Lecture 13: Multi-Cycle Control Unit

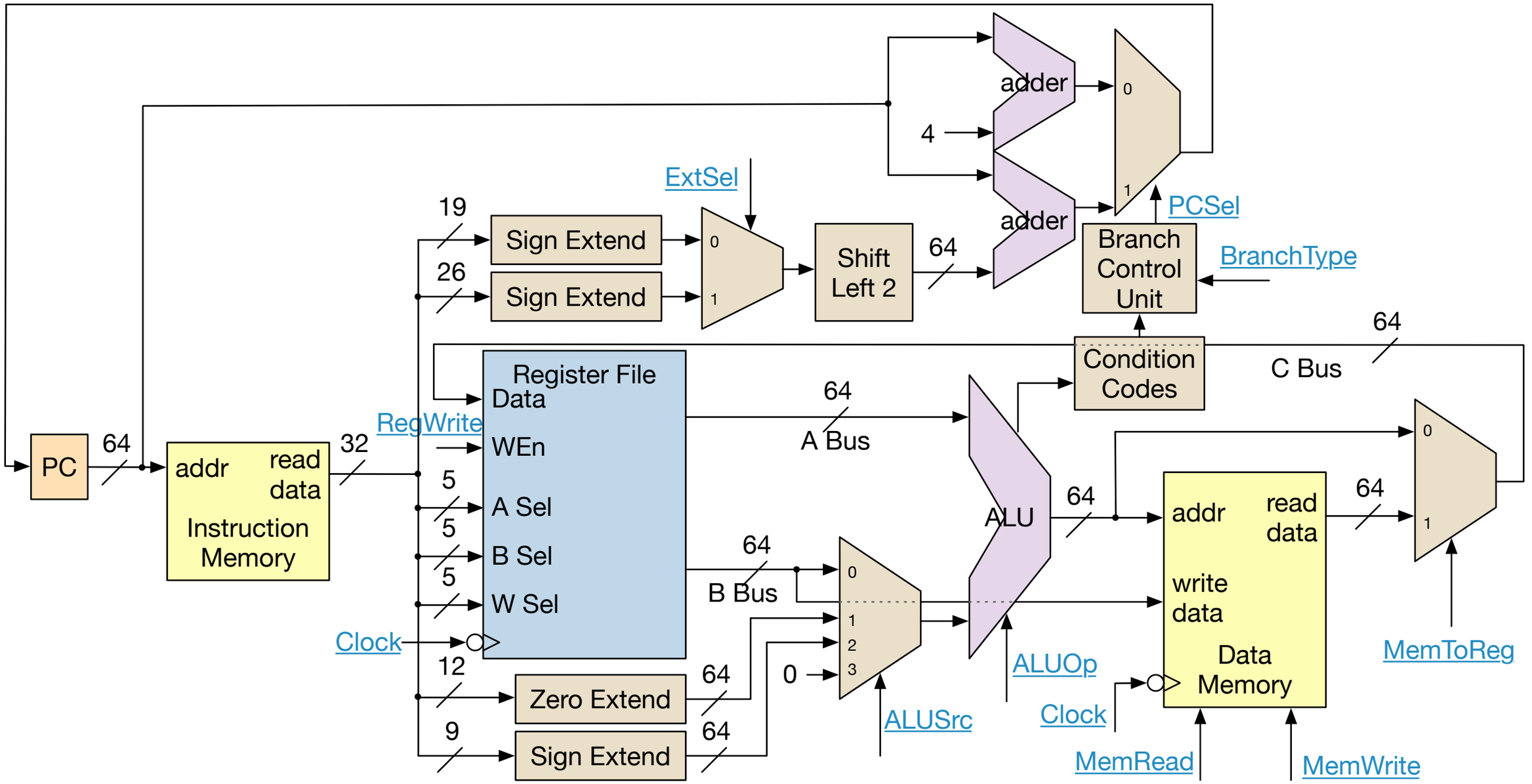
Spring 2024
Jason Tang

Topics

- Multi-cycle datapath
- Multi-cycle implementation
- Multi-cycle control

Single-Cycle Datapath

- A single-cycle datapath has, by necessity multiple ALUs, and also separate data and instruction memories



Motivation for Multi-Cycle Datapath

- Whereas a single-cycle datapath takes 1 clock cycle to execute any instruction, design a system that requires multiple **shorter** cycles to execute

- Allow the same functional unit to be used for different purposes during the datapath
    - Different uses on different clock cycles
- Allows sharing of resources, and thus reduces hardware
    - A single memory unit, for both instructions and data
    - A single ALU for all arithmetic, instead of an ALU and multiple adders

