Modern Computer Architectures

ET4074 - 2018/2019

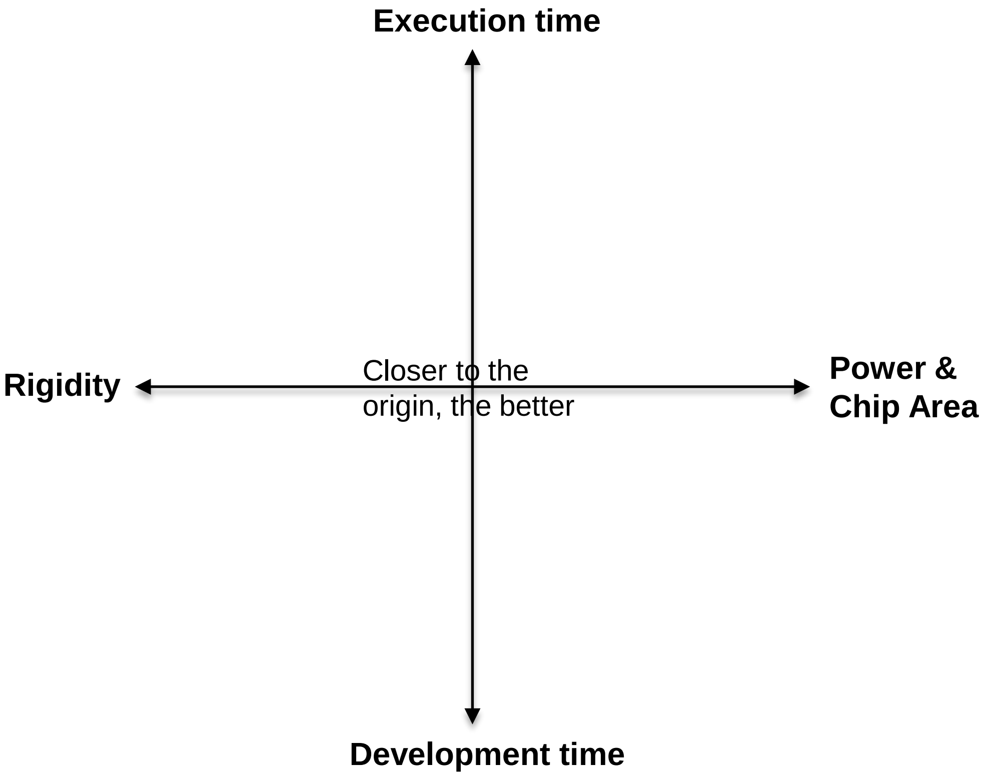
TU Delft

Lab sessions & deadlines

Nov. 27	Lab 1 Kick-off
Dec. 4	Lab question session
Dec. 11	***Assignment 1 submission deadline*** *(40% grade)* Lab 2 Kick-off
Dec. 12	Lab question session
Dec. 19	Lab question session
Jan. 9	Lab question session
Jan. 21	***Assignment 2 submission deadline*** *(60% grade)*

