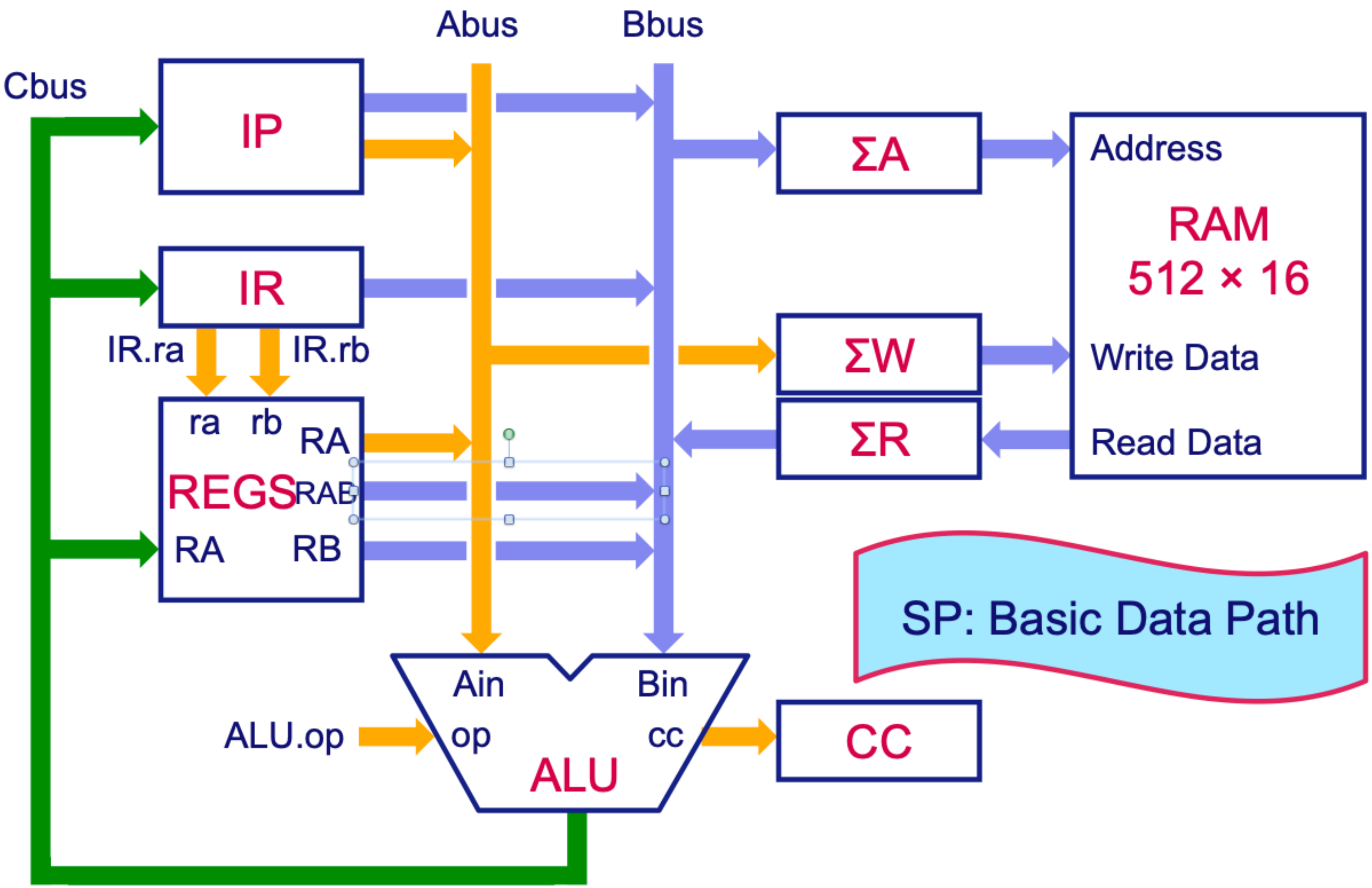
2IC30: Computer Systems Registers, Random-Access Memories and Finite State Automata.

Jan Friso Groote

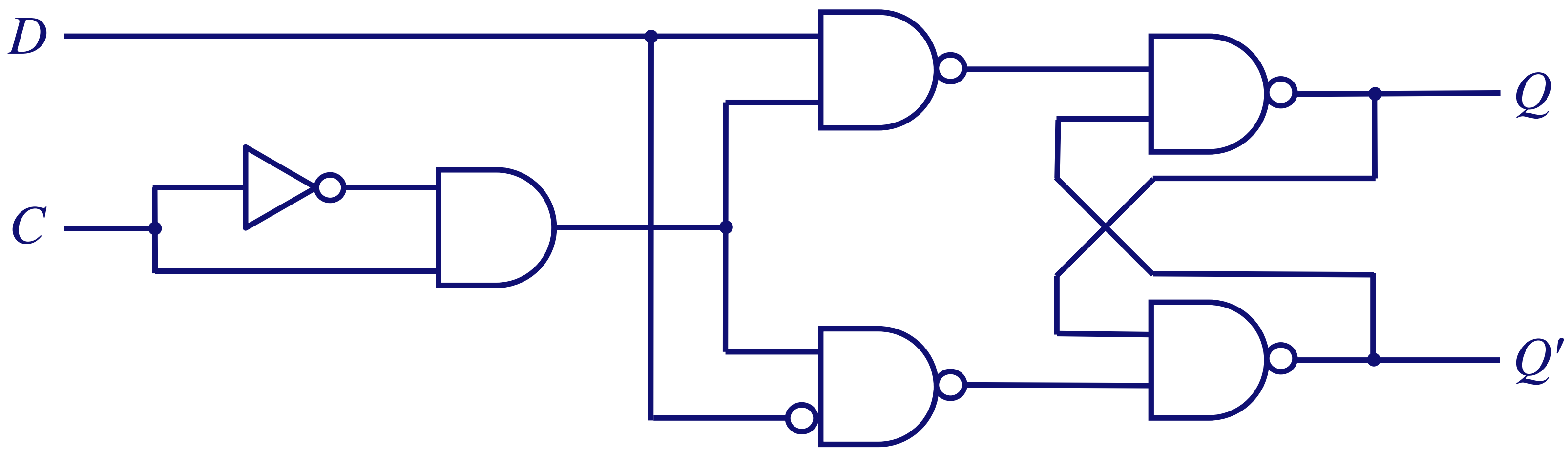
TU/e Technische Universiteit Eindhoven University of Technology

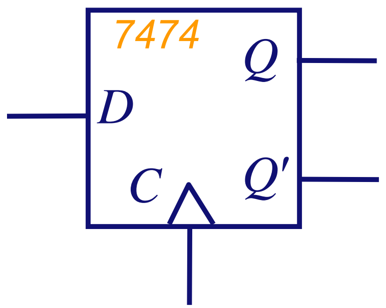
Where innovation starts

A microprocessor stores data in registers



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





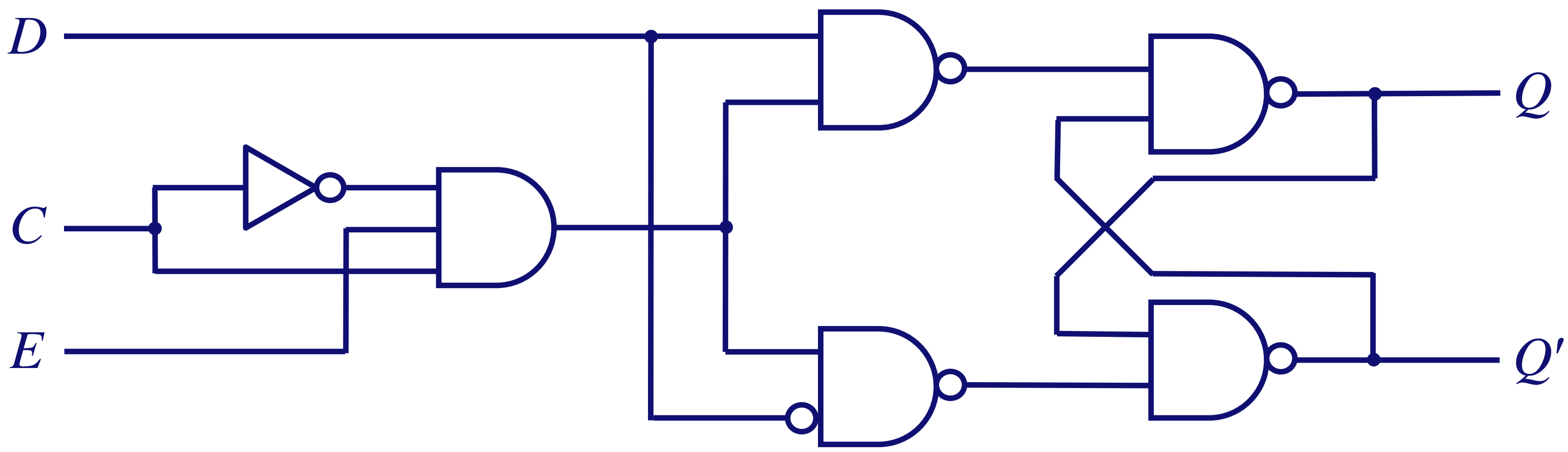
edge-triggered D-flip-flop

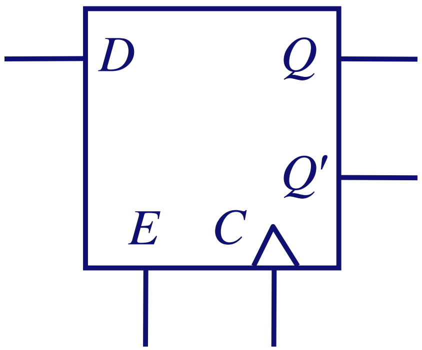
- **on** $C\uparrow$ **then** $Q := D$ and $Q' := \neg D$: the outputs copy D on every *upgoing transition* on C.

