CS4100: 計算機結構

Designing a Single-Cycle Processor

國立清華大學資訊工程學系
一零零學年度第二學期

國立清華大學
National Tsing Hua University

Outline

♦ **Introduction to designing a processor**

♦ **Analyzing the instruction set**

♦ **Building the datapath**

♦ **A single-cycle implementation**

♦ **Control for the single-cycle CPU**

- **Control of CPU operations**
- **ALU controller**
- **Main controller**
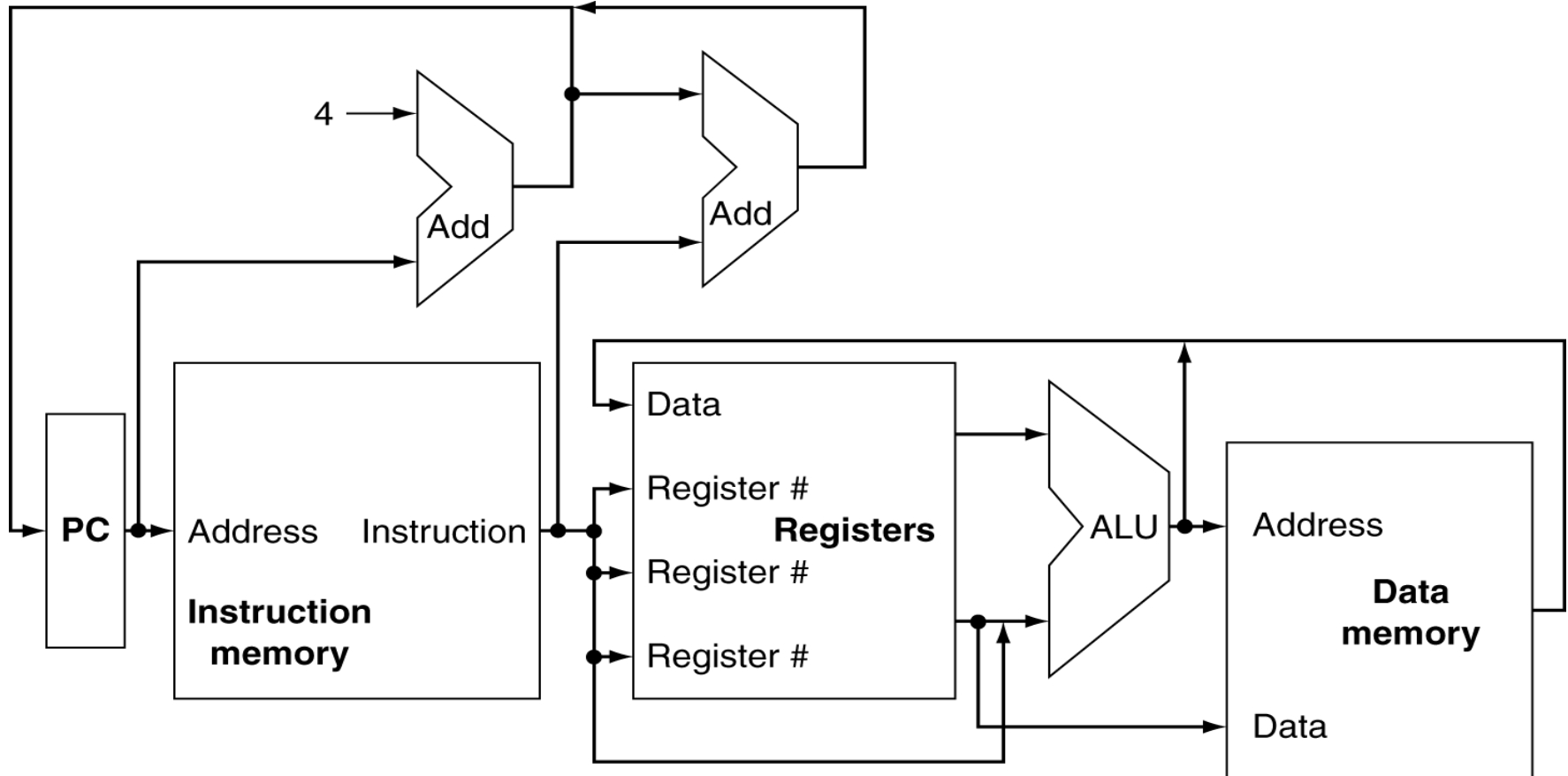
國立清華大學
National Tsing Hua University

Introduction

♦ **CPU performance factors**
  - **Instruction count**
    - ■ **Determined by ISA and compiler**
  - **CPI and Cycle time**
    - ■ **Determined by CPU hardware**

♦ **We will examine two MIPS implementations**
  - **A simplified version**
  - **A more realistic pipelined version**

♦ **Simple subset, shows most aspects**
  - **Memory reference:** `lw, sw`
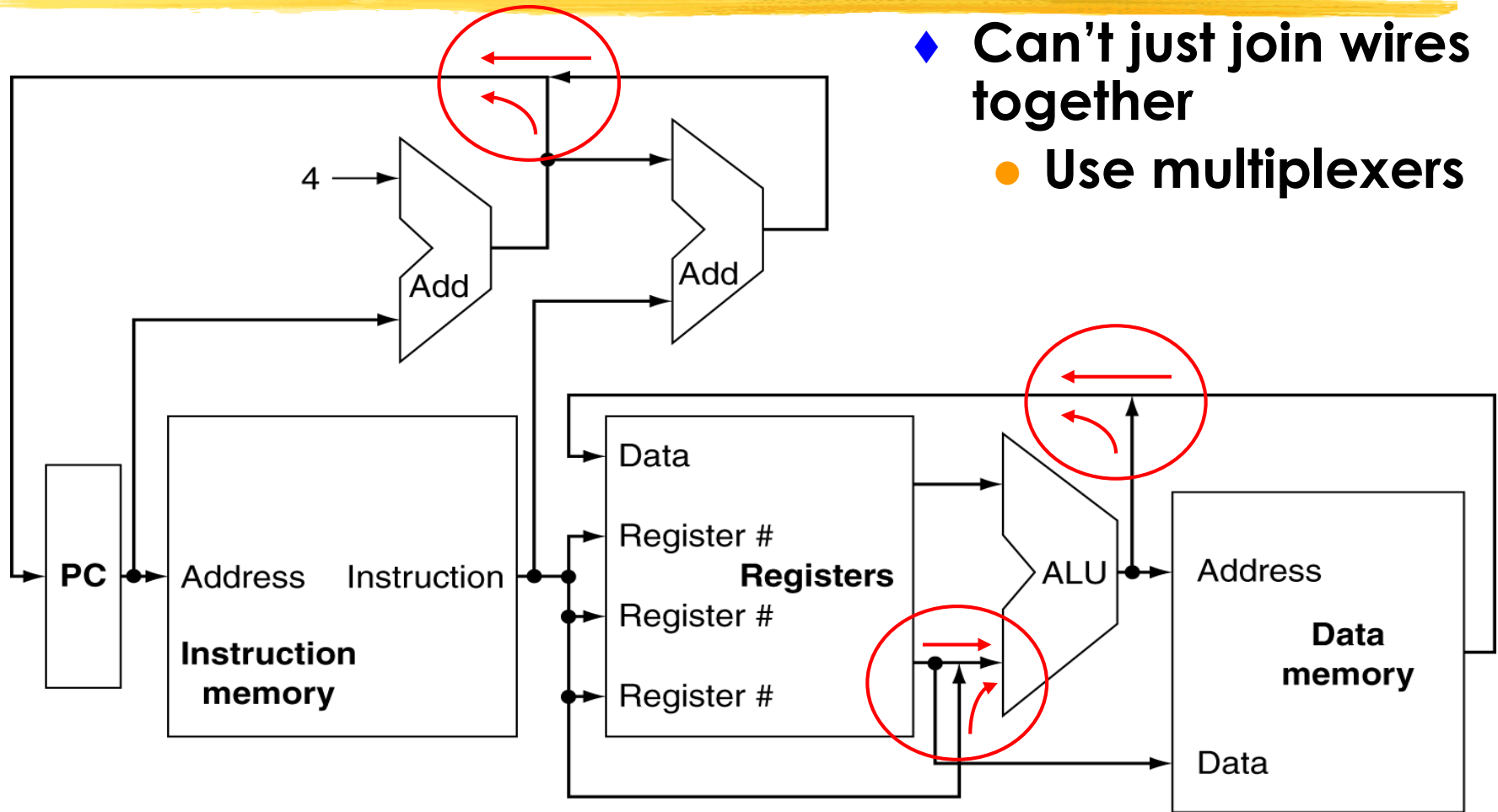 - **Arithmetic/logical:** `add, sub, and, or, slt`
  - **Control transfer:** `beq, j`

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**
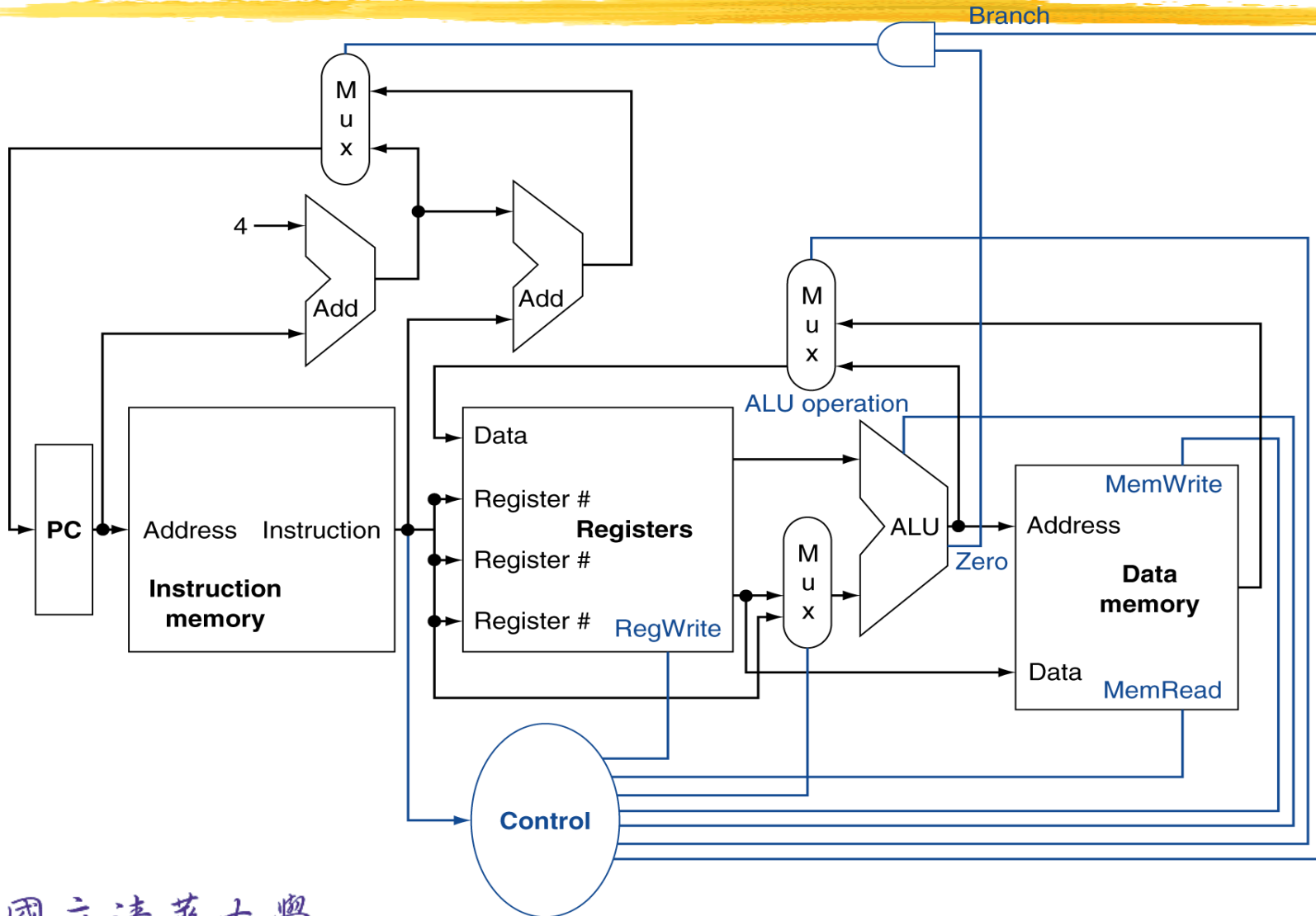- **PC ← target address or PC + 4**

CPU Overview

國立清華大學
National Tsing Hua University

Computer Architecture

Multiplexers

- ◆ **Can't just join wires together**
  - ● **Use multiplexers**

國立清華大學
National Tsing Hua University

Control

Computer Architecture
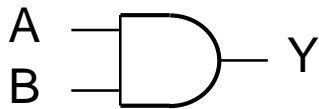
國立清華大學
National Tsing Hua University

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**
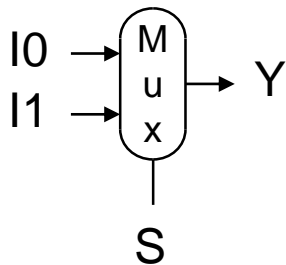- ♦ **State (sequential) elements**
  - ● **Store information**

國立清華大學
National Tsing Hua University

Computer Architecture

Combinational Elements

- ♦ **AND-gate**
  - ● **Y = A & B**

A
B ⟩── Y

- ♦ **Multiplexer**
  - ● **Y = S ? I1 : I0**

I0 →
I1 → Mux → Y
S

- ♦ **Adder**
  - ● **Y = A + B**

A →
B → + → Y
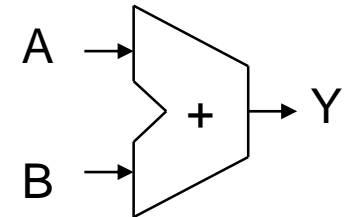
- ♦ **Arithmetic/Logic Unit**
  - ● **Y = F(A, B)**

A →
B → ALU → Y
F

國立清華大學
National Tsing Hua University

Computer Architecture
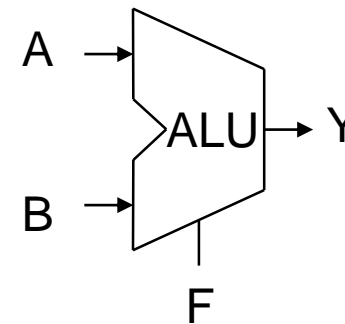
Sequential Elements

- ◆ **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**

National Tsing Hua University

Computer Architecture
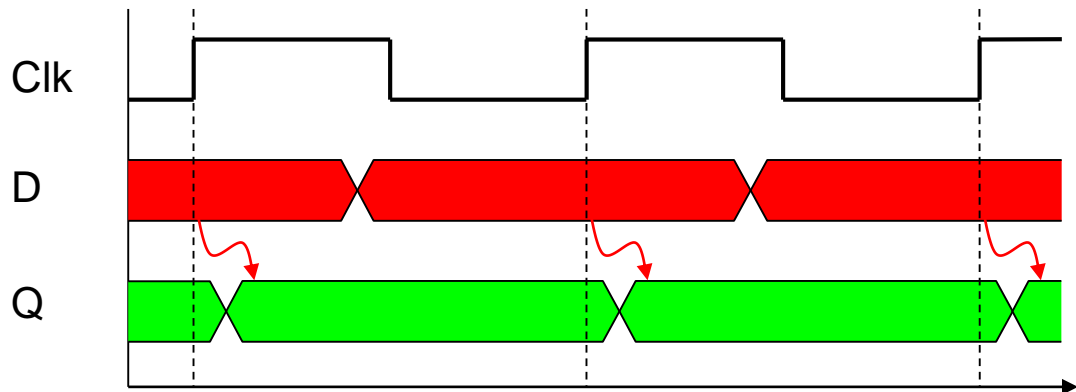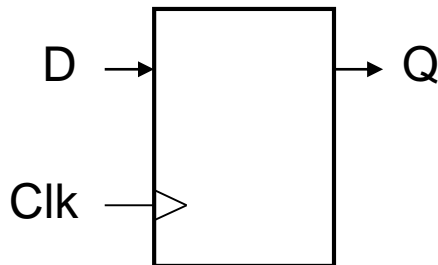
Sequential Elements

♦ **Register with write control**

- **Only updates on clock edge when write control input is 1**

- **Used when stored value is required later**
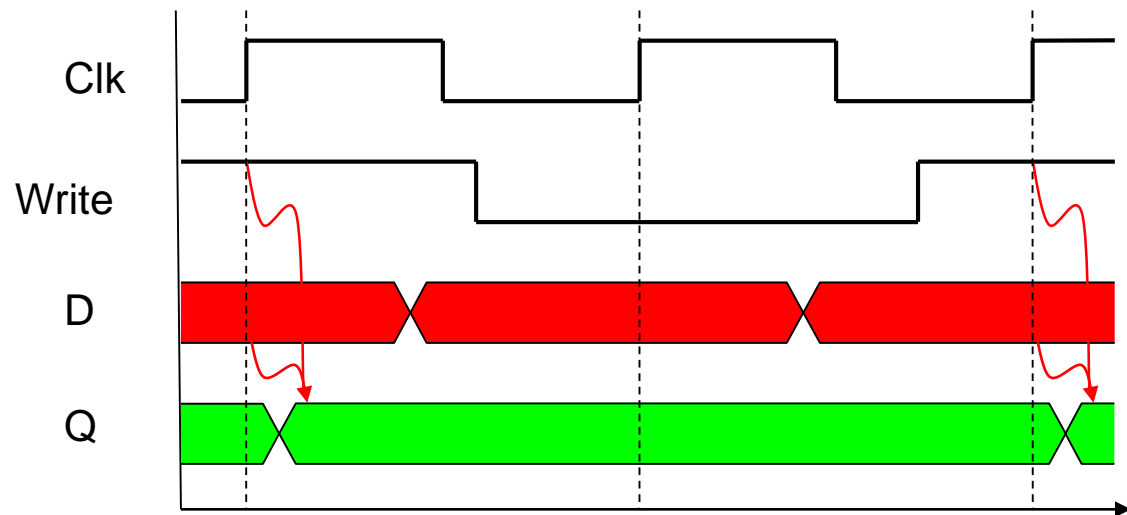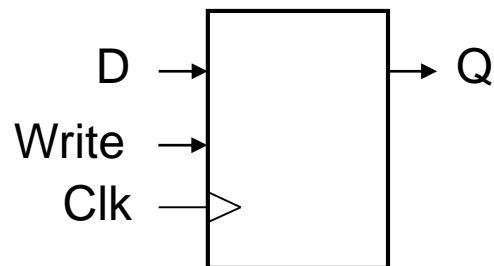
Computer Architecture

National Tsing Hua University

Clocking Methodology

♦ **Combinational logic transforms data during clock cycles**

  ● **Between clock edges**

  ● **Input from state elements, output to state element**

  ● **Longest delay determines clock period**

國立清華大學
National Tsing Hua University

Computer Architecture
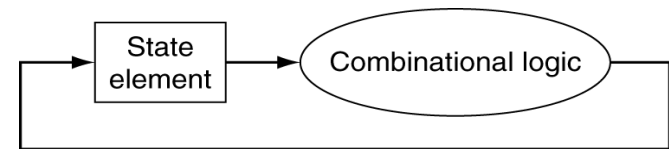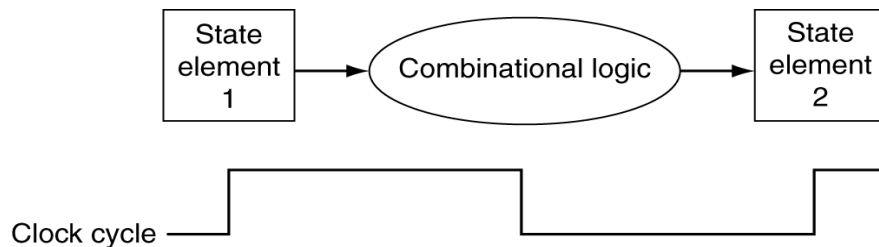
How to Design a Processor?

