

EEC 485 Computer Organization (Fall 2006)

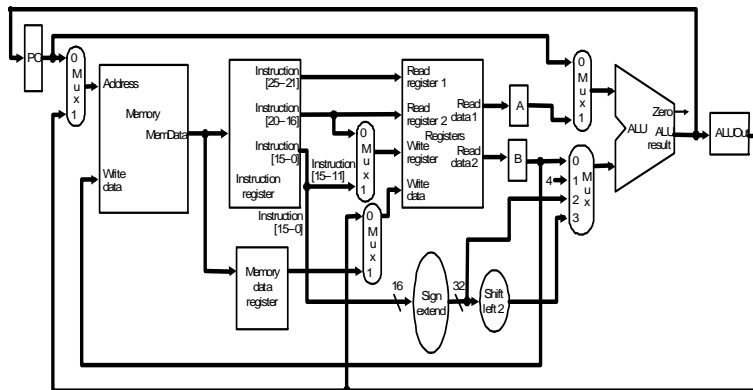
Summary of Pipelined MIPS

Chansu Yu

Cleveland State University

Multicycle MIPS => Pipelined MIPS

5 stages (IF, ID, EX, MEM, WB) are conflict-free with each other (no structural hazards). Is it true?

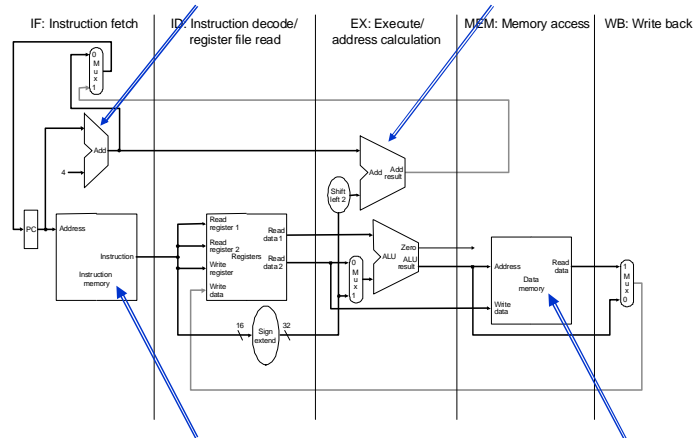


Resource Conflicts

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch		IR = Memory[PC] PC = PC + 4		
Instruction decode/register fetch		A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (sign-extend(IR[15-0] << 2))		
Execution, address computation, branch/jump completion	ALUOut = A op B	ALUOut = A + sign-extend(IR[15-0])	if (A == B) then PC = ALUOut	PC = PC[31-28] (IR[25-0] << 2)
Memory access or R-type completion	Reg[IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory[ALUOut] = B		Memory conflict
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Register file conflict (read or write)

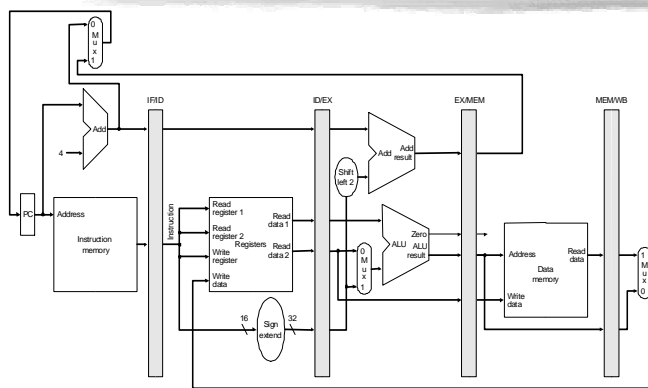
Basic Pipeline



Basic Pipeline

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg[IR[25-21]] B = Reg[IR[20-16]] $ALUOut = PC + (\text{sign-extend}(\text{IR}[15-0]) \ll 2)$			
Execution, address computation, branch/jump completion	ALUOut = A op B	ALUOut = A + sign-extend(IR[15-0])	if (A == B) then PC = ALUOut	PC = PC[31-28] (IR[25-0] << 2)
Memory access or R-type completion	Reg[IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory[ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Pipelined Datapath



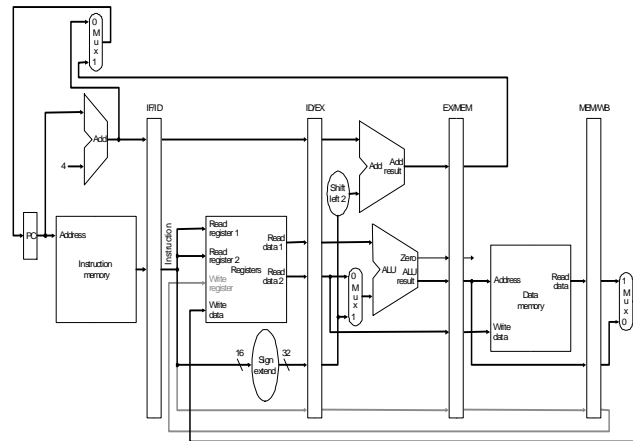
Add to the basic pipeline in order to actually split the datapath into stages.

The info. must be placed in a pipeline register; otherwise, it is lost when the next instruction enters that pipeline stage.

For store instruction,

(?) => ID/EX pipeline register => EX/MEM pipeline register => (?)

Corrected Datapath



Cleveland State
University

7

c.yu91@csuohio.edu

Control Signals of Pipelined MIPS

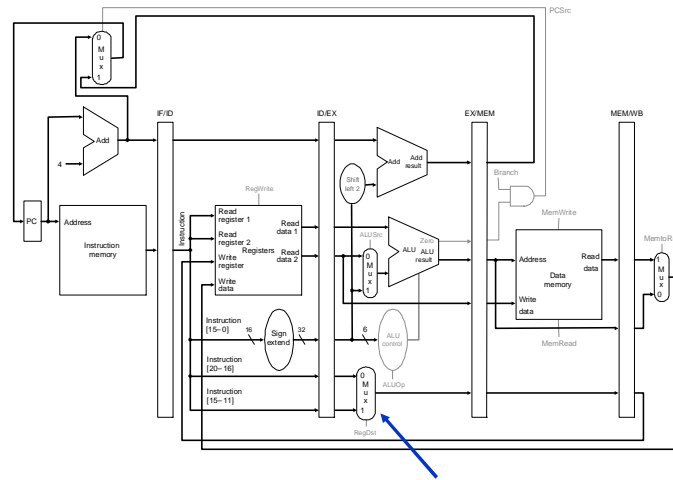
- At IF & ID stages ???
- At EX stage
 - RegDst: 0 for lw, 1 for R-type
 - ALUSrc: 0 for R-type/beq (Reg.), 1 for lw/sw (inst.)
 - ALUOp1/0: 10 for R-type, 00 for lw/sw (add), 01 for beq (sub)
- At MEM stage
 - Branch: Activated if the instruction is “branch”
 - MemWrite / MemRead: 1 for sw and lw, respectively
 - PCSrc: PCSrc = Branch && Zero flag (*derived, not generated*)
- At WB stage
 - RegWrite: Activated if the instruction is R-type or lw
 - MemtoReg: 0 for R-type (ALU), 1 for lw (memory)

Cleveland State
University

8

c.yu91@csuohio.edu

Pipeline Control



Single-cycle => Pipelined MIPS

4 multiplexor selectors
2 write signals
2 ALU controls

	Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Mem Branch (PCSrc)	ALUOp1	ALUOp0
000000	R-format	1	0	0	1	0	0	0	1	0
100011	lw	0	1	1	1	1	0	0	0	0
101011	sw	X	1	X	0	0	1	0	0	0
000100	beq	X	0	X	0	0	0	1	0	1

9 control signals in single-cycle MIPS
 => Can be used in pipelined MIPS
 => But, ...

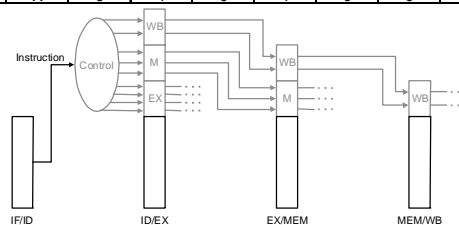
Pipeline Control

- ❑ What's different in pipelined MIPS?
 - add IF ID EX MEM WB
 - lw IF ID EX MEM WB
- ❑ Control signals should also be conflict-free
- ❑ Should we use a finite state machine?
- ❑ Our approach is
 - Generate control signals all at once at ID stage
 - And passed them through stages

Pipeline Control

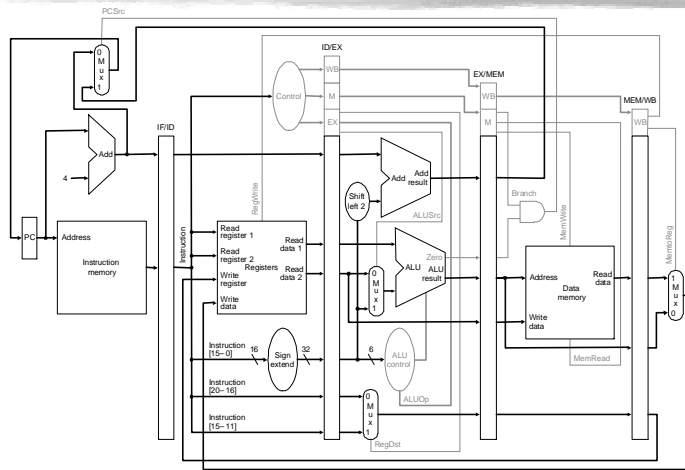
- ❑ Pass control signals along just like the data

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

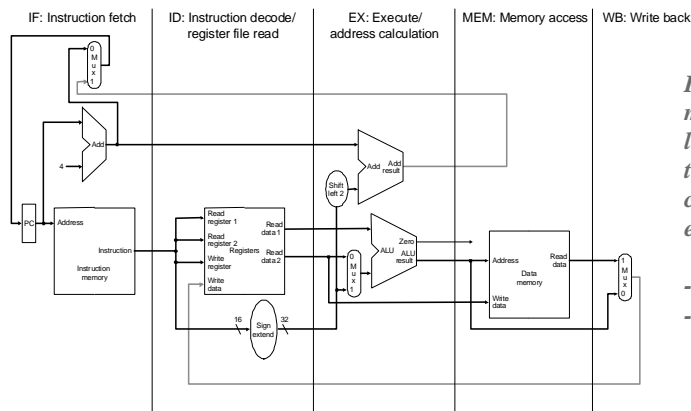


PCSrc is derived at MEM stage by (Branch & Zero flag)

Datapath with Control



Data and Control Hazards

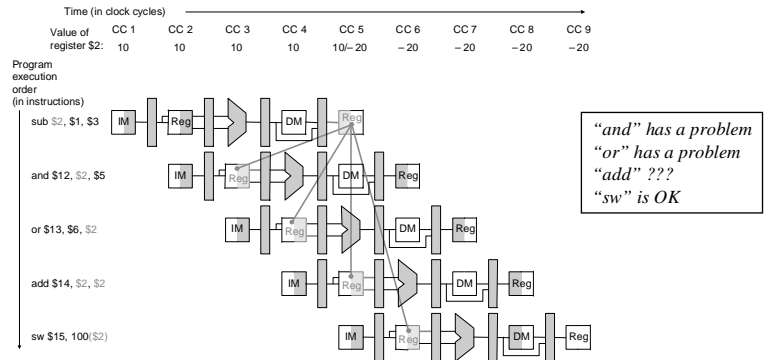


Instructions and data move generally from left to right through the five stages as they complete execution except two cases.

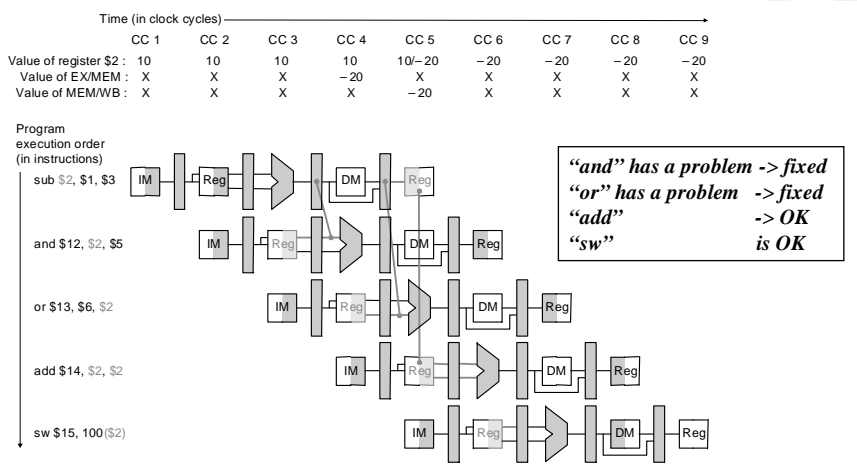
- WB stage
- PC selection

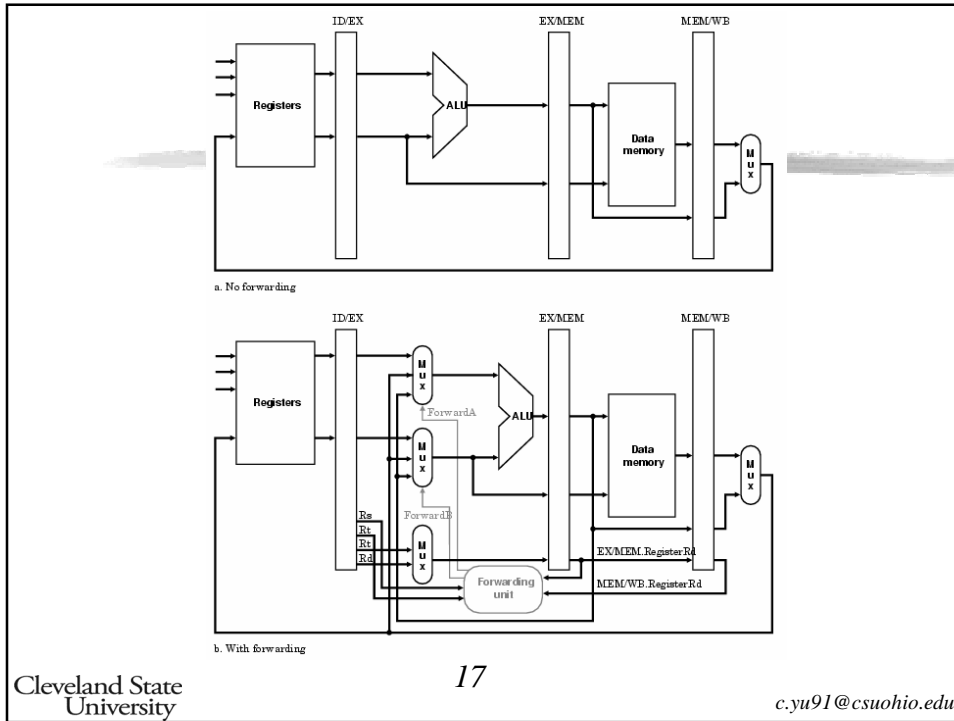
Data Hazards: Dependencies

- Problem with starting next instruction before first is finished
 - dependencies that “go backward in time” are data hazards



Forwarding : All 2 Cases





Forwarding Control

Control logic

ForwardA =

- 10 if (EX/MEM.Rd = ID/EX.Rs) <- get operand from EX/MEM
- 01 if (MEM/WB.Rd = ID/EX.Rs) <- get operand from MEM/WB
- 00, otherwise <- get operand from ID/EX

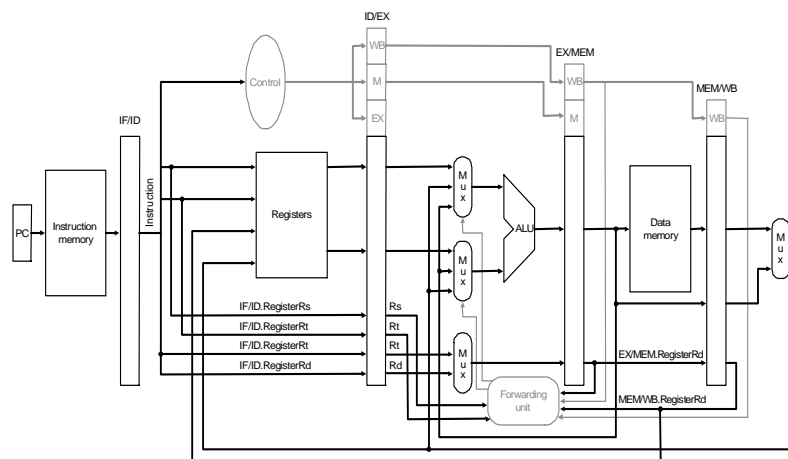
ForwardB =

- 10 if (EX/MEM.Rd = ID/EX.Rt) <- get operand from EX/MEM
- 01 if (MEM/WB.Rd = ID/EX.Rt) <- get operand from MEM/WB
- 00, otherwise <- get operand from ID/EX

Forwarding Control

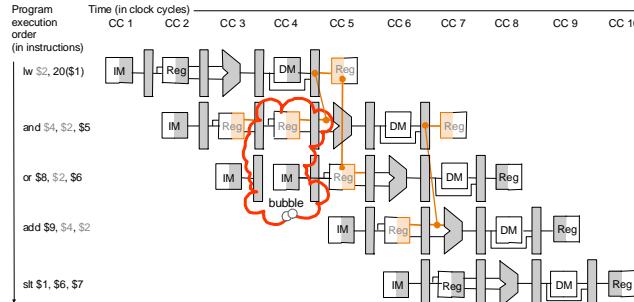
- Complexities
 - RegWrite must be active & destination register must not be \$0
 - ForwardA =
 - 10 if (EX/MEM.Rd = ID/EX.Rs) <- get operand from EX/MEM
 - 01 if (MEM/WB.Rd = ID/EX.Rs) <- get operand from MEM/WB
 - 00, otherwise <- get operand from ID/EX
 - What if (EX/MEM.Rd = ID/EX.Rs) and at the same time (MEM/WB.Rd = ID/EX.Rs) ?
 - When both conditions are satisfied, the most recent result should be forwarded. Example?
- Control logic
 - ForwardA =
 - 10 if ((EX/MEM.Rd = ID/EX.Rs) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rs) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rs))
 - 00, otherwise
 - ForwardB =
 - 10 if ((EX/MEM.Rd = ID/EX.Rt) && EX/MEM.RegWrite && (EX/MEM.Rd ≠ 0))
 - 01 if ((MEM/WB.Rd = ID/EX.Rt) && MEM/WB.RegWrite && (MEM/WB.Rd ≠ 0) && (EX/MEM.Rd ≠ ID/EX.Rt))
 - 00, otherwise

Forwarding : Forwarding Unit



Data Hazards: Stalling

- Stall the pipeline by keeping an instruction in the same stage



*lw-and
lw-or
At CC5, MEM stage is empty !!!*

Data Hazards: Stalling

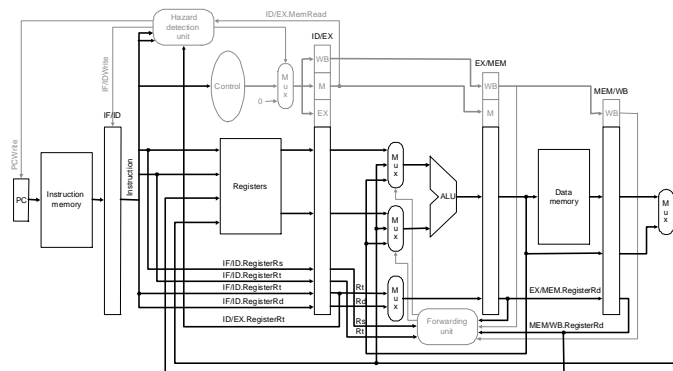
- Stalling detection and control
 - Detects during the ID stage when “lw” instruction is in EX stage
 - The following two instructions are in ID (“and”) and IF (“or”) stages, respectively
 - If detected,
 - Stall the following instruction (in ID stage, “and”) so that it repeats the ID stage again => IF/ID pipeline register should not be changed
 - Stall the second instruction (in IF stage, “or”) so that it repeats the IF stage again => PC should not be changed

Data Hazards: Stalling

- ❑ Hazard detection
 - If (ID/EX.MemRead and ((ID/EX.Rt = IF/ID.Rs) or (ID/EX.Rt = IF/ID.Rt))) stall the pipeline
- ❑ Control signals generated from hazard detection unit
 - IF/IDWrite to prevent IF/ID register from changing
 - PCWrite to prevent PC from changing
 - MUX control to pass “null” control signals

Stalling: Detection Unit

- ❑ Stall by letting an instruction that won't write anything go forward



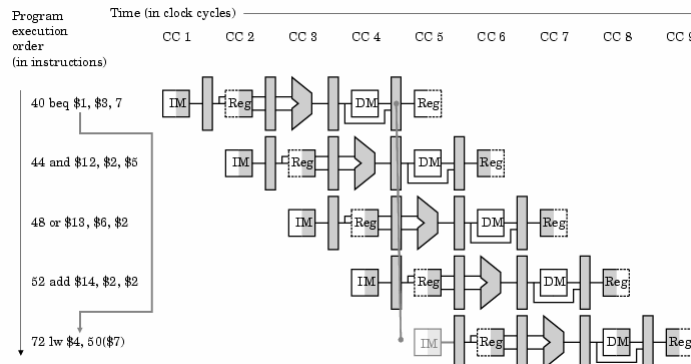
Data Hazards: Stalling

- ❑ Data forwarding for stalled instructions
 - They are needed; otherwise, we need one more stall cycle
 - “and” – forwarding to EX stage from WB stage (MEM/WB pipeline register)
 - “or” – no forwarding is required

- ❑ Hazard detection
 - If (ID/EX.MemRead and ((ID/EX.Rt = IF/ID.Rs) or (ID/EX.Rt = IF/ID.Rt)) stall the pipeline
 - Consider the following two examples

lw \$1, 100(\$2)	lw \$1, 100(\$2)
lw \$1, 200(\$4)	sw \$1, 200(\$3)

Branch (Control) Hazards



Branch Hazards

- ❑ Reducing the cost of taken branch
 - Branch penalty is 3 cycles because
 - IF: PC+4
 - EX: Branch address calculation, ZF evaluation
 - MEM: Branch target is finally selected
 - Reducing the penalty to 1 from 3 cycles by selecting branch address at the ID stage
 - Branch address calculation can be done at ID stage
 - ZF evaluation: Equality can be tested at ID stage by first exclusive ORing respective bits of two read registers and then ANDing all the results

Branch Hazards

- ❑ We can stall the pipeline for every branch instruction
 - Too slow
- ❑ Or, continue execution down the sequential instruction stream assuming that the branch will not be taken (predict “branch not taken”)
 - If the condition is not met, OK ! (prediction is successful)
 - If the condition is met, (prediction is wrong)
 - An unwanted instruction is in the pipeline (IF stage)!
 - Need to “flush” that instruction
- ❑ How to flush?
 - IF.Flush to flush the instruction in IF stage
 - It zeros the instruction field of the IF/ID pipeline register
 - IF.Flush = (IF/ID.Branch && ZF) ?? ← is this same as PCSrc???

Branch Hazards : Flushing

