4/27/2023

Dear EE457 students,

Final Exam (~33.5%): Wednesday, May 10, 2023, 03:30 PM - 06:30 PM PST in THH 301
Exam Hall for Zoom exam for remote students (and extra time for the OSAS students) THH 125

Final Exam is comprehensive, but it focuses on later topics (Midterm topics included). We hope to use the following two scales (as stated in the MT score sheet, the syllabus, and the plan for the first three weeks).

Quiz + MT + Final = 12+26+30 = 68
Quiz + MT + Final = 10+21+37 = 68

I assume that you had gone through your midterm exam solution and understood the same. Sometimes, I may ask a question in the final which is related to a question on the midterm.

Also please go through the two "short homework assignments" on the advanced topics.

Approximate breakdown of the Final exam:

My **desire** for the final exam is as follows:

1. Ask questions in the Virtual memory and Cache topic, weighing about 15% of the final exam.
Reproduced is an extract from your midterm preparation guide on these two topics.
====
4. Chapter #7 Cache and HW#6, you need to complete the assigned parts of HW#6
One major question and one medium question.
The following two questions are important typical questions.
Do go through them and be ready to get 100% on this topic.

4.1 Q#5 on cache organization from the Spring 2015 Midterm:
http://www-classes.usc.edu/ engr/ee-s/457/ee457_Spring2015_exams/ee457_MT_Spring2015.pdf
http://www-classes.usc.edu/ engr/ee-s/457/ee457_Spring2015_exams/ee457_MT_Spring2015_sol.pdf
4.2 Q#5 on cache mapping techniques from Fall 2010 MT.
Please go through its solution and then *time* yourself to solve it.
https://viterbi-web.usc.edu/www-classes/engr/ee-s/457/ee457_Fall2010_exams/ee457_MT_Fall2010.pdf
https://viterbi-web.usc.edu/www-classes/engr/ee-s/457/ee457_Fall2010_exams/ee457_MT_Fall2010_sol.pdf

4.3 Please look at the Q#4.1 Q#4.2 and Q#4.6 of Spring 2017 Midterm and sol.pdf on Cache.
Questions could be "reverse-engineering" type questions like in the Midterm of Fall 2015 Q#3.

9. Virtual Memory topic (normally covered after midterm but covered early in recent semesters including this semester)

9.0. This semester, I finished covering the first lecture of the two lectures on Virtual memory on 3/8/2023 and 3/9/2023. My lecture on Monday 3/20 and Tuesday 3/21 will cover the material for the 2nd lecture.
But in case you wished that the lecture is available now before the Spring break, yes, I made a copy of the Fall 2022 lecture and posted in the week #9 lectures, and it is available now on D2L!
Go to Table of Contents => Week 9 (3/6-3/10) => 2nd lecture of the two-lecture series on the "Virtual Memory" topic from Fall 2022
9.1. Q#4 from MT of Fall 2014. Please see item 7 above.
9.2. HW#7 and its solution ee457_HW7.pdf ee457_HW7_solution_r1.pdf
9.2. Please look at ee457_MT_Sp2012_VM_Ques_sol.pdf , ee457_MT_Sp2013_VM_Ques_sol.pdf ,
9.3. Please go through the Spring 2015 Final question 6.on virtual memory.
9.4. Some discussion about VIPT and PIPT

VIPT (Virtually Indexed Physically Tagged)
vs.
PIPT (Physically Indexed Physically Tagged)
The 9-stage pipeline became an 8-stage pipeline because the I_Cache_Tag_Check stage is hidden behind the instruction decoding stage.
Similarly, can you hide the D_Cache_Tag_Check stage? No! Memory write operation for SW is not like a stand-alone register writing operation, hence it cannot be aborted at the end of a clock if the D_Cache_Tag check fails. So, you cannot hide the D_Cache_Tag_Check stage.
Further if we use VIPT for both I_Tag Access, and D_Tag_Access, we can reduce the 8-stage pipeline to a 6-stage pipeline! But VIPT requires either very large virtual pages or very small L1 caches ☹. Hence you should not attempt to do VIPT to hide stages! VIPT is a concept. It may not be practical in many cases.

