UCLA Samueli School of Engineering

# CS-M151B
# Computer Systems Architecture

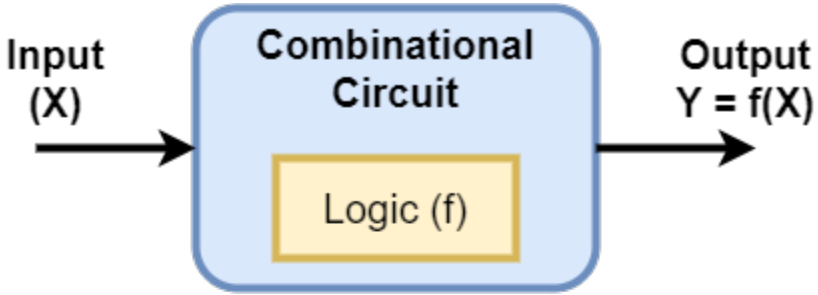
Blaise Tine
UCLA Computer Science

# Review: Digital Circuits

# Combinational Circuits

**Combinational circuits**

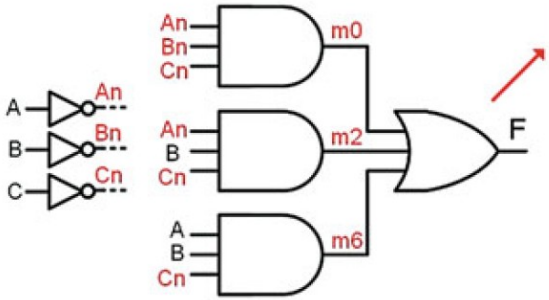
- **Output depends on current inputs only**
- **Consist of a connection of multiple logic gates**
  - **And/Or/Not**



# Combinational Circuits

## Combinational circuits

- **Example1**




```
module SystemX (output wire F,
                input  wire A, B, C);

   wire  An, Bn, Cn;   // internal nets
   wire  m0, m2, m6;

   assign An = ~A;              // Not's
   assign Bn = ~B;
   assign Cn = ~C;

   assign m0 = An & Bn & Cn;  // AND's
   assign m2 = An & B  & Cn;
   assign m6 = A  & B  & Cn;

   assign F  = m0 | m2 | m6;  // OR

endmodule
```

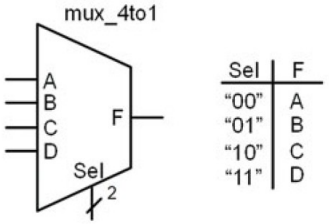


From: Quick start Guide to Verilog (© springer Nature Switzerland)

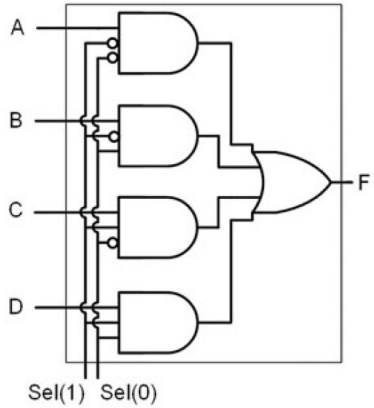
# Combinational Circuits

**Combinational circuits**

- **Example2: multiplexer**




| Sel | F |
|---|---|
| "00" | A |
| "01" | B |
| "10" | C |
| "11" | D |




The following shows how to model the behavior of the mux using continuous assignment and logical operators.

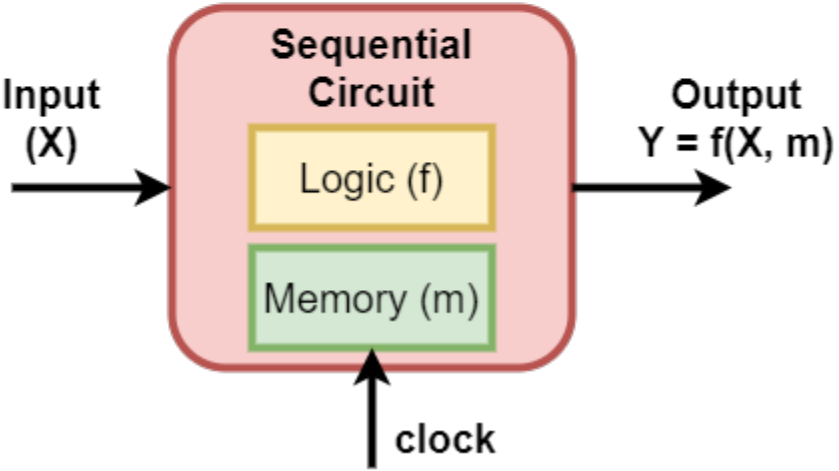
```
module mux_4to1 (output wire F,
                 input  wire A, B, C, D,
                 input  wire [1:0] Sel);

   assign F = (A & ~Sel[1] & ~Sel[0]) |
              (B & ~Sel[1] &  Sel[0]) |
              (C &  Sel[1] & ~Sel[0]) |
              (D &  Sel[1] &  Sel[0]);

endmodule
```

From: Quick start Guide to Verilog (© springer Nature Switzerland)

# Sequential Circuits

**Sequential circuits**

- **Output depends on current inputs and prior state**
- **Need a storage/memory to hold prior state**
- **e.g. storage element**
  - **D-flip-flops**
  - **RAMs**

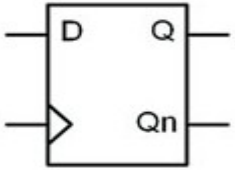


# Sequential Circuits

**D-Flip-flop:**

- Digital storage element
- Input is propagated to output at the rising edge of the clock
- **Registers are implemented using D-Flip-flops**

Example: Behavioral Model of a D-Flip-Flop in Verilog




| Clk | D | Q | Qn | |
|---|---|---|---|---|
| 0 | X | Last Q | Last Qn | Store |
| 1 | X | Last Q | Last Qn | Store |
| ⨍ | 0 | 0 | 1 | Update |
| ⨍ | 1 | 1 | 0 | Update |

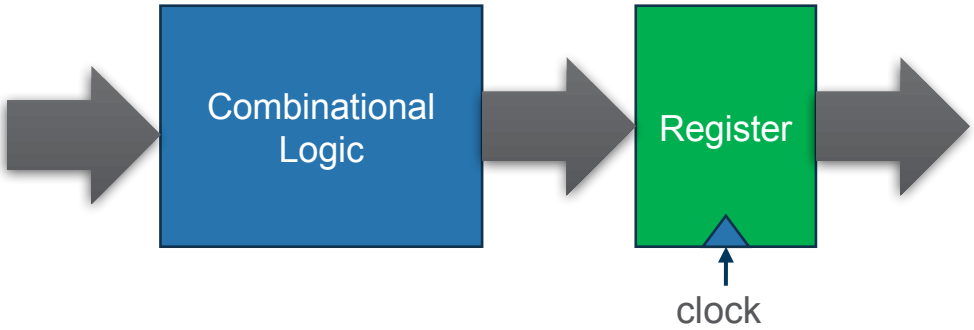
```
module dflipflop (output reg  Q, Qn,
                  input  wire Clock, D);

  always @ (posedge Clock)
     begin
        Q  <= D;
        Qn <= ~D;
     end

endmodule
```

From: Quick start Guide to Verilog (© springer Nature Switzerland)

# Applications of D-Flip-Flop

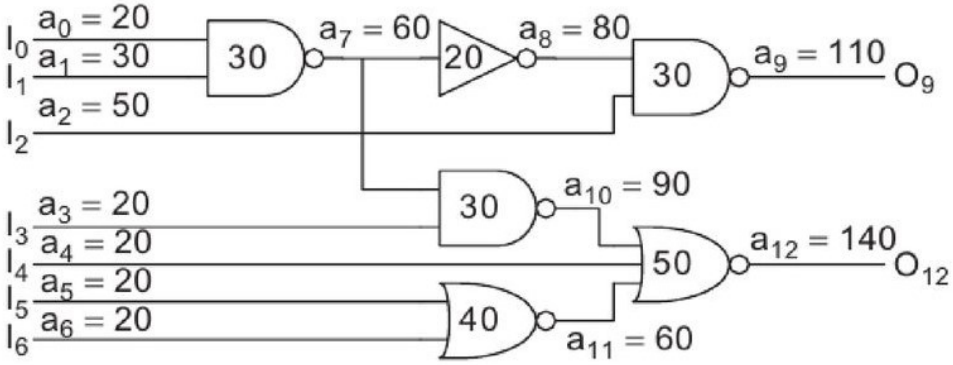
**D-Flip-flop are use for:**

- Data storage
- Data synchronization
- Finite state machines
- Pipelining
- **Timing resolution**
  - Reduced propagation delay of combinational blocks
  - Improve clock speed



# Propagation Delay and Critical Path

- **Propagation delay** (Tp)
  - Time needed for the gate to respond to a change at its inputs
  - F = 1 / Tp
- **Critical path** is the longest propagation delay in the circuit between I/O or registers
  - Fmax = 1 / Tmax







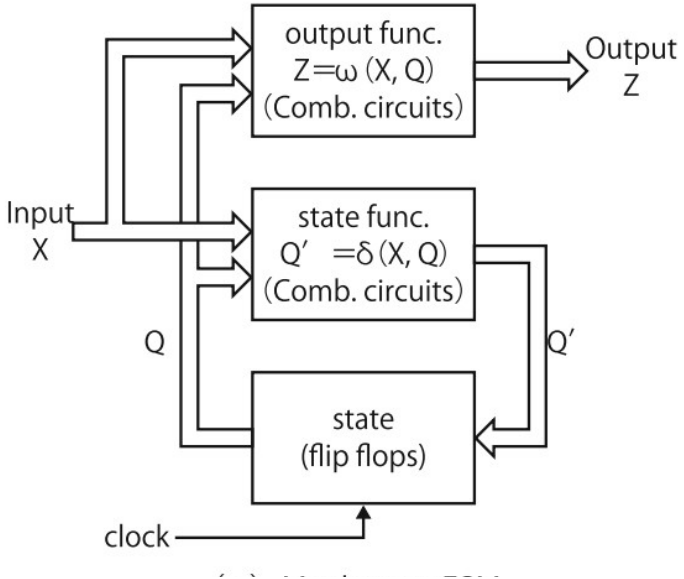

From Rudoff Hunter

# Finite State Machines

**Definition:** State-driven model of computation to design sequential circuits.

**Mealy type FSM**

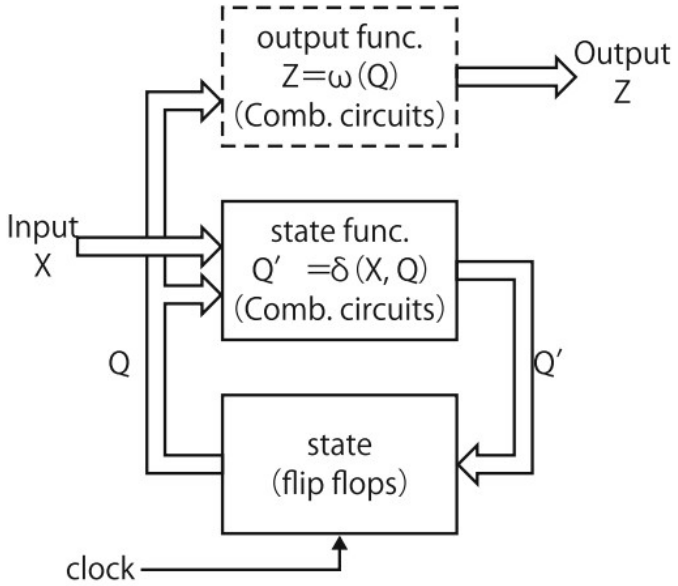
- (state, input) ☾ output
- Pros: Smaller number of FSM states
  - Some input directly contribute to output
- Cons: Input critical path
  - Why?



