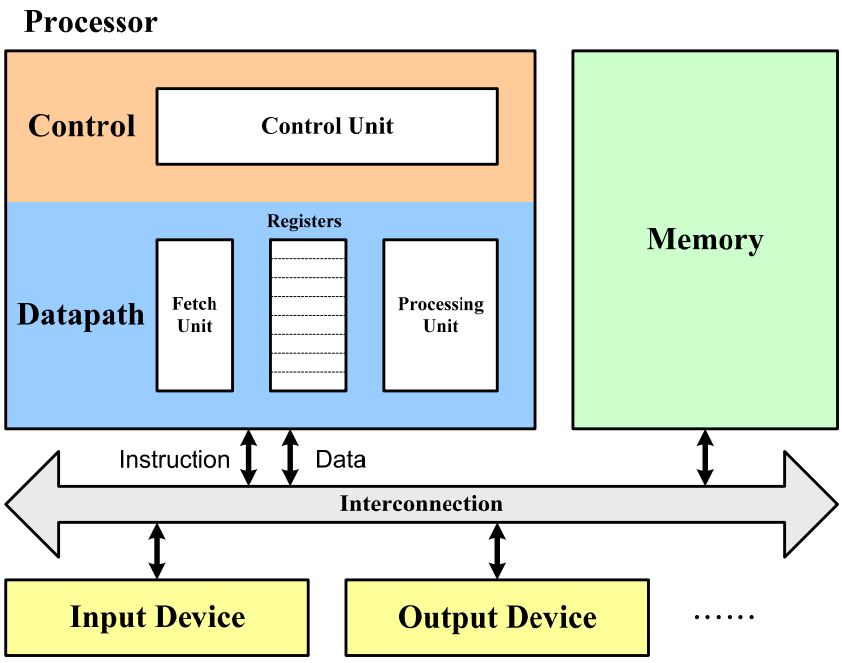
Chapter 4
The Processor (Part 1)

王振傑 (Chen-Chieh Wang)
ccwang@mail.ee.ncku.edu.tw

Department of Electrical Engineering, Feng-Chia University

The Processor : Datapath and Control



Outline

- Introduction (4.1)
- Logic Design Conventions (4.2, C.7, C.8, C.11)
- Building a Datapath (4.3)
- A Simple Implementation Scheme (4.4, D.2)

Department of Electrical Engineering, Feng-Chia University

The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: *lw*, *sw*
  - arithmetic-logical instructions: *add*, *sub*, *and*, *or*, *slt*
  - control flow instructions: *beq*, *j*
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers

  Why? memory-reference? arithmetic? control flow?

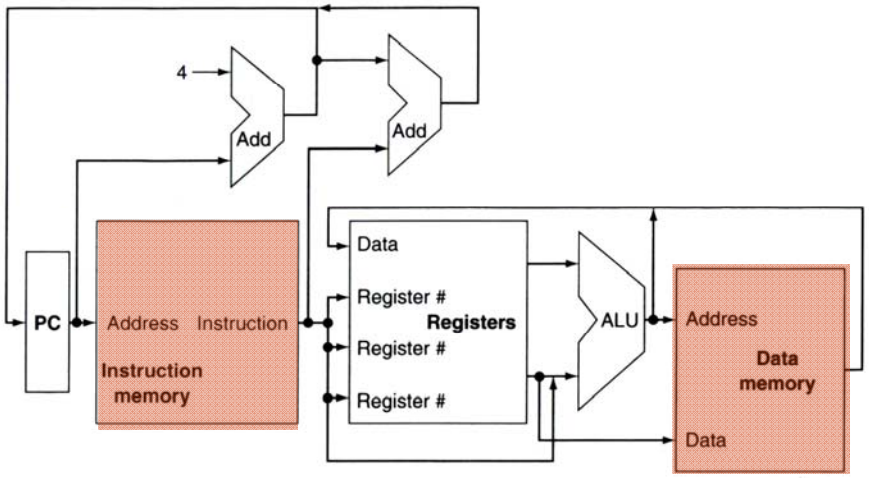
Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

Department of Electrical Engineering, Feng-Chia University

More details: an abstract view

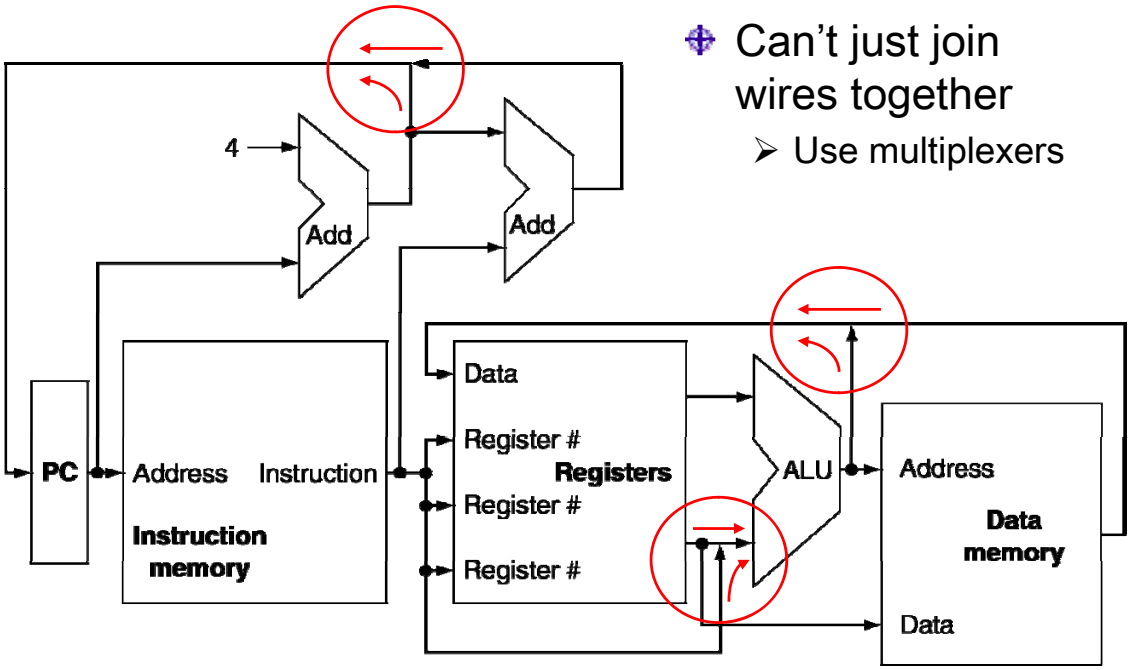
- Abstract / Simplified View:



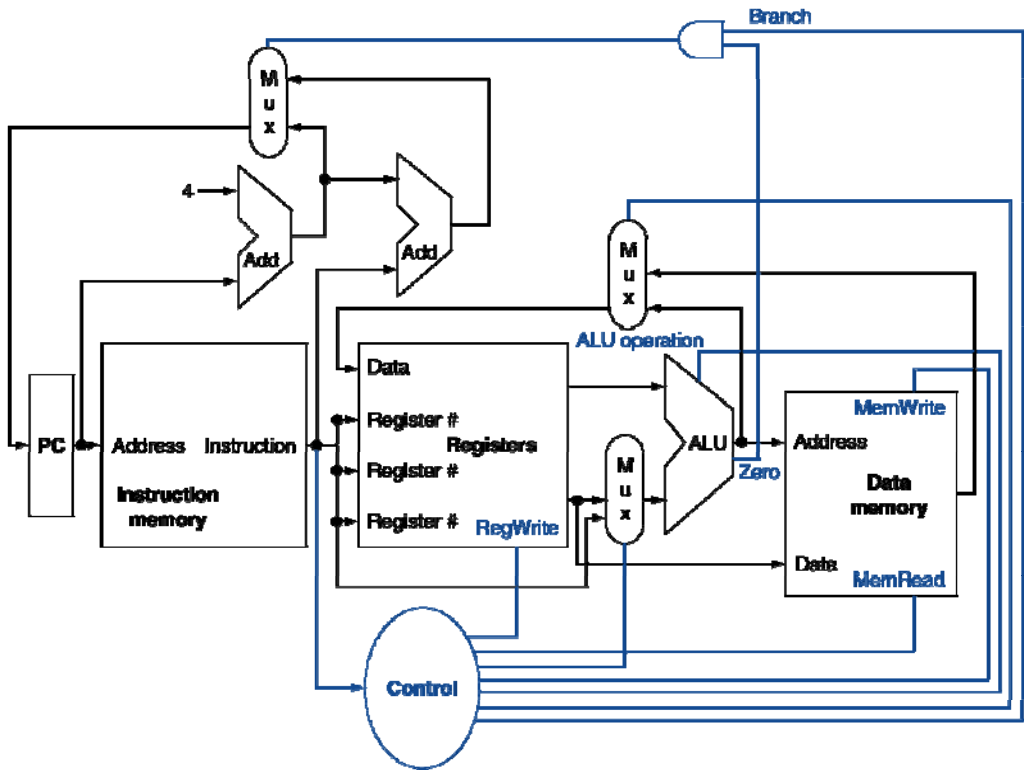
- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (sequential)

Multiplexers

- Can't just join wires together
  - Use multiplexers



Control



Outline

- Introduction (4.1)
- Logic Design Conventions (4.2, C.7, C.8, C.11)
- Building a Datapath (4.3)
- A Simple Implementation Scheme (4.4, D.2)

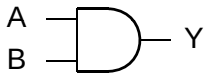
Department of Electrical Engineering, Feng-Chia University

Logic Design Basics

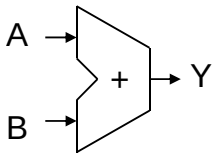
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- Sequential (state) elements
  - Store information

Combinational Elements

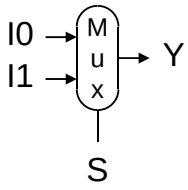
- AND-gate
  - Y = A & B



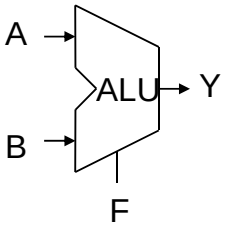
- Adder
  - Y = A + B



- Multiplexer
  - Y = S ? I1 : I0



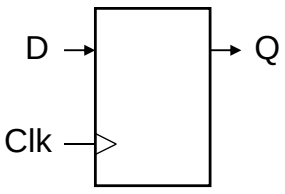
- Arithmetic/Logic Unit
  - Y = F(A, B)

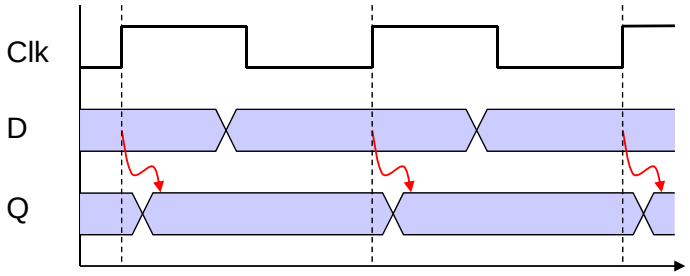


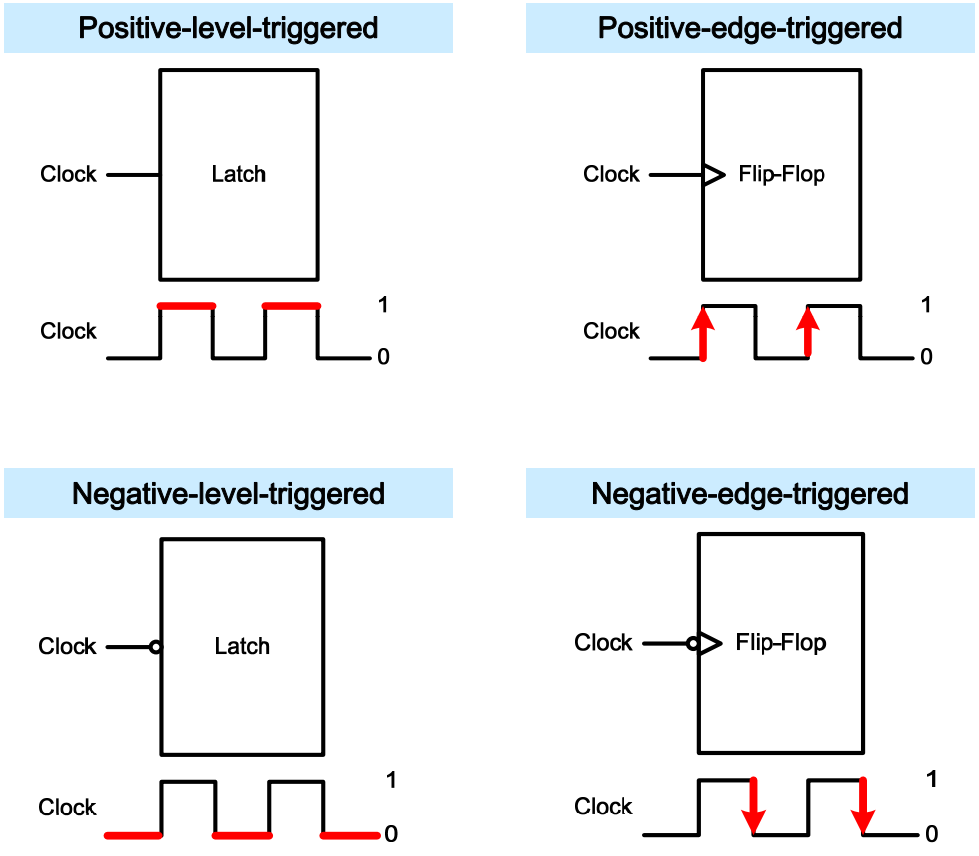
Department of Electrical Engineering, Feng-Chia University

