Q1.

R-type	I-type	lw	sw	Branch
35%	20%	15%	15%	15%

a. Data memory

Instructions that need the data memory are those that read data from memory or that write data to memory. Those instructions are `lw` and `sw`.

The fraction of instructions that need the data memory is thus

$$15\% + 15\% = 30\%.$$

b. ALU

R-type and I-type instructions use the ALU to perform the requested arithmetic operation. The `lw` and `sw` instructions use the ALU to compute the effective address that needs to be read or written. Branches use the ALU to compute the result of the requested comparison.

In total, the fraction of instructions that need the ALU is thus 100%.

c. Sign extension unit

I-type instructions need the sign extension unit to extend the given 12-bit immediate field to a 32-bit value that can then be used as second operand by the ALU. Likewise, `lw` and `sw` instructions need the sign extension unit to extend the given 12-bit byte offset to a 32-bit value that can be added (by the ALU) to the 32-bit base address in `rs1`. Lastly, branches use the sign extension unit to extend the given 12-bit offset to a 32-bit value that can be added (by an adder outside of the ALU) to the 32-bit `pc`.

The fraction of instructions that need the sign extension unit is thus

$$20\% + 15\% + 15\% + 15\% = 65\%.$$

d. MemtoReg stuck to 0

In a functional datapath, MemtoReg is only set to 1 when executing a `lw` instruction. This is indeed the only instruction for which data needs to move from the data memory to the register file. If the signal is stuck to 0, only `lw` instructions will thus fail.

The fraction of instructions that will fail if MemtoReg is stuck to 0 is thus 15%.

e. ALUSrc stuck to 0

In a functional datapath, R-type instructions need ALUSrc to be 0 so that the value of `rs2` can be utilized by the ALU. I-type instructions need ALUSrc to be set to 1 so that the immediate field is forwarded to the ALU. Likewise, `lw` and `sw` instructions need ALUSrc to be 1 so that the offset can be added to `rs1` by the ALU. Lastly, branches need ALUSrc to be 0 because the two operands of the comparison, which is performed by the ALU, are both obtained from registers.

The fraction of instructions that will fail if ALUSrc is stuck to 0 is thus

$$20\% + 15\% + 15\% = 50\%.$$

Q2.

In general, the time at which a signal driven by a subunit becomes ready is equal to the sum of the propagation delay of this subunit and of the latest time of arrival of the inputs of the subunit.

For multiplexers, the time at which the output becomes ready is equal to the sum of the propagation delay of the multiplexer and of the arrival time of the selected input signal.

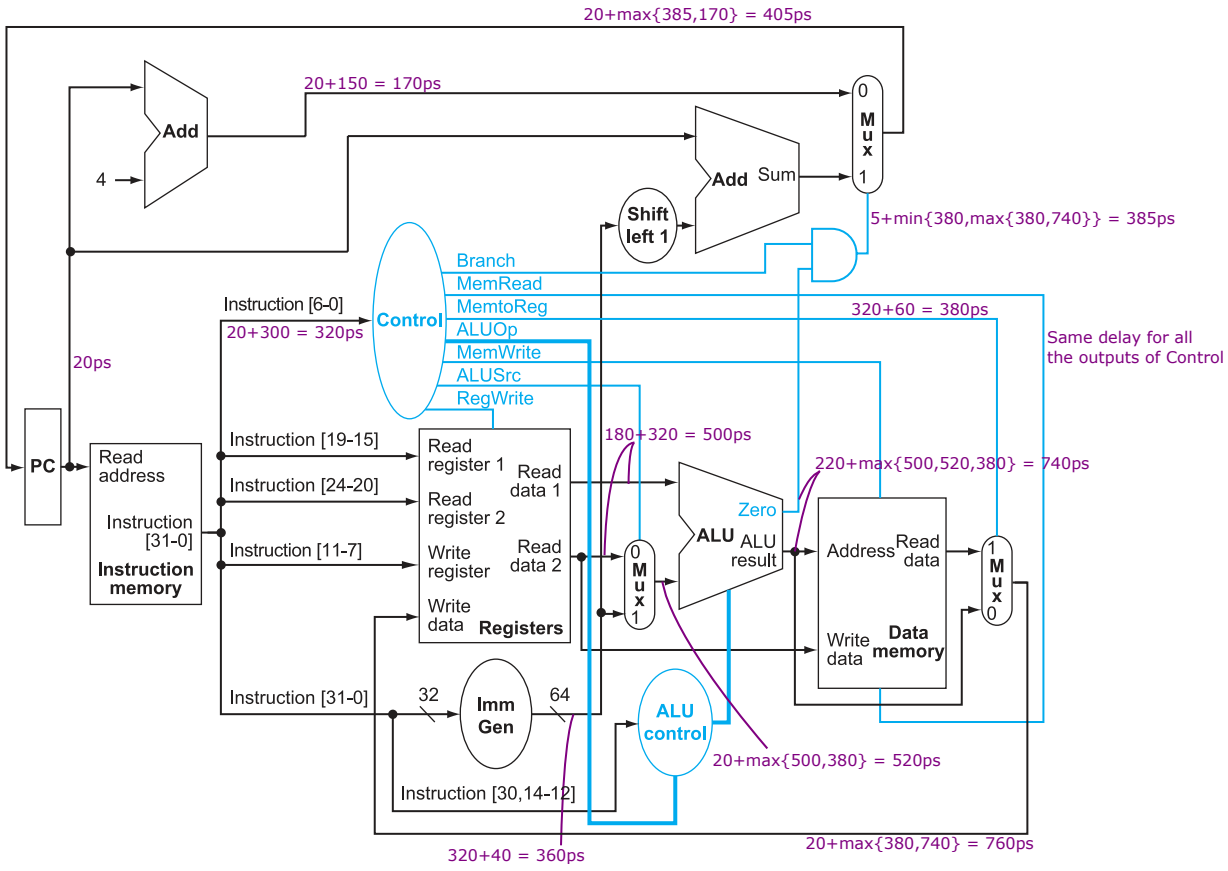
For AND gates, the time at which the output becomes ready is given by

$$\text{propagation delay of AND gate} + \min\{\text{arrival time of known 0 inputs}, \max\{\text{arrival time of any inputs}\}\}$$

Using these rules, the arrival times of all the signals of the datapath can be derived.

a. sub instruction

The arrival times are shown here for the sub instruction:

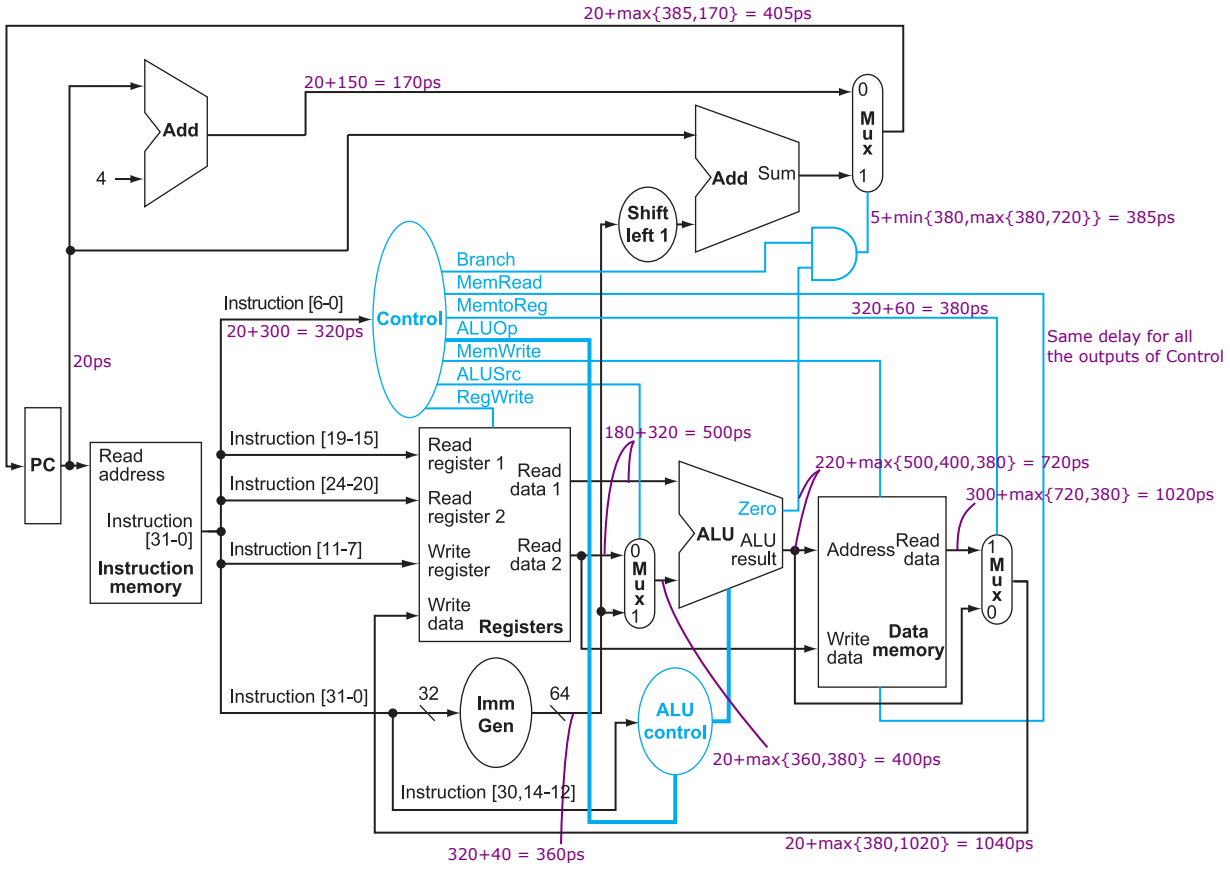


The *RegWrite* and *Write data* signals arrive respectively to the register file after 380ps and 760ps. If we add the time needed to write into the register file, we find that the latency of the sub instruction is

$$10 + \max\{380, 760\} = 770\text{ps}.$$

b. `lw` instruction

The arrival times are shown here for the `lw` instruction:

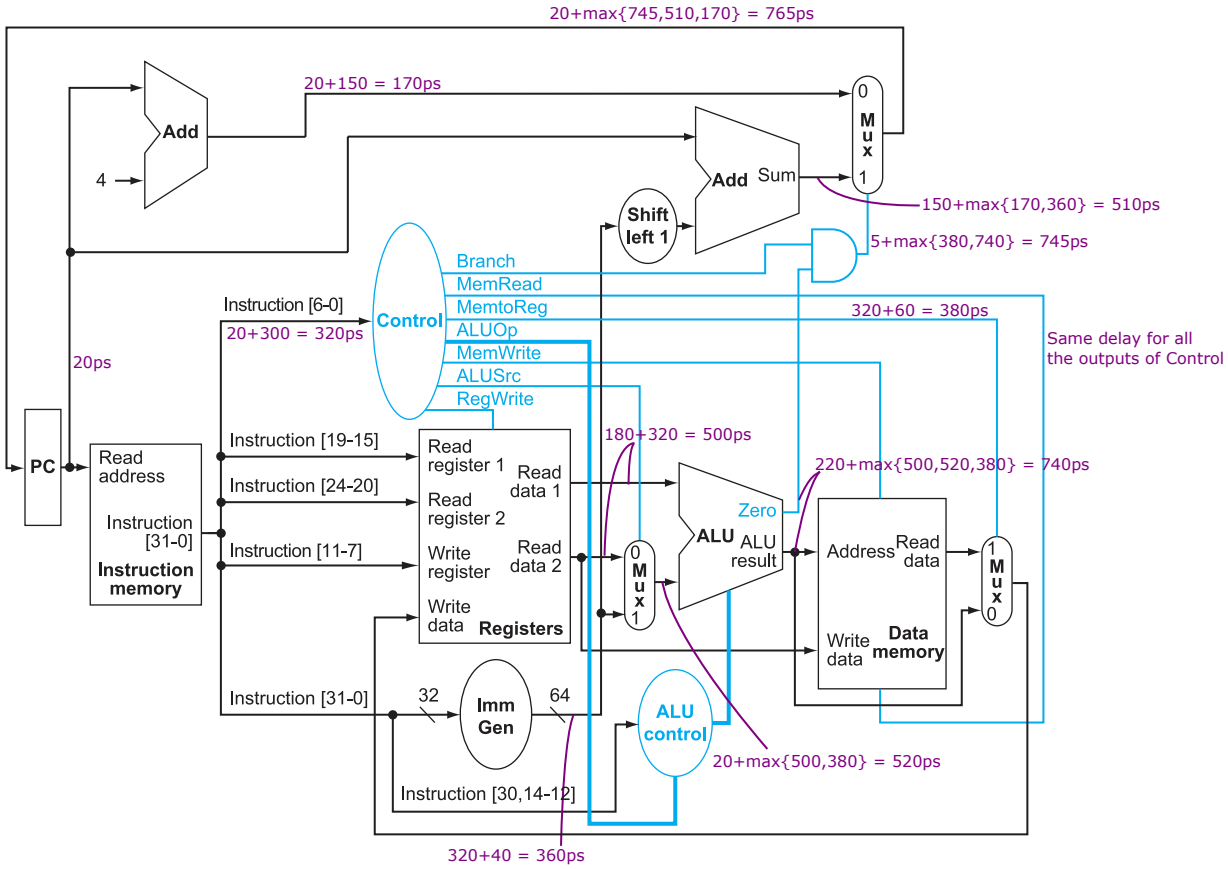


The *RegWrite* and *Write data* signals arrive respectively to the register file after 380ps and 1040ps. If we add the time needed to write into the register file, we find that the total latency of the `lw` instruction is

$$10 + \max\{380, 1040\} = 1050\text{ps}.$$

c. beq instruction

The arrival times are shown here for the beq instruction:



The new value of pc arrives after 765ps. If we consider the time needed to write this value into pc, we find that the total latency of the beq instruction is

$$20 + 765 = 785\text{ps}.$$

Q3.

If we add `swap rs1, rs2` to the ISA, it would be the first and only instruction that writes two registers.

To support this new operation, we could either execute the instruction in two cycles, or add an extra write port to the register file. In either case, the exact changes that need to be made depend on how the new instruction is encoded.

Two valid single-cycle solutions are presented here. In the two solutions, an extra write port is added to the register file with a new RegWrite2 control signal. The control signal RegWrite is renamed to RegWrite1.

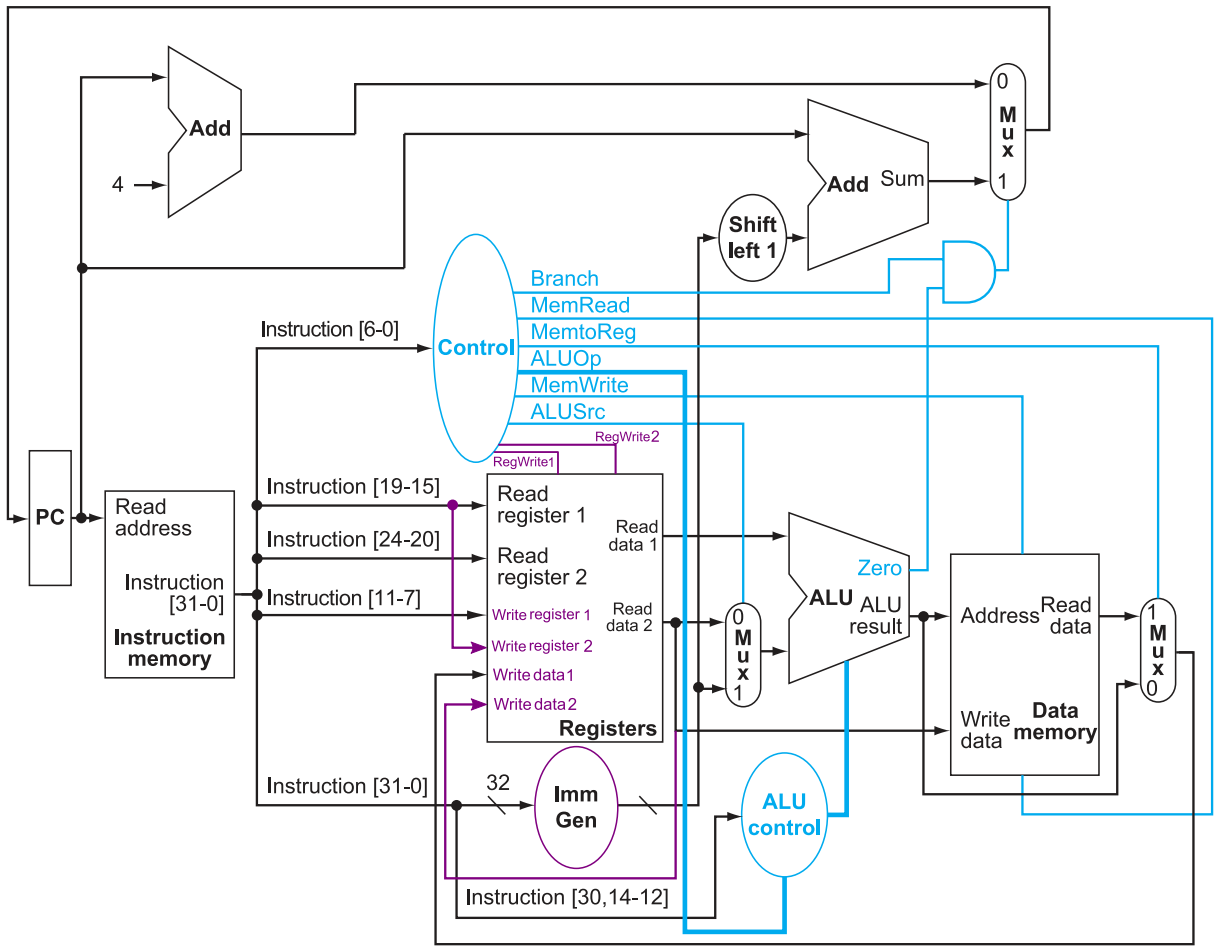
Note that there exist many other ways to add `swap rs1, rs2` to the processor than the two presented solutions.

Option 1: `swap` encoded as an R-type instruction

Assuming that `swap rs1, rs2` is encoded as

31 - 25	24 - 20	19 - 15	14 - 12	11 - 7	6 - 0
0000000	rs2	rs1	000	rs2	new swap opcode

the processor needs to be modified as follows:



The control and immediate generation are modified so that

- When the opcode is equal to the new swap opcode, the generated control signals are identical to the control signals of the `add` instruction, except for ALUSrc which is set to 1.

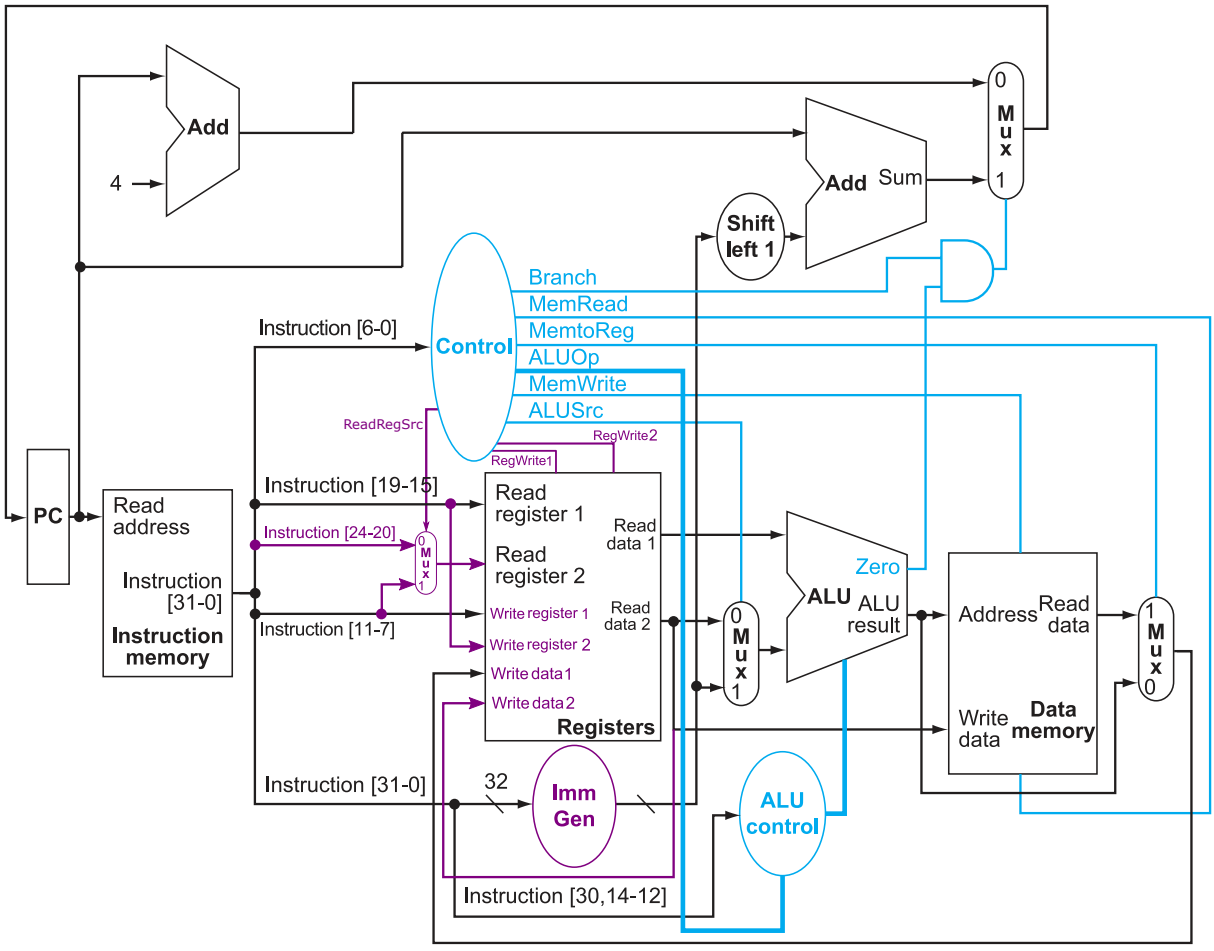
- RegWrite2 is set to 1 if the opcode is equal to the new swap opcode, or set to 0 otherwise.
- When the opcode is equal to the new swap opcode, the generated 32-bit immediate value is 0.

Option 2: `swap` encoded as an I-type instruction

Assuming that `swap rs1, rs2` is encoded as

31 - 20	19 - 15	14 - 12	11 - 7	6 - 0
000000000000	rs1	000	rs2	new swap opcode

the processor needs to be modified as follows:



The control and immediate generation are modified so that

- When the opcode is equal to the new swap opcode, the generated control signals are identical to the control signals of the `addi` instruction.
- A new control signal ReadRegSrc is added. Its value is set to 1 when the opcode is equal to the new swap opcode, and 0 otherwise.
- RegWrite2 is set to 1 if the opcode is equal to the new swap opcode, or set to 0 otherwise.
- When the opcode is equal to the new swap opcode, the immediate generation unit functions as it would for any I-type instruction.

Note that Option 2 might be less favorable than Option 1 because the former adds a multiplexer in the critical path of the datapath.