

Pipelining

Implementation technique

- overlaps execution of different instructions
- increases instruction throughput by executing several instructions “in parallel”
- abstract model: an assembly line

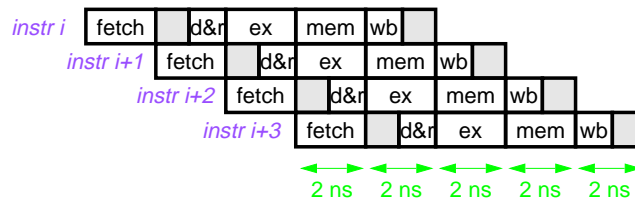
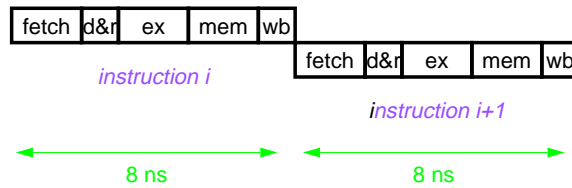
Pipelining

Divide the execution cycle into **stages**

- each stage takes one clock cycle
 - cycle time limited by the longest stage
 - want each stage to do equal work
- instructions advance down the pipeline together
 - want all instructions to do the same amount of total work

In what ways has a RISC architecture been designed for pipelining?

Pipelining



Pipeline Datapath

Stages are independent & isolated from each other

- information generated or retained at stage *i* that is needed in stage *i+1* must be stored in **pipeline registers** that separate each stage
- otherwise subsequent instruction will overwrite information
- examples of information stored in pipeline registers:
 - the destination register number for an ALU result
 - the source register number for the **sw** data
 - offset of immediate value
 - the updated PC
 - control lines

All 5 stages are active at the same time

- cannot share resources among stages
 - similar to the single-cycle datapath
- need pipeline registers to hold the results of one stage for the next stage
 - similar to the multiple-cycle datapath

Pipelining Performance

Pipelining increases instruction throughput

- new instruction can start & complete each cycle (once you fill the pipeline & if there are no pipeline stalls)
- therefore execution time of a program decreases
- decreasing throughput is the most important performance effect

Pipelining slightly increases the latency of an individual instruction

- each instruction takes the maximum number of stages
- the longest stages sets the cycle time
- pipeline registers need to be written and read

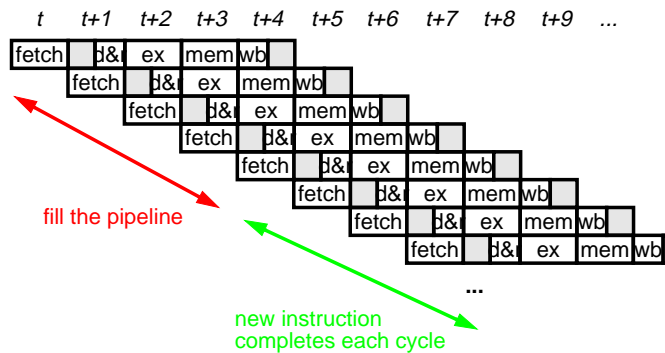
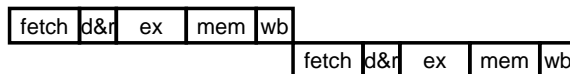
$$T_{\text{without pipeline}} = \text{instructions} \cdot n \text{ cycles per instruction}$$

$$T_{\text{pipelining}} = \text{instructions} + n - 1 \text{ cycles}$$

$$\text{Speedup} = \frac{T_{\text{without pipeline}}}{T_{\text{pipelining}}} = \# \text{ of pipe stages}$$

- assumes:
 - no pipeline stalls
 - no pipeline overhead

Pipelining



Tracing an Instruction Through the Pipeline

Stage 1: instruction fetch & PC increment

- PC is used to access memory
fetch instruction
put instruction into IF/ID (equivalent to IR)
- increment PC
put incremented PC into IF/ID

Main resources needed:

- instruction memory
- an ALU

IF/ID (64 bits) contains:

- instruction
- incremented PC

Tracing an Instruction Through the Pipeline

Stage 2: instruction decode & register read

- decode instruction & read source registers
put register values into ID/EX (equivalent to registers A & B)
- sign-extend immediate field & put into ID/EX

Main resources needed:

- register file

ID/EX (147 bits) contains:

- 2 register values
- sign-extended immediate
- rt, rd
- incremented PC
- control for the next 3 stages
 - ALUop & func field (which ALU operation)
ALUsrc (which operand for the second ALU source)
RegDst (which destination register)
 - Branch (in case this is a `beq`)
MemRead (read from memory)
MemWrite (write to memory)
 - RegWrite (write the register file)
MemtoReg (where the value comes from)

Tracing an Instruction Through the Pipeline

Stage 3: execute

- do an ALU operation
put result & Zero into EX/MEM
- calculate the target address
put target into EX/MEM

Main resources needed:

- 2 ALUs

EX/MEM (139 bits) contains:

- ALU result
- target address
- store value
- destination register # (the one that has been chosen)
- Zero line
- incremented PC (*for exceptions: later*)
- control for the next 2 stages
 - Branch (in case this is a `beq`)
MemRead (read from memory)
MemWrite (write to memory)
 - RegWrite (write the register file)
MemtoReg (where the value comes from)

Tracing an Instruction Through the Pipeline

Stage 4: data memory access

- use effective address in EX/MEM to access memory
store data from EX/MEM if `sw`
put loaded value into MEM/WB if `lw`
- determine if branch should be taken
- R-type instructions do nothing

Main resources needed:

- data memory

MEM/WB (34 bits) contains:

- ALU or `lw` result
- control for the last stage
 - RegWrite (write the register file)
MemtoReg (where the value comes from)

Tracing an Instruction Through the Pipeline

Stage 5: register write

- write loaded value or ALU result
- `sw` & `beq` instructions do nothing

Main resources needed:

- register file

No pipeline register

More on Control Signals

No control signals needed to

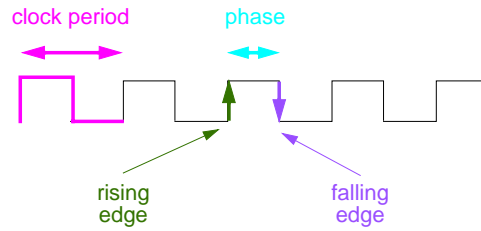
- write PC
- read instruction memory
- write pipeline registers

Control signals are computed in ID & propagated to later stages in the pipeline registers

Clocking

Clock: free-running signal with a fixed cycle time

- typically divided into 2 **clock phases**
 - clock signal high
 - clock signal low
- **edge-triggered clocking**
 - state changes occur on a clock edge
 - write register file on a rising edge
 - read register file on a falling edge



Register Reading & Writing