1. **Analyze instruction set (datapath requirements)**
   - The meaning of each instruction is given by the *register transfers*
   - Datapath must include storage element
   - Datapath must support each register transfer
2. **Select set of datapath components and establish clocking methodology**
3. **Assemble datapath meeting the requirements**
4. **Analyze implementation of each instruction to determine setting of control points effecting register transfer**
5. **Assemble the control logic**

Outline

♦ **Introduction to designing a processor**

♦ **Analyzing the instruction set (step 1)**

♦ **Building the datapath**

♦ **A single-cycle implementation**

♦ **Control for the single-cycle CPU**

 ● **Control of CPU operations**

 ● **ALU controller**

 ● **Main controller**

Step 1: Analyze Instruction Set

- **All MIPS instructions are 32 bits long with 3 formats:**
  - **R-type:**

31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - **I-type:**

31	26	21	16	0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

  - **J-type:**

31	26	0
op	target address
6 bits	26 bits
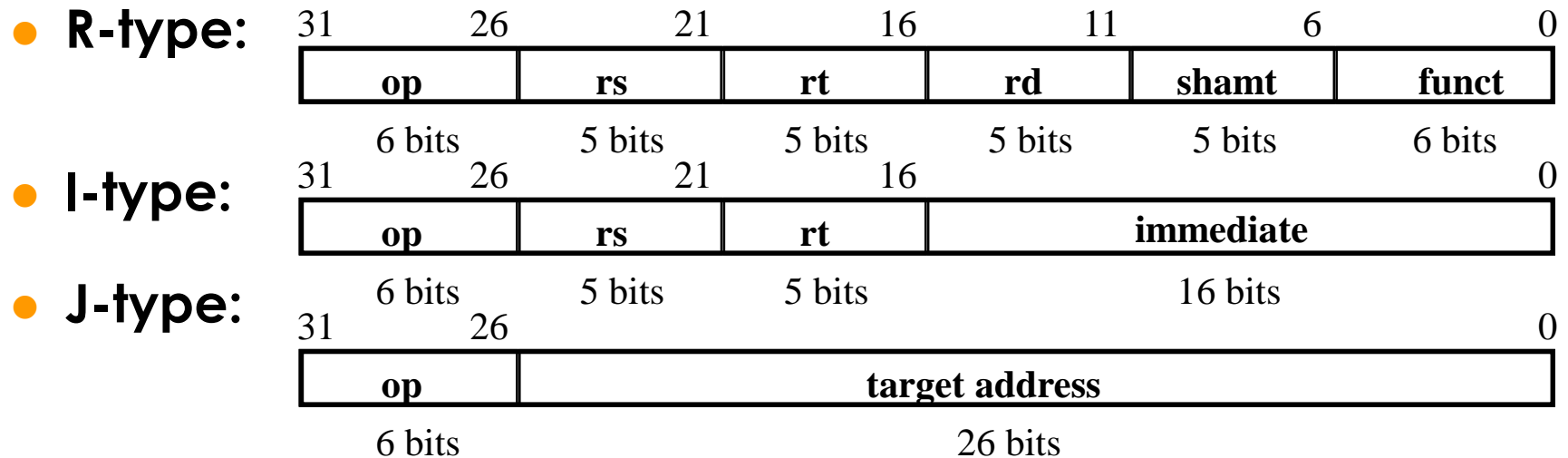
- **The different fields are:**
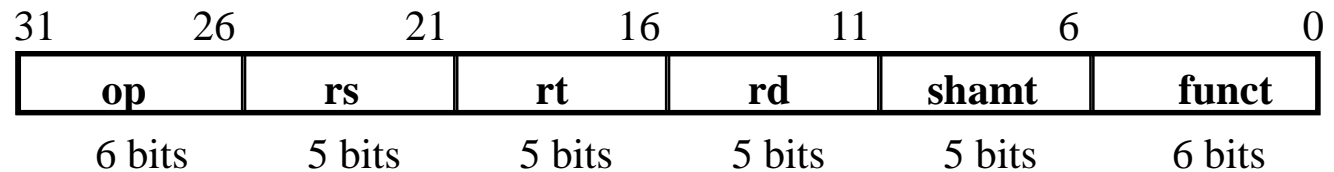 - op: operation of the instruction
  - rs, rt, rd: source and destination register
  - shamt: shift amount
  - funct: selects variant of the "op" field
  - address / immediate
  - target address: target address of jump

Our Example: A MIPS Subset

- **R-Type:**
  - add rd, rs, rt
  - sub rd, rs, rt
  - and rd, rs, rt
  - or rd, rs, rt
  - slt rd, rs, rt

31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **Load/Store:**
  - lw rt,rs,imm16
  - sw rt,rs,imm16

31 26	21	16	0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits
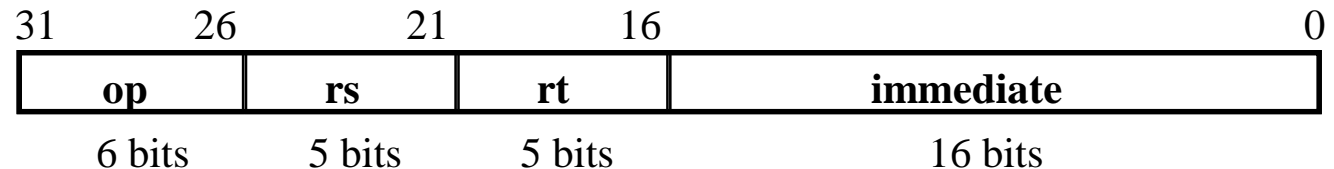
- **Imm operand:**
  - addi   rt,rs,imm16
- **Branch:**
  - beq rs,rt,imm16
- **Jump:**
  - j target

31 26	21 16	0
op	address	
6 bits	26 bits	
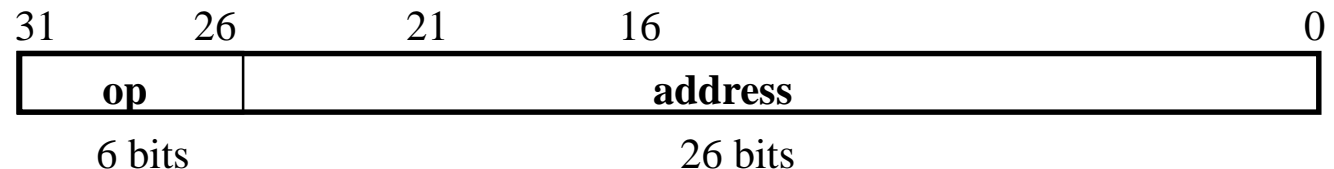
Logical Register Transfers

♦ **RTL gives the <u>meaning</u> of the instructions**

♦ **All start by fetching the instruction, read registers, then use ALU => simplicity and regularity help**

MEM[PC] = op | rs | rt | rd | shamt | funct
    or   = op | rs | rt | Imm16
    or   = op | Imm26 (added at the end)

Inst	Register transfers
ADD	R[rd] <- R[rs] + R[rt]; PC <- PC + 4
SUB	R[rd] <- R[rs] - R[rt]; PC <- PC + 4
LOAD	R[rt] <- MEM[R[rs] + sign_ext(Imm16)]; PC <- PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <-R[rt]; PC <- PC + 4
ADDI	R[rt] <- R[rs] + sign_ext(Imm16)]; PC <- PC + 4
BEQ	if (R[rs] == R[rt]) then PC <- PC + 4 + sign_ext(Imm16)] || 00 else PC <- PC + 4

Requirements of Instruction Set

After checking the register transfers, we can see that datapath needs the followings:

♦ **Memory**
  - store instructions and data

♦ **Registers (32 x 32)**
  - read RS
  - read RT
  - Write RT or RD

♦ **PC**

♦ **Extender for zero- or sign-extension**

♦ **Add and sub register or extended immediate (ALU)**

♦ **Add 4 or extended immediate to PC**

Outline

♦ **Introduction to designing a processor**

♦ **Analyzing the instruction set**

♦ **Building the datapath (steps 2, 3)**

♦ **A single-cycle implementation**

♦ **Control for the single-cycle CPU**

  ● **Control of CPU operations**

  ● **ALU controller**

  ● **Main controller**

國立清華大學
National Tsing Hua University

Computer Architecture

Step 2a: Datapath Components

♦ **Basic building blocks of combinational logic elements :**

Adder

MUX

ALU

National Tsing Hua University

Computer Architecture
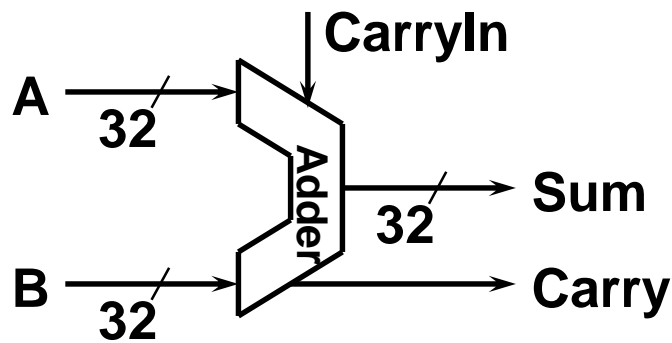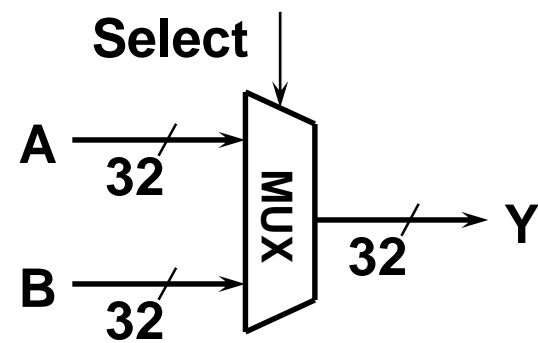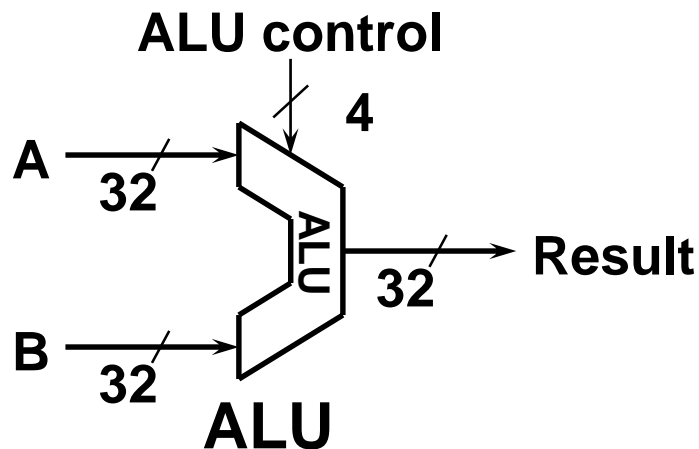
Step 2b: Datapath Components

Storage elements:

♦ **Register:**

- **Similar to the D Flip Flop except**
  - **N-bit input and output**
  - **Write Enable input**
- **Write Enable:**
  - **negated (0): Data Out will not change**
  - **asserted (1): Data Out will become Data In**

Write Enable

Data In — N → **Data Out** — N →

Clk

Computer Architecture

國立清華大學
National Tsing Hua University
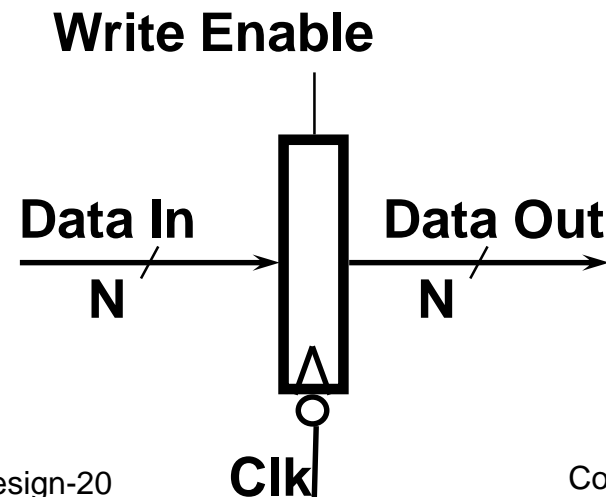
Storage Element: Register File

- ◆ **Consists of 32 registers:**
  - ● **Appendix B.8**
  - ● **Two 32-bit output busses:**
    busA and busB
  - ● **One 32-bit input bus: busW**
- ◆ **Register is selected by:**
  - ● **RA selects the register to put on busA (data)**
  - ● **RB selects the register to put on busB (data)**
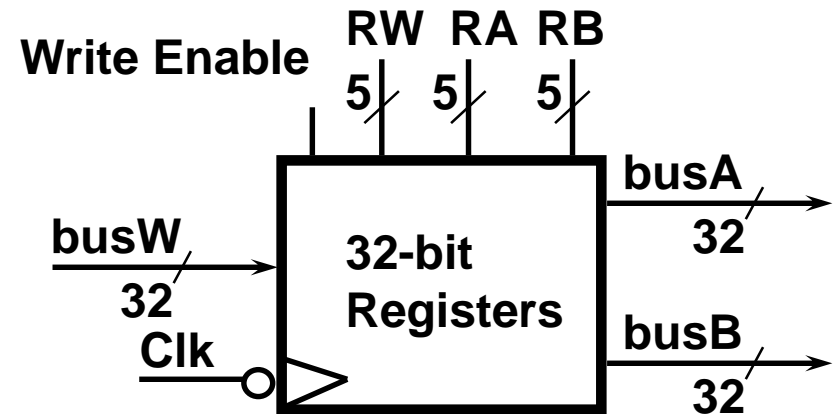 - ● **RW selects the register to be written via busW (data) when Write Enable is 1**
- ◆ **Clock input (CLK)**
  - ● **The CLK input is a factor ONLY during write operation**
  - ● **During read, behaves as a combinational circuit**

Write Enable   **RW RA RB**
5   5   5

busW → **32-bit Registers** → **busA**
32   **32**
Clk   **busB**
                                    32

Storage Element: Memory

♦ **Memory (idealized)**
  - **Appendix B.8**
  - **One input bus: Data In**
  - **One output bus: Data Out**

♦ **Word is selected by:**
  - **Address selects the word to put on Data Out**
  - **Write Enable = 1: address selects the memory word to be written via the Data In bus**

♦ **Clock input (CLK)**
  - **The CLK input is a factor ONLY during write operation**
  - **During read operation, behaves as a combinational logic block:**
    - **Address valid => Data Out valid after access time**
    - **No need for read control**

Write Enable   **Address**

Data In   **DataOut**

32   **32**

Clk

National Tsing Hua University

Computer Architecture
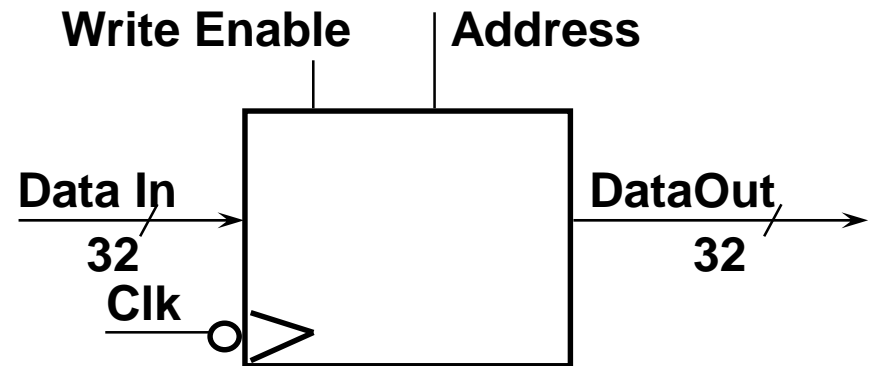
Step 3a: Datapath Assembly

♦ **Instruction fetch unit: common operations**
  - **Fetch the instruction: mem[PC]**
  - **Update the program counter:**
    - ■ **Sequential code: PC <- PC + 4**
    - ■ **Branch and Jump: PC <- "Something else"**

