yz709-CD-sup3

Question 1

What are the differences between interrupts, environment calls (software interrupts) and exceptions?

- Exceptions would disturb the execution flow of the program and may abort the program. Interrupts would only pause the program's execution, doing context switching to change between fetching instructions for the current thread and fetching instructions for the interrupt handler.
- Causes of interrupts, environment calls and exceptions:

    - The (hardware) interrupts are caused by external factors rather than instruction execution, e.g., the timer interrupts for scheduling, DMA interrupts used to indicate the end of a transfer, I/O device interrupts to trigger execution of a device driver.
    - The software interrupts are caused when the operating system needs to change the modes from a less privileged mode to a more privileged one or vice versa.
    - The exception is caused by the instruction stream, e.g., illegal instruction, division by zero and page faults.
- Synchronous or asynchronous & impacts on the execution of the program:
    - The (hardware) interrupts are asynchronous, and the program would resume after the interrupts are handled.
    - The software interrupts are synchronous but may not terminate the program if the trap to the operating system includes a valid operation. (e.g., require the OS to check passwords and resume if match).
    - The exceptions are synchronous and often terminate the program.

Comments:

Question 2(y2008p6q2)

(a) What do *RISC* and *CISC* stand for and what are the differences in practice? [6 marks]

- RISC (Reduced instruction set computer):
    - Few instructions in the instruction set, fewer address modes
    - Each instruction takes about one clock cycle, so suitable for pipelining
    - A fixed instruction format and focused on software to optimise performance
- CISC (Complex instruction set computer):
    - More instructions in the instruction set include complex instruction that is rarely used, more address modes

- Each instruction may take more clock cycles, so fewer pipelined
- A variable instruction format and focuses on hardware to optimise performance

(c) Early computers used just an accumulator rather than a register file or stack. How does the code density compare between accumulator and stack machines? [4 marks]

- Code density is the number of instructions the processor takes to perform an action and the space each instruction takes up. The less space each instruction takes and more work per instruction done means a higher code density.
- For an accumulator, a complex instruction needs to decompose into simple instructions to be performed on the accumulator. Hence there are a lot of machine codes generated.
- For a stack machine, each call stack contains a lot of parameters and local variables, so fewer memory accesses are required implies fewer machine codes, so the code density is much higher than the accumulator.

💡 Comments:

Question 3(y2006p6q2)

(b) In 1946, Burks, Goldstein and von Neumann made the observation "Ideally one would desire an indefinitely large memory capacity such that any particular word would be immediately available. We are forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

(i) Why is this memory design issue even more pressing today? [5 marks]

- We have smaller and smaller physical sizes of computer systems, including mobile devices and tablets. Placing the memory cells(transistors and capacitors) too close would cause charge leakages, thus limiting our physical design.

- The processor speed increases throughout the years, introducing a big difference with the memory access time. Hence we need to have a hierarchy of memories, including fast level 1 cache, level 2 cache, main memory and secondary storage to ensure the processor can execute at its highest potential.
- Memory that can be accessed quickly should be placed near the processor, which is another physical restriction. To compromise between physical design and cache miss rate, we are forced to have a memory hierarchy.

(*ii*) What architectural techniques are used today to approximate a large memory which is nearly instantaneously accessible? [5 marks]

- Virtual memory. Each process would have a virtual memory space that might be bigger than the physical memory. The virtual memory address would be translated into a physical memory address via the page table.
- Virtual memory would usually have a translation look-aside buffer that stores recently-used page table entries, acts as a cache of PTEs. It could enable instantaneous access to the page table entry if we have a TLB hit.
- The page fault system and page replacement algorithms would mitigate rarely used pages to secondary storage, thus giving an illusion of a bigger memory and tolerating long-running applications with memory leaks.

Comments:

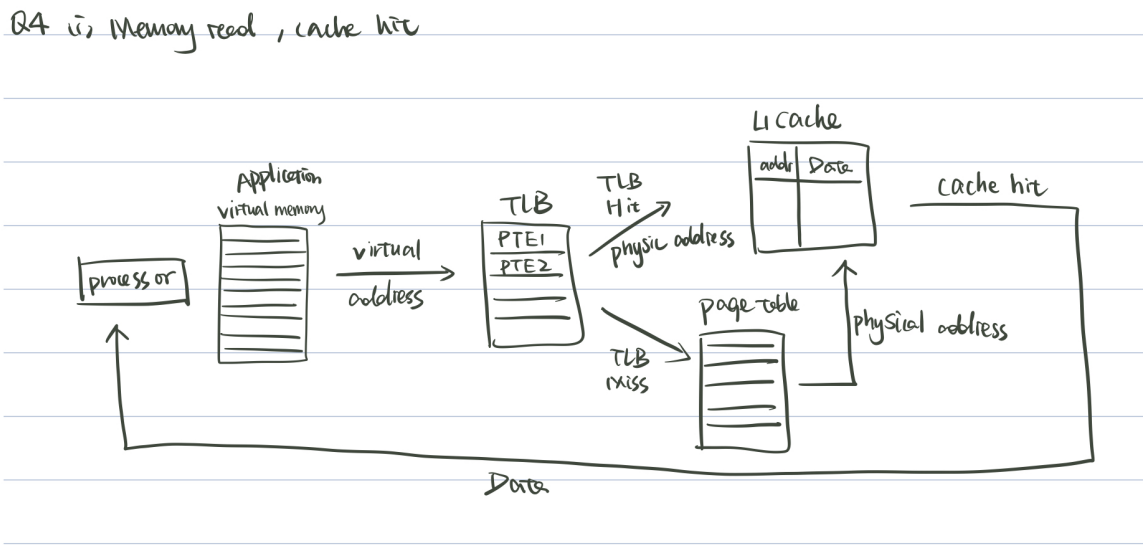
Question 4

Sketch examples of a i) memory read and a ii) memory write, former cache hit, latter cache miss, considering a processor with a single level of cache, virtual memory (TLB, page table), and RAM. Explain both examples.

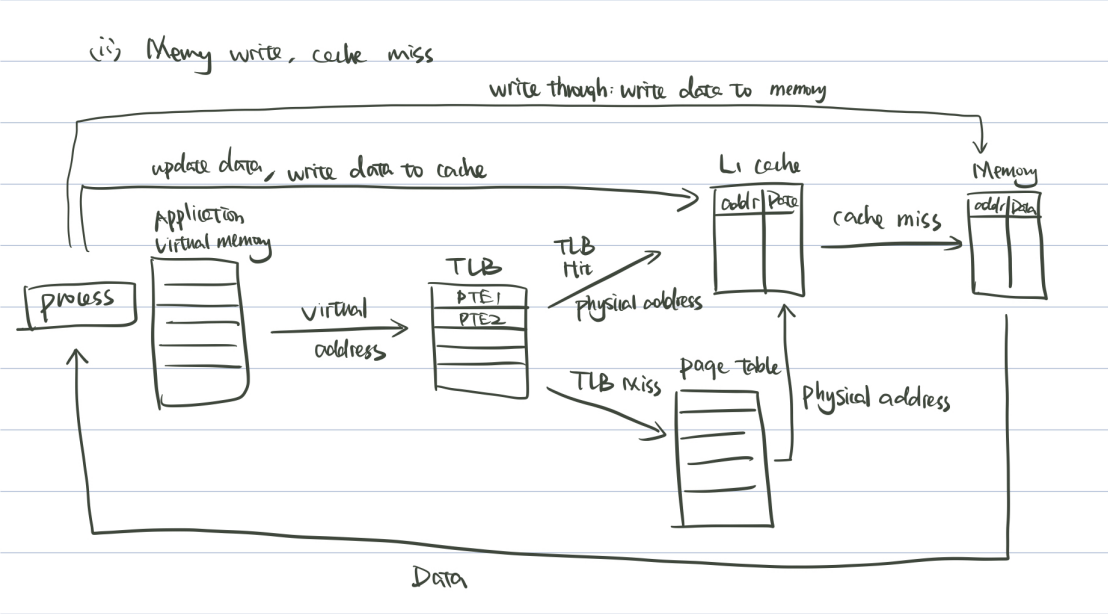
- (i):
  - The processor sends the virtual memory address stored in the application's virtual memory space to the TLB.
  - If we have a TLB hit where the corresponding page table entry is already stored in TLB, then send the physical address to the cache. If we get a TLB

miss, we need to do an address translation using a page table to get the physical address.

- Since we have a cache hit, the data is already stored with the address in the cache, so we can send back the data to the processor directly without accessing the main memory.



- (ii):
  - The processor sends the virtual memory address stored in the application's virtual memory space to the TLB.
  - If we have a TLB hit where the corresponding page table entry is already stored in TLB, send the physical address to the cache. If we get a TLB miss, we need to translate an address using a page table to get the physical address.
  - Since we have a cache miss, so need to fetch the data from the main memory then send it back to the processor. Then update the data and write data back to the cache first with the corresponding address.
  - If we implement a write-through policy, we need to update the data stored in the main memory and introduce some overheads.



