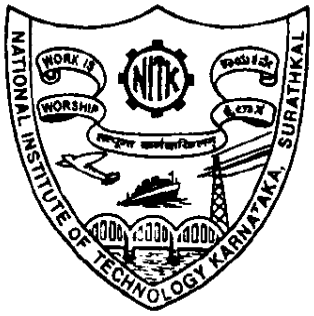
EC792 HPCA

LAB - 5



Report Submission

By

INBASEKARAN.P

201EC226

SAHIT VANKAYALA

201EE149

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

SRINIVASNAGAR 575025 KARNATAKA INDIA

11 March 2024

Exercises

1. 5-Stage Processor Without Forwarding or Hazard Detection

Objective:

- Write a sequence of instructions to show the importance of hazard detection. Your code should give incorrect results when run in this mode.

***Requirements**:*

- Include appropriate screenshots from Ripes.
- Explain the behavior and mention what hazard is causing this behavior.

Solution:

Instructions

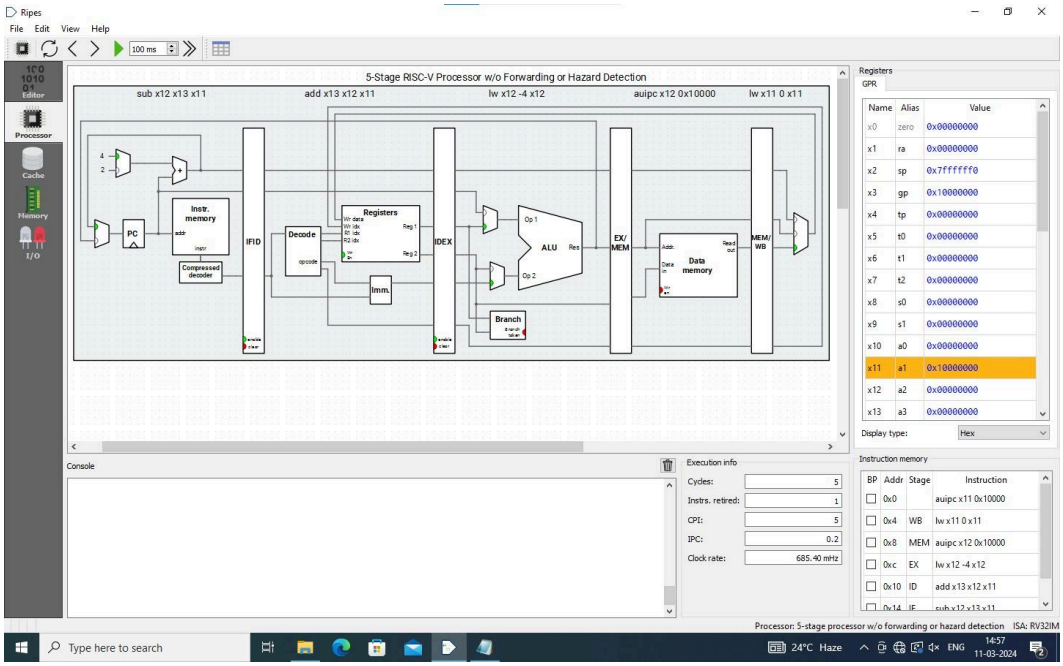
```
.data
aa: .word 5
bb: .word 7

.text
main:
    lw a1 aa
    lw a2 bb
    add a3 a2 a1
    sub a2 a3 a1
```

Explanation

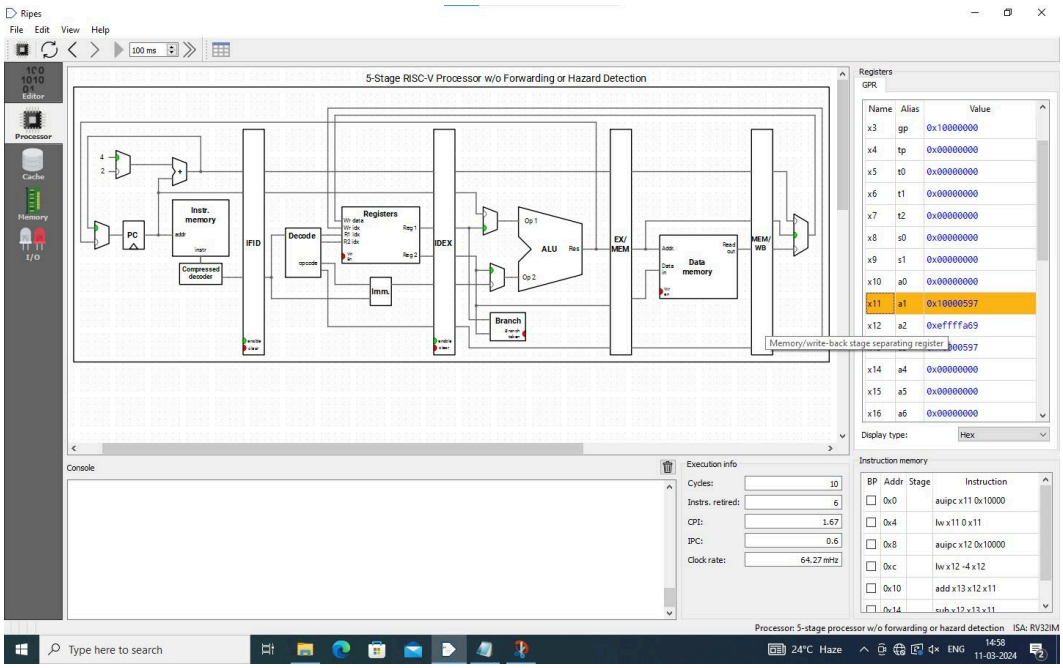
Read After Write (RAW): This hazard occurs when an instruction depends on the result of a previous instruction.

`add a3, a2, a1` depends on the data loaded by the previous `lw a1, aa` instruction. Similarly, `sub a2, a3, a1` depends on the results of `add a3, a2, a1`

In the absence of hazard detection and resolution mechanisms like forwarding or stalls, the `add` and `sub` instructions may execute with incorrect data. Without forwarding, the add instruction in the EX stage at cycle 3 would not have the correct value loaded from memory into a1 yet, as the lw instruction is still in the MEM stage. This causes the add instruction to use a stale or incorrect value for a1, leading to incorrect results.

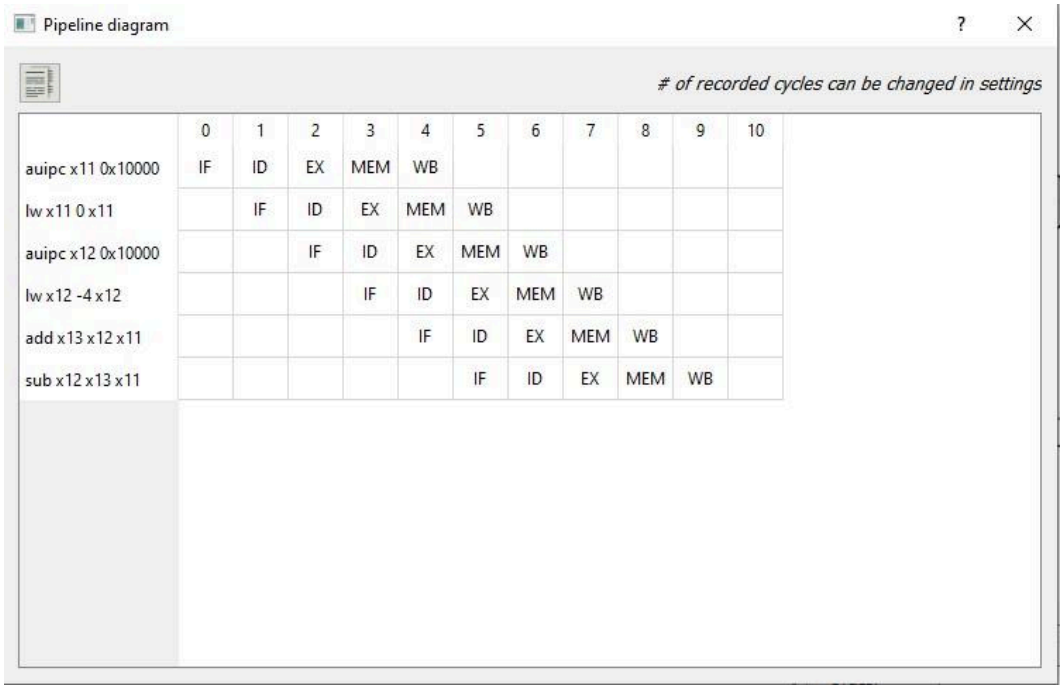


The first screenshot shows the processor after five cycles of execution. Here, the instruction lw x11, 0(x1) has been completed, but the subsequent add instruction may execute with an incorrect value for x11 because the loaded value hasn't been written back yet (due to the lack of forwarding).



The second screenshot shows the processor after ten cycles. The value in x11 (a1) has changed due to the sub-instruction. However, because of the data hazard described earlier, this value is incorrect,

as the sub-instruction should have used the result of the add instruction (which depends on the lw instruction that loaded the value into x11).



	0	1	2	3	4	5	6	7	8	9	10
auipc x11 0x10000	IF	ID	EX	MEM	WB						
lw x11 0 x11		IF	ID	EX	MEM	WB					
auipc x12 0x10000			IF	ID	EX	MEM	WB				
lw x12 -4 x12				IF	ID	EX	MEM	WB			
add x13 x12 x11					IF	ID	EX	MEM	WB		
sub x12 x13 x11						IF	ID	EX	MEM	WB	

The pipeline diagram screenshot shows the overlapping execution stages of the instructions. It clearly demonstrates the data hazard situation with lw, add, and sub instructions, especially how the add instruction overlaps with the lw instruction, indicating a potential RAW hazard.

