

I. Validate the correctness of program with and without naive mode

1. **Without** naive pipeline mode, the program runs normally as in Labwork 1:

The screenshot shows the Pipeline processor simulator interface. The configuration panel on the left has 'Naive pipeline' set to false. The simulation panel shows the PC at 26 and cycle number 318. The memory view shows instructions at addresses 0 to 10. The pipeline view shows the instruction 'SUB R14, R14, 1' in the WB stage. The register files show R0 to R15 and F0 to F15.

Pipeline processor simulator
Copyright : University of Rennes 1 - Enssat - R202 — http://r2d2.enssat.fr
Version 0.1 Beta — 24 janvier 2008

Configuration
Load file: Ex.asm
Reset processor: Reset
Nb of exec stages: 1
Size of register files: 32
Naive pipeline: ☐

Simulation
PC: 26
Cycle num: 318
RUN
Execute the program from PC until pipeline empty
Execute: 0 cycles
Lost cycles: 101
Debug: ☐
Memory trace: ☐ data ☐ instructions

Memory views
Address Label Instruction LI EX
0 LI R10, ARR 1 1
1 LI R14, LEN 1 1
2 LOOP1 LI R0, 0 6 5
3 SUB R14, R14, 1 6 5
4 LI R15, 0 5 5
5 BRZ R14, EXIT 5 5
6 LOOP2 SUB R13, R15, R14 20 20
7 BRZ R13, CHECK_SWAP 20 20
8 ADD R11, R10, R15 20 15
9 LI R1, (R11) 20 15
10 ADD R12, R11, 1 20 15

Pipeline view
Stages Instructions
LI EXIT
DI EXIT
READOP EXIT
EX0 EXIT
WRITE... EXIT
WB NOP (SUB R14, R14...)

Register files
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
0 1 2 -1 0 0 0 0 0 0 32 32 33 0 1 1
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

2. **With** naive pipeline mode, the program is not running correctly:

The screenshot shows the Pipeline processor simulator interface with 'Naive pipeline' set to true. The simulation panel shows the PC at 4 and cycle number 259. The memory view shows instructions at addresses 0 to 10. The pipeline view shows the instruction 'SUB R14, R14, 1' in the LI stage. The register files show R0 to R15 and F0 to F15.

Pipeline processor simulator
Copyright : University of Rennes 1 - Enssat - R202 — http://r2d2.enssat.fr
Version 0.1 Beta — 24 janvier 2008

Configuration
Load file: Ex.asm
Reset processor: Reset
Nb of exec stages: 1
Size of register files: 32
Naive pipeline: ☒

Simulation
PC: 4
Cycle num: 259
RUN
Next cycle
Continue
Execute: 0 cycles
Lost cycles: 0
Debug: ☐
Memory trace: ☐ data ☐ instructions

Memory views
Address Label Instruction LI EX
0 LI R10, ARR 1 1
1 LI R14, LEN 1 1
2 LOOP1 LI R0, 0 7 6
3 SUB R14, R14, 1 7 6
4 LI R15, 0 6 6
5 BRZ R14, EXIT 6 6
6 LOOP2 SUB R13, R15, R14 21 21
7 BRZ R13, CHECK_SWAP 21 21
8 ADD R11, R10, R15 21 21
9 LI R1, (R11) 20 20
10 ADD R12, R11, 1 20 20

Pipeline view
Stages Instructions
LI SUB R14, R14, 1
DI LI R0, 0
READOP BR LOOP1
EX0 BRZ R0, EXIT
WRITE... EXIT
WB EXIT

Register files
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
0 5 2 3 0 0 0 0 0 0 32 32 34 0 0 0
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

3. Explanation:

- When executing in normal pipeline mode, results become more accurate because each instruction is executed in distinct stages of the pipeline. This ensures that the outcome of each instruction is correctly processed before continuing to the next stage. The value from the previous instruction must reach the Write Back (WB) stage and be updated before the next dependent instruction is executed. For example, “add R13, R14, R15”, the result of $R12 = R14 + R15$ will be saved first and then move to the next instruction such as `li R0, R13`.
- The reason why naive pipeline mode will run incorrectly is because in naive pipeline mode, instructions dependent on the results of previous instructions may not wait for those results. For example, if R14 and R15 are loaded from memory and immediately compared or swapped/add/subtract, the compare or swap might execute before the load completes or before moving to the next instruction related to these parameters, leading to the fact that the result will be failed and making the execution incorrect.

II. Analyze the execution of bubble algorithm in pipeline:

1. **Note if the execution is correct or not:** The execution without naive pipeline mode is correct.

The screenshot displays the 'Pipeline processor simulator' interface. The top section includes the title 'Pipeline processor simulator' and copyright information: 'Copyright : University of Rennes 1 - Enssat - R2D2 - http://r2d2.enssat.fr Version 0.1 Beta - 24 janvier 2008'.

