

CSL7070: Computer Architecture

Lecture 7, 20th February 2023

Dip Sankar Banerjee



Indian Institute of Technology, Jodhpur
January-April 2023

Building a Datapath

- Datapath
 - Elements that process data and addresses in the CPU
 - Registers, ALUs, MUX's, memories, ...
- We will build a MIPS datapath incrementally
 - Refining the overview design

Instruction Fetch (IF) RTL

- Common RTL operations

- Fetch instruction

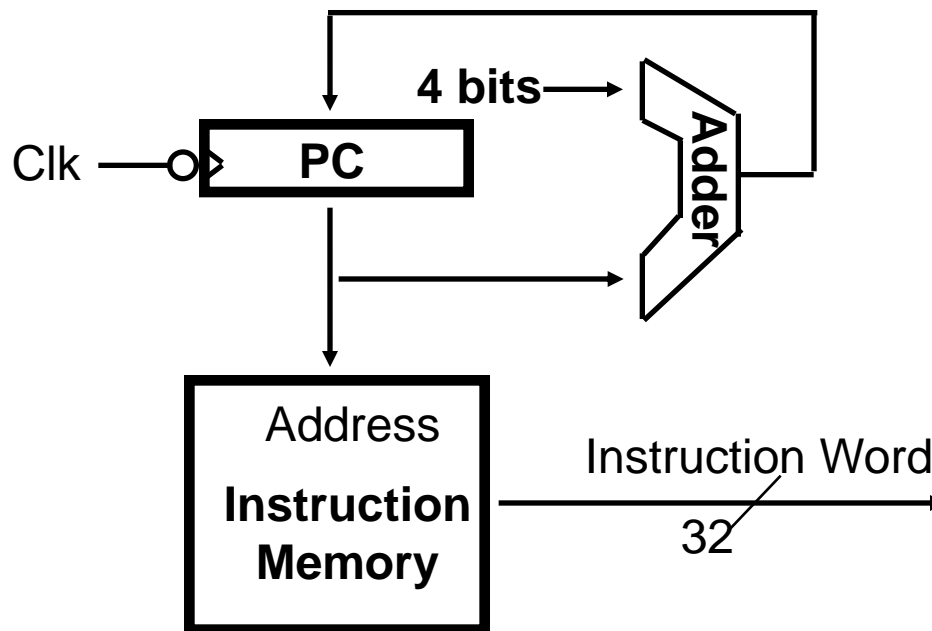
$\text{Mem}[\text{PC}] ;$ Fetch instruction from memory

- Update program counter

- Sequential

$\text{PC} \leftarrow \text{PC} + 4 ;$ Calculate next address

Datapath: Instruction Fetch Unit



Add RTL

- Add instruction

`add rd, rs, rt`

`Mem[PC];` Fetch instruction from memory
`R[rd] <- R[rs] + R[rt];` Add operation
`PC <- PC + 4;` Calculate next address

Bits

6	5	5	5	5	6
OP=0	rs	rt	rd	sa	funct

first second result shift function
 source source register amount code
 register register (=32)

Sub RTL

- Sub instruction

sub rd, rs, rt

Mem[PC]; Fetch instruction from memory
R[rd] <- R[rs] - R[rt]; Sub operation
PC <- PC + 4; Calculate next address

