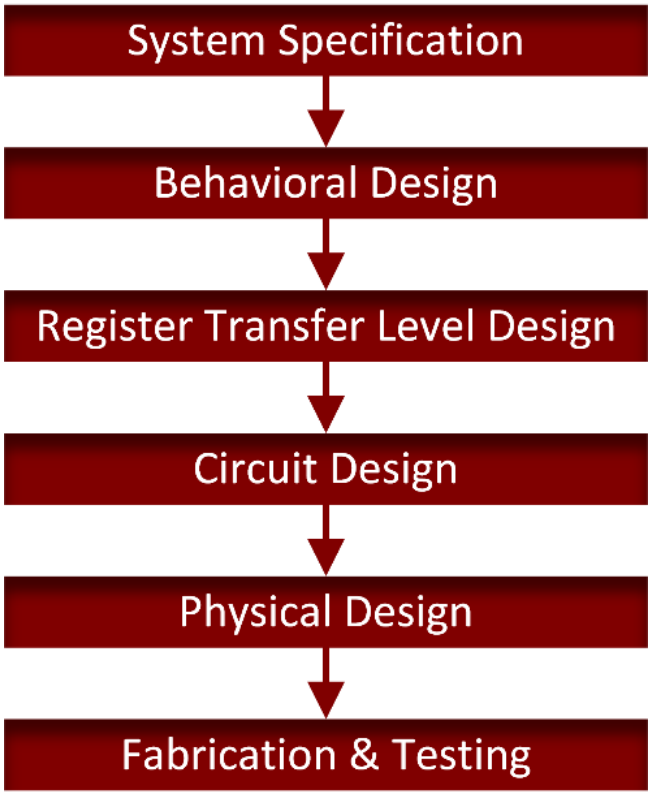
Processor Design Concepts

High Level Synthesis → Logic Synthesis → Layout Synthesis



Factors that influence the design process

- Synthesis is the automatic mapping from a high-level description to a low-level description
- High Level Synthesis or Architectural Synthesis
  - having a description of circuit behavior, create a Register Transfer Level (RTL) architecture that implements the circuit



Design on levels of abstraction
Top-down

Parallelism Types

- 2 types of parallelism (application specific point of view)
  - Data Level Parallelism (DLP) – data that can be processed in the same time
  - Task Level Parallelism (TLP) – independent tasks
- Parallelism classes
  - Instruction Level Parallelism (ILP) – exploits data level parallelism
    - Pipelining, Speculative execution
  - Thread Level Parallelism – exploits DLP & TLP in a hardware model that permits interaction between parallel threads
  - Request Level Parallelism – exploits TLP in de-coupled tasks, specified by the programmer or the OS
- Parallel Architectures
  - Uni-processor systems
  - Multi-processors systems – Multi-Core CPUs
  - Vector Architectures and GPUs – exploits DLP by applying a single instruction to a collection of data, in parallel

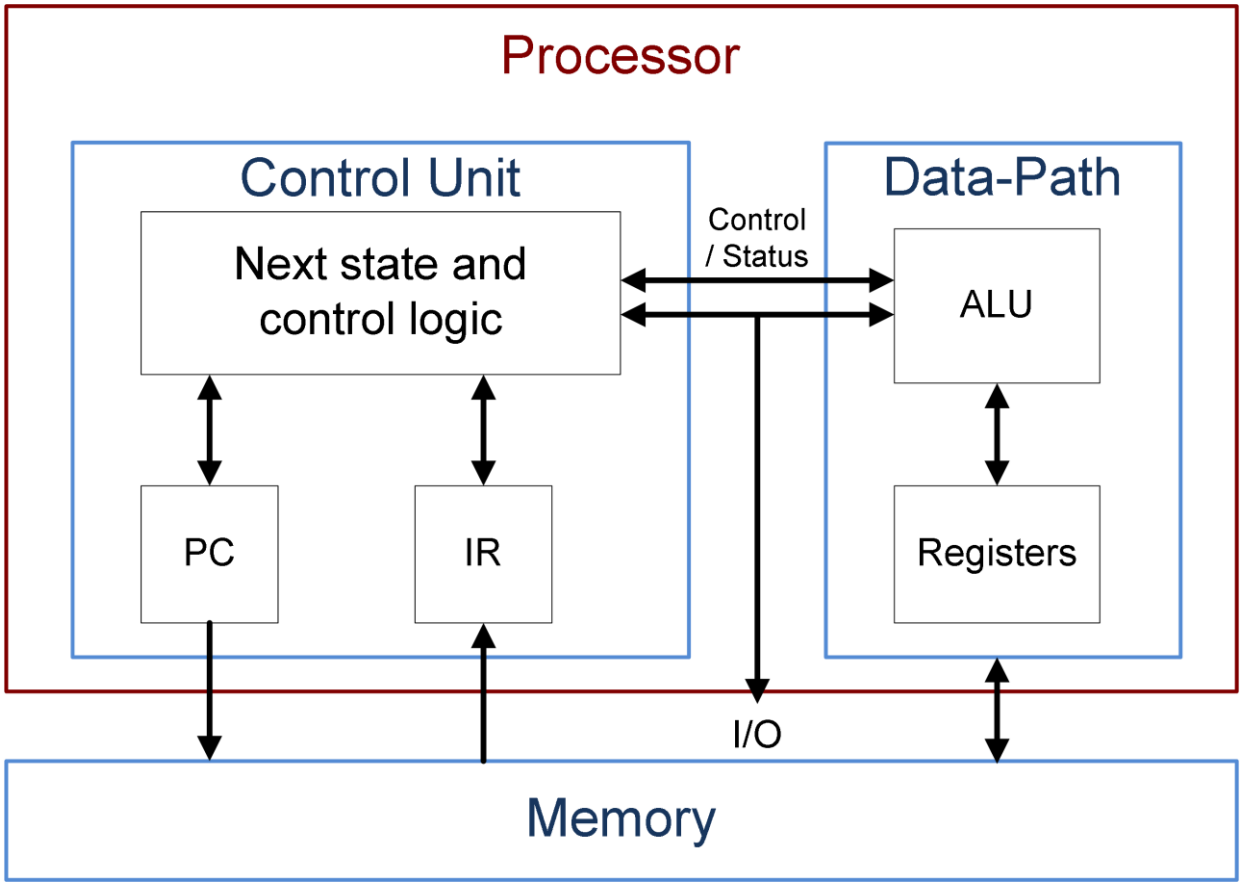
Flynn's Taxonomy

- Simple classification of multi-processing architectures – 1966

Flynn's Taxonomy		
	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

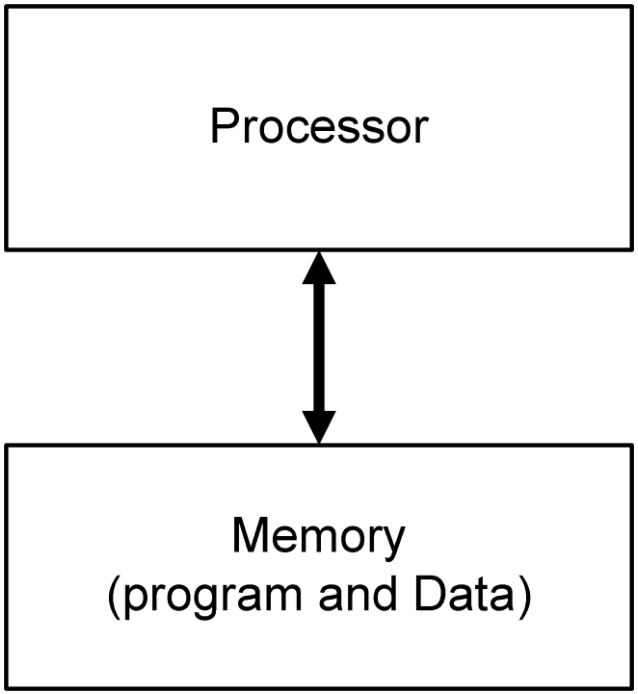
- SISD – Conventional uni-processor systems, can exploit ILP
- SIMD – The same instruction is executed by many processors on different data: Vector Architectures
- MISD – very rare, offers the advantage of redundancy
- MIMD – every processor operates on its own data and instructions, exploits task level parallelism
- A system with N cores is effective when it runs N or more threads concurrently!

General Processor Architecture



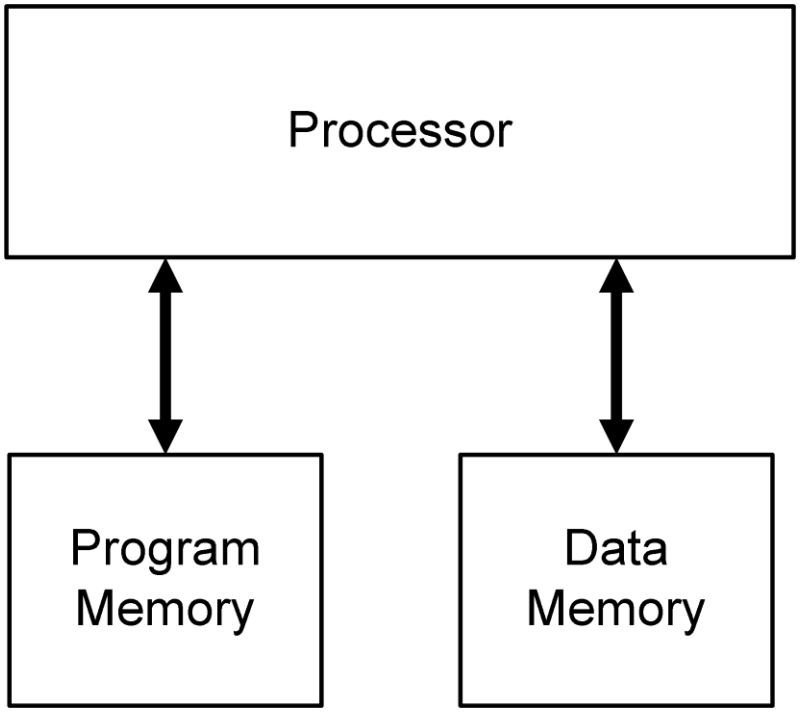
Processor = Data-Path + Control

Uni-processor Classical Architectures



Van Neumann /
Princeton Architecture

A single memory for both Instruction and Data
Stored program computer



Harvard Architecture

Separate memories for Instruction and Data

Computer Architectures

- CPU types
  - Complex Instruction Set Computer (CISC)
    - Complex set of instructions, hard to pipeline, reduced number of registers, ALU operations with memory
    - Memory accesses through many different instructions
    - Many addressing modes
    - Instructions have variable width
  - Reduced Instruction Set Computer (RISC)
    - Reduced set of instructions, easy to pipeline, larger number of registers, ALU operations only with registers
    - Memory accesses only through load / store instructions
    - Reduced number of addressing modes
    - Instructions have fixed width
- Other architectures
  - DSP – digital signal processors
  - Embedded – SoC (system on chip)
  - Reconfigurable – FPGA (field programmable gate arrays)

Instruction Set Architecture, ISA

- The interface between Hardware and low-level Software

- Core ISA elements
  - Memory models (alignment, linear, split address space)
  - Registers (special, general, mixed, kernel), Register model
  - Data types (numeric, non–numeric)
  - Instruction (format, size, types, and set)
  - Operations provided in the instruction set
  - Number of operands for each instruction, type and size of operands
  - Address specification (registers, implicit, ACC, stack)
  - Addressing modes (immediate, direct, register, indexed, stack,…)
  - Flow of Control
  - Input/Output, Interrupts
  - …

Instruction Set Architecture

- ISA design issues
  - Which operation and data types should be supported?
  - Operands: how many, how big?
  - Where do operands reside?
  - How many registers?
  - How important are immediates and how big are they?
  - Which addressing modes dominate usage?
  - How are memory addresses computed?
  - Which control instructions should be supported?
  - How big a branch displacement is needed?
  - How should the instruction format be like, which bits designate what?
  - Instruction length: are all instructions the same length?
  - Can you add contents of memory to a register?
  - …

Instruction Set Architecture

- ISA Classes
  - Most modern ISAs are general purpose register (GPR). ALU operands are registers or memory locations
  - 2 types
    - Register-Memory ISA: x86, x64. ALU operations: reg-reg or reg-mem
    - Register-Register, Load/Store ISA: ARM, MIPS. ALU operations: reg-reg, only Load and Store instructions access memory

- Memory addressing
  - 80x86, ARM, MIPS use byte addressing
  - ARM, MIPS – instructions must be aligned in memory
  - To access an **s-byte** object at address **A** is aligned if **A mod s = 0**
  - 80x86 does not require memory alignment, but the access is faster to aligned operands

ISA – Addressing Modes

Addressing mode	Example Instruction	Meaning
Register	Add R4, R3	Regs[R4] ← Regs[R4] + Regs[R3]
Immediate	Add R4, #3	Regs[R4] ← Regs[R4] + 3
Displacement	Add R4, 100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]
Register Indirect	Add R4, (R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]
Indexed	Add R4, (R1+R2)	Regs[R4] ← Regs[R4] + Mem[Regs[R1] + Regs[R2]]
Direct or Absolute	Add R4, (1001)	Regs[R4] ← Regs[R4] + Mem[1001]
Memory indirect	Add R4, @(R3)	Regs[R4] ← Regs[R4] + Mem[Regs[R3]]
Auto-increment	Add R4, (R3)+	Regs[R4] ← Regs[R4] + Mem[Regs[R3]] Regs[R3] ← Regs[R3] + d (size of element)
Auto-decrement	Add R4, -(R3)	Regs[R3] ← Regs[R3] - d (size of element) Regs[R4] ← Regs[R4] + Mem[Regs[R3]]
Scaled	Add R4, 100(R2)[R3]	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R2] + Regs[R3] * d]

Instruction Set Architecture

- Endianness

3	2	1	0

Bytes in register

Address	0003	0002	0001	0000
Byte #	3	2	1	0

Little Endian
LSB byte at lower address

Address	0003	0002	0001	0000
Byte #	0	1	2	3

Big Endian
MSB byte at lower address

- Type and dimension of operands
  - 80x86, ARM, MIPS support:
    - 8-bit (ASCII character)
    - 16-bit (Unicode character or half word)
    - 32-bit (integer or word)
    - 64-bit (double word or long integer)
    - IEEE 754 floating point: 32-bit (single precision) and 64-bit (double precision)
  - 80x86 also supports 80-bit floating point (extended double precision)

