

CPS104 Computer Organization

Lecture 17

Building a Data-path; Adding Control

October 26 , 2009

Gershon Kedem

Administratrivia

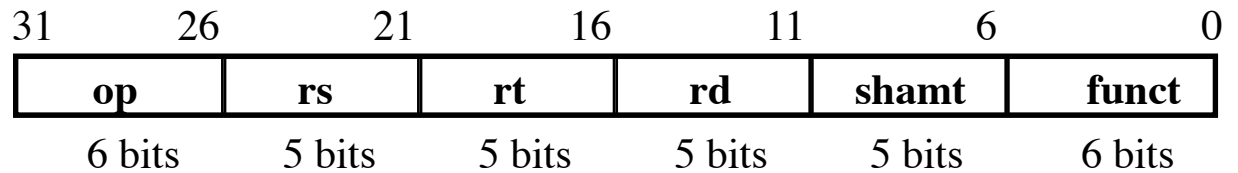
- Homework-4 Posted; **Due today in class**
- Extra Credit Homework is posted; **Due today**

- Reading: Chapter 4.
- Reading: Appendix C
 - ◆ Available on: . . . cps104/Handouts/Appendix-C.pdf

Review: The MIPS Subset (We can't do them all!)

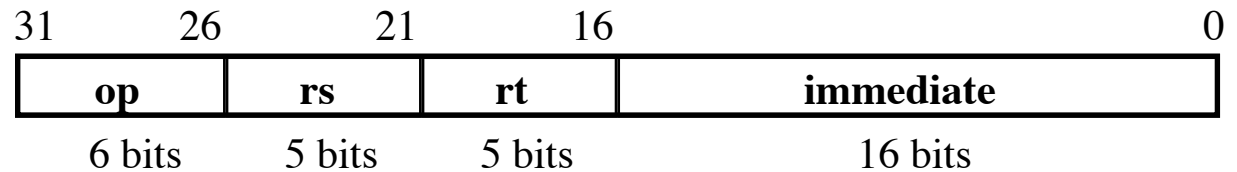
- **ADD and subtract**

- ◆ add rd, rs, rt
- ◆ sub rd, rs, rt



- **OR Immediate:**

- ◆ ori rt, rs, imm16



- **LOAD and STORE**

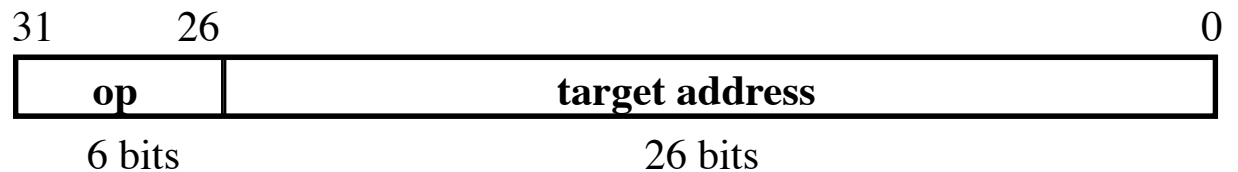
- ◆ lw rt, rs, imm16
- ◆ sw rt, rs, imm16

- **BRANCH:**

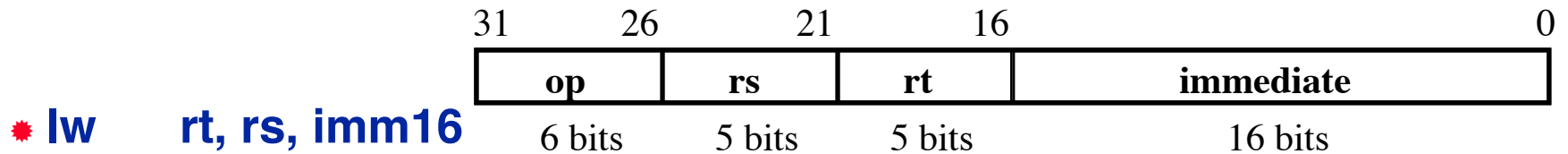
- ◆ beq rs, rt, imm16

- **JUMP:**

- ◆ j target



Review: RTL: The Load Instruction



◆ **mem[PC]** Fetch the instruction from memory

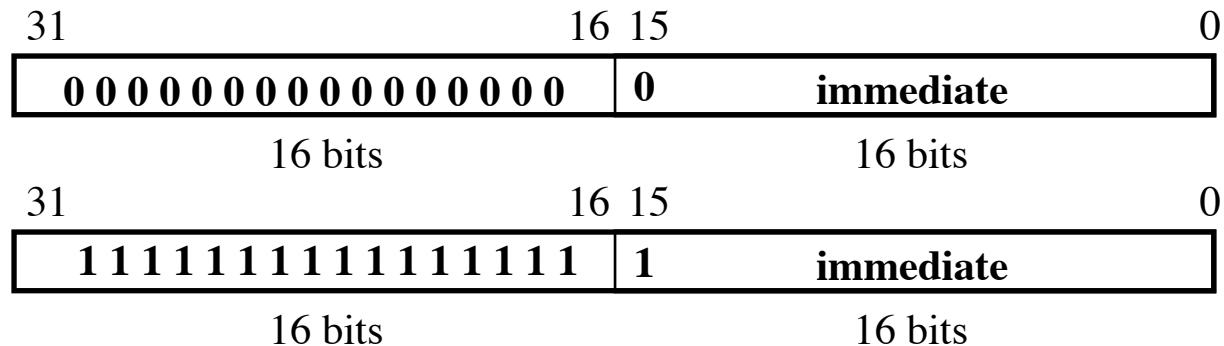
◆ **Addr ← R[rs] + SignExt(imm16)**

Calculate the memory address

◆ **R[rt] ← Mem[Addr]** Load the data into the register

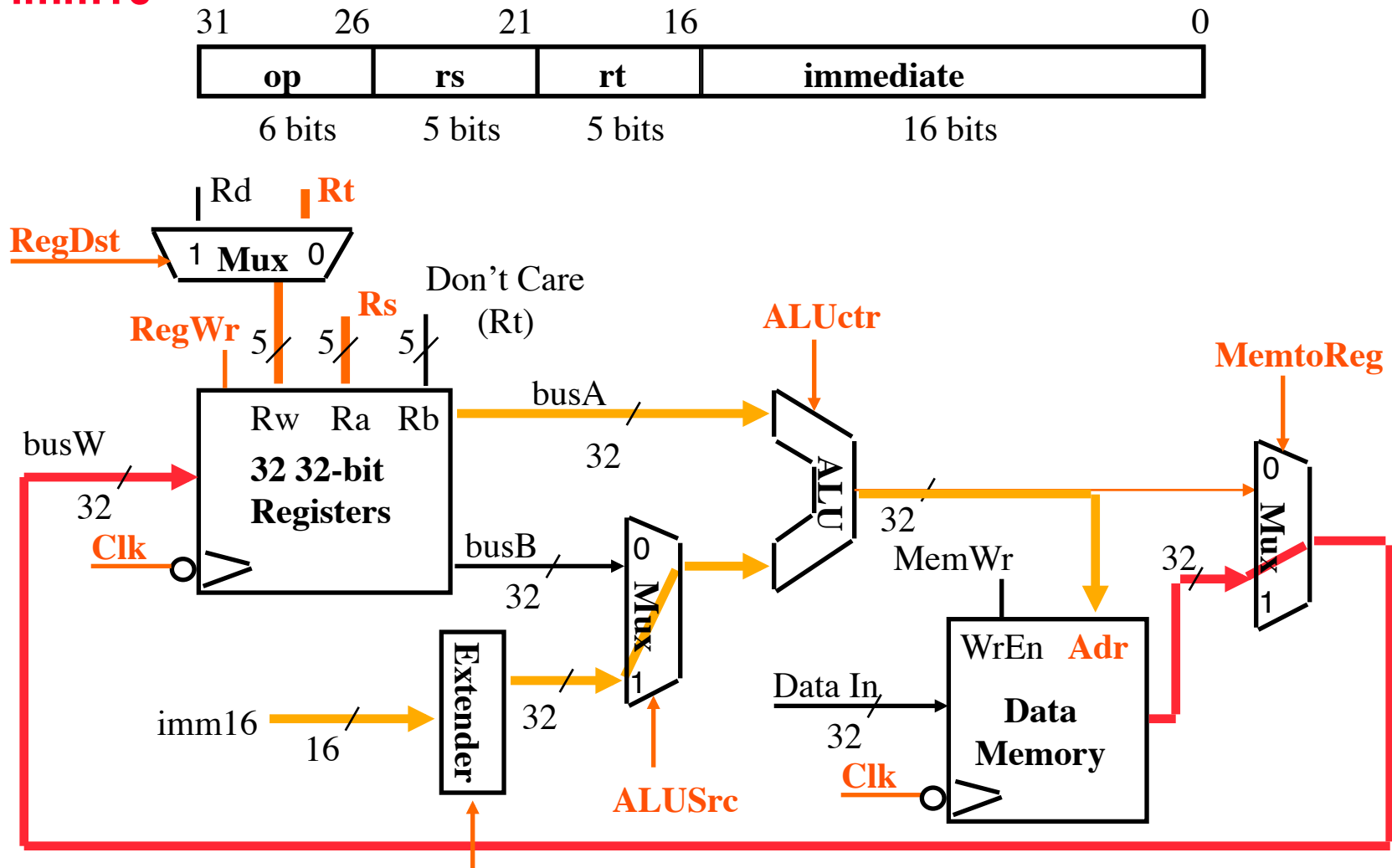
◆ **PC ← PC + 4**

Calculate the next instruction's address

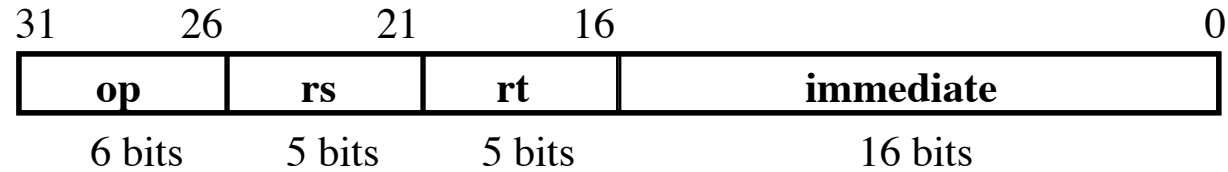


Datapath for Load Operations

- $R[rt] \leftarrow Mem[R[rs] + SignExt[imm16]]$
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

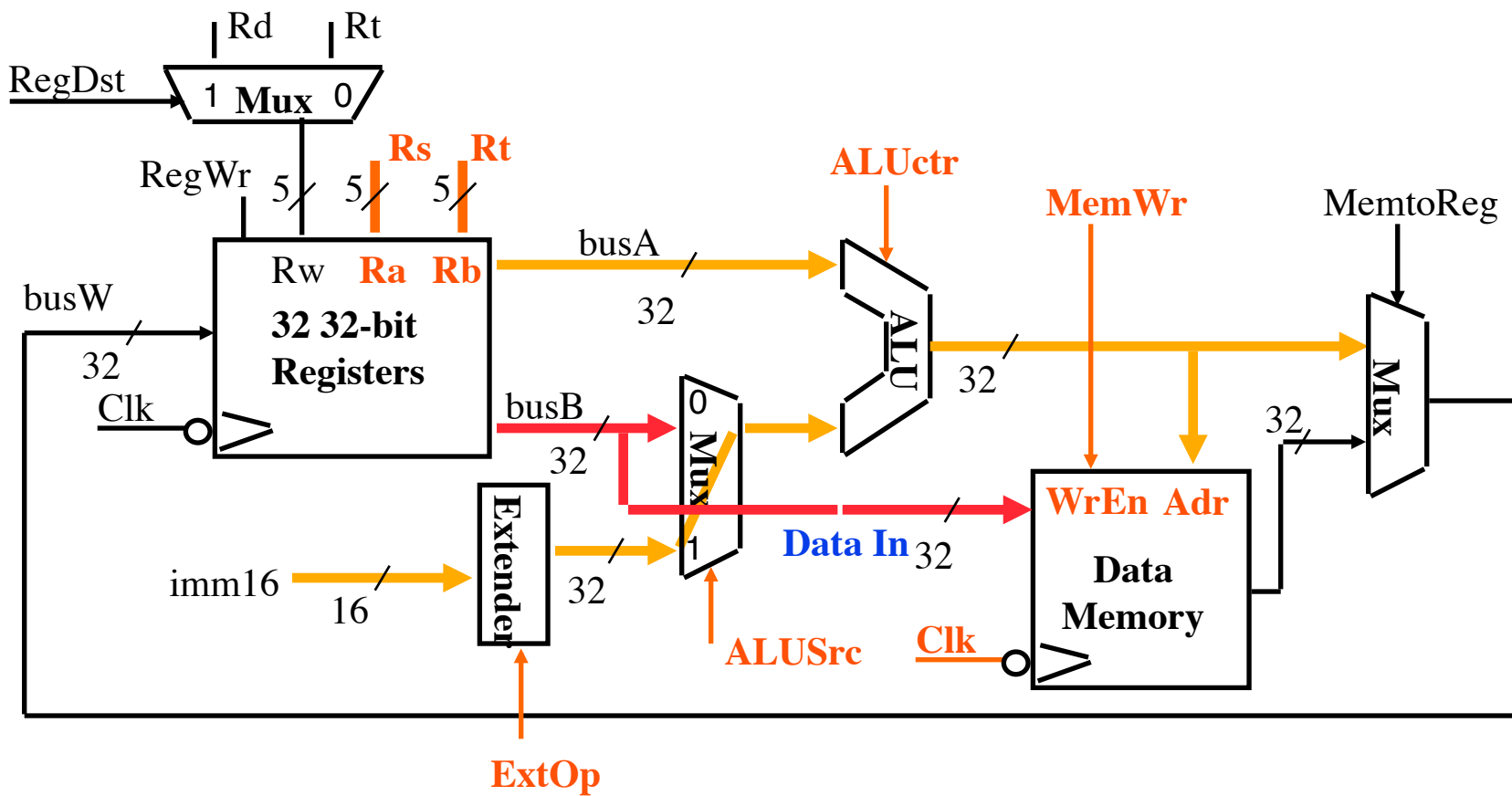
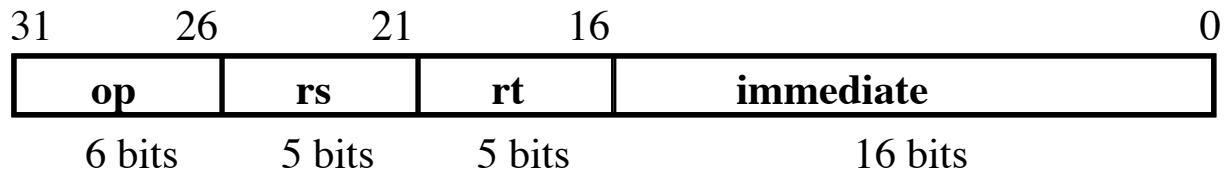
♦ **Mem[Addr]** <- **R[rt]** Store the register into memory

♦ **PC** <- **PC + 4**

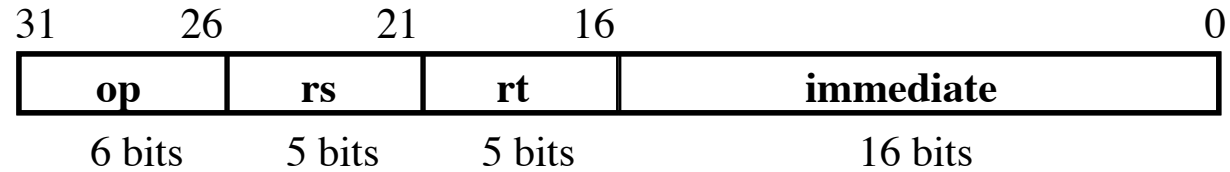
Calculate the next instruction's address

Datapath for Store Operations

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



RTL: The Branch Instruction



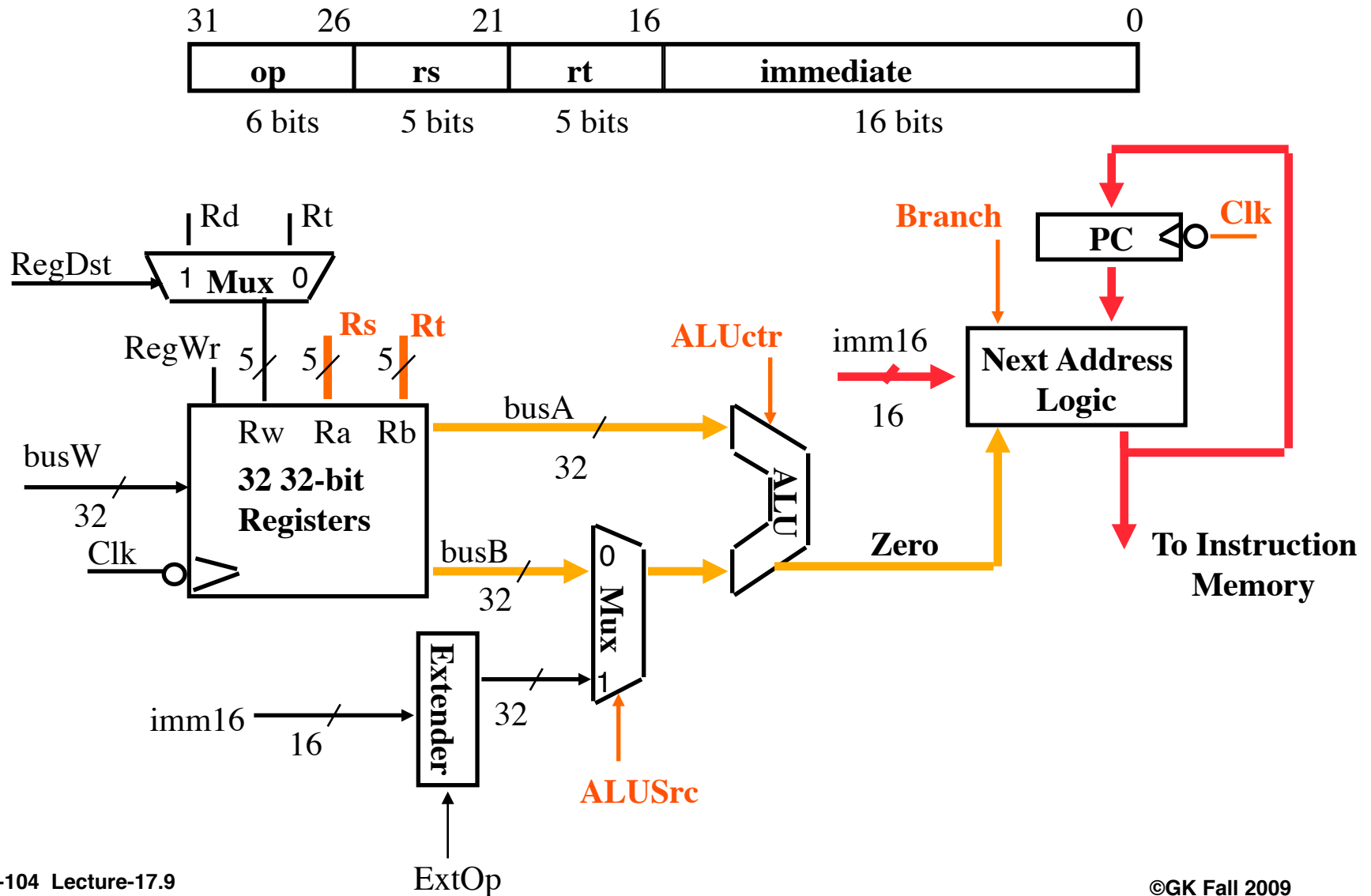
• **beq rs, rt, imm16**

- ◆ **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!



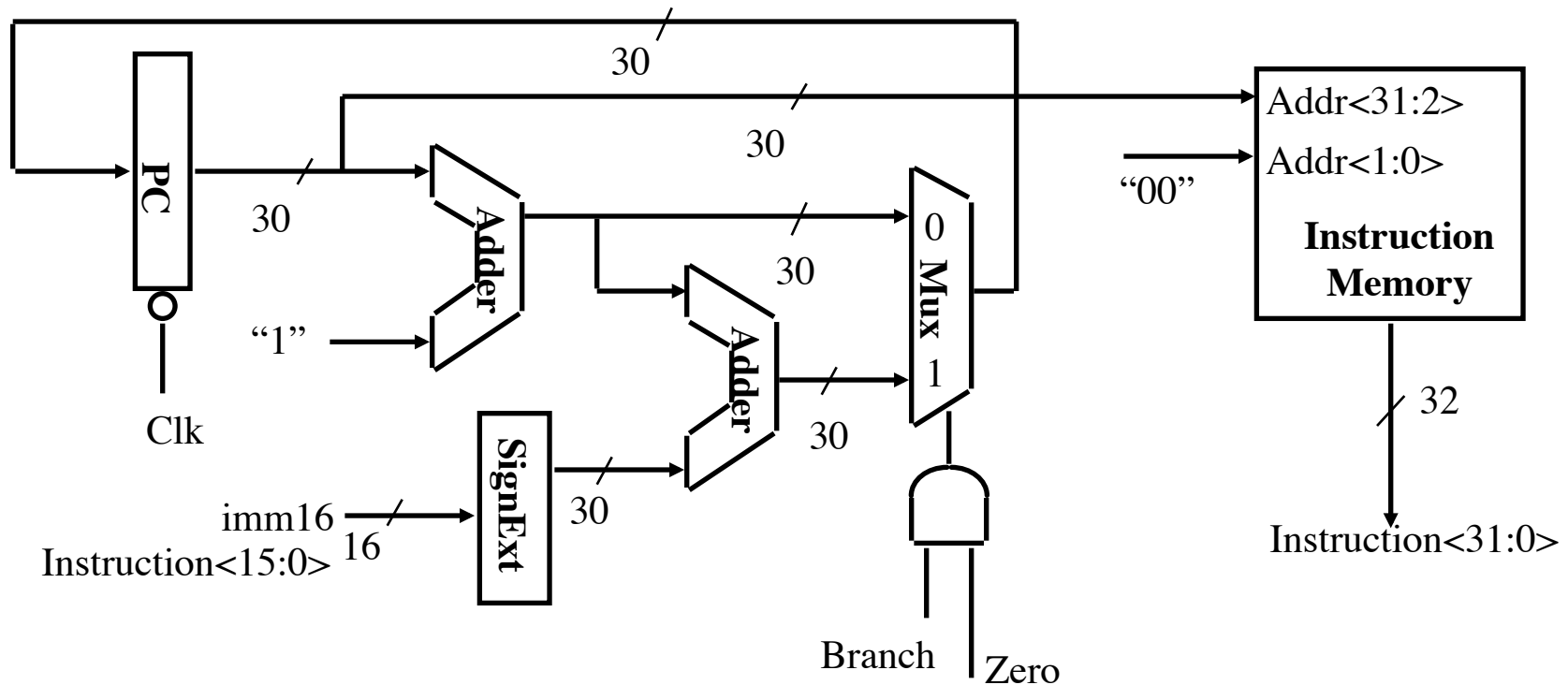
Binary Arithmetic for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
 - ◆ Sequential operation: $PC\langle 31:0 \rangle = PC\langle 31:0 \rangle + 4$
 - ◆ Branch operation: $PC\langle 31:0 \rangle = PC\langle 31:0 \rangle + 4 + \text{SignExt}[\text{Imm16}] * 4$
- 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 $PC\langle 31:2 \rangle$:
 - ◆ Sequential operation: $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1$
 - ◆ Branch operation: $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1 + \text{SignExt}[\text{Imm16}]$
 - ◆ In either case: $\text{Instruction-Memory-Address} = PC\langle 31:2 \rangle \text{ concat } \text{“00”}$

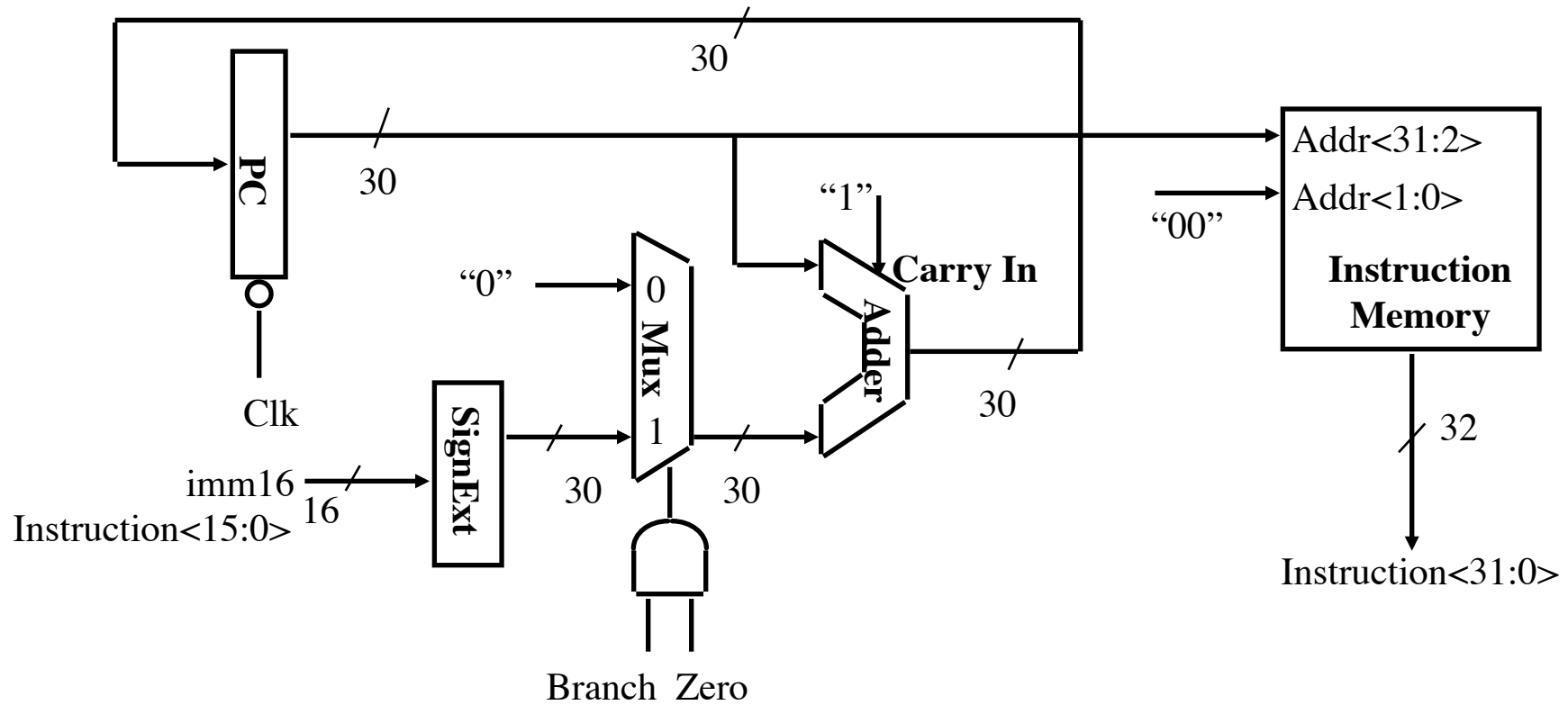
Next Address Logic: Expensive and Fast Solution

- Using a 30-bit PC:

- Sequential operation: $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1$
- Branch operation: $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1 + \text{SignExt}[\text{Imm16}]$
- In either case: $\text{Instruction-Memory-Address} = PC\langle 31:2 \rangle \text{ concat } "00"$



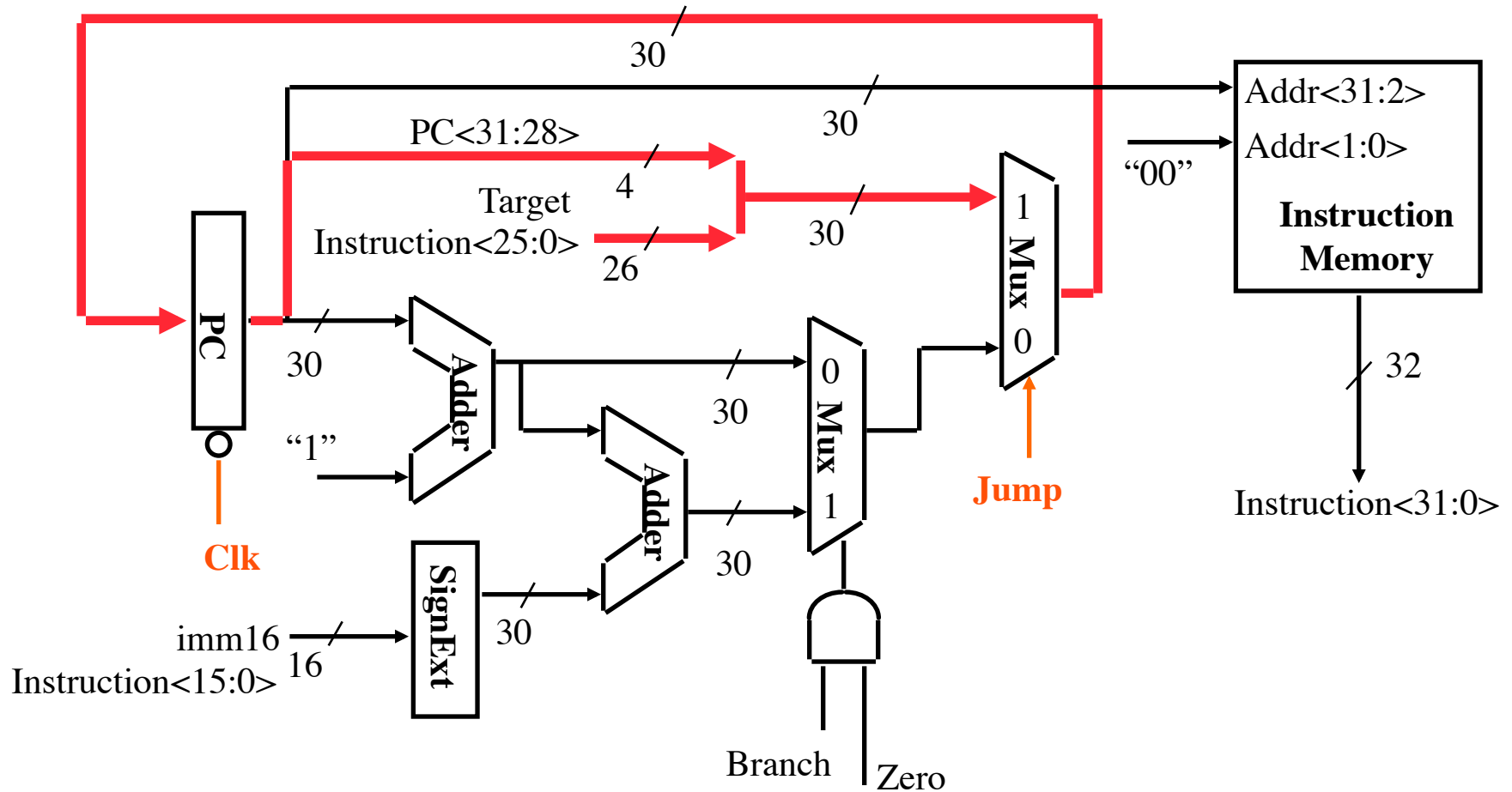
Next Address Logic



Instruction Fetch Unit

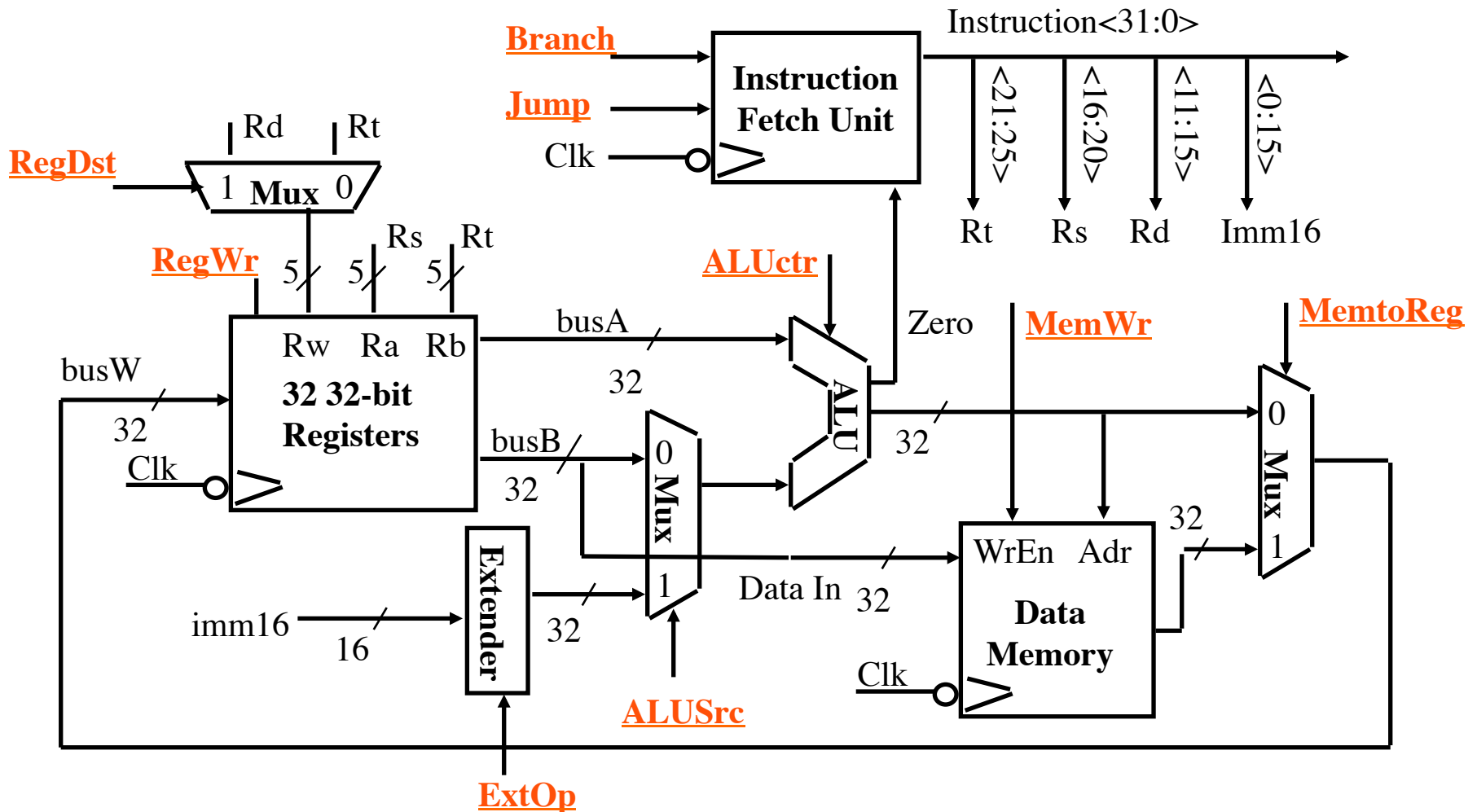
• j target

◆ $PC\langle 31:2 \rangle \leftarrow PC\langle 31:28 \rangle \text{ concat target}\langle 25:0 \rangle$



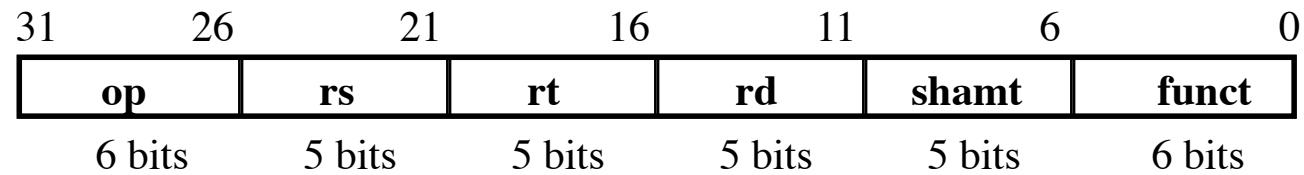
Putting it All Together: A Single Cycle Datapath

- We have everything except **control signals**.



What about Controls?

Review: RTL: The ADD Instruction



• **add rd, rs, rt**

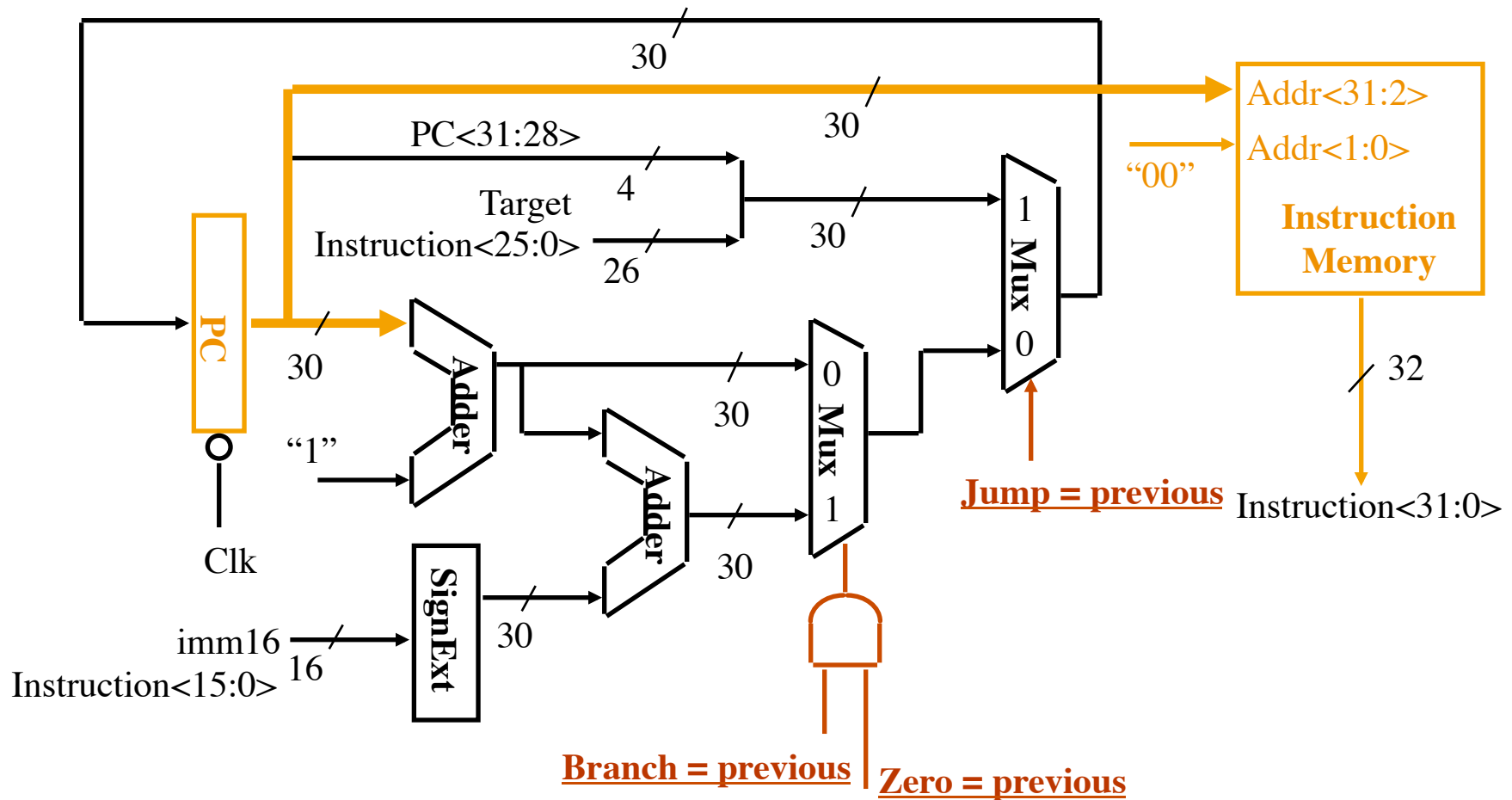
◆ **mem[PC]** **Fetch the instruction from memory**

◆ **$R[rd] \leftarrow R[rs] + R[rt]$** **The actual operation**

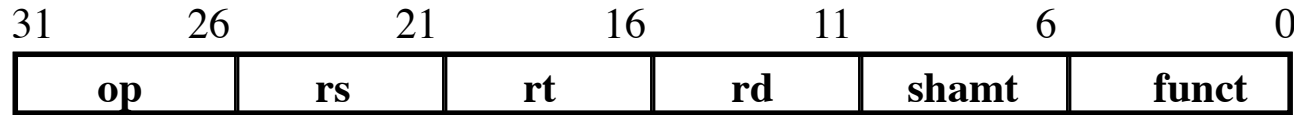
◆ **$PC \leftarrow PC + 4$** **Calculate the next instruction's address**

Instruction Fetch Unit at the Beginning of Add / Subtract

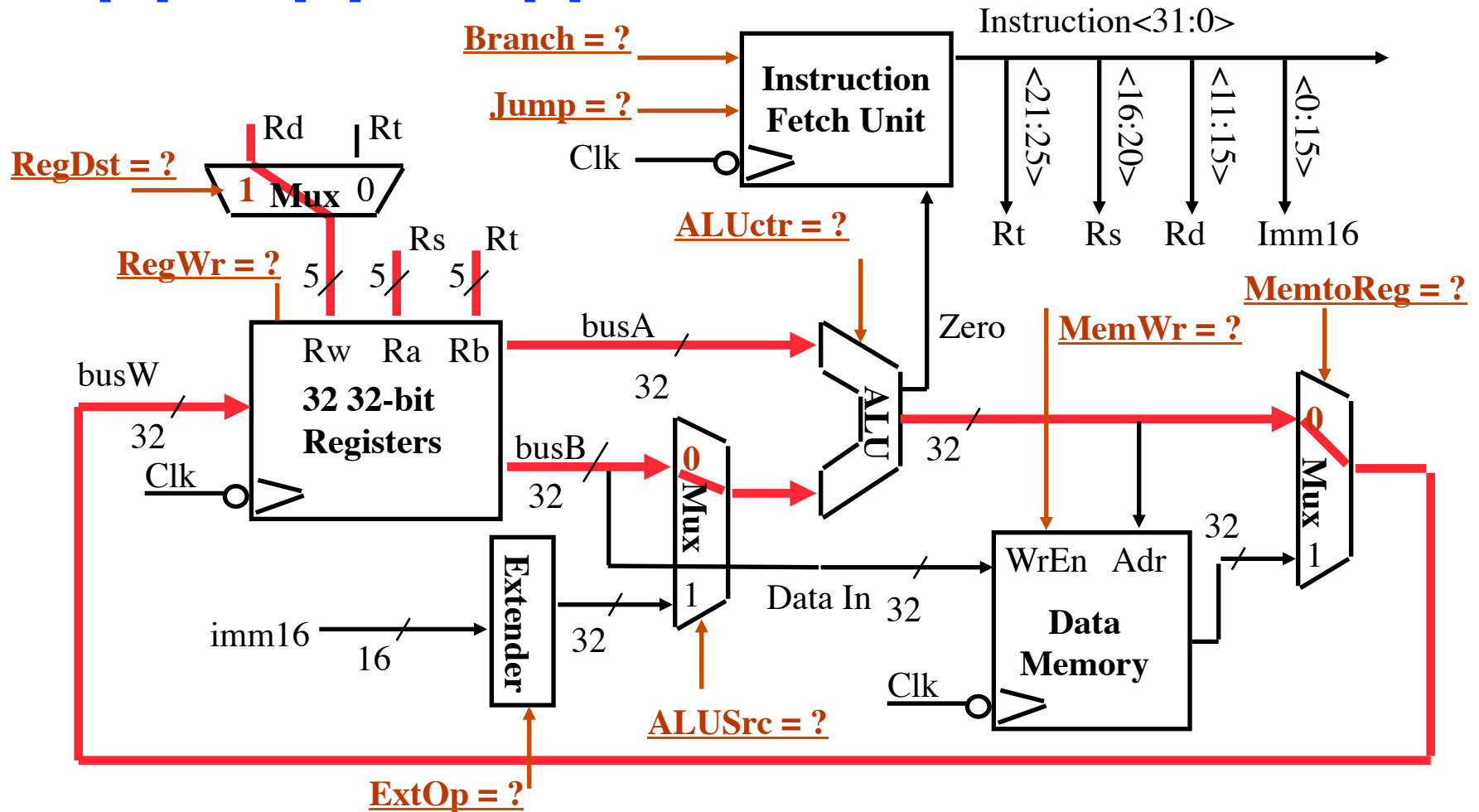
- Fetch the instruction from Instruction memory: $\text{Instruction} \leftarrow \text{mem}[\text{PC}]$
 - This is the same for all instructions



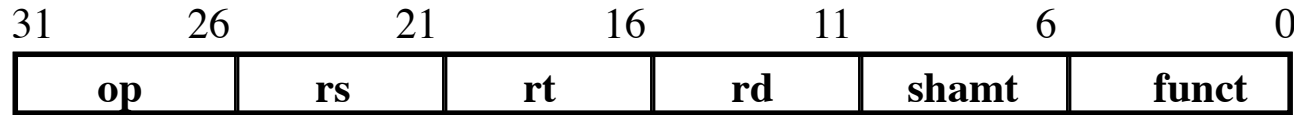
The Single Cycle Datapath during Add and Subtract



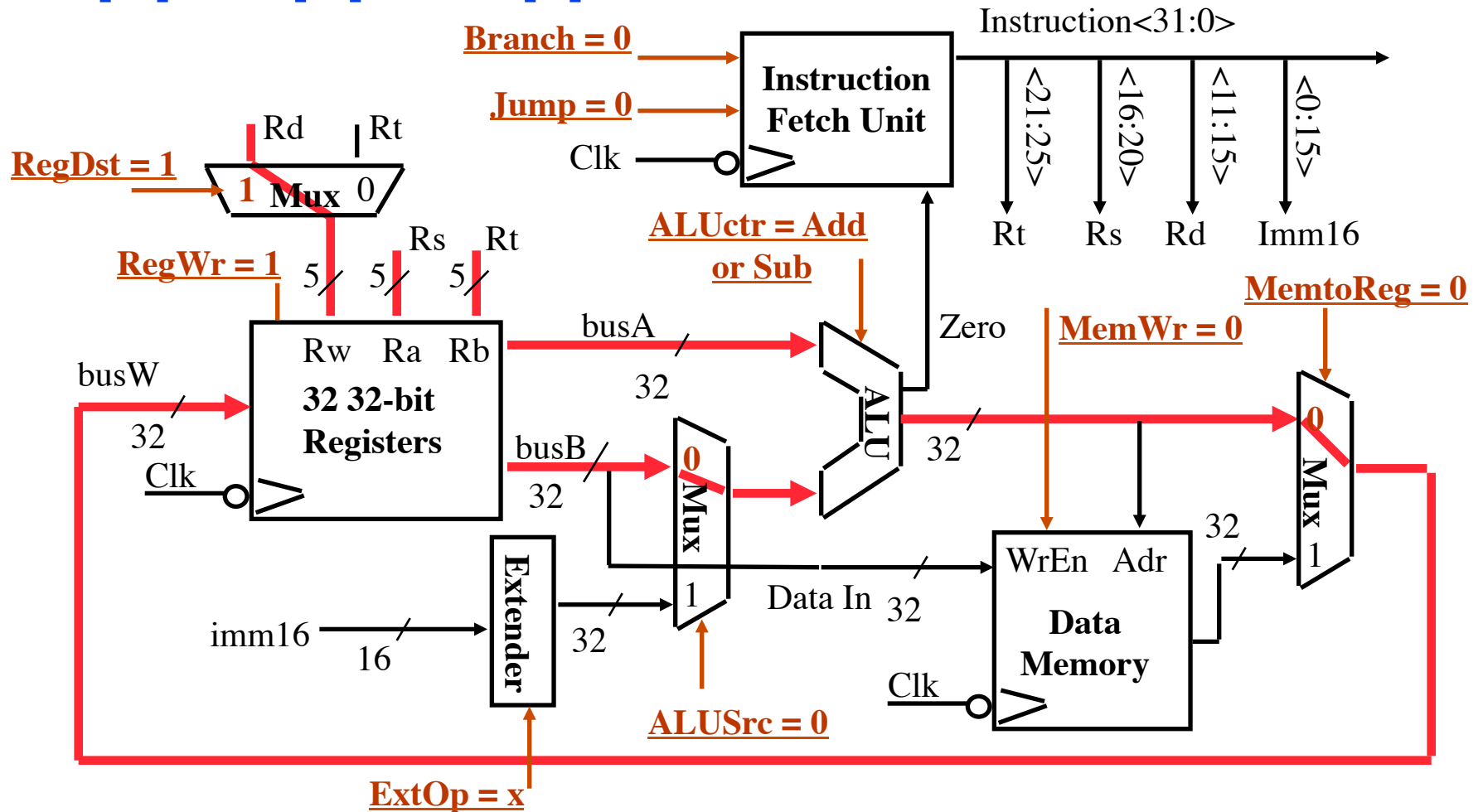
*** $R[rd] \leftarrow R[rs] + / - R[rt]$**



The Single Cycle Datapath during Add and Subtract



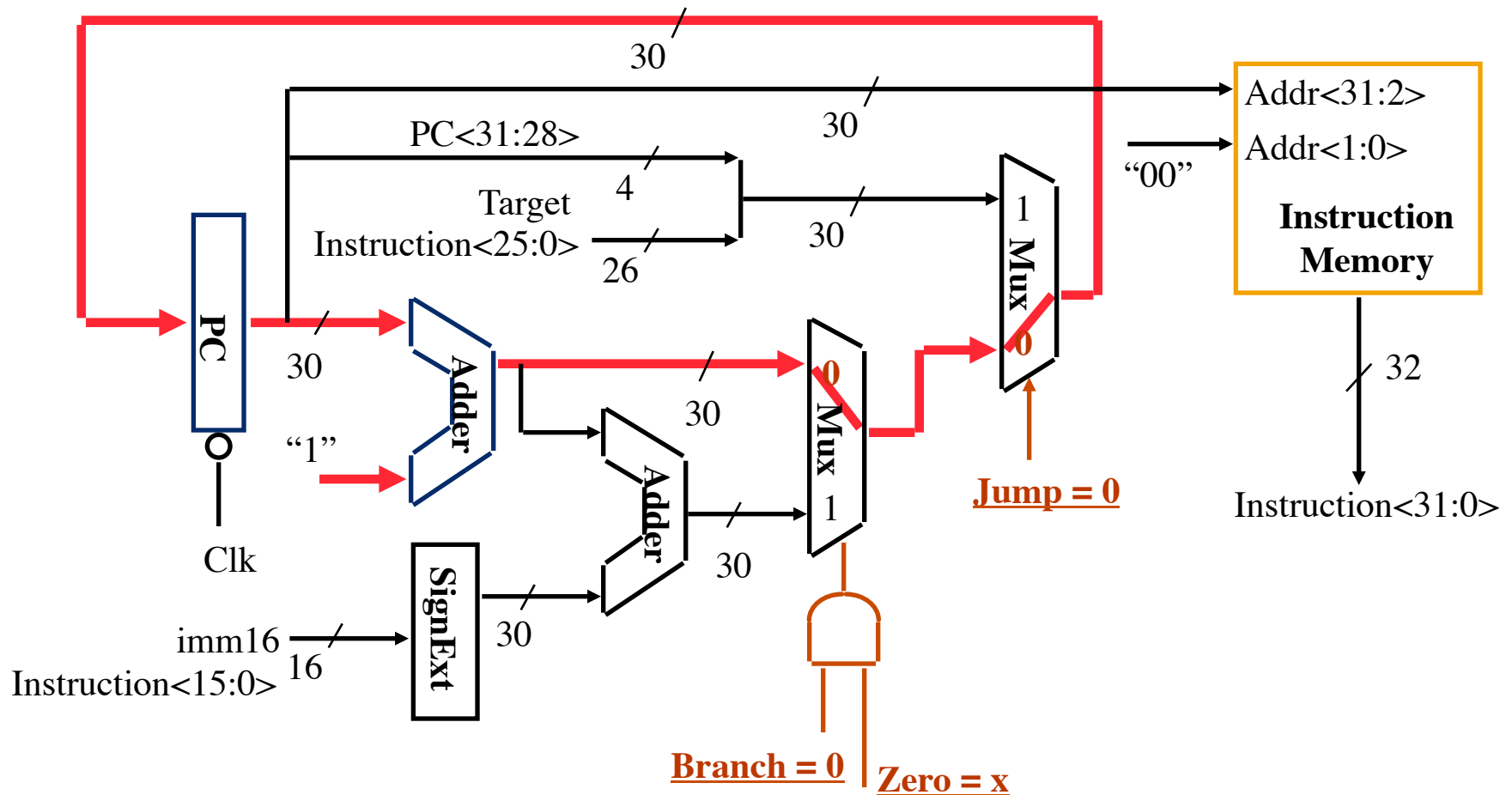
*** $R[rd] \leftarrow R[rs] + / - R[rt]$**



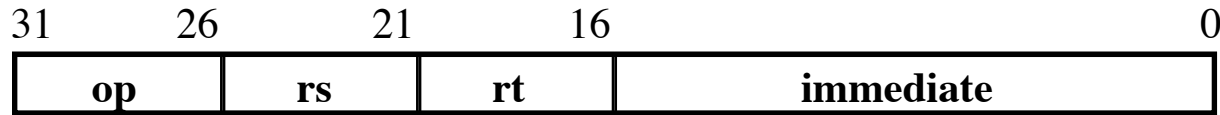
Instruction Fetch Unit at the End of Add and Subtract

• $PC \leftarrow PC + 4$

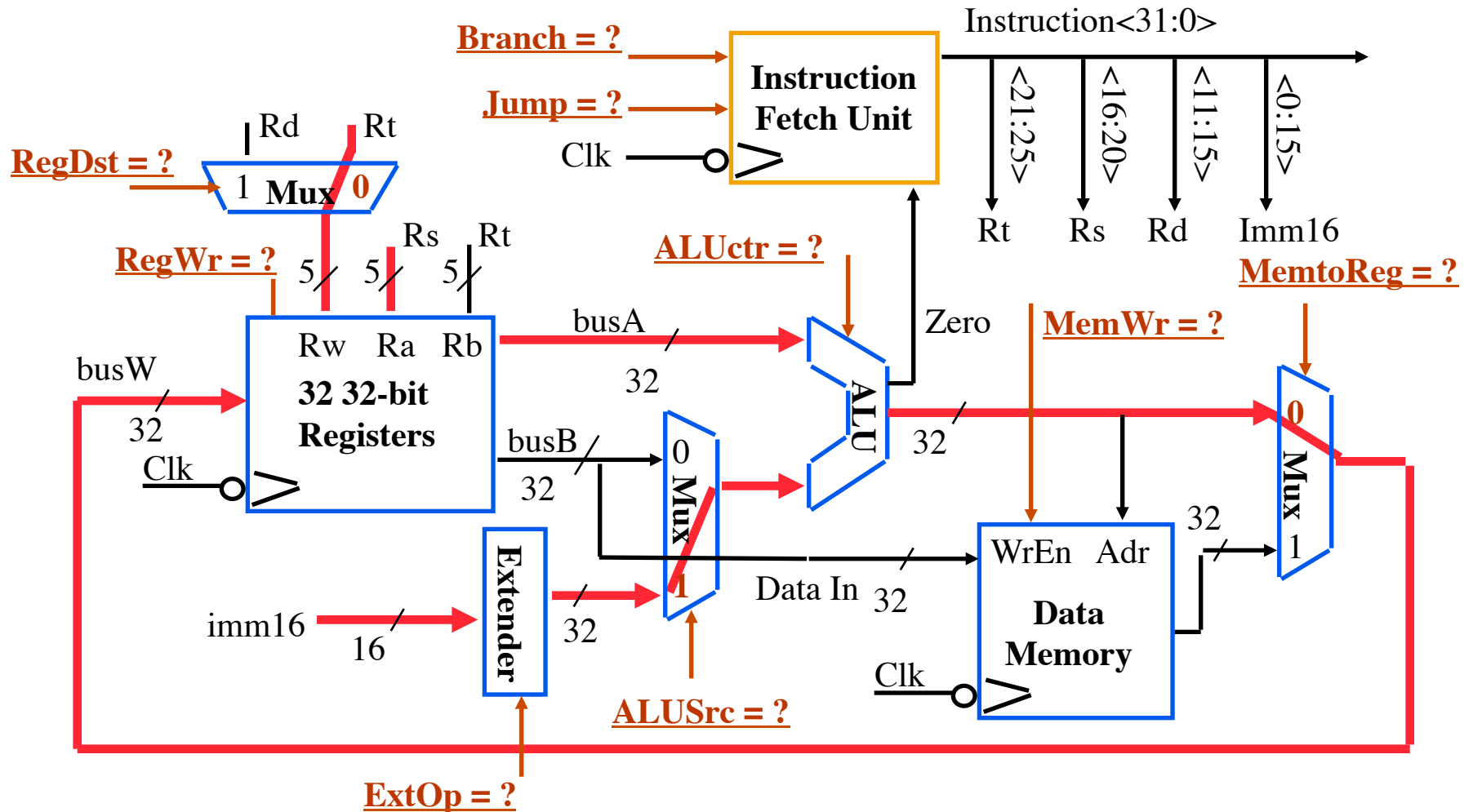
◆ This is the same for all instructions except: Branch and Jump



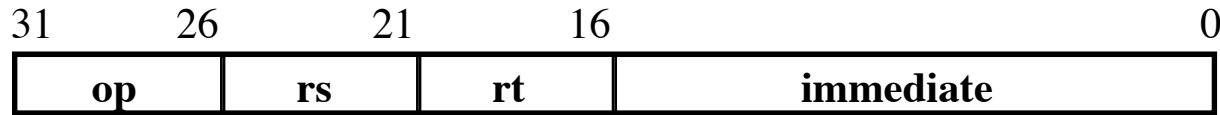
The Single Cycle Datapath during Or Immediate



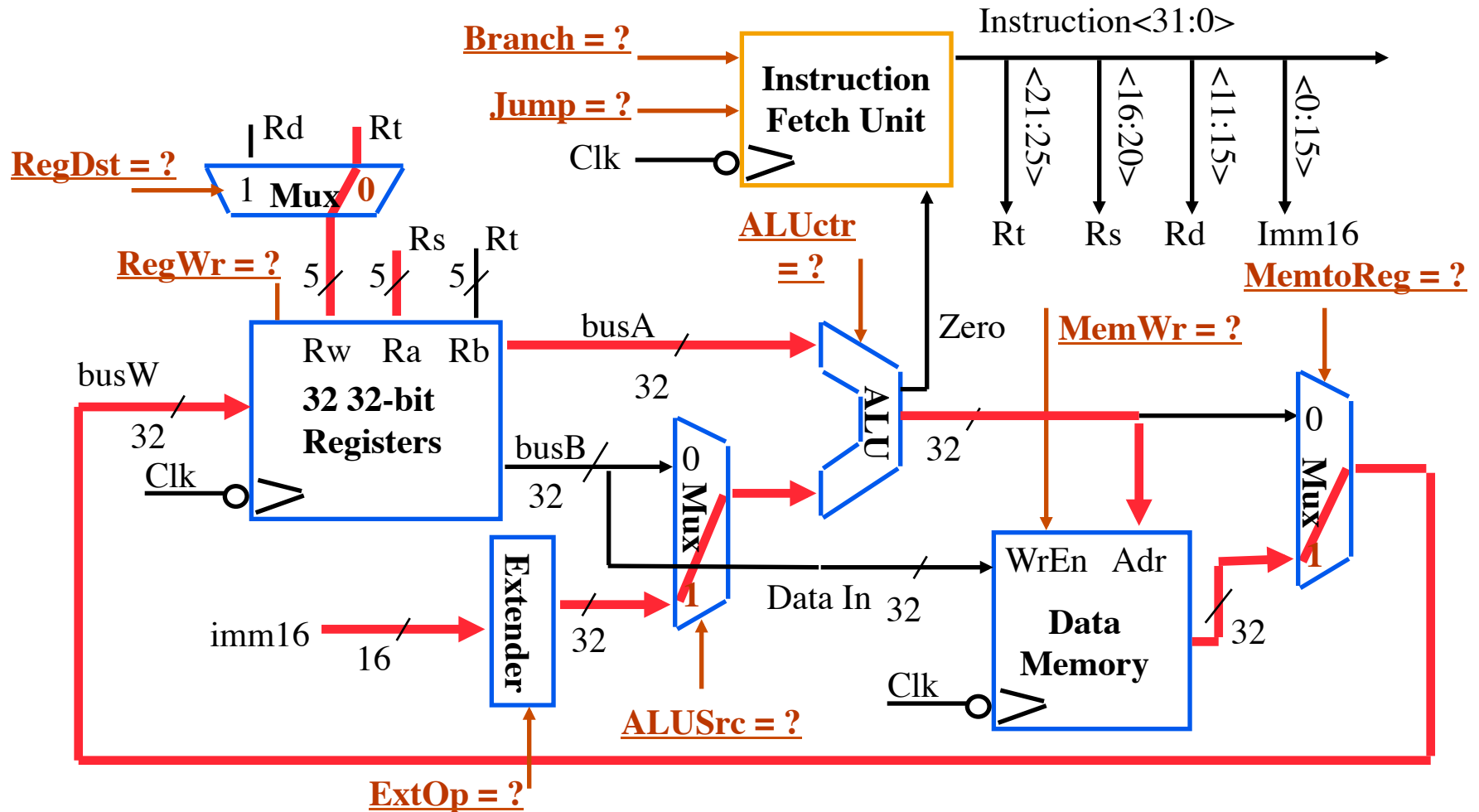
• $R[rt] \leftarrow R[rs] \text{ or } \text{ZeroExt}[\text{Imm16}]$



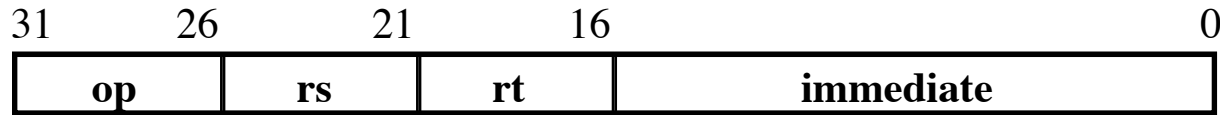
The Single Cycle Datapath during Load



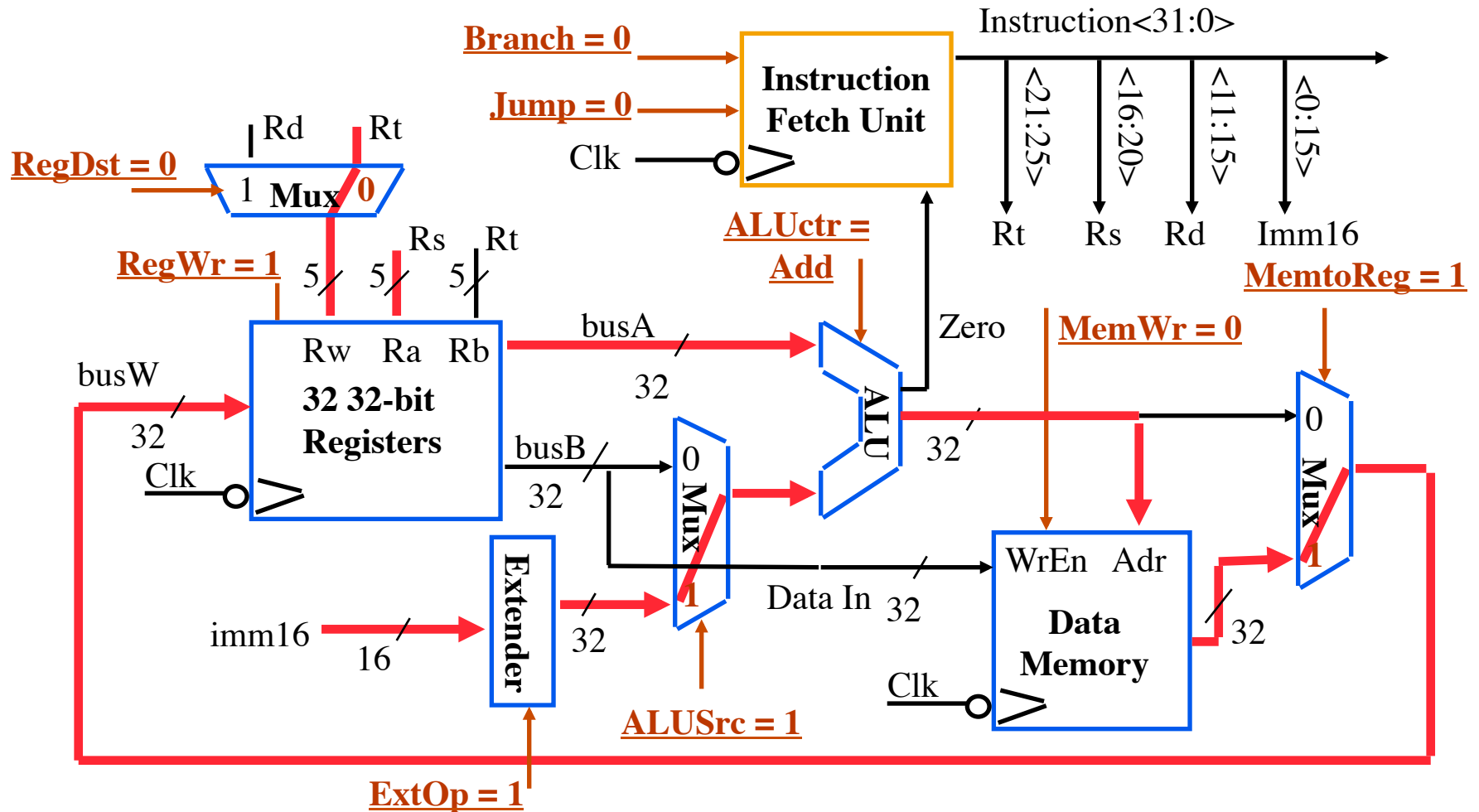
• $R[rt] \leftarrow \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\}$



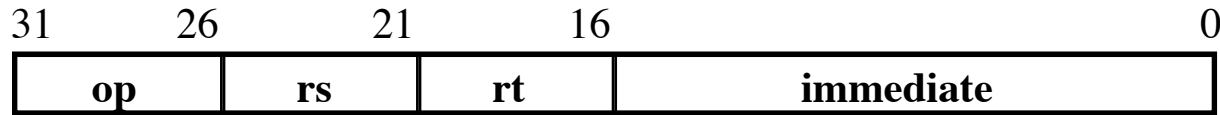
The Single Cycle Datapath during Load



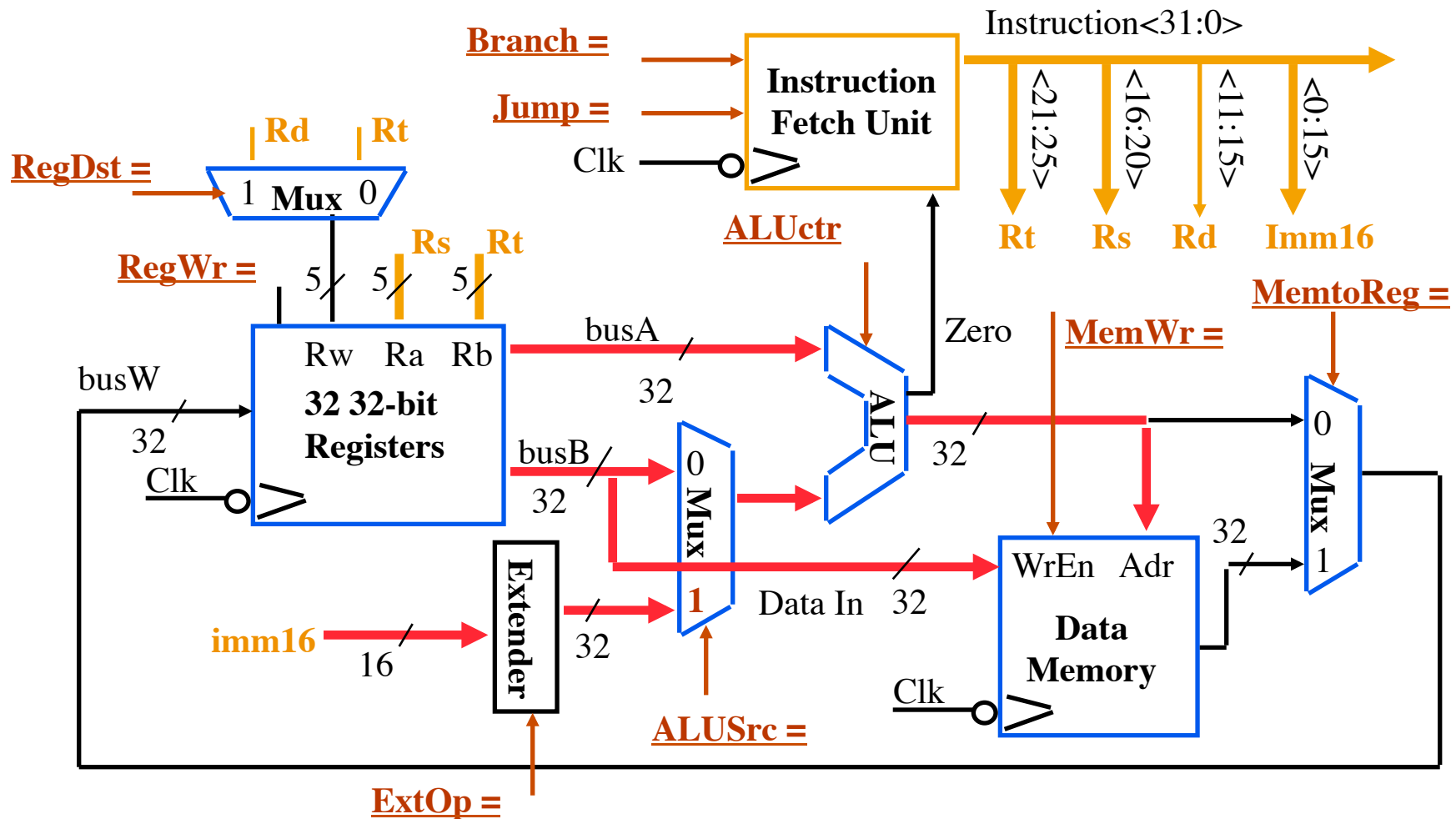
• $R[rt] \leftarrow \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\}$



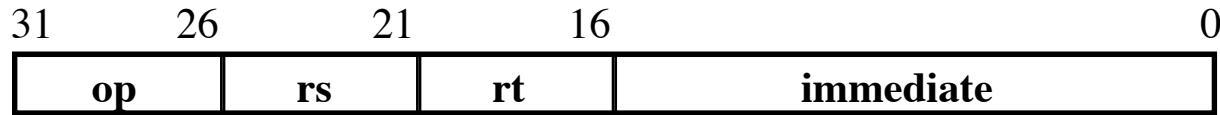
The Single Cycle Datapath during Store



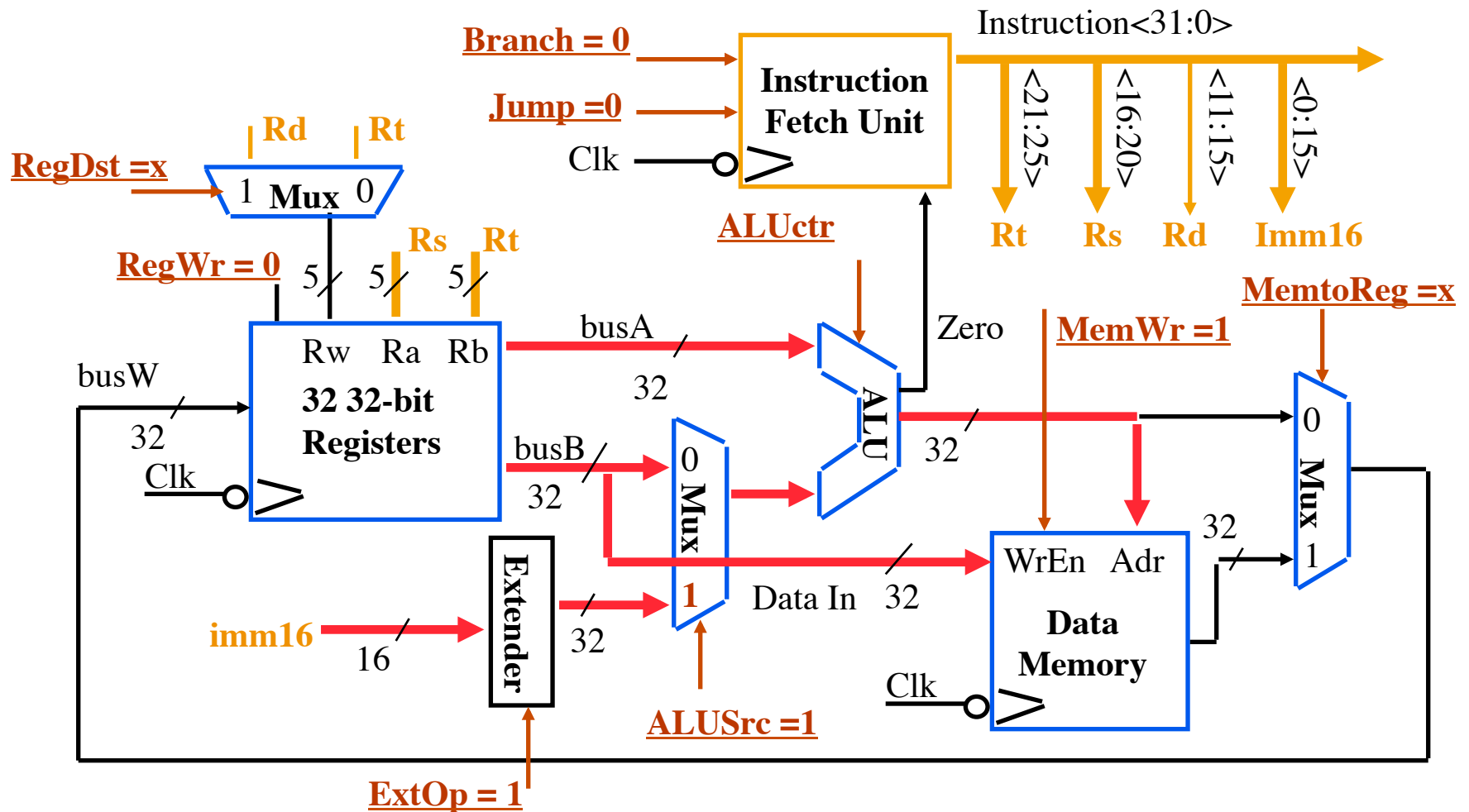
• Data Memory {R[rs] + SignExt[imm16]} ← R[rt]



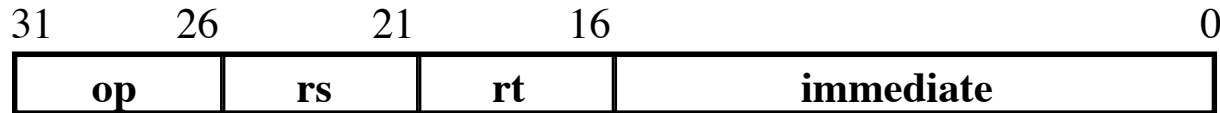
The Single Cycle Datapath during Store



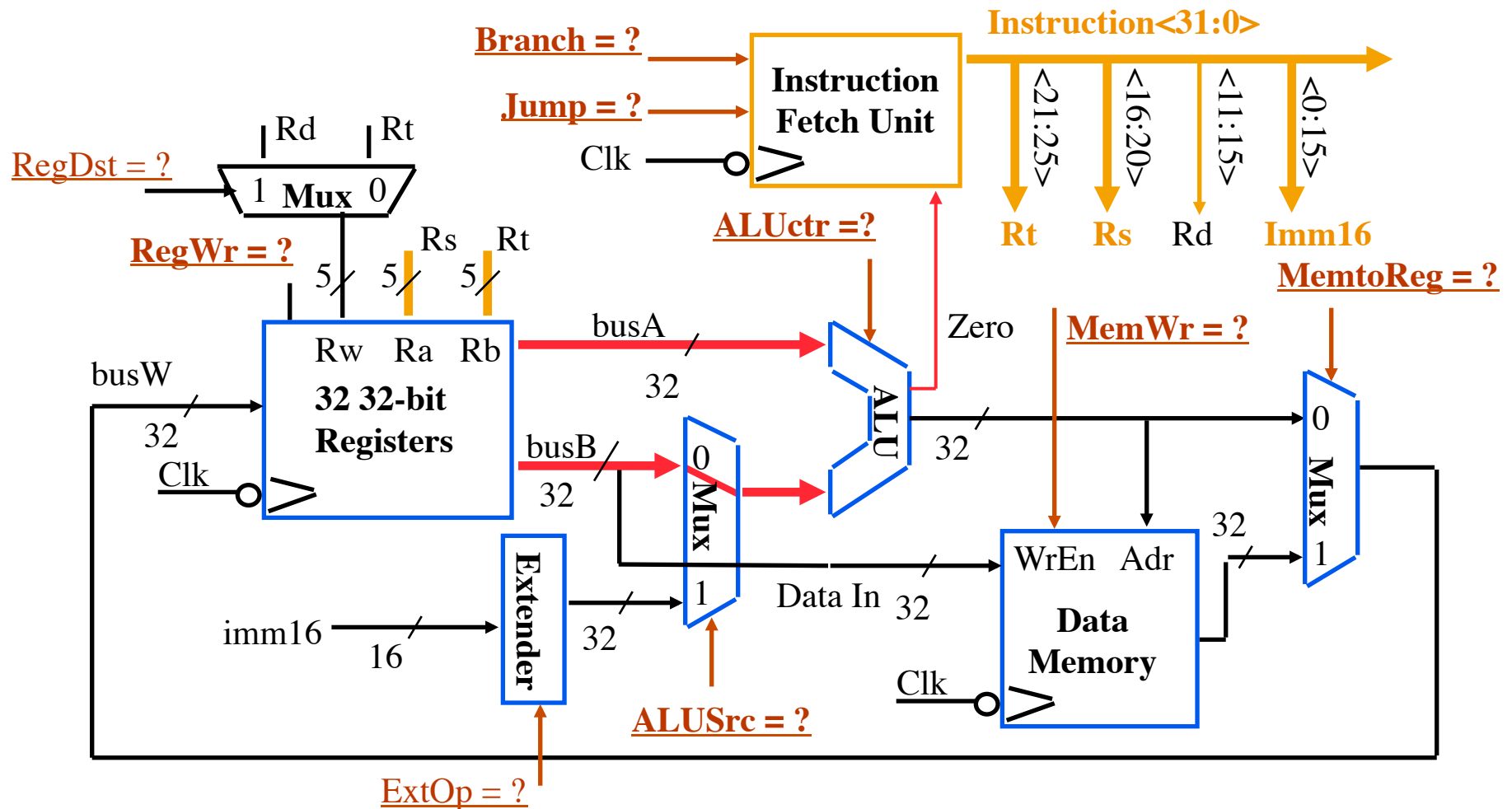
• **Data Memory {R[rs] + SignExt[imm16]} ← R[rt]**



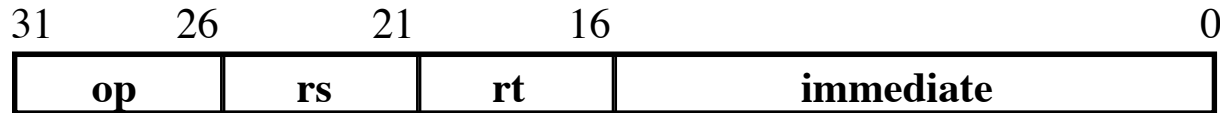
The Single Cycle Datapath during Branch



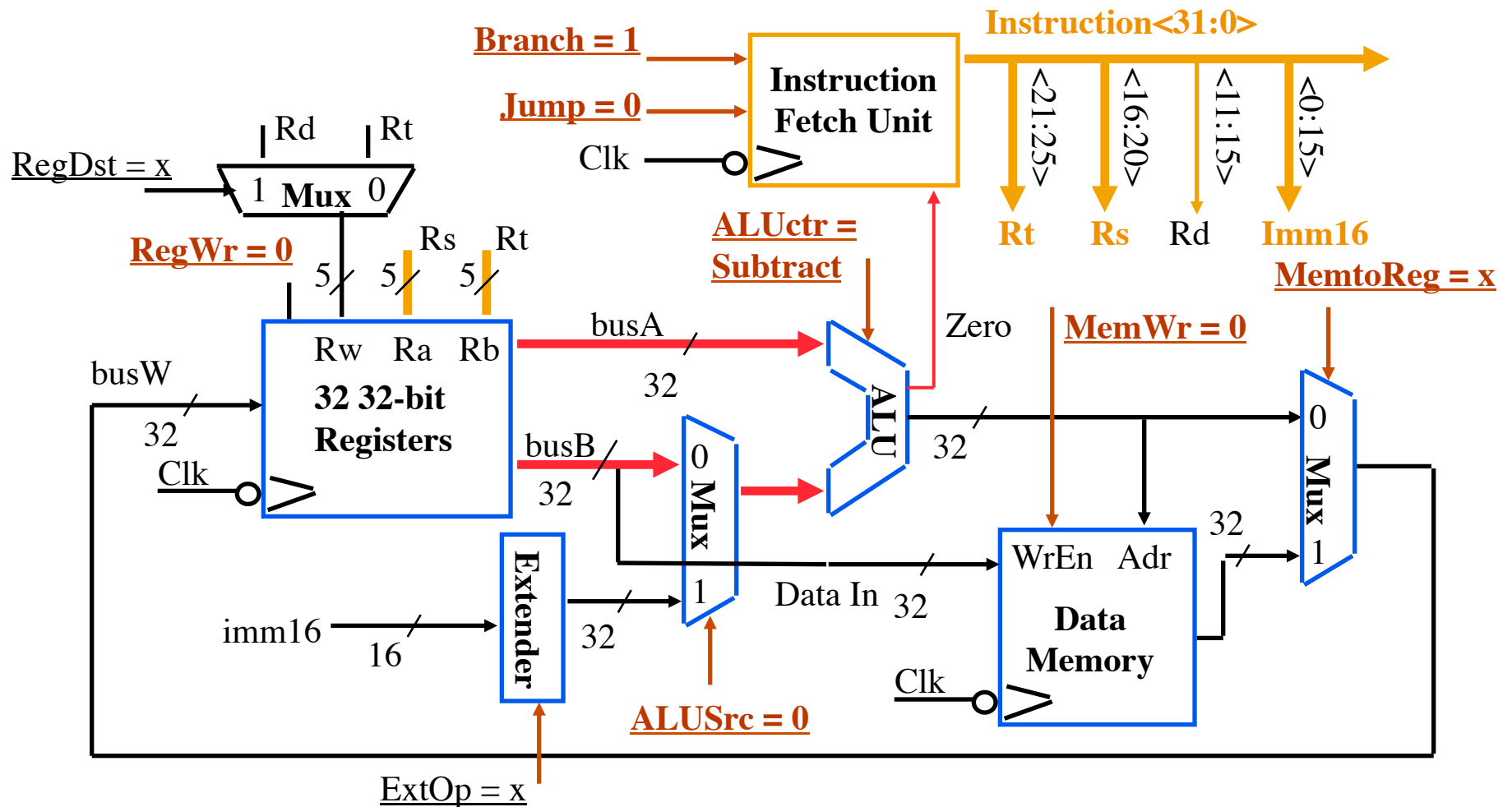
• if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0



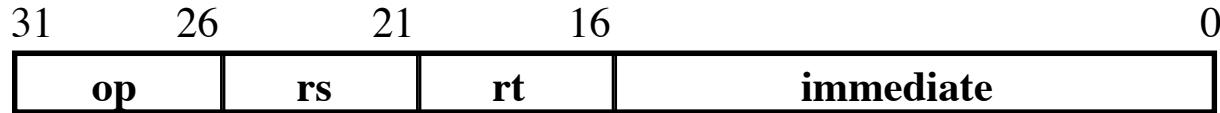
The Single Cycle Datapath during Branch



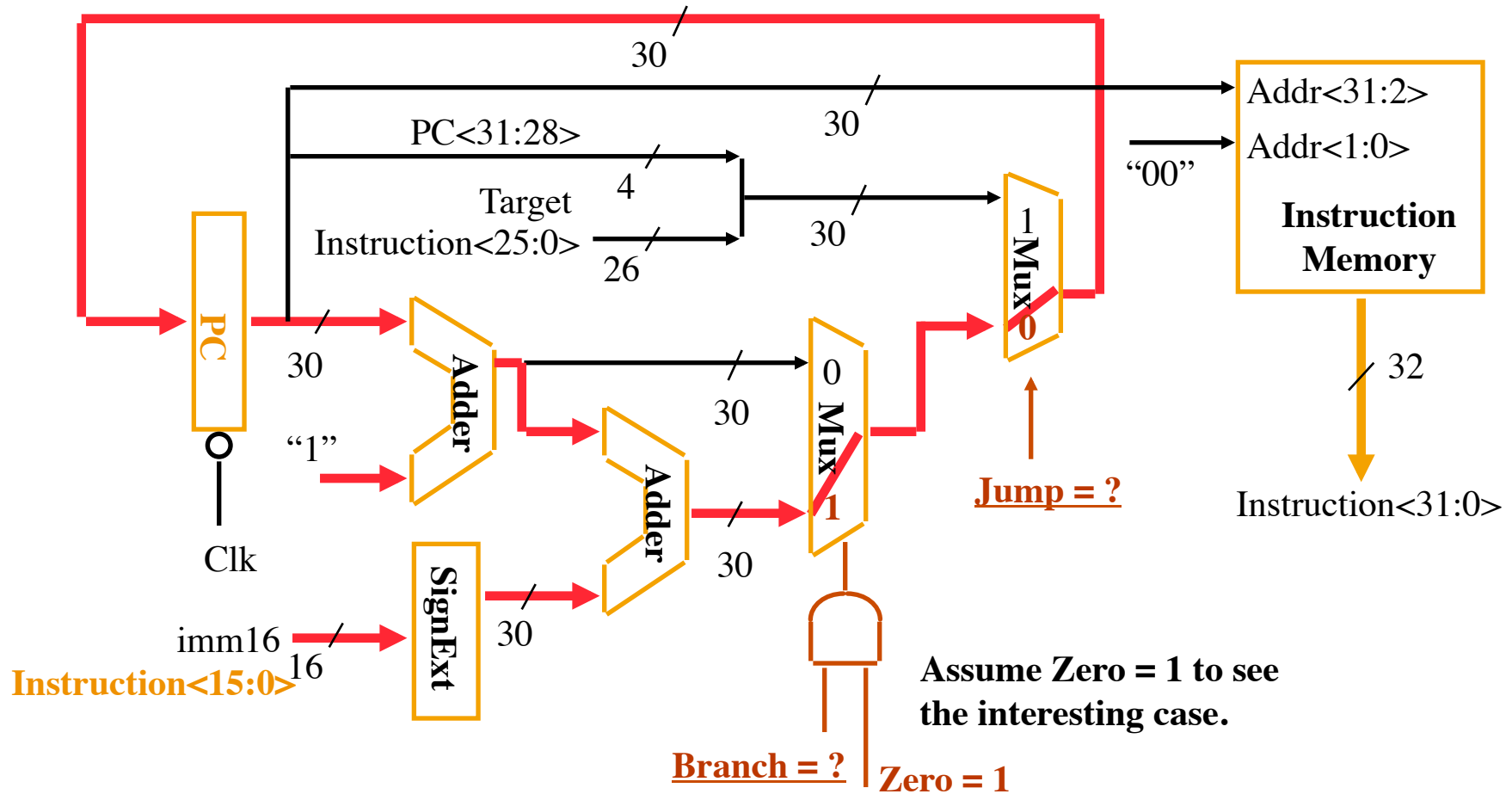
• if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0



Instruction Fetch Unit at the End of Branch



• if (Zero == 1) then $PC = PC + 4 + \text{SignExt}[\text{imm16}] * 4$; else $PC = PC + 4$



The Single Cycle Datapath during Jump



• Nothing to do! Make sure control signals are set correctly!

