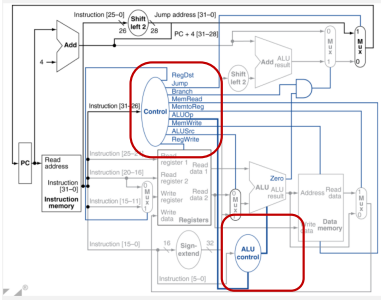
VE370 Review

Single Cycle Processor

Data path



Instruction class	Functional units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

The critical path is load instruction:
IM+Reg+Mux+ALU+DM+Mux+Reg

Longest delay determines the clock period.

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

opcode	Operation	ALUOp	funct	ALU Control	ALU function
lw	load word	00	XXXXXX	0010	add
sw	store word				
beq	branch equal	01	XXXXXX	0110	subtract
R-type	add	10	100000	0010	add
	subtract		100010	0110	subtract
	AND		100100	0000	AND
	OR		100101	0001	OR
	set-on-less-than		101010	0111	set-on-less-than

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Exercise 4.11

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

	Instruction word
a.	10101100011000100000000000010100
b.	00000000100000100000100000101010

4.11.2 [10] <4.4> What are the values of the ALU control unit's inputs for this instruction?

4.11.3 [10] <4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

(a).sw

ALUOp=00

Funct=xxxxxx

Operation =0010 (add)

(b) slt

ALUOp=10

Funct=101010

Operation =0111(slt)

How To Support New Instruction

- Create the new functional blocks, new logic gates or modify the original blocks
  - New Mux, AND gate, OR gate
  - Transform 2 to 1 Mux into 3 to 1 Mux
- Add new control signals or modify the original signals

Exercise 4.2

The basic single-cycle MIPS implementation in Figure 4.2 can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to do that depends, among other things, on the cost and complexity such an addition introduces into the processor datapath and control. The first three problems in this exercise refer to this new instruction:

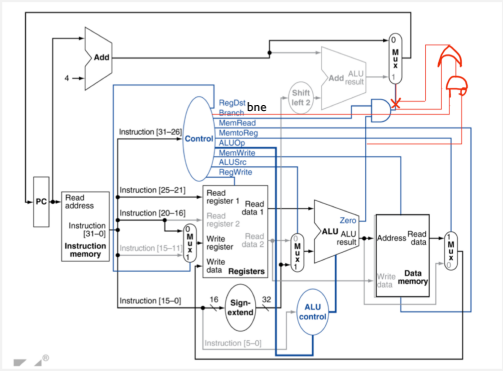
	Instruction	Interpretation
a.	`SEQ Rd,Rs,Rt`	`Reg[Rd]` = Boolean value (0 or 1) of (`Reg[Rs]` == `Reg[Rs]`)
b.	`LWI Rt,Rd(Rs)`	`Reg[Rt]` = `Mem[Reg[Rd]+Reg[Rs]]`

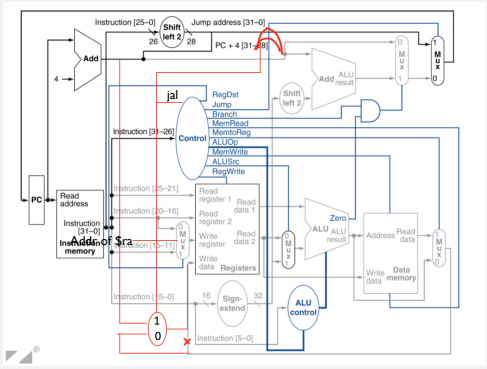
4.2.2 [10] <4.1> Which new functional blocks (if any) do we need for this instruction?

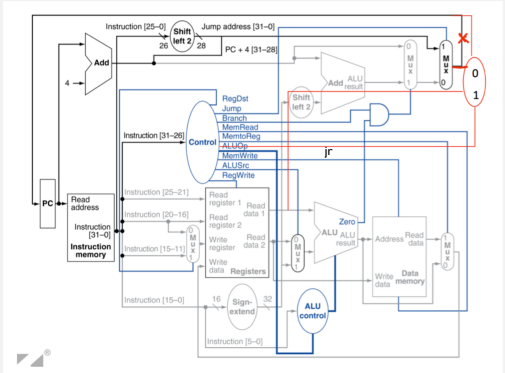
4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

(a). A new mux is required before Write Data of Registers to selected between Zero output from ALU, data read from Data Memory, and Result from ALU.
New signal: ZerotoReg to control the mux.
When it is 1, select Zero to Write Data of Registers

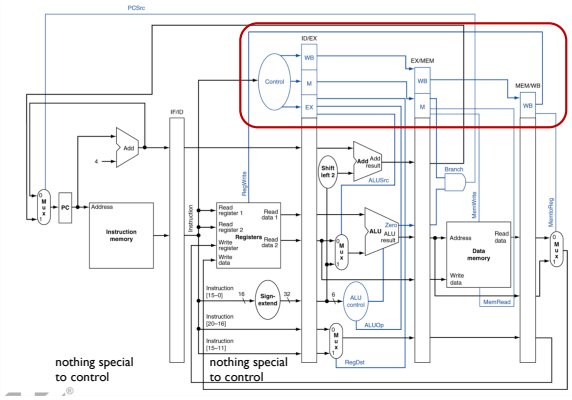
(b). No new functional block.
No new signal
RegDst=1 ALUOp=0010 MemtoReg=1 RegWrite=1
ALUSrc=0 MemRead=1 MemWrite=0 Branch=0
By changing the original signals, we can support this instruction.

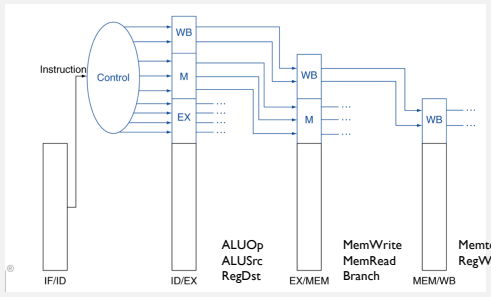






Pipelined Processor





- Remember the corresponding control signals in each stage
- The signals are consumed in appropriate stages

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back 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

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.

- X means that the certain value of this signal will not effect the execution of the instruction.

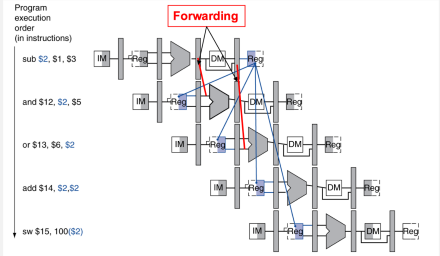
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

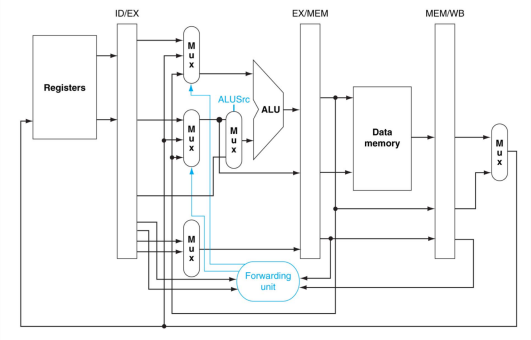
- Raising edge for writing and falling edge for reading
- Separate the read and write latency for Registers.

Data Hazard

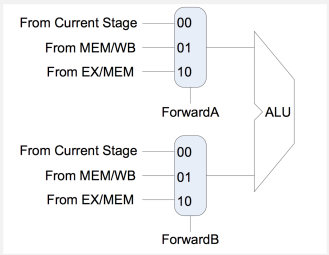
ALU Data Hazard - Forwarding



Cannot connect the input and output directly. Always through the pipeline registers



Forwarding Path and Condition



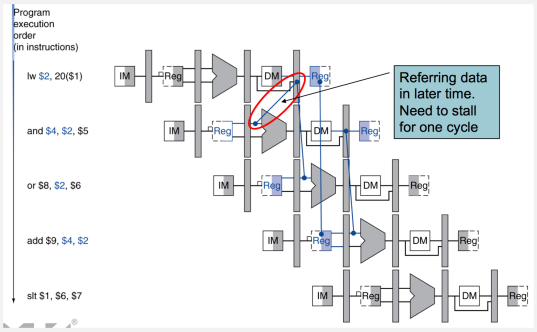
Remember the sequence of the inputs

EX hazard

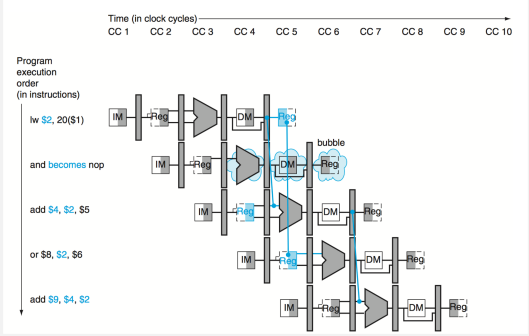
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
  MUX select signal ForwardA = 10
- if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
  MUX select signal ForwardB = 10
- MEM hazard (not EX hazard)
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRs)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (MEM/WB.RegisterRd = ID/EX.RegisterRt)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)))
    ForwardB = 01

There is no hazard in the WB stage because we assume that the register file supplies the correct result if the instruction in the ID stage reads the same register written by the instruction in the WB stage.
The register file performs another form of forwarding within the registers.

Load-Use Data Hazard - Stalls



Forwarding cannot solve the hazard since the result hasn't been read from DM.



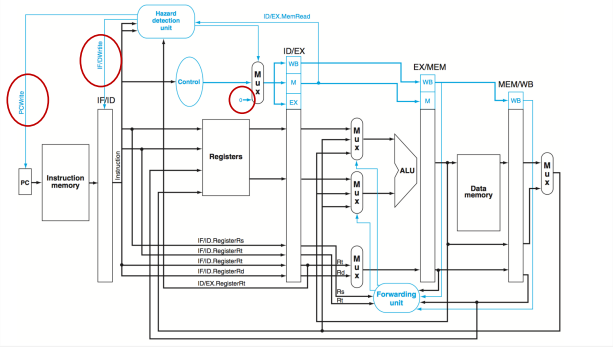
Forwarding cannot solve the hazard since the result hasn't been read from DM.

- Need to check only for a ***load*** instruction, determined by
  - ID/EX.MemRead
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt == IF/ID.RegisterRs) or
    (ID/EX.RegisterRt == IF/ID.RegisterRt))
- Hazard Detection is in **ID** stage
- If detected, stall and insert bubble

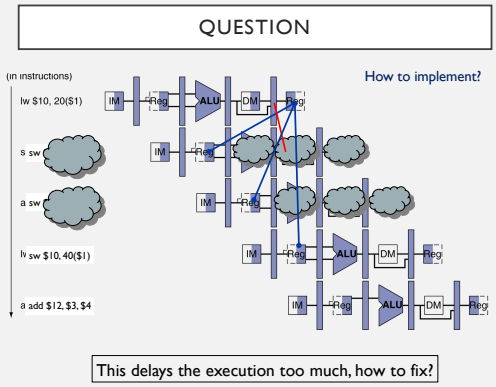
The destination register of lw instruction is going to be used in the next instruction

How to Stall the Pipeline?

- Force control signals in ID/EX register to 0's
  - EX, MEM and WB in following cycles do no-operation (nop) with those 0 control signals
- Prevent updates of PC and IF/ID registers
  - Instruction in IF/ID is held and decoded again
  - PC is held, so the same instruction is fetched again



PCWrite = 0 IF/IDWrite = 0 Control signals in EX, MEM, WB = 0



This delays the execution too much, how to fix?

- 1. Do not directly connect input and output of data memory, always through a pipeline register
- 2. Provide output of data memory to the EX/MEM register and connect the output of pipeline register to the input of the data memory
- 3. Add a mux to select signal to the input of data memory.