CENG 3420
Computer Organization & Design

Lecture 09: Pipeline – Basis

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Textbook: Chapters 4.5 & 4.6)

2024 Spring

Motivations

- Single cycle: the whole datapath is finished in one clock cycle

- It is simple and easy to understand

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instr

- Problematic for more complex instructions like floating point multiply

- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle
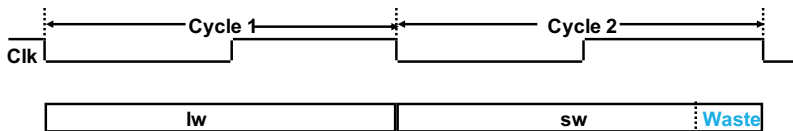
- Though simple, the single cycle approach is not used because it is very slow
- Clock cycle must have the same length for every instruction
- What is the longest path (slowest instruction)? Load instruction!
- It is too long for the store instruction so the last part of the cycle here is wasted.

EX: Instruction Critical Paths

Calculate cycle time assuming negligible delays (for muxes, control unit, sign extend, PC access, shift left 2, wires) except:

- Instruction fetch and update PC (IF), Read/write data from/to data memory (MEM) (4 ns)

- Execute R-type; calculate memory address (EXE) (2 ns)

- Register fetch and instruction decode (ID), Write the result data into the register file (WB) (1 ns)

Instr.	IF	ID	EXE	MEM	WB	Total
R/I-type						
lw						
sw						
beq						
jal						
jalr						

EX: Instruction Critical Paths

Calculate cycle time assuming negligible delays (for muxes, control unit, sign extend, PC access, shift left 2, wires) except:

- Instruction fetch and update PC (IF), Read/write data from/to data memory (MEM) (4 ns)

- Execute R-type; calculate memory address (EXE) (2 ns)

- Register fetch and instruction decode (ID), Write the result data into the register file (WB) (1 ns)

| Instr.	IF	ID	EXE	MEM	WB	Total
R/I-type	4	1	2		1	8
lw	4	1	2	4	1	12
sw	4	1	2	4		11
beq	4	1	2			7
jal						
jalr						

$$\text{CPU time} = \text{CPI} \times \text{CC} \times \text{IC}$$

- Start fetching and executing the next instruction before the current one has completed
  - Pipelining – (all?) modern processors are pipelined for performance
  - Under ideal conditions and with a large number of instructions, the speedup from pipelining is approximately equal to the number of pipe stages
  - A five stage pipeline is nearly five times faster because the CC is "nearly" five times faster

- Fetch (and execute) more than one instruction at a time
  - Superscalar processing – stay tuned

Pipeline Basis
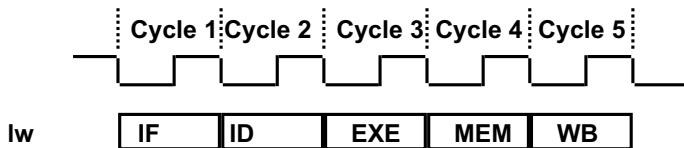
- **IF**: Instruction Fetch and Update PC
- **ID**: Registers Fetch and Instruction Decode
- **EXE**: Execute R-type; calculate memory address
- **MEM**: Read/write the data from/to the Data Memory
- **WB**: Write the result data into the register file

A Pipelined RISC-V Processor

Start the next instruction before the current one has completed

- improves throughput - total amount of work done in a given time
- instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is not reduced
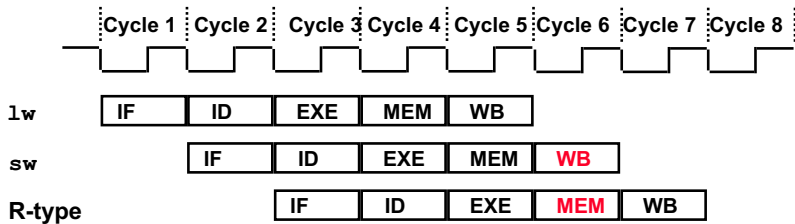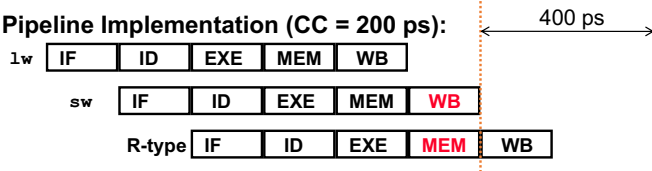