Requirement:

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

Clocked D-Flip-Flop with Enable





clocked D-flip-flop,
with enable

- **on** $C\uparrow$ **then** $Q := D$ and $Q' := \neg D$: the outputs copy D on every *upgoing transition* on C, but *only if* $E = 1$. When $E = 0$ transitions on C have no effect.
- Application: to build *registers* that must not assume new values on every clock transition.

Building a Register

Register

- Collection of Flip-Flops, with
- common clock and enable signals.

n-bits register:

- Register containing n Flip-Flops.

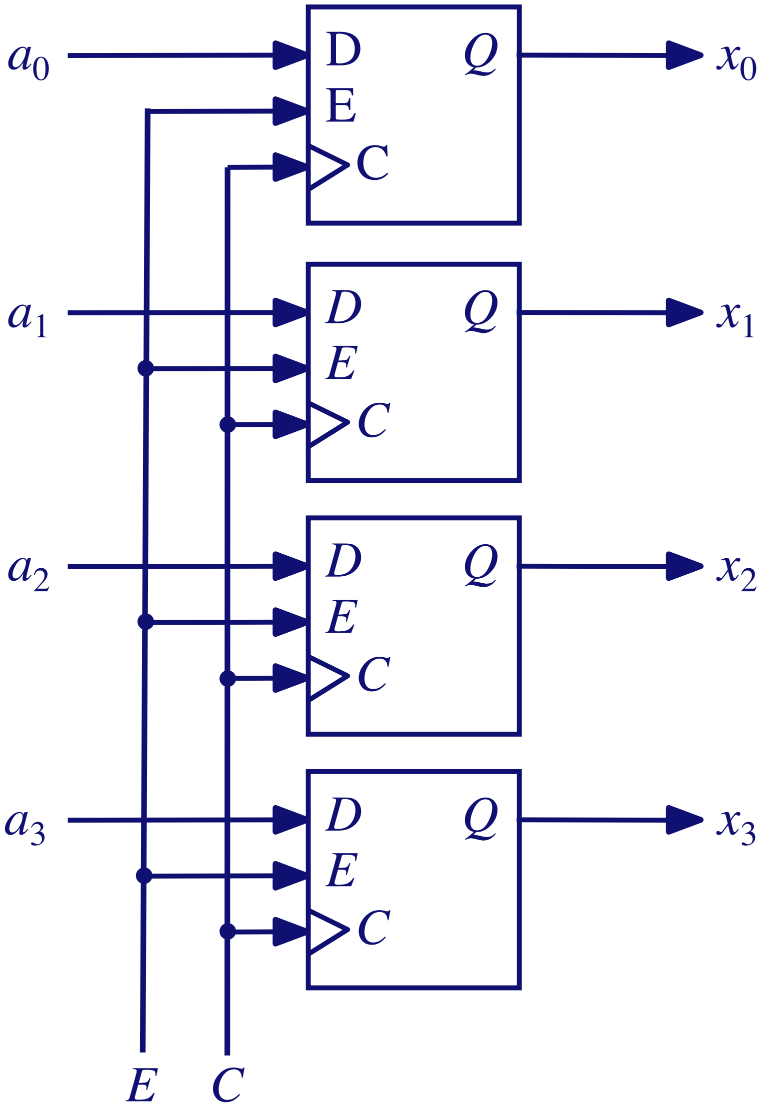
Application

- Number processing systems, like computers

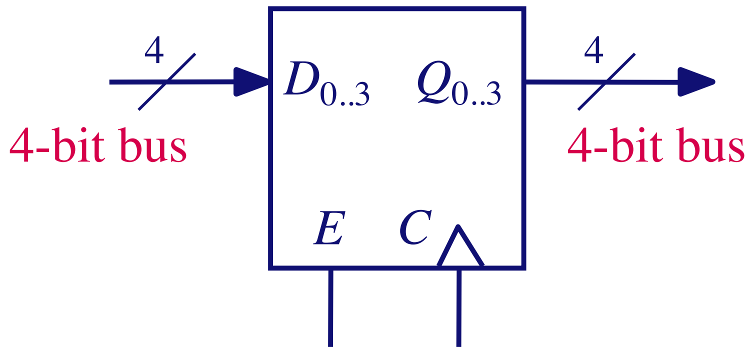
TU/e Technische Universiteit Eindhoven University of Technology

