MUIC: Computer System and Architecture

Date: Thursday, March 31, 2022
Instructor: Rachata Ausavarungnirun

Name: Monthon Kraitheerawit

Problem 1 (35 Points):	
Problem 2 (15 Points):	
Problem 3 (30 Points):	
Problem 4 (20+5 Points):	
Total (100+5 Points):	

Instructions:

1. This exam lasts 24 hours. This is an open-book, open-everything exam. Every single question can be answered through materials we have covered.
2. Clearly indicate your final answer for each problem.
3. Please show your work when needed.
4. Please write your initials at the top of every page.
5. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space required.
6. **DO NOT CHEAT.** If I catch you cheating in any shape or form, you will be penalized based on our plagiarism policy ($N*$ 10% of your total grade, where N is the number of times you plagiarized previously).

Tips:

- **Read everything.** Read all the questions on all pages first and formulate a plan.
- **Be cognizant of time.** Do not spend too much time on one question.
- **Be concise.** You will be penalized for verbosity and unnecessarily long answers.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.
- **Write legibly.** Show your final answer.

1. Potpourri [35 points]

(a) **ISA vs. uArch [7 points]**

For each of the following, circle if the concept is a part of the ISA versus the microarchitecture (circle only one):

Number of cycles needed for the adder
Circle one: ISA Microarchitecture

Program counter
Circle one: ISA Microarchitecture

Pipeline registers
Circle one: ISA Microarchitecture

Number of registers
Circle one: ISA Microarchitecture

Cache replacement policy
Circle one: ISA Microarchitecture

ADD instruction
Circle one: ISA Microarchitecture

The branch predictor
Circle one: ISA Microarchitecture

(b) **Performance Measurement [10 points]**

You and your friends both buy a CPU with 3.5 GHz clock with the exact same motherboard, DRAM and SSD. However, when running the exact same assembly code (Program A), your friend's CPU is somehow 20% faster than your machine. What happen here? Explain your reasoning.

The reason might be Program A itself as it might need something like I/O operation to operate which needs buses for sending/receiving bits from the CPU, and another reason might be that my CPU have lesser core than my friend.

After noticing your friends computer runs Program A faster then your computer, you decide to buy the CPU with your same spec except this time there are 16 cores and a high-end GPU instead of 4 cores. Yet again, your friends' computer is still 20% faster. What can happen in this case? Please explain your reasoning and assumptions in detail.

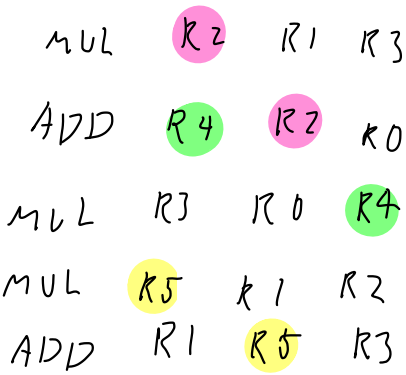
Program A might not work well against CPU with many cores as it might not optimise well with CPU with 16 cores.

(c) **Design Tradeoffs [6 points]**

What is the key benefit of branch prediction over stalling?

Without branch prediction, the processor would have to wait until the condition jump instruction has passed the execute stage before the next instruction can enter the fetch stage in the pipeline. It attempts to avoid wasting time by guessing whether the condition jump is most likely to be taken or not. Faster performance

Please provide a simple code example in MIPS ISA showing data dependency assuming a 5-stage pipeline? You must highlight the dependency.



(d) **DRAM Microarchitecture [4 points]**

List one difference between a DRAM bank and a DRAM channel.

DRAM Chunel is a connection path between the Memory Controller and Dram module

DRAM Bank is a set of independent array inside a DRAM chip

Each Bank of memory is an independent array that can be in different phases of a data access/refresh cycle.

What is the benefit of having more DRAM channels?

As each DRAM channels is physically independent from other channels and has its own memory controller that send series of DRAM commands to read/write data to DRAM cell

so everything will be faster

(e) **Simple Branch Prediction [8 points]**

What is the accuracy of a branch predictor is we always take the branch?

```
for (int i = 0; i < 100; i++) {      /* B1 */
                                    /* TAKEN PATH for B1 */
    if (i % 2 == 0) {               /* B2 */
        j[i] = k[i] - i;            /* TAKEN PATH for B2 */
    }
}
```

$B_1 = 100\%$

$B_2 = 50\%$

within 0 - 100 there are 50 even numbers

2. 5-stage Pipeline [15 points]

In this question, we will use the code below as a reference code we are running. For the instruction below, assume the instruction format is [operation name] [dest], [src1], [src2]. Assume that for every operation, fetch, decode, execute, memory and writeback stages take one cycle.

```
1 ADD R3, R1, R6
1 MUL R2, R0, R3
2 MUL R4, R2, R3
2 ADD R6, R2, R4
3 MUL R6, R4, R4
```

Assume **no pipelining**, and add instructions takes 5 cycles total while multiply takes 6 cycles total. How many cycles would it take to finish running the code above?