2. 5-Stage Processor Without Forwarding, With Hazard Detection

Objective:

a. Write a sequence of instructions that would give stalls without forwarding and would give no stalls when both forwarding and hazard detection are enabled.

b. Write a sequence of instructions where stalls can be avoided by rearranging the instructions without changing the program logic.

Requirements:

- For both parts (a) and (b), add appropriate screenshots from Ripes, and explanations, and mention the hazard causing the behavior.

Solution:

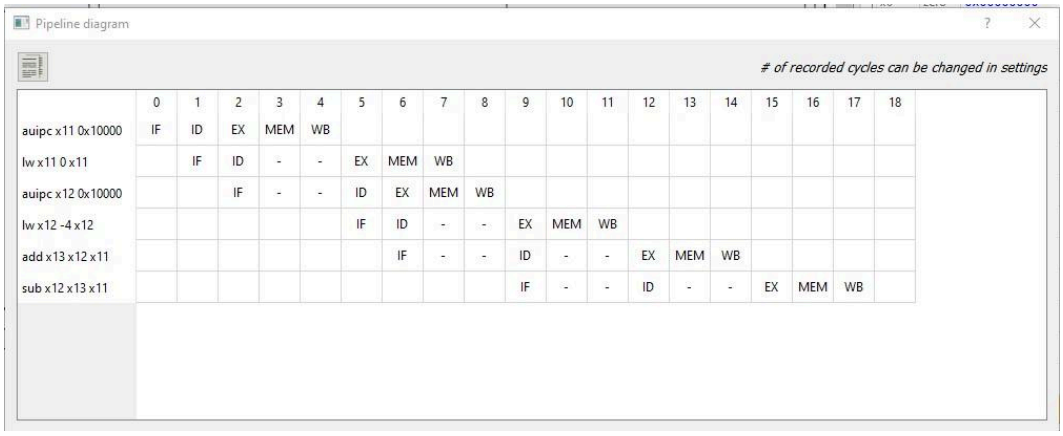
Part a: Stalls Due to Data Hazards Without Forwarding

Instructions

```
.data
aa: .word 5
bb: .word 7

.text
main:
    lw a1 aa
    lw a2 bb
    add a3 a2 a1
    sub a2 a3 a1
```

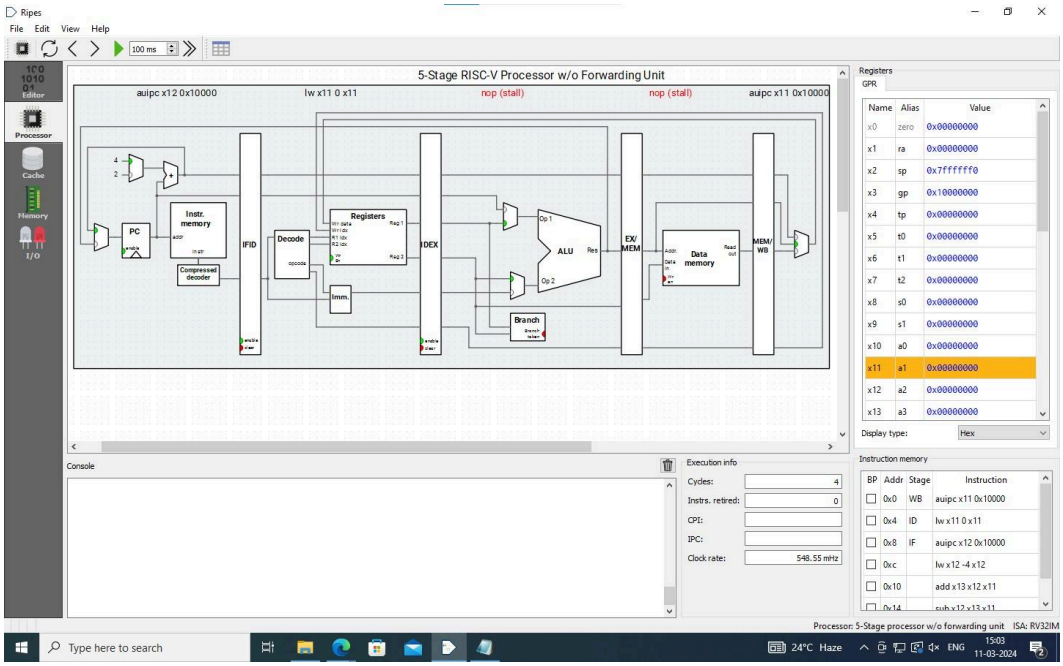
Explanation

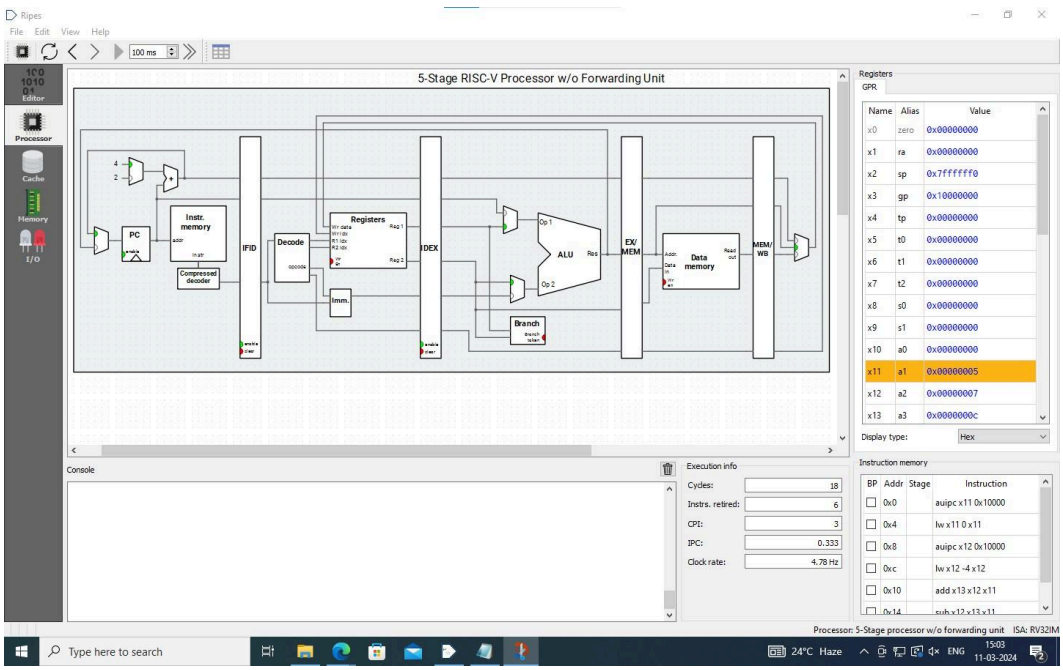
- The add instruction depends on the results of the two lw instructions that precede it. Without forwarding, the add instruction cannot immediately use the loaded values because they are not yet written back to the registers.
- The sub-instruction also encounters a RAW hazard since it depends on the result of the add instruction



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
auipc x11 0x10000	IF	ID	EX	MEM	WB														
lw x11 0 x11		IF	ID	-	-	EX	MEM	WB											
auipc x12 0x10000			IF	-	-	ID	EX	MEM	WB										
lw x12 -4 x12						IF	ID	-	-	EX	MEM	WB							
add x13 x12 x11							IF	-	-	ID	-	-	EX	MEM	WB				
sub x12 x13 x11										IF	-	-	ID	-	-	EX	MEM	WB	

- The pipeline diagram displays the execution flow of instructions over time across different stages (IF, ID, EX, MEM, WB).
- Stalls (indicated by empty slots or 'bubbles' in the pipeline) can be observed after the lw instructions, representing the processor's response to the RAW hazards.
- These stalls indicate that the processor must wait until the necessary data from the load instructions is available before proceeding with the dependent instructions (add and sub).





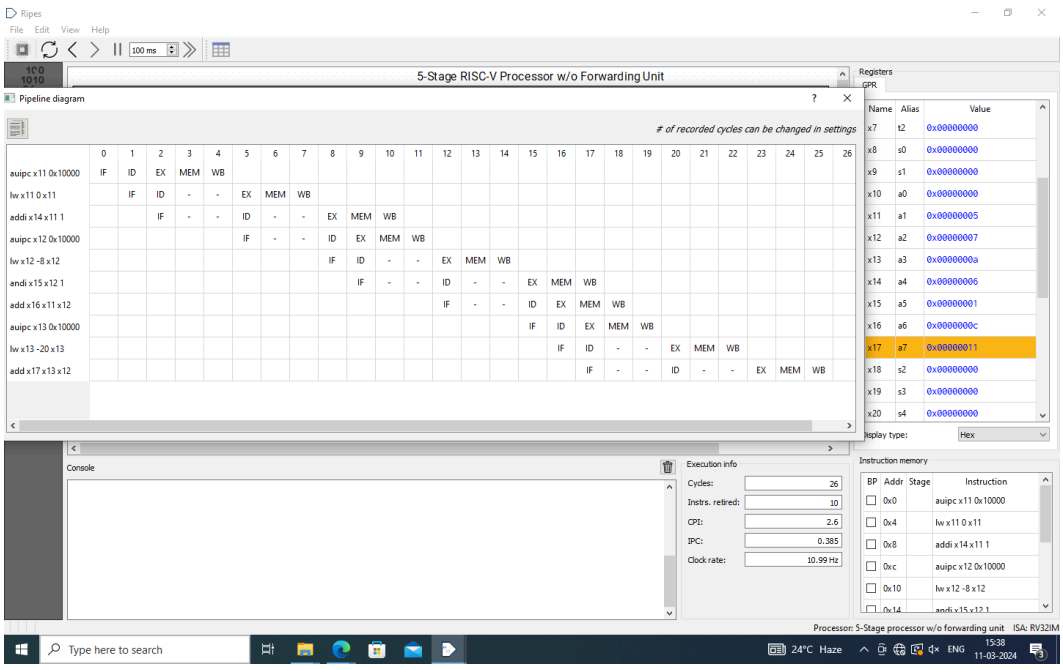
- These screenshots show the processor's register state and the pipeline stages at a specific cycle during execution.
- The highlighted registers (e.g., a1) show the values that have been loaded or computed so far. However, due to the lack of a forwarding unit, subsequent instructions that depend on these values must wait, causing the observed stalls.

Part b: Avoiding Stalls by Rearranging Instructions

Instructions

```
.data
aa: .word 5
bb: .word 7
cc: .word 10

.tex
main:
lw a1 aa
addi a4 a1 1
lw a2 bb
andi a5 a2 1
add a6 a1 a2
lw a3 cc
add a7 a3 a2
```

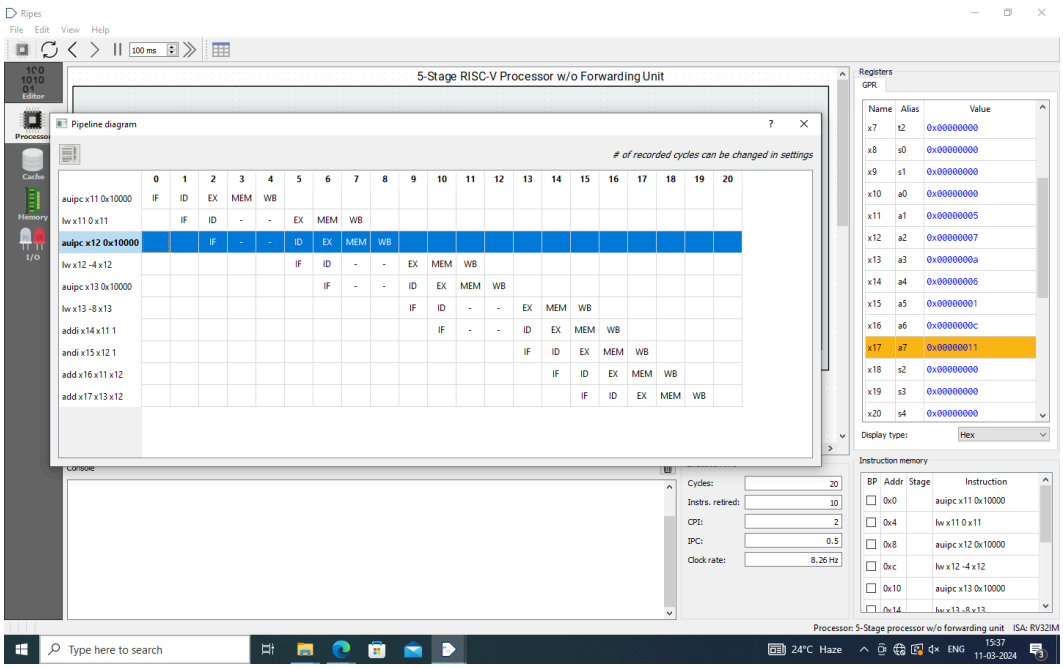
Explanation



Rearrange these instructions to remove RAW dependencies

Instructions

```
.data
aa: .word 5
bb: .word 7
cc: .word 10

.tex
main:
lw a1 aa
lw a2 bb
lw a3 cc
addi a4 a1 1
andi a5 a2 1
add a6 a1 a2
add a7 a3 a2
```



After rearranging the instructions the RAW dependencies have been eliminated and now the number of cycles to run the same program has reduced to 20 from 26.

3. Stalls and Forwarding

Instructions Set:

```
li t0 2
li t1 10
```

```
add t0 t0 t1
li t2 5
add t2 t0 t2
li a7 1
add a0 t2 x0
ecall
```

Tasks:

a. What is the value present in the datapath connecting the address input of data memory and EX/MEM pipeline after the 4th cycle, assuming the processor detects hazards and has forwarding?

b. What is the R2 idx input of the registers block and decoder after the 10th cycle, assuming the processor detects hazards and has no forwarding?

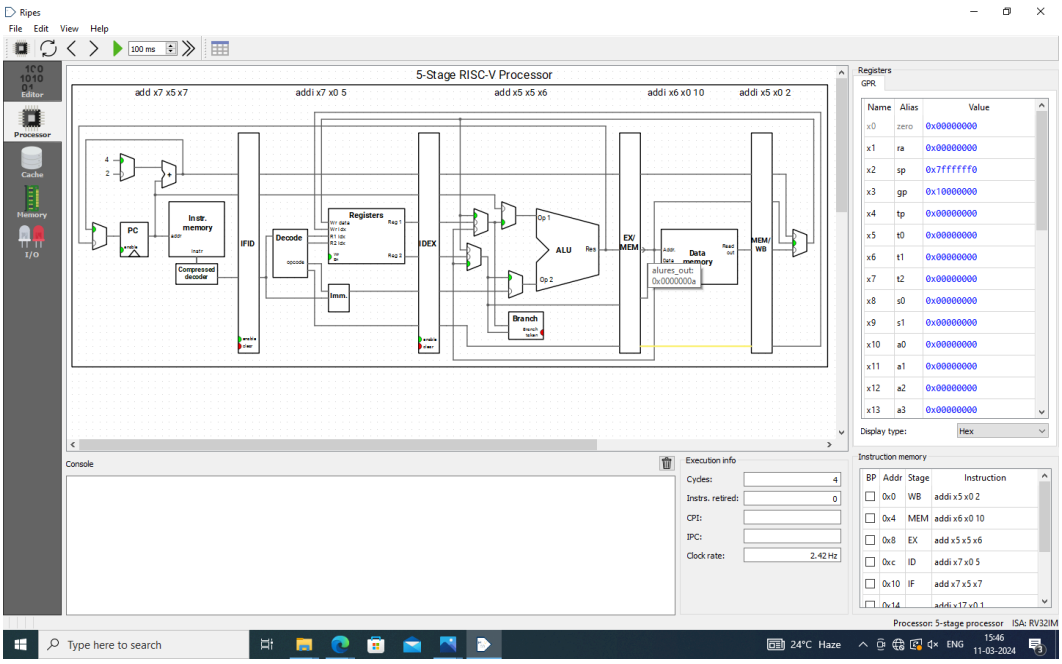
c. Assuming the processor to be a 6-stage dual-issue processor, what is the value stored in the opcode_exec_out datapath (present between IDII and IIEX) after 4 cycles?

d. Provide screenshots of pipeline diagrams for all three types of processors mentioned in parts a, b, and c. Write the number of cycles needed for each type of processor.

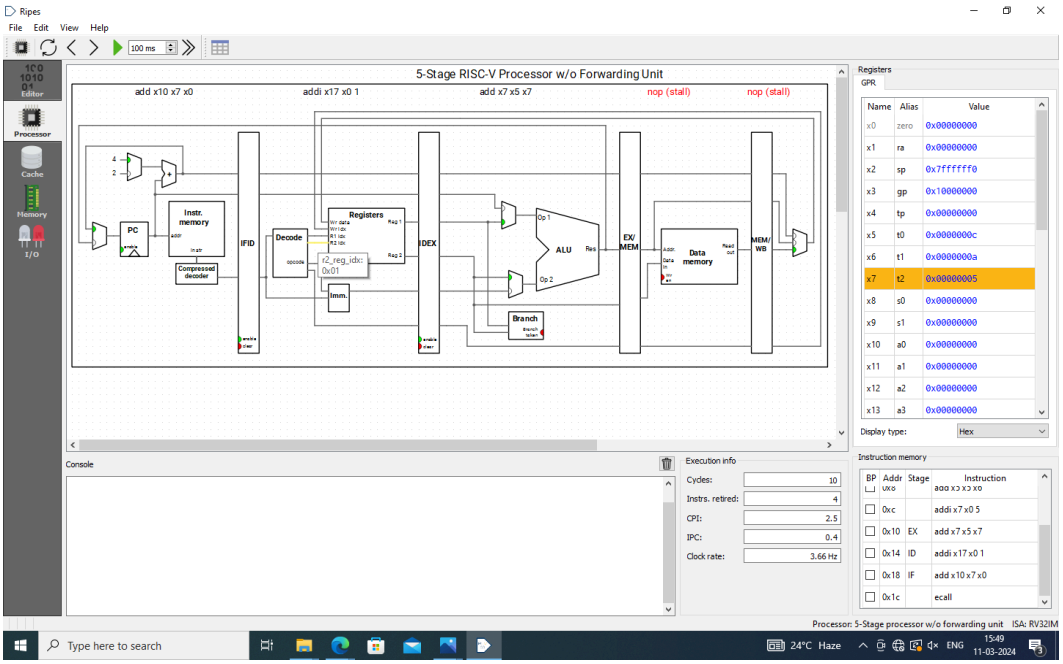
e. Assuming the processor to be a 5-stage processor without hazard detection, what is the output of the code? What is the reason for this output, and how can we overcome this problem to get the correct output? You are expected to modify the code if there's a problem.