Designing Multi-Cycle Datapath

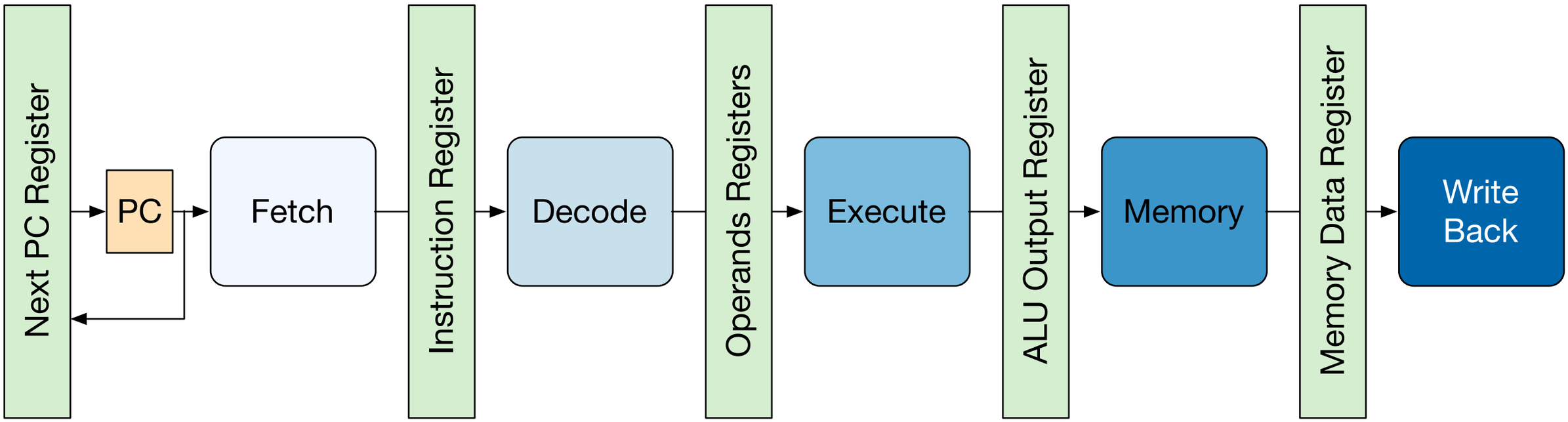
- At end of a clock cycle, **all data to be used** in a subsequent clock cycle must be stored
    - Internal registers added after every functional unit, to hold temporary values
- Operations are speculatively performed, and results are ignored if that operation is not needed
    - Example: A value will be written to **`BSel`** (and thus a value comes out of register file's B Bus), even if the fetched instruction is D-Type (and thus has no second register operand)

Multi-cycle Datapath

- Five classic parts to multi-cycle datapath:
  - Fetch: Read from memory the instruction at PC
  - Decode: Determine instruction type and read from registers
  - Execute: Perform operation as given by instruction
  - Memory: Load or store a value from memory
  - Write Back: Store final value in register file

