Exercise Session 1

Performance, Amdhal's Law, Pipeline

Advanced Computer Architectures

Politecnico di Milano

March 5th, 2025

Alessandro Verosimile <alessandro.verosimile@polimi.it>

POLITECNICO MILANO 1863

POLITECNICO MILANO 1863 NECST laboratory

Who am I

Research interests

- Machine Learning
- Hardware acceleration
- Embedded systems

Alessandro Verosimile
alessandro.verosimile@polimi.it

PhD student in Information Technology @ Politecnico di Milano

Research intern in **Advanced Micro Devices (AMD)**

Material

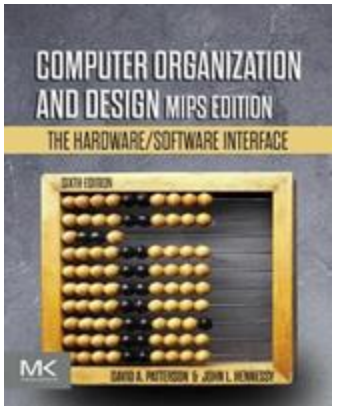
https://santambrogio.faculty.polimi.it/dida/aca/2025/index.htm

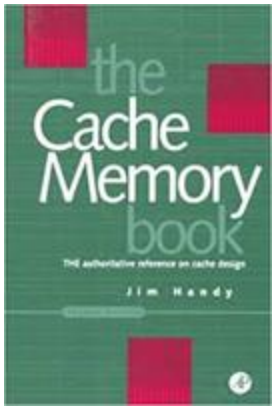
Dropbox url: Exe folder

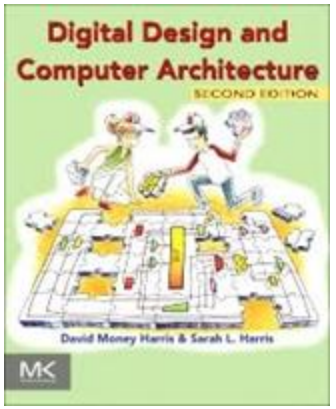
OPTIONAL

Textbook: Hennessy and Patterson, Computer Architecture: A Quantitative Approach

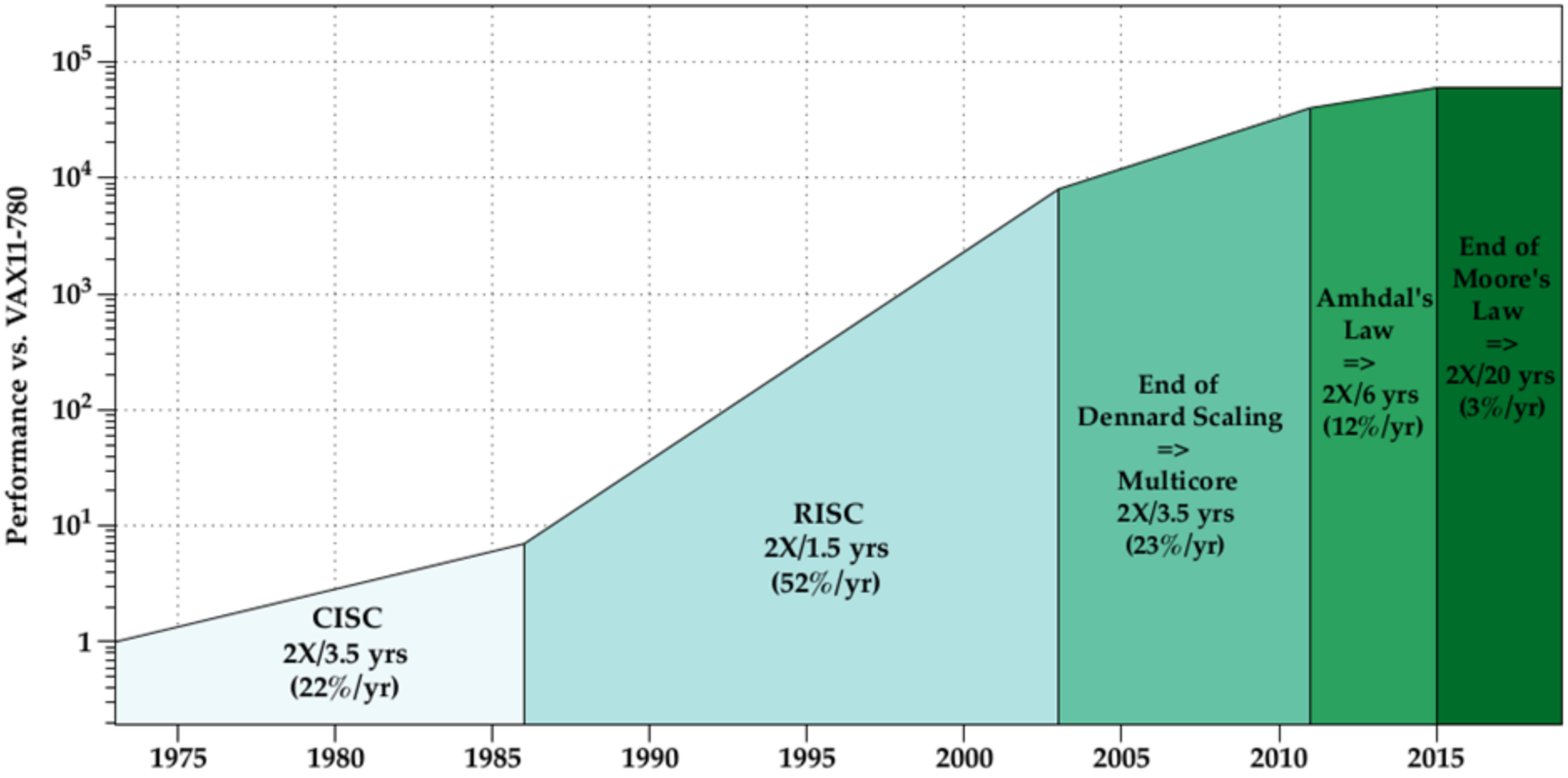
Other Interesting Reference







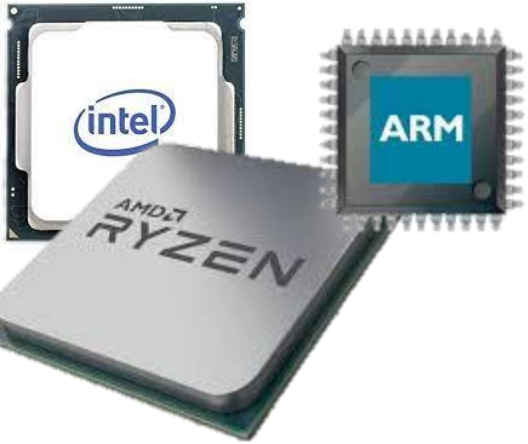
Motivations

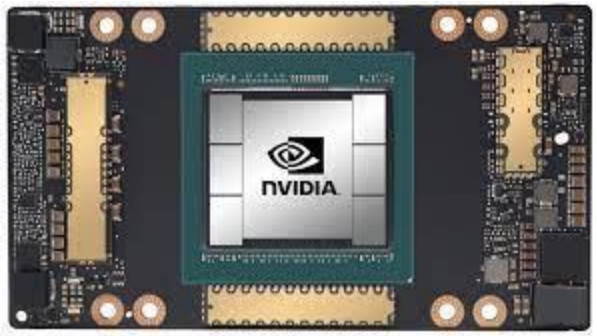


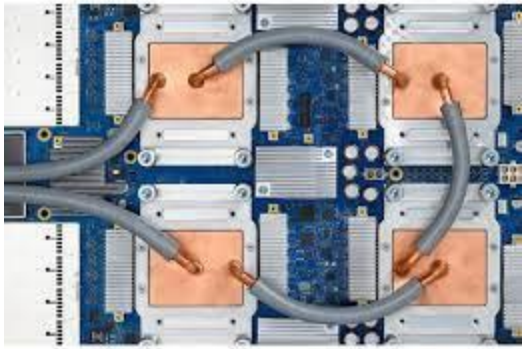
Adapted from E.Del Sozzo. On how to effectively target fpgas from domain specific tools. 2019.
Data from: J. L Hennessy and D. A Patterson. Computer architecture: a quantitative approach 6th edition. Elsevier, 2018.

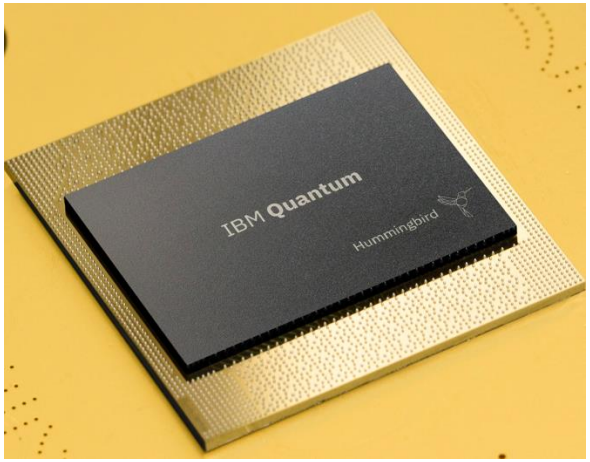
The End Goal is…
Pick the Best Architecture

(it is a matter of trade-offs)









Exe 1: Throughput vs Response time

Recall: Throughput vs Response time

- Two Metrics:

–Computer system user

- Minimize elapsed time for program execution:

response time: execution time = time_end – time_start

–Computer center manager

- Maximize completion rate = #jobs/sec

throughput: total amount of work done in a given time

Exe 1: Throughput vs Response time

Exe 1: Throughput vs Response time

What will happen if…

Exe 1: Throughput vs Response time

(1
)

What will happen if…

(1)we replace with a faster version?