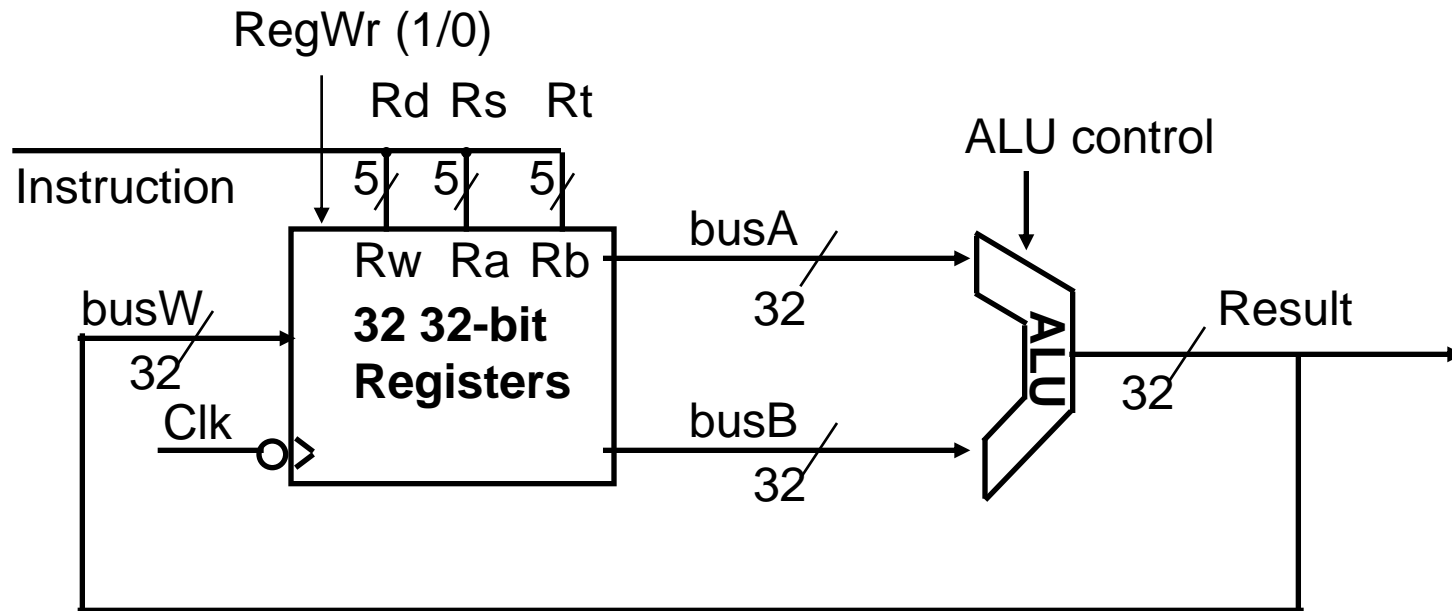
Bits

6	5	5	5	5	6
OP=0	rs	rt	rd	sa	funct

first second result shift function
source source register amount code
register register (=34)

Datapath: Reg-Reg Operations

- $R[rd] \leftarrow R[rs] \text{ op } R[rt];$
 - ALU control and RegWr coming from Control Unit, based on decoded instruction
 - Ra, Rb, and Rw from rs, rt, rd fields



OR Immediate RTL

- OR Immediate instruction

```
ori rt, rs, imm
```

```
Mem[PC];           Fetch instruction from memory
```

```
R[rt] <- R[rs] OR ZeroExt(imm);
```

OR operation with Zero-Extend

```
PC <- PC + 4;       Calculate next address
```

Bits

6

5

5

16

OP	rs	rt	imm
----	----	----	-----

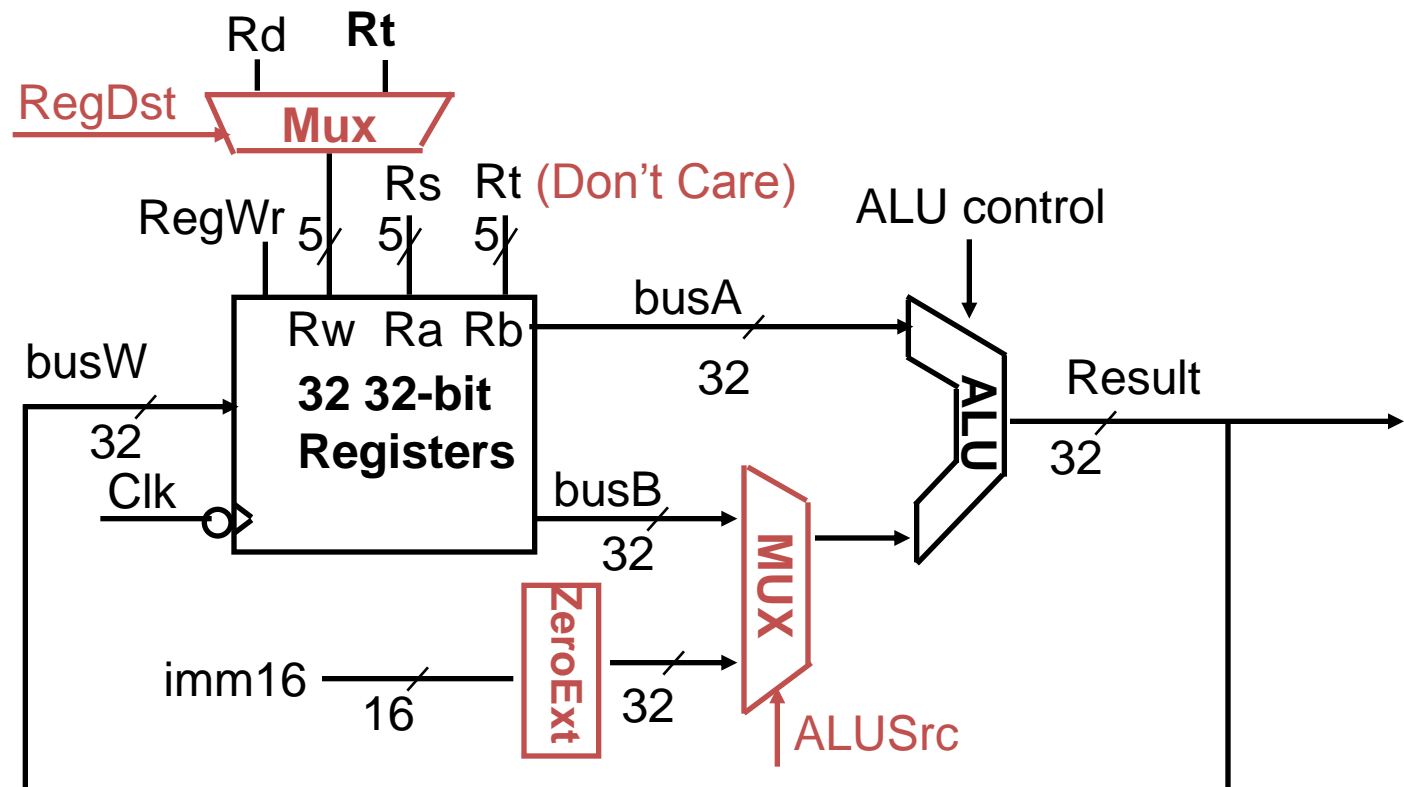
first
source
register

second
register
(dest)

immediate

Datapath: Immediate Ops

- Rw set by MUX and ALU B set as busB or ZeroExt(imm)
- ALUSrc and RegDst set based on instruction

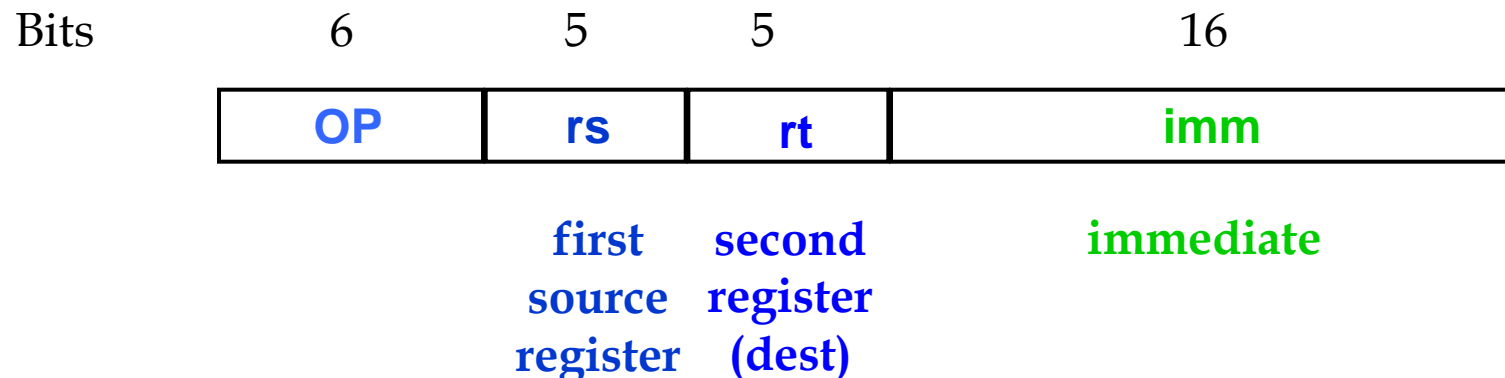


Load RTL

- Load instruction

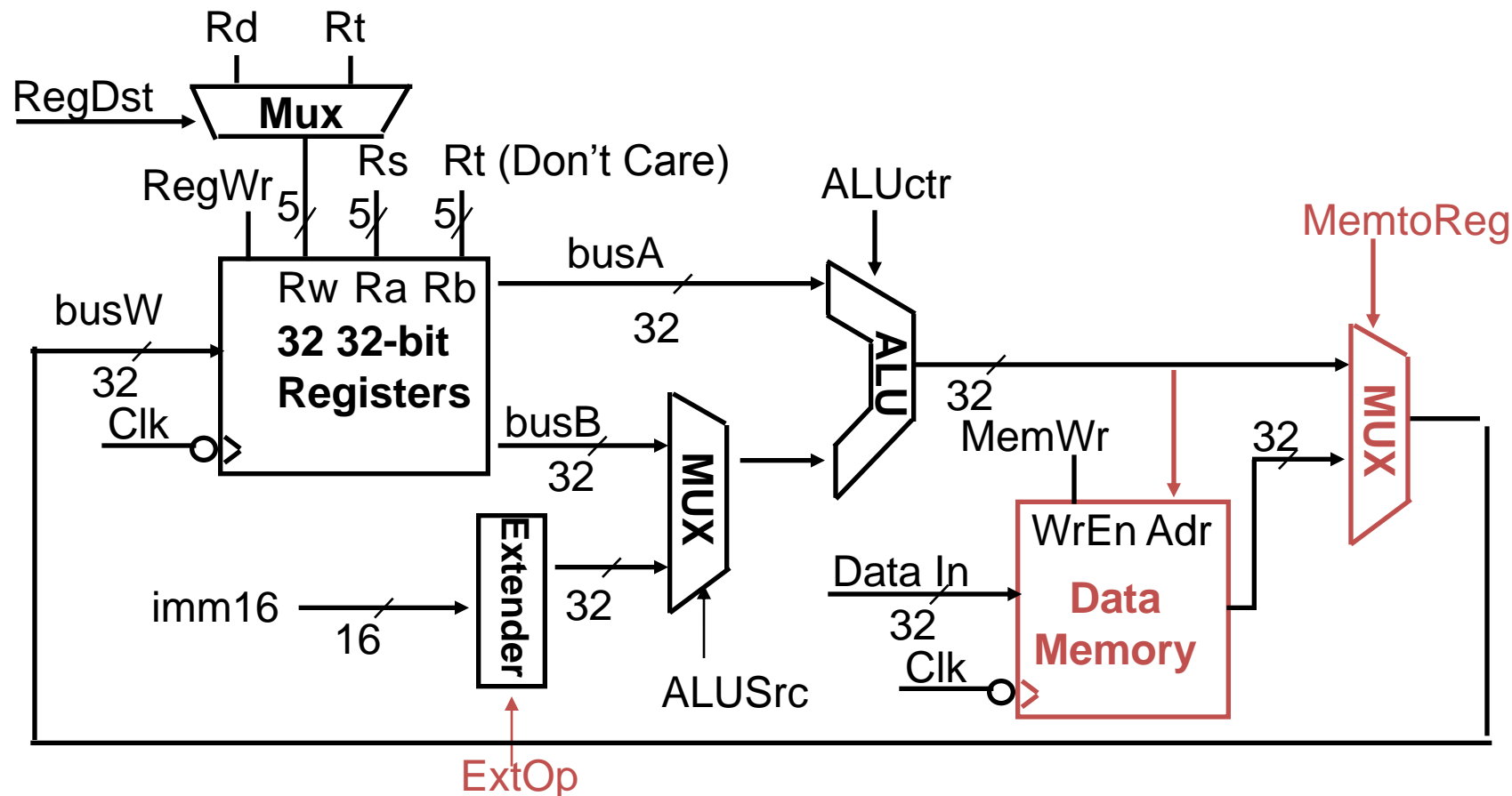
`lw rt, rs, imm`

```
Mem[PC];           Fetch instruction from memory
Addr <- R[rs] + SignExt(imm); Compute memory address
R[rt] <- Mem[Addr]; Load data into register
PC <- PC + 4;       Calculate next address
```



Datapath: Load

- Extender handles sign vs. zero extension of immediate
- MUX selects between ALU result and Memory output



Store RTL

- Store instruction

`sw rt, rs, imm`

```
Mem[PC];           Fetch instruction from memory
Addr <- R[rs] + SignExt(imm); Compute memory addr
Mem[Addr] <- R[rt]; Load data into register
PC <- PC + 4;       Calculate next address
```

Bits

6

5

5

16

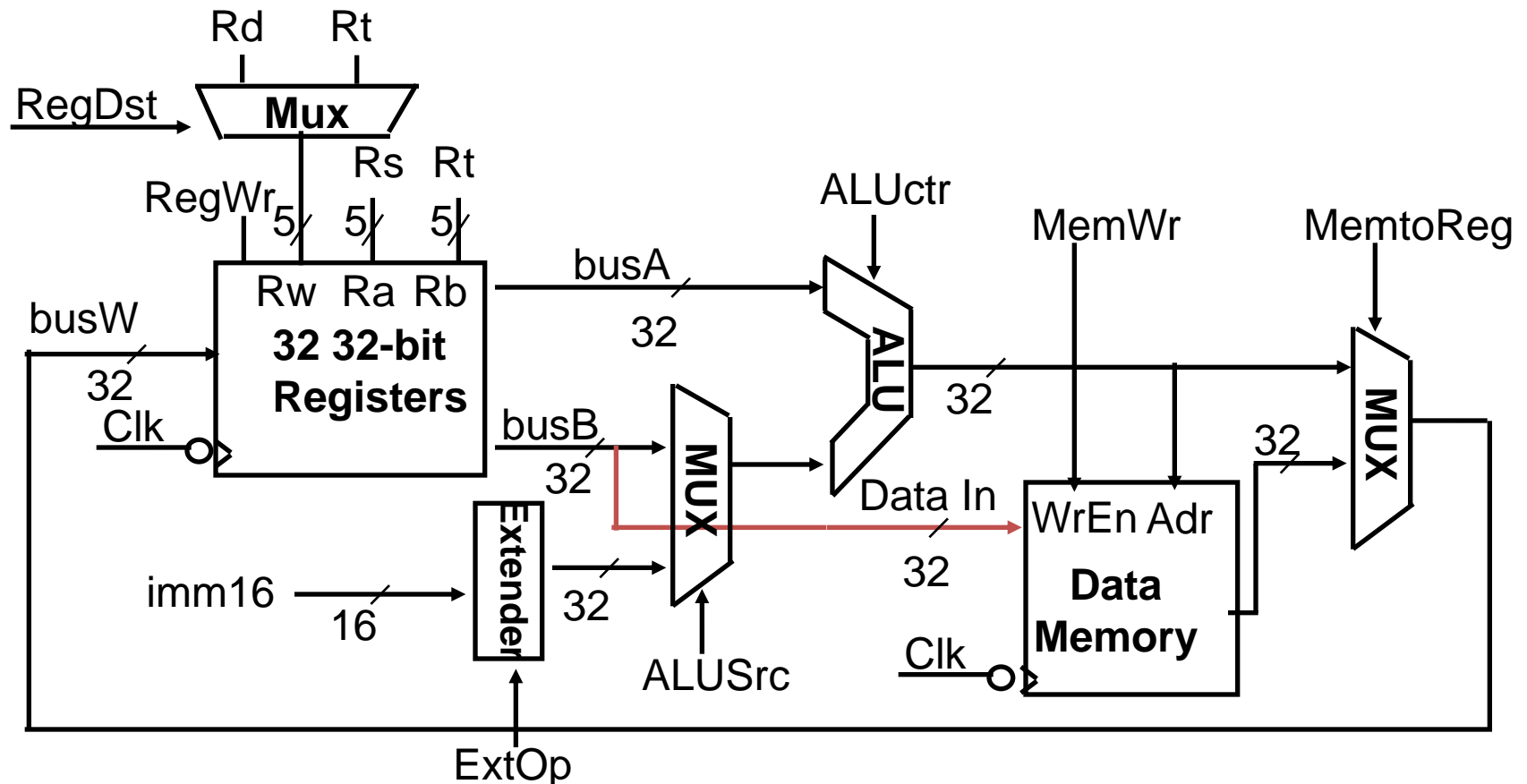
OP	rs	rt	imm
----	----	----	-----

first source register
second source register

immediate

Datapath: Store

- Register rt is passed on busB into memory
- Memory address calculated just as in lw case



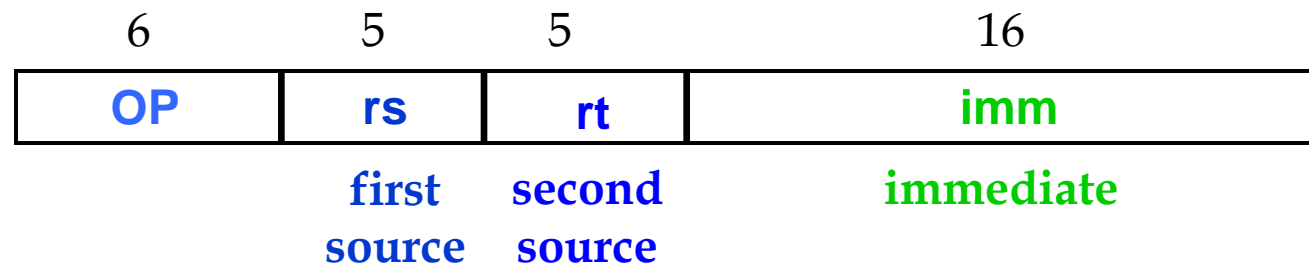
Branch RTL

- Branch instruction

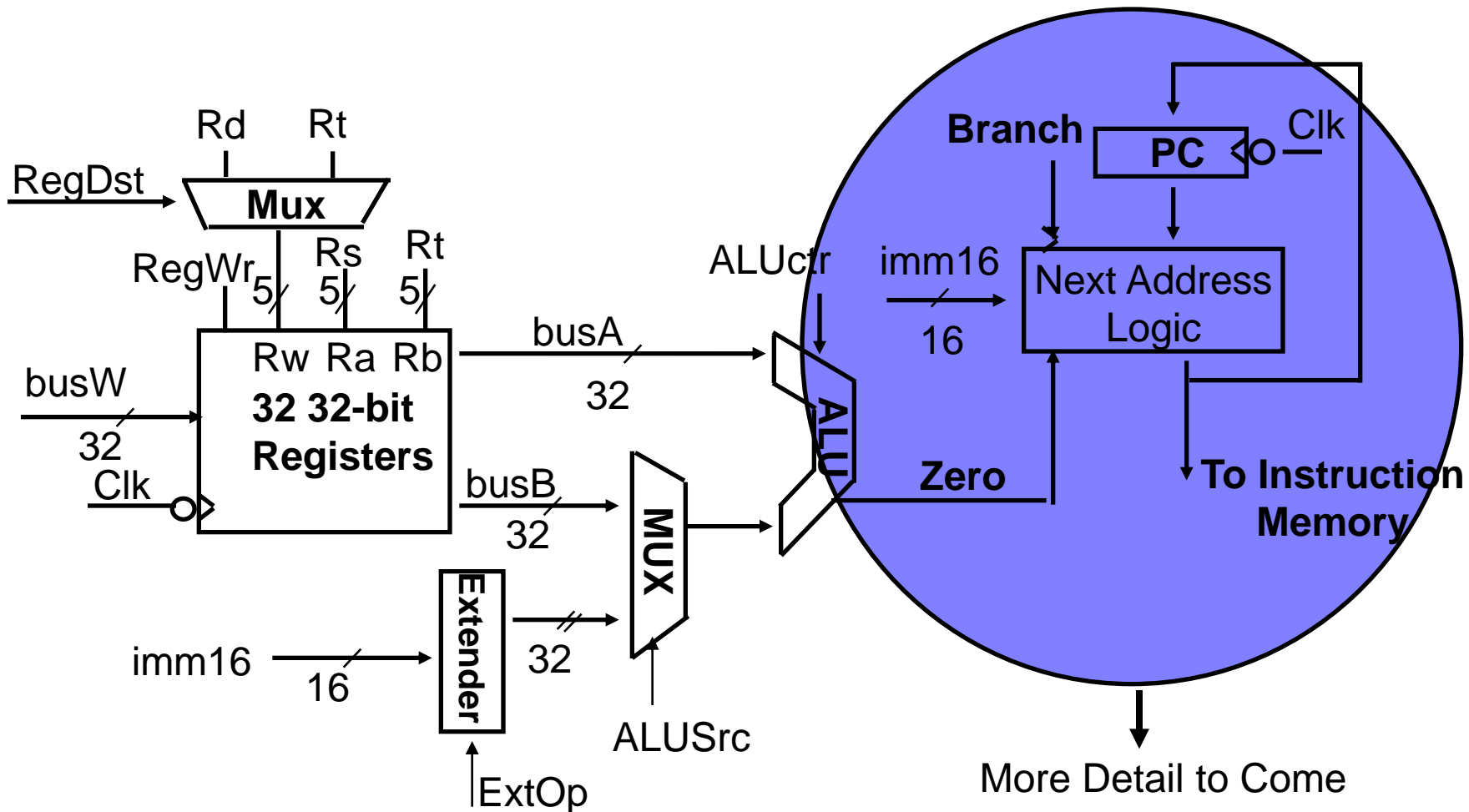
`beq rs, rt, imm`

```
Mem[PC];           Fetch instruction from memory
Diff <- R[rs] - R[rt];  Calculate branch condition
if (Diff eq 0)      Test if equal
    PC <- PC + 4 +
        SignExt(imm) * 4; Calculate PC Relative address
else
    PC <- PC + 4;      Calculate next address
```

Bits



Datapath: Branch



The Next Address

- PC is byte-addressed in instruction memory
 - Sequential

$$PC[31:0] = PC[31:0] + 4$$

- Branch operation

$$PC[31:0] = PC[31:0] + 4 + \text{SignExt}(\text{imm}) \times 4$$

- Instruction Addresses

- PC is byte addressed, but instructions are 4 bytes long
- Therefore 2 LSBs of the 32 bit PC are always 0
- No reason to have hardware keep the 2 LSBs
- ⇒ Simplify hardware by using 30 bit PC

- Sequential

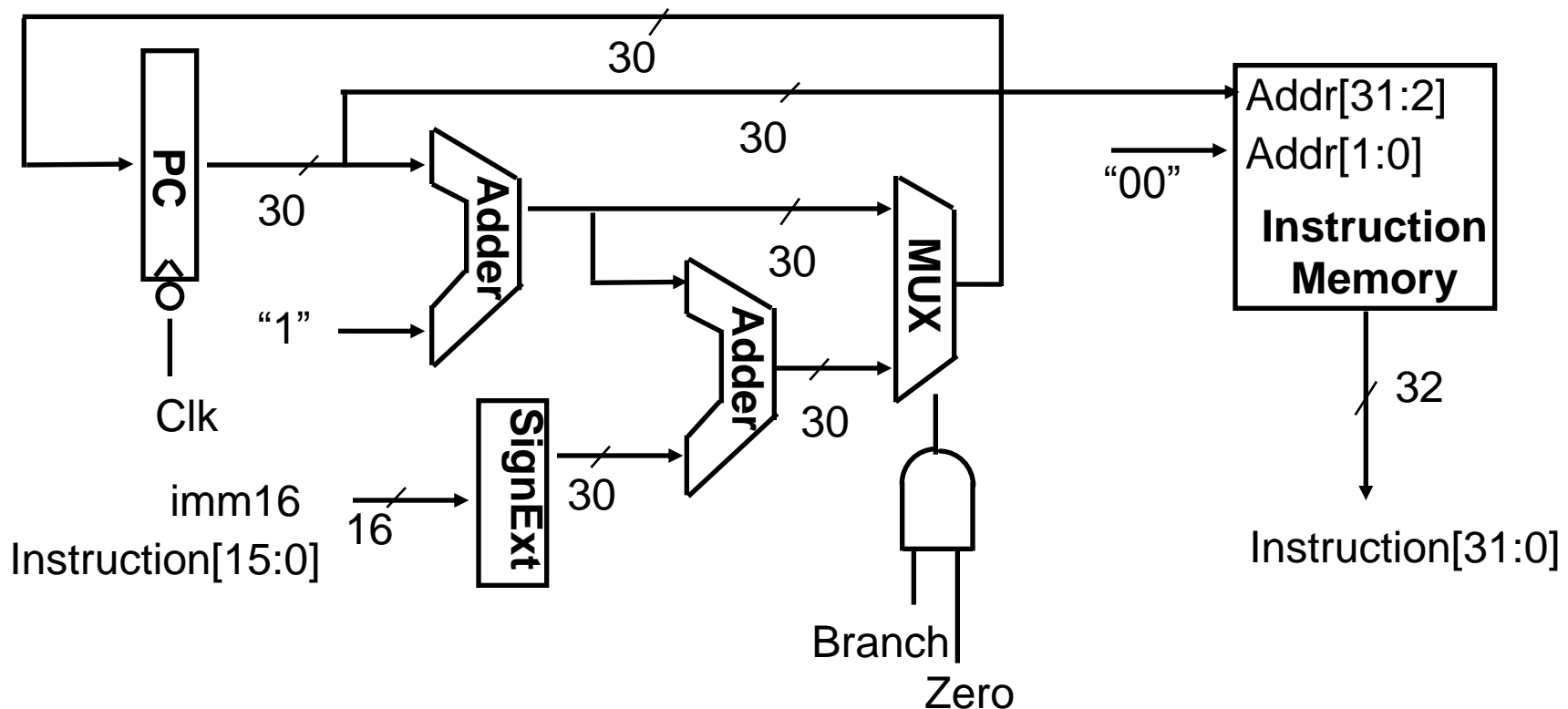
$$PC[31:2] = PC[31:2] + 1$$

- Branch operation

$$PC[31:2] = PC[31:2] + 1 + \text{SignExt}(\text{imm})$$

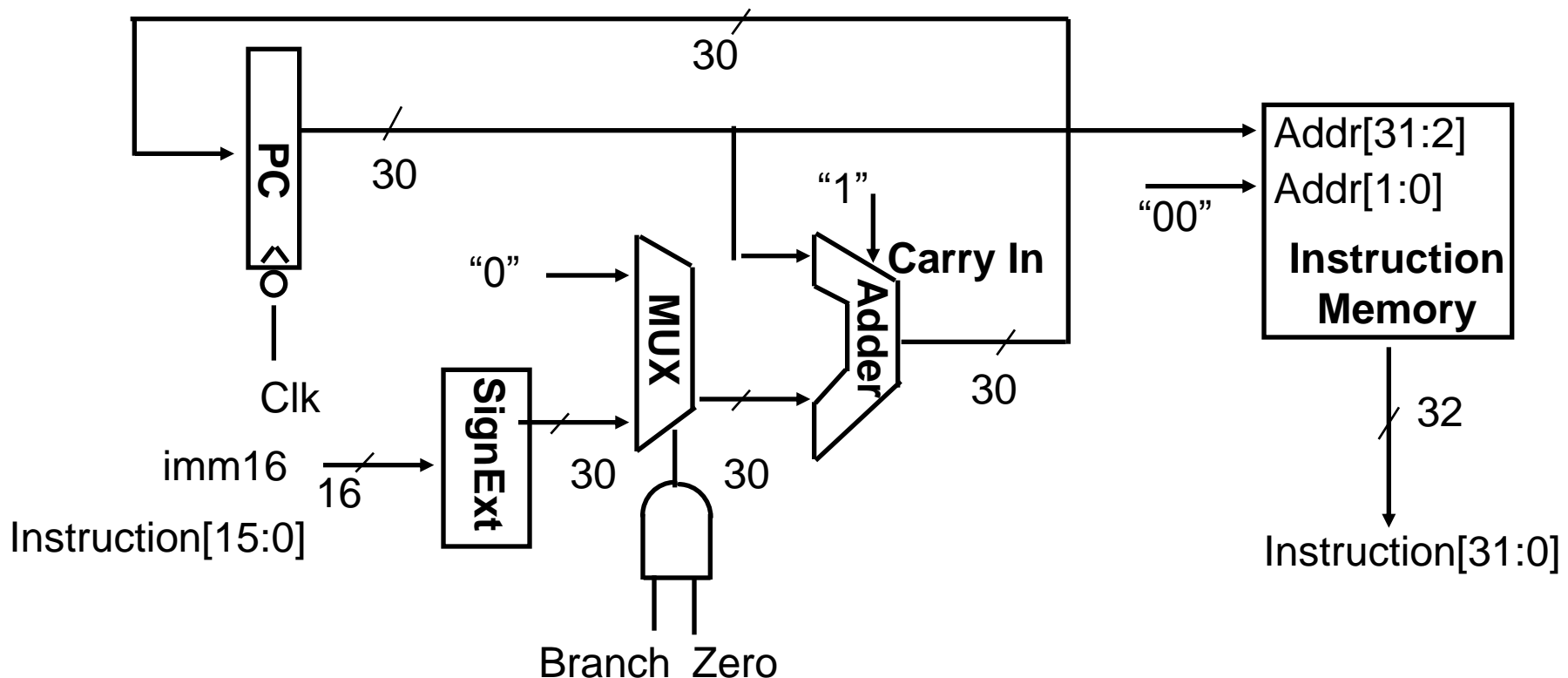
Datapath: Fast, Expensive Next-IF Logic

- PC incremented to next instruction normally
- On beq instruction then can add immediate $\times 4$ to $PC + 4$



Datapath: Slow, Smaller Next-IF Logic

- Slow because cannot start address add until ALU zero
- But probably not the critical path (LOAD is usually)



Jump RTL

- Jump instruction

j target

```
Mem[PC];           Fetch instruction from memory
PC[31:2] <- PC[31:28] ||
    target[25:0];    Calculate next address
```

Bits

6

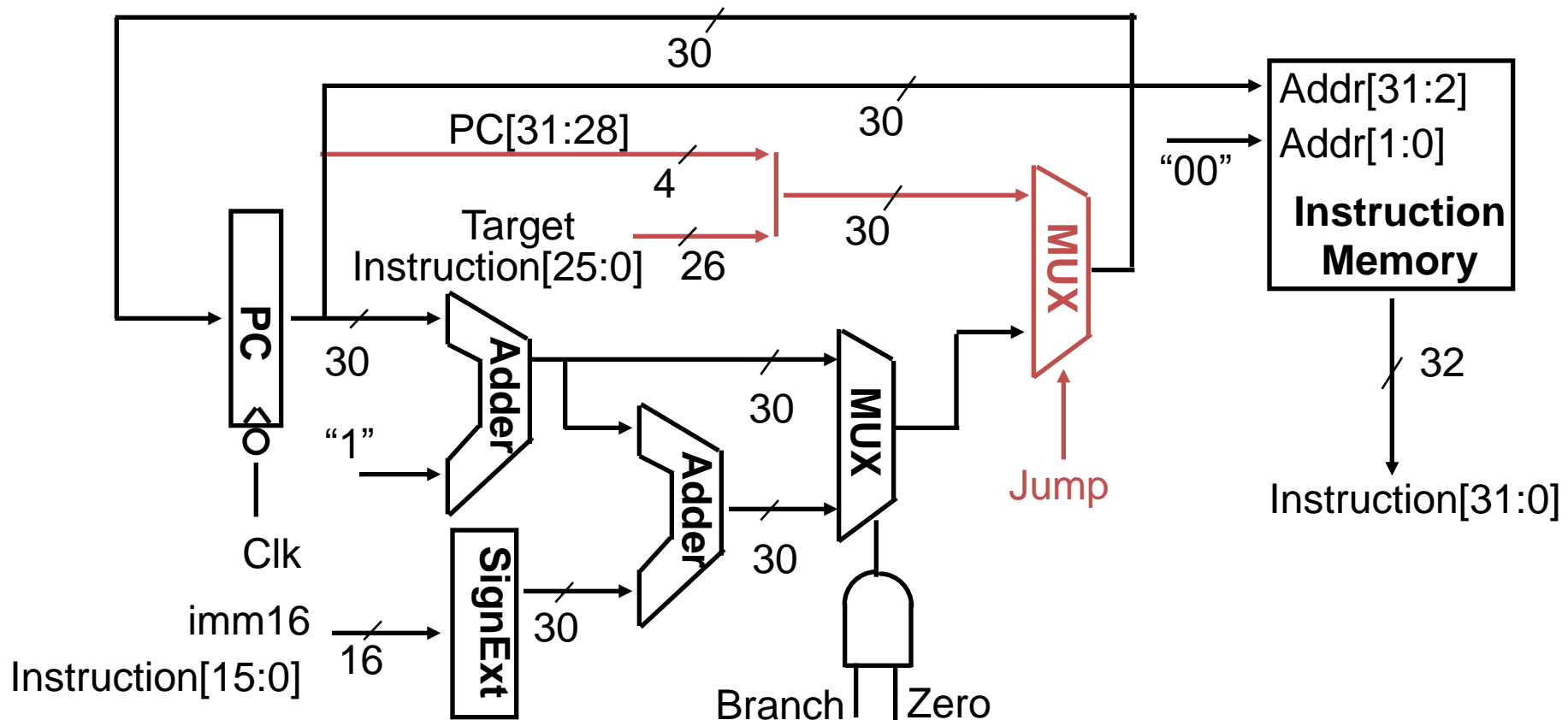
26



jump target address

Datapath: IFU with Jump

- MUX controls if PC is pseudodirect jump



Putting it All Together