Lab Grade = 40% Assignment 1 + 60% Assignment 2

Hardware technology trends



Execution time

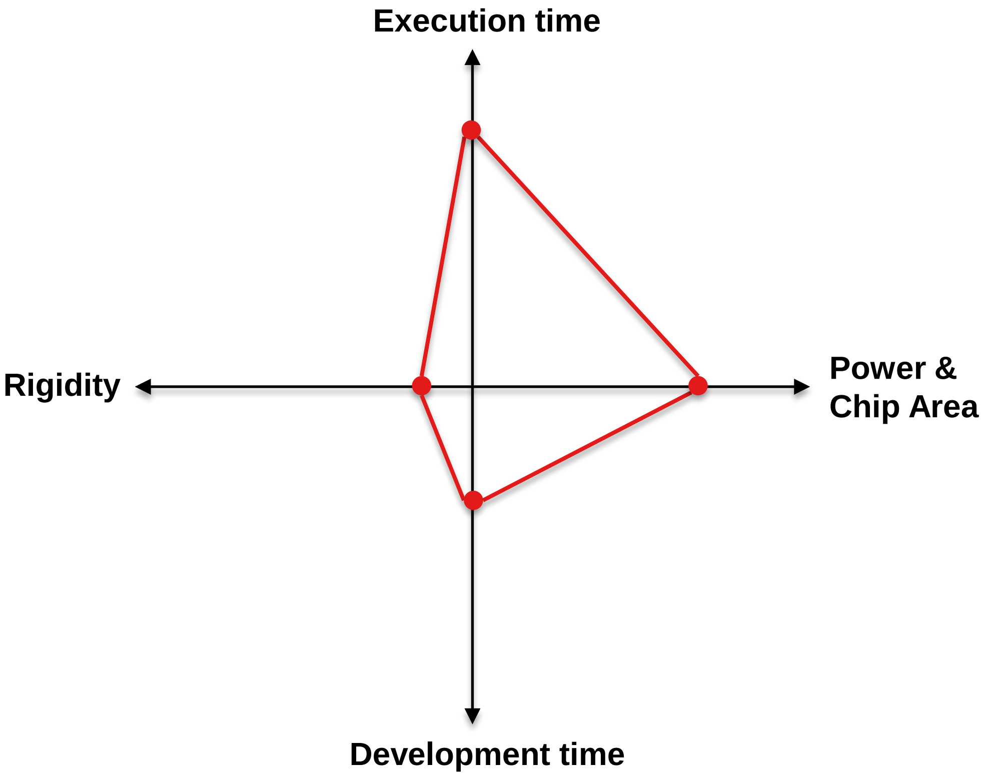
Rigidity

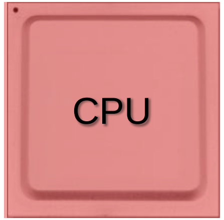
Closer to the origin, the better

Power & Chip Area

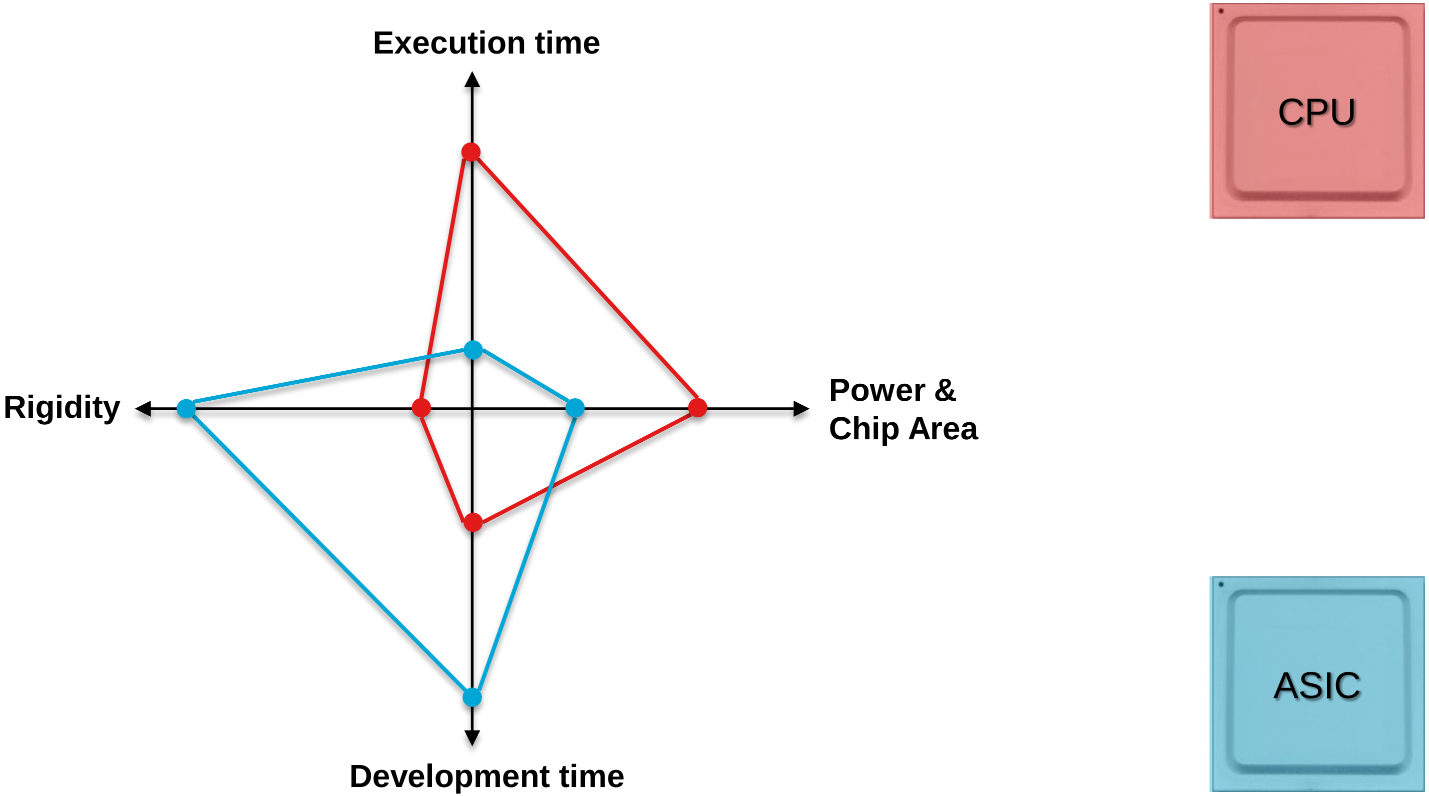
Development time

Hardware technology trends

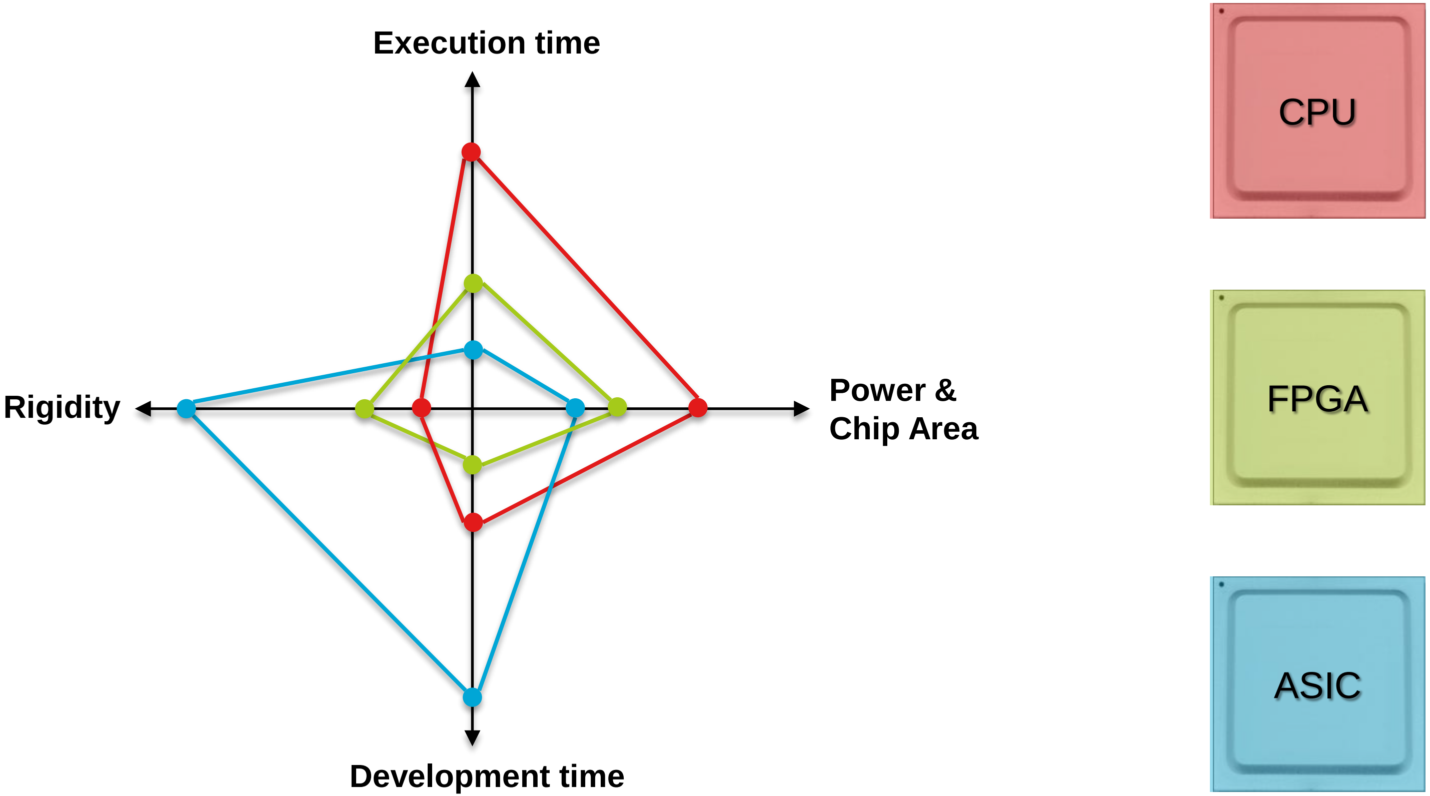




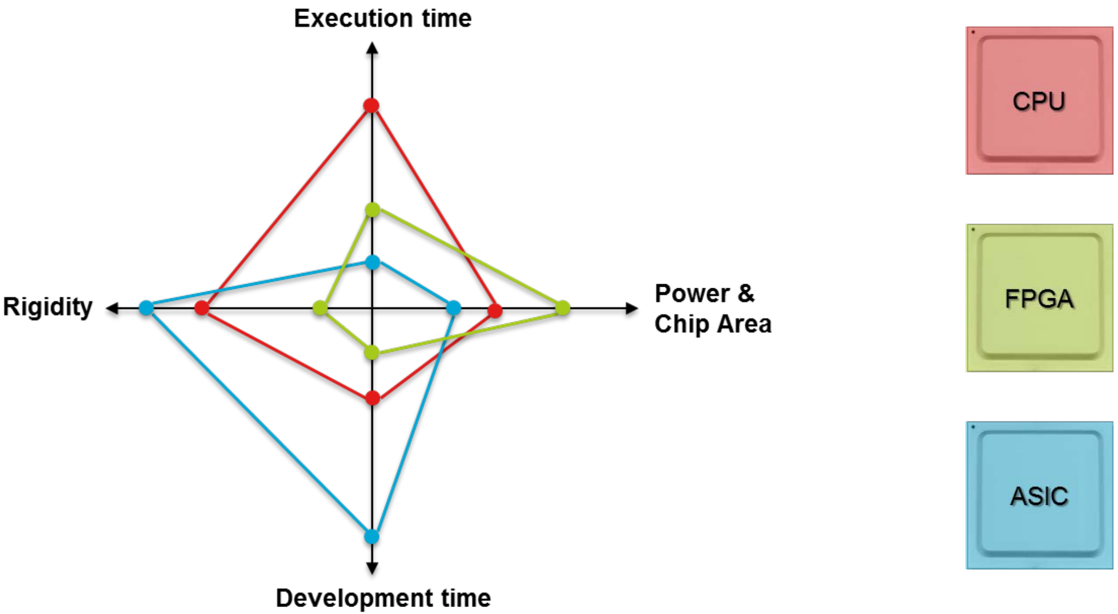
Hardware technology trends



Hardware technology trends

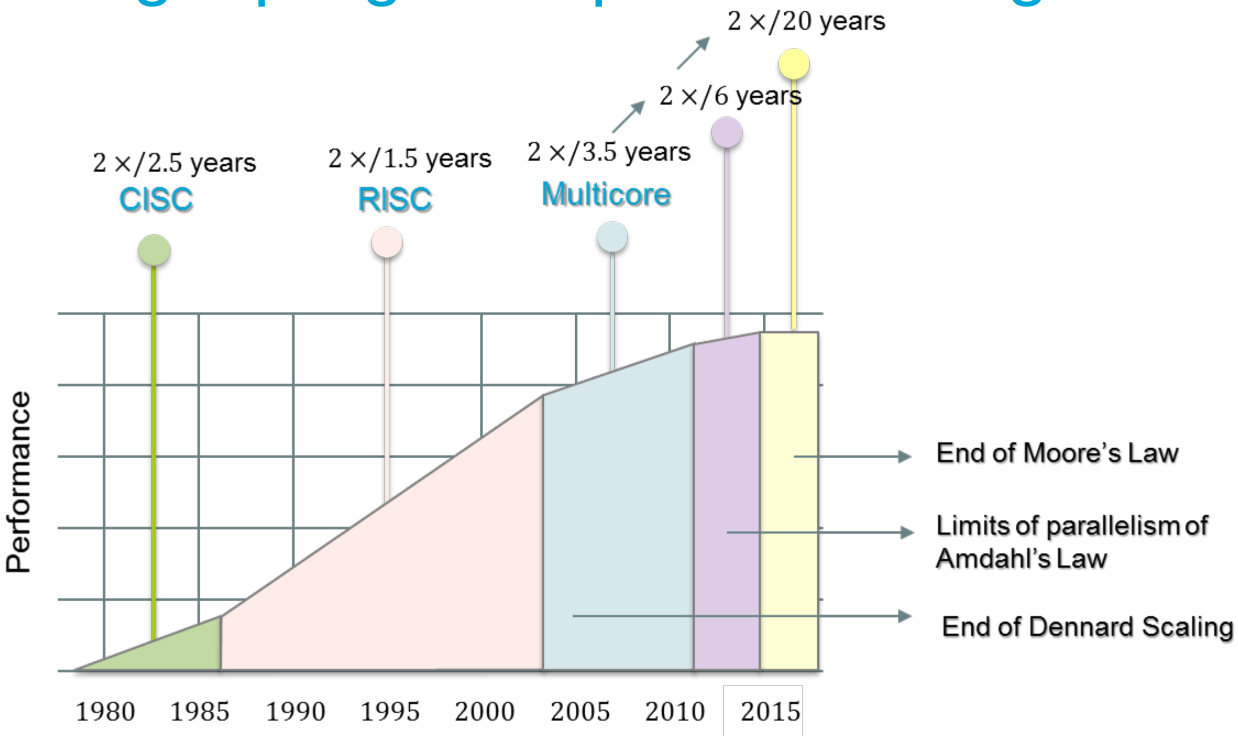


Hardware technology trends

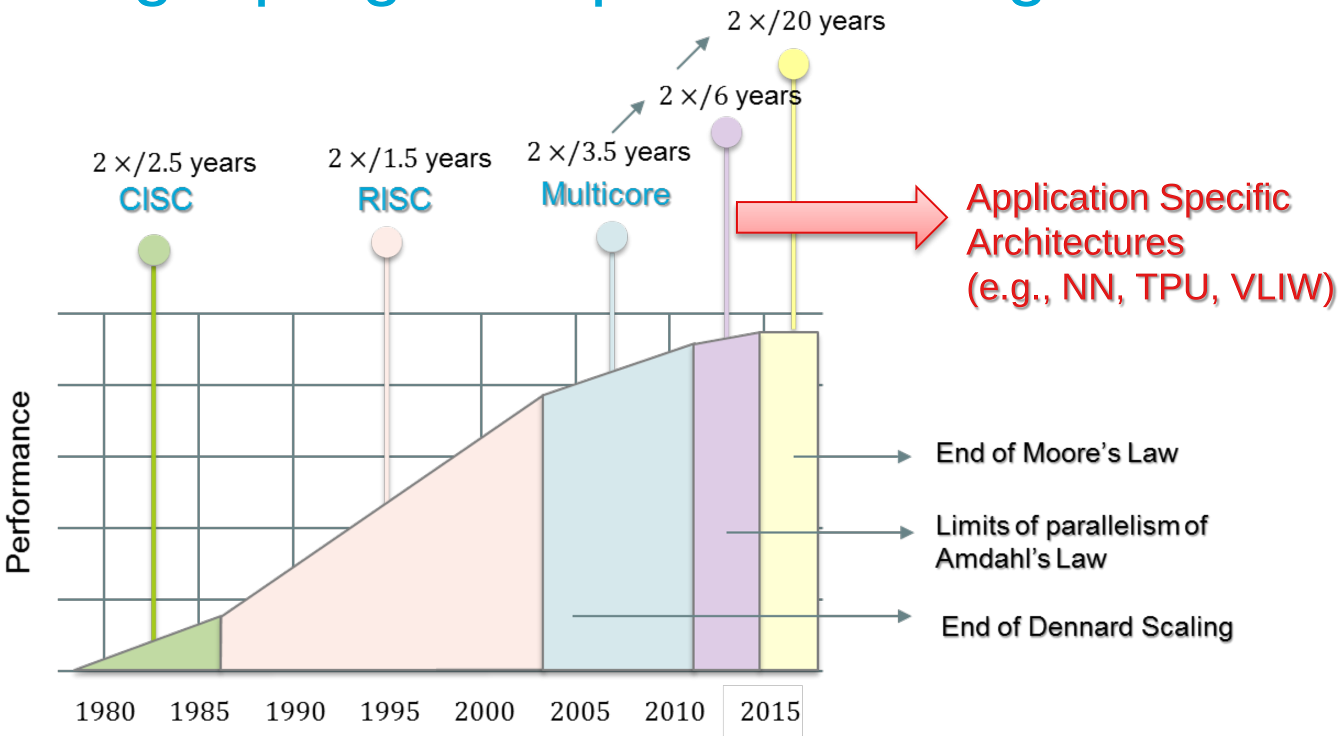


- **Implementation platform: FPGAs** - trade-off between advantages of
  - **general purpose processors** (flexibility, short development time) and
  - **ASICs** (fast execution time)

Single program speed end of growth



Single program speed end of growth



Application Specific Architectures
(e.g., NN, TPU, VLIW)

Application Specific Architectures

- ❑ Achieve higher efficiency by tailoring the architecture to the characteristics of a domain of applications
- ❑ More effective parallelism for a specific domain

VLIW application specific architectures

VLIW characteristics:

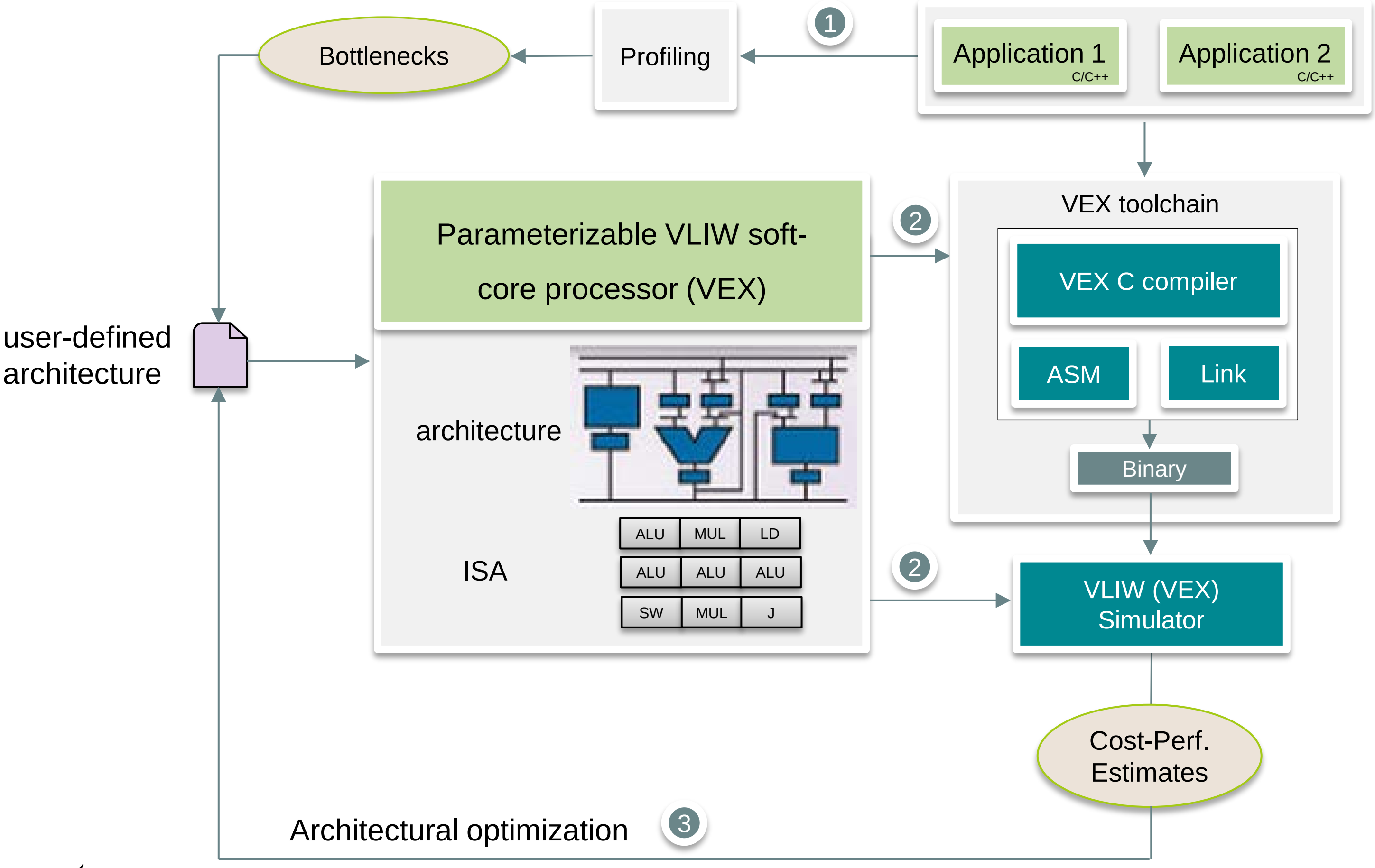
- ❑ VLIW contains multiple primitive RISC-style instructions that can be executed in parallel by the processor functional units
- ❑ The compiler decides which primitive non-interdependent instructions to execute in parallel, and packs them into a very long instruction word

