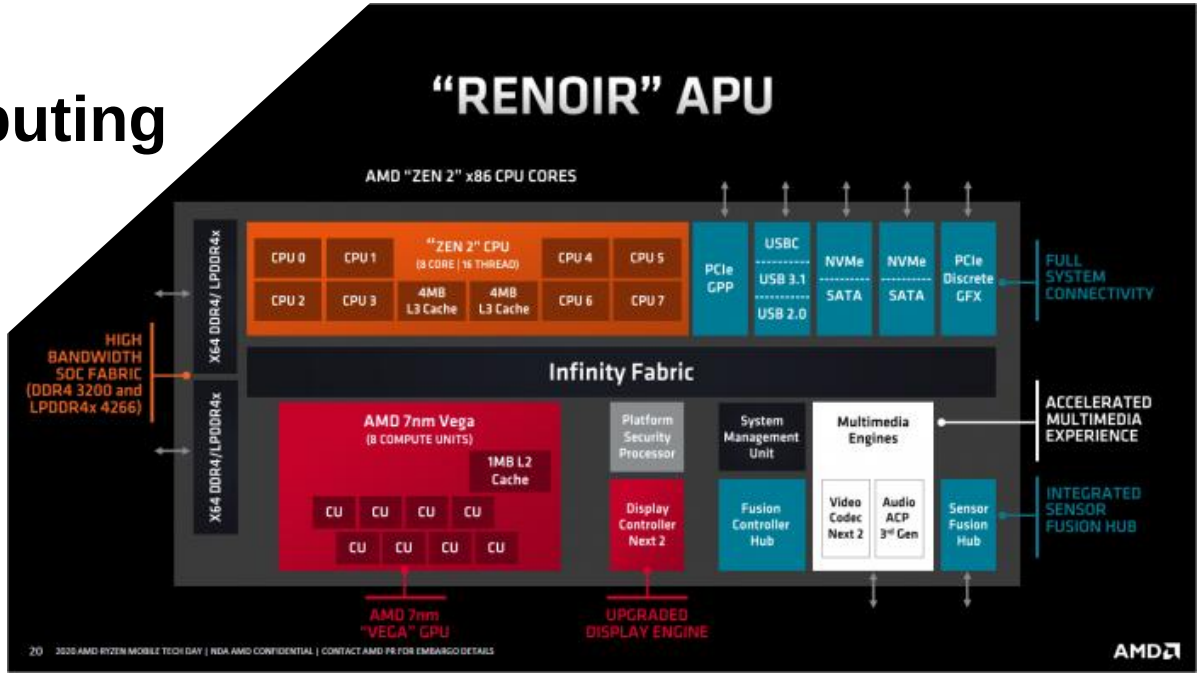
School of Engineering
InES Institute of Embedded Systems

Improving System Performance

Computer Engineering 1

Agenda

- **Motivation**
- **Aspects of Optimization**
- **Bus Architectures**
- **Instruction Set Architectures (ISA)**
- **Pipelining**
- **Parallel Computing**



Motivation

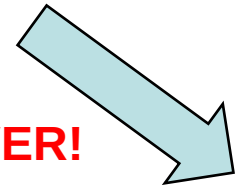
School of Engineering
InES Institute of Embedded Systems

- **Intel Pentium E5800**
  - 3.2 GHz
  - 2 Cores
  - 2 MB cache
  - Released Q4 2010
  - **PassMark: 1184**
  - **TDP: 65 W**

SPEED!

- **Intel Celeron G4930**
  - 3.2 GHz
  - 2 Cores
  - 2 MB cache
  - Q2 2019
  - **PassMark: 2628**
  - TDP: 54 W

LOW POWER!

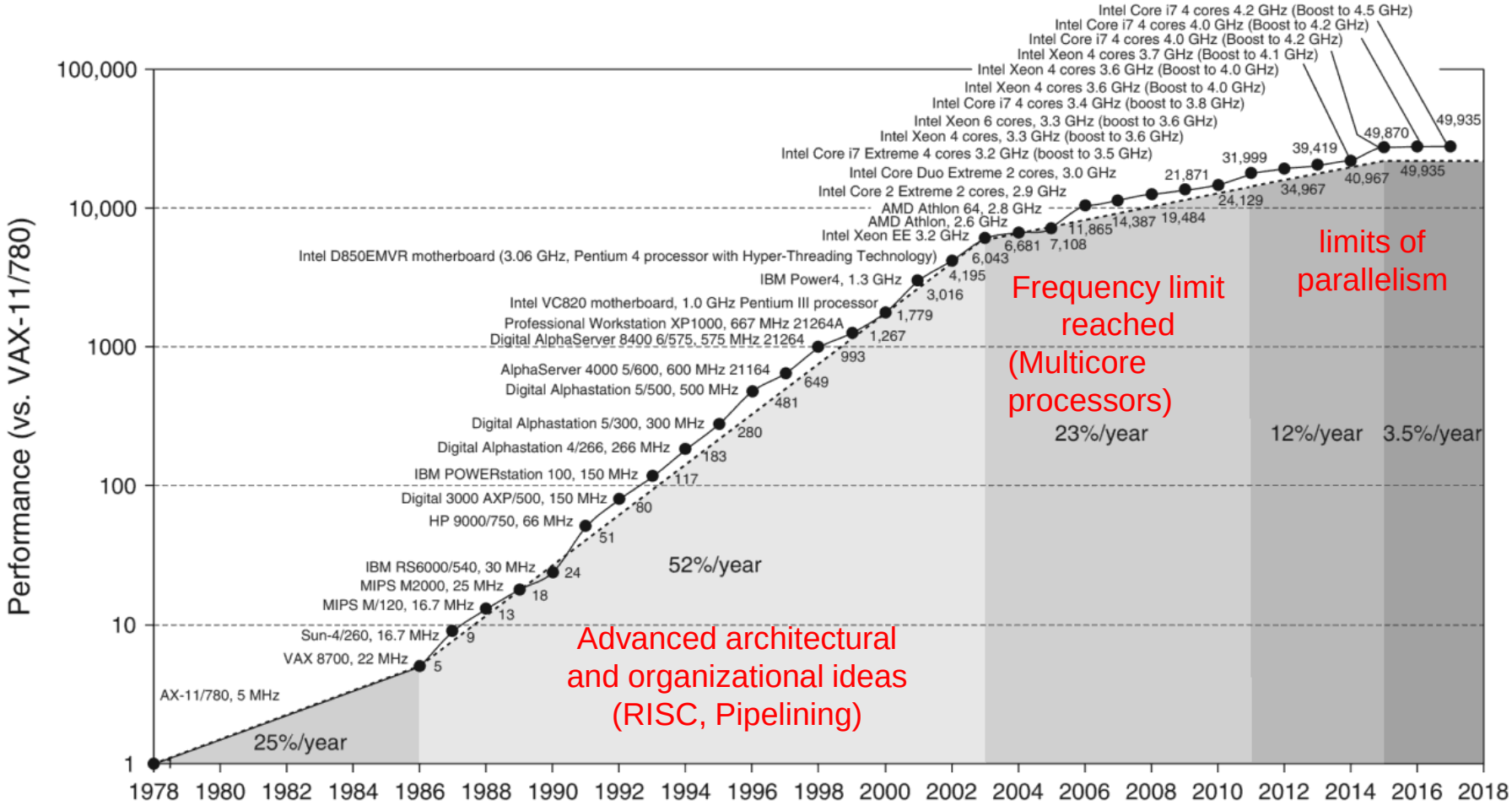


- **AMD Ryzen R1102G**
  - 1.2 GHz
  - 2 Cores
  - 4 MB cache
  - Q1 2020
  - PassMark: ~1600
  - **TDP: 6 W**

Motivation

School of Engineering
InES Institute of Embedded Systems

- **Growth in processor performance**



Source: John L. Hennessy, David A. Patterson (2017). Computer Architecture: A Quantitative Approach. 6th edition.