Comments:

Question 5

How does a Translation Look-aside Buffer TLB improve the performance of virtual memory accesses?

- If we don't have TLB, an address translation from virtual memory address to physical memory address includes two memory access, one memory access to the page table entries and another to the physical memory address.
- TLB is a cache of page table entries; it stores the most-recently-accessed page table entries; thus, when we later want to access these entries, a direct fetch from the TLB implies we don't need memory access to the page table. Therefore only one memory access to the physical address is required. Hence, less latency compared to cache access and main memory access and better performance.

Comments:

Question 6

How is virtual memory used to isolate applications?

- Different applications have their own virtual memory address space and pointers to page tables.
- The virtual memory address is mapped using a page table to a physical memory address, with permission bits determining whether the page can be accessed.
- Hence applications cannot access other applications' virtual memory space if the space is not shared.

💡 Comments:

Question 7

What are the main components of a modern SoC?

- Processors (CPU, GPU, DSP): executes program instructions and do arithmetic or logic computations and I/O operations.
- Memory (ROM, RAM): a system or space used to store intermediate computation results from processors.
- Interconnects (a bus or NoC): connects the processor with memory and other peripheral devices to transfer data.
- External interfaces (USB, ethernet): provides a connection between the computer and the outside world.
- Timing (phase-locked loops): used to synchronise inputs and outputs.

💡 Comments:

Question 8

Suppose that there are four different processor types: A does 15% of the work for an application, B does 20%, C does 5%, and D does 60%. If you speed B up by 2× and D up by 3×, what is the overall application speed up:

- a. When do all four run in parallel?
  - Assume the processors are all the same and have the same performance. Suppose the work needs x hours using one processor; if the four processors execute serially, the total execution time is x hours.
  - Speed B up by 2x implies B only needs to spend 10%x to finish its part. Similarly, speed D up by 3x indicates D only needs to spend 20%x to complete its part.
  - Hence if the four execute in parallel, then total execution time = max(A, B, C, D) = max(15%x, 10%x, 5%x, 20%x) = 20%x.
  - Overall speedup is 80%
- b. When do all four run sequentially?
  - Since after speed up B and D, B needs to spend 10%x and D needs to spend 20%x, the total execution time = A + B + C + D = (15% + 10% + 5% + 20%)x = 50%x
  - Overall speedup is 50%
- c. What's the limit on the speedup you can get from only improving B and D?
  - When B and D have an execution time that tends towards 0%x, we have total execution time $\rightarrow$ A + B + C + D = 20%x. Hence the overall speedup tend towards 80%

Comments:

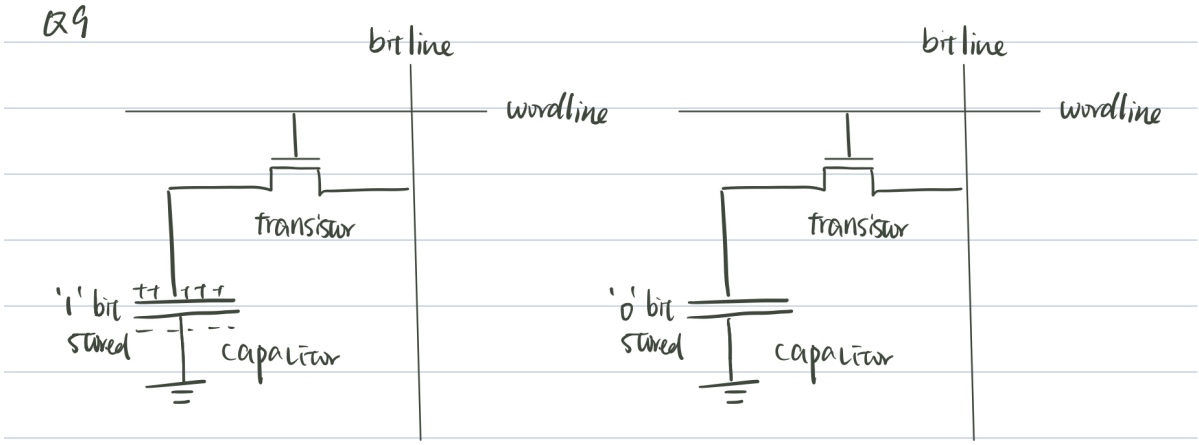
Question 9

Draw a 1T1C DRAM cell. Why does it need a refresh, and what is it?