Degree of set associativity does not need to be a power of 2 but the number of sets shall be a power of 2. Since TLB is a cache of the PT, we can think of all three cache mappings (Fully Associative, Set-Associative, and Direct) for the TLB also. However (perhaps) Direct mapping for TLB is never used because of performance degradation due to conflicts.
Block size in TLB: Unlike in cache, the block size in TLB is always a single entry. Entries in TLB are always singular. Explanation: If you are going to Ralphs to buy milk, you may buy bread also even if you are not sure if you needed it. That is like bringing a block of 4 words where you are not sure if the other three words will be useful. But, if you are going to buy a car, you do not buy an RV (Recreation Vehicle) in anticipation that it may be useful!
But what's the analogy? Well, when you bring a page from the disc to the main memory, you just bring the page that you need; you do not bring a couple of more pages. So, if we have an entry in the PT and you could find the PPFN given the VPN, do not expect to find a block of entries (2 or 4 valid entries including the entry you went for). Hence TLB entries (which are copies of selected PT entries) are always singular!
====

2. One major design question (25%) and one analysis question (15%)
on Lab 7, Lab 6, and the single cycle CPU (total 40%)
i.e., Lab 7 all parts (i.e. Part 1 and 2 and Part 3 (SP1, SP2, SP3, SP4))
and Lab 6 Part 4 and Part 5
Look at the recent midterms and the finals
Also, please browse through EE457 Lab7 Quick familiarity test.pdf!

3. Advanced topics: (About 35% to 40% of the final) => Look at the recent finals.

4. Rest of the 10% is covered by midterm topics and other miscellaneous topics (Carry Look-ahead Adder CLA excluded).

Please make sure that you have enough rest on the night before
the exam so that you will be able to think and design during the exam.

Every semester, a few students keep awake all night, and then fail to answer even simple questions because their minds are too tired to think.
Do not let this happen to you. EE457 exam is NOT about memorization.

Office hours during the week before the final: We will announce some hours, but the TA and the Mentors have their finals too.

Good luck with your preparation.

Best of luck in all your exams,
Gandhi, Tejas, Abhipray, Shubham, Abhilash, Ziyu

Some recommendations for your preparation

I have not written the exam yet. These are only recommendations.
It is not necessary that the exam is exactly based on this.
The exam is a closed-book exam like in the pre-Covid semesters.

The exam is comprehensive. However, Chapter #1, HW #1, Lab #1 (Min/Max), Chapter #2, HW #2, Chapter #3, HW#3, Chapter #4, HW#4 (the Quiz topics) are not important for the final as we tested them adequately in the quiz/midterm. In recent semesters, we have not been covering the topic of a multi-cycle CPU. Hence chapter #5 Part 2, HW#5b, Lab #4 P4 are excluded. However, the single cycle CPU topic (chapter 5 Part 1 and HW#5a) is **not** excluded for the final exam.

0. When looking at the previous years' final exams (**very old finals**), you need to **skip** questions on the following topics as we have not been covering these lately.

CLA (Carry Look-ahead Adders)
Non-linear pipelines
Fast Multipliers based on Carry-Save adders (Wallace Tree Multiplier)

Old exams from year 2011 and before are less important and are not provided.

Go through **3 to 4 of the most recent final exams** (Fall 2022 and before) and their solutions listed on D2L => Past Exams (which are also listed below) by solving each question mentally and then going through the solution. Then try to time yourself and answer one or two **design questions** from exams that you did not go through.

Past Final Exams and Solutions:

Fall 2022 First go through **Q#1** from the midterm, and then through **Q#1** of the Final.
ee457 Final Fall2022.pdf
ee457 Final Fall2022 sol.pdf
ee457 MT Fall2022.pdf
ee457 MT Fall2022 sol.pdf

ee457 Final Sp2022.pdf **Skip Q#1**
ee457 Final Sp2022 sol.pdf **Skip Q#1**

ee457 Final Fall2021.pdf
ee457 Final Fall2021 sol.pdf

ee457 Final Sp2021.pdf **Skip Q#1**
ee457 Final Sp2021 sol.pdf **Skip Q#1**

ee457 Final Fall2020.pdf **Skip Q#1**
ee457 Final Fall2020 sol.pdf **Skip Q#1**

ee457 Final Sp2020.pdf
ee457 Final Sp2020 sol.pdf

ee457 Final Fall2019.pdf
ee457 Final Fall2019 sol.pdf

ee457 Final Spring2019.pdf
ee457 Final Spring2019 sol.pdf

ee457 Final Fall2018.pdf
ee457 Final Fall2018 sol.pdf
========== Focus on the above 9 exams.

ee457 Final Spring2018.pdf (skip Q#2 on addition/subtraction overflows)
ee457 Final Spring2018 sol.pdf

ee457 Final Fall2017.pdf (skip Q#3 on a special processor for military, lengthy question)
ee457 Final Fall2017 sol.pdf

ee457 Final Spring2017.pdf
ee457 Final Spring2017 sol.pdf

ee457 Final Fall2016.pdf (skip Q#6 on CLA)
ee457 Final Fall2016 sol.pdf

ee457 Final Spring2016.pdf
ee457 Final Spring2016 sol.pdf

ee457 Final Fall2015.pdf (skip Q#2)
ee457 Final Fall2015 sol.pdf (skip Q#2 as it is a clumsy design to make it a special processor for CIA)

ee457 Final Spring2015.pdf (skip Q#1.2 on ROB as we did not do the ROB lab)
ee457 Final Spring2015 sol.pdf