Sequential Elements

- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1



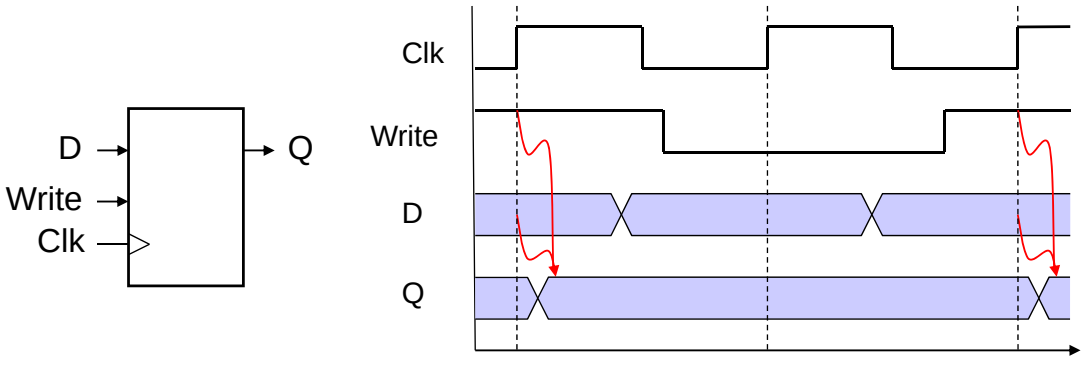




Department of Electrical Engineering, Feng-Chia University

Sequential Elements

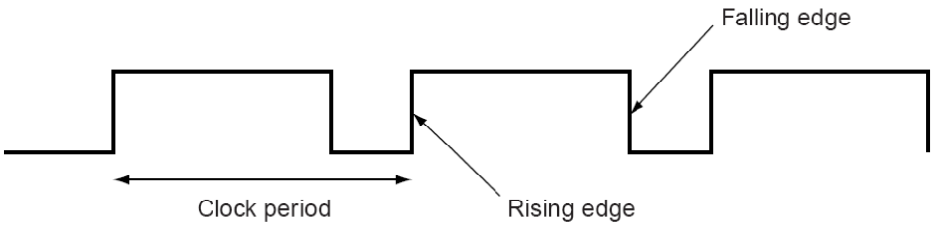
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



Text Book : C48

Sequential Elements

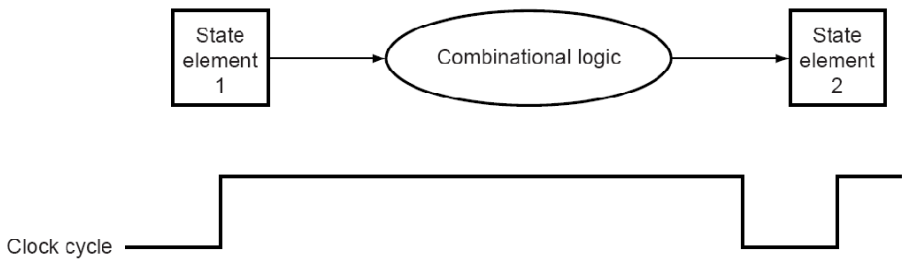
- Clock methodology
  - Define when signals can be read and when they can be written.
- Unclocked vs. Clocked
- Clocks used in synchronous logic
  - when should an element that contains state be updated?



Text Book : C49 ~ C50

Edge-Triggered Methodology

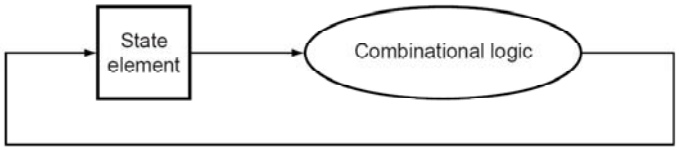
- Typical execution:
  - read contents of some state elements,
  - send values through some combinational logic
  - write results to one or more state elements

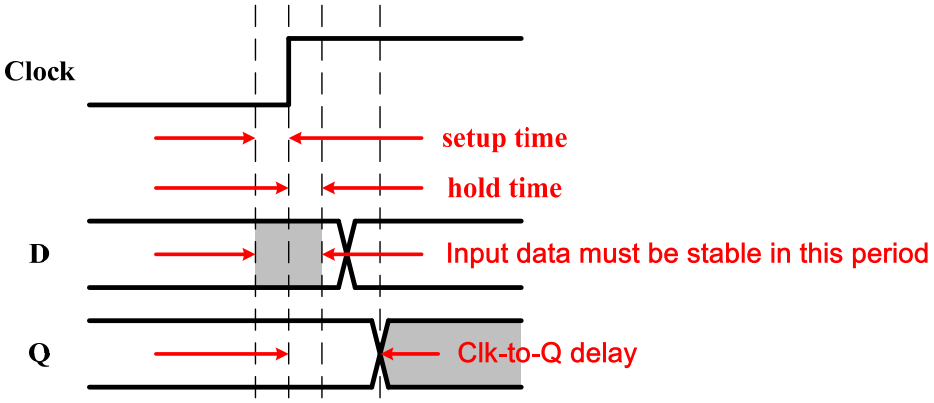


Text Book : C49 ~ C50

Edge-Triggered Methodology

- Allow a state element to be read and written in the same clock cycle without creating a race that could lead to undetermined data values.

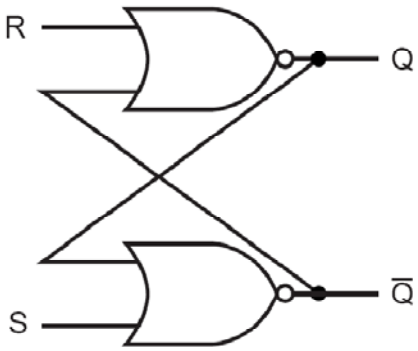




Text Book : C50 ~ C51

An unclocked state element

- The Set-Reset latch
  - output depends on present inputs and also on past inputs



S	R	Q_n	Q_{n+1}	Function
0	0	0		hold (keep value)
0	0	1		
0	1	0		0
0	1	1		
1	0	0		1
1	0	1		
1	1	0		unstable
1	1	1		

Latches and Flip-flops

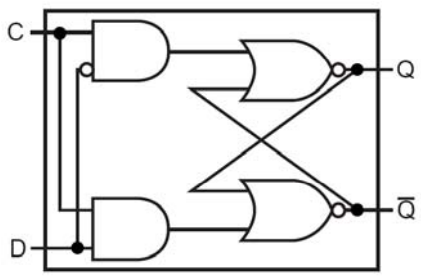
- Output is equal to the stored value inside the element
  (don't need to ask for permission to look at the value)
- Change of state (value) is based on the clock
- Latches: whenever the inputs change, and the clock is asserted
- Flip-flop: state changes only on a clock edge
  (edge-triggered methodology)

