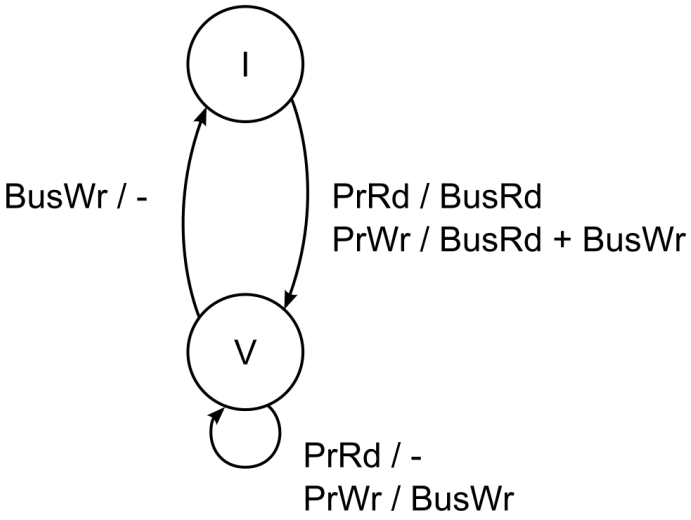
[Exercise 1] Cache Coherence

Consider a shared memory system with two processors, each one with a single level of cache. The data caches are completely separated from the instruction caches. In the following, we will completely ignore the instruction caches. To ensure cache coherence, the caches rely on a *snooping* protocol.

Each data cache is a direct-mapped, 1 MB cache. Each block contains 16 words; words are of 32 bits. The caches are word-addressed.

a) Assume an extremely simple snooping protocol, whose state diagram is shown in the following figure:



Legend:
event / action from the cache controller

- PrRd: The processor wants to read a cache line.
- PrWr: The processor wants to write a cache line.
- BusRd: A read transaction occurs on the bus.
- BusWr: A write transaction occurs on the bus.

i) According to the state diagram, are the caches:

1) *write-through* or *write-back*?

2) *write-allocate* or *not write-allocate* (i.e., in case of write miss, is the data placed in the cache or only in memory)?

Justify your answers.

ii) Simulate the following sequence of memory accesses (P1 means that the memory access has been performed by processor 1, all addresses are given in hexadecimal format and caches are empty at the beginning of the execution of the program).

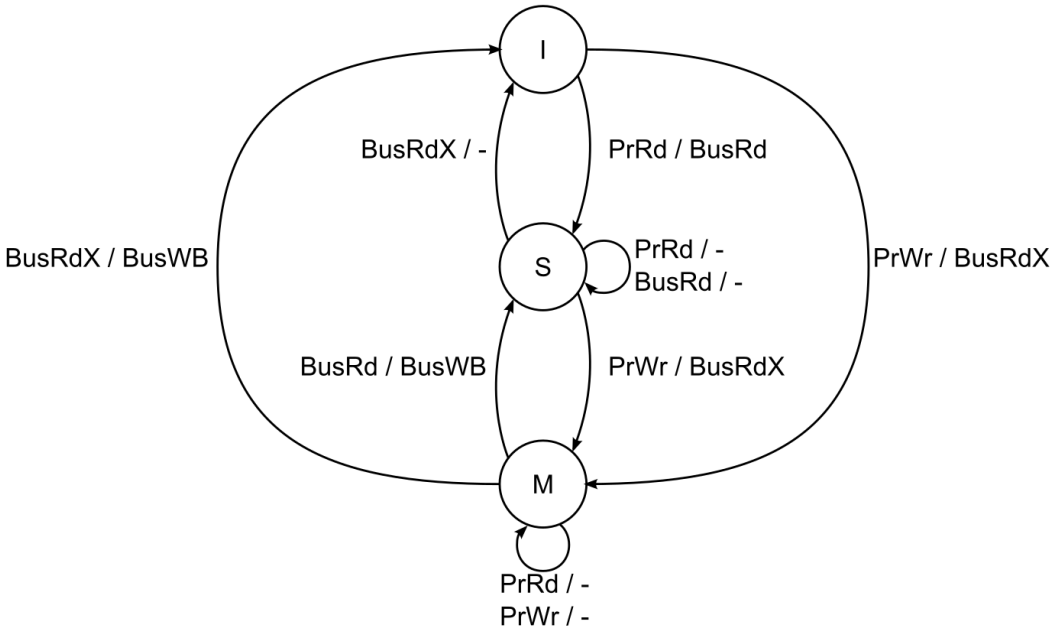
1) P1: write to 0x2000
2) P1: read from 0x2001
3) P2: write to 0x1020
4) P2: write to 0x2008
5) P1: read from 0x2000
6) P1: write to 0x4010
7) P2: read from 0x2008
8) P2: write to 0x2008
9) P2: read from 0x4010
10) P2: read from 0x1020
11) P2: write to 0x1020
12) P1: write to 0x4010
13) P1: write to 0x2001
14) P2: read from 0x2008

For each access, fill in the template on the last page of the exercise by indicating whether it is a *hit* or a *miss*, writing the line that is concerned, the previous state, the new state of each cache, and the bus transactions. For example, in the case of the first access:

miss, line XXX, P1: I $\rightarrow$ V, P2: no change, BusRd+BusWr

iii) Give the final state of all the lines that have been involved in some access.

b) Consider now a more sophisticated cache controller which implements the following state diagram:



The bus transaction BusRdX is similar to BusRd except that it also indicates the intention to modify the data after it has been read.

i) According to the state diagram, are the caches:

   1) *write-through* or *write-back*?
   2) *write-allocate* or *not write-allocate*?

   Justify your answers.

ii) Simulate the same sequence of memory accesses that was listed in question **a)**.

iii) Did the number of misses change? If yes, which kinds of misses appeared or disappeared? Did the number of bus transactions change? If yes, which transactions appeared or disappeared? Explain qualitatively what changed from **a)** to **b)**.

iv) Would a change in the cache associativity result in an improvement of the performance of the system, only based on the access sequence you simulated? If yes, clearly explain why and provide the minimum associativity required to improve the performance for this sequence of accesses.

v) Would a change in the size of the cache lines result in an improvement of the performance of the system, only based on the access sequence you simulated? If yes, clearly explain why and provide the minimum cache line size required to improve the performance of the cache or to make it optimal, for this sequence of memory accesses.

hit/miss	line	P1	P2	transactions

hit/miss	line	P1	P2	transactions

[Exercise 2] Firefly Coherence Protocol

Consider a four-processor system where each processor has its own cache C_0 to C_3, respectively. The **Firefly coherence protocol** is used, such that each cache line can be in one of the three states: *Exclusive*, *Shared*, and *Modified*. The transition diagram of Figure 1 shows the transitions from each state depending on the operations performed by the processor or induced by the bus.

A cache line in the *Exclusive* state means that this cache has the only copy of the line and that the data present in main memory is correct. The *Shared* state indicates that the cache line has the correct copy of the data and that it exists also in other caches. A line in the *Modified* state means that the cache has the only copy of the data and that the value stored in main memory is incorrect.

To apply this protocol, a special bus line (referred to as `sh` in the transition diagram) is introduced to detect if a cache line is shared among several caches. For instance, when a cache line is in the *Shared* state, a processor-write operation results in a bus-write. Once the value is written on the bus, the other caches respond by asserting `sh` if they have a copy of the same cache line. Then, depending on the value of `sh`, the state transition is decided. Writing to a cache line in *Shared* state causes the data to be broadcast over the bus, updating its value in main memory and in all the other caches which share this same cache line.

The state of a line that is being read/written for the first time into a cache (or the first time after eviction) is defined according to the diagram of Figure 1 by the transitions that have no source state.

