

Task1:

5 stage pipeline Simulator

Consider the following instructions. Execute these instructions using 5 stage pipeline - MIPS architecture simulator.

ADD R0, R1, R2

SUB R3, R0, R4

FP_LOAD F1, Offset, R1

FP_ADD F5,F1,F1

The screenshot displays the MIPS Five Stage Pipeline simulator interface. The top section shows the instruction list and execution cycles for four instructions: `int_add (R1, R2, R3)`, `int_sub (R4, R1, R5)`, `fp_ld (F1, Offset, R1)`, and `fp_add (F5, F1, F1)`. The execution cycles are shown in a table with columns for CPU Cycles 1 through 15. The bottom section shows the same instructions with the `Data Forwarding` checkbox checked, which resolves the hazards.

Instruction List:

Instruction	Execution Cycles
FP_Add/Sub	1
FP_Multiply	1
FP_Divide	1
INT_Divide	1

Execution Cycles Table:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 int_add (R1, R2, R3)	IF	ID	+- (I)	MEM	WB										
1 int_sub (R4, R1, R5)		IF	ID	S	S	+- (I)	MEM	WB							
2 fp_ld (F1, Offset, R1)			IF	S	S	ID	EX	MEM	WB						
3 fp_add (F5, F1, F1)						IF	ID	S	S	+- (I)	MEM	WB			

Potential Hazards:

RAW: Instructions 0 and 1. Register R1.
RAW: Instructions 0 and 2. Register R1.
RAW: Instructions 2 and 3. Register F1.

Execution Cycles Table (with Data Forwarding):

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 int_add (R1, R2, R3)	IF	ID	+- (I)	MEM	WB										
1 int_sub (R4, R1, R5)		IF	ID	+- (I)	MEM	WB									
2 fp_ld (F1, Offset, R1)			IF	ID	EX	MEM	WB								
3 fp_add (F5, F1, F1)				IF	ID	S	+- (I)	MEM	WB						

Potential Hazards:

RAW: Instructions 2 and 3. Register F1.

Q1) Check whether there is data dependency among the instructions? If yes, then, how many stall states have been introduced?

A) Yes, 6 stalls have been introduced

Q2) If data forwarding is applied how many stall states have been reduced?

A) From 4 stalls it has been reduced to 1 stall (3 stalls are avoided)

Q3) Mention the total number of clock cycles used with and without data forwarding.

A) Without Data forwarding : 12

With Data forwarding : 9

Task2:

Cache Simulator:

1. A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block.

Assume that the size of each memory word is 1 byte.

(a) Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address using direct mapped Cache.

The screenshot shows the ParaCache Cache Simulator interface. The 'DIRECT MAPPED CACHE' section is active. The 'Cache Size (power of 2)' is set to 2048, and the 'Memory Size (power of 2)' is set to 65536. The 'Offset Bits' is set to 6. The 'Cache Table' is displayed, showing the mapping of memory addresses to cache blocks. The table has columns for Index, Valid, Tag, Data (Hex), and Dirty Bit. The 'Memory Block' section shows the current state of the cache blocks, with the first block (index 0) containing the instruction '00000000'.

Index	Valid	Tag	Data (Hex)	Dirty Bit
0	0	-	0	0
1	0	-	0	0
2	0	-	0	0
3	0	-	0	0
4	0	-	0	0
5	0	-	0	0
6	0	-	0	0
7	0	-	0	0
8	0	-	0	0
9	0	-	0	0
10	0	-	0	0
11	0	-	0	0
12	0	-	0	0
13	0	-	0	0
14	0	-	0	0
15	0	-	0	0
16	0	-	0	0
17	0	-	0	0
18	0	-	0	0
19	0	-	0	0
20	0	-	0	0
21	0	-	0	0
22	0	-	0	0
23	0	-	0	0
24	0	-	0	0
25	0	-	0	0
26	0	-	0	0
27	0	-	0	0
28	0	-	0	0
29	0	-	0	0
30	0	-	0	0
31	0	-	0	0

Calculation

Total 16 bits

Words = 64 = 2^6 , 6 Bits for offset

Number of blocks = Size of cache/Size of each Block

$$2^{11}/2^6 = 2^5$$

5 Bits for index

Tag = 16-(6+5) = 5 bits

(b) When a program is executed, the processor reads data sequentially from the following word addresses:

128, 144, 2176, 2180, 128, 2176

The screenshot shows the ParaCache simulator interface. The main configuration is for a **DIRECT MAPPED CACHE** with a **Cache Size (power of 2)** of 2048 and **Offset Bits** of 6. The **Write Policy** is set to **Write Back** and **Write On Allocate**. The **Instruction Breakdown** shows three instructions: 00001 (5 bit), 00010 (5 bit), and 000100 (6 bit). The **Cache Table** is displayed with 32 rows (Index 0 to 31). The **Memory Block** section shows the current state of the cache blocks. The **Statistics** section shows a **Hit Rate** of 33% and a **Miss Rate** of 67%. The **List of Previous Instructions** shows a sequence of instructions: Load R0 (Miss), Load R0 (Hit), Load R0 (Miss), Load R0 (Hit), Load R0 (Miss), and Load R0 (Miss).

Index	Valid	Tag	Data (Hex)	Dirty Bit
0	0	-	0	0
1	0	-	0	0
2	1	00001	BLOCK 22 WORD 0 - 63	0
3	0	-	0	0
4	0	-	0	0
5	0	-	0	0
6	0	-	0	0
7	0	-	0	0
8	0	-	0	0
9	0	-	0	0
10	0	-	0	0
11	0	-	0	0
12	0	-	0	0
13	0	-	0	0
14	0	-	0	0
15	0	-	0	0
16	0	-	0	0
17	0	-	0	0
18	0	-	0	0
19	0	-	0	0
20	0	-	0	0
21	0	-	0	0
22	0	-	0	0
23	0	-	0	0
24	0	-	0	0
25	0	-	0	0
26	0	-	0	0
27	0	-	0	0
28	0	-	0	0
29	0	-	0	0
30	0	-	0	0
31	0	-	0	0

2. For the above mentioned problem, calculate and execute for 4way set associativity and fully associative mapping technique. For each technique

randomly generate ten addresses and indicate whether the cache access will result in a hit or a miss. Assume block replacement policy as random.

ParaCache

Replacement Policies
☐ FIFO ☐ LRU ☒ Random

Write Policies
☒ Write Back ☐ Write Through
☒ Write On Allocate ☐ Write Around

Cache Size (power of 2): 2048
 Memory Size (power of 2): 65536
 Offset Bits: 6

Reset Submit

Instruction
 Load [in hex] []
 List of next 10 Instructions
 Submit

Information
 The cycle has been completed.
 Please submit another instructions

Next Fast Forward

Statistics
 Hit Rate : 20%
 Miss Rate : 80%

List of Previous Instructions :
 • Load F284 (Miss)
 • Load 8517 (Miss)
 • Load A7A3 (Miss)
 • Load F30B (Miss)
 • Load 050F (Miss)
 • Load 6901 (Miss)
 • Load 602B (Miss)
 • Load 9287 (Miss)
 • Load 602B (Hit)
 • Load 602B (Hit)

4-WAY SET ASSOCIATIVE CACHE

Instruction Breakdown

0110000	000	110000
7 bit	3 bit	6 bit

Cache Table

Index/Valid	Tag	Data (Hex)	Dirty Bit
0	1	30	B: 180 W: 0 - 63
1	0	-	0
2	0	-	0
3	0	-	0
4	0	-	0
5	0	-	0
6	1	79	B: 3CE W: 0 - 63
7	0	-	0

Memory Block

Index/Valid	Tag	Data (Hex)	Dirty Bit
0	0	-	0
1	0	-	0
2	0	-	0
3	0	-	0
4	0	-	0
5	0	-	0
6	1	5	B: 2E W: 9 - 63
7	0	-	0

Calculation

Total 16 bits

Words = 64 = 2^6 , 6 Bits for offset

Number of Sets = Size of cache/Size of each Block/ k-set

$$2^{11}/(2^6 * 2^2) = 2^3$$

3 Bits for index

Tag = 16-(6+3) = 7 bits

ParaCache

Replacement Policies
☐ FIFO ☐ LRU ☒ Random

Write Policies
☒ Write Back ☐ Write Through
☒ Write On Allocate ☐ Write Around

Cache Size (power of 2): 2048
 Memory Size (power of 2): 65536
 Offset Bits: 6

Reset Submit

Instruction
 Load [in hex] []
 List of next 10 Instructions
 Submit

Information
 The cycle has been completed.
 Please submit another instructions

Next Fast Forward

Statistics
 Hit Rate : 30%
 Miss Rate : 70%

List of Previous Instructions :
 • Load 5245 (Miss)
 • Load 2A44 (Miss)
 • Load F3C4 (Miss)
 • Load 5245 (Hit)
 • Load 444F (Miss)
 • Load 1A0 (Miss)
 • Load 2A44 (Hit)
 • Load 9C88 (Miss)
 • Load 5005 (Miss)
 • Load 1A0 (Hit)

FULLY ASSOCIATIVE CACHE

Instruction Breakdown

0000000110	100110
10 bit	6 bit

Cache Table

Index	Valid	Tag	Data (Hex)	Dirty Bit
0	0	-	0	0
1	1	1001110110	BLOCK 276 WORD 0 - 63	0
2	0	-	0	0
3	0	-	0	0
4	0	-	0	0
5	0	-	0	0
6	0	-	0	0
7	1	0101110110	BLOCK 176 WORD 0 - 63	0
8	0	-	0	0
9	0	-	0	0
10	0	-	0	0
11	1	0000000110	BLOCK 6 WORD 0 - 63	0
12	0	-	0	0
13	0	-	0	0
14	0	-	0	0
15	0	-	0	0
16	1	0101000000	BLOCK 148 WORD 0 - 63	0
17	0	-	0	0
18	1	111100011	BLOCK 363 WORD 0 - 63	0
19	0	-	0	0
20	0	-	0	0
21	0	-	0	0
22	0	-	0	0
23	0	-	0	0
24	0	-	0	0
25	0	-	0	0
26	0	-	0	0
27	0	-	0	0
28	0	-	0	0
29	0	-	0	0
30	0	-	0	0
31	1	0100010001	BLOCK 111 WORD 0 - 63	0

Calculation

Total 16 bits

Words = 64 = 2^6 , 6 Bits for offset

Tag = $16 - 6 = 10$ bits

Task3:

Use Arm Simulator

Given a 'c' code convert it in its equivalent Arm Code and execute in ARM

simulator

1) $x = (a + b) - c$

The screenshot shows the ARMSim ARM simulator interface. The main window displays assembly code for an ARM processor. The code is as follows:

```
.data
0000103C:      a: word 5
00001040:      b: word 8
00001044:      c: word 1
00001048:      x: word 0

.text
00001000: 259F0024  LDR R0,=a
00001004: 259F0000  LDR R0,[R0] ;Loads value from a
00001008: E3A01D41  LDR R1,=b
0000100C: 25911000  LDR R1,[R1] ;Loads value from b
00001010: 259F2018  LDR R2,=c
00001014: 25922000  LDR R2,[R2] ;Loads value from c
00001018: 259F3014  LDR R3,=x
0000101C: E0805001  ADD R5,R0,R1
00001020: E0454002  SUB R4,R5,R2
00001024: 25834000  STR R4,[R3] ;Stores the result in x
SWI 0x01
```

