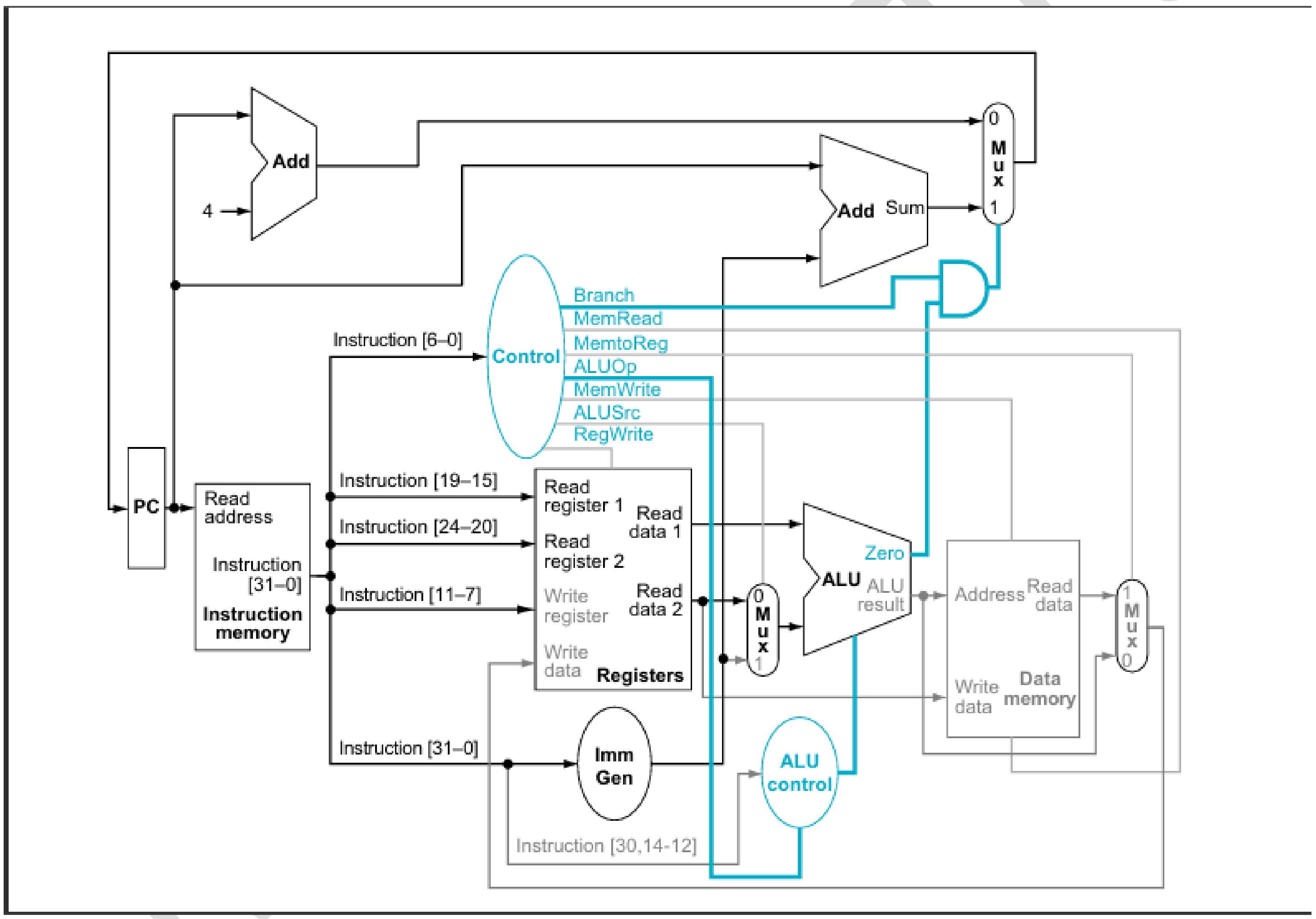
The Datapath of Single-Cycle Implementation RISC-V Processor



Program Counter (PC) – Single Cycle RISC-V Processor

The Program Counter (PC) is a 32-bit register that holds the address of the current instruction being executed. In each clock cycle, the PC provides this address to the Instruction Memory, which fetches the corresponding instruction. After execution, the PC is updated to point to the next instruction address.