Assume that all the caches are **direct-mapped** and initially empty.

a) Consider the following cache accesses executed in the given order and on the specified processors. Each access is expressed in terms of the cache line such that, for example, `P0: Read 01` means that processor P_0 reads the line 1 of its cache C_0.

```
1.  P0: Read  04
2.  P2: Read  04
3.  P1: Read  02
4.  P1: Write 02
5.  P3: Read  03
6.  P0: Write 04
7.  P3: Write 05
8.  P1: Read  02
9.  P2: Read  01
10. P1: Write 06
```

```
11. P0: Read  06
12. P1: Write 06
13. P2: Write 01
14. P3: Read  01
15. P2: Read  01
```

In the provided template, write the state transitions of each cache line. Make sure that you respect the following guidelines:

- Each state is indexed by the instruction that caused the transition. For example, if instruction i causes a transition to the *Modified* state, it should be represented as: $\rightarrow \mathrm{M}_i$.
- A transition from *Exclusive* to *Modified* to *Shared* should be represented as: $\mathrm{E}_i \rightarrow \mathrm{M}_j \rightarrow \mathrm{S}_k$.
- If a cache line is in the *Shared* state due to instruction i, and then instruction j causes a transition but to the same state, this transition should be represented as: $\mathrm{S}_i \rightarrow \mathrm{S}_j$.

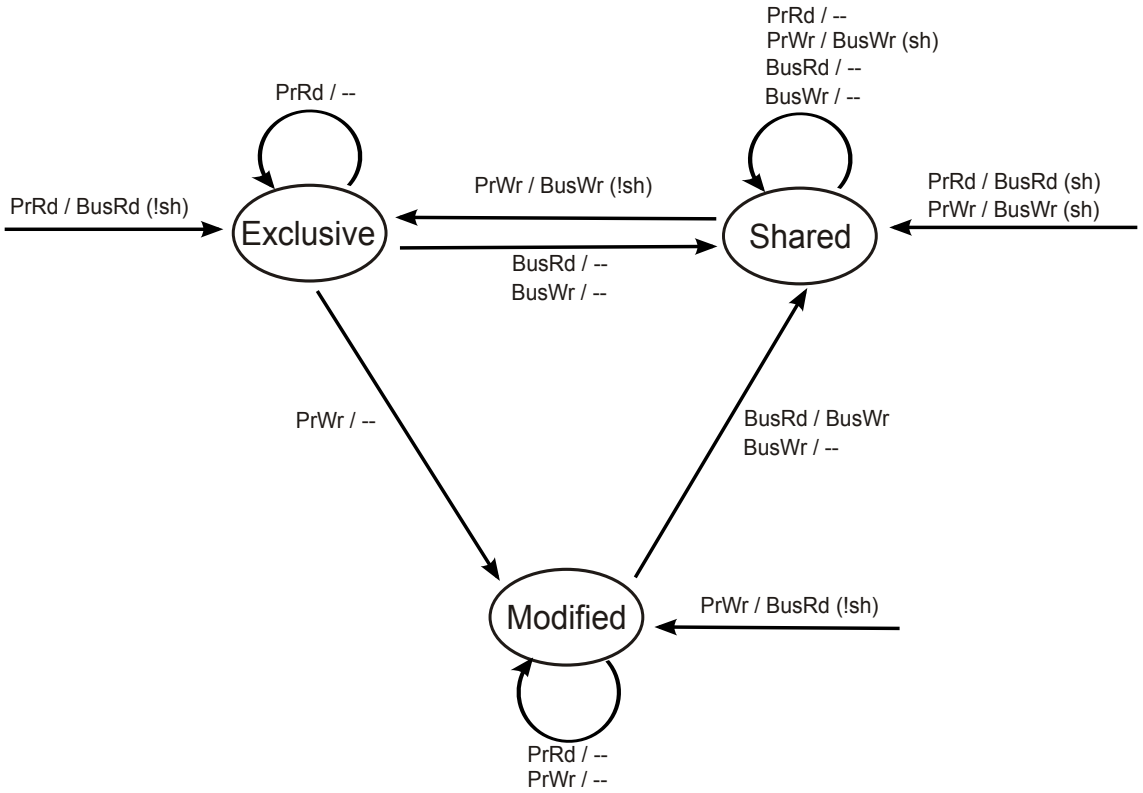


Figure 1: Firefly state transition diagram.

b) Explain why a cache line cannot move from a *Shared* state to a *Modified* state.

c) Contrary to typical coherence protocols (such as MSI, MESI, ...), the Firefly protocol does not have an Invalid state. Why is the Invalid state not needed in this protocol? What can be the main disadvantage of the Firefly protocol?

Cache line	Cache C0	Cache C1	Cache C2	Cache C3
01				
02				
03				
04				
05				
06				

Figure 2: Cache state transition template.