

CS152

Computer Architecture and Engineering

Lecture 10: Designing a Single Cycle Control

February 17, 1995

Dave Patterson (patterson@cs) and
Shing Kong (shing.kong@eng.sun.com)

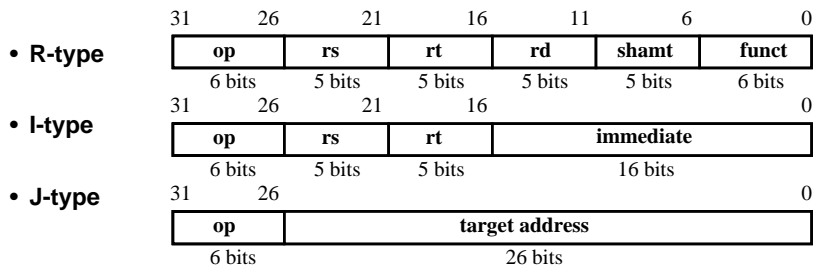
Slides available on <http://http.cs.berkeley.edu/~patterson>

cs 152 control.1

©DAP & SIK 1995

Recap: The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:

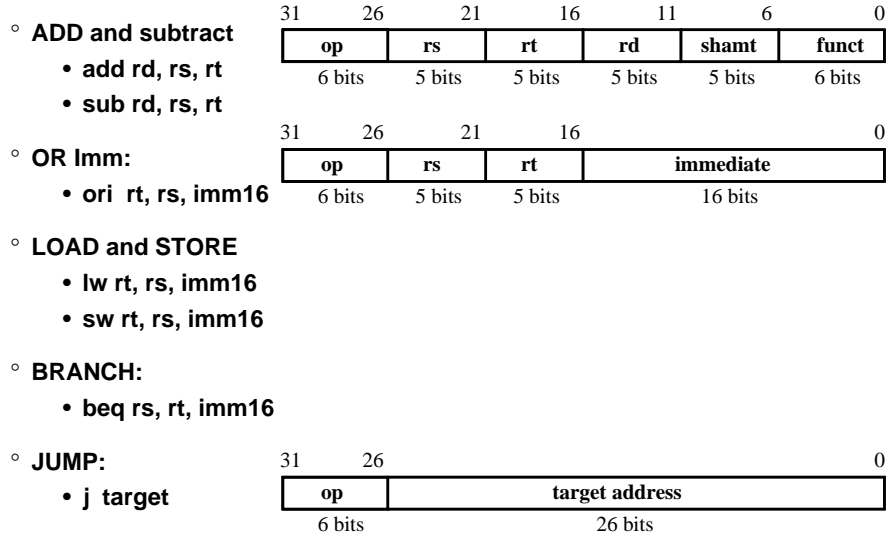


- The different fields are:
 - op: operation of the instruction
 - rs, rt, rd: the source and destination registers specifier
 - shamt: shift amount
 - funct: selects the variant of the operation in the “op” field
 - address / immediate: address offset or immediate value
 - target address: target address of the jump instruction

cs 152 control.2

©DAP & SIK 1995

Recap: The MIPS Subset

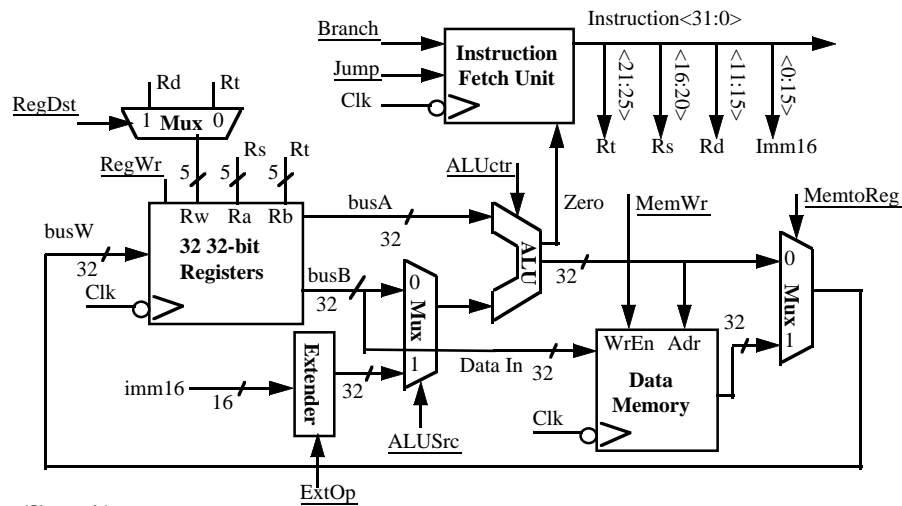


cs 152 control.3

©DAP & SIK 1995

Recap: A Single Cycle Datapath

- We have everything except control signals (underline)
 - Today's lecture will show you how to generate the control signals

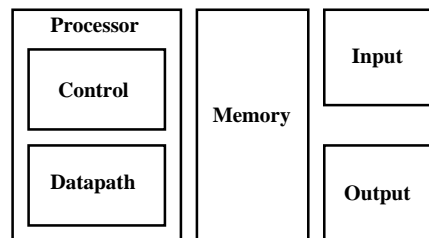


cs 152 control.4

©DAP & SIK 1995

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer

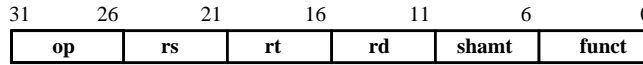


- Today's Topic: Designing the Control for the Single Cycle Datapath

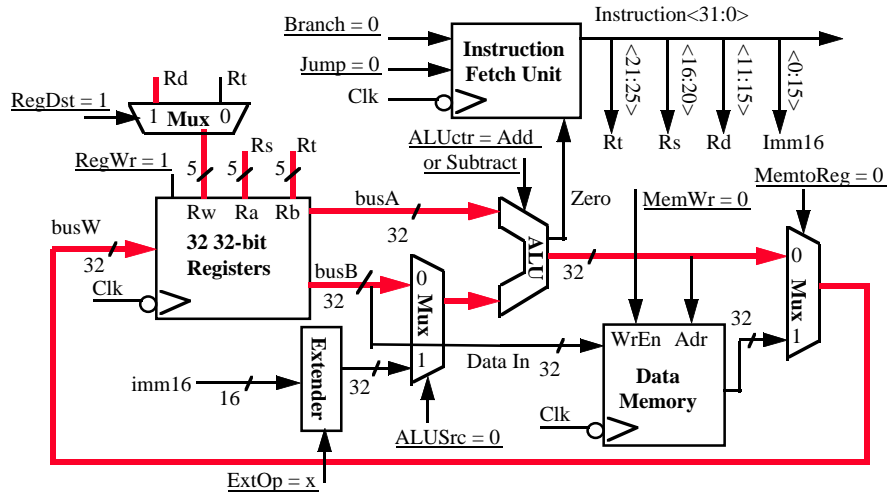
Outline of Today's Lecture

- Recap and Introduction (10 minutes)
- Control for Register-Register & Or Immediate instructions (10 minutes)
- Questions and Administrative Matters (5 minutes)
- Control signals for Load, Store, Branch, & Jump (15 minutes)
- Building a local controller: ALU Control (10 minutes)
- Break (5 minutes)
- The main controller (20 minutes)
- Summary (5 minutes)

The Single Cycle Datapath during Add and Subtract



◦ $R[rd] \leftarrow R[rs] +/- R[rt]$



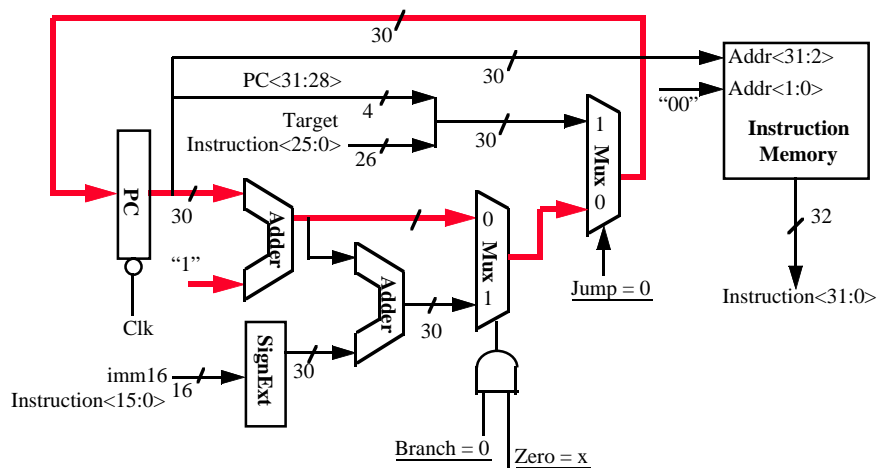
cs 152 control.9

©DAP & SIK 1995

Instruction Fetch Unit at the End of Add and Subtract

◦ $PC \leftarrow PC + 4$

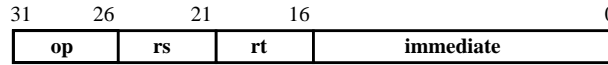
- This is the same for all instructions except: Branch and Jump



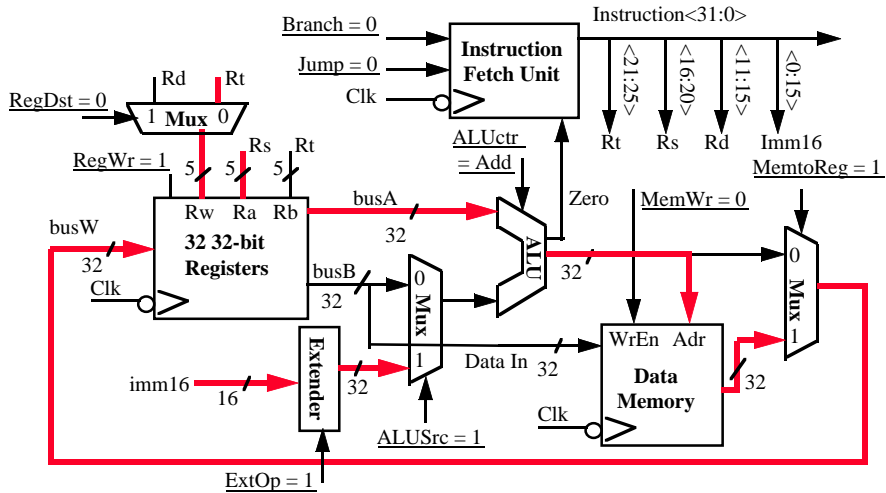
cs 152 control.10

©DAP & SIK 1995

The Single Cycle Datapath during Load



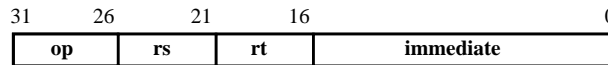
◦ $R[rt] \leftarrow \text{Data Memory} \{R[rs] + \text{SignExt}[\text{imm16}]\}$



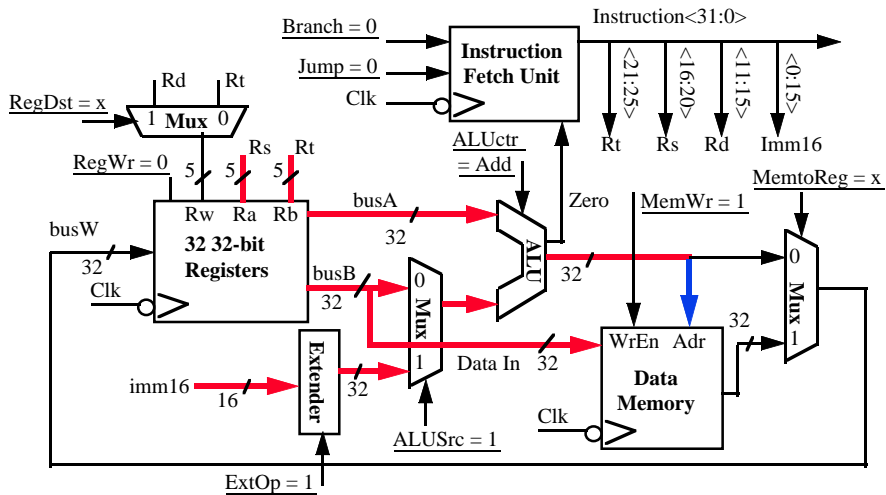
cs 152 control.13

©DAP & SIK 1995

The Single Cycle Datapath during Store



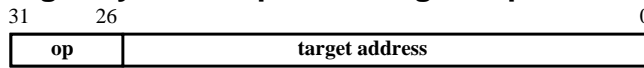
◦ $\text{Data Memory} \{R[rs] + \text{SignExt}[\text{imm16}]\} \leftarrow R[rt]$



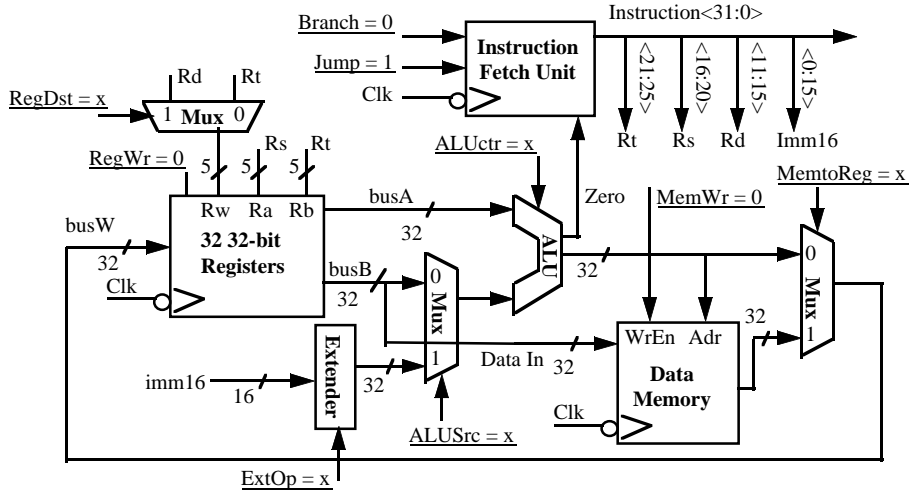
cs 152 control.14

©DAP & SIK 1995

The Single Cycle Datapath during Jump



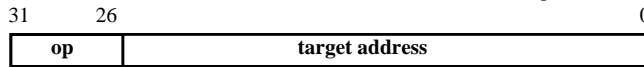
- Nothing to do! Make sure control signals are set correctly!



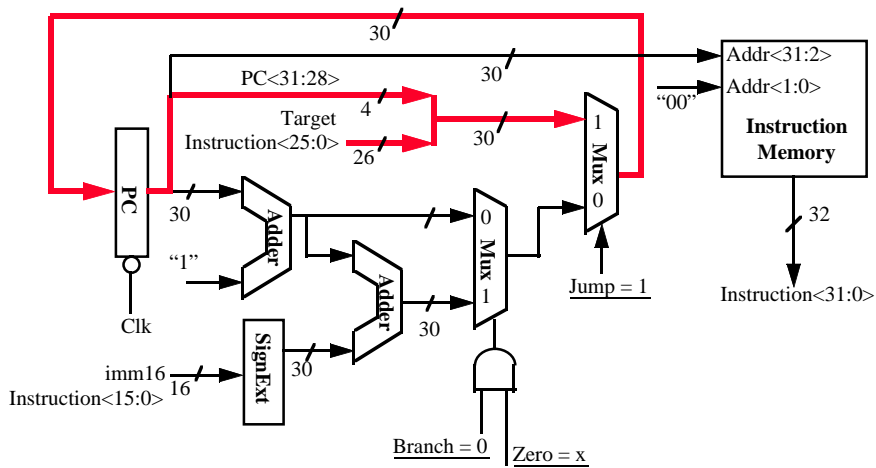
cs 152 control.17

©DAP & SIK 1995

Instruction Fetch Unit at the End of Jump



- $PC \leftarrow PC\langle 31:29 \rangle \text{ concat target}\langle 25:0 \rangle \text{ concat "00"}$



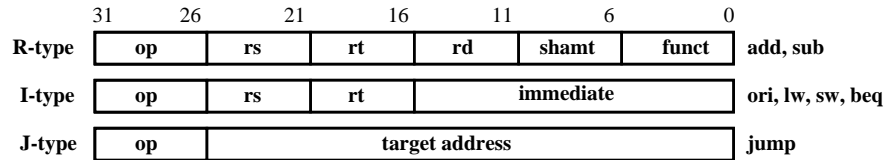
cs 152 control.18

©DAP & SIK 1995

A Summary of the Control Signals

See Appendix A

	func 10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	0	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

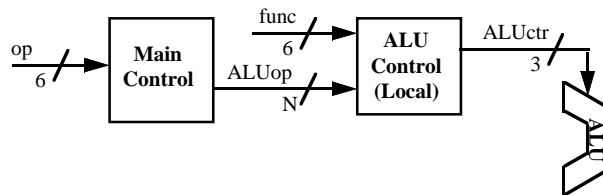


cs 152 control.19

©DAP & SIK 1995

The Concept of Local Decoding

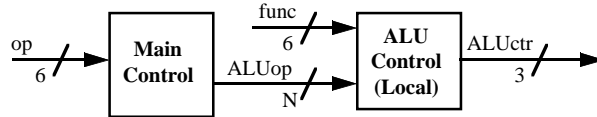
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtract	xxx



cs 152 control.20

©DAP & SIK 1995

The Encoding of ALUop



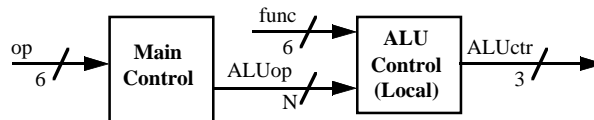
- In this exercise, ALUop has to be 2 bits wide to represent:
 - (1) “R-type” instructions
 - “I-type” instructions that require the ALU to perform:
 - (2) Or, (3) Add, and (4) Subtract
- To implement the full MIPS ISA, ALUop has to be 3 bits to represent:
 - (1) “R-type” instructions
 - “I-type” instructions that require the ALU to perform:
 - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx

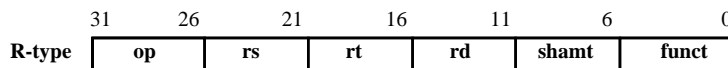
cs 152 control.21

©DAP & SIK 1995

The Decoding of the “func” Field



	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx



Recall ALU Homework (also P. 286 text):

funct<5:0>	Instruction Operation
10 0000	add
10 0010	subtract
10 0100	and
10 0101	or
10 1010	set-on-less-than



ALUctr<2:0>	ALU Operation
000	Add
001	Subtract
010	And
110	Or
111	Set-on-less-than

cs 152 control.22

©DAP & SIK 1995

The Truth Table for ALUctr

ALUop (Symbolic)	R-type "R-type"	ori	lw	sw	beq	func<3:0>	Instruction Op.
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	0000	add
						0010	subtract
						0100	and
						0101	or
						1010	set-on-less-than

ALUop			func				ALU	ALUctr		
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	Operation	bit<2>	bit<1>	bit<0>
0	0	0	x	x	x	x	Add	0	1	0
0	x	1	x	x	x	x	Subtract	1	1	0
0	1	x	x	x	x	x	Or	0	0	1
1	x	x	0	0	0	0	Add	0	1	0
1	x	x	0	0	1	0	Subtract	1	1	0
1	x	x	0	1	0	0	And	0	0	0
1	x	x	0	1	0	1	Or	0	0	1
1	x	x	1	0	1	0	Set on <	1	1	1

Break (5 Minutes)

The Logic Equation for ALUctr<2>

ALUop			func				ALUctr<2>
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	
0	x	1	x	x	x	x	1
1	x	x	0	0	1	0	1
1	x	x	1	0	1	0	1

This makes func<3> a don't care

- $ALUctr<2> = !ALUop<2> \& ALUop<0> +$
 $ALUop<2> \& !func<2> \& func<1> \& !func<0>$

The Logic Equation for ALUctr<1>

ALUop			func				ALUctr<1>
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	
0	0	0	x	x	x	x	1
0	x	1	x	x	x	x	1
1	x	x	0	0	0	0	1
1	x	x	0	0	1	0	1
1	x	x	1	0	1	0	1

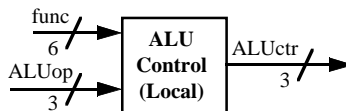
- $ALUctr<1> = !ALUop<2> \& !ALUop<0> +$
 $ALUop<2> \& !func<2> \& !func<0>$

The Logic Equation for ALUctr<0>

ALUop			func				ALUctr<0>
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>	
0	1	x	x	x	x	x	1
1	x	x	0	1	0	1	1
1	x	x	1	0	1	0	1

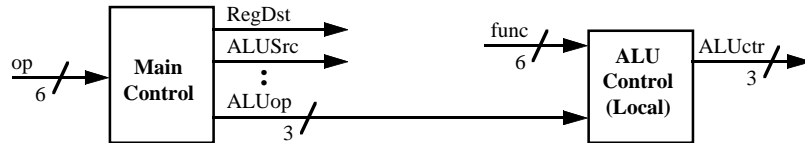
- $ALUctr<0> = !ALUop<2> \& ALUop<0>$
 $+ ALUop<2> \& !func<3> \& func<2> \& !func<1> \& func<0>$
 $+ ALUop<2> \& func<3> \& !func<2> \& func<1> \& !func<0>$

The ALU Control Block



- $ALUctr<2> = !ALUop<2> \& ALUop<0> +$
 $ALUop<2> \& !func<2> \& func<1> \& !func<0>$
- $ALUctr<1> = !ALUop<2> \& !ALUop<0> +$
 $ALUop<2> \& !func<2> \& !func<0>$
- $ALUctr<0> = !ALUop<2> \& ALUop<0>$
 $+ ALUop<2> \& !func<3> \& func<2> \& !func<1> \& func<0>$
 $+ ALUop<2> \& func<3> \& !func<2> \& func<1> \& !func<0>$

The "Truth Table" for the Main Control



op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUOp (Symbolic)	"R-type"	Or	Add	Add	Subtract	xxx
ALUOp <2>	1	0	0	0	0	x
ALUOp <1>	0	1	0	0	0	x
ALUOp <0>	0	0	0	0	1	x

cs 152 control.29

©DAP & SIK 1995

The "Truth Table" for RegWrite

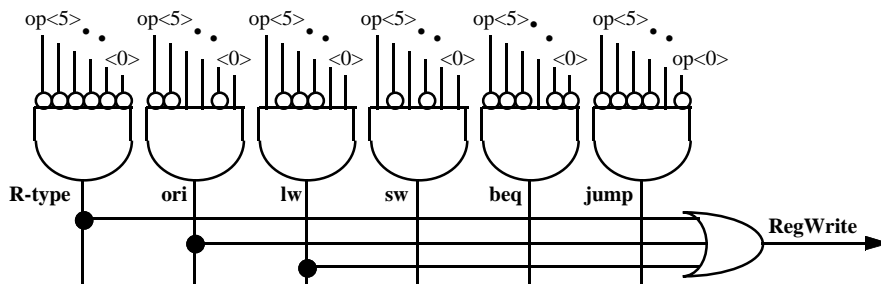
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegWrite	1	1	1	x	x	x

◦ RegWrite = R-type + ori + lw

$$= !op<5> \& !op<4> \& !op<3> \& !op<2> \& !op<1> \& !op<0> \quad (\text{R-type})$$

$$+ !op<5> \& !op<4> \& op<3> \& op<2> \& !op<1> \& op<0> \quad (\text{ori})$$

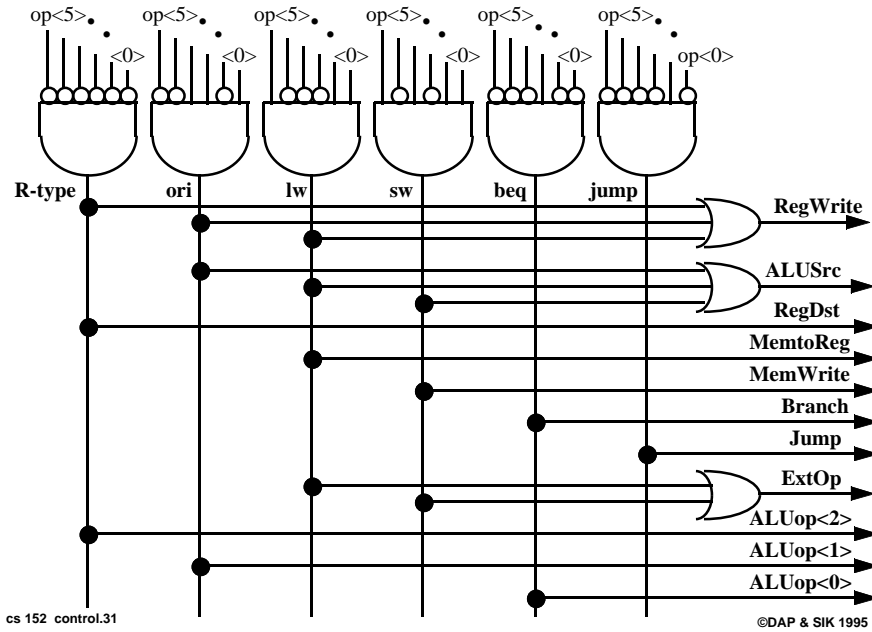
$$+ op<5> \& !op<4> \& !op<3> \& !op<2> \& op<1> \& op<0> \quad (\text{lw})$$



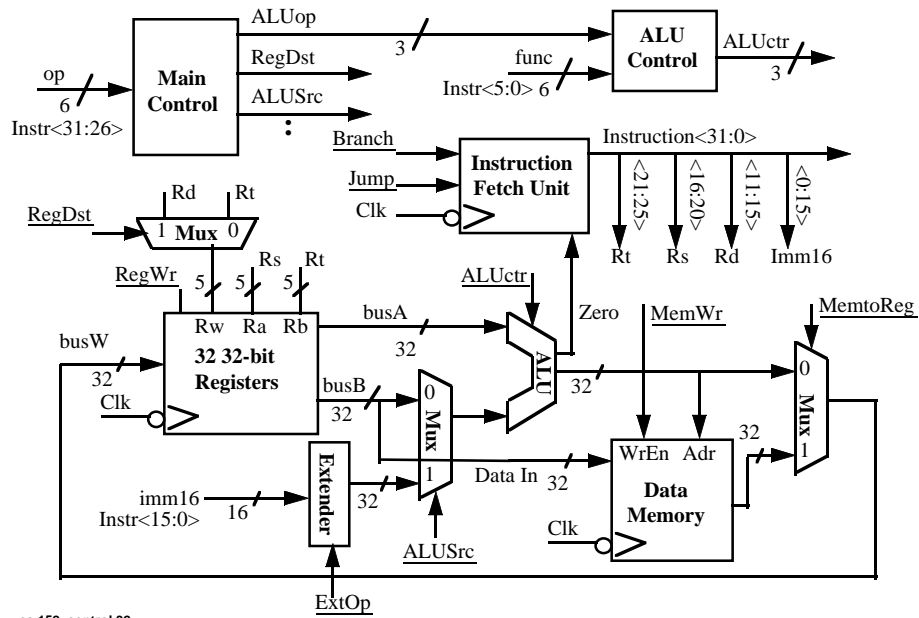
cs 152 control.30

©DAP & SIK 1995

PLA Implementation of the Main Control



Putting it All Together: A Single Cycle Processor



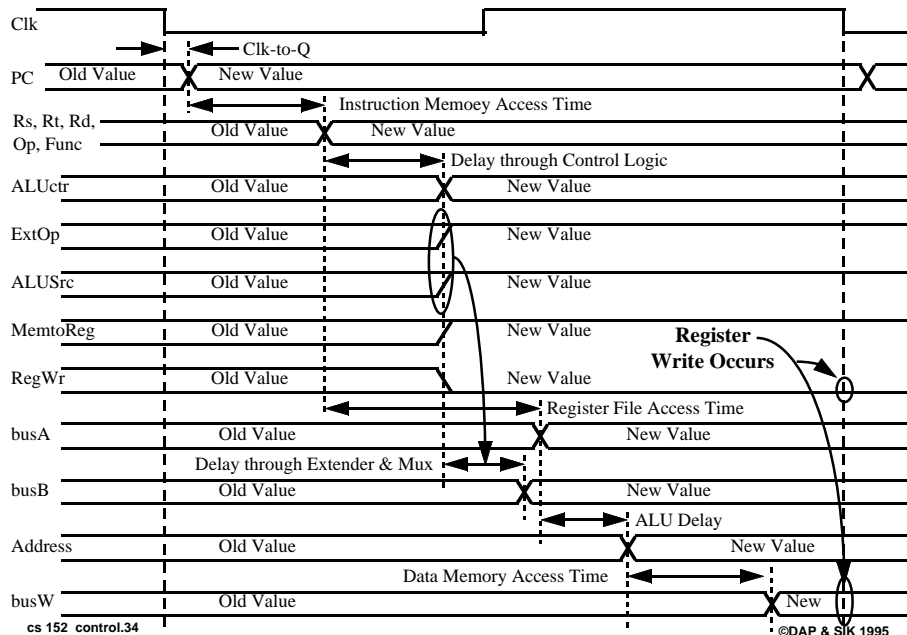
How is this Different from a Real MIPS Processor?

- The effect of load in a real MIPS Processor is delayed:
 - lw \$1, 100 (\$2) // Load Register R1
 - add \$3, \$1, \$0 // Move "old" R1 into R3
 - add \$4, \$1, \$0 // Move "new" R1 into R4
- The effect of load in our single cycle process is NOT delayed
 - lw \$1, 100 (\$2) // Load Register R1
 - add \$3, \$1, \$0 // Move "new" R1 into R3
- The effect of branch and jump in a real MIPS Processor is delayed:
 - Instruction Address: 0x00 j 1000
 - Instruction Address: 0x04 add \$1, \$2, \$3
 - Instruction Address: 0x1000 sub \$1, \$2, \$3
- Branch and jump in our single cycle process is NOT delayed
 - Instruction Address: 0x00 j 1000
 - Instruction Address: 0x1000 sub \$1, \$2, \$3

cs 152 control.33

©DAP & SIK 1995

Worst Case Timing



cs 152 control.34

©DAP & SIK 1995

Drawback of this Single Cycle Processor

- Long cycle time:
 - Cycle time must be long enough for the load instruction:
 - PC's Clock -to-Q +
 - Instruction Memory Access Time +
 - Register File Access Time +
 - ALU Delay (address calculation) +
 - Data Memory Access Time +
 - Register File Setup Time +
 - Clock Skew
- Cycle time is much longer than needed for all other instructions

Where to get more information?

- Chapter 5.1 to 5.3 of your text book:
 - Daid Patterson and John Hennessy, "Computer Organization & Design: The Hardware / Software Interface," Morgan Kaufman Publishers, San Mateo, California, 1994.
- One of the best PhD thesis on processor design:
 - Manolis Katevenis, "Reduced Instruction Set Computer Architecture for VLSI," PhD Dissertation, EECS, U C Berkeley, 1982.
- For a reference on the MIPS architecture:
 - Gerry Kane, "MIPS RISC Architecture," Prentice Hall.