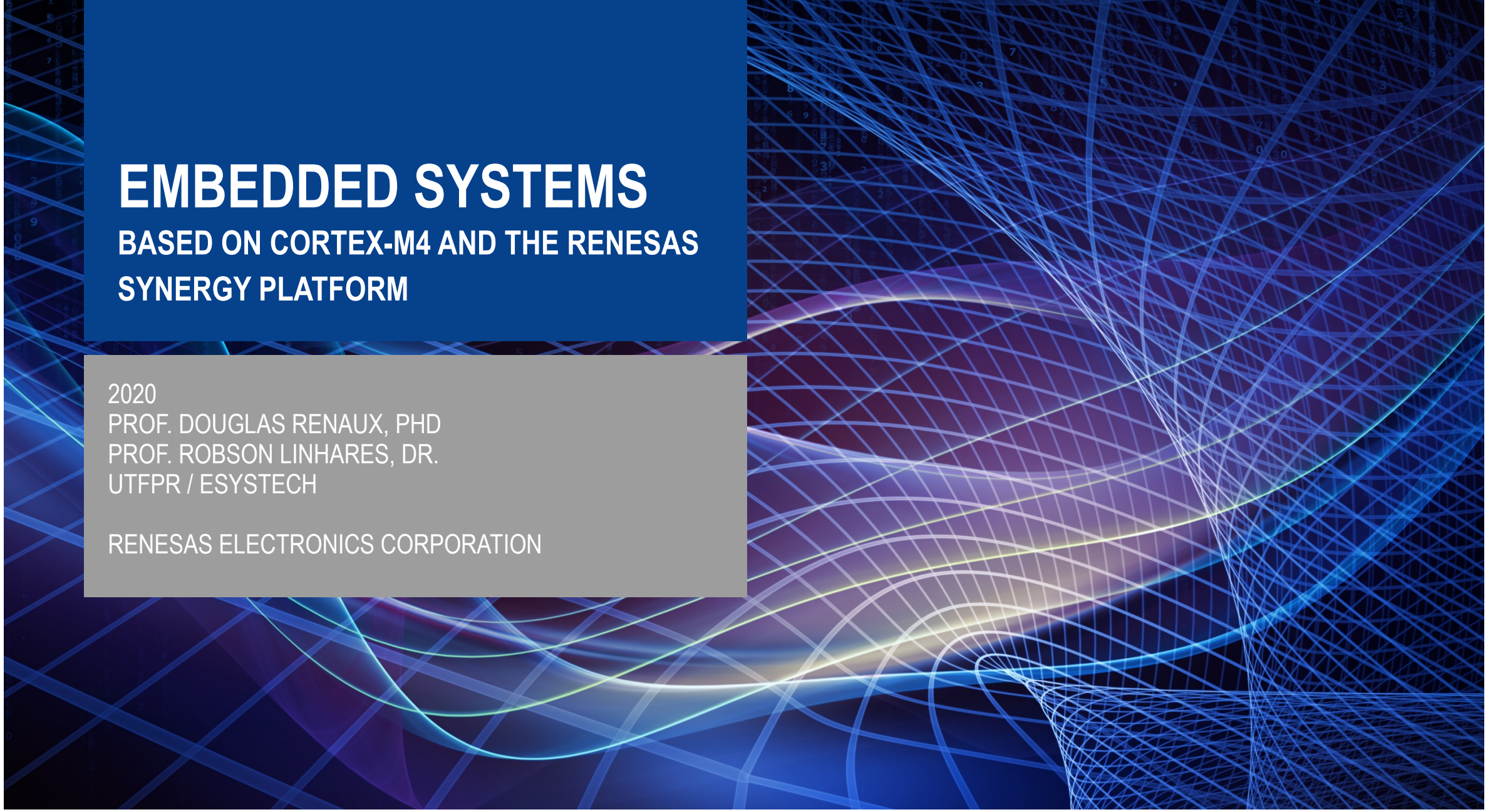


EMBEDDED SYSTEMS

BASED ON CORTEX-M4 AND THE RENESAS SYNERGY PLATFORM

2020
PROF. DOUGLAS RENAUX, PHD
PROF. ROBSON LINHARES, DR.
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

2 – COMPUTER ARCHITECTURE – RISC VS CISC

- Computer Generations
- The RISC Paradigm

DEFINING “COMPUTER”

Computer =

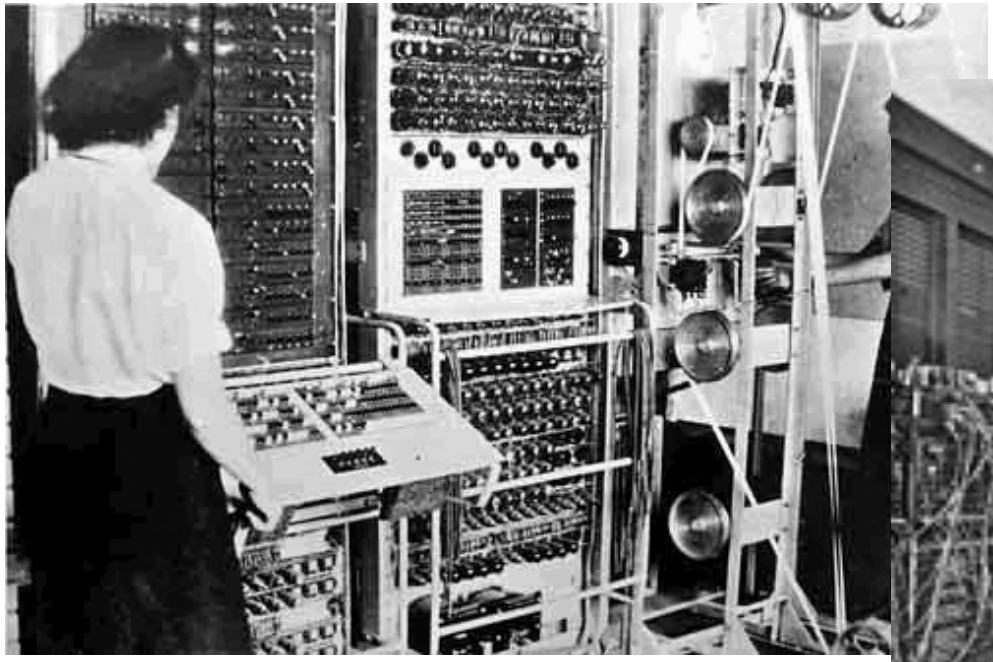
a device, or person, who performs a computation, i.e. a sequence of calculations according to an algorithm.

