2IC30: Computer Systems

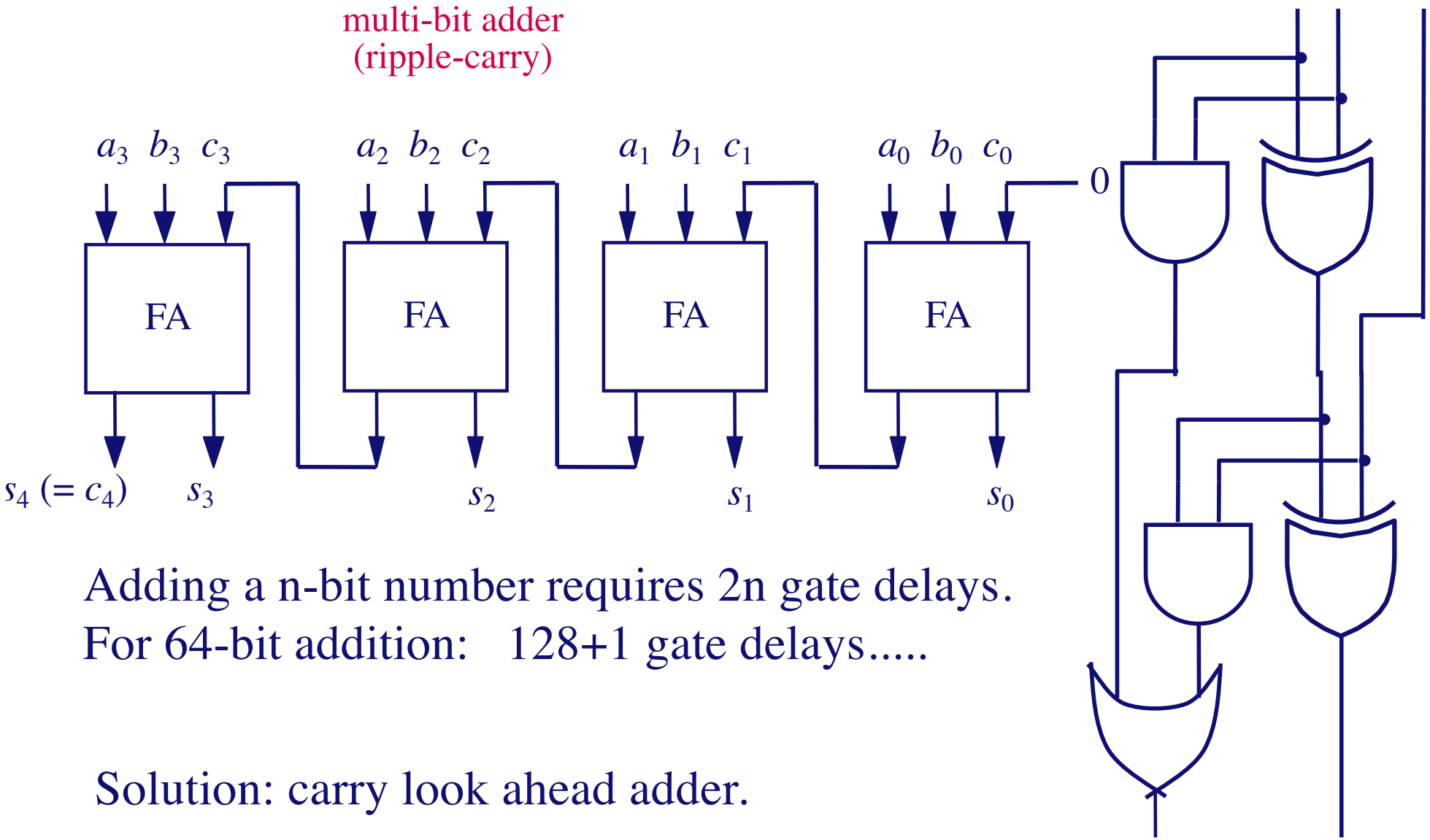
Flip-flops and registers.

Jan Friso Groote

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Binary addition of n-bit numbers



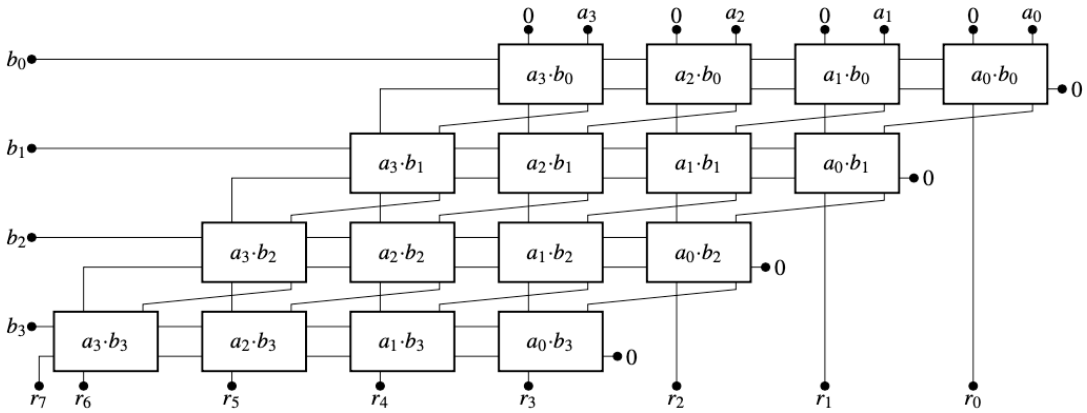
Adding a n-bit number requires 2n gate delays.
For 64-bit addition: 128+1 gate delays.....

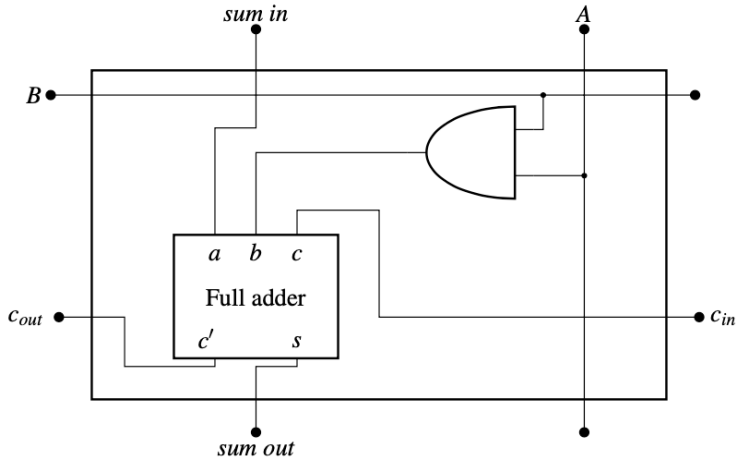
Solution: carry look ahead adder.

Multiplication.

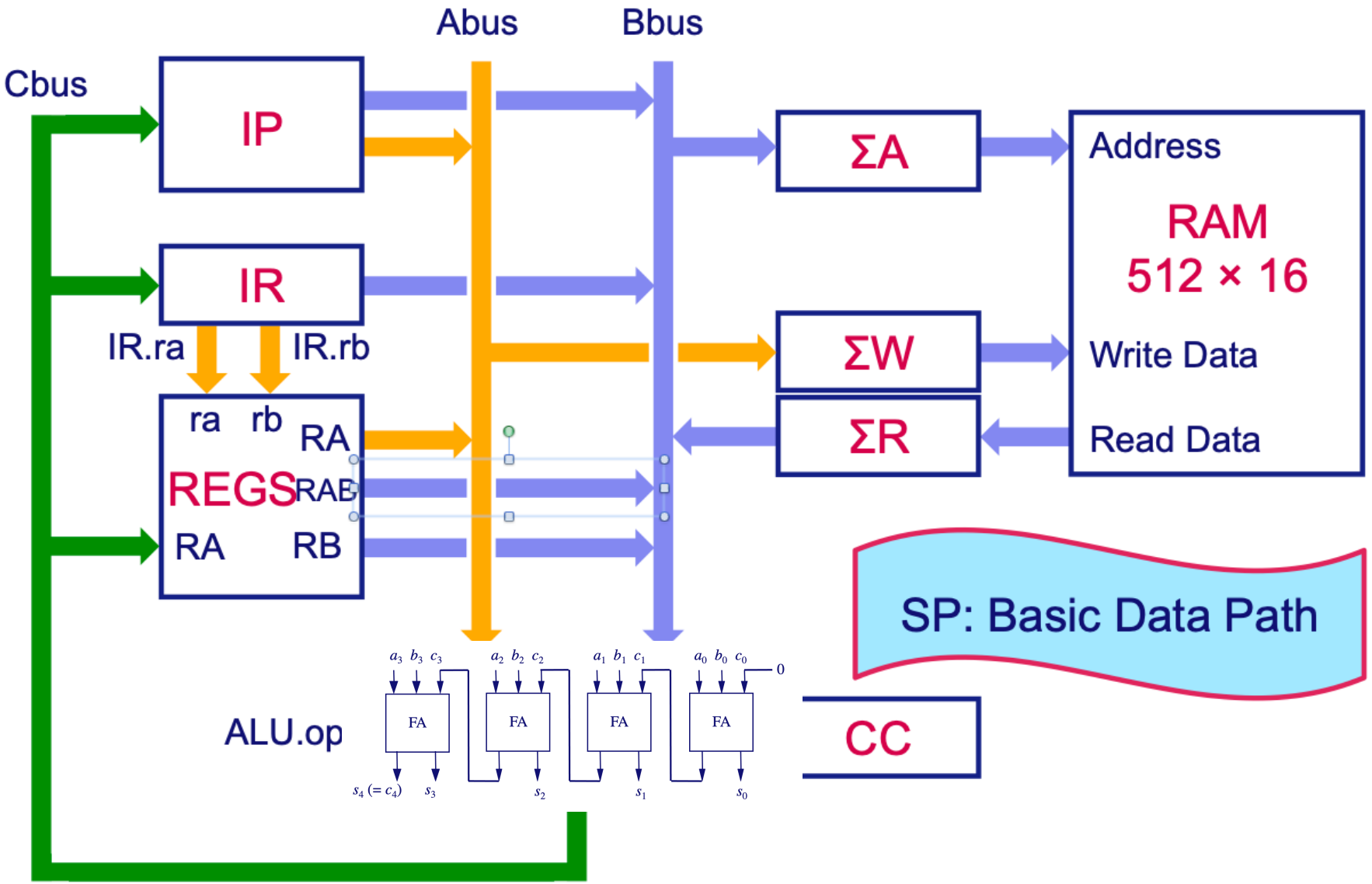
```
  1 0 1
  0 1 1  ×
---------
  1 0 1
1 0 1
0 0 0      +
---------
0 1 1 1 1
```

			a_3	a_2	a_1	a_0	
			b_3	b_2	b_1	b_0	
			$a_3 \wedge b_0$	$a_2 \wedge b_0$	$a_1 \wedge b_0$	$a_0 \wedge b_0$	
		$a_3 \wedge b_1$	$a_2 \wedge b_1$	$a_1 \wedge b_1$	$a_0 \wedge b_1$		+
	r_{51}	r_{41}	r_{31}	r_{21}	r_{11}	r_{01}	
	$a_3 \wedge b_2$	$a_2 \wedge b_2$	$a_1 \wedge b_2$	$a_0 \wedge b_2$			+
r_{62}	r_{52}	r_{42}	r_{32}	r_{22}	r_{12}	r_{02}	
$a_3 \wedge b_3$	$a_2 \wedge b_3$	$a_1 \wedge b_3$	$a_0 \wedge b_3$				+
r_{73}	r_{63}	r_{53}	r_{43}	r_{33}	r_{23}	r_{13}	r_{03}



