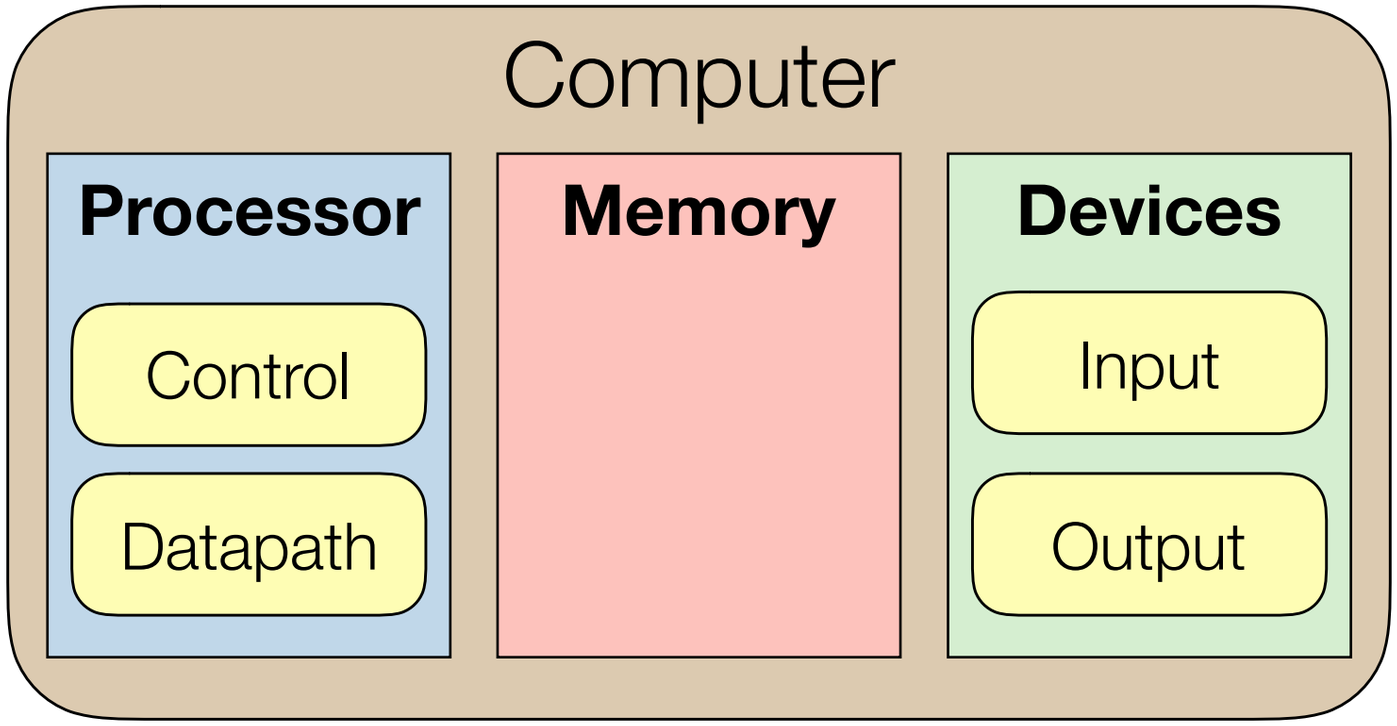
Lecture 11: Datapaths

Spring 2024
Jason Tang

Topics

- Datapaths
- Storage Elements
- Datapath Assemblies

Computer Organization



- So far, discussion has mostly focused on performance (instruction count, CPI, and clock cycle time)

- Design of processor itself (its datapath and control) determines clock cycle time and CPI

Processor Overview

- Analyze instruction set to derive datapath requirements
    - Meaning of instruction is given by register transfers, written in register transfer notation (RTN) syntax
    - Need storage to hold data (temporary registers) as instruction is executed
- Design datapath to move data around processor
    - Analyze instruction implementation to determine control points that affect register transfers
- Assemble control logic

ARMv8-A Instruction Formats

- Most instructions involve one or more internal registers, not just those visible to software

- Examples:

- **`add`** (R-type): take two registers, add them, store the sum in a third register, then update condition code register

- **`ldr`** (I-type): add an immediate value to a register, then use the result as the address from which to load a word from memory into a register

- Even a relative unconditional jump (**`b`**) affects the program counter (PC)

R-Type	opcode	R_m	shamt	R_n	R_d
	11 bits	5 bits	6 bits	5 bits	5 bits

D-Type	opcode	address	op2	R_n	R_t
	11 bits	9 bits	2 bits	5 bits	5 bits

I-Type	opcode	immediate	R_n	R_d
	10 bits	12 bits	5 bits	5 bits

B-Type	opcode	immediate
	6 bits	26 bits

CB-Type	opcode	immediate	R_t
	8 bits	19 bits	5 bits

Register Transfers

- Many instructions can be described by how it affects registers

Instruction	Register Transfer Notation (RTN)	Section†
`add` (shifted register)	(result, -) ← AddWithCarry(X[n], X[m], 0); X[d] ← result; PC ← PC + 4	C6.2.5
`ldur`	offset ← LSL(ZeroExtend(imm9, 64)); address ← X[n] + offset; X[t] ← Mem[address]; PC ← PC + 4	C6.2.202
`mov` (register)	X[d] ← X[m]; PC ← PC + 4	C6.2.224
`b`	offset ← SignExtend(imm26, 64); PC ← PC + (offset × 4)	C6.2.25
`bl`	offset ← SignExtend(imm26, 64); X[30] ← PC + 4; PC ← PC + (offset × 4)	C6.2.34

† Referenced section numbers are from the *ARM Architecture Reference Manual for A-profile architecture*

Stereotypical Datapath

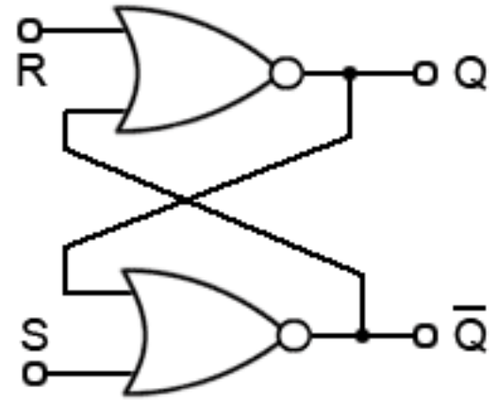
- Fetch next instruction from memory, at the address stored within PC
    - (Optionally) Store that instruction in an Instruction Register
- Decode instruction to determine which registers are affected
- Execute instruction, using the correct registers as operands
- (Usually) Write result back to a memory location
- (Usually) Increment PC afterwards

Register Design

- Registers store data in a computer
    - Built from individual flip-flop circuits
- Uses a clock signal to determine when to update stored value
- Edge-triggered: update value when clock line (Clk) changes
    - Usually leading edge-triggered (when clock changes from 0 to 1)
    - Alternative is falling edge-triggered (when clock changes from 1 to 0)

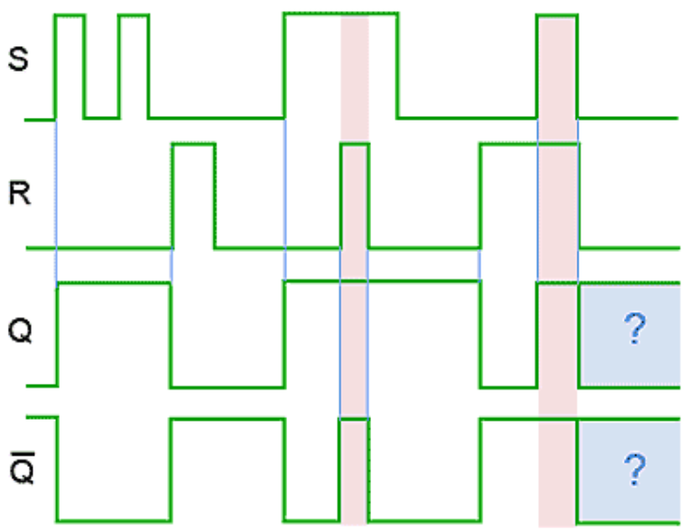
SR Latch

- Simplest memory design is the Set-Reset (SR) Latch
  - Holds a single bit



- No clock line, so any input change affects output *immediately*
- Slight propagation delay from when S is set until Q changes, then $\bar{Q}$ changes

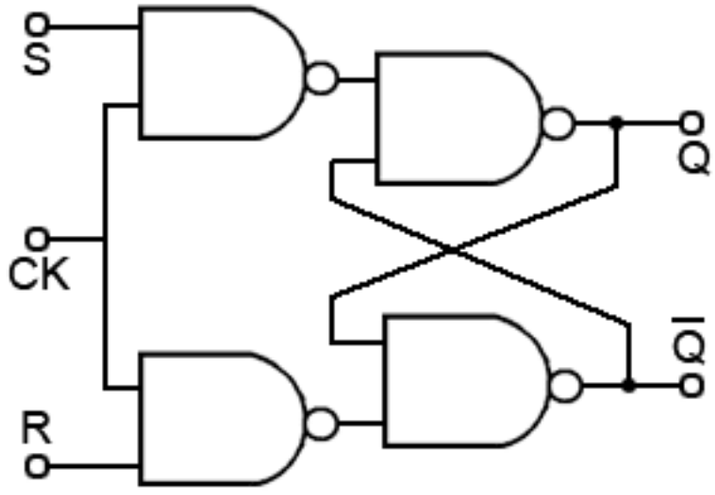
R	S	new Q	new $\bar{Q}$
0	0	previous Q	previous $\bar{Q}$
0	1	1	0
1	0	0	1
1	1	disallowed	disallowed



http://www.learnabout-electronics.org/Digital/dig52.php

Clocked SR Flip-Flop

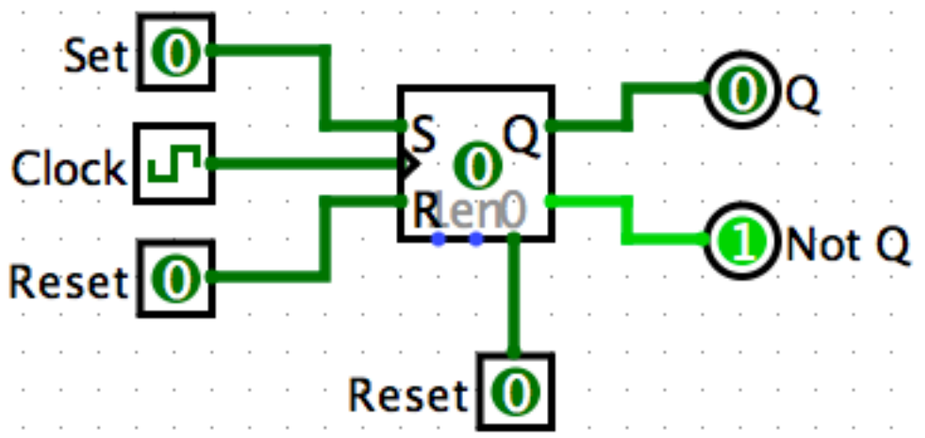
- Add a clock signal, that enables Set (S) and Reset (R) inputs



- S and R cannot change flip-flop state until clock is high

- S and R should settle into stable states while clock is low

  - When clock transitions from 0 to 1, stable S and R values will be used to determine flip-flop's new state

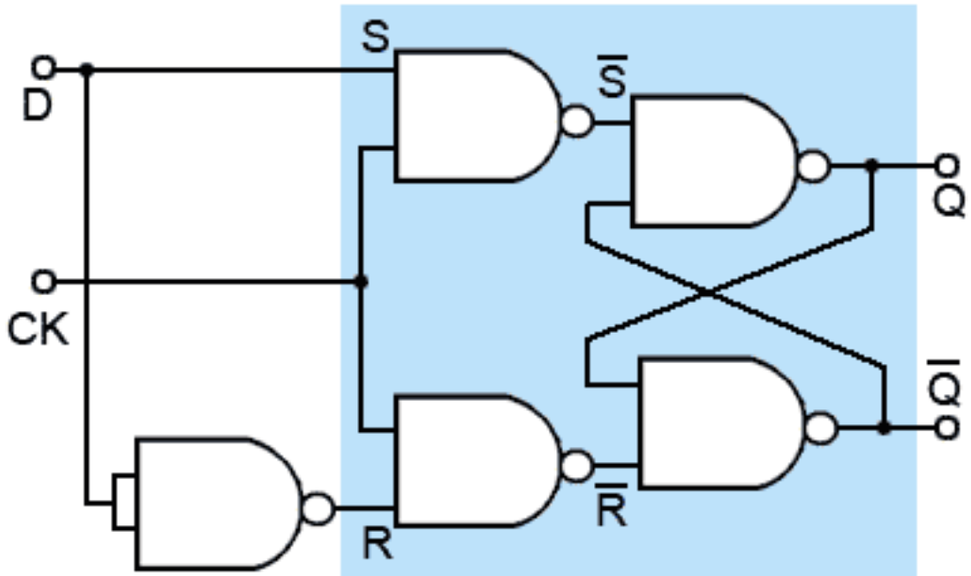


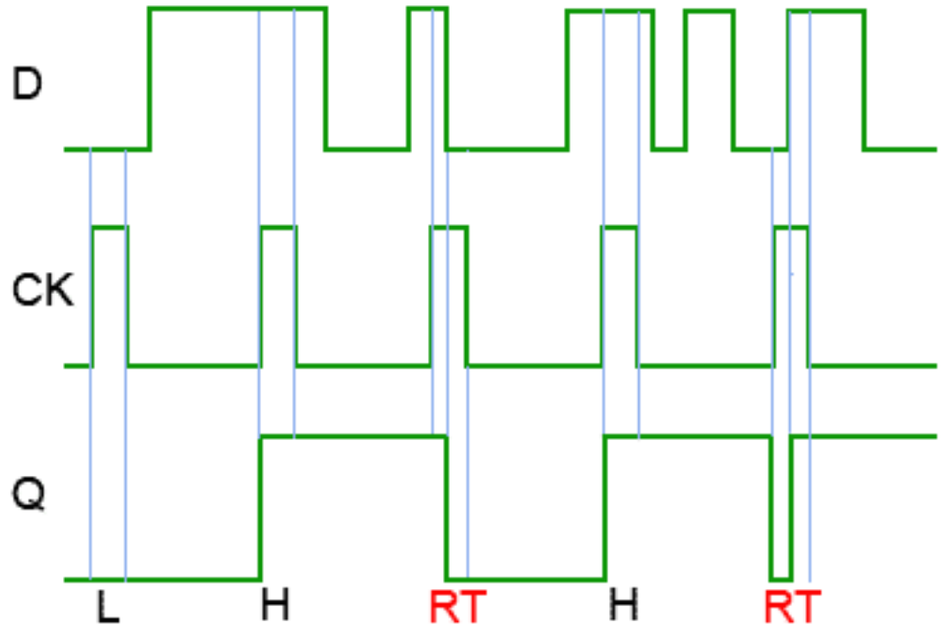
http://www.learnabout-electronics.org/Digital/dig52.php