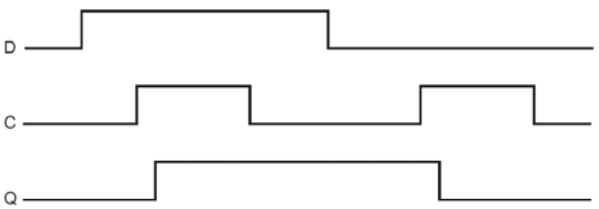
"logically true",
— could mean electrically low

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

D-latch

- Two inputs:
  - the data value to be stored (D)
  - the clock signal (C) indicating when to read & store D
- Two outputs:
  - the value of the internal state (Q) and it's complement

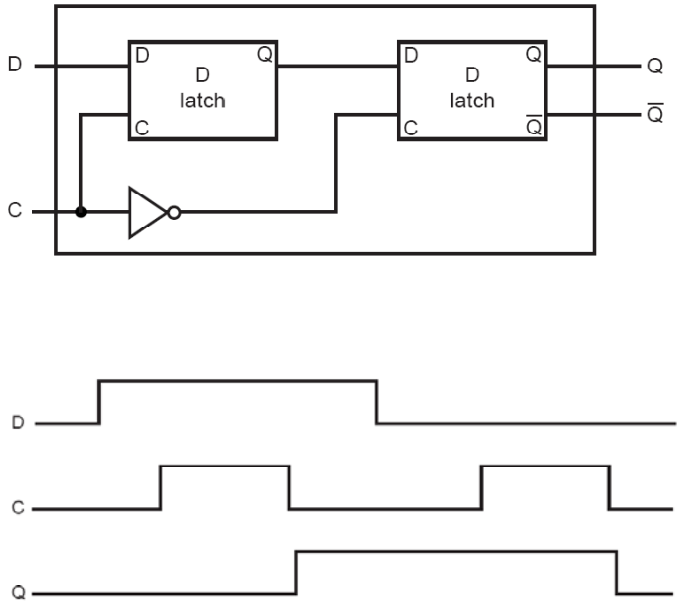




Text Book : C53

D flip-flop

- Output changes only on the clock edge

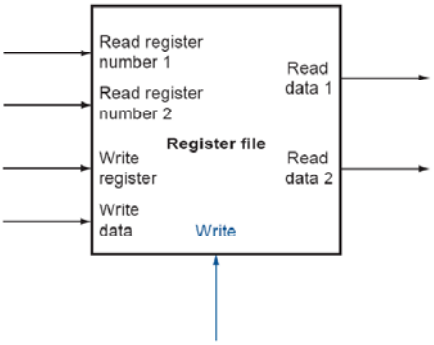


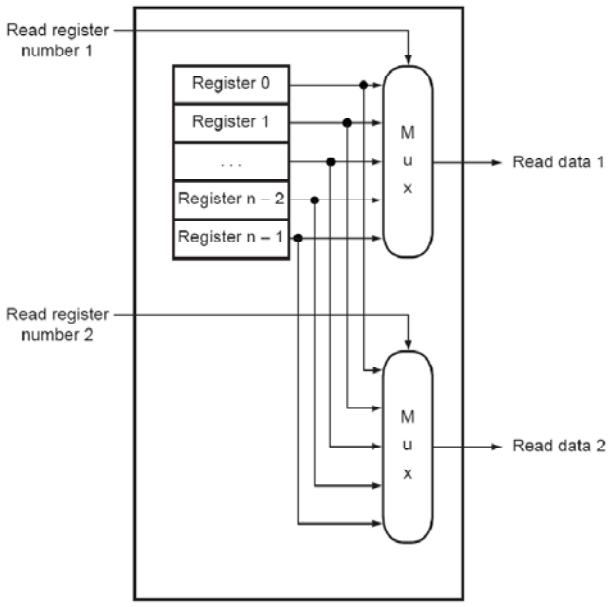
Department of Electrical Engineering, Feng-Chia University

Text Book : C54

Register File (Read)

- Built using D flip-flops



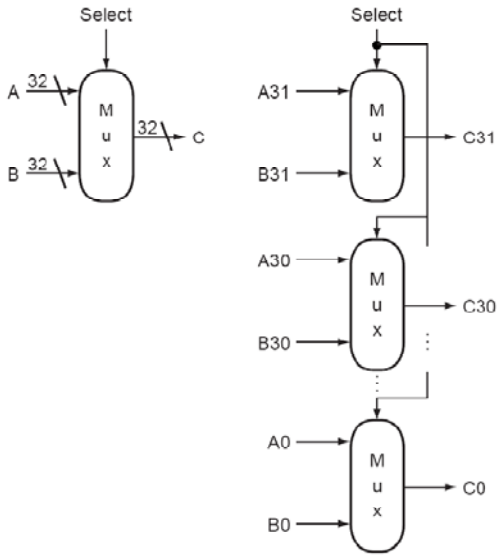


Do you understand? What is the "Mux" above?

Text Book : C55

Abstraction

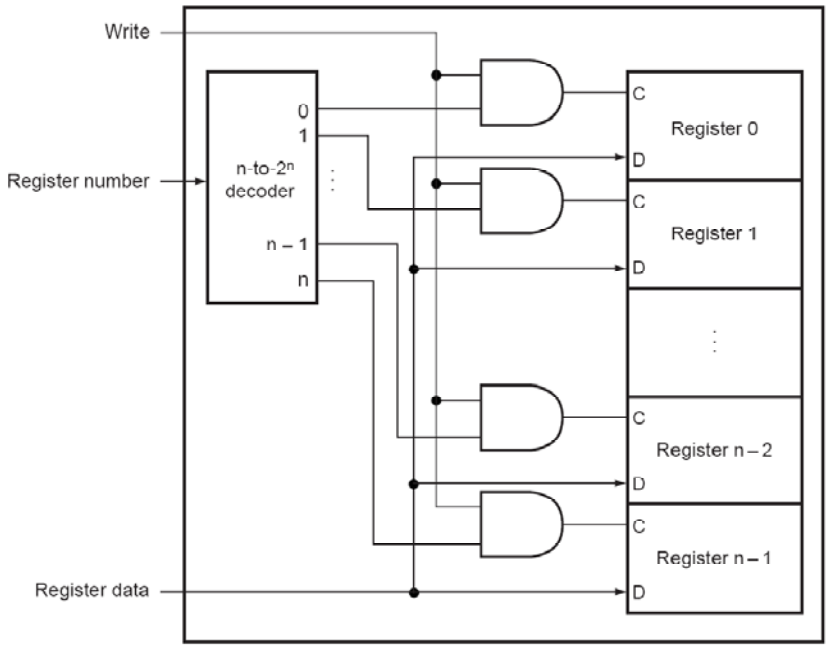
- Make sure you understand the abstractions!
- Sometimes it is easy to think you do, when you don't