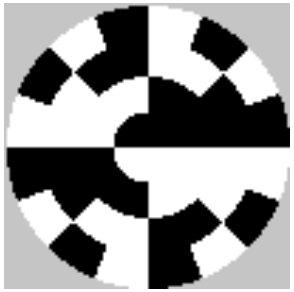


The adder is part of the ALU



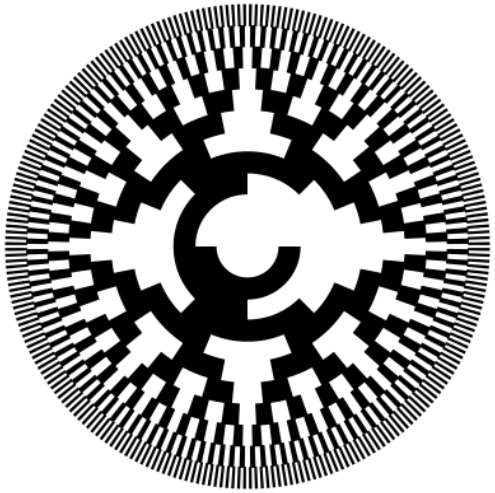
Other number systems: Gray codes

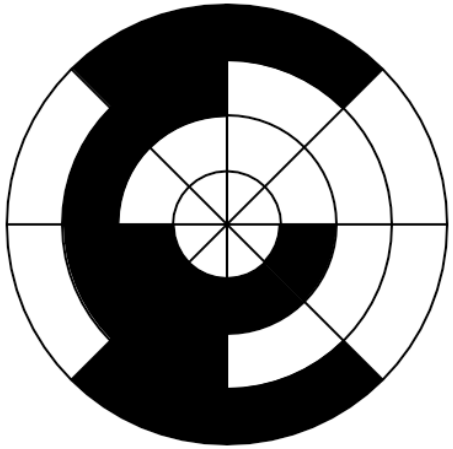
b_3	b_2	b_1	b_0	decimal	hexadecimal	octal
0	0	0	0	0	0	00
0	0	0	1	1	1	01
0	0	1	0	2	2	02
0	0	1	1	3	3	03
0	1	0	0	4	4	04
0	1	0	1	5	5	05
0	1	1	0	6	6	06
0	1	1	1	7	7	07
1	0	0	0	8	8	10
1	0	0	1	9	9	11
1	0	1	0	10	A	12
1	0	1	1	11	B	13
1	1	0	0	12	C	14
1	1	0	1	13	D	15
1	1	1	0	14	E	16
1	1	1	1	15	F	17



Other number systems: Gray codes

b_3	b_2	b_1	b_0	decimal	hexadecimal	octal
0	0	0	0	0	0	00
0	0	0	1	1	1	01
0	0	1	1	2	2	02
0	0	1	0	3	3	03
0	1	1	0	4	4	04
0	1	1	1	5	5	05
0	1	0	1	6	6	06
0	1	0	0	7	7	07
1	1	0	0	8	8	10
1	1	0	1	9	9	11
1	1	1	1	10	A	12
1	1	1	0	11	B	13
1	0	1	0	12	C	14
1	0	1	1	13	D	15
1	0	0	1	14	E	16
1	0	0	0	15	F	17





Error detecting codes: add a parity bit

b_3	b_2	b_1	b_0	parity bit
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

If the parity bit is incorrect data, the data is corrupted.

More expensive memory has parity bits built in.

IBM is contemplating adding parities to all aspects of safety critical processors

Error correcting codes: Hamming codes.

By adding a few more bits to each word, some bits can even be corrected.

In the table below it is indicated how much extra information is required to correct one bit.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X		...
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

source: Wikipedia.

Other codes.

Self delimiting codes: first indicate how long a number is.

1 0 1 0 1 1 - 1 0 0 1 0 1 1 75

Red bits are end of header indicators. Other initial bits indicate length of data to follow. In this case the length is 7 bits, with a header of 6 bits.

ASCII code for characters

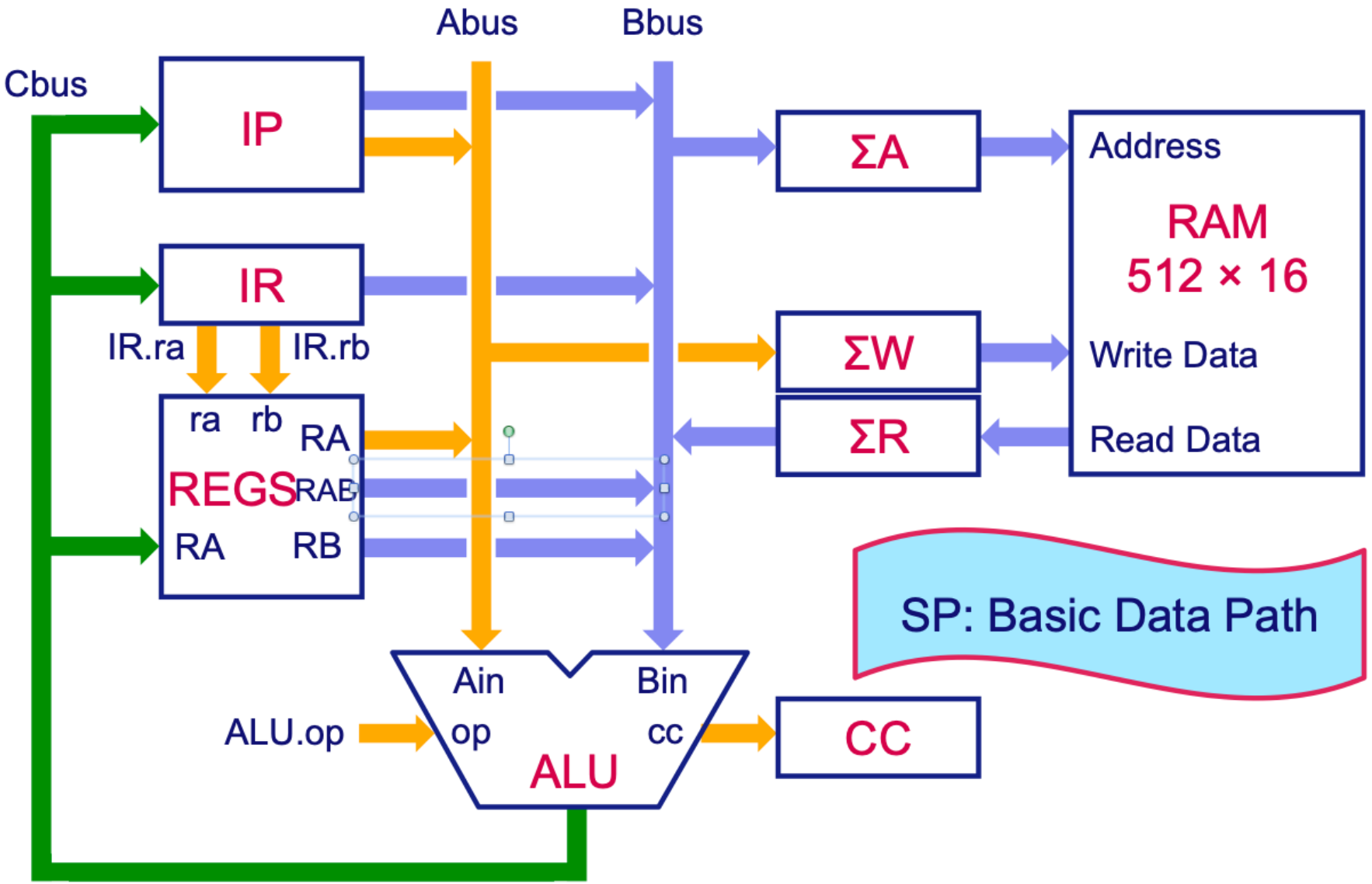
Unicode: 64 bit representation for characters.

Huffman codes: assign shorter bit sequences to more often occurring objects.

Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char	
0x00	0	NULL	null	0x20	32	Space		0x40	64	@		0x60	96	`	
0x01	1	SOH	Start of heading	0x21	33	!		0x41	65	A		0x61	97	a	
0x02	2	STX	Start of text	0x22	34	"		0x42	66	B		0x62	98	b	
0x03	3	ETX	End of text	0x23	35	#		0x43	67	C		0x63	99	c	
0x04	4	EOT	End of transmission	0x24	36	$		0x44	68	D		0x64	100	d	
0x05	5	ENQ	Enquiry	0x25	37	%		0x45	69	E		0x65	101	e	
0x06	6	ACK	Acknowledge	0x26	38	&		0x46	70	F		0x66	102	f	
0x07	7	BELL	Bell	0x27	39	'		0x47	71	G		0x67	103	g	
0x08	8	BS	Backspace	0x28	40	(		0x48	72	H		0x68	104	h	
0x09	9	TAB	Horizontal tab	0x29	41	)		0x49	73	I		0x69	105	i	
0x0A	10	LF	New line	0x2A	42	*		0x4A	74	J		0x6A	106	j	
0x0B	11	VT	Vertical tab	0x2B	43	+		0x4B	75	K		0x6B	107	k	
0x0C	12	FF	Form Feed	0x2C	44	,		0x4C	76	L		0x6C	108	l	
0x0D	13	CR	Carriage return	0x2D	45	-		0x4D	77	M		0x6D	109	m	
0x0E	14	SO	Shift out	0x2E	46	.		0x4E	78	N		0x6E	110	n	
0x0F	15	SI	Shift in	0x2F	47	/		0x4F	79	O		0x6F	111	o	
0x10	16	DLE	Data link escape	0x30	48	0		0x50	80	P		0x70	112	p	
0x11	17	DC1	Device control 1	0x31	49	1		0x51	81	Q		0x71	113	q	
0x12	18	DC2	Device control 2	0x32	50	2		0x52	82	R		0x72	114	r	
0x13	19	DC3	Device control 3	0x33	51	3		0x53	83	S		0x73	115	s	
0x14	20	DC4	Device control 4	0x34	52	4		0x54	84	T		0x74	116	t	
0x15	21	NAK	Negative ack	0x35	53	5		0x55	85	U		0x75	117	u	
0x16	22	SYN	Synchronous idle	0x36	54	6		0x56	86	V		0x76	118	v	
0x17	23	ETB	End transmission block	0x37	55	7		0x57	87	W		0x77	119	w	
0x18	24	CAN	Cancel	0x38	56	8		0x58	88	X		0x78	120	x	
0x19	25	EM	End of medium	0x39	57	9		0x59	89	Y		0x79	121	y	
0x1A	26	SUB	Substitute	0x3A	58	:		0x5A	90	Z		0x7A	122	z	
0x1B	27	FSC	Escape	0x3B	59	;		0x5B	91	[		0x7B	123	{	
0x1C	28	FS	File separator	0x3C	60	<		0x5C	92	\		0x7C	124	|	
0x1D	29	GS	Group separator	0x3D	61	=		0x5D	93	]		0x7D	125	}	
0x1E	30	RS	Record separator	0x3E	62	>		0x5E	94	^		0x7E	126	~	
0x1F	31	US	Unit separator	0x3F	63	?		0x5F	95	_		0x7F	127	DEL	

Questions?

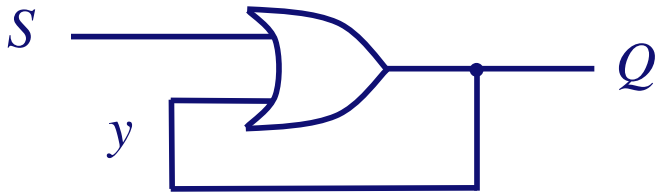
A microprocessor stores data in registers



Remembering the past: Circuits with States

- Circuits with feedback:
  - Set-Reset Latch/Flip-Flop
  - Level triggered Flip-Flop: D-Latch
  - Edge triggered Flip-Flop: D-Flip-Flop

A simple memory cell



simple memory cell

S	y	Q		
0	0	0	$\Leftarrow$	*stable*: $y = 0$
0	1	1	$\Leftarrow$	*stable*: $y = 1$
1	0	1	$\Leftarrow$	*unstable*: $y := 1$
1	1	1	$\Leftarrow$	*stable*: $y = 1$

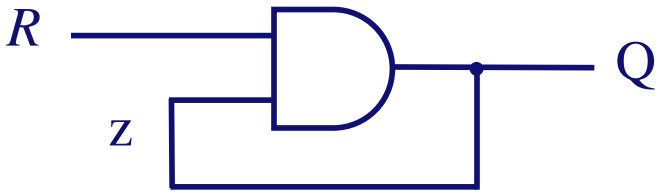
Idea:

- **if** (initially) $Q = 0$ and $S = 0$ **then** Q "remains" 0
- **if** $S = 1$ **then** Q "becomes" 1
- **if** $Q = 1$ **then** Q "remains" 1 (forever…)
- example of *positive feedback* : no signal inversions
- so: $Q = 1$ "remembers" that, once upon a time in the past, S has been 1 for a while

Problem:

- once $Q = 1$ it will stay 1: how to *reset* Q back to 0 ?

Duality: another simple memory cell



simple memory cell

R	z	Q		
0	0	0	$\Leftarrow$	*stable*: $z = 0$
0	1	0	$\Leftarrow$	*unstable*: $z := 0$
1	0	0	$\Leftarrow$	*stable*: $z = 0$
1	1	1	$\Leftarrow$	*stable*: $z = 1$

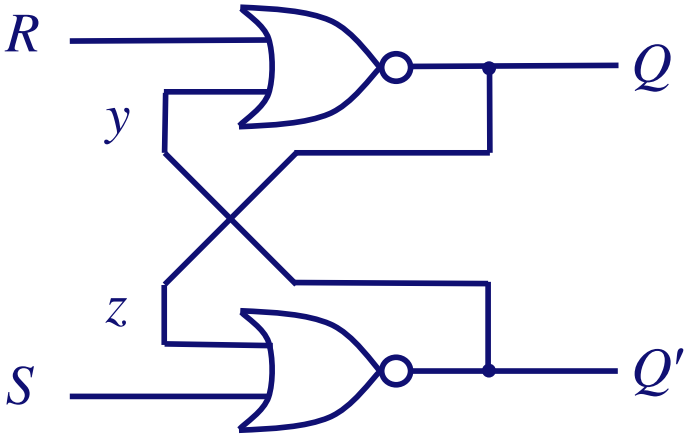
Idea:

- **if** (initially) $Q = 1$ and $R = 1$ **then** Q "remains" 1
- **if** $R = 0$ **then** Q "becomes" 0
- **if** $Q = 0$ **then** Q "remains" 0 (forever…)
- example of *positive feedback* : no signal inversions
- so: $Q = 0$ "remembers" that, once upon a time in the past, R has been 0 for a while

Problem:

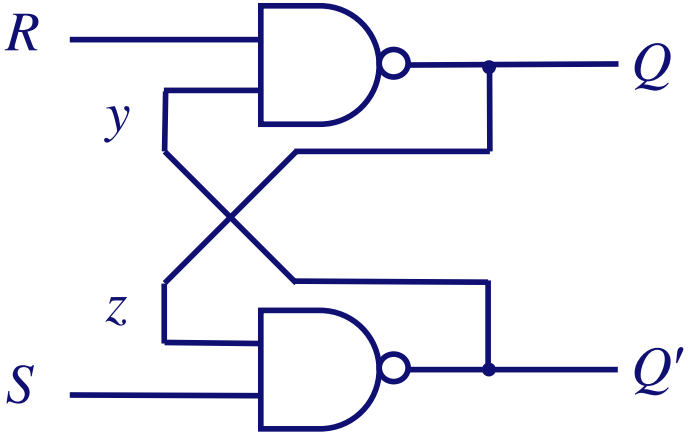
- once $Q = 0$ it will stay 0 : how to *reset* Q back to 1 ?

The Set-Reset Latch (S-R Latch/Flip Flop)



S-R Latch with NORs

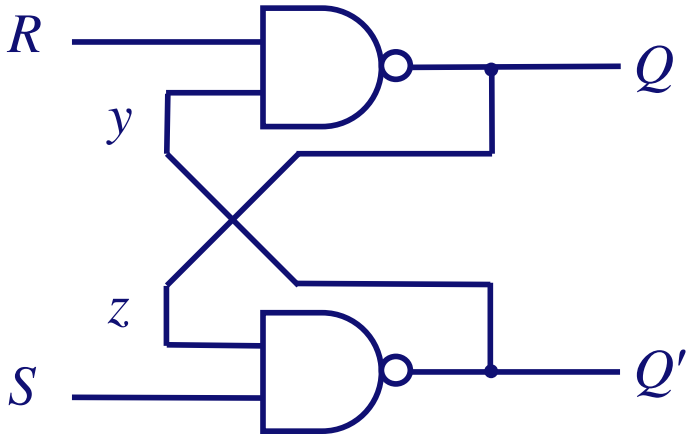
S	R	Q	Q'
0	0	Q	Q'
0	1	0	1
1	0	1	0
1	1	0	0



S-R Latch with NANDs

S	R	Q	Q'
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	Q'