- Refreshment:
  - Because capacitor stores data as electronic charge, but the capacitor leaks charge over time, so without being refreshed, the data would eventually be lost
  - We have memory refresh cycles, and each one would refresh a succeeding area of memory cells, thus repeatedly refreshing all the cells in a

consecutive manner

- We apply external circuitry periodically to read each cell and rewrite it, restoring the charge on the capacitor to its original level



💡 Comments:

Question 10

A memory controller converts processor requests into commands for the DRAM.

- a. Why might it prefer to order requests so that they access different banks on consecutive memory operations rather than the same bank?
  - Because different banks operate individually, enabling parallel execution of consecutive memory operations and vastly improving the overall performance.
- b. When is it beneficial to continuously access the same bank?
  - Suppose the memory addresses have good spatial locality, i.e., when data are closely stored on the same bank. Then accessing the same bank could reduce latency when reading the same wordline repeatedly.

💡 Comments:

Question 11

Why is there a need to perform both a row access and column read when reading data out of DRAM? What does each operation do?

- Because a DRAM array is a matrix with rows and columns, the bit we are interested in needs to be located using row value(wordline) and column value(bitline).
- A row access asserts a wordline and loads all the bits into a sense amplifier.
- A column access selects the correct bitline so that only one bit is written to the data buffer and outputs.

> 💡 Comments:

Question 12

Describe the four classes in Flynn's taxonomy of computing systems. Explain where a multicore processor with a short-vector instruction set fits within Flynn's taxonomy?

- Four classes:
  - SISD: single instruction single data stream, usually implemented in a simple processor
  - SIMD: single instruction multiple data stream, usually used for vector processing where a single instruction is applied on various data streams, thus is more energy efficient
  - MISD: multiple instruction single data stream, usually used for redundancy purposes, where the single data stream is processed for multiple instructions
  - MIMD: multiple instructions multiple data stream, a typical design for multicore, standard general-purpose CPUs
- A multicore processor with a short-vector instruction set:
  - For each core of the multicore processor, they implement SIMD using the short-vector instruction where a single instruction could be applied on multiple data streams

💡 Comments:

Question 13

Describe Amdahl's' and Gustafson's laws. Comment what is happening as we increase the number of cores in the case as observed by Amdahl.

- Amdahl's law assumes a fixed question size, where the speedup from parallelism as the number of cores increases would be:

$$Speedup(n) = \frac{1}{B + (1 - B)/n}$$
$$\text{where B is the sequential fraction, n is the number of cores}$$

  - Hence implies that only the very best parallel programs can make the best use of the cores, the sequential part limits the speedup; therefore, we have to concentrate on minimising the sequential part to improve parallel performance.

- Gustafson's law assumes a fixed execution time, where the workload can scale up with increased cores. The speedup when cores increases, for a sequential fraction B and number of cores n would be:

$$Speedup(n) = n + (1 - n)B$$

  - Implies that when given more computing power, programmers increase the problem size and the parallel part scales, the sequential part remains constant.

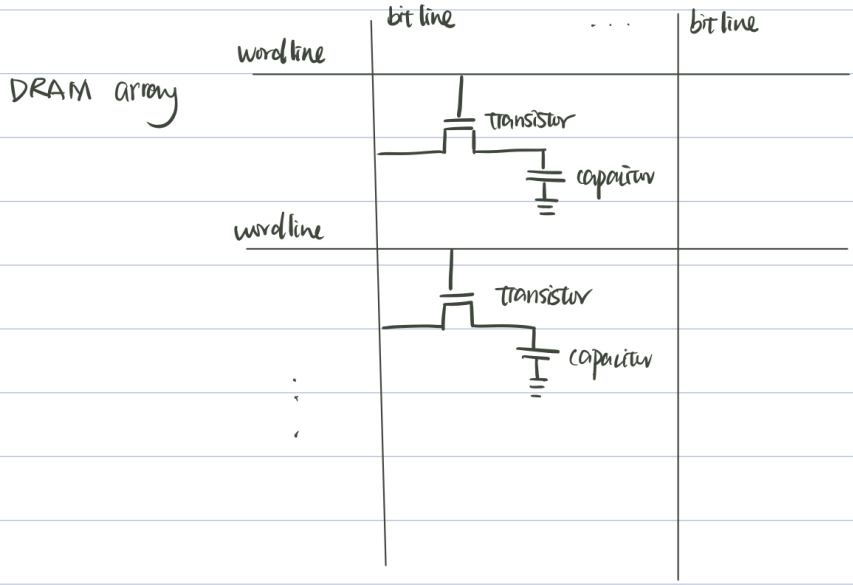
💡 Comments:

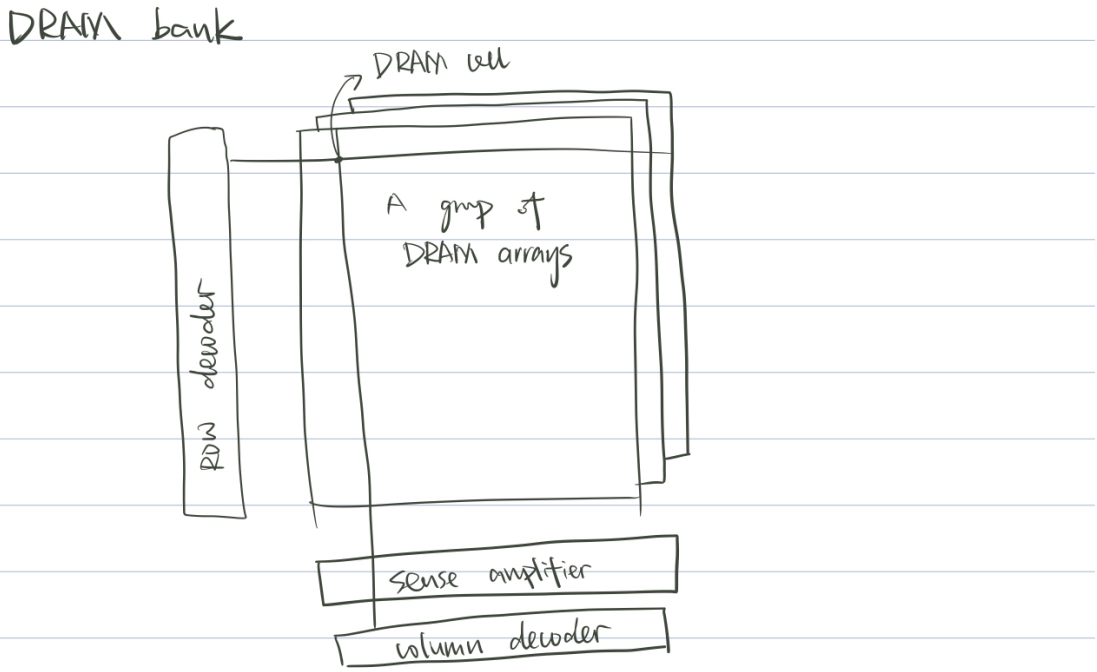
Question 14

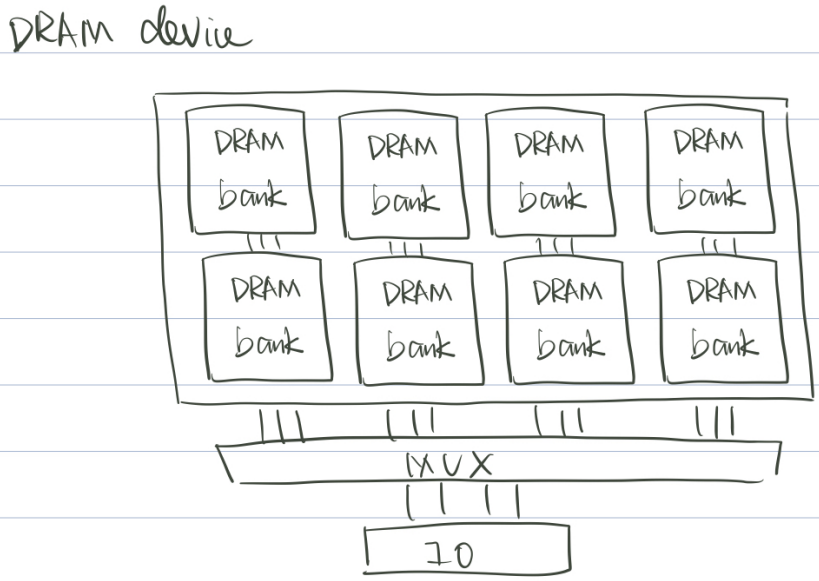
Show, with the aid of a diagram, how DRAM is organised, referring to devices, ranks, banks and arrays.

Q14

- DRAM rank consists of multiple DRAM devices, all operating together
- DRAM devie consists of multiple DRAM banks, each bank operates independently
- DRAM bank contains a group of DRAM arrays, each array outputs one bit
- DRAM array contains n wordlines and m bitlines, so it has n·m DRAM cells, which consists of a transistor and a capacitor.







Comments:

Question 15

Describe the difference between an open-page and closed-page row-buffer policy and the types of access patterns they benefit from.

- Open-page row buffer policy:

    - The sense amplifier doesn't reset, provides a reduced latency when reading from that row(wordline) later. At the same time, it requires explicit reset when accessing another row(wordline)
    - Hence it is excellent for sequential access to different columns(bitlines), such as sequential access to arrays with the spatial locality.
- Closed-page row buffer policy:
    - The sense amplifier does get reset and precharge; hence it is great for jump access, such as pointer chasing in the linked list with little spatial locality.

> 💡 Comments:

Question 16

Why is shared memory a useful concept for programmers? Describe its disadvantages.

- A shared memory aligns with the programmers' view, hence much easier to program.
- Disadvantages:
    - Lack of data coherence because data updates need to be propagated through cores, so we need multicore caching to ensure data coherency
    - Access time degradation when several cores try to access the exact memory location, access time degradation causes contention and high latency, so it cannot scale well.

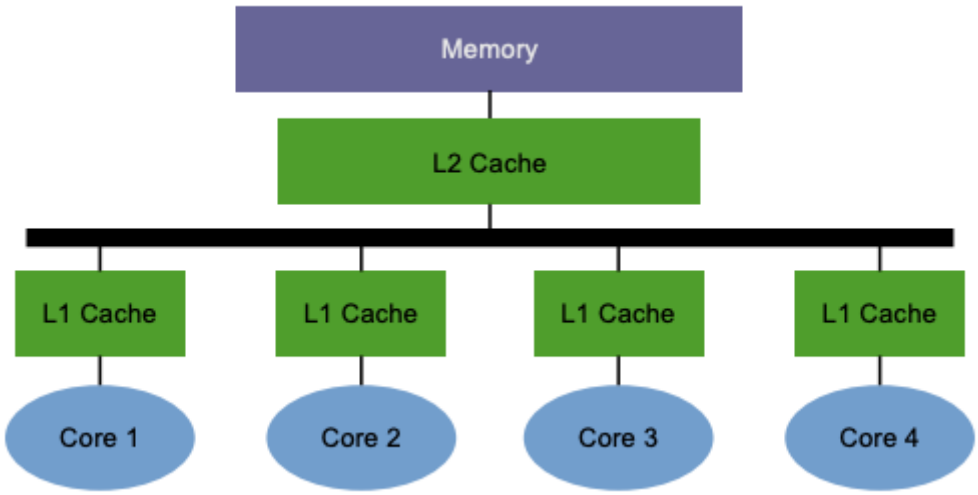
> 💡 Comments:

Question 17

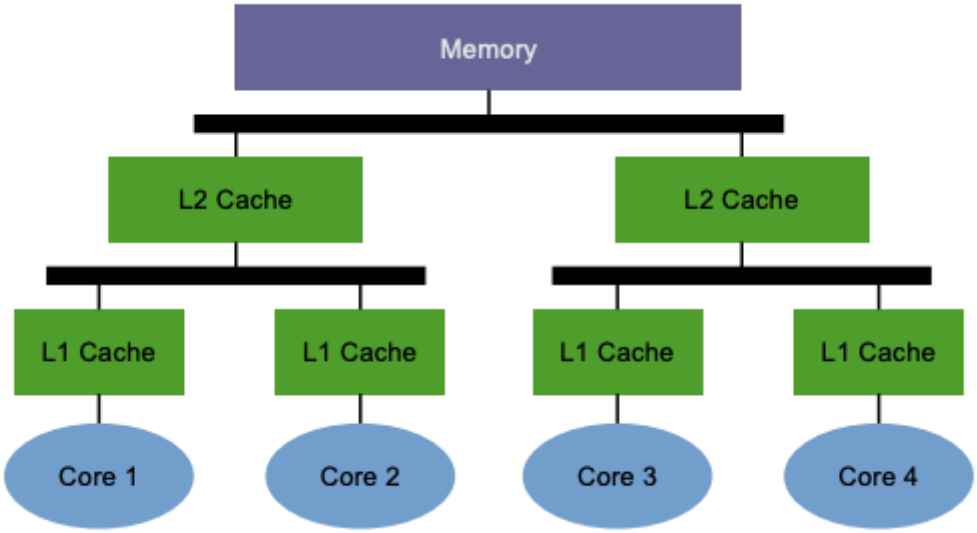
Considering the shared cache organisations on slides 12–14 (Part II of the course, lecture 12) of the lecture notes, what are the benefits and downsides to each approach?

