

EEC 483 Computer Organization (Fall 2001)

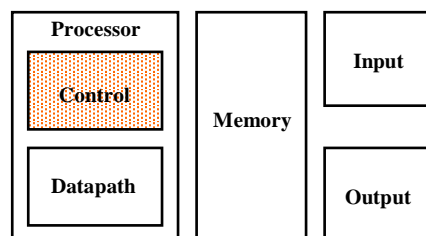
Chapter 5.3 A Simple Implementation Scheme

Chansu Yu

Cleveland State University

The Big Picture

□ The Five Classic Components of a Computer



□ Datapath & Control

Datapath and Control

□ Datapath

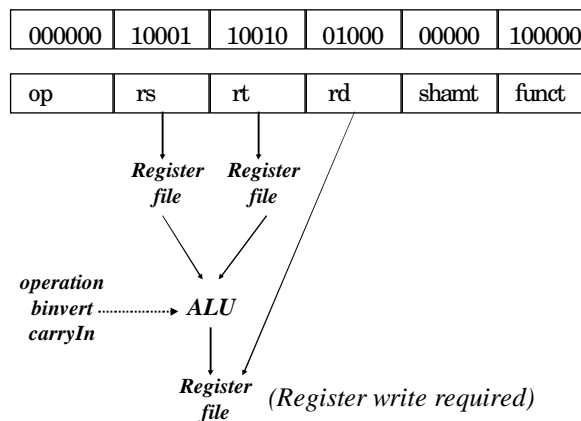
- Between memory, ALU, register file
- When a 32-bit instruction is ready
 - It is decoded to obtain opcode, register numbers, ...
 - Register numbers go to register file and the register values become ready
 - Immediate value becomes ready
 - Memory data is read and ready

□ Control

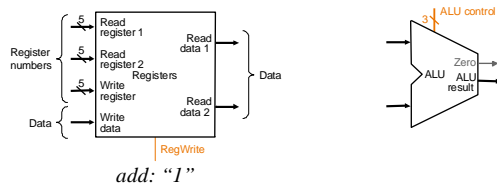
- Need to decide which inputs should be used (multiplexors)
- ALU control such as operation, binvert, carryIn, ...
- Write signal (register write, memory write)

ALU Control, Write Signal

add \$8, \$17, \$18



ALU Control, Write Signal



Some instructions do not write into register file ...

=> write signal required

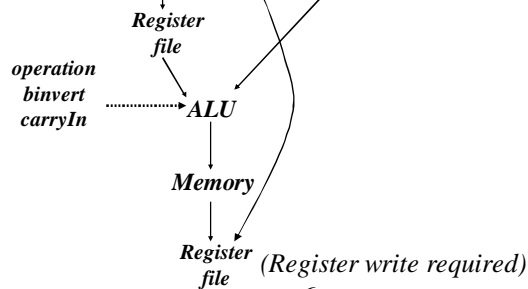
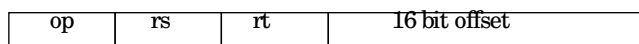
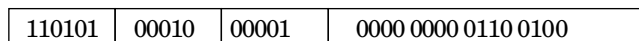
=> read signal required ???

What happen before write value to \$8 is ready ?

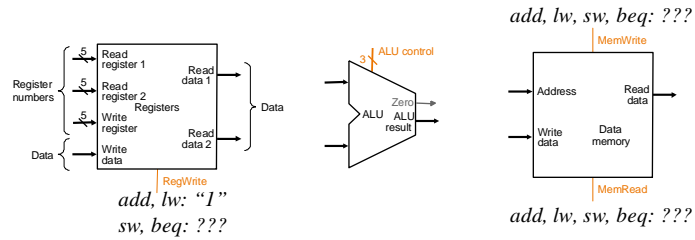
=> need some time to be stabilized (clock)

ALU Control, Write Signal

lw \$1, 100(\$2)

