**A1: Performance (25 points)**

Consider the execution of a program that results in the execution of 2 million instructions on a 400MHz uniprocessor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given in the table below based on the result of a program trace experiment.

| **Instruction type** | **CPI** | **Instruction Mix** |
|---|---|---|
| Arithmetic and logic | 1 | 60% |
| Load/store with cache hit | 2 | 18% |
| Branches | 4 | 12% |
| Memory reference with cache miss | 8 | 10% |

**1.a (4 Points)** Compute the average CPI and the corresponding MIPS rate.

**1.b (6 Points)** Now assume that the program can be executed on eight parallel threads with roughly equal number of instructions executed in each thread. Execution is on an 8-core system with each core having the same performance as the uniprocessor originally used. Coordination and synchronization among threads adds an extra 25,000 instruction executions to each thread. Assume the same instruction mix as in the uniprocessor case above, but increase the CPI for memory reference with cache miss to 12 cycles due to contention for memory among threads. Compute the average CPI and corresponding MIPS rate.

**1.c (9 Points)** Calculate the actual speedup factor and compare it with the theoretical speed up factor determined by Amdhal's law. *Hint:* To answer this question, compute the execution time of the program on the uniprocessor and the execution time on the 8-core system.

**1.d (6 Points)** MIPS has been proven inadequate to compare the performance of different architectures, e.g., to compare CISC and RISC architectures. Briefly explain why and provide a short code example in which the comparison of a CISC and RISC is misleading and incorrect when using MIPS.

## A2: Cache and memory (25 Points)

Consider a computer with the following characteristics:

- Total of 1Mbyte of main memory
- Word size of 1 byte
- Block size (or line size) of 16 bytes
- Cache size of 64 Kbytes

**2.a (5 Points)** Given the main memory address 0xCABBE provide the corresponding tag, cache line address, and word offsets for a direct-mapped cache. How do you determine if the address is a cache hit or a miss?

**2.b (5 Points)** Give another main memory address that maps to the same cache slot as the address 0xCABBE for the direct-mapped cache.

**2.c (5 Points)** Given the main memory address 0xCABBE provide the corresponding tag and offset values for a fully-associative cache. How do you determine if the address is a cache hit or a miss?

**2.d (5 Points)** Briefly define the write policy called write-back. How is this different from the write-through policy in terms of memory traffic?

**2.e (5 Points)** Consider the given cache with a line size of 16 bytes and a main memory that requires 20 ns to transfer a 1-byte word. For any line that is written at least once before being swapped out of the cache, what is the average number of times that the line must be written before being swapped out for a write-back cache to be more efficient than a write-through cache?

## A3: Simple, Bus-Based Multiprocessor (25 Points)

The simple, bus-based multiprocessor illustrated in Figure 1 represents a commonly implemented symmetric shared-memory architecture with four processors or CPUs. Each processor has a single, private cache with coherence maintained using the snooping coherence protocol of Figure 2. Each cache is direct-mapped, with four blocks each holding two words. To simplify the illustration, the cache address tag contains the full address and each word shows only two hex characters, with the least significant word on the right. The coherence states are denoted E, S, and I for Exclusive, Shared, and Invalid.