~~ee457 Final Fall2014.pdf~~ (Skip Q#4 as it is difficult. Moreover, I stopped asking questions involving a branch delay slot together with a cache miss ICMT and DCMT)(skip Q#5 on CLA)
~~ee457 Final Fall2014 sol.pdf~~

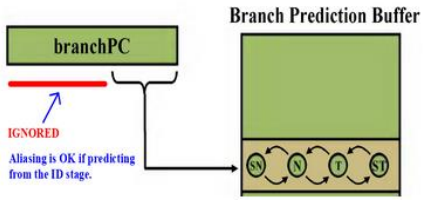
ee457 Final Sp2014.pdf (skip Q#4 on CLA)
ee457 Final Sp2014 sol.pdf

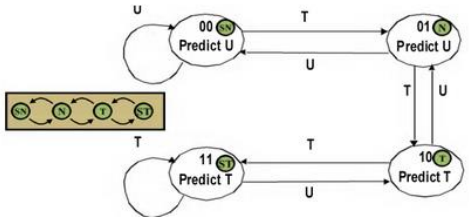
ee457 Final Fall2013.pdf (skip Q#5 on CLA)
ee457 Final Fall2013 sol.pdf

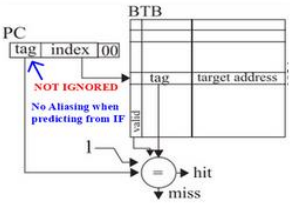
1. About 40% of the final exam will be dedicated to the 6 advanced topics and chapter 9 in class notes.
Please go through questions on special topics in the recent final exams and check your answers with the solutions. The three short homework assignments, where we gathered selected questions from the recent exams, form a good preparation on **Tomasulo, Branch Prediction, and Cache Coherency and Mutual exclusion.**

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.

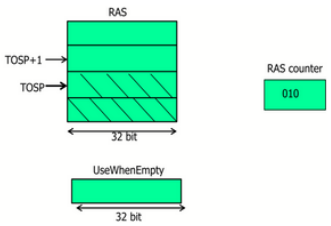






RAS (Return Address Stack) is a hardware stack, quite shallow, yet...
LIFO but ***circular***! Do you continue to push into RAS (as a result of jal), even if it is full? **Y**

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 to do so.
The unrepaired/unrestored RAS may provide wrong return addresses for a few occasions after the branch misprediction. This is considered acceptable since a return address 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).
The **actual return address is fetched by the JR $31** from $31 and an instruction prior to that [lw $31, 0($29)] must have loaded $31 with the return address saved on the system stack.

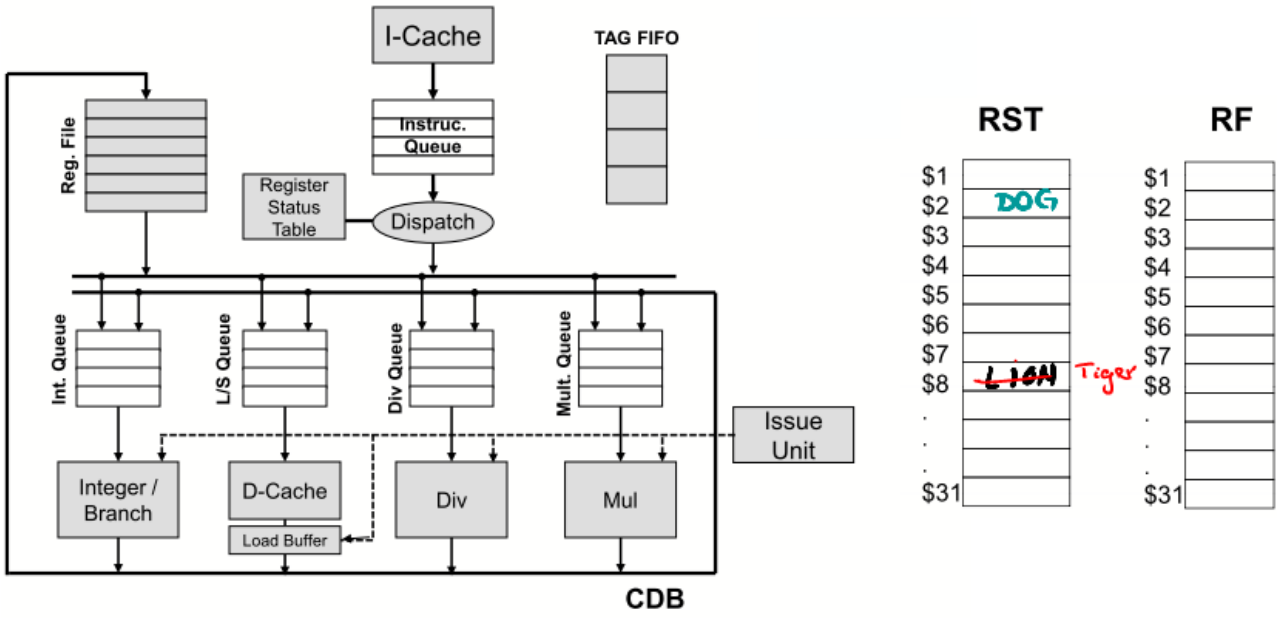
1.3. Out-of-Order Execution

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 two-clock FIFO is included.