- A single level of shared cache:
  - Benefits:
    - Simple structure
    - Dynamic cache allocation for cores so that cores with heavy workload can have more cache memory allocated
    - Reduce miss rate because data fetched by one core could later be used by another core via the shared L2 cache
    - Less duplicated information because there will only be one copy of the data in the L2 cache
    - Interprocessor communication is much easier via the shared memory structure
  - Downsides:
    - More concurrency issues because cores might want to access the same piece of data in the L2 cache
    - One core may occupy the majority of the cache space; hence the unfairness of the relocation may introduce starvation
- Multiple shared caches:
  - Benefits:
    - Fewer concurrency issues because fewer cores share the same L2 cache
  - Downsides:
    - Much hard to implement interprocessor communication between cores that do not have shared memory
    - Hard to decide how to select the two cores which would share the level 2 cache memory
- Multiple private cache levels:
  - Benefits:
    - Reduce miss rate because three levels of cache would store most of the data a core wants during execution
    - Fewer concurrency issues because most data a core wants would be stored in private level 1 and level 2 caches

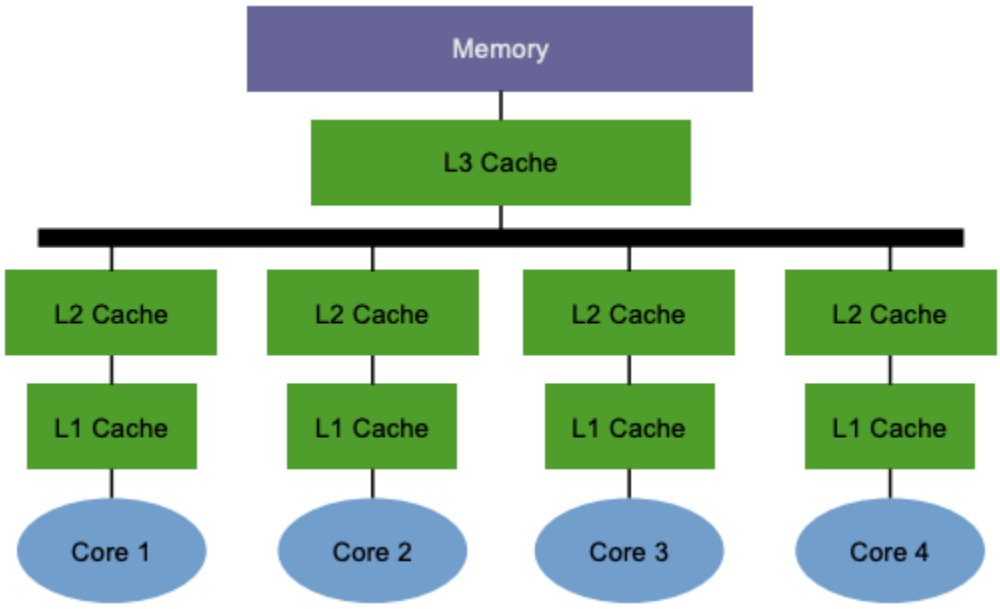
- Downsides:
  - Harder to implement interprocessor communication between cores because the shared memory is at level 3 cache



A single level of shared cache



Multiple shared caches



Multiple private cache levels

💡 Comments:

Question 18

Describe inclusive, exclusive and non-inclusive cache policies.

- An inclusive scheme implies data would be duplicated at all levels
  - Shorter miss latency for an inclusive cache because a cache miss at lower levels indicate the higher level cache need not be searched.
  - Suffer from a higher rate of filling new blocks because we need to copy the data in all cache levels.
  - The memory capacity of the cache is determined by the lower level cache because data needs to be stored in both, so if the lower level cache is full, we cannot store the data solely in the higher level cache, which leads to more memory space wastage.
- An exclusive scheme implies data would only be stored in one of the levels
  - Higher miss latency because we need to search all cache levels.

    - Manageable under the higher rate of filling new blocks because data only need to be stored in one of the cache levels.
    - Better usage of the memory capacity of the cache because higher-level capacity does not depend on lower-level cache capacity.
- The non-inclusive scheme is a mix of inclusive and exclusive schemes, it implies that data may or may not be in all levels, there are no guarantees.
    - Easier to implement but harder for programmers to take advantage of because there is no guarantee whether a particular data piece is stored in one level or all levels.

> 💡 Comments:

Question 19

Describe MSI cache coherence protocol.

- MSI cache coherence protocol is designed to ensure all cores read the most up-to-date values for each memory address.
- There are three states a cache can be in:
    - Modified state:
        - Single up-to-date copy in this cache, value in the main memory is stale
    - Shared state:
        - Copy in this cache matches with the main memory
    - Invalid state:
        - Not valid state and there are no copies in this cache
- These coherency states are maintained through communication between the caches and the main memory.