(2 · 5 cycles) + (3 · 6 cycles)

= 10 + 18

= 28 + total storage = 4 · 5 = 20

= 28 + 20

= 48

Now, let's assume a 5-stage pipeline with no data forwarding and the adder takes 1 cycle to execute, the multiplier takes two cycles to execute, all other stages take one cycle to execute, fill in the table below for how each instruction runs and what stage would each instruction be at each of these clock cycle. Put in **F** for fetch, **D** for decode, **E** for execute, **M** for memory and **W** for writeback. Leave the block blank if the instruction is stalling the pipeline.

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
MUL R3, R1, R6	F	D	E	E	M	W														
ADD R2, R0, R3		F	D				E	M	W											
MUL R4, R2, R3			F				D			E	E	M	W							
ADD R6, R1, R0							F			D	E		M	W						
MUL R6, R4, R4										F	D			E	E	M	W			

Finally, let's assume a 5-stage pipeline `with` data forwarding, fill in the table below for how each instruction runs and what stage would each instruction be at each of these clock cycle. Put in **F** for fetch, **D** for decode, **E** for execute, **M** for memory and **W** for writeback.

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
MUL R3, R1, R6	F	D	E	M	W															
ADD R2, R0, R3		F	D		E	M	W													
MUL R4, R2, R3			F		D	E	E	M	W											
ADD R6, R1, R0					F	D	E		M	W										
MUL R6, R4, R4						F	D	E	E	M	W									

3. Caching in Multicore [30 points]

Assume that we have a processor with 8 cores and 1-level 2kB, 2-way cache associative *shared* cache with a block size of 32 bytes, and assume that an integer is 8 bytes and an address is 32 bits. The array `j` and `k` are also 8 bytes and we are running the code below.

```
    for (int i = 0; i < 20000; i++) {
            j[i] = k[i] - i;
        }
}
```

What is the cache hit rate of the code above assuming that this is the only thread running on the system? Show your work and make sure you specify the tag, index bits.

Index = $\log_2(32)$ = 5 bits

blk = $\log_2(32)$ = 5 bits

tag bits -> remaining = 32 - 10 = 22 bits

Total accesses

Iterate from 0 to 20,000 each J, K = 40,000

miss = $\frac{20,000 + 20,000}{4}$ = 10,000

hit = total - miss = 40,000 - 10,000 = 30,000

hit rate = $\frac{30,000}{40,000}$

= $\frac{3}{4}$ = 75%

Your friend suggest that you can improve the performance of your program by using pthread. So, you ended up spawning four threads, each of which are responsible for a portion of the loop. You also ensure that you load balance the work in each threads (i.e., each thread is responsible for exactly 5000 loop iterations). While this is likely make the program much faster, you observe that the cache hit rate drops.

What can contribute to the drop in the cache hit rage? (Hint: Our cache is a shared cache)

Generally cache hit rate generally drop from the type of access, size of cache and frequency of consistency checks and in this qvestion 4 threads are tryin to access the cache which might be the reason in the drop of hit rate

After running this multithreaded version of the code multiple times, you observe a variation of the cache miss rate. Is this possible or are you dreaming? If this is possible, please explain why and provide the minimum cache hit rate that your new code can actually see. If you are dreaming, please provide a reason to convince yourself so that you wake up from this nightmare.

possible because multithreaded make each thread runs parallel to each other and create an additional stress on memory hierarchy caused by the interference among threads

4. DRAM [20+5 points]

In class, we learn of a few things in DRAM. One of which is the DRAM banks and the DRAM row.

Why do we need a row buffer and what is the effect of having the row buffer to the amount of time it takes to access DRAM?

Charges stored in DRAM is too small, and need to be amplified into a row buffer in order to read the data.

Let's go back to our computer in Question 3. Assume that we have a 1-level 4kB, 4-way cache associative cache with a block size of 32 bytes. Let's also assume a row buffer is 4kB.

Let's assume I have a series of 50 data accesses that results in cache misses from each of the two CPU cores (Core A and Core B), making it 100 data accesses, all of which are cache misses and all go to the same DRAM channel and DRAM bank. Let's also assume that a row buffer hit (i.e., access to the address that is already on the row buffer) takes 15 ns while a row buffer miss (an access to an address not on the row buffer) takes 50 ns. Let's also assume we do not have to worry about any other DRAM timing parameters such as DRAM refreshes.

What is the minimum and the maximum time it takes to process these 100 data accesses? Why?

Worst case = 50 x 100 = 5000 ns

Best case = 15 x 100 = 1500 ns

[Extra credit: 5 points] If the size of my row buffer change from 4kB to 2kB.

What is the minimum and the maximum time it takes to process these 100 data accesses? Why?