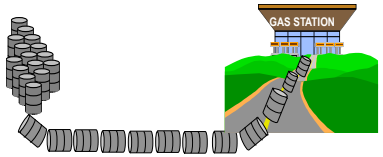
Lecture 10
Pipelining



Fall 2016

Pejman Lotfi-Kamran

Slides by Michael Ferdman @ Stony Brook University

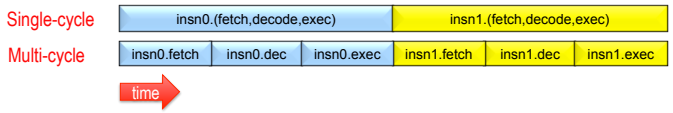
Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

This Lecture
- Pipelining

Next Lecture:
- Evaluation

Before there was pipelining…

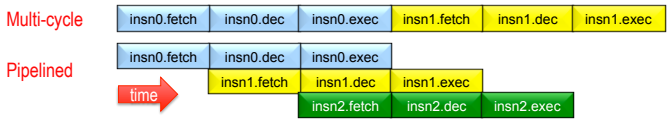


Single-cycle control: hardwired
- Low CPI (1)
- Long clock period (to accommodate slowest instruction)

Multi-cycle control: micro-programmed
- Short clock period
- High CPI

Can we have both low CPI and short clock period?

Pipelining

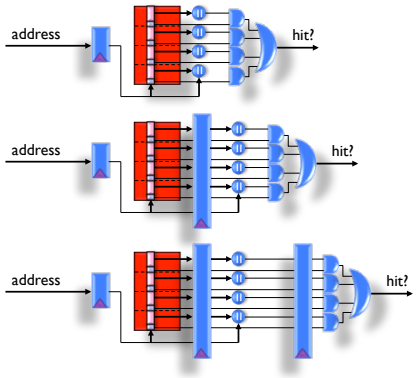


Start with multi-cycle design

When insn0 goes from stage 1 to stage 2
… insn1 starts stage 1

Each instruction passes through all stages
… but instructions enter and leave at faster ***rate***

Pipeline Examples

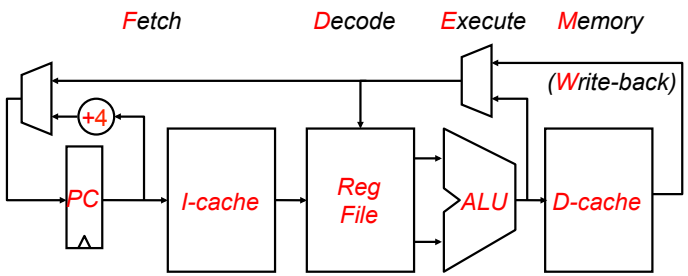


Stage delay = n
Bandwidth = $\sim(1/n)$

Stage delay = $n/2$
Bandwidth = $\sim(2/n)$

Stage delay = $n/3$
Bandwidth = $\sim(3/n)$

Processor Pipeline Review

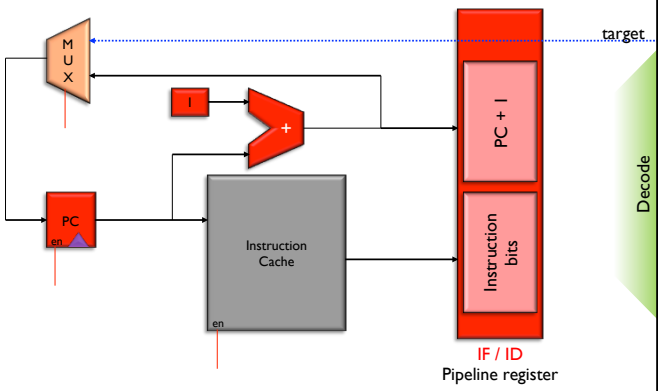


Stage 1: Fetch

Fetch an instruction from memory every cycle
- Use PC to index memory
- Increment PC (assume no branches for now)

Write state to the pipeline register (IF/ID)
- The next stage will read this pipeline register

Stage 1: Fetch Diagram



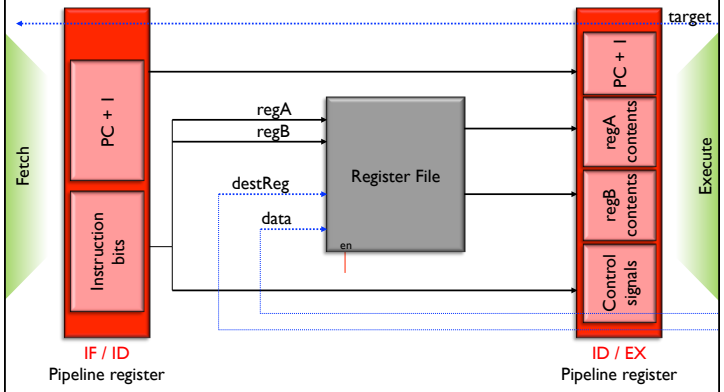
Stage 2: Decode

Decodes opcode bits
- Set up Control signals for later stages

Read input operands from register file
- Specified by decoded instruction bits

Write state to the pipeline register (ID/EX)
- Opcode
- Register contents
- PC+1 (even though decode didn't use it)
- Control signals (from insn) for opcode and destReg