1. clock cycle (pipeline stage time) is limited by the slowest stage
2. for some stages don't need the whole clock cycle (e.g., WB)
3. for some instructions, some stages are wasted cycles (i.e., nothing is done during that cycle for that instruction)

Single Cycle Implementation (CC = 800 ps):

Clk
Cycle 1 — Cycle 2

lw   sw   **Waste**

400 ps
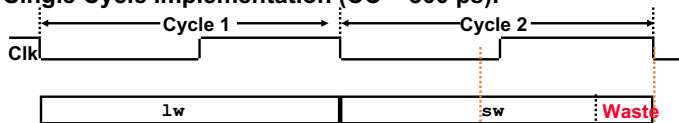
Pipeline Implementation (CC = 200 ps):

lw	IF	ID	EXE	MEM	WB

| sw | IF | ID | EXE | MEM | **WB** |

| R-type | IF | ID | EXE | **MEM** | WB |

- To complete an entire instruction in the pipelined case takes 1000 ps (as compared to 800 ps for the single cycle case). Why ?

- How long does each take to complete 1,000,000 adds ?

What makes it easy

- all instructions are the same length (32 bits)
    - can fetch in the 1st stage and decode in the 2nd stage
- few instruction formats (three) with symmetry across formats
    - can begin reading register file in 2nd stage
- memory operations occur only in loads and stores
    - can use the execute stage to calculate memory addresses
- each instruction writes at most one result (i.e., changes the machine state) and does it in the last few pipeline stages (MEM or WB)
- operands must be aligned in memory so a single data transfer takes only one data memory access

State registers between each pipeline stage to isolate them
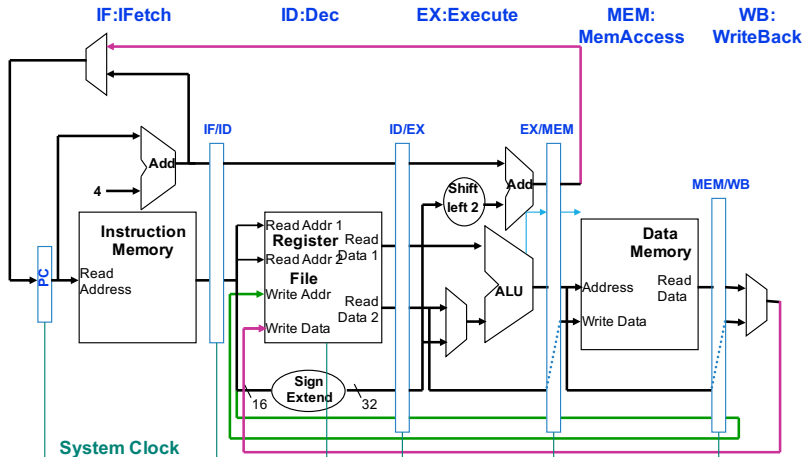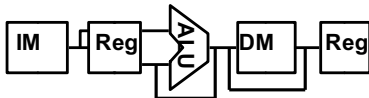
Can help with answering questions like:

- How many cycles does it take to execute this code?

- What is the ALU doing during cycle 4?

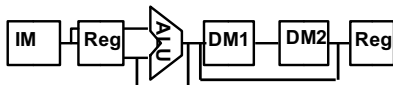- Is there a hazard, why does it occur, and how can it be fixed?

What about the (slow) multiply operation?

- Make the clock twice as slow or ...
- let it take two cycles (since it doesn't use the MEM stage)
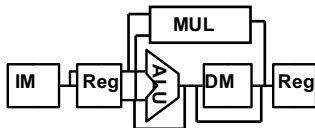
What if the data memory access is twice as slow as the instruction memory?