Learning Objectives

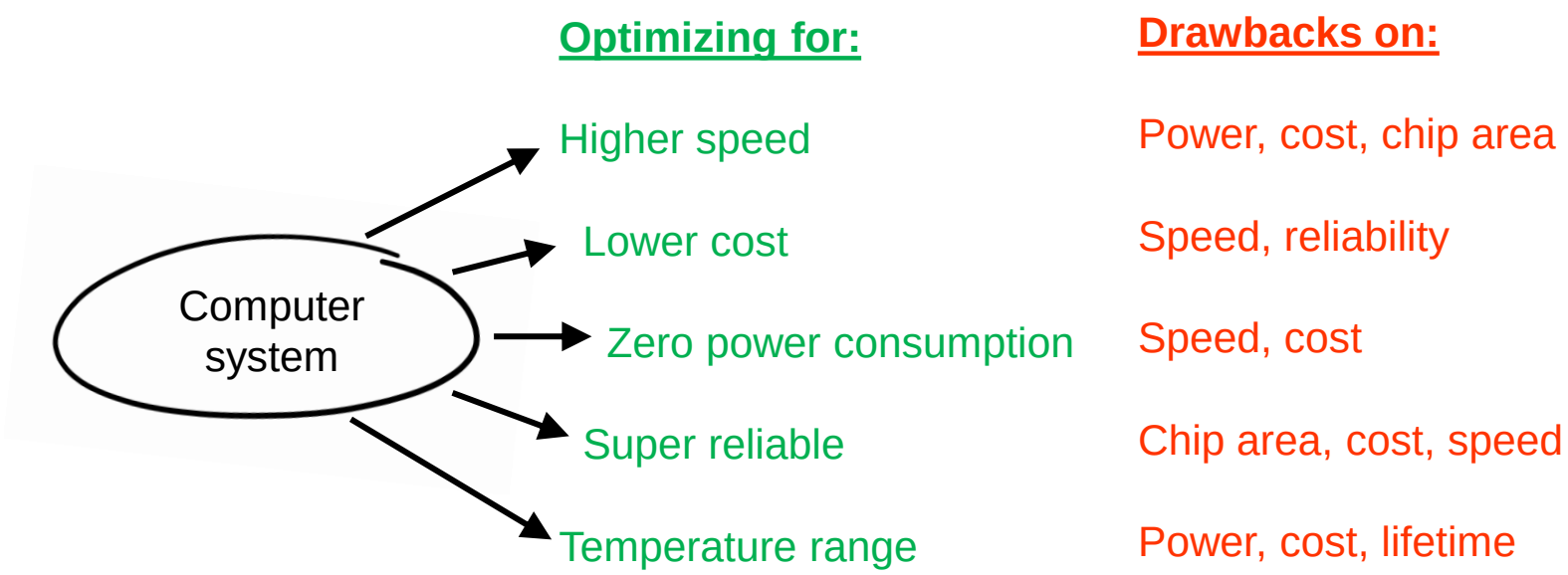
School of Engineering
InES Institute of Embedded Systems

- **At the end of this lesson you will be able**
  - to explain different types of bus architectures
  - to understand the difference between von Neumann and Harvard architecture
  - to understand RISC and CISC paradigms
  - to describe the idea of pipelining
  - to calculate processing performance improvement through pipelining
  - to describe the basics of parallel computing

Aspects of Optimization

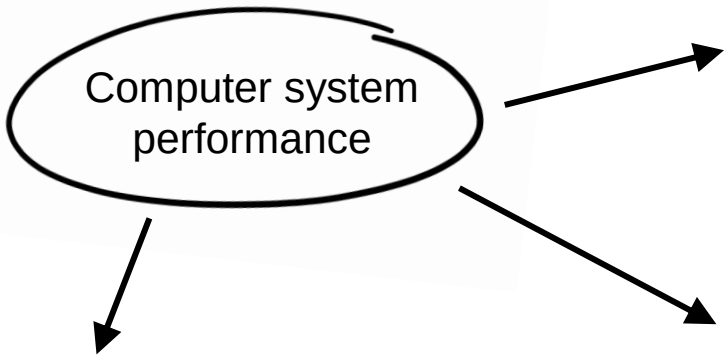
School of Engineering
InES Institute of Embedded Systems

- **‘Wishlist of Optimization’**



- This lecture: Increasing system/CPU speed
- Some other topics: Covered in ‘Microcomputer Systems 1’ (MC1)

How to Increase System Speed?



External factors:

- Better Compilers
- Better Algorithms

CPU improvements:

- Clock Speed
- Cache Memory
- Multiple Cores / Threads
- Pipelined Execution
- Branch Prediction
- Out of Order Execution
- Instruction Set Architecture

System level factors:

- Special Purpose Units
  e.g. Crypto, Video, AI
- Multiple Processors
- Bus Architecture
  e.g. von Neumann / Harward
- Faster components (e.g. SSDs, 1000Base-T, etc.)

Bus Architecture

School of Engineering
InES Institute of Embedded Systems

How is the connection between the CPU and memory organized?

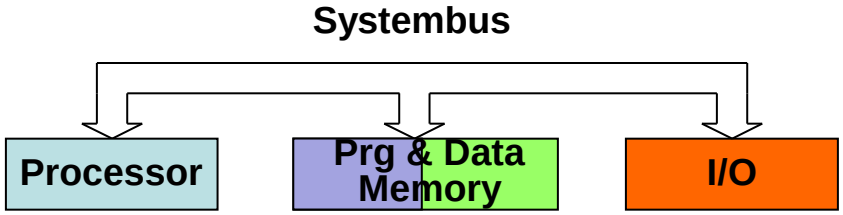
A single system bus: **simple, cheaper** **slower**

Multiple system buses: **faster** **more complexity**

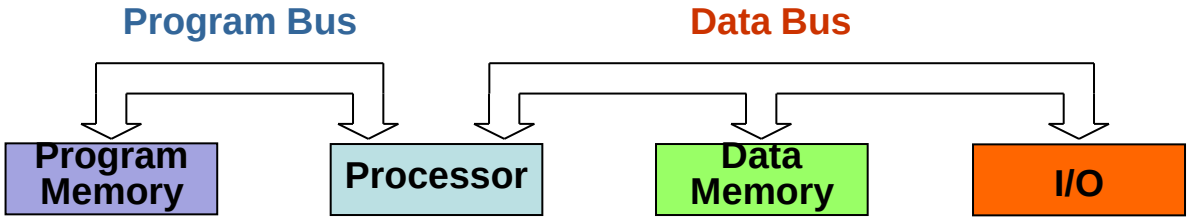
Bus Architecture

School of Engineering
InES Institute of Embedded Systems

- **von Neumann Architecture**
  - Same memory holds program and data
  - Single bus system between CPU and memory



