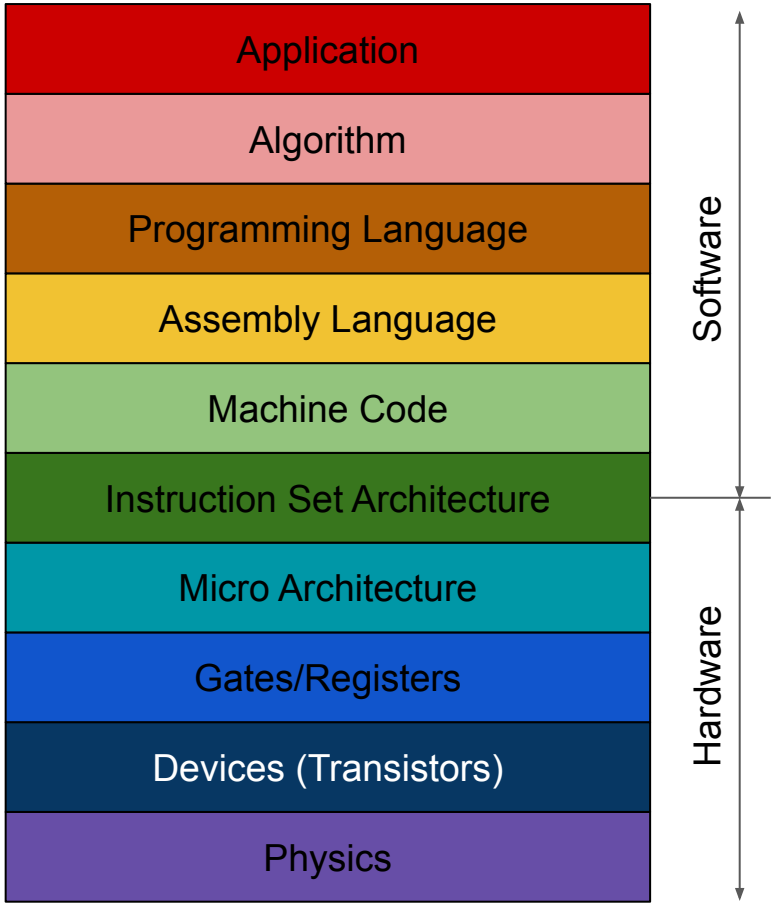
CASS Exercise session 8

Performance & microarchitecture

Improving performance?

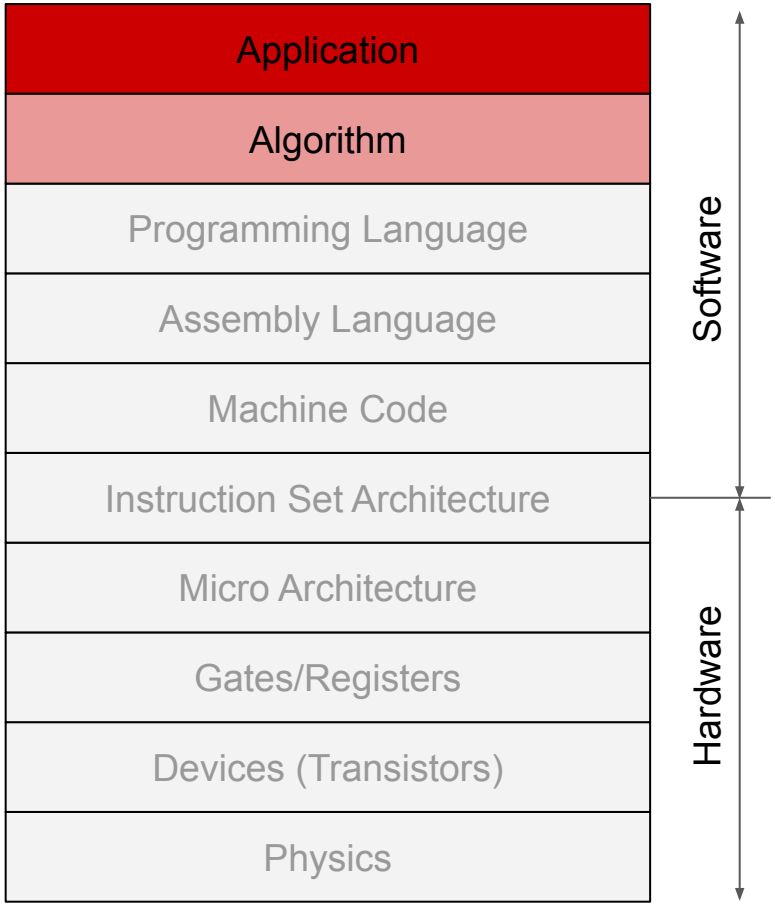
Optimize on *every* abstraction layer



Improving performance?

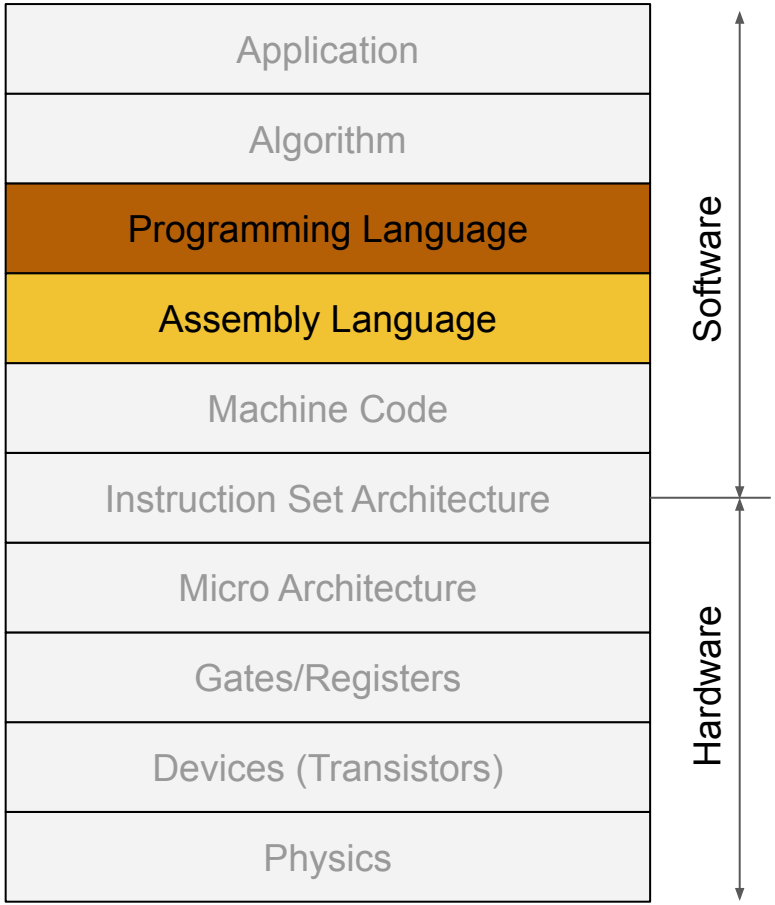
- Choose efficient representations
- Space vs performance trade-offs
- Lazy evaluation
- Memoization
- Choose best algorithm
  - Time complexity
  - Space complexity
  - ...

