1. MESI State Transitions

Describe what happens in the MESI protocol (bus traffic, state changes) if a processor experiences

1.1 a local read miss, while no other processor has the requested value cached

This processor requests data from memory.

cache state: Exclusive

1.2 a local read miss, while another cache holds a copy in exclusive state

This processor requests data from memory, other processor sees the request.

This cache -> Shared

Another cache ->shared

1.3 a local read miss, while another cache has a copy in modified state

This processor requests data from memory, other processor sees the request and write back the modified cache.

This cache state : shared

Another cache, write back, modified-> shared

1.4 a local write hit, while the cacheline is in modified state

State: modified

Value is updated

1.5 a local write hit, while the cacheline is in exclusive state

State : exclusive ->modified

Value is updated.

1.6 a local write hit, while the cache line is in shared state in several caches

Processor broadcasts an invalidate message on the bus.

This cache state: modified

Other caches state: invalid

Local cache: update value

2. Cache coherence

In the following we will assume we are working with a bus-based multiprocessor machine with four processors. Each processor has its own direct-mapped cache. Each cache is 32 Bytes in size,

organized in four sets. The memory in the machine is byte-addressable and the address width of the machine is one byte. 4 组, 4 处理器,

The machine described above uses the MESI protocol to maintain cache coherence. In the following Rp(a) means that processor p reads one byte from the address a. Wp(a) = v means processor p writes v to the memory location a. Addresses are represented in binary. Initially all cache lines are invalid and all memory locations contain 0. Show the transitions of cache lines in the table below.

Action	P0	P1	P2	P3	Latency
R0 (00000000)	Exclusive	invalid	invalid	invalid	8
R1 (00000010)	Shared	Shared	Invalid	Invalid	8(fetch)
R2 (00000011)	Shared	Shared	Shared	Invalid	8(fetch)
W3(00000010) = 1	Invalid	Invalid	Invalid	Modified	2(P3I 变成 m)+8(P3fetch)
W3(00000101) = 2	Invalid	Invalid	Invalid	Modified	1(P3write hit)
R0 (00000100)	Shared	Invalid	Invalid	Shared	8(P3write back)+8(P0 等 P3 写回后 fetch)
W0(00000101) = 3	Modified	Invalid	Invalid	Invalid	2(S 变成 M)+1(cache hit) 因为都是 P0 所以时间相加

Assume a cache hit takes one cycle (both for read and write operations), transferring a cache line (either fetch or write back) takes 8 cycles, and the transition from I or S to M takes 2 cycles. What is the penalty of this code sequence over the best alternative execution sequence that you can think of?

8+8+8+10+1+16+3=54. In total this takes 54 cycles.

Action	P0	P1	P2	P3	Latency
R0 (00000000)	Exclusive	invalid	invalid	invalid	8
R0(00000100)	Exclusive	invalid	Invalid	Invalid	1(利用 cache 时间局部性)
W0(00000101) = 3	Modified	invalid	invalid	Invalid	1
R1(00000010)	Shared	Shared	Invalid	Invalid	8+8
R2(00000011)	Shared	Shared	Shared	Invalid	8
W3(00000100) = 1	Invalid	Invalid	Invalid	Modified	2(P3I 变成 m)+8(P3fetch)
W3(00000101) = 2	Invalid	Invalid	Invalid	Modified	1

8+1+1+16+8+10+1=45

R0(00000000), R0(00000100), W0(00000101) = 3, R1(00000010), R2(00000011), W3(00000100) = 1, W3(00000101) = 2 it takes only 45 cycles.

都用同一个 processor,可以减少其他处理器 Transition 的时间,可以减少 write back 的概率

3. False sharing

The listed code is executed on a bus-based multiprocessor with 2 processors that uses the MESI protocol to maintain cache coherence. Assume the array x takes up an entire cacheline of 64 Byte.

```
int x[16] ;
int tid ;
#pragma omp parallel private (tid ) shared (x)
{
    tid = omp_get_thread_num( ) ;
    for (int i = 0; i <8; i ++) {
    if (tid == 0) x[i*2+ tid ] = 1 ;
    if (tid == 1) x[i*2+ tid ] = 2 ;
    }
}
```

Assume that memory accesses of the above program are interleaved 交错 in a round-robin fashion. Assuming the same access penalties as in the previous exercise what is the total memory access latency encountered by the above program? How could one improve the situation?

