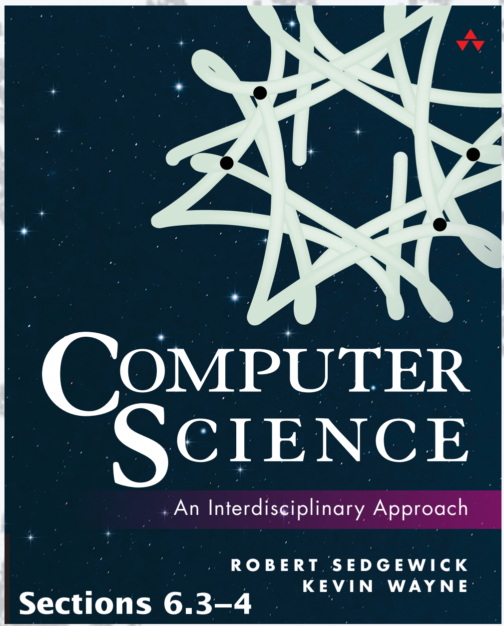
COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

20. Central Processing Unit

20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- Components and connections

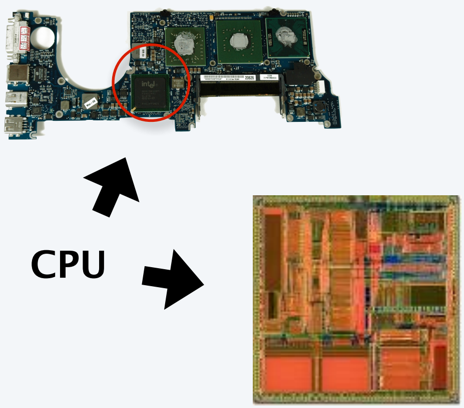
Let's build a computer!

CPU = Central Processing Unit

Computer
- Display
- Touchpad
- Battery
- Keyboard
- ...
- CPU (difference between a TV set and a computer)

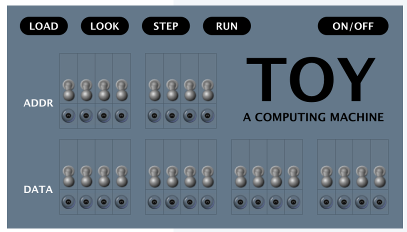
Previous lecture
- Combinational circuits
- ALU (calculator)

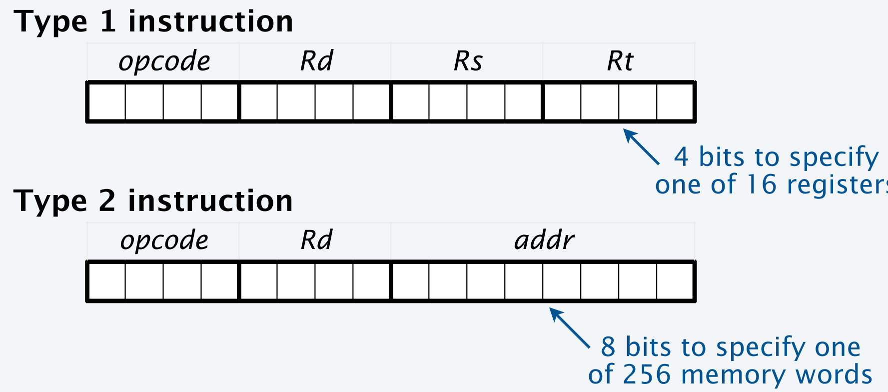
This lecture
- Sequential circuits with *memory*
- CPU (computer)



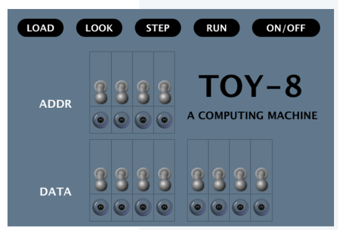
A smaller computing machine: TOY-8

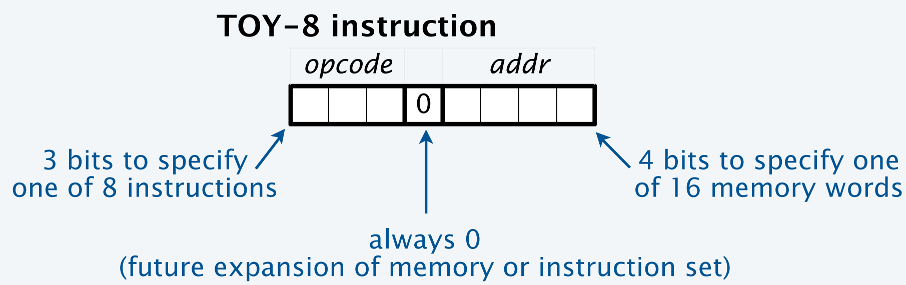
TOY instruction-set architecture.
- 256 16-bit words of memory.
- 16 16-bit registers.
- 1 8-bit program counter.
- 2 instruction types.
- 16 instructions.





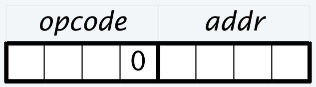
TOY-8 instruction-set architecture.
- 16 8-bit words of memory.
- 1 8-bit register.
- 1 4-bit program counter.
- 1 instruction type.
- 8 instructions.





Purpose of TOY-8. Illustrate CPU circuit design for a "typical" computer.

TOY-8 reference card



opcode	*operation*	*pseudo-code*
0	halt	`halt`
2	add	`R = R + M[addr]`
4	bitwise and	`R = R & M[addr]`
6	bitwise xor	`R = R ^ M[addr]`
8	load addr	`R = addr`
A	load	`R = M[addr]`
C	store	`M[addr] = R`
E	branch zero	`if (R == 0) PC = addr`

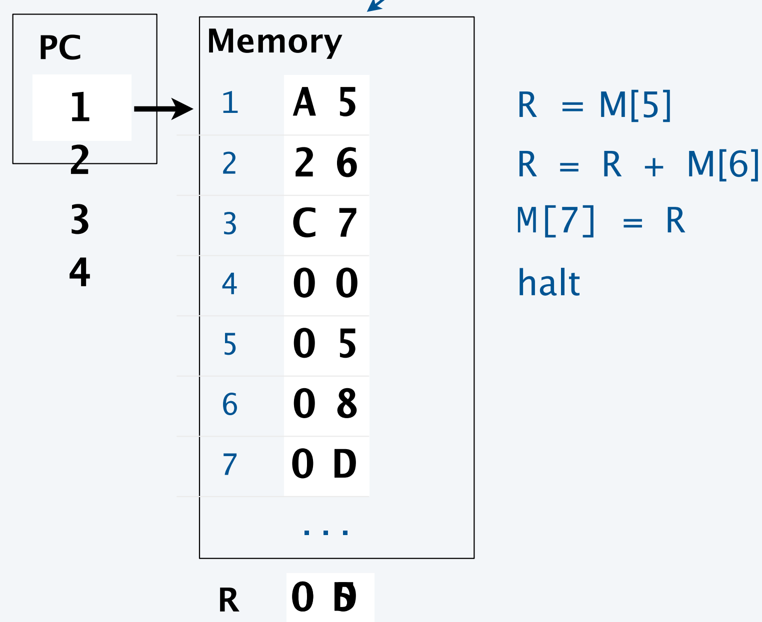
ZERO M[0] is always 0.

STANDARD INPUT Load from M[F].

STANDARD OUTPUT Store to M[F].

A TOY-8 program

Add two numbers from memory and put sum in memory



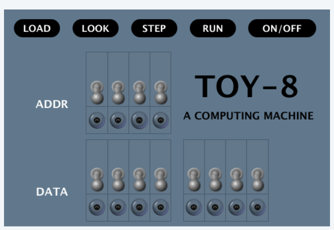
Challenge for the bored:

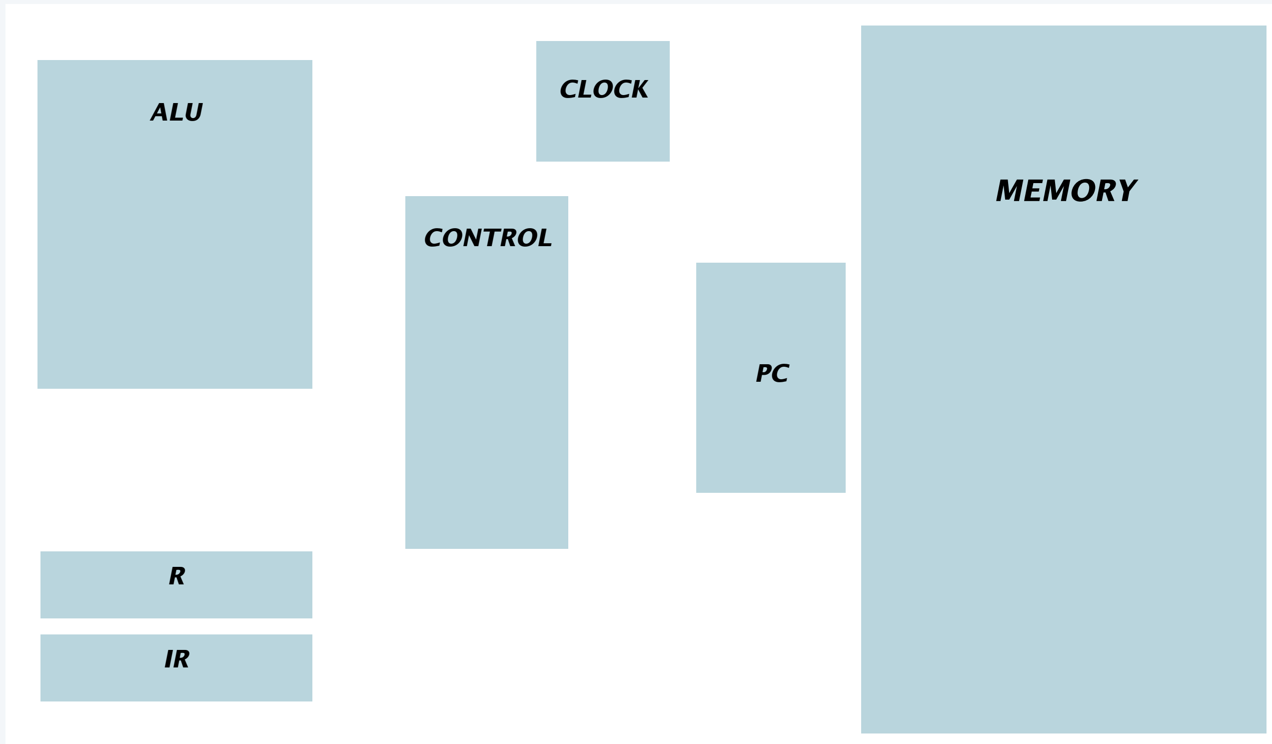
Write Fibonacci seq, add numbers on stdin, ...

CPU circuit components for TOY-8

TOY-8 CPU

- ALU (adder, AND, XOR)
- Memory
- Register (R)
- PC
- IR
- *Control*
- *Clock*





Goal. Complete CPU circuit for TOY-8 (same design extends to TOY and to your computer).

Review: Components, busses, and control lines

Basic design of our circuits

- Organized as *components* (functional units of TOY: ALU, memory, register, PC, and IR).
- Connected by *busses* (groups of wires that propagate information between components).
- Controlled by *control lines* (single wires that control circuit behavior).

Conventions

- Bus inputs are at the top,
  input connections are at the left.
- Bus outputs are at the bottom,
  output connections are at the right.
- Control lines are blue.

These conventions *make circuits easy to understand*.
(Like style conventions in coding.)

Perspective

Q. Why TOY-8?

A. TOY circuit width would be about 5 times TOY-8 circuit width.

Sobering fact. The circuit for your computer is *hundreds* to *thousands* of times wider.

Reassuring fact. Design of all three is based on the same fundamental ideas.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

CS.20.A.CPU.Overview

20. CPU

- Overview
- **Bits, registers, and memory**
- Program counter
- Connections

Sequential circuits

Q. What is a sequential circuit?

A. A digital circuit (all signals are 0 or 1) *with feedback* (loops).

Q. Why sequential circuits?

A. *Memory* (difference between a DFA and a Turing machine).

Basic abstractions

- On and off.
- Wire: Propagates an on/off value.
- Switch: Controls propagation of on/off values through wires.
- Flip-flop: *Remembers* a value (next).

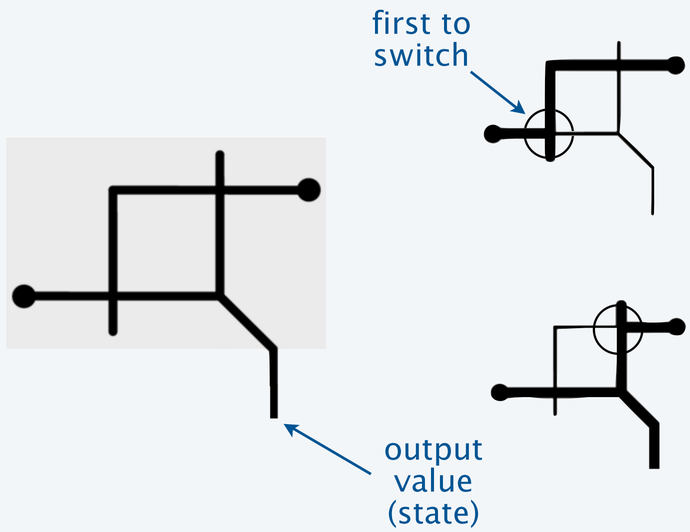
Simple circuits with feedback

Loops in circuits lead to time-varying behavior

- *Sequence* of switch operation matters.
- Need tight control (see next slide).

Example 1. Two switches, each blocked by the other.

- State determined by whichever switches first.
- Stable (once set, state never changes).
- *Basic building block for memory circuits.*



Example 2. Three switches, blocked in a cycle.

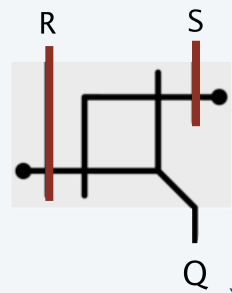
- State determined by whichever switches first.
- *Not* stable (cycles through states).

a "buzzer"

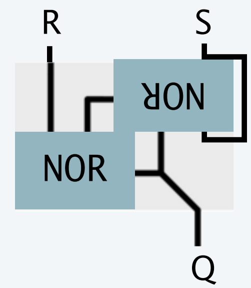
A new ingredient: Circuits with memory

An *SR flip-flop* controls feedback.

- Add control lines to switches in simple feedback loop.
- R (reset) sets state to 0.
- S (set) sets state to 1.
- Q (state) is always available.

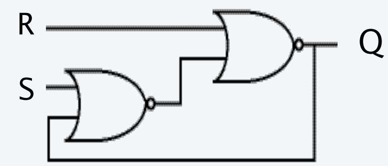


component level



output value (state)

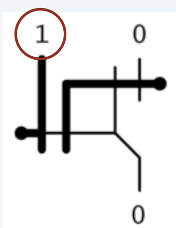
classic notation



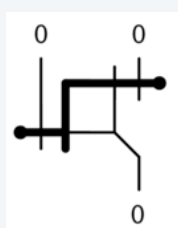
"cross-coupled *NOR* gates"

examples

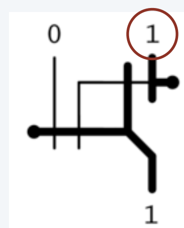
R: set to 0



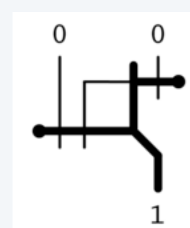
stays 0



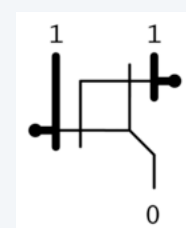
S: set to 1



stays 1



unused

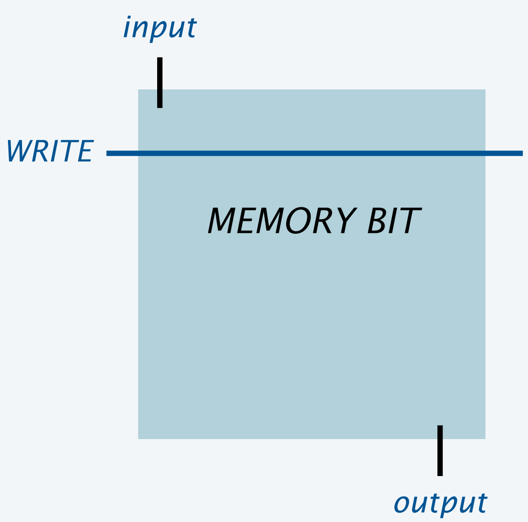


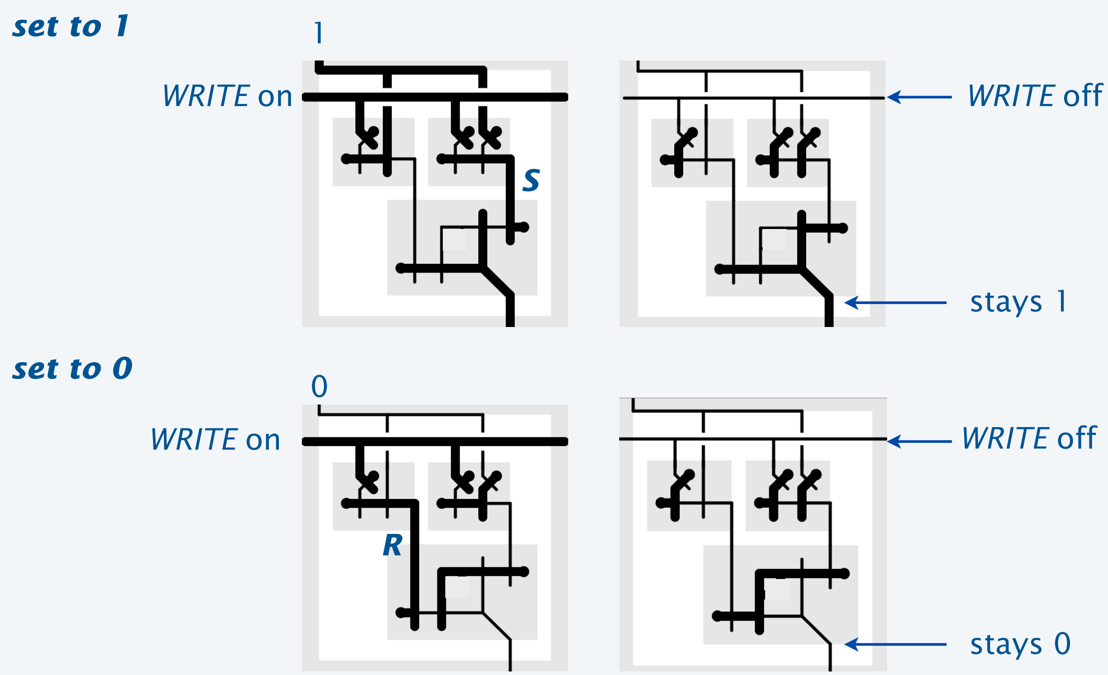
Caveat. Timing of switch vs. propagation delay.

Flip-flop application: Memory bit

Add logic to an SR flip-flop for more precise control

- Provide data value on an *input* wire instead of using S and R controls.
- Use *WRITE* control wire to enable change in flip-flop value.
- Flip-flop value always available as *output*.





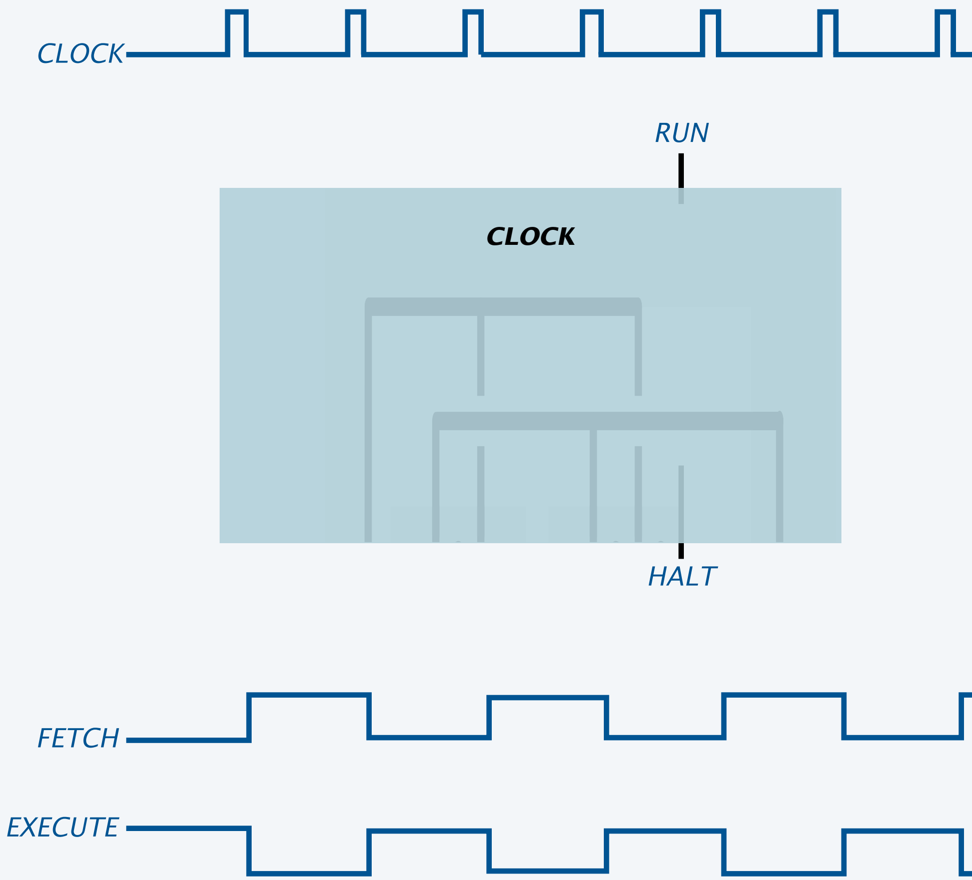
Memory bit application I: fetch/execute clock

Assumptions

- Physical clock provides regular on/off pulses.
- Duration is *just enough* to trigger a flip-flop.
- Space between pulses is long enough to allow the longest chain of flip-flops in the circuit to stabilize.

Fetch/execute clock. Attach clock to a memory bit.

- *RUN* control wire starts clock when on.
- *HALT* control wire stops clock when on.
- Memory bit flips on each clock tick (flip value and feed back to input).
- Result: on/off sequence for *FETCH* and *EXECUTE* control wires that control the CPU (stay tuned).



Fetch/execute clock with write pulses

Generates on/off signals for 4 control wires

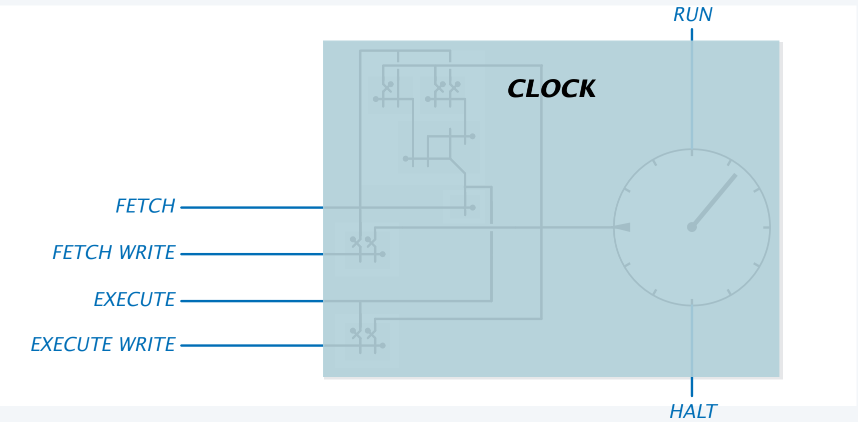
- *FETCH.*
- *FETCH WRITE* pulse.
- *EXECUTE.*
- *EXECUTE WRITE* pulse.

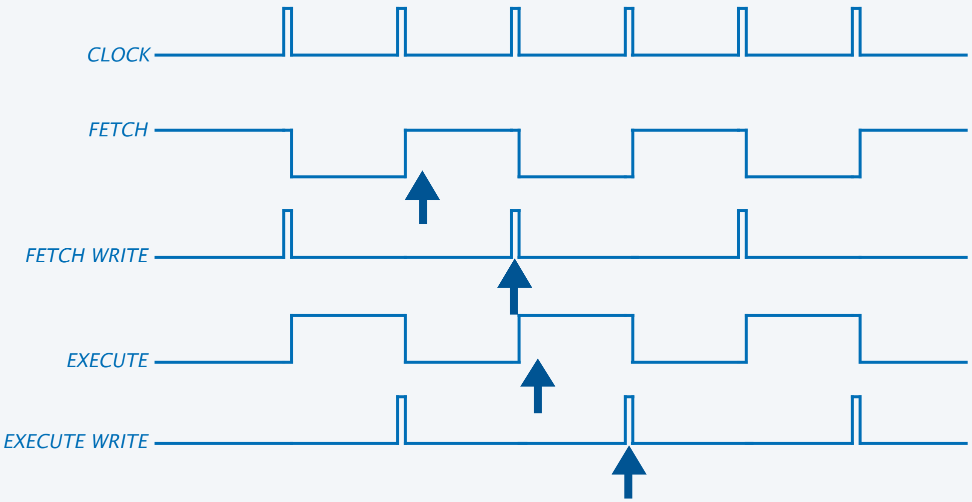
Implementation

- Add AND gates to fetch/execute clock.
- *FETCH WRITE = FETCH AND CLOCK.*
- *EXECUTE WRITE = EXECUTE AND CLOCK.*

Application

- Implements CPU fetch/execute cycles.
- Signals turn on control wires that change the state of PC, R, IR, and memory.





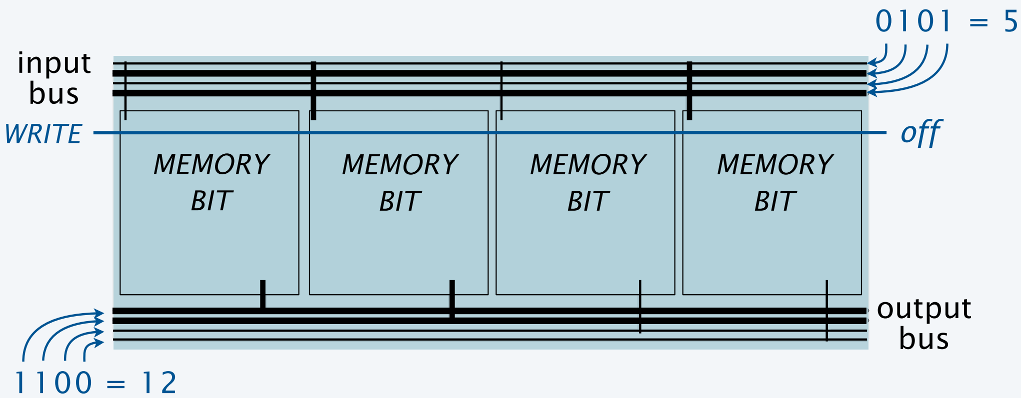
Interesting events occur at four distinct times in the cycle

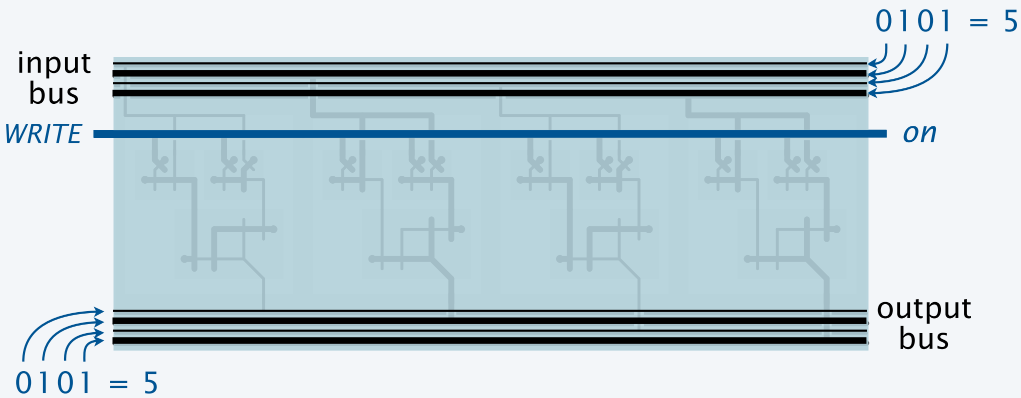
Memory bit application II: Register

Register
- *w* memory bits.
- *w*-bit input bus.
- values available on output bus.
- input loaded on *WRITE* pulse.

Implementation
- Connect memory bits to busses.
- Use *WRITE* pulse for all of them.

Applications for TOY-8
- PC holds 4-bit address.
- IR holds 8-bit instruction.
- R holds 8-bit data value.





Memory bit application III: Memory bank

Memory bank

- 2^n words, each w bits.
- n-bit address input.
- w-bit input bus.
- value *of selected word* available on output bus.
- input loaded *to selected word* on *WRITE* pulse.

Example: 4-words, each 6 bits

Memory bit application III: Memory bank

Memory bank

- 2^n words, each *w* bits.
- *n*-bit address input.
- *w*-bit input bus.
- value *of selected word* available on output bus.
- input loaded *to selected word* on *WRITE* pulse.

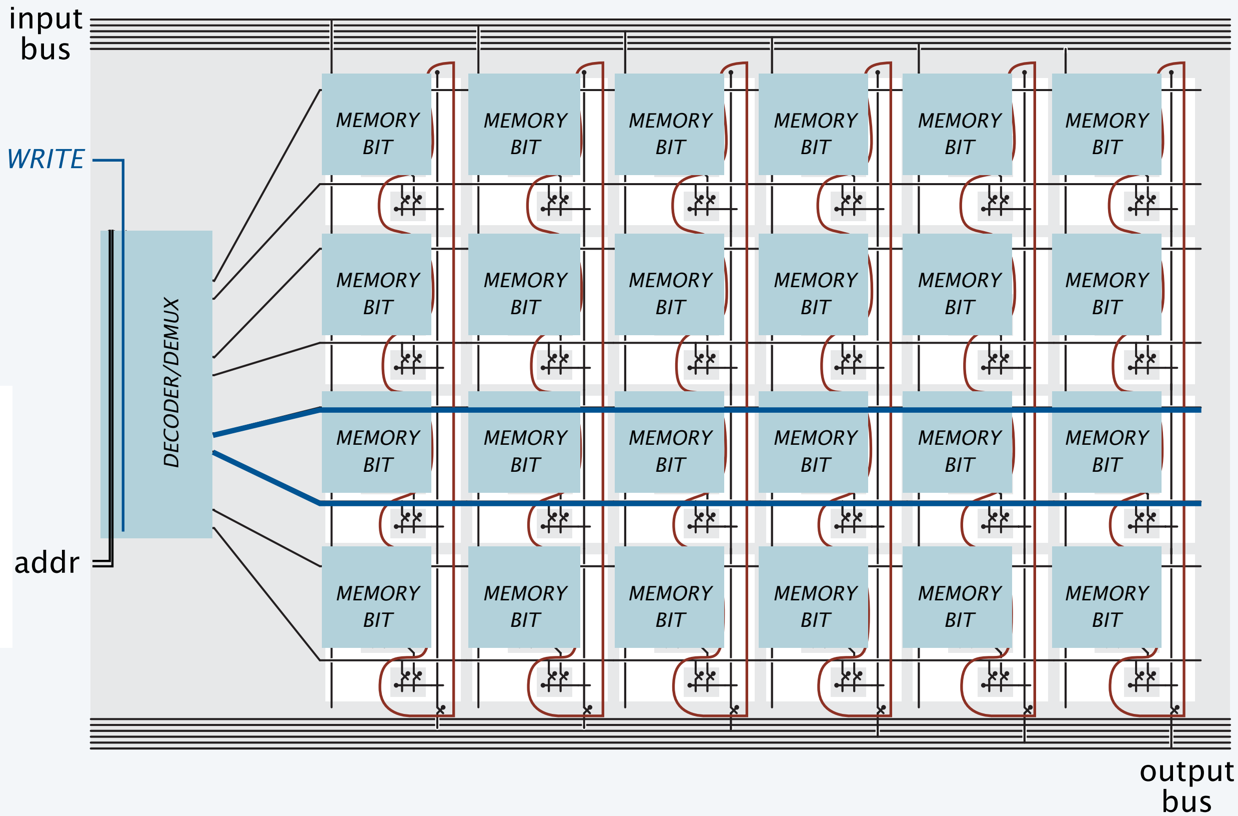
Implementation

- Decoder/demux selects word.
- One-hot muxes take selected word to output bus.

Application for TOY-8

- Main memory.

Example: 4-words, each 6 bits



TOY-8 main memory bank

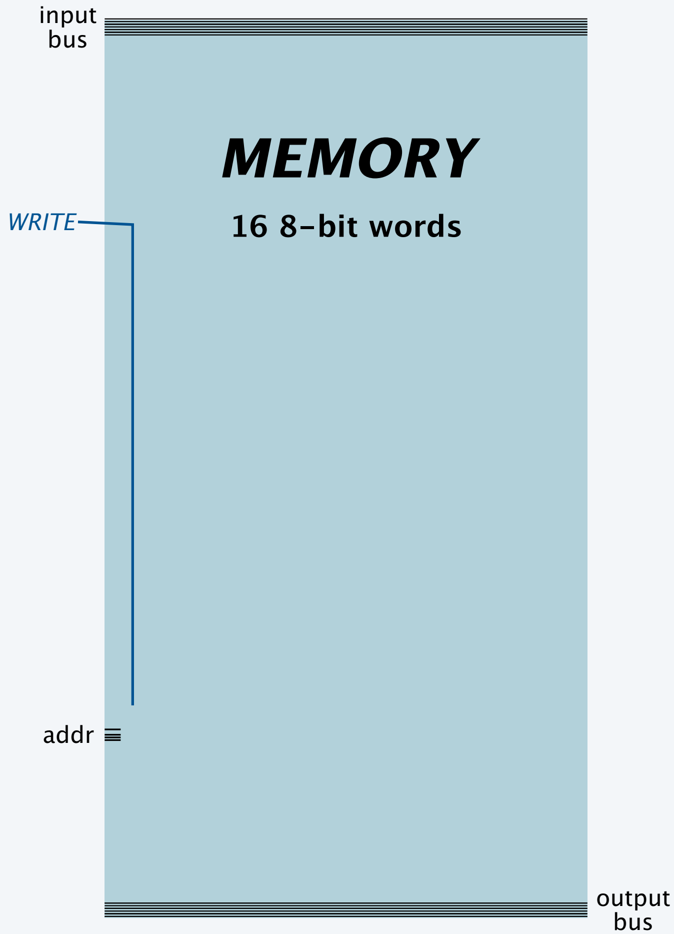
Interface

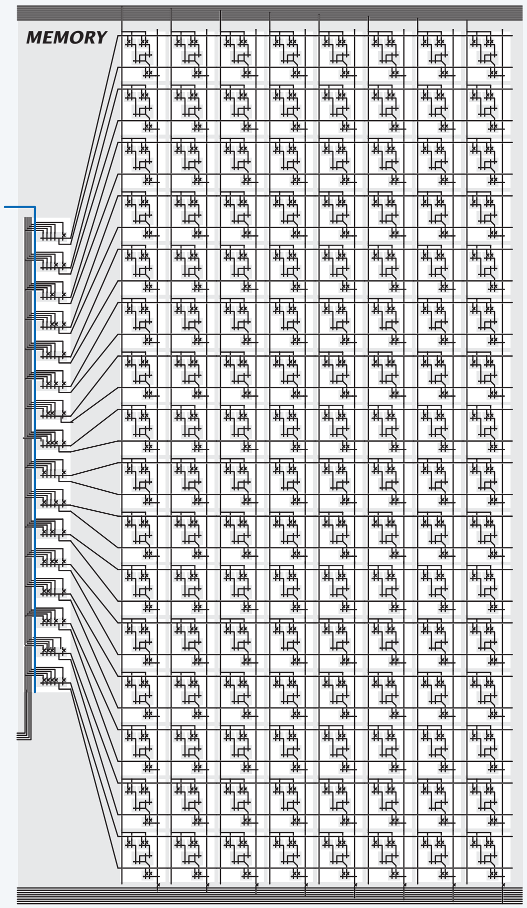
- Input bus (for *store*)
- Output bus (for *load*)
- Address to select a word
- *Write* control signal

Connections

- Input bus from registers
- Output bus to IR and R
- Address bits from PC and IR

	words	*bits/word*	*addr bits*
TOY-8	16	8	4
TOY	256	16	8
your computer	1 billion	64	32





COMPUTER SCIENCE
SEDGEWICK / WAYNE

CS.20.B.CPU.Memory

20. CPU

- Overview
- Bits, registers, and memory
- **Program counter**
- Components and connections

Designing a digital circuit: overview

Steps to design a digital (sequential) circuit

- Design interface: input busses, output busses, control signals.
- Determine components.
- Determine connections.
- Establish control sequence.

Warmup. Design TOY-8 program counter (PC).

First challenge. Need an *incrementer* circuit.

Second challenge. Multiple bus connections.

Pop quiz on combinational circuit design

Q. Design a circuit to compute $x + 1$.

Pop quiz on combinational circuit design

Q. Design a circuit to compute $x + 1$.

A. Start with a bitwise adder
- Delete y inputs, set carry in to 1.
- Compute carry with AND and sum with XOR.

c_4	c_3	c_2	c_1	1
+	x_3	x_2	x_1	x_0
	z_3	z_2	z_1	z_0

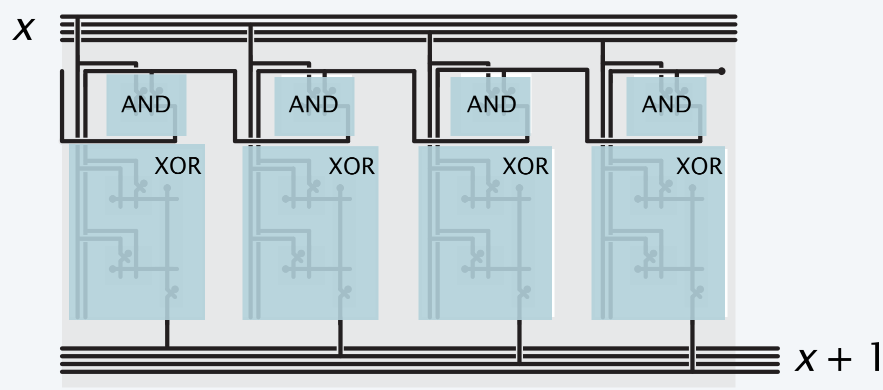
carry bit

x_i	c_i	c_{i+1}	AND
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

sum bit

x_i	c_i	z_i	XOR
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

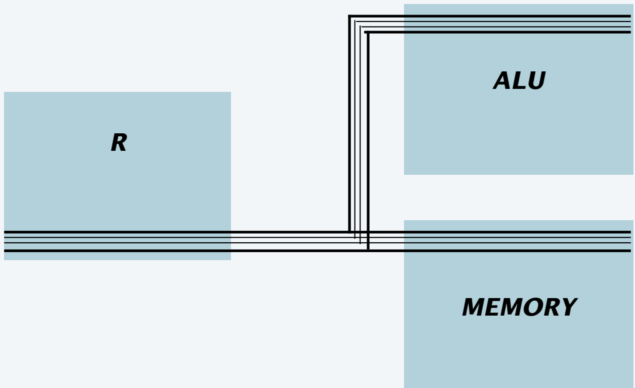
4-bit incrementer



Multiple bus connections

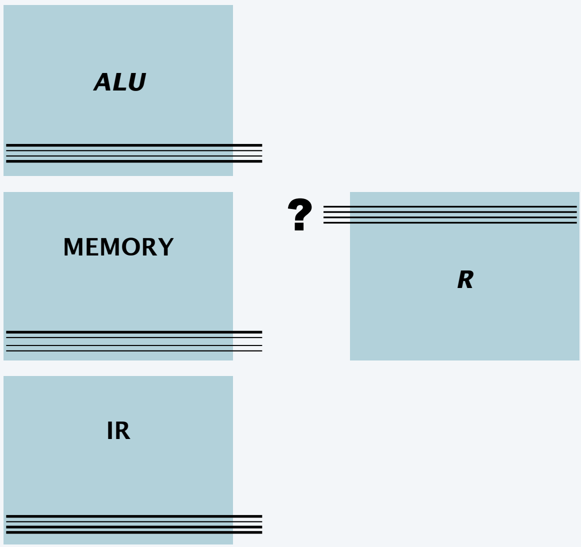
If component *outputs go to* multiple other components

- No problem, just use T connections.
- Values on both busses are the same.
- Example: Register connects to ALU and memory.



If component *inputs come from* multiple other components

- Problem.
- Values on the busses are *different*.
- Example: ALU, memory, and IR connect to register.
- Solution: Need a *selector switch* (bus mux).

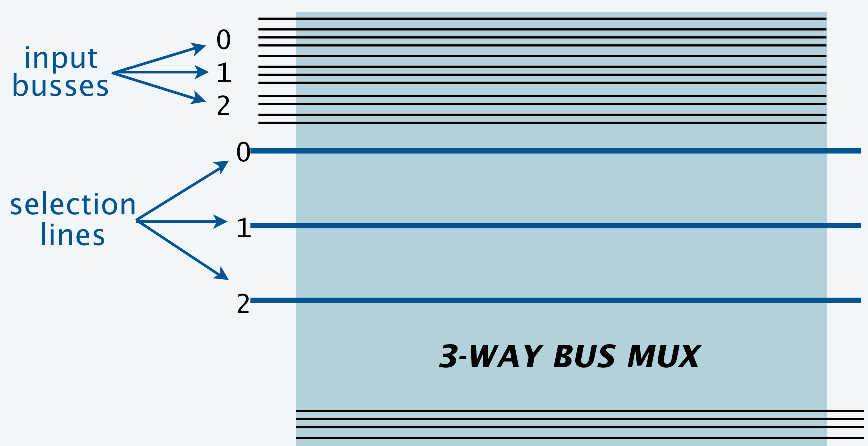


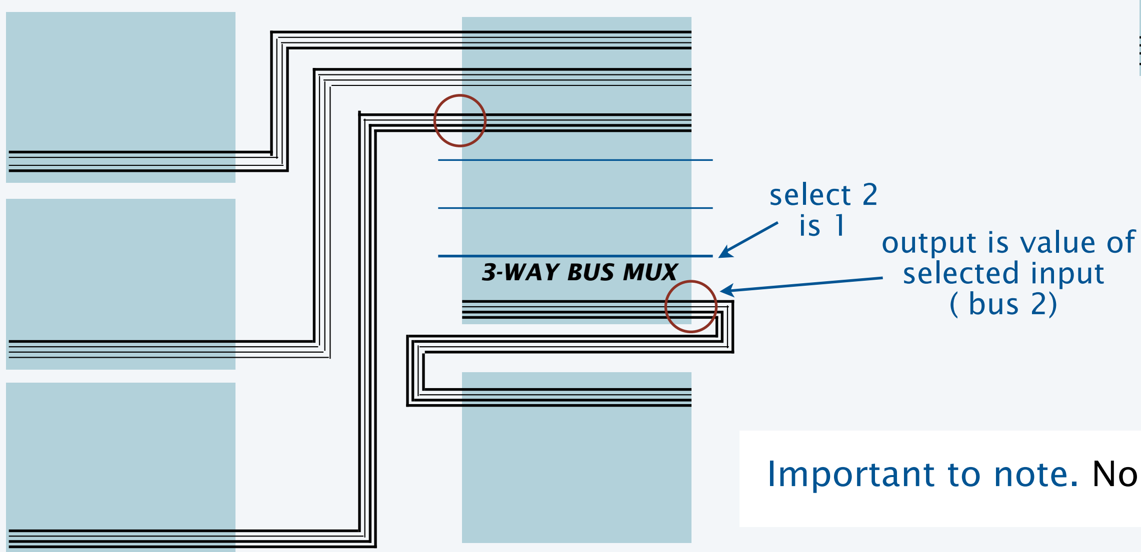
A (virtual) bus selector switch

One-hot *m*-way bus mux

- *m* input busses.
- *m* control lines (selection).
- One output bus.
- At most one selection line is 1.
- Output bus lines have same value as selected input bus lines.

Example: 4-bit 3-way bus mux



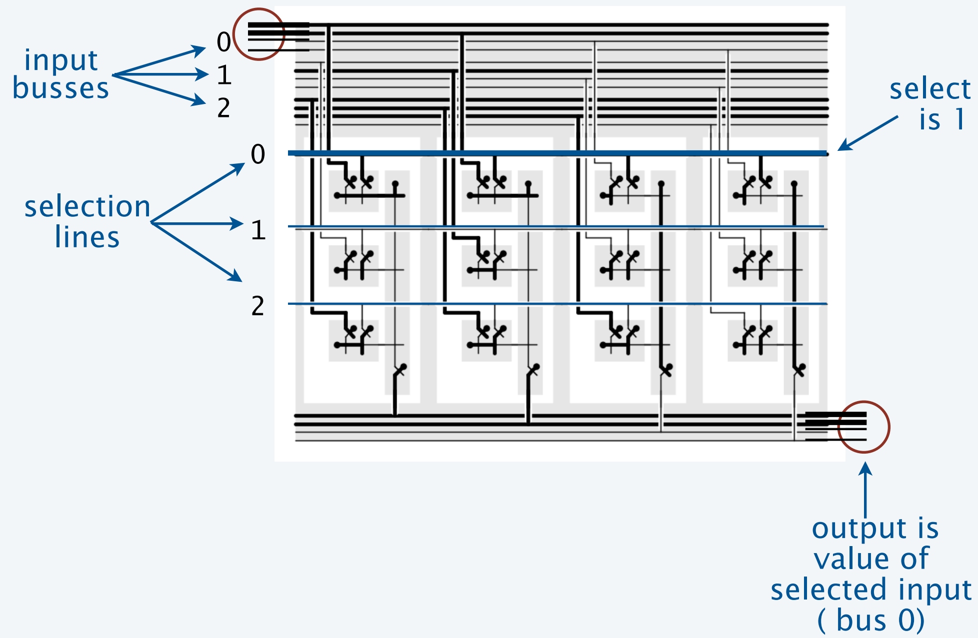


Important to note. No direct connection from input to output.

One-hot bus mux

One-hot *m*-way bus mux

- *m* input busses.
- *m* control lines (selection).
- One output bus.
- At most one selection line is 1.
- Output bus lines have same value as selected input bus lines.



Implementation

- Bitwise one-hot muxes for output.

Application (next): Select among inputs to a component.

Program counter (PC)

The *PC* holds an address and supports 3 control wires:

- *INCREMENT.* Add 1 to value when *WRITE* becomes 1.
- *LOAD.* Set value from input bus when *WRITE* becomes 1.
- *WRITE.* Enable PC address to change as specified.

The current address is always available on the output bus.

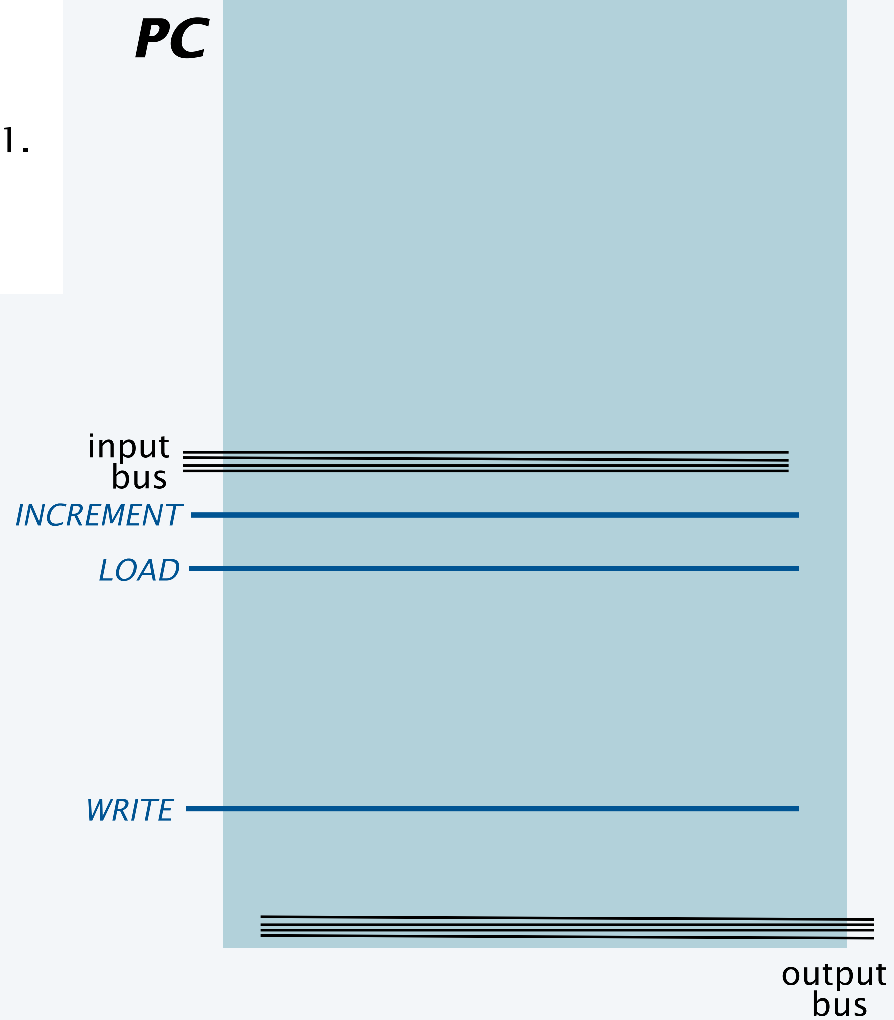
Components

- PC register (4-bit).
- Incrementer (add 1).

Connections

- Input bus to PC register.
- Incrementer to PC register.
- PC register to incrementer.
- PC register to output bus.

need 2-way bus mux



Program counter (PC)

The *PC* holds an address and supports 3 control wires:

- *INCREMENT*. Add 1 to value when *WRITE* becomes 1.
- *LOAD*. Set value from input bus when *WRITE* becomes 1.
- *WRITE*. Enable PC address to change as specified.

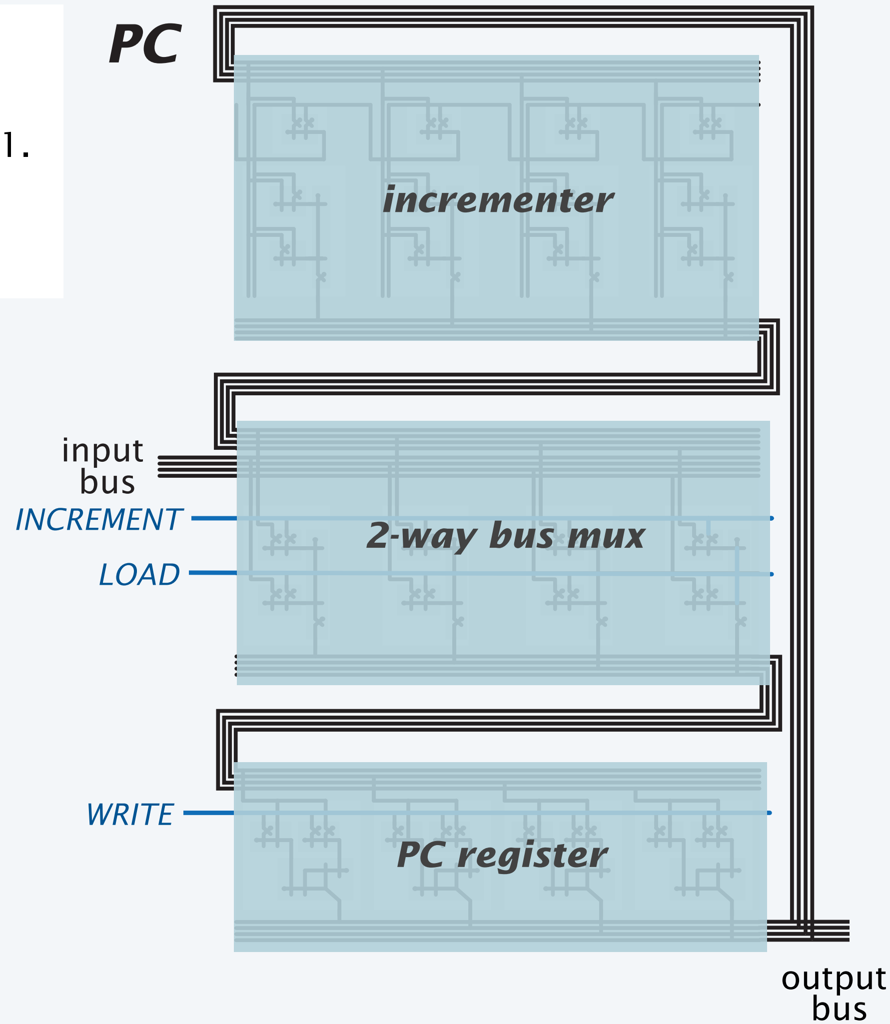
The current address is always available on the output bus.

Components

- PC register (4-bit).
- Incrementer (add 1).
- 2-way bus mux.

Connections

- Input bus to bus mux.
- Incrementer to bus mux.
- Bus mux to PC register.
- PC register to incrementer.
- PC register to output bus.

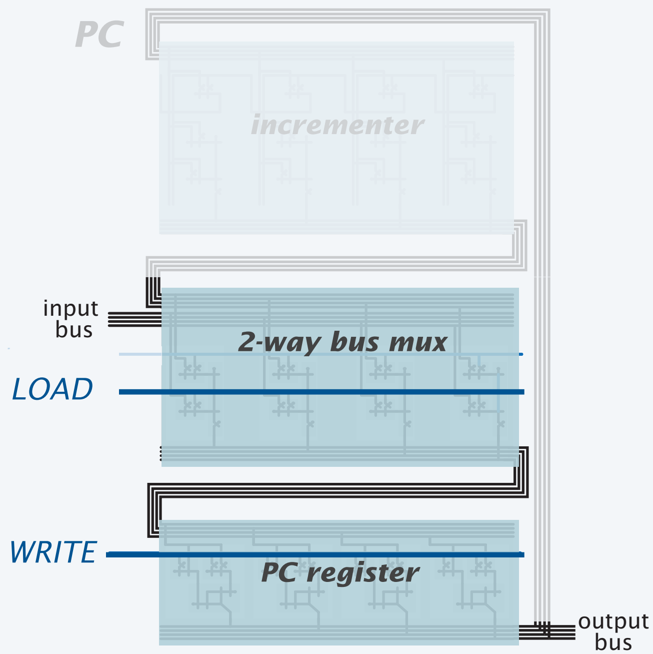


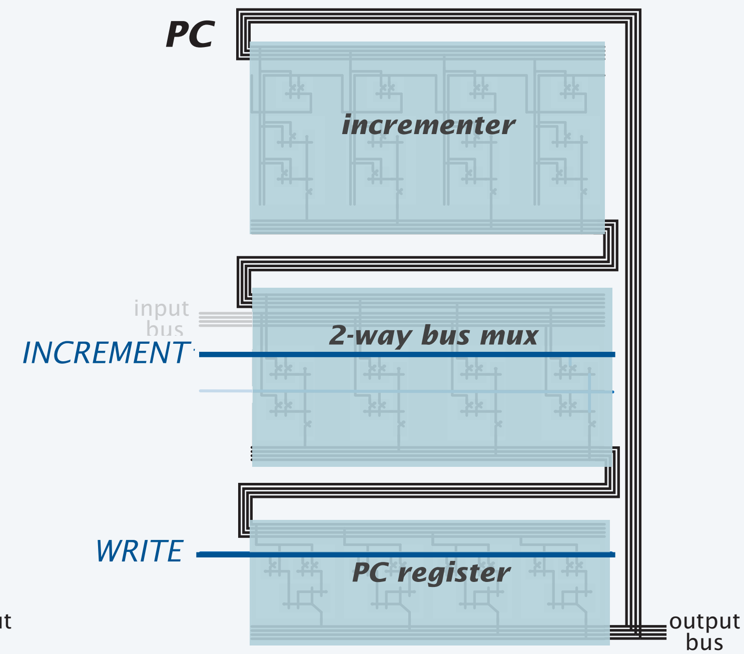
Summary of TOY-8 PC circuit

The *PC* supports two control-signal sequences:

- *Load, then write.* Set address from input bus (example: branch instruction).
- *Increment, then write.* Add one to value.

Address is written to the PC register in both cases and always available on the output bus.





Next. CPU circuit.

Important note: *write* pulse must be very short because of the cycle in this circuit.

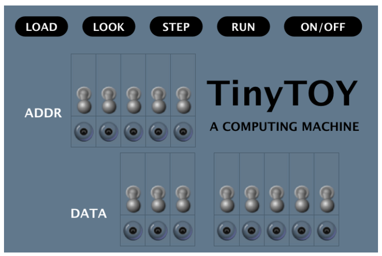
CS.20.C.CPU.PC

20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- **Components, connections, and control**

TOY-8: Interface

CPU is a circuit inside the machine

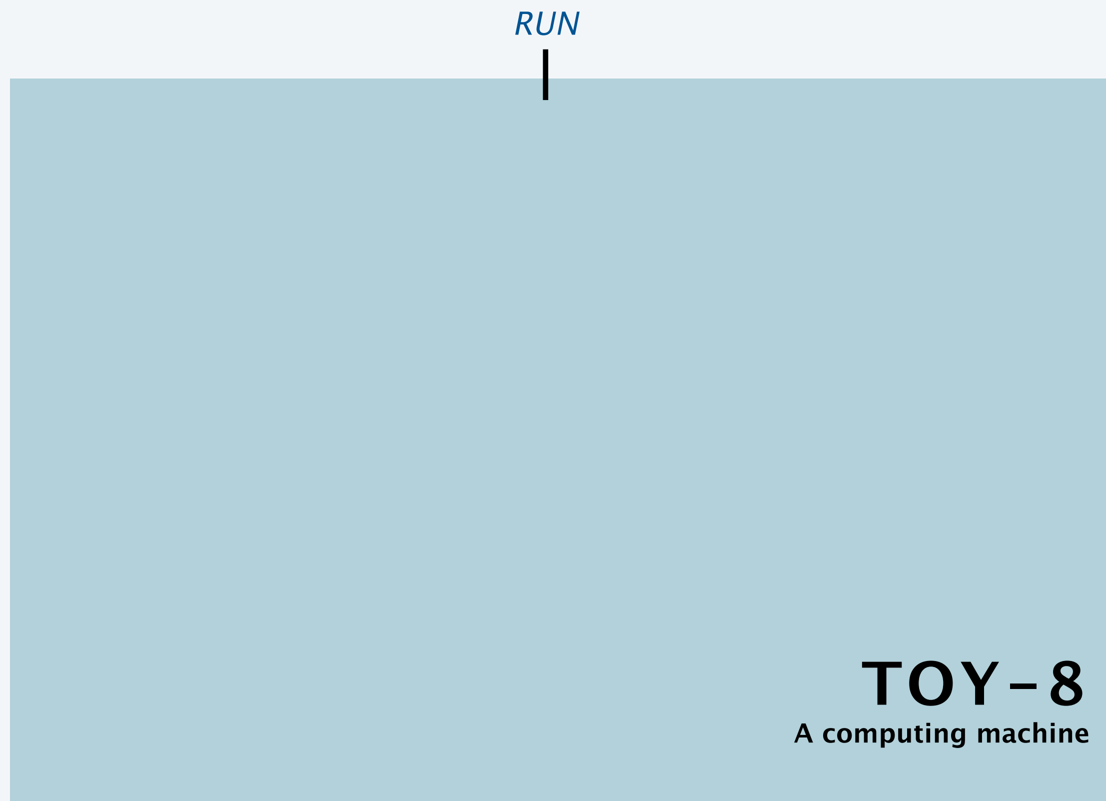


Interface to outside world

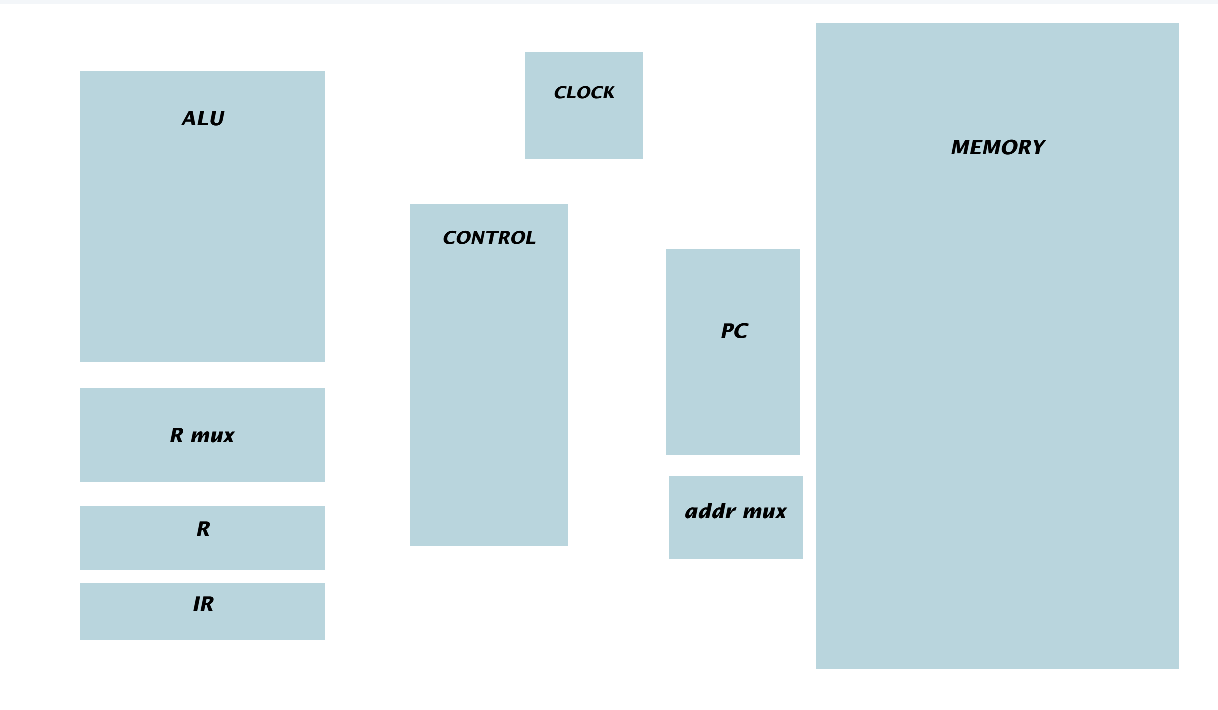
- Switches and lights
- ON/OFF
- RUN

Connections to outside (omitted)

- ADDR to PC
- DATA to memory bank input bus
- Buttons to control lines that activate memory load/store

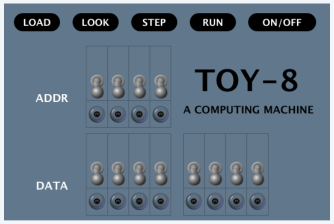


Review: CPU circuit components for TOY-8



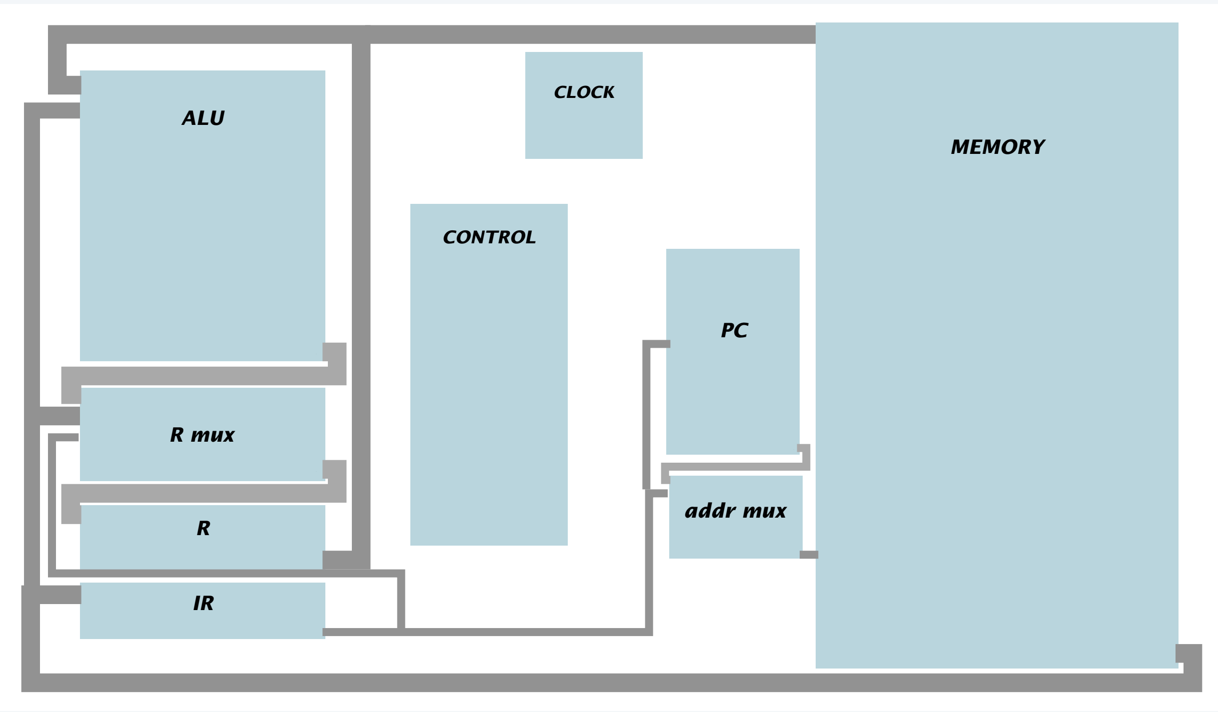
TOY-8 CPU

- ALU (adder, AND, XOR)
- Memory
- Register (R)
- PC
- IR
- Control
- Clock



Also needed (see next slide): Bus muxes for R and memory addr.

Connections for TOY-8 CPU

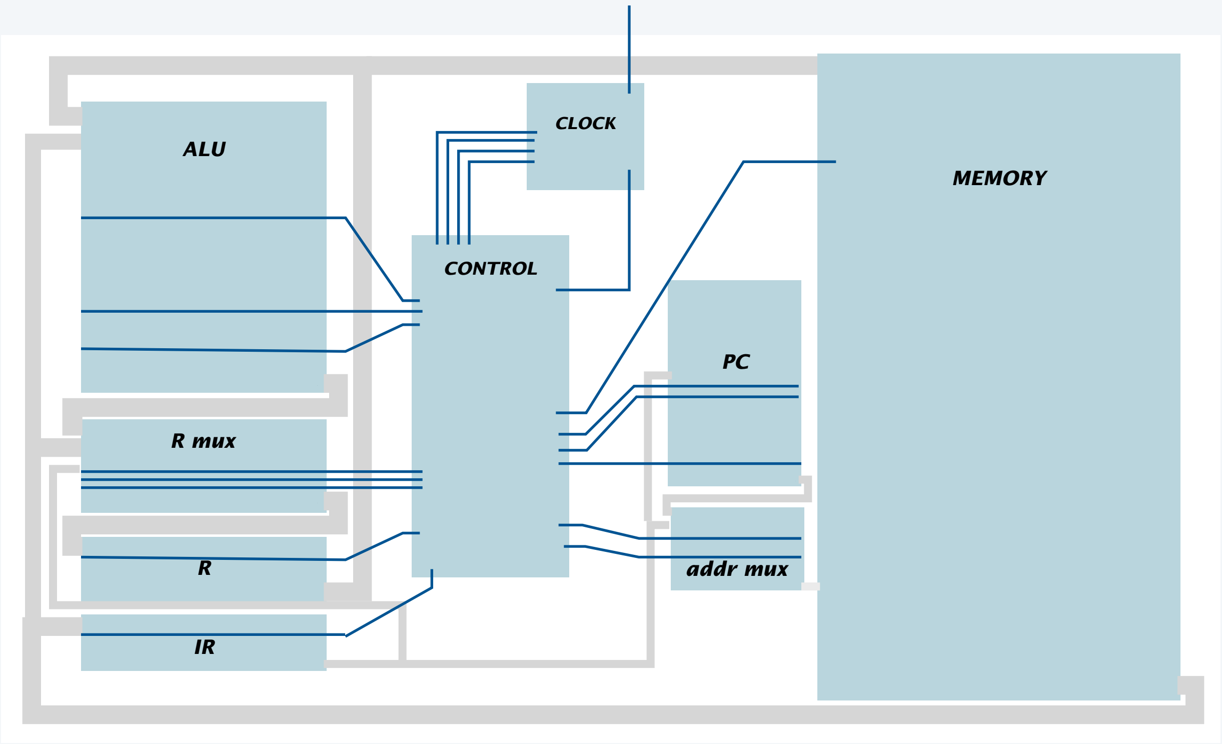


instructions	bus connections
fetch (all)	PC to memory addr
	memory to IR
halt	none
add, and, xor	IR addr to memory addr
	memory to ALU 1
	R to ALU 0
	ALU to R
load address	IR addr to R
load	IR addr to memory addr
	memory to R
store	IR addr to memory addr
	R to memory
branch if zero	IR addr to PC

need 2-way bus mux

need 3-way bus mux

Control wires for TOY-8 CPU



component	control wires
CLOCK	RUN
	HALT
CONTROL	FETCH
	FETCH WRITE
	EXECUTE
	EXECUTE WRITE
ALU	ADD
	XOR
	AND
R mux	R MUX ALU
	R MUX MEM
	R MUX IR
R	R WRITE
IR	IR WRITE
memory	MEMORY WRITE
PC	PC INCREMENT
	PC LOAD
	PC WRITE
addr mux	ADDR MUX PC
	ADDR MUX IR

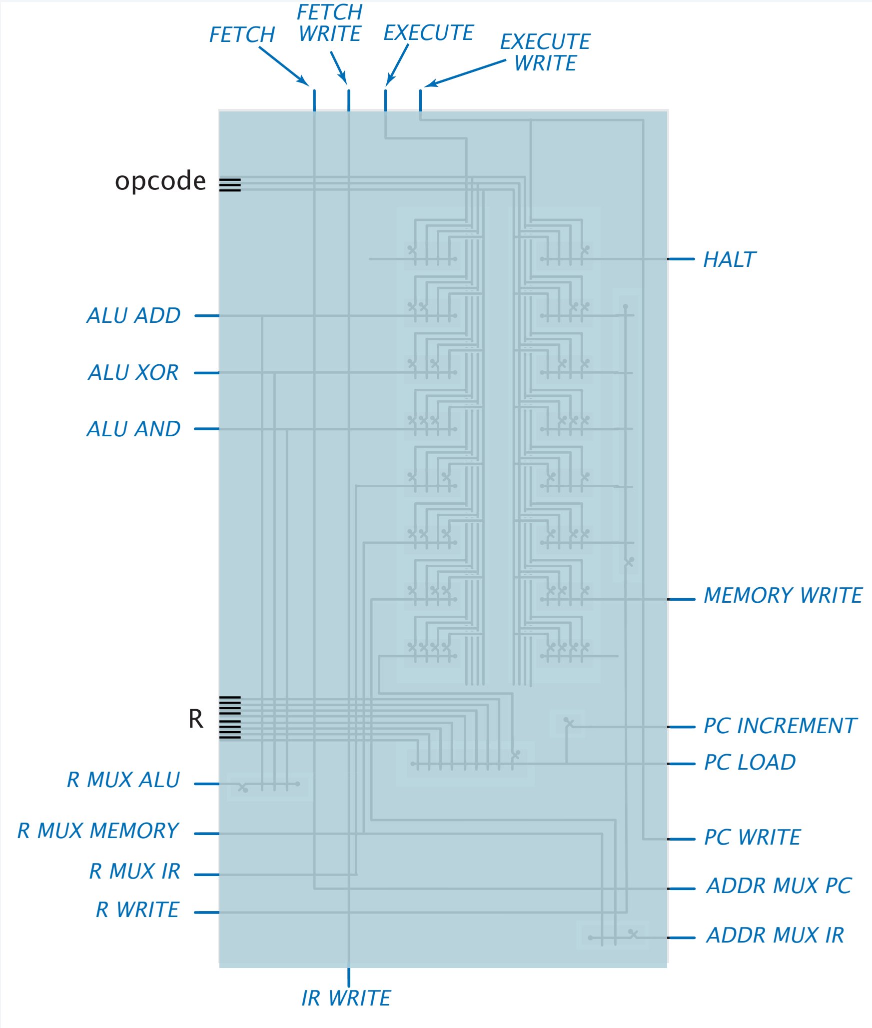
One final combinational circuit: Control

Control. Circuit for *control wire sequencing*.

Inputs
- Four control wires from clock.
- opcode from IR.
- contents of R.

Outputs
- 15 control wires for CPU components.

Key feature. A simple combinational circuit.

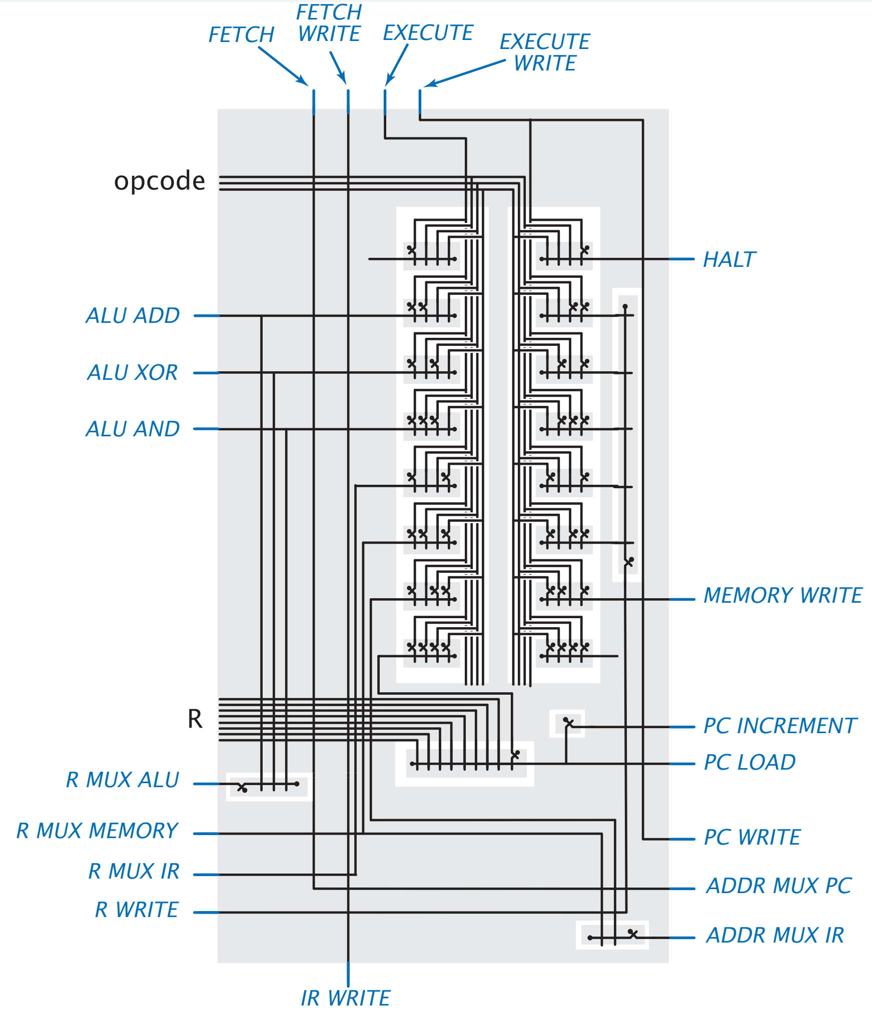


Control wire sequences

	FETCH	FETCH WRITE	EXECUTE WRITE
all instructions	ADDR MUX PC	IR WRITE	PC WRITE
	PC INCREMENT*		

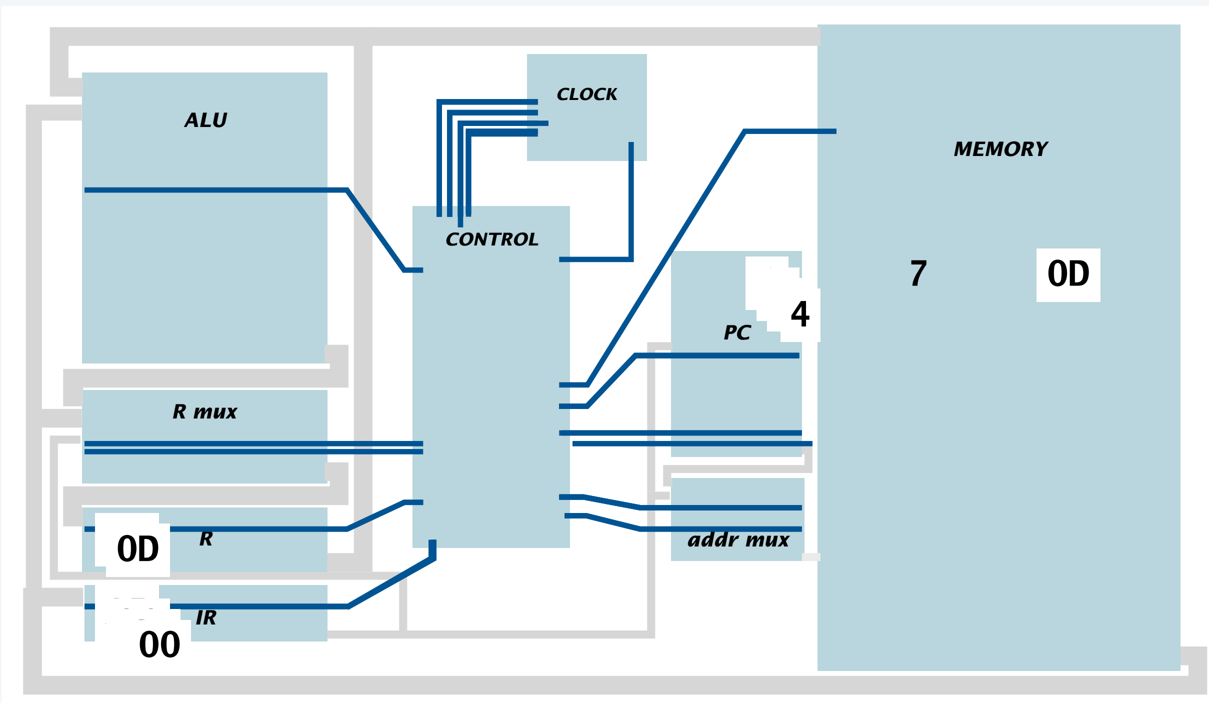
* PC LOAD for branch if 0 if R is 0.

instruction	EXECUTE	EXECUTE WRITE
halt	HALT	
add	ALU ADD	R WRITE
	R MUX ALU	
xor	ALU ADD	R WRITE
	R MUX ALU	
and	ALU ADD	R WRITE
	R MUX ALU	
load address	R MUX IR	R WRITE
load	R MUX MEMORY	R WRITE
	ADDR MUX IR	
store	ADDR MUX IR	MEMORY WRITE



Sample program

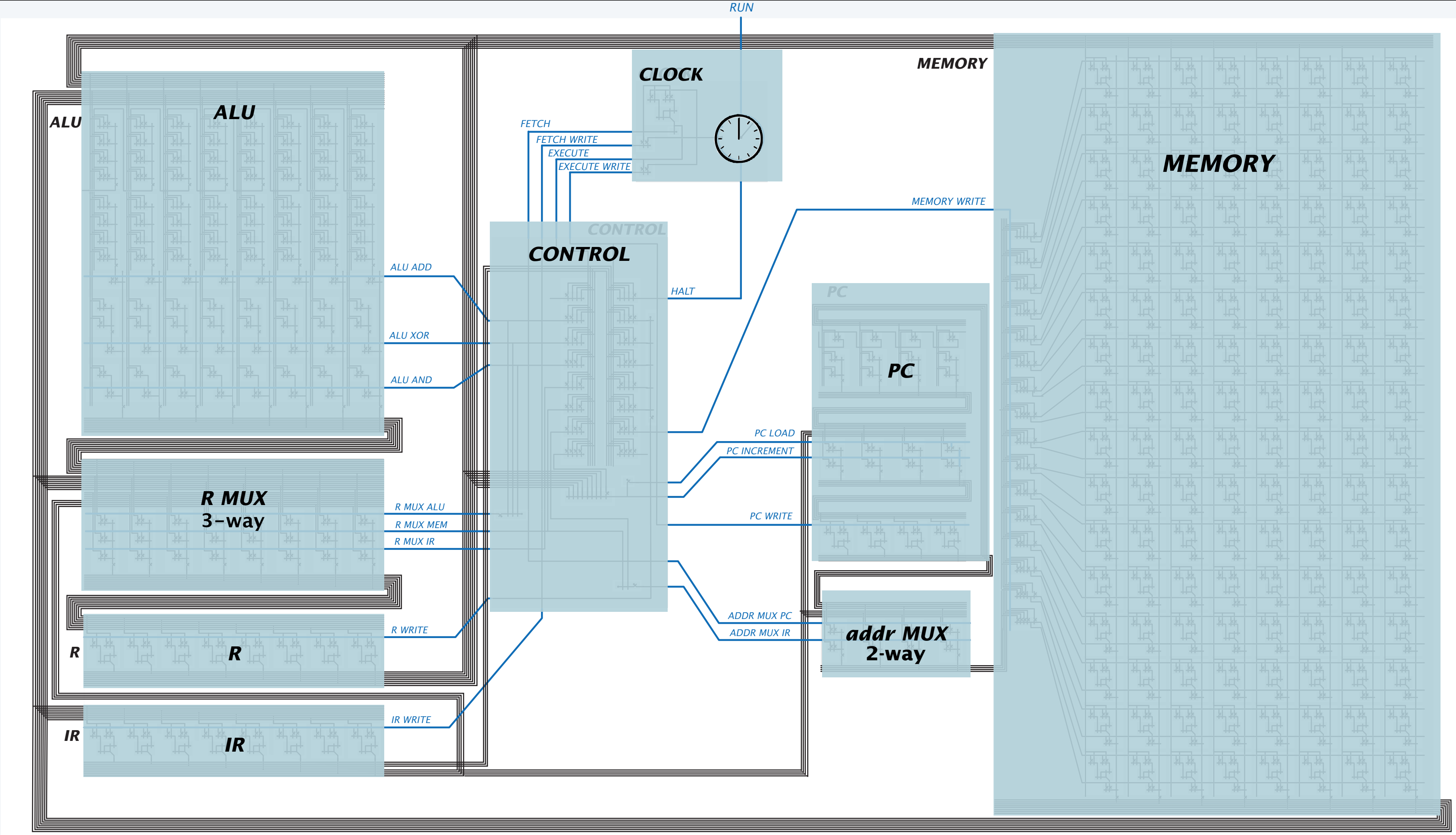
1	A5
2	26
3	C7
4	00
5	05
6	08



	control signal	result
FETCH	ADDR MUX PC	
FETCH WRITE	IR WRITE	IR = A5
EXECUTE	PC INCREMENT	
	ADDR MUX IR	
	R MUX MEMORY	
EXECUTE WRITE	R WRITE	R = 05
	PC WRITE	PC = 2
FETCH	ADDR MUX PC	
FETCH WRITE	IR WRITE	IR = 26
EXECUTE	PC INCREMENT	
	ALU ADD	
	R MUX ALU	
EXECUTE WRITE	R WRITE	R = 0D
	PC WRITE	PC = 3
FETCH	ADDR MUX PC	
FETCH WRITE	IR WRITE	IR = C7
EXECUTE	PC INCREMENT	
	ADDR MUX IR	
EXECUTE WRITE	MEMORY WRITE	M[7] = 0D
	PC WRITE	PC = 4
FETCH	ADDR MUX PC	
FETCH WRITE	IR WRITE	IR = 00
EXECUTE	PC INCREMENT	
	HALT	

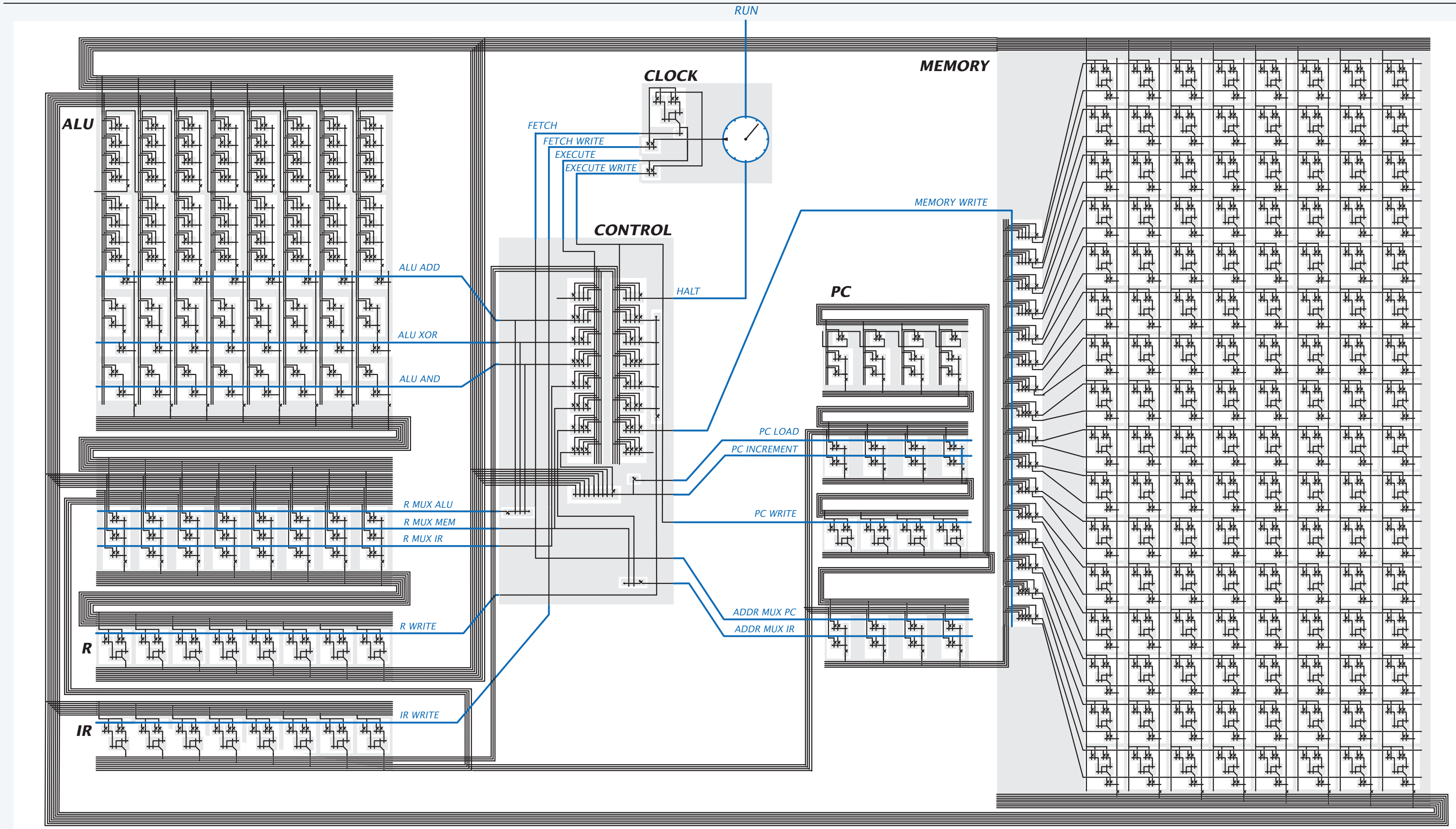
And THAT . . . is how your computer works!

