Advanced Computer Architecture

—

Part I: General Purpose
SimpleScalar Practical Lab (I)

EPFL – I&C – LAP

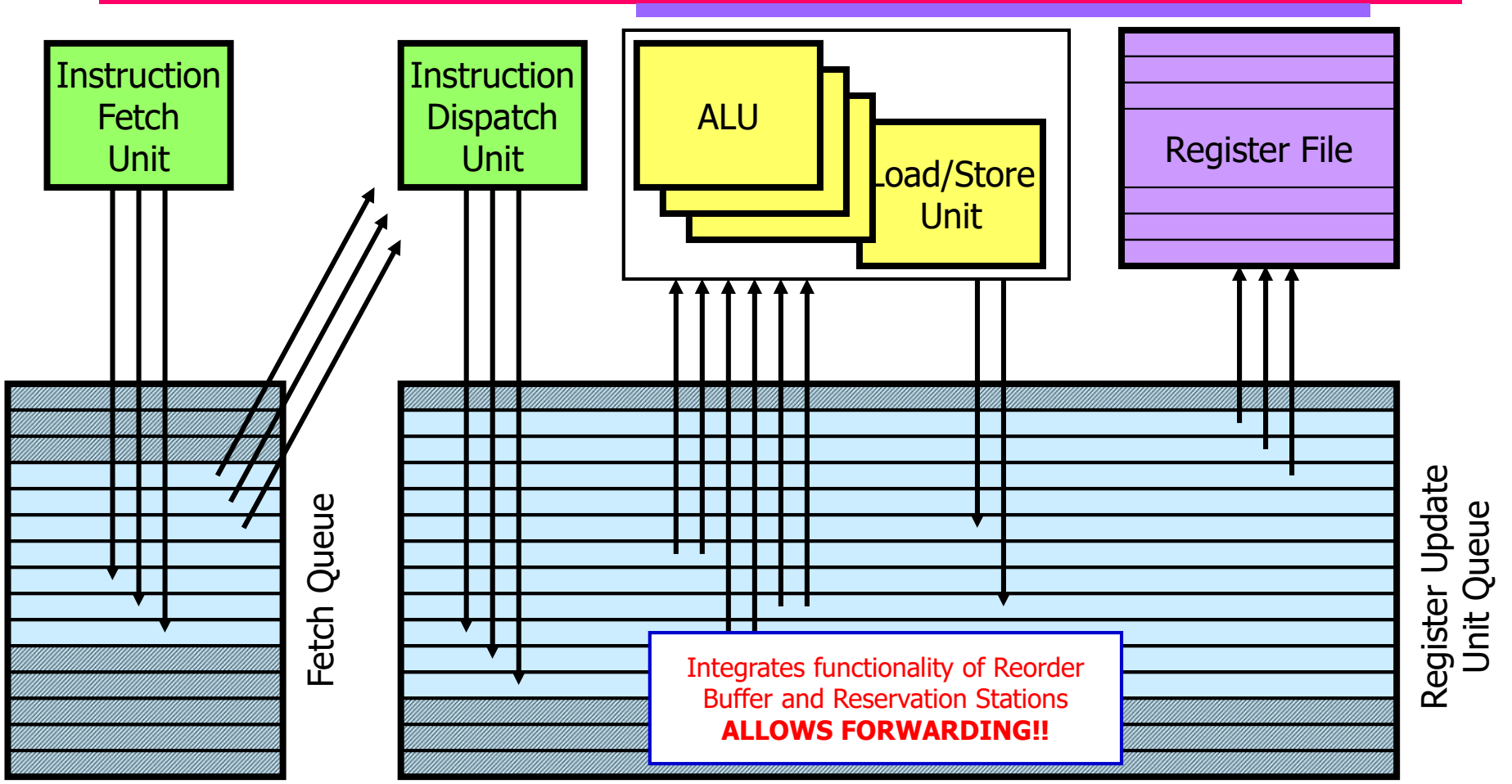
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

SimpleScalar

- An open source computer architecture simulator
- A simulator can
  - Simulate different CPU architectures
  - Quantify CPU architecture performance