- Instruction operations

Operation type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, sub, and, or, multiply, divide
Data transfer	Load, stores, move instructions (on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating Point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, multiply, decimal to character conversion
String	String move, compare, search
Graphics	Pixel and vertex operations, compression/decompression operations

Instruction Set Architecture

- Flow Control Instructions
  - Conditional jumps, unconditional jumps, procedure calls and returns
  - PC relative addressing: next address is an offset added to the PC
    - MIPS (BEQ, BNE, etc.): test the content of a register
    - 80x86, ARM: test the bits of the FLAG register that are affected by arithmetic / logic operations
  - ARM, MIPS procedure call: sets the return address in a register
  - 80x86 procedure call: sets the return address in memory or stack
- Instruction formats – 2 main types: fixed and variable length
  - ARM, MIPS: 32-bit instructions, simple decoding
  - 80x86: variable length instructions (1 – 18 bytes)
  - Variable length instructions occupy less space
  - The number of registers and used addressing modes influence instruction length
  - ARM, MIPS extensions: 16-bit instructions Thumb and MIPS16

Instruction Set Architecture

- Variable (Intel 80x86, VAX)

Operation and no. of operands	Address specifier 1	Address field 1	.....	Address specifier n	Address field n

- Fixed (Alpha, ARM, MIPS, PowerPC, SPARC)

Operation	Address field 1	Address field 2	Address field 3

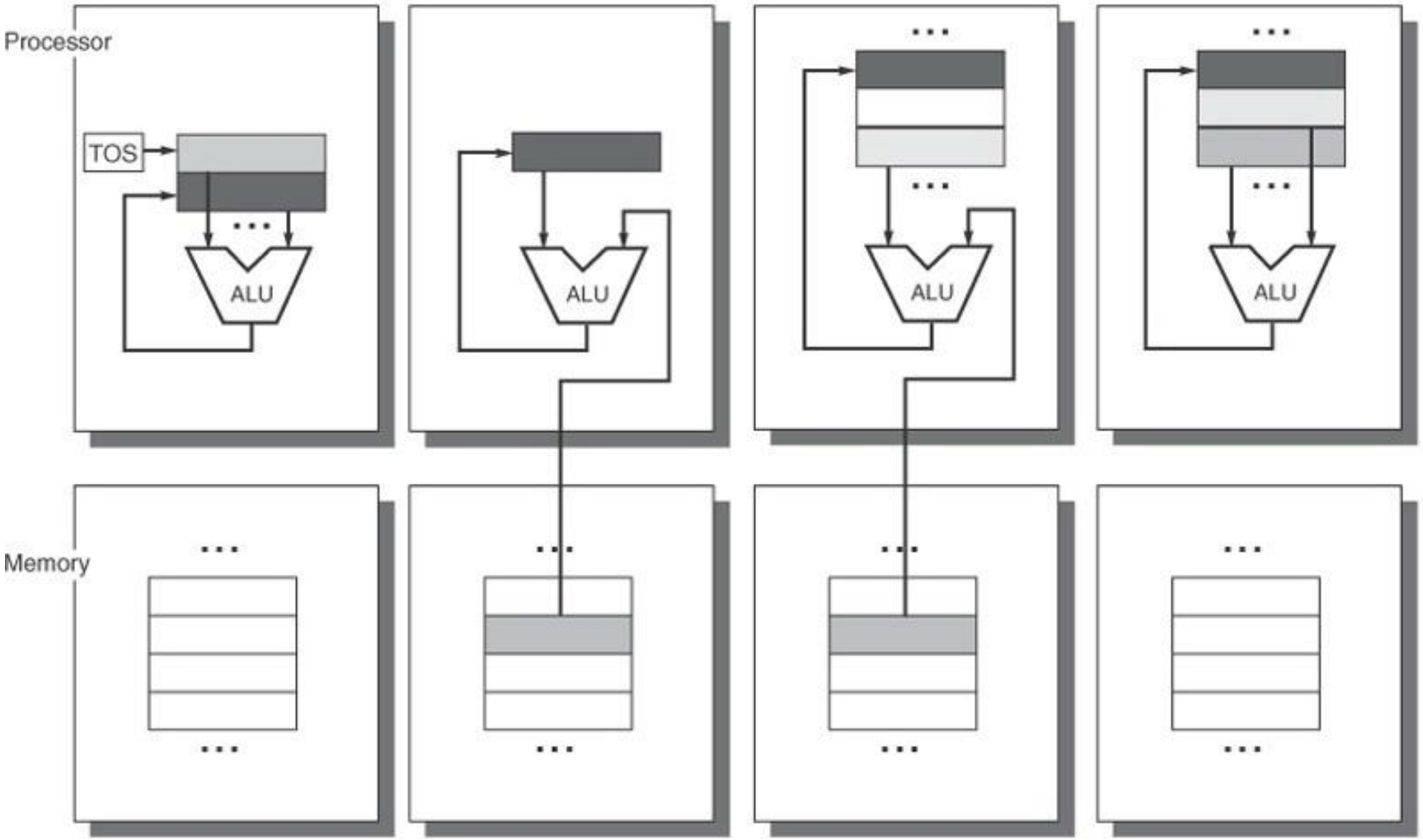
- Hybrid (IBM 360/370, MIPS16, Thumb)

Operation	Address specifier	Address field

Operation	Address specifier 1	Address specifier 2	Address field

Operation	Address specifier	Address field 1	Address field 2

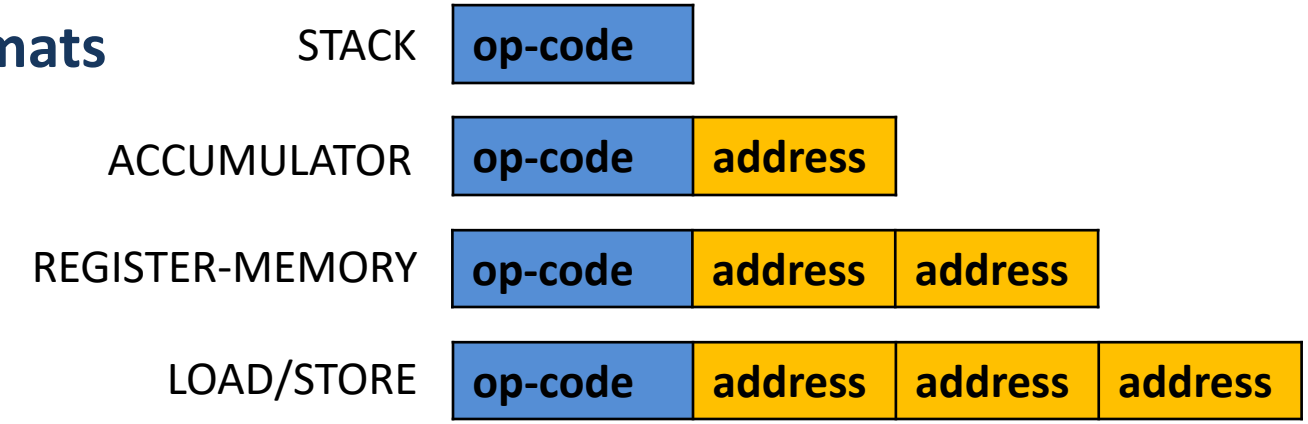
Basic ISA Classes



[2]

Basic ISA Classes

Instruction formats



STACK	ACCUMULATOR	REGISTER-MEMORY	LOAD/STORE
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store R1, C	Add R3, R2, R1
Pop C			Store R3, C

Assembly for C = A + B. Operands A, B, C are in memory

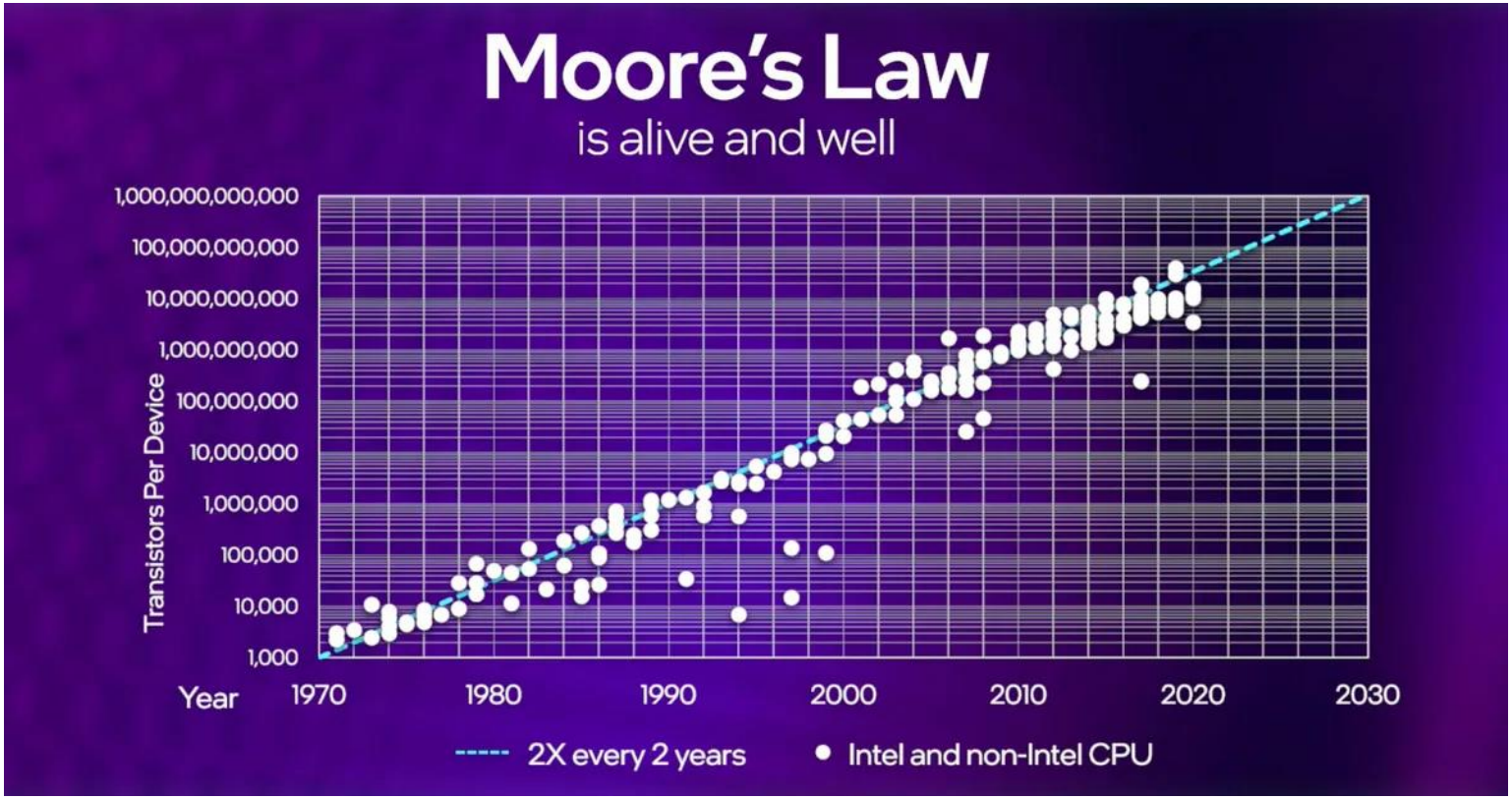
The add instruction has implicit operands for stack and ACC, explicit for GPR

Basic ISA Classes

- Location of operands
  - STACK (0 Address)
    - both operands are implicit TOS (top of stack) and SOS (second on stack)
    - the result goes to TOS
    - Special instructions for memory transfers: PUSH and POP
  - ACCUMULATOR (1 address)
    - one operand is the accumulator register
    - the other operand is given explicit
  - REGISTER-MEMORY (2 address)
    - the operands are registers or memory locations
    - the result is one of the source registers
  - LOAD/STORE (3 address)
    - all operands are registers
    - special instructions for accessing memory locations (load and store)

Technology – Moore's Law

- Moore's Law
  - Gordon Moore (1965): the number of transistors on a chip will double approximately every two years.



https://www.cnet.com/tech/computing/intel-will-outpace-moores-law-ceo-pat-gelsinger-says/

Technology – Power Consumption

- Dynamic Power (Watts)
  - in CMOS chips (switching transistors)

$$Power_{dynamic} = \frac{1}{2} \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

  - Slowing clock rate for a task reduces power consumption
  - Dynamic Power can be reduced by lowering the voltage
    - Voltages dropped from 5V to almost 1V in 20 years
  - Microprocessors stop the clock for inactive modules → energy saving
- Static Power (Watts)
  - Important due to leakage current (even if the transistor is inactive)

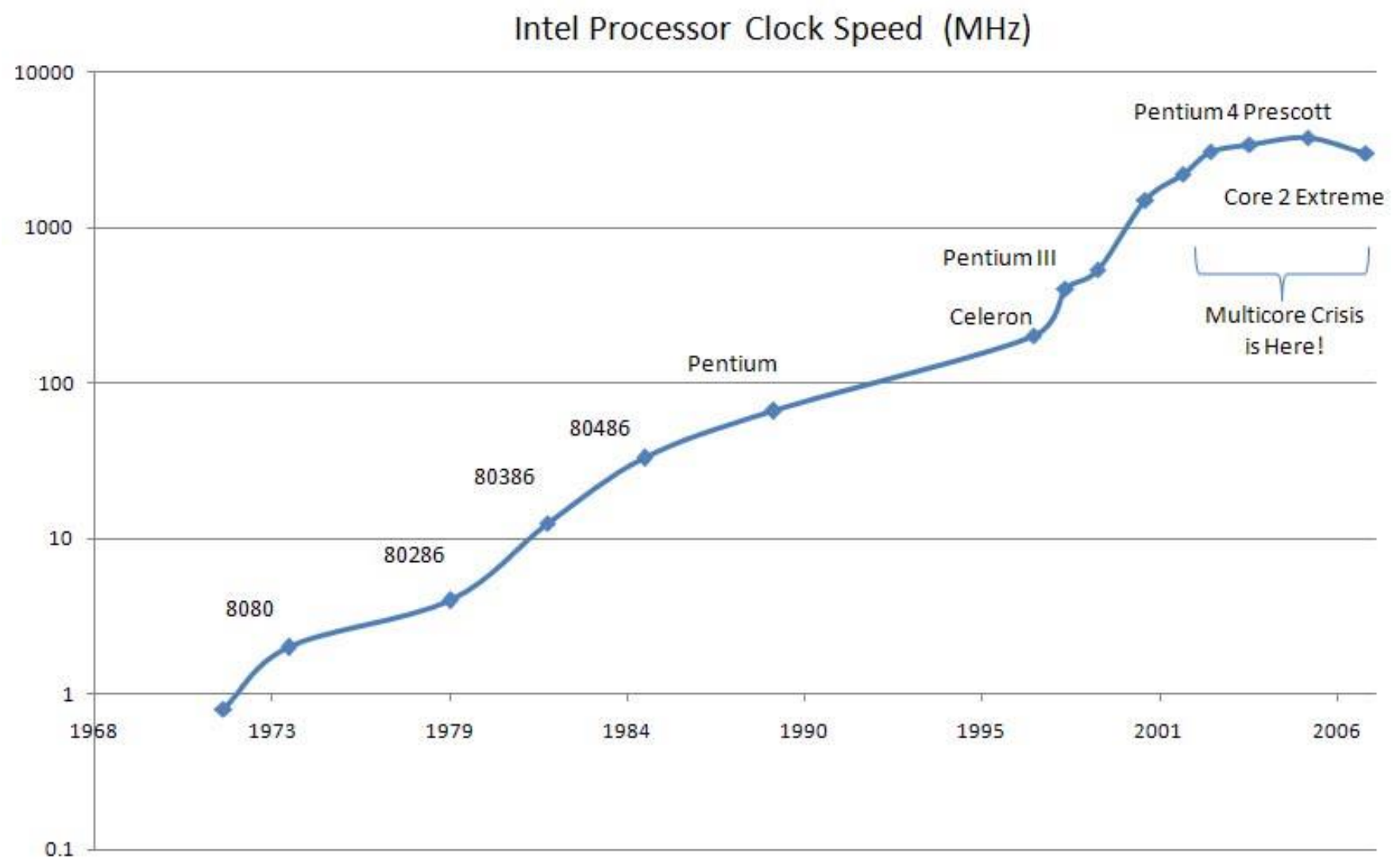
$$Power_{static} = Current_{static} \times Voltage$$

  - Proportional to the number of devices on a chip
  - Leak current increases as transistor size decreases

Technology – Power Consumption

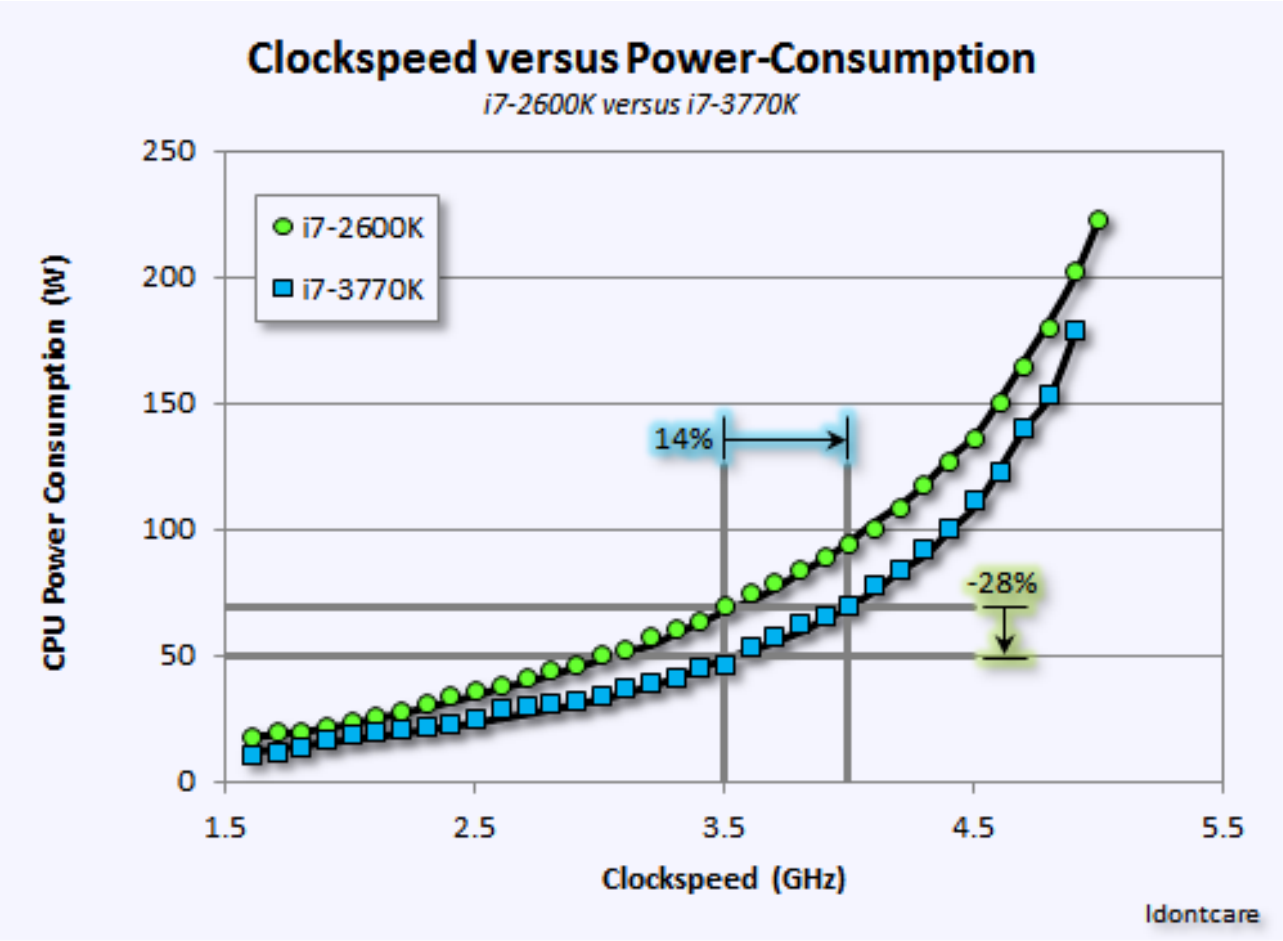
- Systems with reduce power consumption
  - Temperature diodes to reduce activity if the chip get's to hot
  - Reduce voltage and clock frequency or the issue rate of instructions
- In 2011, the target for leaks – 25% of the total power consumption
- First 32-bit microprocessors (Intel 80386) ~ 2 Watts
- Now, 3.3 GHz Intel Core i7 ~ 130 Watts
  - The heat from a chip (1.5 cm) must be dissipated → reach the limits of what can be cooled by air
- Design for power:
  - Sleep modes
  - Partially or totally reduce the clock frequency
  - Maximum operating temperatures → Low
  - The limits of air cooling have led to multiple processors on a chip running at lower voltages and clock rates