Exe 1: Throughput vs Response time

(1)

What will happen if…

(1)we replace with a faster version?

(2) We add multiple parallel systems for independent tasks?

(2)

Case 1: Scale-up

(1
)

Case 1: Scale-up

(1
)

decrease response time and throughput will increase

Case 2: Scale-out

(2
)

Case 2: Scale-out

For sure Throughput will increase

(2
)

Case 2: Scale-out

For sure Throughput will increase

Response time? (2)

Case 2: Scale-out

For sure Throughput will increase

Response time? (2)

Yes, if there were a queue to serve, which was waiting for computing resources

Recall: CPU time

- Instruction Count, IC
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, Instruction Set Architecture
- Cycles per instructions, CPI
  - Determined by ISA and CPU organization
  - Overlap among instructions (pipelining) reduces this term
- Time/cycle
  - Determined by technology, organization and circuit design

Exe 2: Performance Problem

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Given the following frequencies of occurrence of the instructions for the two CPUs

Exe 2: Performance Problem

Consider two CPUs: CPU1 and CPU2.

CPU1 has clock cycle of 2 ns while CPU2 has an operating frequency of 700MHz.

Given the following frequencies of occurrence of the instructions for the two CPUs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Exe 2: Questions

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

A. Compute the average CPI for CPU1 and CPU2

B. Which is the fastest CPU?

Exe 2: AVG CPIs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Recall: $$CPI = \frac{\text{Clock cycles}}{\text{Instruction}}$$ $$CPI = \sum_{i=1}^{n} CPI_i * F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}$$

Exe 2: AVG CPIs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Recall: $CPI = \frac{\text{Clock cycles}}{\text{Instruction}}$ $CPI = \sum_{i=1}^{n} CPI_i * F_i$ where $F_i = \frac{I_i}{\text{Instruction Count}}$

$CPI_1 = 0.3 * 2 + 0.1 * 3 + 0.2 * 4 + 0.3 * 2 + 0.1 * 4 =$

Exe 2: AVG CPIs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Recall: $CPI = \frac{\text{Clock cycles}}{\text{Instruction}}$ $CPI = \sum_{i=1}^{n} CPI_i * F_i$ where $F_i = \frac{I_i}{\text{Instruction Count}}$

$CPI_1 = 0.3 * 2 + 0.1 * 3 + 0.2 * 4 + 0.3 * 2 + 0.1 * 4 =$

$= 0.6 + 0.3 + 0.8 + 0.6 + 0.4 = 2.7$

Exe 2: AVG CPIs

Operation type	Frequency	CPU1 CYCLE	CPU2 CYCLE
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

Recall: $CPI = \frac{\text{Clock cycles}}{\text{Instruction}}$ $CPI = \sum_{i=1}^{n} CPI_i * F_i$ where $F_i = \frac{I_i}{\text{Instruction Count}}$

$CPI_1 = 0.3 * 2 + 0.1 * 3 + 0.2 * 4 + 0.3 * 2 + 0.1 * 4 =$

$= 0.6 + 0.3 + 0.8 + 0.6 + 0.4 = 2.7$

$CPI_2 = 0.3 * 2 + 0.1 * 3 + 0.2 * 3 + 0.3 * 2 + 0.1 * 3 =$

$= 0.6 + 0.3 + 0.6 + 0.6 + 0.3 = 2.4$

Exe 2.b: Which is the fastest CPU?

Exe 2.b: Which is the fastest CPU?

Recall: **"X is n times faster than Y" means**

$$\frac{Performance(X)}{Performance(Y)} = \frac{Exe(Y)}{Exe(X)}$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

Exe 2.b: Which is the fastest CPU?

$$\frac{EXE_{CPU_1}}{EXE_{CPU_2}}$$

Exe 2.b: Which is the fastest CPU?

$$\frac{EXE_{CPU_1}}{EXE_{CPU_2}} = \left(\frac{IC_1 * CPI_1}{F_1}\right) * \left(\frac{F_2}{IC_2 * CPI_2}\right)$$

Exe 2.b: Which is the fastest CPU?

$$\frac{EXE_{CPU_1}}{EXE_{CPU_2}} = \left(\frac{IC_1 * CPI_1}{F_1}\right) * \left(\frac{F_2}{IC_2 * CPI_2}\right)$$

$$\frac{IC_1 * CPI_1 * F_2}{IC_2 * CPI_2 * F_1} = \frac{CPI_1 * F2}{CPI_2 * F_1}$$

$$= \frac{2.7 * 700MHz}{2.4 * 500MHz} = \frac{1890}{1200} = 1.575$$

CPU2 is 1.575 faster than CPU1

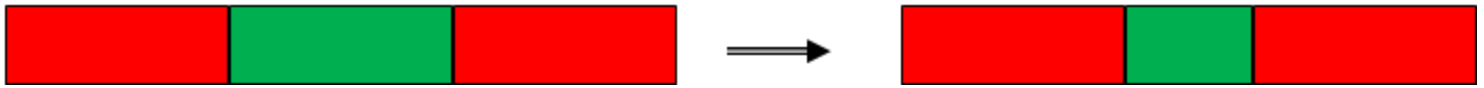
Exe 3: Amdahl's Law

Recall

$$Speedup_{overall} = \frac{Executiontime_{old}}{Executiontime_{new}} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

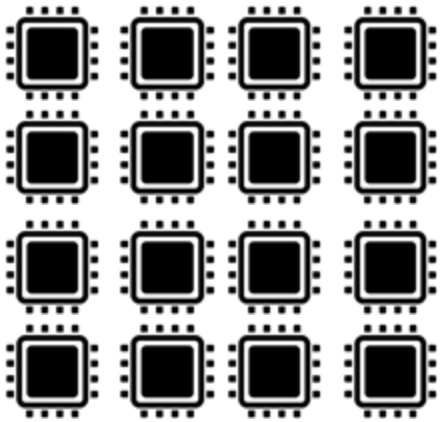
Best you could ever hope to do:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced})}$$

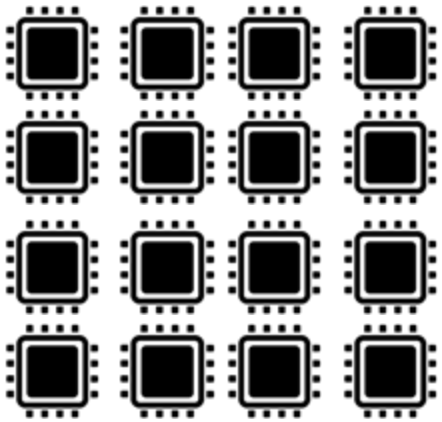


Example of good trade-off: [In-Datacenter Performance Analysis of a Tensor Processing Unit]

Exe 3 : Amdahl's Law



Exe 3 : Amdahl's Law



Exe 3 : Amdahl's Law

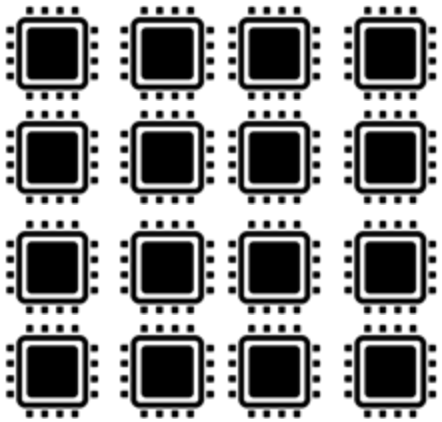


Image Processing task on FPGA is *2.86x*[1] times faster

[1] Conficconi, D., D'Arnese, E., Del Sozzo, E., Sciuto, D., & Santambrogio, M. D. "A framework for customizable fpga-based image registration accelerators." The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2021.

Exe 3 : Questions

A. Which percentage of FPGA processing will result in a speedup of 2?

B. Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis “Net speedup” and x-axis “Percent image processing”

Exe 3 : Questions

A. With what **percentage of processing** will adding FPGA result in a **speedup of 2**?

A. (At home, if you want, Enjoy :D) Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis “Net speedup” and x-axis “Percent image processing”