Functions

1. Fetch instruction: Sends current address to instruction memory.
2. Update address: Calculates the next address as PC + 4 (since each instruction is 4 bytes).
3. Branch/Jump control: If a branch or jump instruction occurs, the PC updates with the branch target address instead of PC + 4.

Connections

- Output of PC → Instruction Memory (for instruction fetch).
- PC + 4 Adder → Computes next sequential address.
- MUX → Selects between PC + 4 and branch target address.
- Selected value → Fed back to PC at next clock.

Example

If initial PC = 0x00000000

- After normal instruction → PC = 0x00000004
- After branch taken → PC = target address

Summary:

The Program Counter controls the flow of instruction execution by holding and updating the instruction address each cycle in the single-cycle RISC-V processor.

Instruction Memory – Single Cycle RISC-V Processor

The Instruction Memory is one of the most important components of the single-cycle RISC-V processor. It is responsible for storing the program instructions and providing them to the processor during execution.

Purpose

The Instruction Memory stores all the machine code instructions of a program and provides the current instruction to be executed based on the Program Counter (PC) value**.**

Working Principle

1. The Program Counter (PC) holds the address of the current instruction.
2. This address (PC) is sent to the Instruction Memory.
3. The Instruction Memory reads the instruction stored at that address.
4. The fetched instruction is then sent to the Control Unit, Register File, and Immediate Generator (ImmGen) for decoding and execution.

Inputs

- Address (PC):The current instruction address, provided by the Program Counter.

Outputs

- Instruction:
  The 32-bit instruction located at the given address.

Operation Example

If PC = 1000,then the Instruction Memory outputs the instruction stored at memory location 1000, for example: Instruction = 0x002082B3 → add x5, x1, x2This instruction is then decoded by the Control Unit.

Used In

- All instruction types: R-type, I-type, S-type, B-type, and J-type.Every instruction (arithmetic, load/store, or branch) must first be fetched from the Instruction Memory.

Connections

Component	**Connection**
Input	**Program Counter (PC) address**
Output	**Instruction (to Control Unit, Register File, ImmGen)**

Example

If your program contains:

Address	Instruction
1000	lw x5, 0(x1)
1004	add x6, x5, x2
1008	sw x6, 4(x1)

When PC = 1000 → Instruction = lw x5, 0(x1)

When PC = 1004 → Instruction = add x6, x5, x2

Summary Table

Feature	Description
Input	Program Counter (PC)
Output	32-bit instruction
Purpose	Fetches instruction for execution
Used In	Every instruction cycle
Type	Read-only (ROM type memory)

Summary

The Instruction Memory in a single-cycle RISC-V processor is a read-only memory that stores the program's machine code.It takes the PC address as input, outputs the corresponding 32-bit instruction, and sends it to the Control Unit and other blocks for decoding.It ensures that the processor fetches one instruction per clock cycle, enabling smooth sequential execution of the program.

Register File – Single Cycle RISC-V Processor

The Register File is a small, fast memory block inside the processor that stores data temporarily for quick access during instruction execution. It contains 32 general-purpose registers (x0–x31), each 32 bits wide in RISC-V architecture.

Functions

1. Read Operands: Provides the values of the source registers (rs1 and rs2) to the ALU for computation.
2. Write Result: Stores the output of the ALU or memory into the destination register (rd) after execution.
3. Constant Zero Register: Register x0 is permanently set to 0 and cannot be modified.