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

1.3.2. IoI-OoE-OoC [Out-of-Order Execution with out-of-order completion (Tomasulo Part 1 algorithm with a TAG FIFO and RST (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 (are made to disappear) 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 junior-most 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 in this OoC design.

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? **Yes**!

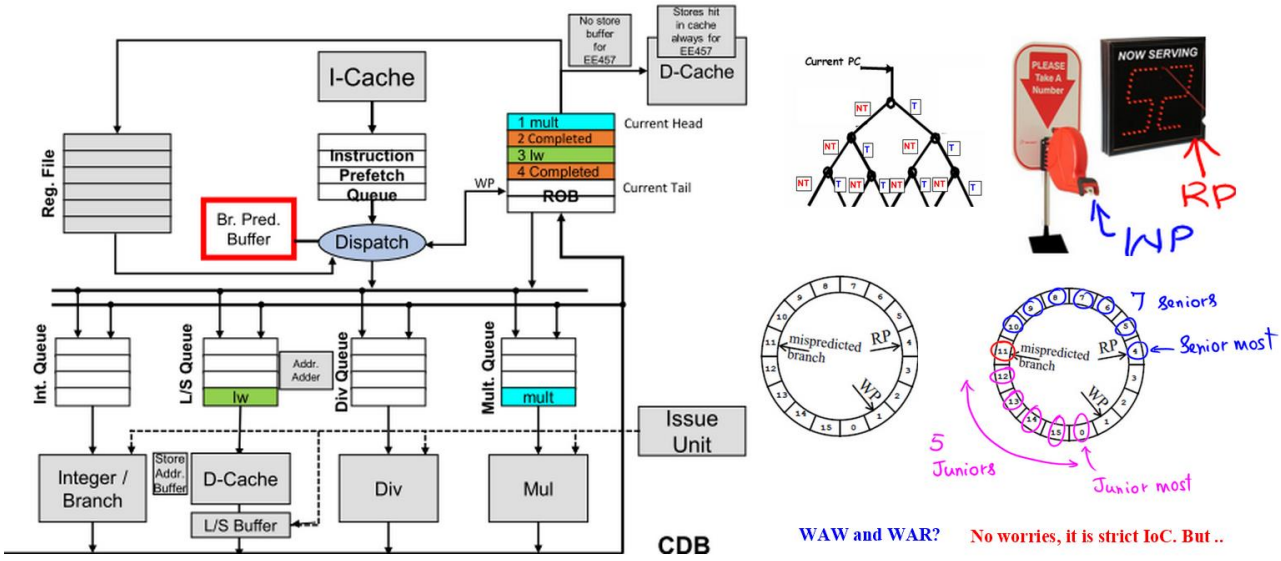
Instructions in loops and how instructions from different iterations of the loop can possibly co-exist in the backend. (Loop unrolling occurs dynamically)

Issue Queues (also called reservation stations): is it necessary or desirable (just desirable but not necessary) 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 designer desire that every execution unit is a fixed-latency execution unit? **Yes**! Then how about the lw instructions, which may incur cache miss? That is why we have placed **Load Buffer** after the cache!

Memory disambiguation rules

1.3.3. 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.

No RST here, because of difficulty in restoring (repairing) RST in the case of branch misprediction.

No Tag FIFO. In place of a Tag like "LION" in Part I of Tomasulo, we use the ID of the ROB slot pointed to by WP during Dispatch. This is called a "ROB Tag".

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

A virtual queue is formed 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!

A 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 _____

(the tail of the queue/the head of the queue / **location allocated to it previously at the time of dispatch**).

To do so, we need a random-access port on 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 associative search for say $2 in a 32-location circular ROB, we used _____ (1/**2**/3/4) 32-input fixed priority resolvers. **Two** for $Rs and **Two** for $Rt for a 2-source instruction such as add $Rd, $Rs, $Rt.

Speculative execution: Based on branch prediction, instructions after a 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**/elder) to the mispredicted branch instruction (or mispredicted JR $31) are called wrong-path instructions. "Who is younger to the

branch instruction" is inferred by computing the distance of all instructions with respect to the senior-most instruction (pointed to by the ___ (WP/**RP**)). There ____ (is a / **is no**) need for comparing the ROB Tag of an instruction in the backend with the ROB Tag of the mispredicted branch. Then, what is the purpose of announcing on CDB the ROB Tag of the mispredicted branch, if other instructions in backend do not want to compare their ROB Tags with this? The mispredicted branch is already announcing its distance from RP!
Answer: Well, the ROB needs to adjust its WP to the ROB Tag of the mispredicted branch in order to flush the wrong-path instructions in the ROB.

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 our ROB (a 32-location FIFO), 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 [WP == 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 mod_64 subtraction (note: mod_64 and not mod_32) subtraction to calculate the depth: Depth = [WP - RP] mod_64

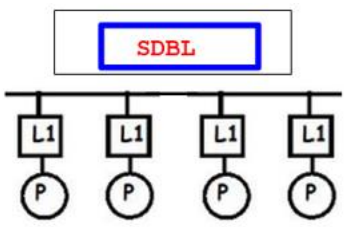
In this case, WP-RP = 000000 represents _____ (**EMPTY**/FULL) and WP-RP = 100000 represents _____ (EMPTY/**FULL**). In our EE560, it had so happened that the ROB designer used 6-bit pointers internal to ROB, all other designers external to ROB have used 5-bot pointers. And there was an interesting way to reconcile these two designs!

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

RAS _____ (is /**is not**) repaired, when you flush a bunch of wrong-path instructions, which may include some jal and some jR instructions, which may have caused some pushes to RAS and some pops from RAS respectively.
RAS, being circular, soon repairs itself! JR $31 carries with itself the predicted return address and compares with the actual $31's contents. If they mismatch, it announces misprediction on CDB and conveys the actual content to the Dispatch unit. Wrong-path instructions in the Backend, in the ROB, and in the IFQ (Instruction Prefetch Queue) are flushed and instructions at the right return address start fetching.

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

1.4.1 Mutual Exclusion, **RWM** Race:



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 **RWM** an atomic operation. So, why "this OLD method of locking the bus" is not good for creating mutual exclusion among threads of the system with multiple cores each core running multiple threads?
Answer: It causes undesirable traffic on the Bus due to constant polling (called busy polling or spinning) by other threads. In our EE560 4C*4T/C (4 cores x 4 Treads per each core), we have a total of 16 threads. If the SCU of one core (on behalf of one thread in that core) gets to do RMW and locks SDBL, then the rest of the 15 threads (3+4+4+4=15)keep polling.

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): Crossbar, hypercube, Omega, Butterfly, Mesh, Torus...

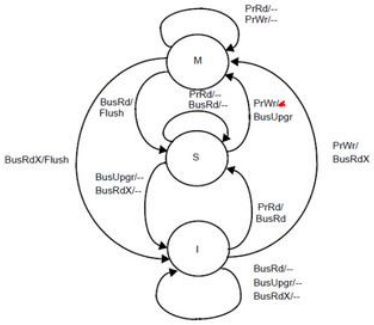
Notice that, allowing shared variables such as SDB_Lock (Student Data Base Lock) to be cached into private L1 caches reduces traffic on the bus (or memory interconnection network) to L2 cache, which saves clocks and reduces congestion on the bus (or memory interconnection network).

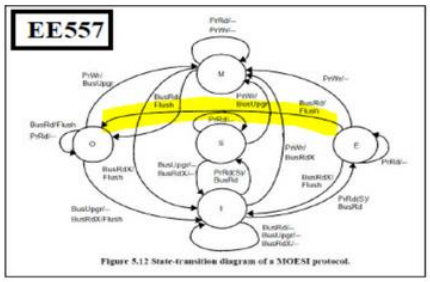
Also note that cache coherency **alone cannot** provide the needed mutual exclusion between competing threads which are competing simultaneously to set a lock (such as the SDB_Lock semaphore above). Cache coherency along with **LL** and **SC** instructions provides the needed **Mutual Exclusion** in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

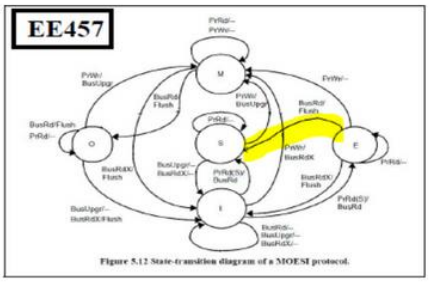
1.4.2. FMM (transferring FMM responsibility)

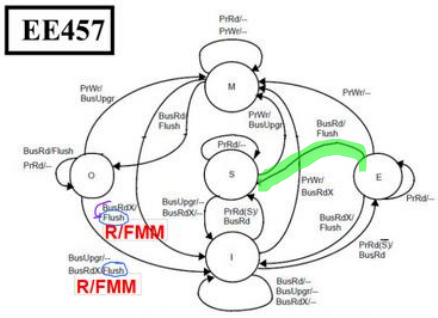
In write-back cache, with cache coherency protocol in place, flushing of dirty blocks to the MM (FMM = Flush to the MM) is postponed as far in time as possible. M-to-I state transition for a block in one L1 cache to allow I-to-M state transition for the same block in another L1 does not cause FMM as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

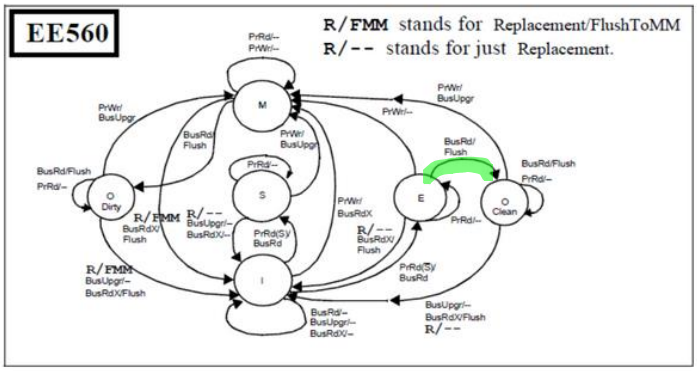
1.4.3 There will be a question on Cache coherency in a multi-processor system. Please practice reading MSI and MOESI (and MOOESI) protocol state diagrams and be in a position to complete a few missing state transition arrows and/or a few missing state transition conditions. If you are given the same 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)

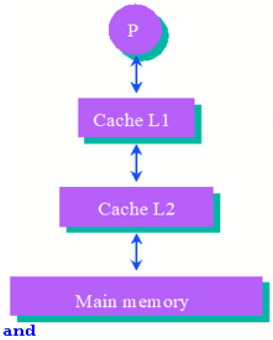
~~Dynamic Power makes CMP a preferred choice over uniprocessor with ramped-up frequency.~~ This subtopic is cancelled for Fall 2020 onwards.

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*20 + 0.01*300
= 1 + 1 + 3 = 5

https://en.wikipedia.org/wiki/Skylake_(microarchitecture)

Example:Intel Skylake Core i7 processor of year 2015
4 core, each core with 32KB I.Cache, 32KB D.Cache for L1, and 256KB unified cache for L2. L3 of 8 MB is common to all the cores.



1ST LEVEL CACHE IS SMALL AND FAST AND BE ON-CHIP
- E.G. 64KB, 2 CLOCK ACCESS TIME
- MPI: 5%
- SPLIT I/D

2ND LEVEL CACHE CAN BE VERY LARGE, SO THAT THE PENALTY OF L1-CACHE MISSES IS MOSTLY THE ACCESS TIME TO L2
- E.G. 12MB, 20 CLOCK ACCESS TIME
- MPI: 1%
- UNIFIED I/D
- ON-CHIP AS WELL.

OFF-CHIP ACCESSES TO DRAM ARE L2 MISSES
- E.G., 300 CLOCK ACCESS TIME

Which of the two below incur high overhead in context switching?
(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**) multi-threading.

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

ISA support for synchronization: instructions **LL** and the **SC** in MIPS.
EE457 Synchronization MutualExclusion LL SC.pdf

Fall 2014 Final

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 _____.

Fall 2014 Final Solution

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 <= on a single core single threaded system disabling interrupts provides needed mutual exclusion.
(b) between interacting threads running on different cores
(c) between interacting threads running on the same core
Even if (However / Even) if there is only one thread per core and you have established MOESI between cores, atomic test and set cannot (can / cannot) be guaranteed with the simple (ordinary) lw and sw instructions.
In MIPs ISA, LL stands for load linked (load locked) and SC stands for store conditional.

Explanation for (a) above: In an old uniprocessor (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 naïve code on the right. Now P1 becomes active, completes all the lines of the code on the right, obtains SDBL lock (Student Database Lock), starts modifying the database. Before it gets to complete all the modifications that it started, another 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 that it has obtained the lock and starts modifying the same database.
To prevent the above, interrupts are disabled before P0 (or P1) starts the code on the right.

case (b) above in the case of the old x86 processors such as 80486: The x86 processor ISA allows prefixing the assembly language instructions with a "lock" prefix to help locking the bus for a sequence of such "lock" prefixed instructions. This avoids RWM race between cores.

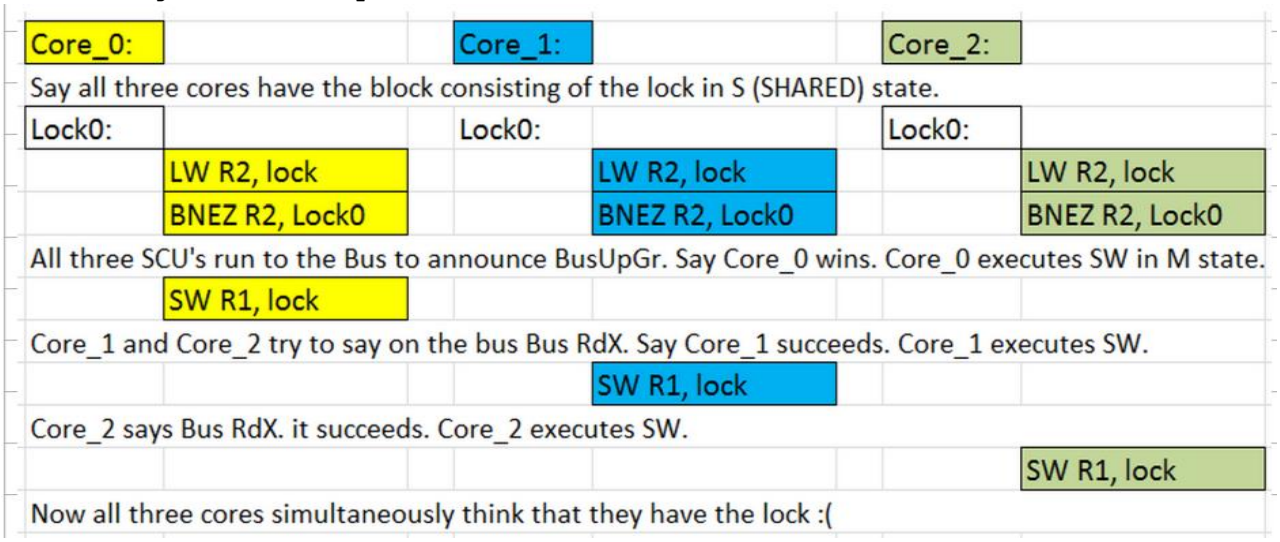
Naive code

```
Lock0: LW R2, lock
       BNEZ R2, Lock0
       SW R1, lock
           // R1 = 1

; Database modification

Unlock: SW R0, lock;
           // R0 = 0
```

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.



Please make sure that you understand the **14-scene sequence** on the 5 pages (27 to 31) of EE457 Synchronization MutualExclusion LL SC.pdf

1.7 Final lecture: In the final lecture we provided a sneak preview of three items (i) Tomasulo Part 3, (ii) PCIe, and (iii) GPGPU.
What is included for the final exam from the final lecture (mainly Tomasulo Part 3 preview) is stated in the three pages 27/34, 28/34, and 29/34 of the following pdf. Yes, you are responsible for these pages.
https://viterbi-web.usc.edu/www-classes/engr/ee-s/457/EE457_Classnotes/ee457_final_lec_Fall2022.pdf
You certainly need to know the following acronyms from slide 28.

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.

What does VLIW stand for? Very Long Instruction Word
Compared to a Superscalar CPU (where instruction scheduling is done in hardware), a VLIW CPU depends more (**more**/ less) on the compiler technology for instruction scheduling.
PCIe intro is moved to a different slide set, and you are not responsible for that.

2. Computer Arithmetic Chapter 4 Part 1 signed and unsigned numbers, Lab #3 (ALU lab)(less important, but may be used as a filler question)

~~Go through the following two questions and understand that an overflow and an underflow (in two parts of an operation) can cancel each other!~~
~~ee457_Final_Spring2012.pdf~~ ~~(Q#2 is interesting, Sp2018 students, please go through it)~~
~~ee457_Final_Spring2012_sol.pdf~~

~~Only Fast adders (EE457_ch4_p2_r1.pdf) are included if covered in Lecture/Discussion. Watch the 50-minute lecture: EE457_ch4_p2_r1.wmv~~
~~Look at the animation of CLA operation.~~
~~Fast multipliers are excluded.~~

~~**3**. Single Cycle CPU design and multicycle CPU design (less important, but may be used as a filler question):~~

~~Multicycle CPU design from both 1st ed. and 2nd edition.~~
~~HW#5a, HW#5b, Lab #4~~

4. Pipelined CPU design:

One major question or a few medium questions.

Understanding of all parts of Lab 7 [Part 1, Part 2, and Part 3 (all P3 subparts SP1, SP2, SP3, and SP4), including question 7 of ee457_lab7_P3_simple_pipeline.pdf] is IMPORTANT.
ee457_pipe_3elem_adder_Verilog.pdf
ee457_lab7_P3_simple_pipeline.pdf
ee457_lab7_P3_simple_pipeline_RTL_coding.pdf
I assume that EVERY student has done all assigned subparts of the lab 7.