Solution:

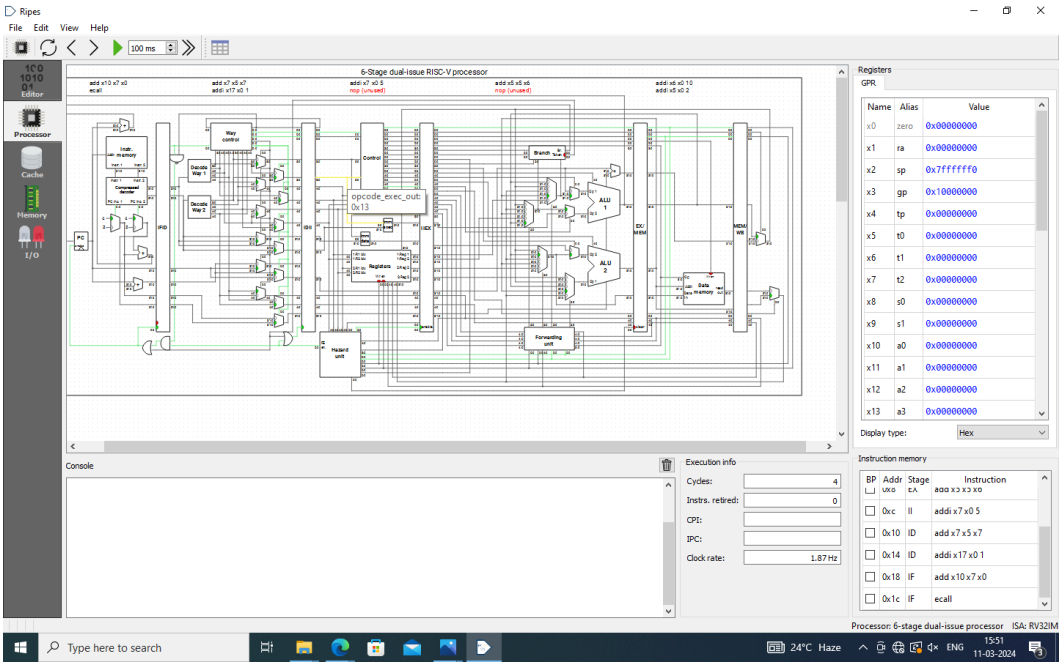
a. The address input of data memory and EX/MEM pipeline after the 4th cycle can be found by hovering over the signal in Ripes. The value is found to be **0x0000000a**



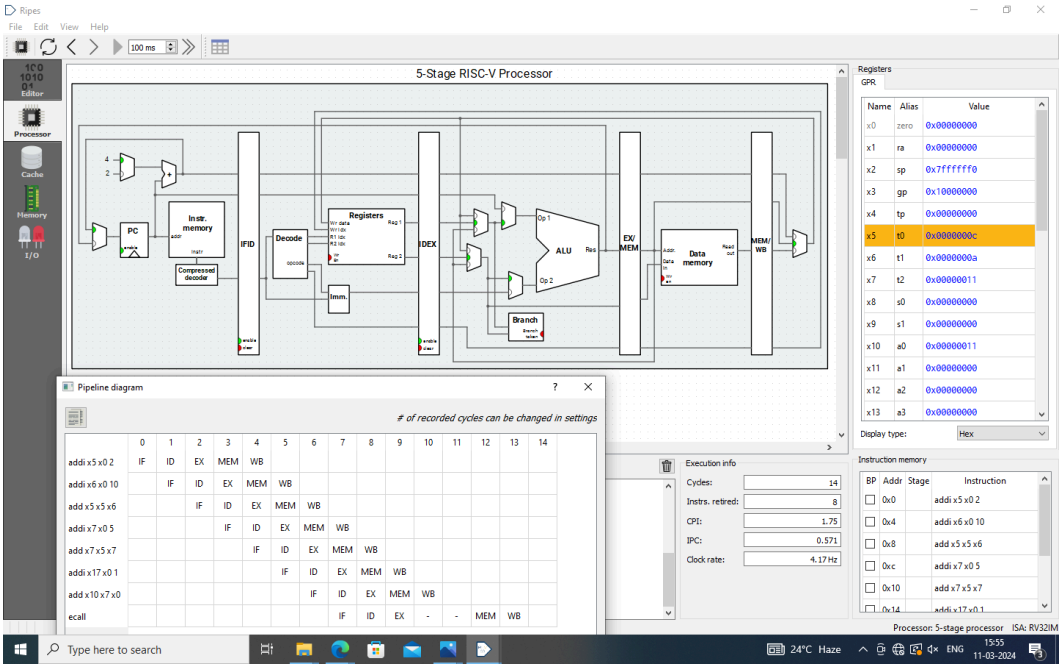
b. The R2 idx input of the registers block and decoder after the 10th cycle, assuming the processor detects hazards and has no forwarding is **0x01.**



c. The value stored in the opcode_exec_out datapath (present between IDII and IIEX) after 4 cycles in a 6-stage dual-issue processor is **0x13**

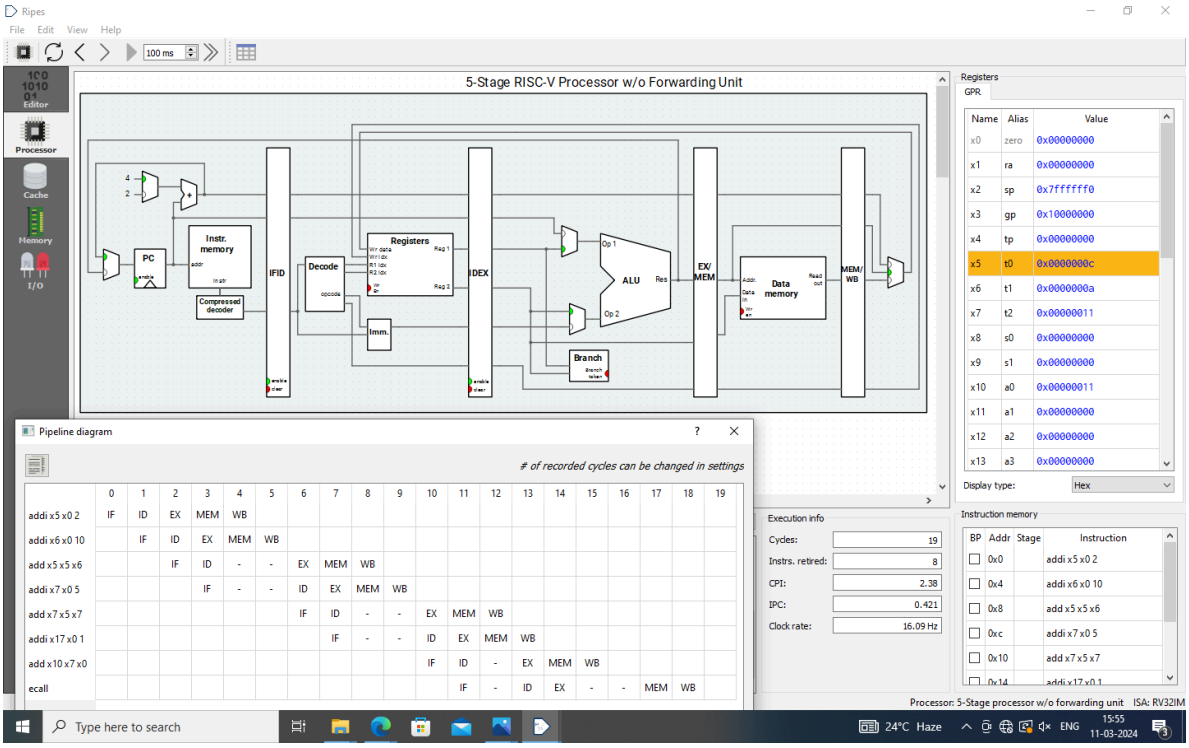


d. Number of cycles needed for the processor detects hazards and has forwarding: 14 Cycles



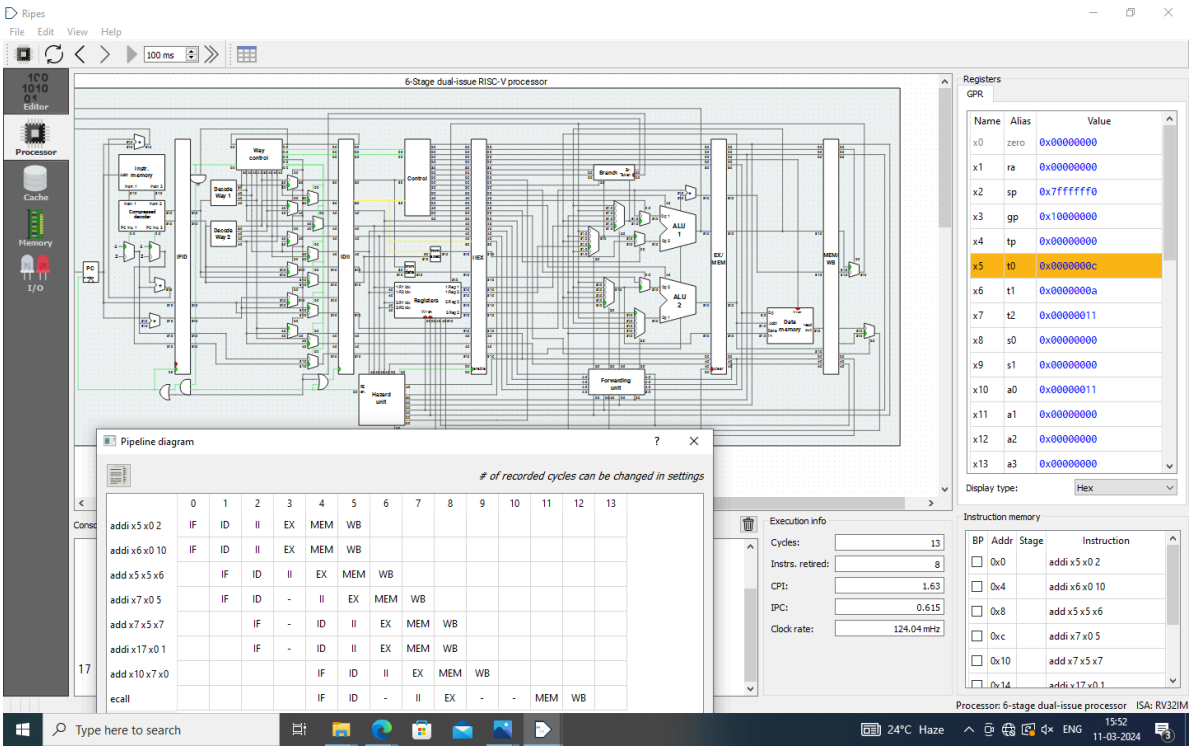
Number of cycles needed for the processor detects hazards and has no forwarding:

18 cycles

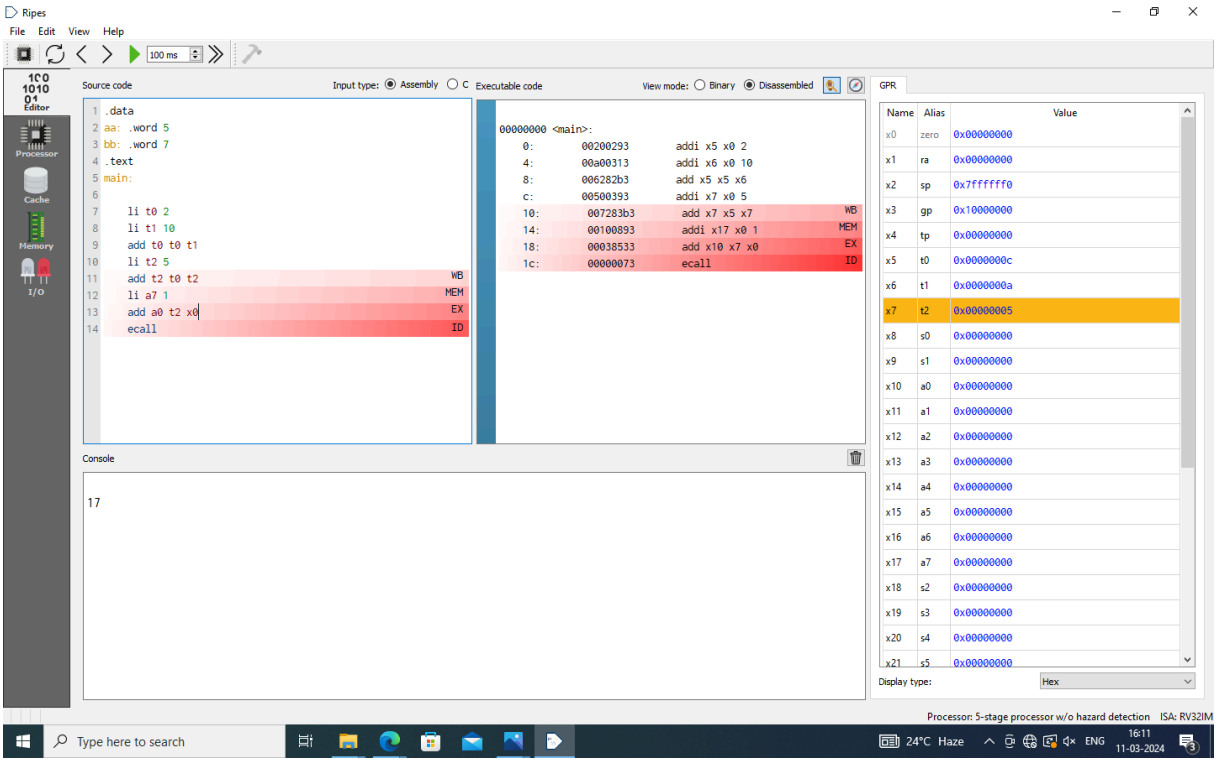


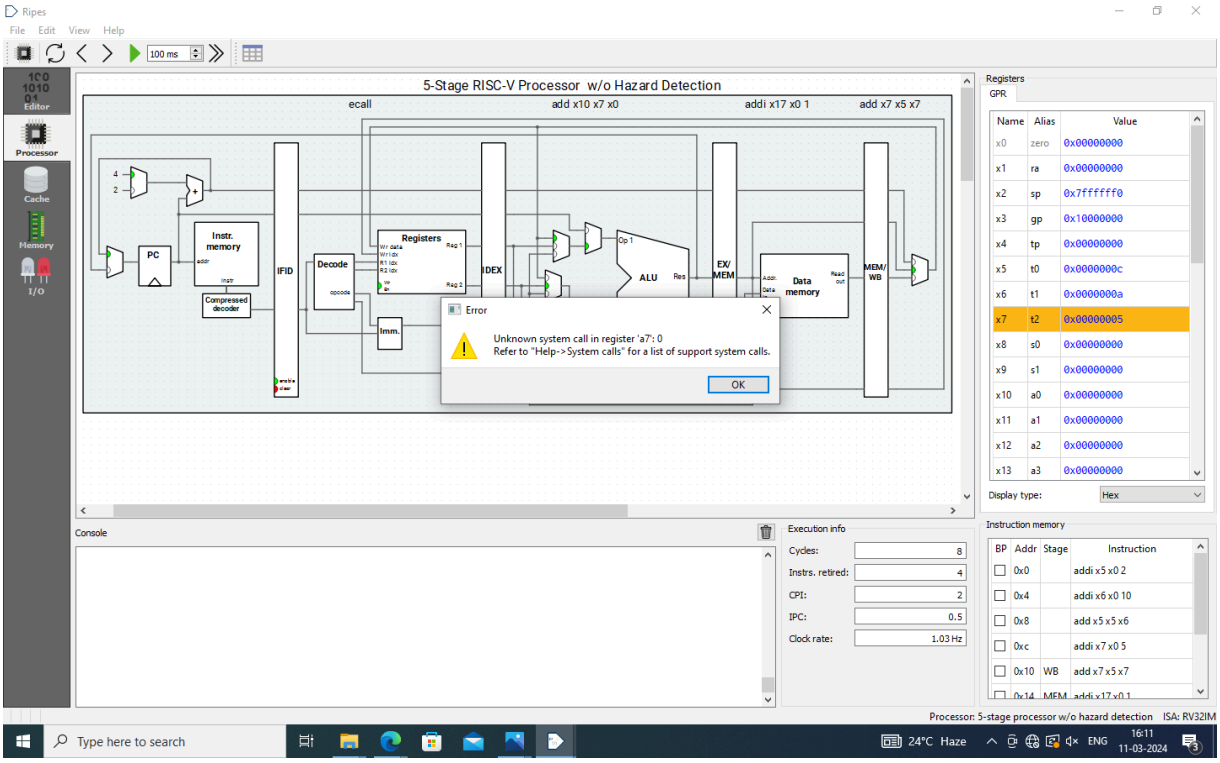
Number of cycles needed for the 6-stage dual-issue processor:

12 cycles

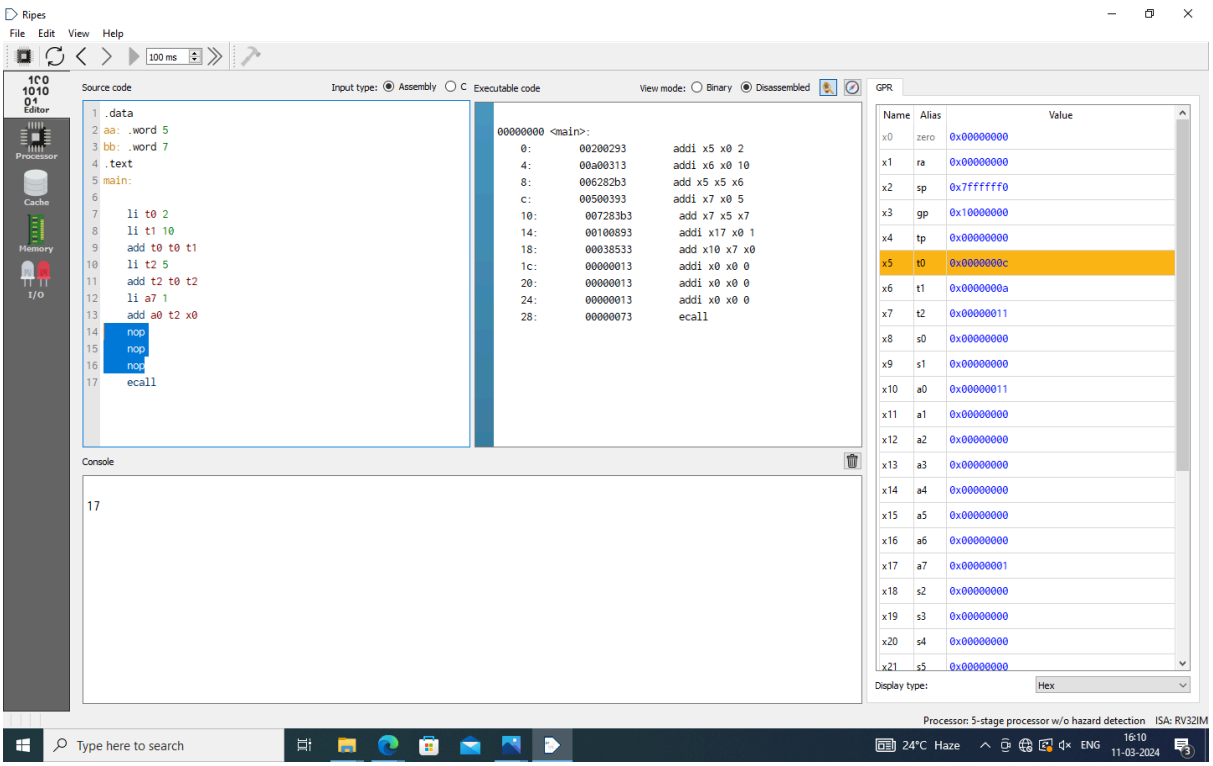


e. output of the code assuming the processor to be a 5-stage processor without hazard detection produces an error. This is because the register a7 does not have data written back to it and the value during the system call is still 0. The value only updates after register a7 has a value written back to it.





We have added **three** stalls (nop) to make sure that the data has been written back to the memory. This fixes the error and makes sure the correct output is displayed, i.e, **17**.



4. Integration and Cache Simulation

Tasks:

- From examples, load the program `ranpi.c`, compile the program, and run for 200ms. Then execute the simulator without updating the UI and observe the result in the console, CPI, and IPC.
- Now select the `matrixmul.c` example. Compile with the linker option `-nostdlib`. Select a single cycle processor.
- Go to the cache tab and execute the code with a time step of 100ms, and observe the cache getting filled.
- Note down the cache performance for various configurations.
- See the effect of matrix size and cache configurations on system performance.

