1. ISA

Compare-and-swap (R1, R2, L) is an atomic synchronization primitive which atomically compares the value in memory location L with R1, and if and only if they are equal, exchanges the values in R2 and L. For instance, if R1=2, R2=3, and L=4, compare and-swap (R1, R2, L) would yield no changes; but if R1=2, R2=3, and L=2, compare-and-swap (R1, R2, L) would result in R2 containing 2 and L containing 3. Compare-and-swap can be used to efficiently emulate many other primitives.

a) Implement an atomic test-and-set on memory address L in assembly using compare-and-swap as the only atomic primitive. Let L=1 when the lock is taken, and L=0 when it is free. You can use any registers you like. The test-and-set instruction is an instruction used to write to a memory location and return its old value as a single atomic (i.e. non-interruptible) operation. Typically the value 1 is written to the memory location. If multiple processes may access the same memory, and if a process is currently performing a test-and-set, no other process may begin another test-and-set until the first process is done.

Sample code for test-and-set:

```c
function Lock (boolean *lock) {
    while (test_and_set(lock) == 1);
}
```

b) Implement the test-and-test-and-set semantics on memory address L in assembly using compare-and-swap as the only atomic primitive. Let L=1 when the lock is taken, and L=0 when it is free. You can use any registers you like as well as ordinary loads and stores.

Sample code for test-and-test-and-set:

```c
boolean locked := false    //shared lock variable
do {
    while (locked == true)
} while TestAndSet(locked)    // actual atomic locking
```
c) Use compare-and-swap to implement an atomic fetch-and-increment(R1,L) in assembly, which atomically copies the value in L to R1, then increments the value in L by 1. Again, you can use any registers you like well as ordinary loads and stores.

2. ISA
You are writing a singly linked list struct and its associated node struct in C. Your compiler is as simple as can be functionally correct for converting C to LC-3b machine code. Some crucial functions will be external linked assembly, written by hand.

a) How can you optimize what you write to allow whoever writes the assembly to gain an asymptotic constant factor speedup in a function to get the nth element of the list?

b) How will this trade-off against the performance of a similarly hand-coded assembly function to tell whether a given value is in the list or not? Assume that your implementation of LC-3b is not pipelined, has no cache, and executes every instruction in equal time.
3. Performance
You are going to write a program using a non-pipelined processor with 2GHz. The program you write has 1000 instructions and roughly 75% of the arithmetic/integer operations and 25% of the memory operations. The cache system embedded takes 2 cycles on a cache hit and 50 cycles on a cache miss. All other integer operations take 1 cycle each. Prior to this step, you already had an analysis and found out the hit rate of your program is 95% for both instruction and data cache. How long does the program take to run?

4. Performance
Consider a multi-cycle processor with 5 stages (fetch, decode, memory, execute, writeback). Every instruction must pass through each stage; there is no cache. A stage with no memory access takes 1 cycle (decode, execute, writeback); a stage with a memory access takes 10 cycles (fetch, memory). The memory access delay is avoided in the memory stage if the instruction is not a load or store. Assume 40% of instructions are loads and stores.

a) What is the CPI of this machine?

b) A cache with a 95% hit rate (1-cycle hit) is added to the machine (services both fetch and memory stages). What is the new CPI of this machine?

c) An additional L2 cache with 99% hit rate (3-cycle hit) is added to the machine. What is the speedup over part A? What is the speedup over part B?
5. Caches
How much overhead storage must a 4-way set-associative cache which keeps two kinds of flag bits (valid and dirty) with 8 byte blocks and 1KB capacity for computer with 32-bit byte addresses have?

6. Caches
a) Draw a flow chart describing the operation of a write-back cache with write allocation. Deal with the cases of reads and writes, hits and misses, both states of the dirty bit, and show when the dirty bit is set/cleared. You need not worry about address translation for this problem.

b) Do the same (where applicable) with a write-through cache without write-allocation.
7. Caches
You are working on the course project and building a processor with 32-bit physical address that are word-addressable. Assigned project is to build the cache system that is 8KB in size, 8-way set associative and each block has 16-bytes each. A word is consisted of 4 bytes for this architecture. This cache uses pseudo-random replacement or least recently used (LRU) policy. Now, you are planning to go through step by step to find the structure of your design details.

a) Find the number of bits needed for each section of the cache implementation. Tag, Index, Offset

b) You are curious how other designs would be different if you had the flexibility of the set associativity. Find the bits of same section as a) for direct-mapped and fully associative

c) How many comparators are needed for each set-associativity; direct-mapped, 8-way set-associative, fully associative

d) For the replacement policy, you have the freedom to choose between two policies. Before choosing, you want to do a short analysis on the cost of each replacement policy. Write down the total number of bits needed for each replacement policy and the complete equation that you used to calculate. Which policy would you choose to implement? Give a good reasoning.
8. Virtual Memory
A hypothetical computer system has 64GB byte-addressable virtual memory with a 16MB page with but only supports 4GB of physical memory. This system has 2-levels of cache, L1 and L2 with each line size of 64 bytes. L2 cache has the capacity of 256KB which uses 8-way set-associative with LRU replacement policy. TLB implemented in the system is direct-mapped consisting of 16 entries. Fill in the blank of the below diagram.