Computer Mathematics

Week 10
Sequential logic circuits

KUAS 京都先端科学大学
KYOTO UNIVERSITY of ADVANCED SCIENCE
Department of Mechanical and Electrical System Engineering

last week

combinational digital circuits

signals and busses

logic gates

- and, or, not
- nand, nor, xor

gate-level logical operations

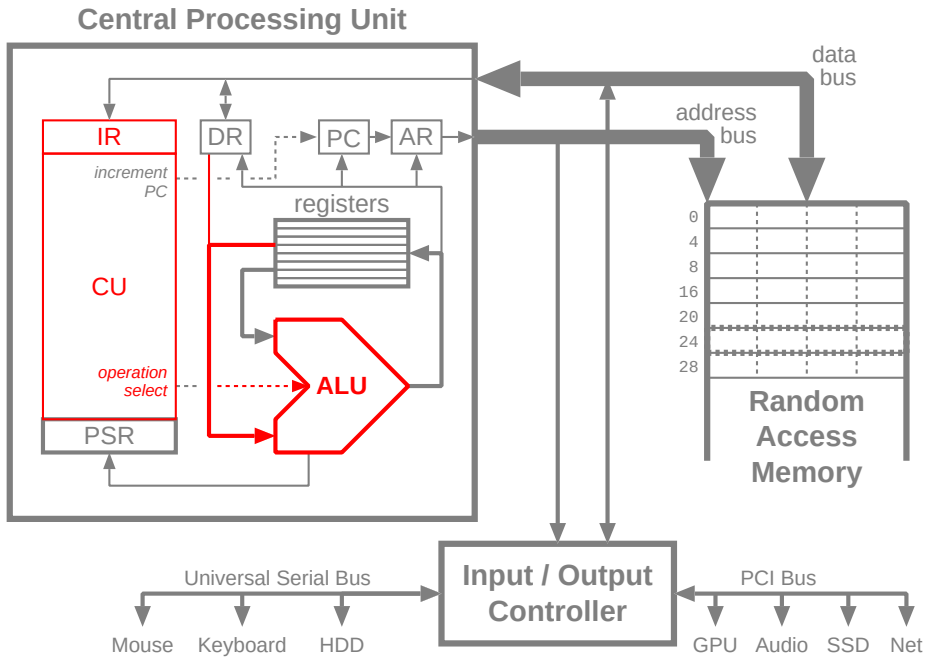
- bitwise: and, or, not

gate-level arithmetic operations

- addition, subtraction (unsigned, 2's complement)

1-of-N selection

- multiplexers



this week

sequential digital circuits

stateful logic

- level-triggered devices
- latches

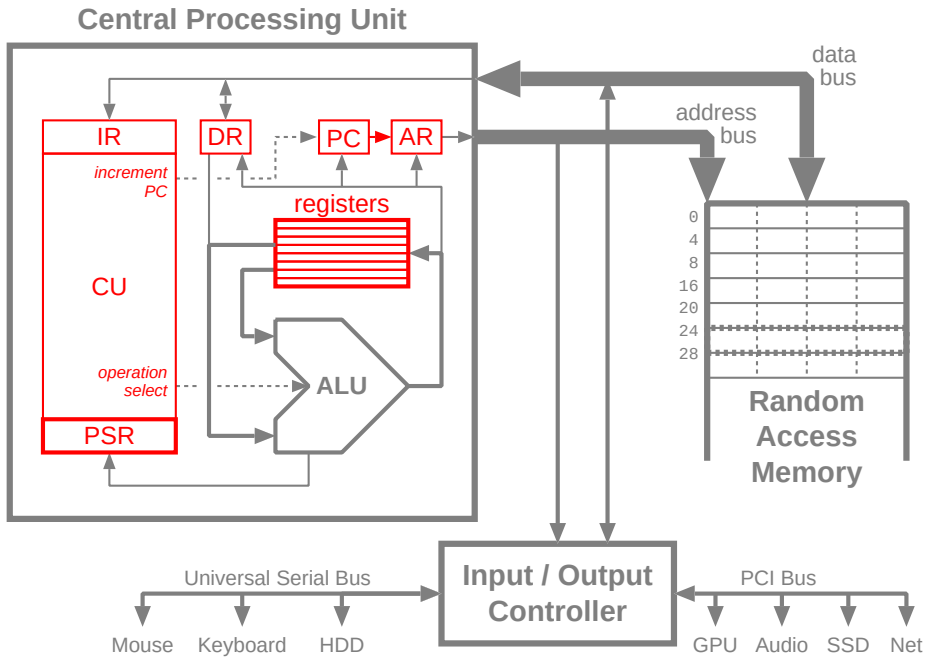
clocks

- edge-triggered devices

synchronous logic

- flip-flops
- registers and memory

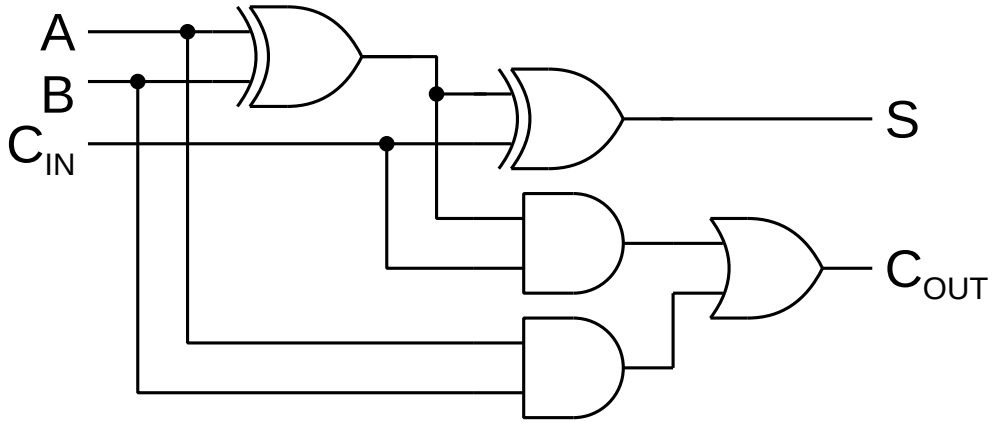
CPU operation according to the clock cycle



combinational vs. sequential logic

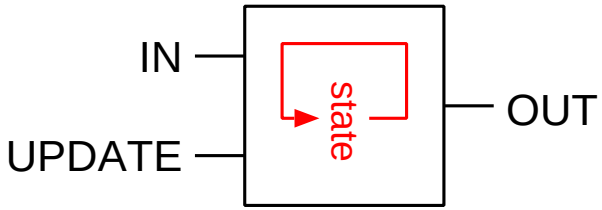
combinational logic

- outputs depend on current inputs
- no memory
- *stateless*



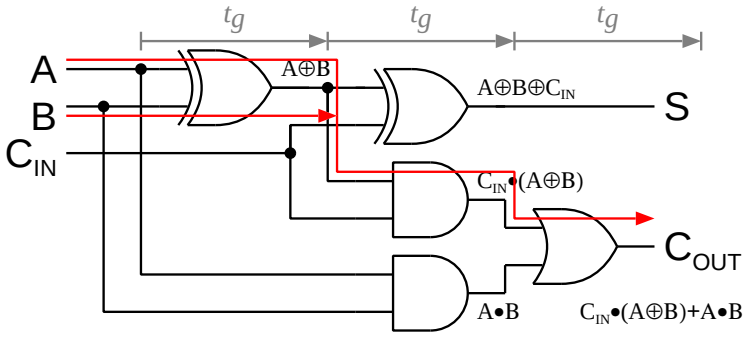
sequential logic

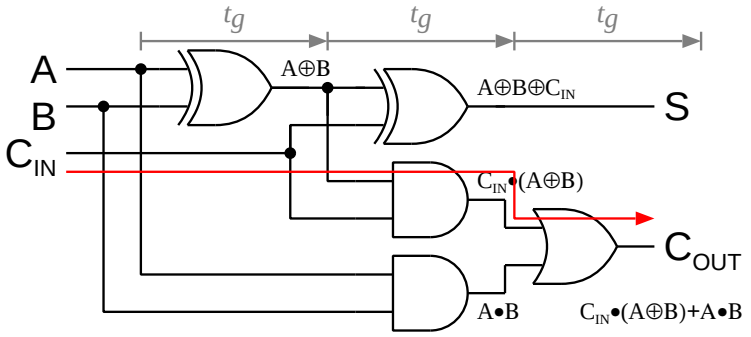
- outputs depend on current *and past* inputs
- memory (history) of previous inputs
- *stateful*



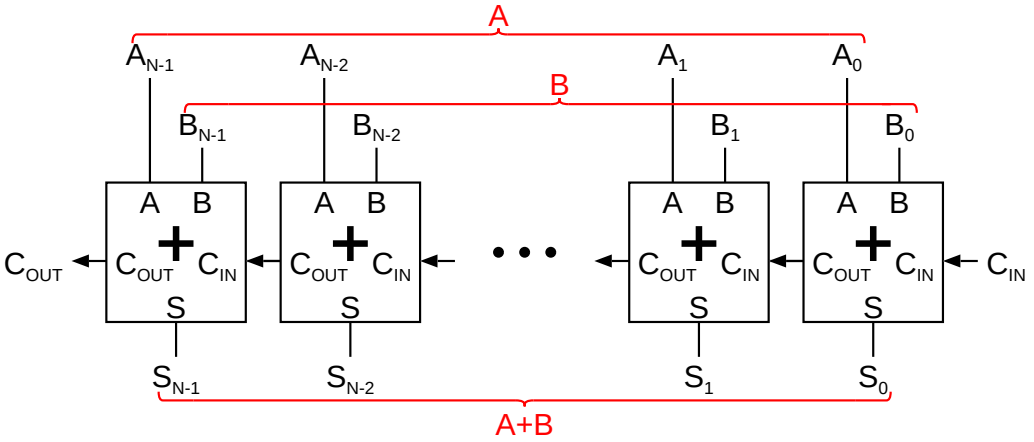
gate delay — adders

full adder: 3 gates to sum and carry out, 2 gates from carry in to carry out





daisy-chained...

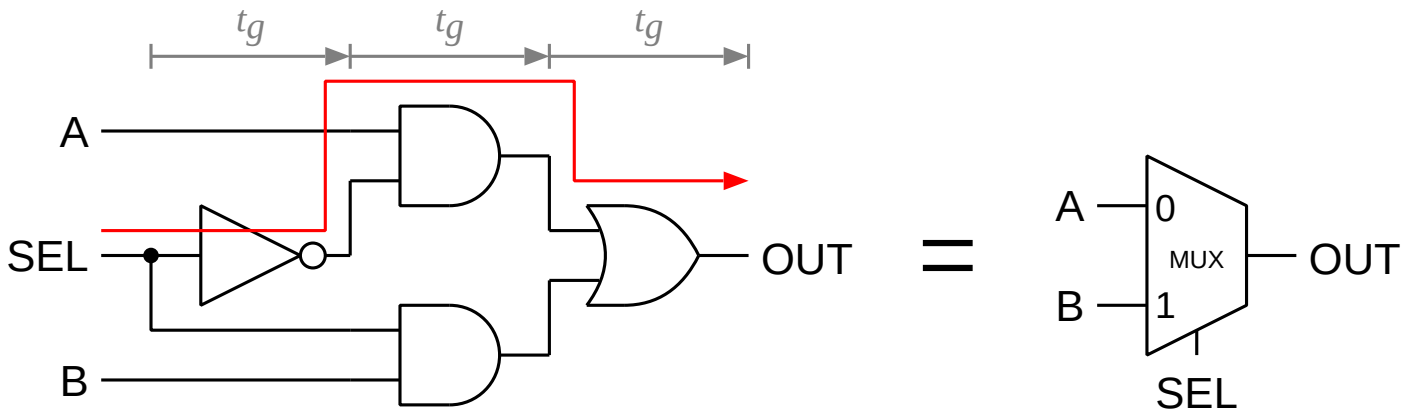


N is limited by *gate delay* $= t_g$

- from A, B, C_{IN} changing, C_{OUT} is available $3 \times t_g$ later
- from C_{IN} changing, C_{OUT} is available $2 \times t_g$ later
- final C_{OUT} is available $(1 + 2N) \times t_g$ after inputs change

gate delay — multiplexer

2-to-1 multiplexer



gate delays from inputs to output

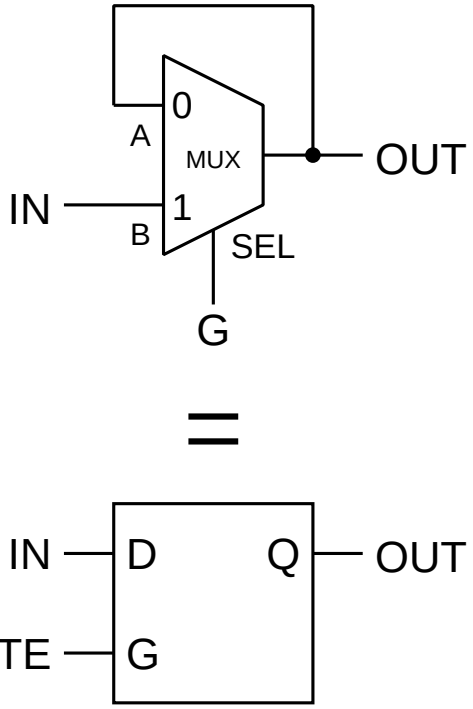
- from SEL changing, $3 \times t_g$ to OUT being stable
- from data input changing, $2 \times t_g$ to OUT changing

this delay can be exploited to produce memory!!

memory — latches

one way to make memory: use a multiplexer

- connect the output back to one of the inputs
  - e.g., *feedback* from the output to input A
- when SEL is 1, input B is copied to output
- when SEL is 0, output is copied (via A) back to itself
  - the output is held constant, *independent* of B



this is called a *transparent latch*

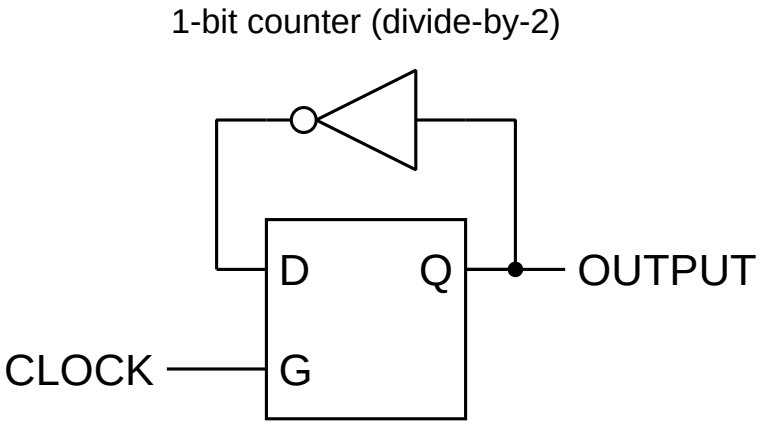
- when the G (gate) input is 1, D (input) is copied to Q (output)
  - the gate is 'open', the device is transparent
- when the G (gate) input is 0, Q (output) remains unchanged
  - the device has 'latched onto' the value of B
    - from the moment the G signal changed from 1 to 0

latch on to/onto
1. To get hold of; obtain.
2. To cling to.

the problem with latches

let's try to make a one-bit counter using a latch

- latch stores current counter value
- a *clock* signal causes the counter to update

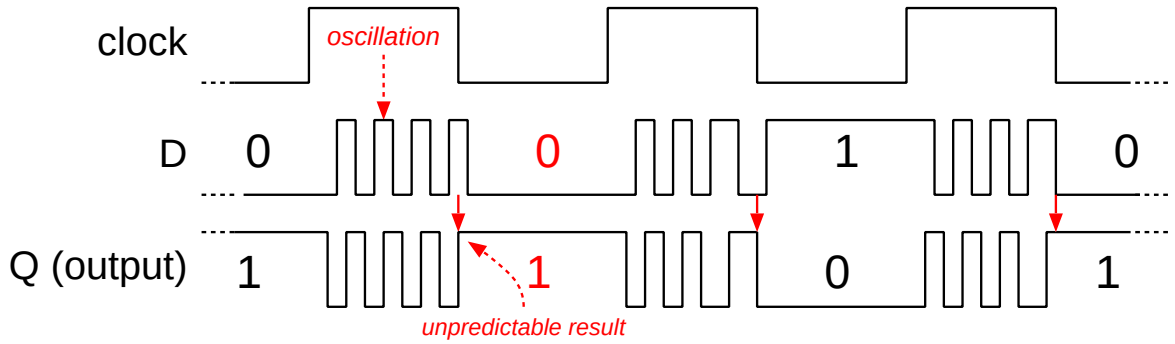


when the clock signal is low

- the output is held constant by the latch
- the inverter computes the next desired output

when the clock signal is high

- the computed next output is copied to the output
- the inverter computes the next desired output (inverse of the current output)
- the next output goes through the 'open' latch and contradicts the current output



level- vs. edge-triggering

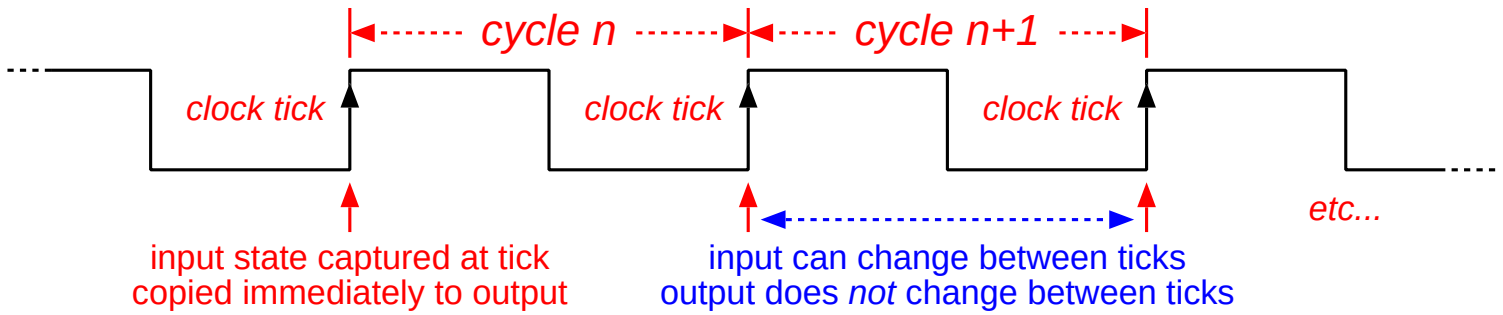
in a *level-triggered* system, when the clock is high

- the outputs can change once, and once only
- until the clock goes low again, closing all the latches

this is not very useful

much better would be an *edge-triggered* system

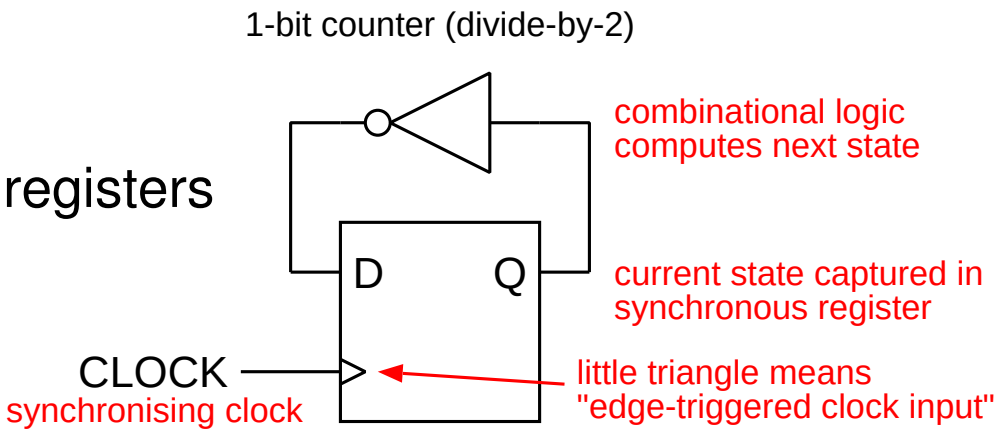
- clock *ticks* coincide with a specified clock *edge*
  - e.g., a clock transition from 0 to 1 (the *rising* edge)
- a snapshot of the input is taken at that instant, and becomes the new output
- the inputs can change arbitrarily at any other time, because. . .
- the output *always* reflects the input *at the last clock tick*



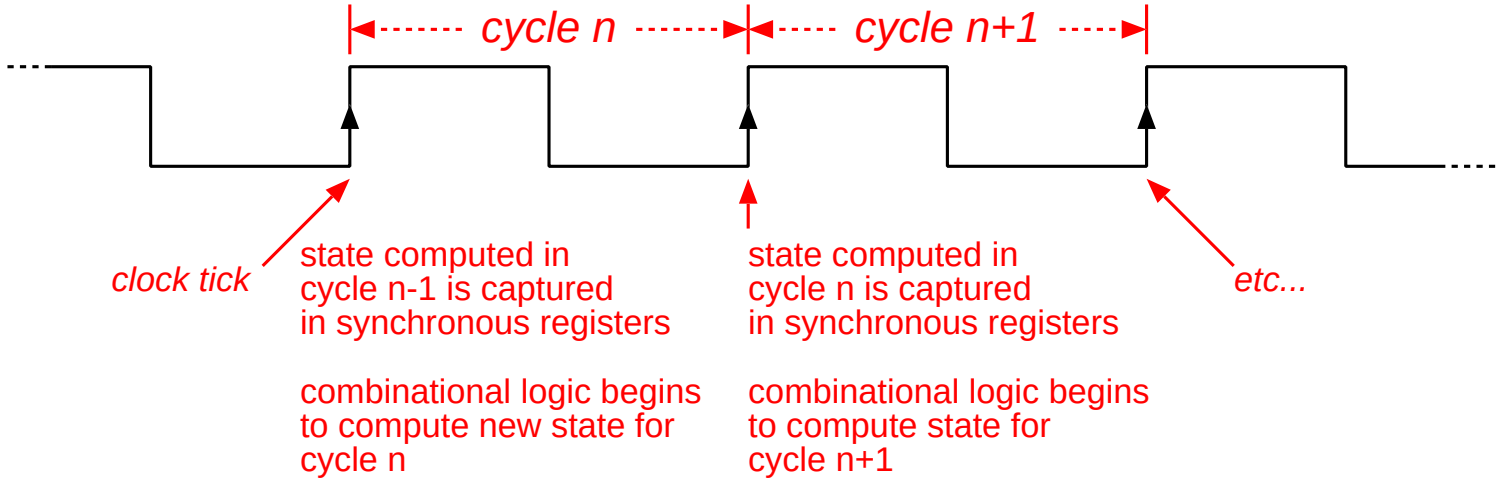
synchronous logic

at the start of each clock cycle

- the state of the machine is captured in registers
- computation of next state begins



'all' we need is an edge-triggered register



synchronous logic — register model

let's construct an edge-triggered register

- using a pair of level-triggered latches

when the clock is low

- the input gate is open
- the output gate is closed

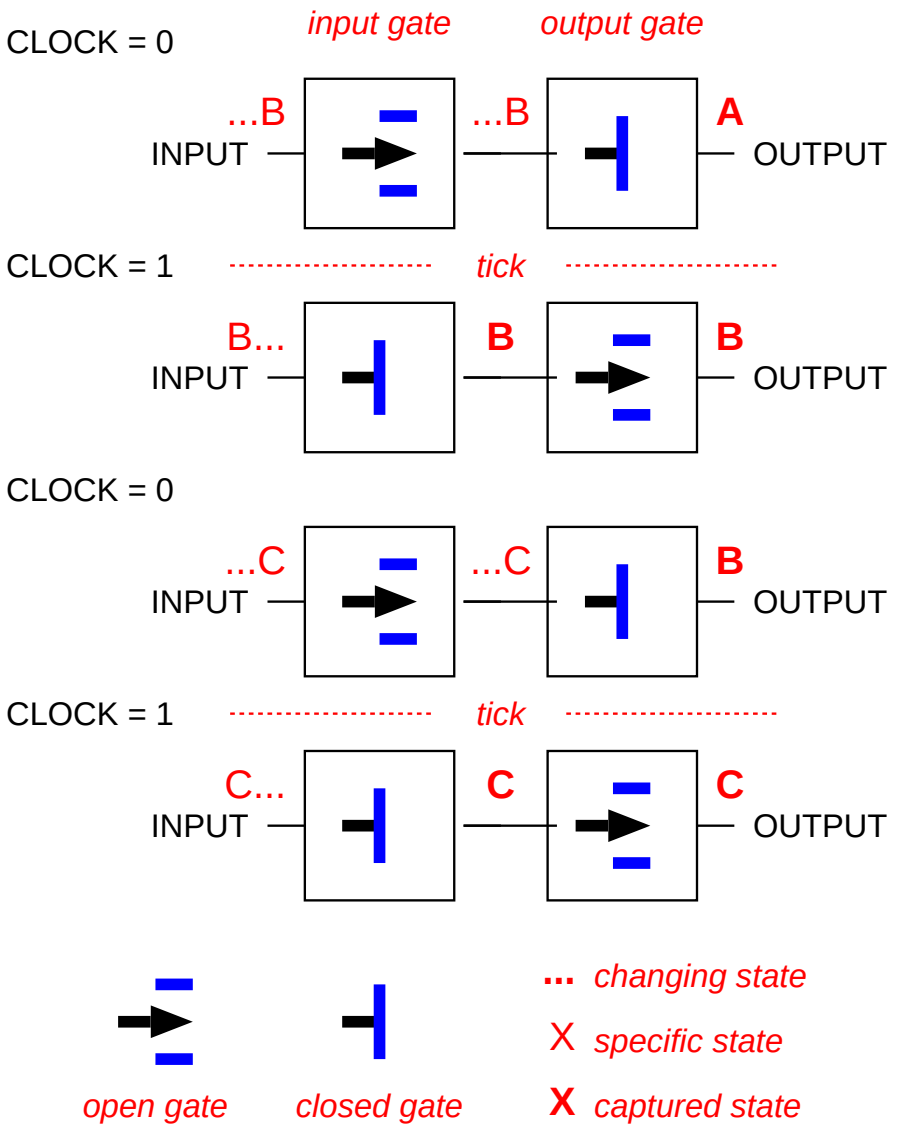
repeat:

when the clock goes high (ticks)

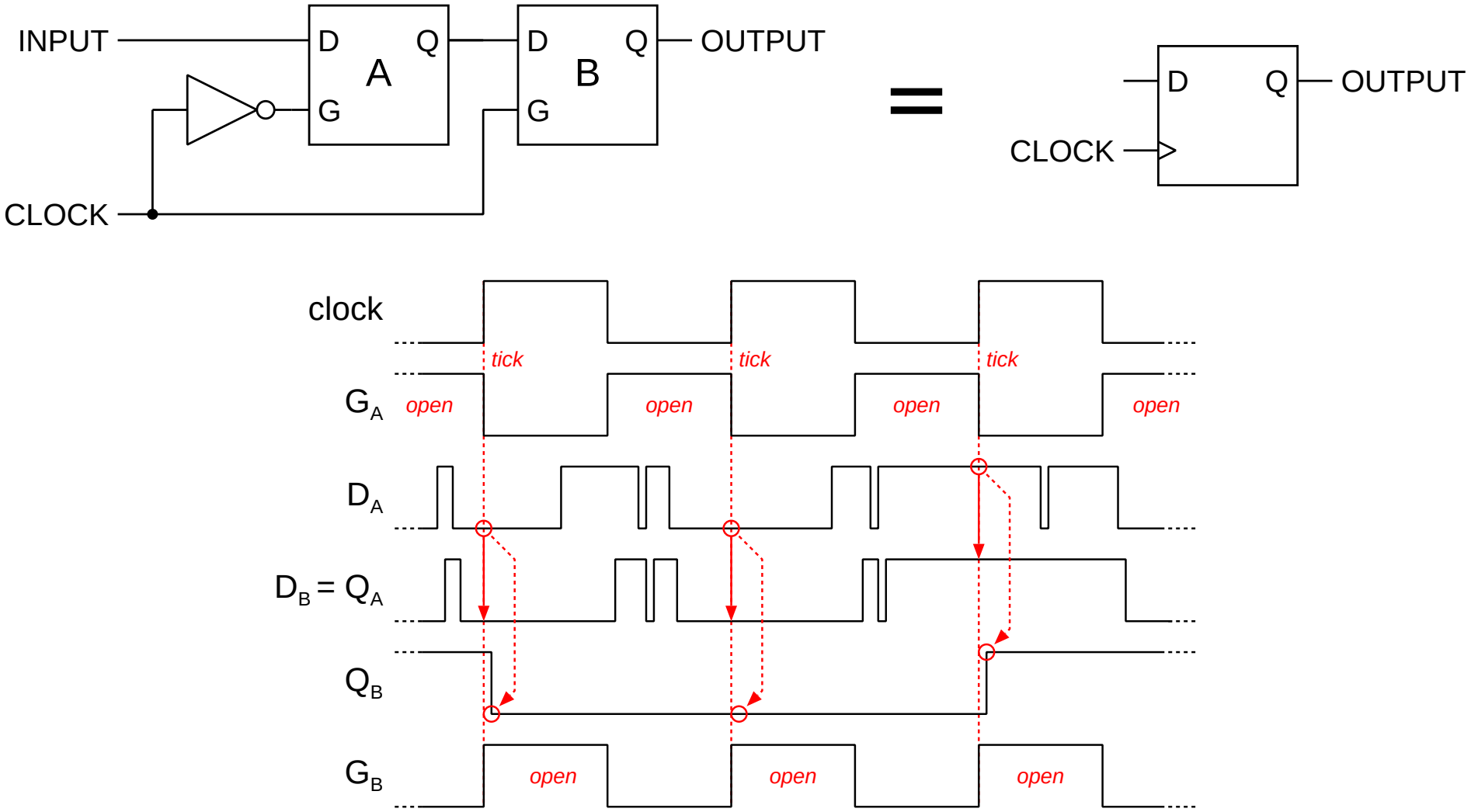
- the input gate is closed
  - the inputs are captured
- the output gate is opened
  - captured inputs are sent to output

when the clock goes low

- the output device is closed
  - captured outputs remain stable
- the input device is opened



synchronous logic — register implementation

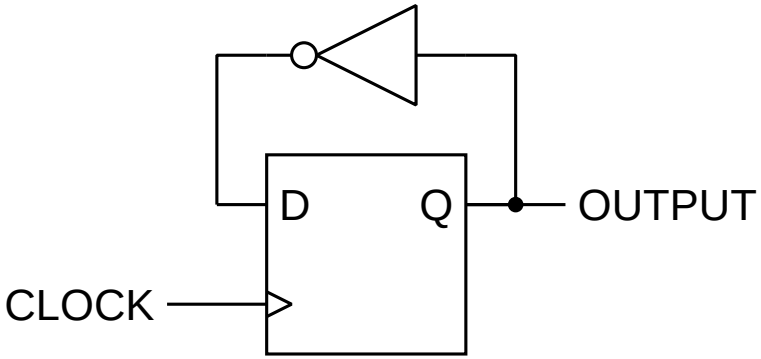


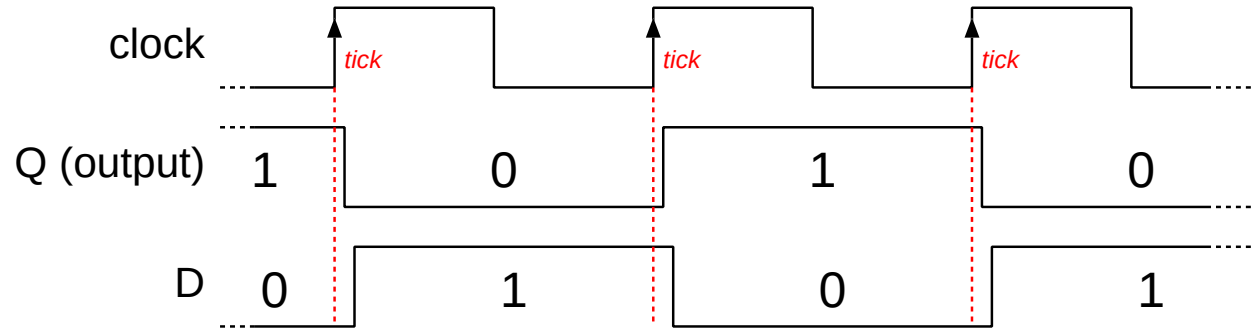
this is a *flip-flop*, a 1-bit synchronous *register*

- while one side is ‘flipping’, the other side is ‘flopping’

synchronous logic example — counters

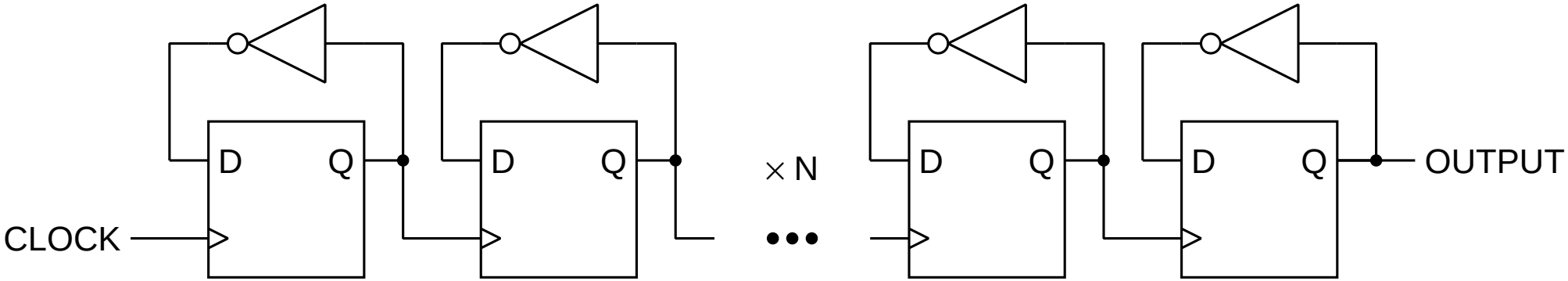
1-bit counter (divide-by-2)



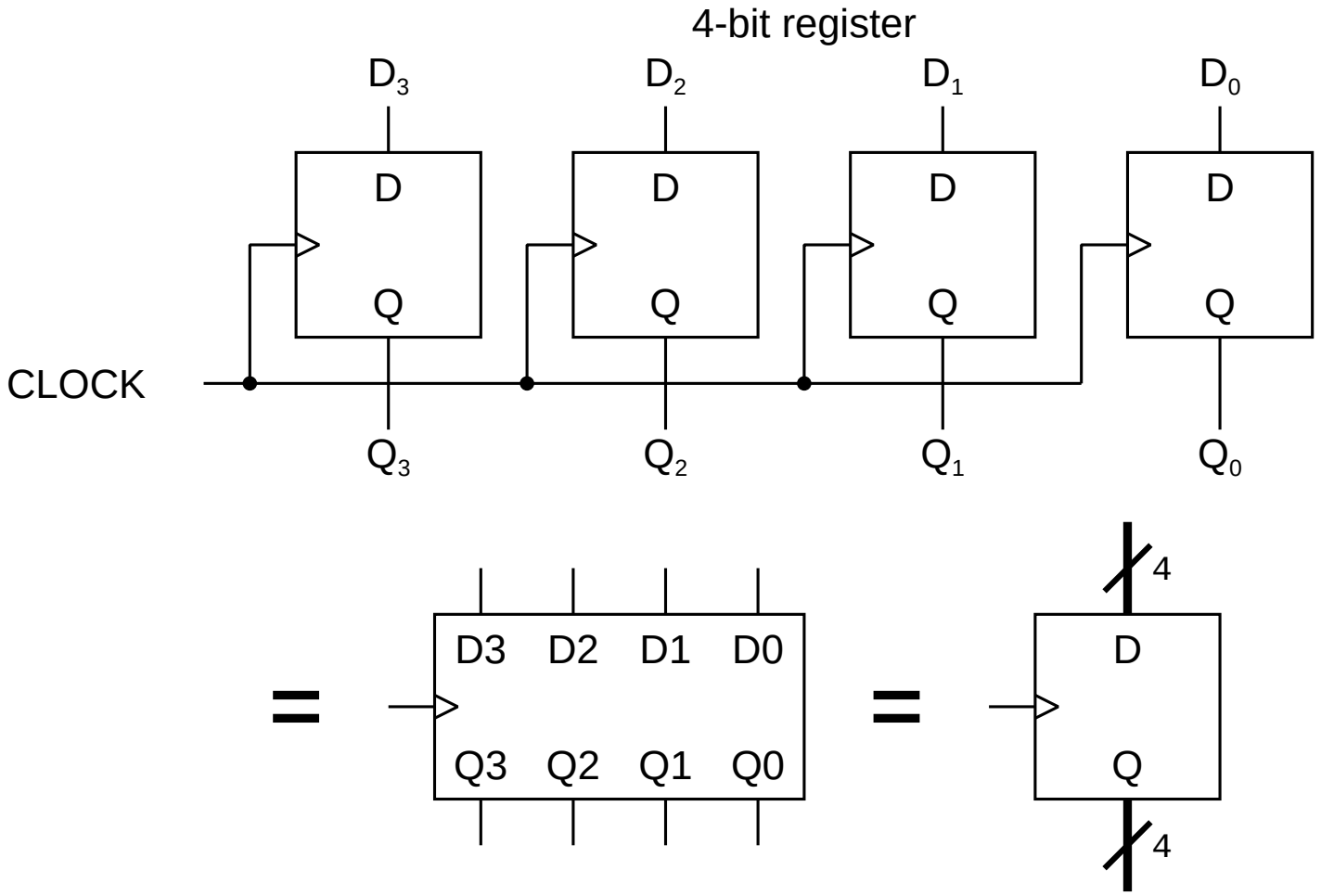


'cascade' N counters to make a 2^N-bit counter...

N-bit counter (divide-by-2^N)



synchronous logic example — multi-bit registers



the global clock signal is often omitted from the diagram

- to make the diagram easier to read
- any unconnected clock input is understood to be connected to the global clock

a simple CPU

one ‘accumulator’ register

12-bit address bus

16-bit instructions and data bus

- `CONSTANT` is a 12-bit constant

  LS 12 bits of opcode

- `OP` is a 4-bit opcode

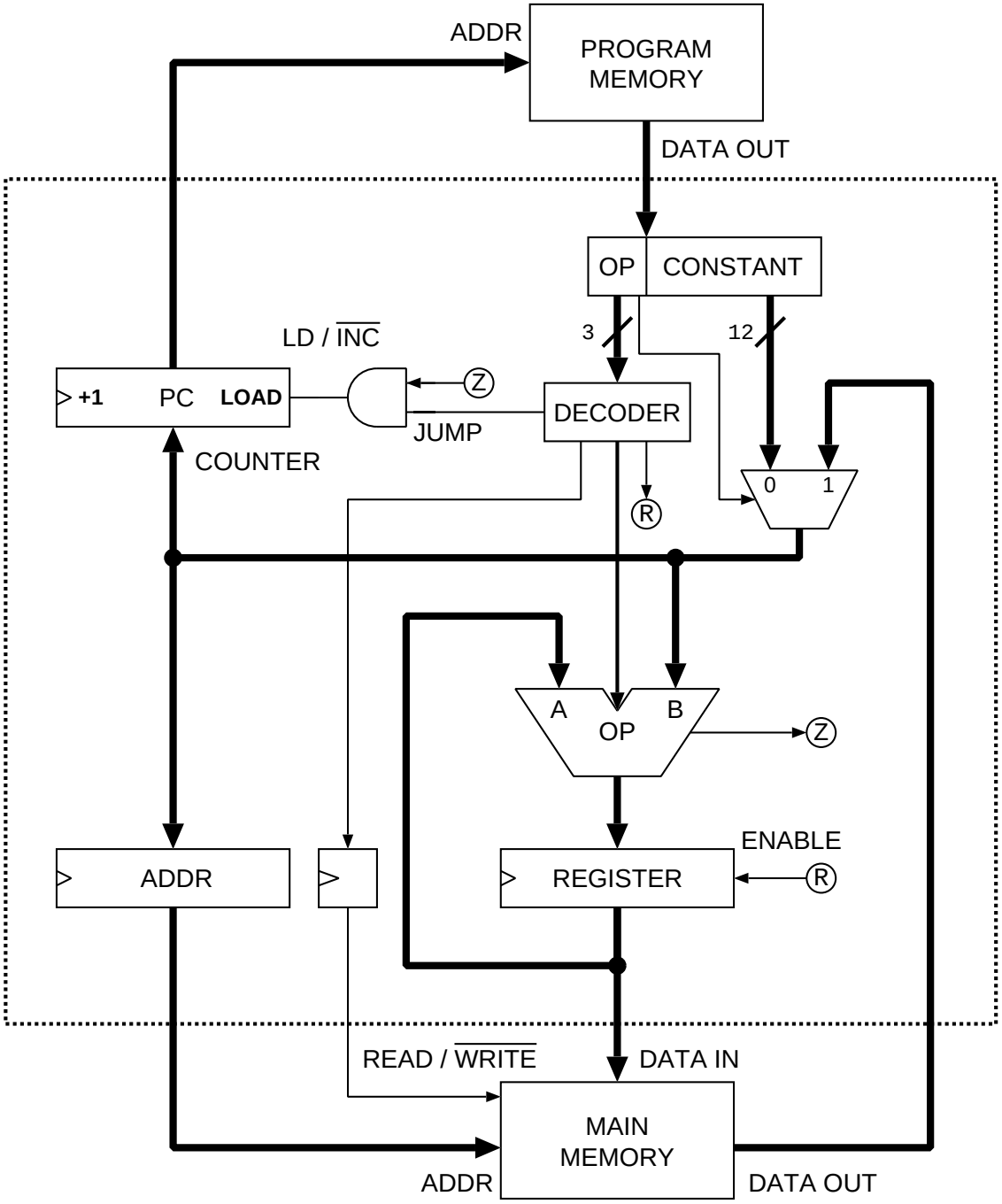
  LS bit = constant/memory select

instruction decoder:

- conditional load of PC
- read or write to RAM
- ALU operation
- register load Ⓡ

Ⓩ output from ALU

- 1 when $A = 0$



a simple CPU

decoder design

- 4-bit opcode
  - lowest bit selects ALU 'B' input: memory or constant
- eight operations

OP	*instruction*	*ALU OP*	Ⓩ	Ⓡ	READ/$\overline{\text{WRITE}}$	JUMP
	ADDR	-	-	0	1	0
	LOAD	B	-	1	1	0
	STORE	-	-	0	0	0
	ADD	A + B	-	1	1	0
	SUB	A − B	-	1	1	0
	JUMP	B	1	0	1	1
	JUMPZ	B	A=0	0	1	1

example program

insn addr	*binary opcode*	*op*	*constant*	*comment*
0:	100 0 000000000010	ADDR	2	prepare to read memory address 2
1:	000 1 000000000000	LOAD		read from memory to register
2:	001 0 000000000001	ADD	1	add 1 to register
3:	011 0 000000000010	STORE	2	store register into memory address 2
4:	101 0 000000000000	JUMP	0	repeat from instruction 0

equivalent high-level program (assuming `count` is at address `2`):

```
while (true) {
    count = count + 1;
}
```

next week

mathematics of control

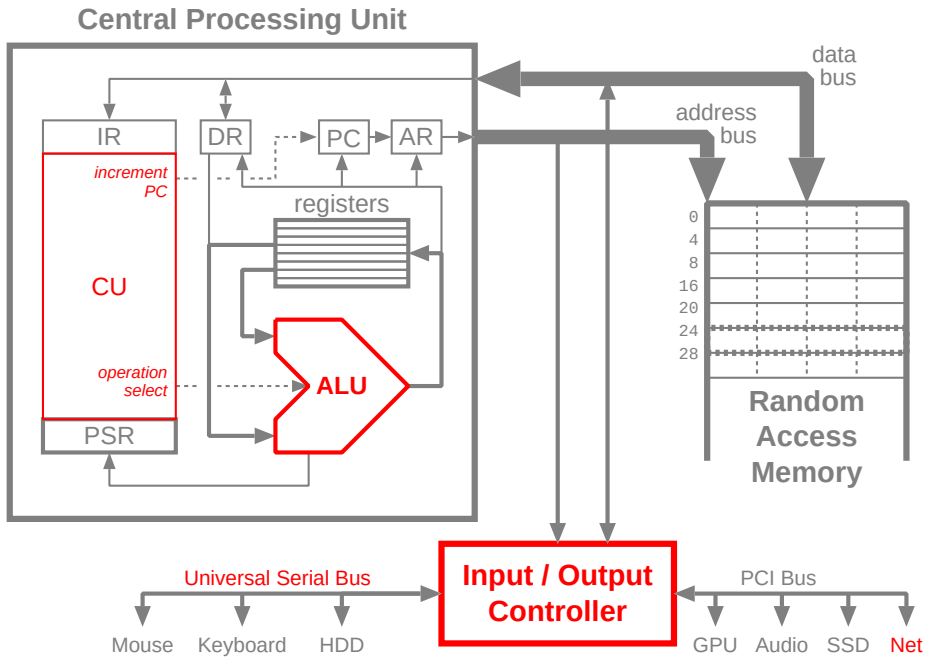
- models of stateful computation

finite state machines

- formal model
- representations

FSM applications

- pattern matching
- pattern generation
- sequencing



homework

study the simple CPU and example program

- simulate it on paper to see how it works
- one instruction at a time

reinforce your understanding

- write a simulator for the simple CPU in Python
  - model the current state of the machine
  - inpect the current instruction
  - compute the next state of the machine
  - update the current state
  - repeat
- print the state at each step
- run a simple program

ask about anything you do not understand

- from any of the classes so far this semester (or the lecture notes)
- it will be too late for you to try to catch up later!
- I am always happy to explain things differently and practice examples with you

glossary

clock — a periodic signal that synchronises the operation of elements within a logic circuit.

complemented — inverted, or ‘flipped’.

edge-triggered — a device that performs its operation when a clock signal changes state.

edge — the vertical part of a square-shaped waveform.

feedback — connecting the output of a device or circuit back to an input.

gate delay — the time taken for the output of a gate to change in response to a changing input.

latch — a device that either copies its input to its output, or holds the output constant, depening on the state of a gate signal.

level-triggered — a device that performs its operation when a clock signal is in a particular state (high or low).

register — a device that can remember one or more bits of information.

rising — the edge of a clock signal corresponding to a change of state from 0 to 1.

stateful — a device that can remember its past inputs and/or outputs.

stateless — a device that has no memory and responds simply to the current set of inputs.

tick — the rising edge of a repeating clock signal, analogous to the ticking of a mechanical clock.