Single-cycle Design-23

國立清華大學
National Tsing Hua University
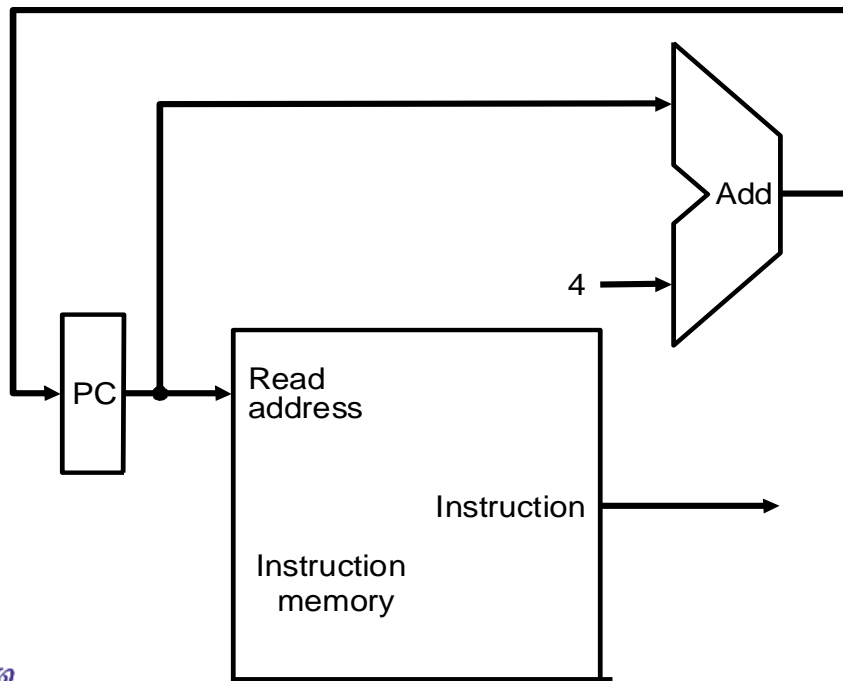
Step 3b: Add and Subtract

♦ **R[rd] <- R[rs] op R[rt]   Ex: add   rd, rs, rt**
  - **Ra, Rb, Rw come from inst.'s rs, rt, and rd fields**
  - **ALU and RegWrite: control logic after decode**

31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
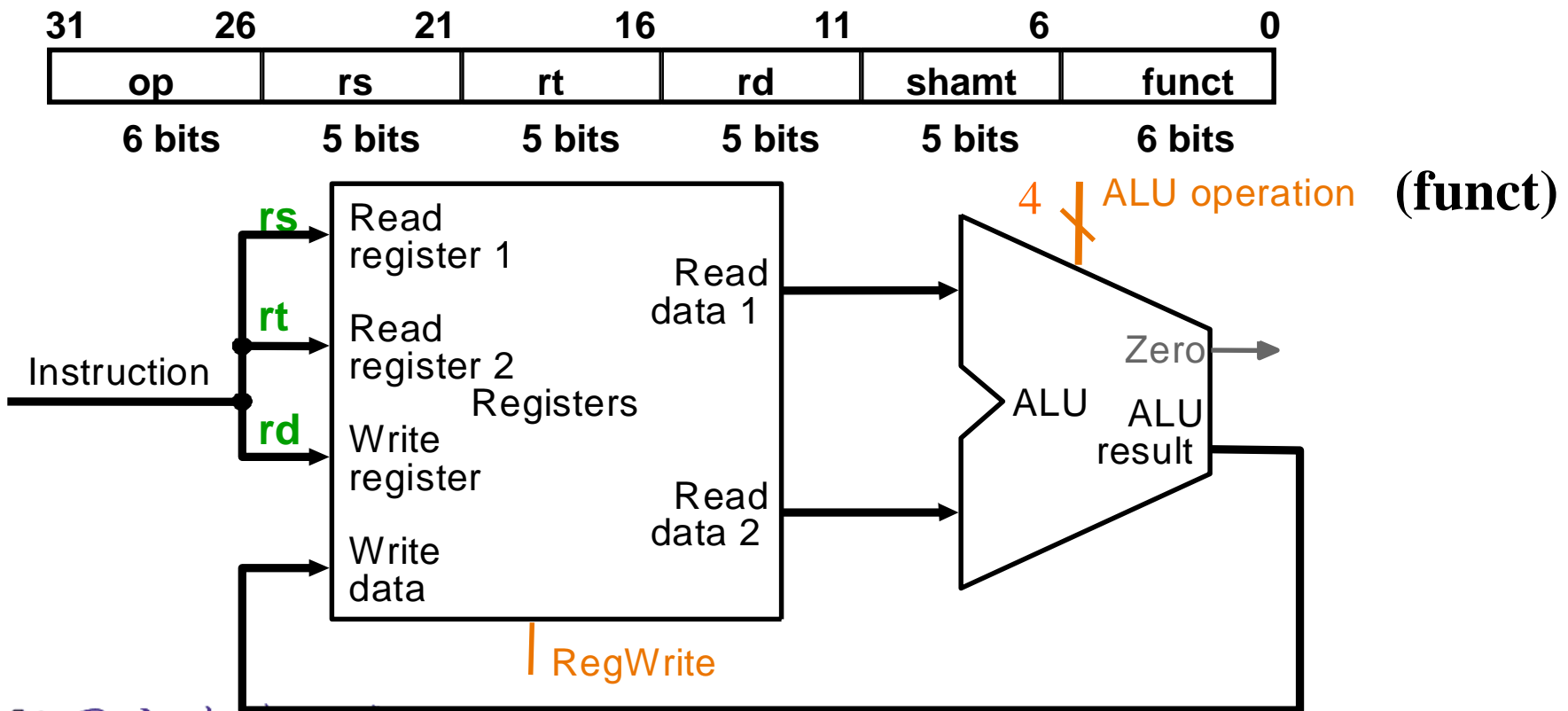
Step 3c: Store/Load Operations

♦ **R[<u>rt</u>]<-Mem[R[rs]+SignExt[imm16]]** **Ex: lw rt,rs,imm16**

Computer Architecture
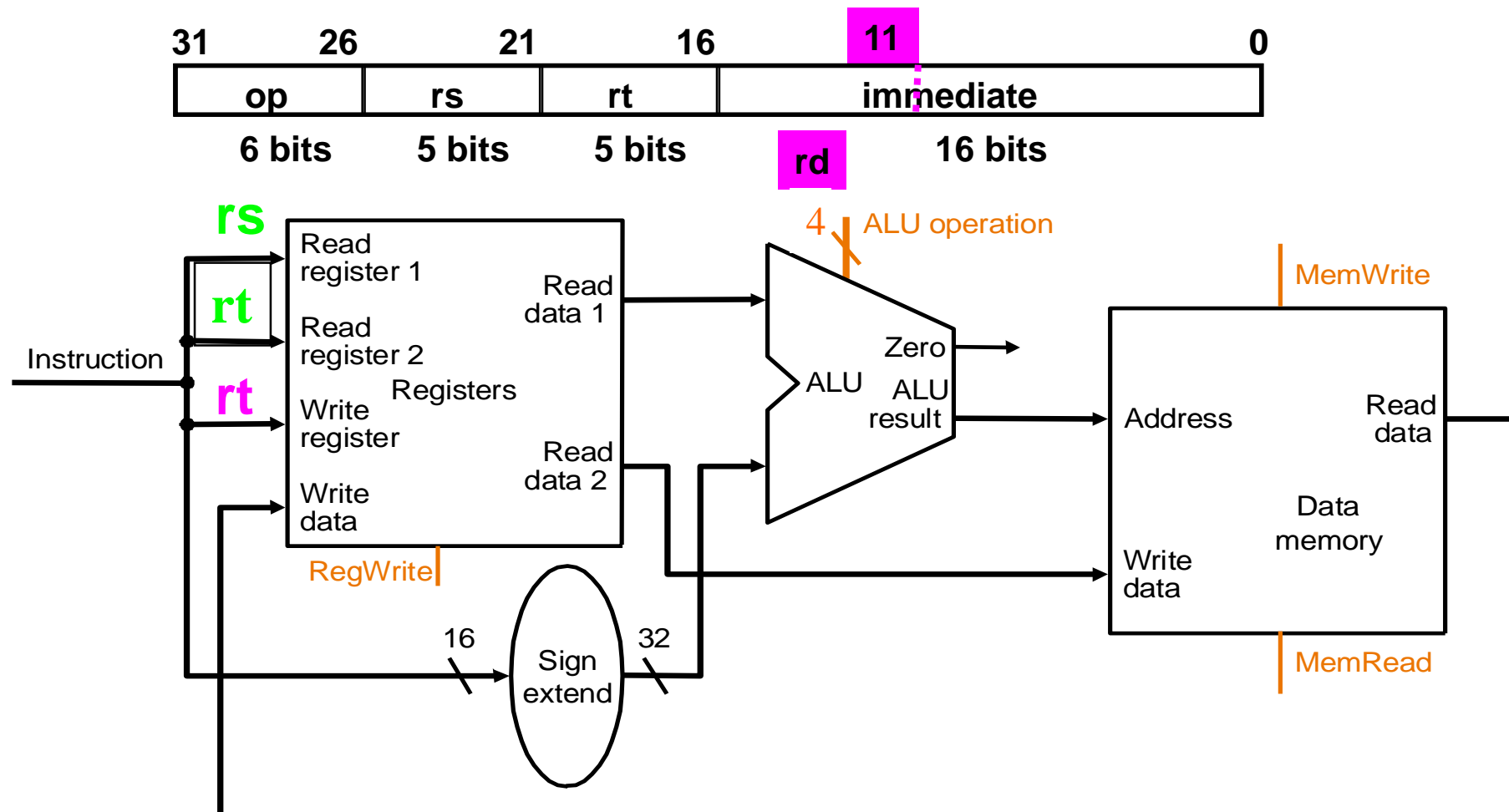
R-Type/Load/Store Datapath

National Tsing Hua University

Computer Architecture
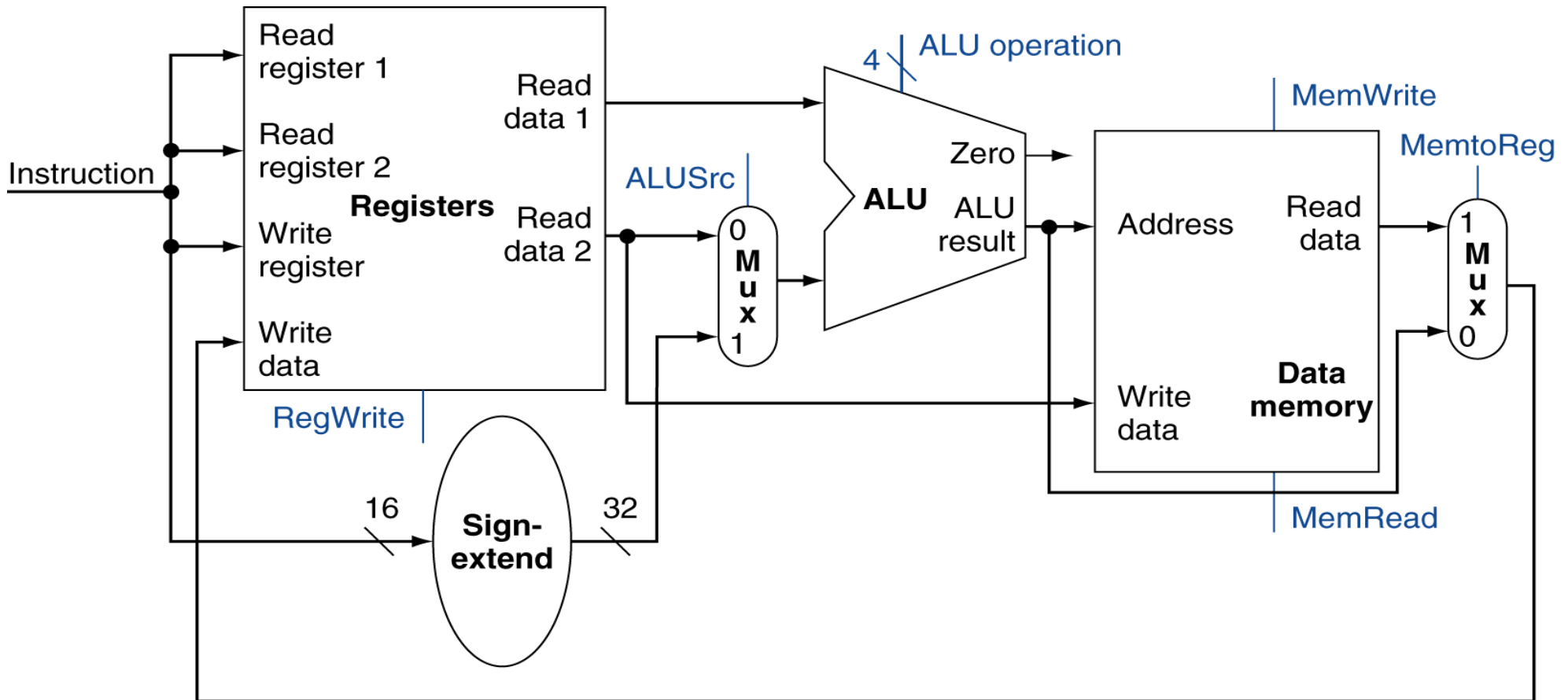
Step 3d: Branch Operations

♦ **beq rs, rt, imm16**

mem[PC]   Fetch inst. from memory

Equal <- R[rs] == R[rt]   Calculate branch condition

if (COND == 0)   Calculate next inst. address
   PC <- PC + 4 + (SignExt(imm16) x 4)
else
   PC <- PC + 4

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	
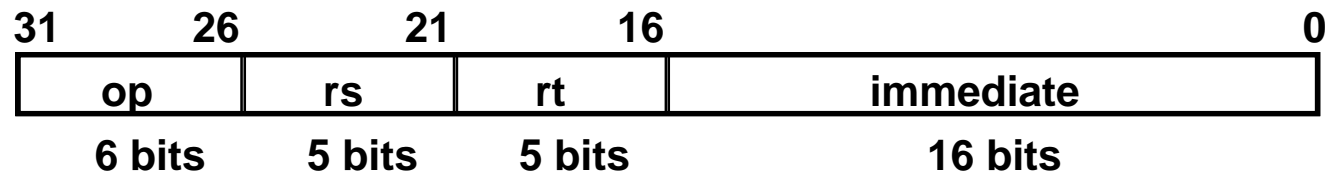
Datapath for Branch Operations
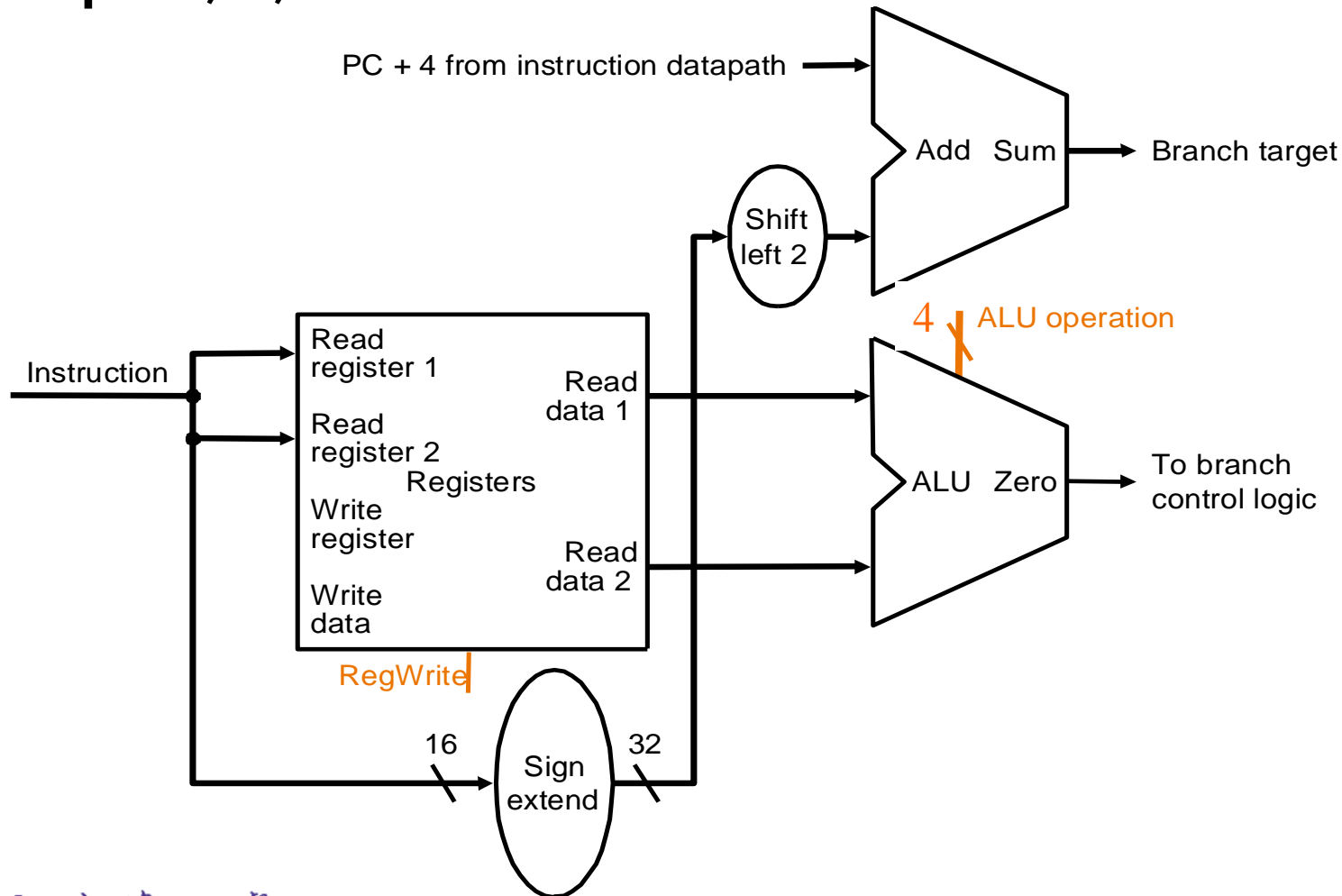
♦ **beq   rs, rt, imm16**

Outline

- **Introduction to designing a processor**
- **Analyzing the instruction set**
- **Building the datapath**
- **A single-cycle implementation**
- **Control for the single-cycle CPU**
  - **Control of CPU operations**
  - **ALU controller**
  - **Main controller**

國立清華大學
National Tsing Hua University

Computer Architecture

A Single Cycle Datapath

國立清華大學
National Tsing Hua University

Computer Architecture
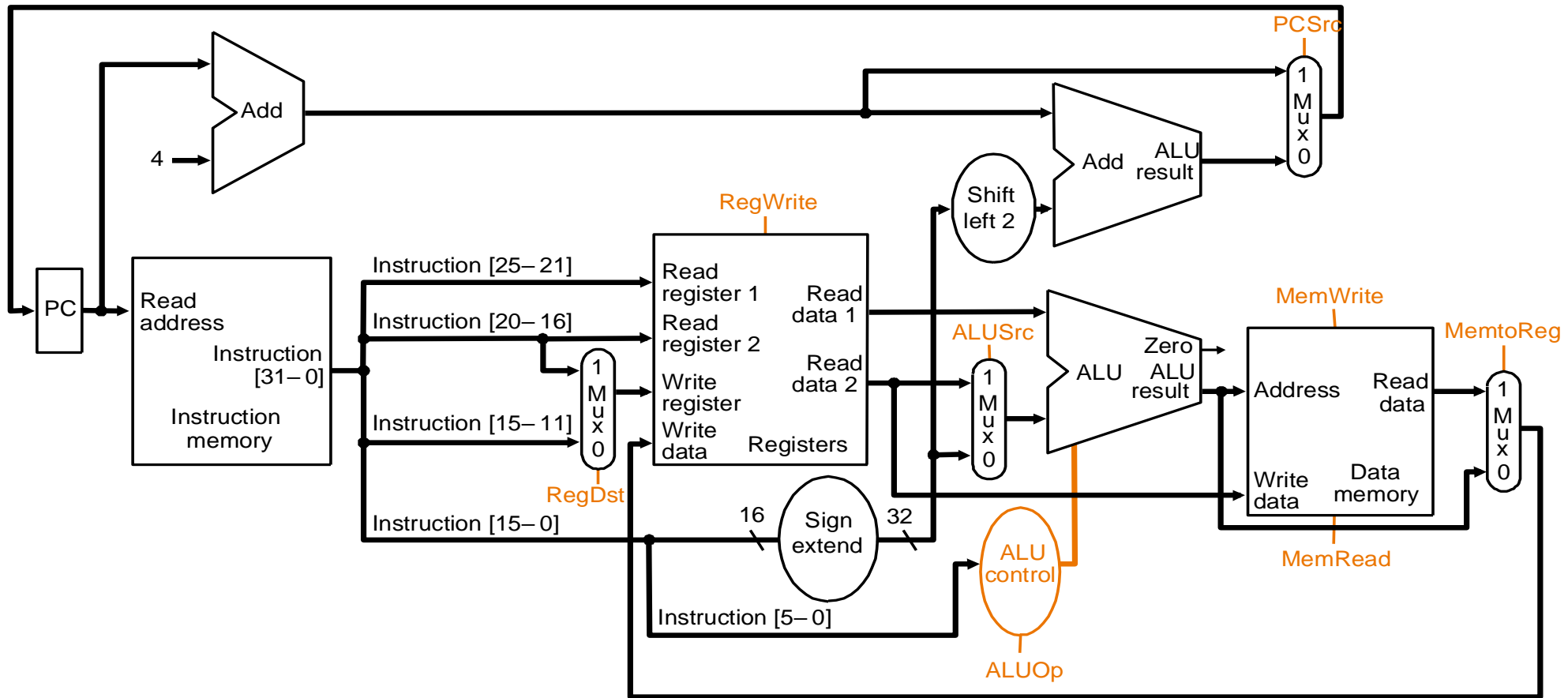
Data Flow during add

- **Clocking**
- **data flows in other paths**

Computer Architecture

國立清華大學
National Tsing Hua University

Clocking Methodology

♦ **Combinational logic transforms data during clock cycles**

- **Between clock edges**
- **Input from state elements, output to state element**
- **Longest delay determines clock period**

國立清華大學
National Tsing Hua University
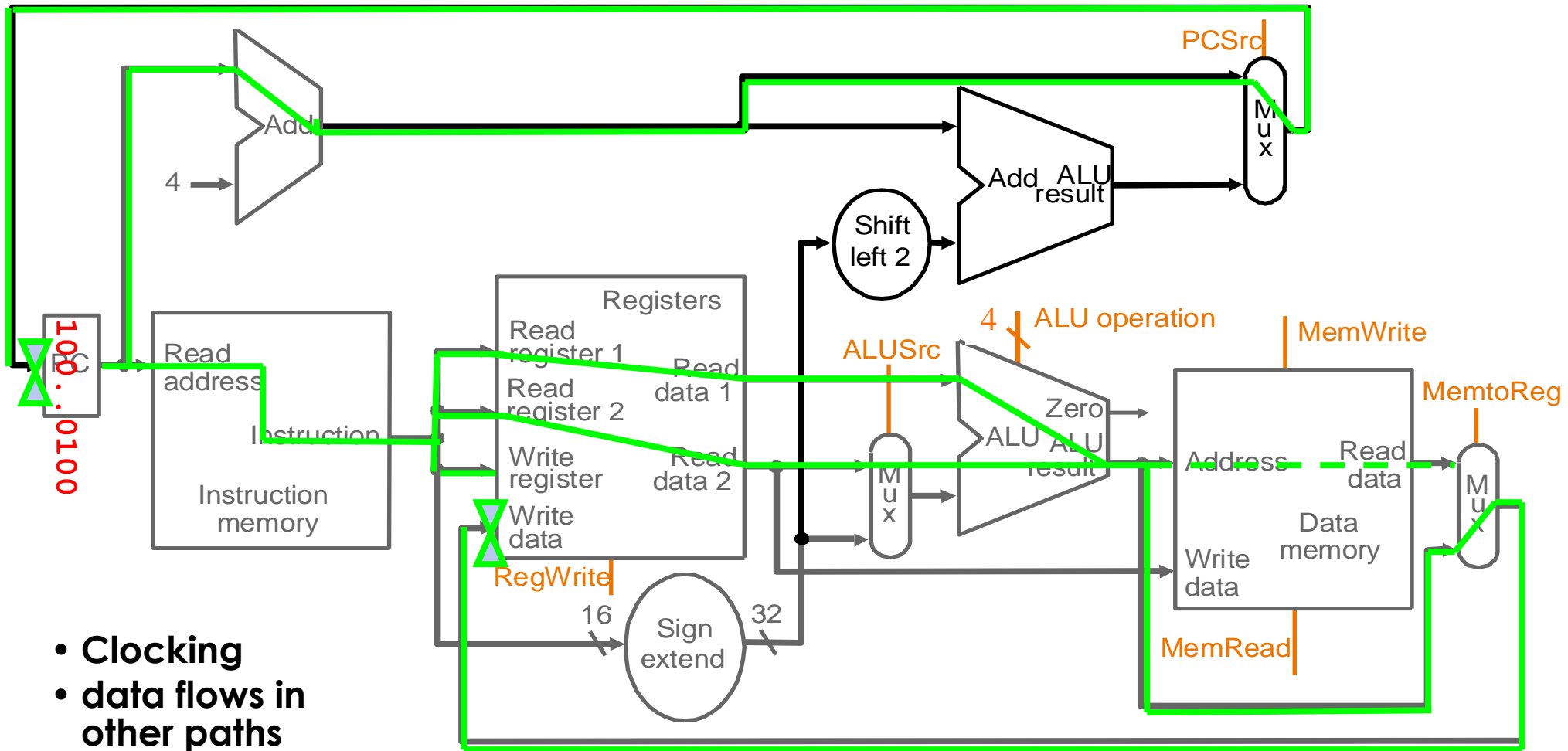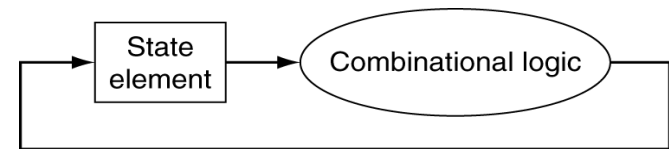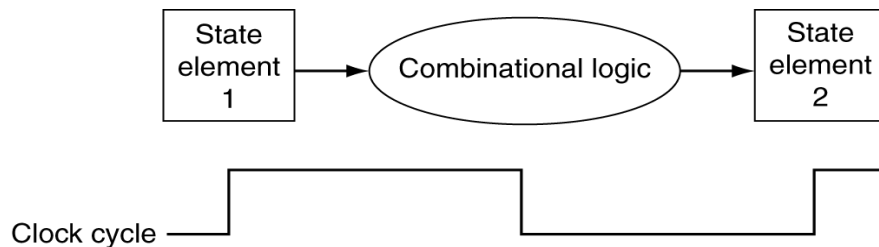
Register-Register Timing

Clk

PC **Old Value** | **New Value**

Clk-to-Q

Rs, Rt, Rd, Op, Func **Old Value** | **New Value**

Instruction Memory Access Time

ALUctr **Old Value** | **New Value**

Delay through Control Logic

RegWr **Old Value** | **New Value**

Register File Access Time

busA, B **Old Value** | **New Value**

ALU Delay

busW **Old Value** | **New Value**

Register Write Occurs Here

Ideal Instruction Memory

PC

Clk

32

RegWr

Rd Rs Rt

5   5   5

Rw Ra Rb

32 32-bit Registers

busW

32

Clk

busA

32

busB

32

ALUctr

ALU

Result

32

The Critical Path

♦ **Register file and ideal memory:**
  ● **During read, behave as combinational logic:**
    ■ **Address valid => Output valid after access time**
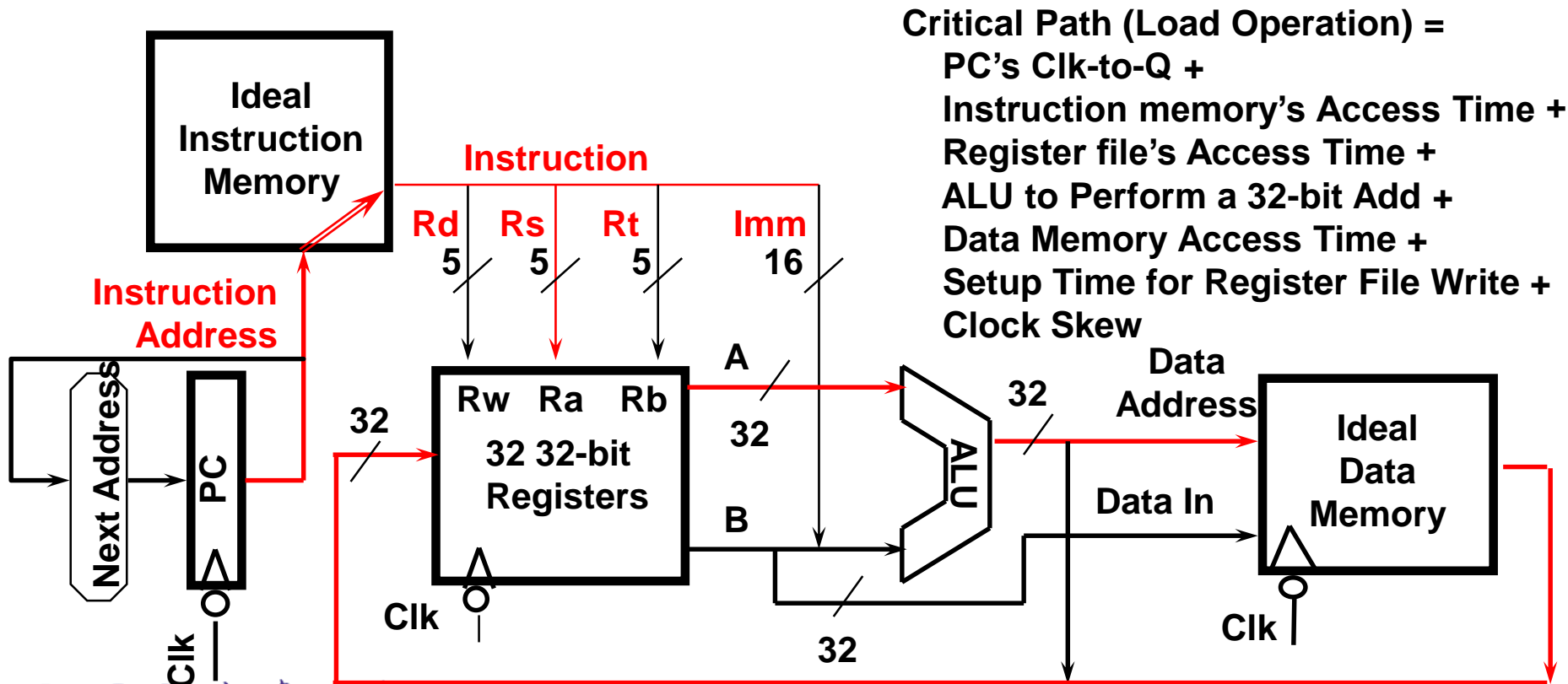
Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction memory's Access Time +
Register file's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew
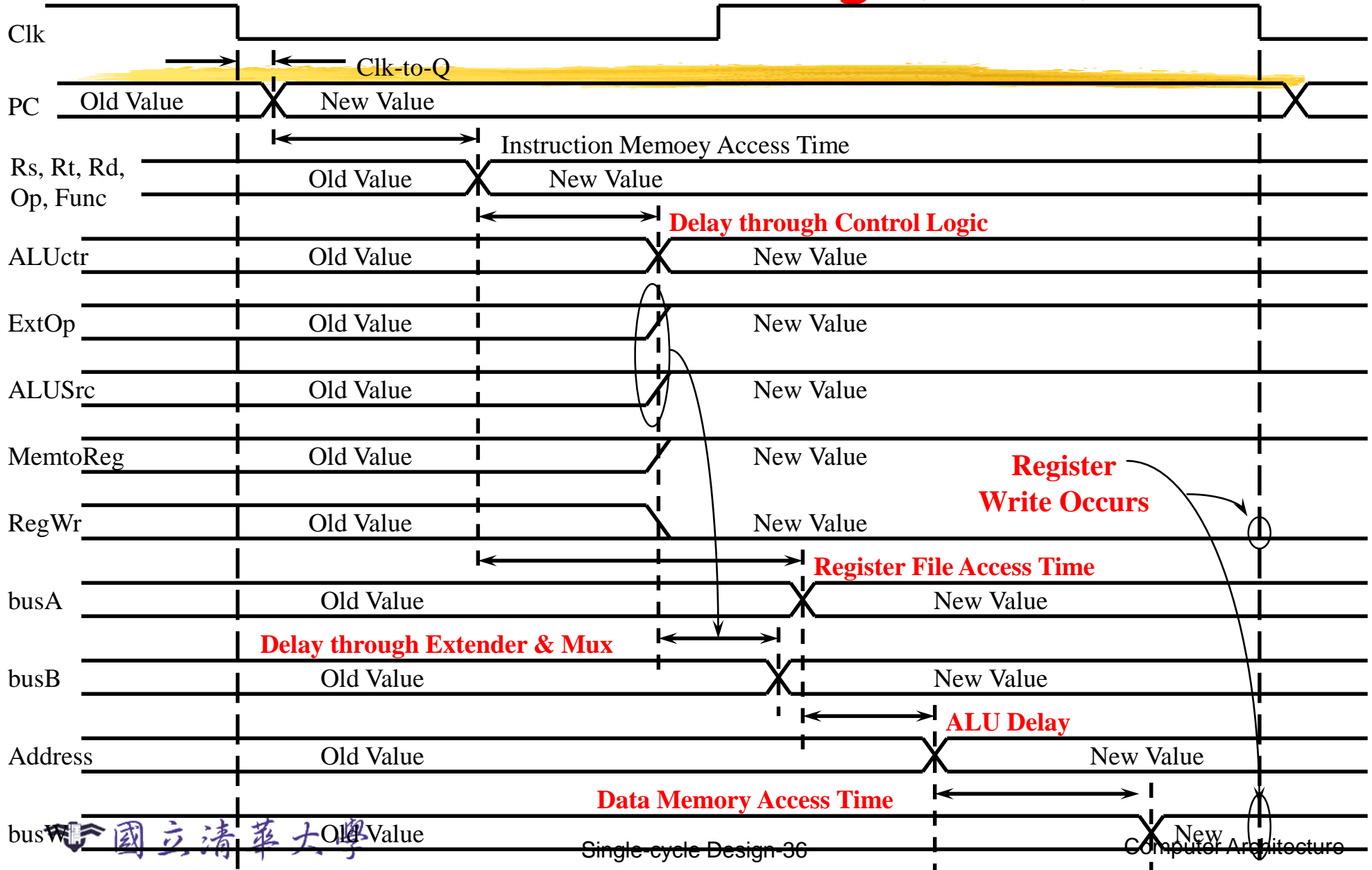
國立清華大學
National Tsing Hua University

Computer Architecture

Worst Case Timing (Load)

Clk

PC: Old Value | New Value
Clk-to-Q

Rs, Rt, Rd, Op, Func: Old Value | New Value
Instruction Memoey Access Time

ALUctr: Old Value | New Value
Delay through Control Logic

ExtOp: Old Value | New Value

ALUSrc: Old Value | New Value

MemtoReg: Old Value | New Value

RegWr: Old Value | New Value

Register Write Occurs

busA: Old Value | New Value
Register File Access Time

Delay through Extender & Mux

busB: Old Value | New Value

ALU Delay

Address: Old Value | New Value

Data Memory Access Time

busW: Old Value | New

Computer Architecture

Outline

♦ **Introduction to designing a processor**

♦ **Analyzing the instruction set**

♦ **Building the datapath**

♦ **A single-cycle implementation**

♦ **Control for the single-cycle CPU**

  ● **Control of CPU operations (step 4)**

  ● **ALU controller**

  ● **Main controller**

國立清華大學
National Tsing Hua University

Step 4: Control Points and Signals

Instruction<31:0>

Inst. Memory

Addr

<21:25> <21:25> <16:20> <11:15> <0:15>

Op Funct Rt Rs Rd Imm16

Control

PCsrc   RegWr   RegDst   MemRd   ALUSrc   ALUctr   MemWr   MemtoReg   Equal

Datapath

國立清華大學
National Tsing Hua University

Computer Architecture
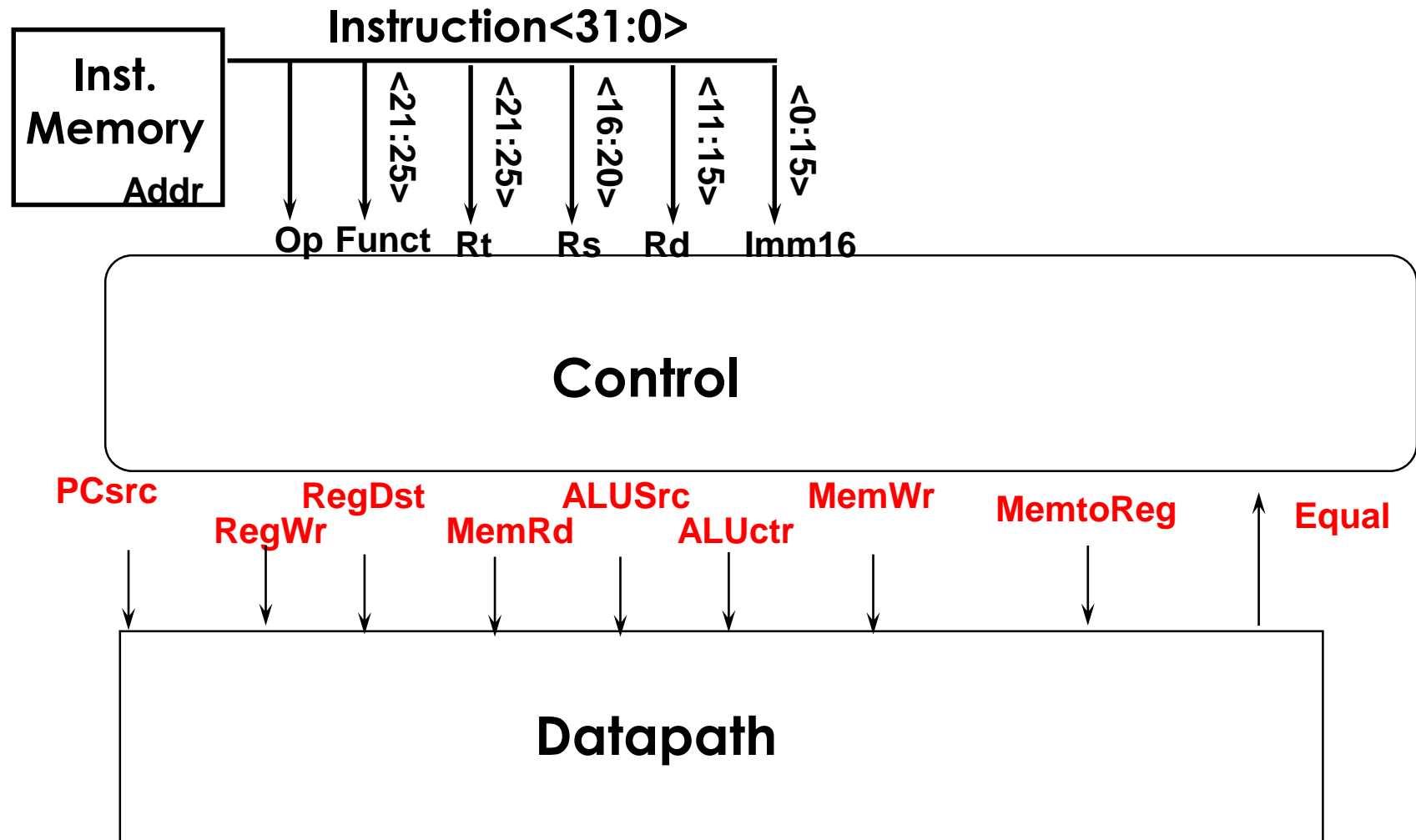
Designing Main Control

♦ **Some observations:**
- ● **opcode (Op[5-0]) is always in bits 31-26**
- ● **two registers to be read are always in rs (bits 25-21) and rt (bits 20-16) (for R-type, beq, sw)**
- ● **base register for lw and sw is always in rs (25-21)**
- ● **16-bit offset for beq, lw, sw is always in 15-0**
- ● **destination register is in one of two positions:**
  - ■ **lw: in bits 20-16 (rt)**
  - ■ **R-type: in bits 15-11 (rd)**
  - **=> need a multiplex to select the address for written register**

Datapath with Mux and Control

國立清華大學
National Tsing Hua University

Datapath with Control Unit

國立清華大學
National Tsing Hua University

Computer Architecture
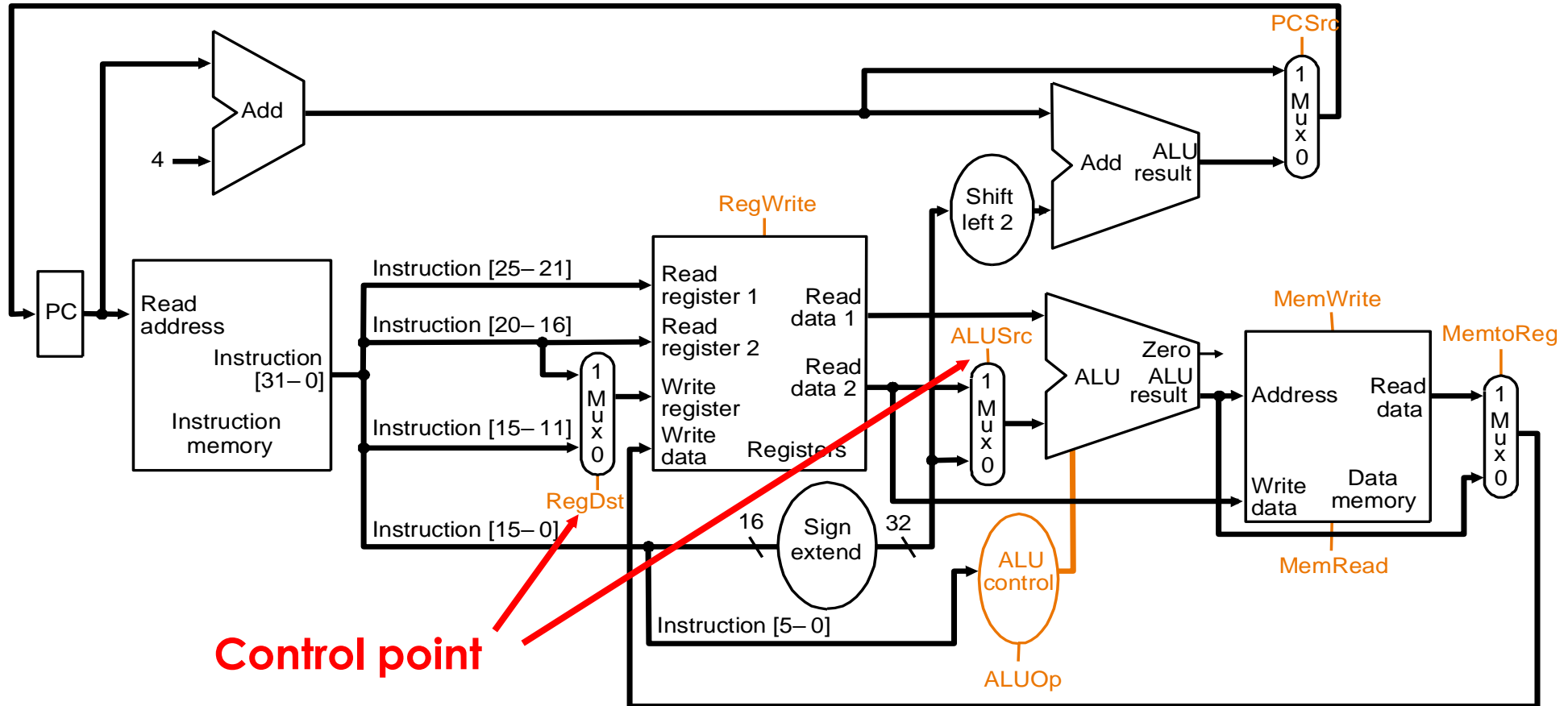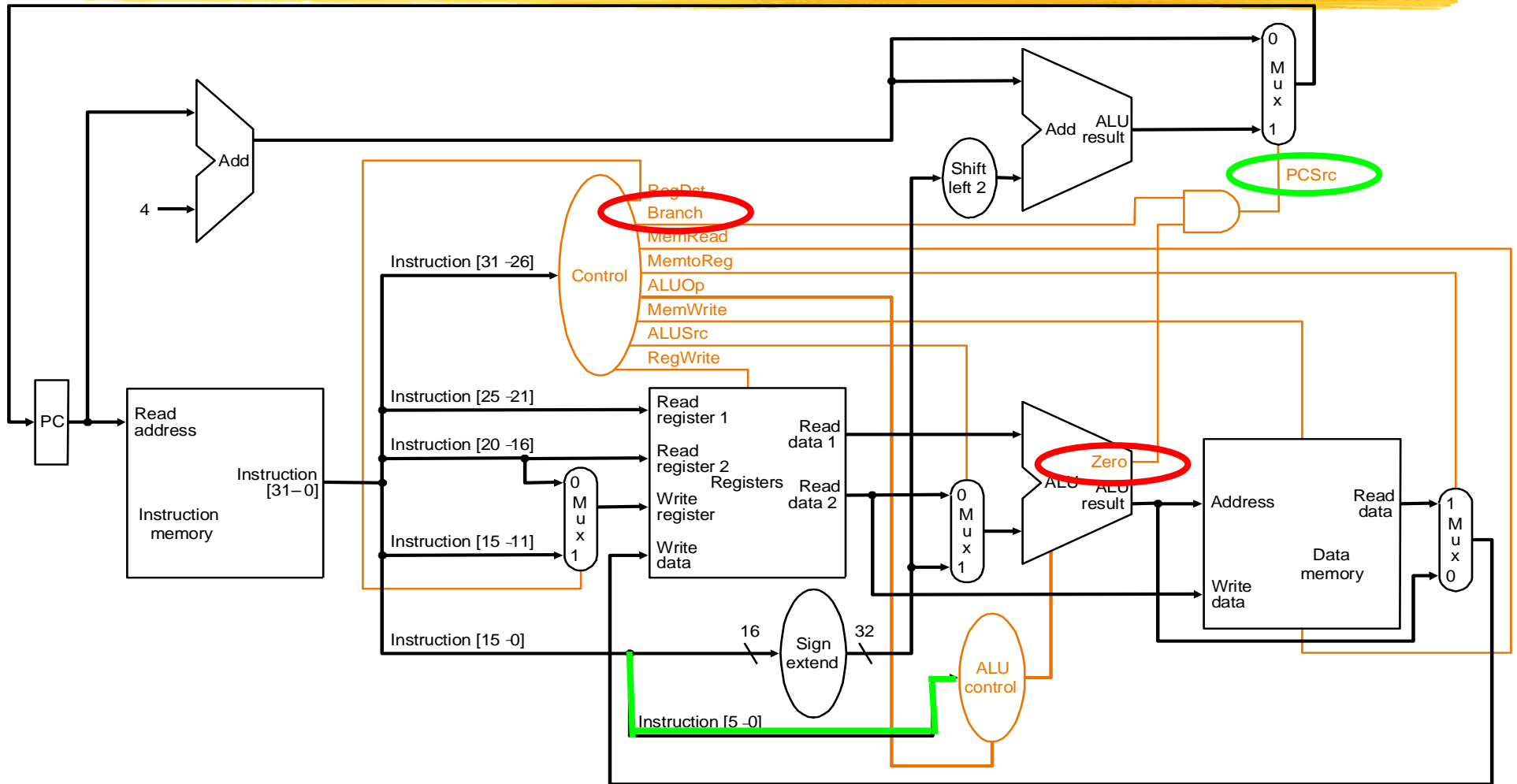
Instruction Fetch at Start of Add

♦ **instruction <- mem[PC];   PC + 4**

National Tsing Hua University

Computer Architecture
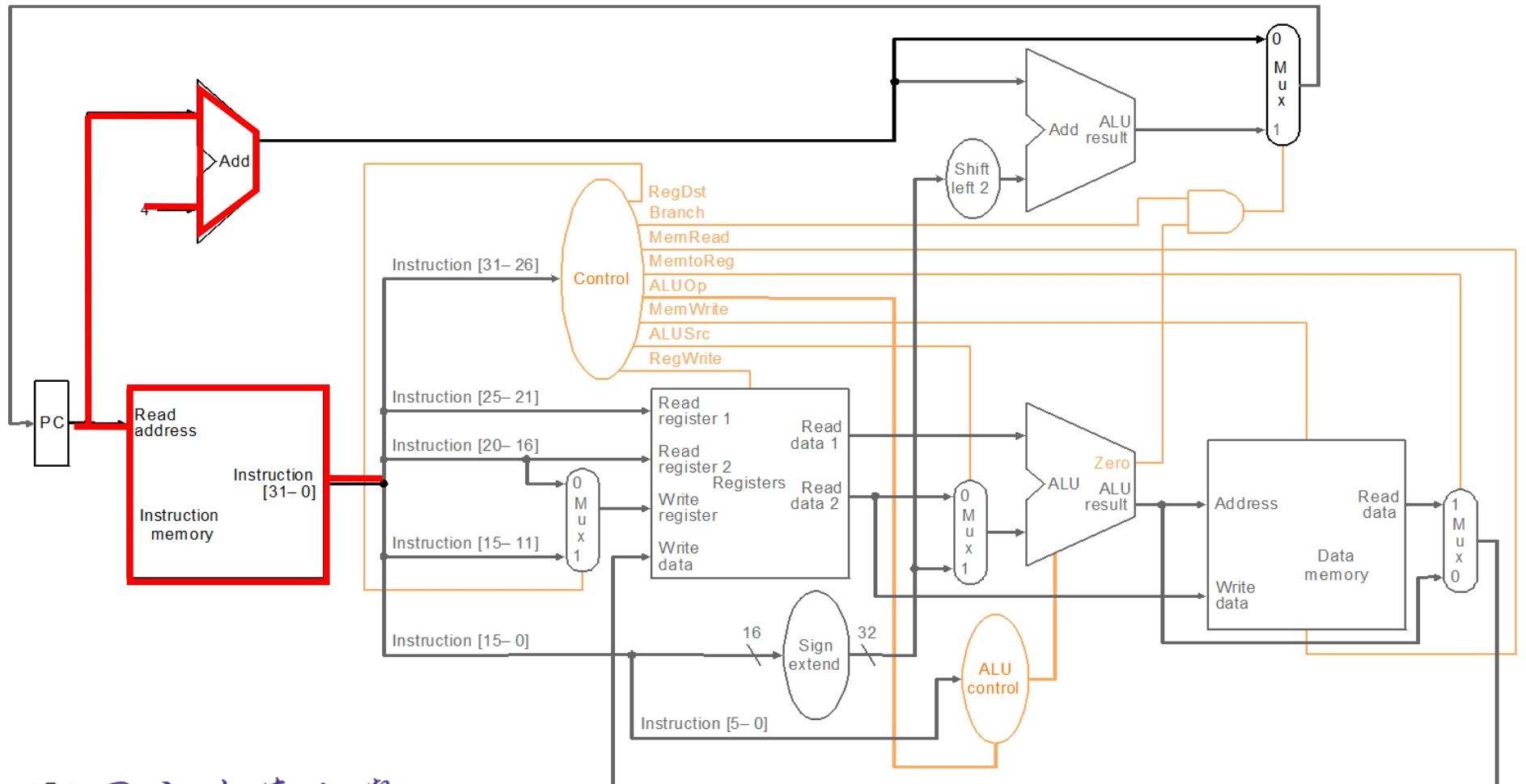
Instruction Decode of Add

♦ **Fetch the two operands and decode instruction:**

Computer Architecture

國立清華大學
National Tsing Hua University
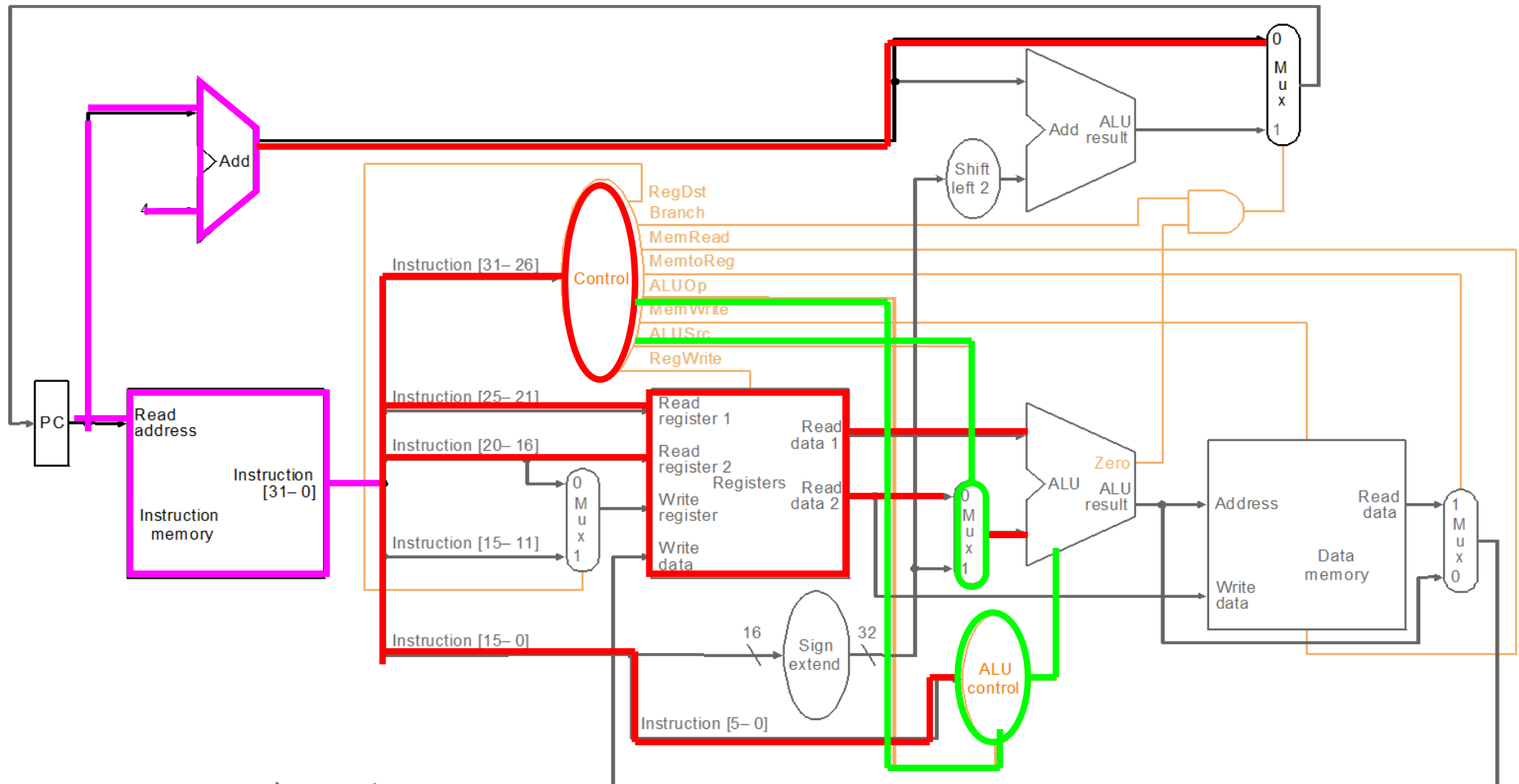
ALU Operation during Add

♦ **R[rs] + R[rt]**

國立清華大學
National Tsing Hua University

Computer Architecture
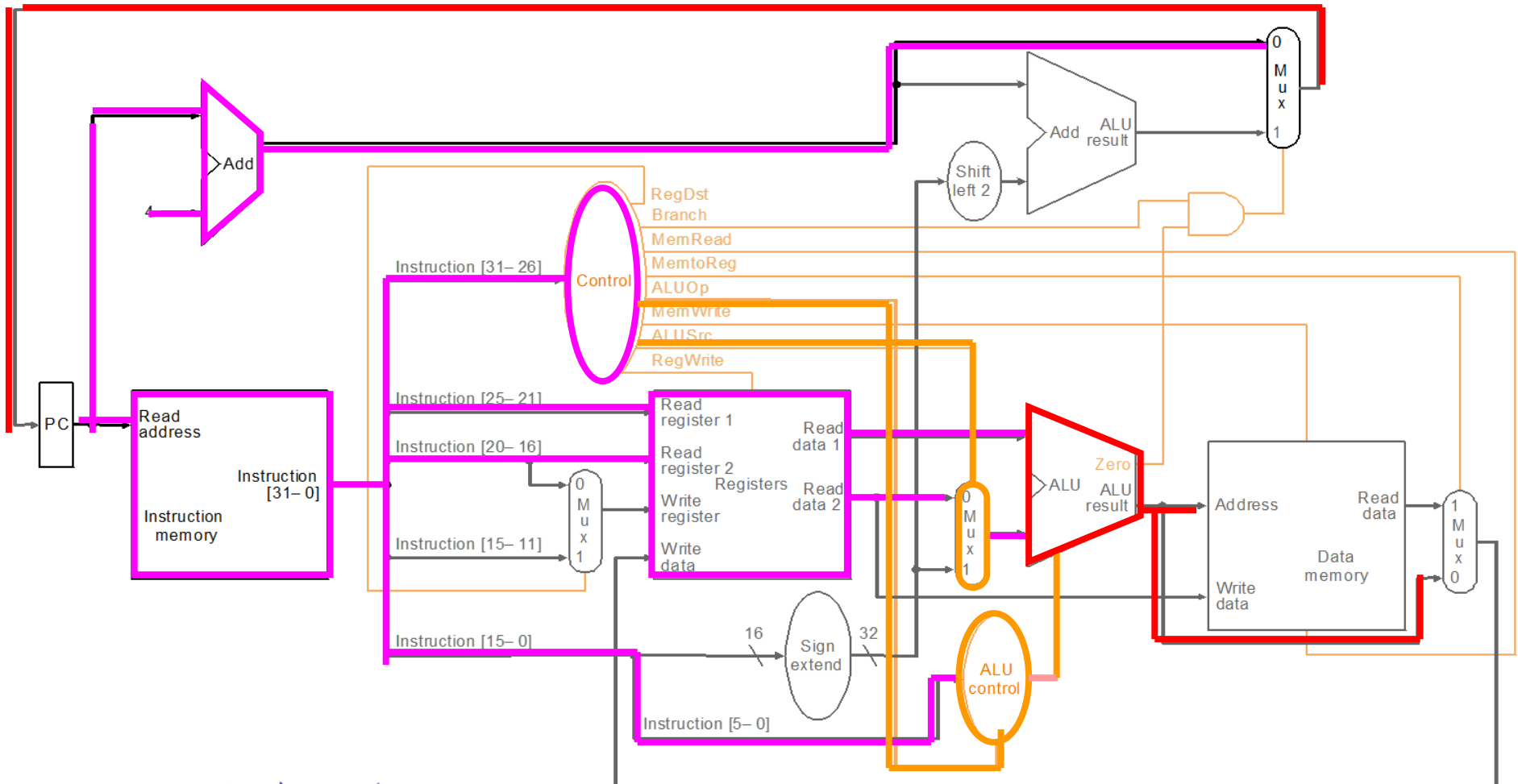
Write Back at the End of Add

♦ **R[rd] <- ALU;   PC <- PC + 4**

國立清華大學
National Tsing Hua University

Computer Architecture
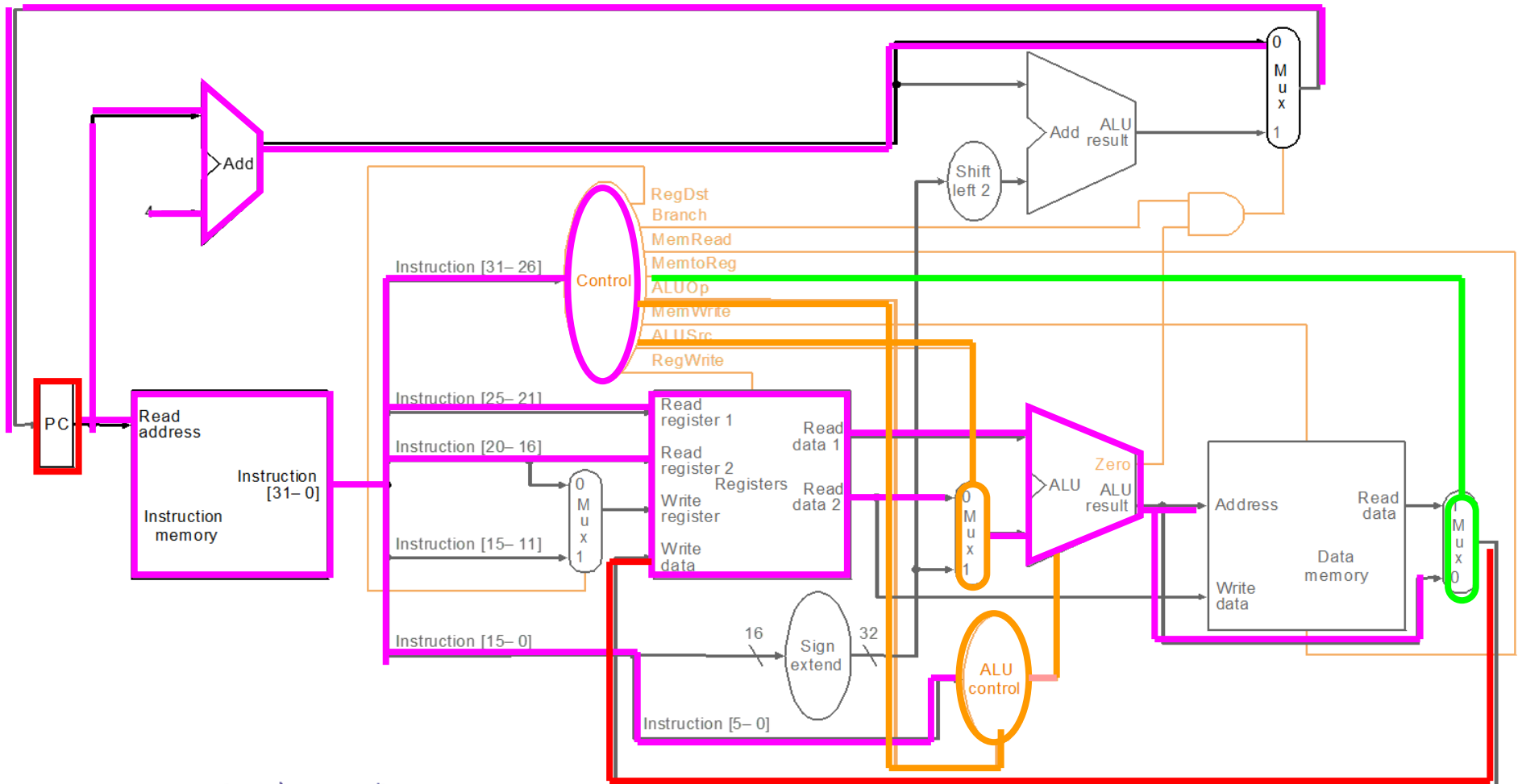
Datapath Operation for lw

♦ **R[rt] <- Memory {R[rs] + SignExt[imm16]}**

國立清華大學
National Tsing Hua University
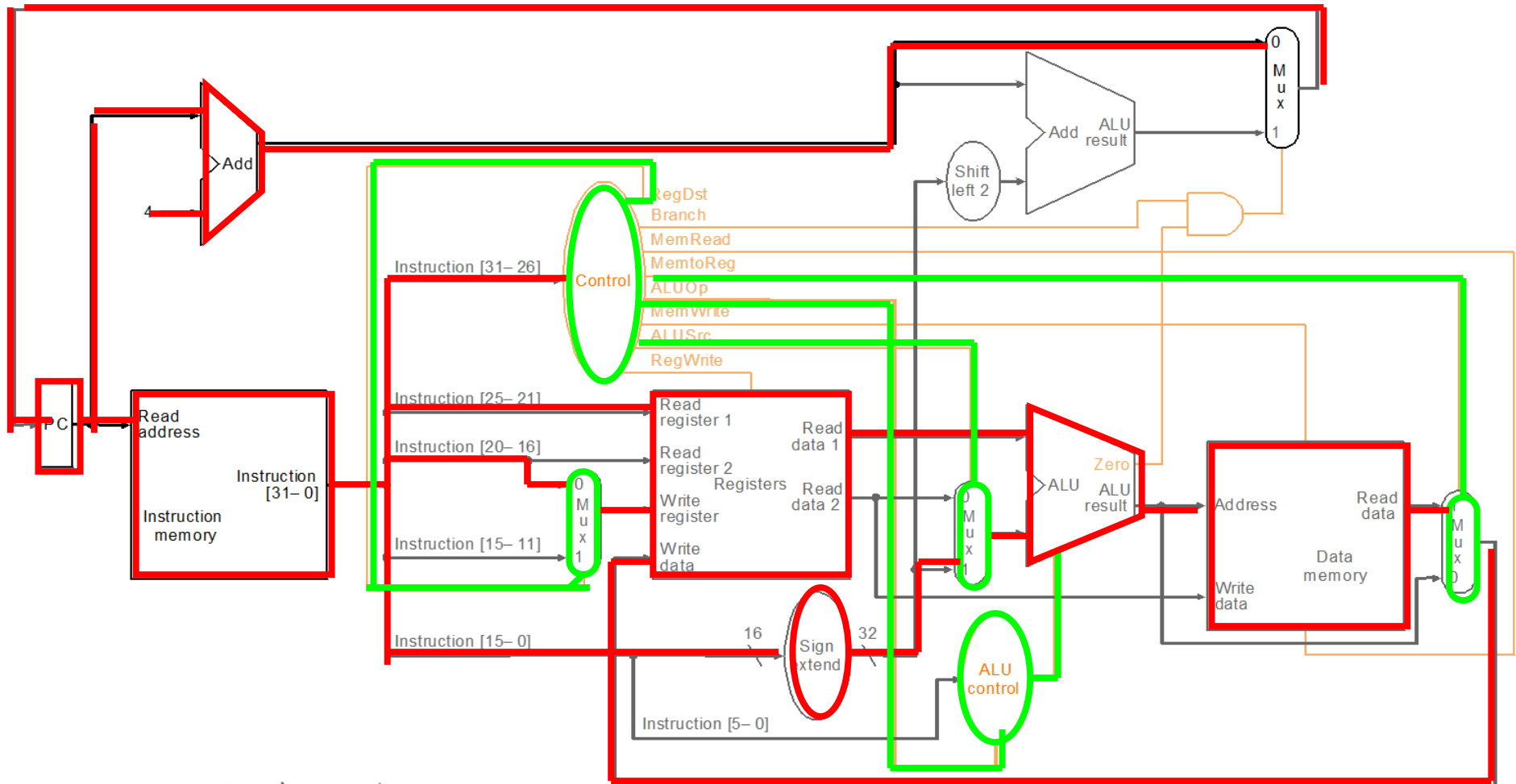
Datapath Operation for beq

if (R[rs]-R[rt]==0) then Zero<-1 else Zero<-0
if (Zero==1) then PC=PC+4+signExt[imm16]*4; else PC = PC + 4

國立清華大學
National Tsing Hua University
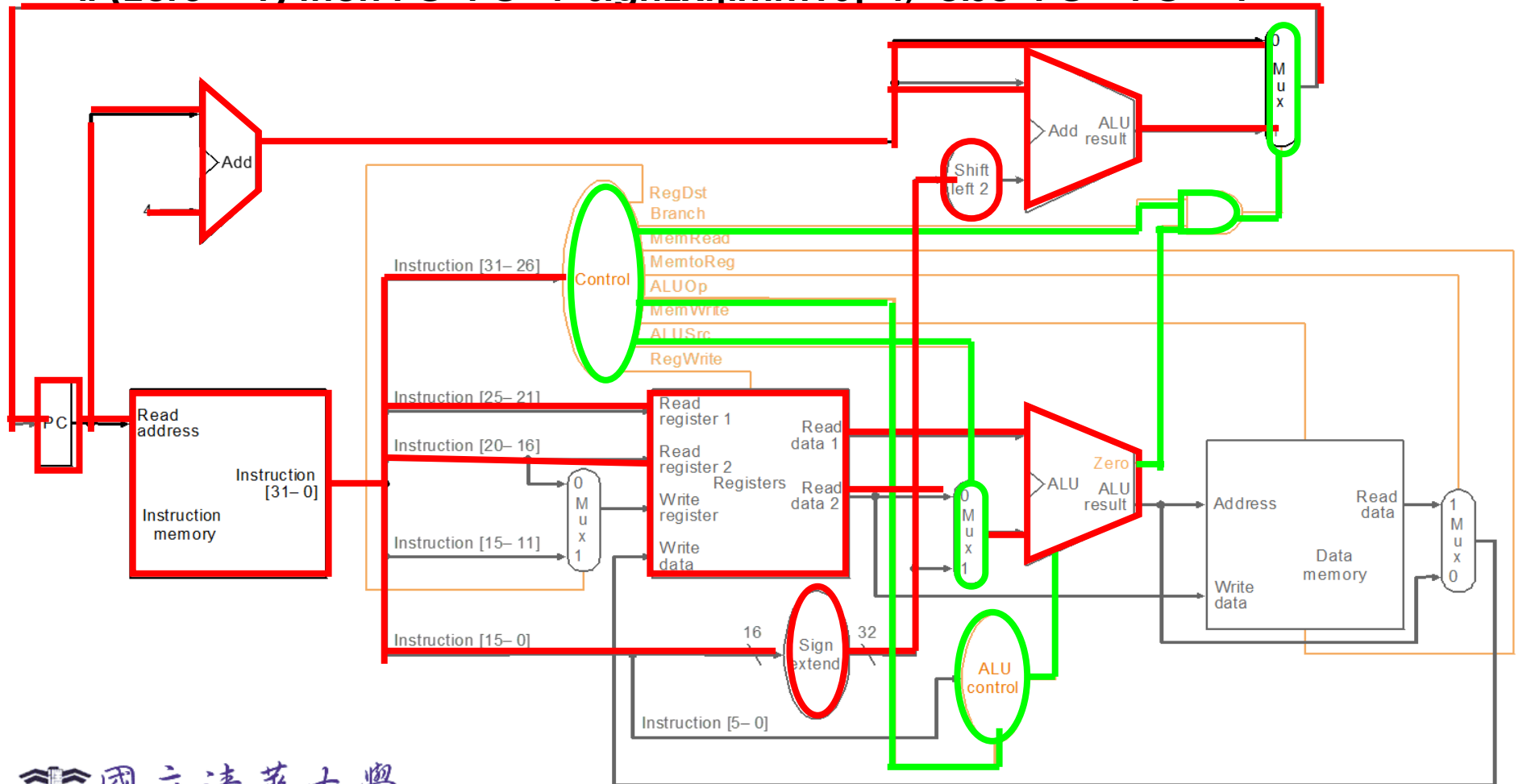
Outline