Pipelined CPU design from both late branch (1st ed. Block Diagram) and early branch (3rd ed. and your lab 6 Block Diagram) are definitely important.
Lab #6 (parts 4, & 5). I know you have not done lab 6 Part 5, but you should be able to handle that in the form of a question if needed. The point is basically noting that replication of comparators can be avoided.
You should be able to handle any Forwarding, Stalling, and Flushing needs of any design. Browse through Lab 6 Part 4 Q#3.1 slides on removing redundant forwarding mux pair (FMP). Watch the video .avi if needed.

Time-Space diagrams: You should be able to draw (fill-in) or analyze.

load-word delay-slot and branch delay-slot <= Load-word delay slot implementation is very easy (reduce or eliminate stalling by HDU).
On the other-hand the Branch-delay slot implementation is complex and clumsy, particularly in the presence of instruction cache miss. Since Branch-delay slots are no more used and are not being implemented, we have recently stopped giving implementation-related major question involving branch delay slots *together* with ICMT (Instruction Cache Miss True) and DCMT (Data Cache Miss True) in the Midterm or the Final exams. Hence, I ask that you skip Q#4 of ee457_Final_Fall2014.pdf .

5. Chapter #7 Cache (HW#6) and Virtual memory (HW#7):

One major question or two medium-size questions.
Virtual Memory: Please look at Q#7 of Fall 2021 Midterm exam.pdf sol.pdf ee457 MT Sp2012 VM Ques sol.pdf , ee457 MT Sp2013 VM Ques sol.pdf ,
and then the Spring 2015 Final question 6.on virtual memory.
Please look at the Q#4.1 Q#4.2 and Q#4.6 of Spring 2017 Midterm and sol.pdf on Cache.
Questions could be "reverse-engineering" type questions like in the Midterm of Fall 2015 Q#3.

VIPT (Virtually Indexed Physically Tagged)
vs.
PIPT (Physically Indexed Physically Tagged)
The 9-stage pipeline became an 8-stage pipeline because the I_Cache_Tag_Check stage is hidden behind the instruction decoding stage.
Similarly, can you hide the D_Cache_Tag_Check stage? No! Memory write operation for SW is not like a stand-alone register writing operation, hence it cannot be aborted at the end of a clock if the D_Cache_Tag check fails. So, you cannot hide the D_Cache_Tag_Check stage.
Further if we use VIPT for both I_Tag Access, and D_Tag_Access, we can reduce the 8-stage pipeline to a 6-stage pipeline! But VIPT requires either very large virtual pages or very small L1 caches ☹

Degree of set associativity need not be a power of 2 but the number of sets shall be a power of 2. Since TLB is a cache of PT, we can think of all three cache mappings (Fully Associative, Set-Associative, and Direct) for TLB also. However (perhaps) Direct mapping for TLB is never used because of performance degradation due to conflicts.
Block size in TLB: Unlike in cache, the block size in TLB is always a one entry. Entries in TLB are always singular.

6. Lab related questions including Verilog coding. Please review the following.

6.1 Labs

6.1.1 Slides providing hints for Lab 7 P3 Subpart3: Hints.pdf and Hints.avi
6.1.2 FIFO lab (Part 1 only .pdf) Fifo is less important for the final as I have tested substantially in the midterm exam.
~~6.1.3 ROB lab (part 2 only .pdf)~~

6.2

Consider the following statement in a clocked always procedural block:
C <= A + B;
Suppose you are asked (in an interview) the following question:
Is it necessary that Verilog statements producing the above A and B appear in your Verilog code before the above statement, so that you follow the simple rule that says, "produce before you consume"?
Mr. Bruin: Yes, of course. I did Lab 7 SP3 :)
Mr. Trojan: A and/or B shall be produced before consuming only if they are intermediate variables produced using blocking assignments in the same clocked block.
For example, if A and B are outputs of some combinational operations such as a subtractor or an adder, then those statements should be written before the above statement.
In the following cases "producing before consuming" rule does not apply:
(i) If A and B are themselves register outputs (produced using non-blocking assignments in the same or in a different clocked always block)
(ii) If A and B are combinational outputs produced in a separate combination always block or by using concurrent assign statements (outside the clocked always block where they are consumed).

Note: We all know that the interaction between concurrent items (such as *always blocks* and *concurrent assign statements)* is governed by event-driven simulation and not by ordering of the concurrent items in our code.

6.3 If you had not previously looked at the following Verilog question, please go through it. It emphasizes the fact that only upstream combinational logic (such as NSL) [and not the downstream combinational logic (such as the OFL)] can be coded in the clocked always block along with the register.
Q#1.1 of EE354L Fall 2017 Midterm exam.pdf solution.pdf

7. Basic Logic Design:
You may be tested on basic logic design from EE354L. We do not expect anyone to have difficulty with the preparatory material from EE354L listed under the EE457 Study Plan for first 3 weeks.pdf
Please go through Q#3 of Fall 2022 final.