Exe 3.a : Find the processing fraction

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

Exe 3.a : Find the processing fraction

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

Exe 3.a : Find the processing fraction

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{2.86}}$$

Exe 3.a : Find the processing fraction

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{2.86}}$$

$$\frac{1}{2} = 1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{2.86}$$

$$\frac{1}{2} = \frac{2.86 - 2.86Fraction_{enhanced} + Fraction_{enhanced}}{2.86}$$

$$\frac{2.86}{2} = 2.86 - 2.86Fraction_{enhanced} + Fraction_{enhanced}$$

$$1.86Fraction_{enhanced} = 1.43$$

Exe 3.a : Find the processing fraction

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

$$2 = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{2.86}}$$

$$\frac{1}{2} = 1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{2.86}$$

$$\frac{1}{2} = \frac{2.86 - 2.86Fraction_{enhanced} + Fraction_{enhanced}}{2.86}$$

$$\frac{2.86}{2} = 2.86 - 2.86Fraction_{enhanced} + Fraction_{enhanced}$$

$$1.86Fraction_{enhanced} = 1.43$$

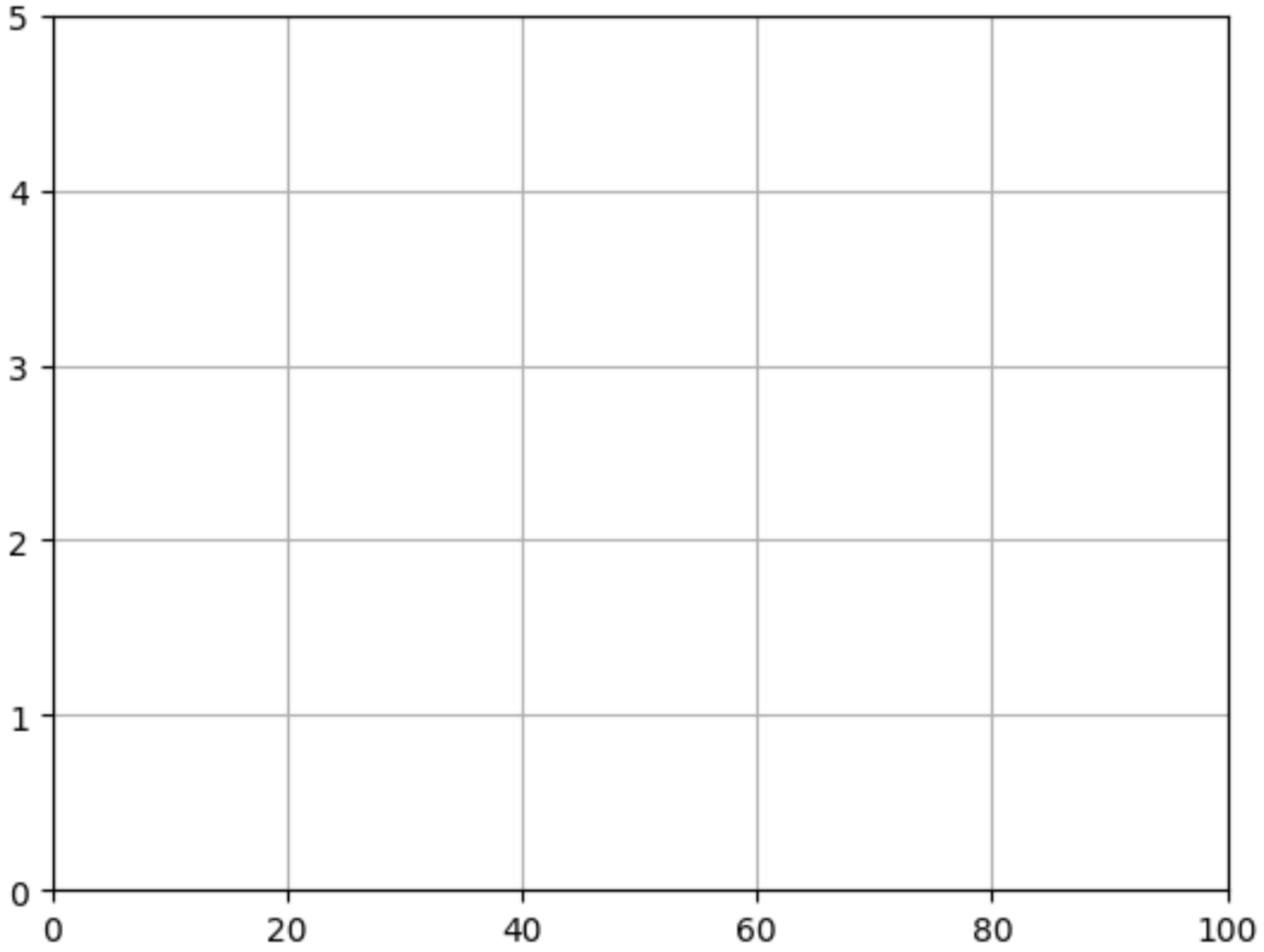
$$Fraction_{enhanced} = 0.768 = 76.8\%$$

Exe 3: Questions

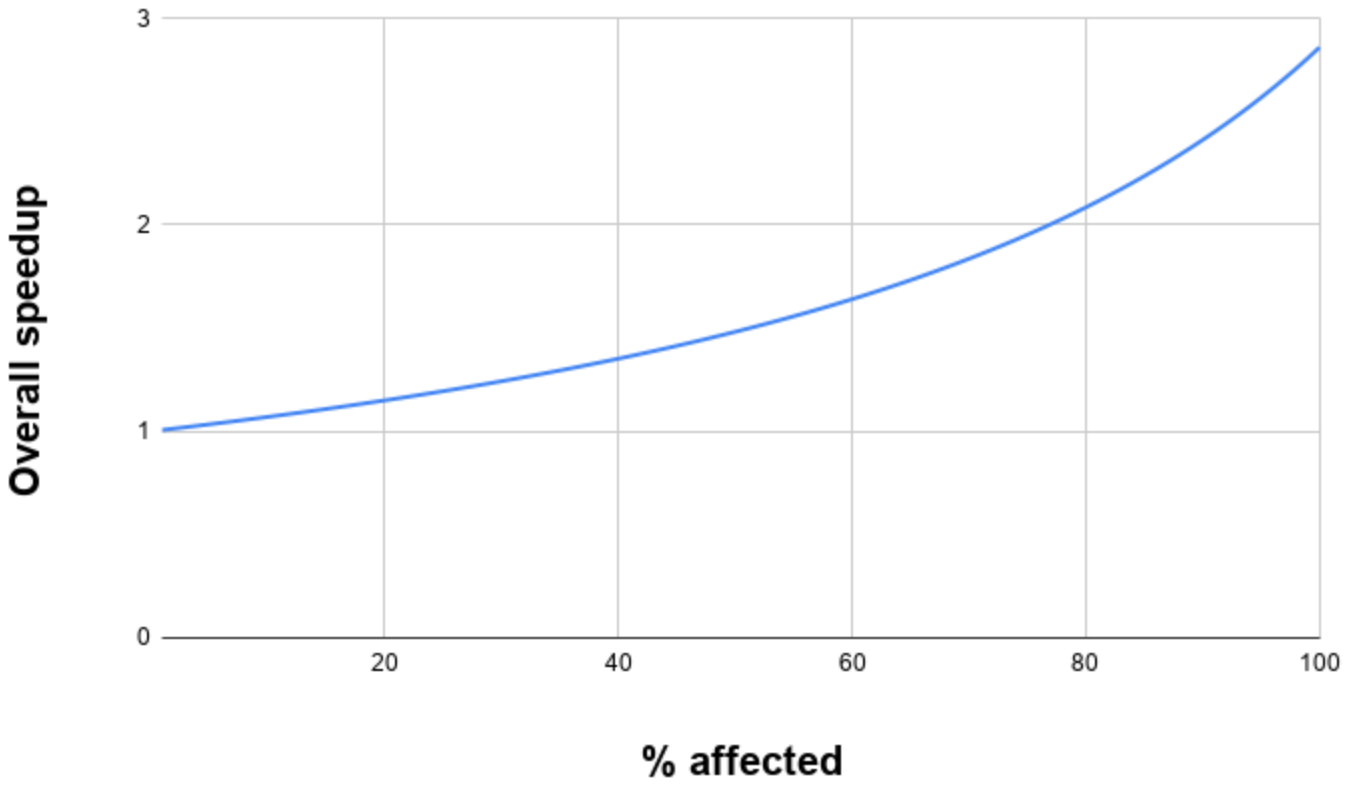
A. With what percentage of processing will adding FPGA result in a speedup of 2?

A. (At home, if you want, Enjoy :D) Draw a graph that plots the speedup as a percentage of the computation spent performing the image processing task. Label y-axis “Net speedup” and x-axis “Percent image processing”

Exe 3.b : Graph solution



Exe 3.b : Graph solution



100 / ((100-x)+x/2.86)

Performance Scaling

How does the **overall performance scale** if we further **increase** the **speedup**?

Let's consider an application where the **number** of used **processors/threads linearly increases** the **performance** of the parallelizable portion