Connections

Inputs:

- rs1, rs2, rd (register addresses from instruction)
- Write Data (from ALU or Memory)
- RegWrite (control signal that enables writing)

Outputs:

- Read Data 1 → value of register rs1
- Read Data 2 → value of register rs2

Operation

1. During Decode, the instruction specifies which registers to read (rs1, rs2).
2. The Register File outputs their values to the ALU inputs.
3. After execution, if RegWrite = 1, the result is written into the destination register (rd) at the next clock edge.

Example

For instruction: add x5, x1, x2

- Read register **x1** and **x2** from Register File.
- ALU computes x1 + x2.
- Result is written back into **x5**.

Summary:

The Register File in a single-cycle RISC-V processor provides fast data access for instructions by reading source operands and writing execution results, ensuring smooth data flow between the Control Unit, ALU, and Memory.

Immediate Generator (ImmGen) – Single Cycle RISC-V Processor

The Immediate Generator (ImmGen) is a hardware block that extracts and sign-extends the immediate (constant) value from an instruction. This value is used by the ALU, branch, or load/store operations in the single-cycle RISC-V processor.

Functions

1. Extract Immediate: Takes specific bits from the instruction depending on its format (I, S, B, U, J types).
2. Sign Extend: Extends the immediate value to 32 bits, preserving the sign bit (for negative values).
3. Send to ALU or PC: Provides the immediate value to the ALU for calculations or to the PC for branch/jump address generation.

Connections

- Input: 32-bit instruction from the Instruction Memory.
- Output: 32-bit immediate value to ALU or Branch Unit.
- Controlled by: Instruction opcode (decides which type of immediate to generate).

Immediate Formats

Type	Bits Used	Example Use	Immediate Construction
I-type	[31:20]	Load, ALU immediate	imm[11:0] = instr[31:20]
S-type	[31:25][11:7]	Store instructions	imm[11:5]=instr[31:25], imm[4:0]=instr[11:7]
B-type	[31], [7], [30:25], [11:8]	Branch instructions	imm[12]=instr[31], imm[10:5]=instr[30:25], imm[4:1]=instr[11:8], imm[11]=instr[7]
U-type	[31:12]	LUI, AUIPC	imm[31:12]=instr[31:12], rest=0
J-type	[31], [19:12], [20], [30:21]	JAL instruction	imm[20]=instr[31], imm[10:1]=instr[30:21], imm[11]=instr[20], imm[19:12]=instr[19:12]

Operation

1. Instruction bits enter ImmGen.
2. ImmGen checks the **opcode** to determine the instruction type.
3. It extracts the corresponding bits, arranges them correctly, and **sign-extends** them to 32 bits.
4. The generated immediate is sent to **ALU input** (for immediate operations) or **PC adder** (for branches/jumps).

Example

Instruction: addi x5, x6, 12

- Type: **I-type**
- Immediate bits = 000000001100 (12 in binary)
- ImmGen Output = 00000000000000000000000000001100 (32-bit value = 12)
- ALU performs x5 = x6 + 12.

Summary:

The Immediate Generator (ImmGen) in a single-cycle RISC-V processor decodes the instruction to extract and sign-extend the immediate value used in ALU, branch, and memory operations, enabling correct execution of immediate-based instructions.

Control Unit – Single Cycle RISC-V Processor

The Control Unit is the main decision-making part of the processor. It interprets the instruction's opcode and generates control signals that direct the operation of all other components in the single-cycle RISC-V processor.

Functions

1. Decode Instruction: Reads the opcode field (bits [6:0]) of the instruction.
2. Generate Control Signals: Produces specific signals that control the ALU, Register File, Data Memory, MUXes, and Program Counter.
3. Coordinate Data Flow: Ensures correct data movement between processor components during a single clock cycle.

Connections

Input:

- Opcode field (7 bits) from the Instruction Memory.

Outputs:

- Multiple control signals such as:
    - RegWrite → enables writing to Register File.
    - ALUSrc → selects between register or immediate for ALU input.
    - MemRead → enables reading from Data Memory.
    - MemWrite → enables writing to Data Memory.
    - MemtoReg → selects ALU result or memory data for register write-back.
    - Branch → controls branch decision.
    - ALUOp → tells ALU which operation to perform.