Stage 2: Decode Diagram

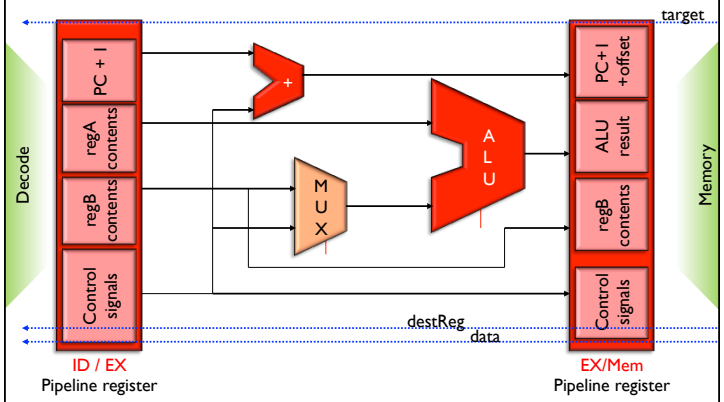


Stage 3: Execute

Perform ALU operations
- Calculate result of instruction
  - Control signals select operation
  - Contents of regA used as one input
  - Either regB or constant offset (from insn) used as second input
- Calculate PC-relative branch target
  - PC+1+(constant offset)

Write state to the pipeline register (EX/Mem)
- ALU result, contents of regB, and PC+1+offset
- Control signals (from insn) for opcode and destReg

Stage 3: Execute Diagram

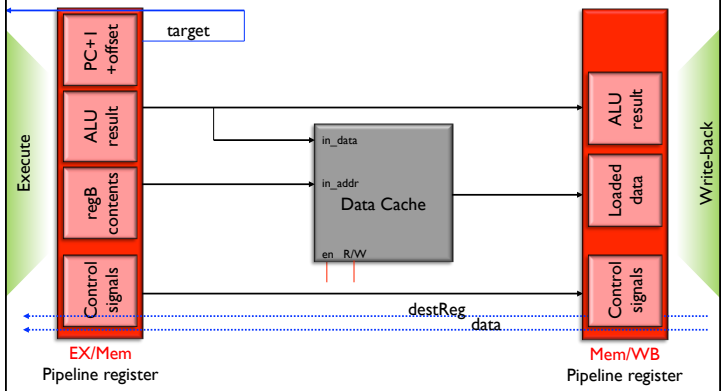


Stage 4: Memory

Perform data cache access
- ALU result contains address for LD or ST
- Opcode bits control R/W and enable signals

Write state to the pipeline register (Mem/WB)
- ALU result and Loaded data
- Control signals (from insn) for opcode and destReg

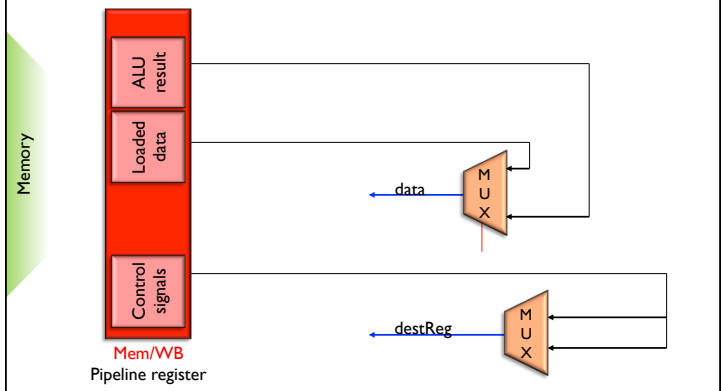
Stage 4: Memory Diagram



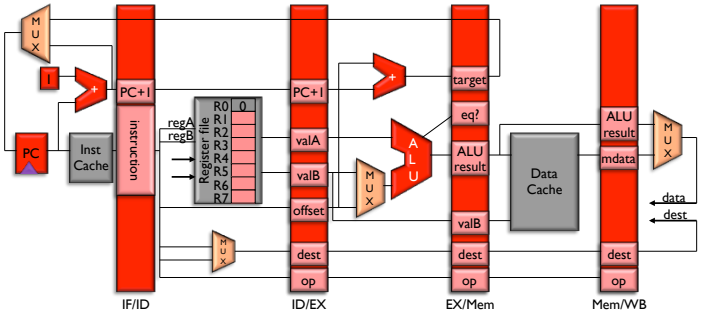
Stage 5: Write-back

Writing result to register file (if required)
- Write Loaded data to destReg for LD
- Write ALU result to destReg for arithmetic insn
- Opcode bits control register write enable signal