Modern Problems Require Modern Formulae

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}})}$$

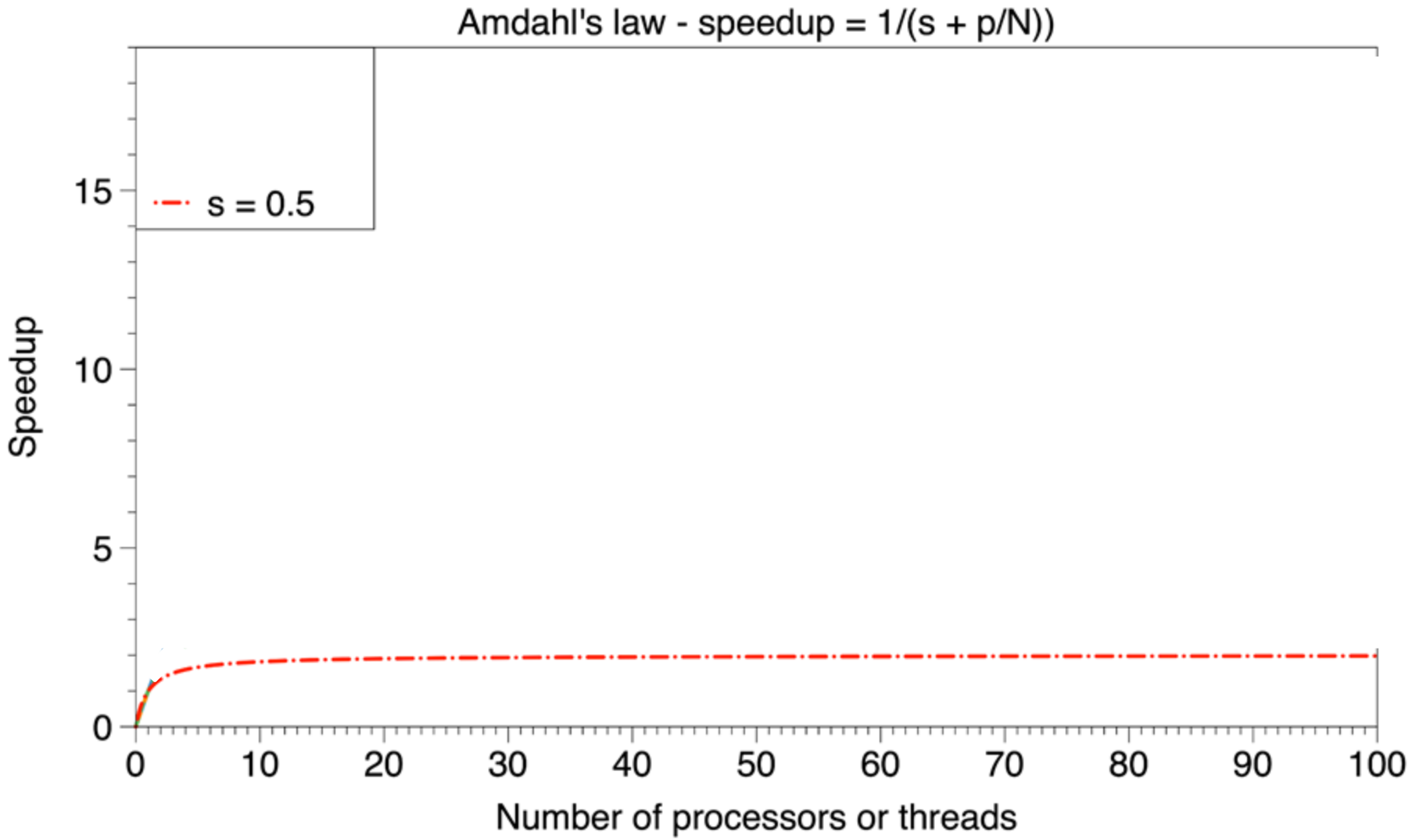
***speedup*$_{overall}$ = 1/(s + p/N)**

s = serial part = *1 - Fraction*$_{enhanced}$

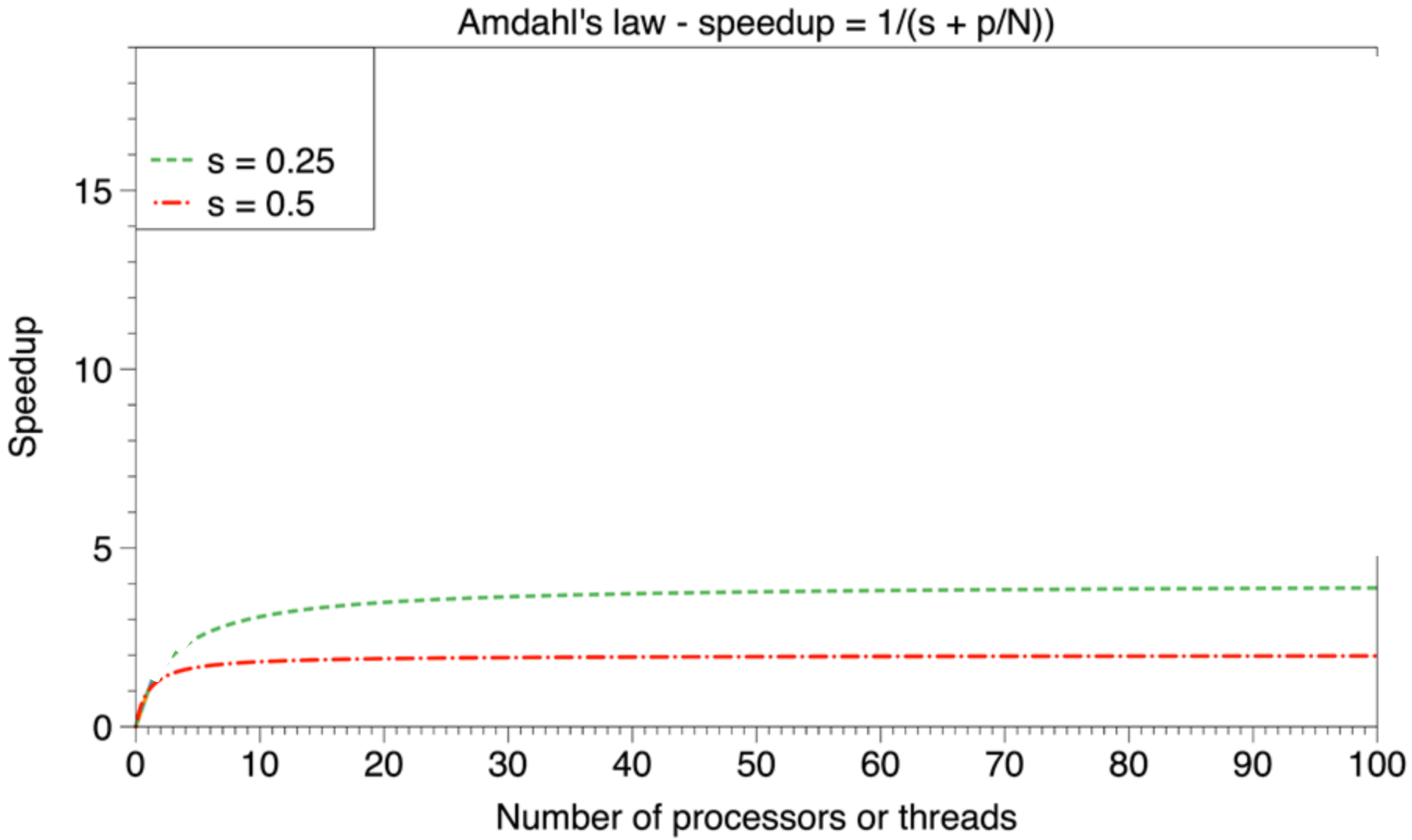
p = *1* - *s* = parallelizable part = *Fraction*$_{enhanced}$

