

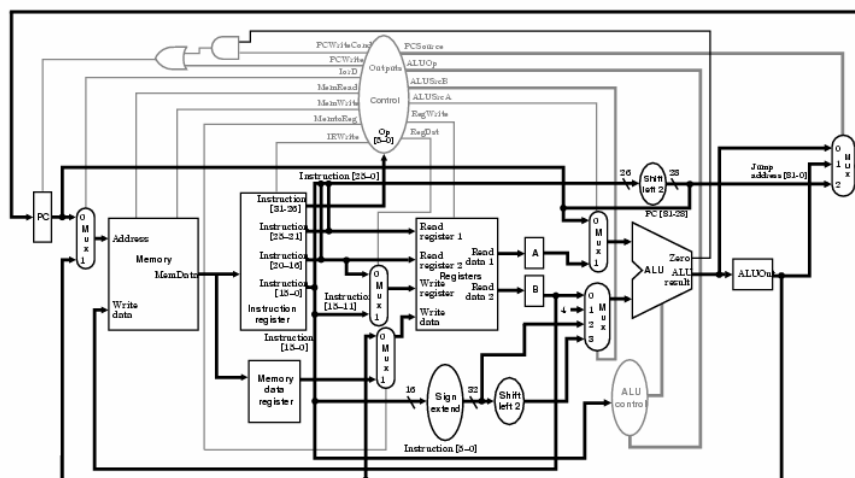
# EEC 485 High Performance Arch. (Fall 2006)

## Chapter 6.3 Pipelined Control

Chansu Yu

Cleveland State University

### Control Signals in Multicycle Implementation



## Multicycle MIPS: Control Signals

### □ Write signals

- IRWrite: Memory output is written into the IR
- PCWrite: PC is written
- PCWriteCond: PC is written if condition is met (“Zero” output from ALU)
- RegWrite : what happen “add \$s0, \$s0, \$s2” ???
- MemWrite / MemRead

### □ ALU control

- ALUOp0
- ALUOp1

## Multicycle MIPS: Control Signals

### □ More multiplexor selectors

- RegDst : destination reg. no. for register file
- MemtoReg : destination reg. data for reg. file
- IorD: Read instruction or data *because memory is shared*
- ALUSrcA: 1<sup>st</sup> ALU operand; PC or register data
- ALUSrcB (ALUSrc is expanded): 2<sup>nd</sup> ALU operand; register data or 16-bit offset => register data, 16-bit offset, 4, or 16-bit offset<<2 *because ALU is shared*
- PCSource (PCSrc is expanded):  
PC+4 or PC+offset =>  
PC+4, PC+offset, (PC[31-28]||IR[25-0]<<2)  
(this is for “j” instruction)

## Pipelined MIPS: Do we need?

### □ Write signals

- IRWrite: No (Pipeline register replaces IR)
- PCWrite: No (In multicycle MIPS, PCWrite was needed because PC is written depending on the condition at EX cycle even though it is always written at IF cycle. In pipelined MIPS, PC is always written at every cycle.)
- PCWriteCond: No
- RegWrite: Yes (Activated if the instruction is R-type or lw.),  
Used at WB stage
- MemWrite / MemRead: Yes (But only for data memory),  
Used at MEM stage

## Pipelined MIPS: Do we need?

### □ ALU control:

- Two other ALUs perform addition only  
(PC+4 and PC+IR<<2 at IF and EX stages)
- ALUOp1/0: Yes (10 for R-type, 00 for lw/sw, 01 for beq),  
Used at EX stage

## Pipelined MIPS: Control Signals

### Multiplexor selectors

- RegDst: Yes, Used at EX stage
- MemtoReg: Yes, Used at WB stage
- IorD: No (Separate instruction and data memory)
  
- ALUSrcA: No (Three ALUs make the control simpler)
- ALUSrcB: Replaced by ALUSrc as in single cycle implementation (0 for R-type/beq, 1 for lw/sw), Used at EX stage
- PCSrc: Replaced by PCSrc as in single cycle implementation (PCSrc = Branch && Zero flag), Used at MEM stage (after EX stage)

