ASSIGNMENT 1

Name	Nihar Shailesh Joshi
NetID	njoshi27@uic.edu
UIN	677063712

1.

a. Advantages of using Simultaneous Multithreading (SMT) cores over Single Threaded cores:

- The first and most obvious advantage is that SMT cores permit multiple independent threads of execution allowing more efficient use of processor resources while single threaded cores allow only a single thread of execution to run on each core.
- SMT architecture, in addition, not only allows multiple threads but also multiple tasks (such as I/O, permissions, page table management, etc.) to run simultaneously and separately from each other. This is not possible in cores with single threaded architecture. [Source: Wiki]
- Due to multi-programming, jobs can have varying degrees of active threads along with dynamic thread-level parallelism. SMT cores can support multi-programmed workloads while single threaded cores cannot.
- Energy efficiency of SMT cores is much higher because SMT architecture enables concurrency with very little additional dynamic power. This means that even for minimal performance gain, energy saving is substantial. [Source: Stijn Eyerman & Lieven Eeckhout]
- SMT architecture provides the benefit of concealing high latencies as SMT cores can hold space and fetch instructions even when a thread is stuck or unresponsive.
- Lastly, since SMT cores support multiple threads per core, extra threads can be used proactively to seed a shared resource like a cache to improve the performance of a single thread. This adds even greater efficiency over single threaded cores. [Source: Wiki]

b. Global Branch History Register (BHR) and Return Address Stack (RAS) should be dedicated to each thread in SMT cores. SMT cores run multiple threads at once and these threads are independent from each other. As a result, each thread has its own instruction history which must be kept separate from that of the other threads. A shared BHR would violate this because it would potentially lead to a lot of conflict between thread histories. This could in turn lead to reduced performance and increased overhead. Additionally, having common BHR for multiple threads means dedicating more memory for the BHR and determining the size of this memory accurately could be a complex process without resulting in a great performance increase. It is simply better to dedicate a BHR to each thread. Speaking of RAS, independent threads will similarly have different return address stacks. These addresses must be kept separate for the same reason we keep the BHRs dedicated to each thread – preventing conflicts. RASs can be shared by threads only if the threads originate from the same process (and same code segment). Lastly, keeping the RAS between threads separate also ensures that return calls of independent threads will not get in the way each other. [Source: Xidong Wang & Ning Li]

2. We have the following code segments:

P1	**P2**	**P3**
(1a) A = 3	(2a) u = B	(3a) w = A
(1b) B = 2	(2b) v = A	(3b) x = B

We can see from the table that u, v, w, x can have the following values:

- u = (0, 2)
- v = (0, 3)
- w = (0, 3)
- x = (0, 2)

Sequential Consistency (SC) is achieved only for the case 1a → 1b, 2a → 2b, 3a → 3b.

We can see this from the following timeline:

u, v, w, x	**Statement Interleaving**
0, 0, 0, 0	2a → 2b → 3a → 3b → 1a → 1b
0, 0, 0, 2	2a → 2b → 3a → 1a → 1b → 3b
0, 0, 3, 0	2a → 2b → 1a → 3a → 3b → 1b
0, 0, 3, 2	2a → 2b → 1a → 1b → 3a → 3b
0, 3, 0, 0	3a → 3b → 2a → 1a → 1b → 2b
0, 3, 0, 2	2a → 3a → 1a → 1b → 2b → 3b
0, 3, 3, 0	1a → 2a → 2b → 3a → 3b → 1b
0, 3, 3, 2	2a → 1a → 1b → 2b → 3a → 3b
2, 0, 0, 0	SC violated: 2b → 1a and 1b → 2a together
2, 0, 0, 2	SC violated: 2b → 1a and 1b → 2a together
2, 0, 3, 0	SC violated: 2b → 1a and 1b → 2a together
2, 0, 3, 2	SC violated: 2b → 1a and 1b → 2a together
2, 3, 0, 0	3a → 3b → 1a → 1b → 2a → 2b
2, 3, 0, 2	3a → 1a → 1b → 2a → 2b → 3b
2, 3, 3, 0	1a → 3a → 3b → 1b → 2a → 2b
2, 3, 3, 2	1a → 1b → 2a → 2b → 3a → 3b