- Our goal is to understand the effects of architectural changes on CPU performance

SimpleScalar Architecture

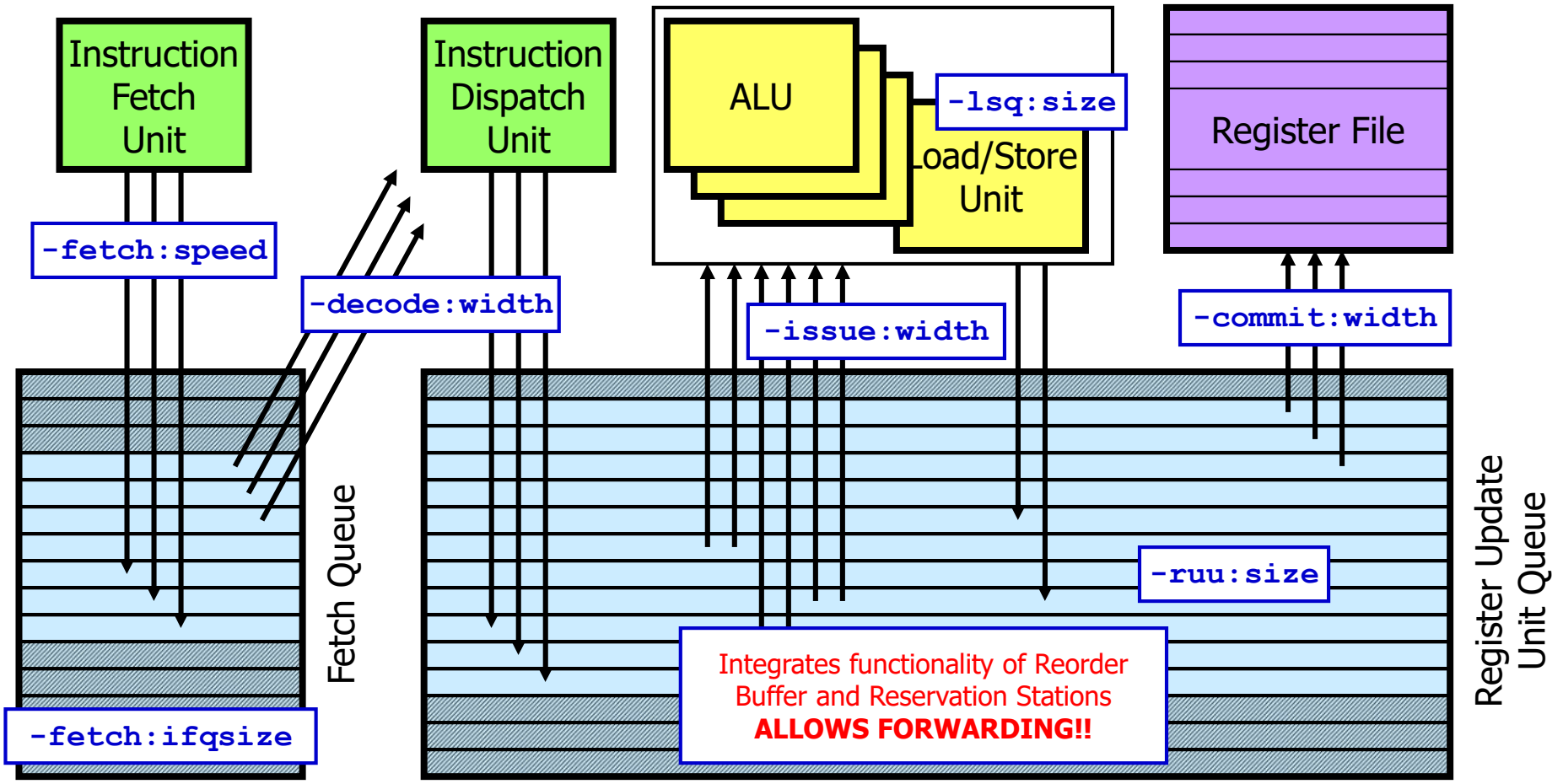


SimpleScalar Configuration

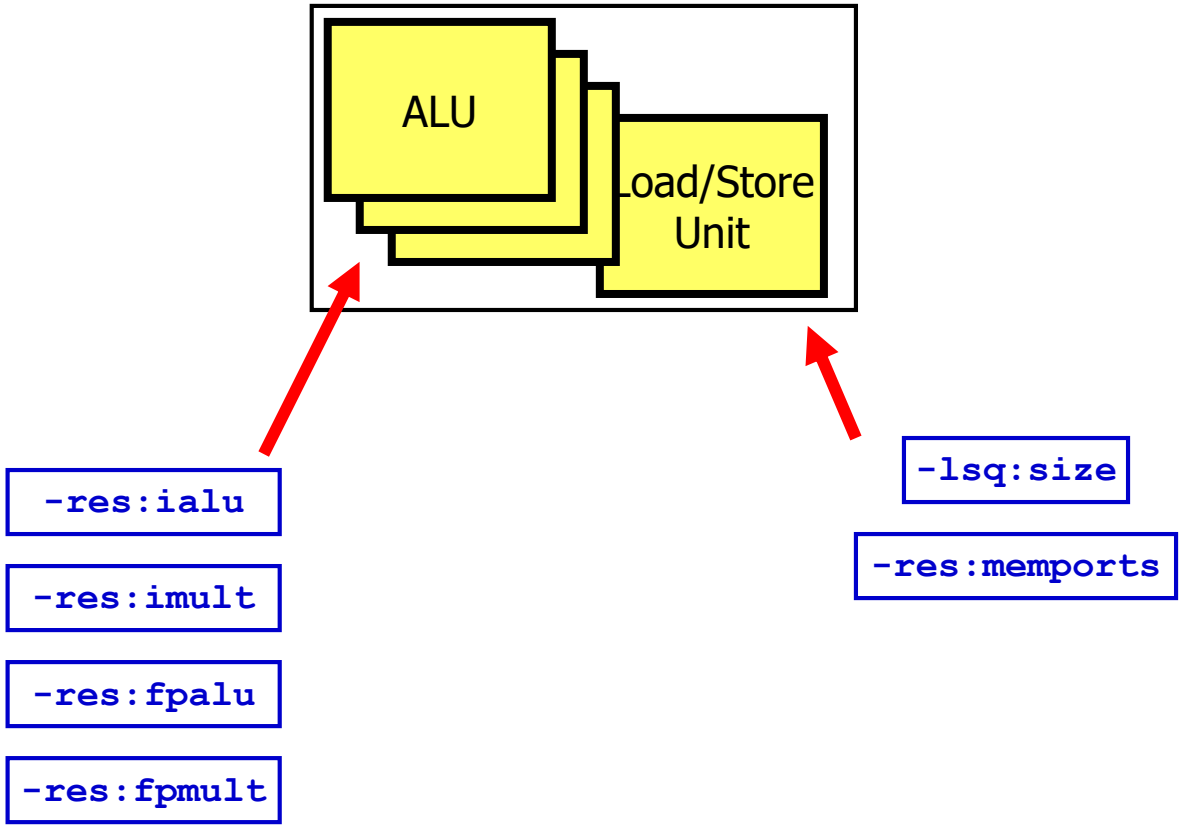
- We can configure SimpleScalar architecture
  - Number of units in the pipeline
  - Size of a unit
  - Behavior of a unit

- We control the architecture in configuration file (.cfg)

