

Computer System Architecture

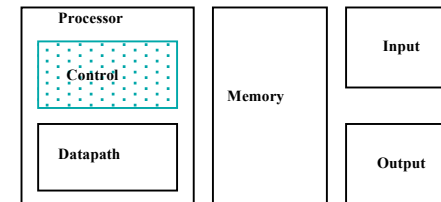
Processor Part II

Chalermek Intanagonwiwat

Slides courtesy of John Hennessy and David Patterson

Where are We Now?

- The Five Classic Components of a Computer



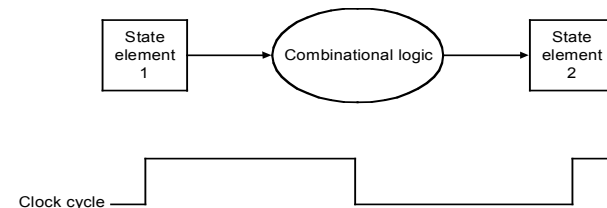
- Today's Topic: Designing the Control for the Single Cycle Datapath

Our Simple Control Structure

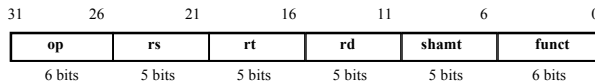
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
 - ALU might not produce "right answer" right away
 - we use write signals along with clock to determine when to write

Our Simple Control Structure (cont.)

- Cycle time determined by length of the longest path

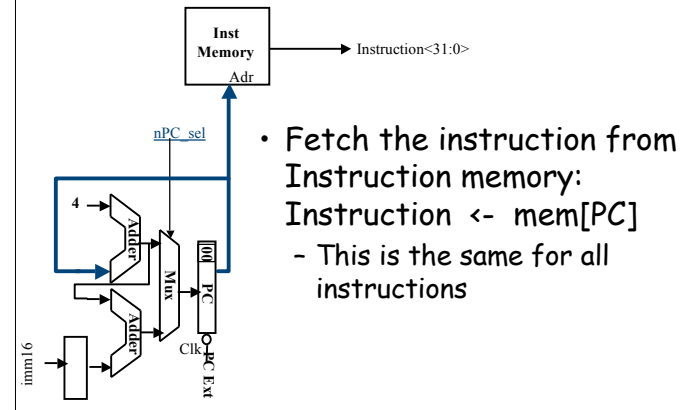


RTL: The Add Instruction

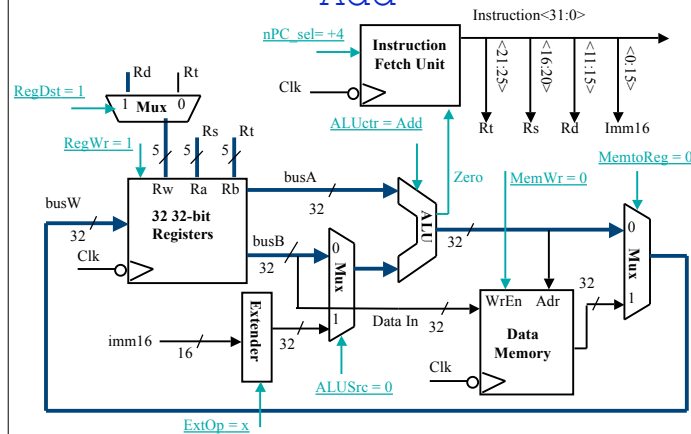


- add rd, rs, rt
 - mem[PC] Fetch the instruction from memory
 - $R[rd] \leftarrow R[rs] + R[rt]$ The actual operation
 - $PC \leftarrow PC + 4$ Calculate the next instruction's address

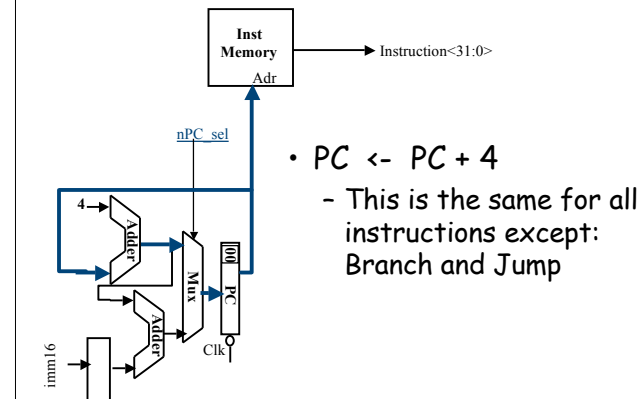
Instruction Fetch Unit at the Beginning of Add



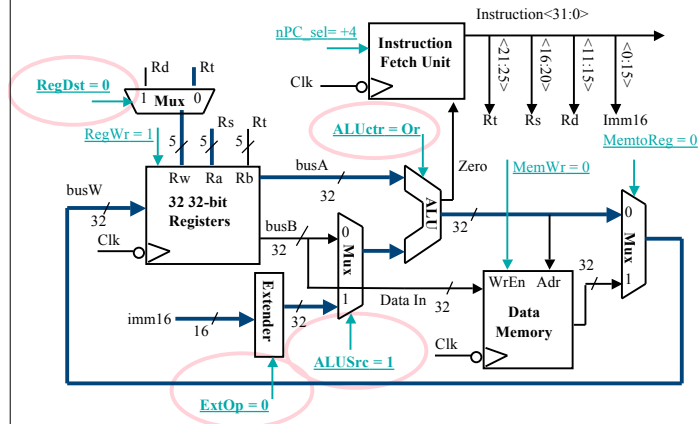
The Single Cycle Datapath during Add



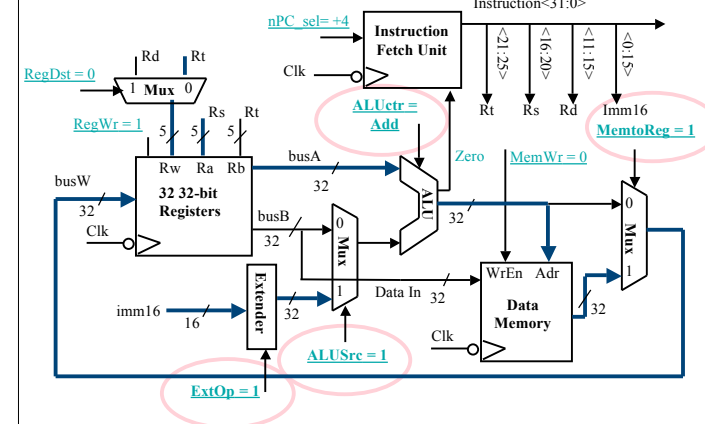
Instruction Fetch Unit at the End of Add



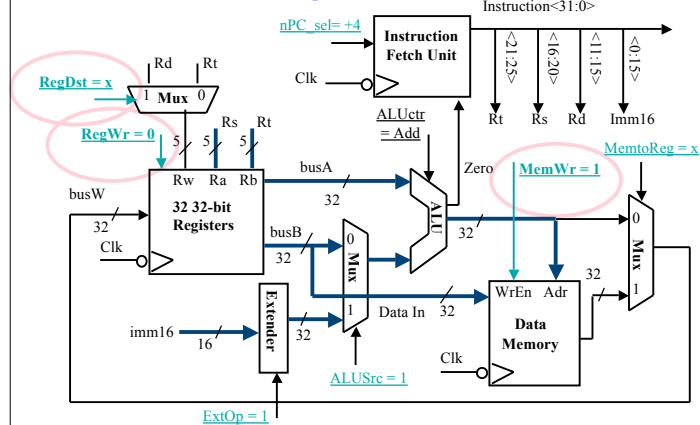
The Single Cycle Datapath during Or Immediate



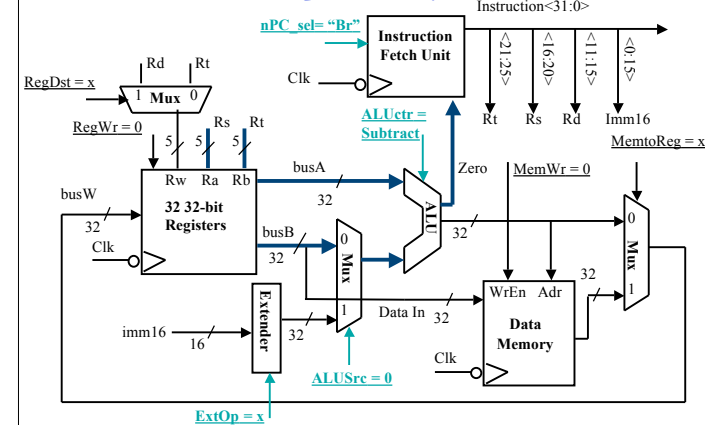
The Single Cycle Datapath during Load



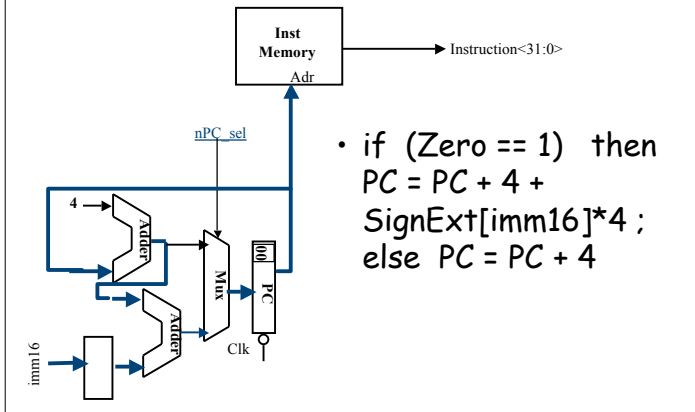
The Single Cycle Datapath during Store



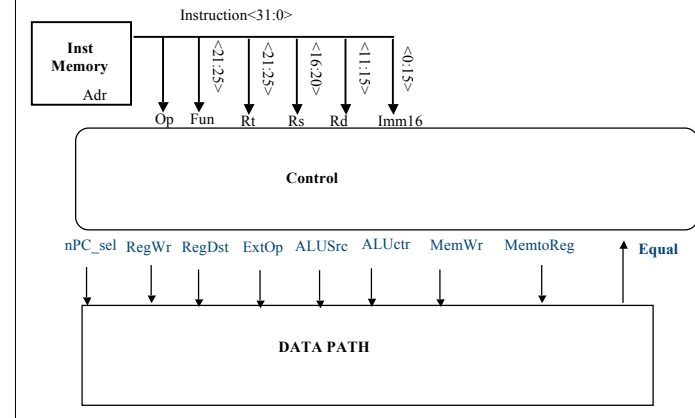
The Single Cycle Datapath during Branch



Instruction Fetch Unit at the End of Branch



Step 4: Given Datapath: RTL -> Control



A Summary of Control Signals

inst Register Transfer

ADD $R[\text{rd}] \leftarrow R[\text{rs}] + R[\text{rt}]; \quad PC \leftarrow PC + 4$
 $\text{ALUSrc} = \text{RegB}, \text{ALUctr} = \text{"add"}, \text{RegDst} = \text{rd},$
 $\text{RegWr}, \text{nPC_sel} = \text{"+4"}$

SUB $R[\text{rd}] \leftarrow R[\text{rs}] - R[\text{rt}]; \quad PC \leftarrow PC + 4$
 $\text{ALUSrc} = \text{RegB}, \text{ALUctr} = \text{"sub"}, \text{RegDst} = \text{rd},$
 $\text{RegWr}, \text{nPC_sel} = \text{"+4"}$

ORI $R[\text{rt}] \leftarrow R[\text{rs}] \mid \text{zero_ext}(\text{Imm16}); \quad PC \leftarrow PC + 4$
 $\text{ALUSrc} = \text{Im}, \text{Extop} = \text{"Z"}, \text{ALUctr} = \text{"or"},$
 $\text{RegDst} = \text{rt}, \text{RegWr}, \text{nPC_sel} = \text{"+4"}$

A Summary of Control Signals (cont.)

LOAD $R[\text{rt}] \leftarrow \text{MEM}[R[\text{rs}] + \text{sign_ext}(\text{Imm16})];$
 $PC \leftarrow PC + 4$
 $\text{ALUSrc} = \text{Im}, \text{Extop} = \text{"Sn"}, \text{ALUctr} = \text{"add"},$
 $\text{MemtoReg}, \text{RegDst} = \text{rt}, \text{RegWr}, \quad \text{nPC_sel} = \text{"+4"}$

STORE $\text{MEM}[R[\text{rs}] + \text{sign_ext}(\text{Imm16})] \leftarrow R[\text{rs}];$
 $PC \leftarrow PC + 4$
 $\text{ALUSrc} = \text{Im}, \text{Extop} = \text{"Sn"}, \text{ALUctr} = \text{"add"},$
 $\text{MemWr}, \text{nPC_sel} = \text{"+4"}$

BEQ if ($R[\text{rs}] == R[\text{rt}]$) then
 $PC \leftarrow PC + \text{sign_ext}(\text{Imm16}) \mid \mid \text{OO}$ else $PC \leftarrow PC + 4$
 $\text{nPC_sel} = \text{"Br"}, \text{ALUctr} = \text{"sub"}$

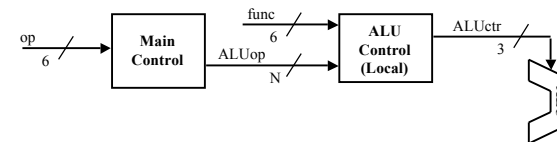
A Summary of the Control Signals (cont.)

See Appendix A

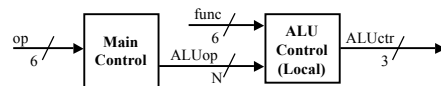
	10 0000	10 0010	We Don't Care :-)				
	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr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

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



The Encoding of ALUop



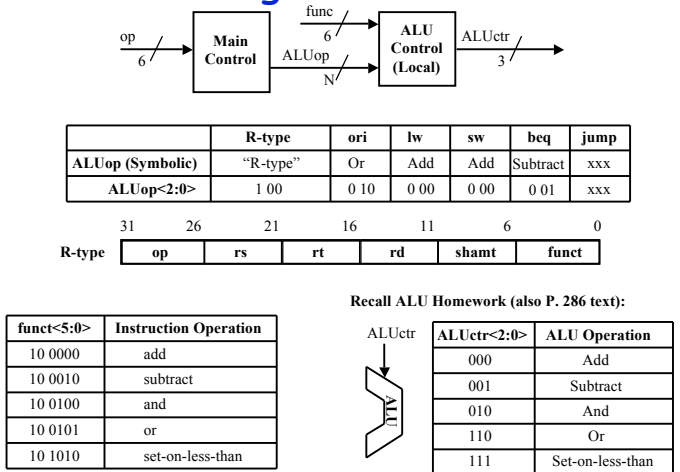
- In this exercise, ALUop has to be 2 bits wide to represent:
 - (1) "R-type" instructions
 - "I-type" instructions that require the ALU to perform:
 - (2) Or, (3) Add, and (4) Subtract

The Encoding of ALUop (cont.)

- 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

The Decoding of the "func" Field



The Truth Table for ALUctr

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

ALUop			func				ALU Operation	ALUctr		
bit<2>	bit<1>	bit<0>	bit<3>	bit<2>	bit<1>	bit<0>		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

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

$$\bullet \text{ALUctr<2>} = \text{!ALUop<2>} \& \text{ALUop<0>} + \text{ALUop<2>} \& \text{!func<2>} \& \text{func<1>} \& \text{!func<0>}$$

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

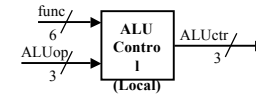
$$\bullet \text{ALUctr<1>} = \text{!ALUop<2>} \& \text{!ALUop<1>} + \text{ALUop<2>} \& \text{!func<2>} \& \text{!func<0>}$$

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



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

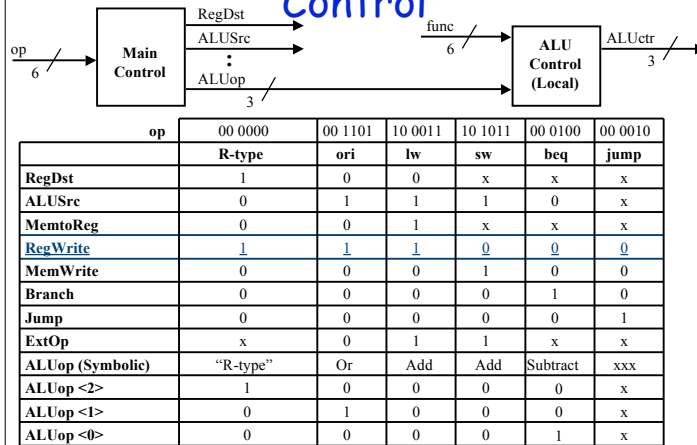
Step 5: Logic for each control signal

- $nPC_sel \leftarrow \text{if } (OP == BEQ) \text{ then } EQUAL \text{ else } 0$
- $ALUsrc \leftarrow \text{if } (OP == "Rtype") \text{ then } "regB" \text{ else } "immed"$
- $ALUctr \leftarrow \text{if } (OP == "Rtype") \text{ then } func \text{ elseif } (OP == ORi) \text{ then } "OR" \text{ elseif } (OP == BEQ) \text{ then } "sub" \text{ else } "add"$

Step 5: Logic for each control signal (cont.)

- $ExtOp \leftarrow \text{if } (OP == ORi) \text{ then } "zero" \text{ else } "sign"$
- $MemWr \leftarrow (OP == Store)$
- $MemtoReg \leftarrow (OP == Load)$
- $RegWr: \leftarrow \text{if } ((OP == Store) \mid (OP == BEQ)) \text{ then } 0 \text{ else } 1$
- $RegDst: \leftarrow \text{if } ((OP == Load) \mid (OP == ORi)) \text{ then } 0 \text{ else } 1$

The "Truth Table" for the Main Control

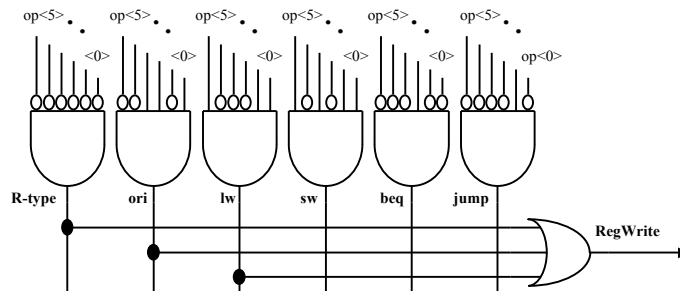


The "Truth Table" for RegWrite

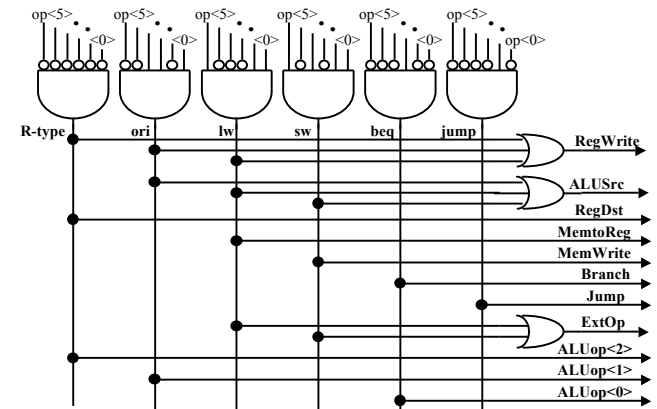
op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	ori	lw	sw	beq	jump
RegWrite	1	1	1	0	0	0

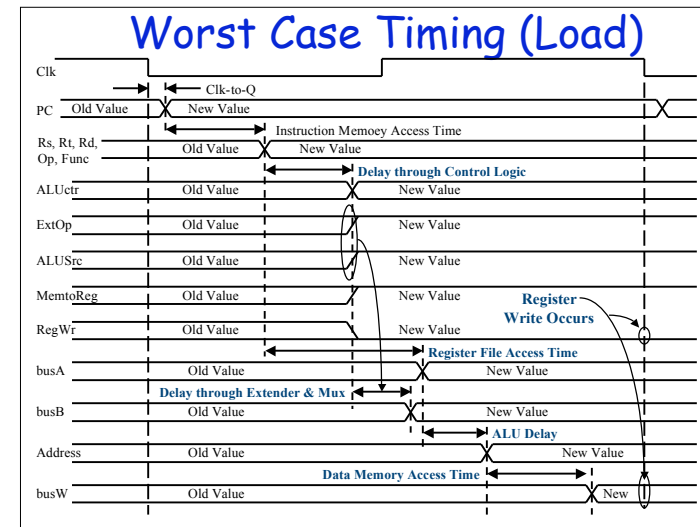
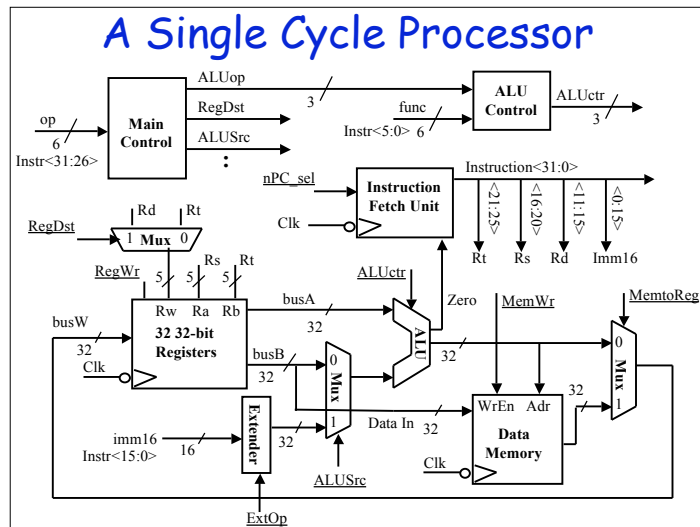
- RegWrite = R-type + ori + lw
- = !op<5> & !op<4> & !op<3> & !op<2> & !op<1> & !op<0> (R-type)
- + !op<5> & !op<4> & op<3> & op<2> & !op<1> & op<0> (ori)
- + op<5> & !op<4> & !op<3> & !op<2> & op<1> & op<0> (lw)

The "Truth Table" for RegWrite (cont.)



PLA Implementation of the Main Control





Drawback of Single Cycle CPU

- 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 for load is much longer than needed for all other instructions

Summary

- Single cycle datapath => CPI=1, CCT => long
- 5 steps to design a processor
 1. Analyze instruction set => datapath requirements
 2. Select set of datapath components & establish clock methodology
 3. Assemble datapath meeting the requirements
 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
 5. Assemble the control logic
- Control is the hard part
- MIPS makes control easier
 - Instructions same size
 - Source registers always in same place
 - Immediates same size, location
 - Operations always on registers/immediates

