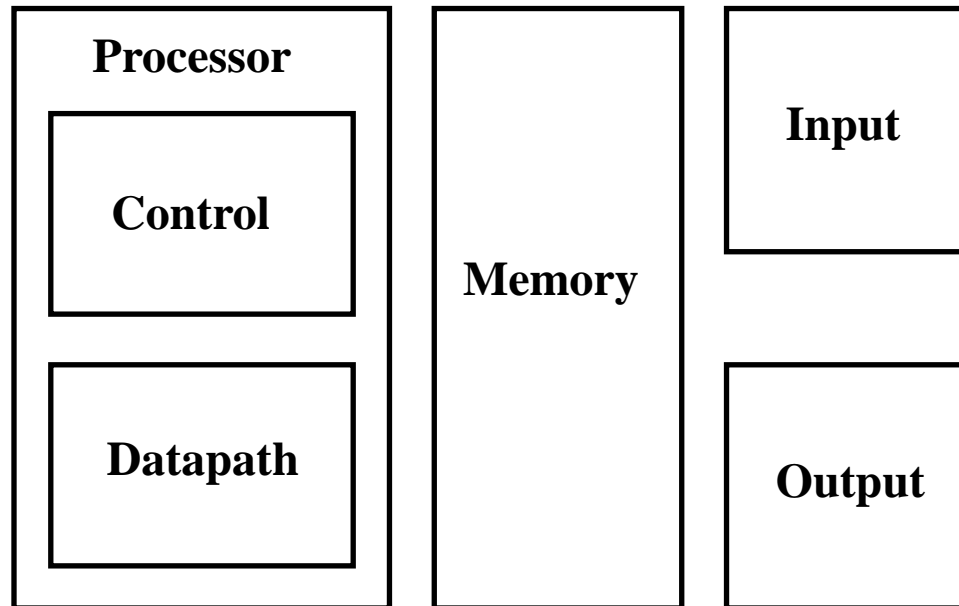


CSE331 – Computer Organization

Lecture 6: Designing a Single Cycle Datapath

The Big Picture

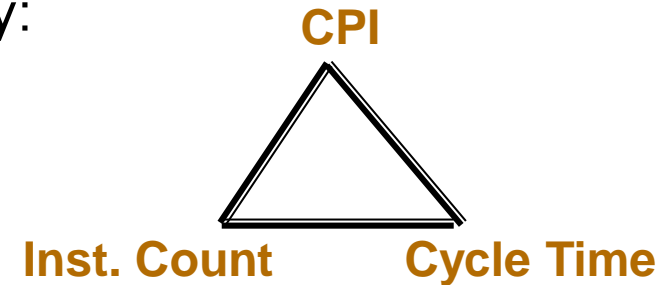
► The Five Classic Components of a Computer



The Big Picture: The Performance Perspective

- ▶ Performance of a machine is 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

- ▶ Single cycle processor – one clock cycle per instruction

- Advantages: Simple design, low CPI
- Disadvantages: Long cycle time, which is limited by the slowest instruction.

How to Design a Processor: step-by-step

- ▶ Analyze instruction set => datapath requirements
 - the meaning of each instruction is given by *register transfers*
$$R[rd] \leftarrow R[rs] + R[rt];$$
 - datapath must include storage element for ISA registers
 - datapath must support each register transfer
- ▶ Select set of datapath components and establish clocking methodology
- ▶ Design datapath to meet the requirements
- ▶ Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
- ▶ Design the control logic

Review: MIPS Instruction Formats

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



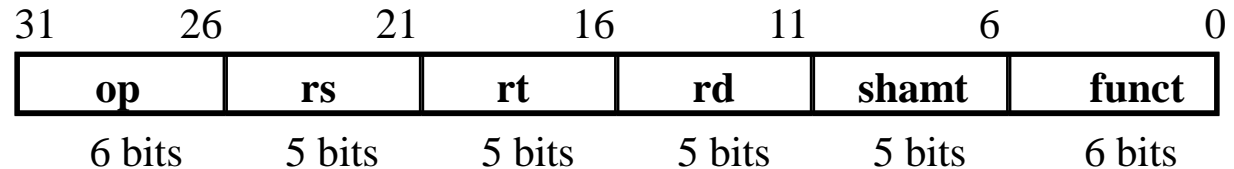
The different fields are:

op : basic operation of the instruction (opcode)
rs, rt, rd : source and destination register specifier
shamt : shift amount
funct : selects the variant of the operation in the “op” field
immediate : address offset or immediate value
target address : target address of the jump instruction

Step 1a: The MIPS Subset for Today

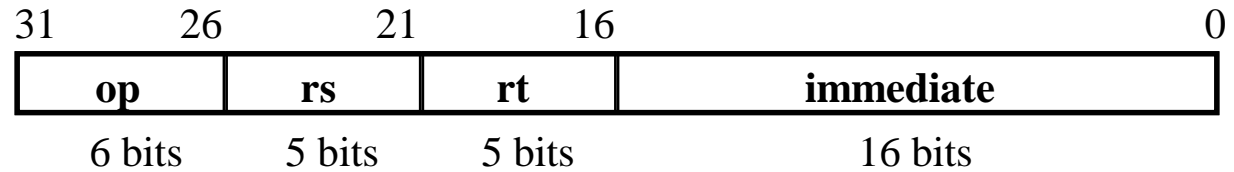
► R-Type:

- add rd, rs, rt
- sub rd, rs, rt
- and rd, rs, rt
- or rd, rs, rt
- slt rd, rs, rt



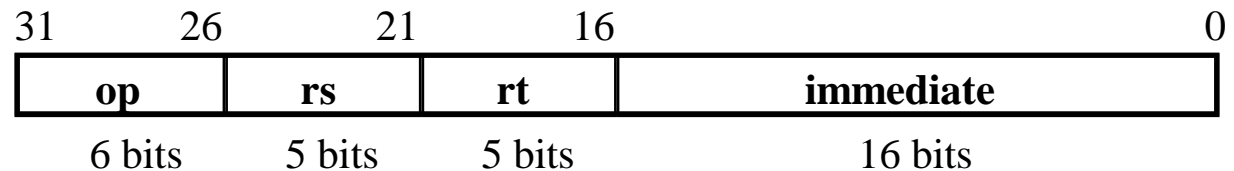
► LOAD and STORE:

- lw rt, rs, imm16
- sw rt, rs, imm16



► BRANCH:

- beq rs, rt, imm16



Register Transfer Logic (RTL)

- ▶ RTL gives the meaning of the instructions
- ▶ All instructions start by fetching the instruction

op | rs | rt | rd | shamt | funct = MEM[PC]

$$\text{op} \mid \text{rs} \mid \text{rt} \mid \text{Imm16} = \text{MEM}[\text{PC}]$$

inst Register Transfers

$$\text{add} \quad \mathbf{R}[\text{rd}] \leftarrow \mathbf{R}[\text{rs}] + \mathbf{R}[\text{rt}]; \quad \mathbf{PC} \leftarrow \mathbf{PC} + 4$$
$$\text{sub} \quad \mathbf{R}[\text{rd}] \leftarrow \mathbf{R}[\text{rs}] - \mathbf{R}[\text{rt}]; \quad \mathbf{PC} \leftarrow \mathbf{PC} + 4$$

```
load    R[rt]  $\leftarrow$  MEM[ R[rs] + sign_ext(imm16)];          PC  $\leftarrow$  PC + 4
```

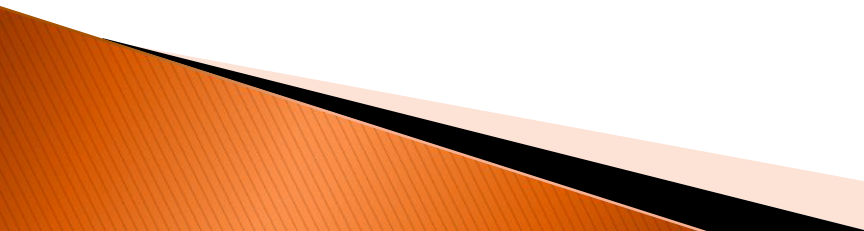
```
store  MEM[ R[rs] + sign_ext(imm16) ]  $\leftarrow$  R[rt];      PC  $\leftarrow$  PC + 4
```

```

beq    if ( R[rs] == R[rt] ) then PC  $\leftarrow$  PC + 4 + sign_ext(imm16)] || 00
          else PC  $\leftarrow$  PC + 4

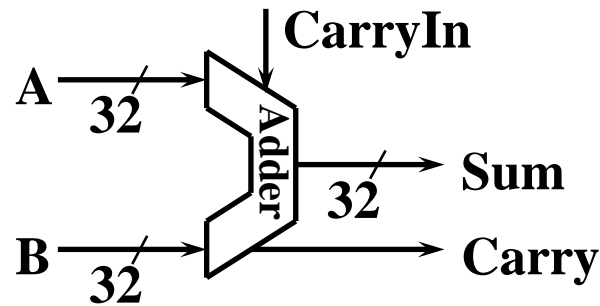
```