Configuration:

- Load file: `Ex.asm` (with a 'Browse...' button)
- Reset processor: `Reset` button
- Nb of exec stages: `1` (dropdown)
- Size of register files: `32` (dropdown)
- Naive pipeline: ☐

Simulation:

- PC: `26`
- Cycle num: `318` (with a 'Save...' button)
- `RUN` button
- `Execute the program from PC until pipeline empty` button
- Execute: `0` cycles
- Lost cycles: `101`
- Debug: ☐
- Memory trace: ☐ data ☐ instructions

Memory views:

Address	Label	Instruction	LI	EX
0		LI R10, ARR	1	1
1		LI R14, LEN	1	1
2	LOOP1	LI R0, 0	6	5
3		SUB R14, R14, 1	6	5
5		LI R15, 0	5	5
5		BRZ R14, EXIT	5	5
6	LOOP2	SUB R13, R15, R14	20	20
7		BRZ R13, CHECK_SWAP	20	20
8		ADD R11, R10, R15	20	15
9		LI R1, (R11)	20	15
10		ADD R12, R11, 1	20	15

Pipeline view:

Stages	Instructions
LI	EXIT
DI	EXIT
READOP	EXIT
EX0	EXIT
WRITE...	EXIT
WB	NOP (SUB R14, R14...

Register files:

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
0	1	2	-1	0	0	0	0	0	0	32	32	33	0	1	1

Below the registers, there are floating-point registers F0 to F15, all showing `0.0`.

2. Capture case when two instructions are dependents and understand what happens

R13 = R15 - R14, and loop until R13 = 0, continue at CHECK Instruction.

R11 = R10 - R15, after that the value of R11 will be load to R1

R12 = R11 - R1, after that the value of R12 will be load to R2

3. Capture case when a branch is executed and understand what happens

When a branch is executed, the instructions inside the branch will be executed in the new flow, and the instructions after the branch will not be executed until the branch has been resolved.

```
brz R14, EXIT    -- if length is zero, exit
```

[illegible]

In this case, firstly, I already have instructions to calculate the length of the initialize array to implement the loop, then, I set this value to R14 and decrease by 1 when one loop is finished. After this value of R14 become 0, the loop will end and exit the program to avoid the infinite loop

Another example is:

```
brn R3, FOR      -- if current element <= next element, continue loop
```

Pipeline processor simulator
Copyright : University of Rennes 1 - Enssat - R2D2 — <http://r2d2.enssat.fr>
Version 0.1 Beta — 24 janvier 2008

Configuration

Load file:

Reset processor:

Nb of exec stages:

Size of register files:

Naive pipeline: ☐

Simulation

PC:

Cycle num:

Execute:

Lost cycles:

Debug: ☐

Memory trace: ☐ data ☐ instructions

Memory views

Address	Label	Instruction	LI	EX
0		LI R10, ARR	1	1
1		LI R14, LEN	1	1
2	WHILE	LI R0, 0	1	1
3		SUB R14, R14, 1	1	1
4		LI R15, 0	1	1
5		BRZ R14, EXIT	1	1
6	FOR	SUB R13, R15, R14	3	2
7		BRZ R13, CHECK	3	2
8		ADD R11, R10, R15	2	2
9		LI R1, (R11)	2	2
10		ADD R12, R11, 1	2	2

Pipeline view

Stages	Instructions
LI	BRZ R13, CHECK
DI	SUB R13, R15, R14
READOP	NOP (LI R0, 1)
EX0	NOP (SI (R12), R1)
WRITE...	NOP (SI (R11), R2)
WB	BRN R3, FOR

Register files

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
0	7	8	-1	0	0	0	0	0	0	32	33	34	-4	5	2

Terminal

```
Processor 0> Load file : OK
Processor 0> Reset processor (empty pipeline) : OK
Processor 0>
```

In this case, the value of R3 is R1 - R2 which R1 is the value of one element in the array and R2 is the next element of R1 in the array. If R3 is negative (brn R3), that means that $R1 < R2 \rightarrow R1$ and R2 will not swap due to the bubble sort algorithm.

4. Note the number of cycles which are lost during the execution with different number of pipeline stages

Nb of exec stages:

Size of register files:

Naive pipeline: ☐

Next cycle:

Execute:

Lost cycles:

Debug: ☐

Nb of exec stages	2	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	204	
Nb of exec stages	3	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	296	
Nb of exec stages	4	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	388	
Nb of exec stages	5	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	347	
Nb of exec stages	6	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	418	Execute on
Nb of exec stages	7	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	489	
Nb of exec stages	8	▼	Execute	0	Next cycle
Size of register files	32	▼	Lost cycles	560	

Nb of exec stages	9	Next cycle
Size of register files	32	Execute 0
		Lost cycles 631
		Debug

Nb of exec stages	10	Next cycle
Size of register files	32	Execute 0
		Lost cycles 702
		Debug

→ **Explanation:** When the instruction is executed in different stages, there are different numbers of instructions executed at the same time, so that there will be different amounts of cycles lost during the execution normally, the bigger the number of stages, the bigger the number of lost cycles. When the number of execution stages is increased, each instruction must pass through more stages before it is completed, which might lead to a higher number of cycles in the pipeline (in some cases, this number of cycles might reduce rather than increase). Even though the total number of cycles and potential lost cycles may rise, the crucial point is that the final output remains correct. Adding more execution stages boosts pipeline performance by enabling concurrent processing of multiple instructions and increasing parallelism.