Introduction to Pipelining

(Notes based on: *Computer Architecture: A Quantitative Approach, 5th Edition*, John L. Hennessy and David A. Patterson, Morgan Kaufmann, 2012)

Multiple instructions in different stages of processing at the same time.

Similar to an assembly line: Break a complex process into several, simpler steps. Each step completes part of an instruction's processing. If no stalls, an instruction completes on every clock cycle (assuming each step completes in one clock cycle).

Each step is called a pipe stage.

Throughput: measured by how often an instruction exits the pipeline (i.e., completes).

Processor cycle time is the time for processing one step of the pipeline. It is usually 1 clock cycle, sometimes 2, rarely more than 2.

Pipe stages should be balanced (i.e., each stage requires approximately the same amount of time). Otherwise, slowest step determines the processor cycle time.

If pipeline is perfectly balanced, then time per instruction is:

Time per instruction (unpipelined) / Number of pipe stages

so that the speed-up equals the number of pipe stages. This is an ideal value, but it can be approached in a properly design pipeline.

We will use the MIPS64 instruction set to illustrate the concepts. Divide the instructions into 3 classes:

(i) ALU instructions: Operands in two registers, or one register and an immediate value, with the result stored in a register.

(ii) Load/store instructions: Compute an effective address (base register + immediate offset). For a load instruction, copy memory to a destination register. For a store instruction, copy from a register to memory.

(iii) Branches/jumps: Branches are conditional, jumps are unconditional. For conditional branches, MIPS does not use condition code bits. Instead, registers are compared. The branch target address is computed by adding a sign-extended offset to the PC.

Divide the instruction processing into 5 stages:

IF (instruction fetch)

ID (instruction decode/register fetch)

EX (execute/effective address computation)

MEM (memory access)

WB (write back)

When the pipeline is full, 5 different instructions will be in 5 different stages of execution. We need to make sure that the same hardware resource is not needed by two different instructions (i.e., in two different pipe stages) at the same time.

IF uses the instruction memory (IM), normally an instruction cache

ID uses the register file

EX uses the ALU

MEM uses the data memory (DM), normally a data cache

WB uses the register file

Note that the register file is used in both ID (for reading) and WB (for writing). To avoid a conflict, we assume that the write happens in the first half of the clock cycle and the reads happen in the second half of the clock cycle. Also, we need to have two read ports and one write port on the register file.

We also need to insert pipeline registers between each pipe stage to save the intermediate results from the preceding pipe stage. These pipeline registers are not visible to the programmer. The registers are named according to the pipe stages that they connect, i.e.: IF/ID, ID/EX, EX/MEM and MEM/WB.

The PC is incremented during the IF stage. Also, there is a separate adder that is used for compute the branch target address during ID.

Practical limits on number of pipe stages: limits on imbalances, pipeline register set-up/hold times, clock skew, interactions and conflicts between the pipe stages.