- make the clock twice as slow or ...
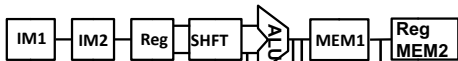- let data memory access take two cycles (and keep the same clock rate)

- ARM7:

PC update   decode
IF access   Reg access

ALU op MEM access shift/rotate
commit result (write back)

- XScale:

PC update   decode
BTB access   reg 1 access
start IF access   ALU op   MEM write
                 reg write

IF access   shift/rotate   start MEM access
        reg 2 access   exception

Time (clock cycles)

I n s t r. O r d e r

Inst 0
Inst 1
Inst 2
Inst 3
Inst 4

Once the pipeline is full, one instruction is completed every cycle, so CPI = 1

Time to fill the pipeline
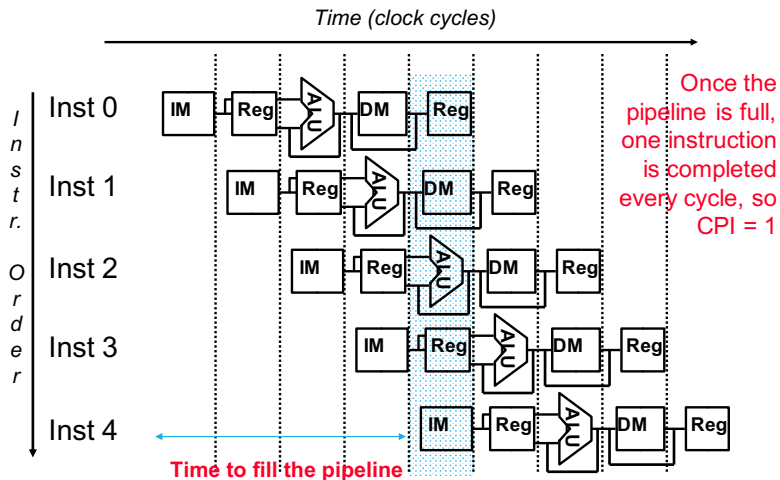
Structural Hazards

Yes! Pipeline Hazards

- **structural** hazards: a required resource is busy

- **data** hazards: attempt to use data before it is ready

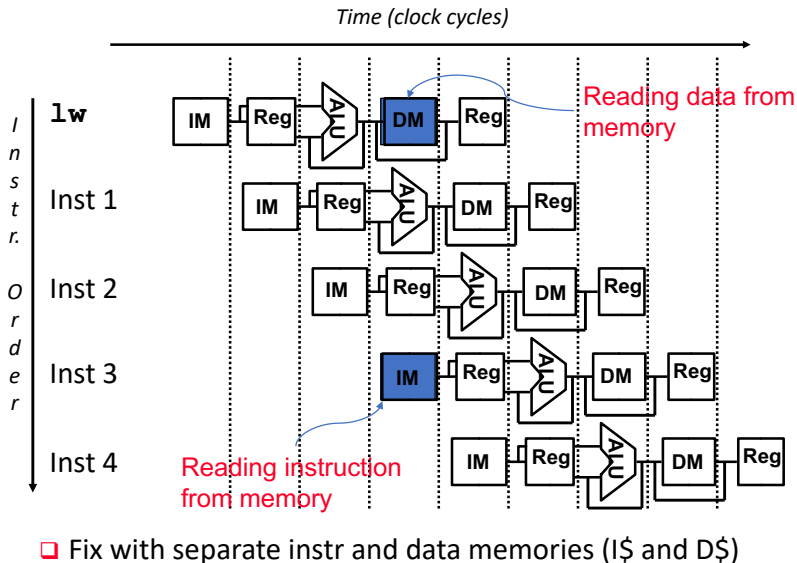- **control** hazards: deciding on control action depends on previous instruction

Can usually resolve hazards by waiting

- pipeline control must **detect** the hazard

- and take action to **resolve** hazards

- Conflict for use of a resource
- In RISC-V pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch requires instruction access
- Hence, pipeline datapaths require separate instruction/data memories
  - Or separate instruction/data caches
- Since Register File

Time (clock cycles)

Reading data from memory

Reading instruction from memory

❑ Fix with separate instr and data memories (I$ and D$)

Fix with separate instr and data memories (I$ and D$)

Time (clock cycles)

Instr. Order

add $1,

Inst 1

Inst 2

add $2,$1,

Fix register file access hazard by doing reads in the second half of the cycle and writes in the first half

clock edge that controls register writing

clock edge that controls loading of pipeline state registers