

Hardware Building Blocks: Some Use Cases in Components Design

Artem Burmyakov

September 27, 2024



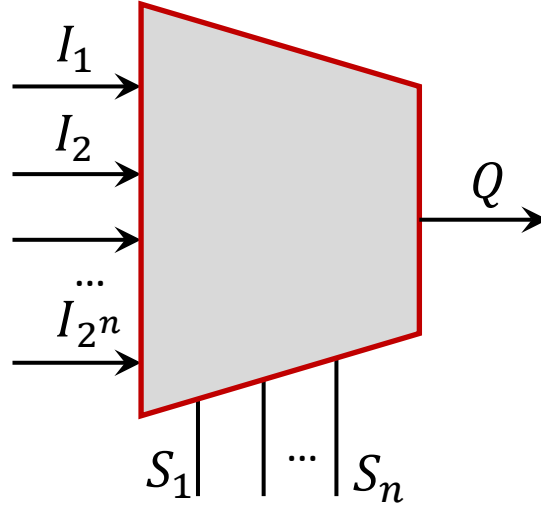
INNOPOLIS
UNIVERSITY

Hardware Building Blocks: **What is It?**

Hardware Building Blocks

Multiplexer

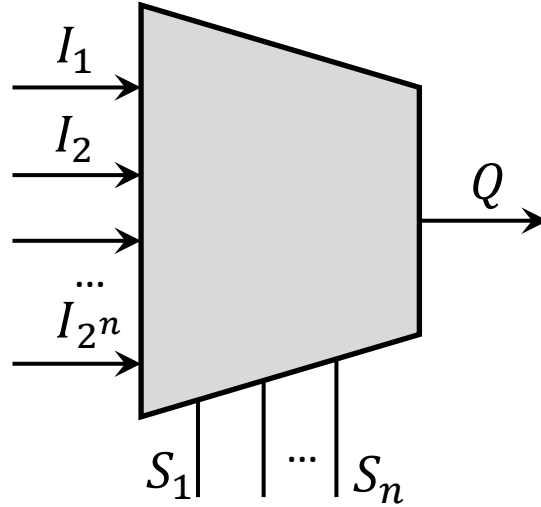
Output Q is set to one of inputs $I_1, \dots, I_{2^n}$, based on selectors $S_1, \dots, S_n$



Hardware Building Blocks

Multiplexer

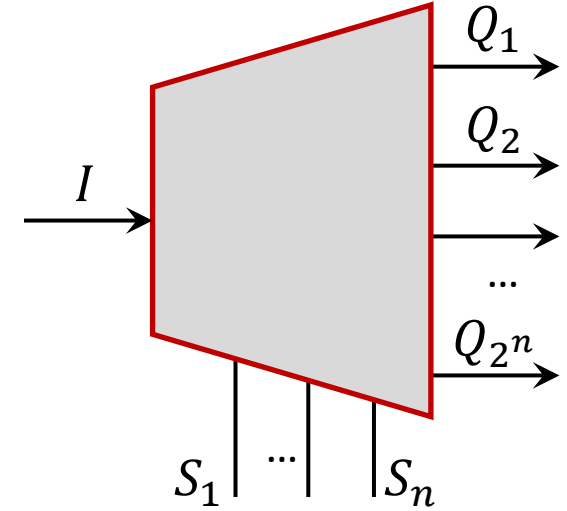
Output Q is set to one of inputs $I_1, \dots, I_{2^n}$, based on selectors $S_1, \dots, S_n$



Demultiplexer

“Flipped” Multiplexer:

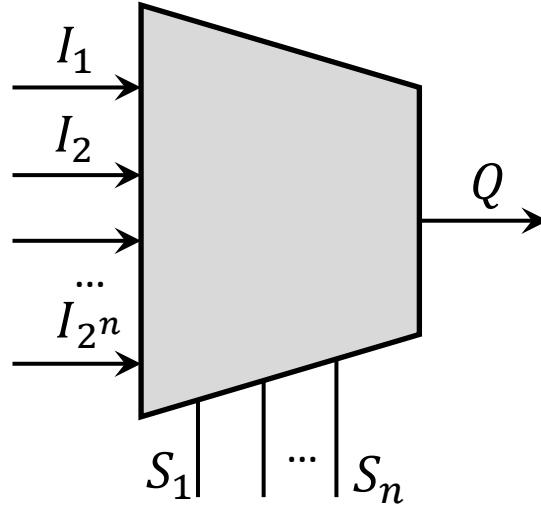
Some output Q_x is set to input I , while others Q_i with $i \neq x$ are set to “0”s



Hardware Building Blocks

Multiplexer

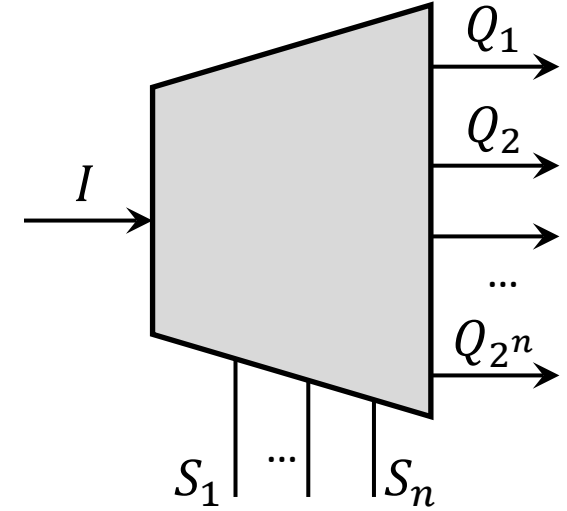
Output Q is set to one of inputs $I_1, \dots, I_{2^n}$, based on selectors $S_1, \dots, S_n$



Demultiplexer

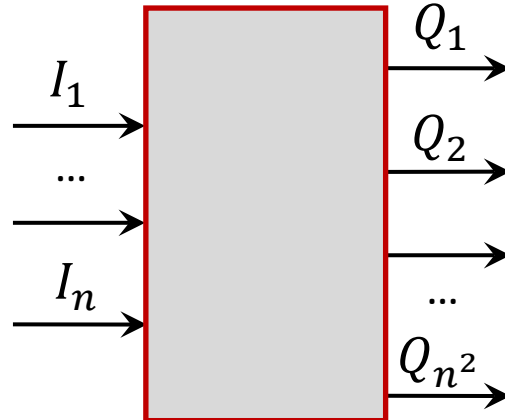
“Flipped” Multiplexer:

Some output Q_x is set to input I , while others Q_i with $i \neq x$ are set to “0”s



Decoder

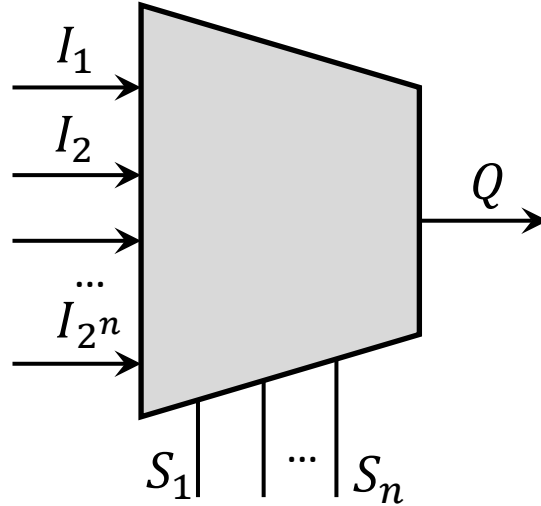
Based on inputs $I_1, \dots, I_{2^n}$, some output Q_x is set to “1”, while others Q_i with $i \neq x$ are set to “0”s



Hardware Building Blocks

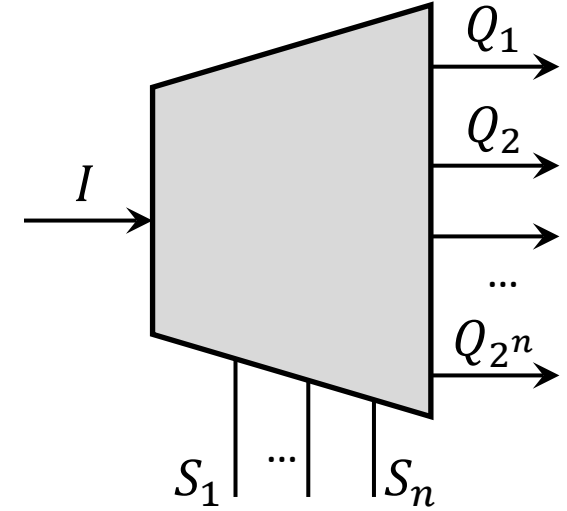
Multiplexer

Output Q is set to one of inputs $I_1, \dots, I_{2^n}$, based on selectors $S_1, \dots, S_n$



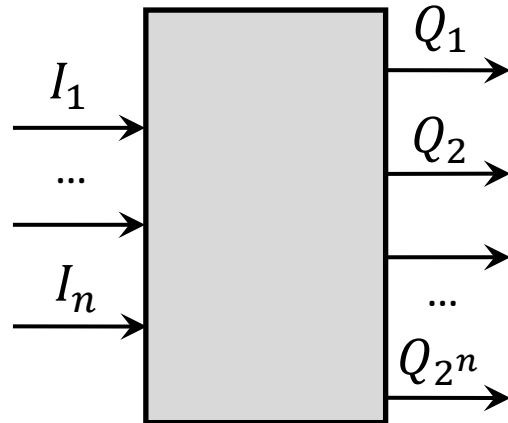
Demultiplexer

“Flipped” Multiplexer:
Some output Q_x is set to input I , while others Q_i with $i \neq x$ are set to “0”s



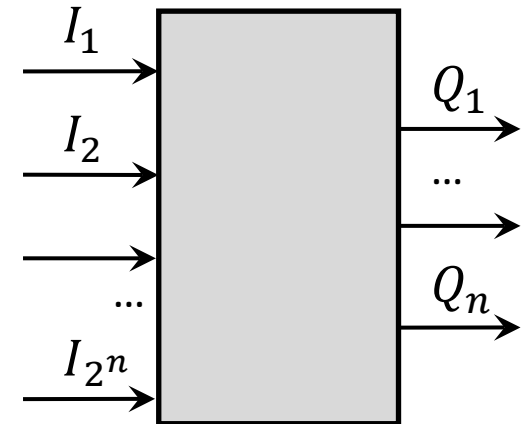
Decoder

Based on inputs $I_1, \dots, I_{2^n}$, some output Q_x is set to “1”, while others Q_i with $i \neq x$ are set to “0”s



Encoder

“Flipped” Decoder:
Only one input in $I_1, \dots, I_{2^n}$ is allowed to be “1”;
For given inputs, there is a pre-defined set of outputs $Q_1, \dots, Q_n$

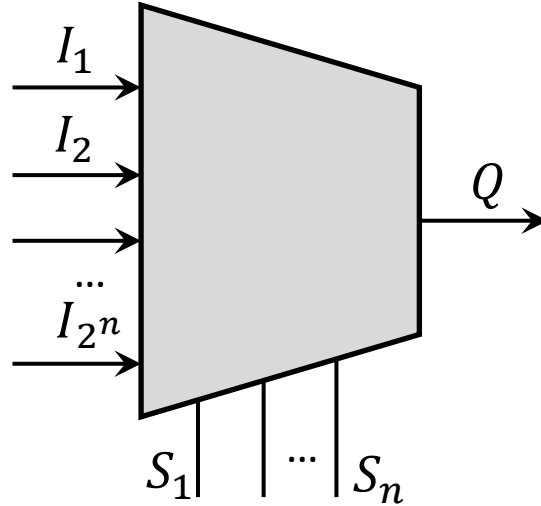


Hardware Building Blocks

Widely used in the implementation of real hardware,
e.g. ALU, caches, system memory

Multiplexer

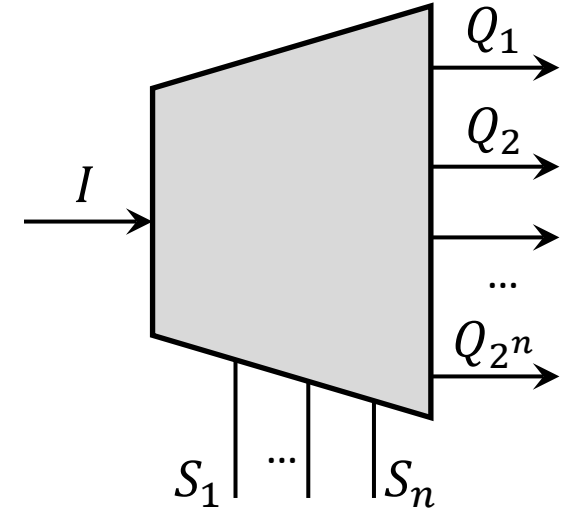
Output Q is set to one of inputs $I_1, \dots, I_{2^n}$, based on selectors $S_1, \dots, S_n$



Demultiplexer

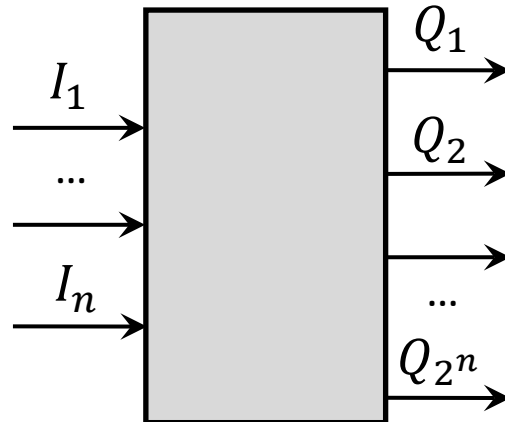
“Flipped” Multiplexer:

Some output Q_x is set to input I , while others Q_i with $i \neq x$ are set to “0”s



Decoder

Based on inputs $I_1, \dots, I_{2^n}$, some output Q_x is set to “1”, while others Q_i with $i \neq x$ are set to “0”s

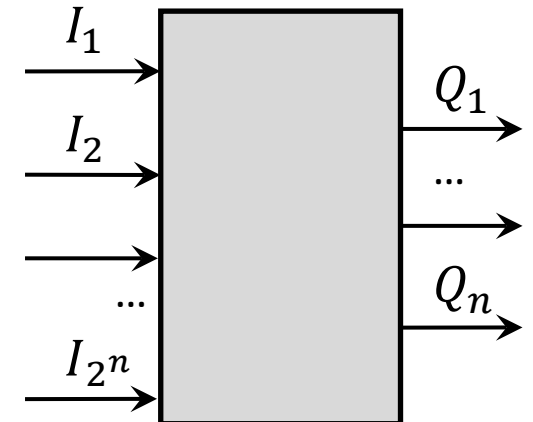


Encoder

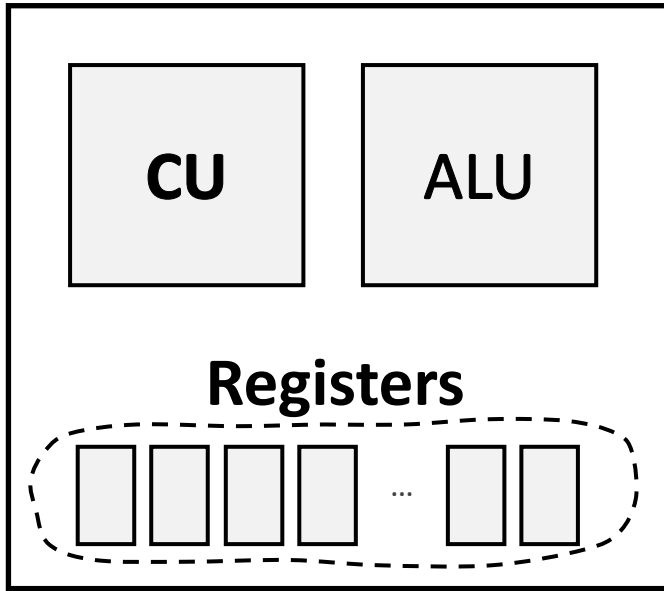
“Flipped” Decoder:

Only one input in $I_1, \dots, I_{2^n}$ is allowed to be “1”;

For given inputs, there is a pre-defined set of outputs $Q_1, \dots, Q_n$

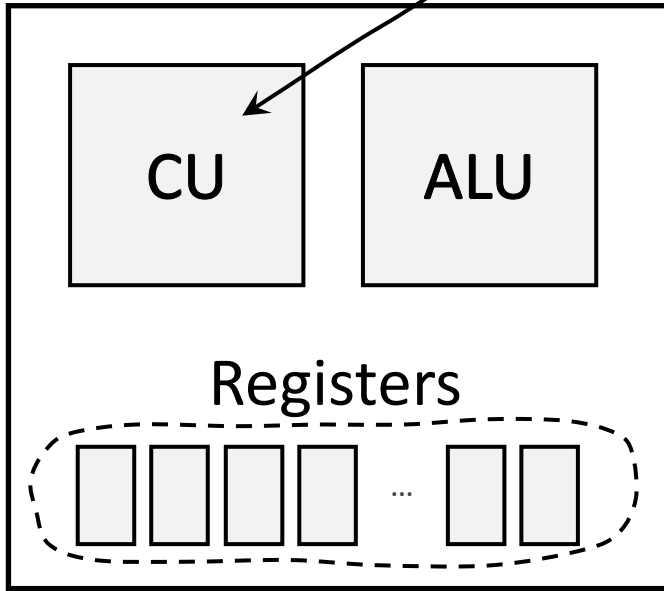


Recap: Key CPU Components

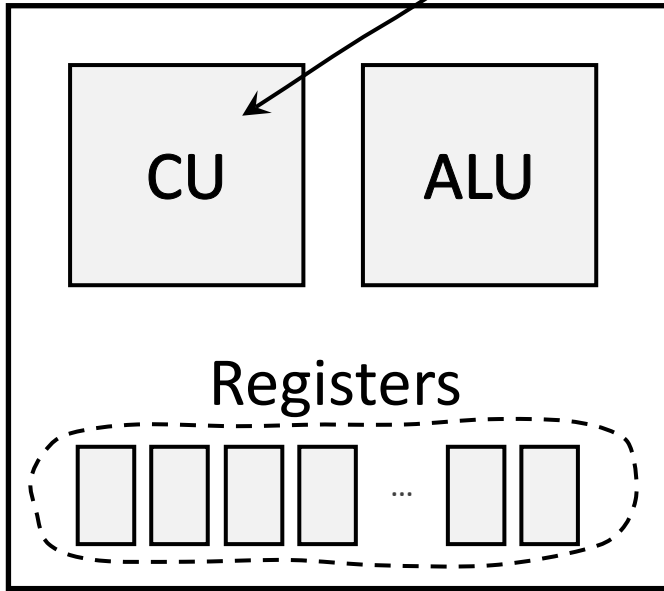


Recap: Key CPU Components

Control Unit (CU) functions?



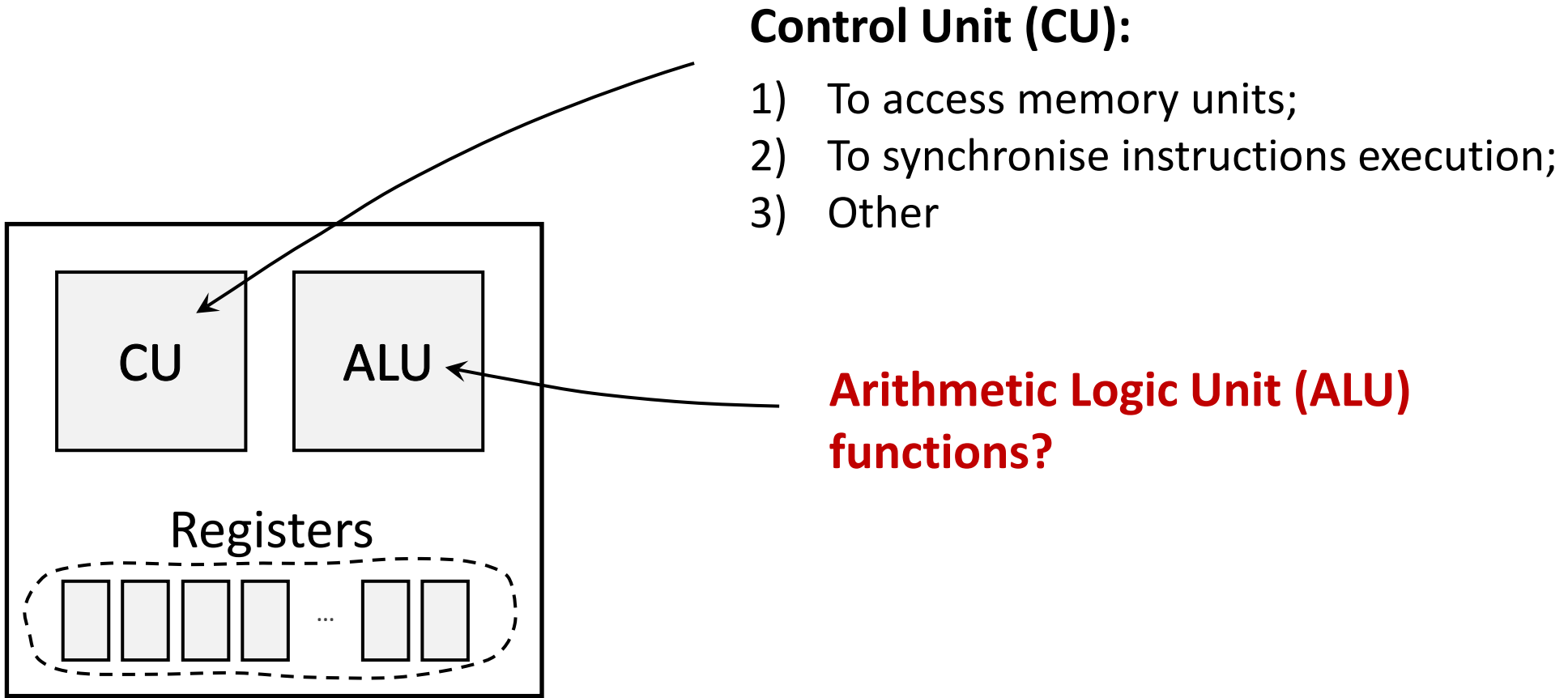
Recap: Key CPU Components



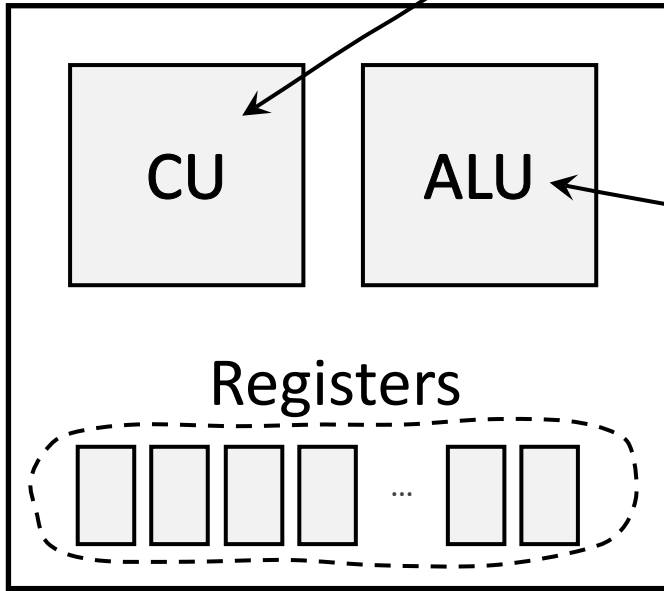
Control Unit (CU):

- 1) To access memory units;
- 2) To synchronise instructions execution;
- 3) Other

Recap: Key CPU Components



Recap: Key CPU Components



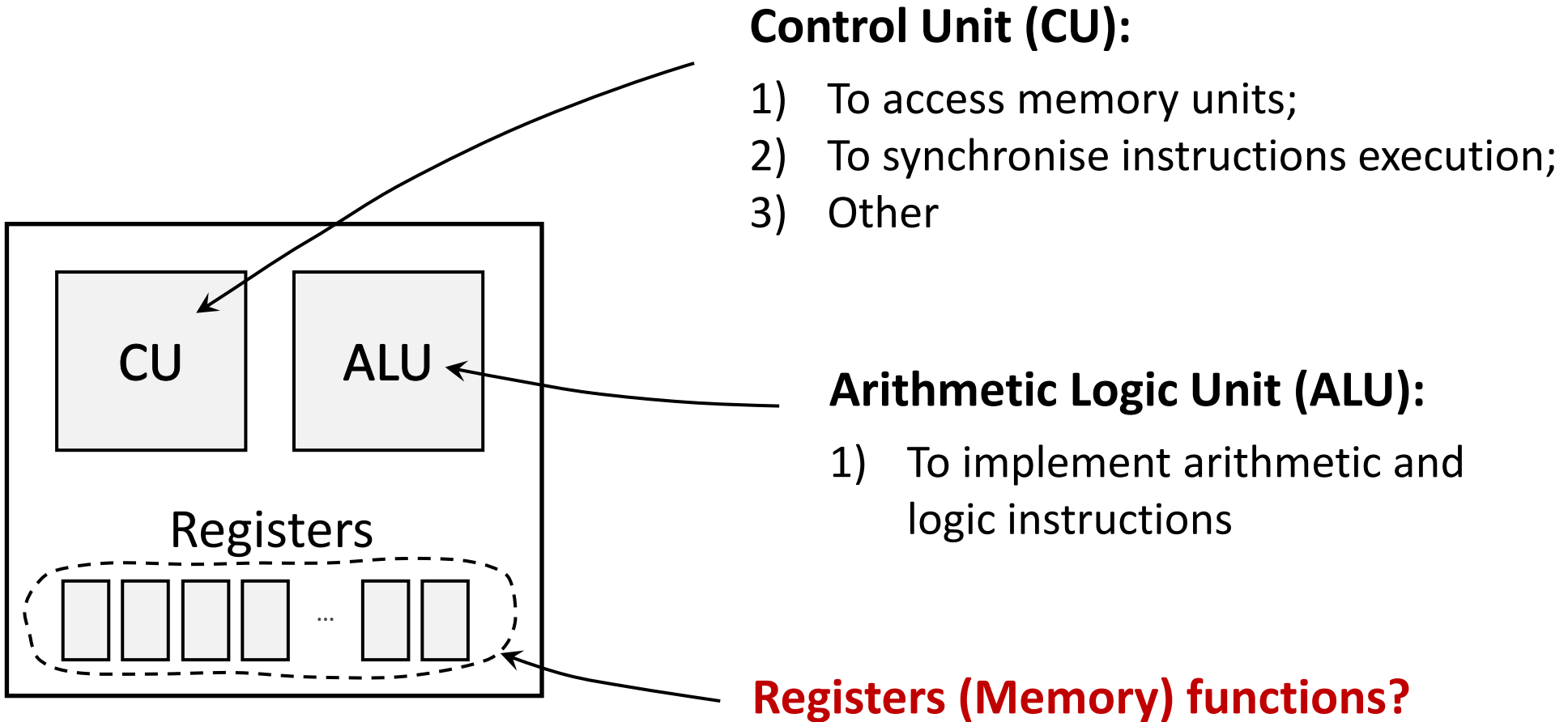
Control Unit (CU):

- 1) To access memory units;
- 2) To synchronise instructions execution;
- 3) Other

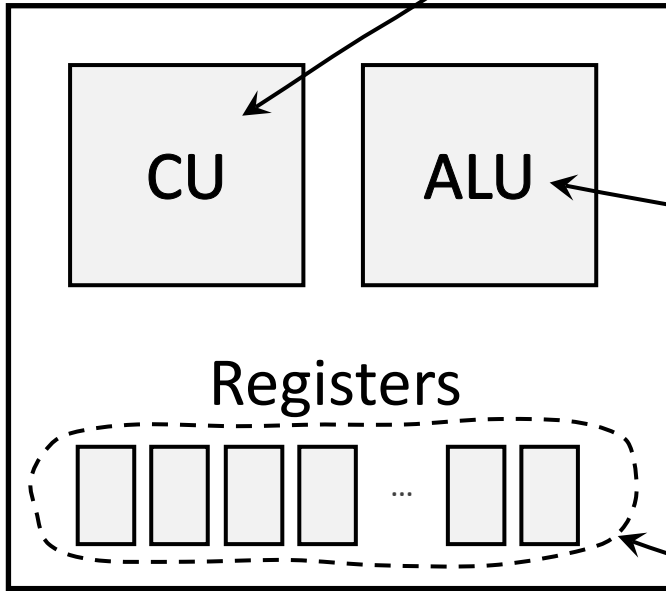
Arithmetic Logic Unit (ALU):

- 1) To implement arithmetic and logic instructions**

Recap: Key CPU Components



Recap: Key CPU Components



Control Unit (CU):

- 1) To access memory units;
- 2) To synchronise instructions execution;
- 3) Other

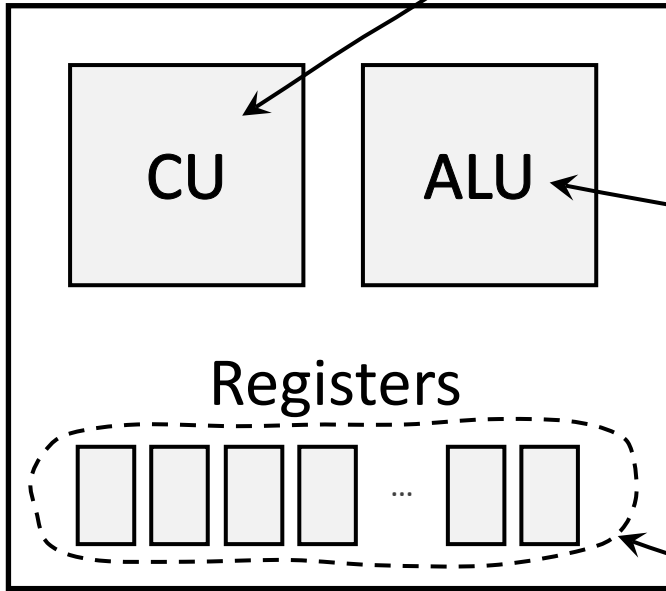
Arithmetic Logic Unit (ALU):

- 1) To implement arithmetic and logic instructions

Registers (Memory):

- 1) To synchronise input arguments provision to ALU;
- 2) To decode instructions;
- 3) To capture an instruction result from ALU

Recap: Key CPU Components



Control Unit (CU):

- 1) To access memory units;
- 2) To synchronise instructions execution;
- 3) Other

Strongly relies on Multiplexer circuits

Arithmetic Logic Unit (ALU):

- 1) To implement arithmetic and logic instructions

Registers (Memory):

- 1) To synchronise input arguments provision to ALU;
- 2) To decode instructions;
- 3) To capture an instruction result from ALU

Case 1: Multiplexer for ALU Implementation

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

ALU: Instruction Subcircuits

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Circuit for “+”

Circuit for “-”

Circuit for “*”

Circuit for “/”

ALU: Input Arguments

Let our ALU support 4 instructions: “+”, “-”, “*”, “/”

Arg. 1

Circuit for “+”

Circuit for “-”

Arg. 2

Circuit for “*”

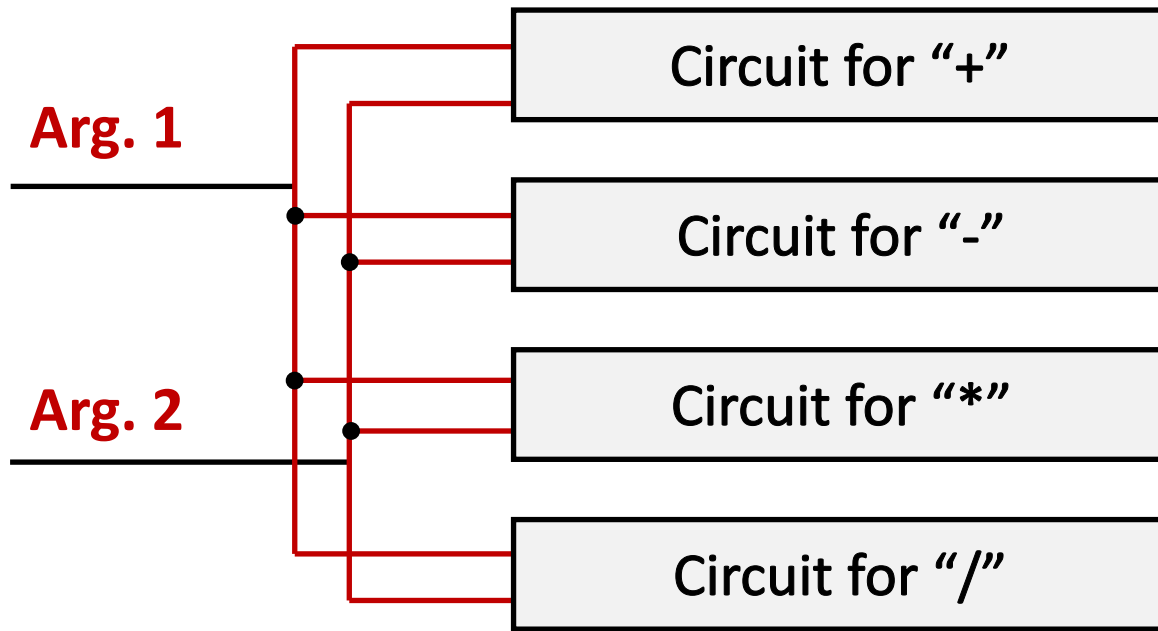
Circuit for “/”

Assumption:

Arguments are 1-bit each

ALU: Input Arguments

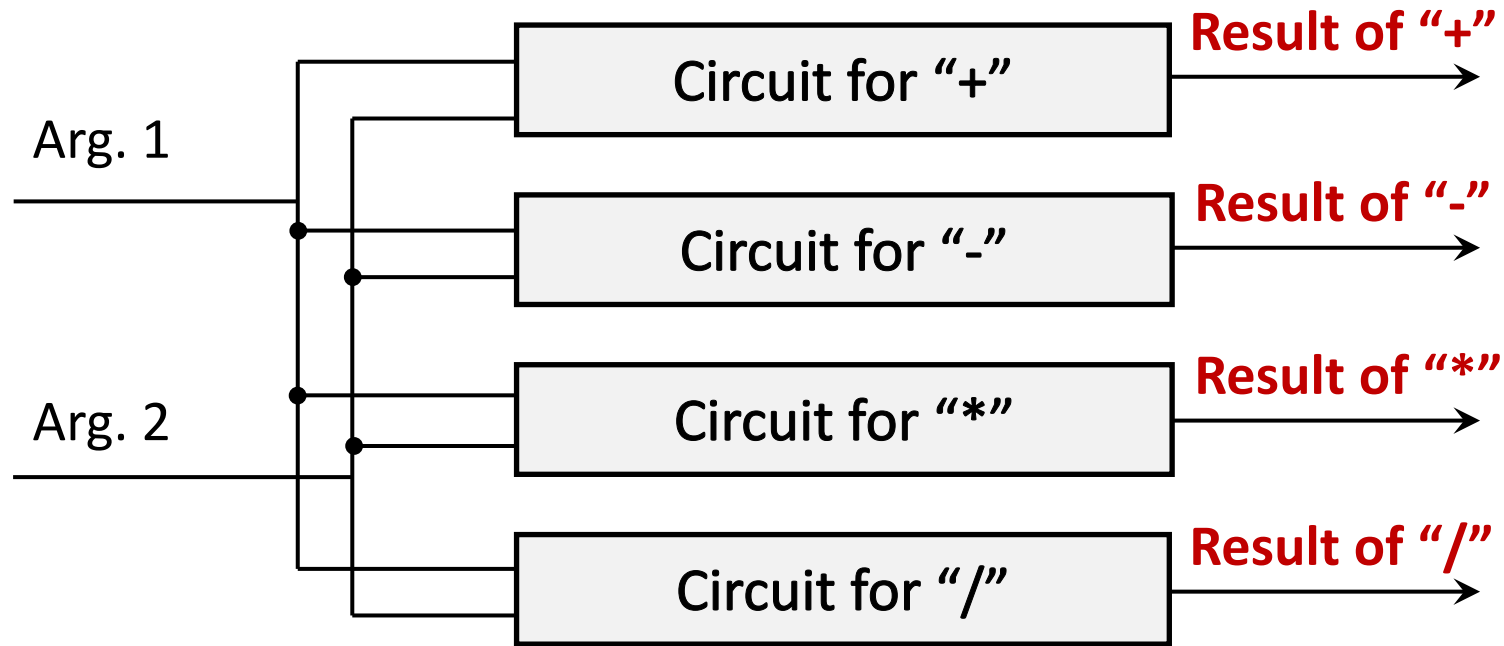
Let our ALU support 4 instructions: "+", "-", "*", "/"



Assumption:
Arguments are 1-bit each

ALU: Instruction Results

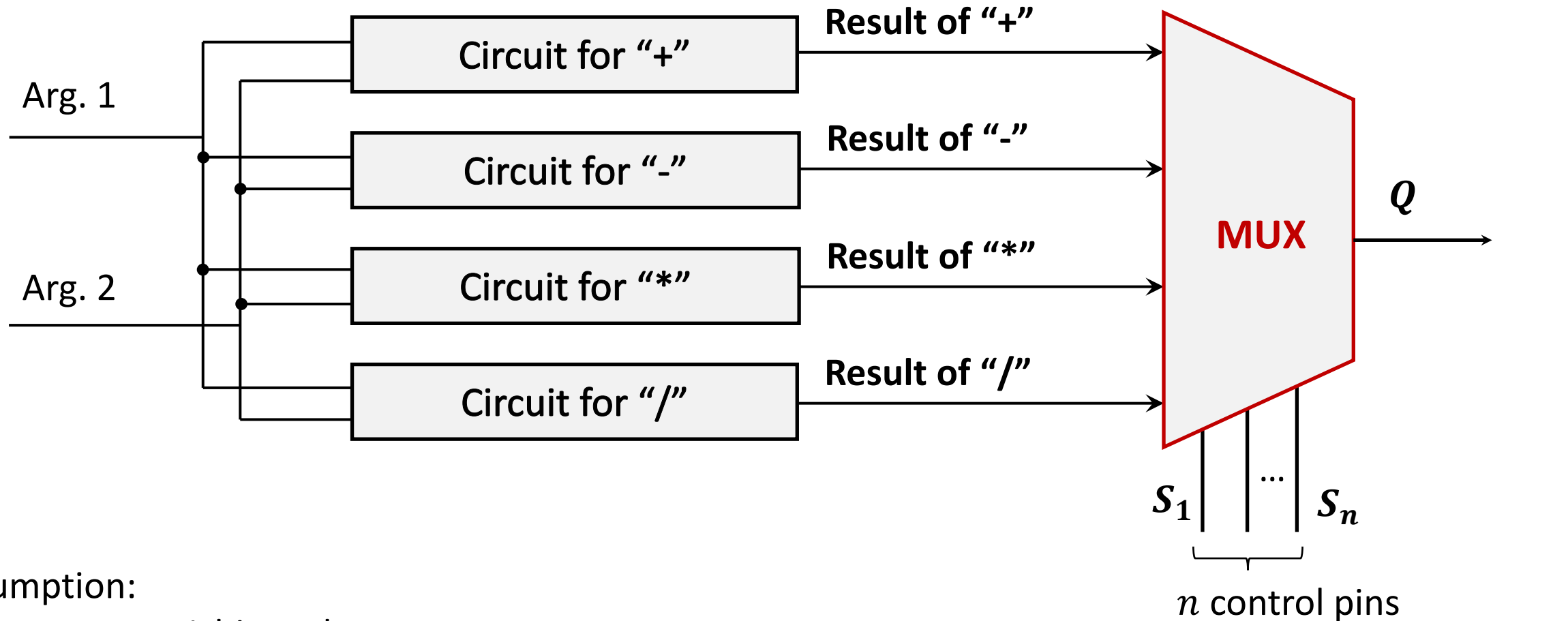
Let our ALU support 4 instructions: "+", "-", "*", "/"



Assumption:
Arguments are 1-bit each

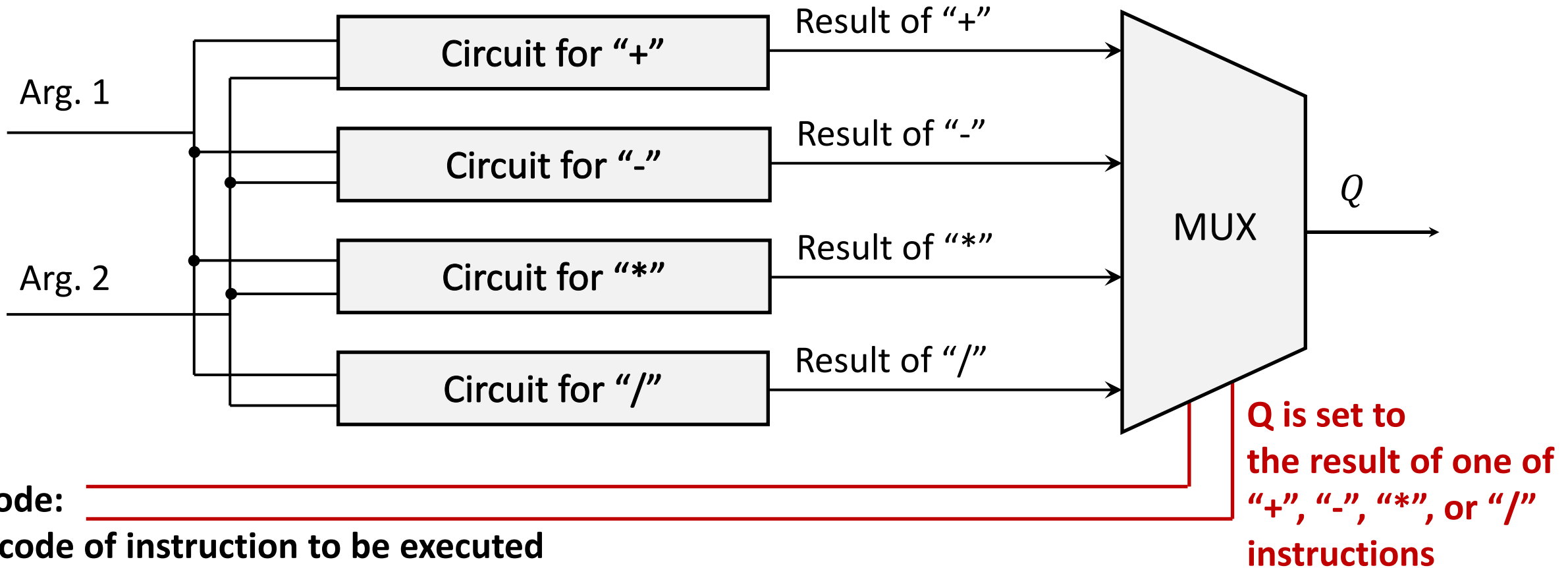
ALU: Multiplexer for Result Choice

Let our ALU support 4 instructions: "+", "-", "*", "/"



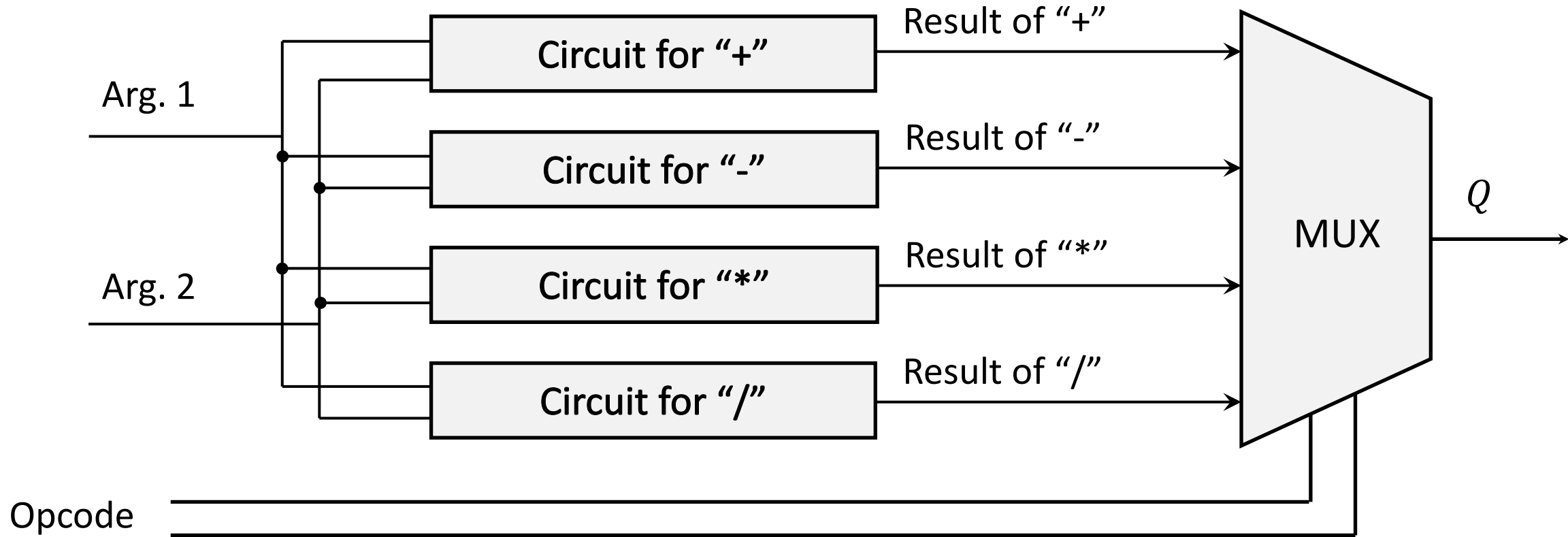
ALU: Instruction Code

Let our ALU support 4 instructions: "+", "-", "*", "/"



ALU: Propagation Delay

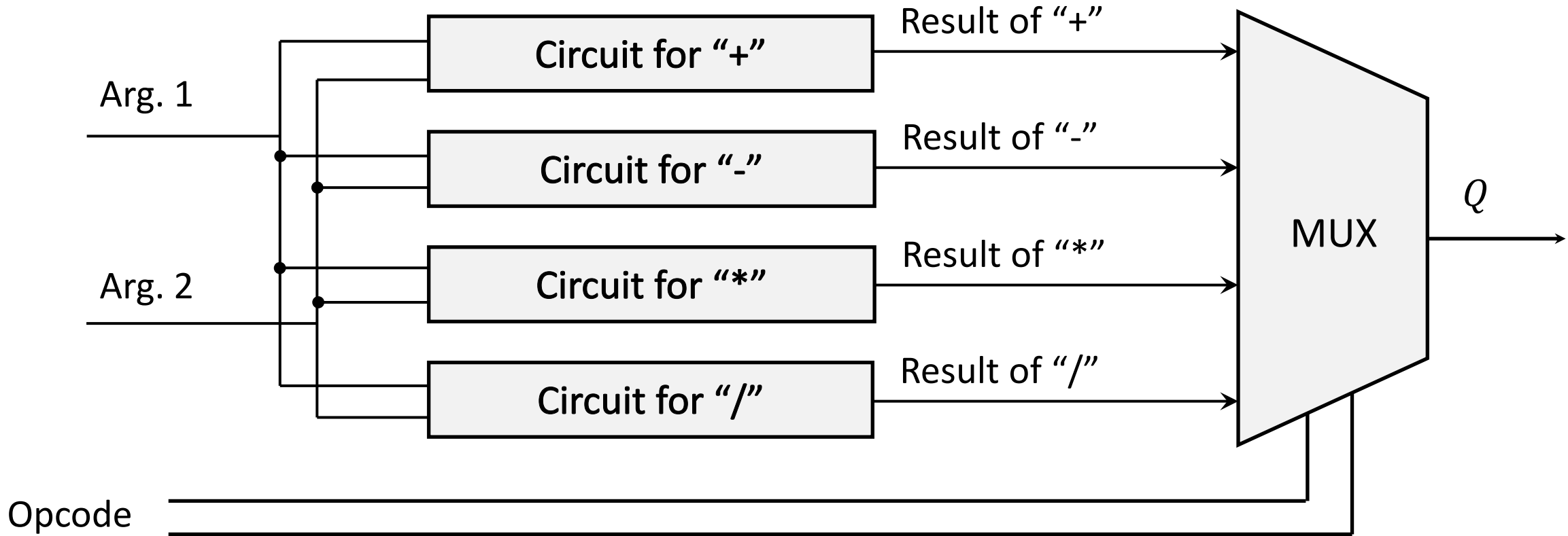
Let our ALU support 4 instructions: "+", "-", "*", "/"



CPU clock cycle length correlates with an execution time of the slowest supported instruction

ALU: Propagation Delay

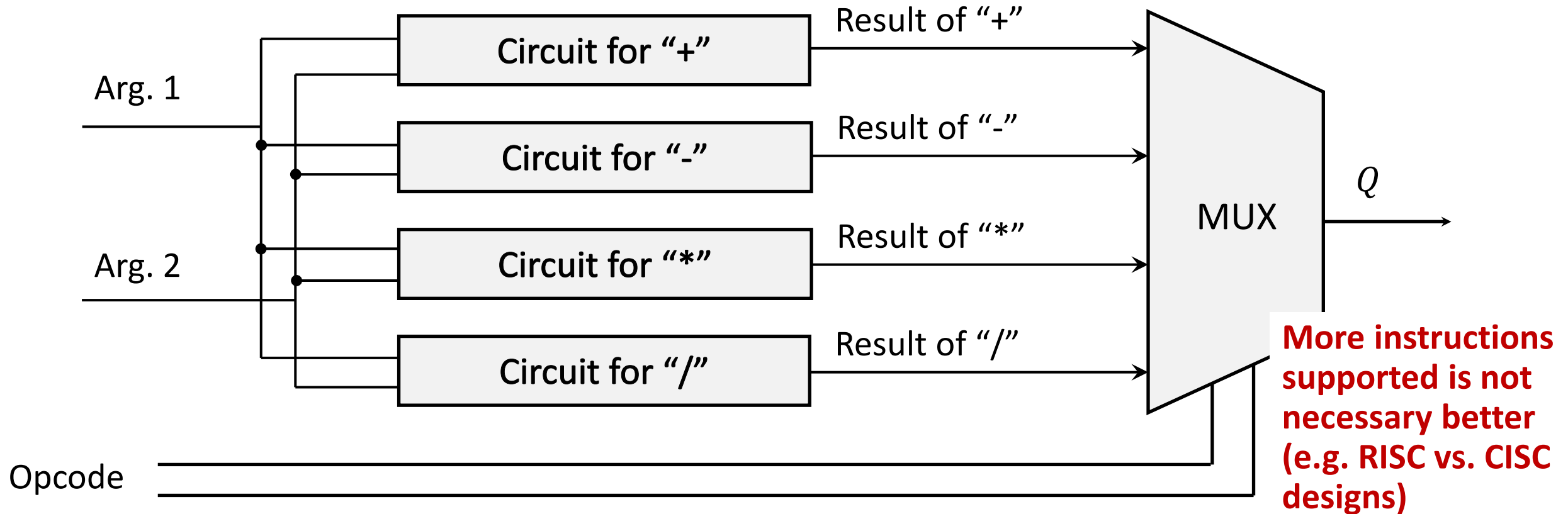
Let our ALU support 4 instructions: "+", "-", "*", "/"



CPU clock cycle length correlates with an execution time of the slowest supported instruction
For example, in a simple CPU design, the slowest ALU instruction must complete within 1 clock cycle

Side note: RISC vs. CISC Architectures

Let our ALU support 4 instructions: "+", "-", "*", "/"



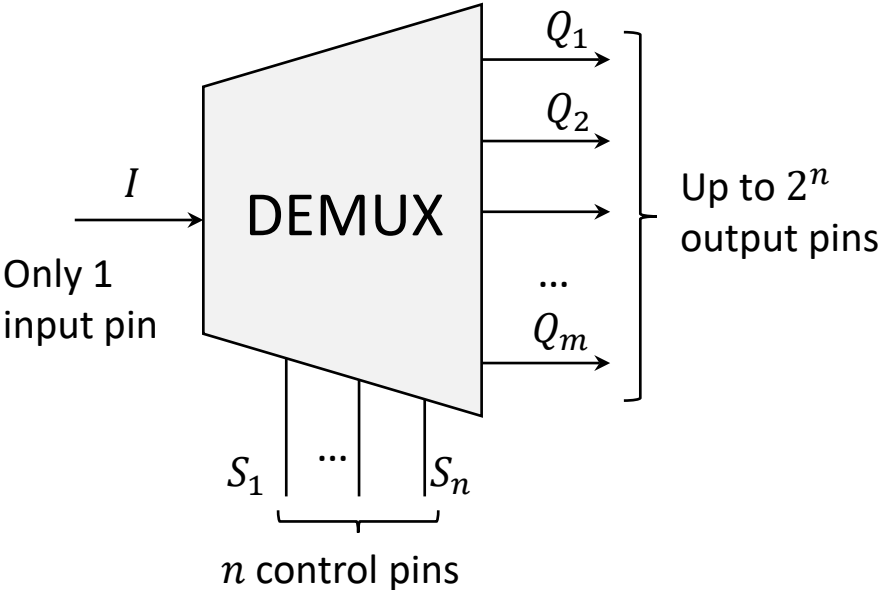
CPU clock cycle length correlates with an execution time of the slowest supported instruction
For example, in a simple CPU design, the slowest ALU instruction must complete within 1 clock cycle

Case 2: Demultiplexer for Memory Unit Write

Memory Unit –
a set of memory cells
(each cell has its address)

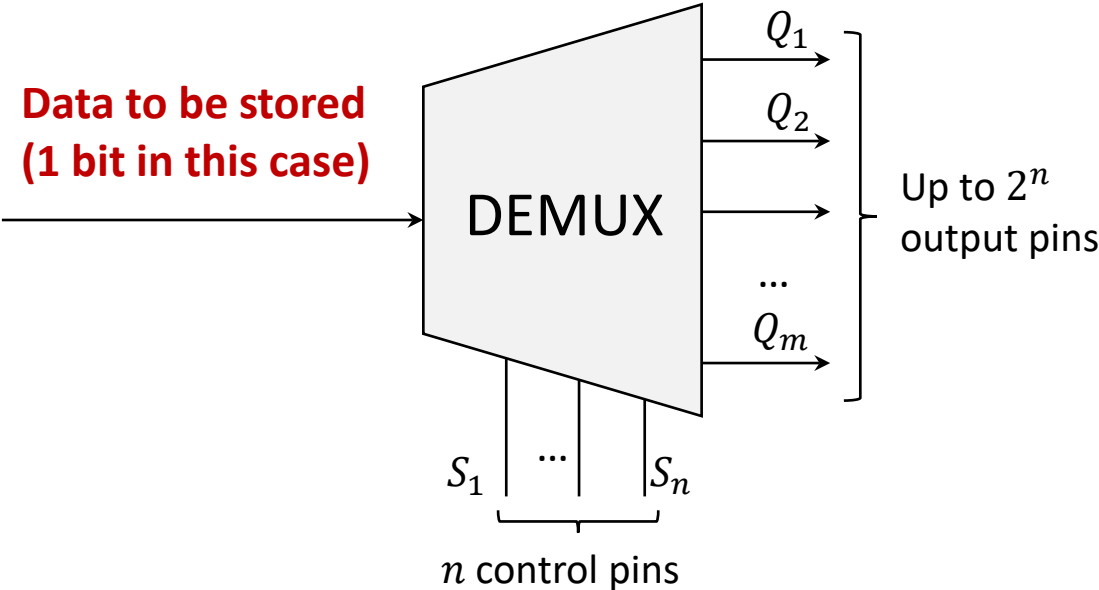
Demultiplexer for Memory Unit Write

Memory Unit –
a set of memory cells
(each cell has its address)



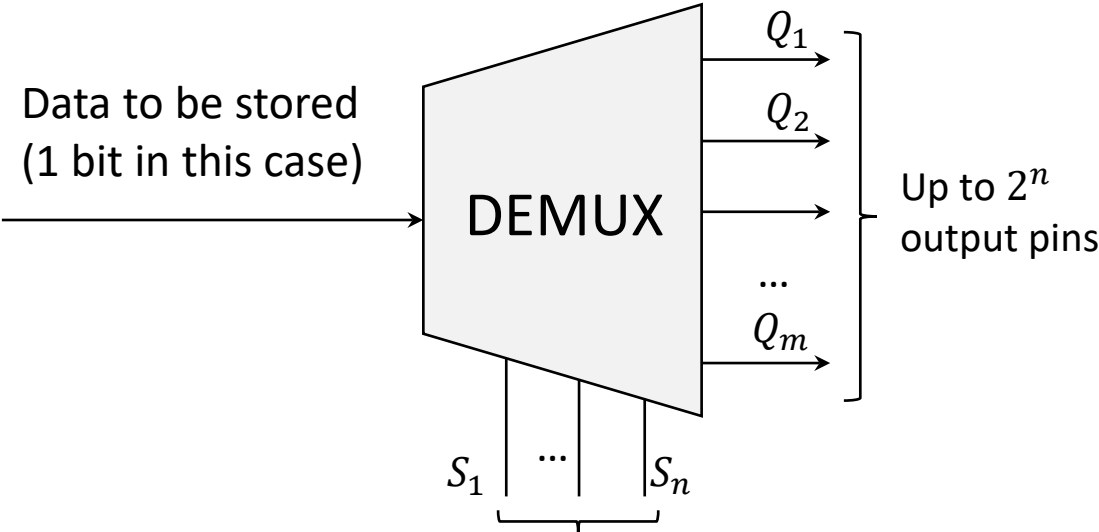
Memory Write: Input Bit

Memory Unit –
a set of memory cells
(each cell has its address)



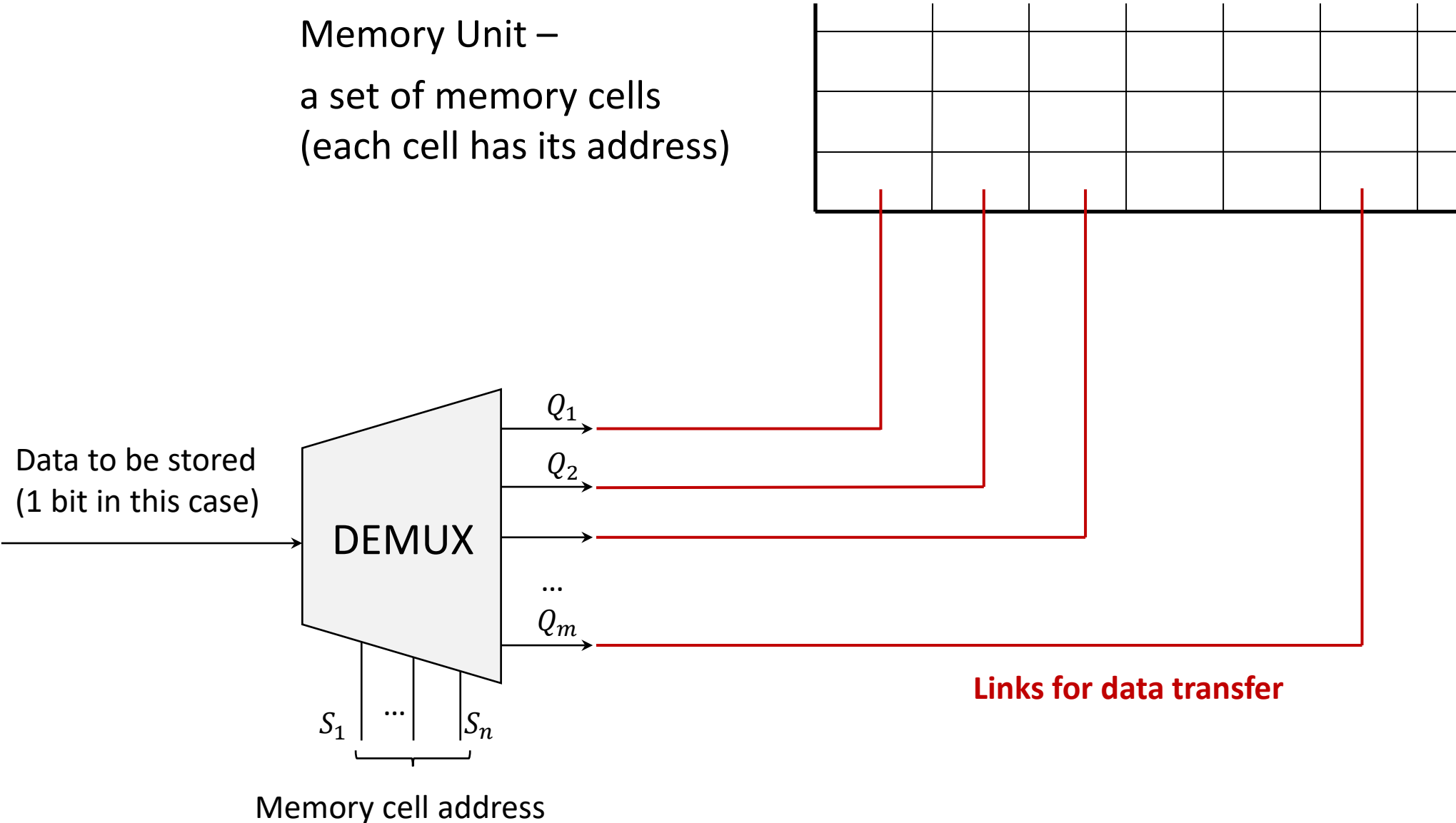
Memory Write: Data Address

Memory Unit –
a set of memory cells
(each cell has its address)

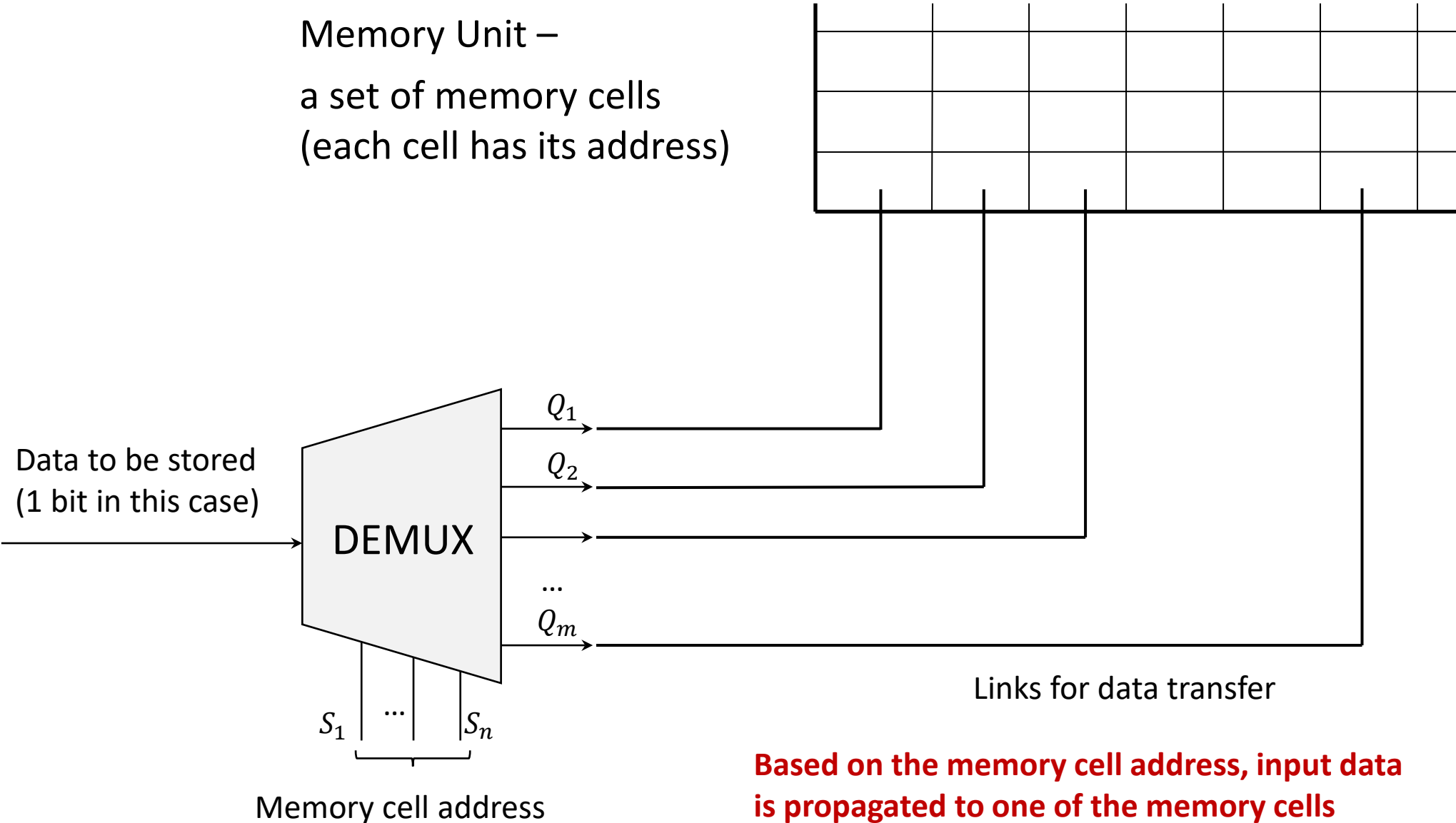


Memory cell address

Memory Write: Data Links



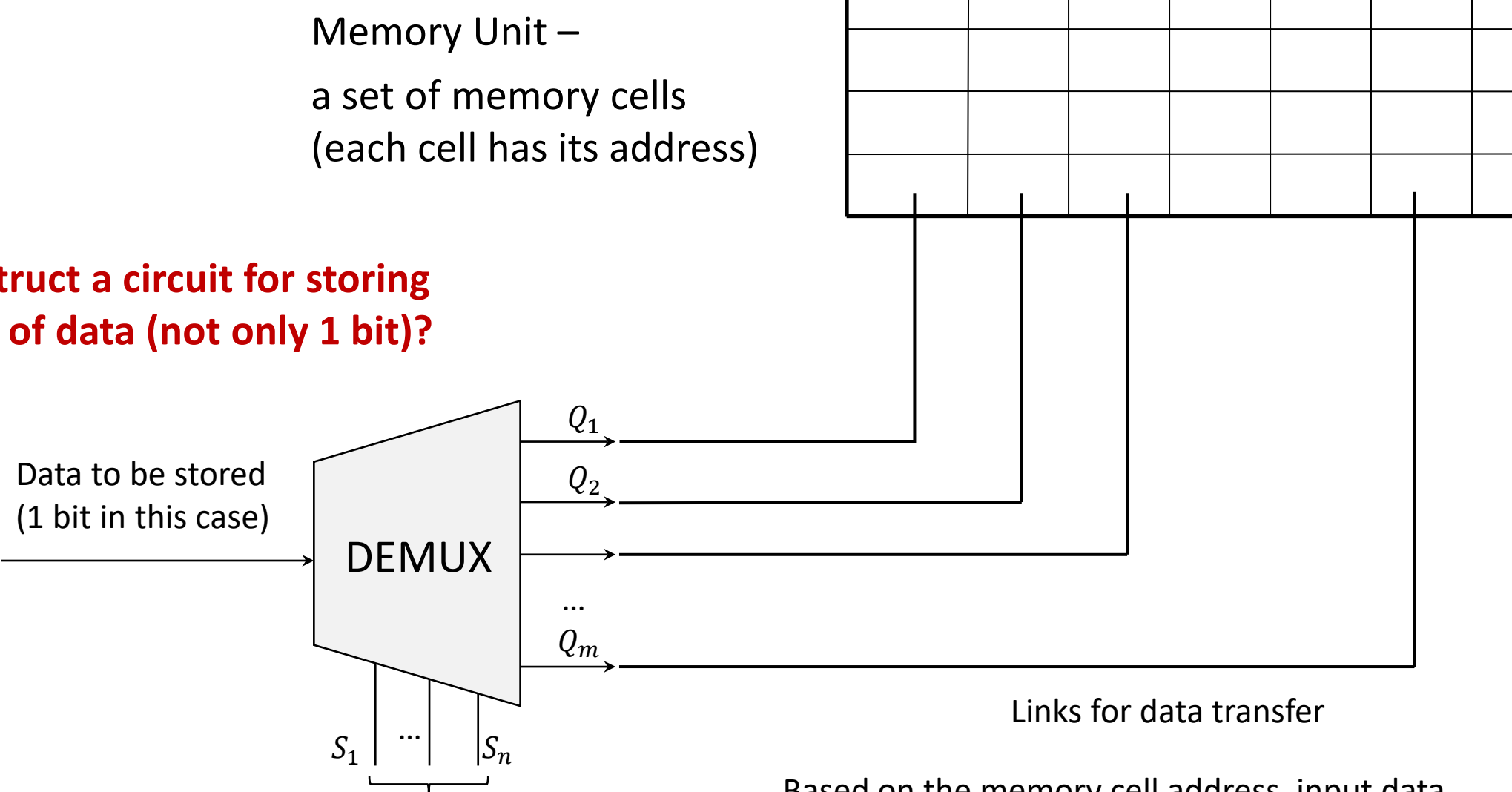
Memory Write: Data Propagation



Memory Write: Simultaneous Multiple Bits Storage

Memory Unit –
a set of memory cells
(each cell has its address)

Question:
How to construct a circuit for storing multiple bits of data (not only 1 bit)?



Memory cell address

Based on the memory cell address, input data is propagated to one of the memory cells

Memory Write: X-to-Y Multiplexer for Multiple Bits Storage

X-to-Y Multiplexer:

