Bryant Chapter 4

4.13 - correct

Stage	Generic `irmovq` V, rB	Specific `irmovq $128, %rsp`
Fetch	icode:ifun $\leftarrow M_1$[PC]	icode:ifun $\leftarrow M_1$[0x016] = 0x3:0x0
	rA:rB $\leftarrow M_1$[PC+1]	rA:rB $\leftarrow M_1$[0x017] = 0xF:0x4
	valC $\leftarrow M_8$[PC+2]	valC $\leftarrow M_8$[0x018] = 0x80
	valP $\leftarrow$ PC + 10	valP $\leftarrow$ 0x016 + 0xA = 0x020
Decode		
Execute	valE $\leftarrow$ 0 + valC	valE $\leftarrow$ 0x0 + 0x80 = 0x80
Memory		
Write back	R[rB] $\leftarrow$ valE	R[%rsp] $\leftarrow$ 0x80
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP = 0x020

4.14 - correct

Stage	Generic `popq` rA	Specific `popq %rax`
Fetch	icode:ifun $\leftarrow M_1$[PC]	icode:ifun $\leftarrow M_1$[0x02C] = 0xB:0x0
	rA:rB $\leftarrow M_1$[PC+1]	rA:rB $\leftarrow M_1$[0x02D] = 0x0:0xF
	valP $\leftarrow$ PC + 2	valP $\leftarrow$ 0x02C + 0x2 = 0x02E
Decode	valA $\leftarrow$ R[%rsp]	valA $\leftarrow$ R[%rsp] = 120
	valB $\leftarrow$ R[%rsp]	valB $\leftarrow$ R[%rsp] = 120
Execute	valE $\leftarrow$ valB + 8	valE $\leftarrow$ 120 + 8 = 128
Memory	valM $\leftarrow M_8$[valA]	valM $\leftarrow M_8$[120] = 9
Write back	R[%rsp] $\leftarrow$ valE	R[%rsp] $\leftarrow$ 128
	R[rA] $\leftarrow$ valM	R[%rax] $\leftarrow$ 9
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ 0x02E

4.43 - correct

Cause	Name	Instruction frequency	Condition frequency	Bubbles	Product
Load/use	*lp*	0.25	0.20	1	0.05
Mispredict	*mp*	0.20	0.35	2	0.14
Return	*rp*	0.02	1.00	3	0.06
Total penalty					0.25

Thus, the CPI is $1 + 0.25 = 1.25$.

Bryant Chapter 6

6.12 - correct

Set index: 3 bits since 8 sets $\rightarrow \log_2 8 = 3$ bits
Block offset: 2 bits since 4-byte block size $\rightarrow \log_2 4 = 2$ bits
Tag: 13 - (3 + 2) = 8 bits

CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CI	CO	CO
12	11	10	9	8	7	6	5	4	3	2	1	0

6.17 - correct

A.

Given the following properties:

- `sizeof(int) = 4`
- `src` is at address 0
- `dst` is at address 16
- One L1 cache - direct mapped, write-through, write-allocate, 8-byte blocks
- Cache has 16 bytes total, is initially empty
- Accesses to `src` and `dst` are the only sources of hits and misses

Set index: Direct mapped $\rightarrow E = 1$, so $\log_2 \frac{16}{8} = 1$ bit
Block offset: 8-byte blocks $\rightarrow \log_2 8 = 3$ bits
Memory addresses will have the form of:

...	CT	CI	CO	CO	CO
...	4	3	2	1	0

Thus,

Array element	Decimal address	Binary address	Tag	Index	Offset
`src[0][0]`	0	...00000	0	0	000
`src[0][1]`	4	...00100	0	0	100
`src[1][0]`	8	...01000	0	1	000
`src[1][1]`	12	...01100	0	1	100
`dst[0][0]`	16	...10000	1	0	000
`dst[0][1]`	20	...10100	1	0	100
`dst[1][0]`	24	...11000	1	1	000
`dst[1][1]`	28	...11100	1	1	100

So the cache will have this kind of history:

Set	Tag + Array access
0	1: dst[0][1]
	~~0: src[0][1]~~
	~~1: dst[0][0]~~
	~~0: src[0][0]~~
1	1: dst[1][1]
	~~0: src[1][1]~~ **hit**
	~~0: src[1][0]~~
	~~1: dst[1][0]~~

`dst` array			`src` array		
	Col. 0	Col. 1		Col. 0	Col. 1
Row 0	m	m	Row 0	m	m
Row 1	m	m	Row 1	m	h

B.

Now, with a cache size of 32 bytes, the cache will have 4 sets.
Set index: 2 bits since $\log_2 \frac{32}{8} = 2$ bits
Block offset: Still 3 bits since 8-byte blocks $\rightarrow \log_2 8 = 3$ bits
Thus, memory addresses will have the form of:

...	CT	CI	CI	CO	CO	CO
...	6	5	4	3	2	1

Therefore,

Array element	Decimal address	Binary address	Tag	Index	Offset
`src[0][0]`	0	...000000	0	00	000
`src[0][1]`	4	...000100	0	00	100
`src[1][0]`	8	...001000	0	01	000
`src[1][1]`	12	...001100	0	01	100
`dst[0][0]`	16	...010000	0	10	000
`dst[0][1]`	20	...010100	0	10	100
`dst[1][0]`	24	...011000	0	11	000
`dst[1][1]`	28	...011100	0	11	100

So the cache will have this kind of history:

Set	Tag + Array access
0	0: src[0][1] **hit** 0: src[0][0]
1	0: src[1][1] **hit** 0: src[1][0]
2	0: dst[0][1] **hit** 0: dst[0][0]
3	0: dst[1][1] **hit** 0: dst[1][0]

`dst` array			`src` array		
	Col. 0	Col. 1		Col. 0	Col. 1
Row 0	m	h	Row 0	m	h
Row 1	m	h	Row 1	m	h

6.18 - correct

A.

Each array element is read twice, so $32 \times 32 \times 2 = 2048$ read operations.

B.

Each block consists of two structs, so each access will alternate in hit/miss. Therefore, there should be 1024 cache misses.

C.

The miss rate is 50%