N = number of processors or threads = *Speedup*$_{enhanced}$

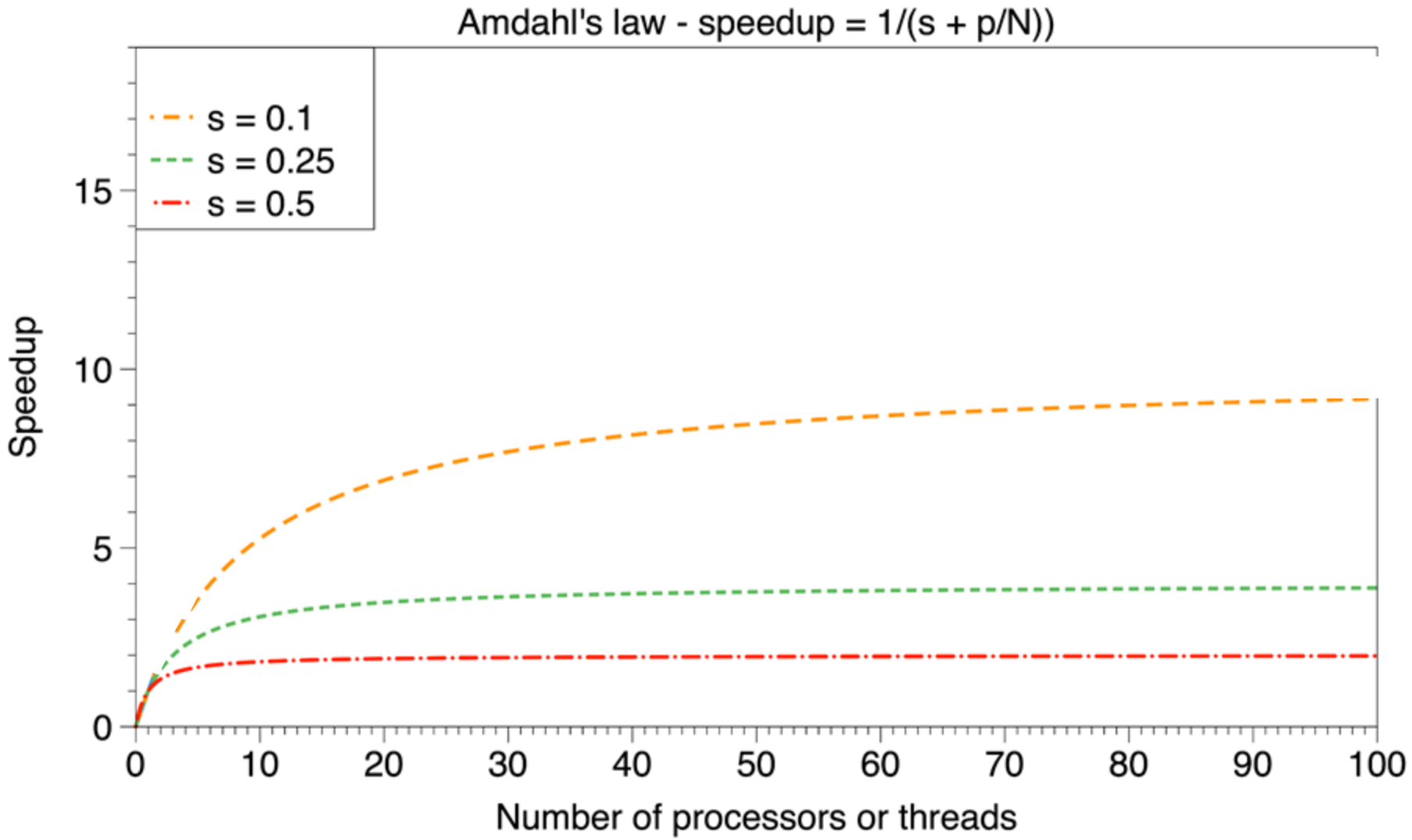
Speedup Scaling vs *N/s* Scaling



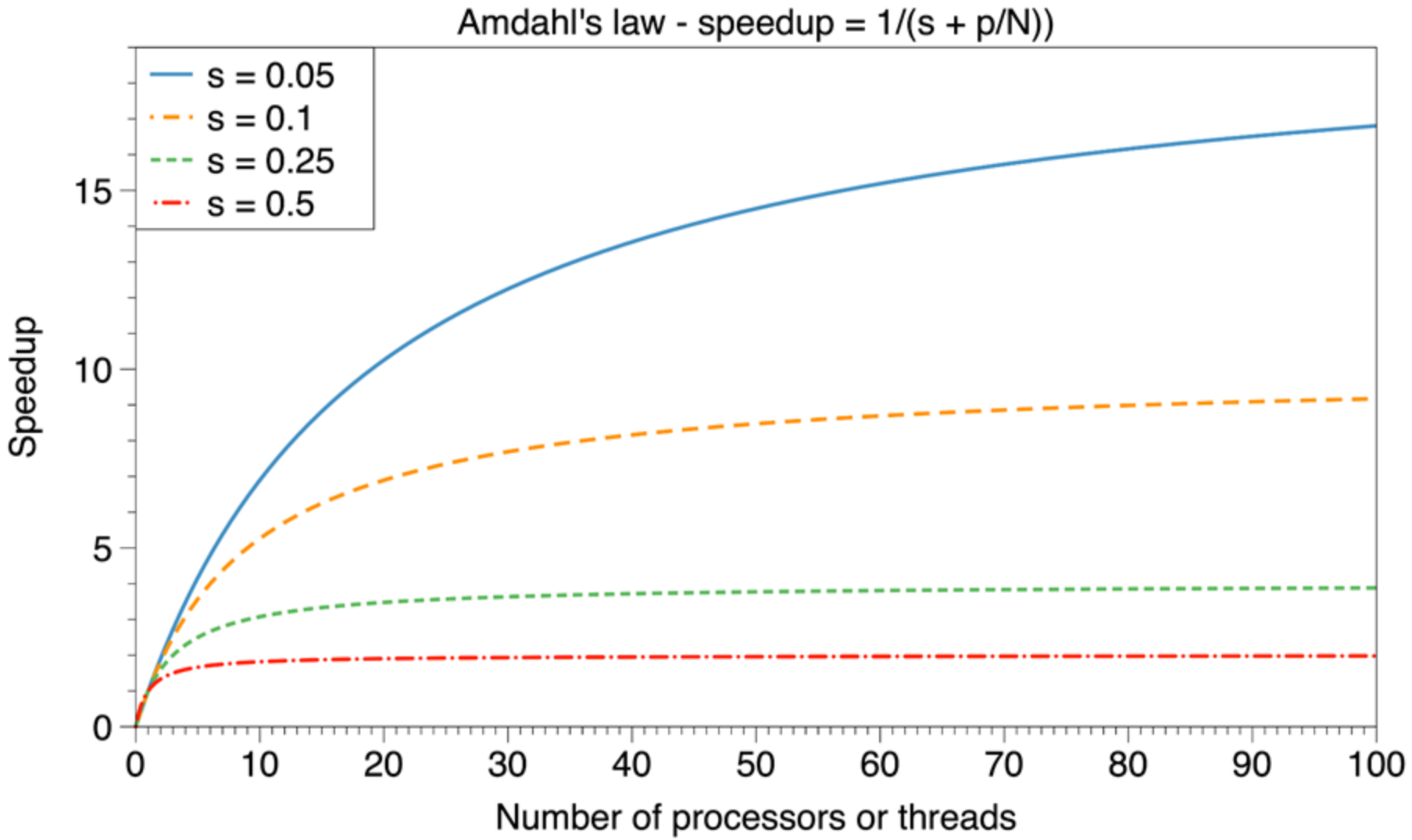
Speedup Scaling vs *N/s* Scaling



Speedup Scaling vs *N/s* Scaling



Speedup Scaling vs *N/s* Scaling



To Sum Up

Amdahl's law states that, for a **fixed** problem, the upper limit of speedup is determined by the serial fraction of the code -> **strong scaling**

$$speedup = 1/(s + p/N)$$

Something More about Performance Scaling

Gustafson's law -> weak scaling

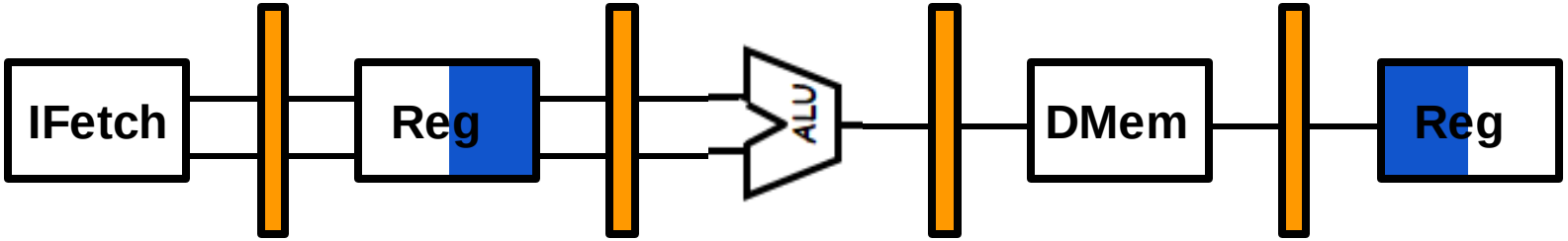
$$scaled\ speedup = s + p \times N$$

If you want more information **JUST FOR CURIOSITY**:

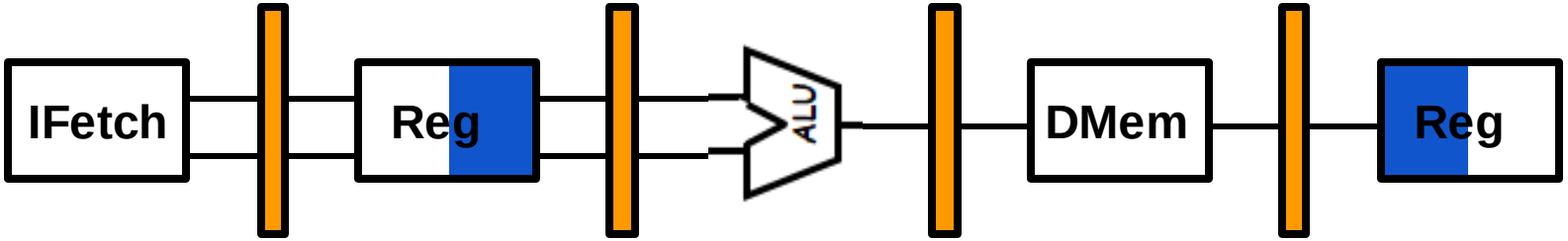
- https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/
- https://dl.acm.org/doi/10.1145/42411.42415

Exe 4 : Pipelining and Performance

Exe 4 : Pipelining and Performance

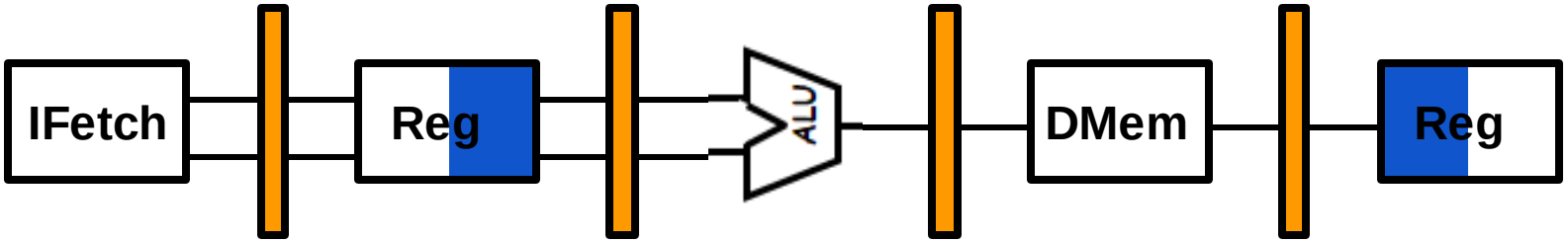


Exe 4 : Pipelining and Performance




```
lw      $1,  OFF($2)
addi    $3, $1, 4
sub     $4, $1, $2
addi    $2, $1, -8
sw      $4, OFF($2)
```

Exe 4 : Pipelining and Performance




```
lw      $1,  OFF($2)
addi    $3, $1, 4
sub     $4, $1, $2
addi    $2, $1, -8
sw      $4, OFF($2)
```

IF Instruction Fetch	ID Instruction Decode	EX Execution	ME Memory Access	WB Write Back

ALU Instructions: `op $x,$y,$z`

Instr. Fetch & PC Increm.	Read of Source Regs. `$y` and `$z`	ALU Op. (`$y op $z`)		Write Back Destinat. Reg. `$x`

Load Instructions: `lw $x,offset($y)`

Instr. Fetch & PC Increm.	Read of Base Reg. `$y`	ALU Op. (`$y+offset`)	Read Mem. `M($y+offset)`	Write Back Destinat. Reg. `$x`

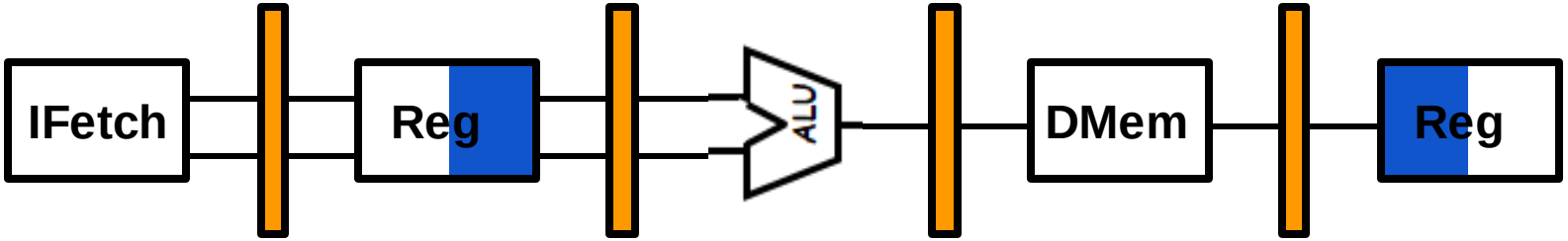
Store Instructions: `sw $x,offset($y)`

Instr. Fetch & PC Increm.	Read of Base Reg. `$y` & Source `$x`	ALU Op. (`$y+offset`)	Write Mem. `M($y+offset)`	

Conditional Branches: `beq $x,$y,offset`

Instr. Fetch & PC Increm.	Read of Source Regs. `$x` and `$y`	ALU Op. (`$x-$y`) & (`PC+4+offset`)	Write PC	

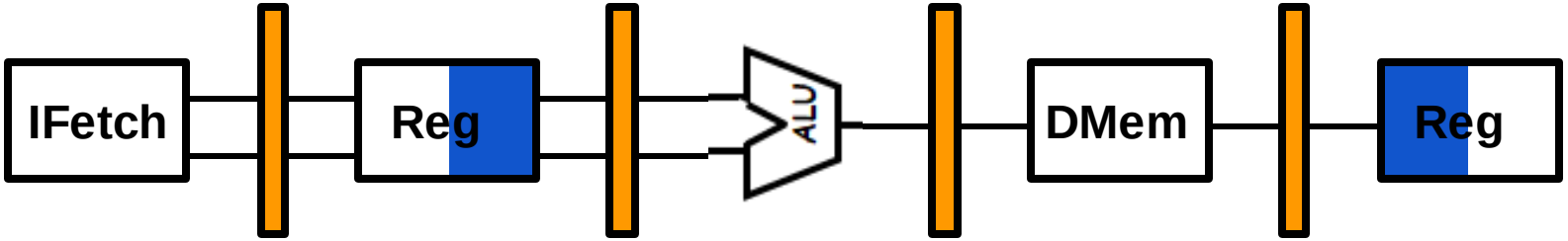
Exe 4 : Pipelining and Performance




```
lw      $1,  OFF($2)
addi    $3, $1, 4
sub     $4, $1, $2
addi    $2, $1, -8
sw      $4, OFF($2)
```

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our "optimization" (i.e., RF access R/W)

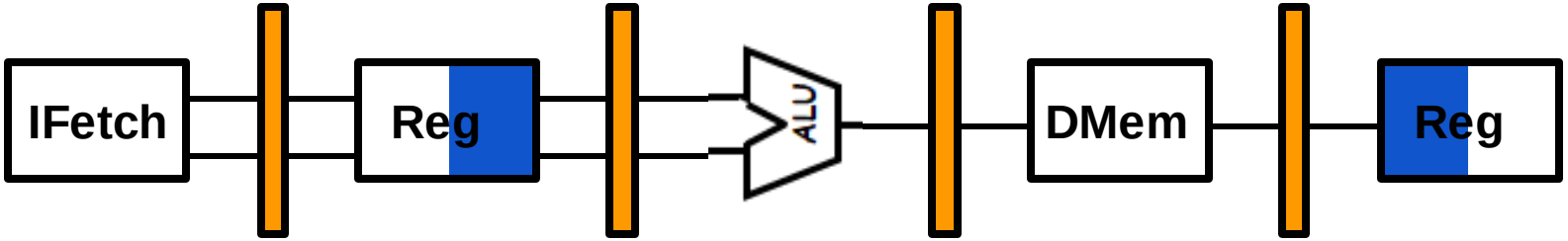
Exe 4 : Pipelining and Performance




```
lw      $1,  OFF($2)
addi    $3, $1, 4
sub     $4, $1, $2
addi    $2, $1, -8
sw      $4, OFF($2)
```

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our “optimization” (i.e., RF access R/W)
The processor has a clock cycle of 2ns

Exe 4 : Pipelining and Performance

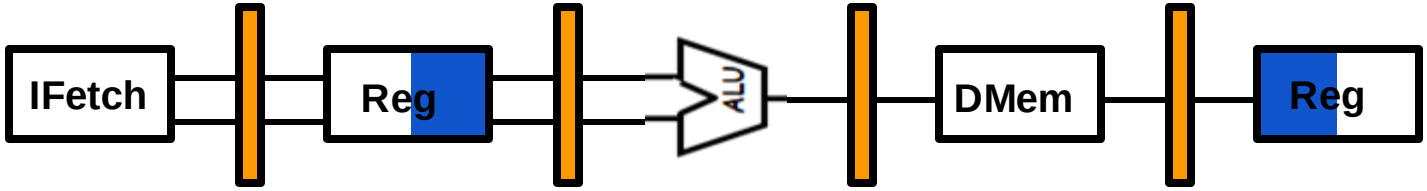



```
lw      $1,  OFF($2)
addi    $3, $1, 4
sub     $4, $1, $2
addi    $2, $1, -8
sw      $4, OFF($2)
```

No optimization in the **MIPS** pipeline (e.g., forwarding paths) just our "optimization" (i.e., RF access R/W)
The processor has a clock cycle of 2ns

A. Draw the pipeline schema and highlight possible hazards
B. Represent the real execution (Insert the stalls)
C. Calculate IC, CPI, MIPS
D. Do the same considering the existence of path forwarding

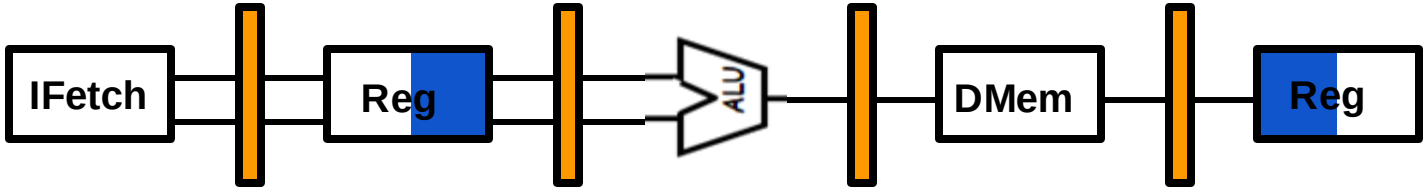
Exe 4.a : Ideal case



CC 0

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)															
addi $3, $1, 4															
sub $4, $1, $3															
addi $2, $1, -8															
sw $5, OFF($2)															

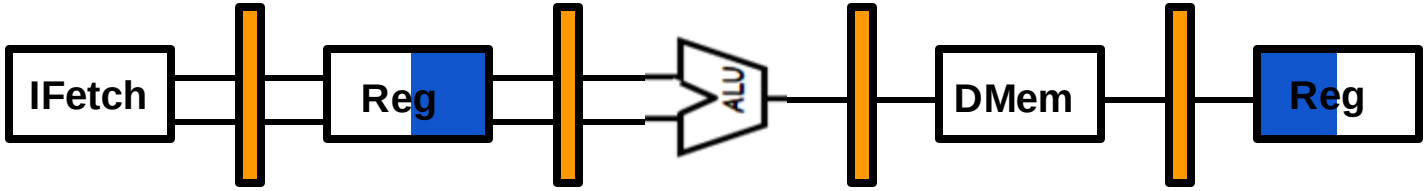
Exe 4.a : Ideal case



CC 2

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D													
addi $3, $1, 4		F													
sub $4, $1, $3															
addi $2, $1, -8															
sw $5, OFF($2)															

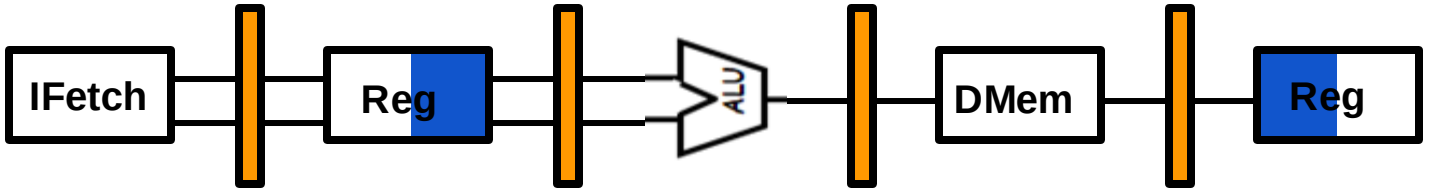
Exe 4.a : Ideal case



CC 3

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E												
addi $3, $1, 4		F	D												
sub $4, $1, $3			F												
addi $2, $1, -8															
sw $5, OFF($2)															