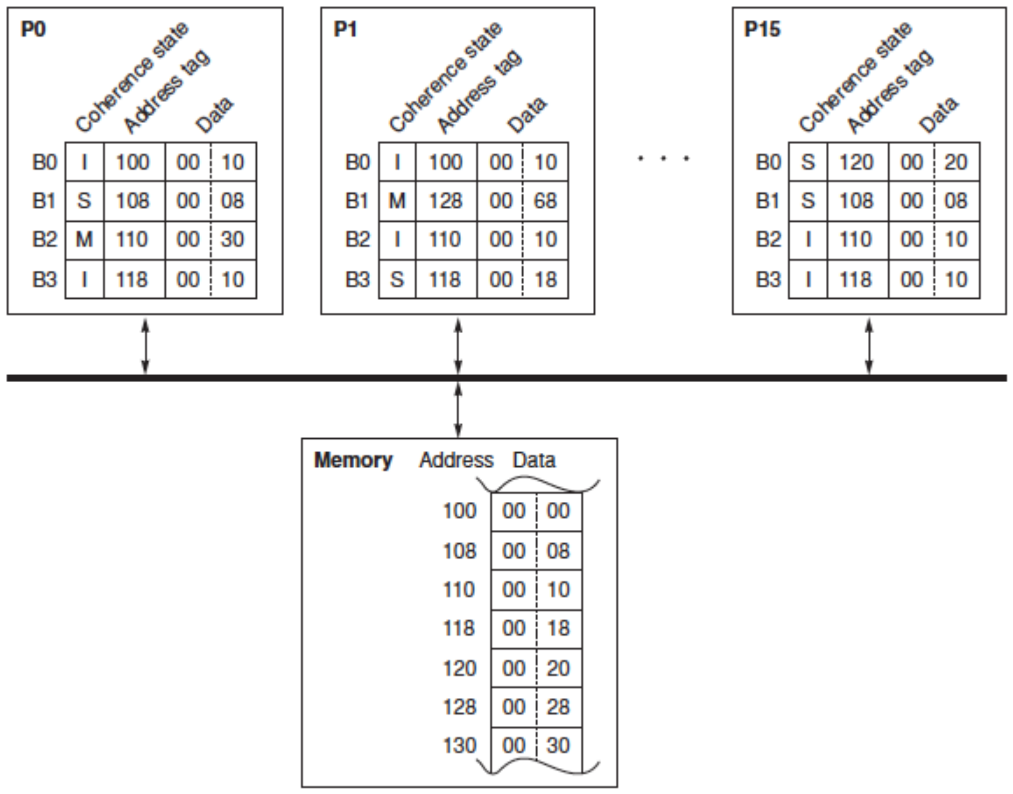

**Figure 1: Bus-based snooping multiprocessor**

Assume the initial cache and memory state as illustrated in Figure 1. Consider a sequence of CPU operations of the form:

P#: <op> <address> [ <-- <value> ]

where P# designates the CPU (e.g., P0), <op> is the CPU operation (e.g., read or write), <address> denotes the memory address, and <value> indicates the new word to be assigned on a write operation.

Also assume that a CPU read miss to a cache block that is exclusive in another processor's cache is faster than a miss to a block in memory, and thus satisfied by the other processor's cache. This is because caches are smaller, and thus faster, than main memory. Contrary, a CPU read miss to cache blocks that are shared is satisfied by memory.

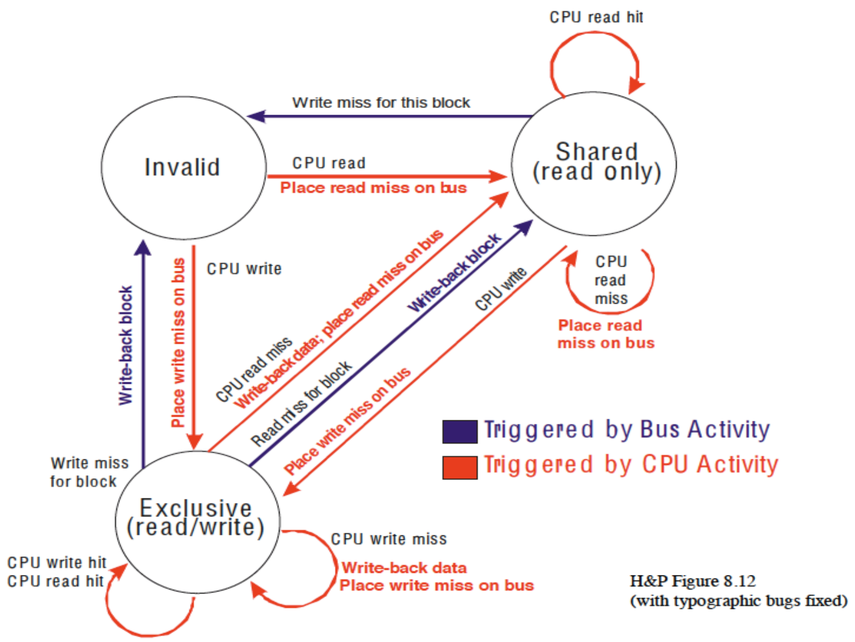

**Figure 2: Cache coherence state diagram with the state translations induced by the local processor (red) and by the bus activities (in blue)**

Consider the sequence of CPU operations below. Assume that the second operation begins after the first completes, the third operation begins after the second completes (even though they are on different processors), and so on.

