W2D2 note

written by czj.

Data stationary control

The original MIPS Datapath cannot be used for pipelining, because the instruction execution speed is not the same. The output of the instruction that execute faster may affect the input of the instruction before it.
The method to improve it is adding inter-mediate/transfer registers between stages. And it works when it receives a pulse.

Pipelining and Hazard

- pipelining: Divide the entire instruction execution process into five stages and different stages of different instructions work concurrently in one cycle.
- Hazards: The design limitation of the pipelining. It prevents the next instruction from executing during its designated clock cycle.

  The types of hazards include: structural hazards, data hazards and control hazards.

Structural Hazard

- Structural hazard happens when multiple instructions need to use the same component(exclusive) at one cycle.
  For example, the first instruction is the load operation which need to access the memory at Cycle 4 and the third instruction needs to get the instruction code from the memory at Cycle 4. Here's the contradiction.

Copying the component can solve the problem. In this example, the memory is divided into I-Mem and D-Mem. I-Mem is used for storing the instructions and D-Mem is used for storing the data. Also, adding bubbles between instructions can also be a possible solution.

Data Hazard

- Data hazard happens when the calculation related to instruction depends on the previous calculation results or the calculation result is covered by the instructions which compute slower in out-of-order execution. There're three kinds of data hazards.
- RAW: Read After Write. In the example below, $\mathbf{Instr_J}$ tries to read operand before $\mathbf{Instr_I}$ writes it and it causes a data hazard on $\mathbf{r_1}$.
  $\mathbf{I: add\ r_1, r_2, r_3}$
  $\mathbf{J: sub\ r_4, r_1, r_3}$
- WAR: Write After Read. In the example below, $\mathbf{Instr_J}$ writes operand before $\mathbf{Instr_I}$ reads it and it causes a data hazard on $\mathbf{r_1}$.
  $\mathbf{I: add\ r_2, r_1, r_3}$
  $\mathbf{J: sub\ r_1, r_4, r_5}$
- WAW: Write After Write. In the example below, $\mathbf{Instr_J}$ writes operand before $\mathbf{Instr_I}$ writes it and it causes a data hazard on $\mathbf{r_1}$.
  $\mathbf{I: add\ r_1, r_2, r_3}$
  $\mathbf{J: sub\ r_1, r_4, r_5}$

It's clear that WAR and WAW can't happen in the pipelining because instructions are executed in order and read always precedes write. It will happen when CPU execute instructions in out-of-order execution.
The possible solution of RAW is forwarding, meaning that the data can be forwarded to the input directly. The possible solution of WAR and WAW is renaming, meaning that the register $\mathbf{r_1}$ is renamed so that it can be reused.

Control Hazard

- Control Hazard caused by the delay between the fetching of instructions and decisions about the changes in control flow(branches and jumps).