4 Runahead Execution [65 points]

Assume an in-order processor that employs Runahead execution, with the following specifications:

- The processor enters Runahead mode when there is a cache miss.
- There is no penalty for entering and leaving the Runahead mode.
- There is a 64KB data cache. The cache block size is 64 bytes.
- Assume that the instructions are fetched from a separate dedicated memory that has zero access latency, so an instruction fetch never stalls the pipeline.
- The cache is 4-way set associative and uses the LRU replacement policy.
- A memory request that hits in the cache is serviced instantaneously.
- A cache miss is serviced from the main memory after X cycles.
- A cache block for the corresponding fetch is allocated *immediately* when a cache miss happens.
- The cache replacement policy does *not* evict the cache block that triggered entry into Runahead mode until after the Runahead mode is exited.
- The victim for cache eviction is picked at the same time a cache miss occurs, i.e., during cache block allocation.
- ALU instructions and Branch instructions take one cycle.
- Assume that the pipeline *never stalls* for reasons *other than data cache misses*. Assume that the conditional branches are always correctly predicted and the data dependencies do not cause stalls (except for data cache misses).

Consider the following program. Each element of Array A is one byte.

```
for(int i=0;i<100;i++){ \\ 2 ALU instructions and 1 branch instruction
    int m = A[i*16*1024]+1; \\ 1 memory instruction followed by 1 ALU instruction
    ... \\ 26 ALU instructions
}
```

(a) [20 points] After running this program using the processor specified above, you find that there are 66 data cache hits. What are **all** the possible values of the cache miss latency X? You can specify all possible values of X as an inequality. Show your work.

$61 < X < 93$.

Explanation. The program makes 100 memory accesses in total. To have 66 cache hits, a cache miss needs to be followed by 2 cache hits. Hence, the Runahead engine needs to prefetch 2 cache blocks. After getting a cache miss and entering Runahead mode, the processor needs to execute 30 instructions to reach the next LD instruction. To reach the LD instruction 2 times, the processor needs to execute at least 62 instructions (2*(29 ALU + 1 Branch + 1 LD)) in Runahead mode. If the processor spends more than 92 cycles in Runahead mode, then it will prefetch 3 cache lines instead of two, which will cause the number of cache hits to be different. Thus, the answer is $61 < X < 93$.

(b) [20 points] Is it possible that *every* memory access in the program misses in the cache? If so, what are **all** possible values of X that will make all memory accesses in the program miss in the cache? If not, why? Show your work.

Yes, for $X < 31$ and $X > 123$.

Explanation. When X is equal to or smaller than 30 cycles, the processor will be in Runahead mode for insufficient amount of time to reach the next LD instruction (i.e., the next cache miss). Thus, none of the data will be prefetched and all memory accesses will get cache miss.
When X is larger than 123 cycles, the processor will prefetch 4 cache blocks. Since the prefetched cache blocks will map to the same cache set, the latest prefetched cache block will evict the first prefetched cache block in Runahead mode (note that the cache block that triggered Runahead execution remains in the cache due to the cache block replacement policy). This will cause a cache miss in the next iteration after leaving the Runahead mode. Thus, the accesses in the program will always miss in the cache.

(c) [25 points] What is the *minimum* number of cache misses that the processor can achieve by executing the above program? Show your work.

25 cache misses.

Explanation. When $92 < X < 124$, the Runahead engine will prefetch exactly 3 cache blocks that will be accessed after leaving the Runahead mode. It is the minimum number of misses that could be achieved since all cache blocks accessed by the program map to the same cache set and the cache is 4-way associative.