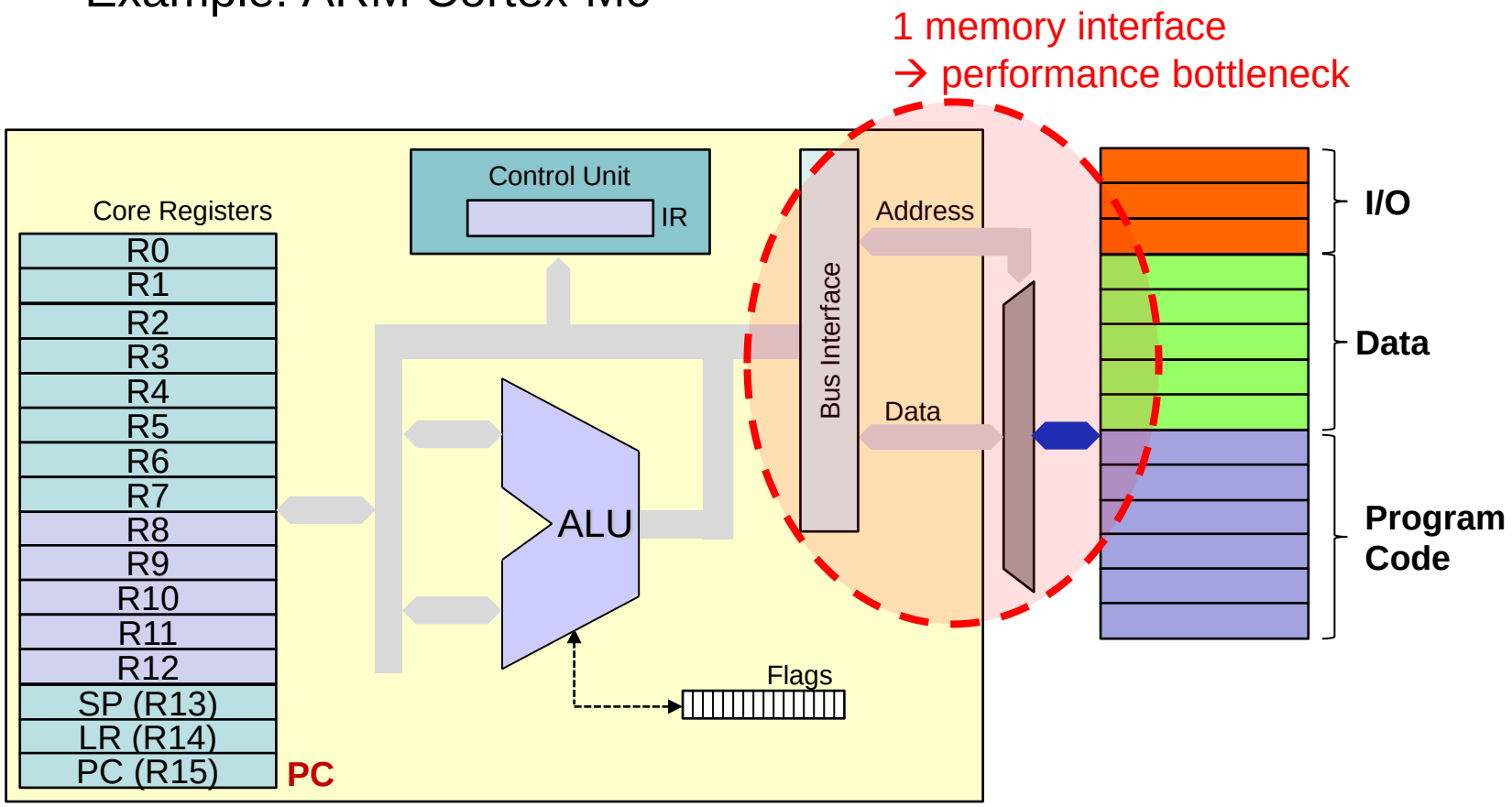
- **Harvard Architecture**
  - “Mark I” at Harvard University (Howard Aiken, 1939-44)
  - Separate memories for program and data
  - Two sets of address/data buses between CPU and memory



Bus Architecture

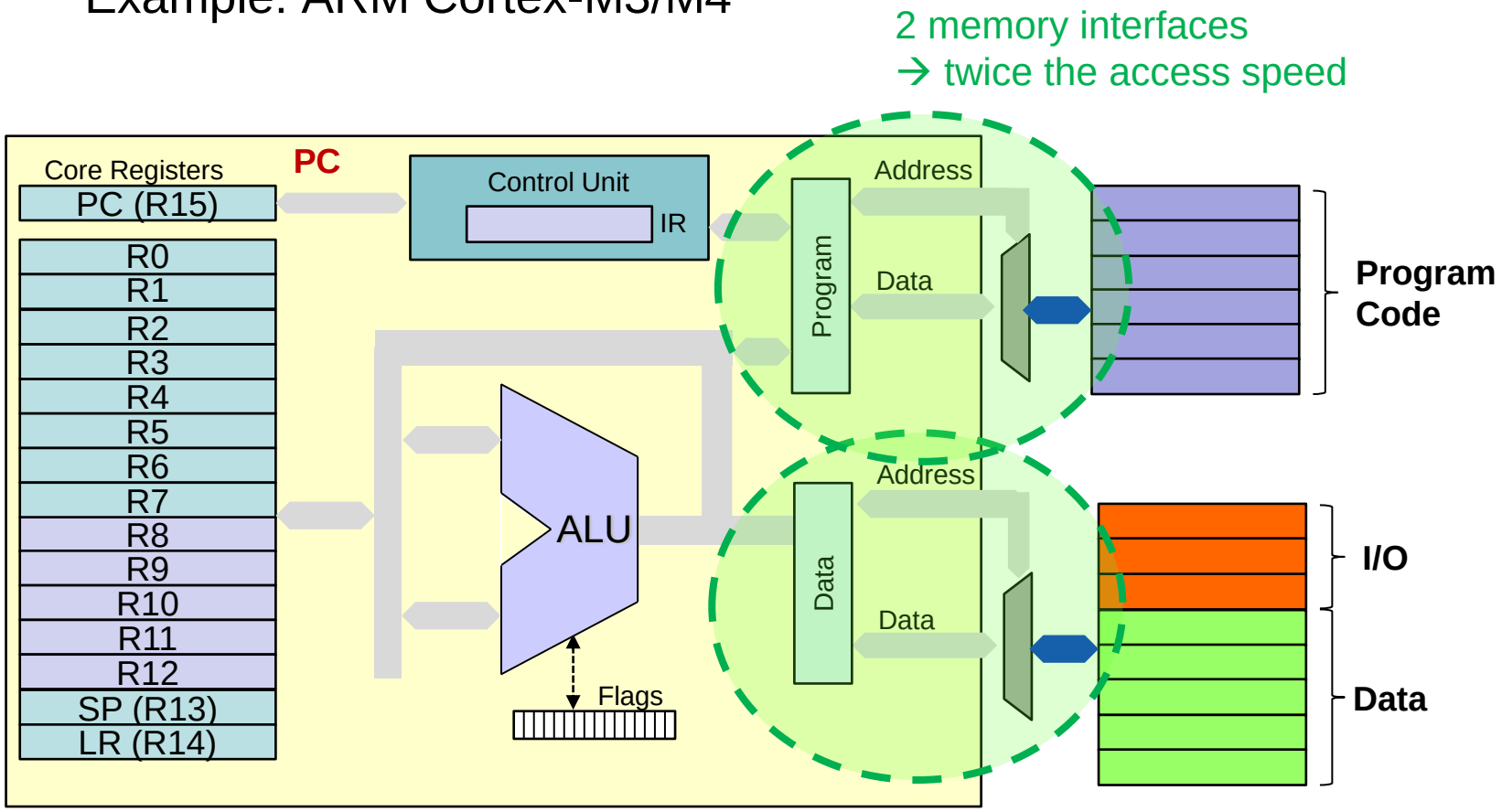
School of Engineering
InES Institute of Embedded Systems

- **von Neumann Architecture**
  - Example: ARM Cortex-M0



Bus Architecture

- **Harvard Architecture**
  - Example: ARM Cortex-M3/M4



Instruction Set Architecture

School of Engineering
InES Institute of Embedded Systems

Powerful, complex instructions?

or

Simple, highly optimizable instructions?

(or something in between?)

Instruction Set Architecture

School of Engineering
InES Institute of Embedded Systems

- **First CPUs**
  - Small address bus, small data bus (8 bit)
  - Few registers, no cache, pipelining, co-processors, multi-core
  - Machine code as finite state machine: Shift, multiply, etc. in loops

- **Consequences for machine code**
  - Differing complexity
  - Different length
  - Variable execution time
  - Individual addressing / arguments

  } for different instructions

→ **More and more complex instructions (CISC)**

Instruction Set Architecture

School of Engineering
InES Institute of Embedded Systems

- **John Cocke (IBM, 1974) proved that**
  - 80% of work was done using only 20% of the instructions
  - Most (complex) instructions were not used at all by compilers

- **RISC (Reduced Instruction Set Computer)**
  - Few instructions, unique instruction format
  - Fast decoding, simple addressing
  - less hardware → allows higher clock rates
  - more chip space for registers (up to 256!)
  - Load-store architecture reduces memory accesses, CPU works at full-speed on registers

Acorn Archimedes A3000
Source: Wikipedia Commons

Remark: Word "CISC" was introduced with "RISC"

Instruction Set Architecture

School of Engineering
InES Institute of Embedded Systems

Example: **Balance = Balance + Credit**

- **RISC**
  - Load / Store Architecture
  - Data processing instructions only available on registers

```
LDR   R0,=Credit
LDR   R1,[R0]
LDR   R0,=Balance
LDR   R3,[R0]
ADDS  R2,R1,R3
STR   R2,[R0]
```

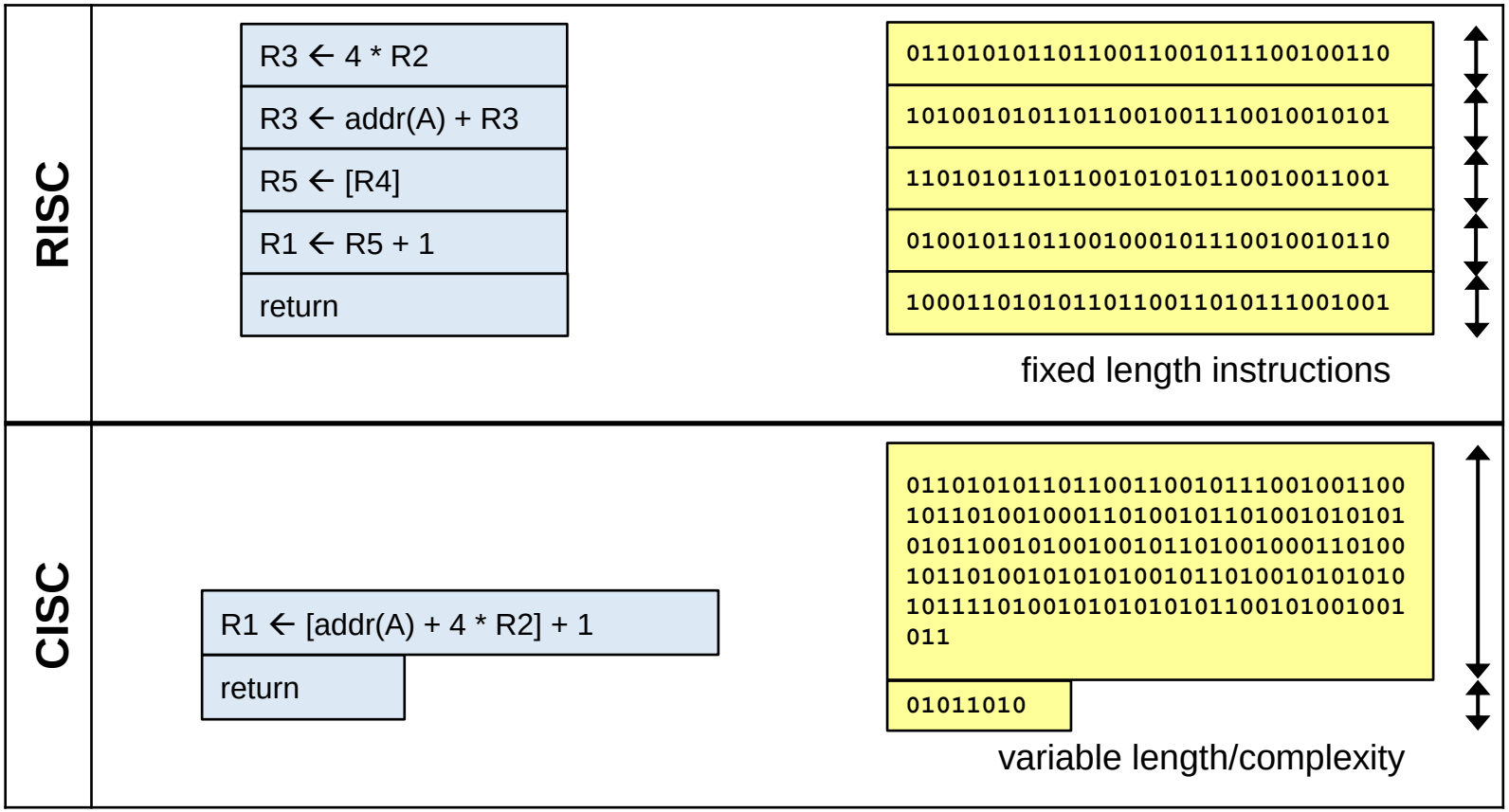
- **CISC**
  - One of the operands of an instruction may directly be a memory location

```
MOV  AX, [Credit]
ADD  [Balance], AX
```

