UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Information and Communication Technology Department

- REPORT MIDTERM

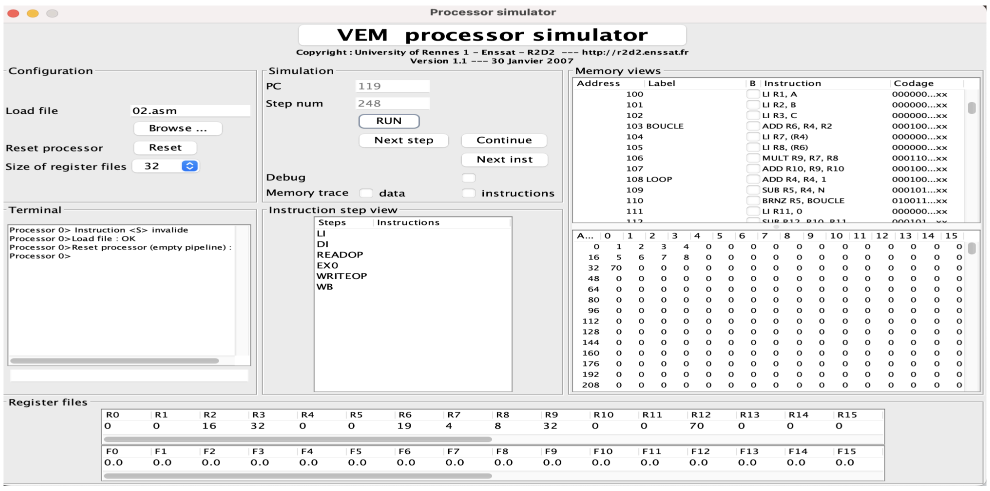
Name : Phạm Phú Hưng
StudentID : BA12-81
Major : Cyber Security

Contents

I. TASK 1

A. CASE 1

- Data 0
  Input x: 1, 2, 3, 4
- Data 16
  Input w: 5, 6, 7, 8



Processor simulator

VEM processor simulator

Copyright : University of Rennes 1 – Enssat – R2D2 --- http://r2d2.enssat.fr
Version 1.1 --- 30 Janvier 2007

Configuration

Load file	02.asm
Reset processor	Reset
Size of register files	32

Simulation

PC	119
Step num	248

Terminal

```
Processor 0> Instruction <S> invalide
Processor 0>Load file : OK
Processor 0>Reset processor (empty pipeline) :
Processor 0>
```

Instruction step view

Steps	Instructions
LI	
DI	
READOP	
EX0	
WRITEOP	
WB	

Memory views

Address	Label	B	Instruction	Codage
100			LI R1, A	000000...xx
101			LI R2, B	000000...xx
102			LI R3, C	000000...xx
103	BOUCLE		ADD R6, R4, R2	000100...xx
104			LI R7, (R4)	000000...xx
105			LI R8, (R6)	000000...xx
106			MULT R9, R7, R8	000110...xx
107			ADD R10, R9, R10	000100...xx
108	LOOP		ADD R4, R4, 1	000100...xx
109			SUB R5, R4, N	000101...xx
110			BRNZ R5, BOUCLE	010011...xx
111			LI R11, 0	000000...xx

A...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	0	0	0	0	0	0	0	0	0	0	0	0
16	5	6	7	8	0	0	0	0	0	0	0	0	0	0	0	0
32	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register files

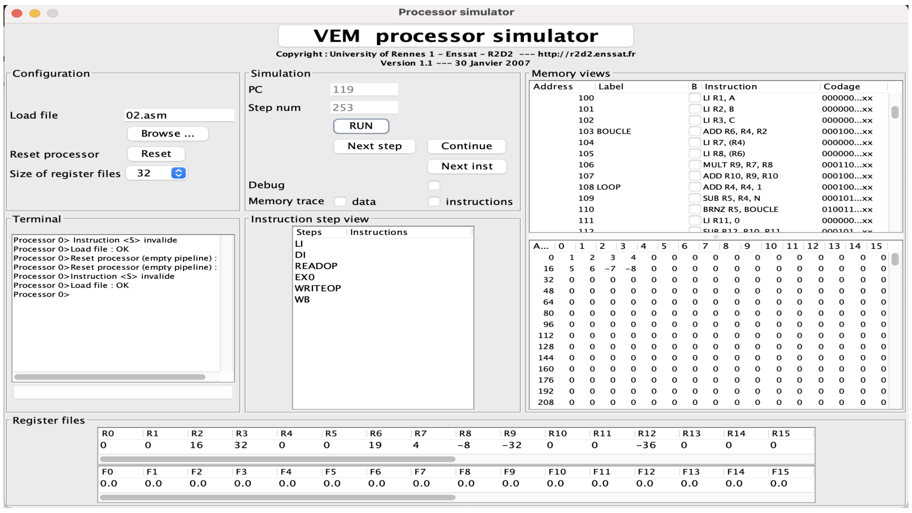
R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
0	0	16	32	0	0	19	4	8	32	0	0	70	0	0	0

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

- Data 32
  Output 70

B. CASE 2

- Data 0
  Input x: 1, 2, 3, 4
- Data 16
  Input w: 5, 6, -7, -8



Processor simulator

VEM processor simulator

Copyright : University of Rennes 1 – Enssat – R2D2 --- http://r2d2.enssat.fr
Version 1.1 --- 30 Janvier 2007

Configuration

Load file	02.asm
Reset processor	Reset
Size of register files	32

Simulation

PC	119
Step num	253

Terminal

```
Processor 0> Instruction <S> invalide
Processor 0>Load file : OK
Processor 0>Reset processor (empty pipeline) :
Processor 0>Reset processor (empty pipeline) :
Processor 0>Instruction <S> invalide
Processor 0>Load file : OK
Processor 0>
```

Instruction step view

Steps	Instructions
LI	
DI	
READOP	
EX0	
WRITEOP	
WB	

Memory views

Address	Label	B	Instruction	Codage
100			LI R1, A	000000...xx
101			LI R2, B	000000...xx
102			LI R3, C	000000...xx
103	BOUCLE		ADD R6, R4, R2	000100...xx
104			LI R7, (R4)	000000...xx
105			LI R8, (R6)	000000...xx
106			MULT R9, R7, R8	000110...xx
107			ADD R10, R9, R10	000100...xx
108	LOOP		ADD R4, R4, 1	000100...xx
109			SUB R5, R4, N	000101...xx
110			BRNZ R5, BOUCLE	010011...xx
111			LI R11, 0	000000...xx

