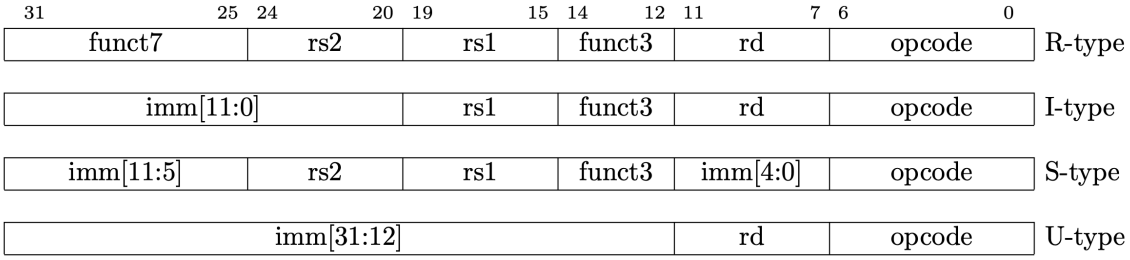
yz709-CD-sup2

Question 1(y2018p5q2)

The RISC-V base ISA instruction formats are reproduced below. Each immediate subfield is labelled with the bit position (imm[x]) in the immediate value being produced, rather than the bit position within the instruction's immediate field as is usually done.



(a) In assembler, what is an *immediate*? [2 marks]

- An immediate operand is a constant value in instruction; this value is immediately available, does not require any register or memory access.
- Immediate values are used in all instruction types (I-type, S-type and B-type etc.) except R-type.

(b) Give examples of *load*, *branch* and *arithmetic* instructions in assembler that use an immediate. Note that it is not critical for the RISC-V assembler syntax to be perfect, but for each instruction please explain your answer. [3 marks]

```
// load instruction
lw x1, 5(x3)
// 1. load a word of data from memory location (address = value in x3 + 5) where 5 is the immediate field
// 2. The immediate field is sign extended to 32 bits in the machine code
// 3. Put the data from (address = value in x3 + 5) to memory address (stored in register x4)

// branch instruction
beq x1, x2, -2
// 1. Branch back by 2 instructions if data stored at register x1 == data stored at register x2
// 2. The immediate field is -2

// arithmetic instruction
addi x1, x1, 1
// 1. Add 1 to register x1 using add immediate
// 2. The immediate field is 1 which is a constant part in the instruction
```

(e) Why can there be many more register-to-register instructions with no immediates than instructions with immediates? [3 marks]

- R-type instructions have no immediate fields, but it has a 7-bit opcode, a 7-bit `func7` field and a 3-bit `func3` field; hence with the 17 bits, there can be a large encoding space for R-type instructions.
- Other instructions with immediate fields have less than 17-bit encoding space, so fewer instructions are available.

💡 Comments:

Question 2

Discuss ISA-level simulation with an example.

- The ISA-level simulator is a program that interprets machine code to emulate a processor; it contains a mechanism to load code into memory, performs the decode, executes, branch, memory access and write-back steps for each instruction.
- Spike is an example of a RISC-V ISA-level simulator; it provides a golden reference for a program running on RISC-V.
- Pros:
  - Good for educational purposes and cost-effective, do not need to wait until the hardware is available.
  - Help to debug and reveal random errors due to the initial loading address in the address space. It could also detect illegal operation on operation systems during simulation.
  - A sufficient amount of simulations could provide hot spot analysis to detect the set of instructions that are most commonly run, and compiler writers could focus on optimising for these instructions.

💡 Comments:

Question 3

What is the difference between the data and control paths, and why do they flow together down the pipeline?

- Control signals are determined by the instruction and some other micro-architectural states (e.g., the result of ALU is not zero might lead to a branch).
- Data flows through the data path, with some data coming from the instruction (e.g., immediate field).
- All control signals on the control path need to be taken into account, but some of the data on the data path may be ignored (e.g., we may ignore the `rs2` field for an I-type instruction).
- Control signals in the control path determine how each part of the data path performs each instruction. Hence we need both the control signals from the control path and the data from the data path to achieve the correct operations.

💡 Comments:

Question 4(y2017p5q2)

Consider the following statements about processors. Why are they fallacies?

(a) A benchmark is a typical program that accurately predicts performance. [4 marks]

- A benchmark can reasonably predict the performance based on recent history or program types.
- Still, there is no way it can accurately predict performance since there are a variety of different behaviours for a program, and we cannot get it right all the time.

(b) Instructions per second is an accurate measure to compare performance among computers. [4 marks]

- Different instructions require various clock cycles; hence, the number of instructions processed by computers is not a good indicator of their speeds.

(c) Peak performance tracks observed performance. [4 marks]

- Peak performance always occurs under ideal circumstances, while observed performance, in reality, may encounter cases like cache misses, which would slow down the execution time.
- Hence peak performance is not a good indication of observed performance.

(d) You can design an ideal processor suitable for all applications. [4 marks]

- Different applications have different requirements, and some of the requirements are trade-offs, so there is no ideal processor that could satisfy all the requirements.
- For instance, one gaming application might focus on user interactivity and performance while another mobile application may prioritise low power consumption.

(e) Adding more pipeline stages always improves performance. [4 marks]

- More pipeline stages increase the number of instructions executed simultaneously, hence a faster clock frequency. However, there would be a higher chance of data hazard, control hazard and structural hazard. In addition, because of more pipeline stages, we would have a much higher branch penalty for wrong branch prediction.
- When mitigating hazards using forwarding, the overheads added might not be worth the speed increase.

💡 Comments:

Question 5

Do pipelines always have five stages?

- No, it depends on the instruction type.
- For store instruction, we do not need to write back, so four stages.
- For branch instruction, we do not need to do memory access and write back. In the execution phase, we would read register operands and compare them using ALU; if the result is zero, then progress to the branch unit to directly update the program counter value.

- We do not need data access for arithmetic instruction (both `add` and `addi`) because the arithmetic or logical operation is performed in the ALU. The result would be written back directly to the destination register.

> 💡 Comments:

Question 6(y2016p5q2)

(a) Why is there a risk of data and control hazards in a pipelined processor? [4 marks]

- Data hazard:
  - Execute-to-execute data hazard: If the previous instruction would update a piece of data that a later instruction requires to read, then when the later instruction reaches the instruction decode phase, and the data is not written back yet by the previous instruction, then there would be a pipeline stall.
  - Load-use data hazard: if the previous instruction loads data into a register, but a later instruction requires the data, then when the later instruction reaches the instruction decode phase, and data is only available after the memory access phase of the previous load instruction, then there would be a pipeline stall.
- Control hazard:
  - We fetch the instructions and start executing them but later encounter a branch instruction, which indicates that the instructions we have fetched are wrong. Thus, we need to flush all fetched instructions, and a pipeline stall occurs.

(b) For modern RISC instruction set architectures like RISC-V, why are control hazards not exposed in the programming model? [4 marks]

- RISC-V has no branch delay slots where programmers can do code optimisation. On modern instruction set architecture, pipelines are more complex, with a much higher branch penalty(15-20 clock cycles), so a one clock-cycle branch delay slot is not worth implementing.
- It has mature branch prediction technology such as dynamic branch prediction, which uses hardware to measure actual branch behaviour and record the recent history of each branch, which helps reduce the number of control hazards.
- The unconditional jumps are undertaken in the instruction decode stage. The conditional branch is typically done in the execution stage so that the processor can know in advance about the flow of the instructions.

(c) How might data hazards be mitigated in each of the three pipelines below? [6 marks]

- Pipeline A:
  - There are no data hazards since register fetch would definitely get the value required as the execution, memory access, and write-back phases of previous instructions all happen in the same time slot.
- Pipeline B:
  - One data hazard, the register fetch phase, may require a value written back in previous instruction.
  - This can be mitigated using forwarding, which forwards the result of execution of the previous instruction to the register-fetch phase of the latter instruction.
- Pipeline C:
  - Two data hazards:
    - The register fetch phase may require a value loaded from memory, which can only progress after the memory access phase. This Load-use data hazard can not be avoided entirely but can be mitigated using forwarding, which forwards the result of the memory access phase to the register-fetch phase. A possible solution is to reorder the codes in the compiler to avoid using the load result in the next instruction.
    - The register fetch phase may need the value written back by a previous instruction, so it can only advance after the write-back phase. This execute-to-execute data hazard can be mitigated using forwarding, which forwards the result of execution of the previous instruction to the register-fetch phase of the latter instruction.

Pipeline A:

instruction fetch	decode	register fetch execute memory access write-back

Pipeline B:

instruction fetch	decode register fetch	execute memory access write-back

Pipeline C:

instruction fetch	decode register fetch	execute	memory access	write-back

(d) Do exceptions introduce a control hazard? Give justification for your answer. [3 marks]

- Yes. Exceptions arise within the CPU (e.g., undefined opcode or system calls). They are unexpected events that change instruction execution flow; we need to jump to instructions pointed by the exception handler, which means we have to flush all fetched instructions, and a pipeline stall occurs.
- The program counter value is saved, so it would jump back to the instruction and resume execution after the handler is executed.

(e) Do interrupts introduce a control hazard? Give justification for your answer. [3 marks]

- Yes. Interrupts arise from an external I/O controller. They would change the flow of instruction execution as well. We might need to save the PC value, do a context switch and then jump to the instruction pointed by the interrupt handler, which implies we have to flush all the instructions in the pipeline a pipeline stall occurs.

💡 Comments:

Question 7

What is branch prediction, and how might it be used to avoid control hazards?

- Branch prediction: always predict the outcome of a branch; the pipeline would only get stalled if the prediction is wrong (i.e., 50% chance of getting the prediction right)
- Static branch prediction would make predictions based on typical branch behaviour (e.g., for loops, it would predict backward branches, which are much more common)
- Dynamic branch prediction would use hardware to measure actual branch behaviour and record the recent history of each branch.
  - It assumes the future will continue the trend when executing a branch.
  - If the prediction is correct, we will avoid control hazards. There would not be any pipeline stalls; if the forecast is wrong, then stall the CPU and re-fetching, an update history by flushing the pipeline and flip prediction. Hence next time we encounter the same branch, we will update our behaviours.

💡 Comments:

Question 8(y2021p5q2)

(c) In detail, describe the hardware mechanisms that are required to support virtual memory for a pipelined processor implementing the 32-bit RISC-V ISA. [6 marks]

- Translation lookaside buffer:
  - A hardware cache used to store recently used page table entries to reduce the number of memory accesses and improve processor performance
- Hardware page table walker:
  - TLB misses causing the hardware page table walker to search for the page table entries. If found, they would get put into the TLB; otherwise, it would result in an exception.
- Valid and permission bits:
  - Enable different modes, privileges instruction and page table information can only be executed or accessed in supervisor mode to ensure safety
- Registers which store pointers to page tables:
  - Would change during context switching to ensure a process would not access other processes' memory via their page tables

💡 Comments:

Question 9(y2019p5q2)

(a) What is the difference between memory latency and bandwidth? [2 marks]

- Memory latency is the time between the request sent to the processor and the request received by the processor.
- Bandwidth is the maximum amount of data sent over per unit of time (maximum bit rate).

(b) What is the difference between an interrupt and an exception? [3 marks]

- Interrupts usually are raised from an external I/O controller, while exceptions usually are raised within the CPU.
- Interrupts may require a context switch to store all the register values, process control block information, while exceptions occur in the same program, so we only need to save the program counter value so that we can resume after handling the exceptions.

(c) Why is a computer's memory implemented as a hierarchy of memories and yet the programming model represents physical memory as a flat linear address space? [5 marks]

- We have trade-offs for different cache sizes.
- Increasing the cache size would decrease capacity misses as we have more spaces to store words but increase access time.
- Increasing the associativity on the cache design could decrease conflict misses but increase access time.
- Increasing the cache line size would decrease compulsory misses as we can store more entries at the start. However, a large cache line size may increase the miss rate due to pollution and increase the miss penalty as we need to load more words for a more extensive cache line.
- Hence a hierarchical memory structure could help reduce the miss penalty and ensure good performance.

(d) What is the difference between a direct-mapped and a set-associate cache, and how do their cache-line replacement policies differ? [5 marks]

- A direct-mapped cache maps each memory address to a single cache line index; hence, it is easier to implement a replacement policy(directly replacing the current one stored in the cache line) and have a higher power efficiency and more direct hardware support. However, it may introduce more conflict misses and a lower hit rate as only one cache line stores the data for an address.
- A set-associative cache combines the efficiency of direct-mapped cache and the higher hit rate of fully associative cache. For a n-way associative cache, the cache is split into n sets; each memory address would map to a set but can be placed anywhere inside the set. Hence the replacement policy can be more flexible and efficient; we can use the Not-last-used algorithm to choose a victim for replacement, avoid some of the pathological cases (i.e., the replaced data is required afterwards)

💡 Comments:

Question 10(y2009p5q3)

(a) Why are control-flow machines sensitive to memory access latency? [4 marks]

- Control-flow machines execute sequential instructions; most of the instructions have a memory access phase after execution.

- Since the memory access speed is comparably slower than other phases, it significantly impacts the processor performance. Hence a high memory access latency implies a bad processor performance.

(*b*) What statistical properties of data access patterns do caches exploit to reduce memory access latency? [4 marks]

- Temporal locality:
  - Items accessed recently are more likely to be reaccessed soon
  - For example, instructions in a loop, loop counter and other repeatedly used variables would be accessed frequently in a particular time slot.
- Spatial locality:
  - Items near those accessed recently are more likely to be reaccessed soon
  - For instance, sequential instruction and array data show spatial locality, and this property fits well with DRAM bursts where we would fetch data requested and its neighbours together.

(*c*) What cache line replacement policies might be used for set-associative and direct-mapped caches? [4 marks]

- The set-associative cache can use the Not-last-used replacement policy, which chooses a victim that is not lastly used. Since an address would map to a set and be placed anywhere inside the set, we have more choices to select a victim entry. The Not-last-used policy approximates the Least-recently-used policy but is simpler to implement for more than an 8-way associative cache.
- The direct-mapped cache can have a simple replacement policy where we replace the one currently stored in the cache line when we hash to it. We can further improve the policy by adding a victim buffer, which would hold the evicted cache line and give it a second chance.

(*d*) What are two write-back policies for a cache? [4 marks]

- On a data-write hit, write-back would update the cache line directly and track whether each is dirty. When a dirty block needs to be replaced with a new cache line, we write it back to memory. We can also use a write buffer to allow the replacement cache-line to be read first and place the old cache line data into the write buffer.
- On a write miss, write-back would allocate on a miss by fetching the cache-line to avoid having a cache line where some words or bytes are invalid. It keeps track of the dirty cache line and writes it back to memory only if a dirty block needs to be replaced.

💡 Comments:

Question 11

What is cache miss rate, and why must it be very low if it does not significantly impact processor performance?

$$\text{Miss rate} = \frac{\text{Num. cache miss}}{\text{Num. memory accesses}}$$

- A high cache miss rate implies many memory accesses are from the main memory, not the cache. Since memory access from the main memory requires much more clock cycles than memory accesses from the cache, the execution time for instructions would increase, thus fewer throughputs and worse processor performance.

💡 Comments:

Question 12

Why can pipeline make the execution of a single instruction slower and a set of instructions faster?

- A pipeline is specified to have a particular number of stages (e.g., a 5-stage pipeline), but some of the instructions would take less than that. Hence, the critical path (i.e., the load instruction) determines the clock period. Ideally, all instruction would take five clock cycles, which slows down the single instruction if it is not a load instruction.
- In the worst case, a set of instructions (say n instructions) with sequential execution would take $5n$ clock cycles. However, with a pipeline, we fetch one instruction for each clock cycle, so ideally, the set of instructions would take $(n + 4)$ clock cycles. By keeping all pipeline stages busy and executing in parallel while preserving the programmer's model of sequential execution, we could improve the performance vastly.

💡 Comments:

Question 13

Discuss the sources of cache misses.

- Compulsory misses occur when we first access a block.

- Capacity misses due to finite cache size, a block being replaced is later reaccessed, the limited capacity leads to the cache miss.
- Conflict misses due to competition for entries in a set in a non-fully associative cache.

💡 Comments:

Question 14

Summarise the main message from lesson 5 in 1-3 sentences?

- The RISC-V instruction types and their various immediate encoding formats are designed to simplify hardware decoding.
- ISA level simulation can help to debug, reveal the random error and provide hotspot analysis. Spike is a golden reference to RISC-V instructions.

💡 Comments:

Question 15

Summarise the main message from lesson 6 in 1-3 sentences?

- The data path is fixed, and control determines how each part of the data path is used to perform each instruction, so these two paths go down the pipeline together.
- The purpose of the five-stage pipeline is used to improve execution performance.

💡 Comments:

Question 16

Summarise the main message from lesson 7 in 1-3 sentences?

- Structural, data and control hazards might occur and produce pipeline stalls.
- We use forwarding, code reordering, and branch prediction to mitigate hazards on pipelines.

💡 Comments:

Question 17

Summarise the main message from lesson 8 in 1-3 sentences?

- The statical properties of programs, including the temporal locality and spatial locality, impact the design of caches.
- The direct-mapped cache is simpler to implement and power-efficient but leads to a high conflict miss and lower hit rate. In contrast, the fully associative cache provides a higher hit rate, the flexibility of replacement policies but is power-hungry and complex to implement. Hence, we use a set-associative cache design to combine the pros of these two types.

> 💡 Comments:

Question 18(*)

Create an example in https://github.com/swm11/riscv-lite/tree/master/jsim