EE457 Advanced Topics Review

About 35% of the final exam will be dedicated to the 6 special topics and chapter 9 in class notes. These will be fairly straightforward questions. Please go through questions on special topics in the recent final exams and check your answers with the solutions.

Also, please go through the following quick topics review, reminding you of the significant parts of each topics.

1.1. Exceptions:
- Difference between Precise exceptions and other exceptions.
- Undefined opcode to "extend" the ISA.
- Offending instruction shall remain silent until it reaches the WB stage.
- Exceptions are taken in ________ (temporal/program) order.

1.2. Branch Prediction:
- BPB (Branch Prediction Buffer) 1-bit vs. 2-bit branch predictors,
- BTB (Branch Target Buffer),
- Correlating branches, (m,n) predictor,
- Aliasing shall be avoided if you wish to predict from the ________ (IF / ID) stage.

RAS (Return Address Stack) is a hardware stack, quite shallow, yet.. LIFO but **circular**! Do you continue to push into RAS (as a result of jal), even if it is full? **Y**

During return from a subroutine (as a result of jr), what if RAS becomes empty? Keep returning the last returned address, because it could be a recursive call! If it is during the return phase of a recursive call, returning the last popped return address, when the shallow RAS becomes empty, can prove to be beneficial.

RAS is **not** repaired or restored during branch misprediction, as it is quite expensive.

The un repaired/unrestored RAS may provide wrong return addresses for a few occasions after the branch misprediction. This is considered acceptable since return value provided by RAS is considered to be a prediction anyways. The actual return address is fetched by the JR $31 and is compared with the return address predicted by the RAS. In case of mismatch, wrong-path instructions are flushed (very much like in the case of a mispredicted conditional branch).
1.3. Out-of-Order Execution

Several multiple-choice questions equivalent to one major question.

1.3.1. FIFO design review from EE201L -- Single clock FIFO and 2-clock FIFO, and the FIFO lab. Yes, the 2-clock FIFO is included though your lab deals with only the 1-clock FIFO.

Slides .pdf  Webcast (44 minutes)  .avi  .wmv  EE457 FIFO lab .pdf  Sample questions .pdf  Also see questions from recent exams.

Slide 38/42 of the above slides-set is reproduced below. Then complete the second sheet of the 2-page pdf and check with the first sheet.

Get to know how and when DEAD LOCK can happen if a 2-clock FIFO is designed carelessly by using the n-bit pointer method (instead of the n+1 bit pointer method) in the case of any of the following two cases:

1. Producer’s clock is much faster than the consumer’s clock
   Say, $f_{WCLK} = 100MHz$  and  $f_{RCLK} = 1KHz$

2. Producer’s clock is much slower than the consumer’s clock
   Say, $f_{WCLK} = 1KHz$  and  $f_{RCLK} = 100MHz$

In both cases, the producer can end up believing that the FIFO is FULL while the consumer can end up believing that the FIFO is empty.

In such cases, the one running at faster clock knows the truth whereas the one running at slower clock is acting on obsolete information as he was sleeping for 1 milli-second while other is actively taking the small FIFO from one extreme condition (FULL or EMPTY) to another extreme condition (EMPTY or FULL).
1.3.2. Out-of-Order Execution with out-of-order completion (Tomasulo algorithm with RST but without ROB)

RAW, WAW, WAR, WAW and WAR are called name dependencies, RAW is the true dependency. How WAW and WAR problems in registers are solved through register renaming, and why it is not practical to do the same for the WAW and WAR problems in memory locations (no MST memory status table as it is too big and too slow).

Some details of register renaming: RST (Register Status Table), forwarding through (from) CDB, neither source register IDs nor destination register IDs are carried into the backend, a new TAG is allocated for the ____ (source/destination) register of each instruction. And the same TAG is conveyed to subsequent instructions if their source register ID matches with this senior instruction's destination. This goes on until
(a) the same register is used as a destination by another junior instruction or
(b) the original senior instruction has completed causing removal of his TAG from the RST.

TAGs need not be issued in any specific order, no virtual queue is formed by TAGs, TAGs are just unique Tokens

TAG FIFO: Is FIFO necessary or is it used for convenience?

Dispatch unit, Dispatch is halted after a branch is issued until it is resolved.

But once the branch is resolved, dispatch continues and hence it is possible that some of the instructions upstream of a conditional branch may coexist with some instructions downstream of a conditional branch.

Issue Queues: is it necessary or desirable to maintain instructions in the order of arrival? Necessary for LSQ, desirable for the rest of the queues.

Purpose of the Issue Unit is to manage traffic on CDB. Does the Issue Unit desire that every execution unit is a fixed-latency execution unit? **Yes!** Then how about the lw instructions, which may incur a cache miss?

That is why we have placed **Load Buffer** after the cache!
Memory disambiguation rules

Instructions in loops and how instructions from different iterations of the loop can possibly co-exist in the backend. (Loop unrolling occurs dynamically)

1.3.3. the ROB lab, Part 2.pdf (excluded)

1.3.4. Out of Order Execution with in-order completion (Tomasulo algorithm with ROB)

Difference between completion of execution and retirement (commitment). This design is capable of supporting precise exceptions. Yes / No
No RST here, because of difficulty in restoring (repairing) RST in the case of a branch misprediction.
No Tag FIFO. In place of the Tag from Part I of Tomasulo, we use the ID of the ROB slot pointed to by WP during Dispatch (called ROB Tag).

Each instruction including sw (store word which does not have a destination register) is allocated a ROB TAG.

A virtual queue is formed (among the instructions in the back-end) because of the ROB slots associated with the dispatched instructions.

ROB is a FIFO, which is a circular buffer. First-in-first-out in ROB means in-order completion!

ROB slot is allocated to a new instruction by the ___________ (Dispatch Unit/Issue Unit).

The ROB Tag allocated to a new instruction is ___________ (WP/WP+1/WP-1/RP/RP+1/RP-1).

At the committing end, the instruction pointed to by the ___________ (WP/WP+1/WP-1/RP/RP+1/RP-1) in ROB is allowed to commit if it has completed execution in the backend.

Upon completion of execution, an instruction comes on CDB and joins ROB at ___________.

To do so, we need a random-access port to the ROB for _______________ (reading/writing/reading as well as writing).

ROB search for the *youngest* senior (junior-most senior) instruction with destination register ID matching with the source ID of the instruction being dispatched. The ROB search is _______________ (a sequential search / an associative meaning parallel search). To perform prioritized parallel search, say for $2, in a 32-location circular ROB, we used ___________ (1/2/3/4) 32-input fixed priority resolvers.

Speculative execution: Based on branch prediction, instructions after branch (either at Target or at fall-through based on prediction) are dispatched. A series of branch predictions can happen and branches _____ (1/2) 1. may get resolved in out of order 2. have to be resolved in-order only.

In case of a misprediction, wrong-path instructions are flushed. All instructions younger to the mispredicted branch instruction are called wrong-path instructions.

In the case of branch misprediction it is direction (T or NT) misprediction. In the case of JR $31, it is target address misprediction. The “other destination address” (which becomes the correct destination address) is given to a beq instruction by the dispatch unit at the time of dispatch so that the dispatch unit does not have to maintain a separate record of where it should branch to if the predicted branch is mispredicted. The mispredicted branch comes on CDB and announces the other address along with the “mispredicted” announcement. In the case of a JR$31, the other address is the actual content of the $31 that the JR $31 reads when it goes through back-end. So, the dispatch unit turns to the RAS for help only if the $31 value is not readily available from the ROB.

"Who is younger to the branch instruction or the JR $31" is inferred by computing the distance of all instructions with respect to the senior-most instruction (pointed to by the ___ (WP/RP)).

Distance = (ROB tag of the instruction - RP) mod_32 (mod_32 if 32 is the depth of the ROB *and* if we use a 5-bit ROB tag).

In EE201L and in the FIFO lab, we taught two methods of distinguishing the EMPTY state of the FIFO from the FULL state of the FIFO.

For the 32-location FIFO (our ROB), we use 5-bit pointers for WP and RP and perform depth calculation by doing the mod-32 subtraction:

Depth = [WP - RP] mod 32

However, when [WP == RP], the ROB can be empty or FULL. We can use a separate Flip-Flop to record whether the ROB was most recently running
Almost-Full or Almost-Empty and use this information to interpret \([\text{WP} = \text{RP}]\) as indicating FULL or EMPTY respectively.

Another method is to use \((n+1)\)-bit pointers. For example, for the 32-location ROB, we could use 6-bit pointers for WP and RP. Then we need to perform \(\text{mod}_64\) (note: \(\text{mod}_64\) and not \(\text{mod}_32\)) subtraction to calculate the depth: \(\text{Depth} = [\text{WP} - \text{RP}] \mod 64\).

In this case, \(\text{WP-RP} = 000000\) represents EMPTY and \(\text{WP-RP} = 100000\) represents FULL.

RAS (Return Address Stack): Even a 4-deep hardware stack (RAS) is able to predict return addresses fairly well.

RAS is *not* repaired, when you flush a bunch of wrong-path instructions, which may include some jal and jr instructions, which may have caused some pushes to and some pops from the RAS respectively.

1.4. Chapter 9 and cache coherency protocols (MSI, MOESI)

OLD method to avoid RMW race: Make all shared variables such as the Student Data Base Lock (SDBL) uncacheable on the other side of the bus. Use a LOCK signal on the bus to assist in making RMW an atomic operation. So why "this OLD method of locking the bus" is not a good idea for creating mutual exclusion among the 4 threads of the four single-threaded cores on the side? Answer: It causes undesirable traffic on the bus due to busy polling (also called spinning) by other threads (other threads from other cores as well as other threads from the same core in multi-threaded cores. So we want the SDBL also cacheable. We will use cache coherency protocols together with LL and SC instructions to implement mutual exclusion between all threads of all cores.

Moreover, a BUS is no more used to interconnect processors in a shared memory multiprocessor system. In EE557, you will be taught a variety of MINs (Memory Interconnection Networks).

Notice that allowing shared variables such as SDB_Lock (Student Data Base Lock) to be cached into private L1 caches reduces traffic on the bus (or memory interconnection network) to L2 cache, which saves clocks and reduces congestion on the bus (or memory interconnection network).

Also note that cache coherency alone cannot provide the needed mutual exclusion between competing threads which are competing simultaneously to set a lock (such as the SDB_Lock semaphore above). Cache coherency along with LL and SC instructions provide the needed Mutual Exclusion in Multiprocessors.

In write-back cache, with cache coherency protocol in place, flushing of dirty blocks to the MM (FMM = Flush to the MM) is postponed as far in time as possible. M-to-I in one L1 cache to allow I-to-M in another L1 does not cause FMM as the “updating MM responsibility” is transferred from the first L1 to the second L1 cache. This is to reduce traffic on the bus (or the memory interconnection network).

There will be a question in the final exam on Cache coherency in a multi-processor system (i.e. multi-core system). Practice reading MSI and MOESI (and MOOESI of EE560) protocol state diagrams and be in a position to complete a few missing state transition arrows and/or a few missing state transition conditions. If you are given the same diagram drawn in a different format, see if you get confused or will still be able to draw missing lines and missing state transition conditions.
1.5. CMP/CMT

OS controlled process switching (software multithreading) vs. hardware multithreading,

Non-Blocking Cache, division of duties between CCU and SCU, MSHRs (Miss Status Handling Registers), RMHRs (Read MSHRs) and WMSHRs (Write MSHRs)

Dynamic Power makes CMP a preferred choice over uniprocessor with ramped-up frequency.

Impact on CPI because of MPI (Miss Rate Per Instruction) (per instruction includes all instructions such as ADD and SUB besides LW and SW), i.e. impact of MPI of L1 on CPI and impact of MPI of L2 caches on CPI.

\[
CPI = 1 + 0.05 \times 20 + 0.01 \times 300 \\
= 1 + 1 + 3 = 5
\]

Example: Intel Skylake Core i7 processor of year 2015

4 core, each core with 32KB 1.Cache, 32KB D.Cache for L1, and 256KB unified cache for L2. L3 of 8 MB is common to all the cores.

Which of the two below incur high overhead
(i) the software multithreading (context switch by the operating system)
(ii) the hardware multithreading (thread switching is through thread-selection stage)

Is "frequent context/thread switch" common with the software multithreading or the hardware multithreading?

Describe/identify (a) coarse-grain multi-threading (b) fine-grain multi-threading (c) SMT Simultaneous multi-threading
Is it true that "rollback due to cache miss" happens only in two of the three types of multi-threading? Yes, only in coarse grain and fine grain (which are used in IoI-IoE-IoC) but not in SMT (which is used in IoI-OoE-IoC).

Intel's HTT (Hyper Threading Technology) is essentially same as ___________ (fine-grain / coarse-grain / simultaneous) multithreading.

1.6. Locks for mutual exclusion, synchronization, barrier synchronization,

ISA support for synchronization: instructions LL and the SC in MIPS.

http://www-classes.usc.edu/engr/ee-s/457/EE457_Classnotes/EE457_Chapter9/EE557_Synchronization_slides.pdf

2.8 ISA support (example: LL and SC in MIPS) is necessary to provide mutual exclusion (circle all applicable) (a) between interacting processes in a multi-processing system (b) between interacting threads running on different cores (c) between interacting threads running on the same core ___________ (However / Even ) if there is only one thread per core and you have established MOESI between cores, atomic test and set ___________ (can / cannot) be guaranteed with the simple (ordinary) lw and sw instructions. In MIPs ISA, LL stands for ___________ and SC stands for ___________.

Solution to the above question

2.8 ISA support (example: LL and SC in MIPS) is necessary to provide mutual exclusion (circle all applicable) (a) between interacting processes in a multi-processing system (b) between interacting threads running on different cores (c) between interacting threads running on the same core ___________ (However / Even ) if there is only one thread per core and you have established MOESI between cores, atomic test and set ___________ (can / cannot) be guaranteed with the simple (ordinary) lw and sw instructions. In MIPs ISA, LL stands for ___________ and SC stands for ___________.

Explanation for (a) above: In an old uniprocessor system (single core single thread system), it is possible that a competing process (P0) may be suspended (because of timer interrupt) after executing the first two lines of the Naive code on the right. Now P1 becomes active completes all the lines of the code on the right, enters the critical section of the code (say database modification, before it gets to complete all the modifications that it started, timer interrupt comes and P1 gets suspended, and P0 gets started. P0 does not execute the first two lines of the code and proceeds with lines 3 and 4 and thinks it has obtained the lock and starts modifying the same database.

Explanation for case (b) above: Here again the code executing on Core_1 and Core_2 will not try to re-execute the first two lines of the naive code as shown diagrammatically below.
1.7 Final lecture: After the final lecture I will inform you through an email what part of this ee457_final_lec.pdf you will be responsible. Perhaps you need to know at minimum the following acronyms.

PRF stands for Physical Register File.
FRL stands for Free Register List.
RAT stands for Register Alias Table.
FRAT stands for Frontend RAT and is updated by the dispatch unit whereas RRAT stands for the Retirement RAT and is updated by the Instruction Retirement unit.

Also we do not know if we get to cover EE560_CMP_Design_Aspects_Simplified_for_EE457.pdf