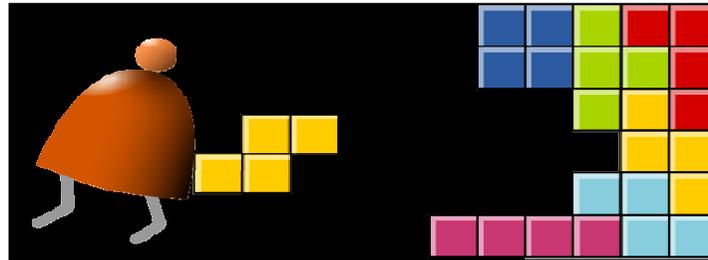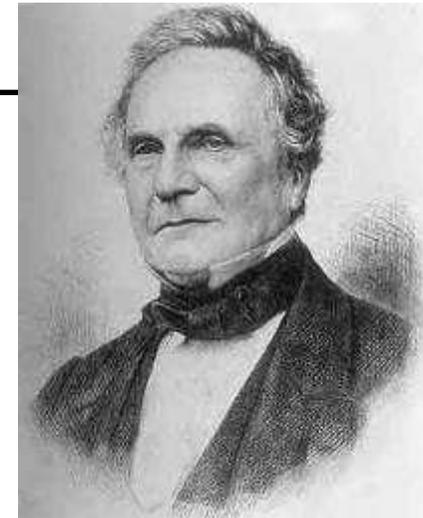# Computer Architecture

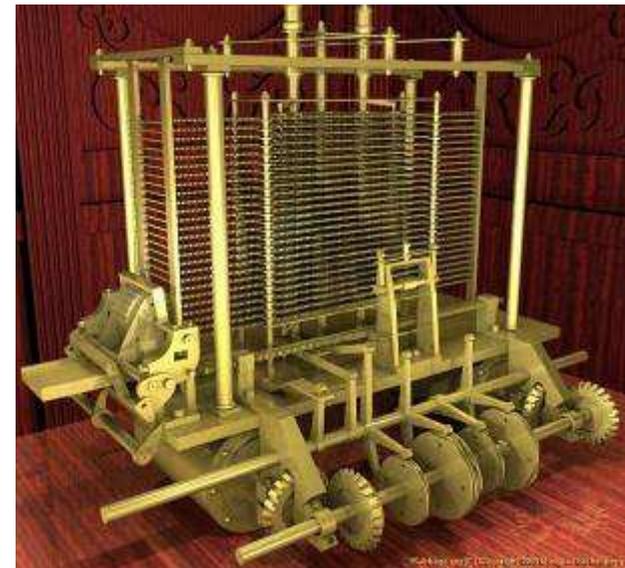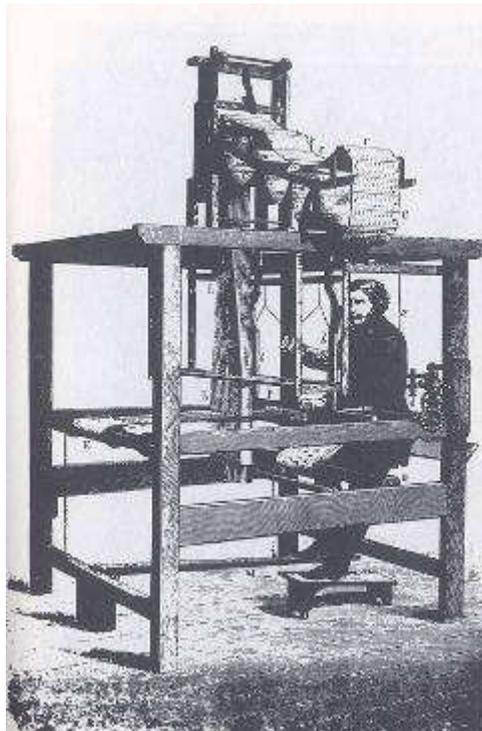*Building a Modern Computer From First Principles*

www.nand2tetris.org

Babbage's Analytical Engine (1835)

"We may say most aptly that the Analytical Engine
weaves algebraic patterns just as the Jacquard-
loom weaves flowers and leaves"
(Ada Lovelace)

Charles Babbage (1791-1871)

Some early computers and computer scientists

Blaise Pascal
1623-1662
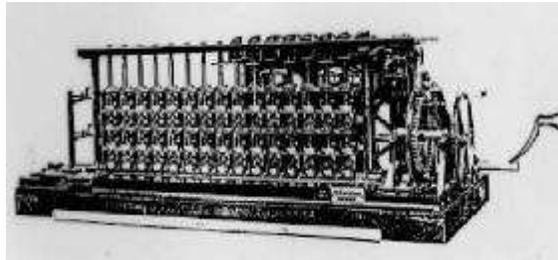
Gottfried Leibniz
1646-1716
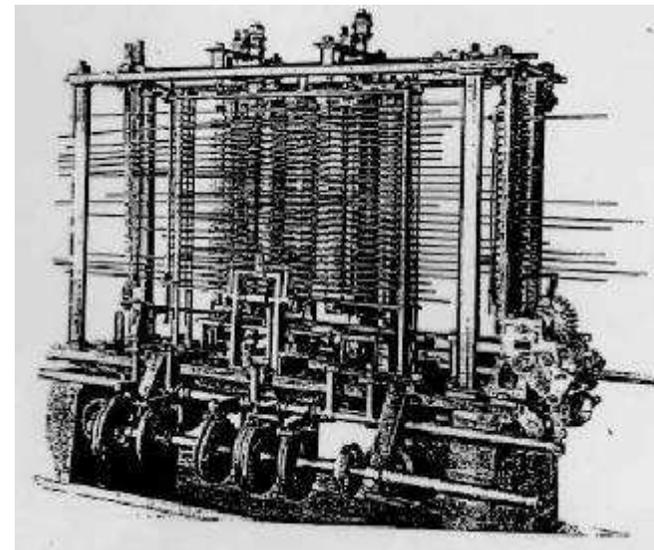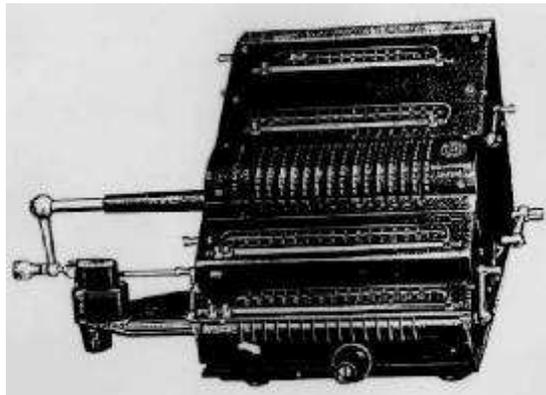
Von Neumann machine (circa 1940)

Memory (data + instructions) ↔ CPU [Arithmetic Logic Unit (ALU), Registers, Control]; Input device → CPU; Control → Output device