Text Book : C56

Register File (Write)

- Note: we still use the real clock to determine when to write

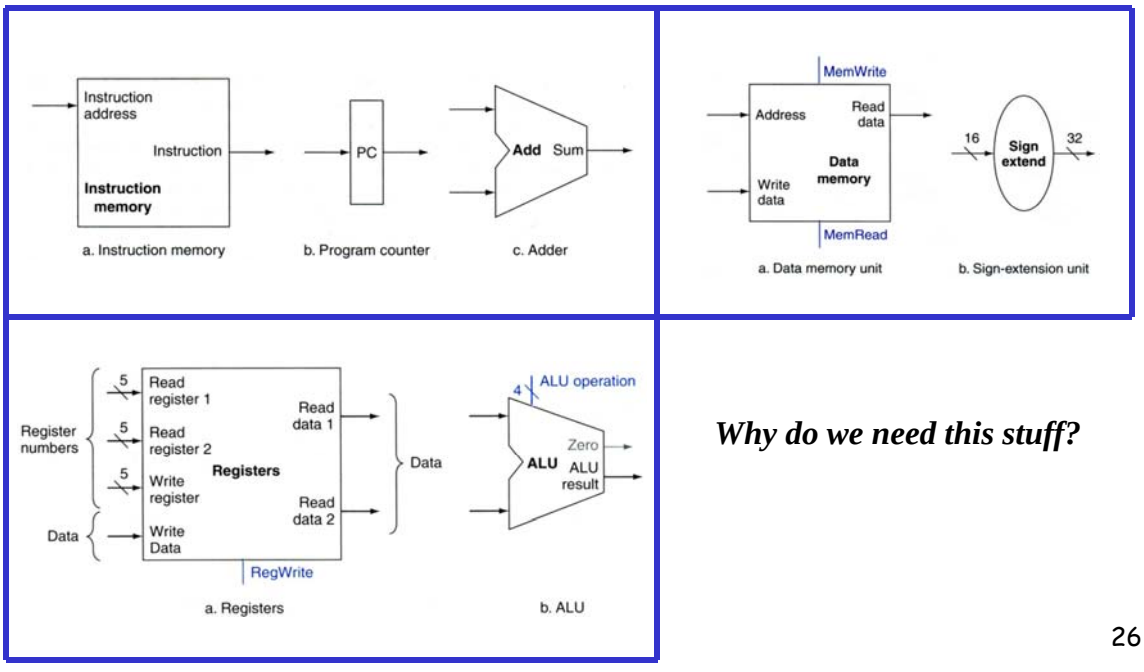


Outline

- Introduction (4.1)
- Logic Design Conventions (4.2, C.7, C.8, C.11)
- Building a Datapath (4.3)
- A Simple Implementation Scheme (4.4, D.2)

Simple Implementation

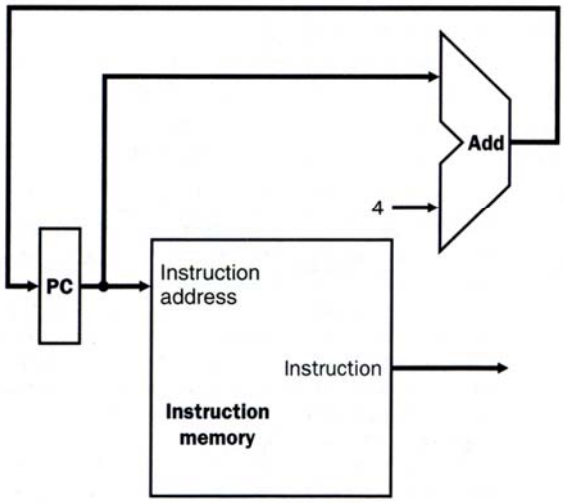
- Include the functional units we need for each instruction



Why do we need this stuff?

Fetch the instruction

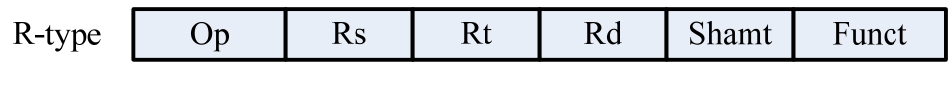
- PC: program counter
- Instruction memory
  - Can be cache

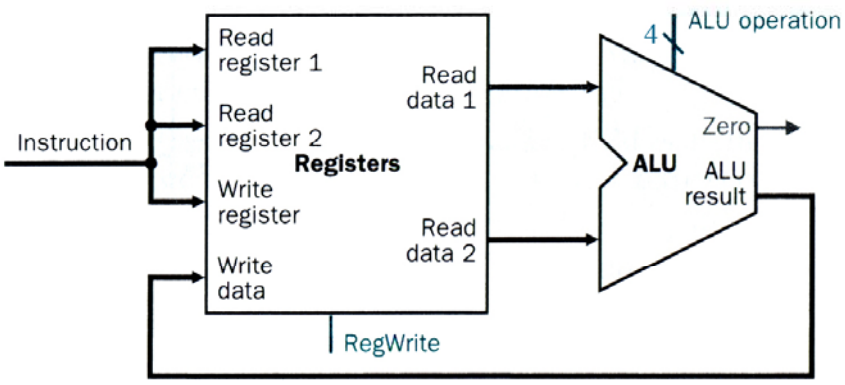


Department of Electrical Engineering, Feng-Chia University

Datapath for R-type instructions

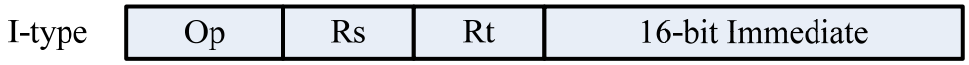
- Read two registers + ALU operation + write one register