P1: read 128
P15: read 128
P0: read 128
P0: read 118
P15: read 118
P1: read 118

Consider these additional conditions:

- CPU **read and write hits** generate no stall cycles,
- CPU **read and write misses** generate $N_{memory}$ and $N_{cache}$ stall cycles if satisfied by memory and cache, respectively,
- CPU **write hits** that generate an invalidate incur $N_{invalidate}$ stall cycles,
- A write back of a block, either due to a cache replacement or due to a cache supplying a block in response to another processor's request, incurs an additional $N_{writeback}$ stall cycles.

Finally, consider the performance characteristics summarized in Table 1.

| Type of stall cycles | Latency |
|---|---|
| $N_{memory}$ | 100 |
| $N_{cache}$ | 60 |
| $N_{invalidate}$ | 20 |
| $N_{writeback}$ | 10 |

**Table1: Snooping coherence latencies.**

3.1 **(8 Points)** For the given sequence, what is the resulting state (i.e., coherence state, tags, and data) of the caches and memory **after each CPU operation is performed**? Show **only** the blocks that change, for example, P0.B0: (I, 120, 00 01) indicates that CPU P0's block B0 has the final state of I, tag of 120, and data words 00 and 01. What value is returned **by each read operation**?

3.2 **(8 Points)** How many stall cycles are generated by the implementation? To answer this question, you have to use the snooping coherence latencies in Table 1. Note that just the number of stall cycles is not sufficient to get full score. For each operation, identify whether the operation is associated to CPU read and write misses or hits. If a miss takes place, indicate if it is satisfied by another cache or the memory.

3.3 (**9 Points**) In the implementation proposed in above, misses are incurring fewer stall cycles when they are supplied by cache than when they are supplied by memory. Some coherence protocols try to improve performance by increasing the frequency of this case. A common protocol optimization is to introduce an Owned state (usually denoted O). The Owned state behaves like the Shared state, in that nodes may only read Owned blocks. But it behaves like the Exclusive state, in that nodes must supply data on other nodes' read and write misses to Owned blocks. A read miss to a block in either the Exclusive or Owned states supplies data to the requesting node and transitions to the Owned state. A write miss to a block in either state Exclusive or Owned supplies data to the requesting node and transitions to state Invalid. This optimized protocol only updates memory when a node replaces a block in state Exclusive or Owned. For the given code, compute the total stall cycles for the optimized protocol. Assume state transitions that do not require bus transactions incur no additional stall cycles.

How many stall cycles are generated by this optimized implementation? To answer this question, again you have to use the snooping coherence latencies in Table 1.Note that just the number of stall cycles is not sufficient to get full score. For each operation, identify whether the operation is associated to CPU read and write misses or hits. If a miss takes place, indicate if it is satisfied by another cache or the memory. Which of the two protocols gives better performance for the given sequence of CPU operations?

*Hints to solve 3.2:*
This example is provided to you to help you with the solution of this exercise. Consider the following sequence of operations assuming the initial cache state in Figure 1. Assume that the second operation begins after the first completes (even though they are on different processors):

P1: read 110
P15: read 110

P1: read 110 P1 stalls $N_{cache}$ stall cycles for **read miss** satisfied by P0 cache
P0 stalls for $N_{writeback}$ cycles while it writes the block back to memory in response to P1's request
Stalls: $N_{cache} + N_{writeback}$

P15: read 110 P15 stalls $N_{memory}$ stall cycles for **read miss** satisfied by satisfied by memory
Stalls: $N_{memory}$

Total Stalls: $N_{cache} + N_{writeback} + N_{memory}$

**A4: Superscalar Instruction Issue Policies (25 Points)**

**4.1 (6 Points)** Briefly define these three types of superscalar instruction issue policies: In-order issue with in-order completion, In-order issue with out-of-order completion, and Out-of-order issue with out-of-order completion.