Not relevant for CASS



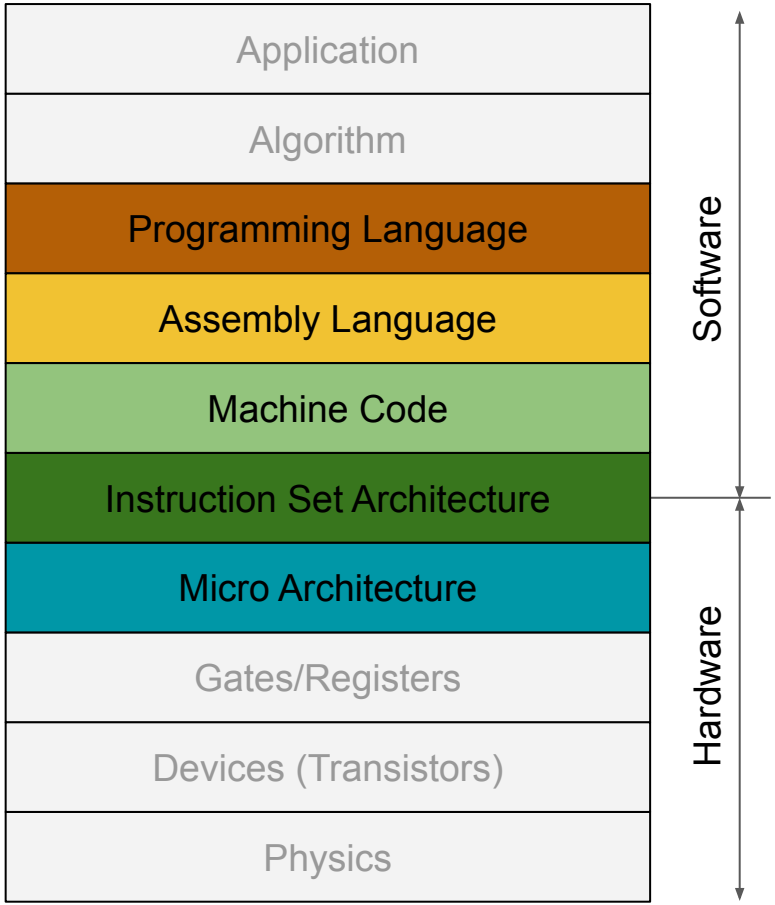
Improving performance?

- Choose lower level languages
  - Compiled vs interpreted
- Compiler optimizations
  - Function inlining
  - Pre-compute results
  - ...



(µ-)architecture awareness

- Optimize cache usage
  - Loop optimizations
- Optimize pipeline usage
  - Avoid hazards/stalls



Loop Fission

Reminder: caching!

- Loop Fission/distribution
  - break a loop into multiple loops over the same index range
  - each new loop takes only part of the original loop's body
  - improve locality of reference for both data and code

example

```
int i, a[100], b[100];
 for (i = 0; i < 100; i++) {
   a[i] = 1;
   b[i] = 2;
 }
```

loop_fission

```
int i, a[100], b[100];
 for (i = 0; i < 100; i++) {
   a[i] = 1;
 }
 for (i = 0; i < 100; i++) {
   b[i] = 2;
 }
```

Loop Unrolling

- Decreases # instructions
  - Less jumps/conditional branches
    - Better for pipeline

Might perform worse due to micro-architecture!

example

```
for (int i=0; i<N; i++)
{
  sum += data[i];
}
```

loop_unrolling

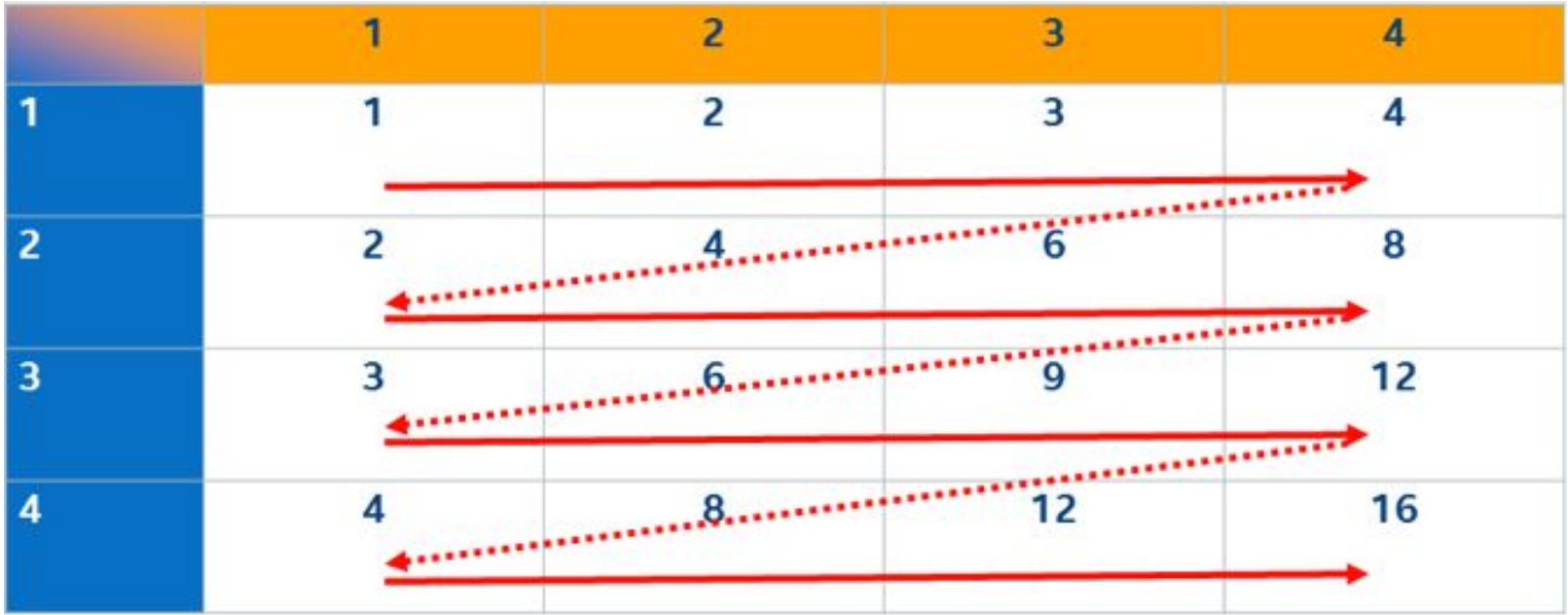
```
/* assume N is a multiple of 4 */
for (int i=0; i<N; i+=4)
{
  sum1 += data[i+0];
  sum2 += data[i+1];
  sum3 += data[i+2];
  sum4 += data[i+3];
}
sum = sum1 + sum2 + sum3 + sum4;
```

Loop Optimization (Loop Tiling)

	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

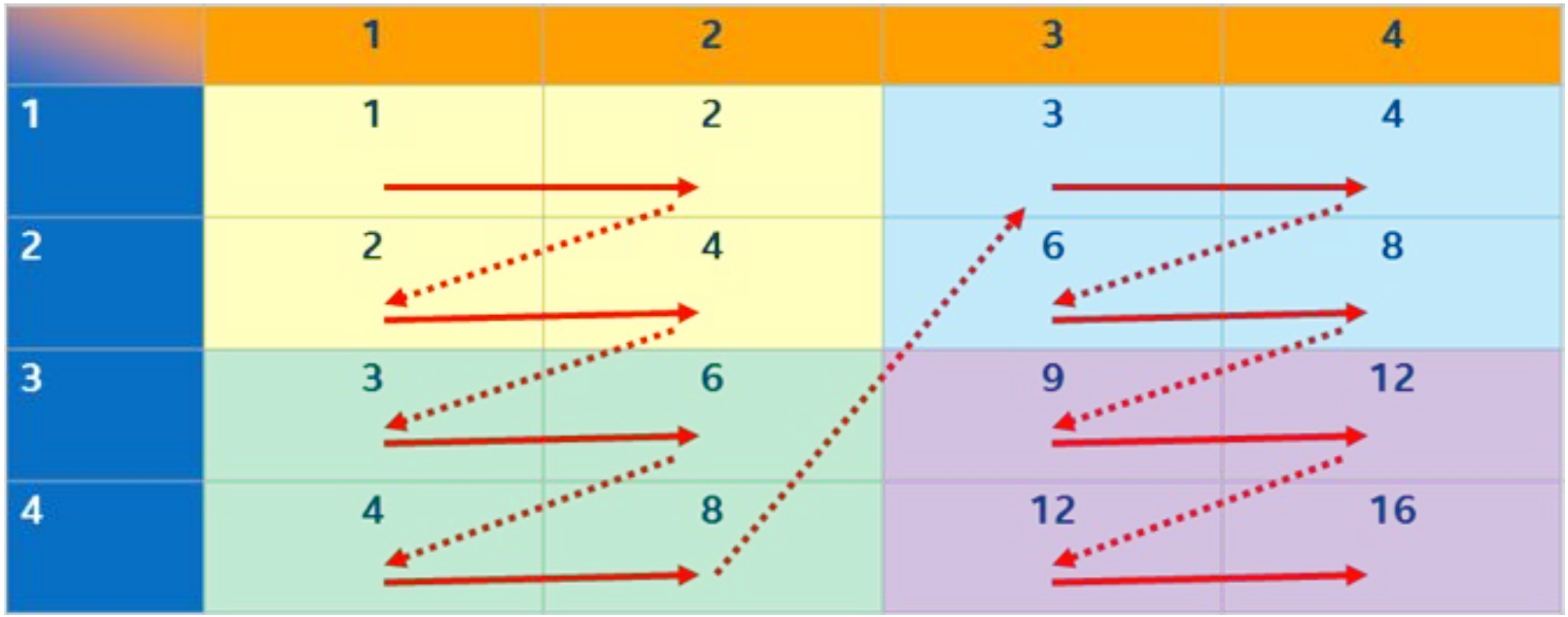
https://software.intel.com/en-us/articles/efficient-use-of-tiling

Loop Optimization (Loop Tiling)



https://software.intel.com/en-us/articles/efficient-use-of-tiling

Loop Optimization (Loop Tiling)



https://software.intel.com/en-us/articles/efficient-use-of-tiling

Background: Clock and CPU-Time

- CPU is driven by a clock
- Clock cycle time
  - the amount of time for one clock period to elapse
- Instructions require some number of clock cycles

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

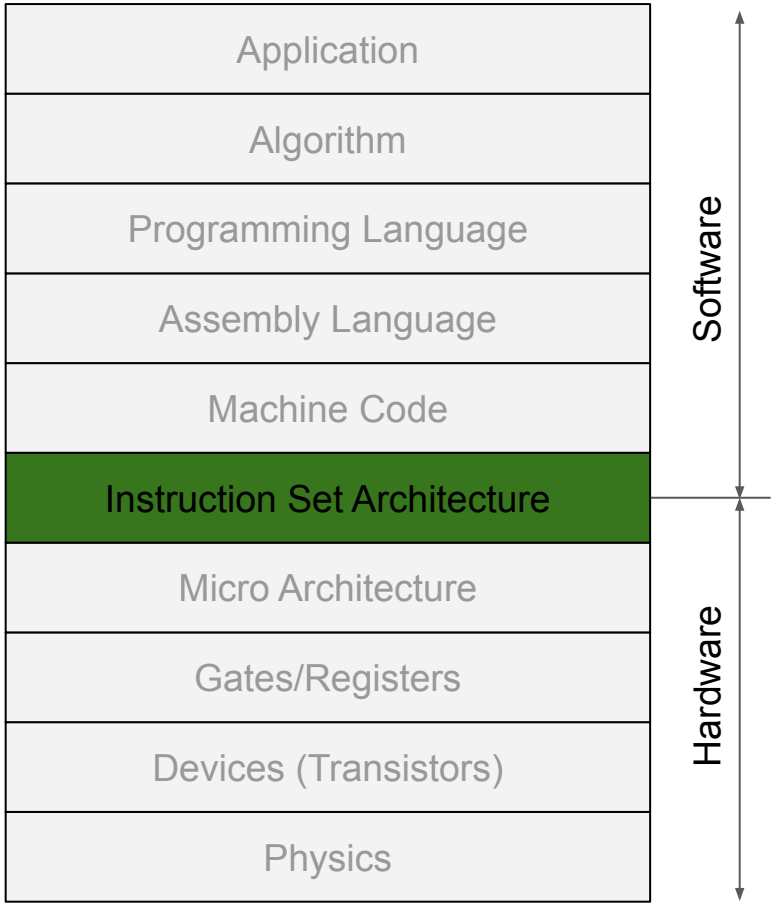
Which would you prefer?

a) Program with only a few instructions
   i) But the instructions are slow

b) Program with many instructions
   i) But the instructions are fast

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/

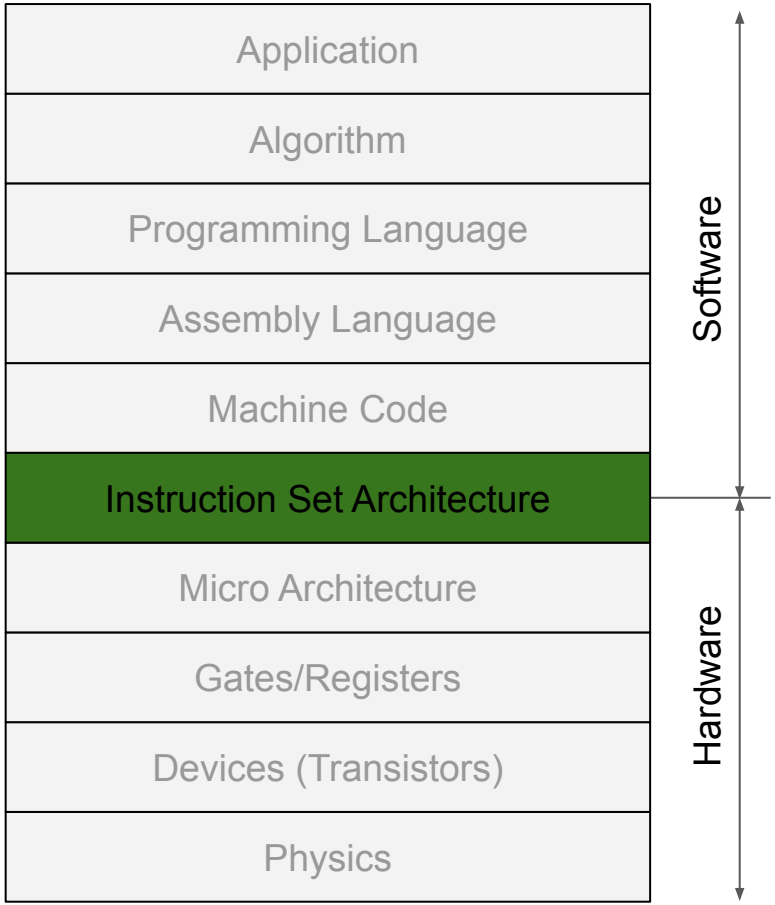


RISC vs CISC

- RISC
  - Reduced Instruction Set Computer
  - Simple, fast instructions
- CISC
  - Complex Instruction Set Computer
  - Complex, slow instructions

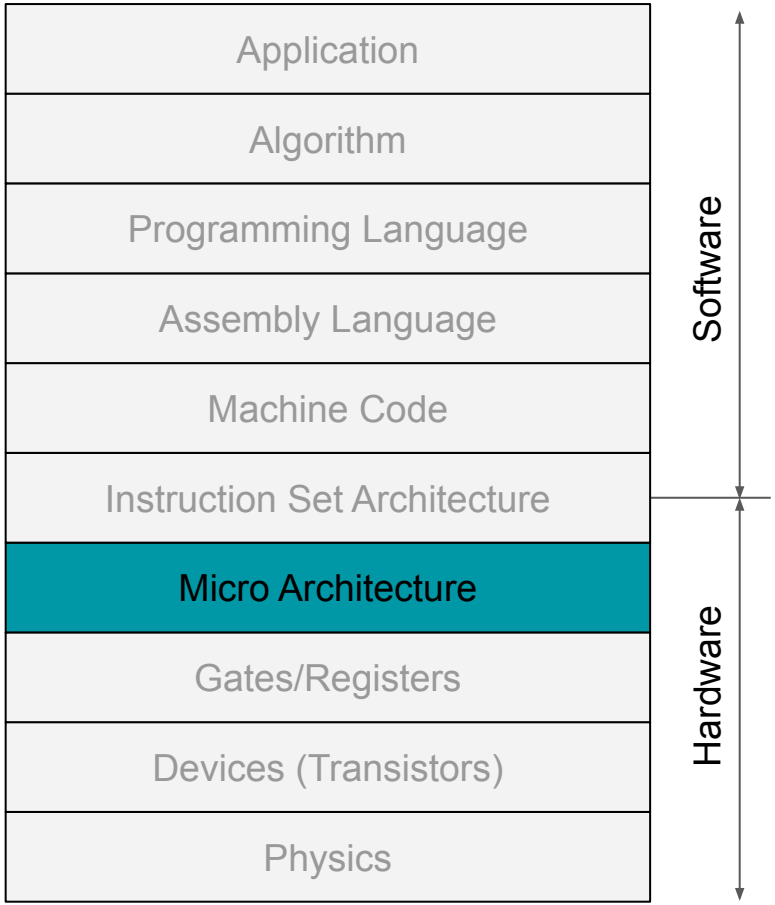
$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/



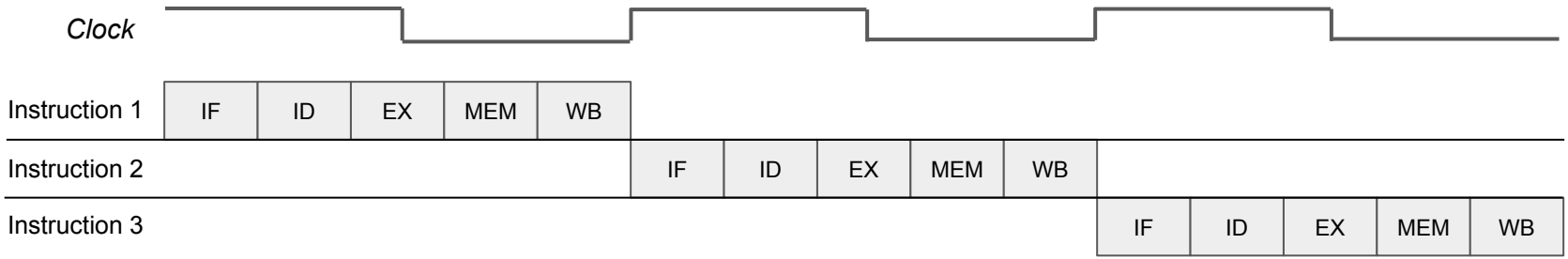
Micro architecture

- *Pipelining*
- Out-of-order execution
- Speculative execution
  - Branch prediction

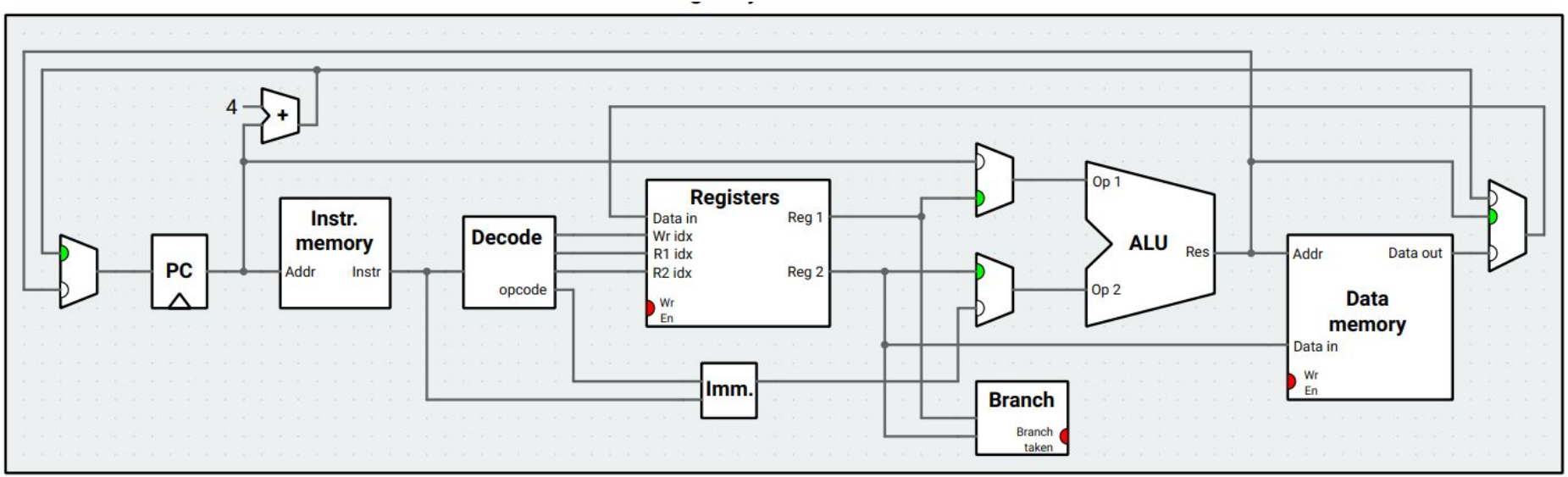


Execution phases

1. Instruction Fetch
2. Instruction Decode
3. Execute
4. Memory access
5. Writeback

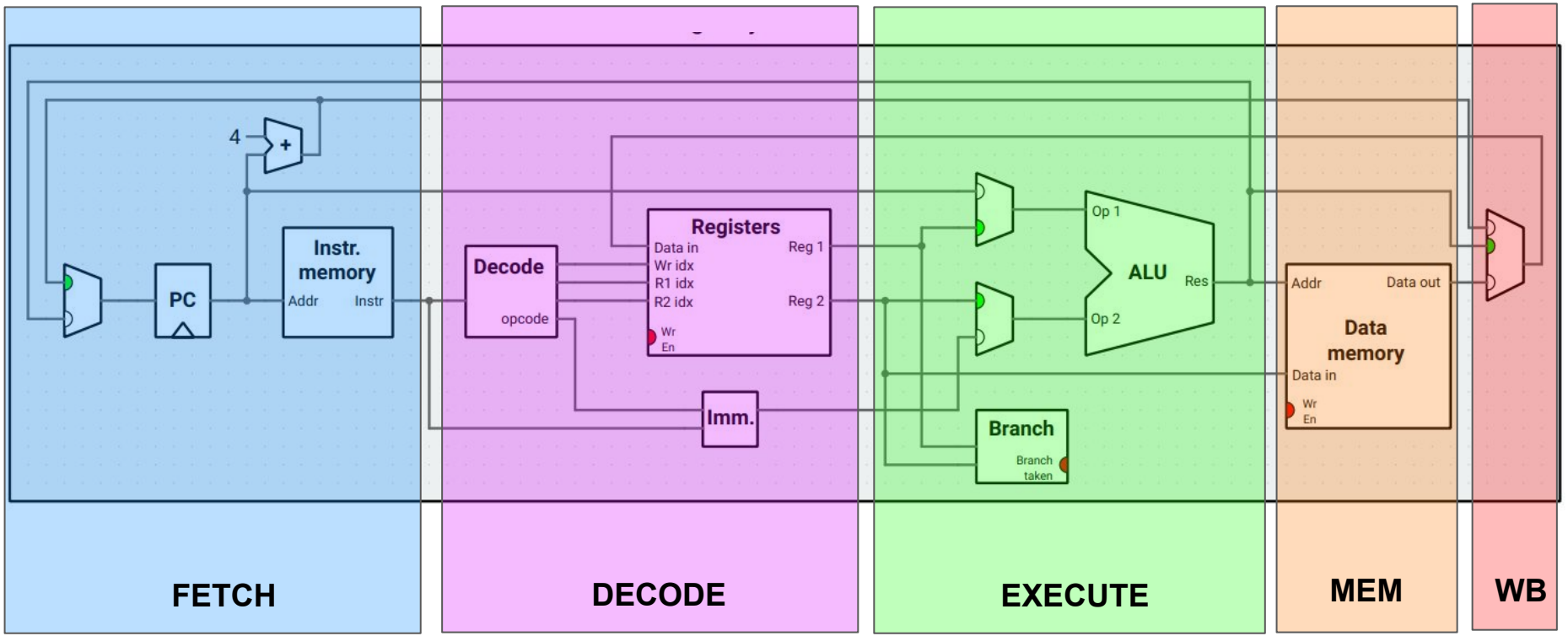


Single-cycle RISC-V processor



Source: https://github.com/mortbopet/Ripes

Single-cycle RISC-V processor



Single-cycle RISC-V processor

Single-cycle processor

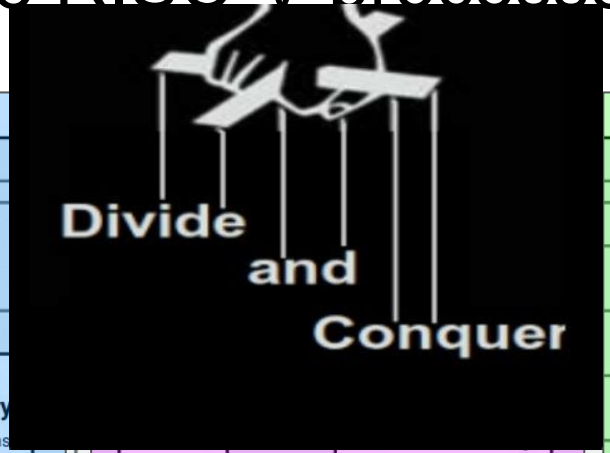
CPU speed limit: instruction has to travel through all 5 stages in a *single cycle*

→ if CPU runs too fast, instruction may never reach write-back.. :(

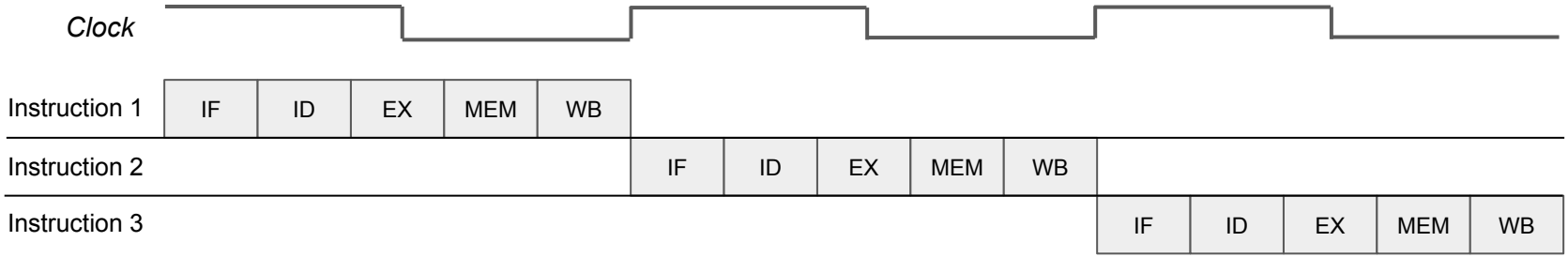
Single-cycle RISC-V processo

Multi-cycle processor

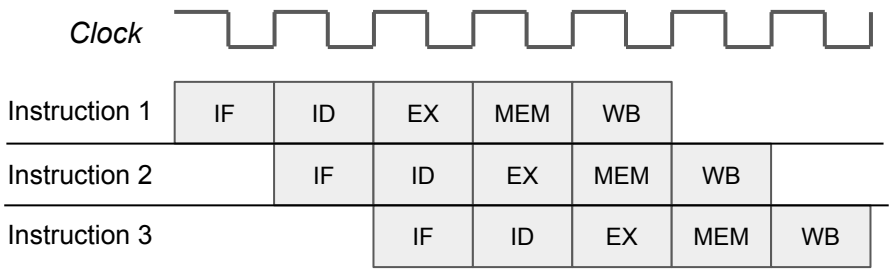
Divide-and-conquer: 1 cycle *per stage* → CPU can run faster (and a single instruction now takes *max* 5 cycles)



Single-cycle design



Pipelined design



Latency

Single cycle: 1 clock cycle

Pipeline: 5 clock cycles

Throughput

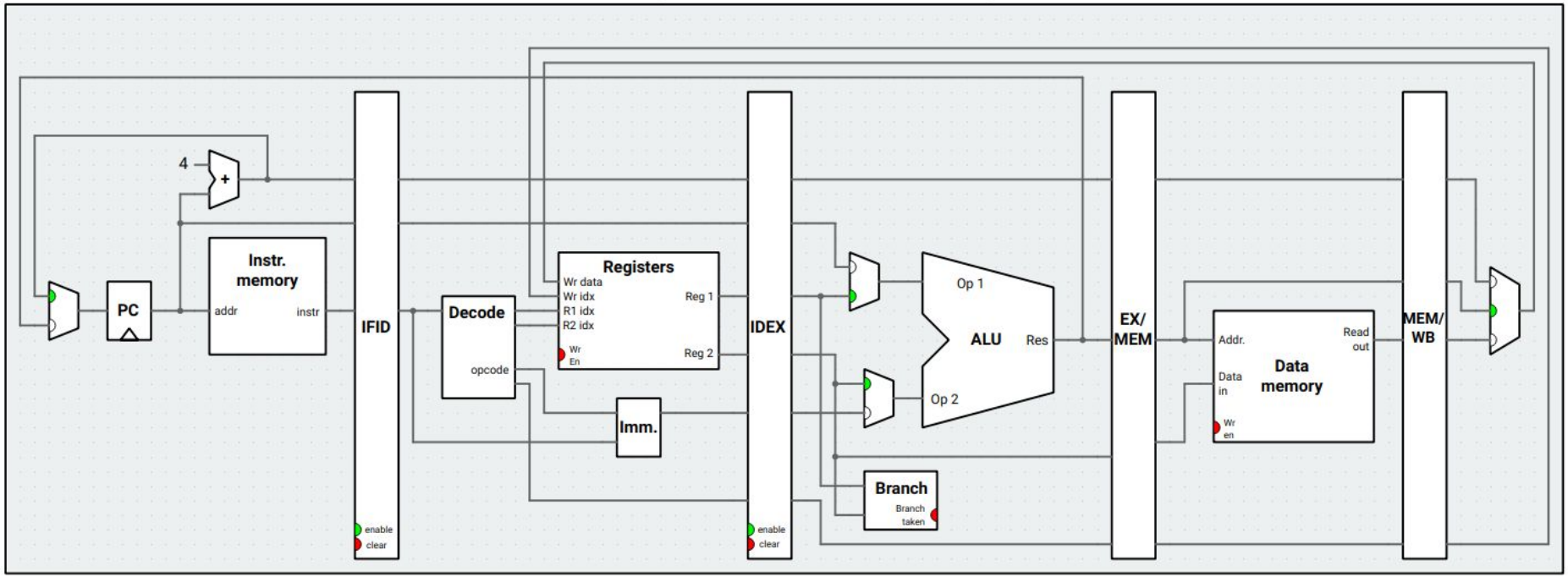
Single cycle: 1 instr/cycle

Pipeline: 1 instr/cycle

Clock speed

pipeline > single-cycle

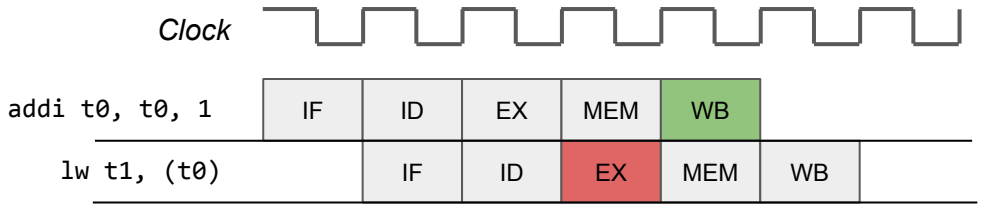
5-stage RISC-V processor



Source: https://github.com/mortbopet/Ripes

Hazards

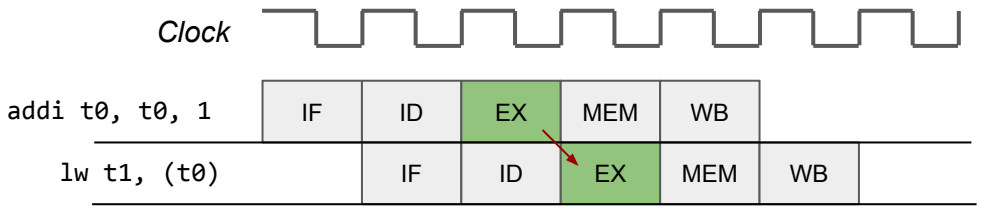
- Pipeline
  - Multiple instructions in parallel!
- But…
  - Dependencies between instructions



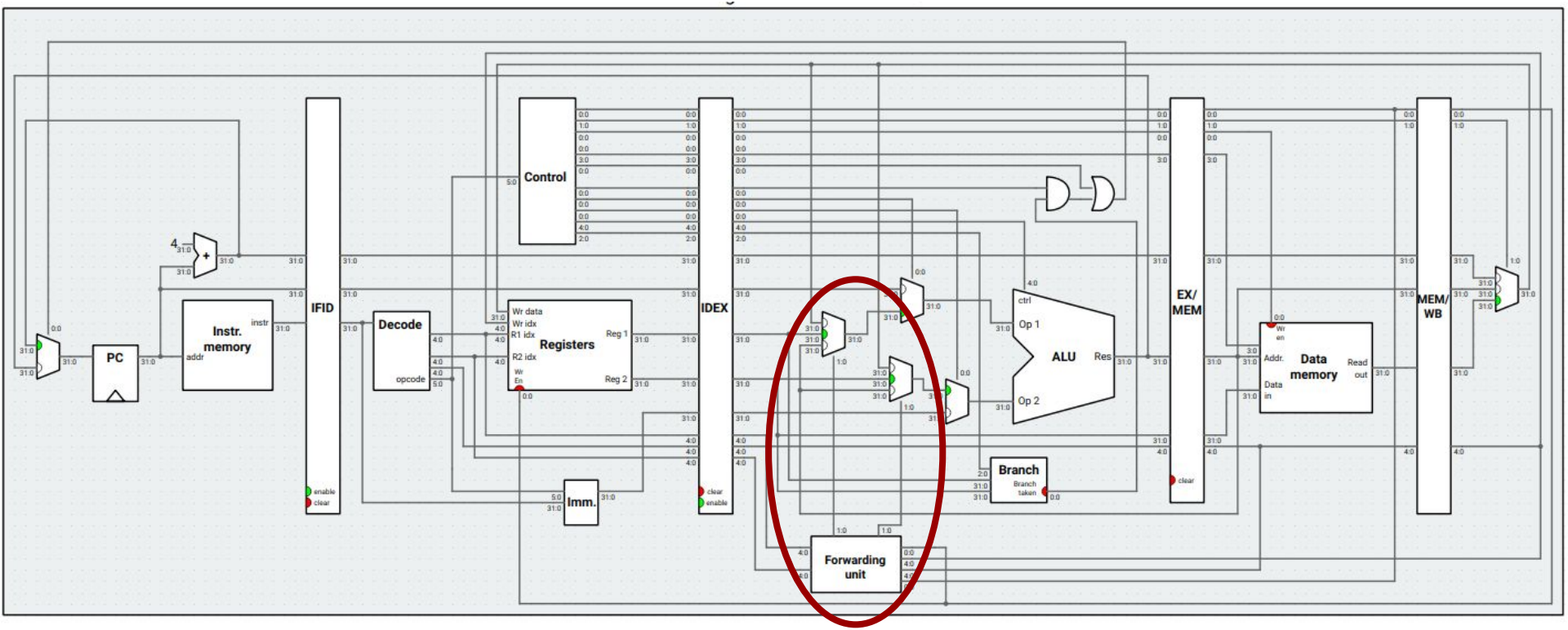
Read After Write (**RAW**)
New value of t0 is committed in **WB**
but already needed in **EX**

Solution: Forwarding

Read After Write (RAW)



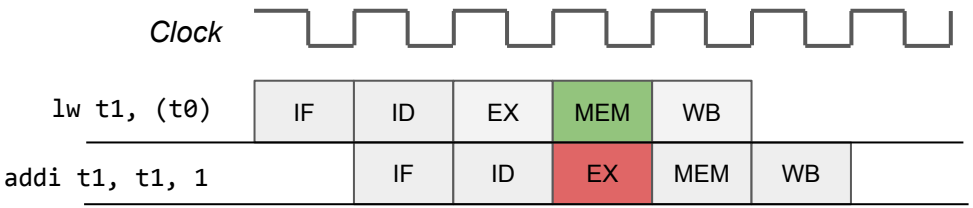
5-stage RISC-V with Forwarding



Source: https://github.com/mortbopet/Ripes

Problem: not always possible

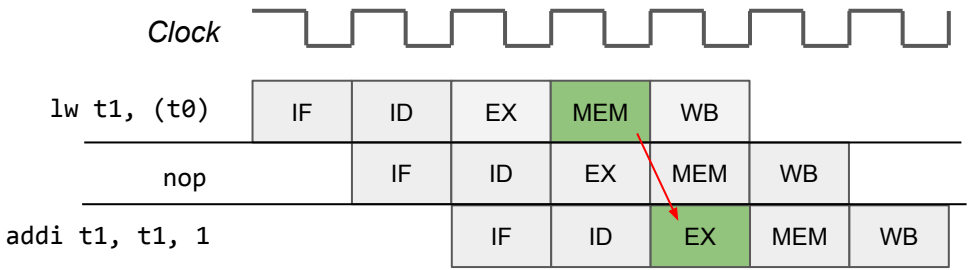
Read After Write (RAW)



Memory phase needs to be completed for loads!

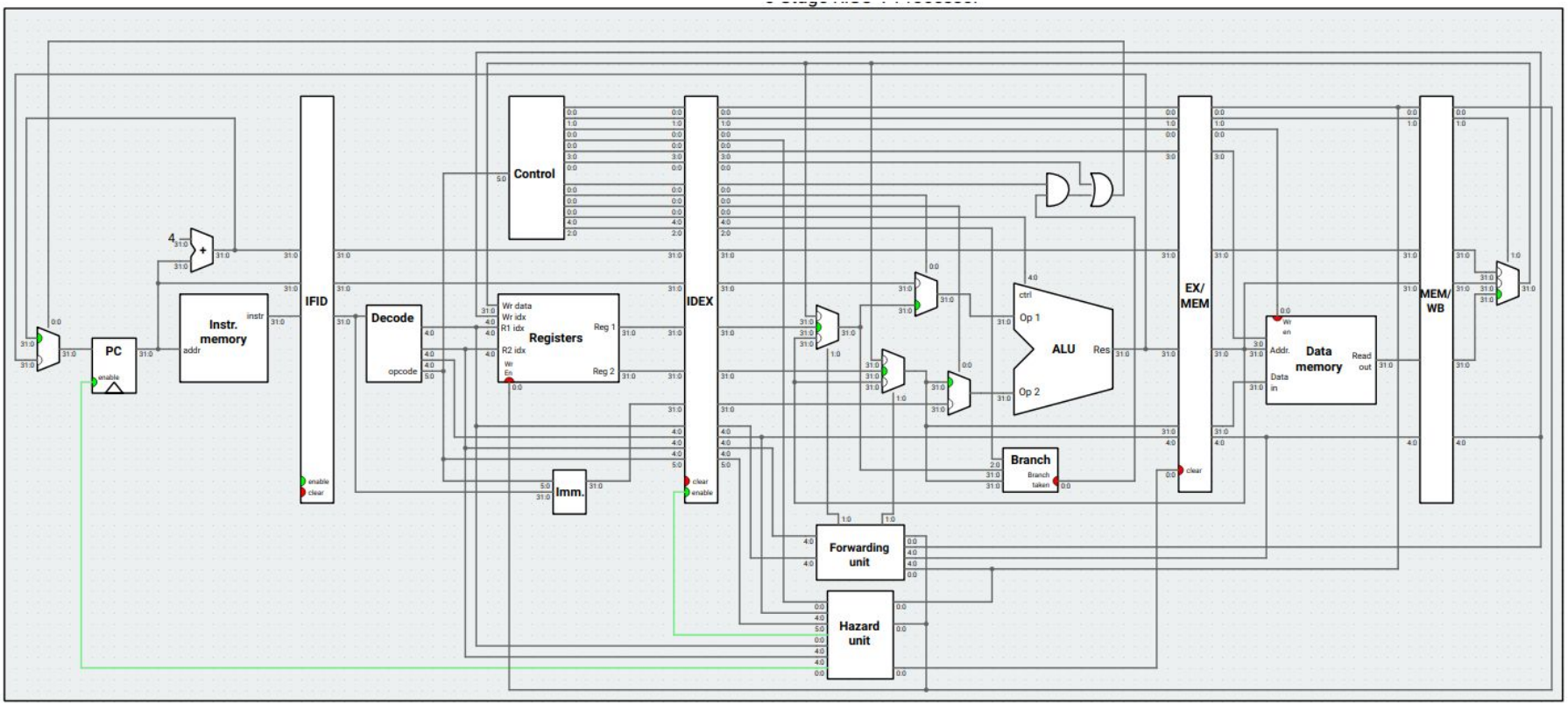
Solution: Forwarding + Stall

Read After Write (RAW)



Insert stall: hardware or software (compiler)?

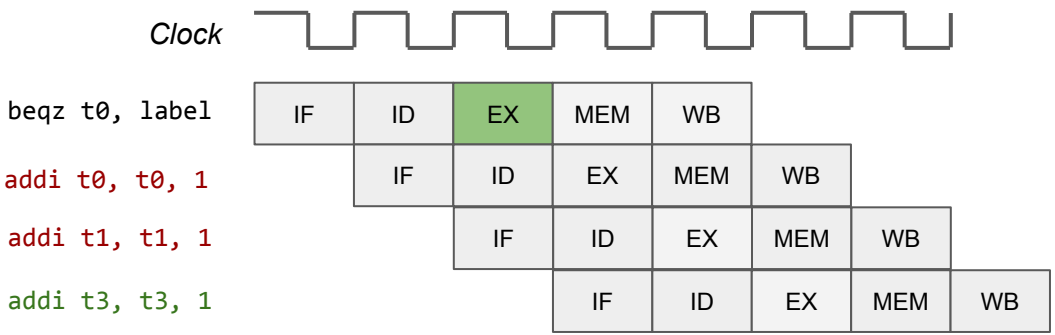
5-state RV w Forwarding and detection (auto stall)



Problem: branches

Instructions shouldn't be executed if t0 = 0!

Control hazard

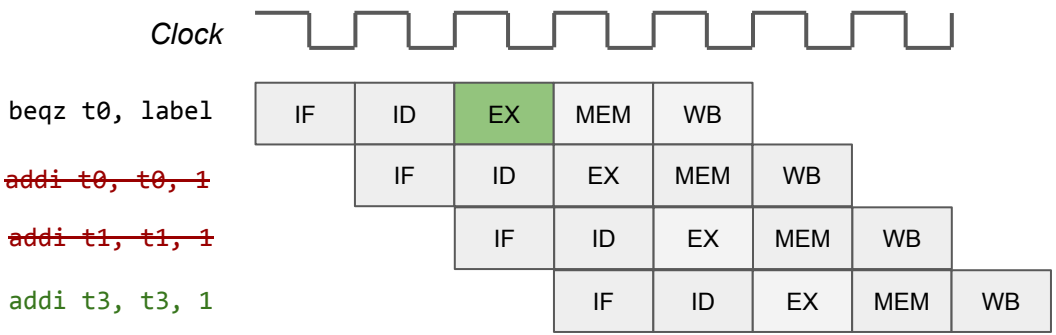



```
     mv   t0, zero
     beqz t0, label
     addi t0, t0, 1
     addi t1, t1, 1
     addi t2, t2, 1
label:
     addi t3, t3, 1
     addi t4, t4, 1
```

Solution: branches

Don’t commit results

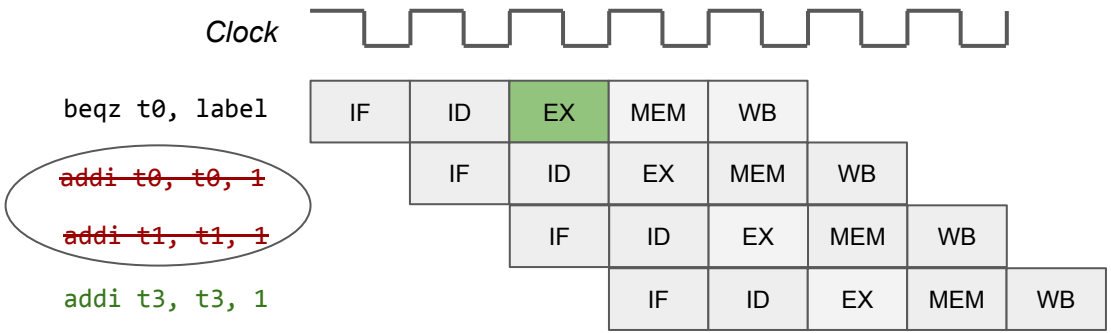
Control hazard




```
    mv   t0, zero
    beqz t0, label
    addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, 1
label:
    addi t3, t3, 1
    addi t4, t4, 1
```

What if… results discarded but already cached?

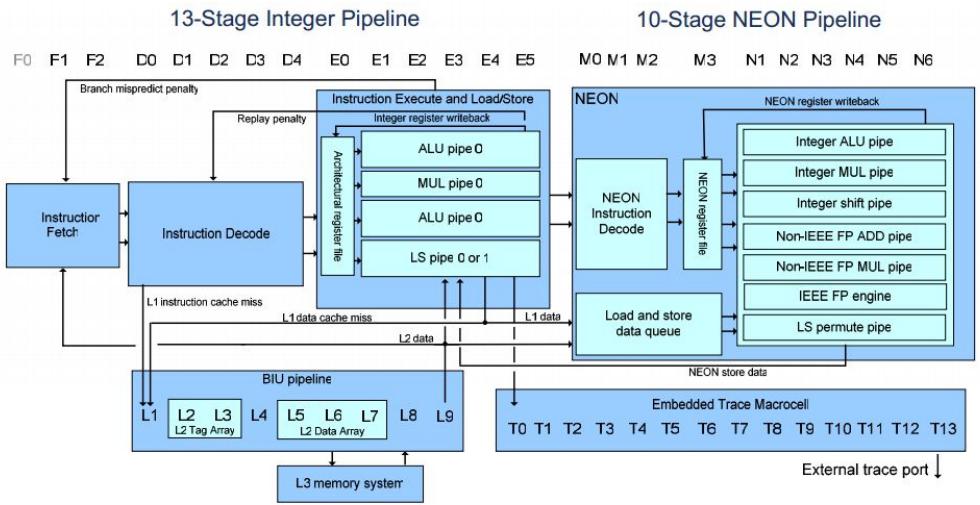
Control hazard

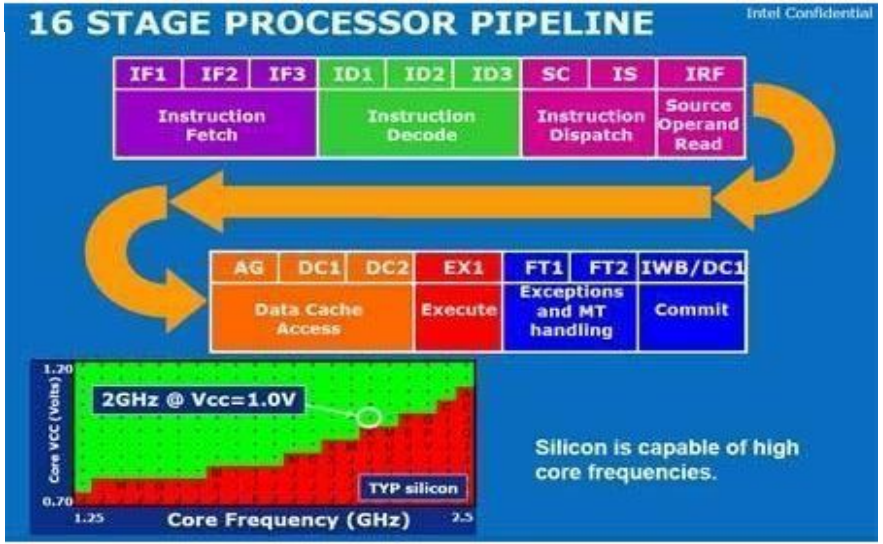



```
    mv   t0, zero
    beqz t0, label
    addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, 1
label:
    addi t3, t3, 1
    addi t4, t4, 1
```

https://meltdownattack.com/

Pipelining in Modern Processors





- ARM Cortex A8
  - Apple's iPad, iPhone 4 smartphone,the iPod Touch (4th generation), and the Apple TV (2nd generation).
- Intel Atom CPU
  - Smartphone, Netbook/Nettop, Tablet, Embedded, Microserver/Server and Consumer electronics