SimpleScalar Architecture



SimpleScalar Execution Units

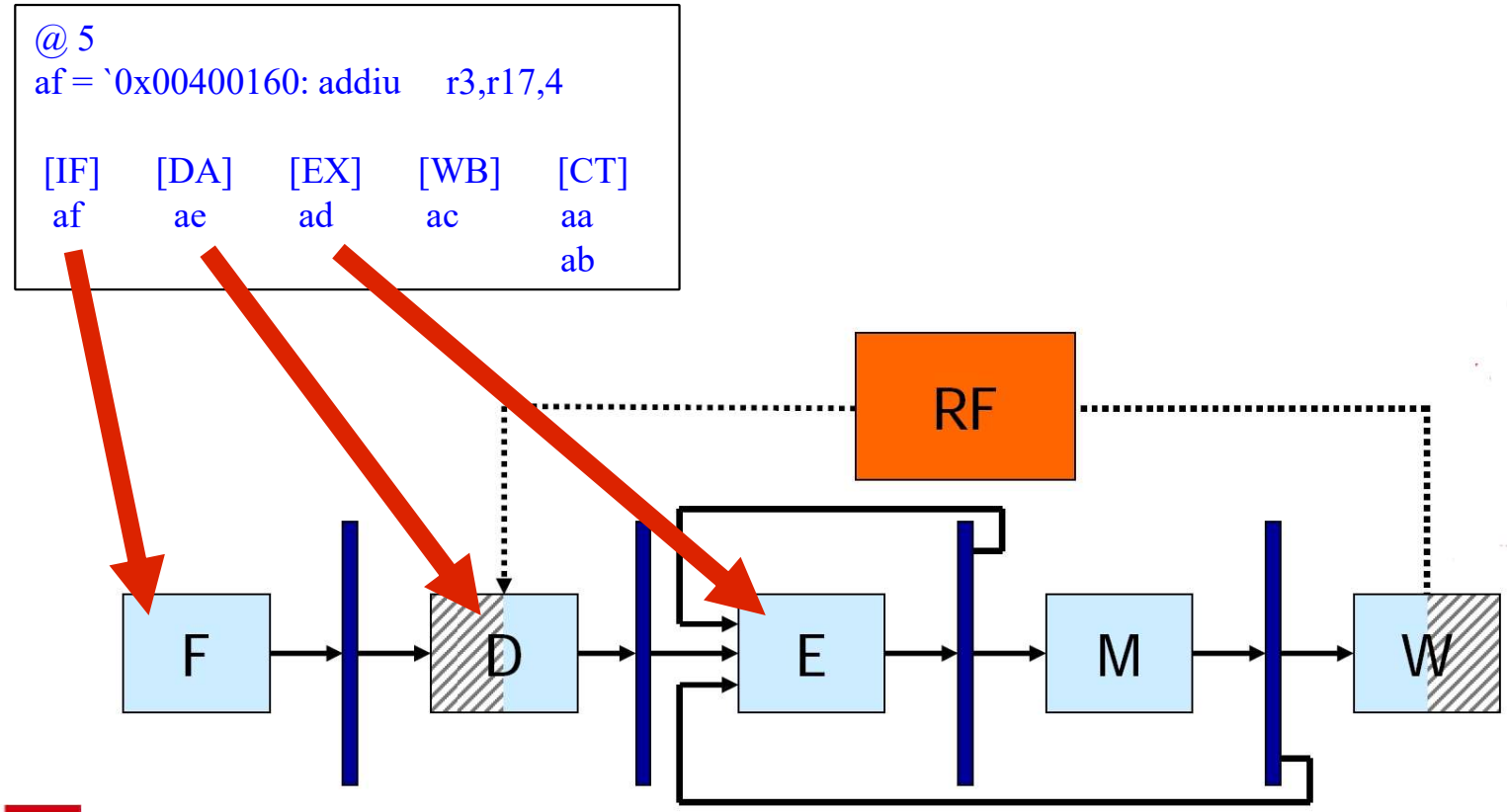


Our Configurations

	resources FU	-fetch: speed	-fetch: ifqsize	-lsq: size	-ruu: size	-issue: width	-commit: width	inorder
a	1	1	1	8	8	1	1	Y
b	1	1	1	8	8	1	1	N
c	4	1	1	8	8	1	1	Y
d	4	1	1	8	8	1	1	N
e	1	4	4	8	8	4	4	N
f	4	4	4	8	16	4	4	N

