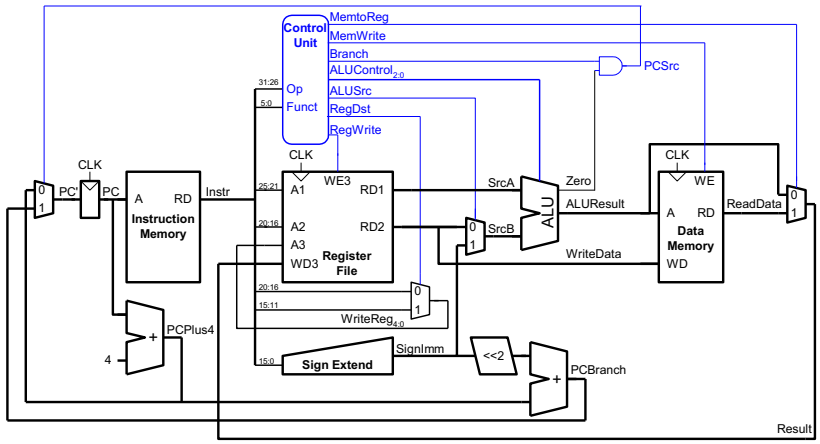
Performance: Basic Concepts

Jan Reineke
Universität des Saarlandes

Roadmap: Computer architecture



1. Combinatorial circuits: Boolean Algebra/Functions/Expressions/Synthesis
2. Number representations
3. Arithmetic Circuits: Addition, Multiplication, Division, ALU

4. Sequential circuits: Flip-Flops, Registers, SRAM, Moore and Mealy automata
5. Verilog

6. Instruction Set Architecture
7. Microarchitecture

8. **Performance: RISC vs. CISC**, Pipelining, Memory Hierarchy

Performance: Possible definitions

Possible definitions for the performance of a computer:

- User with a single program:
  1/execution time
- Data center with a set of users that each execute a set of programs:
  |programs|/time
- Search engine: set of search queries:
  - |processed queries|/time → throughput
  - 1/average response time → latency

→ Basis for the comparison of different computers/systems

Performance: Latency versus Throughput

Latency = time required to perform a single task

Throughput = number of tasks performed in one time unit

Example: **Assembly line in the automobile industry**

An assembly line used to produce cars.

The production of a single car requires 8 hours.

Each day, 960 cars can be produced.

→ Latency = 8 hours

→ Throughput = 960 cars / day or 40 cars / hour

Depending on the situation: optimize latency or throughput.

Execution time of a program

Different **execution time** concepts:

Execution from *start* to *finish*:

- Response time
- Includes waiting time for input/output and preemption by other programs
  → also depends on scheduling (*more about this later*)

Time that the processor actually spent executing the program:

- Processor time (also CPU time)
  → depends primarily on the microarchitecture (*now*)

Processor time

Processor time
= number of processor cycles / clock speed
= number of processor cycles * cycle time

Assumption: number of cycles per instruction is constant

Number of processor cycles
= Number of executed instructions * Cycles per instruction

Thus:

Processor time = Number of executed instructions
* Cycles per instruction
* Cycle time

How to increase performance?

Processor time = Number of executed instructions
* Cycles per instruction
* Cycle time

1. Reduce cycle time/increase clock speed
   technological improvements, microarchitecture
2. Reduce number of executed instructions
   Compiler optimizations
3. Reduce cycles per instruction
   Mikroarchitecture (e.g. pipelining, memory hierarchy)

Instruction set influences all 3 factors (but 2. in particular)

1. Technological developments: Moore's Law

Observation by Gordon Moore 1965:

"Number of transistors per area doubles every year"

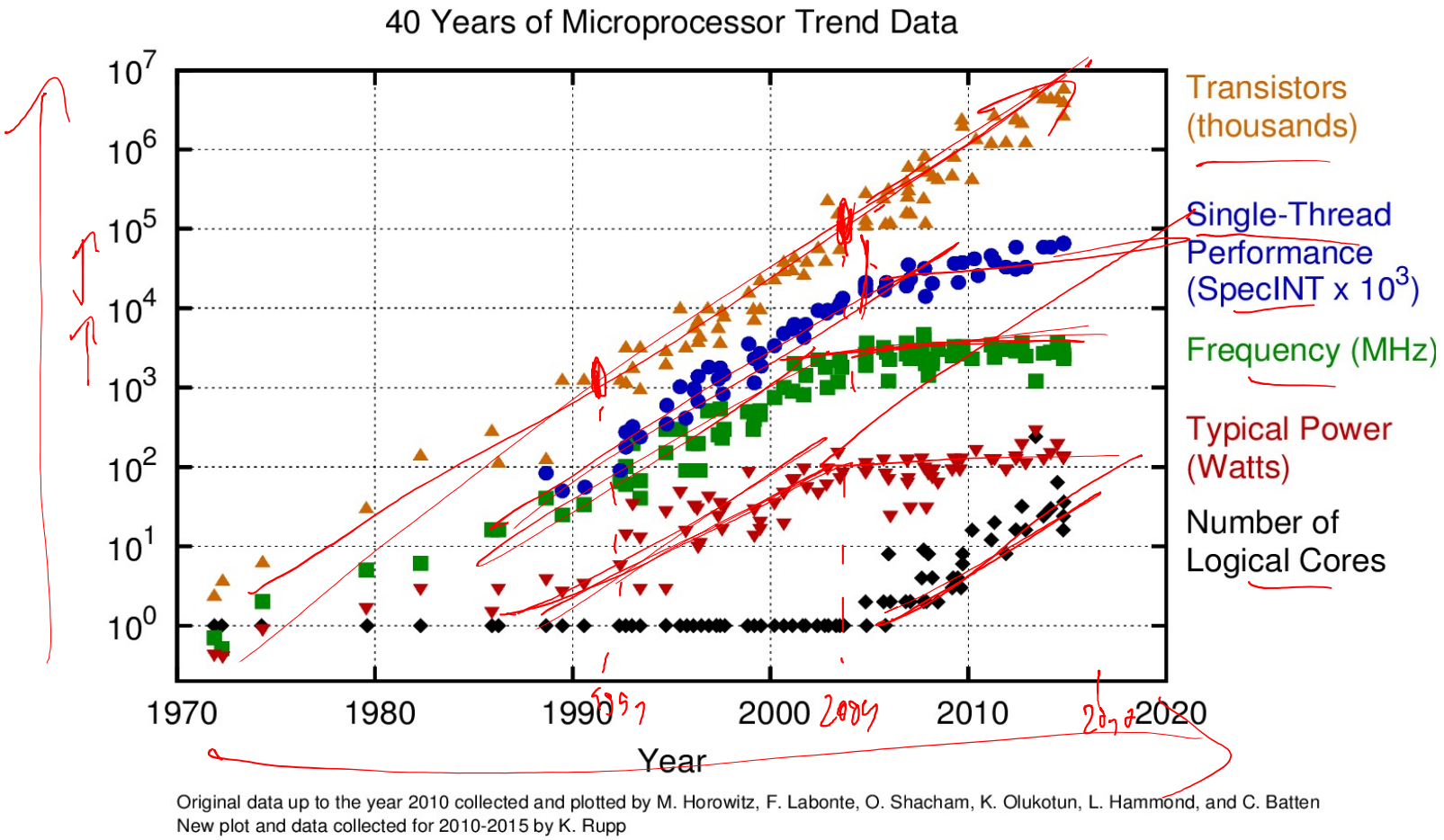
Gordon Moore (1929-)
Intel co-founder

Revision 1975:

"Number of transistors per area doubles every **two** *years"*

No law of nature, but rather empirical observation.

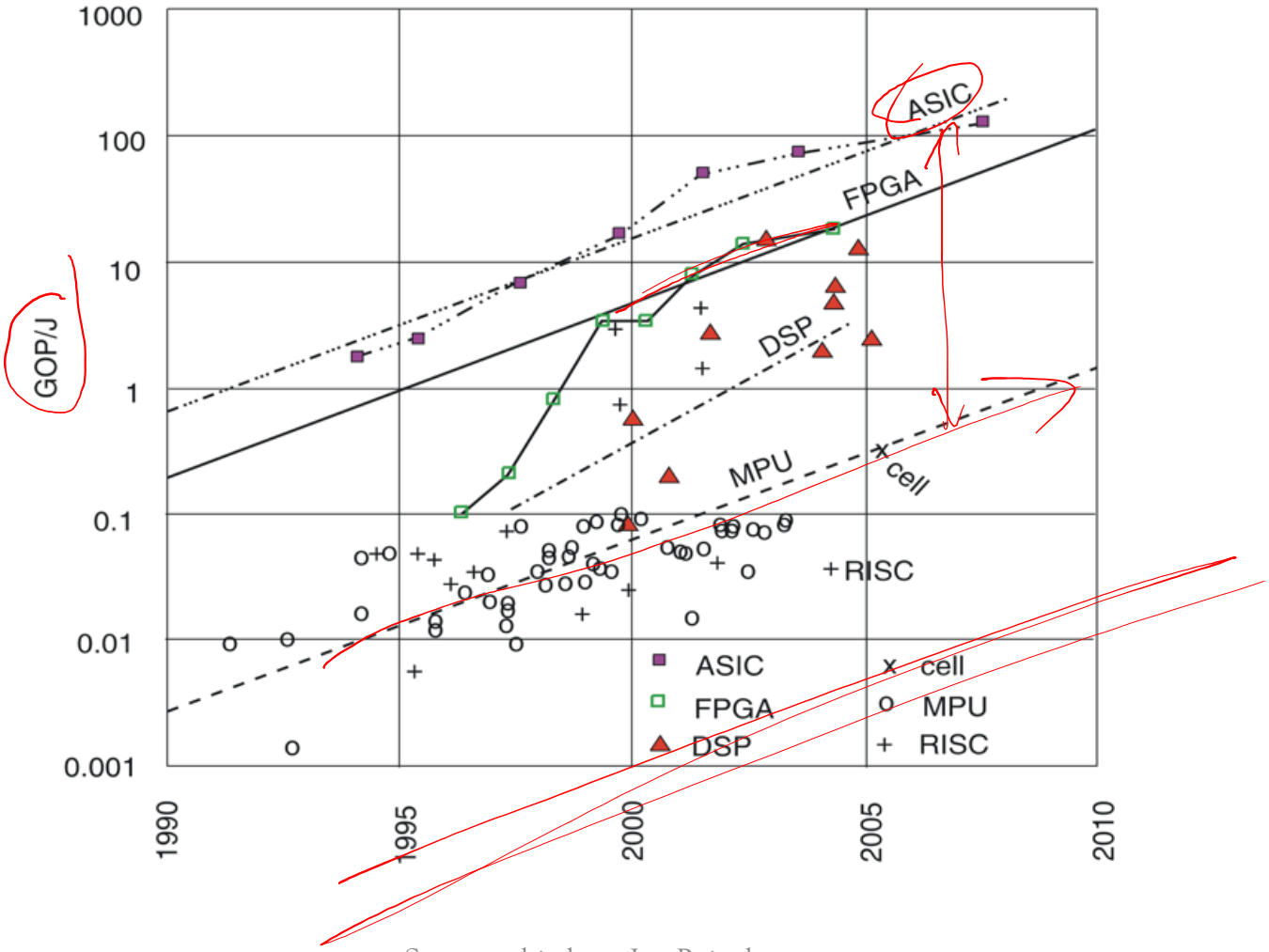
1. Technological developments



Increasing power consumption limits increase of clock speed.

→ Explains transition towards multicores

1. Technological developments: Excursion Energy efficiency of different technologies



2. Number of executed instructions: Instruction Set Architecture

- Loop bounds and recursion depth are fixed (independently of ISA)
- **Reduced Instruction Set Computer (RISC)**, e.g. **MIPS**, SPARC, PowerPC, ARM
  - few instructions, few addressing modes, fixed instruction length, load/store architecture
  - a lot of space for registers (in instruction encoding)
- **Complex Instruction Set Computer (CISC)**, e.g. Motorola 68000, Intel x86
  - complex instructions, many addressing modes, variable instruction length
  - little space for registers (in instruction encoding)

Example: MIPS addressing modes

- Immediate addressing
- Register addressing
- Base addressing
- PC-relative addressing
- Pseudodirect addressing

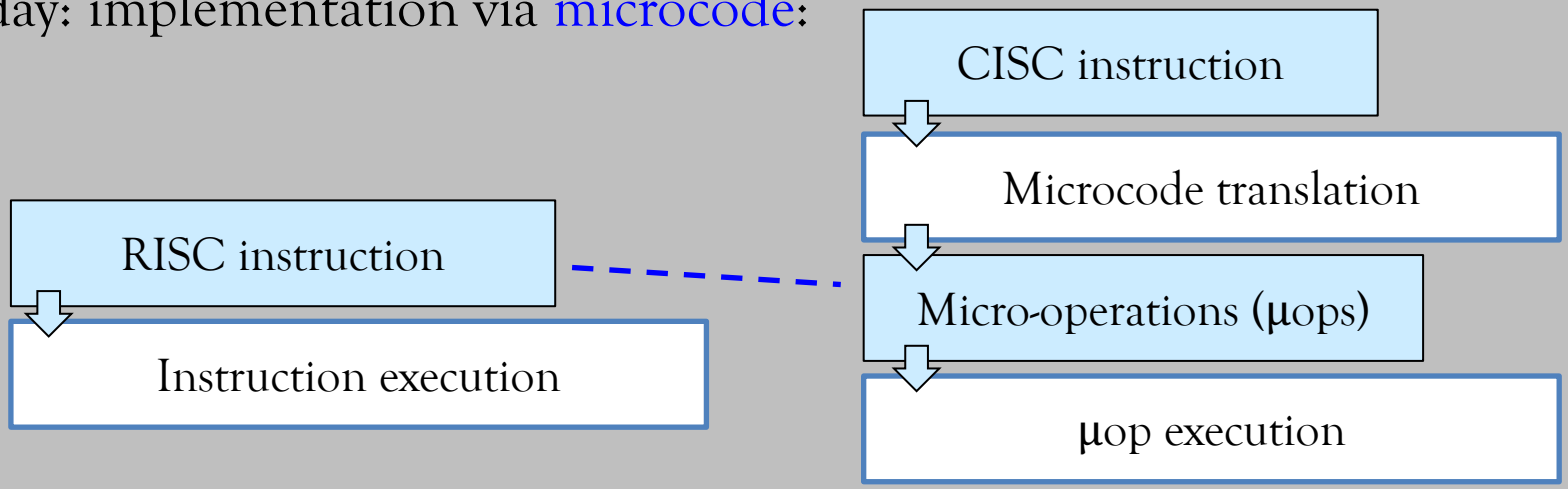
Example: Motorola 68000 addressing modes

1. Data register direct
2. Address register direct
3. Register indirect
4. Register indirect with post-increment
5. Register indirect with pre-decrement
6. Register indirect with displacement
7. Register indirect with index
8. Absolute short
9. Absolute long
10. PC relative with displacement
11. PC relative with index
12. Immediate
13. Quick Immediate
14. Implied register

2. Number of executed instructions: Instruction Set Architecture

Hennessy and Patterson
Turing Award 2017 for RISC

- **Reduced Instruction Set Computer (RISC)**
  - more executed instructions per program
  - but shorter cycle time
- **Complex Instruction Set Computer (CISC)**
  - fewer executed instructions per program
  - but longer cycle time if implemented natively
  - Today: implementation via microcode:



Details for x86: https://uops.info/

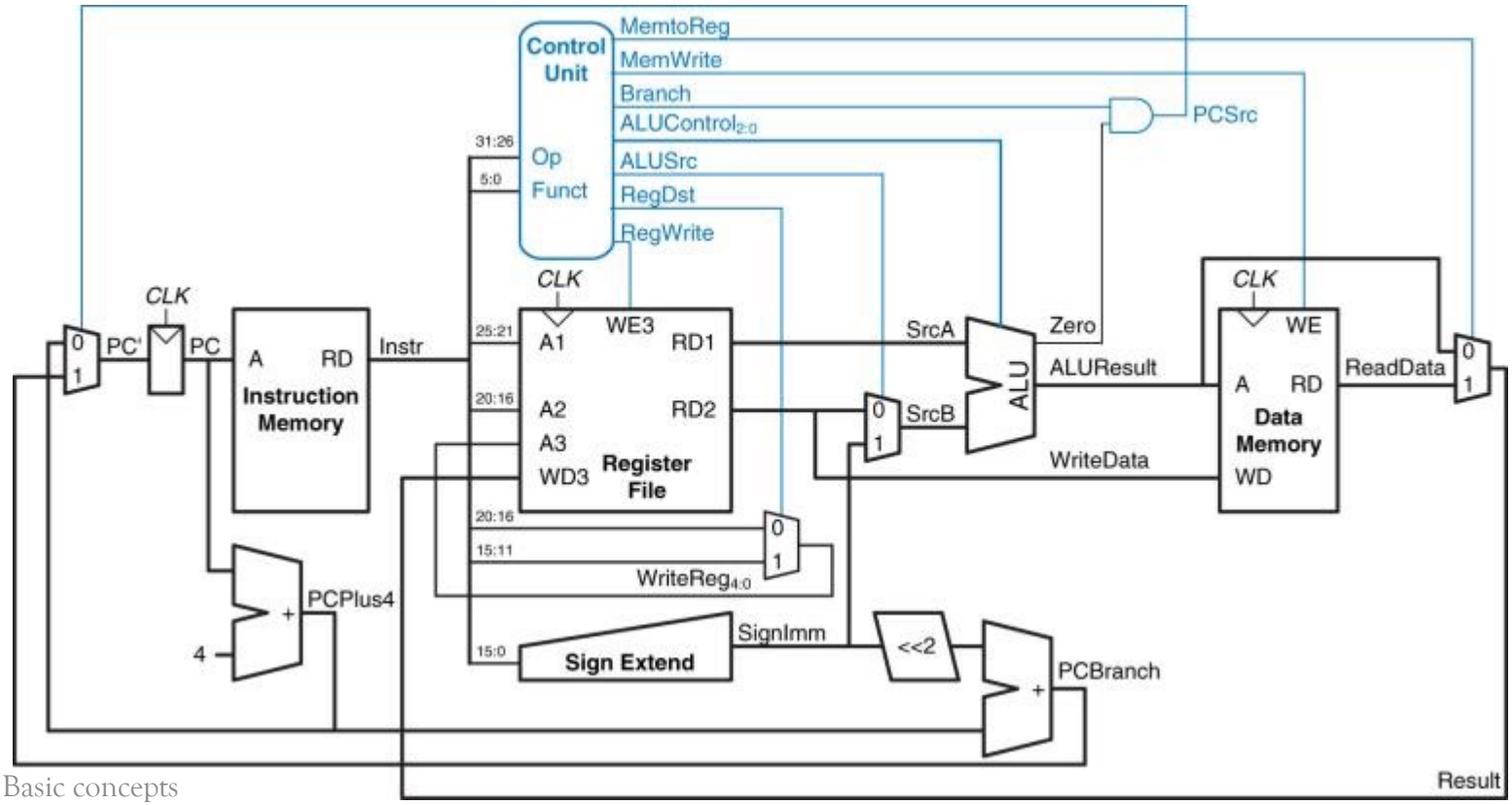
3. Cycles per instruction and cycle time

Single-cycle datapath

- executes each instruction in a single cycle
- **Cycle time** can be **very long**

Multi-cycle datapath

- has **much shorter cycle time** (higher clock speed)
- every instruction requires **multiple cycles**



3. Cycles per instruction and cycle time

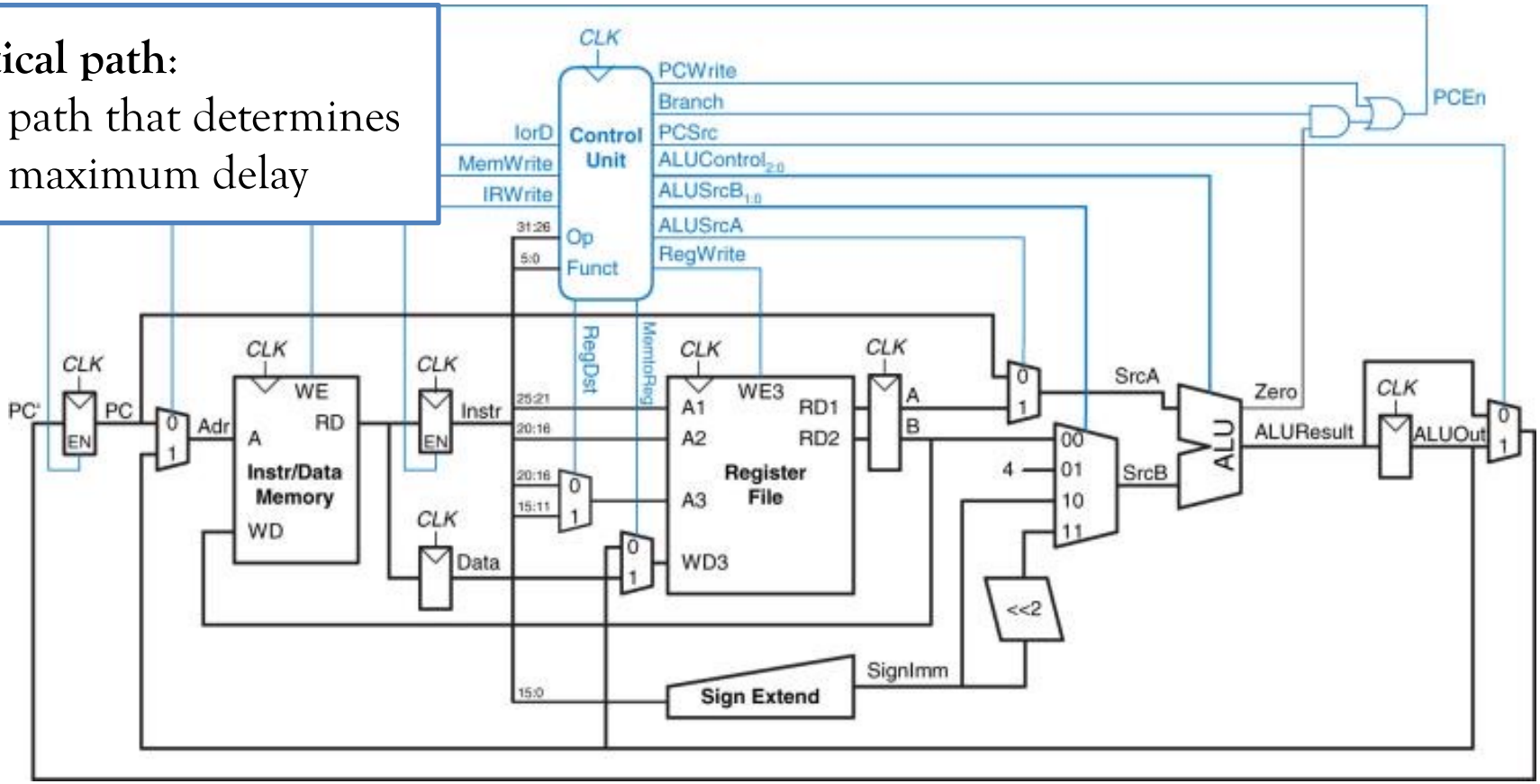
Single-cycle datapath

- executes each instruction in a single cycle
- **Cycle time** can be **very long**

Multi-cycle datapath

- has **much shorter cycle time** (higher clock speed)
- every instruction requires **multiple cycles**

critical path:
the path that determines the maximum delay



Pipelining

Attention: The transition from the single-cycle datapath to the multi-cycle datapath

- **... increases the CPI**!
- ... but it **shortens** the **cycle time**!
- *But*: the product of the two, i.e., the latency of an instruction, is usually slightly higher. **Why?**

CPI = cycles per instruction

Pipelining aims for the "best of both worlds":

- Cycle time is kept short → high clock speed
- An instruction is executed in each clock cycle