# 111
# *Single Cycle CPU*

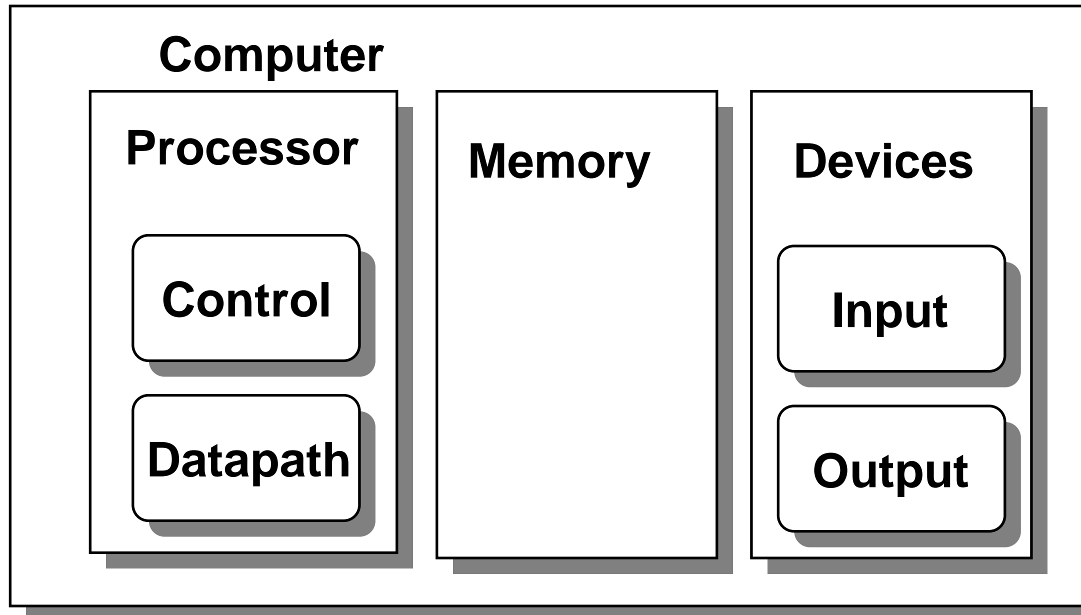
ENGR 3410 – Computer Architecture

Mark L. Chang

Fall 2006

# Datapath & Control

- Readings 5.1-5.4

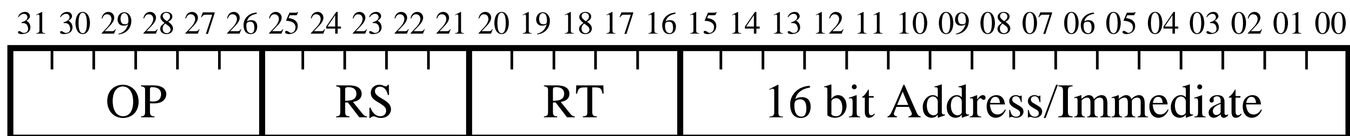



- Datapath: System for performing operations on data, plus memory access.
- Control: Control the datapath in response to instructions.

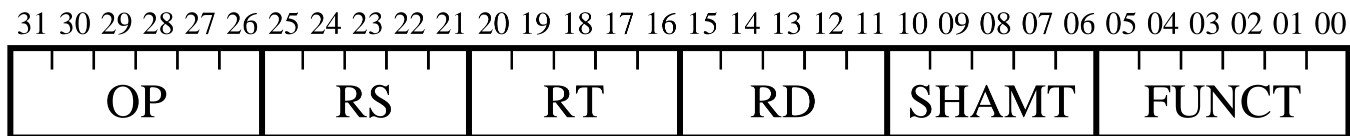
# Simple CPU

Develop complete CPU for subset of instruction set

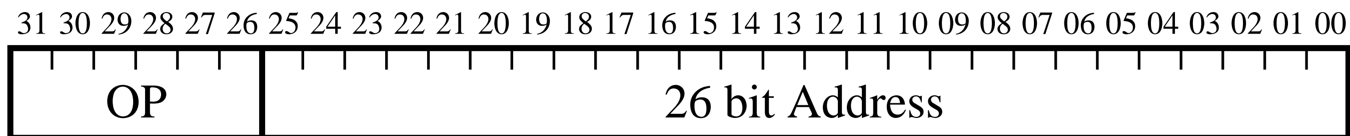
- Memory: `lw, sw`
- Branch: `beq`
- Arithmetic: `addi`




- Arithmetic: `add, sub`

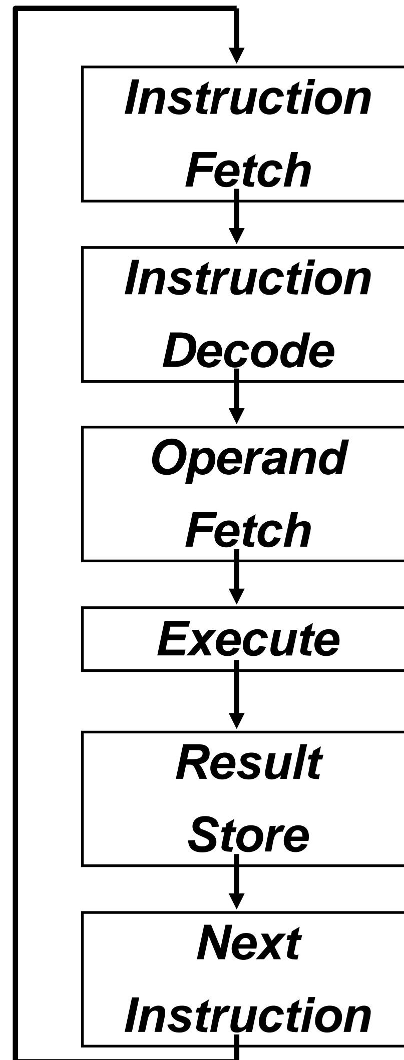



- Jump: `j`




Most other instructions similar

# Execution Cycle

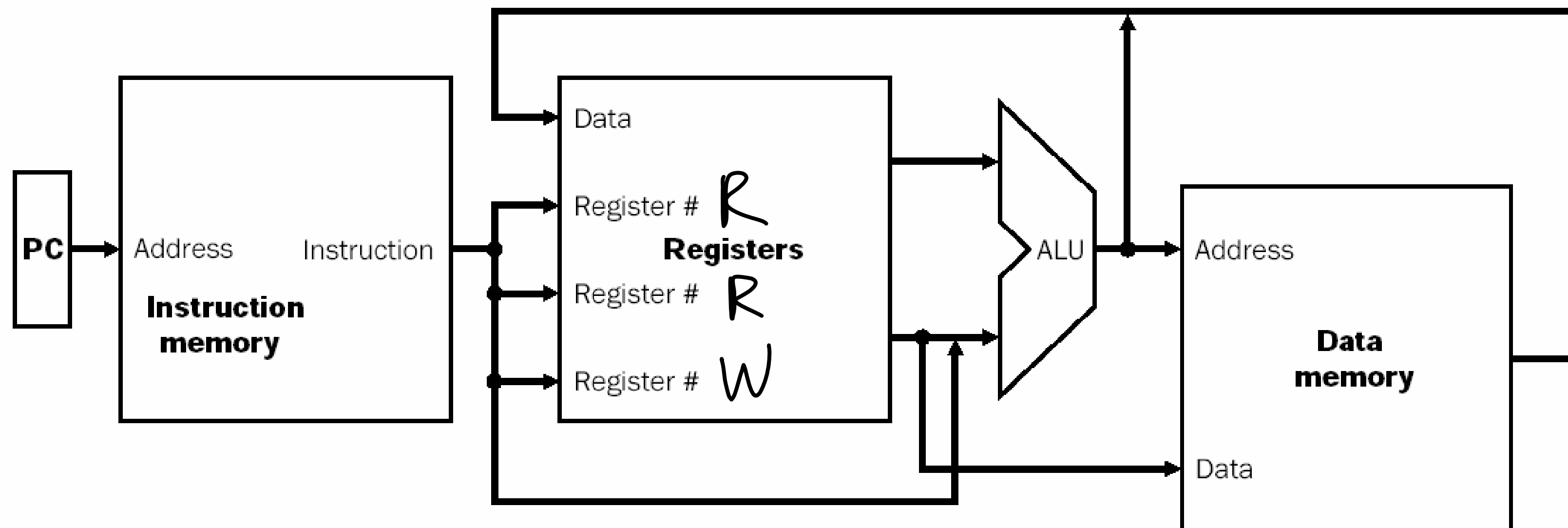



- Obtain instruction from program storage
- Determine required actions and instruction size
- Locate and obtain operand data
- Compute result value or status
- Deposit results in storage for later use
- Determine successor instruction

# Processor Overview

Overall Dataflow

- PC fetches instructions
- Instructions select operand registers, ALU immediate values
- ALU computes values
- Load/Store addresses computed in ALU
- Result goes to register file or Data memory



# Processor Design

Convert instructions to Register Transfer Level (RTL) specification

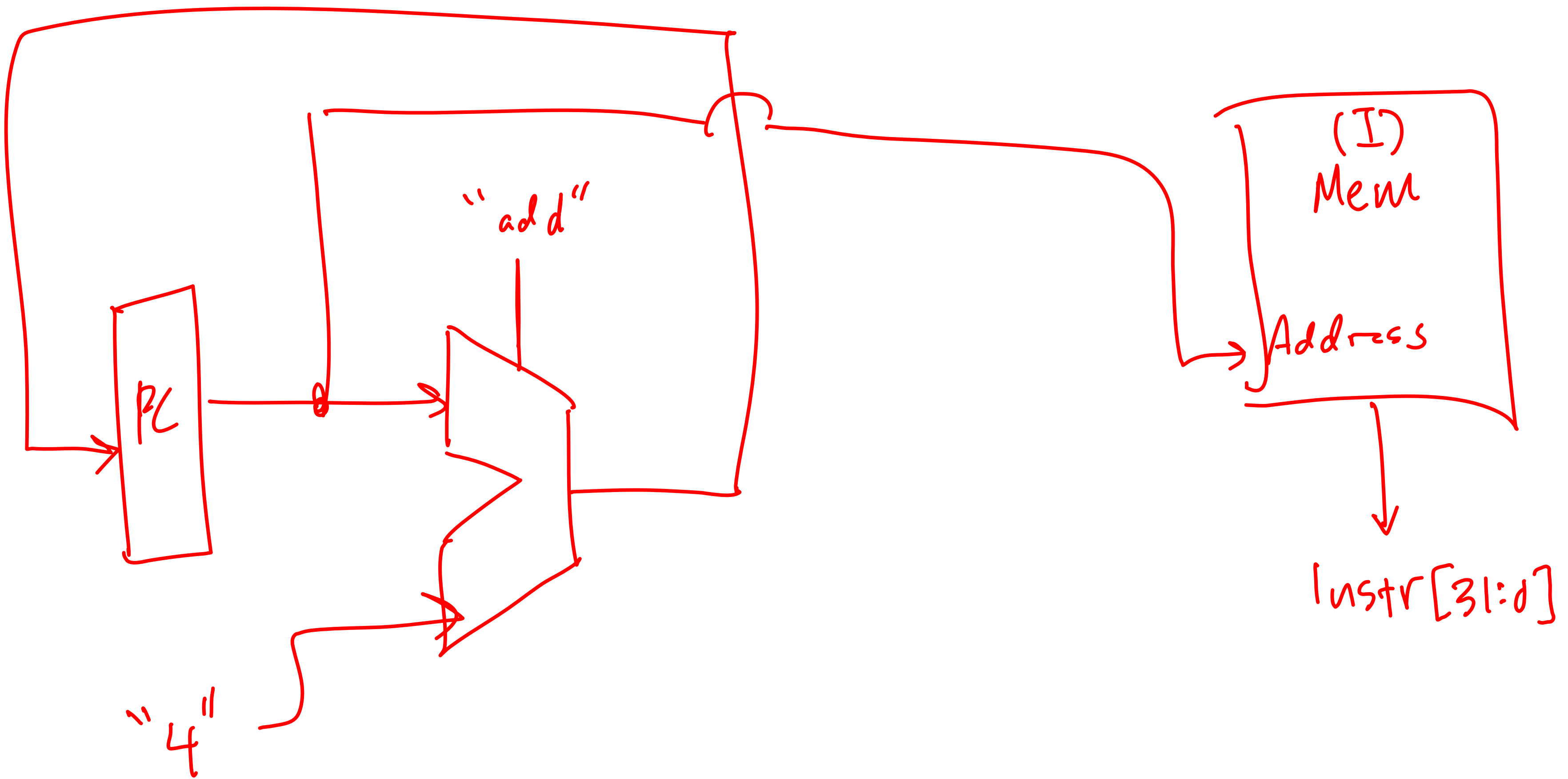
```
Instruction = Memory[PC];
PC = PC + 4;
```

RTL specifies required interconnection of units

Control designed to achieve given paths for each instruction

# Instruction Fetch

```
Instruction = Mem[PC];      // Fetch Instruction
PC = PC + 4;                // Increment PC (32bit)
```



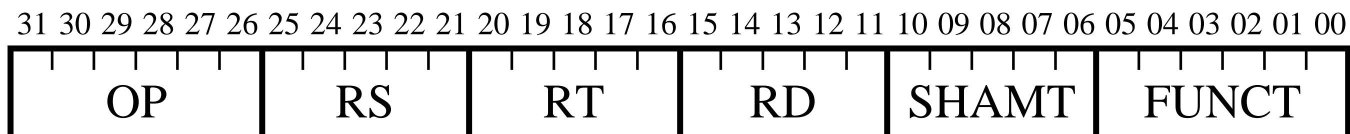
# Add/Subtract RTL

Add instruction: `add rd, rs, rt`

```
Instruction = Mem[PC];
Reg[rd] = Reg[rs] + Reg[rt];
PC = PC + 4;
```

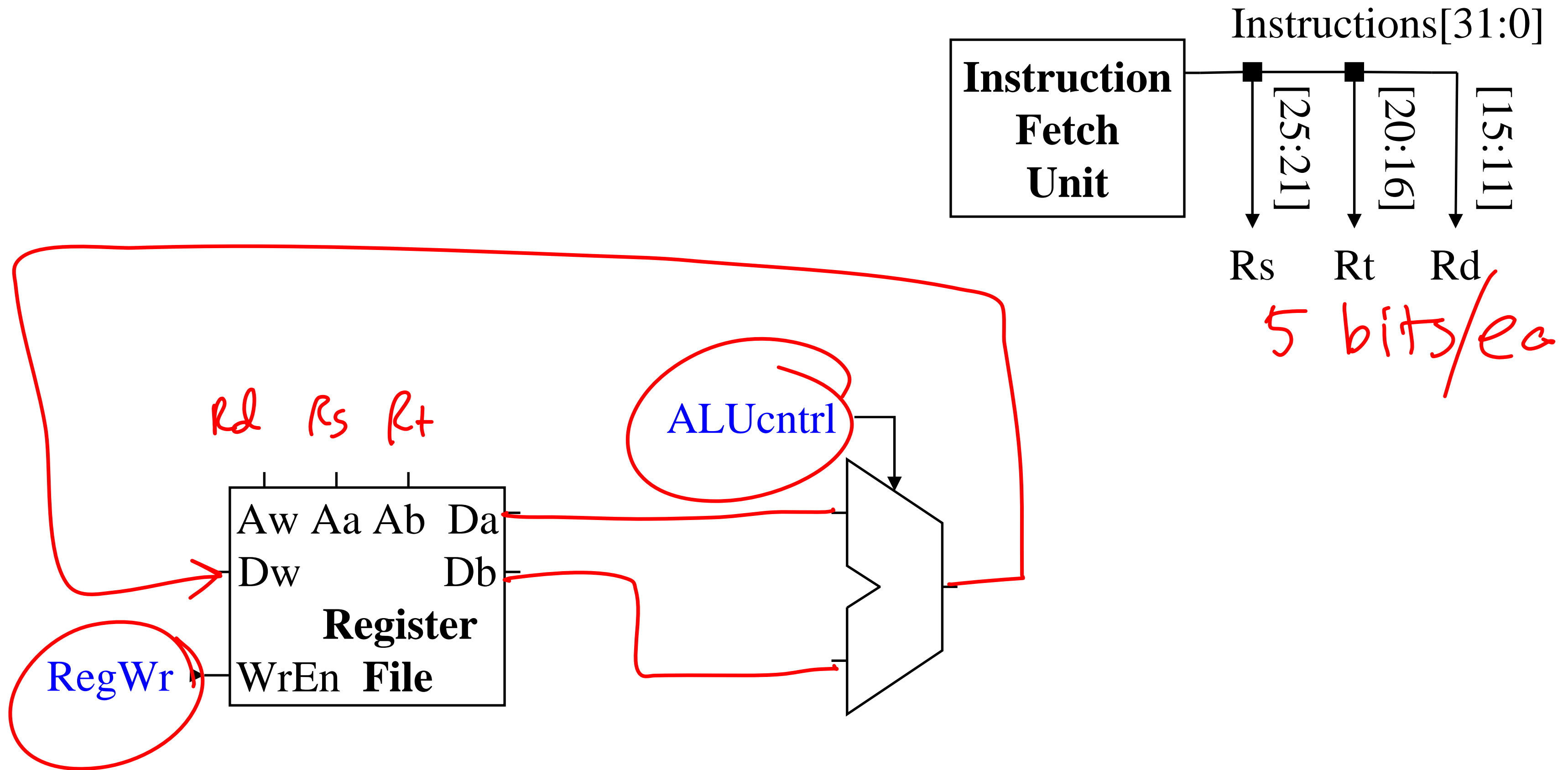
Subtract instruction: `sub rd, rs, rt`

```
Instruction = Mem[PC];
Reg[rd] = Reg[rs] - Reg[rt];
PC = PC + 4;
```



# Datapath for Reg/Reg Ops

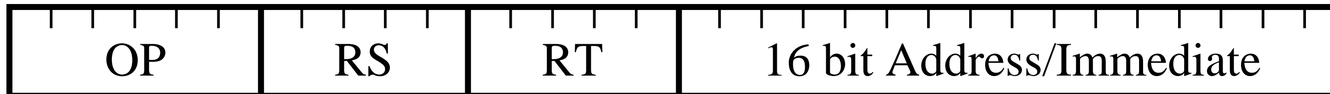
```
Reg[rd] = Reg[rs] op Reg[rt];
```



# Add Immediate RTL

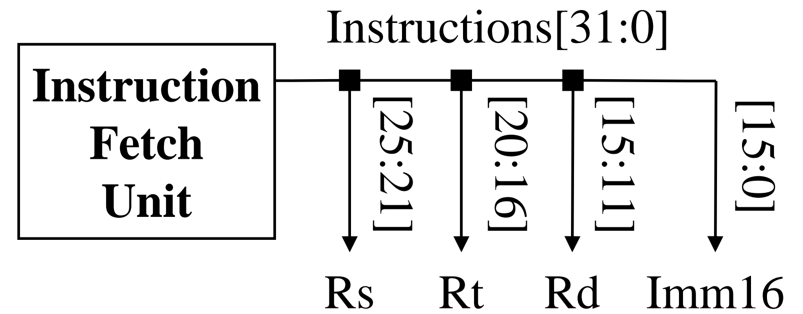
Add immediate instruction: `addi rt, rs, imm`

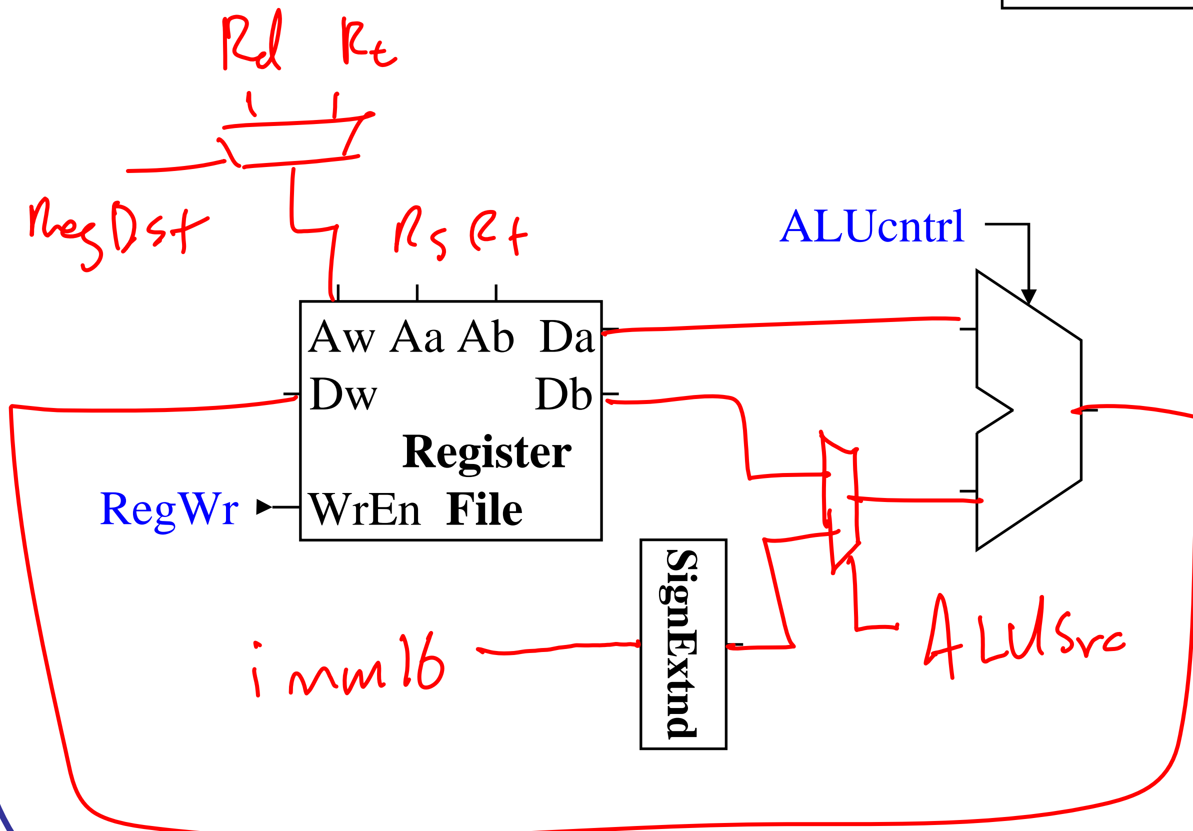
```
Instruction = Mem[PC];
Reg[rt] = Reg[rs] + SignExtend(imm);
PC = PC + 4;
```



# Datapath + Immediate Ops

```
Reg[rt] = Reg[rs] + SignExtend(imm);
```

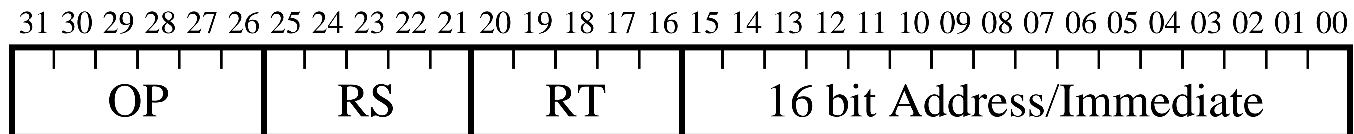


# Load RTL

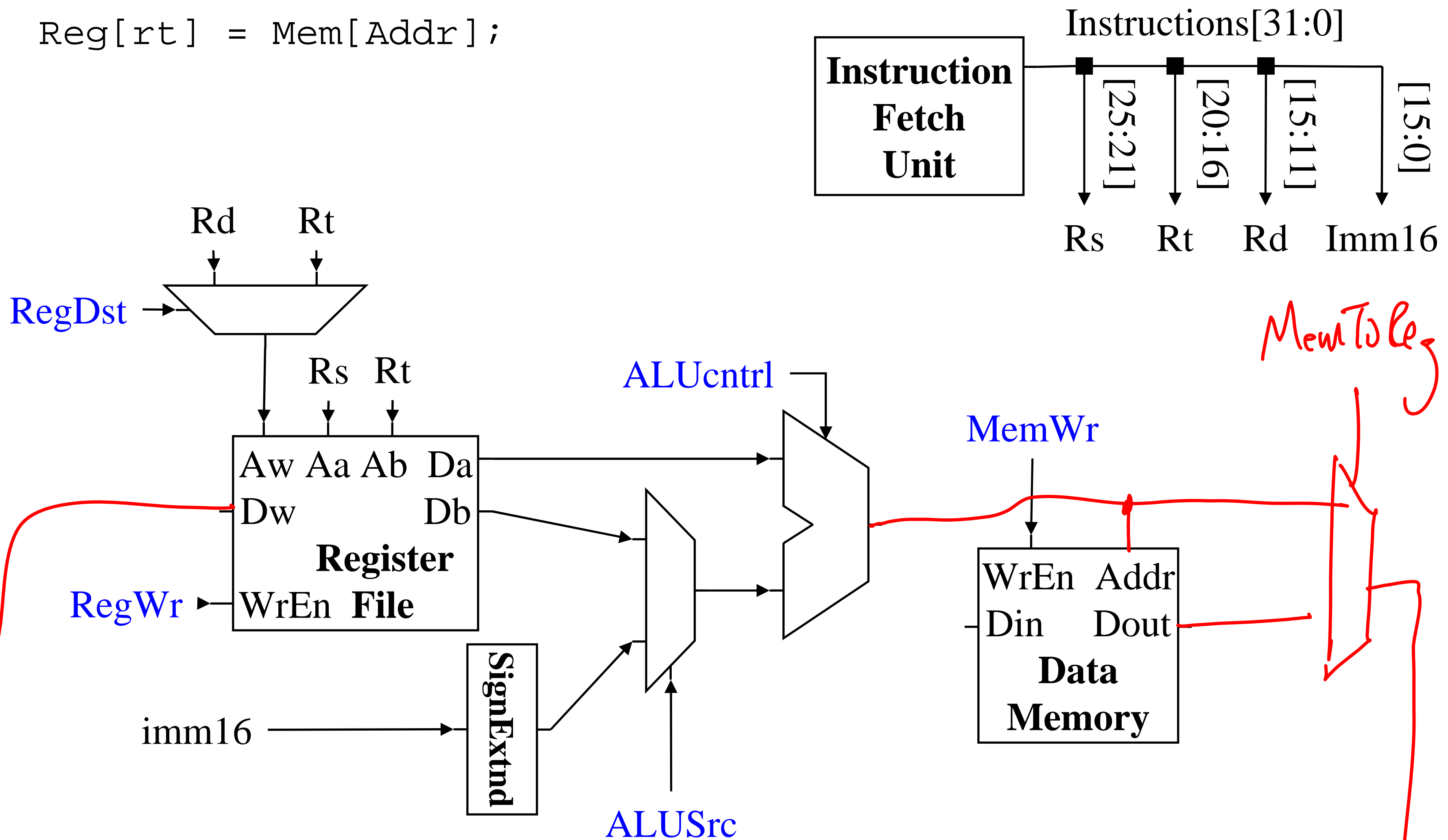
Load Instruction: `lw rt, imm(rs)`

```
Instruction = Mem[PC];
Addr = Reg[rs] + SignExtend(imm);
Reg[rt] = Mem[Addr];
PC = PC + 4;
```



# Datapath + Load

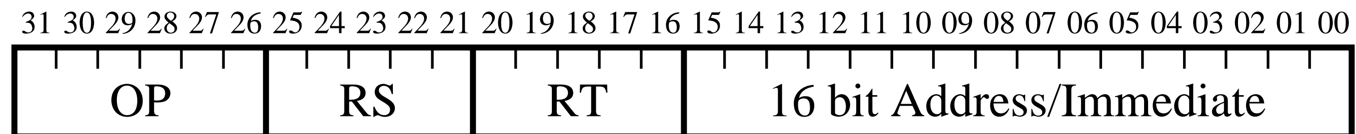
```
Addr = Reg[rs] + SignExtend(imm);
Reg[rt] = Mem[Addr];
```

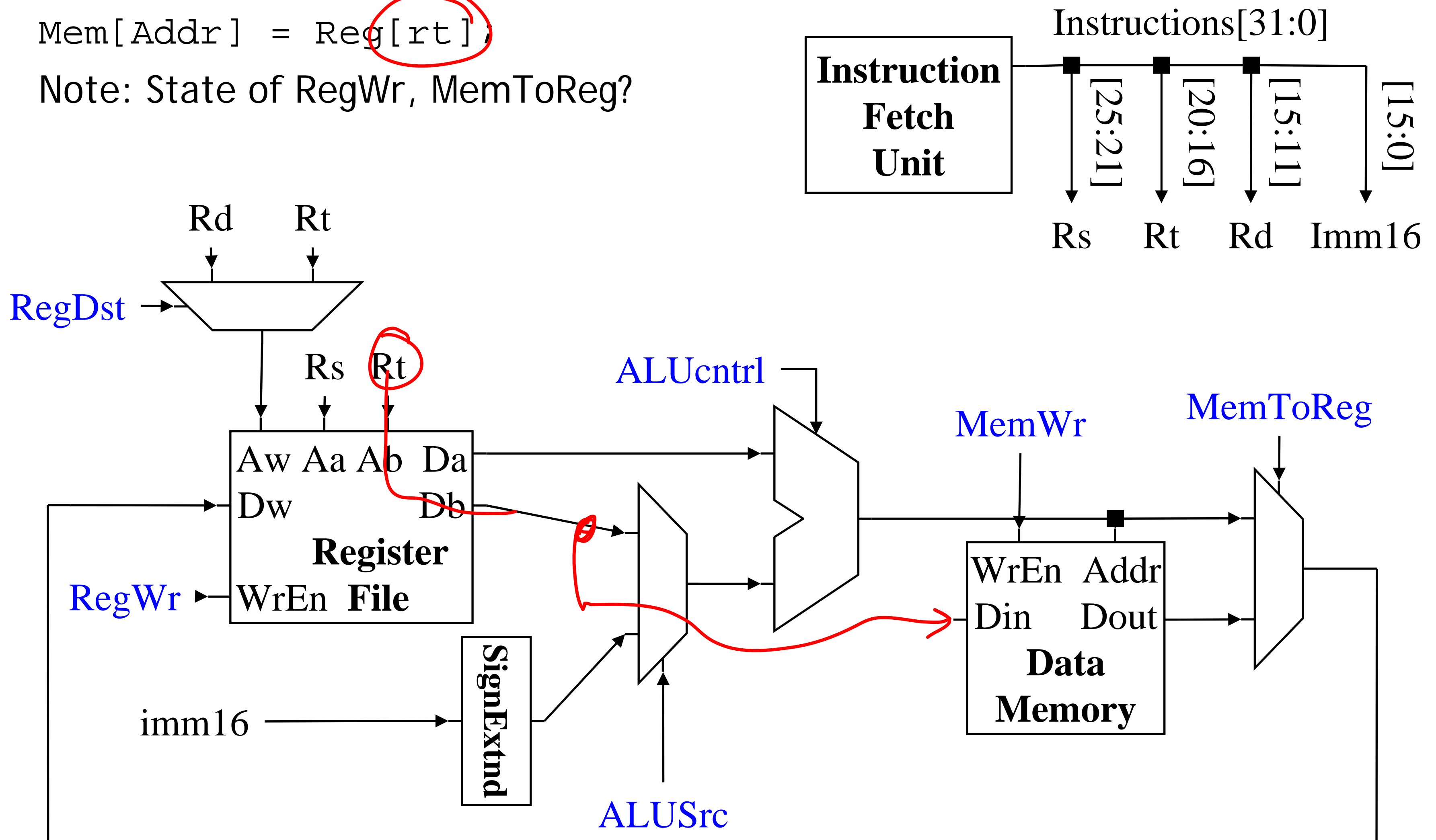


# Store RTL

Store Instruction: `sw rt, imm(rs)`

```
Instruction = Mem[PC];
Addr = Reg[rs] + SignExtend(imm);
Mem[Addr] = Reg[rt];
PC = PC + 4;
```



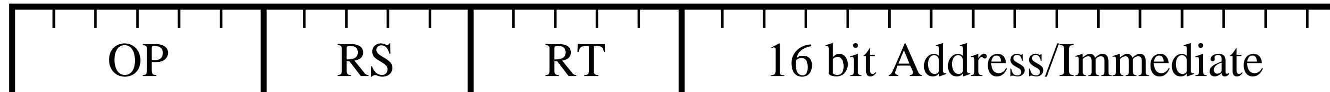


Datapath + Store
Addr = Reg[rs] + SignExtend(imm);
Mem[Addr] = Reg[rt];
Note: State of RegWr, MemToReg?
Instruction Fetch Unit
Instructions[31:0]
[25:21]
[20:16]
[15:11]
[15:0]
Rs
Rt
Rd
Imm16
Rd
Rt
RegDst
Rs
Rt
ALUcntrl
MemWr
MemToReg
Aw Aa Ab Da
Dw Db
Register File
RegWr
WrEn
WrEn Addr
Din Dout
Data Memory
SignExtnd
imm16
ALUSrc

# Branch RTL

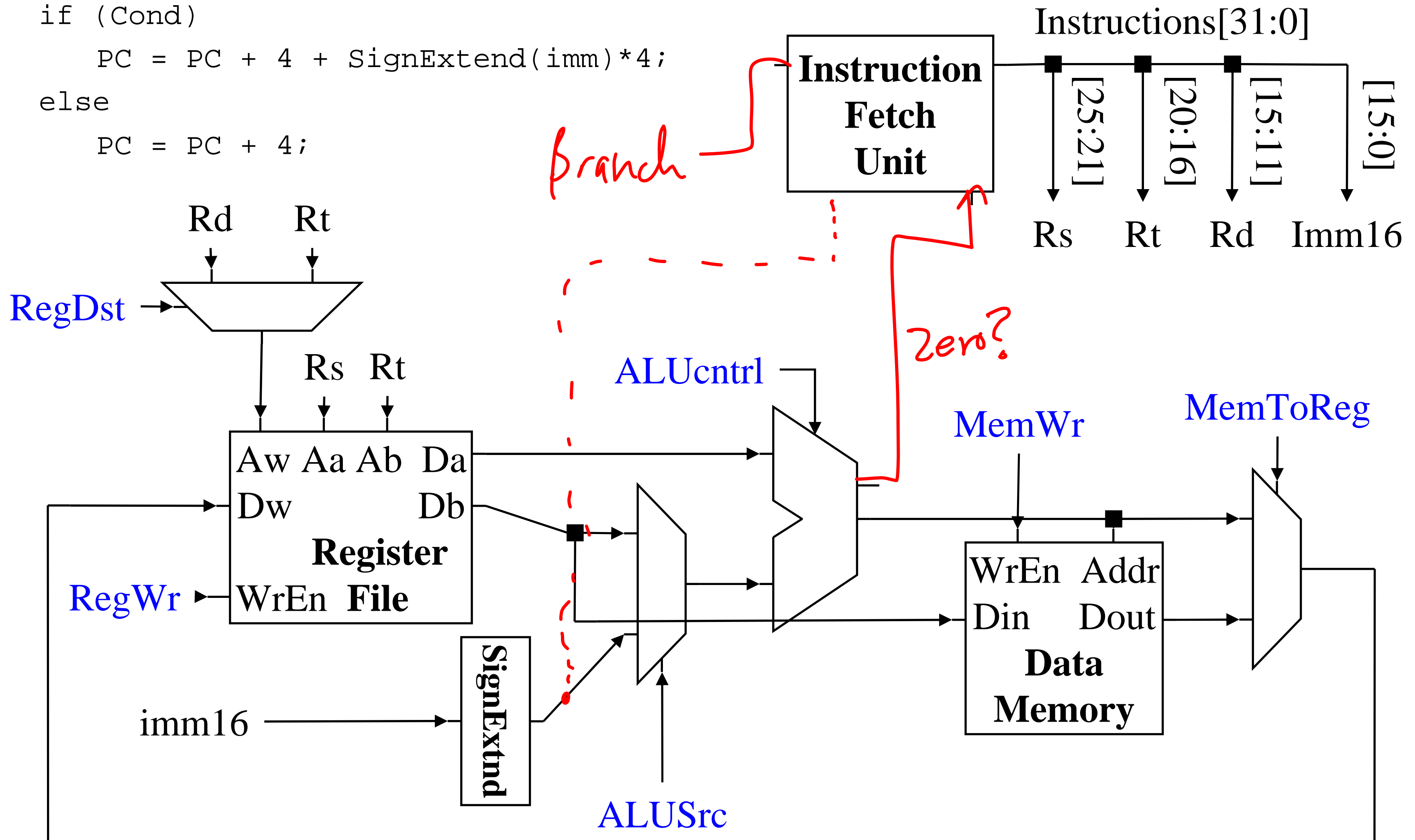
Branch Instruction: `beq rs, rt, imm`

```
Instruction = Mem[PC];
Cond = (Reg[rs] - Reg[rt]) == 0;   // Test equality
if (Cond)
     PC = PC + 4 + SignExtend(imm)*4;      // Neg for backward
                                           // *4: LSbits == 00
else
     PC = PC + 4;
```



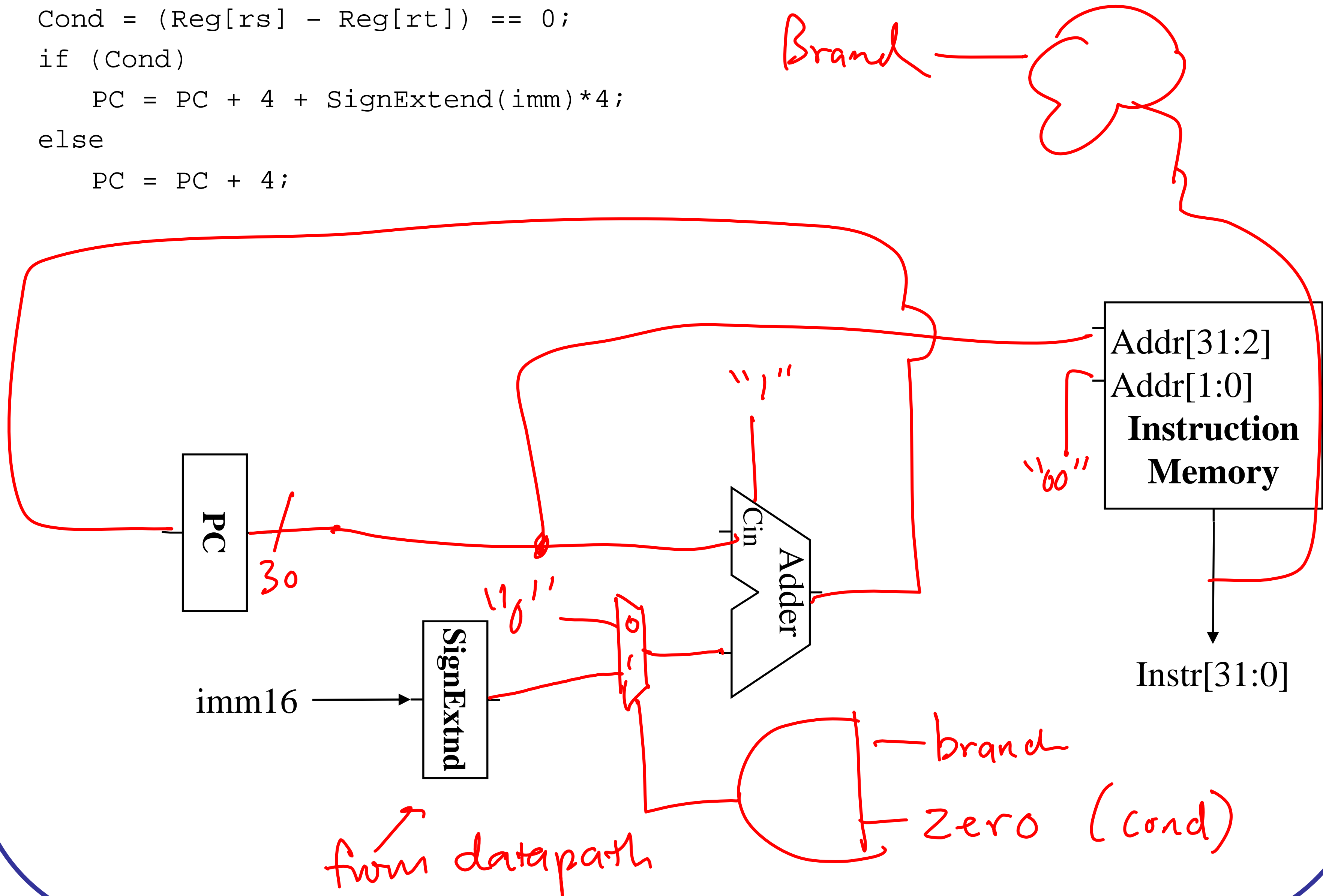
# Datapath + Branch

```
Cond = (Reg[rs] - Reg[rt]) == 0;
if (Cond)
    PC = PC + 4 + SignExtend(imm)*4;
else
    PC = PC + 4;
```



# Instruction Fetch + Branch

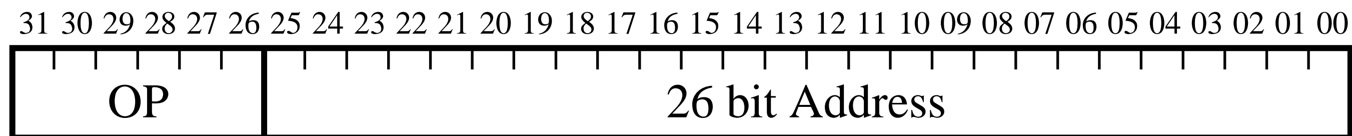
```
Cond = (Reg[rs] - Reg[rt]) == 0;
if (Cond)
    PC = PC + 4 + SignExtend(imm)*4;
else
    PC = PC + 4;
```



# Jump RTL

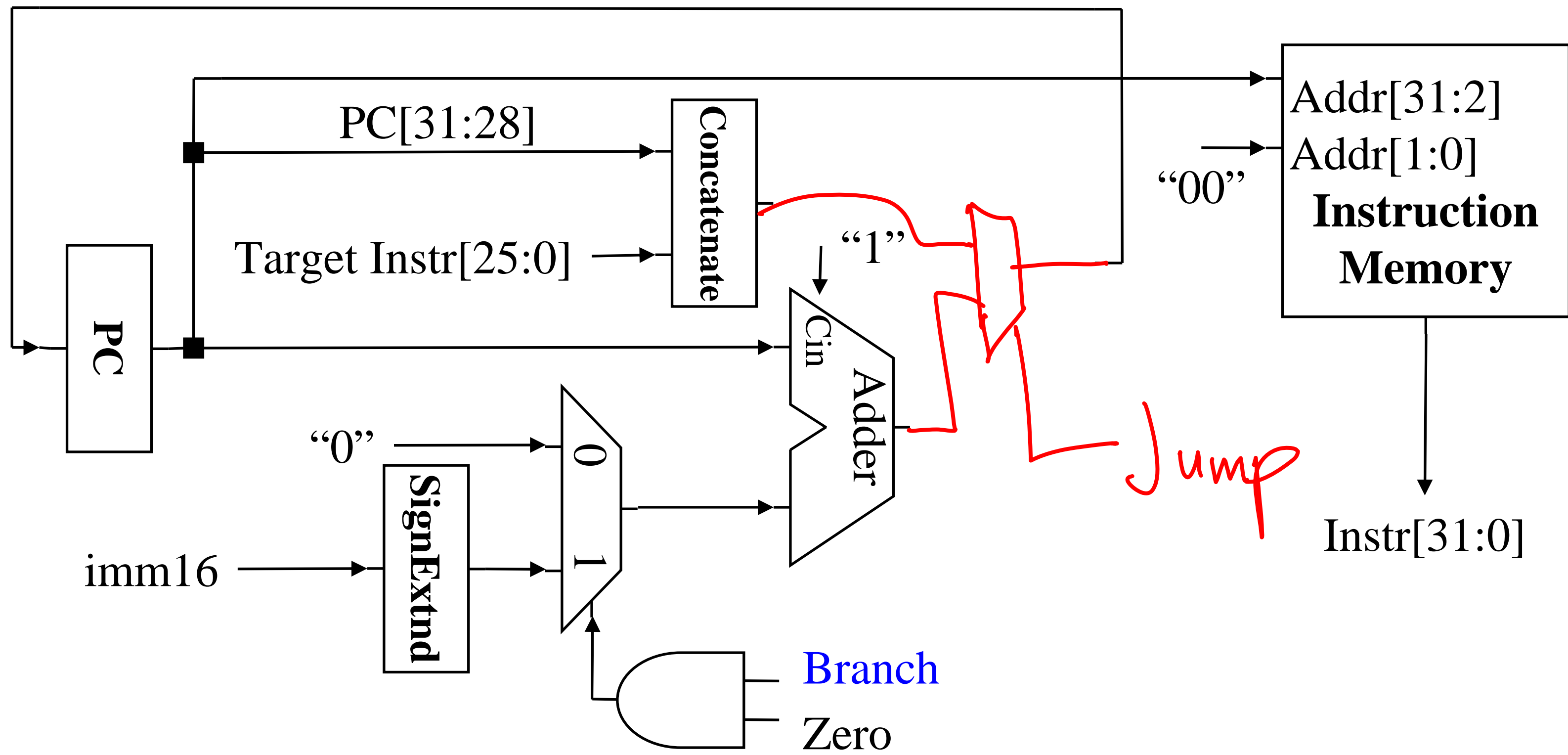
Store Instruction: `j target`

```
Instruction = Mem[PC];
PC = { PC[31:28], target[25:0], “00” };
```

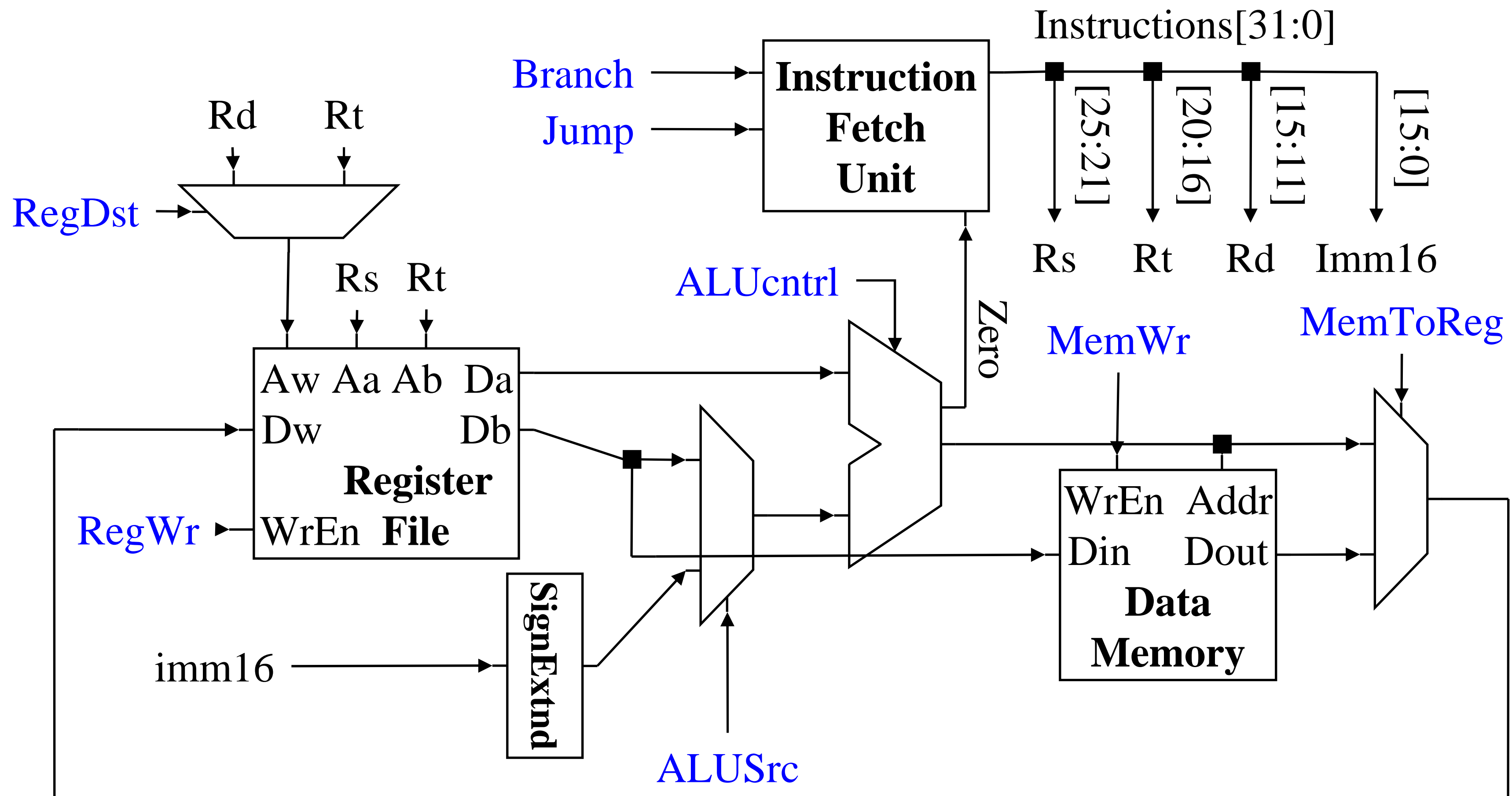


# Instruction Fetch + Jump

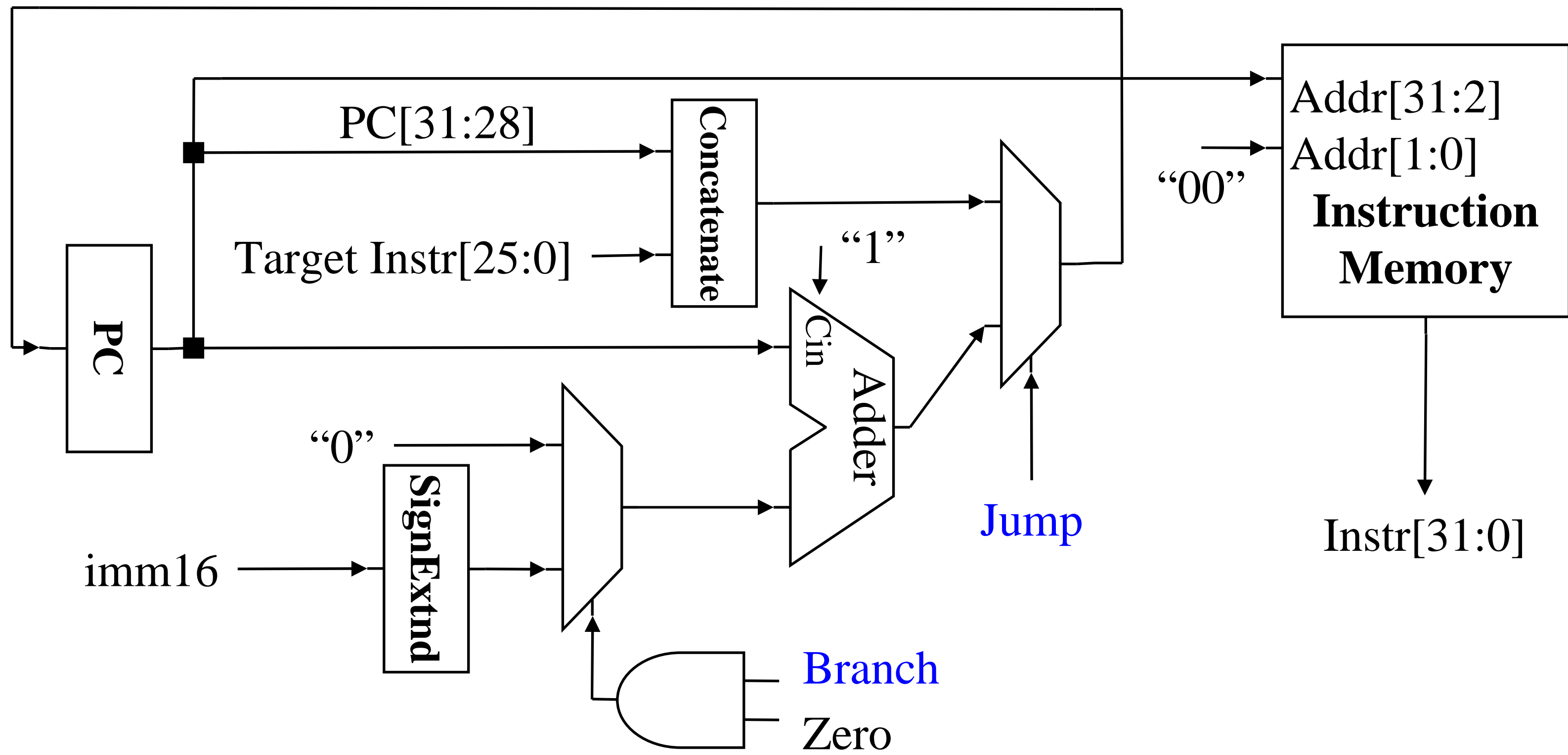
```
PC = { PC[31:28], target[25:0], "00" };
```



# Complete Datapath



# Complete Fetch Unit



# Control

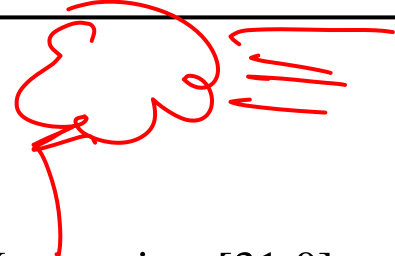
- Identify control points for pieces of datapath
  - Instruction Fetch Unit
  - ALU
  - Memories
  - Datapath muxes
  - Etc.
- Use RTL for determine per-instruction control assignments

# Control Signals

| Func | 100000 | 100010 | XXX | XXX | XXX | XXX |
|---|---|---|---|---|---|---|
| Op | 000000 | 000000 | 100011 | 101011 | 000100 | 000010 |
| | add | sub | lw | sw | beq | j |
| RegDst | | | | | | |
| ALUSrc | | | | | | |
| MemToReg | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

# Add Control

```
add rd, rs, rt
    Instruction = Mem[PC];
    Reg[rd] = Reg[rs] + Reg[rt];
    PC = PC + 4;
```



Instructions[31:0]

0 Branch
0 Jump
Instruction Fetch Unit

[25:21] Rs
[20:16] Rt
[15:11] Rd
[15:0] Imm16

1 RegDst
Rd Rt
1 0

Rs Rt
Aw Aa Ab Da
Dw Db
Register File
WrEn

RegWr 1

Add ALUcntrl

Zero

imm16
SignExtnd

0 1

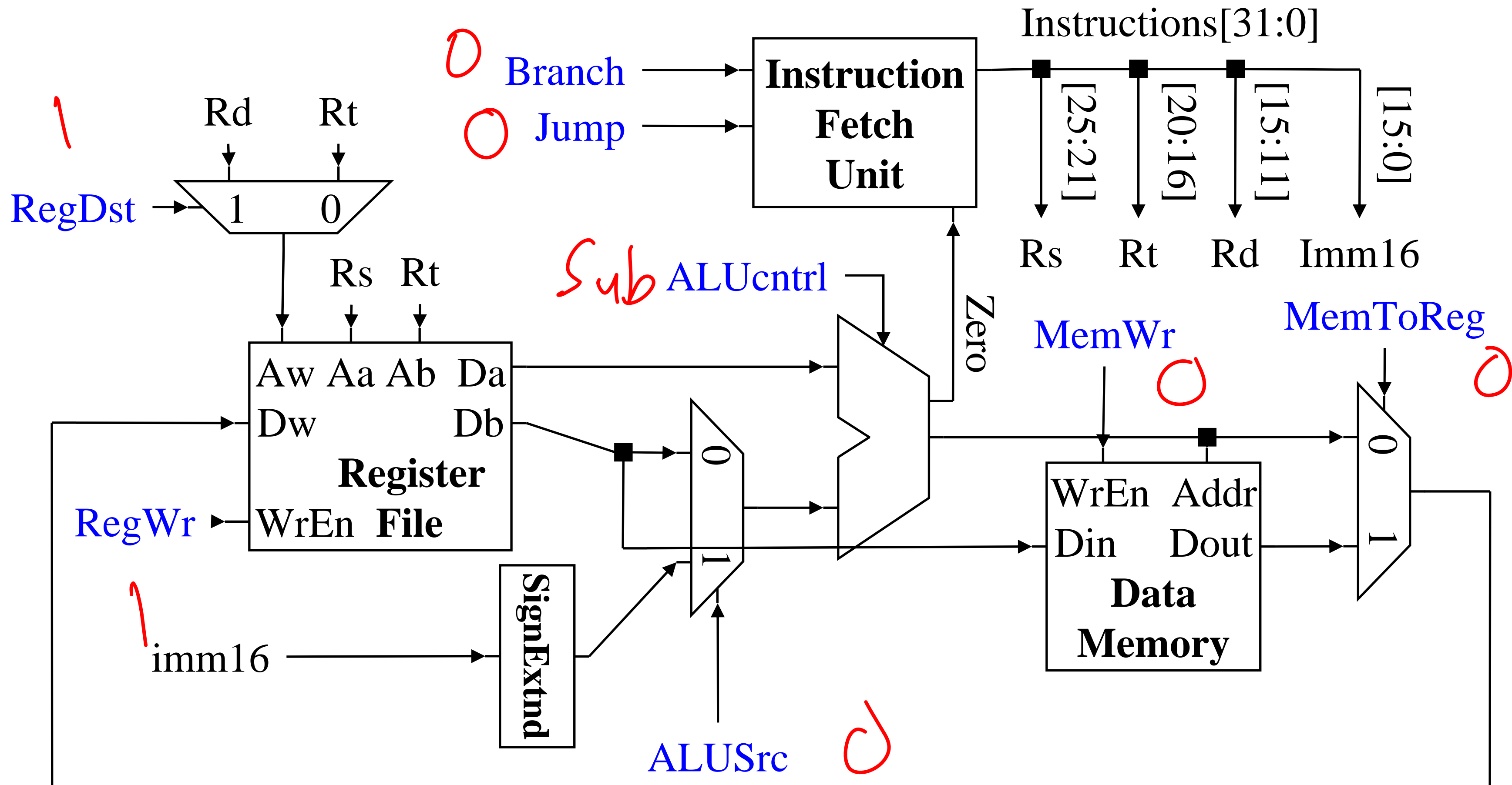
ALUSrc 0

MemWr 0
WrEn Addr
Din Dout
Data Memory

MemToReg 0

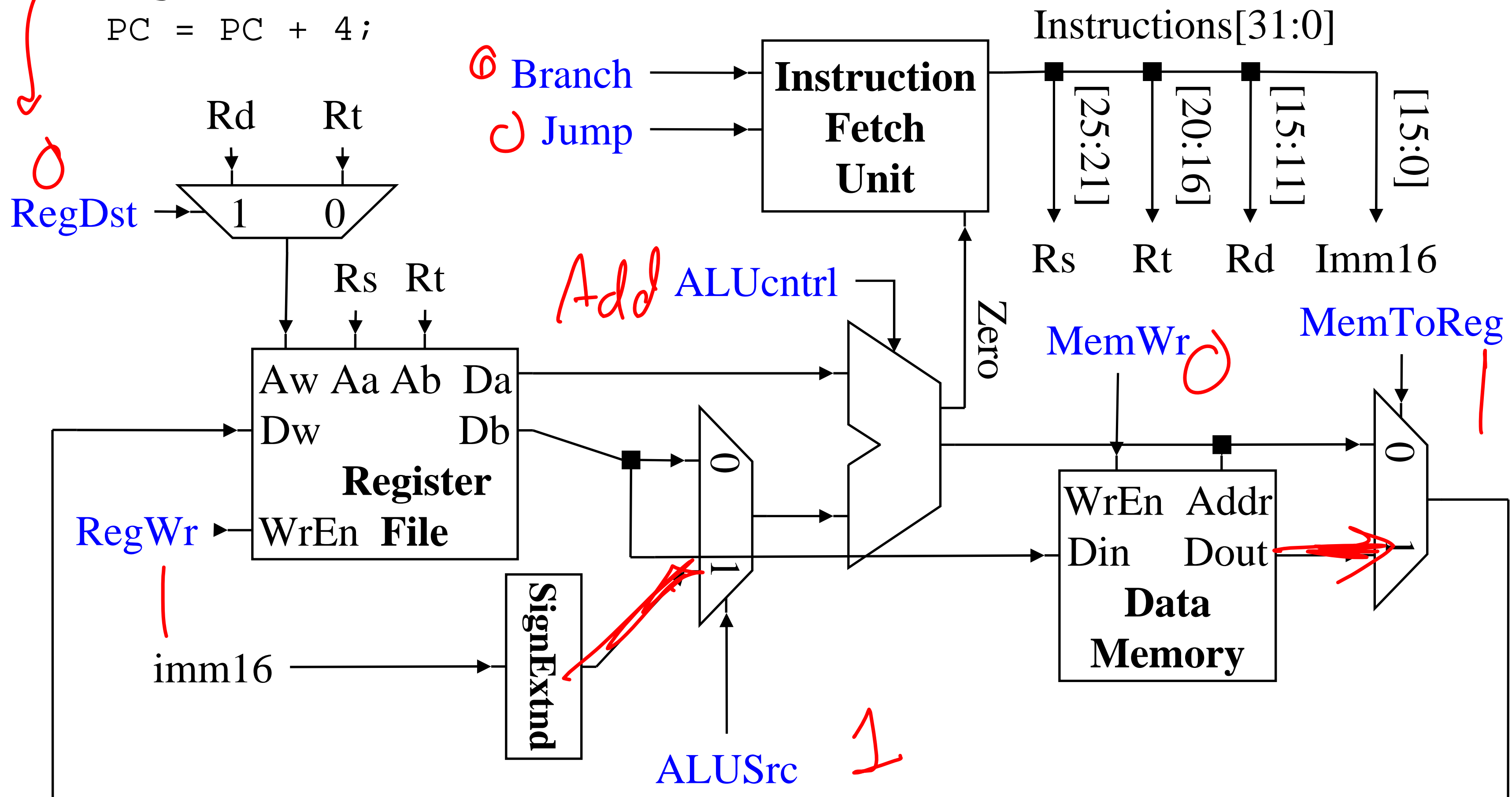
# Subtract Control

```
sub rd, rs, rt
   Instruction = Mem[PC];
   Reg[rd] = Reg[rs] - Reg[rt];
   PC = PC + 4;
```



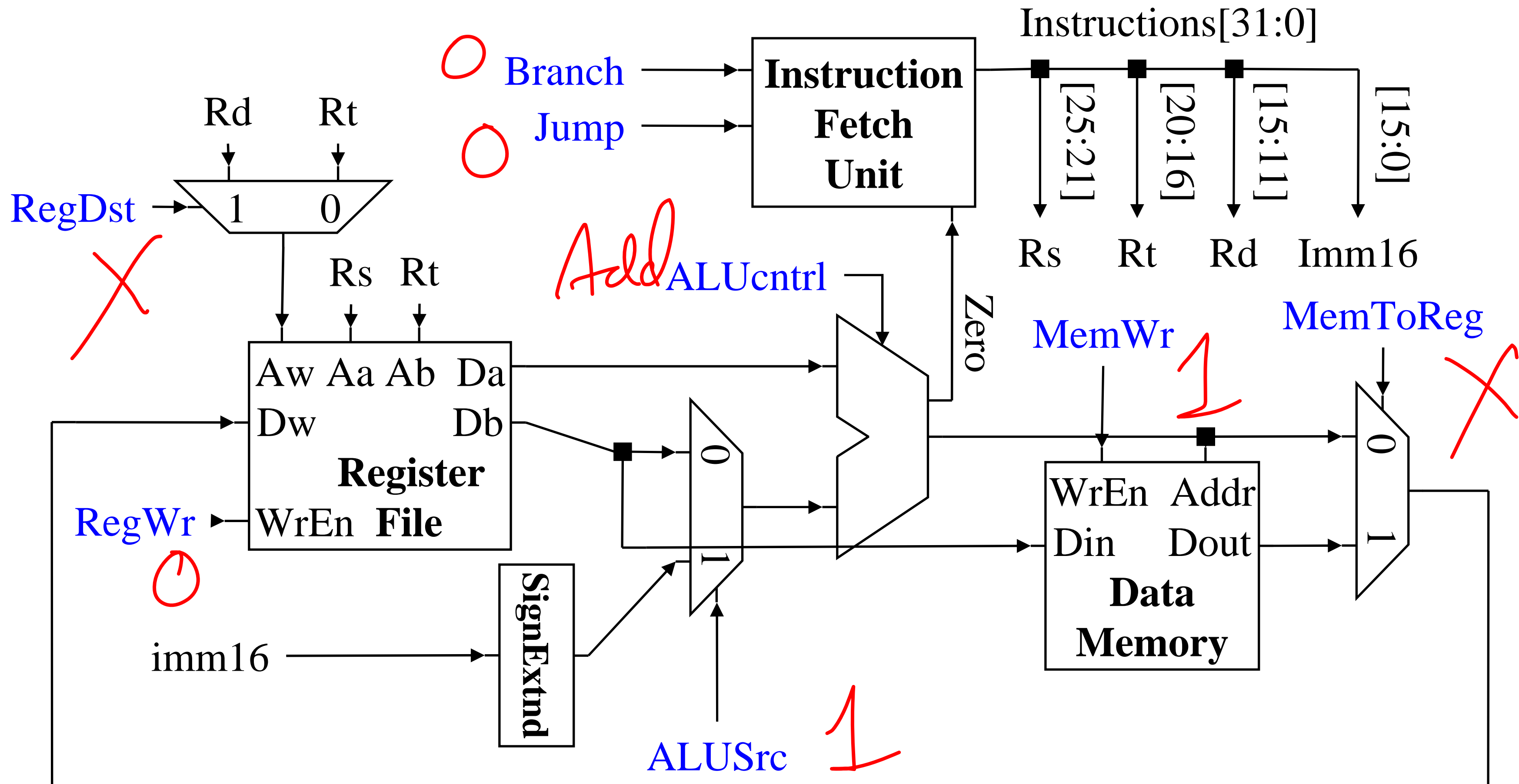
# Load Control

```
lw rt, imm(rs)
   Instruction = Mem[PC];
   Addr = Reg[rs] + SignExtend(imm);
   Reg[rt] = Mem[Addr];
   PC = PC + 4;
```



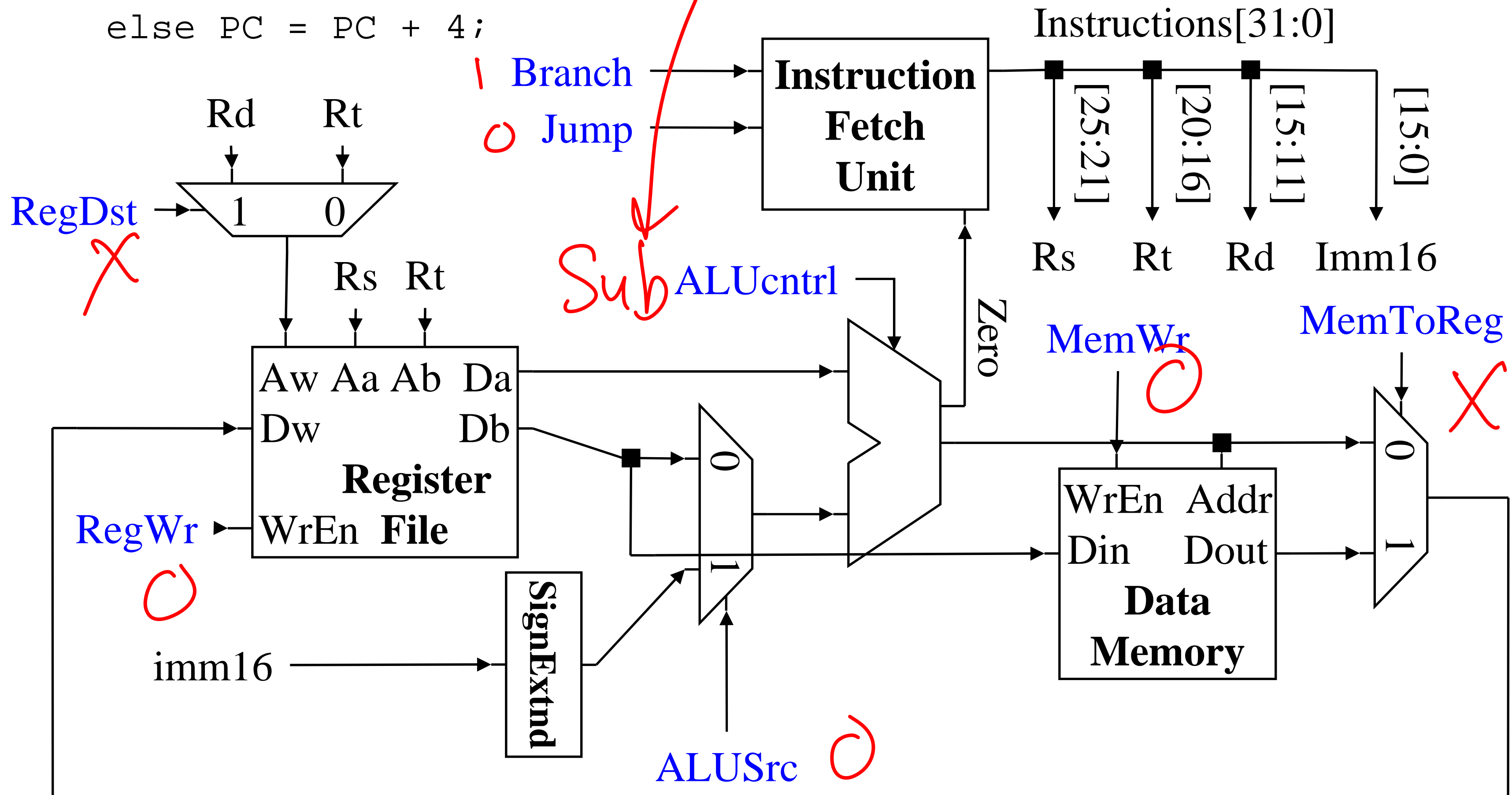
# Store Control

```
sw rt, imm(rs)
    Instruction = Mem[PC];
    Addr = Reg[rs] + SignExtend(imm);
    Mem[Addr] = Reg[rt];
```



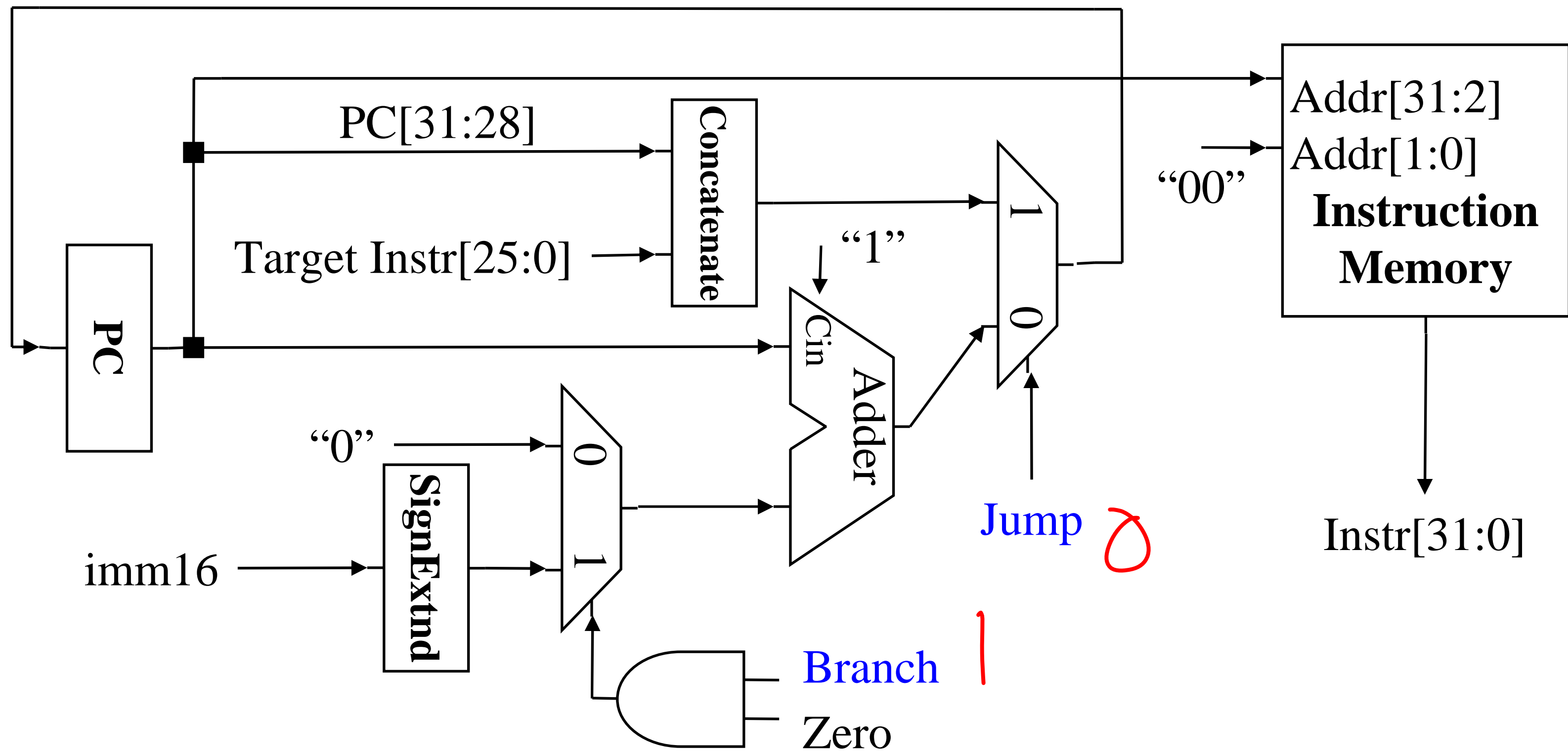
# Branch Control

```
beq rs, rt, imm
    Instruction = Mem[PC];
    Cond = (Reg[rs] - Reg[rt]) == 0;
    if (Cond) PC = PC+4+SignExtend(imm)*4;
    else PC = PC + 4;
```



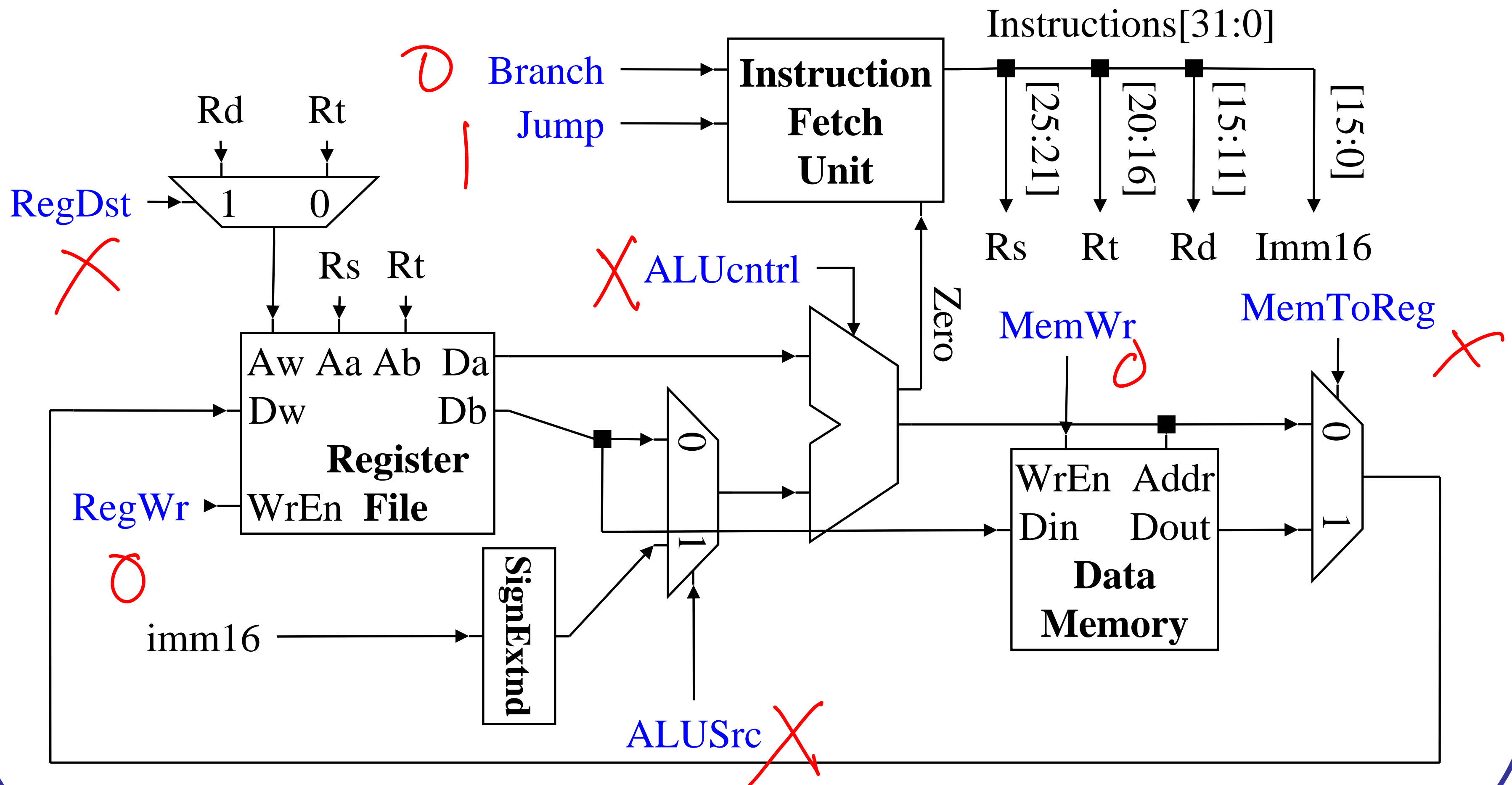
# Branch Control (cont.)

```
beq rs, rt, imm
    Instruction = Mem[PC];
    Cond = (Reg[rs] - Reg[rt]) == 0;
    if (Cond) PC = PC+4+SignExtend(imm)*4;
    else PC = PC + 4;
```



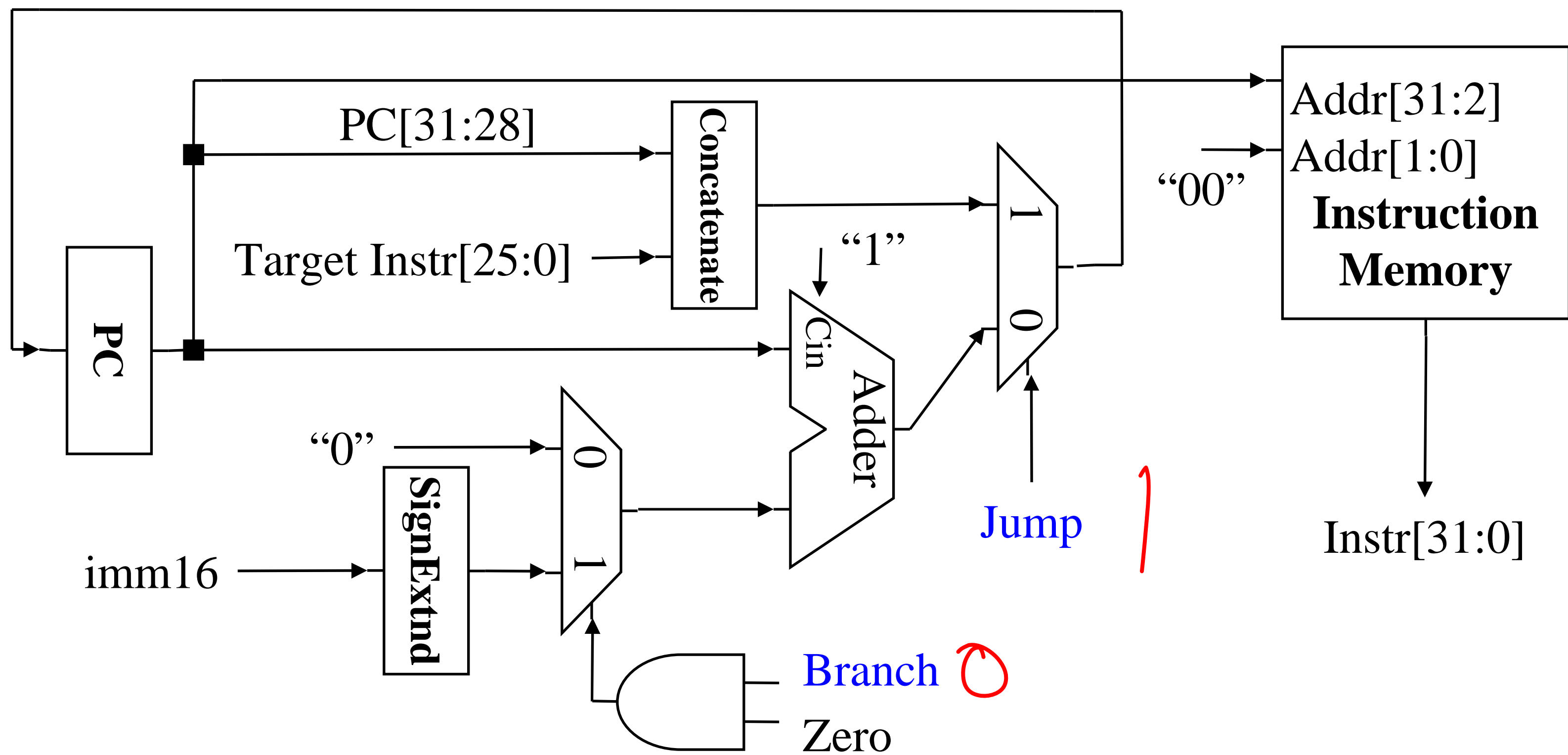
# Jump Control

```
j target
   Instruction = Mem[PC];
   PC = { PC[31:28], target[25:0], "00" };
```



# Jump Control (cont.)

```
j target
   Instruction = Mem[PC];
   PC = { PC[31:28], target[25:0], "00" };
```



# Control Signals

| | | | | | | |
|---|---|---|---|---|---|---|
| Func | 100000 | 100010 | XXX | XXX | XXX | XXX |
| Op | 000000 | 000000 | 100011 | 101011 | 000100 | 000010 |
| | add | sub | lw | sw | beq | j |
| RegDst | 1 | 1 | 0 | X | X | X |
| ALUSrc | 0 | 0 | 1 | 1 | 0 | X |
| MemToReg | 0 | 0 | 1 | X | X | X |
| RegWr | 1 | 1 | 1 | 0 | 0 | 0 |
| MemWr | 0 | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 0 | 1 | X |
| Jump | 0 | 0 | 0 | 0 | 0 | 1 |
| ALUCntrl | Add | Sub | Add | Add | Sub | X |