Overview

- Pipelined datapath structure
- Pipelined datapath operations
- Pipelined control

Pipelining: Implementation (I)

Lecturer: Hui Annie Guo
huig@cse.unsw.edu.au
K17-501F (ext. 57136)

Pipelining datapath

- Recall the datapath execution can be divided into five stages:
  1. IF: Instruction fetch
  2. ID: Instruction decode and register file read
  3. EX: Execution or address calculation
  4. MEM: Data memory access
  5. WB: Write back

- This division naturally leads to a five-stage pipeline, which means that up to five instructions can be processed during a single clock cycle

Notes on the single cycle datapath

- Each processing step can be mapped onto the datapath from left to right. Exceptions are
  - The PC update step, which sends the ALU result to the left, and
  - The write back step, which sends data from memory to the register file

- Data flowing from right to left does not affect the current instruction; only later instructions in the pipeline are influenced by these reverse data movements, which can lead to control and data hazards, respectively.
Single cycle datapath

Execution Timing Diagram

• We can see pipelined execution by pretending each instruction has its own datapath, and placing each of these datapaths on a timeline to show their timing relationship.

Execution Timing Diagram

Program execution order (in instructions)

Program execution order (in instructions)

Pipelined Datapath

• To retain the value of an individual instruction for its other four stages, the value read from instruction memory must be saved in a register. The same is for other stages.
Notes on the pipelined datapath

- The registers inserted between the pipeline stages are called pipeline stage registers.
- Each stage register is named after the two stages separated by that register
  - E.g. The register separating the IF and ID stages is called the IF/ID register
- All instructions advance from one pipeline register to the next on each clock cycle.
- The write back stage does not need a separate register since the register file is updated at the end of that stage – a separate register would be redundant.
- An example of pipeline execution is given in the following slides.
Pipelined LW execution: Memory access

Notes on pipelined LW execution

- The previous five slides show the active portions of the datapath as a load instruction progresses through the pipeline.
- The **right half** of registers are highlighted as they are being read and the **left half** is highlighted as they are being written.
- In detail, the five stages are working as follows:
  1. **Instruction fetch**: The instruction is read from memory using the address in the PC and is stored in IF/ID. PC is incremented by 4 and written back ready for the next instruction. The new PC value is stored in IF/ID for instructions such as `beq`. We store all data that may be needed by subsequent stages.

Notes on pipelined LW execution

- Pipeline stage details continued:
  2. **Instruction decode and register file read**: The immediate field is retrieved from IF/ID and sign extended; the two source registers are read and all three values are stored in ID/EX along with everything else that may be needed by the instruction during a later clock cycle.
  3. **Execute or address calculation**: The contents of register 1 and the sign-extended immediate from the ID/EX register are added using the ALU and the result is placed in the EX/MEM pipeline register.
  4. **Memory access**: Data memory is read using the address from EX/MEM; the read data is loaded into the MEM/WB pipeline register.
  5. **Write back**: The data is read from MEM/WB and written to the register file.
Other instructions?

- Let’s just look at how a store instruction differs from a load instruction.
### SW instruction execution

- The first three stages are identical to those for the load instruction, but the following two stages become:
  4. **Memory access**: the data is written to memory. In order to make the data available during the MEM stage it had to be placed into the EX/MEM register during the EX stage.
  5. **Write back**: for this instruction, nothing happens in the write back stage.

### Key points

- To pass something from an early pipe stage to a later pipe stage, the information must be placed in a pipeline register; otherwise the information is lost when the next instruction enters that pipeline stage.
- Each logical component of the datapath – such as instruction memory, register read ports, ALU, data memory, and register write port – can be used only within a single pipeline stage; otherwise we would have a **structural hazard**. Hence these components, and their control, are associated with a single pipeline stage.
Bug in the design of write back stage?

Correct design: pipe the destination register number through the pipeline registers as well!

Pipelined control

• Add control to pipelined datapath just like we added it to the single cycle datapath:
  1. Label control points
  2. Determine the control settings for each stage of execution of every instruction
  3. Design control logic to implement the control
     • To handle sequencing, control inputs are piped together with data and intermediate results

Control points in the pipelined datapath
Notes:

- To specify control for the pipeline, we need only set the control values during each pipeline stage.
- Since each control line is associated with a component that is active in only one pipeline stage, we divide the control lines into five groups according to the pipeline stage:
  - Instruction fetch: IM read and PC write signals always asserted, so nothing to control.
  - Instruction decode/register file read: The same function occurs every cycle, so no control is necessary.
  - Execution/address calculation: RegDst, ALUOp, and ALUSrc need to be specified.
  - Memory access: Branch, MemRead, and MemWrite need to be specified.
  - Write back: MemtoReg and RegWrite need to be specified.
- Since the pipeline registers are updated each cycle, there is no need to control these.

Recall: Control unit of single cycle processor

- We can use the same design for pipelined datapath

Pipelining the control for the final 3 stages

- Since the control lines start with the EX stage, we can create the control information during the instruction decode stage
- These control signals are then used in the appropriate pipeline stage as the instruction moves down the pipeline

Putting it All Together: Pipelined Processor
Example of Pipeline Execution

• How are the following instructions executed

\[
\begin{align*}
\text{lwa} & \quad \text{lw} \\
\text{sub} & \quad \text{sub} \\
\text{and} & \quad \text{and} \\
\text{or} & \quad \text{or} \\
\text{add} & \quad \text{add} \\
\end{align*}
\]

\[
\begin{align*}
\text{lwa} & \quad \text{lw} \\
\text{sub} & \quad \text{sub} \\
\text{and} & \quad \text{and} \\
\text{or} & \quad \text{or} \\
\text{add} & \quad \text{add} \\
\end{align*}
\]
Execution Example: clock cycle 8/9

IF: after <3>
ID: after <2>
EX: after <1>
MEM: add $14, $8, $9
WB: or...

Execution Example: clock cycle 9/9

IF: after <4>
ID: after <3>
EX: after <2>
MEM: add $14, $8, $9
WB: add...