Example: a 4-bit Register

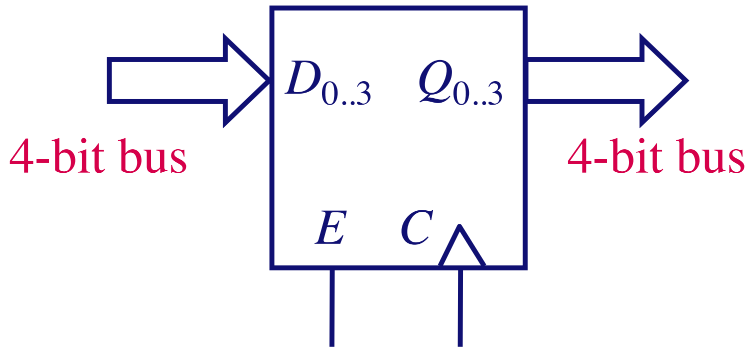
4-bit clocked register, with enable



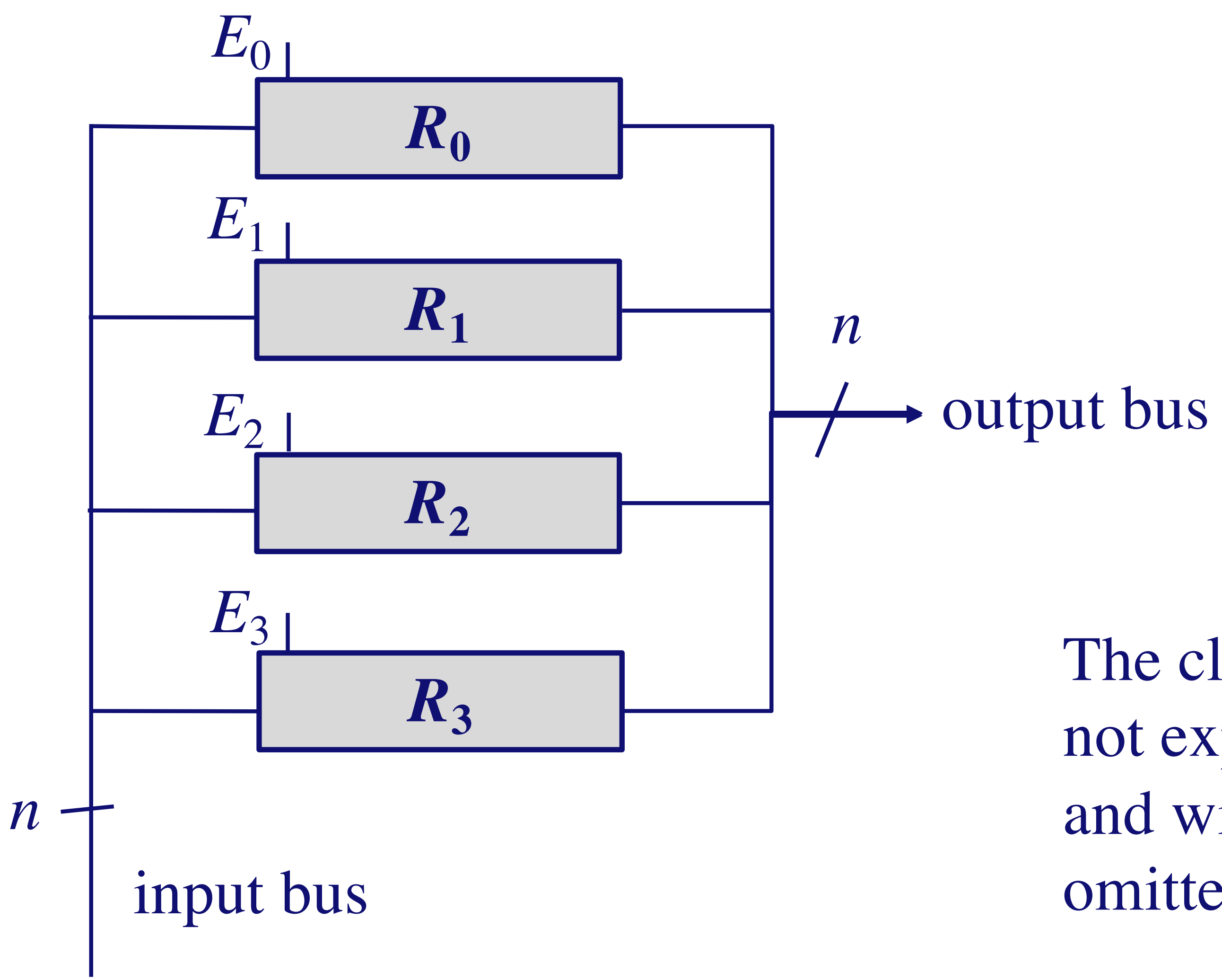
symbol



alternative symbol

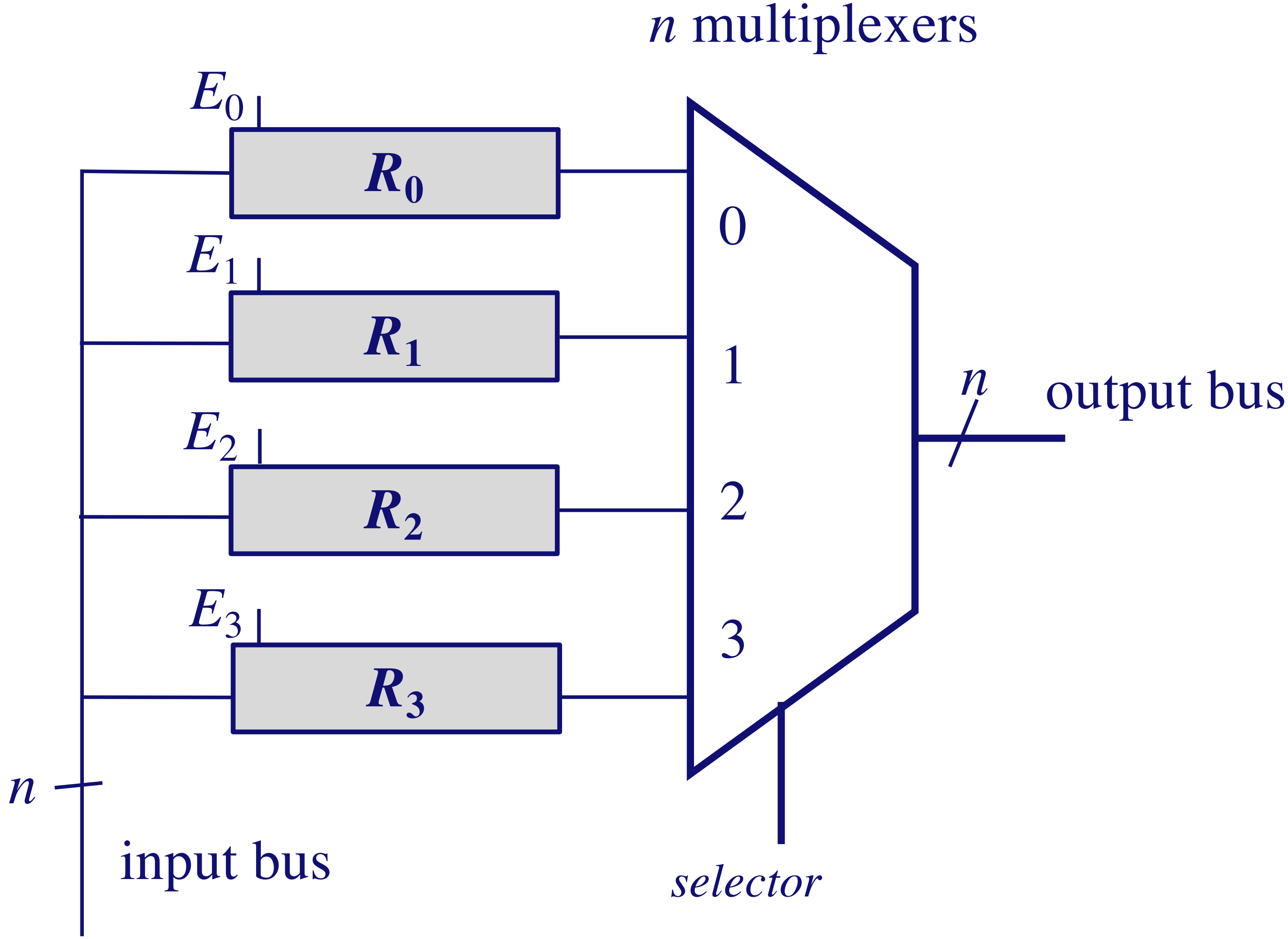


n-bit registers connected to the same input and output buses.

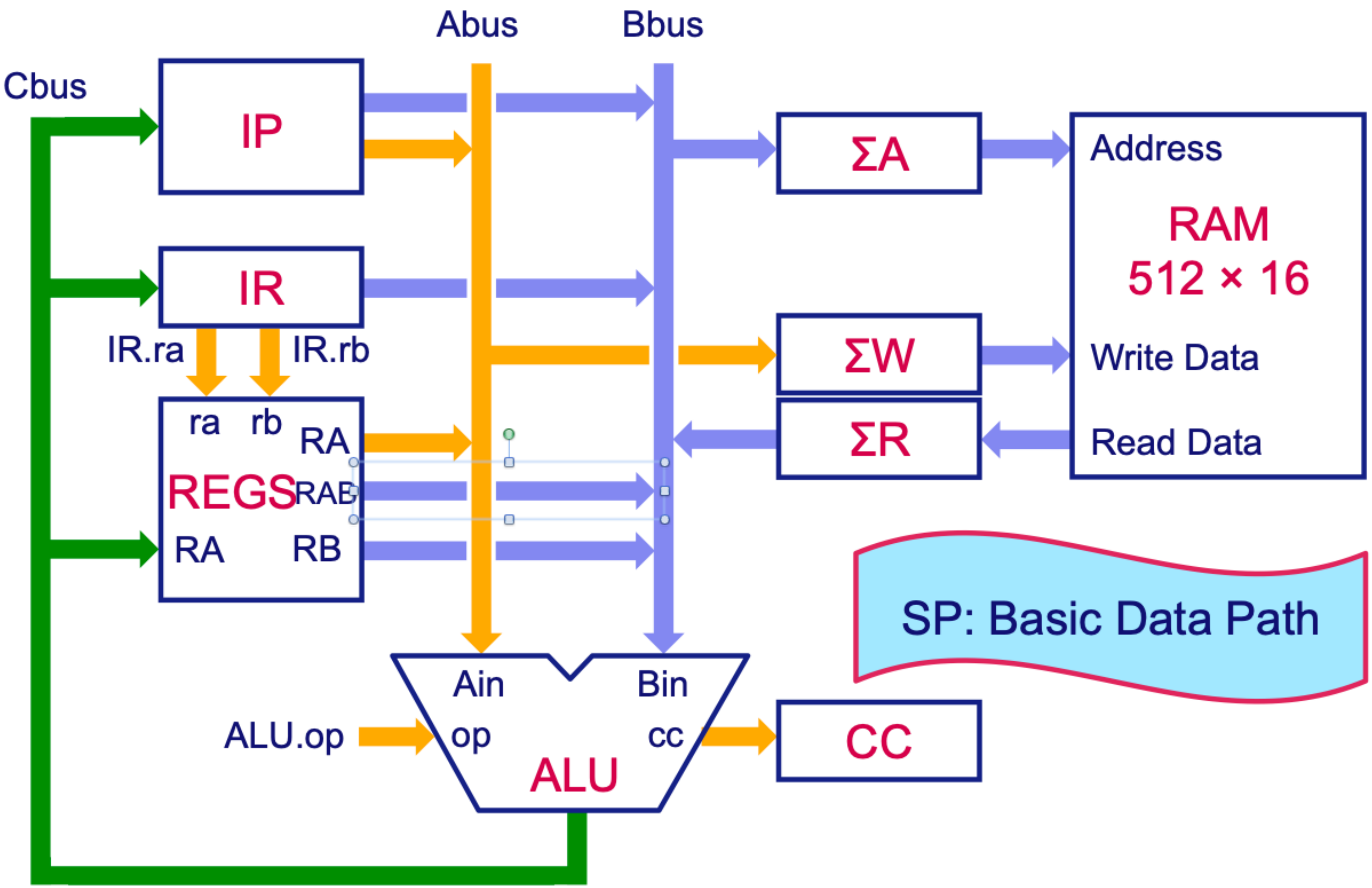


The clock signal is not explicitly drawn, and will in general be omitted.

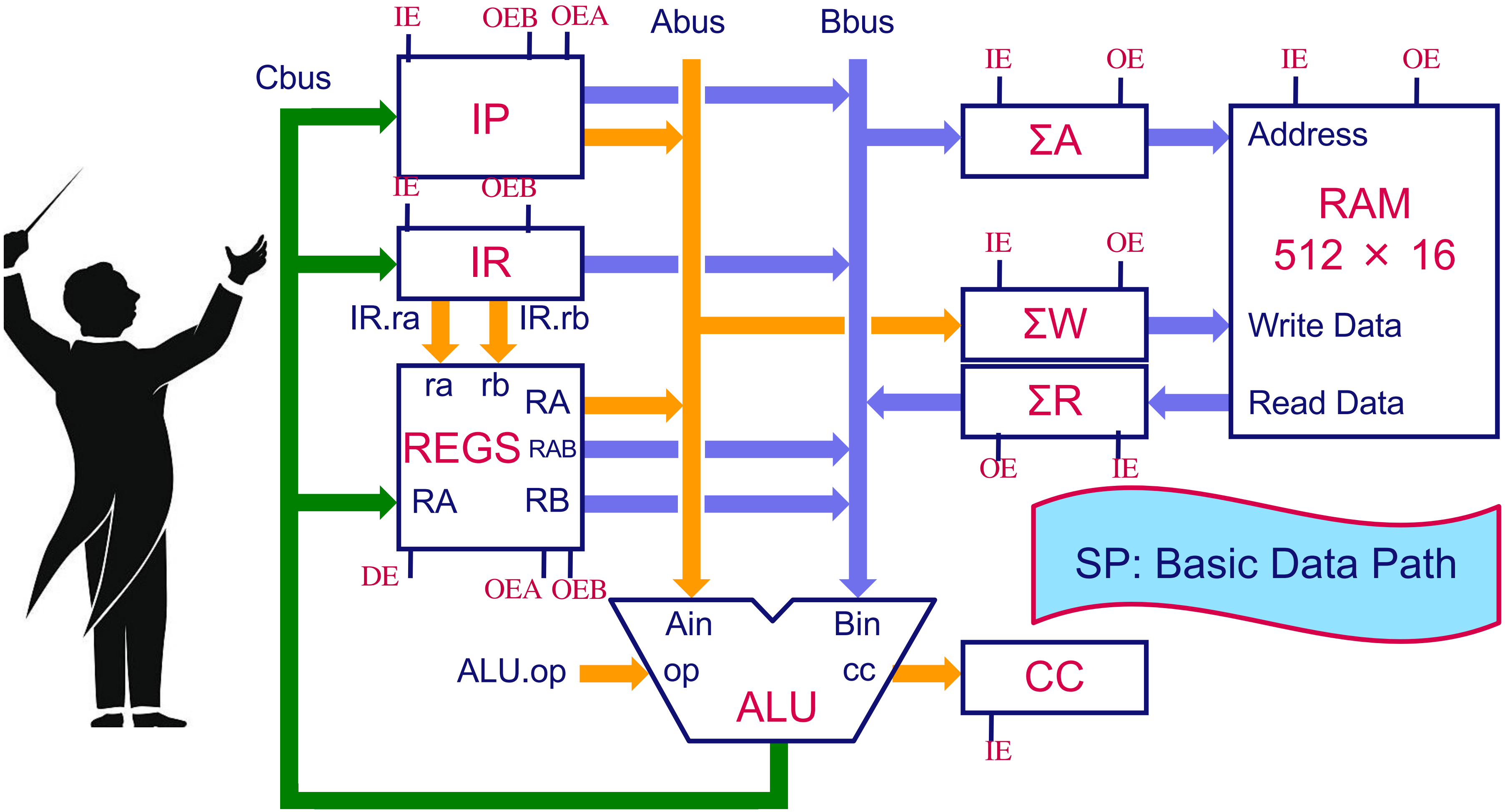
n-bit registers connected to the same input and output buses.



A microprocessor stores data in registers

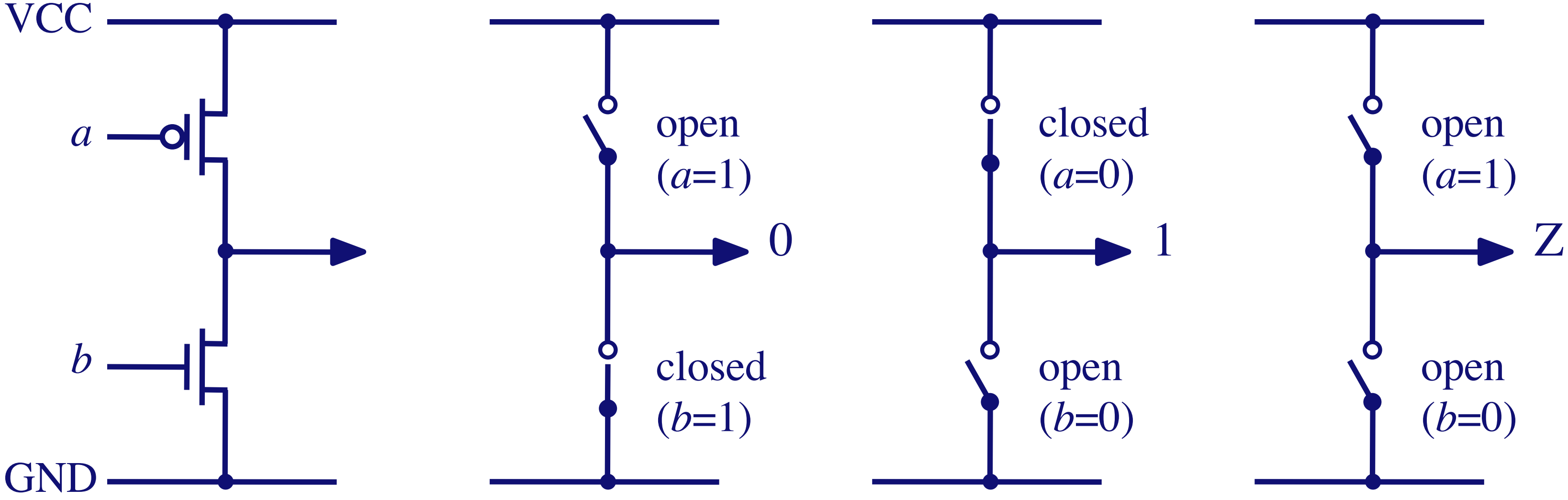


Simple Processor: Data Path



- Abus: IP or RA
- Bbus: IP, IR, ΣR, RB or RAB
- REGS: 2 Reads and 1 Write, simultaneously, ra[1..0] and rb[1..0] select registers

3-State Outputs



CMOS output stage... ... behaves like switches

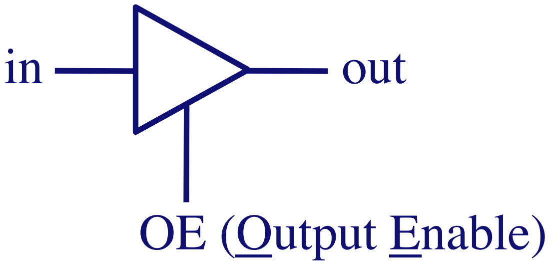
output value "Z" :
not connected, high impedance

a	b	output
0	0	1
1	0	Z
1	1	0
0	1	☠ (short circuit!)

3-State Outputs: Use

symbol

3-state buffer



truth table

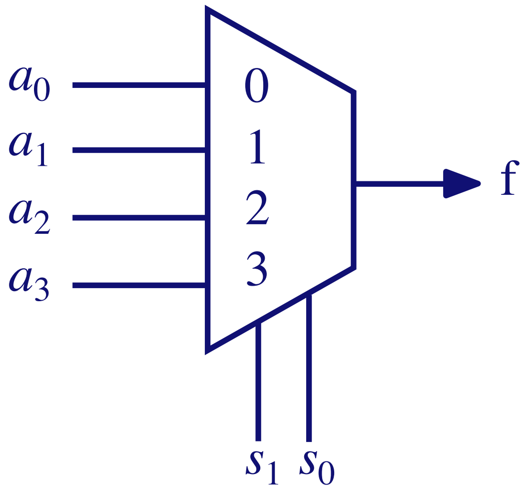
OE	in	out
0	0	Z
0	1	Z
1	0	0
1	1	1

- Application: 3-state outputs may be connected together, *provided* at any moment in time *at most one* of the OE-inputs is 1.
- If all OE-inputs are 0: the output equals *Z*.
- If one OE-input is 1: the output equals the "corresponding" input.
- Thus, cheap and extensible multiplexers can be built.

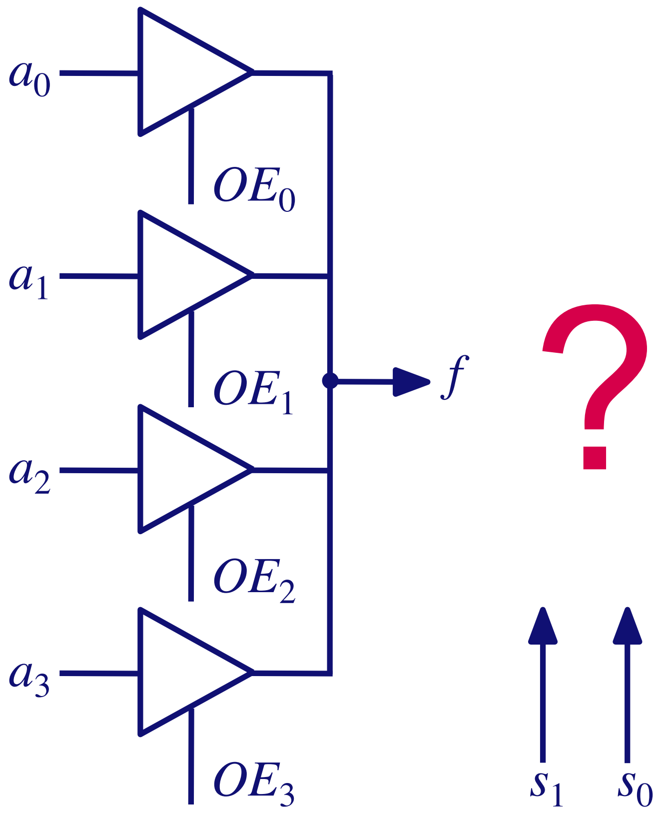
Technische Universiteit Eindhoven University of Technology

3-State Outputs: building a 4-input Multiplexer

4-input MUX (symbol)



4-input MUX from 3-state buffers

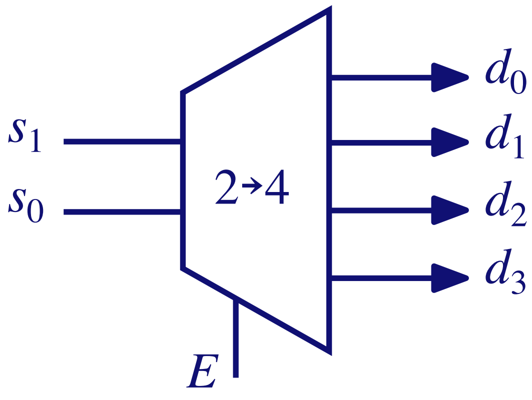


- problem: how to connect the OE-signals?
- solution: use a 2→4-*decoder* !