Problems with the S-R Latch



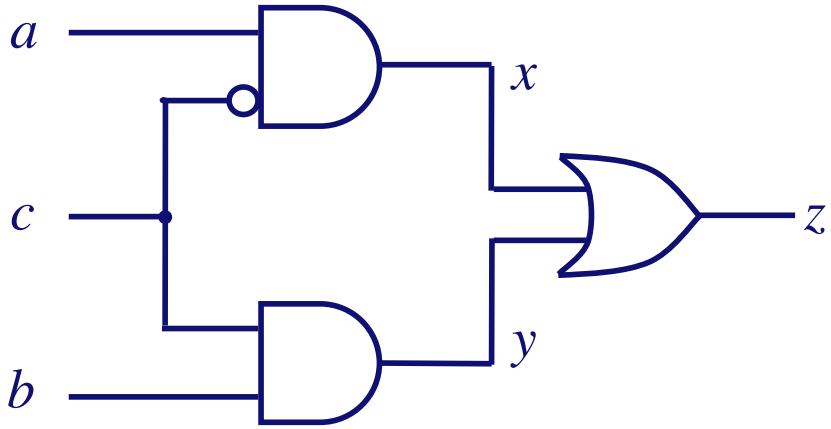
S-R Latch with NANDs

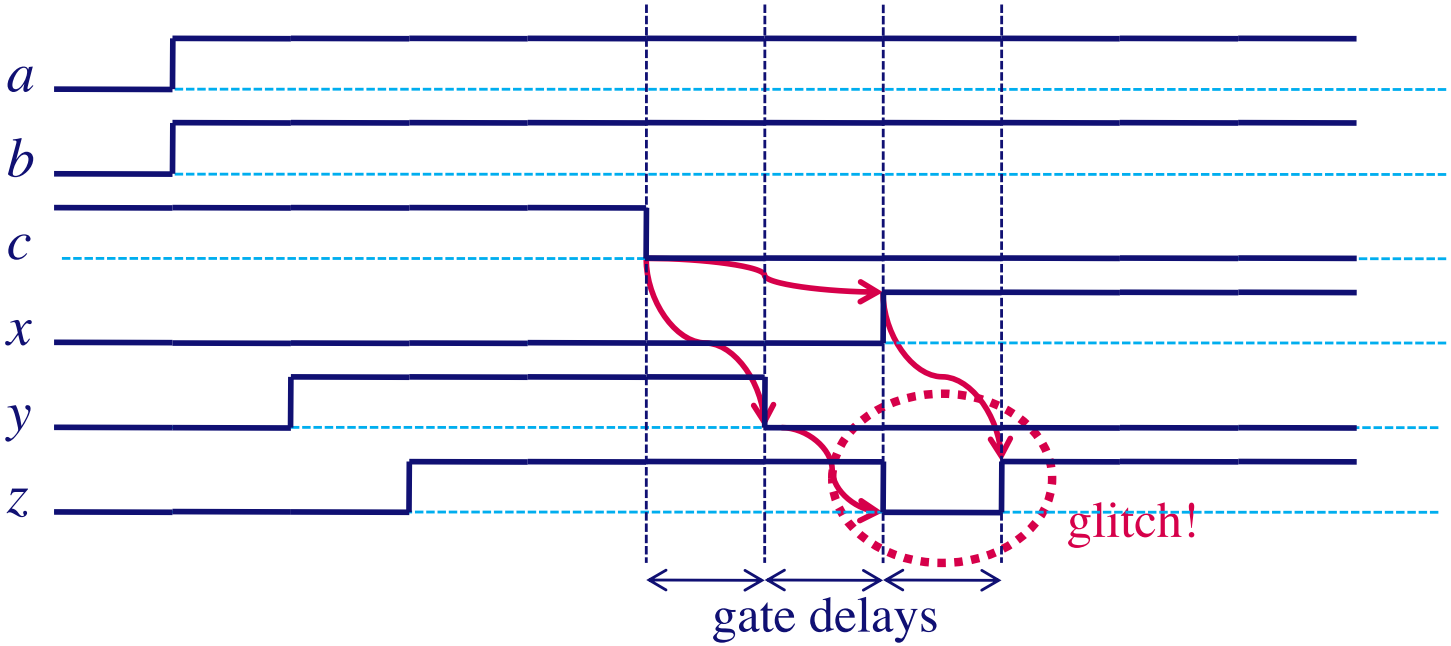
S	R	Q	Q'
0	0	1	1
0	1	0	1
1	0	1	0
1	1	Q	Q'

- Glitches: Unintended temporary changes in S or R may cause unintended state changes in Q (and Q').

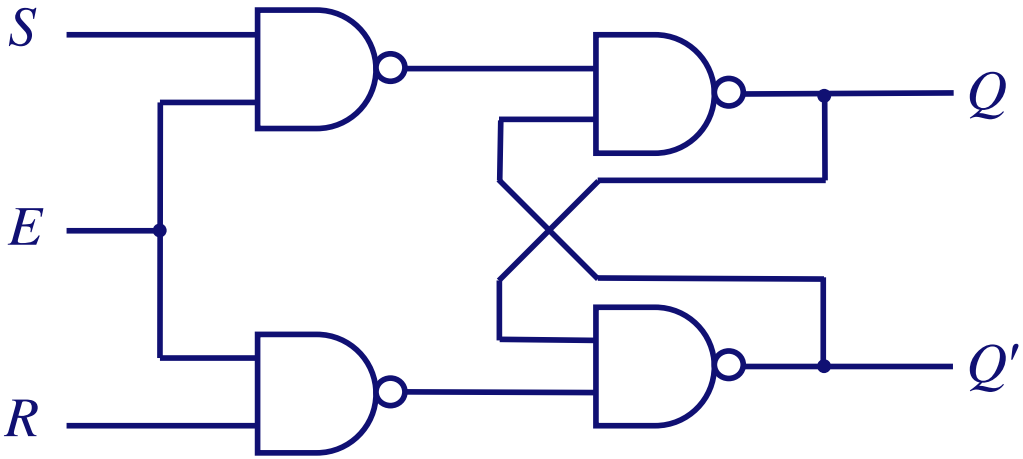
- Metastability: Unpredictable behaviour when S and R make the transition from 0,0 to 1,1 *simultaneously*.

Glitches: temporary signal deviations





Gated S-R Latch

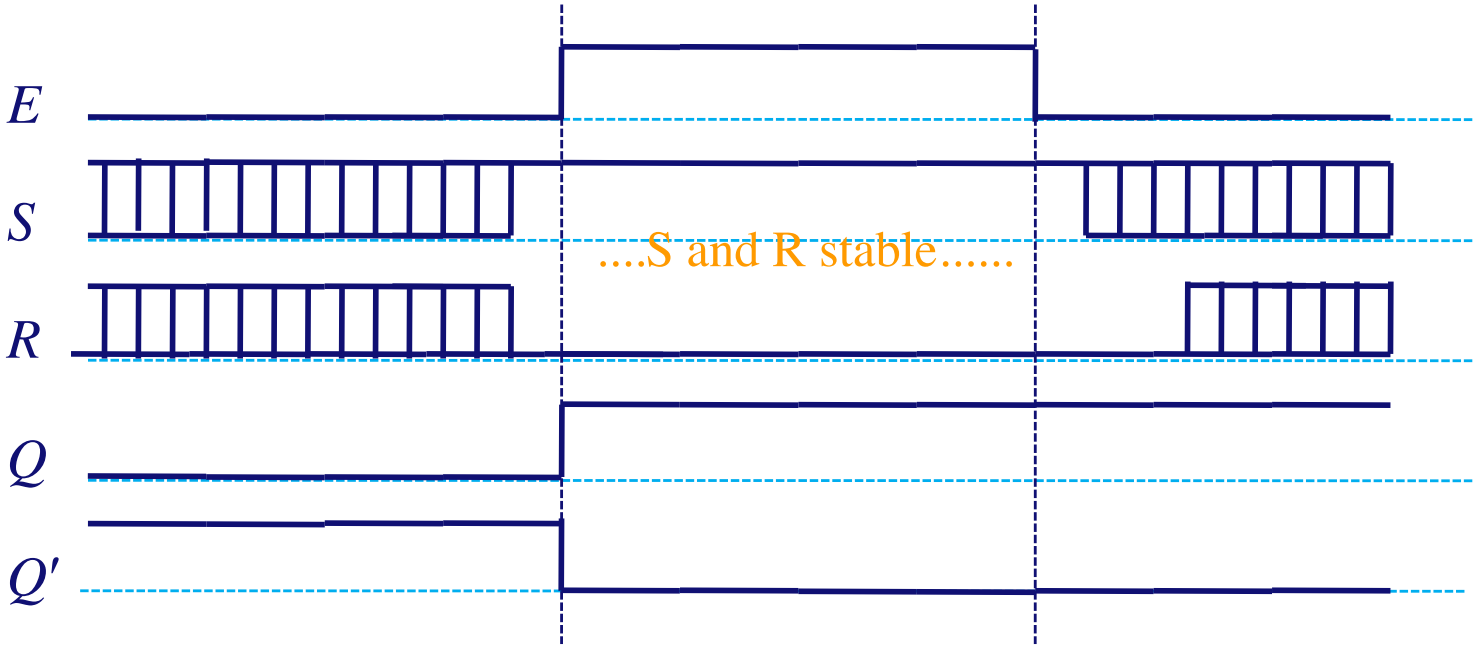


- **if** $E = 0$ **then** Q and Q' are stable and not sensitive to (changes in) S and R; glitches on S or R are harmless
- **if** $E \wedge S \wedge \neg R$ **then** $Q := 1$ and $Q' := 0$
- **if** $E \wedge \neg S \wedge R$ **then** $Q := 0$ and $Q' := 1$
- **if** $E \wedge S \wedge R$ **then** $Q := 1$ and $Q' := 1$: risk of *metastability*
- **if** $E \wedge \neg S \wedge \neg R$ **then** "no change in Q and Q' "

Requirements:

- to prevent *metastability*, $S \neq R$ is needed whenever $E = 1$

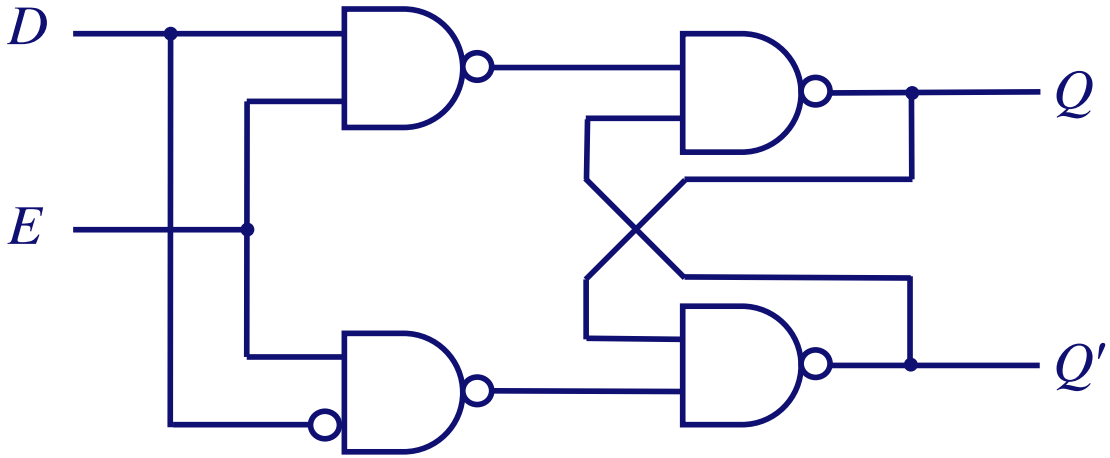
Gated S-R Latch: Timing diagram

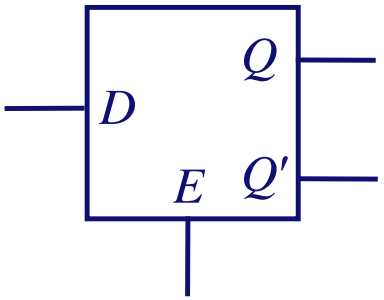


Disadvantage: Signals S and R may be equal and, hence, cause metastability problems.

Improvement: the gated D-Latch.

Gated D-Latch (or: Transparent Latch)





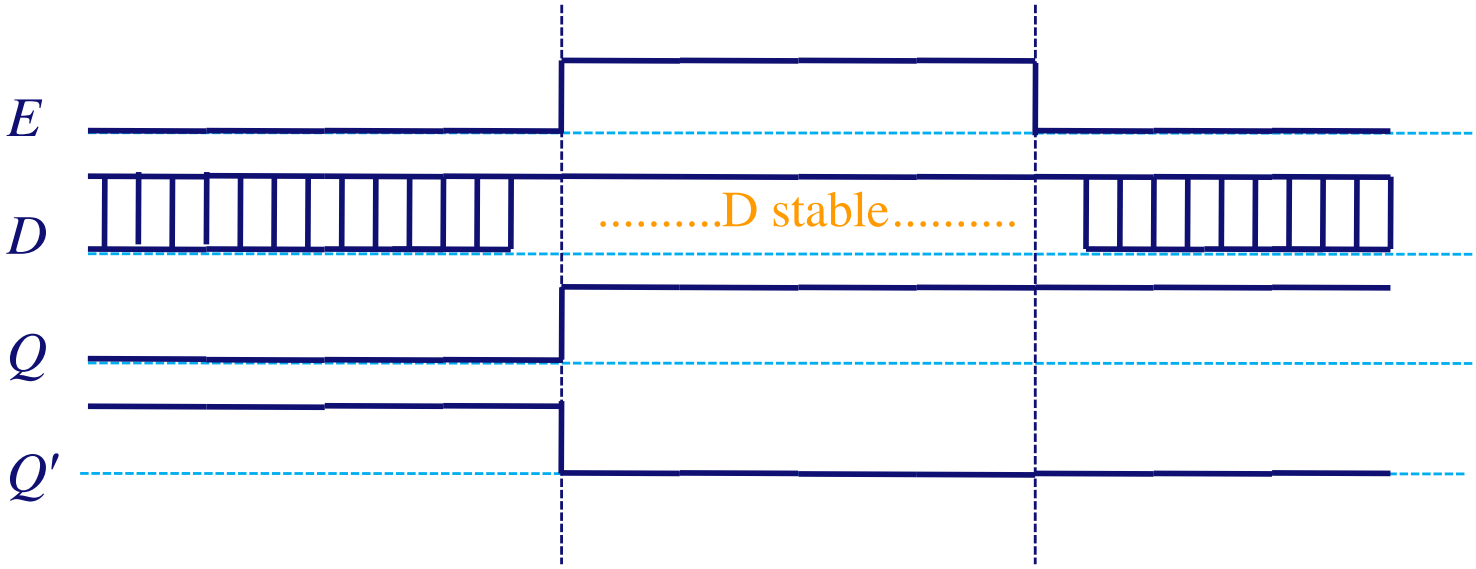
transparent D-latch

- Signals S and R are now *derived* from D, in such a way that always $S \neq R$.
- **If** $E = 0$ **then** Q and Q′ are stable and insensitive to changes in D; glitches on D are harmless.
- **If** $E = 1$ **then** $Q = D$ and $Q' = \neg D$: the outputs follow D.

Requirements:

- As long as $E = 1$ input D must be *stable* .

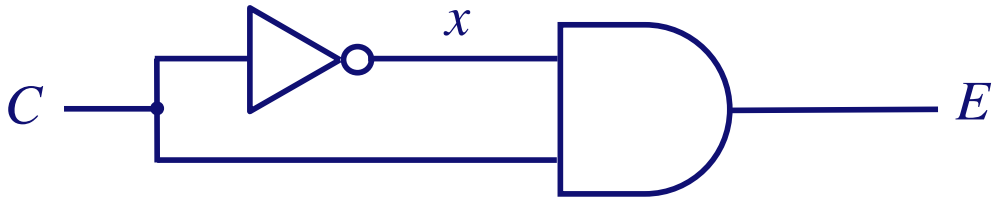
Gated D-Latch: Timing diagram



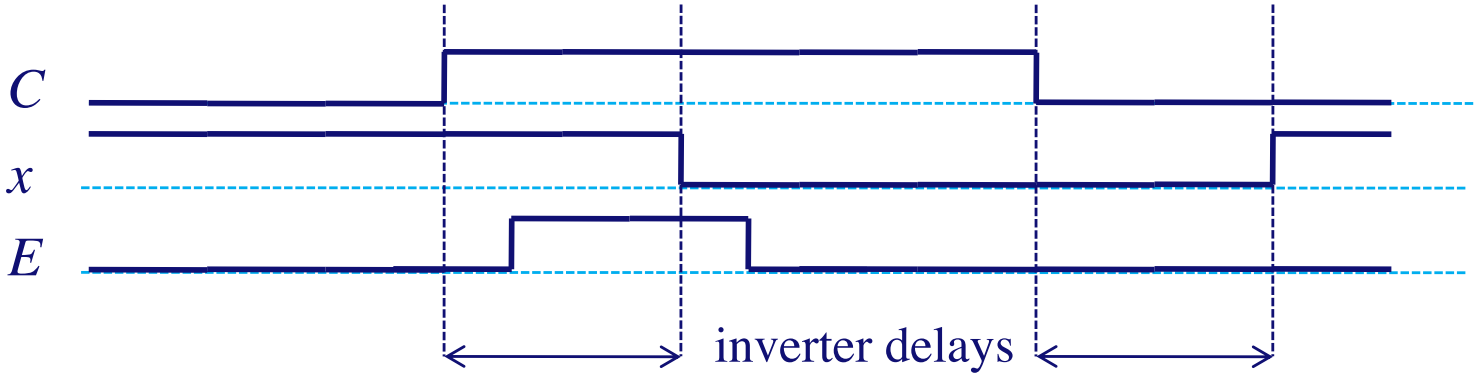
Disadvantage: When E remains 1 for a long time, D must remain stable for an equally long time.

Improvement: the Edge-Triggered D-Flip-Flop.

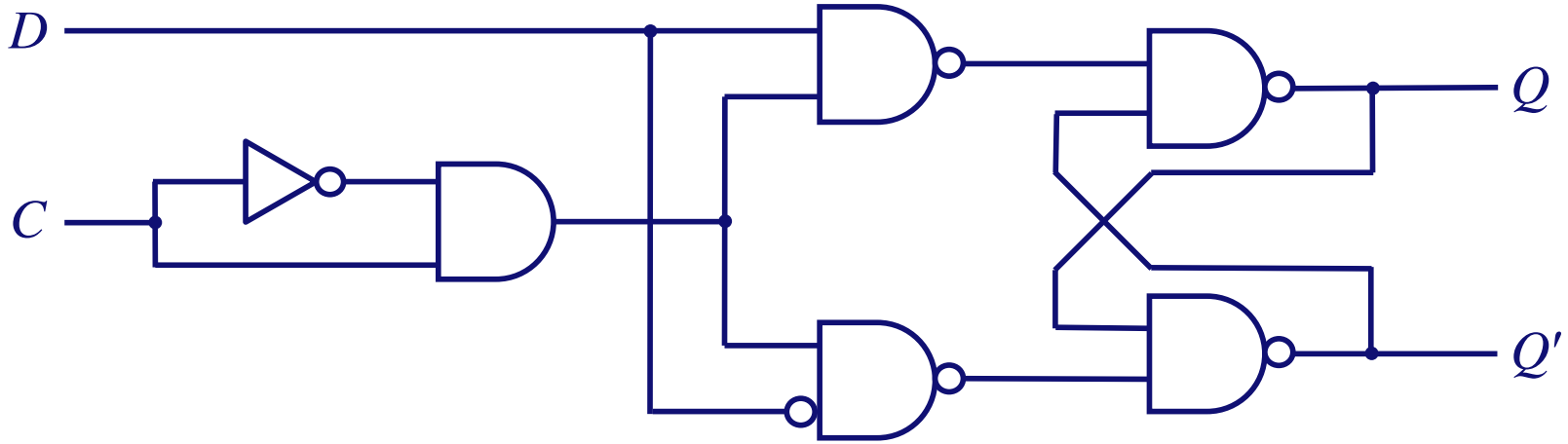
An Edge Detector, or: pulse shaper

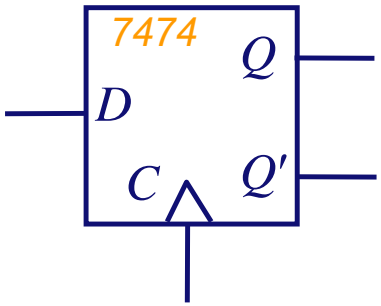


Exploiting the delay of a (slow) inverter,
that is: putting a glitch to good use.



Clocked D-Flip-Flop (or: Edge-Triggered FF)





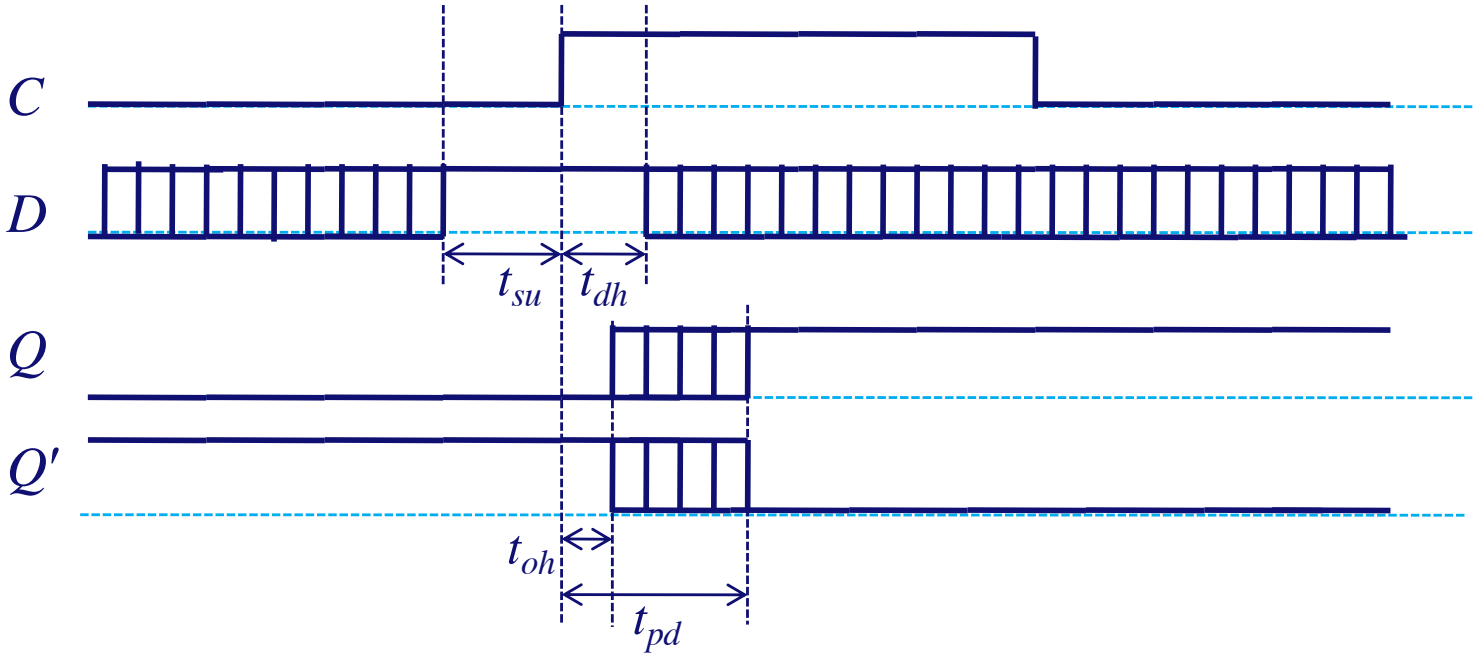
edge-triggered D-flip-flop

- **on** $C\uparrow$ **then** $Q := D$ and $Q' := \neg D$: the outputs copy D on every *upgoing transition* on C.

Requirement:

- Input D must be *stable* only during a short interval around $C\uparrow$: the *setup-and-hold interval*.

Clocked D-Flip-Flop: Timing diagram



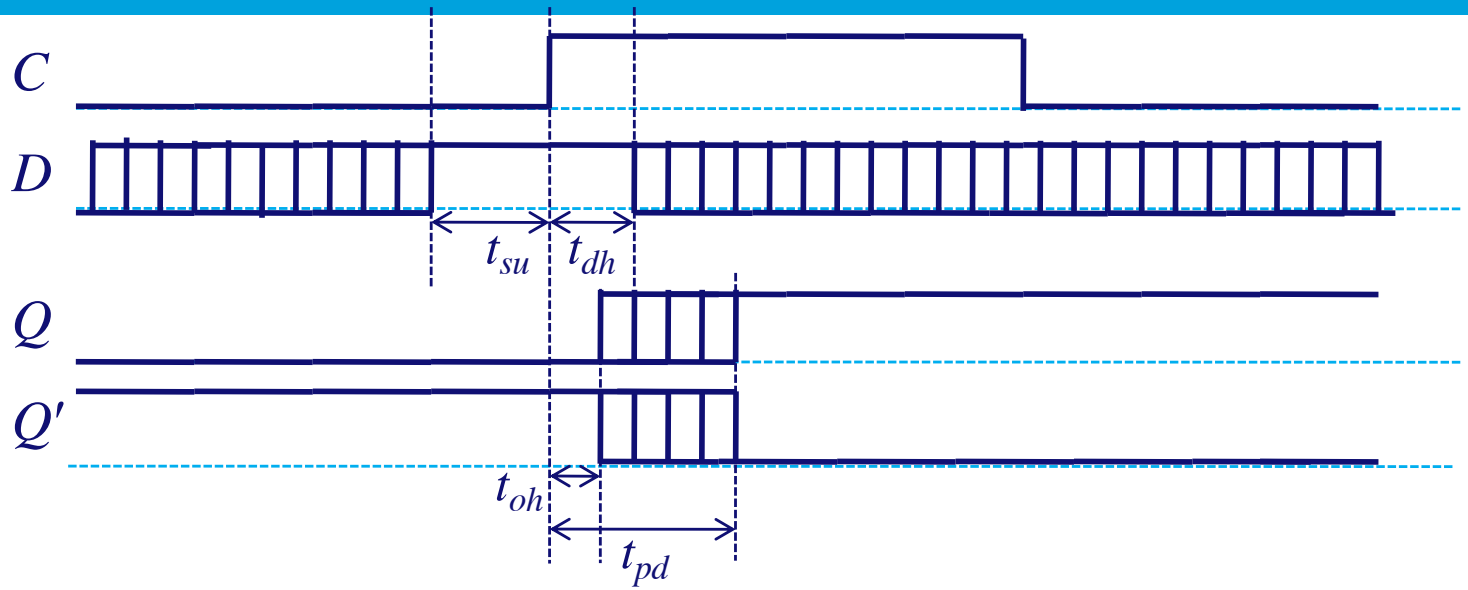
t_{su} (“setup time”): the *minimal* amount of time D must be stable *before* $C\uparrow$.

t_{dh} (“hold time”): the *minimal* amount of time D must be stable *after* $C\uparrow$.

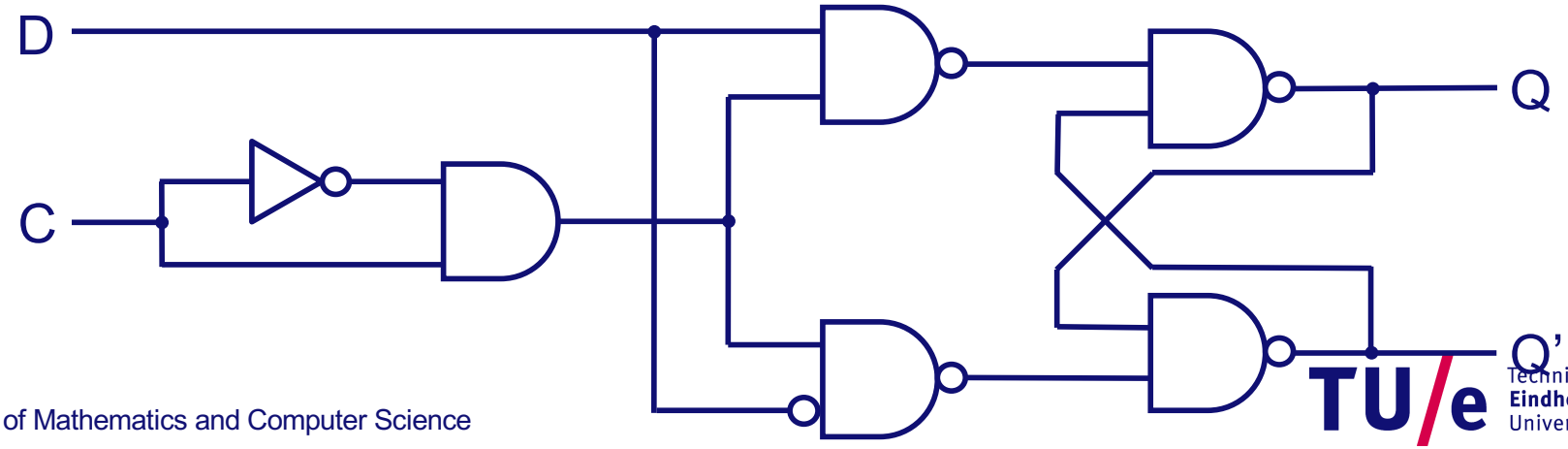
t_{oh} (“output hold time”): the *minimal* amount of time *after* $C\uparrow$ during which Q and Q' will retain their *old* values.

t_{pd} (“propagation delay”): the *maximal* amount of time *after* $C\uparrow$ before Q and Q' have obtained their *new* values.

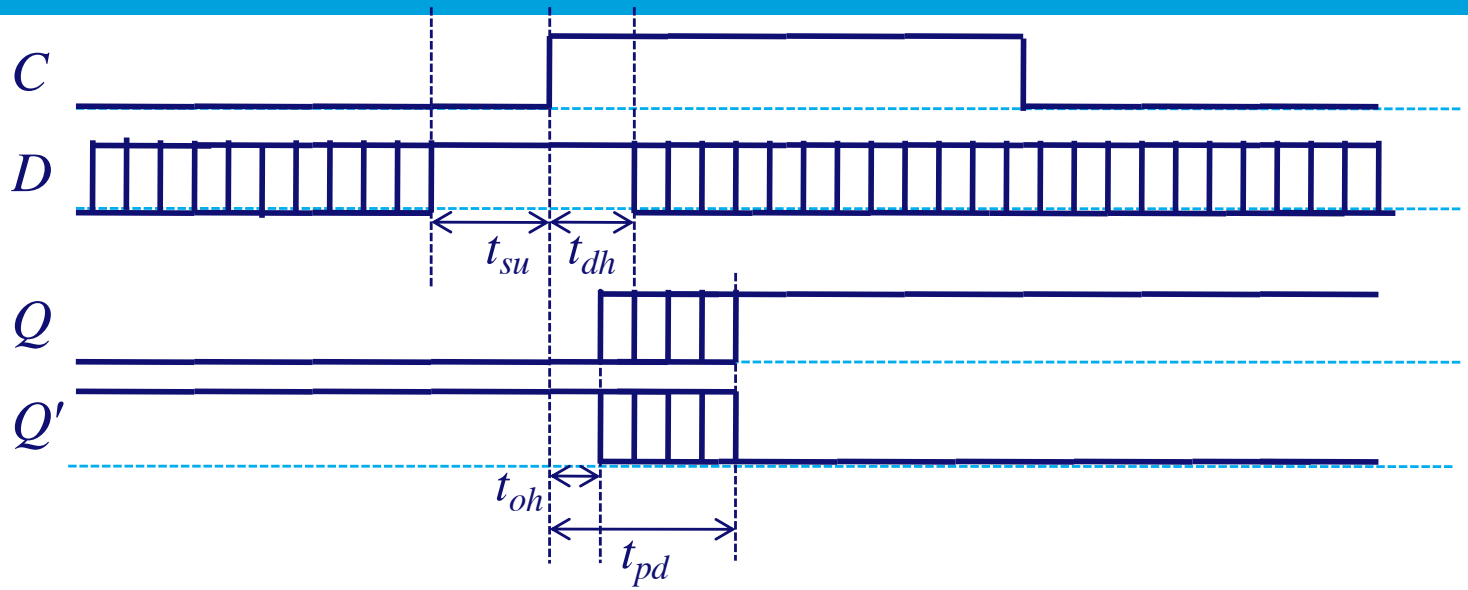
Clocked D-Flip-Flop: Timing properties



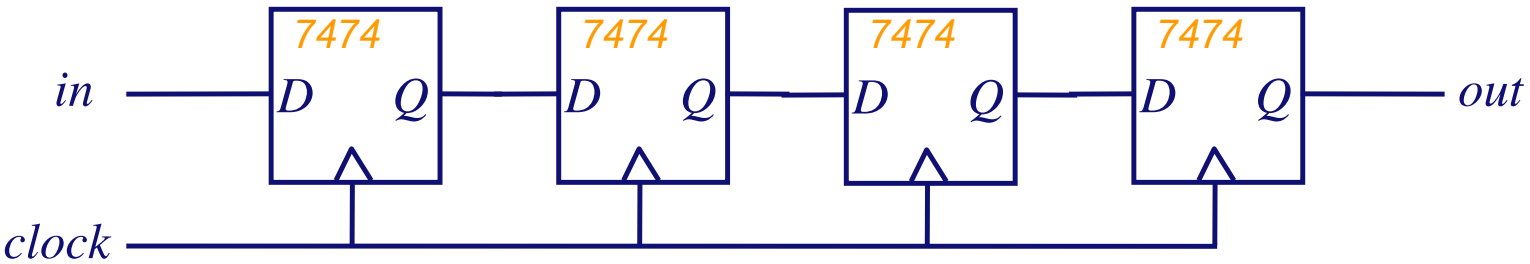
- t_{su} might be *negative*: why?
- *always*: $0 < t_{su} + t_{dh}$: why?
- *always*: $0 \leq t_{oh} \leq t_{pd}$: why?
- if $t_{dh} \leq t_{oh}$ D-Flip-Flops can be *cascaded* (and work correctly):



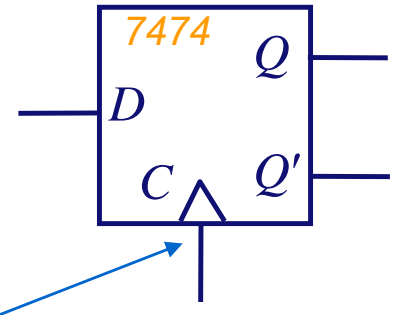
Clocked D-Flip-Flop: Timing properties



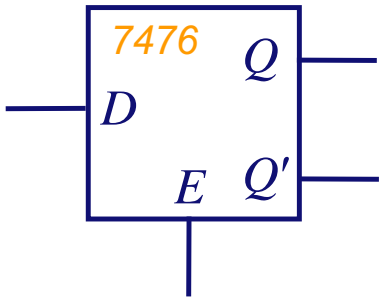
- t_{su} might be *negative*: why?
- *always*: $0 < t_{su} + t_{dh}$: why?
- *always*: $0 \leq t_{oh} \leq t_{pd}$: why?
- if $t_{dh} \leq t_{oh}$ D-Flip-Flops can be *cascaded* (and work correctly):



Latches versus Flip-Flops



edge-triggered D-flip-flop



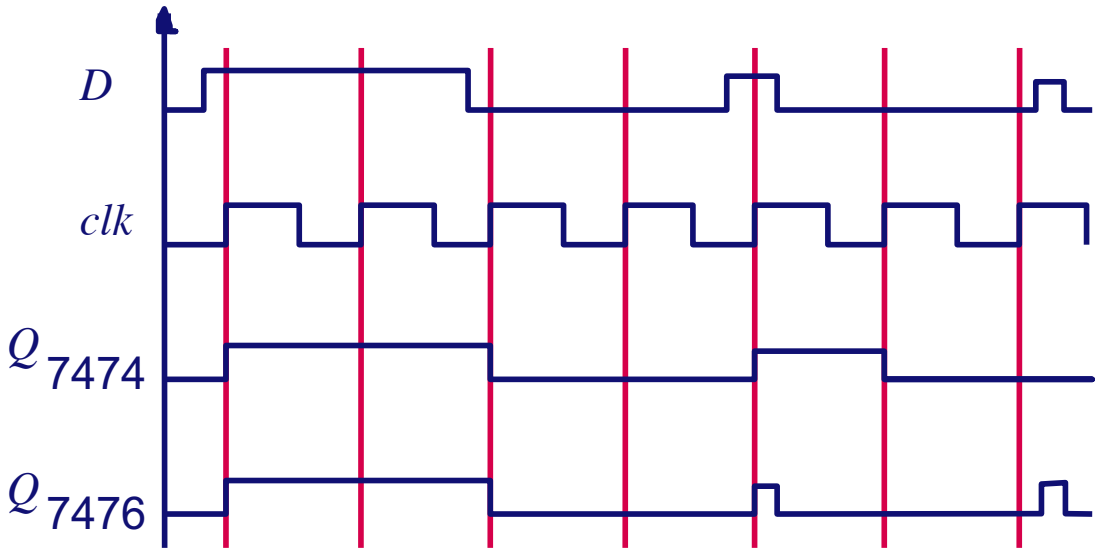
transparent D-latch

circle would indicate
a falling edge
triggered circuit

edge triggered flip-flops copy values during a rising (or: falling) edge of the clock

transparent latches copy values while the enable signal is 1

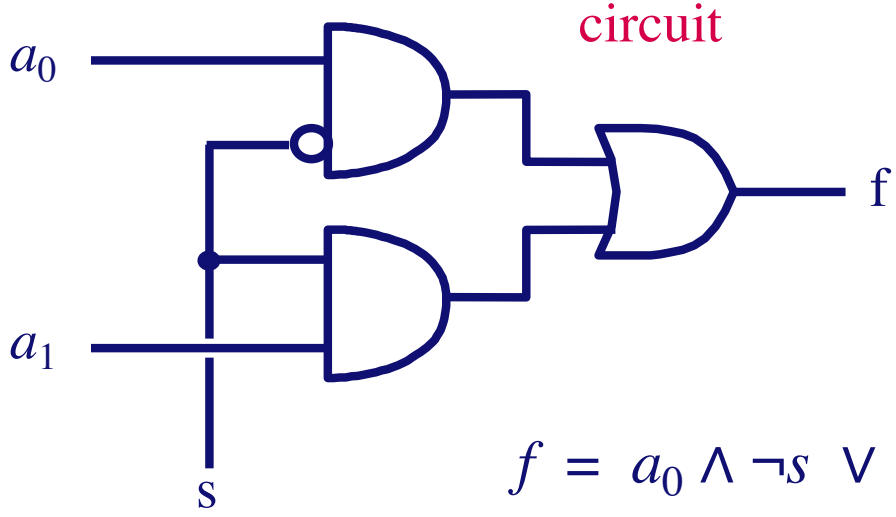
timing diagram:



behaviour is similar, except when input D changes while clk = 1

Questions?

2-input multiplexer (= data selector)



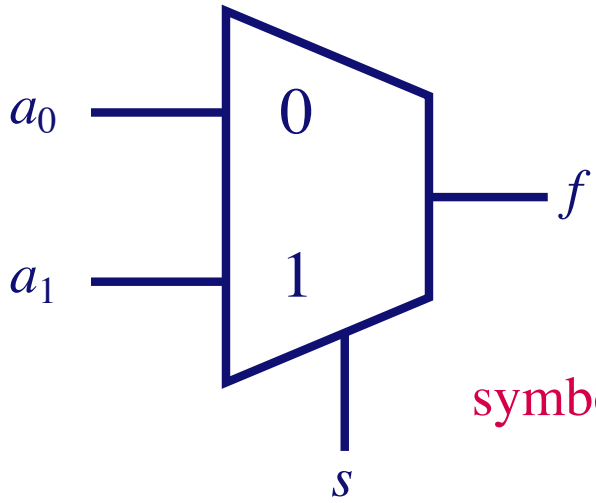
circuit

$$f = a_0 \wedge \neg s \vee a_1 \wedge s$$

$$f = \textit{if } s \textit{ then } a_1 \textit{ else } a_0$$

multiplexer

s	a_0	a_1	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



symbol

2-input multiplexer

s	f
0	a_0
1	a_1

4-input multiplexer (= data selector)

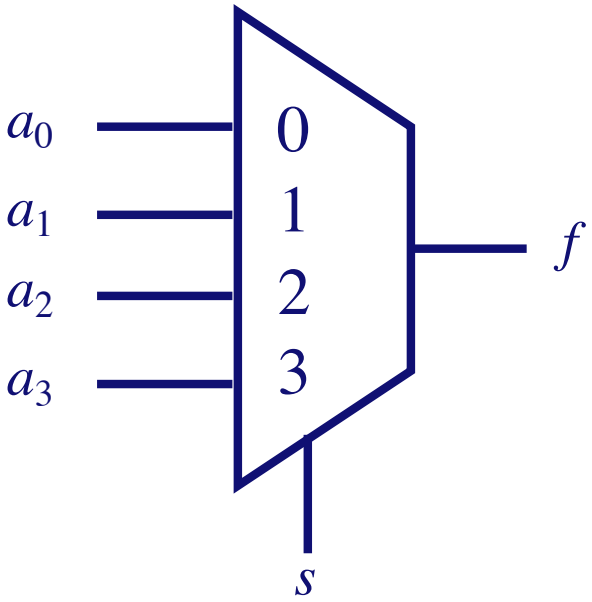
4-input multiplexer (abstract)

s	f
0	a_0
1	a_1
2	a_2
3	a_3

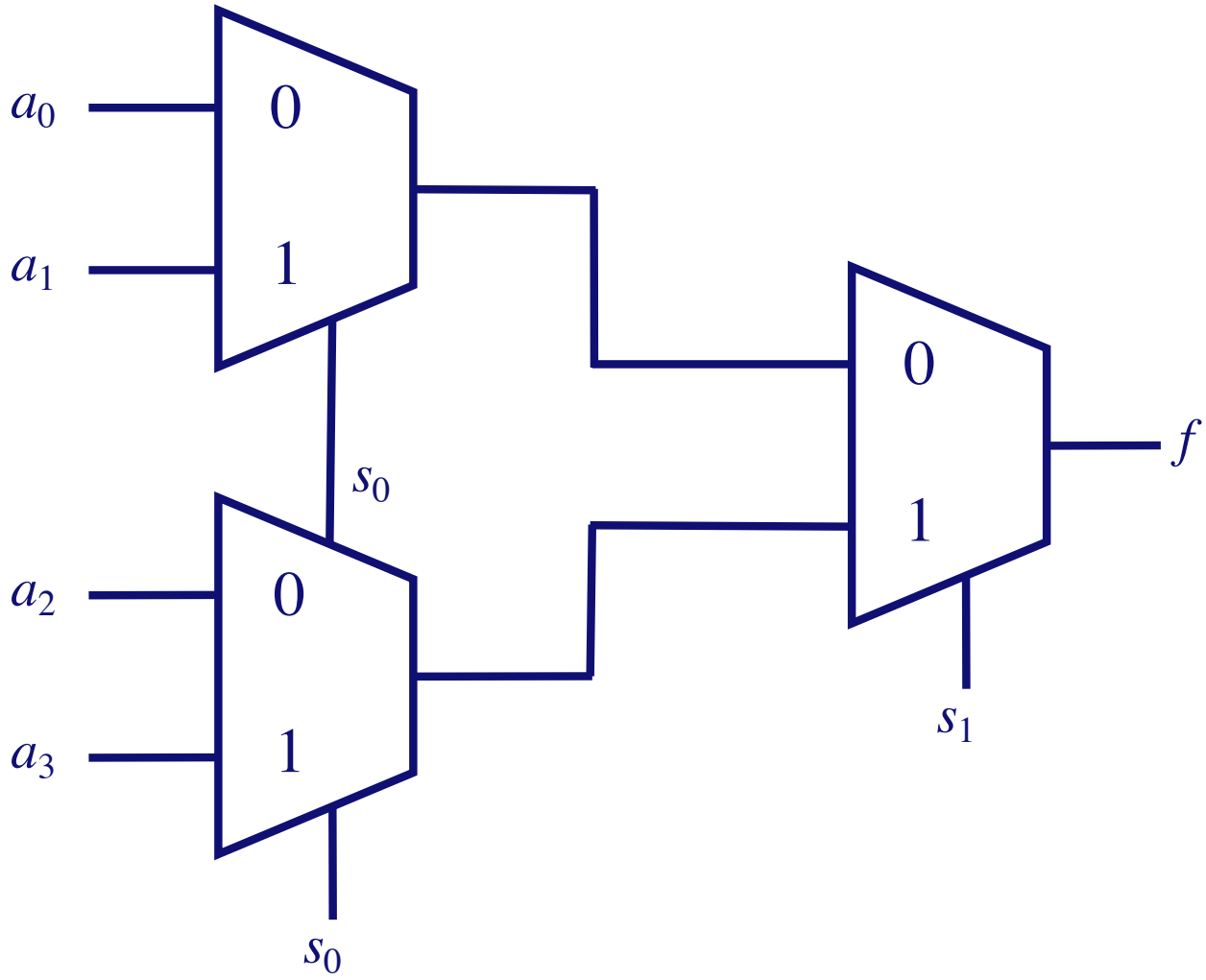
4-input multiplexer (binary)

s_1	s_0	f
0	0	a_0
0	1	a_1
1	0	a_2
1	1	a_3

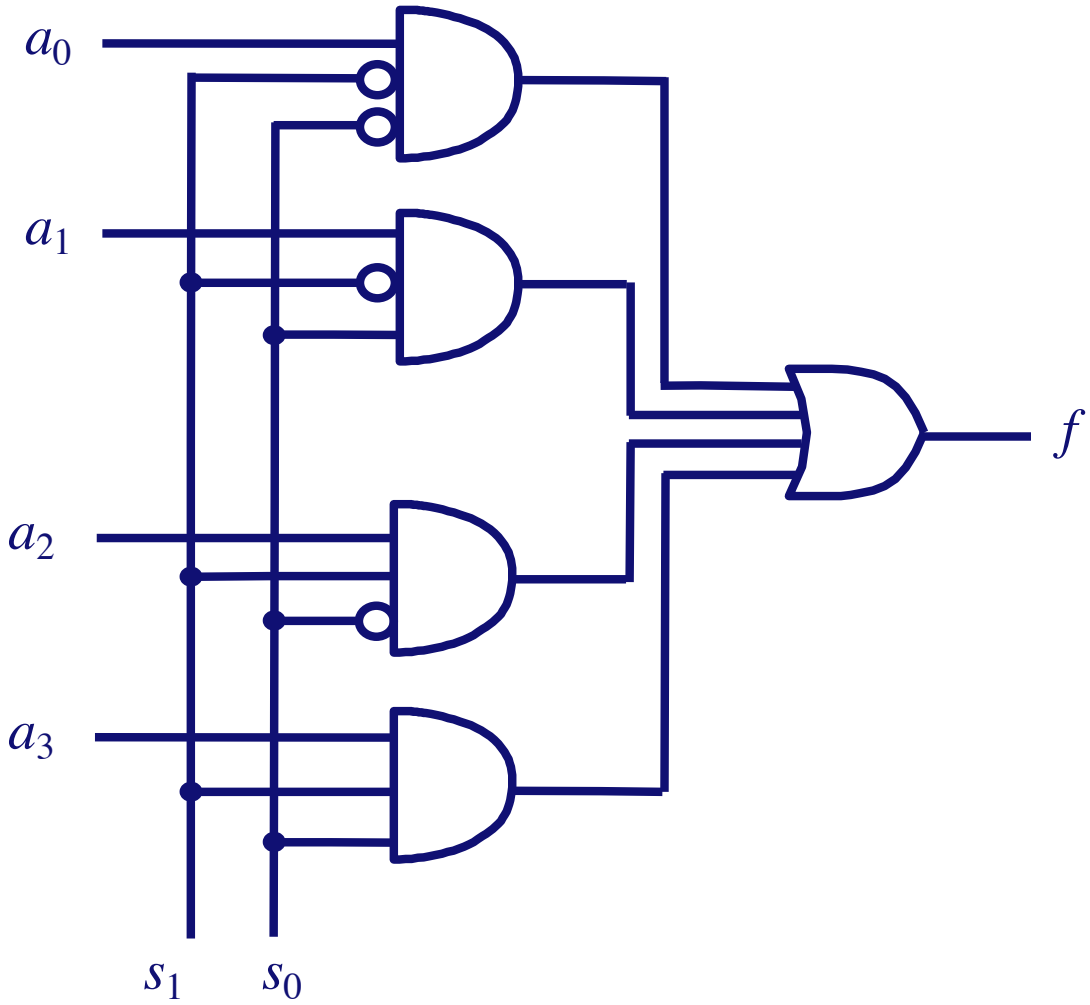
symbol



4-input multiplexer: built from 2-input MUX-es



4-input multiplexer: two-level circuit



Multiplexers as Universal Function Blocks

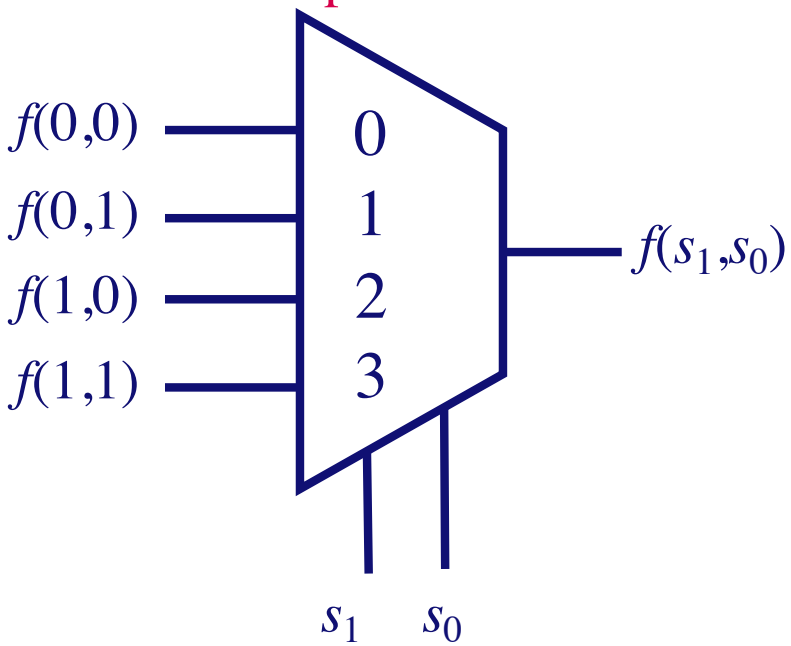
4-valued function (abstract)

s	$f(s)$
0	$f(0)$
1	$f(1)$
2	$f(2)$
3	$f(3)$

boolean function of 2 boolean variables

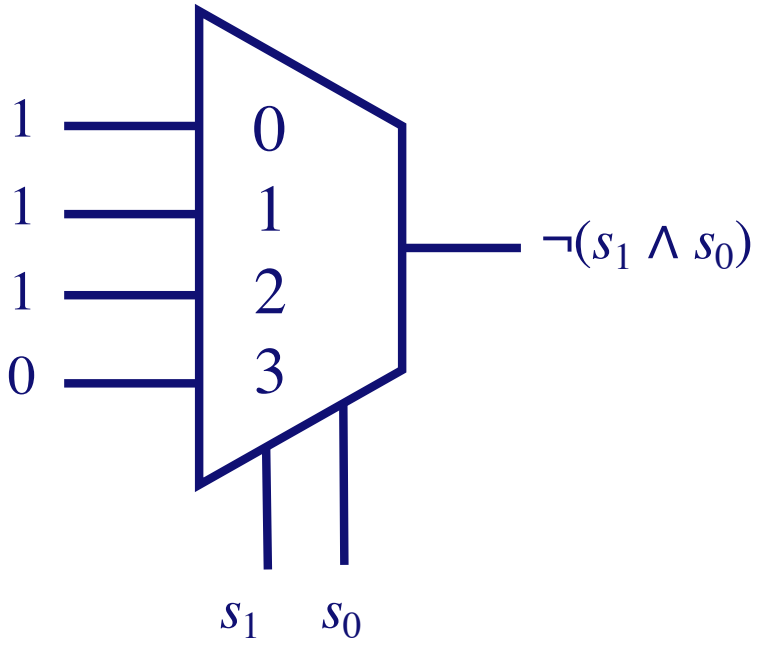
s_1	s_0	$f(s_1,s_0)$
0	0	$f(0,0)$
0	1	$f(0,1)$
1	0	$f(1,0)$
1	1	$f(1,1)$

MUX-implementation

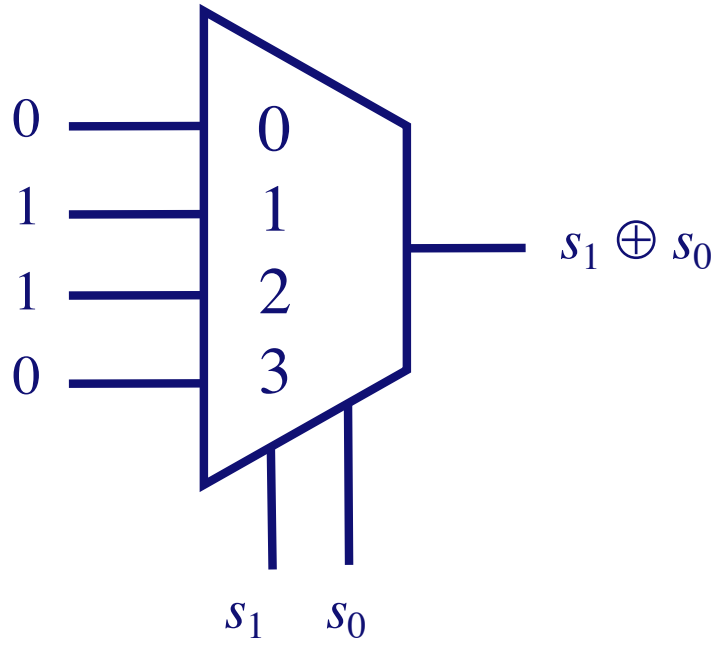


A multiplexer to implement standard gates.

NAND gate



Exclusive OR gate



Questions?

Summary

What did you learn:

- Carry look ahead adder, multiplication, alternative codes.
- S/R-flip-flops, D-latches and D-flip-flops.
- Glitches and meta-stability.
- How to construct larger multiplexers from simple multiplexers.
- How to use multiplexers to construct more combinatorial gates and circuits.