1 (Register Renaming) Instead of using a reorder buffer (ROB), one alternative approach for hardware speculation is the explicit use of an extended set of physical registers combined with register renaming. The architectural destination register of each instruction is mapped to a free physical register in the extended register set. A hardware renaming table is used to keep track of the mapping between two register sets. Unlike the rather straightforward allocation strategy, the deallocation of physical registers is more complicated: a physical register can only be freed up if it no longer corresponds to an architectural register and no further uses of the physical register are outstanding. On the other hand, the deallocation of physical registers should also be performed as early as possible, since instruction fetch will be stalled if no free physical registers are available.

(Part A) The simplest approach that a set of engineers could come up with is “checking the source registers of all the in-flight instructions”. The manager is concerned about whether this checking works or whether it will end up in a product recall. Could you help the manager figure out the correctness of this approach? Give a brief explanation for your decision.

(Part B) Another group has suggested that for an in-order commit processor, a physical register can be deallocated once a subsequent instruction which writes the same architectural register has been committed. Please comment on the correctness of this technique. If you think it is not correct, please provide a reasoning as to why. Otherwise, please provide a discussion of any possible shortcomings.

(Part C) The company is discussing how to handle physical register deallocation for out-of-order commit processors. One suggestion is to just use the deallocation strategy in (Part C), that is, to free up a physical register when a subsequent instruction writing the same architectural register has been committed. Please comment on the correctness of this technique. If you think it is not correct, please provide a reasoning as to why. Otherwise, please provide a discussion of any possible shortcomings.

(Part D) Another engineer is arguing that for out-of-order commit processors, the compiler should provide hints to the hardware in physical register deallocation. He has furthermore suggested two possible hints:

- Marking the last use of each destination architectural register.
- Providing a count of the subsequent use for each destination architectural register.

Would these hints be helpful in developing a cheaper deallocation strategy that works correctly under any conditions? Would these hints be helpful for some DSP codes with no branches? For each hint, if you think the hint is helpful, please give your corresponding deallocation strategy. Otherwise, please give your reasoning as to why it is not helpful.
2 (Misprediction Recovery) A crucial issue in a processor with speculation is the recovery from a mispredicted branch. In general, this recovery can be performed by clearing the ROB entries appearing after the mispredicted branch, allowing the entries before the branch to continue, and restarting the fetch at the correct branch successor. Meanwhile, a semantically correct misprediction recovery mechanism should be able to thread a new fetched instruction to an instruction preceding the branch which is still on the fly, if the former depends on the latter.

(Part A) Two engineers in your design team are arguing regarding the appropriate time at which the instruction fetch should be restarted upon such a misprediction. The first engineer thinks that the fetch of the correct successors should be restarted as soon as the misprediction has been detected so as to reduce the recovery time, while the second engineer thinks that the fetch cannot be restarted until all the instructions before the branch have been committed.

Which of the two engineers’ argument would you vote for? The following tables show the content of reorder buffer and the FP register status when a misprediction of the branch at Entry 5 is detected. Please use these tables as an example to briefly explain your vote.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Instruction</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>L.D F2,0(R1)</td>
<td>Commit</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>MUL.D F4,F0,F2</td>
<td>Execute</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>S.D F4,0(R1)</td>
<td>Execute</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>SUBUI R1,R1,#8</td>
<td>Write result</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>BNEZ R1,Loop</td>
<td>Execute</td>
</tr>
<tr>
<td>6</td>
<td>no</td>
<td>L.D F2,0(R1)</td>
<td>Execute</td>
</tr>
<tr>
<td>7</td>
<td>yes</td>
<td>MUL.D F4,F0,F2</td>
<td>Execute</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder #</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Part B) Assume the architecture supports speculation across multiple branches. Among all the outstanding branches, a misprediction has been detected for the latest branch, while all the preceding branches are still in execution. Please discuss the semantic correctness of the two ideas listed in (Part A) if they are to be applied in this case, and provide a brief reasoning for your statement.

(Part C) While the two groups of engineers are battling it out, a third group in the company under the leadership of their manager is advocating an alternative idea. They propose to add some additional hardware by extending the ROB to include, for each instruction that updates the FP Register status table, the old value of the corresponding Reorder # field. She claims that thus the fetch of the correct successor can be restarted as soon as possible.