Why VLIW:

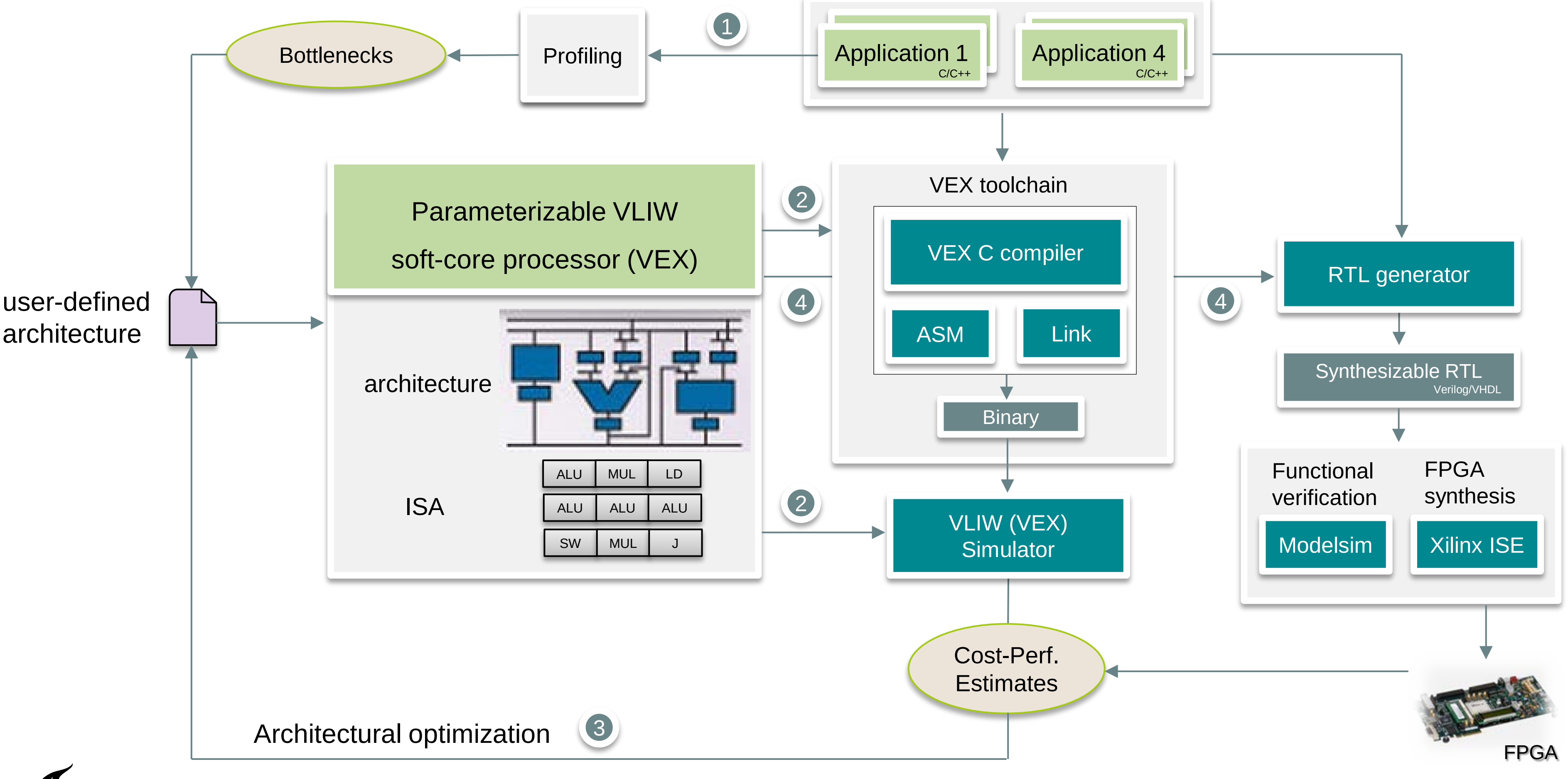
- ❑ Ability to extract a highly parallel instruction stream
- ❑ Hardware is regular (benefits layout) and scalable (issue width and number of execution units)
- ❑ Enables higher performance without the inherent complexity of other architectures (e.g., superscalar)

Lab overview

- **Given:**
  - Parameterizable VLIW soft processor architecture
  - A set of applications with common characteristics
  - FPGA implementation platform
- **Goal:**
  - Modify the VLIW processor architecture such that it can execute in a most efficient manner the given set of applications (in software for Assignment 1; on FPGA for Assignment 2)



Assignment 1 - 1 2 3



Bottlenecks

Profiling

1

Application 1
C/C++

Application 4
C/C++

Parameterizable VLIW
soft-core processor (VEX)

2

4

VEX toolchain

VEX C compiler

ASM

Link

Binary

4

RTL generator

Synthesizable RTL
Verilog/VHDL

user-defined
architecture

architecture

ISA

ALU	MUL	LD
ALU	ALU	ALU
SW	MUL	J

2

VLIW (VEX)
Simulator

Functional verification

FPGA synthesis

Modelsim

Xilinx ISE

Cost-Perf.
Estimates

FPGA

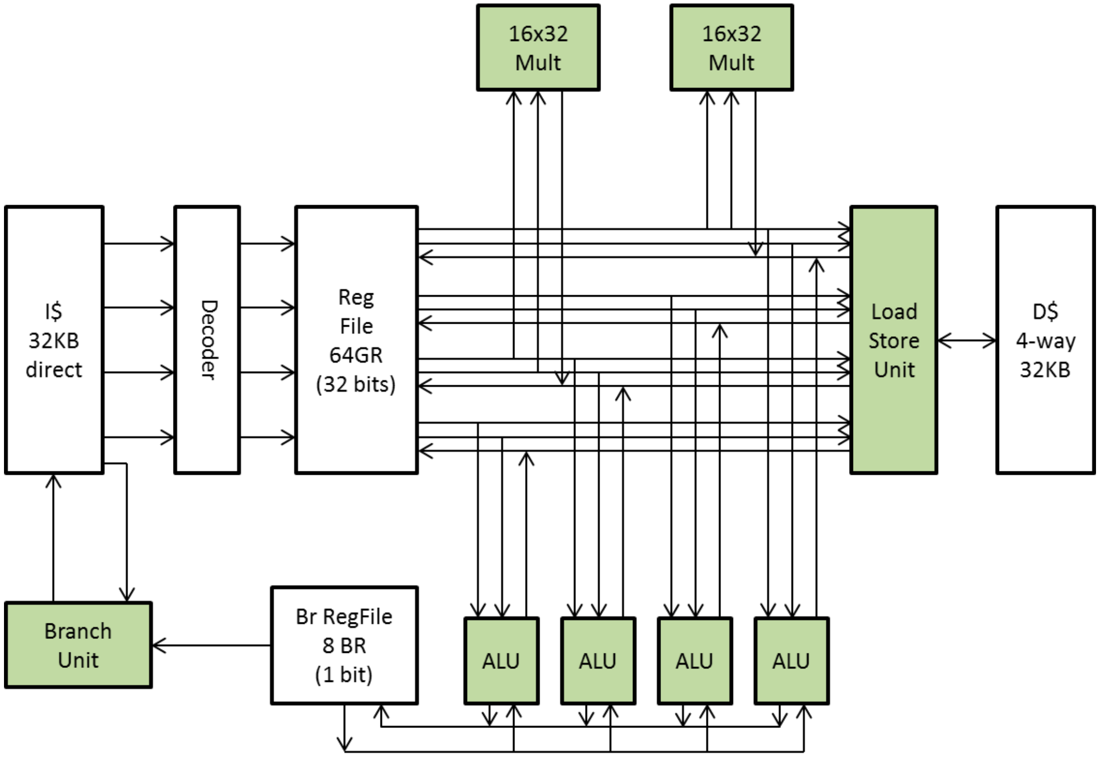
Architectural optimization 3

TU Delft

Assignment 1 - 1 2 3

Assignment 2 - 1 4 3

VEX default architecture

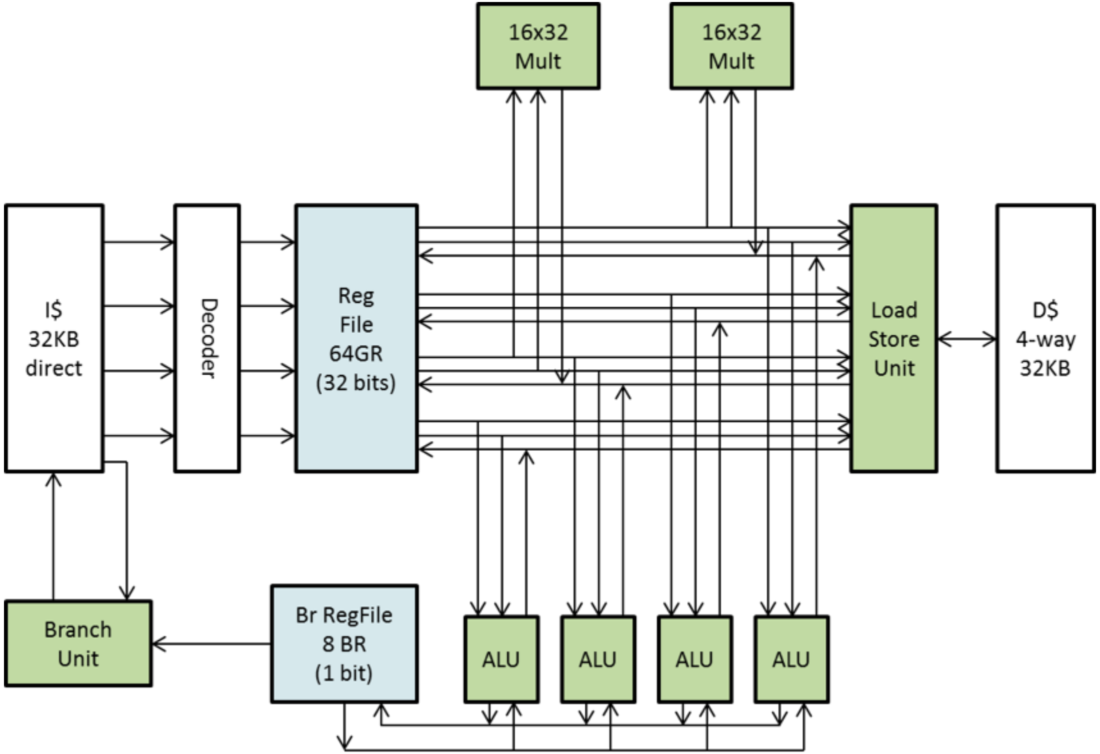


Functional Units

can be accessed in parallel

- ❑ ALU
- ❑ Mult
- ❑ Load/Store Unit
- ❑ Branch Unit

VEX default architecture



Functional Units

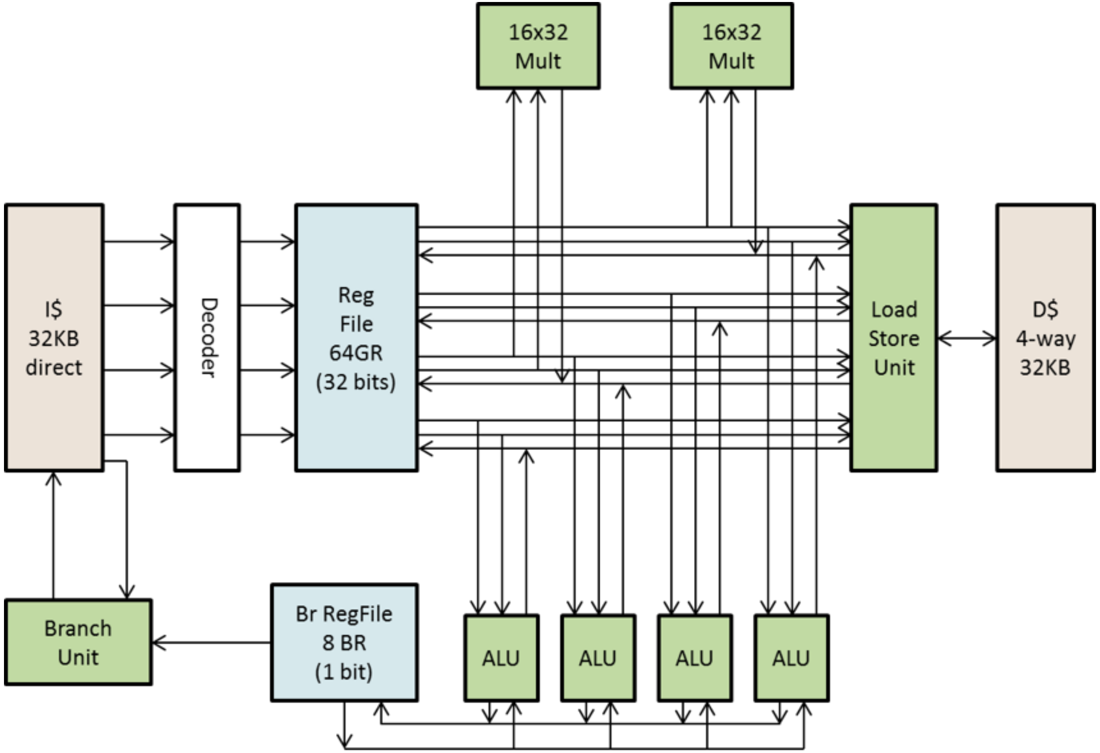
can be accessed in parallel

- ❑ ALU
- ❑ Mult
- ❑ Load/Store Unit
- ❑ Branch Unit

Registers

- ❑ 64 32-bit GR
- ❑ 8 1-bit BR

VEX default architecture



Functional Units

can be accessed in parallel

- ALU
- Mult
- Load/Store Unit
- Branch Unit

Registers

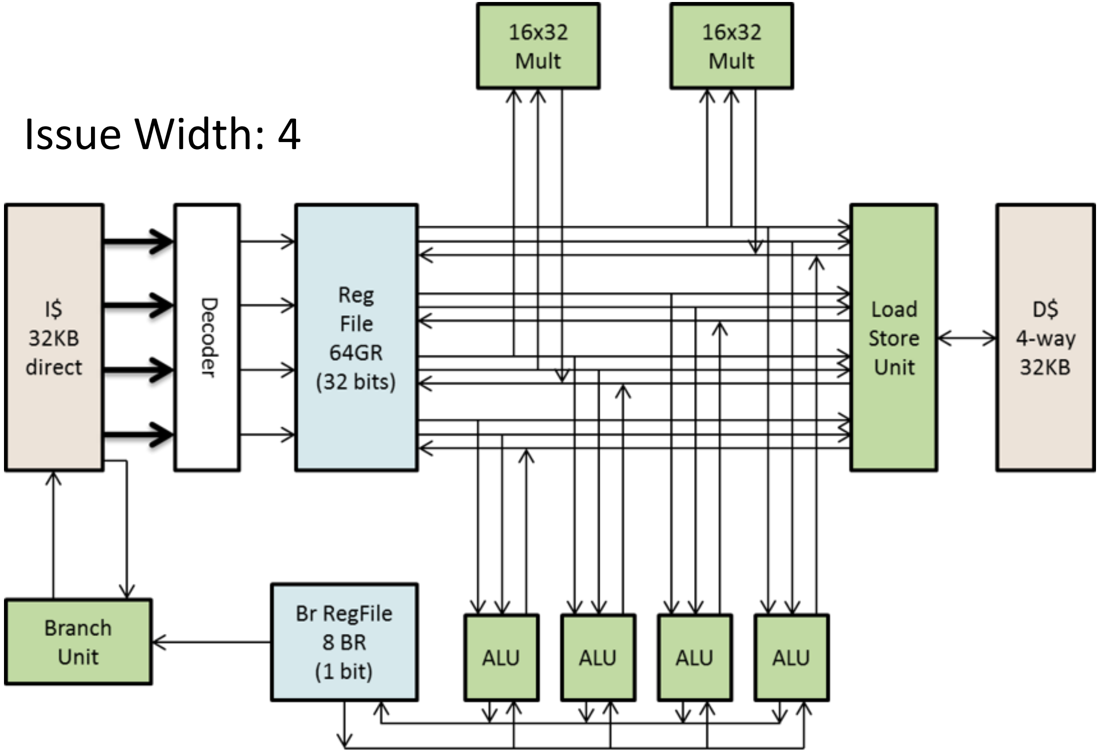
- 64 32-bit GR
- 8 1-bit BR

Memory

- 32KB Data Cache
- 32KB Instr. Cache

VEX default architecture

Issue Width: 4



Functional Units

can be accessed in parallel

- ❑ ALU
- ❑ Mult
- ❑ Load/Store Unit
- ❑ Branch Unit

Registers

- ❑ 64 32-bit GR
- ❑ 8 1-bit BR

Memory

- ❑ 32KB Data Cache
- ❑ 32KB Instr. Cache

TUDelft

Assignment 1 details

Given:

- 2 C-coded applications (common domain) as workload for each group
- Parametrizable VLIW (VEX) soft-core processor
- VEX toolchain

Goal:

- Determine ONE VEX architecture, optimal for both applications in terms of performance and area utilization.

Note: Consider both applications domain: general purpose (fast),
embedded (small area and low pwer), etc.

Method:

- Design Space Exploration (DSE) by changing the architecture configuration parameters.

VEX architecture configuration parameters

File: *configuration.mm*

	RES: IssueWidth	# operations executed in parallel
Functional Units	RES: ALU	# of ALU
	RES: Mpy	# of Mult
	RES: Memory	# of Load/Store Units
Registers	REG: $r0	# of 32-bit general purpose registers (GR)
	REG: $b0	# of 1-bit branch registers (BR)
Data Cache 32-bit connections	RES: MemLoad	# of connections for load operations
	RES: MemStore	# of connections for store operations
	RES: MemPft	# of connections for pre-fetch operations

VEX compiler optimization (OPTIONAL)

- Optimization via compiler flags

  E.g., -On, -autoinline

  (see Section 3.1, Optimization in *Assignment_1_VEX.pdf*)

- #pragma

Performance estimation

- Running the VEX simulator generates many results.

  Most important:

  File: *ta.log.000*

```
Total Cycles:             19308 (193.08 msec)
Execution Cycles:         12461 ( 64.54%)
Stall Cycles:              6847 ( 35.46%)
Nops:                      1204 (  6.24%)
Executed operations:      20579
```

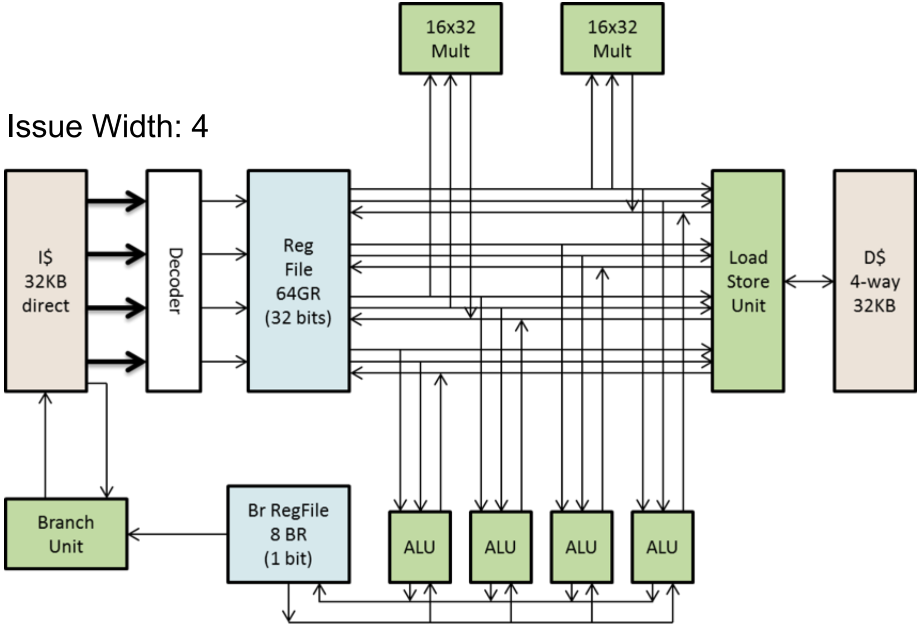
- “Total Cycles” includes cache accesses, which is beyond the scope of Assignment 1!

Area estimation

Area	Units	
A_{1_ALU}	3273	
A_{1_Mult}	40614	
$A_{1\_LW/SW}$	1500	
A_{64_GR}	26388	
A_{8_BR}	258	
A_{misc}	6739	
$A_{1\_connection}$	1000	Between lw/sw unit and data cache

A_{DCache}	568882
A_{ICache}	568882

Issue Width: 4



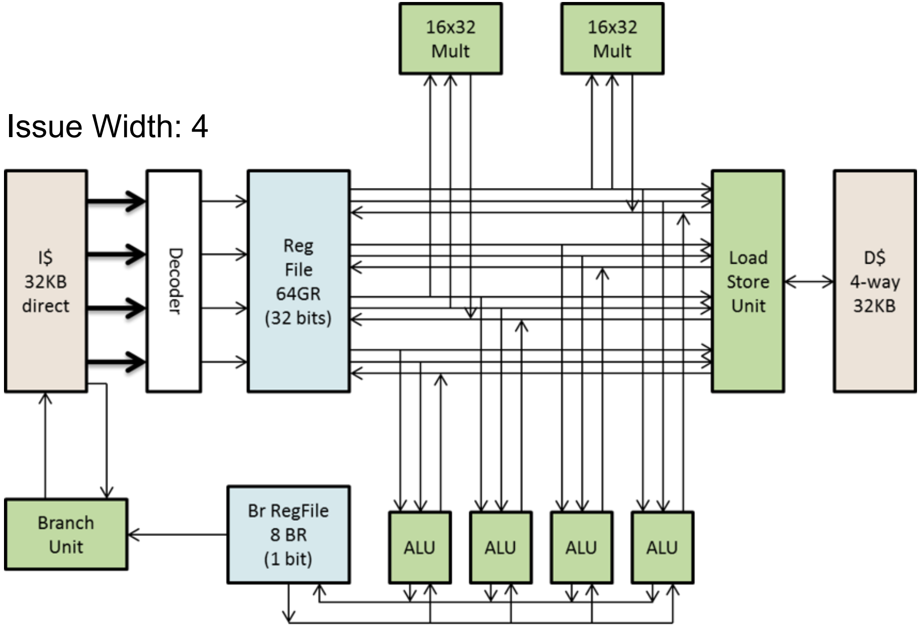
Total area for Assignment 1:

$$A \cong (\#\text{ALU}) * \text{A}_{1\_\text{ALU}} + (\#\text{Mult}) * \text{A}_{1\_\text{Mult}} + (\#\text{LW/SW}) * \text{A}_{1\_\text{LW/SW}} + \text{A}_{\text{GR}} + \text{A}_{\text{BR}} + (\#\text{Connections to LW/SW}) * \text{A}_{1\_\text{connection}}$$

Area estimation

Area	Units	
A_{1_ALU}	3273	
A_{1_Mult}	40614	
$A_{1\_LW/SW}$	1500	
A_{64_GR}	26388	
A_{8_BR}	258	
A_{misc}	6739	
$A_{1\_connection}$	1000	Between lw/sw unit and data cache
A_{DCache}	568882	
A_{ICache}	568882	

Issue Width: 4



Area scaling for register files (BR or GR):

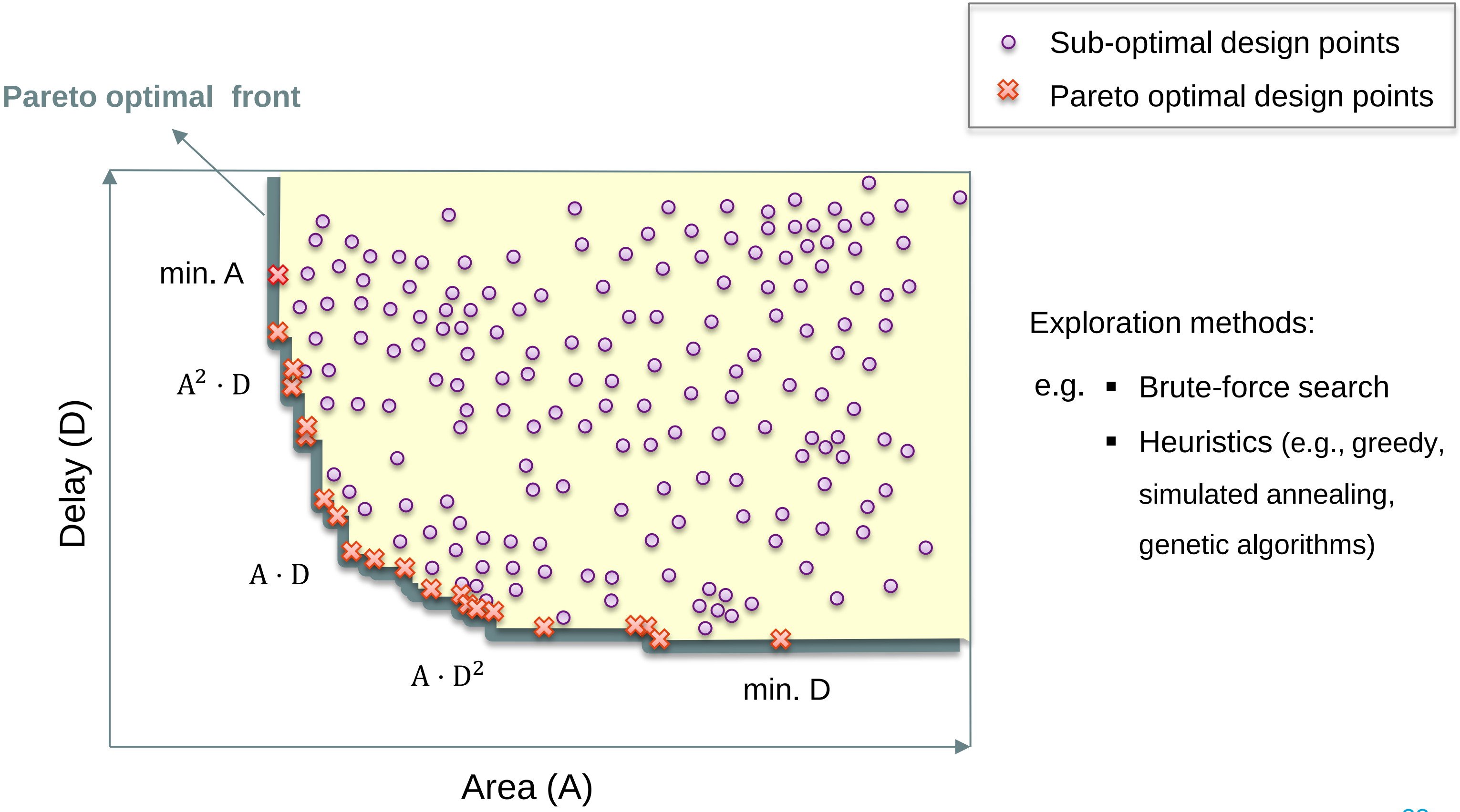
$$A_{REGS} \sim No_{regs} * (\#\text{ read ports}) * (\#\text{ write ports})$$

