

EEC 483 Computer Organization (Spring 2006)

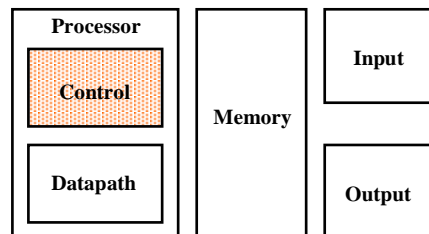
Chapter 5.4 A Simple Implementation Scheme

Chansu Yu

Cleveland State University

The Big Picture

❑ The Five Classic Components of a Computer



❑ Datapath & Control

Datapath and Control

□ Datapath

- Between memory, ALU, register file
- When a 32-bit instruction is ready
 - It is decoded to obtain opcode, register numbers, ...
 - Register numbers go to register file and the register values become ready
 - Immediate value becomes ready
 - Memory data is read and ready

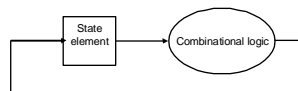
□ Control

- Need to decide which inputs should be used (multiplexors)
- ALU control such as operation, binvert, carryIn, ...
- Write signal (register write, memory write)

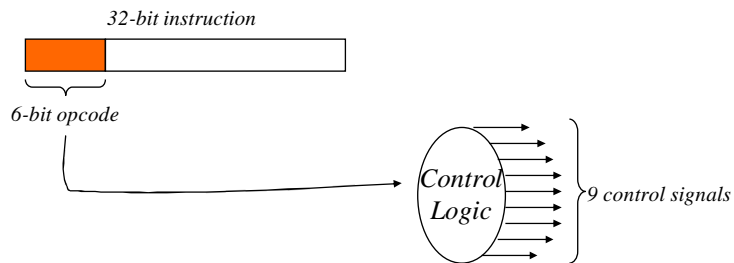
Write Control Signal

□ When does the contents of state element change?

- Edge triggered methodology
 - Every input to update state element must be stable at the active clock edge
 - Feedback cannot occur within the same clock cycle (no race)
- If the contents of state element change every clock cycle, it's OK.
- *If not ???*
 - *We need a write control signal that must be asserted for a write to occur at the clock edge. => RegWrite, MemRead, Mem Write*



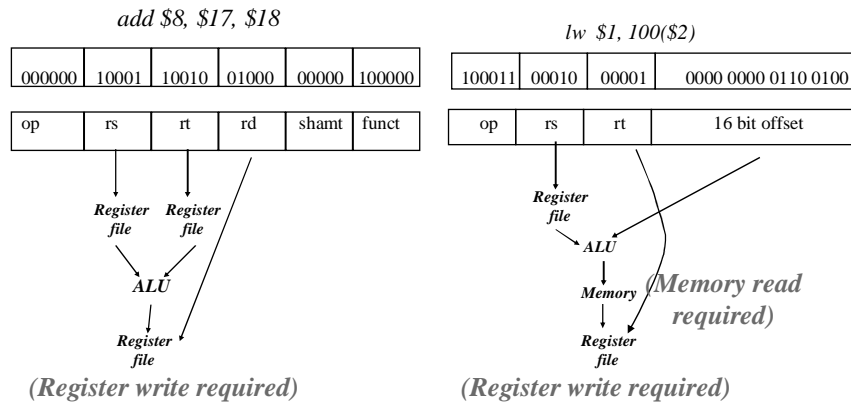
Generation of Control Signals



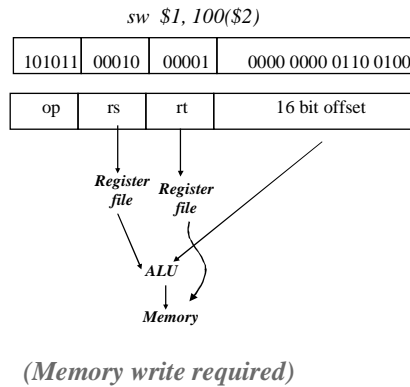
Control Signals (9)

- ❑ Multiplexor selector
 - ALUSrc : second operand for ALU
 - RegDst : destination reg. no. for register file
 - MemtoReg : destination reg. data for reg. file
 - PCSrc : ??? (the only exception that cannot be set based on the given instruction)
- ❑ ALU control
 - ALUOp0
 - ALUOp1
- ❑ Write signals
 - RegWrite : what happen "add \$s0, \$s0, \$s2" ???
 - MemWrite / MemRead