Of course, numerous groups in the company are vociferously busy trying to shoot the new idea down. Here are the various viewpoints being advocated within the company.

A The technique has no impact on the restart of fetch upon branch misprediction but adds unnecessary hardware.

B The technique not only adds unnecessary hardware but also is incorrect and will definitely create semantic violations.
C The technique works but only for the first branch in a group.

D The technique works for all branches and is guaranteed to deliver consistently semantically correct results.

Please decide which of the four viewpoints is correct and provide a brief rationale for your answer.

3 (Compiler-based prefetching) In this question, you are going to evaluate the prefetching technique for the following Control Flow Graph (CFG), which consists of 7 Basic Blocks. All the memory references within the code fragment are shown in the CFG. The CFG also shows the T/NT probabilities, generated by static profiling, of both branches.

Assume the program is going to be executed on a particular processor with the following memory configuration: L1 cache is a direct-mapped cache with 1 word/block, 6 bit index, and 1 cycle hit time. On a cache miss due to a load instruction, the memory would need to be accessed, which takes an additional 20 cycles. On the other hand, a cache miss of a store instruction will not block the processor.

Because of the significant cache miss penalty, you are considering adding an explicit prefetch instruction to reduce the miss rate. If the word to be prefetched is already in the cache, the prefetch instruction will transform itself into a noop.

For the following parts of this question, assume the starting addresses (in byte) of arrays A[], B[] and C[] to be 260, 0, and 400, respectively. All the data stored in A[], B[] and C[] are 32-bit integers. The loop body itself is going to be executed 30 times.
(Part A) Since the basic block B4 is on the most frequently executed trace, you want to add an explicit prefetch instruction for the load of A[i] in B4. Since the prefetch instruction needs to be inserted at least 20 cycles ahead, the only possibility is to insert it in B1 (prefetch instructions across loop iterations are not considered). However, it is also possible that sometimes the insertion of prefetch instructions may cause performance degradation. For this particular program, please analyze whether the inserted prefetch instruction will cause performance degradation or not. How much speedup (or degradation) can be achieved if the prefetch instruction is going to be inserted? Your answer should take into account the T/NT probabilities of branches.

(Part B) Since inserting a prefetch instruction for A[i] may cause performance degradation, another possible optimization is suggested to reduce the execution time of this particular loop. Instead of inserting a prefetch instruction in B1 for the load of A[i] in B4, a prefetch instruction is inserted in B1 for the load of B[i+1] in B5. Compared with the idea in (Part A), is this idea better or not? Compared with the original code, how much speedup (or degradation) can be achieved if this prefetch instruction is going to be inserted? Your answer should take into account the T/NT probabilities of branches.

4 (Skewed Associative Caches) One technique for reducing the number of conflict misses is to use skewed associative caches that employ distinct indexing mechanisms for various ways. For example, in a 2-way skewed-associative cache, way 0 is indexed using the least significant address bits, the same as in a regular cache. The indices of way 1, however, are generated through XORing the original indices with a set of bits selected from the tags. In this way, data originally mapped to the same cache line can be mapped to different lines in way 1, thus reducing the number of conflicts.

(Part A) In a 2-way skewed associative cache design, a crucial issue is the selection of suitable XORing functions for accessing way 1. Specifically, assume that the 32-bit address is partitioned into 3 segments, 4-bit block offset, 10-bit index, and 18-bit tag. Furthermore, it has been found that the program only accesses data within the following two disjoint address ranges: \([0002 0000: 002F FFFF]\) and \([1040 0000: 107F FFFF]\).

To effectively reduce potential conflict misses, which part of the tag segment is more appropriate to be selected for XORing; the most significant bits or the least significant bits? Please clearly state your choice and briefly explain your reasoning.

(Part B) A widely used replacement algorithm in a 2-way associative cache is the least-recently-used (LRU) algorithm. LRU can be simply implemented through adding a single bit to each cache line; the bit is set to 0 if way 0 is accessed most recently, and to 1 if way 1 is accessed most recently.

Can LRU replacement policy be attained using a single bit in a 2-way skewed associative cache? If you think it can, please give the mechanism for setting and resetting the bit. Otherwise, please give a brief reasoning regarding why LRU cannot be attained.