

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568

Part 2

Pipelining - 1

Israel Koren
Fall 2011

ECE568/Koren Part.2.1

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Instruction Execution - Pipelines

- ◆ Execute billions of instructions, so *throughput* is what matters
- ◆ What is desirable in instruction sets for pipelining?
 - Variable length instructions vs. all instructions same length?
 - Memory operands part of any operation vs. memory operands only in loads or stores?
 - Register operand in various places in instruction format vs. registers located in same place?
- ◆ Conclusion: RISC is easier to pipeline

ECE568/Koren Part.2.2

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

"MIPS" - A "Typical" RISC

- ◆ 32-bit fixed length instruction (3 formats)
- ◆ Memory access only via load/store instructions
- ◆ 32 32-bit GPR (R0 contains zero)
- ◆ 32 32-bit FPR - 16 64-bit double-precision
 - DP uses a pair
- ◆ 3-address, reg-reg arithmetic instruction; registers in same place in instruction format
- ◆ Single address mode for load/store: base + displacement
- ◆ Simple branch conditions; addressing modes: PC relative and register indirect
- ◆ Delayed branch

see: some versions of SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, DSP processors
pp. 129-136, A.26-A.37 in textbook

ECE568/Koren Part.2.3

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Data Formats and Memory Addresses

Data formats:

Bytes, Half words, words and double words

- Byte addressing

Big Endian
vs. Little Endian

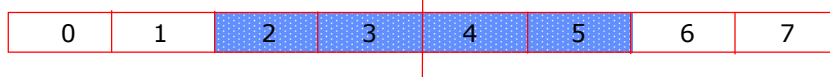
Most Significant Byte Least Significant Byte

0	1	2	3
3	2	1	0

Byte Addresses

- Word alignment

Byte addressable memory
A word address can begin only at 0, 4, 8,



ECE568/Koren Part.2.4

Adapted from UCB and other sources

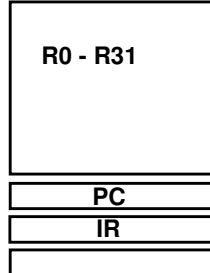
Copyright UCB & Morgan Kaufmann

MIPS Instruction Set Architecture

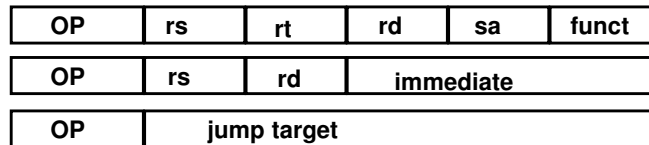
◆ Instruction Categories

- Load/Store
- Computational (Fixed-point etc)
- Floating-Point
- Jump and Branch
- Special

Registers



3 Instruction Formats: all 32 bits wide

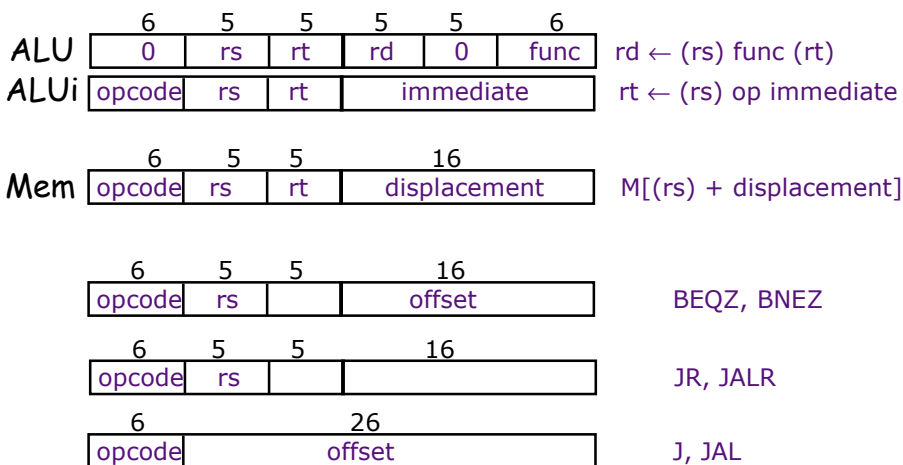


ECE568/Koren Part.2.5

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

MIPS Instruction Formats



ECE568/Koren Part.2.6

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Instruction Execution

Execution of a MIPS instruction involves

1. instruction fetch
2. decode and register fetch
3. ALU operation
4. memory operation (optional)
5. write back to register file (optional)

Control Signals (Microinstructions)

instr fetch: $MA \leftarrow PC$
 $A \leftarrow PC$
 $IR \leftarrow \text{Memory}$
 $PC \leftarrow A + 4$

ALU: $A \leftarrow \text{Reg}[rs]$
 $B \leftarrow \text{Reg}[rt]$
 $\text{Reg}[rd] \leftarrow \text{func}(A,B)$

ALUi: $A \leftarrow \text{Reg}[rs]$
 $B \leftarrow \text{Imm}$ *sign extension ...*
 $\text{Reg}[rt] \leftarrow \text{Opcode}(A,B)$

Control Signals (Microinstructions) - cont'd

LW: $A \leftarrow \text{Reg}[\text{rs}]$
 $B \leftarrow \text{Imm}$
 $\text{MA} \leftarrow A + B$
 $\text{Reg}[\text{rt}] \leftarrow \text{Memory}$

J: $A \leftarrow \text{PC}$
 $B \leftarrow \text{IR}$
 $\text{PC} \leftarrow \text{JumpTarg}(A,B)$

$\text{JumpTarg}(A,B) = \{A[31:28], B[25:0], 00\}$

beqz: $A \leftarrow \text{Reg}[\text{rs}]$
 If zero?(A) then go to bz-taken
 instruction fetch

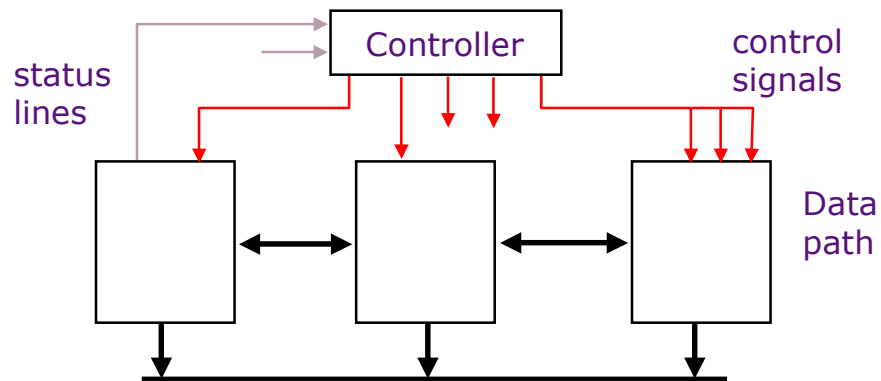
bz-taken: $A \leftarrow \text{PC}$
 $B \leftarrow \text{Imm} \ll 2$
 $\text{PC} \leftarrow A + B$

ECE568/Koren Part.2.9

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Microarchitecture: *Implementation of an ISA*

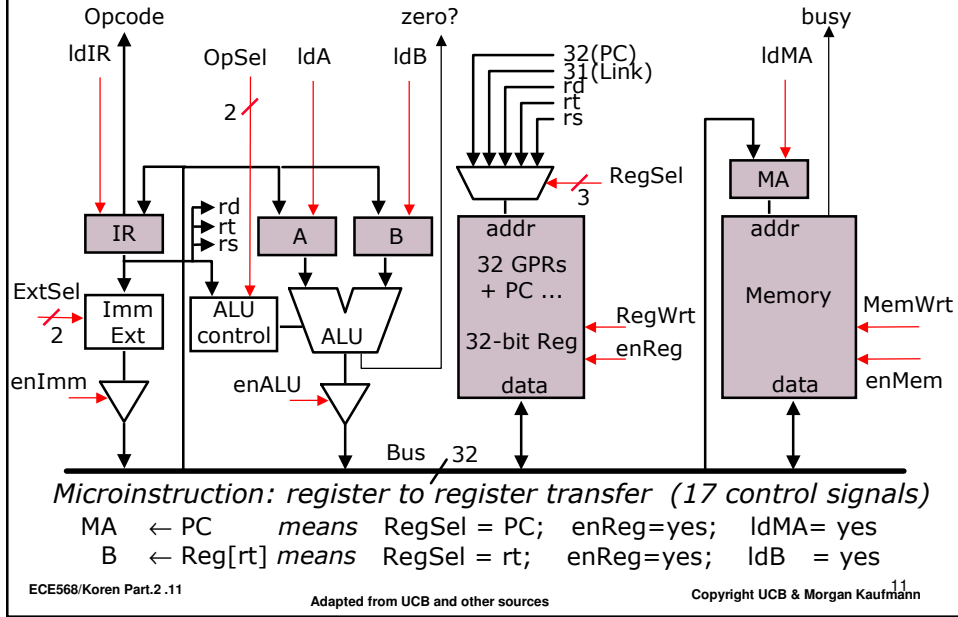


ECE568/Koren Part.2.10

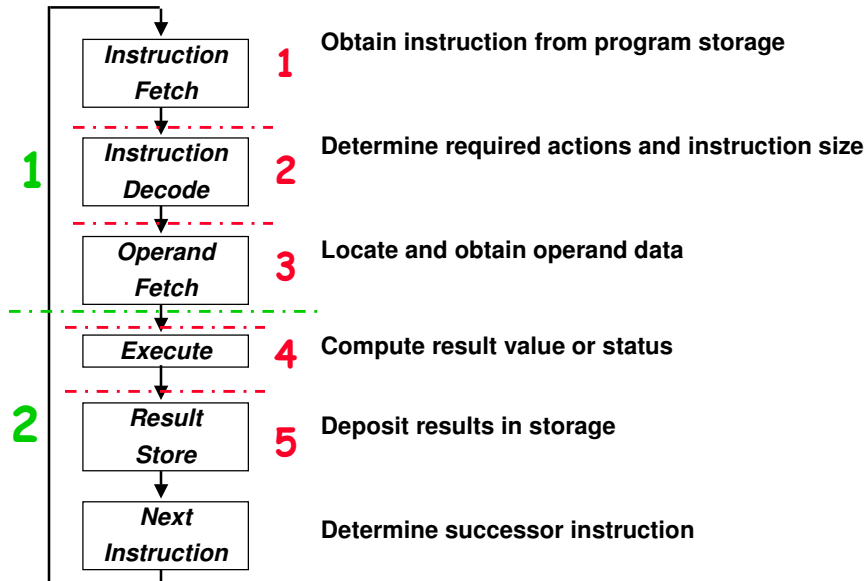
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

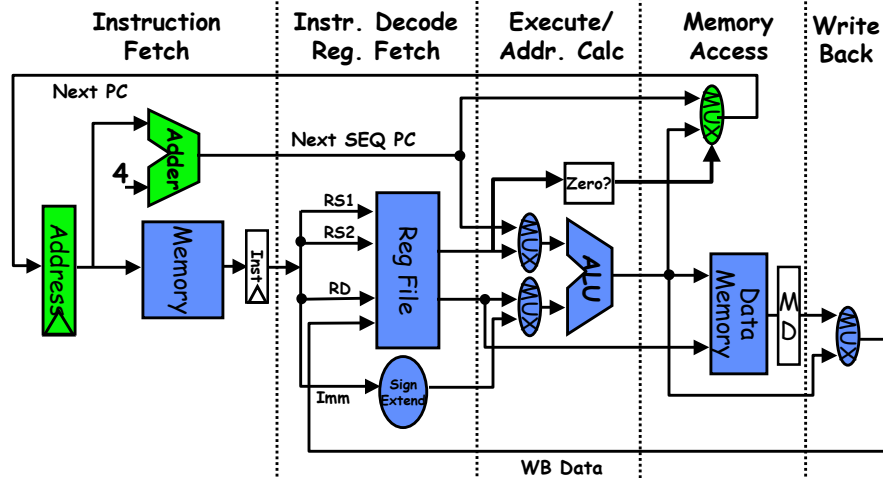
A Bus-based Datapath for MIPS



Execution Cycle - pipeline stages



5 Steps of MIPS Datapath w/o pipelining

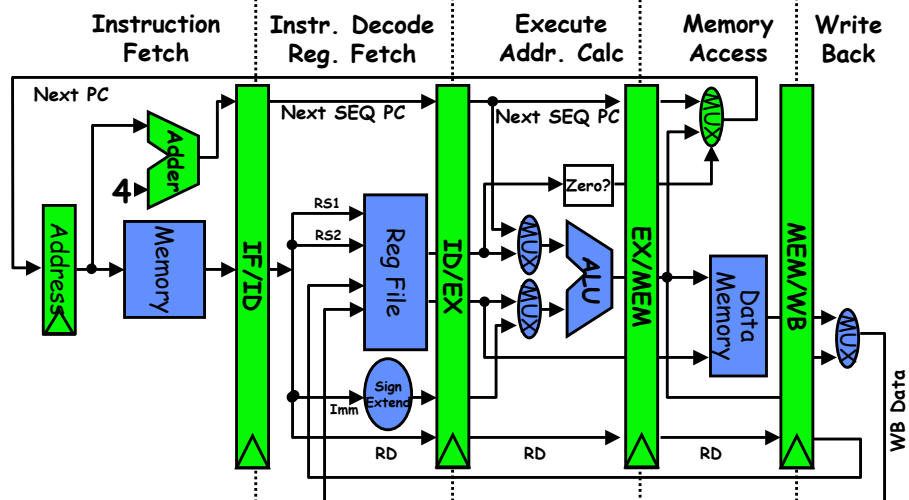


ECE568/Koren Part.2.13

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

5 Steps of MIPS Datapath w/pipelining



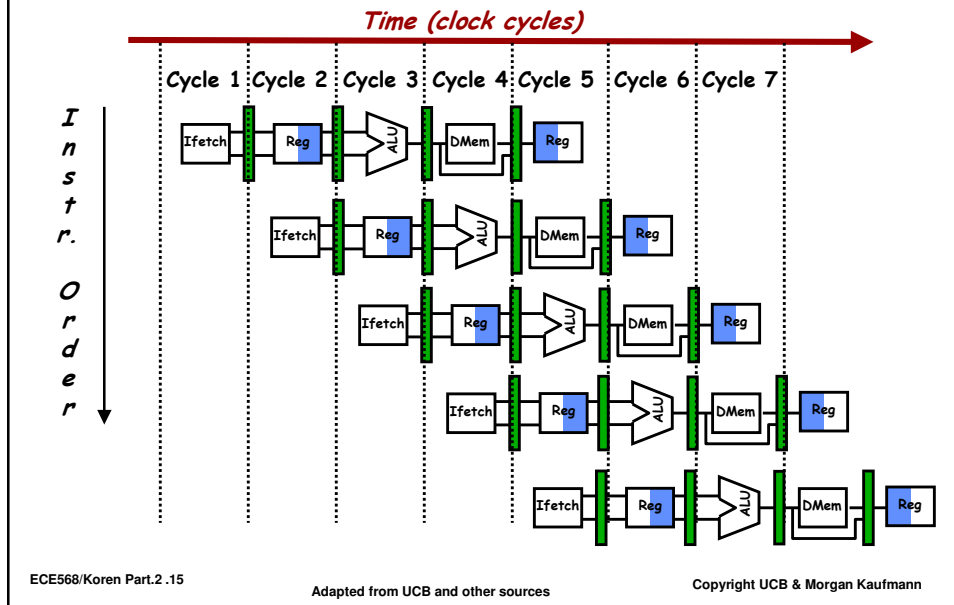
- Local decoded instruction fields for each phase / pipeline stage

ECE568/Koren Part.2.14

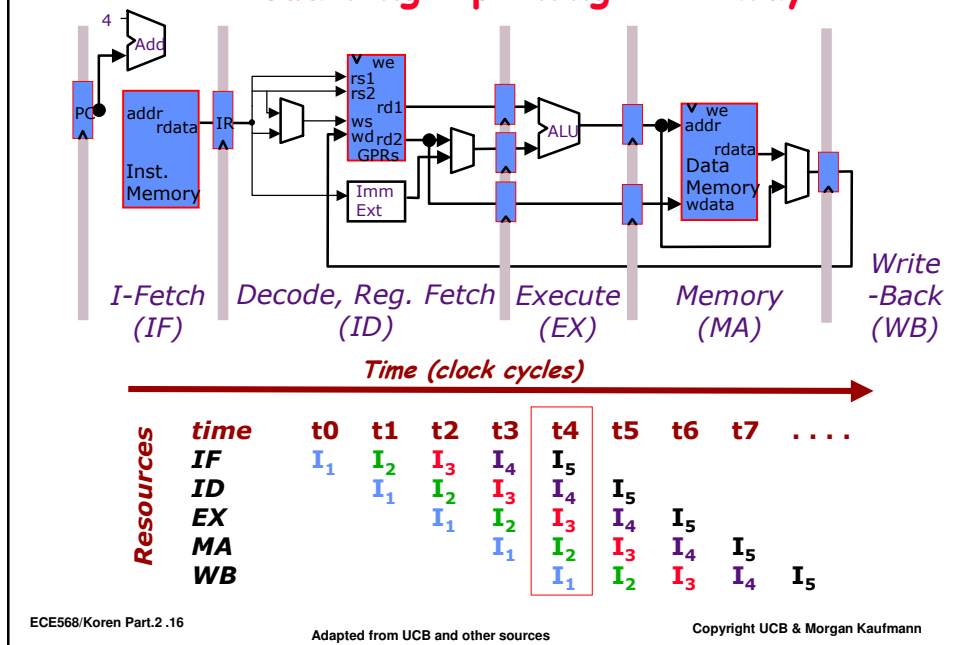
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

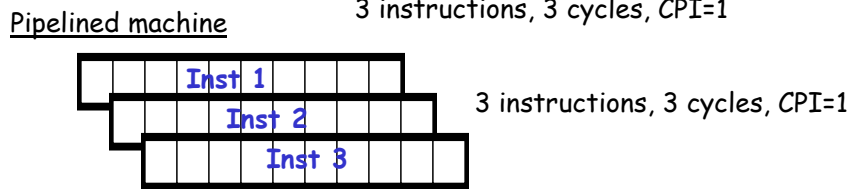
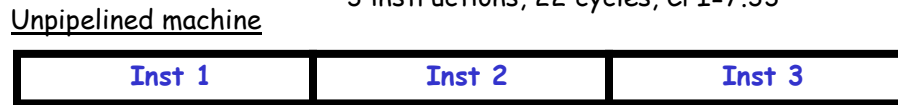
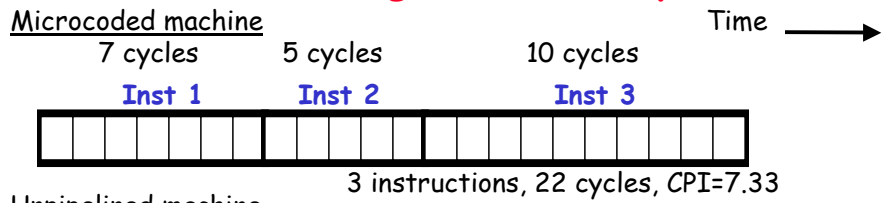
Visualizing Pipelining



Visualizing Pipelining - 2nd way



Calculating CPI - Example



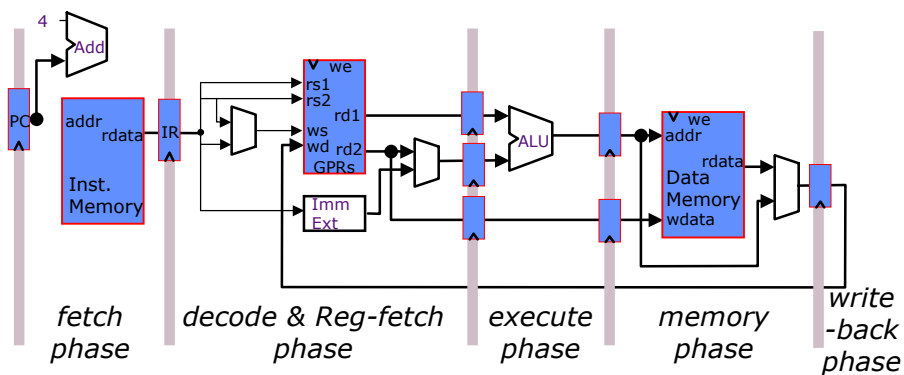
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

ECE568/Koren Part.2 .17

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Pipelined Datapath



$$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\} \quad (= t_{DM} \text{ probably})$$

ECE568/Koren Part.2 .18

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Technology Assumptions

- A small amount of very fast memory (caches) backed up by a large, slower memory
- Fast ALU (at least for integers)
- Multiported Register files (slower!)

Thus, the following timing assumption is reasonable

$$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$$

MIPS pipeline

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC, PC ← (if ((EX/MEM.opcode == branch) & EX/MEM.cond){EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR[rs]]; ID/EX.B ← Regs[IF/ID.IR[rt]]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← sign-extend(IF/ID.IR[immediate field]);		
	ALU instruction	Load or store instruction	Branch instruction
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A <i>func</i> ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A <i>op</i> ID/EX.Imm;	EX/MEM.IR ← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.B ← ID/EX.B;	EX/MEM.ALUOutput ← ID/EX.NPC + (ID/EX.Imm << 2); EX/MEM.cond ← (ID/EX.A == 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR[rd]] ← MEM/WB.ALUOutput; or Regs[MEM/WB.IR[rt]] ← MEM/WB.ALUOutput;	For load only: Regs[MEM/WB.IR[rt]] ← MEM/WB.LMD;	

Instruction pipeline speedup

The pipeline "forces" all instructions to go through all five stages

P4 = % of instructions requiring 4 cycles (e.g., ALU)

P3 = % of instructions requiring 3 cycles (e.g., Branch)

P5 = % of instructions requiring 5 cycles (e.g., Load) = ?

$$CPI_{\text{unpipelined}} = P4 * 4 + P3 * 3 + P5 * 5$$

e.g., $CPI_{\text{unpipelined}} = .5 * 4 + .2 * 3 + .3 * 5 = 4.1$

$$CPI_{\text{pipelined}} = 1 \text{ (ideally)}$$

$$ExTime = (\# \text{ of instr.}) * CPI * T$$

$$Speedup = \frac{CPI_{\text{unpipelined}}}{CPI_{\text{pipelined}}} \times \frac{T_{\text{unpipelined}}}{T_{\text{pipelined}}} < 4.1 < 5 \text{ (ideal speedup)}$$

ECE568/Koren Part.2 .21

Copyright 2011 Koren UMass

Instruction pipelines are not ideal

- ◆ Instructions interact with each other in pipeline
- ◆ **Hazards** prevent next instruction from executing during its designated clock cycle
 - **Structural hazards**: An instruction in the pipeline may need a resource being used by a previous instruction in the pipeline (e.g., operand address calculation using the same fixed-point adder used for addition)
 - **Data hazards**: Instruction depends on (data) result of prior instruction still in the pipeline:

$$A \leftarrow B + C$$

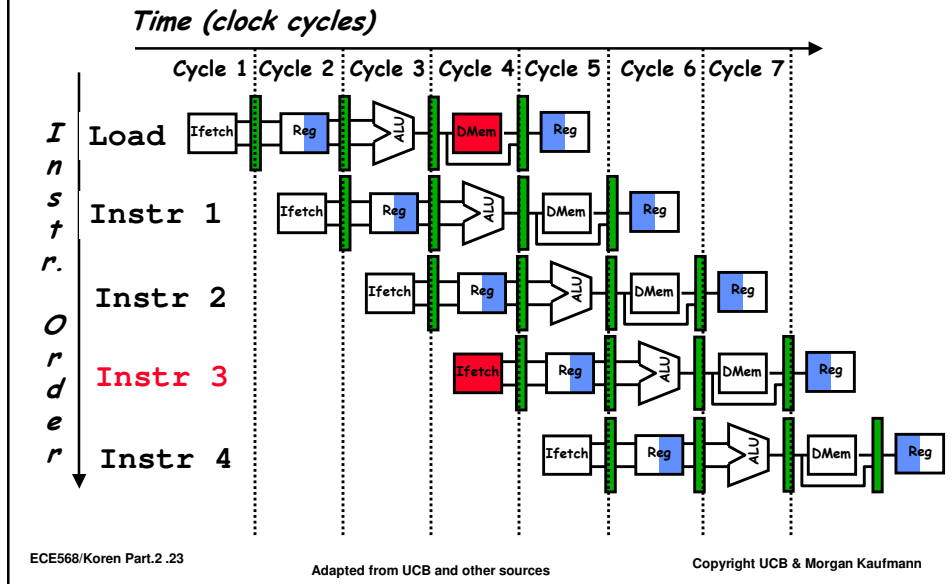
$$D \leftarrow A * B$$
 - **Control hazards**: Branches and jumps
 - **Interrupts/exceptions**
- ◆ **Issues**:
 - How to detect?
 - How to minimize the penalty?

ECE568/Koren Part.2 .22

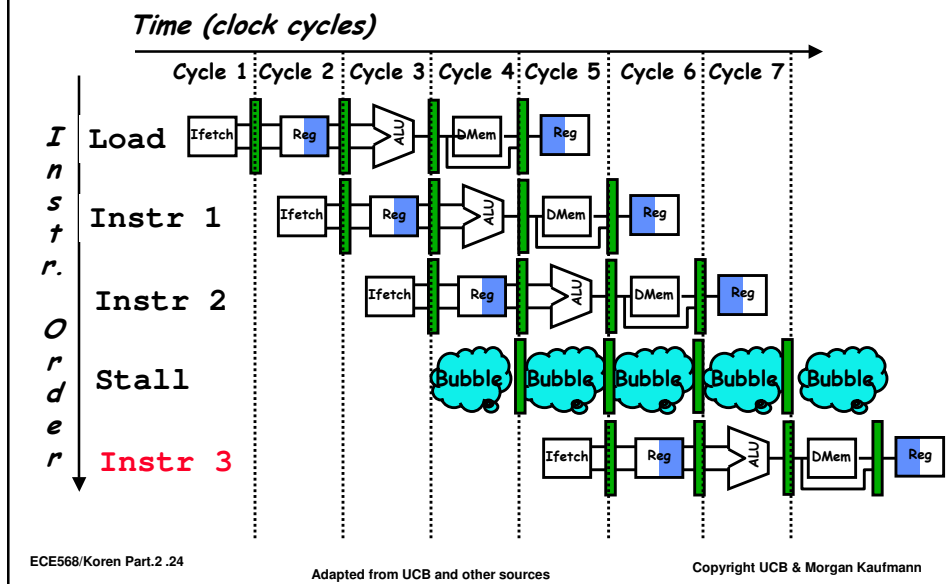
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Structural Hazards - one Memory Port



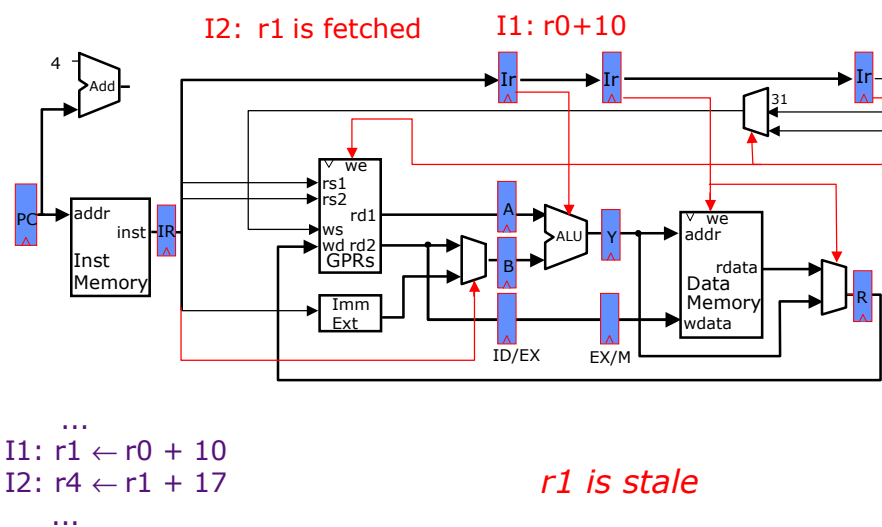
One Memory Port/Structural Hazards



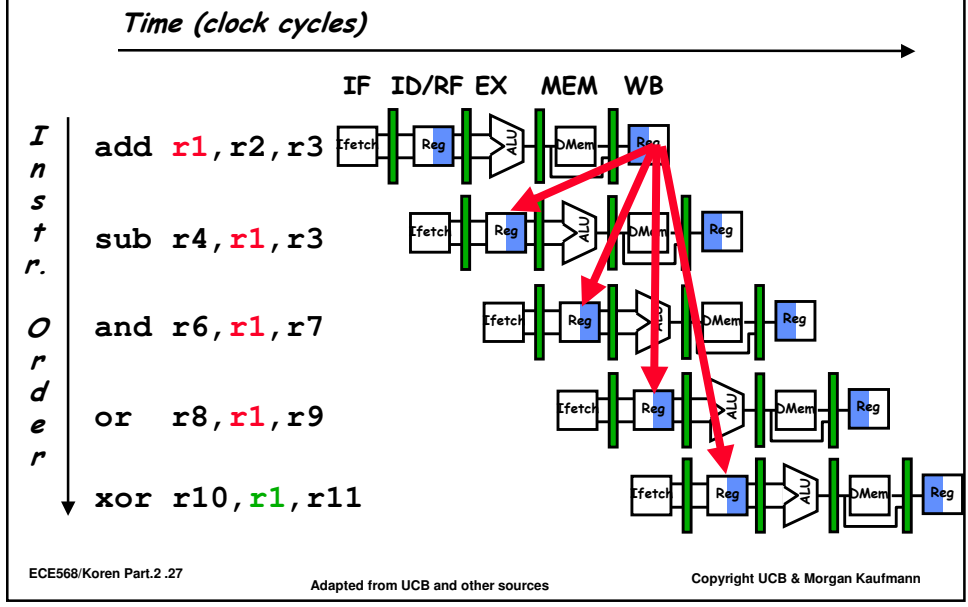
Resolving Structural Hazards

- ◆ Structural hazards occurs when two instruction need same hardware resource at same time
 - Can resolve in hardware by stalling newer instruction till older instruction finished with resource
- ◆ A structural hazard can always be avoided by adding more hardware to design
 - E.g., if two instructions both need a port to memory at same time, could avoid hazard by adding second port to memory
- ◆ Our 5-stage pipe has no structural hazards by design
 - Thanks to MIPS ISA, which was designed for pipelining

Data Hazards



Data Hazard on R1



Pipeline control design (avoid structural hazards)

"Effective control for pipelined computers,"
by Davidson *et al.* (available on the web page)

Reservation Table: Task A t0 t1 t2 t3 t4 t5

Unit s0 is the busiest

3 out of 6 time units

throughput ≤ 2 instr./ 6 cycles

$1/6 \leq$ throughput $\leq 1/3$

How can we find the real throughput?

Define: **Pipeline Collision Vector (PCV)**

PCV = C0 C1 ... C{k-1}; k = # of cycles (t0, ..., t{k-1})

Ci=1 if a 2nd task initiated i cycles after the 1st will result in a collision; Ci=0 otherwise.

s0	X		X		X	
s1		X				
s2		X				X
s3			X	X		
s4			X			X

PCV for Reservation Table A

t=1

	t0	t1	t2	t3	t4	t5
s0	X	X	X	X	X	X
s1		X	X			
s2		X	X		X	X
s3			X	X	X	
s4			X	X	X	X

0 1 2 3 4 5
Initial Value PCV=(1 1 1 1 1 0)

Throughput =

t=2

	t0	t1	t2	t3	t4	t5
s0	X		X	X	X	
s1		X	X			
s2		X	X	X	X	X
s3			X	X	X	
s4			X	X	X	X

t=3

	t0	t1	t2	t3	t4	t5
s0	X		X	X	X	X
s1		X		X		
s2		X		X	X	X
s3			X	X	X	
s4			X	X	X	X

ECE568/Koren Part.2 .29

Copyright 2010 Koren UMass

Forbidden Latencies

A

	t0	t1	t2	t3	t4	t5
s0	X		X		X	
s1		X				
s2		X			X	
s3			X	X		
s4			X		X	

s0: 2,4

Overall=0,1,2,3,4

s1: 0 only

s2: 4

s3: 1

s4: 3

0 1 2 3 4 5
PCV=(1 1 1 1 1 0)

B

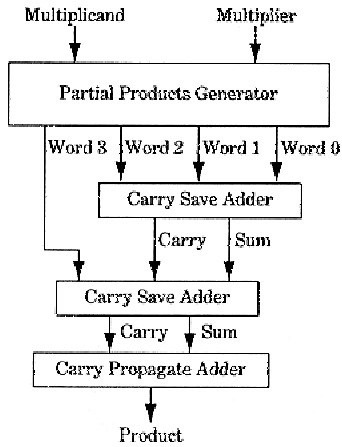
	t0	t1	t2	t3	t4	t5
s0	X	X				
s1	X		X			
s2	X			X		
s3	X				X	
s4	X					X

0 1 2 3 4 5
PCV(Table B)=()

ECE568/Koren Part.2 .30

Copyright 2010 Koren UMass

Reservation Table for a multiplier



	1	2	3	4	5
PPG	X				
CSA1		X			
CSA2			X		
CPA				X	X

Source: "Memory Systems & Pipelined Processor," H. Cragon, Jones & Bartlett, 1996.

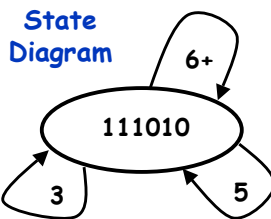
ECE568/Koren Part.2 .31

Copyright 2010 Koren UMass

PCV for Reservation Table C

C

	t0	t1	t2	t3	t4	t5
s0	X		X		X	
s1		X				
s2			X	X		
s3		X				X



Forbidden latencies: 0,1,2,4

$$PCV(\text{Table } C) = (1 \ 1 \ 1 \ 0 \ 1 \ 0)$$

= state of pipeline at t0

At t1: 1 1 0 1 0 0

At t2: 1 0 1 0 0 0

At t3: 0 1 0 0 0 0

If a new task is started at t3:

0 1 0 0 0 0

1 1 1 0 1 0

1 1 1 0 1 0

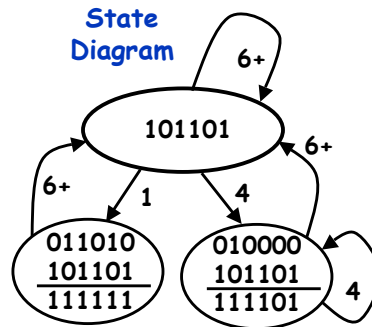
ECE568/Koren Part.2 .32

Copyright 2010 Koren UMass

Reservation Table D

	t0	t1	t2	t3	t4	t5
s0	X		X			X
s1		X				
s2			X	X		
s3					X	

Forbidden Latencies=?
PCV=?



Possible execution cycles: (4),(1,6),(4,6)

If (1,6) is followed: tasks start at 0,1,7,8,14,...

Avg.Delay=Avg.(1/throughput)=(1+6)/2=3.5

If (4) is followed: tasks start at 0,4,8,12,...

(1,6) - greedy cycle

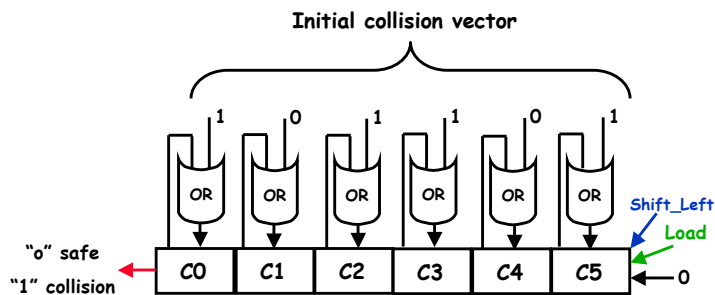
Avg.Delay=4

Minimum Average Latency (MAL)=3.5; $\leq MAL \leq$

ECE568/Koren Part.2 .33

Copyright 2010 Koren UMass

Greedy control



ECE568/Koren Part.2 .34

Copyright 2010 Koren UMass

Two tasks A & B

	t0	t1	t2	t3	t4
s0	A			A	
s1		A			
s2			A		A

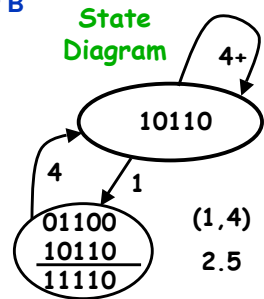
$$M_A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{array}{l} \text{A after A} \\ \text{B After A} \end{array}$$

Forbidden latencies
B after A:
(from B to A)
0, 2, 4

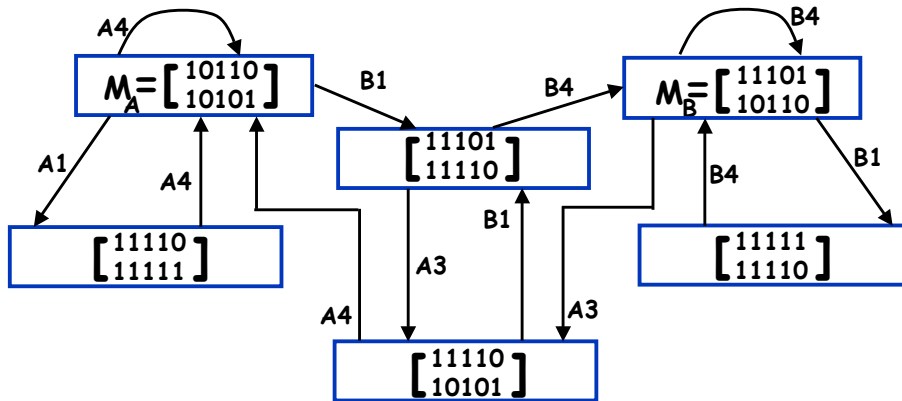
	t0	t1	t2	t3	t4
s0		B			B
s1				B	
s2	B		B		

$$M_B = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{array}{l} \text{A after B} \\ \text{B After B} \end{array}$$

Forbidden latencies
A after B:
(from A to B)
0, 1, 2, 4



Possible Greedy Cycles



- AAA... (A1, A4) Avg.Latency=2.5
- BBB... (B1, B4) Avg.Latency=2.5
- ABAB... (B1, A3) Avg.Latency=2
- BABA... (A3, B1) Avg.Latency=2
- ABBABB.. (B1, B4, A3) Avg.Latency=2.67

Inserting delays to improve throughput

	t0	t1	t2	t3
s0	X	X		
s1	X		X	
s2	X			X

PCV=1111 → throughput=1/4 but MAL=4>2

After delay insertion
PCV=100100
greedy cycle=(1,1,4)
throughput=?

Is the solution unique?

	t0	t1	t2	t3	t4	t5
s0	X			X		
s1	D	X			X	
s2	D	D	X			X

Is it always advisable to insert delays?

# Ops.	RT	M-RT	Pi
1	4 <	6	0.5
2	8 >	7	0.4
3	12 >	8	0.05
4	16 >	12	0.05

Pi=Prob { i consecutive identical instructions}

RT-Avg.Latency=

M-RT-Avg.Latency=