## Summary: 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)

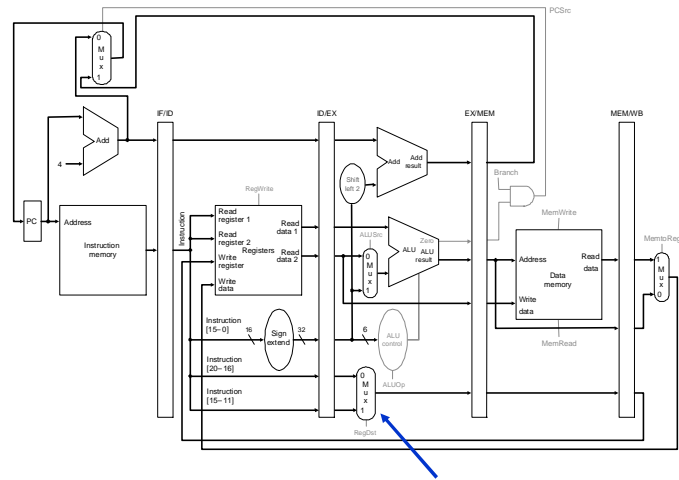
### At MEM stage

- 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)

# Pipeline Control

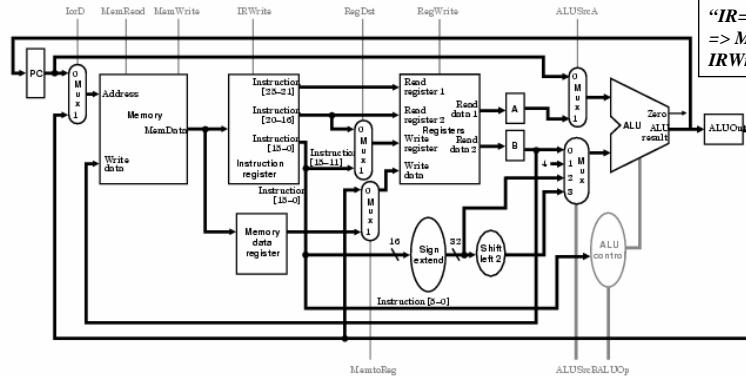


# Pipeline Control

- ❑ How to manage the control signals?
  - When are they generated?
    - Generated at each stage based on the opcode?
      - Opcode must be passed through the stages
    - Generate all at once and passed through the stages?
      - At which stage can they be generated?
  
- ❑ Let's take a look at the multicycle MIPS & single-cycle MIPS
  - Pipelined MIPS is more like a single-cycle MIPS

# Multicycle MIPS => Pipelined MIPS

Control Signals Generated When Executing “add \$1, \$2, \$3”



**Cycle1:**  
 “PC=PC+4”  
 => ALUSrcA=0,  
 ALUSrcB=1, ALUOp=00,  
 PCSource=0, PCWrite  
 “IR=Memory[PC]”  
 => MemRead, IorD=0,  
 IRWrite

# Multicycle MIPS => Pipelined MIPS

Control Signals Generated When Executing “add \$1, \$2, \$3”

