Computer Architecture
Computer Engineering Track
Tutorial 11

The Discussion of the Midterm Exam Questions

Artem Burmyakov, Muhammad Fahim, Alexander Tormasov

November 12, 2020

INNOPOLIS UNIVERSITY

Program Translation: From C to MIPS Binary

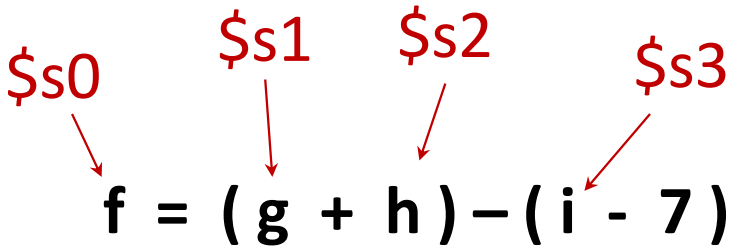
C program instruction to be translated:

f = (g + h) – (i - 7)

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively



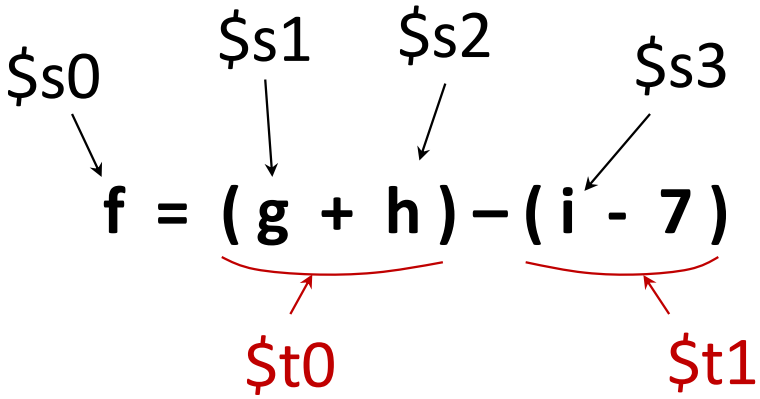
Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results



Program Translation: From C to MIPS Binary

Step 1:

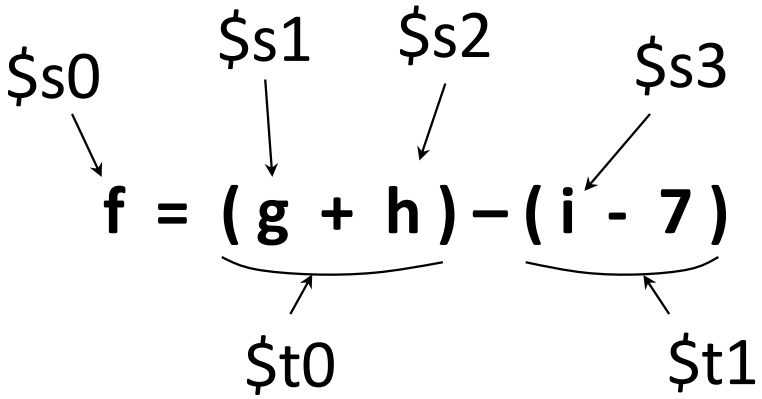
Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

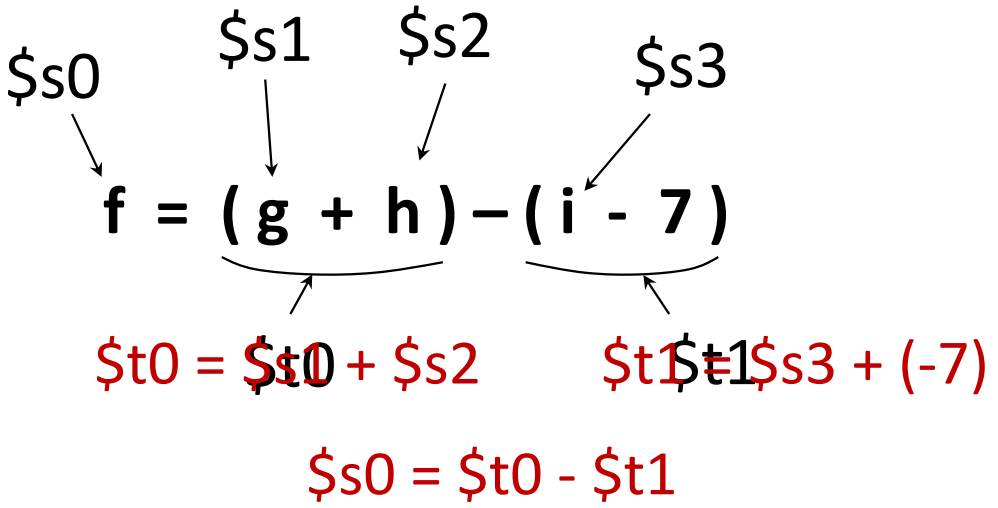
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2
addi  $t1,  $s3,  -7
sub   $s0,  $t0,  $t1
```



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

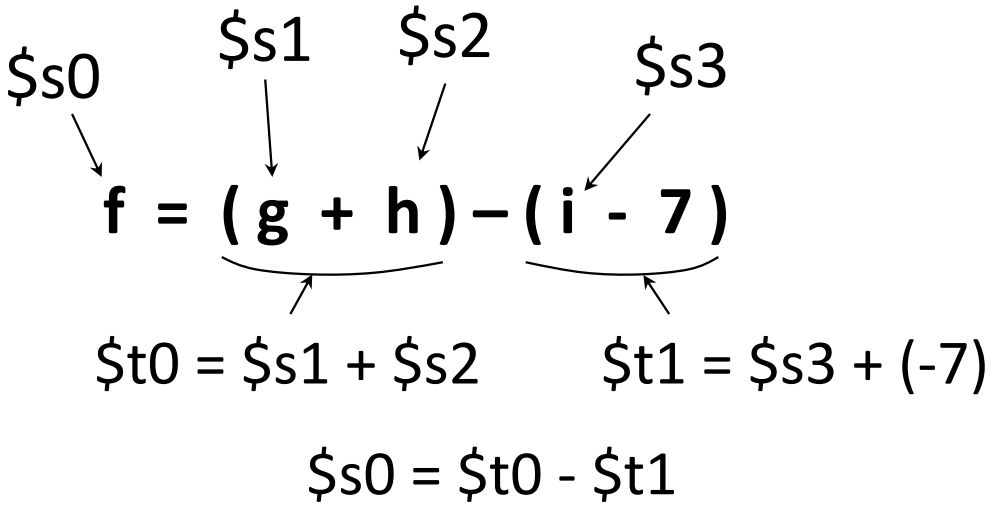
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

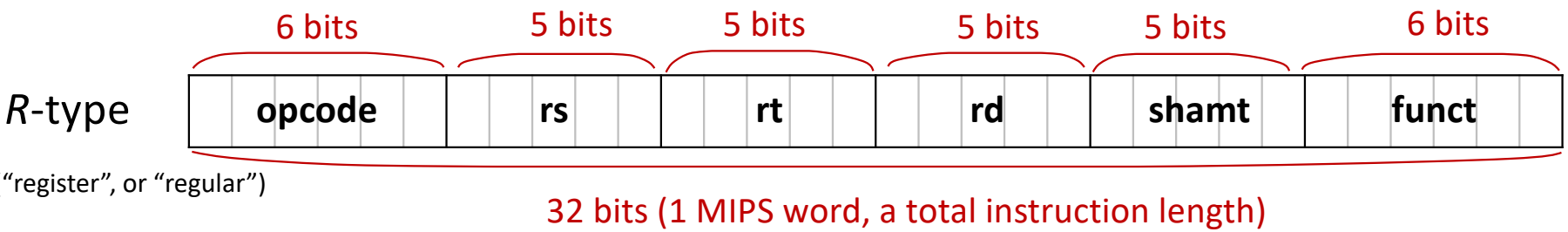


Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

Types of MIPS Processor Instructions: R, I, and J

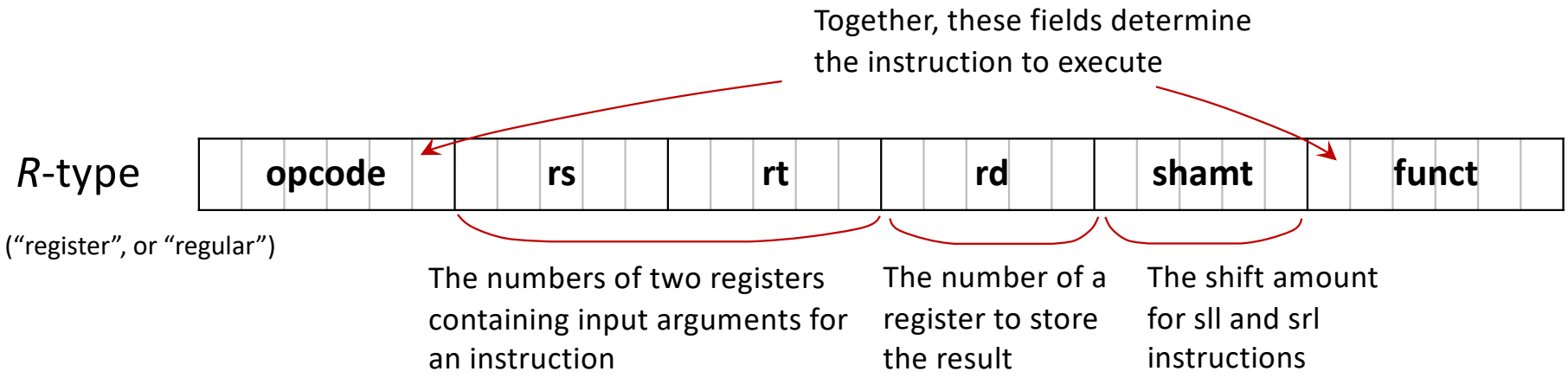
All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



("register", or "regular")

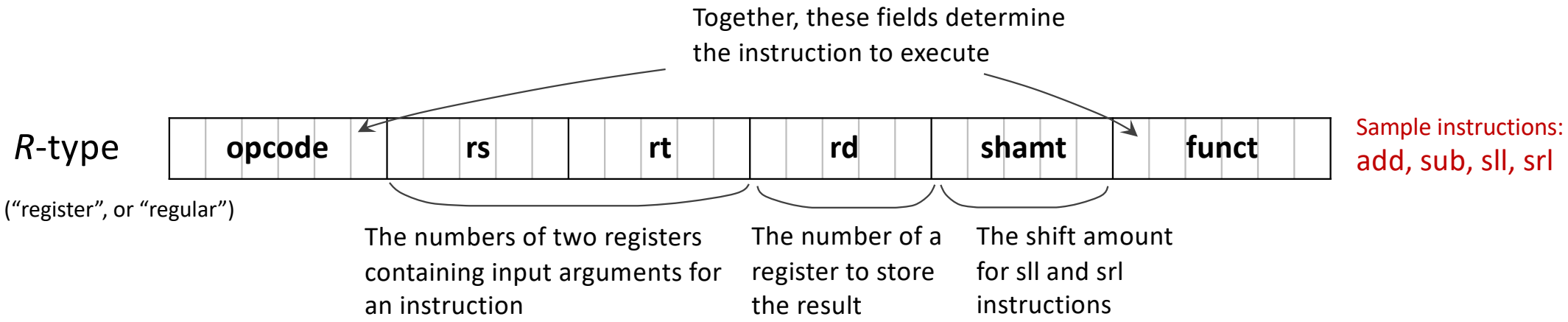
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



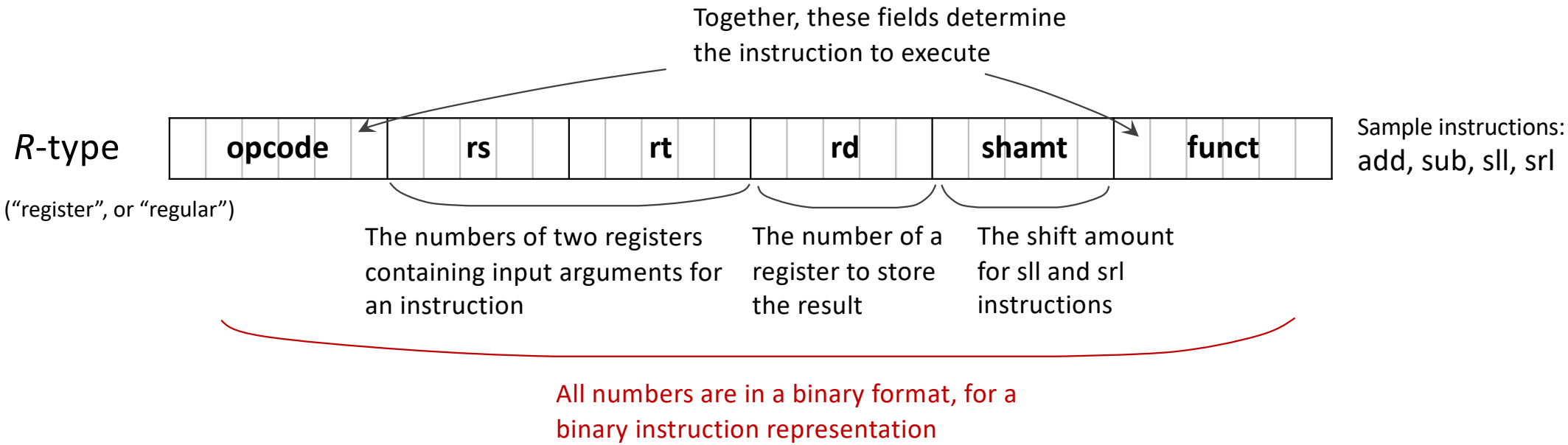
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

R-type

add, sub, sll, srl

("register", or "regular")

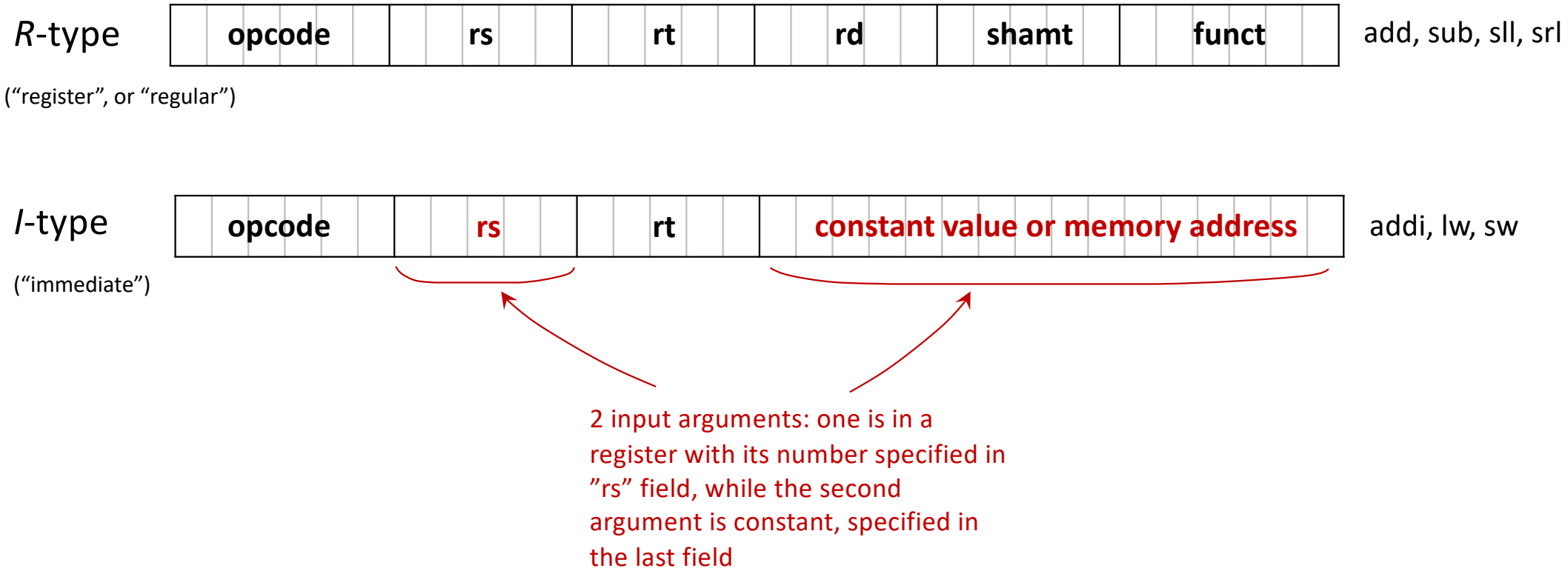
I-type

addi, lw, sw

("immediate")

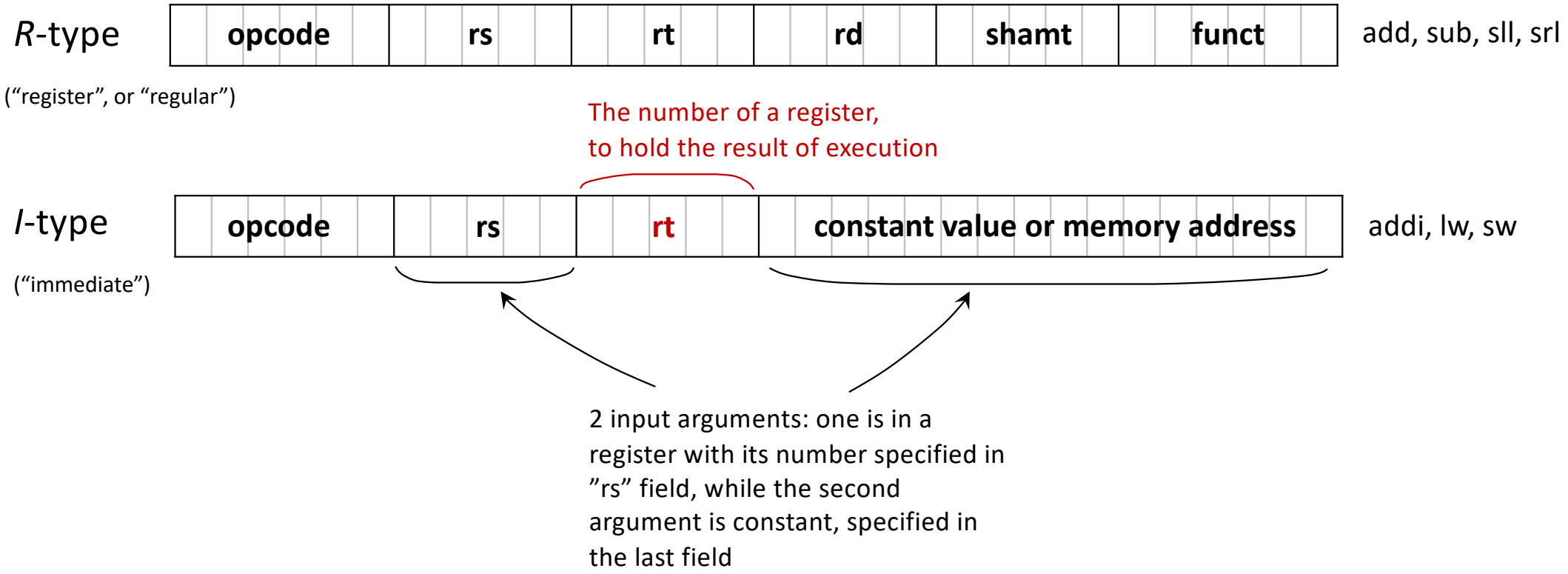
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

R-type

add, sub, sll, srl

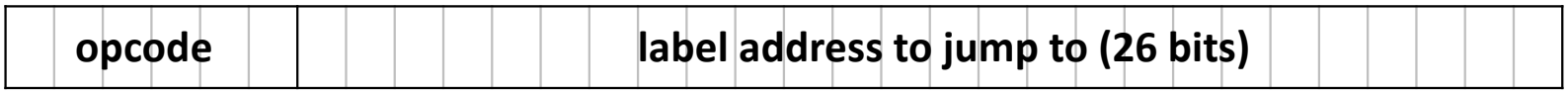
("register", or "regular")

I-type

addi, lw, sw

("immediate")

J-type



j, jr, jal

("jump")

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

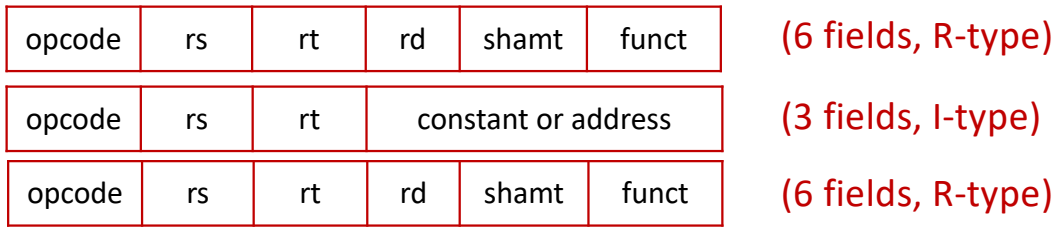
Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

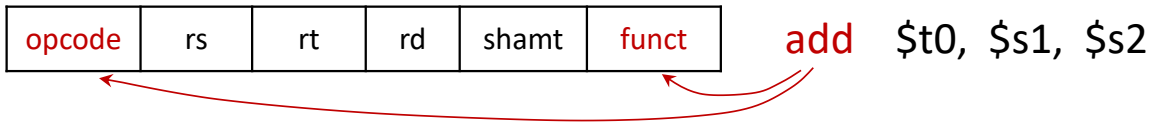
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

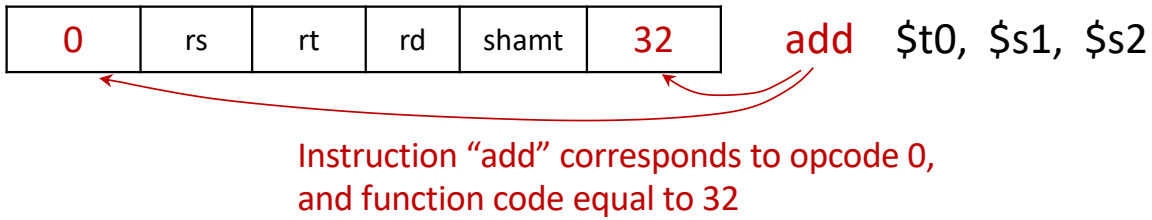
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

I-type instructions do not have "funct" field, but differ in opcode values

All *R*-type instructions share opcode "0", but differ in function codes ("funct")

Step 4:

Convert MIPS instructions into decimal format:

0	rs	rt	rd	shamt	32

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

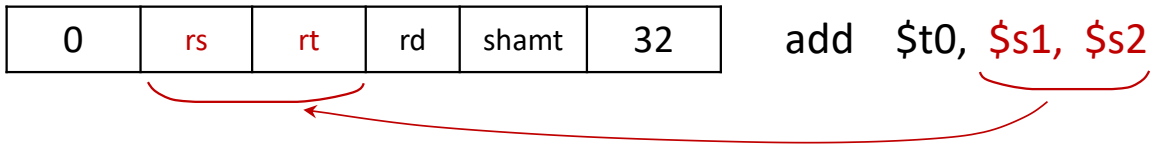
Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:



Place the numbers of registers $s1 and $s2 into fields “rs” and “rt”, respectively; these numbers are taken from MIPS specification

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2    (R-type)
addi $t1, $s3, -7     (I-type)
sub  $s0, $t0, $t1    (R-type)
```

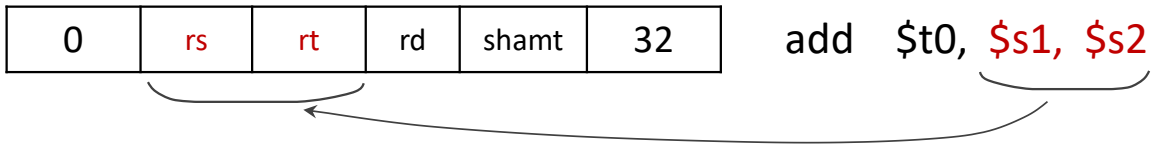
From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

…

Step 4:

Convert MIPS instructions into decimal format:



Place the numbers of registers $s1 and $s2 into fields “rs” and “rt”, respectively; these numbers are taken from MIPS specification

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

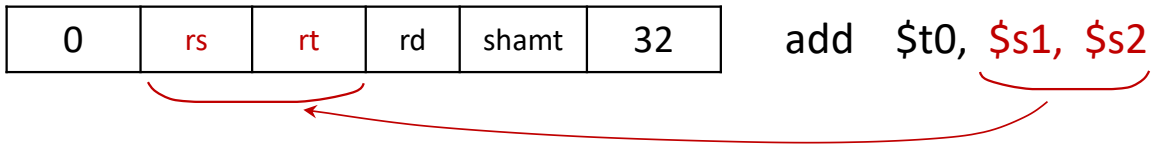
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Registers $s1 and $s2 correspond to numbers 17 and 18, respectively

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

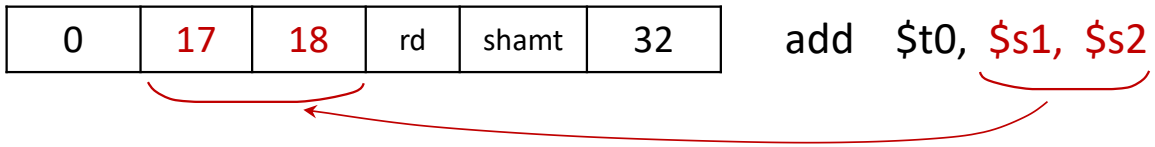
```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Registers $s1 and $s2 correspond to numbers 17 and 18, respectively

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	rd	shamt	32

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

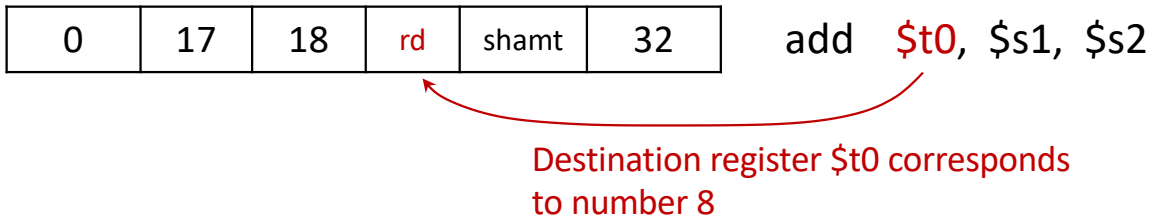
```
add  $t0, $s1, $s2    (R-type)
addi $t1, $s3, -7     (I-type)
sub  $s0, $t0, $t1    (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

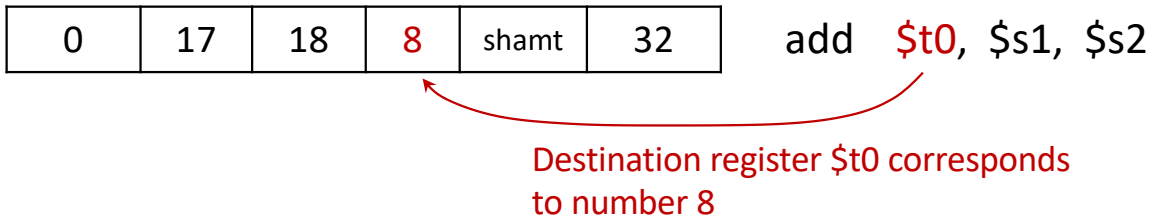
```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	shamt	32

add $t0, $s1, $s2

Whenever field “shamt” is unused, it is reset to 0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

add $t0, $s1, $s2

Whenever field “shamt” is unused, it is reset to 0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
opcode	rs	rt	constant or address		
opcode	rs	rt	rd	shamt	funct

```
add  $t0, $s1, $s2
addi $t1, $s3, -7
sub  $s0, $t0, $t1
```

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

…

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

…

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32	add $t0, $s1, $s2
8	19	9	-7			addi $t1, $s3, -7
0	16	8	9	0	34	sub $s0, $t0, $t1

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

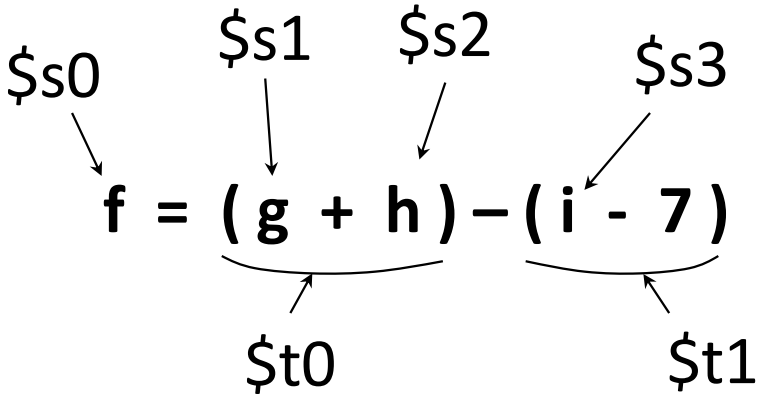
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

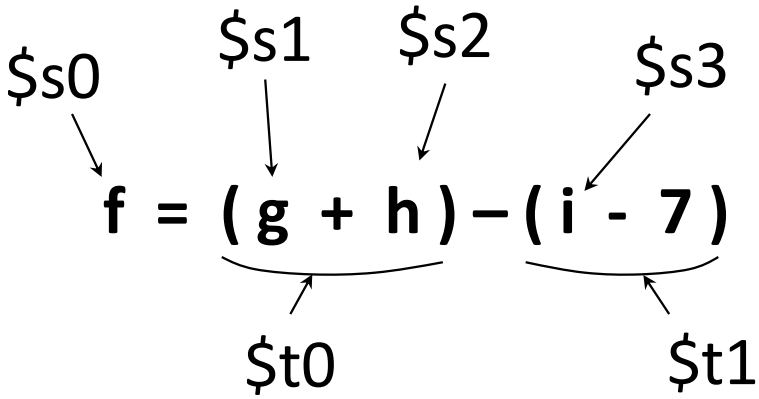
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```



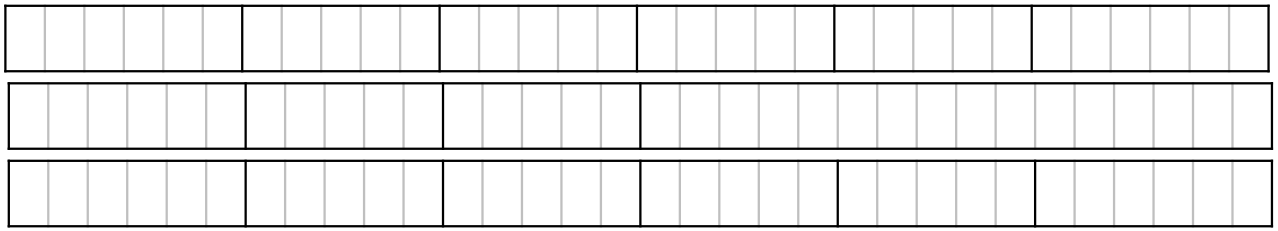
Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

Step 5:

Convert instructions from decimal into binary representation:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

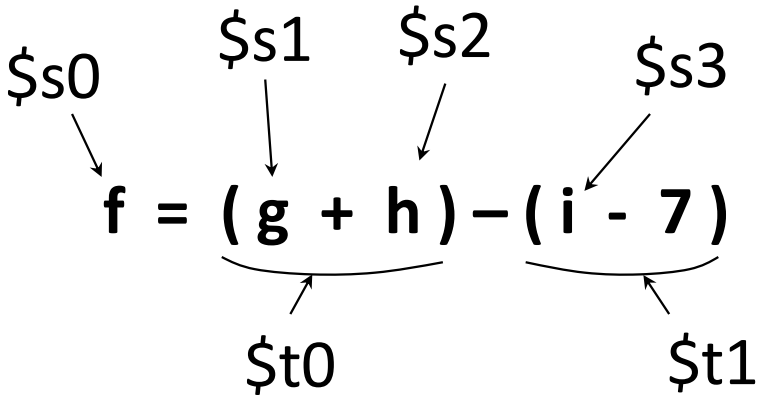
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0																										

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

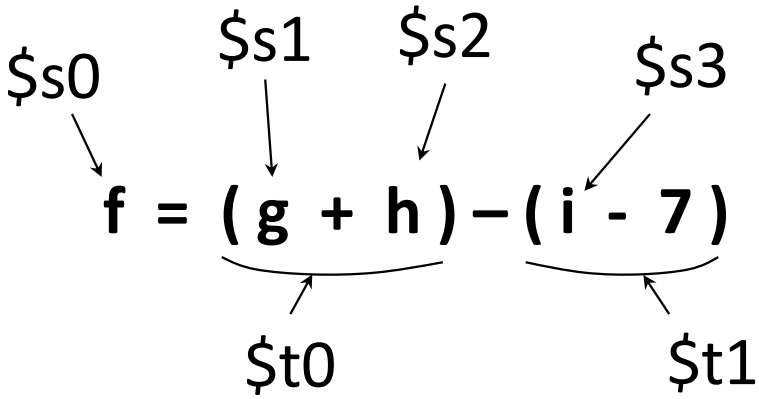
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1																			

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

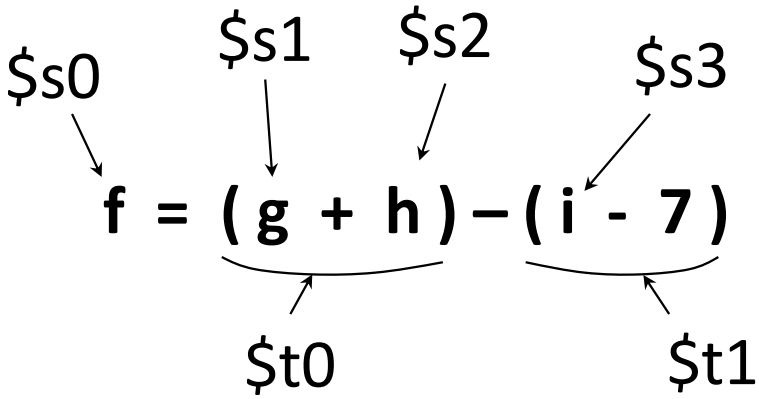
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

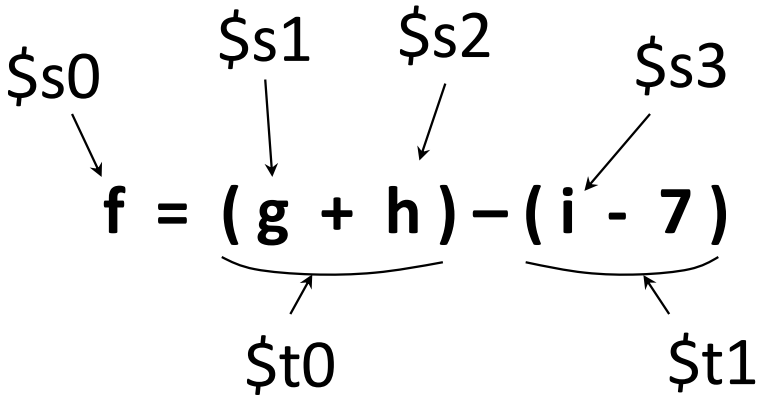
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

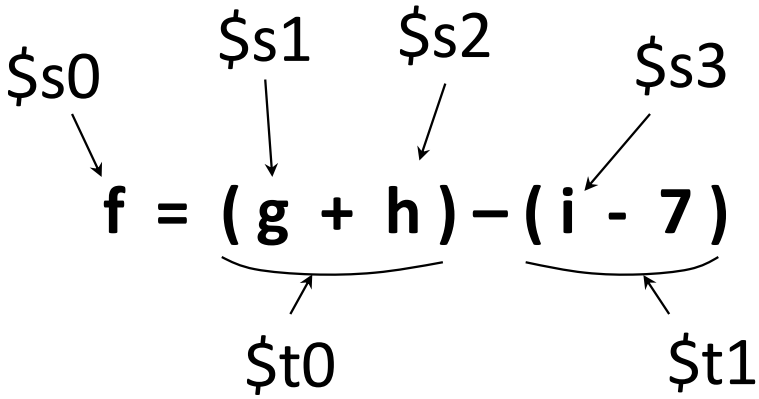
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```



$(-7)_{\text{decimal}} = (1111111111111001)_{\text{binary}}$

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

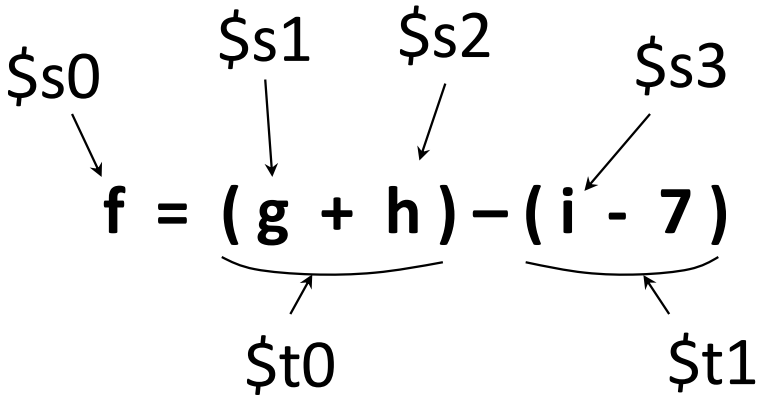
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1		
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0		

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

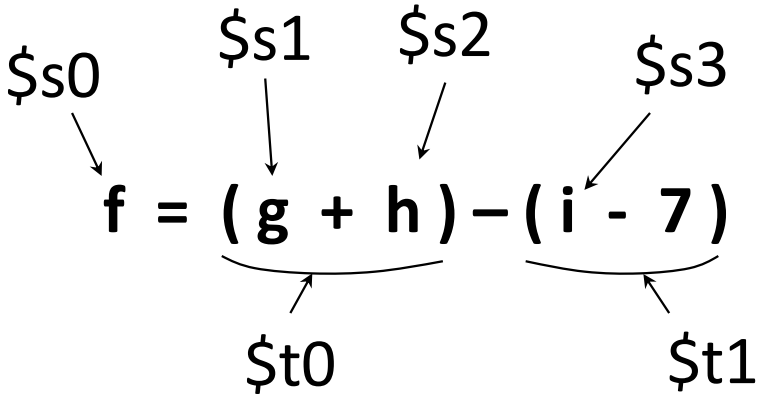
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

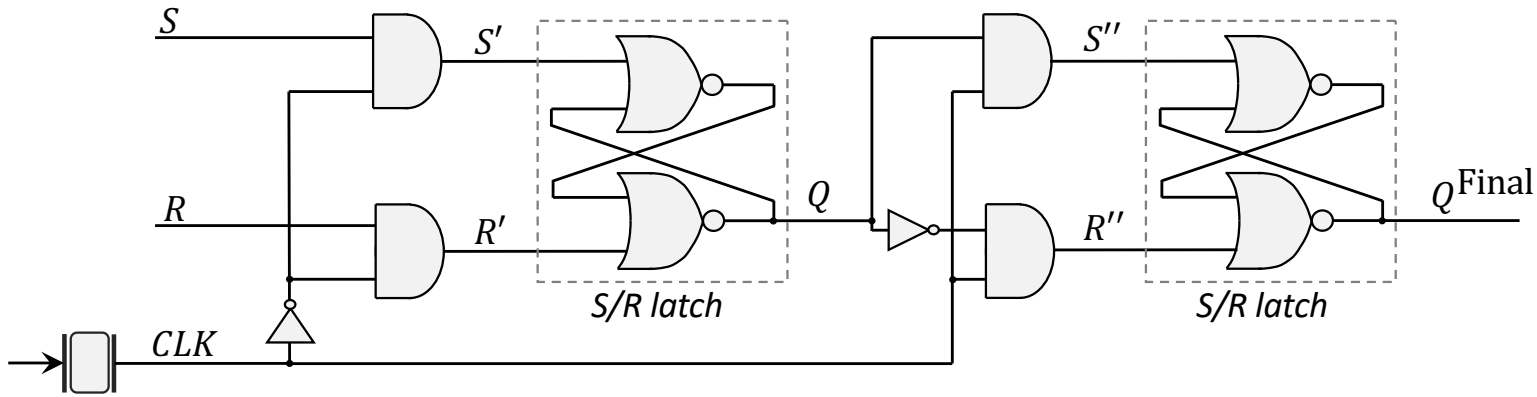
0	17	18	8	0	32
8	19	9	-7		
0	16	8	9	0	34

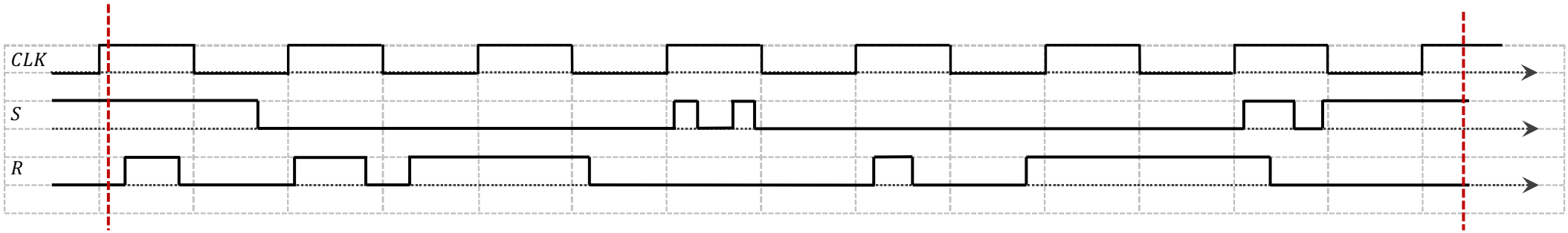
Step 5:

Convert instructions from decimal into binary representation:

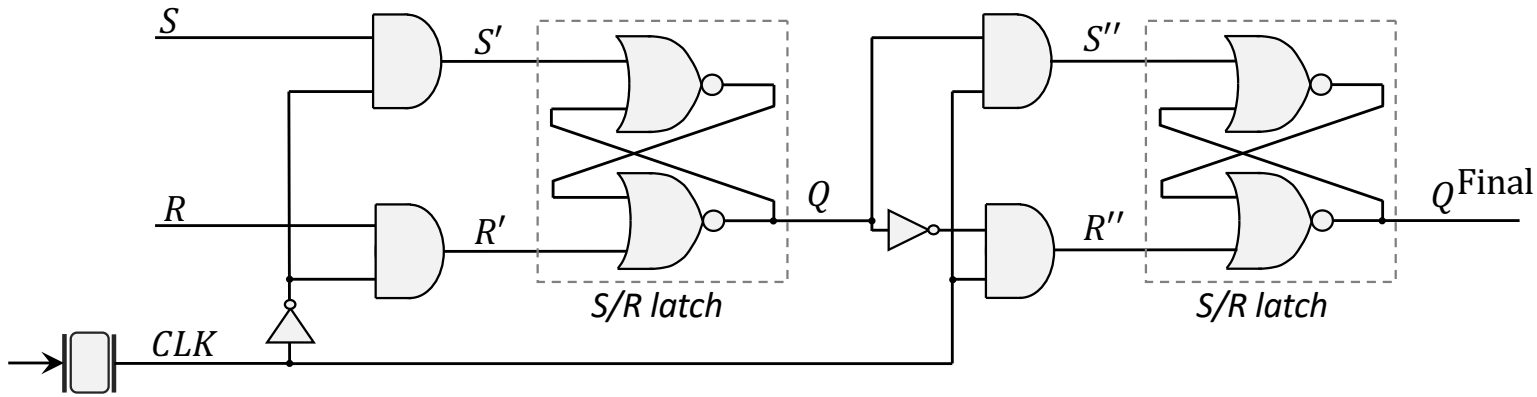
0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0

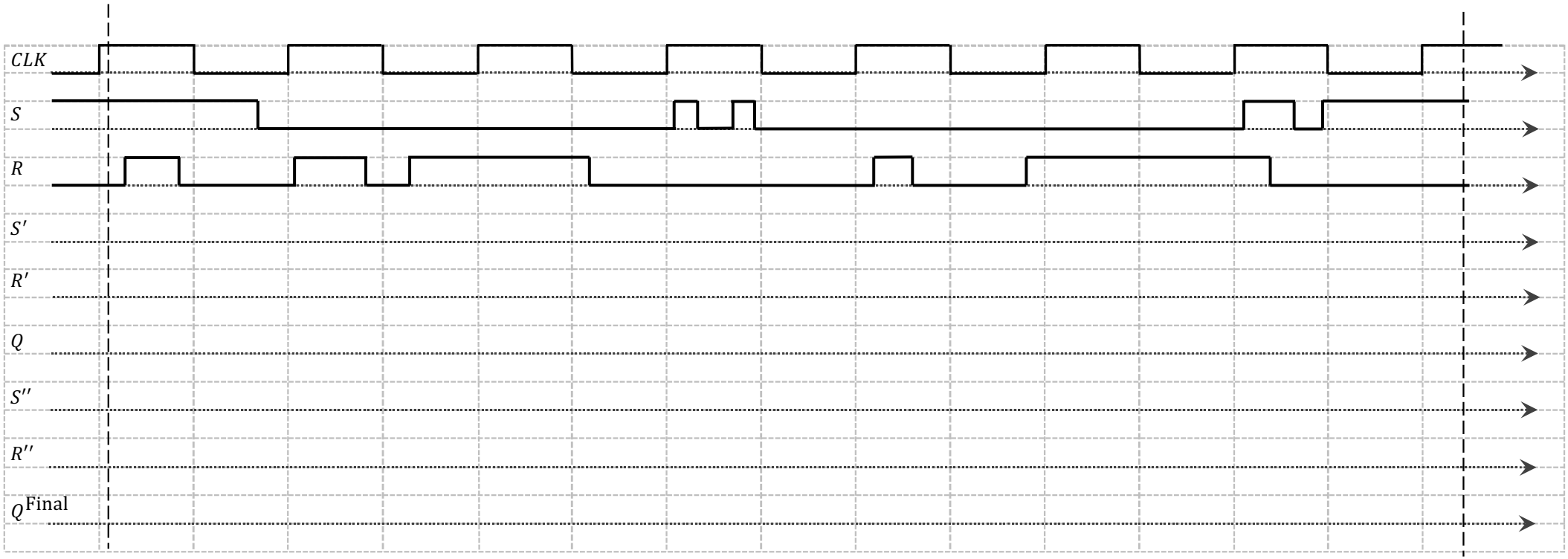
Timing Diagram



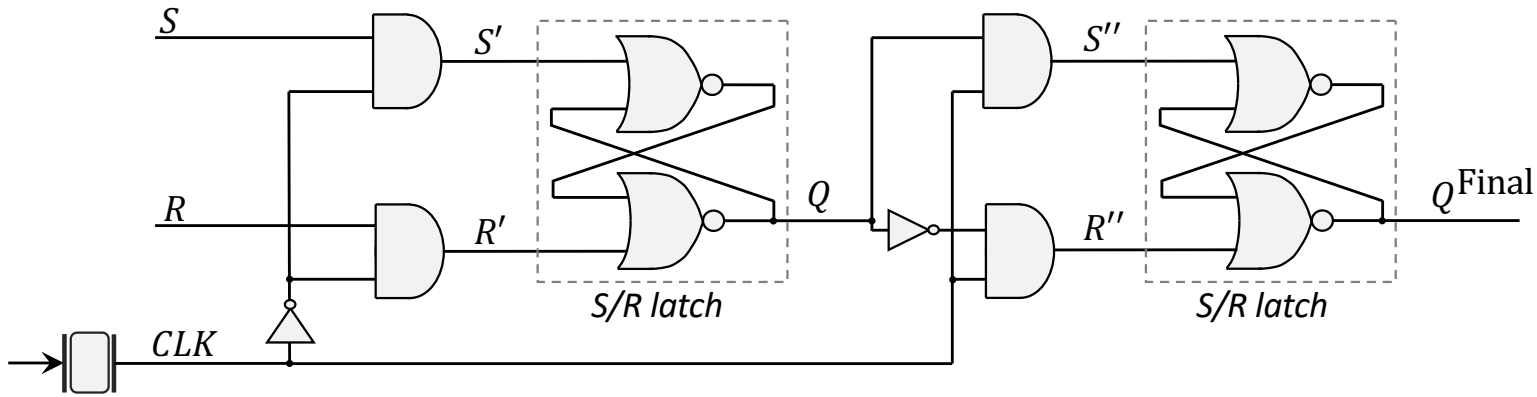


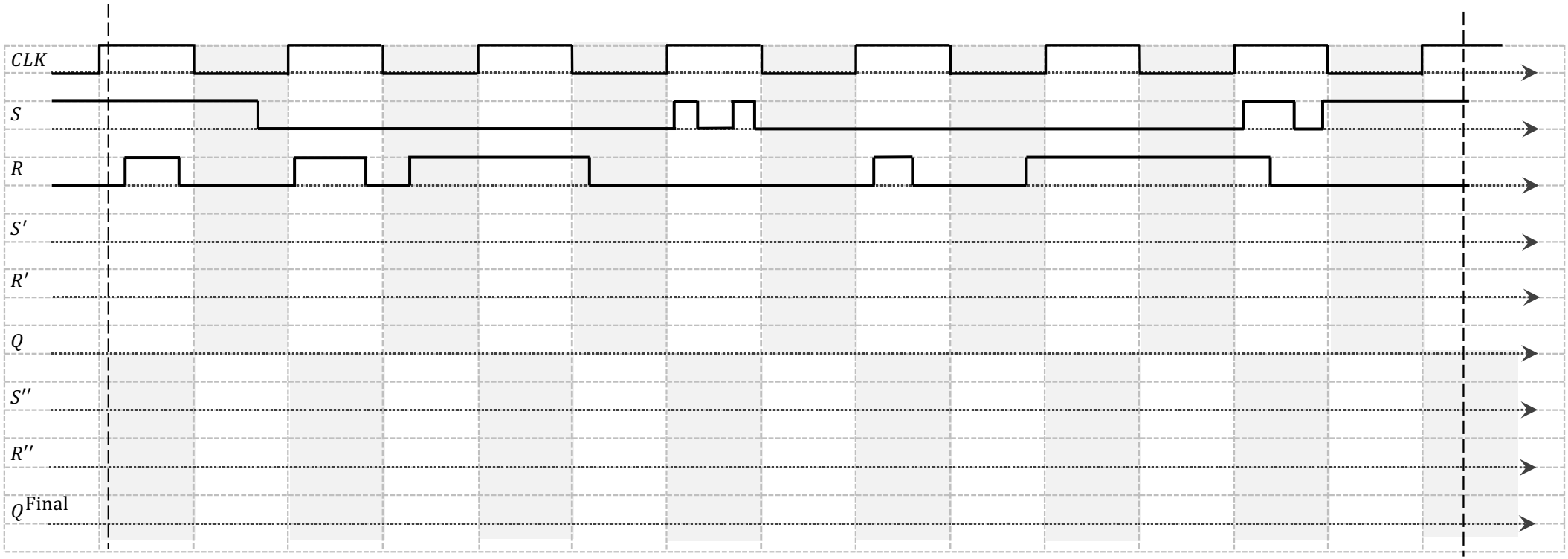
Timing Diagram



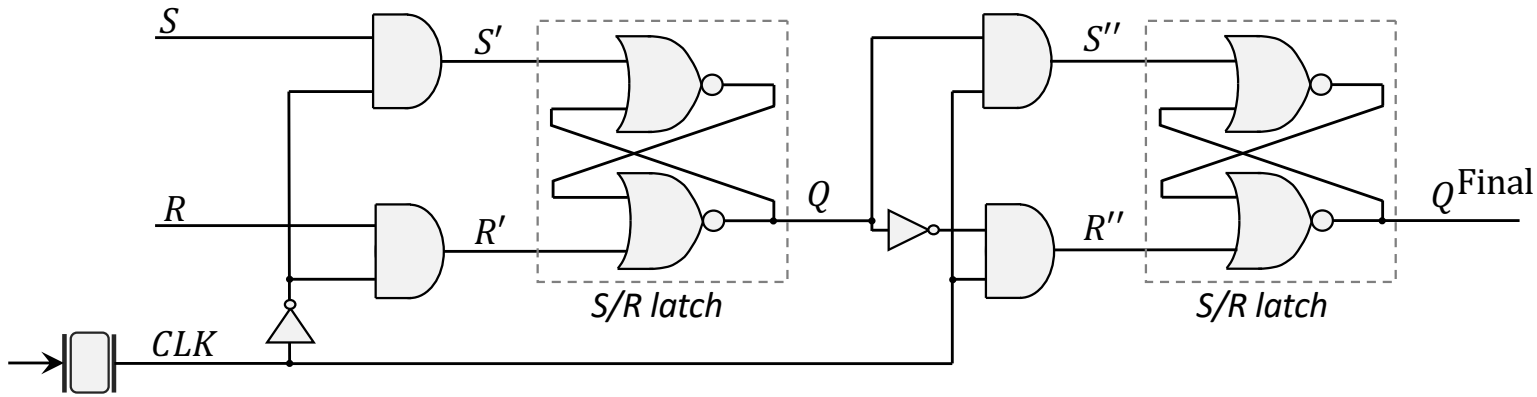


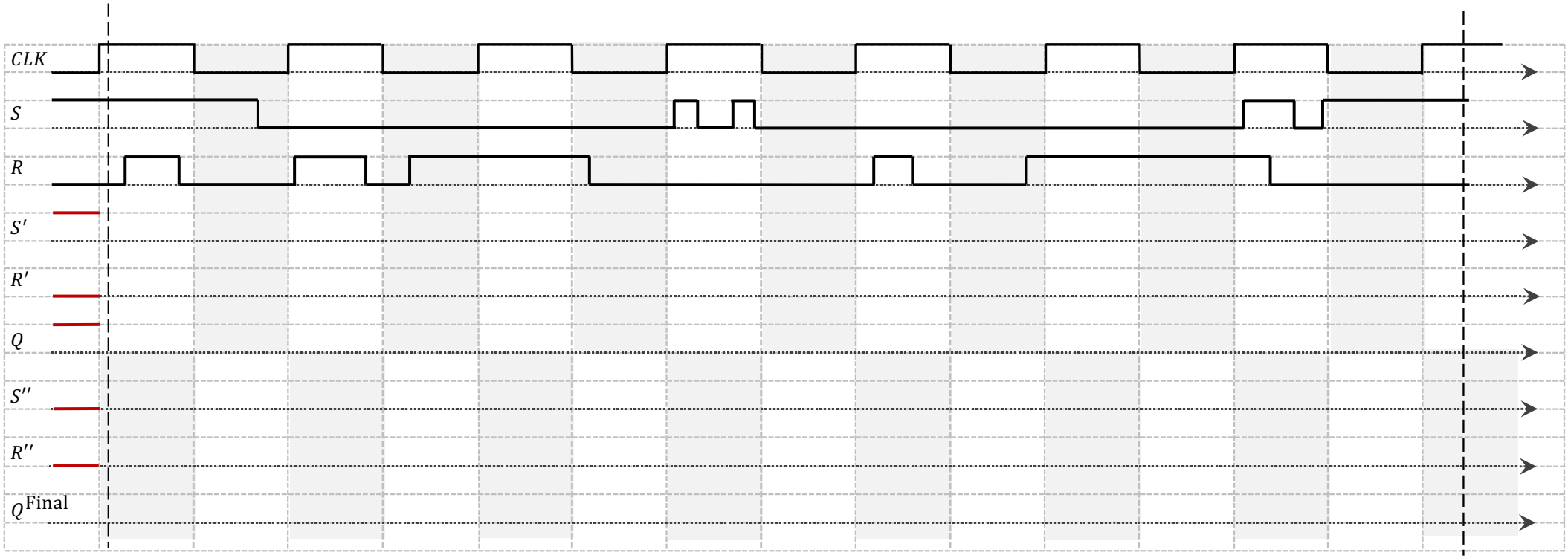
Timing Diagram



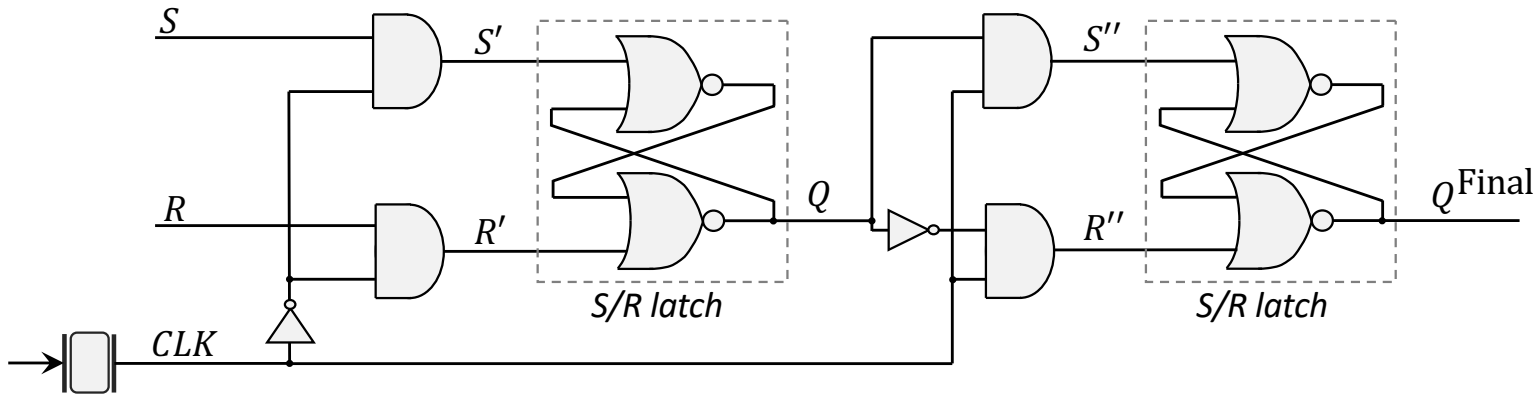


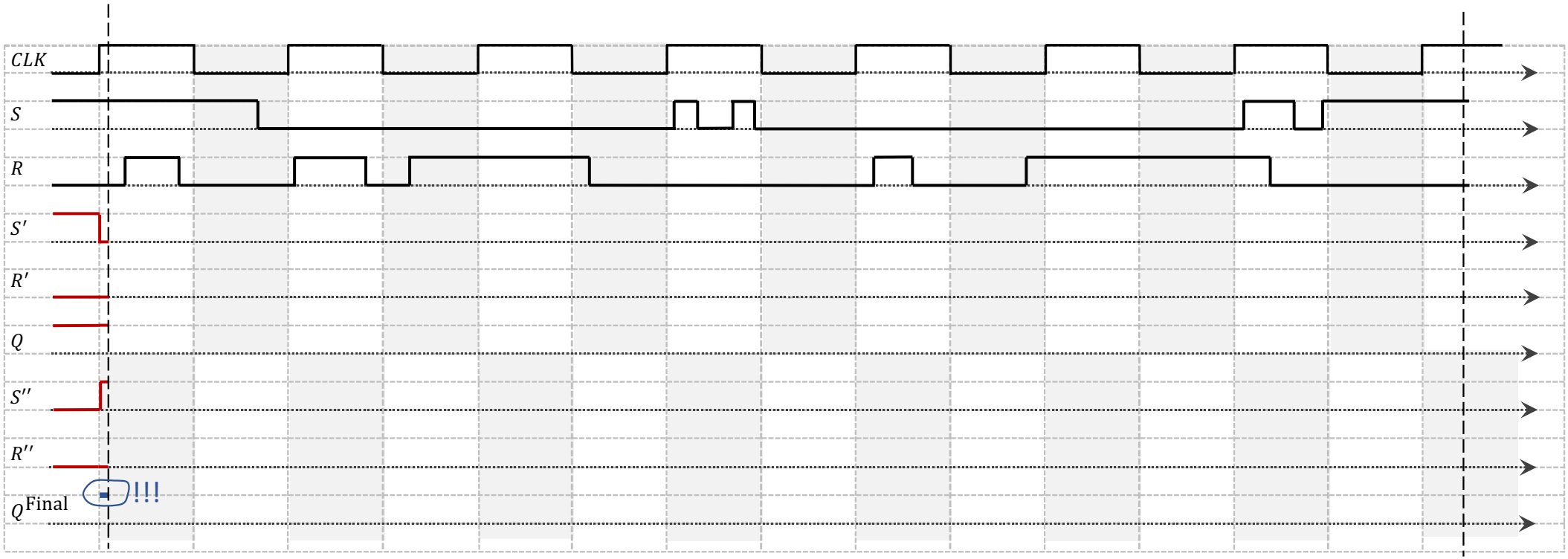
Timing Diagram



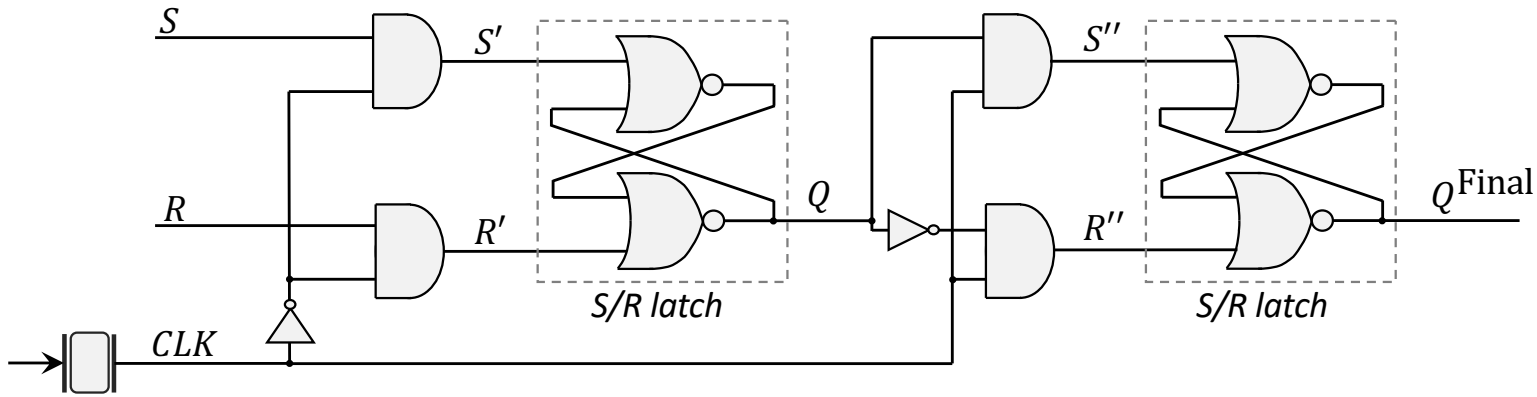


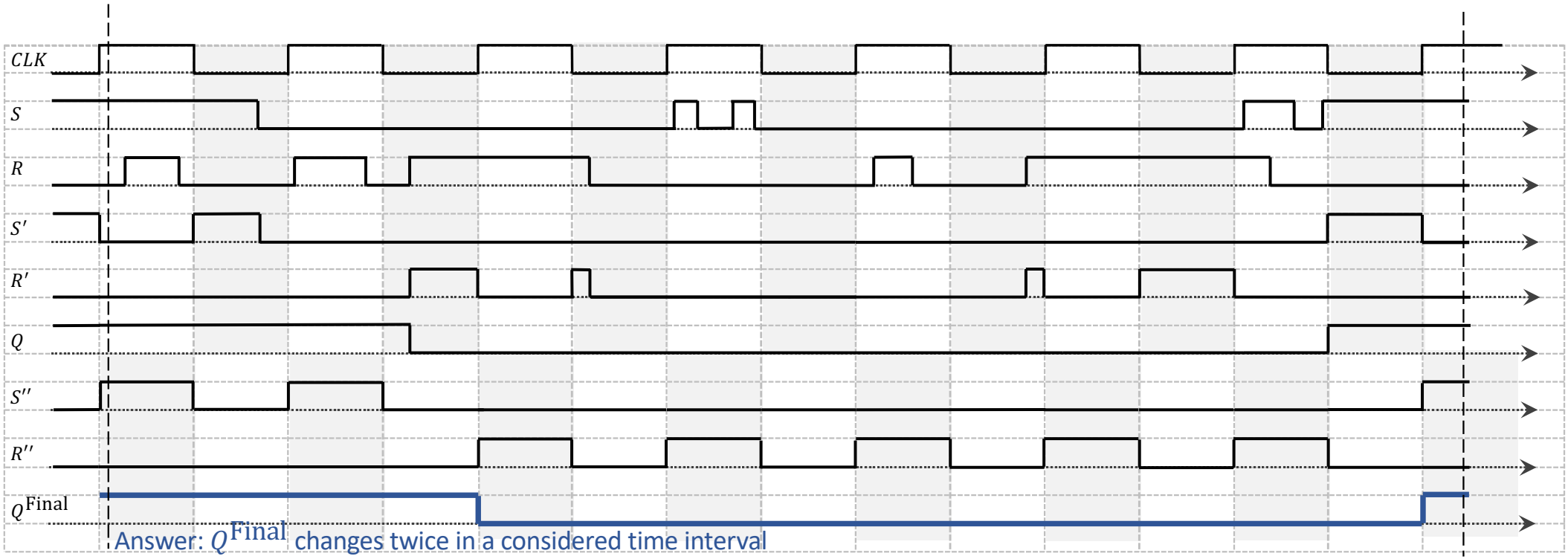
Timing Diagram





Timing Diagram

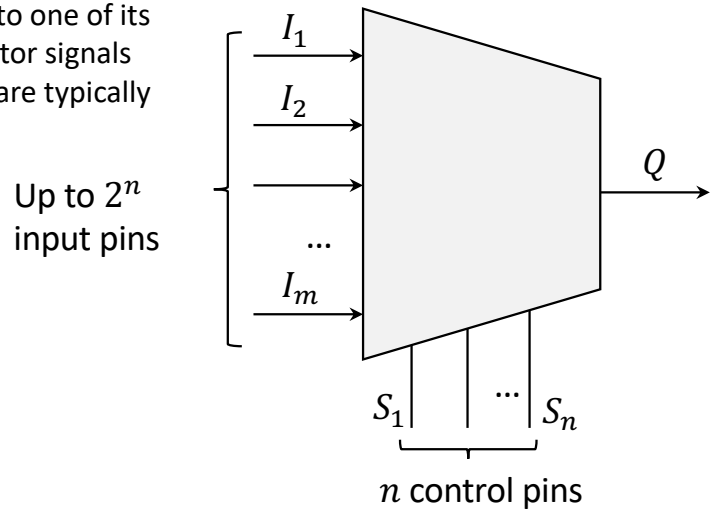




Answer: Q^{Final} changes twice in a considered time interval

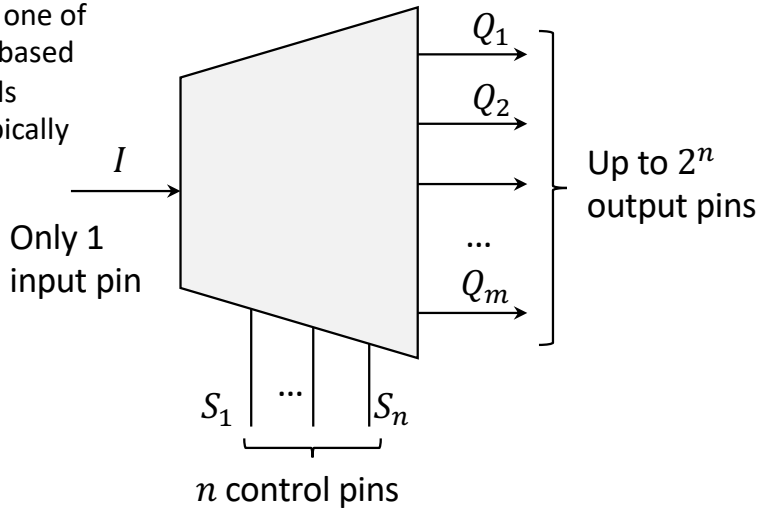
Multiplexor

Sets a specific output to one of its inputs, based on selector signals (all other output pins are typically set to “0”)



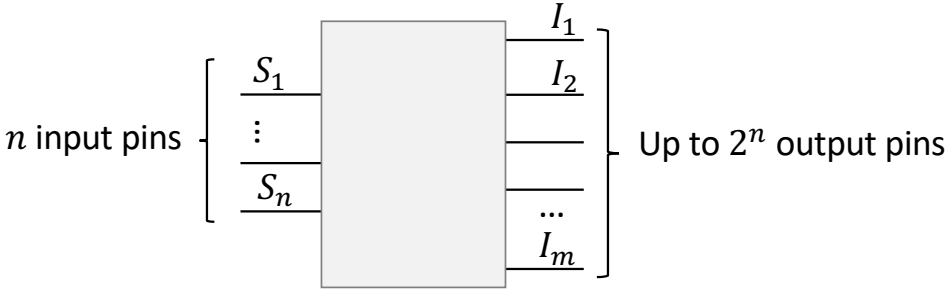
Demultiplexor

Sends input signal to one of its multiple outputs, based on the selector signals (all other outputs typically remain “0”)



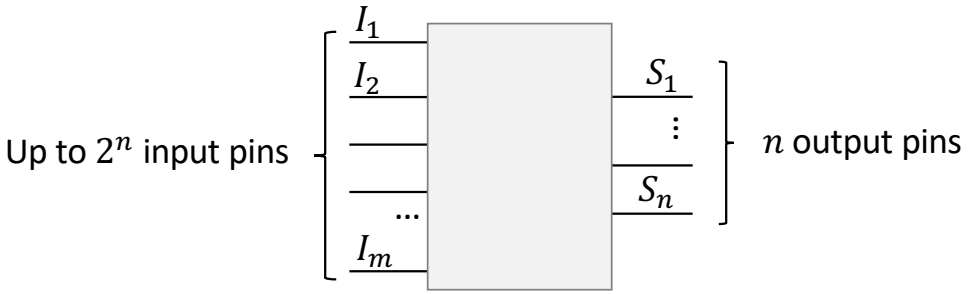
Decoder

Sets to “1” exactly one output pin, which corresponds to the signals of input pins; All other pins are set to “0”



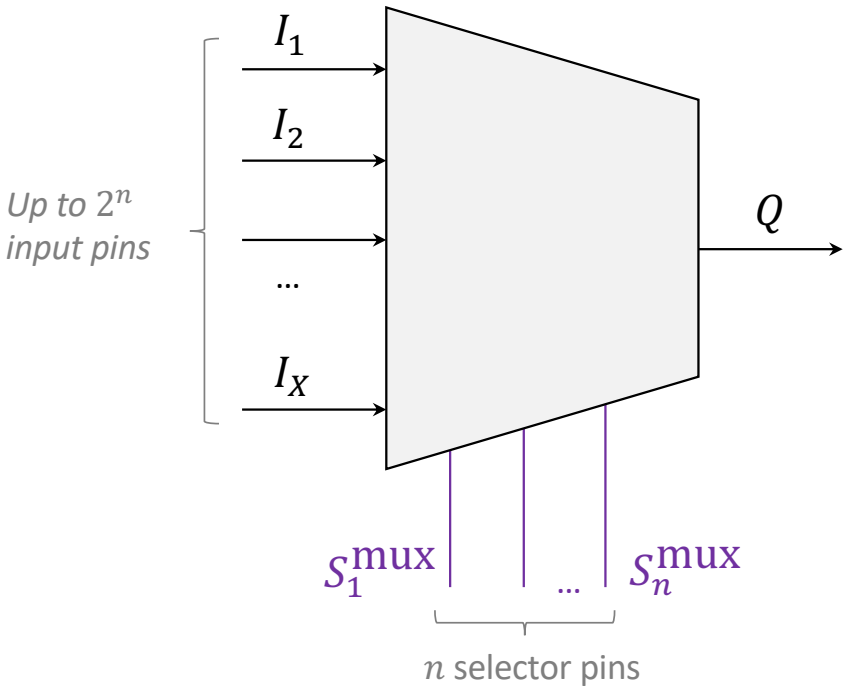
Encoder

The opposite function of a decoder;
Only one input pin is assumed to be “1”, while all others – “0”



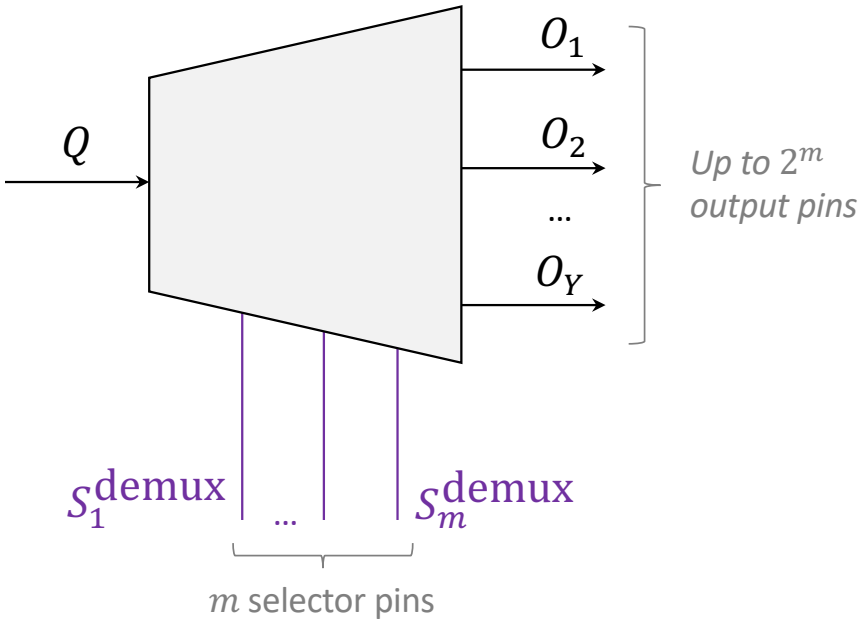
X-to-1 Multiplexer

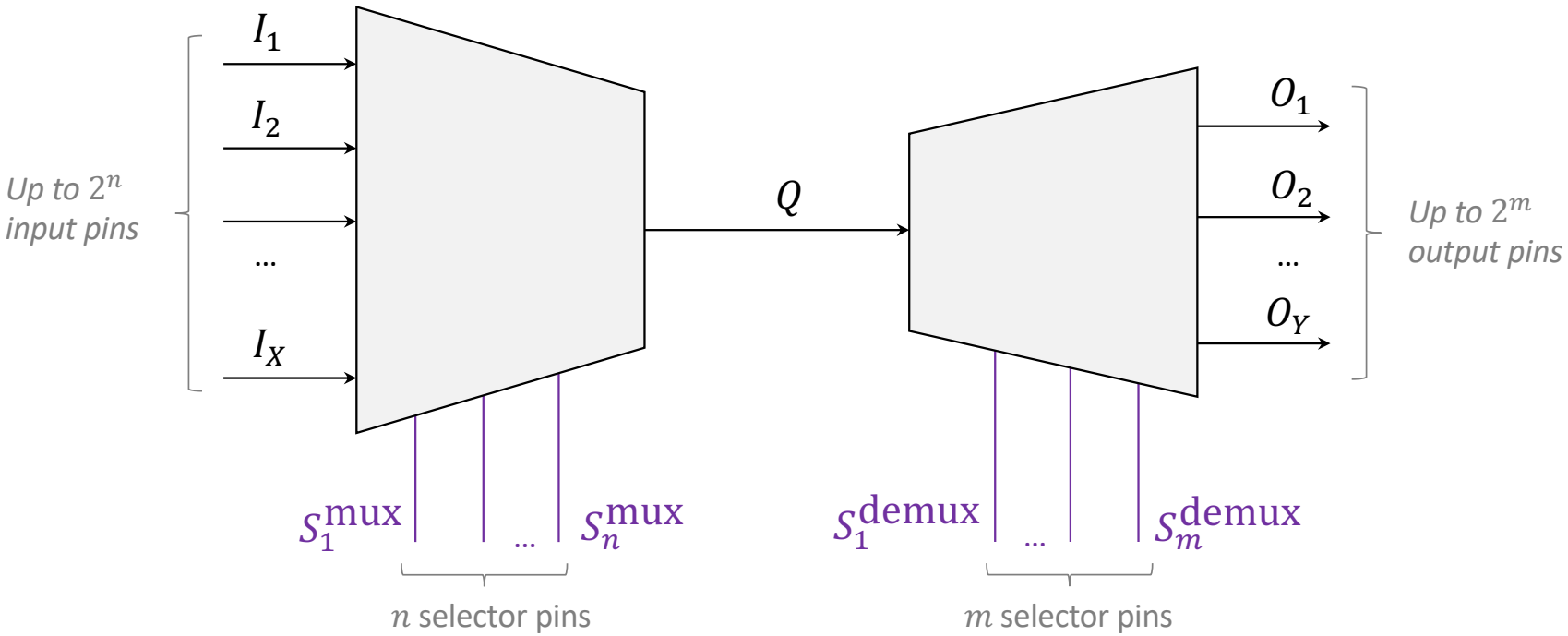
Based on the values of its n selector (control) pins, a multiplexer sets the value of its output Q to the value of one of its $X \leq 2^n$ inputs



1-to-Y Demultiplexer

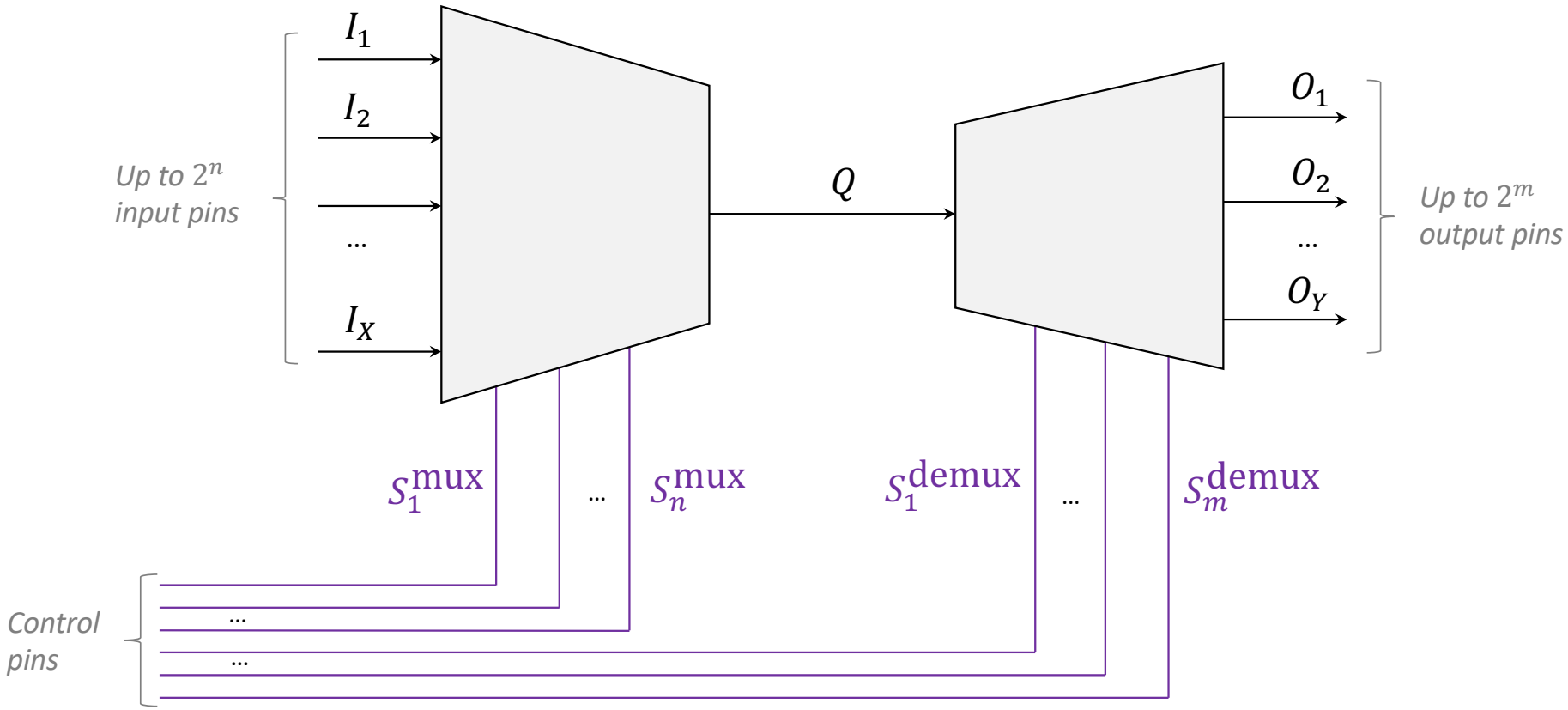
Based on the values of its m selector pins, a demultiplexer outputs its input value Q at one of its $Y \leq 2^m$ output pins, while all other output pins are set to 0s



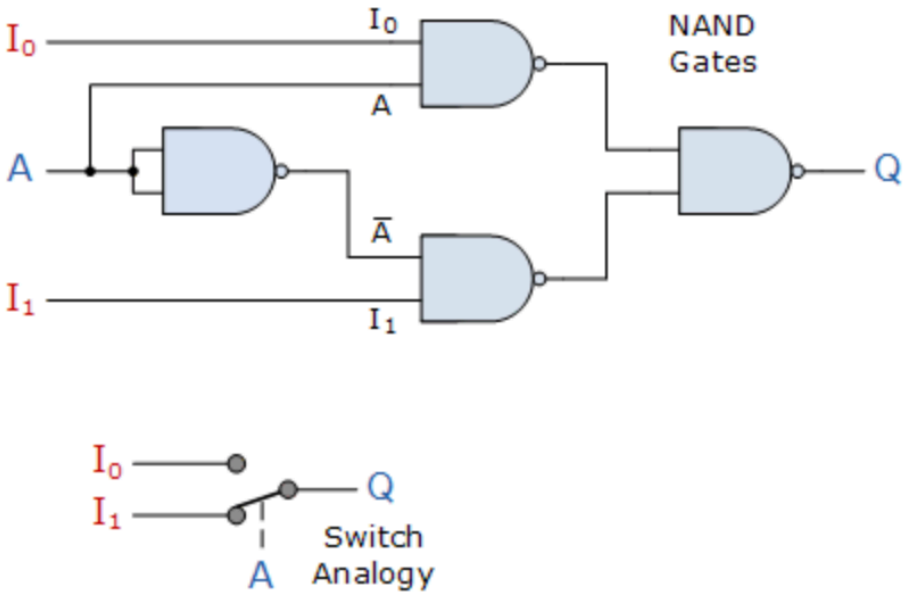

Up to 2^n input pins
I_1
I_2
...
I_X
Q
O_1
O_2
...
O_Y
Up to 2^m output pins
S_1^{mux}
...
S_n^{mux}
n selector pins
S_1^{demux}
...
S_m^{demux}
m selector pins

X-to-*Y* Multiplexer:

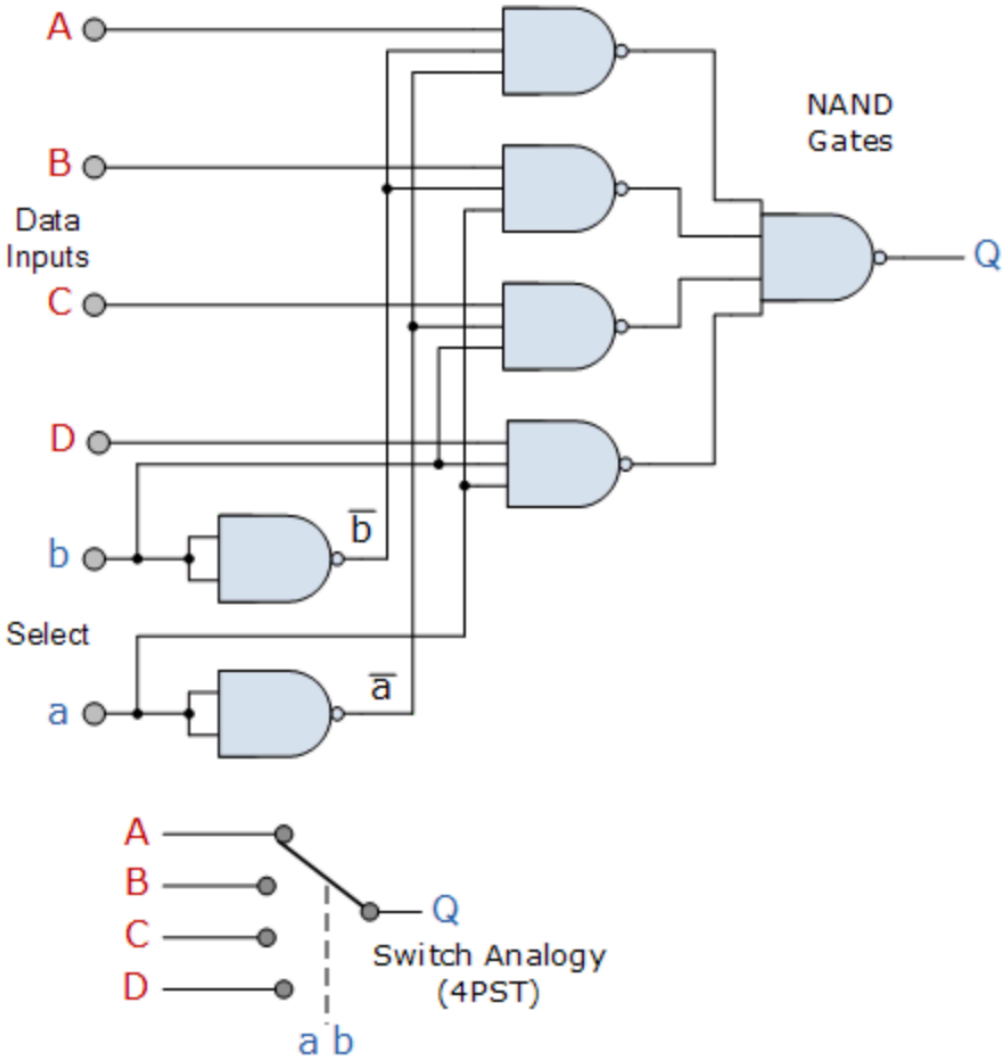
Chooses one of its *X* input pins, and outputs its value at one of its *Y* output pins, while all other output pins are reset to 0s (usually)



2-to-*1* Multiplexer by Using NAND Gates



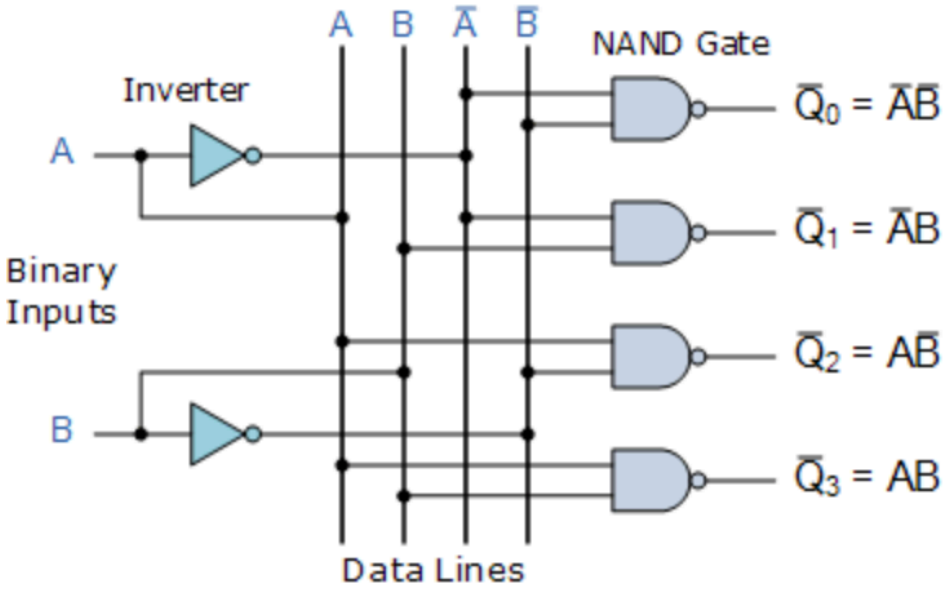
4-to-*1* Multiplexer by Using NAND Gates



The number of logic gates increases significantly with the number of input pins

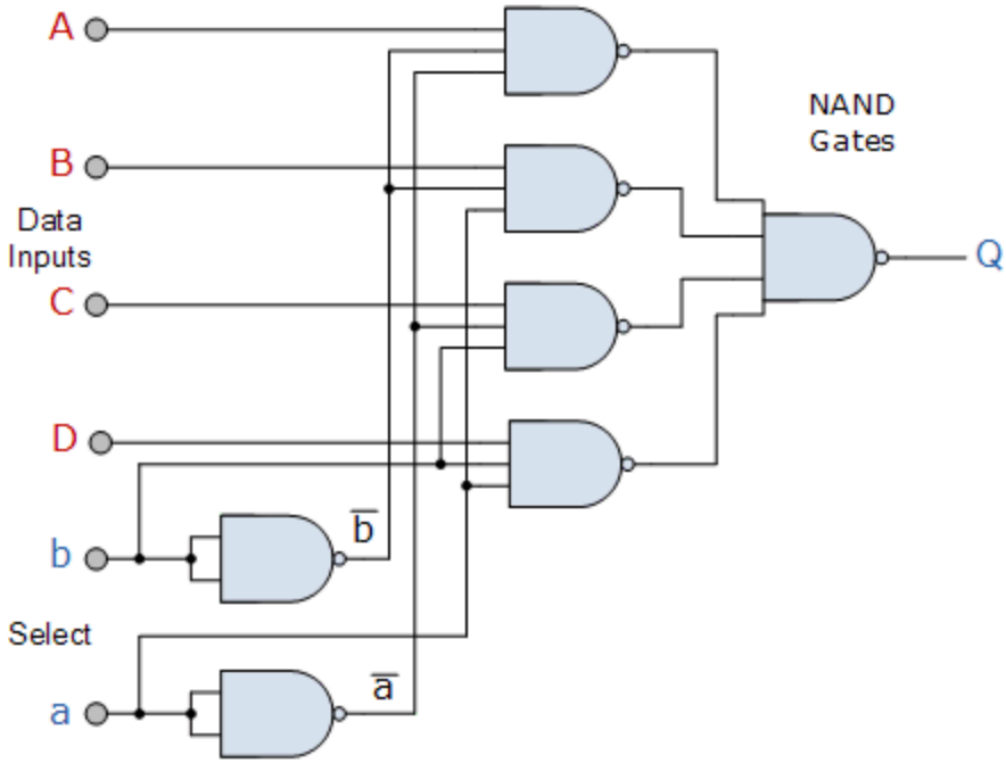
Images are taken from https://www.electronics-tutorials.ws/combination/comb_2.html

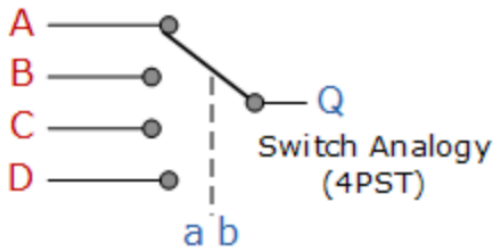
2-to-*4* Decoder by Using NAND Gates



4-to-*2* Encoder is an inverted diagram of the one above

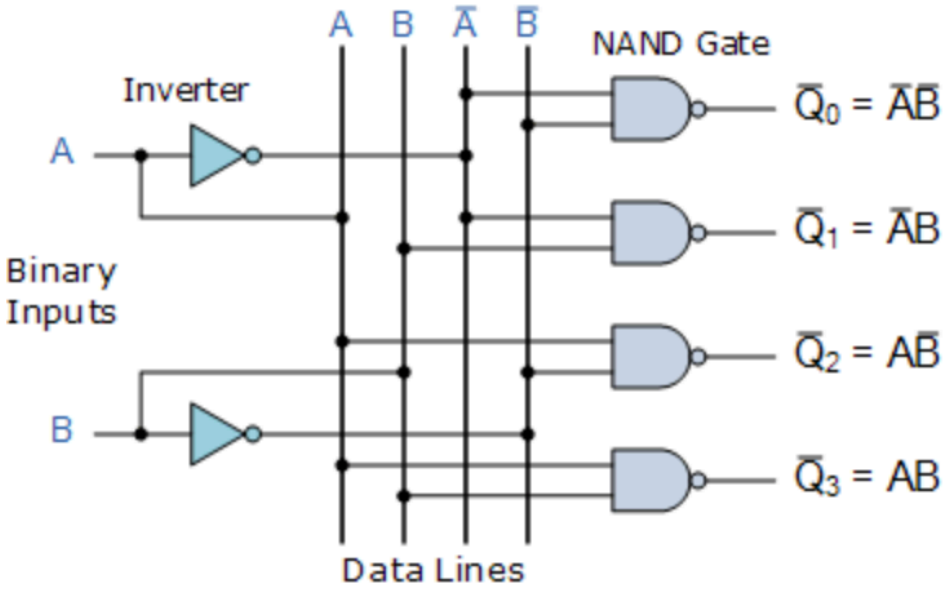
4-to-*1* Multiplexer by Using NAND Gates





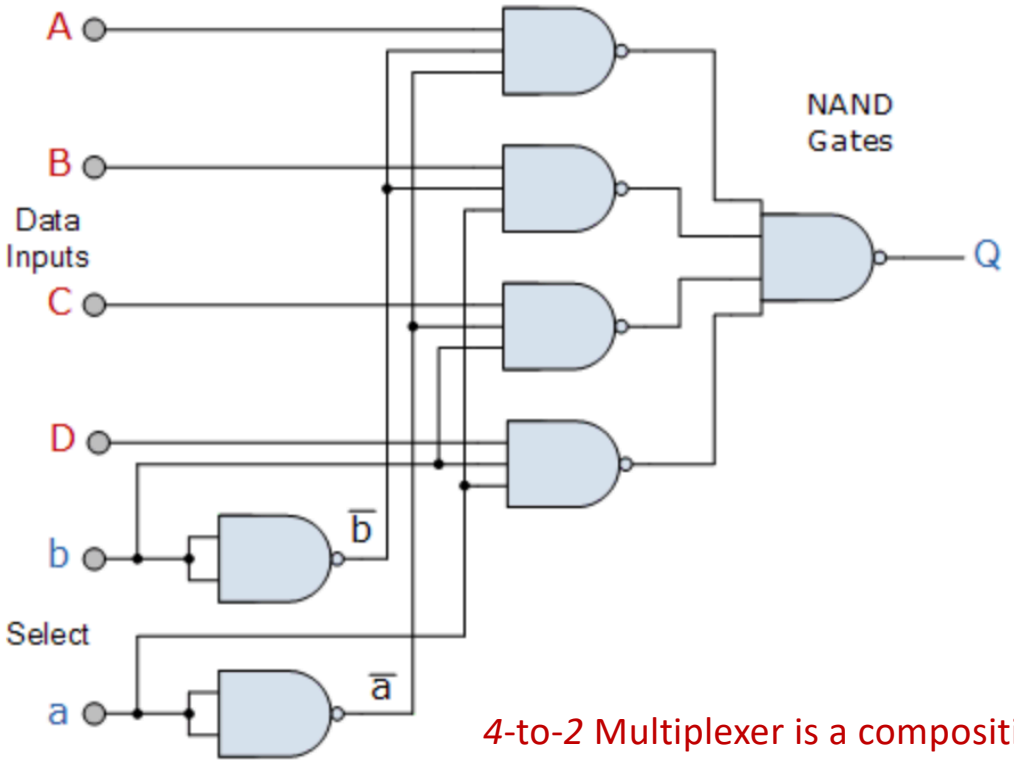
Images are taken from https://www.electronics-tutorials.ws/combination/comb_2.html

2-to-*4* Decoder by Using NAND Gates

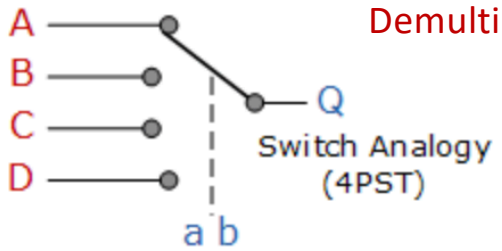


4-to-*2* Encoder is an inverted diagram of the one above

4-to-*1* Multiplexer by Using NAND Gates

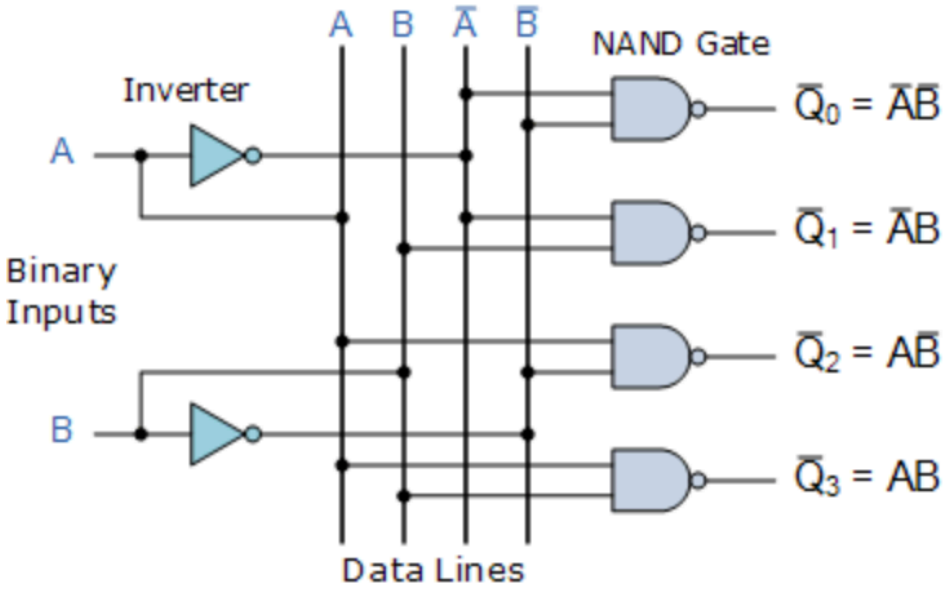


4-to-*2* Multiplexer is a composition of 4-to-1 Multiplexer and 1-to-2 Demultiplexer



Images are taken from https://www.electronics-tutorials.ws/combination/comb_2.html

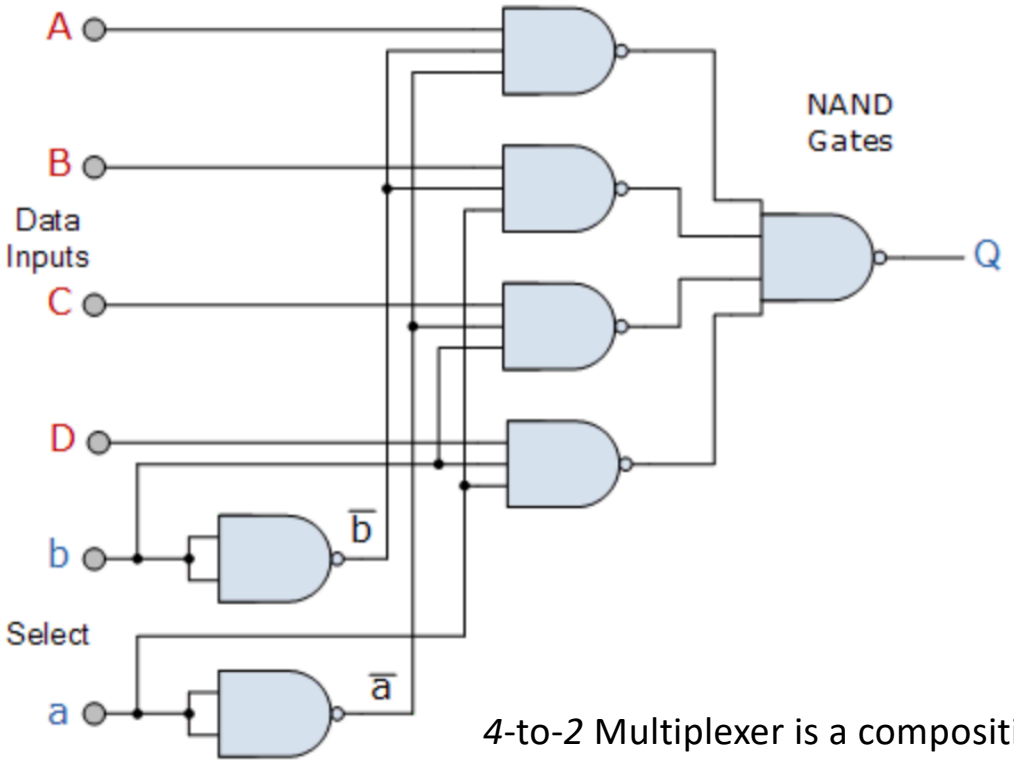
2-to-*4* Decoder by Using NAND Gates



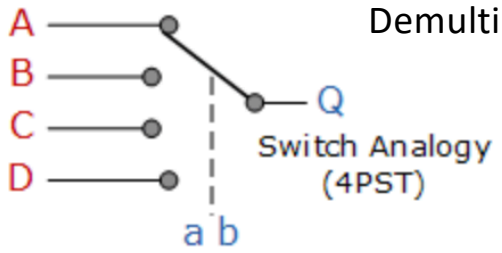
4-to-*2* Encoder is an inverted diagram of the one above

The implementation of a X-to-Y decoder (or encoder) uses significantly less logic gates, and is expected to have a lower propagation delay

4-to-*1* Multiplexer by Using NAND Gates



4-to-*2* Multiplexer is a composition of 4-to-1 Multiplexer and 1-to-2 Demultiplexer



Images are taken from https://www.electronics-tutorials.ws/combination/comb_2.html