**4.2 (19 Points)** Consider the following sequence of instructions, where the syntax consists of an opcode followed by the destination register followed by one or two source registers:

| | |
|---|---|
| I0 | ADD r3, r1, r2 |
| I1 | LOAD r6, [r3] |
| I2 | AND r7, r5, 3 |
| I3 | ADD r1, r6, r0 |
| I4 | SRL r7, r0, 8 |
| I5 | OR r2, r4, r7 |
| I6 | SUB r5, r3, r4 |
| I7 | ADD r0, r1, 10 |
| I8 | LOAD r6, [r5] |
| I9 | SUB r2, r1, r6 |
| I10 | AND r3, r7, 15 |

Assume the use of a four-stage pipeline: fetch, decode/issue, execute, and write back. Assume that all pipeline stages take one clock cycle except for the execute stage. For simple integer arithmetic and logical instructions, the execute stage takes one cycle, but for a LOAD from memory, five cycles are consumed in the execute stage.

Assume you have a simple scalar pipeline, but allow out-of-order execution; we can construct the following table for the execution of the first seven instructions:

| Instruction | | Fetch | Decode | Execute | Writeback |
|---|---|---|---|---|---|
| I0 | ADD r3, r1, r2 | 0 | 1 | 2 | 3 |
| I1 | LOAD r6, [r3] | 1 | 2 | 4 | 9 |
| I2 | AND r7, r5, 3 | 2 | 3 | 5 | 6 |
| I3 | ADD r1, r6, r0 | 3 | 4 | 10 | 11 |
| I4 | SRL r7, r0, 8 | 4 | 5 | 6 | 7 |
| I5 | OR r2, r4, r7 | 5 | 6 | 8 | 10 |
| I6 | SUB r5, r3, r4 | 6 | 7 | 9 | 12 |
| I7 | ADD r0, r1, 10 | | | | |
| I8 | LOAD r6, [r5] | | | | |
| I9 | SUB r2, r1, r6 | | | | |
| I10 | AND r3, r7, 15 | | | | |

The entries under the four pipeline stages indicate the clock cycle at which each instruction begins each phase. In this program, the second ADD instruction (I3) depends on the LOAD instruction (I1) for one of its operands, r6. Because the LOAD instruction takes five clock cycles, and the issue logic encounters the dependent ADD instruction after two clocks, the issue logic must delay the ADD instruction for three clock cycles. With an out of order capability, the processor can stall instruction I3 at clock cycle 4, and then move on to issue the following three independent instructions, which enter execution at clocks 6, 8, and 9. The LOAD finishes execution at clock 9, and so the dependent ADD can be launched into execution on clock 10.

**4.2.a (5 Points)** Complete the table up to the completion of I10

| Instruction | | Fetch | Decode | Execute | Writeback |
|---|---|---|---|---|---|
| I0 | ADD r3, r1, r2 | 0 | 1 | 2 | 3 |
| I1 | LOAD r6, [r3] | 1 | 2 | 4 | 9 |
| I2 | AND r7, r5, 3 | 2 | 3 | 5 | 6 |
| I3 | ADD r1, r6, r0 | 3 | 4 | 10 | 11 |
| I4 | SRL r7, r0, 8 | 4 | 5 | 6 | 7 |
| I5 | OR r2, r4, r7 | 5 | 6 | 8 | 10 |
| I6 | SUB r5, r3, r4 | 6 | 7 | 9 | 12 |
| I7 | ADD r0, r1, 10 | | | | |
| I8 | LOAD r6, [r5] | | | | |
| I9 | SUB r2, r1, r6 | | | | |
| I10 | AND r3, r7, 15 | | | | |

**4.2.b (7 Points)** Redo the table assuming no out-of-order capability. Compare with the table of question a. in which out-of-order execution is available. What is the saving using the capability?

**4.2.c (7 Points)** Redo the table assuming a superscalar implementation that can handle two instructions at a time at each stage and for which out-of-order capability is available.