Technische Universiteit Eindhoven University of Technology

2→4 -decoder

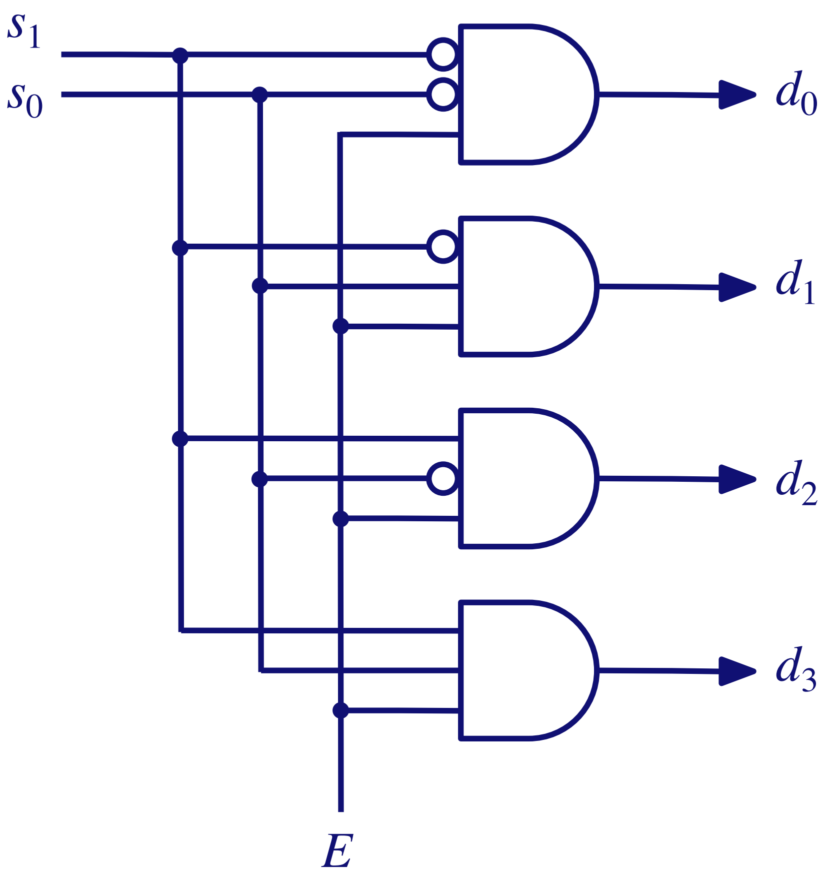
2→4 decoder (symbol)



truth table

E	s_1	s_0	d_3	d_2	d_1	d_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

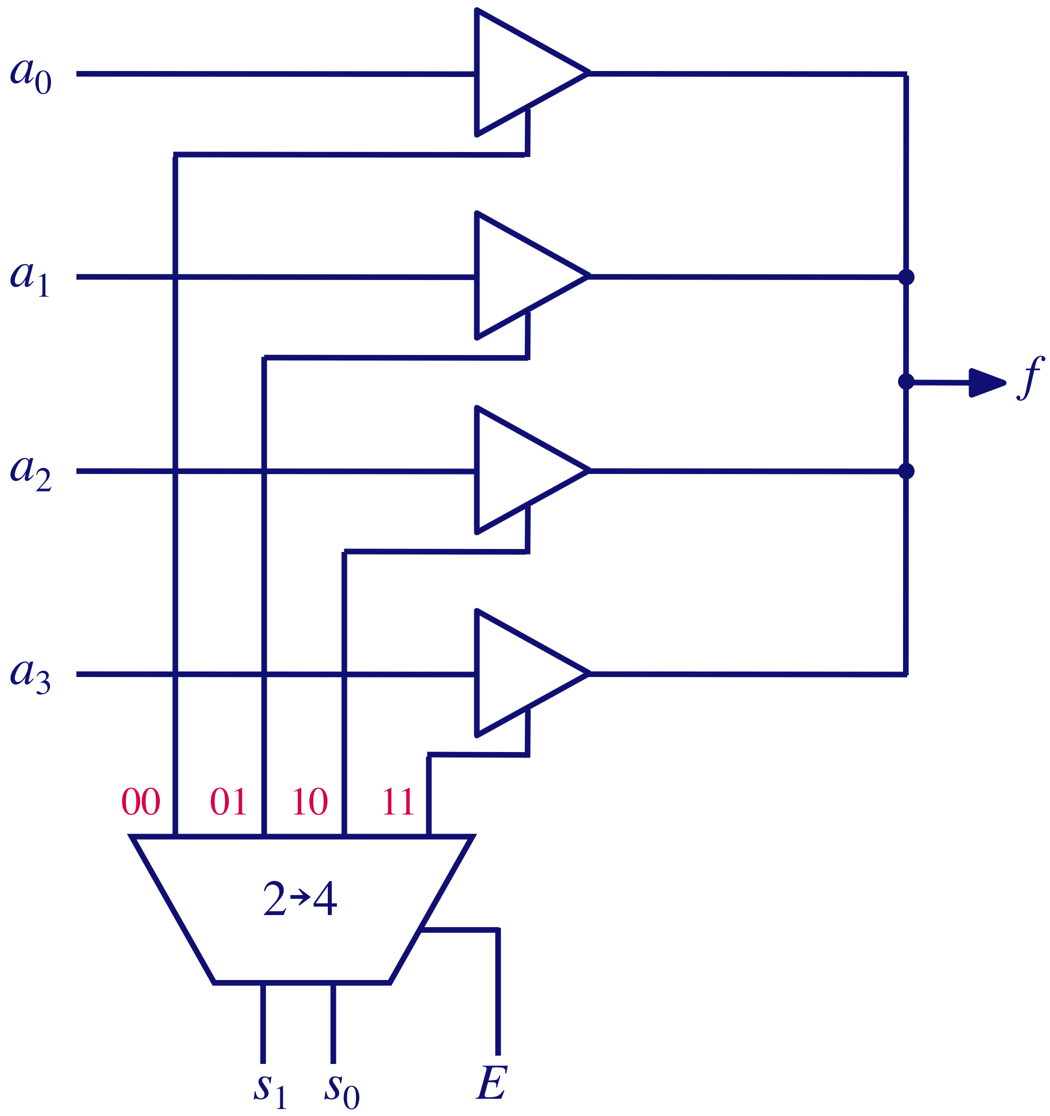
2→4 decoder: circuit



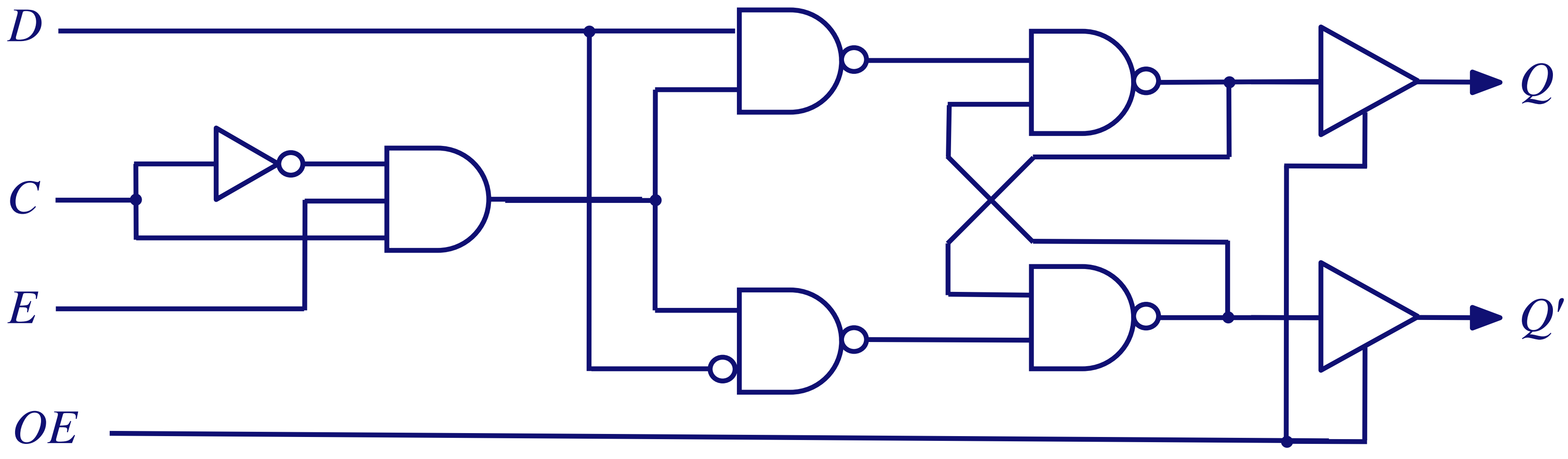
more general: n→2^n decoder

Application: 4-input Multiplexer, with enable

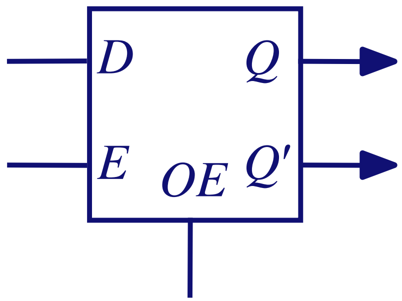
4-input MUX from 3-state buffers



D-Latch with 3-state outputs



(output enable)



