Name: 洪 理 川
ID : B113040056

CH04 : The Processor

4.1 Introduction

A. Basic RISC-V Implementation

- The memory-reference instructions load word (lw) and store word (sw)
- The arithmetic - logical instructions add, sub, and or
- The conditional branch instructions branch if equal (beq)

B. An Overview of the Implementation

- Send the program counter (PC) to the memory that contains the code and fetch the instruction from that Memory
- Read one / two registers, using fields of the instruction to select the registers to read. For the lw instruction, we need to read one register, but most other instructions require reading two registers.

4.2 Logic Design Conventions

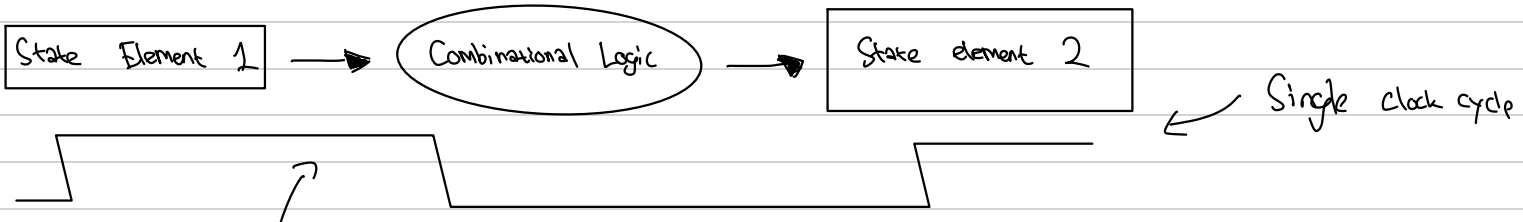
There are 2 different types of logic elements
- The elements that operate on data values are all **combinational** → the output depends only on the current inputs
- The elements that contain state → An element contains state if it has some internal storage

☆ Combinational element → An operational element, such as an AND gate or an ALU
☆ State element → A memory element, such as a register or a memory

Clocking Methology

tell when signal can be read and when signal can be written.

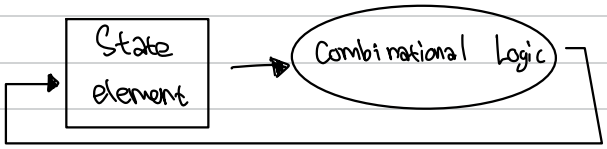
Important → old value, mix value → Computer can't handle unpredictability



Clock Cycles

- asserted → Signal is logically high/true
- deasserted → Signal is logically low/false

Edge - Triggered Clocking → Signals are updated only at clock edges, ensuring stability and predictability.



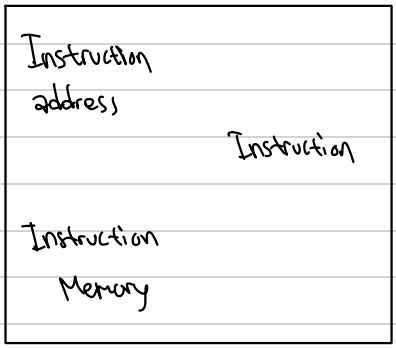
Read → contents of register
Send → through combinational logic
Write → to register

An Edge - triggered methology allows a state element to be read and written in the same clock without creating a race → indeterminate data values

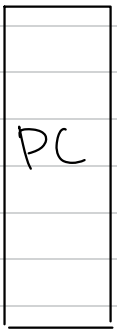
A.3 Building A Datapath

start by determine the major components → 3

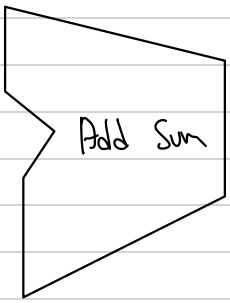
Instruction Memory

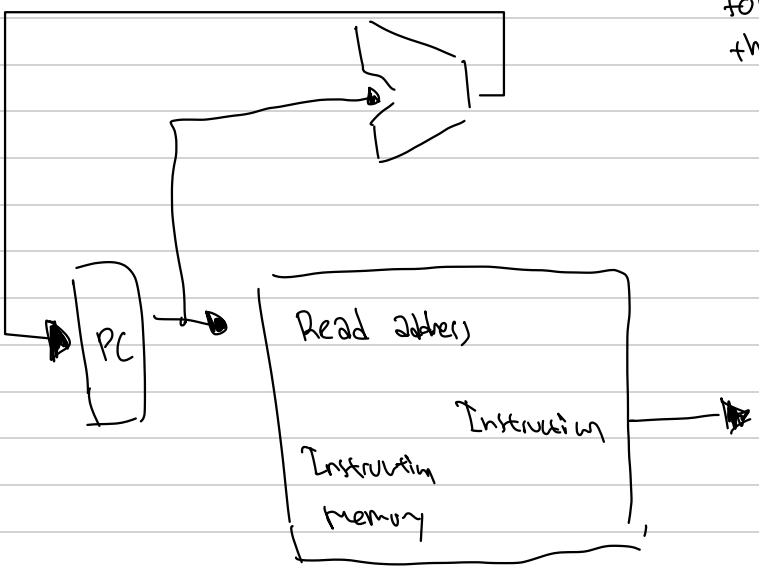


Program Counter

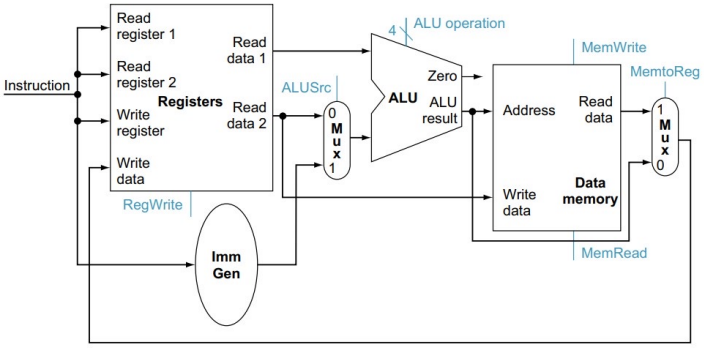


Adder





Built A portion of the datapath used for fetching instructions and incrementing the program counter.



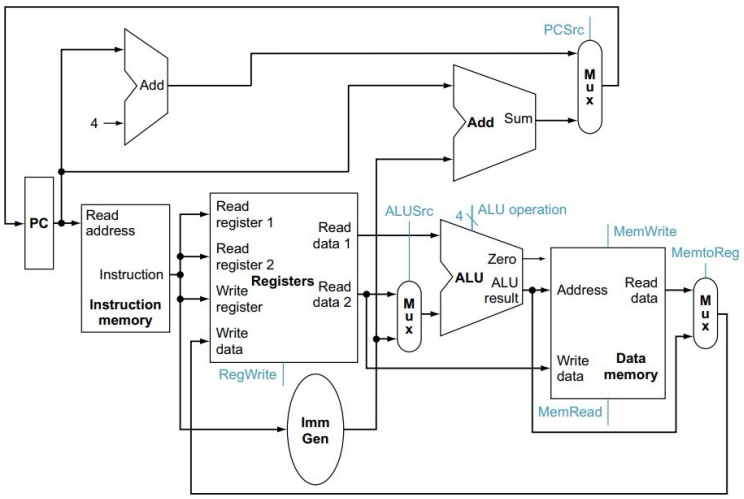
The datapath for the memory instructions and the R-type instructions.

R-format instructions have 3 register operands, so we will need to read 2 data words from the register file and write 1 data word into the register file for each instruction. For each data word to be read from the registers, we need an input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers.

The processor's 32 general-purpose registers are stored in a structure

To write a data word, we will need 2 inputs:
- To specify the register number to be written and one
- To supply the data to be written into the register

The register file always outputs the contents of whatever register numbers are on the Read register inputs. Writes, however, are controlled by the write control signal, which must be asserted for a write to occur at the clock edge.



The simple datapath for the core RISC-V architecture combines the elements required by different instruction classes

4.4 A Simple Implementation Scheme

Simple Implementation = Datapath + A simple control function

A. The ALU Control

ALU Control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

truth table → representation of logical operation by listing all the values of the inputs and then in each case showing what the resulting outputs should be.

In some condition, we don't care about the values of some of the inputs

B. Designing the Main Control Unit

Before designing the Main Control Unit, we need to review the formats of 4 instruction classes

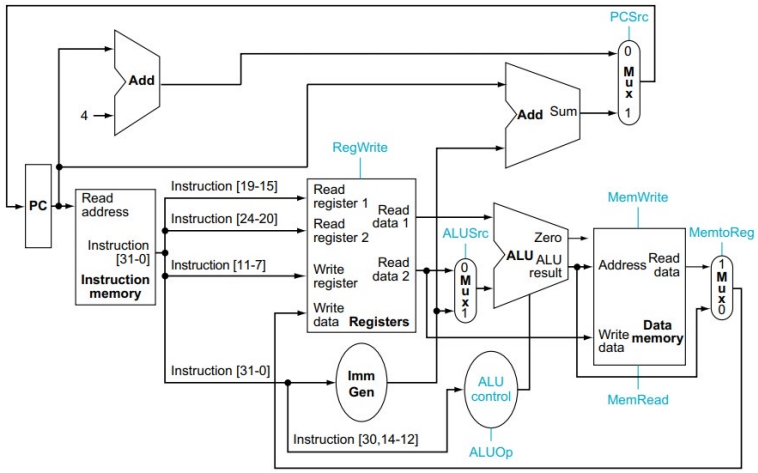
Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

Opcode → extended opcode field

RS1 → base register for load and store instructions

RS2 → register operand that gets copied to memory for store instructions

rd → R-type instructions and load instructions



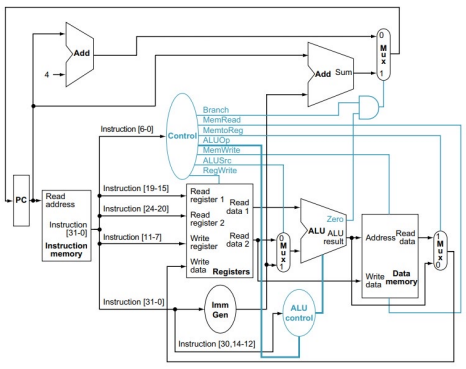
↙ Datapath + Signal Control

	Effects when deasserted	Effects when asserted
RegWrite	Nothing	Register is written with write data input
ALUSrc	second ALU comes from "Read data 2"	second ALU operand is the sign-extended
PCSrc	replaced with output PC + 4	replaced by the adder output
MemRead	Nothing	Data memory contents put on Read data output
MemWrite	Nothing	Data memory contents put on write data input
MemtoReg	fed by ALU Data	fed by data memory

C. Operation of the Datapath

There are 3 flow of different instruction classes.

1.

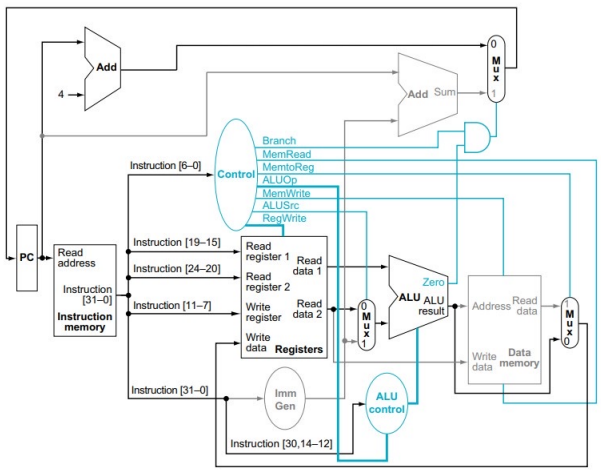


Simple datapath with the control unit

Steps: for add x_1, x_2, x_3

1. Instructions → fetched, PC is incremented
2. x_2, x_3 registers read from the register file; main control unit computes the setting of the control lines
3. ALU use the opcode to generate the ALU function.
4. The result from ALU → written into destination register (x_1)

2.



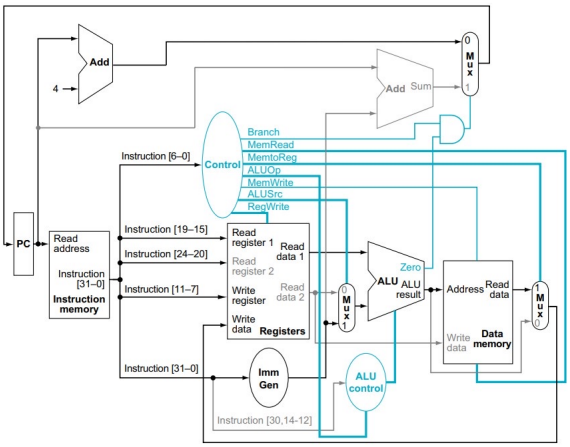
Datapath for R-type instruction

add x1, x2, x3

Steps:

1. Instructions → fetched, PC is incremented
2. x2 is read from register file
3. ALU computes the value from register file & offset
4. The sum from the ALU used for address.
5. Data is written to x1

3.



beq x1, x2 offset

1. Instructions → fetched, PC is incremented
2. x1, x2 read from register file
3. ALU subtracts one value from other value, value added to offset then shifted by one.
4. Zero status information → decide which adder result

D. Finalizing Control

↳ with control implementation

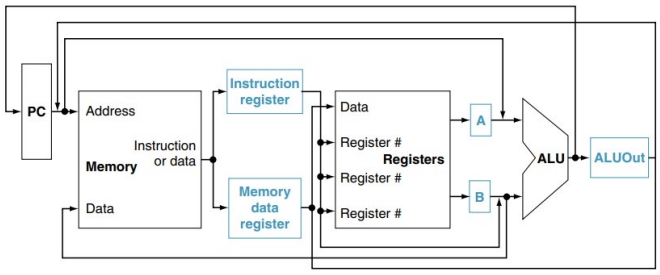
The control function is completely specified by this truth table

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

4.5 A Multicycle Implementation

Similar to single-cycle, but allow a functional unit to be used more than one, as long as the clock cycle are different. each step will take 1 clock cycle.

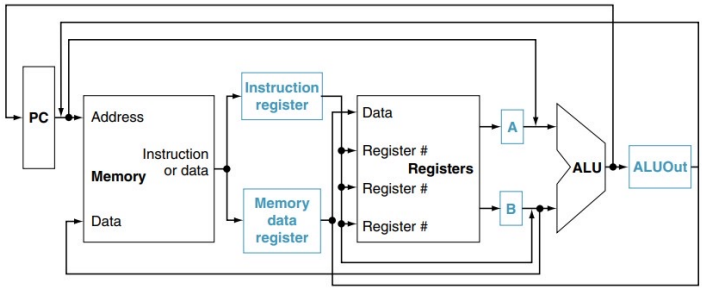
A Multicyle use a single memory unit, single ALU, one or more registers are added after every major functional unit,

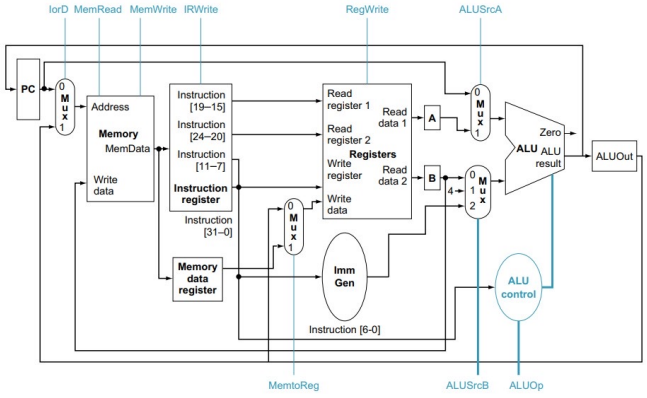


The high-level view of the multicycle datapath

Can handles the basic instructions after adding some changes →

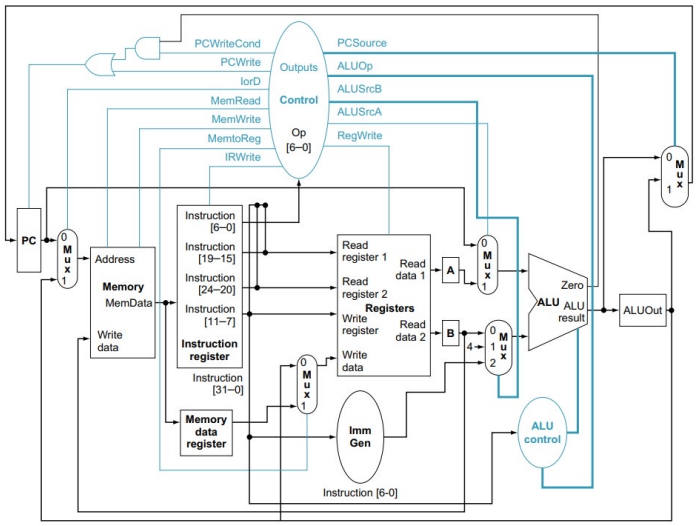
- Instruction register, Memory data register
- Additional multiplexer





With control lines

Complete datapath for Multicycle + Control lines



- Breaking the Instruction Execution into Clock Cyles

1. Instruction fetch step

```
IR <= Memory[PC];
PC <= PC + 4
```

Send the PC to the Memory as the address then Increment PC by 4.

2. Instruction decode and register fetch step

```
A <= Reg[IR[19:15]];
B <= Reg[IR[24:20]];
ALUOut <= PC + Immediate;
```

Access rs1 and rs2 then save into register A and B

Computes the branch target address and stores the address in ALUOut

3. Execution, memory address computation, or branch completion

-Memory reference

```
immediate
```

→ The ALU adding the operands to form the memory adress

- Arithmetic-Logical instruction (R-type)

```
ALUOut <= A op B;
```

→ perform operation specified by the opcode

- Branch

```
if (A == B) PC <= ALUOut
```

→ equal comparison then ALU is used to determine whether or not to branch

4. Memory Address or R-type instruction complete step

- Memory reference

```
MDR <= Memory[ALUOut];
```

→ load operation
data retrieved from Memory → written to MDR

or

```
Memory[ALUOut] <= B;
```

→ Store operation → written to Memory

- Arithmetic-logical instruction (R-type)

```
Reg[IR[11:7]] <= ALUOut;
```

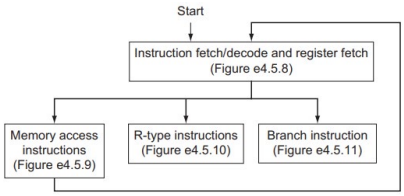
→ Move the data in ALUOut to Result Register

5. Memory read completion step

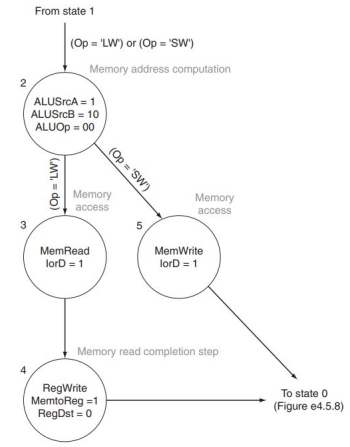
Load:

`Reg[IR[11:7]] <= MDR;` → get the MDR data → write into register file

Defining the control

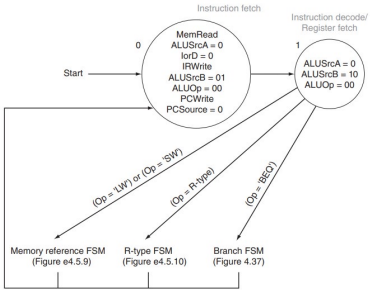


High level view of finite-state machine



Controlling memory reference instructions

fetch and decode portion of every instruction is identical

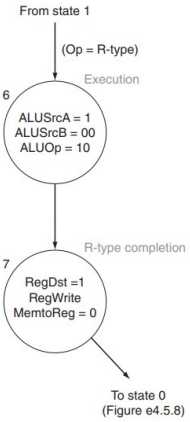


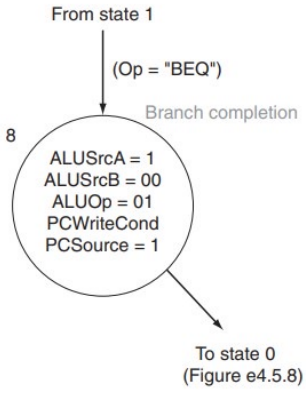
+

+

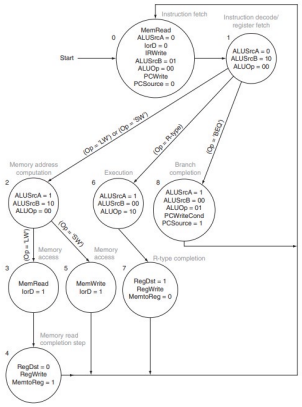
+

R-type instruction can be implemented with a simple 2-state finite state machine

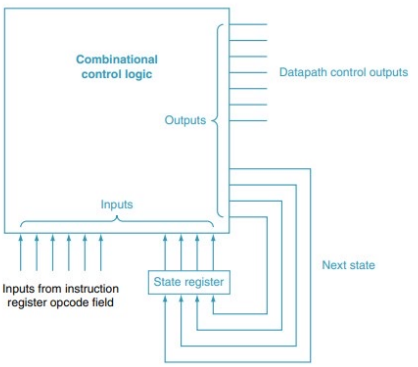




branch instruction requires a single state



=



Complete finite-state machine for datapath

Usually implemented using a block of combinational logic and a register to hold the current state