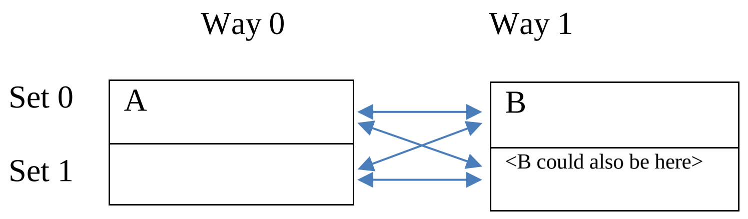
EECS 470 HW4 Winter 2024

1.

a. 2—there are two unique accesses between the first access to "A" and the second.

b.

1. 0 — the cache holds the last 2 accesses, A was just evicted by C.
2. 1 — the cache holds the last 4 accesses, A is one of those.
3. $(7/8)^2 = 49/64$; B and C must both go to a location other than the one A is in.
4. Both B and C must map to the same set as A to evict A. So the chance of a miss is $(1/2)^2$ and the chance of a hit is ¾.
5. Both have 1/4 chance of evicting A. The hashing doesn't impact things because the addresses are randomly chosen already. There is a $(3/4)^2=9/16$ chance that A will be a hit.
6. Without loss of generality say that A goes to way 0. Now in order for the next access to "A" to miss, B must end up in way 1 (in either set) AND C must conflict with both A (in way 0) and B (in way 1). First let us find the probability of B going to way 1. If B conflicts with A (which will happen 50% of the time), B will go to way 1. Otherwise, B will go to way 1 50% of the time[1]. So, we get a (0.5*1+0.5*0.5) = 0.75 chance that B is in way 1. Now, there are 4 possible combinations of how C can get mapped to ways 0 and 1, but only one of those conflicts with both A and B. Therefore, C has a 25% chance of conflict with both A and B, in which case A will be evicted. Multiply the two probabilities and you get a 0.1875 (3/16) chance of a miss on the following, which is an 81.25% (13/16) chance of a hit.



^ Arrows show possible mapping pairs for C

2.

a. The simplest solution is that max cache size is (associativity) * (page size) = 64 KB. To calculate the answer more thoroughly, first notice that 16-byte blocks means there are 4 bits for the block offset. 4KB pages means we have 12 bits for the page offset. Therefore, we can have up to 12 – 4 = 8 bits to virtually index into the cache. Since it is 4-way associative, each index can have 4*16=64 bytes. With 2^8 indices, this gives us a maximum total size of $2^8 * 64$ = 16KB.

b. As calculated above, this is $2^8=256$ sets

c. Increasing the block size by a factor of 4 would remove 2 bits from the index, meaning the number of sets would go down to 64. The total size would not change.

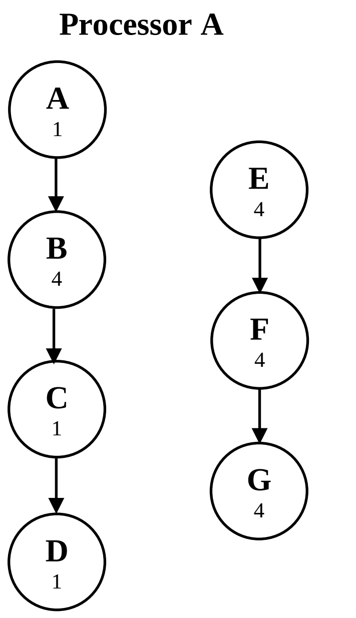
[1] Technically this isn't true. The odds of the first A going to a specific way changes based on the state of the cache. So the conditional probability of B going to the same way as A when B and A don't match in whichever way A is in isn't 50%. But it's really quite close. If anyone can convincingly compute the exact probability by hand (not simulation) I'll give them 10 extra credit homework points (first person/people only) and a donut.

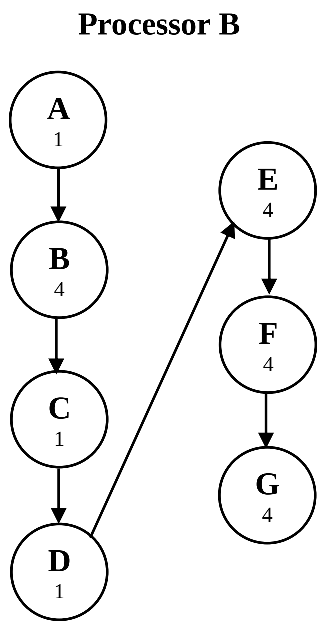
3.
   a. There are a bunch of answers, but more-or-less of the form: **A, B, C, D, C, E, B**.
   b. It looks like a best solution would be **A, B, C, D, C, E, A** or something similar.

4. Looking for: Relatively tight loop so I-cache can hit on all the instructions. Too small of a loop would create branch issues.
   a. No dependencies between instructions within the loop.
   b. Dependencies between subsequent instructions.

5.
   a.
      i. **Memory disambiguation** is when an address for a load or a store is determined. This makes it possible to determine if addresses of different loads/stores match.
      ii. **Store-to-load forwarding** is when a store sends its data directly to a load to the same address that follows the store in program order, instead of having to get the data from the cache (or memory).
      iii. An LSQ helps with memory disambiguation by tracking the order of stores (and loads) along with their addresses (if they are ready) so you can find matches.
   b. (From HW 1)
      i. **Register pressure** is when there aren't as many registers available as you could use.
      ii. **Register spills** are when you move a value to memory because there aren't enough registers.
      iii. **Register fills** (also called reloads) are when you retrieve that value from memory. *Register pressure results in spills and fills*.
   c. If there is high register pressure, register spills and fills will become stores and loads (respectively) to the same addresses in memory. Due to temporal locality in this context, a spill is often paired with a corresponding fill soon after. That translates to a store soon followed by a load from the same address which the LSQ can forward.
   d.
      i. If D is a load, you can be certain that it will need to get its data from memory if 1) all prior stores (potentially instructions A-C) have calculated their address, 2) the load's dependencies have all been resolved, and 3) the load's address matches none of the prior stores.
      ii. You can be certain that D can get its data from a store if instruction D's dependencies are resolved (so it has calculated its address), and A, B, or C is a store and has calculated its address to be the same as load D.

6. Each node is labeled with the execution time of the corresponding instruction.
    a.





    b. By analyzing the critical paths on the diagrams, processor A takes 12 cycles while processor B takes 19 cycles.

7. The synonym problem in a virtually-addressed cache is when two virtual addresses (likely but not necessarily different address spaces) map to the same physical location. In a write-back cache this can cause the data to become inconsistent as the value held in one entry of the cache may not be the same as the other even though there are supposed to be the same location. An example of the synonym problem would be virtual addresses 0x1000 and 0x2000 both mapping to the physical address 0x0000 in a write-back cache.

    The homonym problem is when the same virtual address in two different processes map to different physical memory locations. For example, VA 0x5000 in process 1 mapping to PA 0x1000 while VA 0x5000 in process 2 mapping to PA 0x2000.

8.
    a. (3 billion)*(.03)*(32 bytes) = 2.88 billion bytes/sec
    b. Writes happen when a dirty line is evicted from the cache. The rate is therefore: (3 billion)*(.03)*(32 bytes)*(0.05) = 144 million bytes/sec

9.

Processor	Address	Read/Write	Hit/Miss	Bus transaction(s)	HIT/ HITM	State after Transaction	"4C" miss type (if any)
1	0x120	Read	Miss	BRL	--	E	Compulsory
1	0x120	Write	Hit	--	--	M	--
1	0x100	Read	Miss	BRL/BWL	--	E	Compulsory
1	0x120	Write	Miss	BRIL	--	M	Conflict
2	0x120	Read	Miss	BRL	HITM	S	Compulsory
2	0x110	Write	Miss	BRIL	--	M	Compulsory
1	0x110	Read	Miss	BRL	HITM	S	Compulsory
3	0x100	Write	Miss	BRIL	--	M	Compulsory
1	0x100	Read	Miss	BRL	HITM	S	Capacity
1	0x120	Read	Miss	BRL	HIT	S	Capacity
3	0x130	Read	Miss	BRL	--	E	Compulsory
3	0x100	Read	Hit	--	--	S	--
1	0x100	Write	Miss	BRIL	HIT	M	Conflict

Final state:

	Proc 1	
	Tag	**State**
Set 0	0x100	M
Set 1	0x110	S

	Proc 2	
	Tag	**State**
Set 0	0x120	S
Set 1	0x110	S

	Proc 3	
	Tag	**State**
Set 0	0x100	I
Set 1	0x130	E

Note: having a tag but being in the invalid state is the same as having a blank tag in this context.