For each issue slot there are: 2 read ports, 1 write port

$$\Rightarrow A_{REGS} \sim No_{regs} * 2 * (\text{IssueWidth})^2$$

e.g., $A_{GR} = \frac{A_{64\_GR}}{64} * (\#\text{regs}) * \left(\frac{\text{Issue Width}}{4}\right)^2$

DSE – Area-Delay trade-off



Exploration methods:

e.g.
- Brute-force search
- Heuristics (e.g., greedy, simulated annealing, genetic algorithms)

Report

- Max. 4-page
- Identify application domain based on both benchmarks.
  - Scientific, embedded, general-purpose?
- Method of searching the design space.
- Choose a single processor design.
  - Why is it best for your application domain?
- Justify choices (trade-off for a balanced design).
- Follow requirements from course guide pdf.

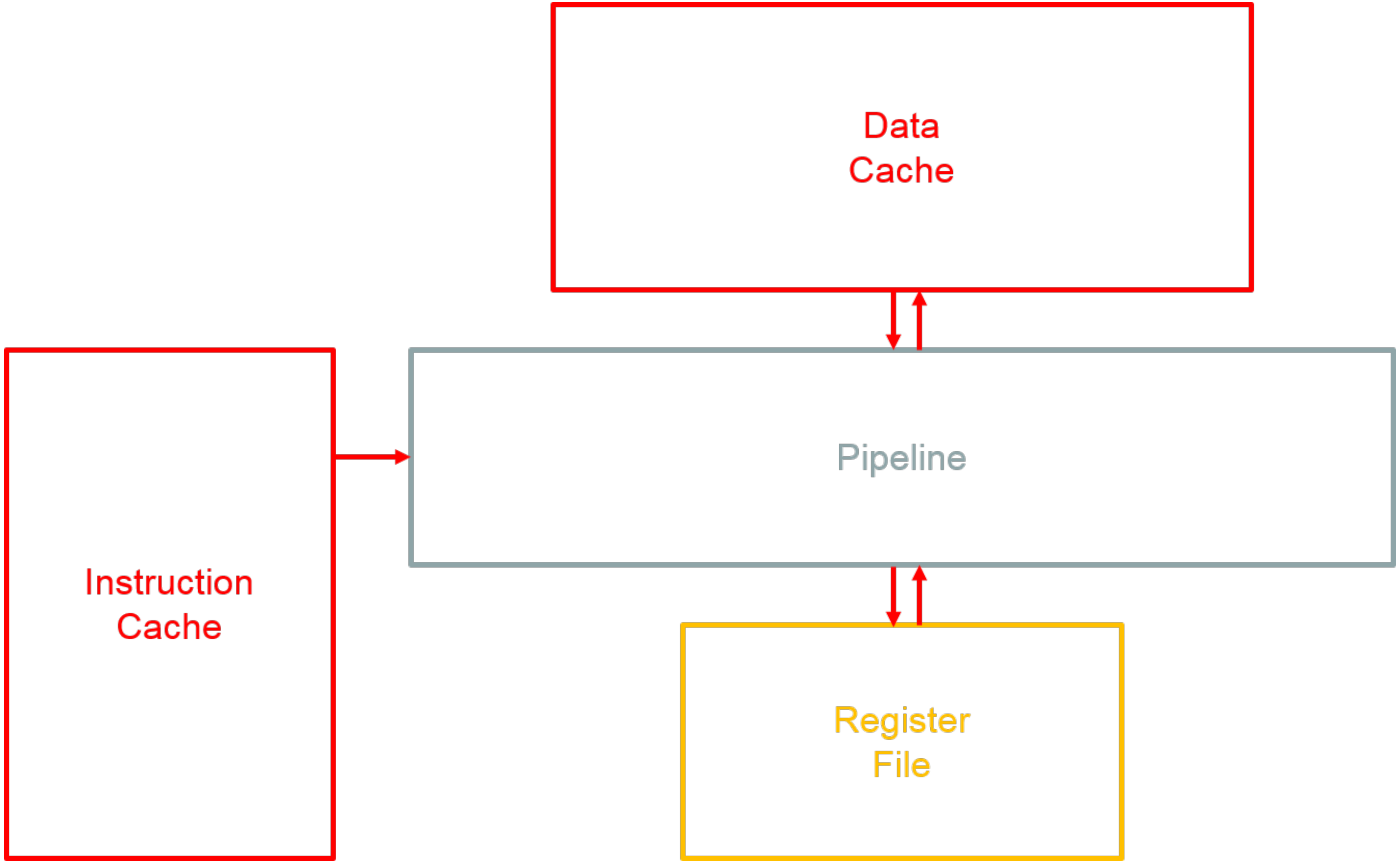
Submission requirements

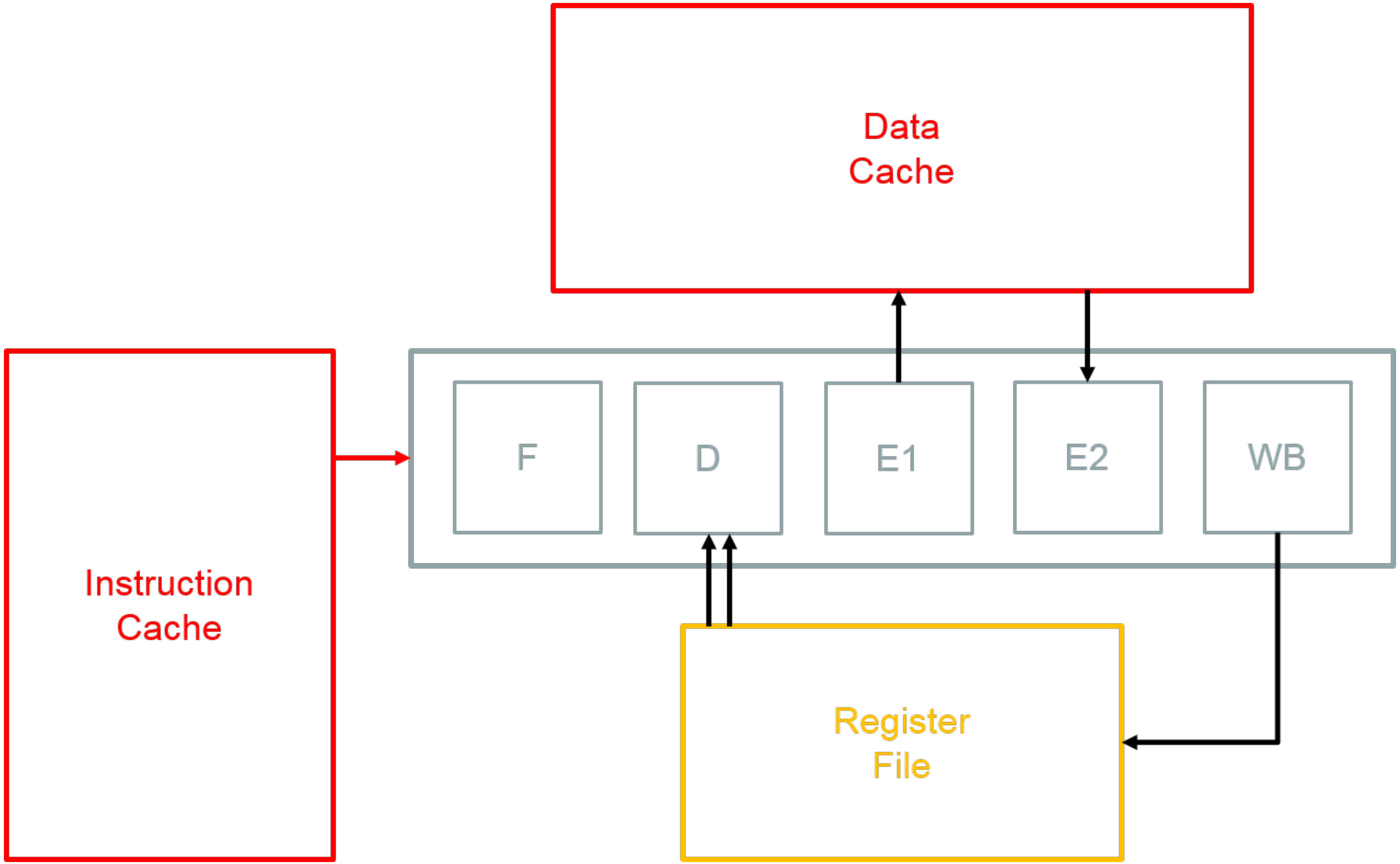
- Max. 4-page report pdf
- Archive with the final machine configuration file (*configuration.mm*)

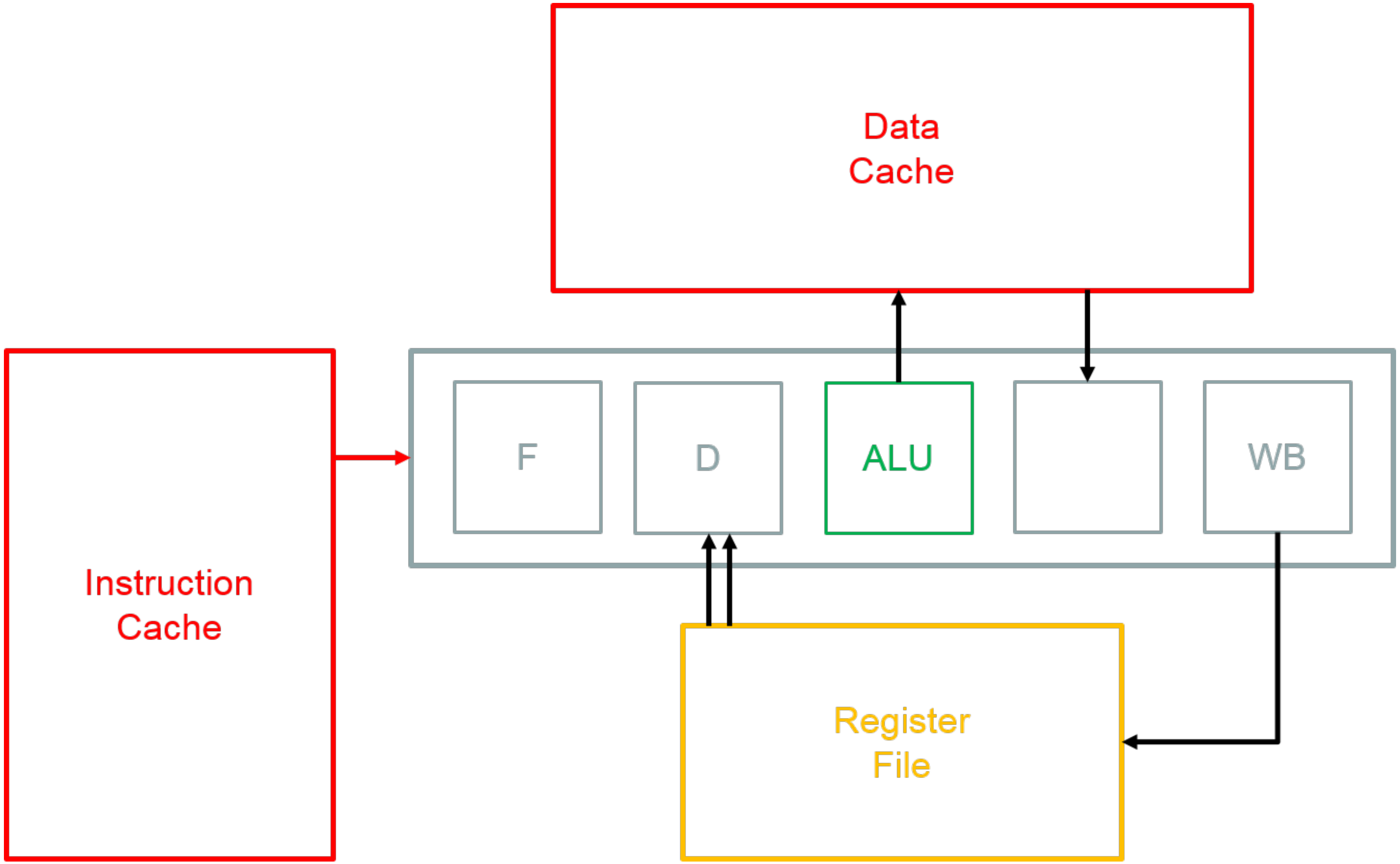
- Naming convention:
  - ET4074_2018_A1_report_group#.pdf
  - ET4074_2018_A1_src_group#.zip

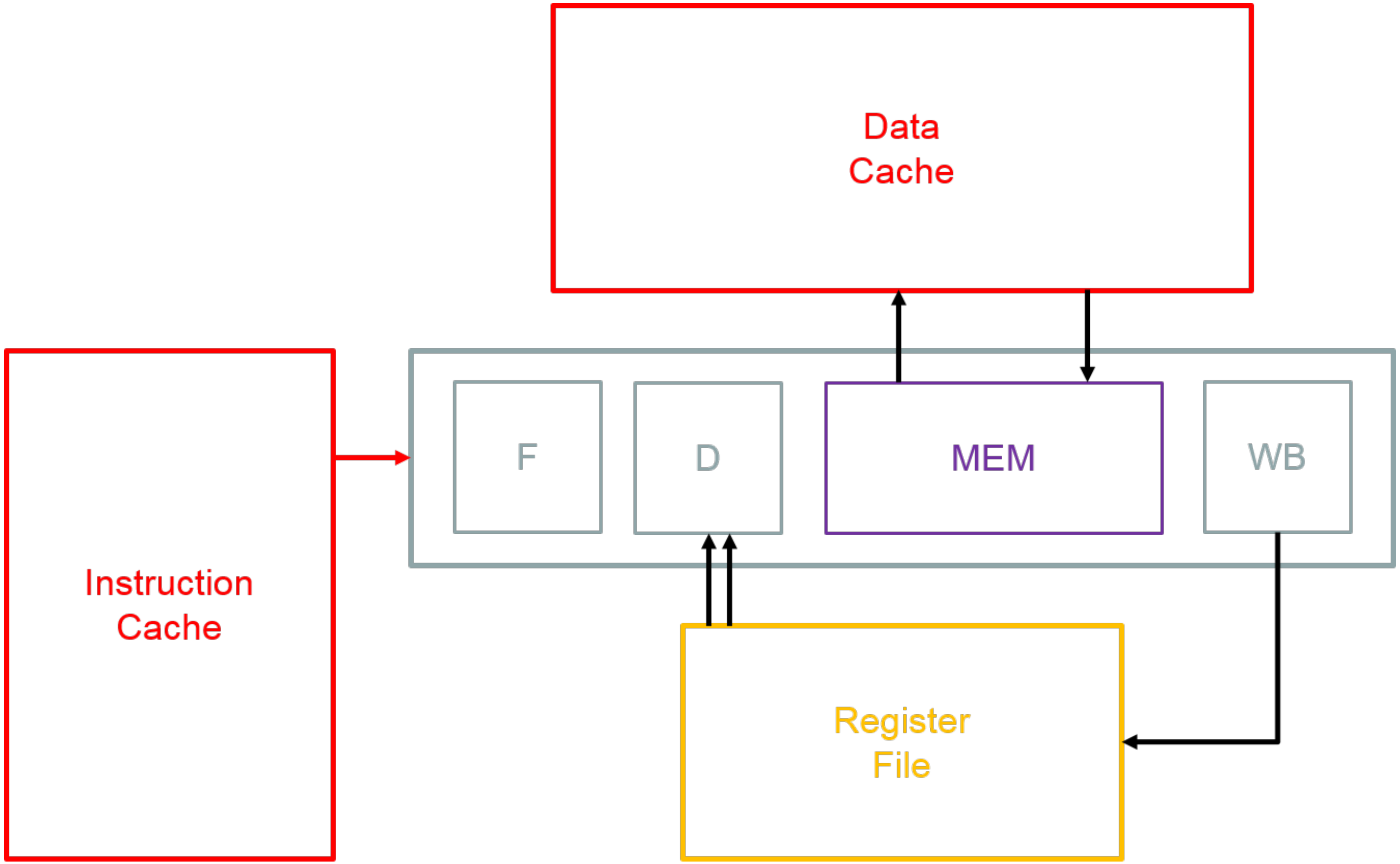
Extra slides

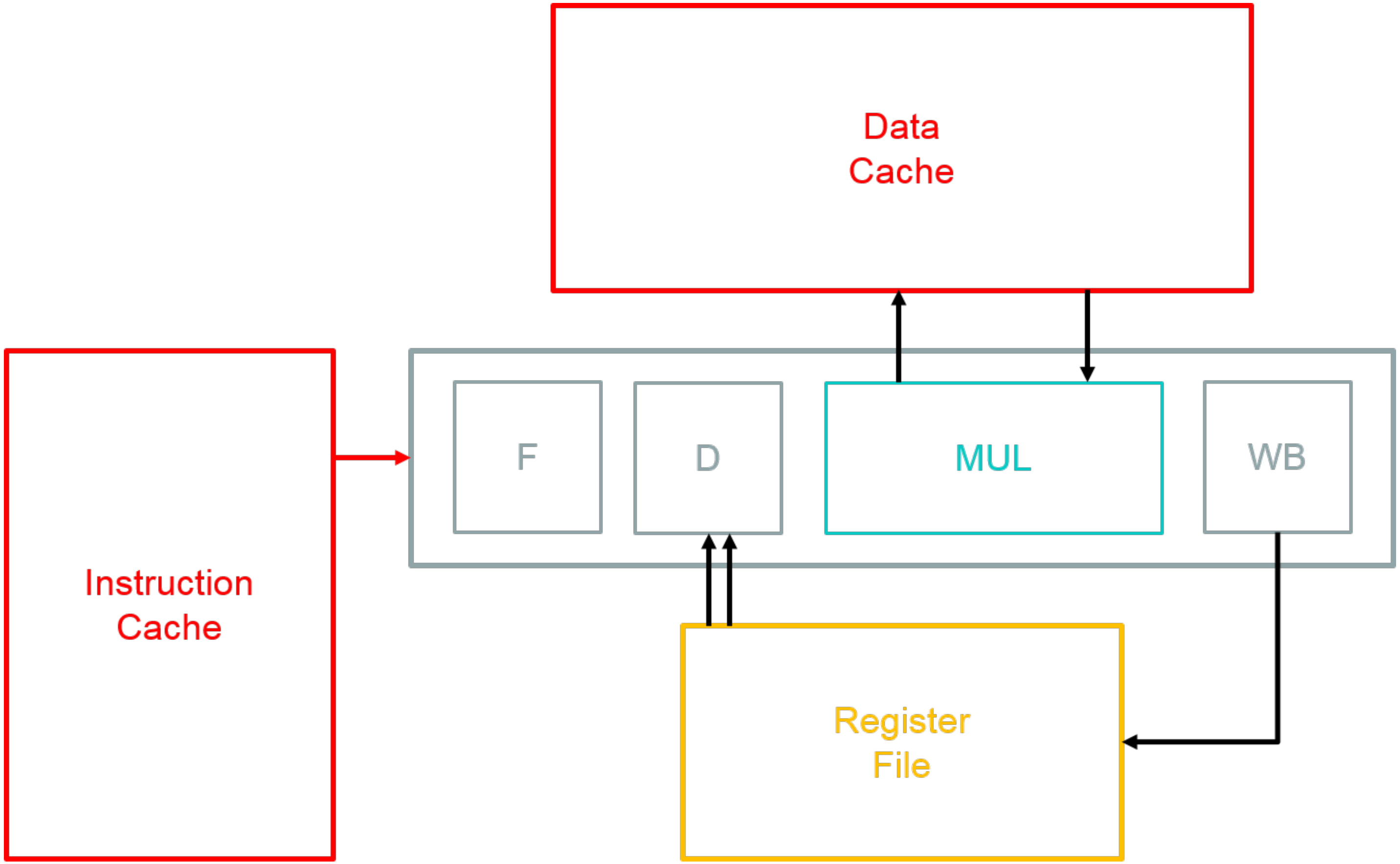
- Example - balancing functional units in a VLIW