Main Control Signals Table

Signal	Function	Example
RegWrite	Enables writing result to Register File	For add, lw
ALUSrc	Selects 2nd ALU input (0=Register, 1=Immediate)	For addi, lw, sw
MemRead	Reads data from Data Memory	For lw
MemWrite	Writes data to Data Memory	For sw
MemtoReg	Selects between ALU output or memory output	For lw
Branch	Enables branch address computation	For beq, bne
ALUOp	Decides ALU function (AND, OR, ADD, SUB, etc.)	Based on instruction type

Operation

1. The instruction opcode is sent to the Control Unit.
2. The Control Unit decodes the opcode and sets control signals accordingly.
3. These signals control data paths and operation types for that instruction.
4. The entire process completes in one clock cycle in the single-cycle processor.

Example

Instruction: lw x5, 0(x1)

- Opcode → Load instruction (0000011)
- Control Signals:
    - RegWrite = 1
    - ALUSrc = 1 (use immediate)
    - MemRead = 1
    - MemWrite = 0
    - MemtoReg = 1
    - Branch = 0

summary:

The Control Unit in a single-cycle RISC-V processor decodes the instruction opcode and generates control signals to coordinate all Datapath components. It ensures the correct operation of the ALU, Register File, Memory, and MUXes, allowing each instruction to execute correctly within a single clock cycle.

ALU Control Unit – Single Cycle RISC-V Processor

The ALU Control Unit is a subcomponent of the main Control Unit that decides which specific operation the ALU should perform, such as addition, subtraction, AND, OR, or set less than.It uses signals from the main Control Unit and the function fields (funct3 and funct7) of the instruction to generate a precise ALU control signal.

Functions

1. Interpret Control Signals: Takes the ALUOp (from the main Control Unit) and funct3/funct7 fields (from the instruction).
2. Generate ALU Operation Code: Produces a smaller control signal (usually 4 bits) that tells the ALU which arithmetic or logic operation to perform.
3. Support Different Instruction Types: Handles operations for R-type, I-type, Branch, Load, and Store instructions.

Connections

Inputs:

- ALUOp → from Main Control Unit (defines instruction type).
- funct3, funct7 → from Instruction (define operation within type).

Output:

- ALUControl → 4-bit signal to ALU to perform the correct operation.

Operation

1. The Main Control Unit sets ALUOp based on instruction type:
   - 00 → Load/Store (use add)
   - 01 → Branch (use subtract)
   - 10 → R-type (use funct3/funct7 to decide)
2. The ALU Control Unit reads funct3 and funct7 bits.
3. Based on both inputs, it generates the ALUControl signal that selects the ALU operation.

Example Table

ALUOp	**funct7**	**funct3**	**ALU Operation**	**ALUControl Code**
00	x	x	ADD (for lw/sw)	0010
01	x	x	SUB (for beq)	0110
10	0000000	000	ADD	0010
10	0100000	000	SUB	0110
10	0000000	111	AND	0000
10	0000000	110	OR	0001
10	0000000	010	SLT	0111

Example

Instruction: sub x5, x6, x7

- ALUOp = 10 (R-type)
- funct7 = 0100000, funct3 = 000
- ALU Control Unit outputs 0110 → tells ALU to perform SUBTRACT operation.

Summary

The ALU Control Unit in a single-cycle RISC-V processor refines the signals from the main Control Unit and instruction fields to produce the exact control signal needed by the ALU.It ensures that the ALU performs the correct operation (add, sub, and, or, etc.) for each instruction type within one clock cycle.

Arithmetic Logic Unit (ALU) – Single Cycle RISC-V Processor

The Arithmetic Logic Unit (ALU) is the main computational component of the single-cycle RISC-V processor. It performs all arithmetic and logical operations required by instructions such as addition, subtraction, AND, OR, and comparisons.