Hence, we consider Generation 0 of computers the generation that precedes the electronic device currently known as computer.

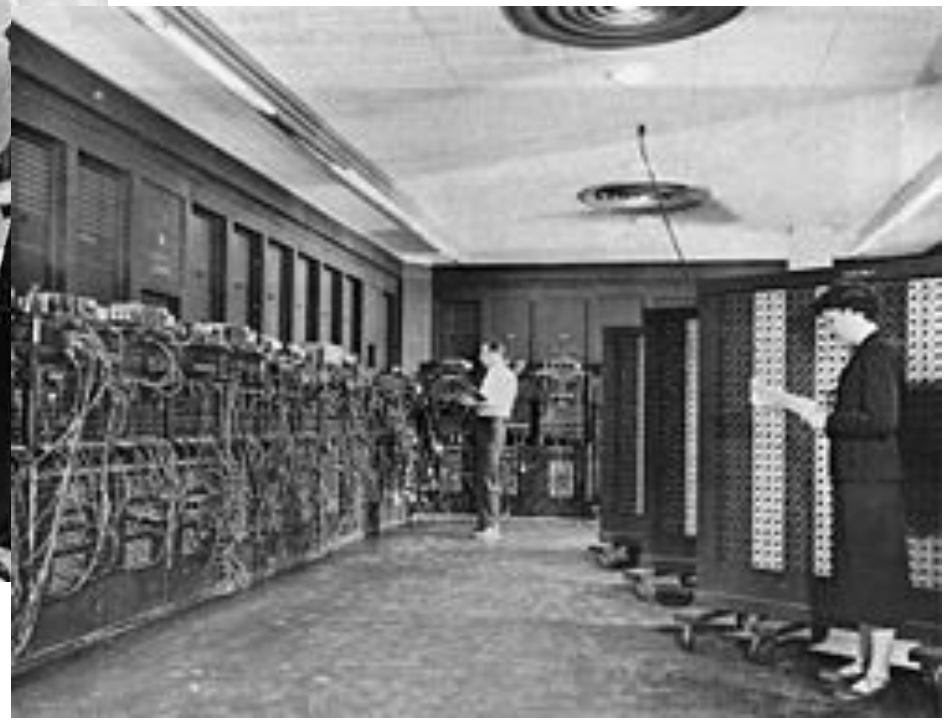
COMPUTER GENERATIONS

Generation	Description
0	Mechanical and Electromechanical Devices
1	40's – Vacuum tubes ENIAC, Zuse
2	50's – Transistors Manchester University, IBM 350
3	60's – SSI Integrated Circuits (logic gates) Apollo Guidance Computer IBM System/360, Digital VAX
4	70's – Microprocessor
5	2010's? – quantic / organic / optical??? AI

FIRST GENERATION



Colossus - UK

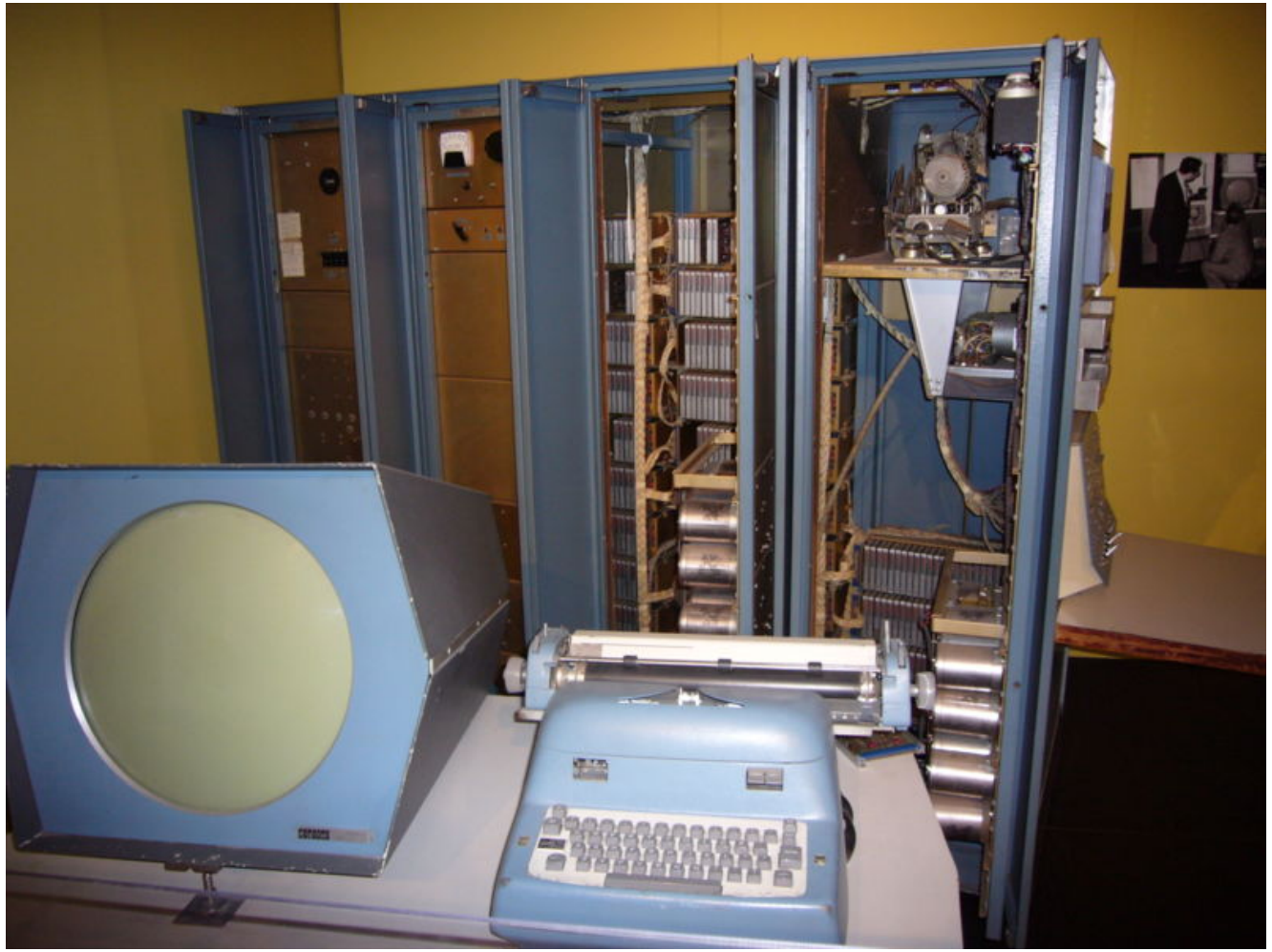


ENIAC - USA

source: wikimedia.org (CC)

