COMPUTER ORGANIZATION

Lecture 8
The Processor

2025 Spring

baiyh@sustech.edu.cn

Recap

- Lec7
  - Representation
    - Normalized scientific: $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
    - IEEE 754: $x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$
  - IEEE 754
    - Format
      - Sign: 1 negative, 0 positive
      - Exponent: true exponent + bias => exponent in register
      - Fraction: significand has a hidden 1
    - Range
      - Overflow and Underflow when out of range (-2x2^127, -1x2^-126], [1x2^-126, 2x2^127)
      - biased exponent = 0, zero or denormalized
      - biased exponent = 255, infinity or NaN
    - Precision
      - Relative Precision: single: ~2^-23(6 decimal digits); double: ~ 2^-52(16 decimal digits)
      - Gard bit, Round bit, Sticky bit
  - Arithmetic
    - Add/Sub
      - 1. Align (small exponent -> big exponent; 2. Add significands; 3. Normalize 4. Round and renormalize if necessary
    - Multiplication
      - 1. Add exponents(careful for duplicate bias); 2. Multiply significands; 3. Normalize 4. Round and renormalize if necessary; 5. Determine sign
  - FP instructions

Introduction

$$\text{CPU Time} = \text{Instructions} \times \text{CPI} \times \text{Clock Period}$$
$$= \frac{\text{Instructions} \times \text{CPI}}{\text{Clock Rate}}$$

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two RISC-V implementations
  - A simplified version
  - A more realistic pipelined version

How to Design a Processor?

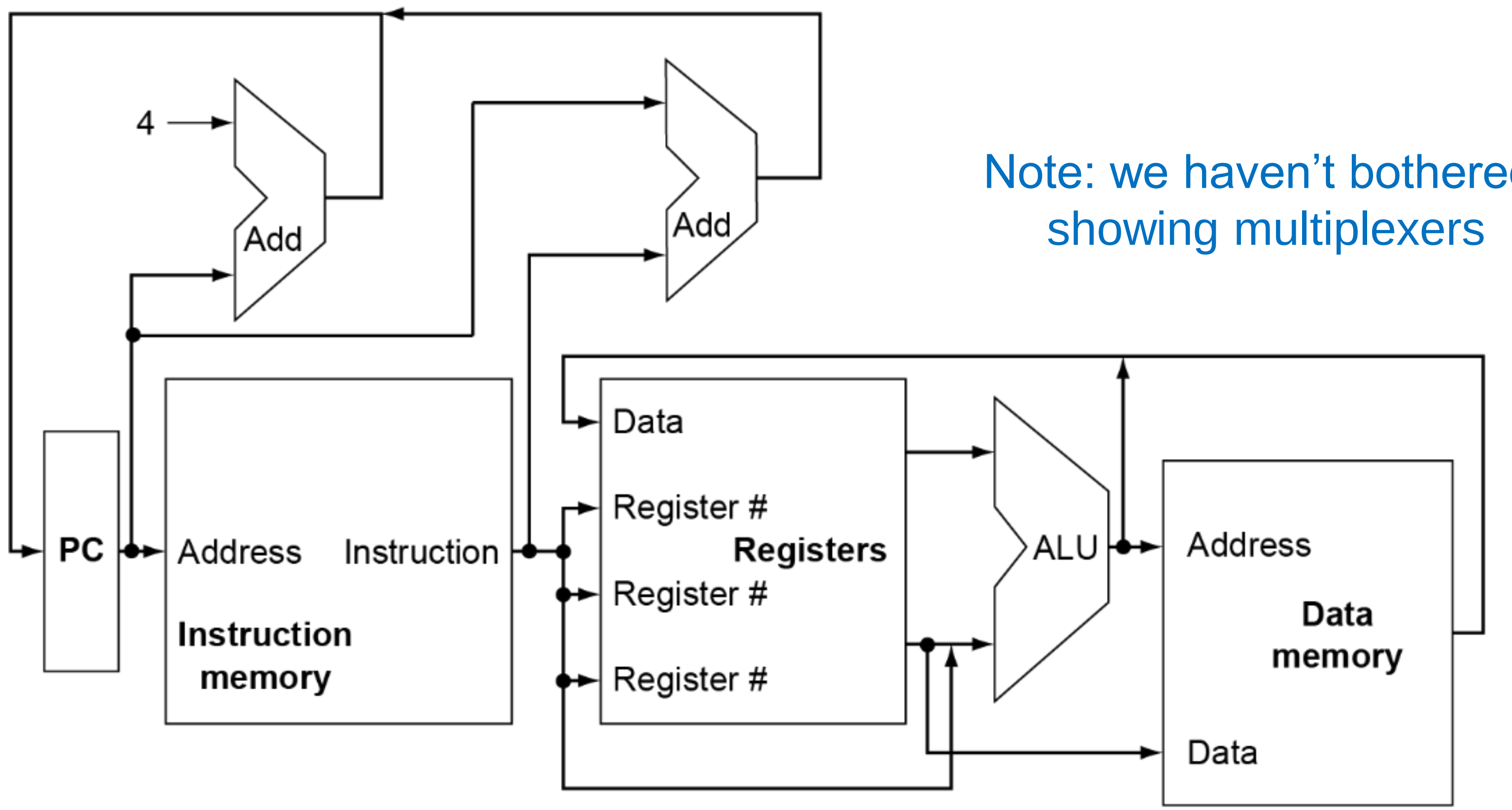
1. Analyze instruction set
   - To find out the datapath requirements
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic

Step 1: Analyze Instruction Set

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

- Implementation Overview
  - We need memory
    - to store instructions
    - to store data
    - for now, let's make them separate units
  - We need registers, ALU, and a lot of control logic
  - CPU operations common to all instructions:
    - use the program counter (PC) to pull instruction out of instruction memory
    - read register values

CPU Overview



Note: we haven't bothered showing multiplexers

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

How to Design a Processor?

1. Analyze instruction set
2. Select set of datapath components and establish clocking methodology
   - combinational and sequential elements needed in datapath
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

Step 2: Select Datapath Building Units

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, MUXs, Memories, …
- We will build a RISC-V datapath incrementally
  - Refining the overview design

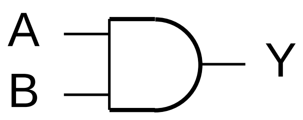
Logic Design Basics

- 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

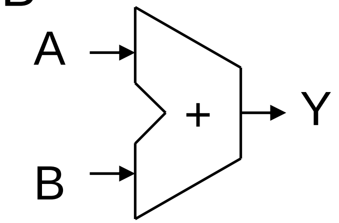
Combinational Elements

- Basic building blocks of combinational logic elements :

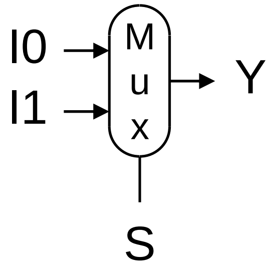
  - AND-gate
    - Y = A & B



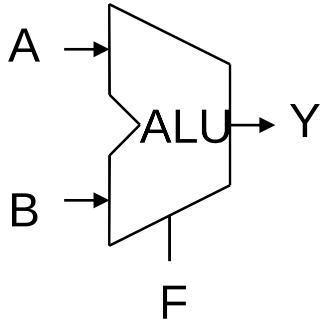
  - Adder
    - Y = A + B



  - Multiplexer
    - Y = S ? I1 : I0

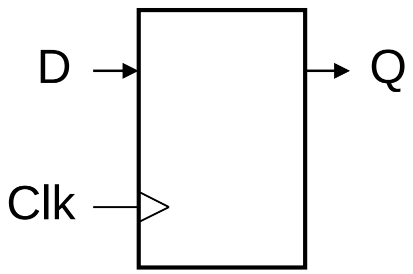


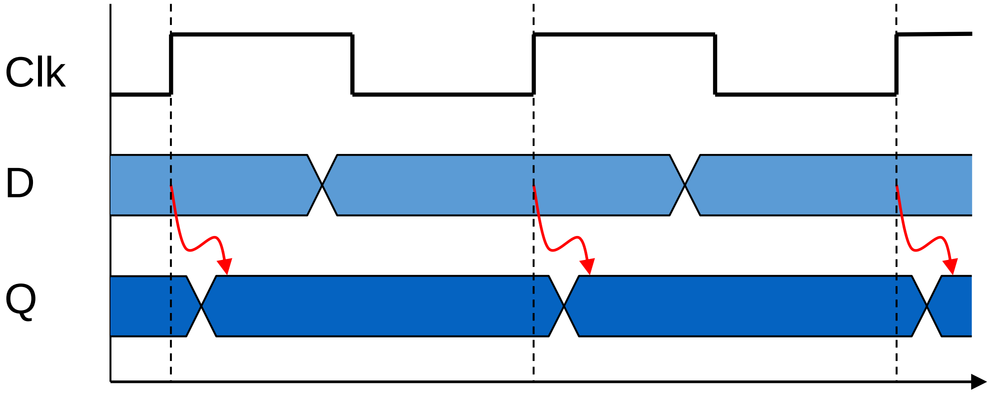
  - Arithmetic/Logic Unit
    - Y = F(A, B)



State Elements (sequential elements)

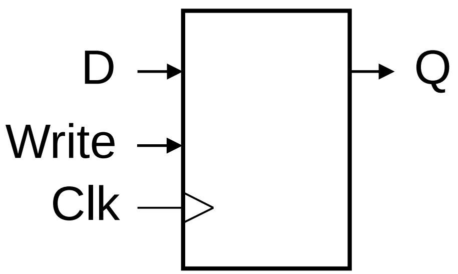
- State element
  - The state element has a pre-stored state
  - It has some internal storage
  - Has at least two inputs and one output (e.g. D-type flip-flop):
    - The data to be written into the element
    - The clock which determines when the data is written
    - The output: the value that was written in earlier cycle
  - Examples: register and memory

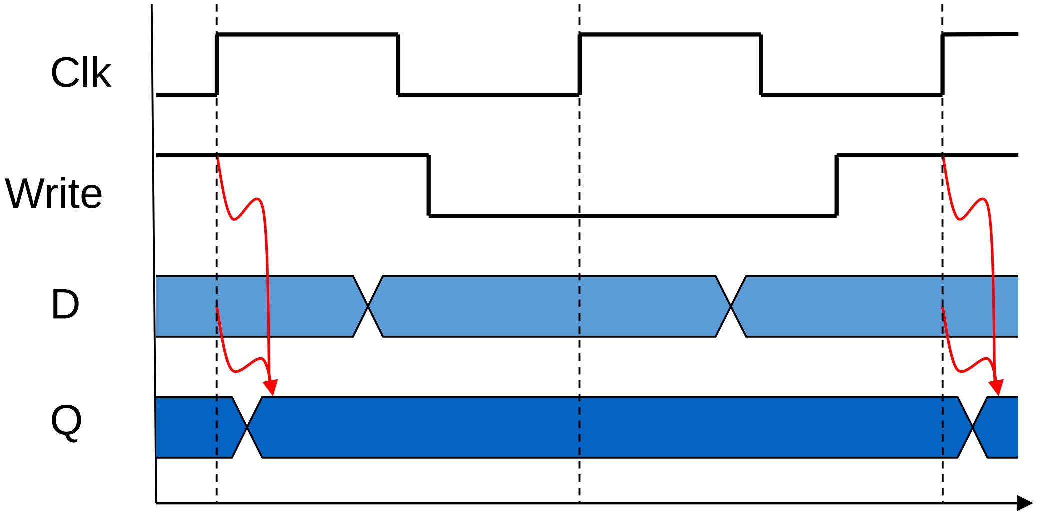




State Elements with write control

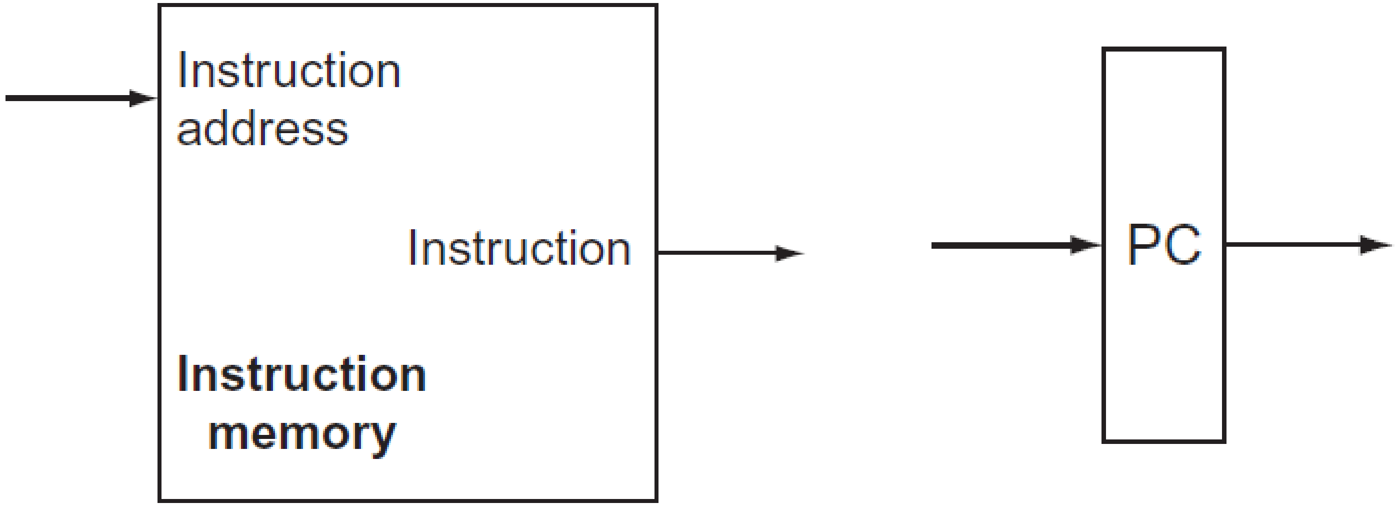
- Register without write control (e.g. program counter)
  - Uses a clock signal to determine when to update
  - Edge-triggered: update when Clk changes from 0 to 1
- Register with write control (e.g. data memory/register)
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later





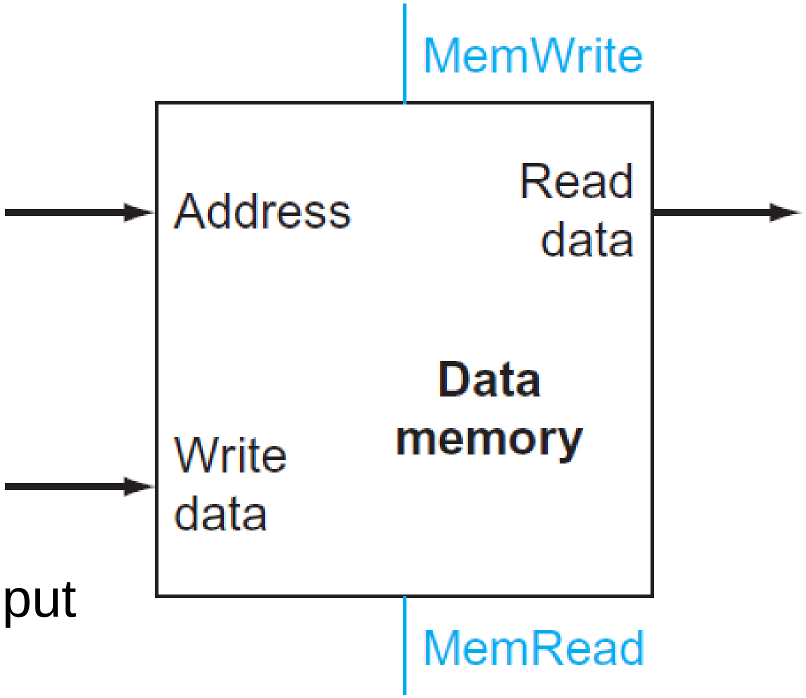
Instruction Memory and PC

- Instruction Memory
  - Input: InstructionAddress (32-bit)
  - Output: Instructions (32-bit)
- Program Counter
  - Input: InstructionAddress (32-bit)
  - Output: InstructionAddress (32-bit)
- No control signals



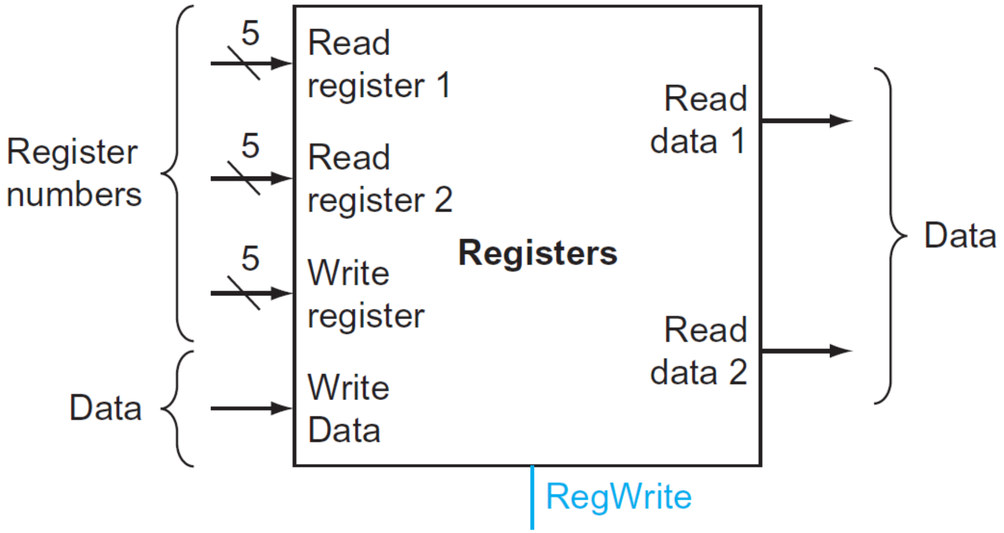
Data Memory

- Data Memory
  - Input:
    - Address(32-bit)
    - write-in data(32-bit)
    - MemWrite (1-bit)
    - MemRead (1-bit)
  - Output:
    - readdata1 (32-bit)
- Word is selected by:
  - Read Enable = 1: Address selects the word to put on Read data bus
  - Write Enable = 1: address selects the memory word to be written via the Write data bus



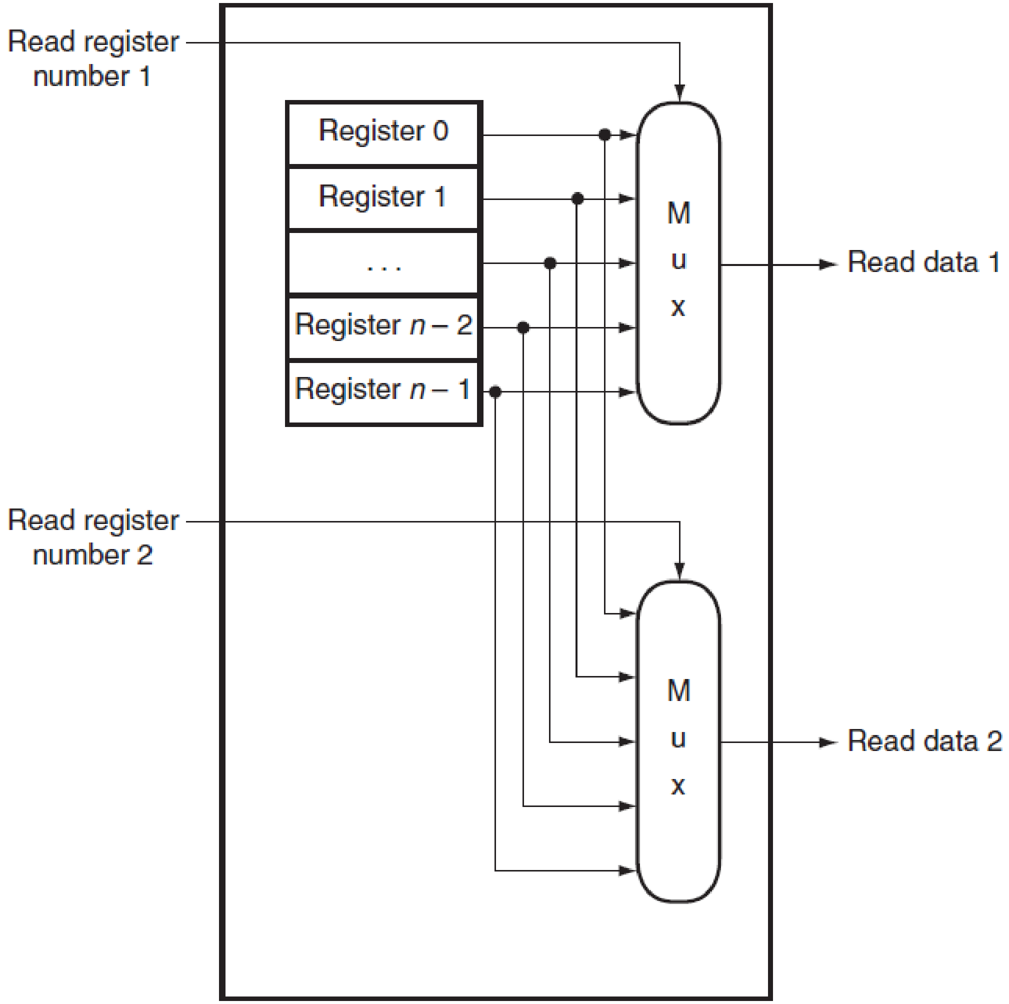
Registers

- Consists of 32 registers (32*32-bit data):
  - Input:
    - three register numbers (5-bit *3)
    - write-in data(32-bit)
    - RegWrite (1-bit)
  - Output:
    - readdata1 (32-bit),
    - readdata2 (32-bit)



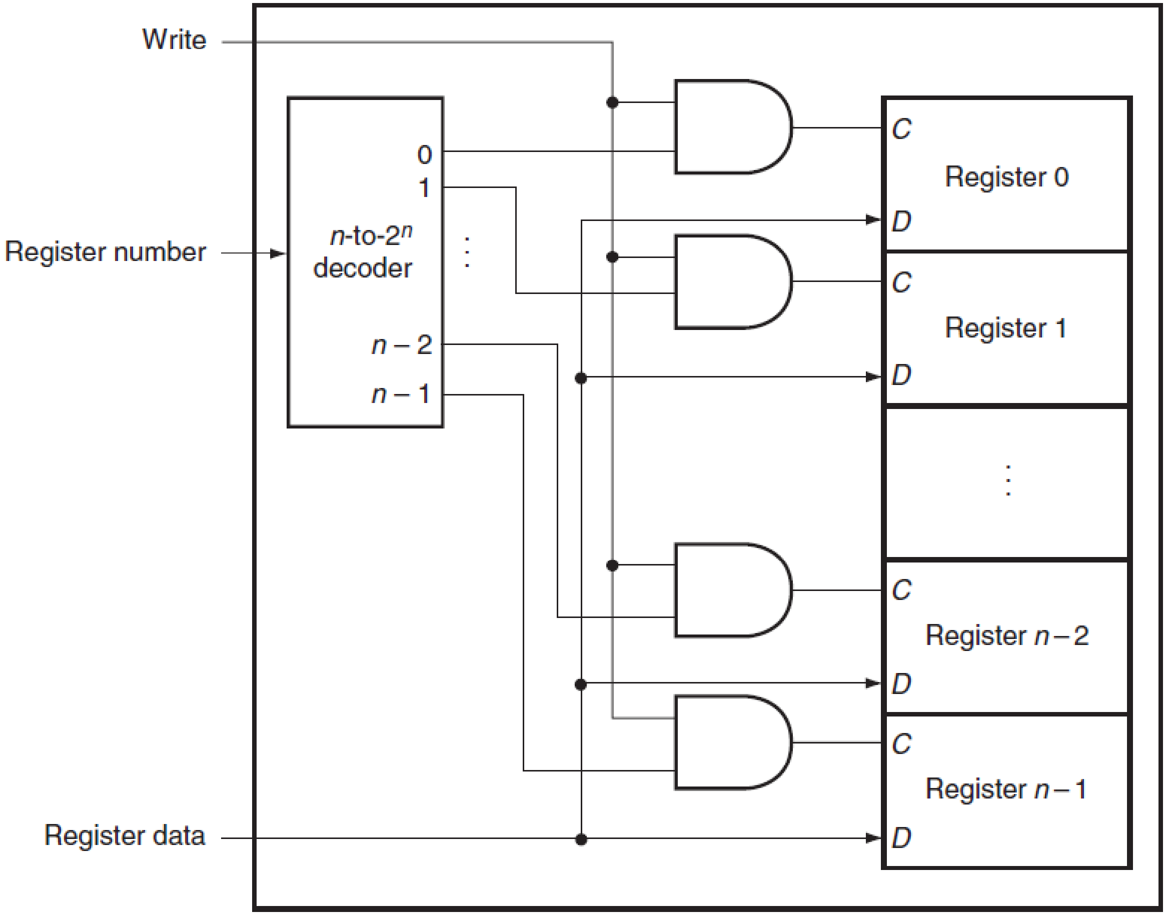
Register Read

- Input: two addresses
- Output: two data
- Key component: two multiplexers



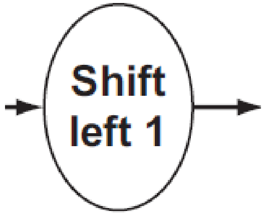
Register Write

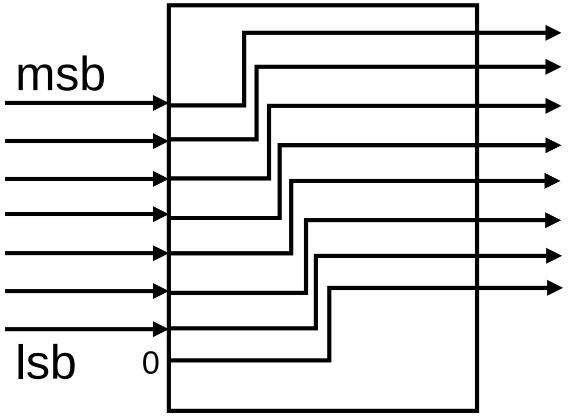
- Input:
  - write control
  - address
  - writing data
- Key component:
  - decoder



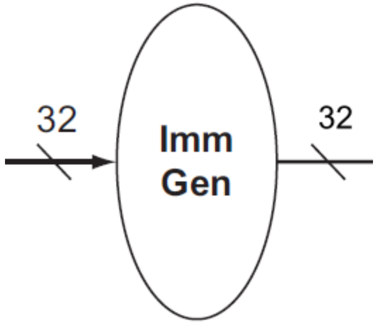
Other Combinational Elements

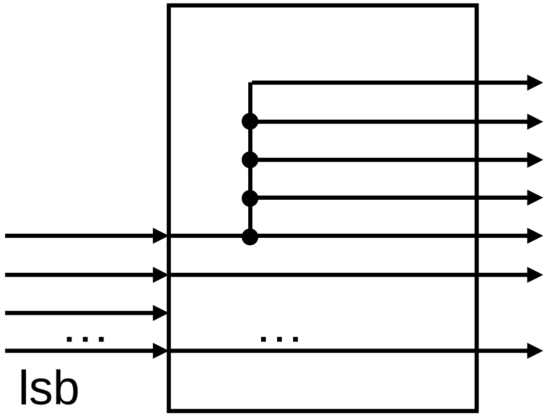
- Other useful Basic building blocks:
  - Shift left 1





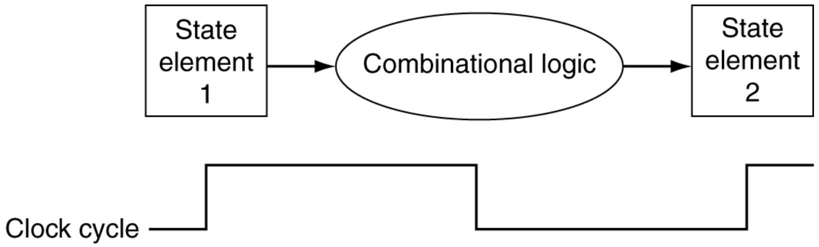
  - Imm-gen

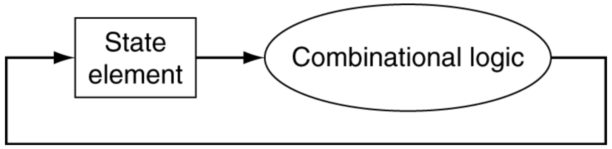




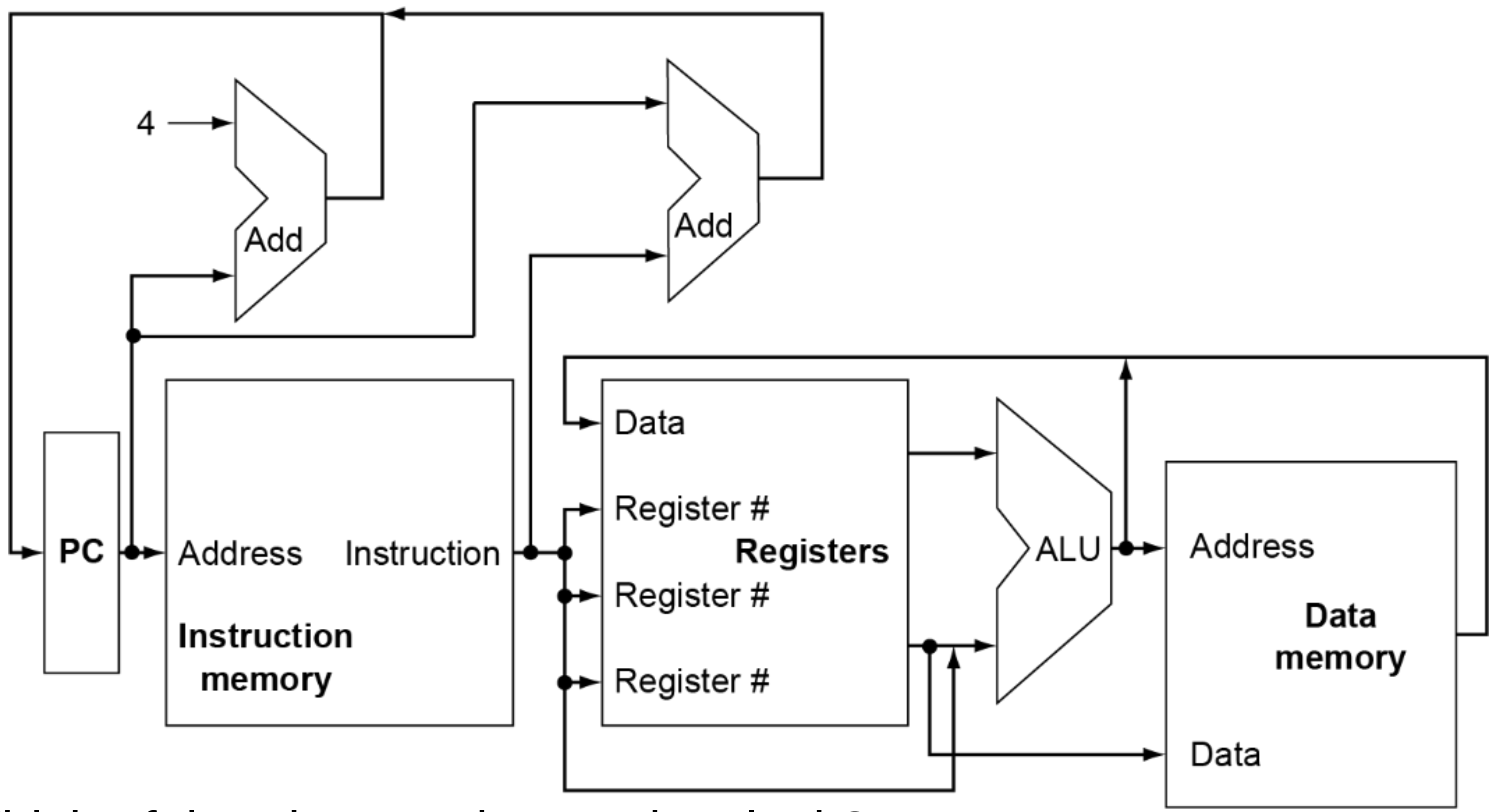
Clocking Methodology

- Defines when signals can be read and when they can be written
  - Edge-triggered clocking: all state changes occur on a clock edge.
- Clock time > the time needed for signals to propagate from SE1through combinatorial element to SE2
- A state element can be read and written in the same clock cycle without creating a race, but the clock cycle should be long enough





Clocking Methodology



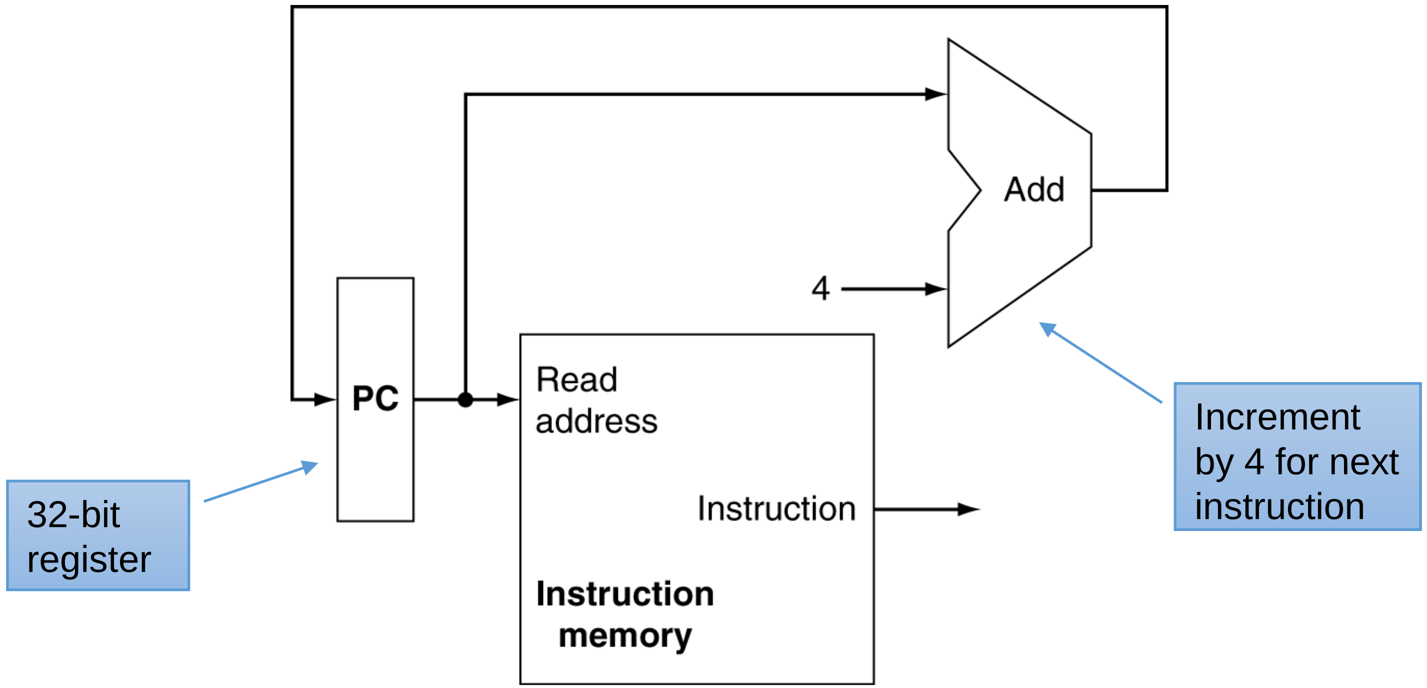
- Which of the above units need a clock?
- What is being saved (latched) on the rising edge of the clock?
- Keep in mind that the latched value remains there for an entire cycle

How to Design a Processor?

1. Analyze instruction set
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
    - See how register transfer and operation performed
4. Assemble the control logic

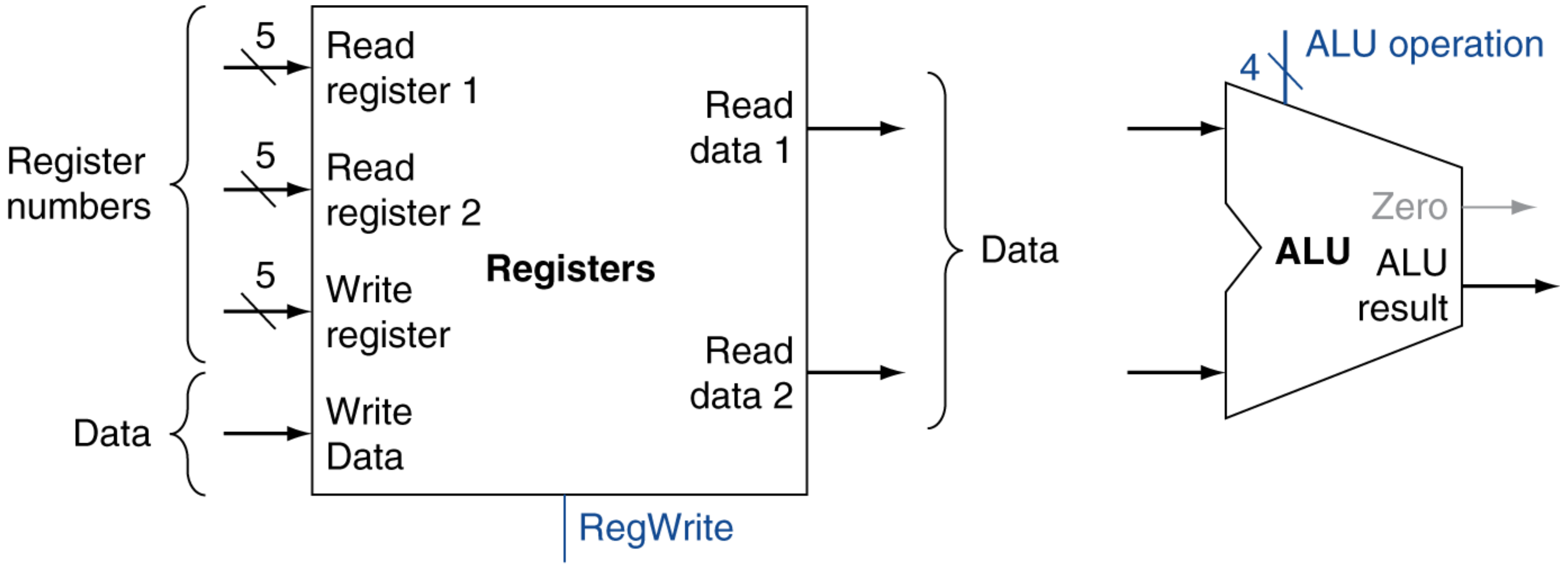
Fetch Instructions

- The instruction is fetched from I-mem and the PC is added by 4
- Components: program counter, instruction memory, adder
  - The PC is a 32-bit register, written at every positive edge of the clock, thus, it does not need write control signal.



Implementing R-Format Instructions

- Instructions of the form add t1, t2, t3
- Explain the role of each signal

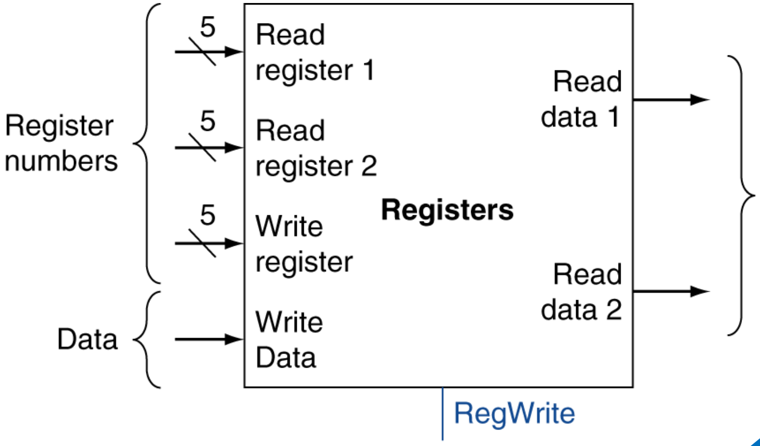


a. Registers

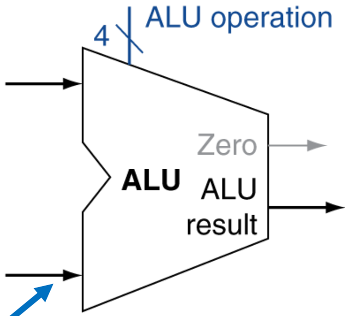
b. ALU

Implementing Load/Store Instructions

- Instructions of the form lw t1, 8(t2) and sw t1, 8(t2)

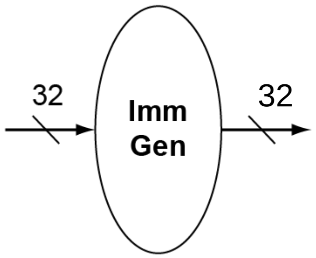


a. Registers

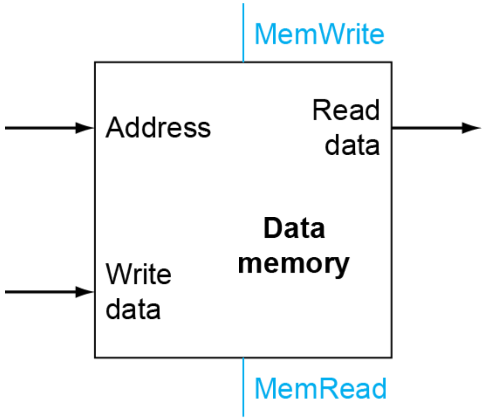


b. ALU

Where does this input come from?



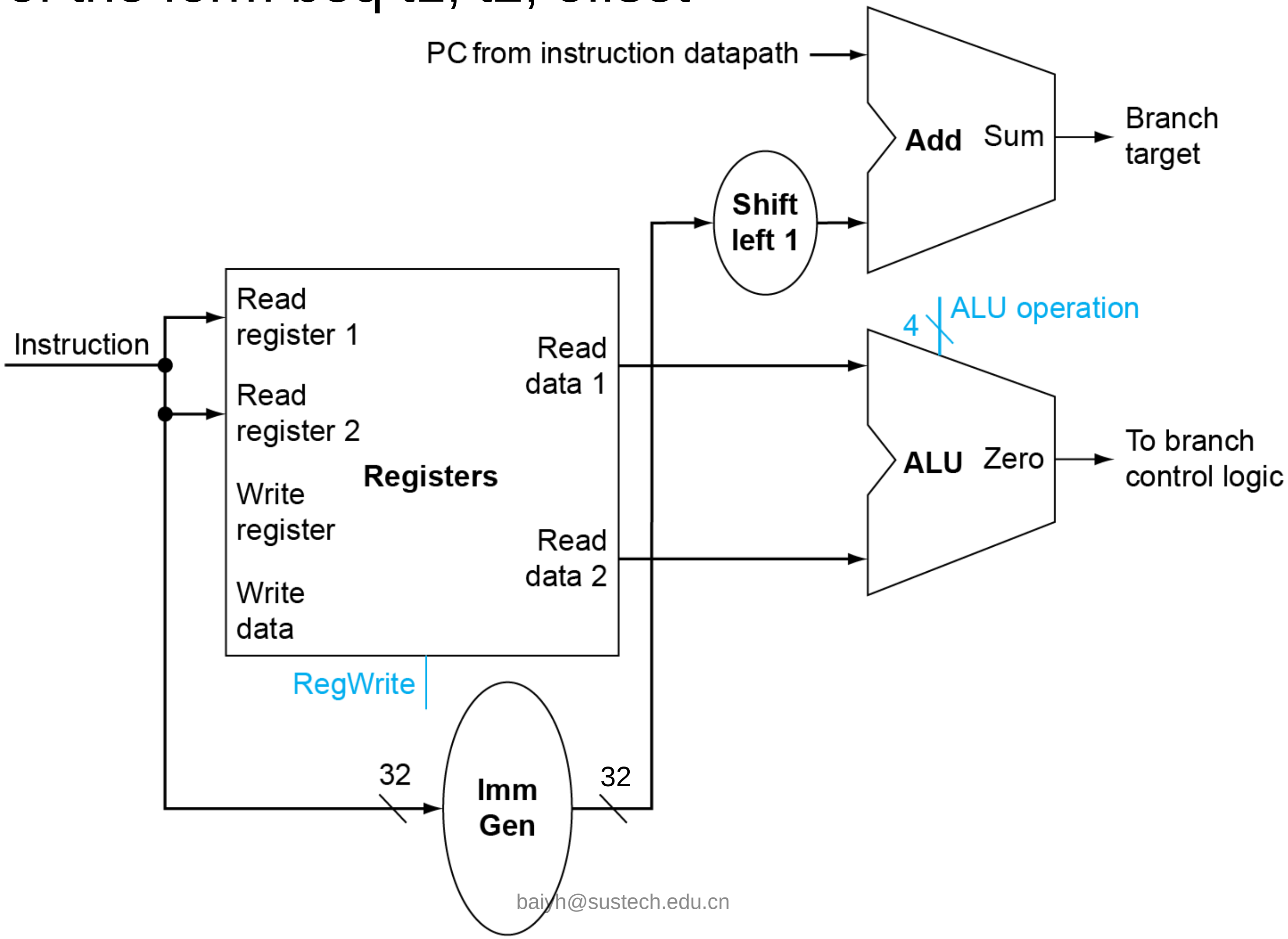
b. Immediate generation unit



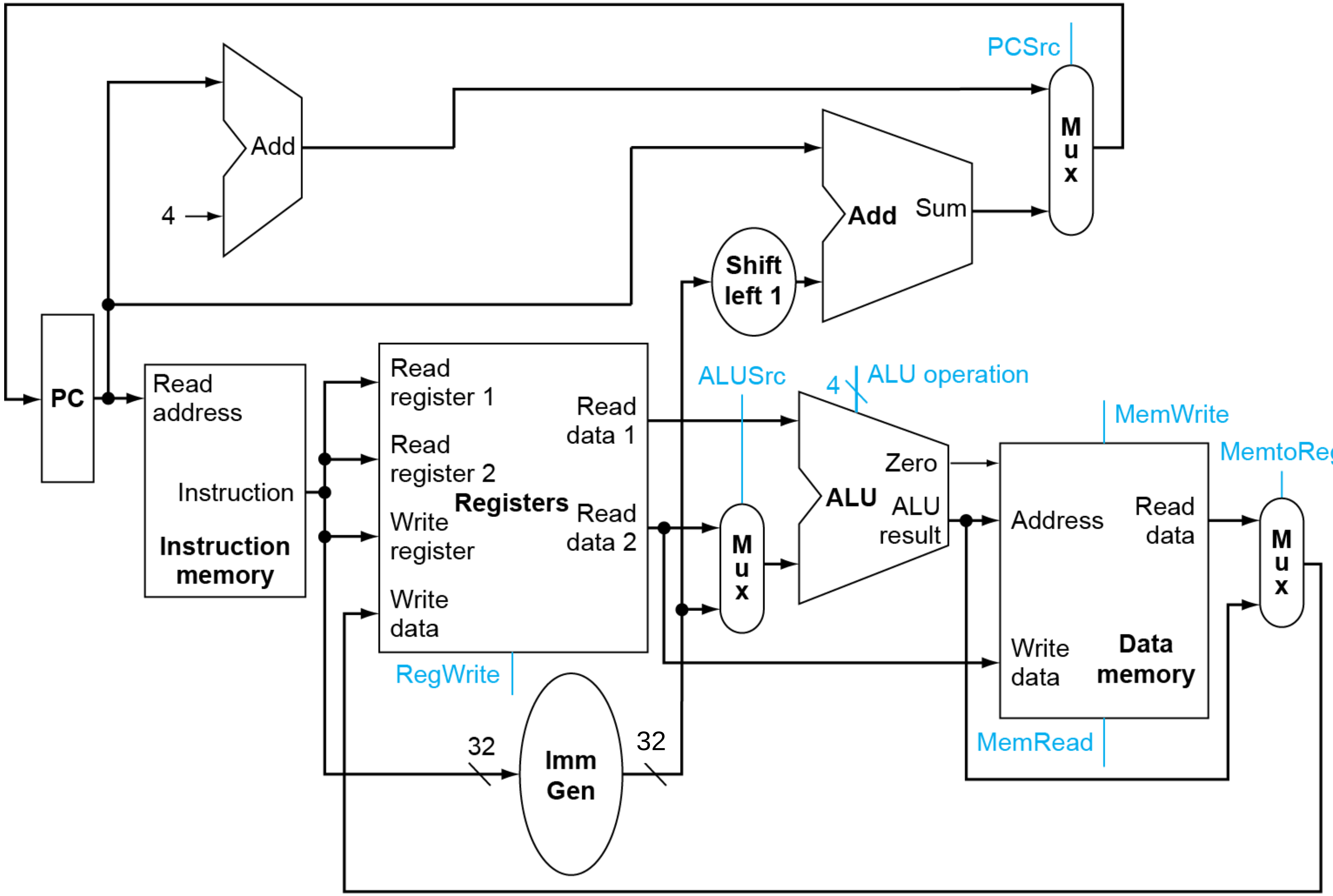
a. Data memory unit

Implementing Branch Instructions

- Instructions of the form beq t1, t2, offset



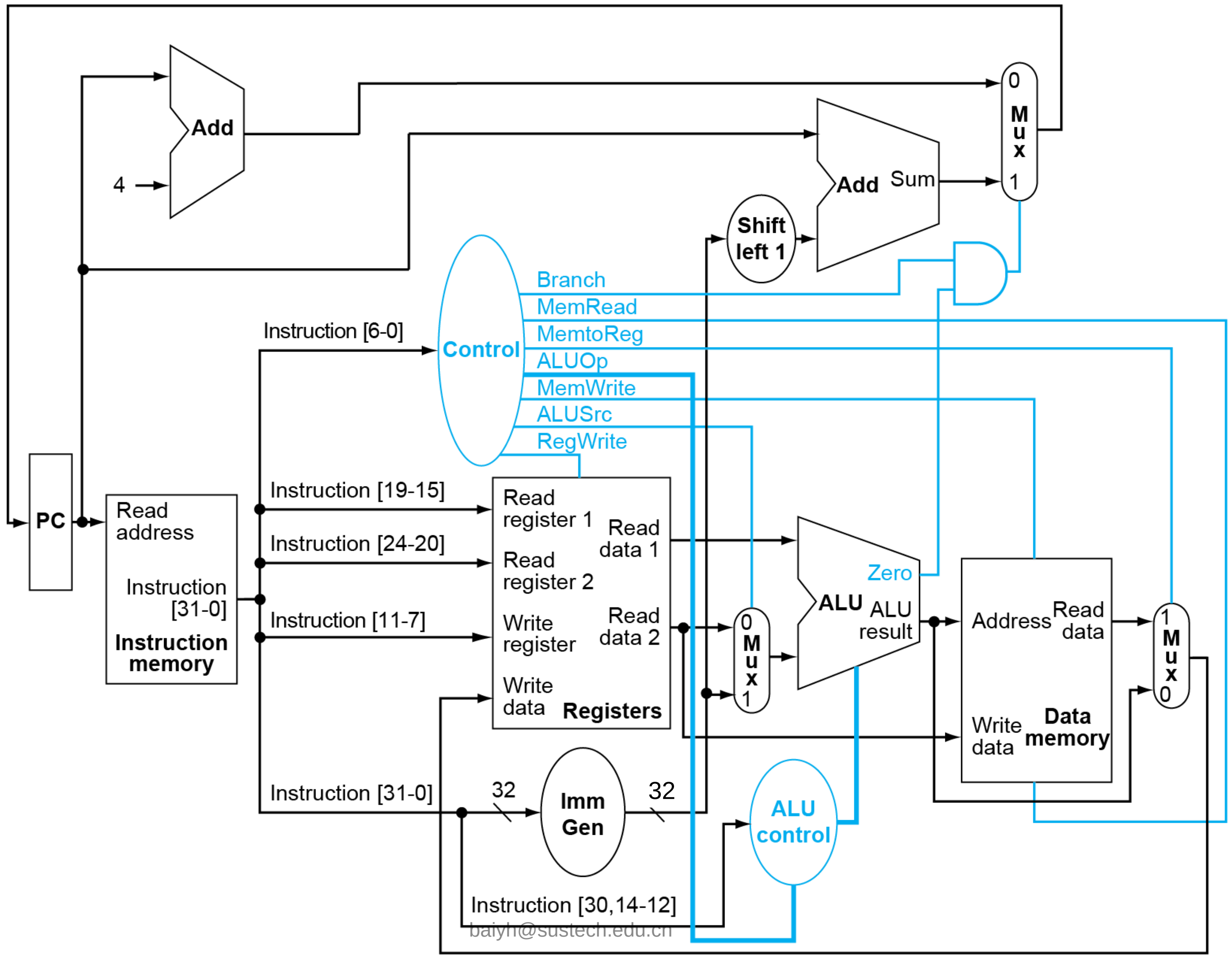
Combined Datapath



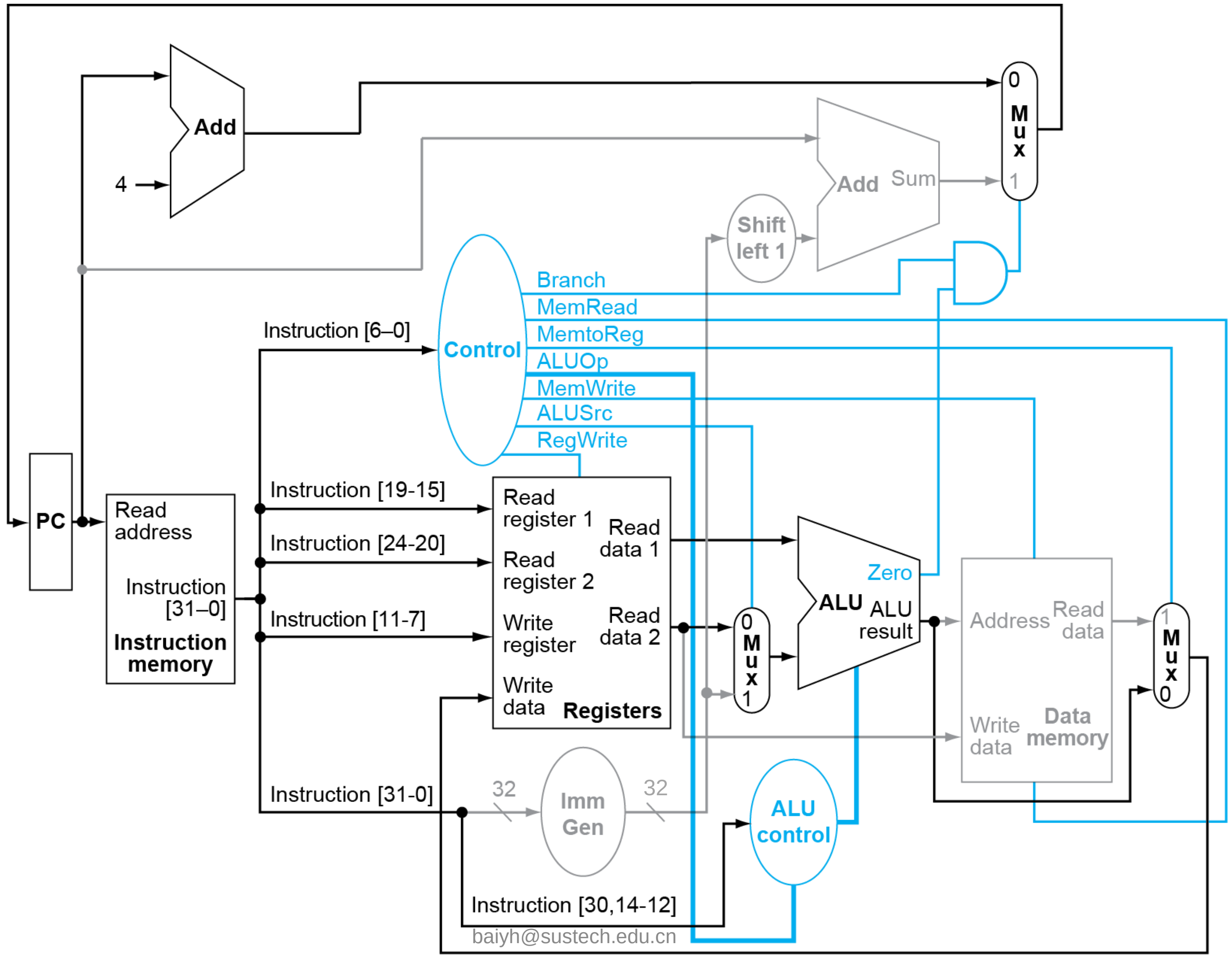
How to Design a Processor?

1. Analyze instruction set
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Assemble the control logic
   - by analyzing implementation of each instruction to determine setting of control points effecting register transfer

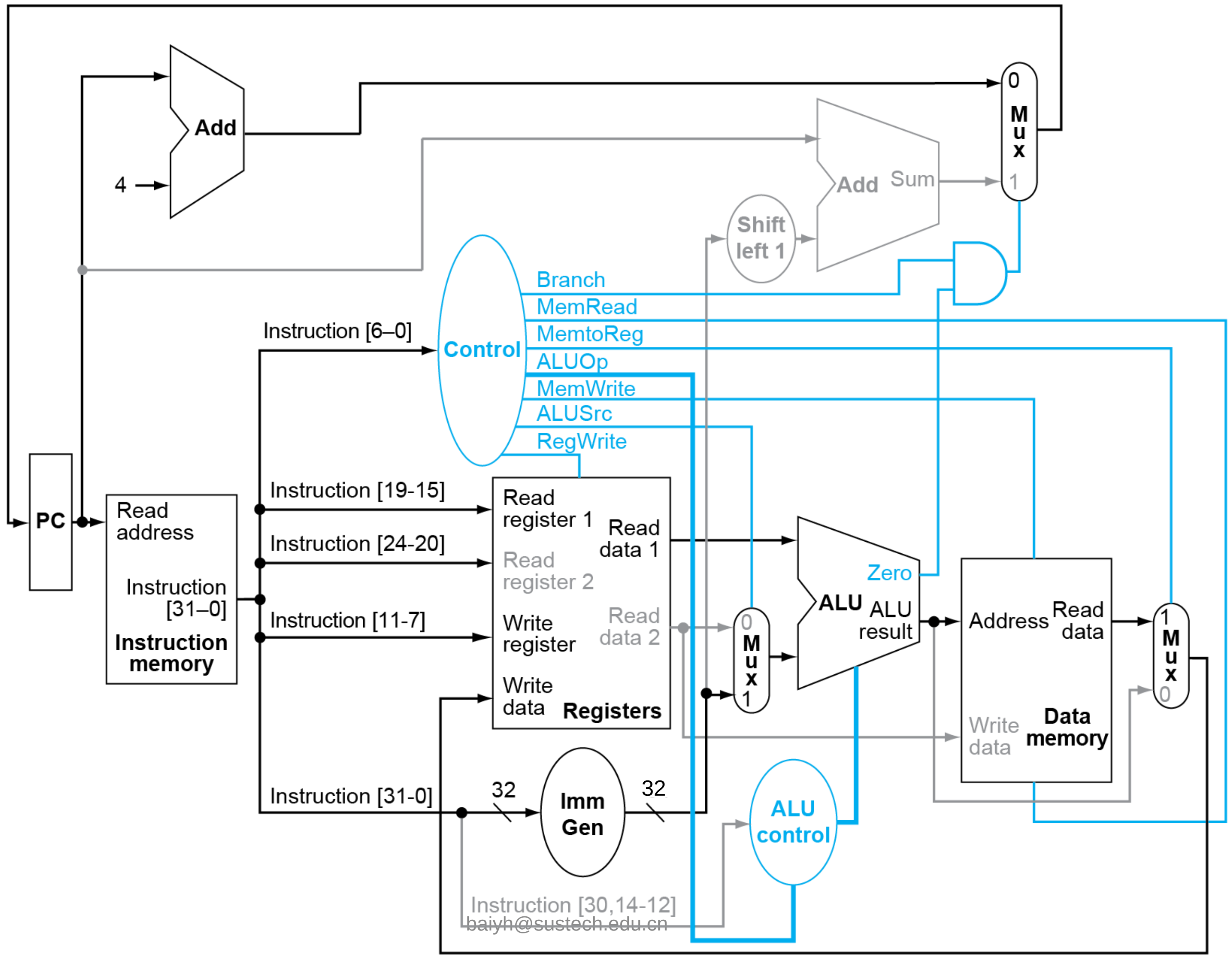
Full Datapath With Control



R-Type Instruction

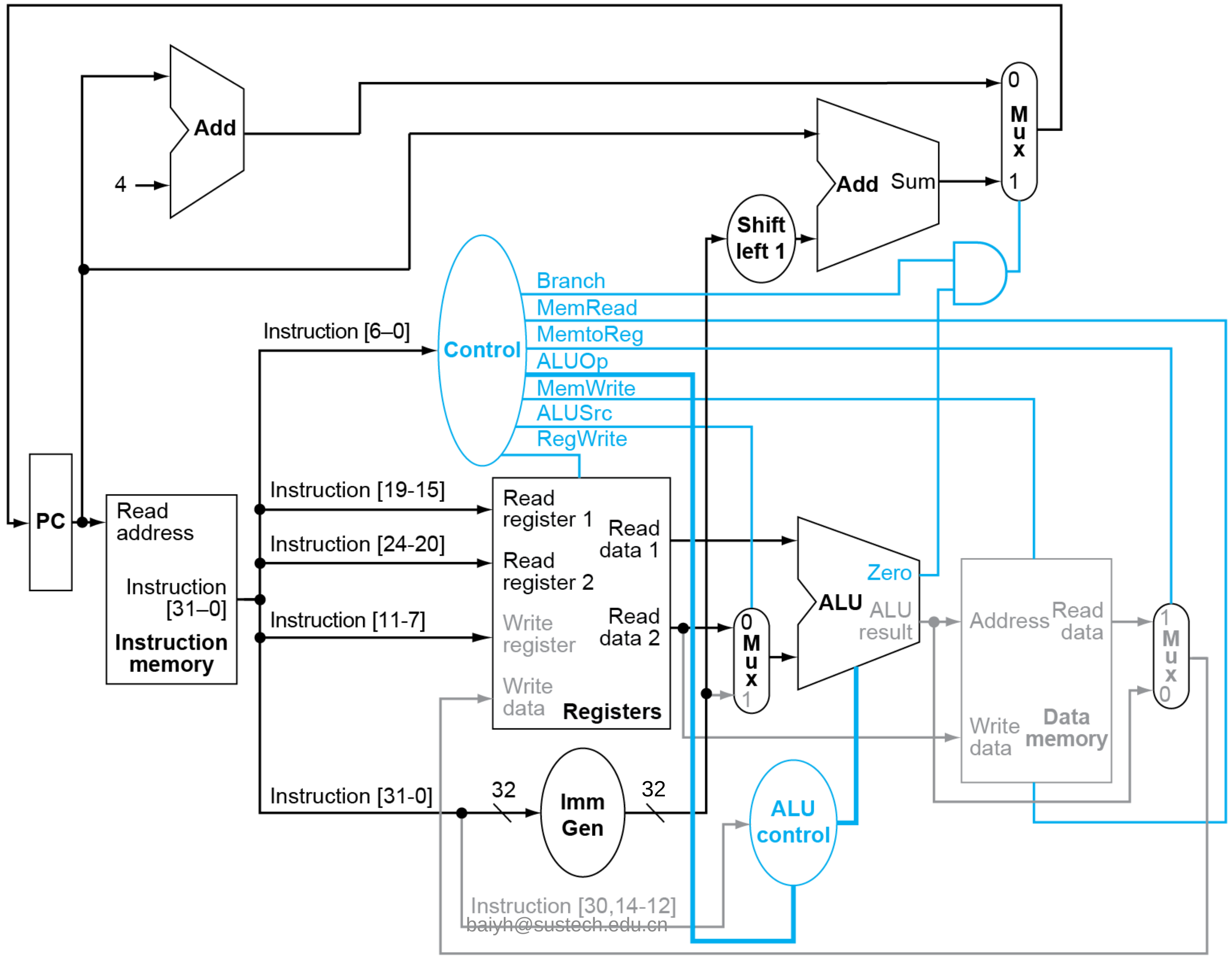


Load Instruction



baiyh@sustech.edu.cn

Beq Instruction



baiyh@sustech.edu.cn

Truth Table for Main Control Unit

Input or output	Signal name	R-format	lw	sw	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	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

ALU Control

- ALU used for
  - Load/Store: add (AluOp = 00)
  - Branch: subtract (AluOp = 01)
  - R-type: depends on opcode (AluOp = 10)
- ALU control
  - ALU control (input of ALU) is derived from ALUOp and Funct

Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

inst[30] inst[14-12]

Control Points and Signals