**Structural** **Hazards** – Hazards, pertaining to hardware limitations in pipelined processing, in which instructions that require multiple processing on a single stage must prevent previous stages within the cycle from sending information further down the pipeline until the multiple processing requiring instruction has completed its current stage. **Solutions** – More hardware such as additional data buses and ALUs can be added to facilitate the concurrency of multi-processing requiring instructions. In addition, bubbles or NOPs can be introduced when an instruction requires multiple processings at a stage so as to allow the instruction more time to process.

**Data** **Hazards** – Data hazards involve instructions seeing data in registers or memory that has not been fully updated by the instructions preceding it. Read after write (RAW), write after read (WAR) and write after write (WAW) are types of data hazards.**Solutions** – One solution for RAW hazards is a hardware solution involving sending register data that is updated by a preceding instruction directly to the following instruction that must read data from that register. Another solution is for the compiler to rearrange instructions so that two instructions, one writing to a register and the other reading from it, are not sequential. A third solution is to add bubbles via adding a hardware feature that prevents previous stages from passing along its processed data, but rather passes a bubble/NOP so as to allow more time for the write process to finish its actions (so that in the following cycle, the read instruction reprocesses, but this time with the updated register data).

**Control Hazards** – Branch instructions require pipelining to process one set of sequential instructions over another. This causes problems when the pipelined instructions are not for the branch that the program flow ends up taking. **Solutions** – The compiler solution to this problem is to add instructions that must execute regardless of what branch the program flow takes. However, due to the large number of instructions that can be pipelined in modern processors, this methodology is untenable and requires a lot of reworking of compilers as architectures advance. Another solution is to anticipate what flow a branch will take the program down on. If a branch instruction forms a part of a loop, the instructions within the loop are anticipated for pipelining. If the branch forms part of a conditional, the instructions following a no-jump are anticipated for pipelining. The anticipated instructions are communicated to the processor via the conditional branch opcode used. Another solution is to keep a history based table of branch/no-branch program flows for branch instructions and pipeline instructions based on past pipelinings.

**2. How does the processor and the device work together in creating/handling interrupts?**

A flag is set upon a device requesting CPU time. After the completion of each atomic opcode instruction the CPU, as part of the instruction cycle, checks for set device flags. Upon finding a set flag, the CPU takes the vector for the interrupting device and consults the interrupt vector table for the appropriate handler to deal with the device. The interrupt service routine will disable interrupts for the sake of saving the current state of the processor (PC, register states) and then re-enable interrupts, when the current run level is saved to the system stack, the mode is shifted to kernel and the device handler code is executed, so as to allow for nested interrupts. Upon completion of the preceding, interrupts are disabled when restoring the context of the original interrupted process and executing an iret instruction to get back to the execution of the originally interrupted process. Afterwards, interrupts are turned on once again.

**5. What are the primary considerations of cache design?**

Design considerations for cache design include: spacial and temporal locality (actual storage vs. overhead),

hit, miss rates and miss penalties, levels of cache, cache type, direct-mapped, fully associative, set-associative, cache replacement policy

**1. What is the reason to have set-associative cache?**

Direct mapped cache runs into issues where memory locations of a process map to similar cache locations and thus the cumulative cost of miss penalties causes issues for pipelining.. Fully associative caches are fairly expensive to implement and are impractical for large caches. Set associative caches allow for the avoidance of collision of physical memory address mappings in cache while not being as expensive to implement as fully associative caches.

**2. Explain why L1 caches usually have lower associativity compared to L2 caches.**

L1 cache is typically configured for fast access as opposed to L2 cache, which is typically geared for higher hit probability. Having a high level of associativity allows for physical addresses that would more likely conflict (on a more direct memory mapped cache) to cohabitate within L2 cache, increasing hit probability.

1.**What is the total disk capacity?**

256 bytes/sector \* 50 sectors/track \* 100 tracks/surface \* 200 surfaces = 256,000,000 bytes

**What is the average rotational latency (see p.446 of textbook)?**

60,000 milliseconds/minute / 2400 rotations/minute \* 0.5= 12.5 milliseconds/rotation

**2. A disk has 20 surfaces (i.e. 10 double sided platters). Each surface has 1000 tracks. Each tracks has 128 sectors. Each sector can hold 64 bytes.**

**a.) How many bytes are contained in one cylinder?**

64 bytes/sector \* 128 sectors/track \* 20 tracks/cylinder= 16,384 bytes/cylinder

**3. Describe in detail the sequence of operations involved in a DMA data transfer.**

The CPU must store within the device controller registers the: command, address of the device, the memory buffer address and the amount of data that will be transferred (count register).

The device controller moves the specified amount of data (in the count register) between its buffer and the memory buffer. When the transfer is complete, the status register within the device controller will be updated. Additionally, if interrupts are enabled for the device register, the controller will signal an interrupt.