1. (6 points) Consider a virtual memory system.

(i) (2 points) Explain the difference between a virtual address and a physical address.

(ii) (2 points) Explain what is meant by a page fault.

(iii) (2 points) Briefly describe the role played by a TLB (Translation Lookaside Buffer).

2. (4 points) Consider the IEEE 754 floating-point format, given below:

\[ \text{Value} = (-1)^{\text{Sign}} \times \text{Significand} \times 2^{\text{Exponent}} \]

(i) (2 points) In this format, the Significand is usually represented in normalized form. Explain what this means and illustrate with a simple example.

(ii) (2 points) The Exponent is represented in the excess format. Explain what this means and why the excess format is preferrable to the 2's complement representation.

3. (6 points) The frequency of occurrence of different types of instructions in a program is given in the following table. The table also gives the average CPI (Cycles Per Instruction) for these instruction types in a computer system.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>Load/Store</td>
<td>30%</td>
<td>4</td>
</tr>
<tr>
<td>Branch</td>
<td>10%</td>
<td>3</td>
</tr>
<tr>
<td>Floating-point</td>
<td>10%</td>
<td>10</td>
</tr>
</tbody>
</table>

(i) (2 points) Calculate the average CPI observed when running the given program in the given computer system.

(ii) (2 points) How much faster would the given program run in the given computer, if the CPI of the load/store instructions is reduced to 2 cycles (with the help of advanced caching techniques)?

(iii) (2 points) If we use an optimizing compiler to remove 50% of the ALU instructions (while
keeping the number of other instructions the same), how much faster would the optimized program run (compared to the unoptimized program)? Assume that load/store instructions have a CPI of 4 cycles, as in part (i).

4. (4 points) With the help of an example code snippet, explain what is meant by \textit{data forwarding} in a pipelined processor. Explain why data forwarding may not always be beneficial.
1. (6 points) Consider a virtual memory system.

(i) (2 points) Explain the difference between a virtual address and a physical address.

A virtual address is the memory address generated by the CPU before accessing the memory management unit (MMU), and refers to a location in the virtual address space. A physical address is the memory address obtained after translation by the MMU, and refers to a location in the physical address space.

(ii) (2 points) Explain what is meant by a page fault.

A page fault refers to the situation where the MMU attempts to perform a virtual address to physical address translation and determines that the referenced virtual page is not currently mapped to any physical page frame. A page fault causes a hardware interrupt, transferring control to the operating system.

(iii) (2 points) Briefly describe the role played by a TLB (Translation Lookaside Buffer).

In modern computer systems, the virtual address space is quite large, necessitating the use of very large page tables. Such large page tables cannot be physically located in the processor chip, and are therefore mapped to the memory address space. In order to speed up address translations in such situations, a spacial cache memory called the translation lookaside buffer is kept in the processor chip to store the recent address translations. For every address translation, the MMU performs a lookup in the TLB; if there is a hit in the TLB, then the page table is not accessed, thereby speeding up address translation.

2. (4 points) Consider the IEEE 754 floating-point format, given below:

\[
\text{Value} = (-1)^{\text{Sign}} \times \text{Significand} \times 2^{\text{Exponent}}
\]

(i) (2 points) In this format, the Significand is usually represented in normalized form. Explain what this means and illustrate with a simple example.

In the floating-point notation, a number can be written in many ways. For example, \(0.00101_2 \times 2^3\), \(1.012 \times 2^0\), and \(10100_2 \times 2^{-4}\) all denote \(1.01_2\). To reduce this profusion of equivalent forms, a specific form is defined to give every floating-point number a unique representation. This process is called normalization. Normalization fixes the position of the radix point of the Significand relative to its most significant non-zero bit, such that the radix point immediately follows the most significant non-zero bit.

\(e.g.\) \(1.0102_2 \times 2^3\)
Thus, the normalized Significand is a fraction that can have values between 1 and almost 2.

(ii) (2 points) The Exponent is represented in the *excess* format. Explain what this means and why the *excess* format is preferrable to the 2’s complement representation.

*In the IEEE floating-point format, the 8-bit exponent value is represented in excess 127 format. This means that the bit patterns stored in the Exponent field represents a value that 127 more than the actual exponent value. The excess format is preferrable, because it makes it possible to use an integer comparator to compare floating-point numbers.*

3. (6 points) The frequency of occurrence of different types of instructions in a program is given in the following table. The table also gives the average CPI (Cycles Per Instruction) for these instruction types in a computer system.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>Load/Store</td>
<td>30%</td>
<td>4</td>
</tr>
<tr>
<td>Branch</td>
<td>10%</td>
<td>3</td>
</tr>
<tr>
<td>Floating-point</td>
<td>10%</td>
<td>10</td>
</tr>
</tbody>
</table>

(i) (2 points) Calculate the average CPI observed when running the given program in the given computer system.

\[
\text{Average CPI} = 0.5 \times 1 + 0.3 \times 4 + 0.1 \times 3 + 0.1 \times 10 = 3
\]

(ii) (2 points) How much faster would the given program run in the given computer, if the CPI of the load/store instructions is reduced to 2 cycles (with the help of advanced caching techniques)?

\[
\text{Average CPI} = 0.5 \times 1 + 0.3 \times 2 + 0.1 \times 3 + 0.1 \times 10 = 2.4
\]

\[
\text{Speedup} = \frac{3}{2.4} = 1.25
\]

(iii) (2 points) If we use an optimizing compiler to remove 50% of the ALU instructions (while keeping the number of other instructions the same), how much faster would the optimized program run (compared to the unoptimized program)? Assume that load/store instructions have a CPI of 4 cycles, as in part (i).

\[
\text{Average CPI} = \frac{1}{0.75} (0.25 \times 1 + 0.3 \times 4 + 0.1 \times 3 + 0.1 \times 10) = 3.67
\]

\[
\text{Speedup} = \frac{3}{3.67 \times 0.75} = 1.091
\]

4. (4 points) With the help of an example code snippet, explain what is meant by *data forwarding* in a pipelined processor. Explain why data forwarding may not always be beneficial.

*Consider the following code snippet:*
R2 <-- R1 + R2
R3 <-- R2 - R4

There is a true data dependency between the two instructions through register R2. When a pipelined processor simultaneously executes these two instructions, there is a possibility of a RAW (read-after-write) hazard occurring. That is, the second instruction might try to read register R2 before the first instruction has updated R2.

In many pipelined data path implementations, the data value to be read (from R2) may be already available in the data path, for instance at the output of the ALU. If this data is directly forwarded to the second instruction, the data hazard can be overcome without stalling the pipeline.

There are situations in which data forwarding may not be beneficial. First, if the first instruction produces the required data towards the end of the pipeline, then the second instruction has to stall. Second, data forwarding may increase the clock cycle time, thereby decreasing the overall performance.