Step 1: Requirements of the Instruction Set

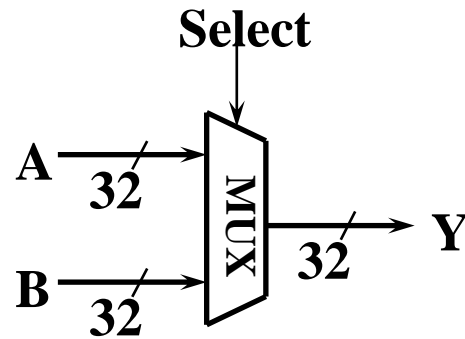
- ▶ Memory
 - instruction & data
 - ▶ Registers (32 x 32)
 - read rs
 - read rt
 - write rt or rd
 - ▶ PC
 - ▶ Sign extender
 - ▶ Add and sub register or extended immediate
 - ▶ Add 4 and/or shifted extended immediate to PC
- 

Step 2: Components of the Datapath

- ▶ Adder

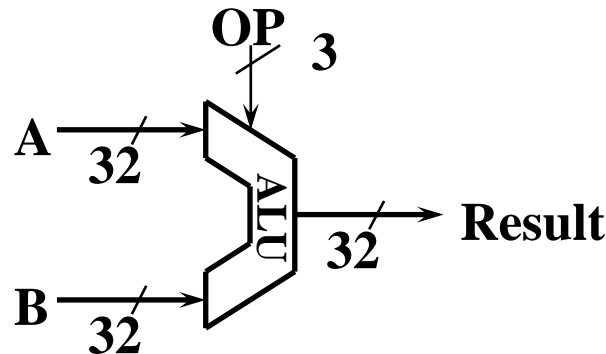


- ▶ MUX



Combinational Logic:
Does not use a clock

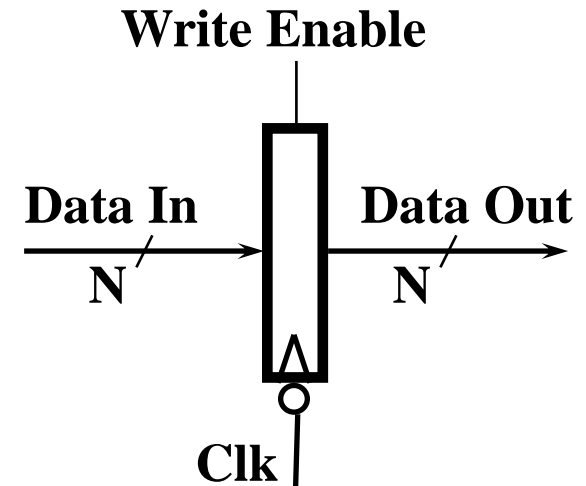
- ▶ ALU



Storage Element: Register (Basic Building Blocks)

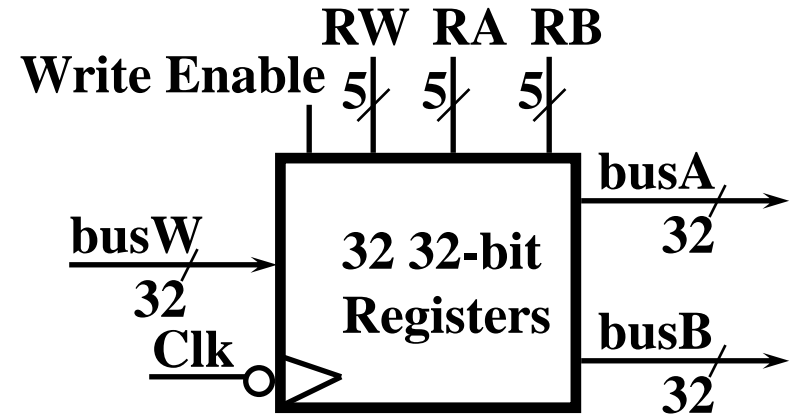
▶ Register

- Similar to the D Flip Flop except
 - N-bit input and output
 - Write enable input
- Write Enable:
 - negated (0): Data Out will not change
 - asserted (1): Data Out will become Data In on the falling edge of the clock

