

CS152 – Computer Architecture and Engineering

Lecture 7 – Single Cycle Control

2003-09-16

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/



Review

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 (This Lecture)

MIPS makes it easier

- Instructions same size; Source registers, immediates always in same place
- Operations always on registers/immediates

Single cycle datapath => CPI=1, CCT => long

On-line Design Notebook

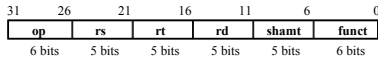
- Open a window and keep an editor running while you work; cut&paste
 - Former CS 152 students (and TAs) say they use on-line notebook for programming as well as hardware design; one of most valuable skills
- Refer to the handout as an example



Recap: The MIPS-lite Subset

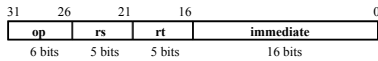
ADD and subtract

- add rd, rs, rt
- sub rd, rs, rt



OR Imm:

- ori rt, rs, imm16



LOAD and STORE

- lw rt, rs, imm16
- sw rt, rs, imm16

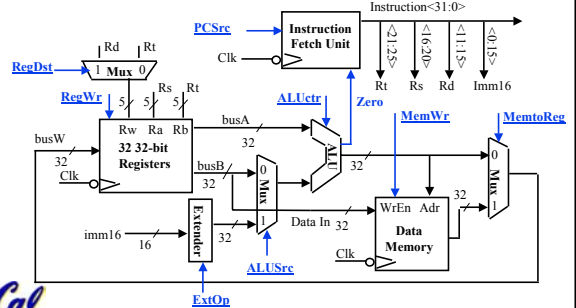
BRANCH:

- beq rs, rt, imm16



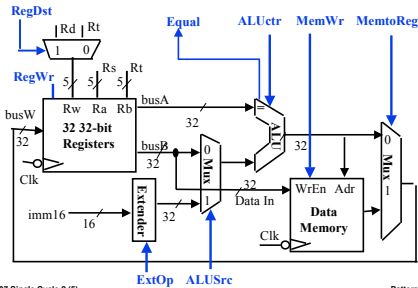
Recap: A Single Cycle Datapath

- Rs, Rt, Rd and lmed16 hardwired into datapath from Fetch Unit
- We have everything except control signals (underline)
- Today's lecture will show you how to generate the control signals



Meaning of the Control Signals

- ExtOp: "zero", "sign"
- ALUSrc: 0 => regB; 1 => immed
- ALUctr: "add", "sub", "or"
- MemWr: 1 => write memory
- MementoReg: 0 => ALU; 1 => Mem
- RegDst: 0 => "rt"; 1 => "rd"
- RegWr: 1 => write register



This lecture vs. Chapter 5 Beta 3/e

This lecture

- MIPS-lite subset:
- AddU, SubU, LW, SW
- BEQ, ORI

Control lines names

- MemtoReg, PCSrc, ALUSrc, RegDst
- MemWr, RegWr
- ExtOp (zero extend or sign extend)
- ALUctr 3 bits (no NOR)
- MemWr=0 => MemRead

Book

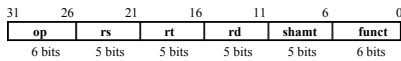
- MIPS-lite subset:
- AddU, SubU, LW, SW
- BEQ, OR
- AND, SLT, J

Control lines names

- MemtoReg, PCSrc, ALUSrc, RegDst
- MemWrite, RegWrite
- No ExtOp since subset immediates sign extend
- ALUoperation 4 bits
- MemRead & MemWrite



The Add Instruction



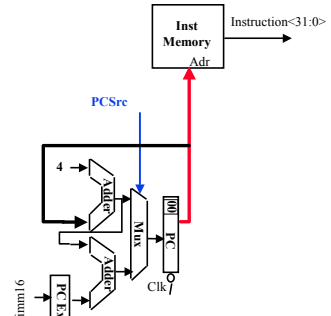
◦ add rd, rs, rt

- 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**

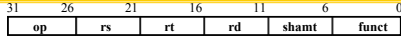


Instruction Fetch Unit at the Beginning of Add

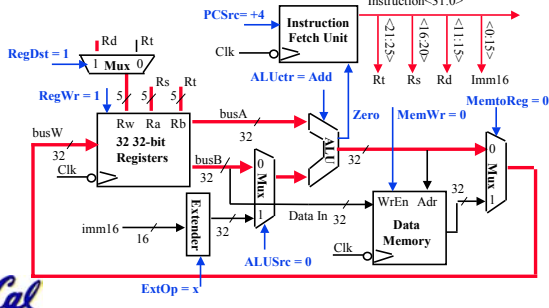
- Fetch the instruction from Instruction memory:
 $Instruction \leftarrow mem[PC]$
- This is the same for all instructions



The Single Cycle Datapath during Add

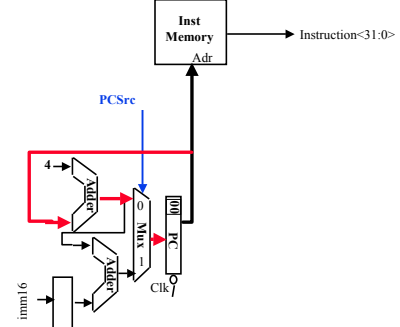


◦ $R[rd] \leftarrow R[rs] + R[rt]$



Instruction Fetch Unit at the End of Add

- $PC \leftarrow PC + 4$
- This is the same for all instructions except: Branch and Jump



Two equivalent ways to specify control

| | Control line 0 | Control line 1 | ... | Control line n | AddU | SubU | ORI | LW | SW | BEQ |
|------|----------------|----------------|-----|----------------|------|------|-----|----|----|-----|
| AddU | A | | | B | | | | | | X |
| SubU | | | | | | | | | | |
| ORI | | | | | | | | | | |
| LW | | | | | | | | | | |
| SW | | | | | | | | | | |
| BEQ | X | | | Y | | | | | | |

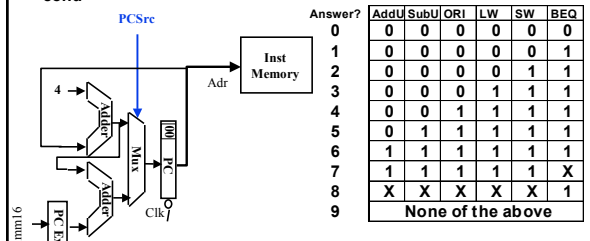
(Rotate about 45degree axis)

- Book does left version (Fig 5.18)
- We'll do right version by committee and transmitter, one control at a time



Setting PC Source Control Signal

- PCSrc: $0 \Rightarrow PC \leftarrow PC + 4$
 $1 \Rightarrow PC \leftarrow PC + 4 + \{SignExt(Imm16), 2b00\}$
- Later in lecture: higher-level connection between mux and branch cond

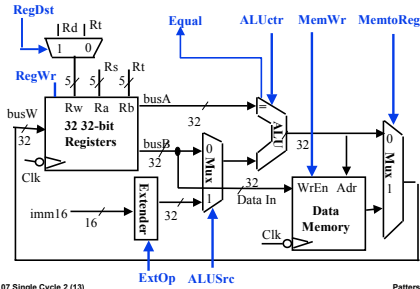


| Answer? | AddU | SubU | ORI | LW | SW | BEQ |
|---------|-------------------|------|-----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | X |
| 8 | X | X | X | X | X | 1 |
| 9 | None of the above | | | | | |



Meaning of the Control Signals

- ExtOp: 0 ⇒ "zero"; 1 ⇒ "sign" MemWr: 1 ⇒ write memory
- ALUSrc: 0 ⇒ regB; 1 ⇒ immed MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- ALUctr: "add", "sub", "or" RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register

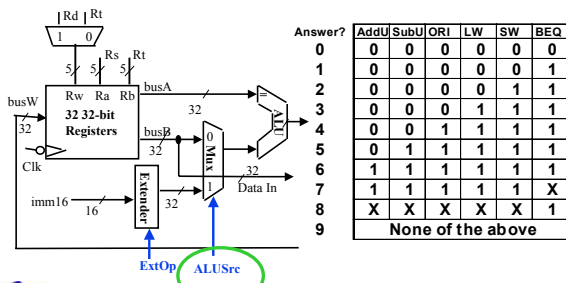


Administrivia

- Office hours in Lab
 - Mon 4 – 5:30 Jack, Mon 3 – 4:30 John
- Dave's office hours Tue 3:30 – 5
- Reading: Sections 5.6, 5.8, 5.11, 5.12 in Beta ed.

Specify ALU source mux Control

- ALUSrc: 0 ⇒ reg as ALU B input; 1 ⇒ immediate as ALU B input

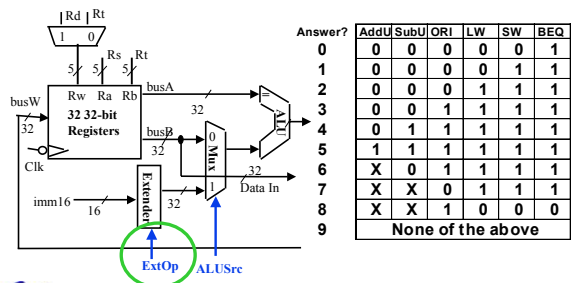


Answer?

| | AddU | SubU | ORI | LW | SW | BEQ |
|---|-------------------|------|-----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | X |
| 8 | X | X | X | X | X | 1 |
| 9 | None of the above | | | | | |

Specify Immediate Extender Op Control

- ExtOp: 0 ⇒ "zero extend immediate"; 1 ⇒ "sign extend imm."

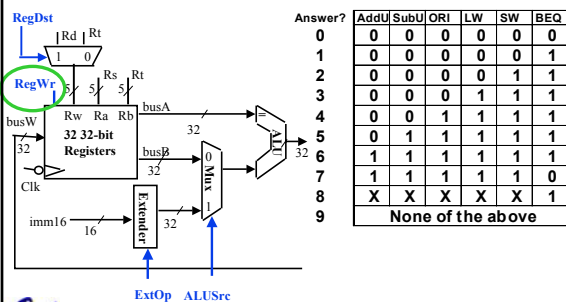


Answer?

| | AddU | SubU | ORI | LW | SW | BEQ |
|---|-------------------|------|-----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | X | 0 | 1 | 1 | 1 | 1 |
| 7 | X | X | 0 | 1 | 1 | 1 |
| 8 | X | X | 1 | 0 | 0 | 0 |
| 9 | None of the above | | | | | |

Specify Register Write Control

- RegWr: 1 ⇒ write register

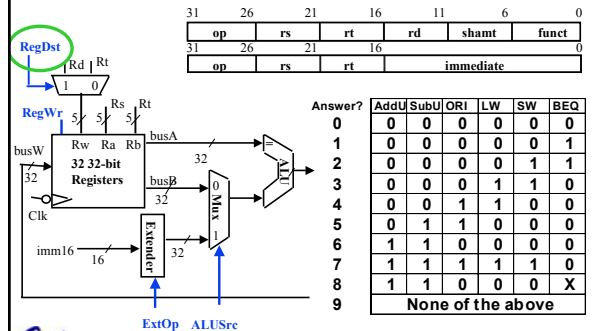


Answer?

| | AddU | SubU | ORI | LW | SW | BEQ |
|---|-------------------|------|-----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 |
| 8 | X | X | X | X | X | 1 |
| 9 | None of the above | | | | | |

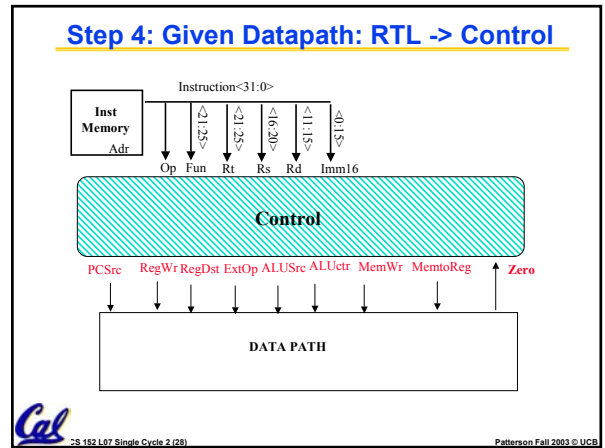
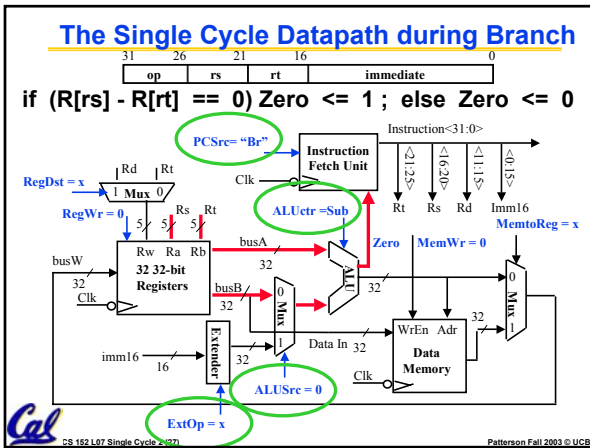
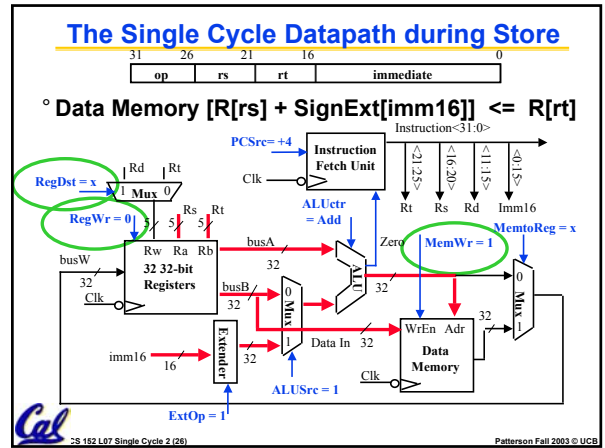
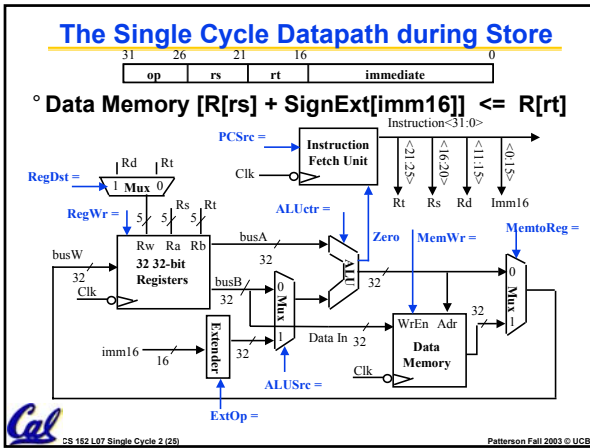
Specify Register Destination Control

- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"



Answer?

| | AddU | SubU | ORI | LW | SW | BEQ |
|---|-------------------|------|-----|----|----|-----|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 |
| 8 | 1 | 1 | 0 | 0 | 0 | X |
| 9 | None of the above | | | | | |



A Summary of Control Signals

| inst | Register Transfer | Control Signals |
|-------|--|--|
| ADD | $R[rd] \leftarrow R[rs] + R[rt]$ | $PC \leftarrow PC + 4$ $ALUSrc = \text{RegB}, ALUctr = \text{"add"}, \text{RegDst} = rd, \text{RegWr}, \text{PCSrc} = \text{"+4"}$ |
| SUB | $R[rd] \leftarrow R[rs] - R[rt]$ | $PC \leftarrow PC + 4$ $ALUSrc = \text{RegB}, ALUctr = \text{"sub"}, \text{RegDst} = rd, \text{RegWr}, \text{PCSrc} = \text{"+4"}$ |
| ORI | $R[rt] \leftarrow R[rs] + \text{zero_ext}(\text{Imm16})$ | $PC \leftarrow PC + 4$ $ALUSrc = \text{Im}, \text{ExtOp} = \text{"Z"}, ALUctr = \text{"or"}, \text{RegDst} = rt, \text{RegWr}, \text{PCSrc} = \text{"+4"}$ |
| LOAD | $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})]$ | $PC \leftarrow PC + 4$ $ALUSrc = \text{Im}, \text{ExtOp} = \text{"Sn"}, ALUctr = \text{"add"}, \text{MementoReg}, \text{RegDst} = rt, \text{RegWr}, \text{PCSrc} = \text{"+4"}$ |
| STORE | $\text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})] \leftarrow R[rs]$ | $PC \leftarrow PC + 4$ $ALUSrc = \text{Im}, \text{ExtOp} = \text{"Sn"}, ALUctr = \text{"add"}, \text{MementoReg}, \text{PCSrc} = \text{"+4"}$ |
| BEQ | if $(R[rs] == R[rt])$ then $PC \leftarrow PC + \text{sign_ext}(\text{Imm16})$ 00 else $PC \leftarrow PC + 4$ | $PCSrc = \text{"Br"}, ALUctr = \text{"sub"}$ |

CS 152 L07 Single Cycle 2 (29) Patterson Fall 2003 © UCB

A Summary of the Control Signals

| Sec | func | 10 0000 | 10 0010 | We Don't Care :-) | | | | |
|-------------|------|---------|----------|-------------------|---------|---------|----------|---------|
| Appendix A | op | 00 0000 | 00 0000 | 00 1101 | 10 0011 | 10 1011 | 00 0100 | 00 0010 |
| RegDst | add | 1 | 1 | 0 | 0 | x | x | x |
| ALUSrc | sub | 0 | 0 | 1 | 1 | 1 | 0 | x |
| MementoReg | ori | 0 | 0 | 0 | 1 | x | x | x |
| RegWrite | lw | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| MemWrite | sw | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| PCSrc | beq | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Jump | 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 |

| R-type | 31 | 26 | 21 | 16 | 11 | 6 | 0 | |
|--------|----|----|----|----|-------|-------|---|----------|
| | op | rs | rt | rd | shamt | funct | | add, sub |

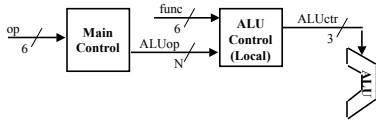
| I-type | 31 | 26 | 21 | 16 | 0 | |
|--------|----|----|----|-----------|---|------------------|
| | op | rs | rt | immediate | | ori, lw, sw, beq |

| J-type | 31 | 26 | 21 | 16 | 0 | | |
|--------|----|----------------|----|----|---|--|------|
| | op | target address | | | | | jump |

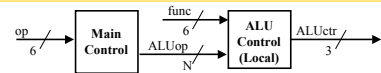
CS 152 L07 Single Cycle 2 (30) Patterson Fall 2003 © UCB

The Concept of Local Decoding

| op | 00 0000 | 00 1101 | 10 0011 | 10 1011 | 00 0100 | 00 0010 |
|------------|----------|---------|---------|---------|----------|---------|
| RegDst | R-type | ori | lw | sw | beq | jump |
| ALUSrc | 1 | 0 | 0 | 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:

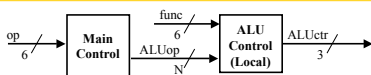
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
 - (2) Or, (3) Add, and (4) Subtract

° To implement the more of MIPS ISA, ALUop has to be 3 bits wide to represent (4 bits in book to include NOR):

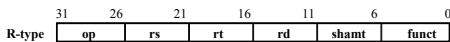
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
 - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

| ALUop (Symbolic) | 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



| ALUop (Symbolic) | 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 |



P. 286 text:

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

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

The Truth Table for ALUctr

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

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

$$° (\text{ALUctr}<2> = \neg \text{ALUop}<2> \& \text{ALUop}<0> + \text{ALUop}<2> \& \text{func}<2> \& \text{func}<1> \& \text{func}<0>$$

The Logic Equation for ALUctr<1>

| ALUop bit<2> | ALUop bit<1> | ALUop bit<0> | func bit<3> | func bit<2> | func bit<1> | func bit<0> | ALUctr<1> |
|--------------|--------------|--------------|-------------|-------------|-------------|-------------|-----------|
| 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 |

$$° (\text{ALUctr}<1> = \neg \text{ALUop}<2> \& \neg \text{ALUop}<1> + \text{ALUop}<2> \& \neg \text{func}<2> \& \neg \text{func}<0>$$

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 |

$$\begin{aligned} \text{ALUctr<0>} &= \text{!ALUop<2>} \& \text{ALUop<1>} \\ &+ \text{ALUop<2>} \& \text{!func<3>} \& \text{func<2>} \& \text{!func<1>} \& \text{func<0>} \\ &+ \text{ALUop<2>} \& \text{func<3>} \& \text{!func<2>} \& \text{func<1>} \& \text{!func<0>} \end{aligned}$$

The ALU Control Block



$$\begin{aligned} \text{ALUctr<2>} &= \text{!ALUop<2>} \& \text{ALUop<0>} \& \text{ALUop<2>} \& \text{!func<2>} \& \text{func<1>} \& \text{!func<0>} \\ \text{ALUctr<1>} &= \text{!ALUop<2>} \& \text{!ALUop<1>} \& \text{ALUop<2>} \& \text{!func<2>} \& \text{!func<0>} \\ \text{ALUctr<0>} &= \text{!ALUop<2>} \& \text{ALUop<1>} \& \text{ALUop<2>} \& \text{!func<3>} \& \text{func<2>} \& \text{!func<1>} \& \text{func<0>} \\ &+ \text{ALUop<2>} \& \text{func<3>} \& \text{!func<2>} \& \text{func<1>} \& \text{!func<0>} \end{aligned}$$

Cal

CS 152 L07 Single Cycle 2 (37)

Patterson Fall 2003 © UCB

Cal

CS 152 L07 Single Cycle 2 (38)

Patterson Fall 2003 © UCB

Step 5: Logic for each control signal

$$\begin{aligned} \text{PCSrc} &<= (\text{OP} == \text{'BEQ'}) ? \text{'Br'} : \text{'plus4'}; \\ \text{ALUsrc} &<= (\text{OP} == \text{'Rtype'}) ? \text{'regB'} : \text{'immed'}; \\ \text{ALUctr} &<= (\text{OP} == \text{'Rtype'}) ? \text{func} : \\ &(\text{OP} == \text{'ORI'}) ? \text{'ORfunction'} : \\ &(\text{OP} == \text{'BEQ'}) ? \text{'SUBfunction'} : \\ &\text{'ADDfunction'}; \\ \text{ExtOp} &<= \underline{\hspace{2cm}}; \\ \text{MemWr} &<= \underline{\hspace{2cm}}; \\ \text{MemtoReg} &<= \underline{\hspace{2cm}}; \\ \text{RegWr} &<= \underline{\hspace{2cm}}; \\ \text{RegDst} &<= \underline{\hspace{2cm}}; \end{aligned}$$

Cal

CS 152 L07 Single Cycle 2 (39)

Patterson Fall 2003 © UCB

Step 5: Logic for each control signal

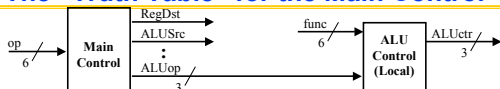
$$\begin{aligned} \text{PCSrc} &<= (\text{OP} == \text{'BEQ'}) ? \text{'Br'} : \text{'plus4'}; \\ \text{ALUsrc} &<= (\text{OP} == \text{'Rtype'}) ? \text{'regB'} : \text{'immed'}; \\ \text{ALUctr} &<= (\text{OP} == \text{'Rtype'}) ? \text{func} : \\ &(\text{OP} == \text{'ORI'}) ? \text{'ORfunction'} : \\ &(\text{OP} == \text{'BEQ'}) ? \text{'SUBfunction'} : \\ &\text{'ADDfunction'}; \\ \text{ExtOp} &<= (\text{OP} == \text{'ORI'}) ? \text{'ZEROextend'} : \text{'SIGNextend'}; \\ \text{MemWr} &<= (\text{OP} == \text{'Store'}) ? 1 : 0; \\ \text{MemtoReg} &<= (\text{OP} == \text{'Load'}) ? 1 : 0; \\ \text{RegWr} &<= ((\text{OP} == \text{'Store'}) \|\| (\text{OP} == \text{'BEQ'})) ? 0 : 1; \\ \text{RegDst} &<= ((\text{OP} == \text{'Load'}) \|\| (\text{OP} == \text{'ORI'})) ? 0 : 1; \end{aligned}$$

Cal

CS 152 L07 Single Cycle 2 (40)

Patterson Fall 2003 © UCB

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 |
| PCSrc | 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 |

Cal

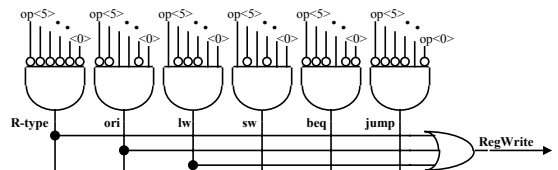
CS 152 L07 Single Cycle 2 (41)

Patterson Fall 2003 © UCB

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 |

$$\begin{aligned} \text{RegWrite} &= \text{R-type} + \text{ori} + \text{lw} \\ &= \text{!op<5>} \& \text{!op<4>} \& \text{!op<3>} \& \text{!op<2>} \& \text{!op<1>} \& \text{!op<0>} \quad (\text{R-type}) \\ &+ \text{!op<5>} \& \text{!op<4>} \& \text{op<3>} \& \text{op<2>} \& \text{!op<1>} \& \text{op<0>} \quad (\text{ori}) \\ &+ \text{op<5>} \& \text{!op<4>} \& \text{!op<3>} \& \text{!op<2>} \& \text{op<1>} \& \text{op<0>} \quad (\text{lw}) \end{aligned}$$

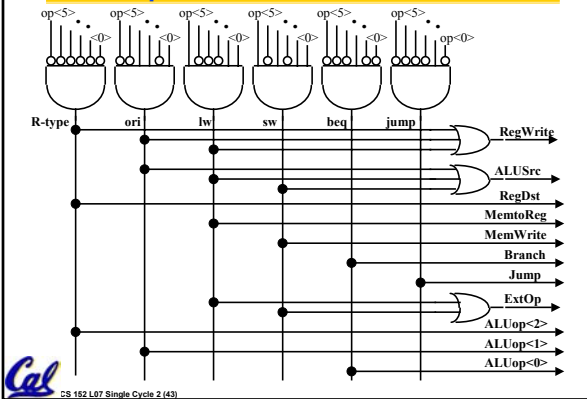


Cal

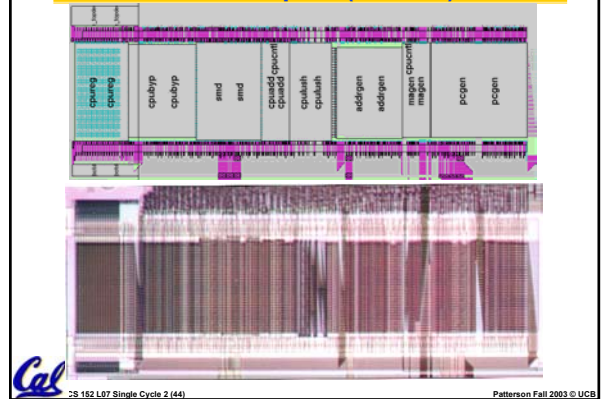
CS 152 L07 Single Cycle 2 (42)

Patterson Fall 2003 © UCB

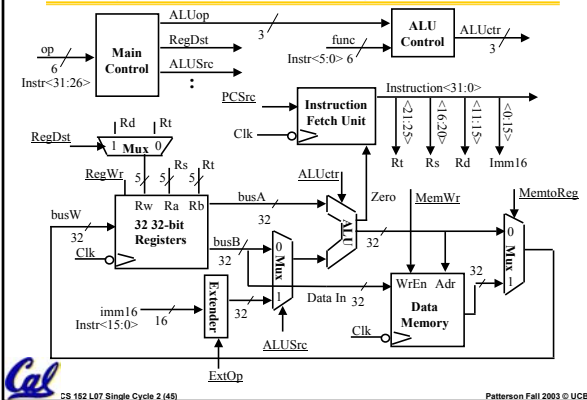
PLA Implementation of the Main Control



A Real MIPS Datapath (CNS T0)



Putting it All Together: A Single Cycle Processor



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 for load is much longer than needed for all other instructions

Preview

Next Time:

MultiCycle Data Path

- CPI ≥ 1 , CycleTime much shorter (~1/5 of time)

Summary

- Single cycle datapath => CPI=1, CCT => long

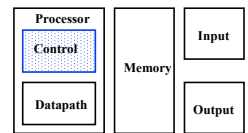
5 steps to design a processor

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



Where to get more information?

- **Chapter 5.1 to 5.4 of your text book:**
 - David Patterson and John Hennessy, "Computer Organization & Design: The Hardware / Software Interface," Third Edition, Morgan Kaufman Publishers, San Mateo, California, 2003.
- **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, Joe Heinrich "MIPS RISC Architecture," Prentice Hall, 2nd edition, 1992

