信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Datapath

Instructors:
Siting Liu & Chundong Wang
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2024/index.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2024/4/9

Administratives

- Lab 6 available.
- HW 3 ddl April 9th today!
- Proj1.1 will be checked next week
- Proj1.2 ddl April 25th
- Proj2.1 will be released soon
- Discussion (teaching center 301) schedule
  - Datapath for proj 2.1
  - The same content for Friday and the next Monday.

Mid-term I

- Midterm I
  - April 11th 8:00 am - 10:00 am
    - We start sharp at 8:00 am!
    - Arrive 7:45 am to check-in (three classrooms likely and seat table will be determined on-site)
    - Arrive later then 8:30 am will get 0 mark.
- Contents:
  - Everything till April 9th lecture
- Switch cell phones **off**!!! (not silent mode)
  - Put them in your bags.
- Bags in the front. On the table: nothing but pen, exam paper, 1 drink, 1 snack, your student ID card and your cheat sheet!
- One of teaching center 301/304/404, check on-site

Mid-term I requirements

- You can bring a cheatsheet (**handwritten only**). **1**-page **A4**, **double-sided** (2-page for the mid-term II and 3-page for the final). Put it on your desk at exam. Cheatsheet that does not apply to the rules would be taken away.
- Greencard shown on the course website is provided with the exam paper.
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch, calculator, computer…
- Anybody touching any electronic device will **FAIL** the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will **FAIL** the course!

Cheat Sheet

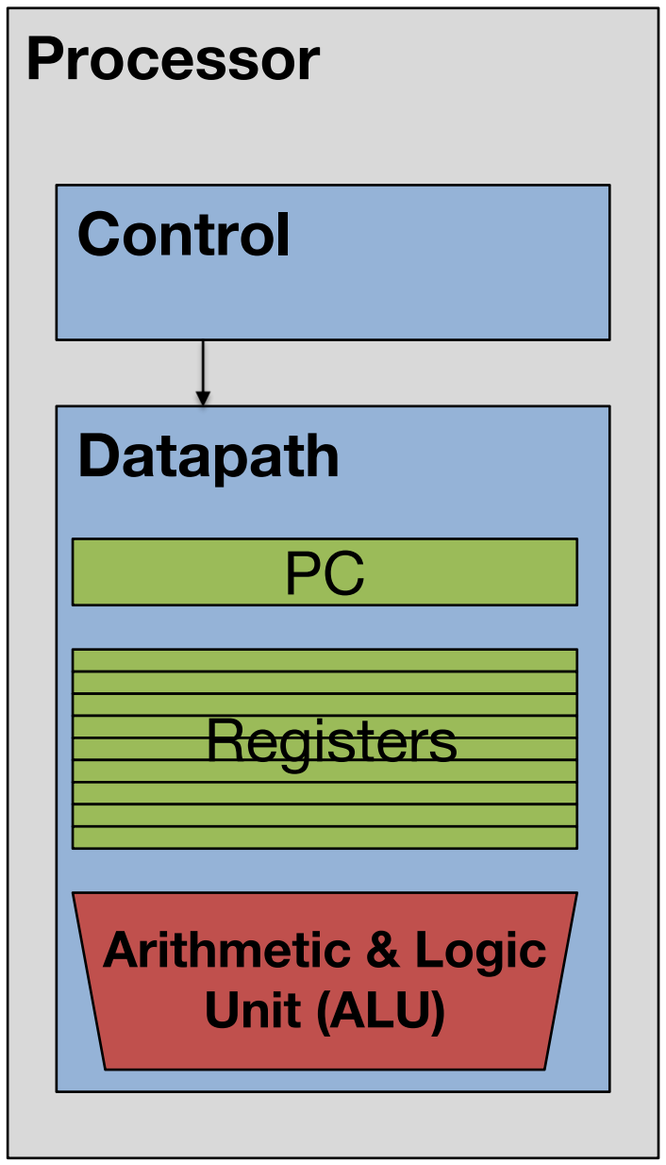
- 1 A4 Cheat Sheet allowed (double sided)
  - Midterm II: 2 pages
  - Final: 3 pages
- Rules:
  - *Hand-written* – **not printed/photocopied!**
  - Your **name** in pinyin on the top!
  - Cheat Sheets not complying to this rule will be **confiscated**!

Outline

- Datapath
  - Add building blocks and control signals for different types of instructions, one type at a time
- Design of the controller
- Timing analysis

Controller & Datapath

- A CPU that support RV32I can have so many states

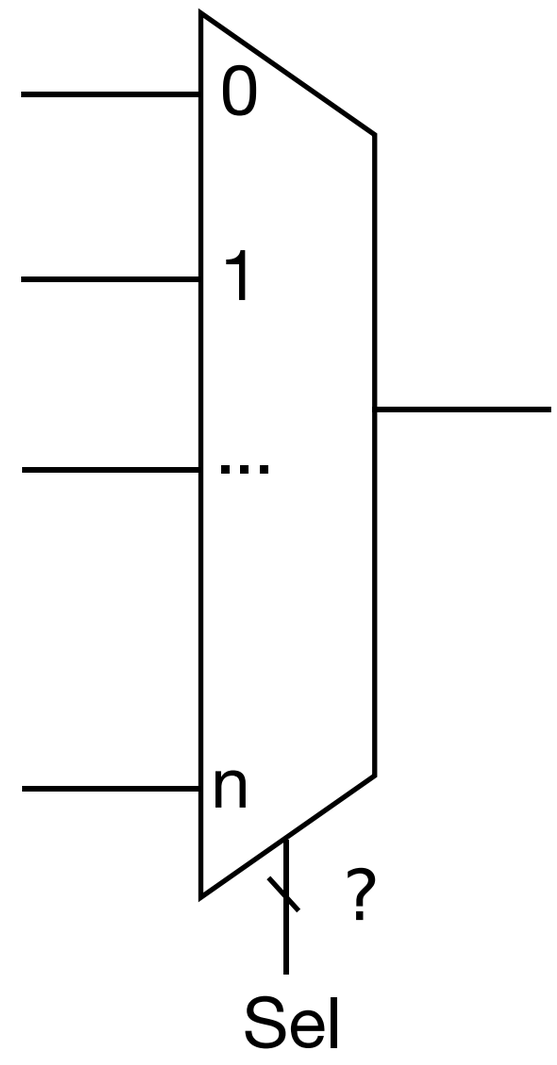


- Consider the 32 registers alone
  - `x0` always 0
  - Each bit in the other registers can be 0 or 1
- Not practical to enumerate all the state transitions
- Top-down design: build small modules and then connect them as needed
- Most digital systems can be divided into datapth and controller
  - Datapath contains data processing and storage
  - Controller controls data flow and state change (still can be modeled as FSM)
- Recall the execution of an instruction

- Our Goal: Implement a RISC-V processor as a synchronous digital system (SDS).
- Each RV32I instruction can be done within 1 clock cycle (single-cycle CPU).

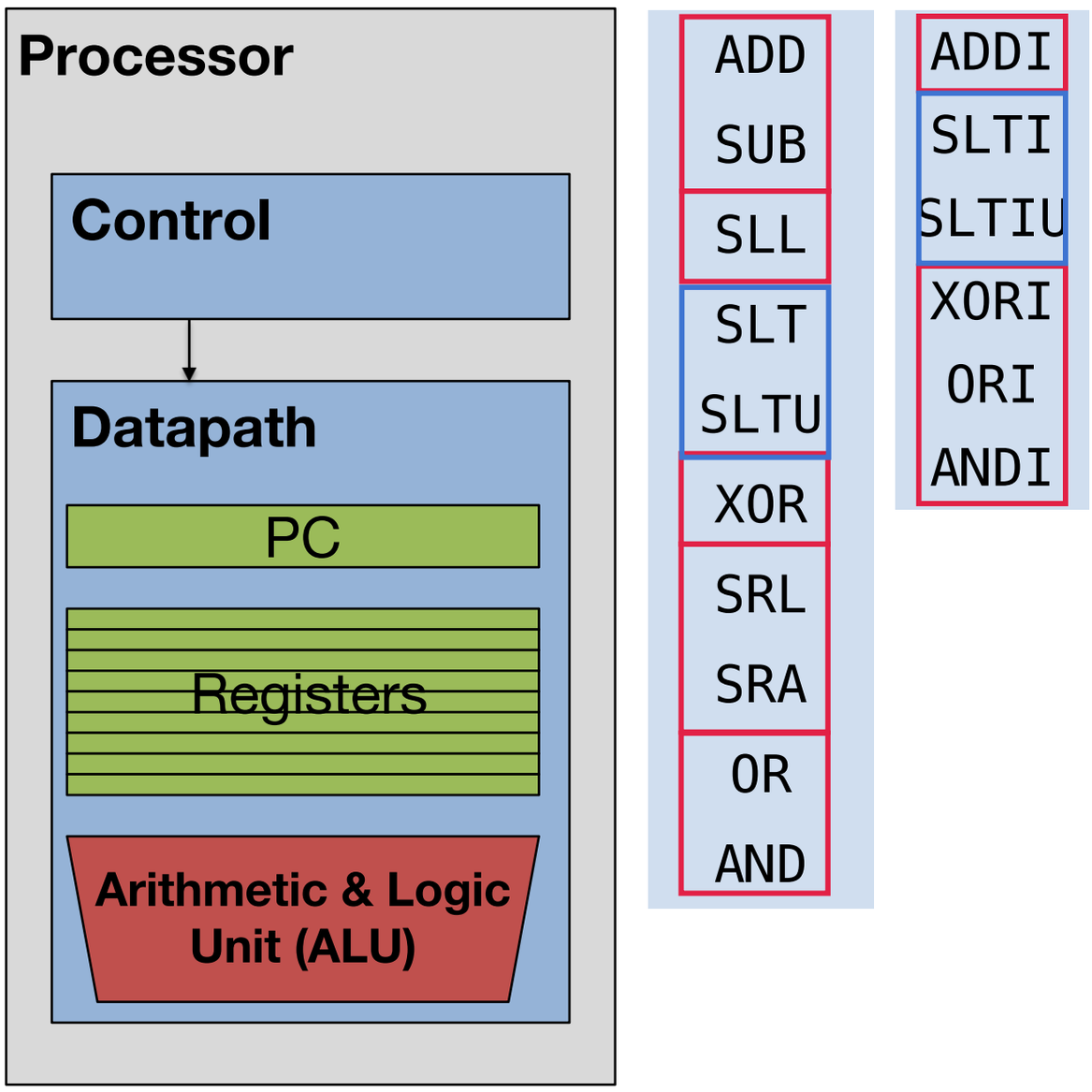
Multiplexer

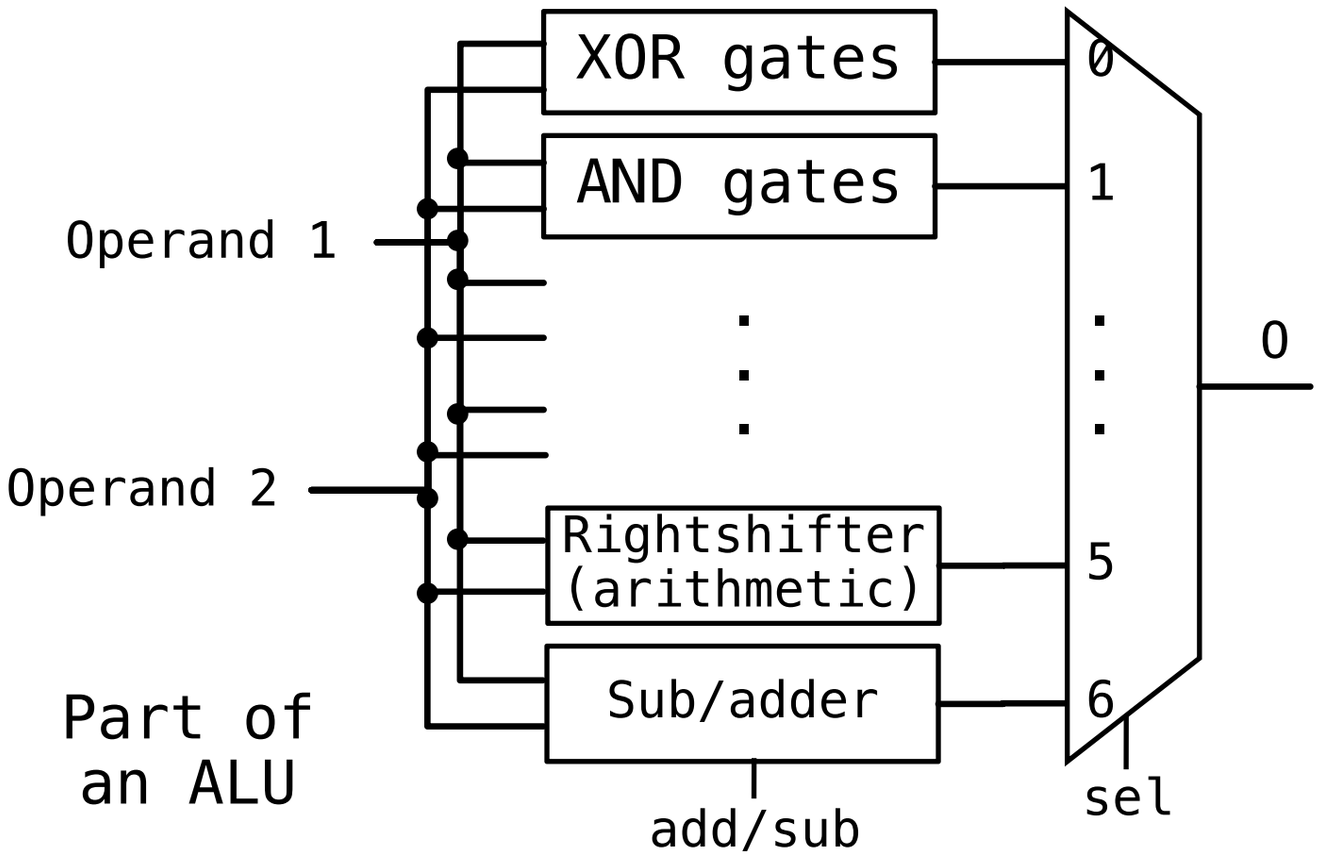
- n-to-1 multiplexer symbol



ALU

- An ALU should be able to execute all the arithmetic and logic operations

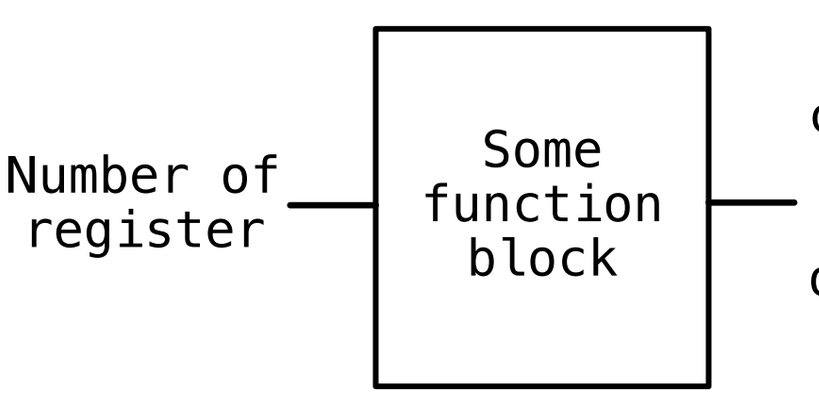




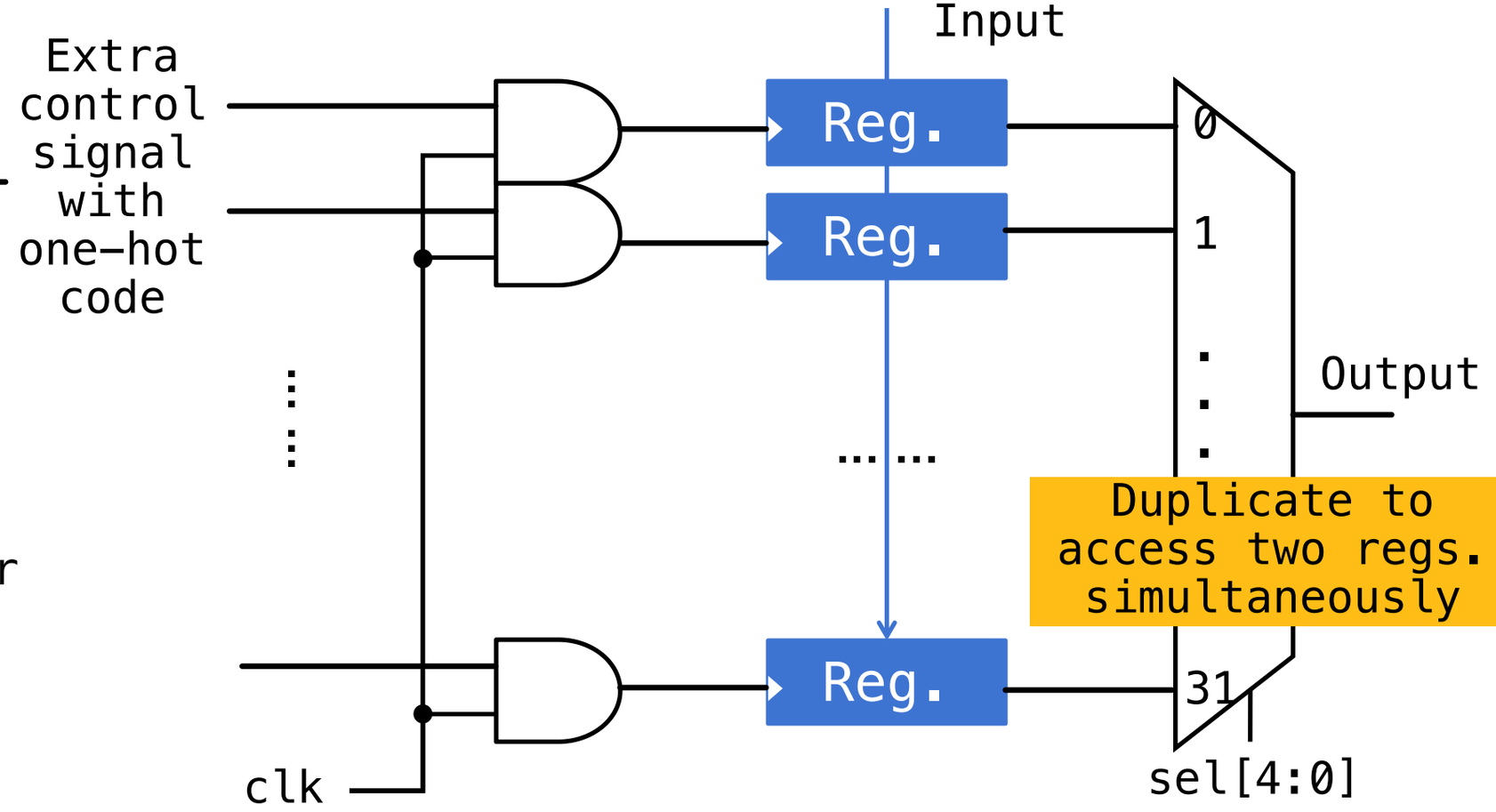
- ALU design that supports R-/I-arithmetic and logic operations completed

Register file

- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value
  - How do we change values of a specific reg.?



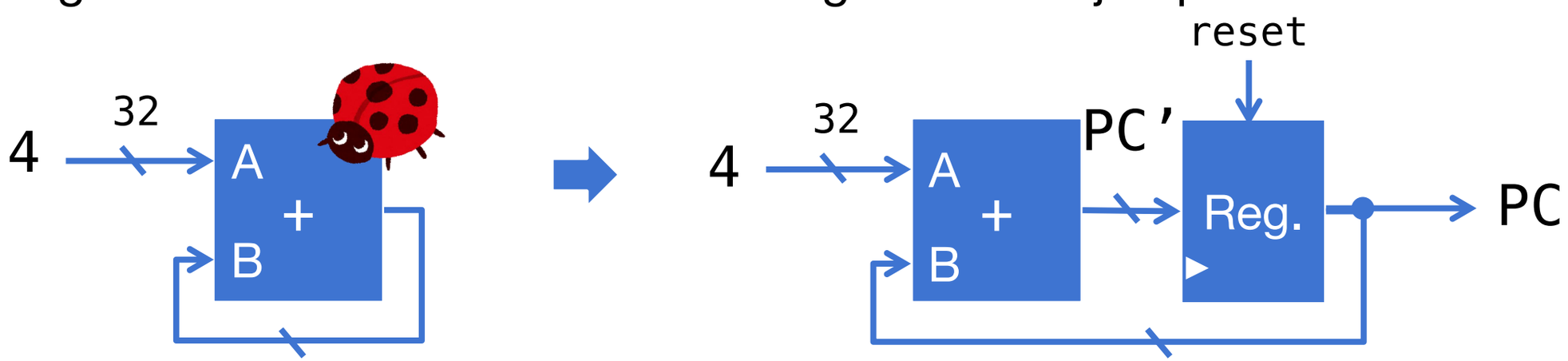
(Address) decoder
or
demultiplexer



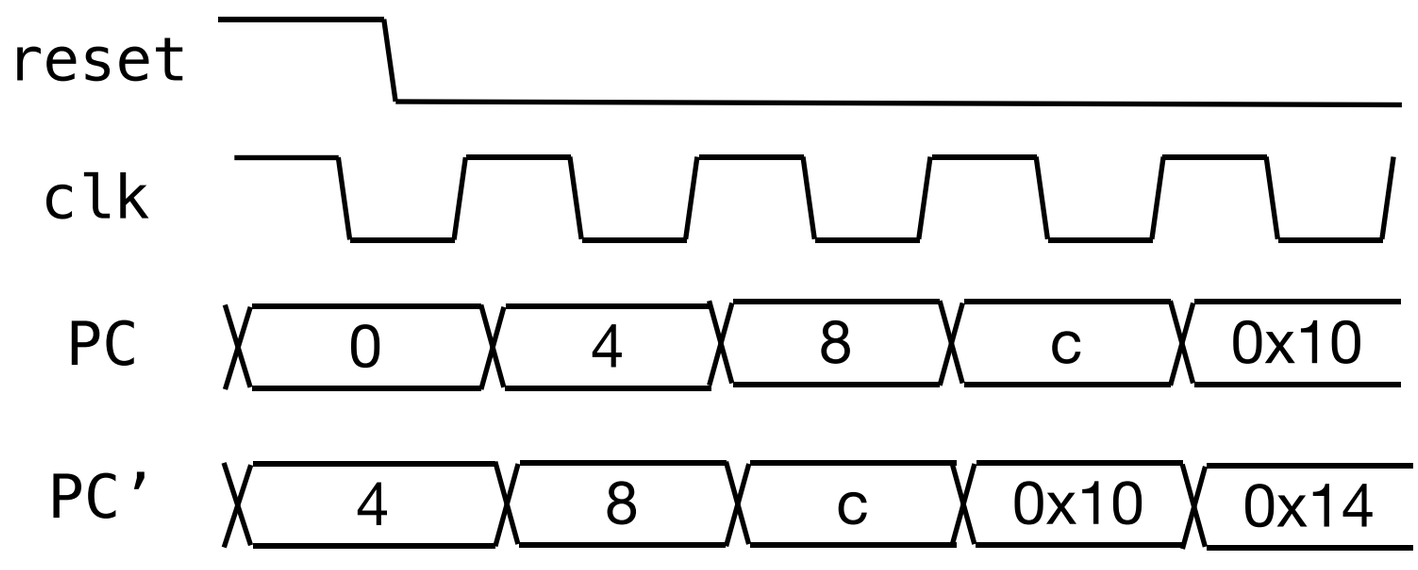
- Reg. file design completed

We have covered PC register previously

- Synchronous digital circuit can have feedback, e.g., iterative accumulator
  - e.g. PC = PC + 4 without considering branch or jump

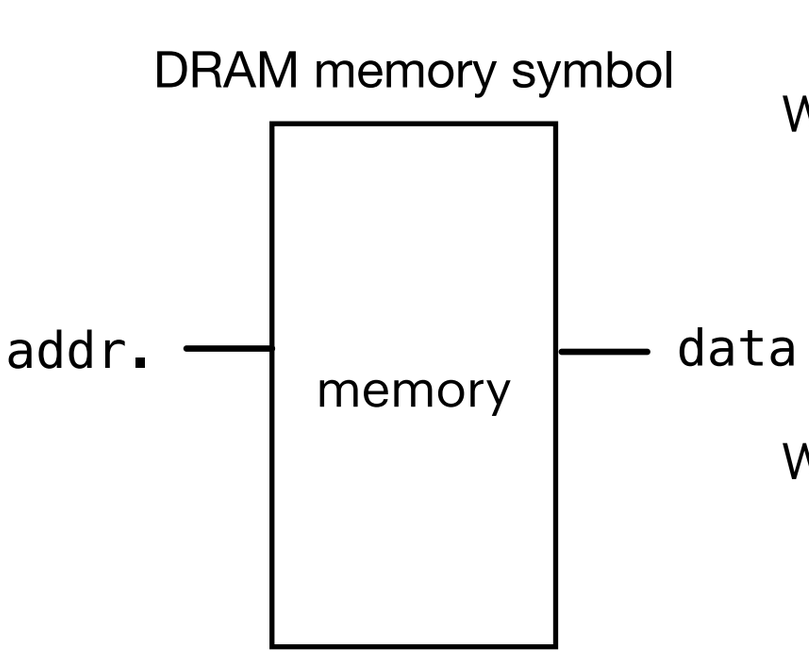


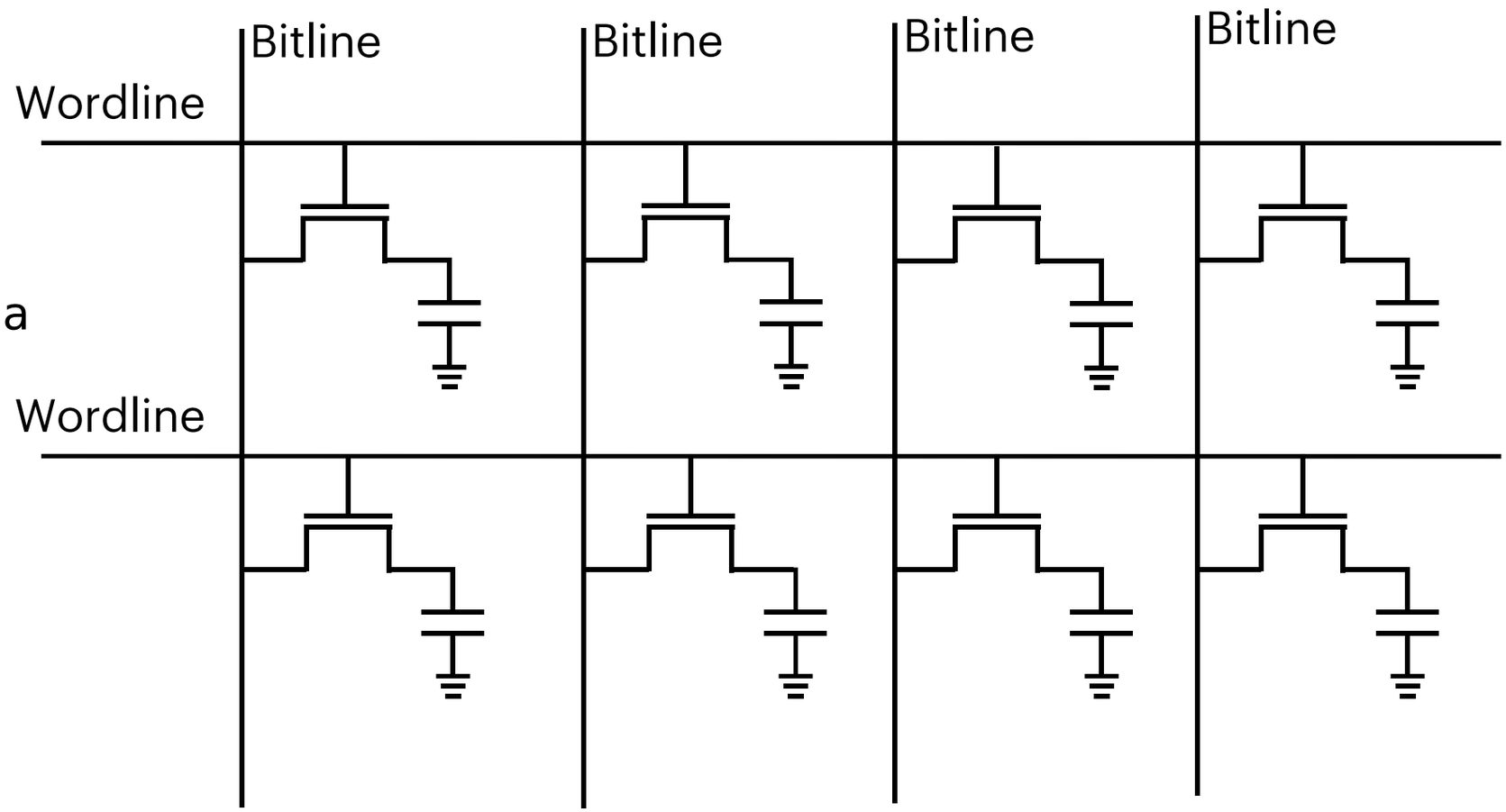
- Timing diagram



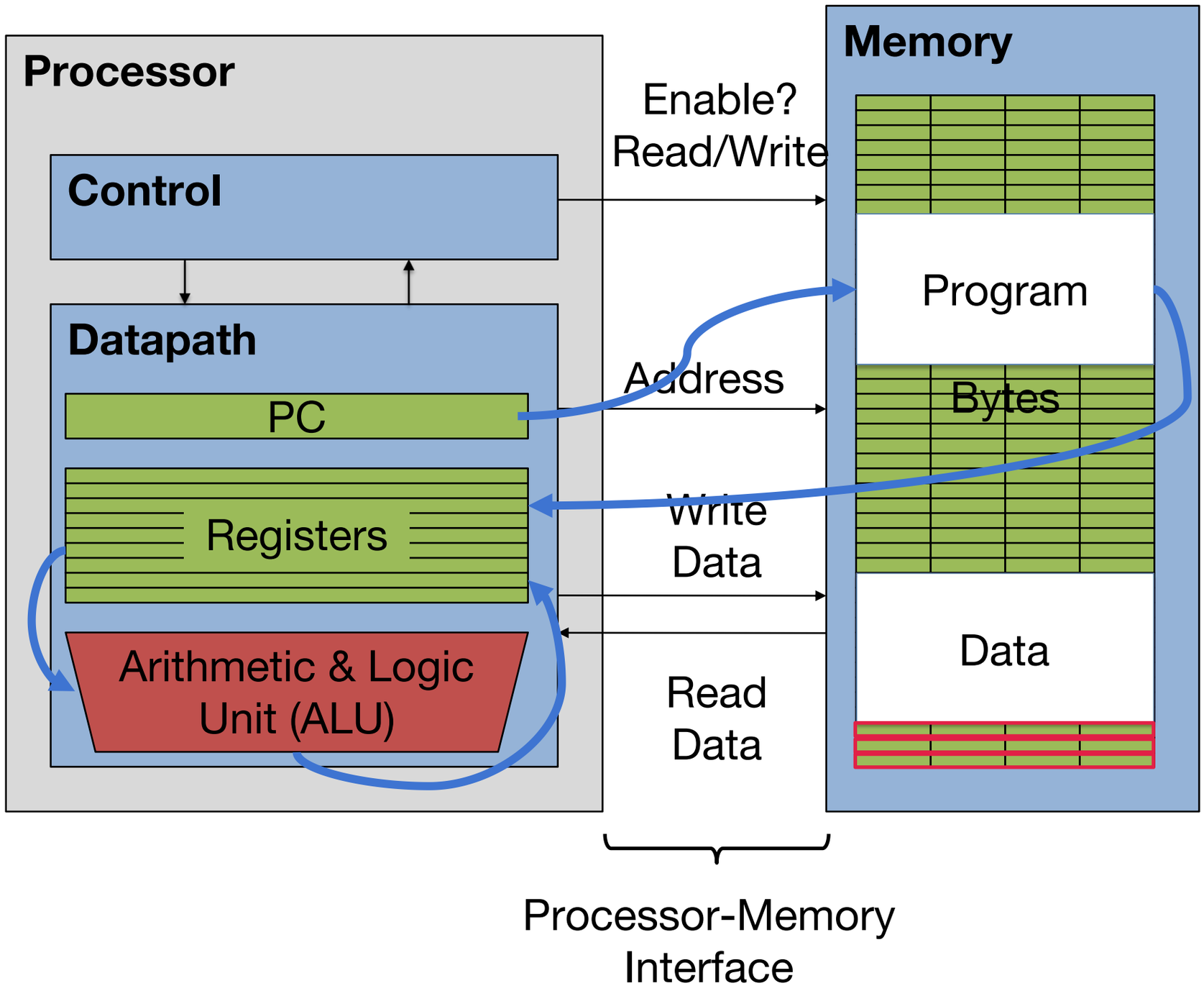
Useful building blocks-Memory

- Memory similar to register file except that the basic cell design is different
- Requires refresh for DRAM
- For ease of implementation, we only use its behavior model

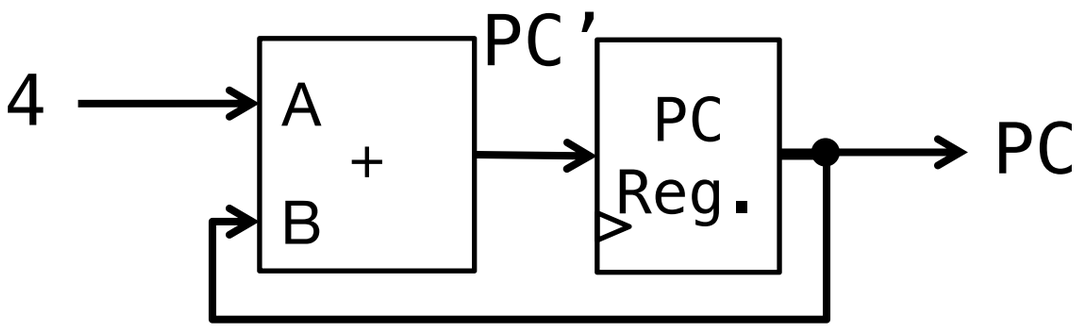




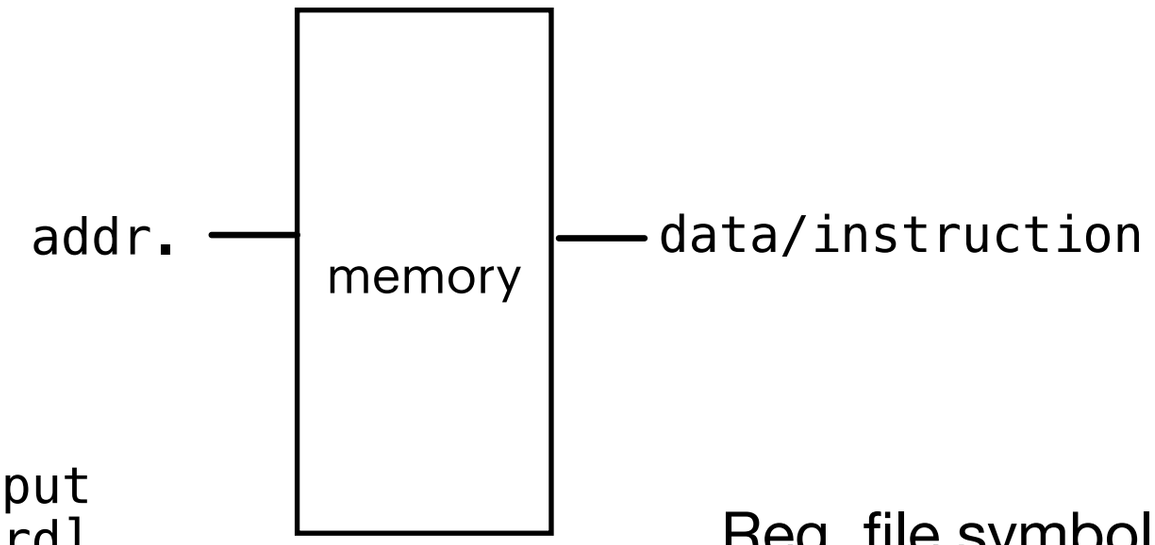
Datapath


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Enable?
Read/Write
Address
Write Data
Read Data
Memory
Program
Bytes
Data
Processor-Memory Interface

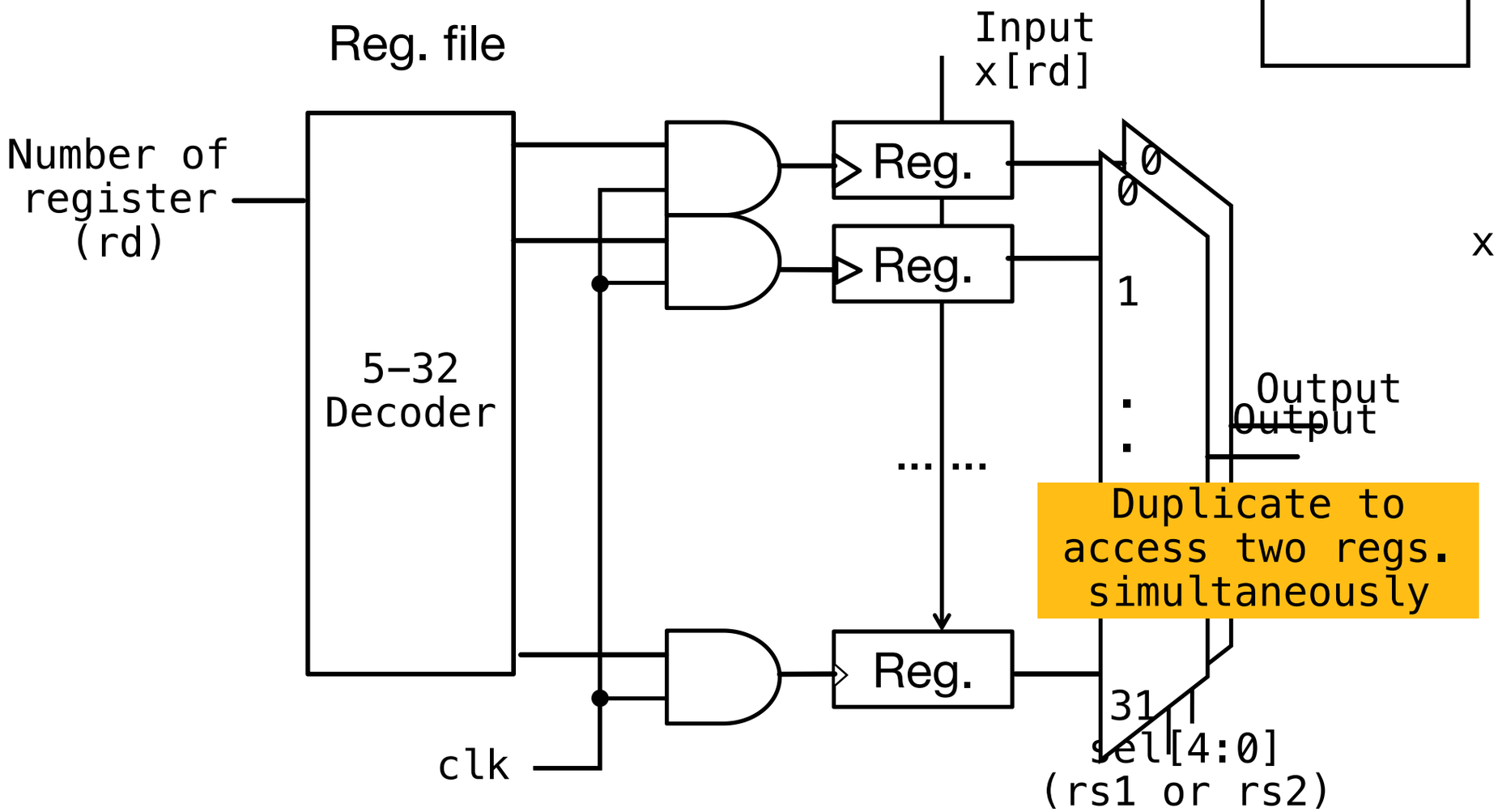
Datapath



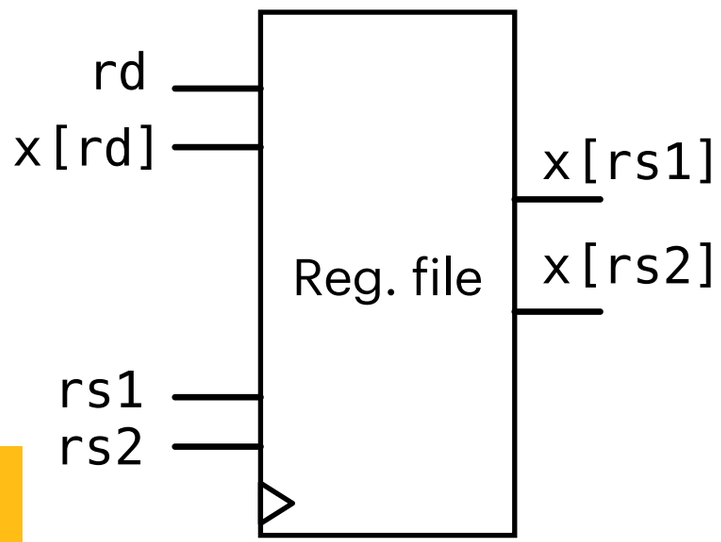
DRAM memory symbol



Reg. file

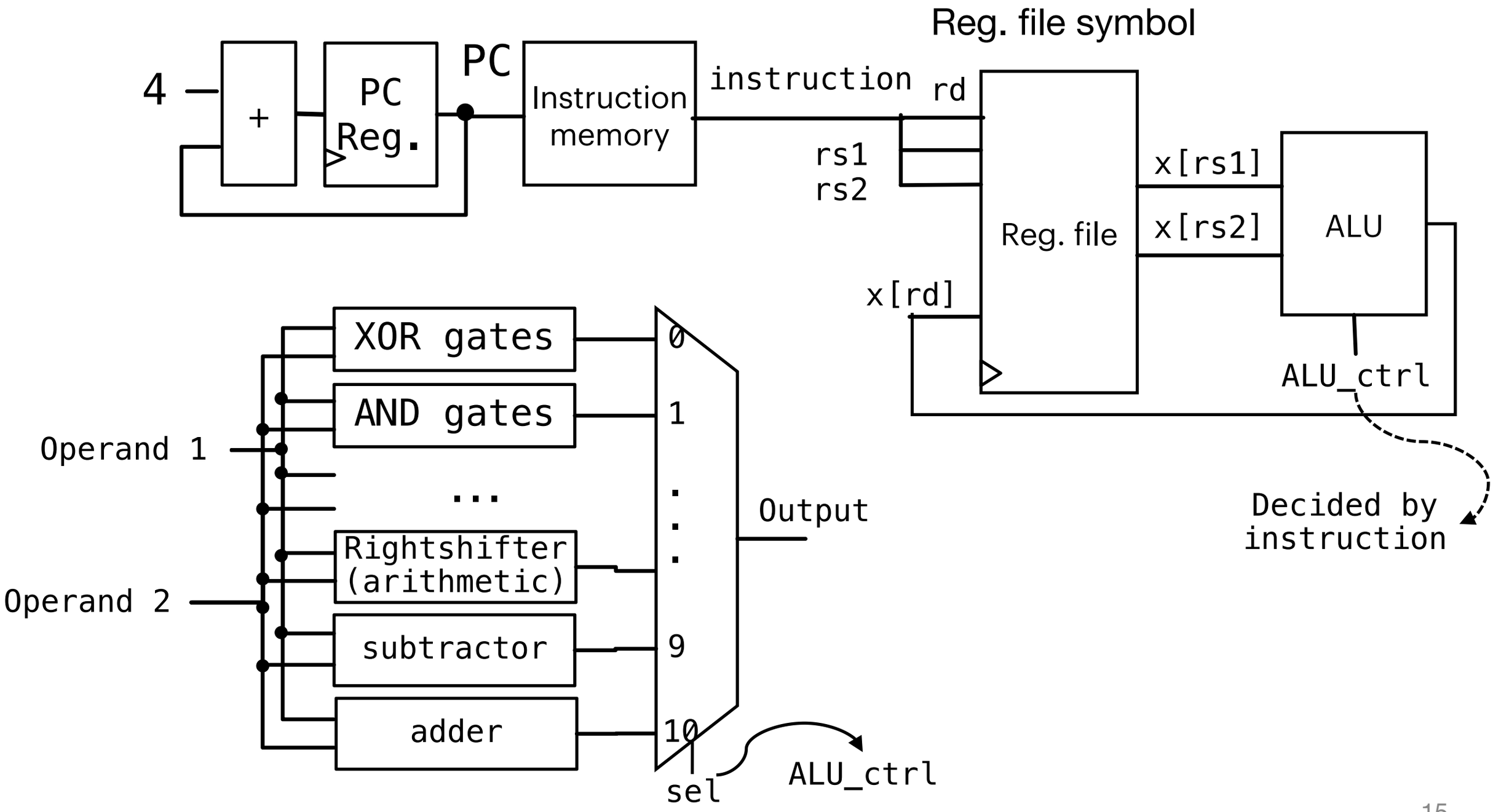


Reg. file symbol



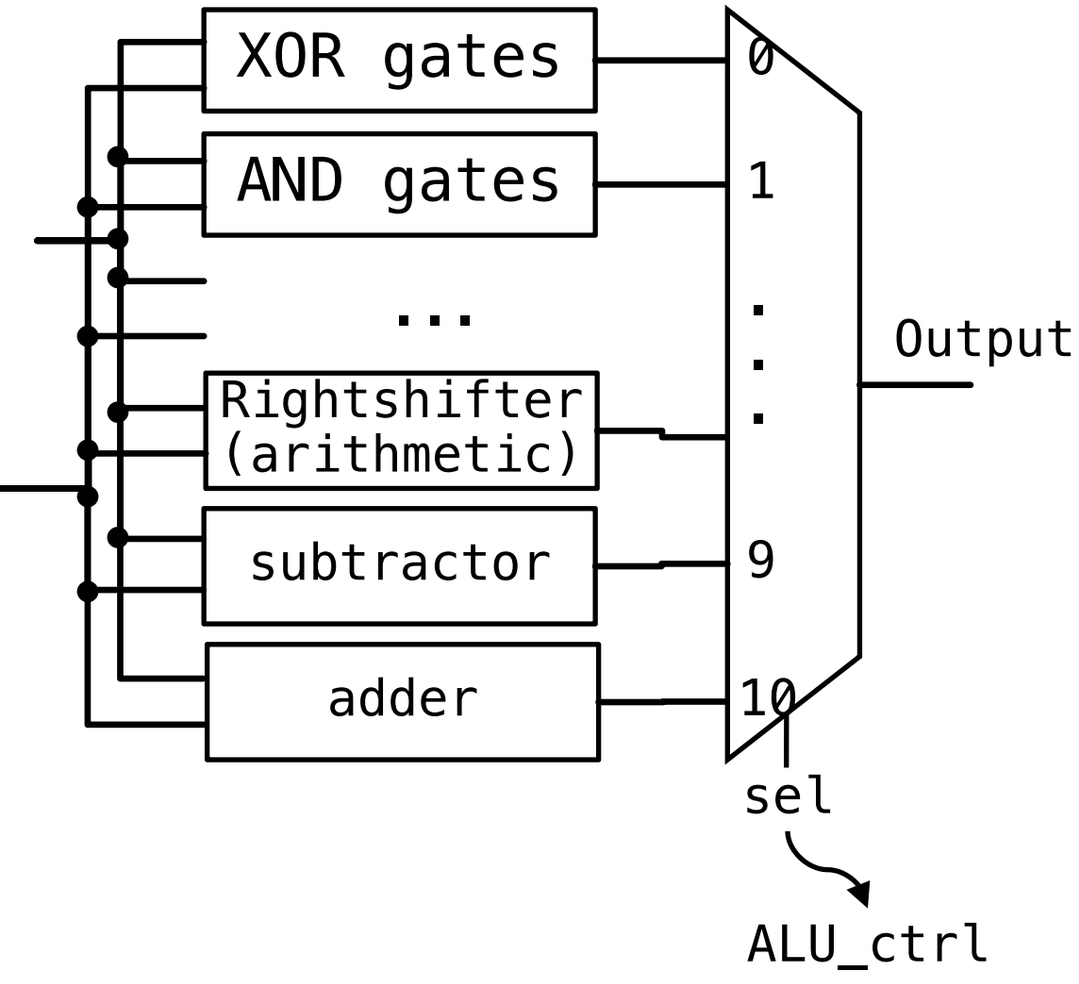
Datapath for R-type

- We have all the building blocks to execute R-type instructions



Datapath for R-type

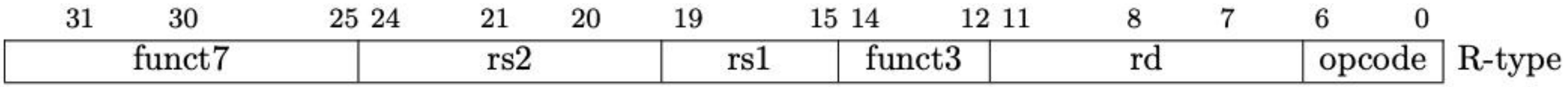
- We have all the building blocks to execute R-type instructions

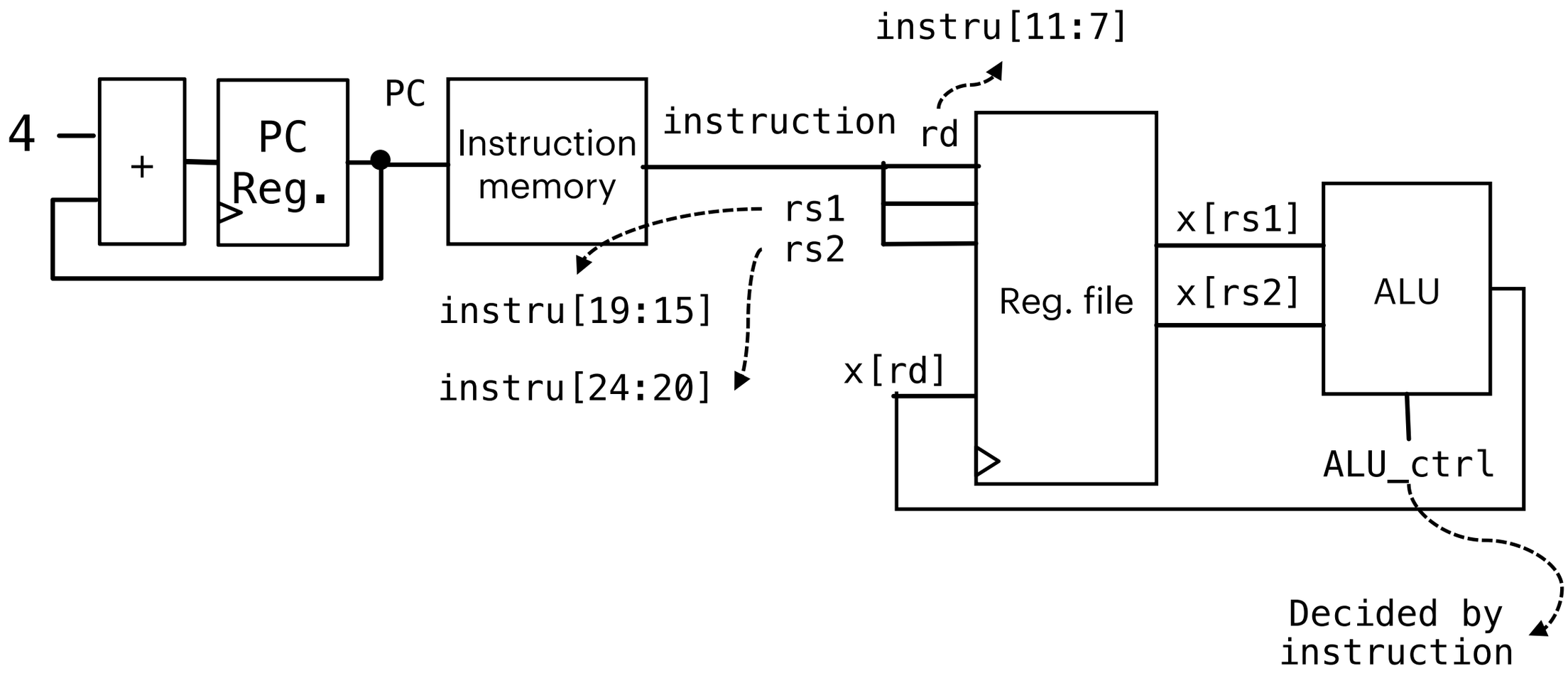


0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

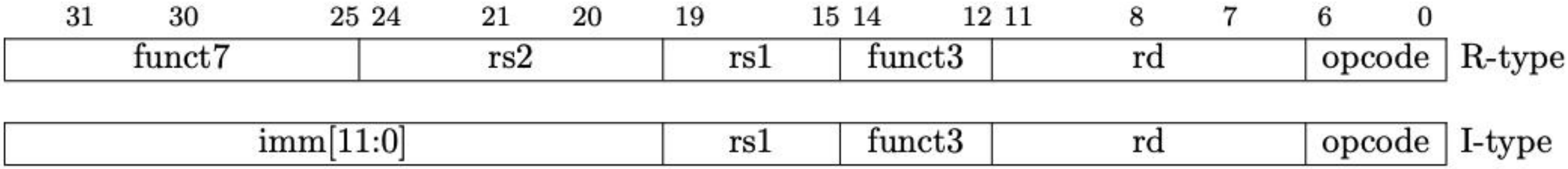
Datapath for R-type

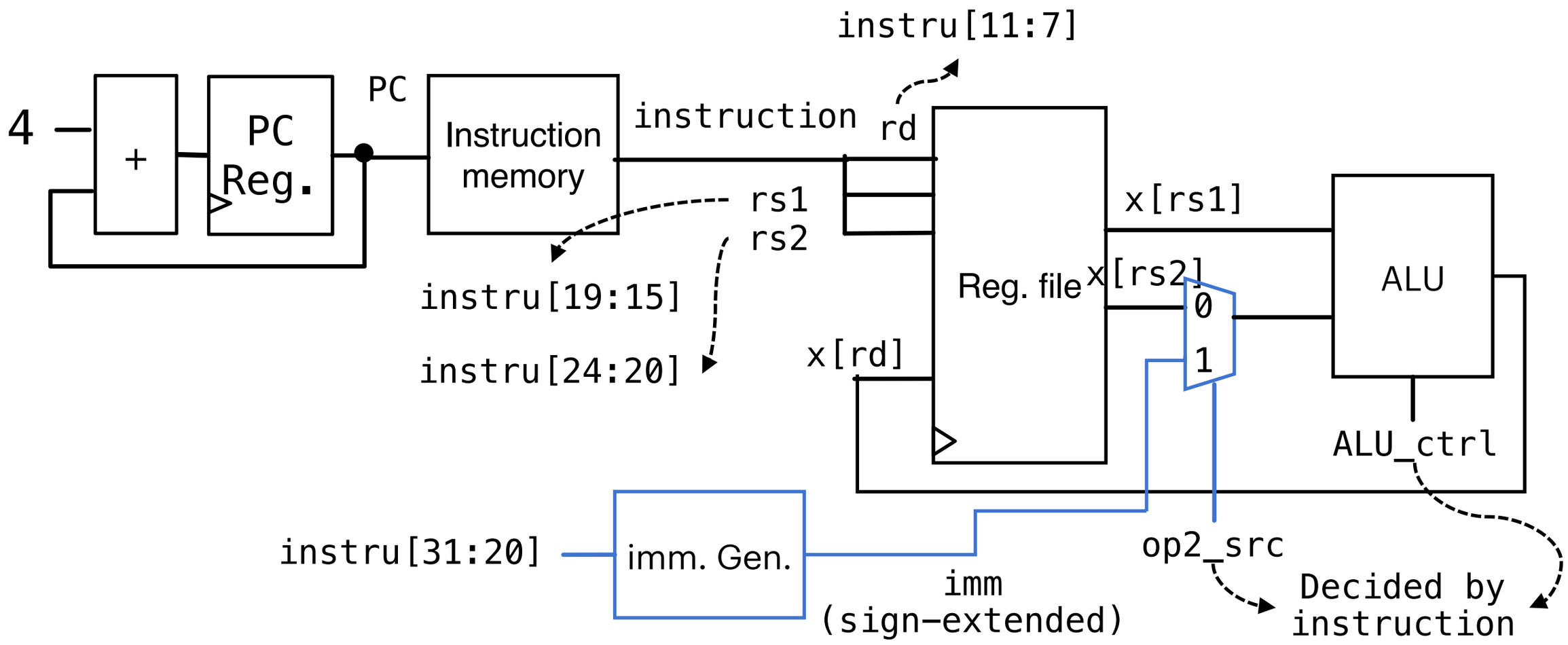
- We have all the building blocks to execute R-type instructions



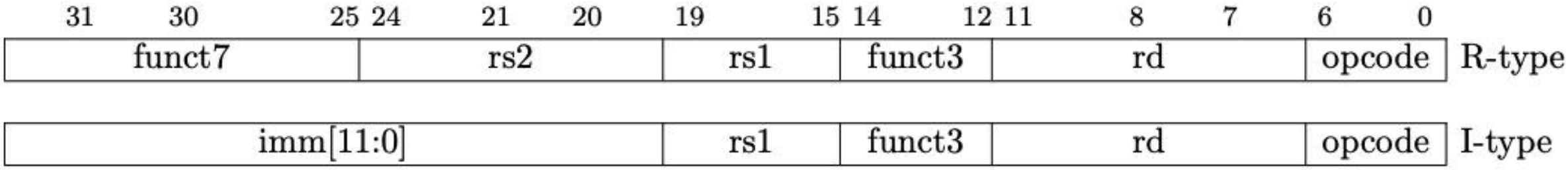


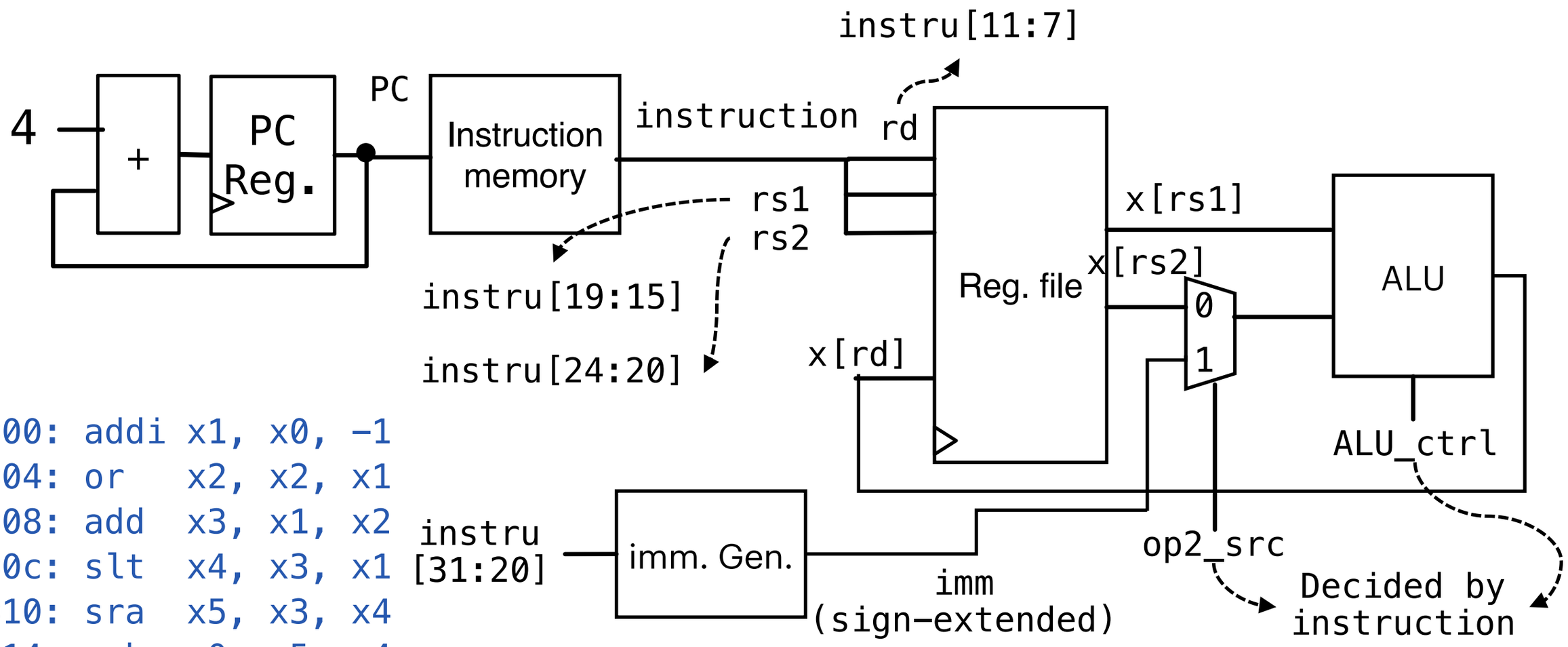
Datapath for I-type arithmetic and logic





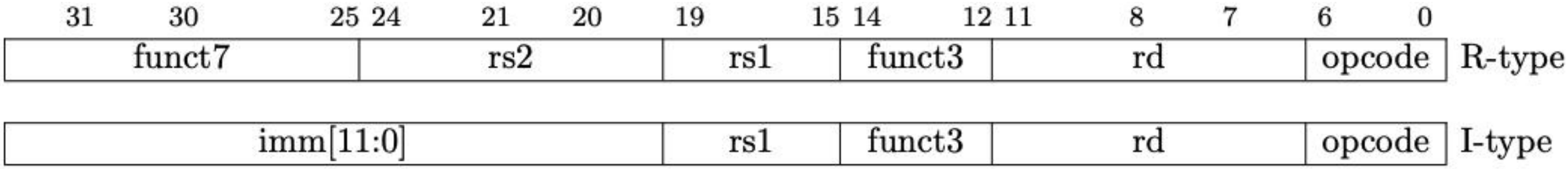
Example





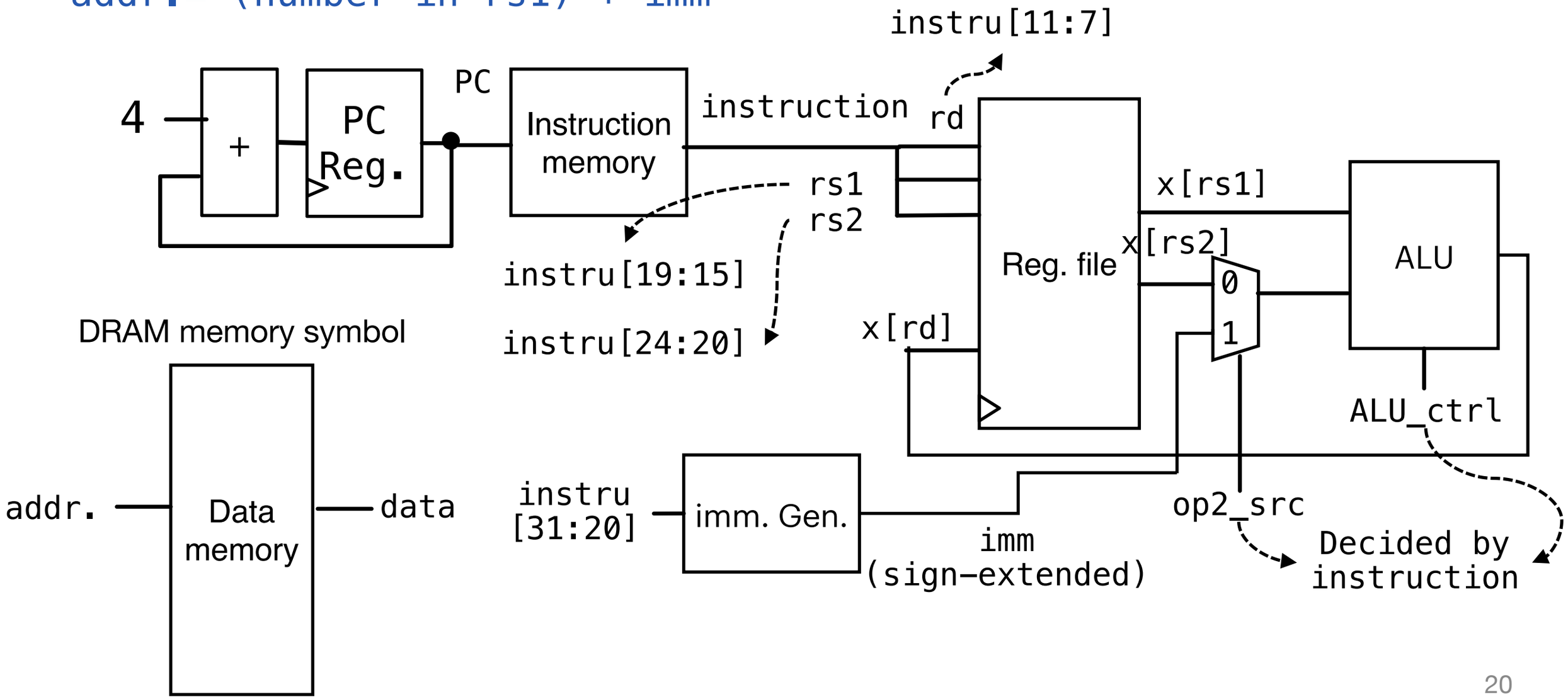

```
0x1000: addi x1, x0, -1
0x1004: or   x2, x2, x1
0x1008: add  x3, x1, x2
0x100c: slt  x4, x3, x1
0x1010: sra  x5, x3, x4
0x1014: sub  x0, x5, x4
```

Datapath for more types ...

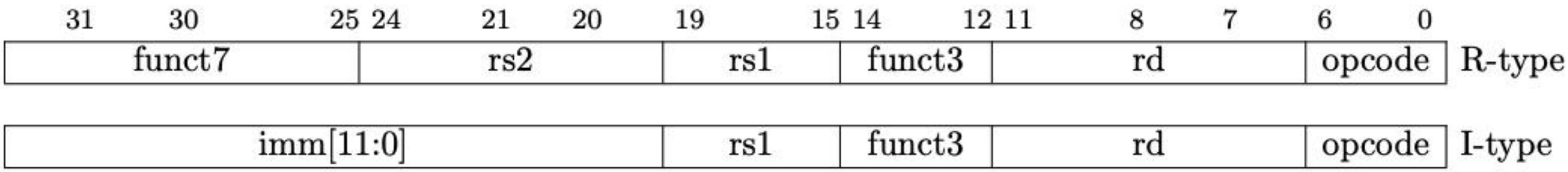


- `lw rd, imm(rs1)` : Load word at `addr.` to register `rd`

`addr.= (number in rs1) + imm`

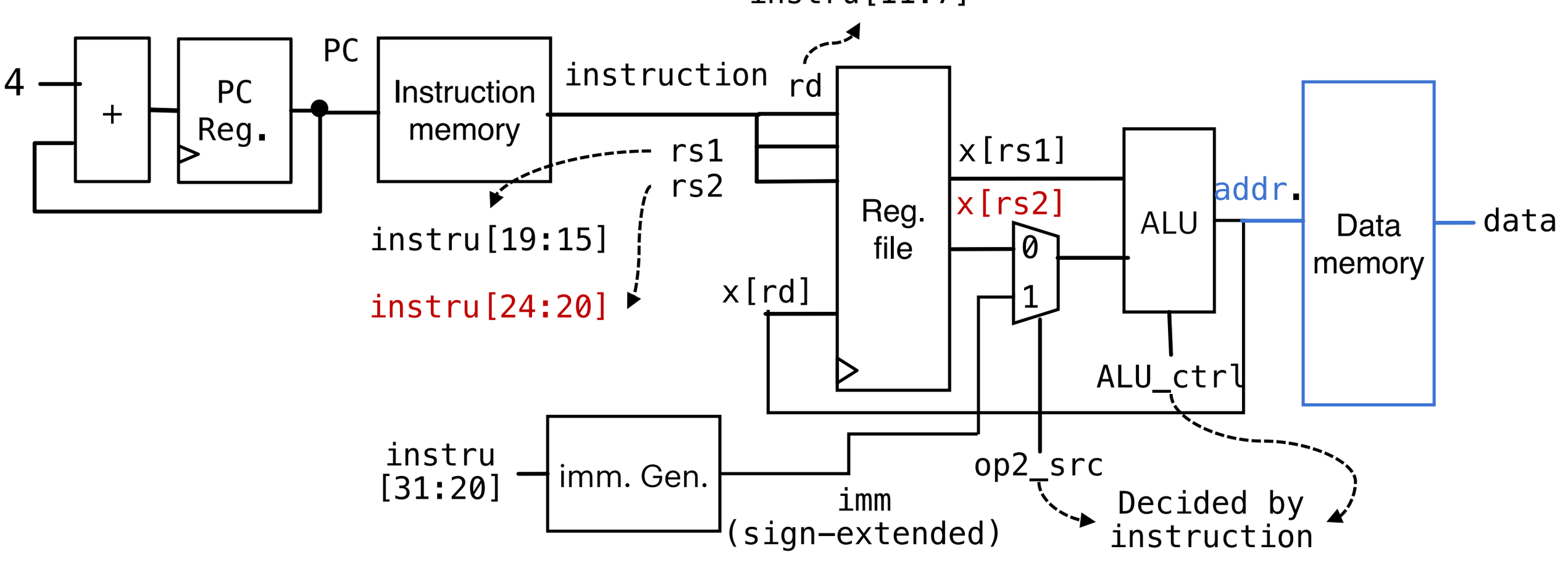


Datapath for I-type load

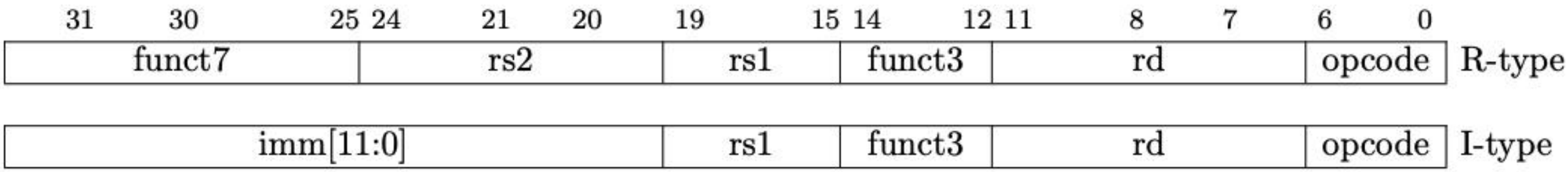


- `lw rd, imm(rs1)` : Load word at `addr.` to register `rd`

`addr.= (number in rs1) + imm`

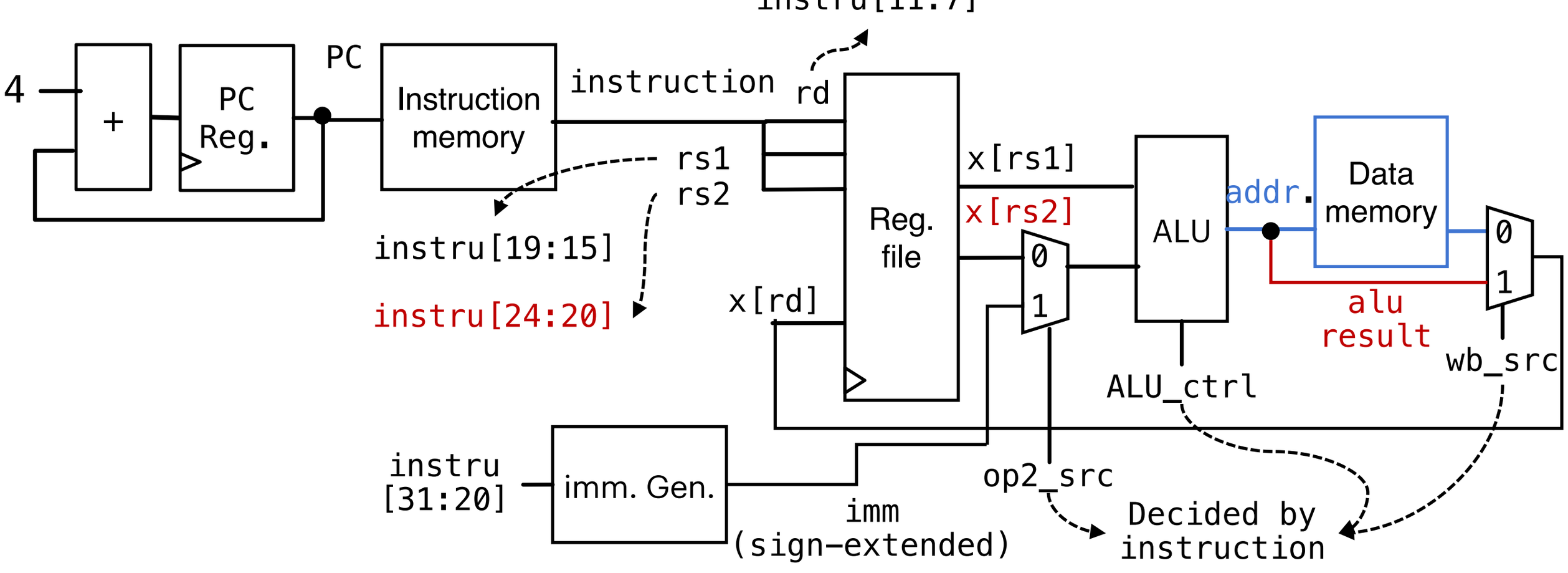


Datapath for I-type load

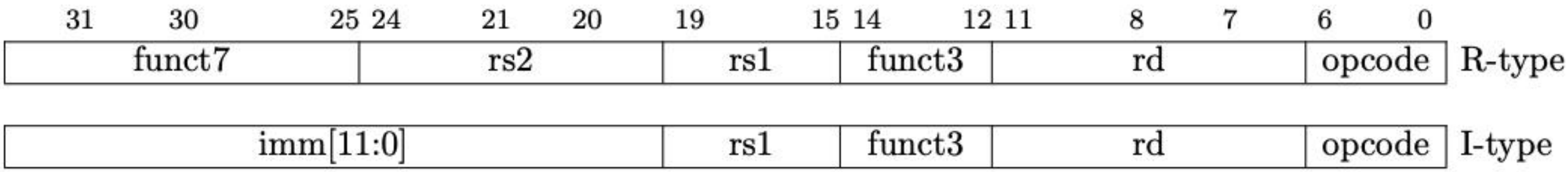


- `lw rd, imm(rs1)` : Load word at `addr.` to register `rd`

`addr.= (number in rs1) + imm`



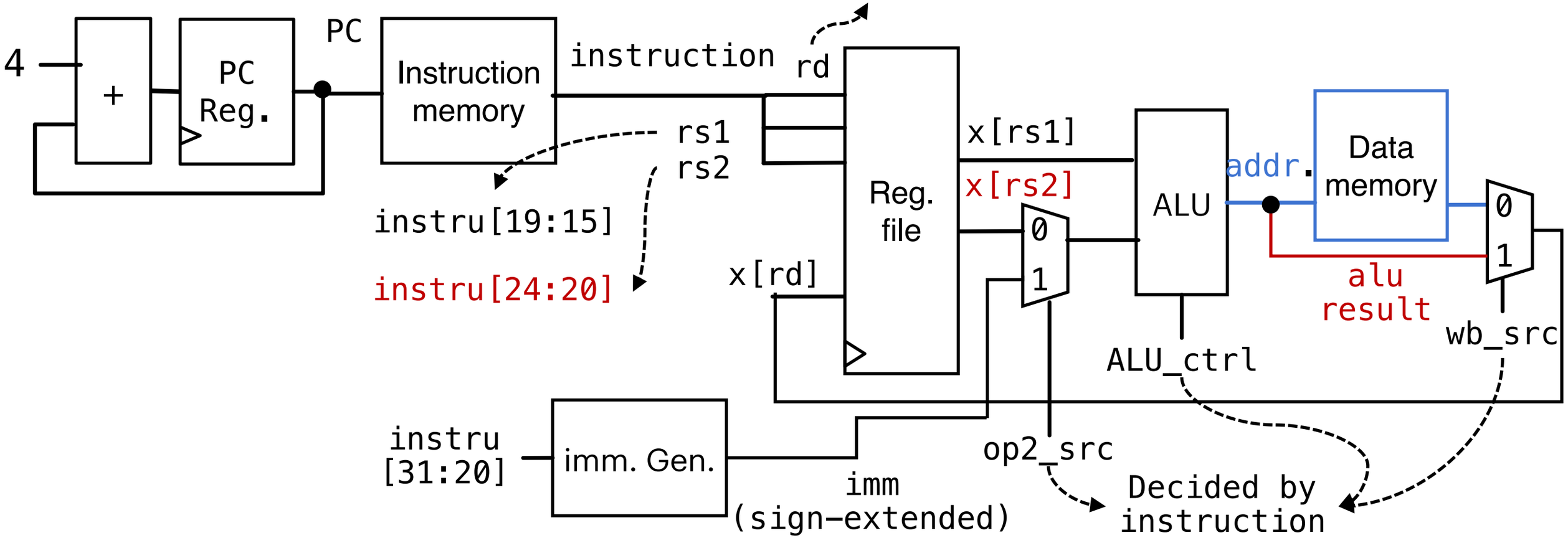
Datapath for I-type load



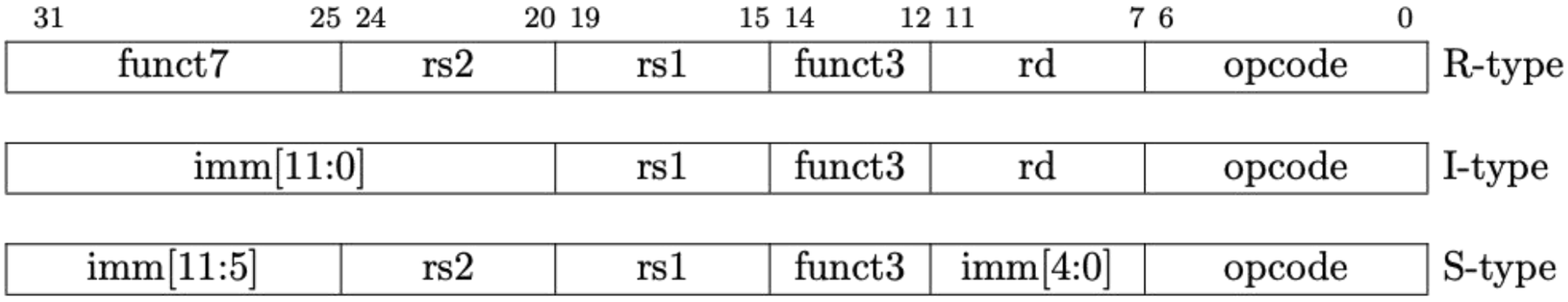
- `lw rd, imm(rs1)` : Load word at `addr.` to register `rd`

`addr.= (number in rs1) + imm`

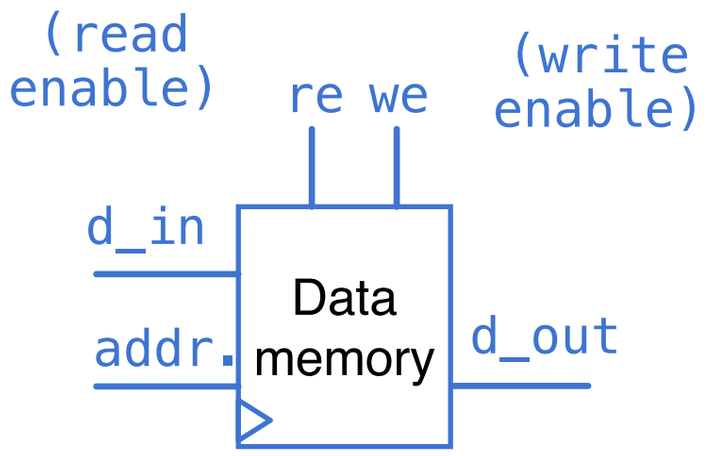
How would you deal with lb(u)/lh(u)?



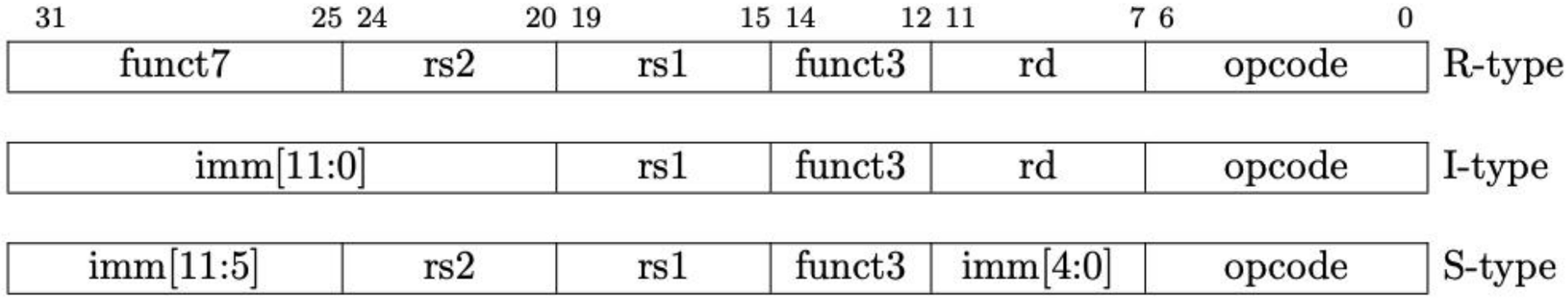
Datapath for S-type store



- Recall that in an FSM, only when there is a trigger (clk edge), the state can change.
- We assume that the change of data memory (memory-write) is also governed by clk edge.
- Assume behavior model of data memory:
  - When we=1 && re=0, at clk rising edge, data[addr.]=d_in; d_out stays at high-resistance (output nothing)
  - When we=re=0, d_out stay at high-resistance (output nothing, state would not change); we=re=1 is forbidden
  - When we=0 && re=1, d_out=data[addr.]

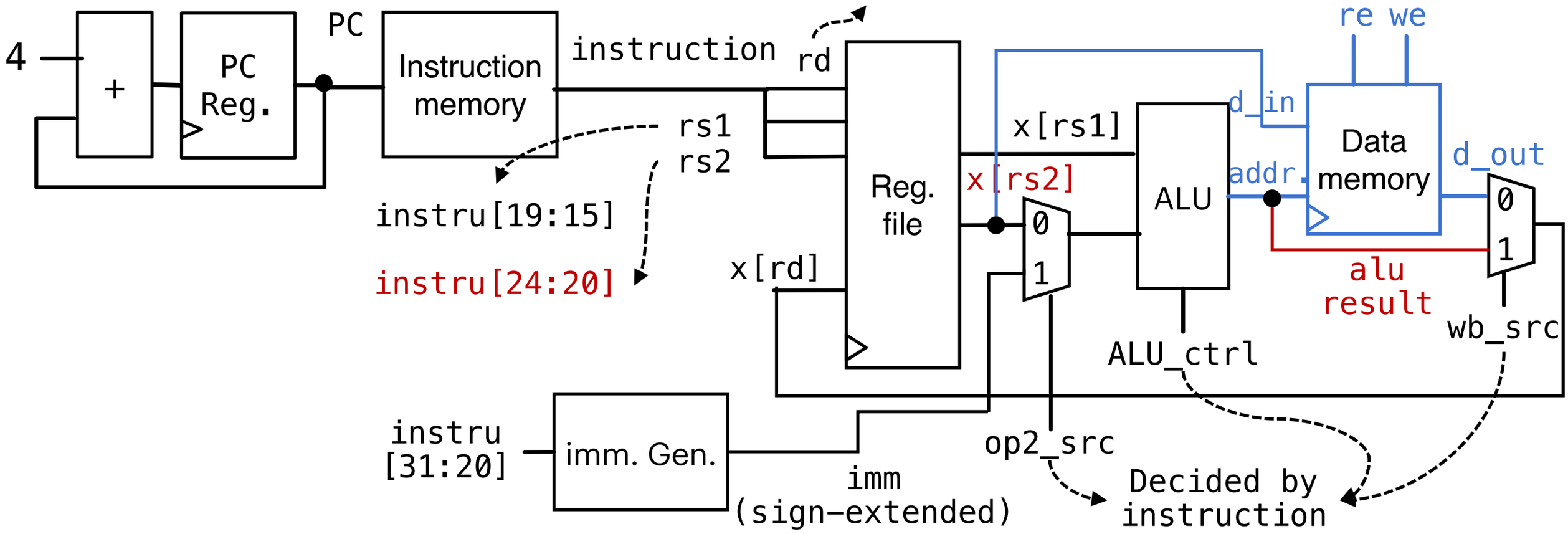


Datapath for S-type store

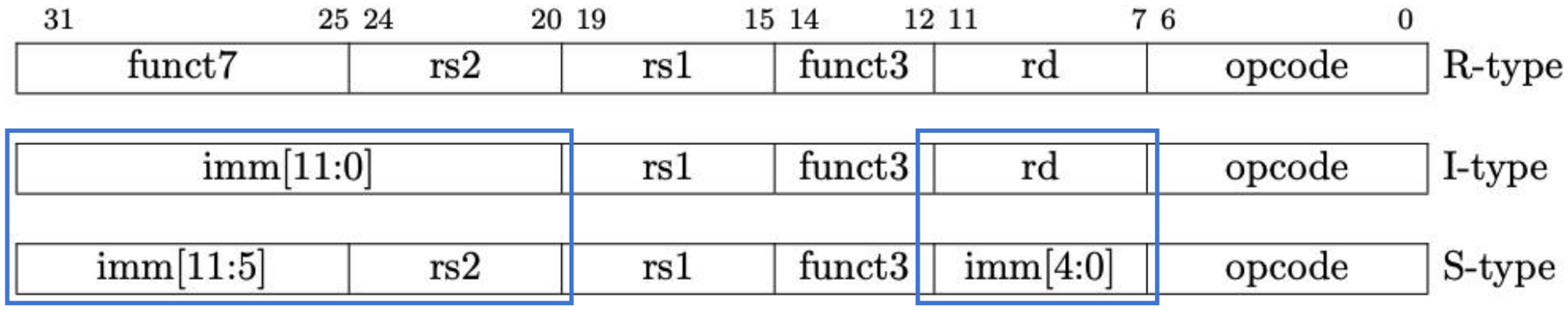


- `sw rs2, imm(rs1)`: Store word at `rs2` to memory `addr`.

`addr.= (number in rs1) + imm`

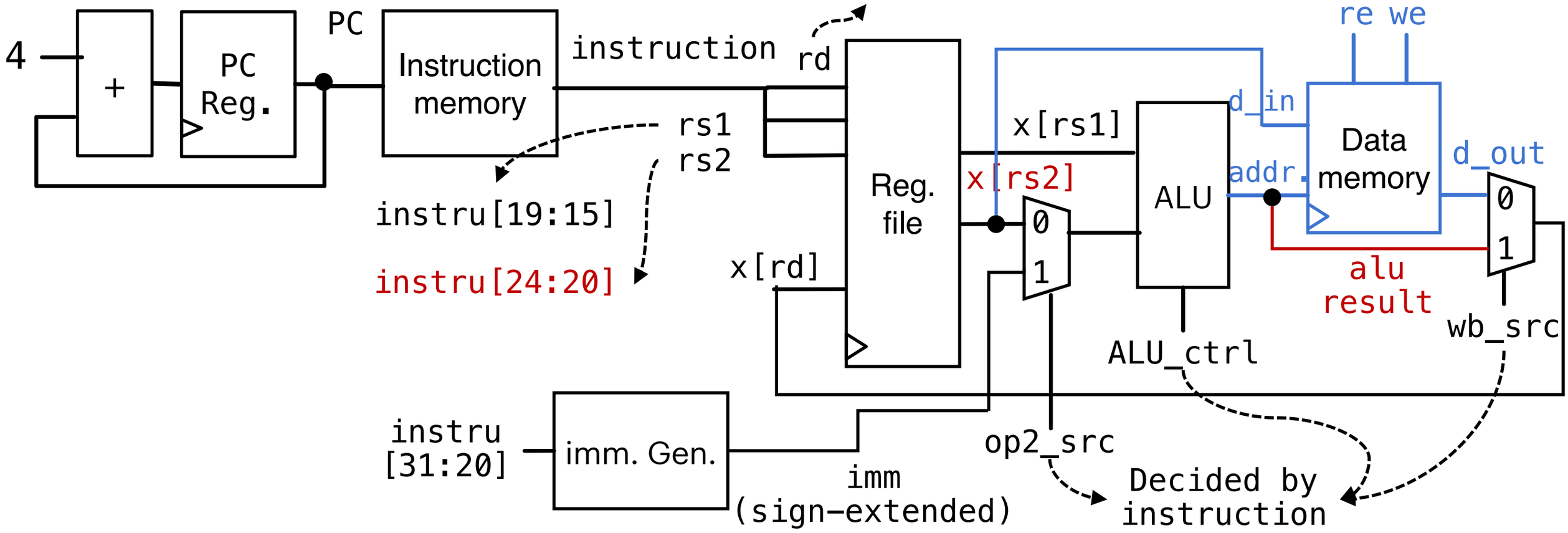


Datapath for S-type store

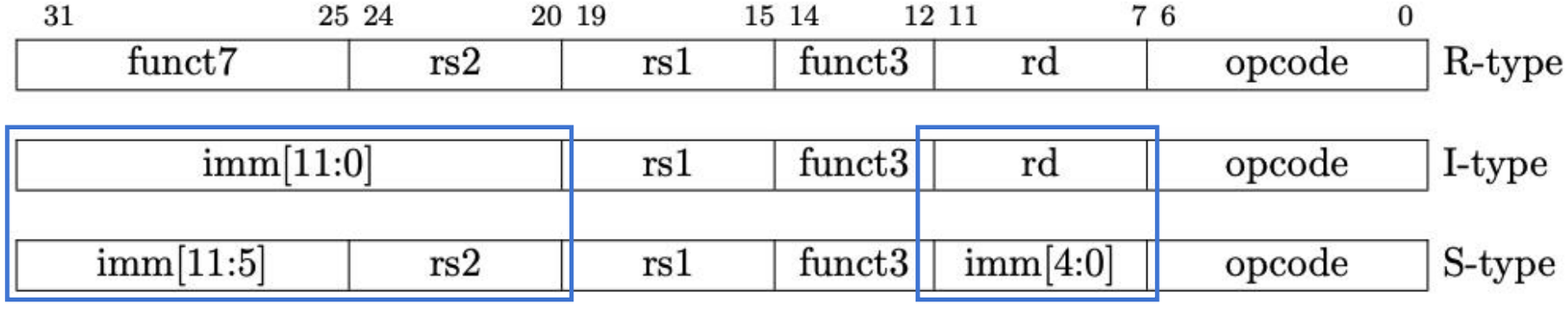


- `sw rs2, imm(rs1)`: Store word at `rs2` to memory `addr.`

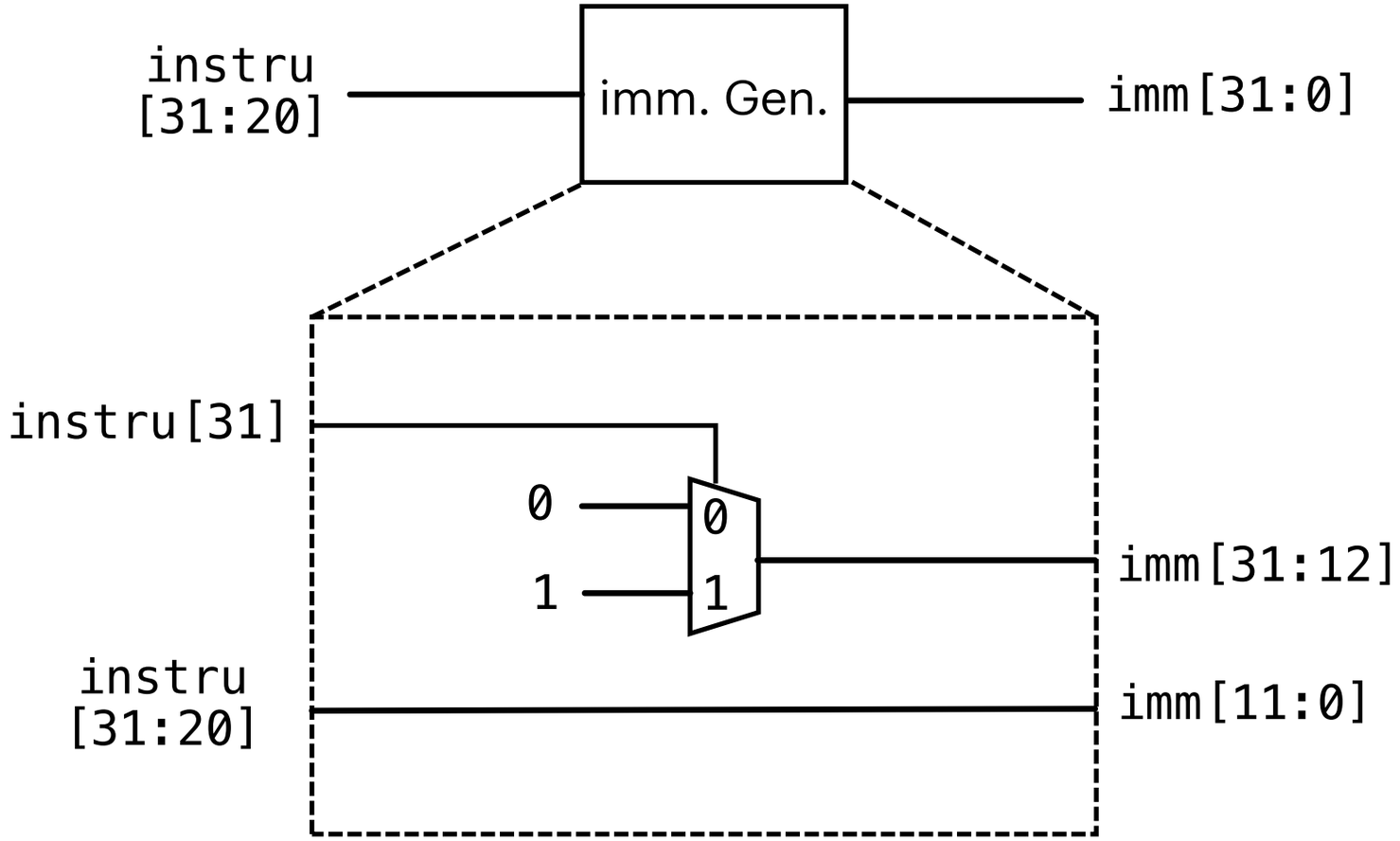
`addr.= (number in rs1) + imm`



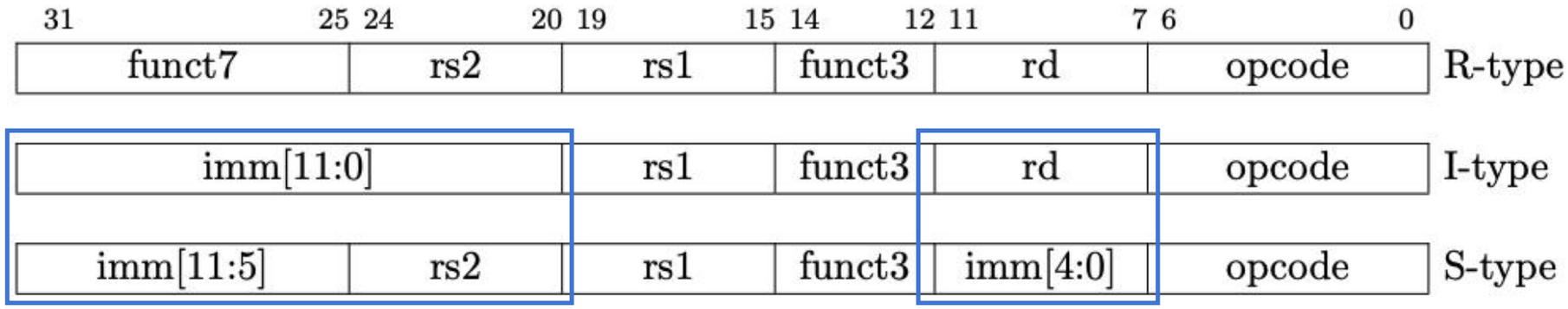
Imm. Gen.



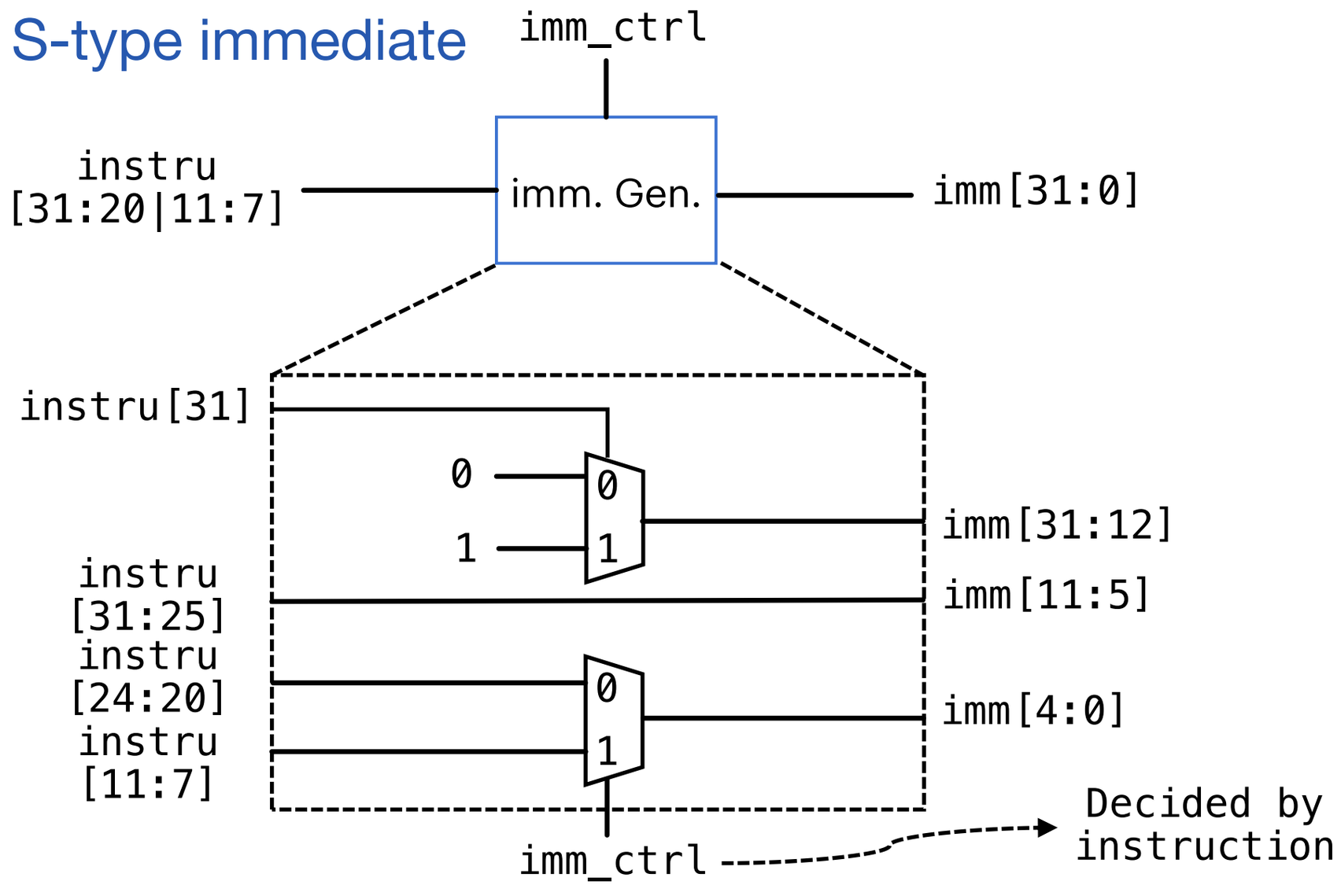
- I-type immediate



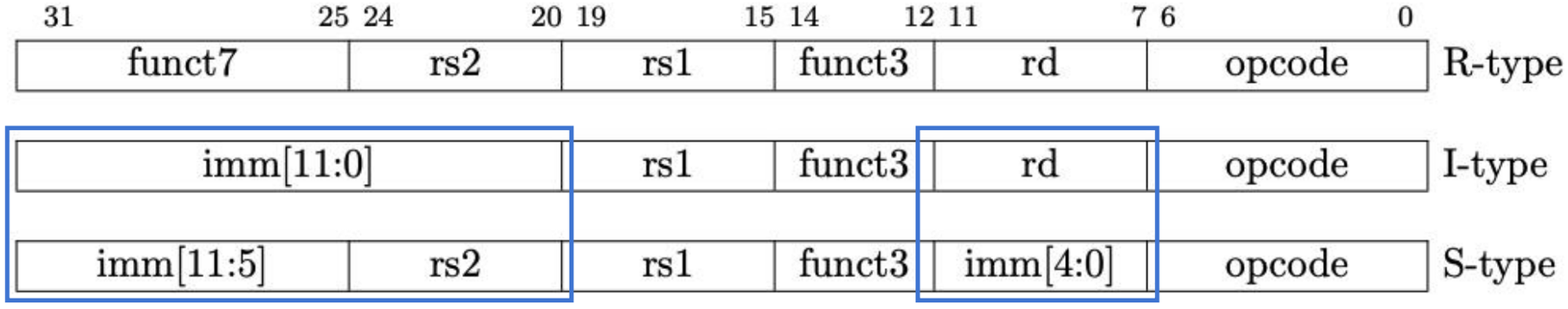
Imm. Gen.



- I-type & S-type immediate



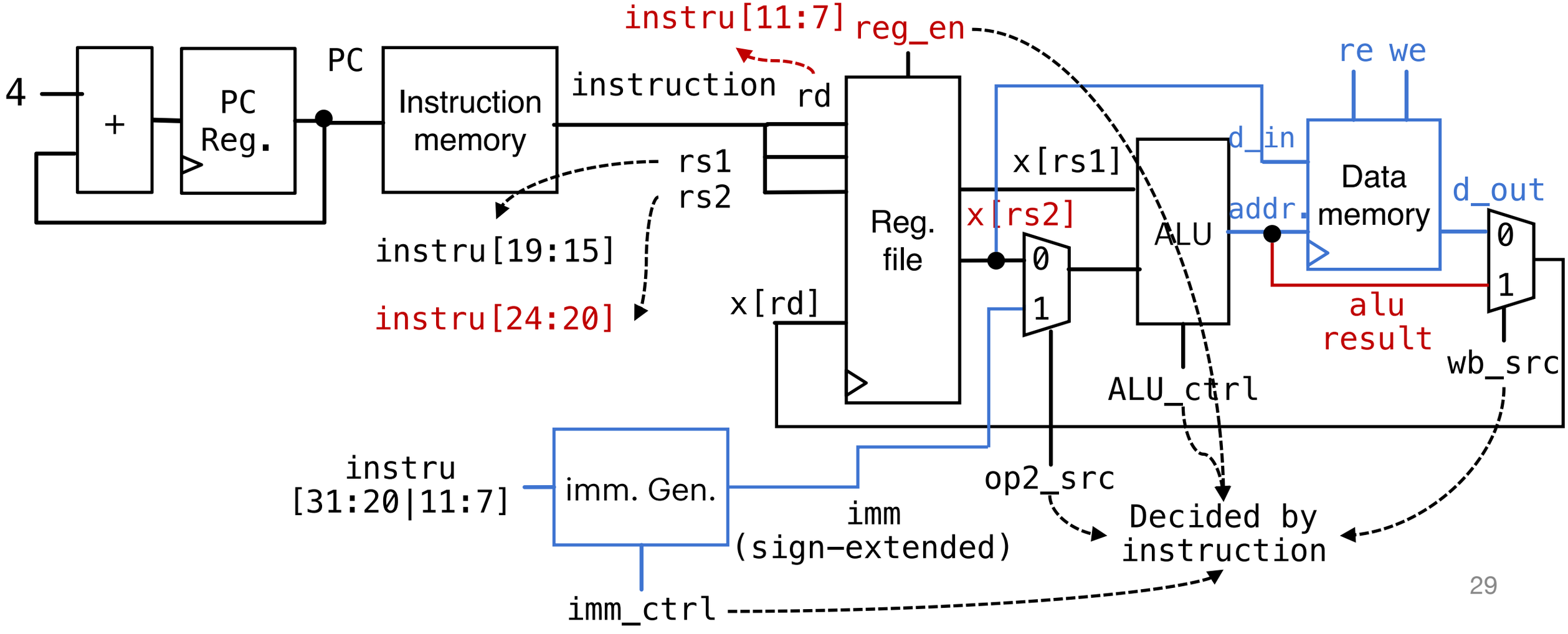
Datapath for S-type store



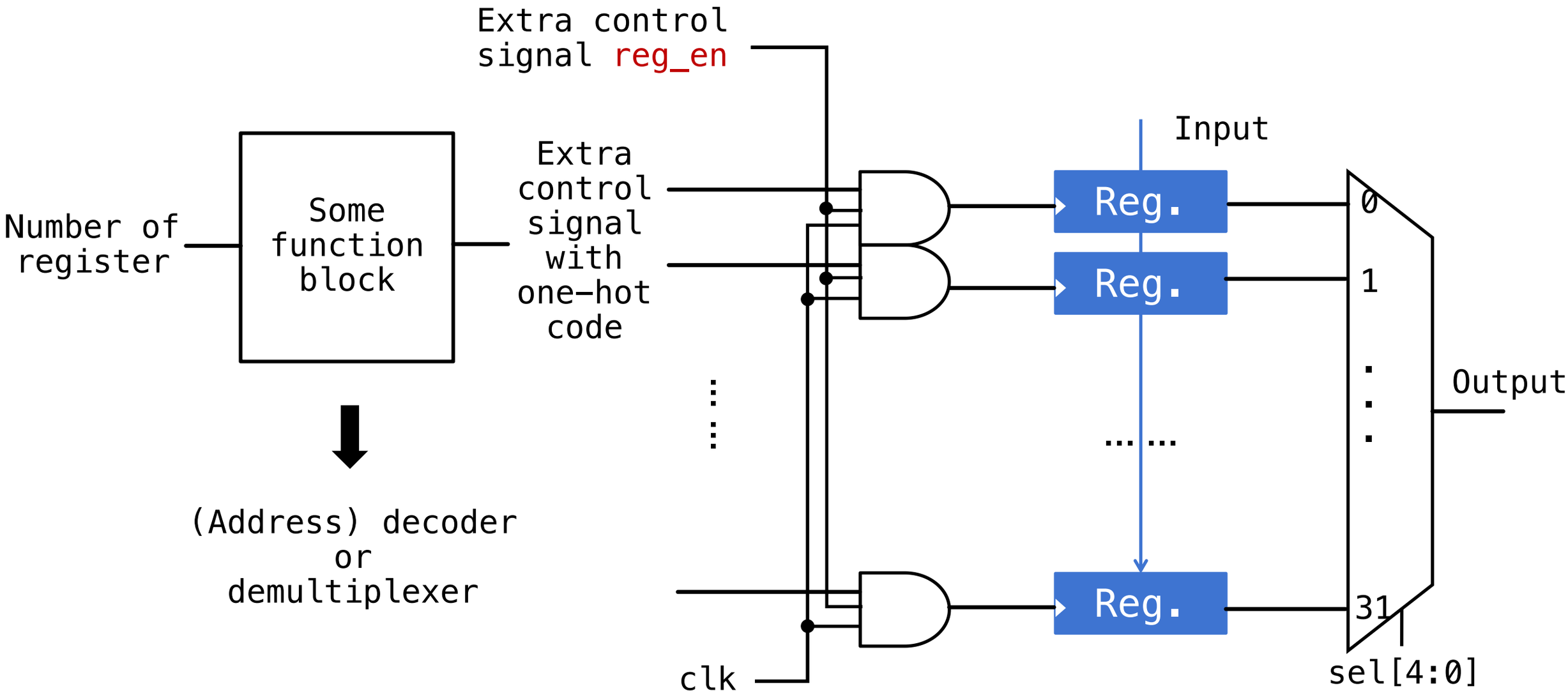
- `sw rs2, imm(rs1)`: Store word at `rs2` to memory `addr.`

`addr.= (number in rs1) + imm`

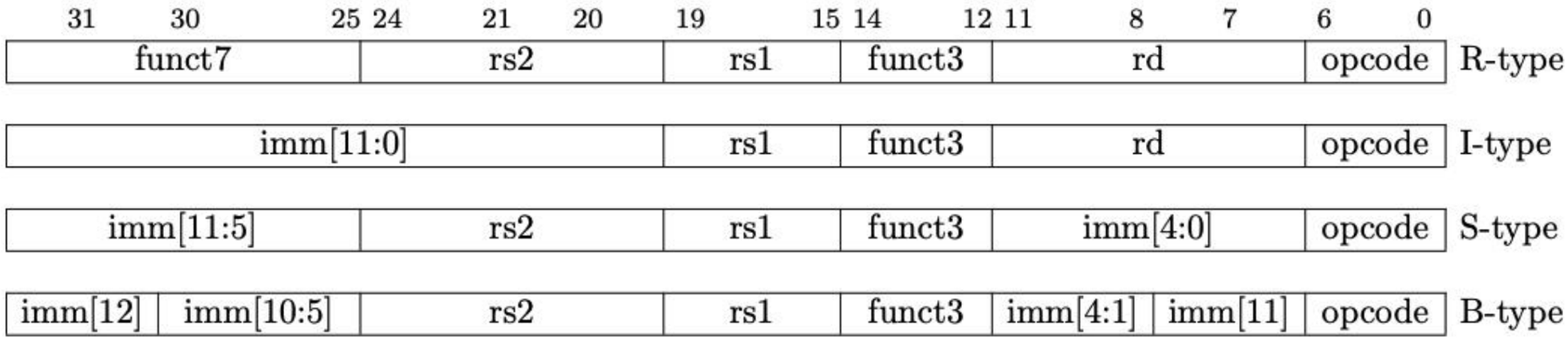
Something wrong!



Regfile modification

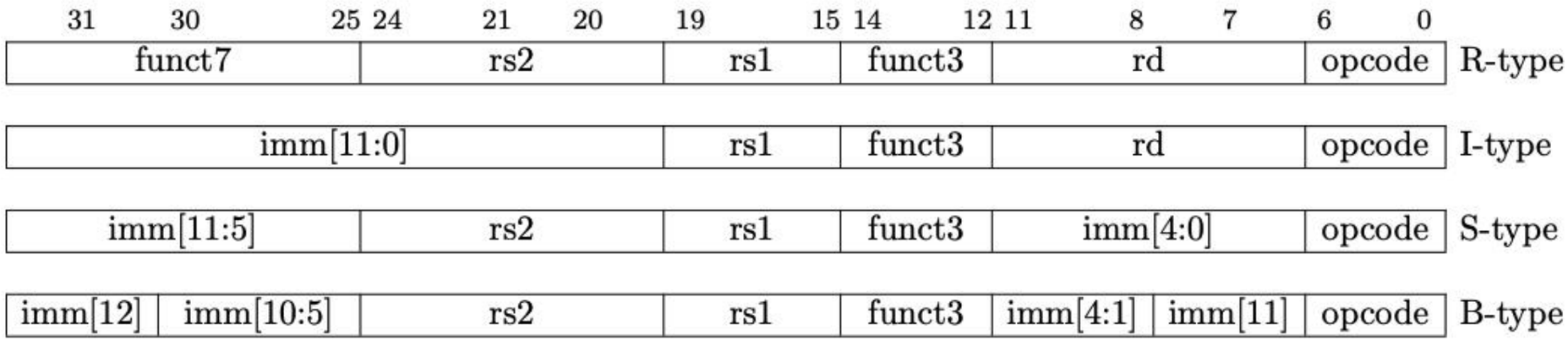


Datapath for B-type

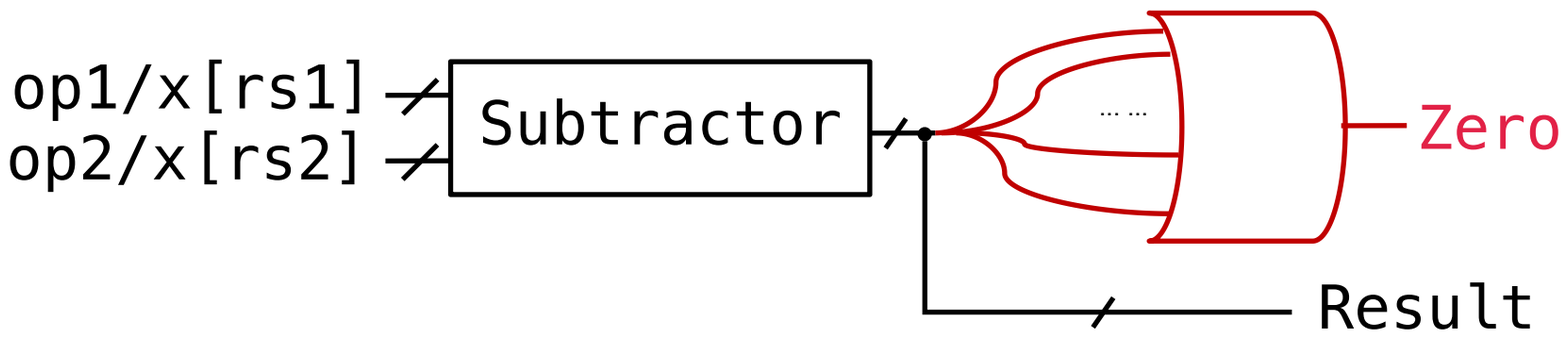


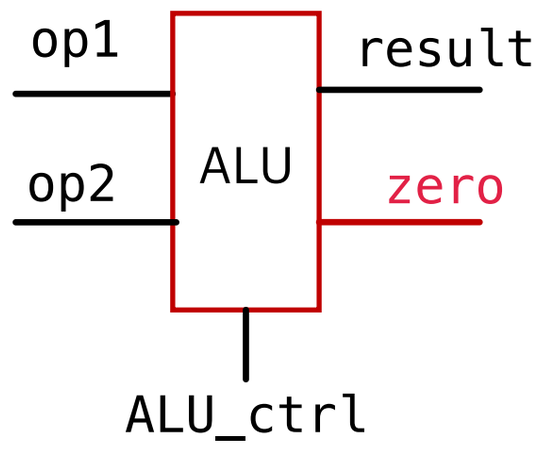
- `beq rs1,rs2,L(imm/label)`
- Go to `label` if `x[rs1] == x[rs2]`; otherwise, go to next statement

Datapath for B-type



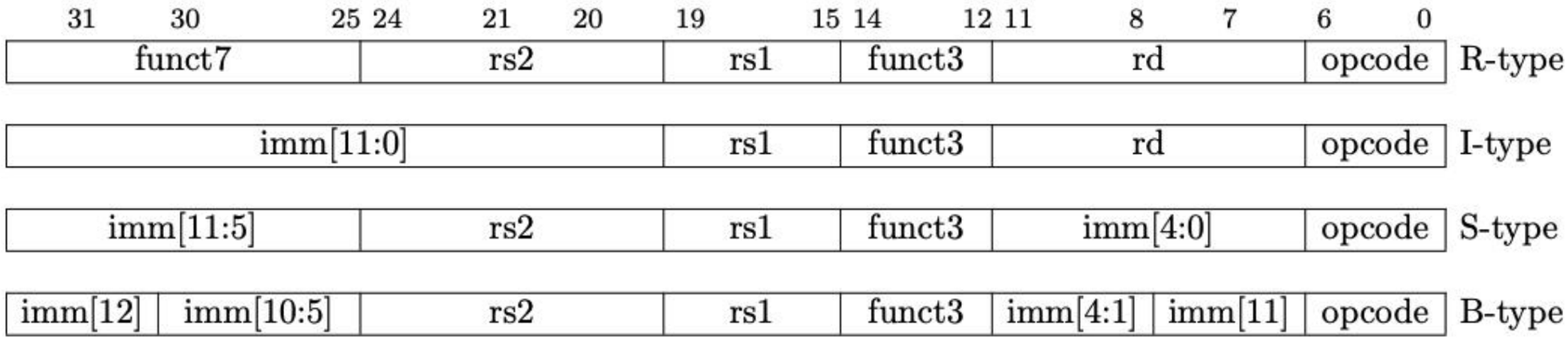
- `beq rs1,rs2,L(imm/label)`
- Go to `label` if `x[rs1]==x[rs2]`; otherwise, go to next statement
- Recall in ALU



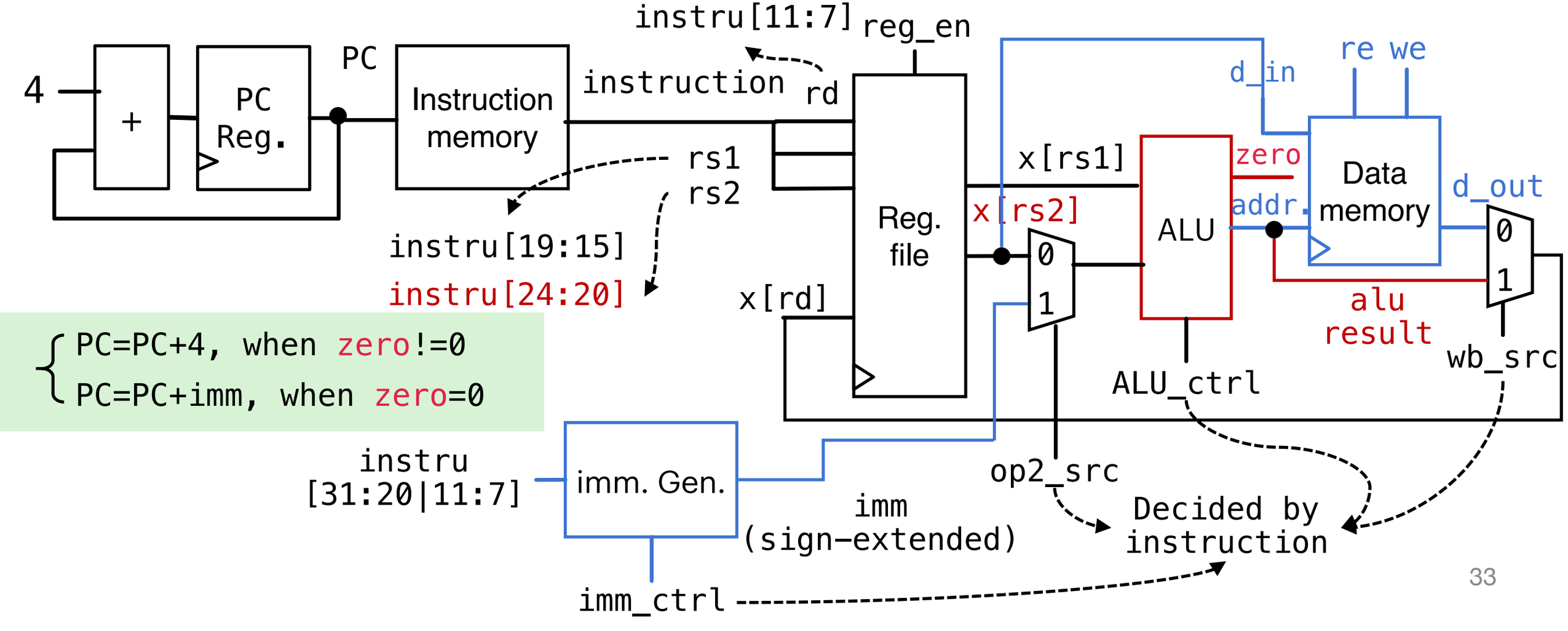


$x[rs1]==x[rs2] \leftrightarrow x[rs1]-x[rs2]==0$

Datapath for B-type



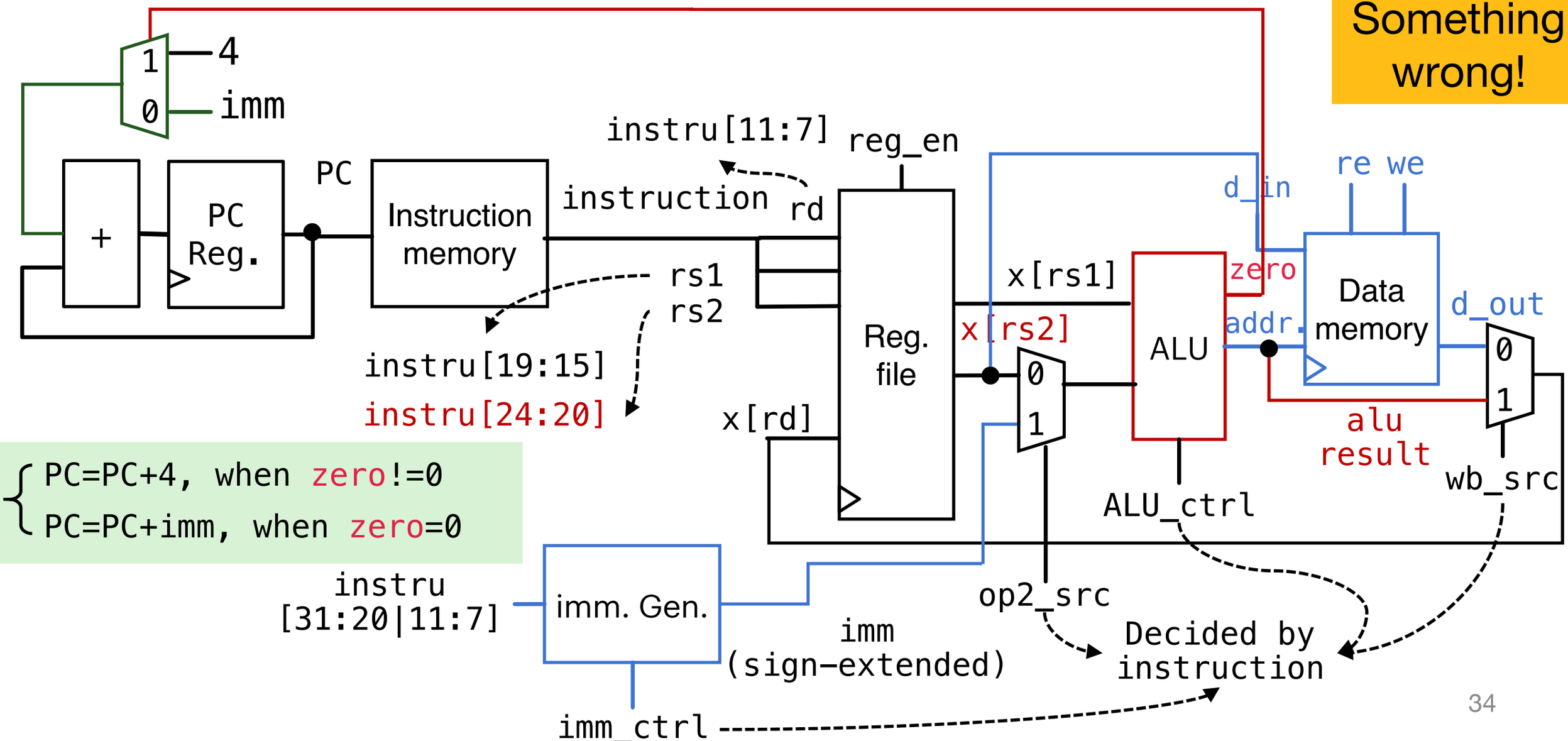
- `beq rs1,rs2,L(imm/label)`
- Go to `label` if x[rs1]==x[rs2]; otherwise, go to next instruction



Datapath for B-type

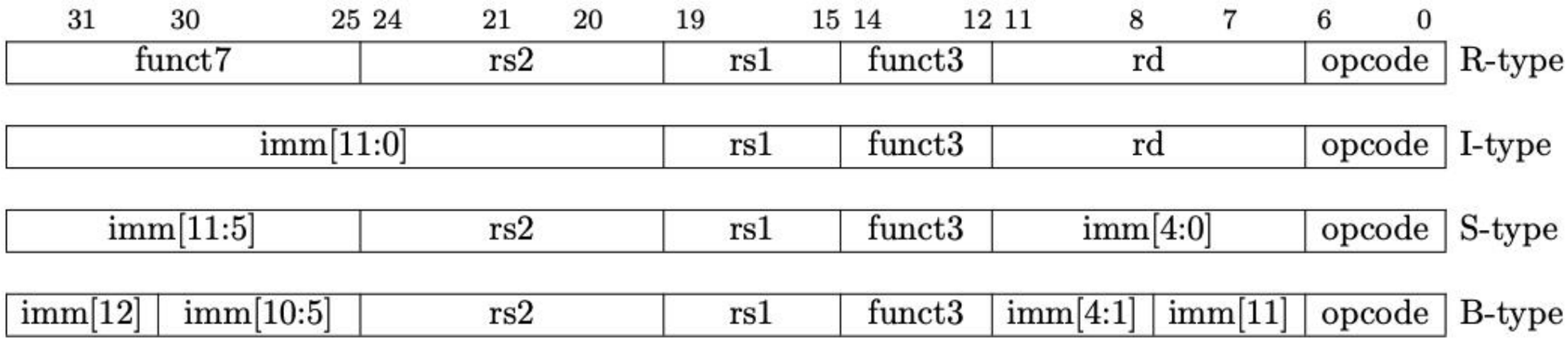
31 30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7	rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]			rs1	funct3	rd		opcode	I-type
imm[11:5]	rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

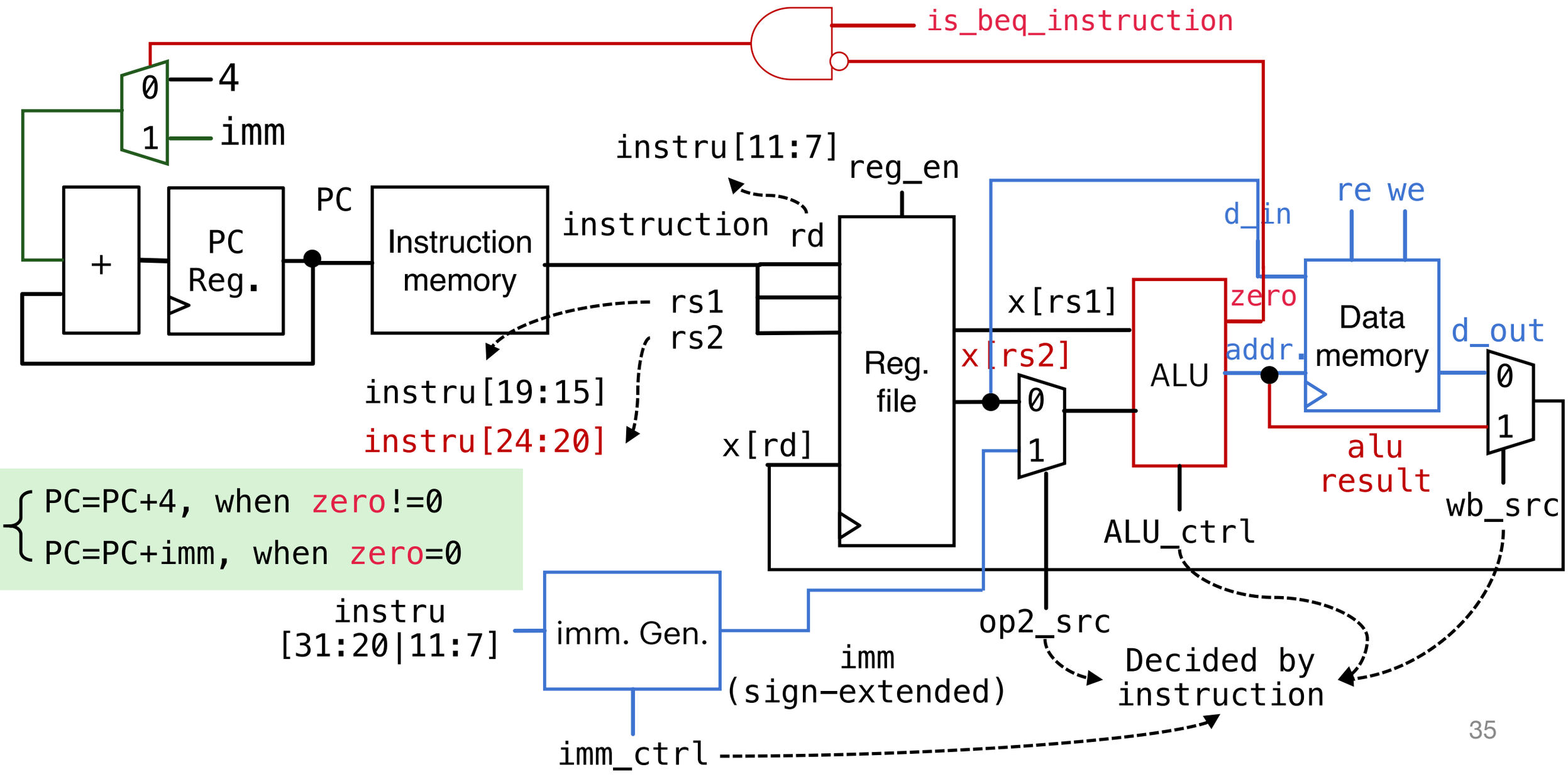
Something wrong!



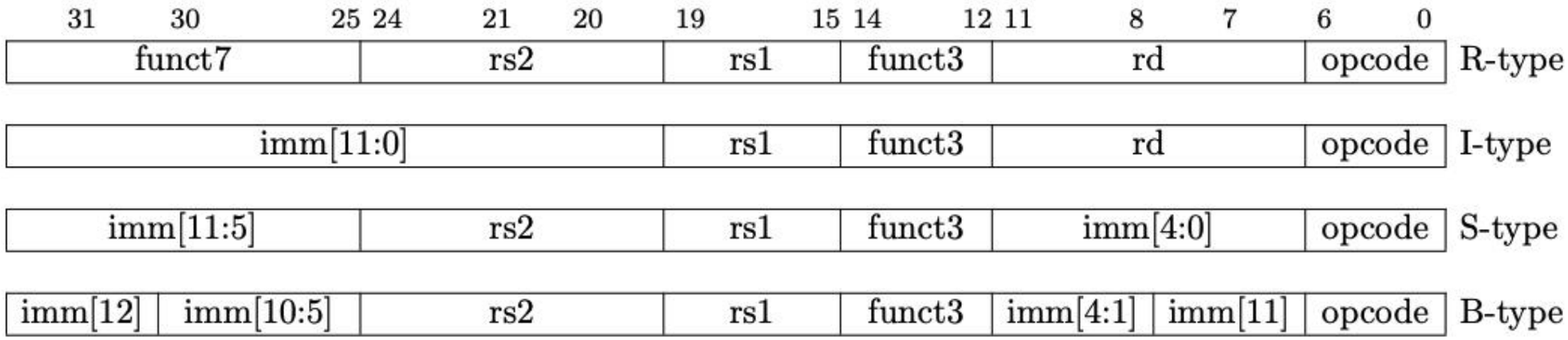
{ PC=PC+4, when zero!=0
  PC=PC+imm, when zero=0

Datapath for B-type





Datapath for B-type



`is_beq_instruction`

- Recall `beq`

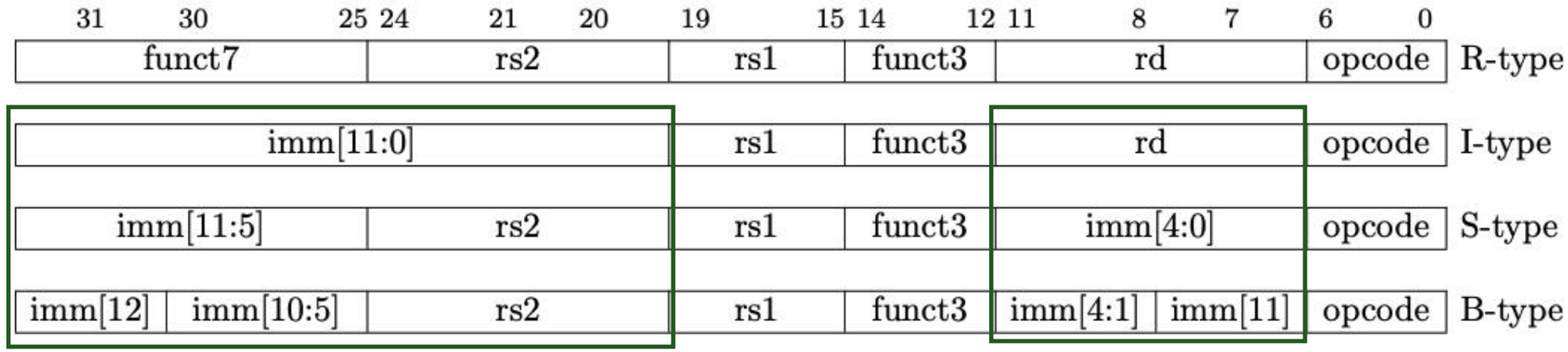
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

Truth table

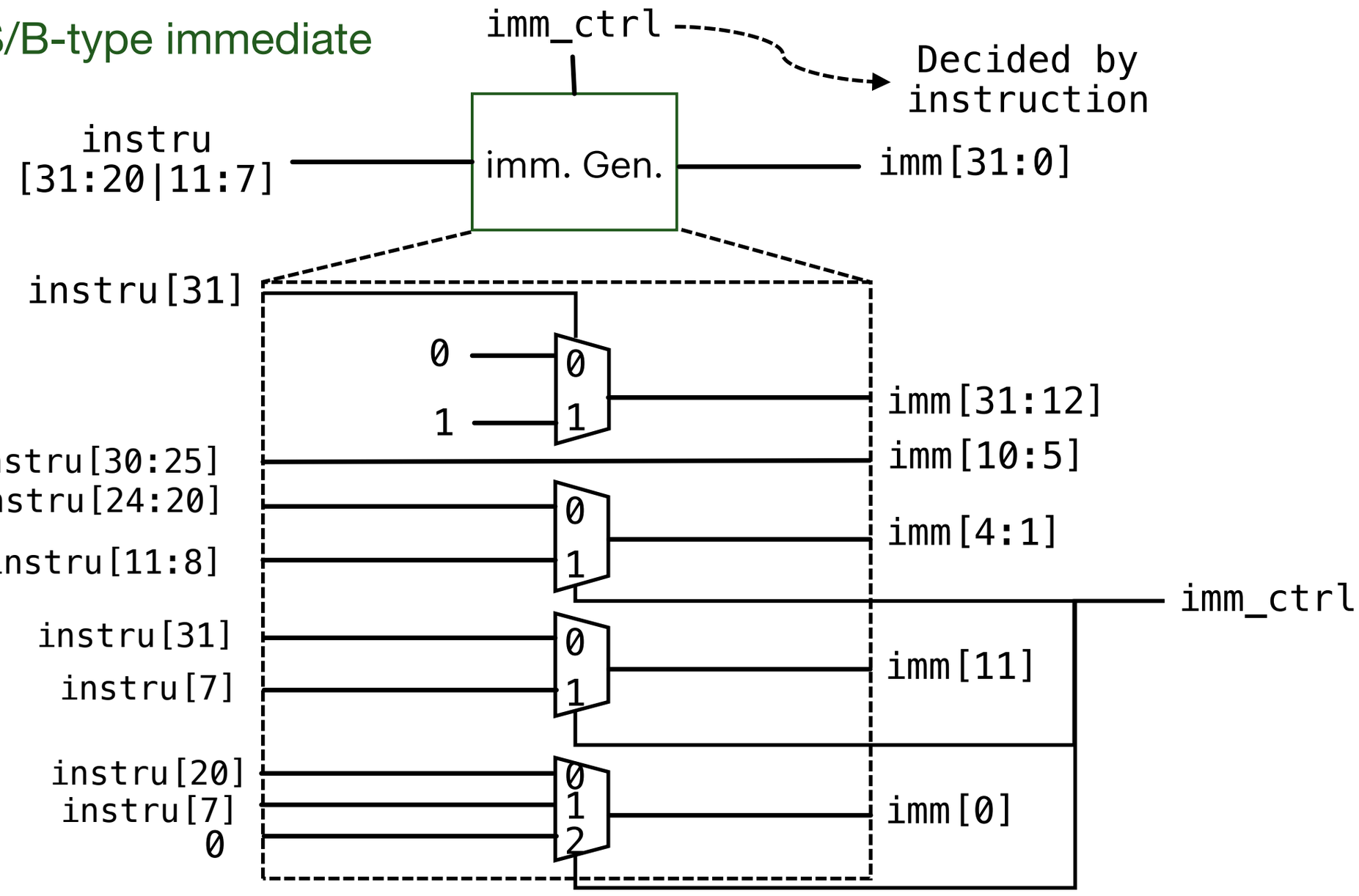
Instru[14:12|6:0]	is_beq_instruction
000 1100011	1
All the other cases	0

$$is\_beq\_instruction = \overline{i[14]}\,\overline{i[13]}\,\overline{i[12]}\,i[6]\,i[5]\,\overline{i[4]}\,\overline{i[3]}\,\overline{i[2]}\,i[1]\,i[0]$$

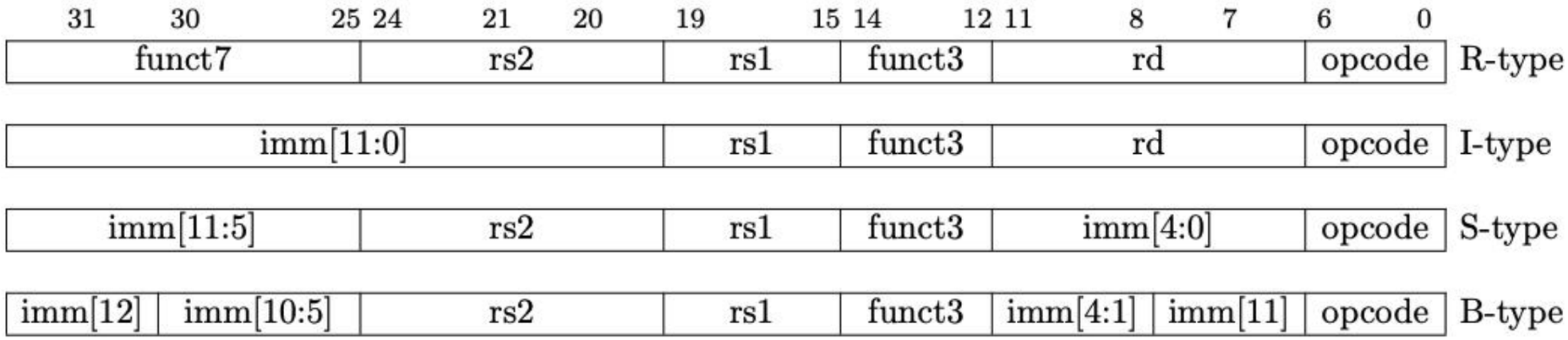
Datapath for B-type

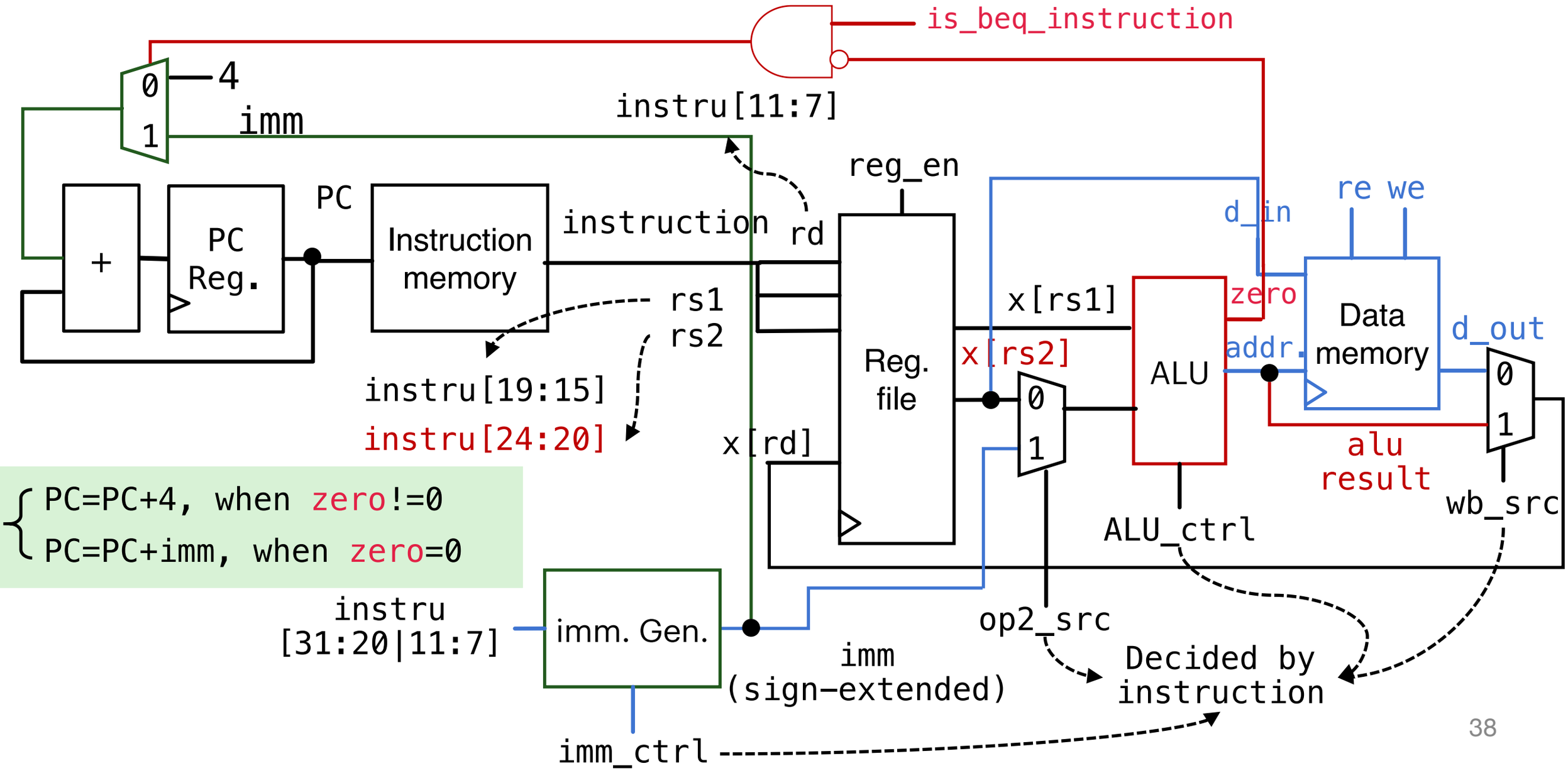


- I-type & S/B-type immediate

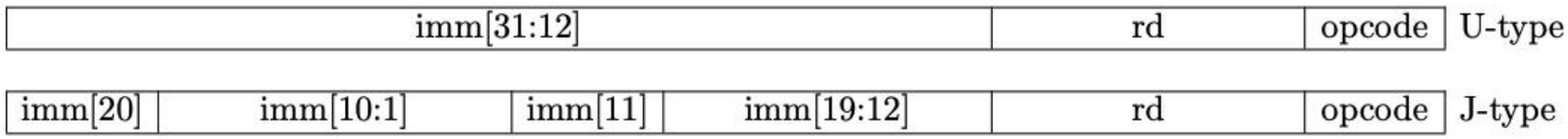


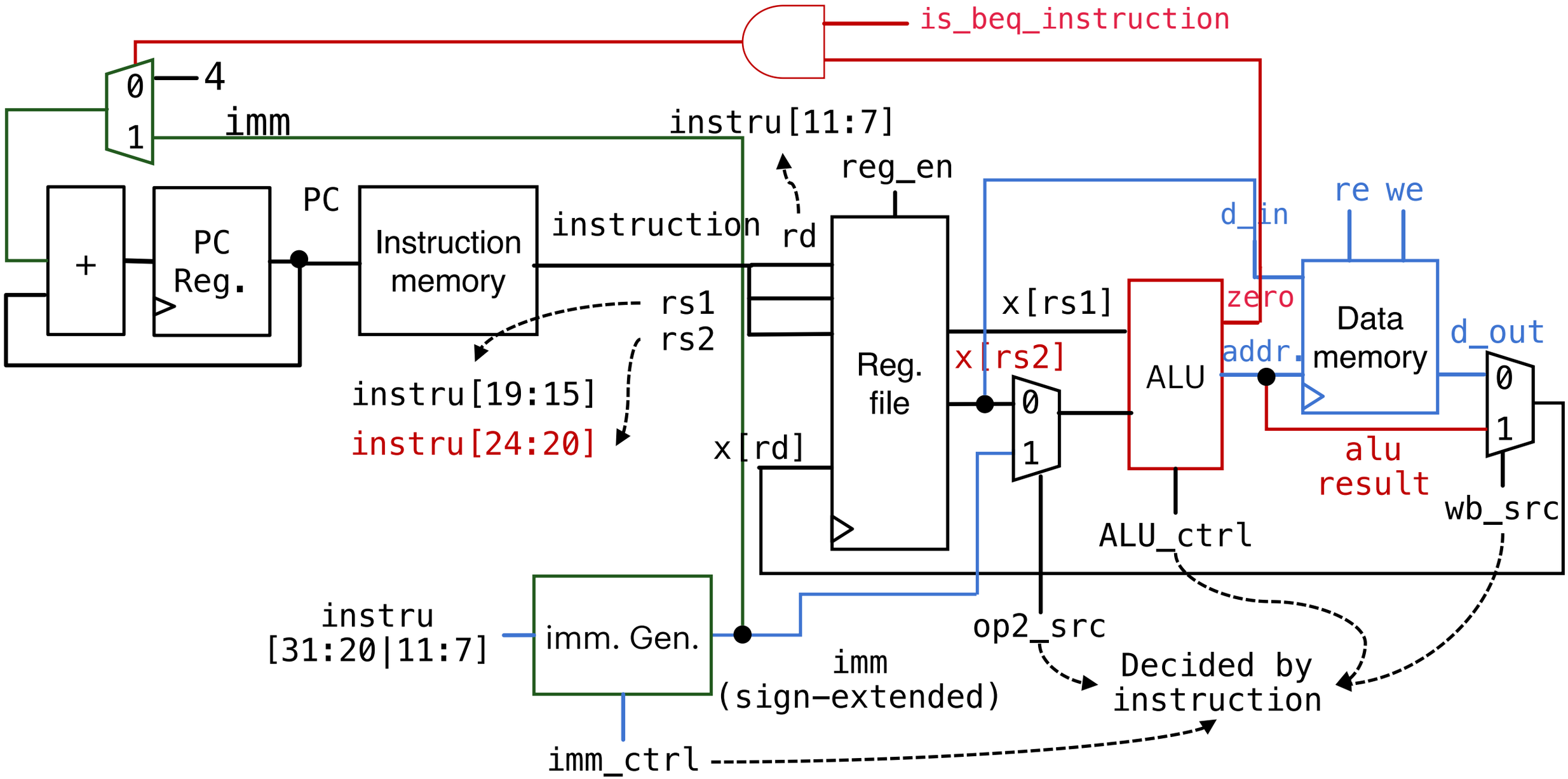
Datapath for B-type





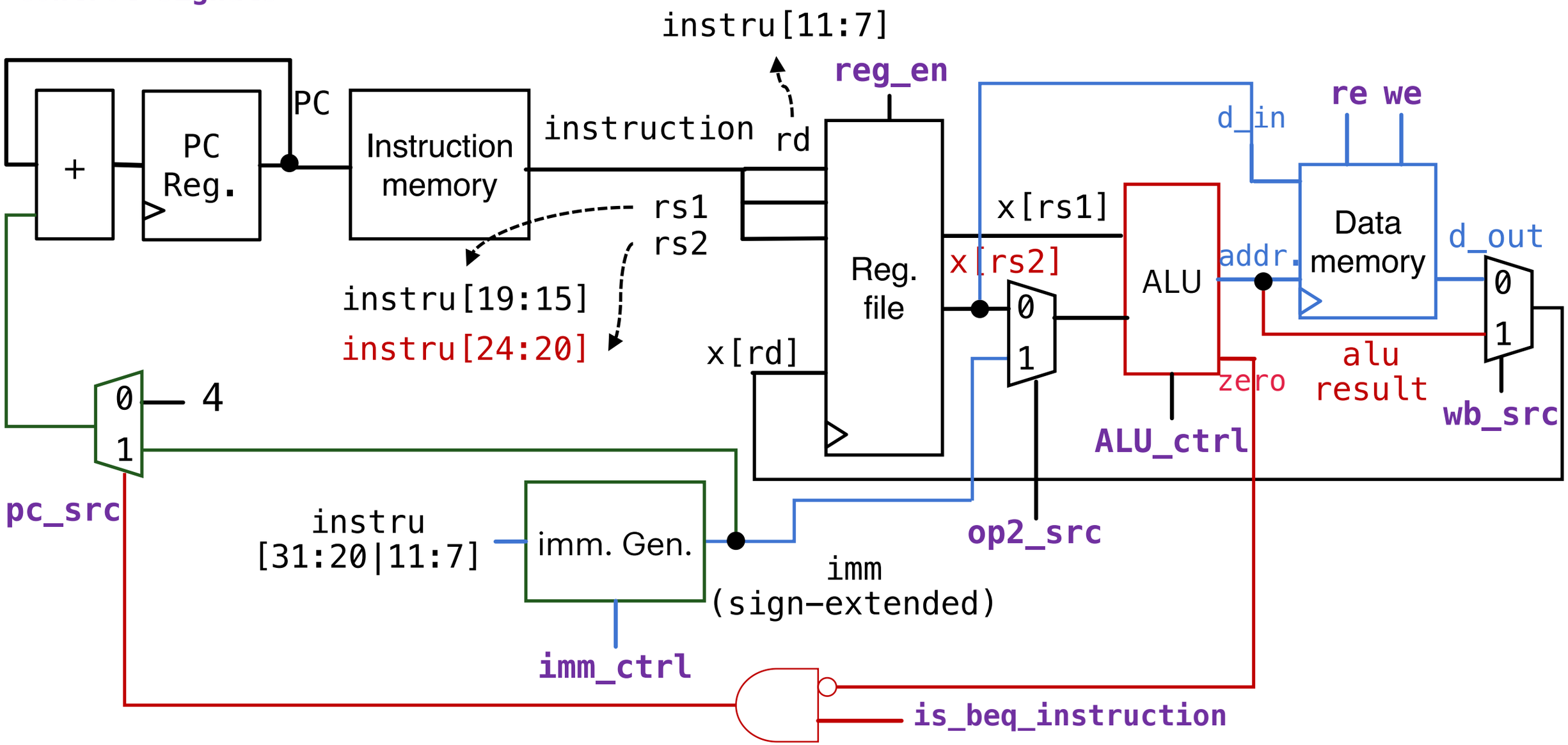
Datapath for the other types



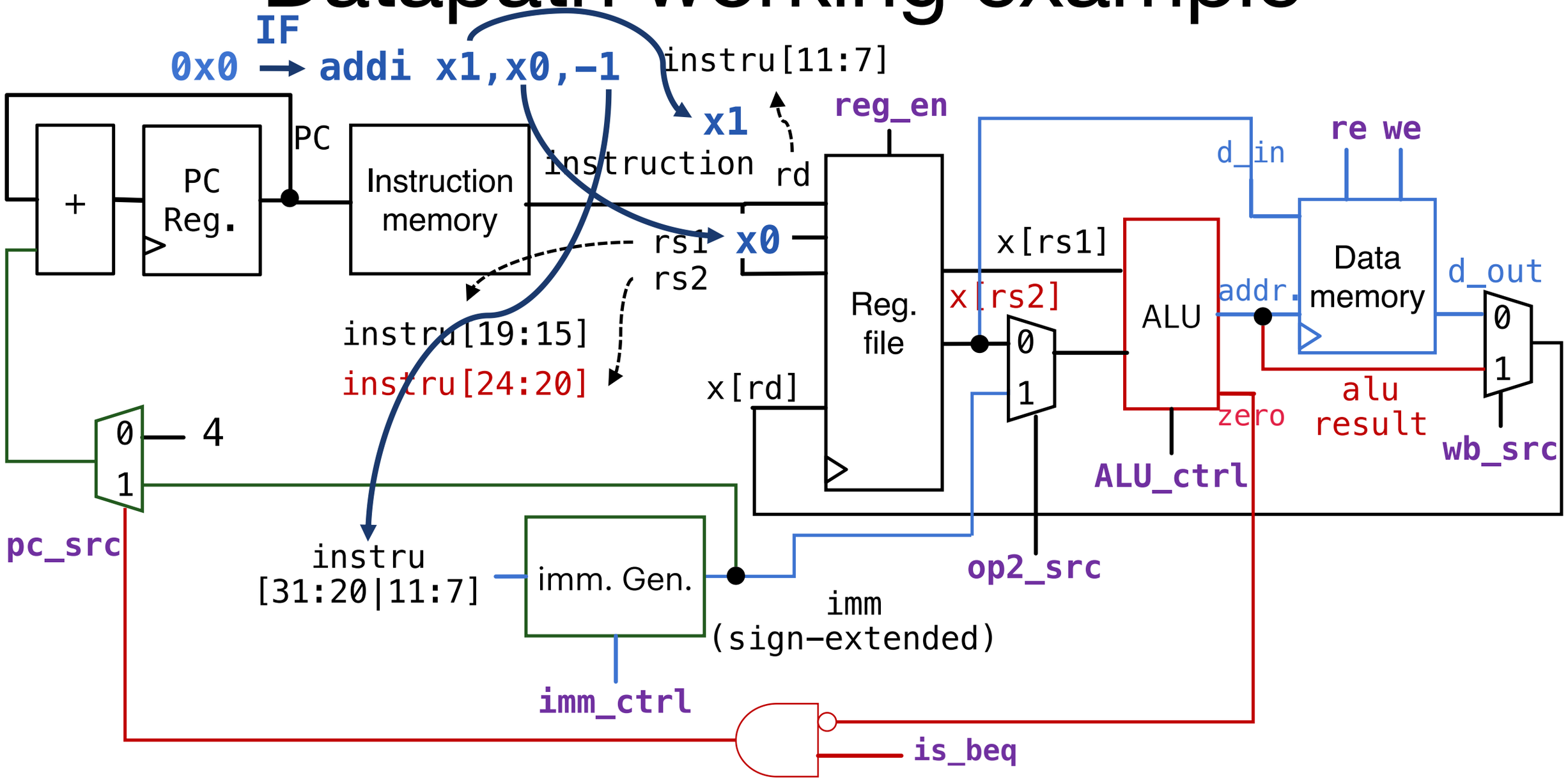


Control signals

- This is a datapath that supports R-type & I-type arithmetic and logic operations, `lw`, `sw` and `beq`



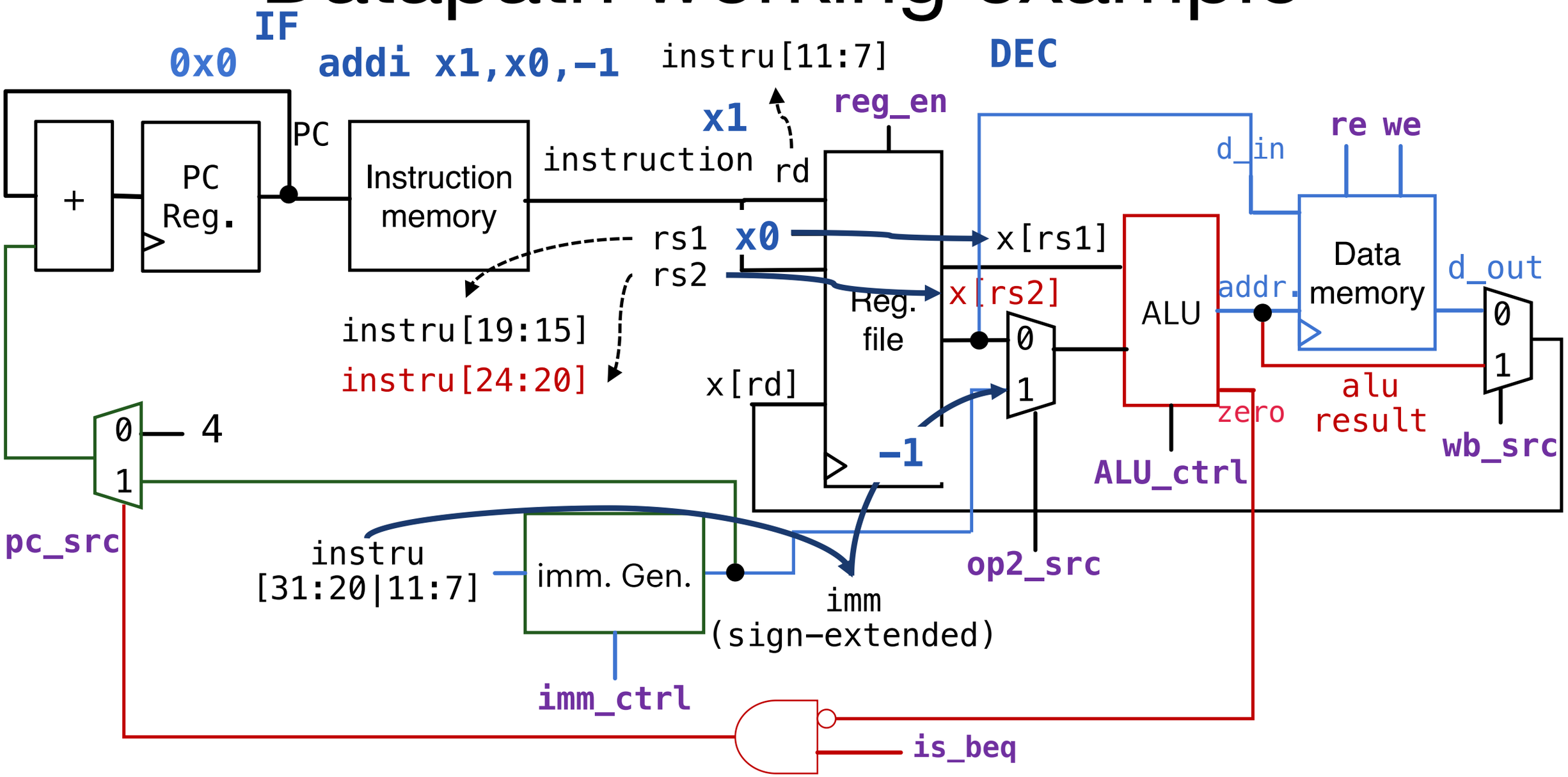
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
addi	1	0	0	add	I-type	1	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

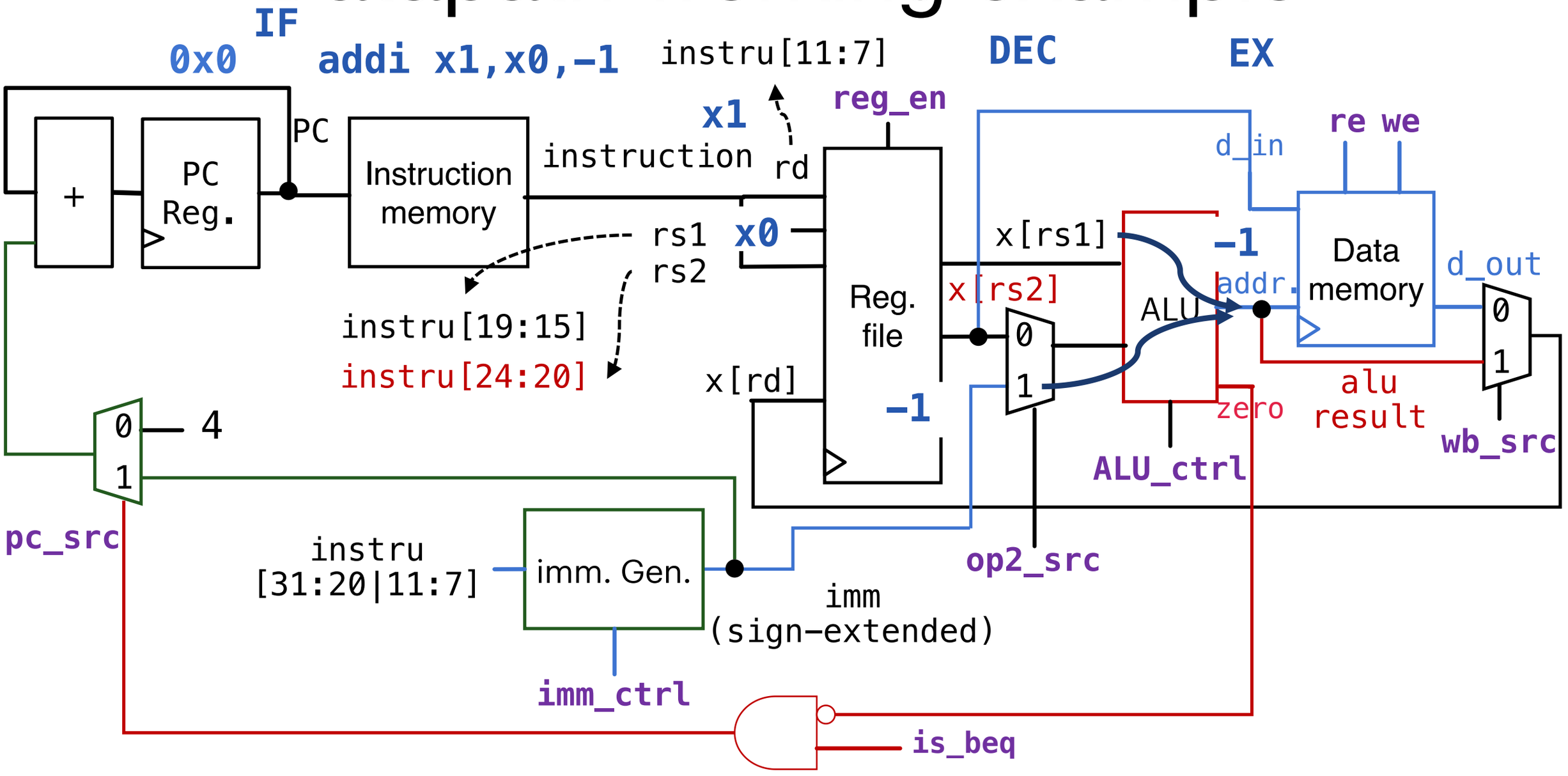
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
addi	1	0	0	add	I-type	1	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

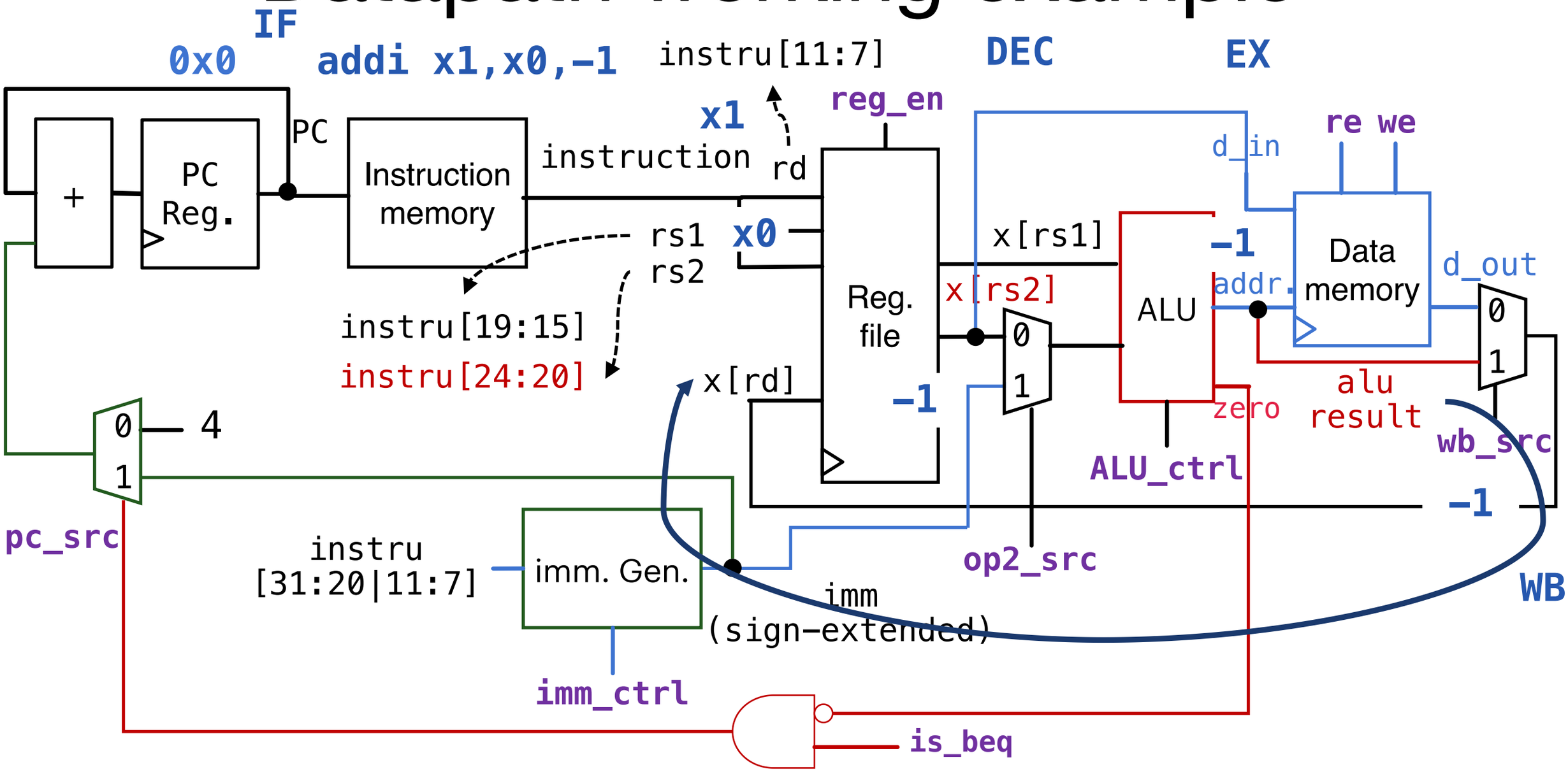
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
addi	1	0	0	add	I-type	1	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

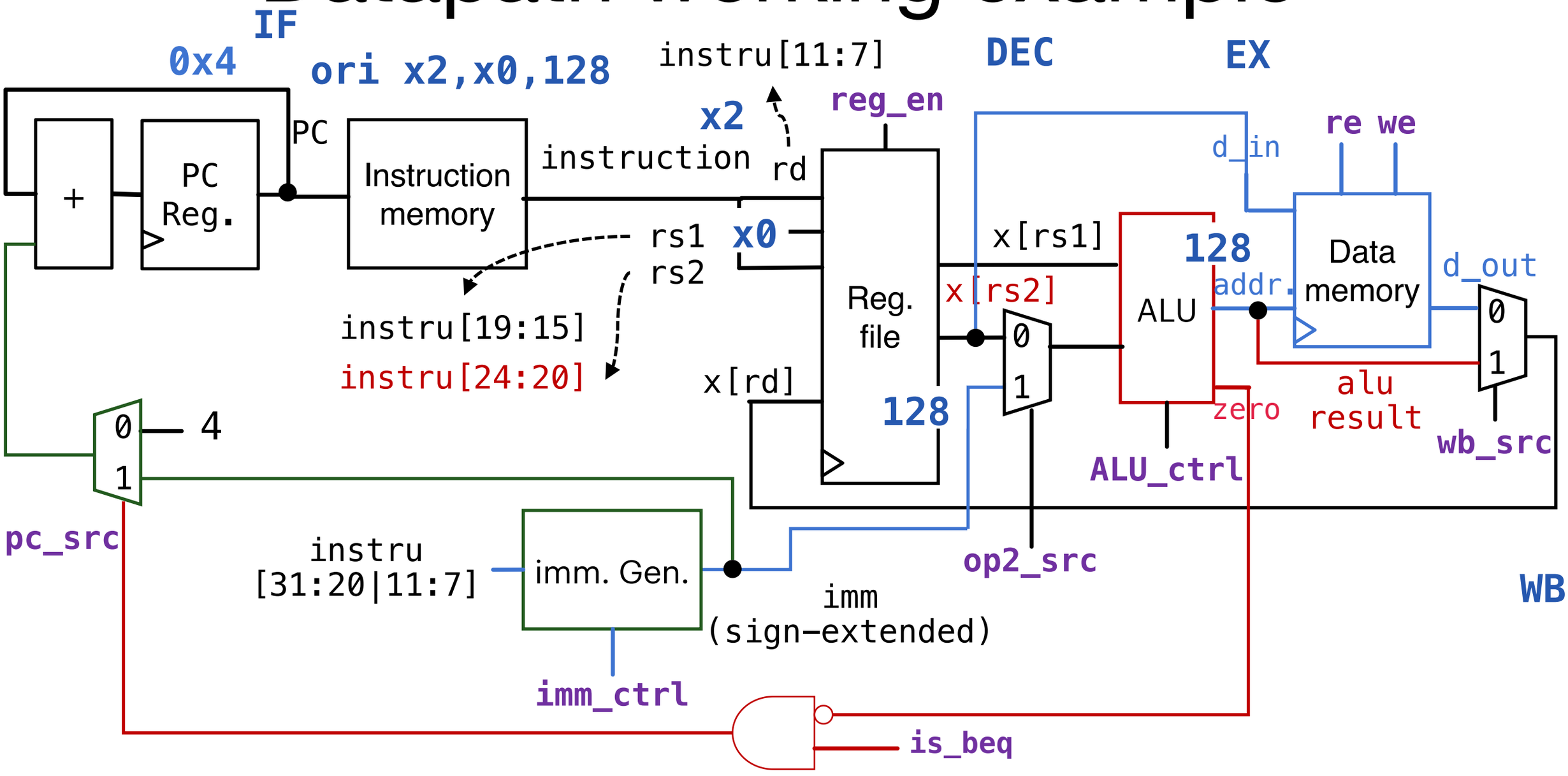
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
addi	1	0	0	add	I-type	1	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

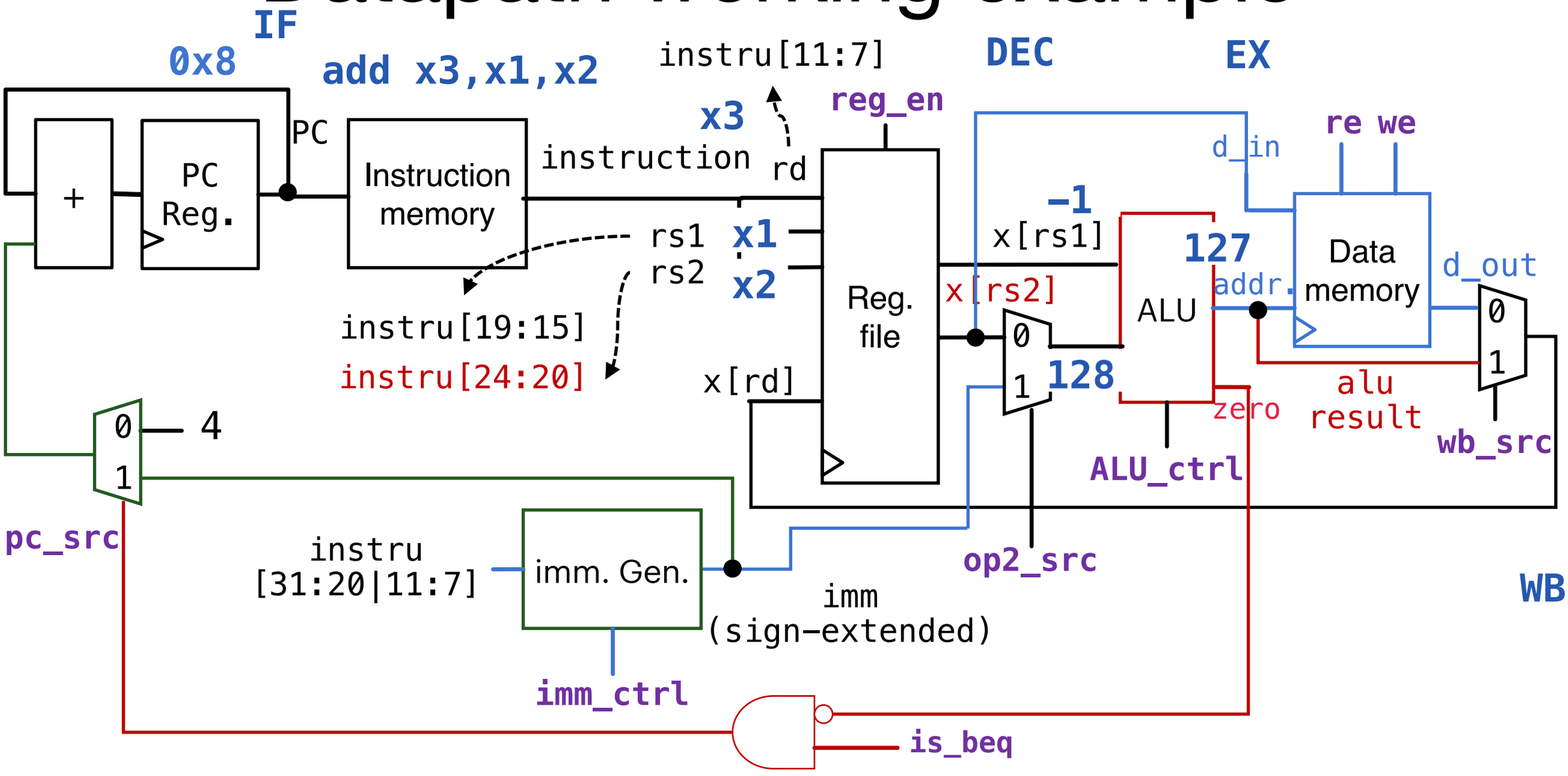
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
ori	1	0	0	or	I-type	1	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

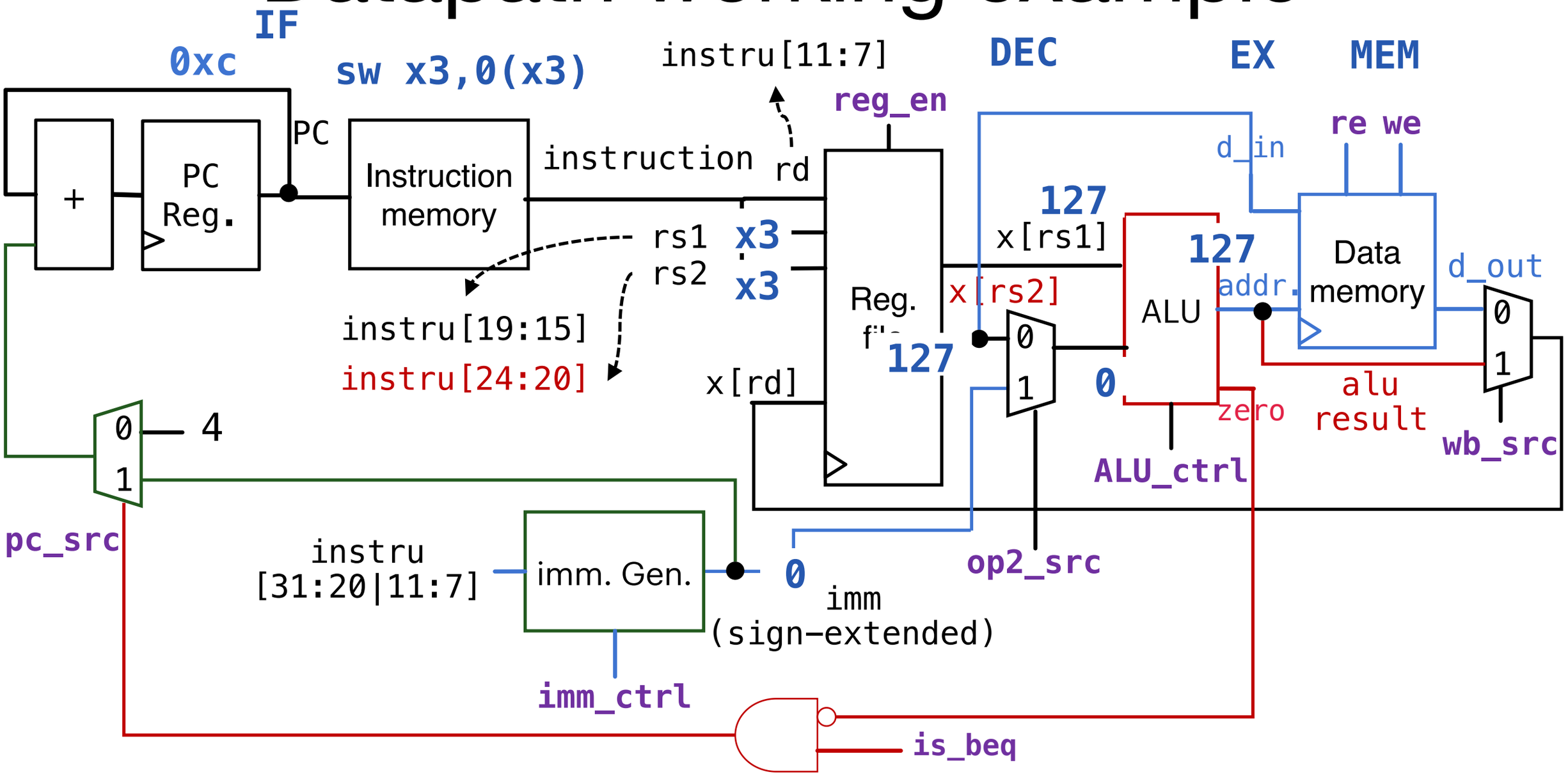
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
add	1	0	0	add	I-type	1	0	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

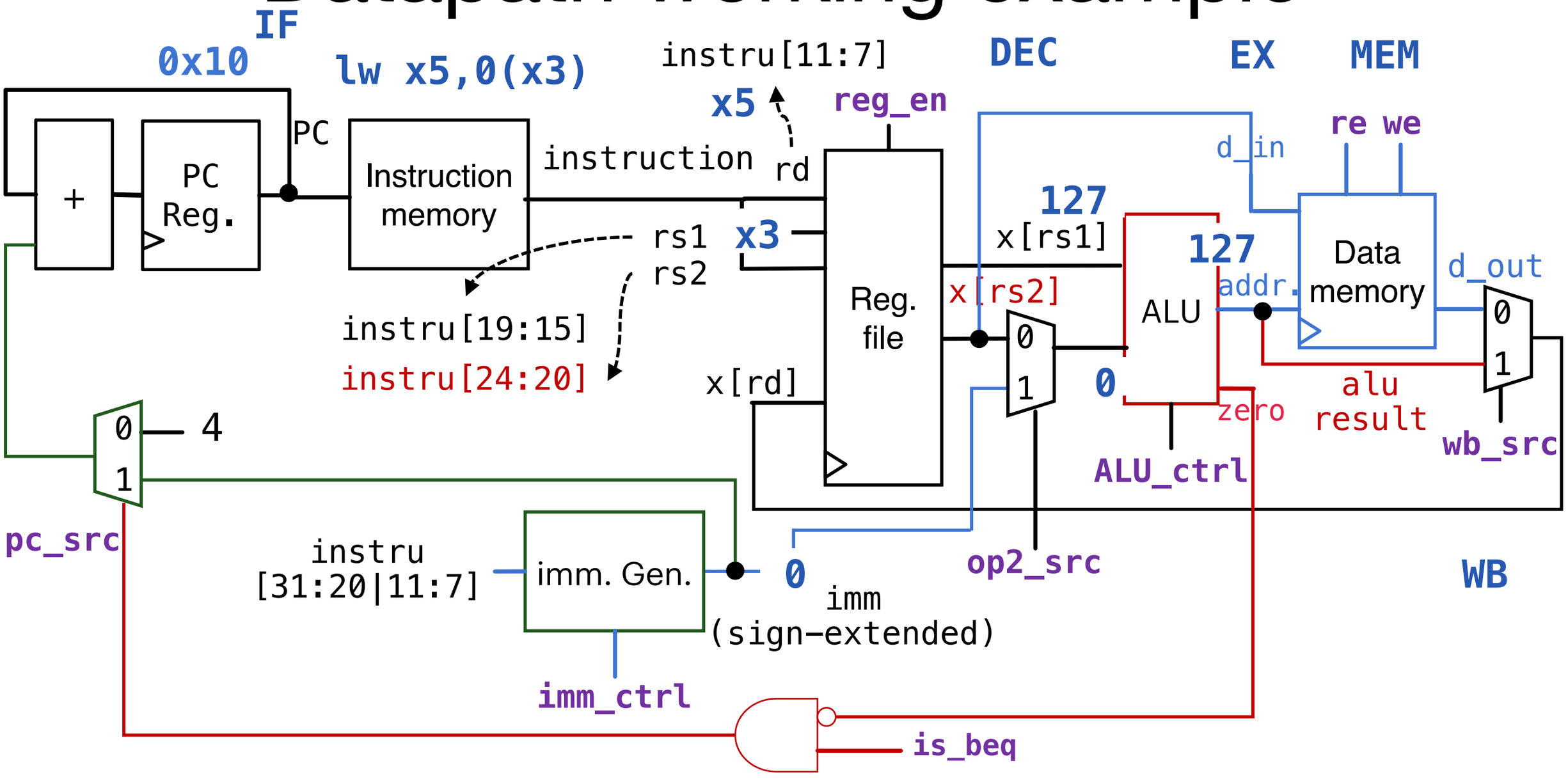
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
sw	0	0	1	add	S-type	X	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

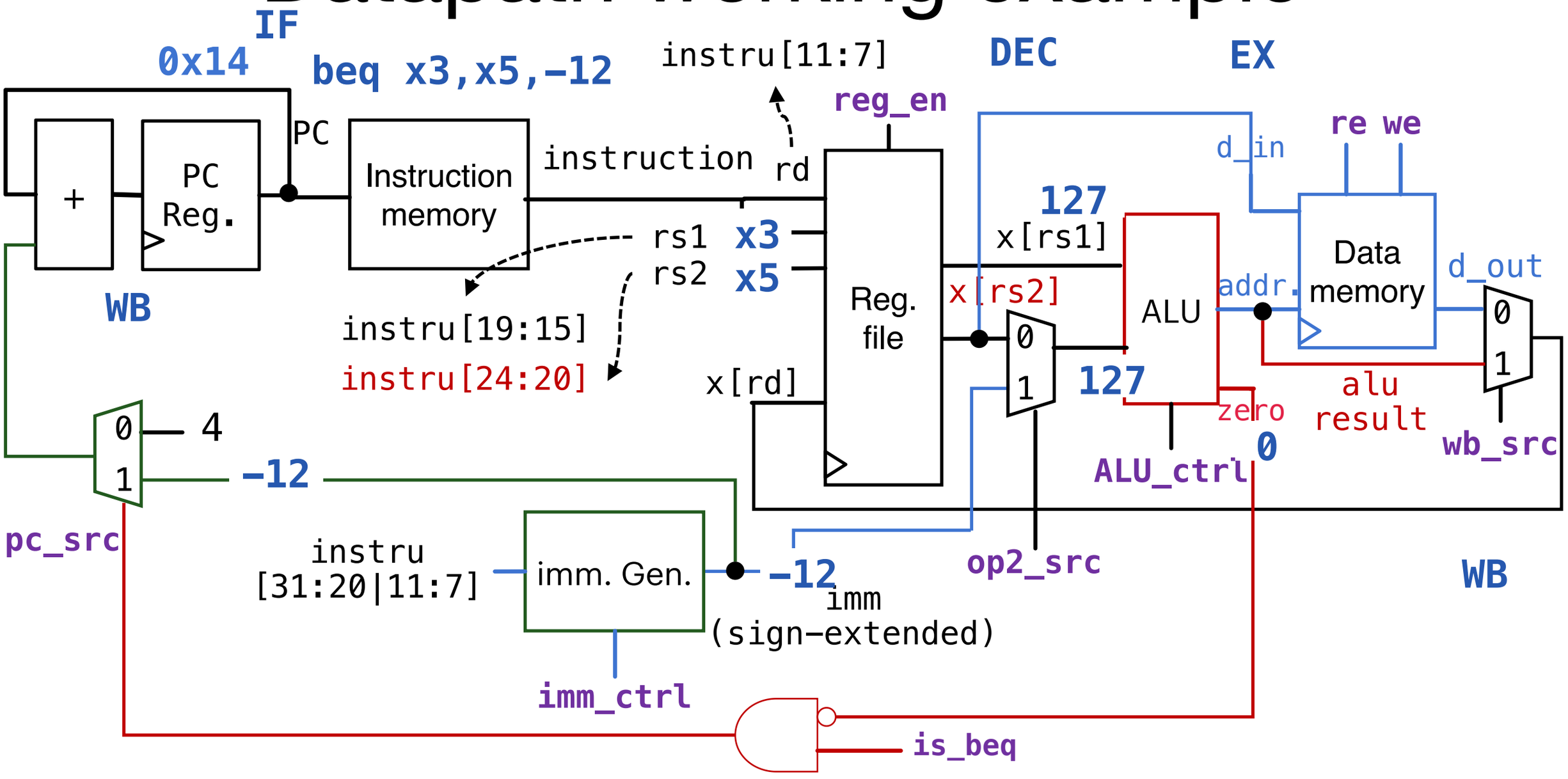
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
lw	1	1	0	add	I-type	0	1	0

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

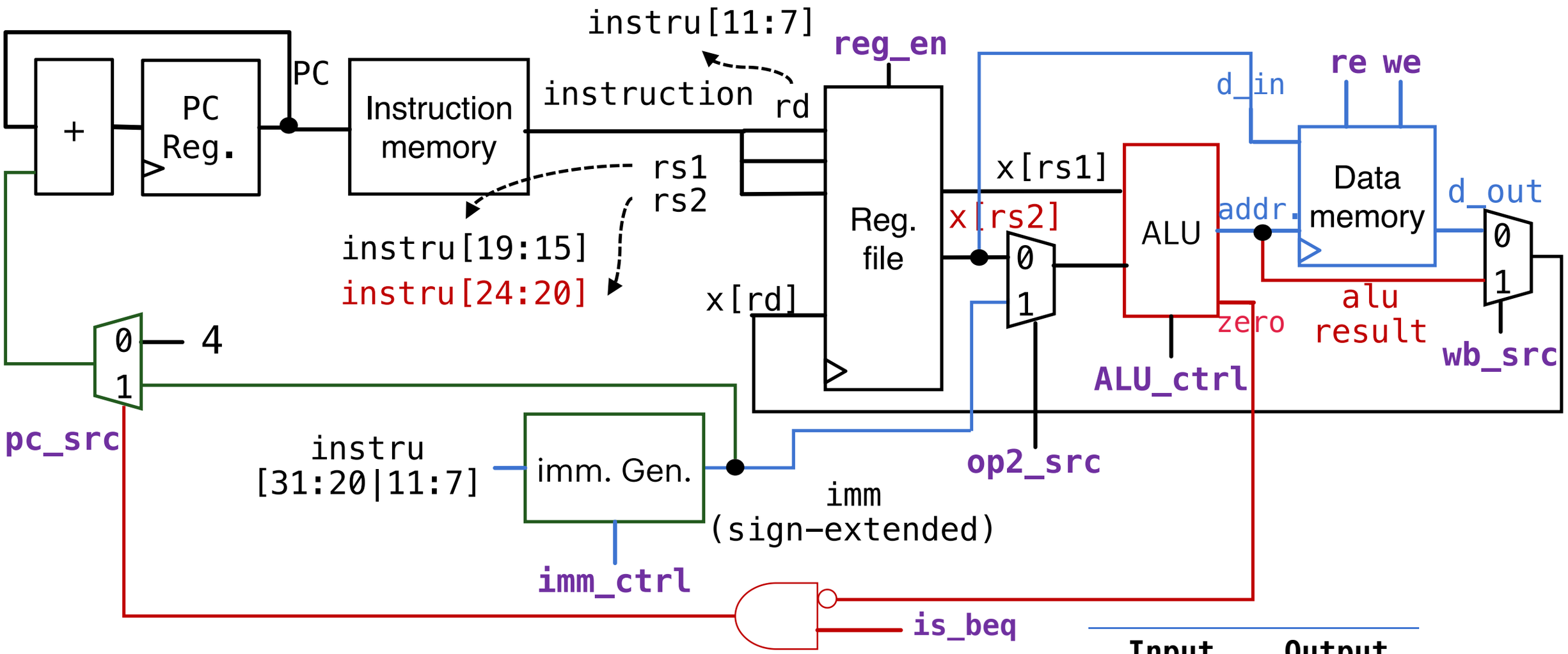
Datapath working example



	reg_en	re	we	alu_ctrl	imm_ctrl	wb_src	op2_src	is_beq
beq	0	0	0	sub	B-type	X	0	1

```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

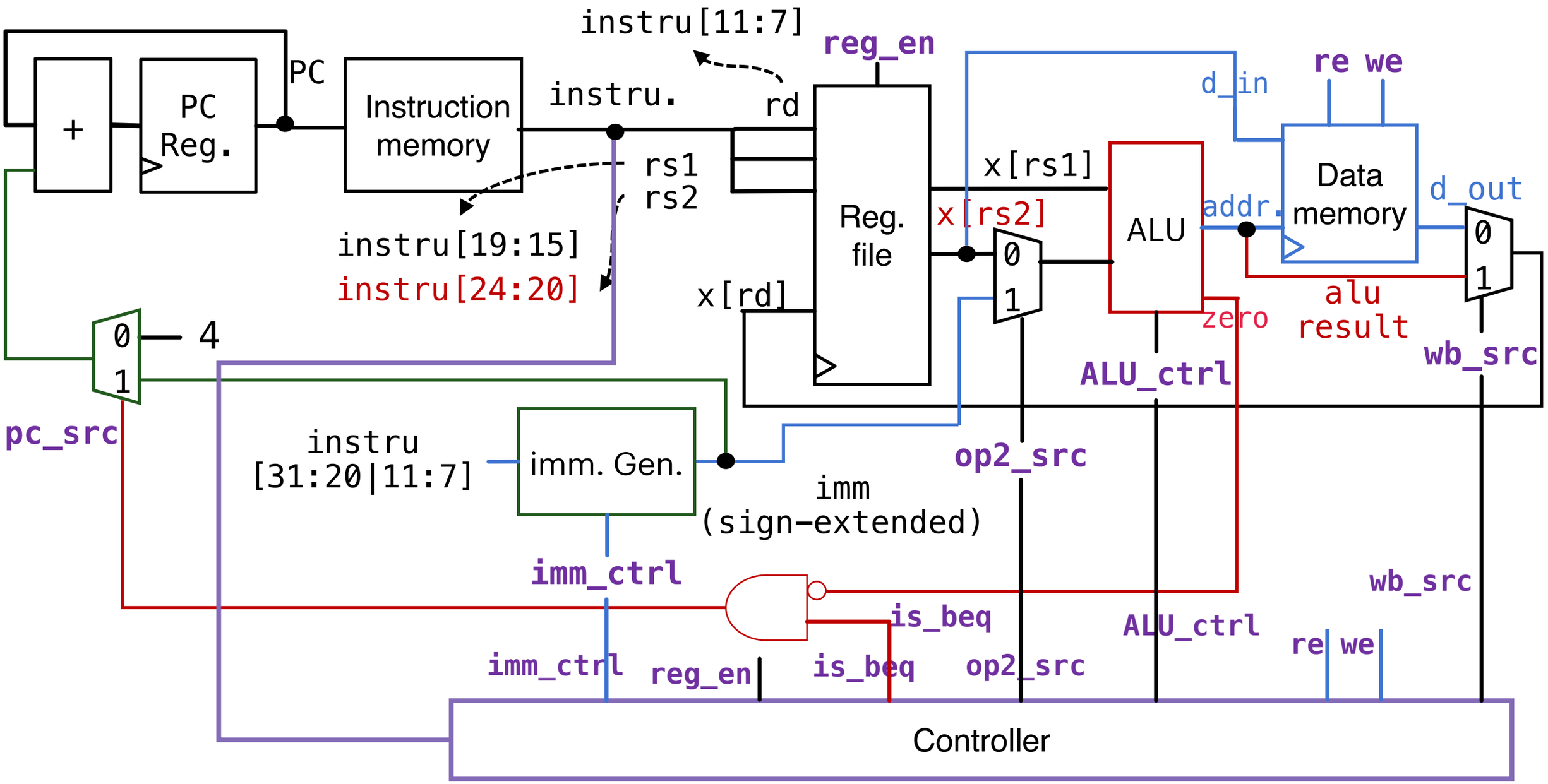
Controller



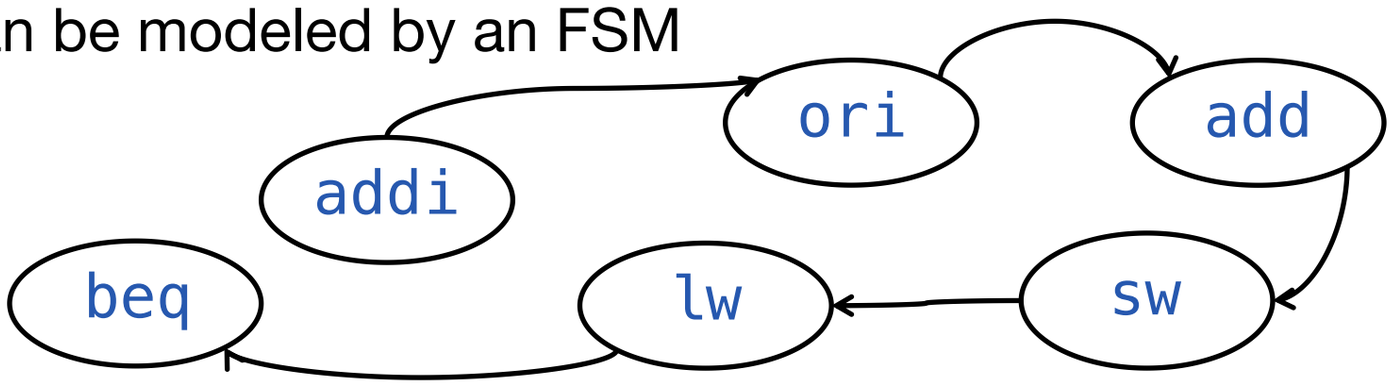
- Generate control signals guiding the datapath to execute instructions
- The inputs are instructions, the outputs are the control signals
- Once the type of instruction is determined, the control signal is determined

Input	Output
Instru.	reg_en
	re
	we
	alu_ctrl
	imm_ctrl
	wb_src
	op2_src
	is_beq

Controller

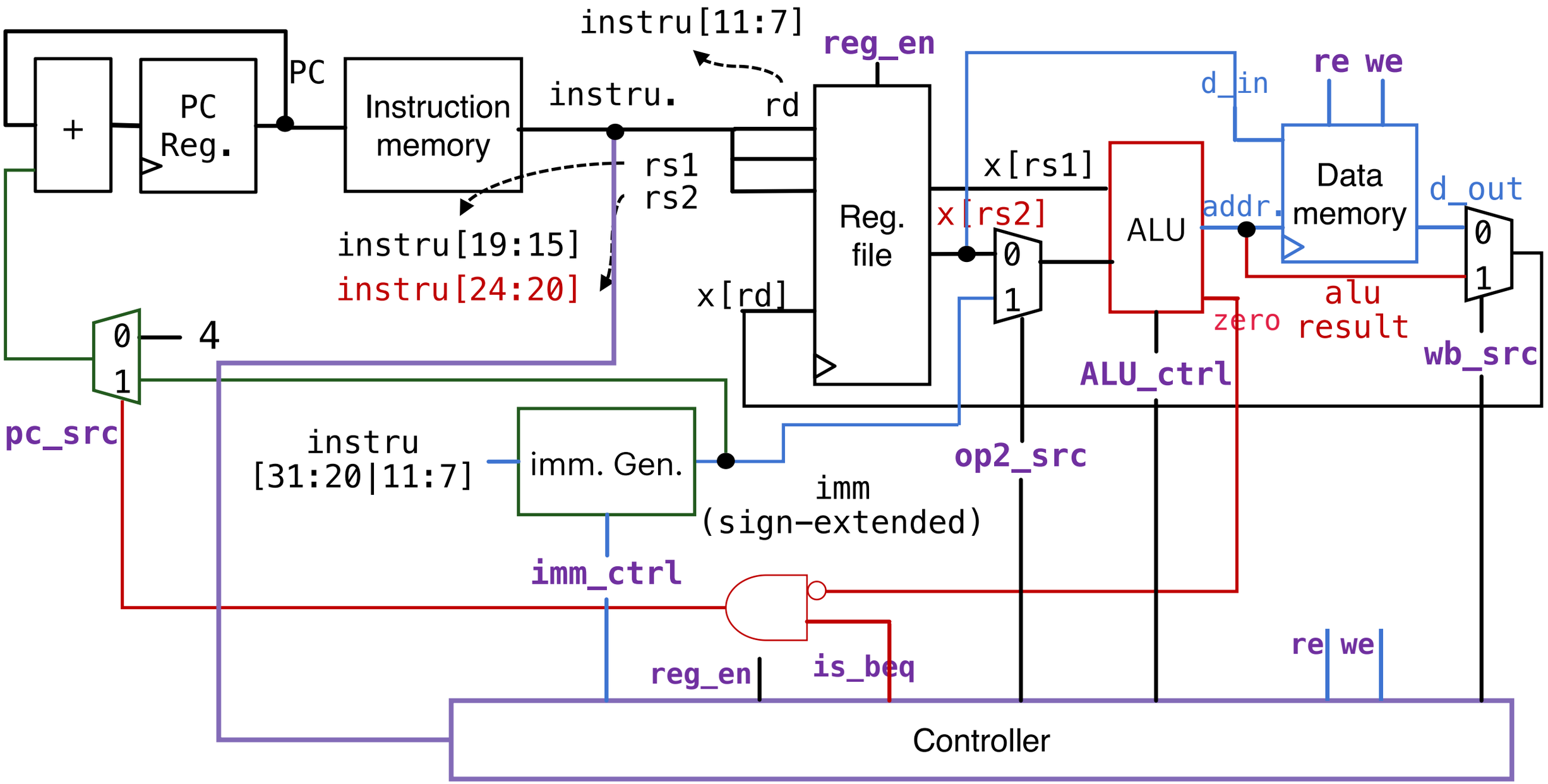


- Can be modeled by an FSM

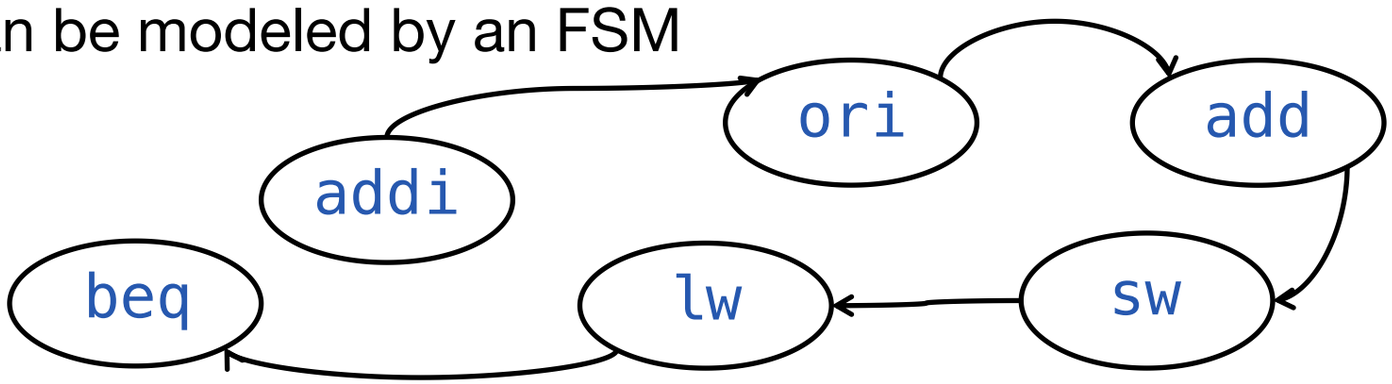



```
0x0: addi x1, x0, -1
0x4: ori  x2, x0, 128
0x8: add  x3, x1, x2
0xc: sw   x3, 0(x3)
0x10:lw   x5, 0(x3)
0x14:beq  x3, x5, -12
```

Controller

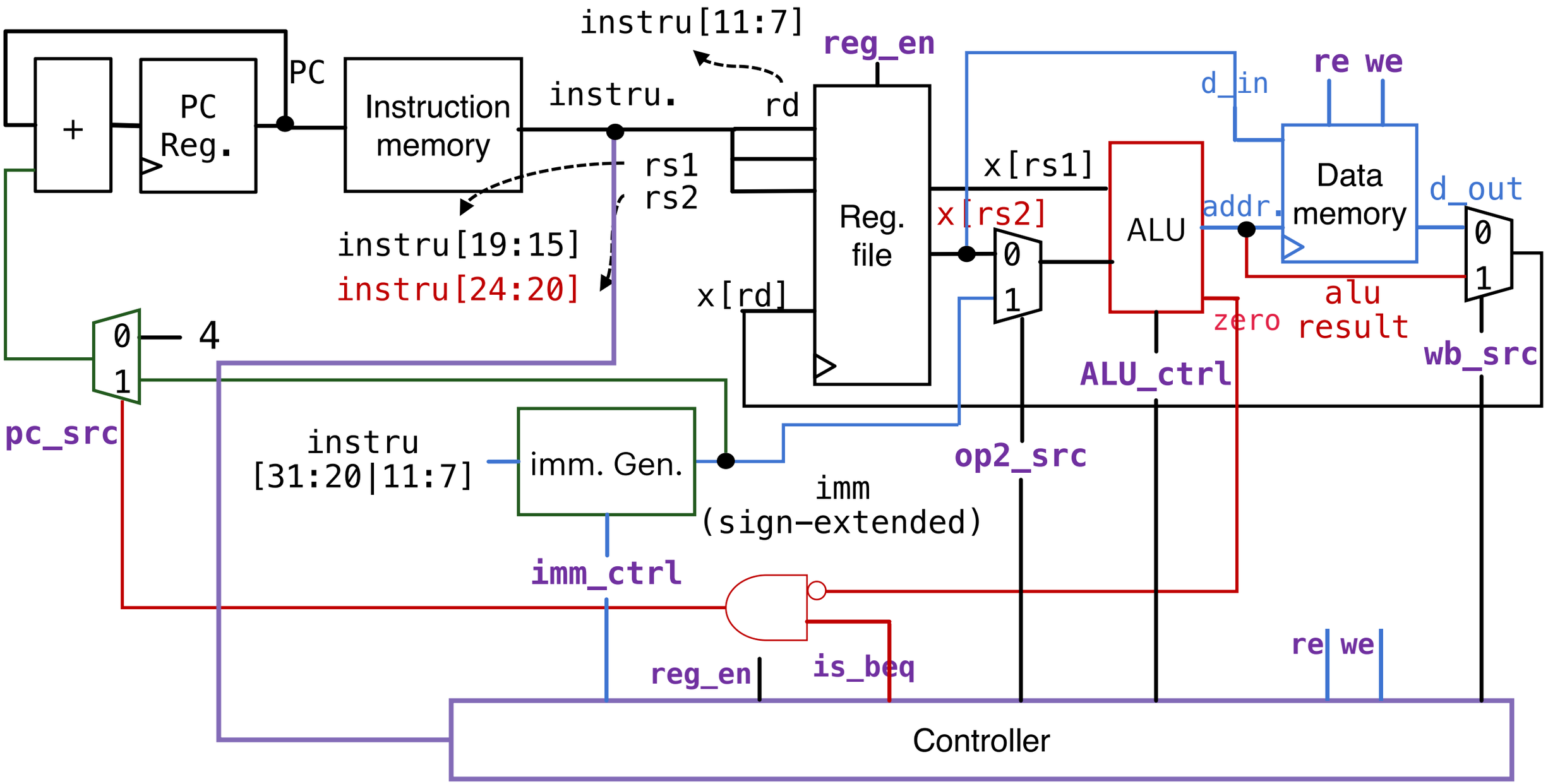


- Can be modeled by an FSM

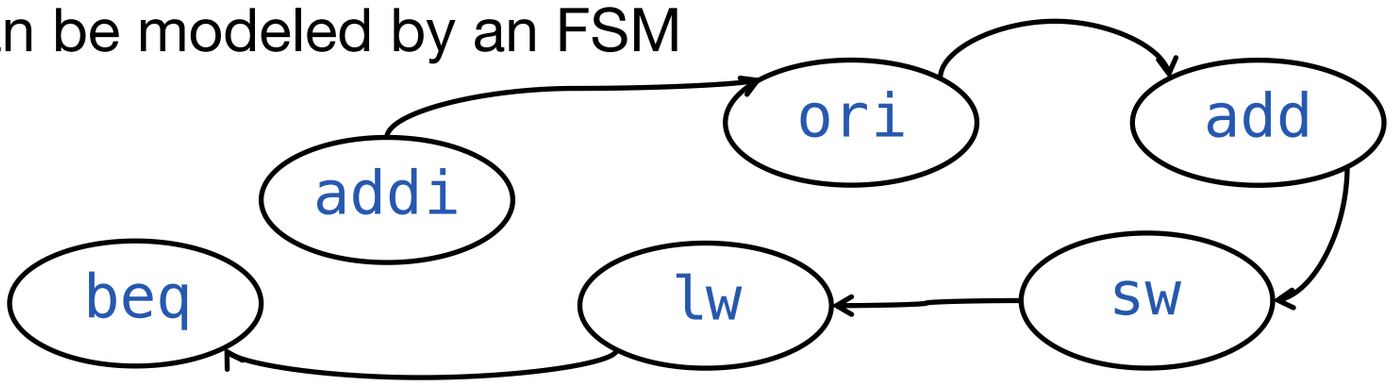


- Can be implemented by a Mealy machine

Controller

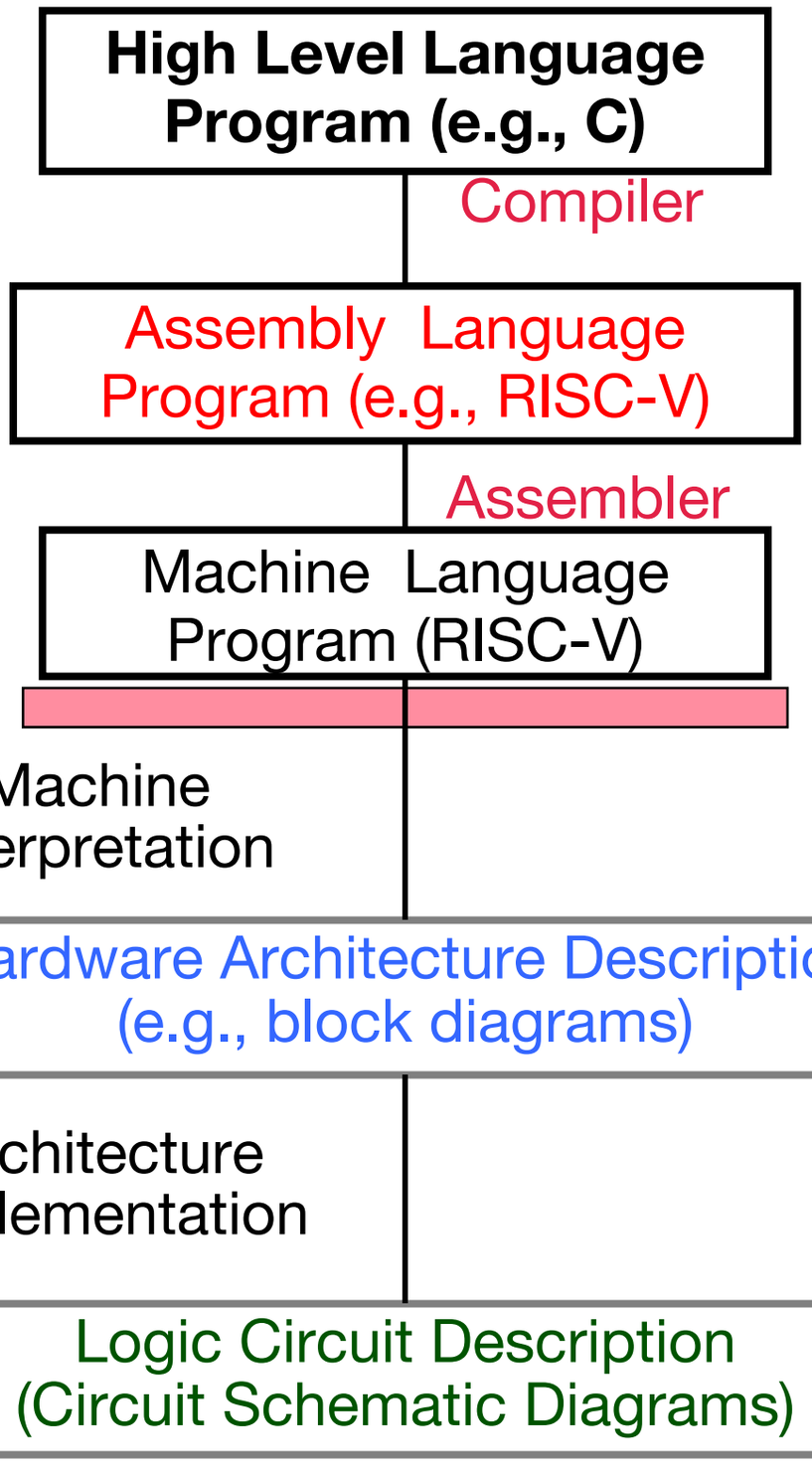


- Can be modeled by an FSM



- Can also be implemented by read-only memory (ROM), optional

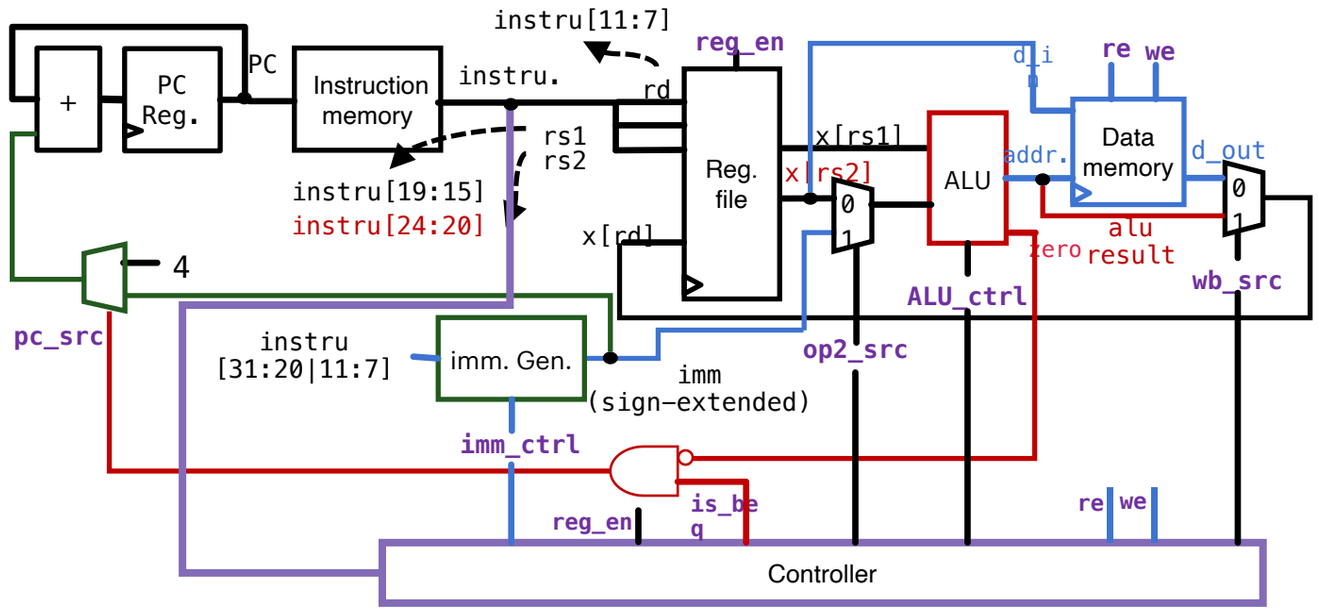
Full stack explained



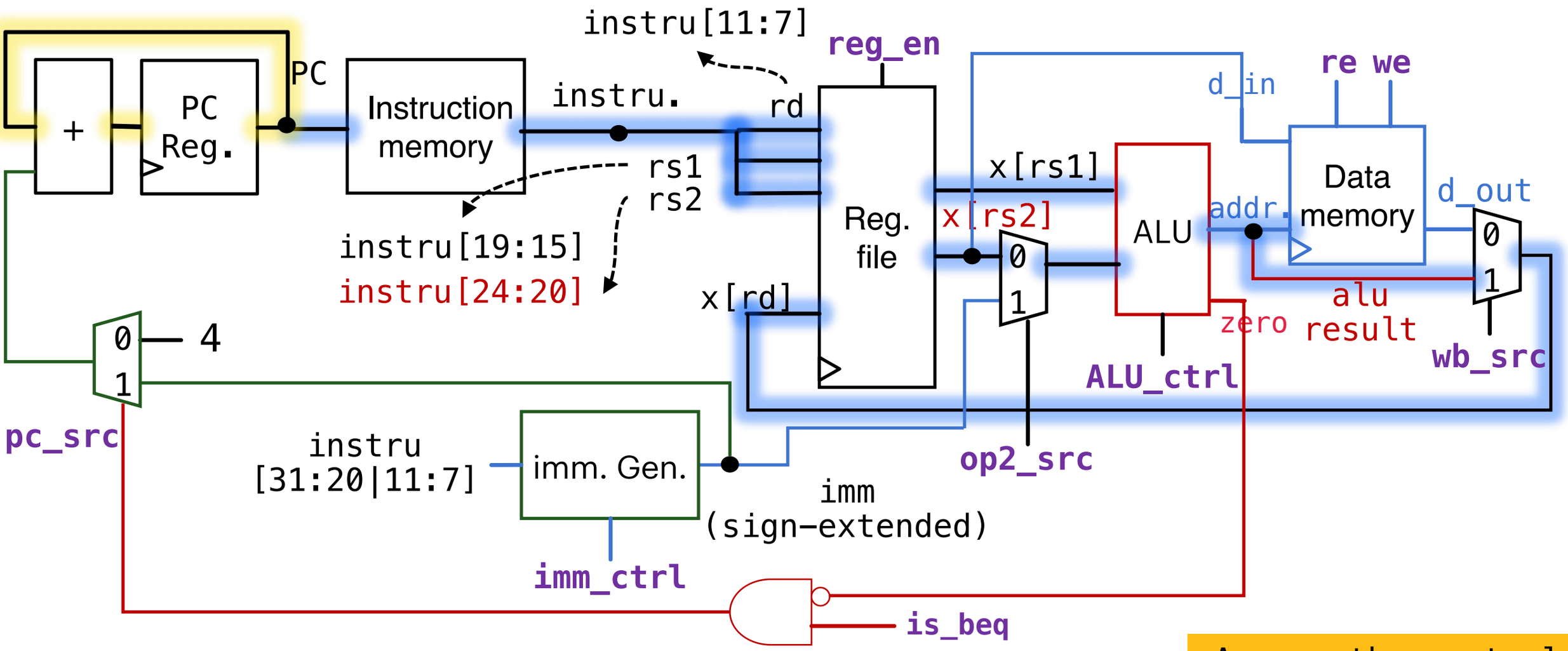

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   t0, 0(s2)
lw   t1, 4(s2)
sw   t1, 0(s2)
sw   t0, 4(s2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



Datapath timing analysis



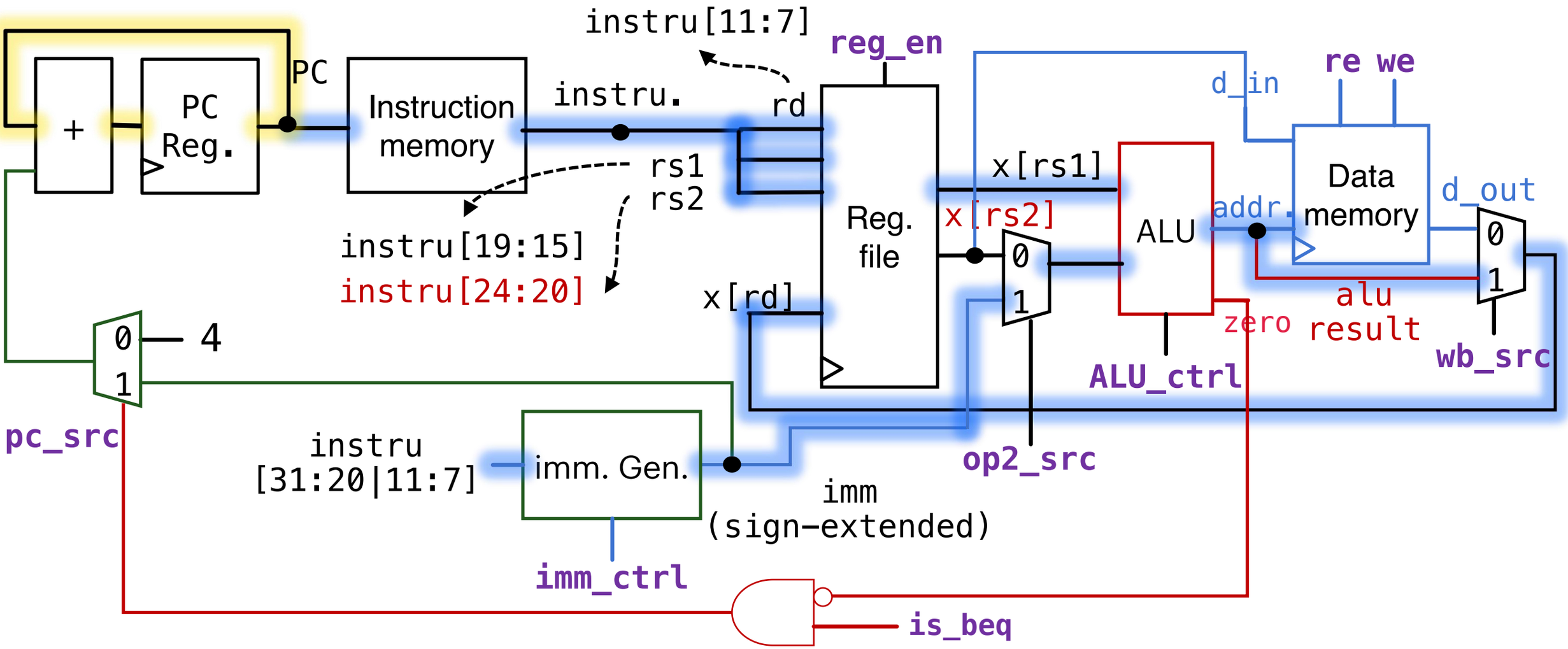
Assume the control signal is fast

$t_{clk-to-q} + t_{add} + t_{setup}$

$$\underbrace{t_{clk-to-q} + t_{Imem}}_{t_{IF}} + \underbrace{t_{reg} + t_{mux}}_{t_{DEC}} + \underbrace{t_{alu}}_{t_{EX}} + \underbrace{t_{mux} + t_{setup}}_{t_{WB}}$$

R-type datapath

Datapath timing analysis



$t_{clk-to-q} + t_{add} + t_{setup}$

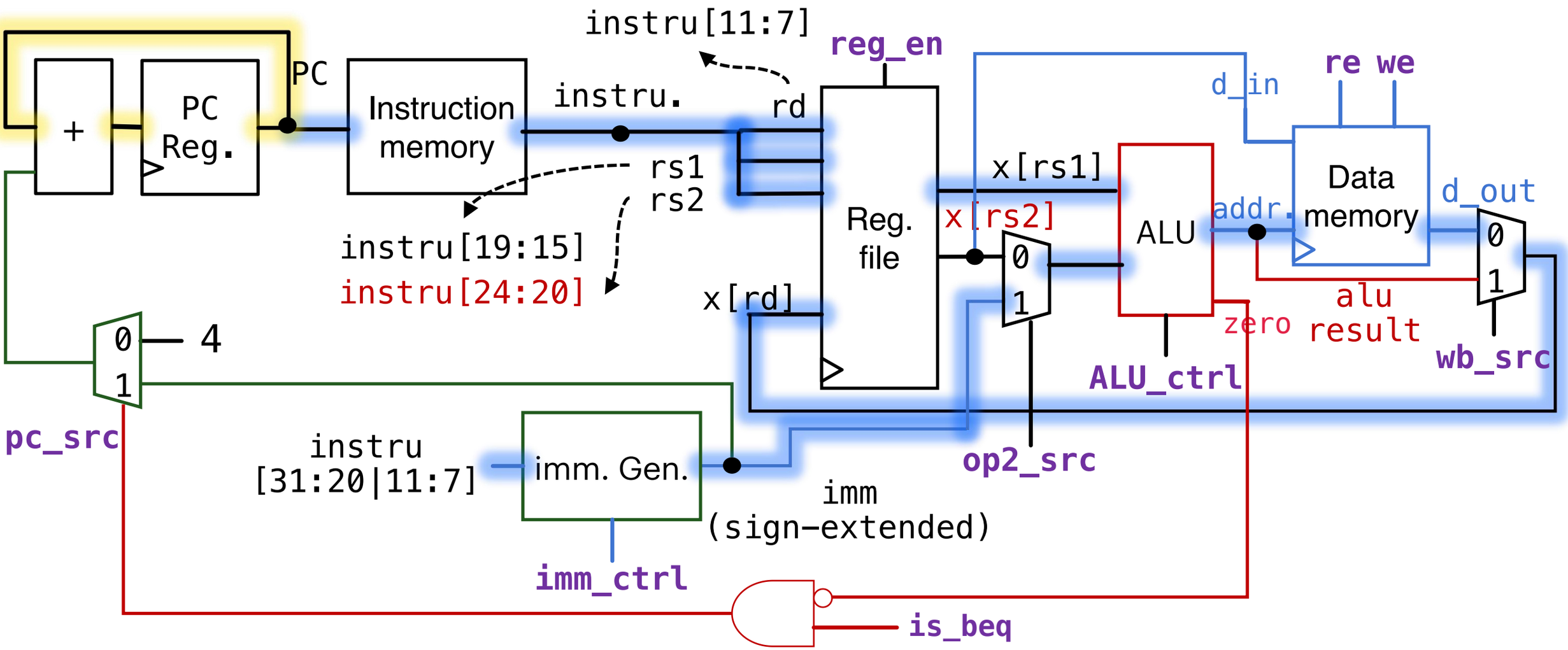
$t_{clk-to-q} + t_{Imem} + t_{reg} + t_{alu} + t_{mux} + t_{setup}$

$t_{clk-to-q} + t_{Imem} + t_{imm} + t_{mux} + t_{alu} + t_{mux} + t_{setup}$

} max $t_{IF} + t_{DEC} + t_{EX} + t_{WB}$

I-type arithmetic & logic datapath

Datapath timing analysis



$t_{clk-to-q} + t_{add} + t_{setup}$

$t_{IF} + t_{DEC} + t_{EX} + t_{MEM} + t_{WB}$

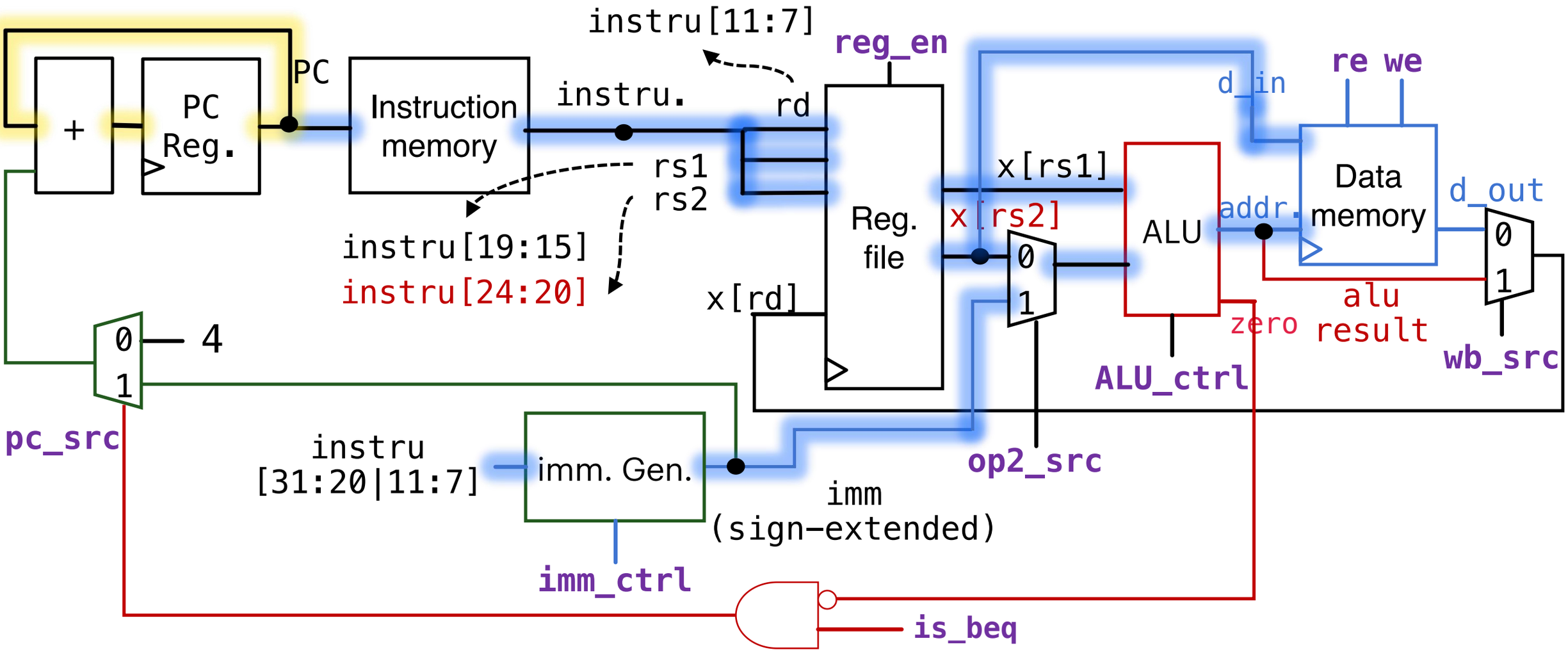
$t_{clk-to-q} + t_{Imem} + t_{reg} + t_{alu} + t_{Dmem} + t_{mux} + t_{setup}$

$t_{clk-to-q} + t_{Imem} + t_{imm} + t_{mux} + t_{alu} + t_{Dmem} + t_{mux} + t_{setup}$

} max

I-type load datapath

Datapath timing analysis



$t_{clk-to-q} + t_{add} + t_{setup}$

$t_{clk-to-q} + t_{Imem} + t_{reg} + t_{Dmem}$

$t_{clk-to-q} + t_{Imem} + t_{reg} + t_{alu} + t_{Dmem}$

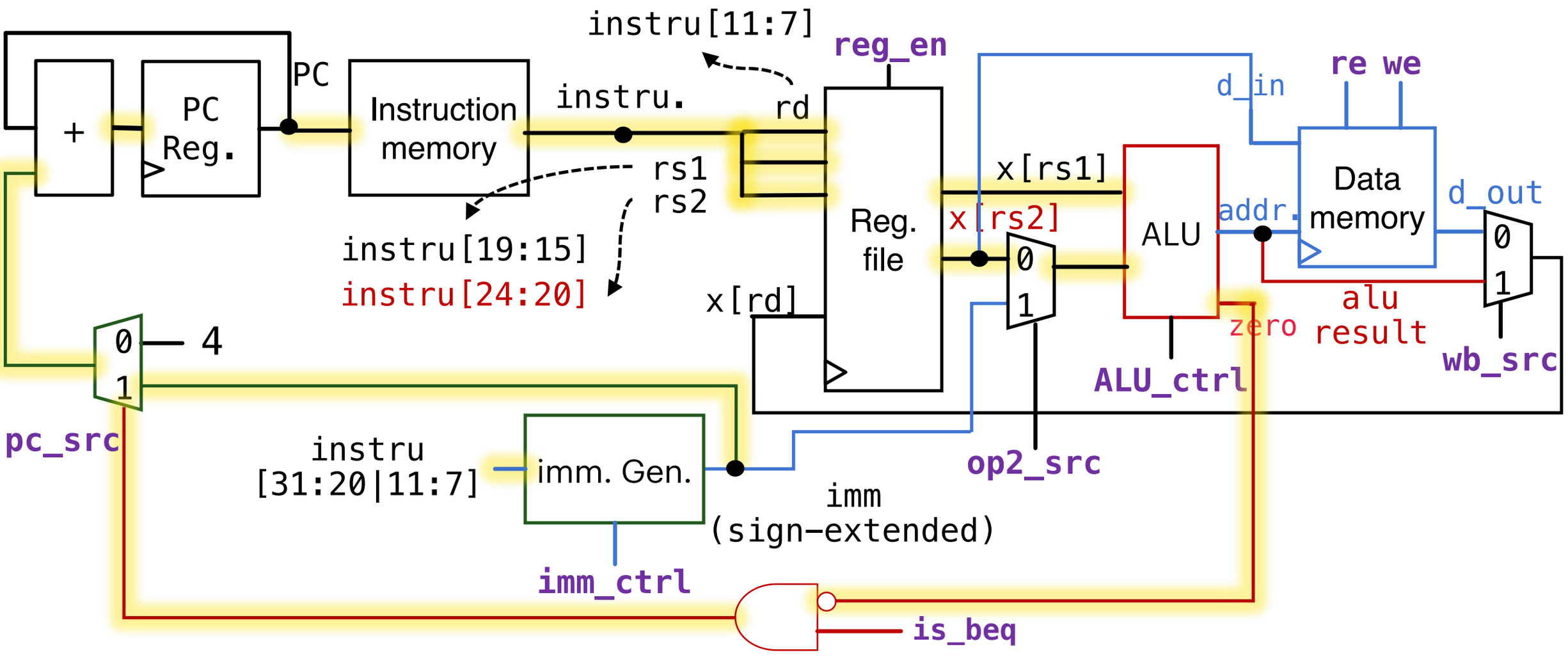
$t_{clk-to-q} + t_{Imem} + t_{imm} + t_{mux} + t_{alu} + t_{Dmem}$

} max

$t_{IF} + t_{DEC} + t_{EX} + t_{MEM}$

S-type datapath

Datapath timing analysis



$$t_{clk-to-q} + t_{Imem} + t_{reg} + t_{mux} + t_{alu} + t_{and} + t_{mux} + t_{add} + t_{setup}$$

$$t_{clk-to-q} + t_{Imem} + t_{imm} + t_{mux} + t_{add} + t_{setup}$$

} max $t_{IF} + t_{DEC} + t_{EX} + t_{WB}$

B-type datapath

Summary

- We have built a single-cycle CPU
- It supports R-type, I-type arithmetic & logic and load (`lw`), S-type `sw` and `beq`
- Datapath and controller are built seperately
- Different instruction activates different parts or steps/stages (IF/DEC/EXE/MEM/WB) of the datapath, thus has different delays. The longest delay (critical path) is used to estimate maximum frequency
- Nearly no CPU uses single-cycle design today.