> 💡 Comments:

Question 20

You hold data at address X when you snoop a BusRd transaction from the bus. What actions do you take with the MSI cache coherence protocol if you start in:

- a. State M?
  - The cache needs to flush the data back to the main memory and transfer $M \rightarrow S$
  - Hence, the data the other cache read would be the most up-to-date value (updated by my cache) stored in the main memory.
- b. State S?
  - Do nothing because a read does not update the value at all.
- c. State I?
  - We could get the data from memory and keep a copy in the local cache, transfer $I \rightarrow S$

Comments: I assume other cache wants to read the data at address X

Question 21(y2018p5q3)

(b) The MOSI cache coherence protocol adds a new owned (O) state to the basic MSI protocol.When a cache holding a line in M state snoops a read request from another cache, it transitions to O state and forwards the data to the requestor. Subsequent snoops for read requests are also fulfilled by this owner cache. An owned line is dirty and only one cache can hold a line in O state at any time.

(i) Describe the difference between cache coherence and memory consistency. [2 marks]

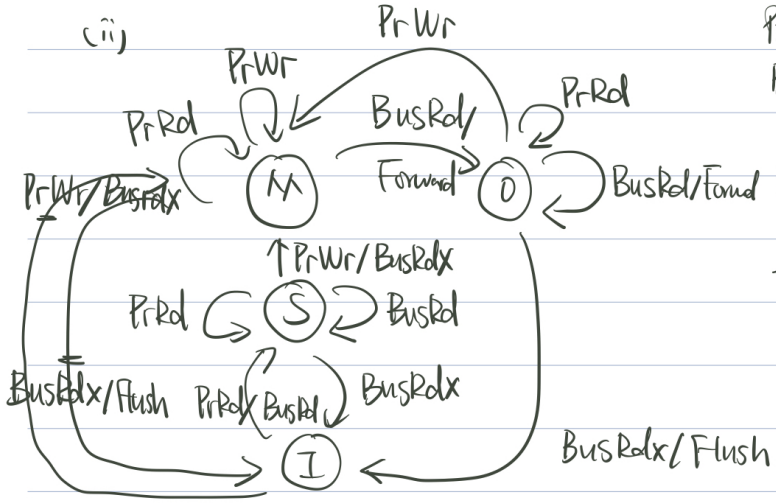
- Cache coherence refers to having the same value in individual memory address across all caches
- Memory consistency refers to the ordering of accesses to the main memory, including reading and writing operations

- A system that is memory consistent must be cache coherent, but not the reverse

(*ii*) Draw a state transition diagram for the MOSI protocol, using a new action *Forward* to indicate data being forwarded from one cache to another. [6 marks]

Q21

(ii)



PrRd : processor read request
PrWr : processor write request
BusRd : read request from other caches
Flush : flush data to the main memory
Forward : forward data to the read requestor
BusRdx : exclusive read request

(*iii*) Draw a table showing how the state of a line in one cache limits the states the same line can have in a different cache. [2 marks]

(iii)

	M	S	I	O
M	X	X	✓	X
S	X	✓	✓	✓
I	✓	✓	✓	✓
O	X	✓	✓	X

(*iv*) Give two benefits of adding this extra owned state to the basic MSI protocol. [2 marks]

- Less overhead for interprocessor communication because data can be directly propagated from one cache to another without going through the main memory
- Introduce redundancy - when one core is at state O, others can at state S, so if one core is down, the data copy will not be lost

💡 Comments:

Question 22

Summarise the main message from lesson 9 in 1-3 sentences?

- Various hardware supports for the operating systems include execution modes, exceptions, interrupts, and software interrupts privileged instructions and virtual memory.

💡 Comments:

Question 23

Summarise the main message from lesson 10 in 1-3 sentences?

- Instruction sets other than RISC include CISC with variable length complex instructions, Java bytecode which focuses on portability and CHERI-RISC-V, which focuses on ensuring safety and reducing attack surfaces.

💡 Comments:

Question 24

Summarise the main message from lesson 11 in 1-3 sentences?

- System on-chip can be classified into four general groups in Flynn's Taxonomy based on the nature of their instruction and data streams.

- Parallelism on chips is restricted by the fraction of sequential cores based on Amdahl's law, which assumes a fixed problem size.
- Details of the structure of DRAM storage where a DRAM consists of multiple independent ranks. Each rank contains various devices working together. Each device has various banks operating independently and consists of a group of arrays. Each array provides a single bit.

💡 Comments:

Question 25

Summarise the main message from lesson 12 in 1-3 sentences?

- In multicore processor organisations, we used shared memory or message passing for communication between multiple cores.
- Multicore caching needs to be implemented in shared memory because it has a high latency.

💡 Comments: