1. (25 points, all or nothing) In the CPU.circ you designed for p4, how did your circuit deal with subtraction opcodes need to subtract? Describe your components, wiring, and reasoning in enough detail so that we can determine that you actually designed your circuit. This may only take two or three sentences. You may use a drawing to help with your explanation.

To handle subtraction, I negated the second operand by creating its 2's complement. I did this by having MUX guide the B input into inverter to create a 1's complement, and having the Cin for my first bit’s ALU unit set high if there was a subtraction. The subtraction wire was set by an OR gate of the opcode decoder outputs for the two subtraction opcodes.

2. (10 points) What is the goal of the memory hierarchy? What principle makes it possible to achieve this goal? Give the two types, and explain what they are.

The goal is to have the overall performance close to that of the fastest device with the cost close to the cheapest device. The principle of locality states that only a small subset of a program's instructions and data near/in the CPU are accessed any given time, and that subset tends to change gradually. Inboard memory is typically volatile and built of semiconductor materials, and includes registers, caches, and main memory (RAM). Outboard memory is non-volatile and utilizes magnetism or burned materials to store the data, and includes CDs, magnetic disk drives, and DVDs.

3. (10 points) What is Cache Coherence, and why does it matter? Is it a concern only in parallel computers? Why or why not?

Cache coherence is the consistency of copies of the data from the same address in main memory. If there is inconsistency, then a process that relies on the outdated data could make a miscalculation. Even sequential computers require coherence because devices other than the CPU read from main memory, e.g. DMA devices.

4. (10 points) Caches can be either Virtually Addressed or Physically Addressed. Explain the difference, and give one advantage and one disadvantage to using Virtually addressed caches.

Virtually addressed caches rely on the logical addresses used by the machine code that assumes that the program starts at location zero. Physically addressed caches rely on the real addresses in RAM of the data. Virtually addressed caches do not have to spend time waiting for the memory management unit to convert logical addresses to physical addresses. However, since the logical address spaces of all the programs overlap, the cache must either be purged with each context switch, or each cache line must have extra information to differentiate the processes associated with them.

5. (51 points) Consider a memory system that uses a 24-bit address, is byte addressable, and each line in cache holds 16 bytes of data.

a) (18 points) Assume a direct-mapped cache with a tag field in the address of 12 bits. Answer the following questions. Please state the address format in terms bit ranges, e.g “block: 0-5” where 0 is the LSB. Showing your work may earn you partial credit.

1) (3 points) Number of lines in the cache: \(2^{24-12} = 2^8 = 256\).
2) (3 points) Size of the cache (in data bytes): \(256 \times 16 = 4096 = 4\)KB
3) (3 points) Size of the tag in bits: \(12\) bits
4) (3 points) Address format: Tag: 12-23, Line #: 4-11, Block: 0-3.
5) (6 points) We are trying to read a byte with address 0x49BE51. What line number(s) (in hex) would we look at? 0xE5

If the byte is the cache, what is the tag (in hex) of its line? 0x49B

b) (12 points) Assume a fully-associative cache.

1) (3 points) How big is the tag in bits? 24 – 4 = 20 bits.
2) (3 points) Address format: Tag: 4-23, Block: 0-3.
3) (6 points) We are trying to read a byte with address 0xAB1483. What line number(s) would we look at? All of them

If the byte is the cache, what is the tag (in hex) of its line? 0xAB1488

c) (21 points) Assume a 2-way set associative cache with 8K lines. Answer the following questions. Please state the address format in terms bit ranges. Showing your work may earn you partial credit.

1) (3 points) Size of the cache (in data bytes): \(8K \times 16 = 128\)KB
2) (3 points) Number of lines per set: 2
3) (3 points) Number of sets in the cache: \(8K/2 = 4K = 2^{12}\)
4) (3 points) Size of the tag: 24 – 12 - 4 = 8 bits
5) (3 points) Address format: Tag: 16-23, Set #: 4-15, Block: 0-3.
6) (6 points) We are trying to read a byte with address 0xD2F456. What set number(s) (in hex) would we look at? 0xF45

If the byte is the cache, what is the tag (in hex) of its line? 0xD2

(6 points) A processor has an 16-bit data bus with the read bus timing shown below. Assuming a processor clock rate of 8 MHz, what is the maximum data transfer rate?

The clock period is 1/8 MHz = 125 ns. One bus read cycle takes 4 * 125 = 500 ns = 0.5 µs. If the bus cycles repeat one after another, we can achieve a data transfer rate of (2 bytes/transfer) / (0.5 µs / transfer) = 4MB/s.
7. (15 points) What is the maximum clock frequency possible for the following circuit? (In other words, what is the maximum clock frequency that will still guarantee correct behavior?) Use the following delay values, and assume all input signals become valid at time 0:

AND: 4ns  NAND: 3ns  NOT: 2ns  MUX: 5ns
Tprop (TFF): 7ns  Tsetup (TFF): 3ns  Thold (TFF): 1ns
Tprop (DFF): 8ns  Tsetup (DFF): 3ns  Thold (DFF): 1ns

Note: You must show the path in order to get credit.

\[ \text{Tprop(DFF)} + \text{NOT} + \text{AND} + \text{AND} + \text{MUX} + \text{Tsetup} = 8 + 2 + 4 + 4 + 5 + 3 = 26 \text{ns} \] so frequency = \( \frac{1}{26 \text{ns}} \)
8. (15 points) Given the following program fragment, only 4 data registers in the CPU, and 32-byte data cache with 4-byte memory blocks. Thus the cache has eight lines. Assume addition is done from left to right in the CPU. Assume all communication with RAM must be through the cache, and cache misses take quite a while. Assume LRU replacement for two-way set associative mappings

Which of the two cache mapping functions (direct, or two-way set associative) would make the program for-loop fastest and why? Be sure to explain why the other mapping function would cause the program to be slower. I used four sentences to answer.

```c
int i, b[1020]; // 4-byte ints.
... // b is initialized
for(i = 0; i < 1000; i++)
    b[i] = b[i + 1] + b[i + 2] + b[i + 3] + b[i + 4];
```

**Direct caching would be fastest.** Since the ints are consecutive, there would not be any cache line collisions for either of the mapping functions to have the needed elements for a given i summation. However, direct mapping has the fastest finds for a given address within the cache. Direct need only check the one tag, while two-way associative needs to look at two.