CS 533 **Name:** ____________________
Spring 2024
Homework 2 **NetID:** ____________________
Assigned: 2/13/2024
Due: 3/7/2024 **UIN:** ____________________

This homework contains 7 pages (including this cover page) and 6 questions.
Please clearly print or typeset your answers.
Note: up to 6 points (out of 70) may be deducted for illegible submissions

Problem Breakdown

Question	Points	Score
1	8	
2	16	
3	13	
4	10	
5	11	
6	12	
Total:	70	

1. (8 points) Prefetching 1

   Consider the following loop:

```
for (int i = 0; i < 128; i++){
    for (int j = 0; j < 32; j++){
        A[i, j] = B [i] * C[j]
    }
}
```

   Assume the following:

   - The cache is large enough to avoid any conflict or capacity miss.
   - Cache block size is 64 bytes.
   - Each element of arrays A, B and C is 8-byte long.
   - It takes 64 cycles to service a cache miss.
   - Multiplication of line (3) takes 16 cycles if all memory accesses are cache hits.
   - It is an in-order processor and an on-demand miss stalls the execution.
   - At most two prefetch misses can be pending at any time. The third one will be ignored by the processor.
   - Size of array A is 128x32, B is 128 and C is 32.
   - A[i, j] and A[i, j + 1] are contiguous in memory, and so are A[i, 31] and A[i + 1, 0] (row-major layout).
   - Ignore the time spent performing loop boundary calculations and any other control-flow instructions.
   - `PREFETCH` and `PREFETCH EX` instructions are free (take 0 cycles to execute)

   (a) What is the total execution time of the above loop? Calculate this by adding cache miss penalties to the 16-cycle iteration time given above.

   (b) Try to improve performance by prefetching those references that you expect to miss in the cache. Explain your line of reasoning. Do not use loop unrolling. Avoid unnecessary prefetches.

   Assume that a read prefetch is expressed as `PREFETCH(&variable)` and it fetches the entire cache line in which the variable resides in shared mode. A read-exclusive prefetch operation, which is expressed as `PREFETCH EX(&variable)`, fetches the line in exclusive mode.

2. (16 points) Prefetching 2

   (a) What problem does prefetching attempt to tackle? Can prefetching hurt performance?

   (b) Is hardware prefetching more effective for single-issue statically scheduled processors or multiple issue dynamically scheduled processors? Why?

   (c) How early can a binding prefetch for variables in a critical section be issued in a cache coherent system?

   (d) Prefetch tries to bring a cache line before it is requested. Can this lead to any violation of memory consistency? Explain.

   (e) Briefly explain the intuition behind Runahead execution[1].

   (f) Contemporary processors have HW prefetchers that automatically prefetch the cache lines. Identify one memory access pattern that can be automatically prefetched. How can you detect the pattern via hardware?

   (g) In the paper detailing prefetching in the SUIF compiler[2] , the authors discuss the tradeoffs associated with trying to perform a prefetch when the prefetch buffer is full. What are the possible options and advantages of each solution?

   (h) Some processors replace coherent caches with scratchpad memories [3] where the memory is cached only by software instructions. Explain one advantage and one disadvantage of this approach.

[1]O. Mutlu et al. “Runahead Execution: An Effective Alternative to Large Instruction Windows,” HPCA, 2003

[2]T. Mowry et al. “Design and Evaluation of a Compiler Algorithm for Prefetching,” ASPLOS, 1992

[3]R. Banakar et al. ”Scratchpad memory: design alternative for cache on-chip memory in embedded systems”, CODES 2002

3. (13 points) Synchronization 1

   (a) Consider the following code segment:

```
BARRIER(bar_name, p) {
  local_sense = !local_sense;  // toggle private sense variable
  LOCK(bar_name.lock);
  mycount = bar_name.counter++; // mycount is a private variable
  if (bar_name.counter == p){ // last to arrive at this barrier
    UNLOCK(bar_name.lock);
    bar_name.counter = 0;      // reset counter for next barrier
    bar_name.flag = local_sense; // release waiting processes
  } else {
    UNLOCK(bar_name.lock);
    while (bar_name.flag != local_sense) {
        ; //busy wait for release
    }
  }
}
```

   Consider this barrier algorithm with sense reversal. Would there be a problem if the UNLOCK statement were placed just after the increment of the counter rather than after each branch of the if condition?

   (b) Can you implement the above barrier with fetch & inc operation?

   (c) Suppose you have a uniprocessor running multithreaded programs written using the pthreads library. Do you still need lock and unlock operations? Would you need special instructions (like test & set or LL/SC) to implement lock/unlock in that uniprocessor? If not, how else can you implement them?

   (d) Implement compare & swap and fetch & inc using LL/SC.

   (e) Lock-free algorithms are an alternative to lock-based synchronization.

      i. What is the high-level idea behind lock-free algorithms?

      ii. What advantages do they have over traditional locks?

      iii. Are there any disadvantages?

      iv. How are they implemented on machines?

4. (10 points) Synchronization 2: Spin locks

   Suppose all 8 processors in a bus-based machine try to acquire a test & test & set lock simultaneously. Assume all processors are spinning on the lock in their caches and are invalidated by a release at the beginning.

   (a) How many bus transactions will it take until all processors have acquired the lock if all the critical sections are empty (each processor only executes a LOCK operation, immediately followed by an UNLOCK operation with no other operations in between)?

   (b) Assuming that the bus is fair (services pending requests before new ones) and that every bus transaction takes 50 cycles, how long would it take before the first processor acquires and releases the lock? How long before the last processor to acquire the lock is able to acquire and release it?

   (c) If the variables used for implementing locks are not cached, will a test & test & set lock still generate less traffic than a test & set lock? Explain.

   (d) Why would one use Array-Based Queue Locks (ABQLs) instead of test & test & set? Are there any downsides to ABQLs? Does the MCS lock solve these problems?

   (e) Can you implement the MCS lock using LL/SC instead of compare & swap and fetch & store? Please explain how you can implement it, or why it is not implementable.

5. (11 points) SMT

   (a) How does SMT differ from superscalars and traditional multithreading?

   (b) Between an in-order and out-of-order superscalar processor, which would benefit more from adding SMT capabilities? Why?

   (c) Indicate for each of the following structures whether it should be shared or duplicated in a SMT machine, or if it can be both? Explain.

      i. Branch predictor
      ii. Return address stack
      iii. Register file
      iv. TLB
      v. Register Aliasing Table (RAT)
      vi. Load-Store Queue (LSQ)

   (d) What are the factors against building wide-issue superscalars? You can look at this paper[4] for valuable insights.

   (e) In multiscalar processors, list conditions under which a task may get squashed. Do all the successors of a squashed task also need to be squashed for correctness?

[4]S. Palacharla et al. "Complexity-Effective Superscalar Processors," ISCA, 1997

6. (12 points) SMT and CMP

   (a) For which types of applications is it better to have a SMT machine instead of a CMP machine? What about applications that have higher performance on a CMP machine over an SMT machine? Assume both machines use the same die size.

   (b) Compare SMTs and CMPs in terms of hardware complexity. Carefully explain all the aspects that contribute to it.

   (c) Several papers explain how to transform a machine's structure based on the codes being run[5],[6] (CMP $\rightarrow$ SMT, OoO processor $\rightarrow$ SMT in-order processor).

      i. Explain the high-level transformation process of a CMP machine $\rightarrow$ SMT machine. Give details about the hardware structures needed to be modified.

      ii. Explain the limitations of the Core Fusion.

      iii. Describe the approach MorphCore takes to handle workload diversity. Give details about the hardware structures that have to be modified.

   (d) Many modern processors incorporate heterogeneous big.little [7] architecture. Explain one benefit of big.little over SMT. Also, identify one technical challenge of heterogeneous architecture.

[5] E. Ipek et al. "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors" ISCA, 2007

[6] Khubaib et al. "MorphCore: An Energy-Efficient Microarchitecture for High Performance ILP and High Throughput TLP" MICRO, 2012

[7] https://www.arm.com/technologies/big-little