Instruction Set Architecture

School of Engineering
InES Institute of Embedded Systems

- **Instruction / opcode complexity**



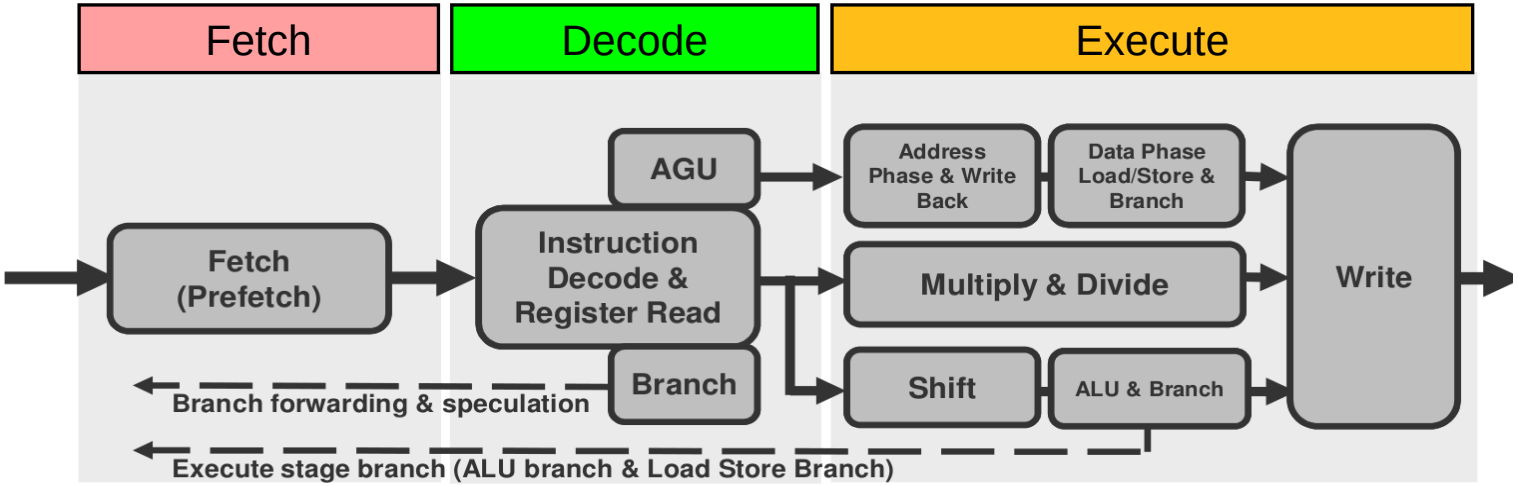
No RISC – No Fun

School of Engineering
InES Institute of Embedded Systems

- **RISC advantages**
  - Simple and fast instruction decoding
  - More registers/cache on silicon area (less memory access)
  - Several data paths possible (superscalar computers)
  - Higher clock frequencies
  - Effective compiler optimization with limited, generic instructions
  - Easy pipelining, shorter pipelines (instruction size / duration)
- **CISC advantages**
  - Less program memory needed with complex instructions
  - Short programs may work faster with less memory accesses
- **CISC and RISC more and more indistinguishable**
  - Modern x86 CPUs convert instructions (CISC to RISC)

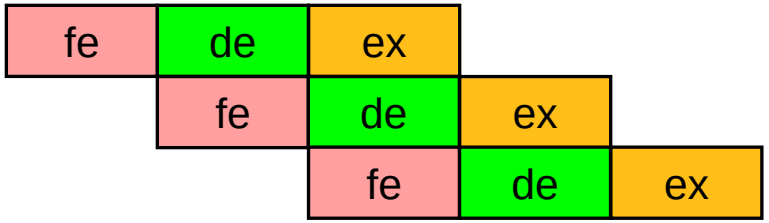
Pipelining

- **ARM Cortex-M3 sequence:**



Source: ARM

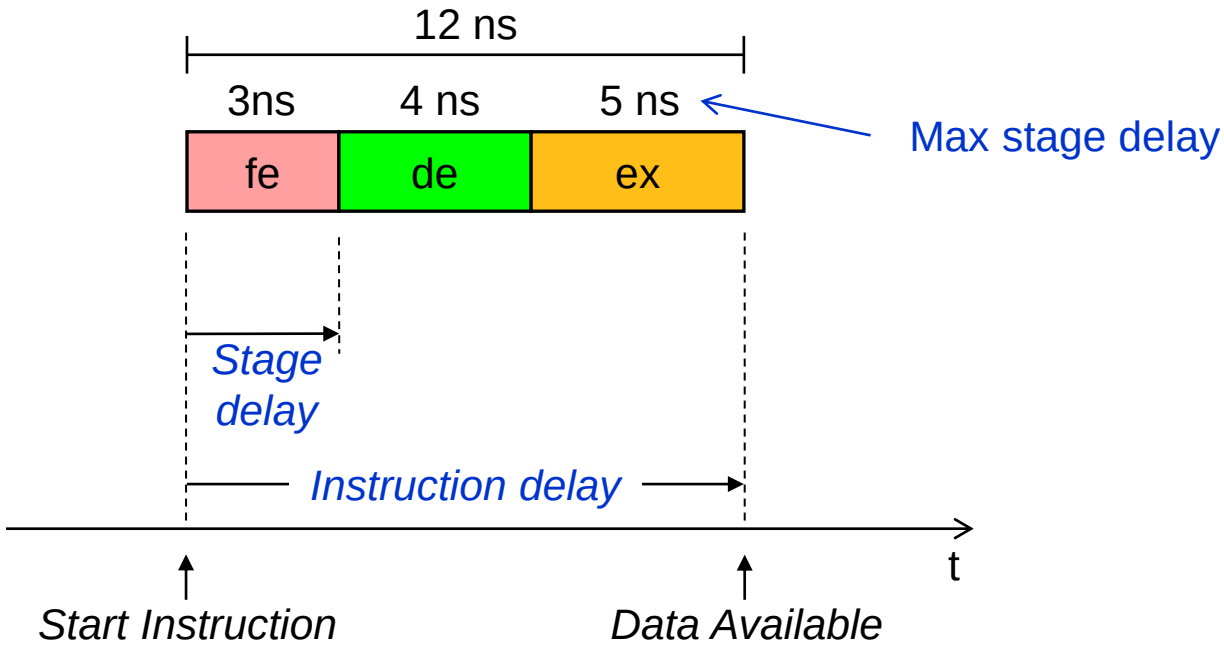
- **Idea: Why not fetch the next instruction, while the current one decodes?**



Pipelining

Timings and definitions

- Example: Cortex-M3
  - **fe: fetch** Read instruction 3 ns[1]
  - **de: decode** Decode instruction, read register or memory 4 ns[1]
  - **ex: execute** Execute instruction, write back result 5 ns[1]

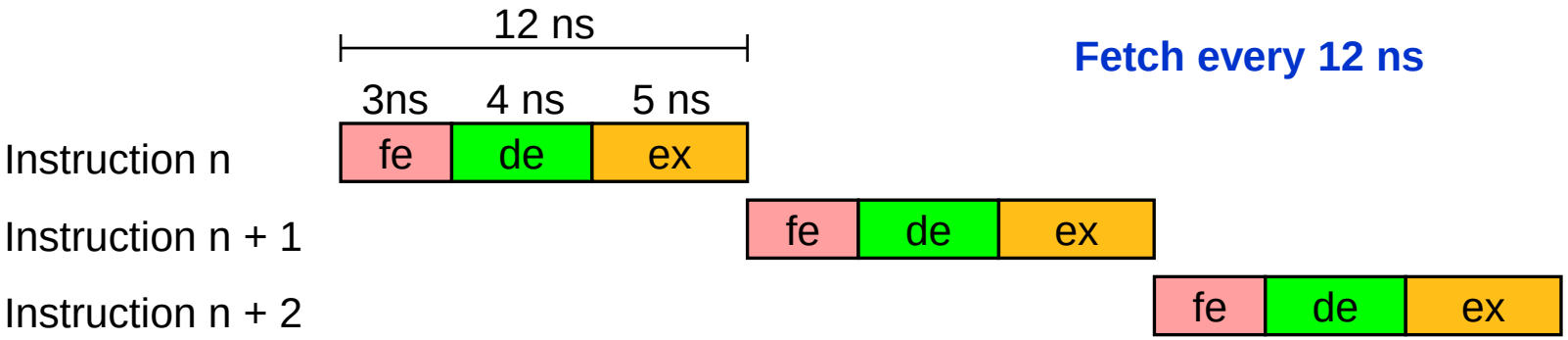


[1]*timings estimated*

Pipelining

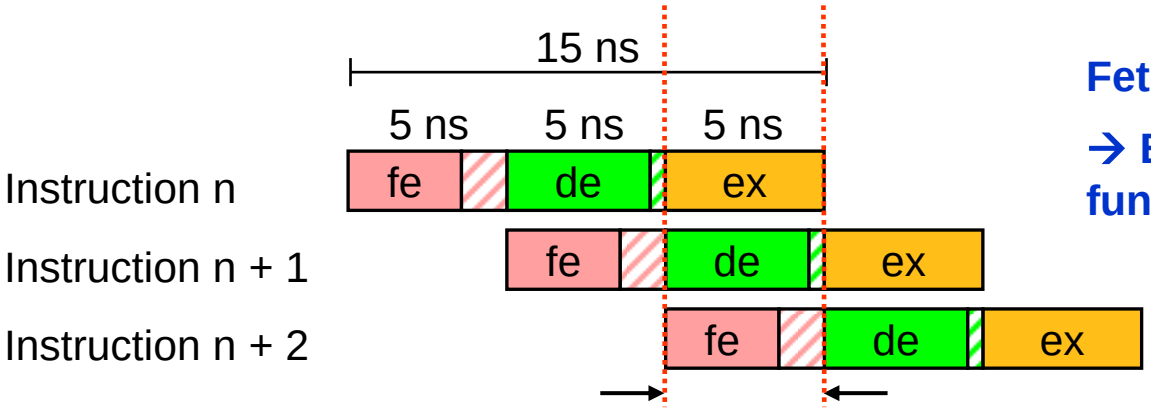
School of Engineering
InES Institute of Embedded Systems

Sequential execution



Fetch every 12 ns

Pipelined execution



Fetch every 5 ns

→ Better usage of functional units

Longest stage (ex) defines pipeline cycle: 5 ns

Pipelining

School of Engineering
InES Institute of Embedded Systems

- **Instructions per second**
  - Without pipelining

$$\textit{Instructions per second} = \frac{1}{\textit{Instruction delay}}$$

  - With pipelining
    - Pipeline needs to be filled first
    - After filling, instructions are executed after every stage

$$\textit{Instructions per second} = \frac{1}{\textit{Max stage delay}}$$

- **Example Cortex-M3**
  - Without pipelining: 1/12ns = 83,3 million instructions per second
  - With pipelining: 1/5ns = 200 million instructions per second

Pipelining

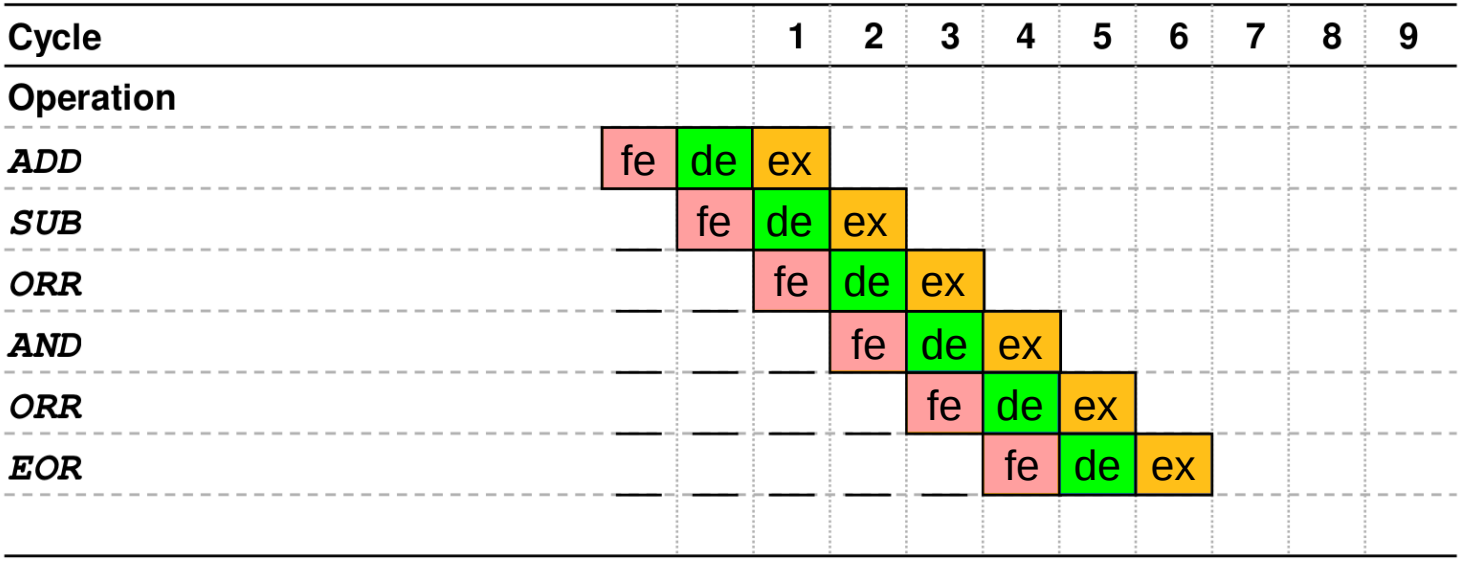
School of Engineering
InES Institute of Embedded Systems

- **Advantages**
  - All stages are set to the same execution time
  - Massive performance gain
  - Simpler hardware at each stage allows for a higher clock rate
- **Disadvantages**
  - A blocking stage blocks whole pipeline
  - Multiple stages may need to have access to the memory at the same time
- **General**
  - Number of execution stages is design decision
  - Typically 3 (ARM Cortex M4) – 20 stages (Pentium 4)

