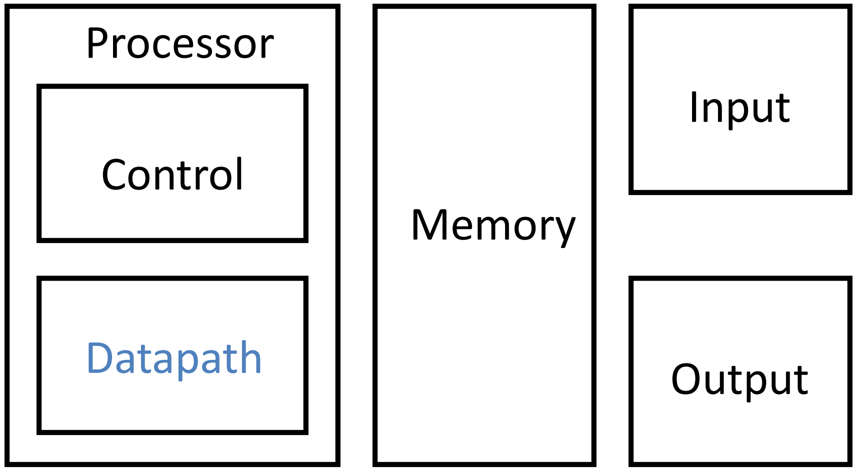
Single Cycle Datapath

The Big Picture: Where are We?

- The Five Classic Components of a Computer



The Performance Perspective

- Performance of a machine was determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction

The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type

31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - I-type

31 26	21	16	0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

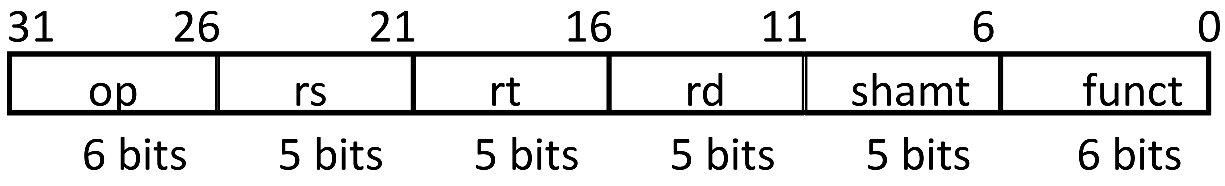
  - J-type

31 26	0
op	target address
6 bits	26 bits

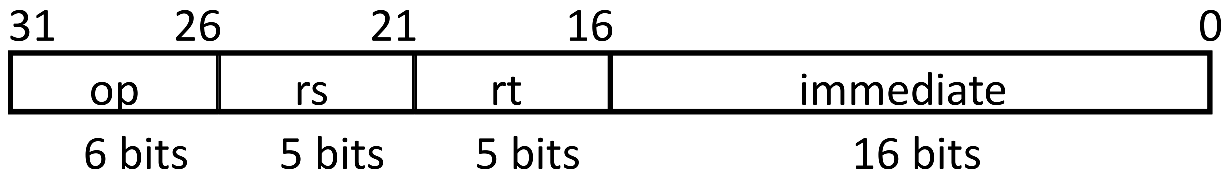
- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the “op” field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

The MIPS Subset

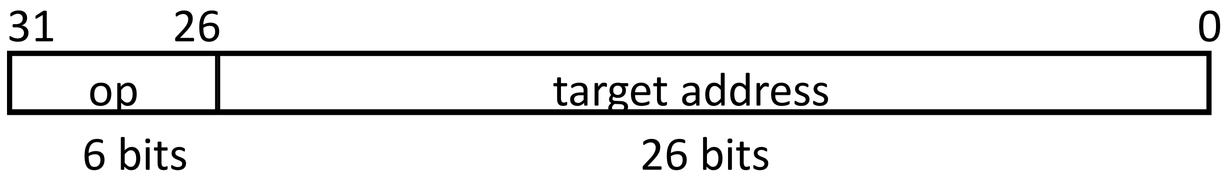
- ADD and subtract
  - add rd, rs, rt
  - sub rd, rs, rt



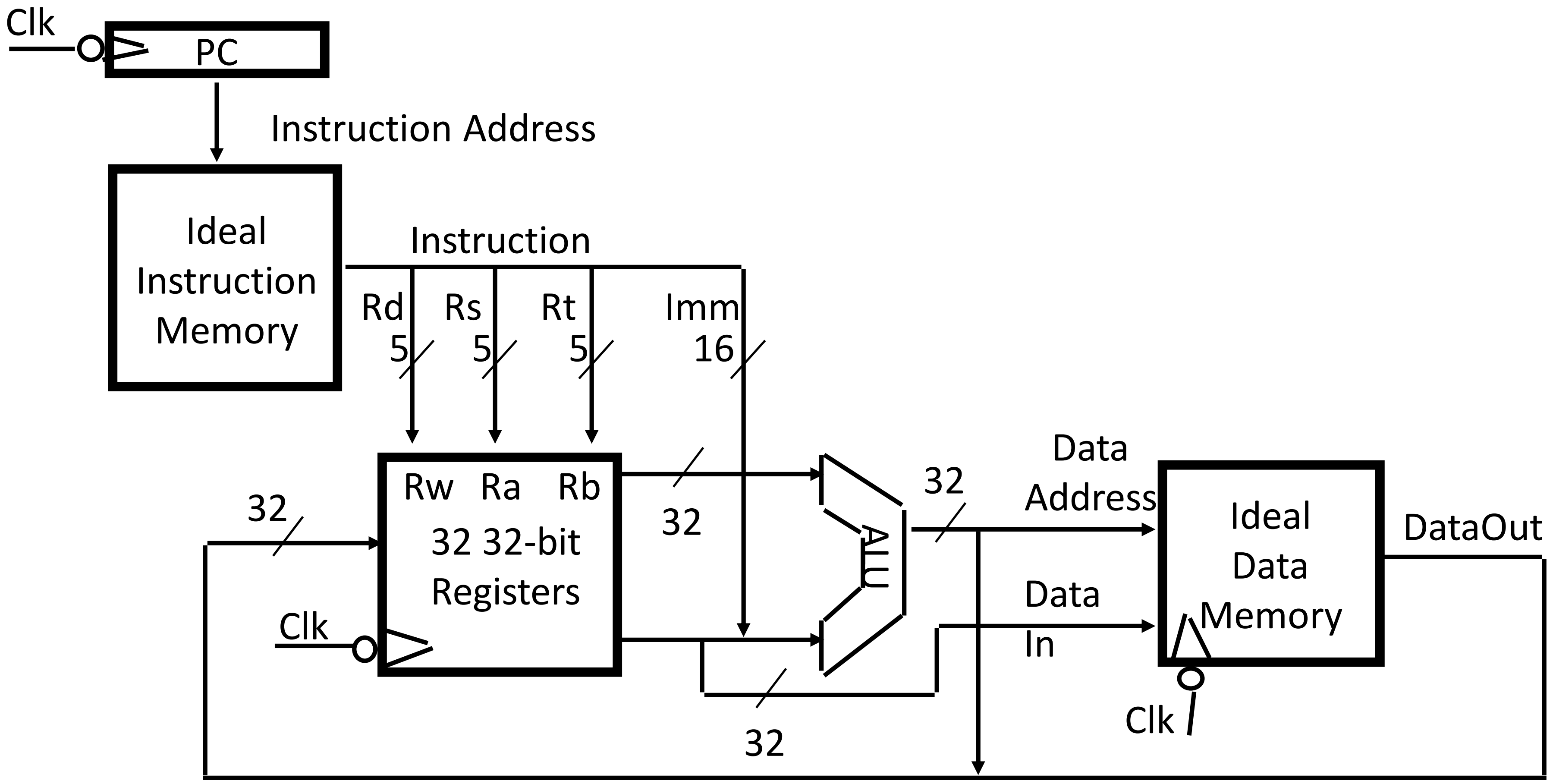
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16
- BRANCH:
  - beq rs, rt, imm16



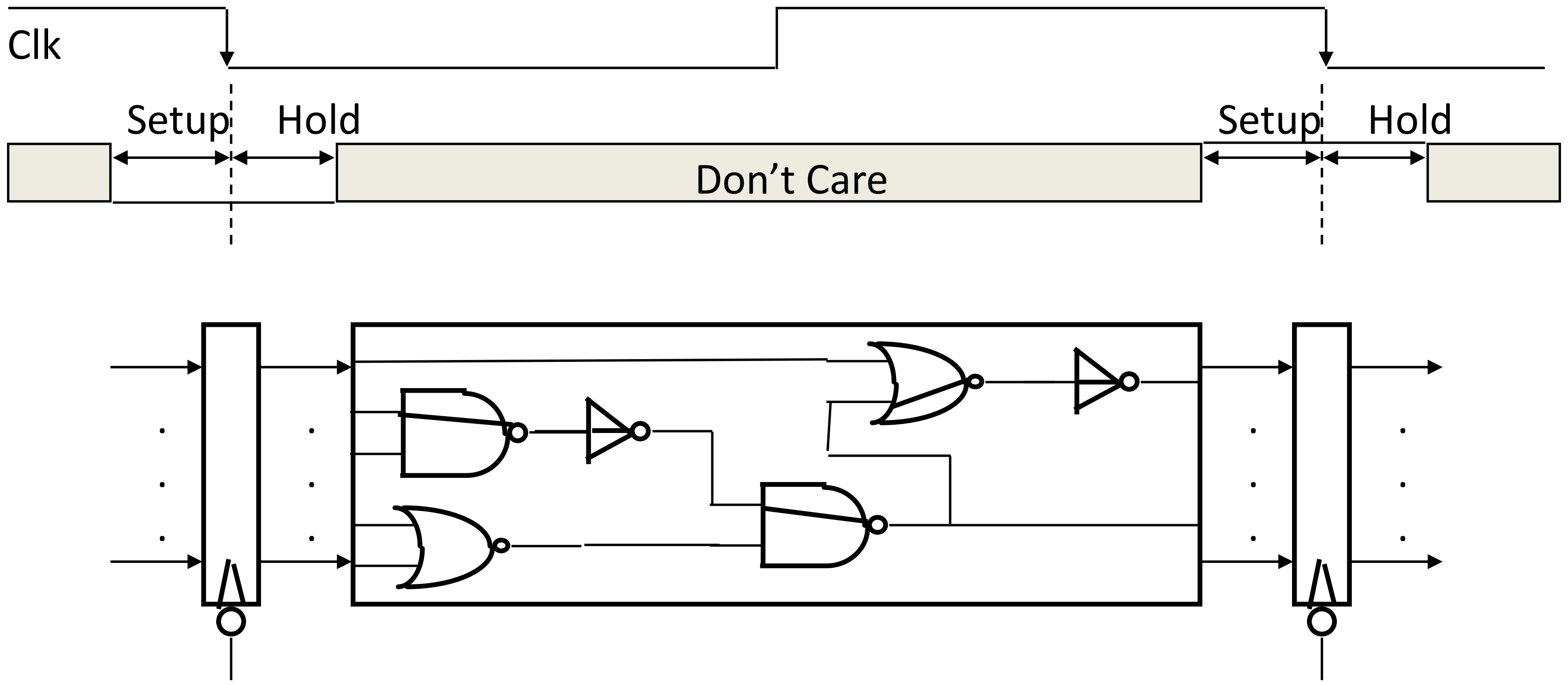
- JUMP:
  - j target



An Abstract View of the Implementation



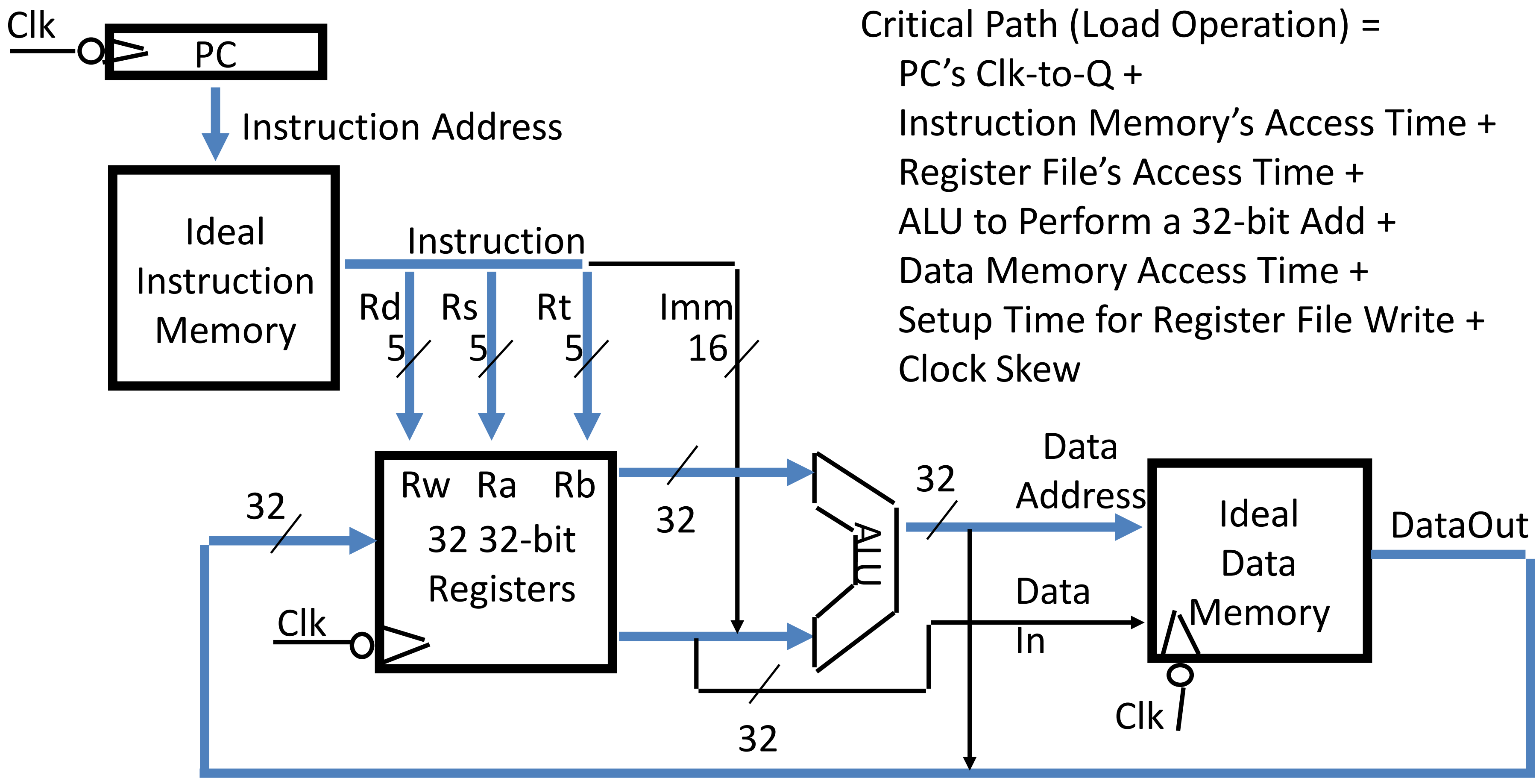
Clocking Methodology



- All storage elements are clocked by the same clock edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew

An Abstract View of the Critical Path

- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."



Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew

The Steps of Designing a Processor

- Instruction Set Architecture => Register Transfer Language
- Register Transfer Language => Datapath Design
  - Datapath components
  - Datapath interconnect
- Datapath components => Control signals
- Control signals => Control logic => Control Unit Design

RTL Example 1: The ADD Instruction

- add rd, rs, rt

– mem[PC]	Fetch the instruction from memory
– R[rd] <- R[rs] + R[rt]	The ADD operation
– PC <- PC + 4	Calculate the next instruction’s address

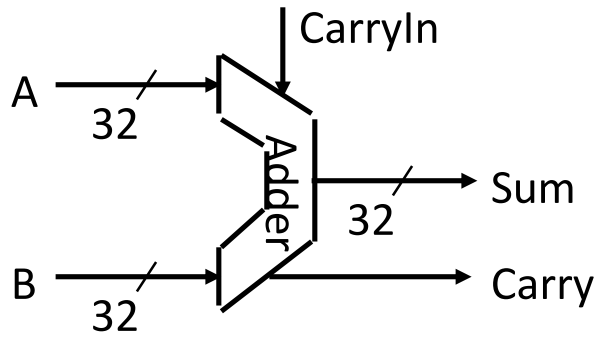
RTL Example 2: The Load Instruction

- **lw   rt, rs, imm16**

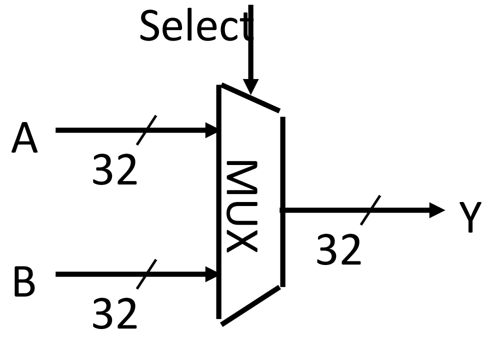
– mem[PC]	Fetch the instruction from memory
– Addr <- R[rs] + SignExt(imm16)	Calculate the memory address
– R[rt] <- Mem[Addr]	Load the data into the register
– PC <- PC + 4	Calculate the next instruction’s address

Need of Combinational Logic Elements

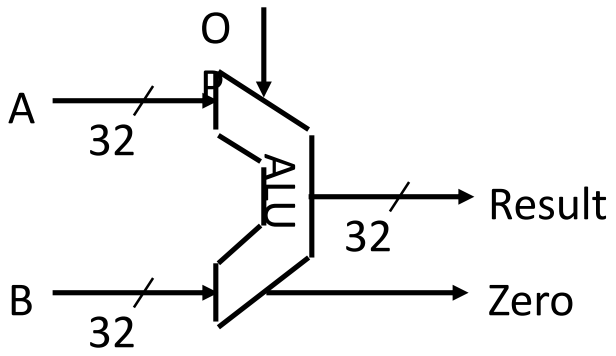
- Adder



- MUX

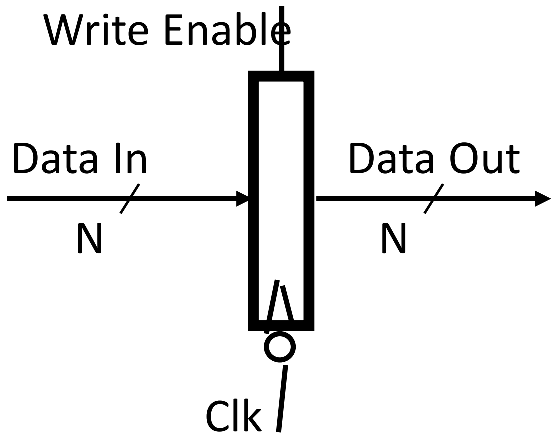


- ALU

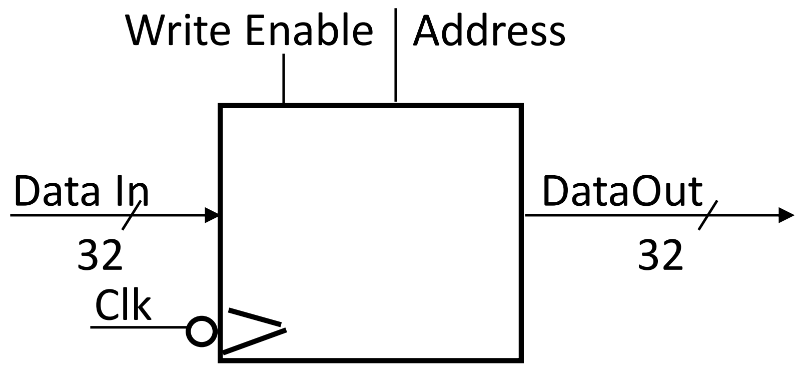


Storage Element: Register

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In

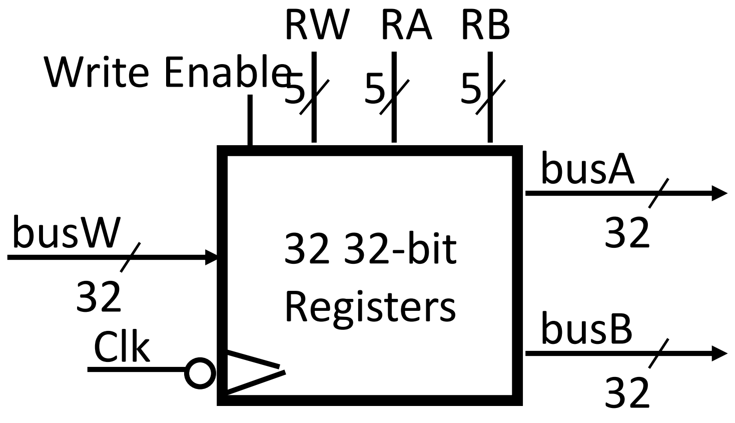


Storage Element: Idealized Memory



- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Write Enable = 0: Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory 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.”

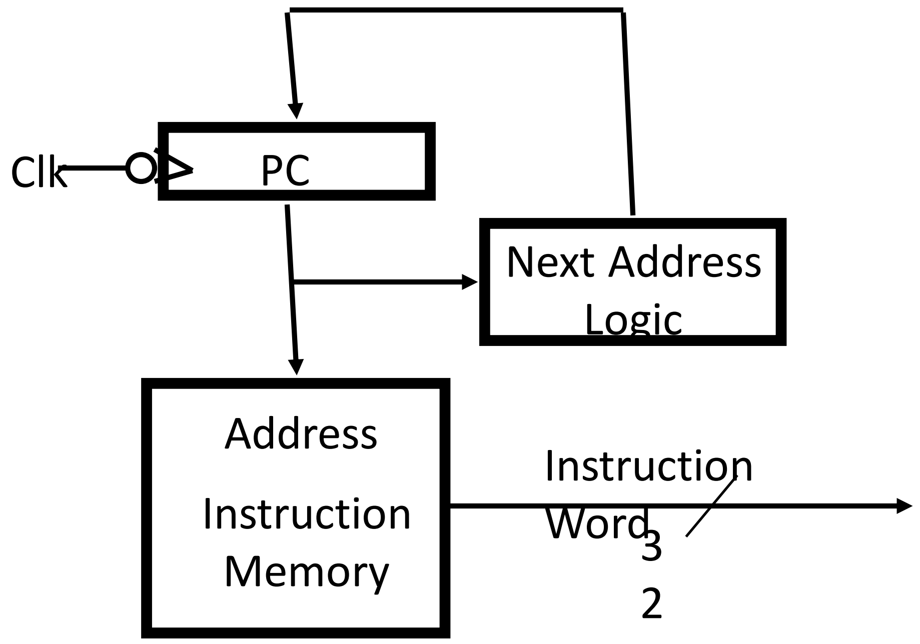
Storage Element: Register File



- Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA selects the register to put on busA
  - RB selects the register to put on busB
  - RW selects the register to be written via busW 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."

Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump   PC <- “something else”



RTL: The ADD Instruction

- add rd, rs, rt

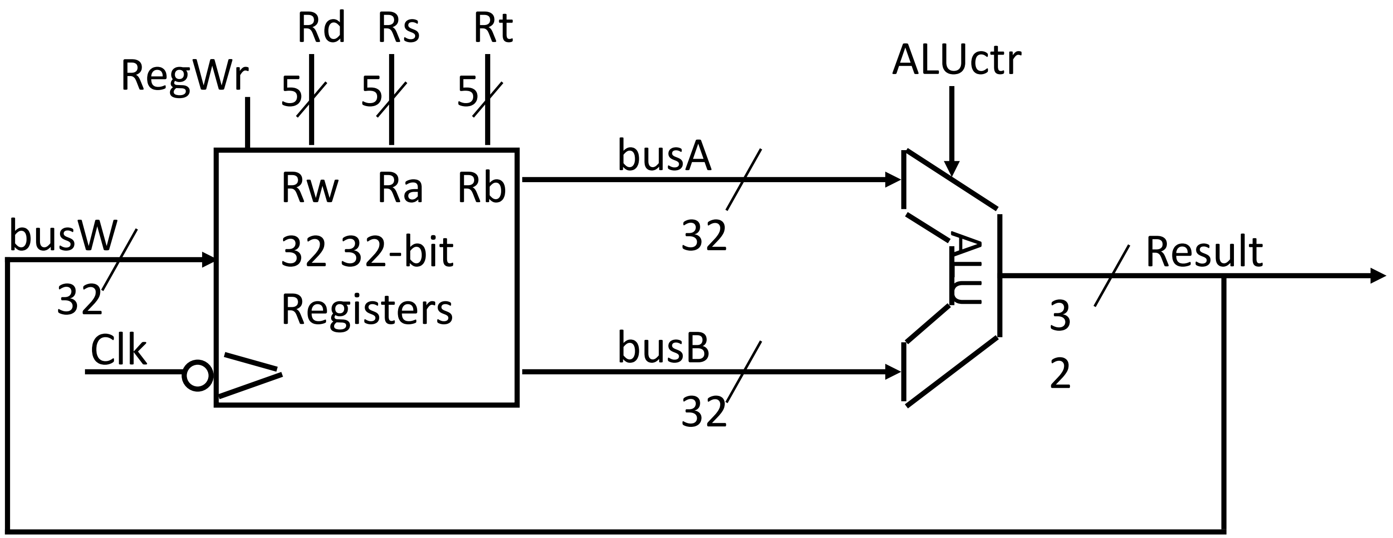
– mem[PC]	Fetch the instruction from memory
– R[rd] <- R[rs] + R[rt]	The ADD operation
– PC <- PC + 4	Calculate the next instruction’s address

Datapath for Register-Register Operations

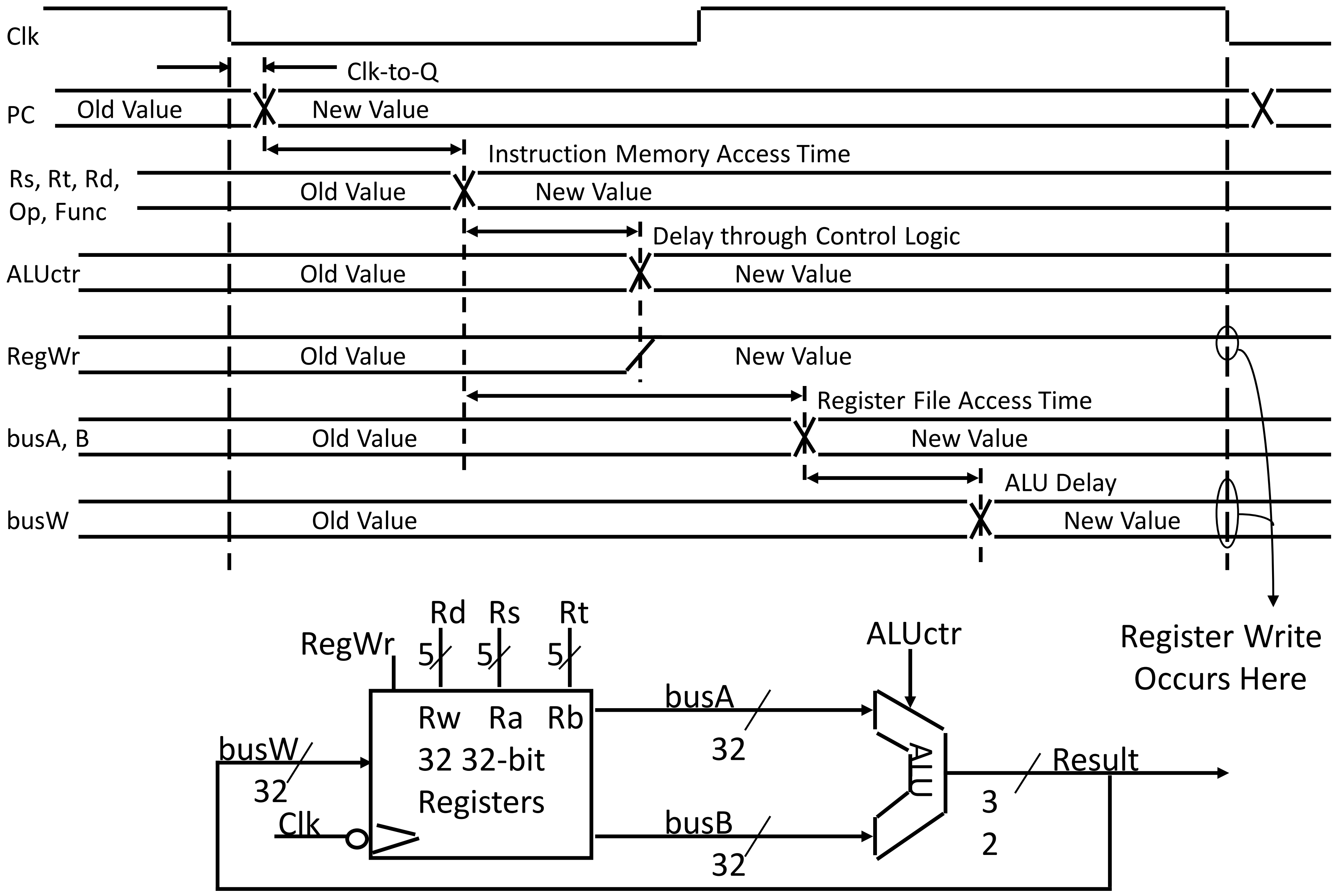
- R[rd] <- R[rs] op R[rt] Example: add rd, rs, rt
  - Ra, Rb, and Rw comes from instruction’s rs, rt, and rd fields
  - ALUctr and RegWr: control logic after decoding the instruction

31 26 21 16 11 6 0

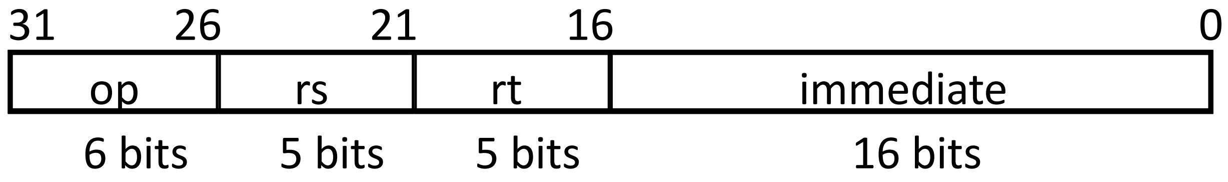
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



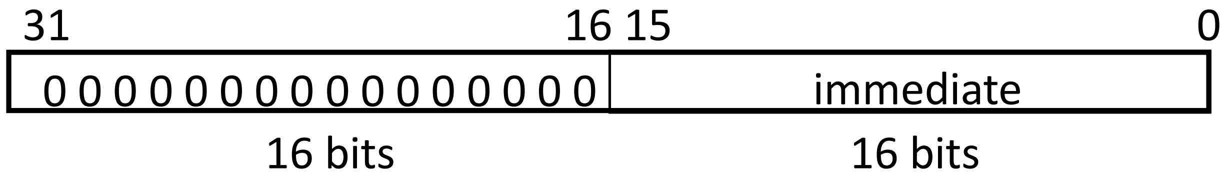
Register-Register Timing



RTL: The OR Immediate Instruction

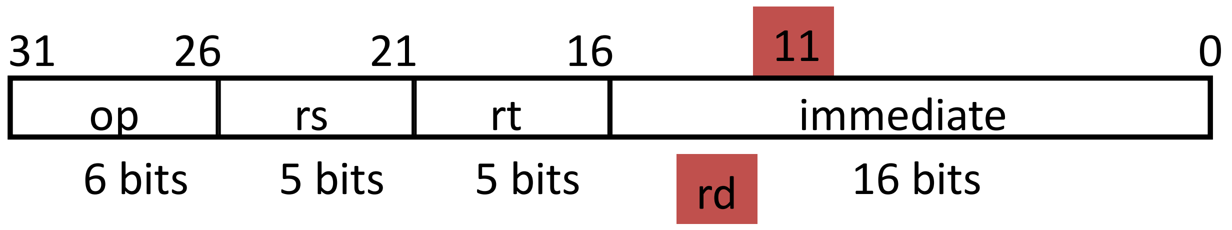


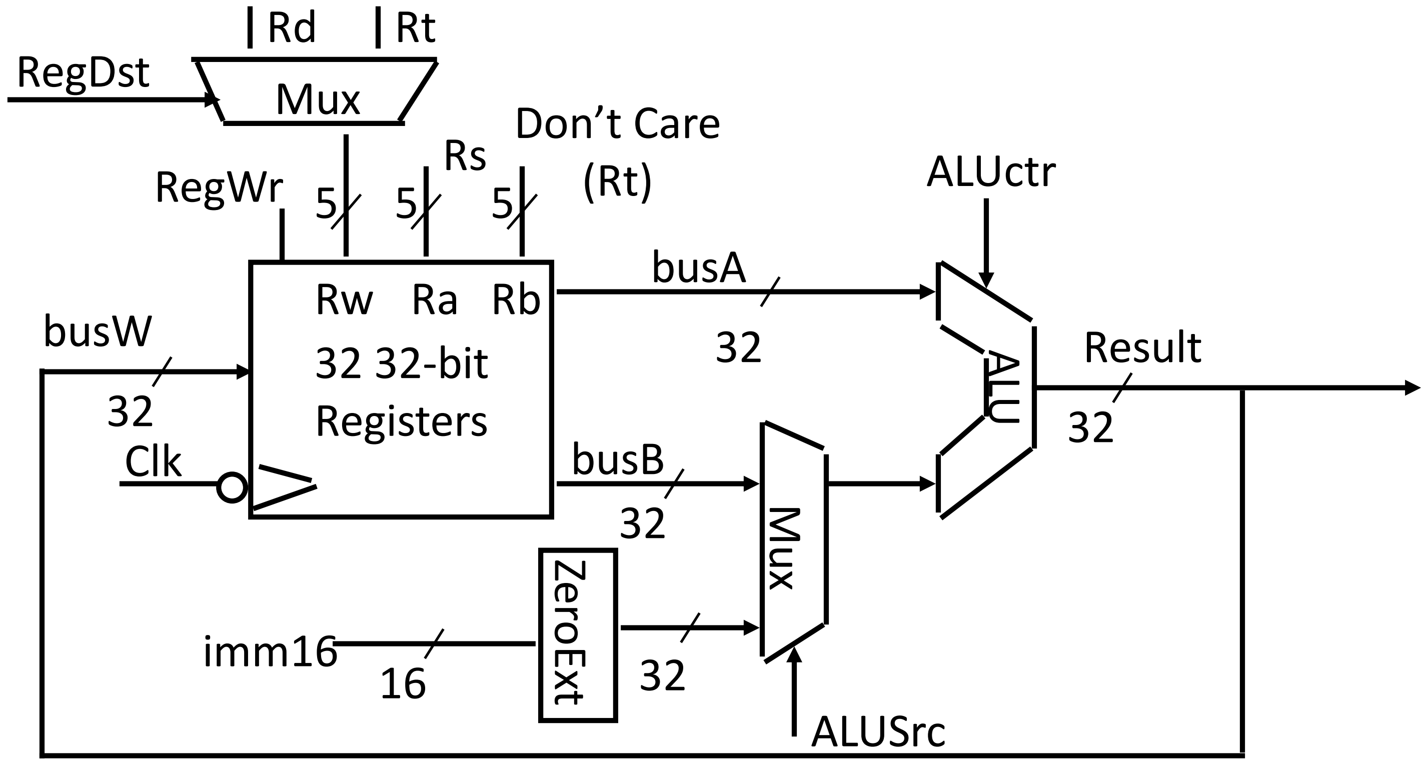
- ori rt, rs, imm16
  - mem[PC] Fetch the instruction from memory
  - R[rt] <- R[rs] **OR** ZeroExt(imm16) The OR operation
  - PC <- PC + 4 Calculate the next instruction's address



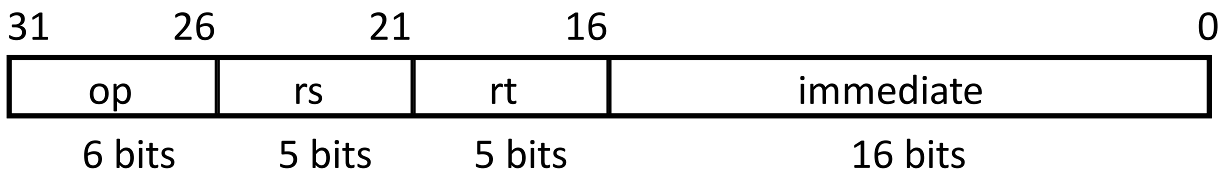
Datapath for Logical Operations with Immediate

- R[rt] <- R[rs] op ZeroExt[imm16]] Example: ori rt, rs, imm16



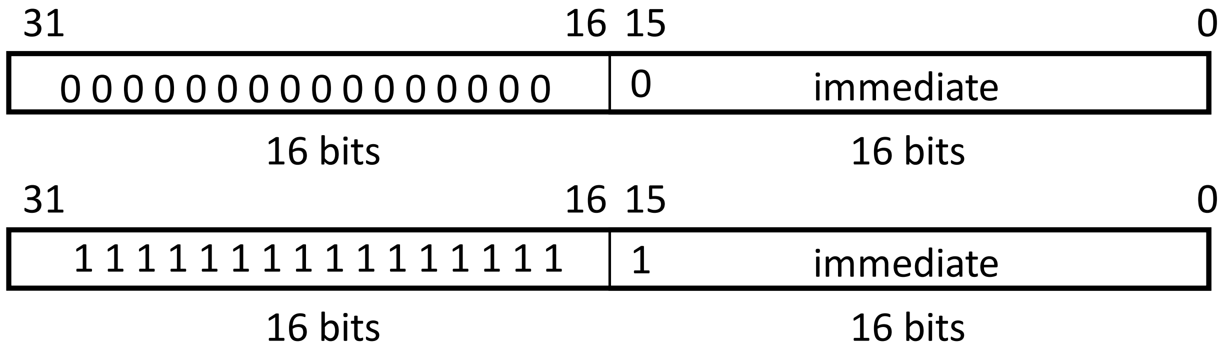


RTL: The Load Instruction



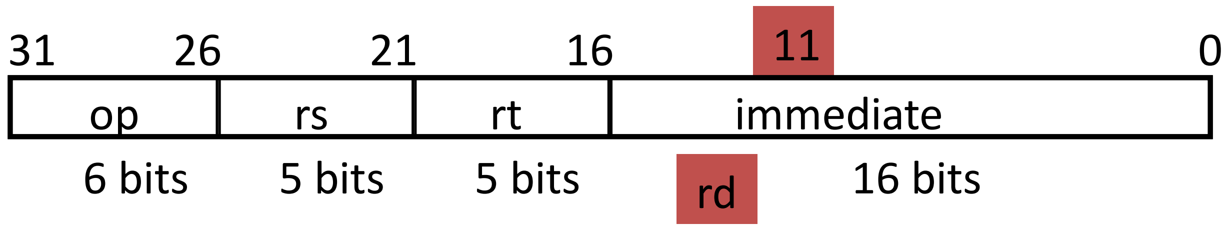
- lw rt, rs, imm16
  - mem[PC] Fetch the instruction from memory
  - Addr <- R[rs] + SignExt(imm16) Calculate the memory address
  - R[rt] <- Mem[Addr] Load the data into the register
  - PC <- PC + 4 Calculate the next instruction's address

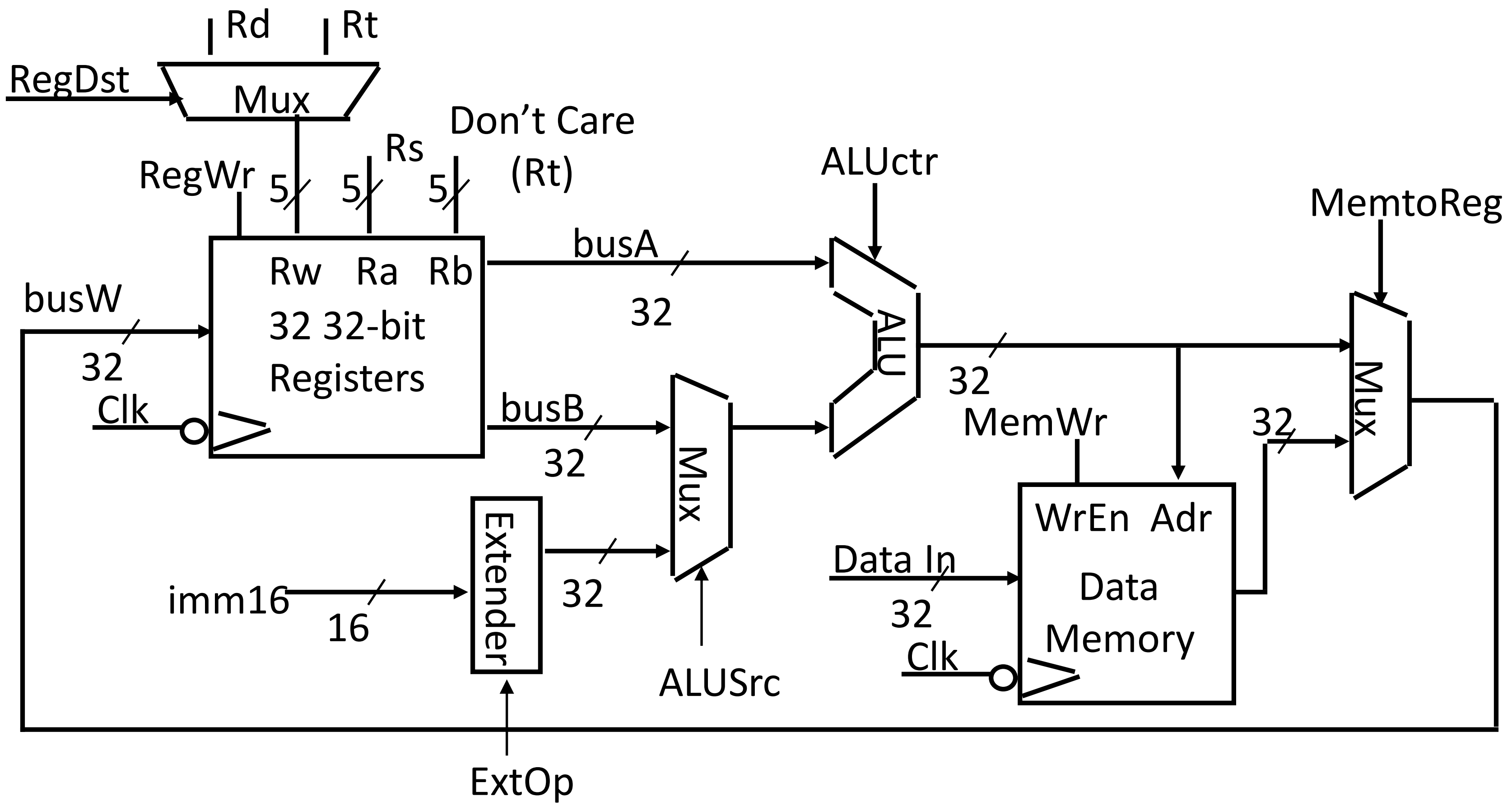
SignExt operation



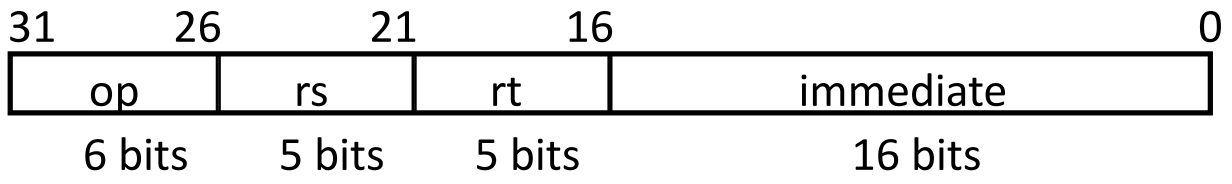
Datapath for Load Operations

- R[rt] <- Mem[R[rs] + SignExt[imm16]]   Example: lw   rt, rs, imm16





RTL: The Store Instruction



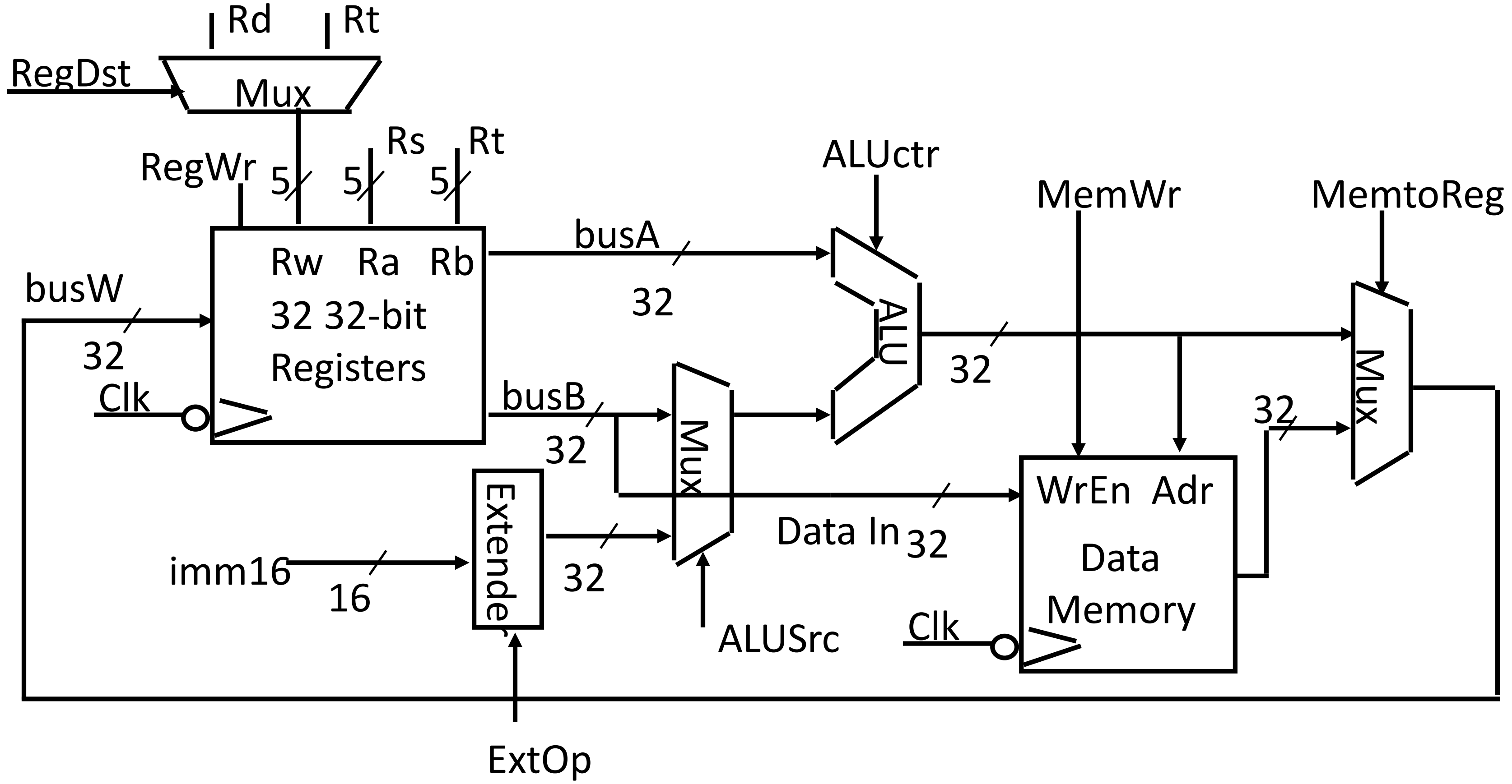
- sw rt, rs, imm16
  - mem[PC] — Fetch the instruction from memory
  - Addr <- R[rs] + SignExt(imm16) — Calculate the memory address
  - Mem[Addr] <- R[rt] — Store the register into memory
  - PC <- PC + 4 — Calculate the next instruction’s address

Datapath for Store Operations

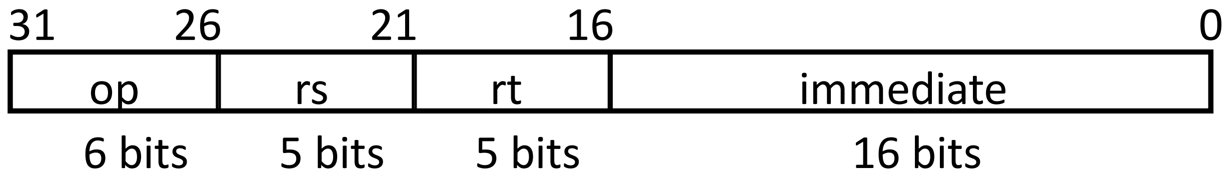
- Mem[R[rs] + SignExt[imm16] <- R[rt]]   Example: sw rt, rs, imm16

31 26 21 16 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits



RTL: The Branch Instruction



- beq   rs, rt, imm16
  - mem[PC]   Fetch the instruction from memory
  - Cond <- R[rs] - R[rt]   Calculate the branch condition
  - if (COND eq 0)   Calculate the next instruction's address

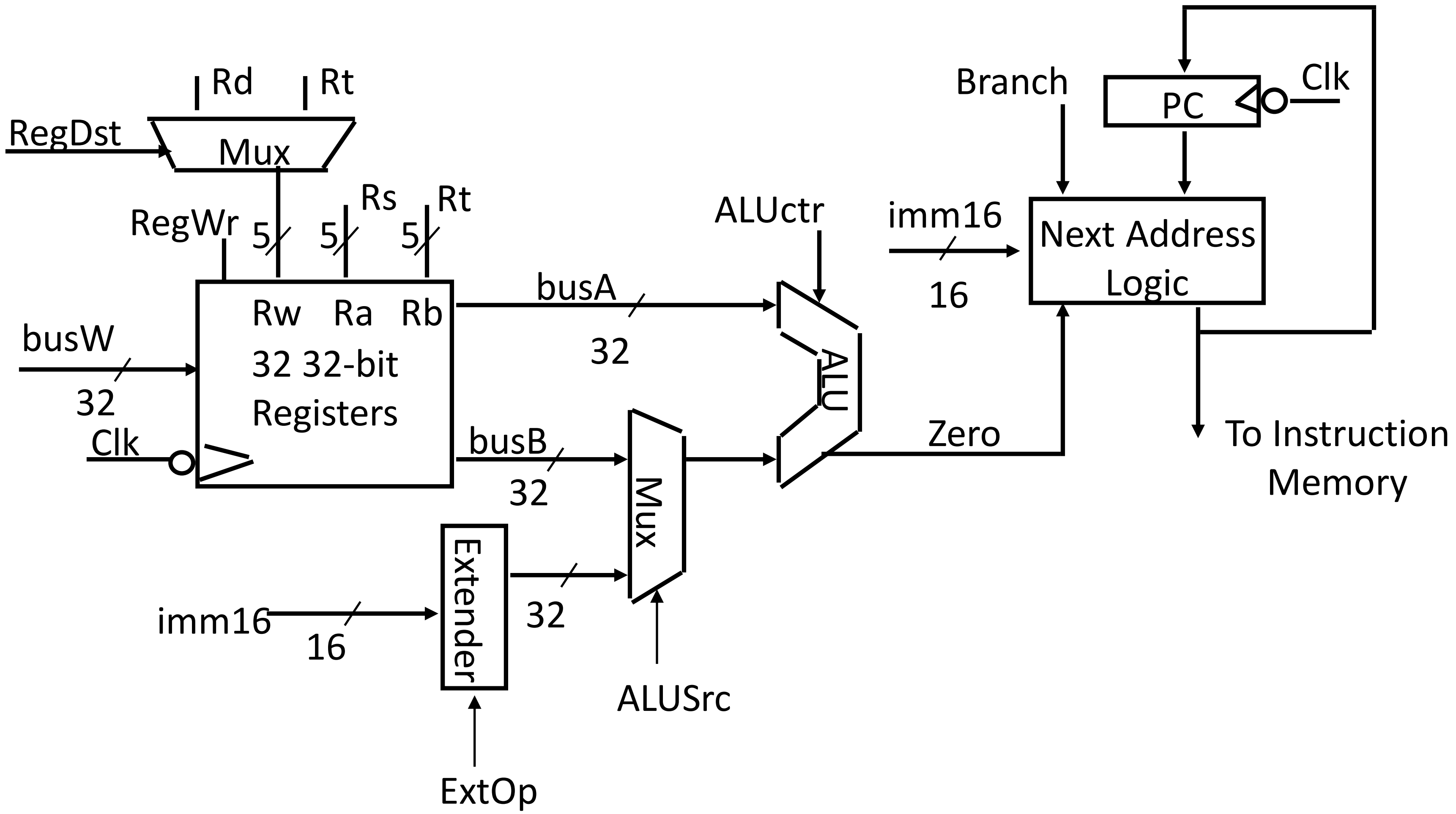
    PC <- PC + 4 + (SignExt(imm16) x 4)

    else PC <- PC + 4

Datapath for Branch Operations

- beq rs, rt, imm16 We need to compare Rs and Rt!

31 26	21	16	0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

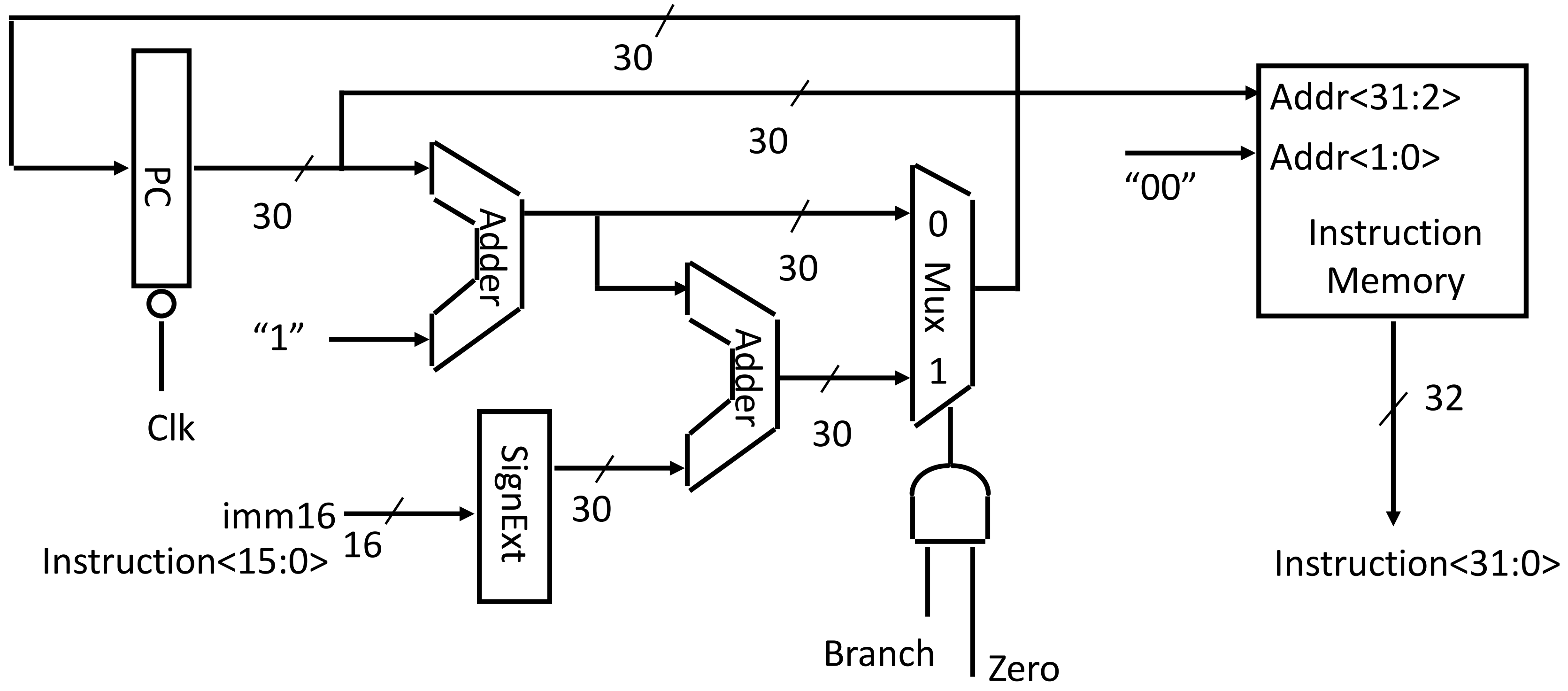


Binary Arithmetics for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4
- The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
- In other words:
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs
- In practice, we can simplify the hardware by using a 30-bit PC<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

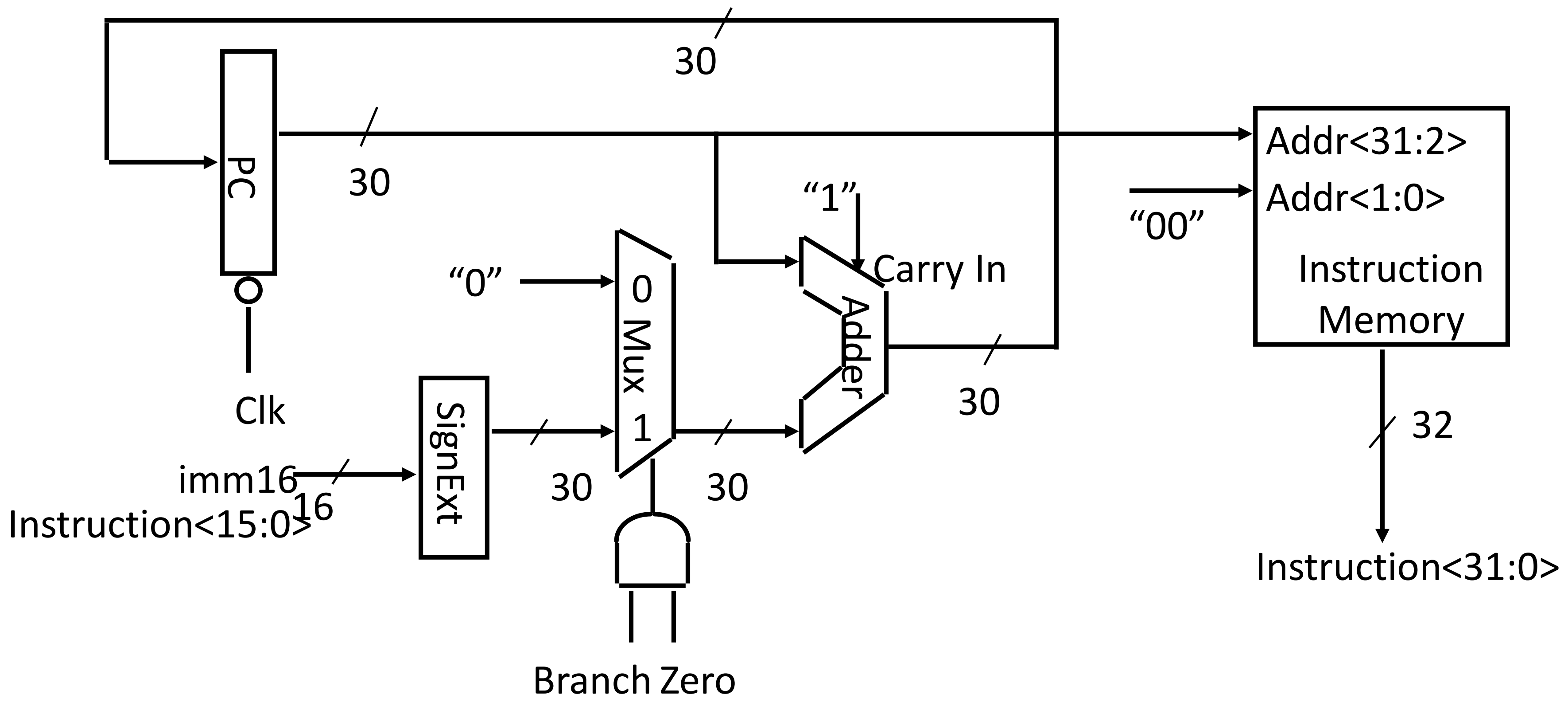
Next Address Logic: Expensive and Fast Solution

- Using a 30-bit PC:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

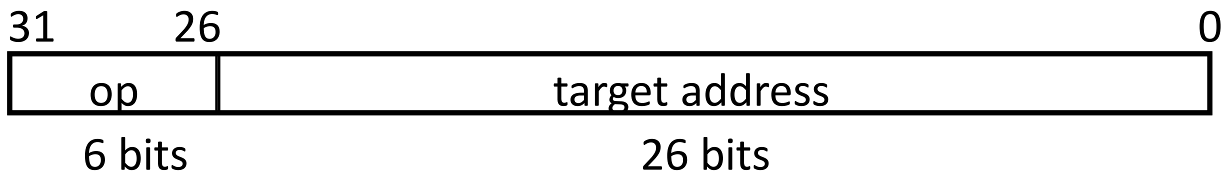


Next Address Logic: Cheap and Slow Solution

- Why is this slow?
  - Cannot start the address add until Zero (output of ALU) is valid
- Does it matter that this is slow in the overall scheme of things?
  - Probably not here. Critical path is the load operation.



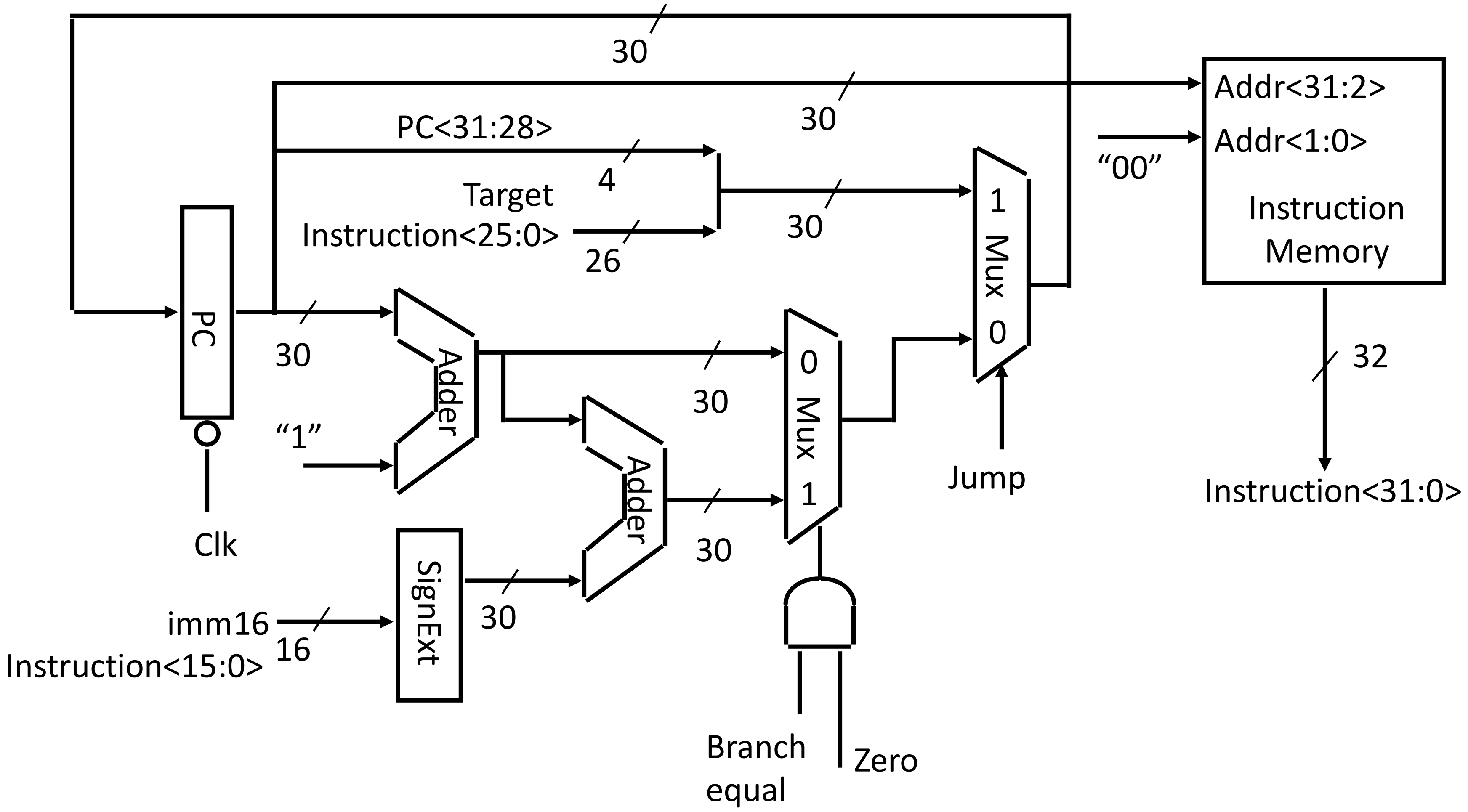
RTL: The Jump Instruction



- j target
  - mem[PC] Fetch the instruction from memory
  - PC<31:2> <- PC<31:28> concatenate target<25:0>
    Calculate the next instruction’s address

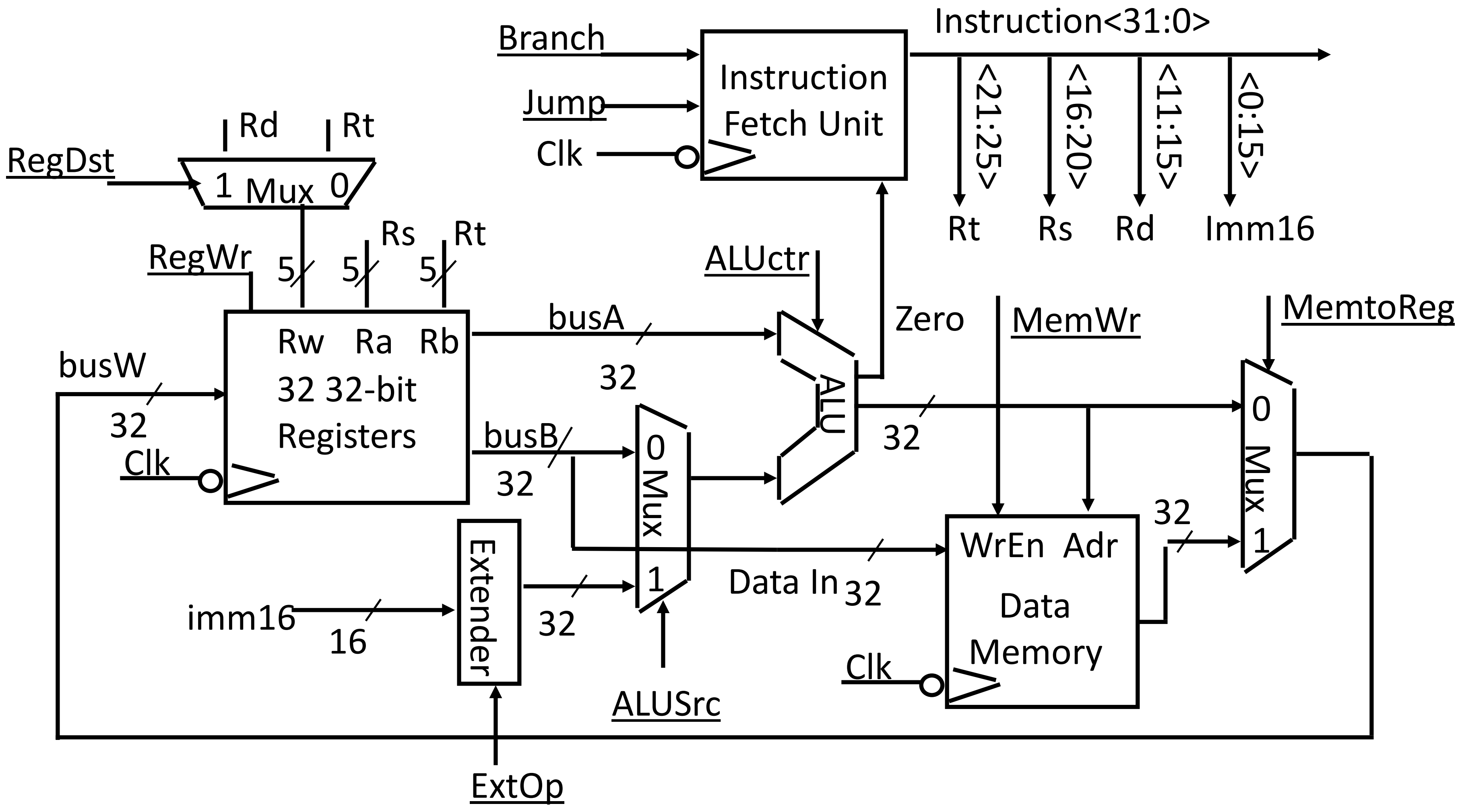
Instruction Fetch Unit

- j target
  - PC<31:2> <- PC<31:28> concat target<25:0>



Putting it All Together: A Single Cycle Datapath

- We have everything except control signals (underline)



References

- John L. Hennessy and David A Patterson, Computer Architecture: A quantitative Approach, Morgan Kaufmann / Elsevier, Sixth Edition, 23rd November 2017