Solution:

ranpi.c Simulation:

matrixmul.c Simulation:

Cache Performance Observations:

Effect of Matrix Size and Cache Configuration:

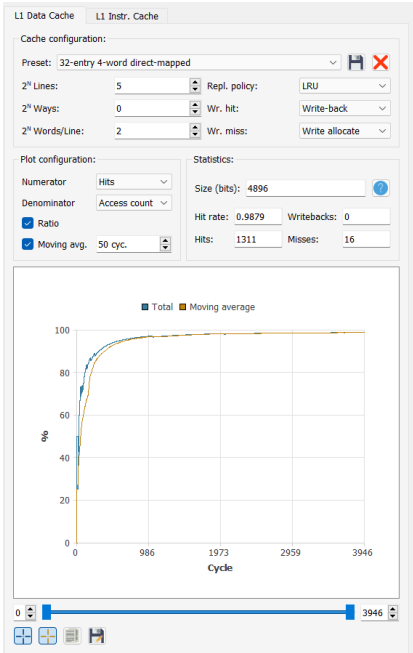
Since the Instruction cache is not affected by changing the configuration after a certain point the number of instructions is constant. The experiment was conducted on the Data-Cache on the following parameters and the following results were observed.

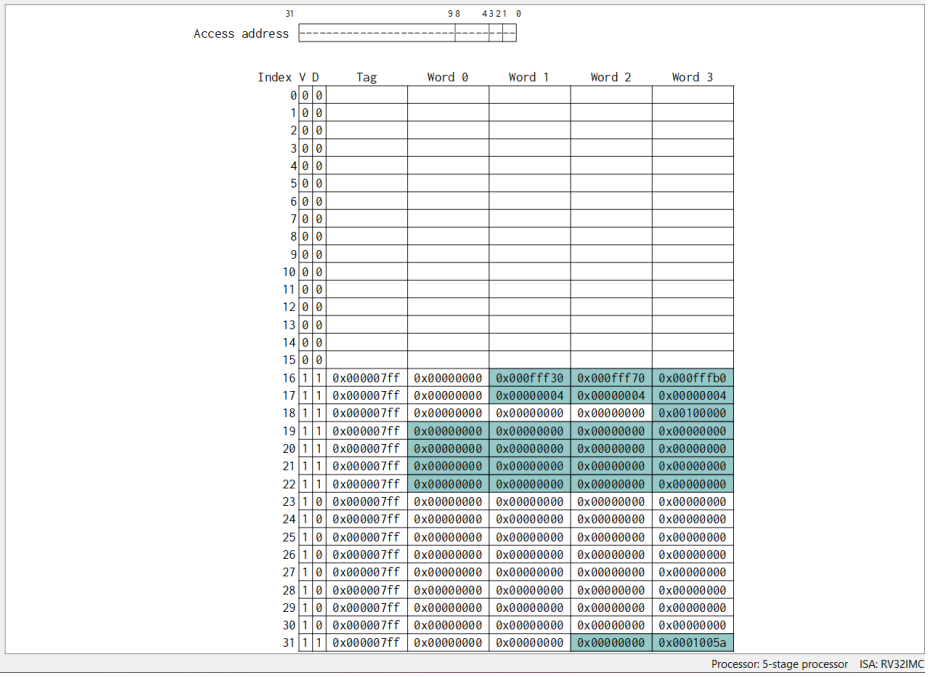
- Varying the size
- Variying the associativity
- Varying the write-policy

Varying the size of the matirx and keeping the cache configurations constant.

32 entry direct mapped 4-word cache

Size = 4

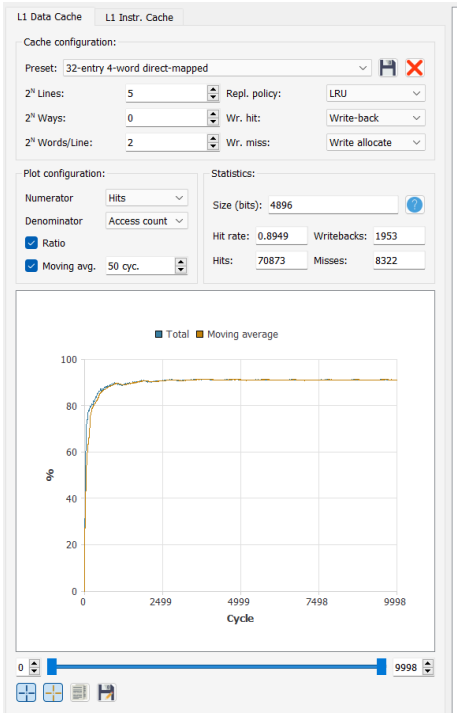


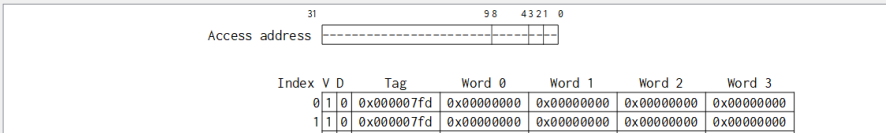


Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	0	0					
1	0	0					
2	0	0					
3	0	0					
4	0	0					
5	0	0					
6	0	0					
7	0	0					
8	0	0					
9	0	0					
10	0	0					
11	0	0					
12	0	0					
13	0	0					
14	0	0					
15	0	0					
16	1	1	0x000007ff	0x00000000	0x000fff30	0x000fff70	0x000fffb0
17	1	1	0x000007ff	0x00000000	0x00000004	0x00000004	0x00000004
18	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00100000
19	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
20	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
21	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
22	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
29	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
30	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
31	1	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

Processor: 5-stage processor ISA: RV32IMC

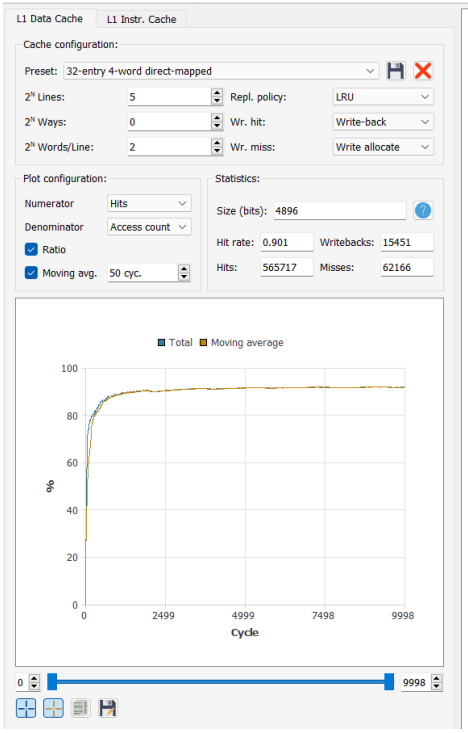
Size = 16

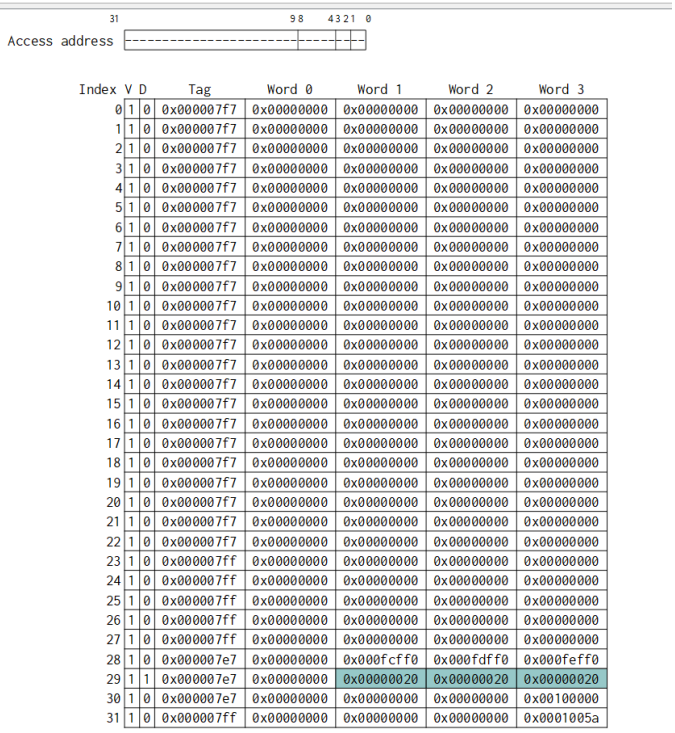




Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0	0x000007f9	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
29	1	1	0x000007f9	0x00000000	0x00000010	0x00000010	0x00000010
30	1	0	0x000007f9	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

Size = 32





Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0	0x000007f7	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0	0x000007e7	0x00000000	0x000fcff0	0x000fdff0	0x000feff0
29	1	1	0x000007e7	0x00000000	0x00000020	0x00000020	0x00000020
30	1	0	0x000007e7	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

Larger matrices can increase the chance of cache misses due to the increased memory footprint, potentially causing more cache evictions.

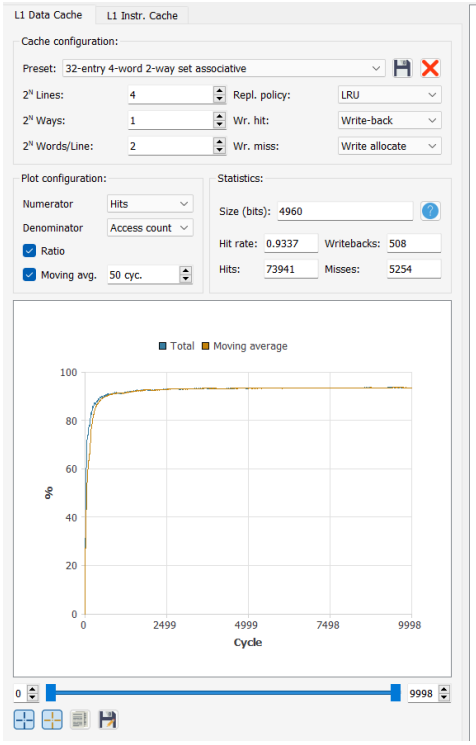
Smaller Matrices: Tend to fit better in the cache, reducing the need for frequent memory accesses. This usually results in higher cache hit rates and, consequently, improved performance due to reduced access times.

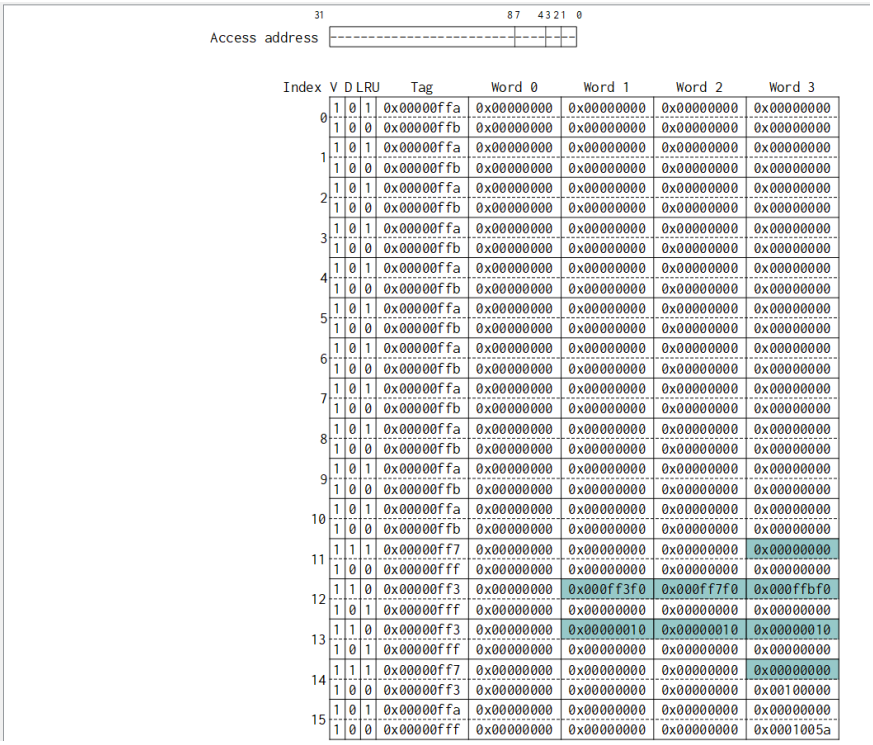
Larger Matrices: This may exceed the cache capacity, leading to increased cache misses. This forces the system to access slower main memory more frequently, which can significantly degrade performance.

Varying the associativity and keeping the size of the matrix constant.

Size=16

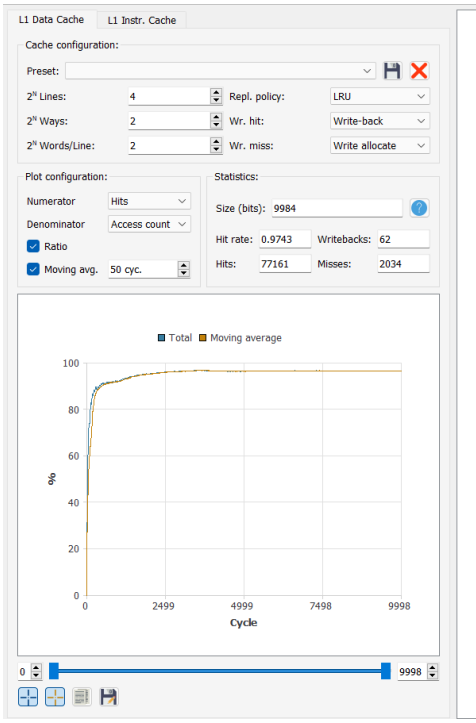
32 entry, 4-word, 2-way set associative cache

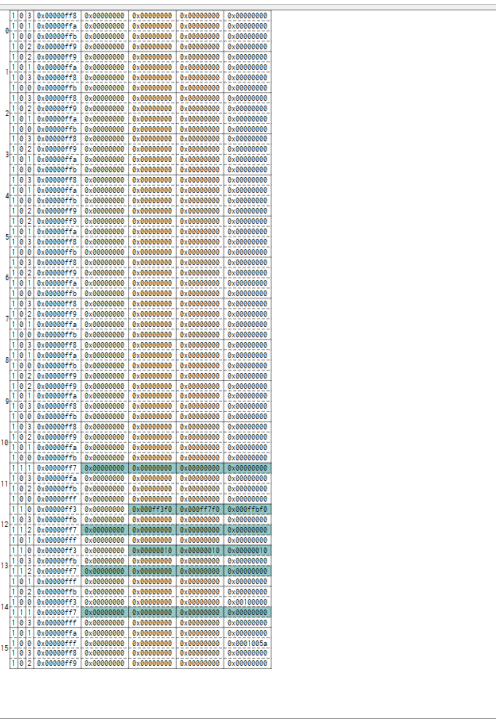




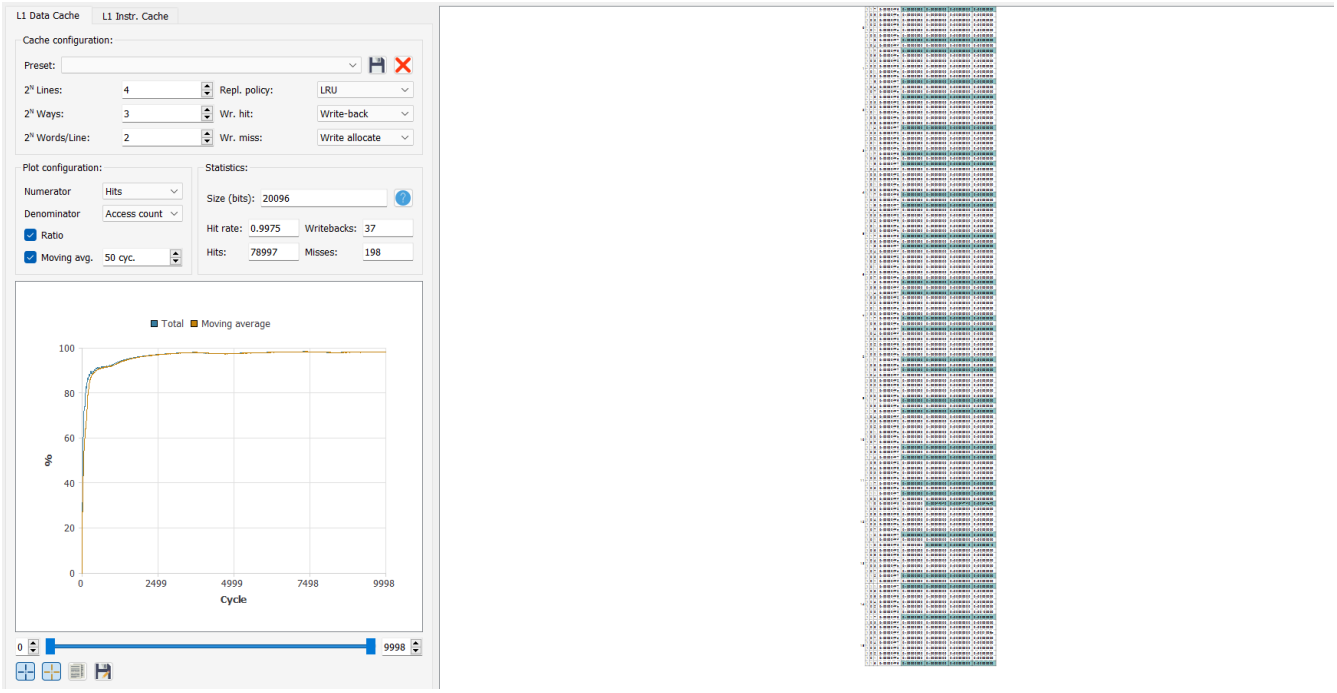
Index	V	D	LRU	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ffb	0x00000000	0x00000000	0x00000000	0x00000000
11	1	1	1	0x00000ff7	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000fff	0x00000000	0x00000000	0x00000000	0x00000000
12	1	1	0	0x00000ff3	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
	1	0	1	0x00000fff	0x00000000	0x00000000	0x00000000	0x00000000
13	1	1	0	0x00000ff3	0x00000000	0x00000010	0x00000010	0x00000010
	1	0	1	0x00000fff	0x00000000	0x00000000	0x00000000	0x00000000
14	1	1	1	0x00000ff7	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000ff3	0x00000000	0x00000000	0x00000000	0x00100000
15	1	0	1	0x00000ffa	0x00000000	0x00000000	0x00000000	0x00000000
	1	0	0	0x00000fff	0x00000000	0x00000000	0x00000000	0x0001005a

32 entry, 4-word, 4-way set associative cache





32 entry, 4-word, 8-way set associative cache



Associativity affects how many blocks can be stored in a cache set and can potentially reduce conflict misses.

Low Associativity (Direct-Mapped): This can lead to more cache misses due to conflicts, even if the cache is not fully utilized. This happens because each block can only be placed in one specific set, leading to potential overwrites of useful data.

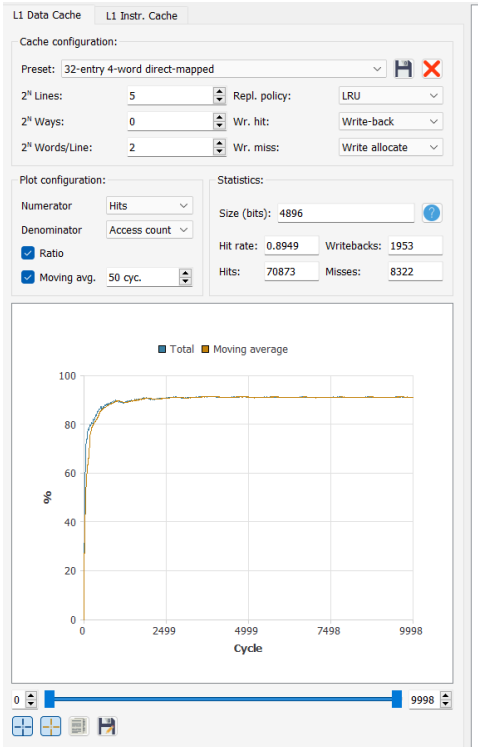
High Associativity (Fully Associative): Minimizes conflict misses by allowing a block to be placed in any cache line. While this can significantly reduce misses, it also requires more complex hardware and can increase the time it takes to check for hits.

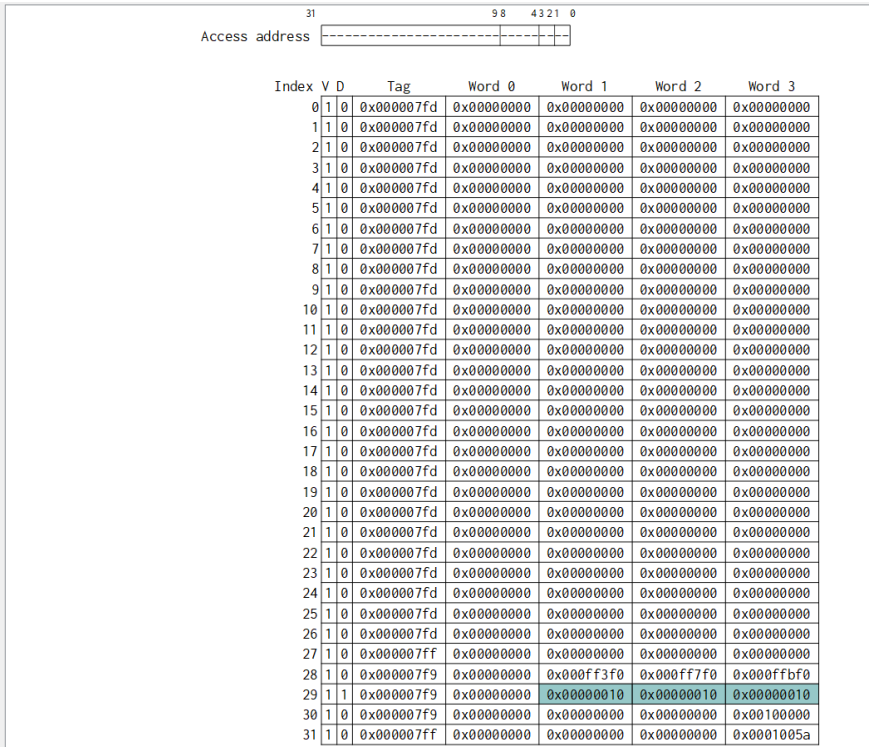
Set Associativity: A middle ground, balancing hit time, miss rate, and complexity. Allowing a block to be placed in any of several lines in a set, reduces conflict misses compared to direct-mapped caches while avoiding the complexity and longer hit times of fully associative caches.

Varying the write-back and write-allocate configurations

32 entry direct mapped 4-word cache

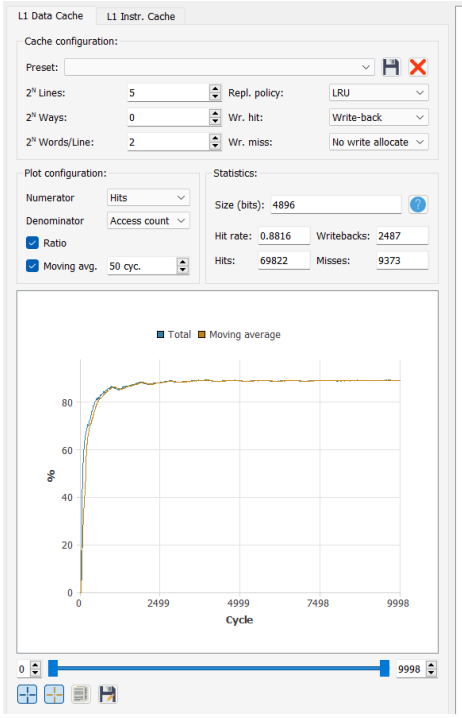
Write-back/Write-allocate

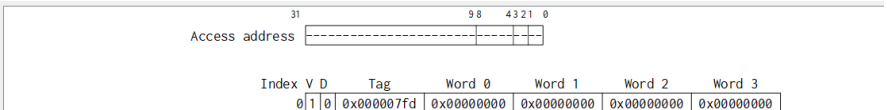




Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0	0x000007f9	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
29	1	1	0x000007f9	0x00000000	0x00000010	0x00000010	0x00000010
30	1	0	0x000007f9	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

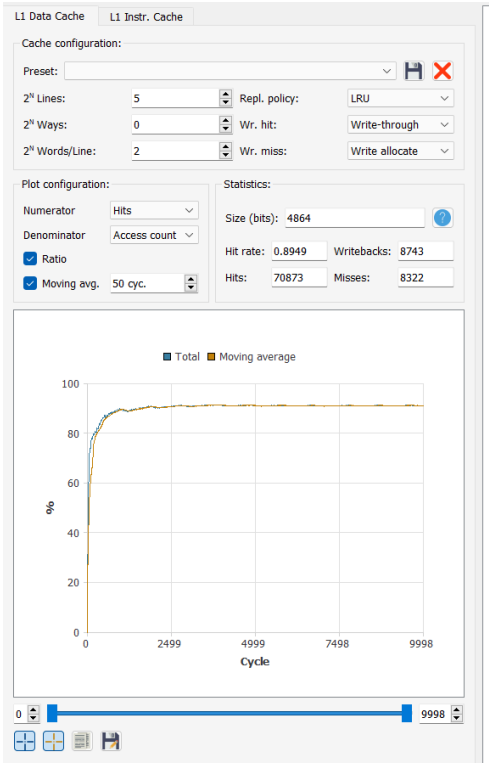
Write-back/No-Write-allocate

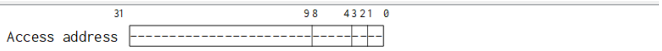




Index	V	D	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0	0x000007f9	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
29	1	1	0x000007f9	0x00000000	0x00000010	0x00000010	0x00000010
30	1	0	0x000007f9	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

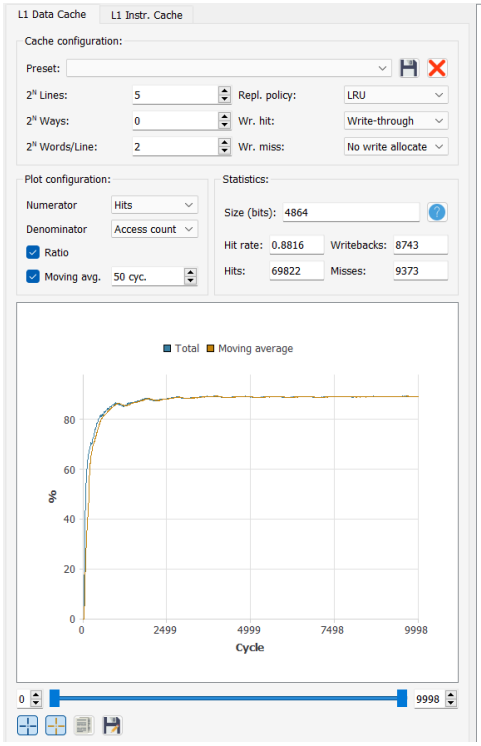
Write-though/Write-allocate





Index	V	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0x000007f9	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
29	1	0x000007f9	0x00000000	0x00000010	0x00000010	0x00000010
30	1	0x000007f9	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

Write-though/No-Write-allocate



0	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
1	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
2	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
3	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
4	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
5	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
6	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
7	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
8	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
9	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
10	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
11	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
12	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
13	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
14	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
15	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
16	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
17	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
18	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
19	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
20	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
21	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
22	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
23	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
24	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
25	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
26	1	0x000007fd	0x00000000	0x00000000	0x00000000	0x00000000
27	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x00000000
28	1	0x000007f9	0x00000000	0x000ff3f0	0x000ff7f0	0x000ffbf0
29	1	0x000007f9	0x00000000	0x00000010	0x00000010	0x00000010
30	1	0x000007f9	0x00000000	0x00000000	0x00000000	0x00100000
31	1	0x000007ff	0x00000000	0x00000000	0x00000000	0x0001005a

Write policies (write-back vs. write-through) and write allocation policies (write allocate vs. no-write allocate) impact how the cache interacts with the main memory.

Write-Through: Updates are written to both the cache and the main memory simultaneously. This ensures consistency between cache and memory but can lead to lower performance due to the high cost of writing to main memory.

Write-Back: Updates are only written to the cache. The modified cache lines are written back to the main memory only when they are evicted. This can significantly improve performance because writes to main memory are reduced. However, it requires additional complexity to handle the dirty bits (indicating changed lines) and can lead to inconsistencies between cache and memory if not managed correctly.

No-Write-Allocate: In the no-write-allocate policy, writes that miss in the cache are not used to bring new data into the cache. Instead, the write is directly forwarded to the lower level of the memory hierarchy (e.g., main memory). This approach reduces the cache pollution that might occur due to bringing in blocks that will only be written once and not read, which can be beneficial for write-intensive applications where the written data is not reused shortly after. However, it can result in lower performance for scenarios where written data will be read soon after because the data will have to be fetched from the slower main memory on subsequent reads.

Write-Allocate: With write-allocate, on a write miss, the cache block is loaded into the cache, and then the write is performed. This policy is based on the assumption that once a write occurs, subsequent reads or writes to nearby data are likely, to benefit from having the data in the cache. Write-allocate can improve performance for applications with the temporal locality in their write patterns, as subsequent accesses to the newly written data benefit from cache speeds. However, it might increase cache miss rates initially when the data is first brought into the cache and could also lead to unnecessary data being loaded into the cache if the subsequent accesses do not occur as anticipated.