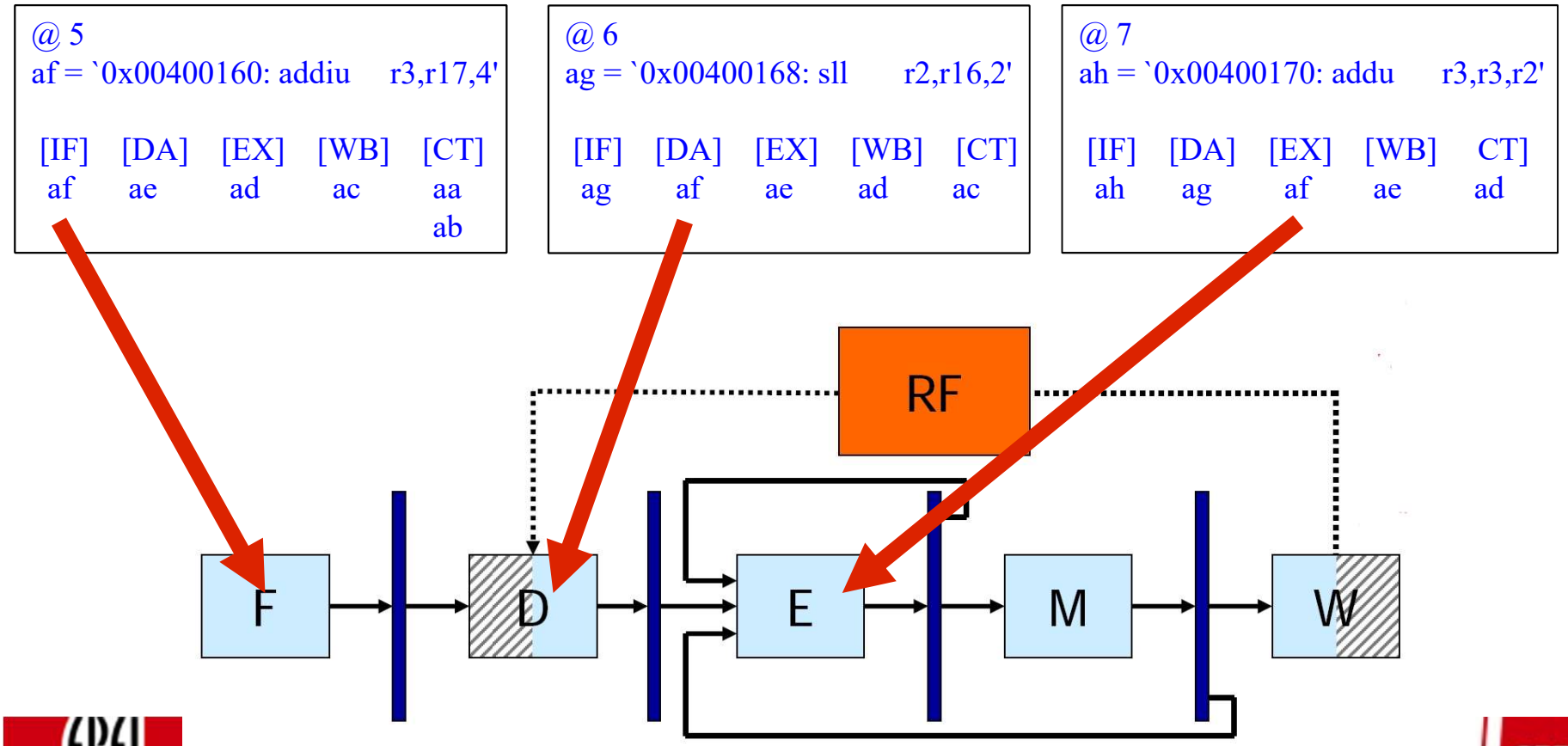
SimpleScalar Simulation

- Each line indicates state of processor at end of a cycle



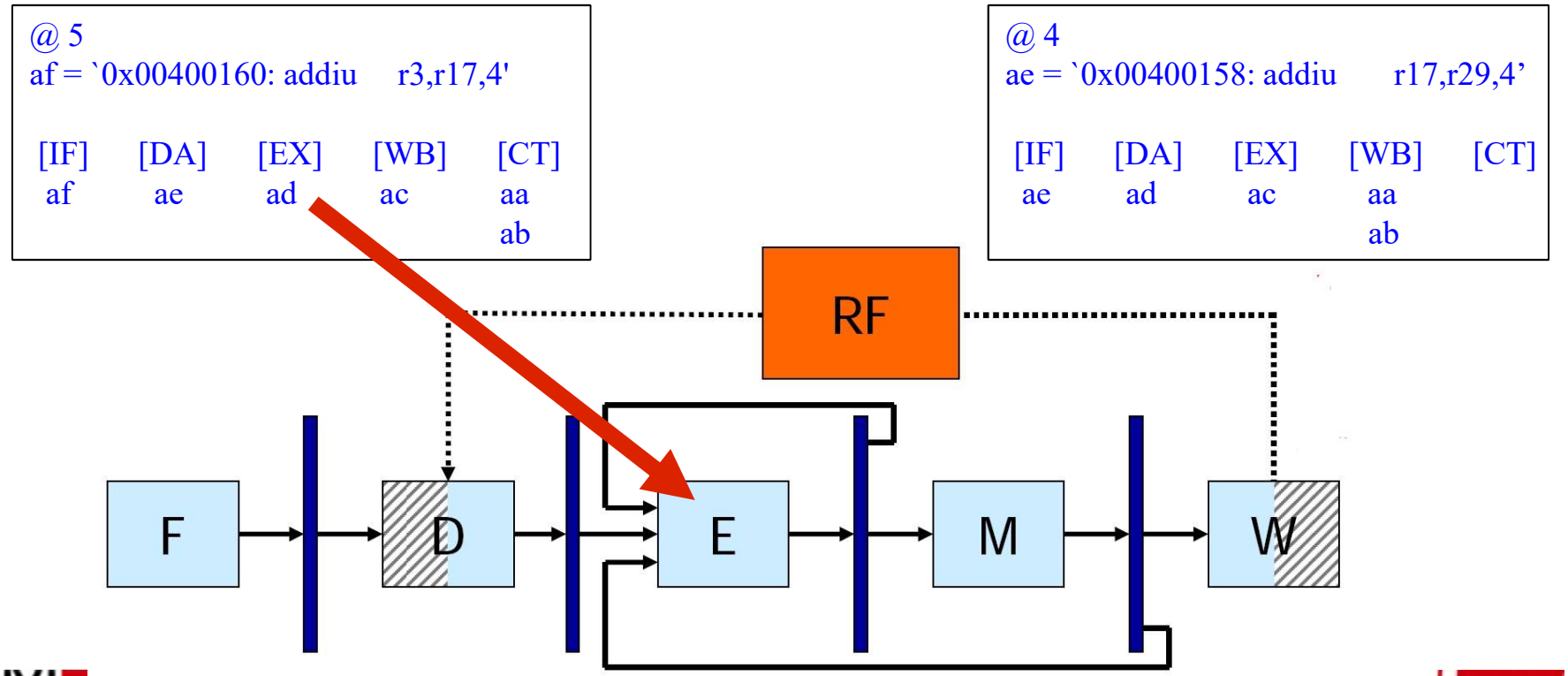
Following an Instruction

- One can easily follow an instruction step by step



Forwarding Example

- Thanks to forwarding, at cycle 5, ad can use the result of ac computed at cycle 4



Simulation Execution

- We can execute any program with SimpleScalar
- Trace output (.trc) contains details of execution
  - "-ptrace" in configuration file
- Other useful metrics are produced as well
  - We can save them in a text file (.out)

Visualization

- We can visualize instruction execution using the perl script "pipeview.pl"

- The script loads the instruction execution details from the simulation trace output file and display them on terminal

- Very useful to observe effects of architectural changes

Benchmarking

- We evaluate different architecture configurations
- Run the same "standard" benchmark and compare relative performance
- For this lab we use "test-math" (in bin.little)
  - Does not need input
  - Generates a list of arithmetic operations as output
  - Calls both integer and floating-point instructions.

Under the Hood

- SimpleScalar directory

  `/usr/simplescalar/simplesim-3.0`

- Configuration files and benchmark files copied to

  `~/simplescalar`

- Simulator binary (alias soo)

  `/usr/simplescalar/simplesim-3.0/sim-outorder`

- Visualization perl script (alias pv)

  `/usr/simplescalar/simplesim-3.0/pipeview.pl`

Commands

- Run a simulation

```
soo -config config/<filename>.cfg
tests/bin.little/test-math 2&> <filename>.out
```

- Observe trace with pipeview

```
pv <filename>.trc | less
```

- Execute all following commands in `~/simplescalar`

In-Order / Out-of-Order

- Compare configurations *a* and *b*
  - Check ~/simplescalar/config folder
- Launch a simulation with configurations *a* and *b*

```
soo -config config/a.cfg tests/bin.little/test-math 2&> a.out
soo -config config/b.cfg tests/bin.little/test-math 2&> b.out
```

- Compare their performance, taken as the instruction per cycle (IPC)
  - Search for "sim_IPC" in a.out and b.out
- Justify the variation of performance

Back to the (BSc) basics

- If the concepts seen so far are not clear, do not hesitate to look at the slides of the course below and/or ask the assistant(s)/instructor(s):

Architecture des Systems-on-Chip

Pipelining

http://lap2.epfl.ch/courses/advcomparch/slides_archsoc/Pipelining.pdf

Dependencies

- Observe a's execution trace using pipeview

```
pv a.trc | less
```

- Consider the 2 following instructions:

```
ac     =   lui r28, 0x1001
ad     =   addiu r28, r28, -24528
```

- Does the data dependency between the 2 instructions introduce a stall? Relate to the architecture

Out-of-Order

- Consider cycle 15 in trace of configuration *b*, instruction ap executes out-of-order, are there any dependencies?

- Commit stage [CT] reorders the execution of instructions
  - Why is it important?
  - Does performance rely on it?

The special case of `lw`

- The **`lw`** and **`sw`** instructions are decomposed in 2 parts
- Give a possible explanation of what is happening
  - Hint: ALU and LW/SW units operate in the same stage
- Explain why **`lw`** takes 6 cycles and **`sw`** take 5

Resources

- Compare *a* and *c* configurations
- Launch a simulation with configuration *c*

```
soo -config config/c.cfg tests/bin.little/test-math 2&> c.out
```

- Compare the performance of *a* and *c*
  - Justify the performance variation

Number of Execution Units (I)

- Launch a simulation with configuration *d*

```
soo -config config/d.cfg tests/bin.little/test-math 2&> d.out
```

- Observe the execution trace. Notice during the 3rd clock cycle that ac and ad are still in the [DA] stage
  - What architecture bottleneck is blocking instructions' execution?
- Change the configuration to increase performance
  - Verify by running a simulation with the updated configuration

Number of Execution Units (II)

- Launch a simulation with configuration *e*

```
soo -config config/e.cfg tests/bin.little/test-math 2&> e.out
```

- Why does configuration *e* outperform configuration *b* ?
- [EX] stage executes ab and ae in parallel in 3rd clock cycle but that does not happen in other cycles
  - Where is the bottleneck in the architecture?
  - Change the configuration to increase performance
  - Verify with a simulation

Number of Execution Units (III)

- In the first 2 clock cycles, the processor fetches 4 instructions per cycle

- Explain why the processor does not fetch any instruction for the 3rd cycle

- Change the configuration to allow this fetch
  - Verify with a simulation

Dependencies

- Launch a simulation with configuration *f*

```
soo -config config/f.cfg tests/bin.little/test-math 2&> f.out
```

- Observe the different data dependencies:
  - Read after Write (RAW):
    - `ab = lui r28, 0x1001`
    - `ac = addiu r28, r28, -24528`
  - Write after Read (WAR):
    - `ay = sw r16, 16(r29)`
    - `az = addu r16, r0, r5`
- Explain the execution of these two cases

Extra Exercise
Explore the Design Space

- Start from configuration *f*
- Independently, increase the number of each resource

  (-res:ialu, -res:imult, -res:fpalu, -res:fpmult)

- Draw graphs showing the performance evolution as a function of the resources
- How does each resource affect performance?