Action	P0	P1	Latency
W0 (00000000)=1	modified	invalid	2+8(transition 和 fetch)
W1(00000001)= 2	invalid	modified	2+8+8(write back, transition 和 fetch)
W0(00000010) = 1	Modified	invalid	2+8+8(write back, transition 和 fetch)
W1(00000011) = 2	invalid	modified	2+8+8(write back, transition 和 fetch)
W0(00000100) = 1	modified	invalid	2+8+8(write back, transition 和 fetch)
W1(00000101) = 2	Invalid	modified	2+8+8(write back, transition 和 fetch)

Each write operation of the processor will send an invalid signal.

我们应该让并行线程处理距离比较远的数据,也就是不要访问地址相近的内存. 或者重排序

4. Directory-based Cache Coherence Update Protocols

Ben wants to convert the directory-based invalidate cache coherence protocol from the handout into an update protocol. He proposes the following scheme. Ben 希望将基于目录的无效缓存一致性协议从讲义转换为更新协议。 他提出以下方案。一个 site 应该就是一个 processor

Caches are write-through 写到 cache 和内存, not write allocate(miss 了写到内存). When a processor wants to write to a memory location, it sends a WriteReq to the memory, along with the data word that it wants written. The memory processor updates the memory, and sends an UpdateReq(通知其它 site) with the new data to each of the sites caching the block, unless that site is the processor performing the store, in which case it sends a WriteRep(回复) containing the new data. 站点 site 就是 各个 processor

If the processor performing the store is caching the block being written, it must wait for the reply from the home site(什么是 home site?) to arrive before storing the new value into its cache. If the processor performing the store is not caching the block being written, it can proceed after issuing the WriteReq. 如果执行存储的处理器(这是啥)正在对正在写入的块进行缓存，则在将新值存储到其缓存之前，它必须等待来自主站点的答复到达。 如果执行存储的处理器未缓存正在写入的块，则可以在发出 WriteReq 之后继续进行。

Ben wants his protocol to perform well, and so he also proposes to implement silent drops. When a cache line needs to be evicted, it is silently evicted and the memory processor is not notified of this event.

Note that WriteReq and UpdateReq contain data at the word-granularity 字粒度, and not at the block-granularity. Also note that in the proposed scheme, memory will always have the most up-to-date data and the state C-exclusive is no longer used 方案中，内存将始终具有最新数据，不用 C-exclusive 状态

As in the lecture, the interconnection network guarantees that message-passing is reliable, and free from deadlock, livelock 活锁是加不上就放开已获得的资源重试此时 P1 放弃占有 A 重新开始，P2 放弃占有 B 重新开始。则 P1、P2 可能会出现重复不断的开始-回滚循环。这种情况我们称之为**活锁**, and starvation 当某个进程无限期地延迟时会发生饥饿，因为总是优先考虑其他进程。在饥饿期间，至少有一个涉及的线程保留在就绪队列中。. Message-passing is FIFO. Each home site keeps a FIFO queue of incoming requests, and processes these in the order received.

4.1 Alyssa claims that Ben's protocol does not preserve sequential consistency because it allows two processors to observe stores in different orders. Describe a scenario in which this problem can occur.

The memory processor A 收到 writeReq1 要求 block1，The memory processor B 收到 writeReq2 要求 block2

A 发送 UpdateReq 给 cacheC 和 D,C 和 D 都同时有 block1 和 block2.

C 先收到 block1 update, D 先收到 block2 update. 所以他们 observe stores in different orders

4.2 Noting that many commercial systems do not guarantee sequential consistency, Ben decides to implement his protocol anyway. Fill in the following state transition tables (Table 4-1 and Table P4-2) for the proposed scheme. (Note: the tables do not contain all the transitions for the protocol).

No.	Current State	Event Received	Next State	Action
1	C-nothing	Load	C-transient	ShReq(id, Home, a)
2	C-nothing	Store	C-transient	WriteReq(id,Home,a)发送写请求
3	C-nothing	UpdateReq	C-nothing	None
4	C-shared	Load	C-shared	processor reads cache
5	C-shared	Store	C-transient	WriteReq(id,home,a) 发送写请求
6	C-shared	UpdateReq	C-shared	Data->cache
7	C-shared	(Silent drop)	C nothing	Nothing
8	C-transient	ShRep	C-shared	data → cache, processor reads cache
9	C-transient	WriteRep	C-shared	Data->cache
10	C-transient	UpdateReq	C-transient	Data->cache

Table P4-1: Cache State Transitions

No.	Current State	Message Received	Next State	Action
1	R(dir) & id ∉ dir	ShReq	R(dir + {id})	ShRep(Home, id, a) shared
2	R(dir) & id ∉ dir	WriteReq	R(dir + {id})	Data->memory,
3	R(dir) & id ∈ dir	ShReq	R(dir)	ShRep(Home, id, a)
4	R(dir) & id ∈ dir	WriteReq	R(dir)	Data->memory,

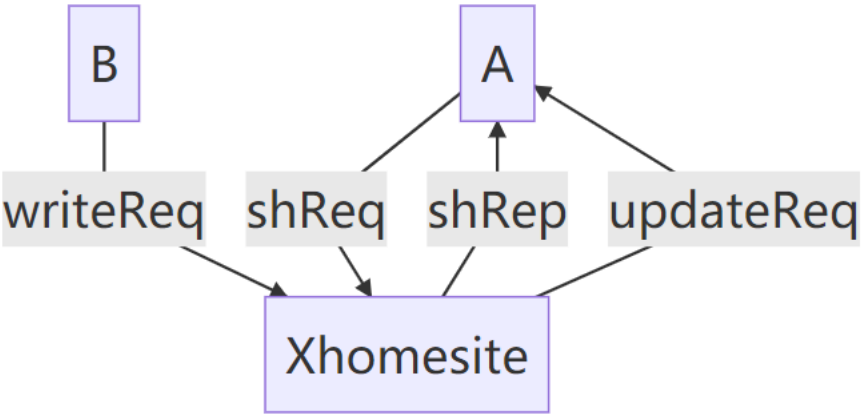
Table P4-2: Home Directory State Transitions (N = “is not in”)
R(dir) 是说 这个内存 line is shared by the sites specified in dir(dir 是 site 的集合). Data in memory 是有效的,

4.3 After running a system with this protocol for a long time, Ben finds that the network is flooded with UpdateReqs. Alyssa says this is a bug in his protocol. What is the problem and how can you fix it? 在使用该协议运行了很长时间的系统后，Ben 发现网络上充斥着 UpdateReqs。 阿丽莎说这是他协议中的错误。 有什么问题，如何解决?

因为替换行时 cache 不会通知主站点，所以存储块的集合 S 不断增加。 最终，每个曾经加载了特定块的站点都将在该块的集合 S 中，从而导致该块为了通知其它所有 site, 会发送许多 updateReq, 网络上出现许多 UpdateReq.即使许多更新的接收者已经替换了该缓存行。

解决方案是在替换缓存行时通知主站点，并在收到此类通知时让主站点从 S 中删除站点。

4.4 FIFO message passing is a necessary assumption for the correctness of the protocol. If the network were non-FIFO, it becomes possible for a processor to never see the result of another processor's store. Describe a scenario in which this problem can occur.



1.站点 A 将 X 块的 ShReq 发送到 X 的本地站点。

2. X 的主站点接收 ShReq 并发出 ShRep。

3.站点 B 向 X 的本地站点发送针对 X 块的 WriteReq。

4. X 的 home site 接收 WriteReq，并向 A 发出 UpdateReq。

5.在网络中 ShReq 和 UpdateReq 顺序变了。

6. UpdateReq 到达 A。由于 A 在等待 ShRep，状态为 C-transient。 它使用 UpdateReq 数据更新其缓存 ,依旧是 C-transient.

7. ShRep 到达 A。将过时的数据写入其 cache，然后读取结果。尽管 A 收到了 UpdateReq，但除非 replace cache line and reload，否则它将永远看不到 B 的存储结果。

5. Sequential Consistency

For this problem we will be using the following sequences of instructions. These are small programs, each executed on a different processor, each with its own cache and register set. In the following R is a register and X is a memory location. Each instruction has been named (e.g., B3) to make it easy to write answers.