A...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	0	0	0	0	0	0	0	0	0	0	0	0
16	5	6	−7	−8	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register files

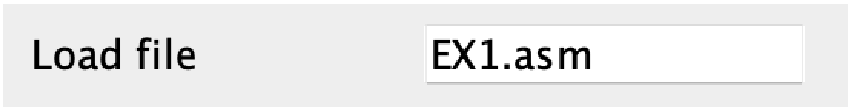
R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
0	0	16	32	0	0	19	4	−8	−32	0	0	−36	0	0	0

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

- Data 32
  Output 0

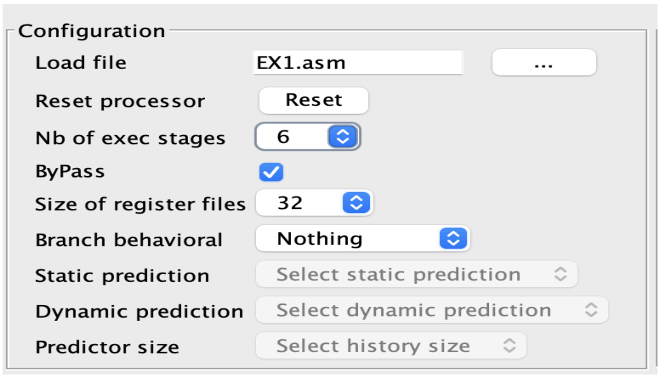
II. TASK 2

A. Load the code to JSimRiscz



Load file: EX1.asm

B. Configure the processor with 6 pipeline stages



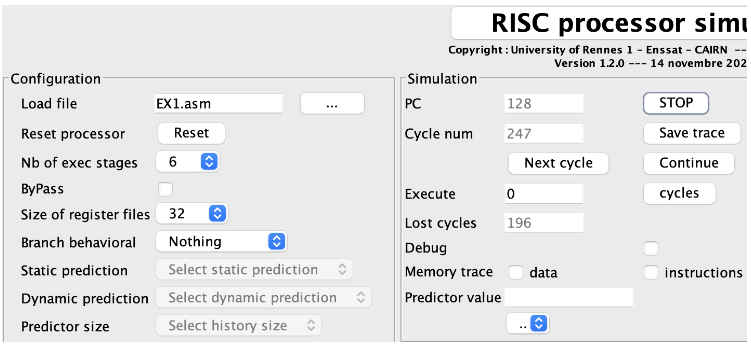
Configuration

Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	6
ByPass	✓
Size of register files	32
Branch behavioral	Nothing
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

C. Analysis of Cycle Savings with Bypass

1. **Without Bypass:**

- **Cycle Number**: 247
- **Lost cycles:** 196



RISC processor simu

Copyright : University of Rennes 1 – Enssat – CAIRN --
Version 1.2.0 --- 14 novembre 202

Configuration

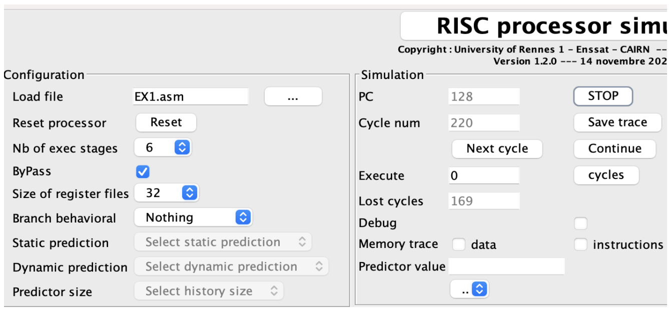
Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	6
ByPass	
Size of register files	32
Branch behavioral	Nothing
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

Simulation

PC	128
Cycle num	247
Execute	0
Lost cycles	196
Debug	
Memory trace	data, instructions
Predictor value	

2. **With Bypass:**

- **Cycle Number :** 220
- **Lost Cycles :** 169


RISC processor simu

Copyright : University of Rennes 1 – Enssat – CAIRN --
Version 1.2.0 --- 14 novembre 202

Configuration

Load file: EX1.asm
Nb of exec stages: 6
ByPass: ✓
Size of register files: 32
Branch behavioral: Nothing
Static prediction: Select static prediction
Dynamic prediction: Select dynamic prediction
Predictor size: Select history size

Simulation

PC: 128
Cycle num: 220
Execute: 0
Lost cycles: 169
Debug: ☐
Memory trace: ☐ data ☐ instructions
Predictor value:

3. **Calculation of saved cycles:**

- **Cycles Saved = Cycle Number without Bypass − Cycle Number with Bypass**
- **Cycles Saved**=247−220=27

- **Lost Cycles Saved = Lost Cycles without Bypass − Lost Cycles with Bypass**
- **Lost Cycles Saved**=196−169=27

D. Analyze where are the main problem in the code

To analyze the main problems in the code, especially focusing on the issues related to the RISC pipeline, we'll look at common sources of inefficiency and potential hazards. These include data hazards, control hazards

Data Hazards

Data hazards occur when instructions that exhibit data dependency are close together in the pipeline. There are three types of data hazards: RAW (Read After Write), WAR (Write After Read), and WAW (Write After Write).

1. RAW Hazard:

- Example:

  ```

  add R6, R4, R2

  li R7, (R4)

  li R8, (R6)

  mult R9, R7, R8

  add R10, R9, R10
  ```

\`\`\`

- Explanation: Here, `li R8, (R6)` depends on the result of `add R6, R4, R2`, and `mult R9, R7, R8` depends on `li R8, (R6)`. If these instructions are too close, the pipeline might stall because it has to wait for the data to be available.

2. WAW Hazard:

   - Example:

```assembly
li R10, 0

sub R12, R10, R11
```

   - Explanation:

     If there are multiple writes to the same register in close succession, the pipeline must ensure the correct order of writes is maintained.

Control Hazards

Control hazards occur due to branching instructions. When a branch is taken, the pipeline might have already fetched several instructions that need to be flushed if the branch changes the flow.

1. Branch Instructions:

   - Example:

```
brnz R5, boucle

brp R12, SKIPRELU
```

   - Explanation:

The pipeline must determine if the branch is taken, which can cause delays. Modern processors use branch prediction techniques to mitigate this, but mispredictions can still cause stalls.

E. Data dependencies

1. **Instruction: add R6, R4, R2**
   - **Dependencies**: R4 and R2 must be available for R6 to be computed.
   - **Potential Hazard**: If a previous instruction modifies R4 or R2, this instruction must wait until those modifications are complete.
2. **Instruction: li R7, (R4)**
   - **Dependencies**: R4 must be available to load memory content into R7.
   - **Potential Hazard**: If R4 is modified by a previous instruction, this load instruction must wait until R4 is updated.
3. **Instruction: li R8, (R6)**
   - **Dependencies**: R6 must be available to load memory content into R8.
   - **Potential Hazard**: Since R6 is calculated in the previous instruction (add R6, R4, R2), this instruction depends on the completion of the add operation.
4. **Instruction: mult R9, R7, R8**
   - **Dependencies**: R7 and R8 must be available to compute R9.
   - **Potential Hazard**: This instruction depends on the completion of the previous two load instructions (li R7, (R4) and li R8, (R6)).
5. **Instruction: add R10, R9, R10**
   - **Dependencies**: R9 and R10 must be available to compute the new value of R10.
   - **Potential Hazard**: R9 is produced by the multiplication instruction (mult R9, R7, R8), so this add instruction must wait for the multiplication to complete.
6. **Instruction: add R4, R4, 1**
   - **Dependencies**: R4 must be available to increment it by 1.
   - **Potential Hazard**: If there are previous instructions modifying R4, this must wait for those to complete.
7. **Instruction: sub R5, R4, n**
   - **Dependencies**: R4 must be available to compute the new value of R5.

        - **Potential Hazard**: This depends on the result of the previous instruction (add R4, R4, 1).

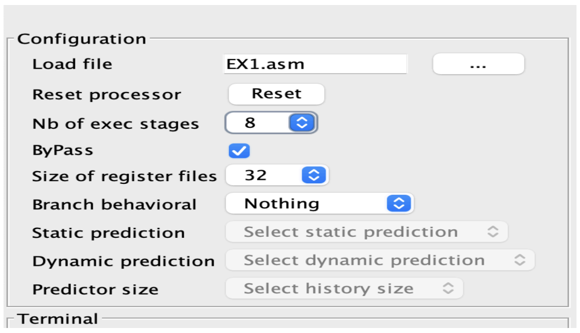
8. **Instruction: brnz R5, boucle**
    - **Dependencies**: R5 must be available to decide branching.
    - **Potential Hazard**: Depends on the subtraction instruction (sub R5, R4, n).

III. TASK 3

A. Load the code to JSimRiscz

Load file: EX1.asm

B. Configure the processor with 8 pipeline stages



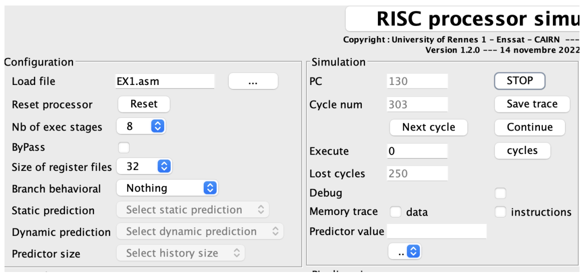
Configuration

Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	8
ByPass	✓
Size of register files	32
Branch behavioral	Nothing
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

Terminal

C. Analyze how many cycles are saved with the bypass technique

1. **Without Bypass:**
    - **Cycle Number**: 303
    - **Lost cycles:** 250



RISC processor simu

Copyright : University of Rennes 1 – Enssat – CAIRN ---
Version 1.2.0 --- 14 novembre 2022

Configuration

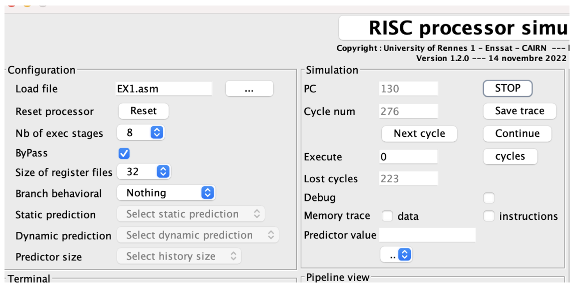
Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	8
ByPass	
Size of register files	32
Branch behavioral	Nothing
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

Simulation

PC	130
Cycle num	303
Execute	0
Lost cycles	250
Debug	
Memory trace	data, instructions
Predictor value	

2. **With Bypass:**

- **Cycle Number :** 276
- **Lost Cycles :** 223



RISC processor simu

Copyright : University of Rennes 1 – Enssat – CAIRN ---
Version 1.2.0 --- 14 novembre 2022

Configuration

Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	8
ByPass	✓
Size of register files	32
Branch behavioral	Nothing
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

Simulation

PC	130
Cycle num	276
Execute	0
Lost cycles	223
Debug	☐
Memory trace	☐ data ☐ instructions
Predictor value	

3. **Calculation of saved cycles:**
   - **Cycles Saved = Cycle Number without Bypass − Cycle Number with Bypass**
   - **Cycles Saved**=303 - 276 =27

   - **Lost Cycles Saved = Lost Cycles without Bypass − Lost Cycles with Bypass**
   - **Lost Cycles Saved**= 250 − 223 =27

D. Conclusion about the Bypass Technique:

The bypass technique consistently demonstrates significant performance improvements by saving 27 cycles and reducing the number of lost cycles by 27 in an 8-stage pipeline configuration. This aligns with earlier findings from the 6-stage configuration, highlighting the bypass technique's robustness and effectiveness in mitigating data hazards, improving processor efficiency, and enhancing overall performance. The consistent savings across different configurations confirm the scalability and practical benefits of the bypass technique in pipelined processor architectures.

IV. TASK 4

A. Explain which technique for branch instructions could be interesting:

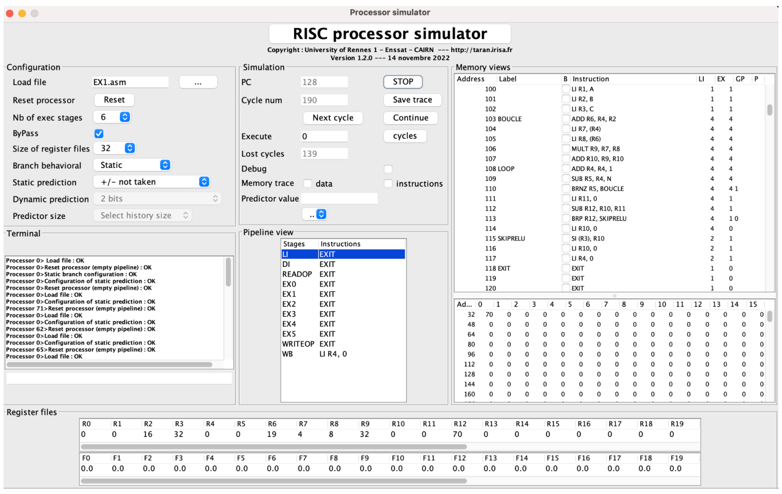
1. **Nothing:**
   - This option means no branch prediction is used.
   - The processor will always wait to resolve the branch, leading to potential stalls.
2. **Delay:**
   - This option introduces a delay slot after the branch instruction.
   - Instructions in the delay slot are executed regardless of the branch outcome, which can be utilized to improve performance.
3. **Static:**

- This option uses a simple static branch prediction strategy.
- Common static predictions include "always taken" or "always not taken."
- More advanced static strategies include "backward taken, forward not taken."

4. **Dynamic:**
   - This option employs dynamic branch prediction techniques.
   - Dynamic predictors use runtime information to make predictions and adapt to the actual branch behavior.
5. **Local History:**
   - This option uses local history-based branch prediction.
   - It keeps track of the history of individual branches to make predictions.

B. Test static and dynamic prediction

1. Configuration for Static Prediction:
   - **Pipeline Stages:** 6
   - **Branch Behavioral:** Static
   - **Static Prediction:** +/- not taken



Processor simulator

RISC processor simulator

Copyright : University of Rennes 1 – Enssat – CAIRN --- http://taran.irisa.fr
Version 1.2.0 --- 14 novembre 2022

Configuration

Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	6
ByPass	✓
Size of register files	32
Branch behavioral	Static
Static prediction	+/– not taken
Dynamic prediction	2 bits
Predictor size	Select history size

Simulation

PC	128
Cycle num	190
Execute	0
Lost cycles	139
Debug	
Memory trace	data / instructions
Predictor value	

Terminal

```
Processor 0> Load file : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Static branch configuration : OK
Processor 0>Configuration of static prediction : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 71>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 62>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 65>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
```

Pipeline view

Stages	Instructions
LI	EXIT
DI	EXIT
READOP	EXIT
EX0	EXIT
EX1	EXIT
EX2	EXIT
EX3	EXIT
EX4	EXIT
EX5	EXIT
WRITEOP	EXIT
WB	LI R4, 0

Memory views

Address	Label	B	Instruction	LI	EX	GP	P
100			LI R1, A	1	1		
101			LI R2, B	1	1		
102			LI R3, C	1	1		
103	BOUCLE		ADD R6, R4, R2	4	4		
104			LI R7, (R4)	4	4		
105			LI R8, (R6)	4	4		
106			MULT R9, R7, R8	4	4		
107			ADD R10, R9, R10	4	4		
108	LOOP		ADD R4, R4, 1	4	4		
109			SUB R5, R4, N	4	4		
110			BRNZ R5, BOUCLE	4	4 1		
111			LI R11, 0	4	1		
112			SUB R12, R10, R11	4	1		
113			BRP R12, SKIPRELU	4	1 0		
114			LI R10, 0	4	0		
115	SKIPRELU		SI (R3), R10	2	1		
116			LI R10, 0	2	1		
117			LI R4, 0	2	1		
118	EXIT		EXIT	1	0		
119			EXIT	1	0		
120			EXIT	1	0		

Ad...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

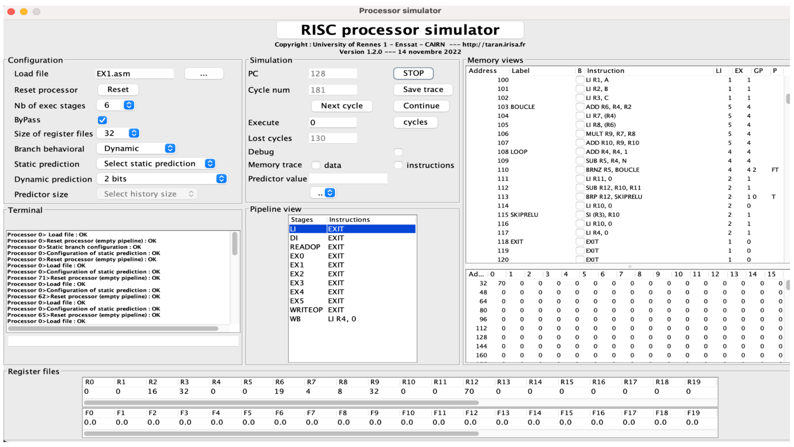
Register files

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
0	0	16	32	0	0	19	4	8	32	0	0	70	0	0	0	0	0	0	0

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19
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	0.0	0.0	0.0	0.0

   Results for Static Prediction:

   - **Cycle Number:** 190
   - **Lost Cycles:** 139
2. Configuration for Dynamic Prediction:
   - **Pipeline Stages:** 6
   - **Branch Behavioral:** Dynamic
   - **Dynamic Prediction:** 2-bit predictor


Processor simulator

RISC processor simulator

Copyright : University of Rennes 1 – Enssat – CAIRN --- http://taran.irisa.fr
Version 1.2.0 --- 14 novembre 2022

Configuration

Load file	EX1.asm
Reset processor	Reset
Nb of exec stages	6
ByPass	✓
Size of register files	32
Branch behavioral	Dynamic
Static prediction	Select static prediction
Dynamic prediction	2 bits
Predictor size	Select history size

Simulation

PC	128
Cycle num	181
Execute	0
Lost cycles	130
Debug	
Memory trace	data / instructions
Predictor value	

Terminal

```
Processor 0> Load file : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Static branch configuration : OK
Processor 0>Configuration of static prediction : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 71>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 62>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 65>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
```

Pipeline view

Stages	Instructions
LI	EXIT
DI	EXIT
READOP	EXIT
EX0	EXIT
EX1	EXIT
EX2	EXIT
EX3	EXIT
EX4	EXIT
EX5	EXIT
WRITEOP	EXIT
WB	LI R4, 0

Memory views

Address	Label	B	Instruction	LI	EX	GP	P
100			LI R1, A	1	1		
101			LI R2, B	1	1		
102			LI R3, C	1	1		
103	BOUCLE		ADD R6, R4, R2	5	4		
104			LI R7, (R4)	5	4		
105			LI R8, (R6)	5	4		
106			MULT R9, R7, R8	5	4		
107			ADD R10, R9, R10	5	4		
108	LOOP		ADD R4, R4, 1	4	4		
109			SUB R5, R4, N	4	4		
110			BRNZ R5, BOUCLE	4	4	2	FT
111			LI R11, 0	2	1		
112			SUB R12, R10, R11	2	1		
113			BRP R12, SKIPRELU	2	1	0	T
114			LI R10, 0	2	0		
115	SKIPRELU		SI (R3), R10	2	1		
116			LI R10, 0	2	1		
117			LI R4, 0	2	1		
118	EXIT		EXIT	1	0		
119			EXIT	1	0		
120			EXIT	1	0		

Ad...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register files

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
0	0	16	32	0	0	19	4	8	32	0	0	70	0	0	0	0	0	0	0

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19
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	0.0	0.0	0.0	0.0

Results for Dynamic Prediction:

- **Cycle Number:** 181
- **Lost Cycles:** 130

3. Conclusion:
   - **Performance Comparison:**
     - Dynamic branch prediction using a 2-bit predictor resulted in a lower cycle count (181) compared to static prediction (190).
     - Similarly, the lost cycles were fewer with dynamic prediction (130) compared to static prediction (139).
   - **Effectiveness of Dynamic Prediction:**
     - Dynamic branch prediction is more effective in reducing the number of pipeline stalls due to branch mispredictions.
     - The 2-bit predictor adapts to the actual behavior of the branches, providing more accurate predictions and thus reducing the number of lost cycles and improving overall performance.

V. TASK 5

A. What you need to do before executing the code with the delayed branch technique

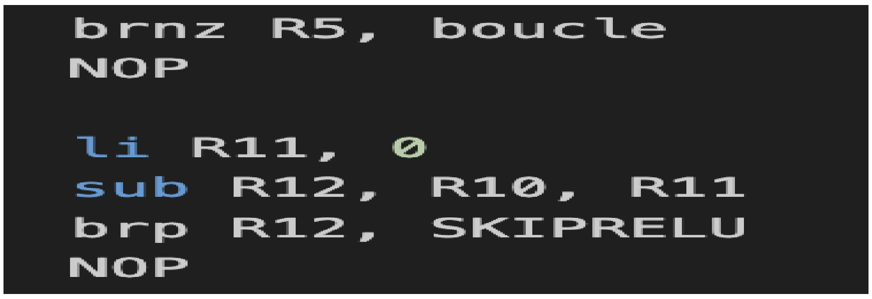
technique:

- **Identify Branch Instructions:**
  - Find all branch instructions in your assembly code.
- **Insert Delay Slot Instructions:**
  - Place a useful instruction in the delay slot immediately following each branch instruction.

- ○ This could be a NOP (no operation) if no useful instruction can be placed, but ideally, it should be an instruction that performs a necessary operation.

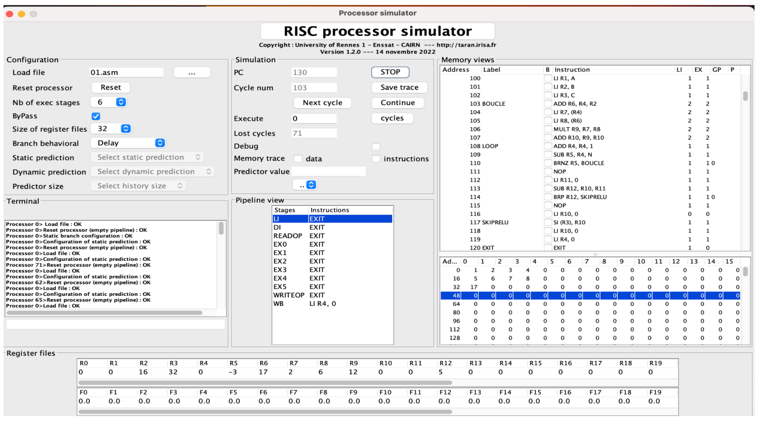
Adding Delay Slots:

- **After brnz R5, boucle:**
    - ○ Insert a useful instruction or a NOP after this branch.
- **After brp R12, SKIPRELU:**
    - ○ Insert a useful instruction or a NOP after this branch.



```
brnz R5, boucle
NOP

li R11, 0
sub R12, R10, R11
brp R12, SKIPRELU
NOP
```

B. Without Optimization:



Processor simulator

RISC processor simulator

Copyright : University of Rennes 1 – Enssat – CAIRN --- http://taran.irisa.fr
Version 1.2.0 --- 14 novembre 2022

Configuration

Load file	01.asm
Reset processor	Reset
Nb of exec stages	6
ByPass	✓
Size of register files	32
Branch behavioral	Delay
Static prediction	Select static prediction
Dynamic prediction	Select dynamic prediction
Predictor size	Select history size

Simulation

PC	130
Cycle num	103
Execute	0
Lost cycles	71
Debug	
Memory trace	data / instructions
Predictor value	

Terminal

```
Processor 0> Load file : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Static branch configuration : OK
Processor 0>Configuration of static prediction : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 71>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 62>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 65>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
```

Pipeline view

Stages	Instructions
LI	EXIT
DI	EXIT
READOP	EXIT
EX0	EXIT
EX1	EXIT
EX2	EXIT
EX3	EXIT
EX4	EXIT
EX5	EXIT
WRITEOP	EXIT
WB	LI R4, 0

Memory views

Address	Label	B	Instruction	LI	EX	GP	P
100			LI R1, A	1	1		
101			LI R2, B	1	1		
102			LI R3, C	1	1		
103	BOUCLE		ADD R6, R4, R2	2	2		
104			LI R7, (R4)	2	2		
105			LI R8, (R6)	2	2		
106			MULT R9, R7, R8	2	2		
107			ADD R10, R9, R10	2	2		
108	LOOP		ADD R4, R4, 1	1	1		
109			SUB R5, R4, N	1	1		
110			BRNZ R5, BOUCLE	1	1 0		
111			NOP	1	1		
112			LI R11, 0	1	1		
113			SUB R12, R10, R11	1	1		
114			BRP R12, SKIPRELU	1	1 0		
115			NOP	1	1		
116			LI R10, 0	0	0		
117	SKIPRELU		SI (R3), R10	1	1		
118			LI R10, 0	1	1		
119			LI R4, 0	1	1		
120	EXIT		EXIT	1	0		

Ad...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	0	0	0	0	0	0	0	0	0	0	0	0
16	5	6	7	8	0	0	0	0	0	0	0	0	0	0	0	0
32	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

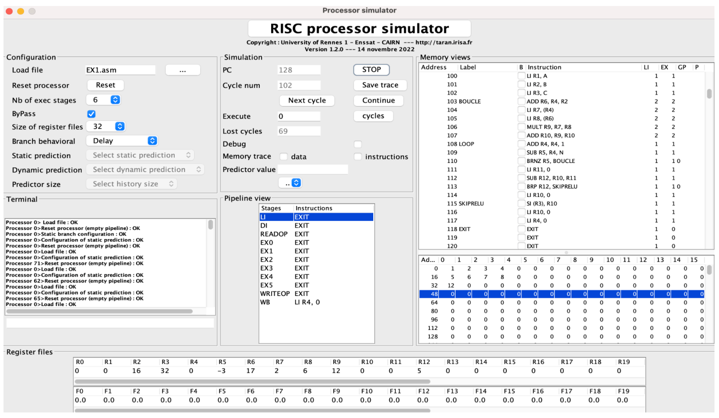
Register files

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
0	0	16	32	0	-3	17	2	6	12	0	0	5	0	0	0	0	0	0	0

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19
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	0.0	0.0	0.0	0.0

- **Cycle Number:** 103
- **Lost Cycles:** 71

C. With Optimization


Processor simulator

RISC processor simulator

Copyright : University of Rennes 1 – Enssat – CAIRN --- http://taran.irisa.fr
Version 1.2.0 --- 14 novembre 2022

Configuration
Load file: EX1.asm
Reset processor: Reset
Nb of exec stages: 6
ByPass: ✓
Size of register files: 32
Branch behavioral: Delay
Static prediction: Select static prediction
Dynamic prediction: Select dynamic prediction
Predictor size: Select history size

Simulation
PC: 128
Cycle num: 102
Execute: 0
Lost cycles: 69
Debug: ☐
Memory trace: ☐ data ☐ instructions
Predictor value:

Terminal
```
Processor 0> Load file : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Static branch configuration : OK
Processor 0>Configuration of static prediction : OK
Processor 0>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 71>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 62>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
Processor 0>Configuration of static prediction : OK
Processor 65>Reset processor (empty pipeline) : OK
Processor 0>Load file : OK
```

Pipeline view

Stages	Instructions
LI	EXIT
DI	EXIT
READOP	EXIT
EX0	EXIT
EX1	EXIT
EX2	EXIT
EX3	EXIT
EX4	EXIT
EX5	EXIT
WRITEOP	EXIT
WB	LI R4, 0

Memory views

Address	Label	B	Instruction	LI	EX	GP	P
100		☐	LI R1, A	1	1		
101		☐	LI R2, B	1	1		
102		☐	LI R3, C	1	1		
103	BOUCLE	☐	ADD R6, R4, R2	2	2		
104		☐	LI R7, (R4)	2	2		
105		☐	LI R8, (R6)	2	2		
106		☐	MULT R9, R7, R8	2	2		
107		☐	ADD R10, R9, R10	2	2		
108	LOOP	☐	ADD R4, R4, 1	1	1		
109		☐	SUB R5, R4, N	1	1		
110		☐	BRNZ R5, BOUCLE	1	1	0	
111		☐	LI R11, 0	1	1		
112		☐	SUB R12, R10, R11	1	1		
113		☐	BRP R12, SKIPRELU	1	1	0	
114		☐	LI R10, 0	1	1		
115	SKIPRELU	☐	SI (R3), R10	1	1		
116		☐	LI R10, 0	1	1		
117		☐	LI R4, 0	1	1		
118	EXIT	☐	EXIT	1	0		
119		☐	EXIT	1	0		
120		☐	EXIT	1	0		

Ad...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	0	0	0	0	0	0	0	0	0	0	0	0
16	5	6	7	8	0	0	0	0	0	0	0	0	0	0	0	0
32	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register files

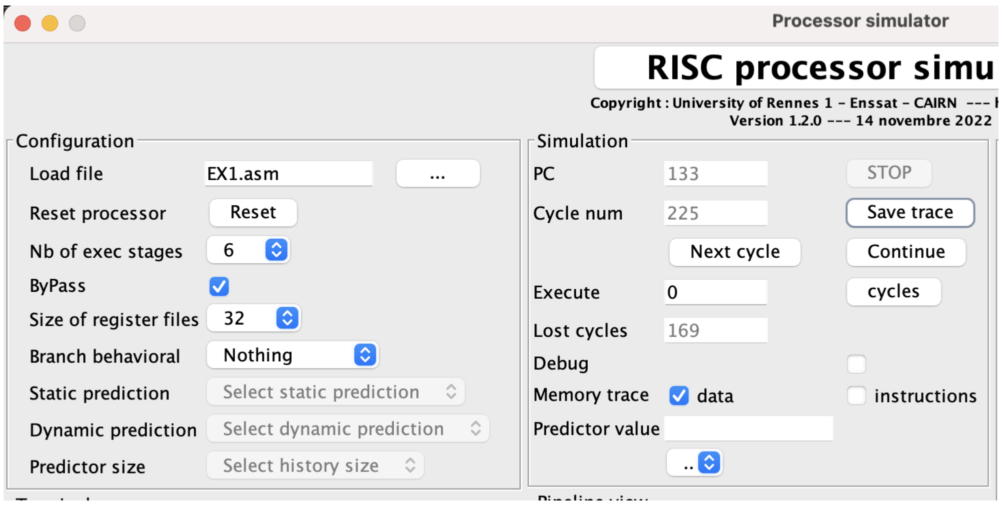
R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	R17	R18	R19
0	0	16	32	0	-3	17	2	6	12	0	0	5	0	0	0	0	0	0	0

F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19
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	0.0	0.0	0.0	0.0

- **Cycle Number:** 102
- **Lost Cycles:** 69

VI. TASK 6

A. Extract the data memory accesses with JSimRisc (6 pipeline stages)


Processor simulator

RISC processor simu

Copyright : University of Rennes 1 – Enssat – CAIRN ---
Version 1.2.0 --- 14 novembre 2022

Configuration
Load file: EX1.asm
Reset processor: Reset
Nb of exec stages: 6
ByPass: ✓
Size of register files: 32
Branch behavioral: Nothing
Static prediction: Select static prediction
Dynamic prediction: Select dynamic prediction
Predictor size: Select history size

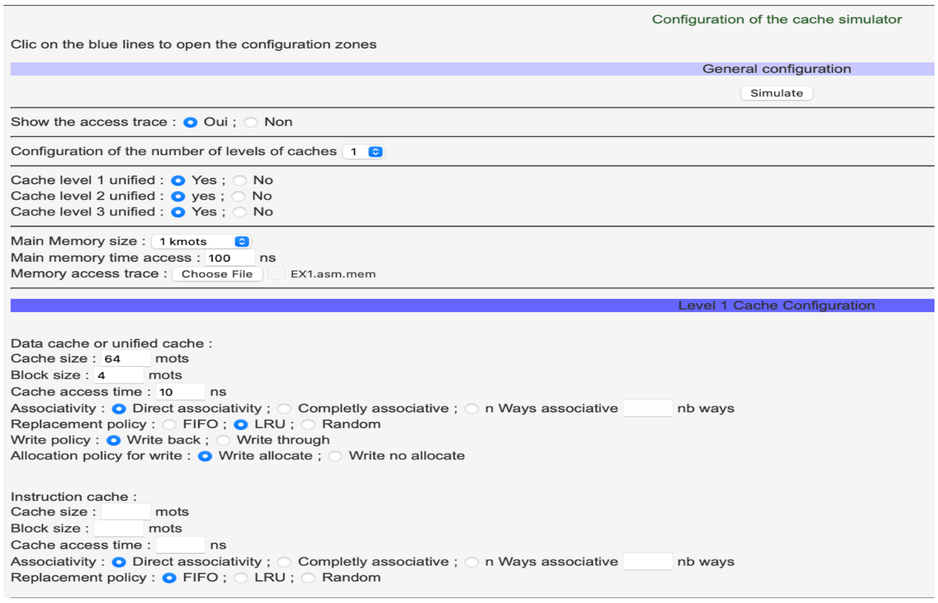
Simulation
PC: 133
Cycle num: 225
Execute: 0
Lost cycles: 169
Debug: ☐
Memory trace: ✓ data ☐ instructions
Predictor value:

Configuration and Simulation Steps

1. Load file: EX1.asm
2. Nb of exec stages: 6
3. Memory trace: data
4. Run
5. Save trace

B. With a cache size of 64 elements, try different cache configurations

1. **Configuration 1**



Configuration of the cache simulator

Clic on the blue lines to open the configuration zones

General configuration

Show the access trace : [✓ Oui] ; Non

Configuration of the number of levels of caches 1

Cache level 1 unified : [✓ Yes] ; No
Cache level 2 unified : [✓ yes] ; No
Cache level 3 unified : [✓ Yes] ; No

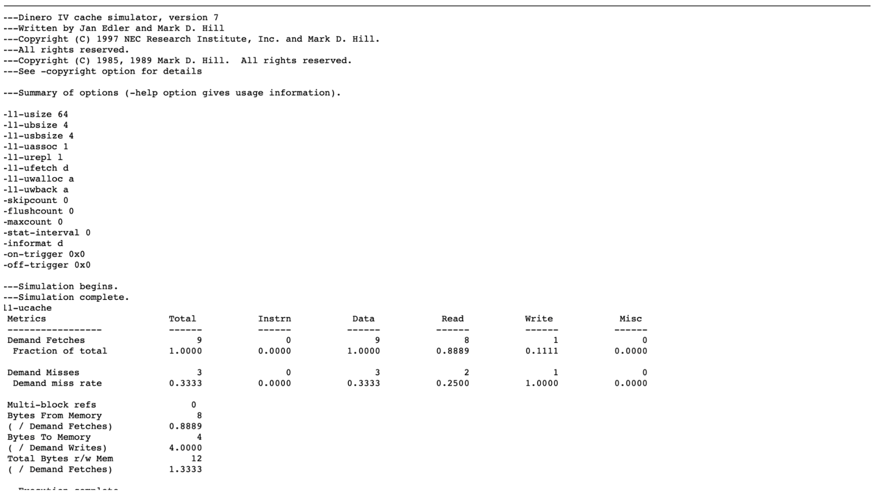
Main Memory size : 1 kmots
Main memory time access : 100 ns
Memory access trace : Choose File EX1.asm.mem

Level 1 Cache Configuration

Data cache or unified cache :
Cache size : 64 mots
Block size : 4 mots
Cache access time : 10 ns
Associativity : [✓ Direct associativity] ; Completly associative ; n Ways associative nb ways
Replacement policy : FIFO ; [✓ LRU] ; Random
Write policy : [✓ Write back] ; Write through
Allocation policy for write : [✓ Write allocate] ; Write no allocate

Instruction cache :
Cache size : mots
Block size : mots
Cache access time : ns
Associativity : [✓ Direct associativity] ; Completly associative ; n Ways associative nb ways
Replacement policy : [✓ FIFO] ; LRU ; Random

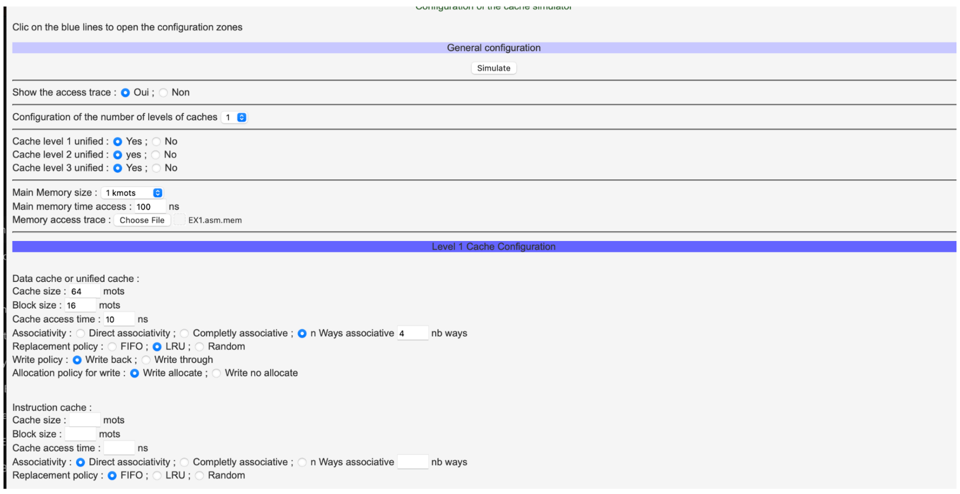
2. Results



```
---Dinero IV cache simulator, version 7
---Written by Jan Edler and Mark D. Hill
---Copyright (C) 1997 NEC Research Institute, Inc. and Mark D. Hill.
---All rights reserved.
---Copyright (C) 1985, 1989 Mark D. Hill.  All rights reserved.
---See -copyright option for details

---Summary of options (-help option gives usage information).

-l1-usize 64
-l1-ubsize 4
-l1-usbsize 4
-l1-uassoc 1
-l1-urepl l
-l1-ufetch d
-l1-uwalloc a
-l1-uwback a
-skipcount 0
-flushcount 0
-maxcount 0
-stat-interval 0
-informat d
-on-trigger 0x0
-off-trigger 0x0

---Simulation begins.
---Simulation complete.
l1-ucache
 Metrics                      Total           Instrn           Data            Read            Write            Misc
 -----------------            ------          ------          ------          ------          ------          ------
 Demand Fetches                   9               0               9               8               1               0
  Fraction of total          1.0000          0.0000          1.0000          0.8889          0.1111          0.0000

 Demand Misses                    3               0               3               2               1               0
  Demand miss rate           0.3333          0.0000          0.3333          0.2500          1.0000          0.0000

 Multi-block refs                 0
 Bytes From Memory                8
 ( / Demand Fetches)         0.8889
 Bytes To Memory                  4
 ( / Demand Writes)          4.0000
 Total Bytes r/w Mem             12
 ( / Demand Fetches)         1.3333
```

- Demand Miss Rate: 33.33%
- Bytes from Memory: 8
- Bytes to Memory: 4
- Total Bytes Read/Write: 12

3. Configuration 2


Clic on the blue lines to open the configuration zones

General configuration

Show the access trace : [✓ Oui] ; Non

Configuration of the number of levels of caches 1

Cache level 1 unified : [✓ Yes] ; No
Cache level 2 unified : [✓ yes] ; No
Cache level 3 unified : [✓ Yes] ; No

Main Memory size : 1 kmots
Main memory time access : 100 ns
Memory access trace : Choose File EX1.asm.mem

Level 1 Cache Configuration

Data cache or unified cache :
Cache size : 64 mots
Block size : 16 mots
Cache access time : 10 ns
Associativity : Direct associativity ; Completly associative ; [✓ n Ways associative] 4 nb ways
Replacement policy : FIFO ; [✓ LRU] ; Random
Write policy : [✓ Write back] ; Write through
Allocation policy for write : [✓ Write allocate] ; Write no allocate

Instruction cache :
Cache size : mots
Block size : mots
Cache access time : ns
Associativity : [✓ Direct associativity] ; Completly associative ; n Ways associative nb ways
Replacement policy : [✓ FIFO] ; LRU ; Random

4. **Results**
   - **Demand Miss Rate: 33.33%**
   - **Bytes from Memory: 48**
   - **Bytes to Memory: 16**
   - **Total Bytes Read/Write: 64**

C. Detailed Differences:

1. **Number of Misses/Hits:**
   - Both configurations resulted in 3 misses and 6 hits.
2. **Bytes Transferred:**
   - First Configuration: 8 bytes from memory, 4 bytes to memory.
   - Second Configuration: 48 bytes from memory, 16 bytes to **memory.**
3. **Block Size Impact:**
   - The larger block size in the second configuration increases the bytes fetched from memory, which can potentially reduce miss rates in other scenarios.
4. **Associativity Impact:**
   - The 4-way associativity in the second configuration helps in reducing conflict misses, even though it did not affect the miss rate in this specific case.

D. The best cache configuration, and explain

1. General Configuration
   - Show the access trace: `Oui`
   - Configuration of the number of levels of caches: `1`
   - Cache level 1 unified: `Yes`
   - Cache level 2 unified: `No`

- Cache level 3 unified: `No`
- Main Memory size: `1 kmots`
- Main memory time access: `100 ns`
- Memory access trace: `EX1.asm.mem` (file should be uploaded)

2. Level 1 Cache Configuration
    - Cache size: `64 mots`
    - Block size: `4 mots`
    - Cache access time: `2 ns`
    - Associativity: Select `n Ways associative` and enter `2`
    - Replacement policy: `LRU`
    - Write policy: `Write back`
    - Allocation policy for write: `Write allocate`
3. **Explaining**
    - **Cache Size (64 mots)**:
        - A moderate cache size ensures that the cache can store a reasonable amount of data without being too large to manage efficiently. A cache size of 64 mots strikes a balance between capacity and complexity.
    - **Block Size (4 mots)**:
        - Smaller block sizes tend to be beneficial for workloads with high spatial locality, where consecutive memory accesses are close to each other. With a block size of 4 mots, we reduce the chances of loading unnecessary data, thereby improving the miss penalty and making efficient use of the cache.
    - **Cache Access Time (2 ns)**:
        - A cache access time of 2 ns is reasonable and indicates that the cache can be accessed quickly, which is crucial for maintaining high performance in a pipelined processor environment.
    - **2-way Set Associative**:
        - **Pros**: This configuration reduces conflict misses compared to a direct-mapped cache (1-way associativity) without significantly increasing complexity and cost. By allowing two lines per set, we effectively reduce the likelihood of multiple data items competing for the same cache line.
        - **Cons**: While not as simple as a direct-mapped cache, the slight increase in complexity is justified by the improved performance due to reduced conflict misses.
    - **Replacement Policy (LRU)**:
        - LRU (Least Recently Used) is a commonly used and effective replacement policy that ensures that the least recently used data is replaced first. This is effective in improving cache performance

    for many typical access patterns where recently accessed data is more likely to be accessed again.

- **Write Policy (Write Back)**:
    - Write back is chosen over write through because it tends to offer better performance. With write back, writes are only made to the cache, and the actual memory is updated only when the cache line is replaced. This reduces the number of write operations to the slower main memory, improving overall performance.
- **Allocation Policy for Write (Write Allocate)**:
    - Write allocate (also known as fetch on write) is chosen because it ensures that cache lines are allocated for write misses, which can improve performance for write-heavy workloads. This policy works well with the write-back policy to maintain consistency and efficiency.