Clock Frequency Evolution



https://smoothspan.files.wordpress.com/2007/09/clockspeeds.jpg

Frequency vs. Power Consumption



https://www.quora.com/Why-havent-CPU-clock-speeds-increased-in-the-last-5-years

Computer Performance – Metrics

- Bandwidth over Latency
  - Bandwidth or throughput
    - Total amount of work in a given time
    - Number of tasks completed per unit time
    - Important when we run several tasks
  - Latency or execution time or response time (delay)
    - The time period to complete a task
    - Important if we have to run a time critical task
- Processor Performance Equation
  - IC – instruction count
  - CPI – average number of clock cycles per instruction
  - CCT – clock cycle time

$$CPUtime = \frac{CPU\ clock\ \text{cycles for a program}}{Clock\ rate} \qquad CPI = \frac{\text{CPU clock cycles for a program}}{Instruction\ \text{count}}$$

$$CPUtime = IC \cdot CPI \cdot CCT = \frac{Instructions}{\text{Pr}ogram} \cdot \frac{Cycles}{Instruction} \cdot \frac{Seconds}{Cycle} = \frac{Seconds}{\text{Pr}ogram}$$

Computer Performance – Metrics

- Computer Performance depends on
  - CCT → hardware and organization
  - CPI → organization and ISA
  - IC → ISA and compiler
- ISA influences the three components of computer performance
- Performance equation

$$Performance_x = \frac{1}{Execution\ \text{time}_x}$$

- Running speed of a program: MIPS (millions instructions per second)

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

Amdahl's Law

- "the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used"
- Speedup

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

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

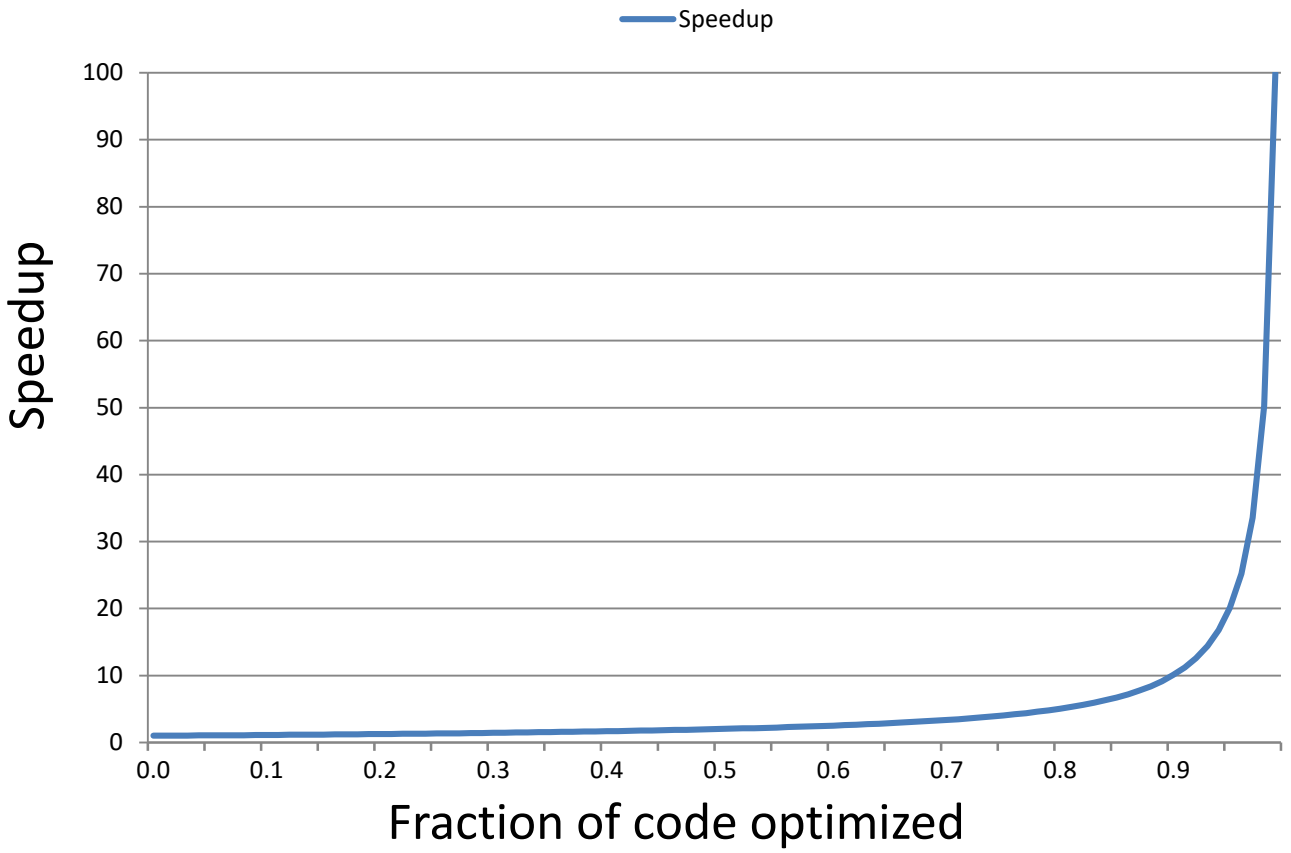
- Speedup depends on 2 factors:
  - The fraction of time that can benefit from enhancement $Fraction_{enhanced} = f_x$
  - The gain obtained by using the enhancement $Speedup_{enhanced} = S_x$

$$Execution\ \text{time}_{new} = Execution\ \text{time}_{old} \times \left((1 - f_x) + \frac{f_x}{S_x} \right)$$

Amdahl's Law

$$Speedup_{overall} = \frac{\text{Execution time}_{old}}{Execution\ time_{new}} = \frac{1}{(1 - f_x) + \frac{f_x}{S_x}}$$

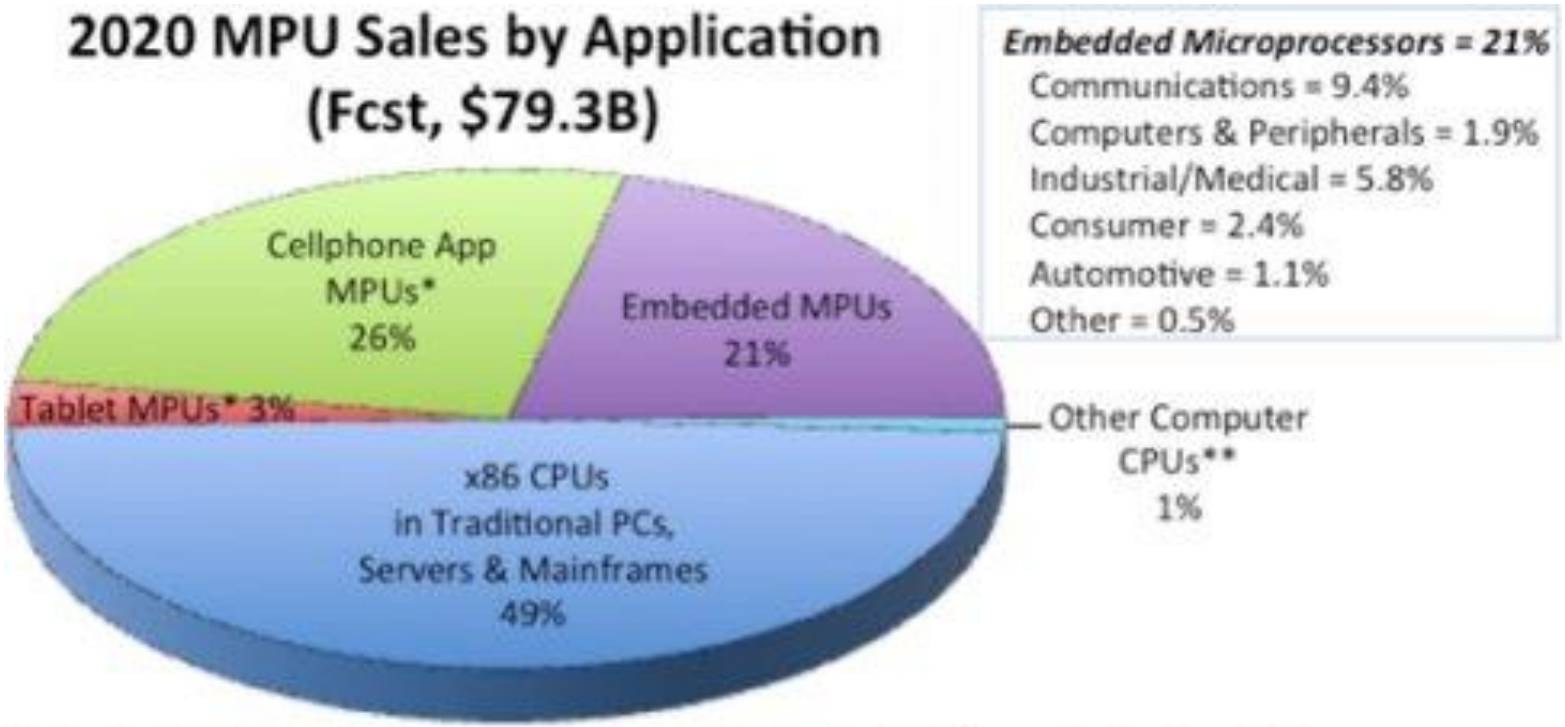
If S_x=100, what is the overall speedup as a function of f_x