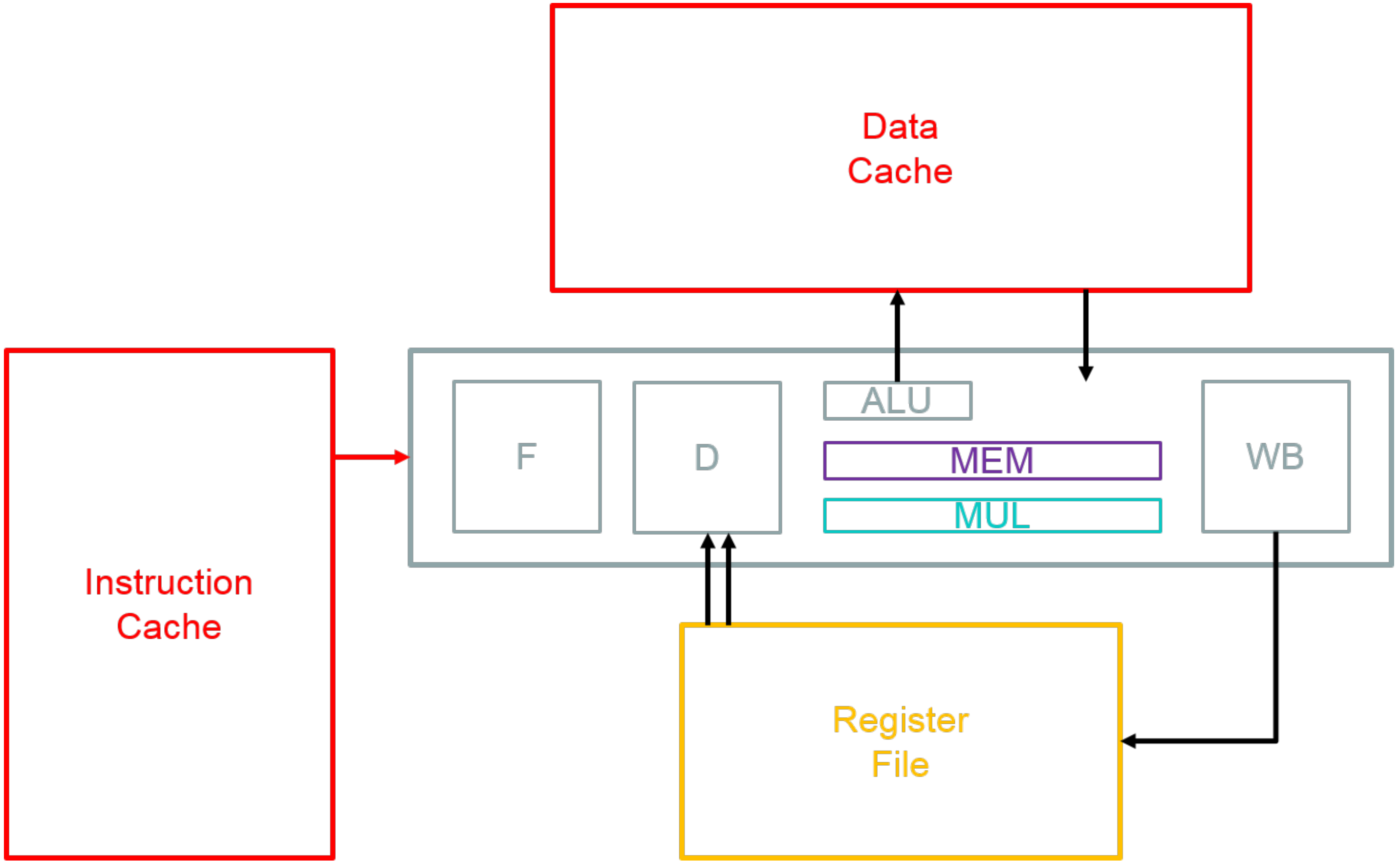

Data
Cache
Instruction
Cache
Pipeline
Register
File


Data
Cache
Instruction
Cache
F
D
E1
E2
WB
Register
File

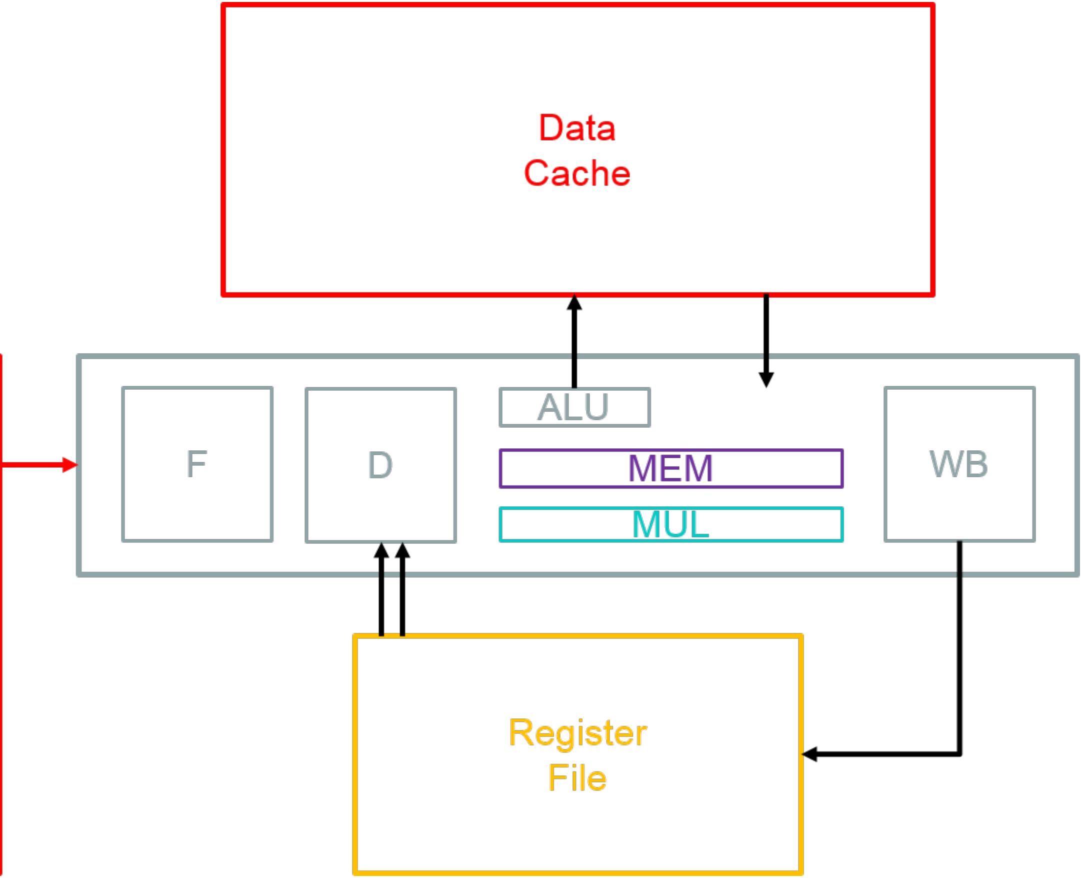

Data Cache
Instruction Cache
F
D
ALU
WB
Register File

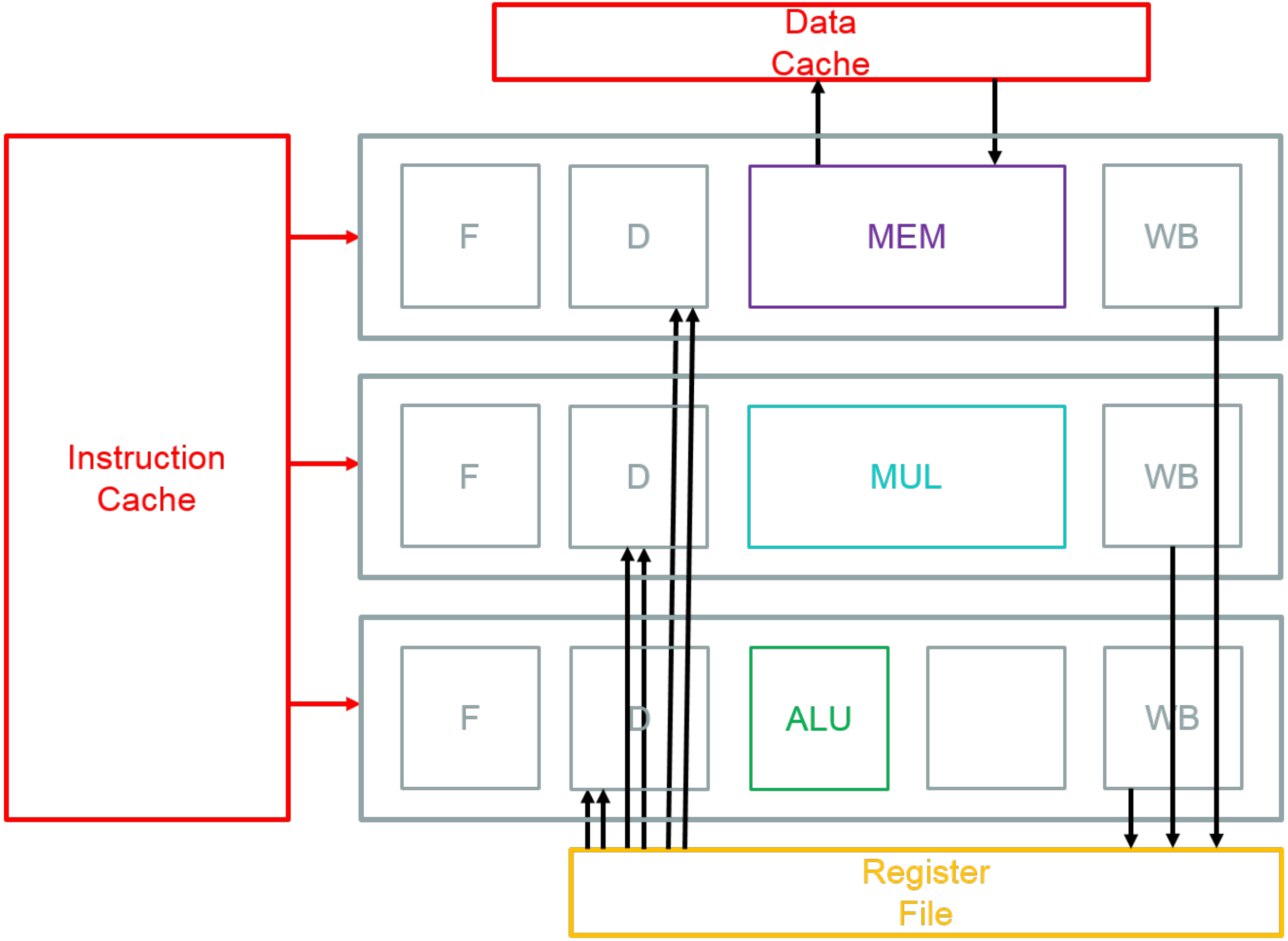

Data
Cache
Instruction
Cache
F
D
MEM
WB
Register
File

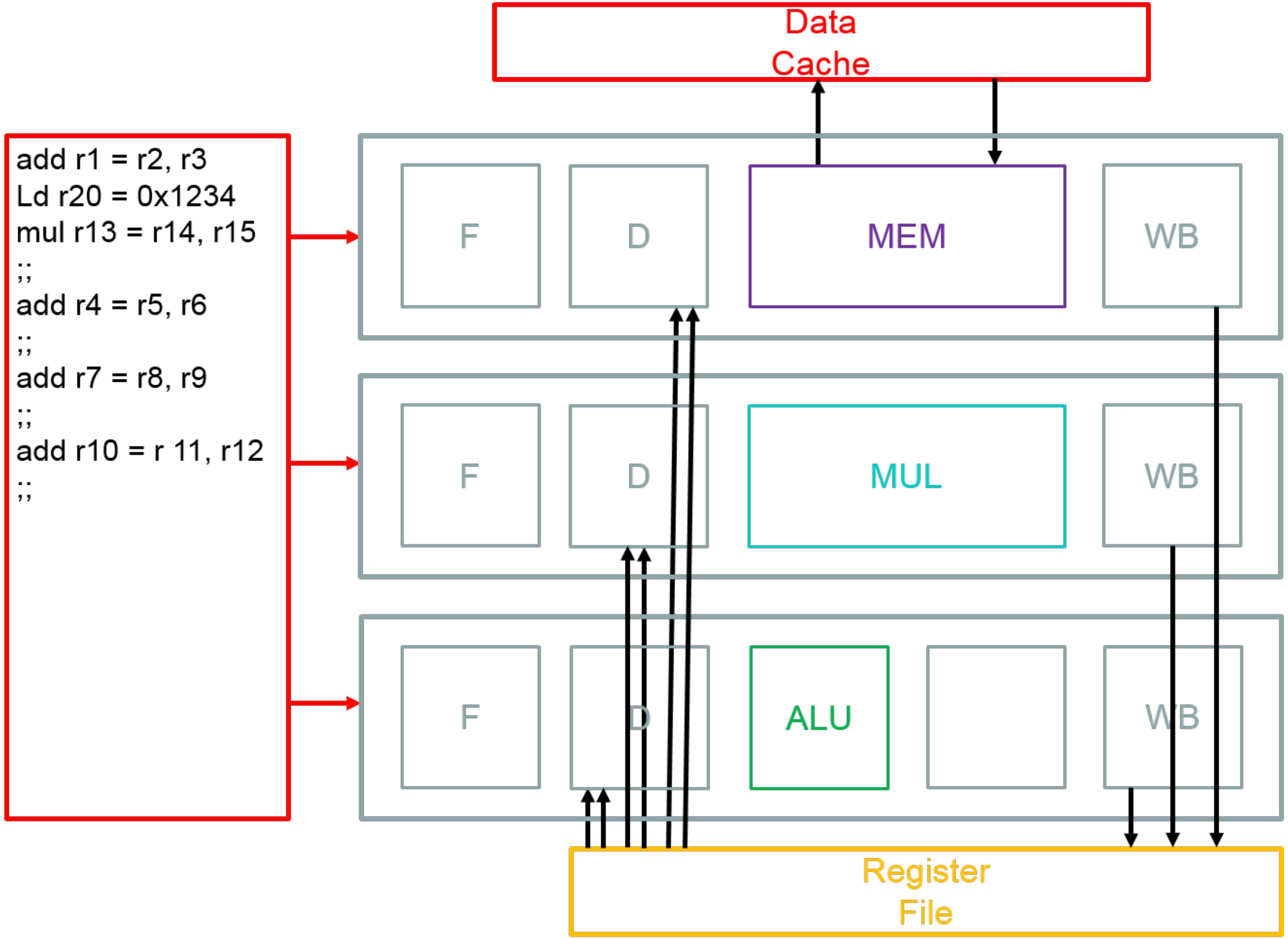

Data
Cache
Instruction
Cache
F
D
MUL
WB
Register
File


Data
Cache
Instruction
Cache
F
D
ALU
MEM
MUL
WB
Register
File

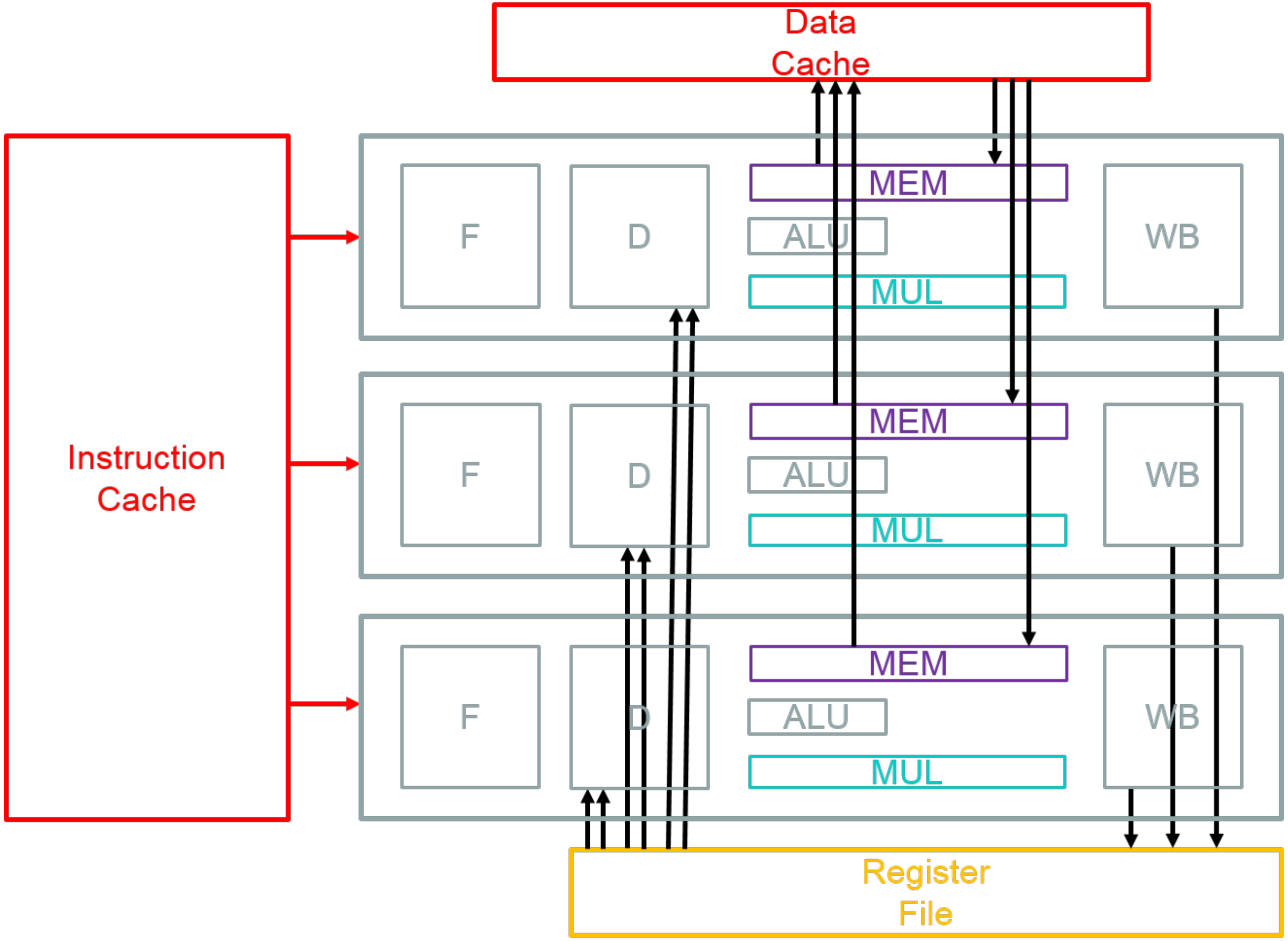
```
add r1 = r2, r3
;;
add r4 = r5, r6
;;
add r7 = r8, r9
;;
add r10 = r 11, r12
;;
mul r13 = r14, r15
;;
Ld r20 = 0x1234
;;
```