**Cycle2:**  
 “A=Reg[IR[25-21]]”  
 “B=Reg[IR[20-16]]”  
 “ALUOut=PC+Offset<<2”  
 => ALUSrcA=0,  
 ALUSrcB=3, ALUOp=00

**Cycle3:**  
 “ALUOut=A op B”  
 => ALUSrcA=1,  
 ALUSrcB=0, ALUOp=10

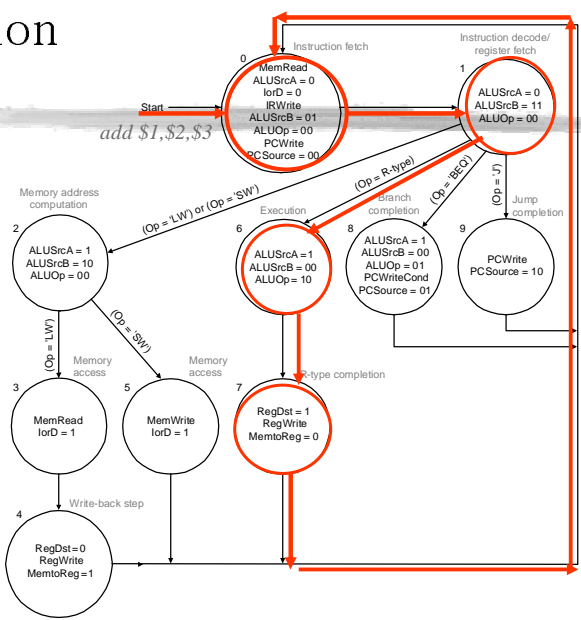
**Cycle4:**  
 “Reg[IR[15-11]]=ALUOut”  
 => RegDst=1, MemtoReg=0,  
 RegWrite

- At IF stage
- At ID stage
- At EX stage
  - RegDst: 1 (R-type)
  - ALUSrc: 0 (R-type)
  - ALUOp1/0: 10 (R-type)
- At MEM stage
  - Branch: 0
  - MemWrite / MemRead: 0
  - PCSrc: 0 (automatically generated)
- At WB stage
  - RegWrite: 1 (R-type)
  - MemtoReg: 0 (R-type)

# Implementation of FSM

**add instruction:**

-how many cycles?  
 -after three cycles, it is changed to "state 7" and outputs three control signals (RegDst=1, RegWrite, MemtoReg=0) and the next state is "state 0"



## Single-cycle => Pipelined MIPS

Control Signals Generated When Executing "add \$1, \$2, \$3"

4 multiplexor selectors      2 write signals      2 ALU controls

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	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 the difference in pipelined MIPS?

- add IF ID EX MEM WB
- lw IF ID EX MEM WB

*RegDst=1*

❑ Control signals should also be conflict-free

*RegDst=0*

❑ Should we use a finite state machine?

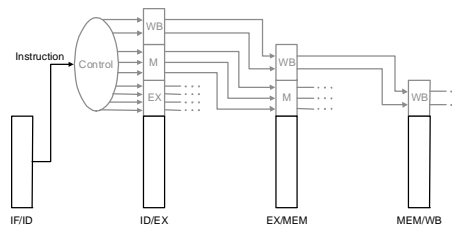
❑ Our approach is

- Generate control signals all at once at ID stage
- And passed them through stages

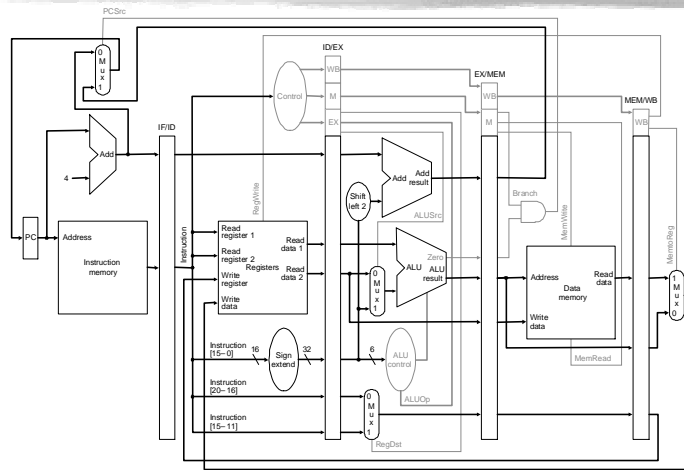
# Pipeline Control

❑ 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



# Datapath with Control



# Example

□ Five instructions go through the MIPS pipeline:

lw	\$10, 20(\$1)
sub	\$11, \$2, \$3
and	\$12, \$4, \$5
or	\$13, \$6, \$7
add	\$14, \$8, \$9