Pipelining

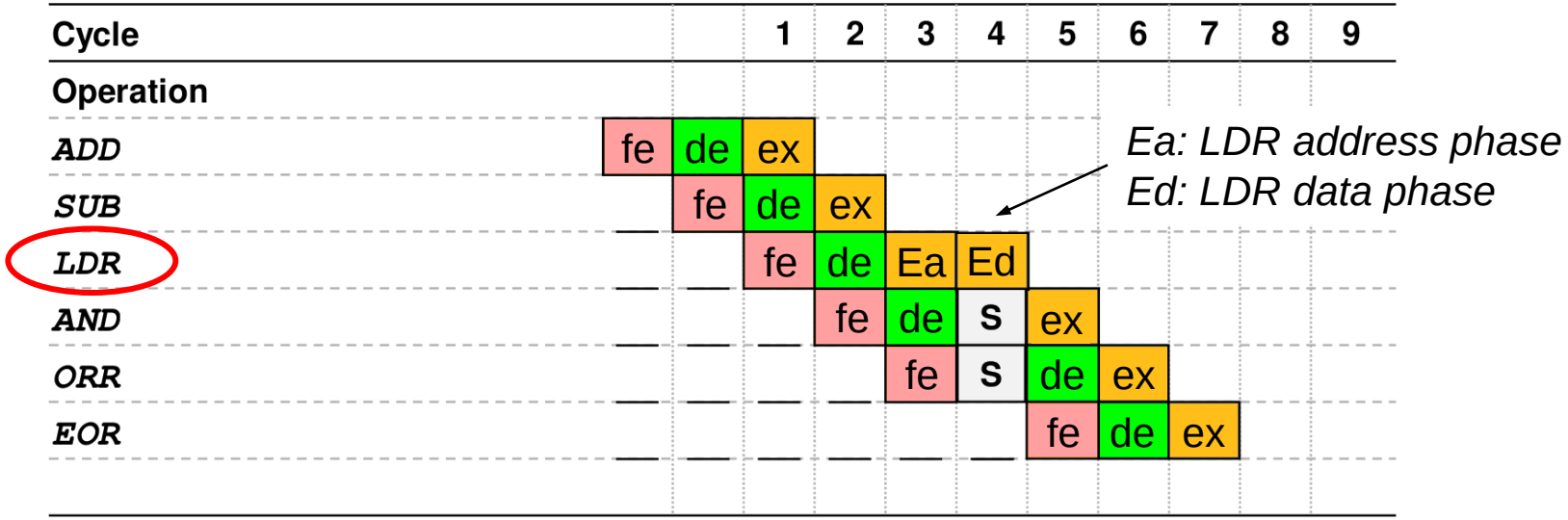
School of Engineering
InES Institute of Embedded Systems

- **Optimal Pipelining**
  - All operations here are on registers (single cycle execution)
  - In this example it takes 6 clock cycles to execute 6 instructions
  - Clock cycles per instruction (CPI) = 1



Pipelining

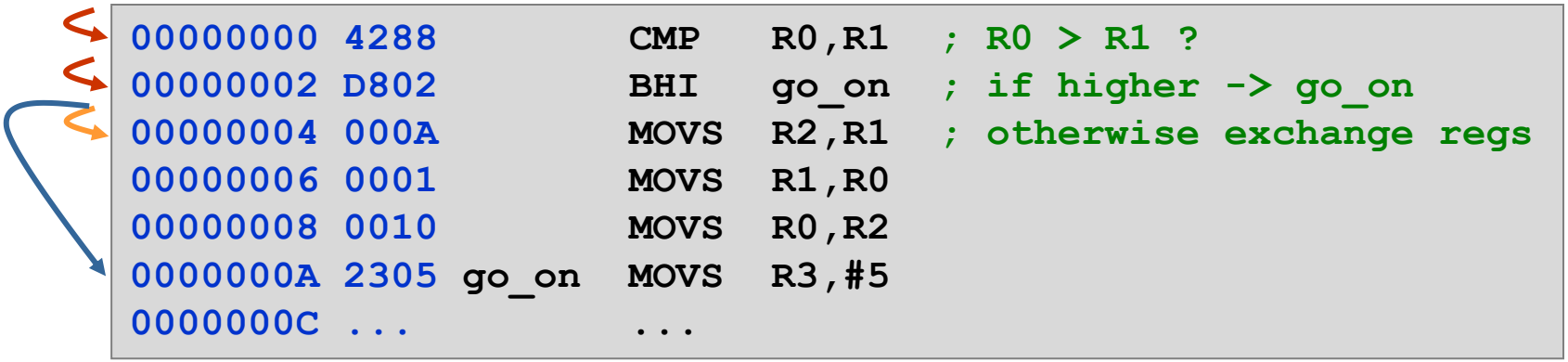
- **Special situation: LDR**
  - In this example it takes 7 clock cycles to execute 6 instructions
  - Read cycle must complete on the bus before LDR instruction can complete
  - Next 2 instructions must wait one pipeline cycle (S = stall)
  - Clock cycles per instruction (CPI) = 1.2



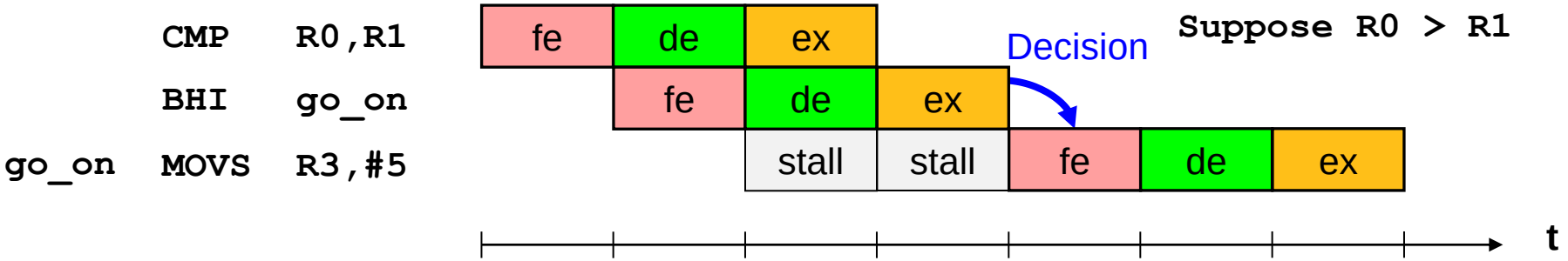
Pipelining

School of Engineering
InES Institute of Embedded Systems

- **Control Hazards**
  - Branch / jump decisions occur in stage 3 (ex)
  - Worst case scenario – conditional branch taken:



```
00000000 4288          CMP   R0,R1   ; R0 > R1 ?
00000002 D802          BHI   go_on   ; if higher -> go_on
00000004 000A          MOVS  R2,R1   ; otherwise exchange regs
00000006 0001          MOVS  R1,R0
00000008 0010          MOVS  R0,R2
0000000A 2305 go_on    MOVS  R3,#5
0000000C ...           ...
```



Pipelining

School of Engineering
InES Institute of Embedded Systems

- **Reduce control hazards**
  - Loop fusion reduces control hazards
  - Simple example for illustration

```
/* Two loops → double number of control hazards*/
for (i = 0; i < N; i++) {
        a[i] = 1/b[i] * c[i];
}
for (i = 0; i < N; i++) {
        d[i] = a[i] + c[i];
}

/* Better performance: Fusion into one loop */
for (i = 0; i < N; i++) {
        a[i] = 1/b[i] * c[i];
        d[i] = a[i] + c[i];
}
```

2*N Conditional Branches

N Conditional Branches

Pipelining

School of Engineering
InES Institute of Embedded Systems

- **Ideas to further improve pipelining:**

  - Branch prediction:
    - Store last decision made for each conditional branch
      → probability is high that the same decision is taken again

  - Instruction prefetch:
    - Fetch several instructions in advance
      → better use of the system bus
      → possibility of ‘Out of Order Execution’

  - Out of Order Execution:
    - If one instruction stalls (e.g. a LDR from slow memory), it might be possible to already execute the next instruction

Pipelining

- **Limits of optimization**
  - Complex Optimizations → severe security problems (e.g. Out of Order Execution)
  - Instructions (speculatively) executed, that would throw access violations under ‘In Order’ circumstances.
  - ‘Meltdown’ and ‘Spectre’ attacks: allow a process to access the data of another process.