The left pane shows the Register View with the following values:

Register	Value
R0	00000005
R1	00000008
R2	00000001
R3	00001048
R4	0000000c
R5	0000000d
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10 (s1)	00000000
R11 (fp)	00000000
R12 (ip)	00000000
R13 (sp)	00005400
R14 (lr)	00000000
R15 (pc)	00011400

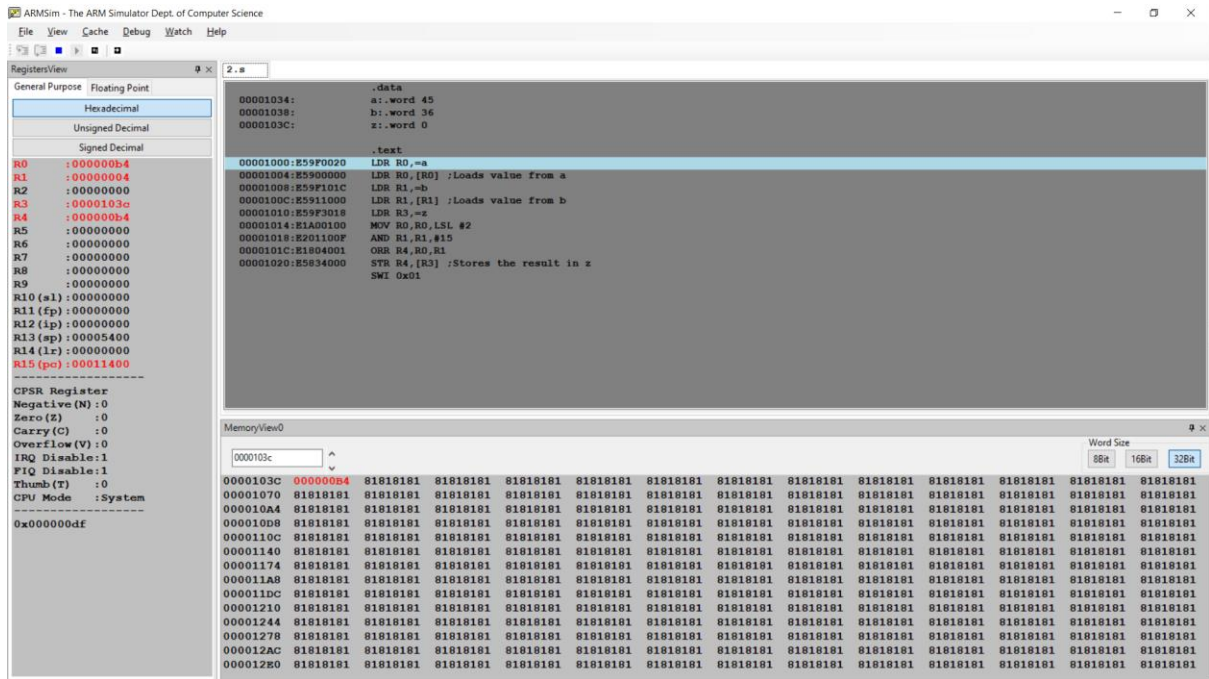
The bottom pane shows the Memory View with the following values:

Address	Value
00001048	0000000c
0000107C	81818181
00001080	81818181
00001084	81818181
00001118	81818181
0000114C	81818181
00001180	81818181
000011B4	81818181
000011E8	81818181
0000121C	81818181
00001250	81818181
00001284	81818181
000012B8	81818181
000012EC	81818181

Calculation

$X = (5 + 8) - 1 = 12$ (0xC)

2) $z = (a \ll 2) | (b \& 15)$



Calculation

$$45 \cdot 2^2 = 180$$

$$36 \& 15 = 100100 \& 001111 = 000100 = 4$$

$$180 \mid 4 = 10110100 \mid 00000100 = 10110100 = B4$$