Conclusions

- In 2004 Intel has canceled its uni-processor projects and has declared, together with IBM and SUN, that higher performances can be obtained by using more processors on a chip instead of making uni-processor systems more faster
- This is a historical turnaround from instruction level parallelism to thread and data level parallelism
- The compiler and the hardware exploit ILP implicitly
- For exploiting TLP and DLP the programmer is involved in developing faster codes
- Next: Multiprocessors, Multi-cores, Many-cores, etc.
- Processor market 2010:
  - 1.8 billion PMDs (90% cell phones), 350 mil. desktop PCs, 20 mil. servers
  - 19 billion embedded processors
  - ARM (RISC) ~ 6.1 billion caps, ~ 20 times more than x86

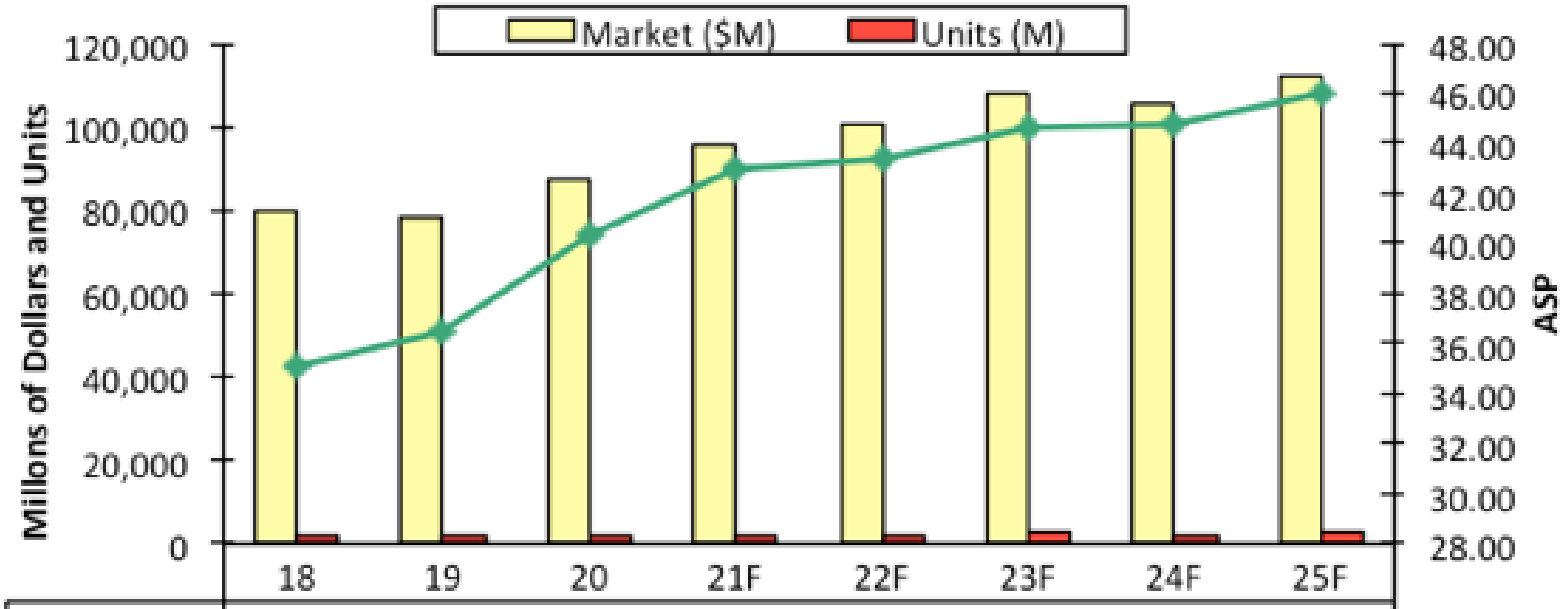
Processor market



https://www.design-reuse.com/news/48622/2020-mpu-sales-by-application.html

Processor market

Total Microprocessor Market History and Forecast



	18	19	20	21F	22F	23F	24F	25F
Market ($M)	80,131	78,237	87,673	95,495	100,895	108,347	105,872	112,191
Units (M)	2,284	2,146	2,175	2,226	2,323	2,429	2,368	2,440
ASP ($)	35.08	36.45	40.31	42.91	43.44	44.60	44.71	45.97

Source: IC Insights

https://www.design-reuse.com/news/49429/total-microprocessor-market-forecast.html

Problems – Homework

- Write a program using instructions defined by you for the 0, 1, 2 and 3 addresses processors to implement the following expression: e= a·b·c + d. The operands a, b, c, d and the result e are memory locations.
- For the 0, 1, 2 and 3 addresses machines write a program to evaluate the following expression: e=a·b + c·d.
- Describe the differences between big endian and little endian.
- …

References

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013
2. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011
3. D. M. Harris, S. L. Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann, San Francisco, 2007
4. D. A. Patterson, J. L. Hennessy, “ORGANIZAREA SI PROIECTAREA CALCULATOARELOR. INTERFATA HARDWARE/SOFTWARE”, Editura ALL, Romania, ISBN: 973-684-444-7

- …

VHDL – Remember

- Types of Circuits
  - Combinational Circuits
  - Sequential circuits

- Basic building blocks
  - Logic Gates
  - Multiplexers
  - Decoders
  - D-Latches and D-Flip-Flops
  - Counters
  - Memories

VHDL – To Remember

- **Rules of VHDL coding!!!**
  - Not EVERYTHING is a component. Do not create components for basic building blocks like: logic gates, latches, flip-flops, tri-state buffers, counters, decoders, etc.
  - Do not abuse of structural design at the logic gate granularity!
  - You will generally use the behavioral type of describing your design.
  - You will create a new component **only when a part of your design has meaning** (or when the TA explicitly tells you to do so).

