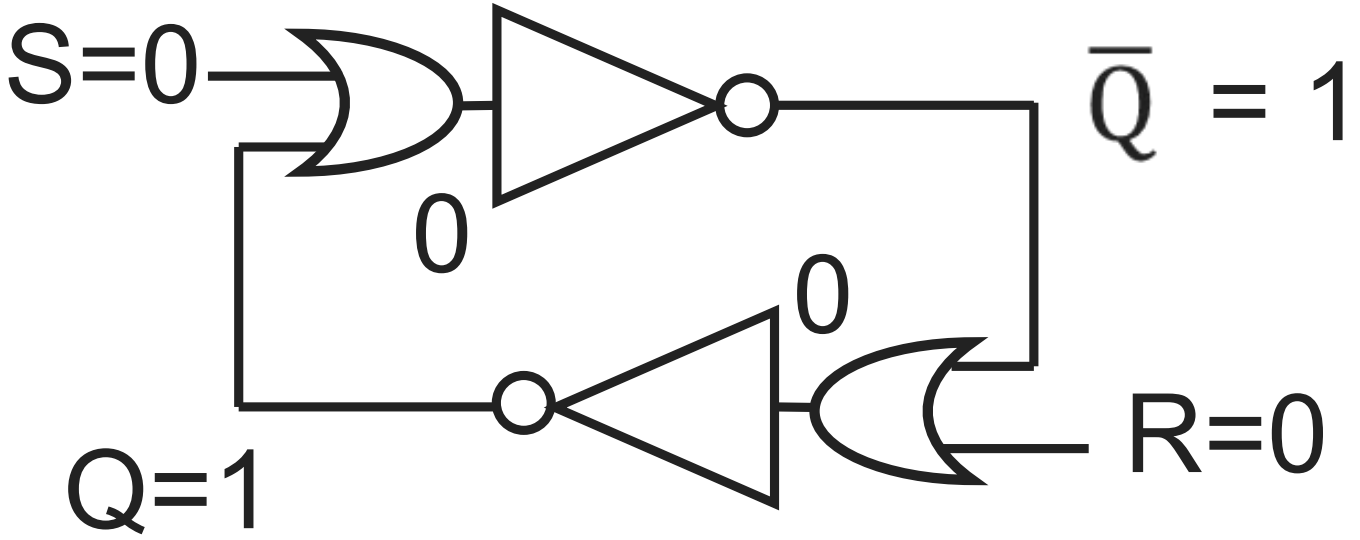
Review: State

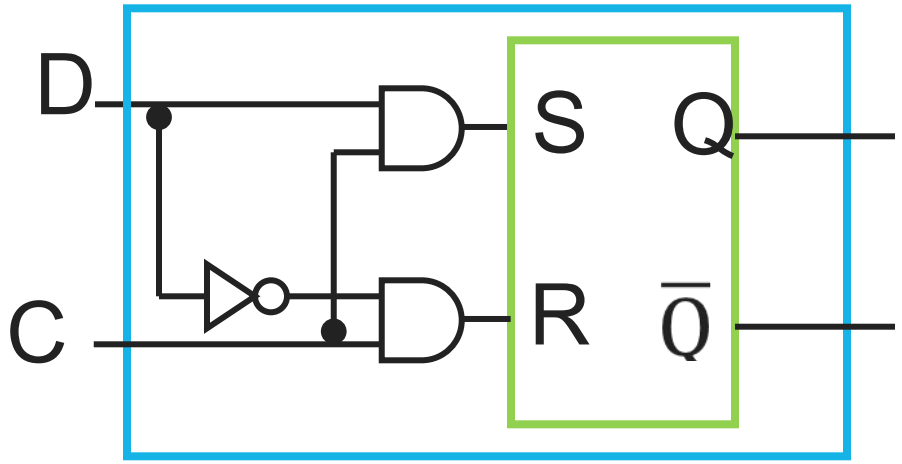
CS 3410: Computer System Organization and Programming

Set-Reset Latch



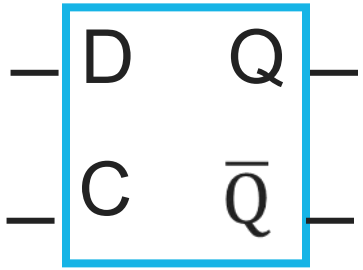
S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			Forbidde

D Latch

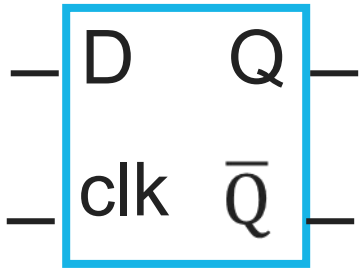


- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state

C = 1, D Latch *transparent*:
set/reset (according to D)
C = 0, D Latch *opaque*:
keep state (ignore D)

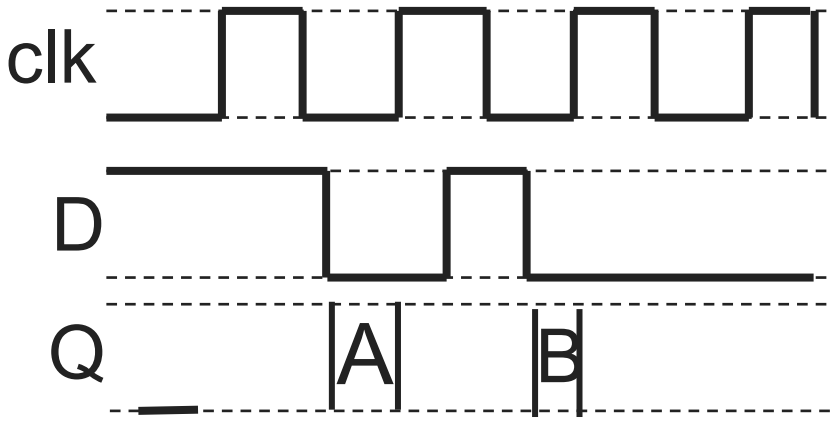


C	D	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
1	0	0	1	*Reset*
1	1	1	0	*Set*

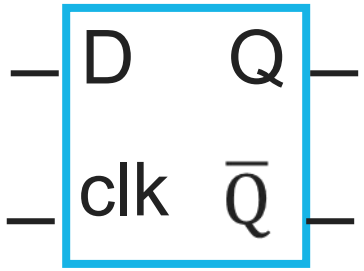


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1

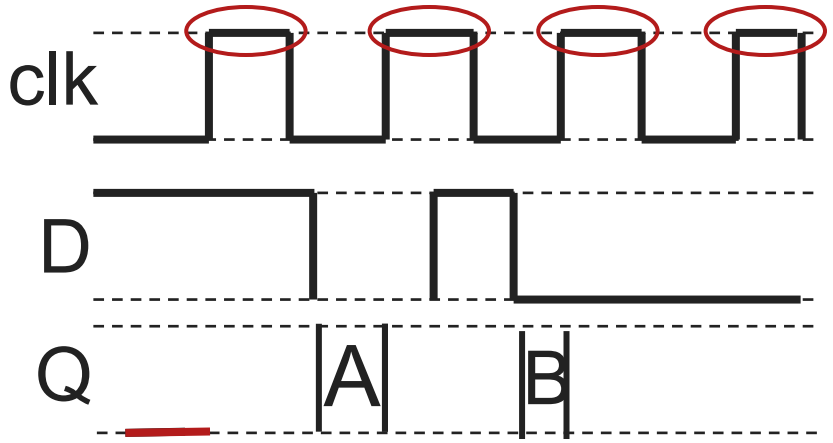


clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

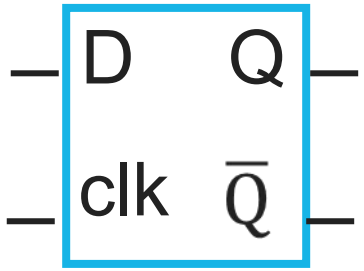


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1

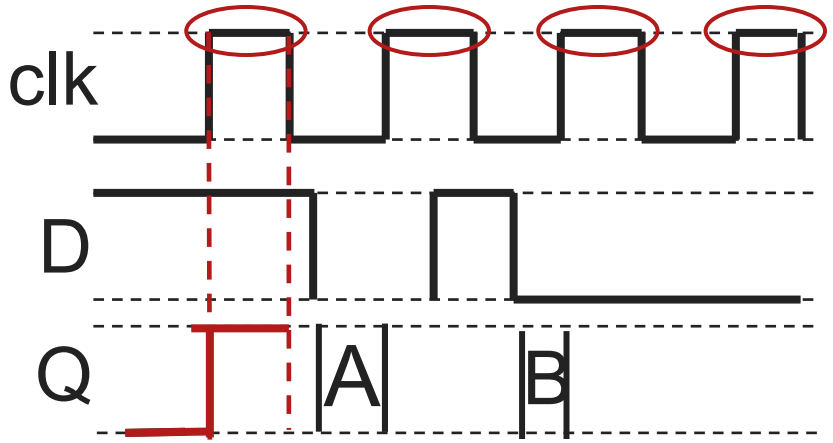


clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

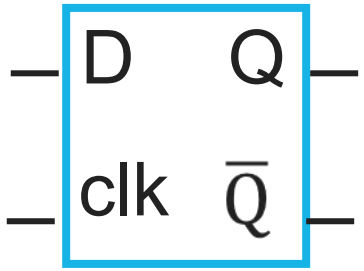


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1

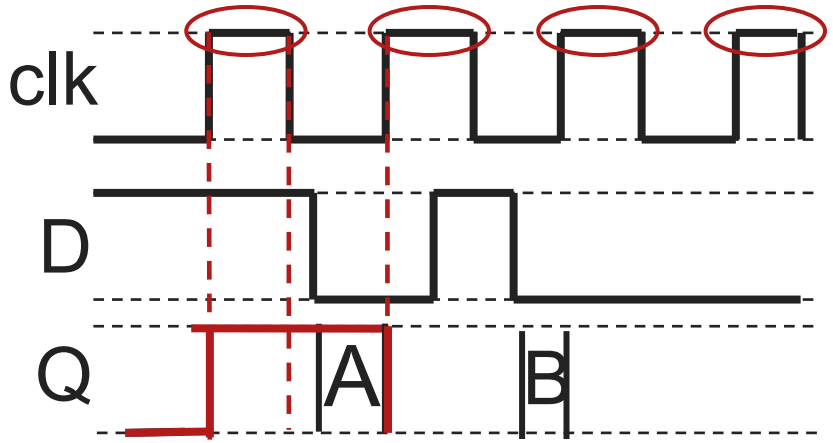


clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

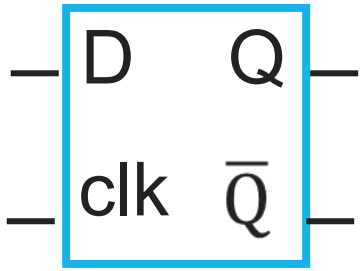


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1

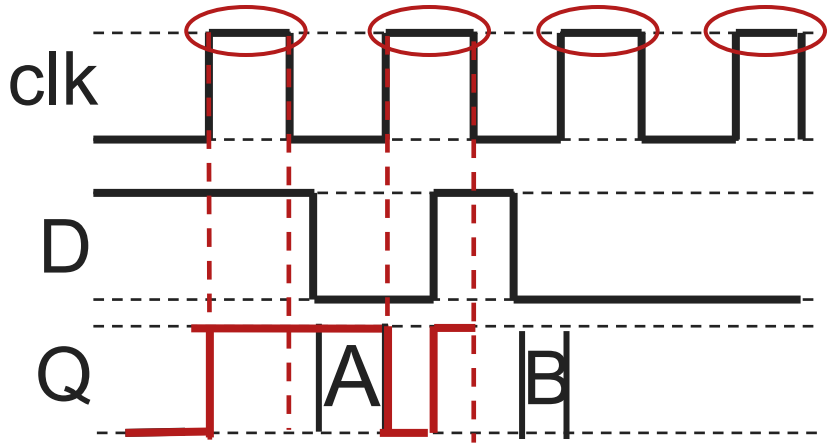


clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

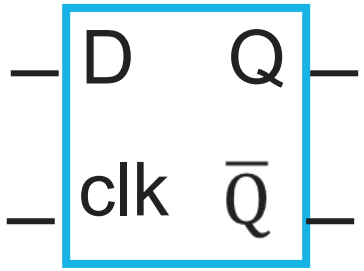


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1

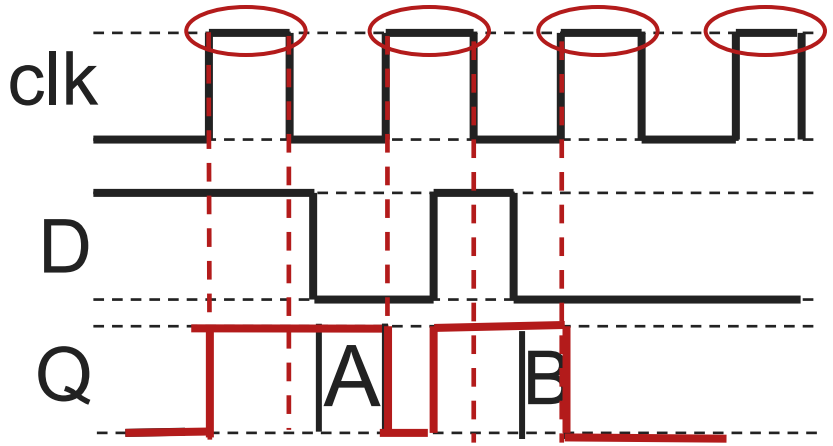


clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

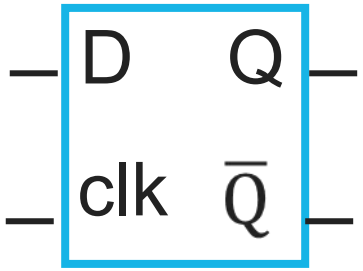


What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1



clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0



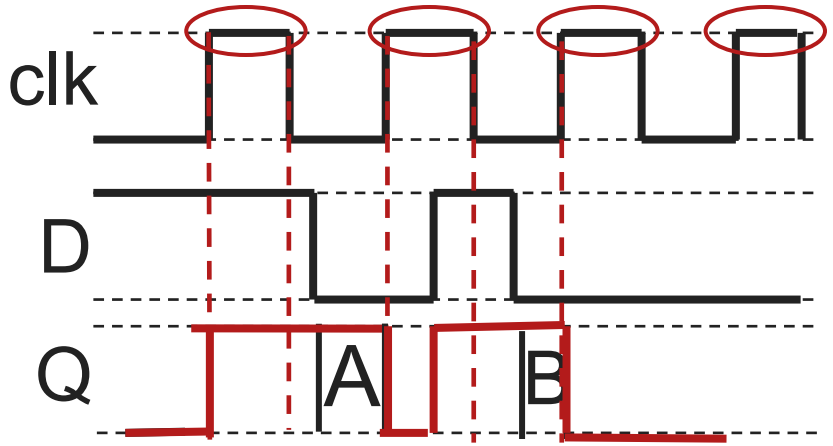
Level Sensitive D Latch

Clock high:

set/reset (according to D)

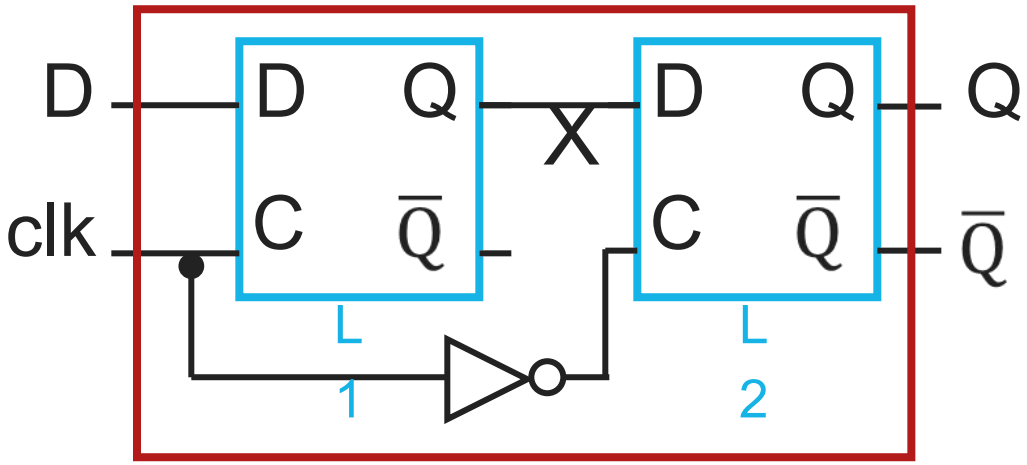
Clock low:

keep state (ignore D)



clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

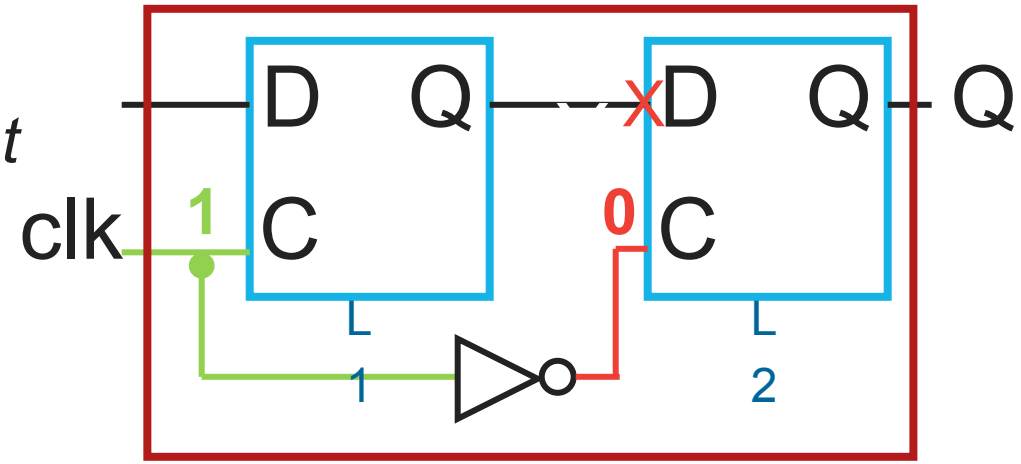
D Flip-Flop



- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges

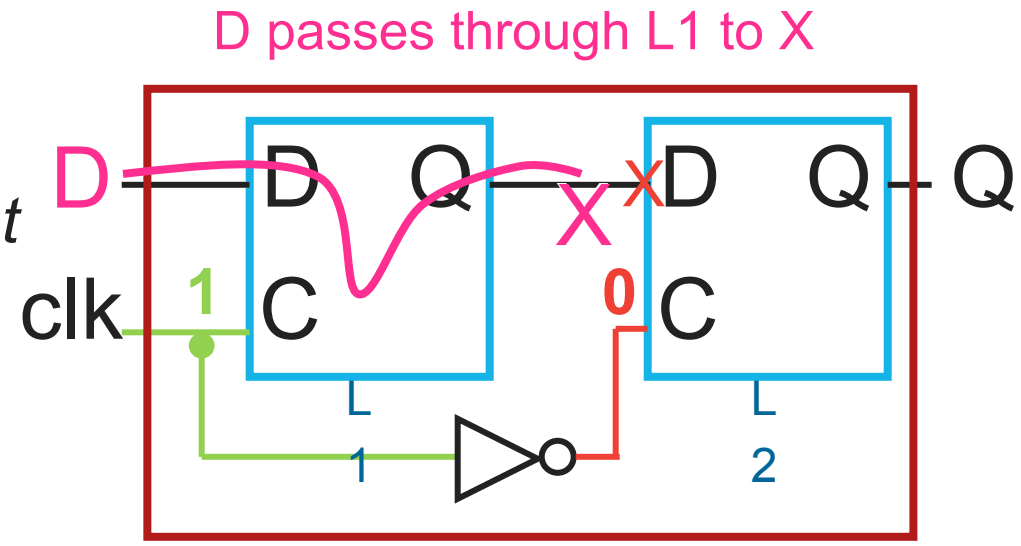
D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*



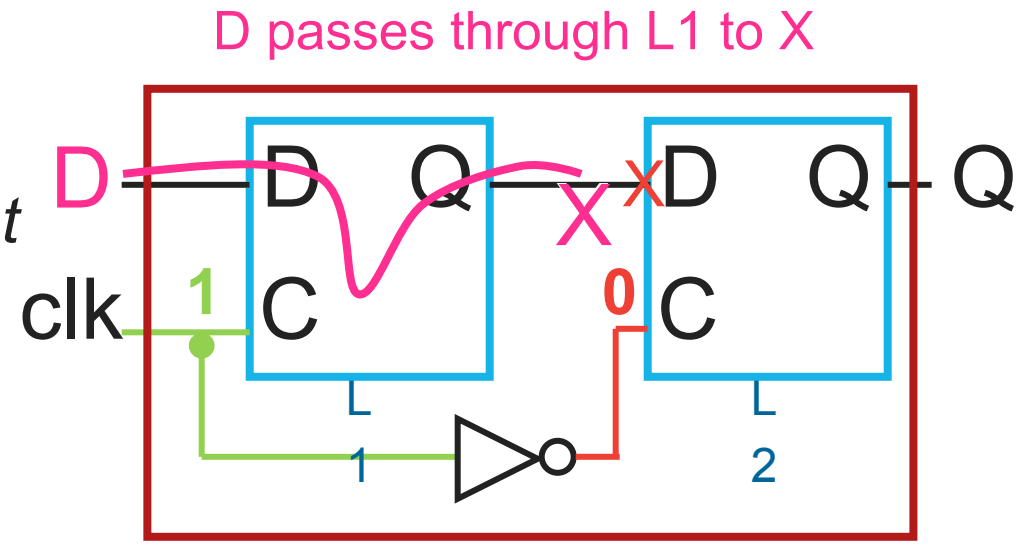
D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*



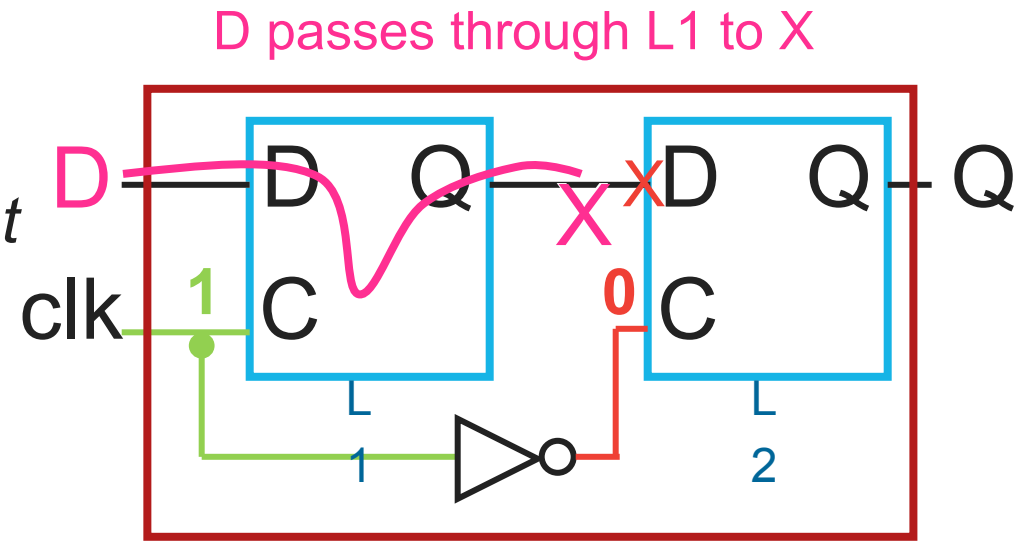
D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*

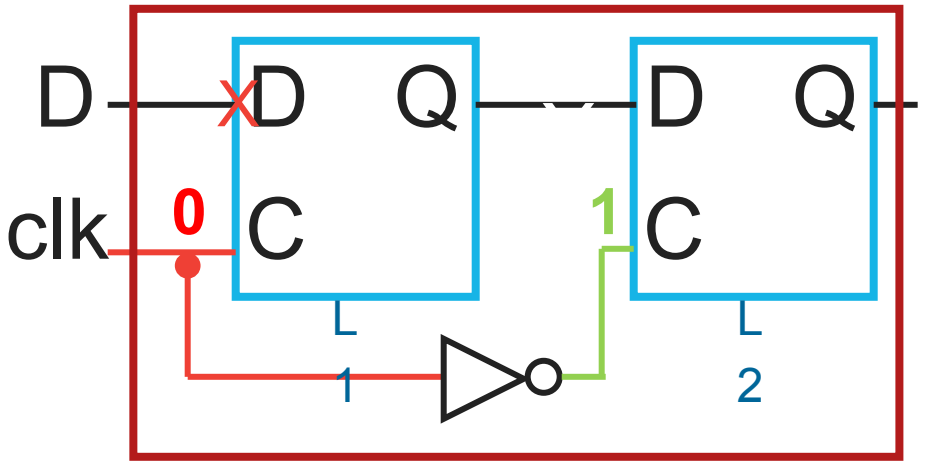


D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*

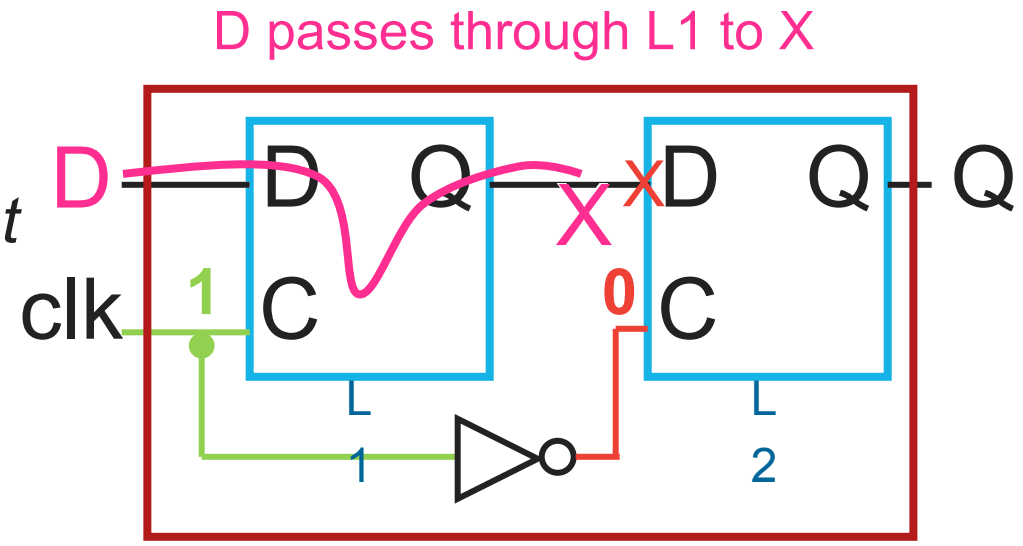


Clock = 0: L1 *opaque*
L2 *transparent*

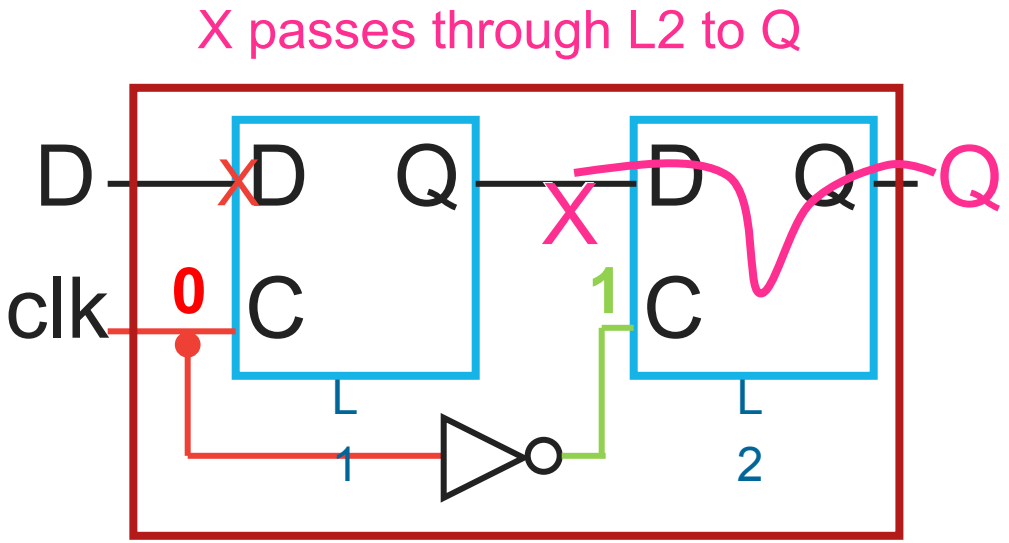


D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*

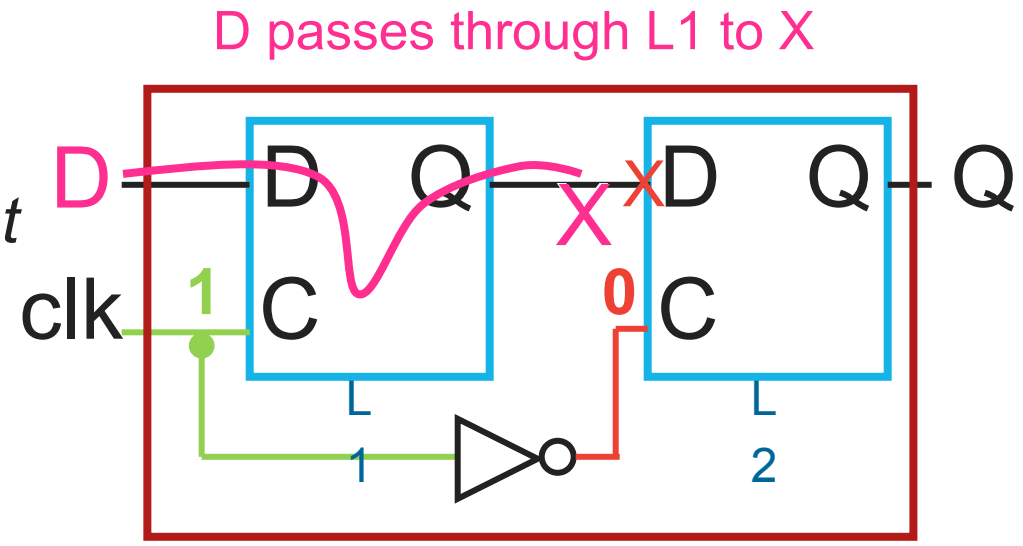


Clock = 0: L1 *opaque*
L2 *transparent*

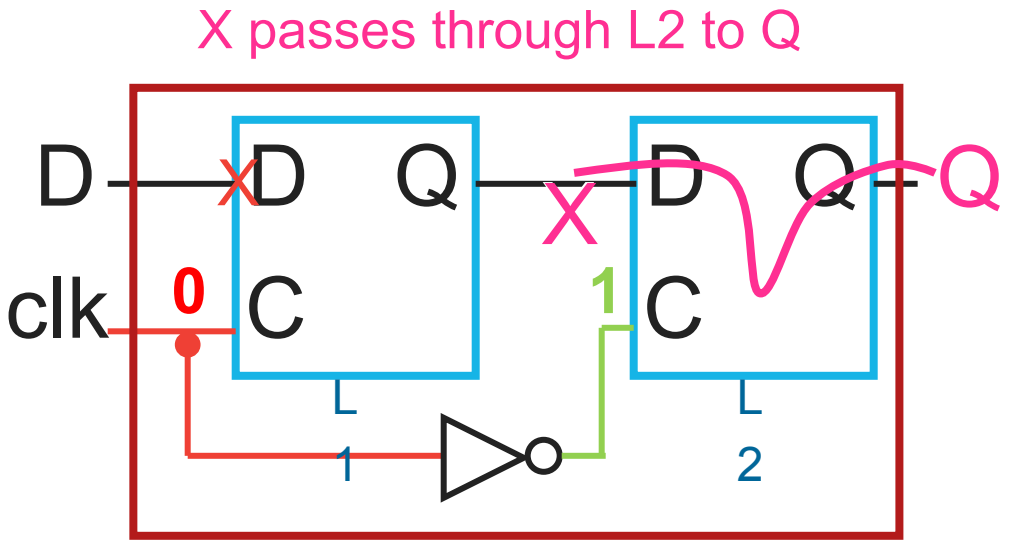


D Flip-Flop

Clock = 1: L1 *transparent*
L2 *opaque*

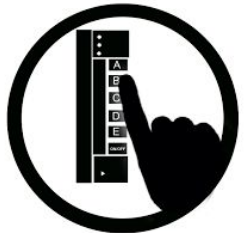


Clock = 0: L1 *opaque*
L2 *transparent*

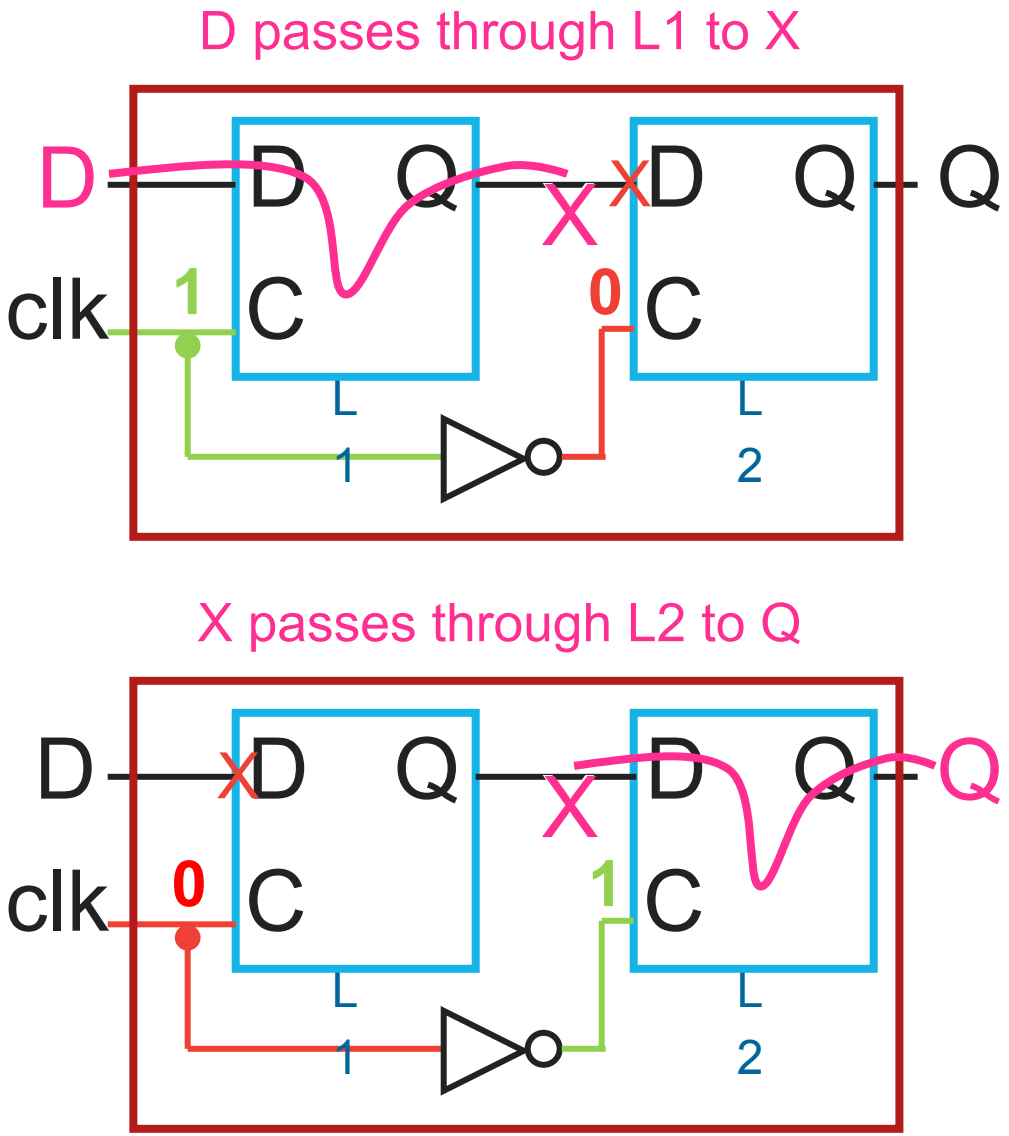


D Flip-Flop

When is data sampled?



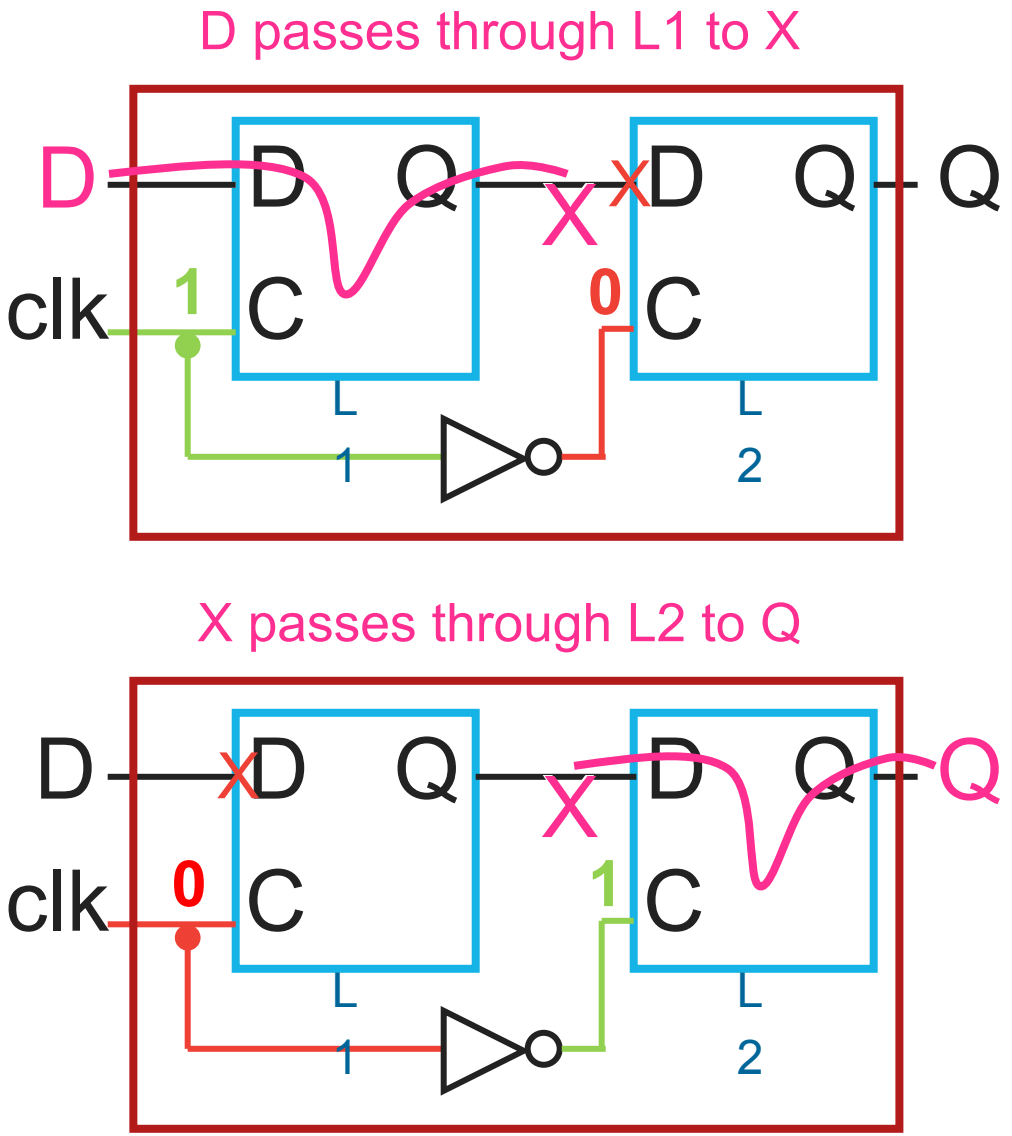
PollEv.com/cs3410



D Flip-Flop

When is data sampled?

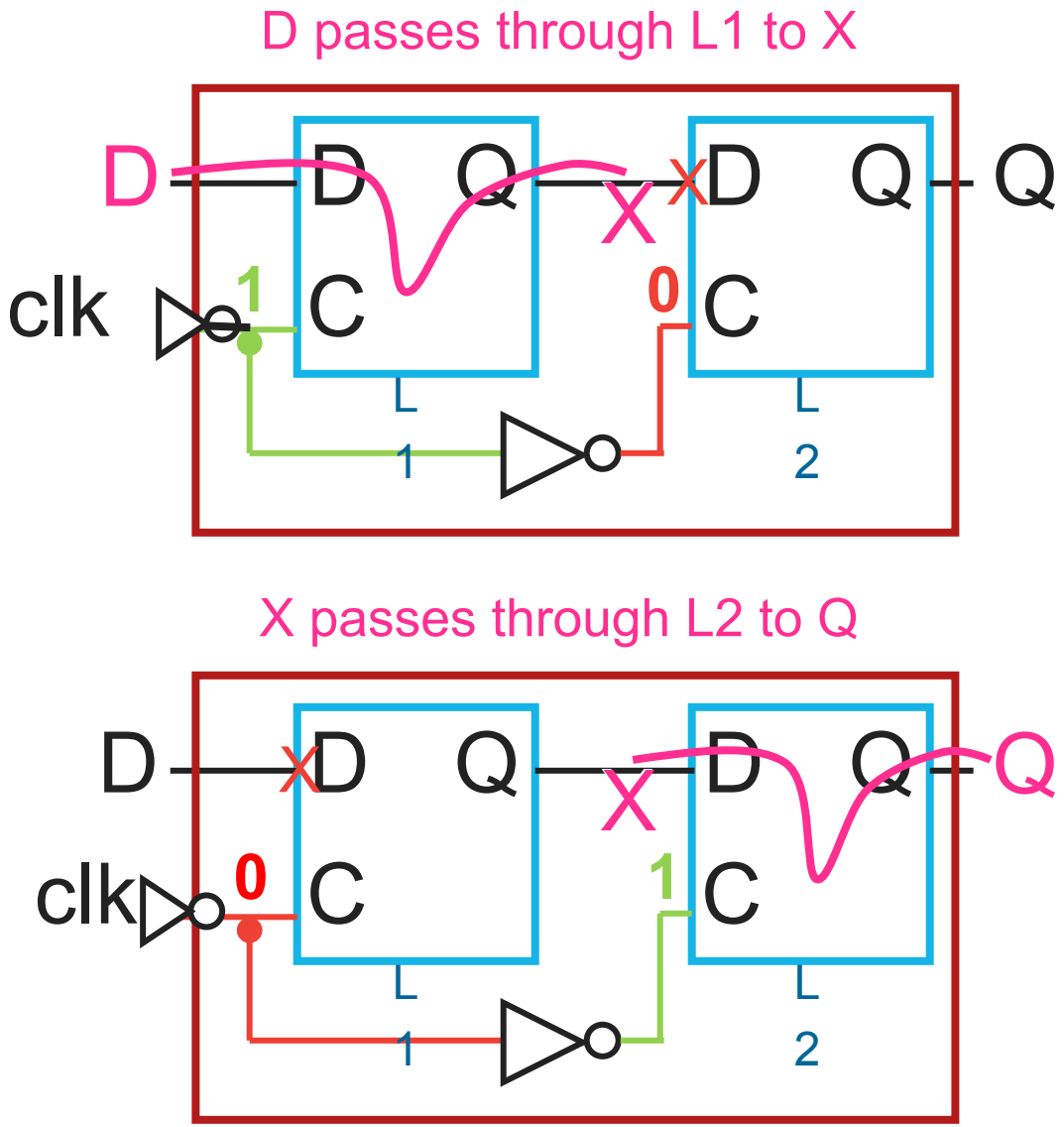
How can we change when data is sampled?

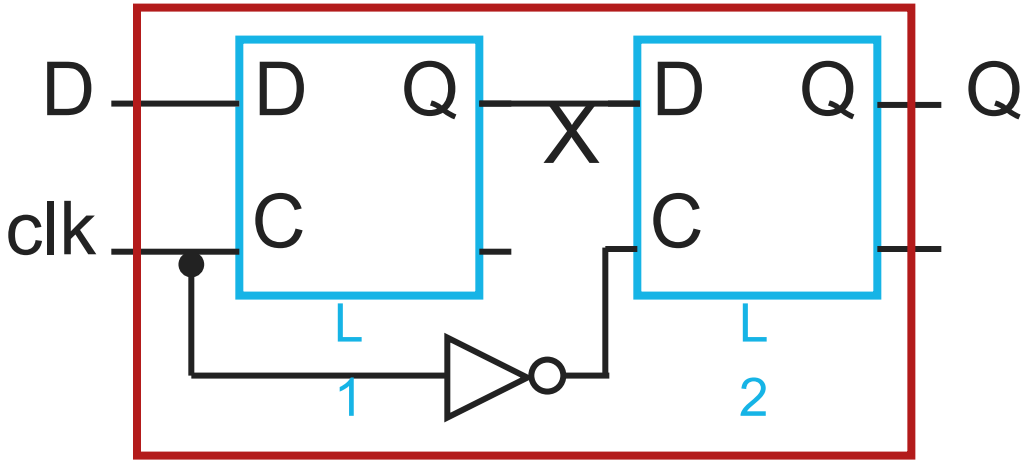


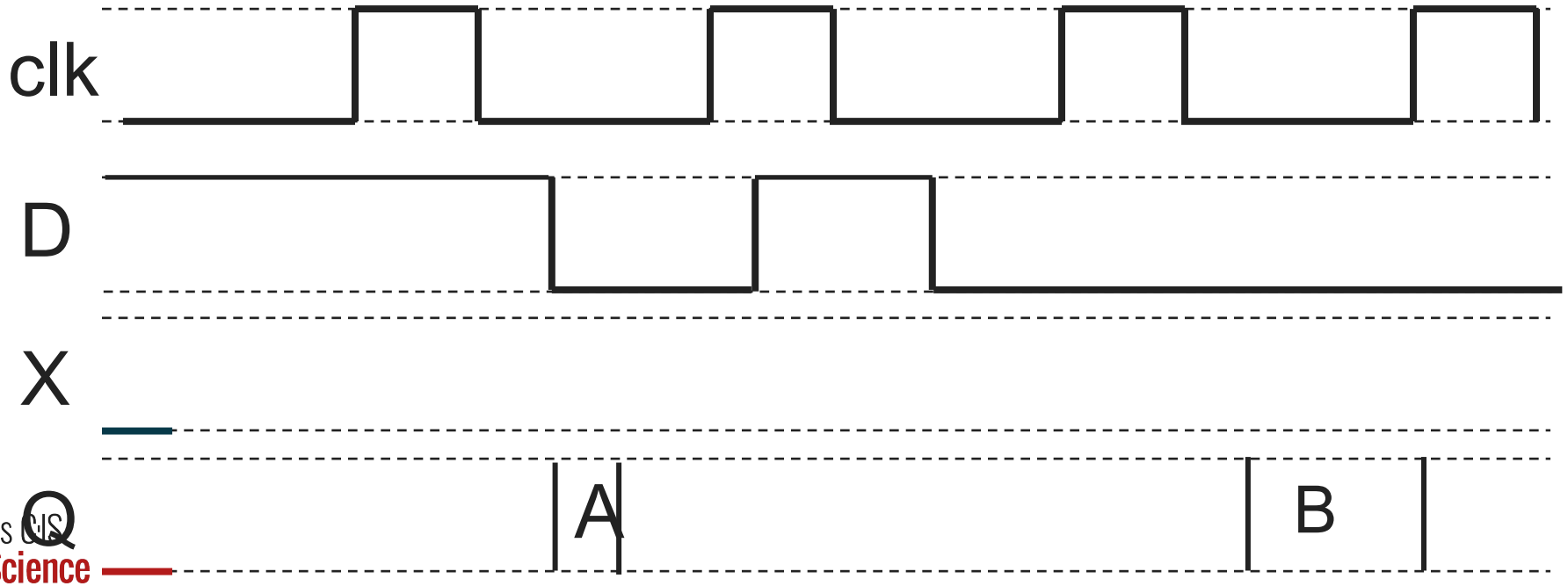
D Flip-Flop

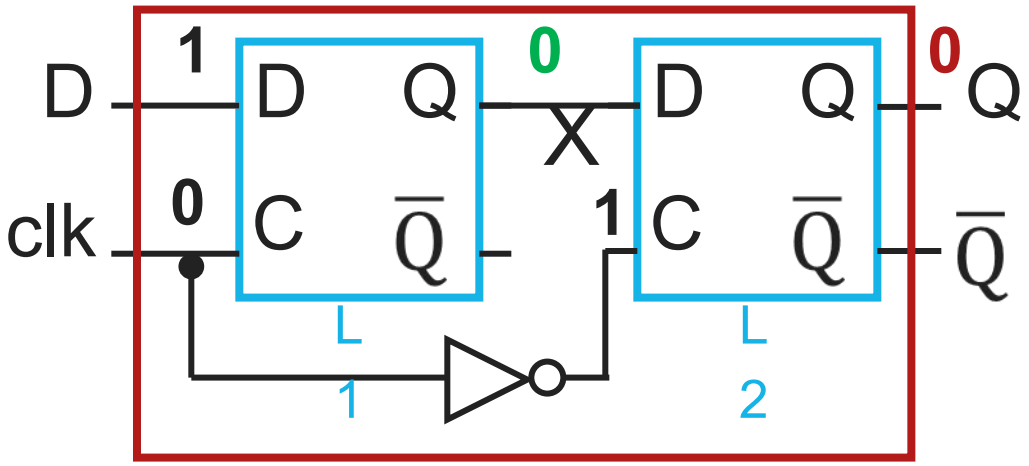
When is data sampled?

How can we change when data is sampled?



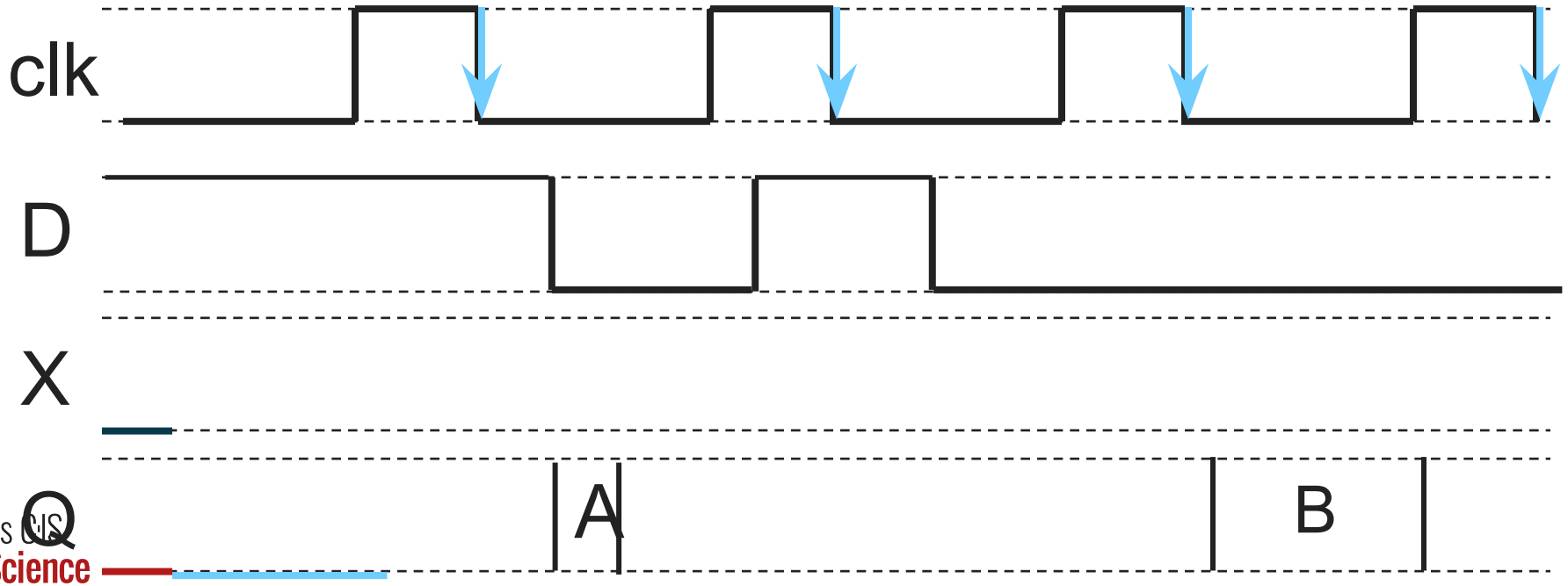

D
clk
D
Q
C
L
1
X
D
Q
C
L
2
Q

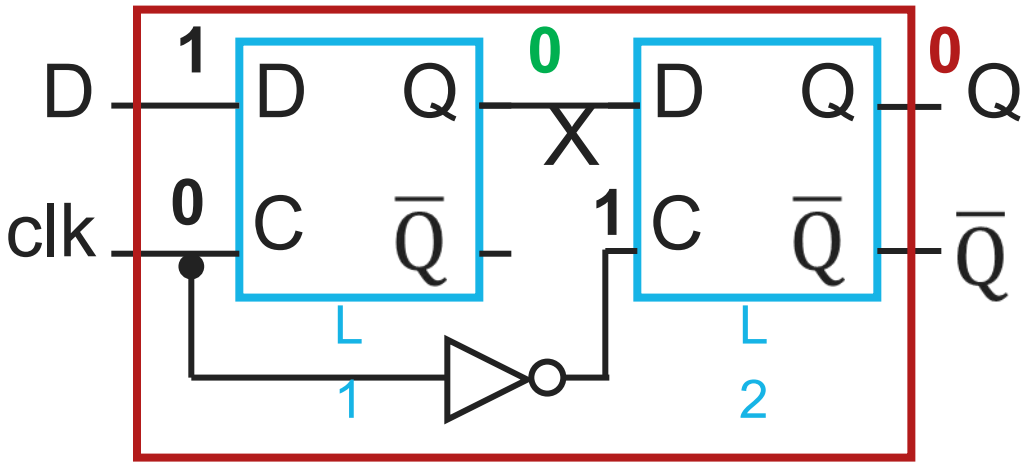

clk
D
X
Q
A
B



D Flip-Flop

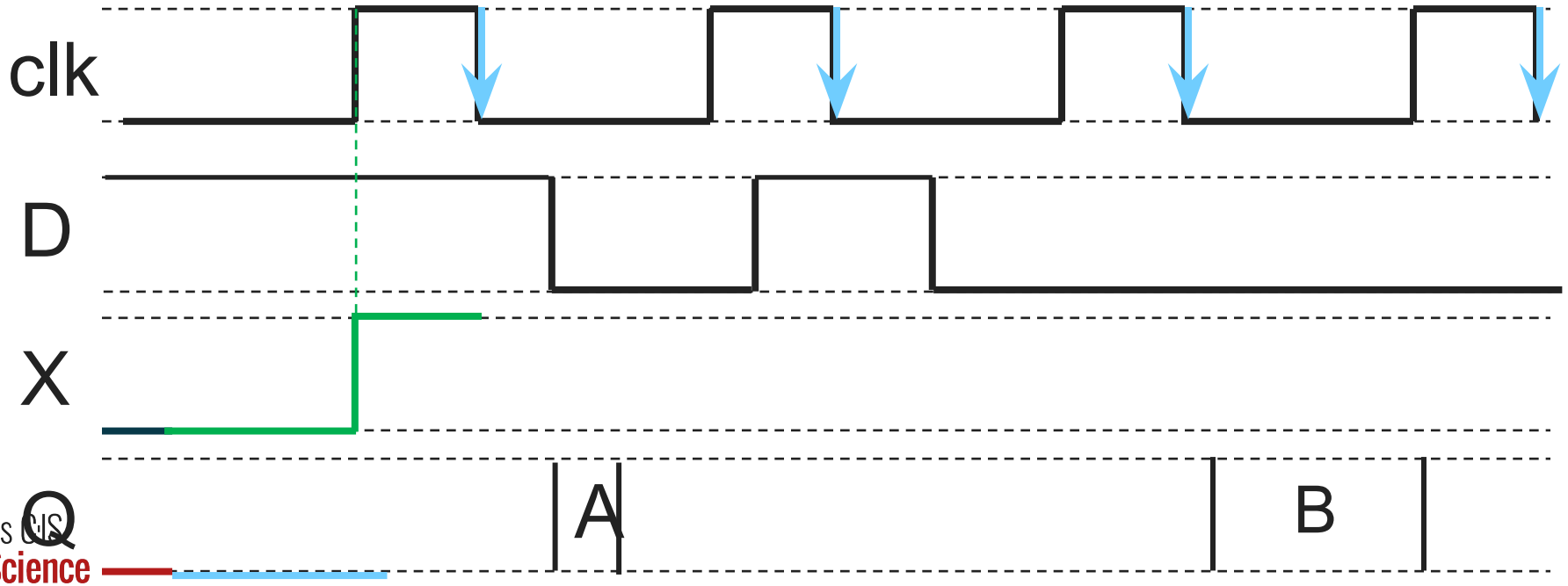
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

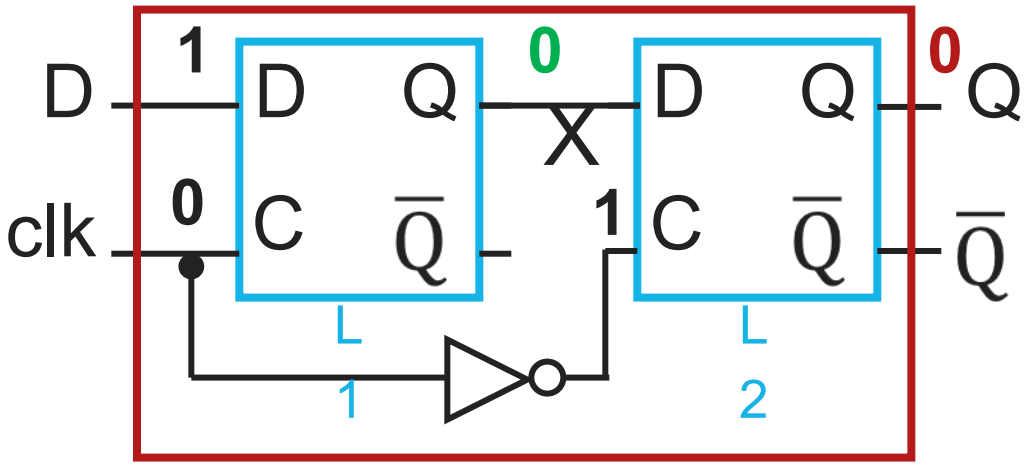




D Flip-Flop

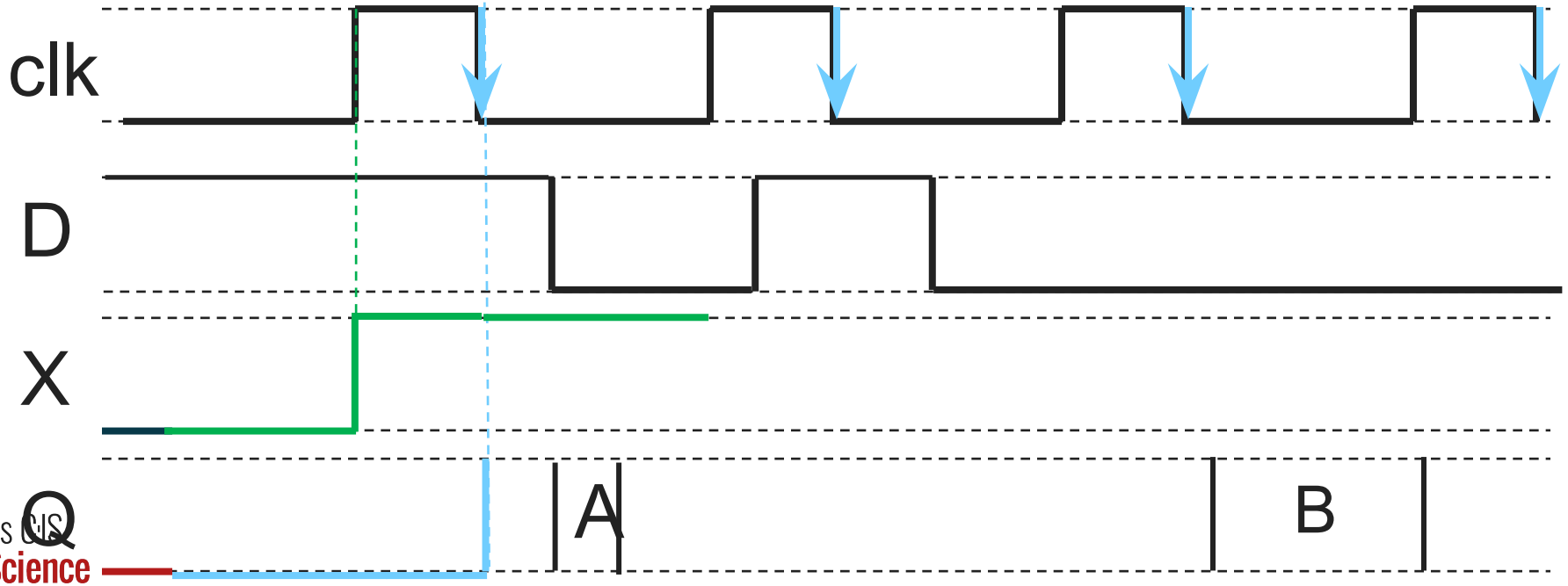
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

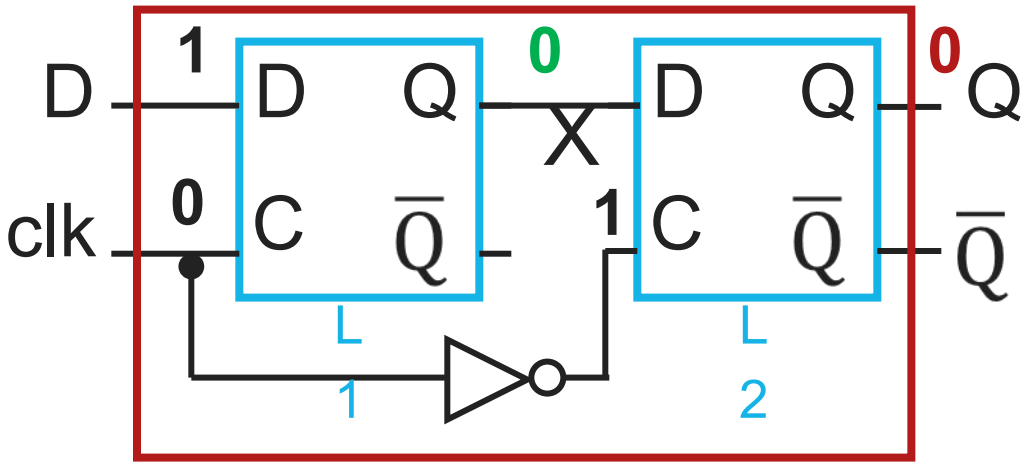




D Flip-Flop

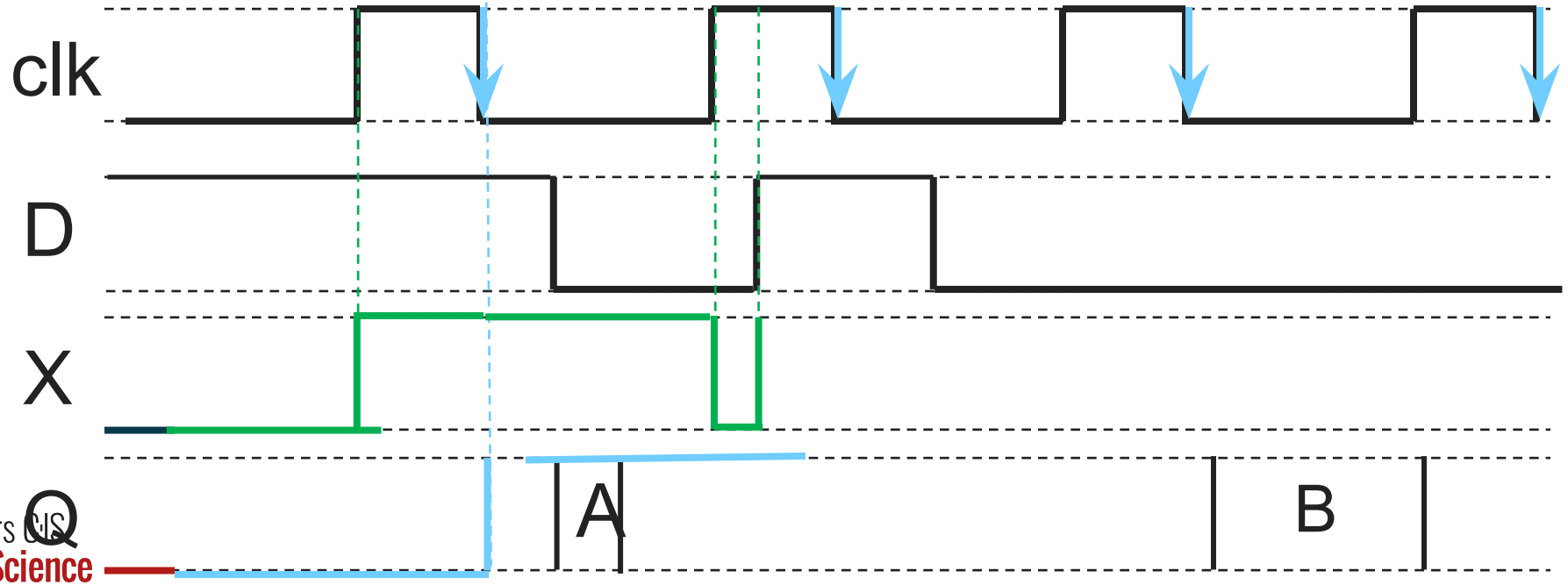
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

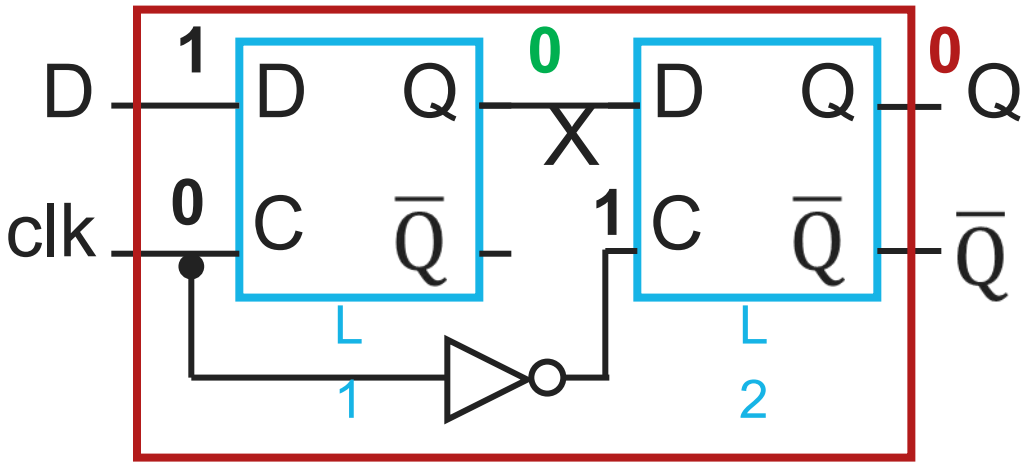




D Flip-Flop

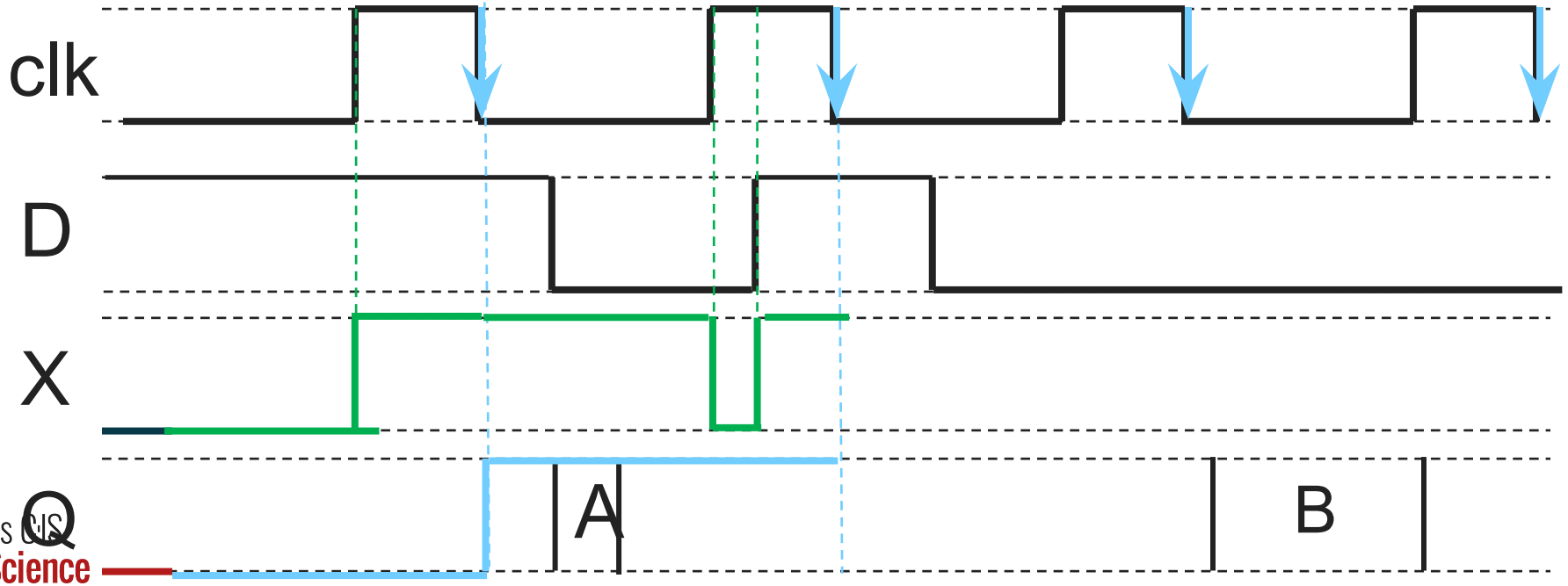
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

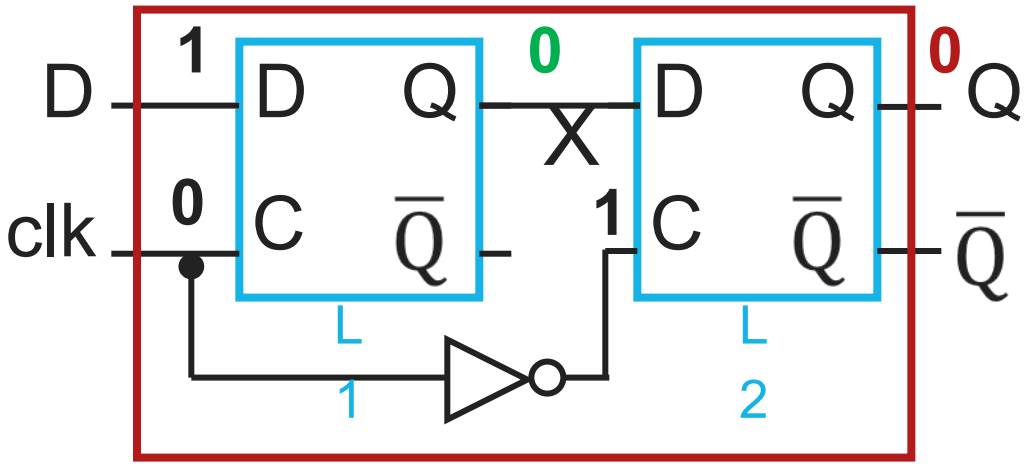




D Flip-Flop

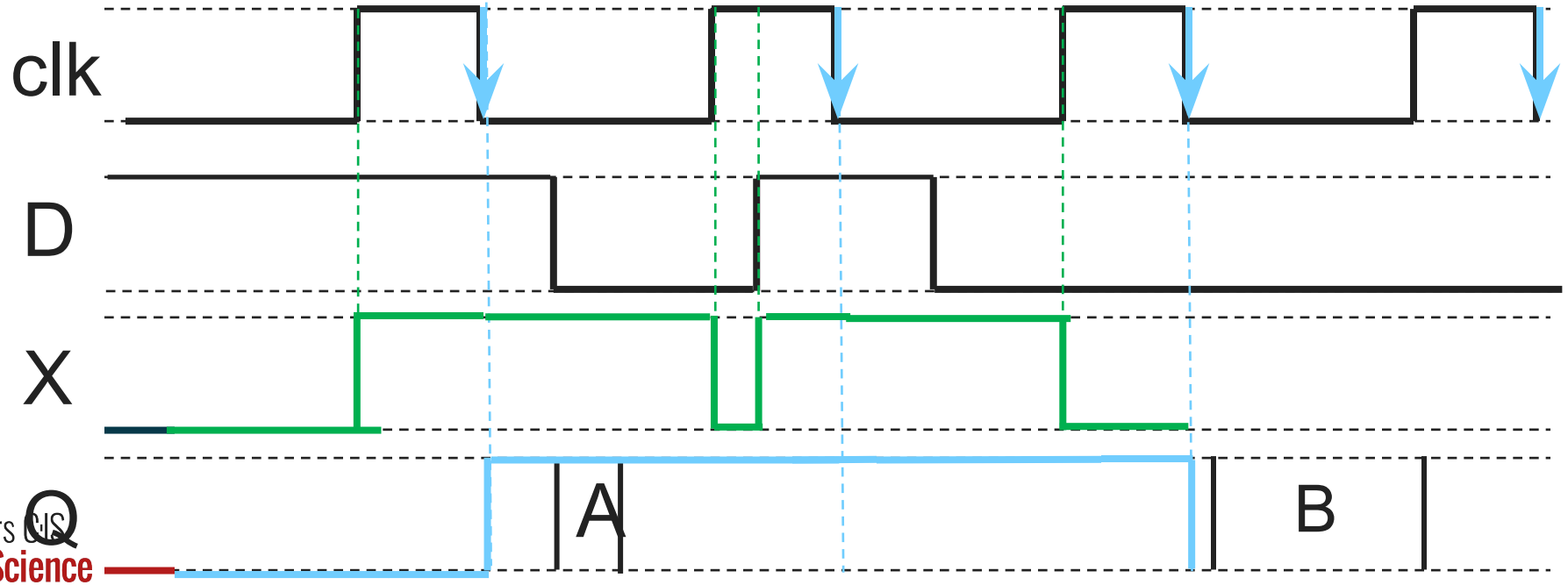
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

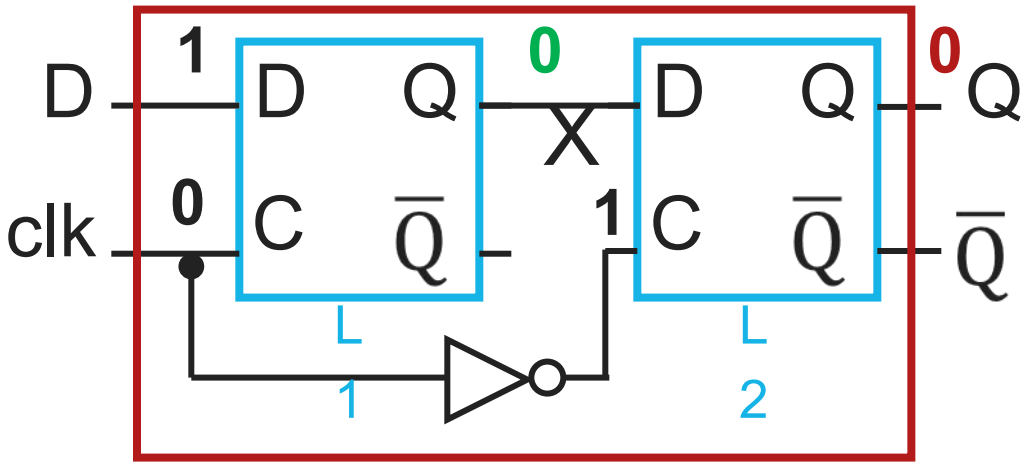




D Flip-Flop

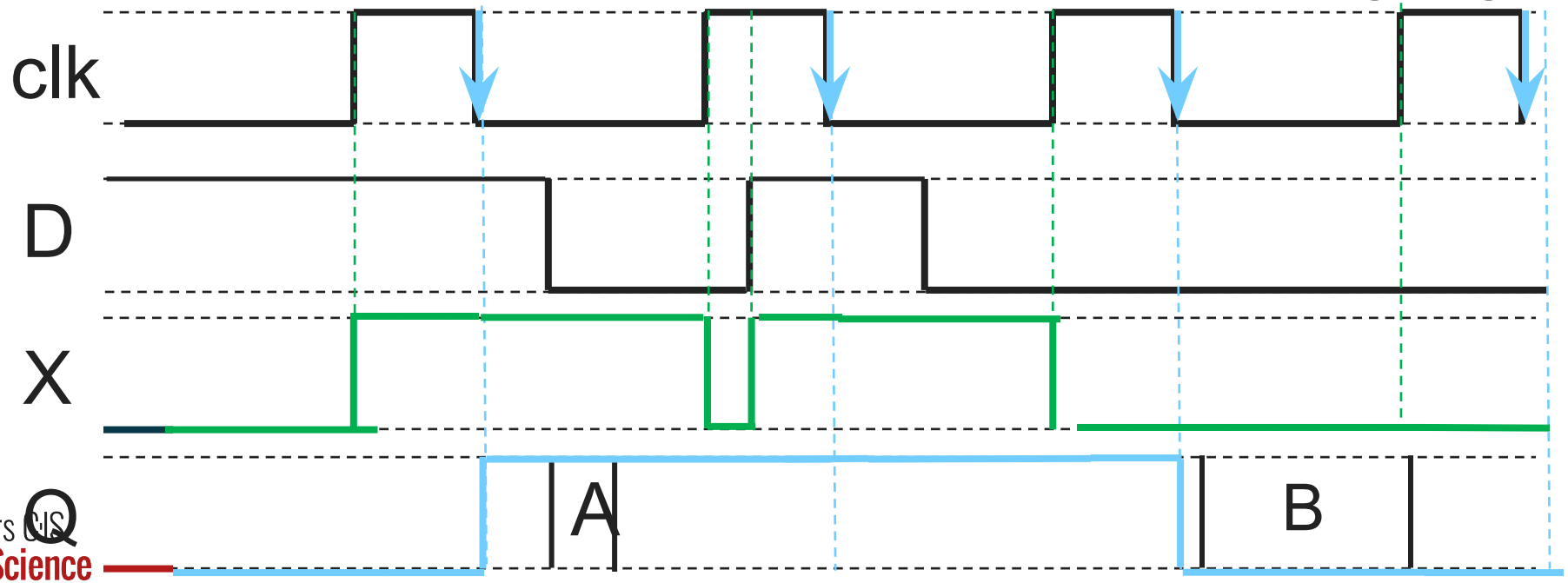
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

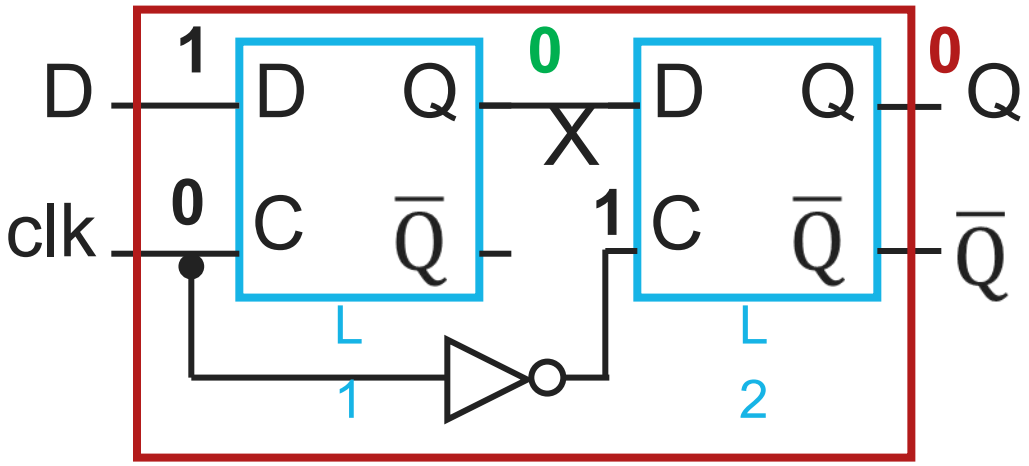




D Flip-Flop

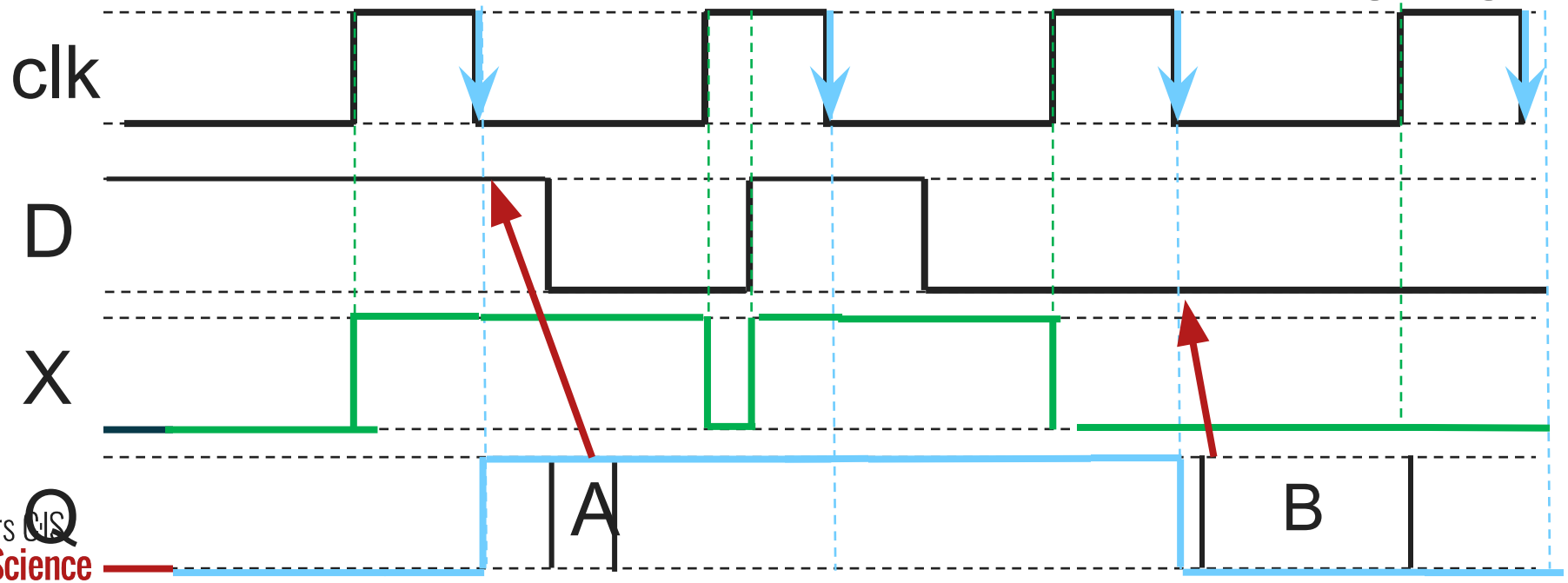
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges



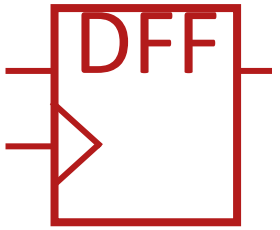


D Flip-Flop

- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

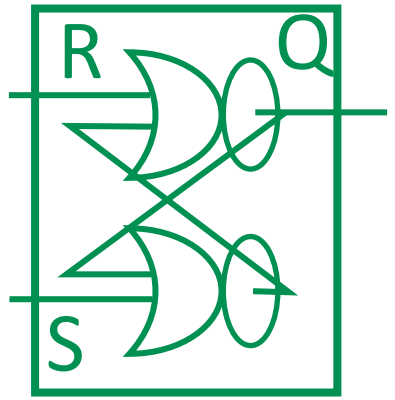


Building a D Flip Flop (DFF)

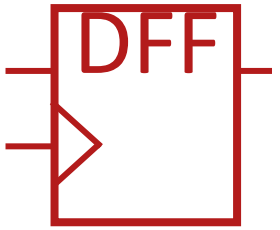


Step 1: Create an SR Latch

Set	Reset	Q
0	0	Q
0	1	0
1	0	1
1	1	?



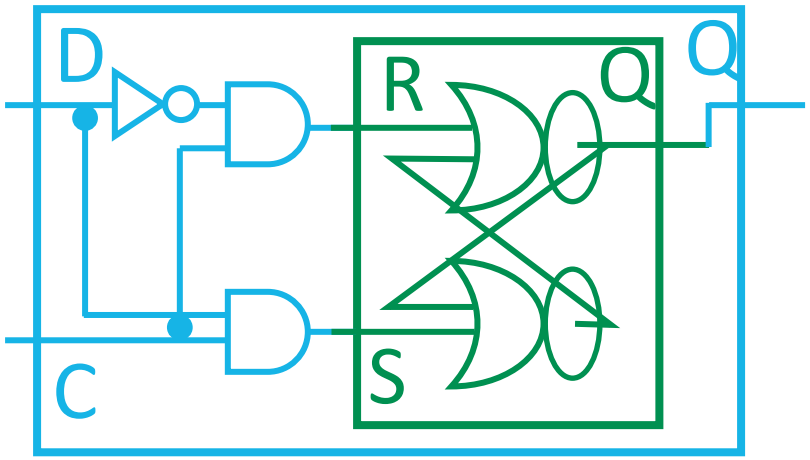
Building a D Flip Flop (DFF)



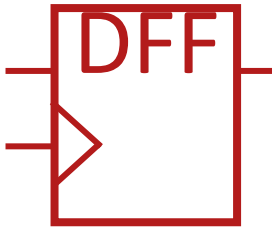
Step 1: Create an SR Latch

Step 2: Create a D Latch

Clk	Data	Q
0	0	Q
0	1	Q
1	0	0
1	1	1



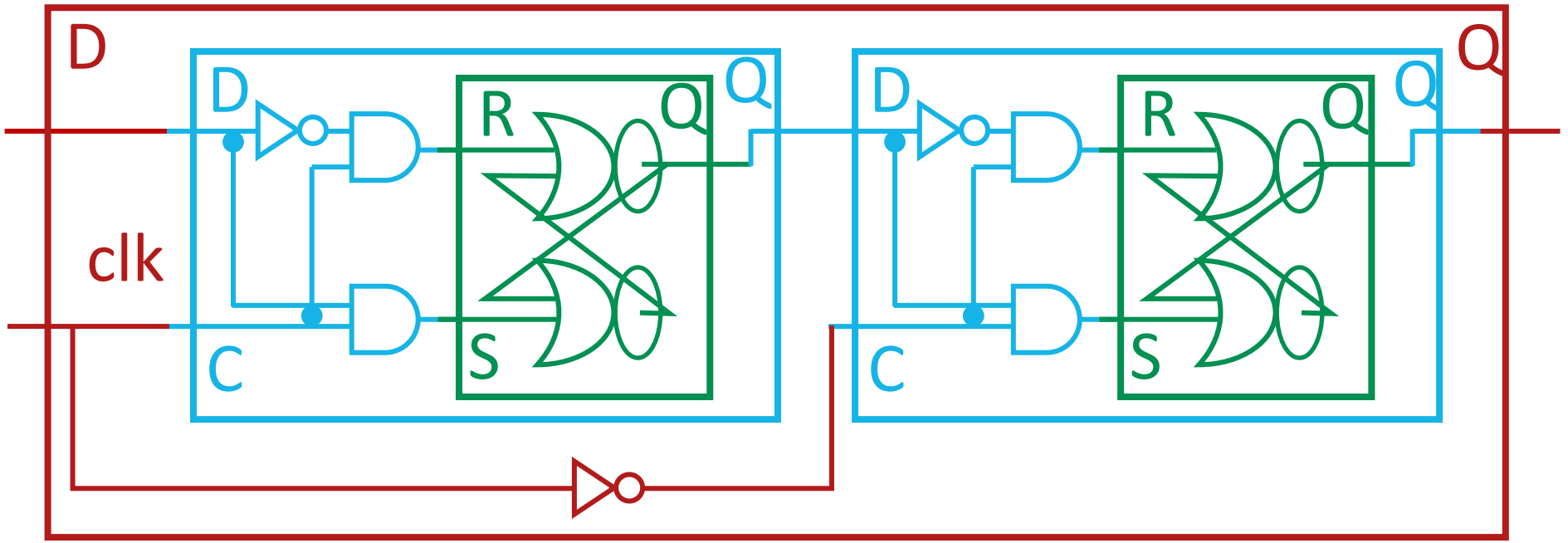
Building a D Flip Flop (DFF)



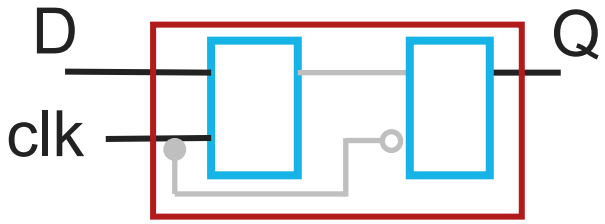
Step 1: Create an SR Latch

Step 2: Create a D Latch

Step 3: Duplicate the D Latch, chain together



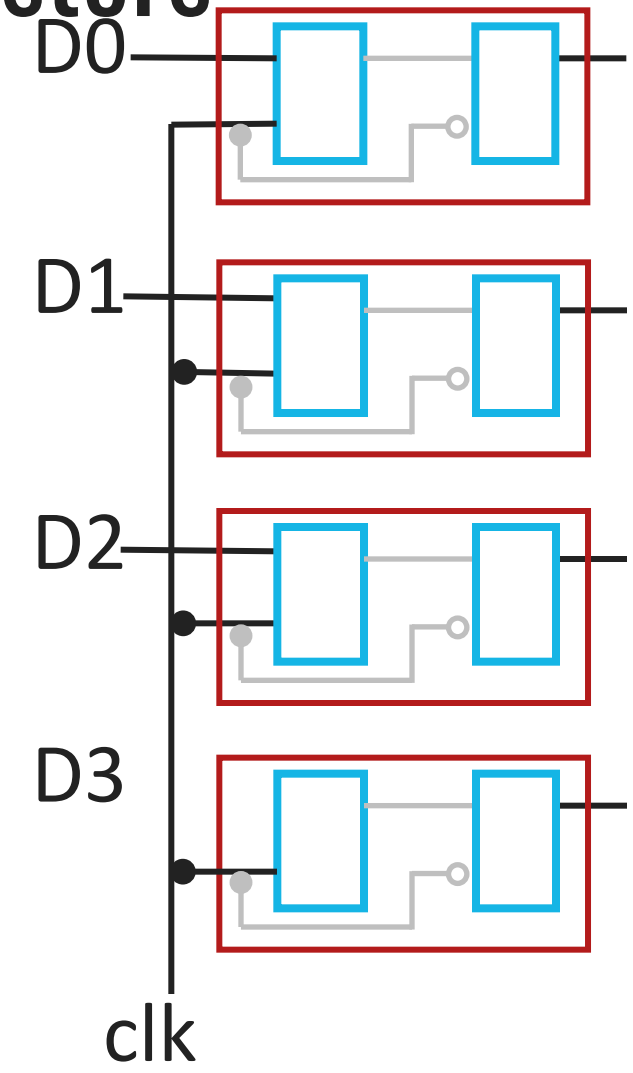
Registers



Register

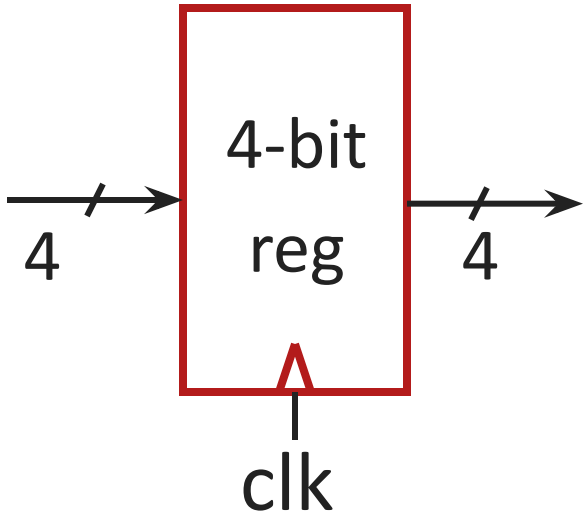
- D flip-flops in parallel

Registers

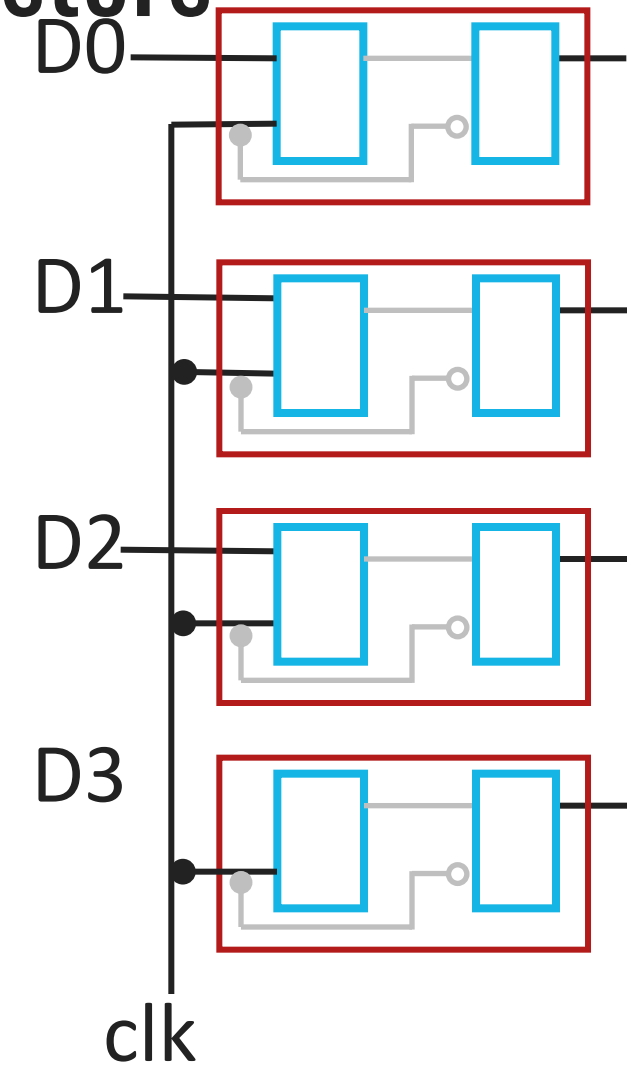


Register

- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …

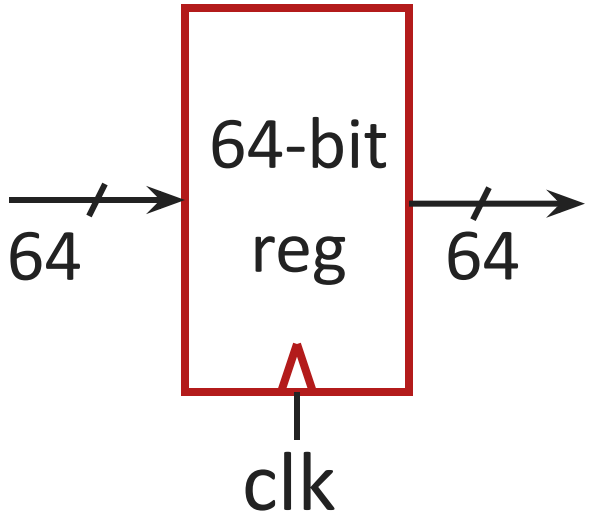


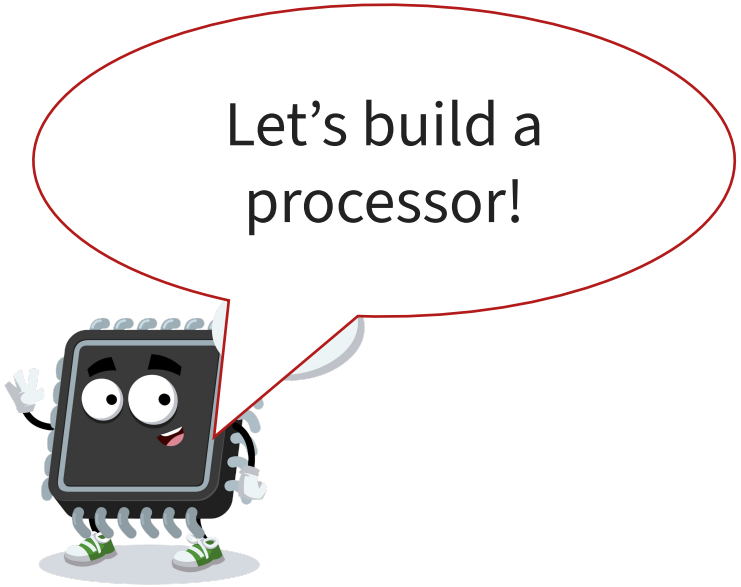
Registers



Register

- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …

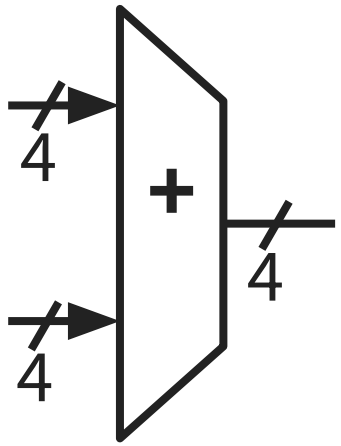


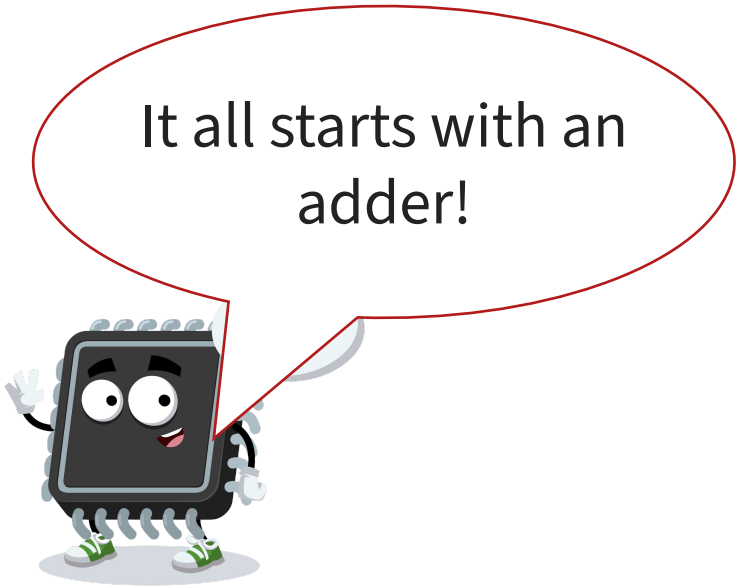


Femto Proc

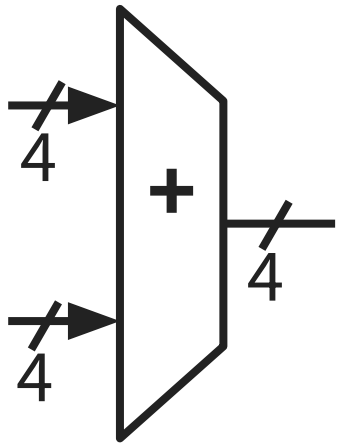
CS 3410: Computer System Organization and Programming

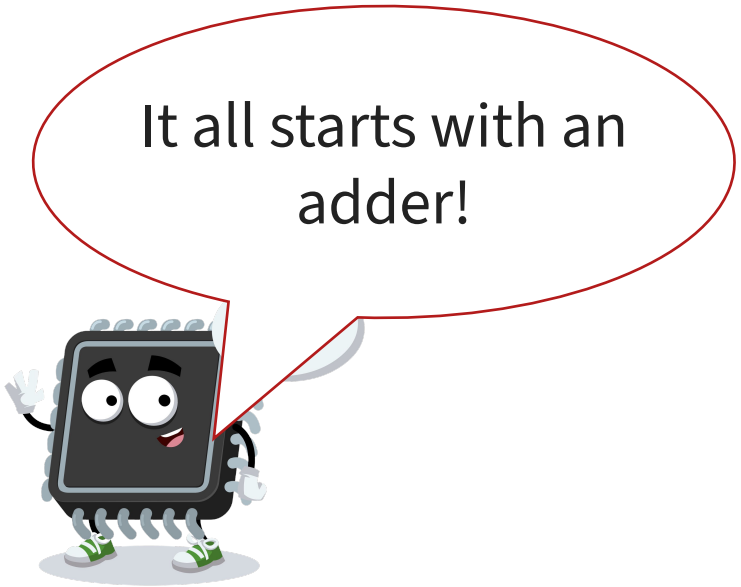
Fall 2025


4
+
4
4

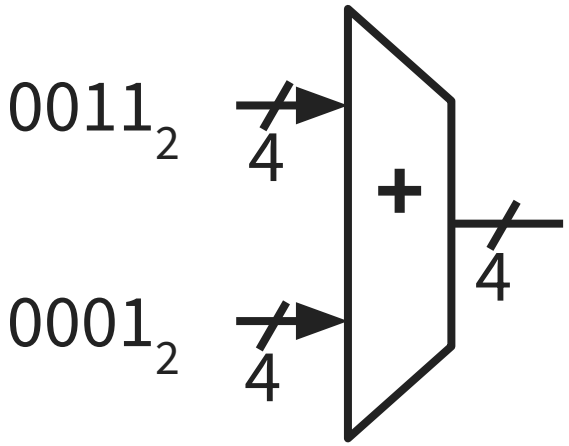

It all starts with an adder!

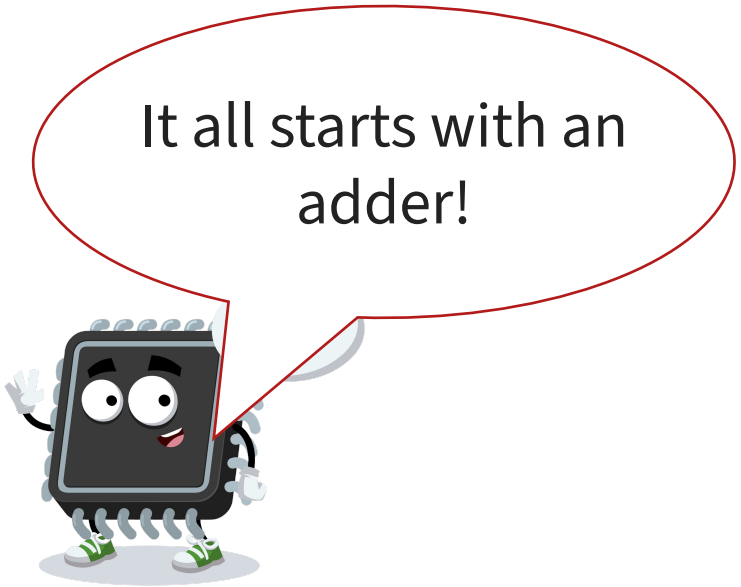
```
add 3, 1
```



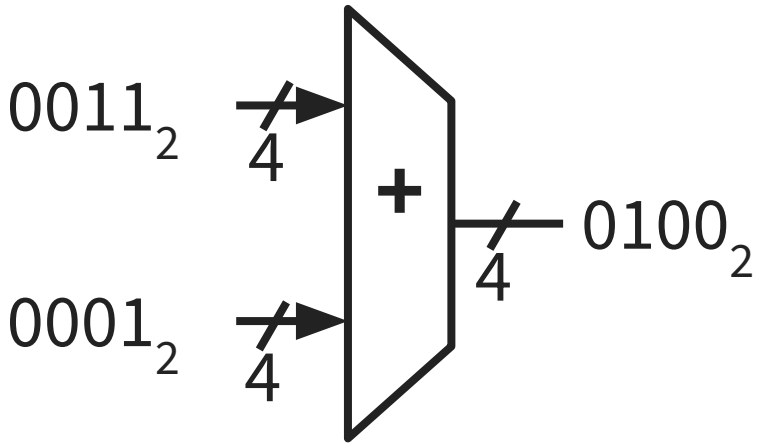


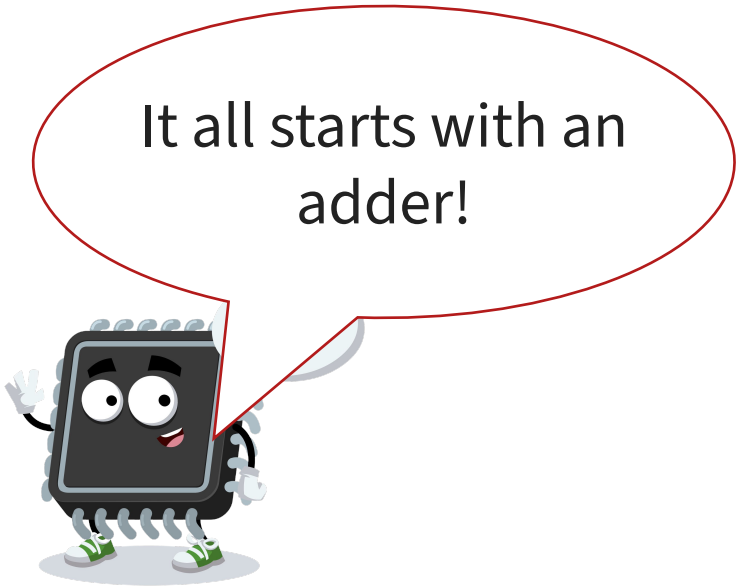
```
add 3, 1
```



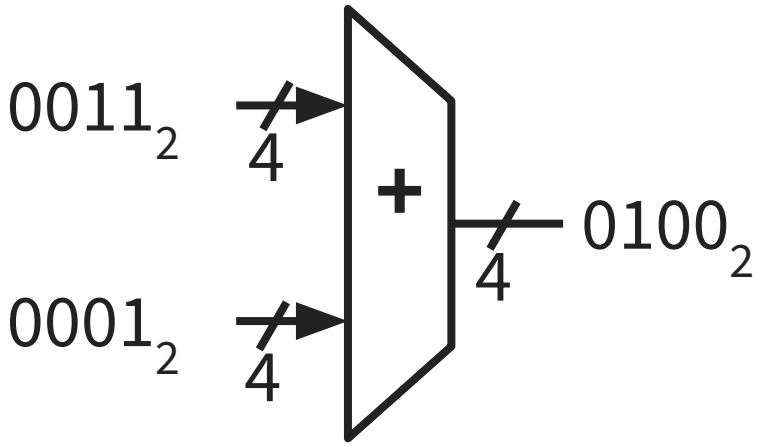


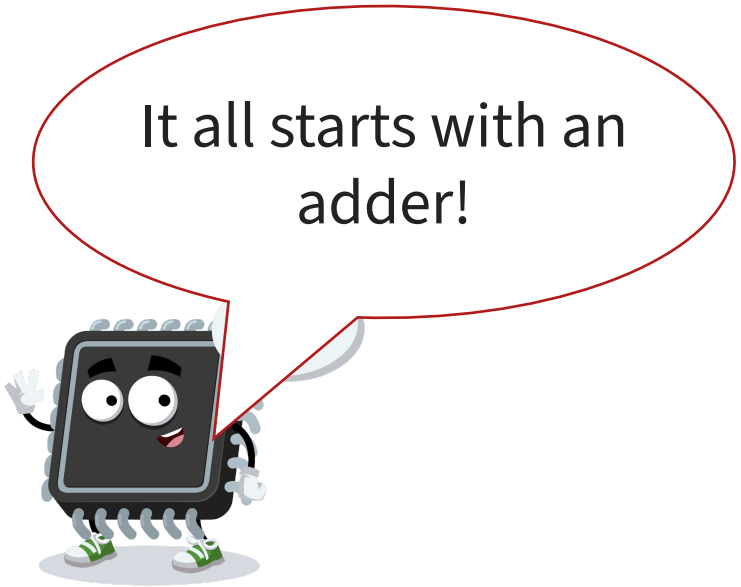
```
add 3, 1
```



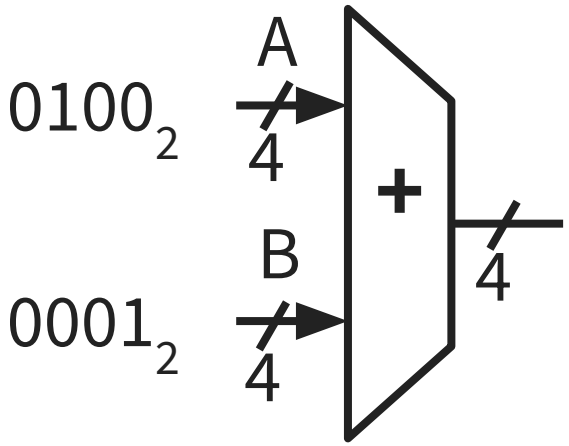


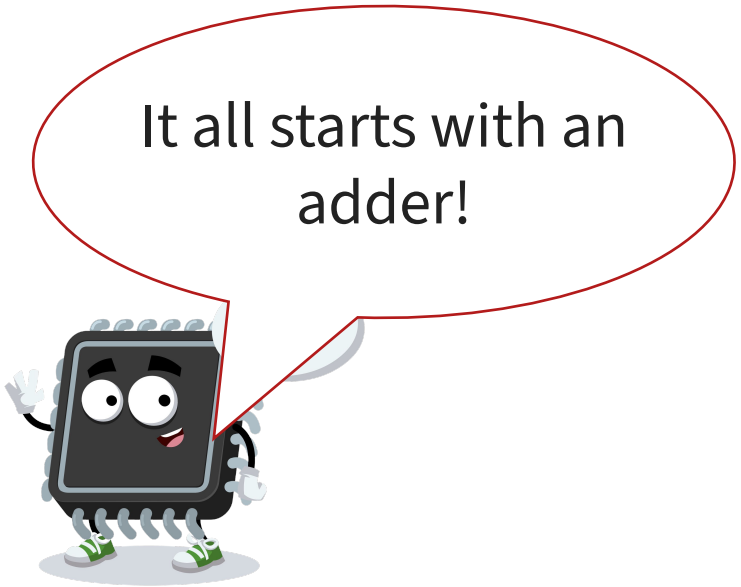
```
add 3, 1
add 4, 1
```



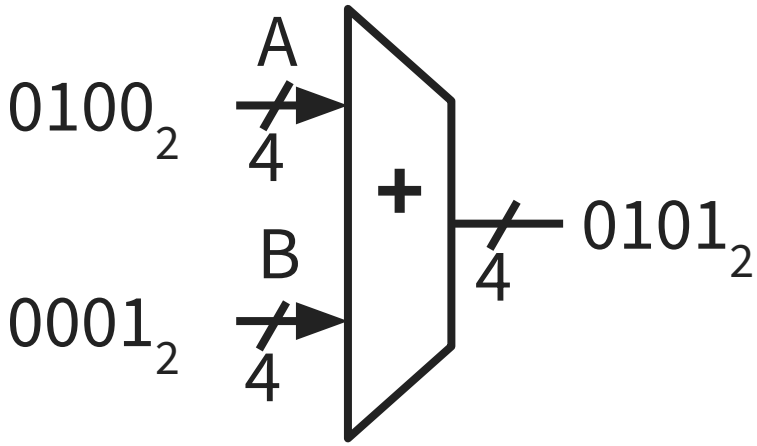


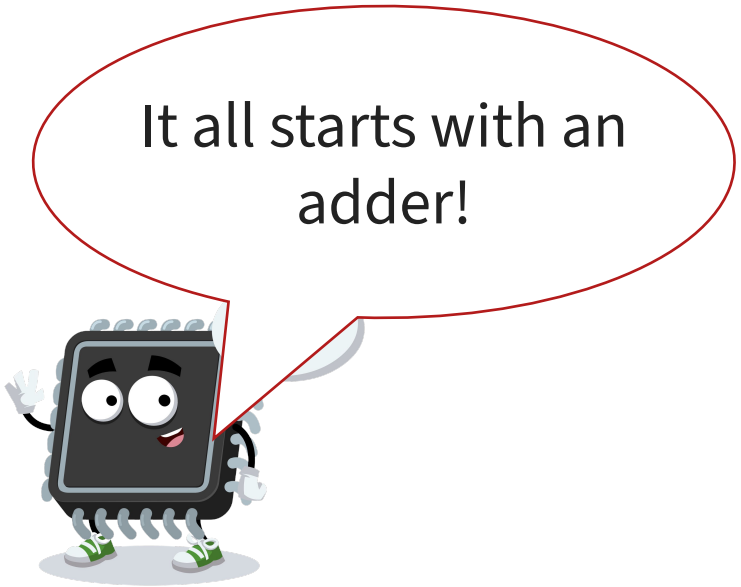
```
add 3, 1
add 4, 1
```



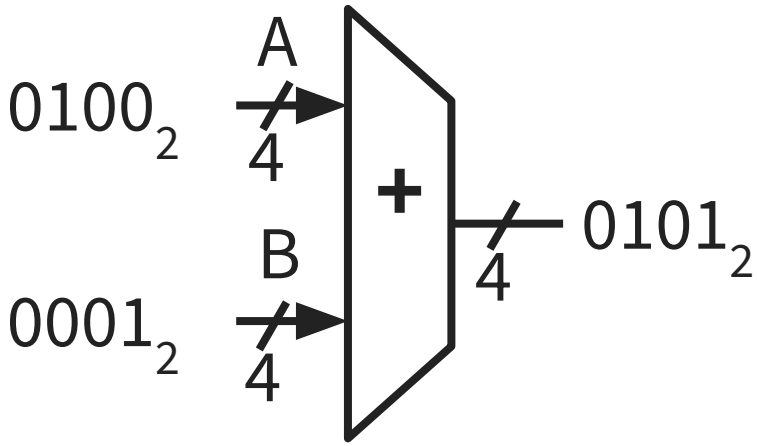


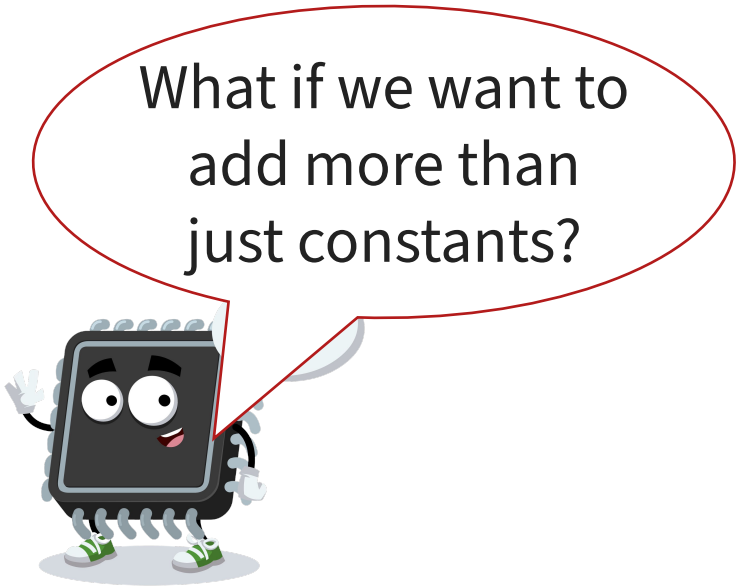
```
add 3, 1
add 4, 1
```



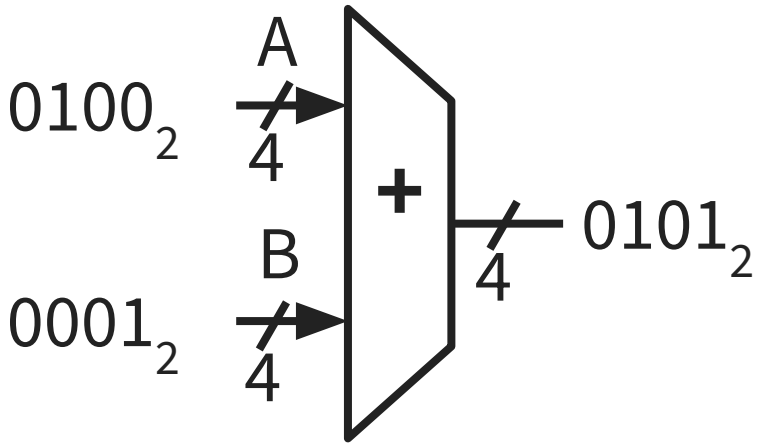


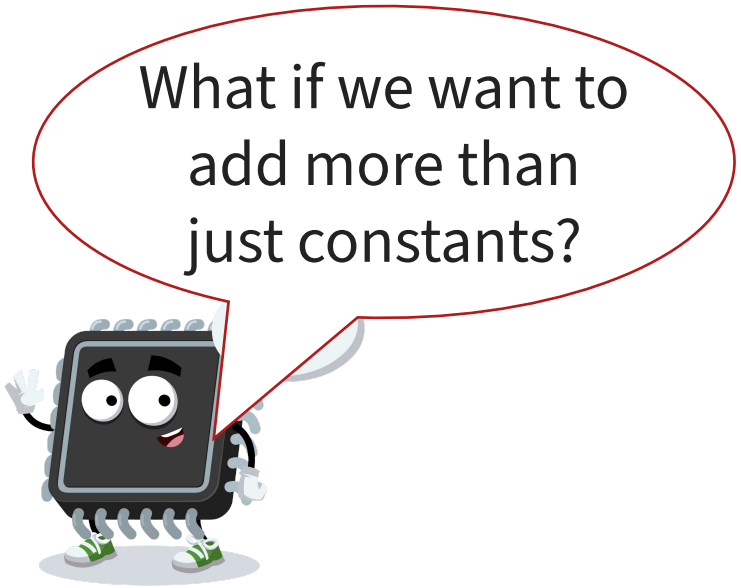
```
add 3, 1
add 4, 1
```


A
0100_2
4
+
B
0001_2
4
4
0101_2

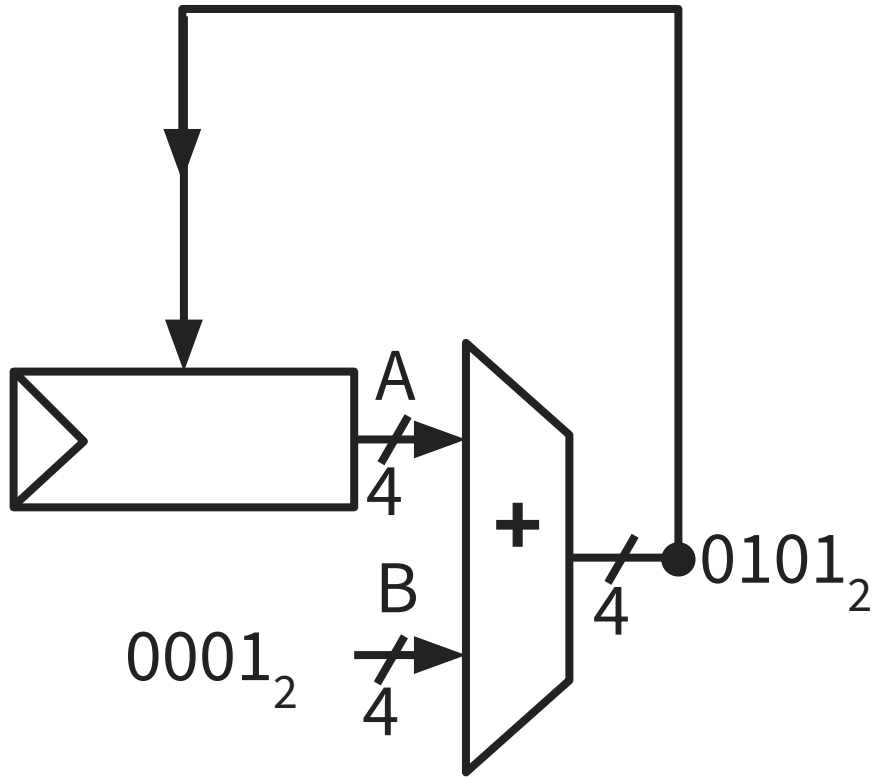

What if we want to add more than just constants?

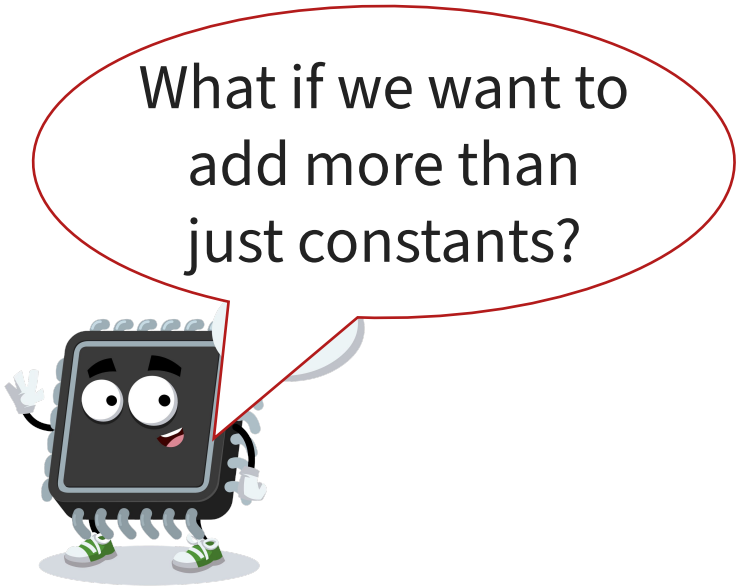
```
add 3, 1
add 4, 1
add “previous result”, 1
```



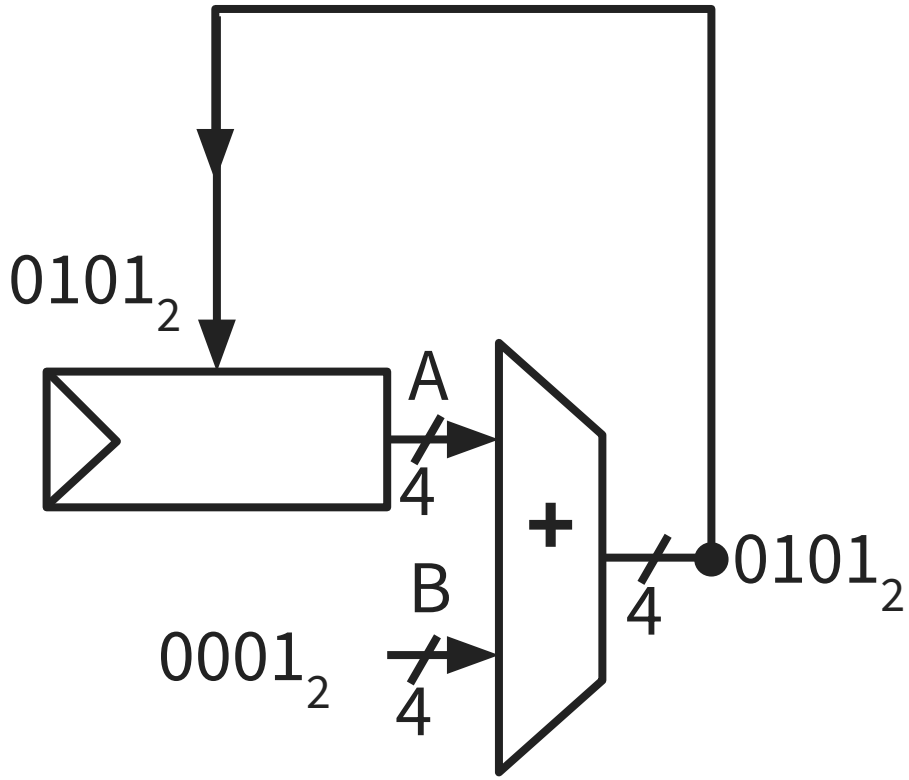


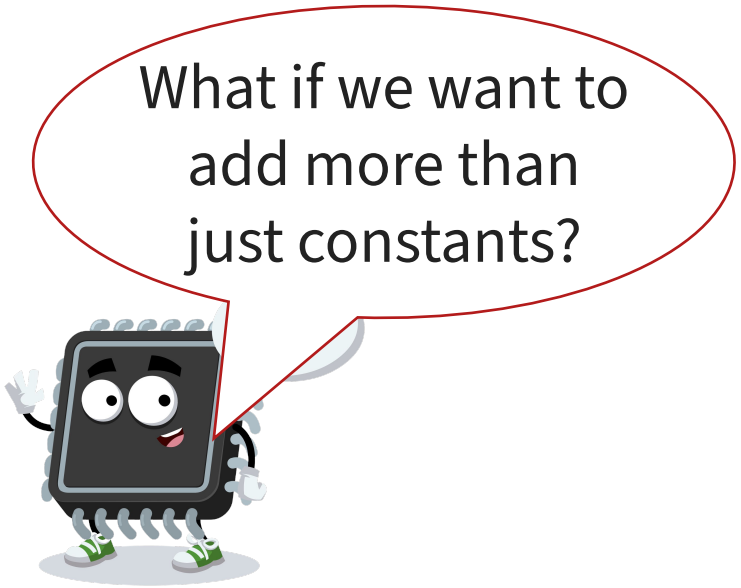
```
add 3, 1
add 4, 1
add “previous result”, 1
```



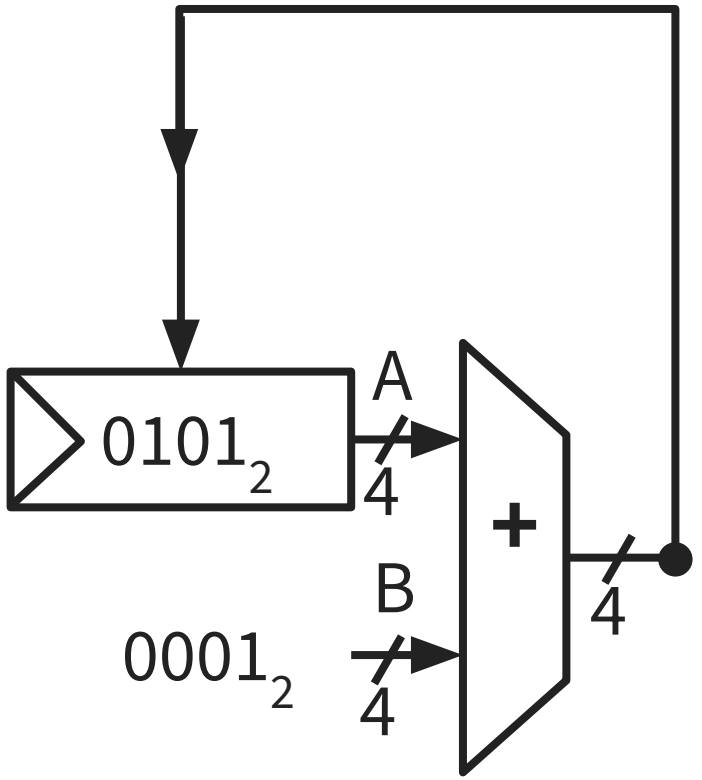


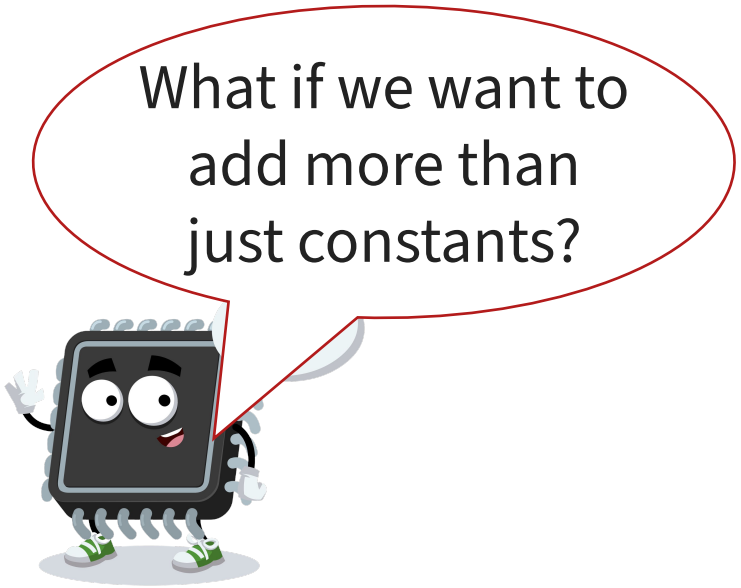
```
add 3, 1
add 4, 1
add “previous result”, 1
```



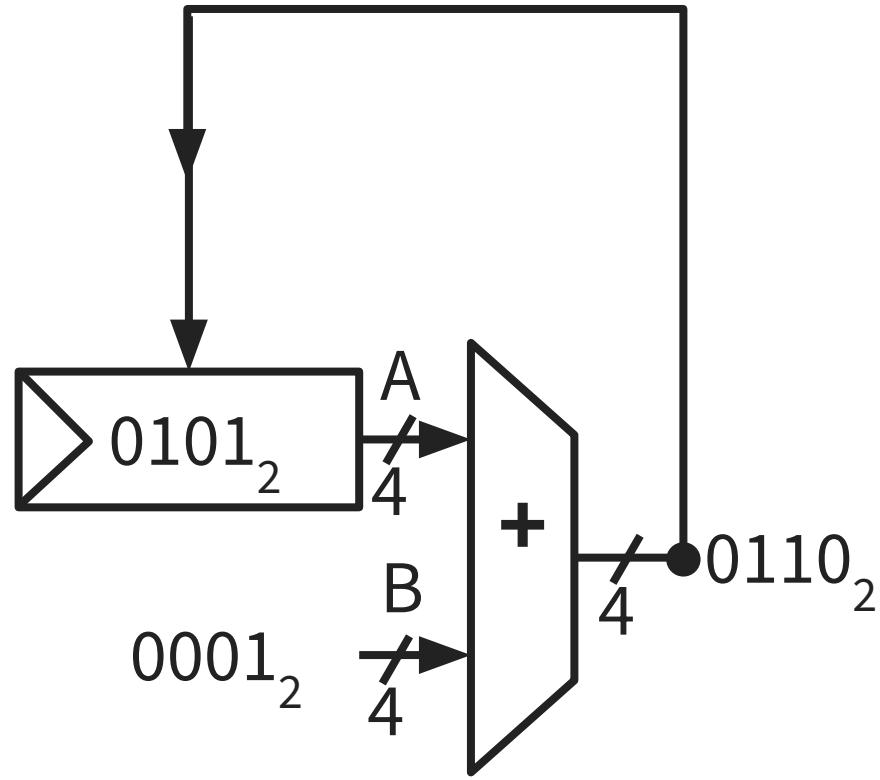


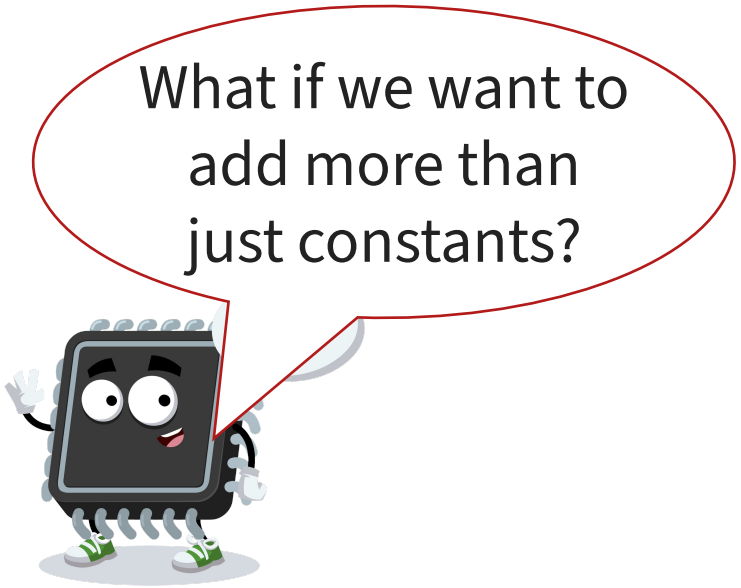
```
add 3, 1
add 4, 1
add “previous result”, 1
```



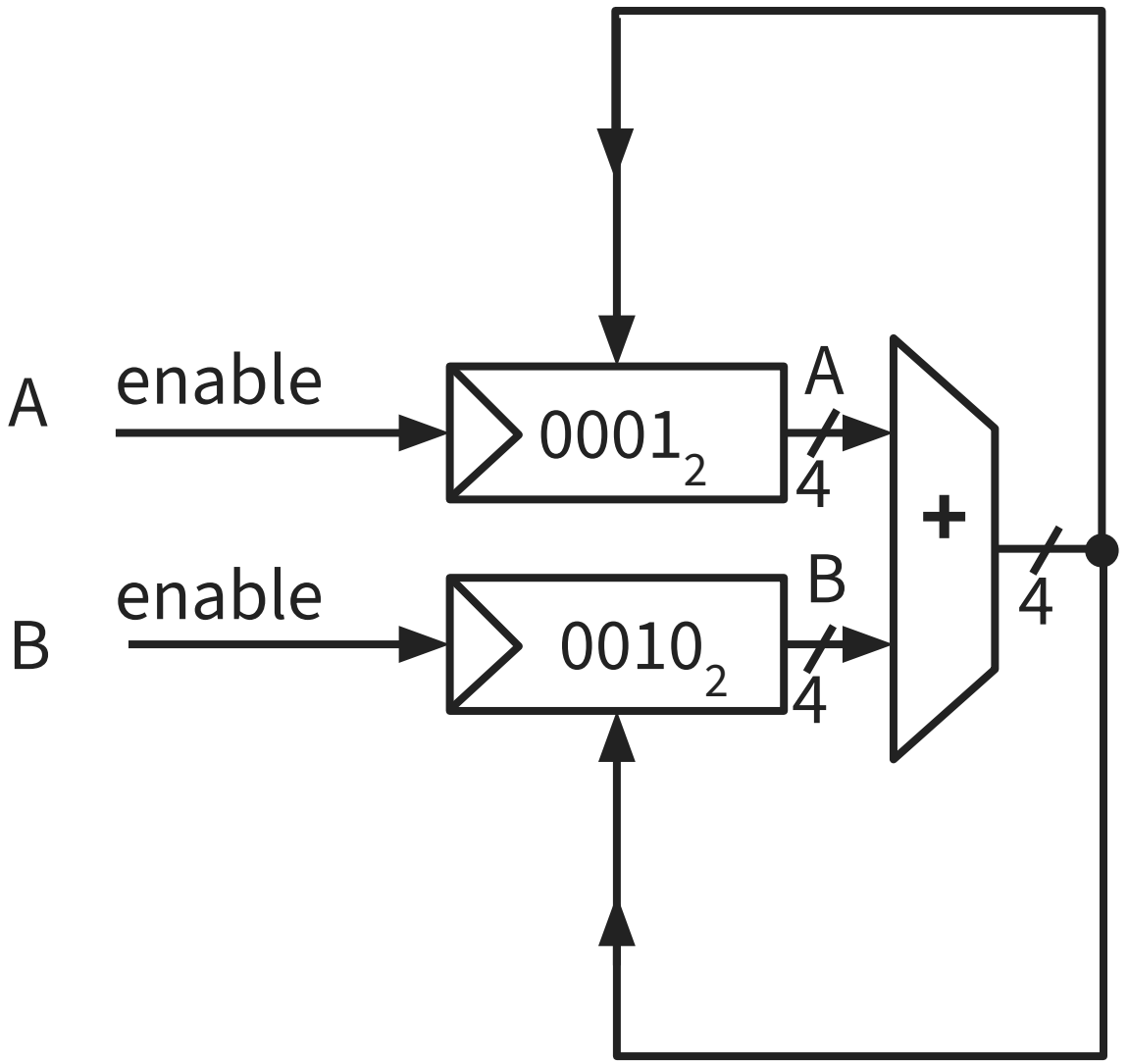


```
add 3, 1
add 4, 1
add “previous result”, 1
```



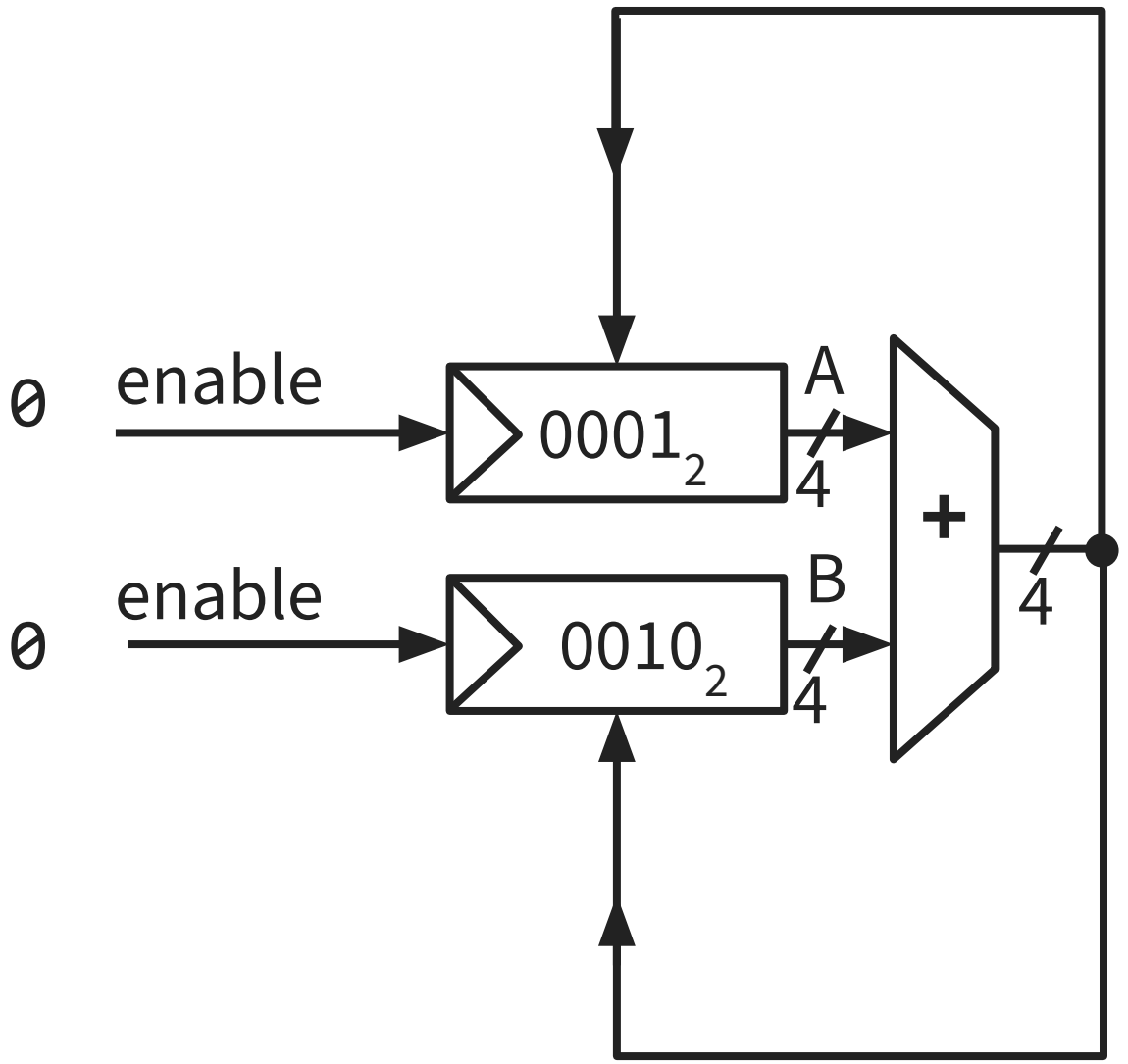


Four “Instructions”:



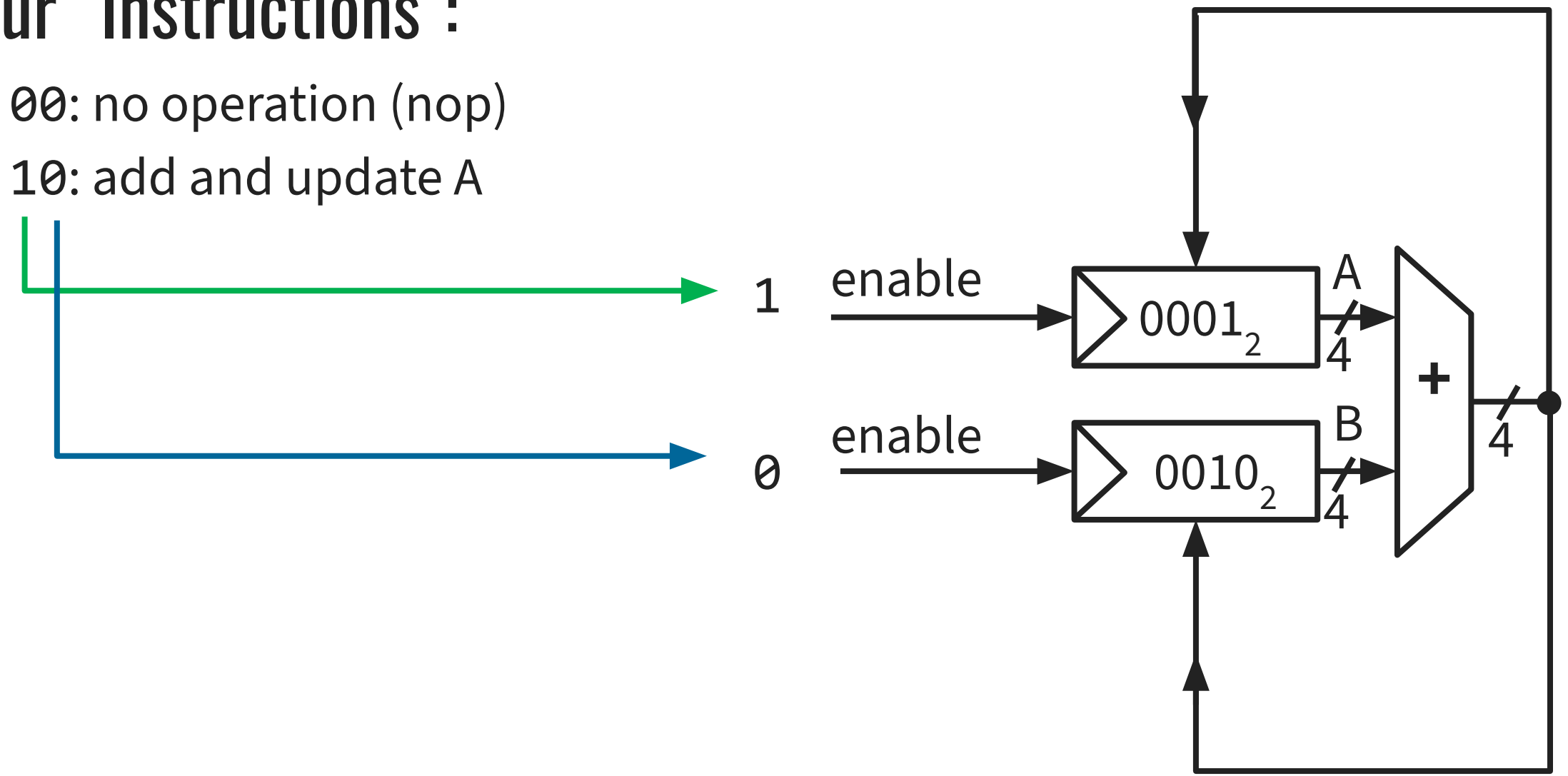
Four “Instructions”:

- `00`: no operation (nop)



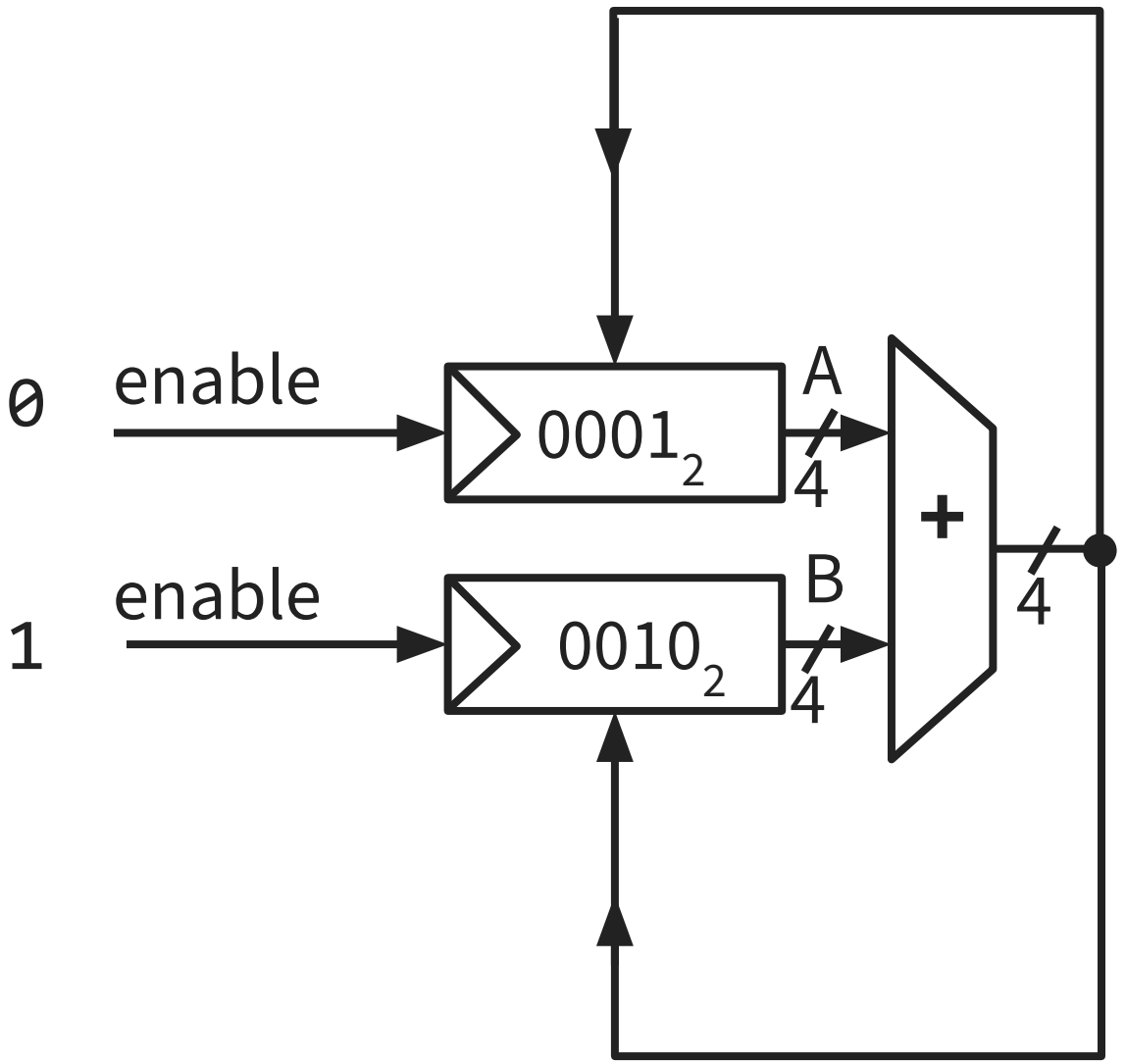
Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A



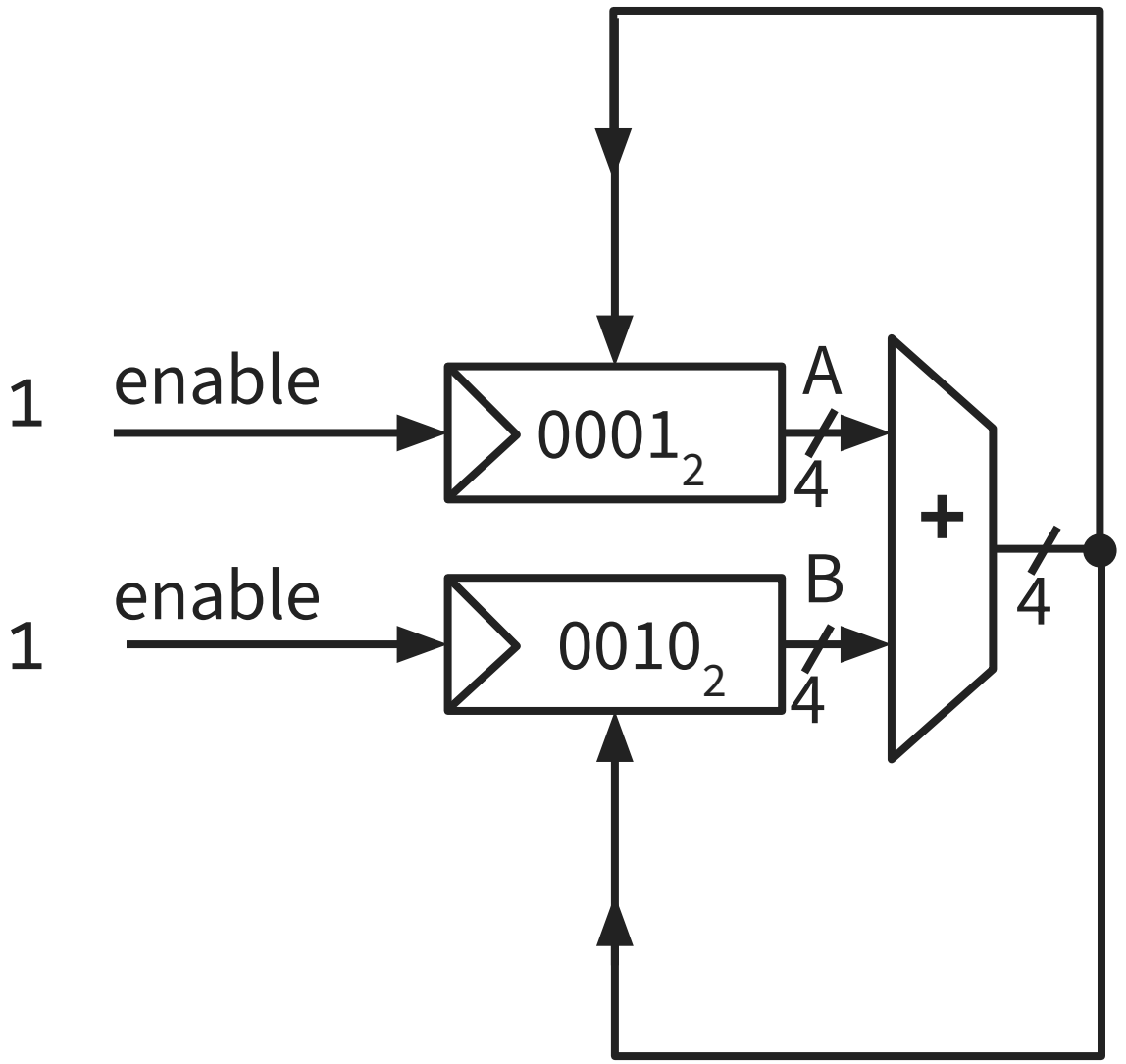
Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B



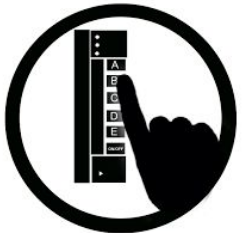
Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B



Four "Instructions":

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

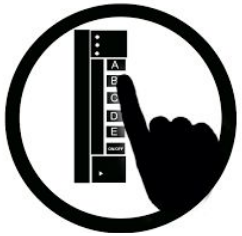


Shortest program to get from A=1, B = 2 to result = 5?

PollEv.com/cs3410

Four "Instructions":

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B



Shortest program to get from A=1, B = 2 to result = 5?

Shortest program to get from A=1, B = 2 to result = 6?

PollEv.com/cs3410

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
	1	2	3

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
00	1	2	3

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
00	1	2	3
	1	2	3

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
10	1	2	3

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
10	1	2	3
	3	2	5

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
10	1	2	3
10	3	2	5
	5	2	7

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
10	1	2	3
01	3	2	5
	3	5	8

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
01	1	2	3

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
01	1	2	3
	1	3	4

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

Instruction	A	B	result
01	1	2	3
01	1	3	4
	1	4	5

Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

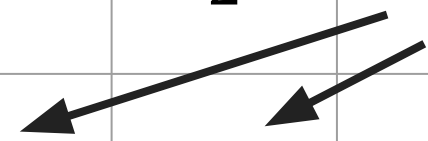
Shortest program to get from A=1, B = 2 to result = 6?

Instruction	A	B	result
01	1	2	3
01	1	3	4
01	1	4	5
	1	5	**6**

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

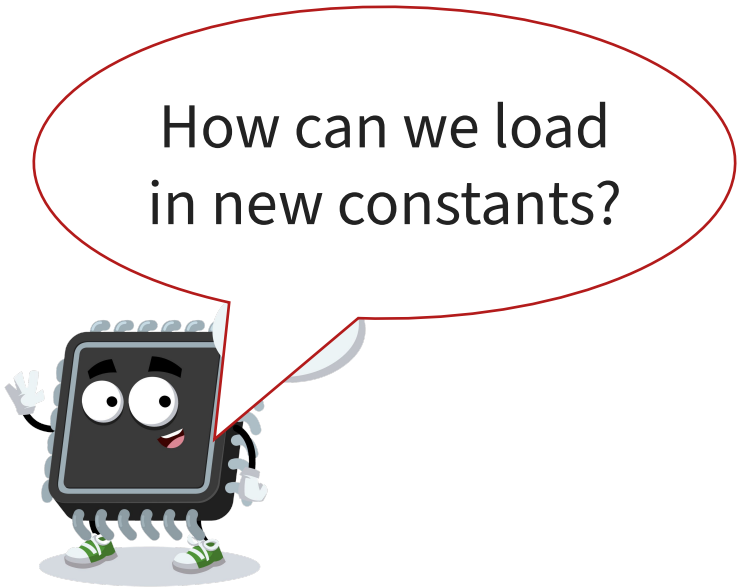
Instruction	A	B	result
11	1	2	3
	3	3	6

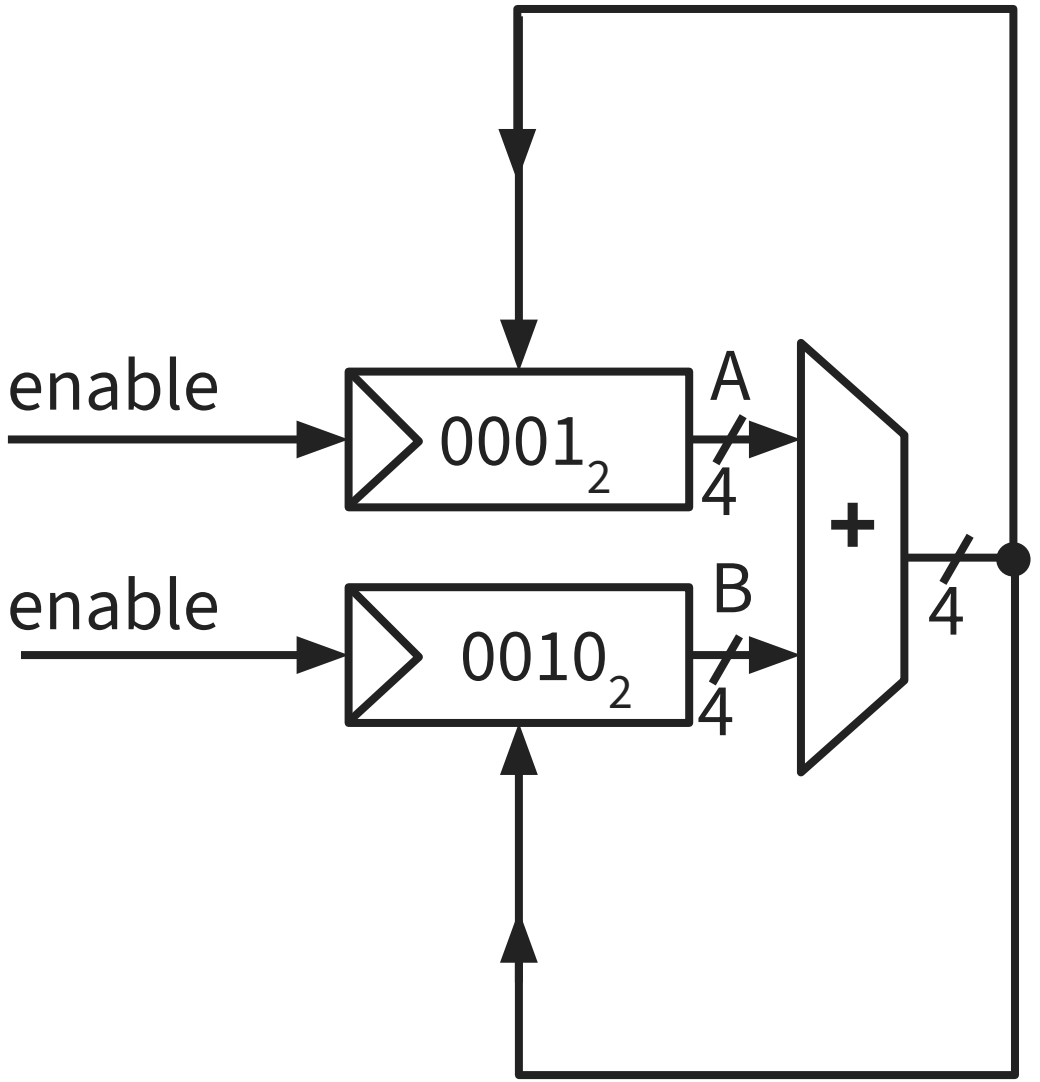


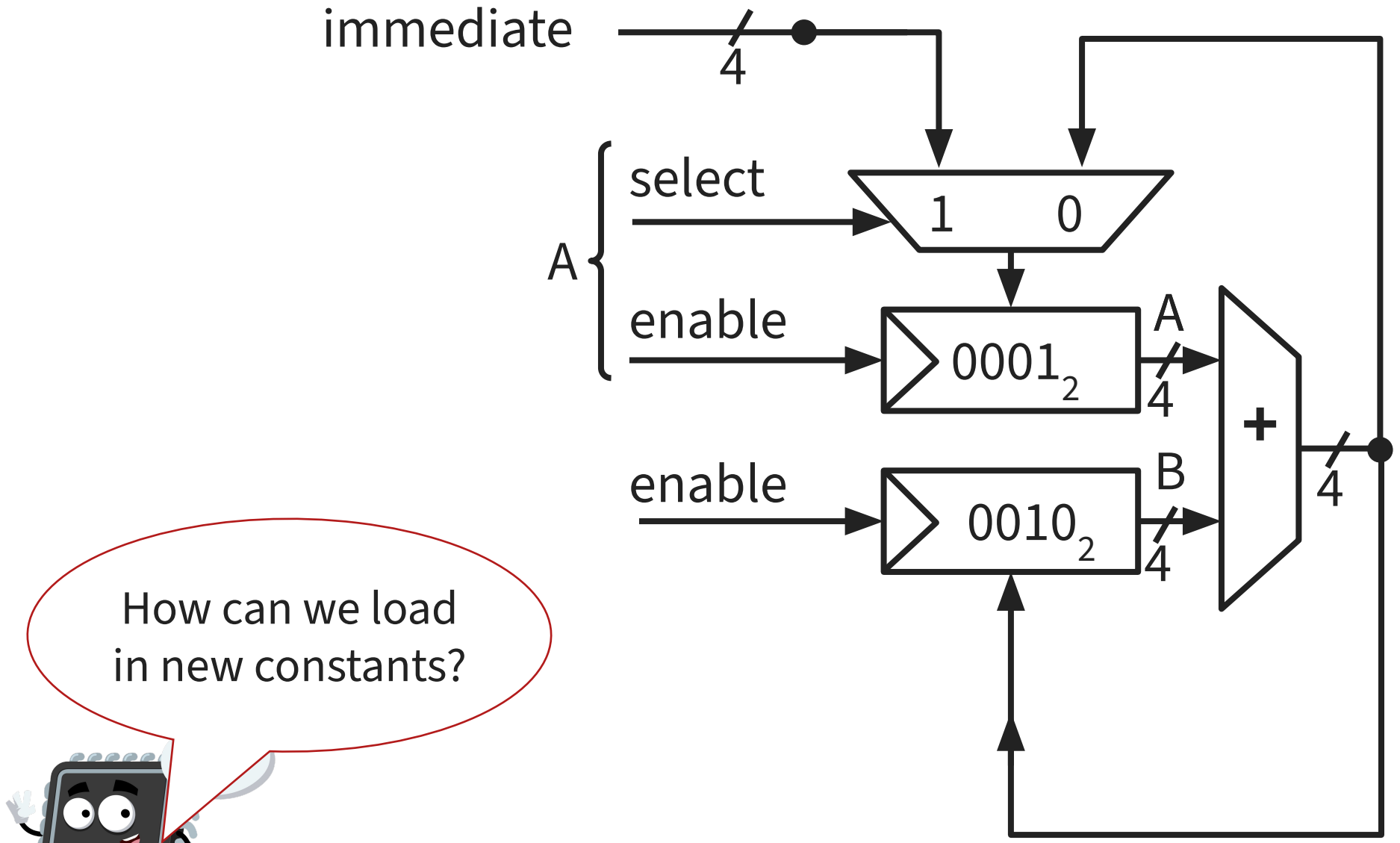
Shortest program to get from A=1, B = 2 to result = 6?

Four “Instructions”:

- `00`: no operation (nop)
- `10`: add and update A
- `01`: add and update B
- `11`: add and update A and B

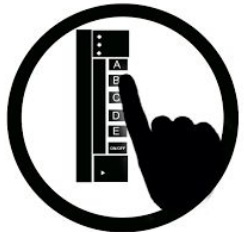


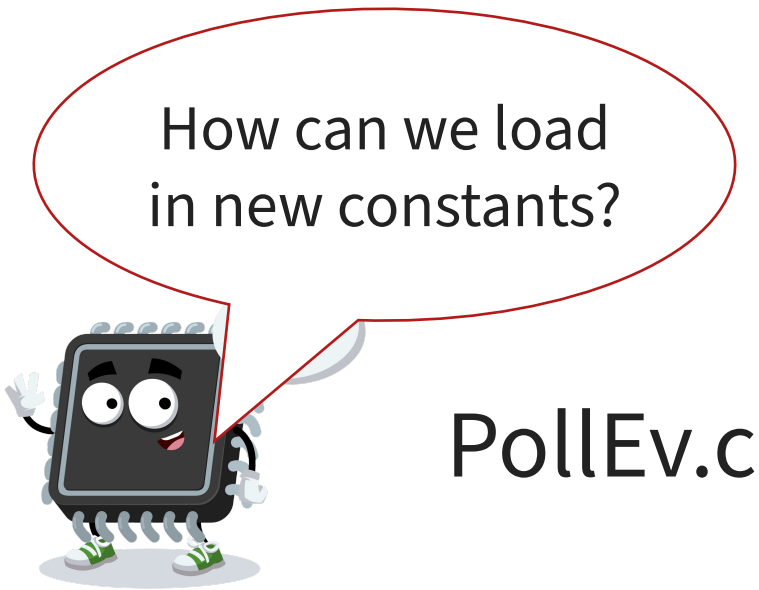


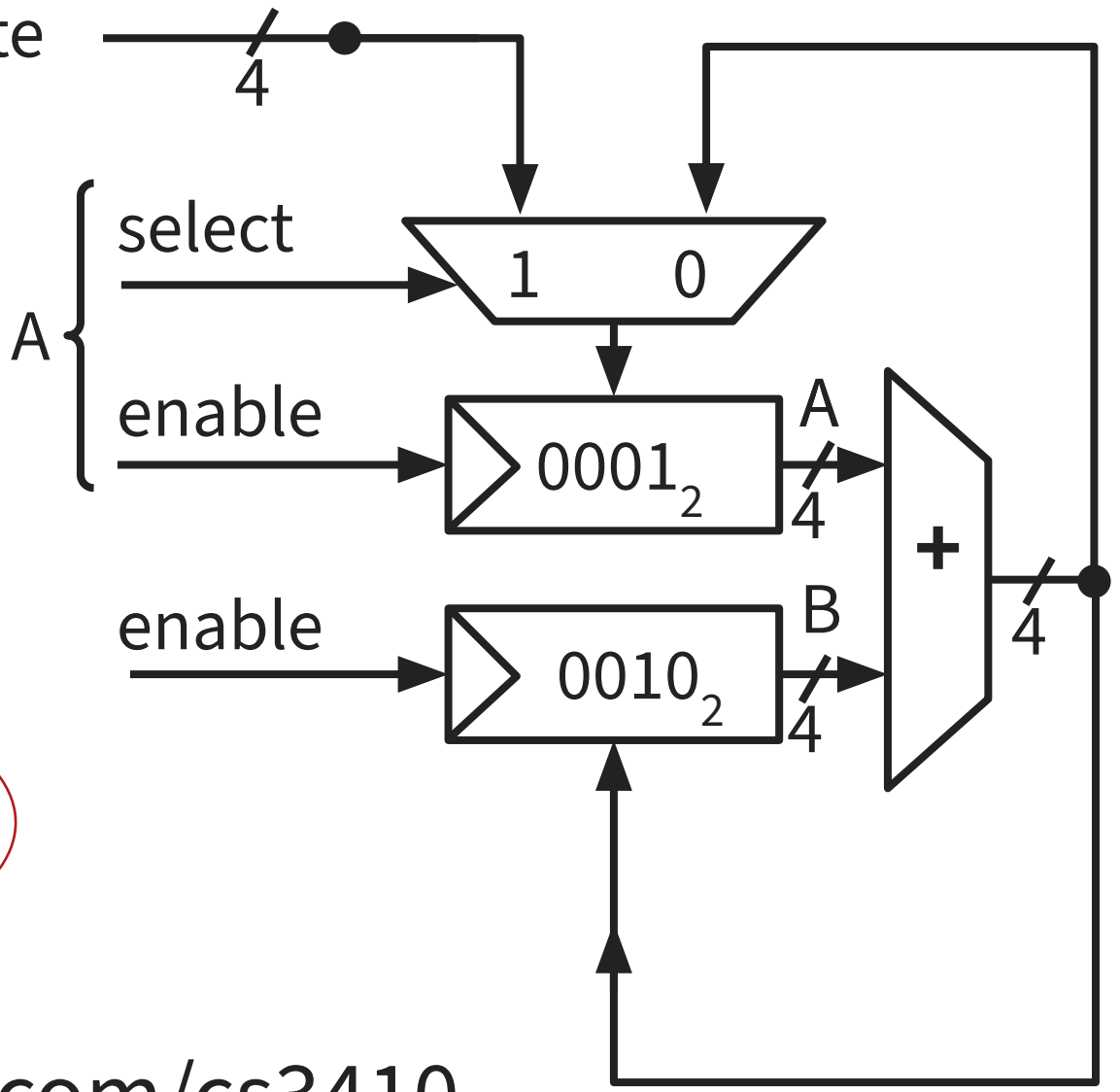

immediate
4
select
A
enable
1
0
0001₂
A
4
+
4
enable
0010₂
B
4

How can we load
in new constants?

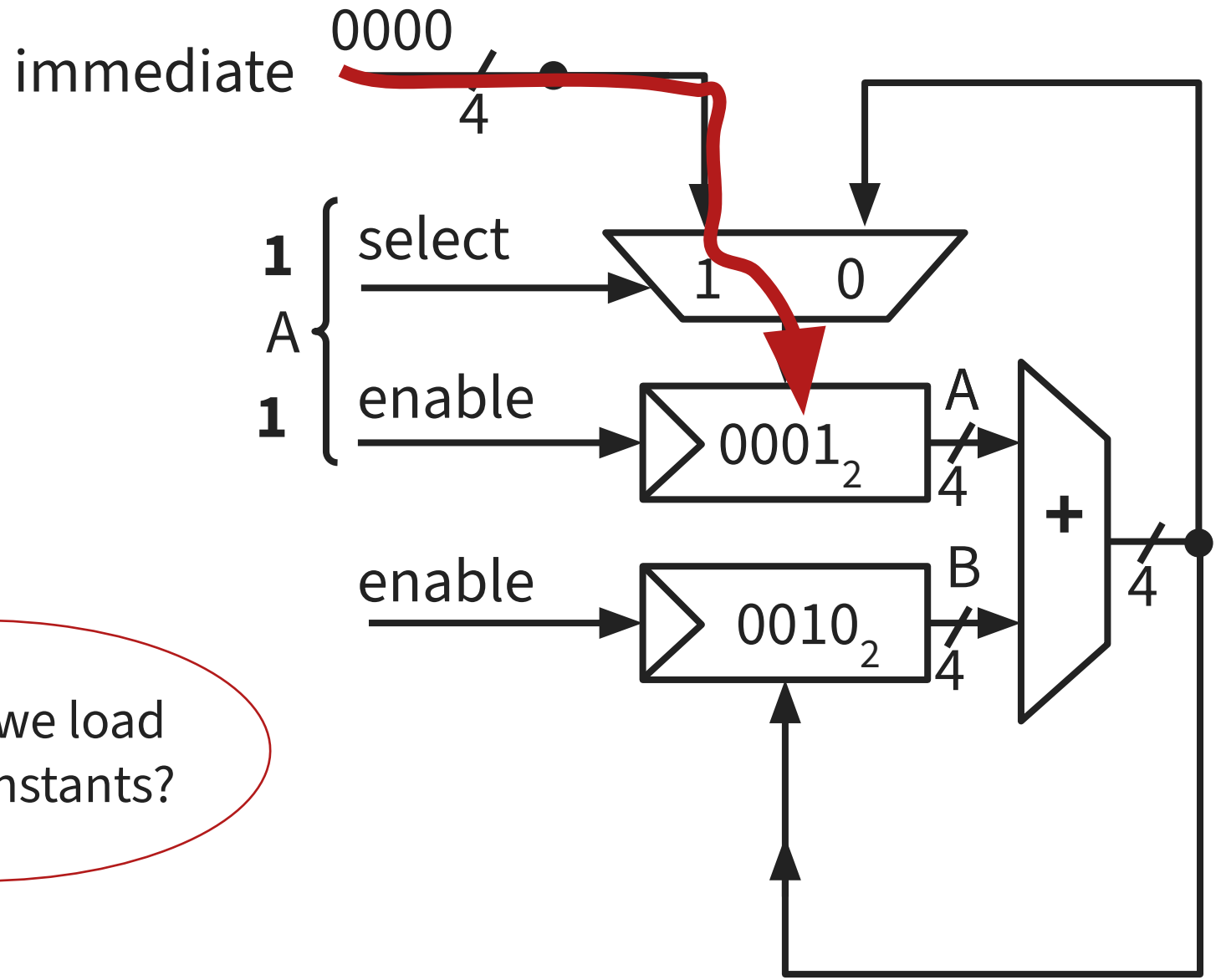
What should be the values of A.select and A.enable to load a new immediate value?

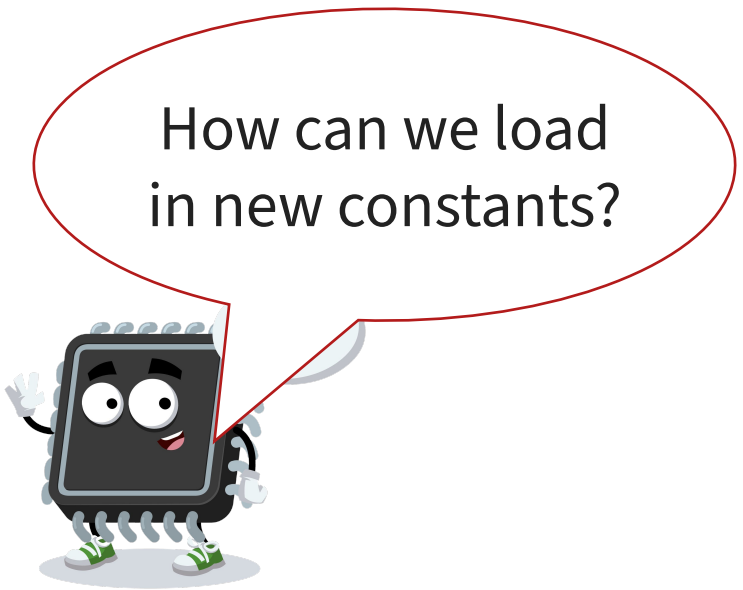




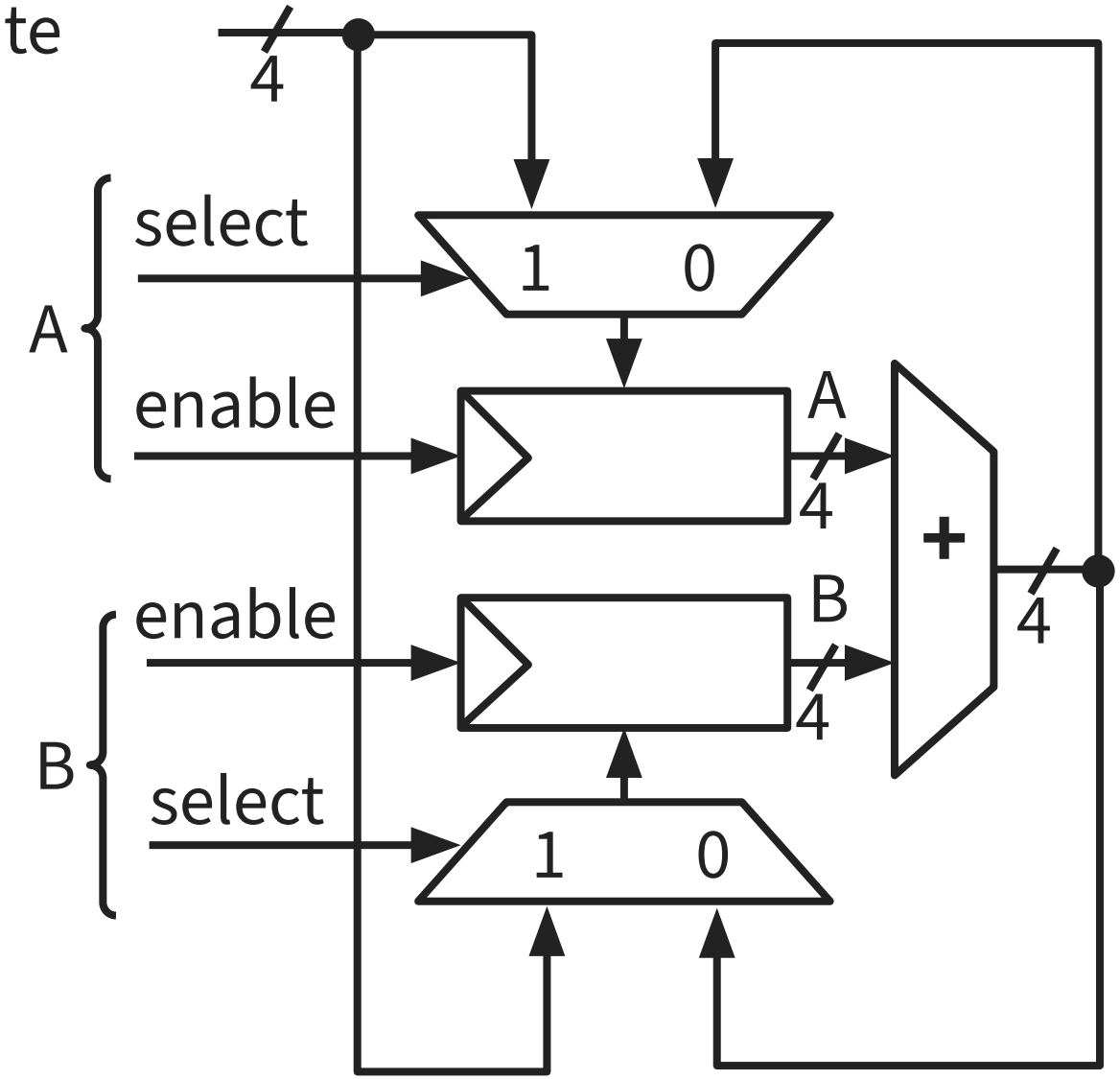


PollEv.com/cs3410


immediate
0000
4
1
A
1
select
enable
enable
1
0
0001_2
0010_2
A
B
4
4
4
+

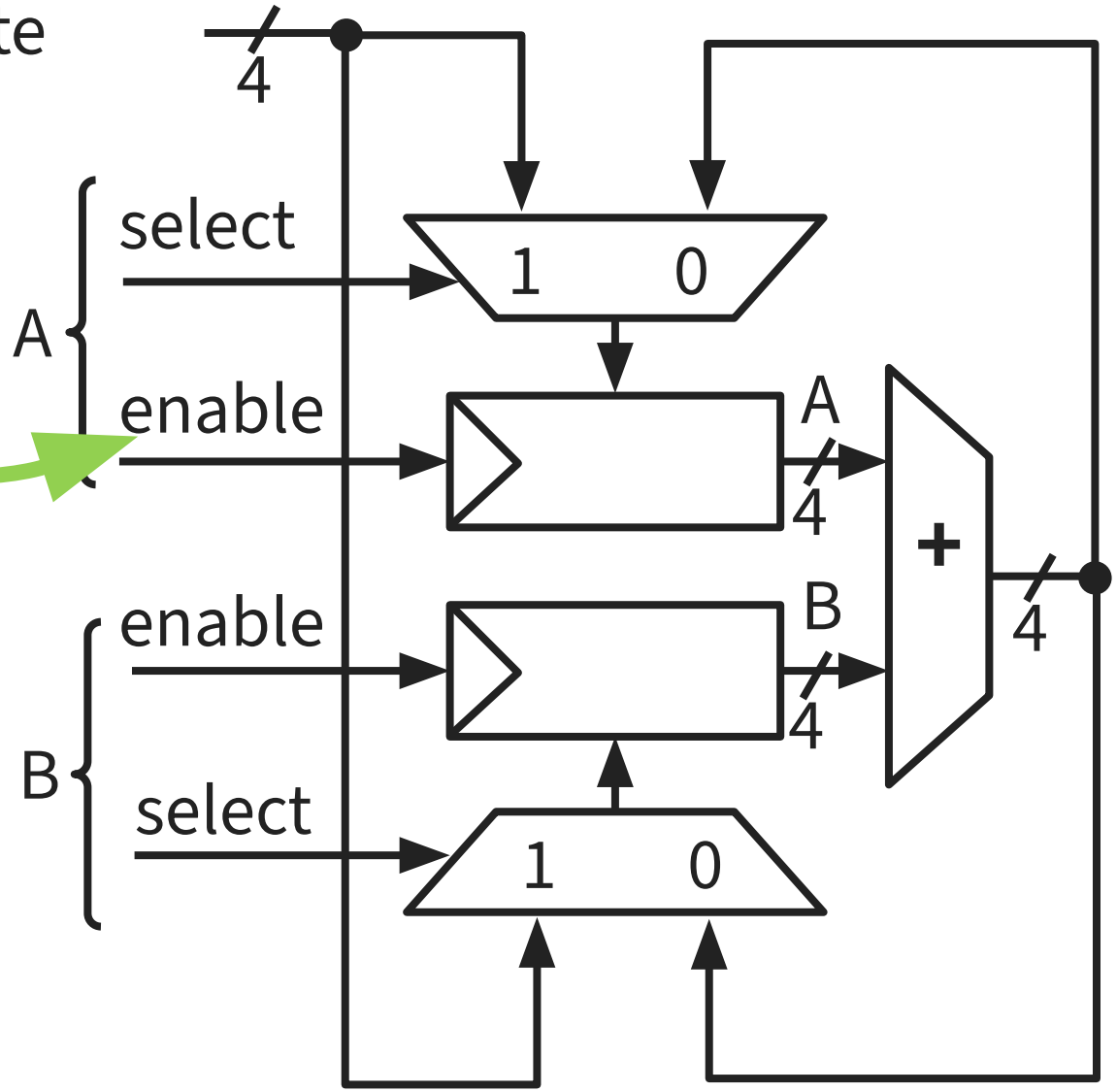

How can we load
in new constants?

Instruction Format: immediate

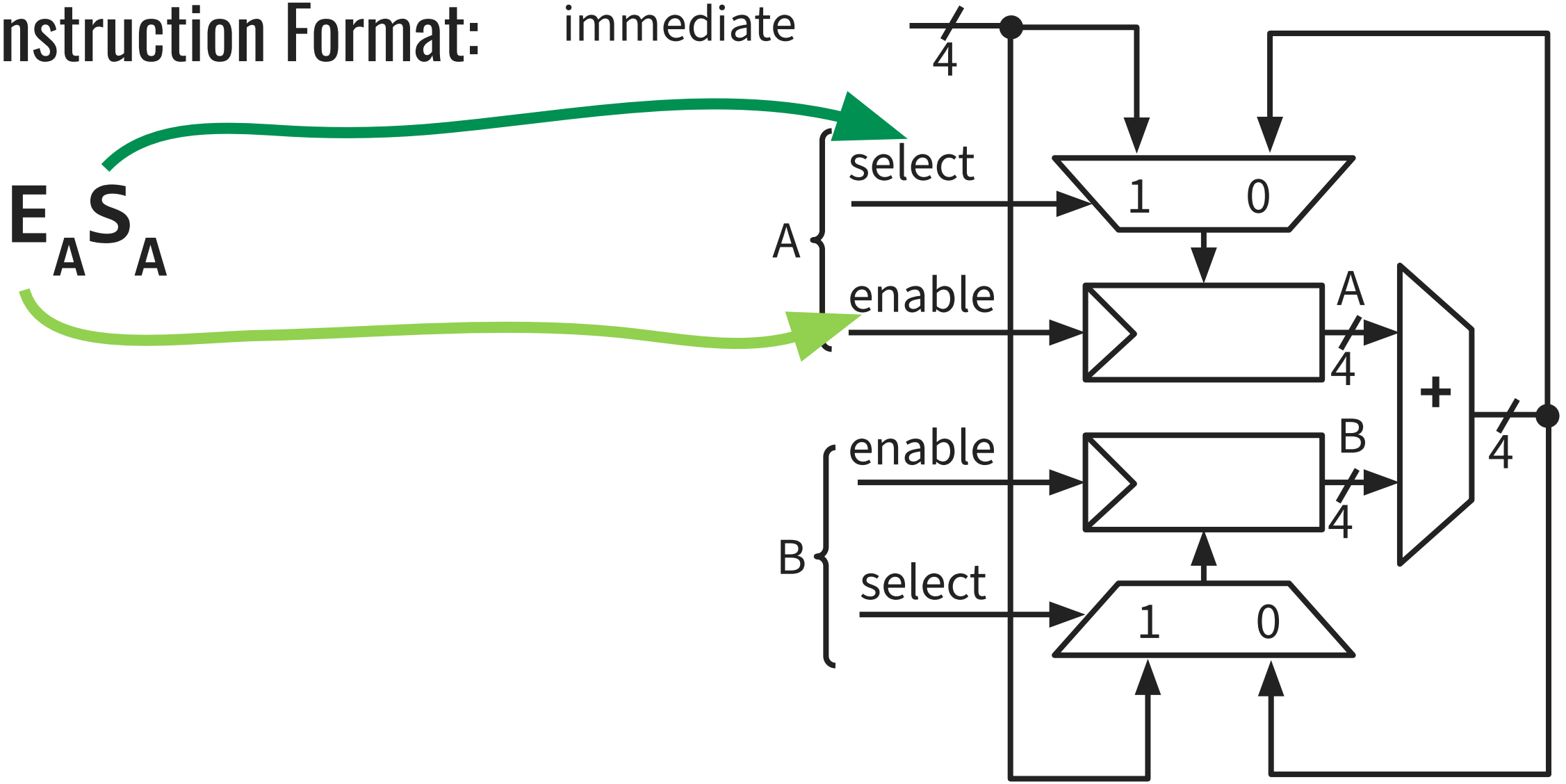


Instruction Format: immediate

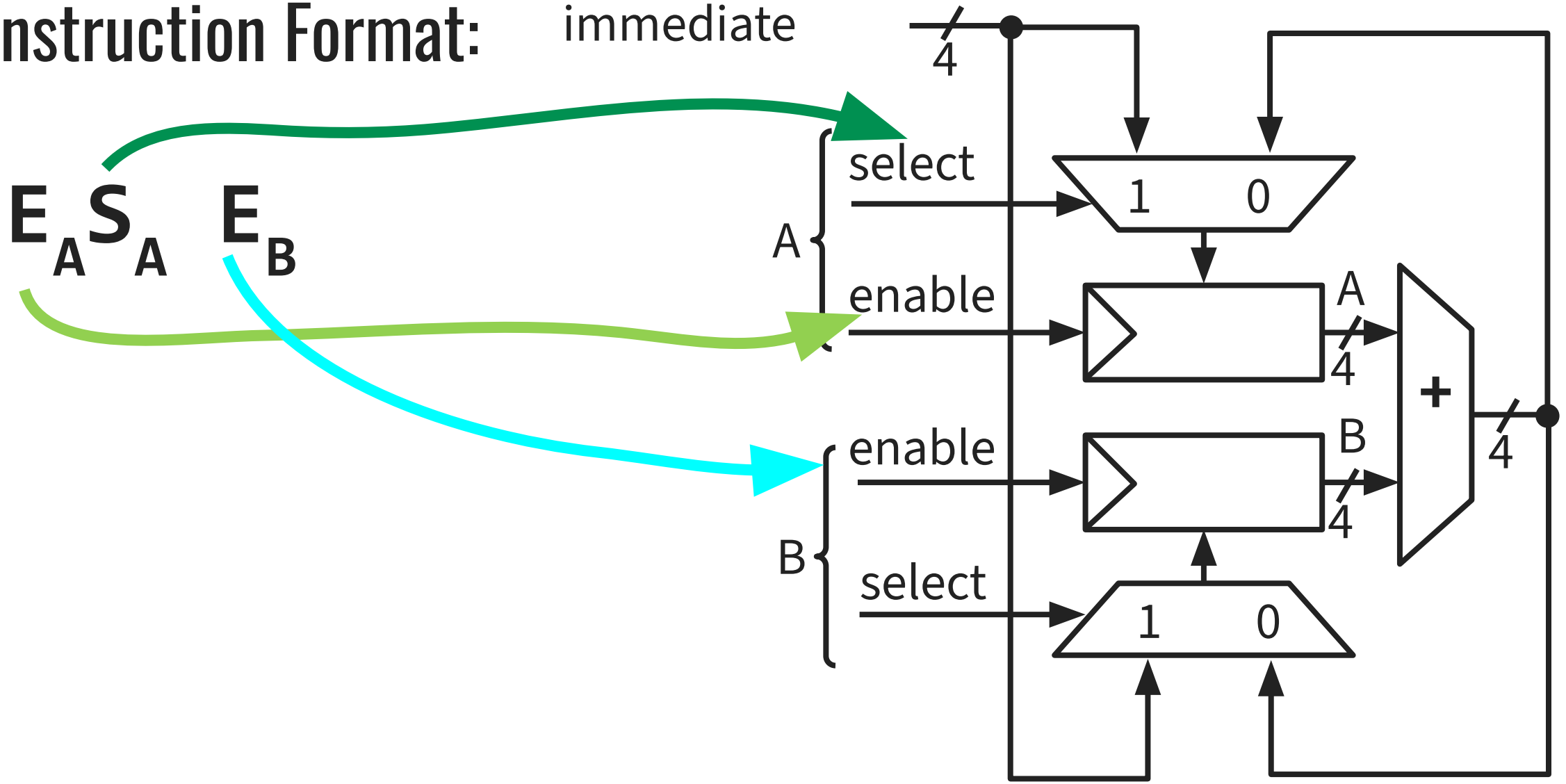
E_A



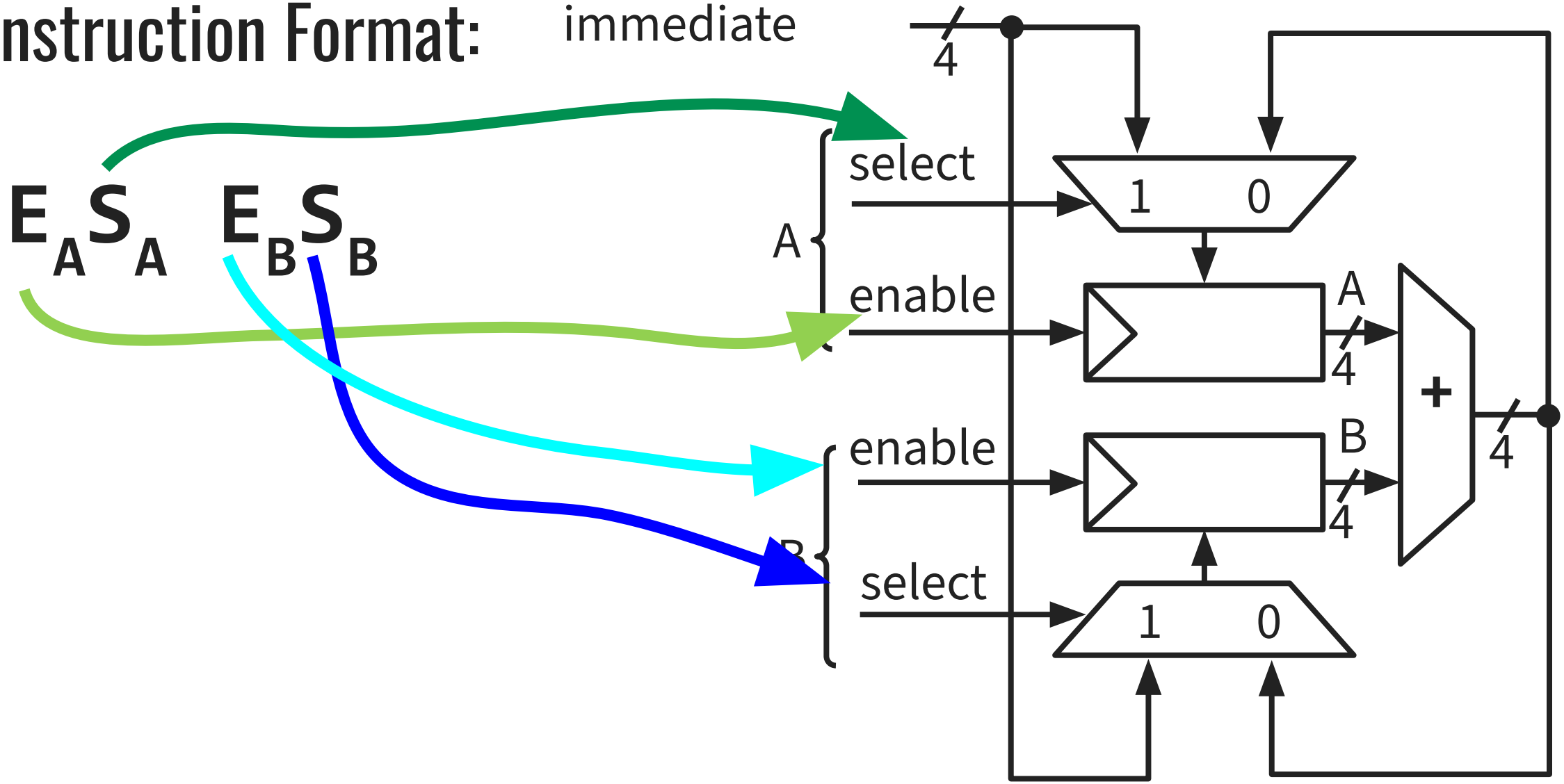
Instruction Format:



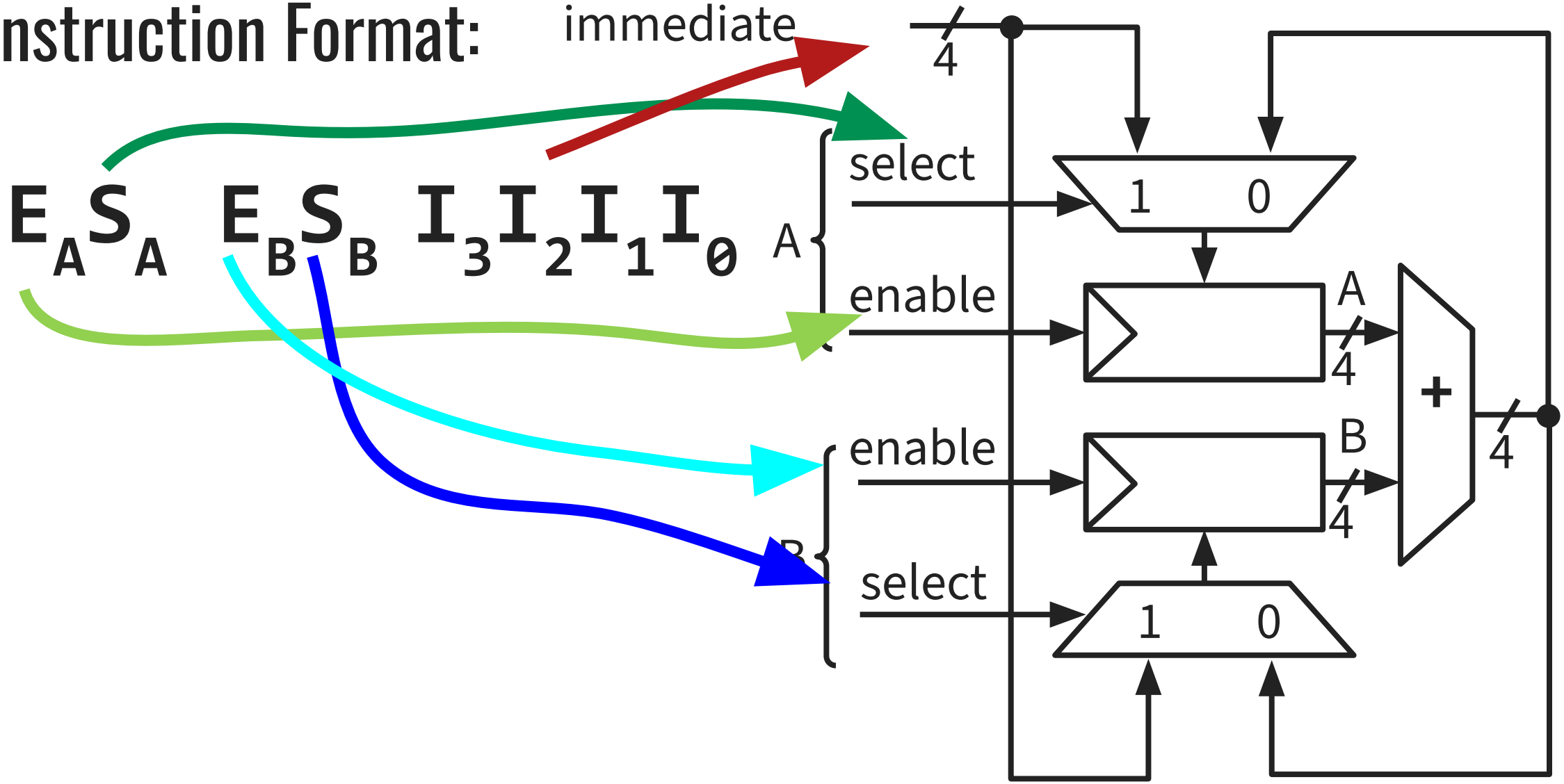
Instruction Format:



Instruction Format:



Instruction Format:



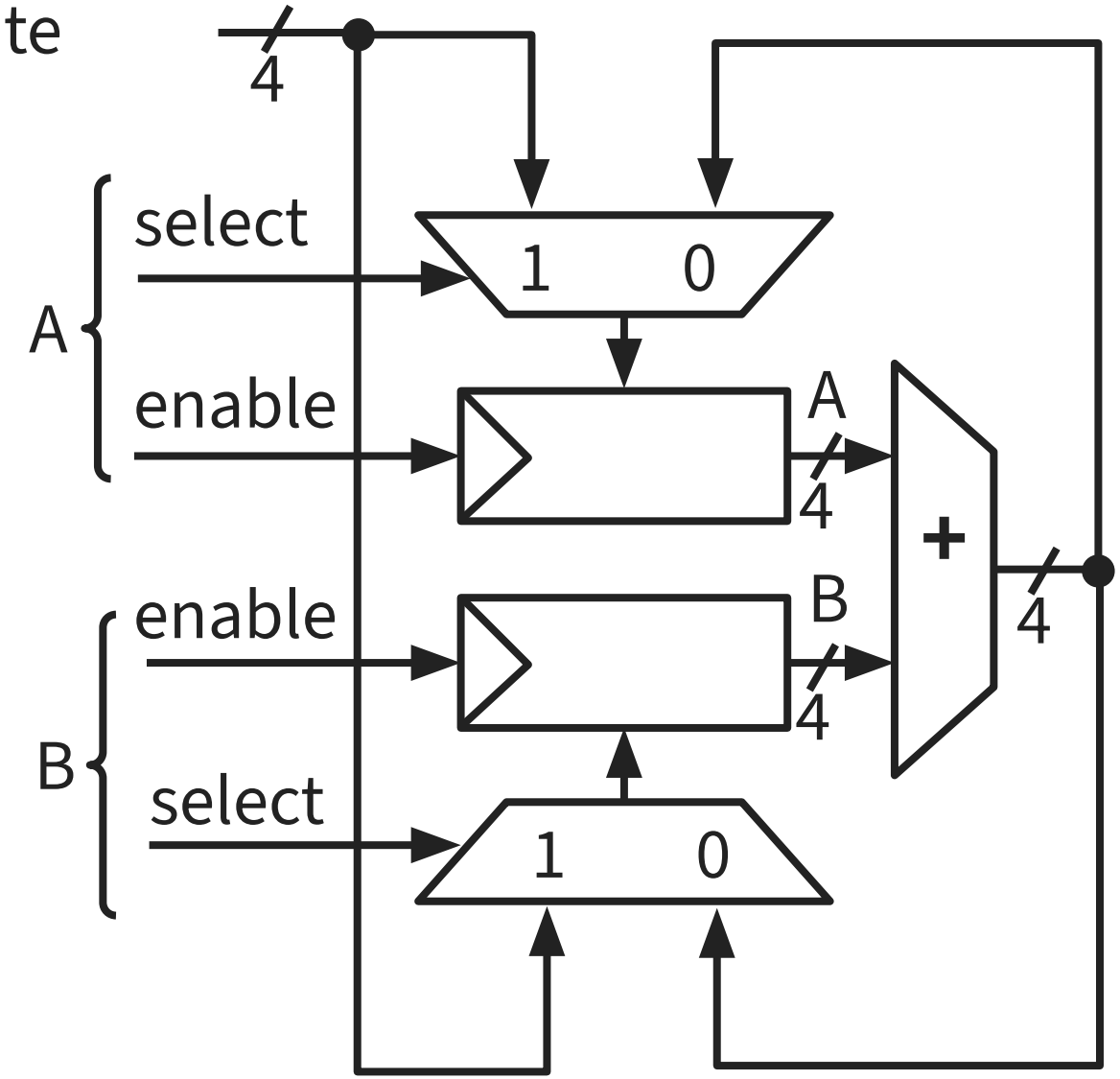
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

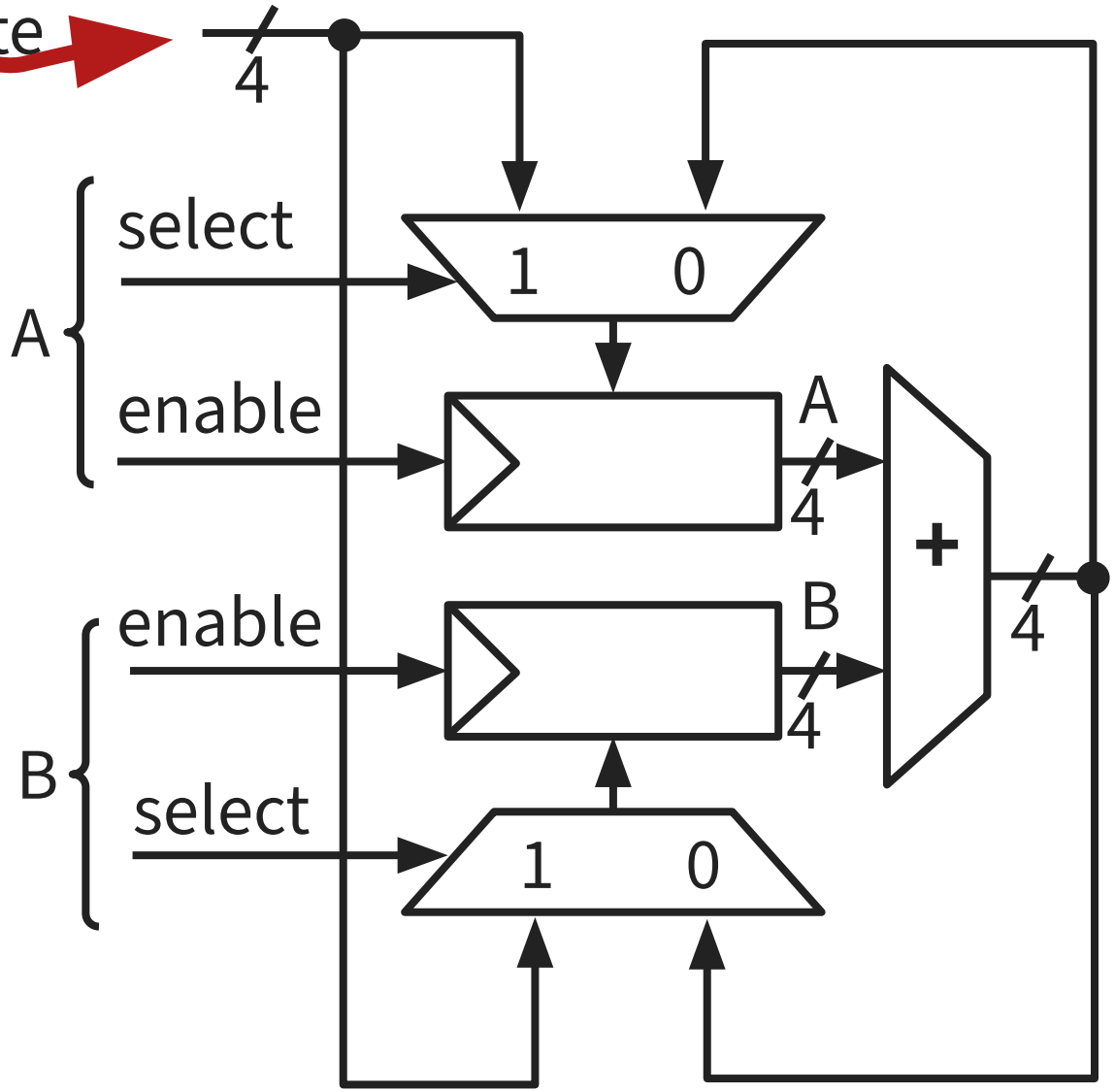
Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

immediate

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

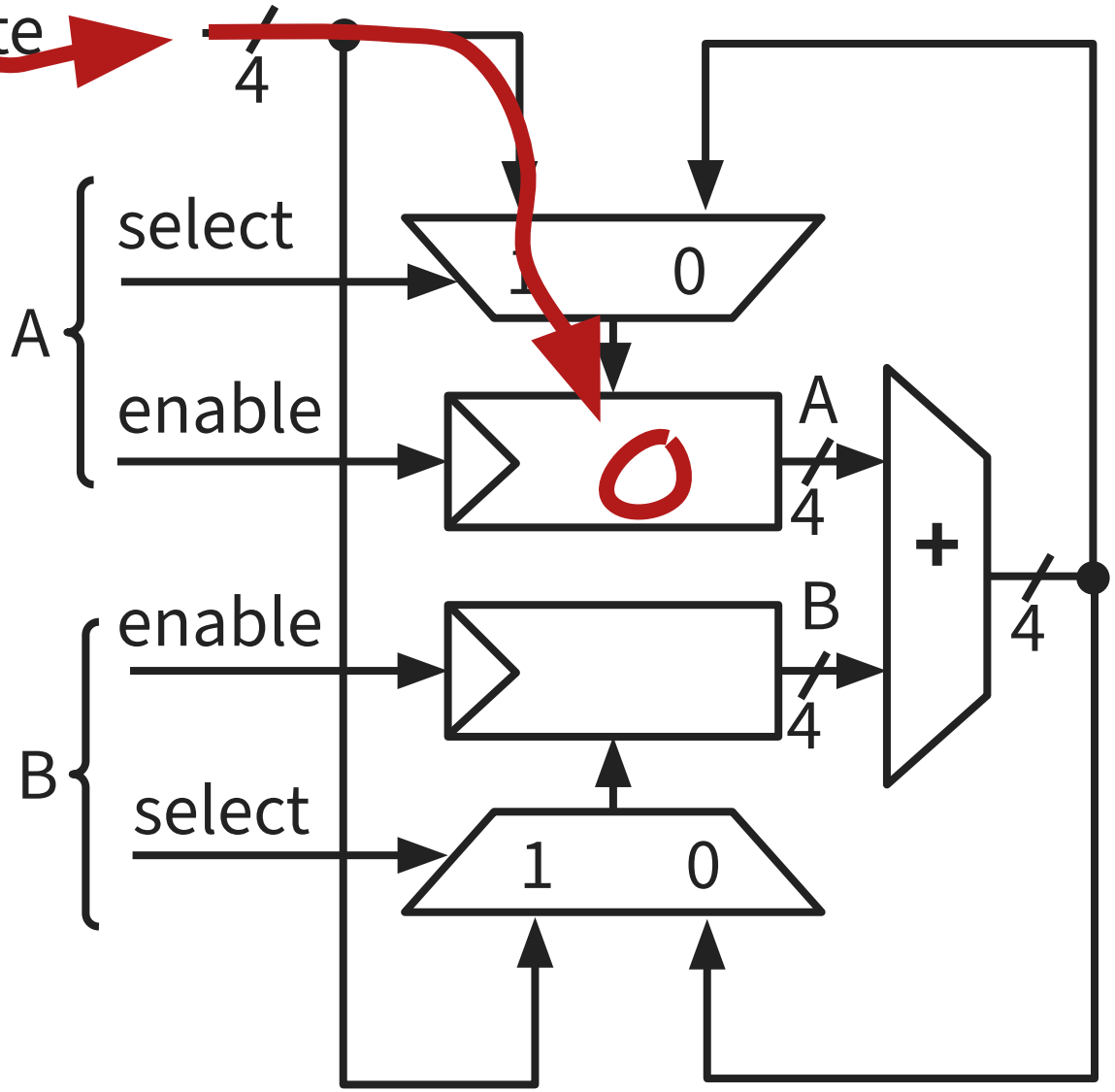
PollEv.com/cs3410

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

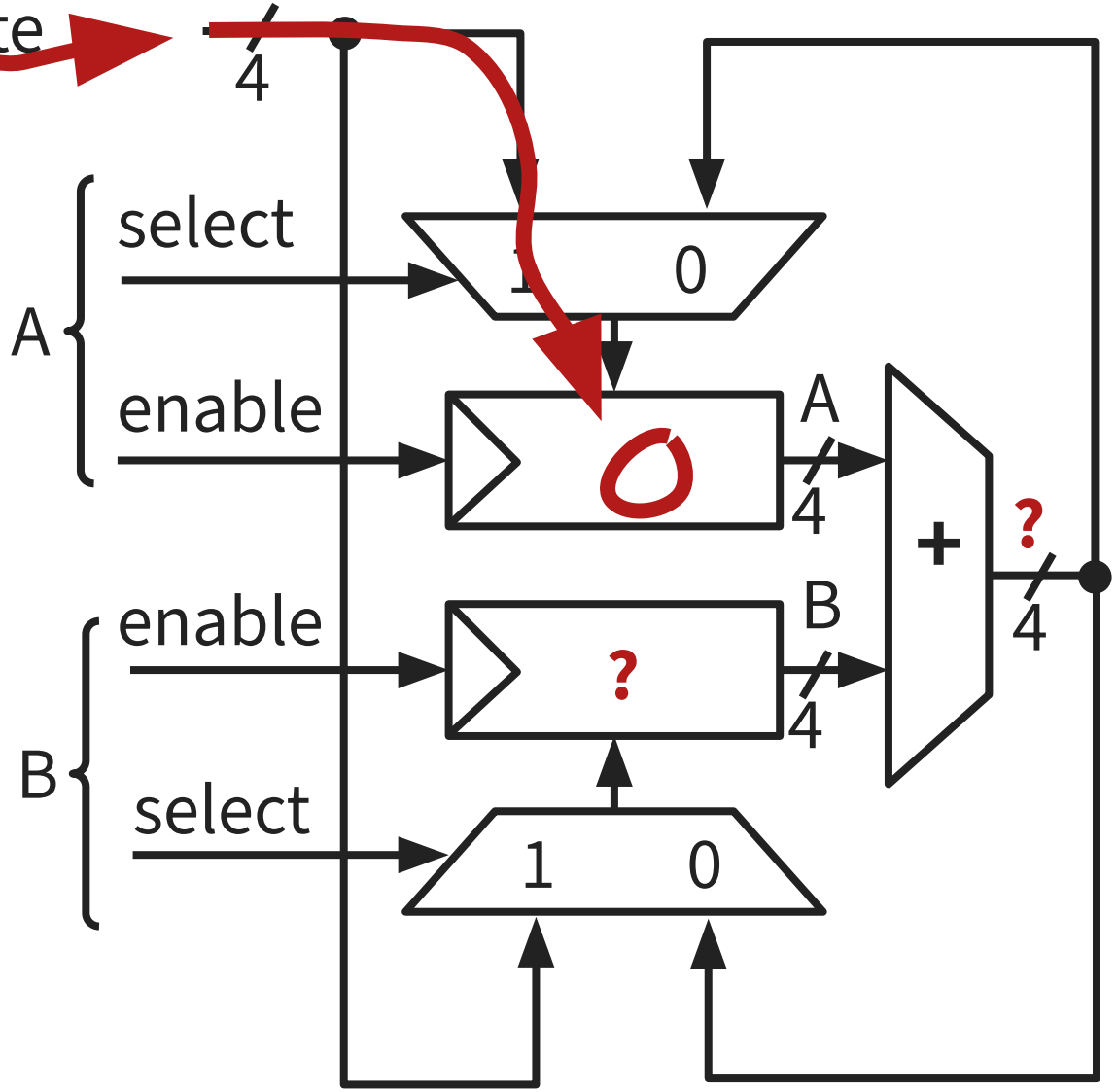
PollEv.com/cs3410

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

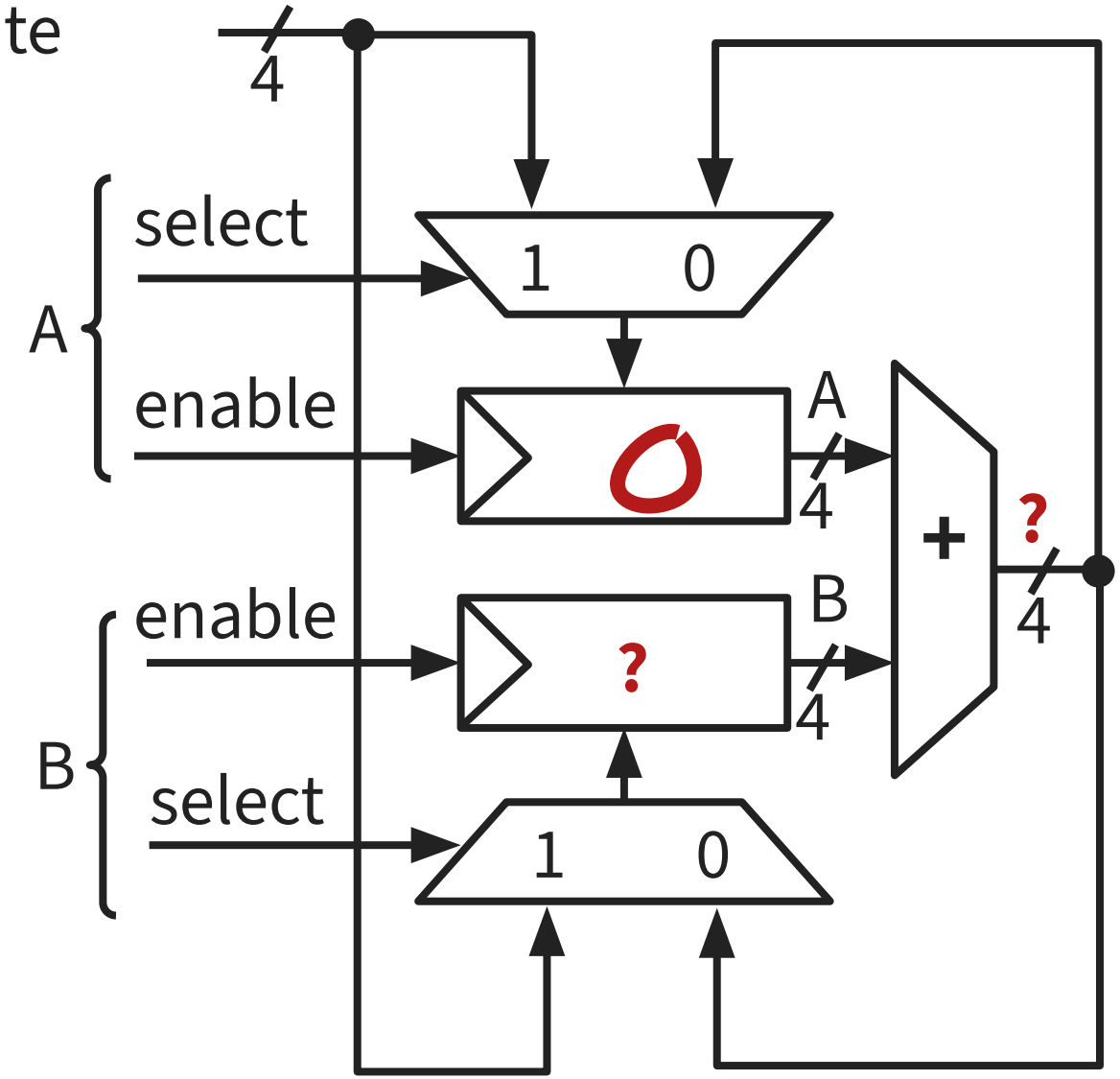
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

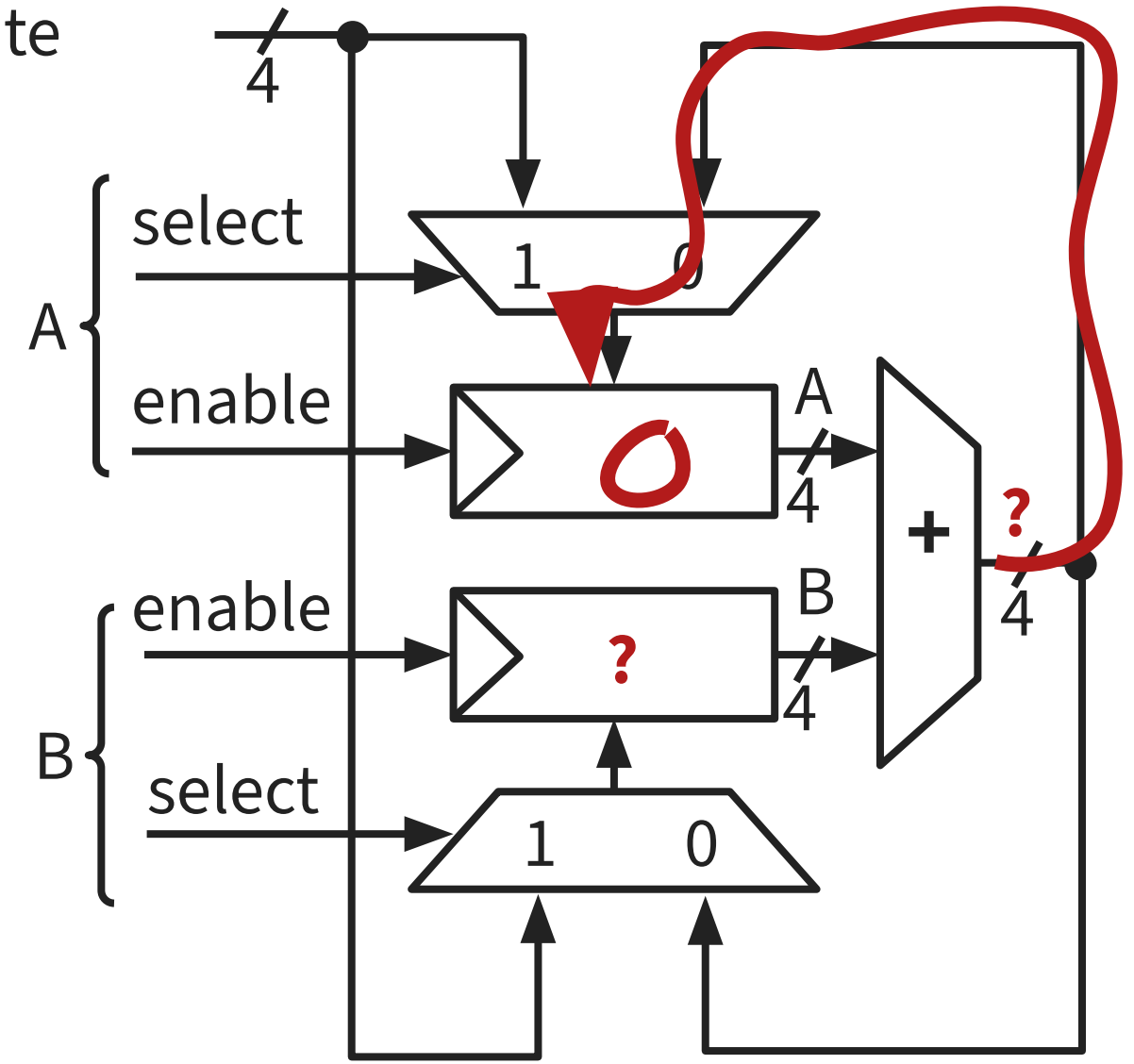
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

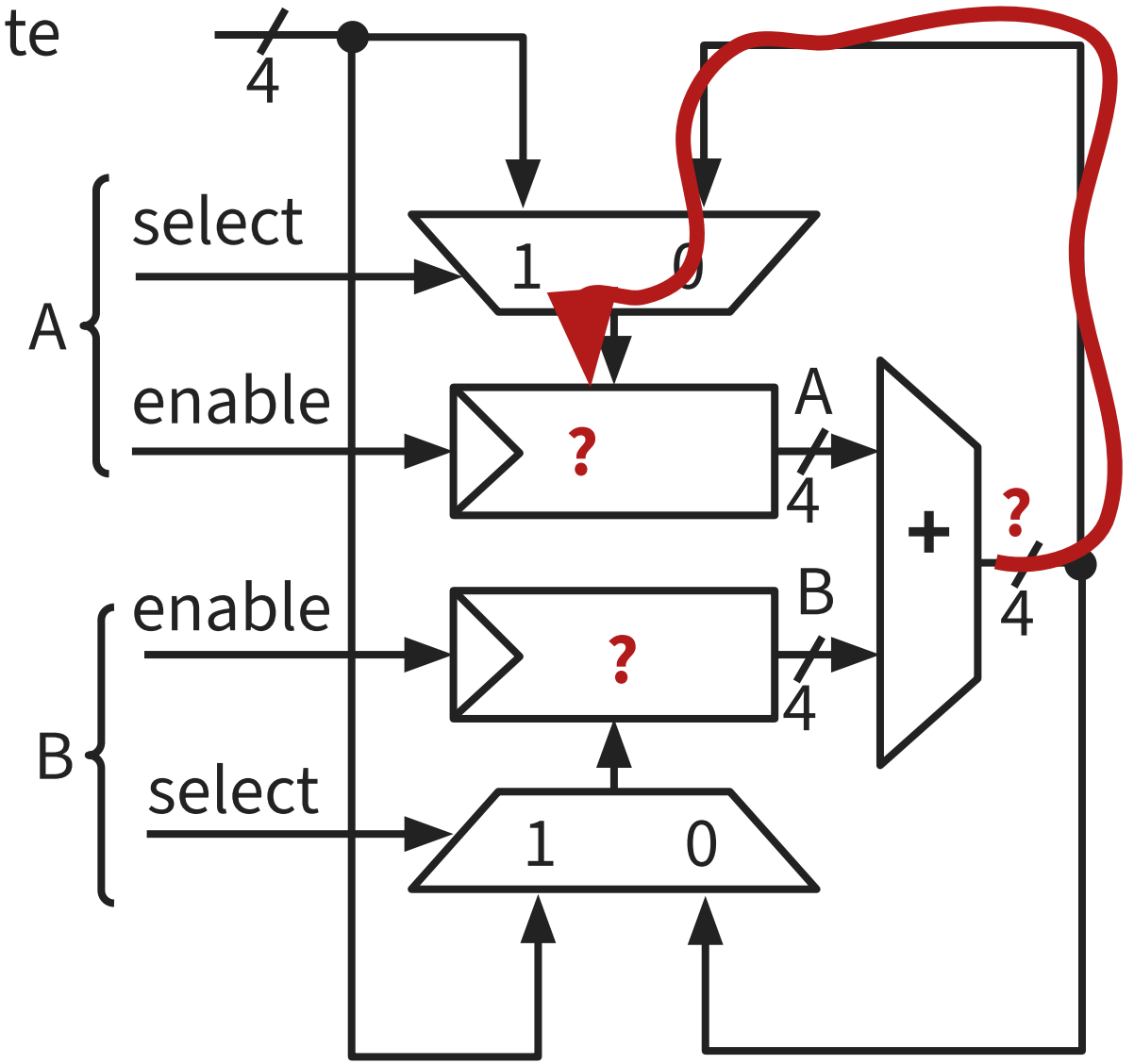
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #1:

```
11 00 0000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

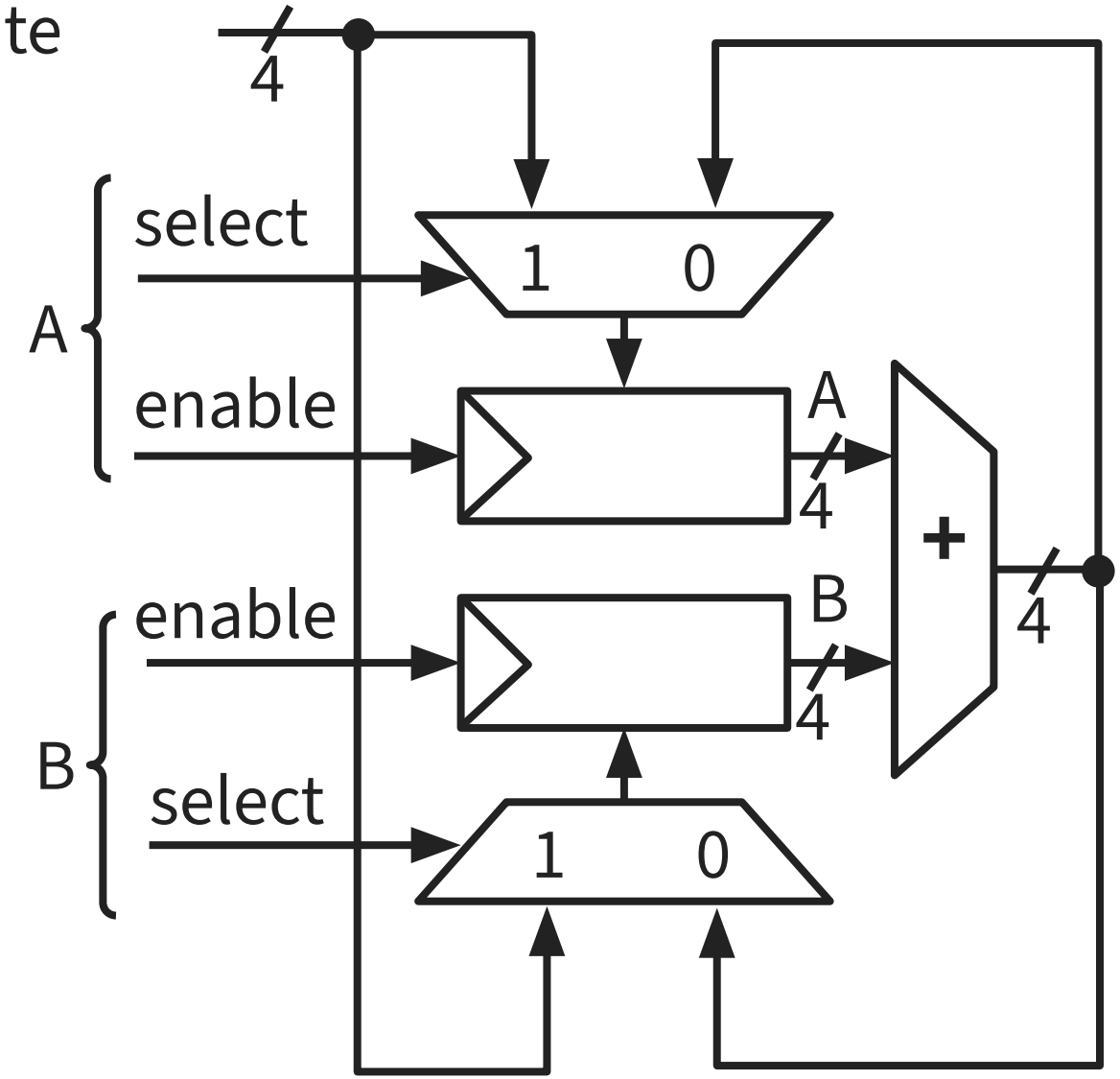
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

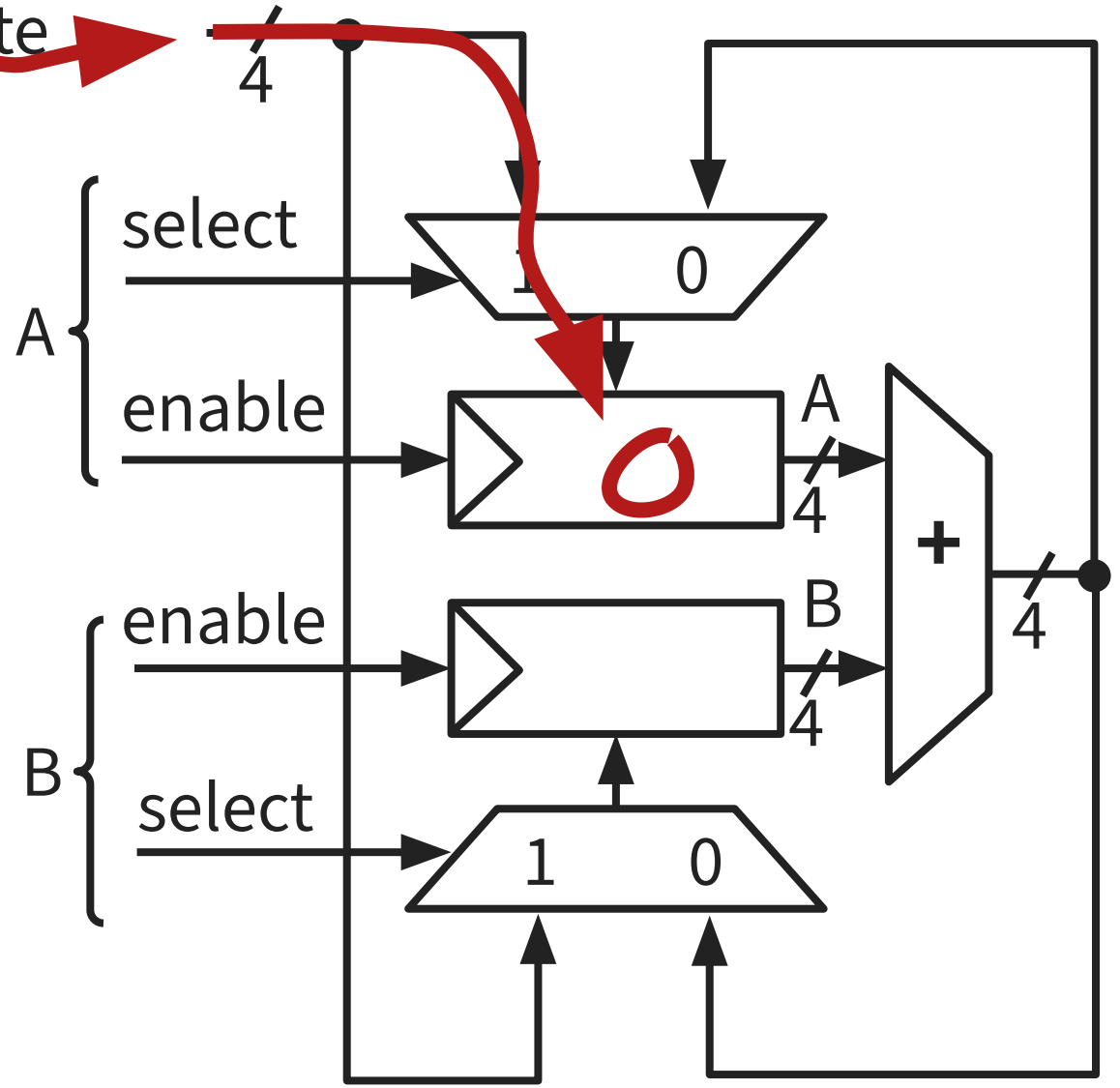
PollEv.com/cs3410

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

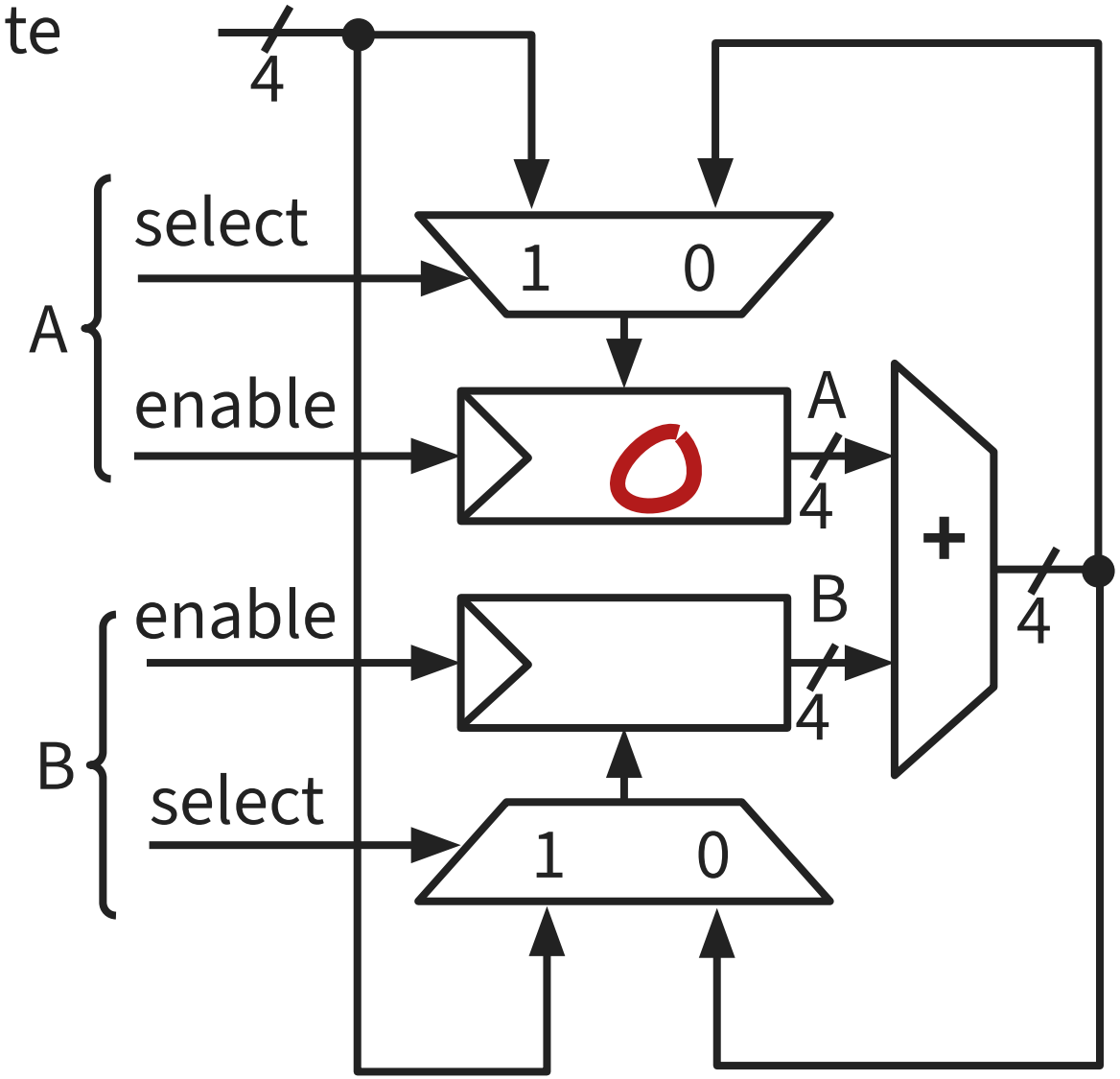
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

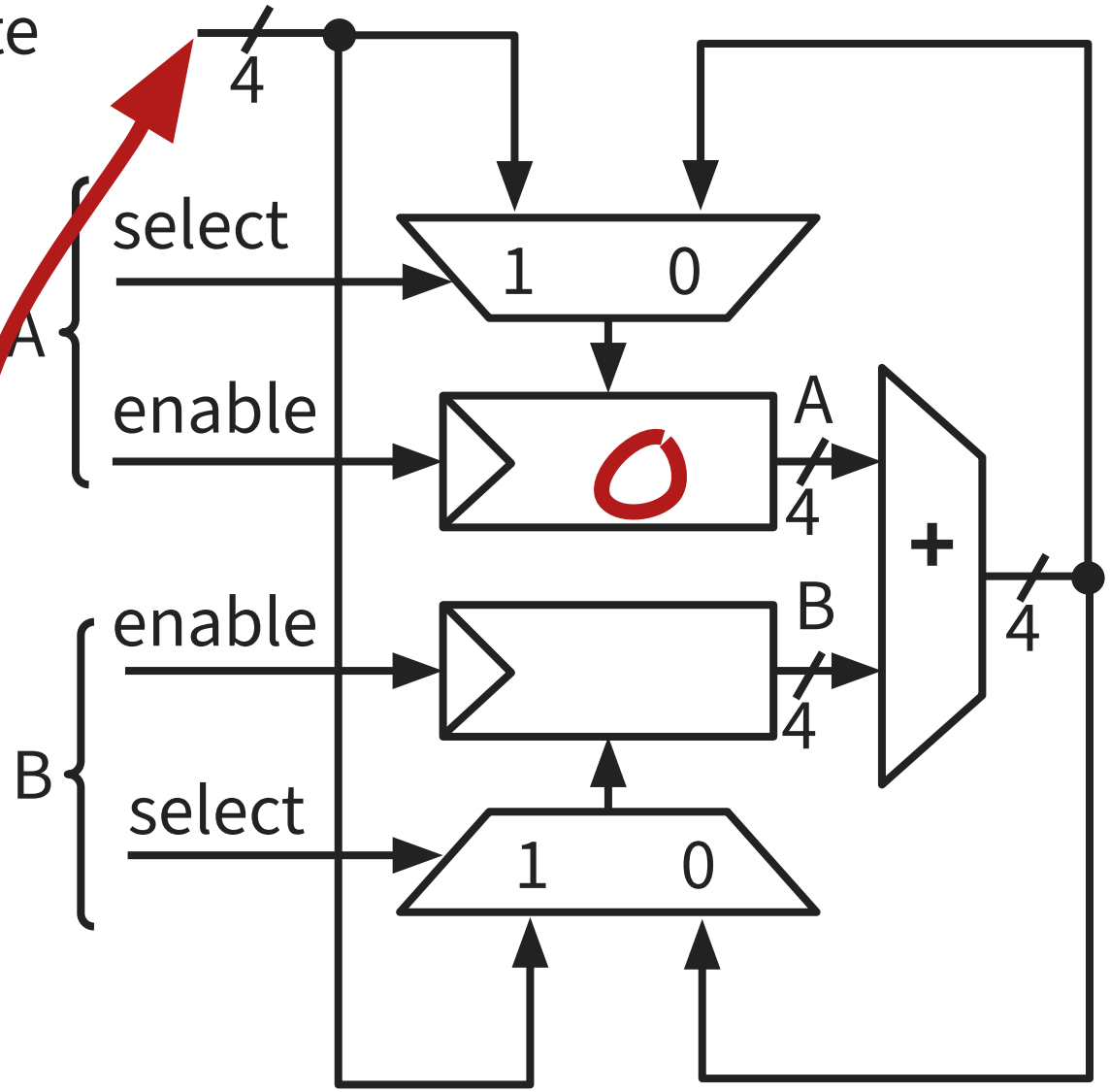
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

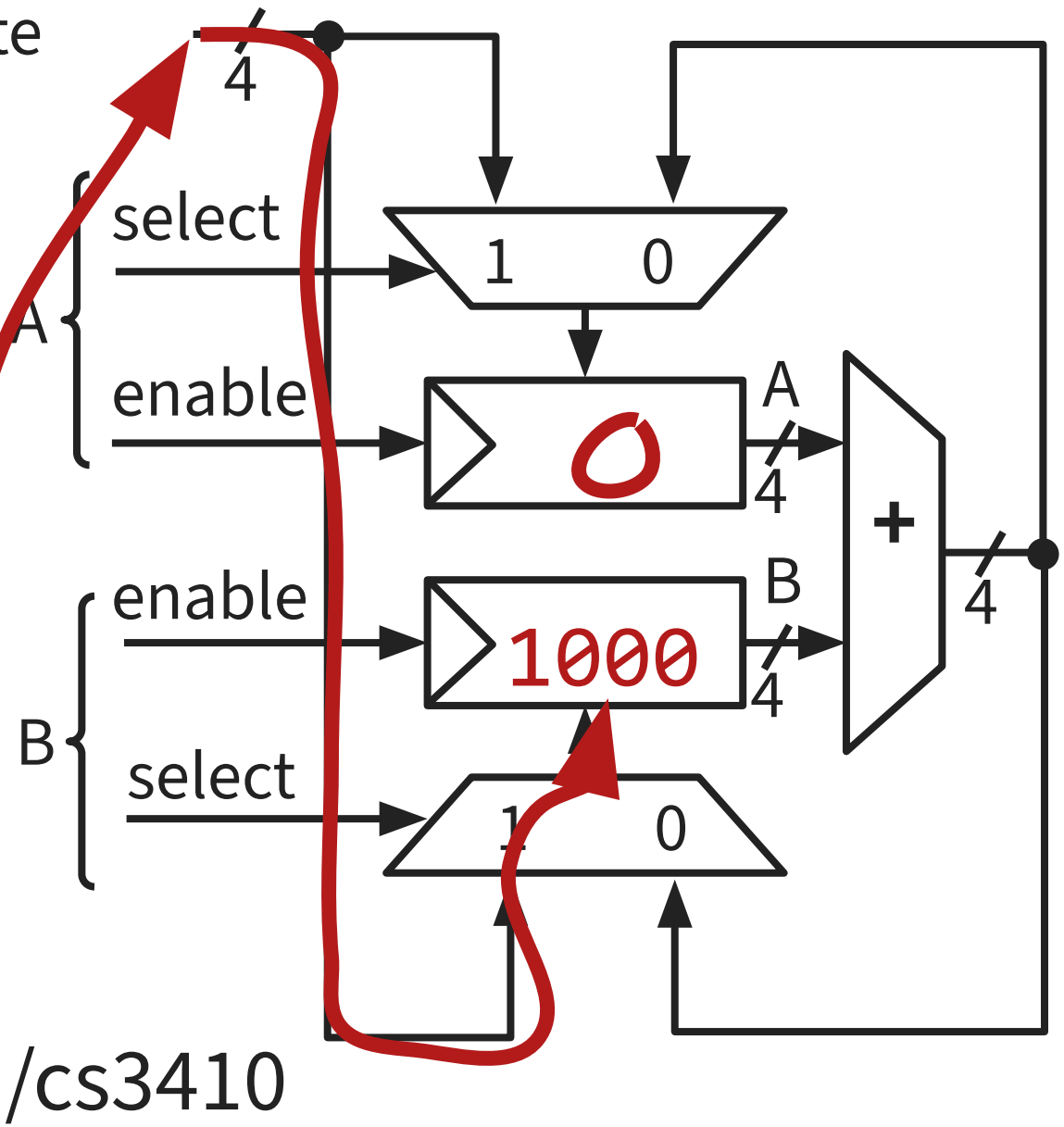
Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

immediate

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

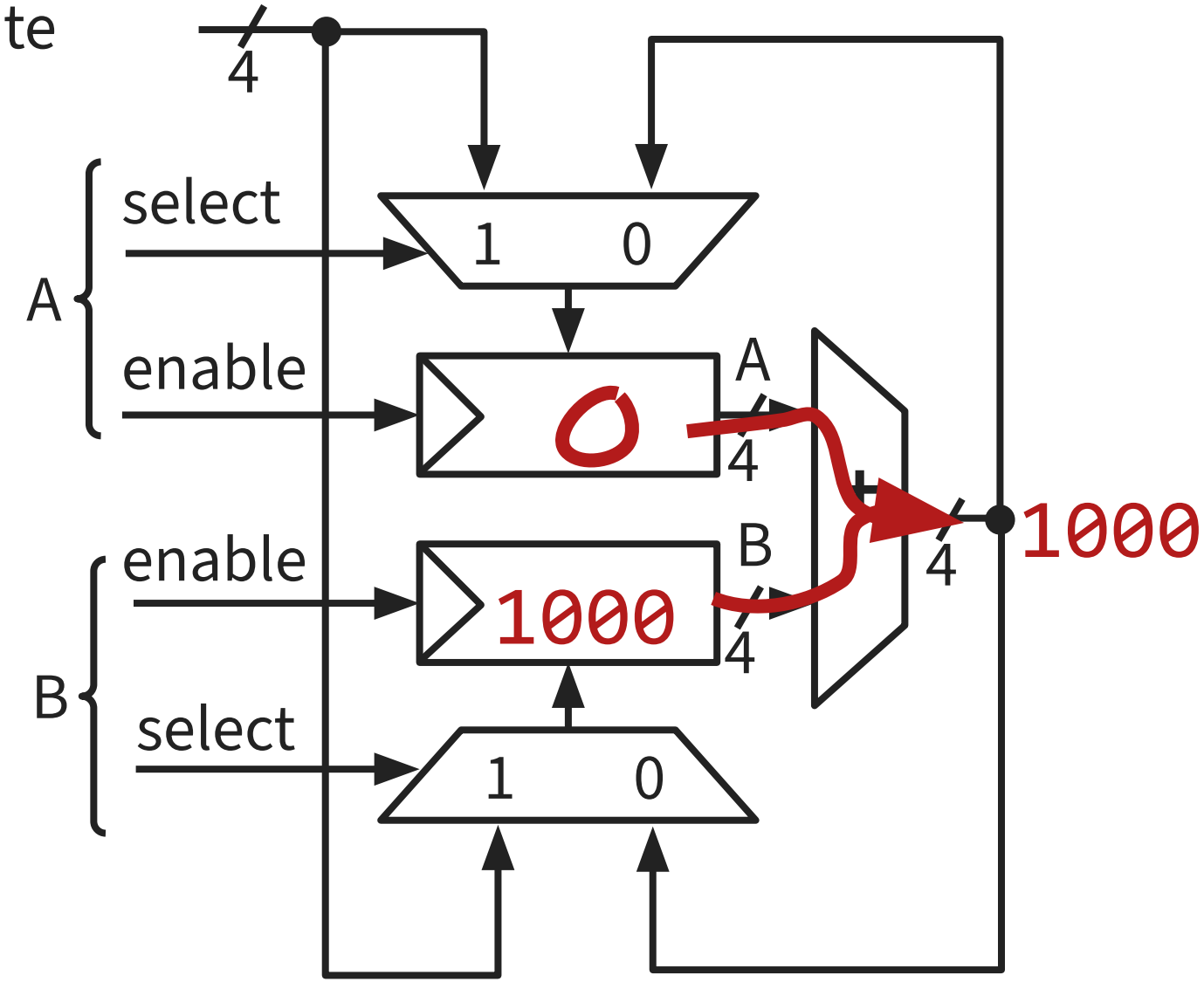
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

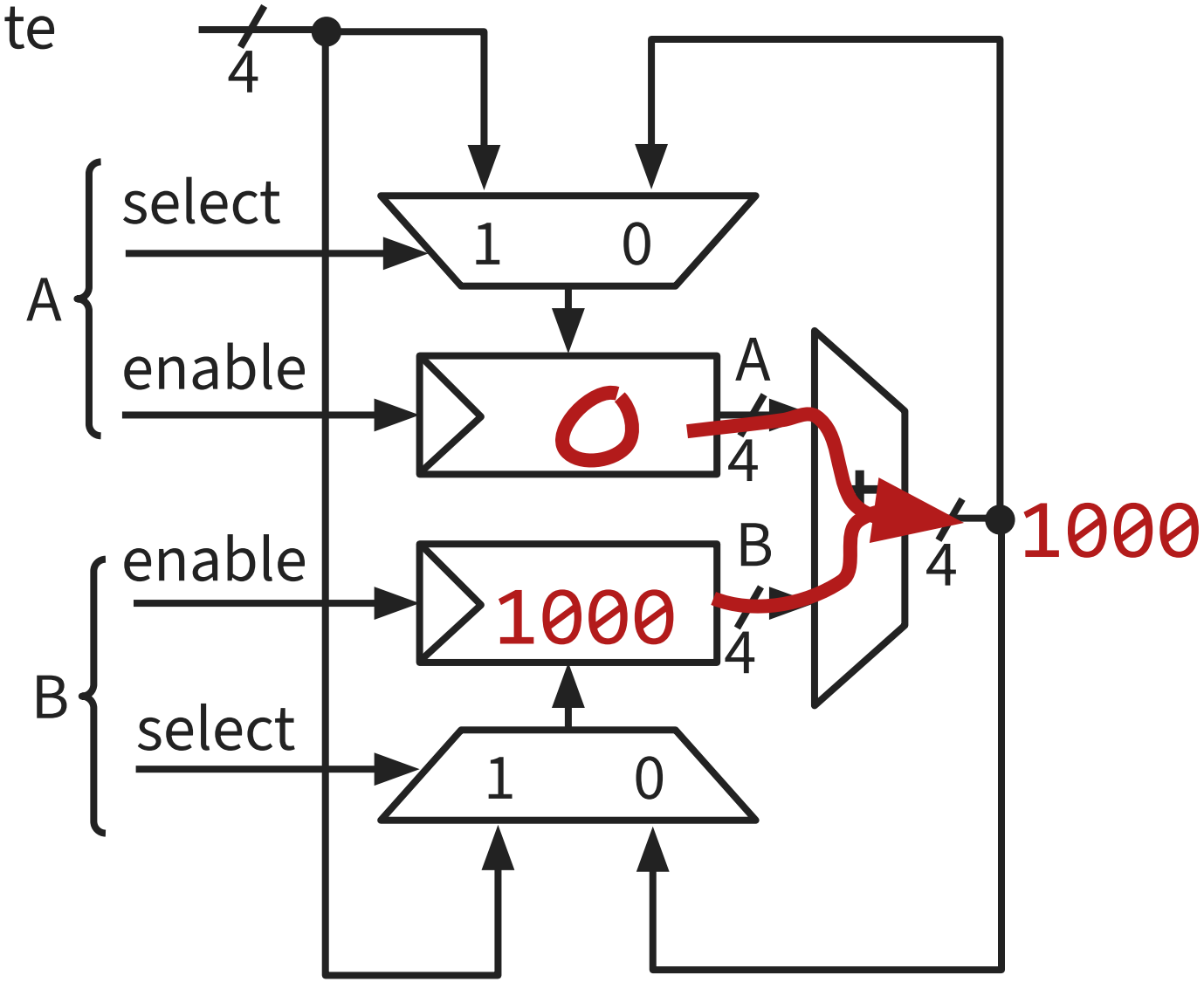
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

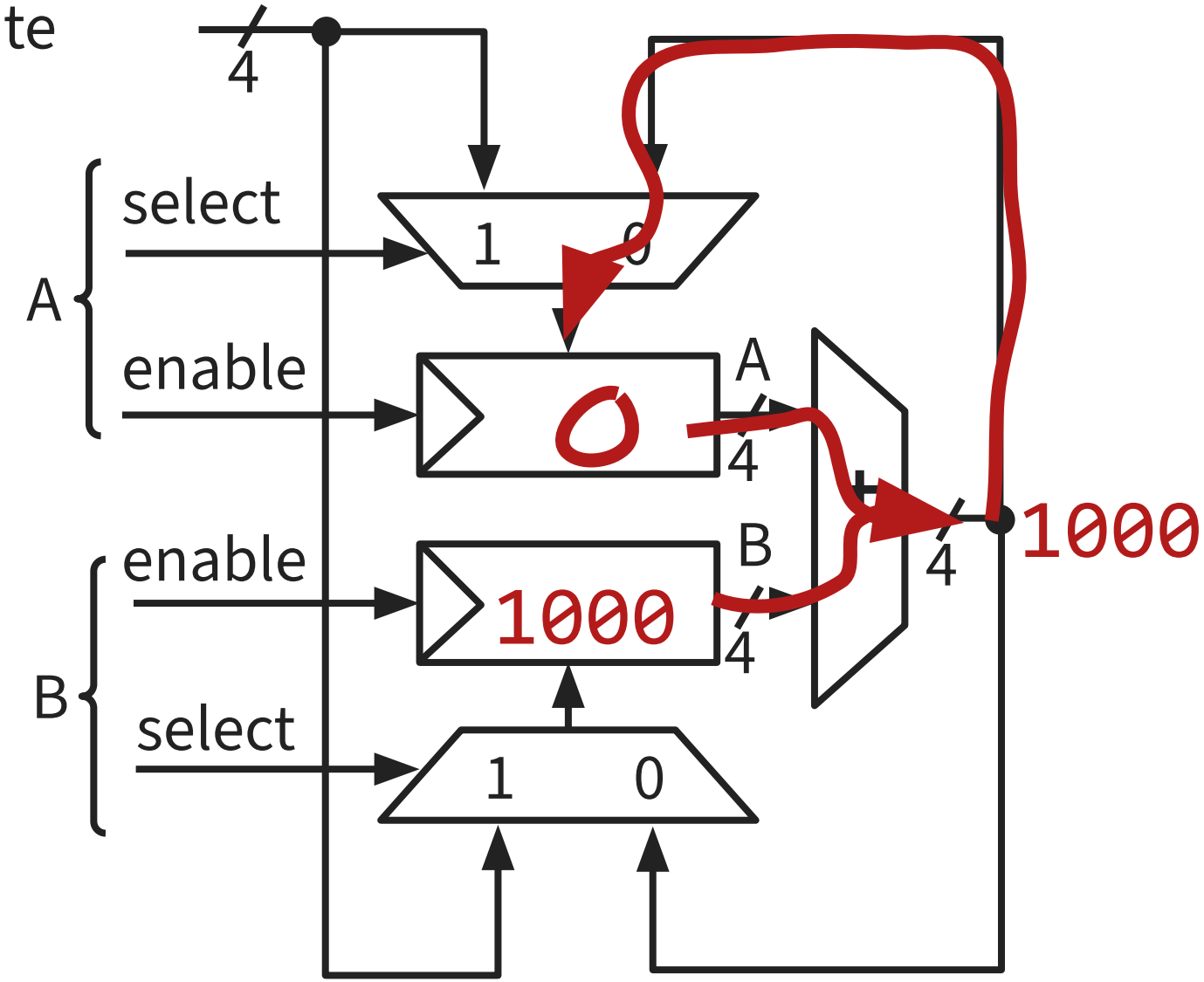
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

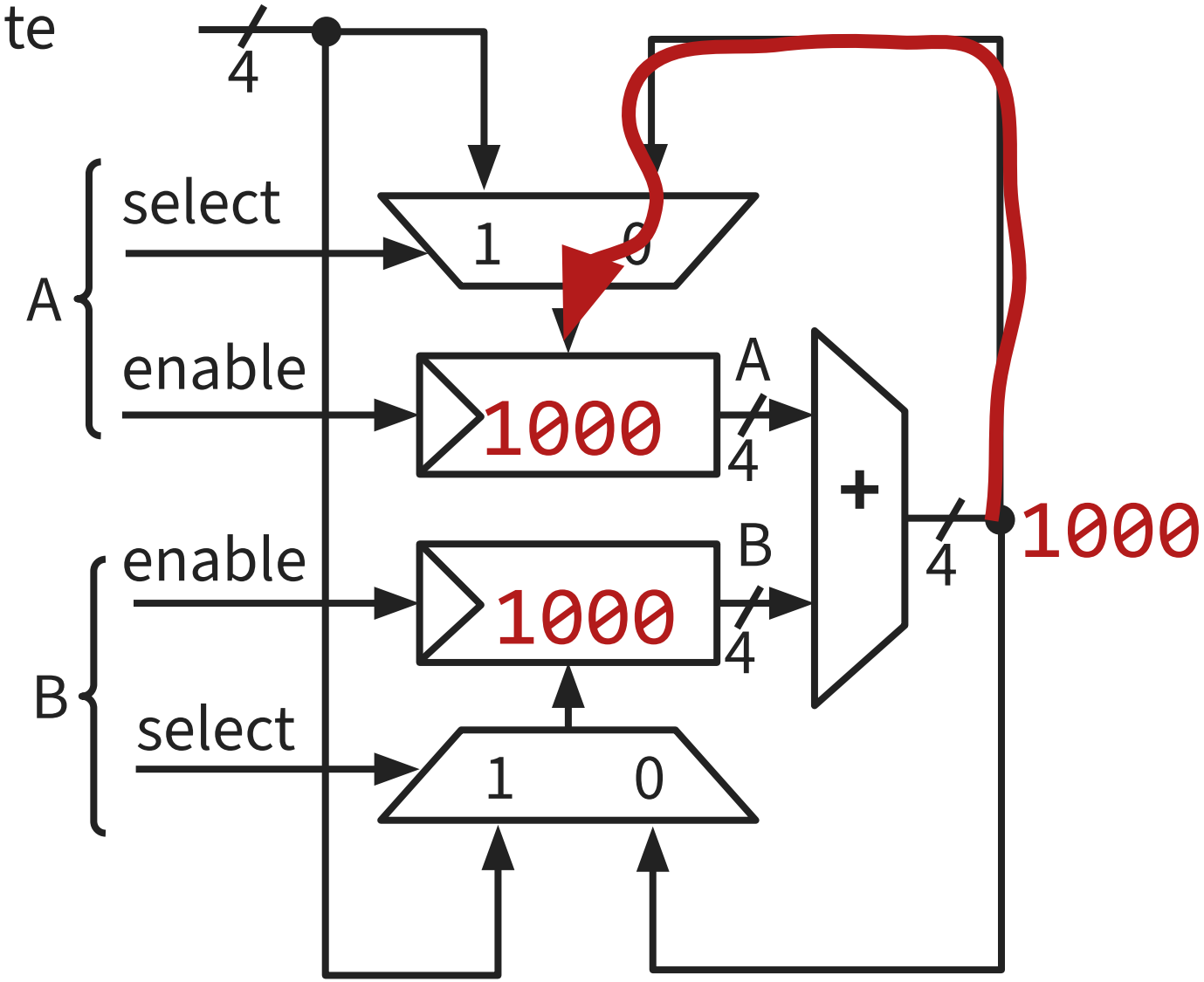
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 00 0000
01 11 1000
10 00 0101
```



What is the *result*?

PollEv.com/cs3410

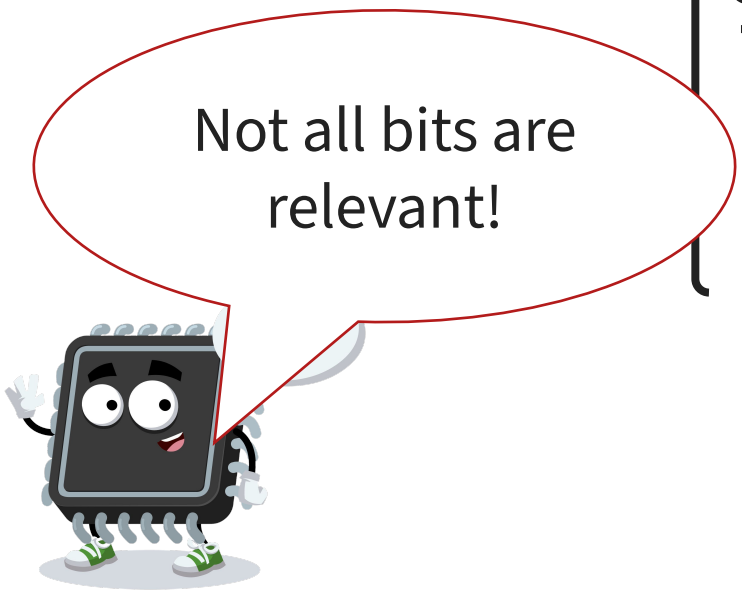
Instruction Format:

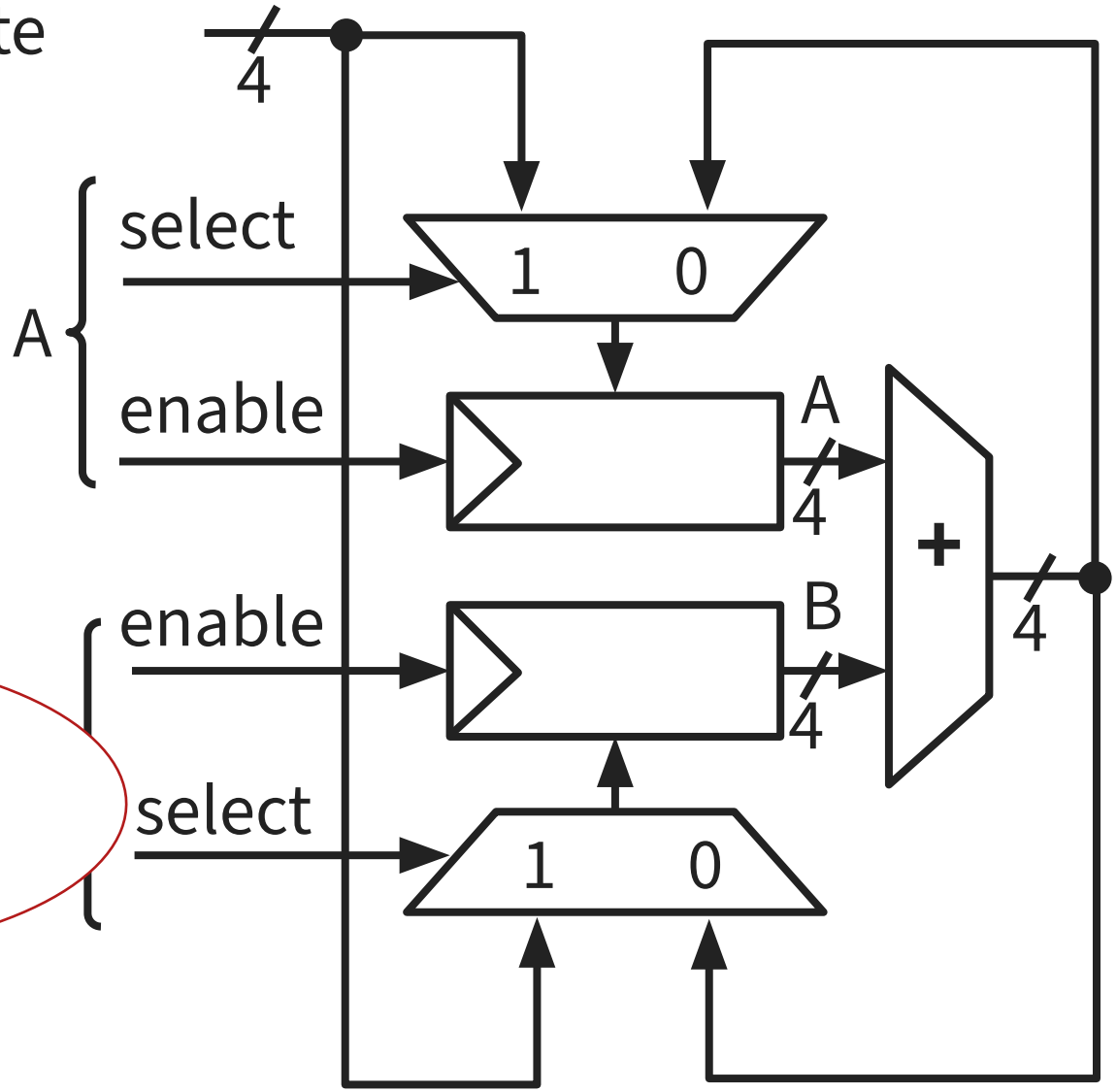
immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 0? 0000
0? 11 1000
10 00 0101
```





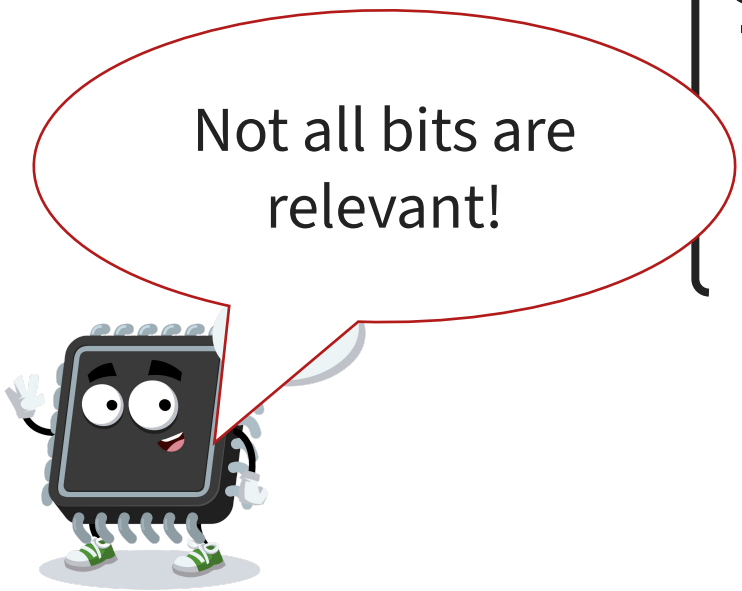
Instruction Format:

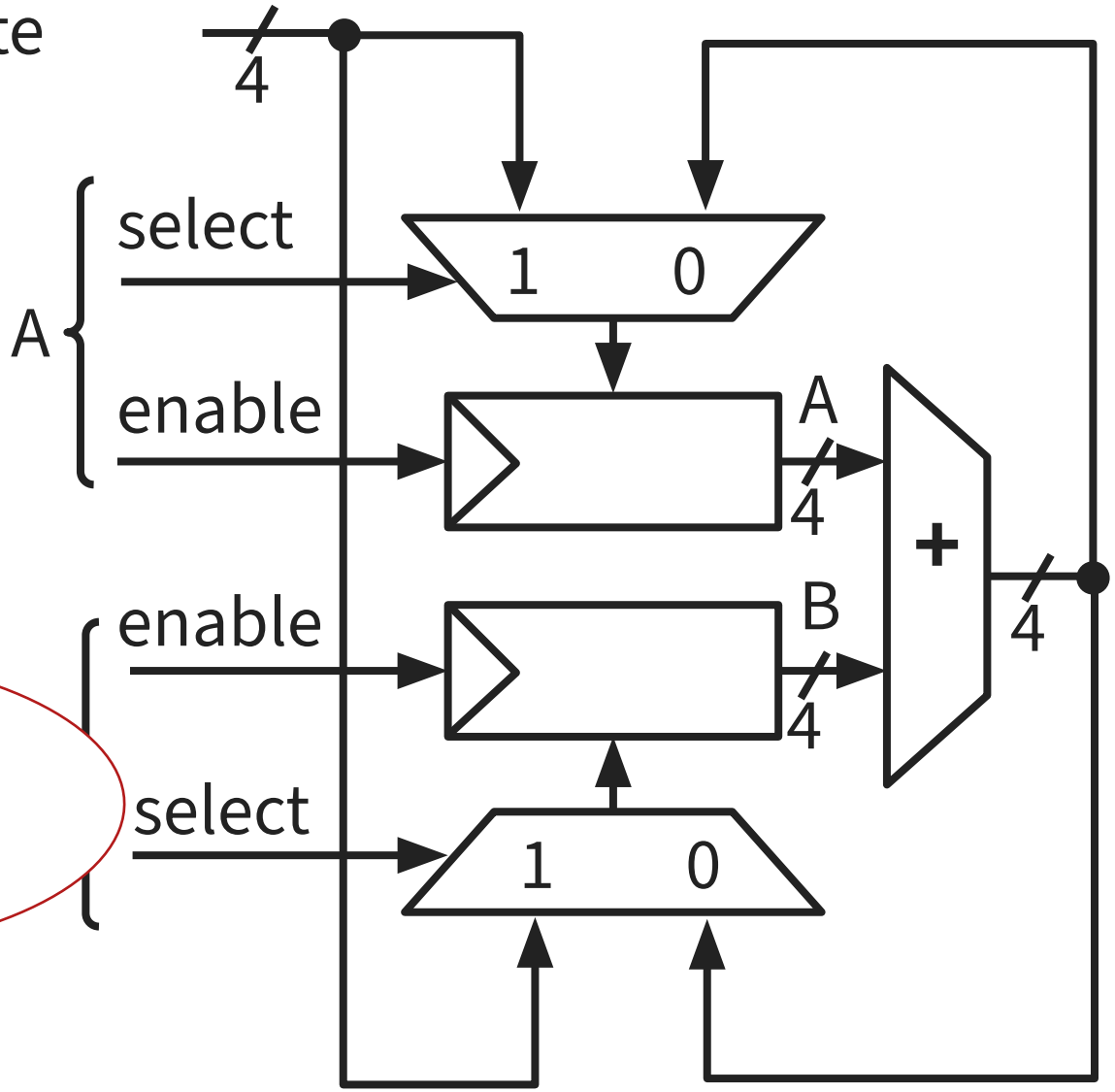
immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Program #2:

```
11 0? 0000
0? 11 1000
10 0? ????
```





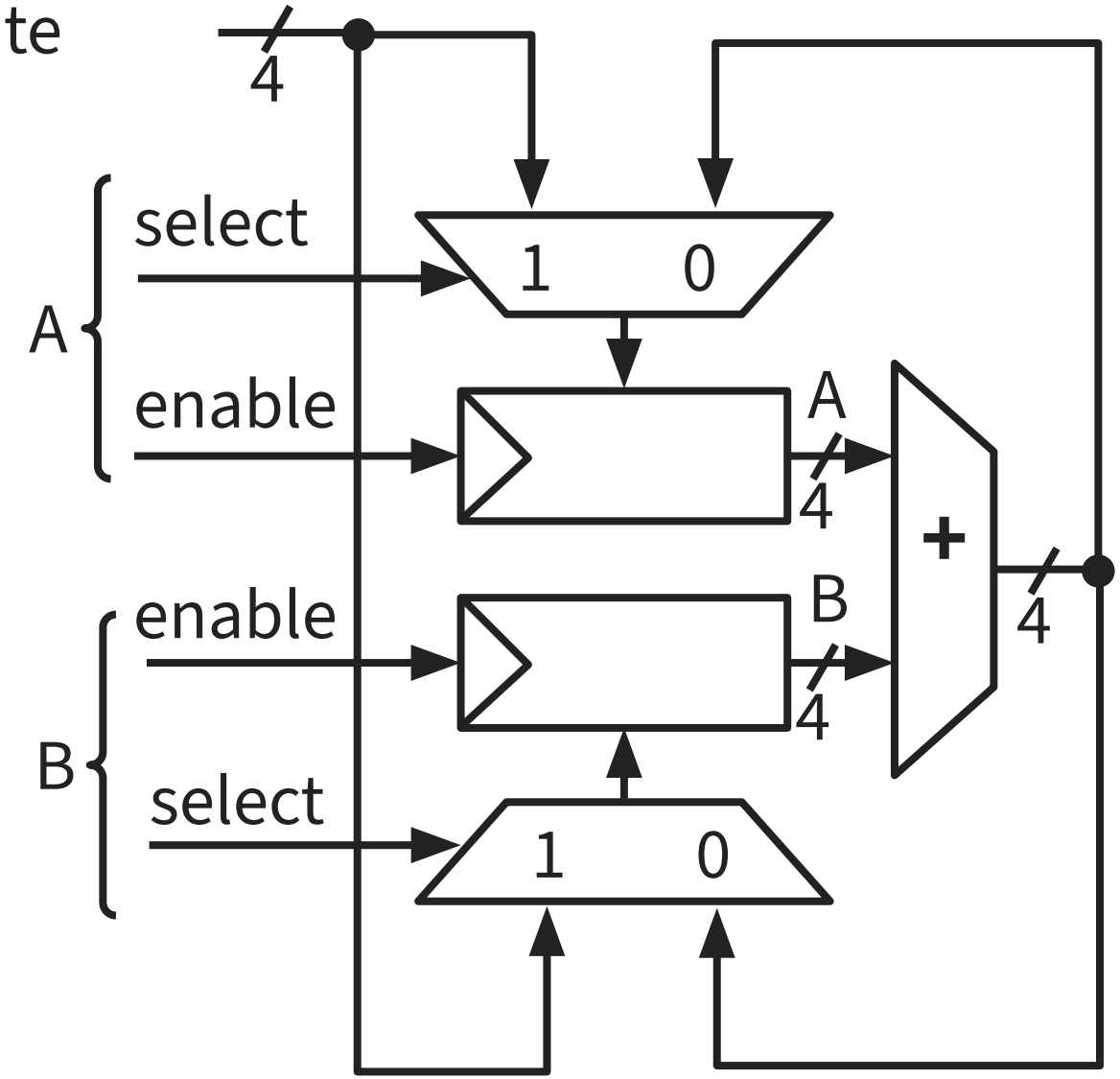
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic

```
11 0? 0000
0? 11 1000
10 0? ????
```



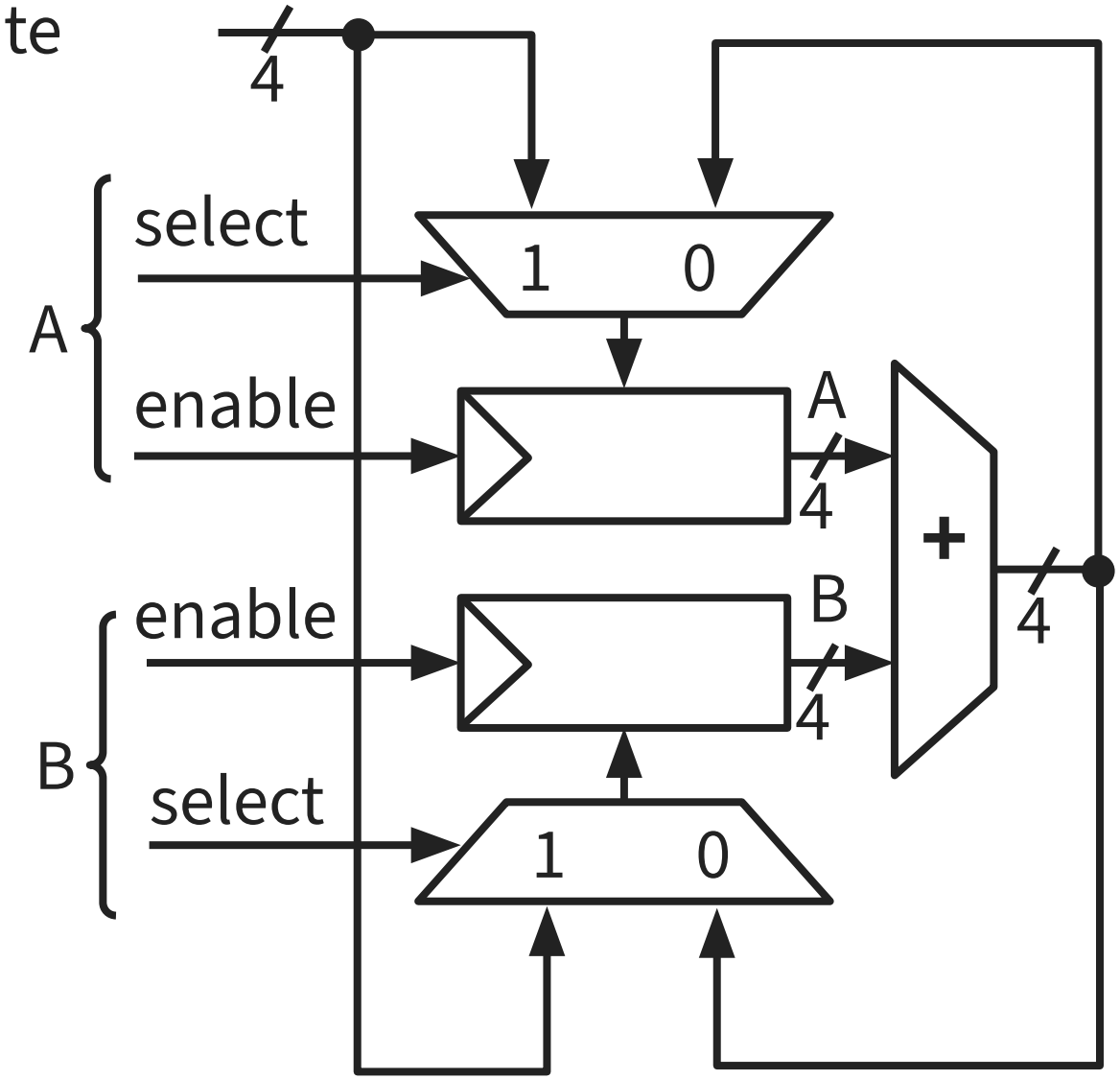
Instruction Format:

immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic

```
11 0? 0000 wrimm A, 0
0? 11 1000
10 0? ????
```

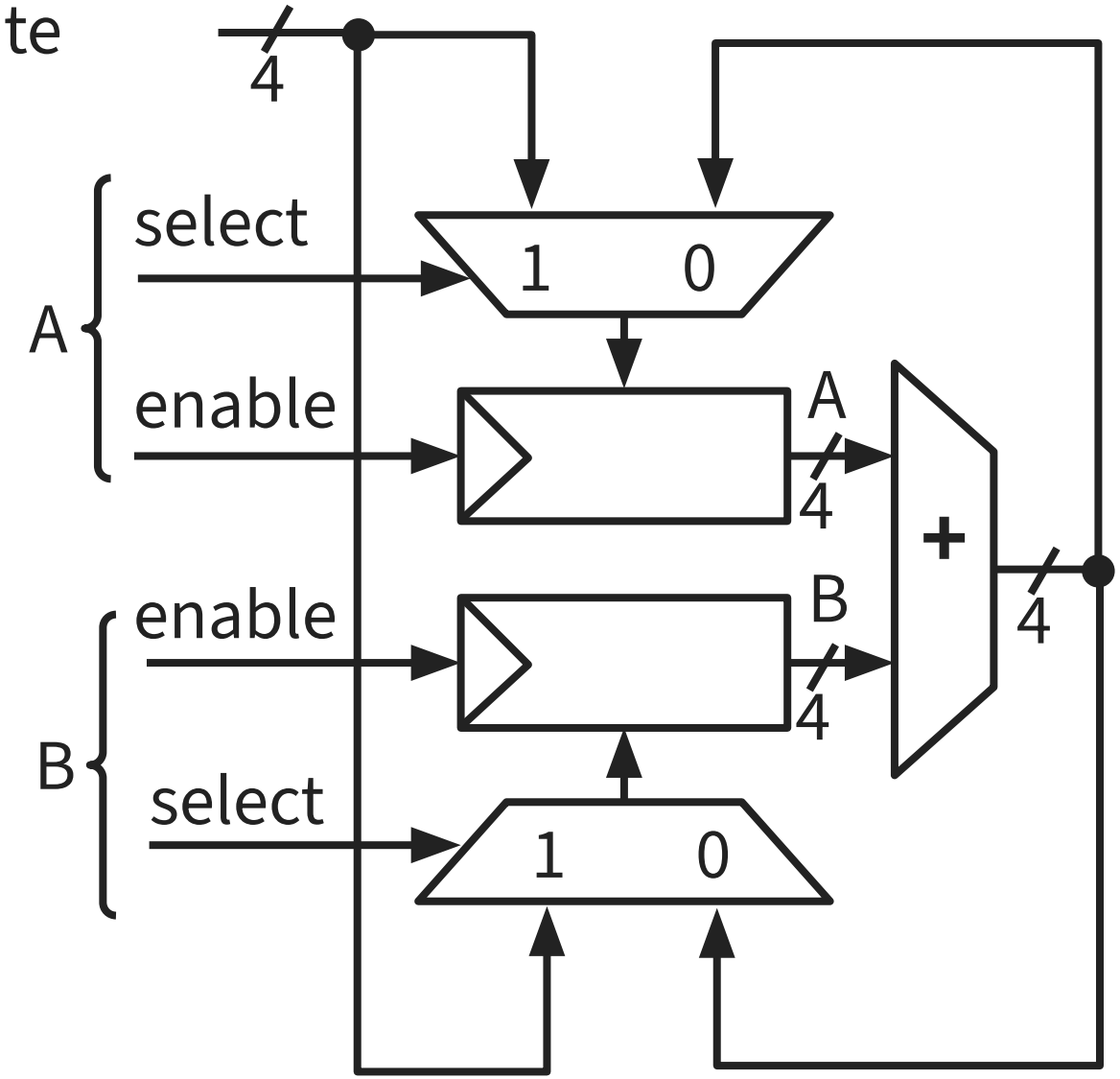


Instruction Format: immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic

```
11 0? 0000 wrimm A, 0
0? 11 1000 wrimm B, 8
10 0? ????
```

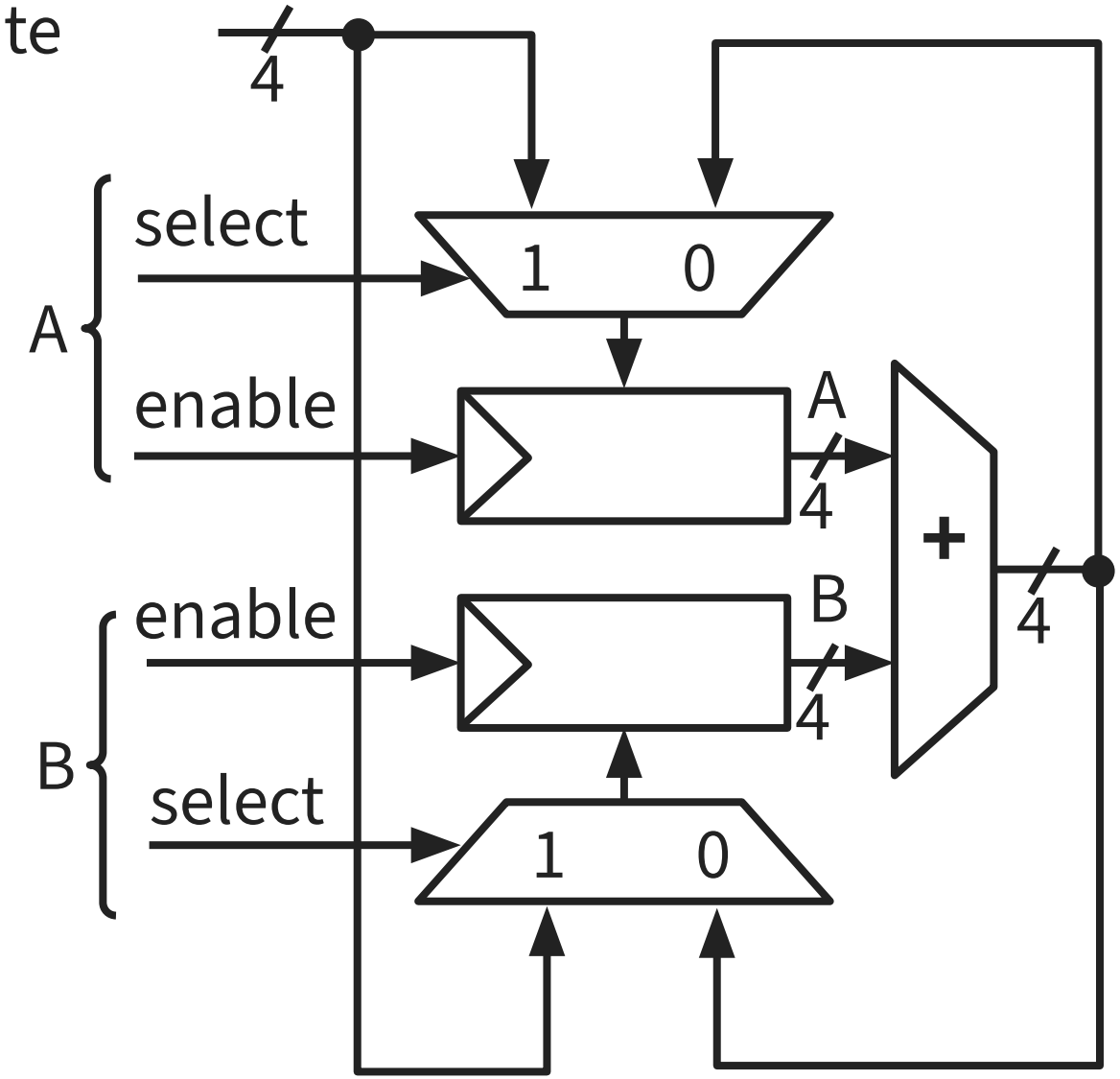


Instruction Format: immediate

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic

```
11 0? 0000 wrimm A, 0
0? 11 1000 wrimm B, 8
10 0? ???? add A
```



Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Assembly Language Manual:

	Mnemonic
11 0? 0000	wrimm A, 0
0? 11 1000	wrimm B, 8
10 0? ????	add A

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic

```
11 0? 0000 wrimm A, 0
0? 11 1000 wrimm B, 8
10 0? ???? add A
```

Assembly Language Manual:

`wrimm {A, B}, N:`

Write the value N to the specified register.

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Mnemonic	
11 0? 0000	wrimm A, 0
0? 11 1000	wrimm B, 8
10 0? ????	add A

Assembly Language Manual:

`wrimm {A, B}, N:`

Write the value N to the specified register.

`add {A, B}:`

Add the value of A and B and write the result into the specified register.

Instruction Format:

$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

		Mnemonic
11	0? 0000	wrimm A, 0
0?	11 1000	wrimm B, 8
10	0? ????	add A

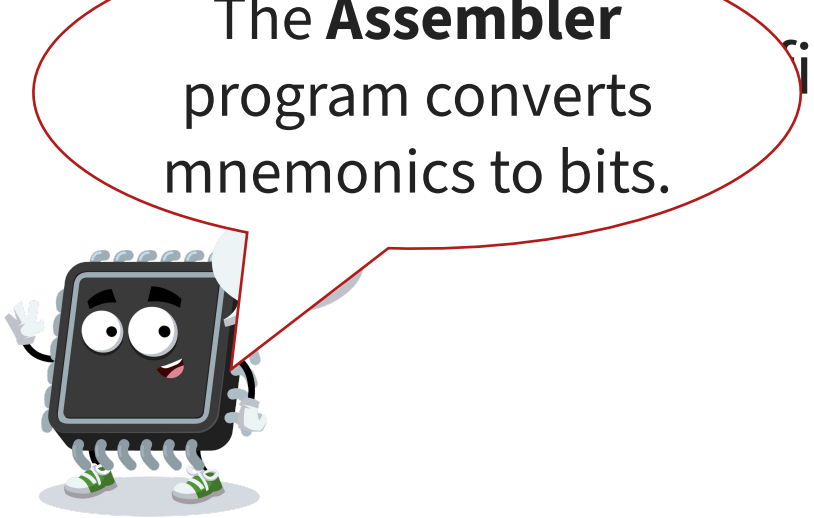
Assembly Language Manual:

`wrimm {A, B}, N:`

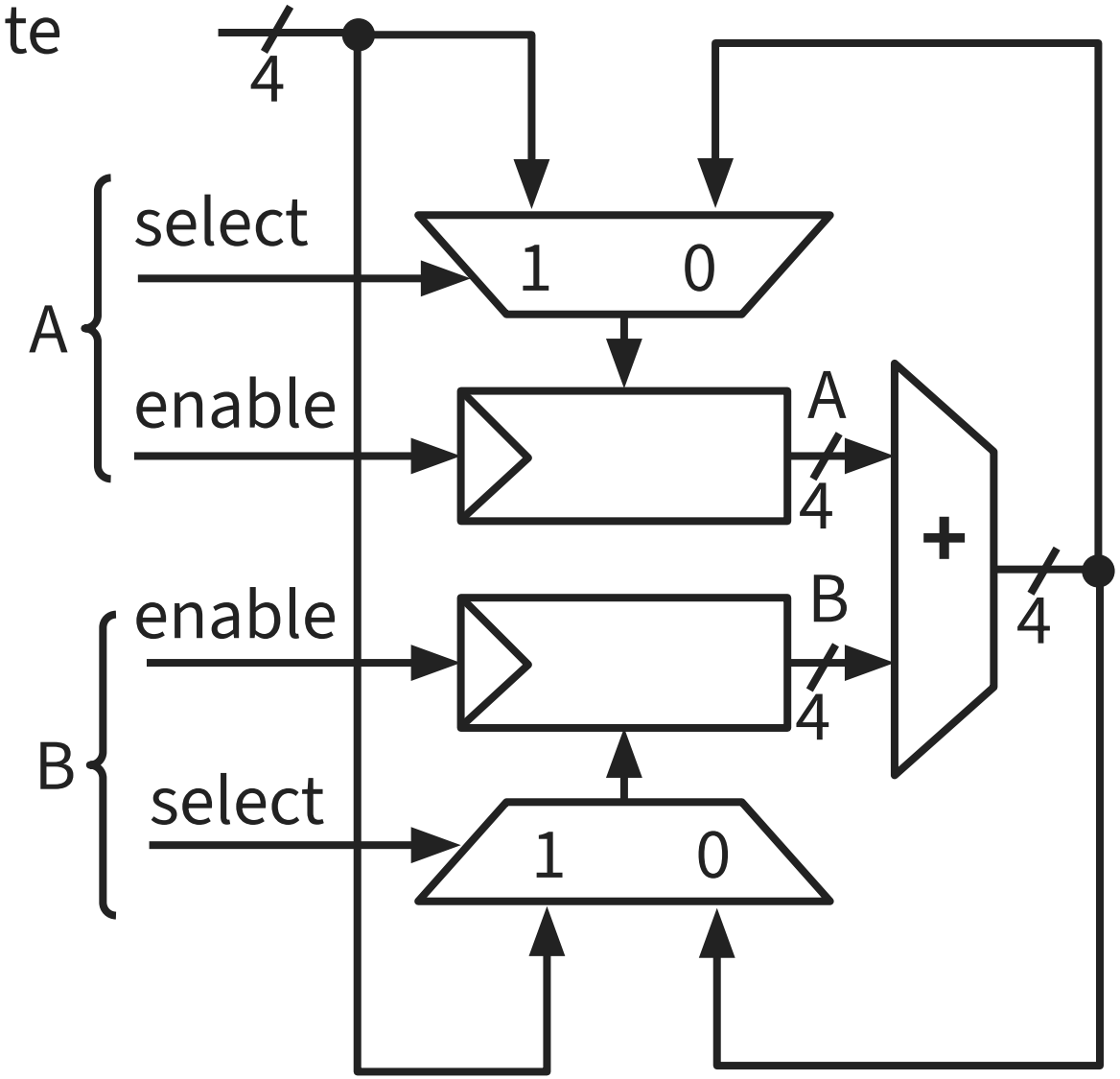
Write the value N to the specified register.

`add {A, B}:`

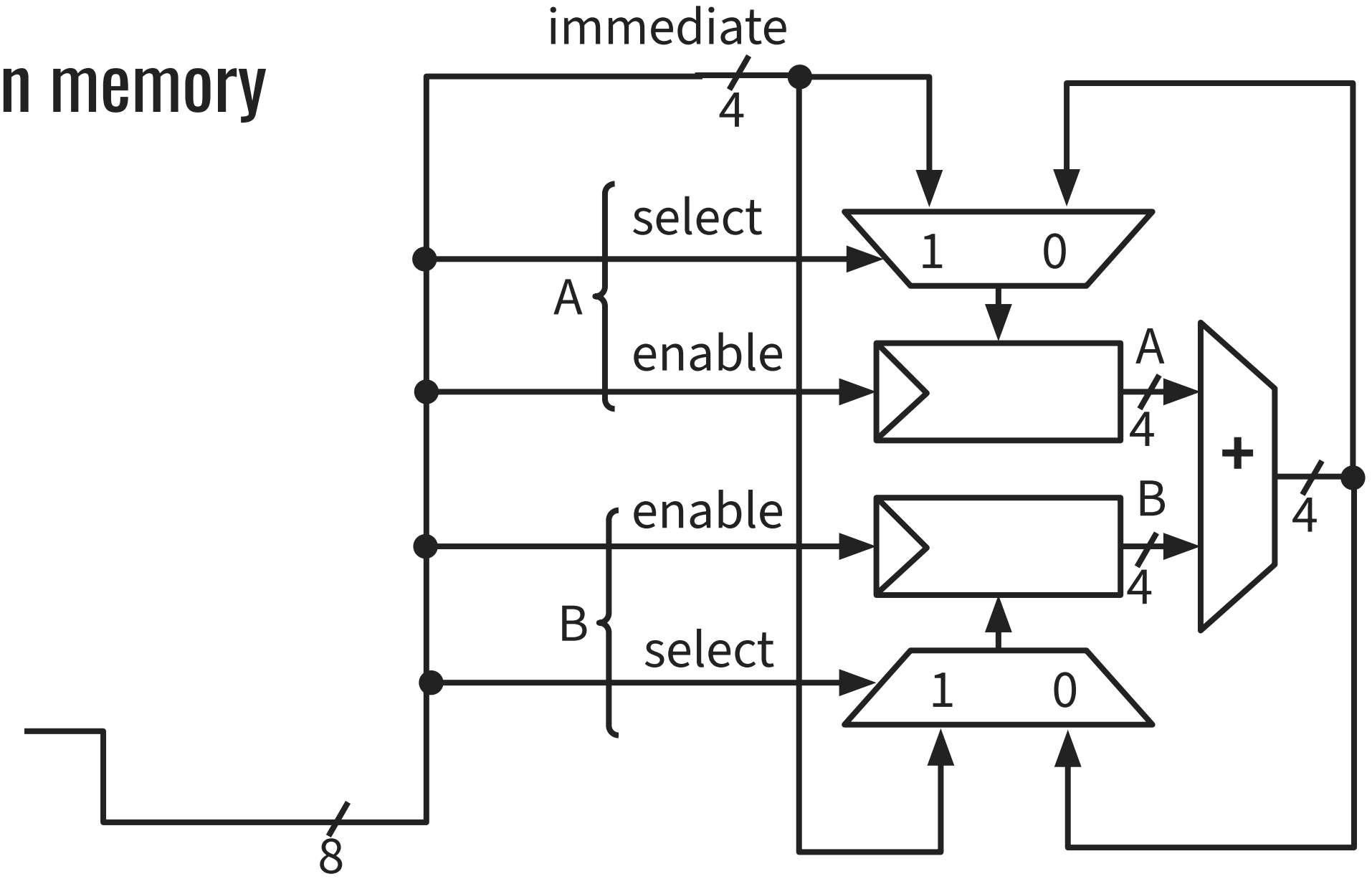
Add th[illegible] of A and B and write [illegible]ied register.



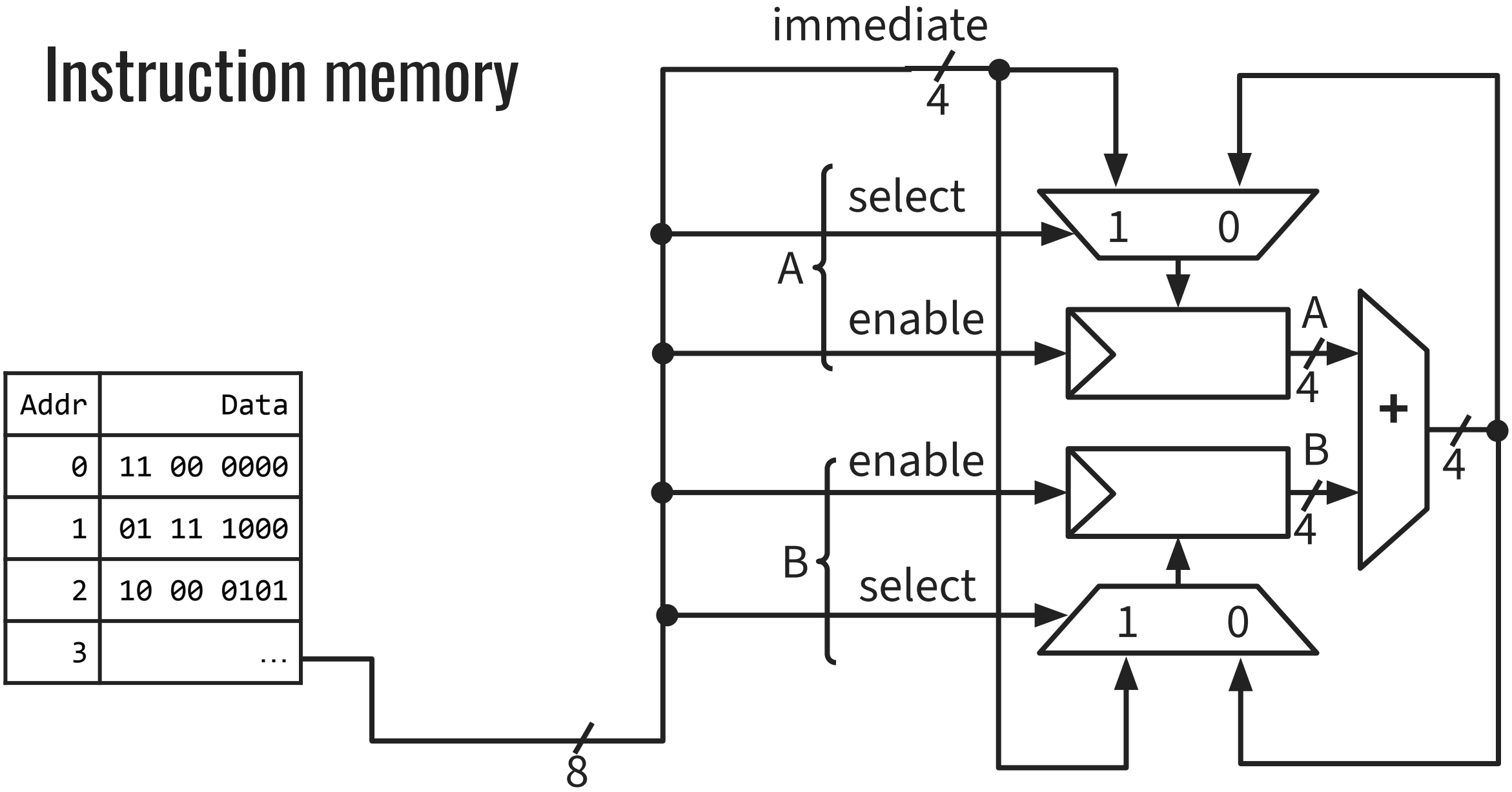
Instruction memory



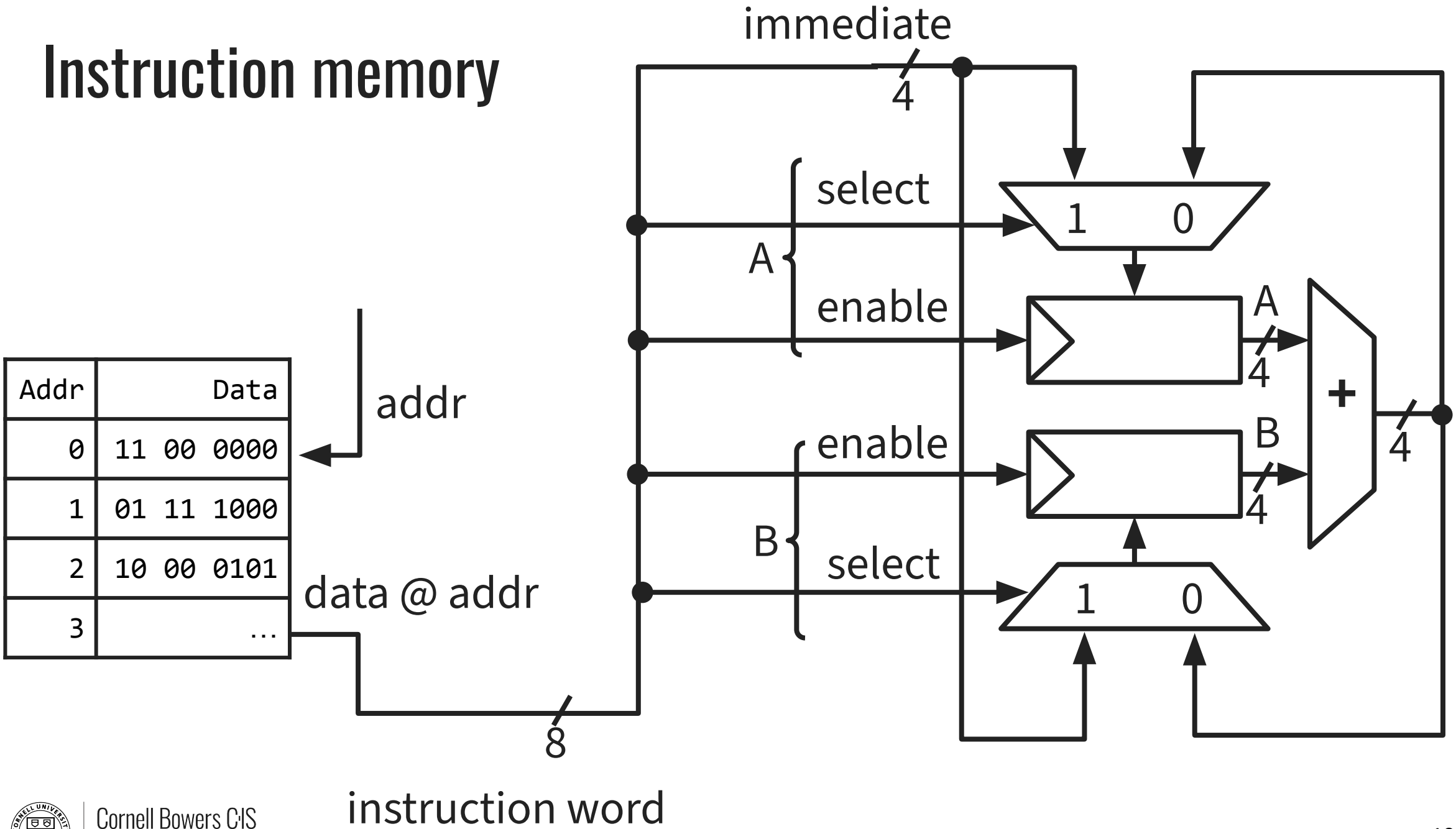
Instruction memory



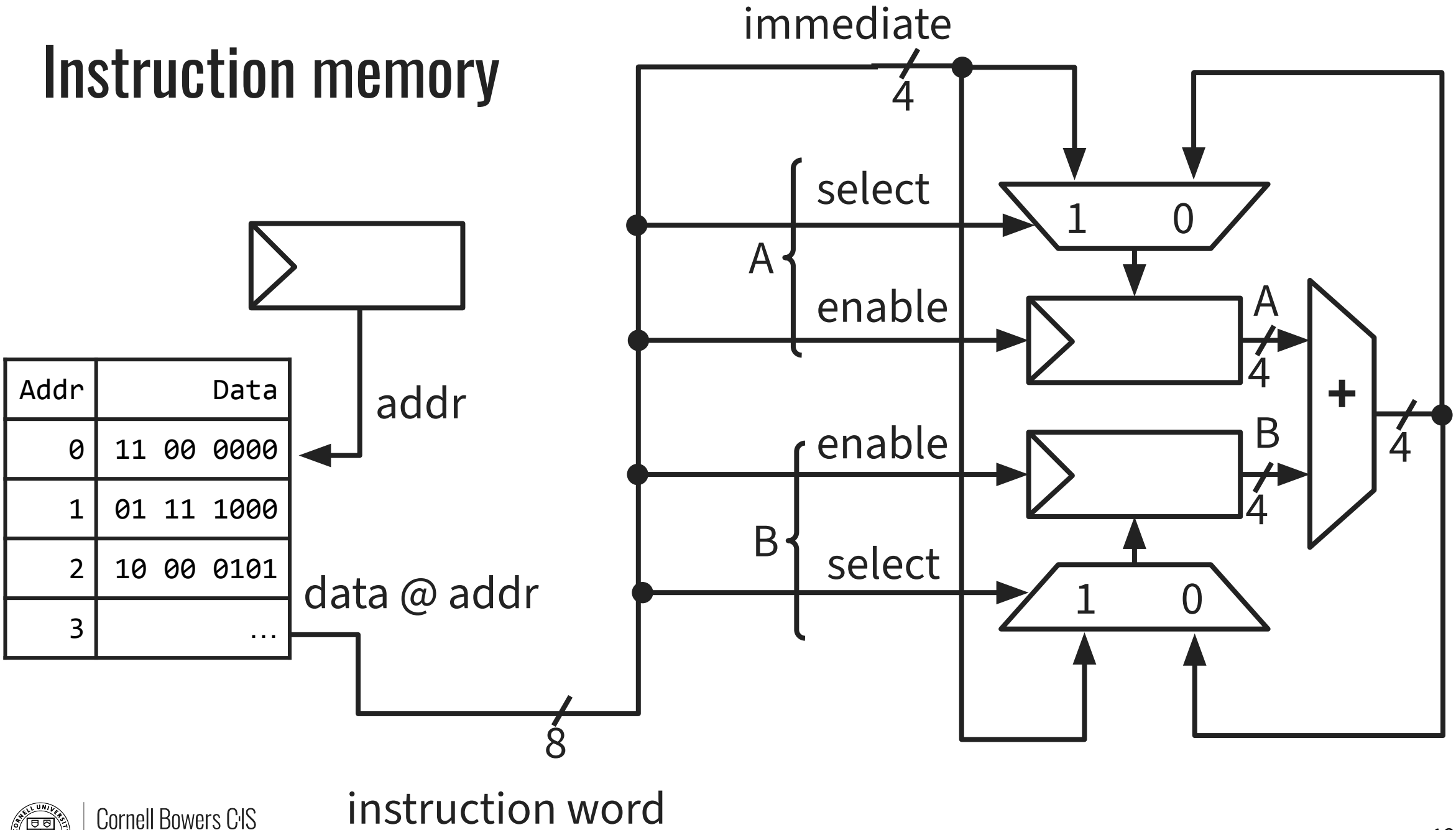
Instruction memory



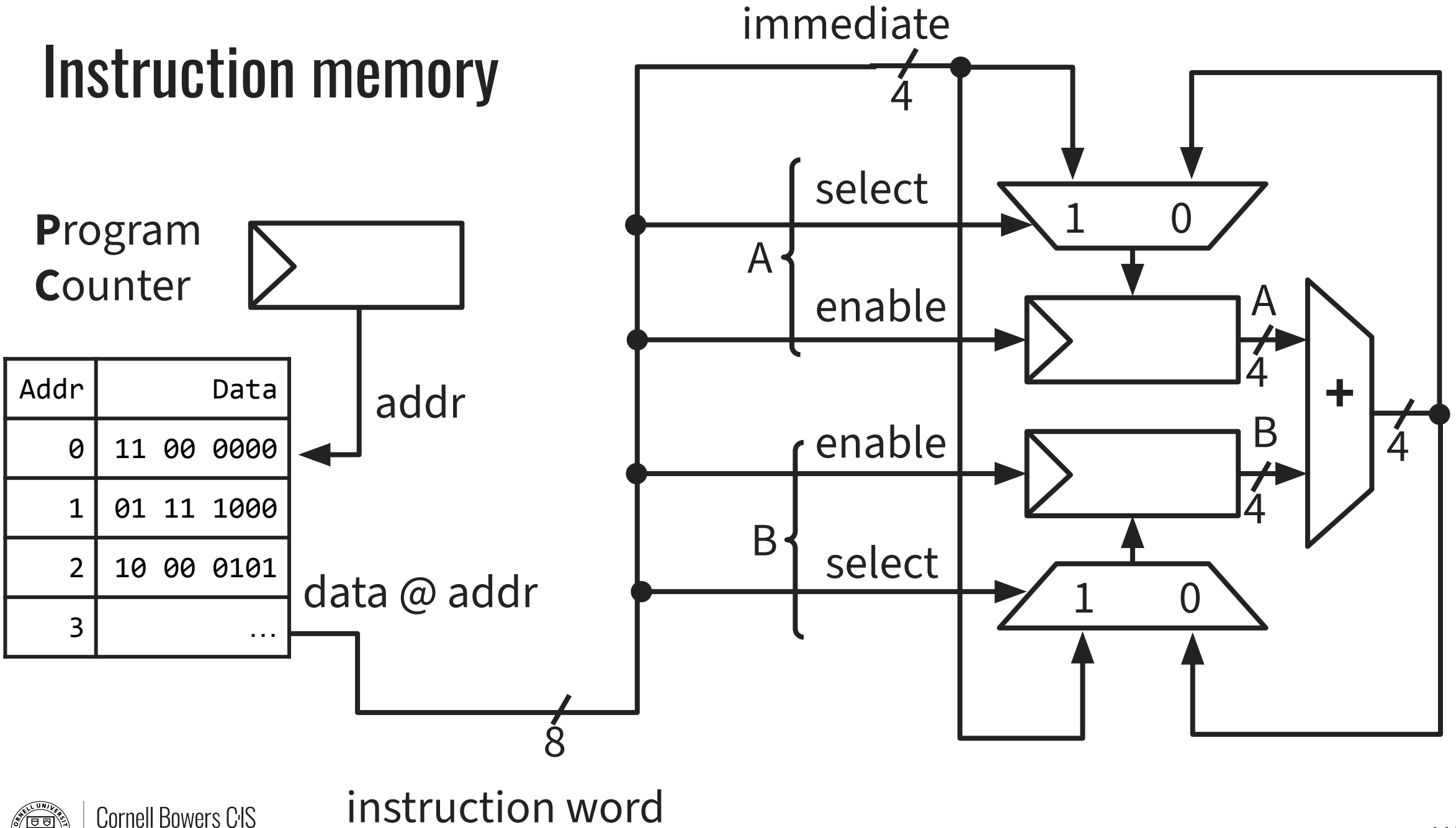
Instruction memory



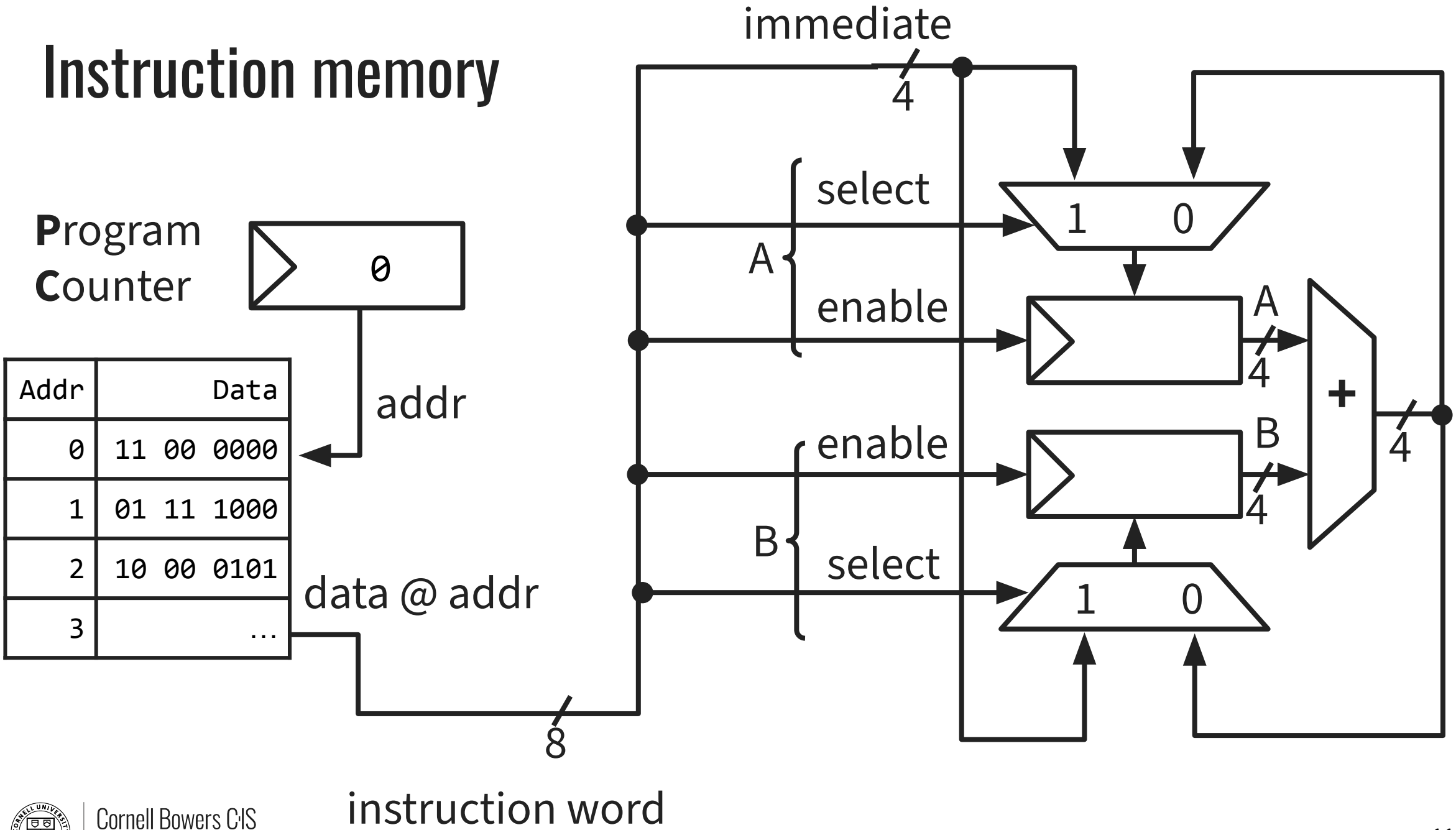
Instruction memory



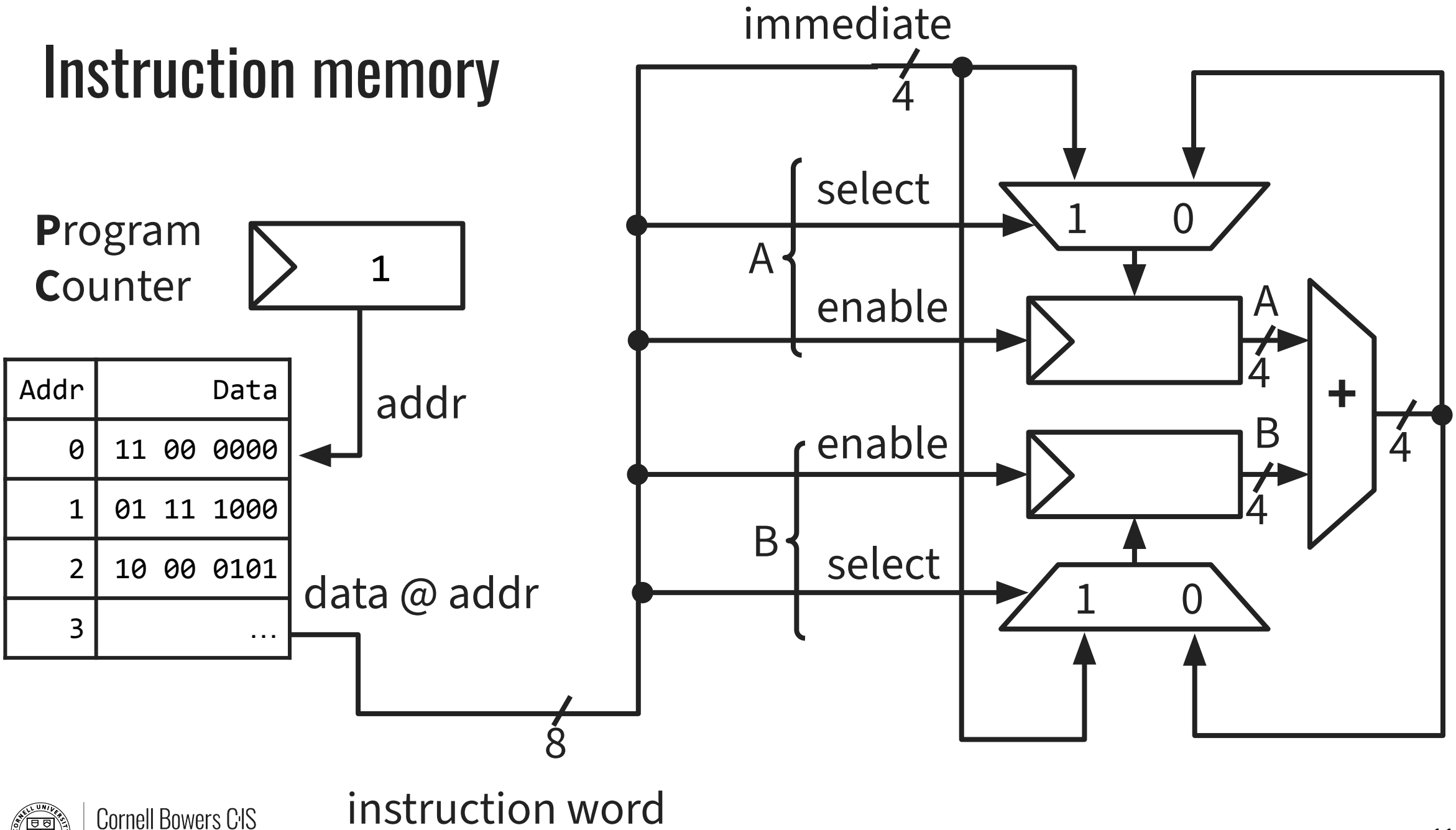
Instruction memory



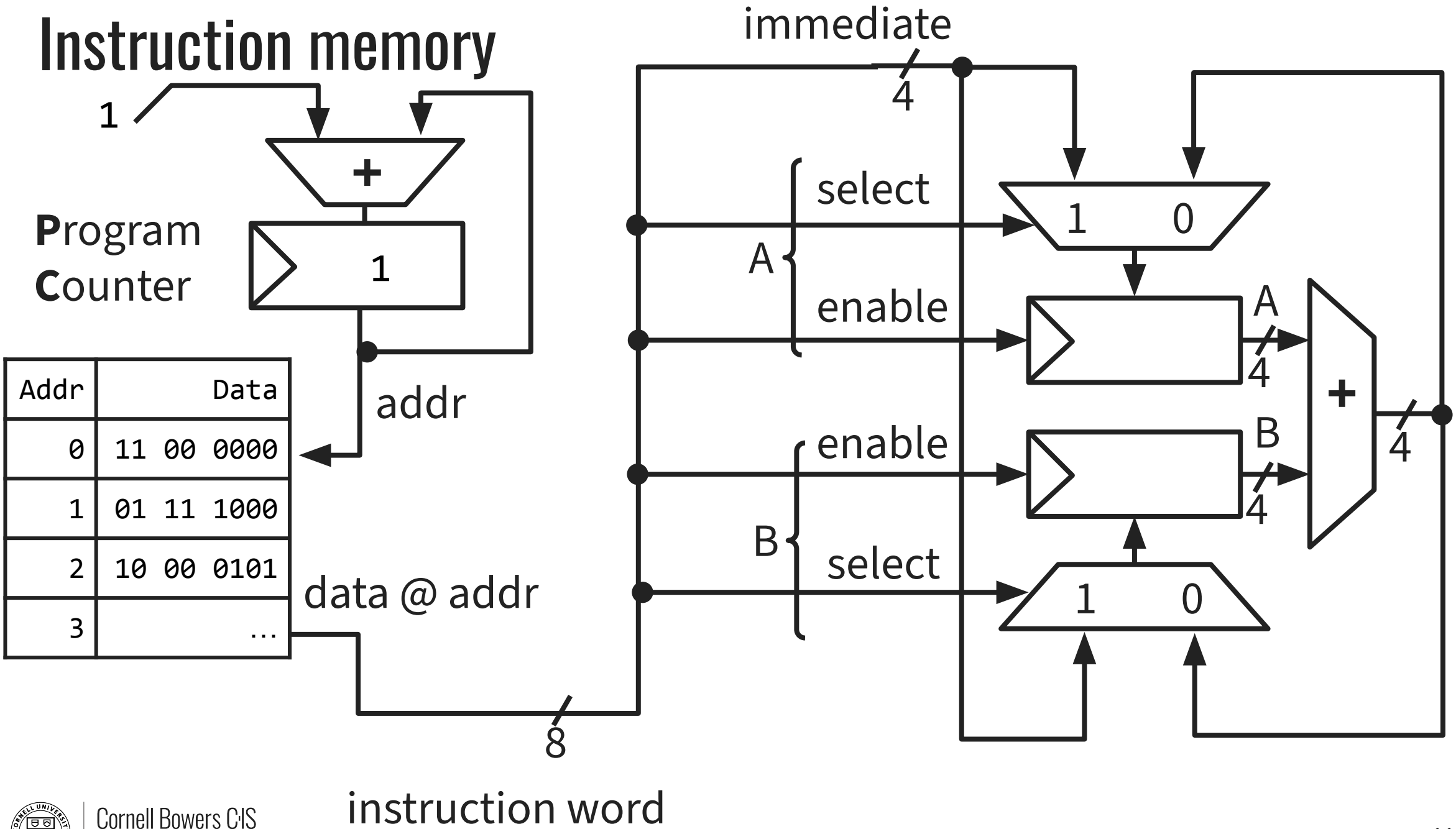
Instruction memory



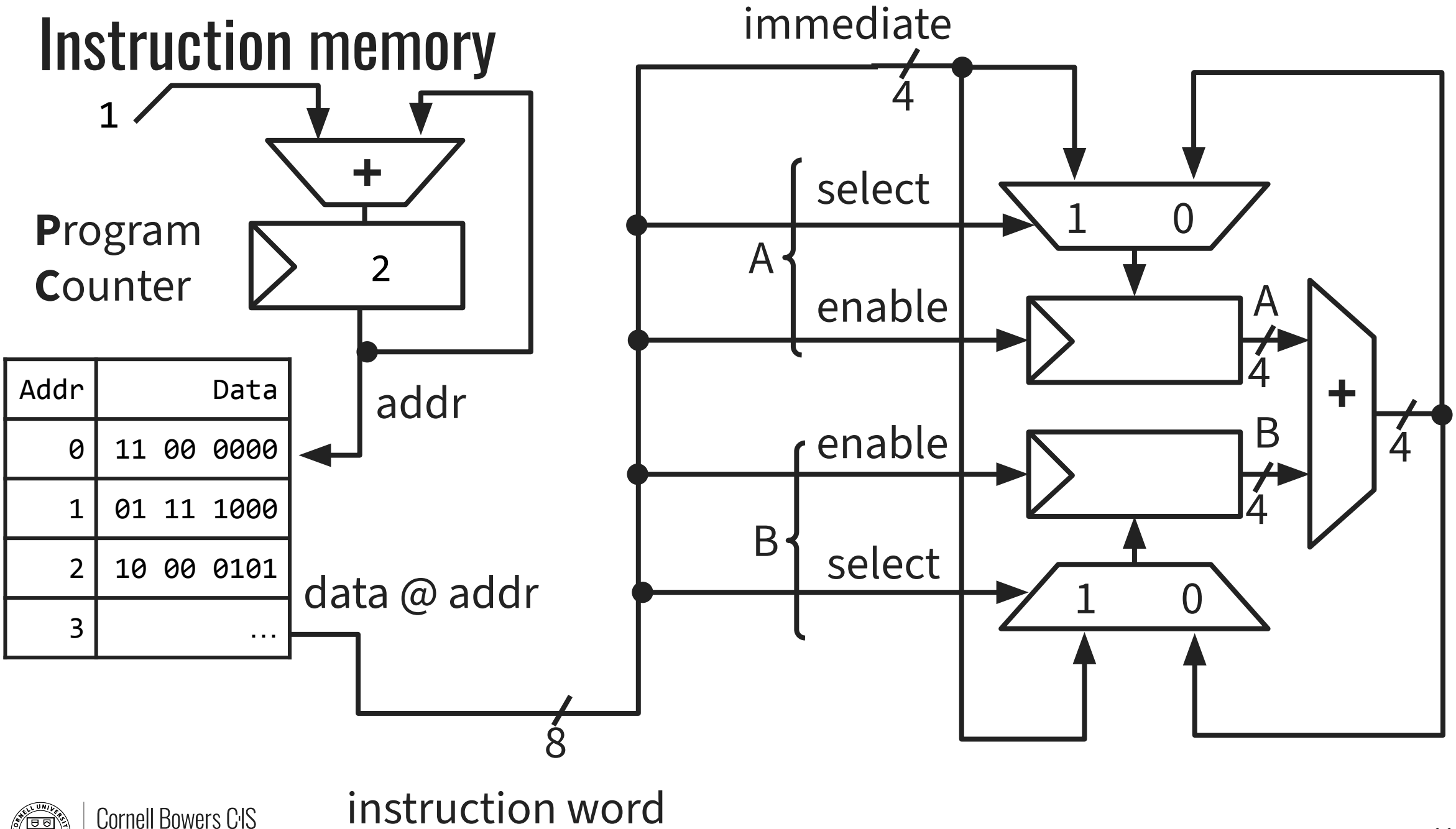
Instruction memory



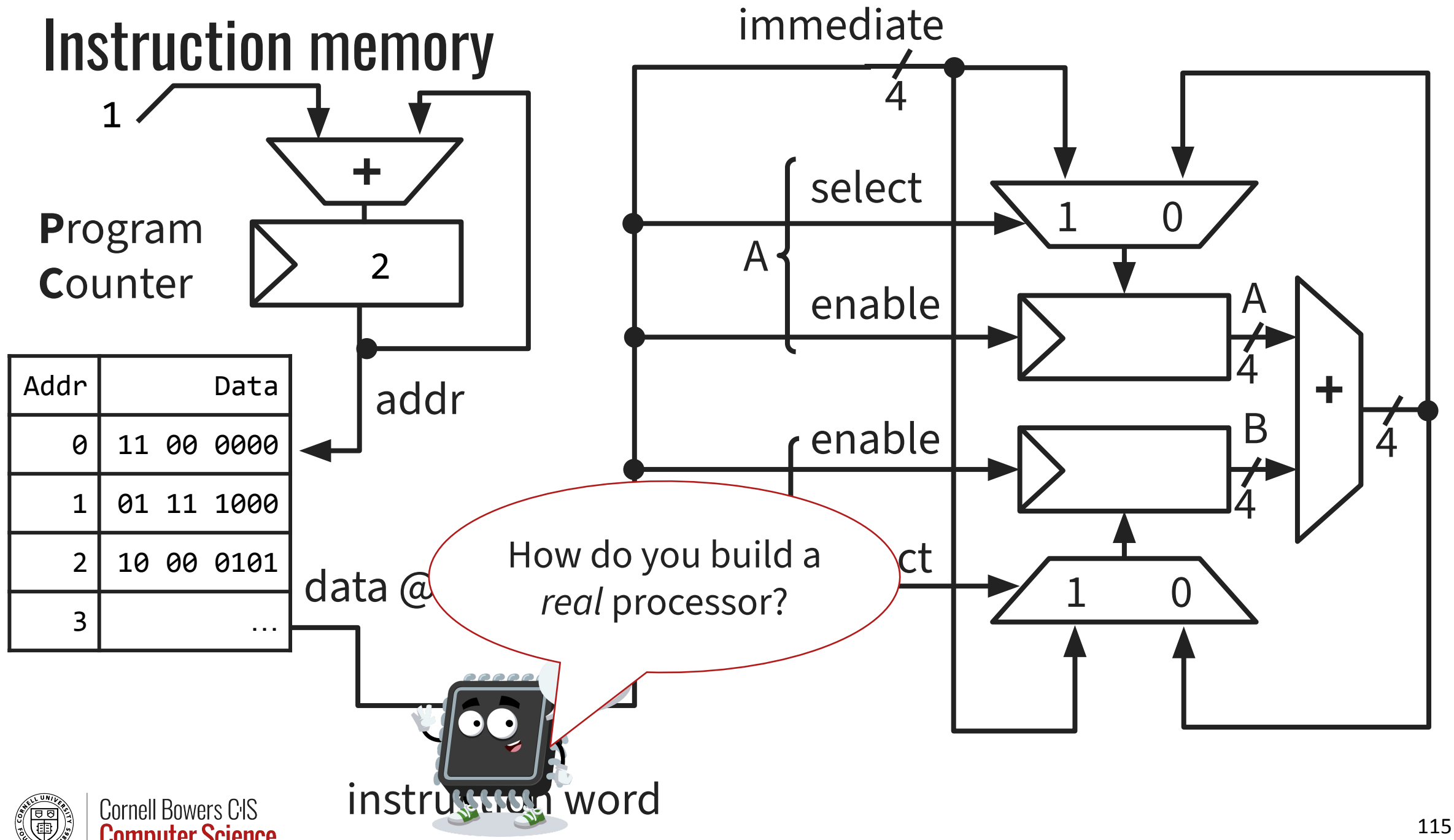
Instruction memory

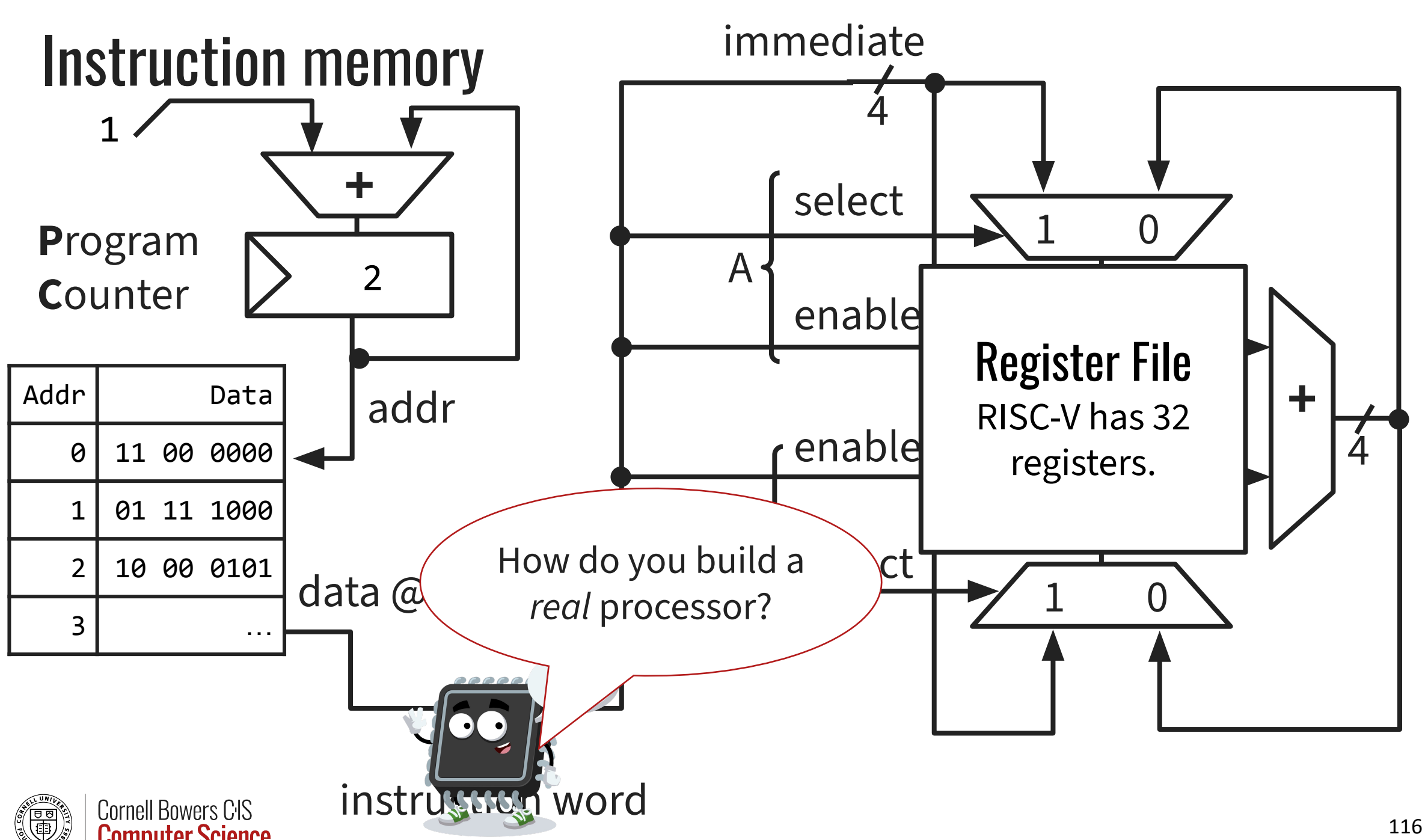


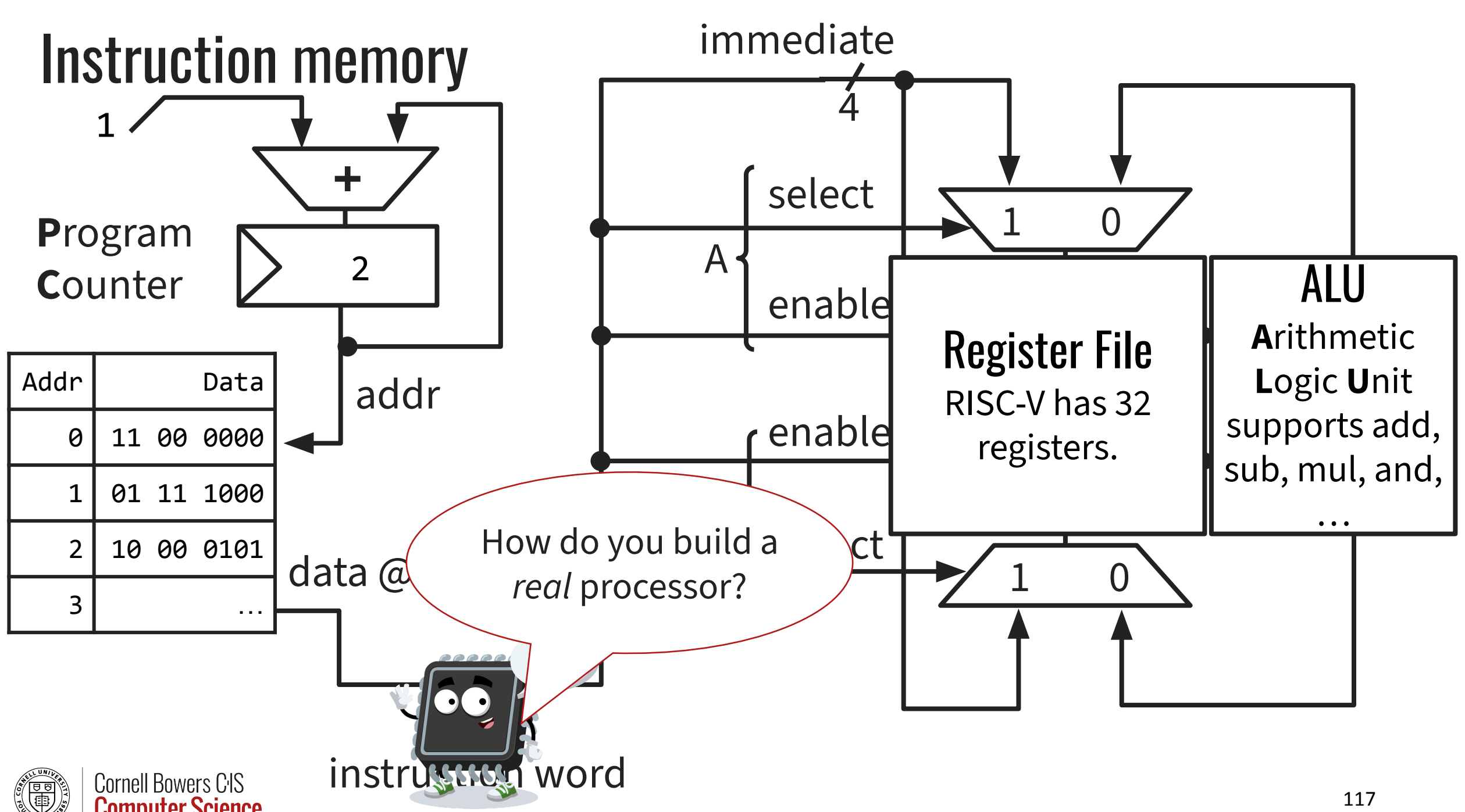
Instruction memory

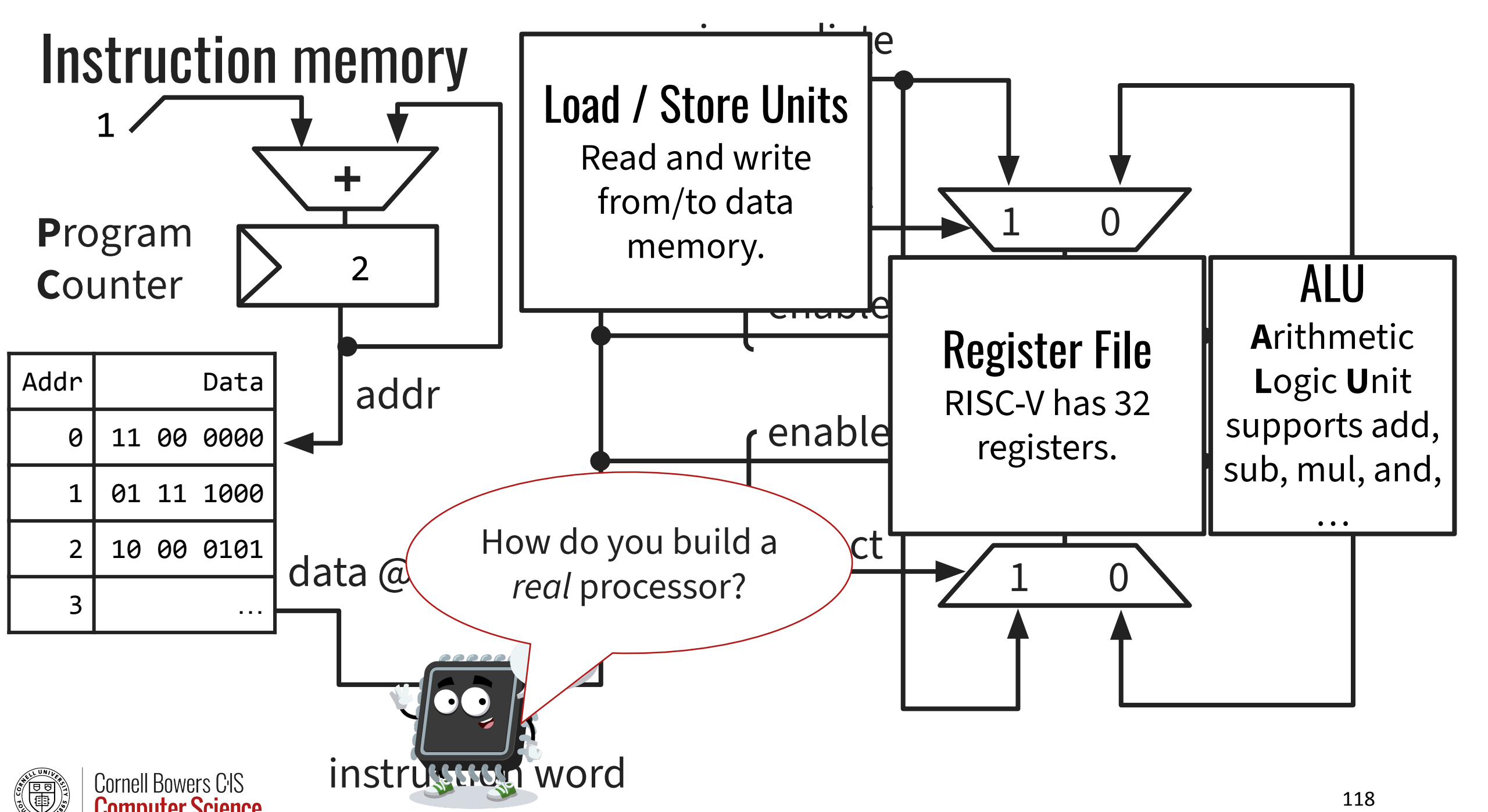


Instruction memory




Instruction memory
1
+
Program Counter
2
addr
Addr
Data
0
11 00 0000
1
01 11 1000
2
10 00 0101
3
…
data @
instruction word
How do you build a *real* processor?
immediate
4
select
A
enable
enable
1
0
Register File
RISC-V has 32 registers.
+
4
ct
1
0


Instruction memory
1
+
Program Counter
2
addr
Addr
Data
0
11 00 0000
1
01 11 1000
2
10 00 0101
3
…
data @
instruction word
How do you build a *real* processor?
immediate
4
select
A
enable
enable
1
0
Register File
RISC-V has 32 registers.
ALU
Arithmetic Logic Unit supports add, sub, mul, and, …
ct
1
0


Instruction memory
1
+
Program Counter
2
addr
Addr
Data
0
11 00 0000
1
01 11 1000
2
10 00 0101
3
…
data @
instruction word
Load / Store Units
Read and write from/to data memory.
enable
1
0
Register File
RISC-V has 32 registers.
ALU
Arithmetic Logic Unit supports add, sub, mul, and, …
How do you build a *real* processor?

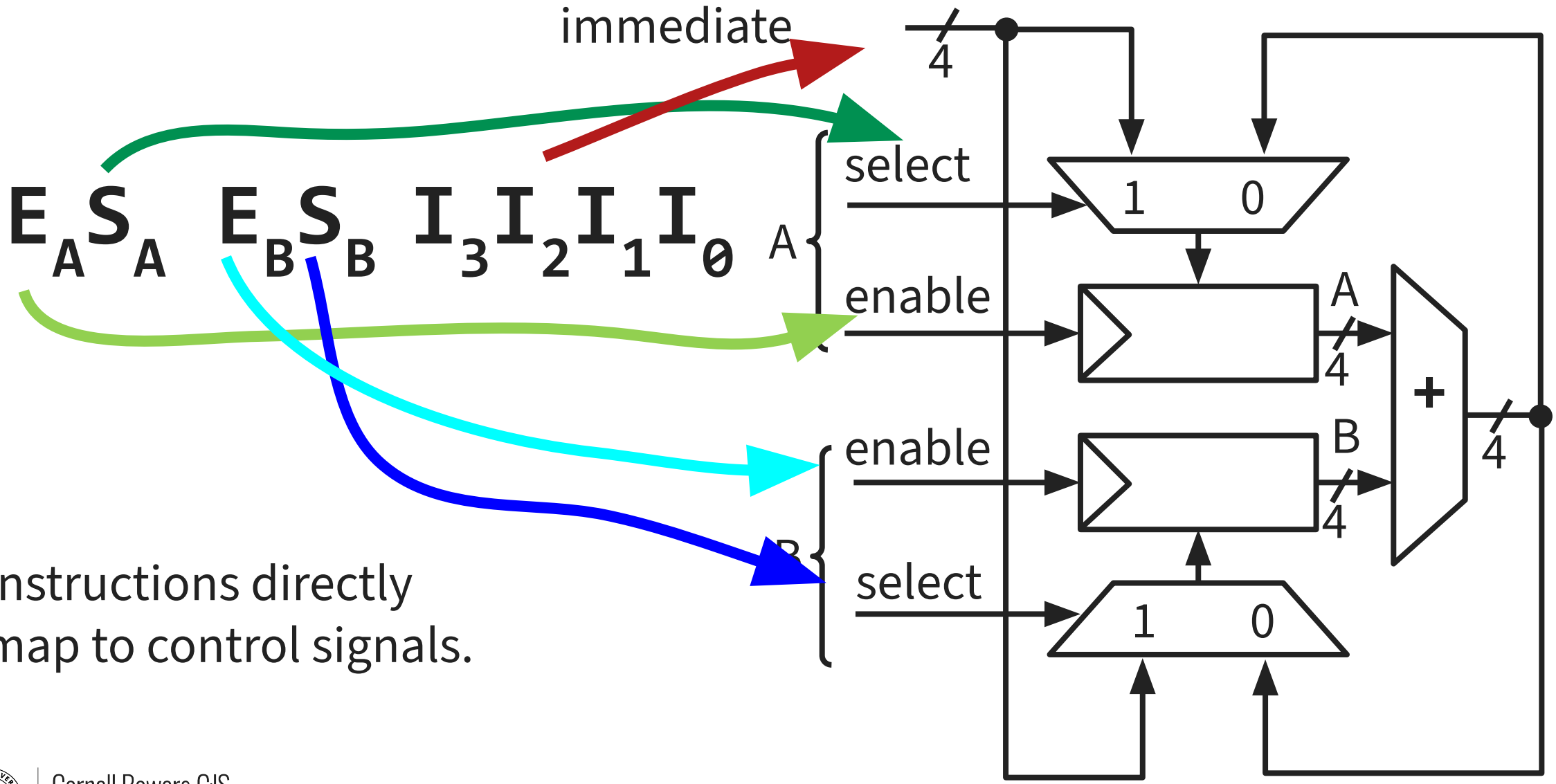
RISC-V

CS 3410: Computer System Organization and Programming

Fall 2025

RISC-V, ARM, x86 — what is the difference?

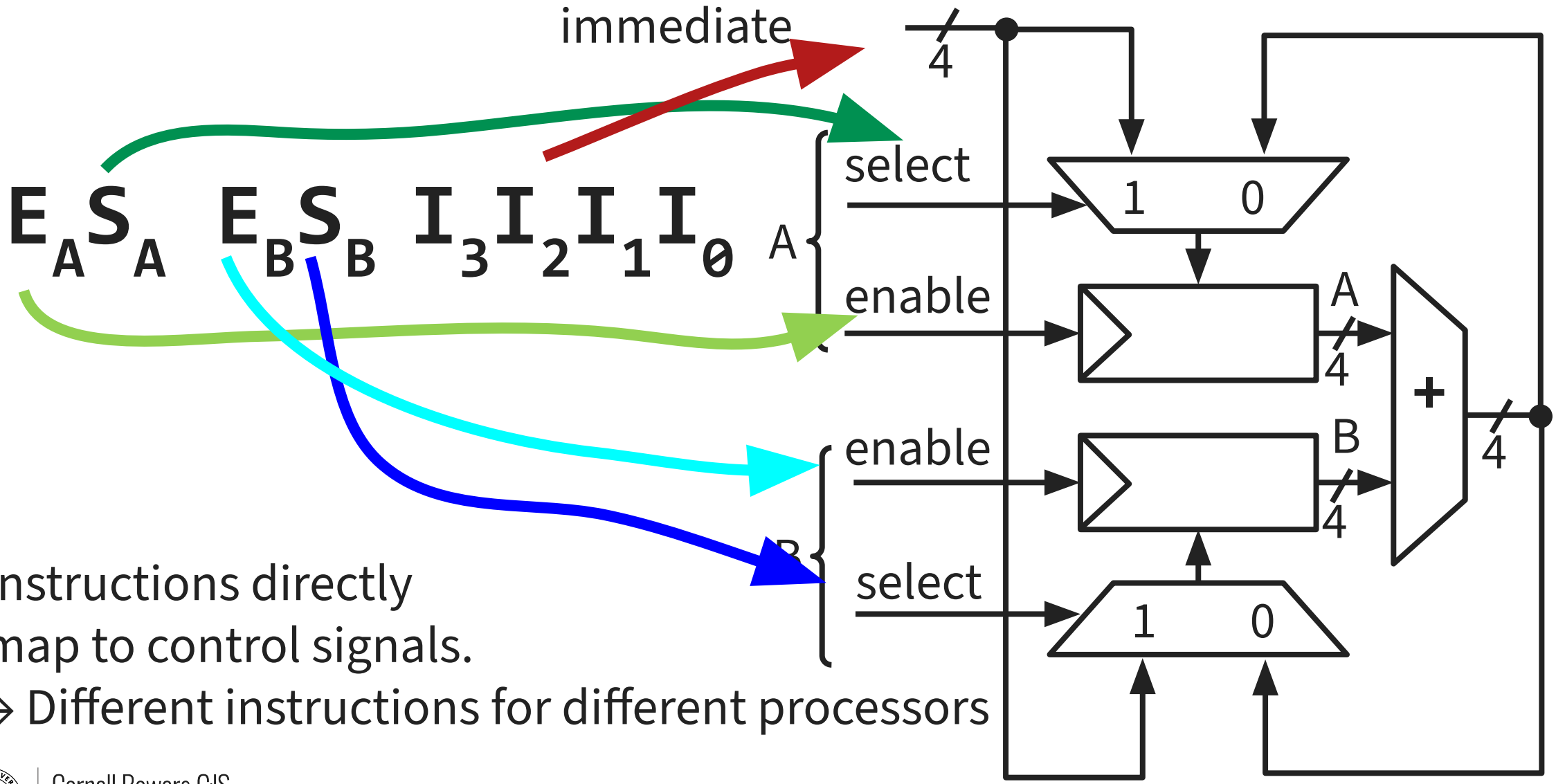
RISC-V, ARM, x86 — what is the difference?



$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Instructions directly
map to control signals.

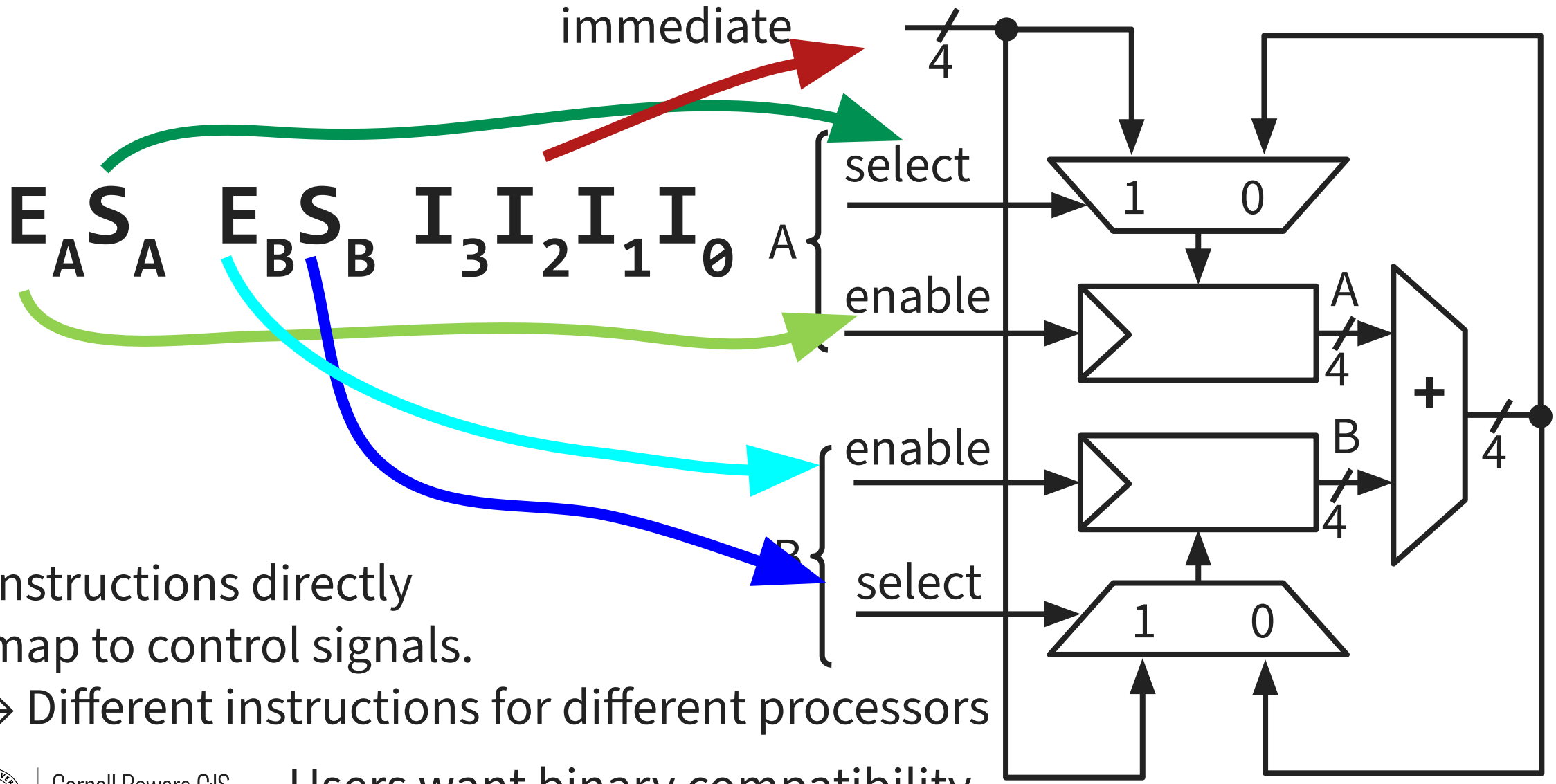
RISC-V, ARM, x86 — what is the difference?



$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Instructions directly
map to control signals.
→ Different instructions for different processors

RISC-V, ARM, x86 — what is the difference?

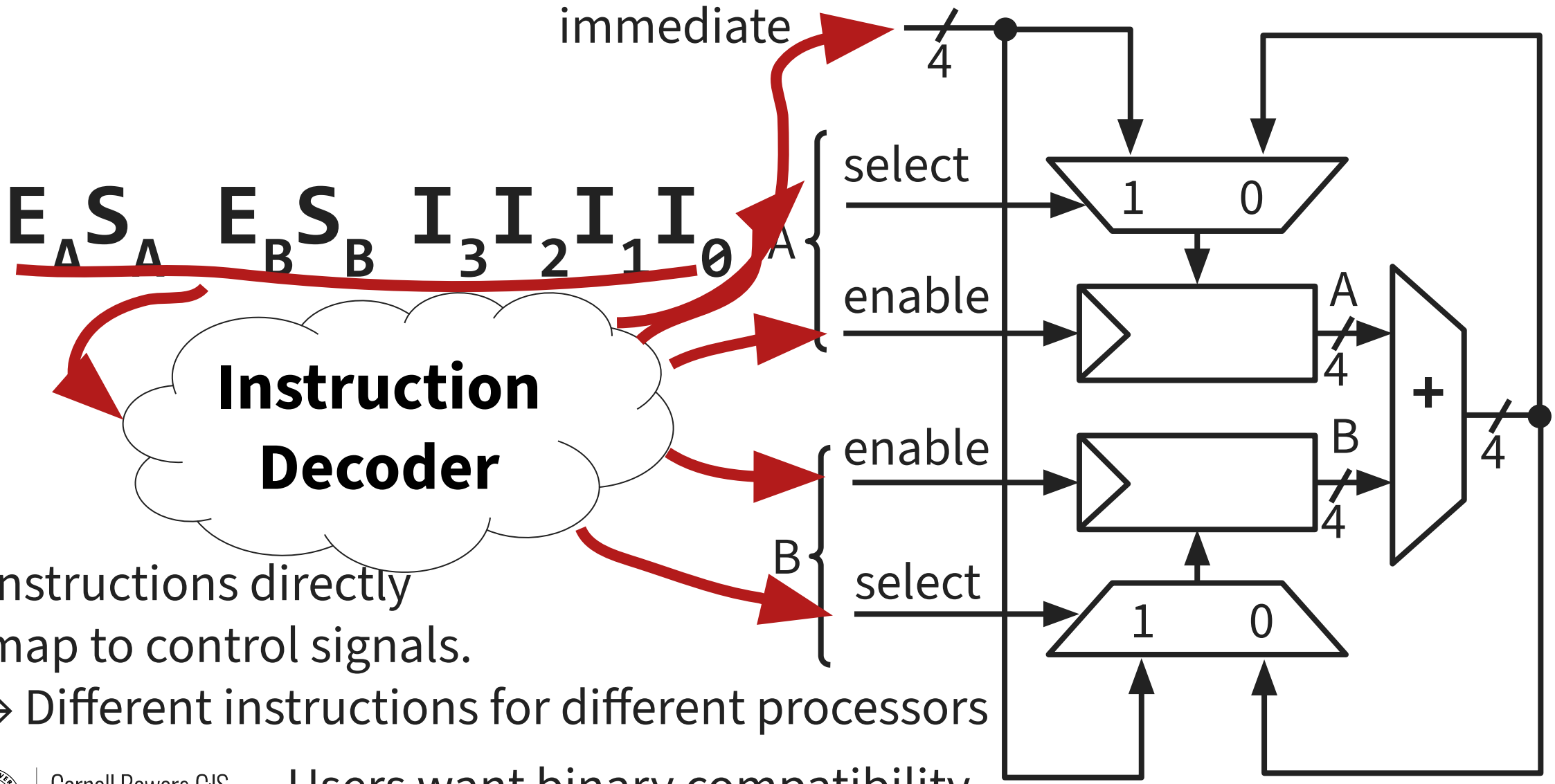


$E_A S_A \; E_B S_B \; I_3 I_2 I_1 I_0$

Instructions directly
map to control signals.
→ Different instructions for different processors

Users want binary compatibility.

RISC-V, ARM, x86 — what is the difference?

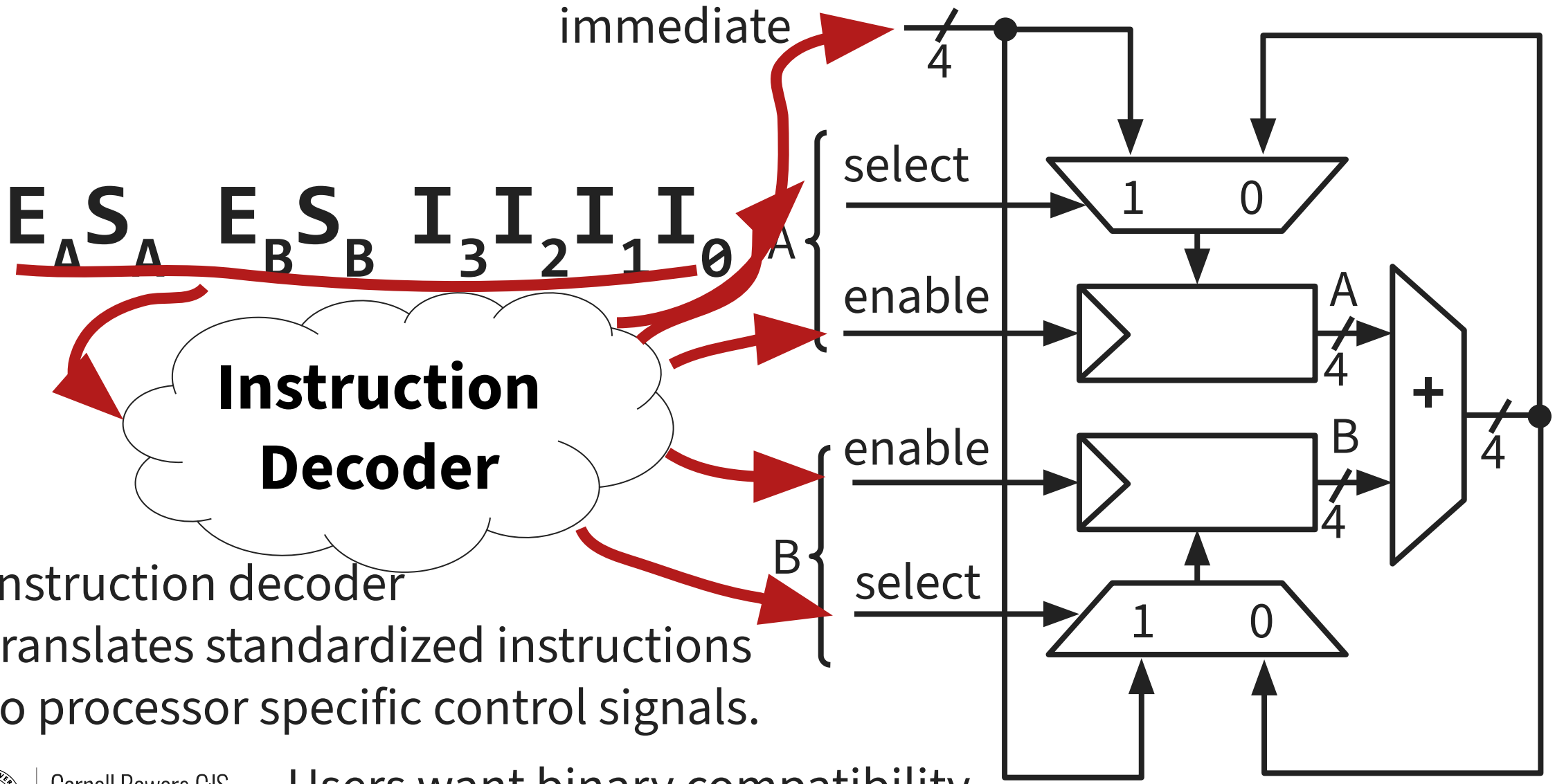


Instructions directly map to control signals.

→ Different instructions for different processors

Users want binary compatibility.

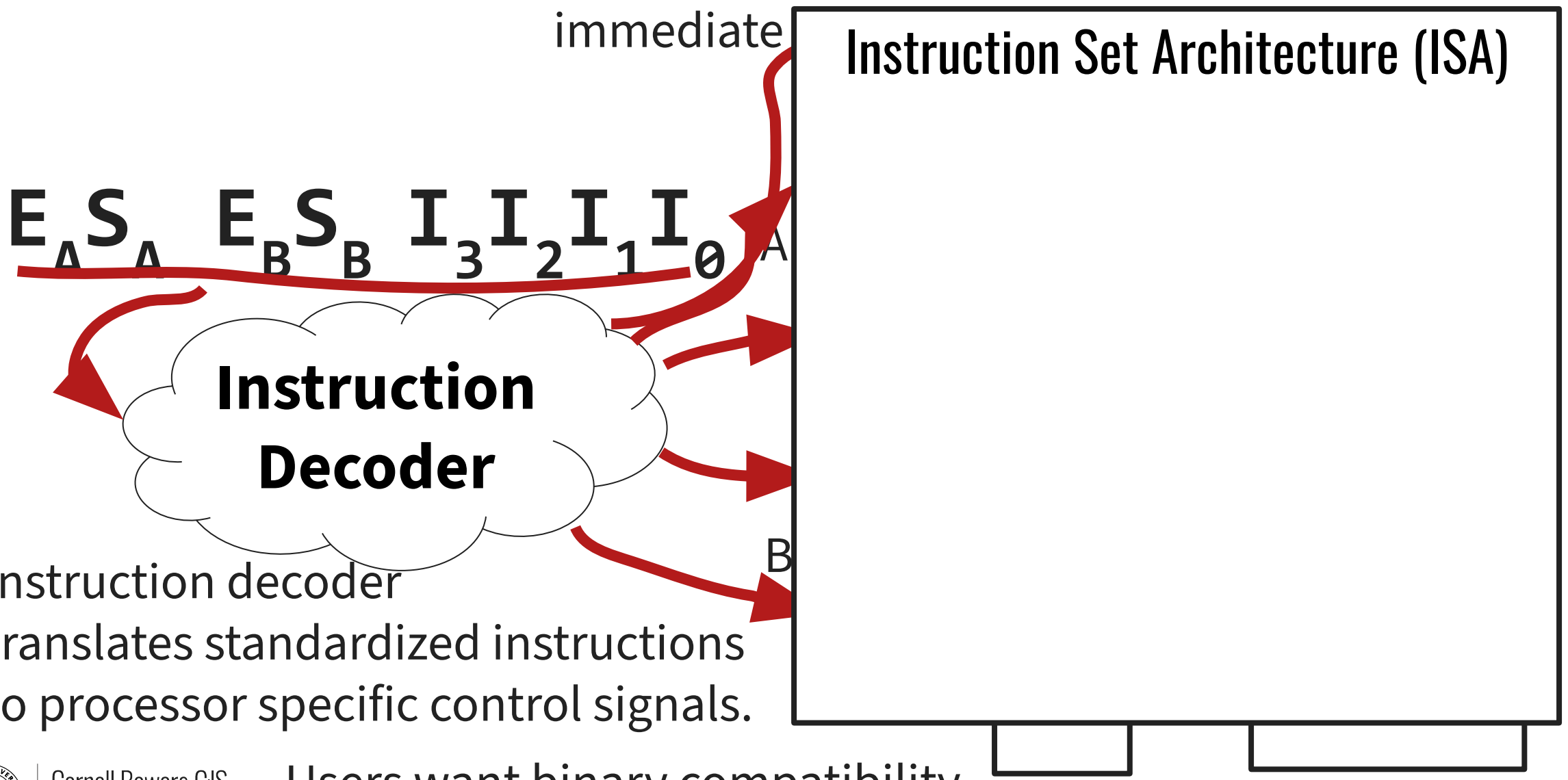
RISC-V, ARM, x86 — what is the difference?



Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

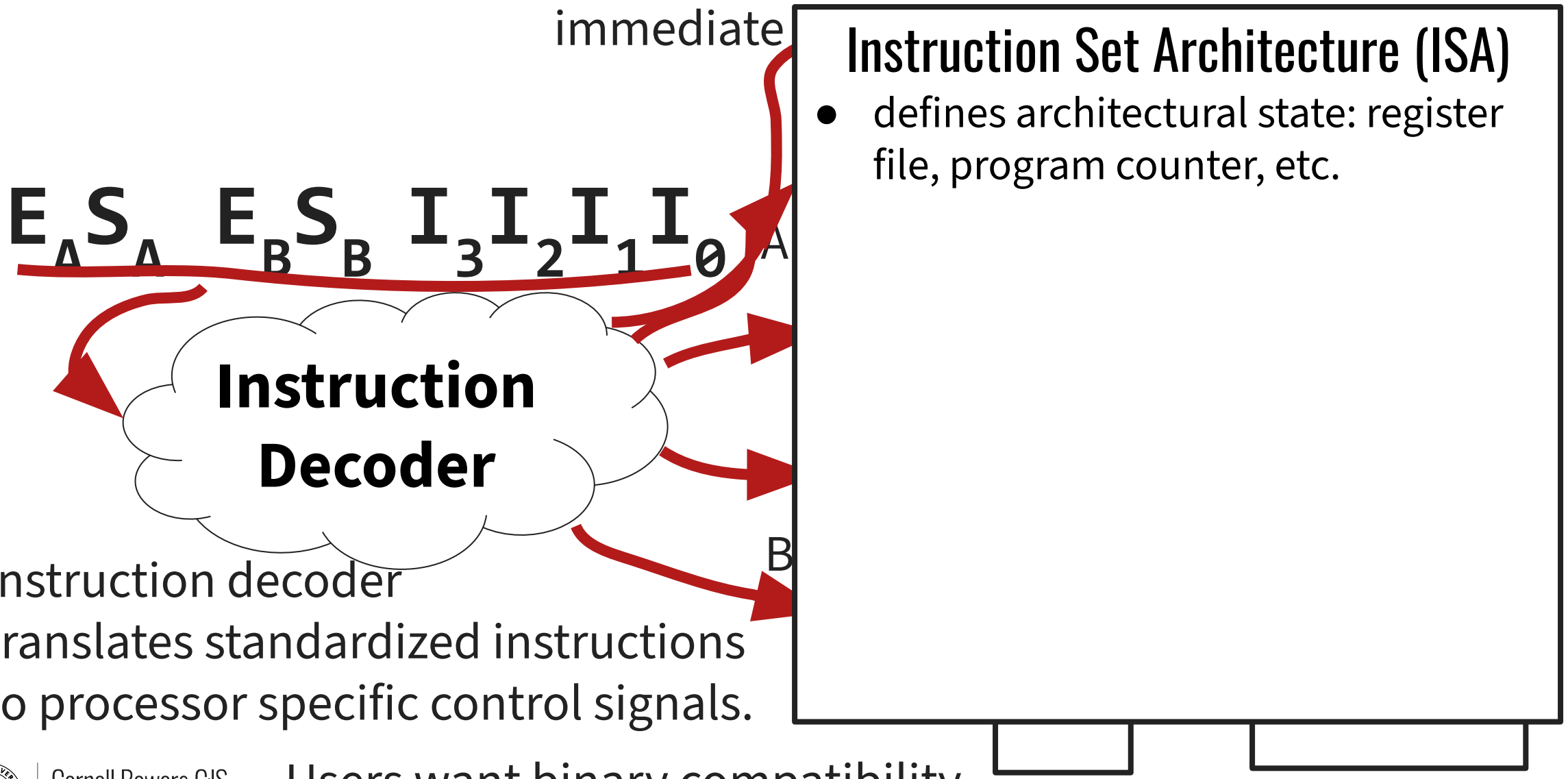
RISC-V, ARM, x86 — what is the difference?



Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

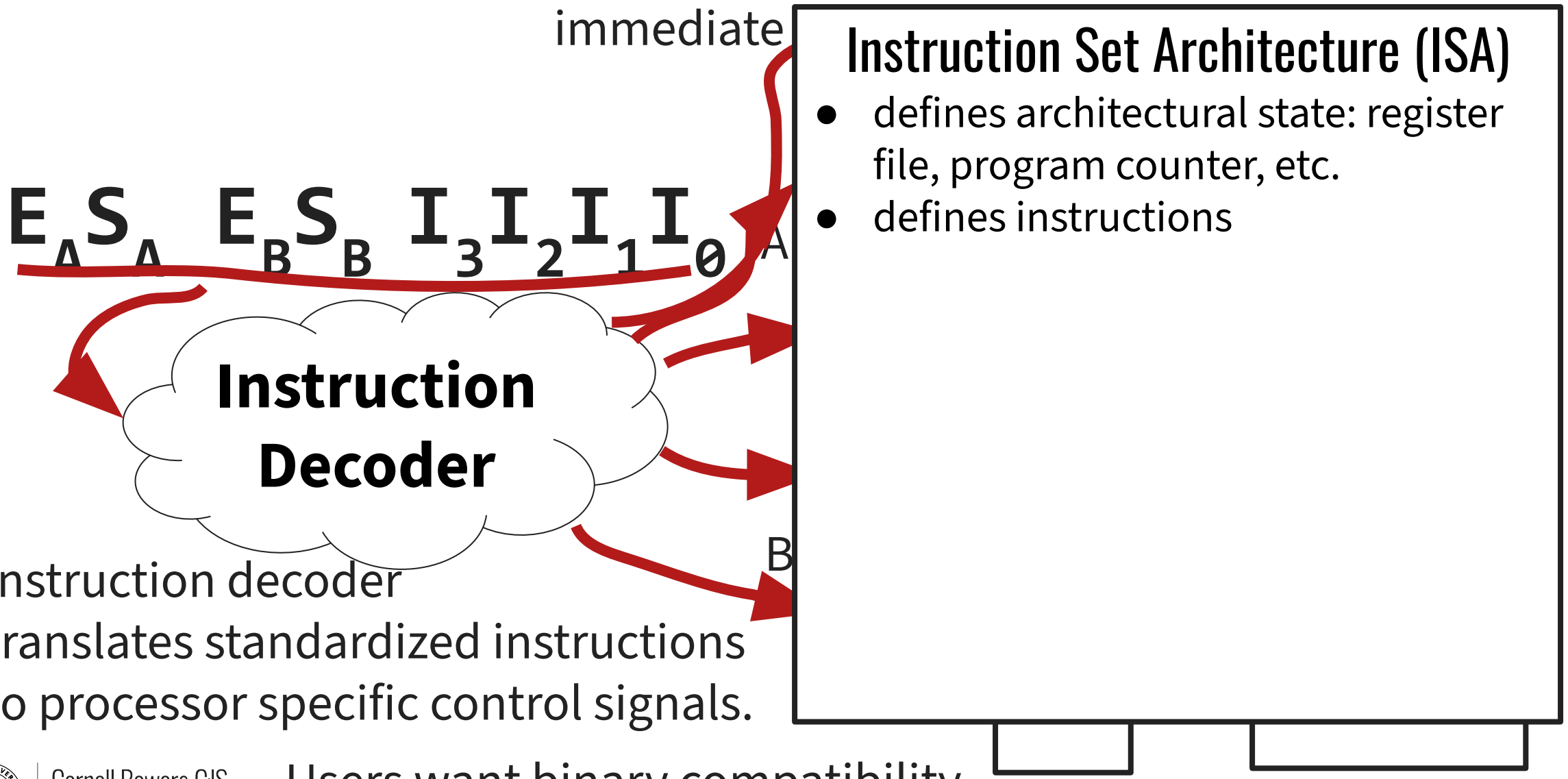
RISC-V, ARM, x86 — what is the difference?



Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

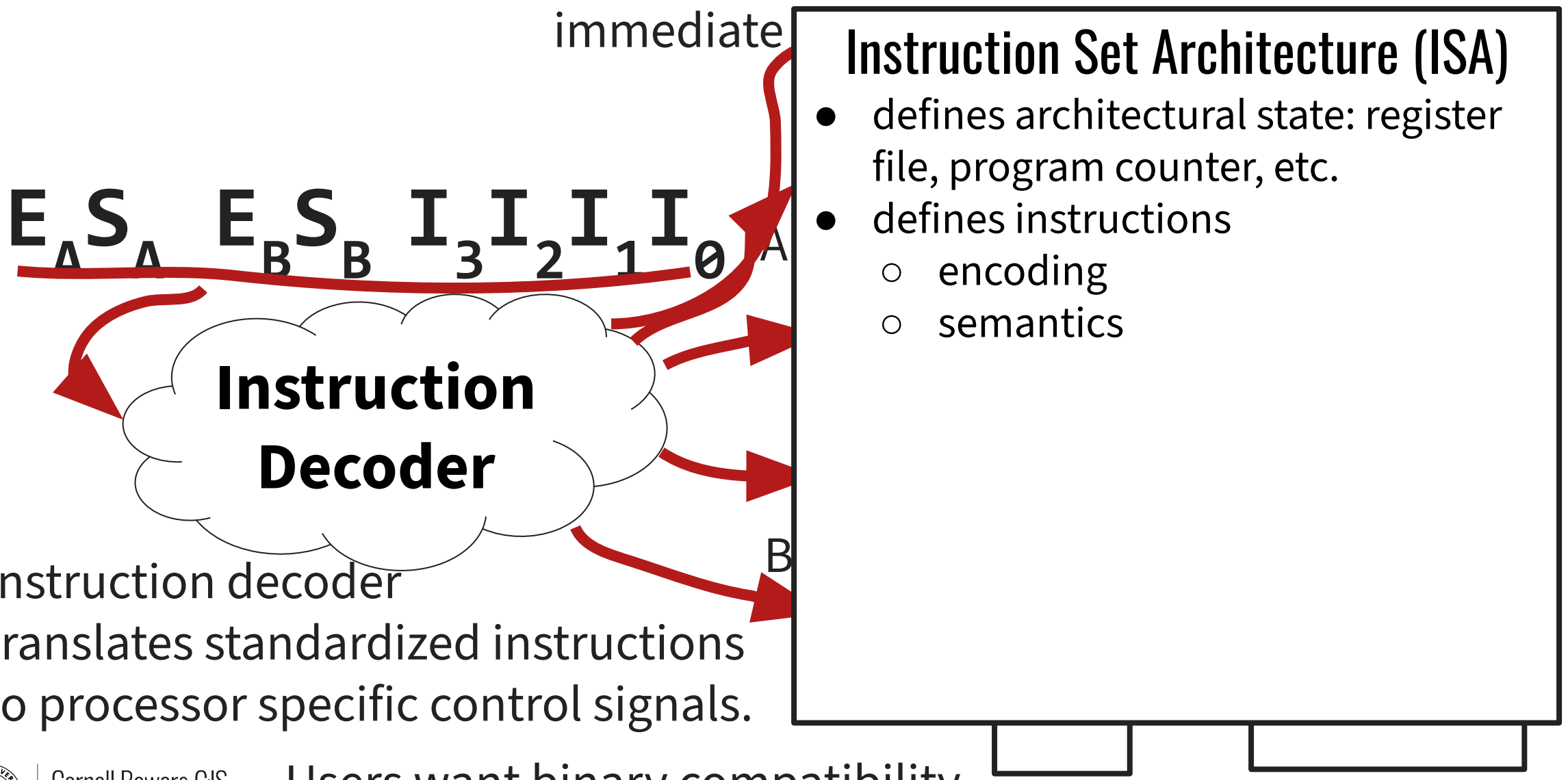
RISC-V, ARM, x86 — what is the difference?



Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

RISC-V, ARM, x86 — what is the difference?



Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

RISC-V, ARM, x86 — what is the difference?

immediate

Instruction Set Architecture (ISA)

- defines architectural state: register file, program counter, etc.
- defines instructions

E S E S I I I I

Instruction	Name	Description	Opcode	Funct 3	Funct7
`add rd rs1 rs2`	*ADD*	`R[rd] = R[rs1] + R[rs2]`	`011 0011`	`000`	`000 0000`
`sub rd rs1 rs2`	*SUBTRACT*	`R[rd] = R[rs1] - R[rs2]`	`011 0011`	`000`	`010 0000`

B

Instruction decoder translates standardized instructions to processor specific control signals.

Users want binary compatibility.

RISC-V, ARM, x86 — what is the difference?

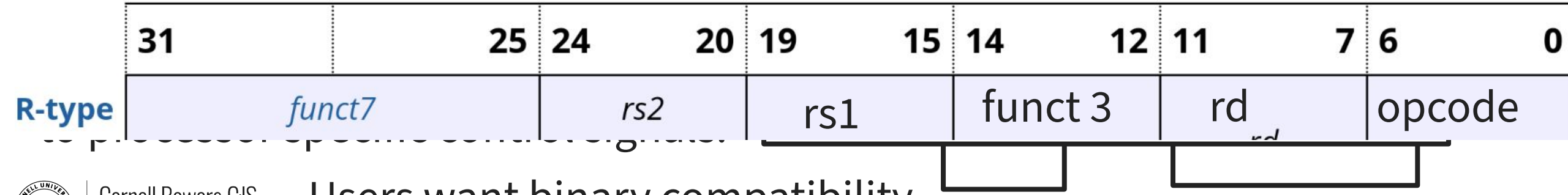
immediate

Instruction Set Architecture (ISA)

- defines architectural state: register file, program counter, etc.
- defines instructions

E S E S I I I I

Instruction	Name	Description	Opcode	Funct 3	Funct7
`add rd rs1 rs2`	*ADD*	`R[rd] = R[rs1] + R[rs2]`	`011 0011`	`000`	`000 0000`
`sub rd rs1 rs2`	*SUBTRACT*	`R[rd] = R[rs1] - R[rs2]`	`011 0011`	`000`	`010 0000`



Users want binary compatibility.

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64
- ARM is a processor manufacturer, there are several generations of ISAs

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64
- ARM is a processor manufacturer, there are several generations of ISAs
  - ARMv9 is the latest, implemented by several different companies

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64
- ARM is a processor manufacturer, there are several generations of ISAs
  - ARMv9 is the latest, implemented by several different companies
- RISC-V: Instruction set from Berkeley

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64
- ARM is a processor manufacturer, there are several generations of ISAs
  - ARMv9 is the latest, implemented by several different companies
- RISC-V: Instruction set from Berkeley
  - designed to be simple
  - came out of vector processor design project
  - no licenses, no patents (hopefully)

RISC-V, ARM, x86 — what is the difference?

- All are Instruction Set Architectures (ISAs)
- x86 is the oldest, defined by Intel
  - x86 specifically refers to older 32-bit ISA, new 64-bit ISA is often called AMD64
- ARM is a processor manufacturer, there are several generations of ISAs
  - ARMv9 is the latest, implemented by several different companies
- RISC-V: Instruction set from Berkeley
  - designed to be simple
  - came out of vector processor design project
  - no licenses, no patents (hopefully)