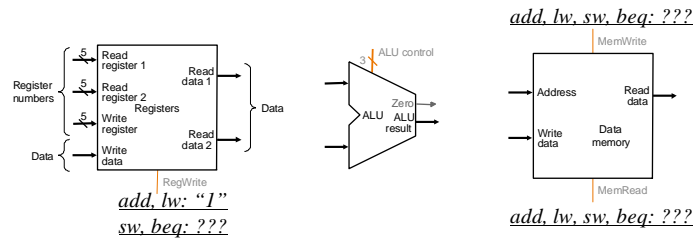
Write Signal



Write Signal



Write Signal



What happen before write value to \$1 is ready ?
 => need some time to be stabilized
 => this will be the longest path (critical path)
 => determines the clock cycle for the CPU

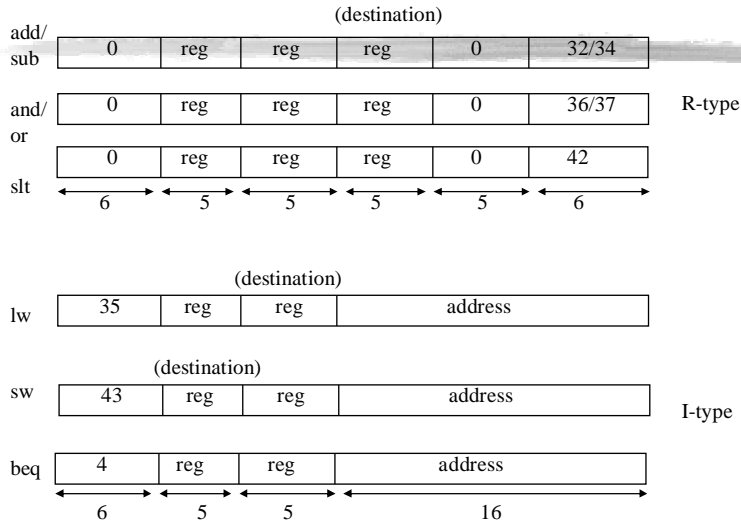
Control Signals (9)

3 write signals

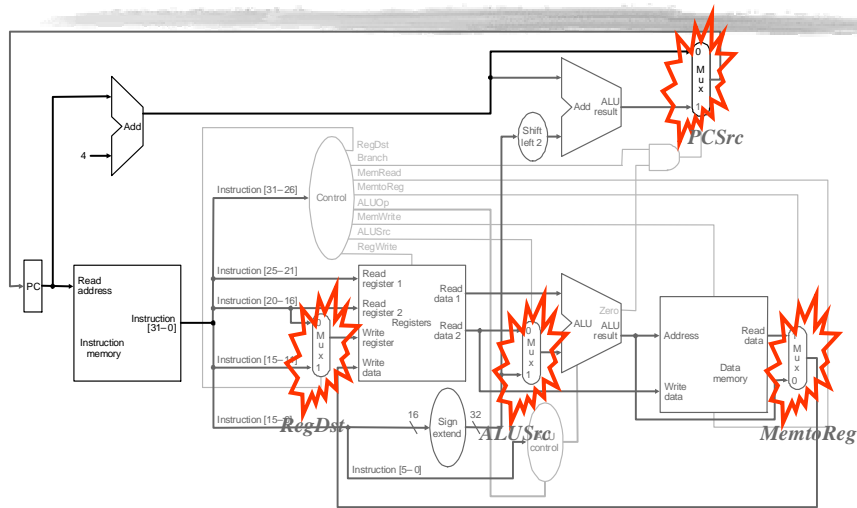
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	(PCSrc) Branch	ALUOp1	ALUOp0
000000 R-format	1	0	0	1	0	0	0	1	0
100011 lw	0	1	1	1	1	0	0	0	0
101011 sw	X	1	X	0	0	1	0	0	0
000100 beq	X	0	X	0	0	0	1	0	1

9 control signals

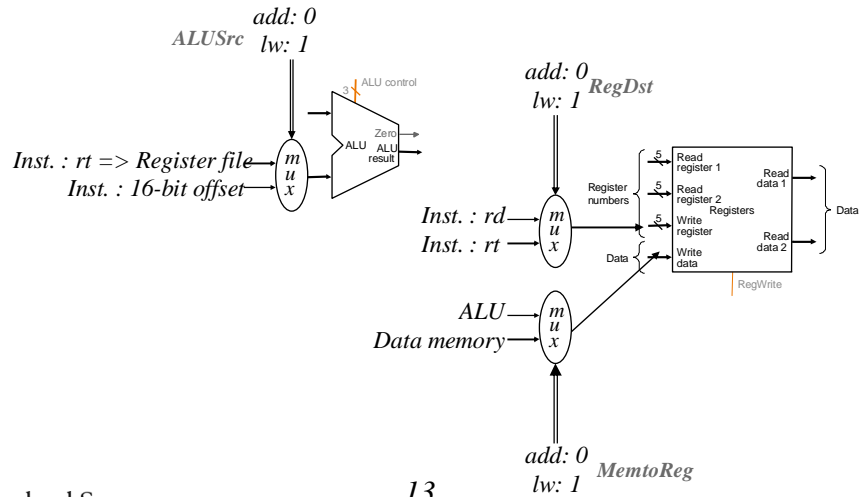
MIPS Instruction Types



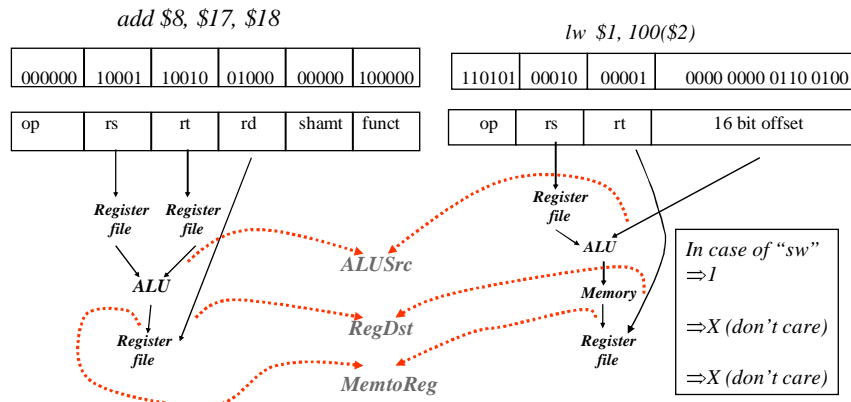
Multiplexer Selector



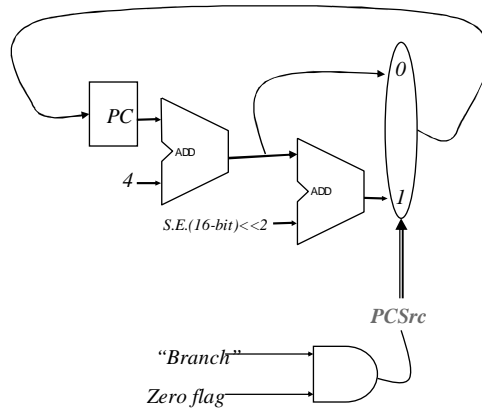
Multiplexor Selector



Multiplexor Selector



One More Multiplexor Selector



Control Signals (9)

4 multiplexor selectors

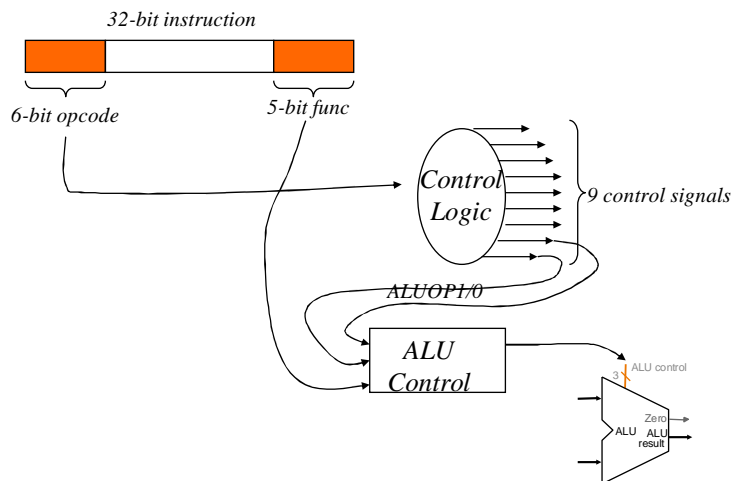
Instruction	RegDst	ALUSrc	Mem-Reg	Reg Write	Mem Read	Mem Write	(PCSrc) Branch	ALUOp1	ALUOp0
000000 R-format	1	0	0	1	0	0	0	1	0
100011 lw	0	1	1	1	1	0	0	0	0
101011 sw	X	1	X	0	0	1	0	0	0
000100 beq	X	0	X	0	0	0	1	0	1

9 control signals

ALU Control

- ❑ What should the ALU do with this instruction ?
 - Information comes from the 32 bits of the instruction
 - ALU's operation based on instruction type and function code
- ❑ Multi-level control (for simplifying control logic)
 - Instruction's opcode (bit31-bit26)
 - => ALUOp1 & ALUOp0
(with instruction's funct (bit4-bit0))
 - => ALU inputs: binvert (= carryin), operation

Generation of Control Signals



Supporting MIPS Instructions

- and/or/add/sub
- slt: subtract and output r0
based on "set" output from the 1-bit adder of the last-bit ALU
r1-r31 is "0" ("less" input)
- beq: subtract and output "zero" flag based on all "r" bits

Operation input

- 0: and
- 1: or
- 2: add/sub/beq
- 3: slt

Binvert input

- 0: and/or/add
- 1: sub/beq/slt

Cleveland State University 41 c.yu91@csuohio.edu

ALU Control

□ ALU control inputs

	binvert=carryin	operation	opcode
lw	0	0	2 100011 <= "add" for address calculation
sw	0	0	2 101011 <= "add" for address calculation
beq	1	1	2 000100 <= "sub" for comparison
add	0	0	2 000000
sub	1	1	2 000000
and	0	0	0 000000
or	0	0	1 000000
slt	1	1	3 000000

ALU Control: Multi-level

- 1st level control: Instruction's opcode (bit31-bit26) => ALUOp1 & ALUOp0
 - lw (100011) => 00
 - sw (101011) => 00
 - beq (000100) => 01
 - arithmetic (000000) => 10

ALUOp computed from instruction type (multiple levels of control to reduce the size of control unit)

- 2nd level control: ALUOp1 & ALUOp0 + funct (bit5-bit0) => binvert (=carryin), operation

ALUOp		Funct field						Operation	
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0		
0	0	X	X	X	X	X	X	010	"add"
X	1	X	X	X	X	X	X	110	"add"
1	X	X	X	0	0	0	0	010	"add" (100000)
1	X	X	X	0	0	1	0	110	"sub" (100010)
1	X	X	X	0	1	0	0	000	"and" (100100)
1	X	X	X	0	1	0	1	001	"or" (100101)
1	X	X	X	1	0	1	0	111	"slt" (101010)

lw/sw
beq

arithmetic

21

Implement with gates based on this truth table e.g. bit0 = $A1(F3 + F0)$ since there is only one "1" for F3 and F0

Cleveland State University

c.yu91@csuohio.edu

Control

- Must describe hardware to compute 4-bit ALU control input
 - given instruction type
 - 00 = lw, sw
 - 01 = beq,
 - 10 = arithmetic
 - function code for arithmetic

ALUOp computed from instruction type

- Describe it using a truth table (can turn into gates):

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

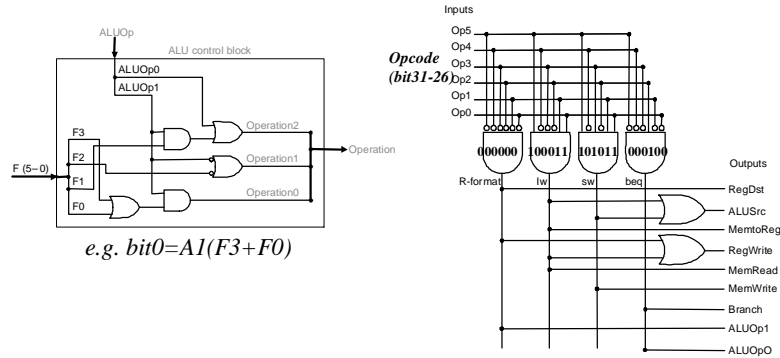
FIGURE 5.13 The truth table for the three ALU control bits (called Operation). The inputs are the ALUOp and function code field. Only the entries for which the ALU control is asserted are shown. Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so the truth table can contain entries 1X and X1, rather than 10 and 01. Also, when the function field is used, the first two bits (F5 and F4) of these instructions are always 10, so they are don't-care terms and are replaced with XX in the truth table.

Cleveland State University

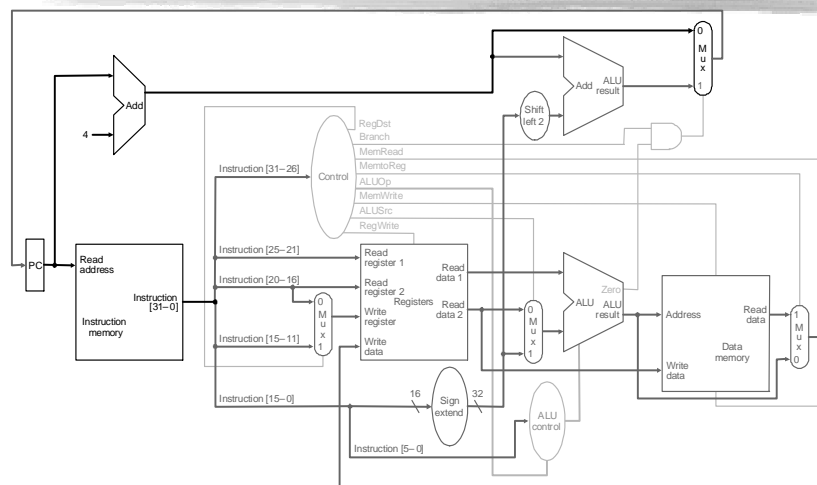
c.yu91@csuohio.edu

Control Signals (9)

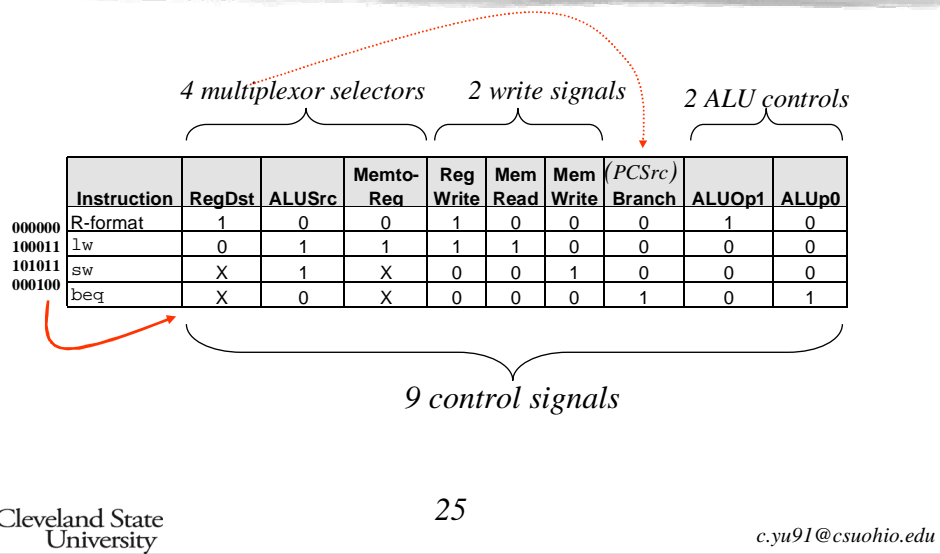
Simple combinational logic (truth tables)



Control Signals (9)

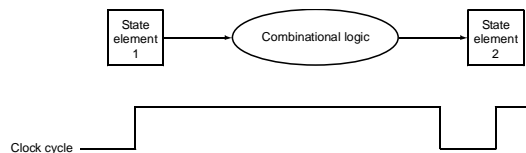


Control Signals (9)



Our Simple Control Structure

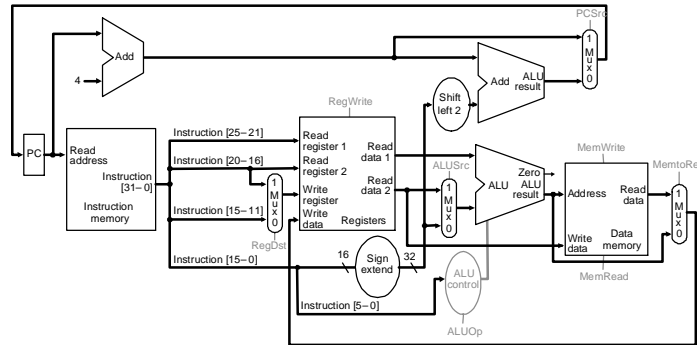
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
 - ALU might not produce “right answer” right away
 - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



We are ignoring some details like setup and hold times

Single Cycle Implementation

- ❑ Calculate cycle time assuming negligible delays except:
 - memory (2ns), ALU and adders (2ns), register file access (1ns)



Where we are headed

- ❑ Single Cycle Problems:
 - what if we had a more complicated instruction like floating point?
 - wasteful of area (duplicated function blocks)
- ❑ One Solution:
 - use a “smaller” cycle time
 - have different instructions take different numbers of cycles
 - a “multicycle” datapath:

