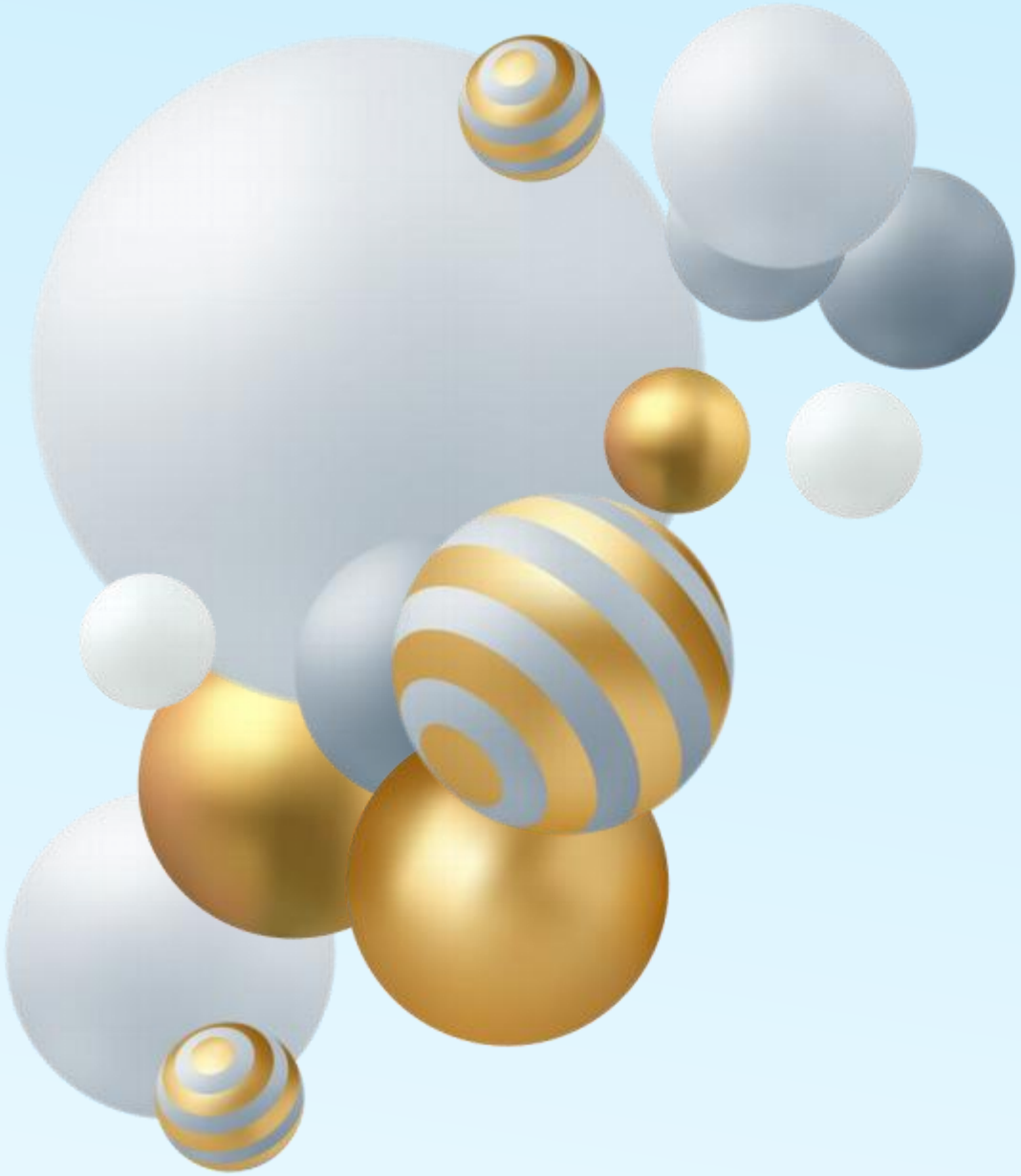
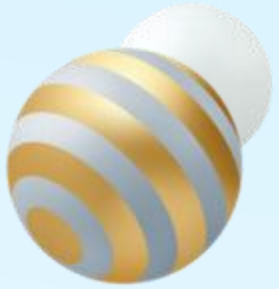


Computer Organization

Lab9 CPU Design(1)

ISA, Control + Data
IFetch,Decoder



Topic

- CPU Design(1)
  - ISA
  - CPU = Control Path + Data Path

- Instruction Fetch
  - update PC register
  - fetch the instruction according to PC register
- Instruction analysis
  - Decoder of Data Path

RISC-V32I

RISC-V-Reference-Data.pdf

I type {imm[11:0], rs1, funct3, rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
lb	**000_0011**	0		**03**/0/-
lh	**000_0011**	1		03/1/-
lw	**000_0011**	10		03/2/-
ld	**000_0011**	11		03/3/-
lbu	**000_0011**	100		03/4/-
lhu	**000_0011**	101		03/5/-
jalr	**110_0111**	0		**67/0/-**
addi	**001_0011**	0		**13/0/-**
slli	**001_0011**	1	0	13/1/0
slti	**001_0011**	10		13/2/-
sltiu	**001_0011**	11		13/3/-
xori	**001_0011**	100		13/4/-
srli	**001_0011**	101	0	13/5/0
srai	**001_0011**	101	010_0000	13/5/20
ori	**001_0011**	110		13/6/-
andi	**001_0011**	111		13/7/-

S type {imm[11:5], rs2, rs1, funct3, imm[4:0], opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
sb	**010_0011**	0		**23/**
sh	**010_0011**	1		
sw	**010_0011**	10		
sd	**010_0011**	11		

R type {func7, rs2, rs1, func3, rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
add	**011_0011**	0	0	**33/**
sub	**011_0011**	0	010_0000	
sll	**011_0011**	1	0	
slt	**011_0011**	10		
sltu	**011_0011**	11		
xor	**011_0011**	100		
srl	**011_0011**	101	0	
sra	**011_0011**	101	010_0000	
or	**011_0011**	110		
and	**011_0011**	111		

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

SB type {imm[12|10:5], rs2, rs1, funct3, imm[4:1|11], opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
beq	**110_0011**	0		**63/**
bne	**110_0011**	1		
blt	**110_0011**	100		
bge	**110_0011**	101		
bltu	**110_0011**	110		
bgeu	**110_0011**	111		

U type {imm[31:12], rd, opcode}

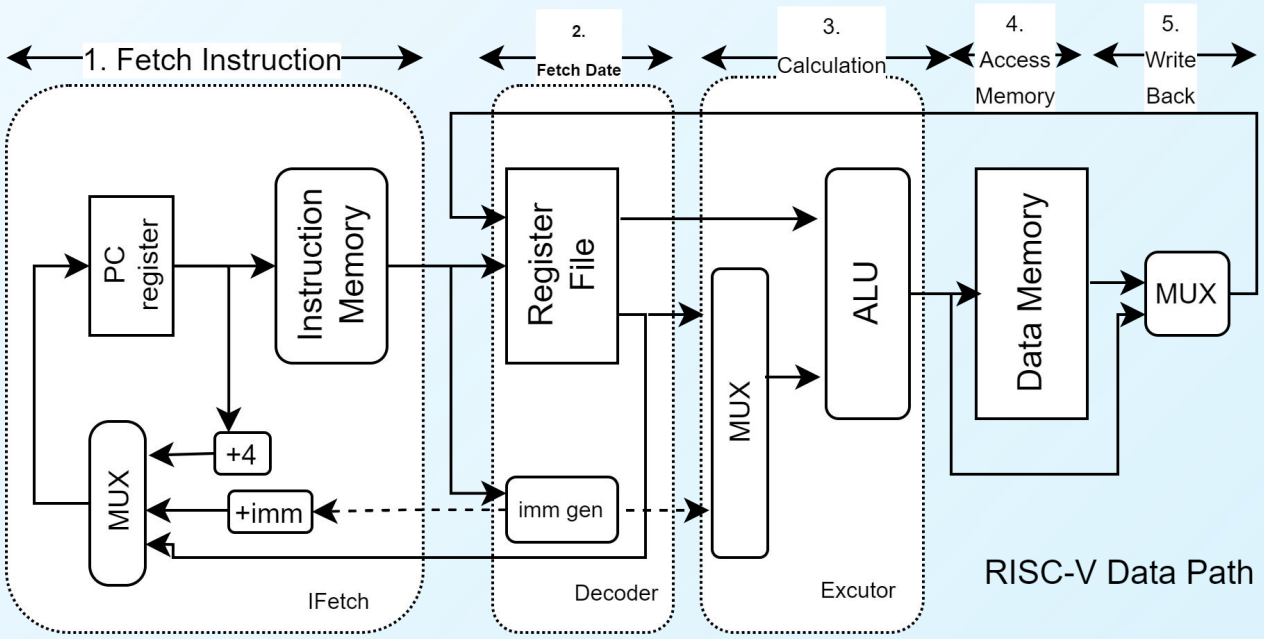
inst	OPCODE	FUNCT3	FUNC7	hex
auipc	**001_0111**			**17/**
lui	**011_0111**			**37/**

UJ type {imm[20|10:1|11|19:12], rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
jal	**110_1111**			**6f/**

Data Path(1)

Data Path:
The parts in CPU with componets which are involved to execute instructions.

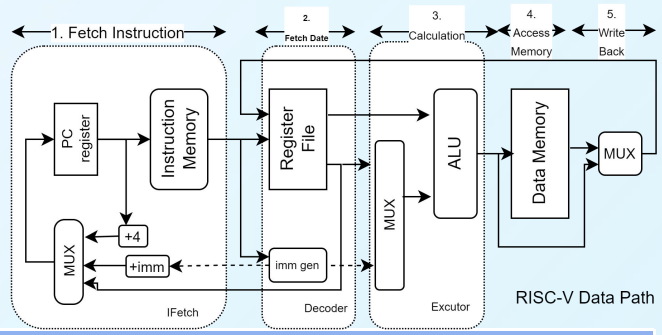


	1. Instruction fetch	2. Data Fetch	3. Instruction Execute	4. Memory Access	5. Register WriteBack
add[R]	Y	Y	Y		Y
addi[I]	Y	Y	Y		Y
sw[S]	Y	Y	Y	Y	
lw[I]	Y	Y	Y	Y	Y
branch[SB]	Y	Y	Y		
jalr[I]	Y	Y	Y		Y
jal [UJ]	Y	Y	Y		Y

Data Path(2)

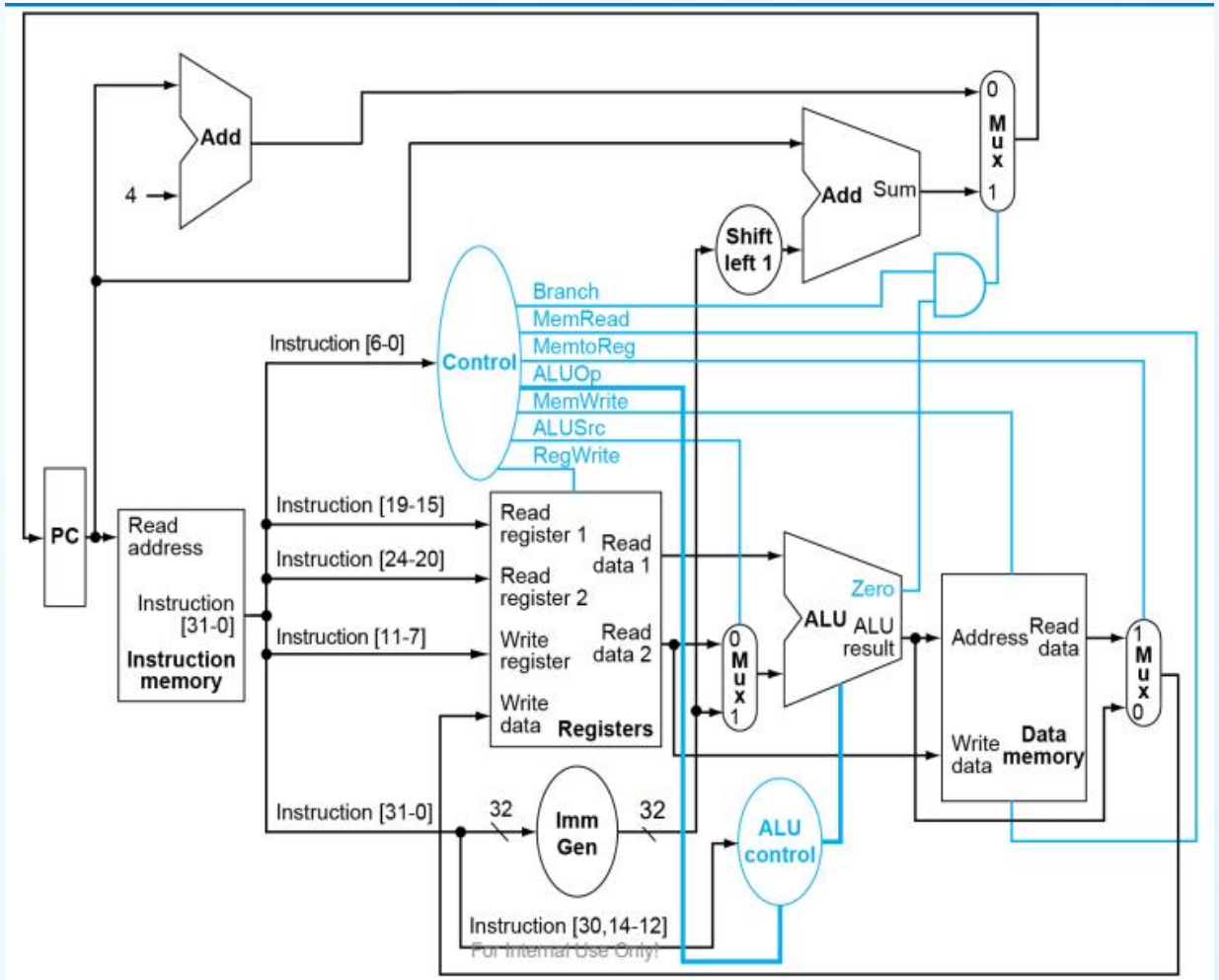
CORE INSTRUCTION FORMATS

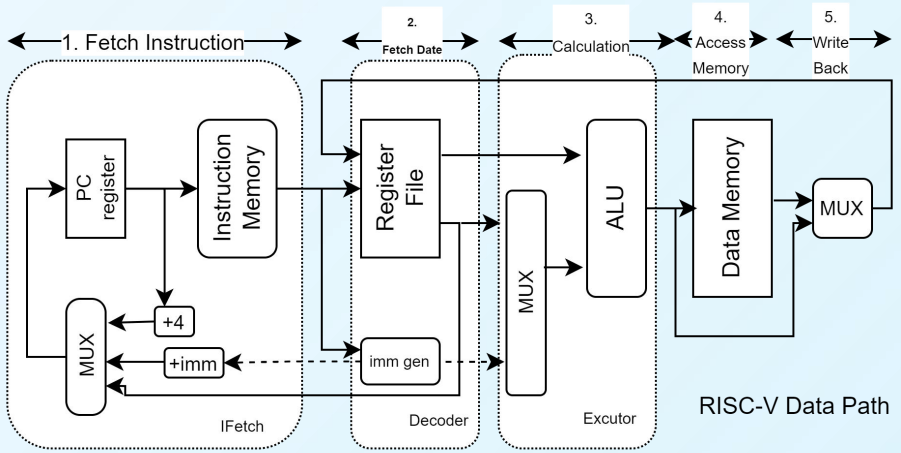
	31–27	26–25	24–20	19–15	14–12	11–7	6–0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode



Module	How To Process	Instructions and Comments
IFetch	Determine how to update the value of PC register	1. pc+immediate(sign extended) (bne [**b**], jal[J]) 2. the value of the specified register (ra by default) (jalr [I]) 3. pc+4 (other instruction except branch, jal and jal)
Decoder	Determine whether to write register or not	lw [I], R type, U type vs beq([B]) vs sw([S])
	Get the source of data to be written Get the address of register to be written	1. Data memory (lw[I]) -> rd 2. ALU ([R]) -> rd 3. address of instruction (jal[J]) -> the specified register
	Determine whether to get immediate data from the instruction and expand it to 32bit	add([R]) vs addi([I]) vs lui([U]) vs sw([S]) vs beq([B]) vs jal([J])
Memory	Determine whether to write memory or not	(sw[S]) vs lw[I]
	Get the source of data to be written	rs2 of registers (sw[S])
	Get the address of memory unit to be written	the output of ALU (sw[S])
ALU	Determine how to calculate the datas	add, sub, or, sll, sltiu, sra, slt, branch ...
	determin the source of one operand from register or immediate extended	R(register), I(sign extended immediate)

Control Path





RISC-V Data Path

Determine the control signals and the related destination
Complete the table according to your design

Name	defination	Destination
Branch		
MemRead	1’b1 means read from Memory, 1’b0 means not	Data Memory
MemtoReg		
MemWrite	1’b1 means write memory, 1’b0 means not	Data Memory
ALUSrc		
RegWrite	1’b1 means write , 1’b0 means not	Decoder / Rigster File
...	...	...

Practice 1

1. Please draw the structure diagram of your single cycle CPU and determine the relationship between each module.

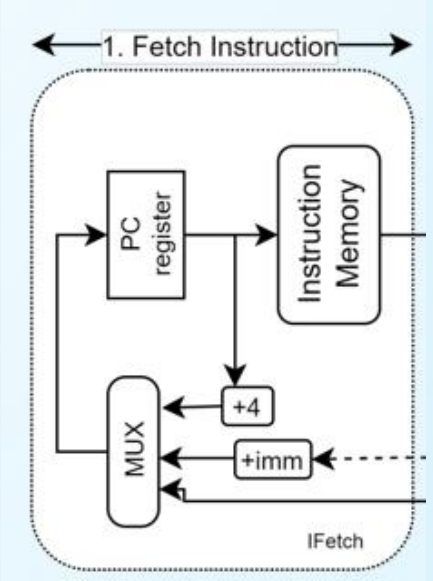
2. Please determine the port specifications for each submodule in the data path of your design, the source and target modules of the ports, and the relevant descriptions of the ports. Here is an example of Decoder.

Decoder ports	name	direction	bitwidth	from	to	comments
	rs1	input	5	ifetch	decoder	
	rs2	input	5	ifetch	decoder	
	rd	input	5	ifetch	decoder	
	clk	input	1	cpu_top		
	regWrite	input	1	controller		1'b1 means wirte the data of wdata to the register specified by rd, otherwise not
	rdata1	output	32		alu	
	rdata2	output	32		alu	
	wdata	input	32		decoder	

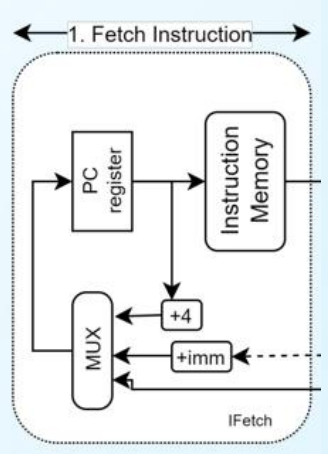
Instruction Fetch

- **Task 1:** update the data in PC register
  - WHEN: After the instruction is fetched from Instruction Memory and analyzed (at posedge/negedge of clk)
  - HOW: write the PC register with the output of the MUX

- **Task 2:**fetch the instruction from Instruction Memory according to the data in PC register
  - WHEN: After the PC is ready (at posedge/negedge of clk)
  - HOW: using the value of PC register as address to read the data stored in the Instrucion Memory

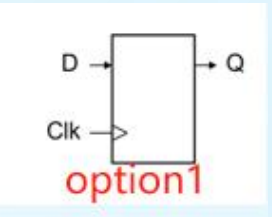


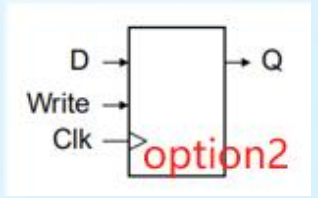
practice2



- Implement the sub-module of CPU: IFetch, build a testbench to verify its function
  - input ports1: clk, reset
  - input ports2: control signals as the select signals of MUX
  - input ports3: data from Decoder (imm ...)
  - output ports: instruction
  - Tips: The submodule 'Instruction Memory' has already been done in lab8. It is suggested to instance 'Instruction Memory' in the IFetch module to speed up the coding.

- **Answer the following question:**
  - There are 2 types of registers(option1 and option2), only one type is suitable for PC register, which one?
  - How to determine the initial value of PC register?
  - What's the relationship between the value of PC register and the address used to fetch the instruction?





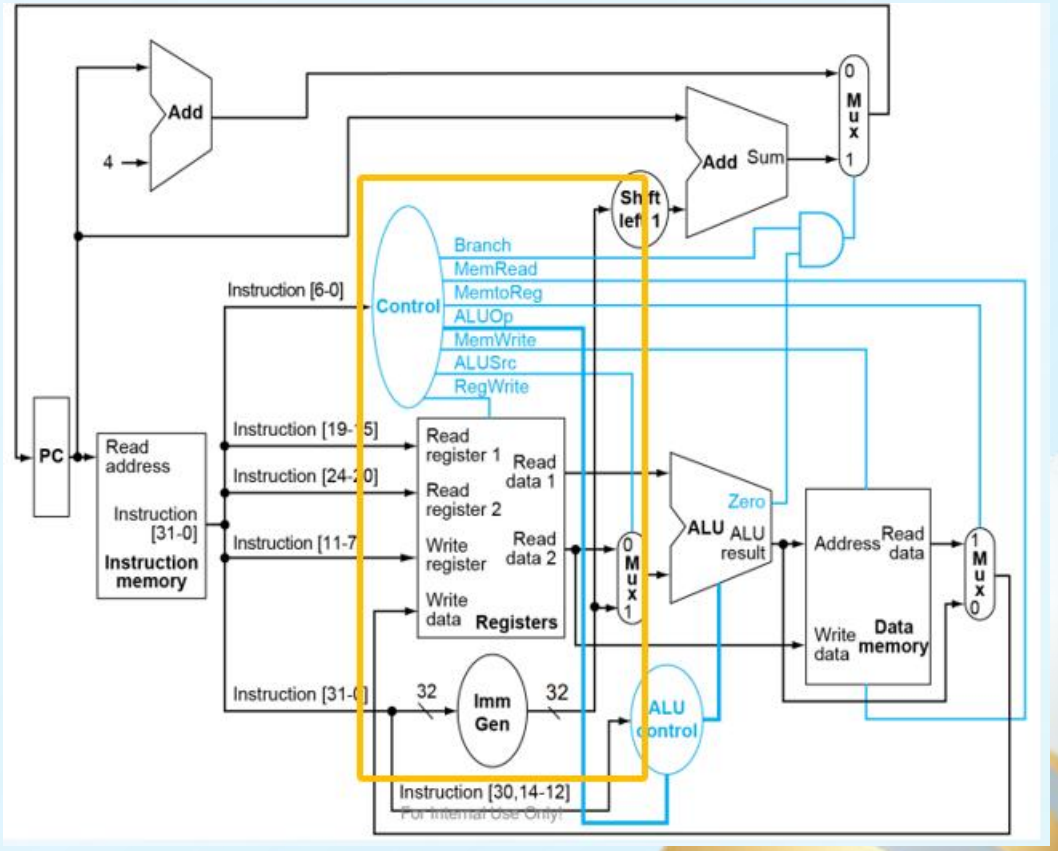
Instruction Analysis

- **Task 1: get information about the data from the instruction (Decoder in Data Path)**
  - address of **registers**: **rs2**(Instruction[24:20]), **rs1**(Instruction[19:15]) and **rd**(Instruction[11:7])
  - **shift mount**(instruction[24:20]) (R type and I type)
  - **immediate(12bits for I/S/SB, 20bits for U/UJ)**

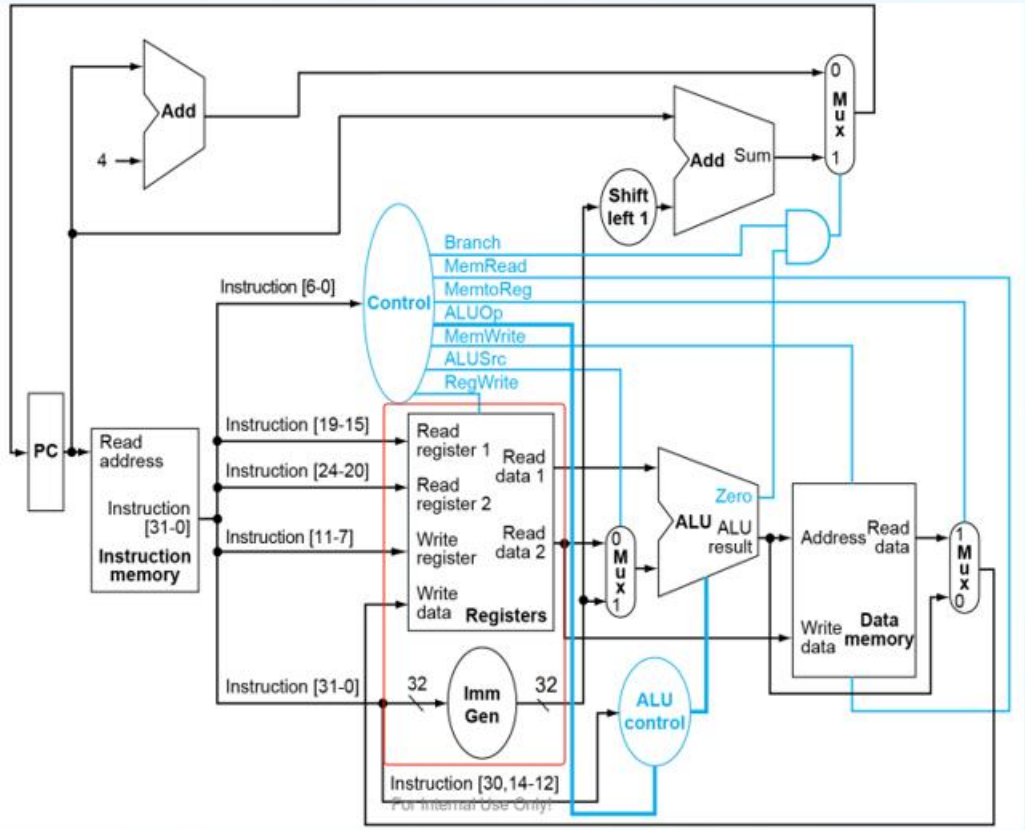
- **Task 2: generated control signals according to the instruction (Controller in Control Path)**
  - **get Operation code**(inst[**6:0**]) **and function code**(funct7,funct3) **in the instruction**
  - **generate control signals** to submodules of CPU

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode



Data Path(1) Decoder



- **Get data** from the instruction directly or indirectly
  - opcode, function code : how to get data, where to get data
  - **immediate data** needs to be **signextended to 32bits for calculation.**
  - **data in the register**, the address of the register is coded in the instruction. e.g. rs2 = Instruction[24:20];
  - **data in the memory**, the **address** of the memory unit need to be calculated by ALU with **base address** stored in the **register** and **offset** as **immediate data in the instruction**

- **Read/Write data from/to Register File**

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Tips: The submodule ‘Imm Gen’ has already been done in lab8. It is suggested to instance ‘Imm Gen’ in the Decoder to speed up the coding.

Decoder continued

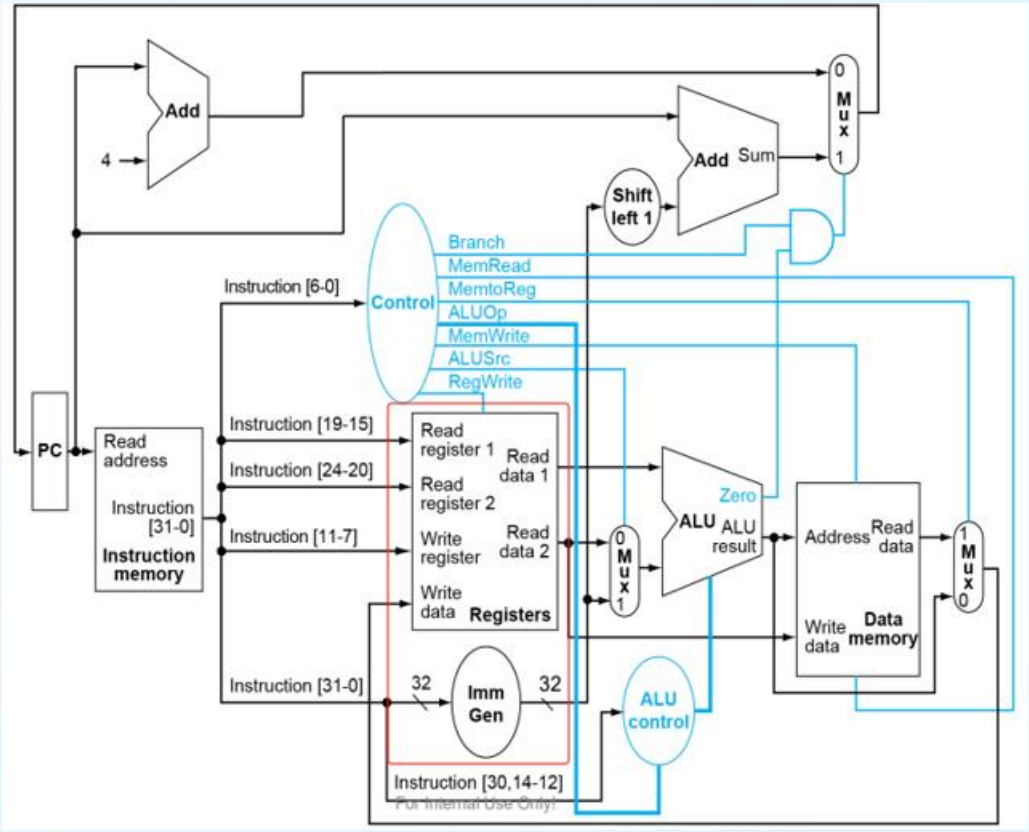
CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

- **Registers(Register File)**

  -Inputs

  - **read address**
    - [R] add: rs1,rs2
    - [B] beq : rs1, rs2
    - [I_calculate] addi: rs1
    - ...
  - **write address**
    - [R] add: rd
    - [J] jal : rd/[31]
    - [I_calculate] addi: rd
    - ...
  - **write data**
    - [R]/[I_calculate] add: data from alu_result
    - [I_load] lw: data from memory
    - ...
  - **control signal**
    - [R]/[I]/[J] RegWrite
    - [J] jal
    - ...



- **Registers(Register File)**

  -Outputs

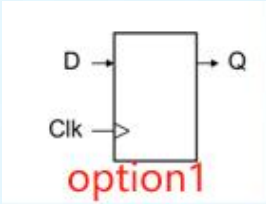
  - read data1
  - read data2
  - extended Immi

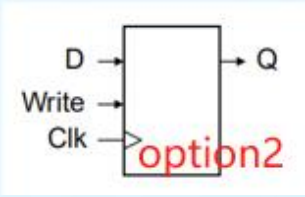
Decoder continued

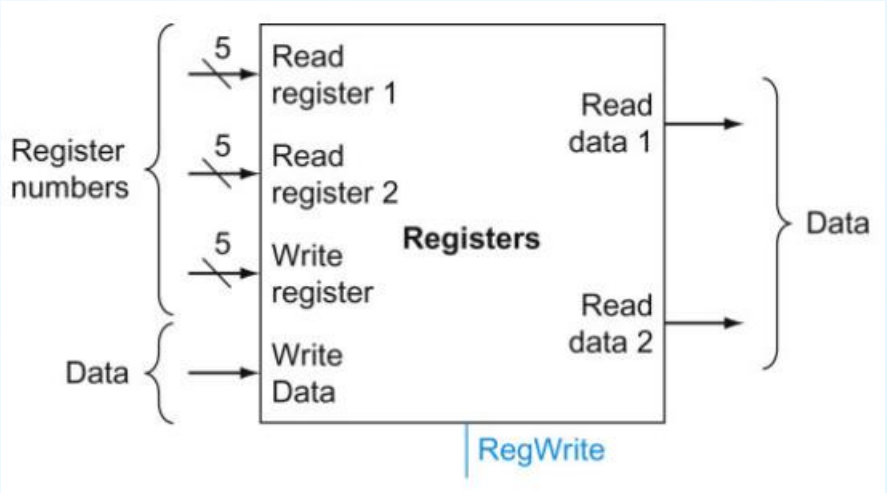
CORE INSTRUCTION FORMATS

	31–27	26–25	24–20	19–15	14–12	11–7	6–0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Register File(Registers):
Almost all the instructions need to read or write register file in CPU;
32 common registers with same bitwidth: 32








```
//verilog tips:
reg[31:0] register[0:31];
assign Rdata1 = register[Read register 1];
register[Write register] <= WriteData;
```

Q1:
How to avoid the confliction between register read and register write?

Q2：Which kind of circuit is this register file, combinatorial circuit or sequential circuit?

Q3: How to determine the size of address bus on register file?

Practice3

CORE INSTRUCTION FORMATS

	31–27	26–25	24–20	19–15	14–12	11–7	6–0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

- 1. Implement the sub-module of CPU: Decoder
  - There are **32** registers(each register is **32bits),** All the registers are **readable** and **writeable** except **x0**, **x0 is readonly**.
  - The **reading** should be done at any time while **writing** only happens on the **posedge** of the clock.
  - The register file should support **R/I/S/SB/U/UJ** type instructions(extend the immediate to be 32bits if needed).
    - such as: **add; addi; jalr; lw; sw; beq;jal;**
- 2. Verify its function by simulation. NOTES: The verification should be done on the full set of testcase.
- 3. List the signals which are used by the Decoder (NOTE: Signals' name are determined by designer)

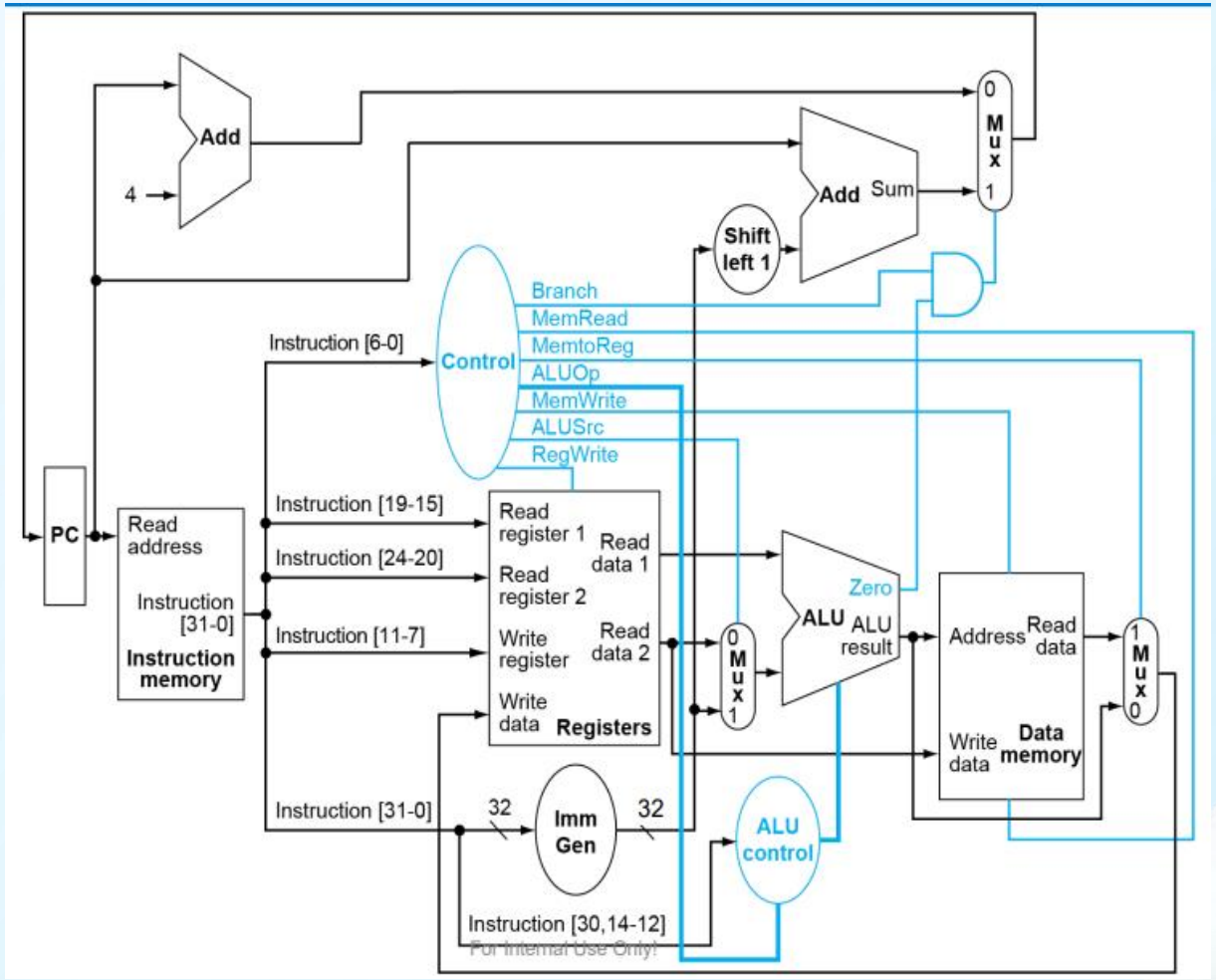
  tips: following table could be used as a reference

name	from	to	bits	function
regWrite	Controller	Decoder	1	1 means write the register identified by writeAdress
imm32	Decoder	IFetch	32	the signextended immediate
instruction	IFetch	Decoder	32	the instruction
rdata1	Decoder	ALU	32	the data read from the register which is specified by rs1
…				

Tips: Control Path & Data Path of CPU

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode



Control Path: Interprete instructions and generate signals to control the data path to execute instructions

Data Path: The parts in CPU with componets which are involved to execute instructions