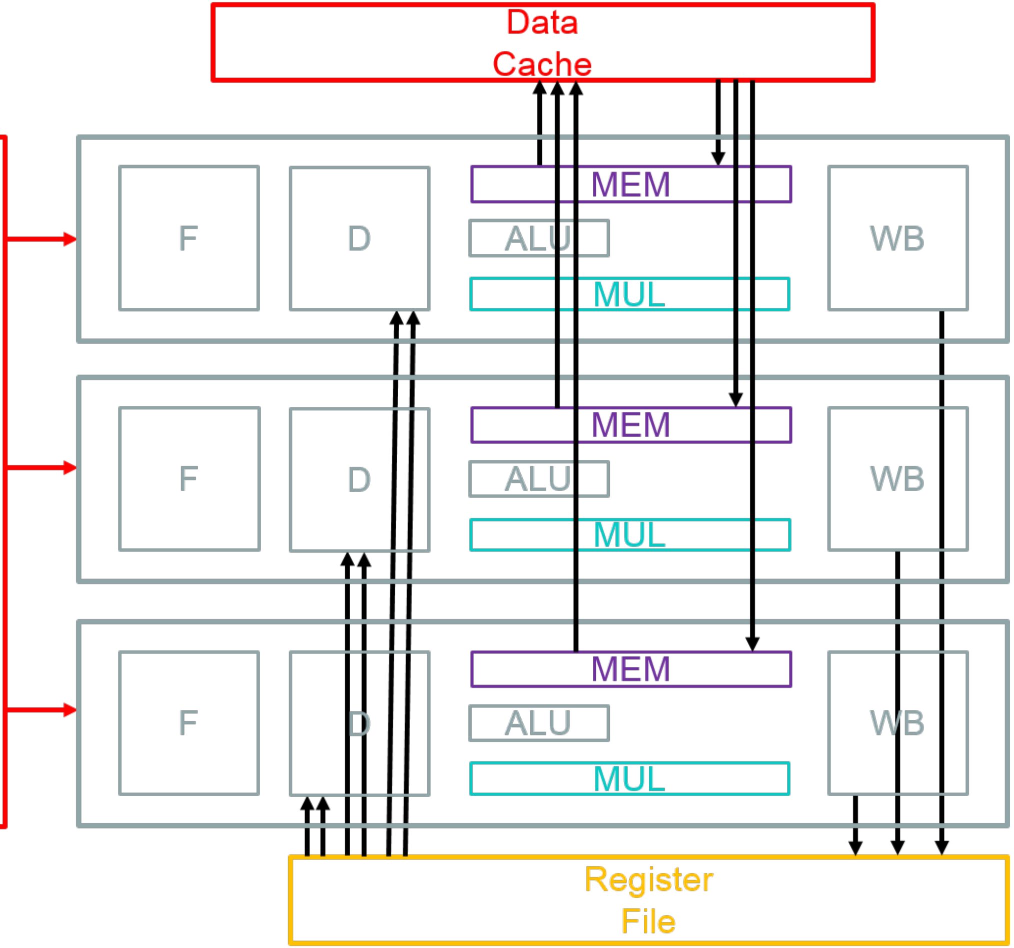

Data Cache
Instruction Cache
F
D
MEM
WB
F
D
MUL
WB
F
D
ALU
WB
Register File

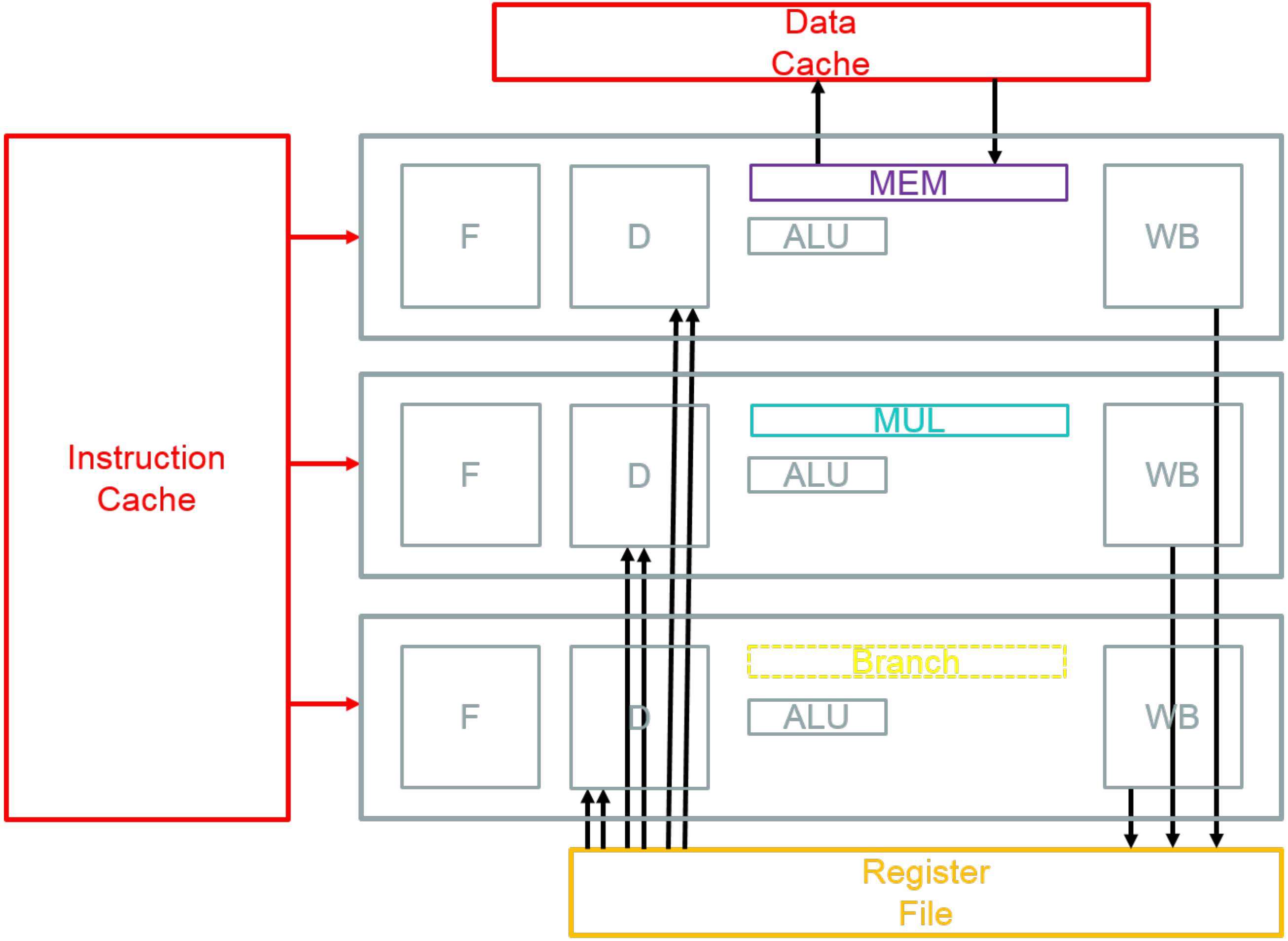


```
add r1 = r2, r3
Ld r20 = 0x1234
mul r13 = r14, r15
;;
add r4 = r5, r6
;;
add r7 = r8, r9
;;
add r10 = r 11, r12
;;
```

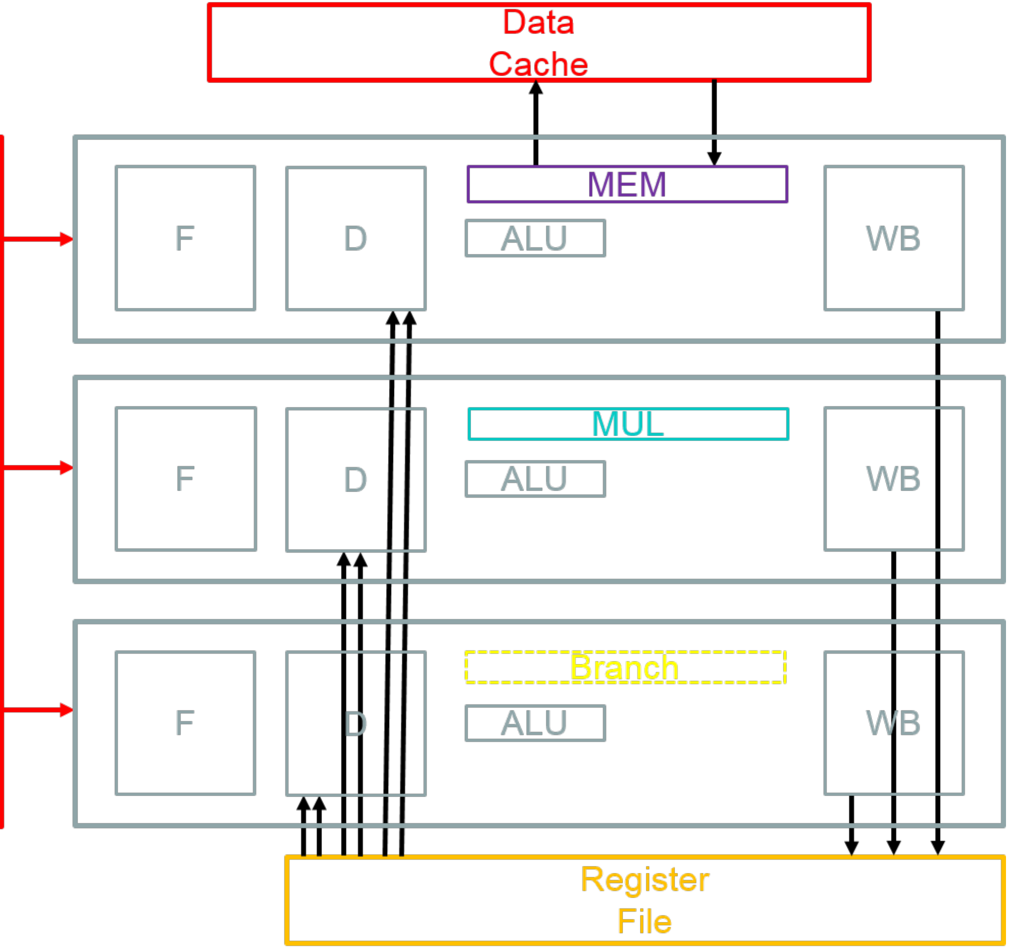

Data Cache
Instruction Cache
F
D
MEM
ALU
MUL
WB
F
D
MEM
ALU
MUL
WB
F
D
MEM
ALU
MUL
WB
Register File

```
add r1 = r2, r3
Ld r20 = 0x1234
mul r13 = r14, r15
;;
add r4 = r5, r6
add r7 = r8, r9
add r10 = r 11, r12
;;
```




Data
Cache
Instruction
Cache
F
D
MEM
ALU
WB
F
D
MUL
ALU
WB
F
D
Branch
ALU
WB
Register
File

```
add r1 = r2, r3
Ld r20 = 0x1234
mul r13 = r14, r15
;;
add r4 = r5, r6
add r7 = r8, r9
add r10 = r 11, r12
;;
```


Data
Cache
F
D
MEM
ALU
WB
F
D
MUL
ALU
WB
F
D
Branch
ALU
WB
Register
File