VHDL – Remember

- 1-bit signal declaration

  signal sig_name : std_logic := ‘0’;

- N-bit signal declaration

  signal sig_name: std_logic_vector(N-1 downto 0): =“00….0”;

- Initialization

  16-bit signal “0000000000000000”;

  16-bit signal x“0000”;

  16-bit signal (others => ‘0’);

VHDL – Remember

- Logic Gates – A & B – inputs, O – output

NOT

O <= not A;

AND

O <= A and B;

OR

O <= A or B;

NAND

O <= A nand B;

NOR

O <= A nor B;

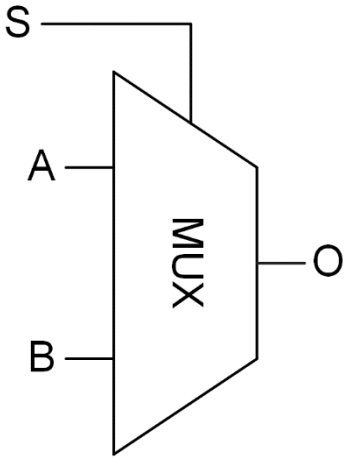
XOR

O <= A xor B;

- Do not declare an entity, only signals if needed!

VHDL – Remember

- 2:1 Multiplexer

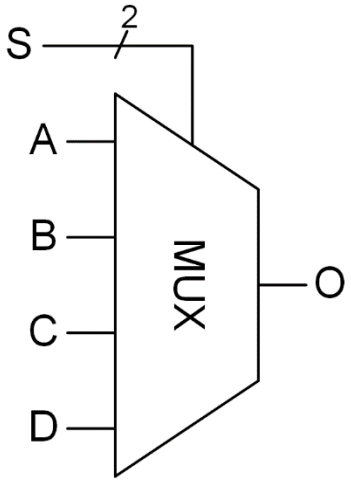


Do not declare an entity, only signals if needed!

```
O <= A when S = '0' else B;
```

```
process(S, A, B)
begin
   if(S = ‘0’) then
      O <= A;
   else
      O <= B;
   end if;
end process;
```

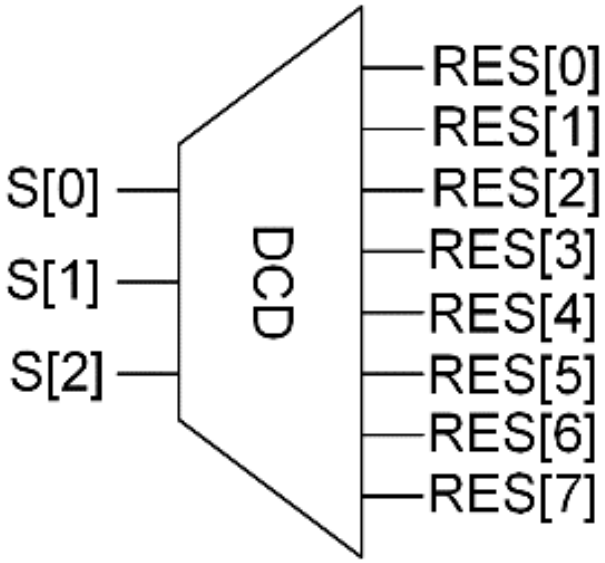
- 4:1 Multiplexer




```
process(S, A, B, C, D)
begin
   case S is
      when "00"    => O <= A;
      when "01"    => O <= B;
      when "10"    => O <= C;
      when others => O <= D;
   end case;
end process;
```

VHDL – Remember

- 3:8 Decoder

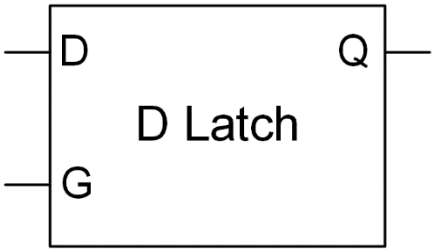



```
process(S)
begin
   case S is
      when "000"   => RES <= “00000001”;
      when “001"   => RES <= “00000010”;
      when “010"   => RES <= “00000100”;
      when “011"   => RES <= “00001000”;
      when “100"   => RES <= “00010000”;
      when “101"   => RES <= “00100000”;
      when “110"   => RES <= “01000000”;
      when others  => RES <= “10000000”;
   end case;
end process;
```

- Do not declare an entity, only signals if needed!

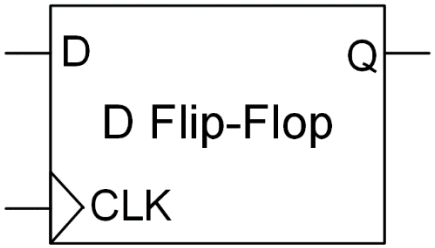
VHDL – Remember

- D-Latch




```
process(G, D)
begin
   if(G = '1') then
      Q <= D;
   end if;
end process;
```

- D-Flip-Flop

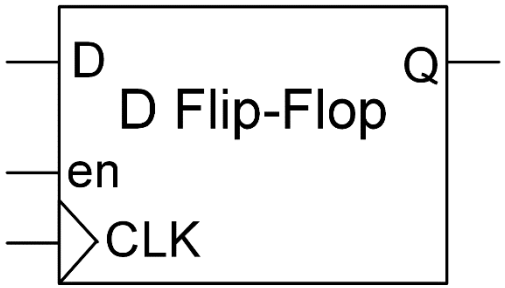



```
process(clk)
begin
   if rising_edge(clk) then
      Q <= D;
   end if;
end process;
```

- Do not declare an entity, only signals if needed!

VHDL – Remember

- D-Flip-Flop with enable

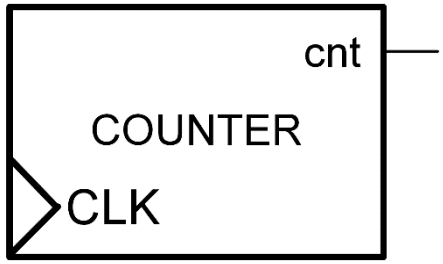



```
process(clk, en)
begin
   if rising_edge(clk) then
      if en = '1' then
         Q <= D;
      end if;
   end if;
end process;
```

- rising_edge(clk) is equivalent to clk'event and clk = '1' but shorter
- NEVER use rising_edge(clk) and en = '1' a.k.a. Gated Clock!
- Do not declare an entity, only signals if needed!

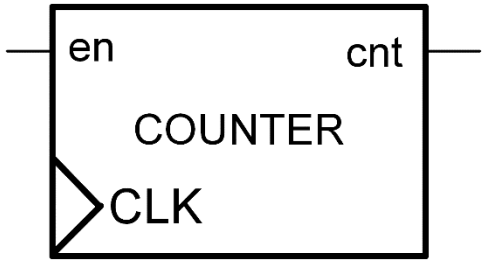
VHDL – Remember

- Up Counter




```
process(clk)
begin
   if rising_edge(clk) then
      cnt <= cnt + 1;                    -- '1'
   end if;
end process;
```

- Up Counter with enable signal




```
process(clk, en)
begin
   if rising_edge(clk) then
      if en = '1' then
         cnt <= cnt + 1;
      end if;
   end if;
end process;
```

- Do not declare an entity, only signals if needed!