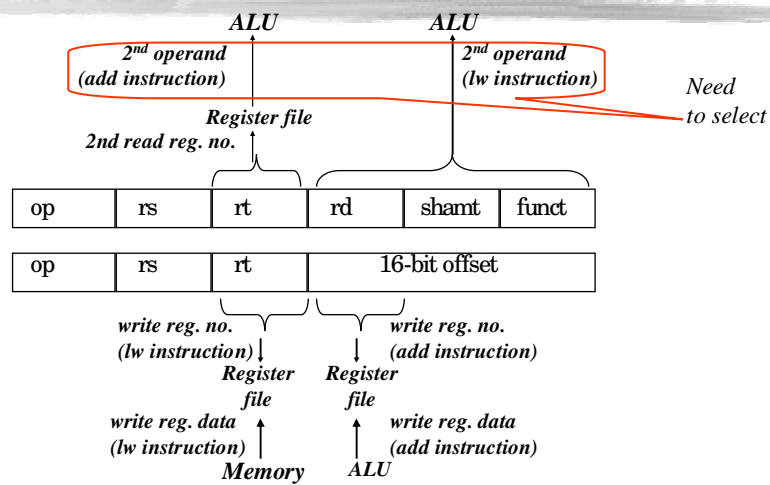


ALU Control, Write Signal

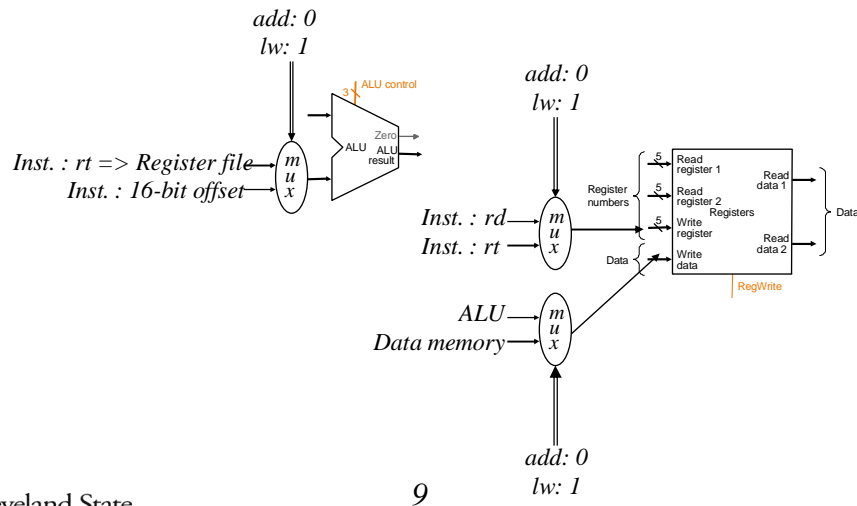


- What happen before write value to \$1 is ready ?**
- => need some time to be stabilized**
- => this will be the longest path (critical path)**
- => determines the clock cycle for the CPU**

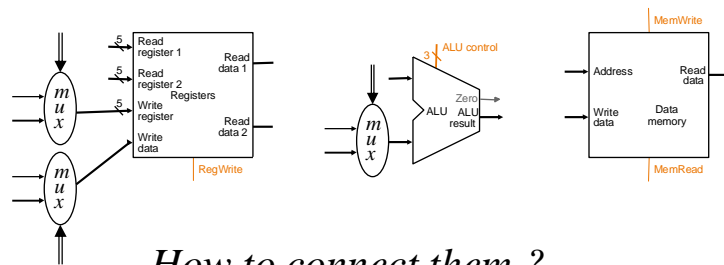
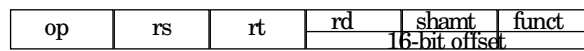
Multiplexor Select Input



Multiplexor Select Input



Control Signals

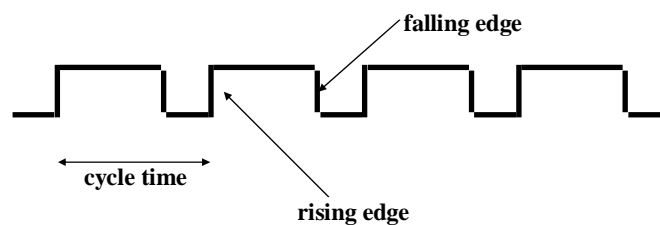


Control Signals (9)

- ❑ 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

State Elements

- ❑ Unlocked vs. Clocked
- ❑ Clocks used in synchronous logic
 - when should an element that contains state be updated?



Latches and Flip-flops

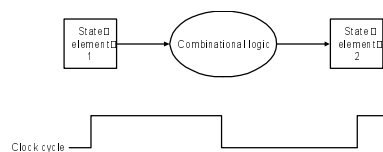
- ❑ Output is equal to the stored value inside the element
(don't need to ask for permission to look at the value)
- ❑ Change of state (value) is based on the clock
- ❑ Latches: whenever the inputs change, and the clock is asserted
- ❑ Flip-flop: state changes only on a clock edge
(edge-triggered methodology)

"logically true",
— could mean electrically low

A clocking methodology defines when signals can be read and written
— wouldn't want to read a signal at the same time it was being written

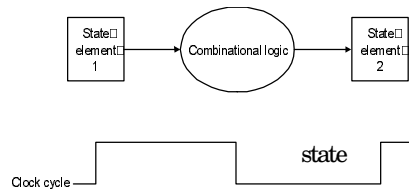
Our Implementation

- ❑ 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



Our Implementation

- ❑ An 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)
- ❑ Typical execution (ex: `add $s0, $s0, $s2`):
 - read contents of some state elements (`$s0, $s2`)
 - send values through some combinational logic (`adder in ALU`)
 - write results to one or more elements (`$s0`)



ALU Control

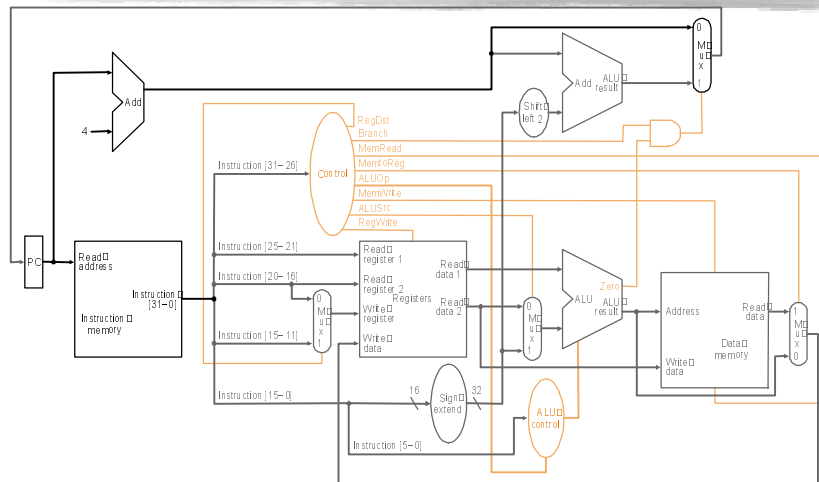
- ❑ What should the ALU do with this instruction ?
 - Information comes from the 32 bits of the instruction
 - ALU's operation based on instruction type and function code
- ❑ ALU control input
 - 000 AND
 - 001 OR
 - 010 add <- add, load, store
 - 110 subtract <- sub, branch
 - 111 set-on-less-than
- ❑ Why is the code for subtract 110 and not 011?

ALU Control

- ❑ Must describe hardware to compute 3-bit ALU control input
 - given instruction type
 - 00 = lw, sw
 - 01 = beq,
 - 11 = arithmetic
 - function code for arithmetic
- } **ALUOp computed from instruction type (multiple levels of control to reduce the size of control unit)**
- ❑ 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	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Control Signals (9)



Control Signals (9)

4 multiplexor selectors
(PCSrc is over there...)

2 write signals

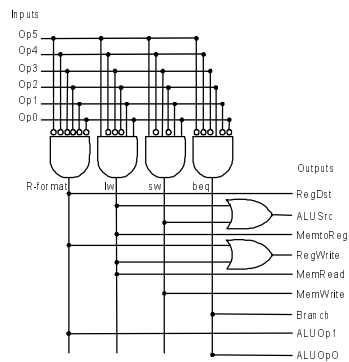
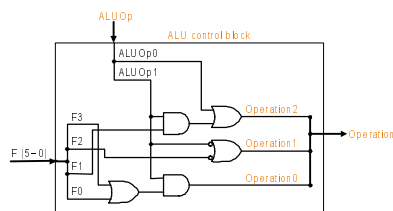
2 ALU controls

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

9 control signals

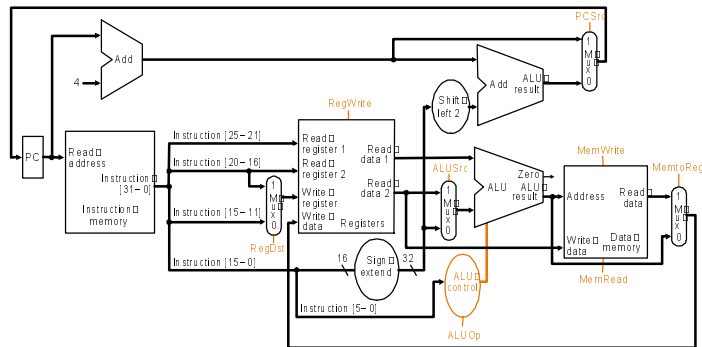
Control Signals (9)

Simple combinational logic (truth tables)



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:

