

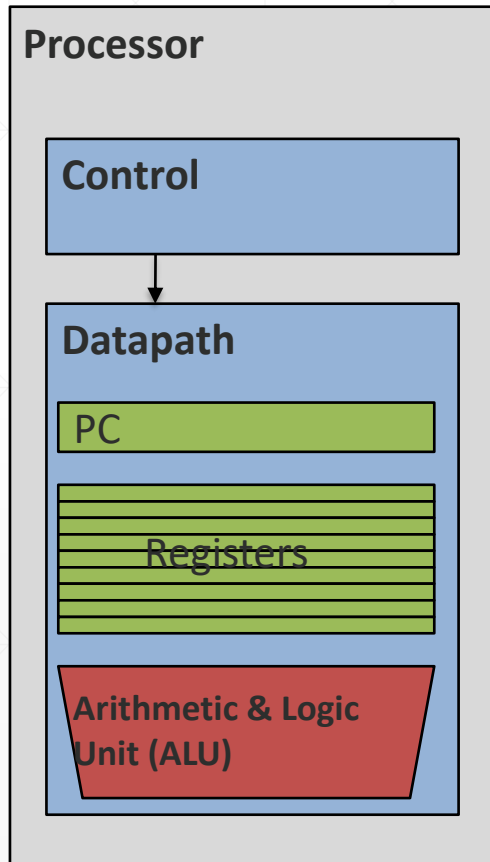
CS110A

Datapath and Control Logic (I)

Chundong Wang

Controller & Datapath

- A CPU that support RV32I can have so many states

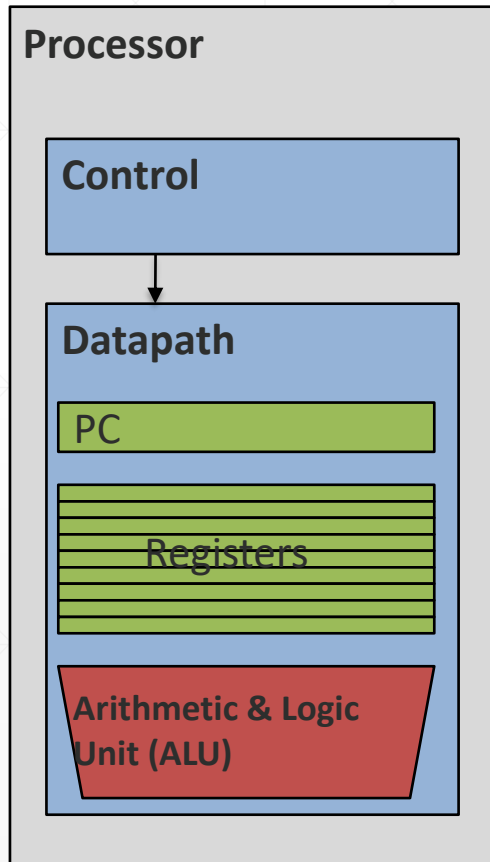


- Consider the 32 registers alone
 - x0 always 0
 - Each bit in the other registers can be 0 or 1
- Not practical to enumerate all the state transitions
- Top-down design: build small modules and then connect them as needed
- Most digital systems can be divided into datapath and controller
 - Datapath contains data processing and storage
 - Controller controls data access (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).

Controller & Datapath

- A CPU that support RV32I can have so many states

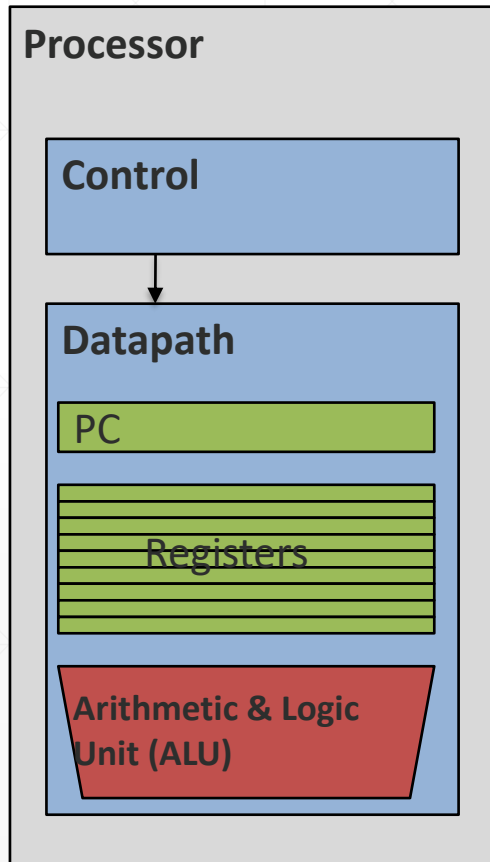


- Datapath
 - Start with **basic building blocks**
 - Add building blocks to the digital system with added supported instructions
- Controller
 - Can be considered as an FSM

- 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).

Useful building blocks

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



ADD

SUB

SLL

SLT

SLTU

XOR

SRL

SRA

OR

AND

ADDI

SLTI

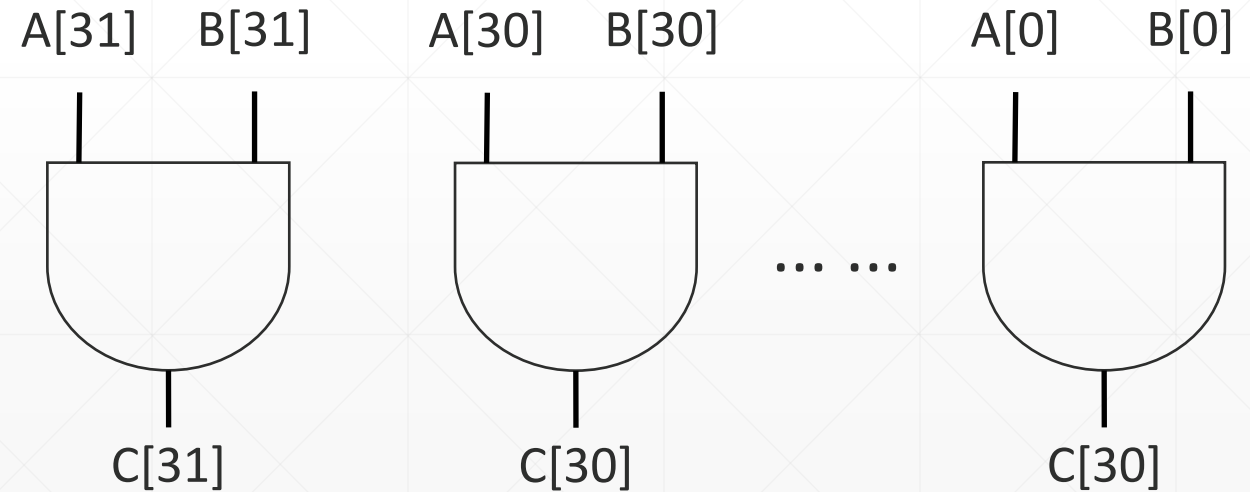
SLTIU

XORI

ORI

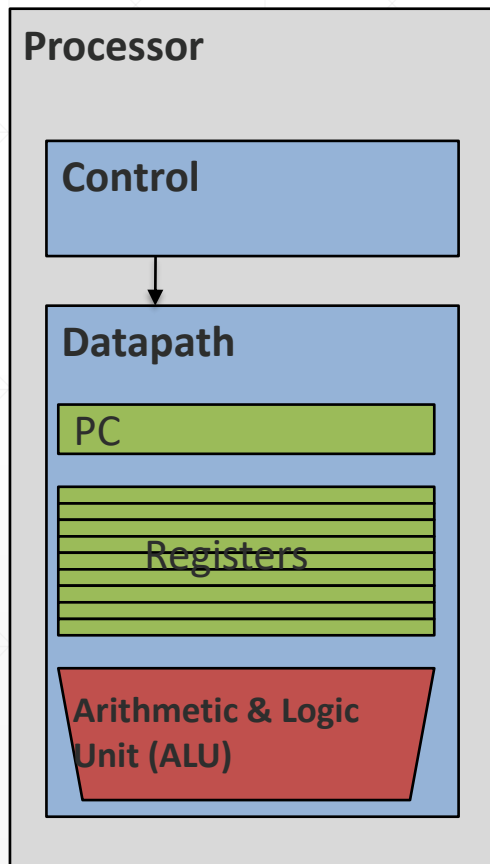
ANDI

- AND as an example
 - 2 32-bit inputs A and B
 - 1 32-bit output C



Useful building blocks

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



ADD

SUB

SLL

SLT

SLTU

XOR

SRL

SRA

OR

AND

ADDI

SLTI

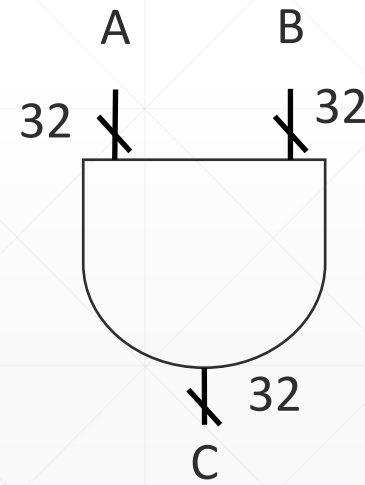
SLTIU

XORI

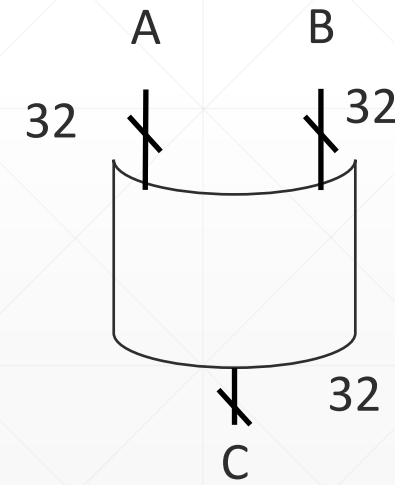
ORI

ANDI

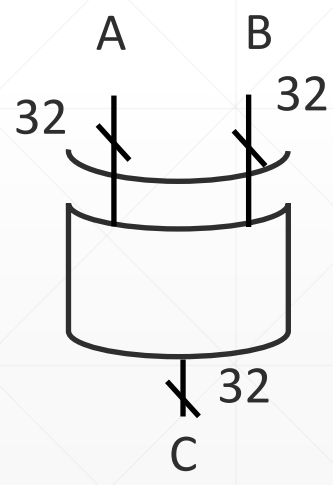
- AND as an example
 - 2 32-bit inputs A and B
 - 1 32-bit output C



A simplified AND
gate array symbol



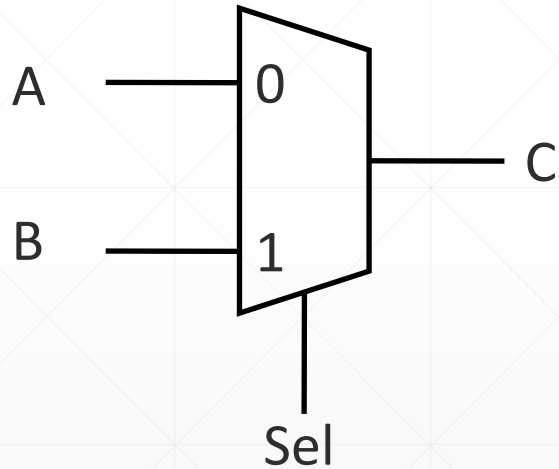
OR



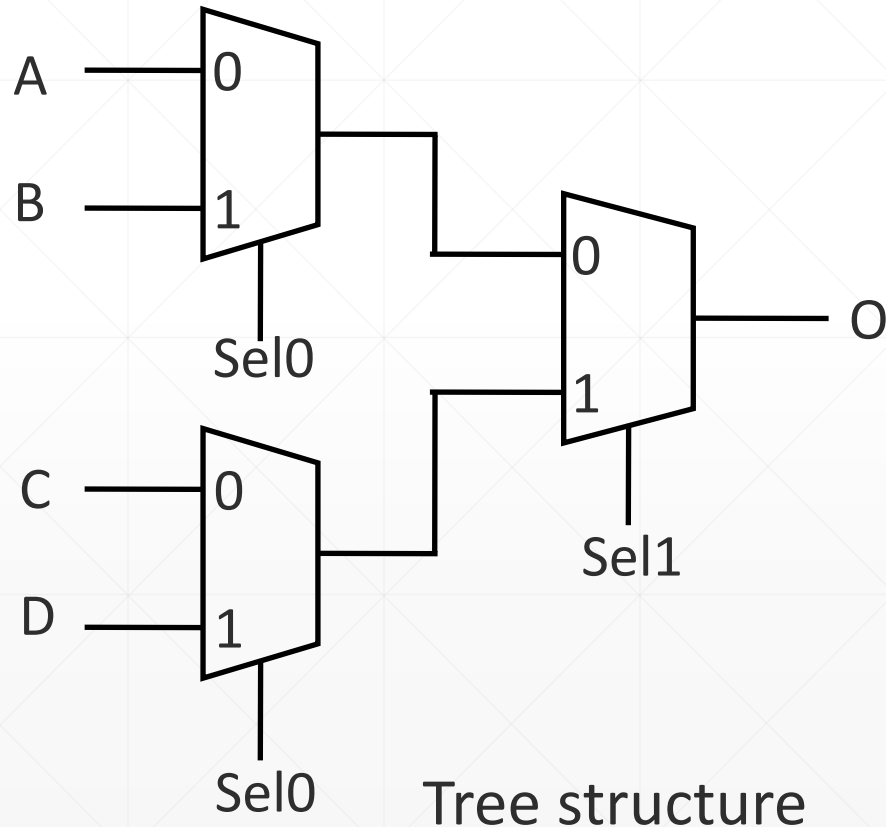
XOR

Useful Combinational Circuits

- Multiplexer (2-to-1)

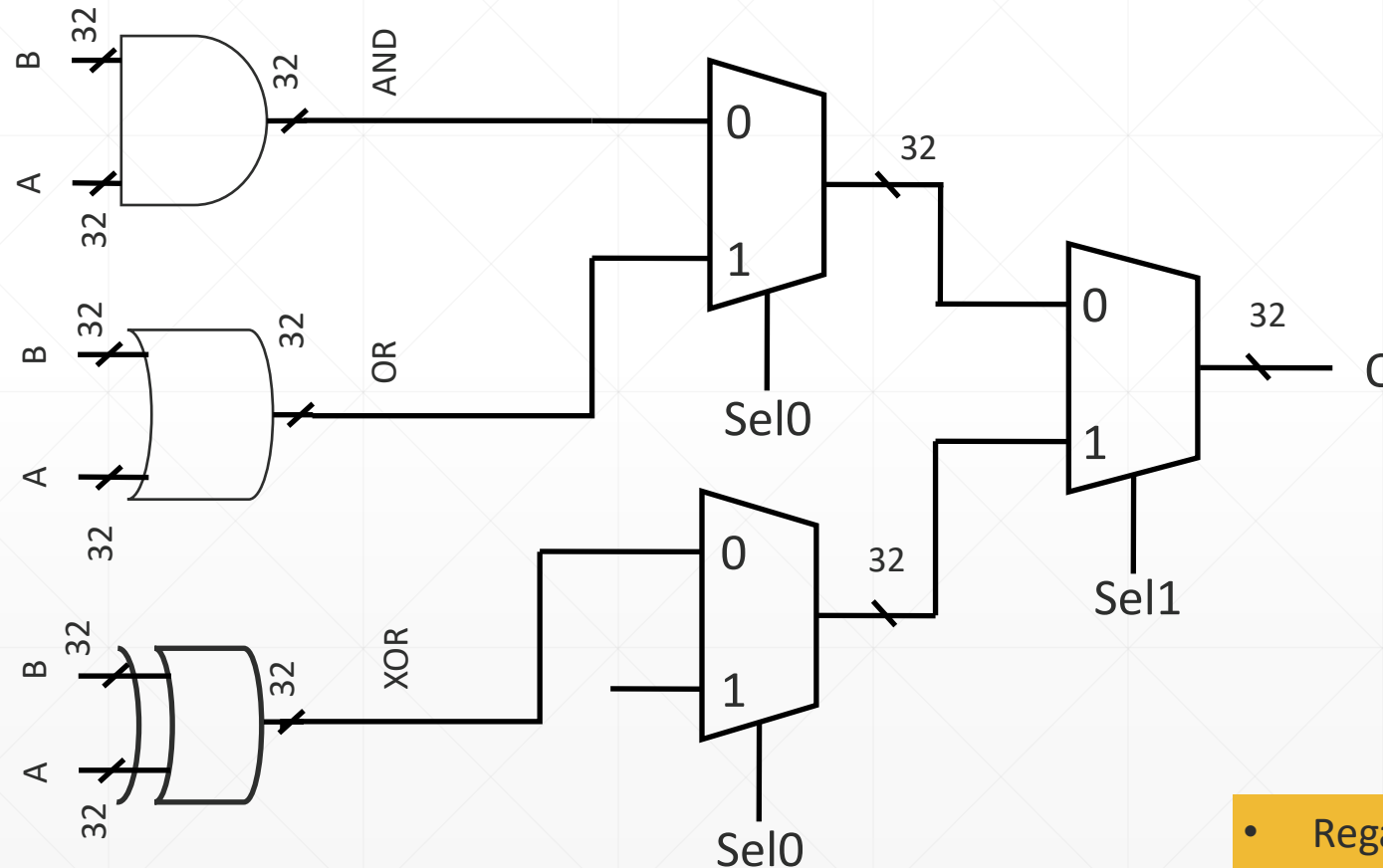


- Multiplexer (2^n -to-1)



Control through selection

- 32-bit Multiplexer and logic gates to support some logic instructions

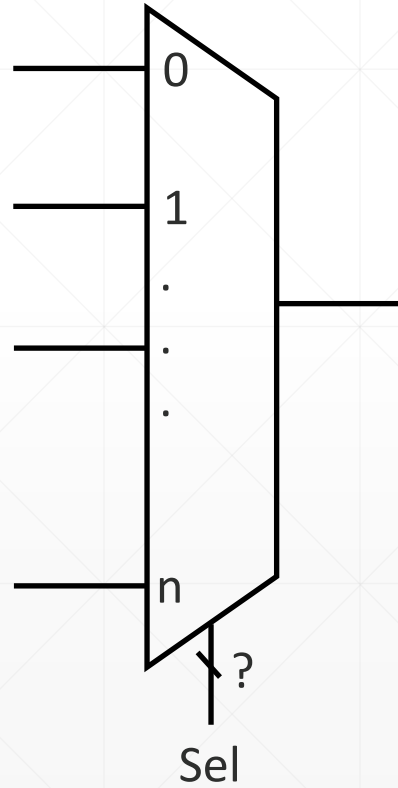


- Regard $(Sel1 Sel0)_2$ as an unsigned number

- More layers of multiplexer to select from more inputs

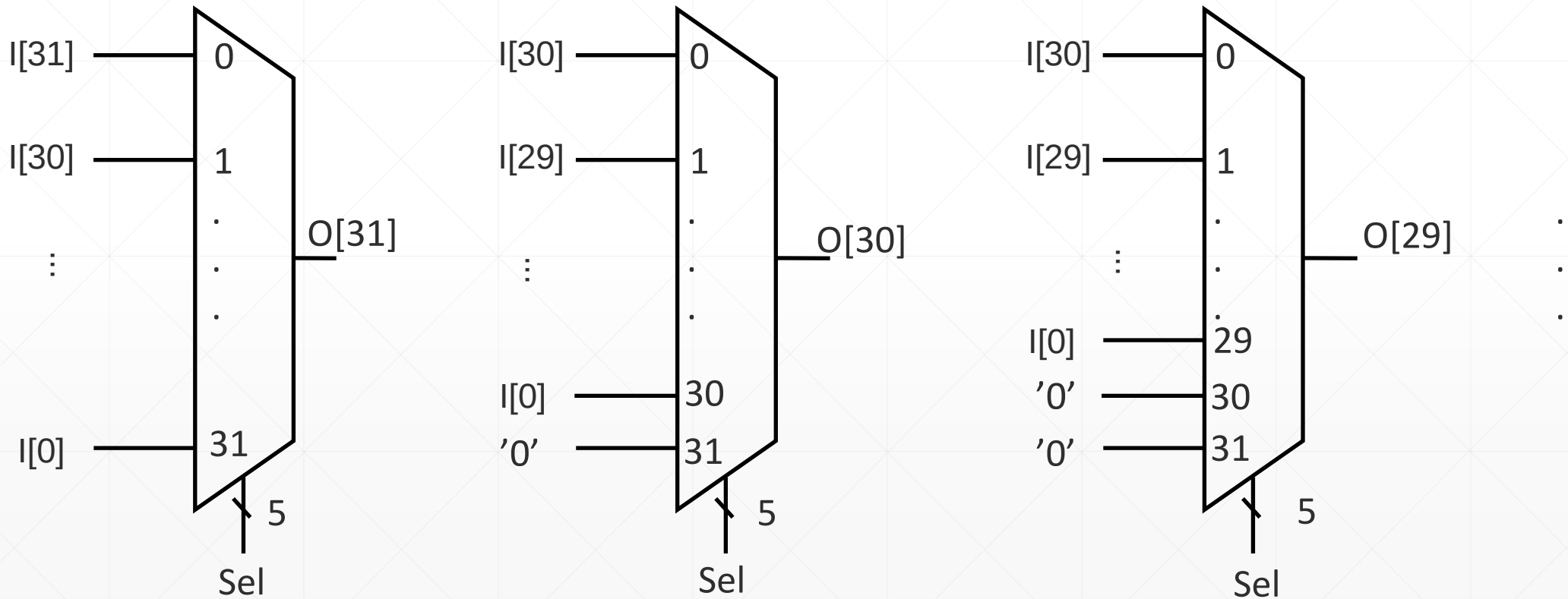
Multiplexer

- n-to-1 multiplexer symbol



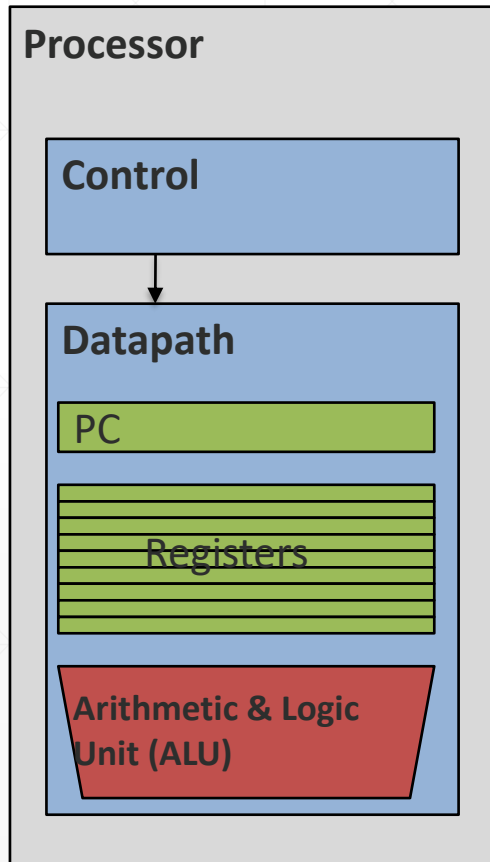
Multiplexers used for shifter

- Left shift a single bit -> left shift multiple single bits
- Other shifter designs such as barrel shifter



Useful building blocks

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



ADD

SUB

SLL

SLT

SLTU

XOR

SRL

SRA

OR

AND

ADDI

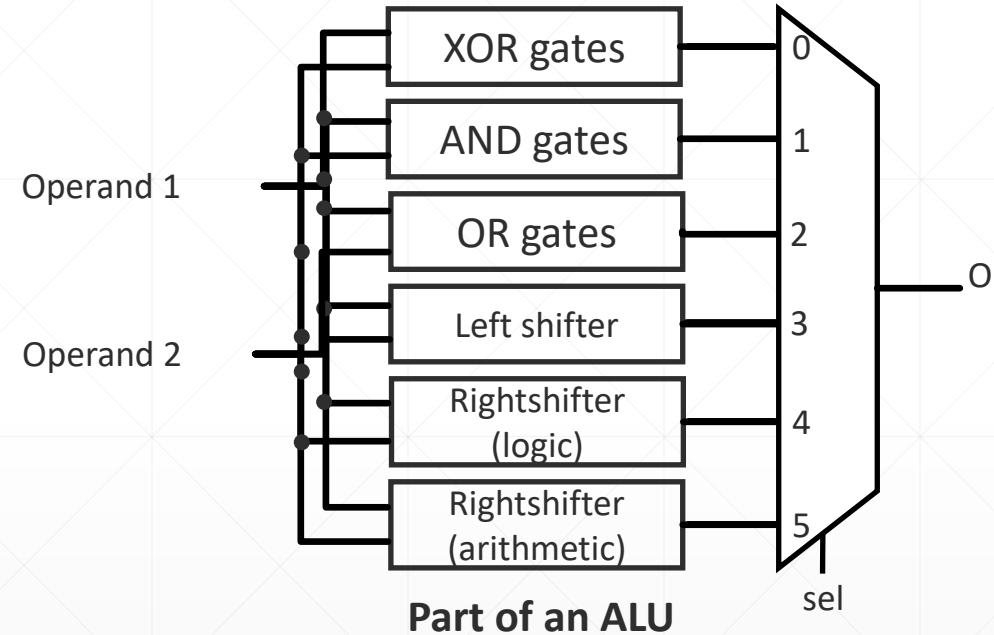
SLTI

SLTIU

XORI

ORI

ANDI

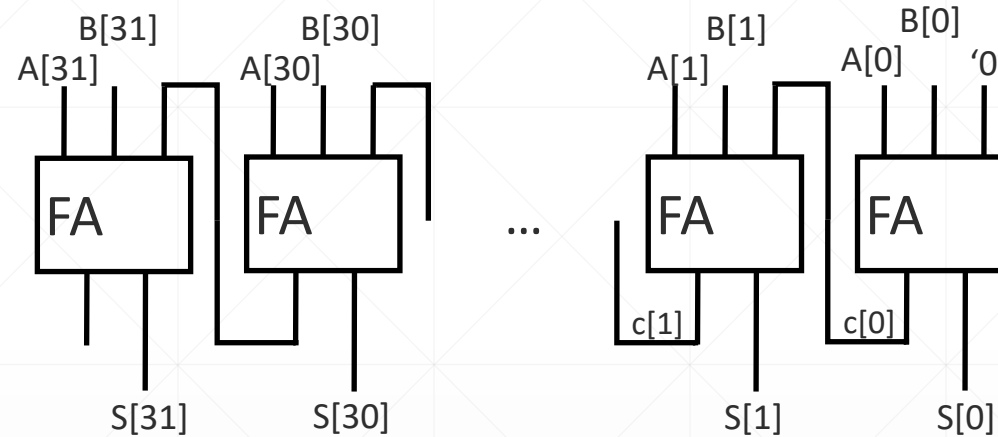


Note that all the signals except the selection signals are 32-bit.

- 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).

Adder & subtractor

- An adder design

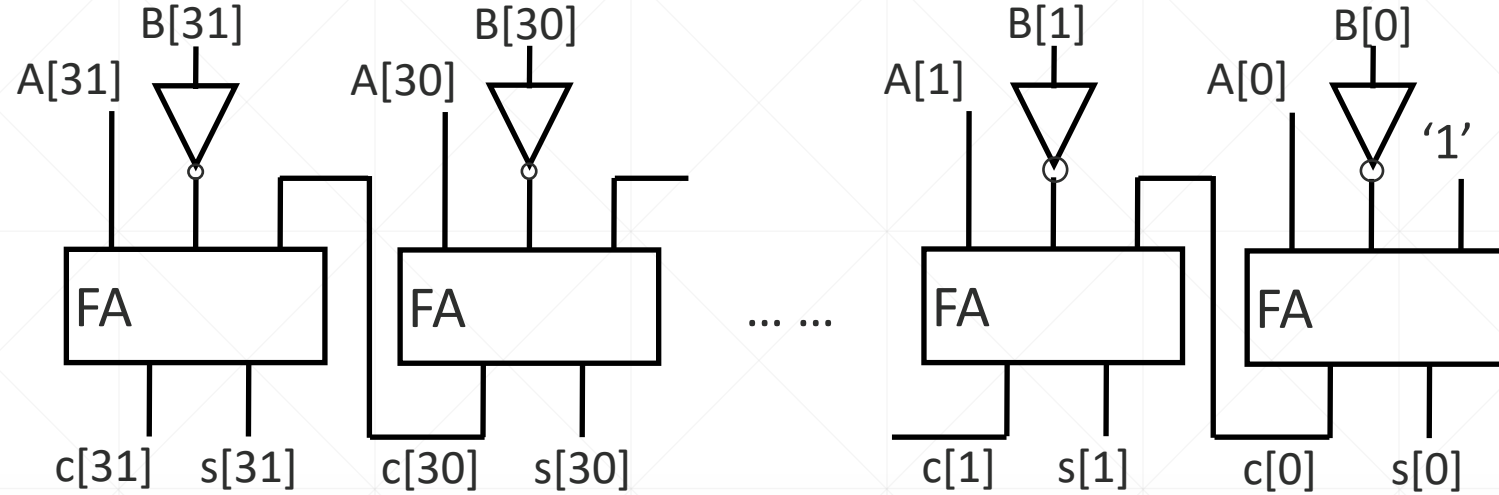


A 32-bit adder

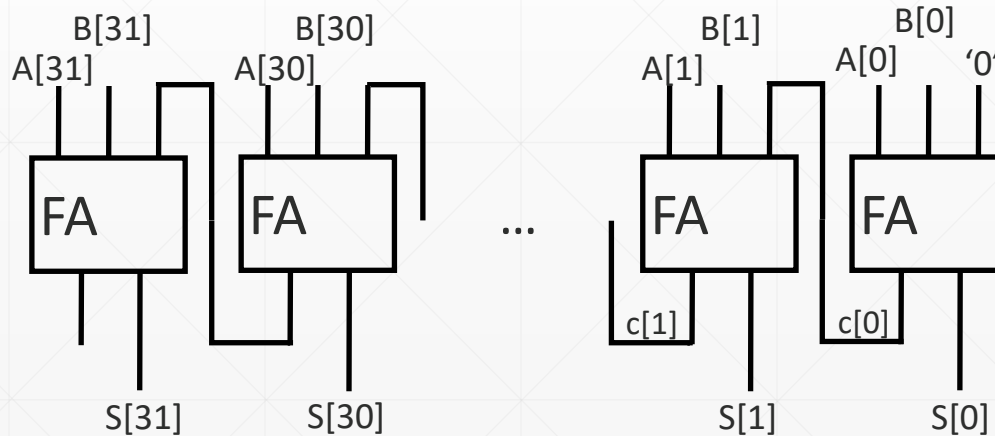
- A smart subtractor design
 - Recall that subtracting a number is equivalent to adding its negative version

A smart subtractor design

$$A - B = A + (-B) = A + \bar{B} + 1 \quad (\text{mod } 2^{N-1})$$

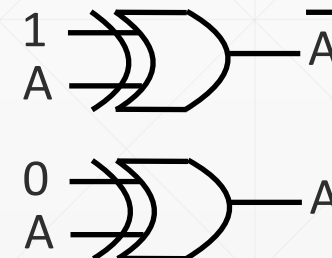


A 32-bit subtractor



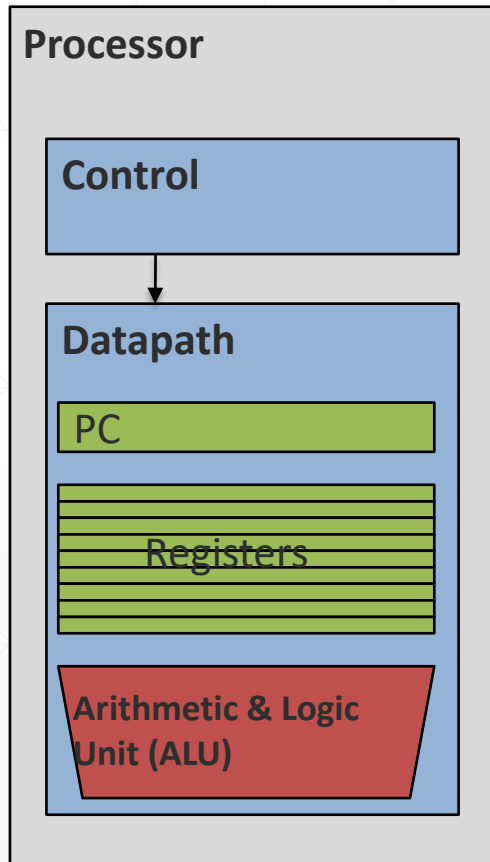
A 32-bit adder

- Recall XOR gate



Useful building blocks

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



ADD

SUB

SLL

SLT

SLTU

XOR

SRL

SRA

OR

AND

ADDI

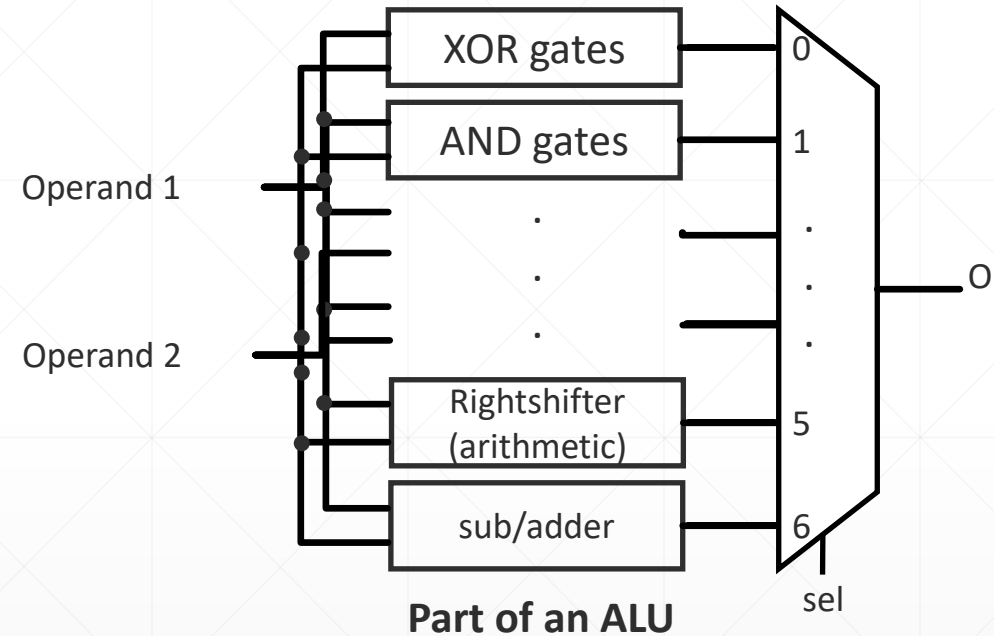
SLTI

SLTIU

XORI

ORI

ANDI

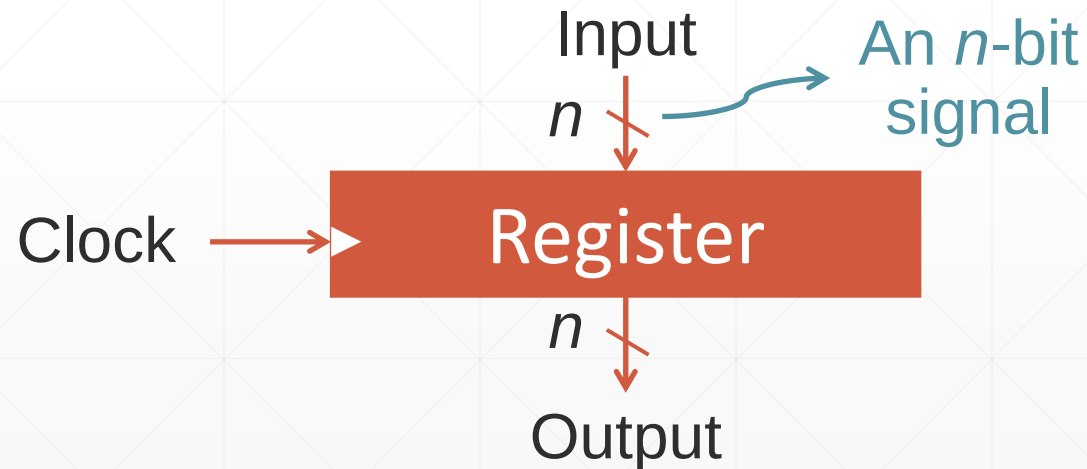
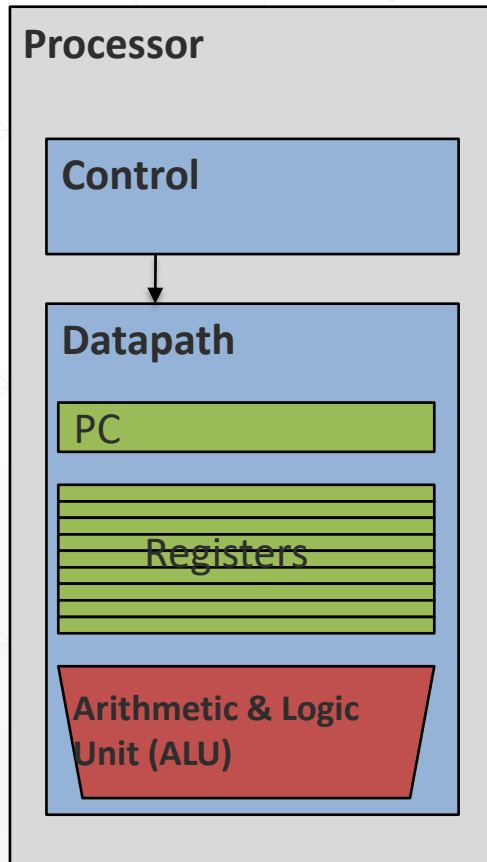


Note that all the signals except the selection signals are 32-bit.

- 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).

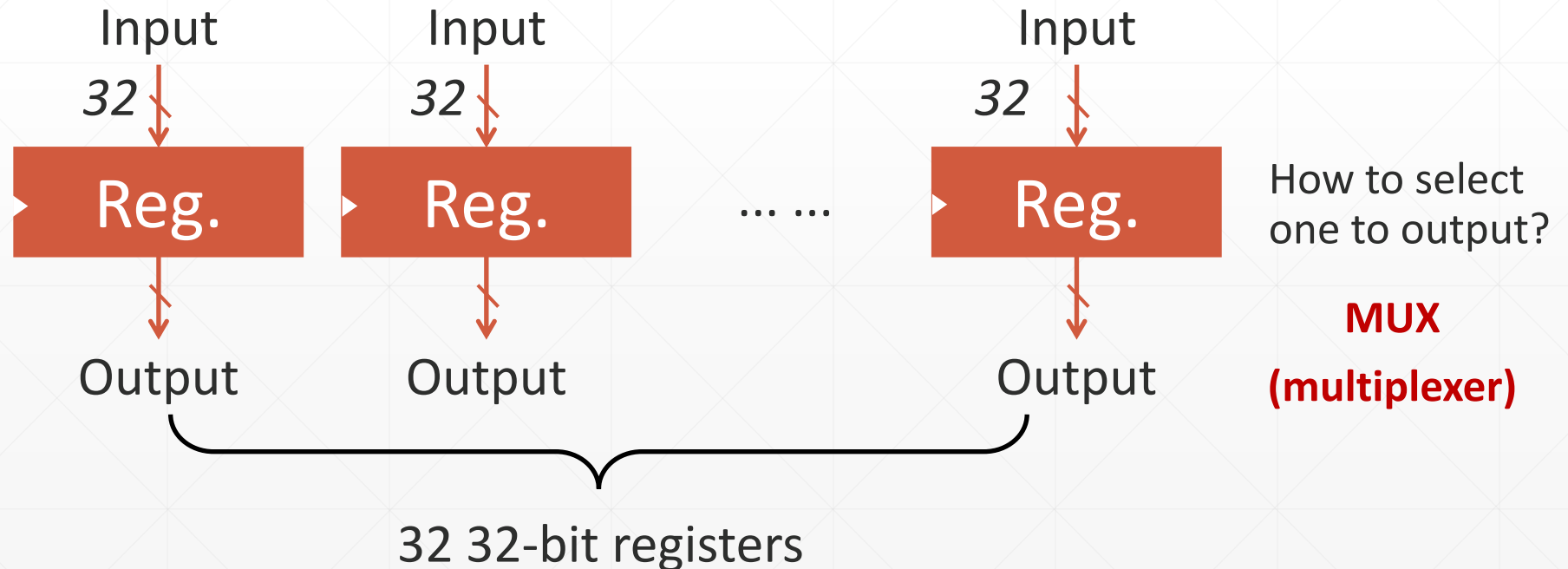
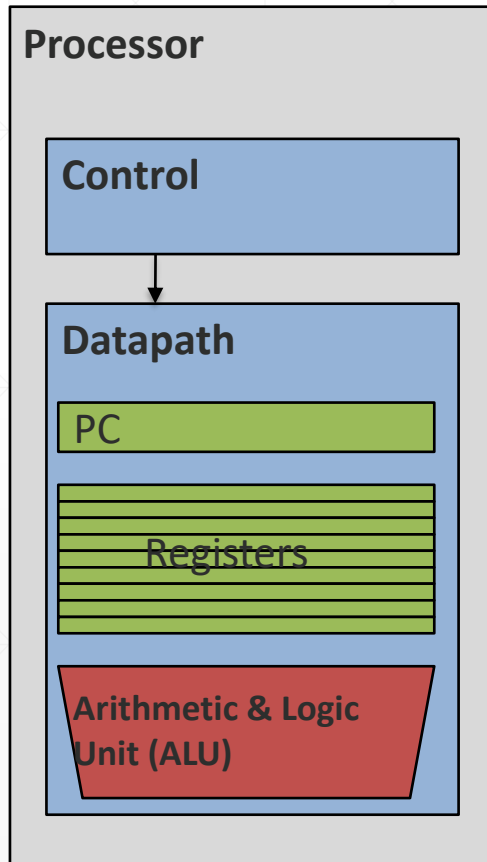
Useful building blocks – register file

- 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
 - Recall we have registers that store values



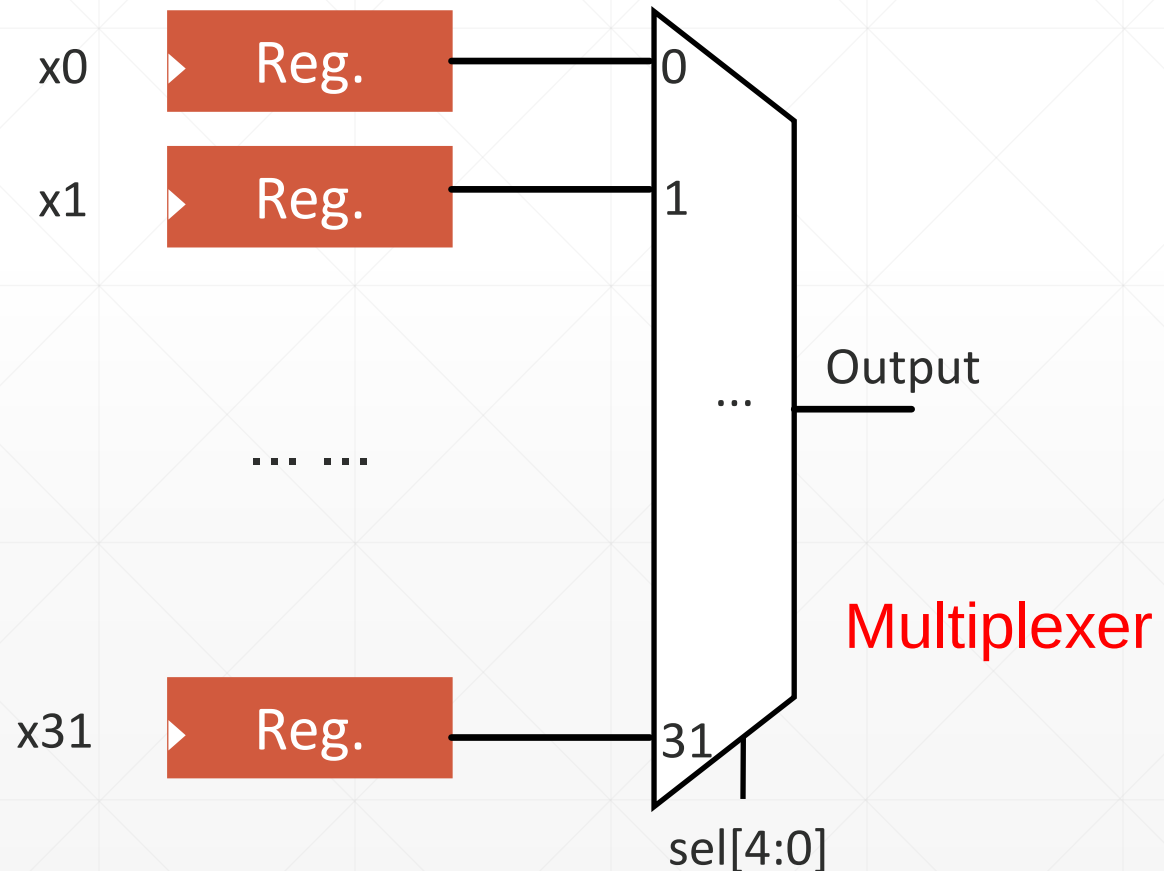
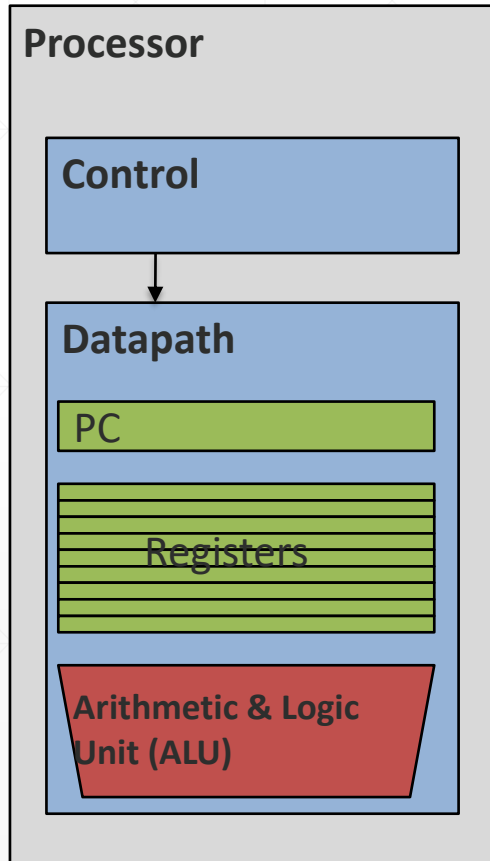
Useful building blocks – register file

- 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
 - Recall we have registers that store values



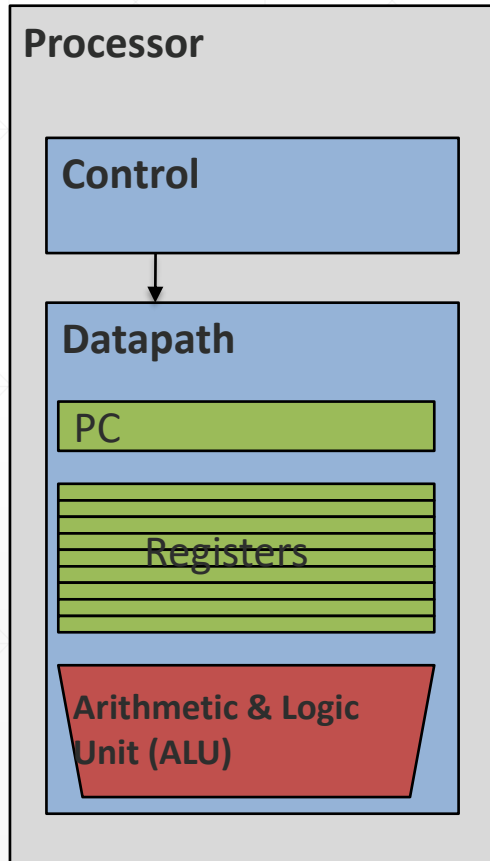
Useful building blocks – register file

- 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

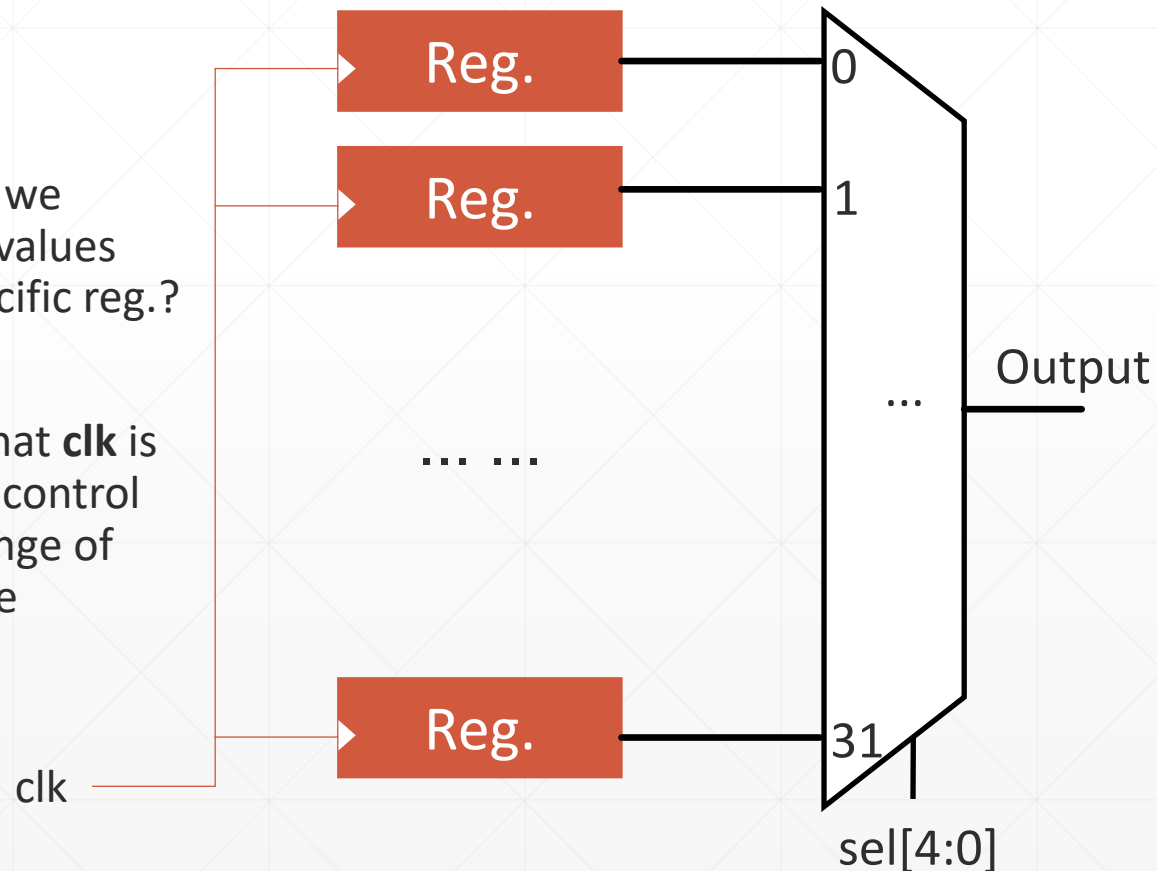


Useful building blocks – register file

- 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

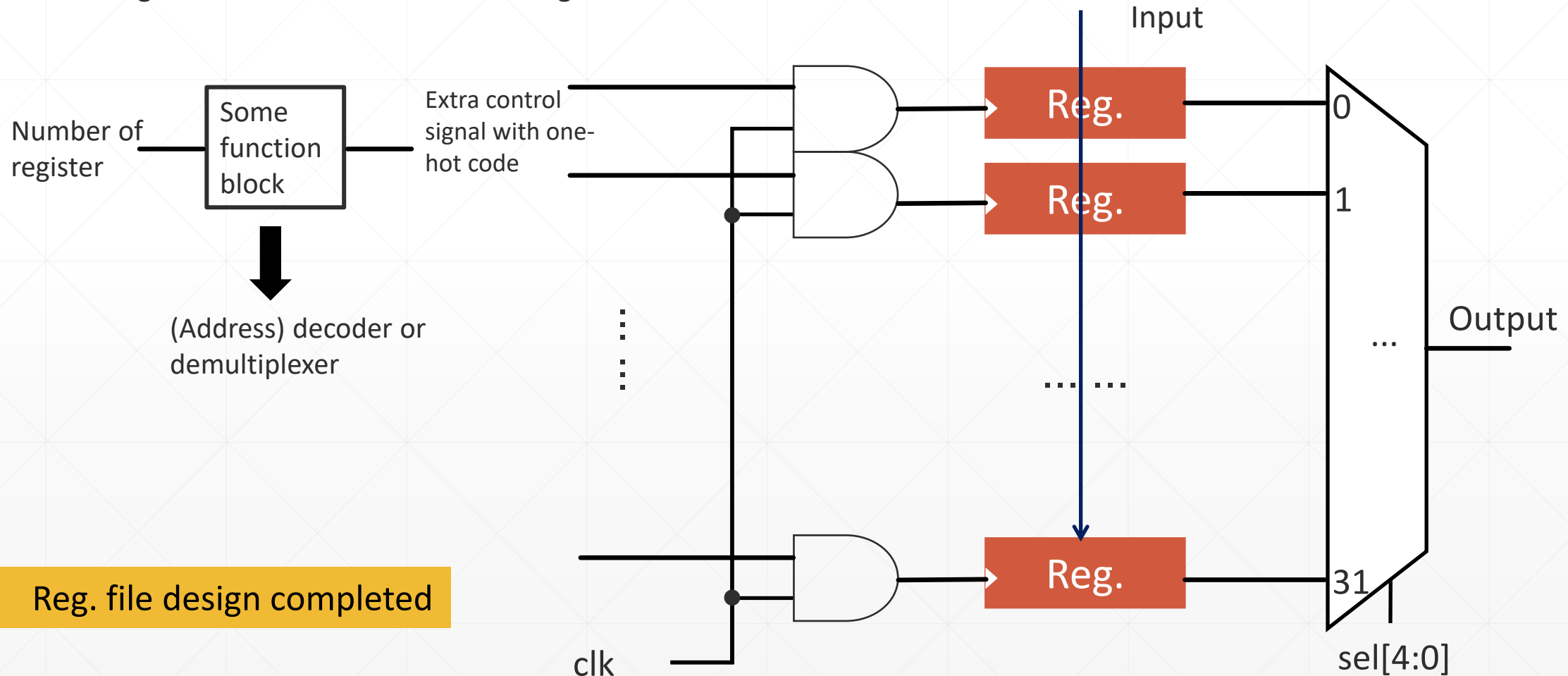


- How do we change values of a specific reg.?
- Recall that **clk** is used to control the change of the state



Useful building blocks – register file

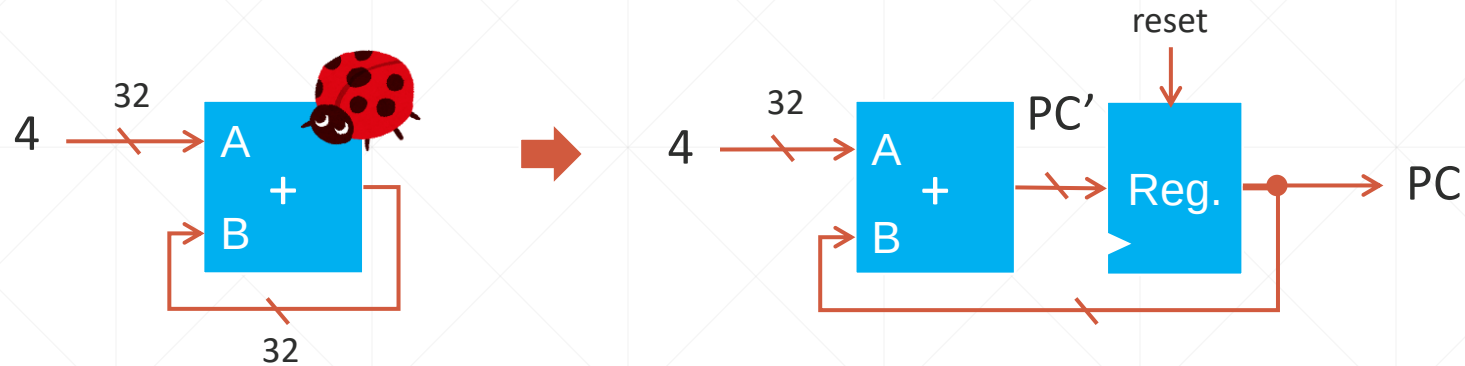
- The register file is the component that contains all the general purpose registers of the microprocessor
- A register file should provide data given the register numbers
- A register file should be able to change the stored value



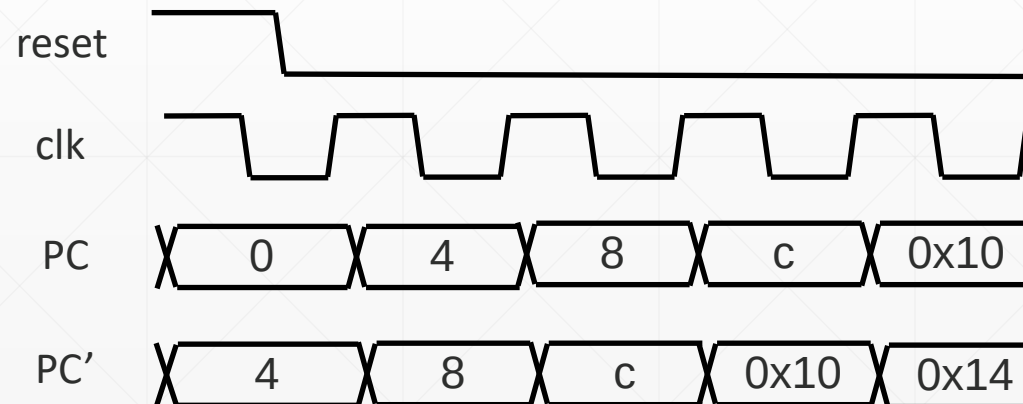
• Reg. file design completed

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

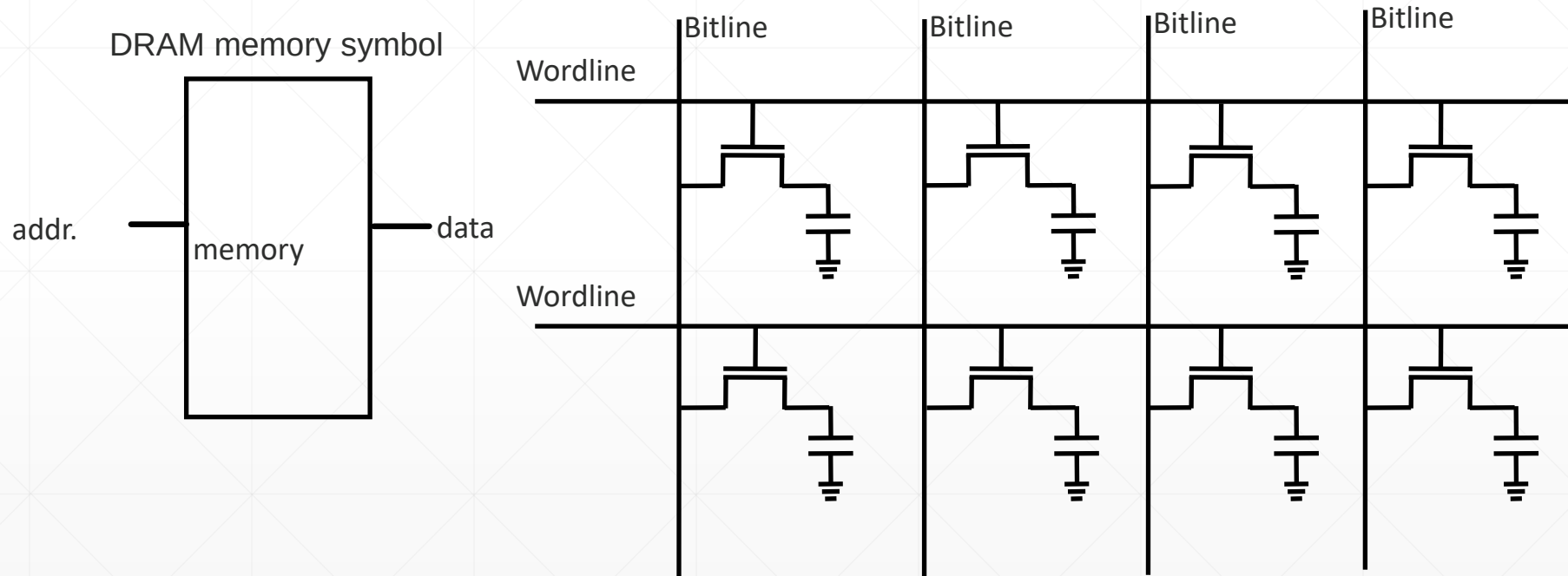


- Timing diagram

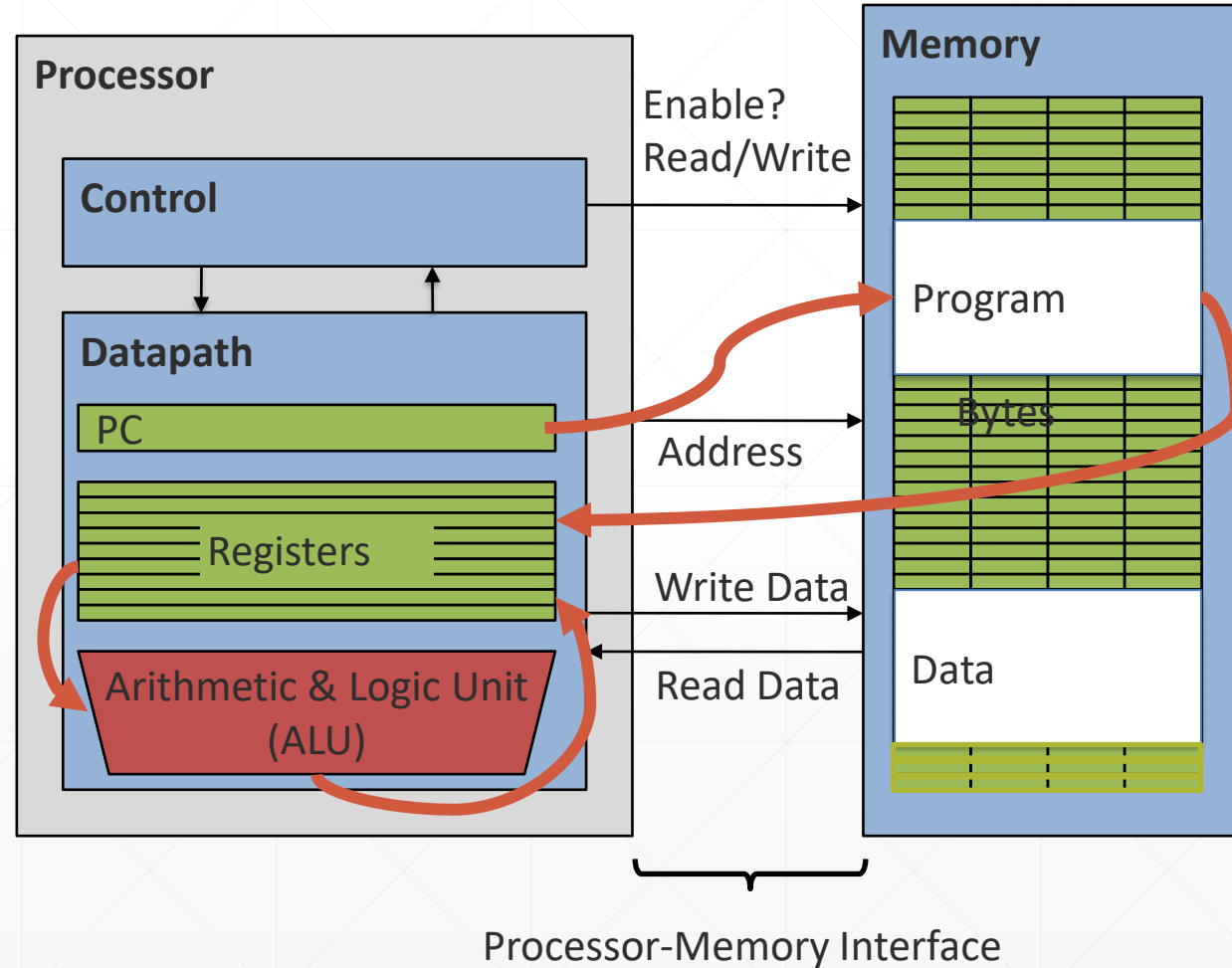


Useful building blocks-Memory

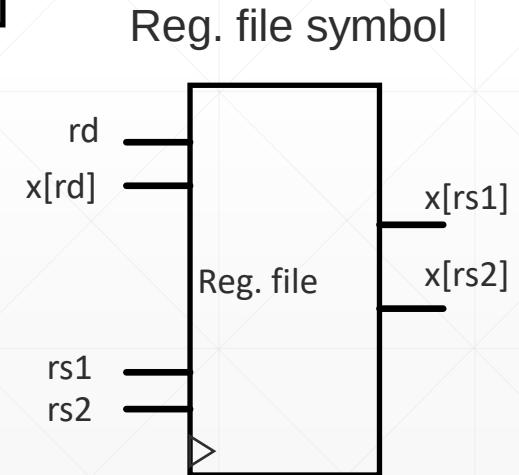
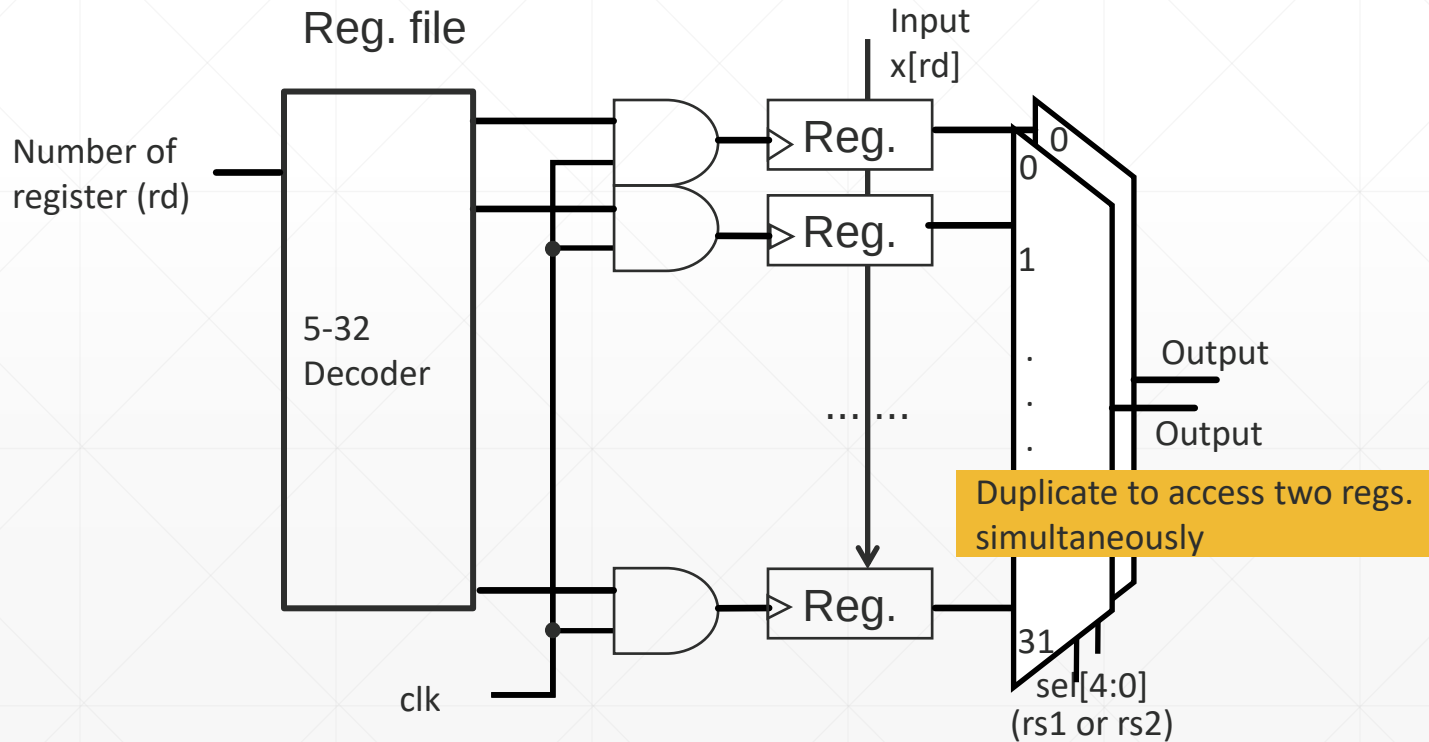
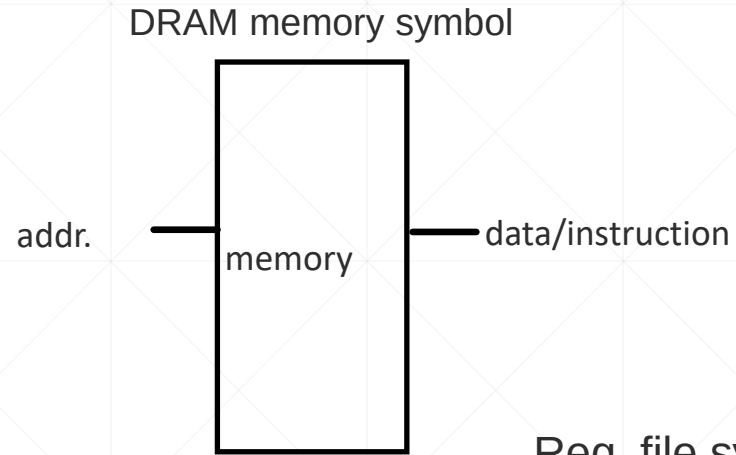
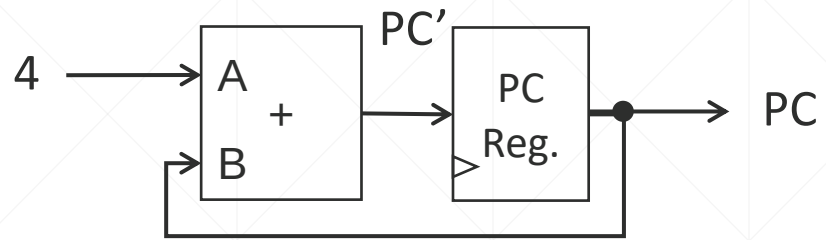
- Memory similar to register file except that the basic cell design is different
- Requires refresh for DRAM



Datapath

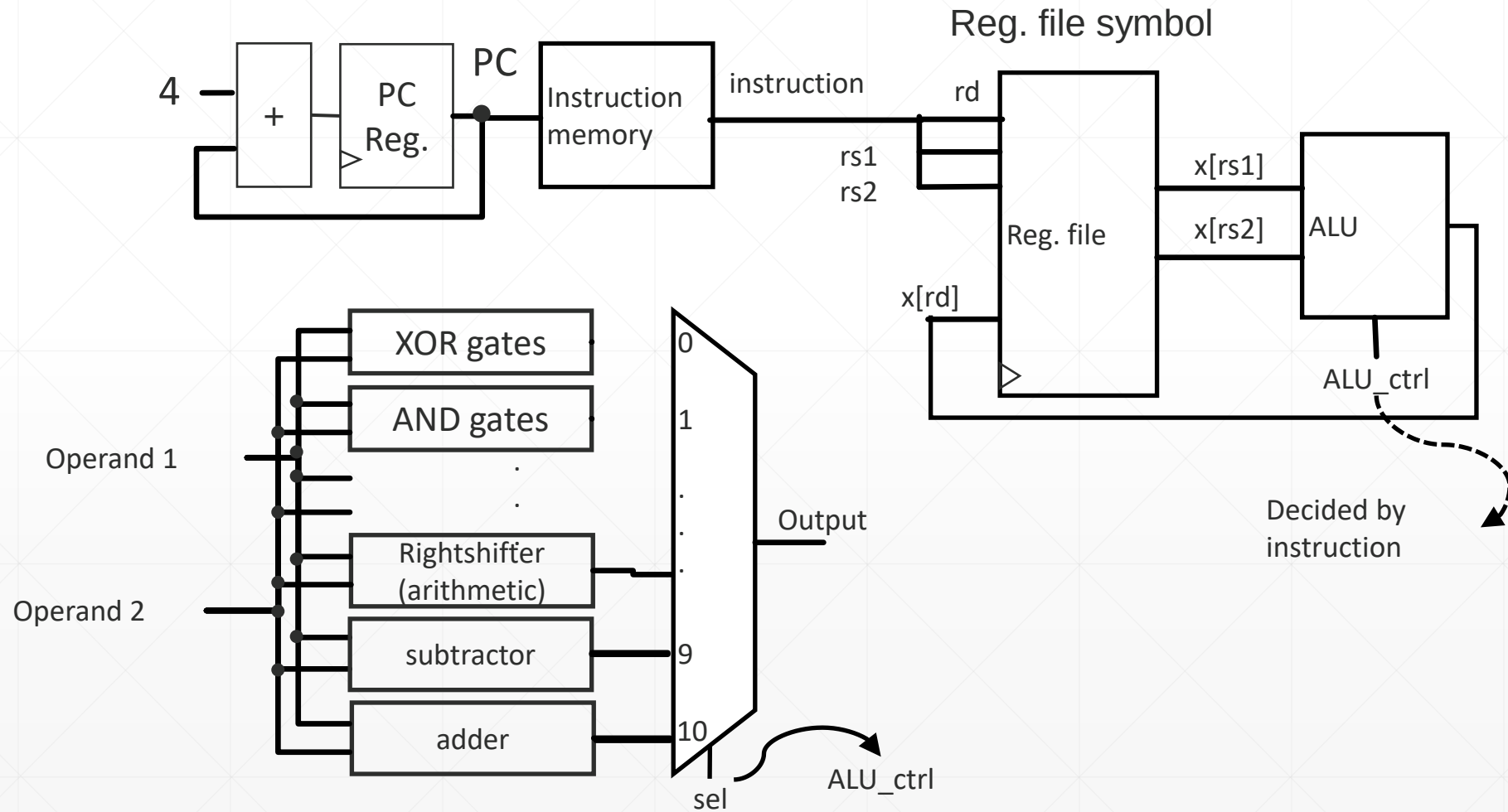


Datapath



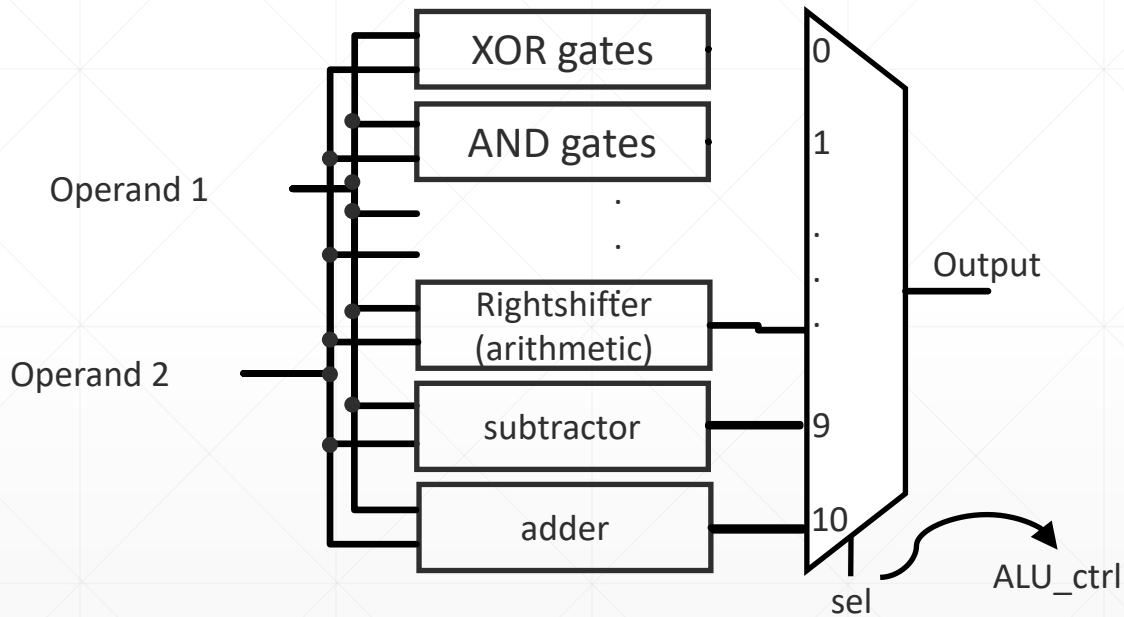
Datapath for R-type

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



Datapath for R-type

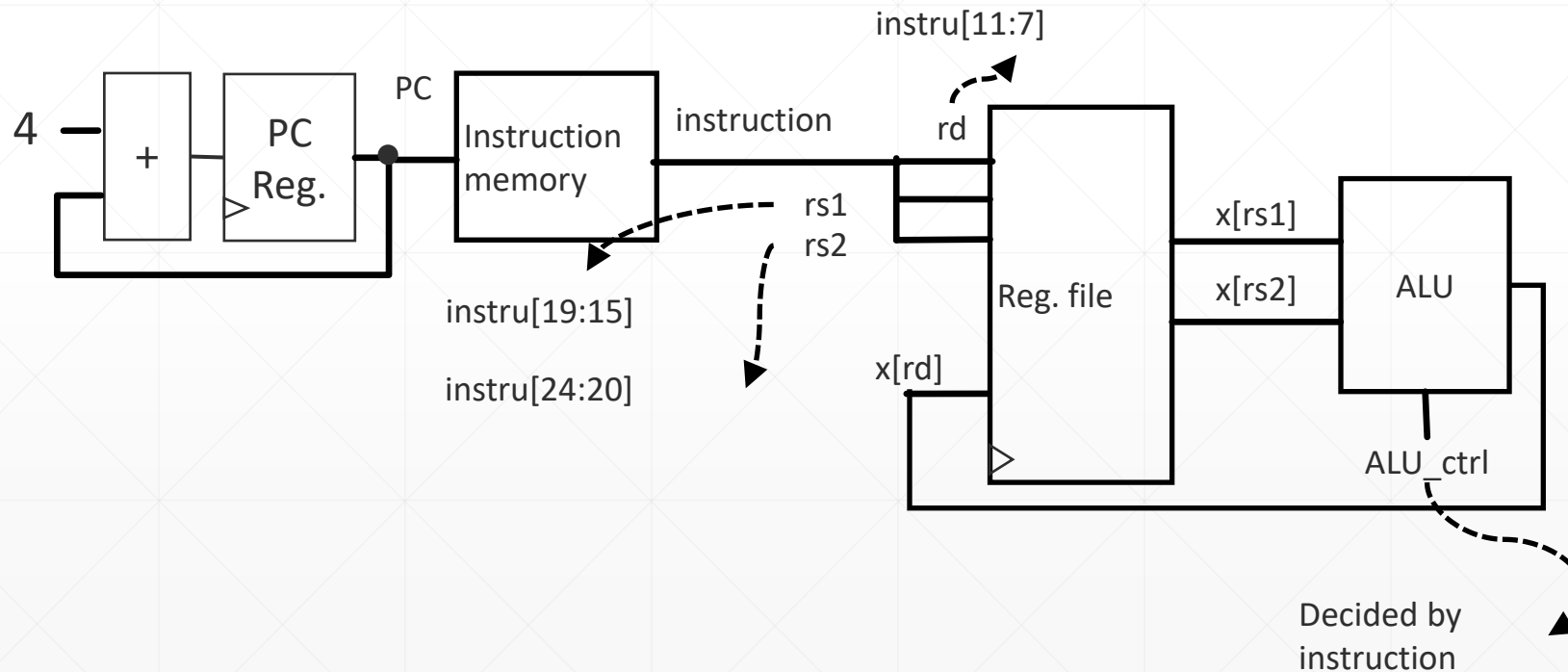
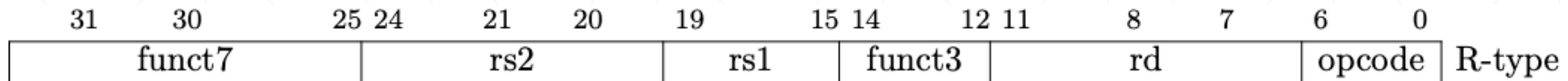
- We have all the building blocks to execute R-type instructions



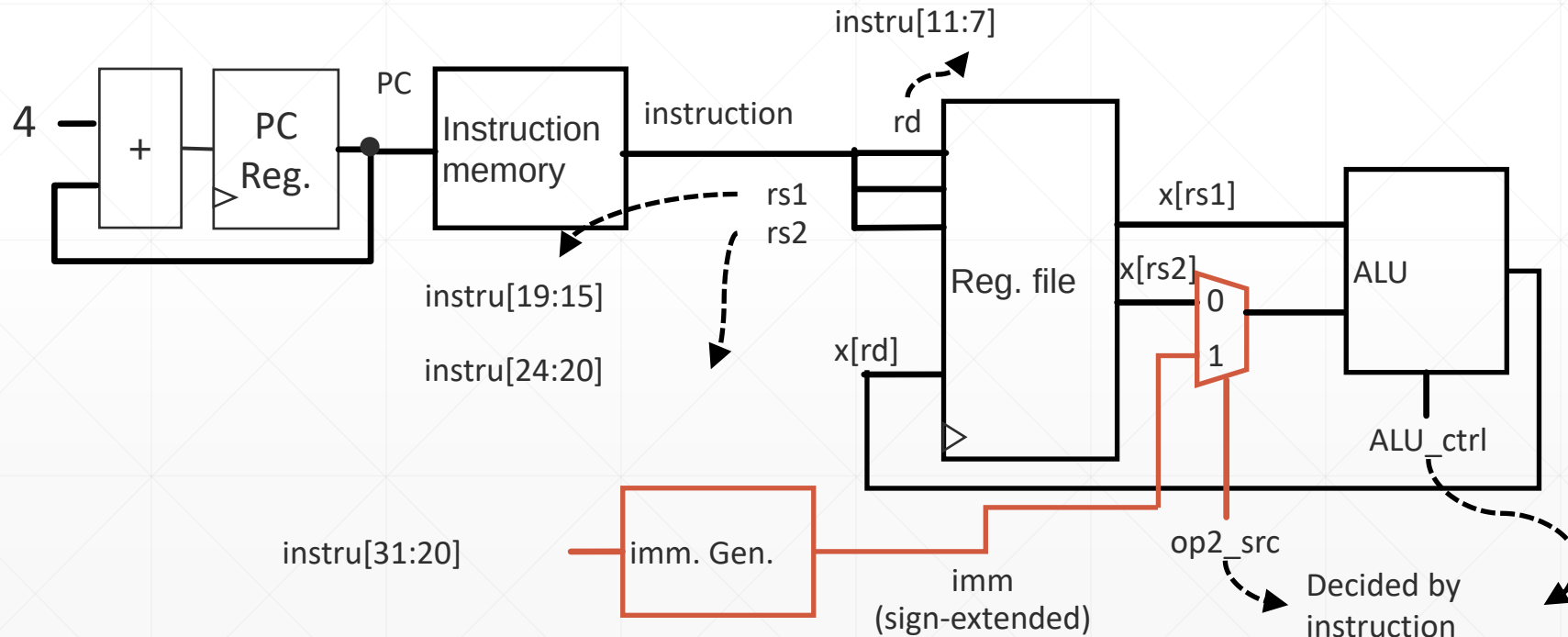
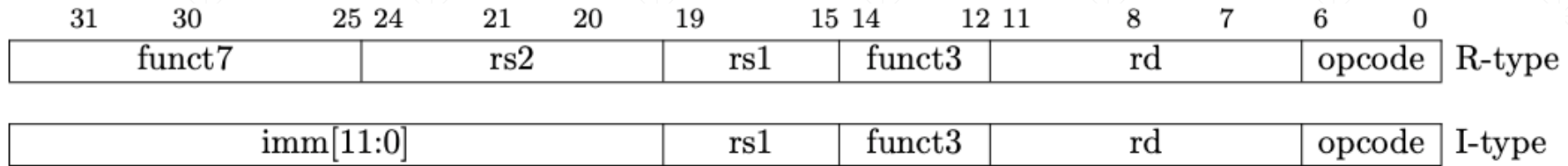
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110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

Datapath for R-type

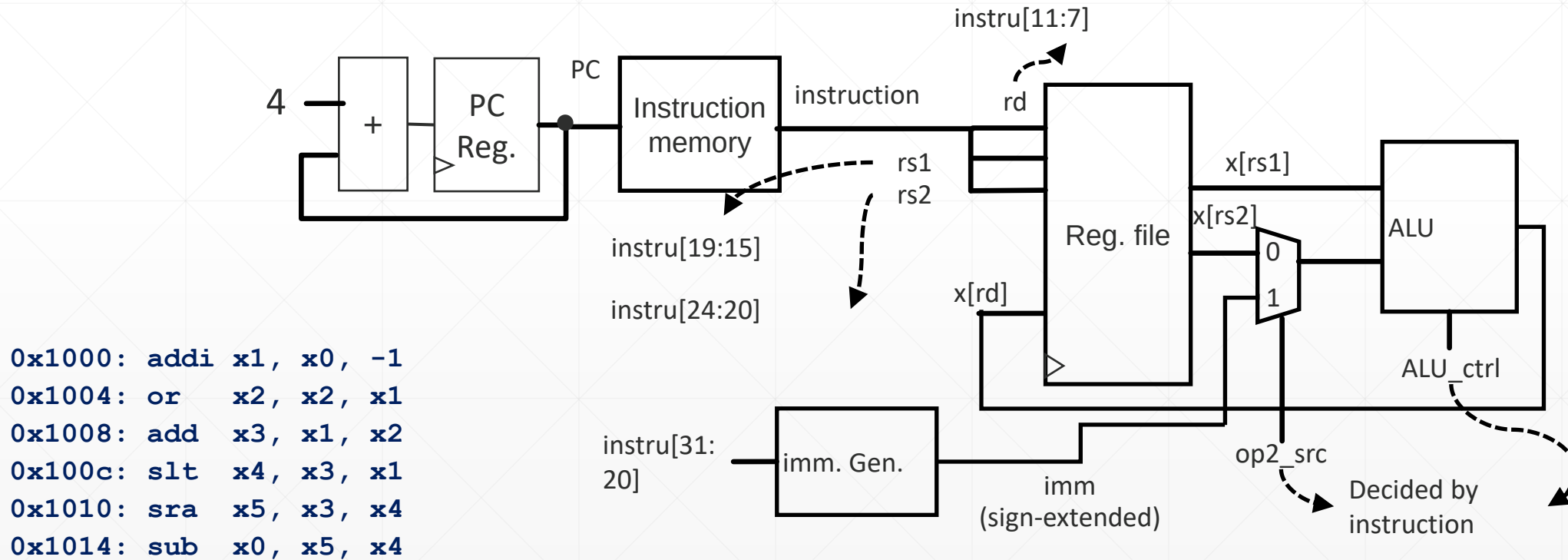
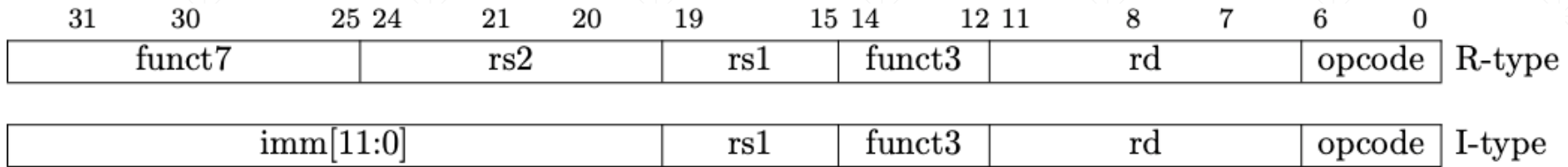
- We have all the building blocks to execute R-type instructions



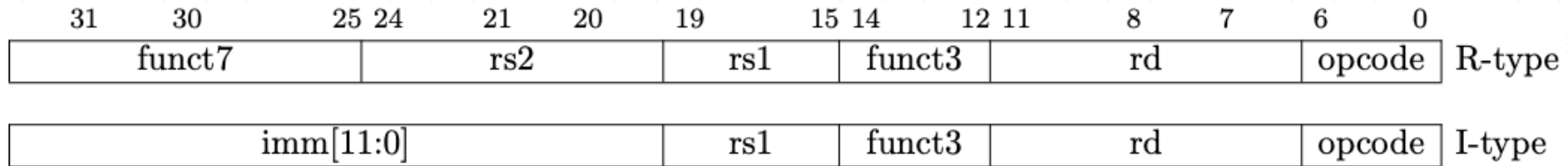
Datapath for I-type arithmetic and logic



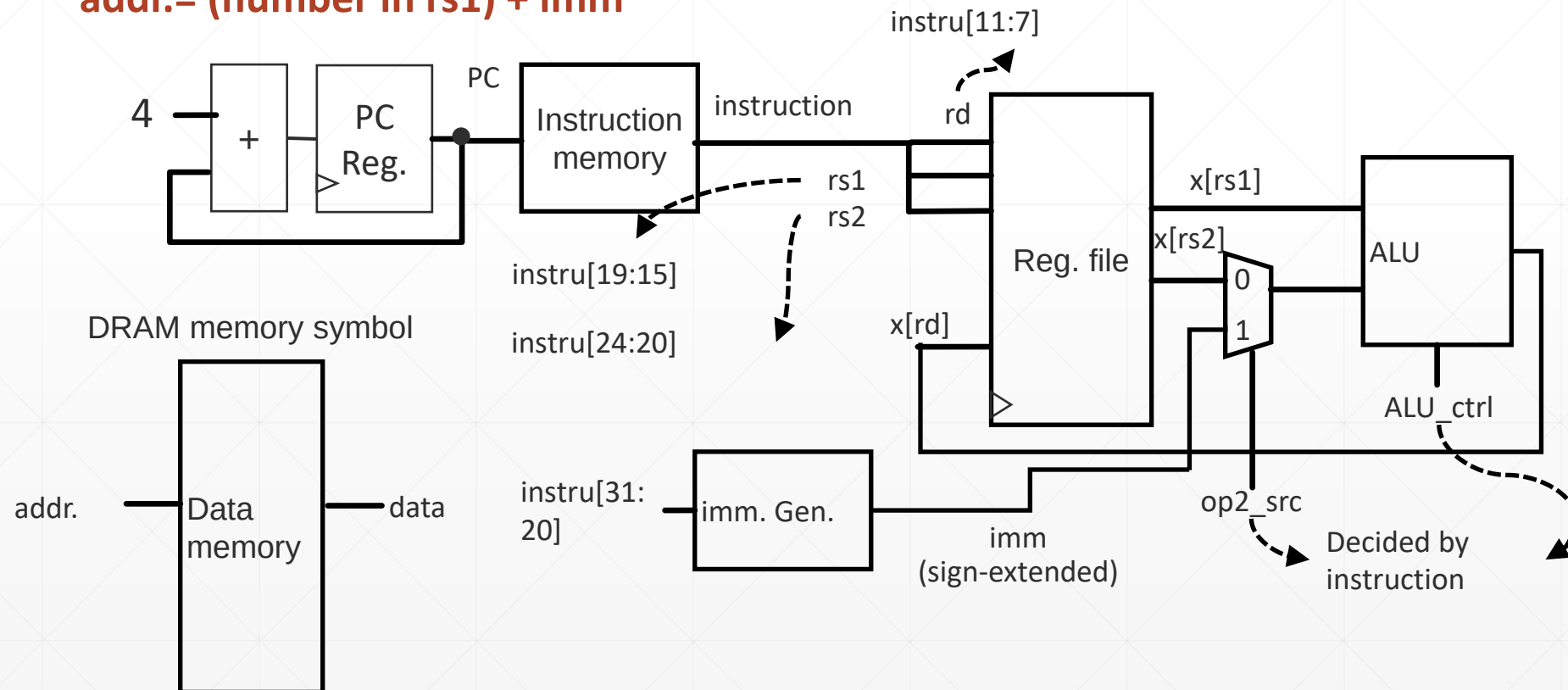
Example



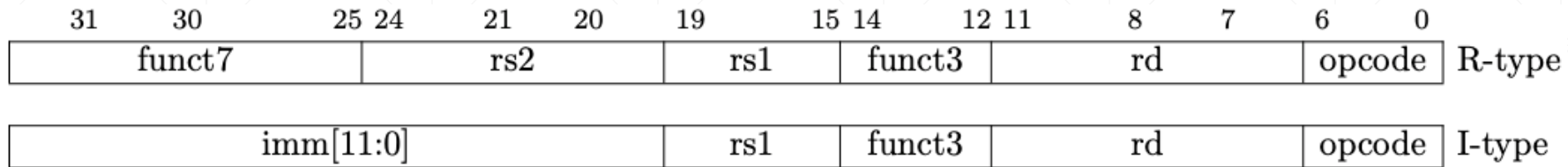
Datapath for more types ...



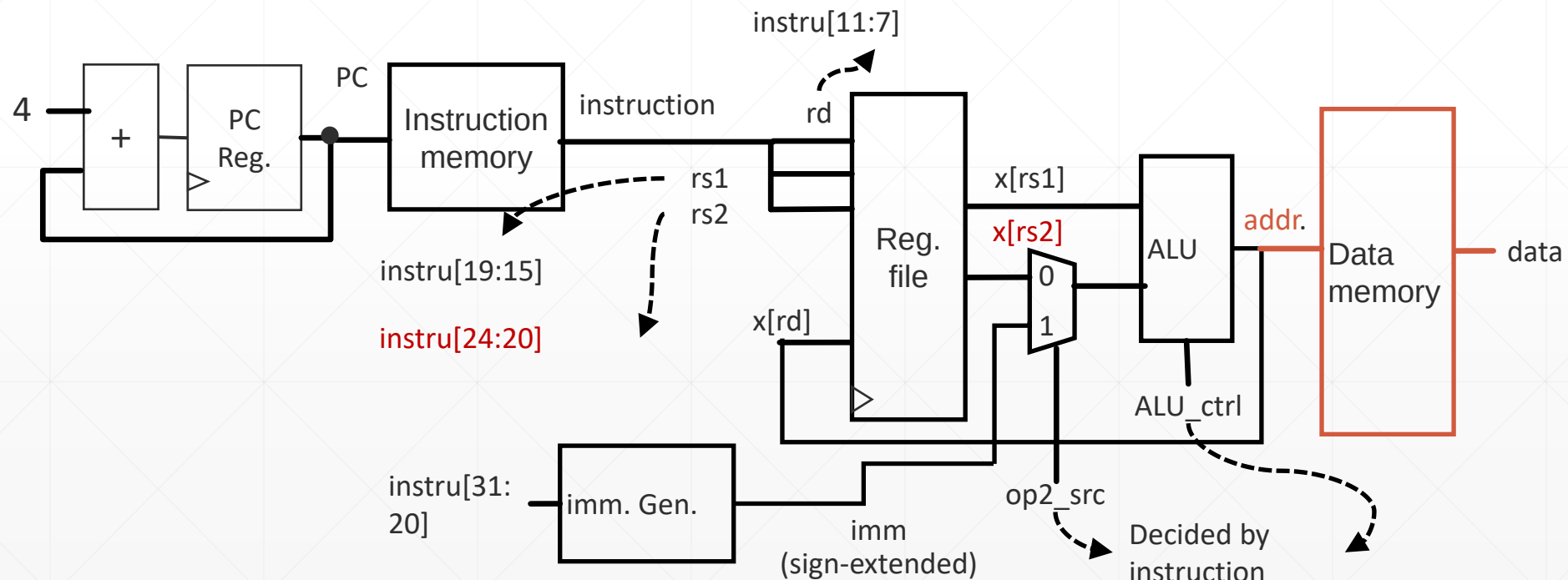
- **lw rd, imm(rs1) : Load word at addr. to register rd**
addr.= (number in rs1) + imm



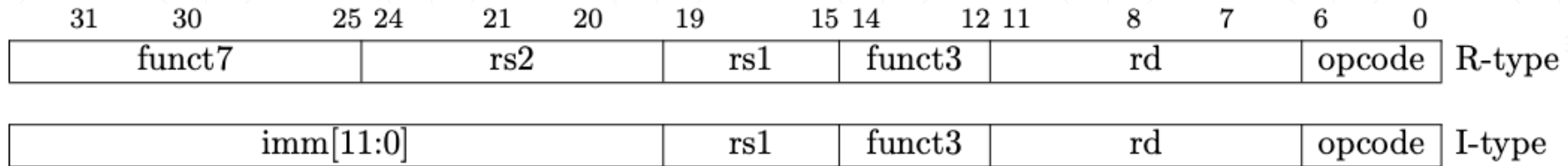
Datapath for I-type load



- lw rd, imm(rs1) : Load word at addr. to register rd
 $\text{addr.} = (\text{number in rs1}) + \text{imm}$

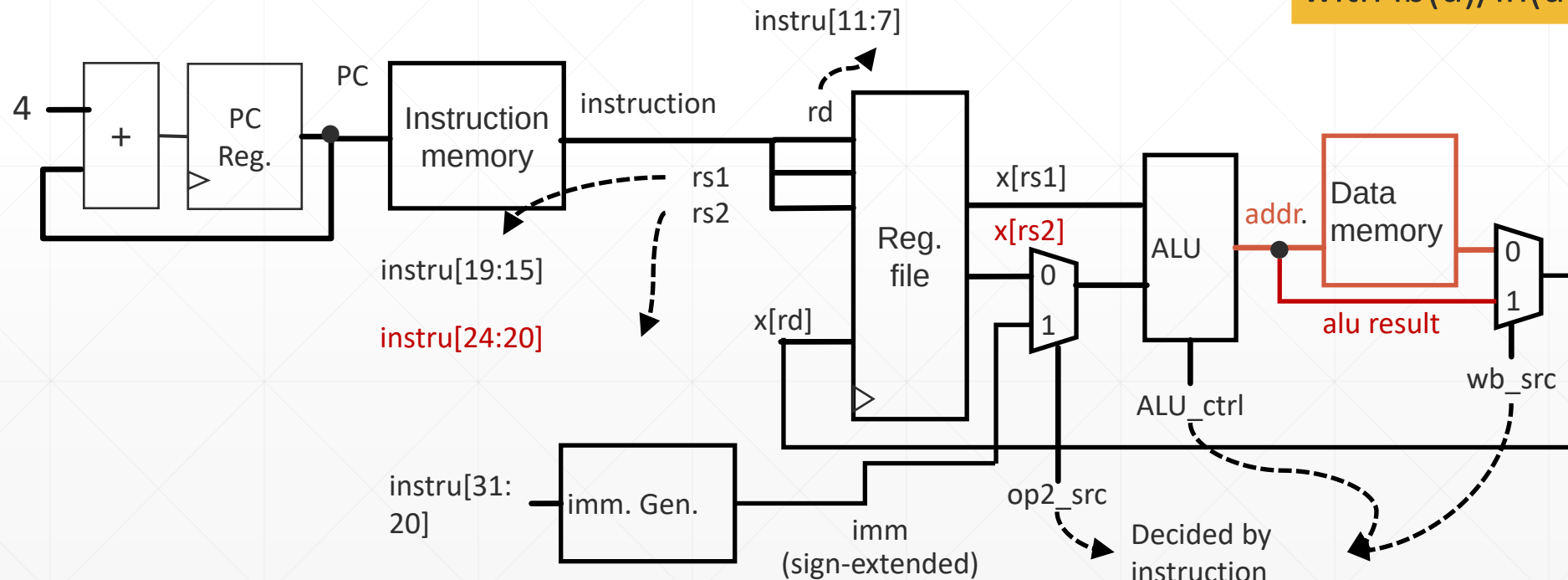


Datapath for I-type load

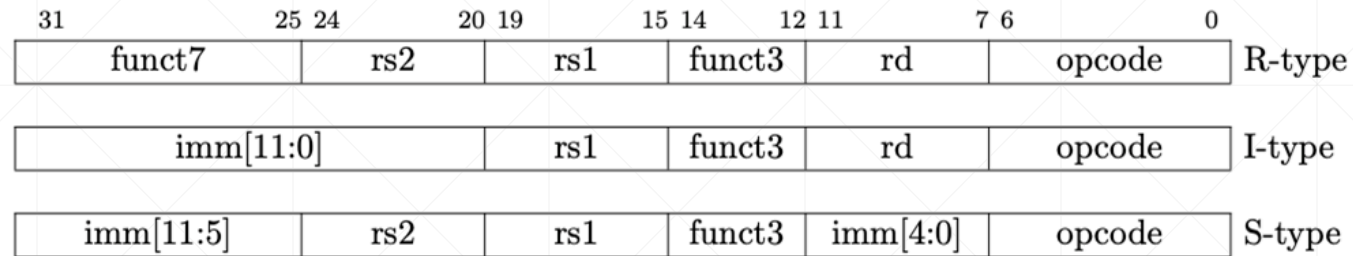


- `lw rd, imm(rs1)` : Load word at addr. to register rd
 $\text{addr.} = (\text{number in rs1}) + \text{imm}$

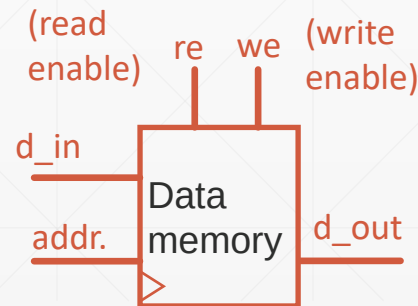
How would you deal with `lb(u)/lh(u)`?



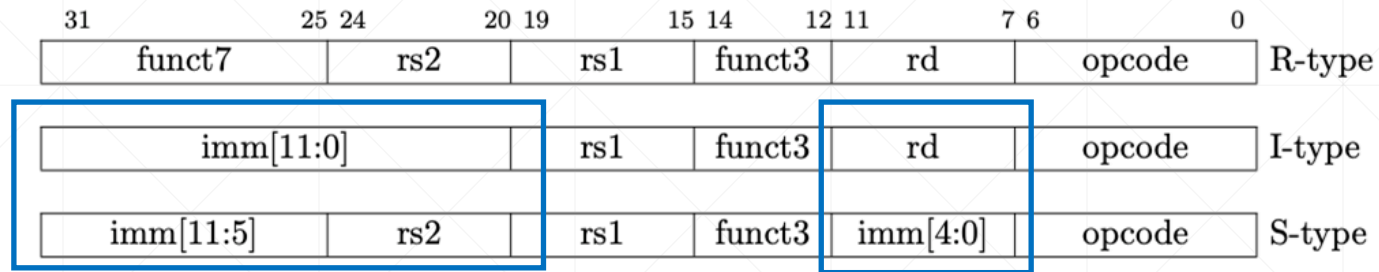
Datapath for S-type store



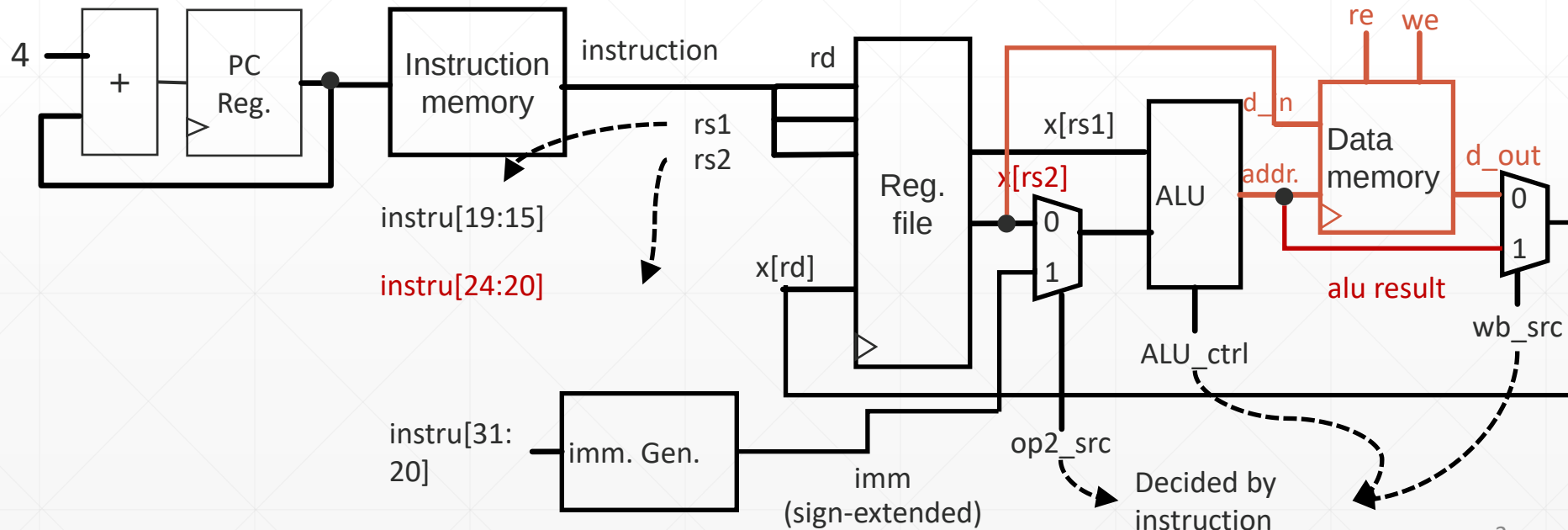
- Recall that in an FSM, only when there is a trigger (clk edge), the state can change.
- We assume that the change of data memory (memory-write) is also governed by clk edge.
- Assume behavior model of data memory:
 - When $we=1$ & $re=0$, at clk rising edge, $data[addr.] = d_in$; d_out stays at high-resistance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



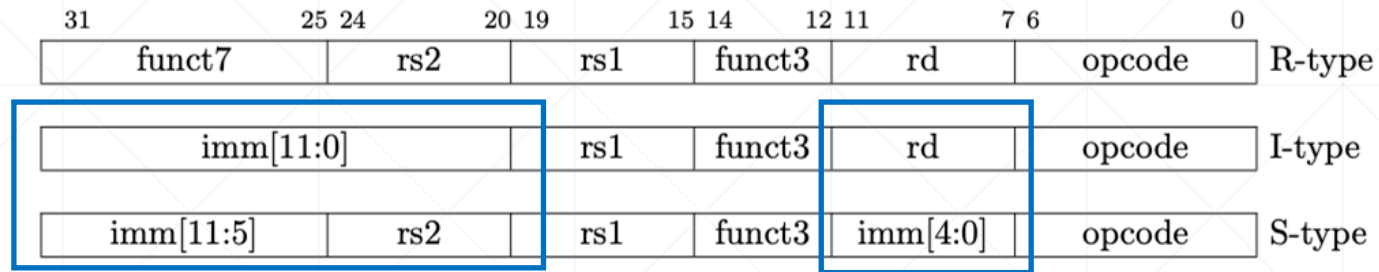
Datapath for S-type store



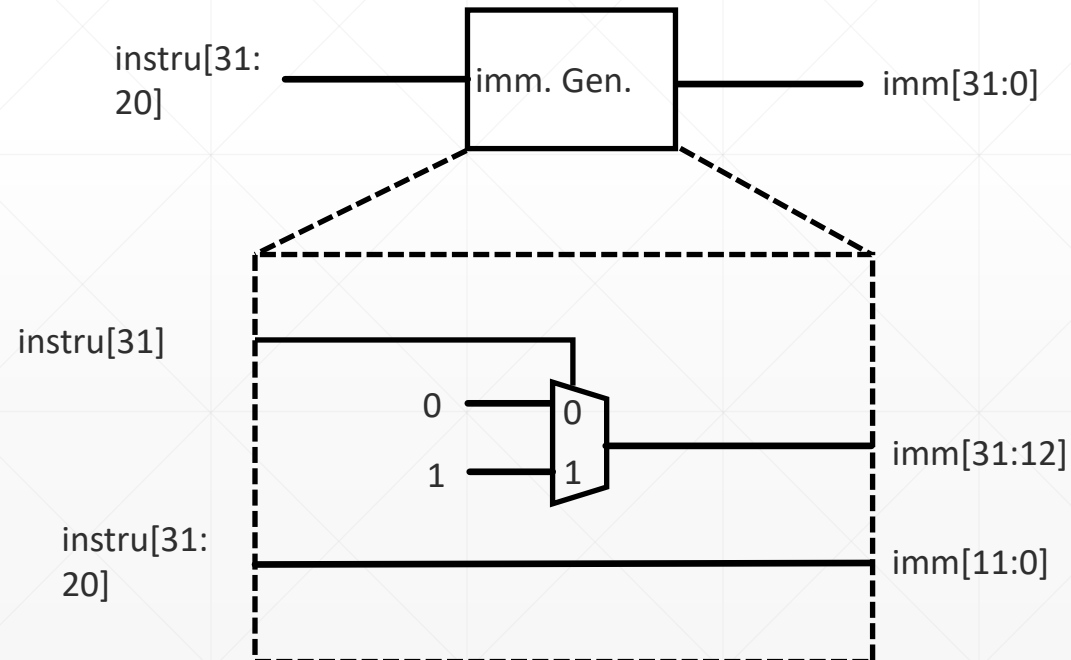
- $sw\ rs2,\ imm(rs1)$: Store word at rs2 to memory addr.
 $addr. = (\text{number in } rs1) + imm$



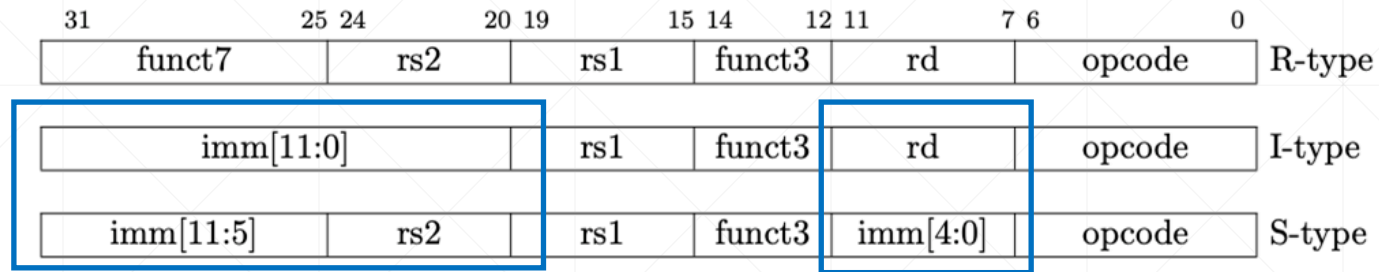
Imm. Gen.



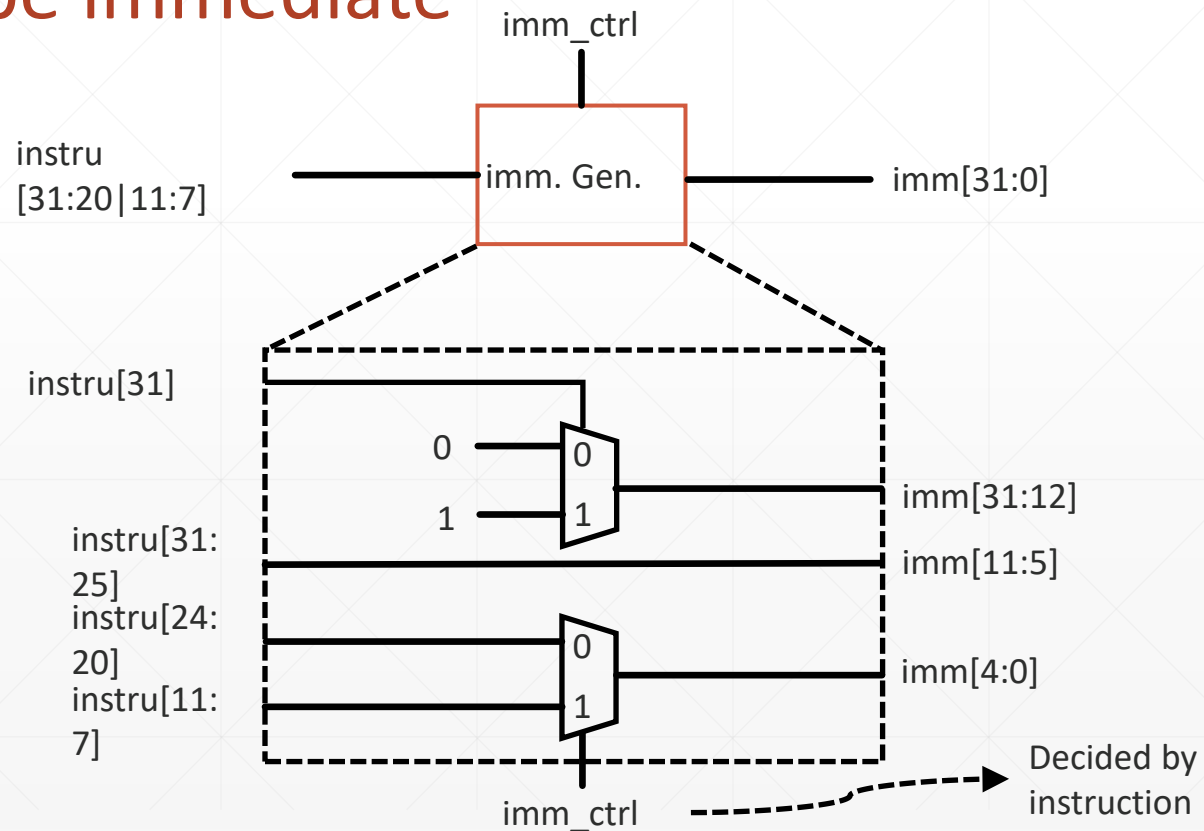
- I-type immediate



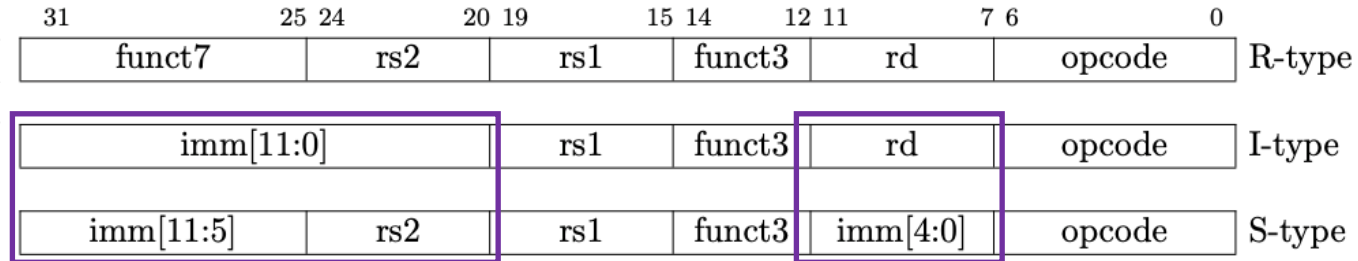
Imm. Gen.



- I-type & S-type immediate

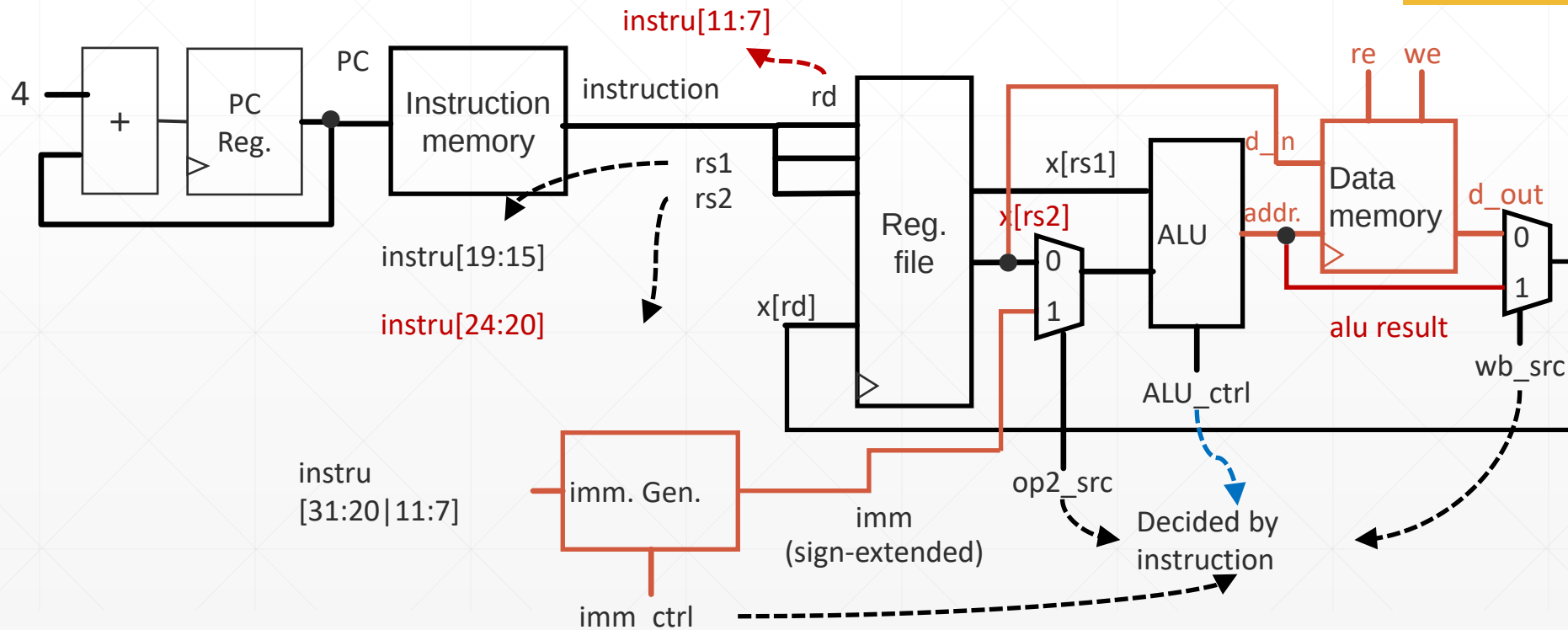


Datapath for S-type store

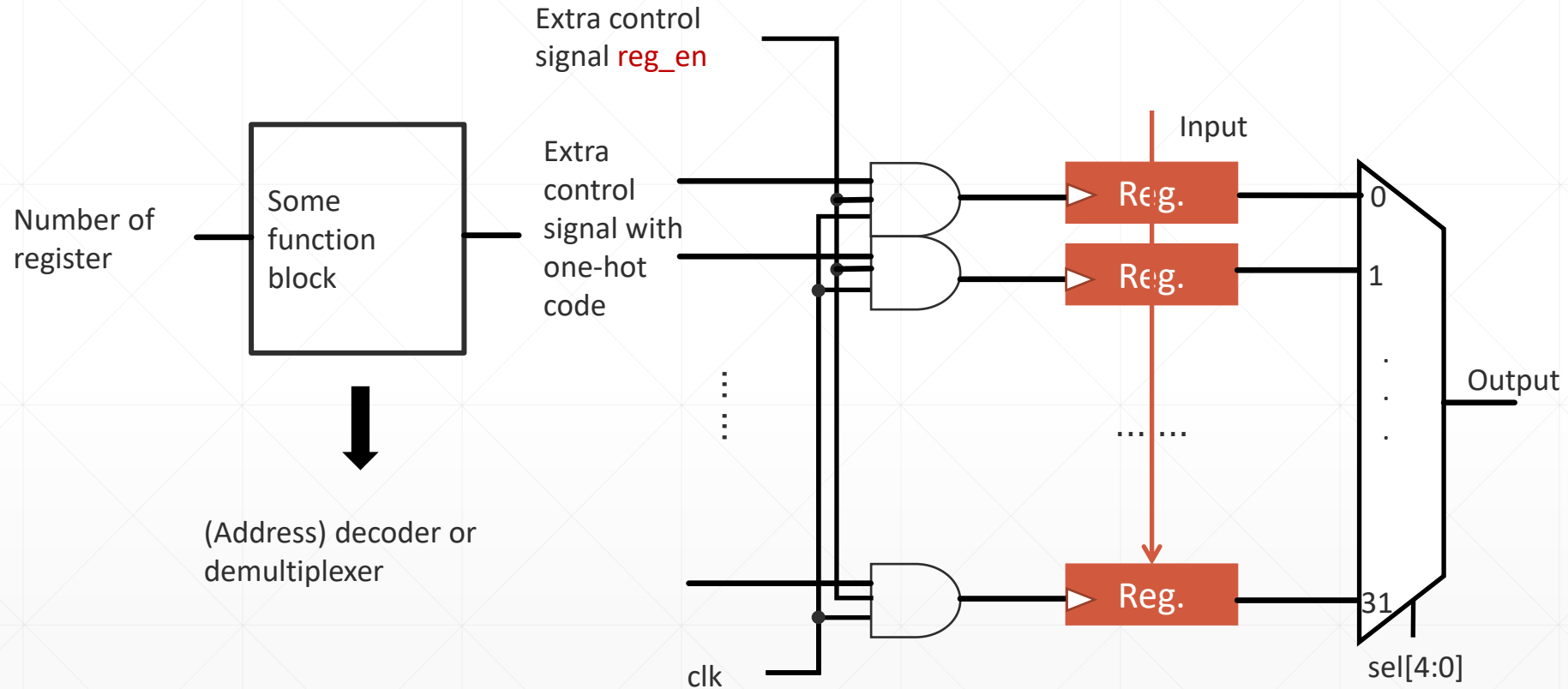


- $sw\ rs2,\ imm(rs1)$: Store word at rs2 to memory addr.
 $addr. = (\text{number in } rs1) + imm$

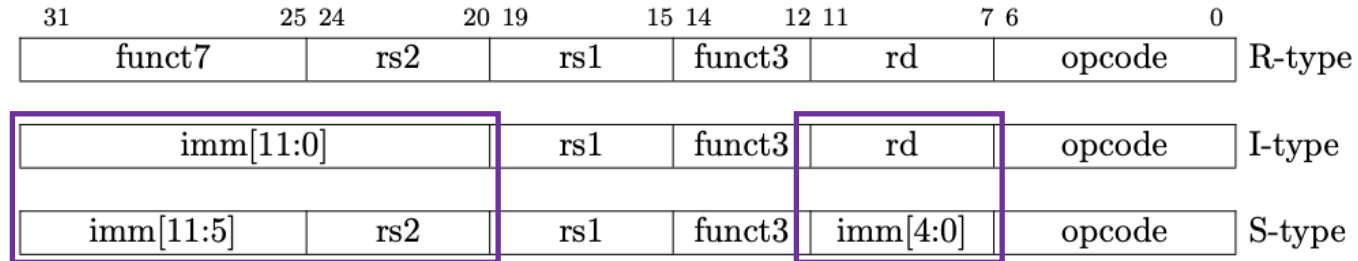
Something wrong!



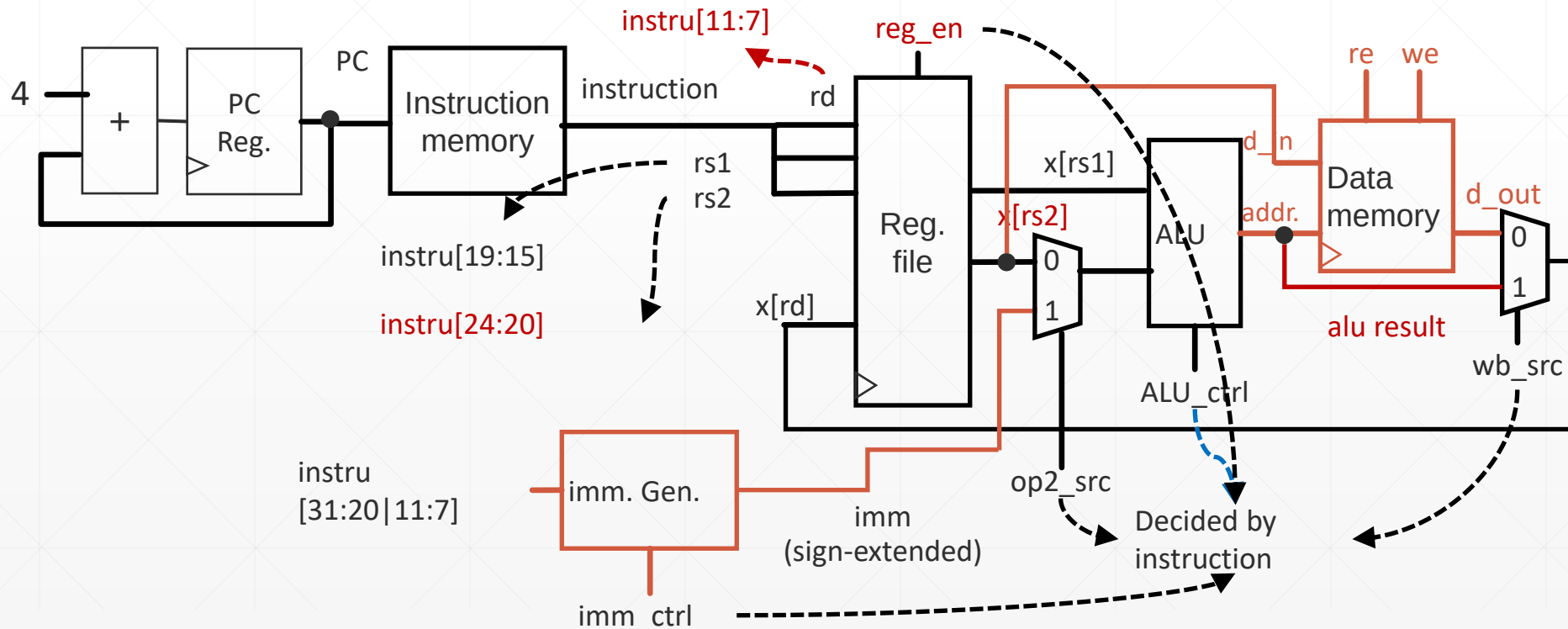
Regfile modification



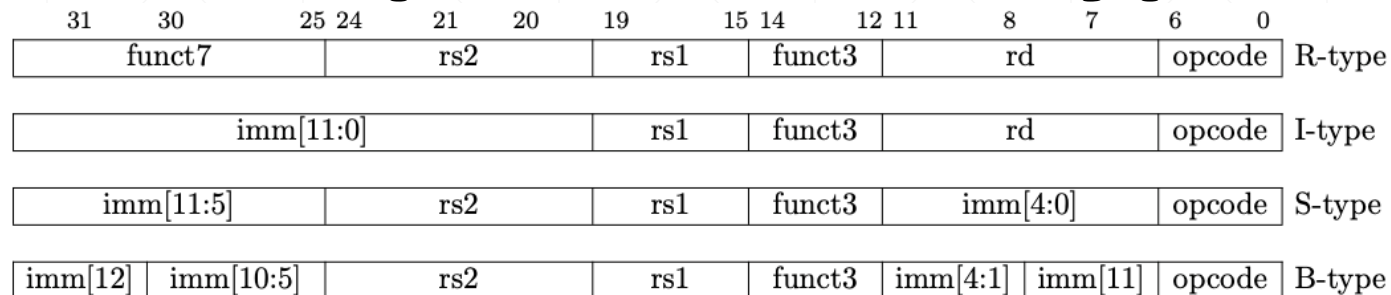
Datapath for S-type store



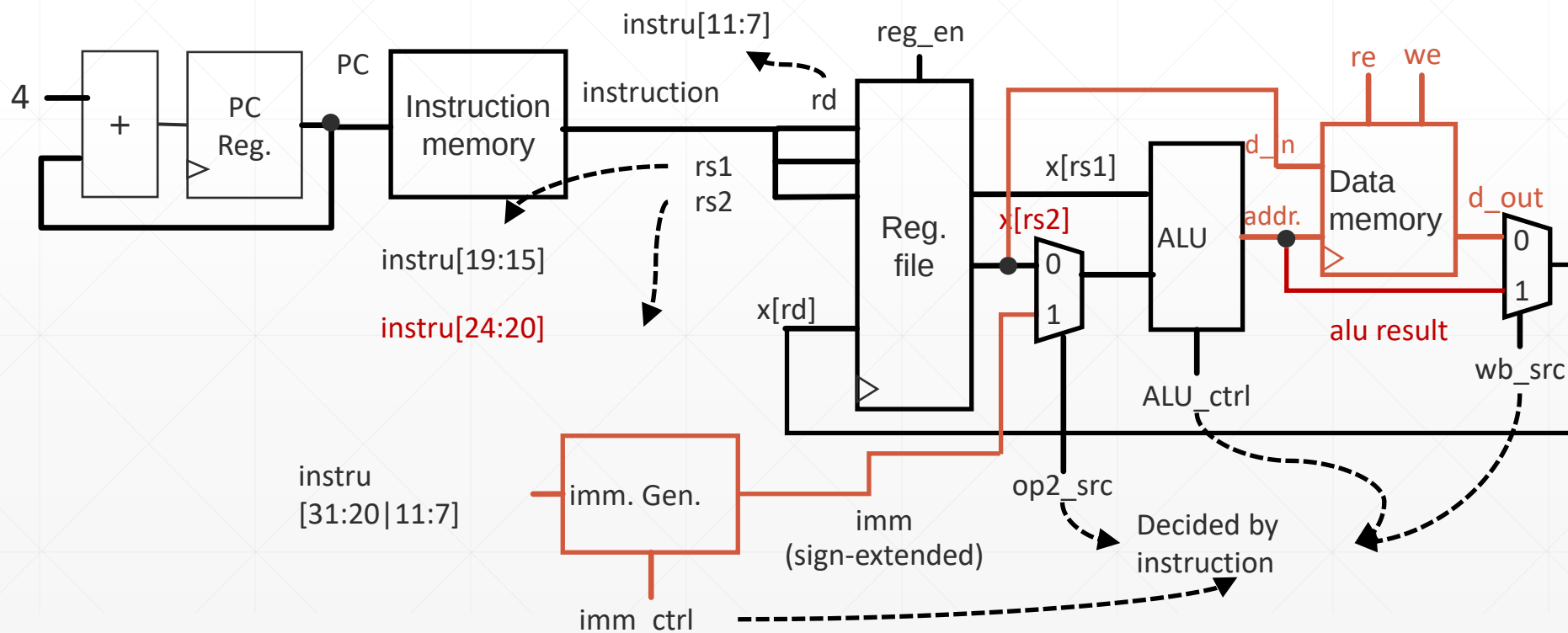
- $sw\ rs2,\ imm(rs1)$: Store word at rs2 to memory addr.
 $addr. = (\text{number in } rs1) + imm$



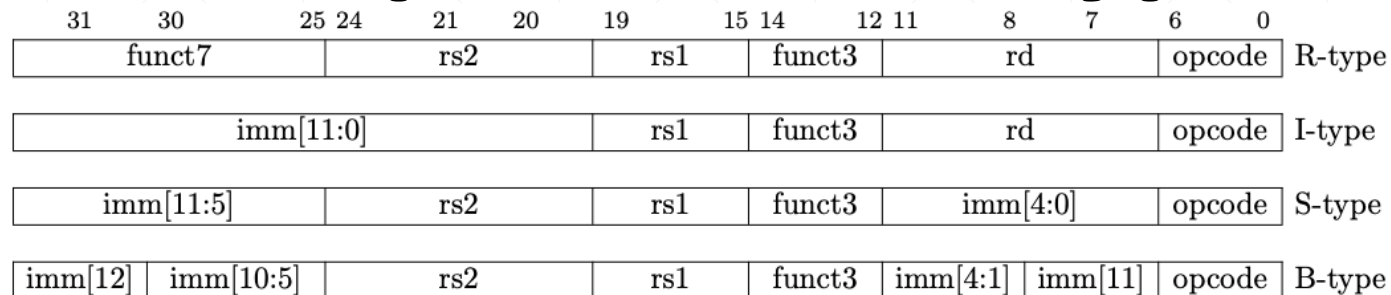
Datapath for B-type



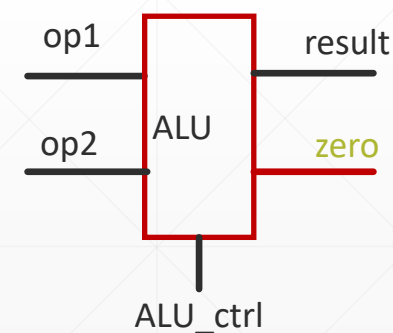
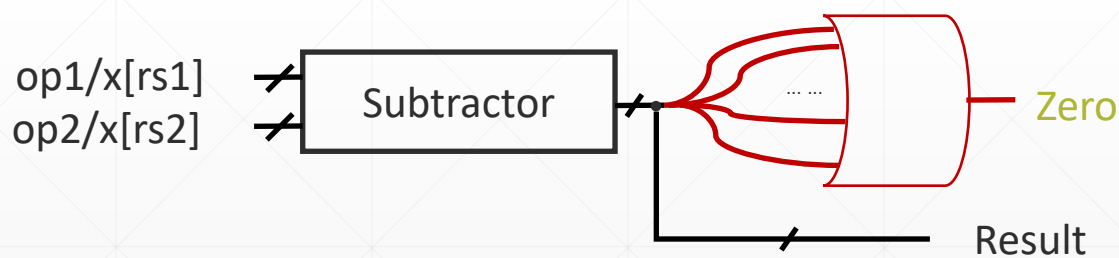
- **beq rs1,rs2,L(imm/label)**
- Go to label if $x[rs1] == x[rs2]$; otherwise, go to next statement



Datapath for B-type

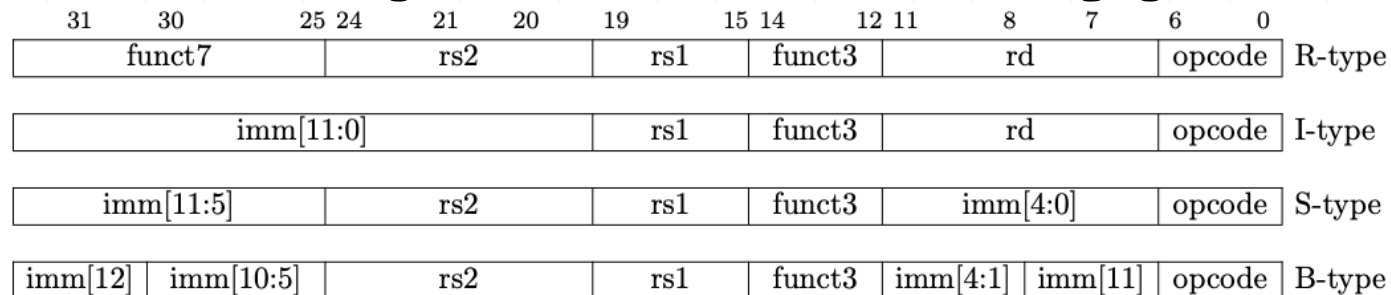


- **beq rs1,rs2,L(imm/label)**
- Go to label if $x[rs1] == x[rs2]$; otherwise, go to next statement
- Recall in ALU

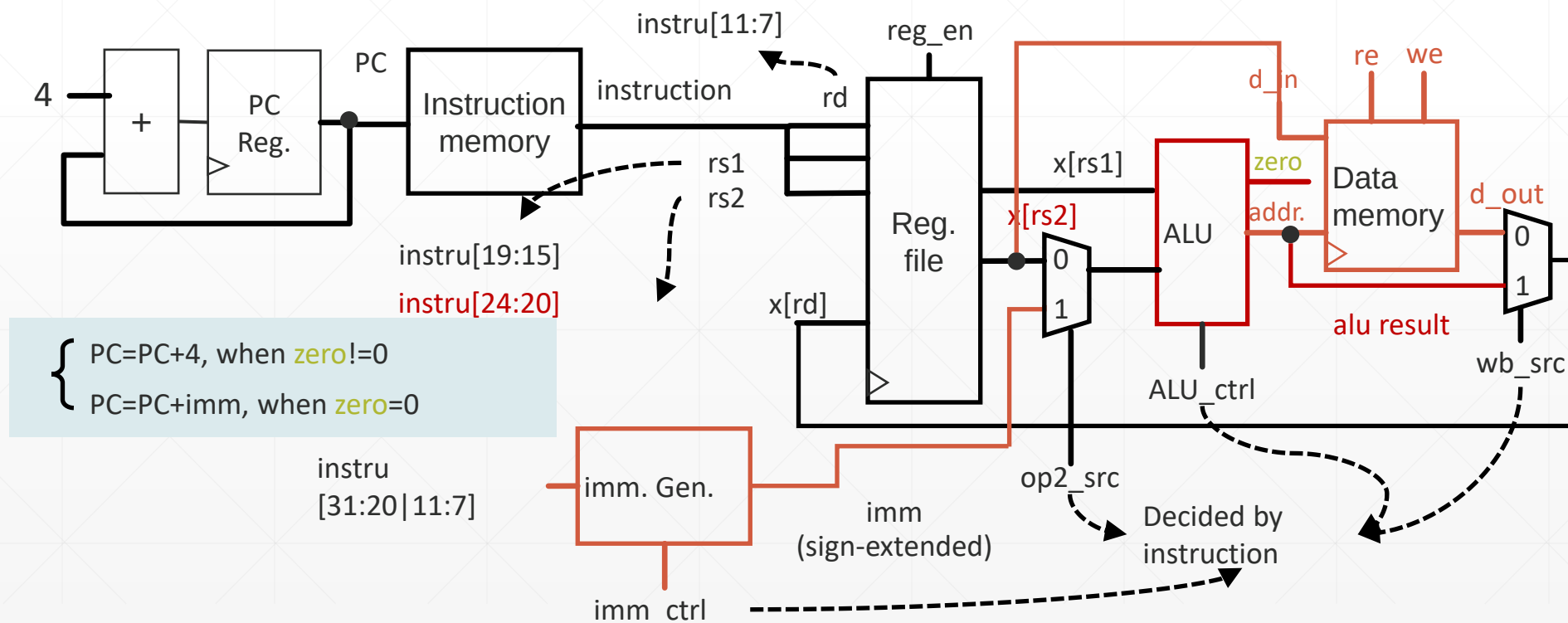


$$x[rs1] == x[rs2] \leftrightarrow x[rs1] - x[rs2] == 0$$

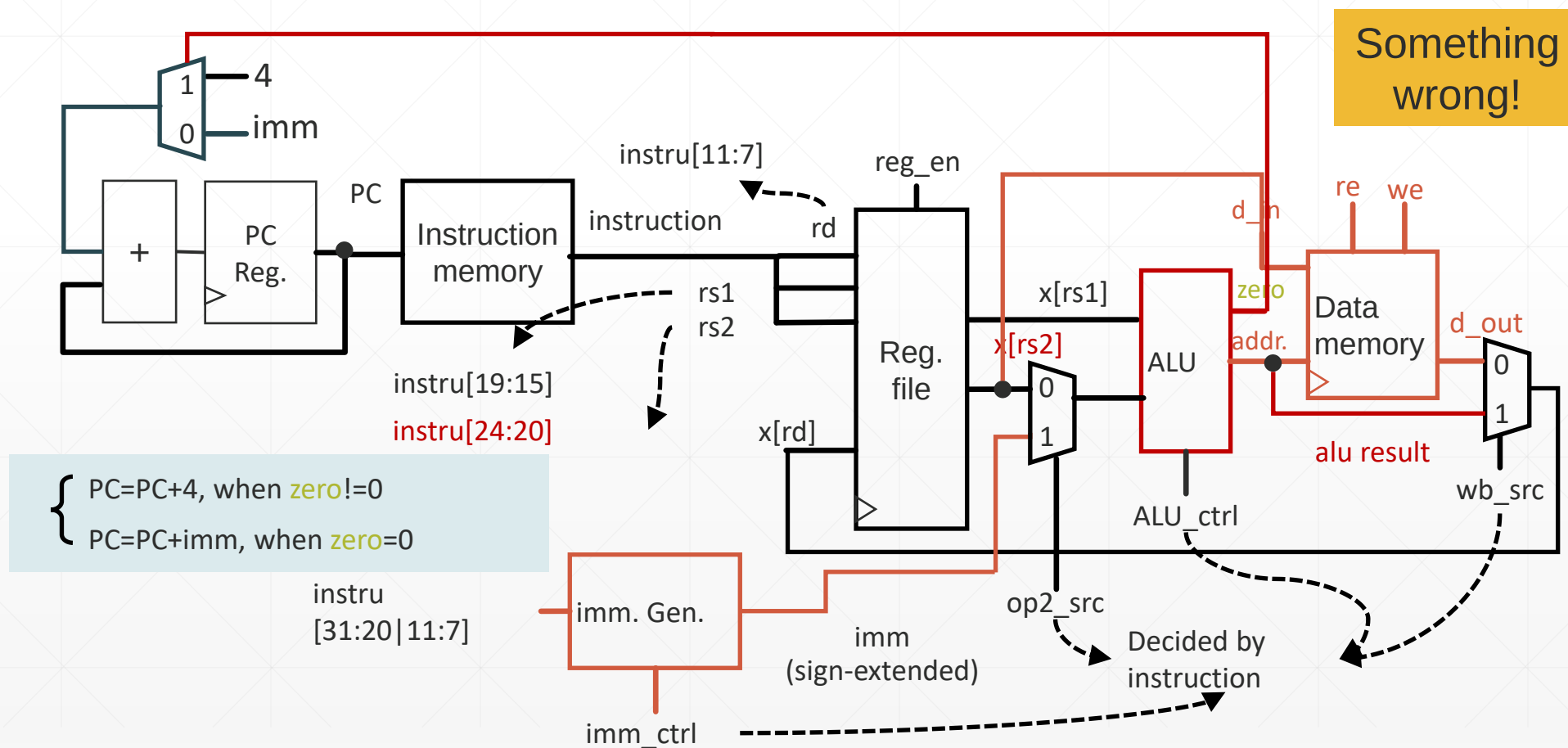
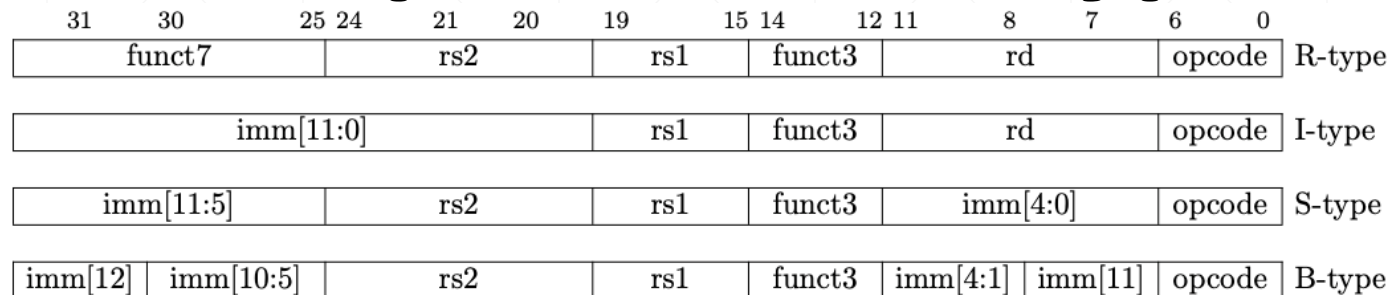
Datapath for B-type



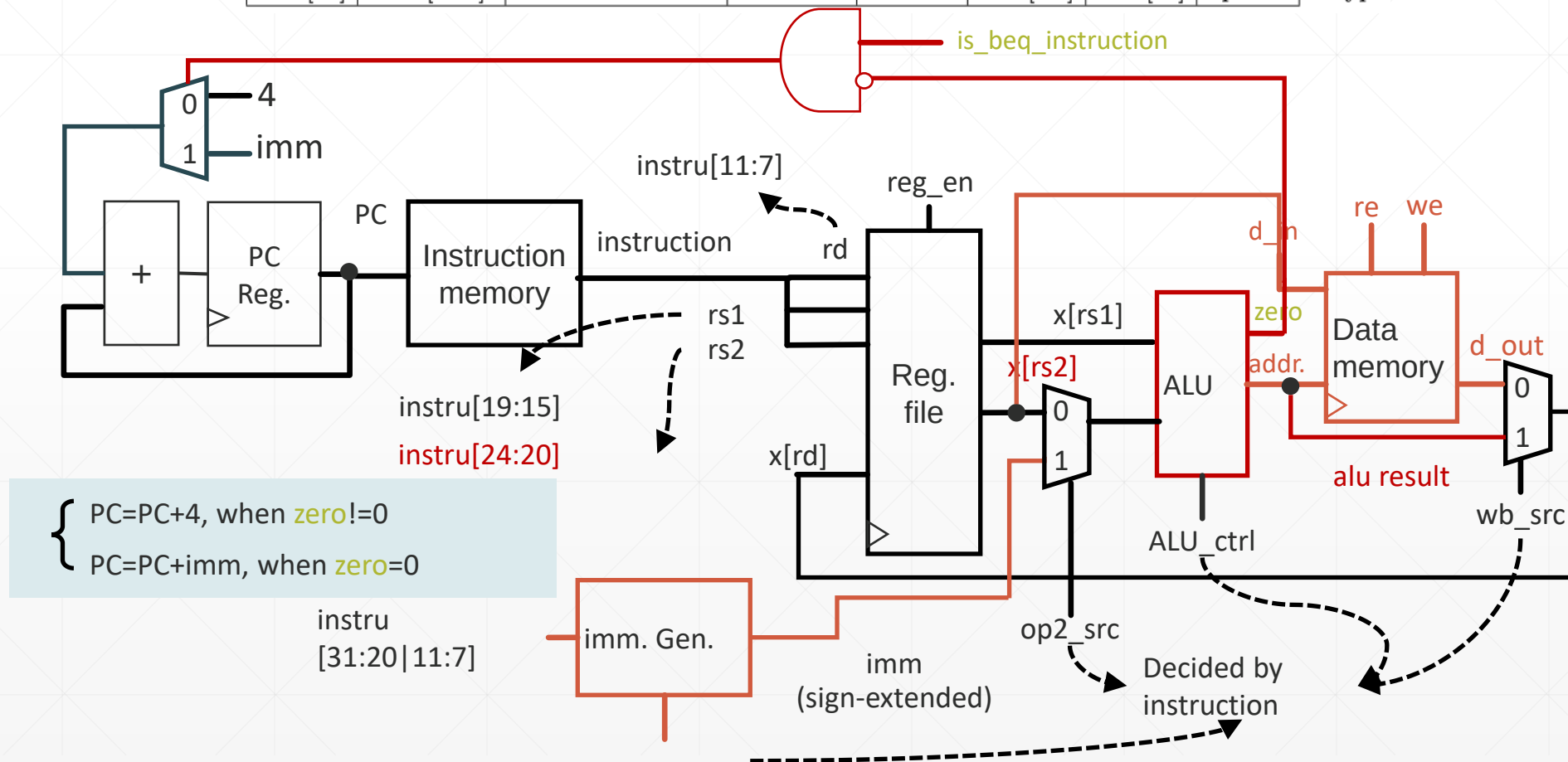
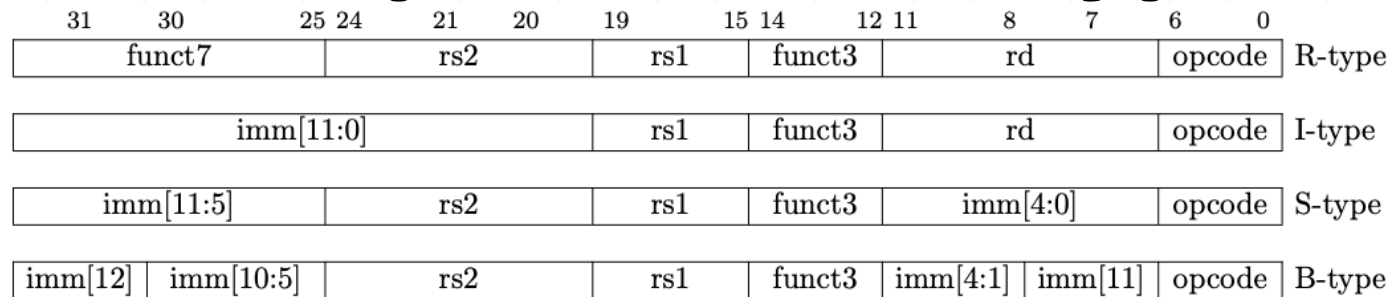
- **beq rs1,rs2,L(imm/label)**
- Go to label if $x[rs1] == x[rs2]$; otherwise, go to next statement



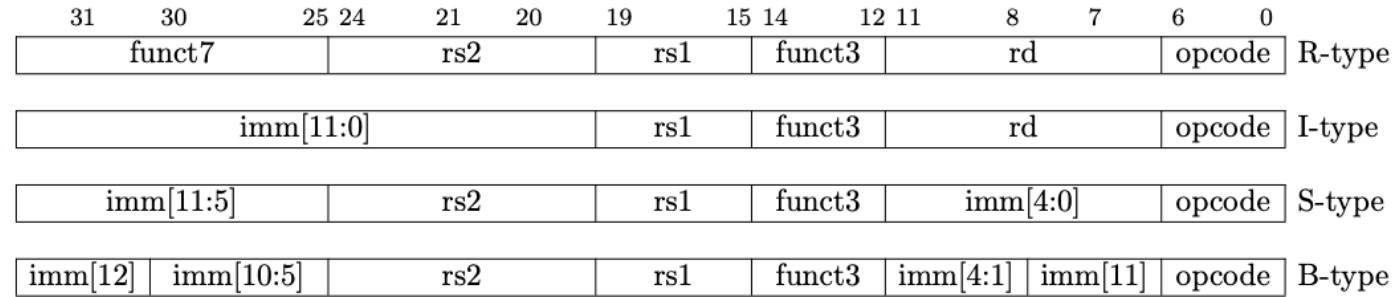
Datapath for B-type



Datapath for B-type

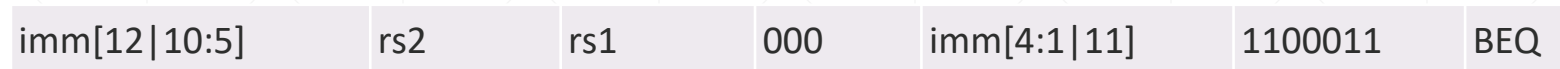


Datapath for B-type



`is_beq_instruction`

- Recall beq

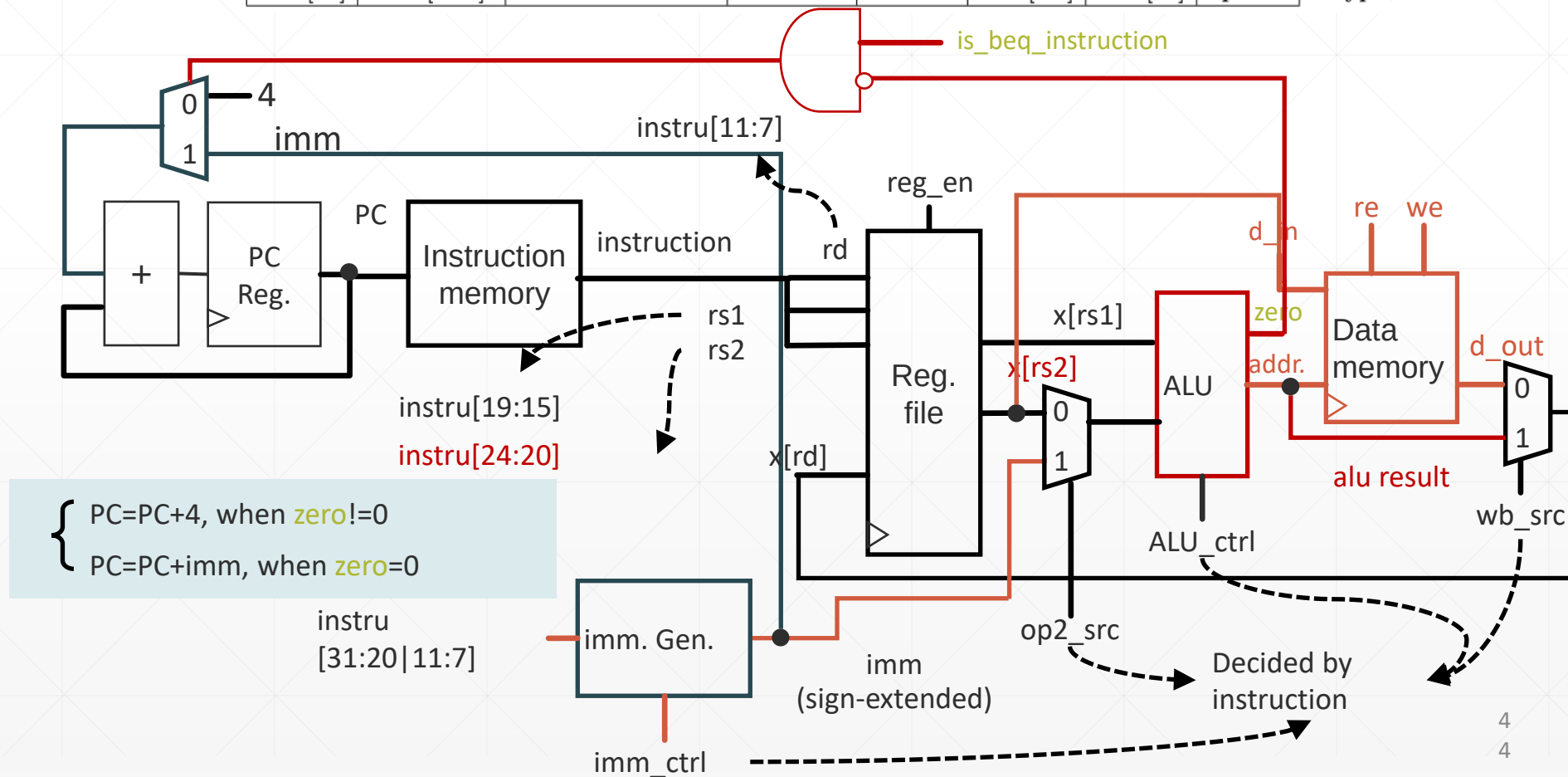
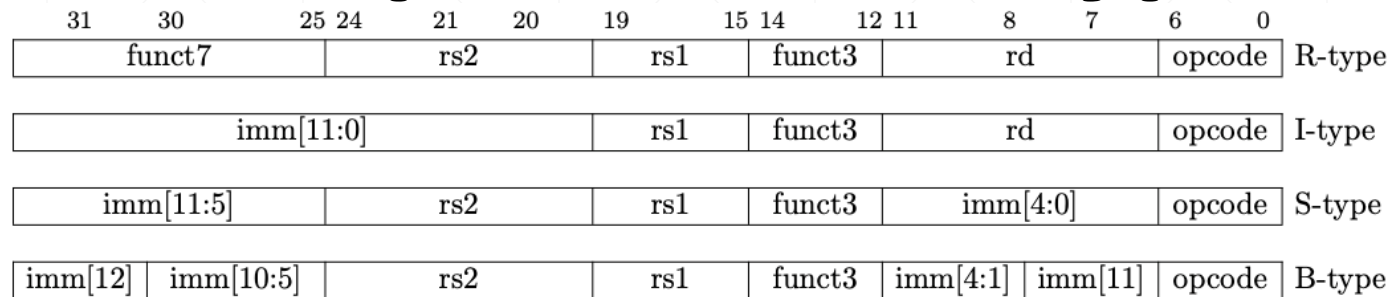


Truth table

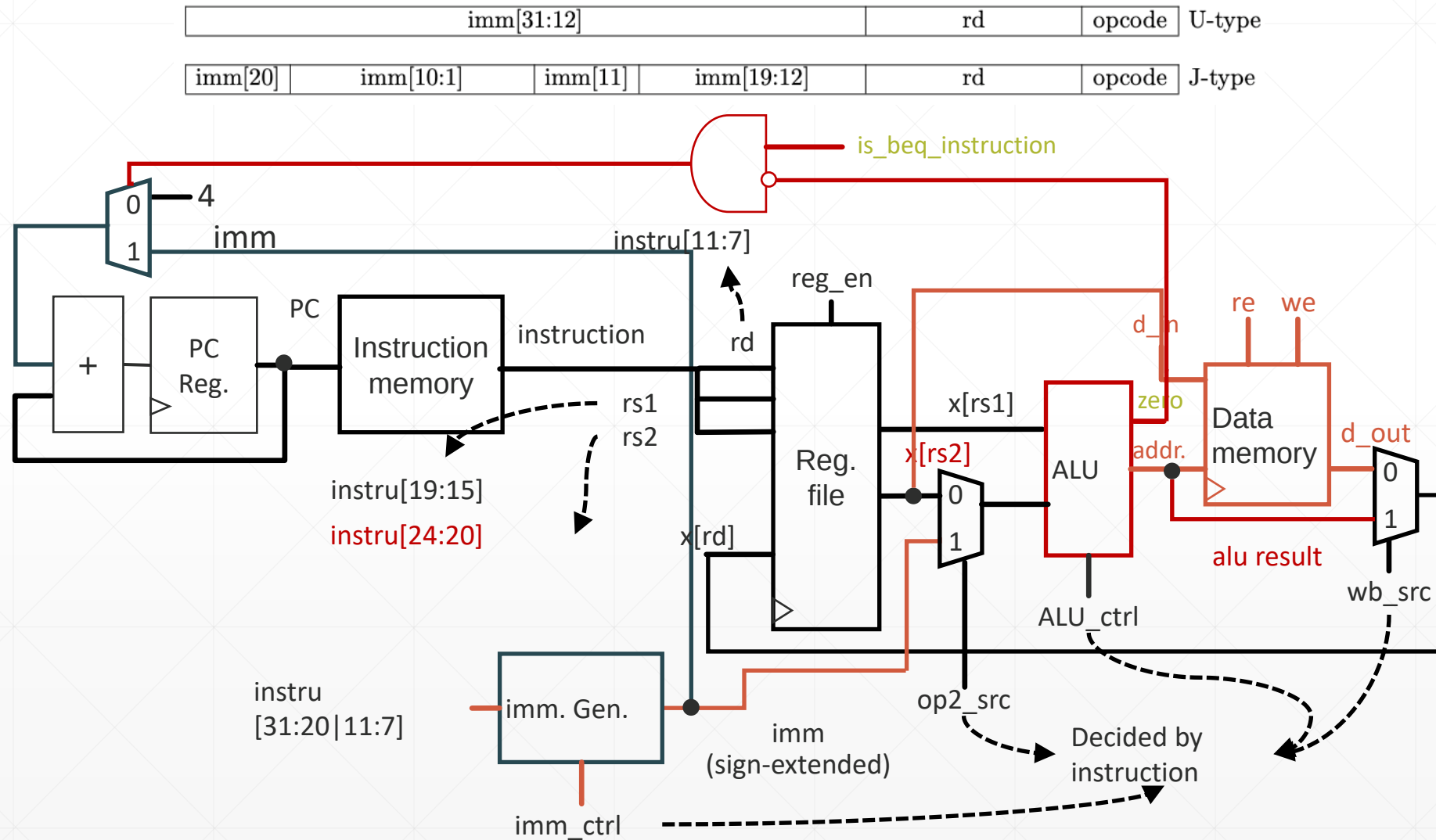
Instru[14:12 6:0]	<code>is_beq_instruction</code>
000 1100011	1
All the other cases	0

`is_beq_instruction = i[14]i[13]i[12]i[6]i[5]i[4]i[3]i[2]i[1]i[0]`

Datapath for B-type



Datapath for the other types



Summary

- Real-world datapath is much more complicated than this lecture