CSL7070: Computer Architecture
Lecture 7, 9th February 2022

Dip Sankar Banerjee

Indian Institute of Technology, Jodhpur
January-April 2022

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

A language to define hardware

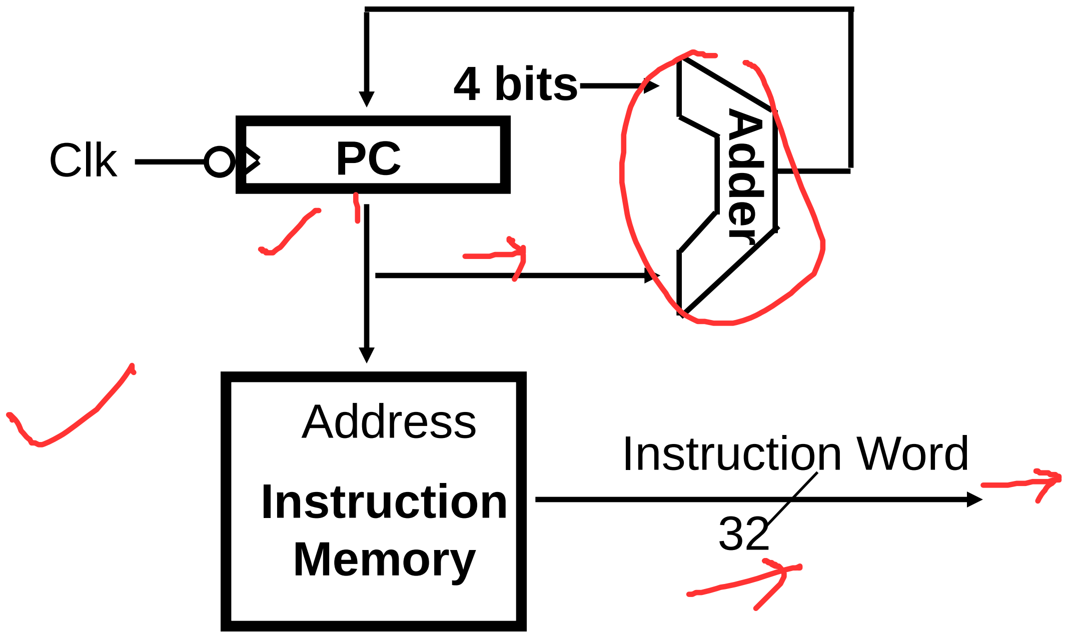
- Common RTL operations
  - Fetch instruction

    `Mem[PC];` Fetch instruction from memory

  - Update program counter
    - Sequential

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

Datapath: Instruction Fetch Unit



Add RTL

R type

- Add instruction

```
add rd, rs, rt
```

add t3, t1, t2
d s t

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

~~ISA~~ Assemb.
ISA

sll slli srl srli

Bits	6	5	5	5	5	6
	OP=0	rs	rt	rd	sa	funct
		first source register	second source register	result register	shift amount	function code (=32)

Sub RTL

- Sub instruction

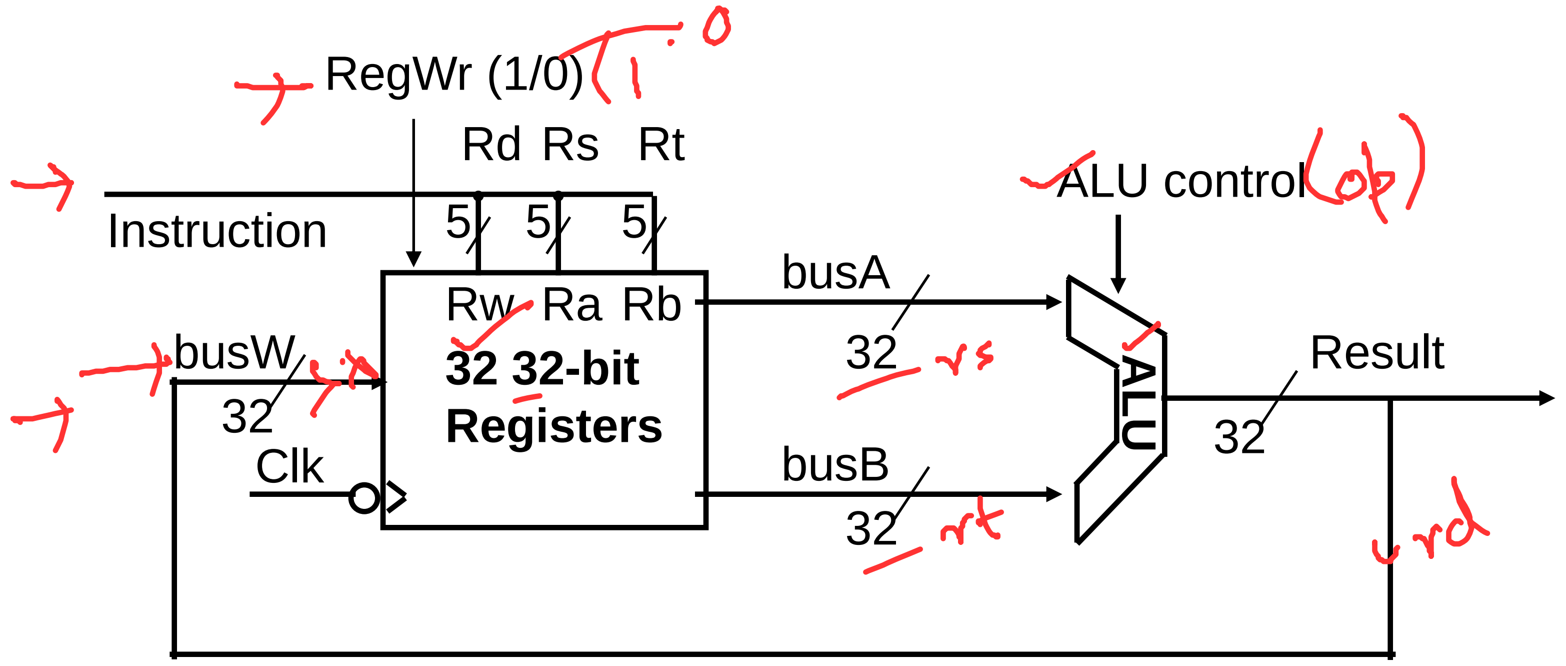
```
sub rd, rs, rt
```

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

Datapath: Reg-Reg Operations

- `R[rd] <- R[rs] 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
```

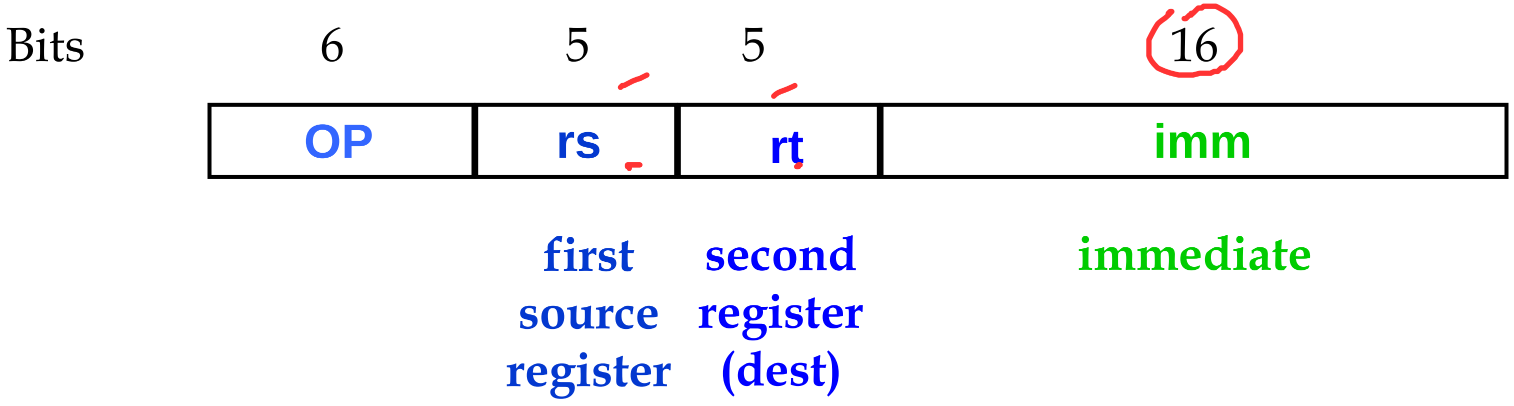
~~ad~~ addi;

```
      Mem[PC];                    Fetch instruction from memory
      R[rt] <- R[rs] OR ZeroExt(imm);
                          OR operation with Zero-Extend
      PC <- PC + 4;               Calculate next address
```

+ +

$-2^8 \rightarrow (2^8-1)$

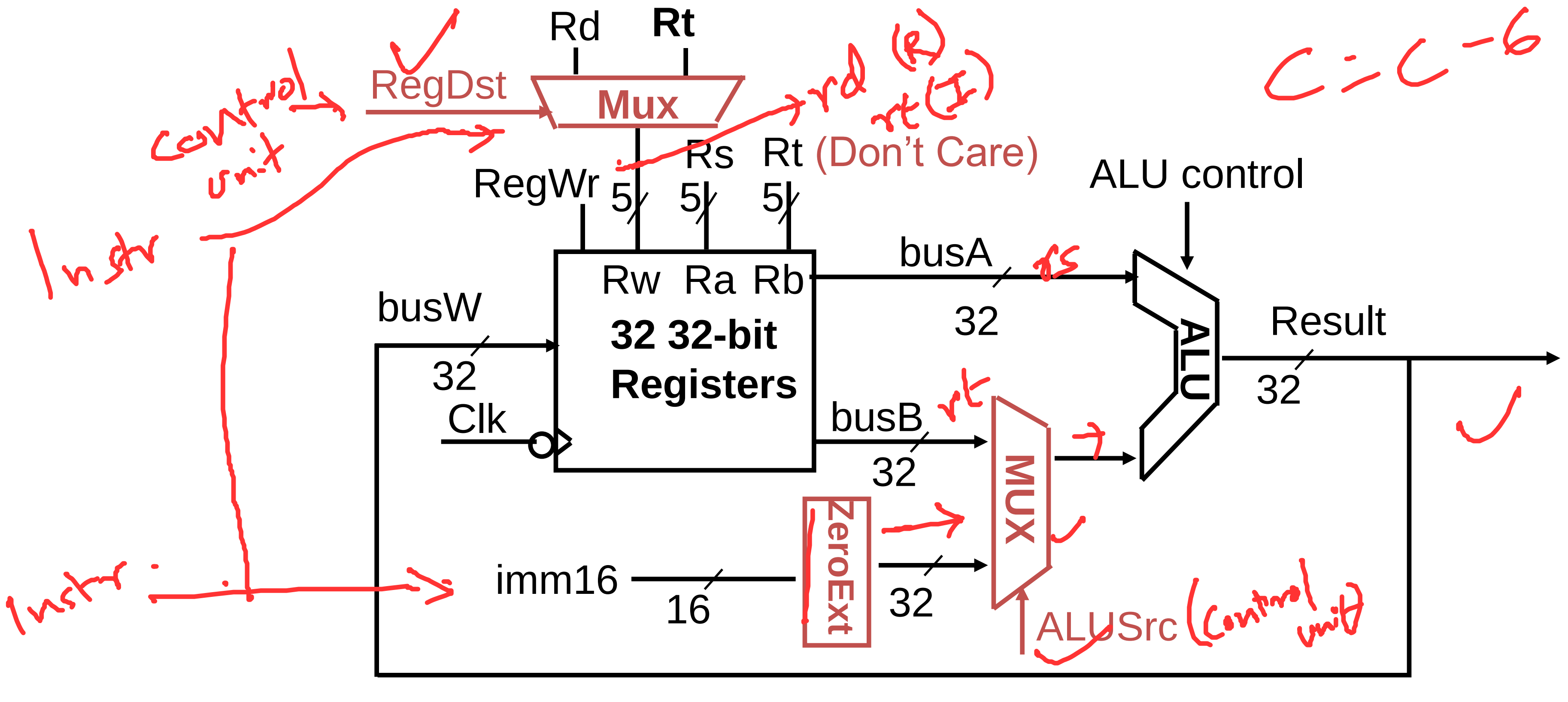
2^{16}



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
```

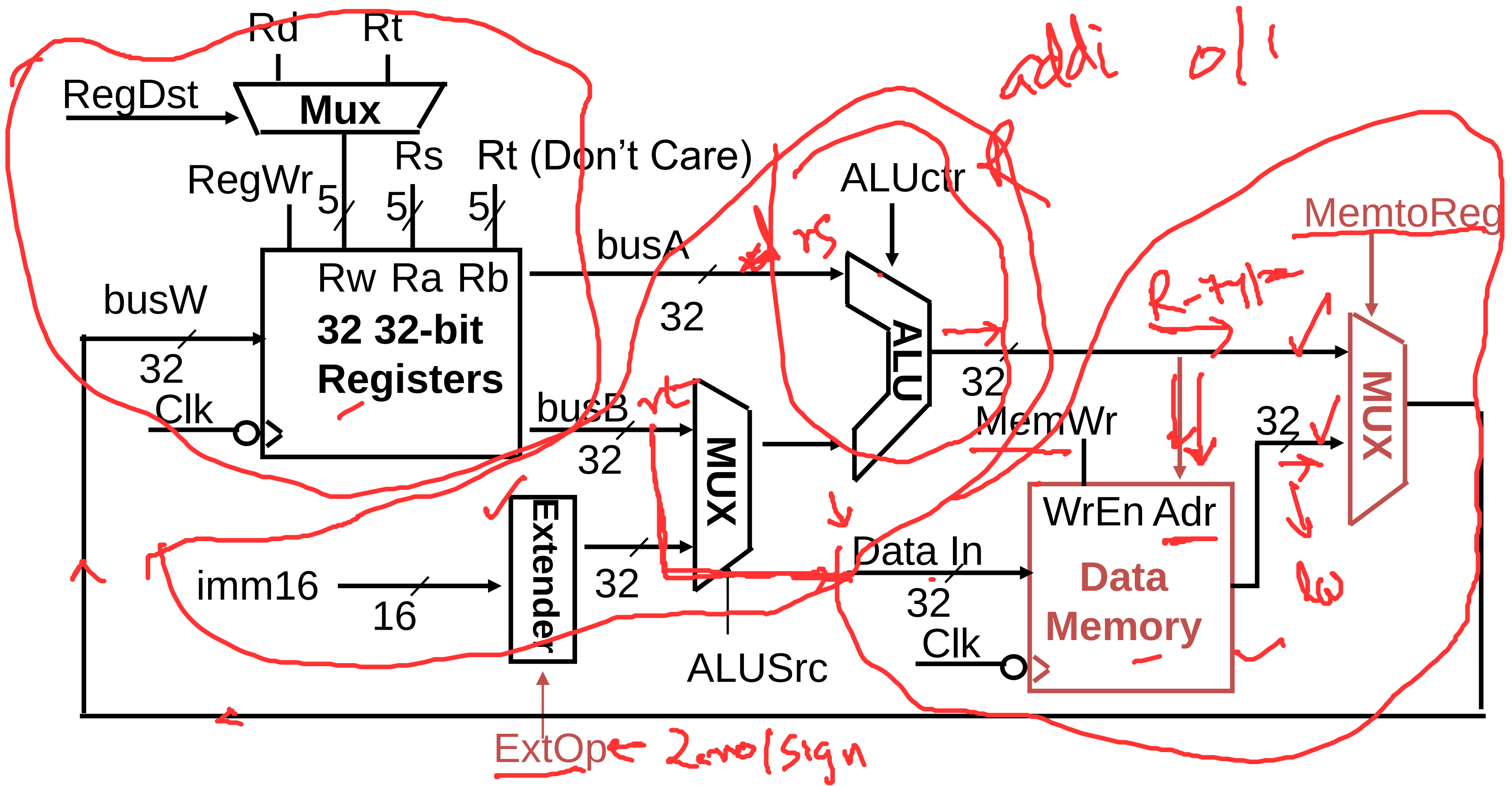
```
Mem[PC];                      Fetch instruction from memory
Addr <- R[rs]+ SignExt(imm);  Compute memory address
R[rt] <- Mem[Addr];           Load data into register
PC <- PC + 4;                 Calculate next address
```

Bits	6	5	5	16
	OP	rs	rt	imm
		first source register	second register (dest)	immediate

Datapath: Load

sw rt, rs, 44(नंबर)

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



Store RTL

- Store instruction

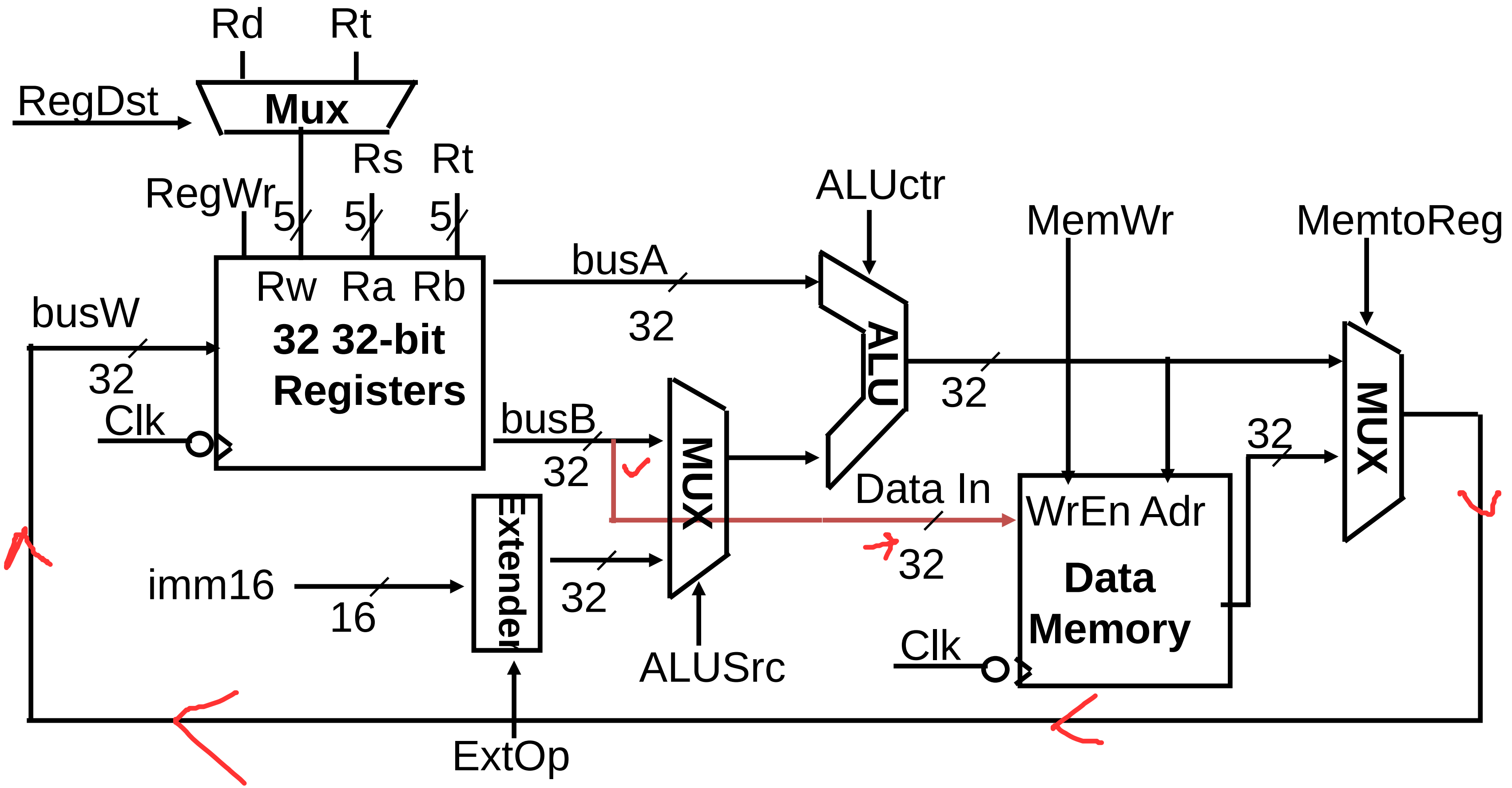
```
sw rt, rs, imm
```

```
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

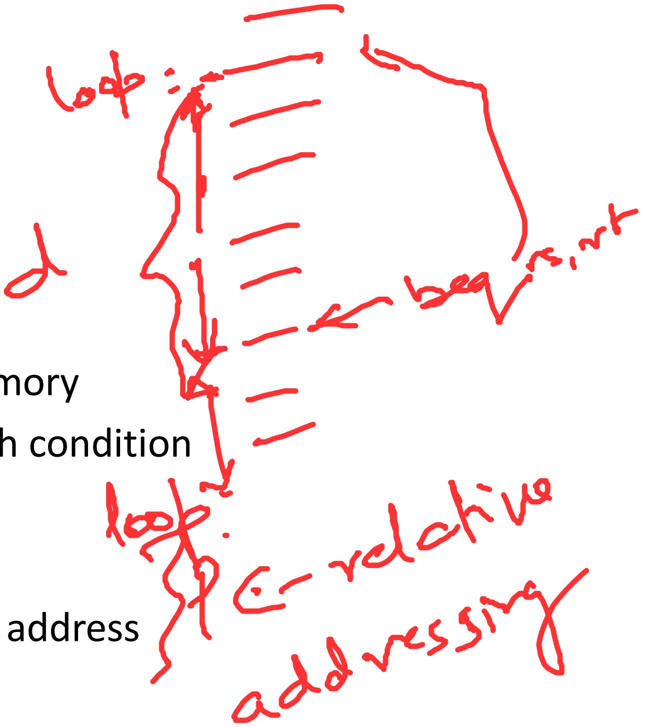


Branch RTL

- Branch instruction

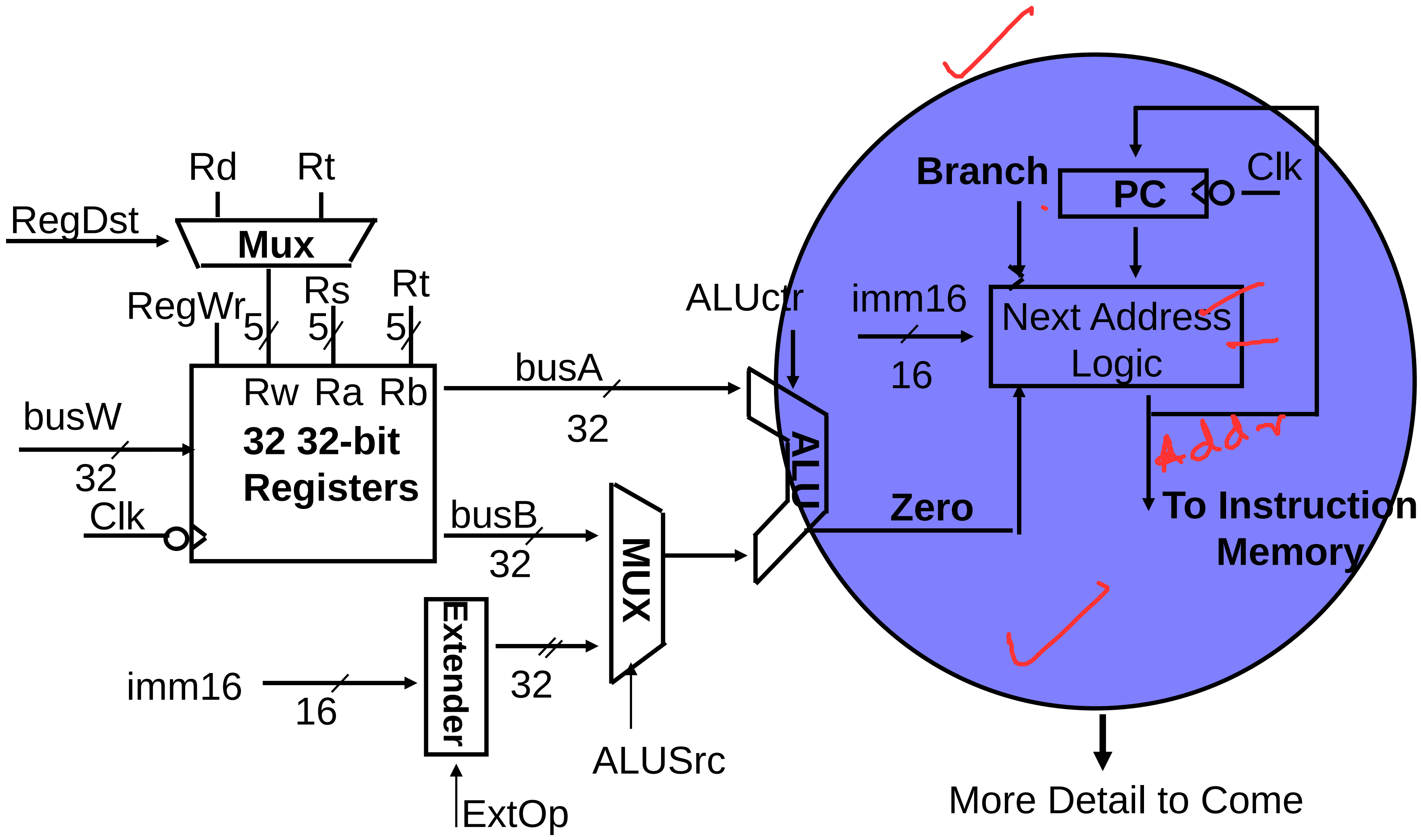
```
beq rs, rt, imm

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	6	5	5	16
	OP	rs	rt	imm
		first source	second source	immediate

Datapath: Branch



The Next Address

- PC is byte-addressed in instruction memory
  - Sequential

    `PC[31:0] = PC[31:0] + 4` 0+1

  - Branch operation

    `PC[31:0] = PC[31:0] + 4 + SignExt(imm) × 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

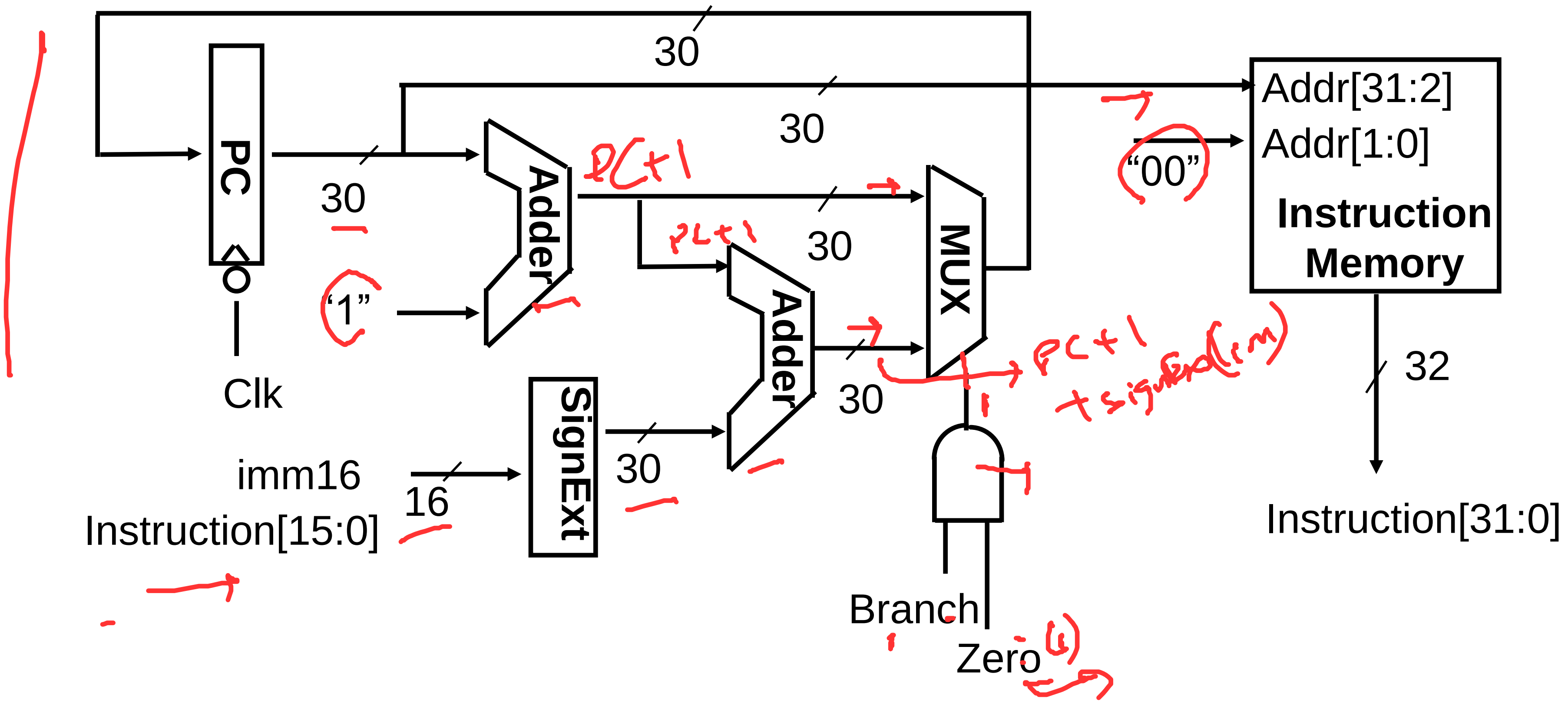
    `PC[31:2] = PC[31:2] + 1`

  - Branch operation

    `PC[31:2] = PC[31:2] + 1 + SignExt(imm)`

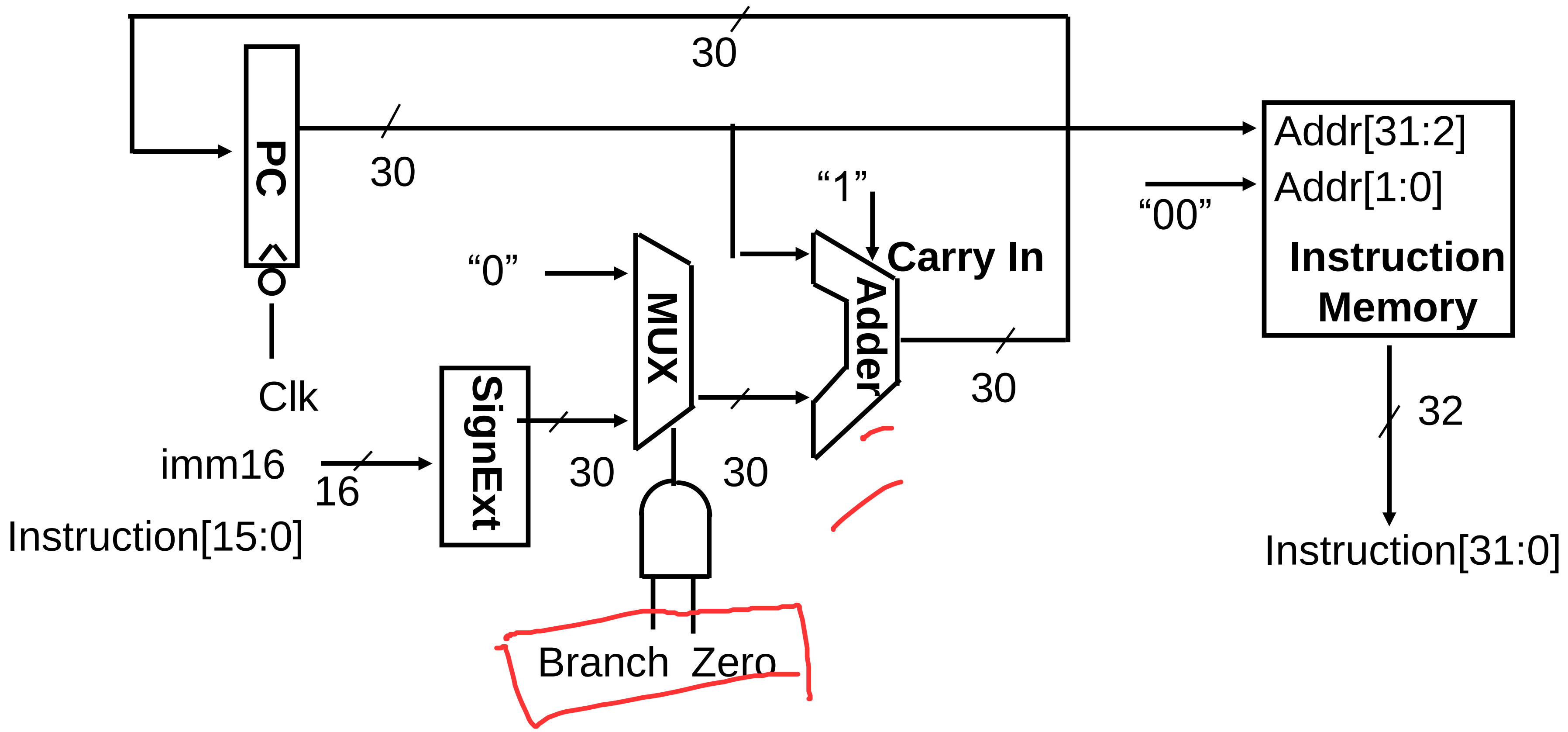
Datapath: Fast, Expensive Next-IF Logic

- PC incremented to next instruction normally
- On beq instruction then can add immediate × 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

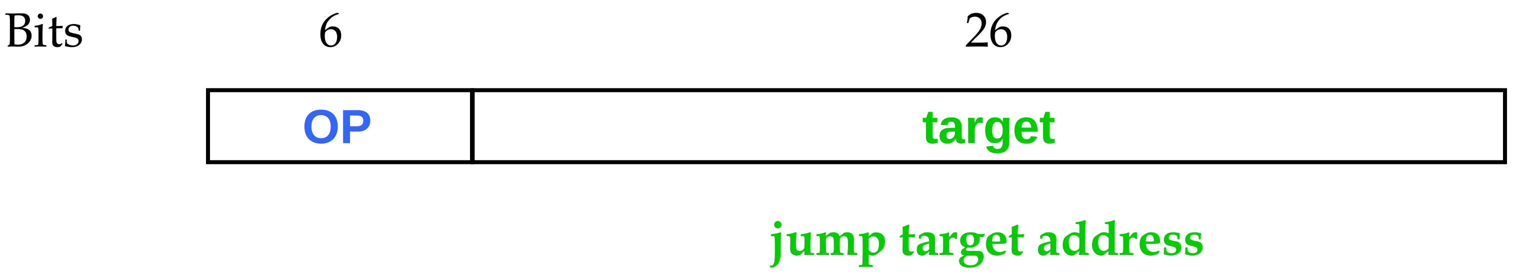
- Jump instruction

```
j target
```

```
Mem[PC];
PC[31:2] <- PC[31:28] ||
  target[25:0];
```

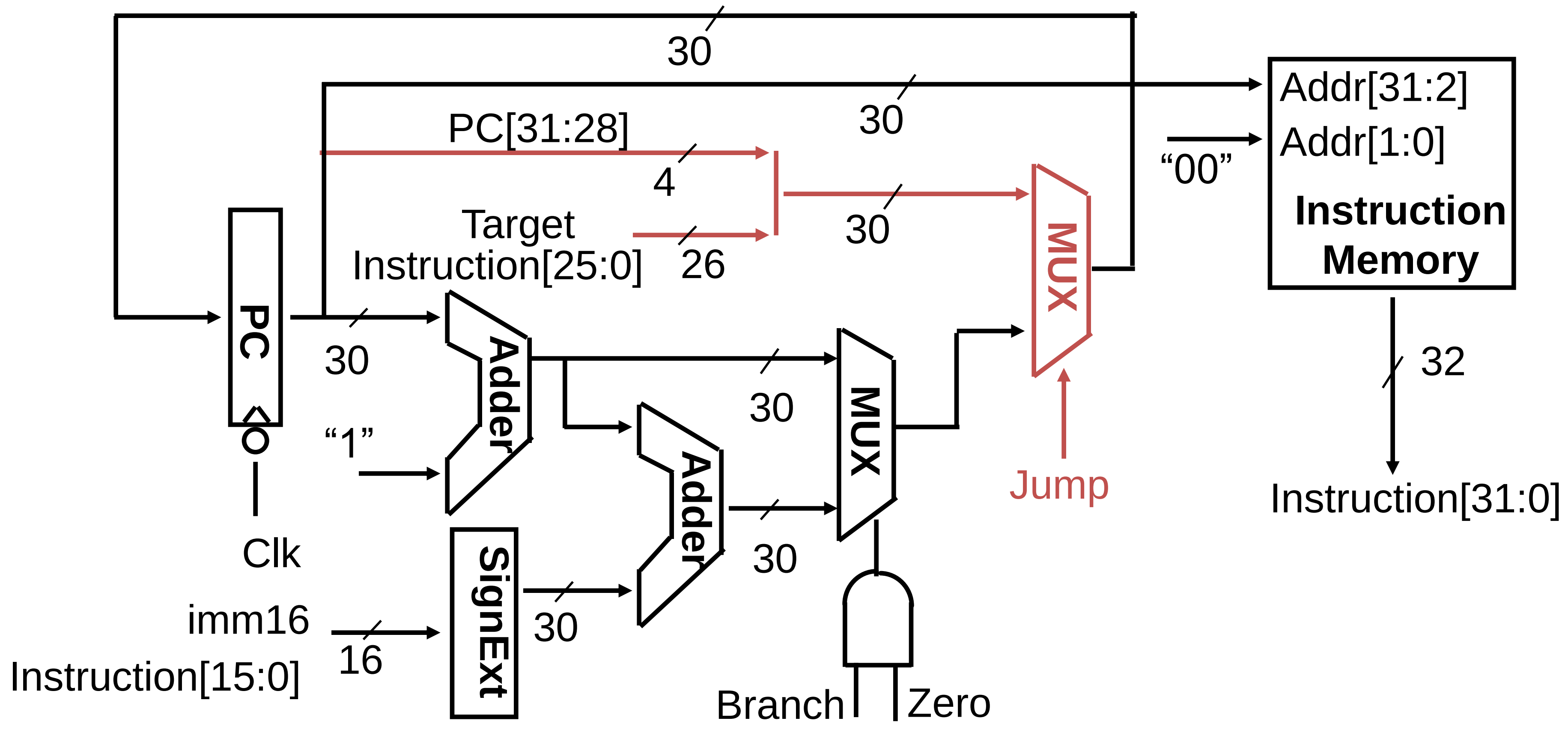
Fetch instruction from memory

Calculate next address

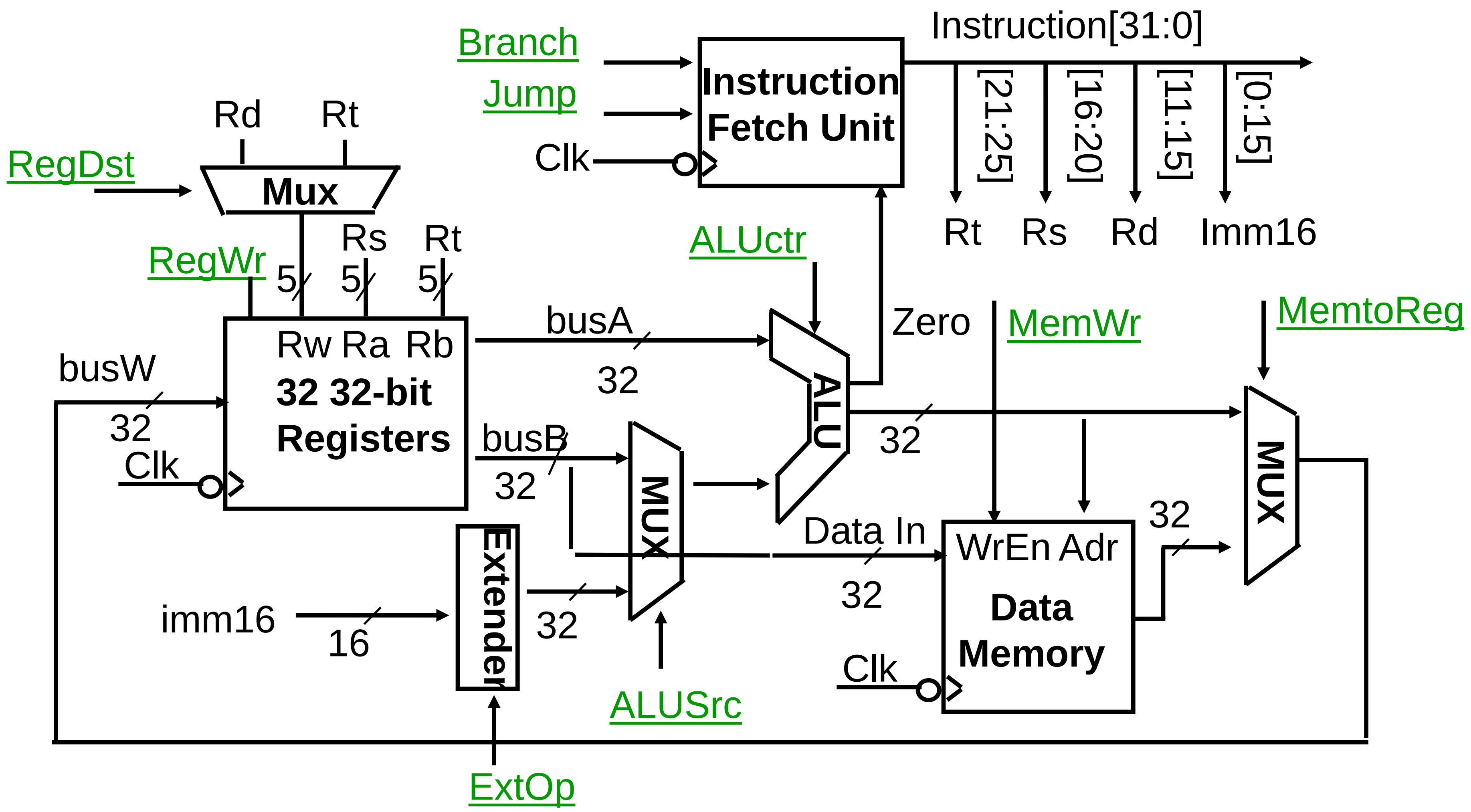


Datapath: IFU with Jump

- MUX controls if PC is pseudodirect jump



Putting it All Together


Branch
Jump
Clk
Instruction Fetch Unit
Instruction[31:0]
[21:25]
[16:20]
[11:15]
[0:15]
Rt
Rs
Rd
Imm16
RegDst
Rd
Rt
Mux
RegWr
5
Rs
5
Rt
5
ALUctr
Zero
MemWr
MemtoReg
busA
32
busW
32
Clk
Rw Ra Rb
32 32-bit Registers
busB
32
ALU
32
MUX
MUX
Data In
32
WrEn Adr
Data Memory
Clk
32
imm16
16
Extender
32
ALUSrc
ExtOp