Pipeline Processor Architecture

by: Tassadaq Hussain
Director Centre for AI and BigData
Professor Department of Electrical Engineering
Namal University Mianwali

Collaborations:
Barcelona Supercomputing Center, Spain
European Network on High Performance and Embedded Architecture and Compilation
Pakistan Supercomputing Center

Topics

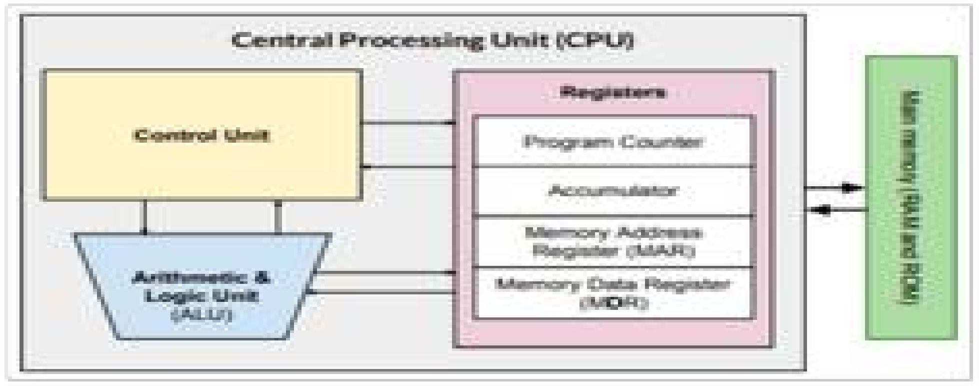
1. **Basic Processor Architecture**
2. Problems with Single Cycle Processor Architecture
3. Processor Architectures and Types

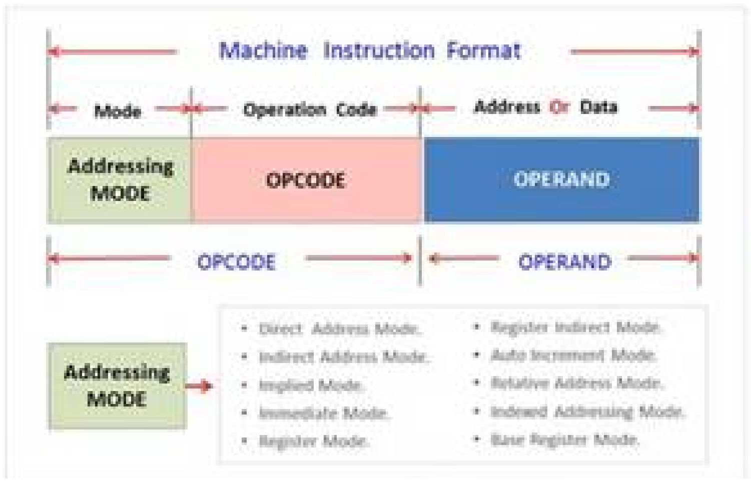
Basic Processor Architecture

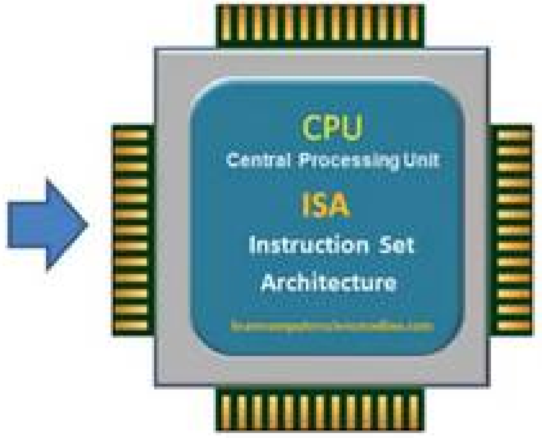
Processor Architecture refers to the design and organization of a processor's central processing unit (CPU).

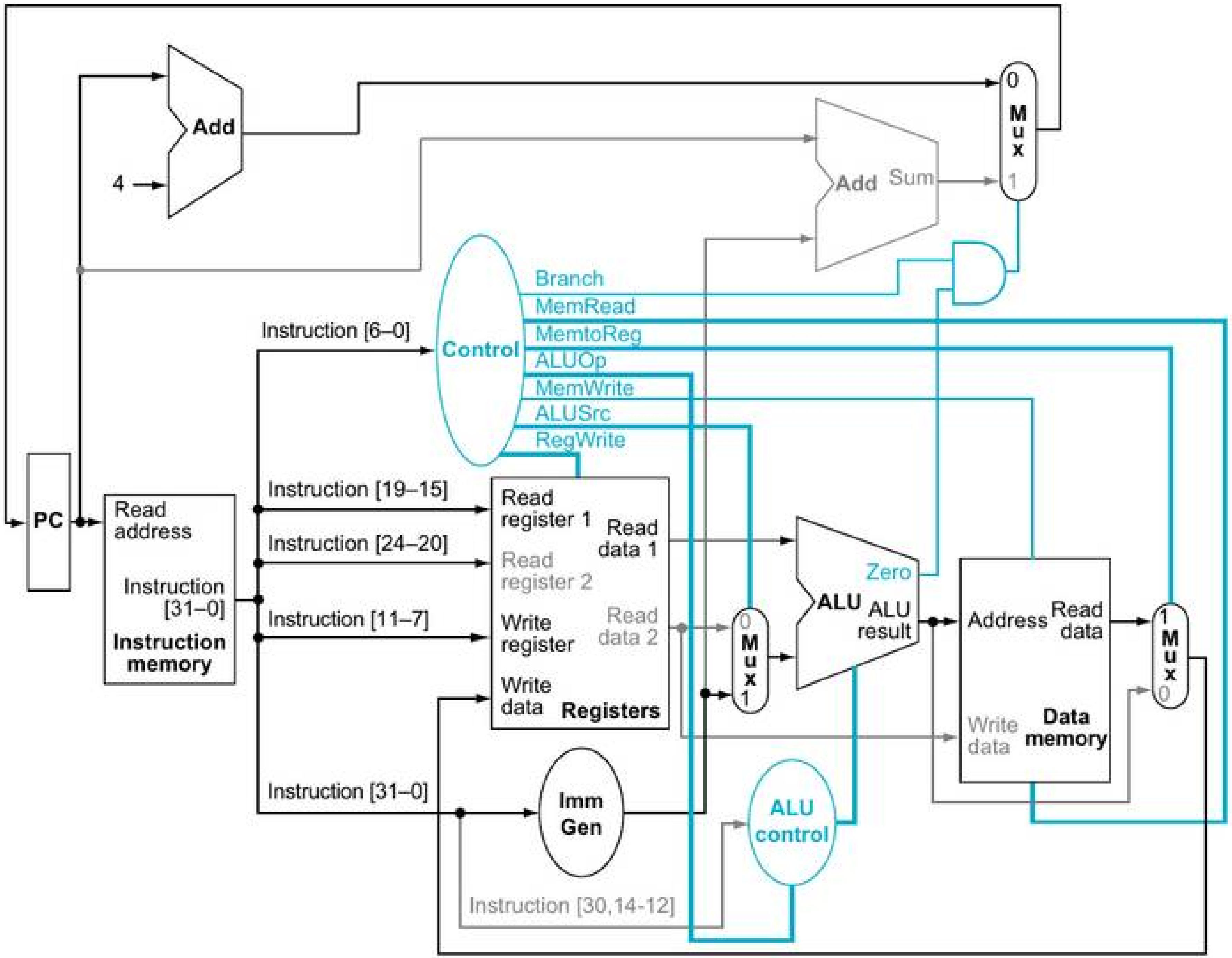
Components of Processor:

- **Arithmetic and Logic Unit:** Performs mathematical calculations.
- **Control Unit:** Control the overall processing of the processor.
- **Decoders Unit:** Convert coded instructions into signals that can control other components.
- **Registers:** Hold data, instructions, and addresses temporarily during processing.
- **Buses:** Electrical pathways that transmit data and signals between components. Types include the data bus, address bus, and control bus.


Central Processing Unit (CPU)
Control Unit
Registers
Program Counter
Accumulator
Memory Address Register (MAR)
Memory Data Register (MDR)
Arithmetic & Logic Unit (ALU)
Main memory (RAM and ROM)






Add
4
Add
Sum
0
Mux
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6–0]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [19–15]
Instruction [24–20]
Instruction [11–7]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0
Mux
1
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
1
Mux
0
Instruction [31–0]
Imm Gen
ALU control
Instruction [30,14-12]

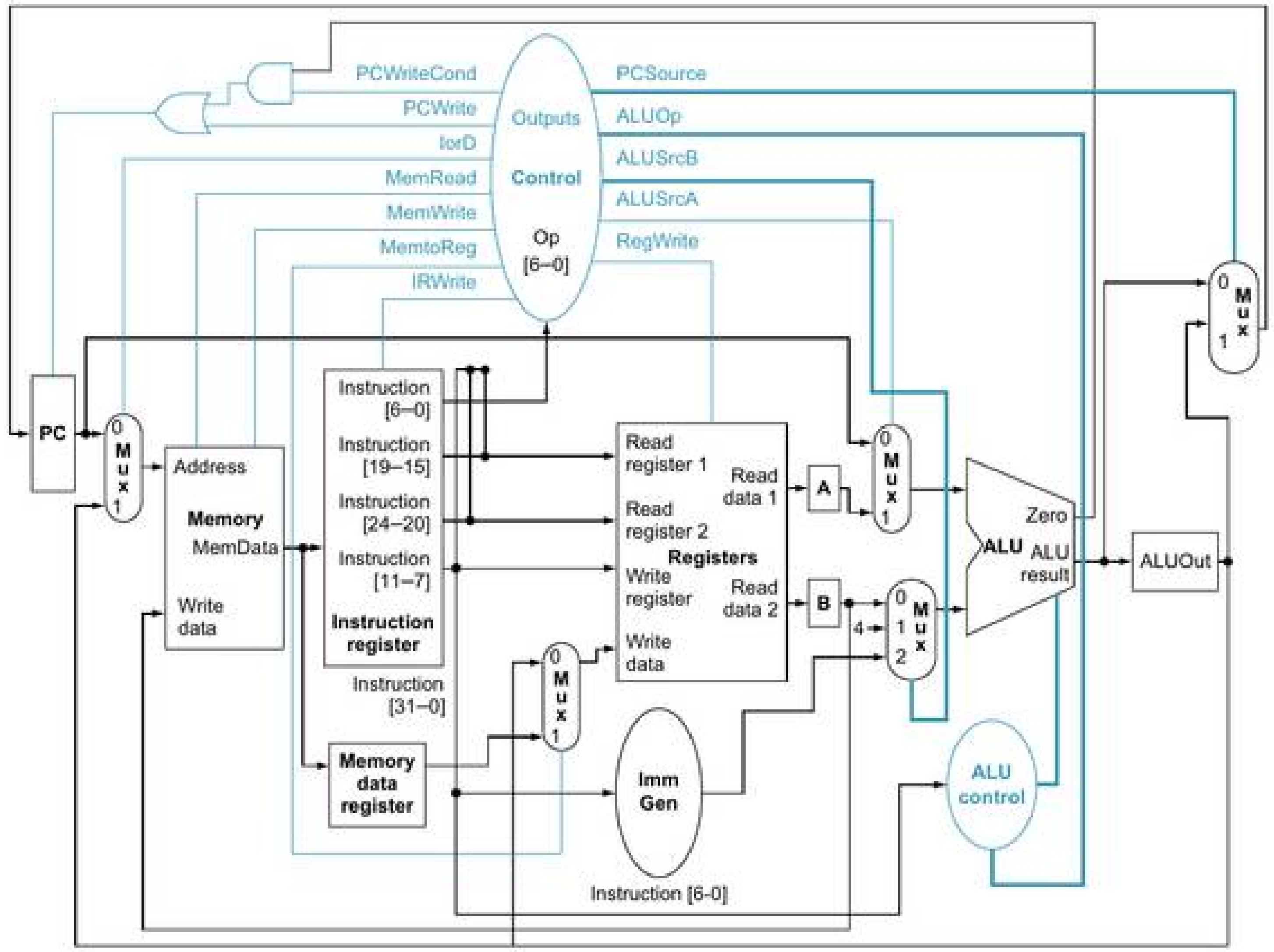
Topics

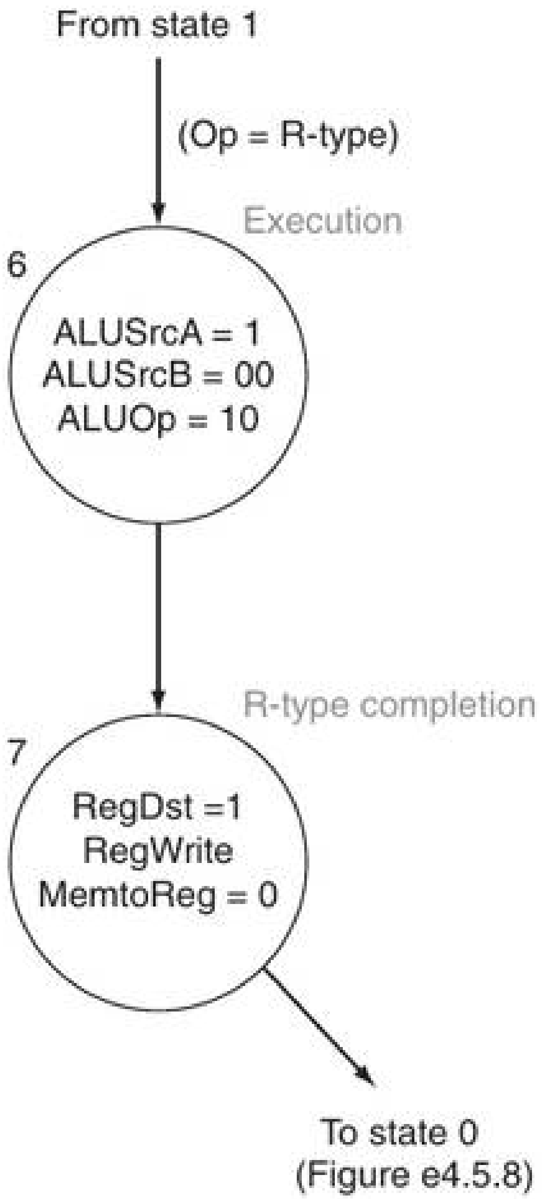
1. Basic Processor Architecture
2. **Problems with Single Cycle Processor Architecture**
3. Programming

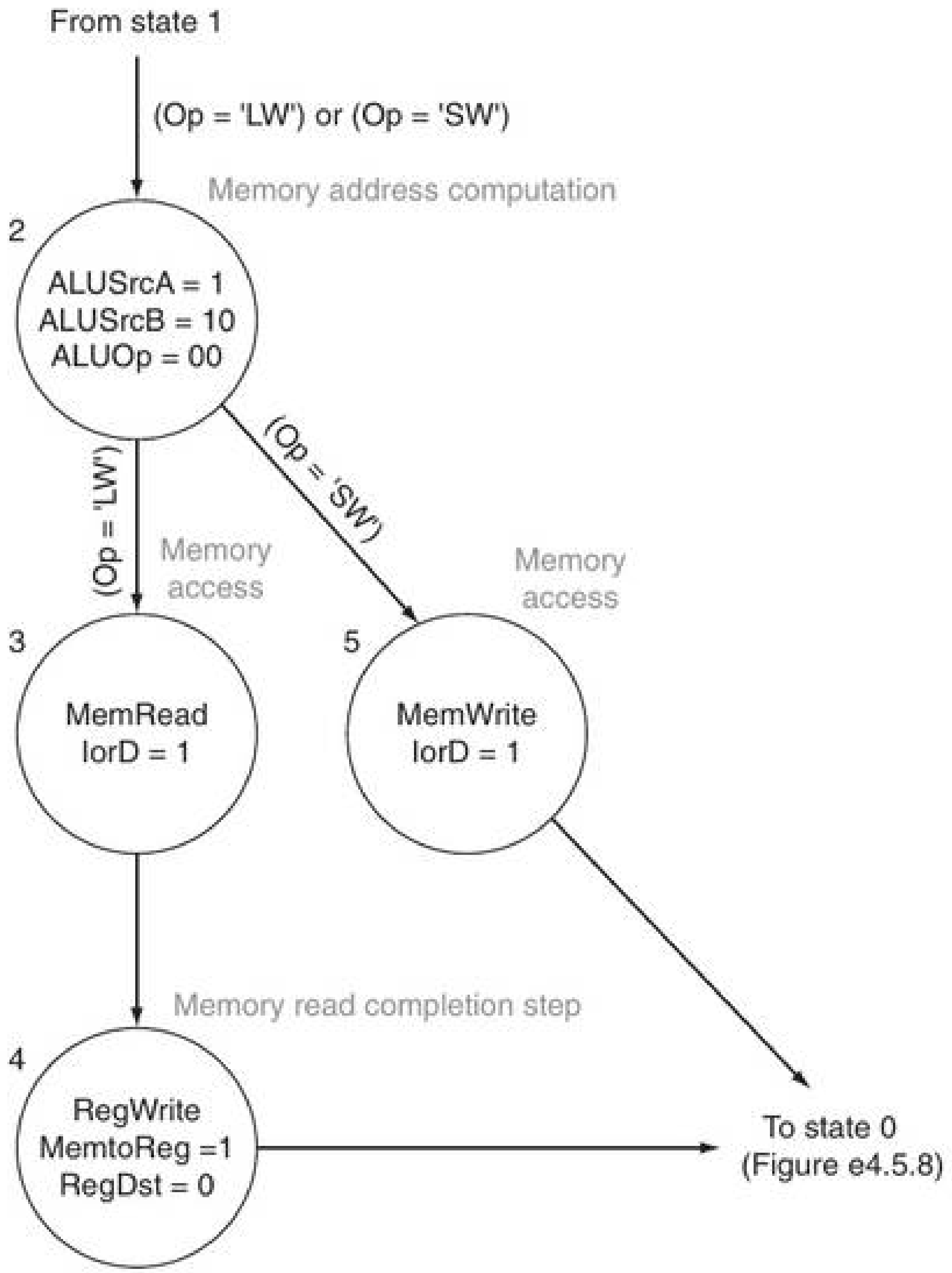
Drawback of Single Cycle Processor

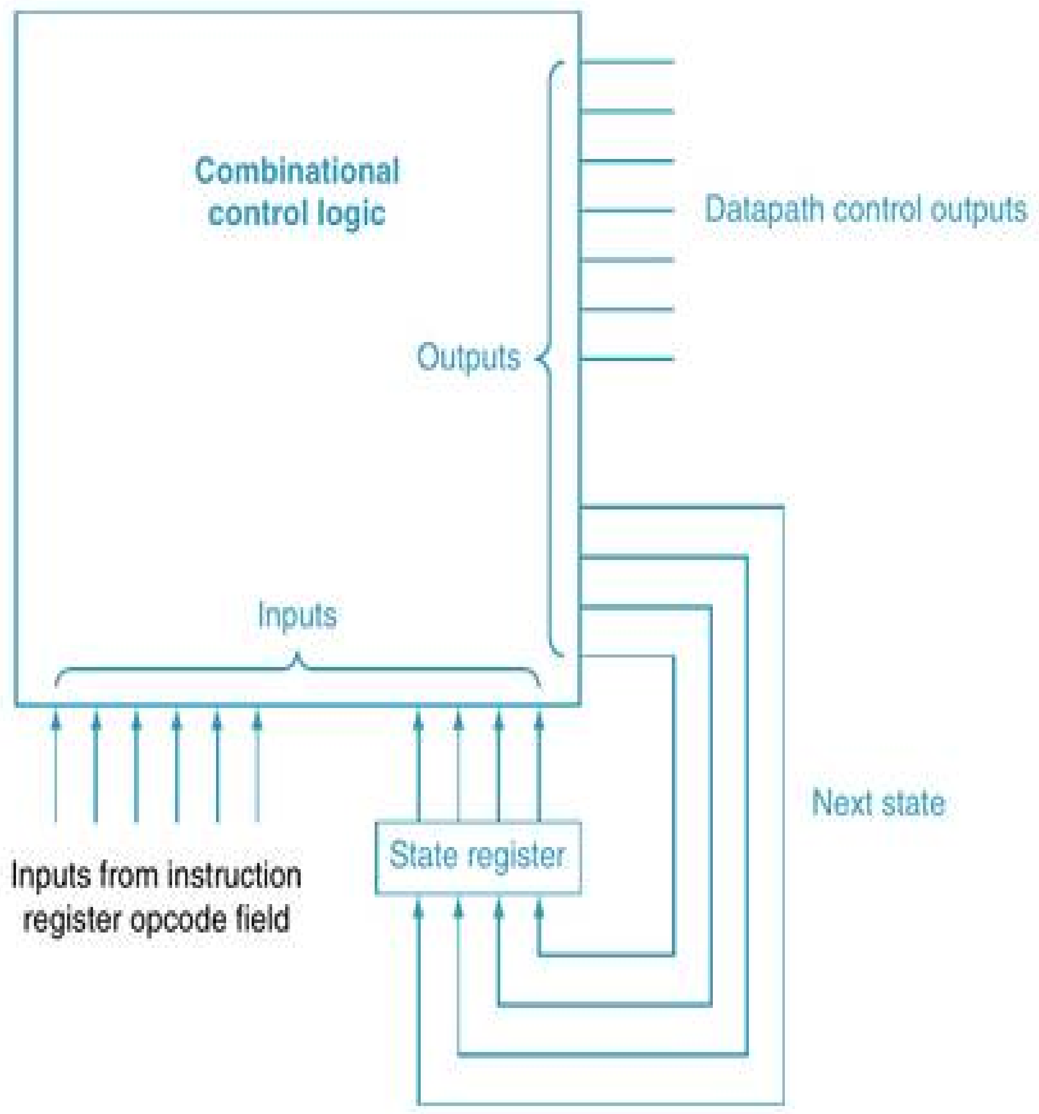
- Clock Cycle
- Fixed Clock Cycle
-

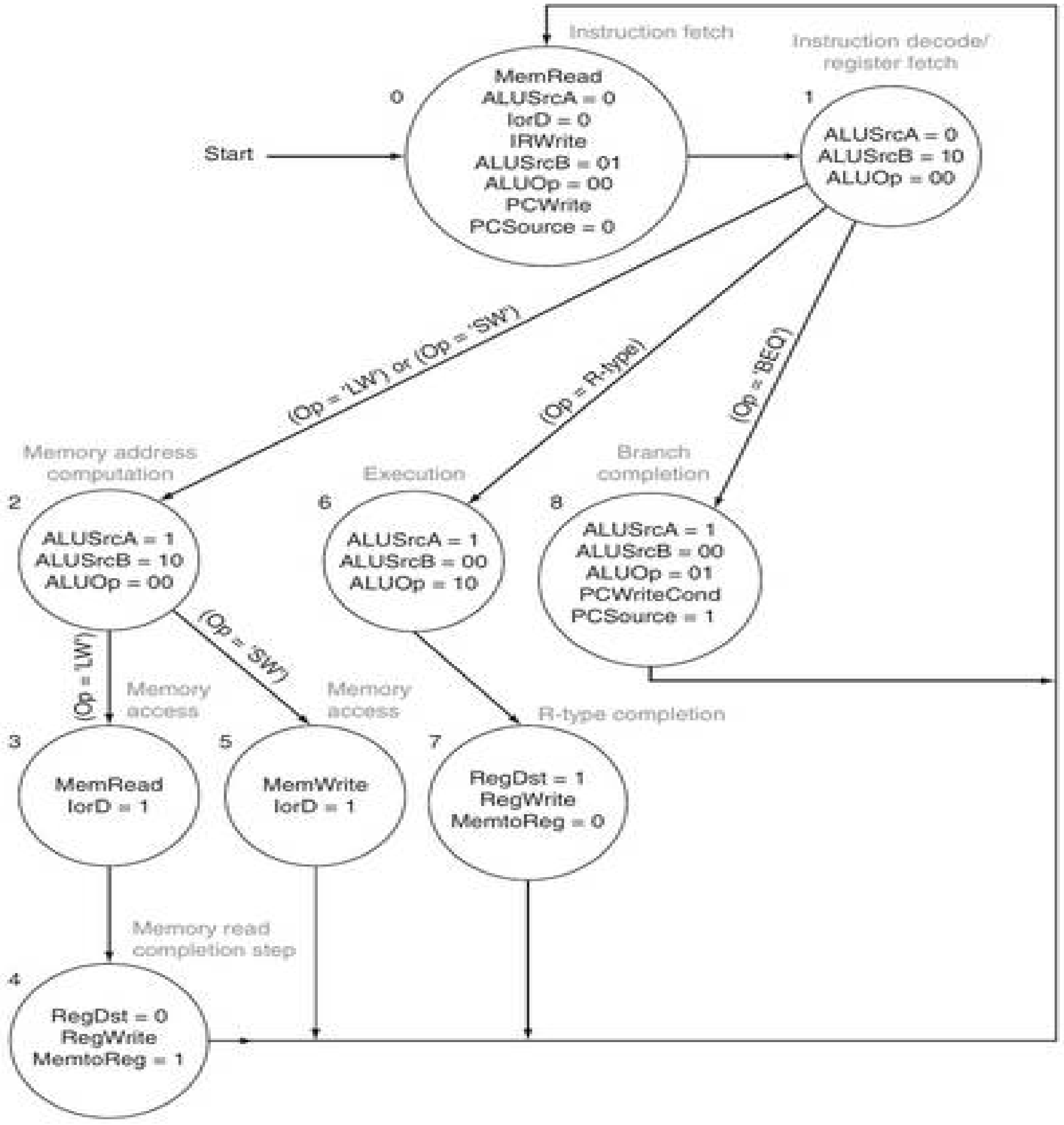
Multi-Cycle Implementation



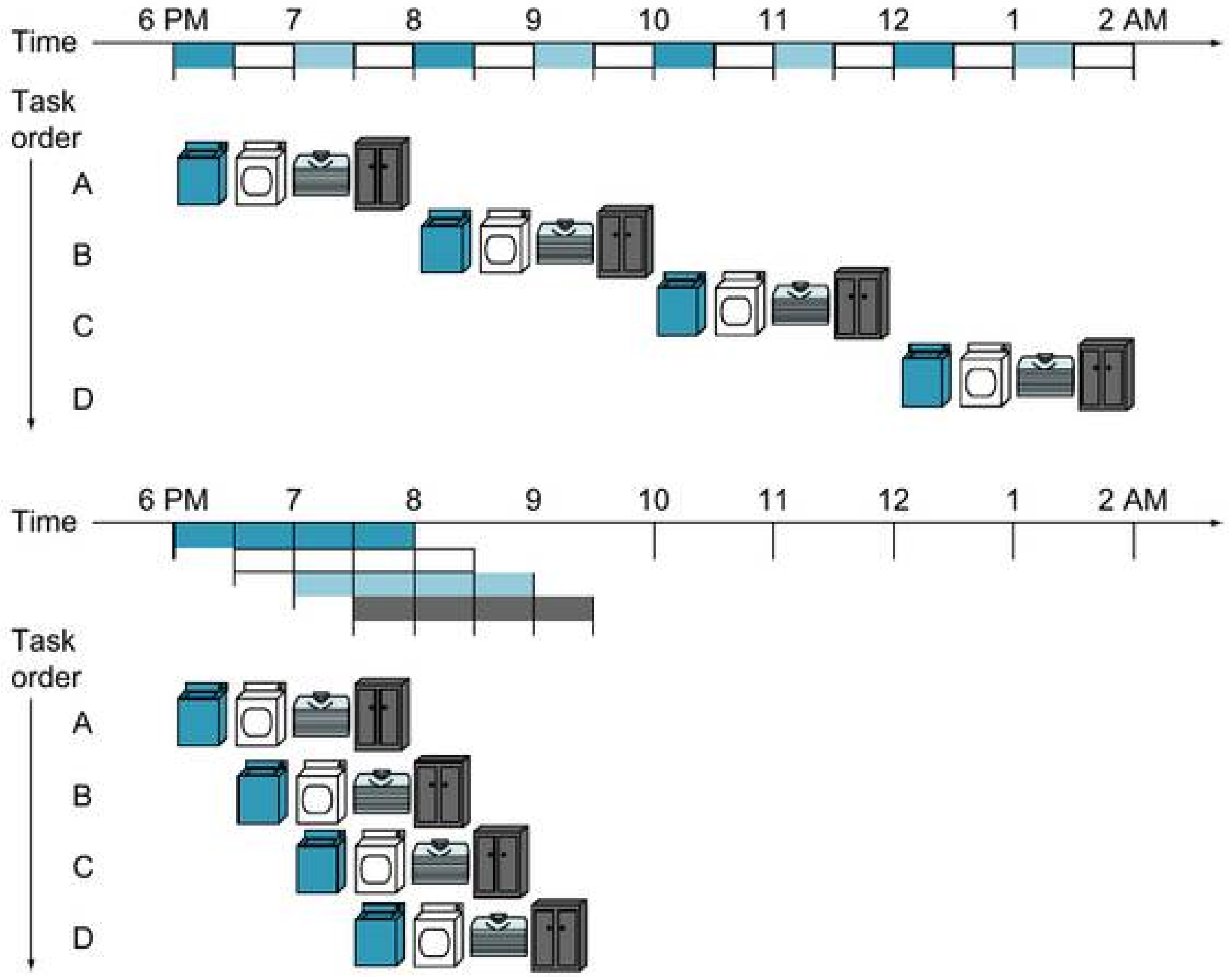

From state 1
(Op = R-type)
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
R-type completion
7
RegDst =1
RegWrite
MemtoReg = 0
To state 0
(Figure e4.5.8)


From state 1
(Op = 'LW') or (Op = 'SW')
Memory address computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
(Op = 'LW')
(Op = 'SW')
Memory access
Memory access
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
Memory read completion step
4
RegWrite
MemtoReg =1
RegDst = 0
To state 0
(Figure e4.5.8)

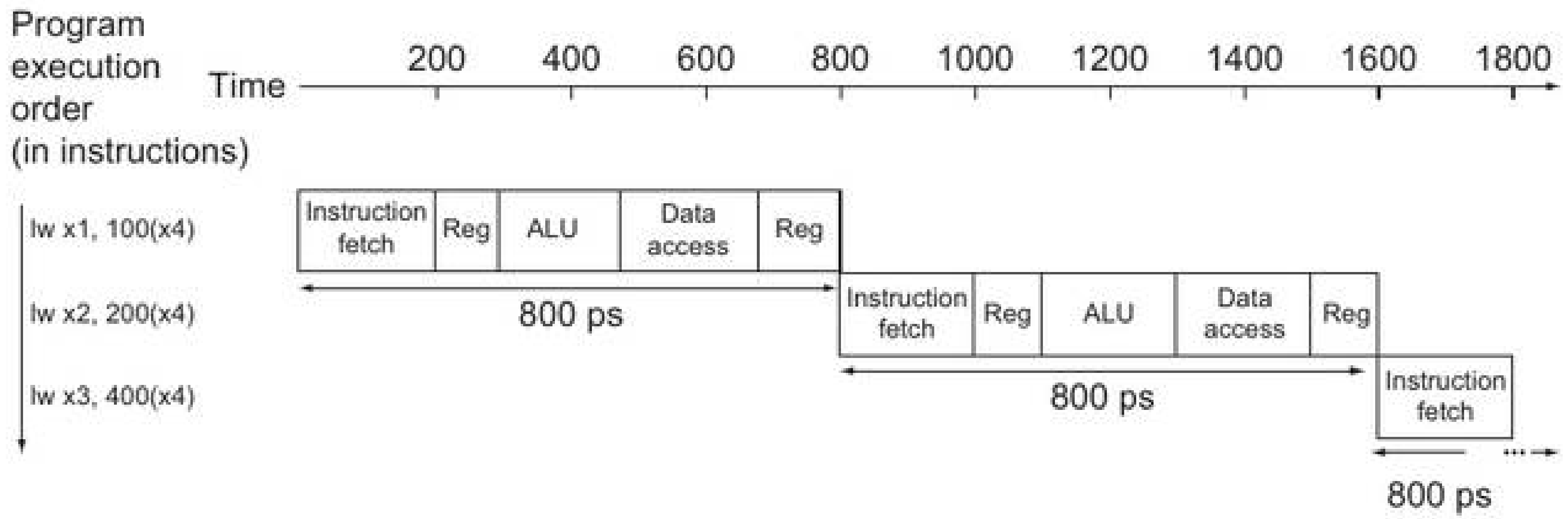

Combinational control logic
Datapath control outputs
Outputs
Inputs
Next state
State register
Inputs from instruction register opcode field

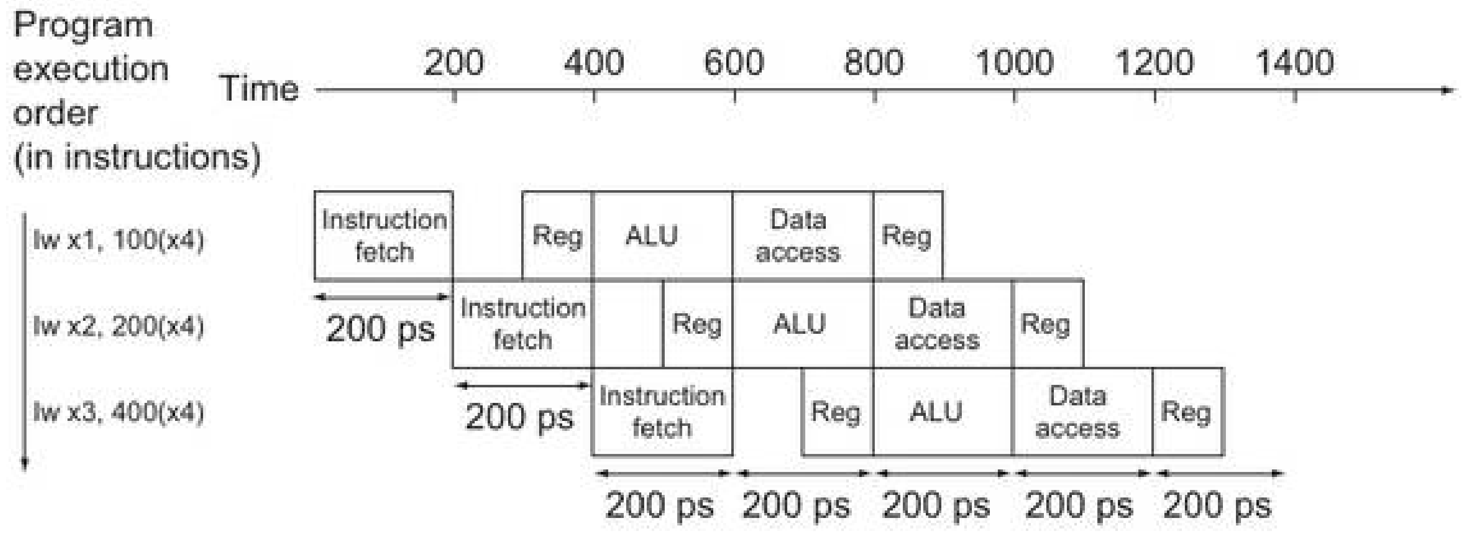

Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 0
Start
Instruction decode/ register fetch
1
ALUSrcA = 0
ALUSrcB = 10
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
Memory address computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 1
(Op = 'LW')
(Op = 'SW')
Memory access
3
MemRead
IorD = 1
Memory access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read completion step
4
RegDst = 0
RegWrite
MemtoReg = 1

Pipelining


Time
6 PM
7
8
9
10
11
12
1
2 AM
Task order
A
B
C
D
Time
6 PM
7
8
9
10
11
12
1
2 AM
Task order
A
B
C
D

Pipeline Instruction





Pipeline Hazards

- **Data Hazards**
  Instruction dependencies on data not yet available.
- **Control Hazards**
  Pipeline's uncertainty about branch outcomes leading to wrong instruction fetching.
- **Structural Hazard**
  Resource conflict when multiple instructions need the same hardware.

Data Hazard

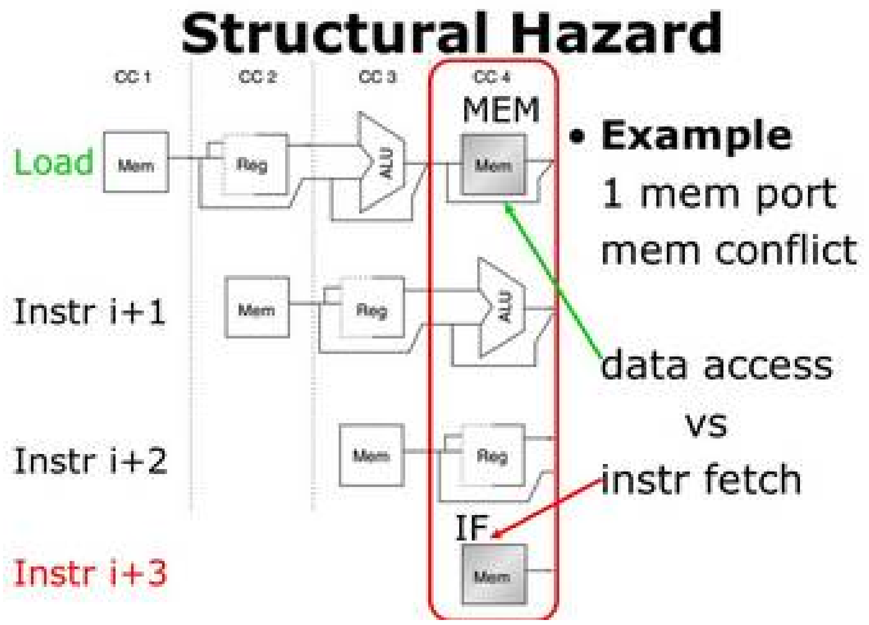
- A data hazard occurs when an instruction depends on the result of a previous instruction that has not yet completed in the pipeline. This means the required data is not yet available for the instruction to proceed.
- Types:
- Read after Write (RAW): The most common type, where an instruction needs to read a value that a previous instruction is still writing.
- Write after Read (WAR): An instruction writes a value before an earlier instruction reads it, potentially causing the wrong value to be read.
- Write after Write (WAW): Two instructions write to the same register in overlapping pipeline stages, and the wrong order could overwrite the result.
-
```
   add x5, x1, x2     # Instruction 1: x5 = x1 + x2
   sub x6, x5, x3     # Instruction 2: x6 = x5 - x3 (data hazard here)
   mul x7, x6, x4     # Instruction 3: x7 = x6 * x4
```
- **Solution: Data hazards can be mitigated by techniques like data forwarding (bypassing), where the result is passed directly to a dependent instruction, or by stalling until the necessary data is available.**

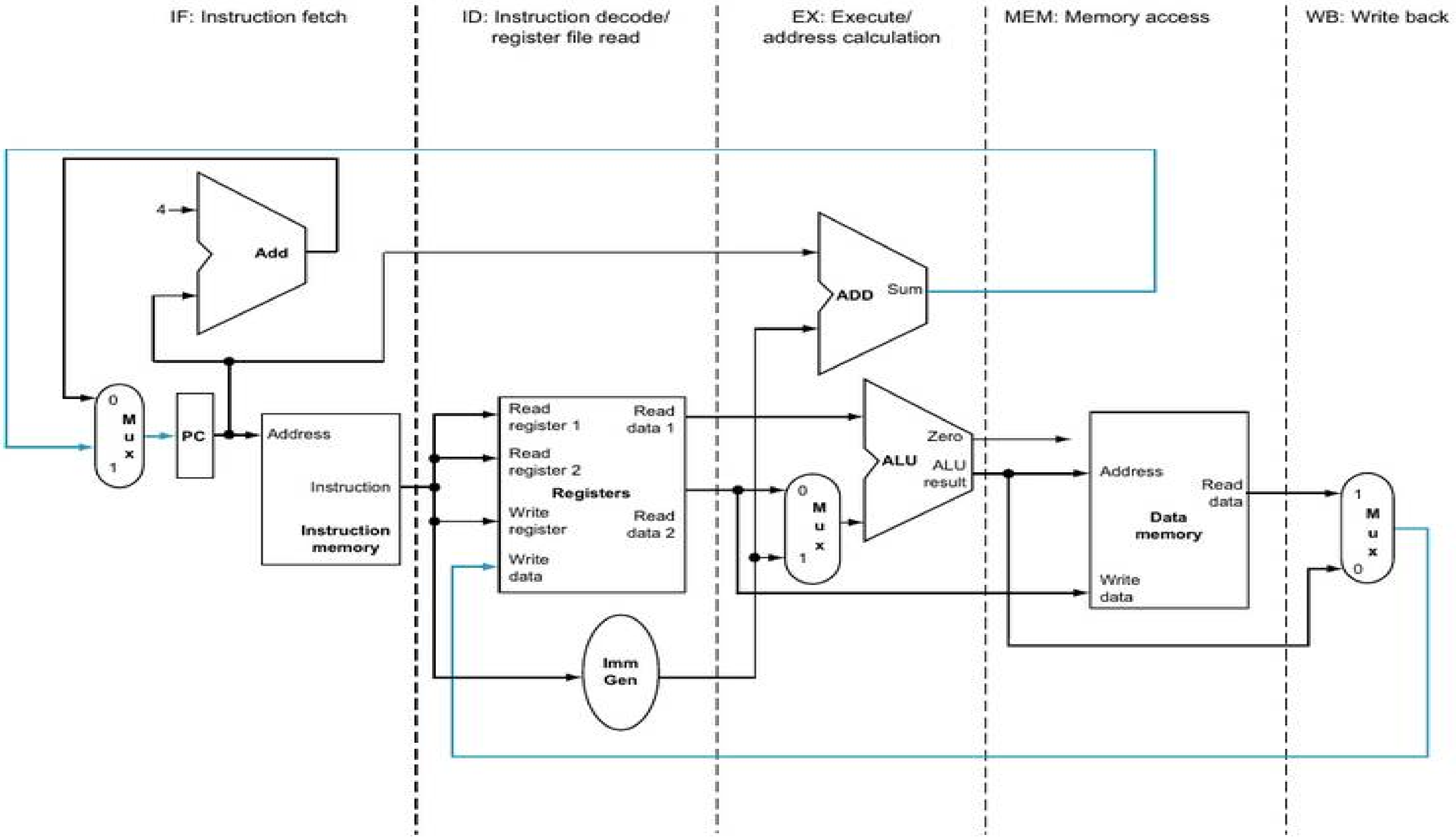
• Control Hazard

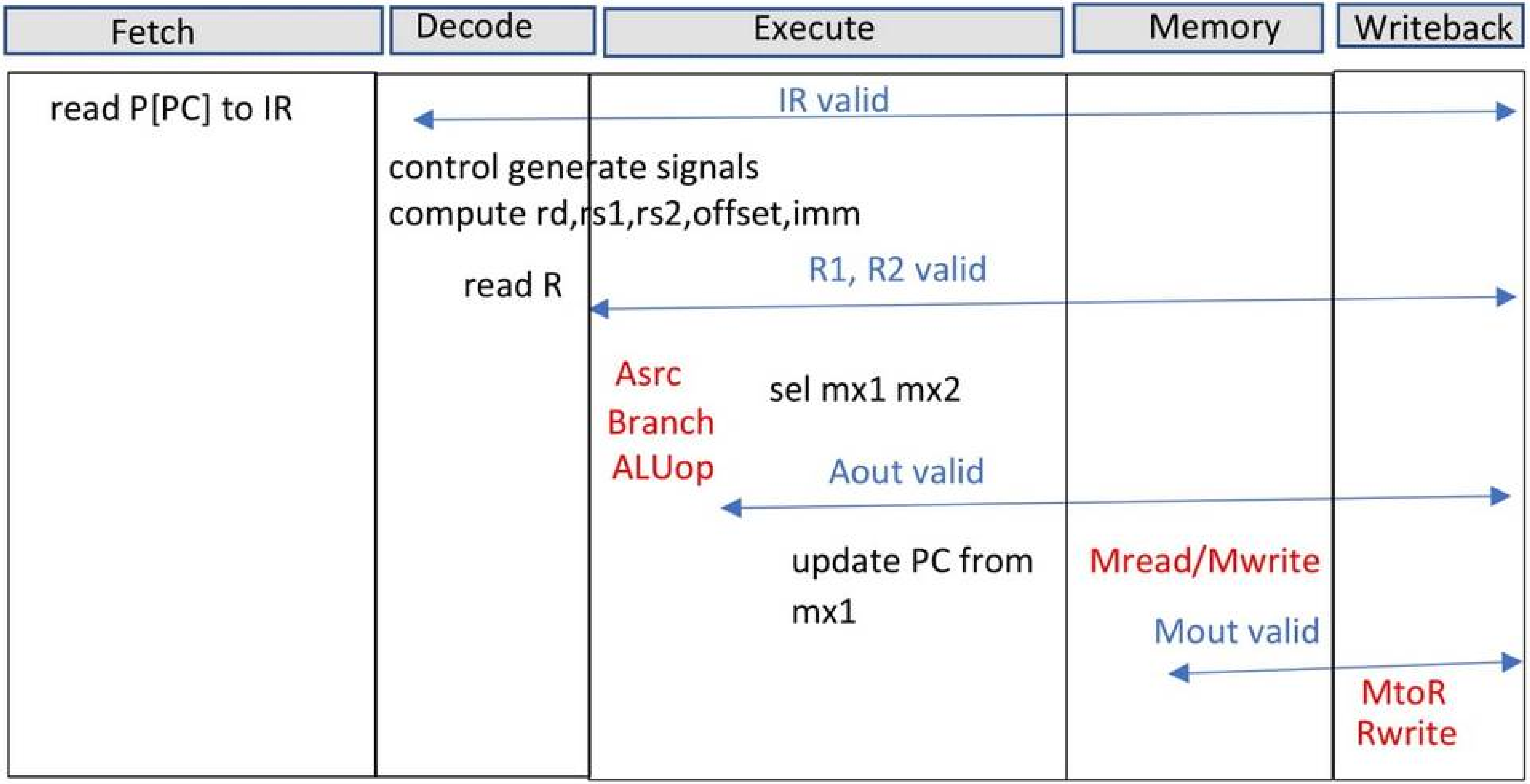
- Definition: A control hazard, or branch hazard, arises when the pipeline makes wrong assumptions about the outcome of a branch (conditional jump) instruction. The pipeline may start fetching the wrong set of instructions while waiting for the branch decision.
- Example: In the case of a branch instruction (e.g., an "if" statement), the processor may not know which path to take (branch taken or not taken) until the branch instruction is fully processed.
-
```
   beq x1, x2, LABEL   # Instruction 1: Branch if x1 == x2
   add x3, x4, x5      # Instruction 2: Continue execution if no branch
   sub x6, x7, x8      # Instruction 3: Another instruction
LABEL:
  mul x9, x1, x2      # Instruction 4: Executed if branch is taken
```
- Solution: Techniques like branch prediction (guessing the outcome of a branch), delayed branching, and speculative execution are used to minimize the impact of control hazards.

Structural Hazard

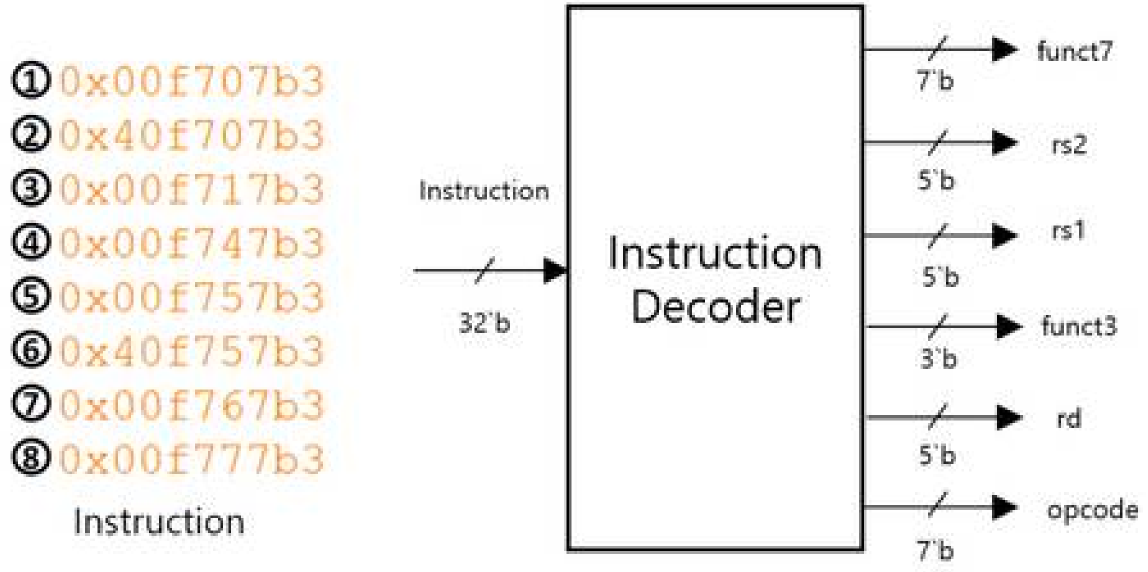
- A structural hazard occurs when two or more instructions need to use the same hardware resource at the same time. In other words, the hardware cannot support all the simultaneous instructions in the pipeline.
- Example: If a CPU has a single memory unit and two instructions (e.g., a load and a store) try to access memory at the same time, a structural hazard arises because only one memory access can occur at a time.
- lw x5, 0(x1)   # Instruction 1: Load word from memory (needs memory access)
  sw x6, 4(x1) # Instruction 2: Store word to memory (also needs memory access)
- Solution: Resource duplication (e.g., having separate instruction and data caches) or delaying one of the instructions (stalling) can resolve structural hazards.

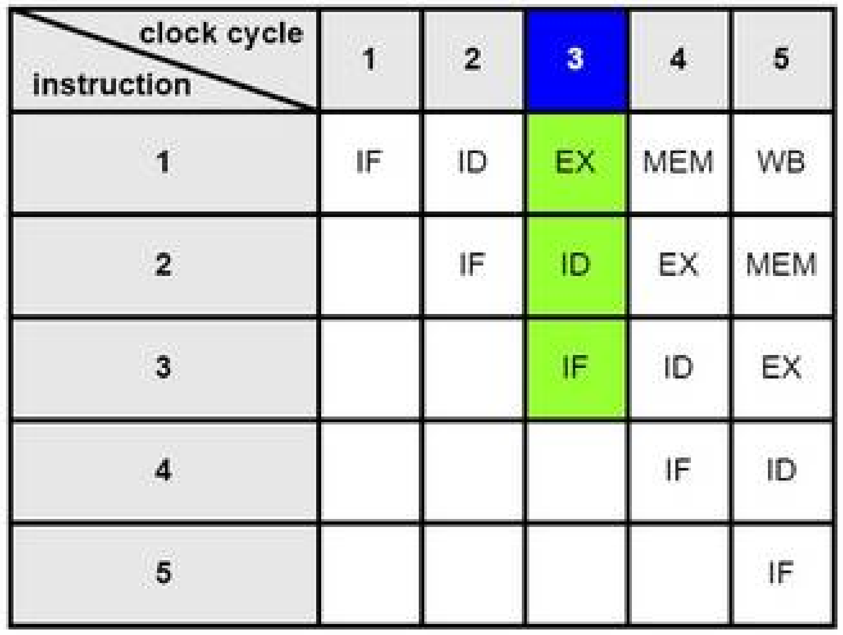



IF: Instruction fetch
ID: Instruction decode/ register file read
EX: Execute/ address calculation
MEM: Memory access
WB: Write back
4
Add
0
Mux
1
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Imm Gen
ADD
Sum
0
Mux
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
Mux
0


Fetch
Decode
Execute
Memory
Writeback
read P[PC] to IR
IR valid
control generate signals
compute rd,rs1,rs2,offset,imm
read R
R1, R2 valid
Asrc
Branch
ALUop
sel mx1 mx2
Aout valid
update PC from mx1
Mread/Mwrite
Mout valid
MtoR
Rwrite

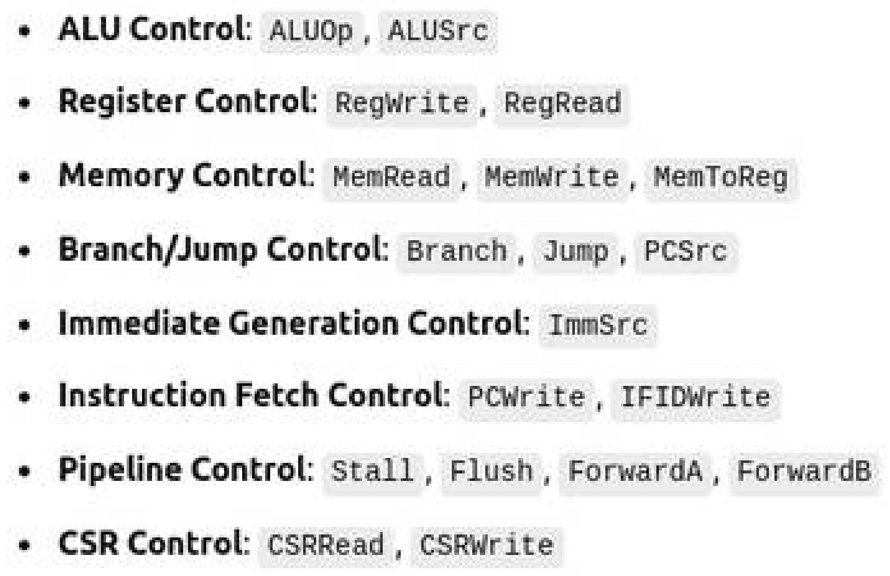
- **1. Instruction Decoder:**
- Decodes the fetched instruction from memory into signals that specify the operation to be performed.
  Breaks down machine code into control signals that tell various parts of the processor what to do next.
- **2. Sequencing Logic:**
- Controls the order in which operations are carried out by determining the next instruction to be executed (through the Program Counter).
  Manages the fetching, decoding, and execution cycle of instructions.
  Synchronizes the processor's operation, often tied to the system clock.





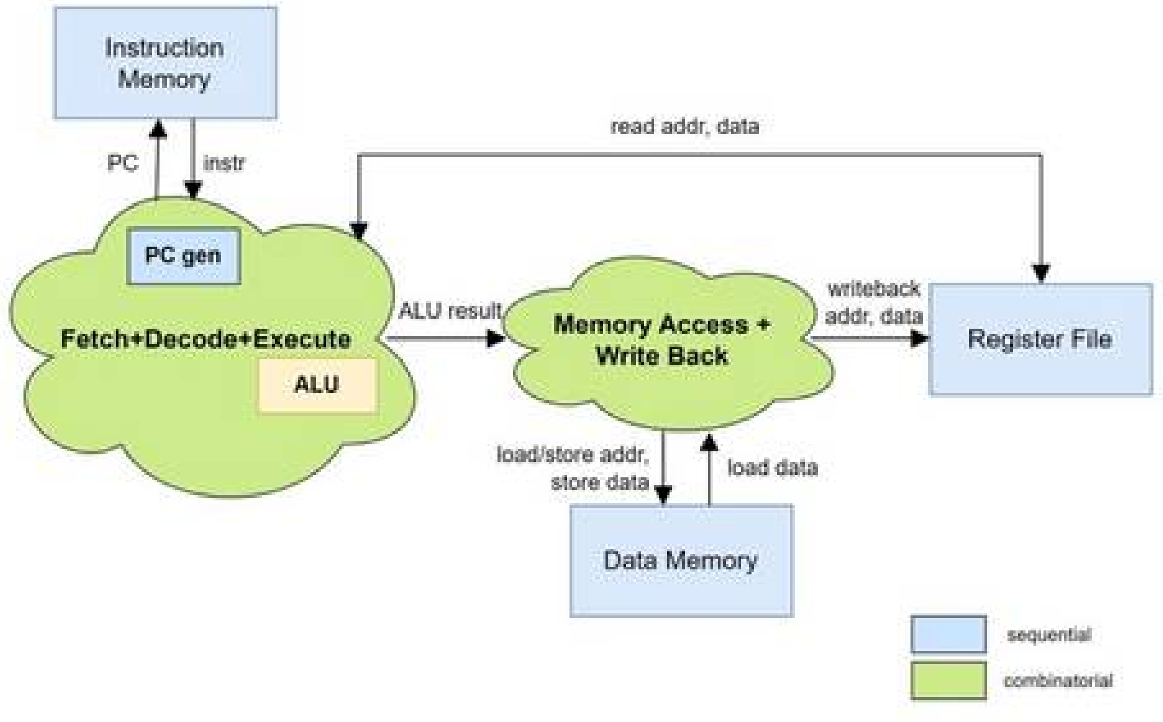
clock cycle / instruction	1	2	3	4	5
1	IF	ID	EX	MEM	WB
2		IF	ID	EX	MEM
3			IF	ID	EX
4				IF	ID
5					IF

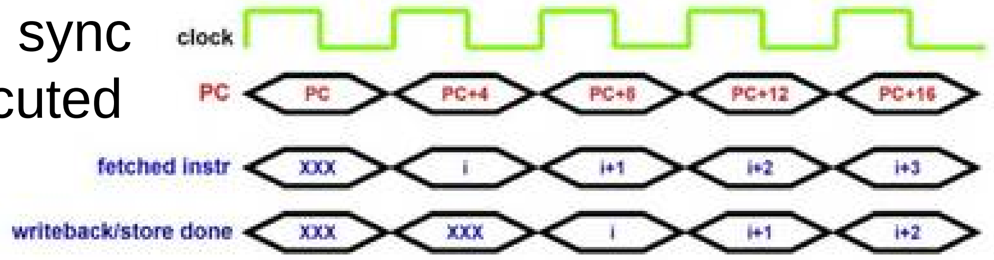
- **3. Control Logic Circuit:**
- Contains the logic gates and combinational circuits that generate control signals based on the instruction decoded.
  These control signals manage the internal data flow, timing, and operation of functional units (ALU, registers, etc.).
- **4. Control Signal Generator:**
- Generates the necessary control signals that dictate the actions of other parts of the CPU (ALU, memory interface, etc.).
  These signals direct data movement, ALU operation, register writes, and memory accesses.



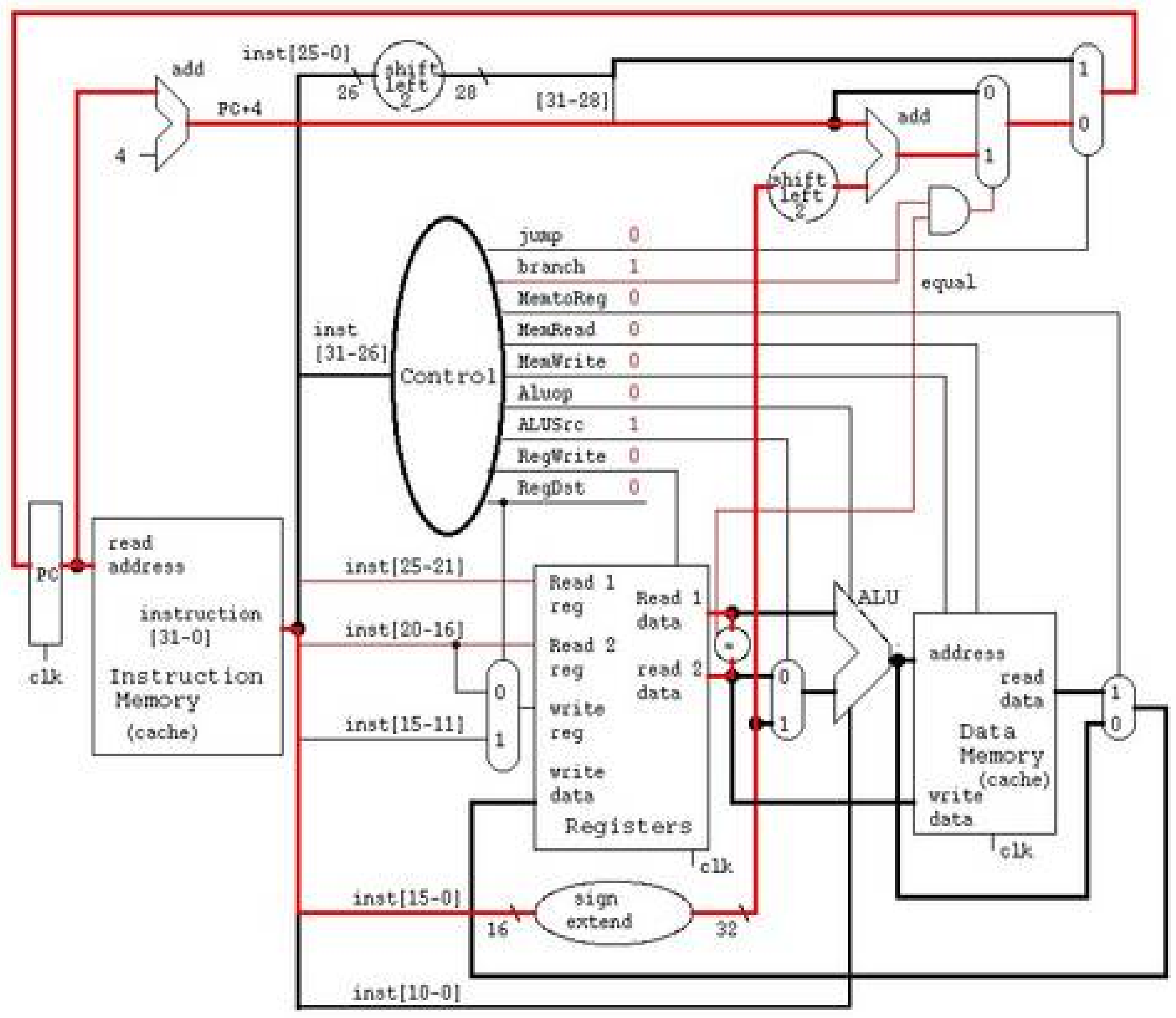
- **ALU Control**: `ALUOp`, `ALUSrc`
- **Register Control**: `RegWrite`, `RegRead`
- **Memory Control**: `MemRead`, `MemWrite`, `MemToReg`
- **Branch/Jump Control**: `Branch`, `Jump`, `PCSrc`
- **Immediate Generation Control**: `ImmSrc`
- **Instruction Fetch Control**: `PCWrite`, `IFIDWrite`
- **Pipeline Control**: `Stall`, `Flush`, `ForwardA`, `ForwardB`
- **CSR Control**: `CSRRead`, `CSRWrite`

- **5. Control Instruction (PC):**
  Manages the Program Counter, which holds the address of the next instruction to be executed.
  Handles instruction sequencing, updating the PC after each instruction or adjusting it for branch and jump operations.
- **6. Status Flag Register:**
  Contains flags that hold status information about the result of previous operations (e.g., Zero, Carry, Overflow, Sign flags).
  These flags help the Control Unit make decisions regarding branching and conditional operations.
- **7. Timing and Clock Control:**
  Coordinates the timing of operations across the CPU with the help of clock signals.
  Ensures that all parts of the processor operate in sync and that each step of the instruction cycle is executed at the correct time.

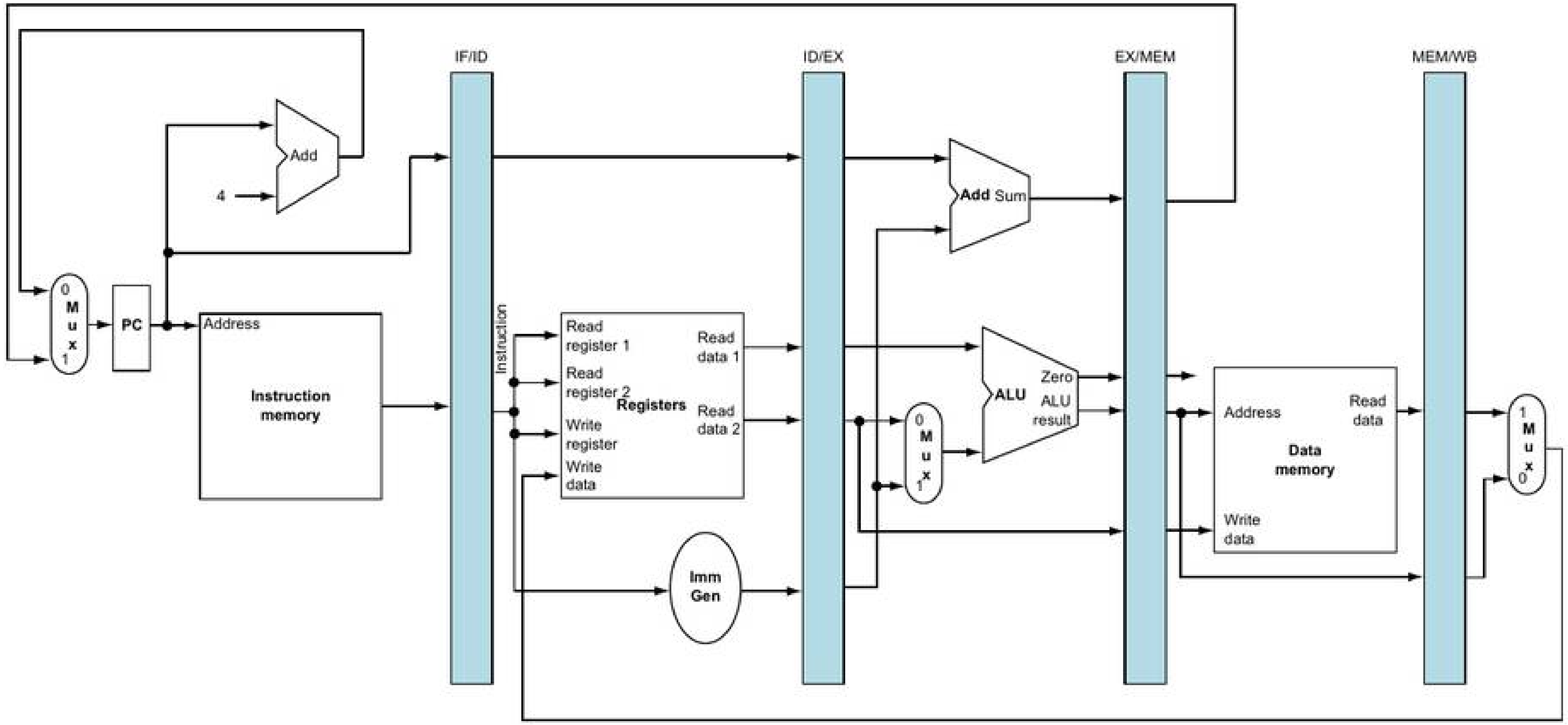




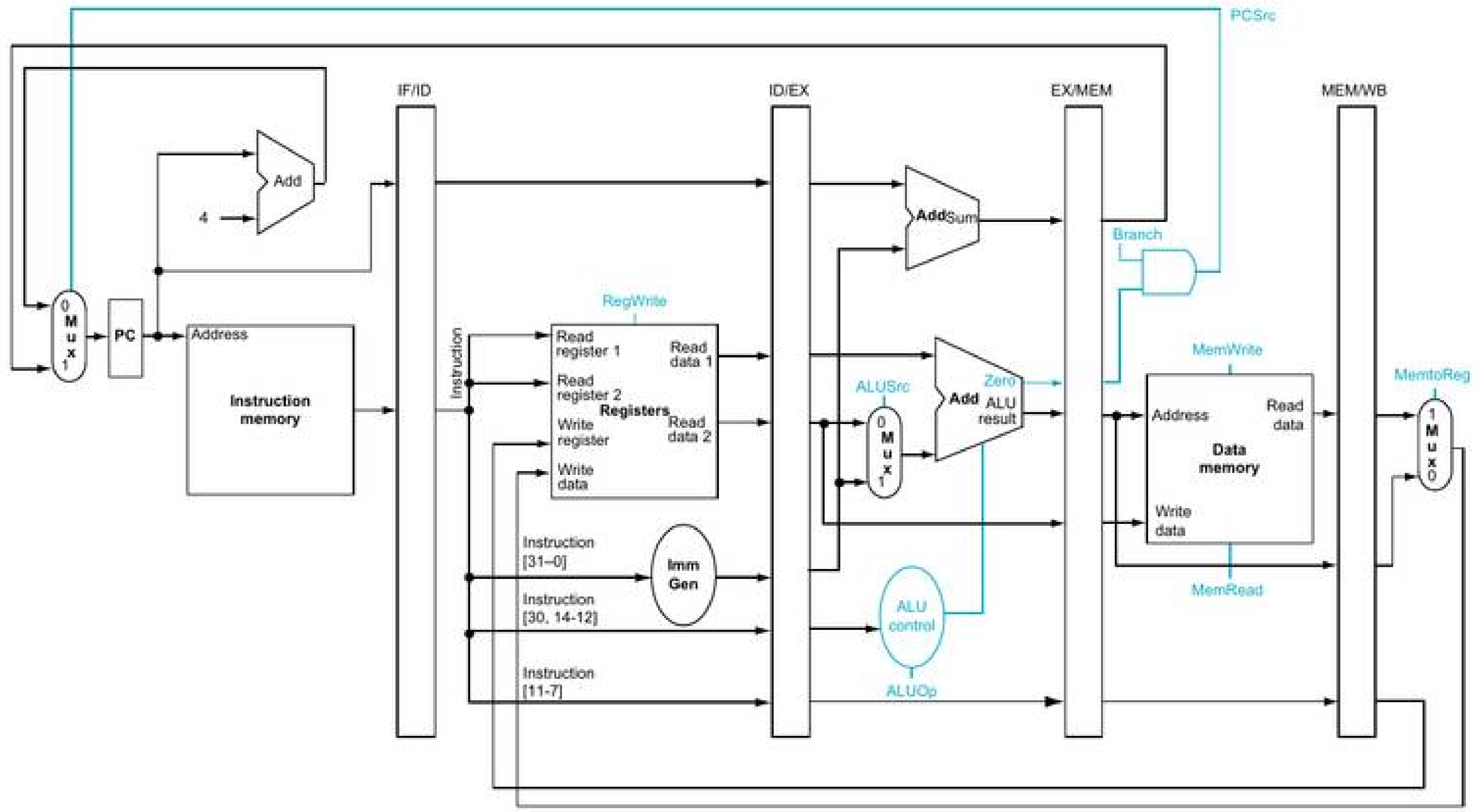
- **8. Branch and Jump Control:**
- Manages control transfer instructions, such as branches, jumps, and calls.
  Works with the Branch Prediction Unit (in modern processors) to optimize branching and minimize delays caused by pipeline stalls.
- **9. Interrupt Control:**
  Handles interrupts by suspending the current execution and transferring control to the appropriate interrupt service routine.
  Prioritizes interrupts and manages interrupt requests.
- **10. Microprogram Control Storage (in microprogrammed Cus):**
  In microprogrammed control units, the control signals are generated by executing a sequence of microinstructions stored in a microprogram memory (control memory).
  Each instruction in the CPU is mapped to a set of microinstructions that control specific low-level operations.

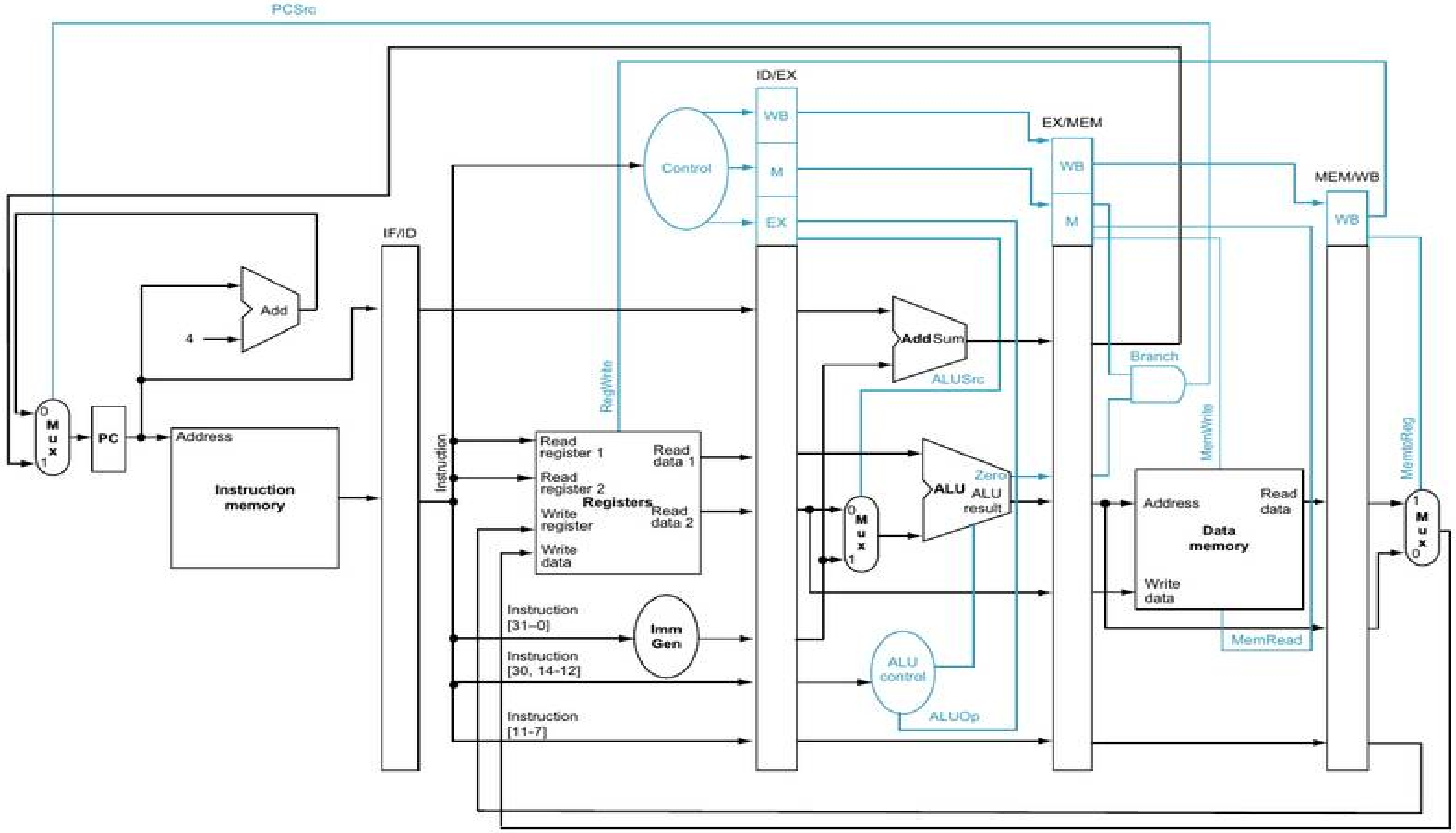


Pipeline Version of Data-path



Pipeline Version of Control Path




PCSrc
ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
WB
M
WB
Add
4
Add Sum
Branch
ALUSrc
RegWrite
MemWrite
MemtoReg
Mux
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
Zero
ALU
ALU result
Address
Read data
Data memory
Write data
Instruction [31–0]
Imm Gen
Instruction [30, 14-12]
ALU control
ALUOp
MemRead
Instruction [11-7]

Topics

1. Basic Processor Architecture
2. Problems with Single Cycle Processor Architecture
3. **Processor Architectures and Types**

- **Processor Architecture**
- **Types of Processors**

Procesor Architectures

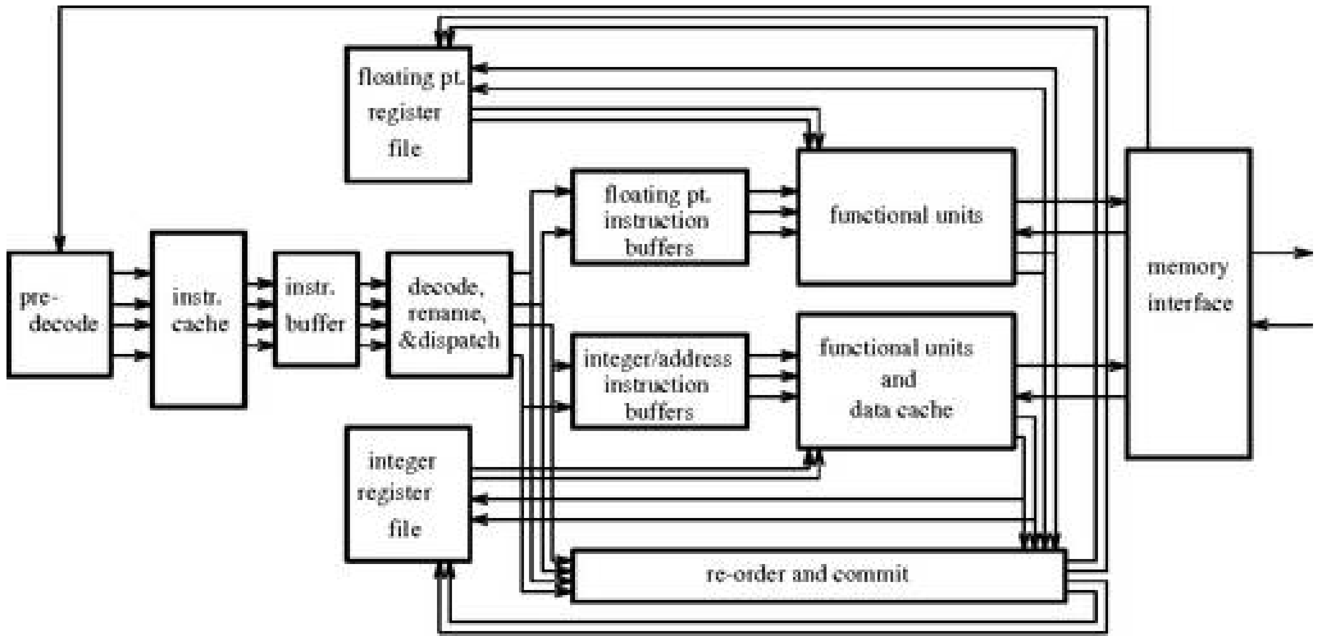
- Pipeline Architecture
- Superscalar Architecture
- Out-of-Order Execution (OoOE)
- In-Order Execution
- VLIW (Very Long Instruction Word) Architecture
- SIMD (Single Instruction, Multiple Data) Architecture
- MIMD (Multiple Instruction, Multiple Data) Architecture

- Pipeline Architecture

- In pipelining, each instruction goes through several stages (fetch, decode, execute, etc.).
- Each stage processes a different instruction simultaneously, similar to an assembly line.

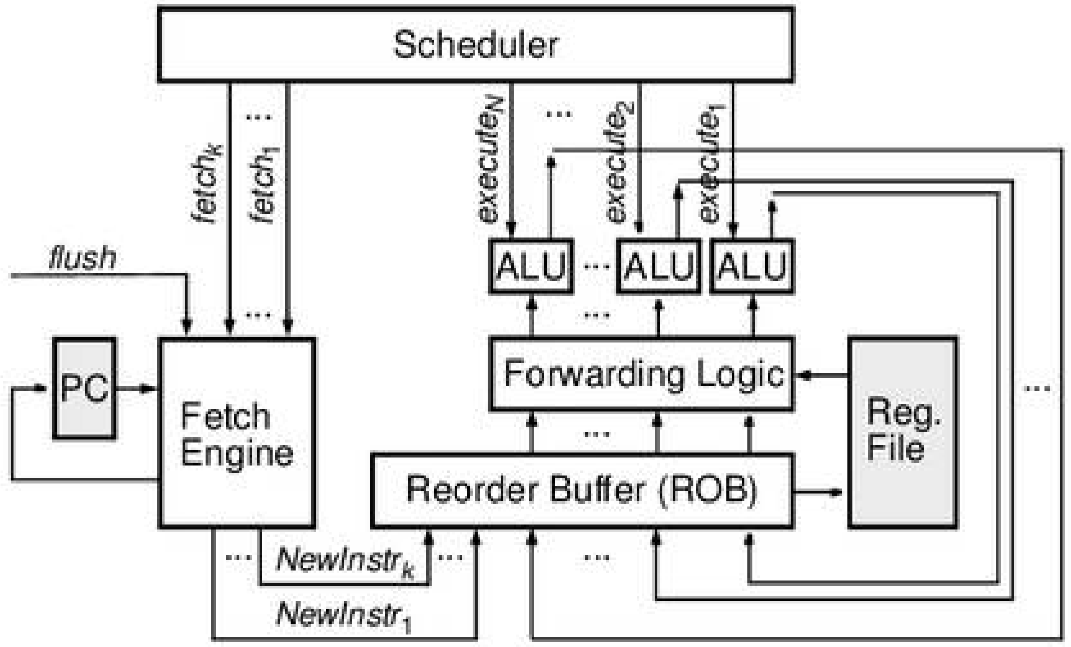
• Superscalar Architecture

- Like pipelining, but instead of processing just one instruction per stage, superscalar processors can handle multiple instructions in the same stage, using multiple pipelines.
- It increases performance by allowing the processor to handle more instructions in parallel.


floating pt. register file
floating pt. instruction buffers
functional units
pre-decode
instr. cache
instr. buffer
decode, rename, &dispatch
integer/address instruction buffers
functional units and data cache
memory interface
integer register file
re-order and commit

• Out-of-Order (OoO) and In-Order Execution

- Instructions don’t need to be executed in the same order they appear in the program. The processor executes instructions when their inputs are ready, optimizing the use of execution units. It avoids idle time by allowing other instructions to proceed when one is stalled (e.g., waiting for data).
- In-Order Executes the Instructions strictly in the order they are fetched. Simplicity and lower power consumption.



• VLIW, SIMD, MIMD

- VLIW: Multiple operations are packed into a single "long instruction" that the processor executes in parallel. Increases parallelism, but it’s the compiler's job to group instructions efficiently.
- SIMD: One instruction operates on multiple data points at the same time, useful in tasks like multimedia processing (e.g., working on many pixels simultaneously). High efficiency for tasks with lots of repetitive operations.
- MIMD: Different processors or cores execute different instructions on different data simultaneously, often in multi-core processors. True parallelism for different tasks or programs running at the same time.

Processor Types

- ARM (Advanced RISC Machine)
- MIPS (Microprocessor without Interlocked Pipeline Stages)
- x86 (Intel and AMD)
- SPARC (Scalable Processor Architecture)
- PowerPC (Power Performance Computing)
- DSP (Digital Signal Processors)
- VLIW (Very Long Instruction Word) Architectures
- GPU (Graphics Processing Unit) Architectures