Stage 5: Write-back Diagram



Putting It All Together



Pipelining Idealism

Uniform Sub-operations
- Operation can partitioned into uniform-latency sub-ops

Repetition of Identical Operations
- Same ops performed on many different inputs

Repetition of Independent Operations
- All repetitions of op are mutually independent

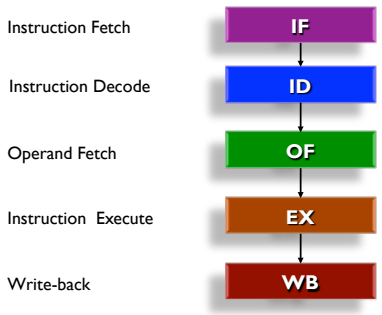
Pipeline Realism

Uniform Sub-operations … NOT!
- Balance pipeline stages
  - Stage quantization to yield balanced stages
  - Minimize internal fragmentation (left-over time near end of cycle)

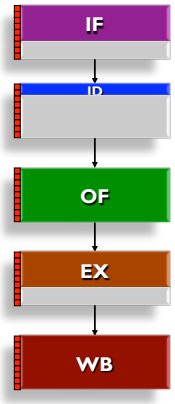
Repetition of Identical Operations … NOT!
- Unifying instruction types
  - Coalescing instruction types into one "multi-function" pipe
  - Minimize external fragmentation (idle stages to match length)

Repetition of Independent Operations … NOT!
- Resolve data and resource hazards
  - Inter-instruction dependency detection and resolution

The Generic Instruction Pipeline



Balancing Pipeline Stages



T_{IF}= 6 units

T_{ID}= 2 units

T_{ID}= 9 units

T_{EX}= 5 units

T_{OS}= 9 units

Without pipelining

$T_{cyc} \approx T_{IF}+T_{ID}+T_{OF}+T_{EX}+T_{OS}$
$= 31$

Pipelined

$T_{cyc} \approx \max\{T_{IF},T_{ID},T_{OF},T_{EX},T_{OS}\}$
$= 9$

Speedup= 31 / 9

Balancing Pipeline Stages (1/2)

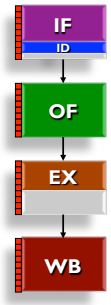
Two methods for stage quantization
- Merge multiple sub-ops into one
- Divide sub-ops into smaller pieces

Recent/Current trends
- Deeper pipelines (more and more stages)
- Multiple different pipelines/sub-pipelines
- Pipelining of memory accesses

Balancing Pipeline Stages (2/2)

Coarser-Grained Machine Cycle:
4 machine cyc / instruction

Finer-Grained Machine Cycle:
11 machine cyc /instruction



$T_{IF\&ID}$= 8 units

T_{OF}= 9 units

T_{EX}= 5 units

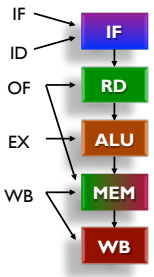
T_{OS}= 9 units

stages = 4
T_{cyc}= 9 units

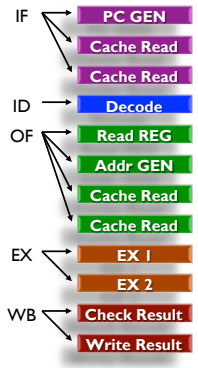
stages = 11
T_{cyc}= 3 units

Pipeline Examples

MIPS R2000/R3000



AMDAHL 470V/7



Instruction Dependencies

Data Dependence

- *Read-After-Write* (*RAW*) (only true dependence)
  - Read must wait until earlier write finishes
- *Anti-Dependence* (*WAR*)
  - Write must wait until earlier read finishes (avoid clobbering)
- *Output Dependence* (*WAW*)
  - Earlier write can't overwrite later write

Control Dependence (a.k.a. Procedural Dependence)

- Branch condition must execute before branch target
- Instructions after branch cannot run before branch

Hardware Dependency Analysis

Processor must handle

- Register Data Dependencies (same register)
  - RAW, WAW, WAR
- Memory Data Dependencies (same address)
  - RAW, WAW, WAR
- Control Dependencies

Pipeline Terminology

Pipeline Hazards

- Potential violations of program dependencies
- Must ensure program dependencies are not violated

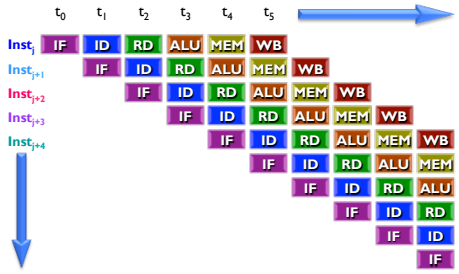
Hazard Resolution

- Static method: performed at compile time in software
- Dynamic method: performed at runtime using hardware
- Two options: Stall (costs perf.) or Forward (costs hw.)