Partitioning Datapath

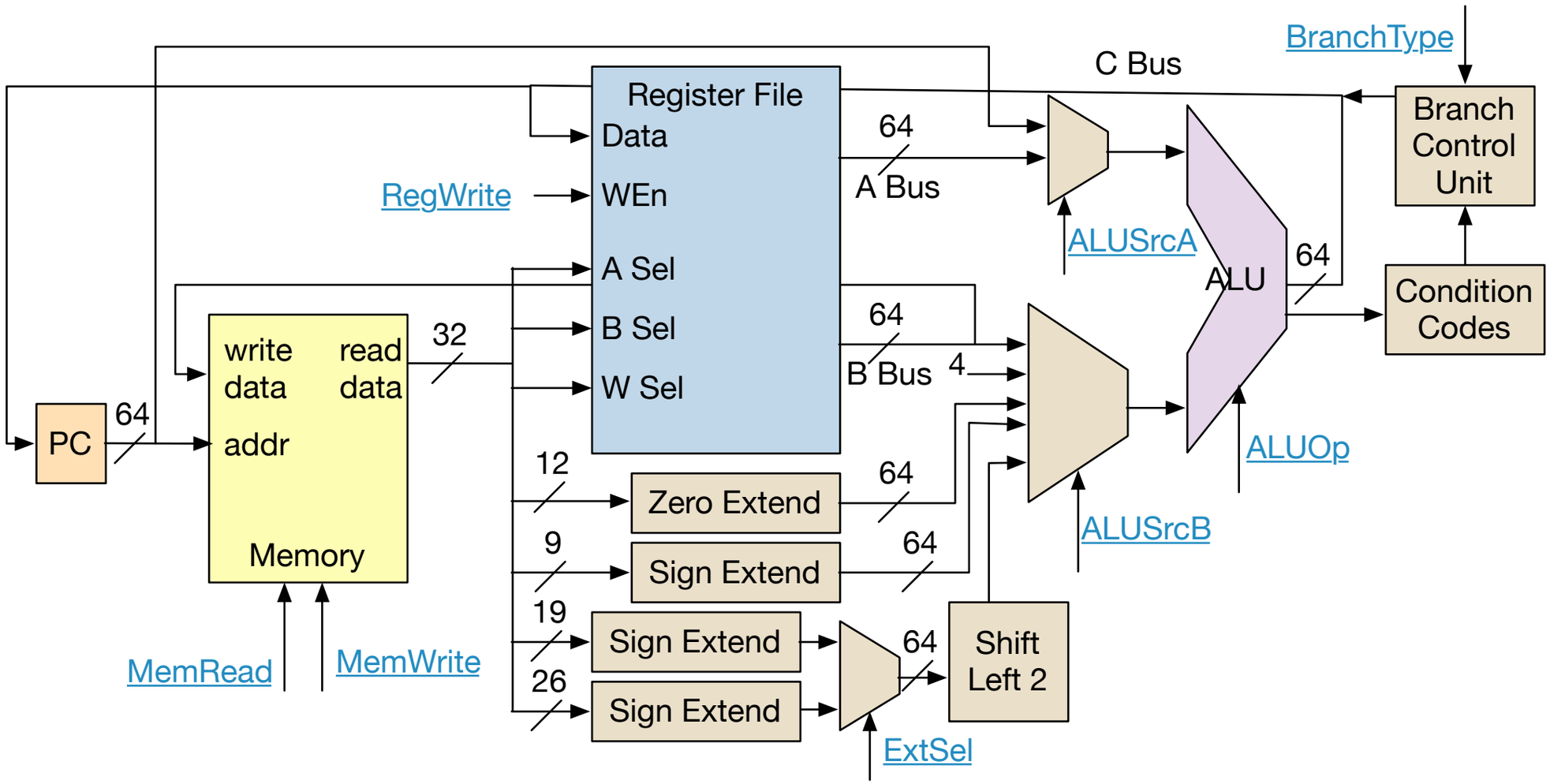
- Internal registers added between each stage to hold temporary values



- Not every stage is used for every instruction
  - Example: An unconditional branch to a PC-relative address skips Memory and Write Back stages

Incomplete Multi-cycle Datapath Design

- Instruction and data memory merged into single unit
- Single ALU also used to update PC



Multi-cycle RTN for `ldur`

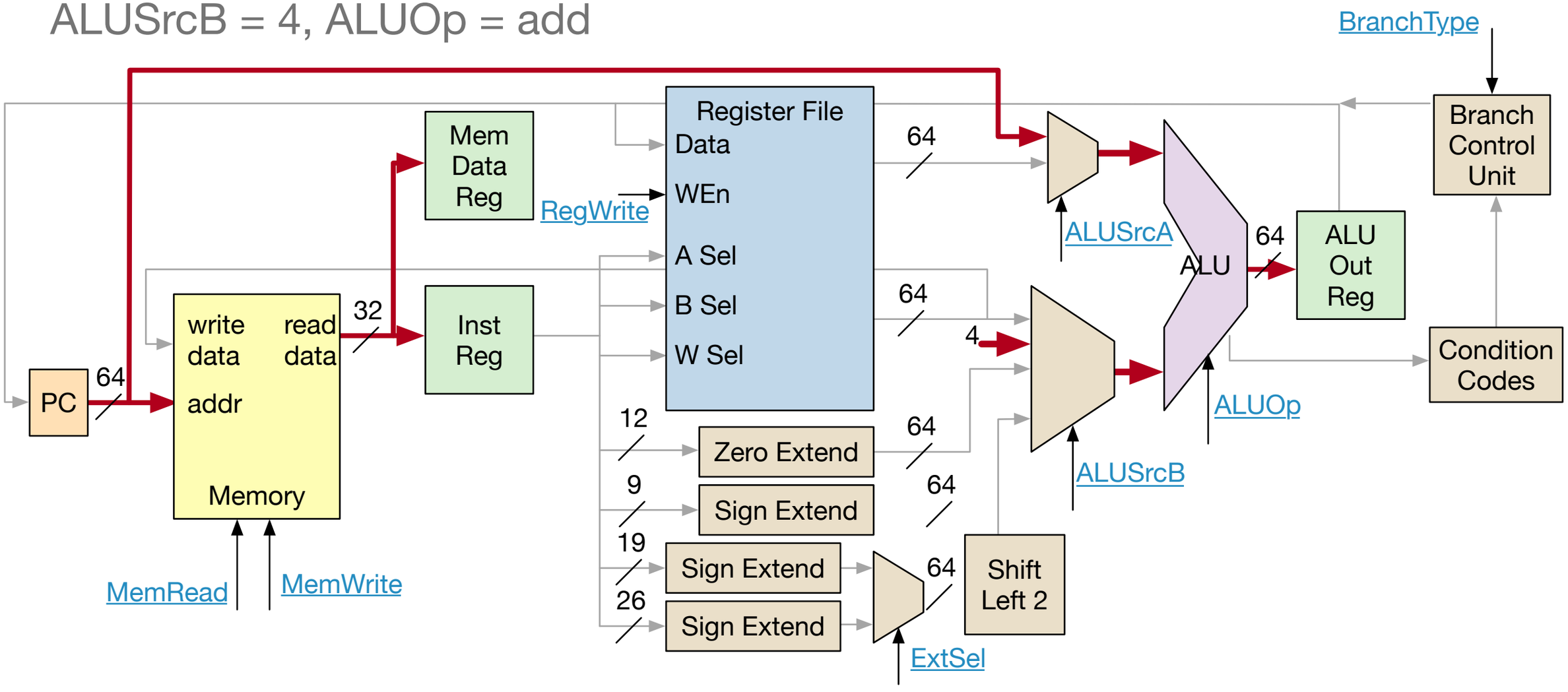
- `ldur` had one of the longest datapaths on ARMv8-A

- RTNs for each substage are:

Stage	Register Transfers	Controls Set
Fetch	IR ← Mem[PC]; ALUOut ← PC + 4	MemRead = 1, ALUSrcA = PC, ALUSrcB = 4, ALUOp = add
Decode	PC ← ALUOut; RegA ← X[n]; Imm9Reg ← SignExtend(imm9)	ASel = n, PCWrite = 1
Execute	ALUOut ← RegA + Imm9Reg	ALUSrcA = RegA, ALUSrcB = imm9, ALUOp = add
Memory	MemDataRegister ← Mem[ALUOut]	MemAddrSrc = ALUOut, MemRead = 1
Write Back	X[t] ← MemDataRegister	WSel = t, MemToReg = MDR, RegWrite = 1

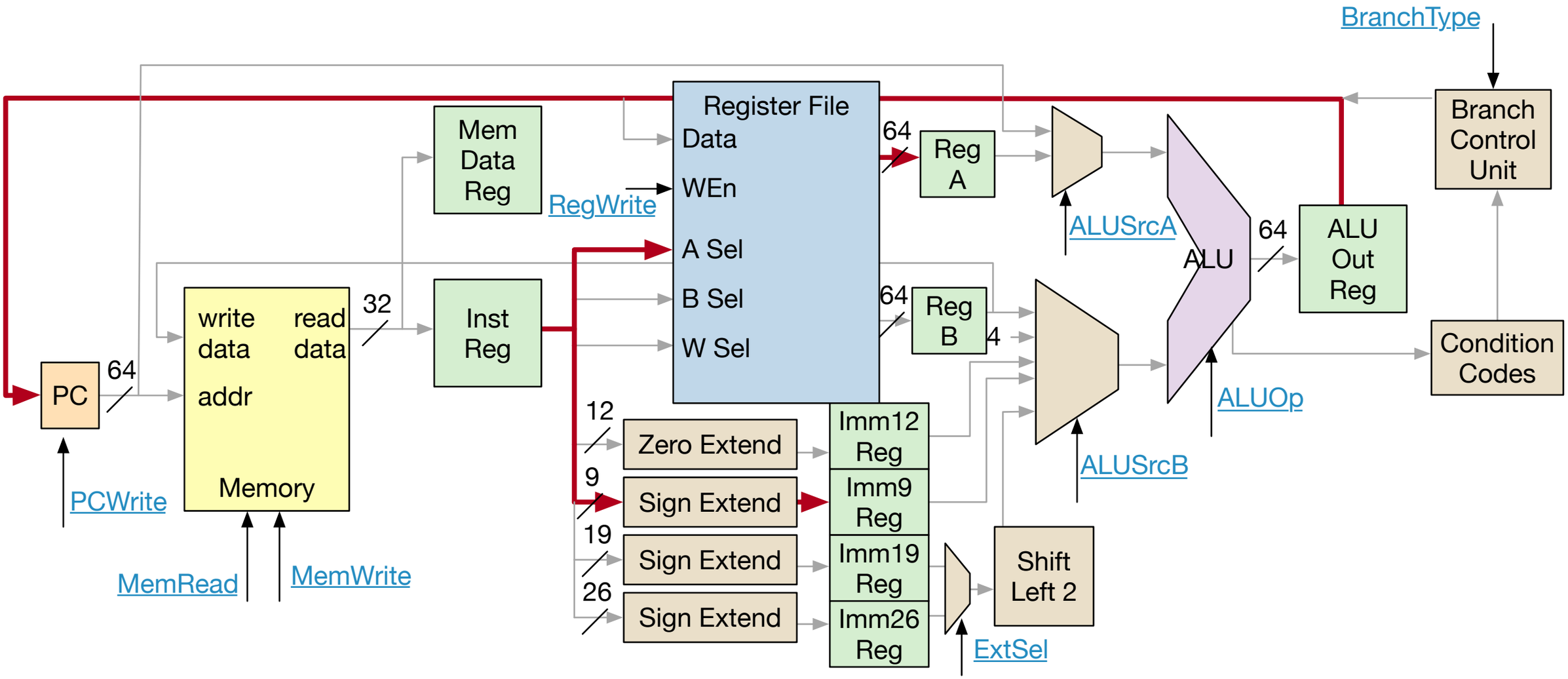
Multi-cycle `ldur` Fetch

- Controls set: MemAddrSrc = PC (not shown), MemRead = 1, ALUSrcA = PC, ALUSrcB = 4, ALUOp = add



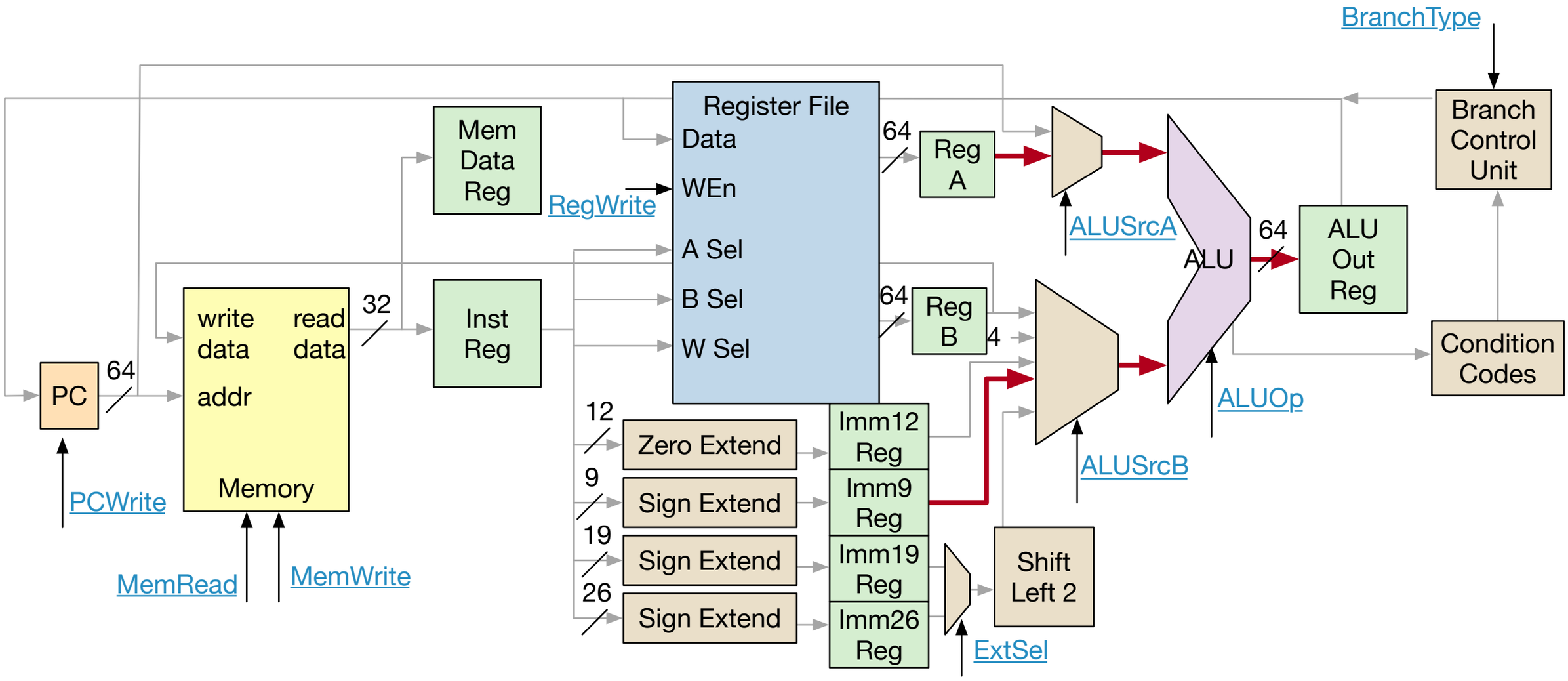
Multi-cycle `ldur` Decode

- Controls set: ASel = n, PCWrite = 1



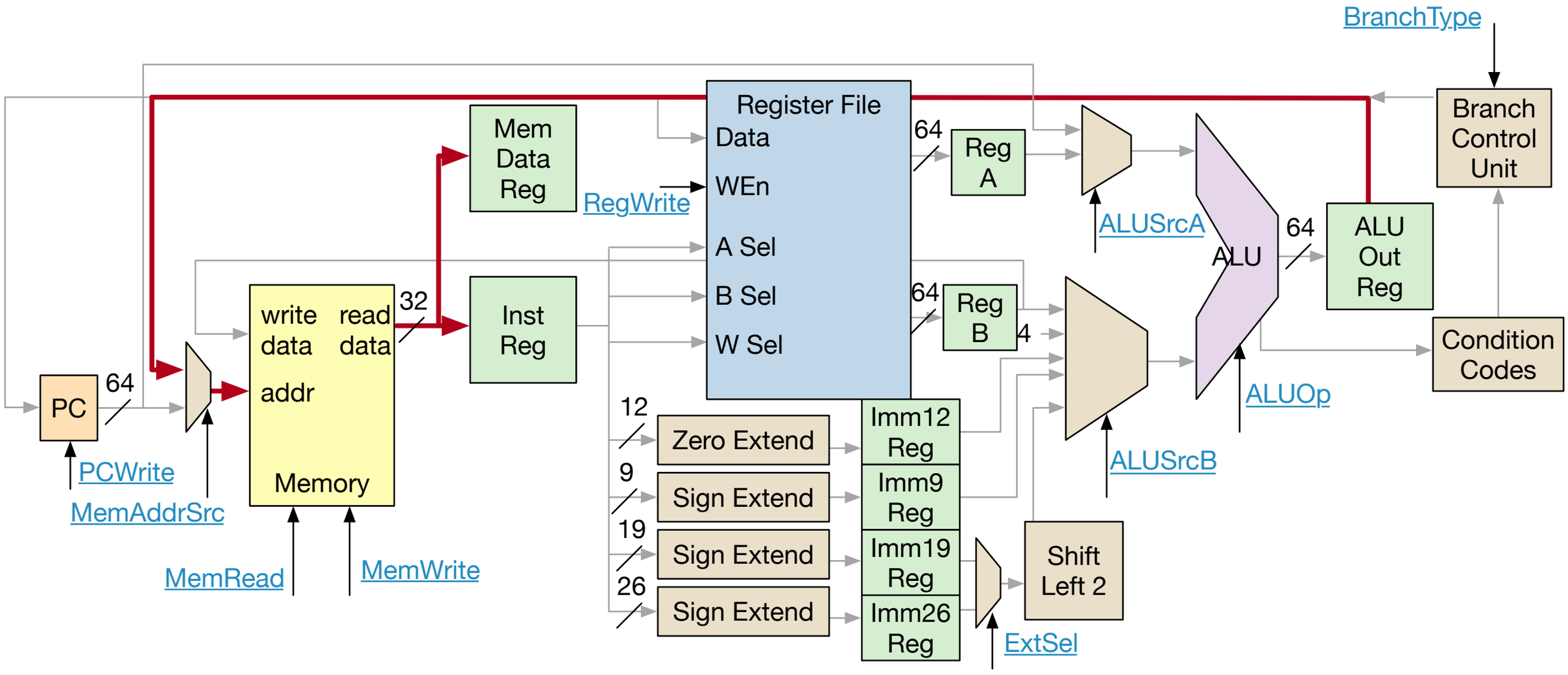
Multi-cycle `ldur` Execute

- Controls set: ALUSrcA = RegA, ALUSrcB = imm9, ALUOp = add



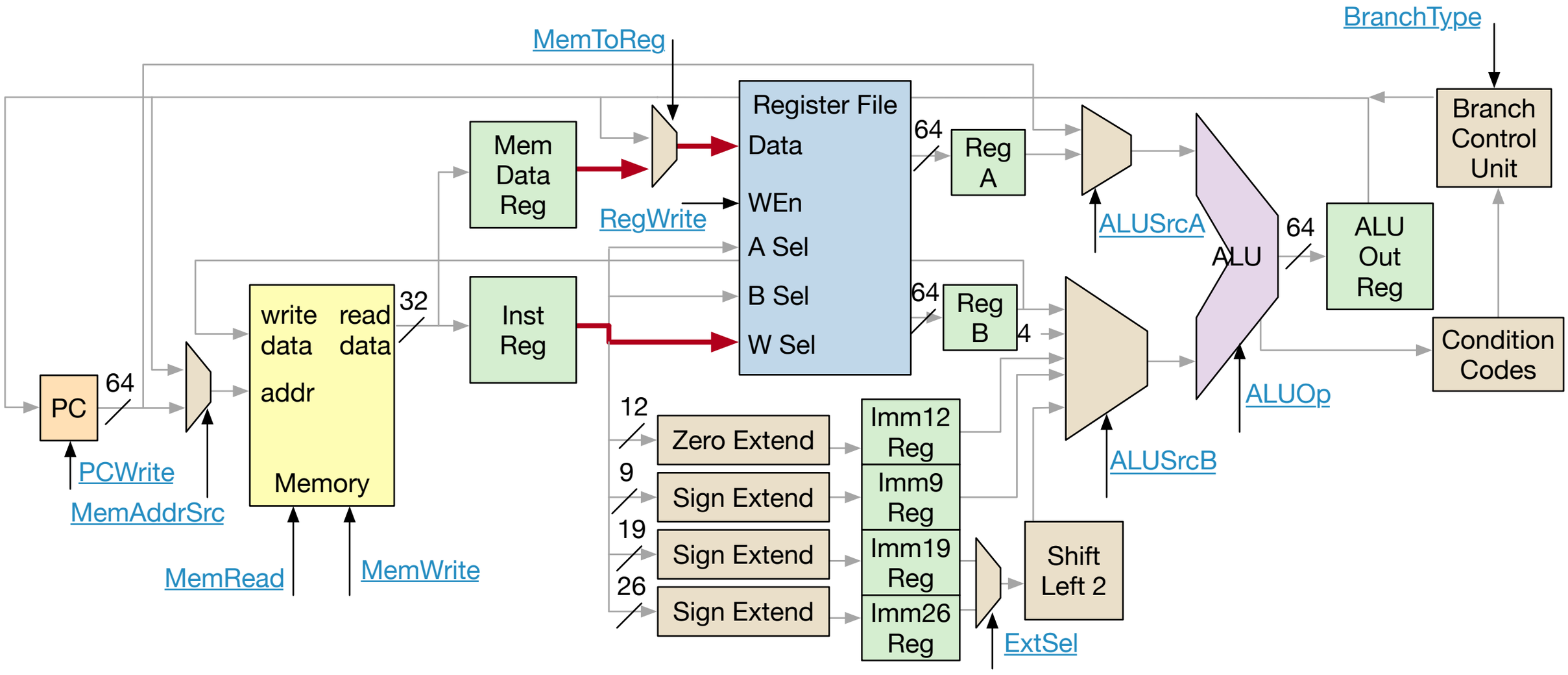
Multi-cycle `ldur` Memory

- Controls set: MemAddrSrc = ALUOut, MemRead = 1



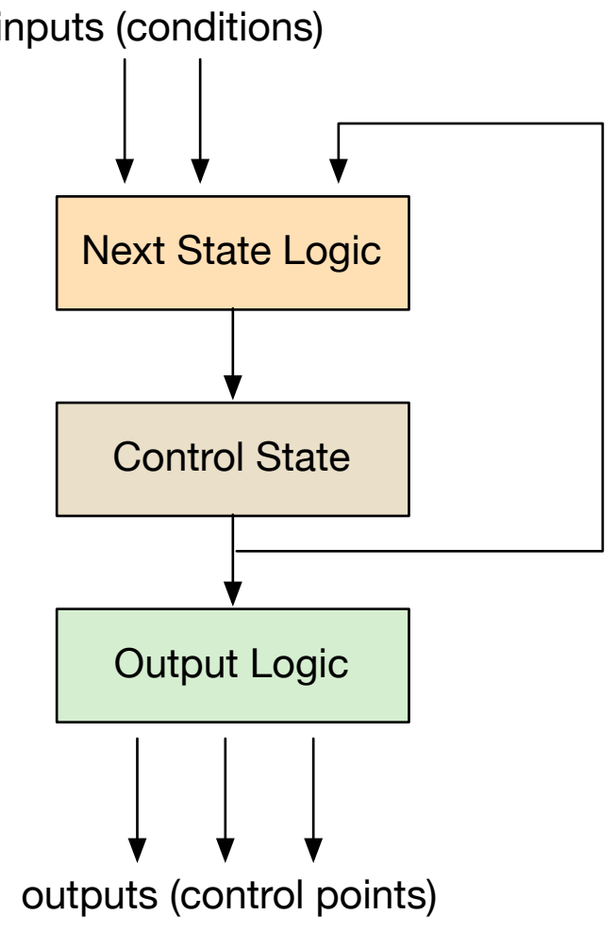
Multi-cycle `ldur` Write Back

- Controls set: WSel = t, MemToReg = MDR, RegWrite = 1

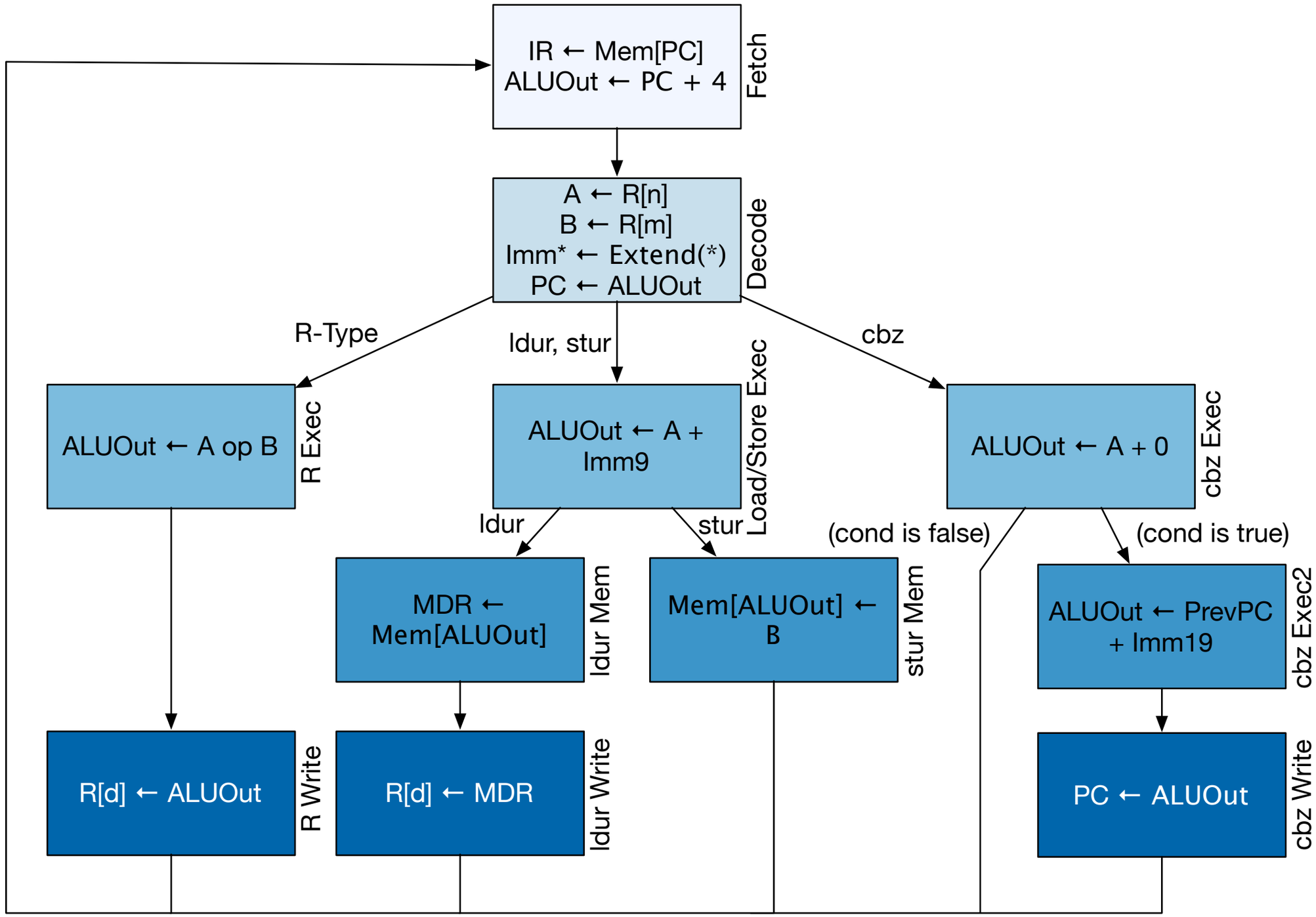


Control Model

- States specify control points for register transfers
- Transfer occurs upon exiting state (same clock edge)



Control Specification for Multi-Cycle Datapath



Detailed Control Specification

State	Instruction	Next State	ALUSrcB Control
Fetch	????	Decode	Constant 4
Decode	R-Type	R Exec	X
	ldur	Load/Store Exec	X
	stur	Load/Store Exec	X
	cbz	cbz Exec	X
R Exec	R-Type	R Write	RegB
Load/Store Exec	ldur	ldur Mem	Imm9
	stur	stur Mem	
cbz Exec	cbz	cbz Exec2 & Z	Constant 0
		Fetch & $\bar{Z}$	
ldur Mem	ldur	ldur Write	X
stur Mem	stur	Fetch	X
cbz Exec2	cbz	cbz Write	Imm19
R Write	R-Type	Fetch	X
ldur Write	ldur	Fetch	X
cbz Write	cbz	Fetch	X

Performance Evaluation

- State diagram gives CPI for each instruction, workload gives frequency of each type
  - Example:

Instruction	CPI	Example Frequency	CPI × Frequency
R-Type	4	40%	1.6
ldur	5	30%	1.5
stur	4	10%	0.4
cbz (taken)	5	20%	1.0
		Average CPI:	4.5