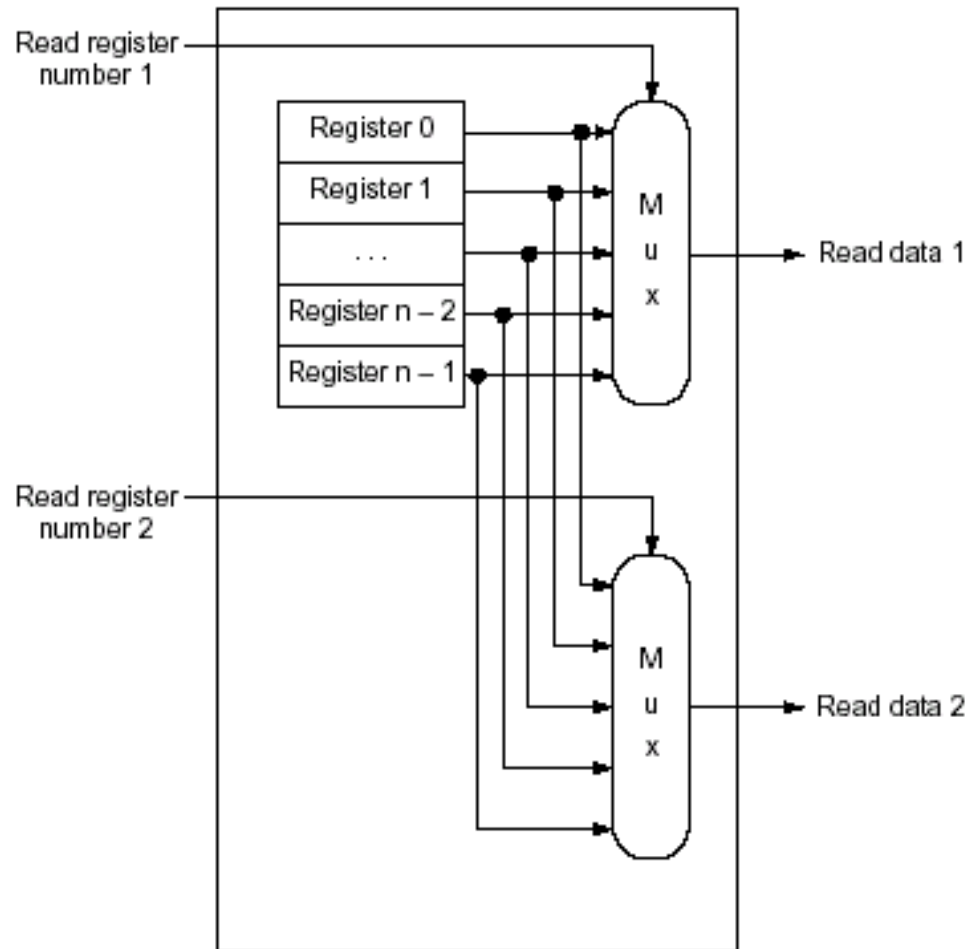


Storage Element: Register File

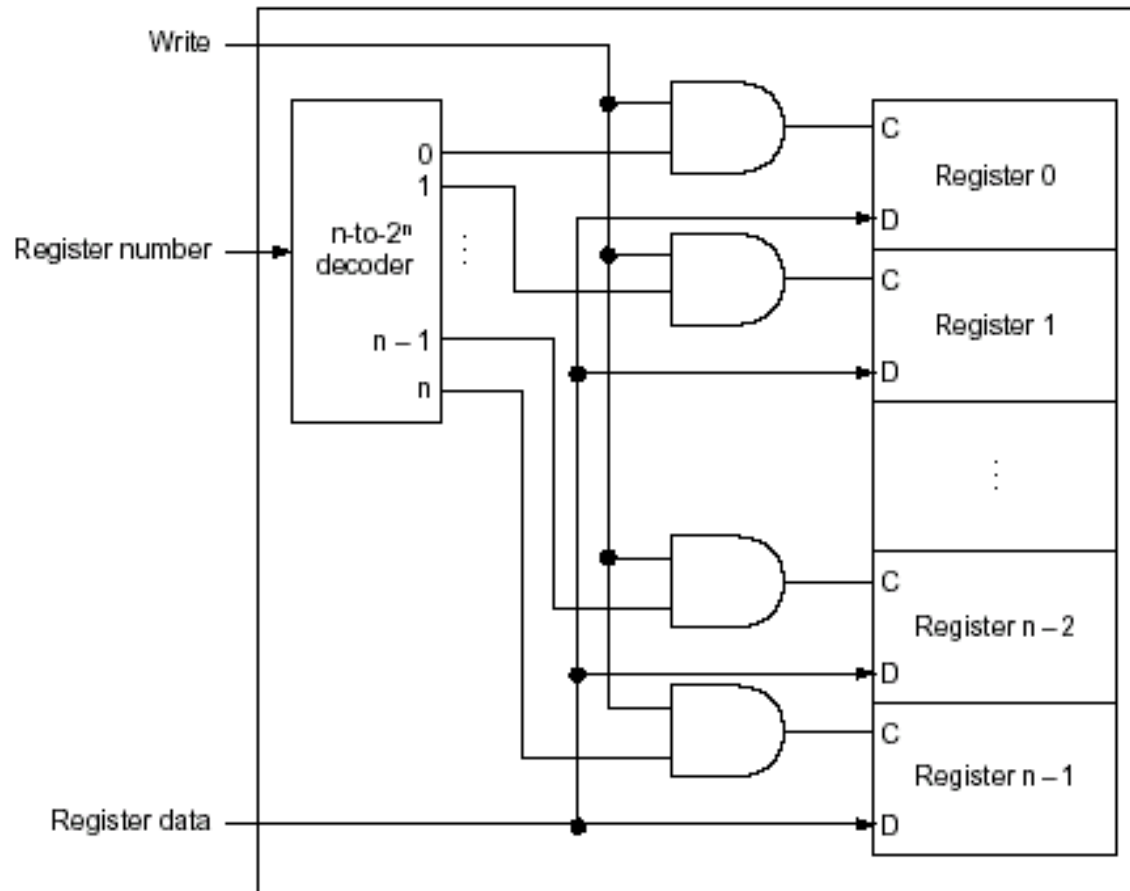
- ▶ Register File consists of 32 registers:
 - Two 32-bit output busses: busA and busB
 - One 32-bit input bus: busW
- ▶ Register is selected by:
 - RA (number) selects the register to put on busA (data)
 - RB (number) selects the register to put on busB (data)
 - RW (number) selects the register to 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.”



Read from Register File

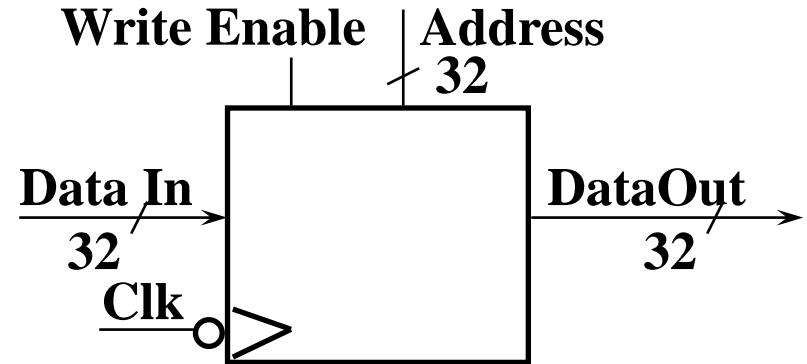


Write to Register File



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, memory behaves as a combinational logic block:
 - Address valid \Rightarrow Data Out valid after “access time.”

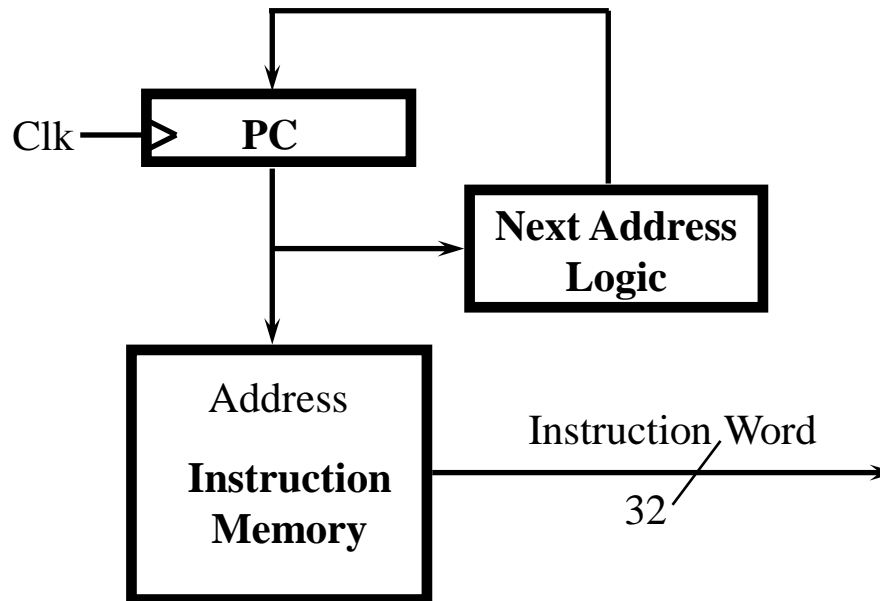


Step 3

- ▶ Register Transfer Requirements -> Datapath Design
 - Instruction Fetch
 - Decode instructions and Read Operands
 - Execute Operation
 - Write back the result

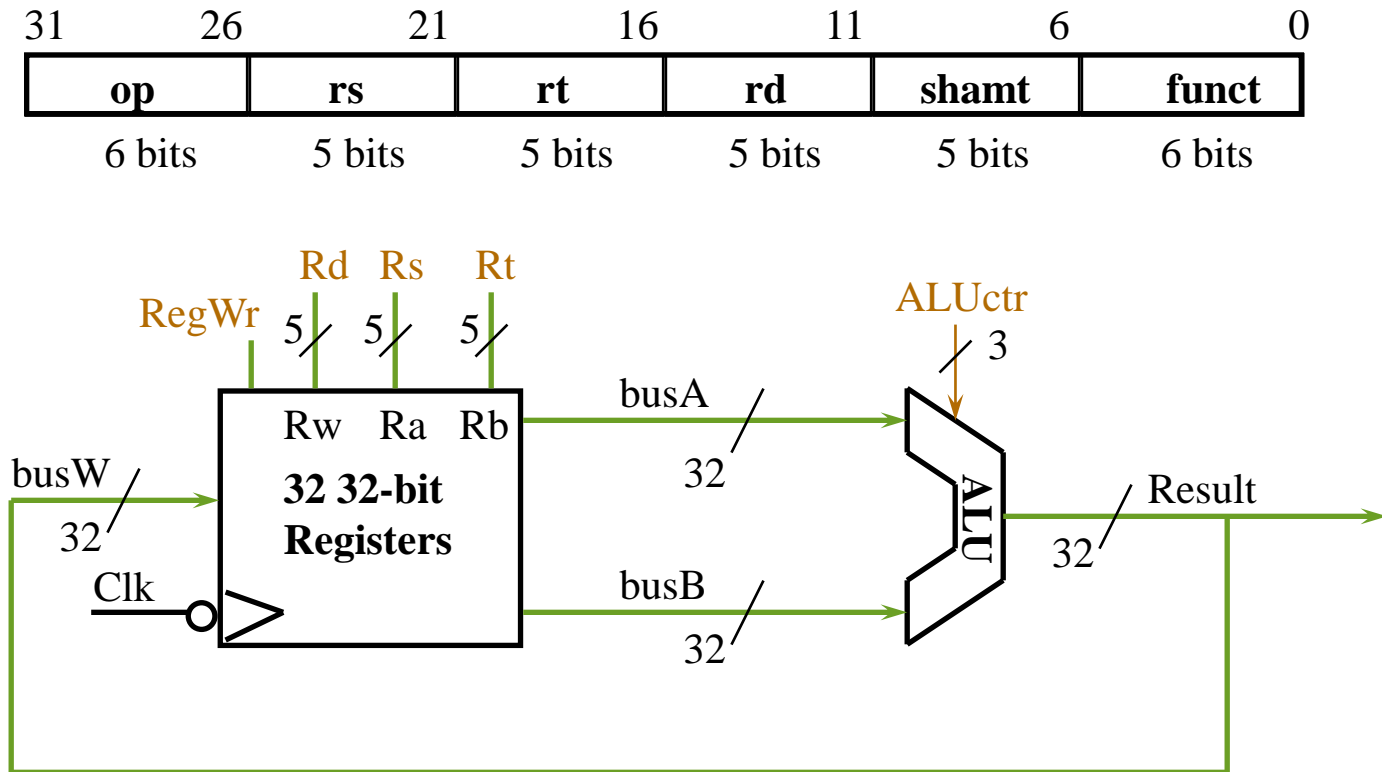
3a: Overview of the Instruction Fetch Unit

- ▶ The common RTL operations
 - Fetch the Instruction: $\text{mem}[\text{PC}]$
 - Update the program counter:
 - Sequential Code: $\text{PC} \leftarrow \text{PC} + 4$
 - Branch and Jump: $\text{PC} \leftarrow \text{"something else"}$



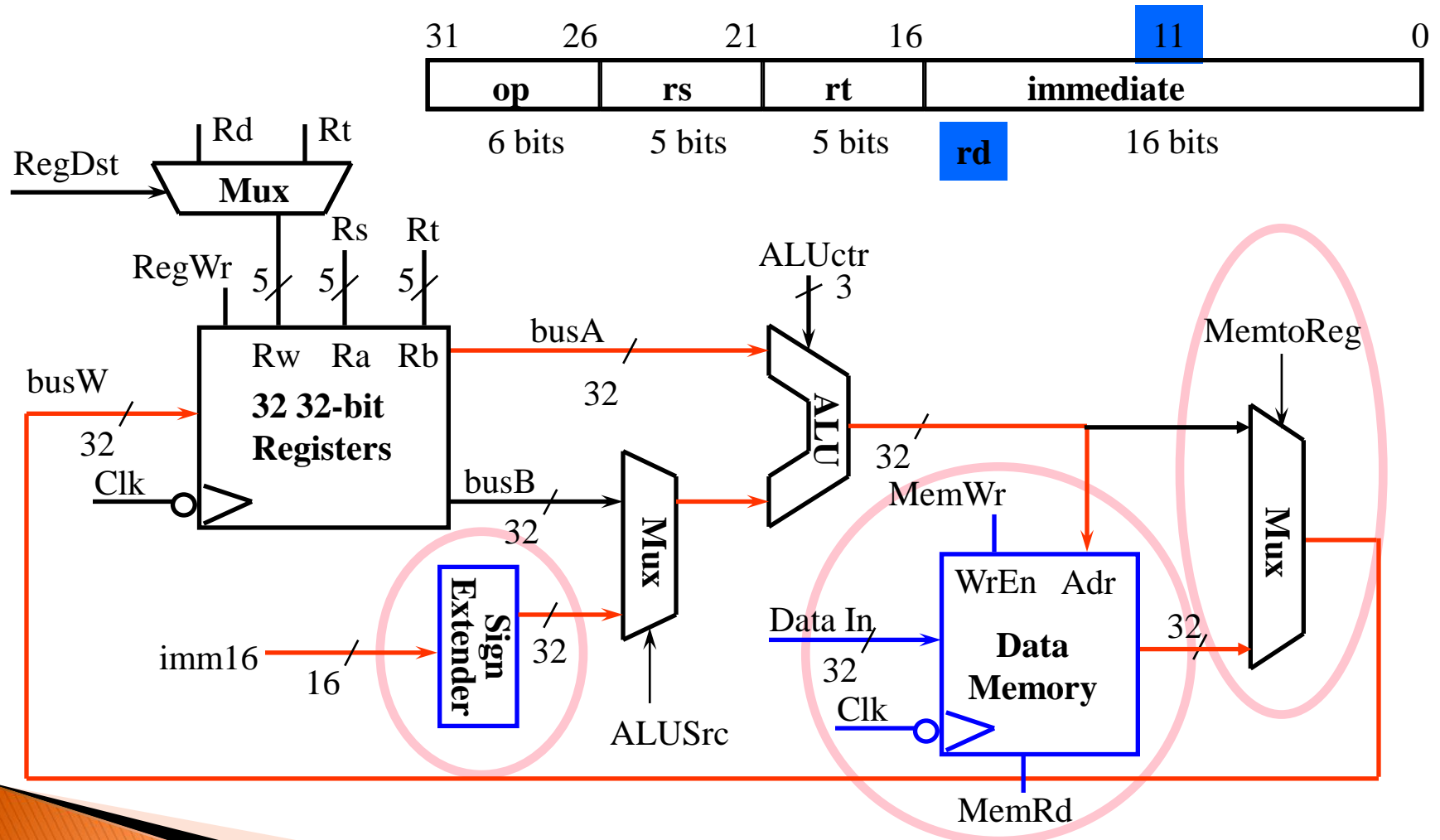
3b: R-Type Instructions

- ▶ $R[rd] \leftarrow R[rs] \text{ op } R[rt]$ Example: add rd, rs, rt
 - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
 - ALUctr and RegWr: control logic after decoding the instruction



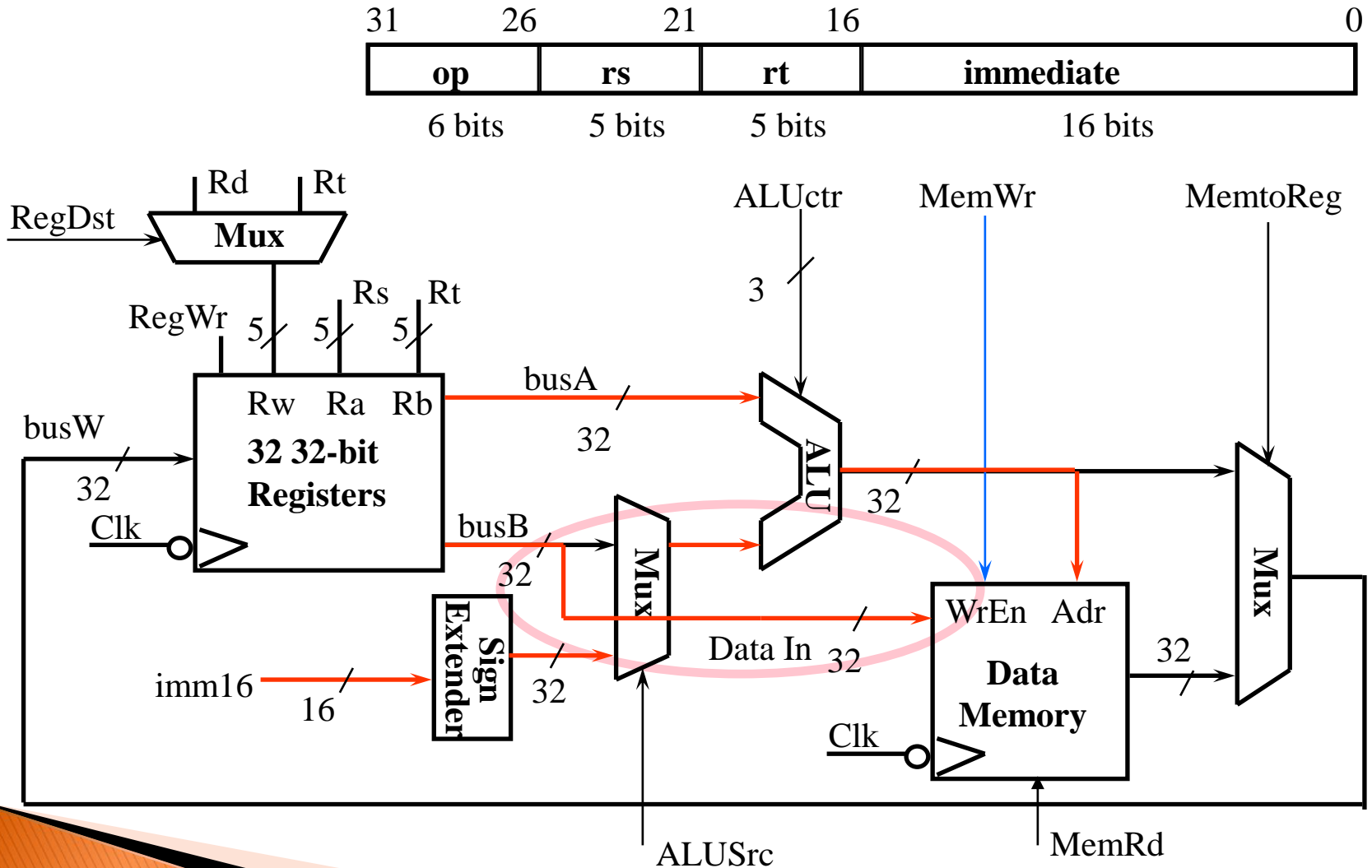
3c: Load Operations

- ▶ $R[\underline{rt}] \leftarrow \text{Mem}[R[\text{rs}] + \text{SignExt}[\text{imm16}]]$ Example: lw rt, rs, imm16

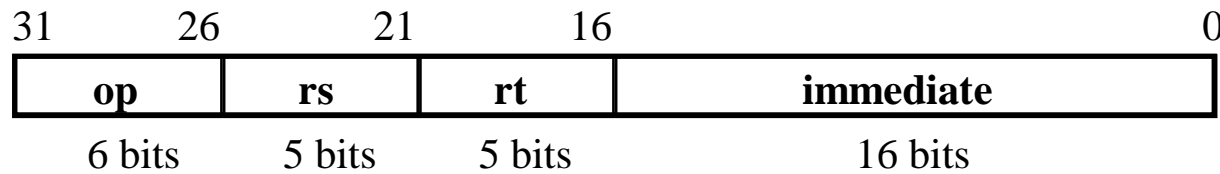


3d: Store Operations

- $\text{Mem}[\text{R}[\text{rs}] + \text{SignExt}[\text{imm16}]] \leftarrow \text{R}[\text{rt}]$ Example: `sw rt, rs, imm16`



3e: The Branch Instruction

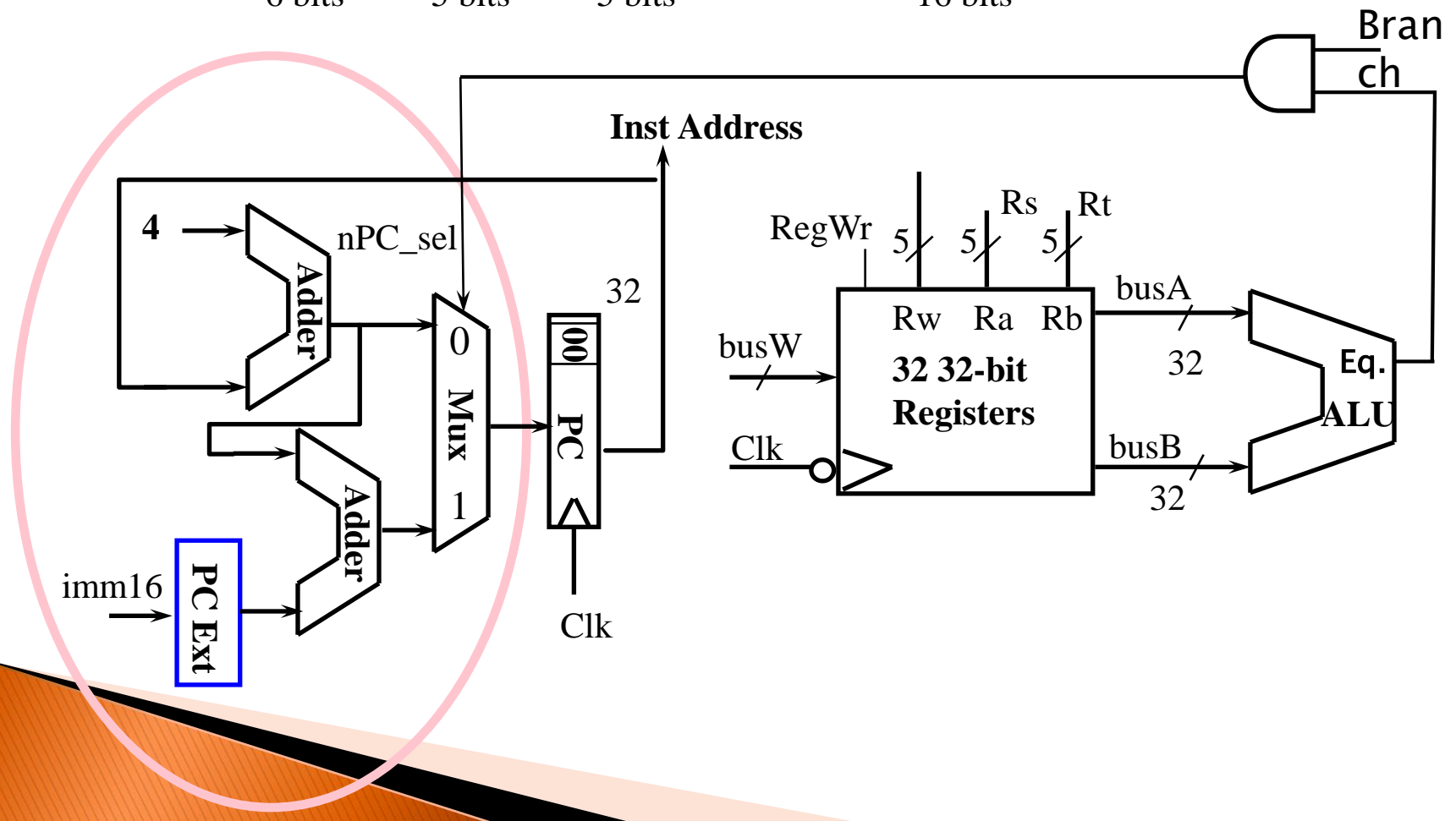
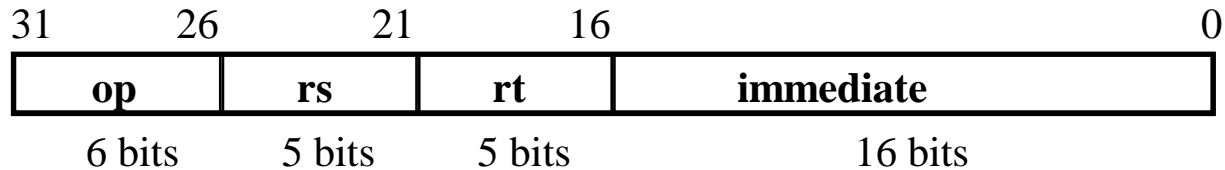


► **beq** rs, rt, imm16

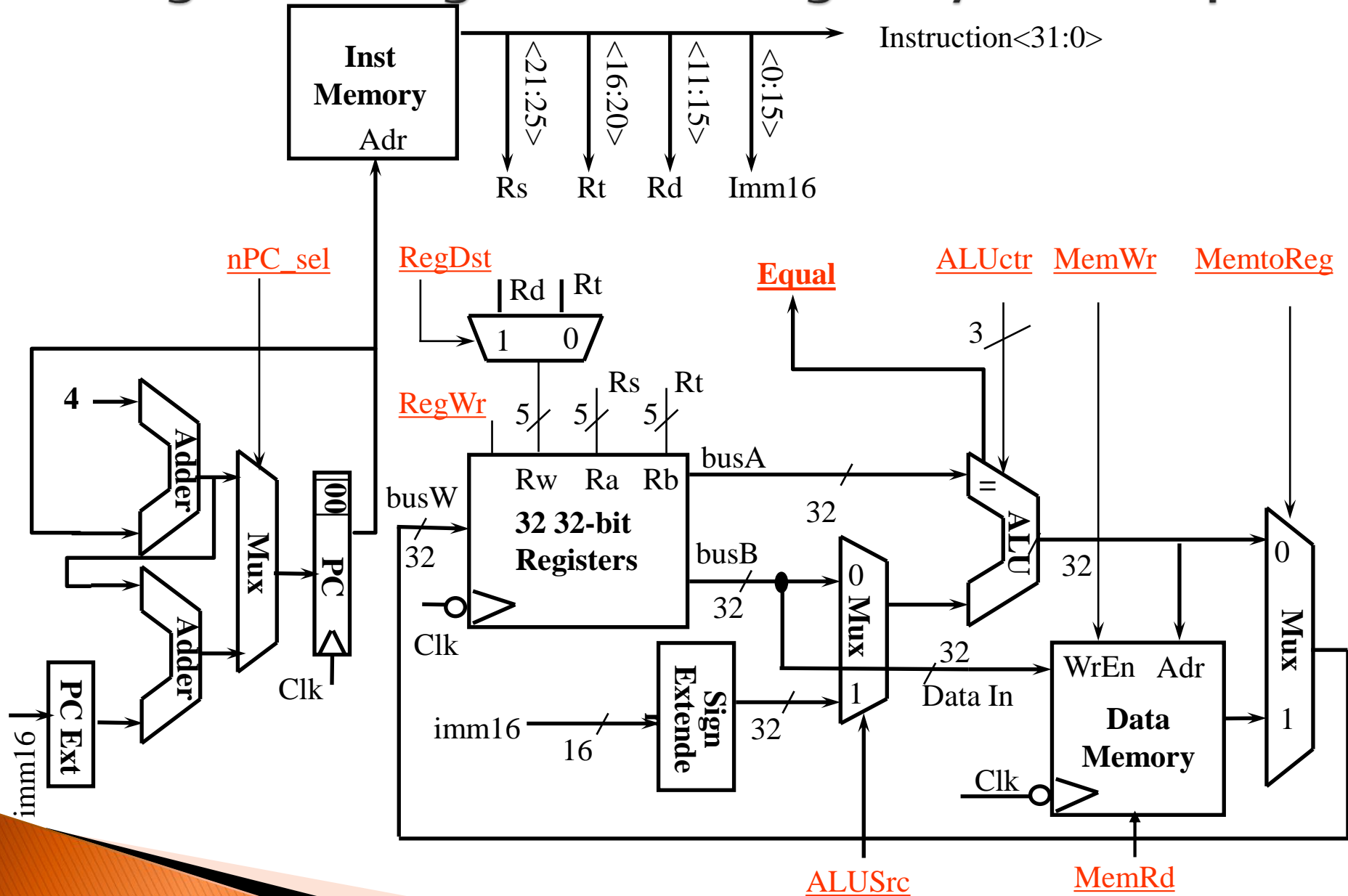
- **mem[PC]** Fetch the instruction from memory
- **Equal $\leftarrow R[rs] == R[rt]$** Calculate the branch condition
- **if (Equal && Branch Instr.)** Calculate the next instruction's address
 - **$PC \leftarrow PC + 4 + (\text{SignExt}(\text{imm16}) \times 4)$**
- else**
 - **$PC \leftarrow PC + 4$**

Datapath for Branch Operations

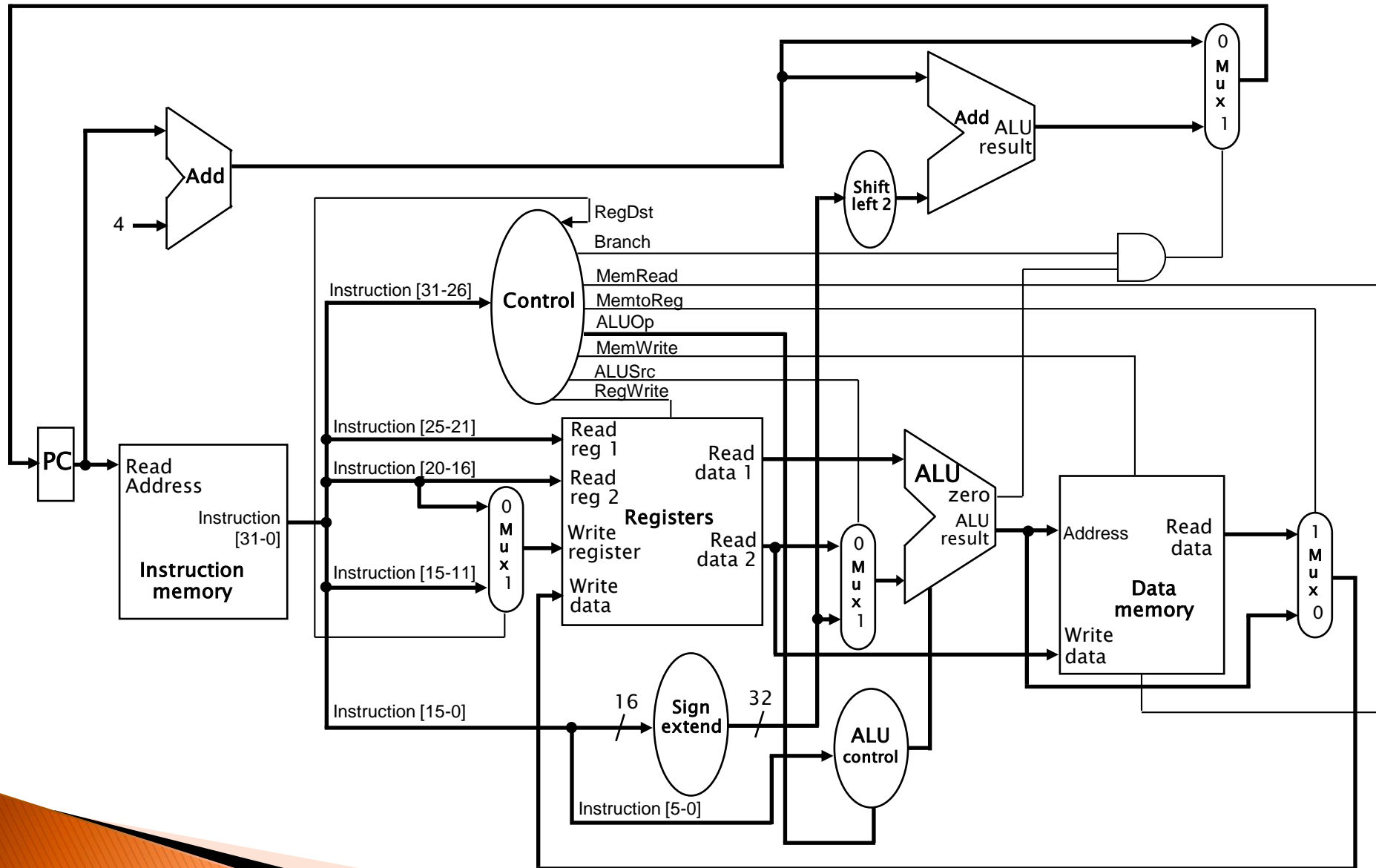
► beq rs, rt, imm16



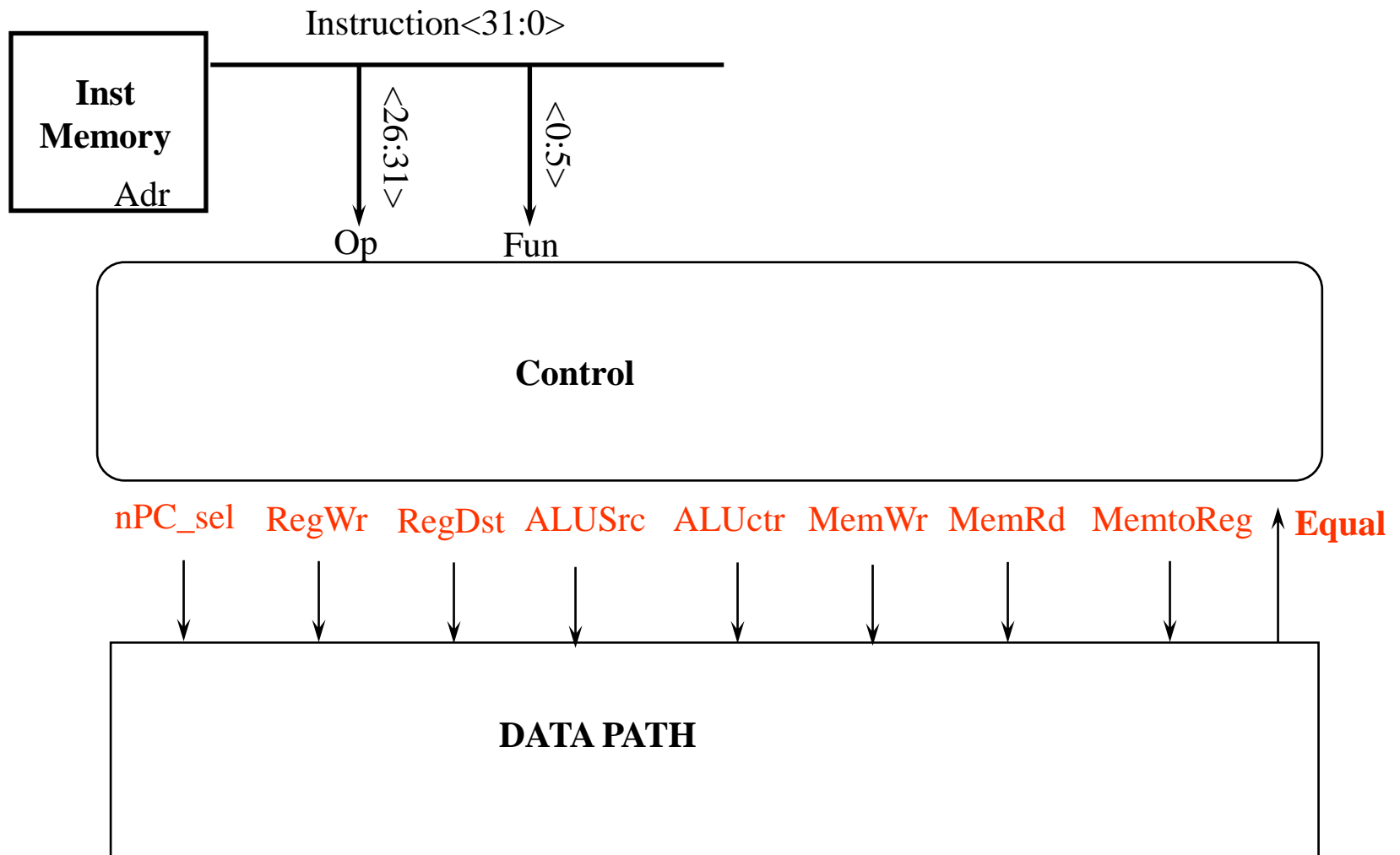
Putting it All Together: A Single Cycle Datapath



Different View of Same Implementation (From Book)

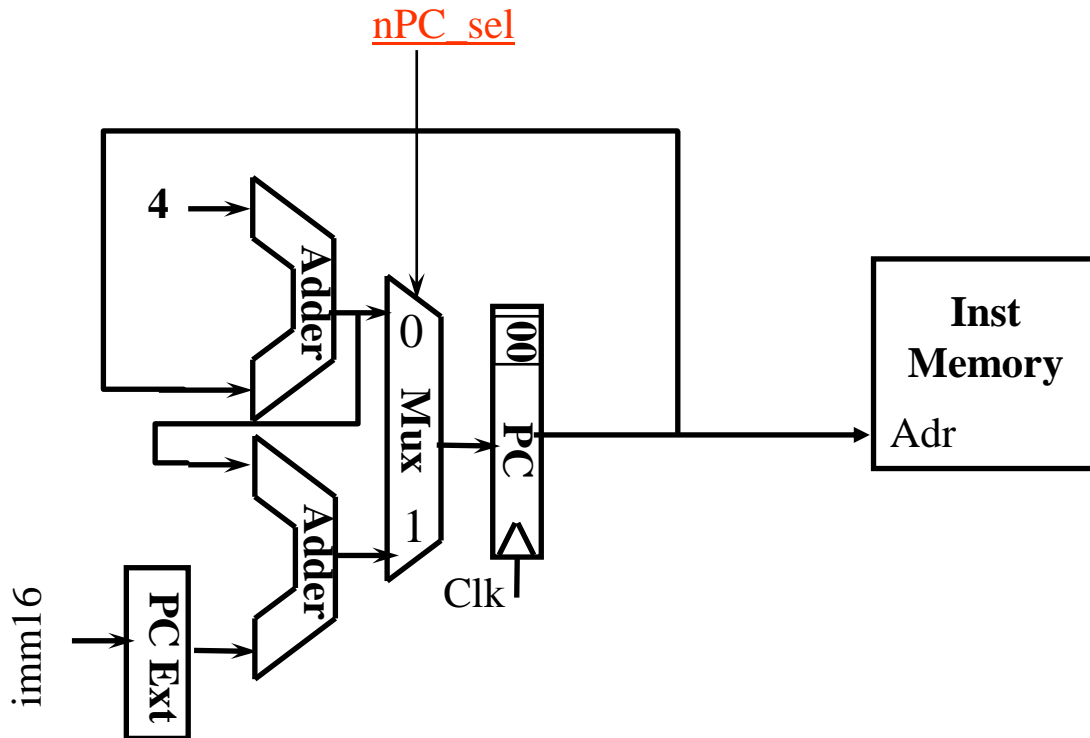


Step 4: Given Datapath: RTL \rightarrow Control



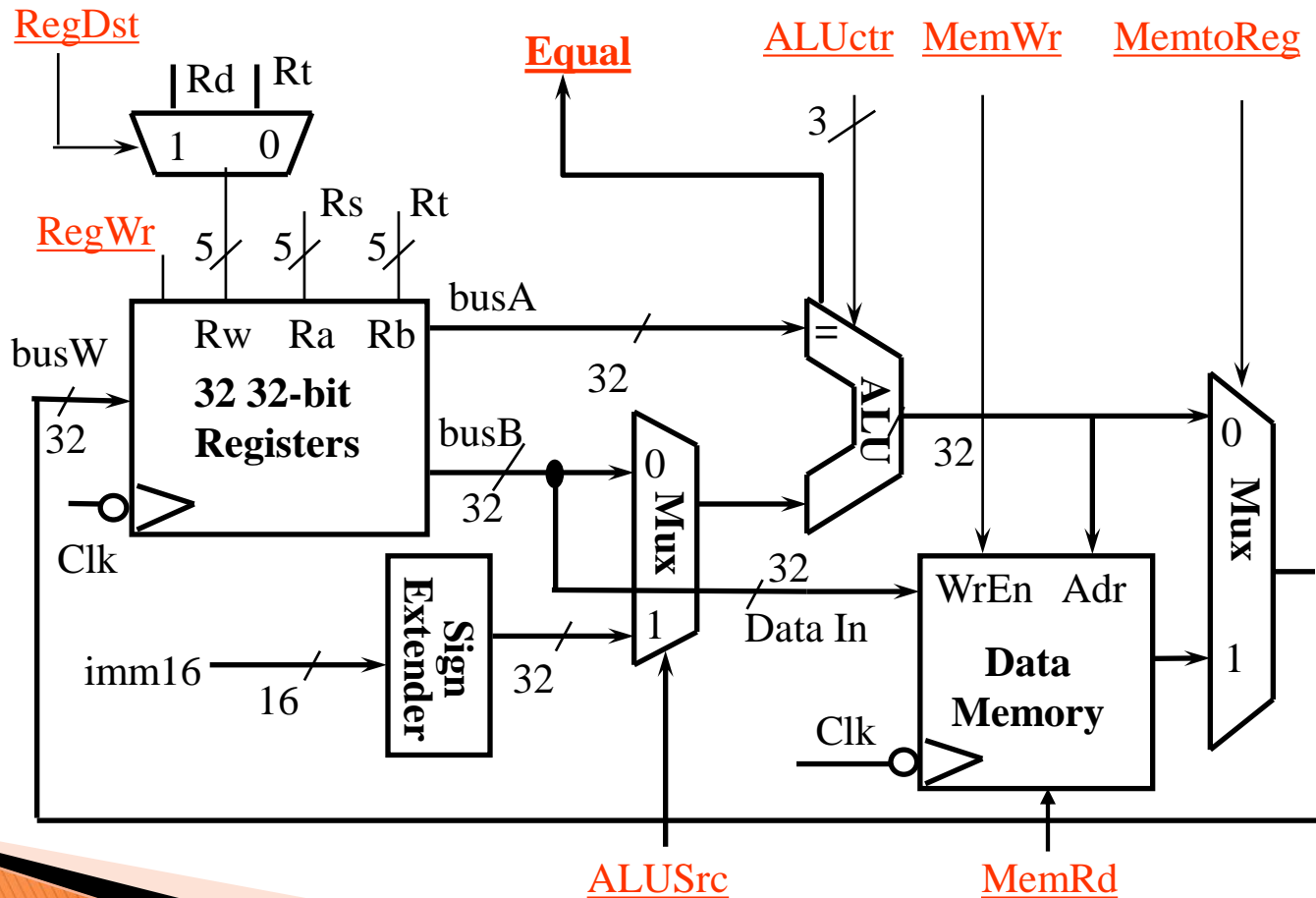
Meaning of the Control Signals

- ▶ Rs, Rt, Rd and Imm16 hardwired into datapath
- ▶ nPC_sel: $0 \Rightarrow PC \leftarrow PC + 4;$
 $1 \Rightarrow PC \leftarrow PC + 4 + \text{SignExt}(\text{Imm16}) \parallel 00$

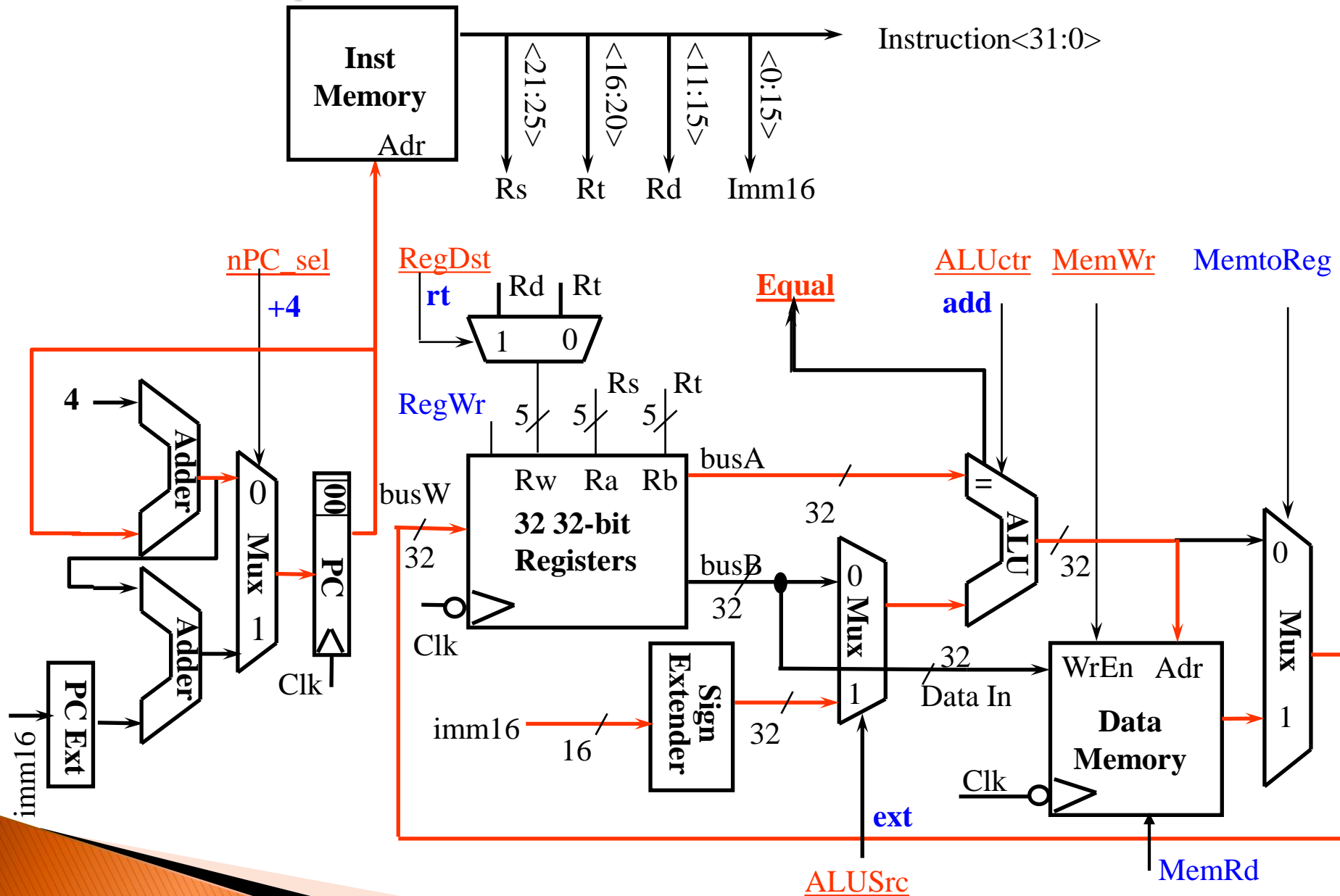


Meaning of the Control Signals

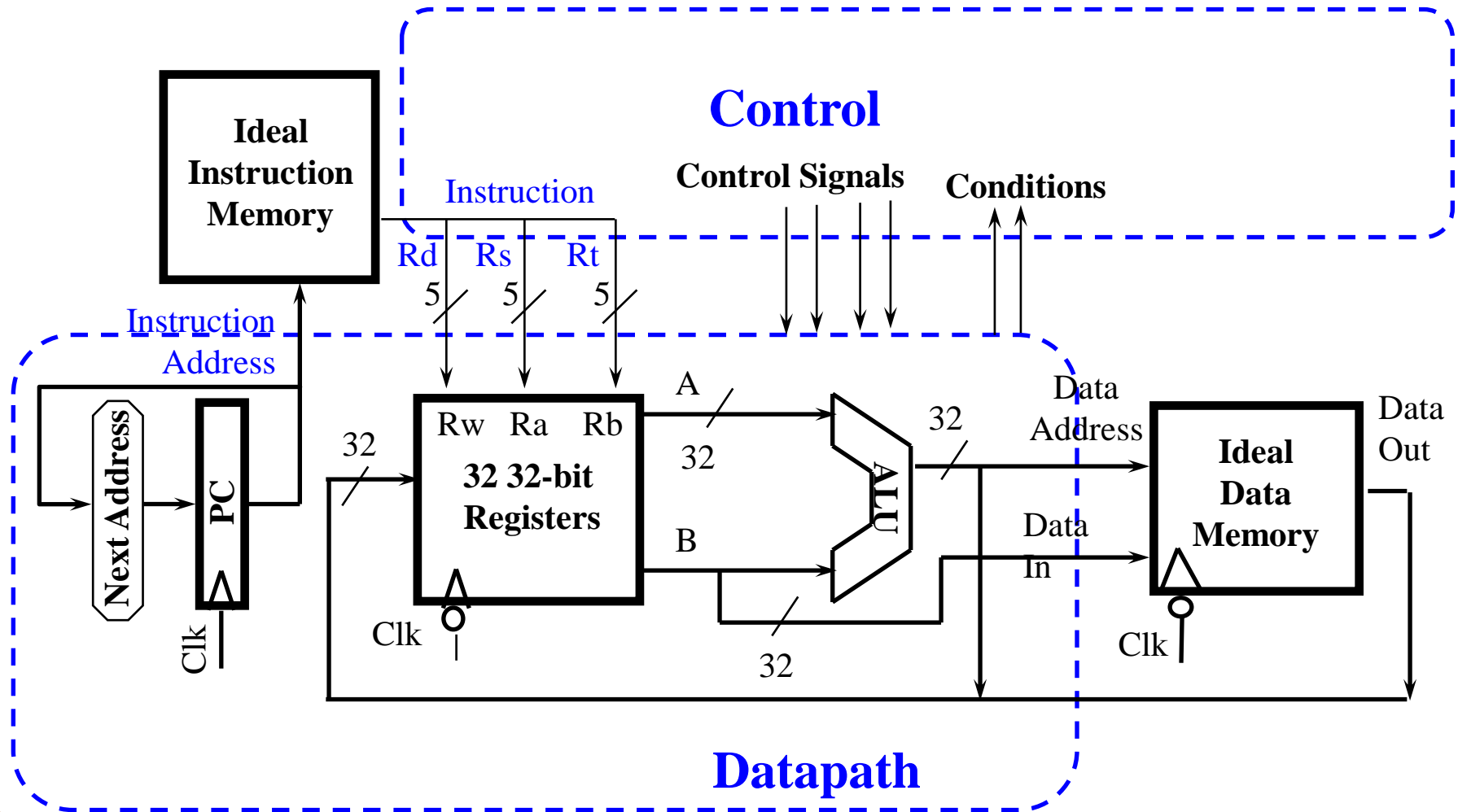
- ▶ **ALUSrc:** 0 => regB; 1 => immedi
- ▶ **MemRd:** read memory
- ▶ **MemWr:** write memory
- ▶ **ALUctr:** “add”, “sub”, “and”, “or”, “set less than”
- **RegWr:** write dest register
- **MemtoReg:** 0 => ALU; 1 => Mem
- **RegDst:** 0 => “rt”; 1 => “rd”



Example: Load Instruction



An Abstract View of the Implementation



Summary

- ▶ 5 steps to design a processor
 - 1. Analyze instruction set => datapath requirements
 - 2. Select set of datapath components & establish clock methodology
 - 3. Design datapath meeting the requirements
 - 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
 - 5. Design the control logic

- ▶ MIPS makes it easier
 - Instructions same size
 - Source registers always in same place
 - Immediates same size, location
 - Operations always on registers/immediates

- ▶ Single cycle datapath => $CPI=1$, $CCT \Rightarrow$ long