Parallel Computing

School of Engineering
InES Institute of Embedded Systems

Streaming/Vector Processing

- One instruction processes multiple data items simultaneously

Multithreading

- Multiple programs/threads share a single CPU

Multicore Processors

- One processor contains multiple CPU cores

Multiprocessor Systems

- A computer system contains multiple processors

Parallel Computing

School of Engineering
InES Institute of Embedded Systems

- **Instructions working on more than one data item**

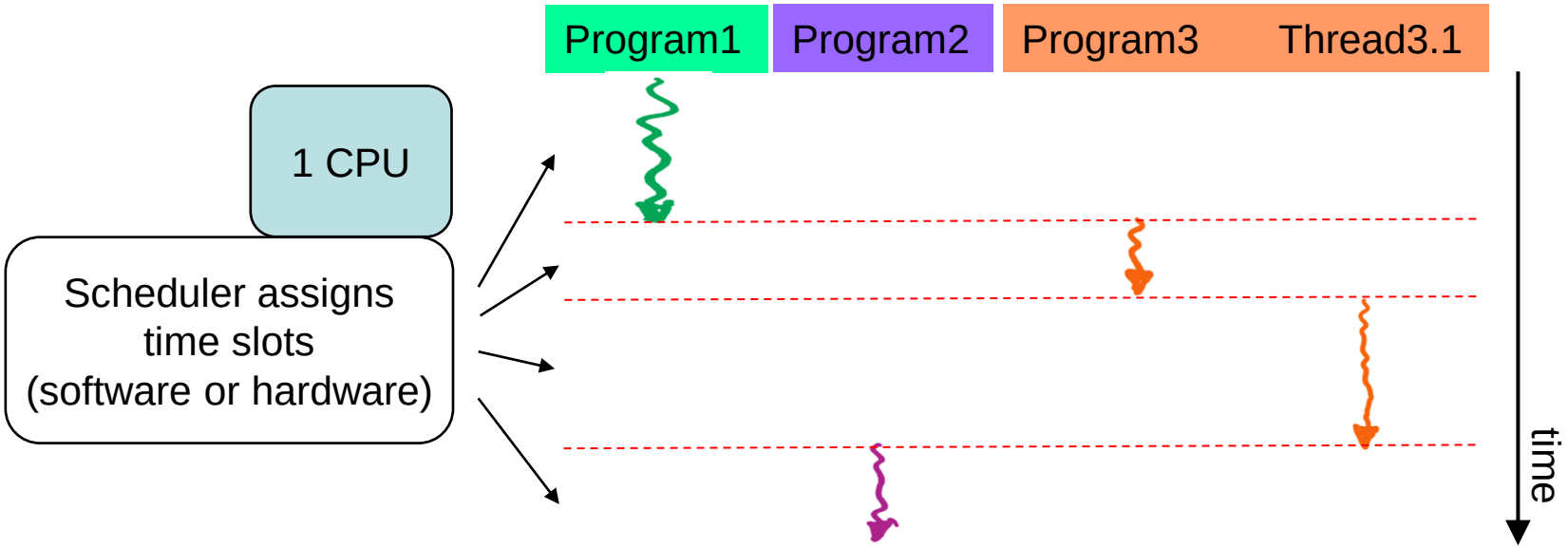
		Data streams	
		Single	Multiple
Instruction streams	Single	**SISD**: Single processor with one instruction, data stream	**SIMD**: Data Vectors, all data streams react to one instruction
	Multiple	(no examples)	**MIMD**: Multiprocessor with SIMD-instructions

Source: David A. Patterson, John L. Hennessy

- **SIMD examples (x86):**

  MMX (Multimedia Extension), AMD 3D-Now,
  SSE (Streaming SIMD Extensions),
  AVX (Advanced Vector Extensions)

- **Multithreading**
  - Scheduler: Assigns time slots to programs/threads
  - Programs/threads only **seem** to run in parallel (1 CPU)

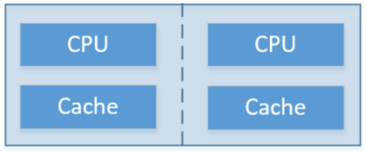


Remark: More on Multithreading in MC1

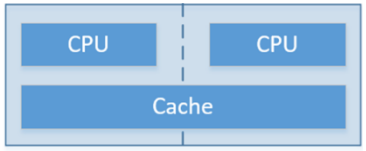
Parallel Computing

School of Engineering
InES Institute of Embedded Systems

- **Parallelism on CPU / processor level:**
  - Multicore processor
    - All on one chip
    - Less traffic (cores integrated on one chip)
    - Possibility to share memories on-chip
    - Cheaper

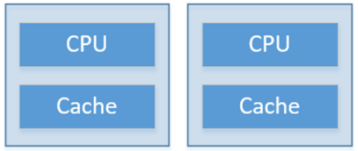


Multi-core



Multi-core with shared Cache

  - Multiprocessor
    - Multiple Chips
    - Longer distances between CPUs
    - More expensive

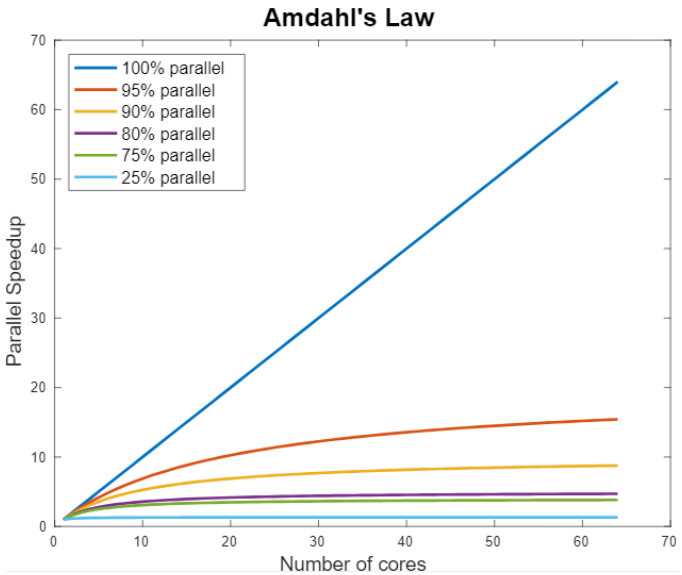


Multiprocessor

Source: https://www.queryhome.com/tech/106320/what-the-difference-between-multicore-and-multiprocessor

Conclusion

- **CPU clock frequency not the only factor**
- **Different ways to increase processor performance**
  - RISC / CISC
  - Pipelining
  - Out-of-order execution
  - Parallel computing
- **Today‘s CPUs combine most of these technologies**
- **Other components with influence**
  - SSD, memory, interfaces, algorithms …



Source:https://researchcomputingservices.github.io/parallel-computing/02-speedup-limitations/