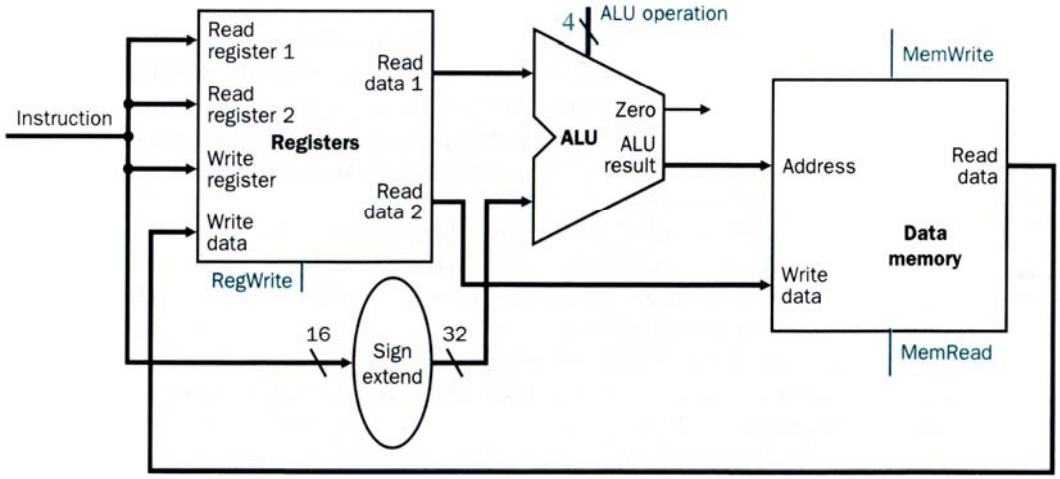




Datapath for a load or store

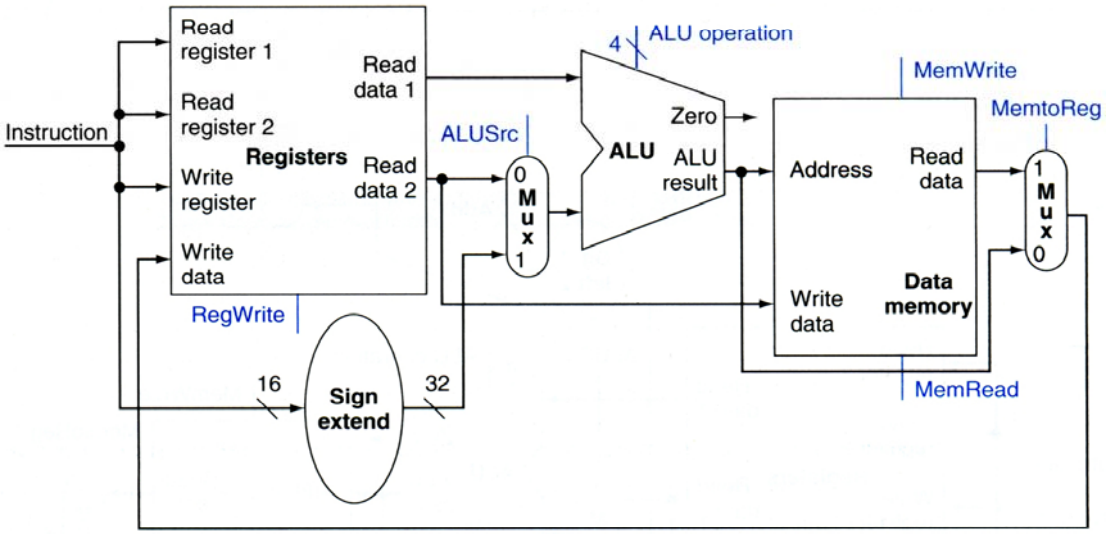
- ALU computes the address used in accessing the memory





Datapath for memory instructions and R-type instructions

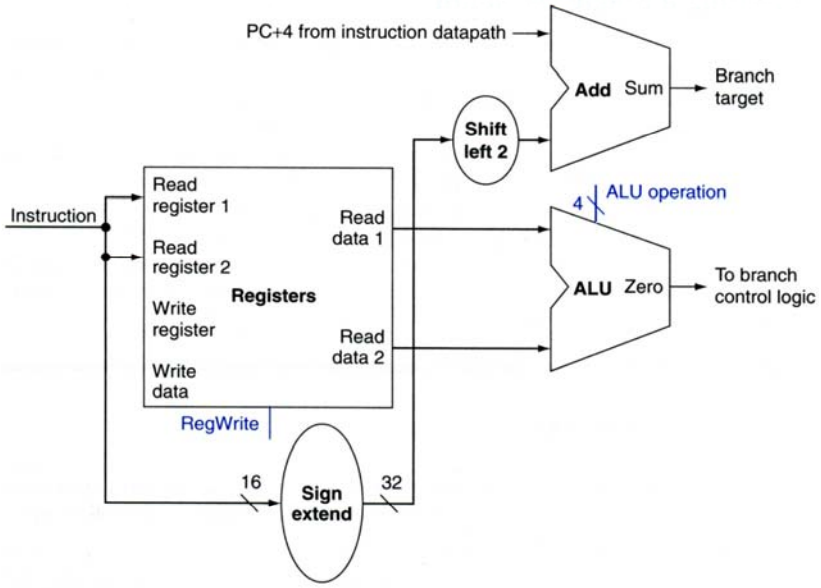
- Combine what we have so far



Datapath for a branch

- Another ALU computes the branch target address

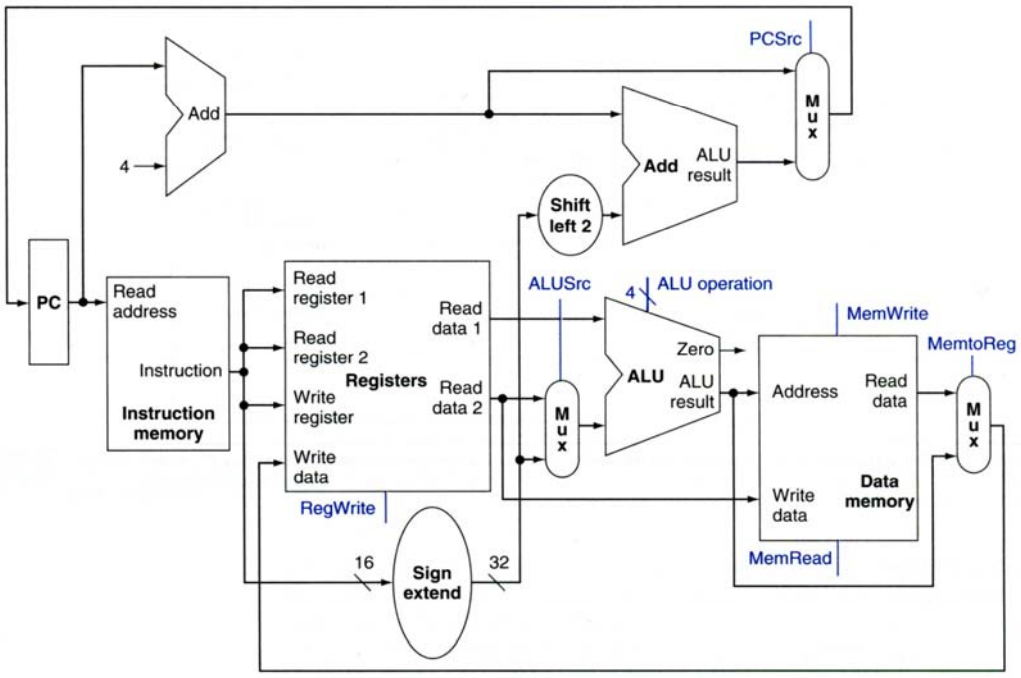
I-type	Op	Rs	Rt	16-bit Immediate



Department of Electrical Engineering, Feng-Chia University

Building the Datapath

- Use multiplexors to stitch them together



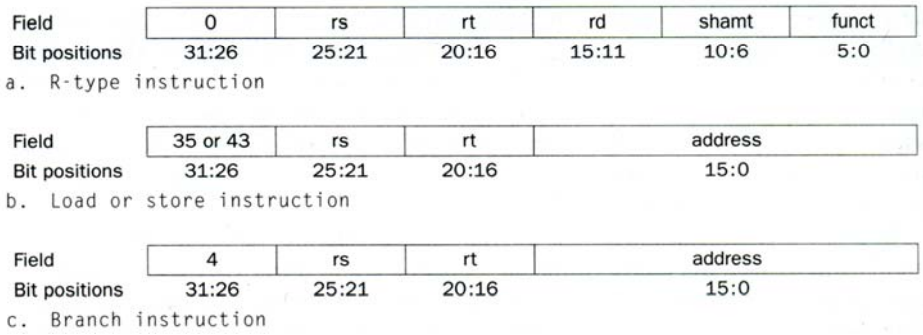
Computer Organization and Architecture, Fall 2010