# Finite State Machines

**Moore type FSM**

- (state) ☾ output
- Pros: High-speed
  - Why?
- Cons: Large number of states
  - due to full inputs handling
  - can affect circuit size



# RISC-V CPU Implementation

# Basic RISC-V CPU Datapath

**RISC-V ISA subset:**

- R-type
- I-type
- LD/ST
- Branch

| | | | | | | |
|---|---|---|---|---|---|---|
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | ADD |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | SUB |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | OR |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | AND |

| | | | | | |
|---|---|---|---|---|---|
| imm[11:0] | rs1 | 000 | rd | 0010011 | ADDI |
| imm[11:0] | rs1 | 110 | rd | 0010011 | ORI |
| imm[11:0] | rs1 | 111 | rd | 0010011 | ANDI |

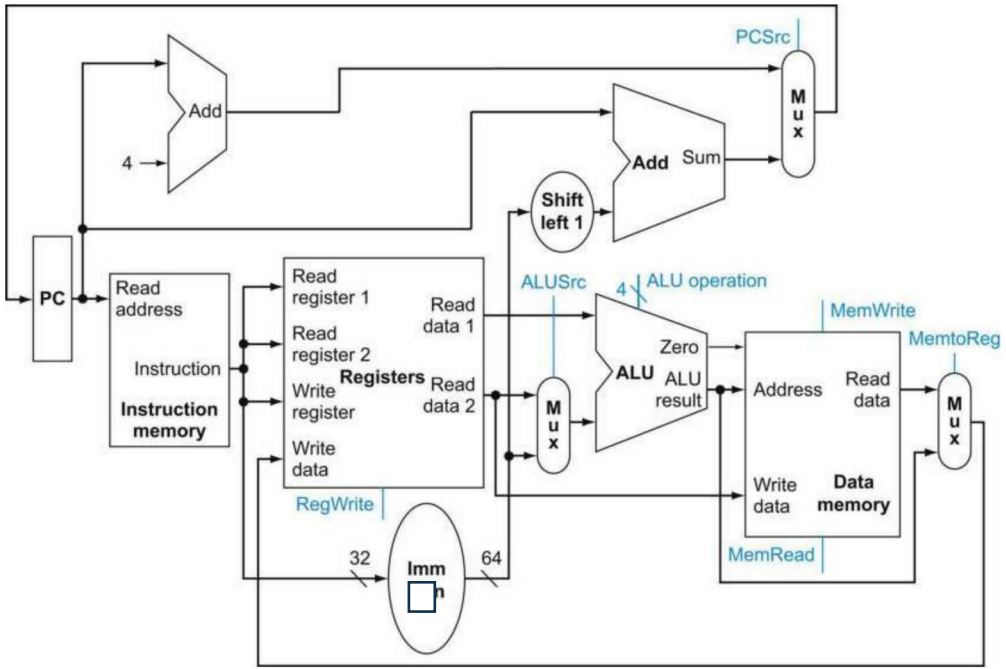
| | | | | | | |
|---|---|---|---|---|---|---|
| imm[11:0] | | rs1 | 010 | rd | 0000011 | LW |
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | SW |

| | | | | | | |
|---|---|---|---|---|---|---|
| imm[12\|10:5] | rs2 | rs1 | 000 | imm[4:1\|11] | 1100011 | BEQ |

# Basic RISC-V CPU Datapath

**Basic Steps:**

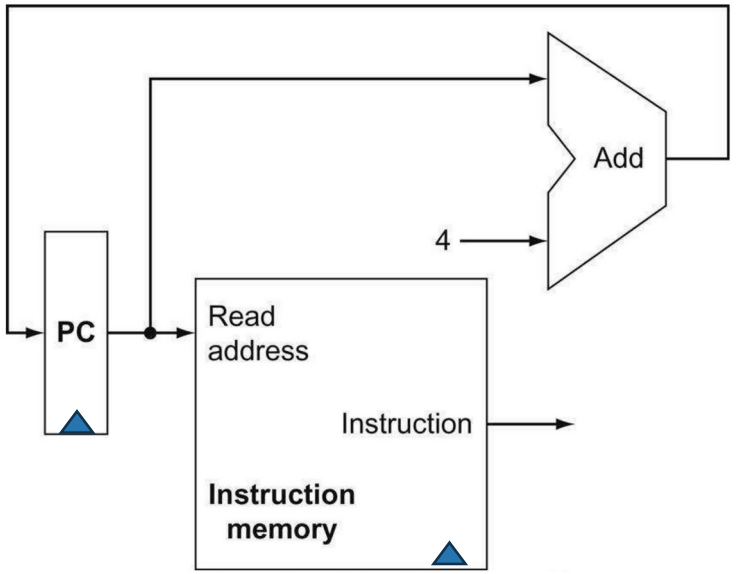
1. Fetch instruction from PC
2. Register file access
3. ALU execution
4. Memory access
5. Next PC update



# Basic RISC-V CPU Datapath

**Next PC calculation:**

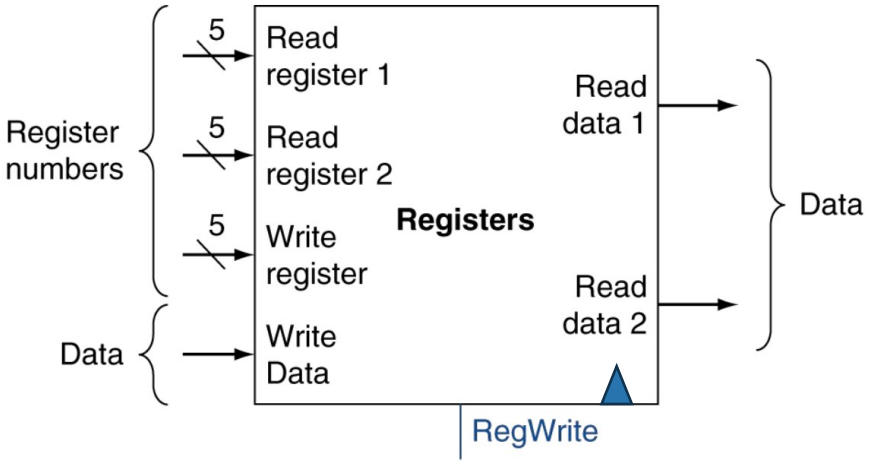
1. Program counter stored in PC
2. Can perform the addition in parallel
3. While reading the next instruction
4. Why adding 4 to PC?



# Basic RISC-V CPU Datapath

**Register File:**

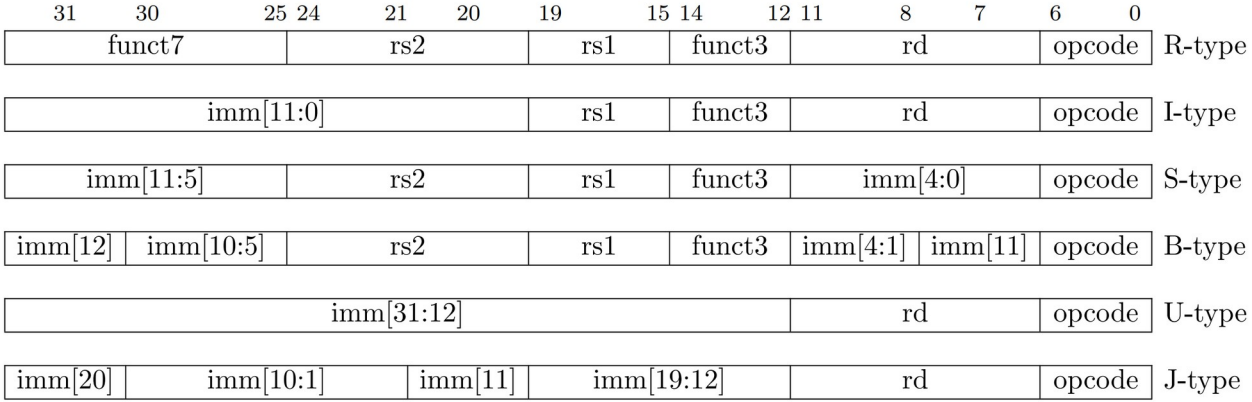
- 32 total registers (5-bit addressing)
- 2x 5-bit source register addresses
- 1x 5-bit destination register address
- RegWrite: write enable
- How to decode register address?



# Basic RISC-V CPU Datapath

**Register File:**

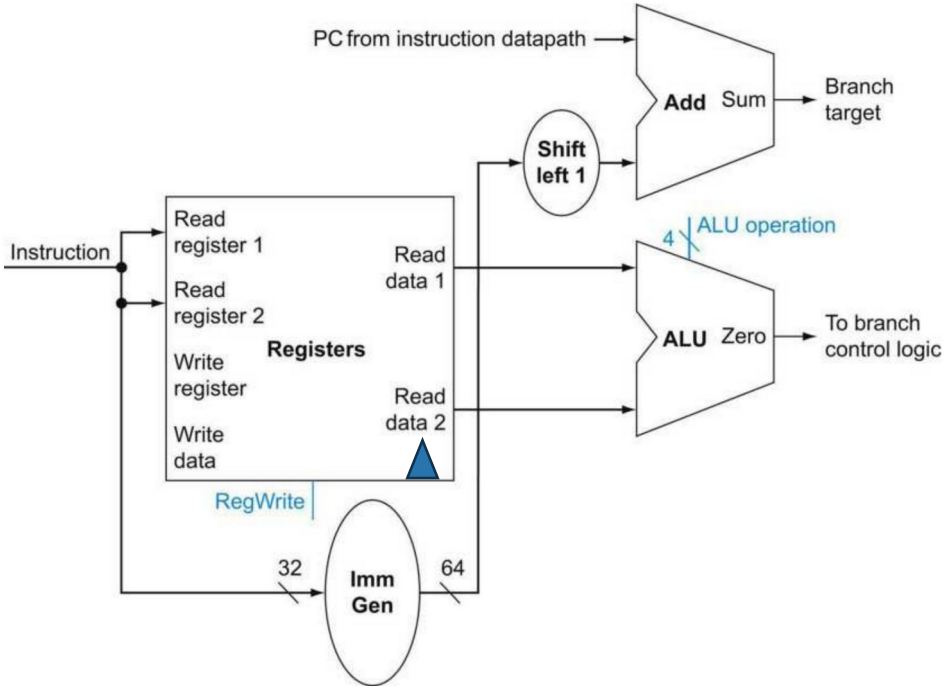
- How to decode register address?
  - No logic needed
  - why?



# Basic RISC-V CPU Datapath

**Branch operations:**

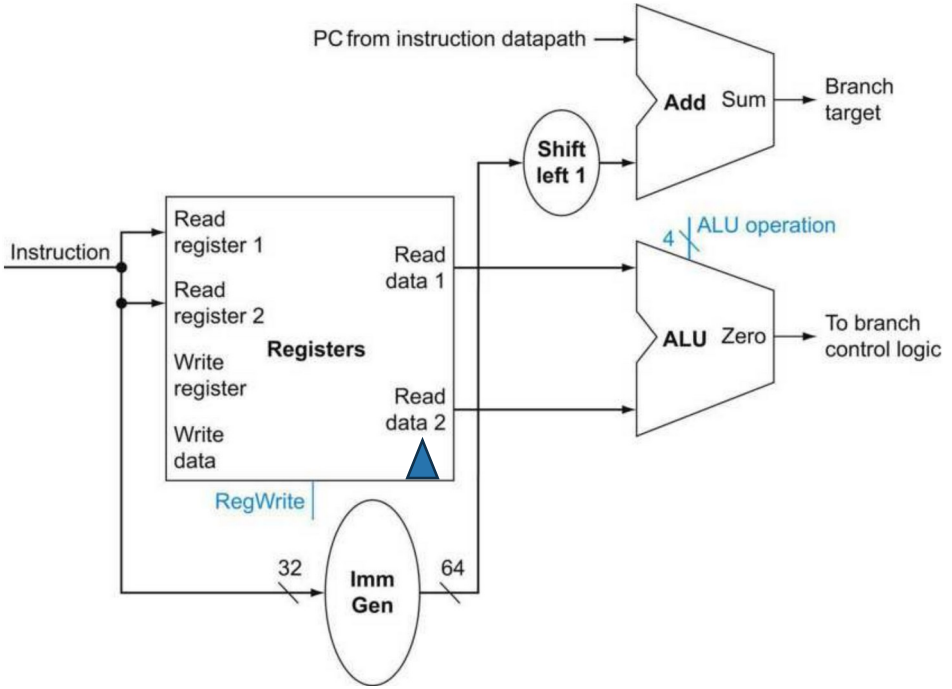
1. Register access
2. Immediate generation
3. Immediate shift
4. Branch target calculation
5. Branch condition evaluation
6. Why shift left 1?
7. Why not doing it inside Imm Ge



# Basic RISC-V CPU Datapath

**Branch operations:**

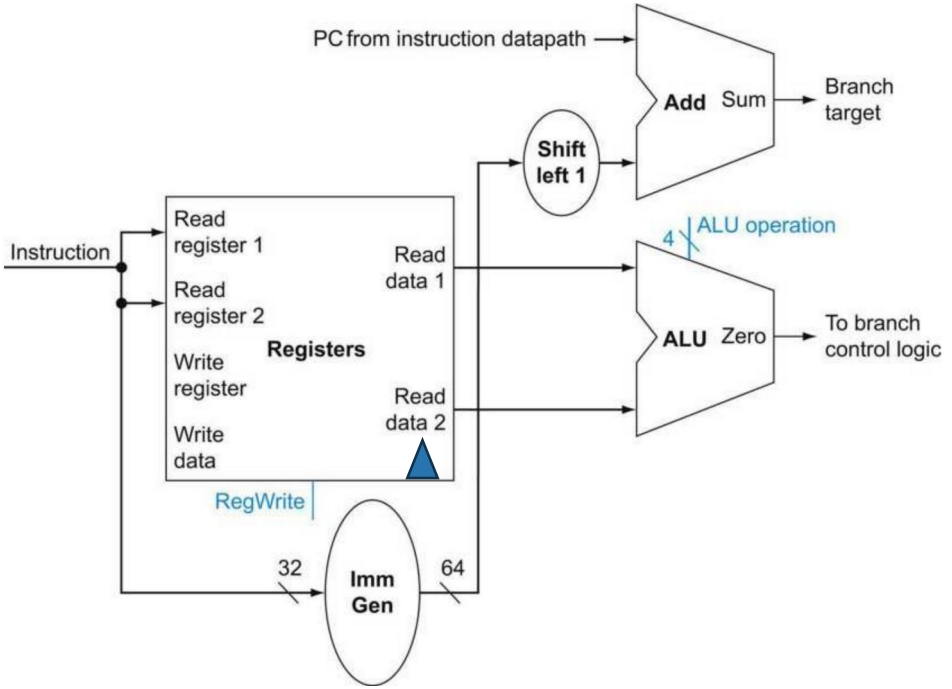
1. Register access
2. Immediate generation
3. Immediate shift
4. Branch target calculation
5. Branch condition evaluation
6. Parallelism?
7. Critical path?