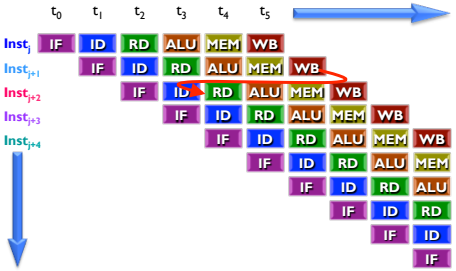
Pipeline Interlock

- Hardware mechanism for dynamic hazard resolution
- Must detect and enforce dependencies at runtime

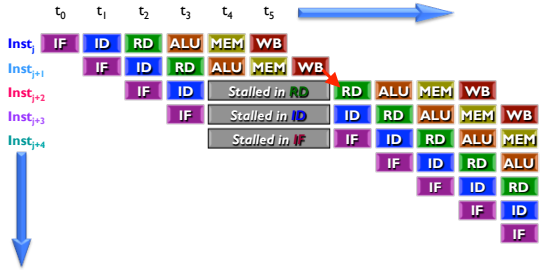
Pipeline: Steady State



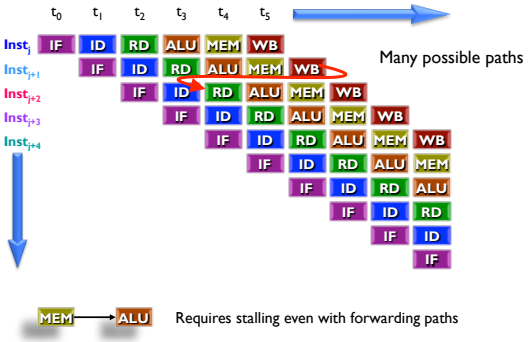
Pipeline: Data Hazard



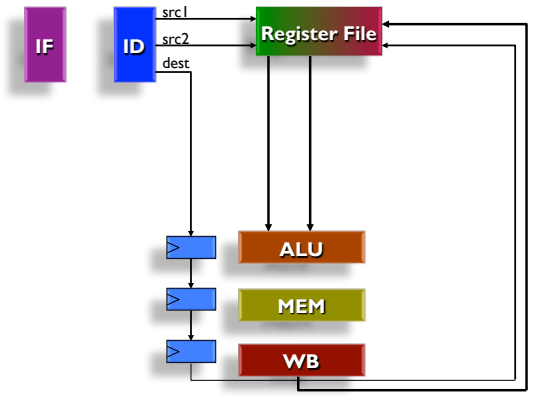
Option 1: Stall on Data Hazard



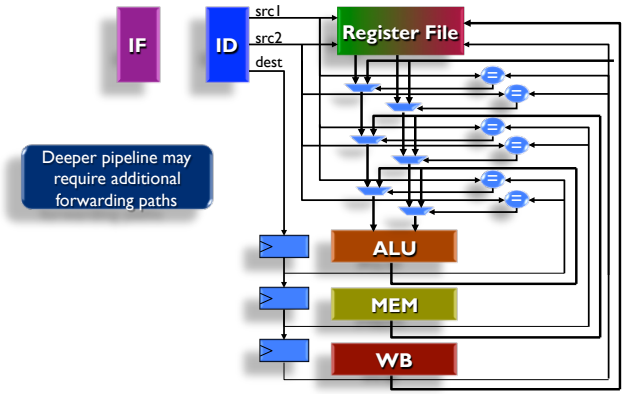
Option 2: Forwarding Paths (1/3)



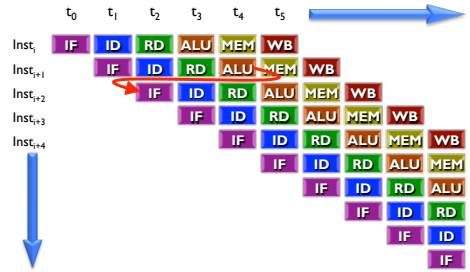
Option 2: Forwarding Paths (2/3)



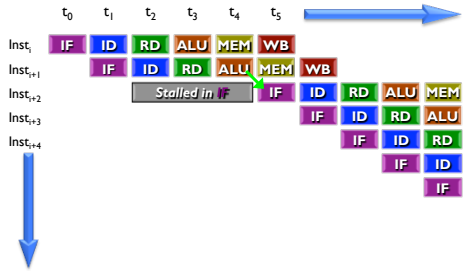
Option 2: Forwarding Paths (3/3)



Pipeline: Control Hazard



Pipeline: Stall on Control Hazard



Pipeline: Prediction for Control Hazards