Outline

- Introduction (4.1)
- Logic Design Conventions (4.2, C.7, C.8, C.11)
- Building a Datapath (4.3)
- A Simple Implementation Scheme (4.4, D.2)

Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Instruction Format:



- ALU's operation based on instruction type and function code

Control

- Example: add $8, $17, $18

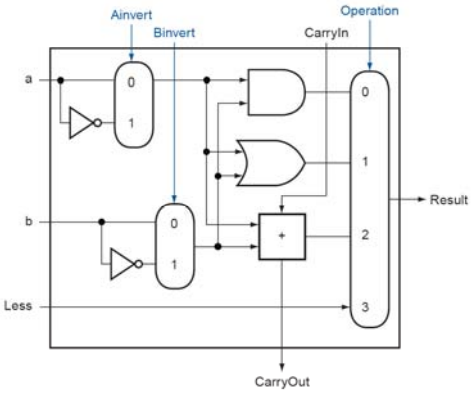
op	rs	rt	rd	shamt	funct
000000	10001	10010	01000	00000	100000

- Example: lw $1, 100($2)

op	rs	rt	16 bit offset
100011	00010	00001	0000000001100100

- ALU control input

ALU control lines	Function
0 0 0 0	AND
0 0 0 1	OR
0 0 1 0	add
0 1 1 0	subtract
0 1 1 1	set on less than
1 1 0 0	NOR



Department of Electrical Engineering, Feng-Chia University

Two level decode

- ALUOp is assigned based on instruction type

Instruction	Instruction opcode	ALUOp	Funct field	Desired ALU action	ALU control input
lw	LW (35)	00	XXXXXX	add	0010
sw	SW (43)	00	XXXXXX	add	0010
beq	Branch (4)	01	XXXXXX	subtract	0110
add	R-type (0)	10	100000	add	0010
sub	R-type (0)	10	100010	subtract	0110
and	R-type (0)	10	100100	and	0000
or	R-type (0)	10	100101	or	0001
slt	R-type (0)	10	101010	Set on less than	0111

ALU control unit

- Must describe hardware to compute 4-bit ALU control input
  - given instruction type
    00 = lw, sw
    01 = beq,
    10 = arithmetic
    → **ALUOp Assigned from instruction type**
  - function code for arithmetic
  - "11" is not used, so "10" and "01" can be "1x" and "x1" respectively.
- Describe it using a truth table (can turn into gates):

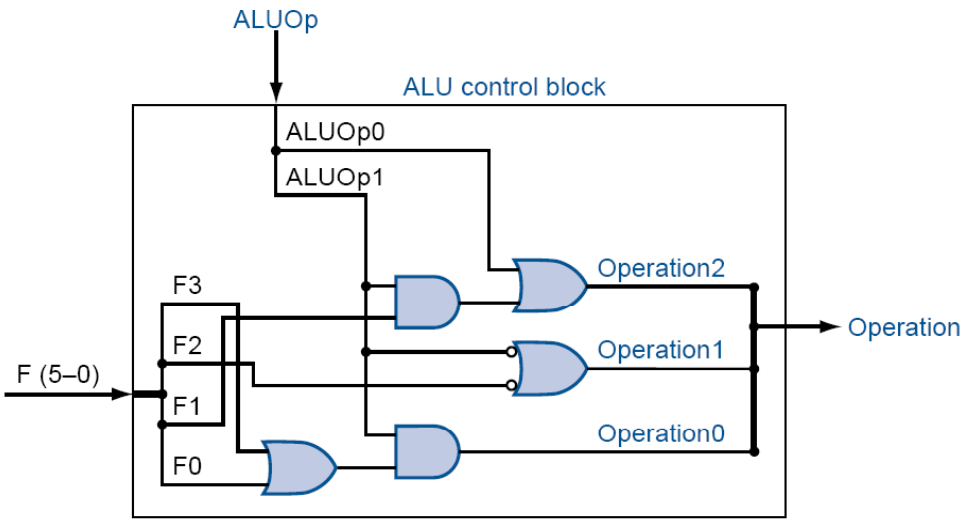
ALUOp		Funct field						Operation
ALUOp[1]	ALUOp[0]	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

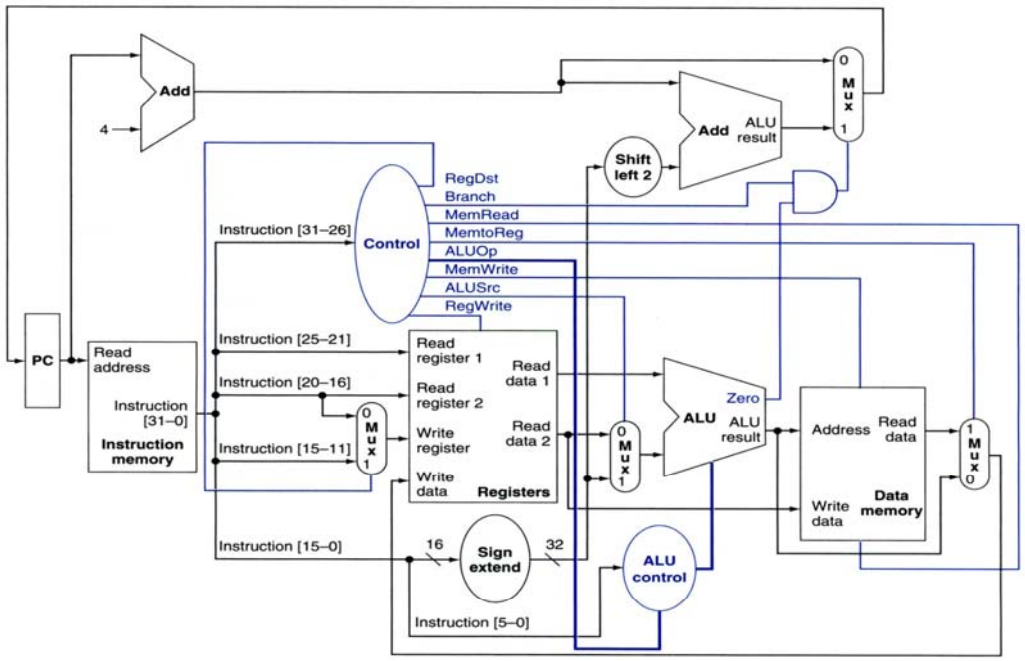
Department of Electrical Engineering, Feng-Chia University

