Computer Architecture Homework 6

110062219

June 11, 2023

Contents

1 Cache Cases **1**
- 1.1 Cache Insertion . . . 2
- 1.2 Write Hit . . . 2
- 1.3 Write Back . . . 2

2 Cache Improvement **6**

3 Cache Configurations **7**
- 3.1 Sizes . . . 7
- 3.2 Example . . . 7
  - 3.2.1 Accesses . . . 7
  - 3.2.2 Final Contents . . . 7

4 Hamming Codes **8**
- 4.1 Single Error Correction . . . 8
- 4.2 Single Error Correction & Double Error Detection . . . 8
- 4.3 Yet Another Single Error Correction & Double Error Detection 8

5 Cache Performance **8**
- 5.1 Miss Penalties & Memory Stall Cycles . . . 8
- 5.2 Effective CPI . . . 9

6 Virtual Memory **9**

1 Cache Cases

Since there is no any instruction write to memory in my assembly codes of prior assignment, I did the experiments on the provided `KMP.elf`.

1.1 Cache Insertion

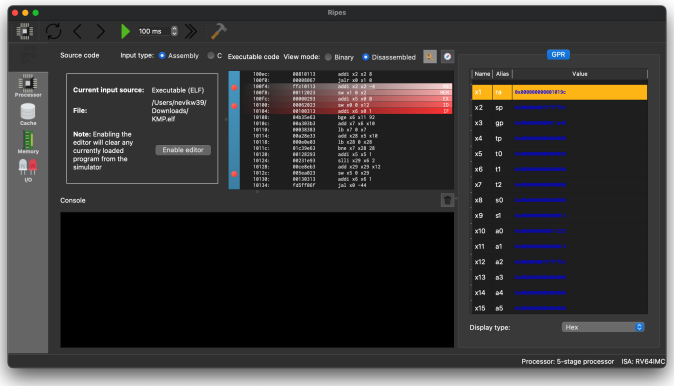
The screenshots are in Figure 1. Before the `sw` instruction, the set of index 3 contained exact 1 valid way whose LRU bit is 0. After that instruction, the LRU bit of the original way becomes 1 and the one of the newly-allocated way is 0.

1.2 Write Hit

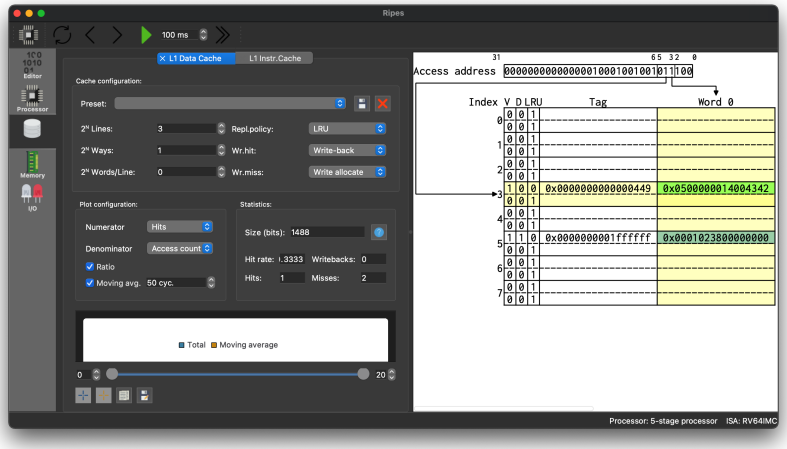
The screenshots are in Figure 2. Before the `sw` instruction, the word at address `0b01111111111111111111111110011000` was fetched in cache already with dirty bit unset. After the write instruction, its dirty bit becomes set thereby.

1.3 Write Back

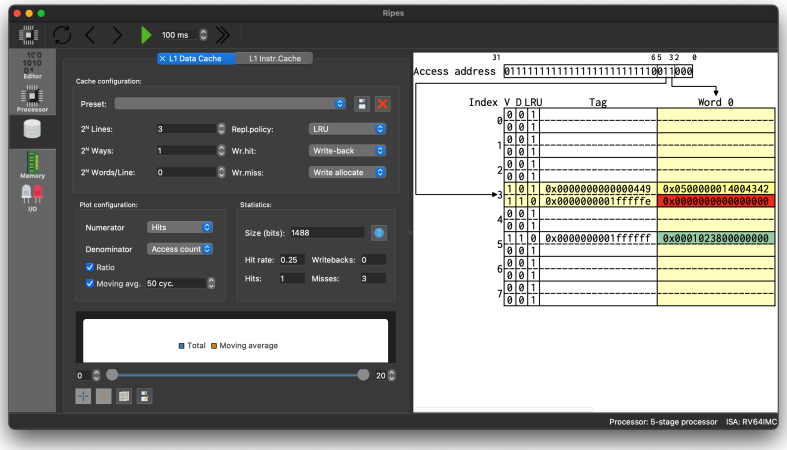
The screenshots are in Figure 3. Before the `sw` instruction, the set of index 3 was full already and the way whose LRU bit is 1 is dirty. After the instruction conflict to the third set, the dirty way is replaced and written back to upper level of memory.



(a) The instruction

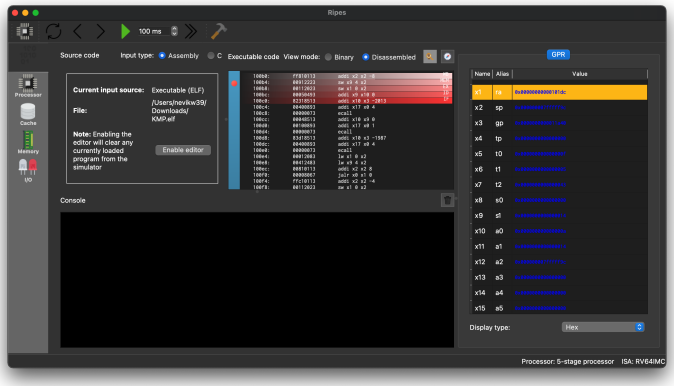


(b) Before

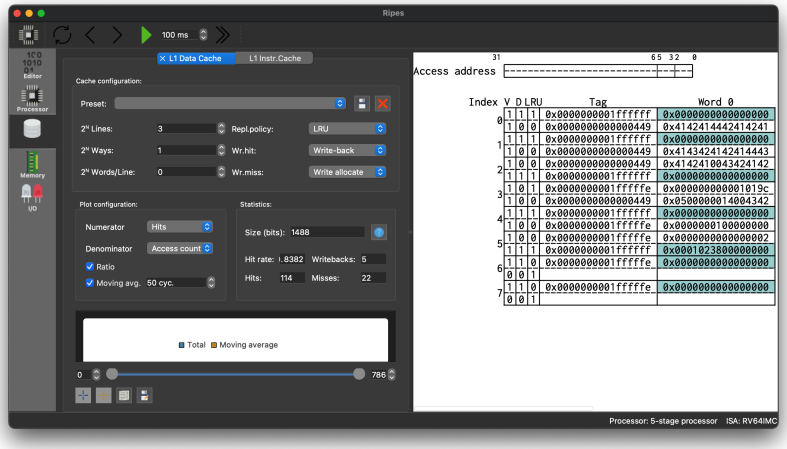


(c) After

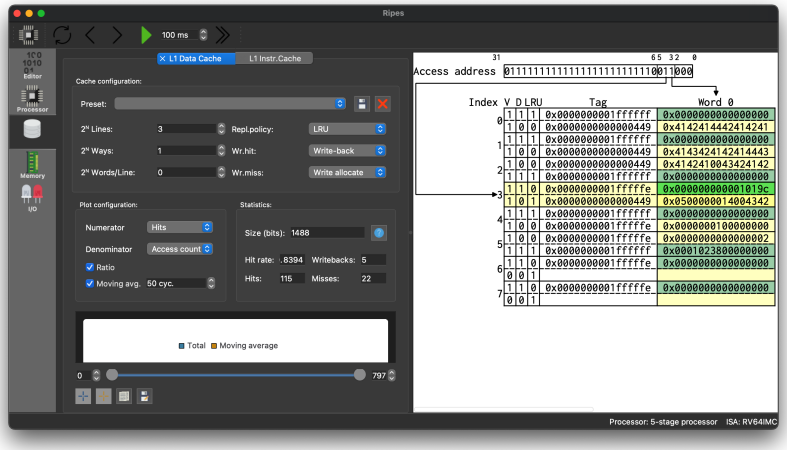
Figure 1: Cache Insertion



(a) The instruction

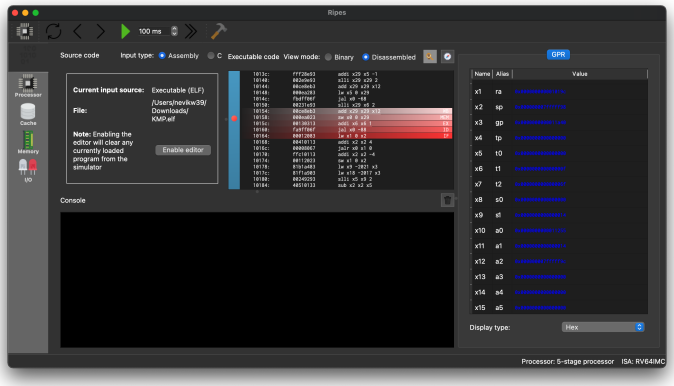


(b) Before

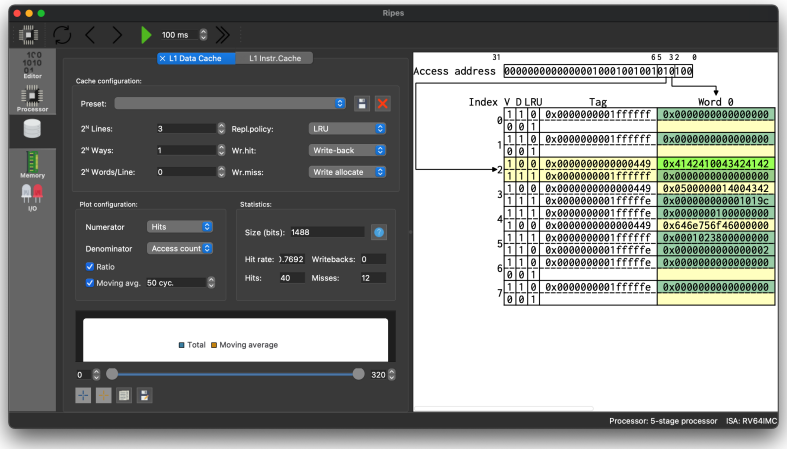


(c) After

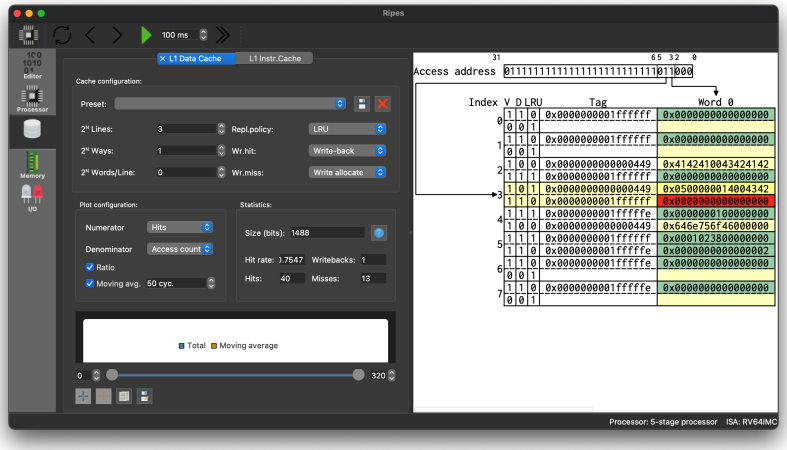
Figure 2: Write Hit



(a) The instruction



(b) Before

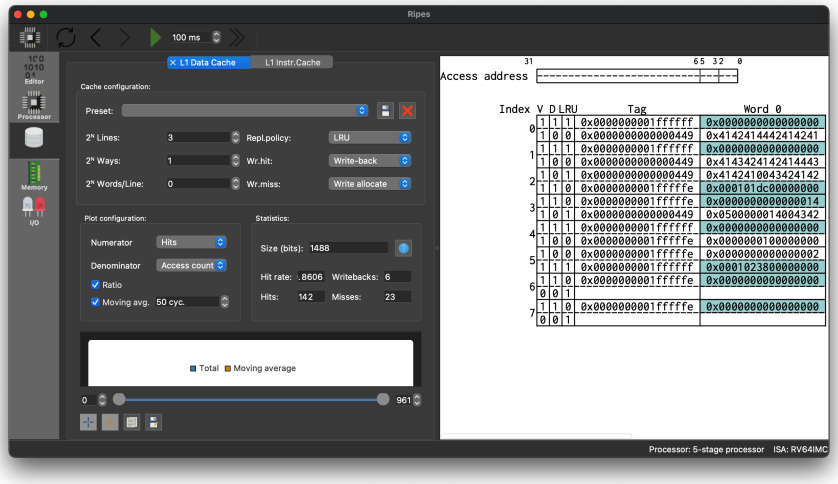


(c) After

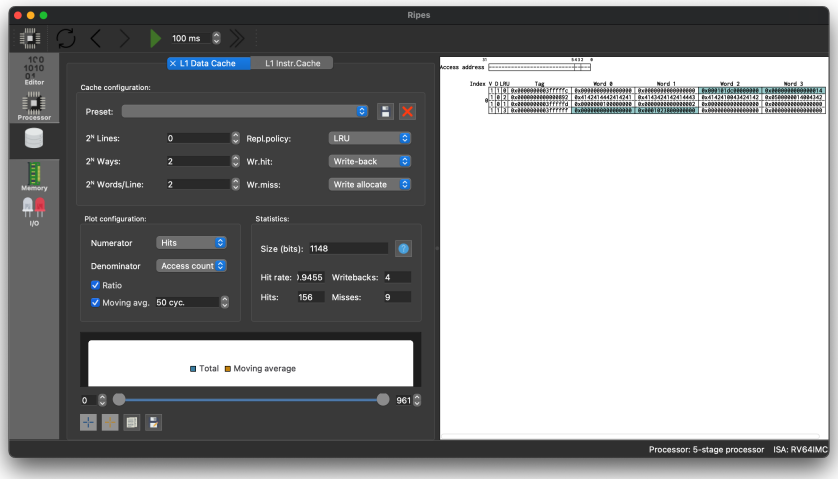
Figure 3: Write Back

2 Cache Improvement

The original hit rate was 0.8606. Taking the same size of 16 words, I decide to have 4 words per block and increase the number of ways to $2^2 = 4$, which is equivalent to fully associative. As a consequence, the spatial and temporal localities are well utilized and the hit rate becomes 0.9455.



(a) Orignal



(b) Improvement

Figure 4: Cache Improvement

3 Cache Configurations

3.1 Sizes

Assume that the machine is word-addressable and addresses are to words.

Table 1: Cache Sizes

#	Tag	Index	Offset	Total size
1	$20 - \lg 16 - \lg 1 = 16$	$\lg 16 = 4$	$\lg 1 = 0$	784
2	$20 - \lg \frac{16}{4} - \lg 4 = 16$	$\lg \frac{16}{4} = 2$	$\lg 4 = 2$	580
3	$20 - \lg 16\frac{16}{2\times 2} - \lg 2 = 17$	$\lg \frac{16}{2\times 2} = 2$	$\lg 2 = 1$	656
4	$20 - \lg 4 = 18$	0	$\lg 4 = 2$	588

The number of bits of total size is $(1+\#\text{BITS(tag)})\times\frac{16}{\text{words per block}}+16\times 32$.

3.2 Example

3.2.1 Accesses

Table 2: Access Results

Address	#2	#3	#4
16	Compulsory miss	Compulsory miss	Compulsory miss
17	Hit!!	Hit!!	Hit!!
18	Hit!!	Compulsory miss	Hit!!
19	Hit!!	Hit!!	Hit!!
20	Compulsory miss	Compulsory miss	Compulsory miss
48	Compulsory miss	Compulsory miss	Compulsory miss
49	Hit!!	Hit!!	Hit!!
17	Conflict miss	Hit!!	Hit!!
48	Conflict miss	Hit!!	Hit!!
49	Hit!!	Hit!!	Hit!!
17	Conflict miss	Hit!!	Hit!!
5	Compulsory miss	Compulsory miss	Compulsory miss
6	Hit!!	Hit!!	Hit!!
7	Hit!!	Compulsory miss	Hit!!

3.2.2 Final Contents

4 Hamming Codes

4.1 Single Error Correction

$$\begin{cases} h_1 = 1 \oplus 0 \oplus 1 \oplus 0 = 0 \\ h_2 = 1 \oplus 0 \oplus 0 \oplus 0 = 1 \\ h_4 = 1 \oplus 1 \oplus 0 \oplus 0 = 0 \end{cases}$$

, so we have $H = \{0, 1, 0\} = 2$, indicating that the second bit, $p2$, is an error.

4.2 Single Error Correction & Double Error Detection

$$\begin{cases} h_1 = 1 \oplus 1 \oplus 0 \oplus 0 = 0 \\ h_2 = 0 \oplus 1 \oplus 1 \oplus 0 = 0 \\ h_4 = 1 \oplus 0 \oplus 1 \oplus 0 = 0 \\ h_8 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1 \end{cases}$$

, so we have $H = 0$ and $p_8 = 1$, indicating that the last bit, p_8, is an error.

4.3 Yet Another Single Error Correction & Double Error Detection

$$\begin{cases} h_1 = 1 \oplus 0 \oplus 1 \oplus 1 = 1 \\ h_2 = 0 \oplus 0 \oplus 0 \oplus 1 = 1 \\ h_4 = 0 \oplus 1 \oplus 0 \oplus 1 = 0 \\ h_8 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0 \end{cases}$$

, so we have $H = \{0, 1, 1\} = 3$ but $h_8 = 0$, indicating that the double error occurred.

5 Cache Performance

5.1 Miss Penalties & Memory Stall Cycles

Table 3: Cache #2

Index	Valid	Tag	Data
00	1	1	words of memory address 16, 17, 18, 19
01	1	0	words of memory address 4, 5, 6, 7
10	0		
11	0		

5.2 Effective CPI

$$\text{Base CPI} = 1.7 - (4\% + \frac{1}{3} \times 3\%) \times (5 + 1) = 1.4$$

6 Virtual Memory

$$\#\text{blocks in cache} = \frac{64 \times 1024}{256} = 256$$

Table 4: Cache #3

Set	Valid	Tag	Data
00	1	10	words of memory address 16, 17
00	1	110	words of memory address 48, 49
01	1	10	words of memory address 18, 19
01	0		
10	1	1	words of memory address 20, 21
10	1	0	words of memory address 4, 5
11	1	0	words of memory address 6, 7
11	0		

Table 5

#	1	2
Miss Penalty	$5+4=9$	$5+2=7$
Mem. Stall Cyc.	$70000 \times (4\% + \frac{1}{3} \times 3\%) \times 9 = 31500$	$70000 \times (4\% + \frac{1}{3} \times 3\%) \times 7 = 24500$

Table 6

#	1	2
CPI	$1.4 + (4\% + \frac{1}{3} \times 3\%) \times (5+4) = 1.85$	$1.4 + (4\% + \frac{1}{3} \times 3\%) \times (5+2) = 1.75$

Table 7

Virtual address	Physical address	Cache	TLB
`0x954a16c2`	`0x3b9416c2`	Miss	Miss
`0x6542c746`	`0x0ae6c746`	Miss	Miss
`0x954a1647`	`0x3b941647`	Hit!!	Hit!!
`0x6542c412`	`0x0ae6c412`	Miss	Hit!!
`0x2b74c4d3`	`0x14acc4d3`	Miss	Miss
`0x6542c46a`	`0x0ae6c46a`	Miss	Hit!!
`0x954a16dd`	`0x3b9416dd`	Hit!!	Miss
`0x6542c417`	`0x0ae6c417`	Hit!!	Hit!!
`0x2b74c723`	`0x14acc723`	Miss	Miss