**B1. Computer Networks:** Reliable Data Transfer (25 points)

**(a) (4 points)** In an unreliable network, what could go wrong to a data packet? For each of these problems, specify the mechanisms used to detect and correct the problem for the purpose of providing reliable data transfer.

**(b) (9 points)** Pipelined protocols are effective solutions to provide reliable data transfer. There are two basic approaches to pipelined reliable transfer: Go-Back-N and Selective Repeat. You are to design a reliable data transfer protocol for a 1-Mbps point-to-point link between Earth and a satellite 3 x 10^4 km apart using a data frame size of 1,000 bytes. What is the minimum number of bits needed for the sequence number in Go-Back-N and Selective Repeat, respectively, to maximize channel utilization. The speed of light is 3 x 10^8 meter per second.

**(c) (7 points)** Using the same network parameters given in (b), what is the maximum achievable channel utilization for

(1) alternating-bit protocol with 4-bit sequence number;
(2) Go-Back-N protocol with 4-bit sequence number;
(3) Selective Repeat protocol with 4-bit sequence number.

(Assume ACKs are always piggybacked onto data frames and overhead of protocol header is ignored.)

**(d) (5 points)** In designing a new TCP-like reliable transport protocol, the size (number of bits) of "Sequence Number" field and the "Receive Window" field in the protocol header need to be determined. Please describe succinctly how the parameters of network bandwidth (BW), maximum round-trip-time (RTT), and maximum segment lifetime (MSLT) should be used to determine the minimum number of bits needed in the "Sequence Number" field and the "Receive Window" field.

**B2. Computer Networks:** Miscellaneous Topics (25 points)

**(a) (7 points)** Describe how virtual circuit networks operate and give at least one example. Compare and contrast virtual circuit networks and datagram networks in terms of e.g. resource sharing (multiplexing strategies) and connection establishment.

**(b) (5 points)** Define and describe DNS. What are its Resource Records (RRs)? Name at least two kinds.

**(c) (3 points)** How does DNS manage caching?

**(d) (5 points)** Compare and contrast link state and distance vector routing protocols.

**(e) (5 points)** Define and describe CIDR. Give an example of CIDR notation and address aggregation.

**B3. Computer Networks:** Network Layer (25 points)

**(a)** **(3 points)** Network Address Translation (NAT) is finding widespread use in the Internet these days. Describe the purpose of NAT and what problem is solved by it.

**(b)** **(6 points)** Explain how NAT works with a brief example.

**(c)** **(6 points)** List at least three objections that are commonly raised against the use of NAT.

**(d)** **(5 points)** IPv6 was proposed much earlier than NAT, yet it has not been as widely adopted as NAT. Analyze the reasons why IPv6 has been so slow in finding acceptance, and contrast with why NAT has been so successful.

**(e)** **(5 points)** Reassembly of IP fragments is usually performed at the destination host. Consider an IP datagram that is sent to a host behind a NAT router. If this datagram is fragmented at some point soon after its transmission, who should be responsible for its reassembly? Explain your answer.

**B4. Computer Networks:** LANs (25 points)

**(a)** **(3 points)** The Ethernet protocol does not include an Acknowledgement frame. Explain why that is the case. How does a sending node in an Ethernet network know that the packet either has or has not reached its destination?

**(b)** **(5 points)** Describe the binary exponential backoff algorithm used in Ethernet.

**(c)** **(5 points)** Suppose a fixed backoff algorithm is used in which the backoff interval is randomly chosen from a fixed set of integers 0, 1, 2, …, N-1. Would it be better to use a small value of N (such as 2 or 4), or a large value of N (such as 16 or 32)? Consider two cases, one in which only two packets collide, and a second in which a large number of packets collide.

**(d)** **(3 points)** Why does Ethernet use the exponential backoff in preference to the fixed backoff?

**(e)** **(5 points)** State clearly in the form of an equation the relationship between Ethernet's data rate and the maximum distance between two nodes on the network. How does this relationship impact the distance spanned by an Ethernet when the data rate is increased?

**(f)** **(4 points)** Do you agree with the statement: "Gigabit Ethernet is really not an Ethernet technology." Why or why not? Given the reality of Gigabit Ethernet, what is the one true constant feature of all Ethernet technologies?

**C1. Operating Systems** (25 points)

**a) (2 points)** On a uniprocessor system, what is the root cause of mutual exclusion violation?

**b) (13 points)** Consider the following two functions that are executed by multiple threads:

```
static int count = 0;
int increment(void)
{
  count++;

  if (count > 3) {
    printf("++ counter value (%d) > 3\n", count);
    return 0;
  }

  return 1;
}

int decrement(void)
{
  while (count > 3) {
    printf("-- counter value (%d) > 3\n", count);
    count--;
  }

  if (count == 0) return 0;
  else return 1;
}
```

Use either a semaphore or mutex to make the two functions atomic with maximum concurrency.

**c) (10 points)** The following two processes AA and BB are executed concurrently and share (1) the semaphores S and R (each initialized to 1) and (2) the integer variable x (initialized to 0).

```
process AA()                        process BB()
{                                   {
     while (1) {                        while (1) {
       P(S);                                P(R);
       P(R);                                P(S);
       x++;                                 x--;
       V(S);                                V(S);
      V(R);                                 V(R);
    }                                   }
}                                   }
```

(1) Could the concurrent execution of AA and BB result in one or both being blocked forever? If yes, give an execution sequence in which one or both are blocked forever.

(2) Could the concurrent execution of AA and BB result in the starvation of one of them? If yes, give an execution sequence in which one is starved.

## **C2 Operating Systems** (25 points)

**a) (13 points)** Given a demand-paging memory management system with the following parameters:

page size = 4K = 2^12 bytes
physical address space = 2^24 bytes
logical address space = 2^32 bytes
TLB size = 2^6 bytes

**(1) (2 points)** How many bits are needed to specify a logical address? Give the number of bits for the page number and the offset.

**(2) (3 points)** How many page table entries can fit in the TLB if each entry only contains the information needed for logical to physical translation.

**(3) (3 points)** How many page table entries are needed when a program uses its full logical address space?

**(4) (5 points)** Part (3) indicates a serious problem that arises from having a very large logical address space. What is this problem and how could an OS solve it? Succinctly discuss the consequences of your solution for runtime overhead during program execution.

**b) (12 points)** Consider the following program.

```
#define Size 64
int A[Size,Size], B[Size,Size], C[Size,Size];
int register i, j;
for (j = 0; j < Size; j ++)
  for (i = 0; i < Size; i++)
    C[i,j] = A[i,j] + B[i,j];
```

**(1) (5 points)** How frequently would page faults occur (in terms of number of times the statement C[i,j] = A[i,j] + B[i,j] are executed).

**(2) (7 points)** Modify the program to minimize the page fault frequency, and show the frequency of page faults after your modification.

## C3 Operating Systems (25 points)

a) **(6 points)** Consider this list of processes:

| Process | Arrival | Length |
|---|---|---|
| p1 | 1 | 18 |
| p2 | 2 | 5 |
| p3 | 5 | 15 |
| p4 | 10 | 10 |

Show each process as it enters, executes, waits, and leaves the system for scheduling based on First Come, First Served (FCFS), Round Robin, quantum=3 (RR Q=3), RR Q=5, and Shortest Remaining Time First (SRTF). You have the option of writing a timeline, a Gantt chart or simply the time when each process finishes execution. The downside of the latter is that if you have an error in your calculations, there is no way to get partial credit.

b) **(5 points)** What is the major problem with priority based scheduling algorithms. What modification to an algorithm can address this problem?

c) **(5 points)** What factor or factors should be addressed when selecting a scheduling quantum?

d) **(5 points)** What factors motivate preemptive vs. non-preemptive scheduling?

e) **(4 points)** What is processor affinity? Is it still applicable for multicore processors? Why or why not?

**C4. Operating Systems** (25 points)

a) **(9 points)** What is a system call? Describe the steps involved in one.
b) **(8 points)** What is a Virtual Machine? How is a VM implemented? What are its benefits and drawbacks?
c) **(8 points)** What is a device driver? What is an interrupt hander (or interrupt-handler routine)? Describe both in terms of implementation and functionality. How are they related?