Clock Delay Latch (D Flip-Flop)

- SR flip-flops have two inputs, S and R
- Use Delay (D) Flip-Flop to delay progress of data through a circuit
- If D changes while clock is high, then output will also change (ripple through)

CK	D	new Q	new $\bar{Q}$
0	X	previous Q	previous $\bar{Q}$
1	0	0	1
1	1	1	0

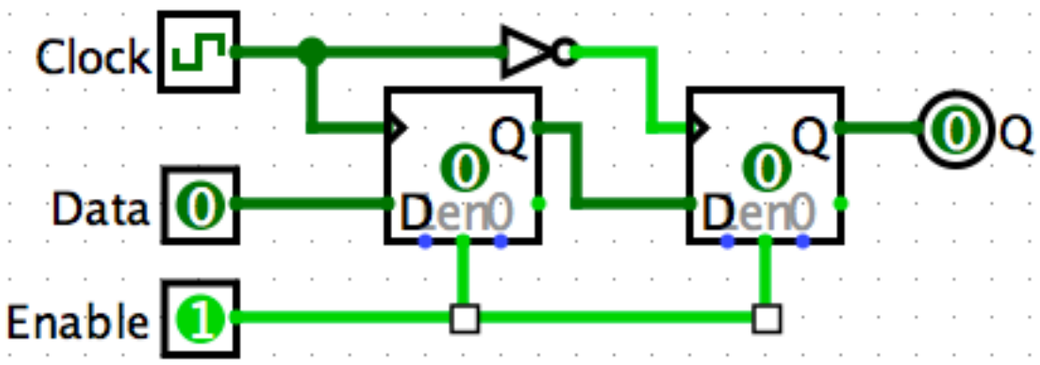


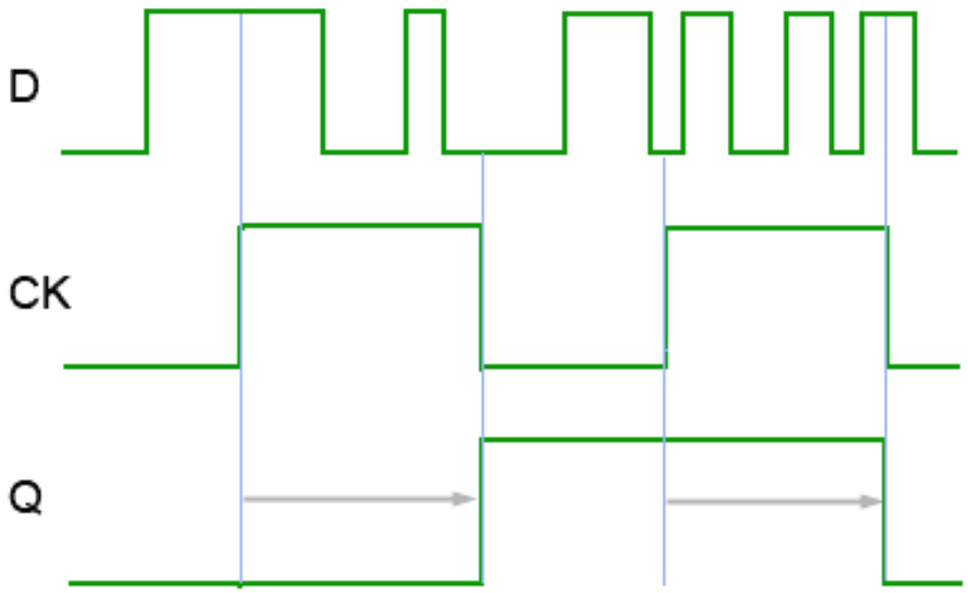


http://www.learnabout-electronics.org/Digital/dig53.php

Using D Flip-Flops

- Typically have a write-enable input that only updates when it is 1
- If output is used for some calculation, cannot write back to D flip-flop on the *same* clock cycle (else output will be unstable)
- Using a master/slave D flip-flop, the combined circuit will change at most once per rising clock edge

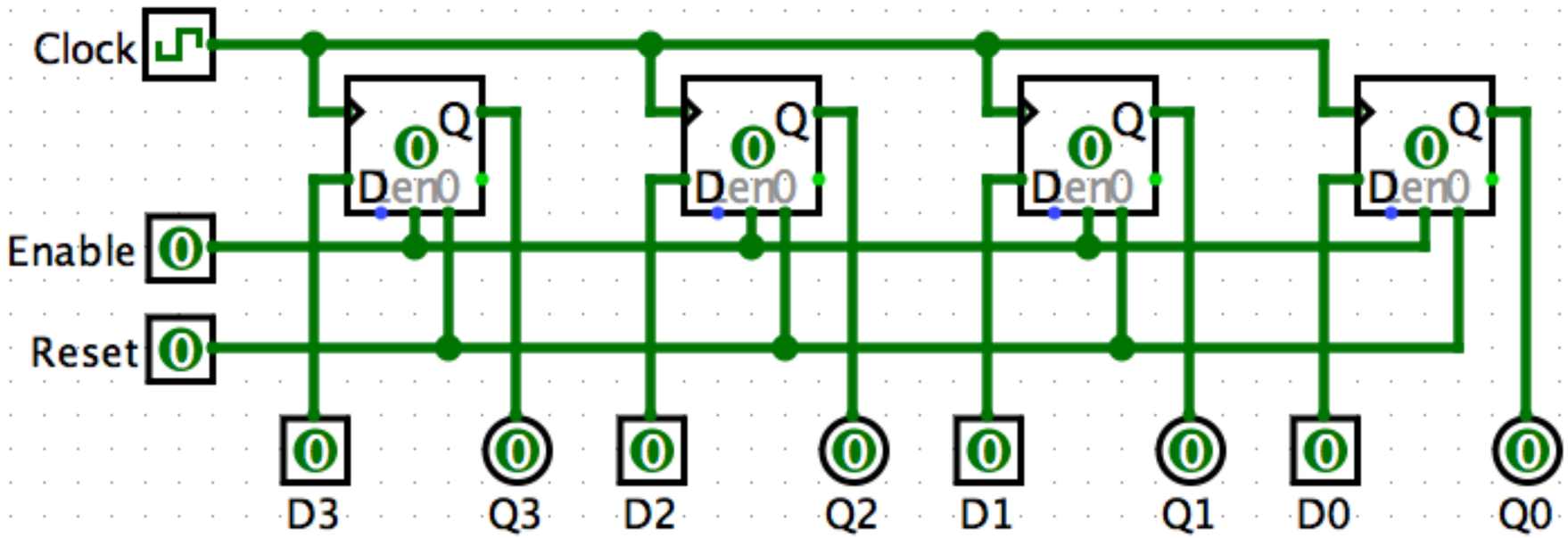




http://www.learnabout-electronics.org/Digital/dig53.php

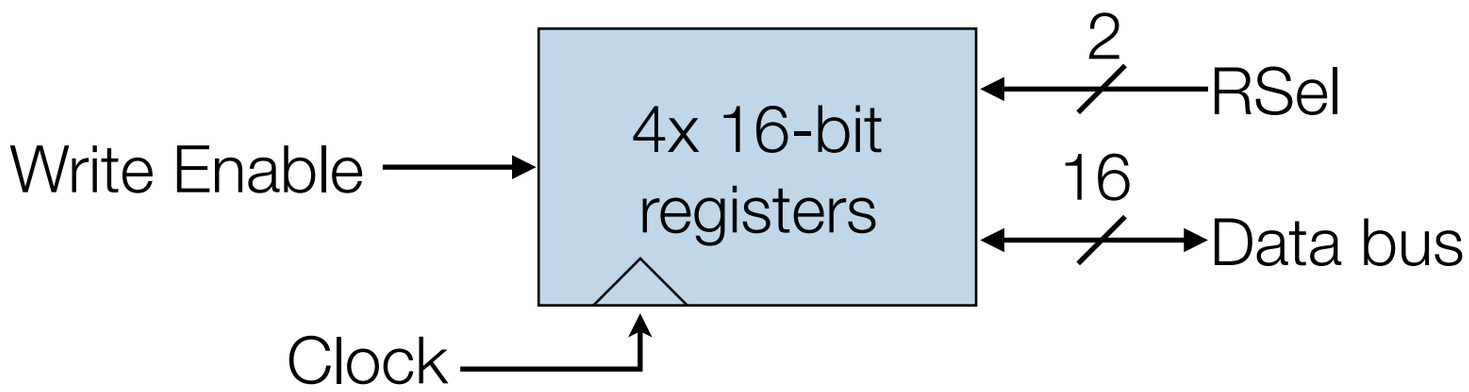
Register

- Composed by a set of flip-flips to store multiple bits
  - Categorized by number of stored bits
  - Common clock and enable lines, to latch bits simultaneously
- Often have an asynchronous reset line, that forces all stored bits to 0



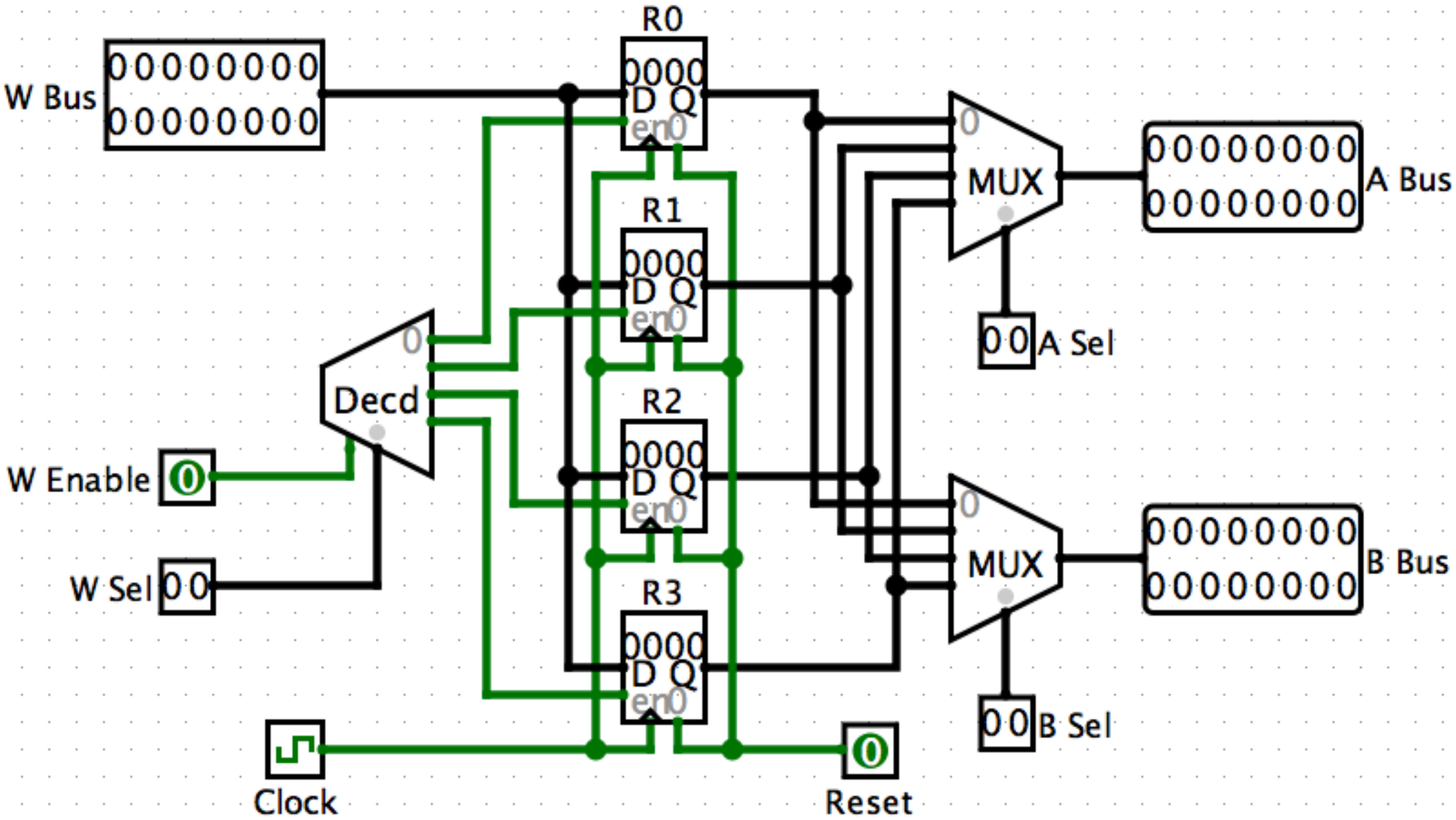
Register File

- Multiple registers combined into a single component
  - In simplest case, a single data bus for both reading and writing
  - Also **need** a clock, write-enable, and register select inputs
    - Clock only matters when write-enable is true
- Example register file with 4 registers, 16-bits each:



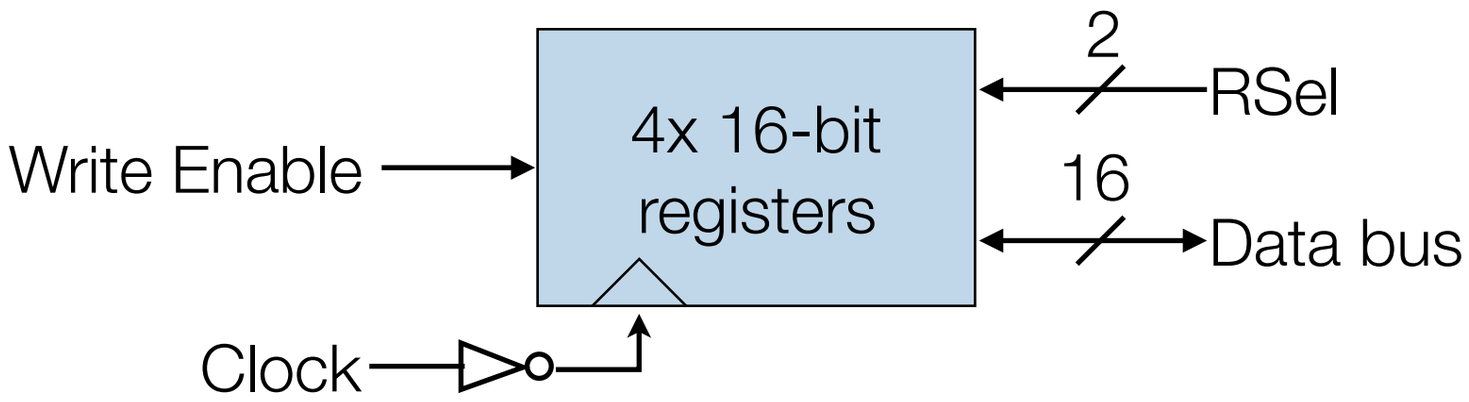
Multi-Bus Register File

- Fancier register files have two independent output buses (“A” and “B” buses) and a separate input bus (called either “C” or “W”) (a so-called 3-bus design)



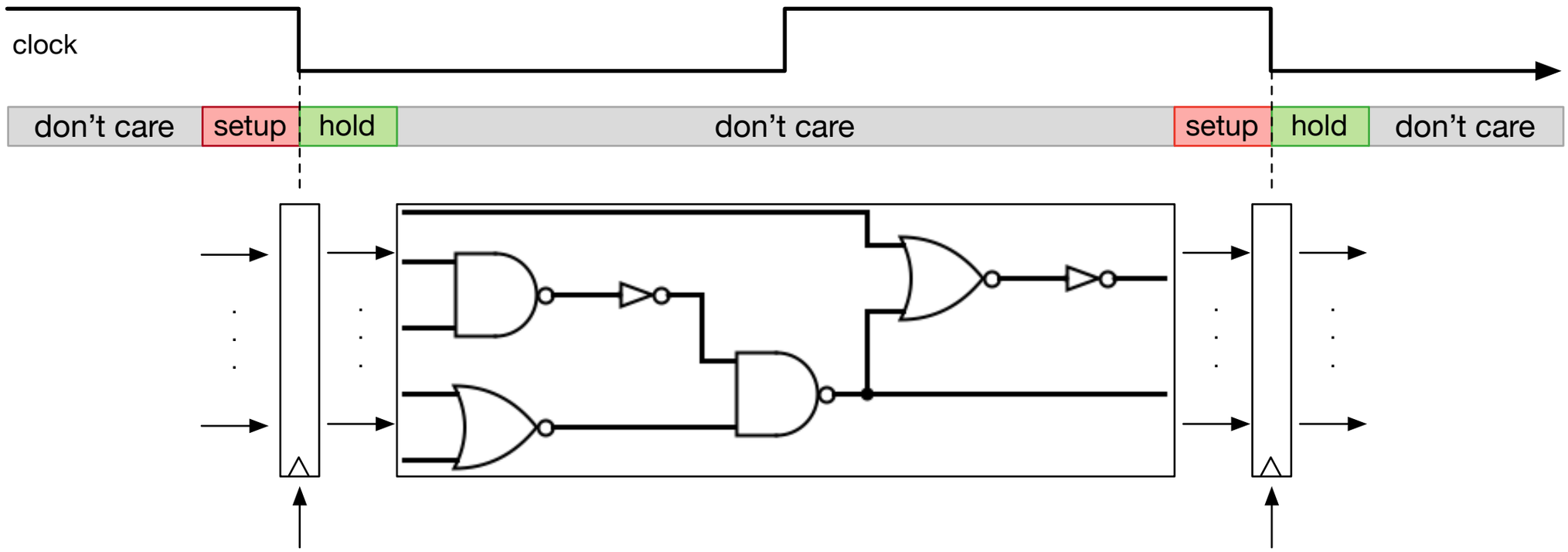
Clocking Methodology

- Data are changed during clock cycles, between clock edges
  - Input from state elements goes through combinatorial logic to create new output, which is then stored in some output state element
  - If same element is read and updated on **same clock edge**, then processing could involve old, new, or a mix of values
- Can achieve a read and write on the same clock cycle if the read occurs on a leading-edge and the write on a falling-edge, or via master/slave D flip-flops



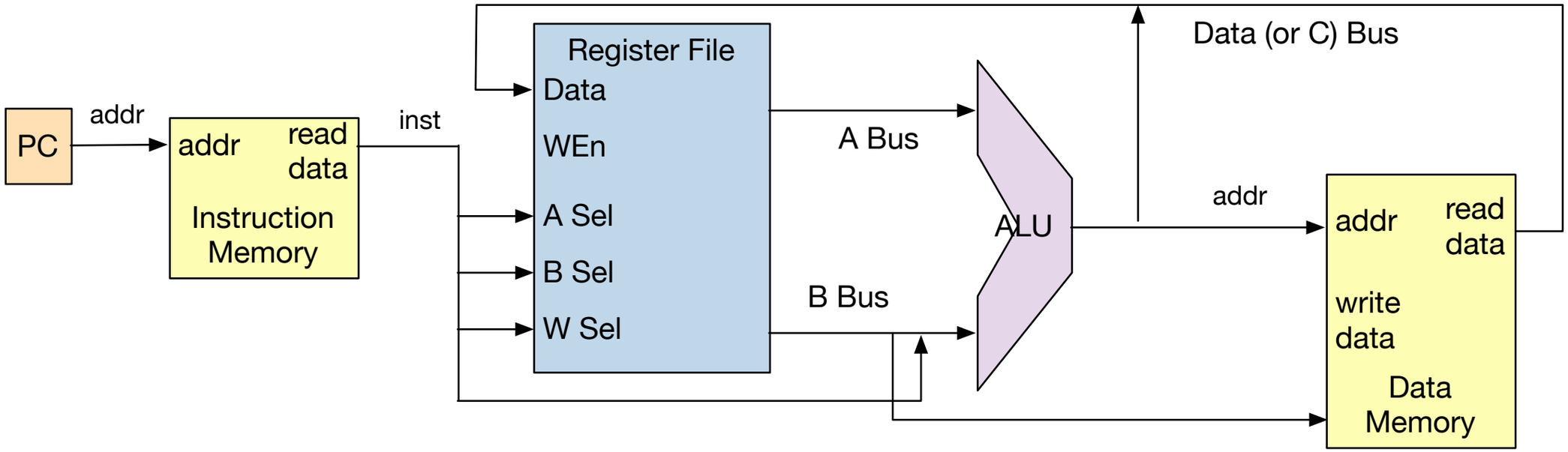
Clocking Methodology

- Longest propagation delay determines clock period
- Cycle Time = Hold + Longest Delay Path + Setup + Clock Skew
  - Hold and Setup times are needed to ensure voltages are stable



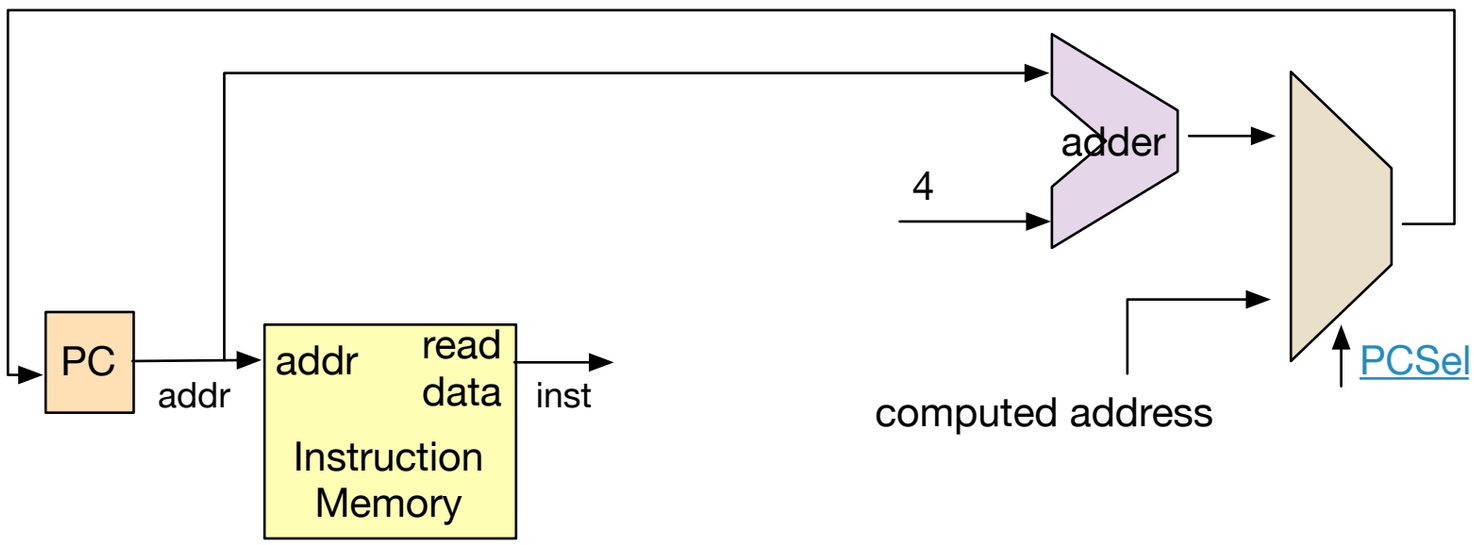
Simplified 3-Bus Datapath Assembly

- Datapath needs to support fetching, decoding, execution, and writing



Instruction Fetch

- Program counter (PC) is a register like any other register
  - Can be directly set by software on x86-64, but not on ARMv8-A
- **`IR`** ← InstrMem[**`PC`**]
- If branching, **`PC`** ← *computed address* else **`PC`** ← **`PC`** + 4 (for ARMv8-A)



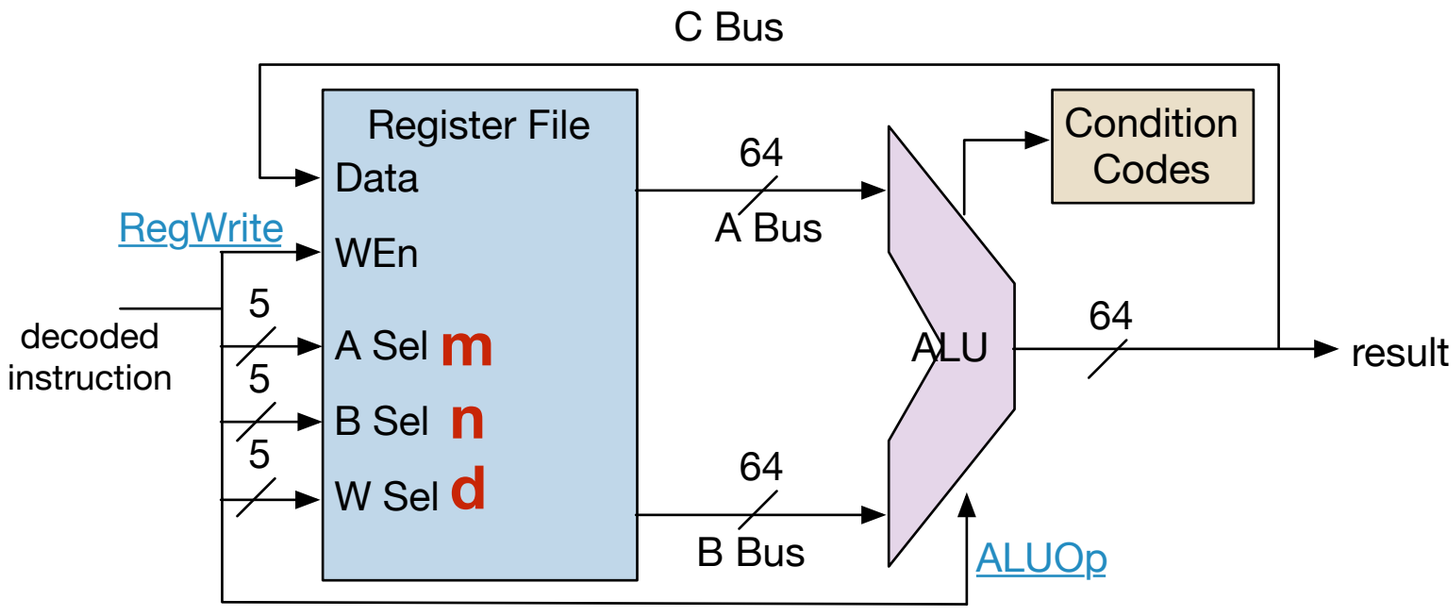
Executing Register-Register Instructions

- Many instructions are of the form **`X[d]`** ← **`X[m]`** *op* **`X[n]`**, where **m**, **n**, and **d** come from decoding the instruction

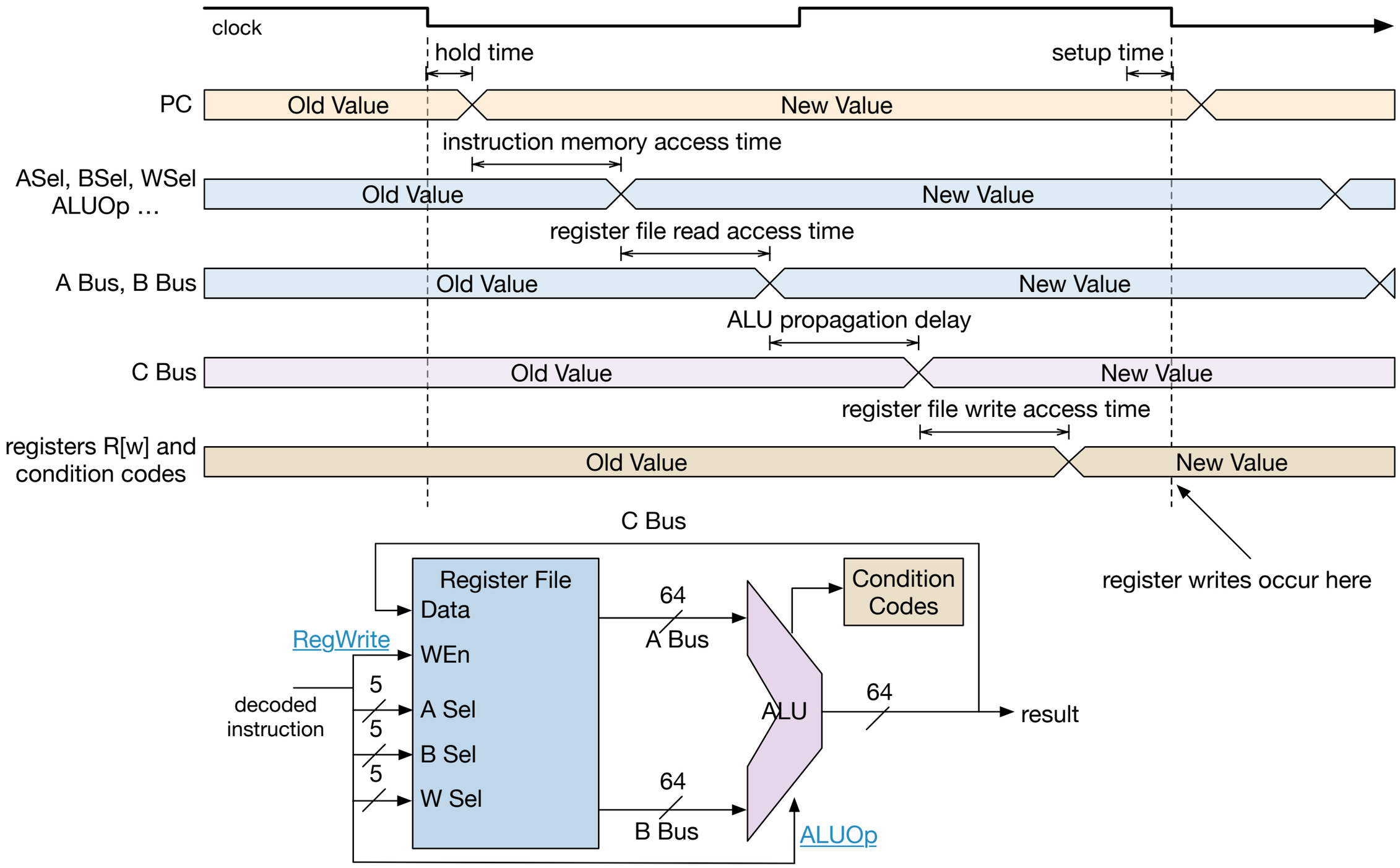
- **`ALUOp`** and **`WEn`** are control logic that also come from instruction decoding

R-Type	opcode	R_m	shamt	R_n	R_d
	11 bits	5 bits	6 bits	5 bits	5 bits

`add` (shifted register):
`X[d]` ← **`X[m]`** + **`X[n]`**



Register-Register Timing

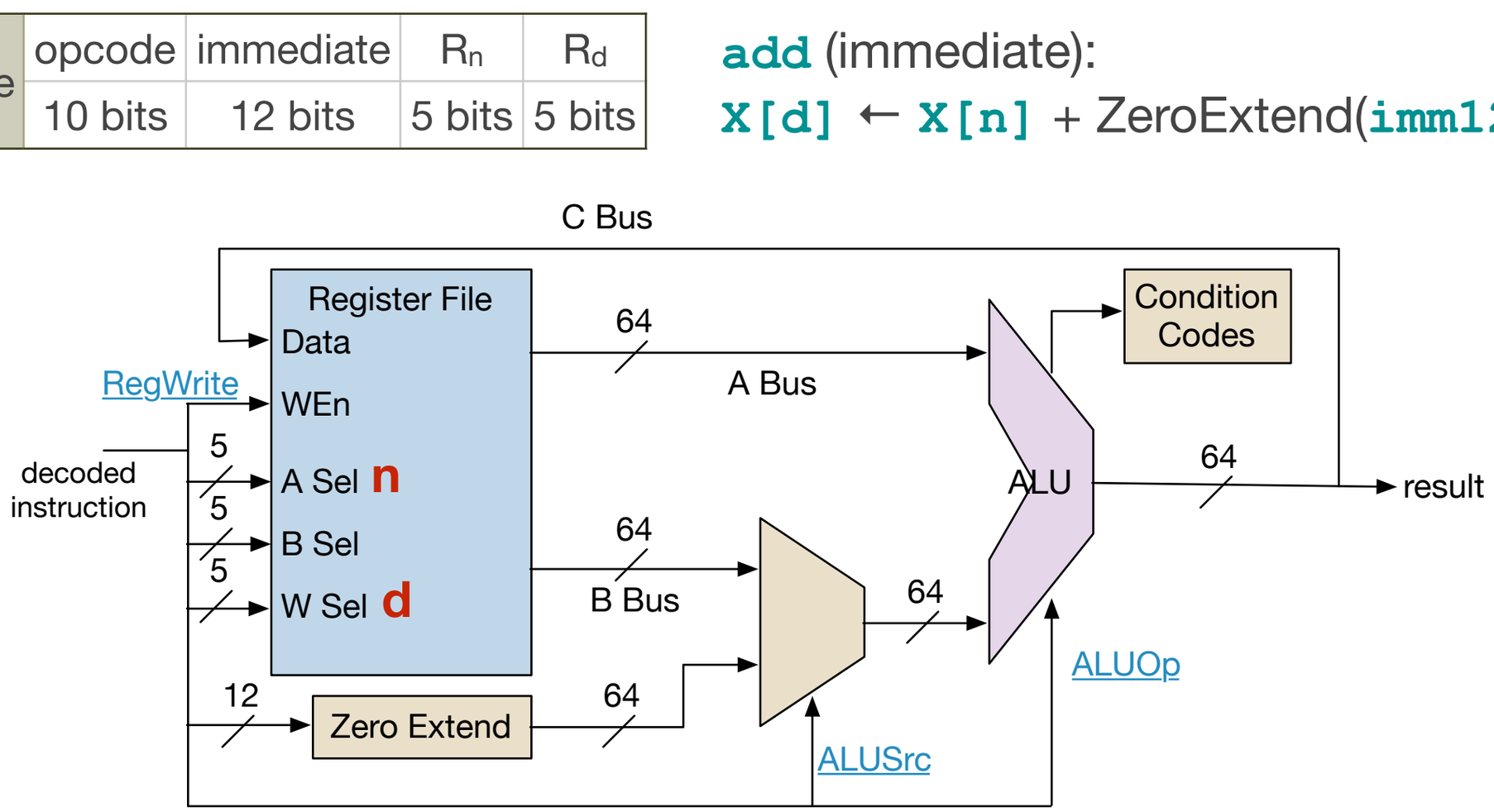


Executing Immediate Instructions

- For immediate instructions, the B Bus has an extended immediate value instead of a register value

I-Type	opcode	immediate	R_n	R_d
	10 bits	12 bits	5 bits	5 bits

`add` (immediate):
`X[d]` ← **`X[n]`** + ZeroExtend(**`imm12`**)

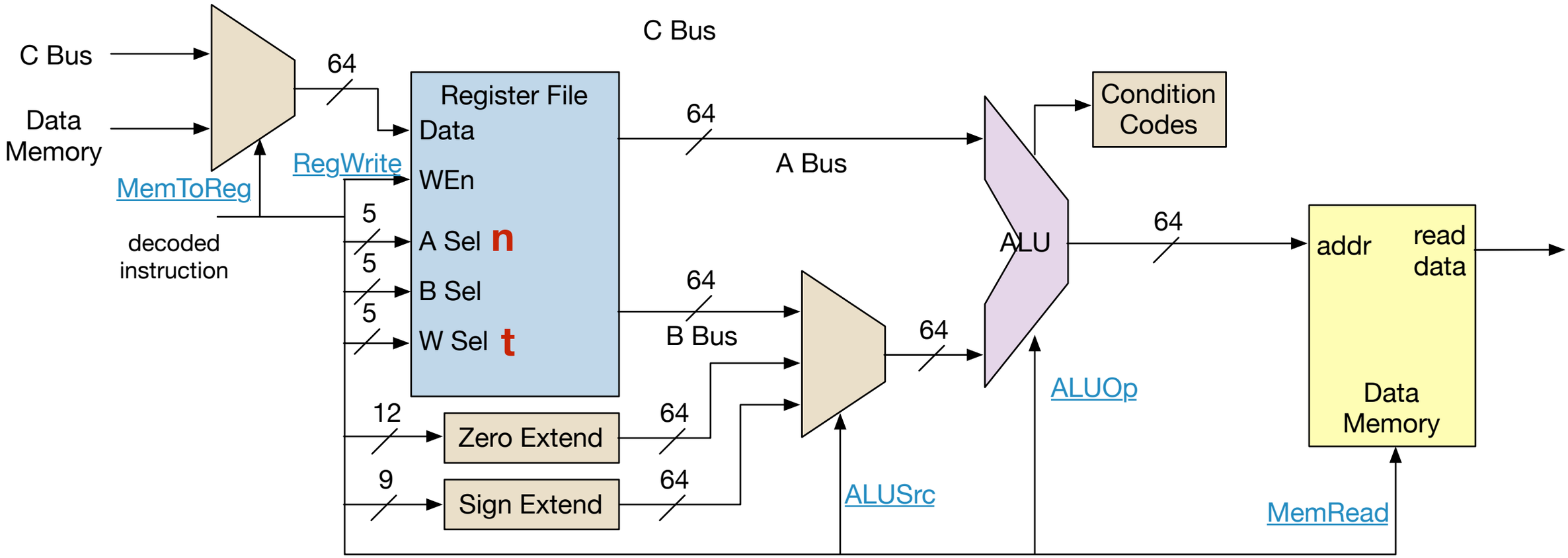


Executing Load Instructions

- Output of ALU is used as the address from which to read memory

D-Type	opcode	address	op2	R_n	R_t
	11 bits	9 bits	2 bits	5 bits	5 bits

`ldur`:
`X[t]` ← Mem[**`X[n]`** + SignExtend(**`imm9`**)]

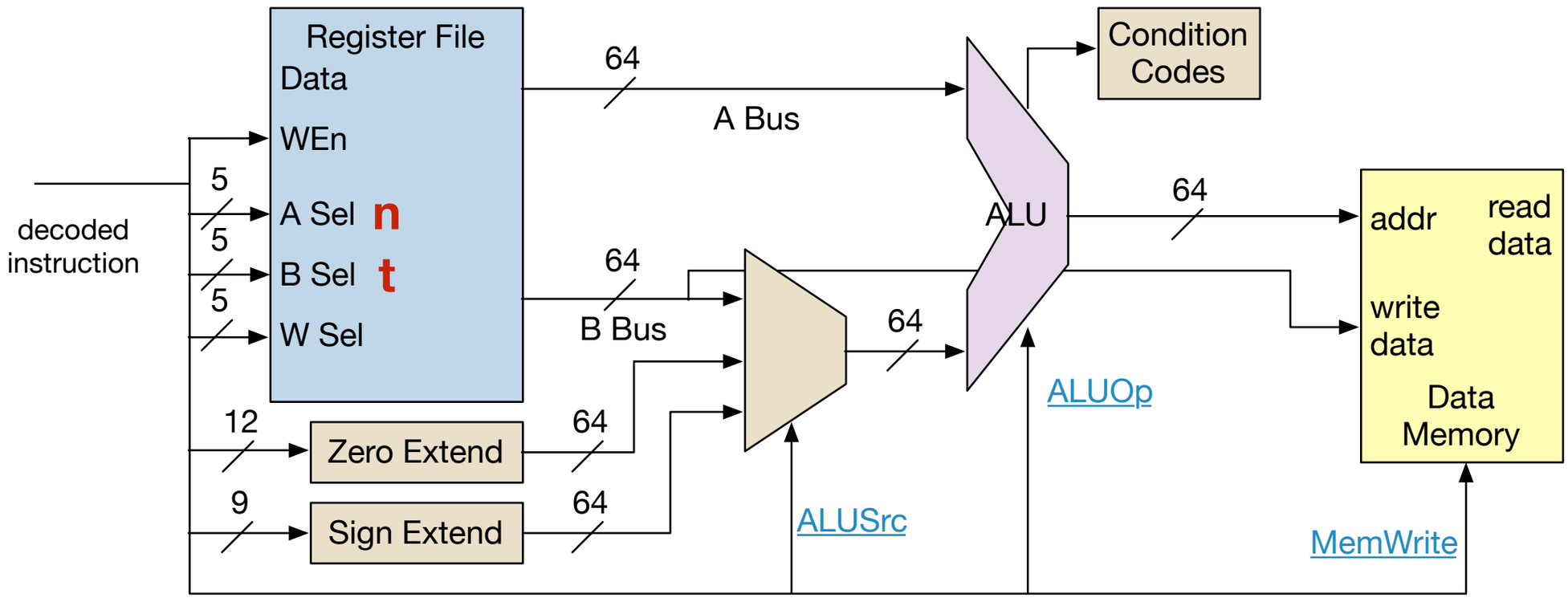


Executing Store Instructions

- Similar to a load, but also sets data memory via **`MemWrite`**

D-Type	opcode	address	op2	R_n	R_t
	11 bits	9 bits	2 bits	5 bits	5 bits

`stur`:
Mem[**`X[n]`** + SignExtend(**`imm9`**)] ← **`X[t]`**



Executing Branch Instructions

- Unconditional and conditional branching are similar, in that they set **`PCSel`**

B-Type	opcode	immediate
	6 bits	26 bits

CB-Type	opcode	immediate	R_t
	8 bits	19 bits	5 bits

`b`:

`PC` ← **`PC`** + (SignExtend(**`imm26`**)) × 4

`b.cond`:

if (cond) **`PC`** ← **`PC`** + (SignExtend(**`imm19`**)) × 4