Functions

1. Perform Arithmetic Operations: Executes add, subtract, set less than, etc.
2. Perform Logical Operations: Executes AND, OR, XOR, shift, etc.
3. Generate Status Flags: Produces results like Zero (used in branch decisions).
4. Support All Instruction Types: Used in R-type, I-type, Load/Store, and Branch operations.

Connections

Inputs:

- Operand A → from Register File (rs1).
- Operand B → from MUX (either Register File rs2 or Immediate from ImmGen).
- ALUControl → from ALU Control Unit (decides operation type).

Output:

- Result → sent to Register File, Data Memory, or Branch Unit.
- Zero Flag → sent to Control Unit for branch decisions.

Operation

1. The Control Unit sets control signals based on the instruction type.
2. The ALU Control Unit specifies the exact operation to perform (via ALUControl).
3. The ALU takes two operands (A and B) and performs the selected arithmetic/logical operation.
4. The result is sent to the destination register or memory as required.

Common ALU Operations

Operation	Description	Example
ADD	Adds two operands	add x5, x1, x2 → x5 = x1 + x2
SUB	Subtracts second operand from first	sub x5, x1, x2 → x5 = x1 - x2
AND	Bitwise AND	and x5, x1, x2
OR	Bitwise OR	or x5, x1, x2
SLT	Set Less Than (1 if x1 < x2)	slt x5, x1, x2
ADD (for lw/sw)	Used to calculate memory address	lw x5, 0(x1) → address = x1 + 0

Example

Instruction: beq x1, x2, LABEL

- ALU inputs: x1, x2
- ALU operation: SUB (x1 - x2)
- If result = 0 → Zero Flag = 1 → branch is taken.

Summary

The ALU in a single-cycle RISC-V processor is responsible for executing all arithmetic and logical computations. It receives inputs from the Register File or Immediate Generator, performs the operation selected by the ALU Control Unit, and provides the result to the Register File, Data Memory, or Branch Unit — all within a single clock cycle.

Data Memory – Single Cycle RISC-V Processor

The Data Memory is a memory unit in the single-cycle RISC-V processor that is used to store and retrieve data during program execution.It is mainly accessed by load (lw) and store (sw) instructions.

Functions

1. **Read Data**: For load instructions (lw) — reads data from the specified memory address.
2. **Write Data:** For store instructions (sw) — writes data from a register into the specified memory address.
3. **Address Access**: The memory address is generated by the ALU, usually as the sum of a base register and an immediate offset.

Connections

Inputs:

- Address → from **ALU output** (memory location).
- Write Data → from **Register File (rs2)** (data to store).

- MemRead and MemWrite → control signals from **Control Unit**.

Outputs:

- Read Data → to **MUX** (and then to **Register File**) when reading data.

Operation

1. For **Load Instruction (lw)**
   - ALU computes memory address = rs1 + immediate.
   - MemRead = 1, MemWrite = 0.
   - Data is read from memory and sent to the Register File.
2. For **Store Instruction (sw)**
   - ALU computes memory address = rs1 + immediate.
   - MemRead = 0, MemWrite = 1.
   - Data from register (rs2) is written to that address.
3. For other instructions (like arithmetic), both MemRead and MemWrite are **0**, so Data Memory remains **inactive**.

Example

Instruction	**Action**	**Description**
lw x5, 0(x1)	Load	Reads data from memory address = x1 + 0 into register x5
sw x5, 4(x1)	Store	Writes content of x5 into memory address = x1 + 4

Control Signals Used

Signal	**Purpose**
MemRead	Enables reading from memory
MemWrite	Enables writing to memory
MemtoReg	Selects whether to write ALU result or memory data to register

Summary

The Data Memory in a single-cycle RISC-V processor stores and retrieves data for load and store instructions. It receives the address from the ALU, uses control signals from the Control Unit to determine read/write operations, and transfers data to or from the Register File — all within one clock cycle.

Multiplexers (MUX) in Single-Cycle RISC-V Processor

In a single-cycle RISC-V processor, several Multiplexers (MUXes) are used to control data flow between different units. A MUX (Multiplexer) selects one input from multiple sources based on a control signal and sends it to the output. It acts like a switch that decides which data path to follow at each stage of instruction execution.There are mainly three important MUXes in the single-cycle RISC-V processor.

1. MUX Between Register File and ALU (ALUSrc MUX)

Purpose:

This MUX selects the second input of the ALU — either a register value or an immediate value.

Inputs:

- Input 0 → Register file output (**ReadData2**)
- Input 1 → Immediate value (**ImmGen output**)

Control Signal:

- **ALUSrc**
    - 0 → Select Register operand (for R-type instructions)
    - 1 → Select Immediate operand (for I-type or memory instructions)

Used In:

- R-type: ALU gets both operands from registers.
- I-type / lw / sw: ALU gets one operand from register, other from ImmGen.

2. MUX Between ALU Output and Data Memory (MemtoReg MUX)

Purpose:

This MUX decides which data will be written back to the Register File.

Inputs:

- Input 0 → ALU result (arithmetic/logical result)
- Input 1 → Data Memory output (loaded data)

Control Signal:

- **MemtoReg**
    - 0 → Write ALU result to Register (for R-type, I-type)
    - 1 → Write Memory data to Register (for lw)

Used In:

- **R-type**: Write ALU output.
- **lw:** Write data read from memory.

3. MUX for Next PC Selection (PCSrc MUX)

Purpose:

This MUX selects the next value of the Program Counter (PC) — deciding whether to go to the next instruction or branch target.

Inputs:

- Input 0 → PC + 4 (next sequential instruction)
- Input 1 → Branch Target Address (from adder)

Control Signal:

- **PCSrc**
  - 0 → Normal sequence (next instruction)
  - 1 → Branch taken (jump to branch address)

Used In:

- **beq / bne (Branch Instructions)**
  - If branch condition true → select Branch Address.
  - Else → select PC + 4.

Summary Table

MUX Name	**Inputs**	**Control Signal**	**Purpose**
ALUSrc MUX	Register or Immediate	ALUSrc	Selects ALU 2nd operand
MemtoReg MUX	ALU Output or Memory Data	MemtoReg	Selects data to write in Register
PCSrc MUX	PC+4 or Branch Address	PCSrc	Selects next Program Counter

Example:

For instruction lw x5, 0(x1)

1. ALUSrc MUX → selects Immediate (offset 0)
2. MemtoReg MUX → selects Data Memory output
3. PCSrc MUX → selects PC + 4 (no branch)

Summary

The three MUXes in a single-cycle RISC-V processor ensure correct data path selection:

- One controls ALU input,
- One controls Register write data, and
- One controls Program Counter update.These MUXes, guided by Control Unit signals, make the processor flexible for different instruction types — all within a single clock cycle.

AND Gate in Single-Cycle RISC-V Processor

The AND gate in the single-cycle RISC-V processor plays a small but very important role in branch decision making.It helps determine whether the Program Counter (PC) should move to the next instruction (PC + 4) or jump to a branch target address.

Purpose

The AND gate is used to combine two control signals:

1. The Branch signal (from the Control Unit)
2. The Zero signal (from the ALU)

Its output controls the PCSrc MUX, which decides the next value of the Program Counter.

Working Principle

1. The Control Unit generates a signal called Branch:
    - Branch = 1 → instruction is a branch (e.g., beq)
    - Branch = 0 → normal instruction (e.g., add, lw, sw)
2. The ALU performs subtraction between two registers (for branch comparison) and sets:
    - Zero = 1 → if both operands are equal
    - Zero = 0 → if operands are not equal
3. The AND gate receives both signals:
4. PCSrc = Branch AND Zero
    - If both are 1, the branch condition is true, and the PC will jump to the branch target address.
    - If any is 0, the next instruction (PC + 4) will be executed.