Exe 4.a : Ideal case



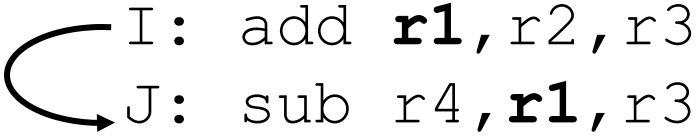
CC 9

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Recall: Type of Data Hazard

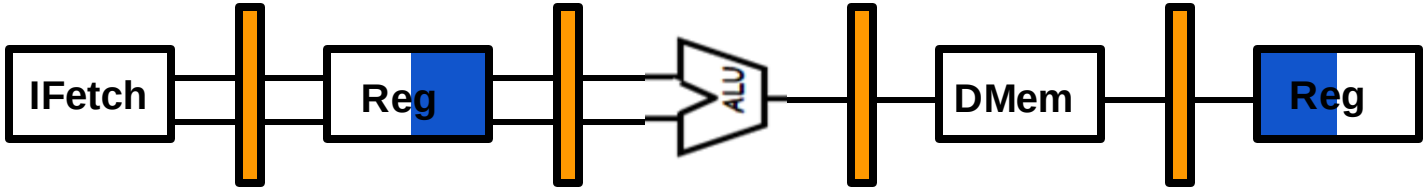
Read After Write (RAW)

Instr$_J$ tries to read operand before Instr$_I$ writes it



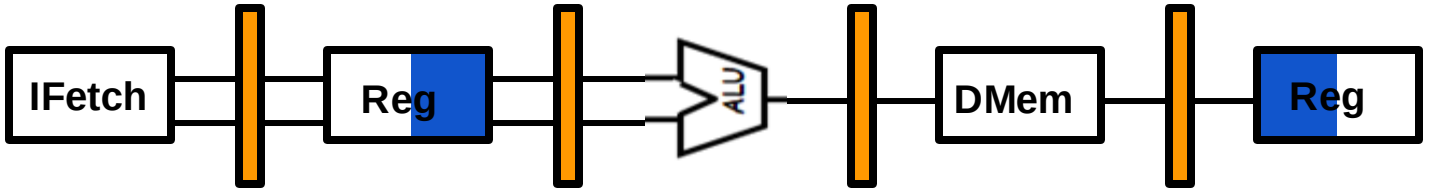
Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Exe 4.a : Ideal case



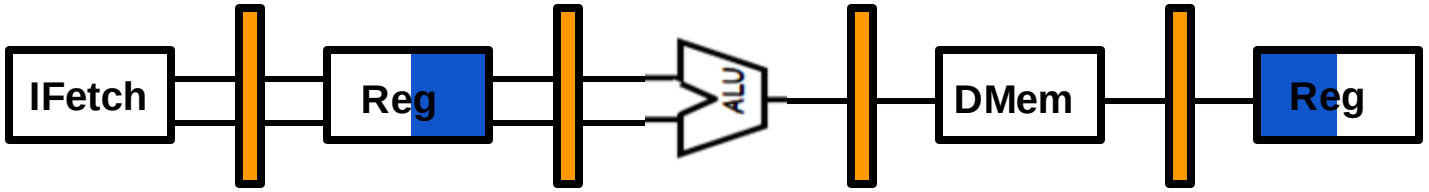
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Exe 4.a : The Hazard



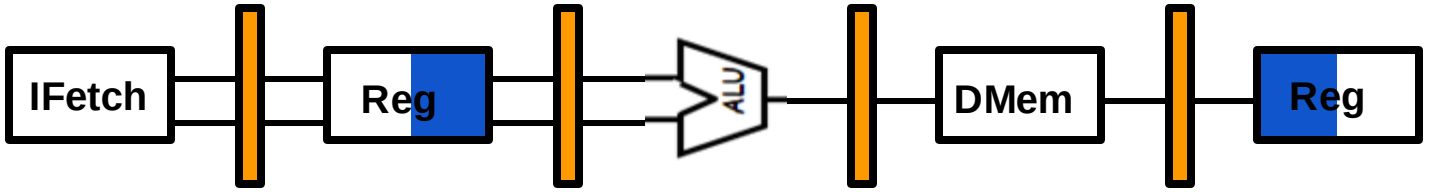
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Exe 4.a : The Hazard



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

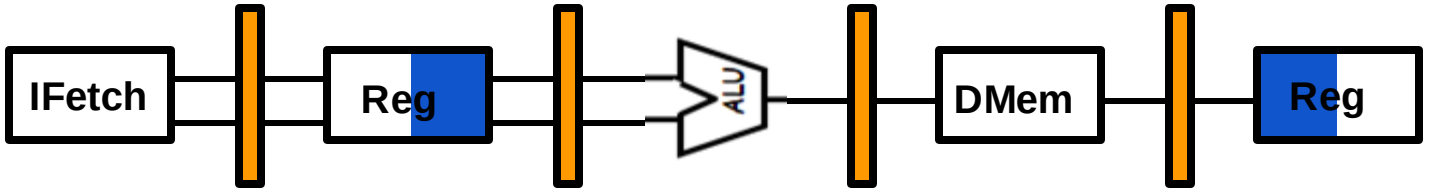
Exe 4.a : The Hazard



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

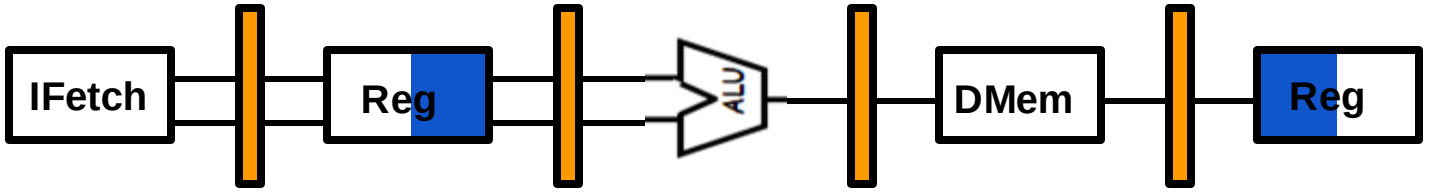
Not a real Hazard

Exe 4.a : The Hazard



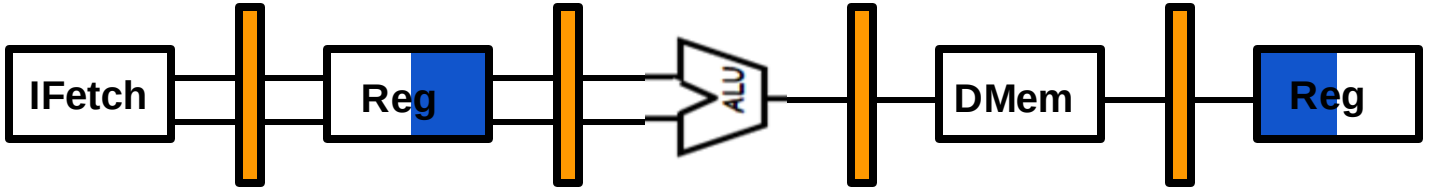
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Exe 4.a : The Hazard



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Exe 4.a : The Hazard

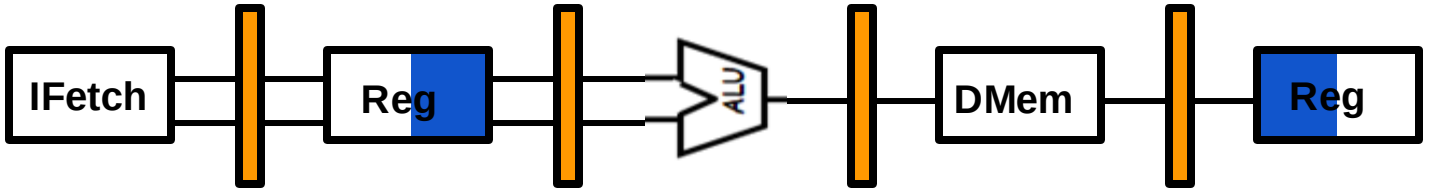


Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	D	E	M	W									
sub $4, $1, $3			F	D	E	M	W								
addi $2, $1, -8				F	D	E	M	W							
sw $5, OFF($2)					F	D	E	M	W						

Recall: Data Hazards: Possible Solutions

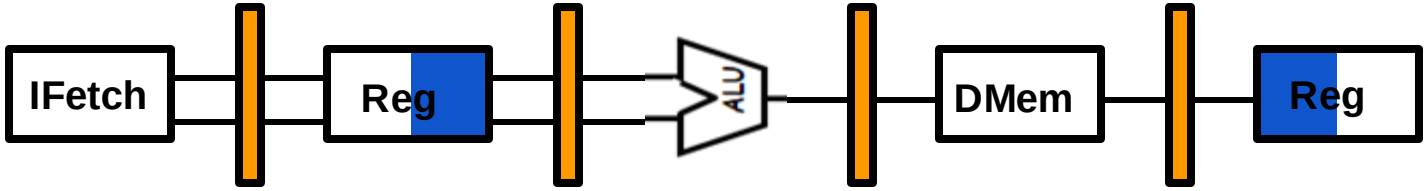
- Compilation Techniques:
  - Insertion of nop (no operation) instructions
  - Instructions Scheduling to avoid that correlating instructions are too close
    - The compiler tries to insert independent instructions among correlating instructions
    - When the compiler does not find independent instructions, it insert nops.
- Hardware Techniques:
  - Insertion of “bubbles” or stalls in the pipeline
  - Data Forwarding or Bypassing

Exe 4.b : Bubble insertion



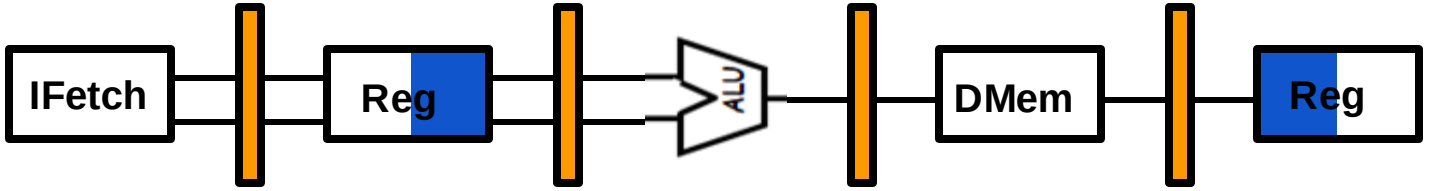
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	D	E	M	W						
addi $2, $1, -8						F	D	E	M	W					
sw $5, OFF($2)							F	D	E	M	W				

Exe 4.b : Bubble insertion



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	D	E	M	W		

Exe 4.b : Bubble insertion



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

$$IC = 5$$

Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

$$IC = 5$$

$$CPI = \frac{CCs}{IC}$$

Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

$$IC = 5$$

$$CPI = \frac{CCs}{IC} = \frac{15}{5}$$

Recall: MIPS - Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

$$IC = 5$$

$$CPI = \frac{CCs}{IC} = \frac{15}{5}$$

$$MIPS = \frac{ClockFrequency}{CPI * 10^6}$$

Exe 4.c : Performance

Calculate **IC, CPI, MIPS**

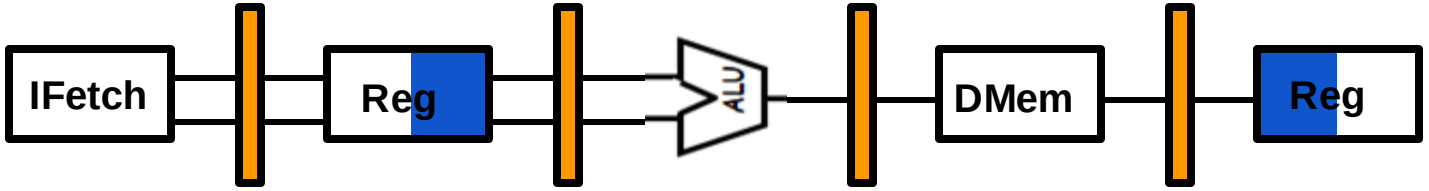
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	Stall	D	E	M	W							
sub $4, $1, $3					F	Stall	Stall	D	E	M	W				
addi $2, $1, -8								F	D	E	M	W			
sw $5, OFF($2)									F	Stall	Stall	D	E	M	W

$$IC = 5$$

$$CPI = \frac{CCs}{IC} = \frac{15}{5}$$

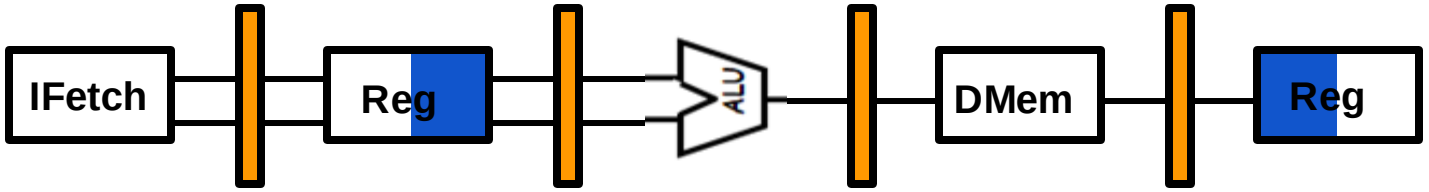
$$MIPS = \frac{ClockFrequency}{CPI * 10^6} = \frac{\frac{1}{2} * 10^9}{3 * 10^6} = 166$$

Exe 4.d : Path Forwarding



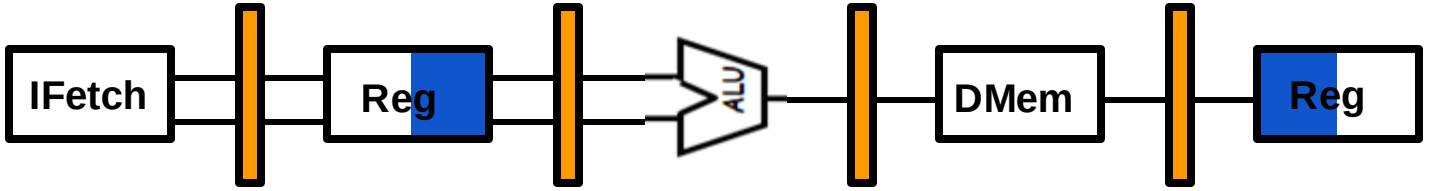
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4															
sub $4, $1, $3															
addi $2, $1, -8															
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



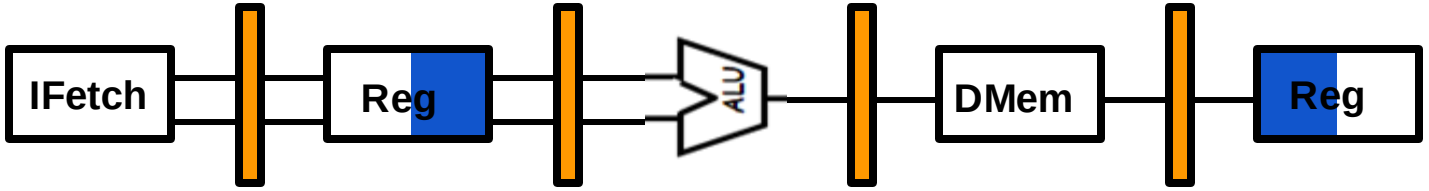
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3															
addi $2, $1, -8															
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



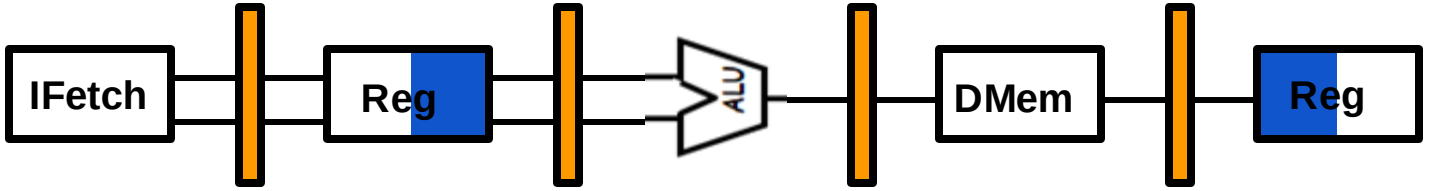
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3															
addi $2, $1, -8															
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



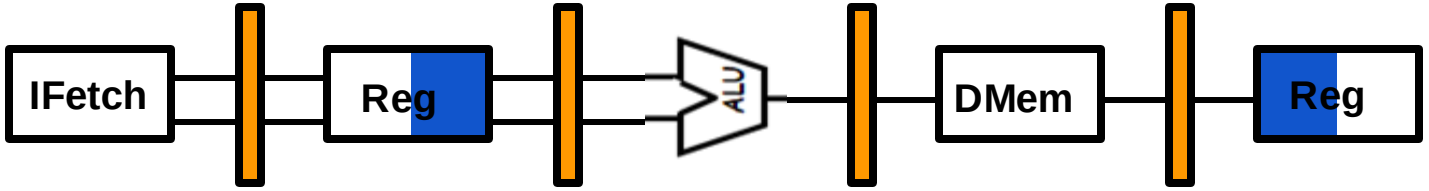
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3				F	D	E	M	W							
addi $2, $1, -8															
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



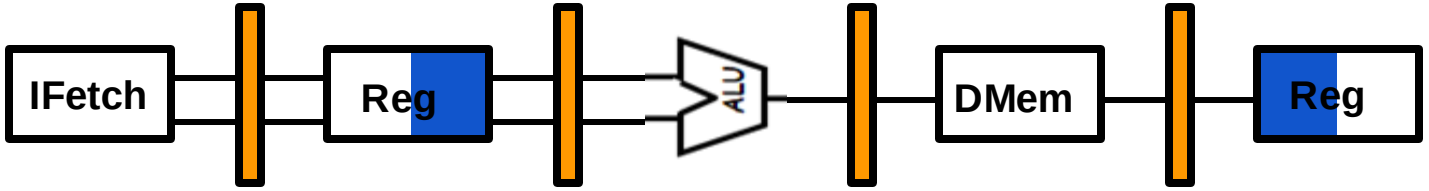
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3				F	D	E	M	W							
addi $2, $1, -8					F	D	E	M	W						
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3				F	D	E	M	W							
addi $2, $1, -8					F	D	E	M	W						
sw $5, OFF($2)															

Exe 4.d : Path Forwarding



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw $1, OFF($2)	F	D	E	M	W										
addi $3, $1, 4		F	Stall	D	E	M	W								
sub $4, $1, $3				F	D	E	M	W							
addi $2, $1, -8					F	D	E	M	W						
sw $5, OFF($2)						F	D	E	M	W					

Thank you for your attention
Questions?

Alessandro Verosimile <alessandro.verosimile@polimi.it>

Acknowledgements

Davide Conficconi, E. Del Sozzo, Marco D. Santambrogio, D. Sciuto

Part of this material comes from:

- Hennessy and Patterson Turing Lecture
- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- “Digital Design and Computer Architecture” Harris and Harris