Assume data in location X is initially 0.

Processor A	Processor B	Processor C
A1: ST X, 1	B1: R := LD X	C1: ST X, 6
A2: R := LD X	B2: R := ADD R, 1	C2: R := LD X
A3: R := ADD R, R	B3: ST X, R	C3: R := ADD R, R
A4: ST X, R	B4: R:= LD X	C4: ST X, R
	B5: R := ADD R, R	
	B6: ST X, R	

For each of the questions below, please circle the answer and provide a short explanation assuming the program is executing under the SC model. **No points will be given for just circling an answer!**

5.1 Can X hold value of 4 after all three threads have completed? Please explain briefly.

Yes, C1. B1->B6, X =14

A1->A4, X =2

C2->C4, X = 4

5.2 Can X hold value of 5 after all three threads have completed?

No 因为有 3 个 add r r，操作有一个 1, 一个 6, 3 个翻倍,一个加 1 , 不可能得到 5.

5.3 Can X hold value of 6 after all three threads have completed?

Yes, C1->C4

A1->A4 X 为 2,

B1->B6 X 为 6

5.4 For this particular program, can a processor that reorders instructions but follows local dependencies produce an answer that cannot be produced under the SC model?

No, 存和读取会顺序执行因为是一个地址.

6. Cache coherence: MESI

Assume you are designing a MESI snoopy bus cache coherence protocol for write-back private caches in a multi-core processor. Each private cache is connected to its own processor core as well as a common bus that is shared among other private caches.

There are 4 input commands a private cache may get for a cacheline. Assume that bus commands and core commands will not arrive at the same time:

- CoreRd: Read request from the cache ś core
- CoreWr: Write request from the cache ś core
- BusGet: Read request from the bus
- BusGetI: Read and Invalidate request from the bus (invalidates shared data in the cache that receives the request) 把自己的 data 无效化

There are 4 actions a cache can take while transferring to another state:

- Flush: Write back the cacheline to lower-level cache
- BusGet: Issue a BusGet request to the bus
- BusGetI: Issue a BusGetI request to the bus
- None: Take no action

There is also an "is shared (S)" signal on the bus. S is asserted upon a BusGet request when at least one of the private caches shares a copy of the data (BusGet (S)). Otherwise S is deasserted (BusGet (not S)). 总线上还有一个"共享（S）"信号。 当至少一个私有缓存共享数据副本（BusGet（S））时，将在 BusGet 请求时声明 S。 否则，S 被置为无效（BusGet（不是 S））。其实我不太懂 getI 是啥意思

Assume upon a BusGet or BusGetI request, the inclusive lower-level cache will eventually supply the data and there is no private cache to private cache transfers.

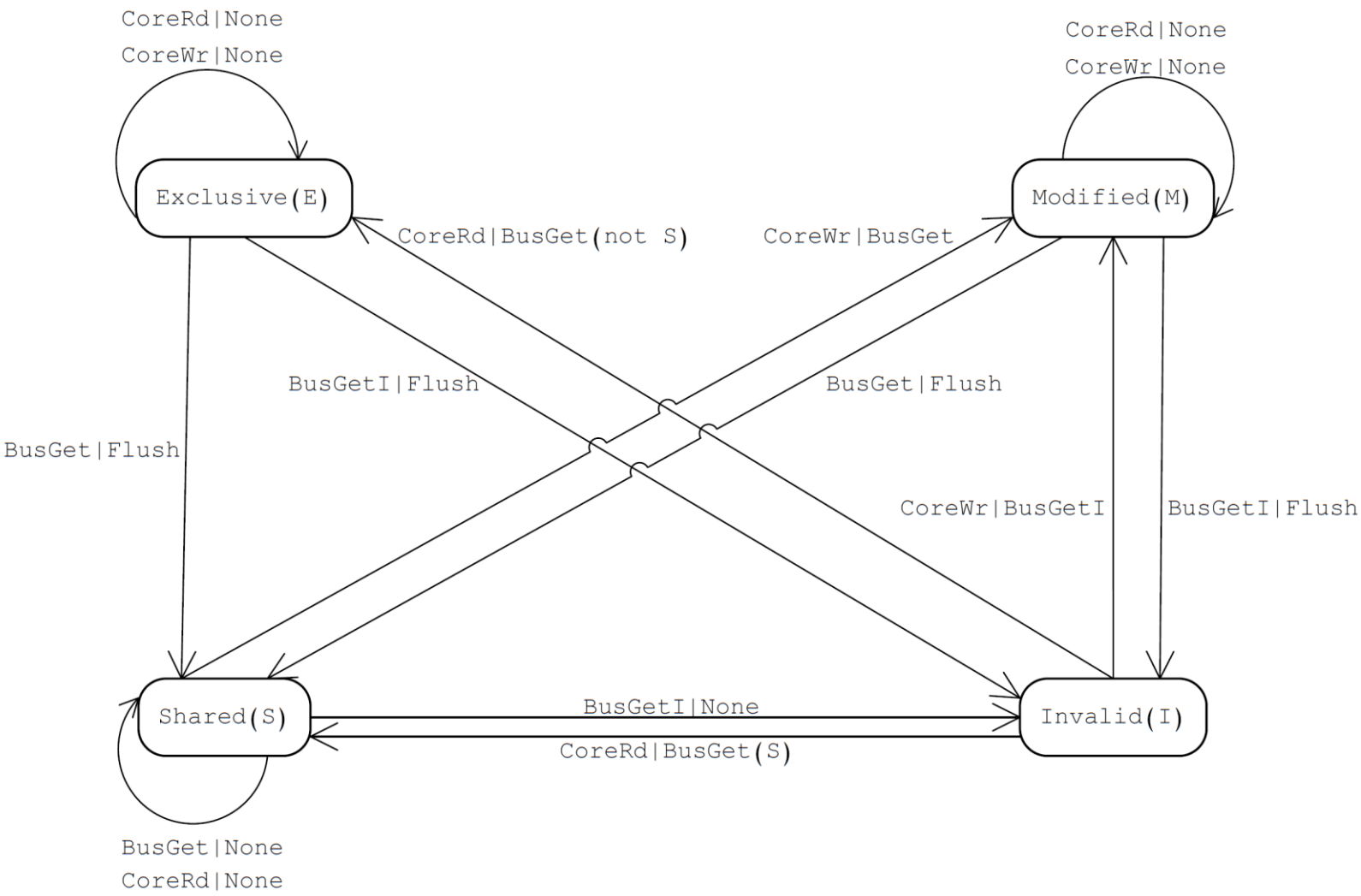
On the next page, Rachata drew a MESI state diagram. There are 4 mistakes in his diagram. **Please show the mistakes and correct them.** You may want to practice on the scratch paper first before finalizing your answer. If you made a mess, clearly write down the mistakes and the changes below.

第一个,exclusive coreWr 的时候不应该是还是 exclusive, 应该是 CoreWr/BusGetI , 然后变成 modified 状态 . 就是给其他处理器发送 invalid 信号.

第二个 exclusive busget|flush 不对, 不需要 flush 别的 cache, 因为是独占的,所以应该是 busget|none

第三个, shared corewr 变成 modified 不应该是 busget 而是 busgetI, . 就是给其他处理器发送 invalid 信号.

第四个, exclusive busgetI/flush 不对，不需要 flush 别的 cache, 因为是独占的,所以应该是 busgetI|none


CoreRd|None
CoreWr|None
Exclusive(E)
CoreRd|None
CoreWr|None
Modified(M)
CoreRd|BusGet(not S)
CoreWr|BusGet
BusGetI|Flush
BusGet|Flush
BusGet|Flush
CoreWr|BusGetI
BusGetI|Flush
Shared(S)
BusGetI|None
CoreRd|BusGet(S)
Invalid(I)
BusGet|None
CoreRd|None