TOY-8 CPU

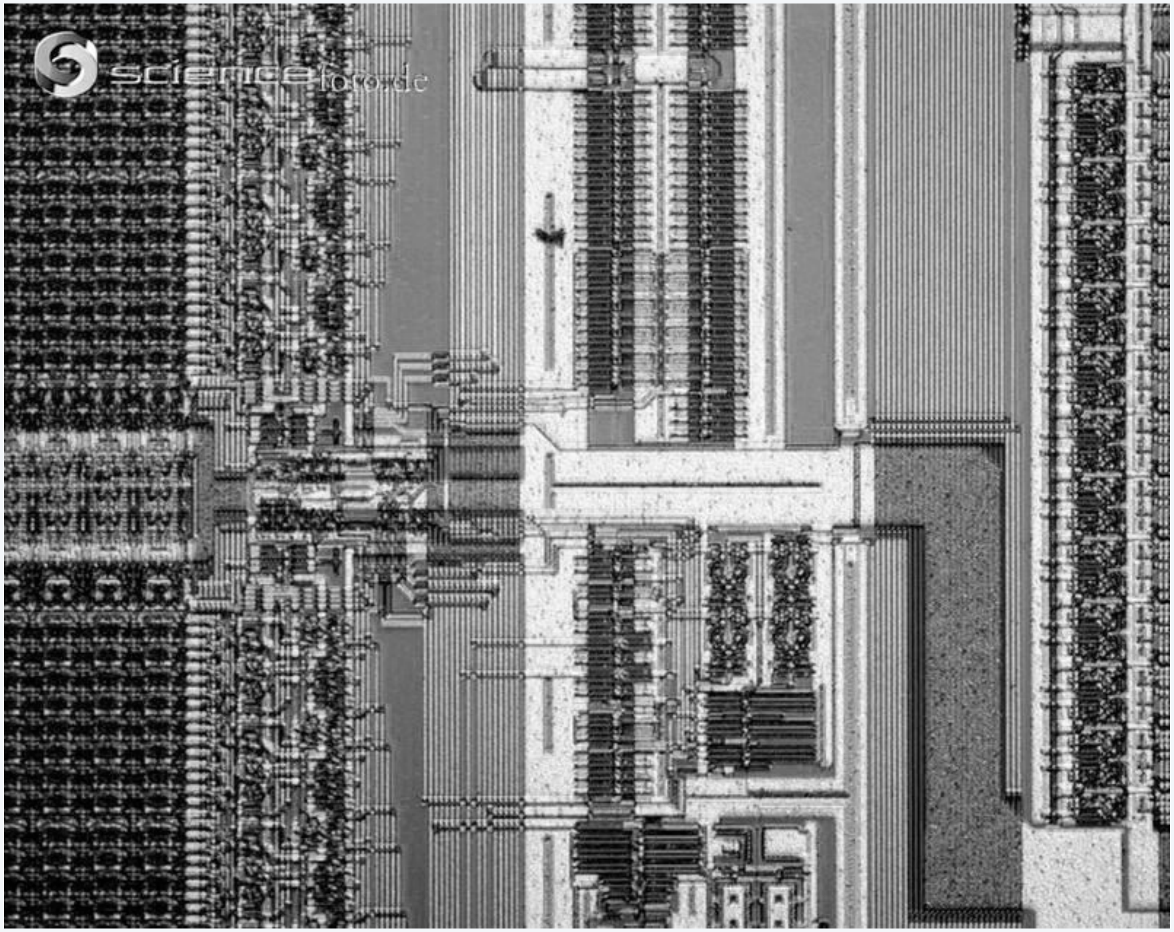


RUN
CLOCK
MEMORY
ALU
ALU
FETCH
FETCH WRITE
EXECUTE
EXECUTE WRITE
MEMORY
MEMORY WRITE
CONTROL
CONTROL
ALU ADD
HALT
PC
ALU XOR
PC
ALU AND
PC LOAD
PC INCREMENT
R MUX
3-way
R MUX ALU
R MUX MEM
R MUX IR
PC WRITE
ADDR MUX PC
ADDR MUX IR
R WRITE
addr MUX
2-way
R
R
IR WRITE
IR
IR

TOY-8 CPU


RUN
CLOCK
MEMORY
ALU
FETCH
FETCH WRITE
EXECUTE
EXECUTE WRITE
MEMORY WRITE
CONTROL
ALU ADD
HALT
PC
ALU XOR
ALU AND
PC LOAD
PC INCREMENT
R MUX ALU
R MUX MEM
R MUX IR
PC WRITE
ADDR MUX PC
ADDR MUX IR
R WRITE
R
IR WRITE
IR

Scanning electron microscope image of a real microprocessor

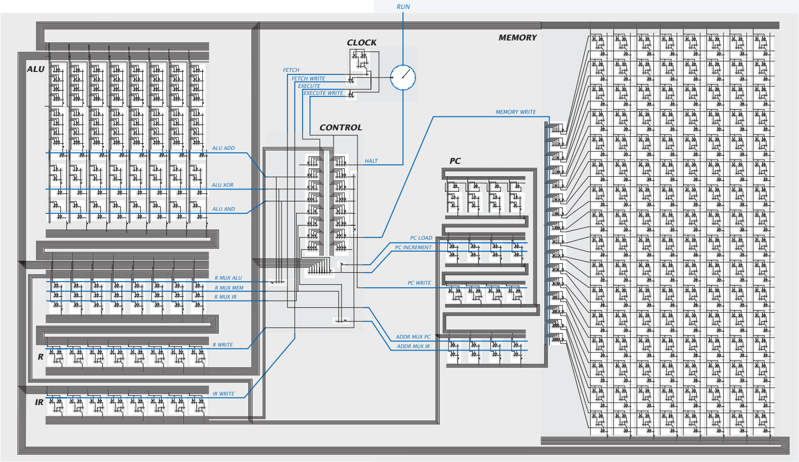


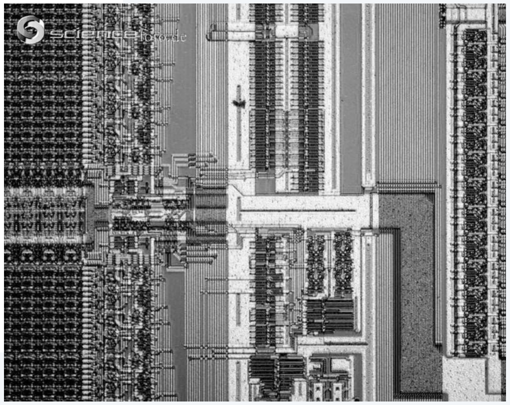
Memory bits per square cm	
modern microprocessor	25 billion
TOY-8	1

How does your computer work?

A not-so-short answer, in case someone asks...

- A circuit known as the *CPU* is built from *switches* connected by wires.
- The CPU performs operations on information encoded in binary, including its own instructions.
- Circuits with feedback implement *memories*.
- Instructions move information among memories, specify the next operation, or implement mathematical functions based on *Boolean logic*.
- Clock pulses activate sequences of *control signals*, which cause state changes that implement machine instructions.
- Virtually everything else is implemented as *layers of software*, each layer adding additional power and scope.



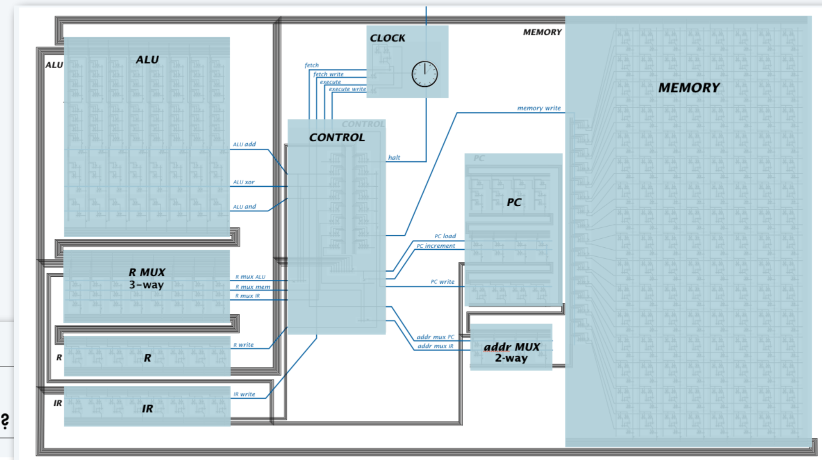


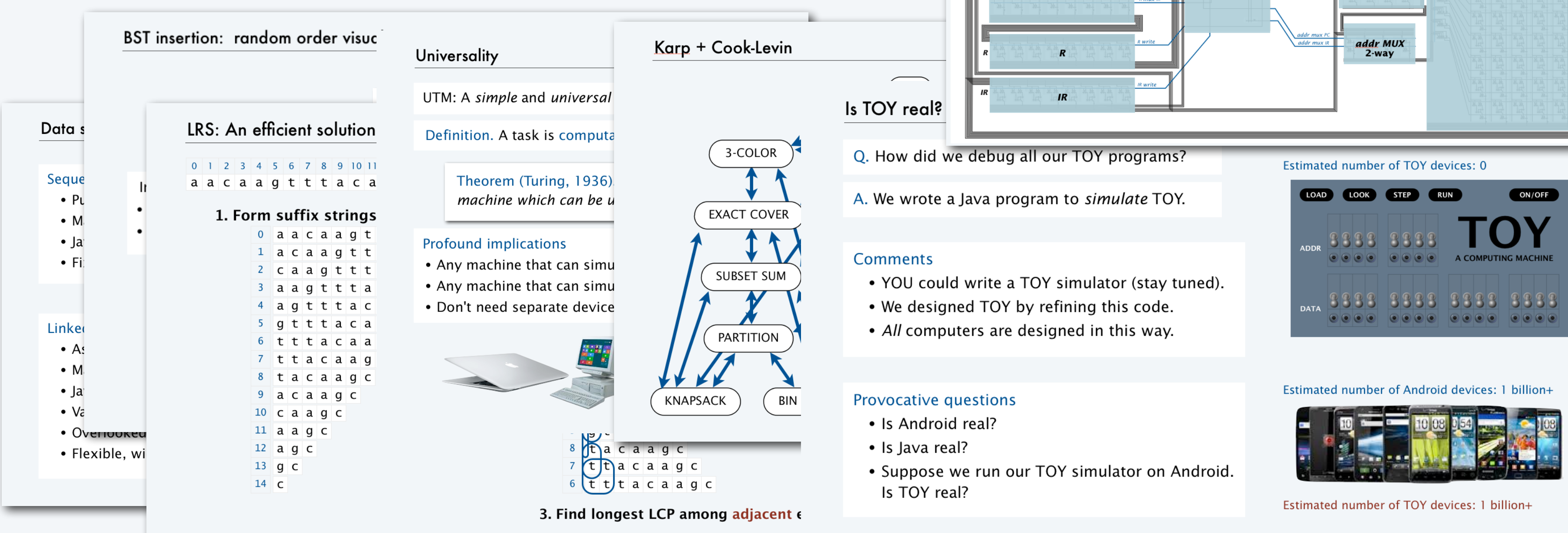
What is this course about?

A broad introduction to computer science.

Goals

- Empower you to exploit available technology. ✓
- Build awareness of intellectual underpinnings. ✓
- Demystify computer systems. ✓

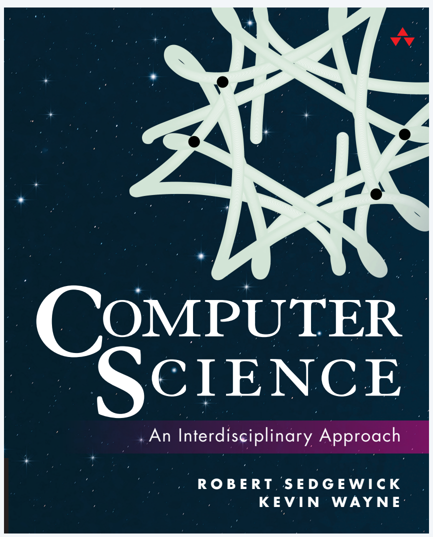




What is this course about?

A broad introduction to computer science.

Goals
- Empower you to exploit available technology. ✓
- Build awareness of intellectual underpinnings. ✓
- Demystify computer systems. ✓



Next: Algorithms.

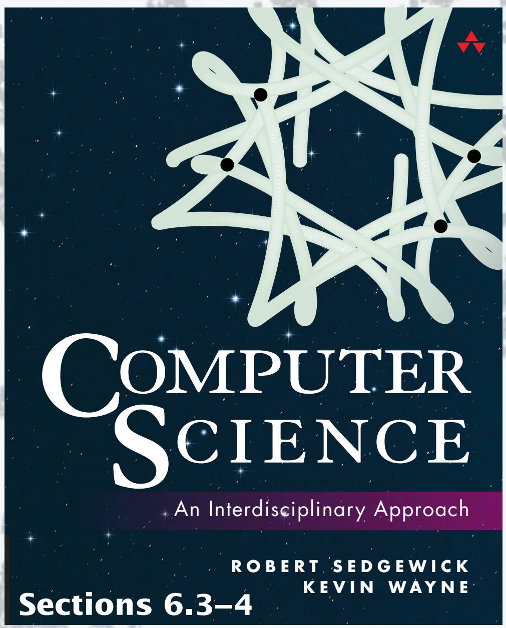
Image sources

http://download.intel.com/pressroom/images/corefamily/Westmere4.jpg

http://www.sciencefoto.de/detail.php?rubrik=Nano&id=214956&lang=en&q=&qrubrik=

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

20. Central Processing Unit