D-flip flop with
3-state outputs

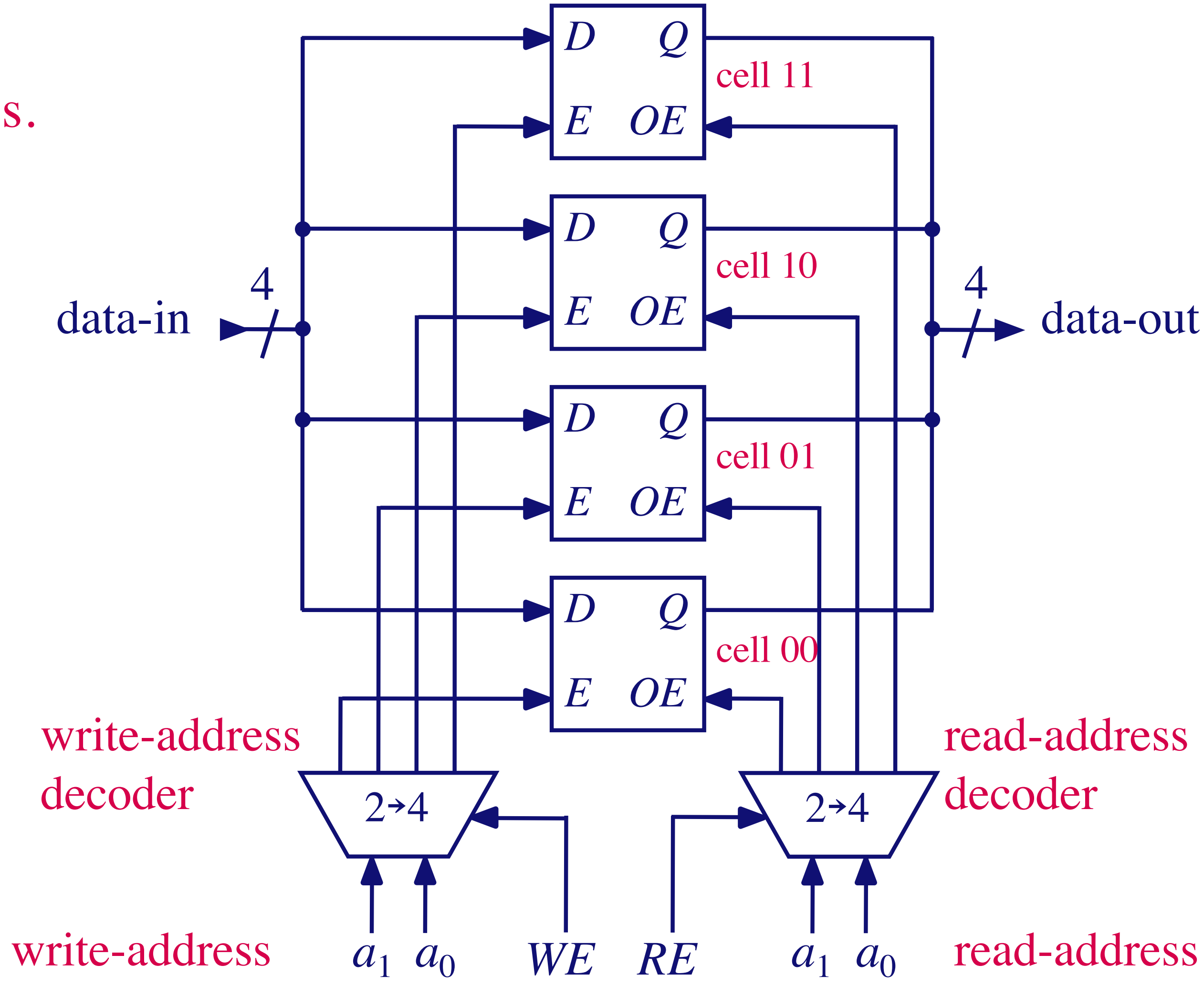
Property

- The *output buffers* are placed *behind* the latch and its internal feedback wiring: the latch can be changed state *even* if the outputs are *disabled*.

Technische Universiteit Eindhoven University of Technology

RAM: Random-Access Memory

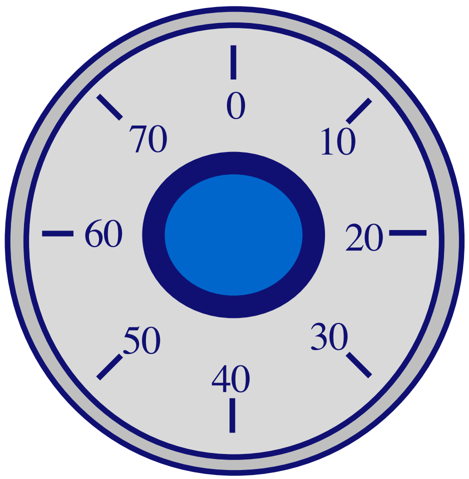
4-bit RAM
with four address.



Questions?

Systems with memory

- Combinatorial circuits have no memory: output values only depend on on current input values
- To allow behaviour to depend on (values from) the past, we need different circuits: *sequential circuits*
- A sequential lock only opens if a series of input values is presented in a very specific order



Behaviour of a system (circuit) is determined by the current input values and (input) values from the past

Finite Automata: the On-Off Switch

Specification

If the push button is pressed, the light goes “on”, and stays “on”; if the same button is pressed again the light goes “off”, and stays “off”, and so on indefinitely.

Variables

$\neg p$: the push button is “open”

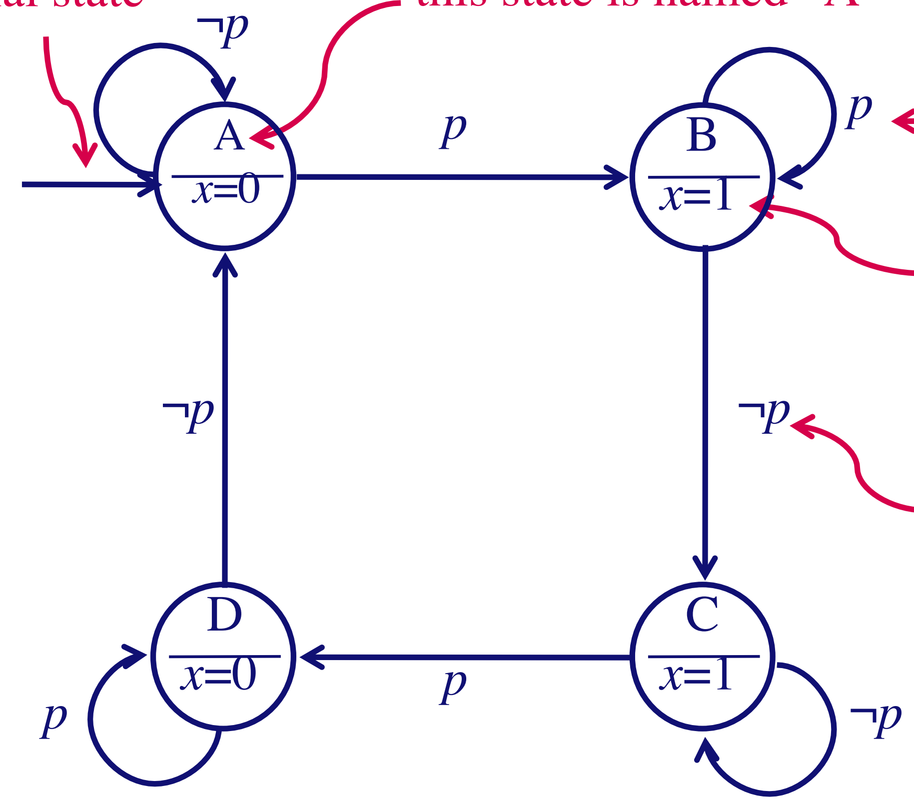
p : the push button is “closed”

$\neg x$: the light is “off”

x : the light is “on”

a *incoming arrow* identifies the “initial state”

this state is named “A”



if p the automaton remains in state B

in this state the output x equals 1

if $\neg p$ the automaton makes a *transition* from state B to state C

Technische Universiteit Eindhoven University of Technology

Finite Automata: Considerations

- *Different* states are needed whenever *different* behaviour is required for the *same* input values.
- States may be *named* arbitrarily: letters, numbers, … ; a binary representation is chosen later. The *starting state*, or: *initial state*, must be clearly indicated.
- *Transitions* between states are labelled with Boolean expressions, called *guards*: a transition from one state to a next happens only if the guard of that transition is **true**.
- *Moore Machine*: the outputs depend on the state only, not (directly) on the inputs (we mainly use Moore Machines).
- *Mealy Machine*: the outputs may depend on the state and on the inputs.

Technische Universiteit Eindhoven University of Technology

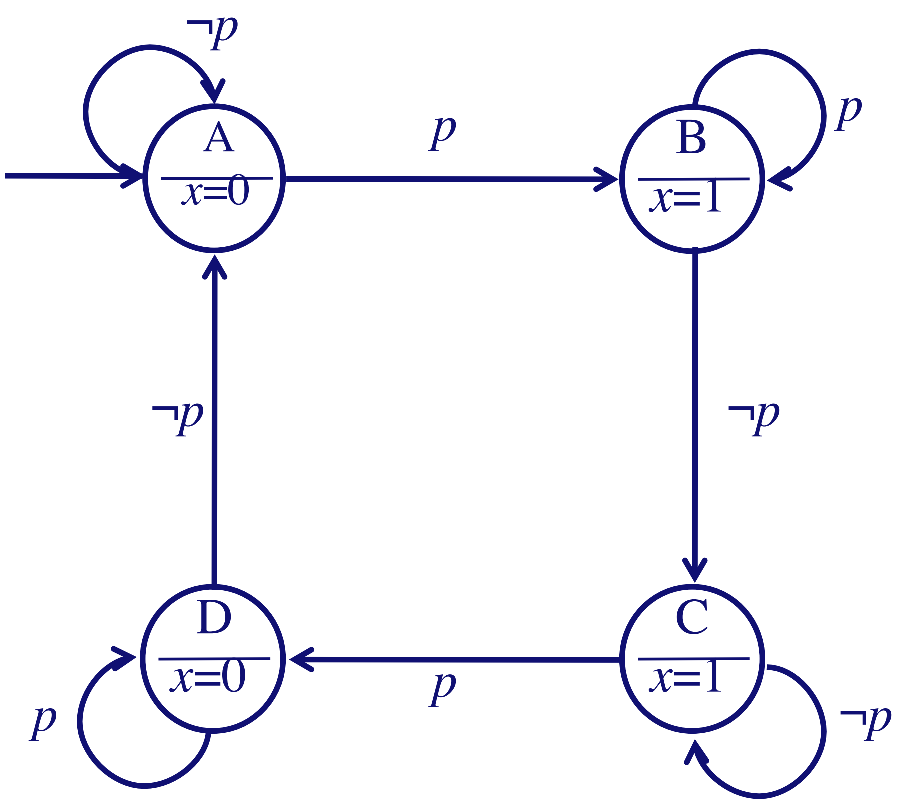
Example: the On-Off Switch (1)

Specification

If the push button is pressed, the light goes "on", and stays "on"; if the same button is pressed again the light goes "off", and stays "off", and so on indefinitely.

Variables

$\neg p$: the push button is "open"
p : the push button is "closed"
$\neg x$: the light is "off"
x : the light is "on"



Convention

- If, in any state, no outgoing transition is *enabled* –has a true guard–, then the automaton remains in its current state;
- Transitions from a state to itself often are not drawn: yields a cleaner drawing.

Example: the On-Off Switch (2)

Output table and State-transition table

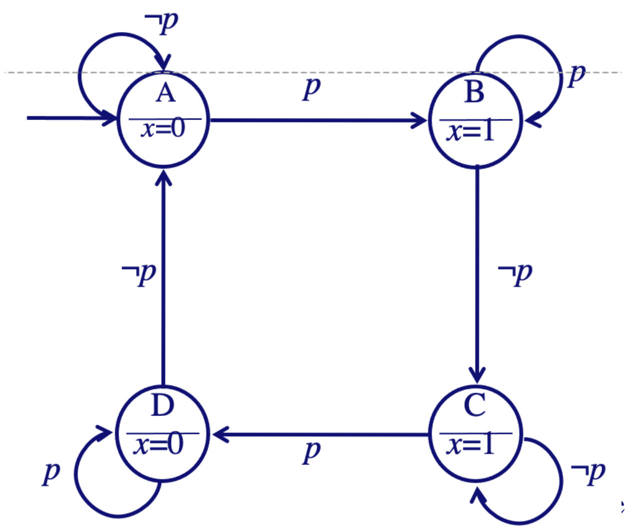
From the state-transition diagram we derive an *output table* and an (abstract) *state-transition table*, in terms of the (abstract) names assigned to the states.

output table

state	x
A	0
B	1
C	1
D	0

state-transition table

state	p	$state_{new}$
A	0	A
A	1	B
B	1	B
B	0	C
C	0	C
C	1	D
D	1	D
D	0	A



Example: the On-Off Switch (3)

State assignment

Several different state assignments are possible; binary representation of 4 states requires *at least* 2 bits, for example:

just counting

state	s	t
A	0	0
B	0	1
C	1	0
D	1	1

Gray code

state	s	t
A	0	0
B	1	0
C	1	1
D	0	1

one-hot encoding

state	s_A	s_B	s_C	s_D
A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1

Gray code: because the states A, B, C, D are passed *cyclically*, in a Gray encoding only a *single state variable* changes in each transition.

On-Off Switch example: the Gray encoding is very attractive because now:

$s=1 \Leftrightarrow x=1$, so the output x just equals state variable s.

One-hot encoding: In every state, *exactly one* state variable has value 1; although this requires more variables, hence: more D-flip-flops, this may yield simpler combinatorial circuits.

TU/e Technische Universiteit Eindhoven University of Technology

Example: the On-Off Switch (4)

Boolean Functions

We choose the Gray encoding for the states, because this makes the function for the output simple. From the (abstract) state-transition table we derive a (concrete) state-transition table:

output

s	t	x
0	0	0
1	0	1
1	1	1
0	1	0

output function: $x = s$

new states

state	p	state$_{\text{new}}$
A	0	A
A	1	B
B	1	B
B	0	C
C	0	C
C	1	D
D	1	D
D	0	A

new states

s	t	p	s_{new}	t_{new}
0	0	0	0	0
0	0	1	1	0
1	0	1	1	0
1	0	0	1	1
1	1	0	1	1
1	1	1	0	1
0	1	1	0	1
0	1	0	0	0

Example: the On-Off Switch (5)

Karnaugh-map for s_{new}

p \ s, t	$s=0$, $t=0$	$s=1$, $t=0$	$s=1$, $t=1$	$s=0$, $t=1$
0	0	1	1	0
1	1	1	0	0

Karnaugh-map for t_{new}

p \ s, t	$s=0$, $t=0$	$s=1$, $t=0$	$s=1$, $t=1$	$s=0$, $t=1$
0	0	1	1	0
1	0	0	1	1

just a 2-input MUX:

$$s_{\text{new}} = p \wedge \neg t \vee \neg p \wedge s$$

another 2-input MUX:

$$t_{\text{new}} = \neg p \wedge s \vee p \wedge t$$

Observe risk for glitches!

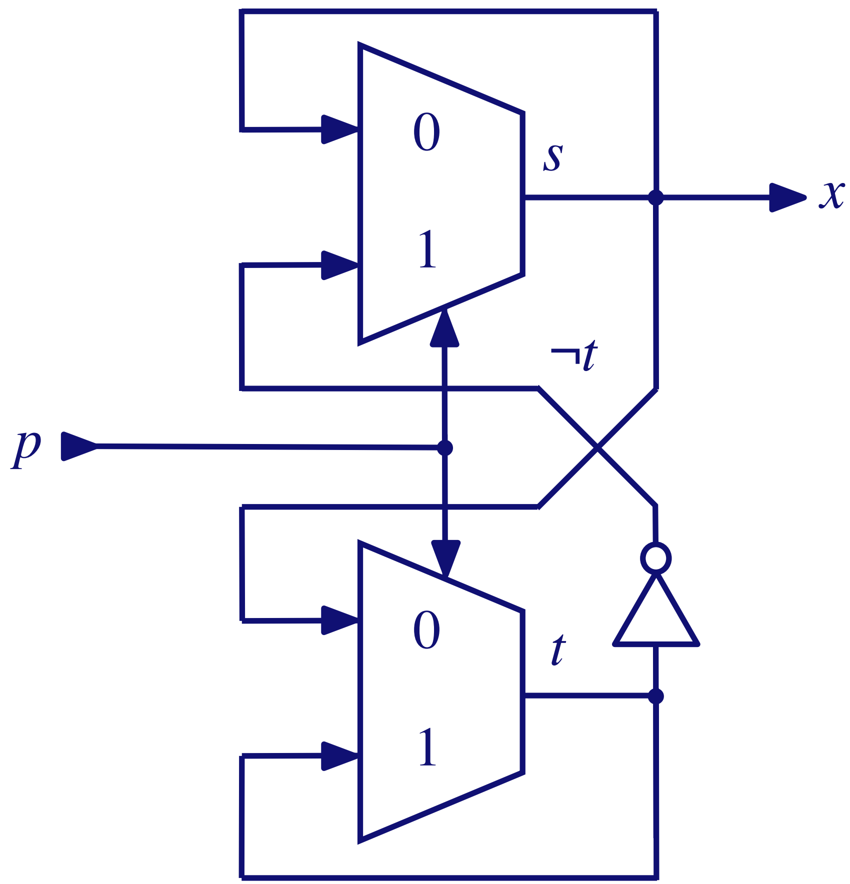
Example: the On-Off Switch (6)

Asynchronous sequential circuit

Just two 2-input multiplexers with feedback:

$$s_{\text{new}} = p \wedge \neg t \vee \neg p \wedge s$$

$$t_{\text{new}} = \neg p \wedge s \vee p \wedge t$$



Advantages

- Very fast responses to input changes
- Very compact circuits
- No clock needed
- Low power consumption

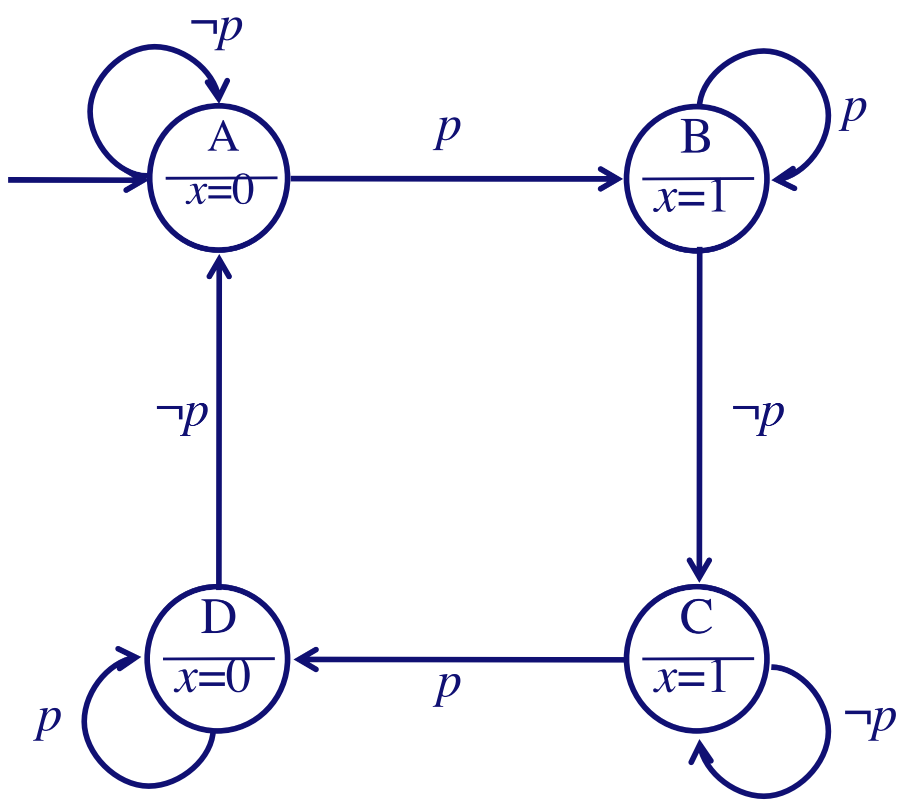
Disadvantages

- Requires *extremely careful* design:
- Very sensitive to glitches.