Example:

Instruction: beq x1, x2, LABEL

- ALU computes: x1 - x2
- If x1 == x2, then Zero = 1
- Control Unit sets Branch = 1 for beq
- AND gate output = 1 × 1 = 1
- PCSrc = 1 → PC updates to branch target address (LABEL)

If x1 ≠ x2, then **Zero = 0**, so:

- AND gate output = 1 × 0 = 0
- PCSrc = 0 → PC moves to next instruction (PC + 4)

Connections

Inputs:

- Branch (from Control Unit)
- Zero (from ALU)

Output:

- PCSrc (to the PC MUX)

Summary Table

Signal	**Value**	**Effect on PC**
Branch = 0, Zero = 0	0	Next instruction (PC + 4)
Branch = 1, Zero = 0	0	No branch taken
Branch = 1, Zero = 1	1	Branch taken
Branch = 0, Zero = 1	0	No branch taken

Summary

The AND gate in a single-cycle RISC-V processor ensures correct branch control by combining the Branch signal (from Control Unit) and Zero flag (from ALU).If both are high, the processor jumps to the branch address; otherwise, it continues sequentially. This simple logic enables conditional branching within one clock cycle.

Adders in Single-Cycle RISC-V Processor

In a single-cycle RISC-V processor, there are two main adders used to perform different types of additions. Both are essential for instruction sequencing and branch address calculation.Each adder performs simple arithmetic addition, but they serve different purposes in the datapath.

1. First Adder (PC + 4 Adder)

Purpose

This adder calculates the address of the next sequential instruction.Since each RISC-V instruction is 32 bits (4 bytes) long, the next instruction's address is PC + 4.

Working

- Input: Current Program Counter (PC)
- Operation: Adds 4 to the current PC value
- Output: PC + 4 → the address of the next instruction

Use

- Used for normal (non-branch) instructions like add, sub, lw, sw, etc.
- The output (PC + 4) is sent to the MUX (PCSrc MUX).
- If no branch is taken, this value is chosen as the next PC.

Example

If PC = 1000,then PC + 4 Adder Output = 1004,so the next instruction will be fetched from address 1004.

2. Second Adder (Branch Target Adder)

Purpose

This adder calculates the branch target address, which is the possible jump location when a branch instruction is taken (like beq, bne).

Working

- Inputs:
    1. PC + 4 (from the first adder)
    2. Shifted Immediate value (output from the "Shift Left 1" block of ImmGen)
- Operation: Adds PC + 4 and the shifted immediate to get the branch target address.
- Output: Branch Target Address

Use

- Used only for branch instructions.
- The output of this adder goes to the PCSrc MUX.
- If the AND gate output (Branch AND Zero) is 1, this branch address becomes the next PC.

Example

For instruction beq x1, x2, 16,

- Immediate = 16 (shifted left by 1 → 32)
- PC + 4 = 1004
- Branch Target = 1004 + 32 = 1036

So, if the branch is taken, the next instruction is fetched from address 1036.

Connections Summary

Adder	Inputs	Output	Purpose
Adder 1 (PC + 4)	PC, 4	PC + 4	Next sequential instruction
Adder 2 (Branch Target)	PC + 4, Shifted Immediate	Branch Address	Branch jump target

Interaction with Other Components

- **Adder 1** output (PC + 4) → goes to:
    - **Instruction Memory** (for next fetch)
    - **Adder 2** (for branch target calculation)
    - **MUX (PCSrc)** (as one possible next PC)
- **Adder 2** output (Branch Address) → goes to:
    - **MUX (PCSrc)** to select if branch is taken.

Summary

In a single-cycle RISC-V processor:

- First Adder (PC + 4) → Always active, calculates the next sequential instruction address.
- Second Adder (Branch Target) → Active only during branch instructions, computes the jump address using the immediate offset.
  Together, these adders ensure correct instruction flow for both sequential and branching executions — all within one clock cycle.