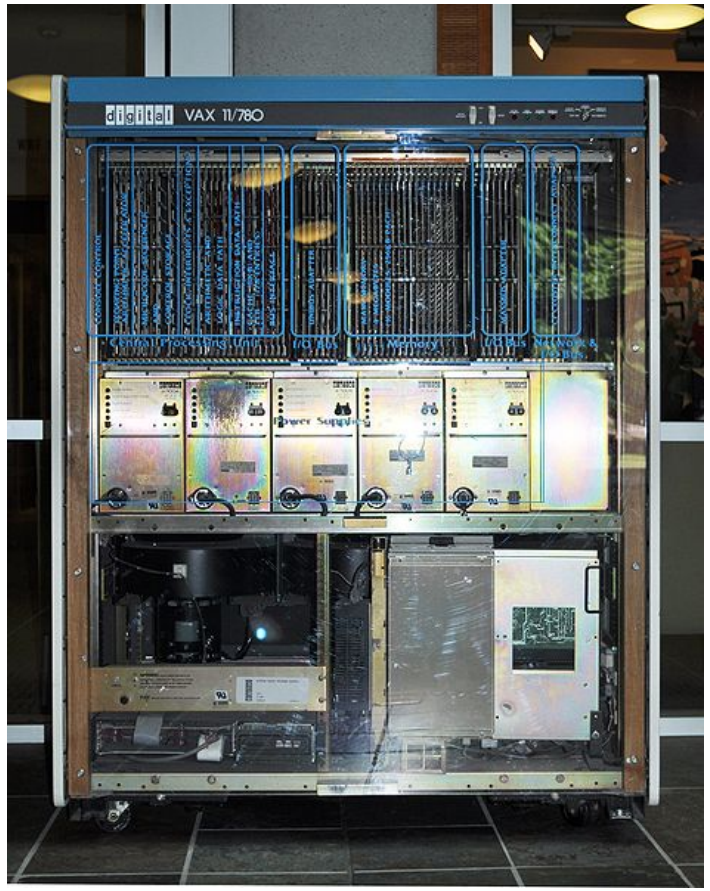
SECOND GENERATION

PDP-1



source: wikimedia.org (CC)

THIRD GENERATION



VAX 11/780

source: wikimedia.org (CC)

EXAMINING THE EVOLUTION OF THE FIRST GENERATIONS – 1

From the 40’s to early 80’s, electronic hardware had a significant improvement:

vacuum tubes -> transistors -> SSI -> MSI -> microprocessors

providing the computer architect with a significant increase in the amount of processing power.

- How did the architecture of computers evolve during this period?
- What was the added complexity?
- How did the evolution of the architecture affect the performance of the machines?

EXAMINING THE EVOLUTION OF THE FIRST GENERATIONS – 2

- As HW provided the means, computer architecture grew increasingly more complex.
- Semantic contents of the instructions increased as to reduce the semantic gap to High Level Languages (HLL).
- Aim was to reduce the number of machine language instructions needed to implement an HLL program.
- Instructions got so complex that a small program was executed for each instruction -> microcode.
- In practice, a microprocessor embedded in the processor interpreting each instruction as they were executed.

RESEARCH PRECEDING THE RISC PARADIGM SHIFT

- Careful examination of the execution of actual programs concluded that:
  - Often, complex instructions were not used and the equivalent effect was obtained by a sequence of simpler instructions.
  - The inclusion of a single complex instruction in the instruction set could impact overall performance by imposing a lower clock rate.
  - Since the instructions did not have a regular execution sequence (i.e. each one had its execution determined by its microcode) the implementation of a pipeline was close to impossible.

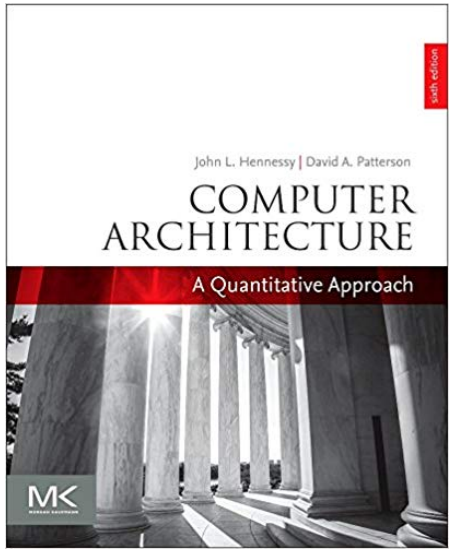
RESEARCH PRECEDING THE RISC PARADIGM SHIFT

What is the final goal?

- From the user's perspective, the goal is **performance of the application programs**

- **RISC thesis**: a processor whose instruction set is made of simple instructions with a regular execution, allowing the implementation
  of a pipeline, will have a higher performance than a CISC
  (Complex Instruction Set Computer)

recommended reading:
Computer Architecture: A Quantitative Approach 6th Edition
by John L. Hennessy, David A. Patterson



RISC ARCHITECTURAL FEATURES

Features that characterize the RISC paradigm are:

- An instruction set with a reduced number of very simple instructions.
- LD/ST architecture, meaning that only two instructions (and their variants) access memory: LD (load) reads data from memory, and ST (store) saves data to memory.
- All other instructions operate on registers. There is a large number of general-purpose registers, avoiding access to memory.
- All instructions follow the same logical execution sequence. Hence, a pipelined architecture can be implemented, significantly improving performance. Typically 1 instruction is executed every clock cycle.

PIPELINING

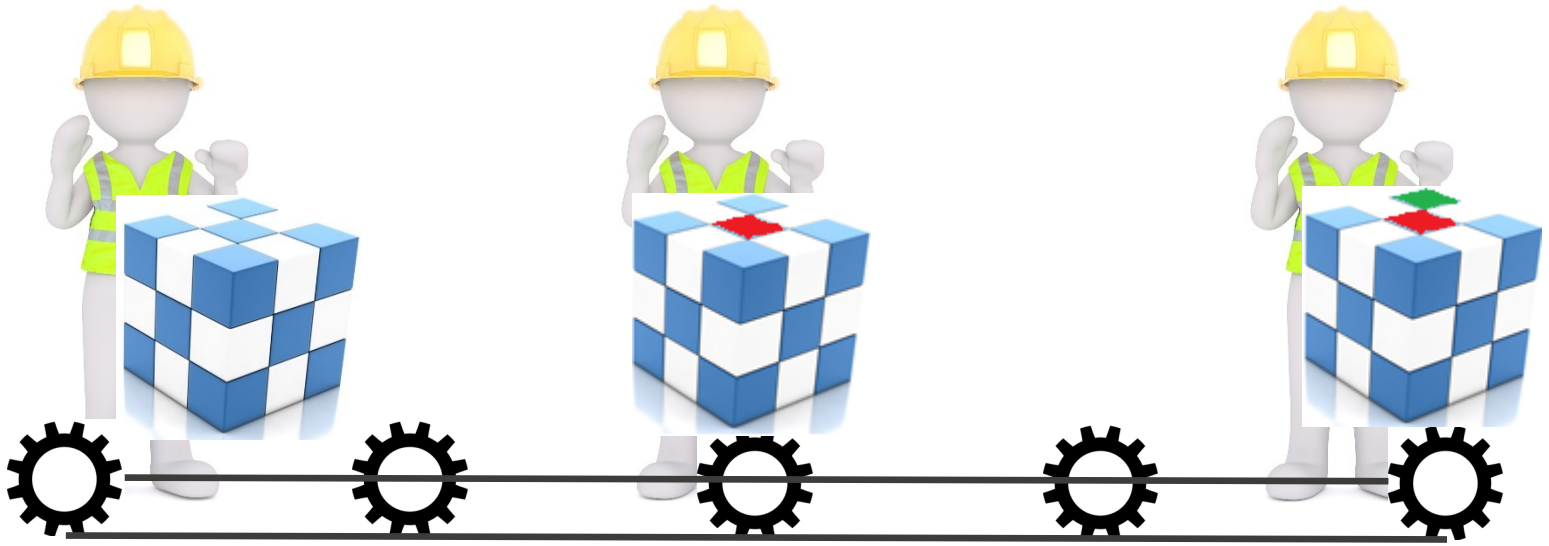
Illustrating the concept of Pipelining using workers

No pipeline = idle workers available = resources wasted

source: pixabay.com (CC)

PIPELINING

Illustrating the concept of Pipelining using workers



Using a pipeline = all workers executing activities = no resources wasted

source: pixabay.com (CC)

PIPELINING

Requirements for the implementation of a Pipeline in a processor:

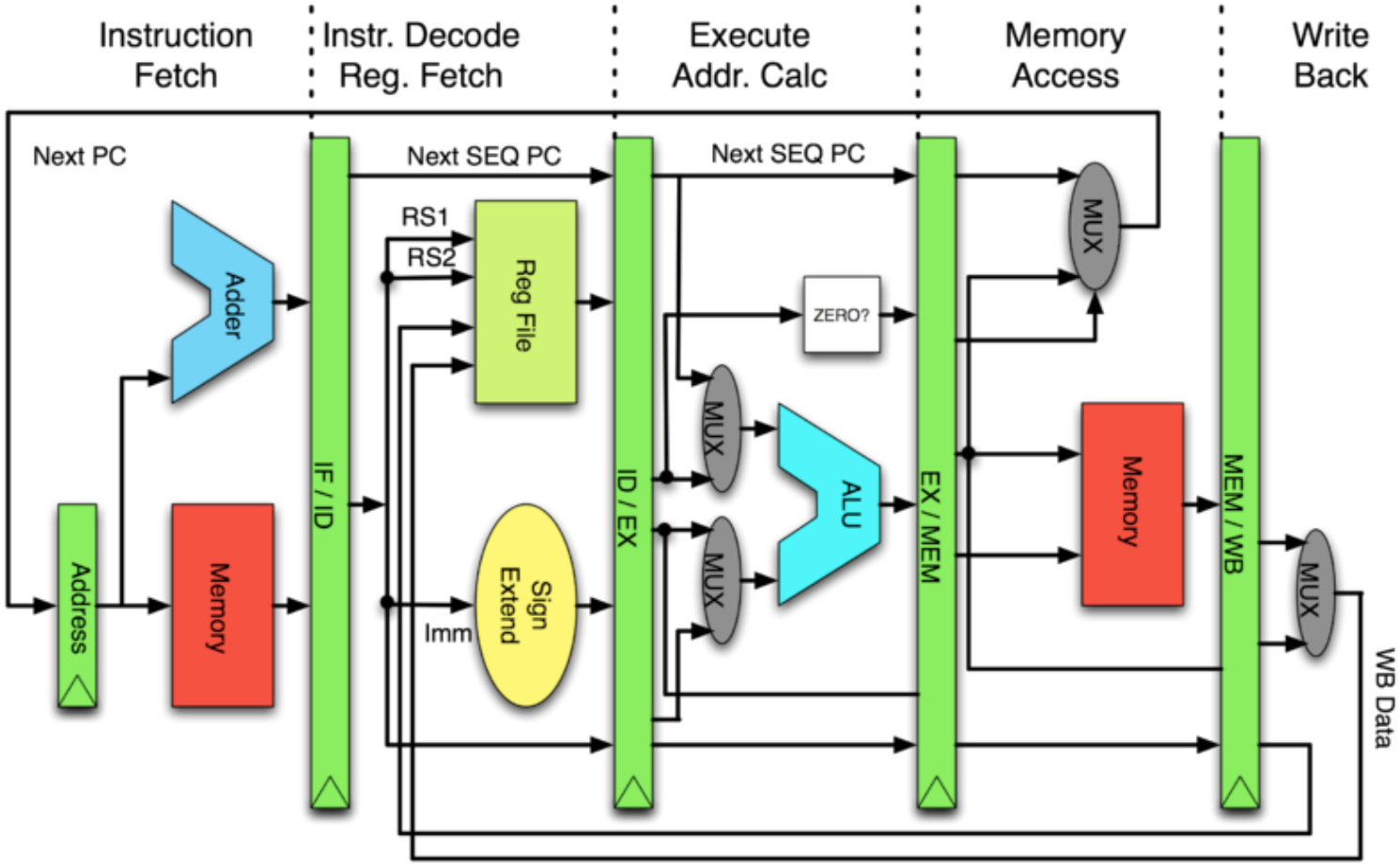
- The instruction set must be designed so that all instructions have the same execution steps
- Instructions must be regular with respect to:
  - Size
  - Addressing modes
  - Decoding
  - Operands

PIPELINING

Example of a Pipeline with 5 stages:

1. Fetch
2. Decode and read operands in registers
3. Use ALU to execute instruction or to calculate memory address
4. Read or Write to memory
5. Write result back to Register file

Remark: not all instructions use step 4 and 5



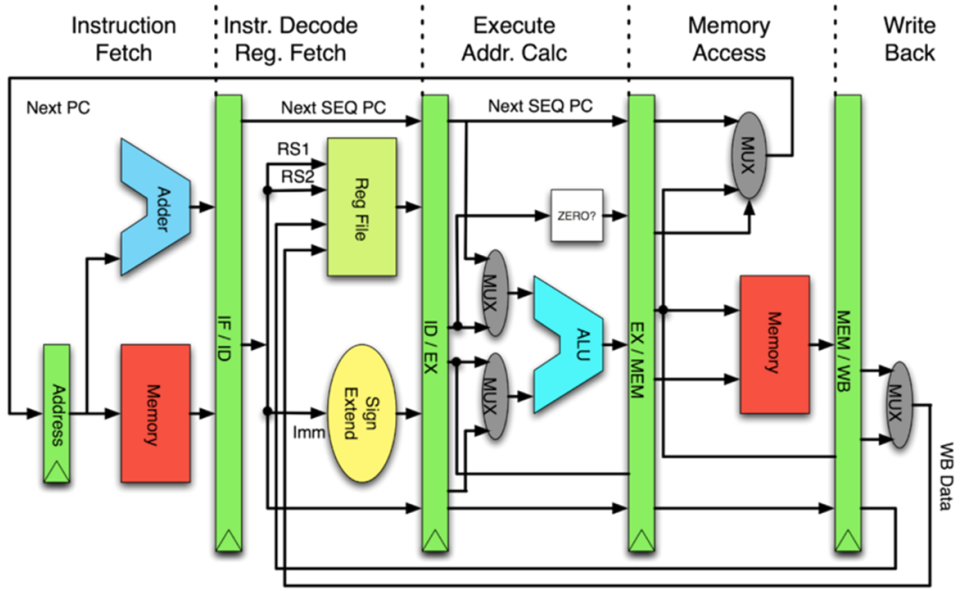
source: wikimedia.org (CC)

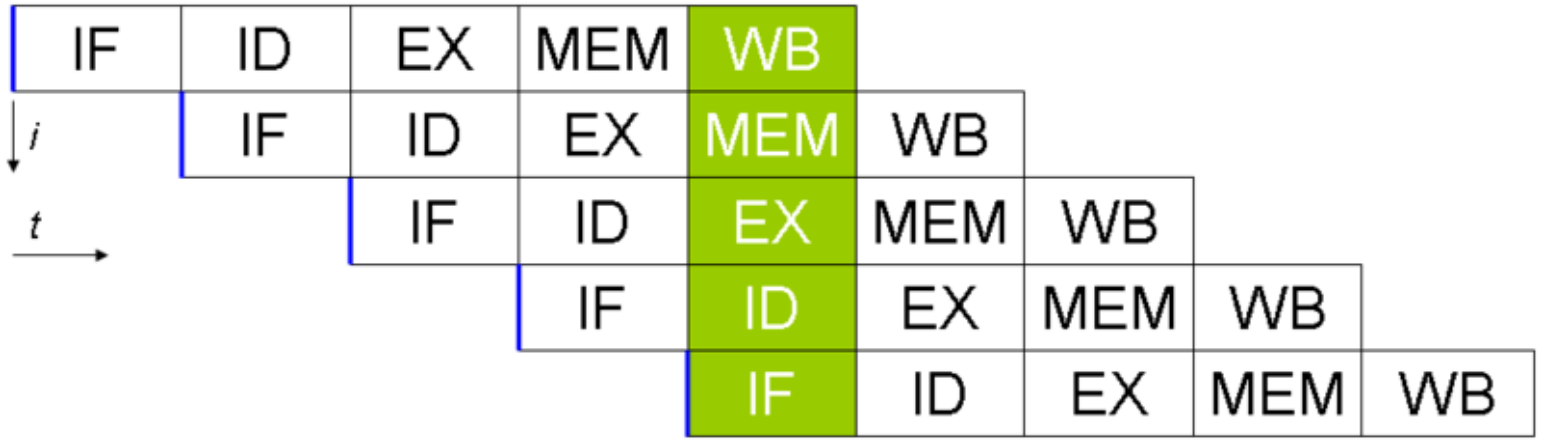
BENEFITS OF PIPELINING

For the previous example of a 5 stage pipeline, the
execution over time is presented in this diagram.

Note that although each instruction takes 5 clock cycles to execute, due to pipelining, after filling the pipeline, on every clock another instruction finished its execution.

Effectively, the number of instructions executed per second is the same as the clock rate.





source: wikimedia.org (CC)

LIMITATIONS OF PIPELINING

Consequence:
the average number of instructions executed per second is lower than the clock rate.

- Data Hazard:

  if an instruction requires as input a value produced by the previous instruction, this value may not be available at stage 2 (decode and fetch operands) since the previous instruction only saves its results at stage 5.

  Solutions: stall the pipeline (instruction must wait for result of previous)

  OR use a technique called bypassing to forward the result of the previous instruction at stage 3 of its execution

  OR reorder instructions to avoid this dependency

- Control Hazard:

  a change of control flow (e.g. a branch instruction) causes the pipeline to be flushed, meaning that following instructions that already were fetched and decoded will be discarded.

COMPARING RISC X CISC

RISC	CISC
Instruction set has a reduced number of instructions	Instruction set has a large number of instructions
Instructions are very simple	Instructions are complex, i.e. have a high semantic content
Large number of general purpose register	Small number of registers
Instruction codes have fixed size	Instructions are coded in a variable number of bytes
Instruction decoding is very simple and typically performed by a table in ROM	Instruction decoding is complex
Instruction execution is simple, typically requiring a single clock cycle	Instruction execution is complex and typically uses microprogramming, i.e. interpretation by microcode
Regular execution of the instructions	Execution steps varies from instruction to instruction
Execution time is regular, typically a single clock cycle	Execution time varies significantly among instructions
Architecture is prone to the use of a pipeline in the implementations, resulting in N times performance improvement (N is the number of pipeline stages).	Architecture is not prone to the use of Pipeline.

COMPARING RISC X CISC

When the same source program is compiled to a RISC processor and to a CISC processor:

- The number of machine instructions of the compiled program is typically larger on the RISC
- The size of the compiled program (measured in bytes) is typically larger on the RISC
- The performance (inverse of the time to execute the program) is typically better on the RISC

COMPARING RISC X CISC

From a HW perspective, the design of a RISC processor is significantly simpler than that of a CISC:

- Shorter design cycle,
- Lower number of transistors to implement,
- Less area on silicon die.

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

The cost of a chip is determined mainly by:

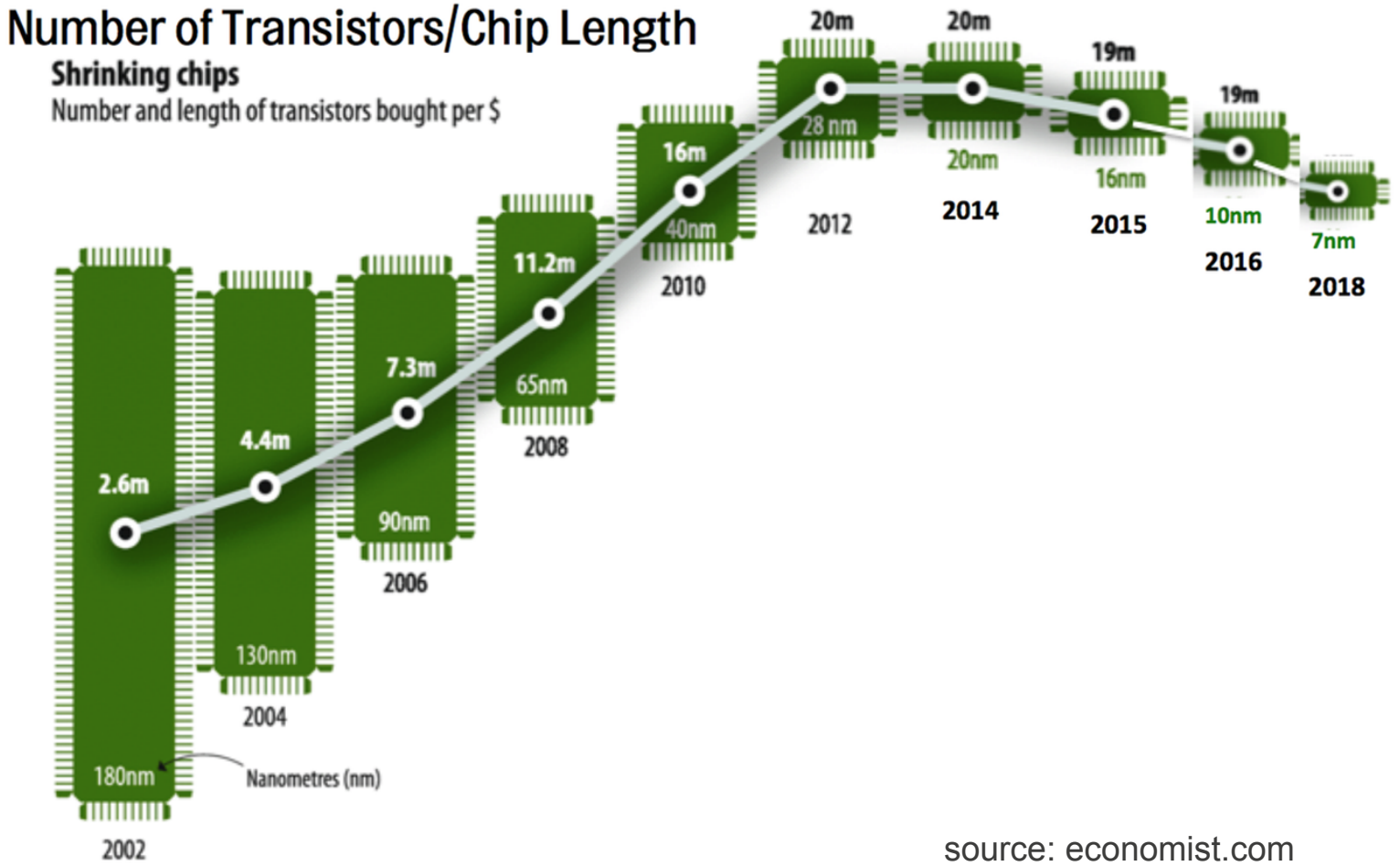
- Cost of the die (cost of manufacturing a wafer divided by the number of good dies per wafer)
- Testing costs (both for die testing and testing after packaging)
- Packaging costs
- Yield (percentage of functional chips).

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

Curve shows how many transistors can be purchased with one dollar for each manufacturing technology over time.

This graph represents the situation in 2018. At the time, the most cost-effective technology was 20 nm.

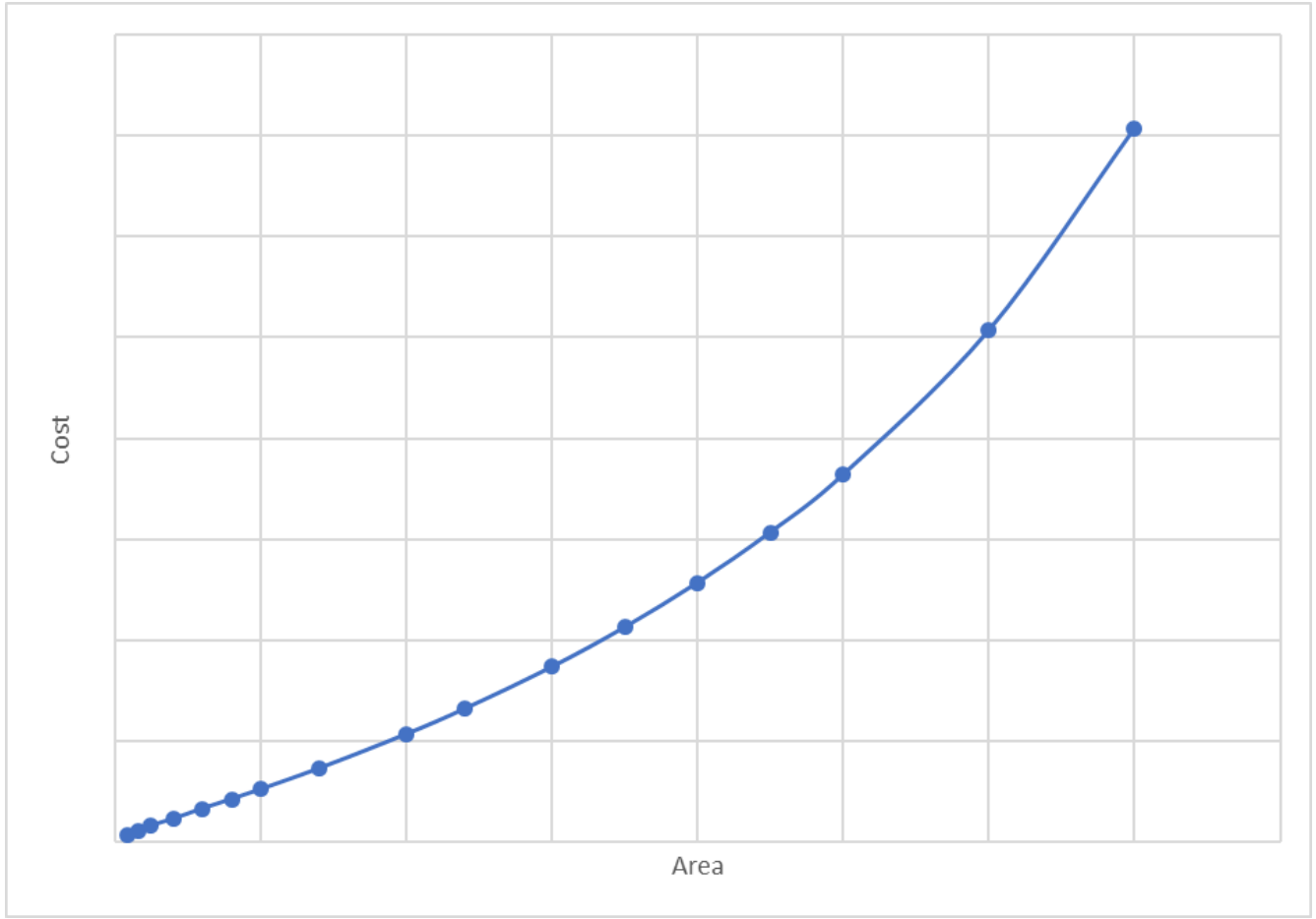
Over time, as the processes mature, transistors cost lowers. Until a given technology enters obsolescence.



source: economist.com

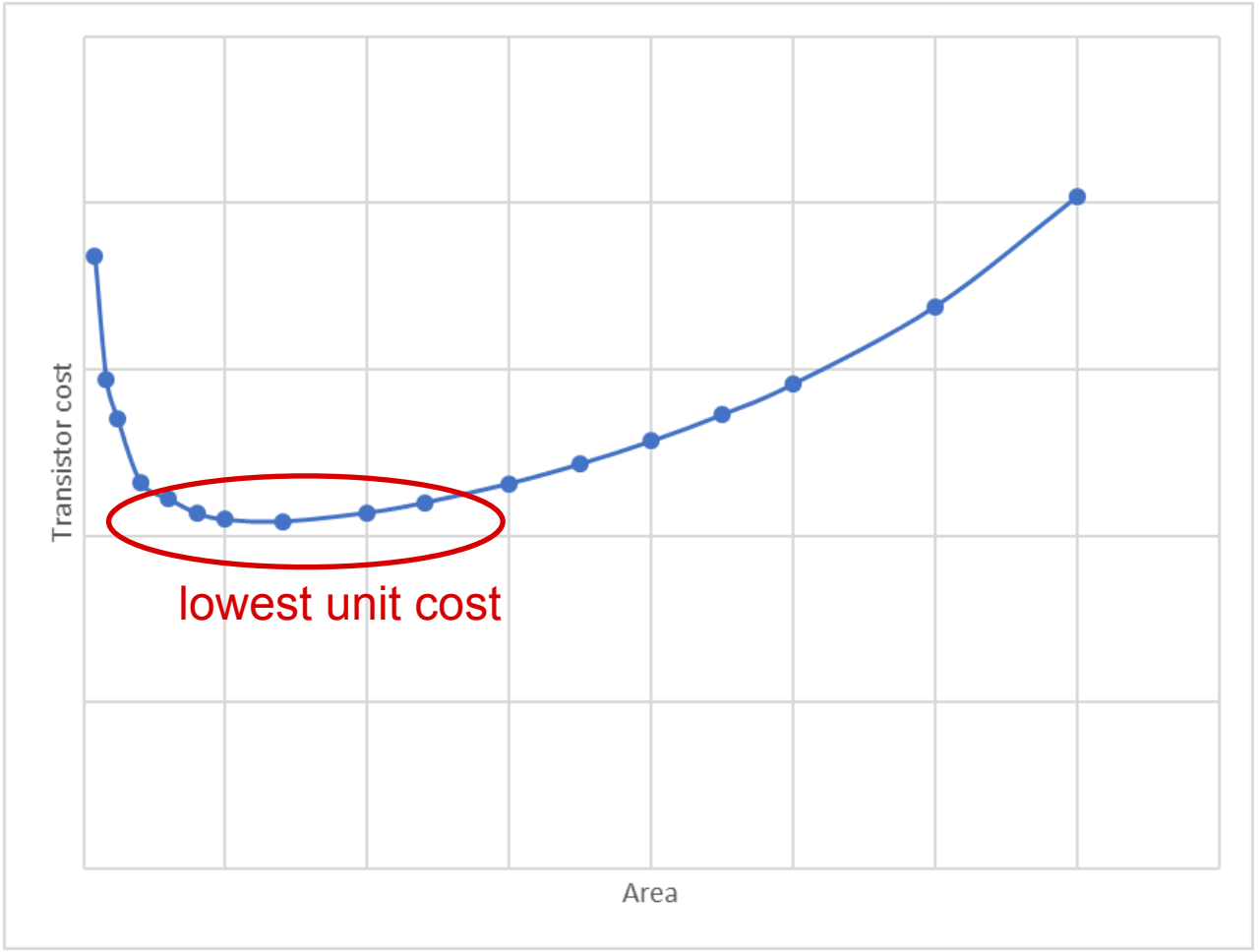
ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

As the chip gets larger the yield reduces, hence, chip costs does not grow linearly with the area.



ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

Plotting the cost of each transistor, as the chip gets larger, shows that there is an area range with the lower transistor costs.



ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

In a highly competitive market, manufactures aim at the best possible selection of features:

- Manufacturing process with the highest number of transistors / dollar (20 nm) per “number of transistors/chip length” curve
- Chip size in the optimal region (previous slide)

These two decisions lead to a given number of transistors (this number changes over time as manufacturing processes improve).

ECONOMICS OF INTEGRATED CIRCUITS MANUFACTURING

Key design decision:

If currently the “sweet spot” (best value for money) is a given manufacturing technology and a transistor count of T millions transistors, **what is the best use for this asset?**

1. CISC core + little Cache/Memory/Peripherals
2. RISC core + large amount of Cache/Memory/Peripherals

Most silicon vendors select the second alternative

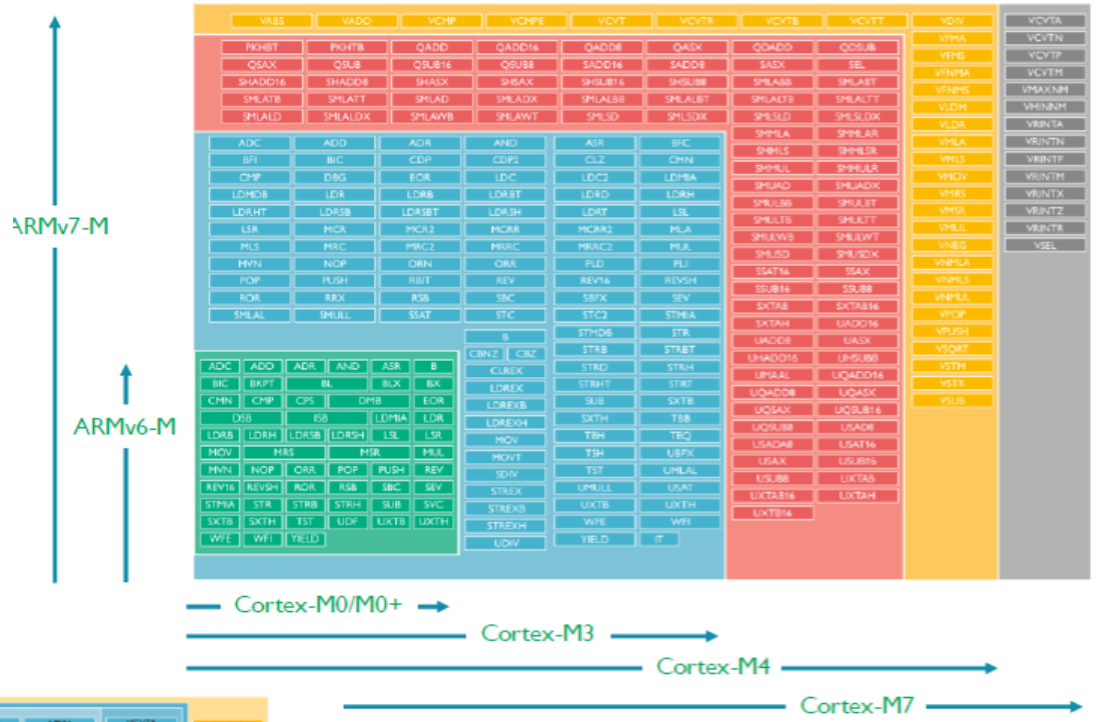
RISC VS CISC TODAY

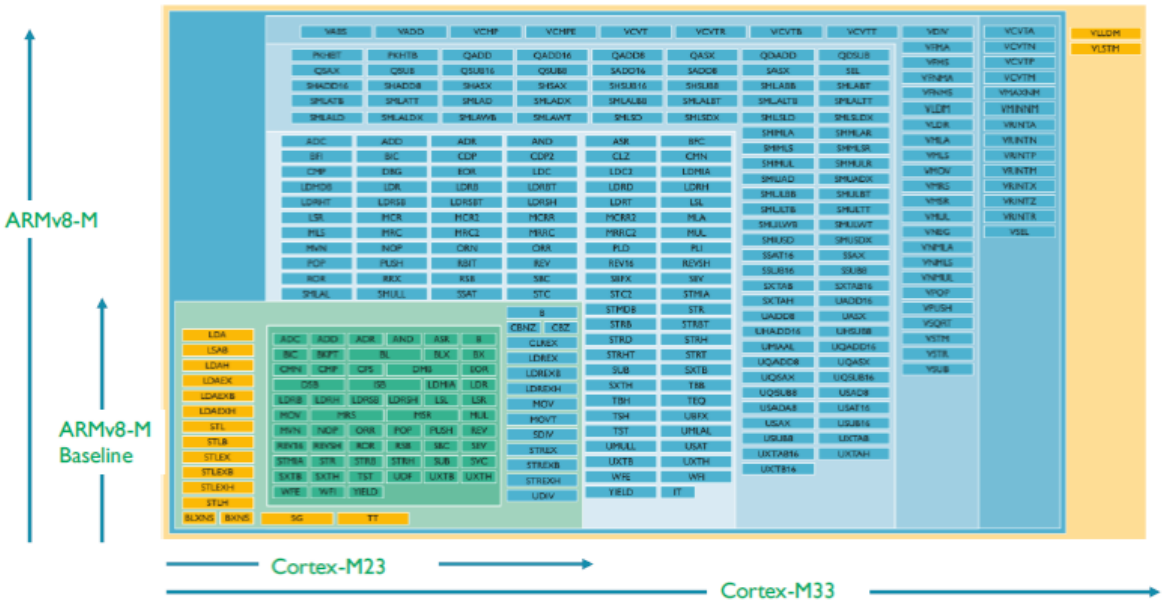
Currently, there is a trend to mix the best features of both paradigms. The convergence of RISC and CISC has not been named yet, but the characteristics of novel processors are:

- Large instruction set
- Regular instructions, prone to pipelining
- Pipelines from 3 to 20 stages
- No microcode

RISC VS CISC TODAY

Instruction set of Cortex-M including Cortex-M23 and Cortex-M33





source: ARM
ARM Cortex-M for Beginners - Yiu

3 – ARM CORTEX-M ARCHITECTURE

- History of ARM
- Cortex-M Features
- Cortex-M Instruction Set Architecture
- Instruction Set
- Memory Access
  - Memory-mapped I/O
- Exception Handling

REFERENCES AND RECOMMENDED READING

- ARMv7-M Architecture Reference Manual (2014) ARM DDI 0403E.b
- ARM® Cortex® -M4 Processor Revision: r0p1 (2015) Technical Reference Manual ARM 100166_0001_00_en
- Cortex™-M4 Devices Generic User Guide (2010) ARM DUI 0553ª
- ARM Cortex-M for Beginners: An Overview of the ARM Cortex-M processor family and comparison - Joseph Yiu - link
- ARM Cortex-M3 Introduction - ARM University Relations - presentation
- ARM Architecture Overview - presentation
- The ARM Architecture (with focus on Cortex-M3) - Joe Bungo, ARM University Program - presentation
- ARM Cortex-M Processor Family - presentation
- The Future is in Your Hands - Peter Middleton - presentation
- How to Choose Your ARM Cortex-M Processor - Tim Menasveta

Renesas.com