Text Book : D4 ~ D6

ALU control unit

- Simple combinational logic





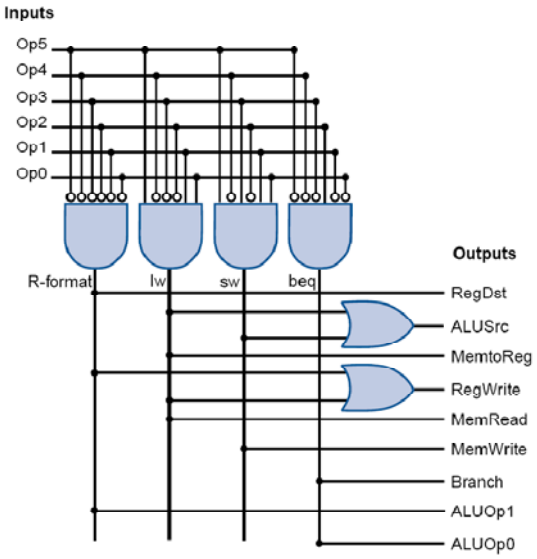
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp2
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Text Book : D7 ~ D8

Main control unit

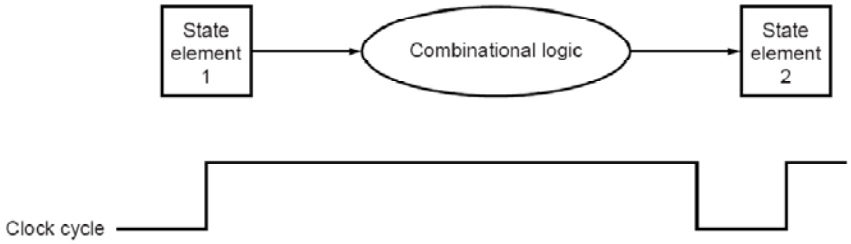
- Truth table for the main control unit

	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



Our Simple Control Structure

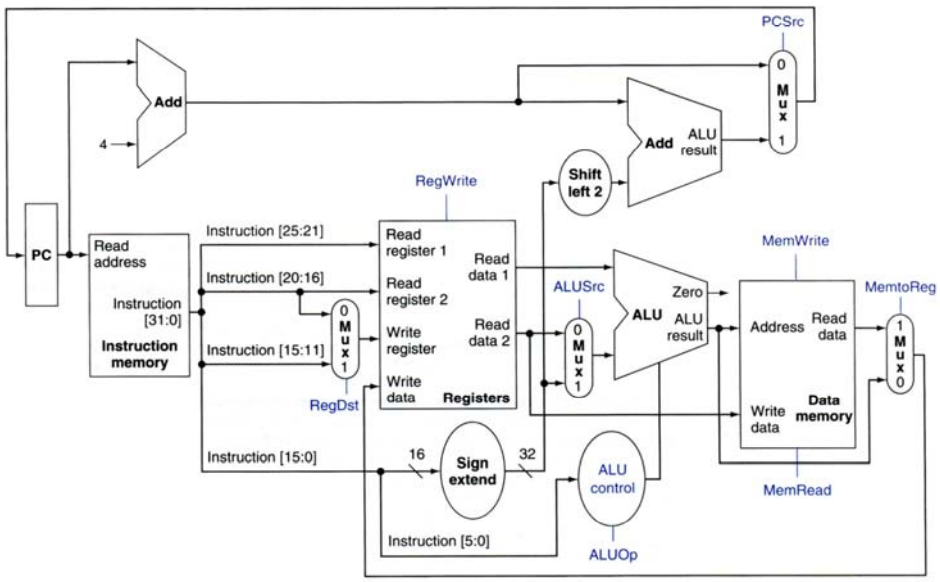
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
  - ALU might not produce "right answer" right away
  - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



We are ignoring some details like setup and hold times

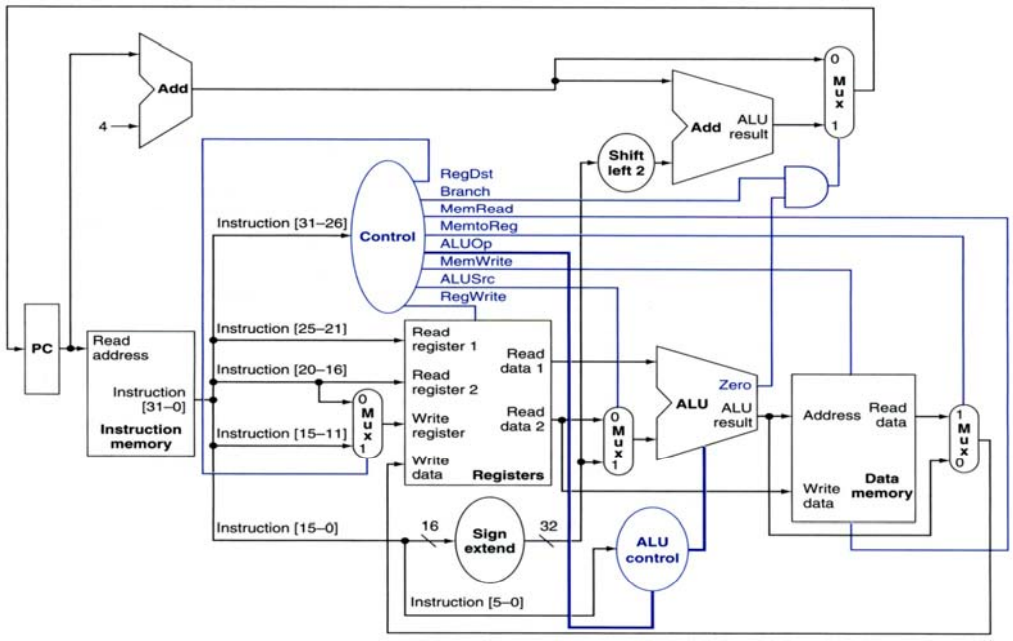
Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - memory (200ps), ALU and adders (100ps), register file access (50ps)



Minimum clock period

- F/F propagation time + datapath delay + setup time



Department of Electrical Engineering, Feng-Chia University

Performance of Single-Cycle Machines

	Instruction mix	Instruction fetch	Register read	ALU operation	Data memory	Register write	Total
R-type	45%	200	50	100	0	50	400 ps
Load word	25%	200	50	100	200	50	600 ps
Store word	10%	200	50	100	200	0	550 ps
Branch	15%	200	50	100	0	0	350 ps
Jump	5%	200	0	0	0	0	200 ps