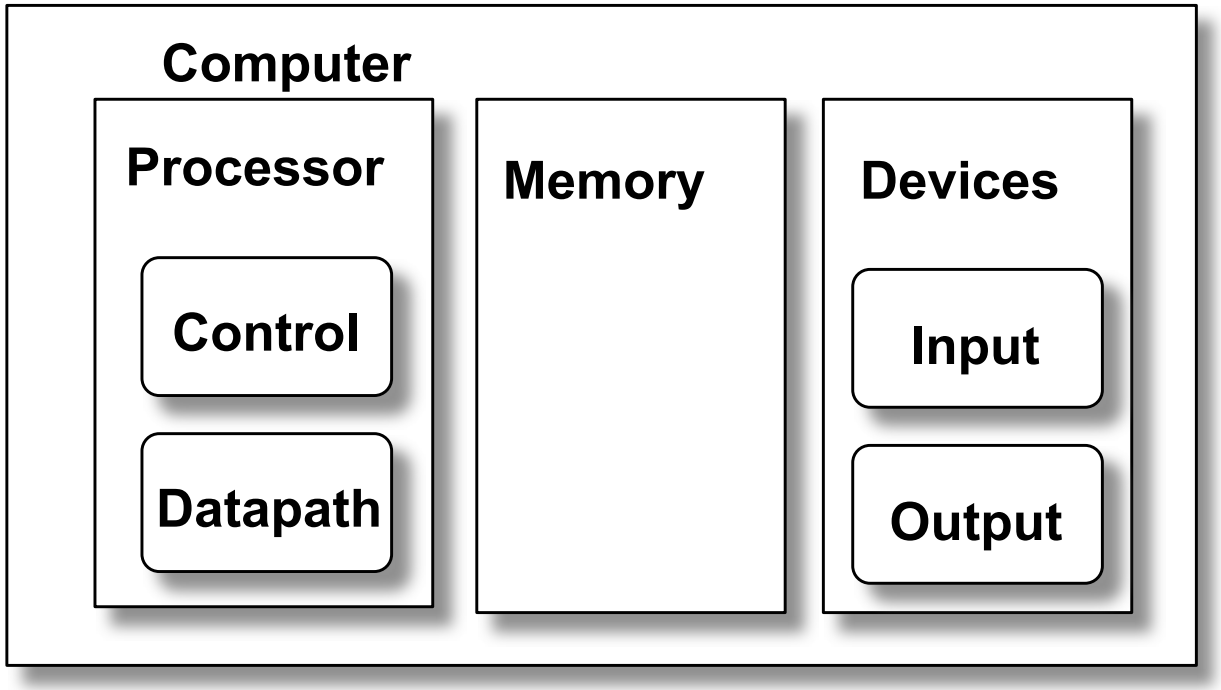
Datapath & Control

Readings: 4.1-4.4



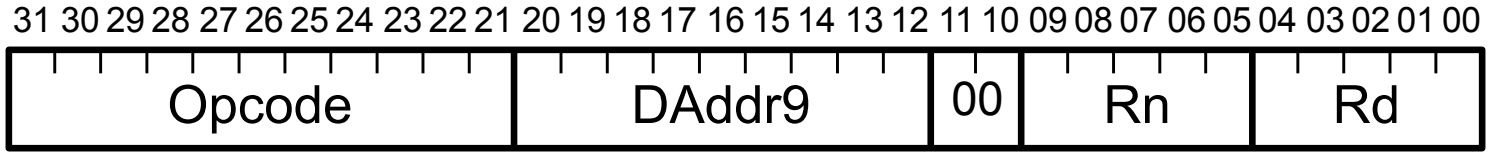
Datapath: System for performing operations on data, plus memory access.

Control: Control the datapath in response to instructions.

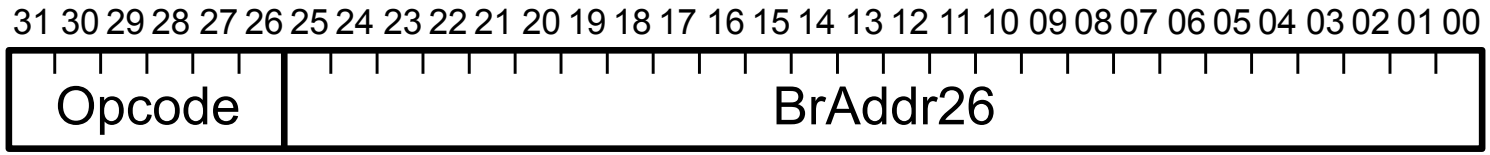
Simple CPU

Develop complete CPU for subset of instruction set

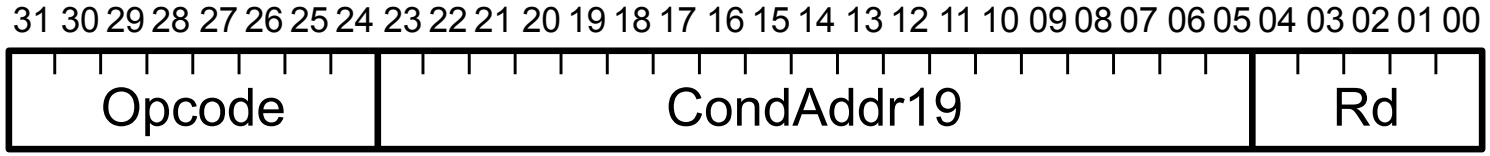
Memory: `LDUR, STUR`



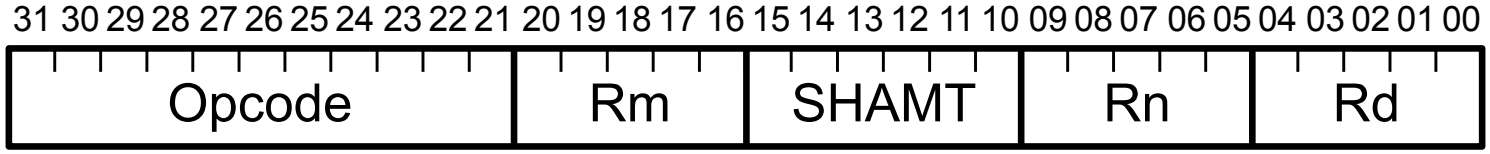
Branch: `B`



Conditional Branch: `CBZ`

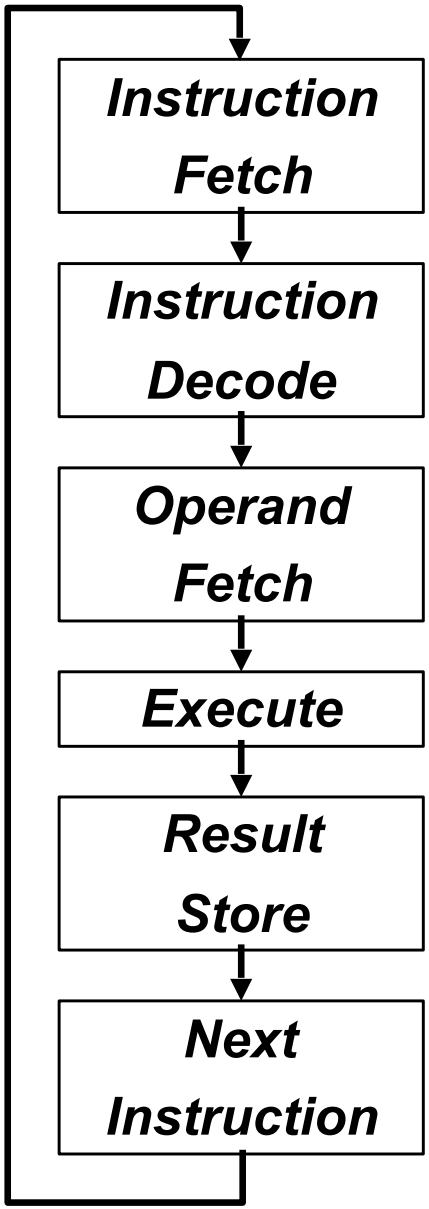


Arithmetic: `ADD, SUB`



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

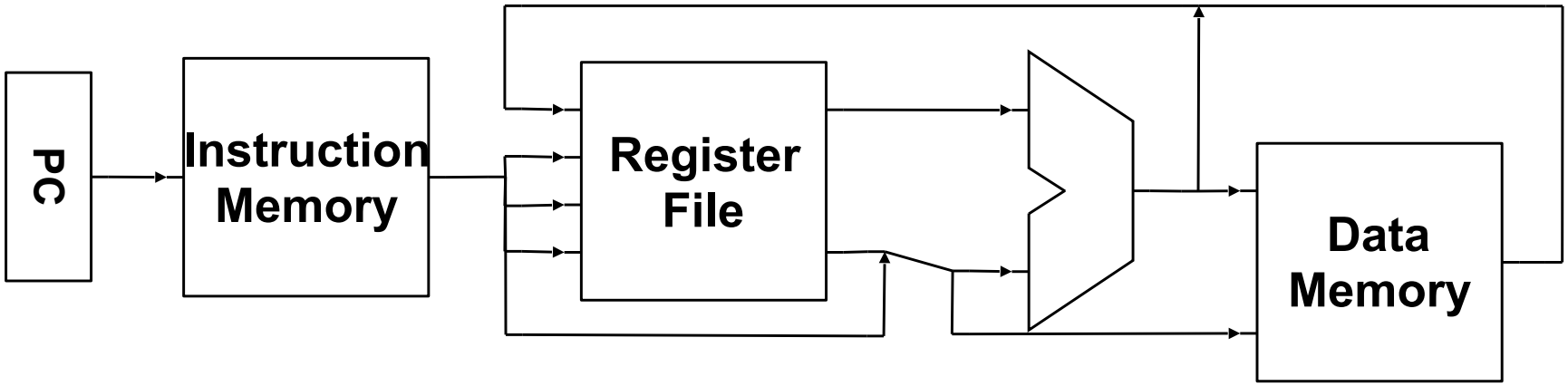
Deposit results in storage for later use

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



RTL & Processor Design

Convert instructions to Register Transfer Level (RTL) specification

```
RegA = RegB + RegC;
```

RTL specifies required interconnection of units, control

Math unit example:

(add): A = A + B; I++;

(mult): A = A * B: I++;

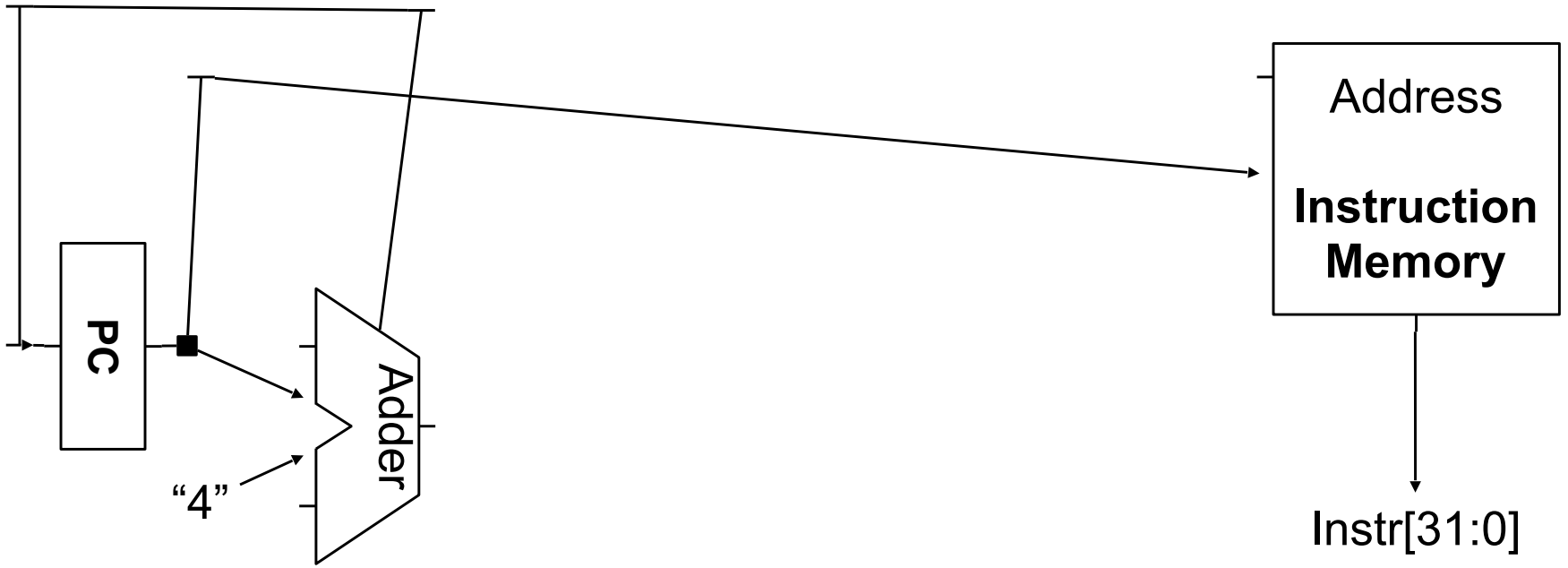
(hold): A = A; I++;

(init): A = Din; I++;

Instruction Fetch

Instruction Fetch

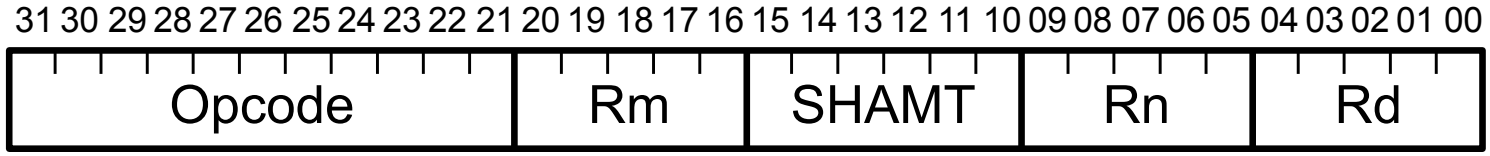
```
Instruction = Mem[PC];      // Fetch Instruction
PC = PC + 4;                // Increment
```



Add/Subtract RTL

Add instruction: `ADD Rd, Rn, Rm`

Subtract instruction: `SUB Rd, Rn, Rm`



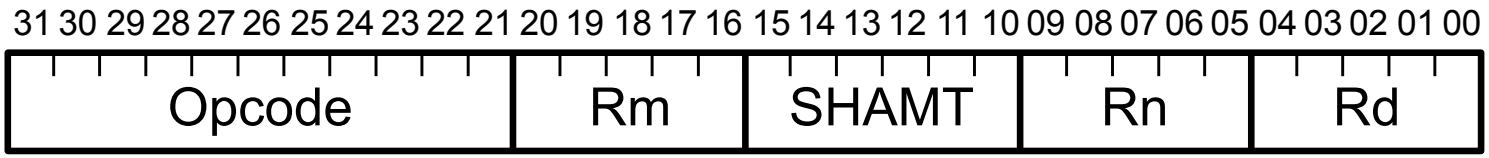
Add/Subtract RTL

Add instruction: `ADD Rd, Rn, Rm`

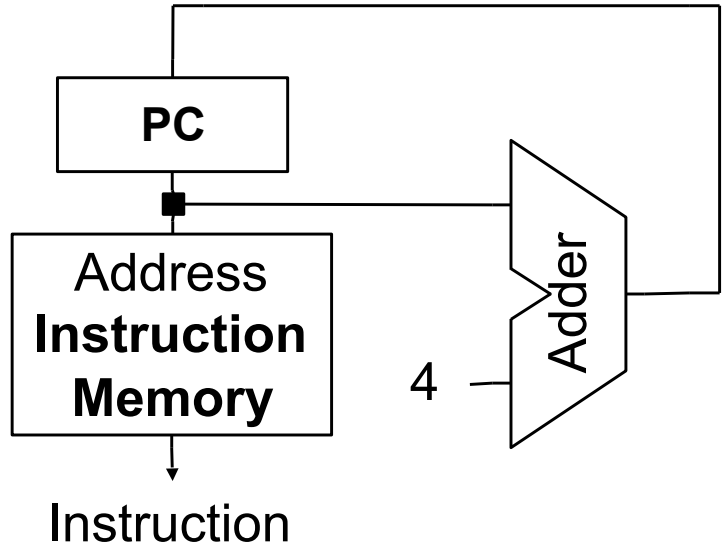
```
Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] + Reg[Rm];
PC = PC + 4;
```

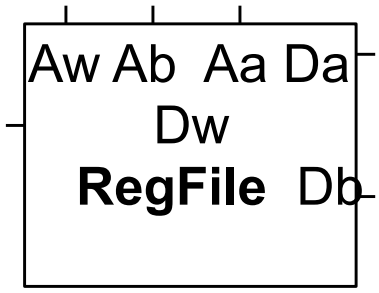
Subtract instruction: `SUB Rd, Rn, Rm`

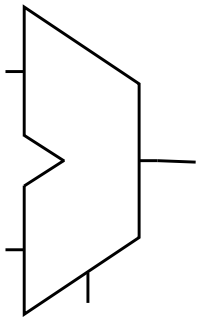
```
Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
```



Add/Subtract Datapath

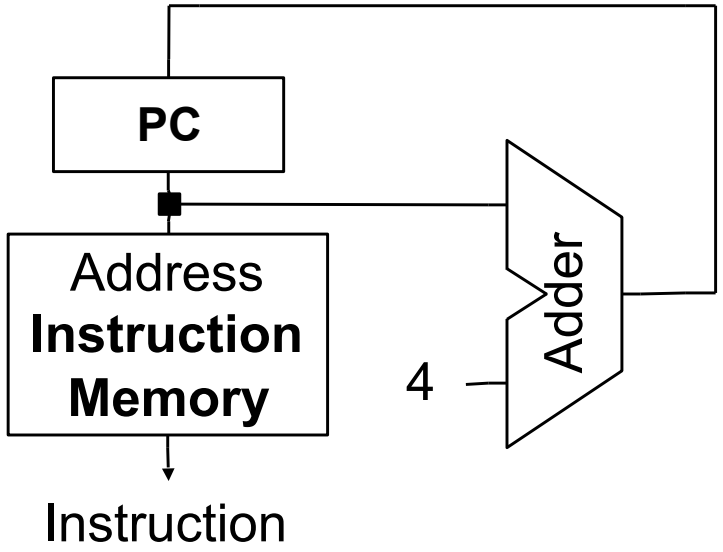


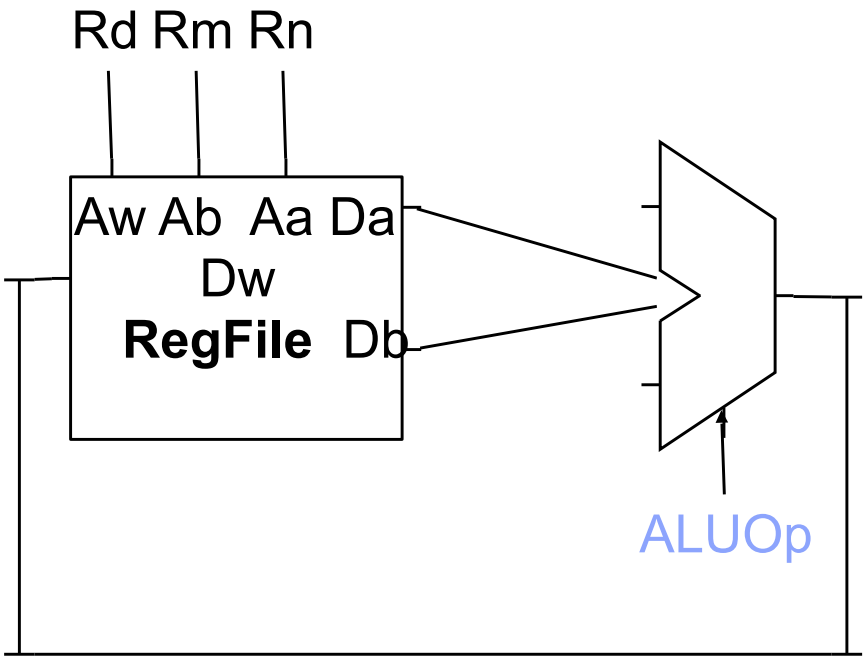




Add/Subtract Datapath

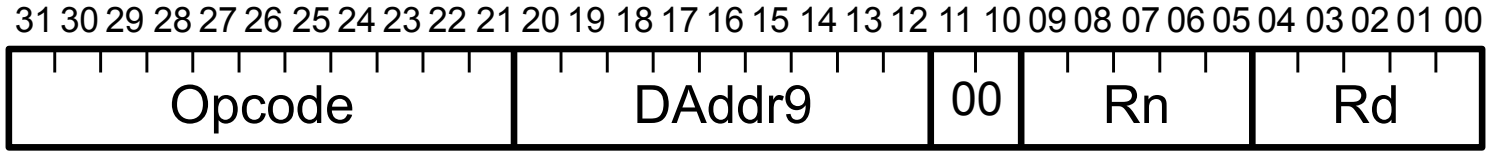
```
Reg[rd] = Reg[rn] op Reg[rm];
```





Load RTL

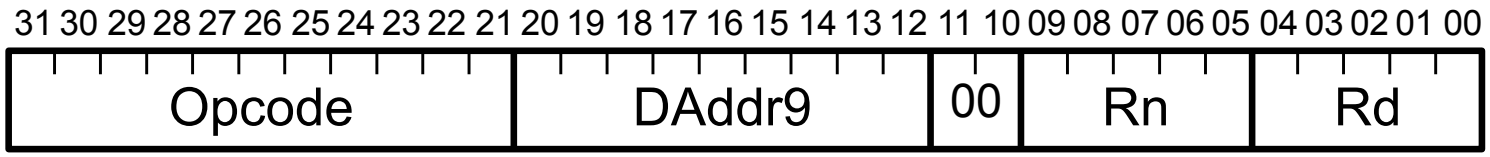
Load Instruction: `LDUR Rd, [Rn, DAddr9]`



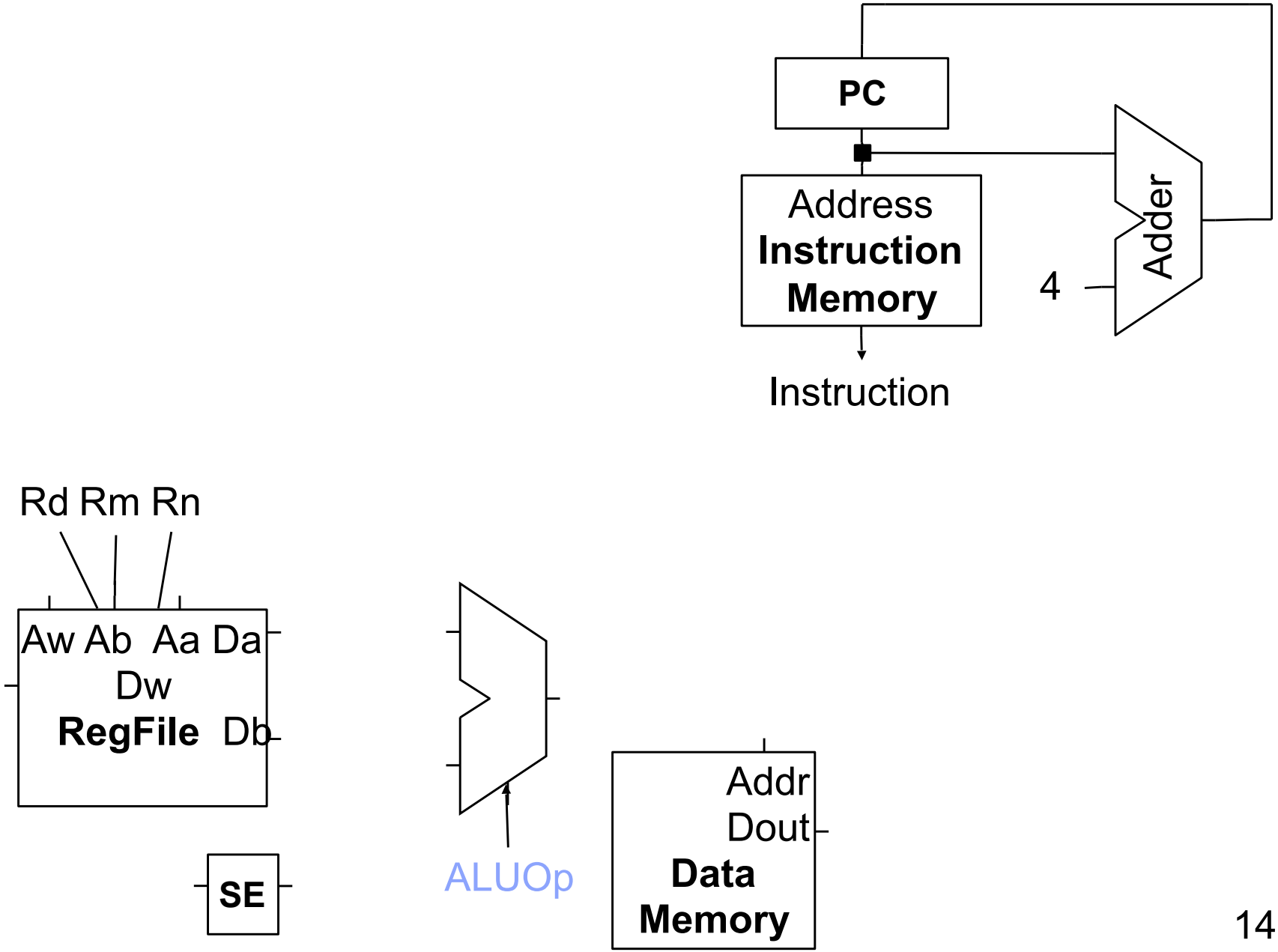
Load RTL

Load Instruction: `LDUR Rd, [Rn, DAddr9]`

```
Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Reg[rd] = Mem[Addr];
PC = PC + 4;
```

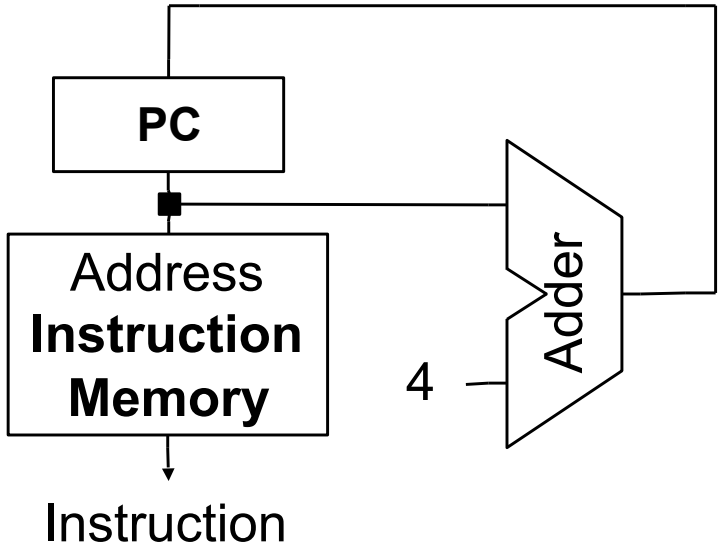


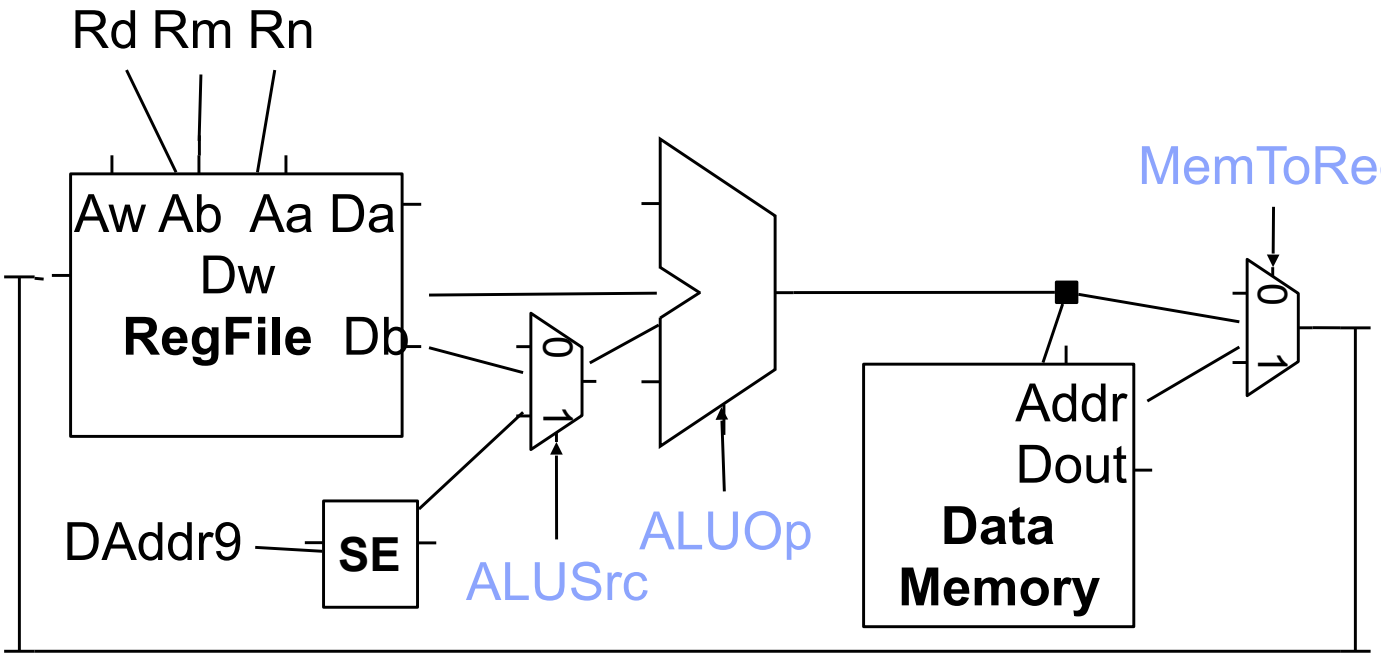
Datapath + Load



Datapath + Load

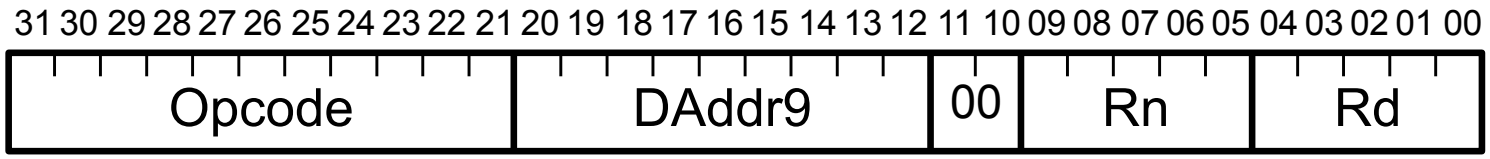
```
Addr = Reg[Rn] + SignExtend(DAddr9);
Reg[Rd] = Mem[Addr];
```





Store RTL

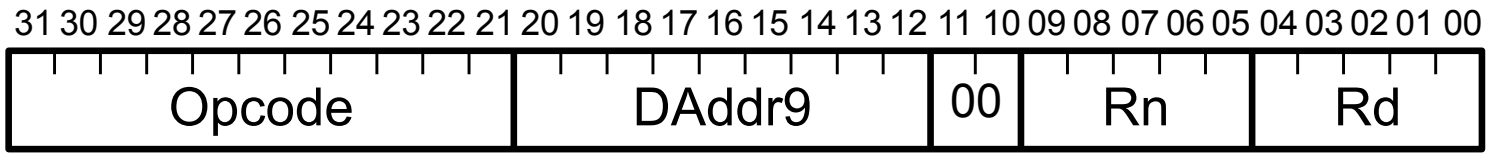
Store Instruction: `STUR Rd, [Rn, DAddr9]`



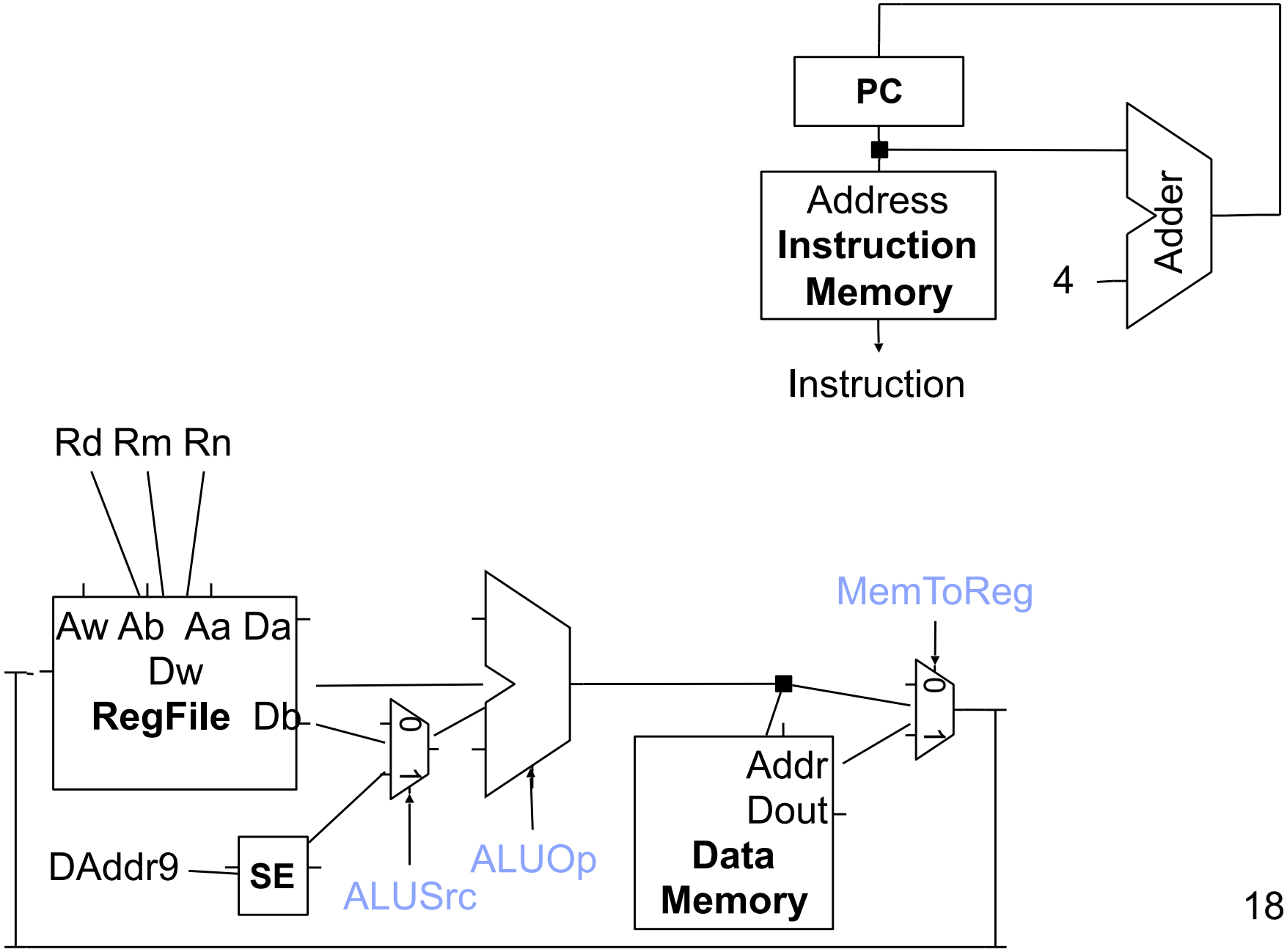
Store RTL

Store Instruction: `STUR Rd, [Rn, DAddr9]`

```
Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Mem[Addr] = Reg[Rd];
PC = PC + 4;
```

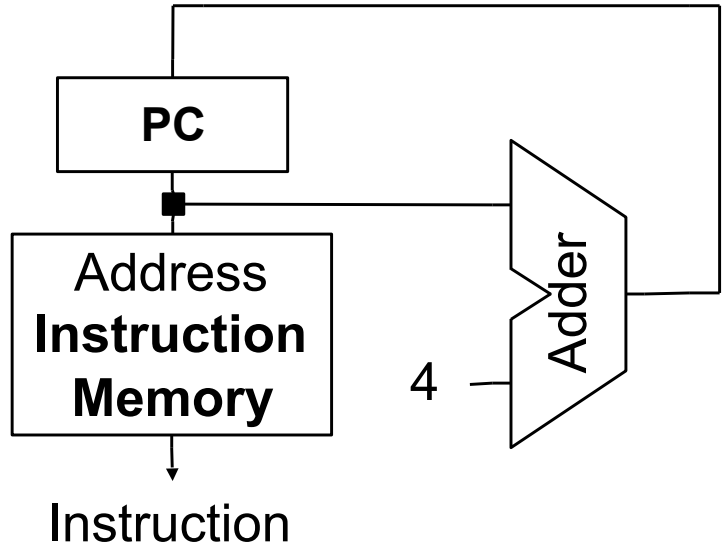


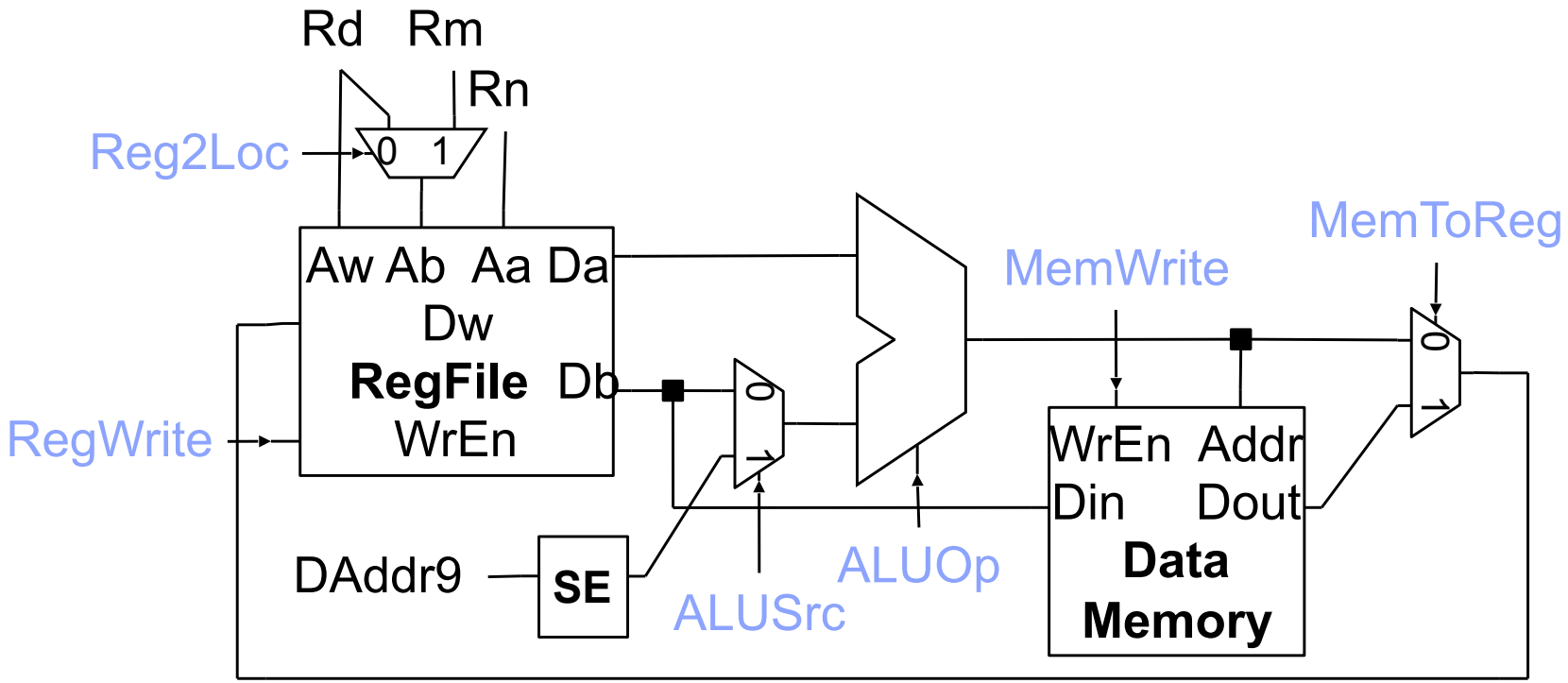
Datapath + Store



Datapath + Store

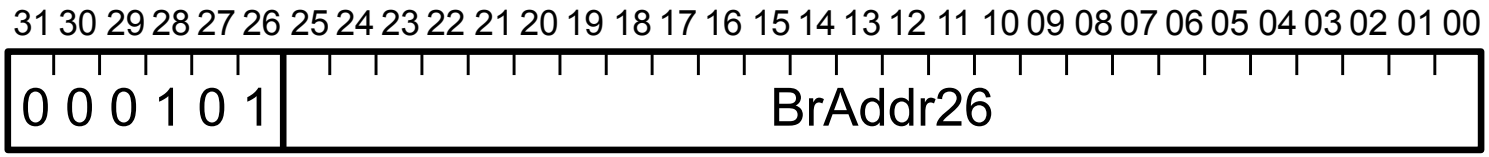
```
Addr = Reg[Rn] + SignExtend(DAddr9);
Mem[Addr] = Reg[Rd];
```





Branch RTL

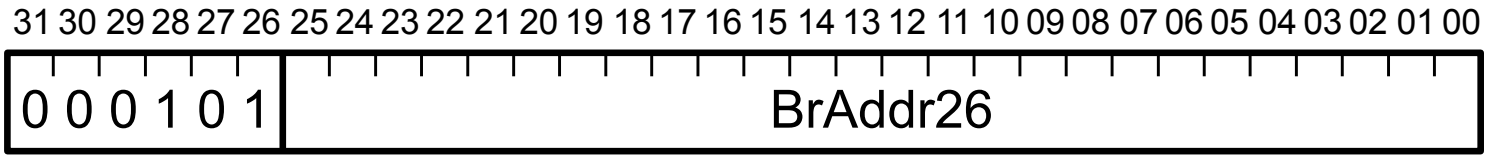
Branch Instruction: `B BrAddr26`



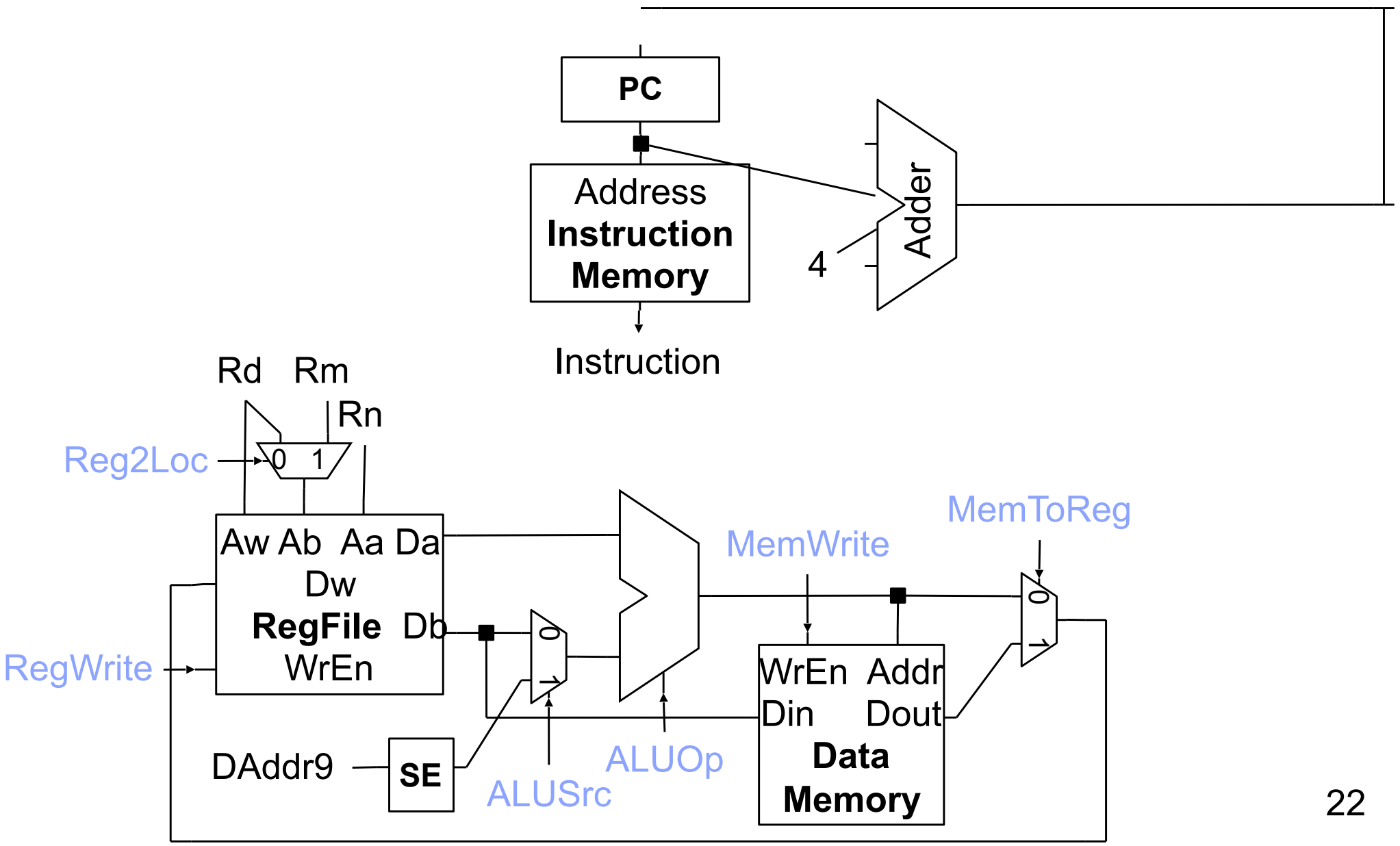
Branch RTL

Branch Instruction: `B BrAddr26`

```
Instruction = Mem[PC];
PC = PC + SignExtend(BrAddr26)<<2;
```

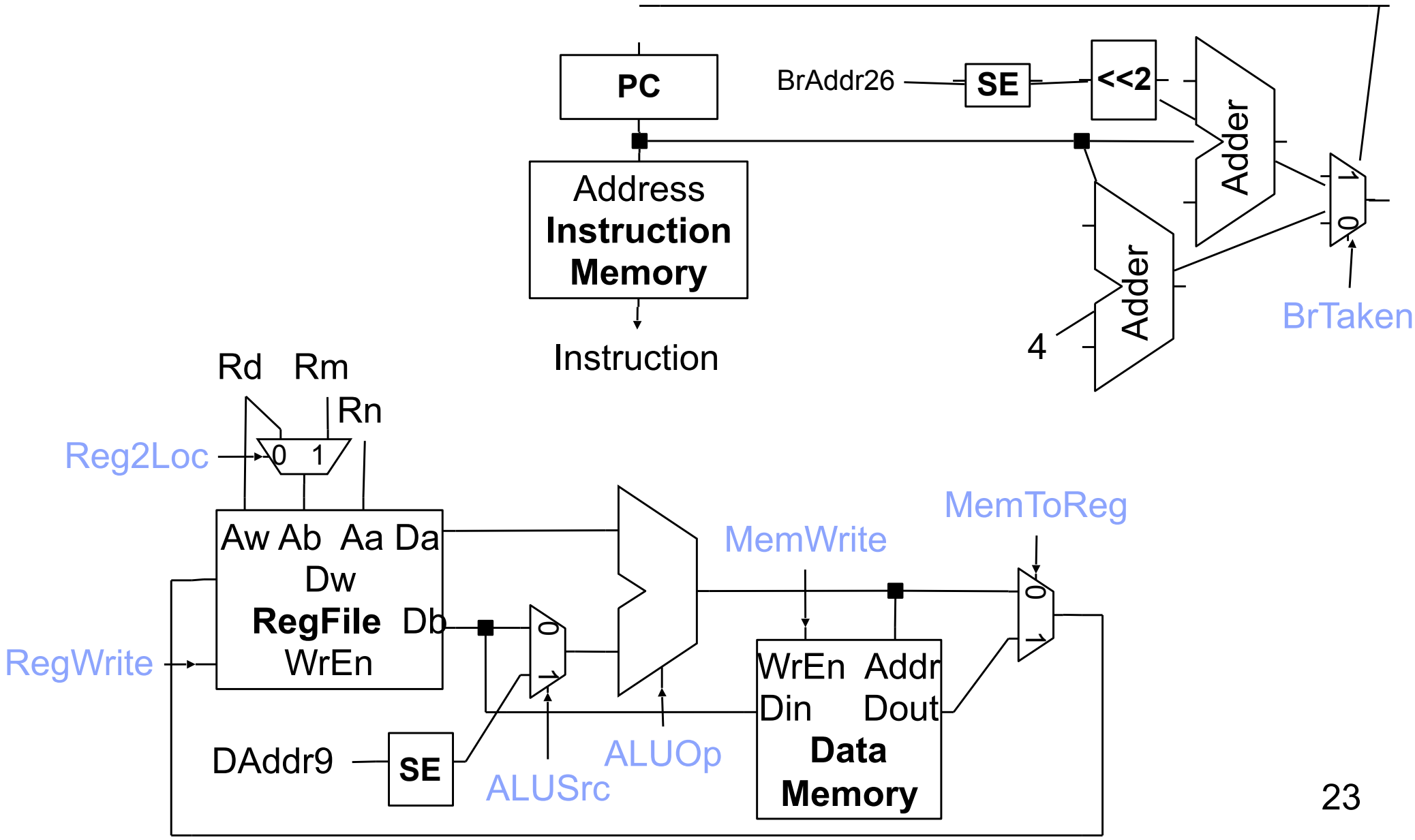


Datapath + Branch



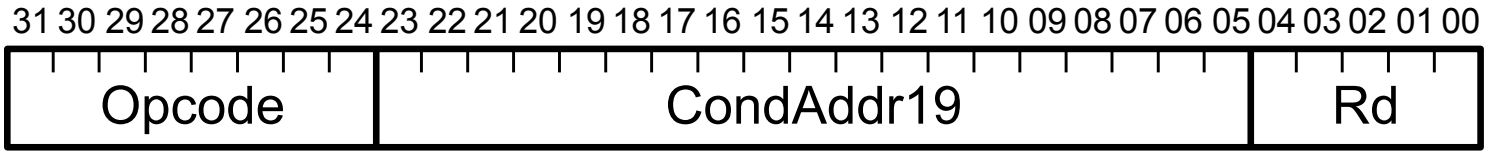
Datapath + Branch

```
PC = PC + SignExtend(BrAddr26)<<2;
```



Conditional Branch RTL

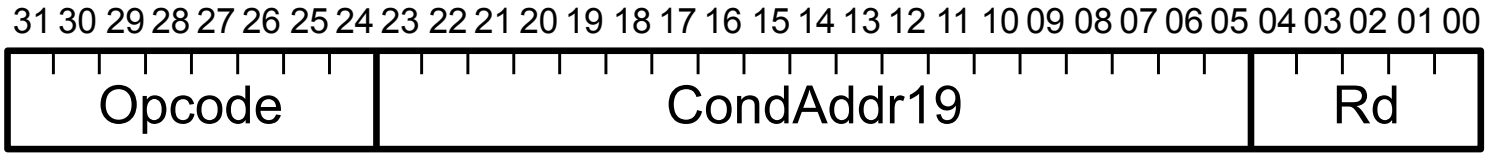
Conditional Branch Instruction: `CBZ Rd, CondAddr19`



Conditional Branch RTL

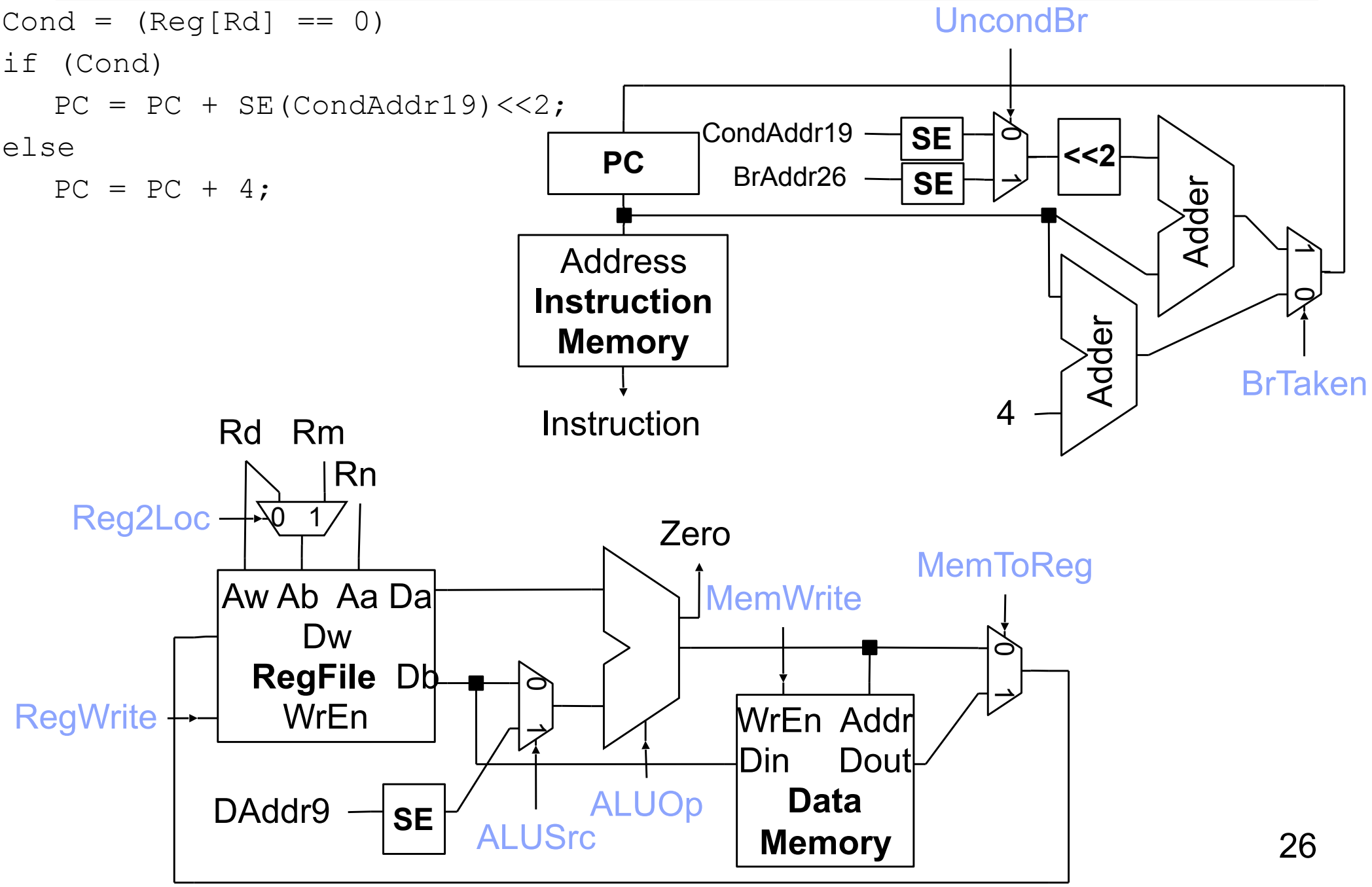
Conditional Branch Instruction: `CBZ Rd, CondAddr19`

```
Instruction = Mem[PC];
Cond = (Reg[Rd] == 0);
if (Cond)
    PC = PC + SignExtend(CondAddr19)<<2;
else
    PC = PC + 4;
```

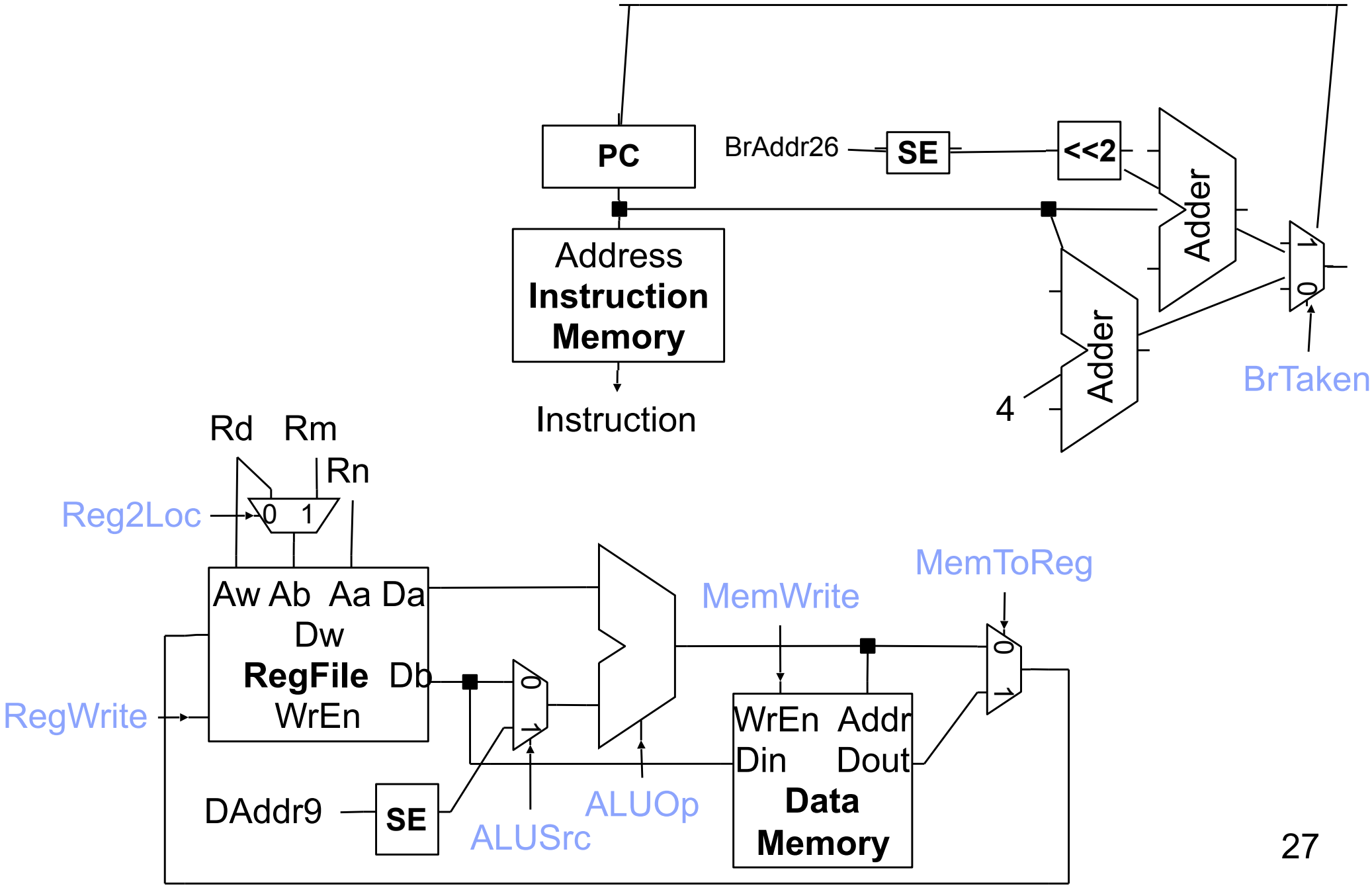


Datapath + Conditional Branch

```
Cond = (Reg[Rd] == 0)
if (Cond)
   PC = PC + SE(CondAddr19)<<2;
else
   PC = PC + 4;
```



Datapath + Conditional Branch

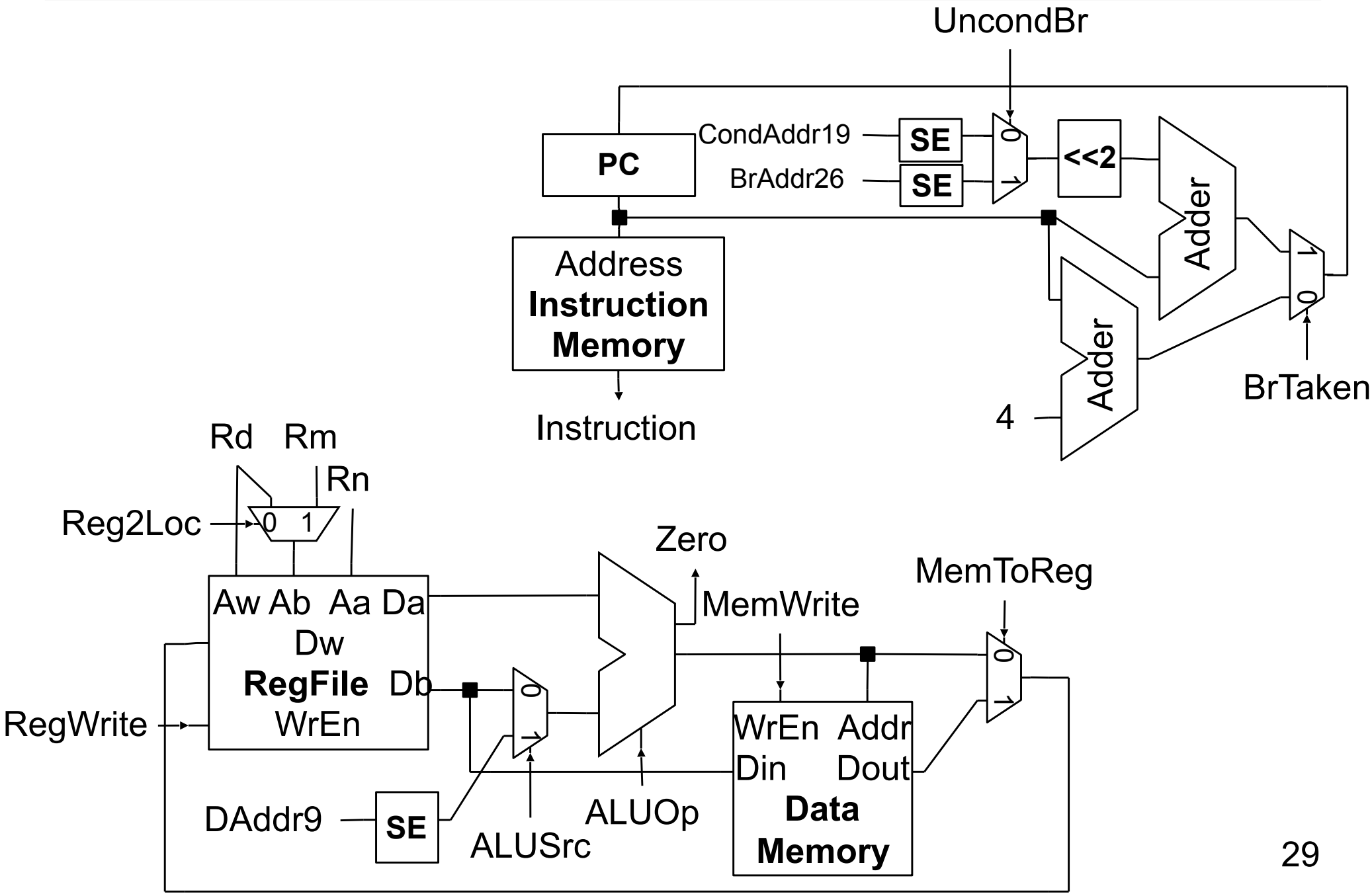


Control

Identify control points for pieces of datapath
- Instruction Fetch Unit
- ALU
- Memories
- Datapath muxes
- Etc.

Use RTL for determine per-instruction control assignments

Complete Datapath



Control Signals

Opcode[31:26] Opcode[25:21]	1000101 1000	1100101 1000	111110 00010	111110 00000	000101x xxxx	1011010 0xxx
	ADD	SUB	LDUR	STUR	B	CBZ

Control Signals

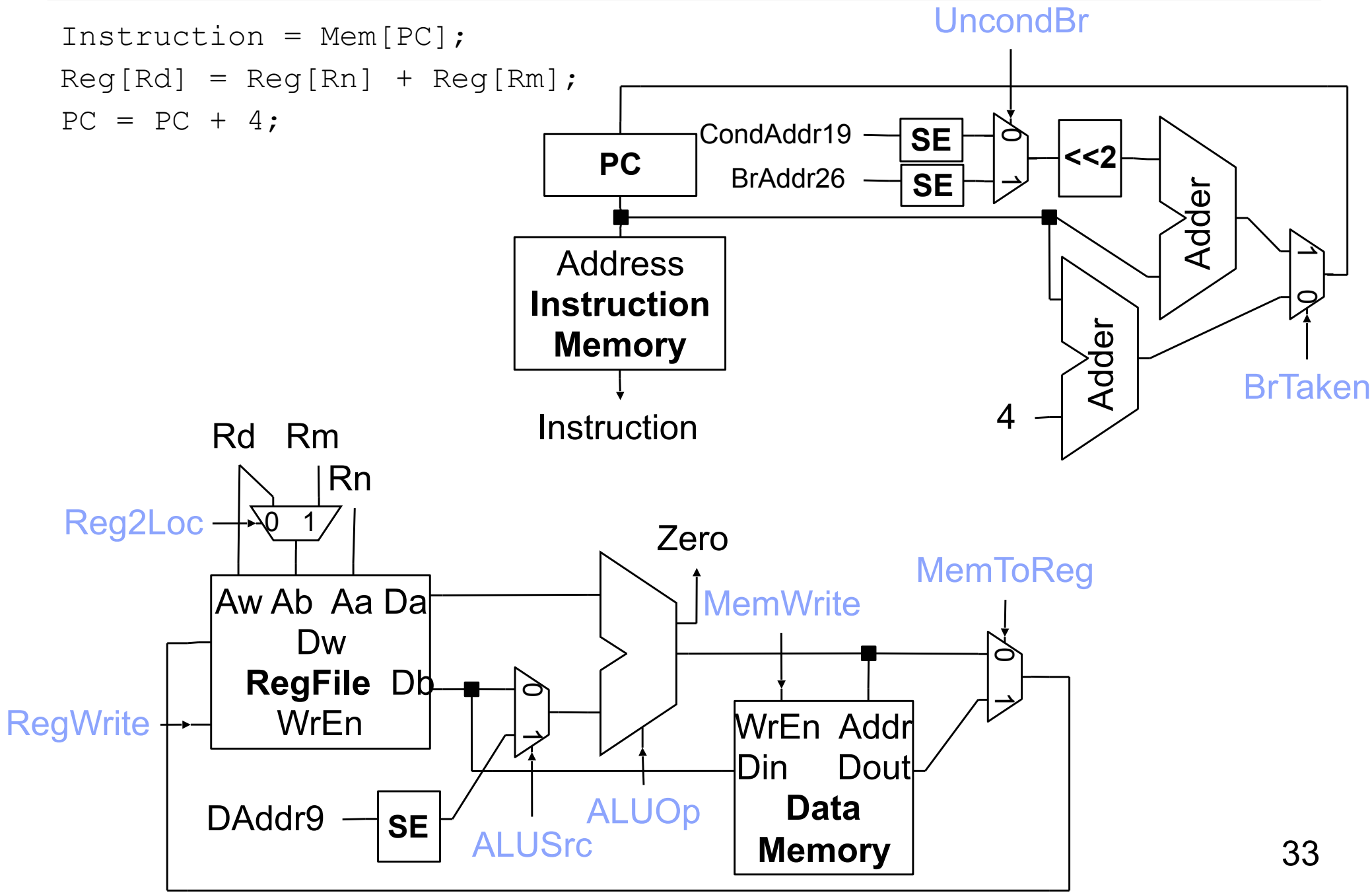
Opcode[31:26] Opcode[25:21]	1000101 1000	1100101 1000	111110 00010	111110 00000	000101x xxxx	1011010 0xxx
	ADD	SUB	LDUR	STUR	B	CBZ
Reg2Loc						
ALUSrc						
MemToReg						
RegWrite						
MemWrite						
BrTaken						
UncondBr						
ALUOp						

Control Signals

Opcode[31:26] Opcode[25:21]	1000101 1000	1100101 1000	111110 00010	111110 00000	000101x xxxx	1011010 0xxx
	ADD	SUB	LDUR	STUR	B	CBZ
Reg2Loc	1	1	X	0	X	0
ALUSrc	0	0	1	1	X	0
MemToReg	0	0	1	X	X	X
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
BrTaken	0	0	0	0	1	(zero)
UncondBr	X	X	X	X	1	0
ALUOp	+	-	+	+	X	B==0?

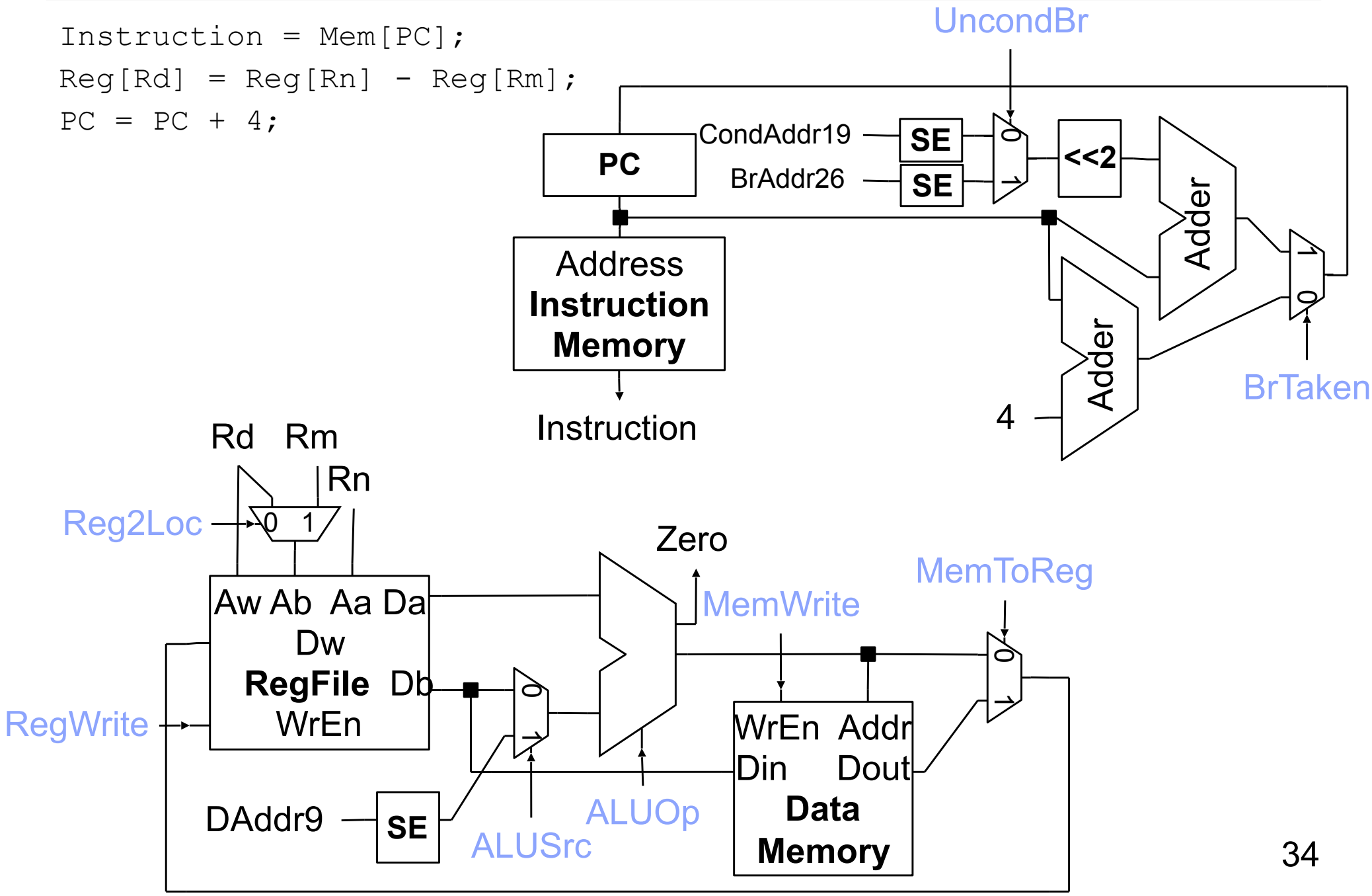
ADD Control

```
Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] + Reg[Rm];
PC = PC + 4;
```



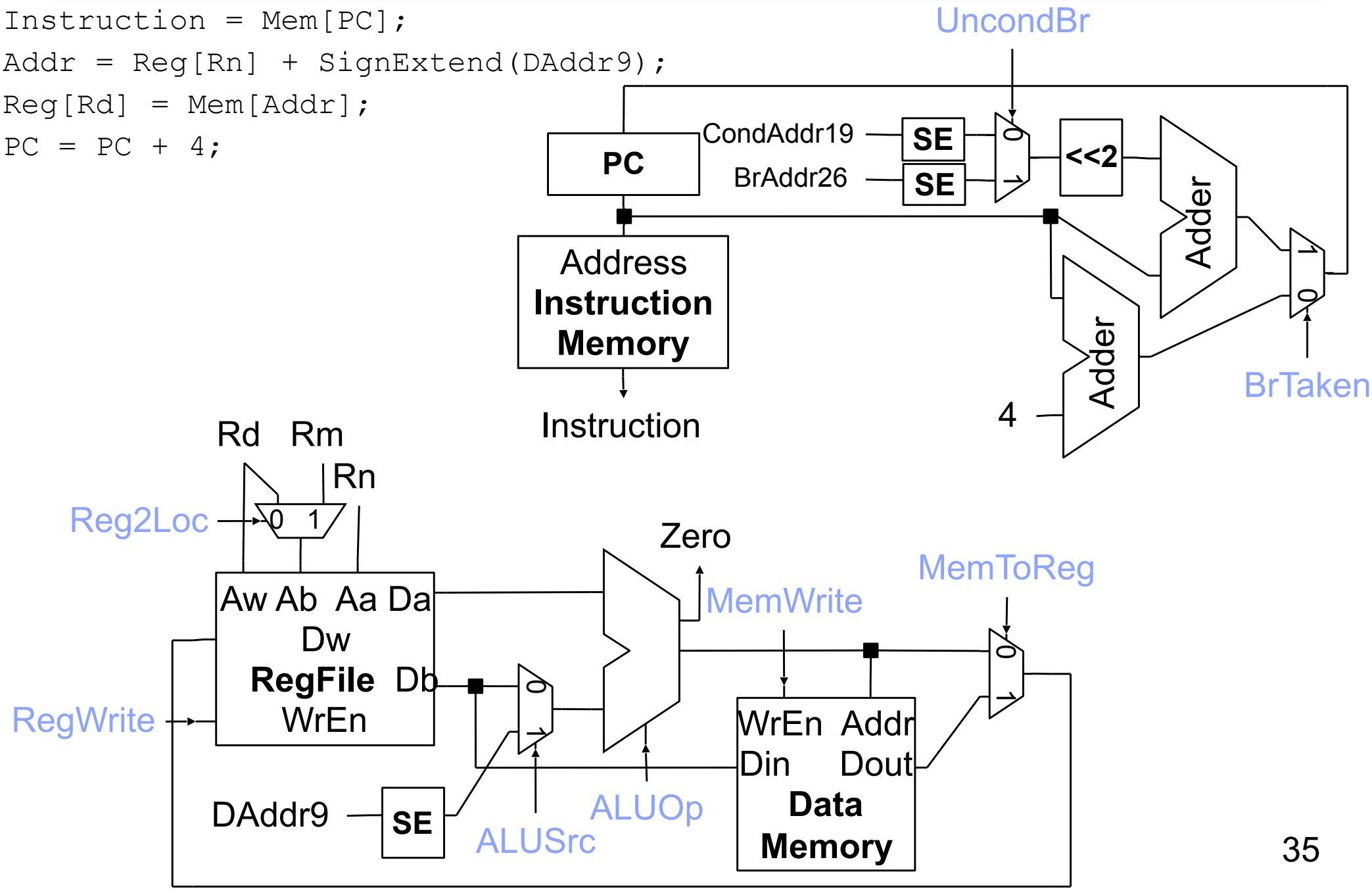
SUB Control

```
Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
```



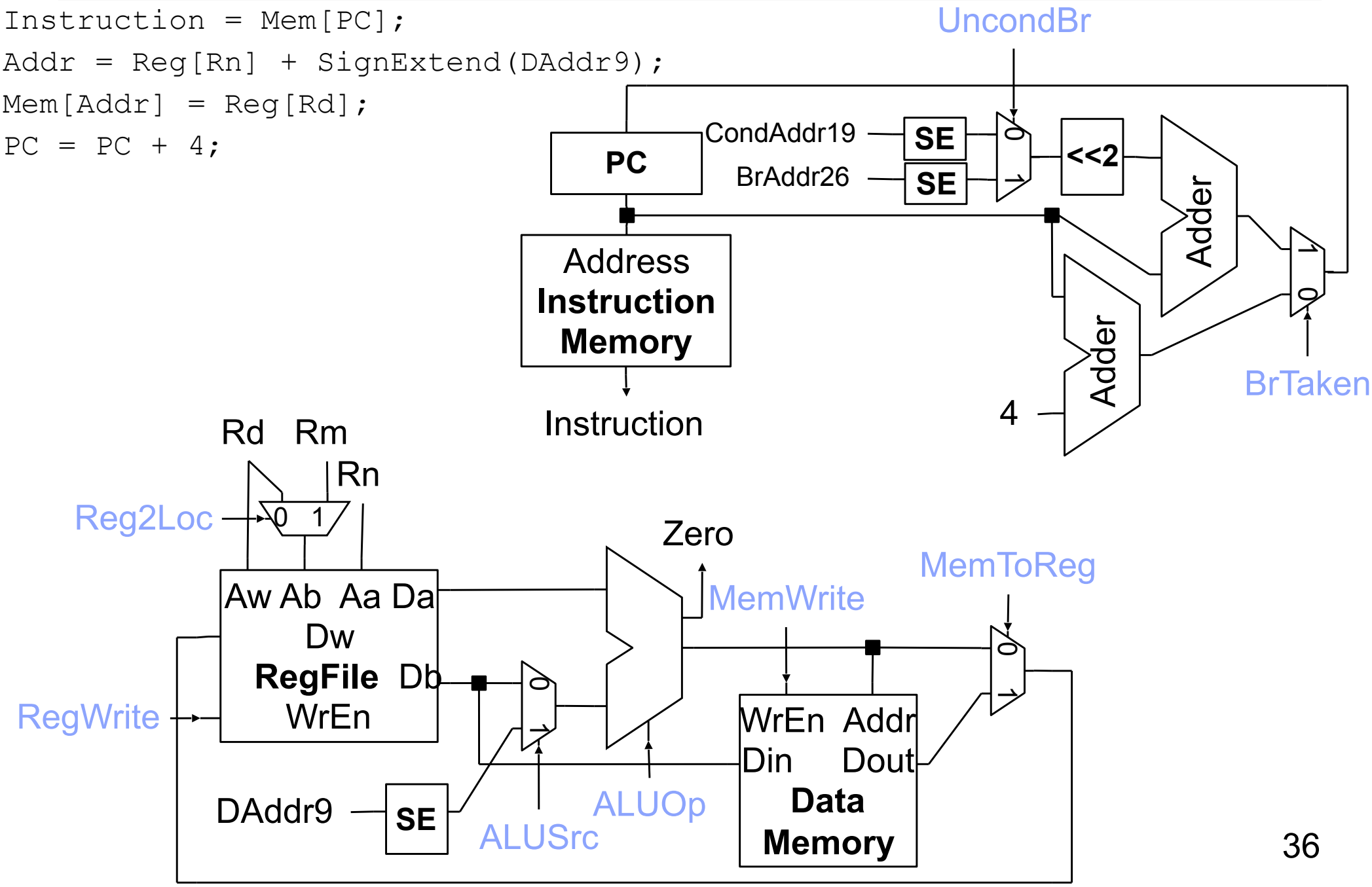
LDUR Control

```
Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Reg[Rd] = Mem[Addr];
PC = PC + 4;
```



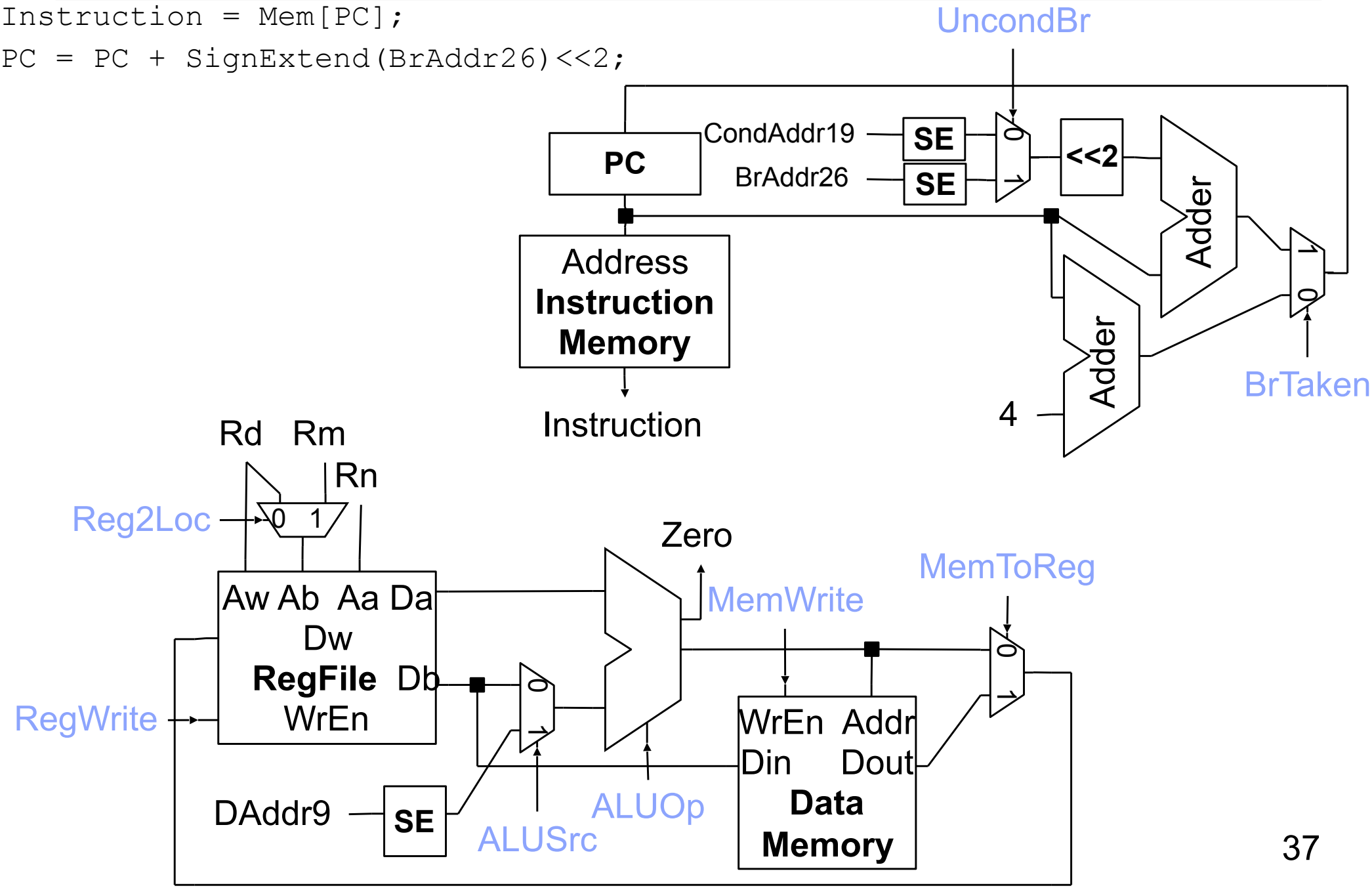
STUR Control

```
Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Mem[Addr] = Reg[Rd];
PC = PC + 4;
```



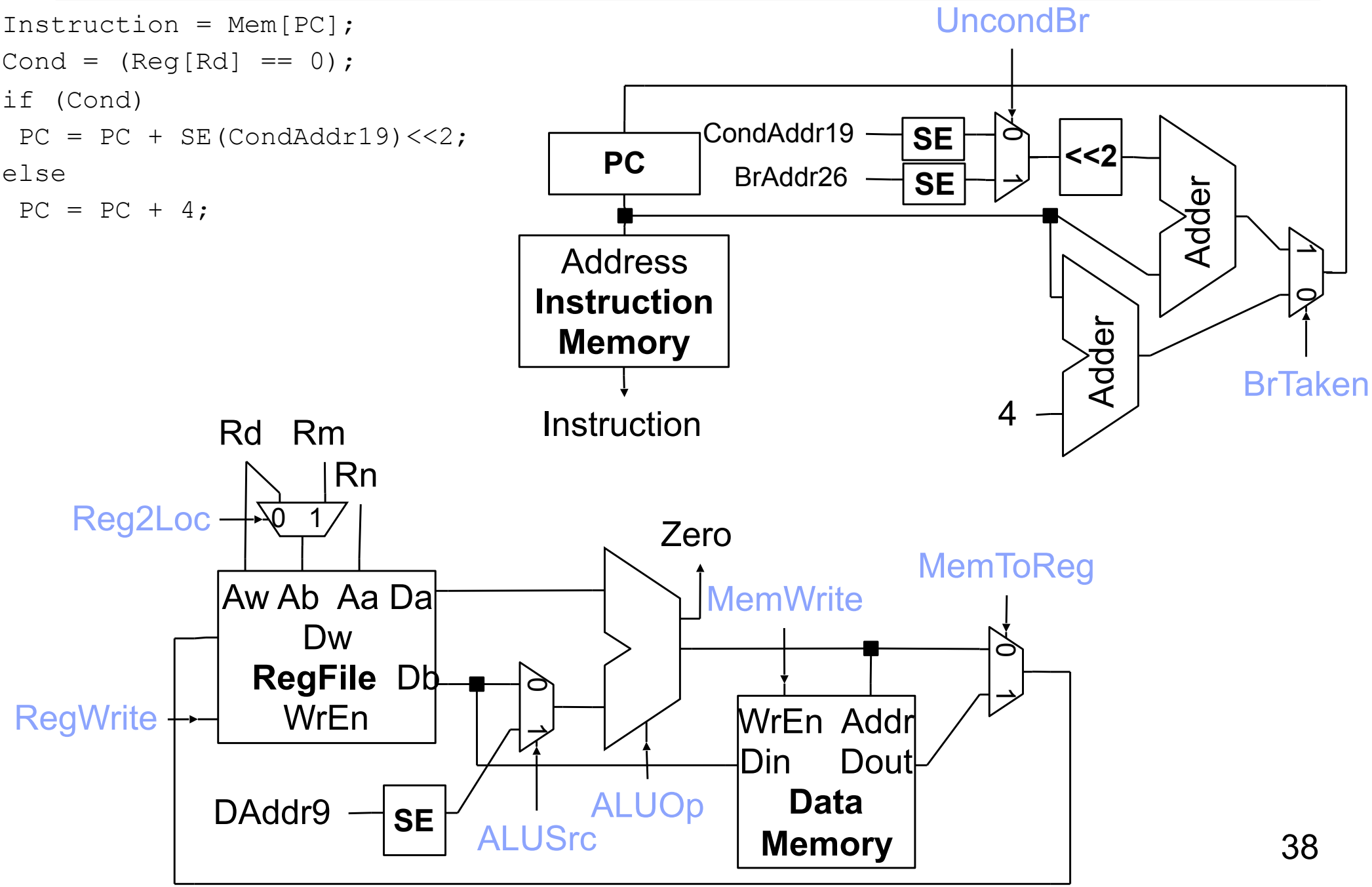
B Control

```
Instruction = Mem[PC];
PC = PC + SignExtend(BrAddr26)<<2;
```

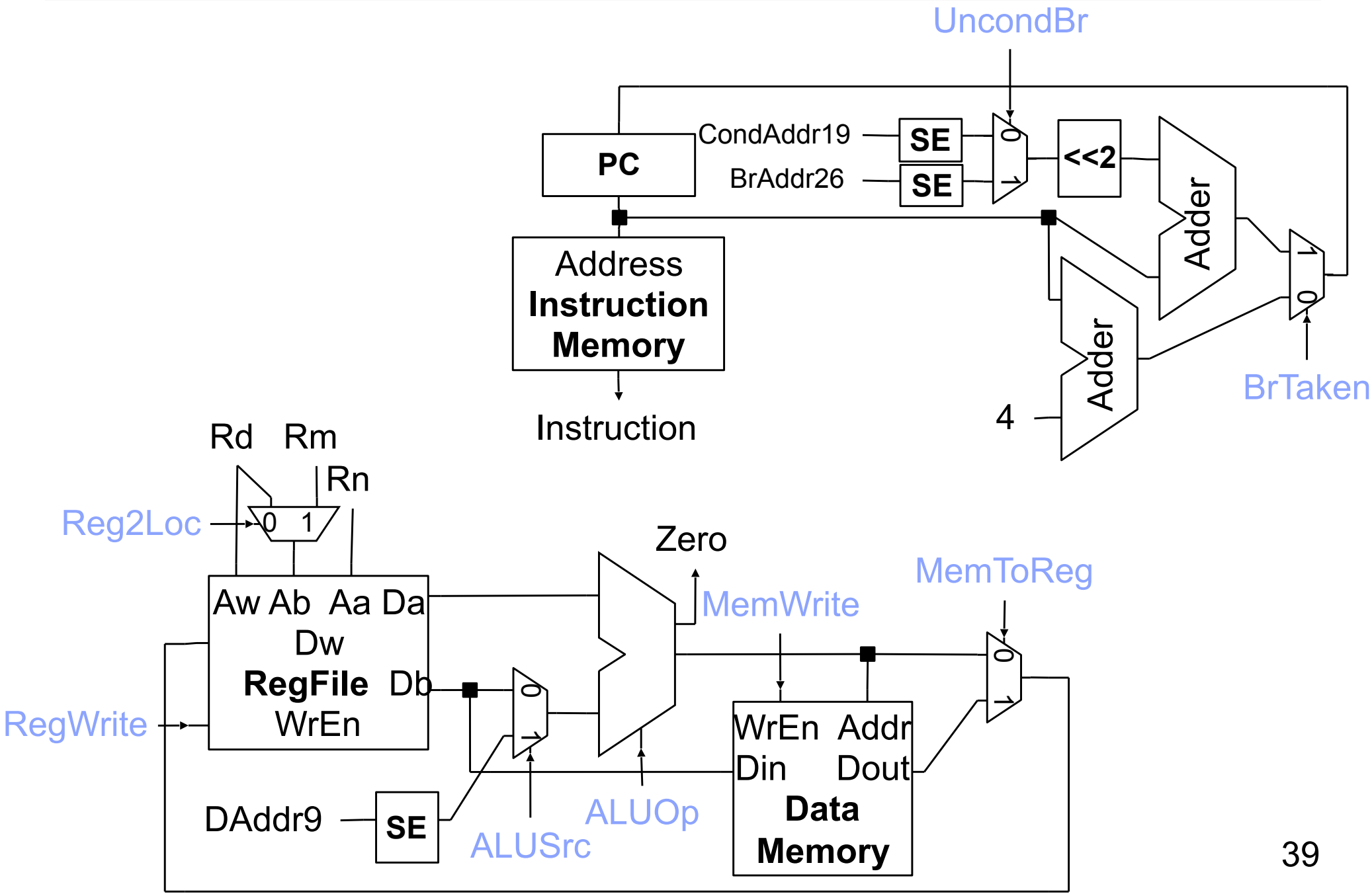


CBZ Control

```
Instruction = Mem[PC];
Cond = (Reg[Rd] == 0);
if (Cond)
 PC = PC + SE(CondAddr19)<<2;
else
 PC = PC + 4;
```

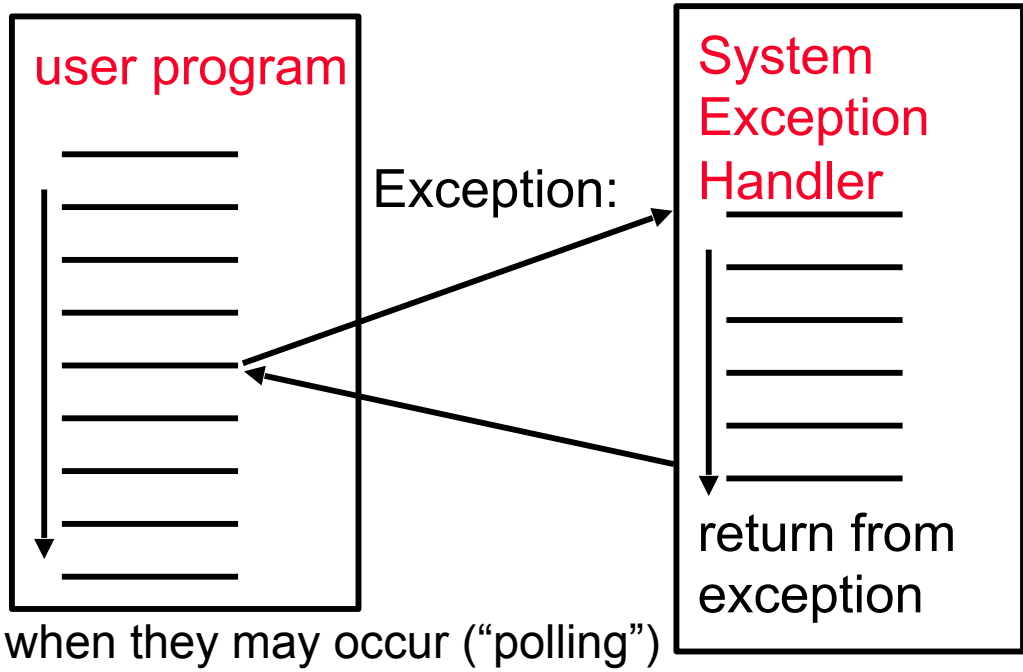


Complete Datapath



Advanced: Exceptions

Exception = unusual event in processor
- Arithmetic overflow, divide by zero, …
- Call an undefined instruction
- Hardware failure
- I/O device request (called an “interrupt”)



Approaches
- Make software test for exceptional events when they may occur (“polling”)
- Have hardware detect these events & react:
  - Save state (Exception Program Counter, protect the GPRs, note cause)
  - Call Operating System
    - If (undef_instr) PC = C0000000
    - If (overflow) PC = C0000020
    - If (I/O) PC = C0000040
    - …

Performance of Single-Cycle Machine

CPI?

ADD, SUB

LDUR

STUR

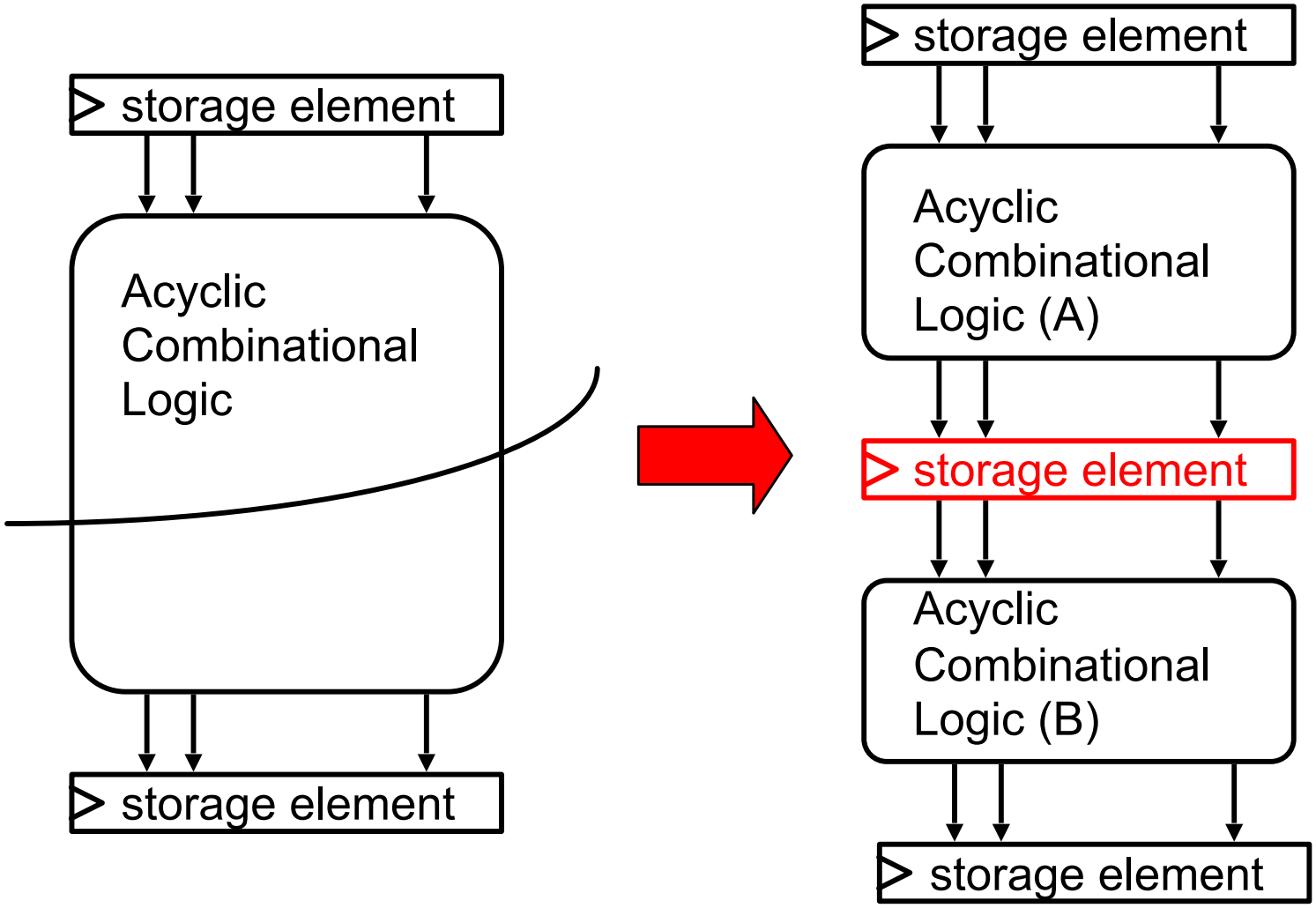
CBZ

B

Reducing Cycle Time

Cut combinational dependency graph and insert register / latch

Do same work in two fast cycles, rather than one slow one



Pipelined Processor Overview

Divide datapath into multiple stages