ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2023

Datapaths

Xin Li & Dawei Liu
Duke Kunshan University

Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

Last time

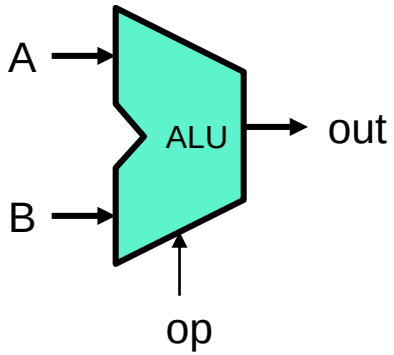
- What did we do last time?
  - MIPS Assembly

Now confluence of MIPS + digital logic

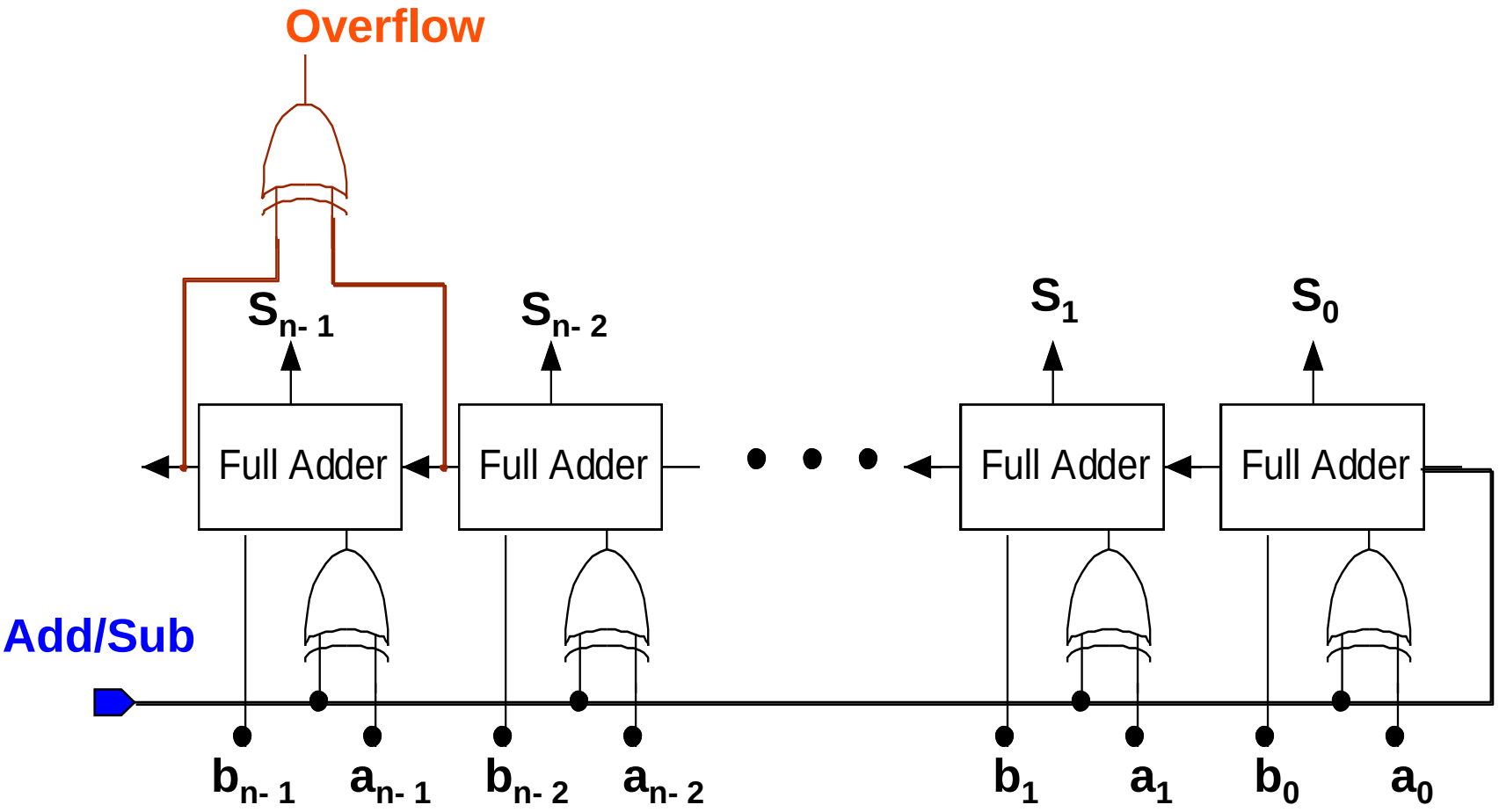
- Start of semester: Digital Logic
  - Building blocks of digital design
- Most recently: MIPS assembly, ISA
  - Lowest level software
- Now: where they meet…
  - Datapaths: hardware implementation of processors

Necessary ingredient: the ALU

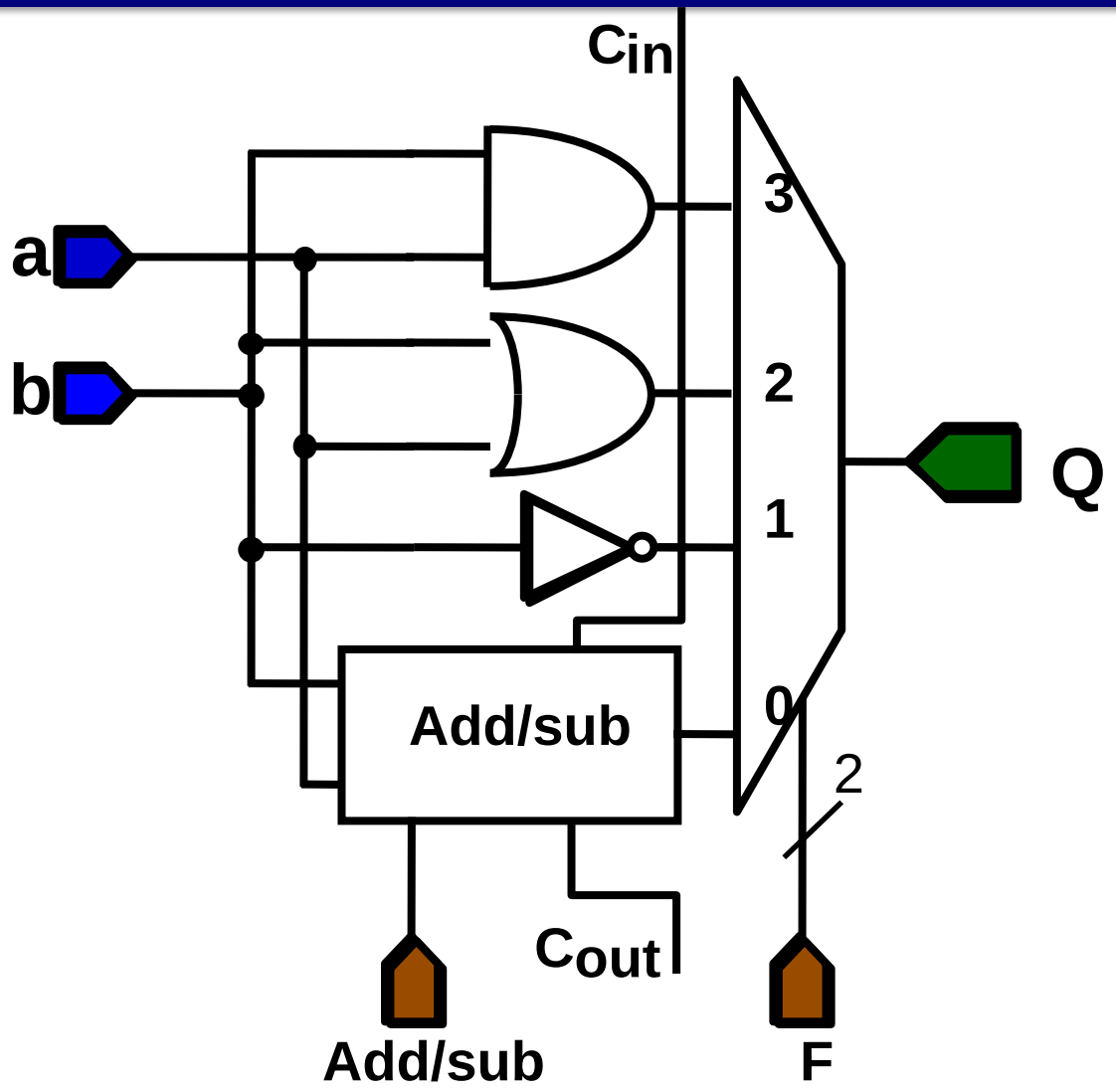
- **ALU: Arithmetic/Logic Unit**
  - Performs any supported math or logic operation on two inputs
  - Which operation is chosen by a third input



Add/Subtract With Overflow Detection

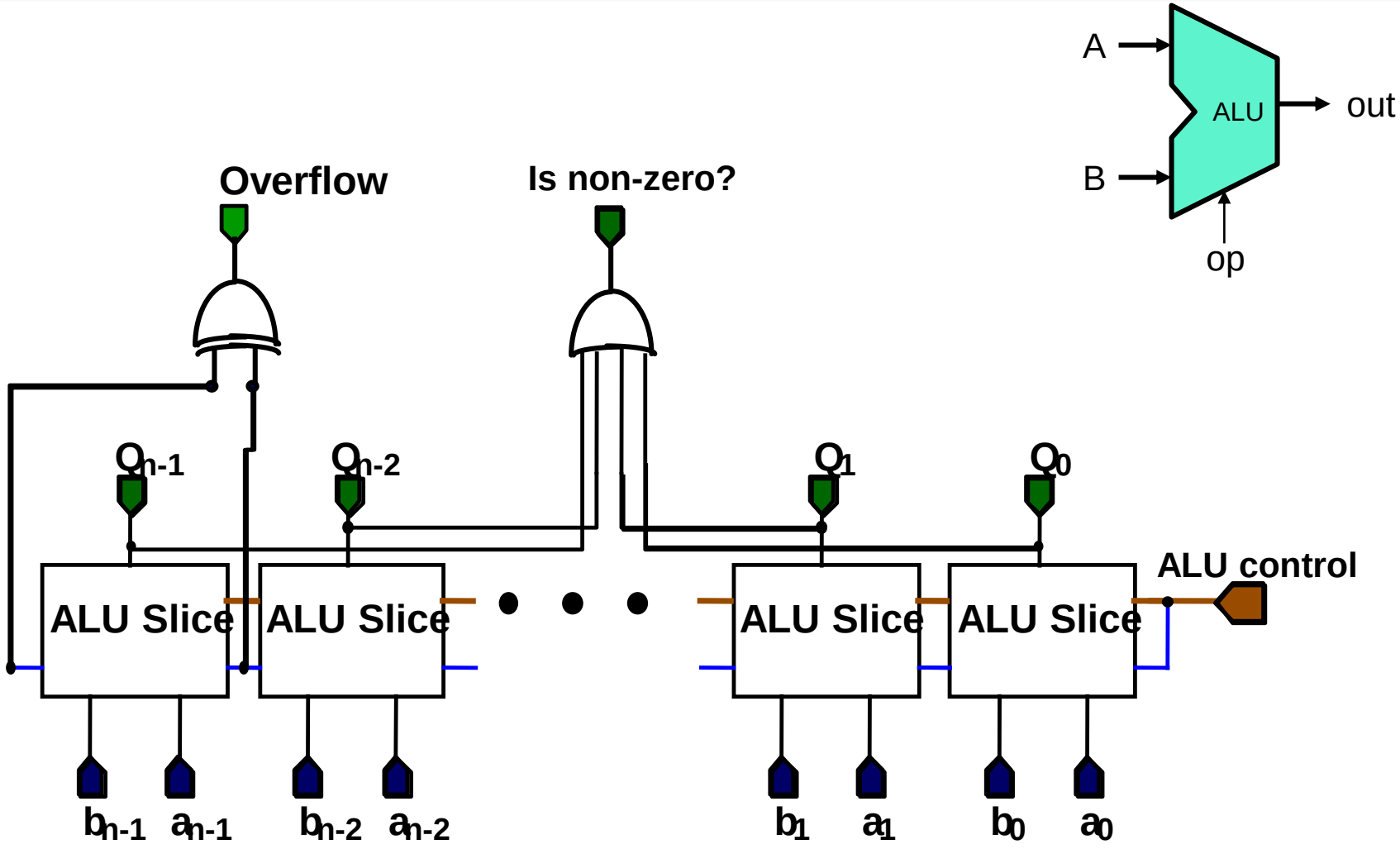


ALU Slice



A	F	Q
0	0	a + b
1	0	a - b
-	1	NOT b
-	2	a OR b
-	3	a AND b

The ALU


A
B
ALU
out
op
Overflow
Is non-zero?
O_{n-1}
O_{n-2}
O_1
O_0
ALU control
ALU Slice
ALU Slice
ALU Slice
ALU Slice
b_{n-1} a_{n-1}
b_{n-2} a_{n-2}
b_1 a_1
b_0 a_0

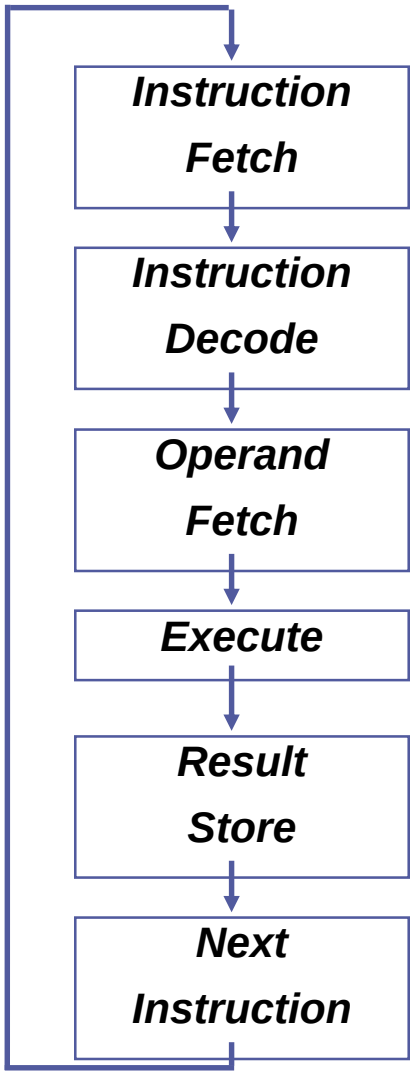
Datapath for MIPS ISA

- Consider only the following instructions

```
add $1,$2,$3
addi $1,2,$3
lw $1,4($3)
sw $1,4($3)
beq $1,$2,PC_relative_target
j absolute_target
```

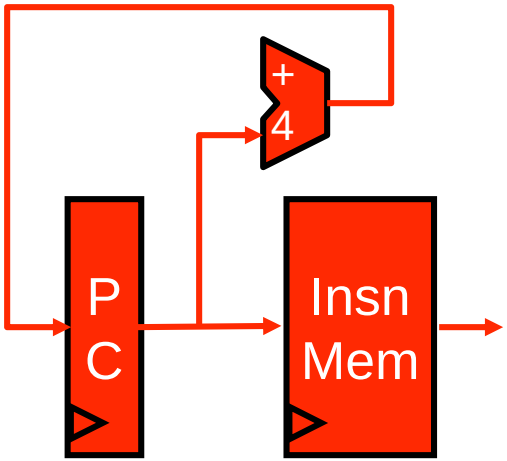
- Why only these?
  - Most other instructions are the same from datapath viewpoint
  - The one's that aren't are left for you to figure out

Remember The von Neumann Model?



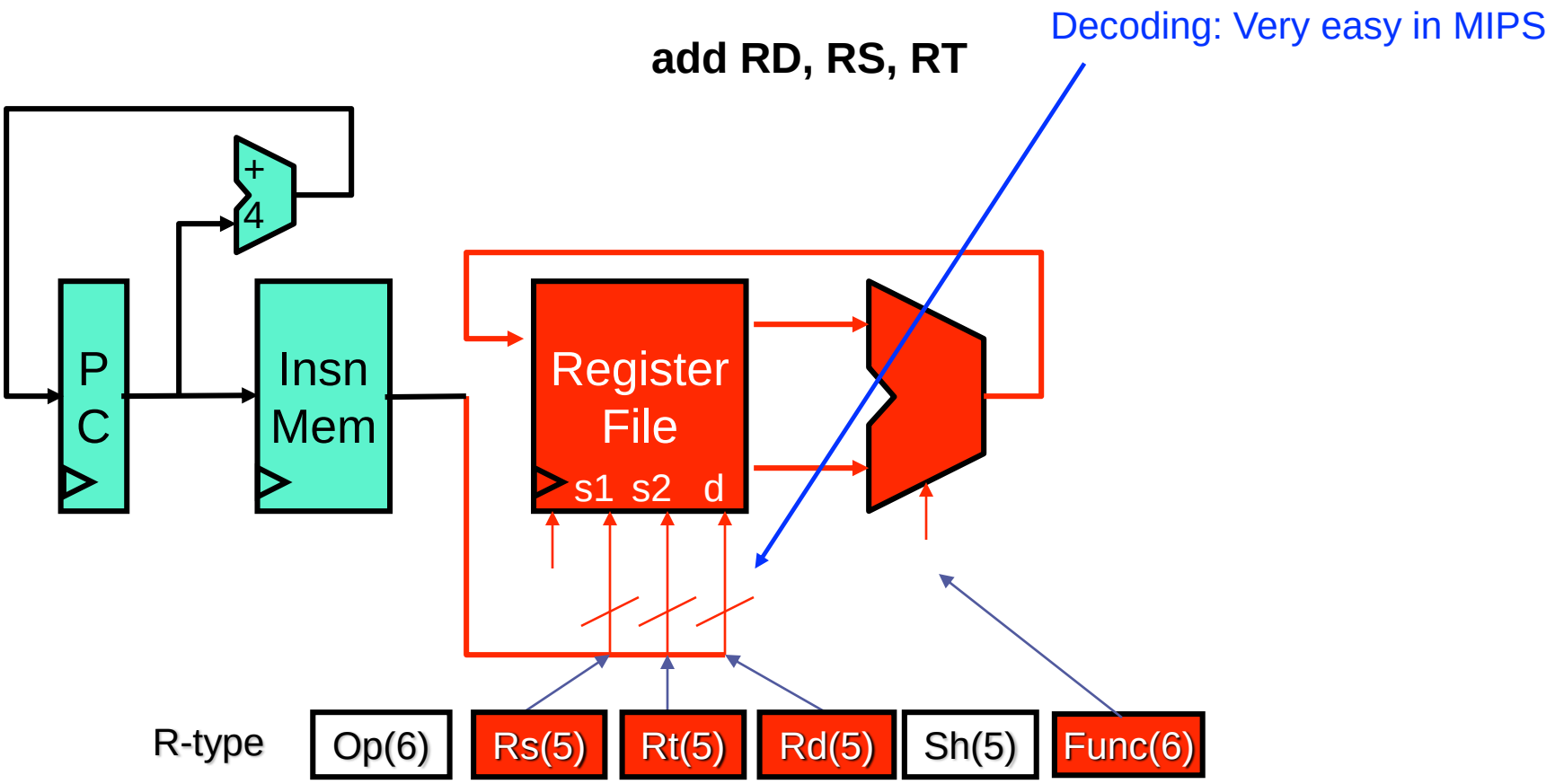
- Instruction Fetch:
  Read instruction bits from memory
- Decode:
  Figure out what those bits mean
- Operand Fetch:
  Read registers (+ mem to get sources)
- Execute:
  Do the actual operation (e.g., add the #s)
- Result Store:
  Write result to register or memory
- Next Instruction:
  Figure out mem addr of next insn, repeat

Start With Fetch



- Same for all instructions (don't know insn yet)
- PC and instruction memory
- A +4 incrementer computes default next instruction PC
  - Details of Insn Mem: later…
  - For now: just assume a bunch of DFFs

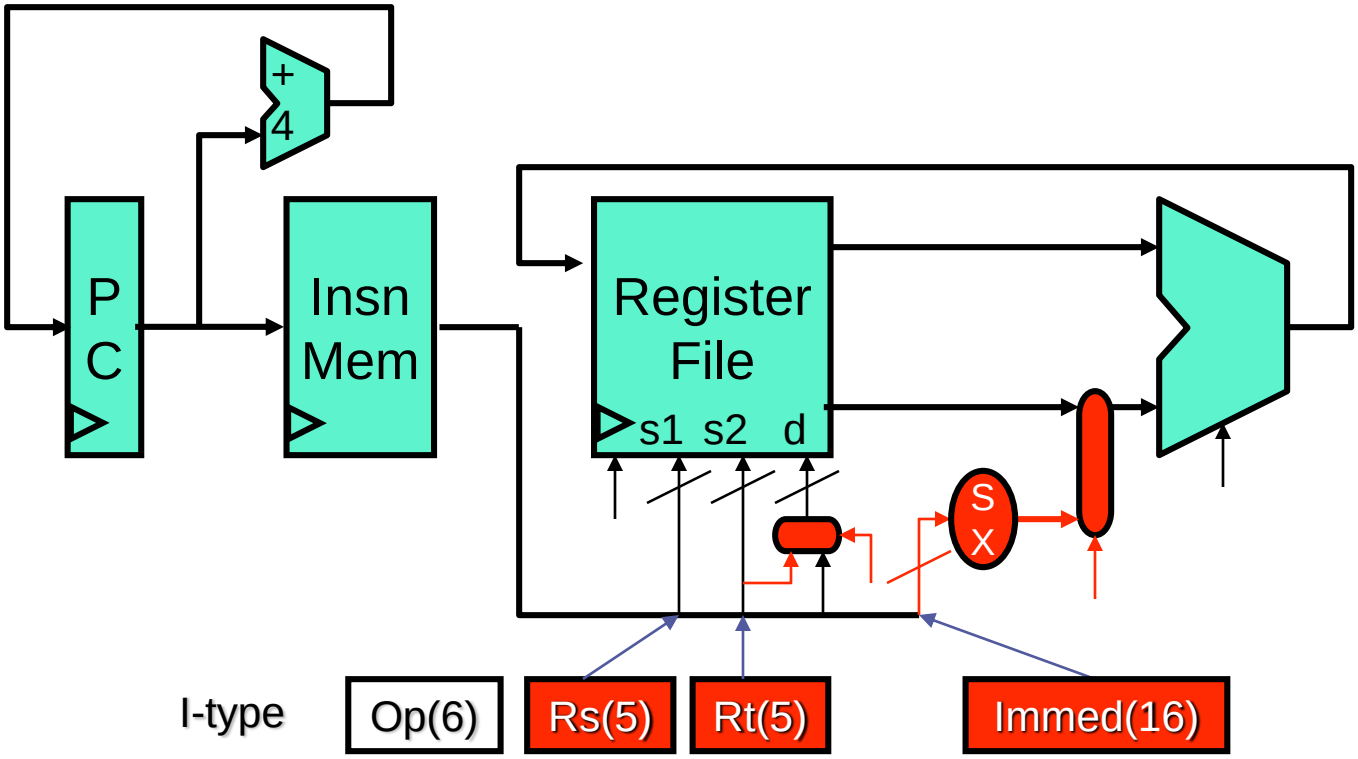
First Instruction: add



- Add register file and ALU

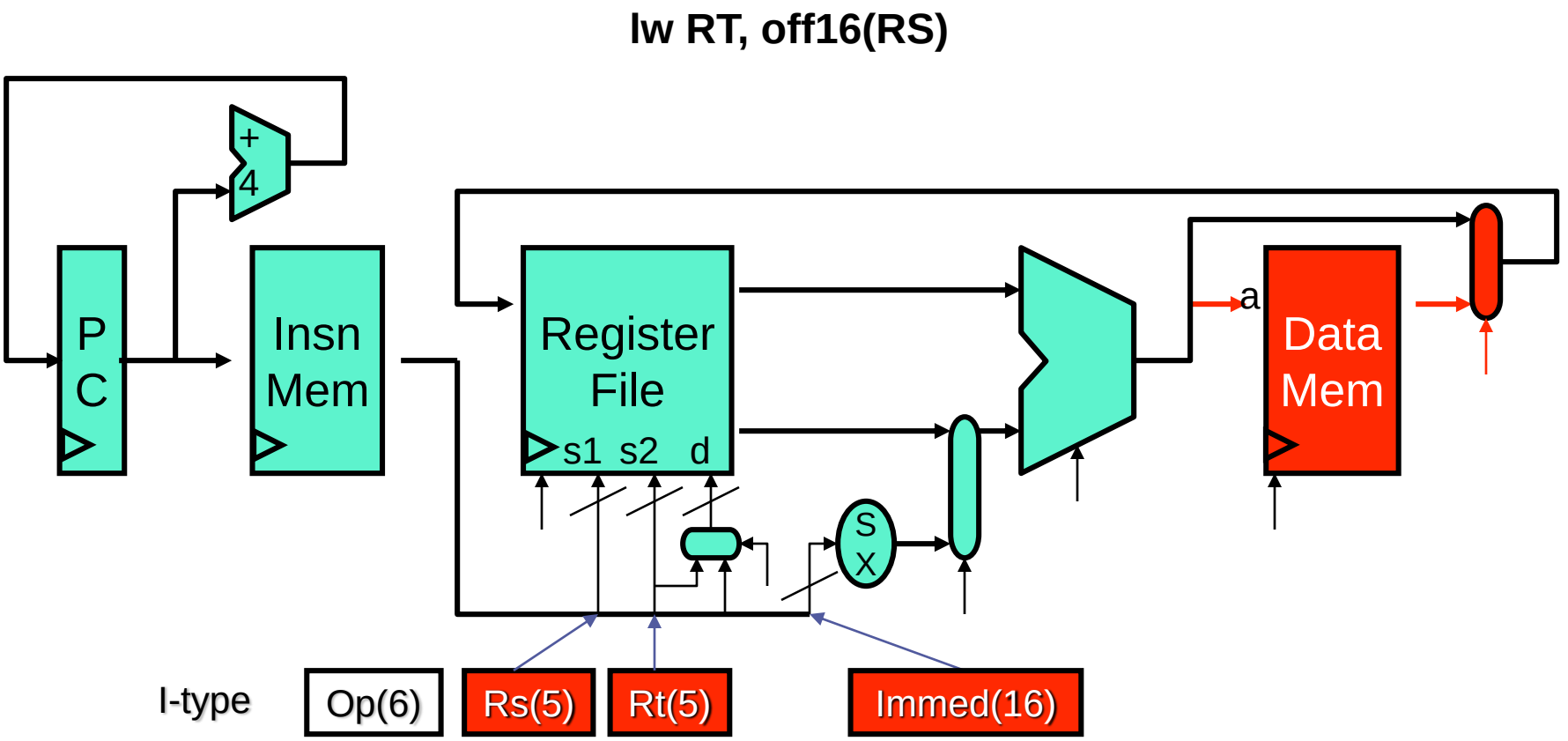
Second Instruction: addi

addi RT, RS, const16



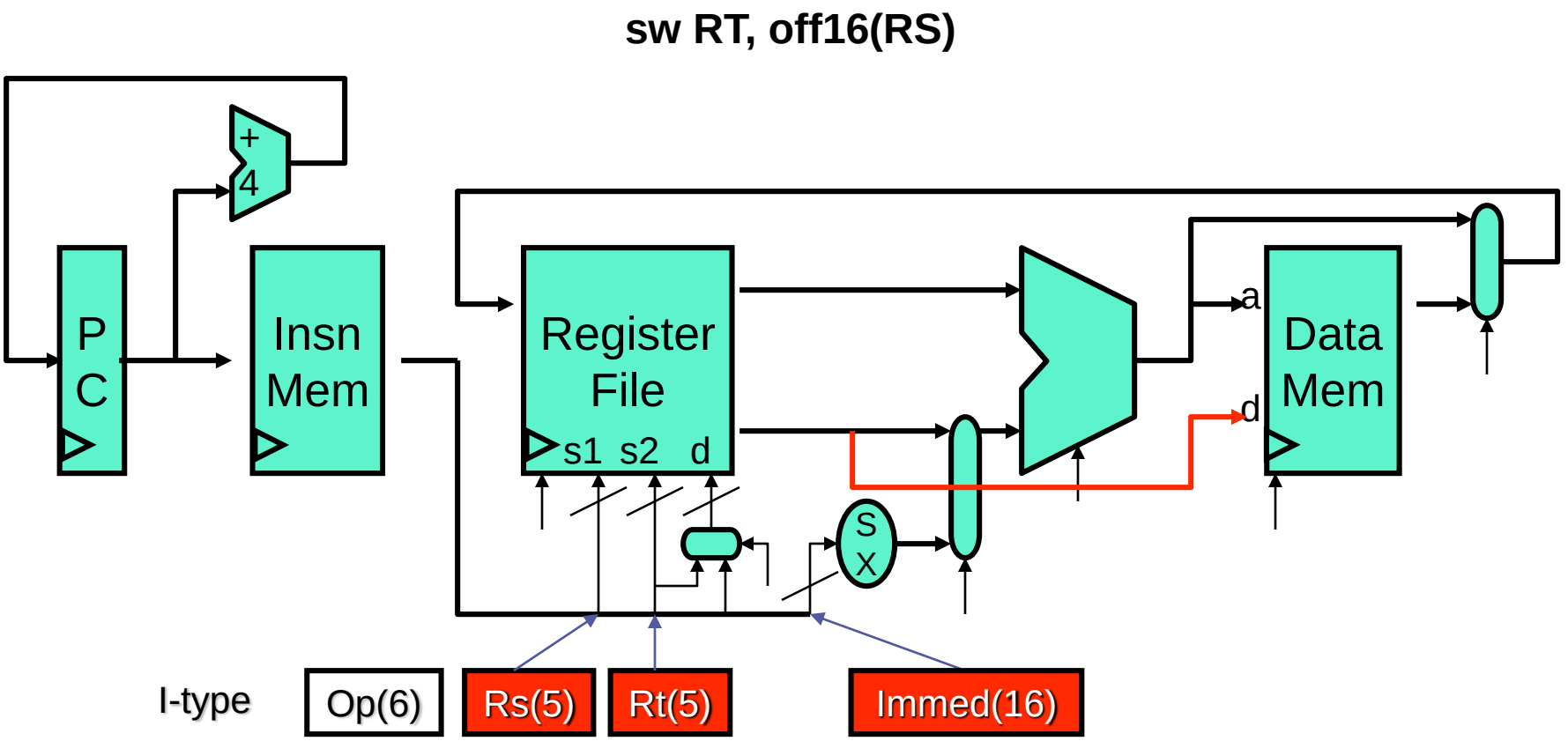
- Destination register can now be either Rd or Rt
- Add sign extension unit and mux into second ALU input

Third Instruction: lw



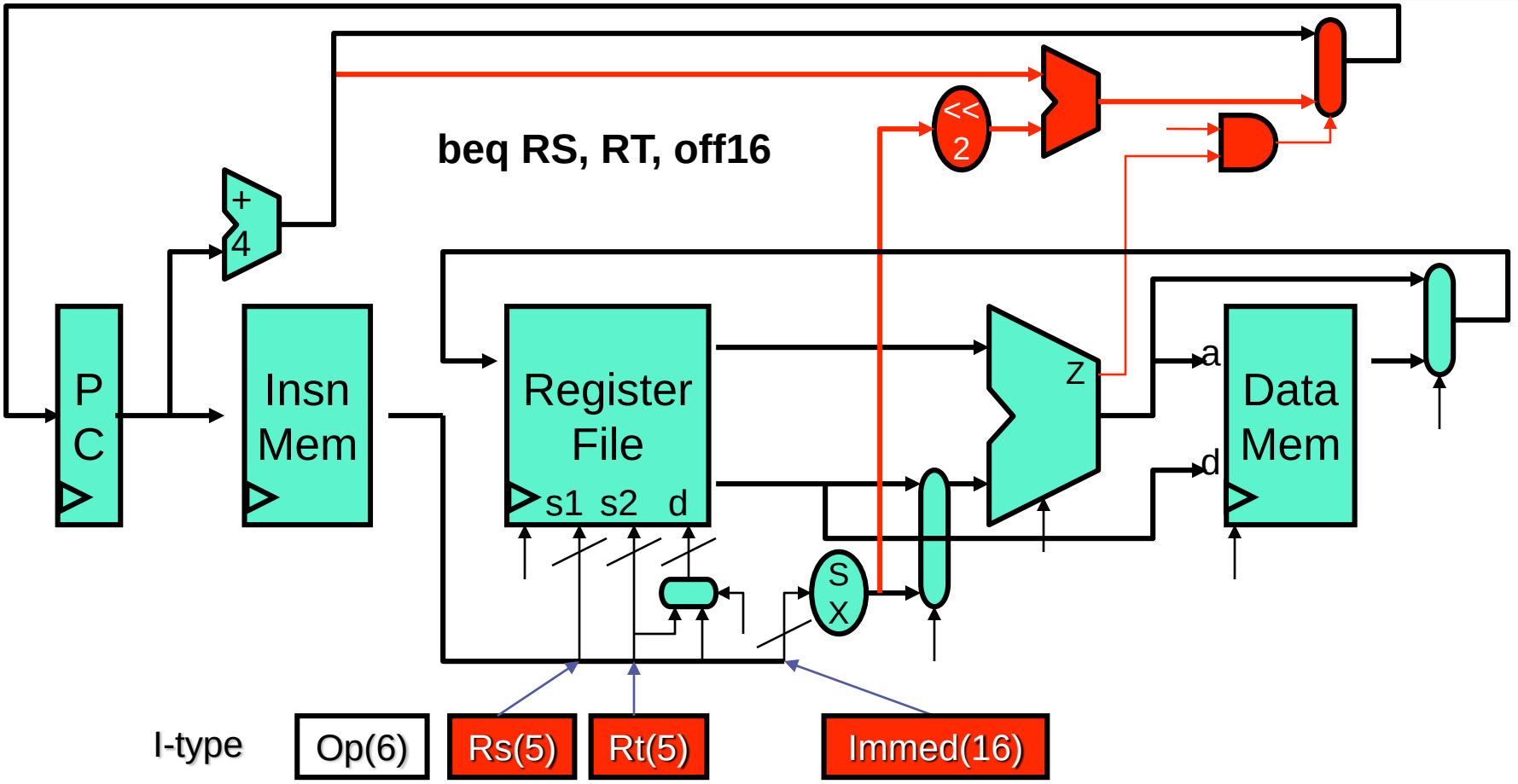
- Add data memory, address is ALU output
- Add register write data mux to select memory output or ALU output

Fourth Instruction: sw



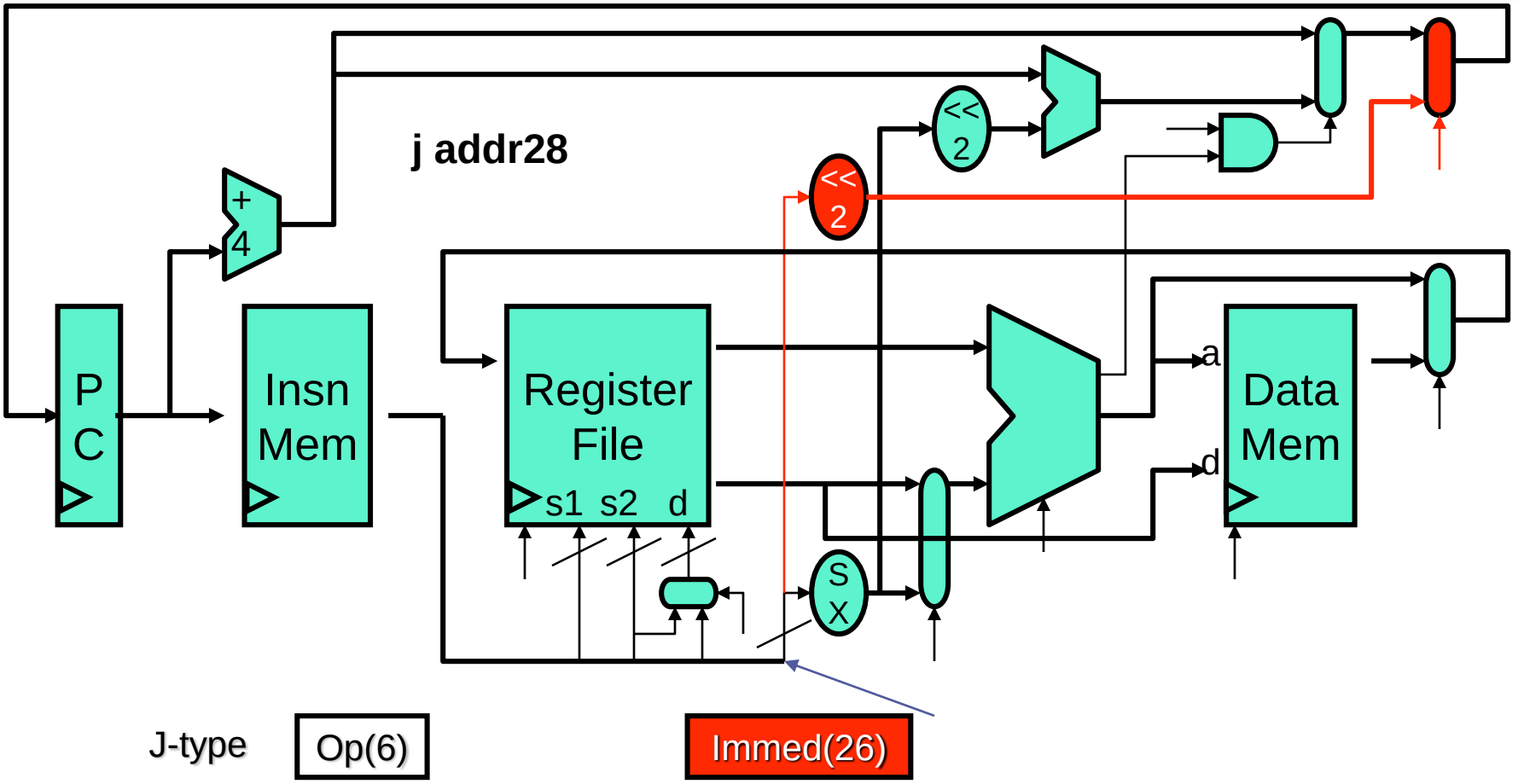
- Add path from second input register to data memory data input

Fifth Instruction: beq



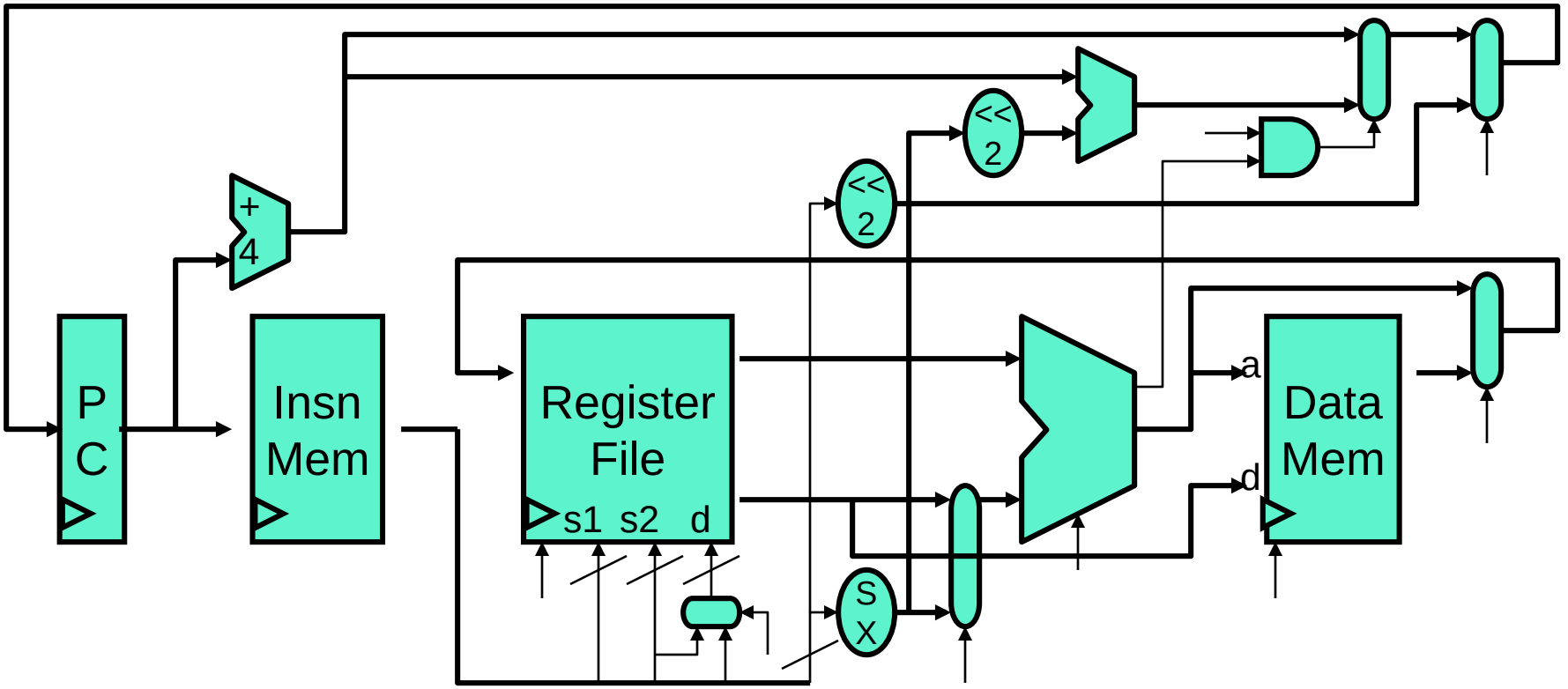
- Add left shift unit and adder to compute PC-relative branch target
- Add PC input mux to select PC+4 or branch target
  - Note: shift by fixed amount very simple

Sixth Instruction: j



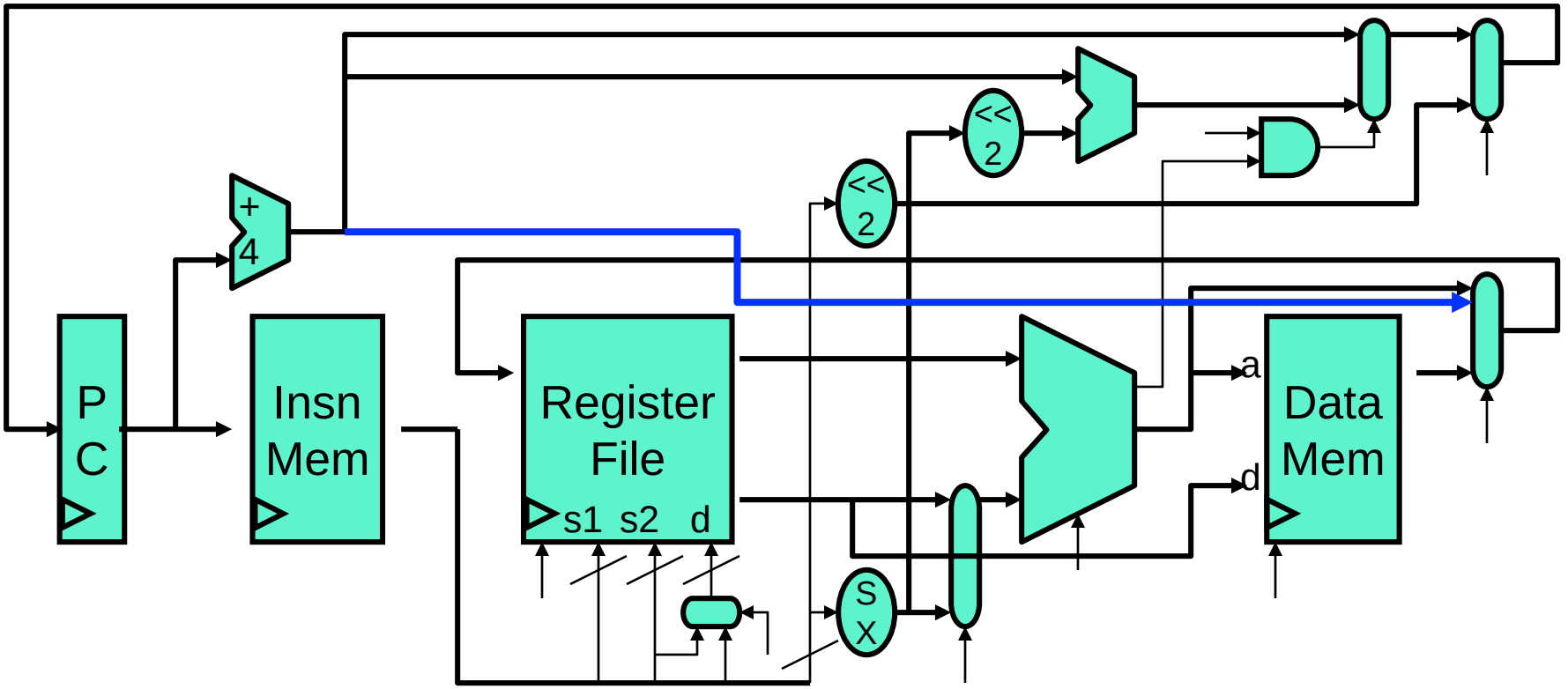
- Add shifter to compute left shift of 26-bit immediate
- Add additional PC input mux for jump target

More Instructions…



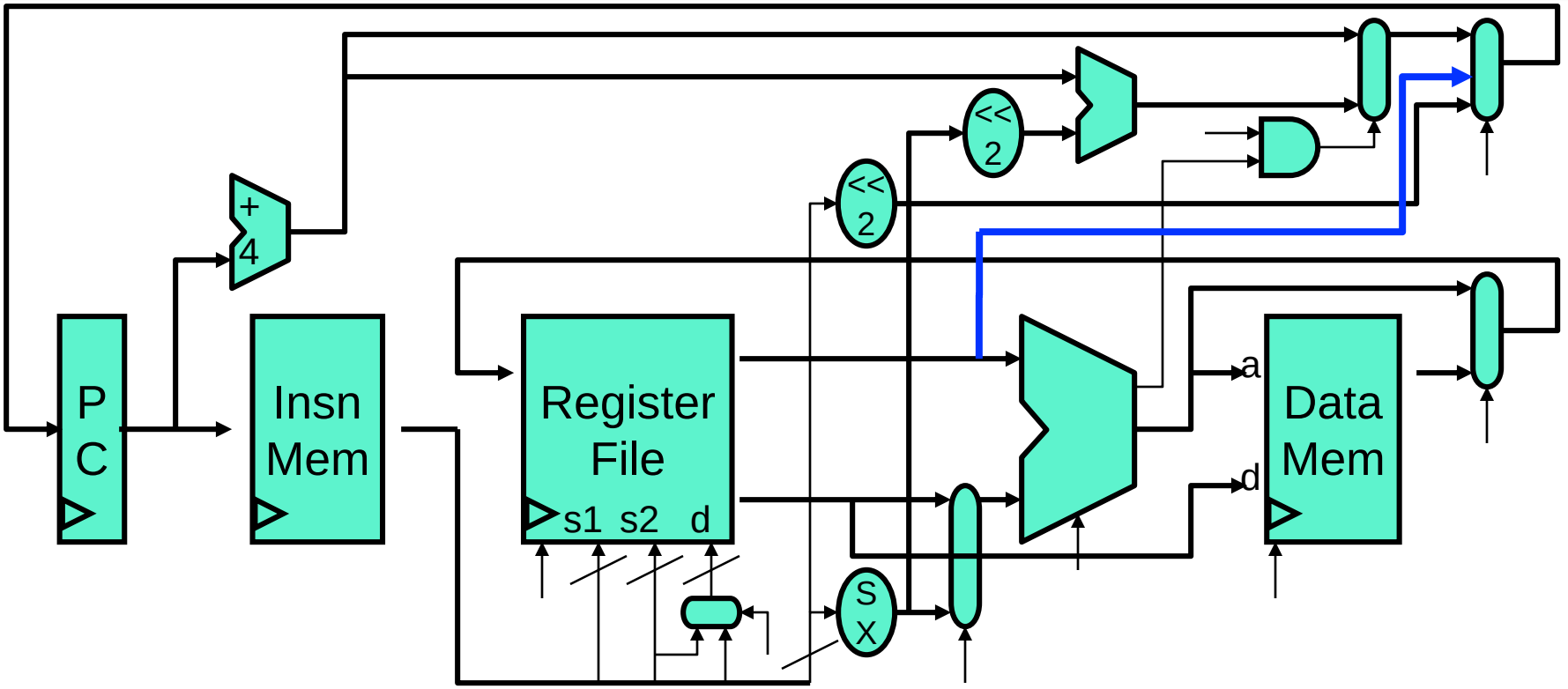
- Figure out datapath modifications for
  - jal (J-type)
  - jr (R-type)

Jal



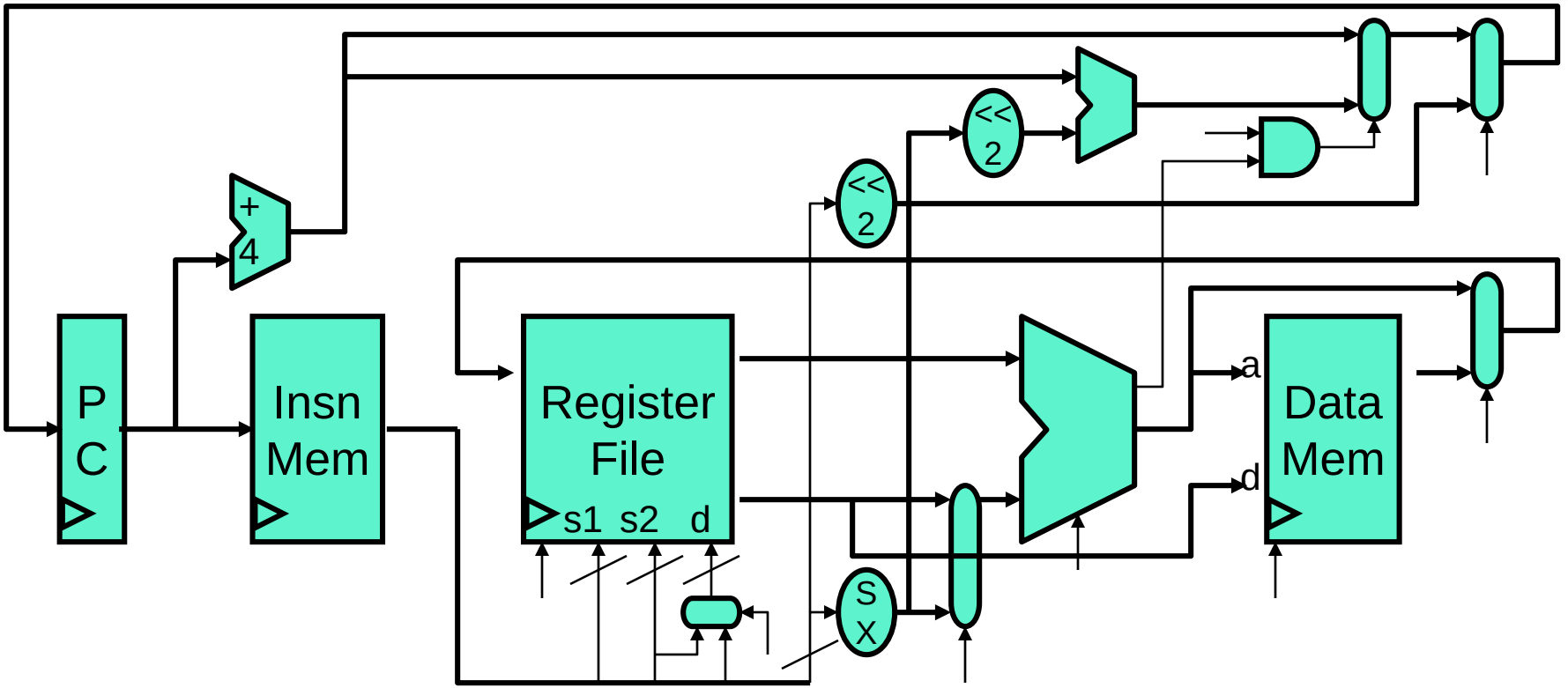
- For jal, need to get PC+4 to RF write mux

JR



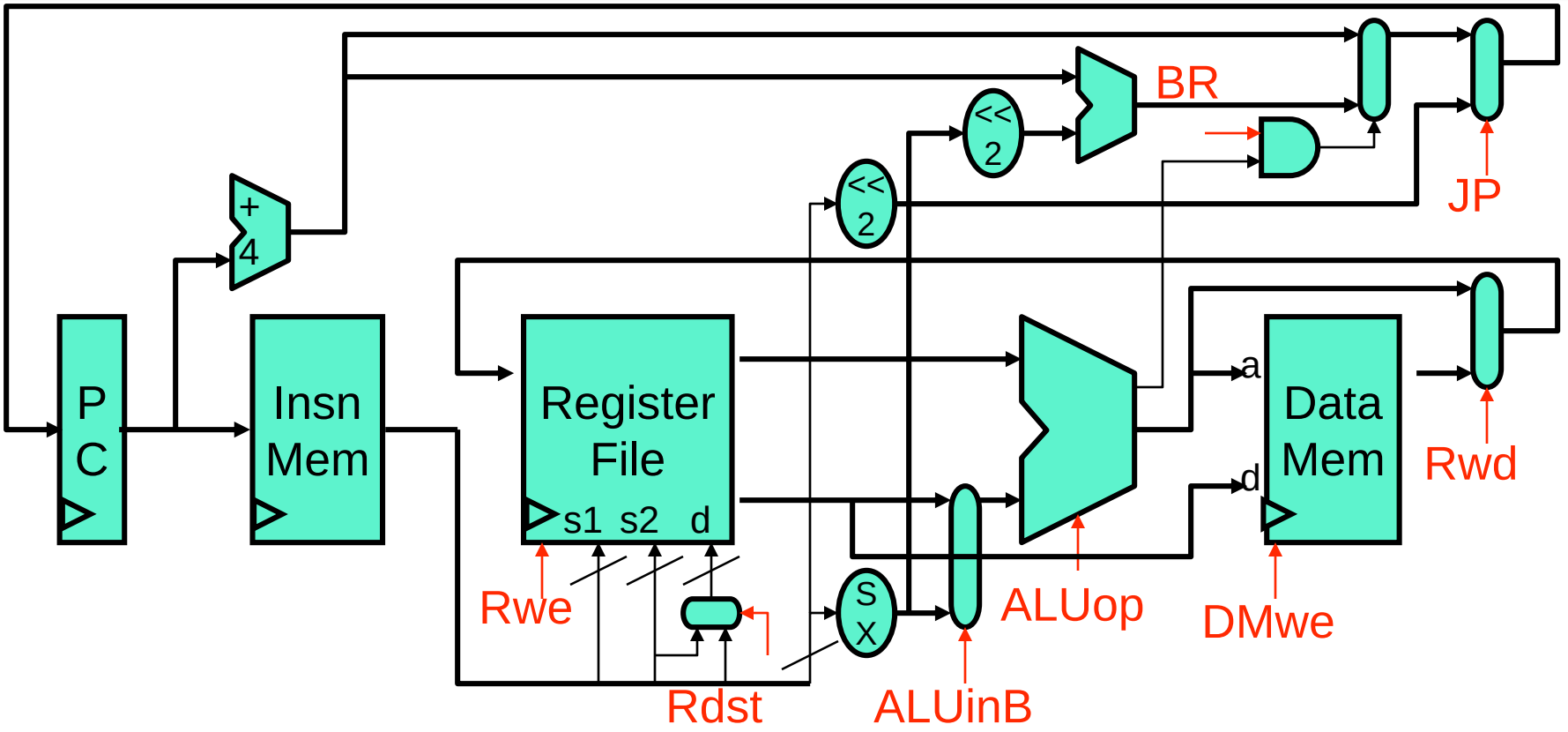
- For JR need to get RF read value to next PC mux

Good practice: Try other insns



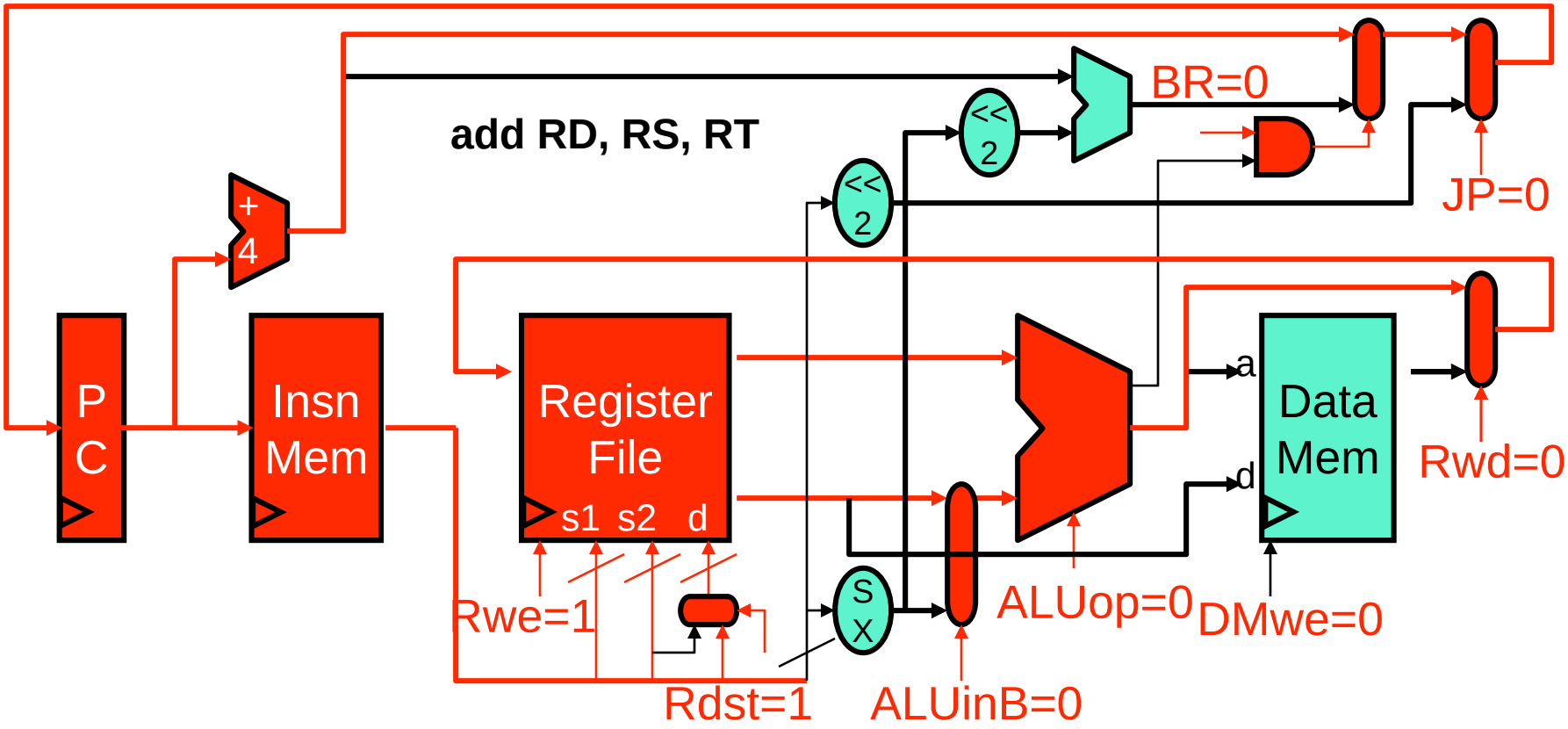
- Pick other MIPS instructions, contemplate how to add them

What Is Control?



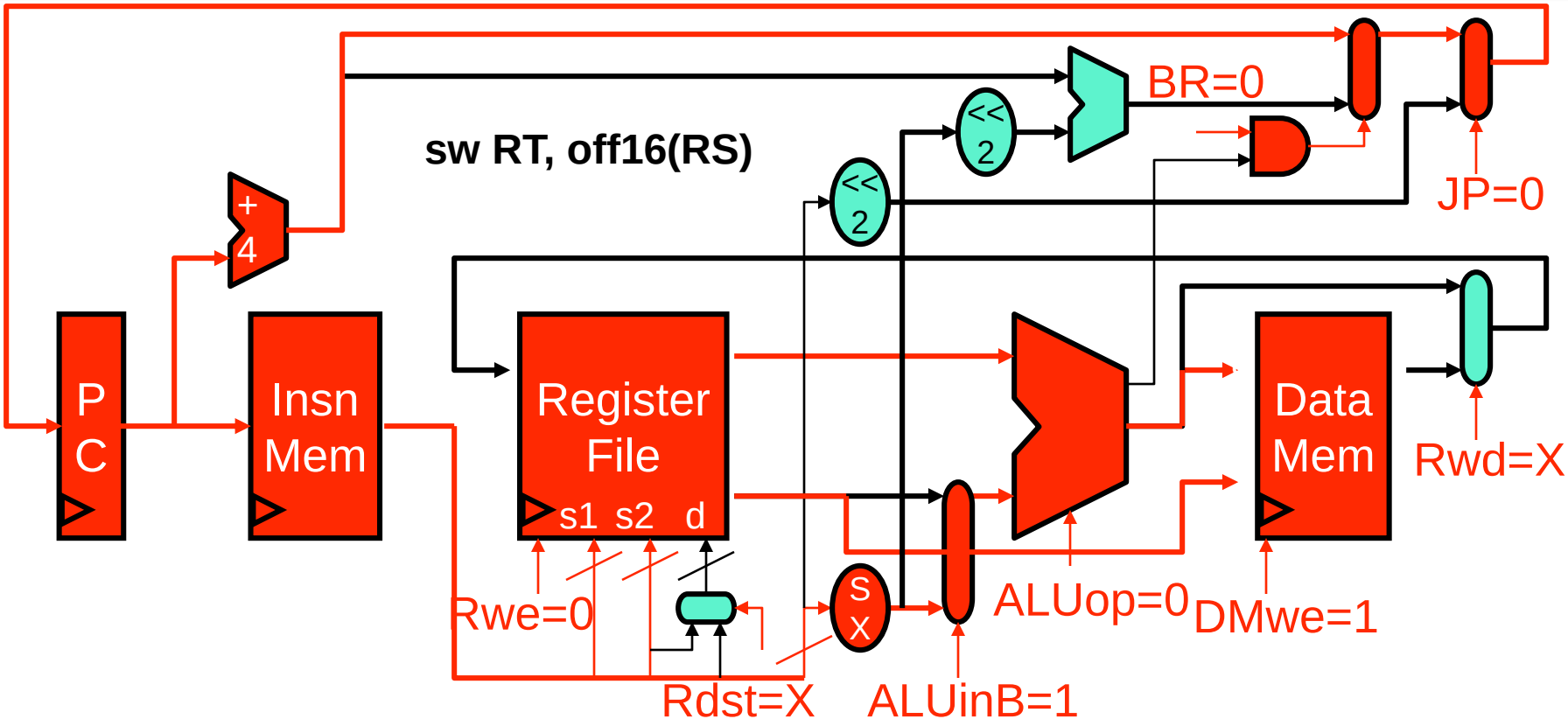
- 8 signals control flow of data through this datapath
  - MUX selectors, or register/memory write enable signals
  - A real datapath has 300-500 control signals

Example: Control for add



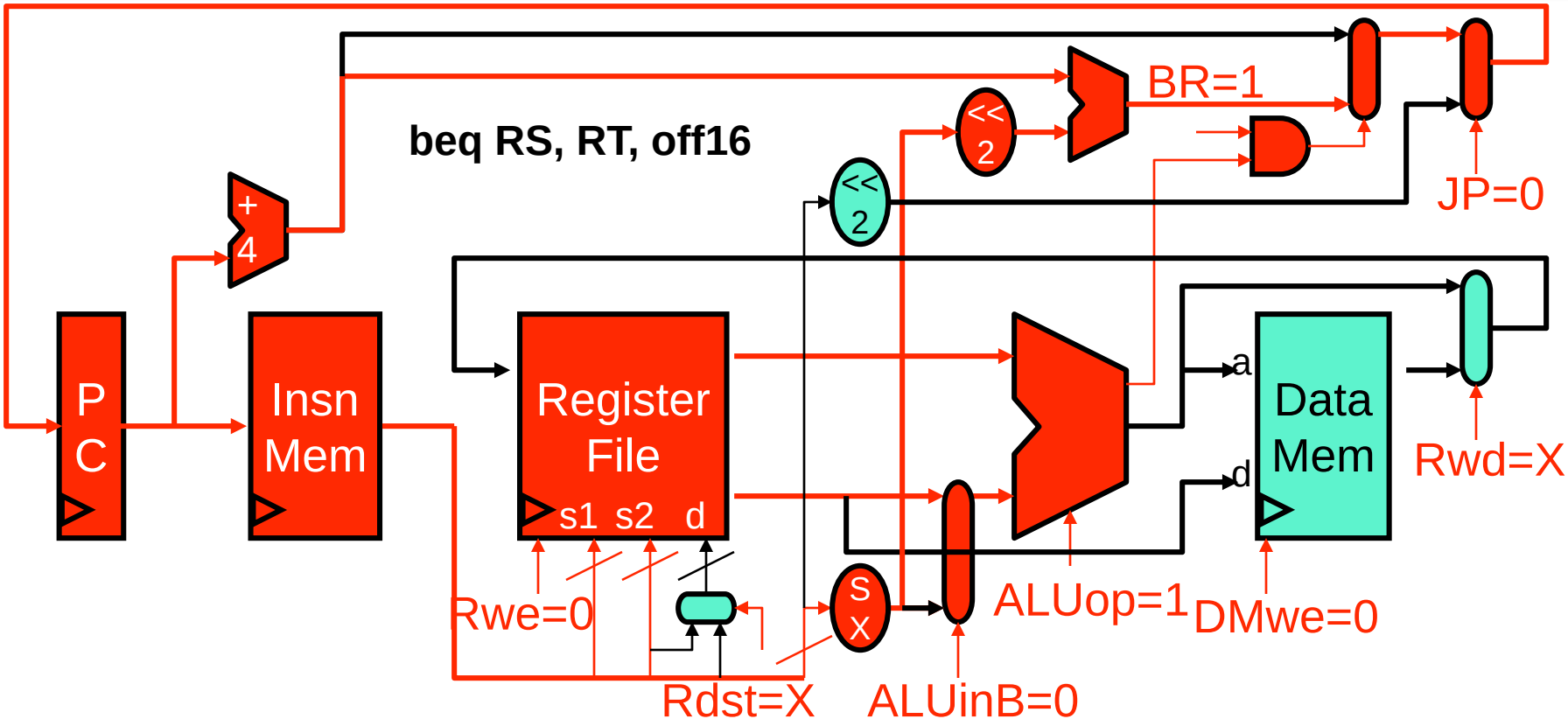
- Control for an instruction:
  - Values of all control signals to correctly execute it

Example: Control for sw



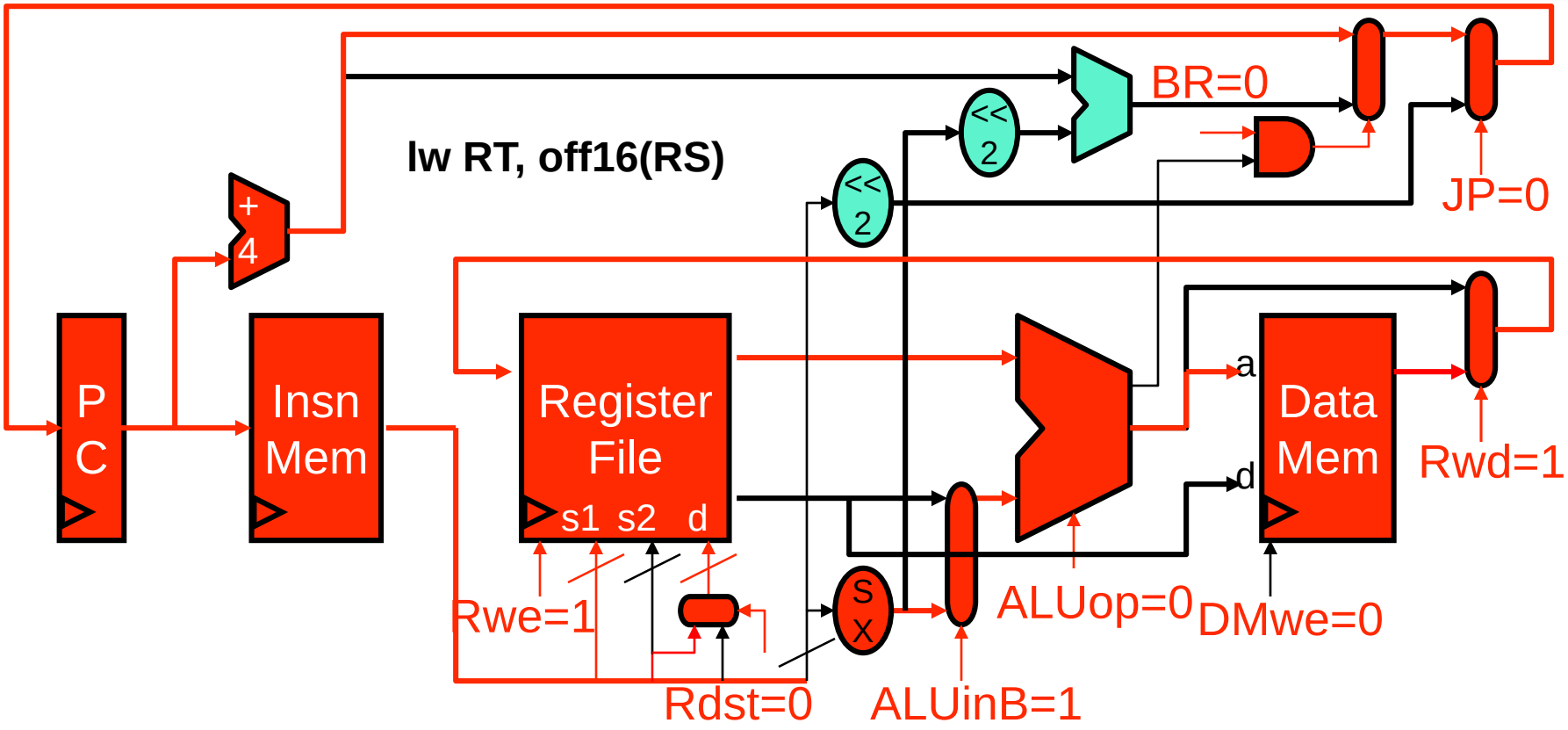
- Difference between **`sw`** and **`add`** is 5 signals
  - 3 if you don't count the X (don't care) signals

Example: Control for beq



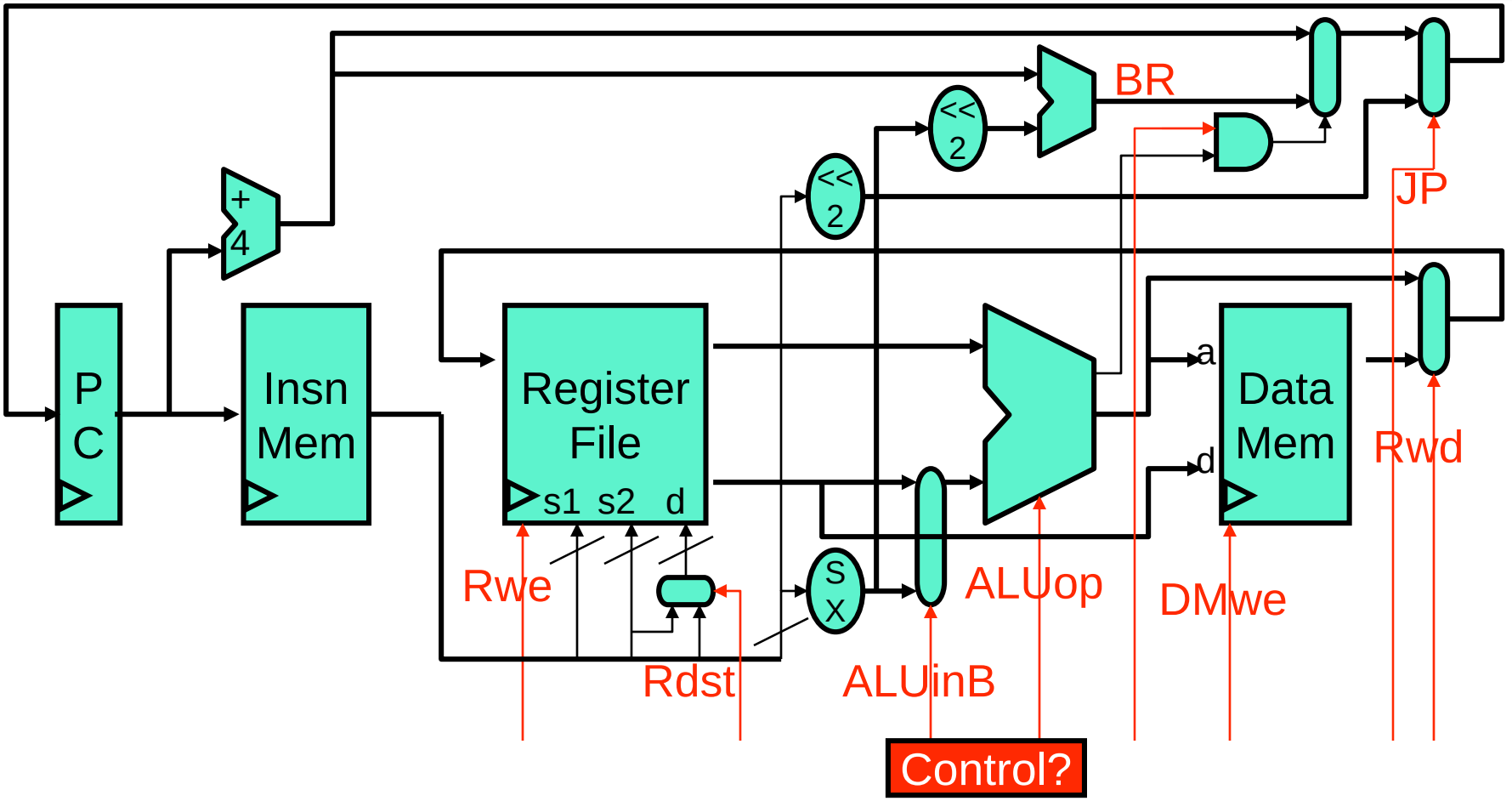
- Difference between **`sw`** and **`beq`** is only 4 signals

Example: Control for LW



- Here is the solution

How Is Control Implemented?



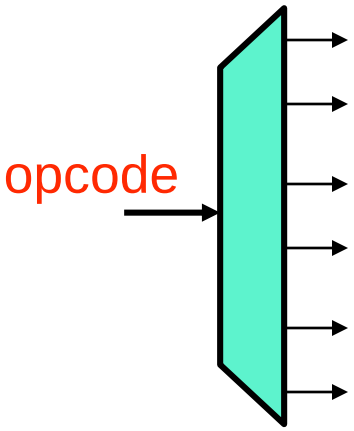
Implementing Control

- Each insn has a unique set of control signals
  - Most are function of opcode
  - Some may be encoded in the instruction itself
    - E.g., the ALUop signal is some portion of the MIPS Func field

    + Simplifies controller implementation
    - Requires careful ISA design

Control Implementation: ROM

- **ROM (read only memory)**: think rows of bits
  - Bits in data words are control signals
  - Lines indexed by opcode
  - Example: ROM control for 6-insn MIPS datapath
  - X is "don't care"

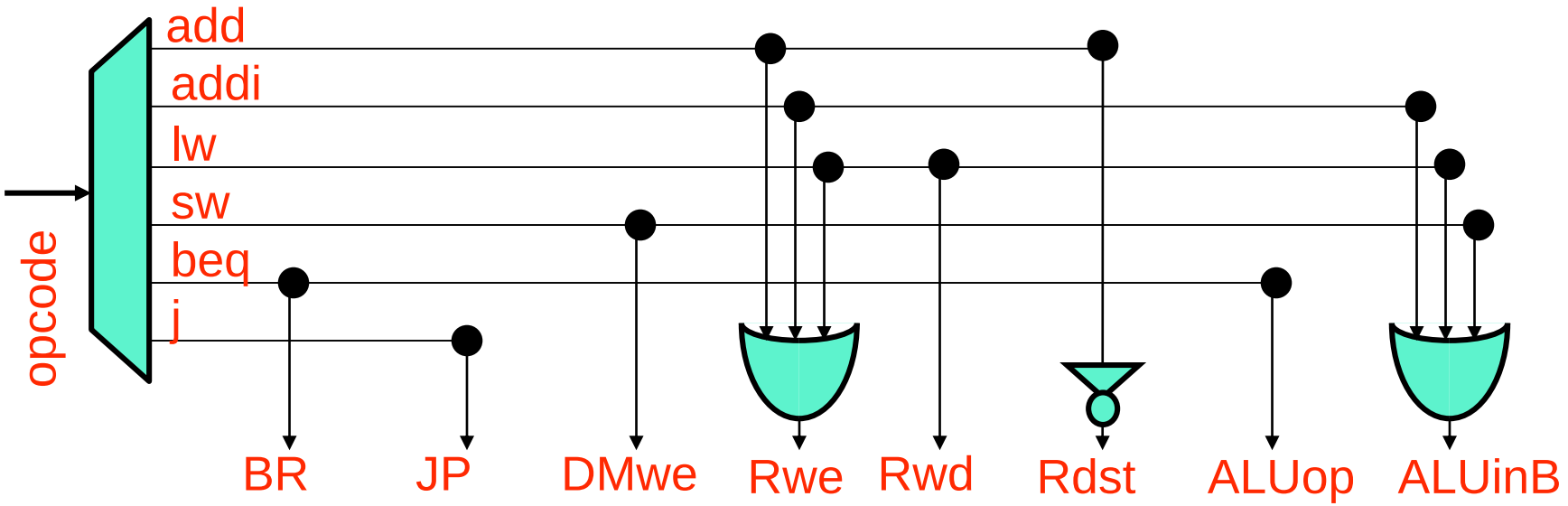


	BR	JP	ALUinB	ALUop	DMwe	Rwe	Rdst	Rwd
add	0	0	0	0	0	1	0	0
addi	0	0	1	0	0	1	1	0
lw	0	0	1	0	0	1	1	1
sw	0	0	1	0	1	0	X	X
beq	1	0	0	1	0	0	X	X
j	0	1	0	0	0	0	X	X

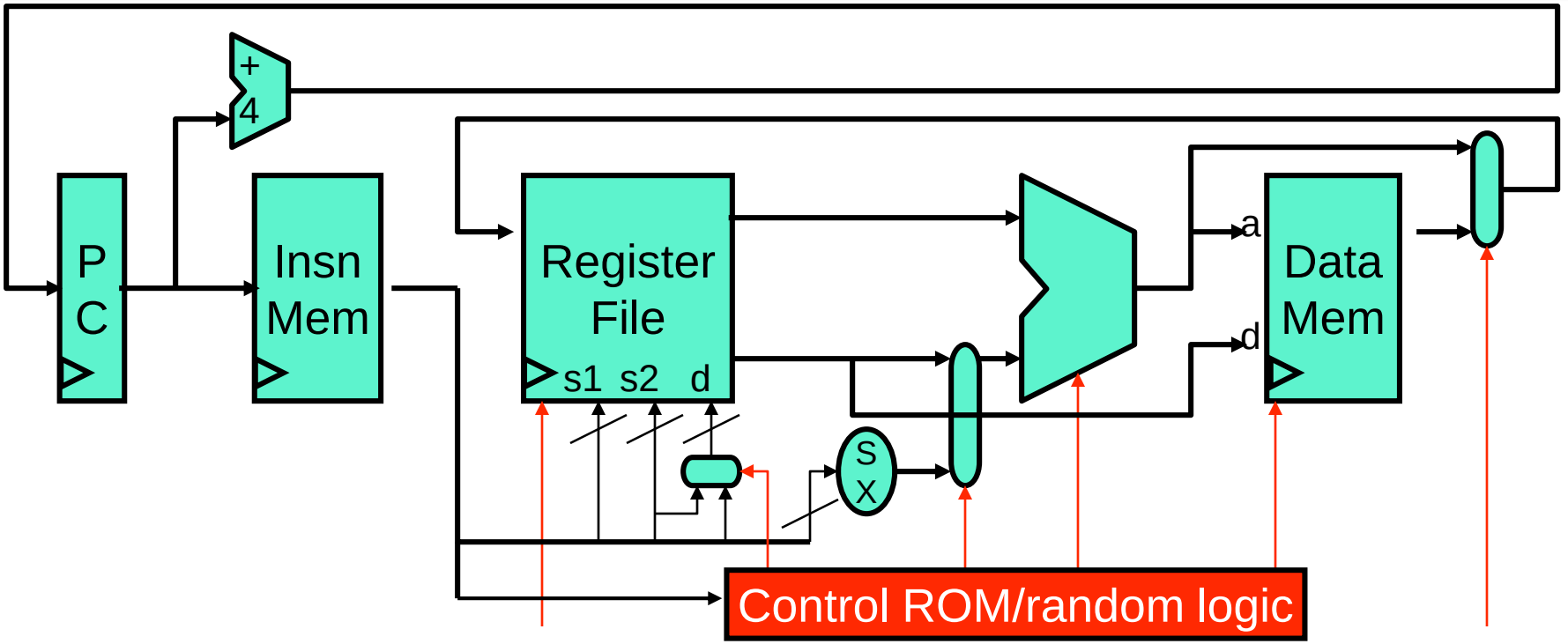
Control Implementation: Random Logic

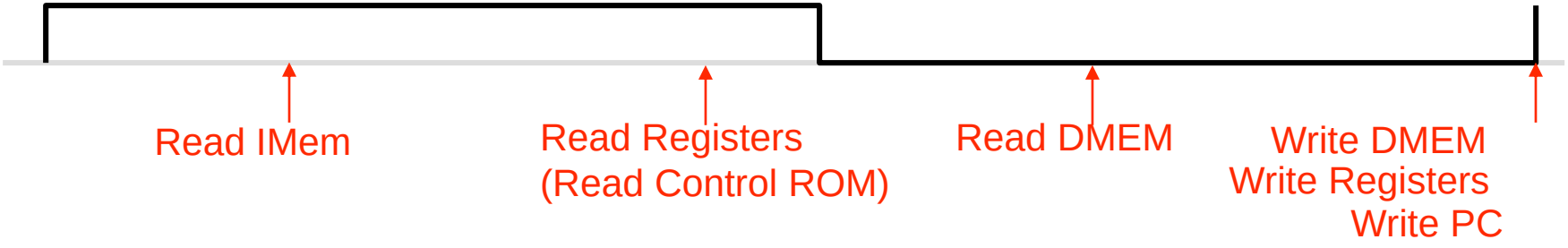
- Real machines have 100+ insns 300+ control signals
  - 30,000+ control bits (~4KB)
  - Not huge, but hard to make faster than datapath (important!)
- Alternative: random logic (random = 'non-repeating')
  - Exploits the observation: many signals have few 1s or few 0s
  - Example: random logic control for 6-insn MIPS datapath

Yes, "random logic" is a very dumb and misleading name for this concept. Sorry.



Datapath and Control Timing





Summary

- Datapaths:
  - What do we need?
  - How do we control?
  - How are control signals implemented?