♦ **Designing a processor**

♦ **Analyzing the instruction set**

♦ **Building the datapath**

♦ **A single-cycle implementation**

♦ **Control for the single-cycle CPU**

   ● **Control of CPU operations**

   ● **ALU controller (step 5a)**

   ● **Main controller**

國立清華大學
National Tsing Hua University

Computer Architecture

Datapath with Control Unit

國立清華大學
National Tsing Hua University
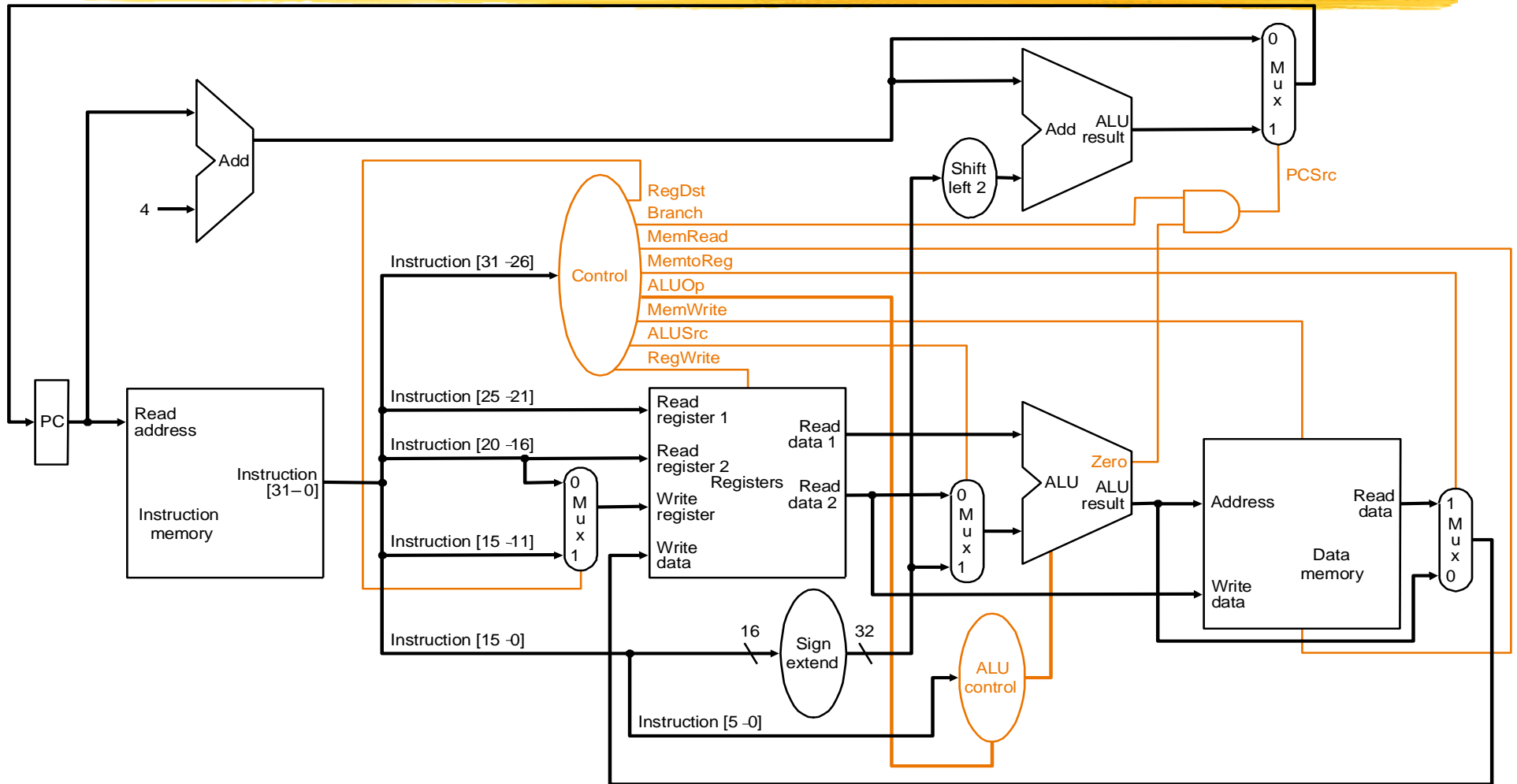
Step 5a: ALU Control

♦ **ALU used for**
  - **Load/Store: F = add**
  - **Branch: F = subtract**
  - **R-type: F depends on funct field**

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

Our Plan for the Controller
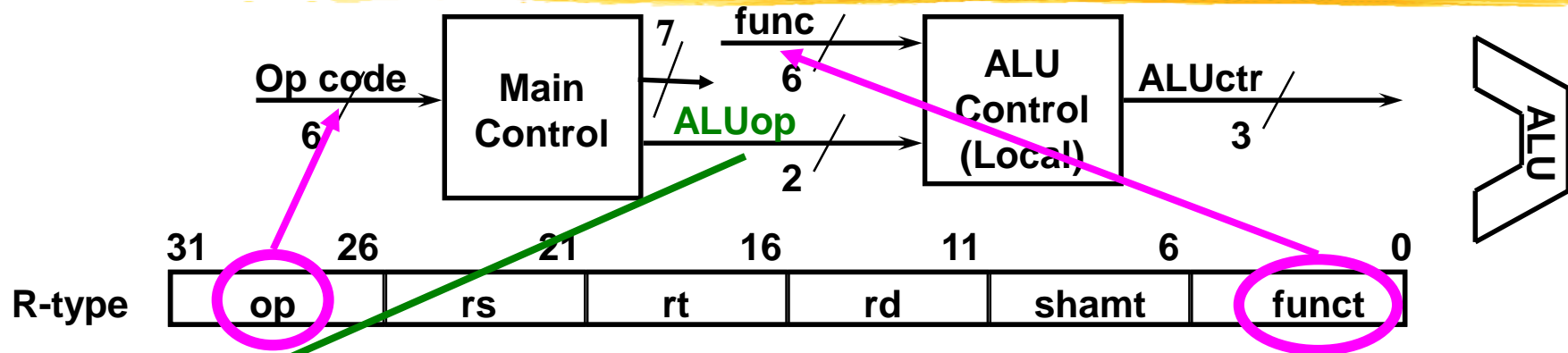
- ♦ **ALUop** is 2-bit wide to represent:
  - ● "I-type" requiring the ALU to perform:
    - ■ (00) add for load/store and (01) sub for beq
  - ● "R-type" (10), need to reference `func` field

	R-type	lw	sw	beq	jump
ALUop (Symbolic)	"R-type"	Add	Add	Subtract	xxx
ALUop<1:0>	10	00	00	01	xxx

ALU Control

♦ **Assume 2-bit ALUOp derived from opcode**
  ● **Combinational logic derives ALU control**

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

國立清華大學
National Tsing Hua University

Computer Architecture

Logic Equation for ALUctr

ALUop		func							ALUctr			
bit<1>	bit<0>	bit<5>	bit<4>	bit<3>	bit<2>	bit<1>	bit<0>		bit<3>	bit<2>	bit<1>	bit<0>
0	0	x	x	x	x	x	x		0	0	1	0
x	1	x	x	x	x	x	x		0	1	1	0
1	x	x	x	0	0	0	0		0	0	1	0
1	x	x	x	0	0	1	0		0	1	1	0
1	x	x	x	0	1	0	0		0	0	0	0
1	x	x	x	0	1	0	1		0	0	0	1
1	x	x	x	1	0	1	0		0	1	1	1

國立清華大學
National Tsing Hua University

Computer Architecture

Logic Equation for ALUctr2

ALUop		func						ALUctr<2>
bit<1>	bit<0>	bit<5>	bit<4>	bit<3>	bit<2>	bit<1>	bit<0>	
x	1	x	x	x	x	x	x	1
1	x	x	x	0	0	1	0	1
1	x	x	x	1	0	1	0	1

This makes func<3> a don't care

ALUctr2 = ALUop0
+ ALUop1·func2'·func1·func0'

國立清華大學
National Tsing Hua University

Computer Architecture

Logic Equation for ALUctr1

ALUop		func						ALUctr<1>
bit<1>	bit<0>	bit<5>	bit<4>	bit<3>	bit<2>	bit<1>	bit<0>	
0	0	x	x	x	x	x	x	1
x	1	x	x	x	x	x	x	1
1	x	x	x	0	0	0	0	1
1	x	x	x	0	0	1	0	1
1	x	x	x	1	0	1	0	1

$$ALUctr1 = ALUop1'$$
$$+ ALUop1 \cdot func2' \cdot func0'$$

Logic Equation for ALUctr0

ALUop		func						ALUctr<0>
bit<1>	bit<0>	bit<5>	bit<4>	bit<3>	bit<2>	bit<1>	bit<0>	
1	x	x	x	0	1	0	1	1
1	x	x	x	1	0	1	0	1

ALUctr0 = ALUop1
 ·func3'·func2·func1'·func0
 + ALUop1'·func3
 ·func2'·func1·func0'

National Tsing Hua University

Computer Architecture

The Resultant ALU Control Block
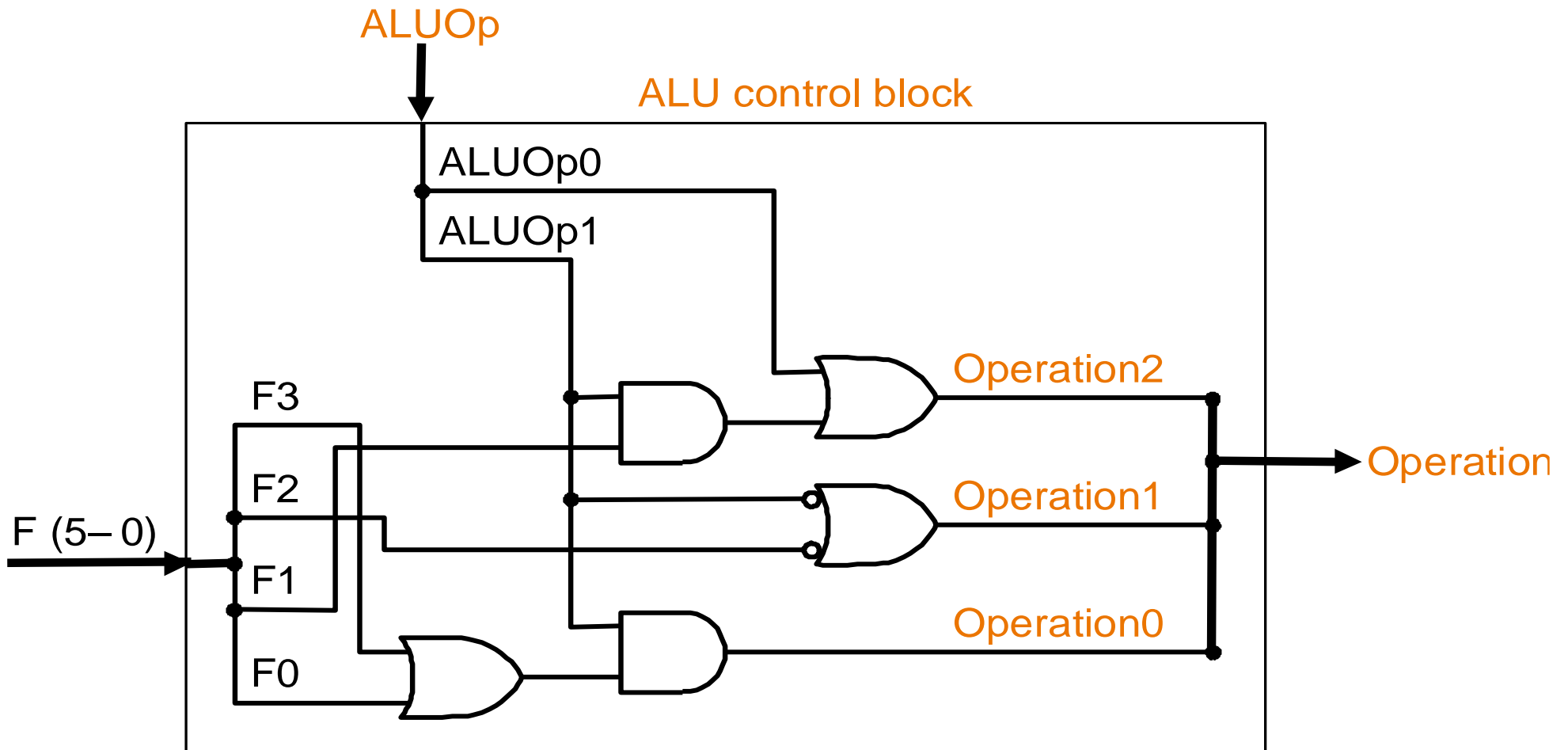
National Tsing Hua University

Outline

♦ **Introduction to designing a processor**
♦ **Analyzing the instruction set**
♦ **Building the datapath**
♦ **A single-cycle implementation**
♦ **Control for the single-cycle CPU**
  ● **Control of CPU operations**
  ● **ALU controller**
  ● **Main controller (step 5b)**

Step 5b: The Main Control Unit

♦ **Control signals derived from instruction**

R-type	0	rs	rt	rd	shamt	funct
	31:26	25:21	20:16	15:11	10:6	5:0

Load/ Store	35 or 43	rs	rt	address		
	31:26	25:21	20:16	15:0		

Branch	4	rs	rt	address		
	31:26	25:21	20:16	15:0		

opcode   **always read**   **read, except for load**   **write for R-type and load**   **sign-extend and add**
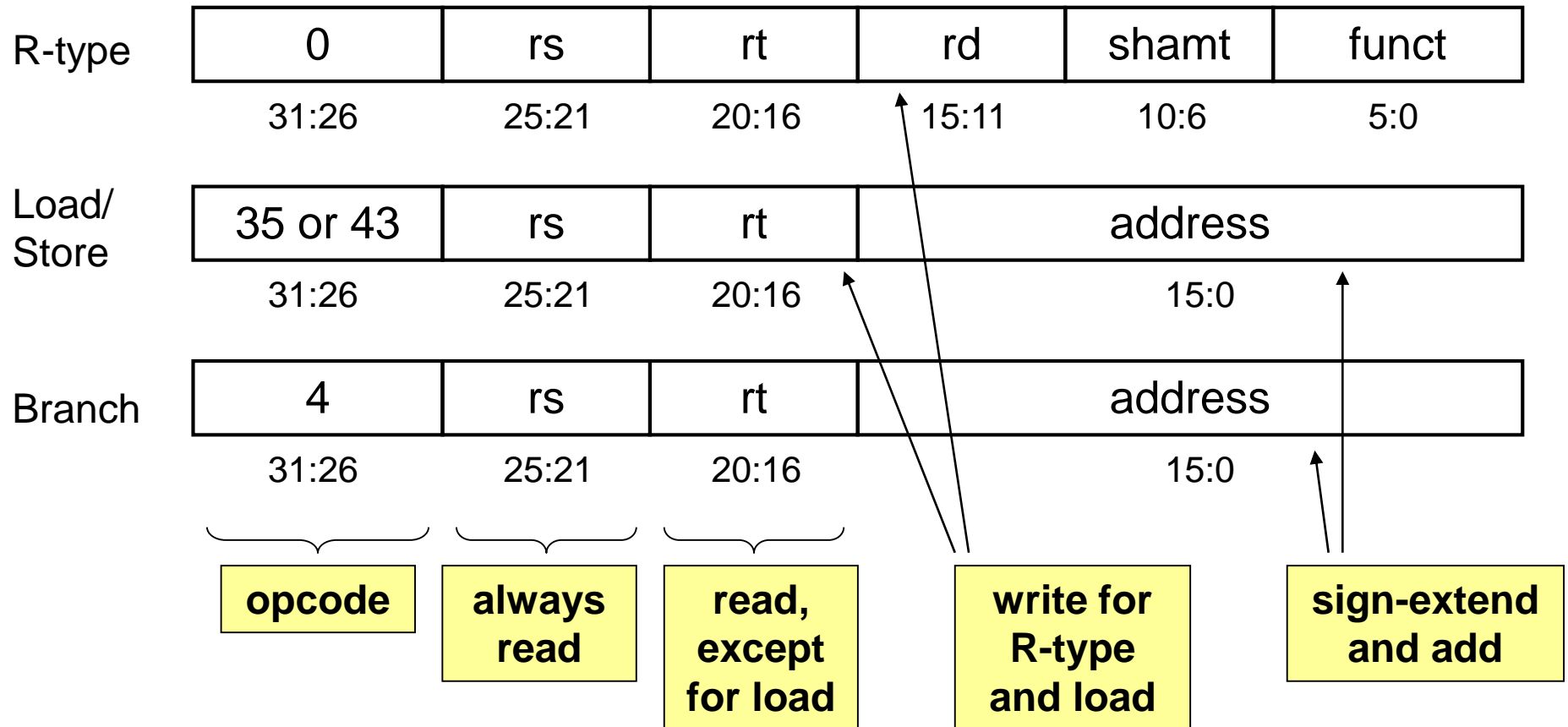
Truth Table of Control Signals

See ——→ func	10 0000	10 0010	We Don't Care :-)		
Appendix A ——→ op	00 0000	00 0000	10 0011	10 1011	00 0100
	add	**sub**	**lw**	**sw**	**beq**
RegDst	1	1	0	x	x
ALUSrc	0	0	1	1	0
MemtoReg	0	0	1	x	x
RegWrite	1	1	1	0	0
MemRead	0	0	1	0	0
MemWrite	0	0	0	1	0
Branch	0	0	0	0	1
ALUop1	1	1	0	0	0
ALUop0	0	0	0	0	1

Op code /6 → **Main Control** →
- **RegDst**
- **ALUSrc**
- **:**
- **ALUop** /2

func /6 → **ALU Control (Local)** → **ALUctr** /4

國立清華大學
National Tsing Hua University

Computer Architecture
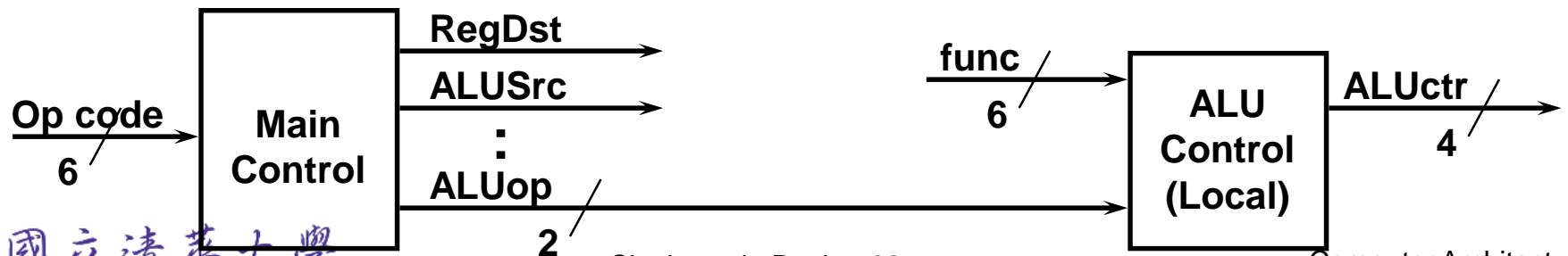
Truth Table for RegWrite

Op code	00 0000	10 0011	10 1011	00 0100
	R-type	lw	sw	beq
RegWrite	1	1	0	0

RegWrite = R-type + lw

= op5'·op4'·op3'·op2'·op1'·op0'   (R-type)

+ op5·op4'·op3'·op2'·op1· op0 (lw)

Computer Architecture

國立清華大學
National Tsing Hua University
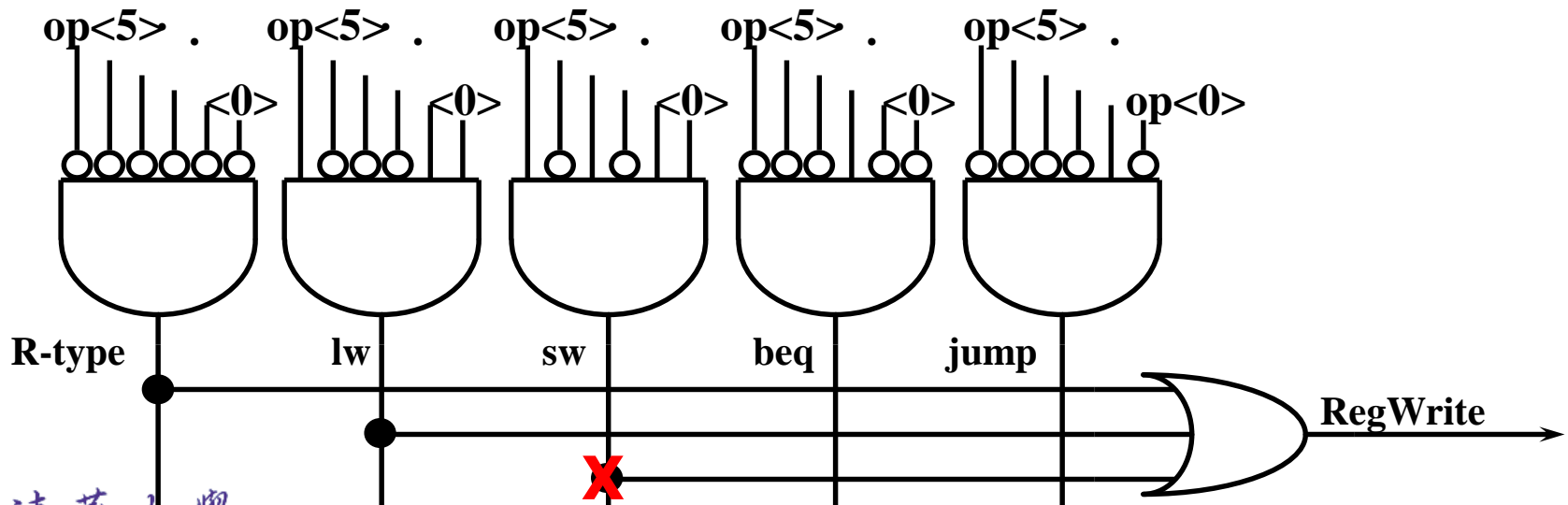
PLA Implementing Main Control

National Tsing Hua University
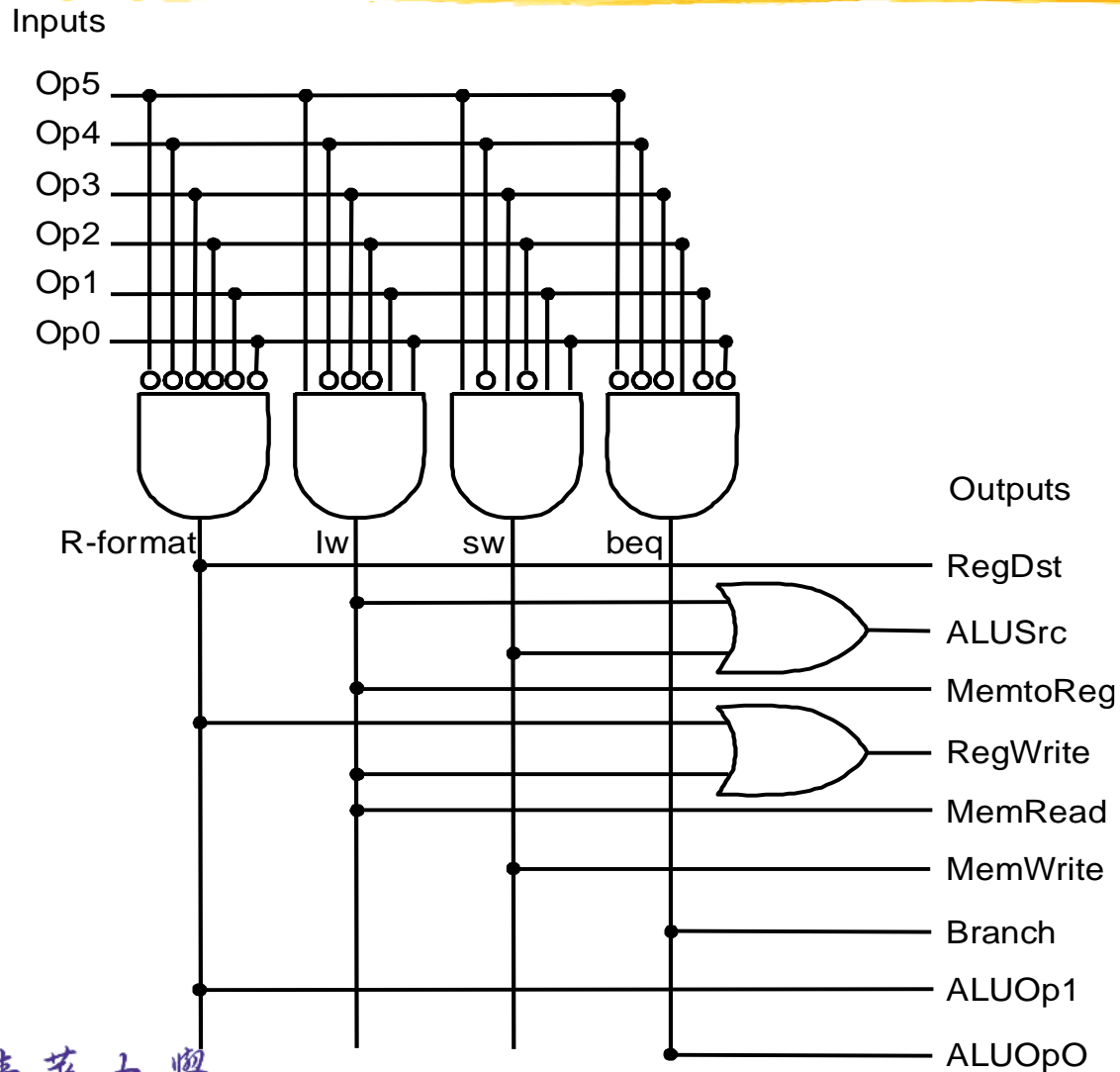
Implementing Jumps

Jump	2	address
	31:26	25:0
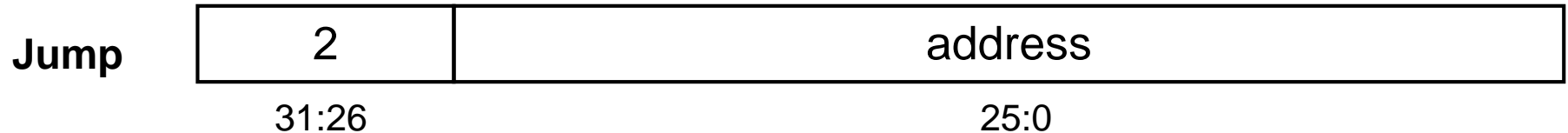
- ◆ **Jump uses word address**
- ◆ **Update PC with concatenation of**
  - **Top 4 bits of old PC**
  - **26-bit jump address**
  - **00**
- ◆ **Need an extra control signal decoded from opcode**

Putting it Altogether (+ jump instruction)

國立清華大學
National Tsing Hua University

Worst Case Timing (Load)

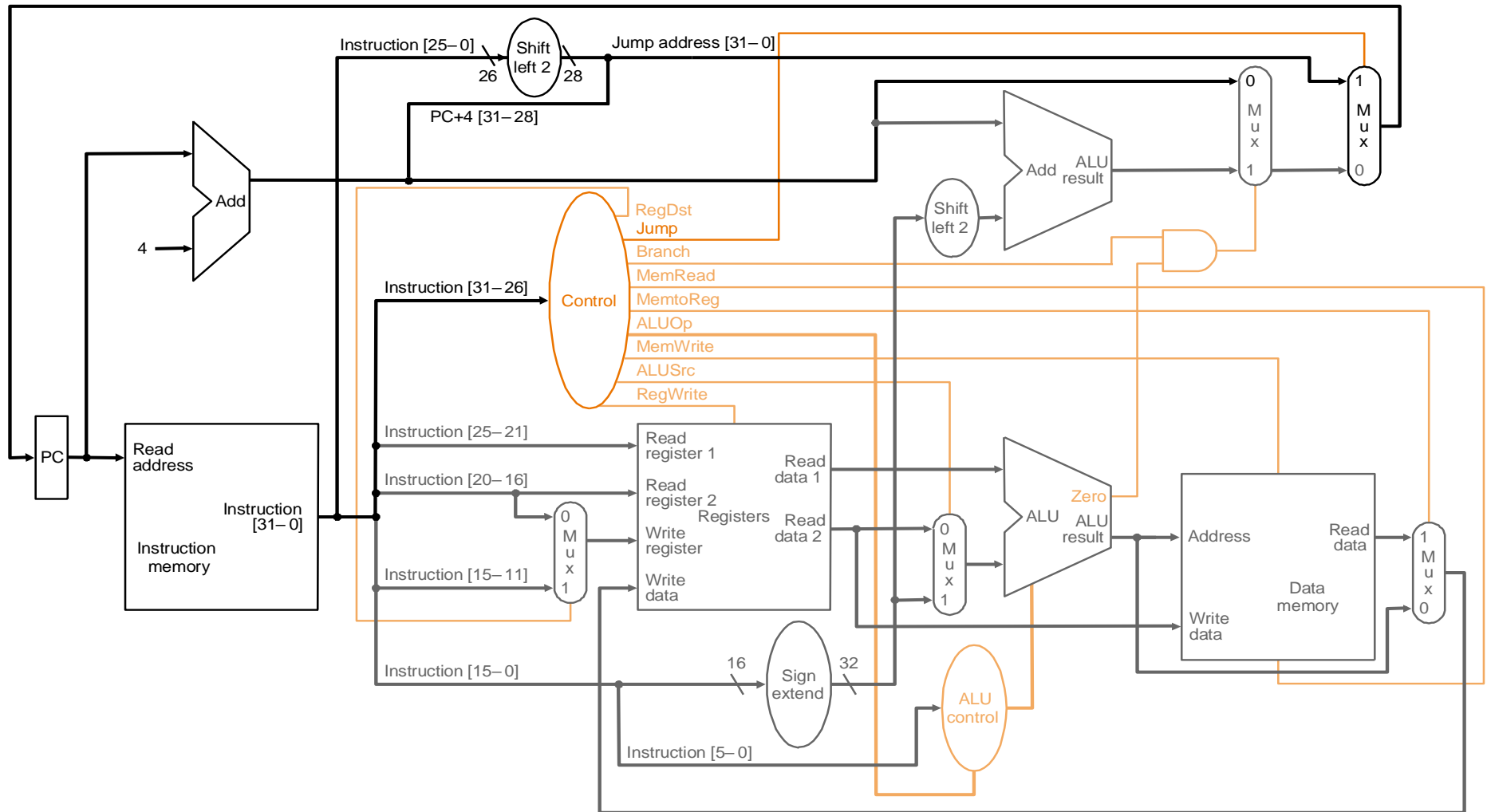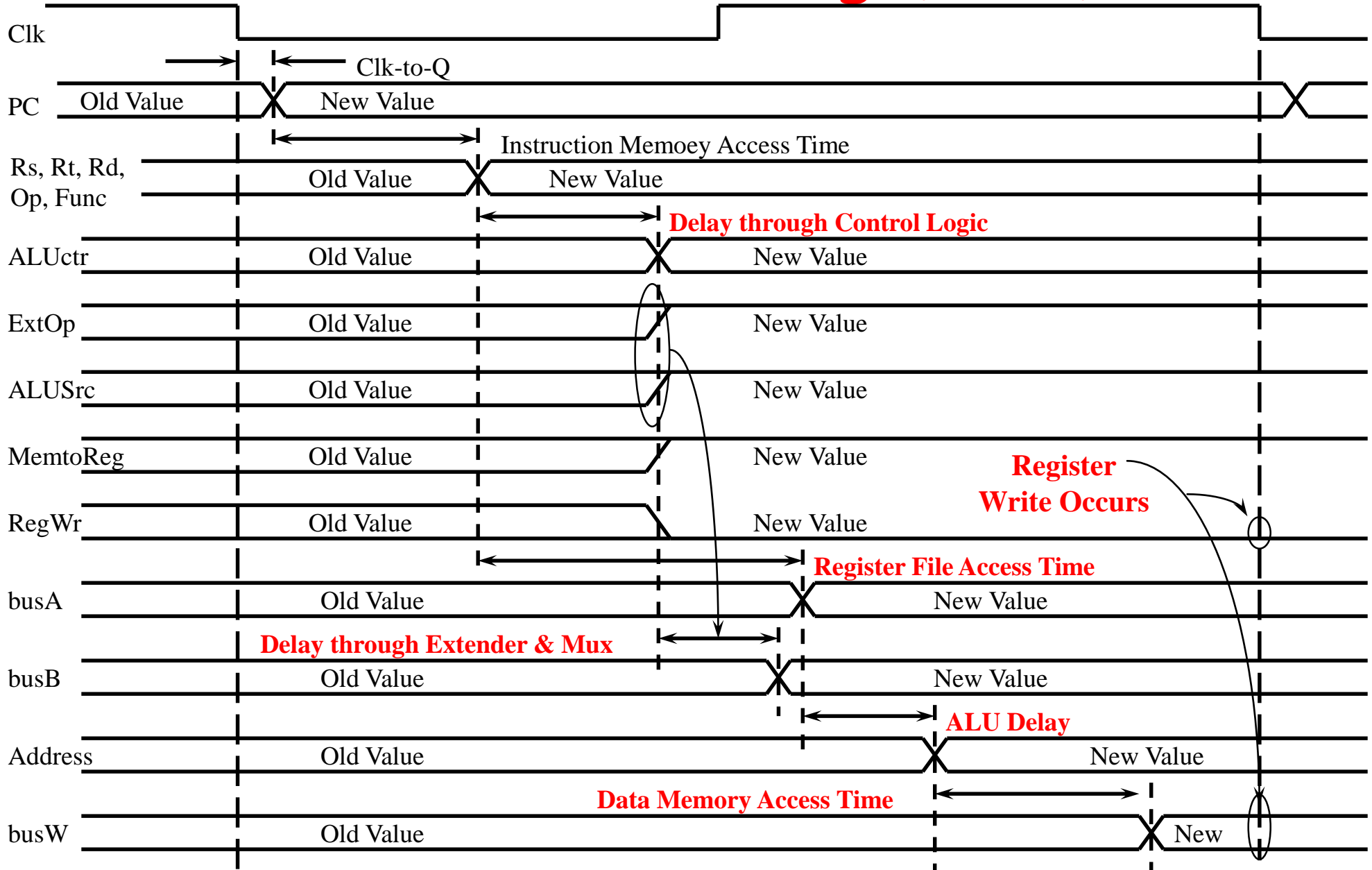| Signal		
Clk		
PC	Old Value	New Value
Rs, Rt, Rd, Op, Func	Old Value	New Value
ALUctr	Old Value	New Value
ExtOp	Old Value	New Value
ALUSrc	Old Value	New Value
MemtoReg	Old Value	New Value
RegWr	Old Value	New Value
busA	Old Value	New Value
busB	Old Value	New Value
Address	Old Value	New Value
busW	Old Value	New

Clk-to-Q

Instruction Memoey Access Time

Delay through Control Logic

Register Write Occurs

Register File Access Time

Delay through Extender & Mux

ALU Delay

Data Memory Access Time

Drawback of Single-Cycle Design

◆ **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**

國立清華大學
National Tsing Hua University

Computer Architecture

Summary

♦ **Single cycle datapath => CPI=1, Clock cycle time long**

♦ **MIPS makes control easier**
  - **Instructions same size**
  - **Source registers always in same place**
  - **Immediates same size, location**
  - **Operations always on registers/immediates**

Computer Architecture

National Tsing Hua University