# Basic RISC-V CPU Datapath

**Branch operations:**

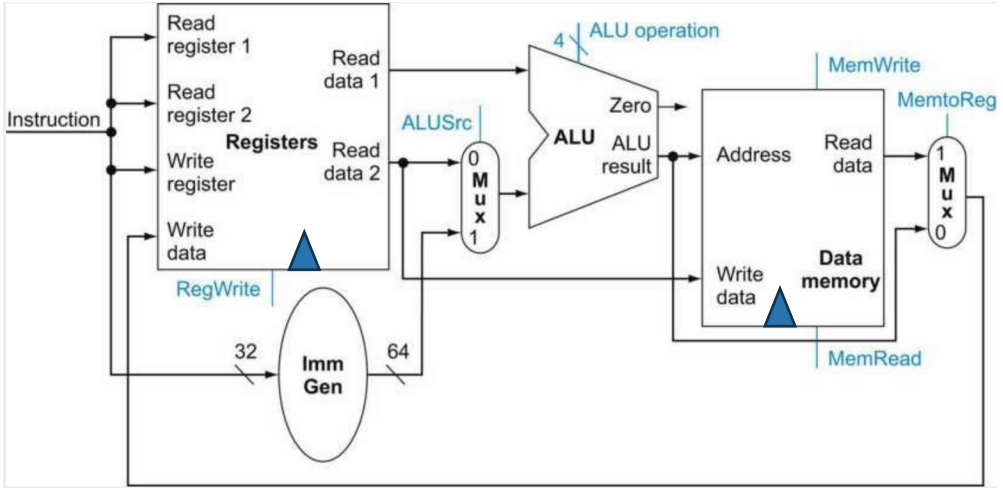
1. Register access
2. Immediate generation
3. Immediate shift
4. Branch target calculation
5. Branch condition evaluation
6. Parallelism
   1->5, 2->3->4
7. Critical path
   1->5



# Basic RISC-V CPU Datapath

**Memory Operations:**

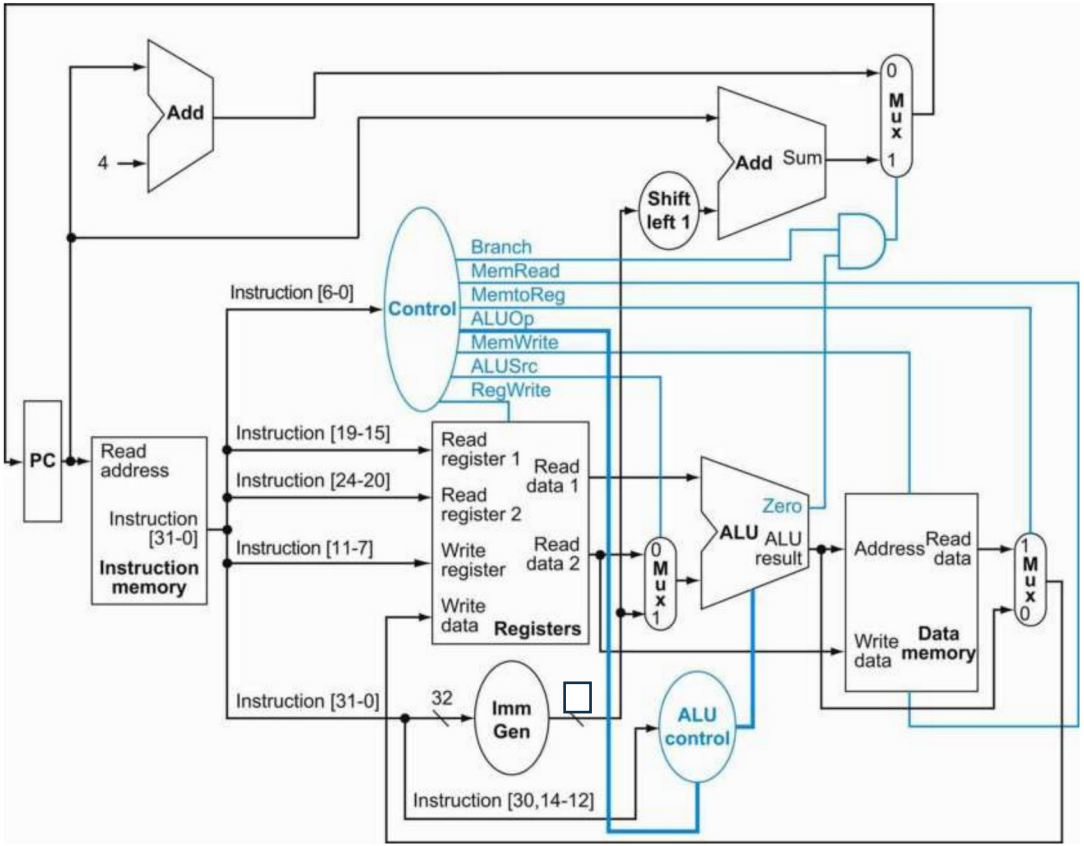
1. Address calculation
2. Memory access
3. ALUSrc select
   - 0: register
   - 1: immediate
4. Parallelism?
5. Critical path?



# Basic RISC-V CPU Control Path

**CPU control logic:**

- Main control
  - Register File
    - RegWrite
    - MemtoReg
  - ALU
    - ALUOp
    - ALUSrc
  - Data memory
    - MemRead
    - MemWrite
  - Branch
- ALU control
  - ALU operation



# Basic RISC-V CPU Datapath

**RISC-V ISA subset:**

- R-type
- I-type
- LD/ST
- Branch

| | | | | | | |
|---|---|---|---|---|---|---|
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | ADD |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | SUB |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | OR |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | AND |

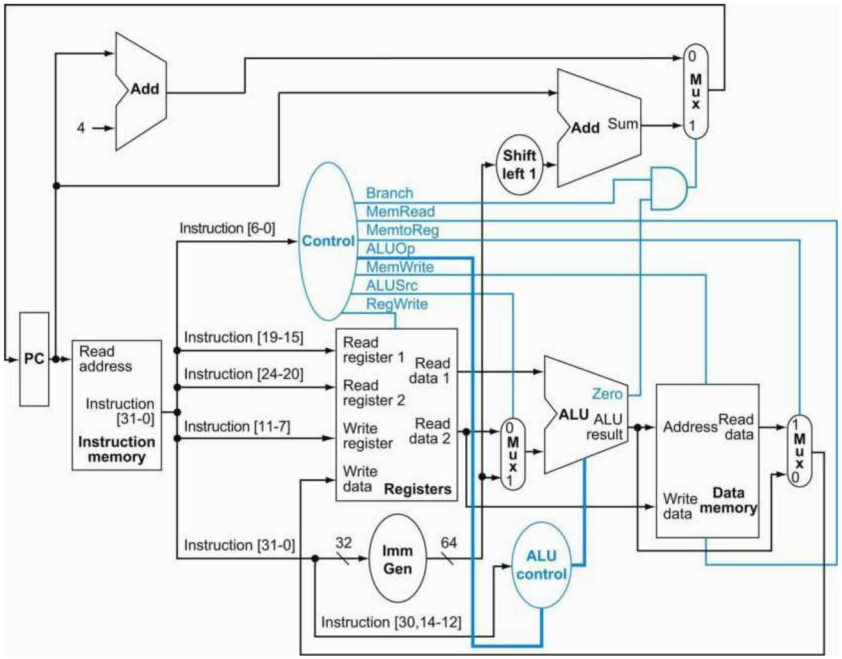
| | | | | | |
|---|---|---|---|---|---|
| imm[11:0] | rs1 | 000 | rd | 0010011 | ADDI |
| imm[11:0] | rs1 | 110 | rd | 0010011 | ORI |
| imm[11:0] | rs1 | 111 | rd | 0010011 | ANDI |

| | | | | | |
|---|---|---|---|---|---|
| imm[11:0] | | rs1 | 010 | rd | 0000011 | LW |
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | SW |

| | | | | | | |
|---|---|---|---|---|---|---|
| imm[12\|10:5] | rs2 | rs1 | 000 | imm[4:1\|11] | 1100011 | BEQ |

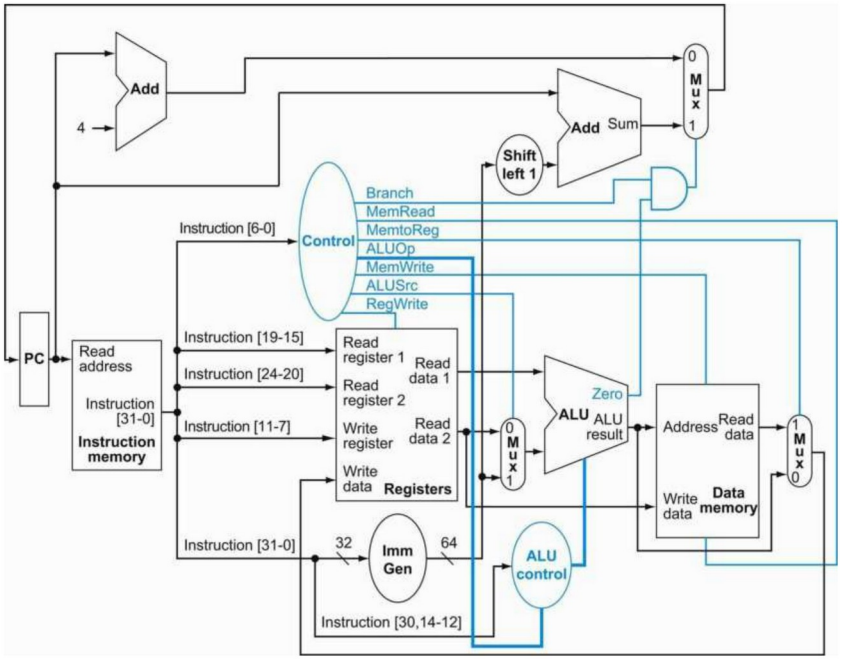
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| R-Type | 0110011 | | | | | | |



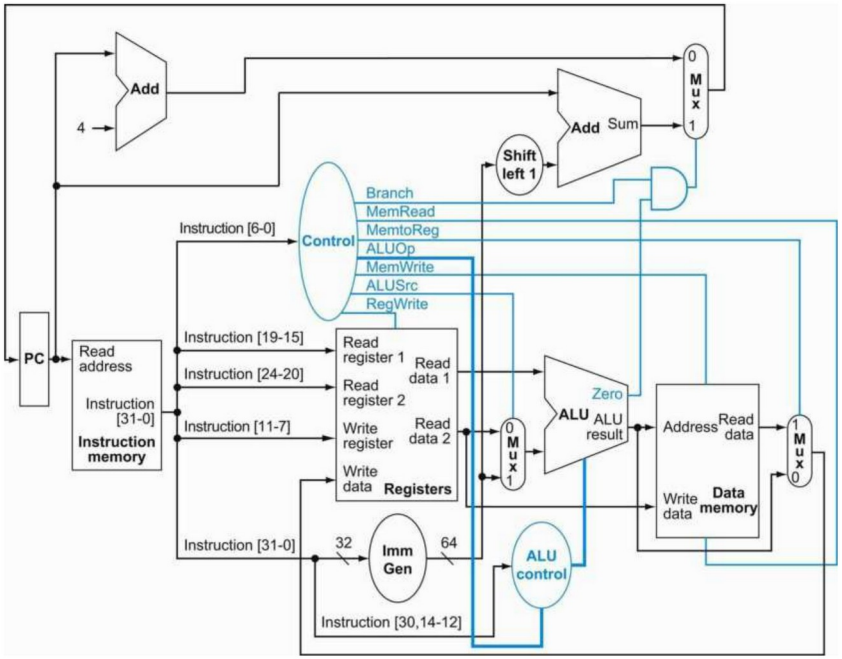
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| R-Type | 0110011 | 1 | 0 | 0 | 0 | 0 | 0 |



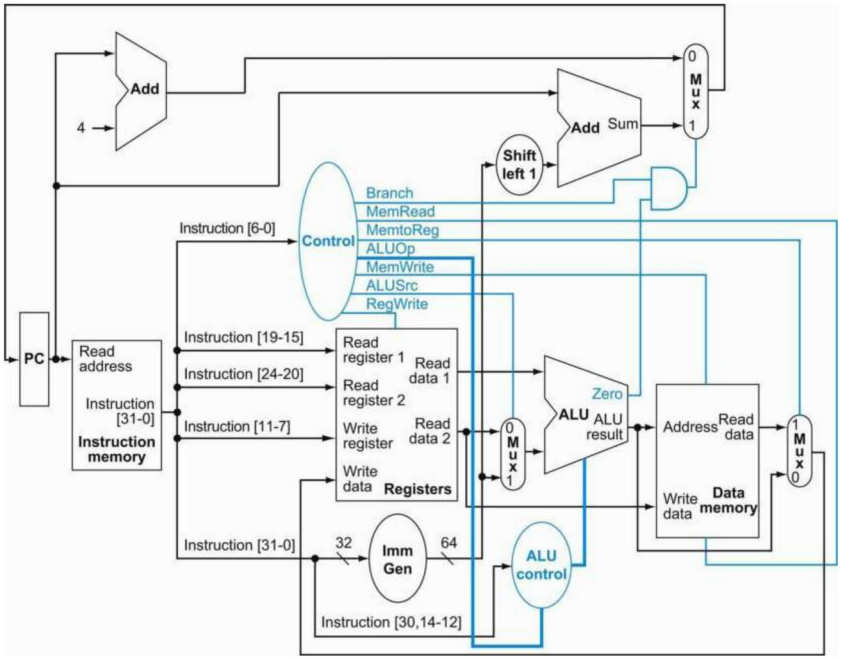
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| I-Type | 0010011 | | | | | | |



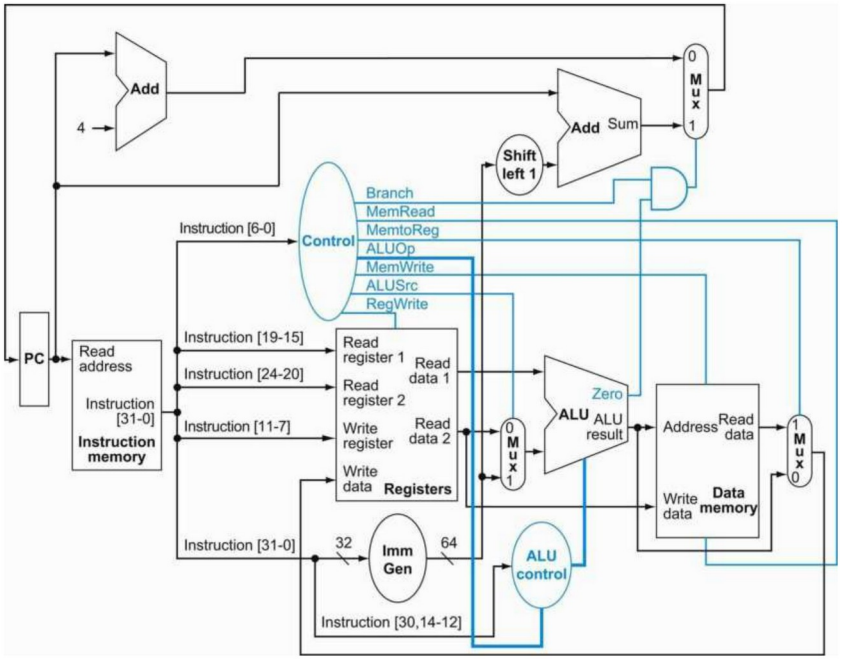
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| I-Type | 0010011 | 1 | 1 | 0 | 0 | 0 | 0 |



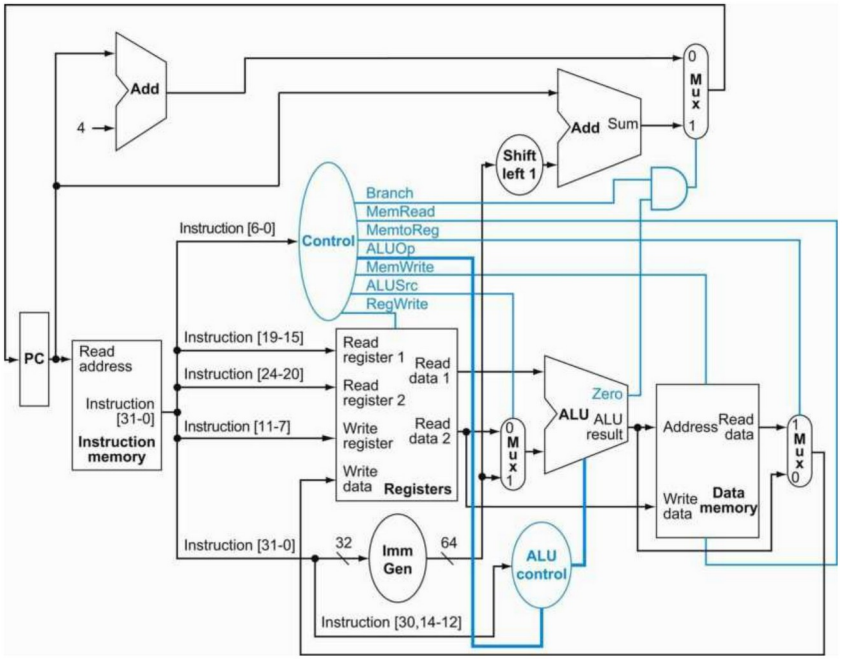
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| LW | 0000011 | | | | | | |



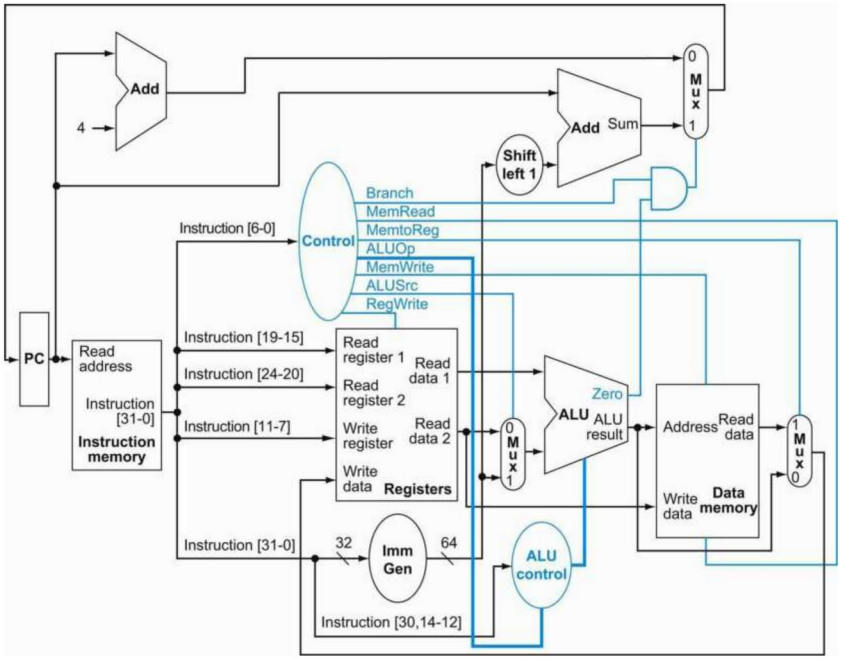
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| LW | 0000011 | 1 | 1 | 0 | 1 | 0 | 1 |



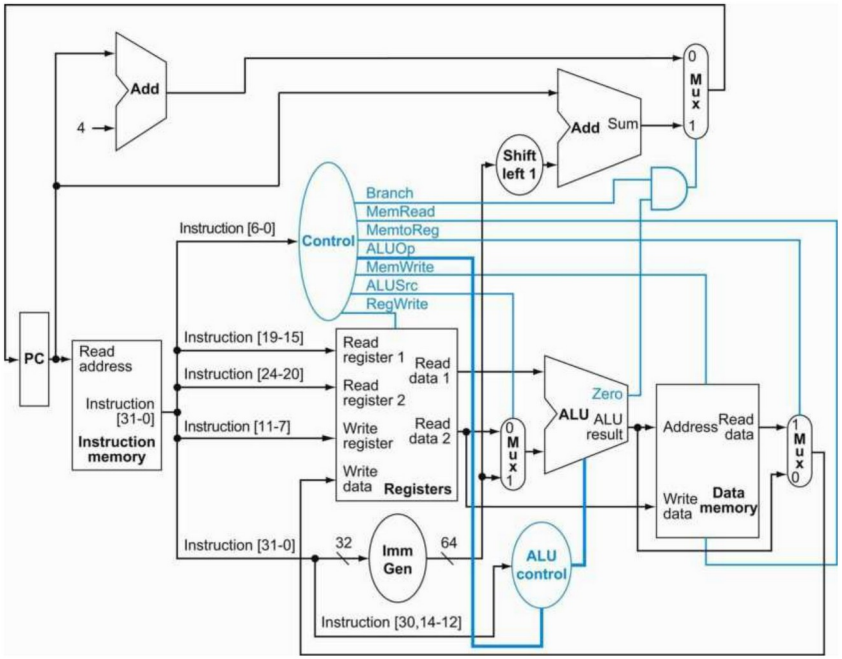
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| SW | 0100011 | | | | | | |



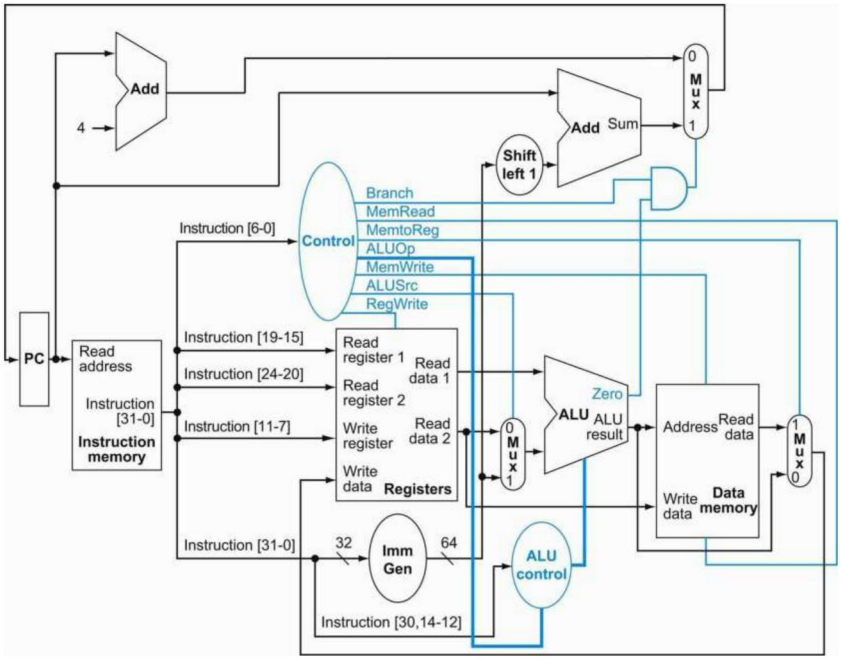
# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| SW | 0100011 | 0 | 1 | 0 | 0 | 1 | 0 |



# Basic RISC-V CPU Control Path

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| BEQ | 1100011 | 0 | 0 | 1 | 0 | 0 | 0 |



# Basic RISC-V CPU Control Path

**How to determine the type of instruction with minimal logic?**

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| R-Type | 0110011 | 1 | 0 | 0 | 0 | 0 | 0 |
| I-Type | 0010011 | 1 | 1 | 0 | 0 | 0 | 0 |
| LW | 0000011 | 1 | 1 | 0 | 1 | 0 | 1 |
| SW | 0100011 | 0 | 1 | 0 | 0 | 1 | 0 |
| BEQ | 1100011 | 0 | 0 | 1 | 0 | 0 | 0 |

# Basic RISC-V CPU Control Path

- **R-Type := intrs[4] & instrs[5]**
- **I-Type := ?**
- **LW := ?**
- **SW := ?**
- **BEQ := ?**

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| R-Type | 0110011 | 1 | 0 | 0 | 0 | 0 | 0 |
| I-Type | 0010011 | 1 | 1 | 0 | 0 | 0 | 0 |
| LW | 0000011 | 1 | 1 | 0 | 1 | 0 | 1 |
| SW | 0100011 | 0 | 1 | 0 | 0 | 1 | 0 |
| BEQ | 1100011 | 0 | 0 | 1 | 0 | 0 | 0 |

# Basic RISC-V CPU Control Path

- R-Type := intrs[4] & instrs[5]
- I-Type := intr[4] & ~instr[5]
- LW := ~instr[4] & ~instr[5]
- SW := ?
- BEQ := ?

| Instruction | Opcode | RegWrite | AluSrc | Branch | MemRead | MemWrite | MemtoReg |
|---|---|---|---|---|---|---|---|
| R-Type | 0110011 | 1 | 0 | 0 | 0 | 0 | 0 |
| I-Type | 0010011 | 1 | 1 | 0 | 0 | 0 | 0 |
| LW | 0000011 | 1 | 1 | 0 | 1 | 0 | 1 |
| SW | 0100011 | 0 | 1 | 0 | 0 | 1 | 0 |
| BEQ | 1100011 | 0 | 0 | 1 | 0 | 0 | 0 |

# Basic RISC-V CPU Control Path

**ALUOp =?**

**Only 4 operations:**

- AND (00)
- OR (01)
- ADD (10)
- SUB (11)

**BEQ := (rs1 – rs2)**

| Instruction | Opcode | Func3 | Func7 | ALU Fn | ALU Op |
|---|---|---|---|---|---|
| ADD | 0110011 | 000 | 0000000 | ADD | 10 |
| SUB | 0110011 | 000 | 0100000 | SUB | 11 |
| OR | 0110011 | 110 | 0000000 | OR | 01 |
| AND | 0110011 | 111 | 0000000 | AND | 00 |
| ADDI | 0010011 | 000 | xxxxxxx | ADD | 10 |
| ORI | 0010011 | 110 | xxxxxxx | OR | 01 |
| ANDI | 0010011 | 111 | xxxxxxx | AND | 00 |
| LW | 0000011 | 010 | xxxxxxx | ADD | 10 |
| SW | 0100011 | 010 | xxxxxxx | ADD | 10 |
| BEQ | 1100011 | 000 | xxxxxxx | SUB | 11 |

# Attendance

- Please Fill the form with today’s keyword to register your attendance.

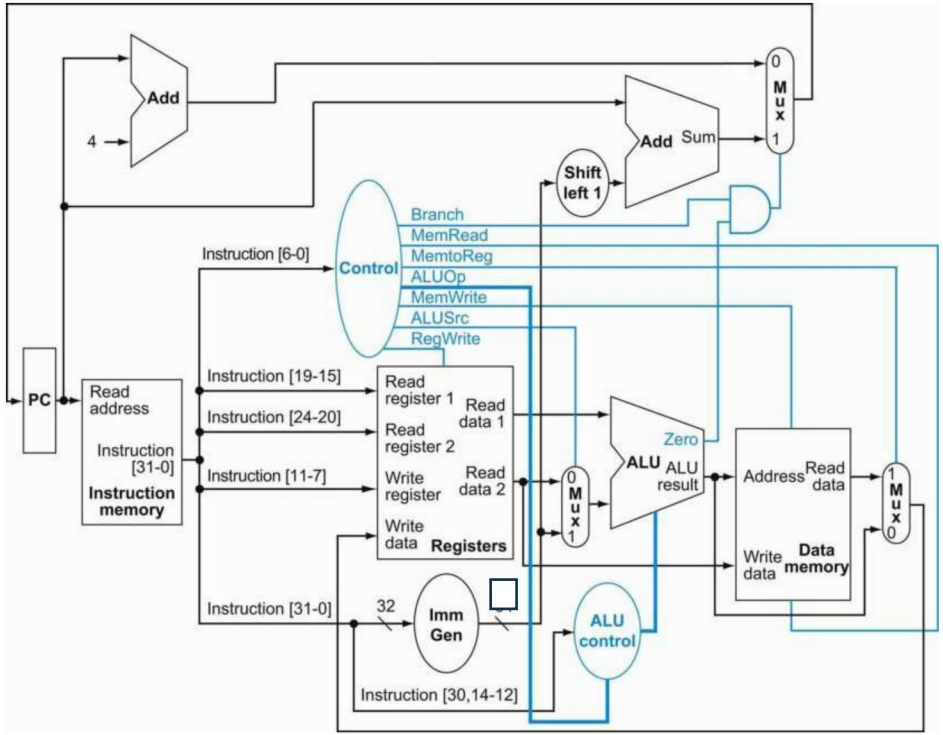
# Single-Cycle Processor

**Single-cycle Processor:**

- Cycle I
  - Instruction memory read
  - Register file access
  - Control Units produce output
  - ALU produces output
  - Data memory read
  - Next PC calculation

**Note:** Assume Instruction and data memory read is asynchronous (same cycle), however memory updates are only visible in next cycle.

Advantages?

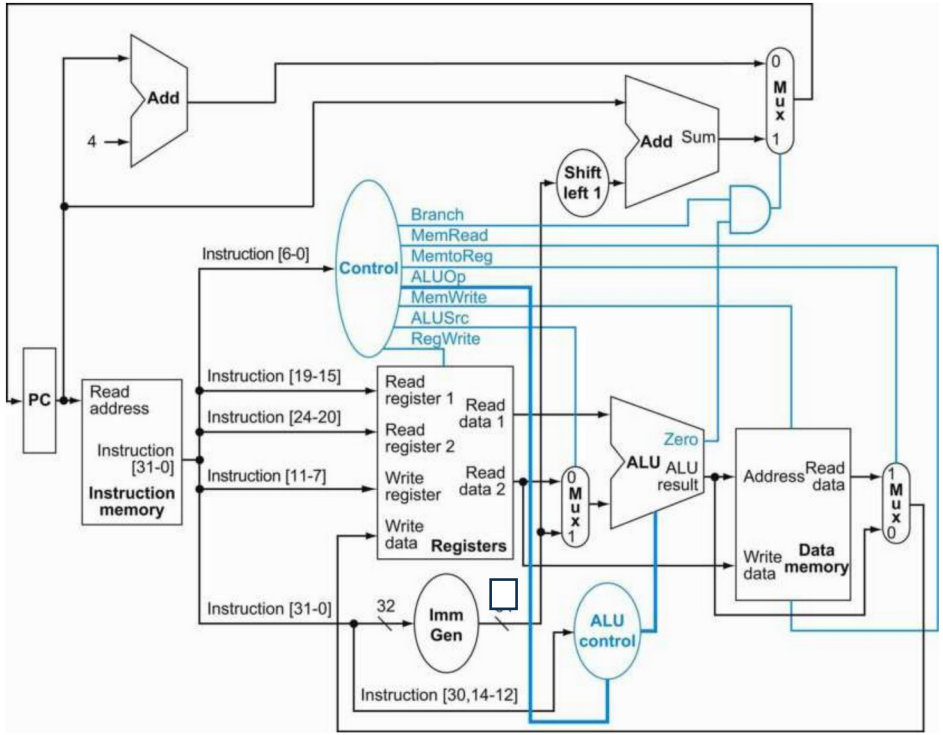


# Single-Cycle Processor

**Single-cycle Processor Advantages:**

- Simple to implement
- Efficient design (resource usage)

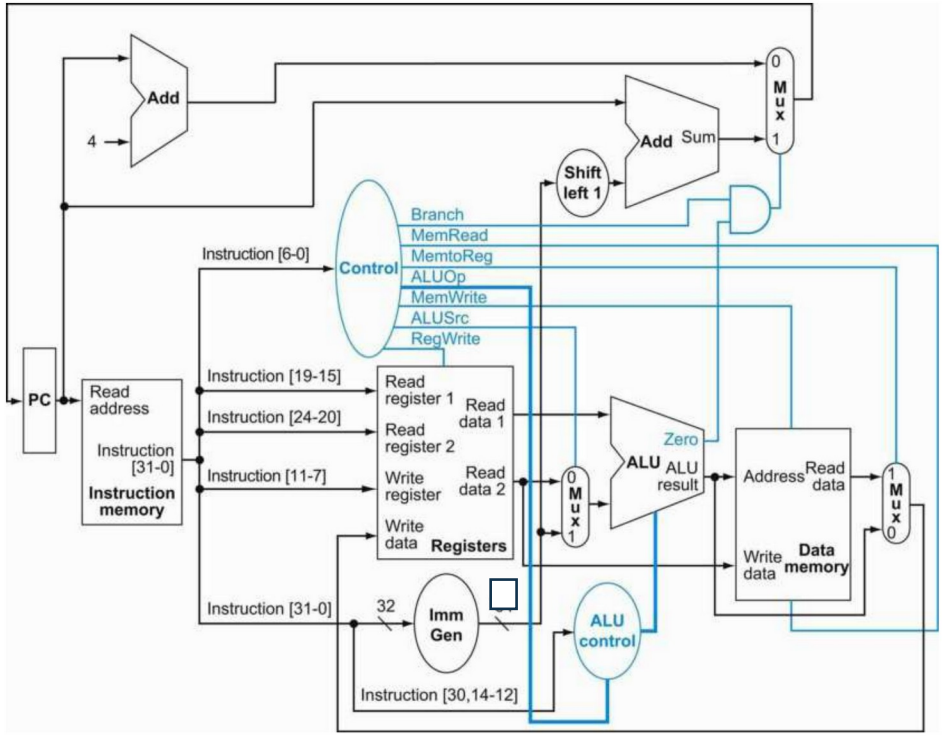
Limitations?



# Single-Cycle Processor

**Single-cycle Processor Limitations:**

- Weak performance
  - Long latency
- Critical path?

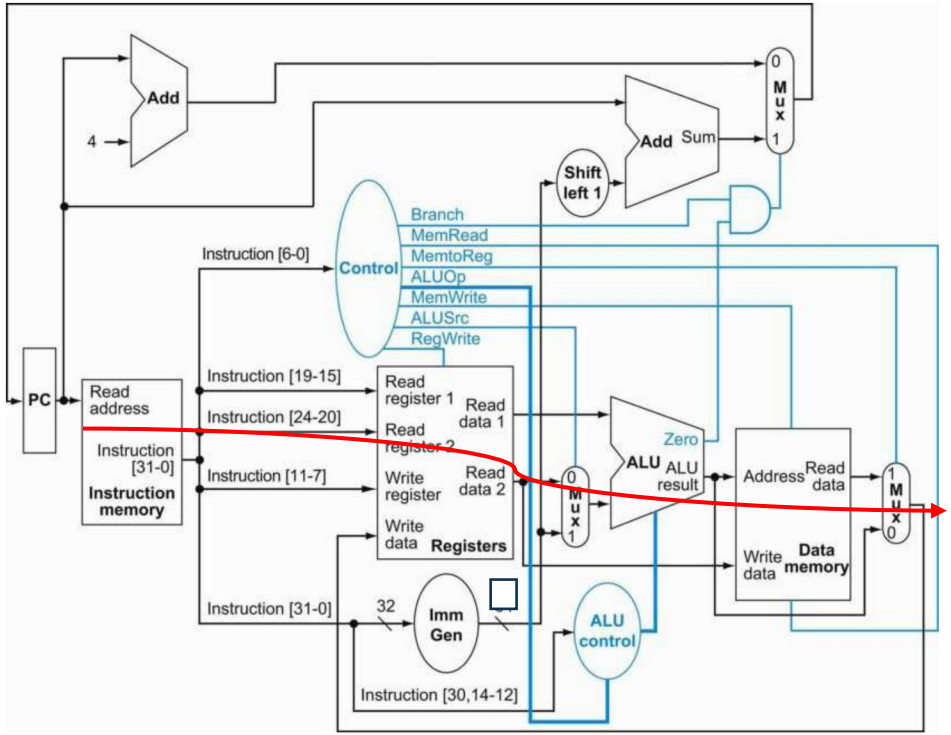


# Single-Cycle Processor

**Single-cycle Processor Latency:**

- Long propagation delay
  - Signal propagation from source to destination
  - **Source**: input or register
  - **Destination**: output or register
- Imbalanced datapath
- Which path is the longest?

Why it affects latency?



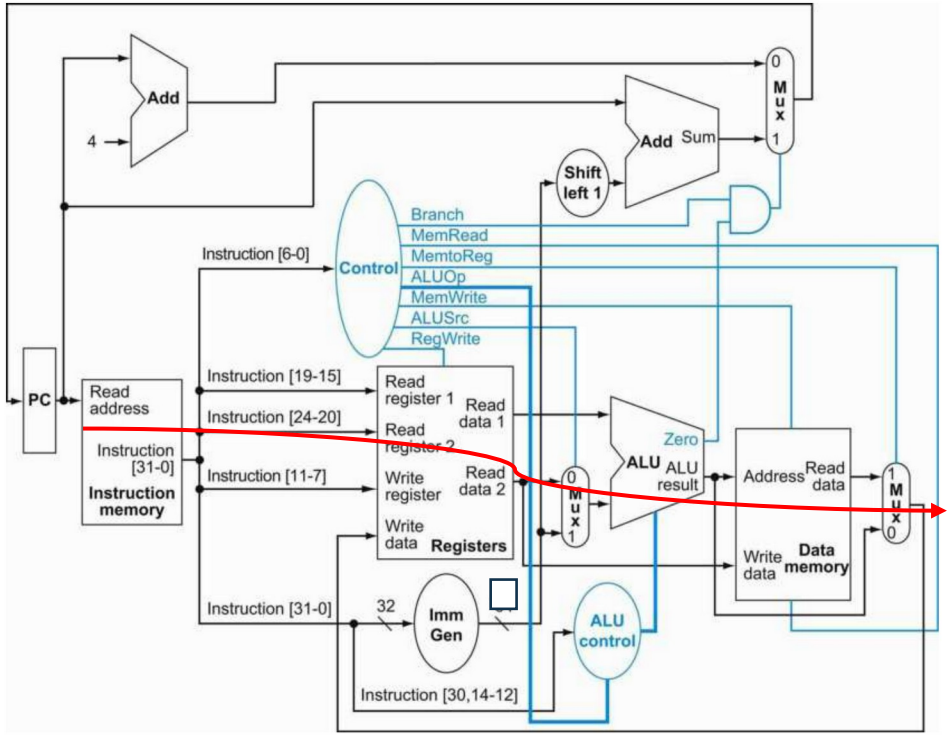
# Single-Cycle Processor

**Single-cycle Processor Latency:**

- How this affects CPU latency?
- Longest delay determines the clock speed
- CPU clock affects performance

CPU Time = InstCount * CPI * **CycleTime**

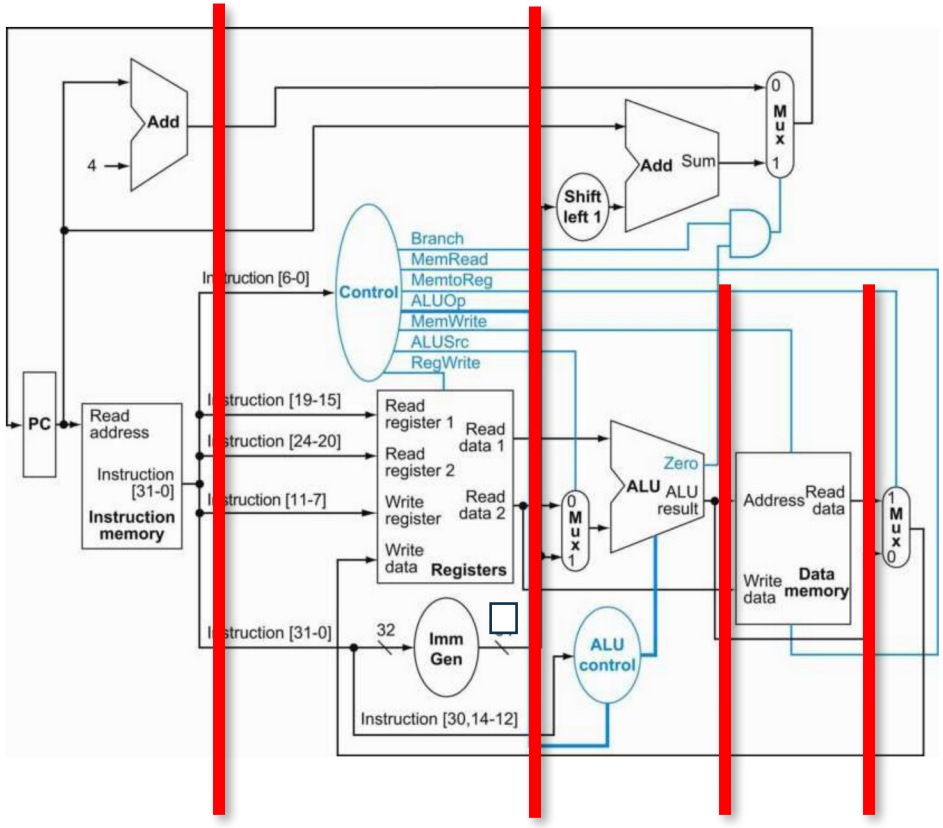
How to reduce propagation delay?



# Single-Cycle Processor

## Reducing Propagation Delay

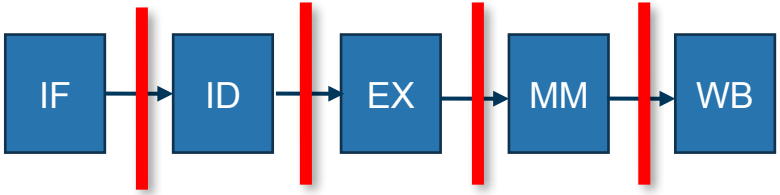
- Split the datapath into smaller chunks
- Insert registers in between
- Update control logic
  - Control registers inputs/outputs

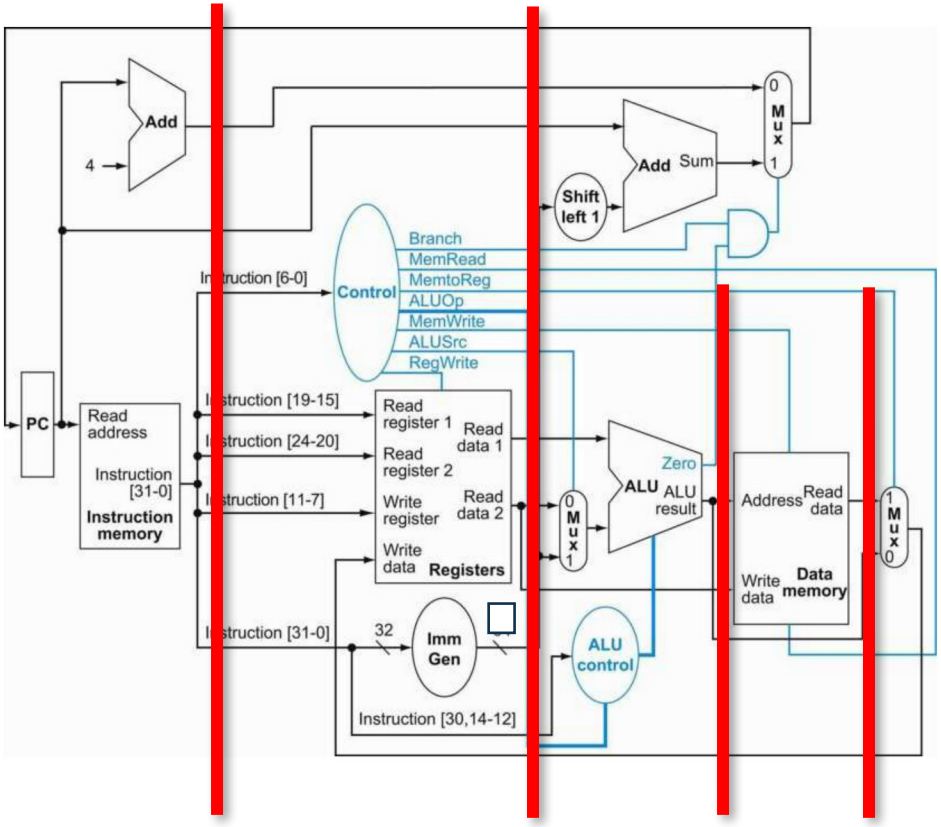


# Multi-Cycle Processor

## Change to multi-cycle architecture

- 5 stages
  - IF: Instruction Fetch
  - ID: Register access
  - EX: ALU operation
  - MEM: Memory access
  - WB: Write back



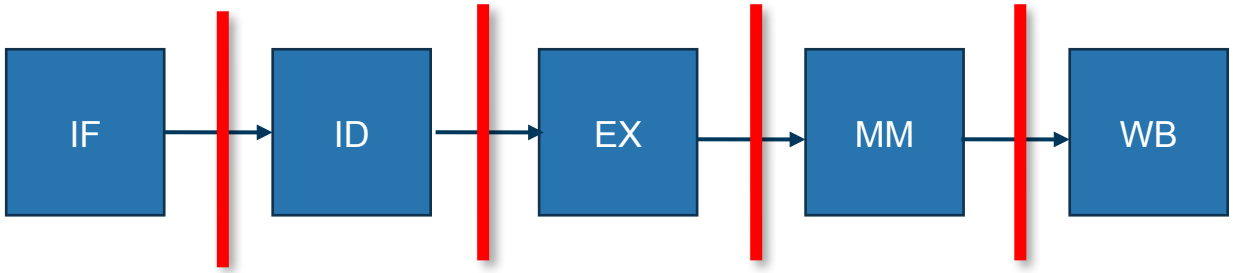



# Multi-Cycle Processor

## Improved Propagation Delay

- Longest propagation delay?
  - max(IF, ID, EX, MEM, WB)
  - Should be smaller

How is the overall performance impacted?

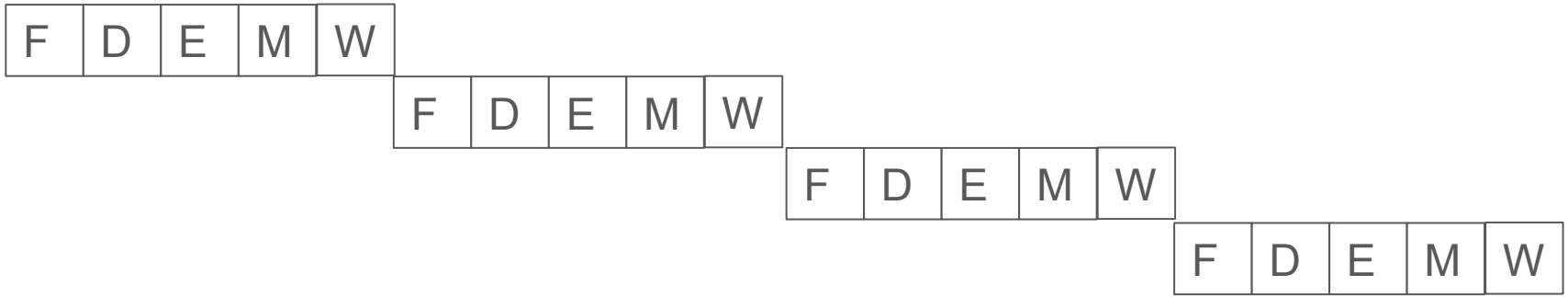


# Multi-Cycle Processor

## Multi-cycle Processor Performance

- **Pros**: Cycle Time is reduced
- **Cons**: CPI is increased: a single instruction takes multiple cycles to complete

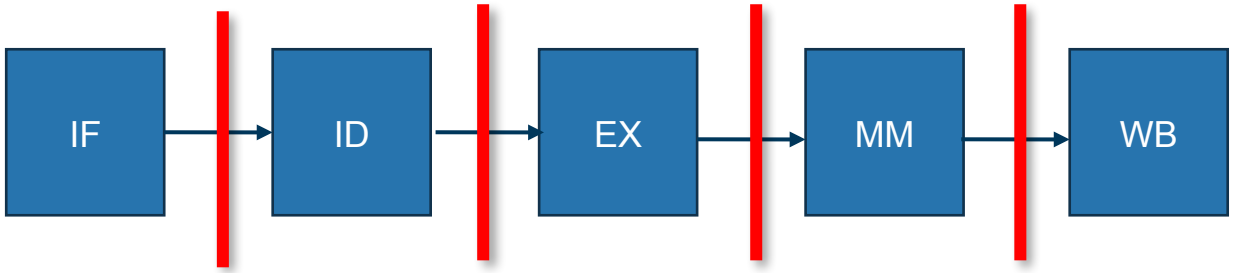
CPU Time = InstCount * **CPI** * **CycleTime**



# Pipelined Processor

## Observations?

- **H/W resources are under utilized**
- When an instruction is stage WB
- IF, ID, E, and MEM stages are idle (do nothing)

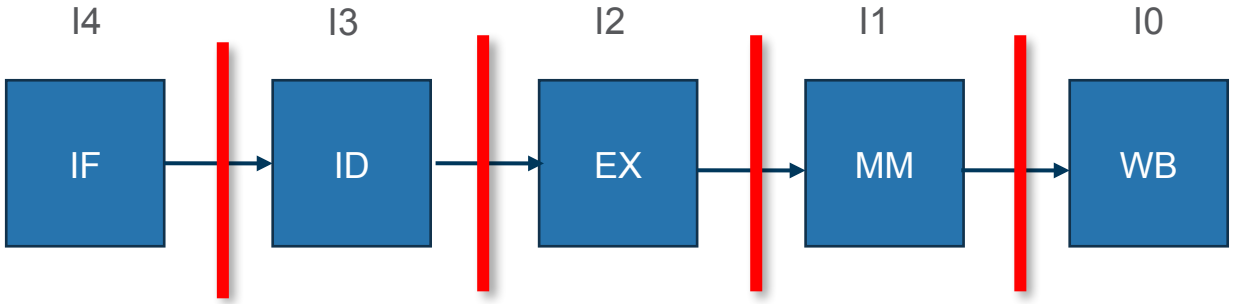


# Pipelined Processor

## Solution

- **Fetch a new instruction each cycle**

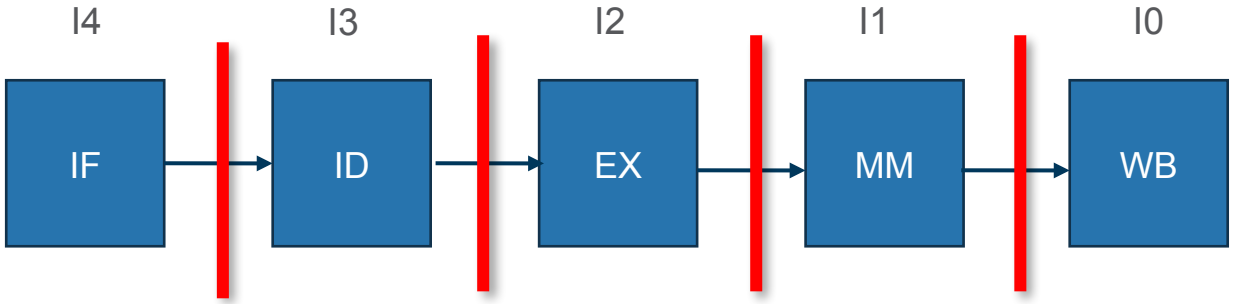
e.g: Executing instructions sequence: I0, I1, I2, I3, I4



# Pipelined Processor

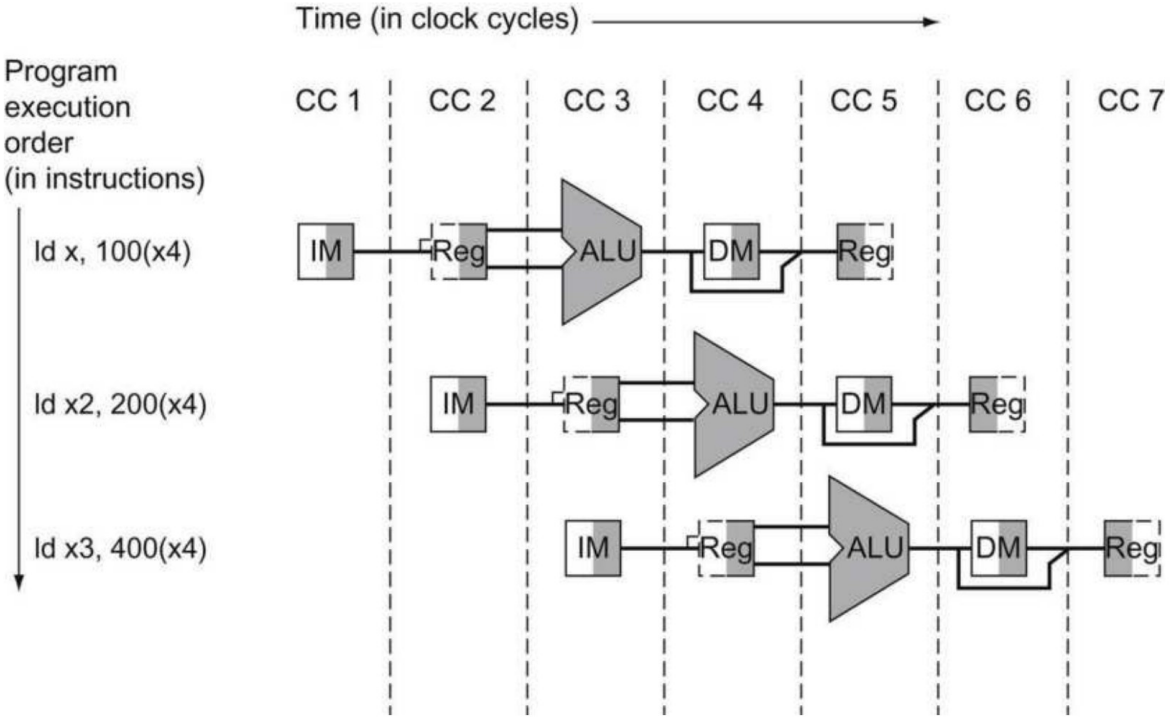
**How it work?**

- The PC should be incremented each cycle
- Online instructions are kept in pipeline registers



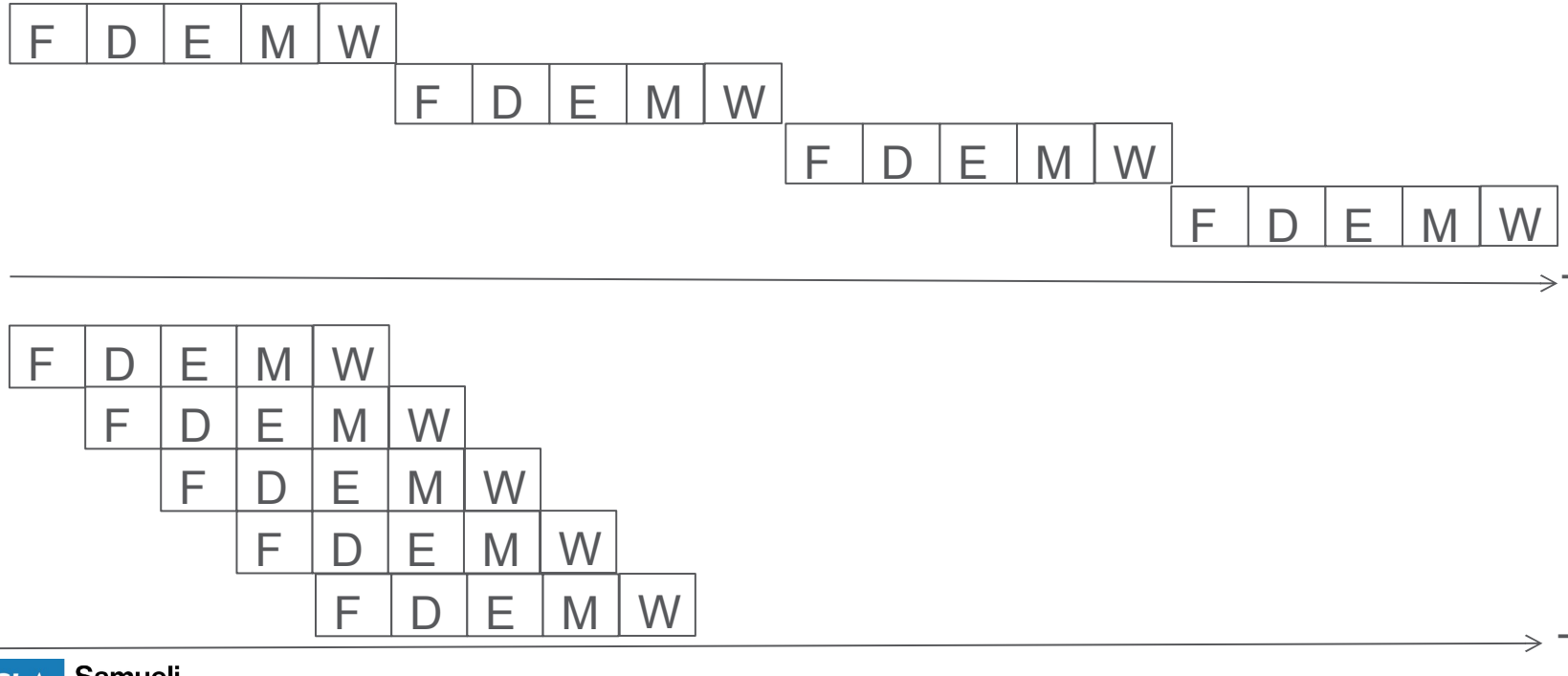
# Pipelined Processor

## Resource utilization



# Pipelined Processor

## Multi-cycle vs Pipelined performance



# Pipelined Processor

**Multi-cycle vs Pipelined performance**

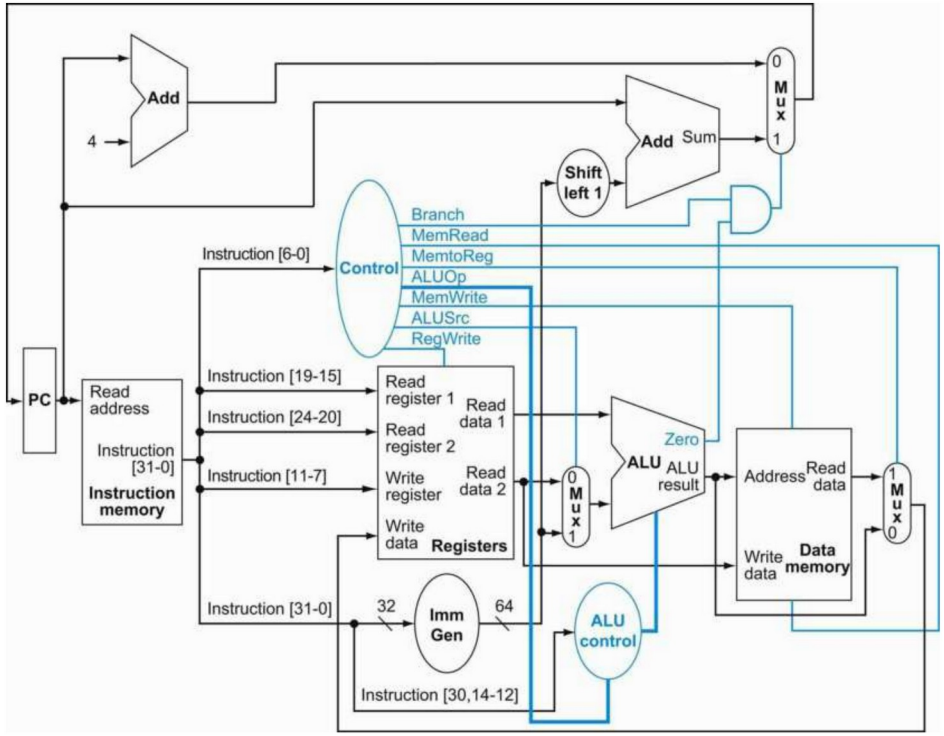
- The CPI is reduced
- Ideal pipelined CPI is 1
  - One instruction complete each cycle

CPU Time = InstCount * **CPI** * CycleTime

# Exercise: Single-Cycle vs Pipeline

- Calculate cycle time (CT)

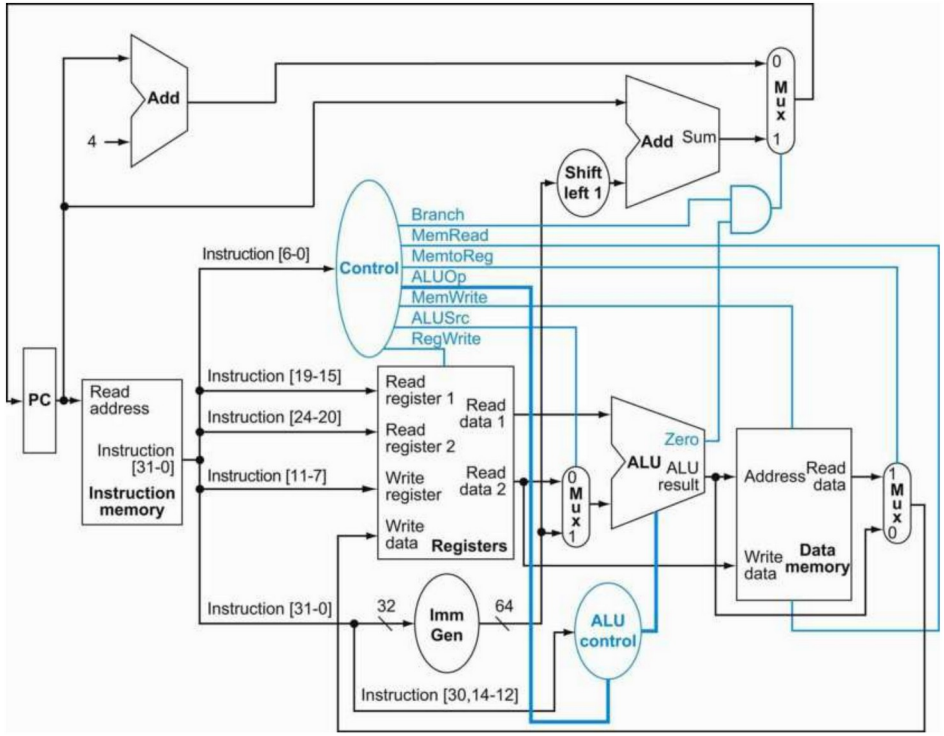
| Unit | Latency (ps) |
|---|---|
| Instruction Fetch | 400 |
| Register Access | 300 |
| ALU | 200 |
| Memory Access | 400 |



# Exercise: Single-Cycle vs Pipeline

- Single-Cycle Datapath

| Instr | Data Path |
|---|---|
| R/I-type | IF+RR+ALU+RW |
| LW | IF+RR+ALU+MR+RW |
| SW | IF+RR+ALU+MW |
| BEQ | IF+RR+ALU |



# Exercise: Single-Cycle vs Pipeline

- Single-Cycle Datapath

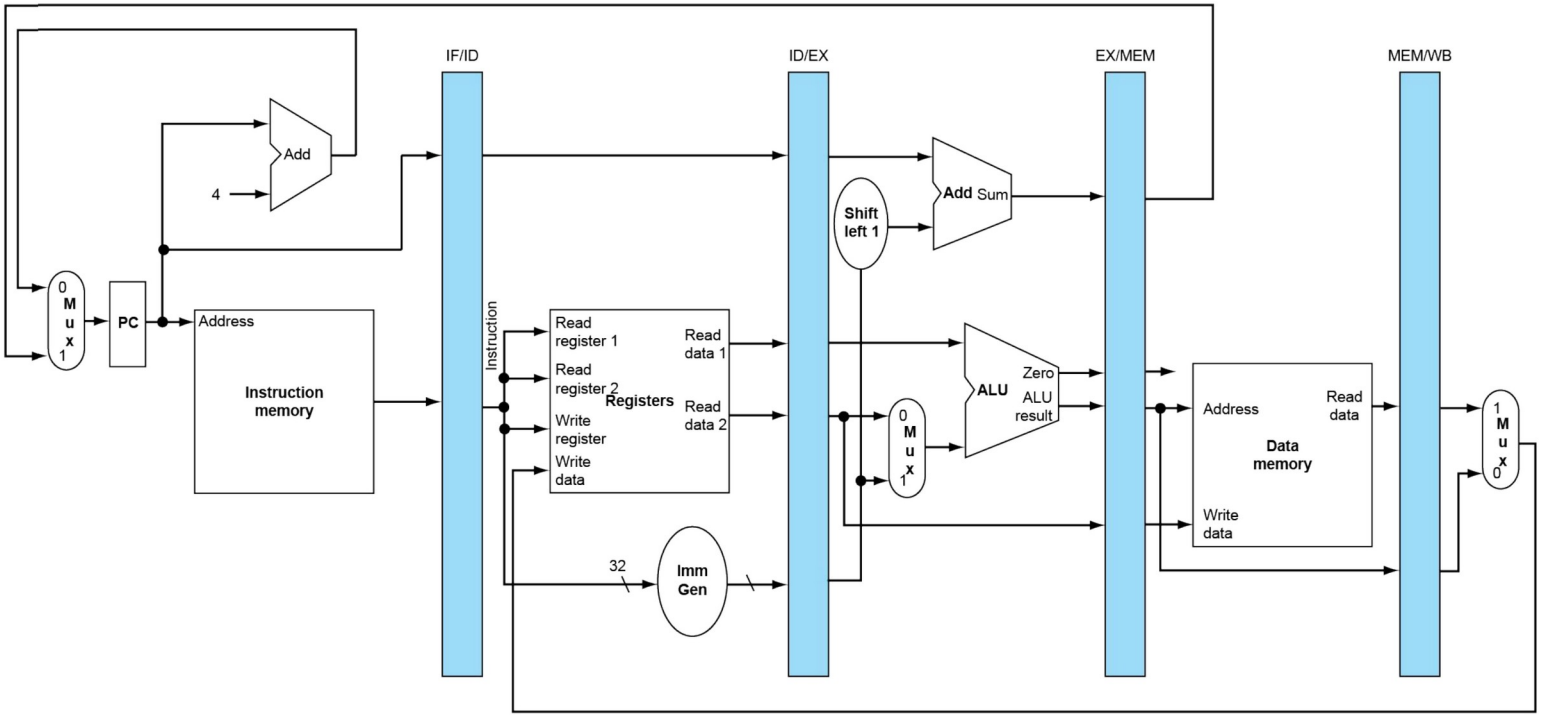
| Instr | Data Path | Cycle Time | Total Time |
|---|---|---|---|
| R/I-type | IF+RR+ALU+RW | 400+300+200+300 | 1200 ps |
| LW | IF+RR+ALU+MR+RW | 400+300+200+400+300 | 1600 ps |
| SW | IF+RR+ALU+MW | 400+300+200+400 | 1300 ps |
| BEQ | IF+RR+ALU | 400+300+200 | 900 ps |

# Exercise: Single-Cycle vs Pipeline

- Pipeline Datapath

| Instr | Data Path | Cycle Time | Total Time |
|---|---|---|---|
| R/I-type | IF\|RR\|ALU\|RW | Max(400,300,200,300) | 400 ps |
| LW | IF\|RR\|ALU\|MR\|RW | Max(400,300,200,400,300) | 400 ps |
| SW | IF\|RR\|ALU\|MW | Max(400,300,200,400) | 400 ps |
| BEQ | IF\|RR\|ALU | Max(400,300,200) | 400 ps |

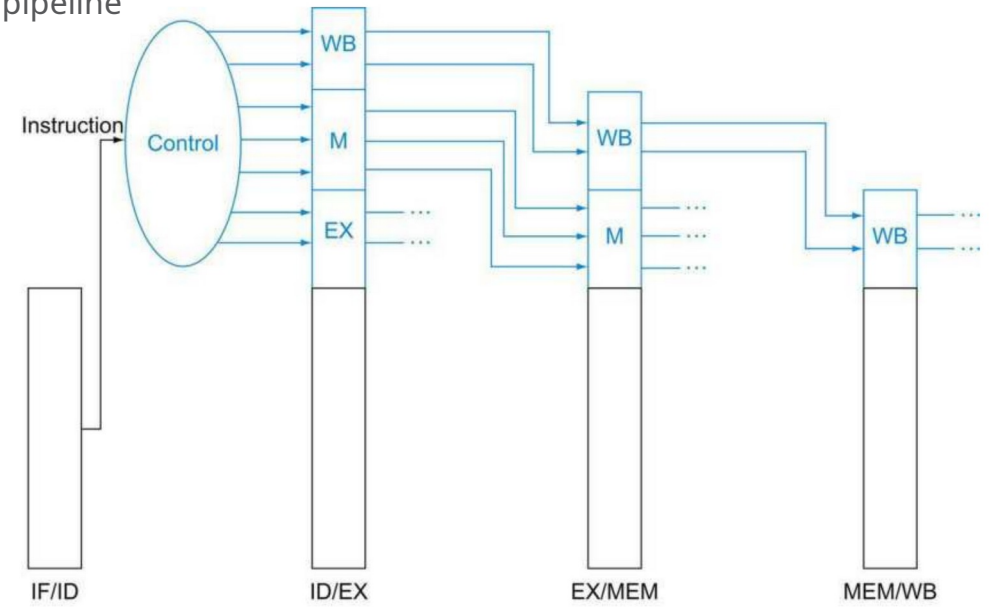
# 5-Stage Pipeline RISC-V Processor



# 5-Stage Pipeline RISC-V Processor

## Pipelined control logic

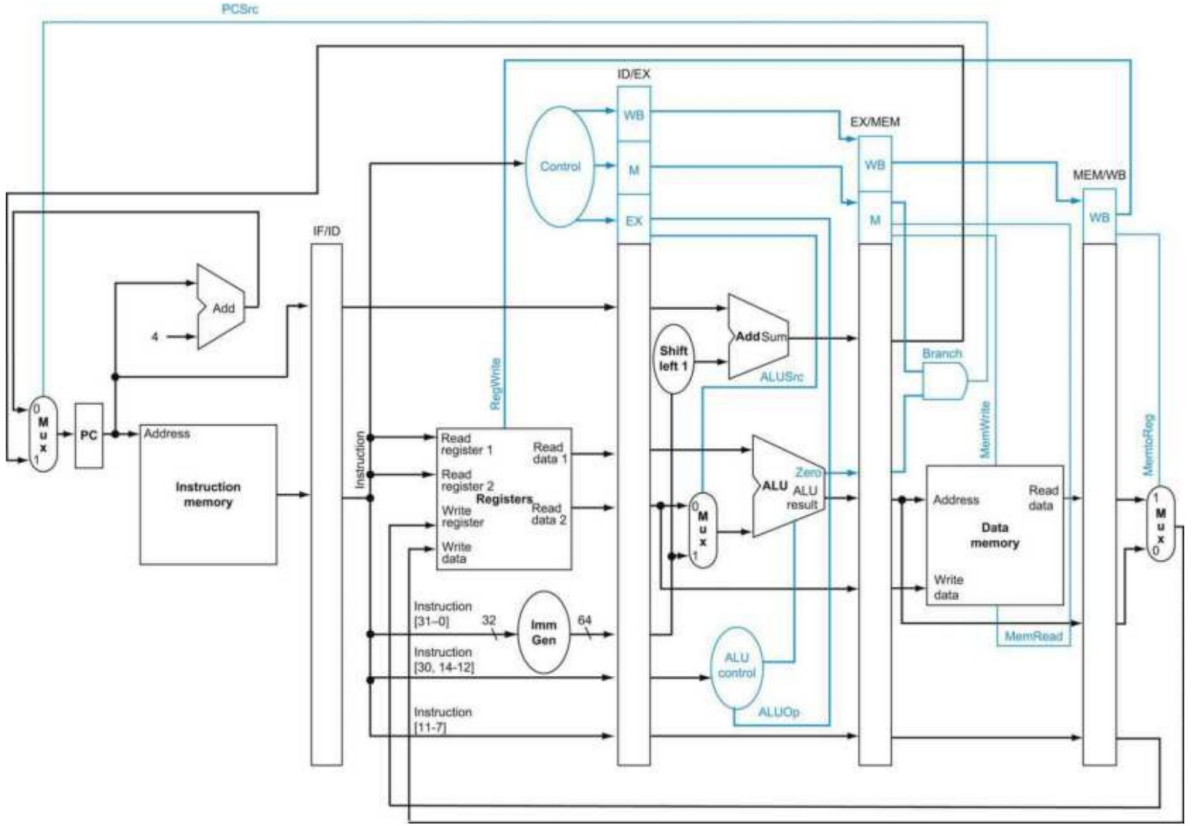
- Propagate the flags along the pipeline



# Basic RISC-V CPU Data & Control Path

**Data path**

**Control Path**



# Q&A