- Particularly glitch sensitive if several state variables change simultaneously. (Hence the Gray code!)

Example: the On-Off Switch

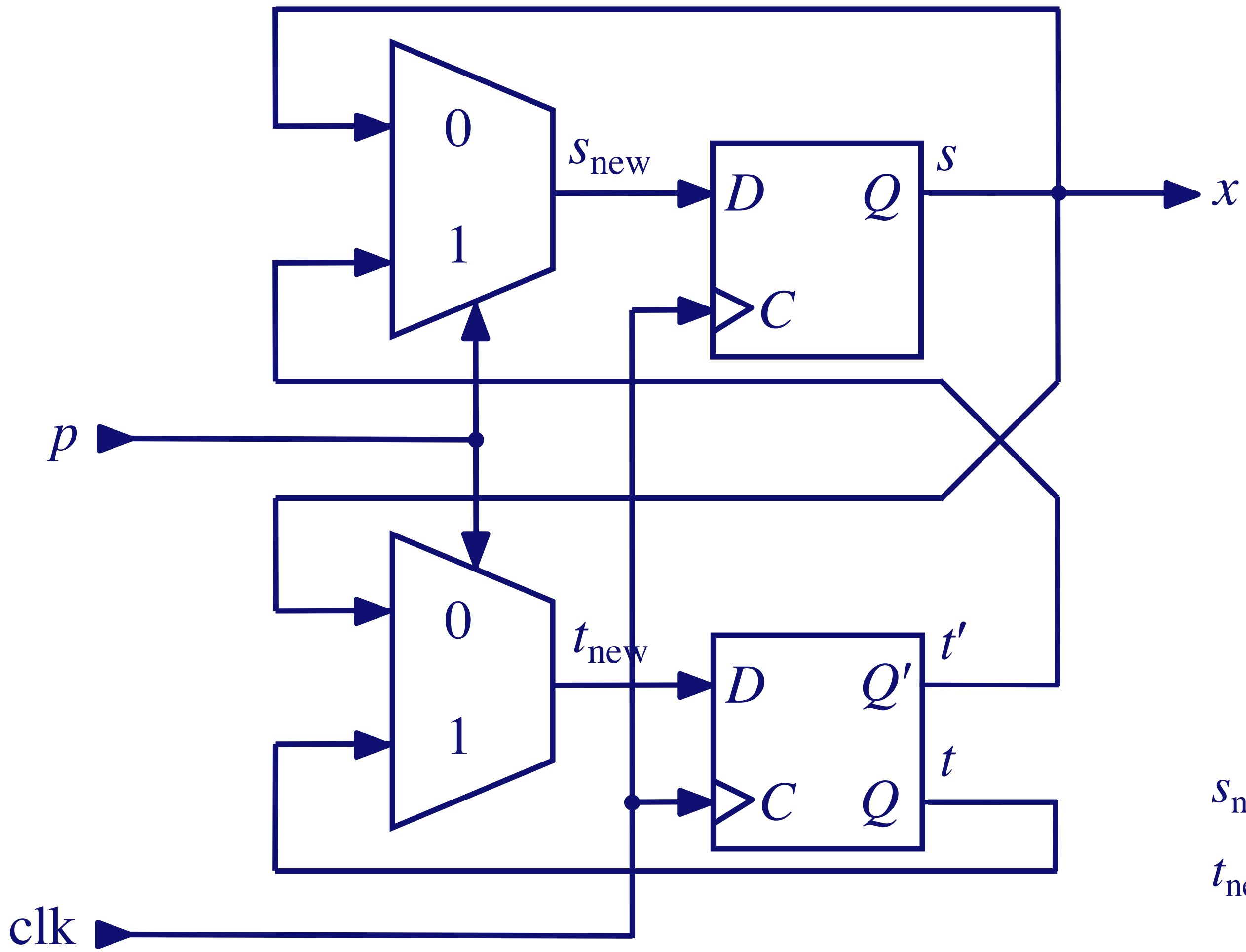
State-transition diagram



Example: the On-Off Switch (7)

Synchronous sequential circuit: driven by a periodic "clock"

The same circuit with feedback, but via D-Flip-Flops:



$$s_{new} = p \wedge \neg t \vee \neg p \wedge s$$

$$t_{new} = \neg p \wedge s \vee p \wedge t$$

TU/e Technische Universiteit Eindhoven University of Technology

Example: the On-Off Switch (8)

Synchronous sequential circuit:

The same circuit with feedback, but via D-Flip-Flops.

Properties:

- response speed to input changes increases with the clock frequency: the faster the better.
- but: the maximal clock frequency f_{max} is limited by: $f_{\mathrm{max}} \leq 1/\mathrm{T}$, where

(*)..... $t_{\mathrm{pdc}} + t_{\mathrm{pdf}} + t_{\mathrm{su}} = \mathrm{T}$; here:

t_{pdc} = "propagation delay of the combinatorial circuit",

t_{pdf} = "propagation delay of the D-Flip-Flop",

t_{su} = "setup time of the D-Flip-Flop".

- if requirement (*) is met: glitches in the combinatorial circuit are harmless: they will have died out before the next clock transition.

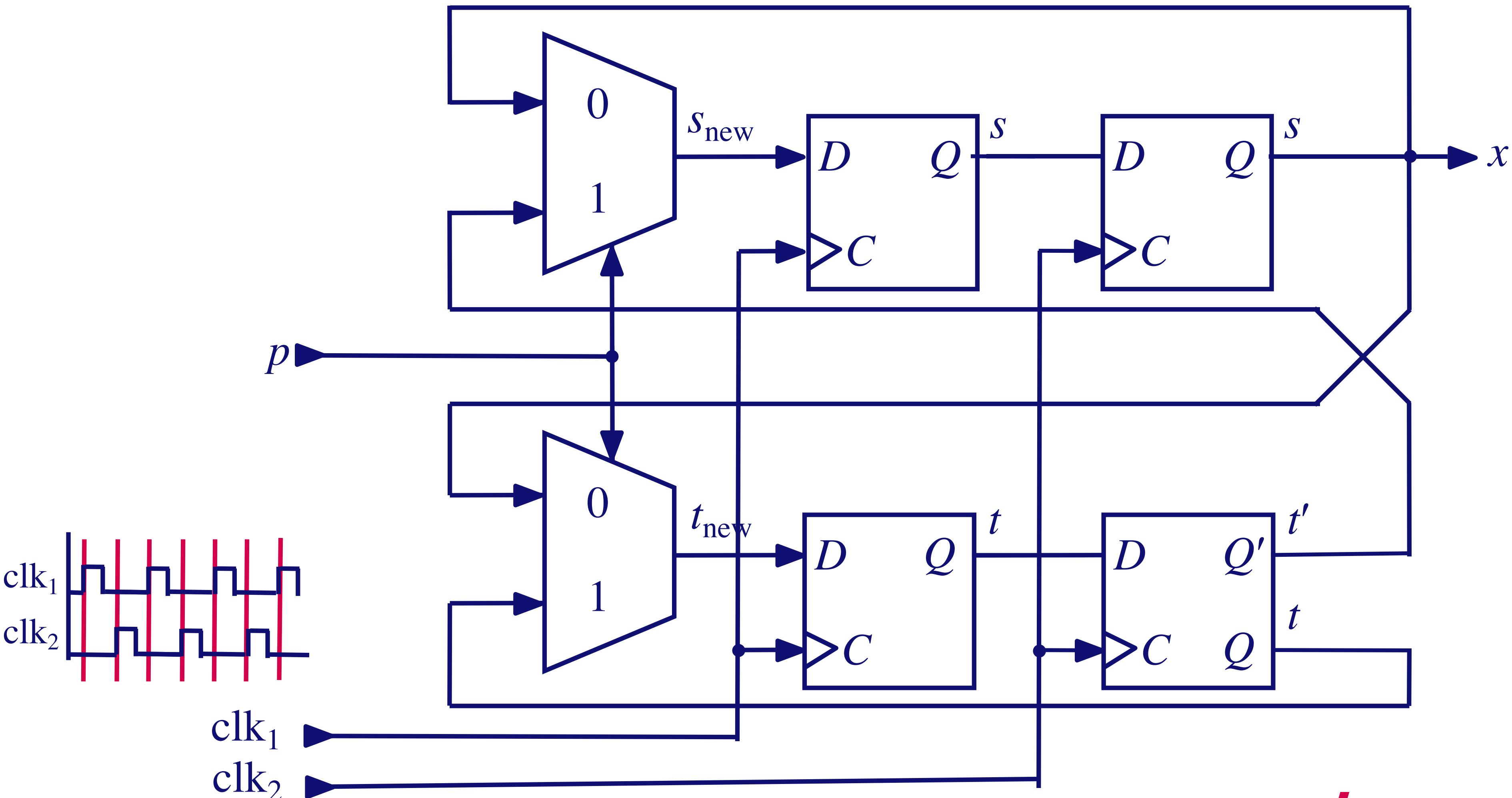
Example: the On-Off Switch with a double clock

Synchronous sequential circuit: driven by a double "clock"

The same circuit with feedback, but via four D-Flip-Flops:

$$s_{\text{new}} = p \wedge \neg t \vee \neg p \wedge s$$

$$t_{\text{new}} = \neg p \wedge s \vee p \wedge t$$



Summary

What did you learn:

- Registers are made from multiple flip-flops.
- Using an input enable signal registers can be instructed to read.
- Using three state outputs and an output enable results from multiple register can be selected.
- Specify machines with memory as state machines.
- Implement state machines using flip-flops and combinatorial circuit.
- Mealy machine transfer their input directly to the output. This is faster than Moore machines, and generally leads to a smaller number of states.