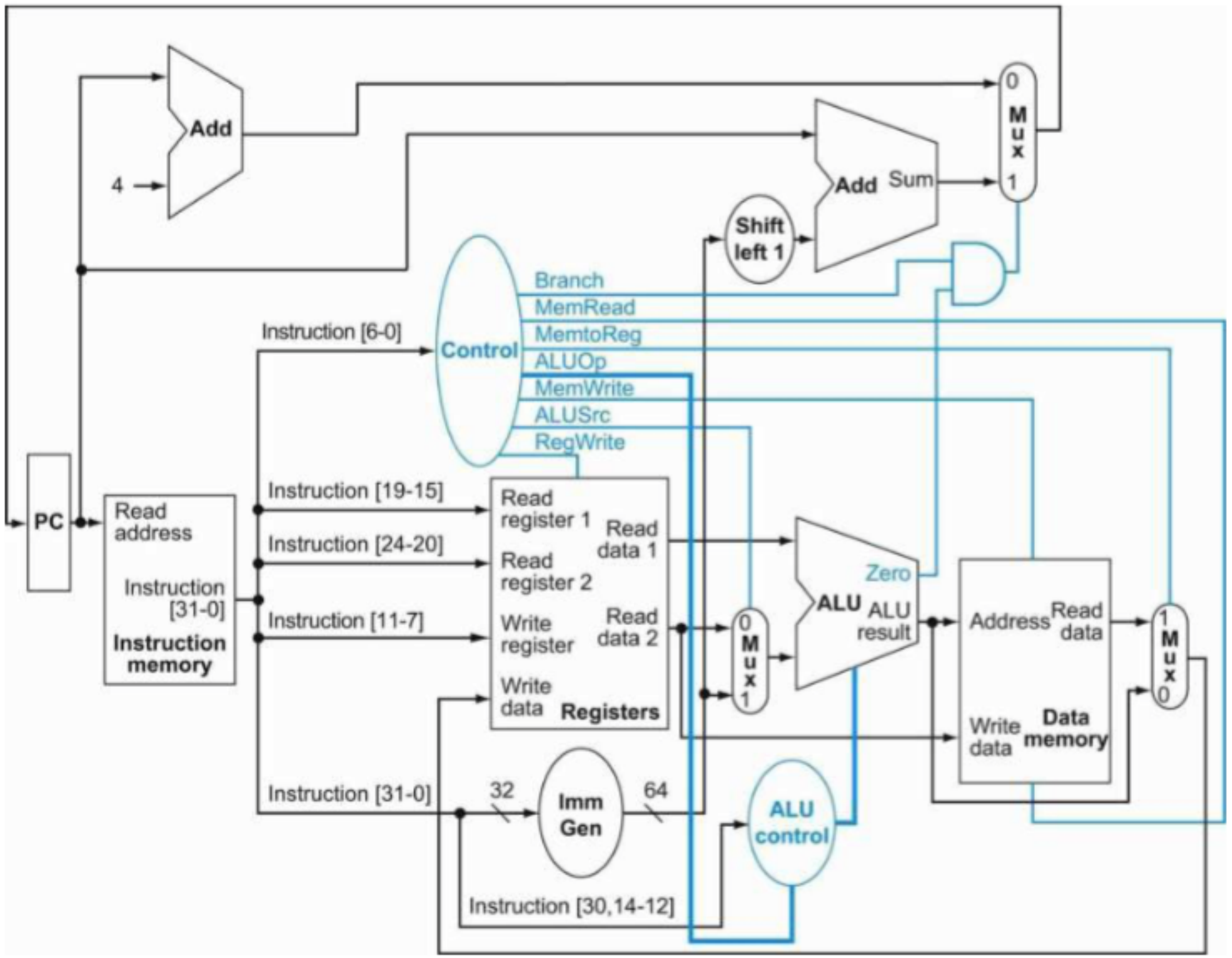
Lecture 6:
Building Single-Cycle Datapath and Control Unit


Add
4
Add
Sum
Shift left 1
0 Mux 1
Control
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [6-0]
Instruction [19-15]
Instruction [24-20]
Instruction [11-7]
Instruction [31-0]
PC
Read address
Instruction [31-0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0 Mux 1
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
1 Mux 0
32
Imm Gen
64
ALU control
Instruction [30,14-12]

How to Design a Processor: step-by-step

- 1. Analyze instruction set => datapath requirements
  - the meaning of each instruction is given by the *register transfers*
  - datapath must include storage element for ISA registers
  - datapath must support each register transfer
- 2. Select set of datapath components and establish clocking methodology
- 3. Assemble datapath meeting the requirements
- 4. Analyze implementation of each instruction to determine setting of control points that effect the register transfer.
- 5. Assemble the control logic

Step 1: The RISC-V Subset for today

- ADD and SUB
  - – add rd, rs1, rs2
  - – sub rd, rs1, rs2
- OR Immediate:
  - – ori rd, rs, imm12
- LOAD and STORE
  - – ld rd, imm12(rs1)
  - – sd rs2, imm12(rs1)
- BRANCH:
  - – beq rs1, rs2, imm12

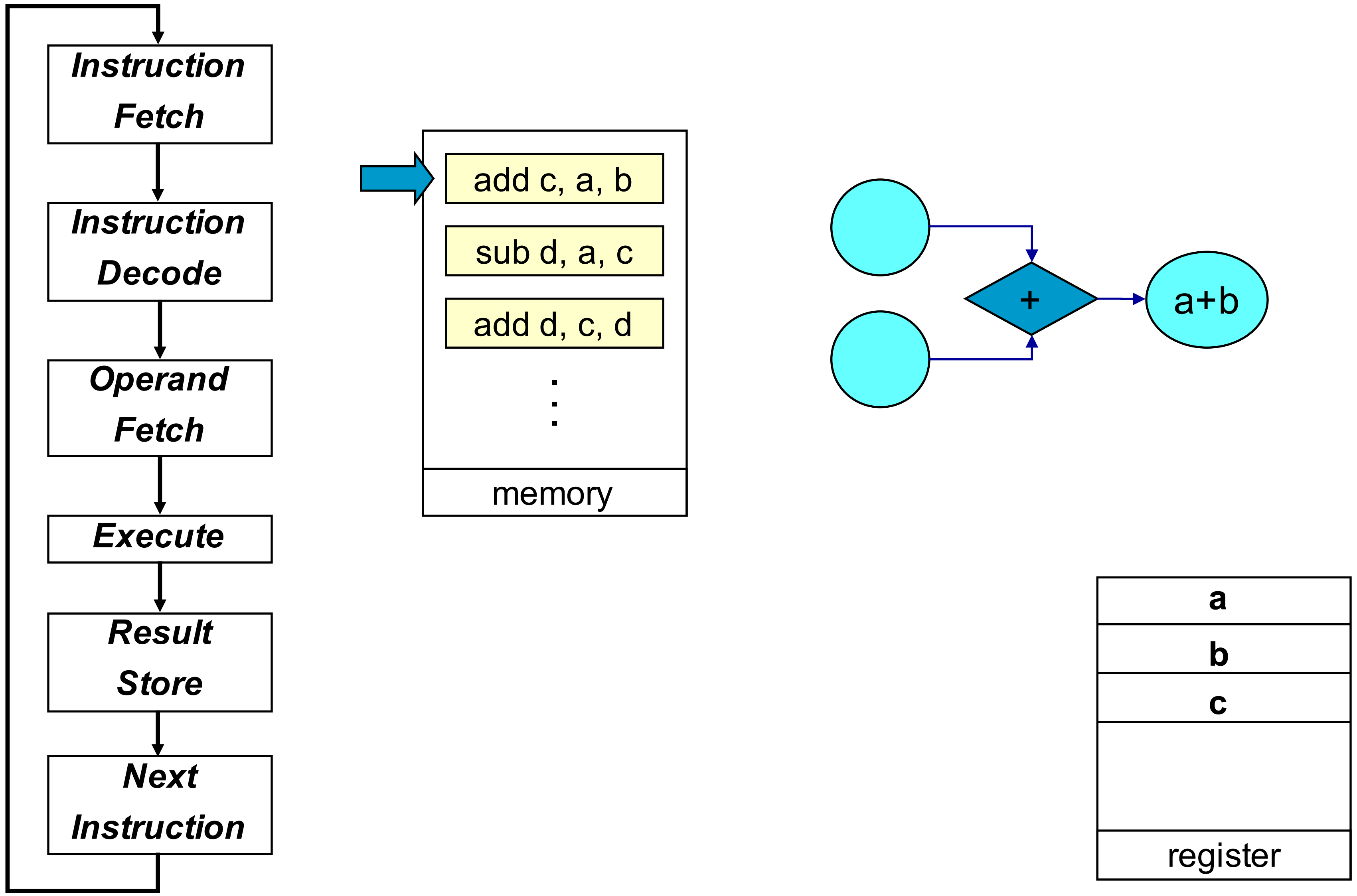
funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

	imm [10:5]	rs2	rs1	funct3	imm [4:1]		opcode
imm[12]						imm[11]	

Execution Flow



Instruction <=> Register Transfers

- RTL (Register Transfer Languages) gives the meaning of the instructions
- All start by fetching the instruction

inst	Register Transfers	
ADD	R[rd] <– R[rs1] + R[rs2];	PC <– PC + 4
SUB	R[rd] <– R[rs1] – R[rs2];	PC <– PC + 4
ORi	R[rd] <– R[rs] + sign_ext(Imm12);	PC <– PC + 4
LOAD	R[rd] <– MEM[R[rs1] + sign_ext(Imm12)];	PC <– PC + 4
STORE	MEM[R[rs1] + sign_ext(Imm16)] <– R[rs2];	PC <– PC + 4
BEQ	if (R[rs1] == R[rs2]) then PC <– PC + sign_ext(Imm12) else PC <– PC + 4	

Step 1: Requirements of the Instruction Set

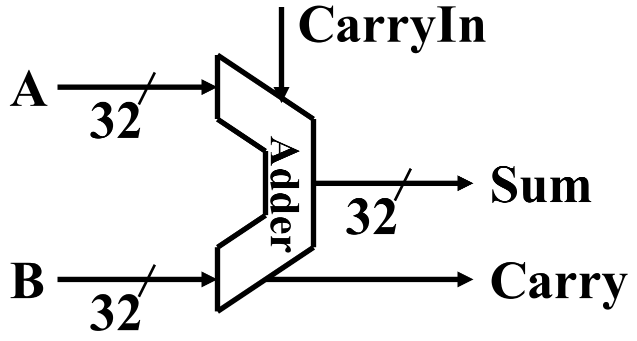
- Memory
  - instruction & data
- Registers (32 x 64)
  - read RS1
  - read RS2
  - Write RD
- PC
- Extender
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC

Step 2: Components of the Datapath

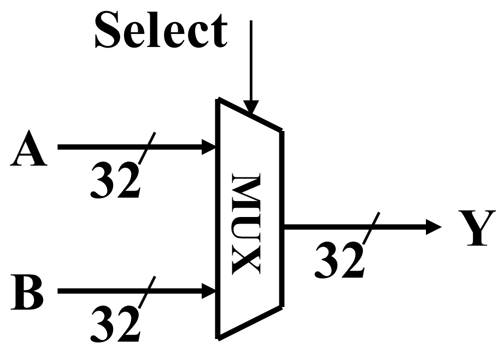
- Combinational Elements
  - The outputs only depend on the current inputs.
  - Example: ALU
- Storage Elements (state element)
  - The outputs depend on both their inputs and the contents of the internal state.
  - At least two inputs and one output
    - Inputs – input data and clock (clocking methodology)
    - Output – the value stored in a state element
  - Example: D Flip-Flop, register and memory

Combinational Logic Elements (Basic Building Blocks)

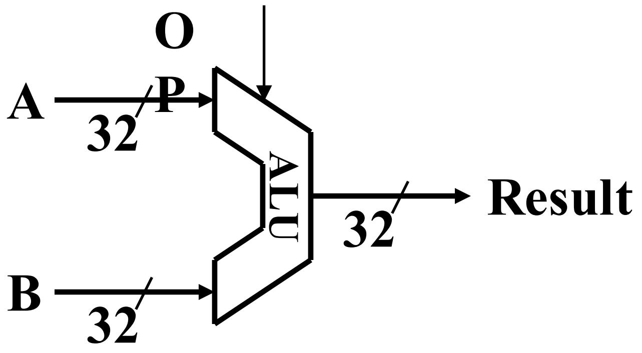
- Adder



- MUX

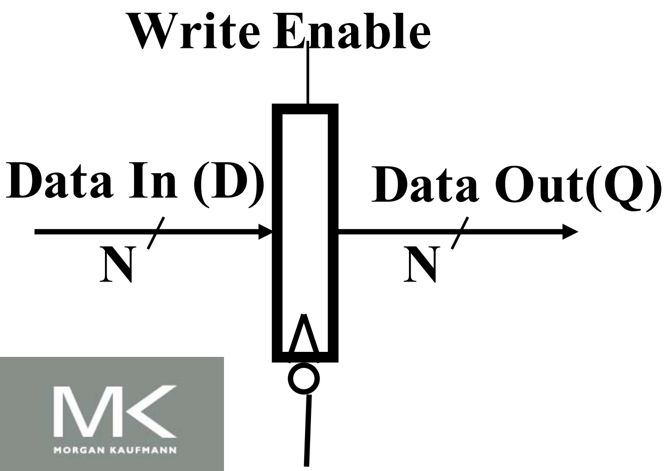


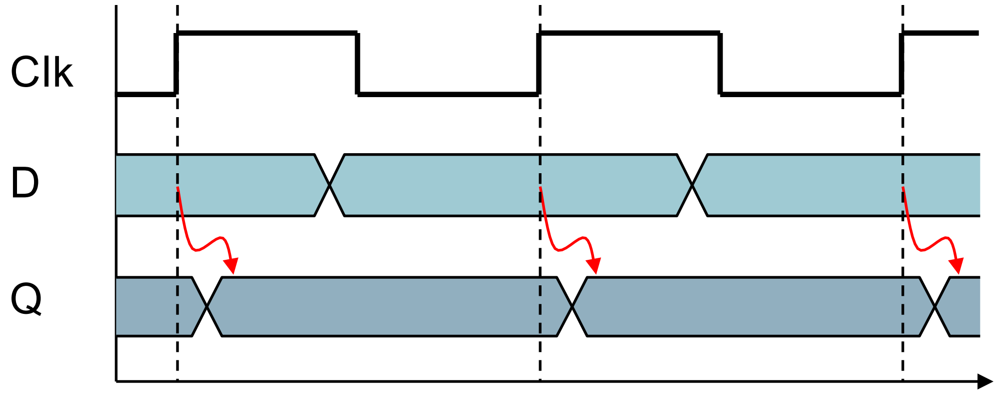
- ALU



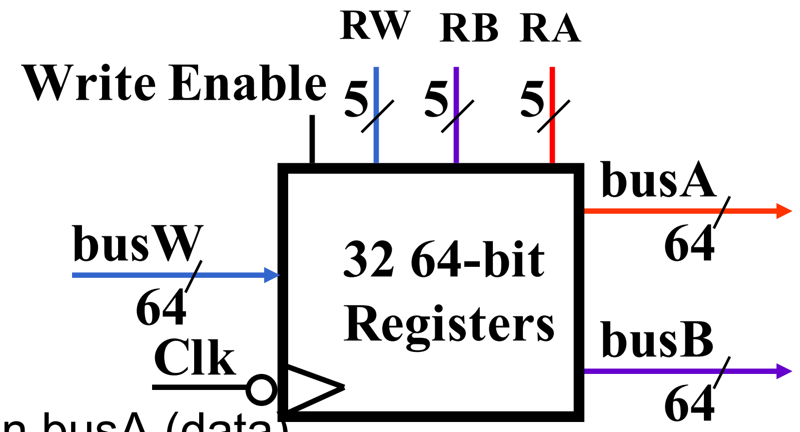
Storage Element: Register (Basic Building Block)

- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
    - Edge-triggered: update when Clk changes from 0 to 1
  - Write Enable:
    - Asserted -> update the register contents





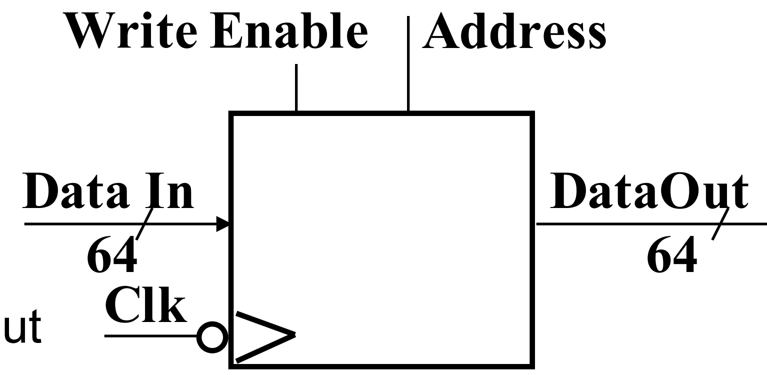
Storage Element: Register File



- Register File consists of 32 registers:
  - Two 64-bit output busses:
    busA and busB
  - One 64-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

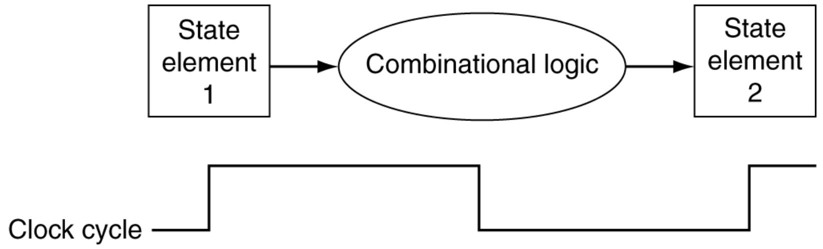
Storage Element: Memory

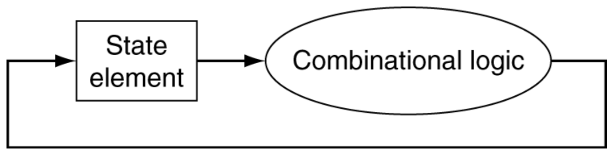
- Memory
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after "access time."



Clocking Methodology

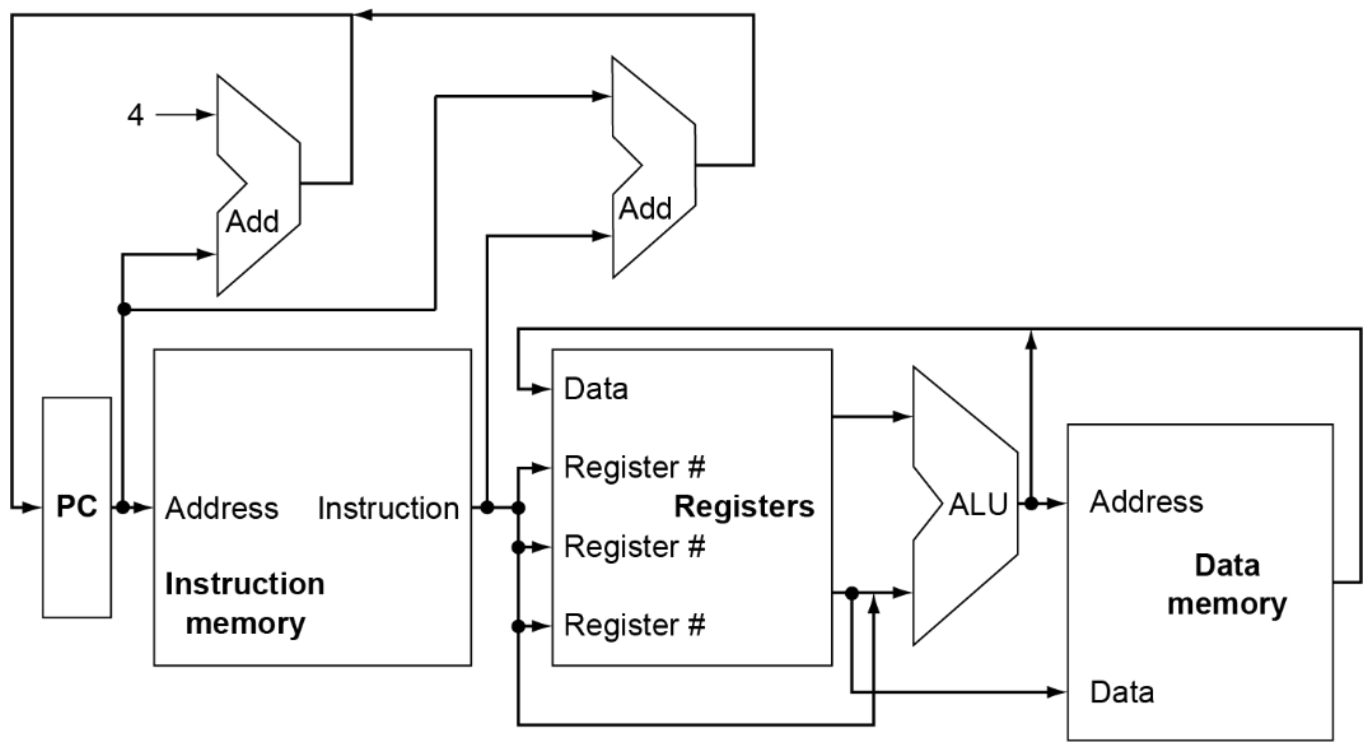
- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period





Step 3 : Assemble Datapath

- Register Transfer Requirements –> Datapath Assembly
  - Instruction Fetch
  - Read Operands and Execute Operation
  - Memory Read/Write
  - Register Update



3a: Instruction Fetch Unit

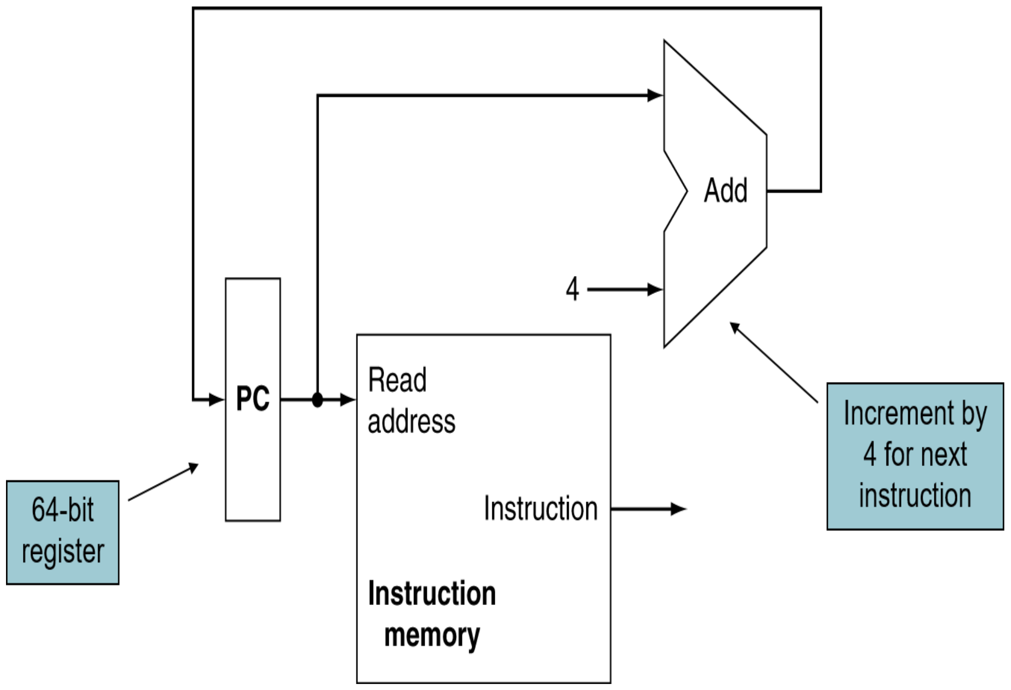
- Instruction fetch unit: common operations
  - Fetch the instruction: `mem[PC]`
  - Update the program counter:

    Sequential code

    ```
    PC <- PC + 4
    ```

    Branch and Jump (later)

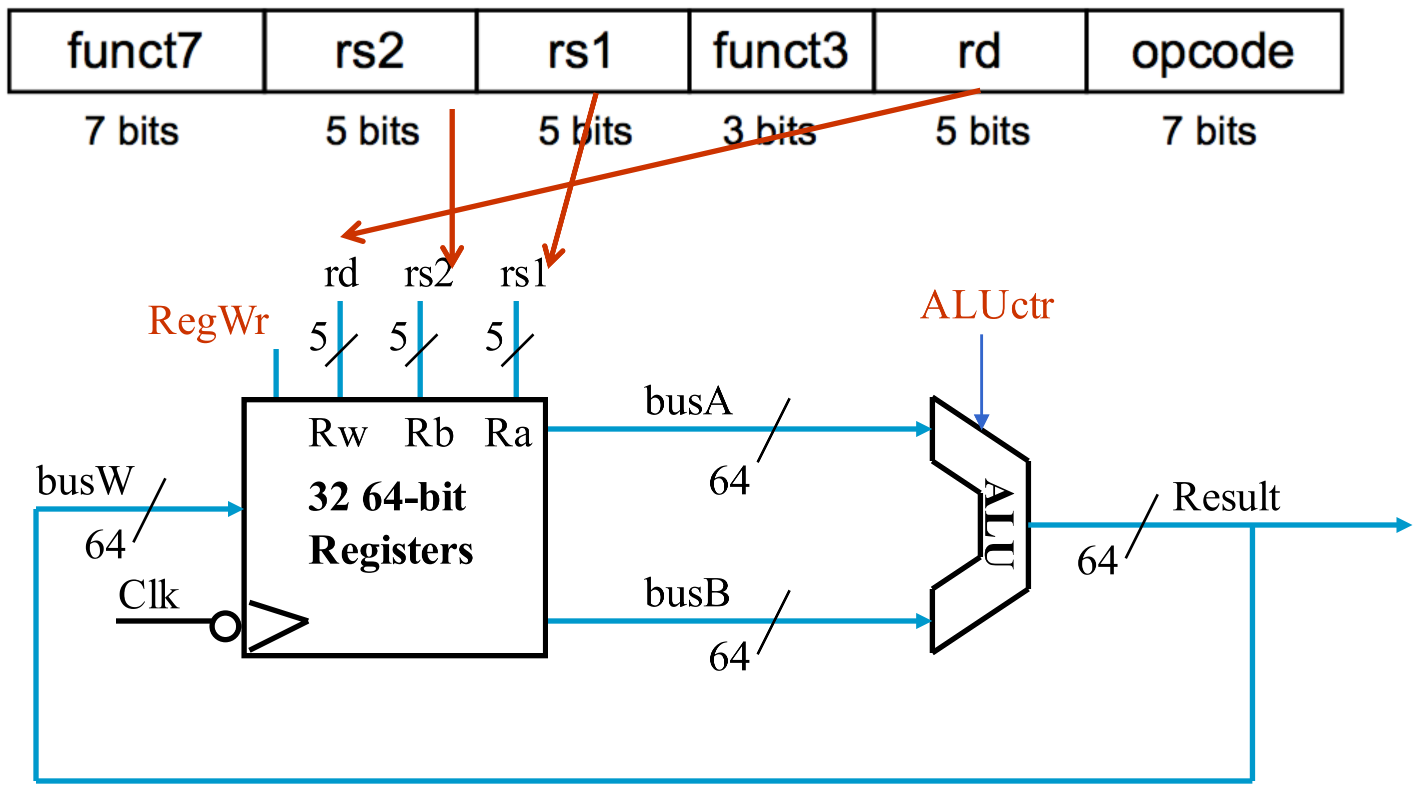
    ```
    PC <- Target addr.
    ```



3b: Add & Subtract

R[rd] <- R[rs1] op R[rs2]   Example: add   rd, rs1, rs2

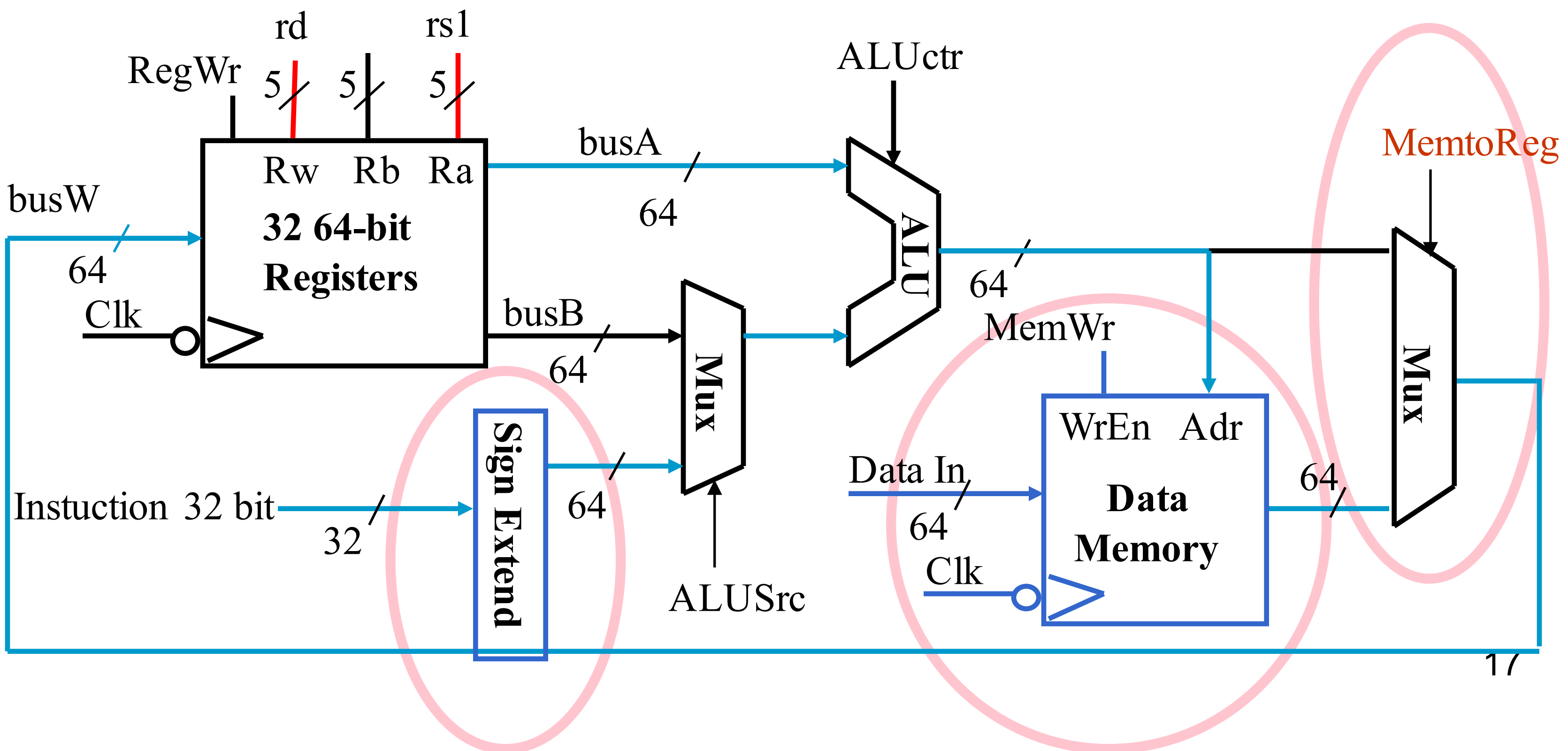
- Ra, Rb, and Rw come from instruction's rs1, rs2, and rd fields
- ALUctr and RegWr: control logic after decoding the instruction



3c: Load Operations

R[rd] <- Mem[R[rs1] + SignExt[imm12]] # ld rd, imm12(rs1)

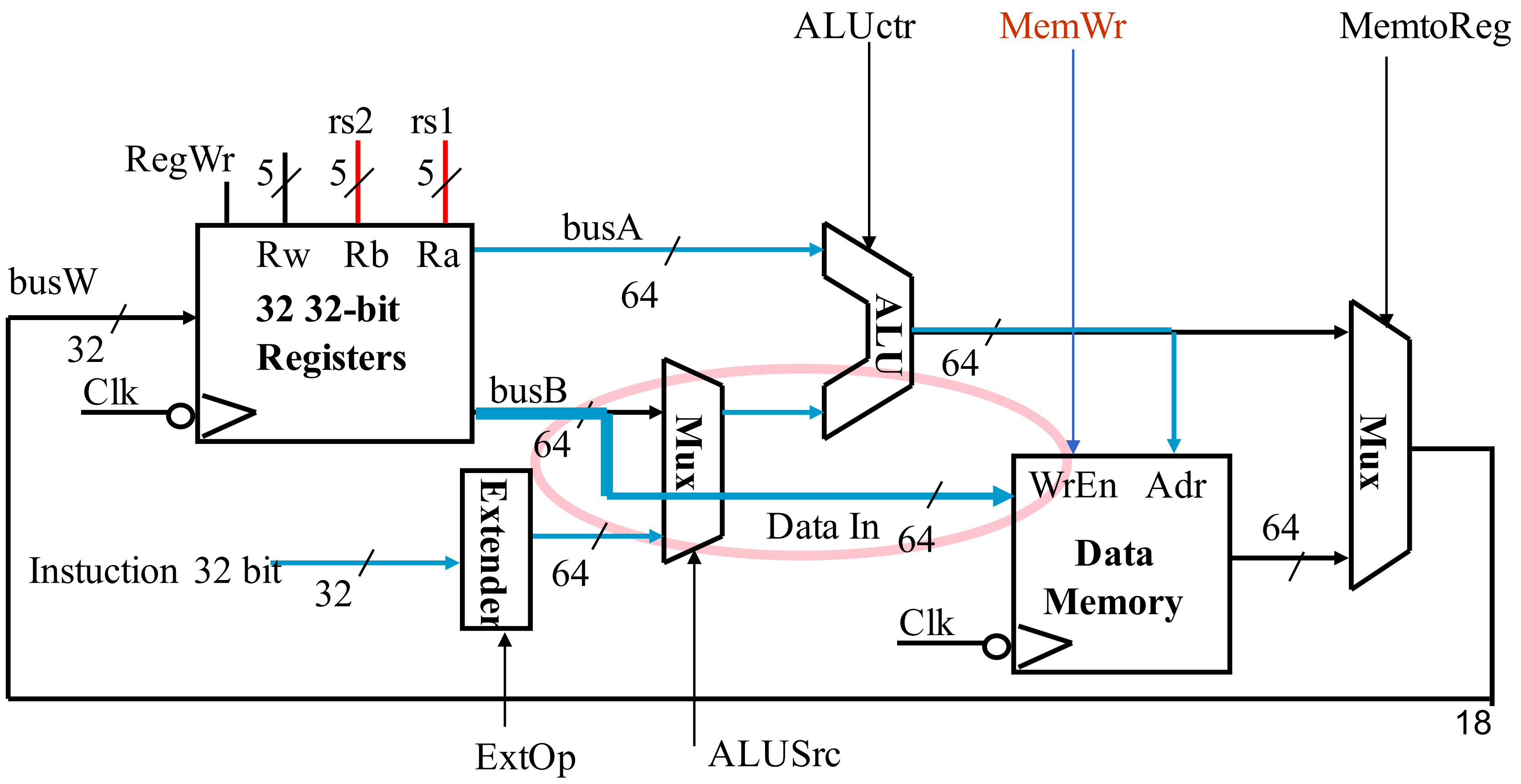
immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits



3d: Store Operations

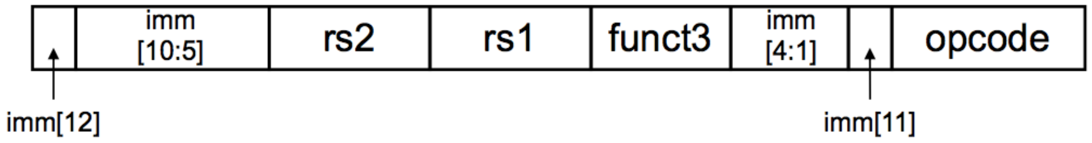
Mem[R[rs1] + SignExt[imm12]] <- R[rs2]   #sd   rs2, imm12(rs1)

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

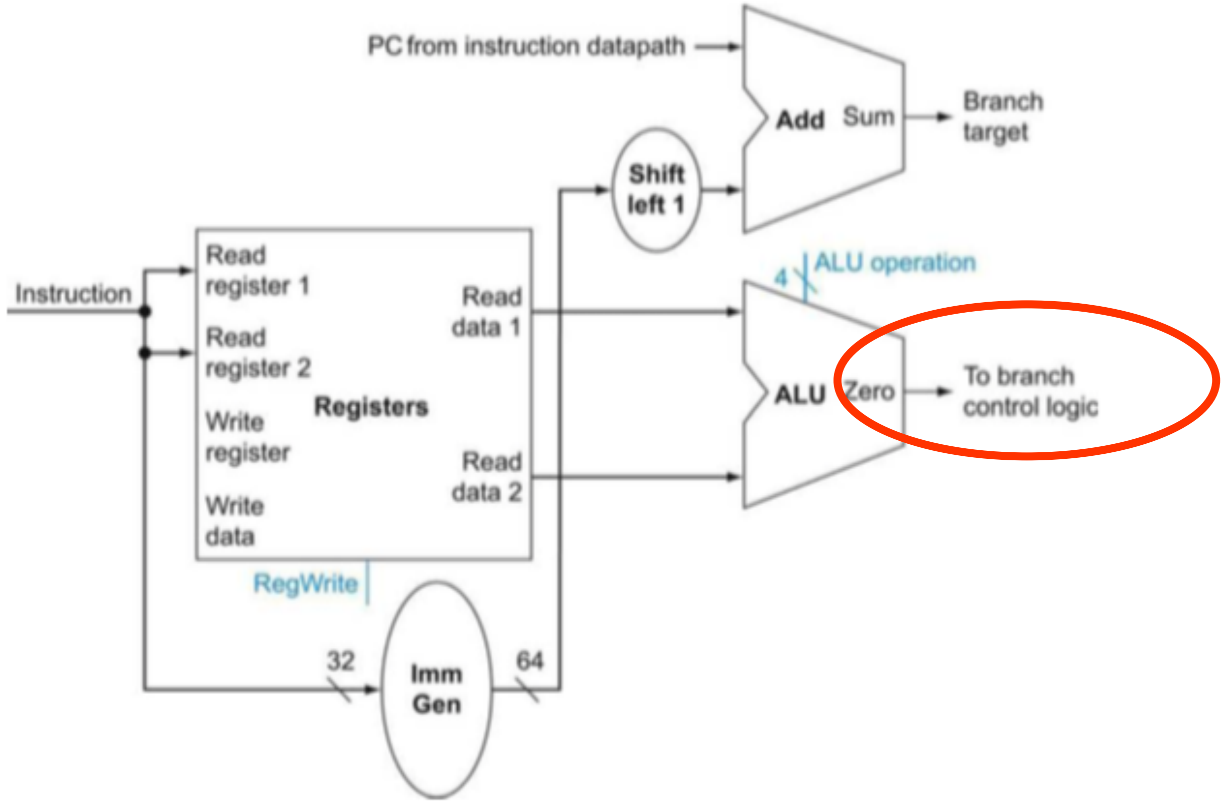


3e: Branch Operations

beq   rs1, rs2, imm12

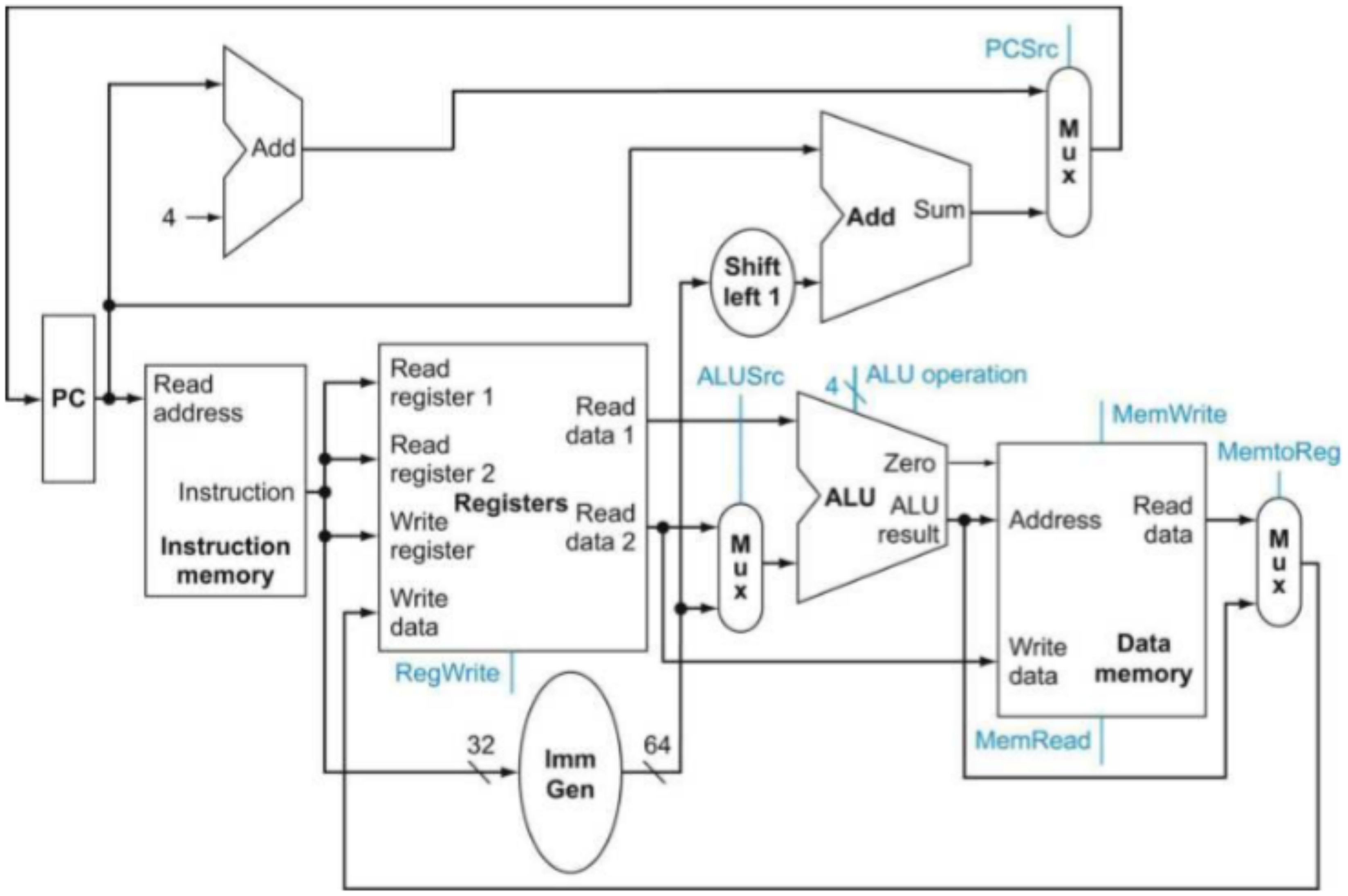


```
if (rs = = rt)
        PC  <-  PC + ( SignExt(imm12) x 2 )
Else
        PC  <-  PC + 4
```

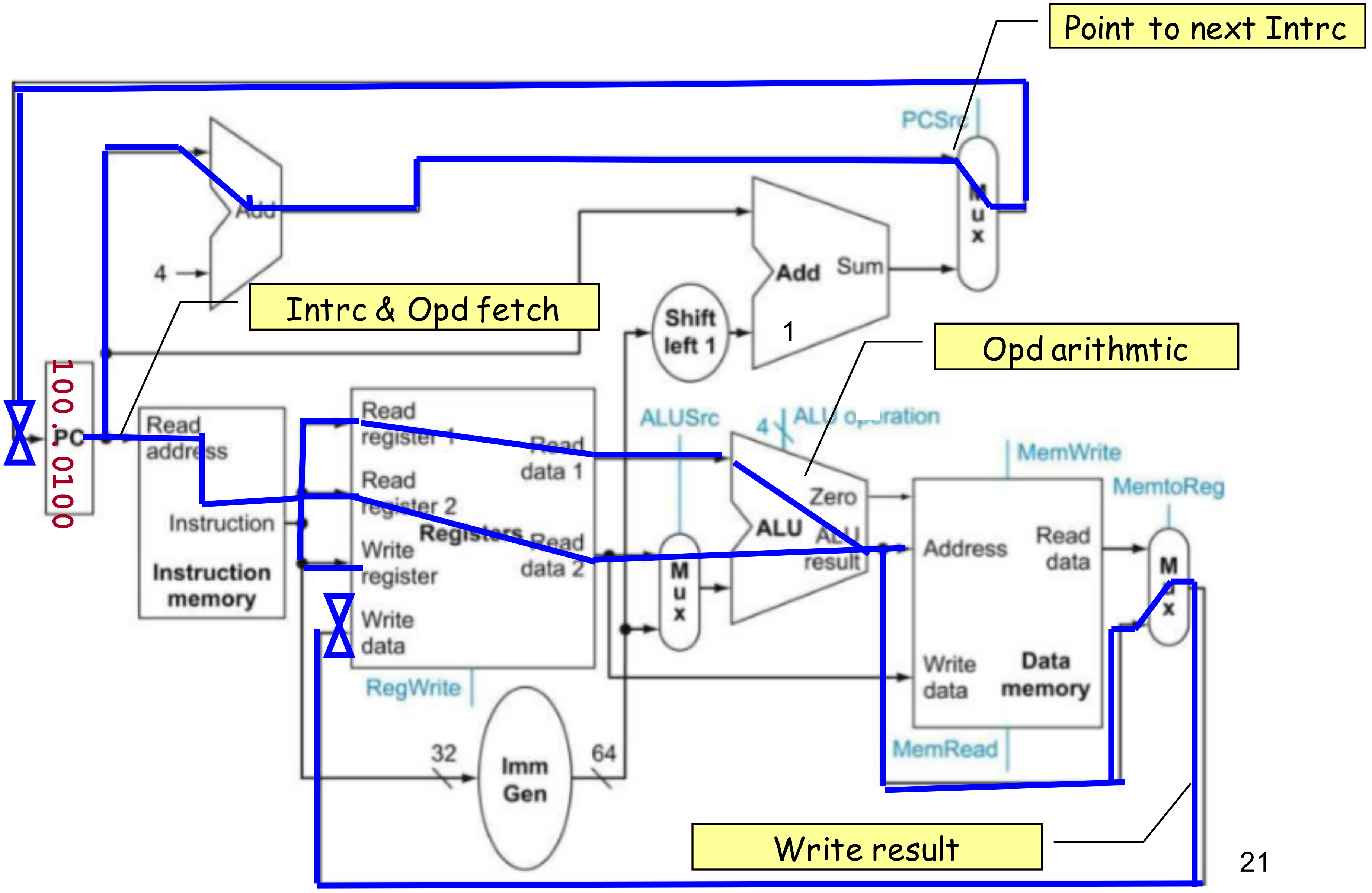


Single-Cycle Datapath

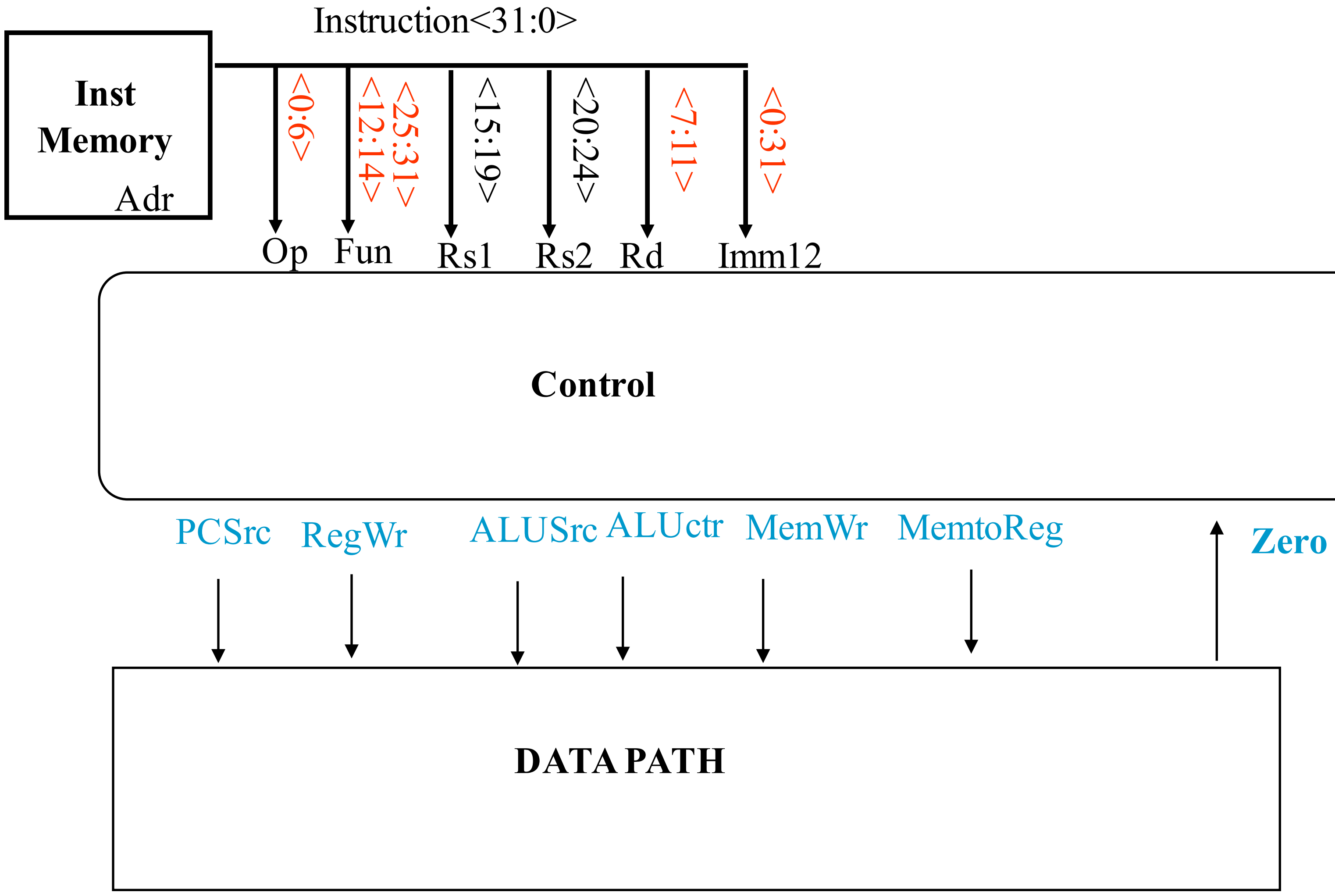
- Attempt to execute all instructions in one clock-cycle.



Data Flow of *add* Instruction



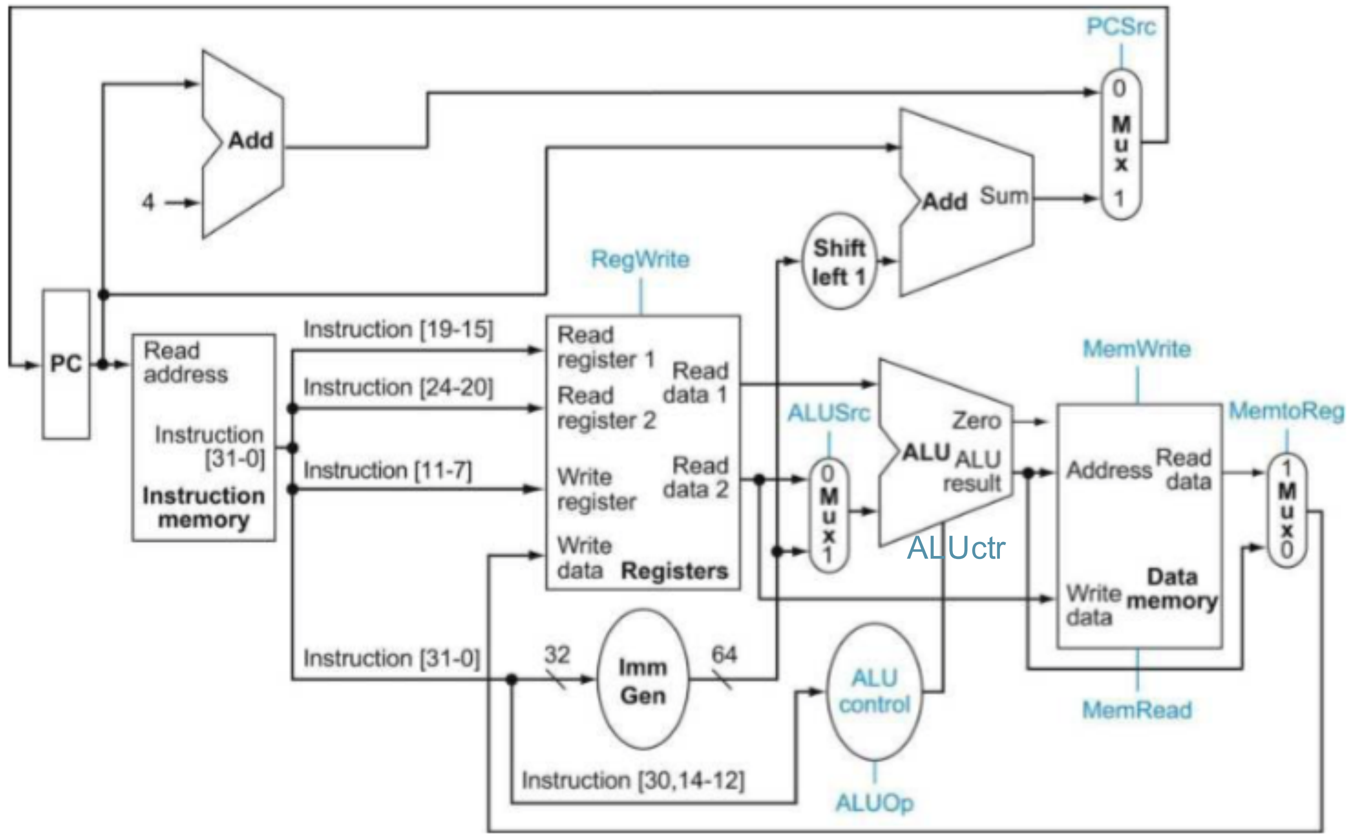
Step 4: Given Datapath: RTL -> Control



- Rs1, Rs2, Rd and Imed12 hardwired into datapath

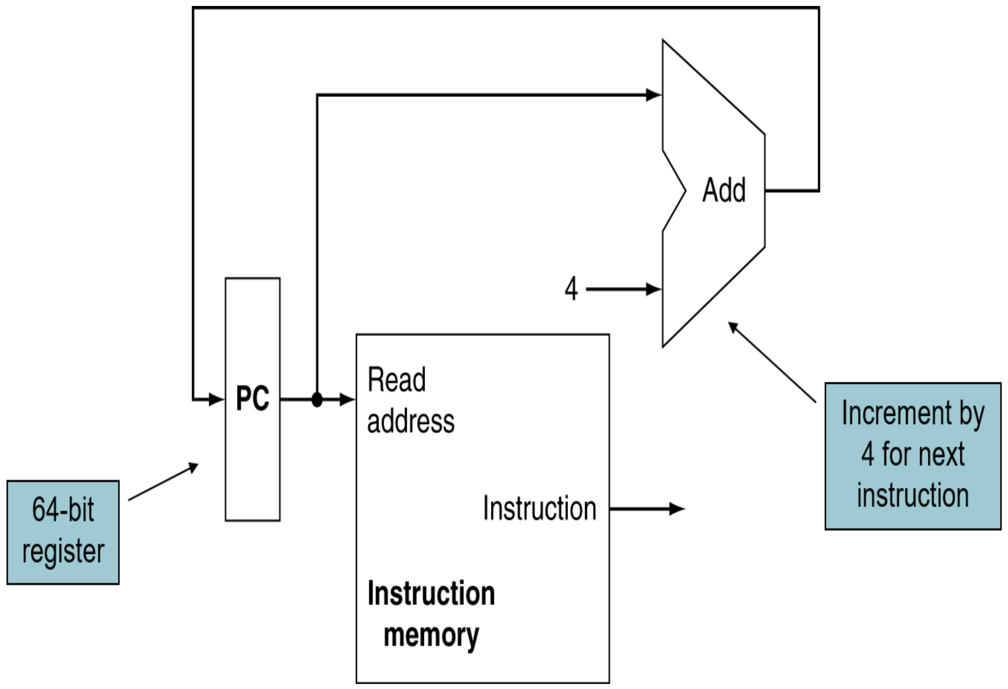
Meaning of Control Signals

- MemWr: write memory
- MemRd: read memory
- MemtoReg: 0 => ALU output 1 => Mem
- RegWr: write dest register
- ALUsrc: 0=> regB; 1=>immed
- ALUctr: "add", "sub", "or","and"
- PCSrc: 0=> PC = PC + 4; 1=> PC = branch target address
-



Examine control signals: **Add**

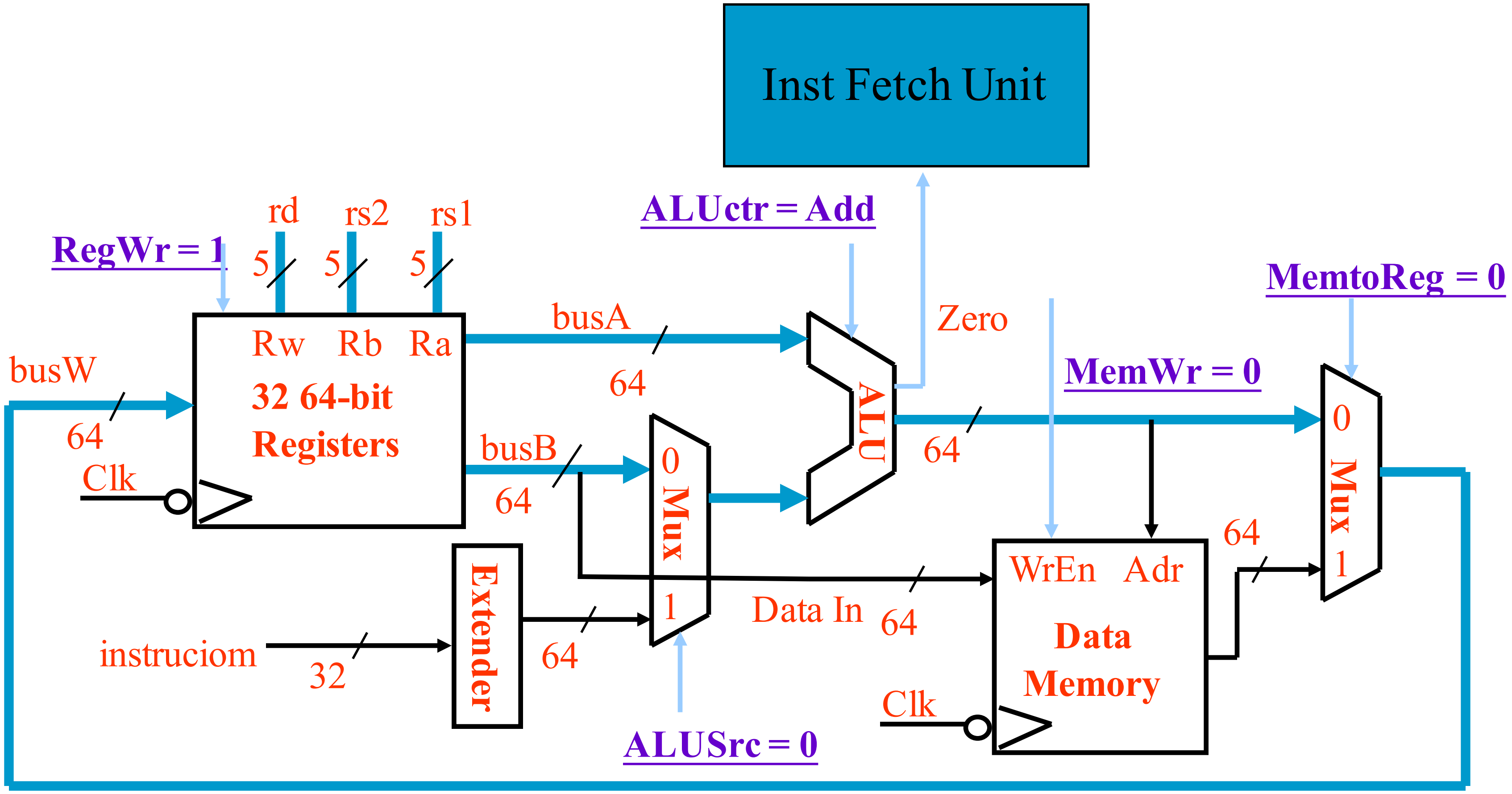
- Fetch the instruction from Instruction memory: Instruction <- mem[PC]
  - This is the same for all instructions



The Single Cycle Datapath during **Add**

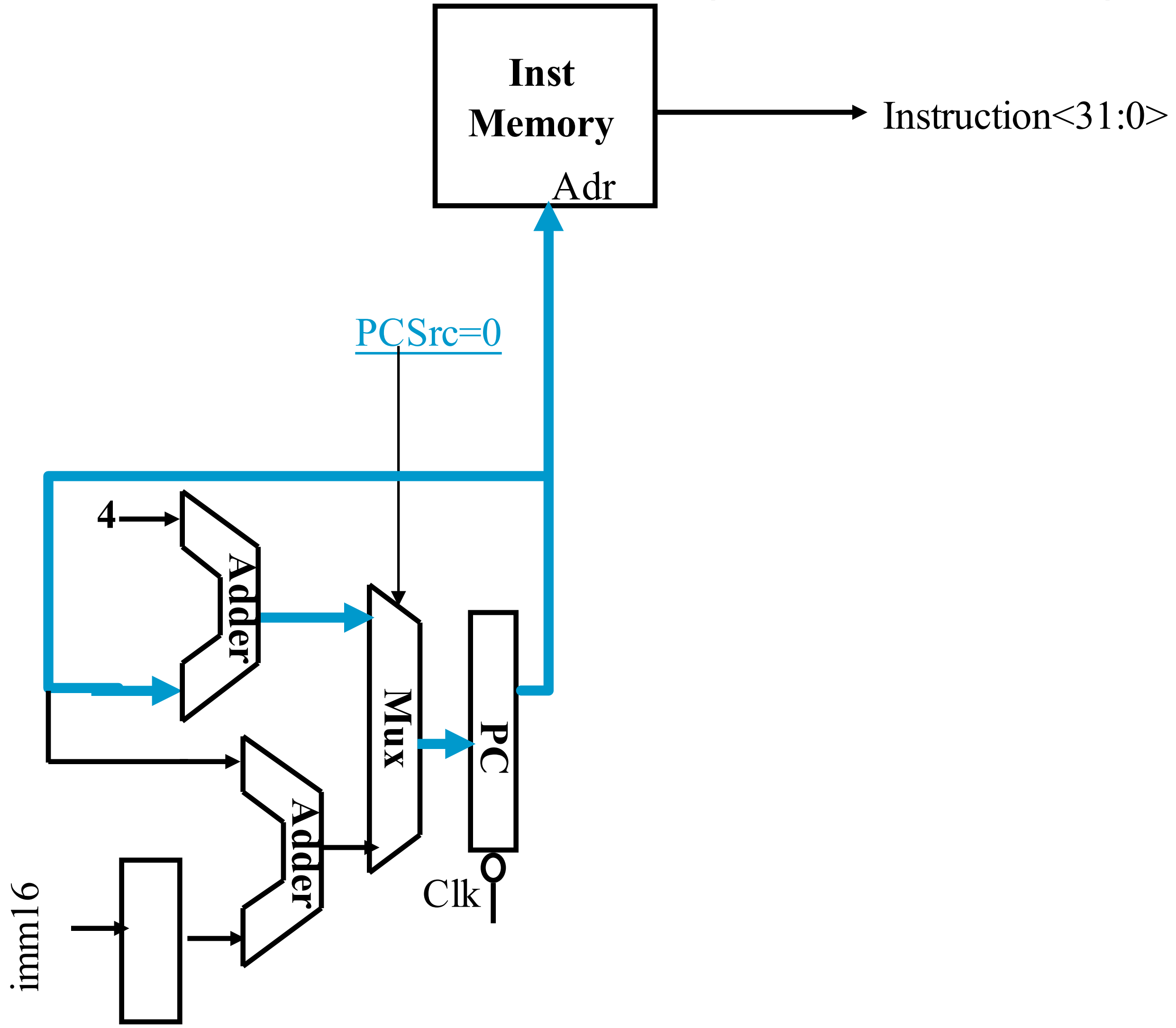
funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- R[rd] <- R[rs1] + R[rs2]



Instruction Fetch Unit at the End of Add

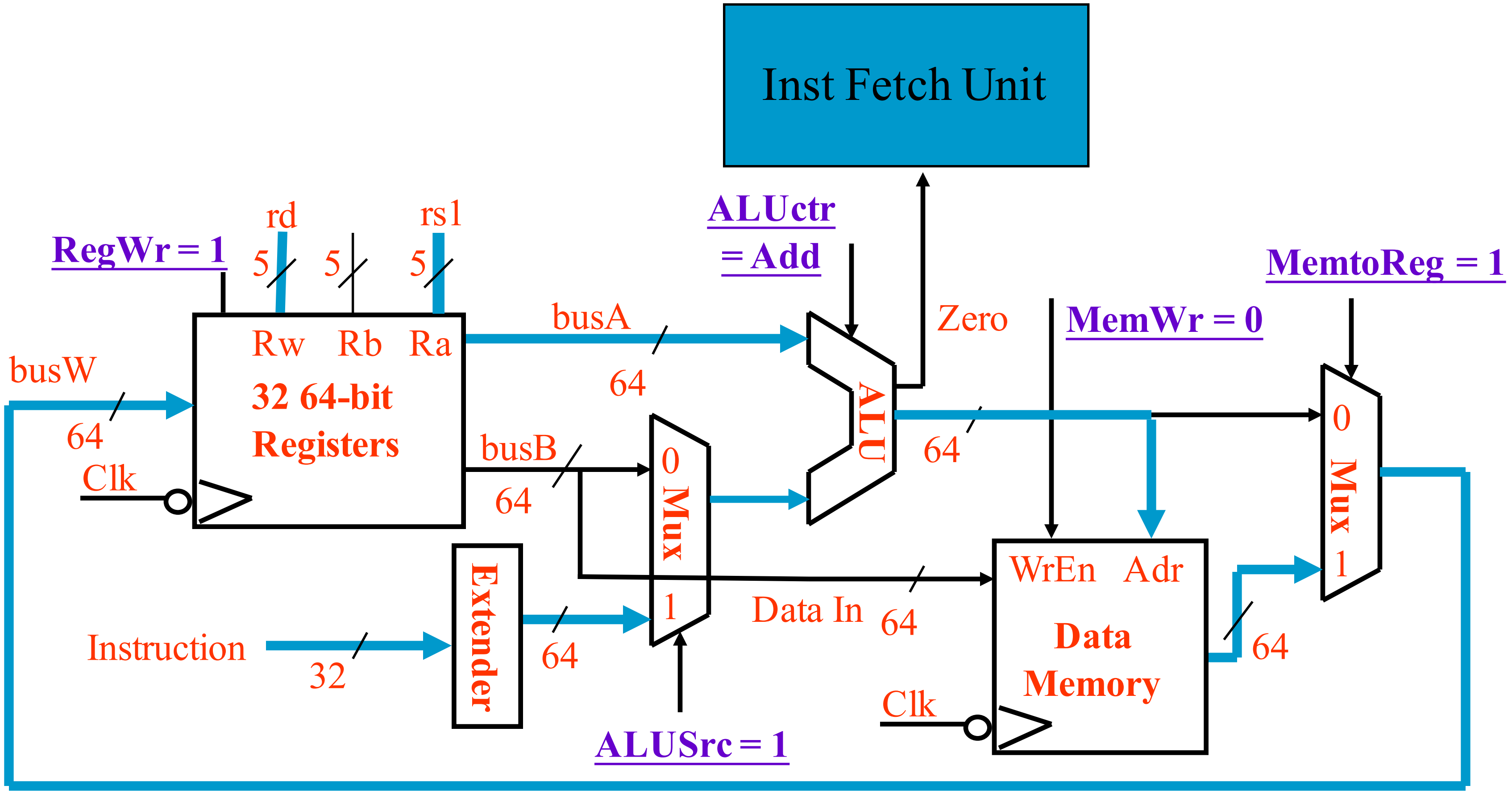
- PC <- PC + 4
  - This is the same for all instructions except: Branch and Jump



The Single Cycle Datapath during Load

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

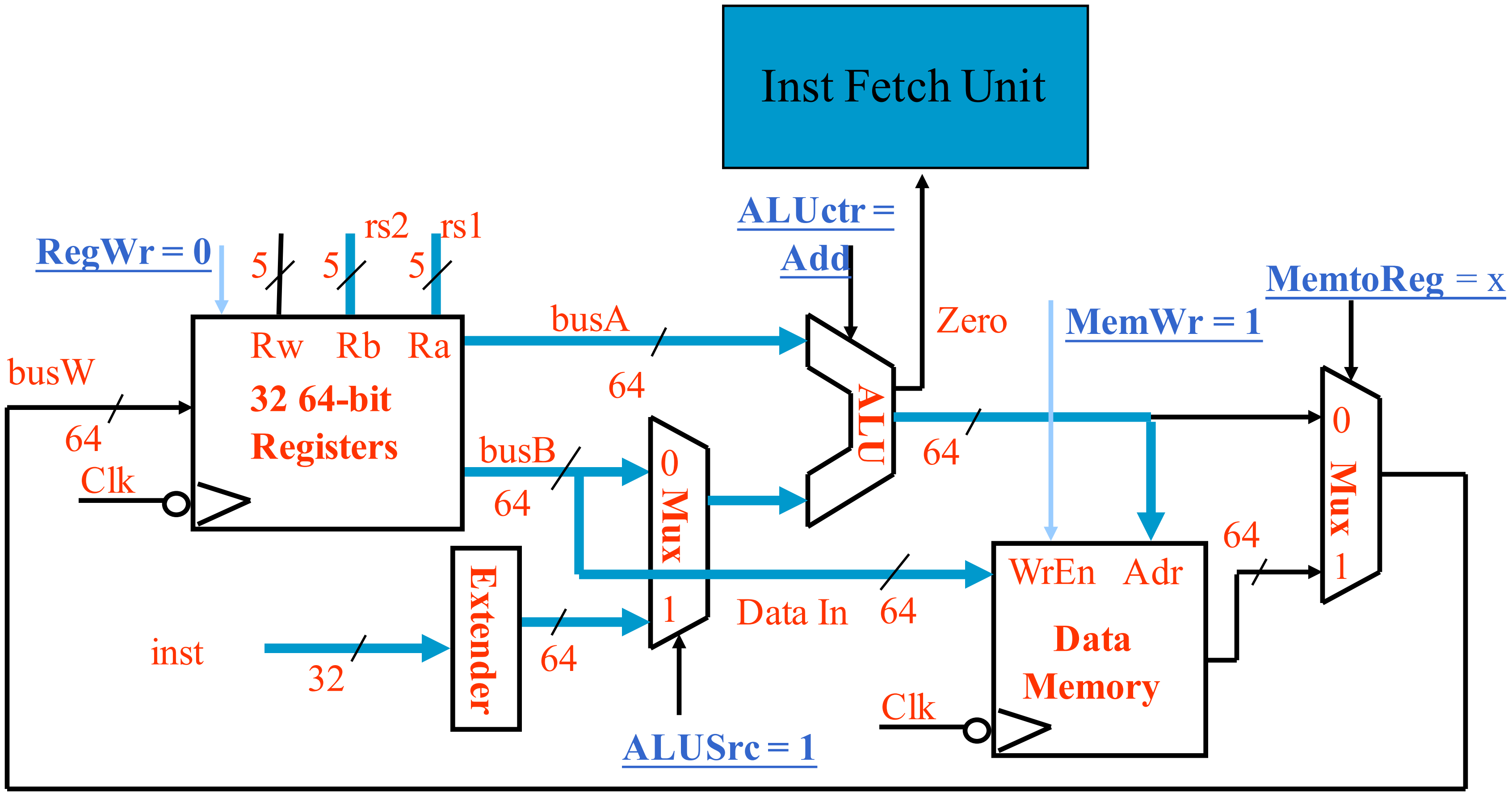
- R[rd] <- Data Memory {R[rs1] + SignExt[imm12]}



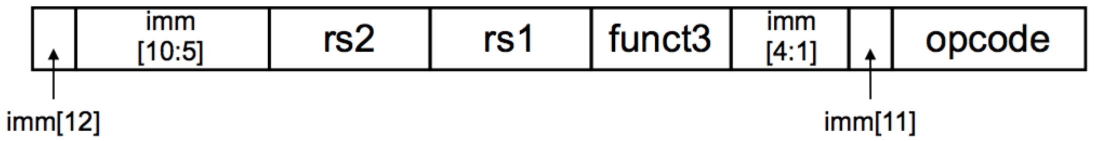
The Single Cycle Datapath during Store

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

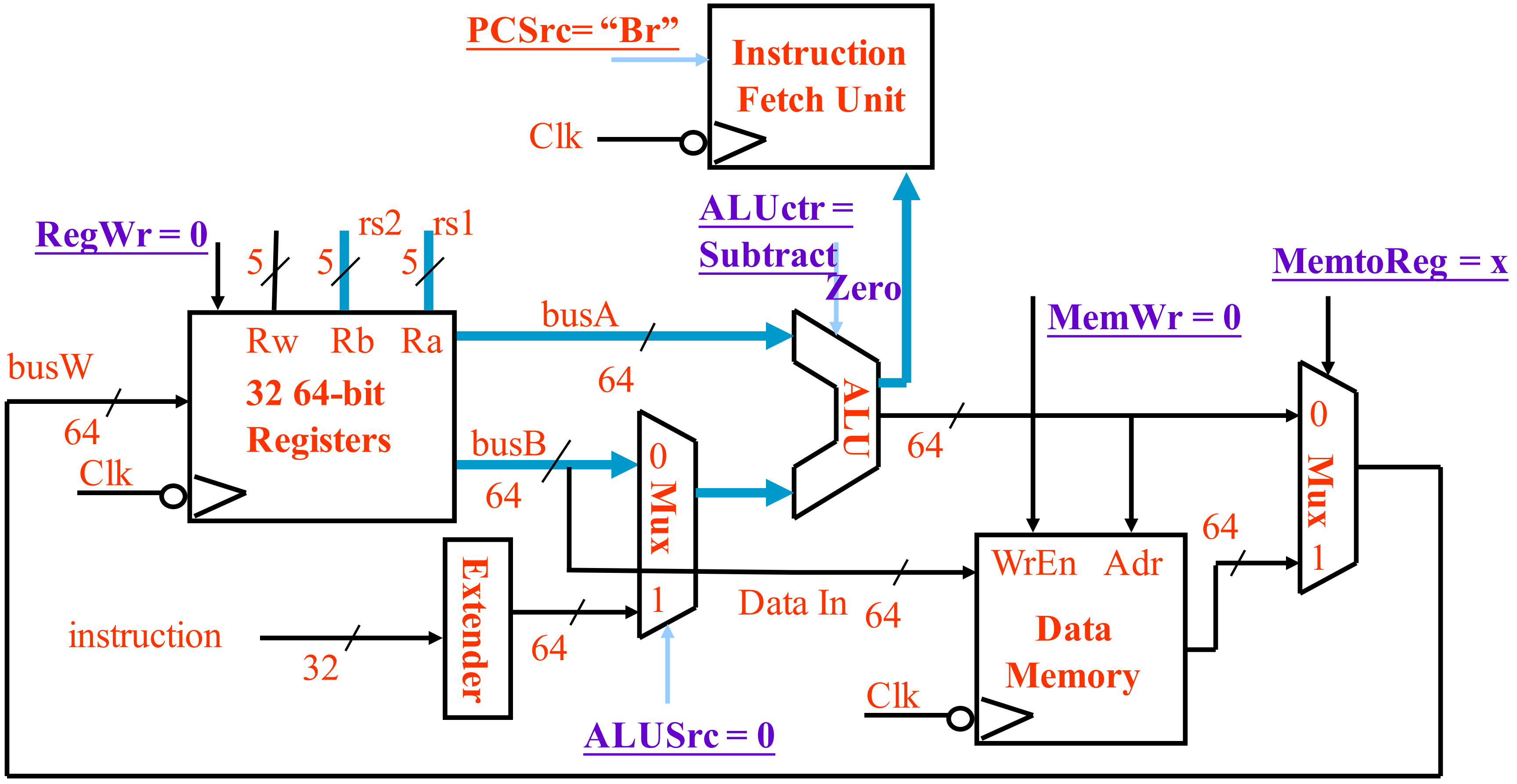
- Data Memory { R[rs1] + SignExt[imm12]} <- R[rs2]



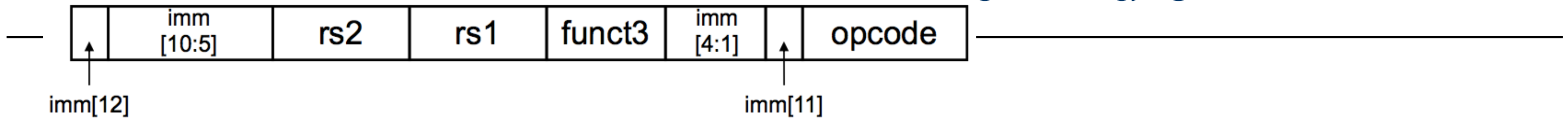
The Single Cycle Datapath during Branch (beq)



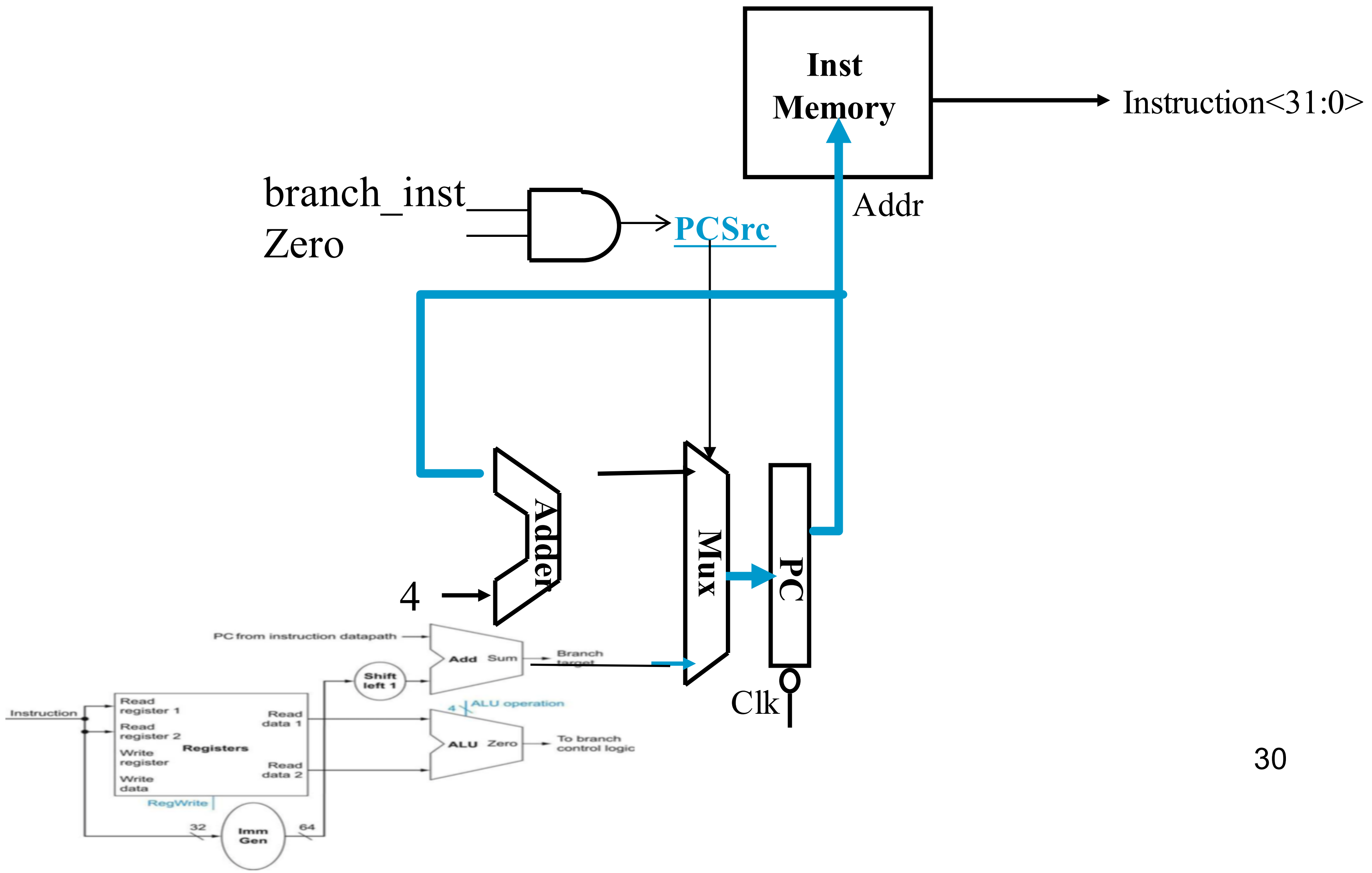
- if (R[rs1] - R[rs2] == 0) then Zero <- 1 ; else Zero <- 0



Instruction Fetch Unit at the End of



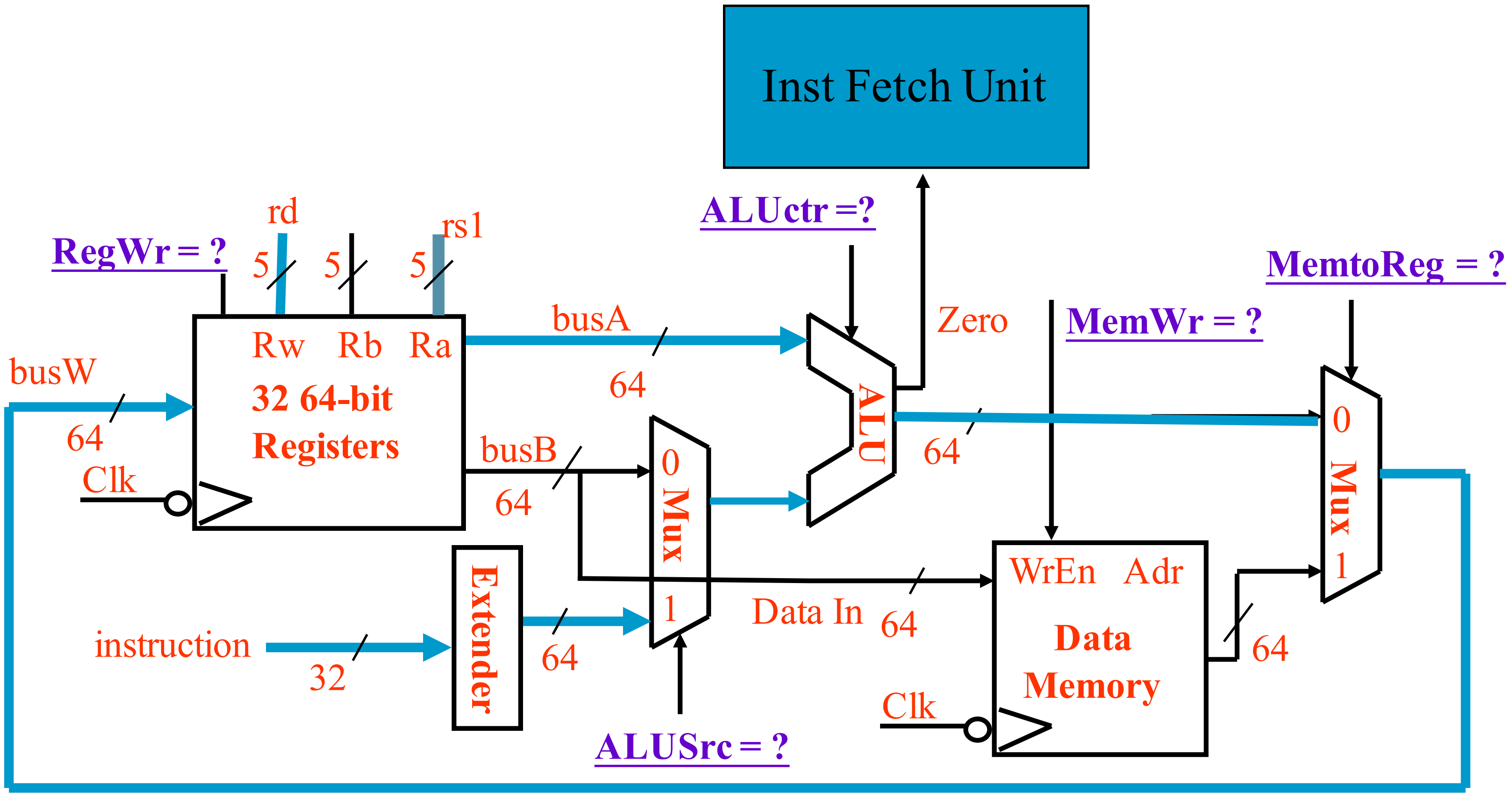
- if (Zero == 1) then PC = PC + SignExt[imm12]*2 ; else PC = PC + 4



Exercise: The Single Cycle Datapath during Ori

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

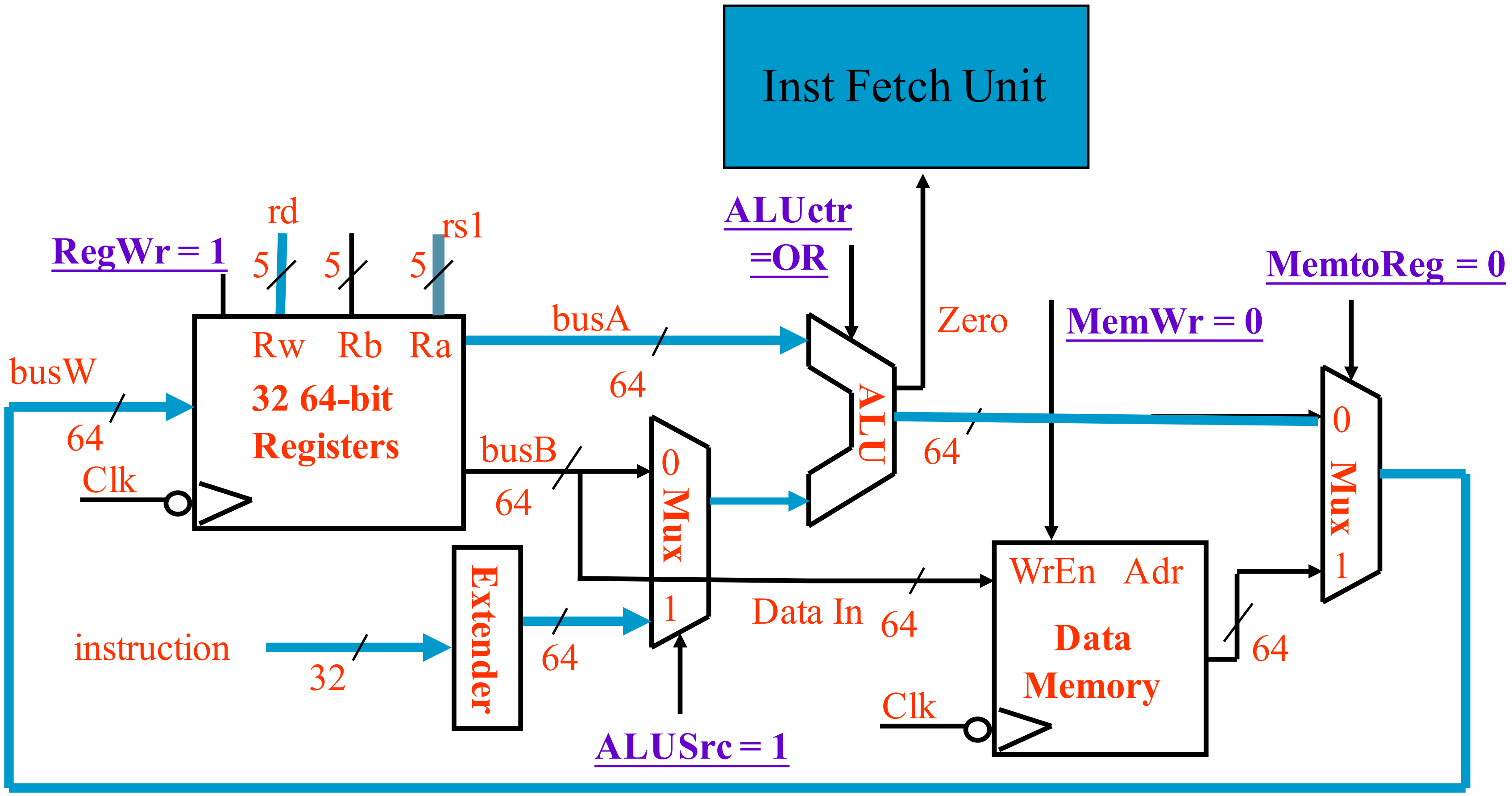
- R[rd] <- R[rs1] or ZeroExt[imm12]}



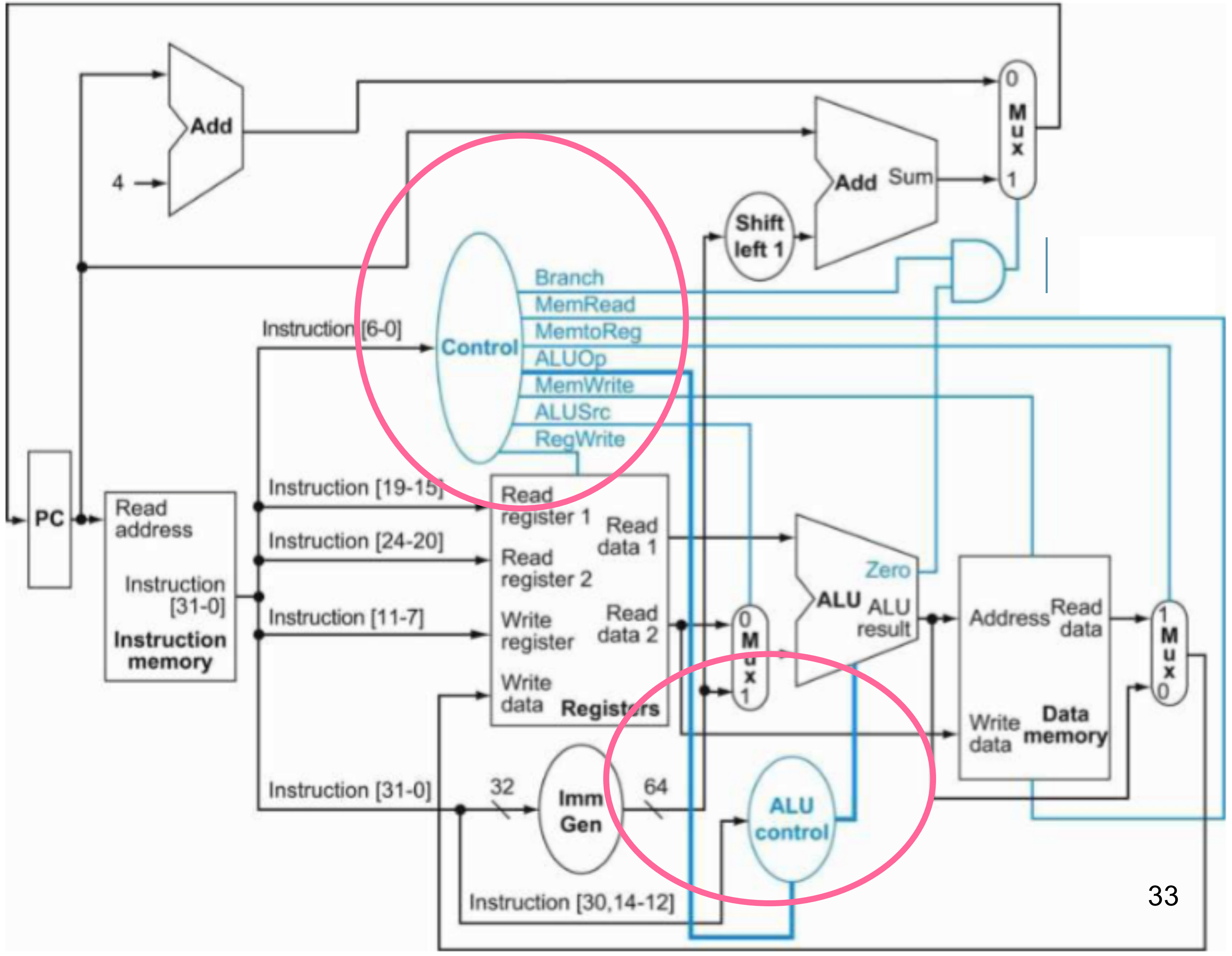
Exercise: The Single Cycle Datapath during Ori

immediate	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

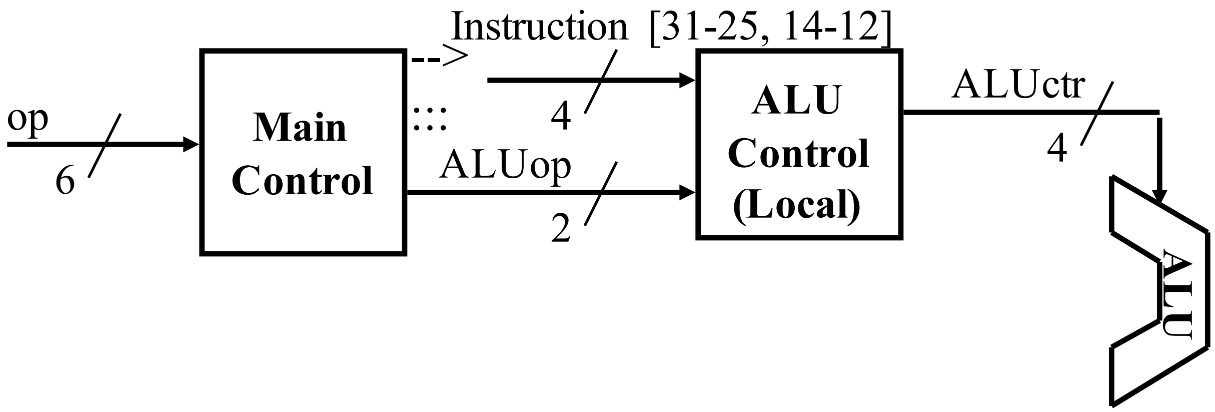
- R[rd] <- R[rs1] or ZeroExt[imm12]}



Step 5: Assemble the Control Unit

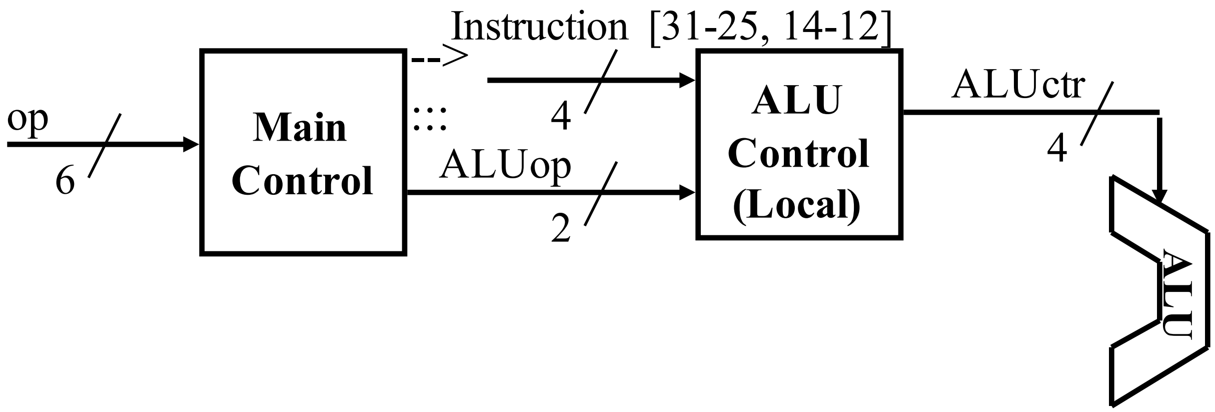


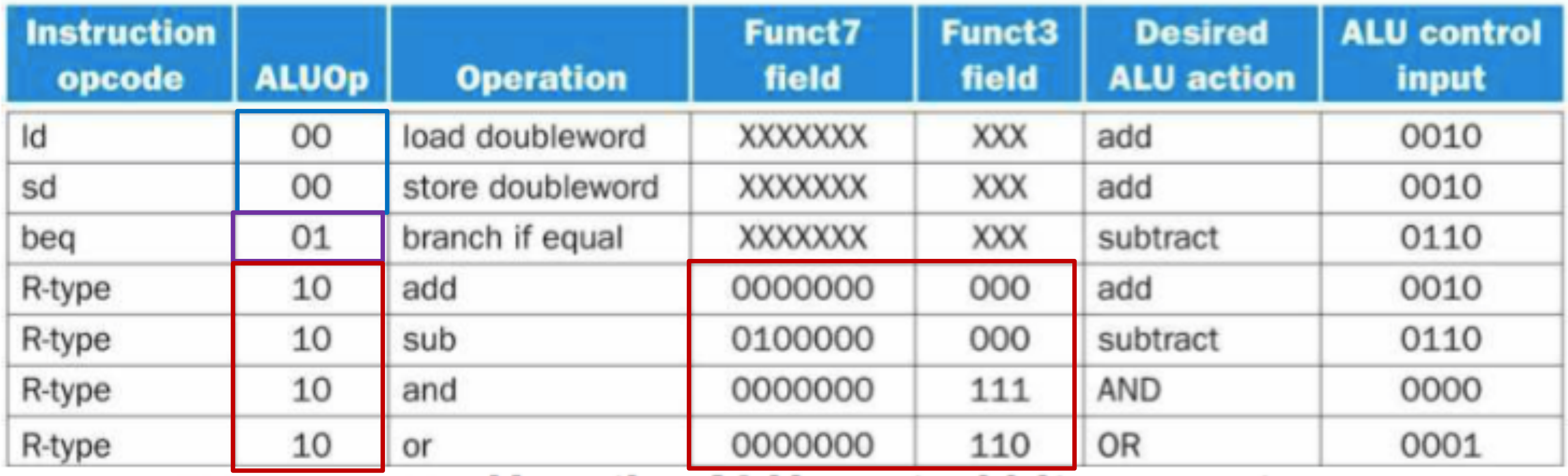
The Concept of Multi-level Decoding



Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

The Concept of Multi-level Decoding



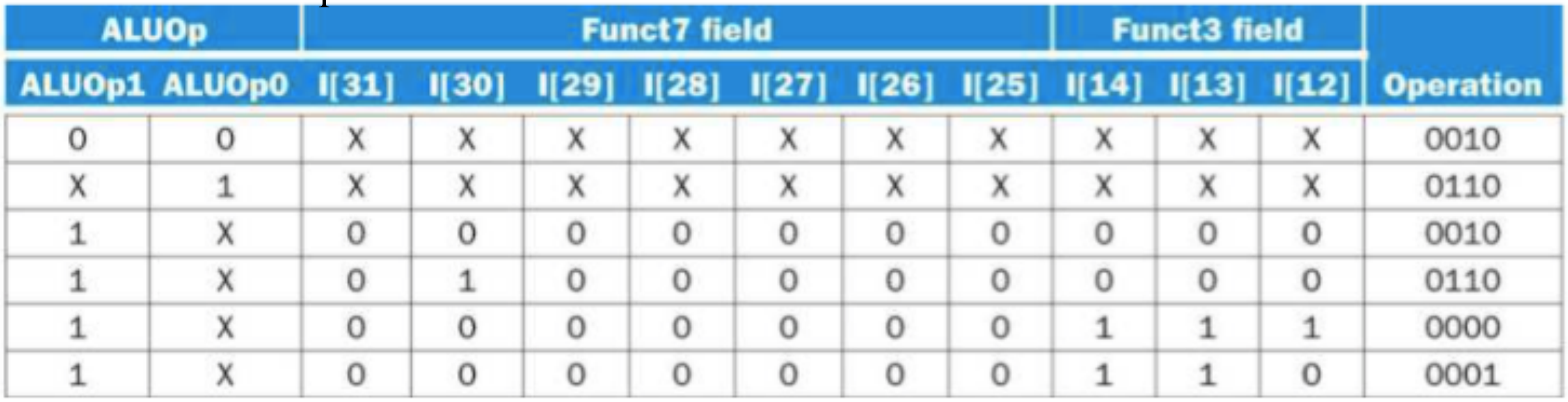


Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

The Truth Table for ALUctr

Input

output



ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

Appendix A: Basics of Logic Design: how to turn true table to logic gates

Truth-Table for the Main Controller

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Performance of Single-Cycle Machines

- **Assumption**
  - **Memory units: 200 ps**
  - **ALU and adders: 100 ps**
  - **Register file (read or write): 50 ps**
  - **Instruction mix:**
    - **25% loads, 10% stores, 45% ALU instructions, 15% branches, and 5% jumps.**
- **Comparison**
  - **Every instruction operates in 1 clock cycle of a fixed length.**
  - **Every instruction executes in 1 clock cycle using a variable-length clock.**

Instruction Class	Functional Units used by the instruction class				
R-Type	Inst Fetch	Register Access	ALU	Register Access	
Load Word	Inst Fetch	Register Access	ALU	Memory Access	Register Access
Store word	Inst Fetch	Register Access	ALU	Memory Access	
Branch	Inst Fetch	Register Access	ALU		
Jump	Inst Fetch				

Performance of Single-Cycle Machines (cont.)

- Recall

  CPU execution time = Instruction count $\times$ CPI $\times$ Clock cycle time

Since CPI must be 1, we can simplify this to

  CPU execution time = Instruction count $\times$ Clock cycle time

- For fixed clock cycle implementation
  - The clock cycle for each instruction is determined by the longest instruction, load, which is 600 ps (200+50+100+200+50).
- For variable clock cycle implementation
  - The average time per instruction with a variable clock is
  - 600 $\times$ 25% + 550 $\times$ 10% + 400 $\times$ 45% + 350 $\times$ 15% + 200 $\times$ 5% = 447.5 ps
- The variable clock implementation would be faster by

$$\frac{600}{447.5} = 1.34$$

NEXT TIME: Pipelining