Stored program concept!
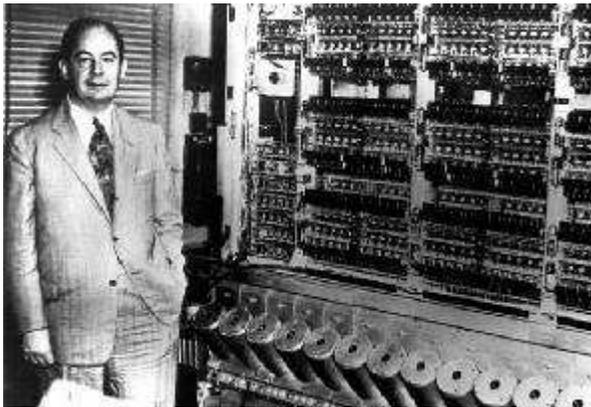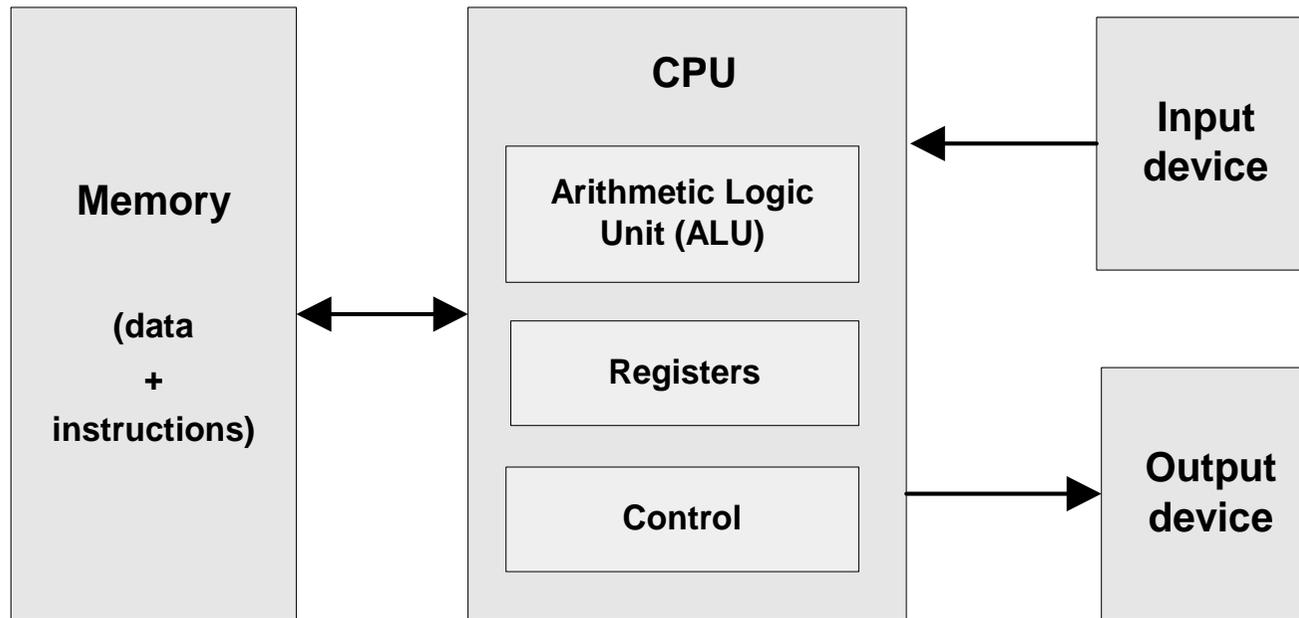
John Von Neumann (and others) ... made it possible

Andy Grove (and others) ... made it small and fast.

Processing logic: fetch-execute cycle
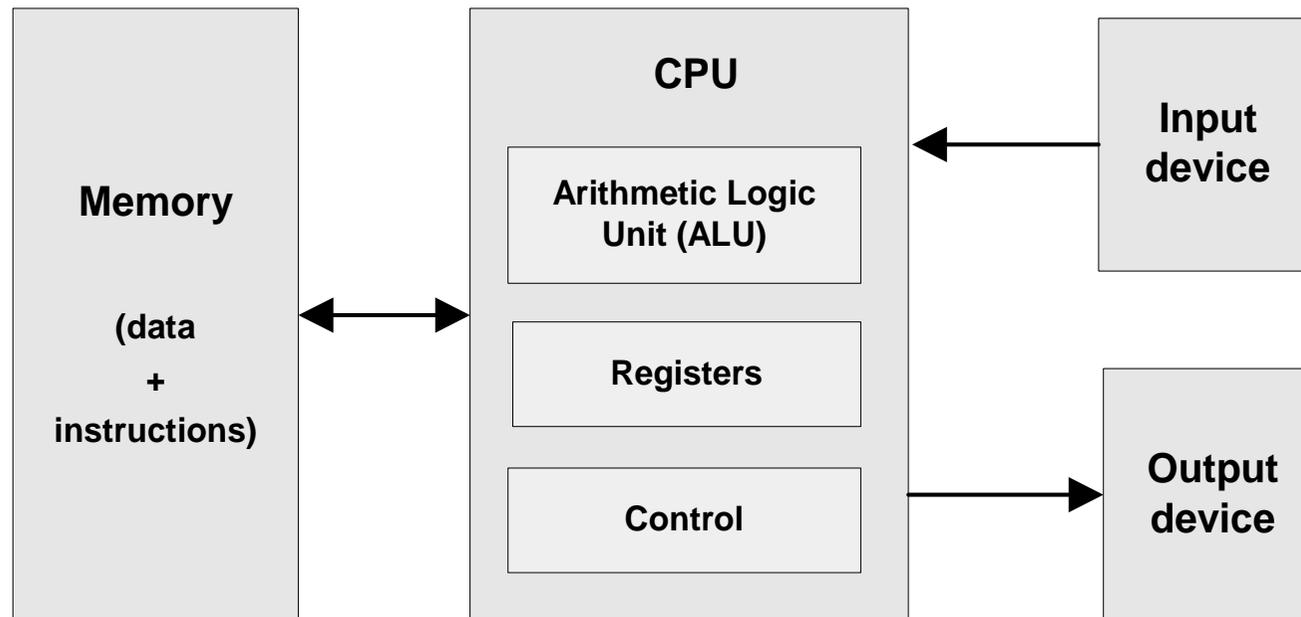
Executing the *current instruction* involves one or more of
   the following micro-tasks:

- ❑ Have the ALU compute some function $out = f\,(register\ values)$

- ❑ Write the ALU output to selected registers

- ❑ As a side-effect of this computation,
  figure out which instruction to fetch and execute next.

The Hack chip-set and hardware platform

Elementary logic gates

- Nand
- Not
- And
- Or
- Xor
- Mux
- Dmux
- Not16
- And16
- Or16
- Mux16
- Or8Way
- Mux4Way16
- Mux8Way16
- DMux4Way
- DMux8Way

done

Combinational chips

- HalfAdder
- FullAdder
- Add16
- Inc16
- ALU

done

Sequential chips

- DFF
- Bit
- Register
- RAM8
- RAM64
- RAM512
- RAM4K
- RAM16K
- PC

done

Computer Architecture

- Memory
- CPU
- Computer

this lecture

The Hack computer

- A 16-bit Von Neumann platform

- The *instruction memory* and the *data memory* are physically separate

- Screen: 512 rows by 256 columns, black and white

- Keyboard: standard

- Designed to execute programs written in the Hack machine language

- Can be easily built from the chip-set that we built so far in the course

Main parts of the Hack computer:

- Instruction memory (ROM)

- Memory (RAM):

  - Data memory

  - Screen (memory map)

  - Keyboard (memory map)

- CPU

- Computer (the logic that holds everything together).

Lecture / construction plan

- **Instruction memory**

- Memory:

    - Data memory

    - Screen

    - Keyboard

- CPU

- Computer

Instruction memory
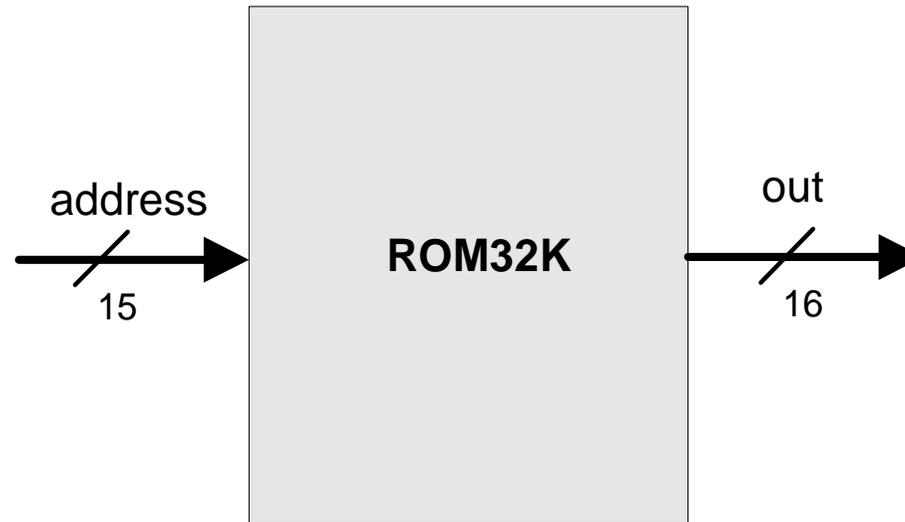
Function:

- The ROM is pre-loaded with a program written in the Hack machine language

- The ROM chip always emits a 16-bit number:

  `out = ROM32K[address]`

- This number is interpreted as the *current instruction*.

Data memory

Low-level (hardware) read/write logic:

To read RAM[k]: set address to k,
    probe out

To write RAM[k]=x: set address to k,
    set in to x,
    set load to 1,
    run the clock
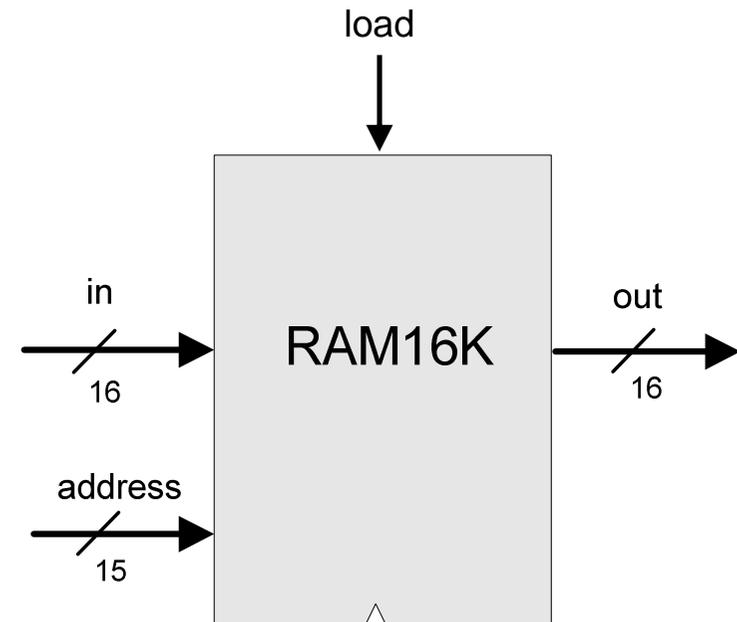
High-level (OS) read/write logic:

To read RAM[k]:   use the OS command out = peek(k)

To write RAM[k]=x: use the OS command poke(k,x)

peek and poke are OS commands whose implementation should effect the same
    behavior as the low-level commands

More about peek and poke this later in the course, when we'll write the OS.

Lecture / construction plan

✓ ■ **Instruction memory**

■ **Memory:**

✓ ❑ **Data memory**

⟹ ❑ **Screen**

❑ **Keyboard**

■ **CPU**

■ **Computer**

Screen

load

in

/
16

Screen

out

/
16

address

/
15

The bit contents of the
Screen chip is called the
"screen memory map"

Physical
Screen

Simulated screen:

The simulated
256 by 512
B&W screen
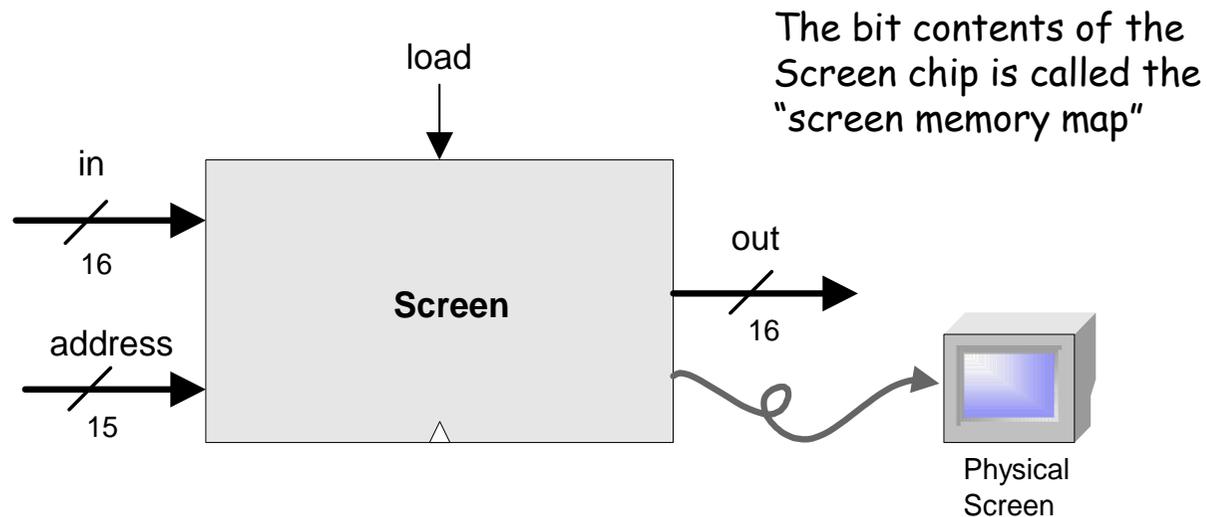
The Screen chip has a basic RAM chip functionality:

❑ read logic: `out = Screen[address]`

❑ write logic: `if load then Screen[address] = in`

Side effect:

Continuously refreshes a 256 by 512 black-and-white
screen device

When loaded into the
hardware simulator, the
built-in `Screen.hdl` chip
opens up a screen window;
the simulator then
refreshes this window
from the screen memory
map several times each
second.

Screen memory map

In the Hack platform, the screen is implemented as an 8K 16-bit RAM chip.

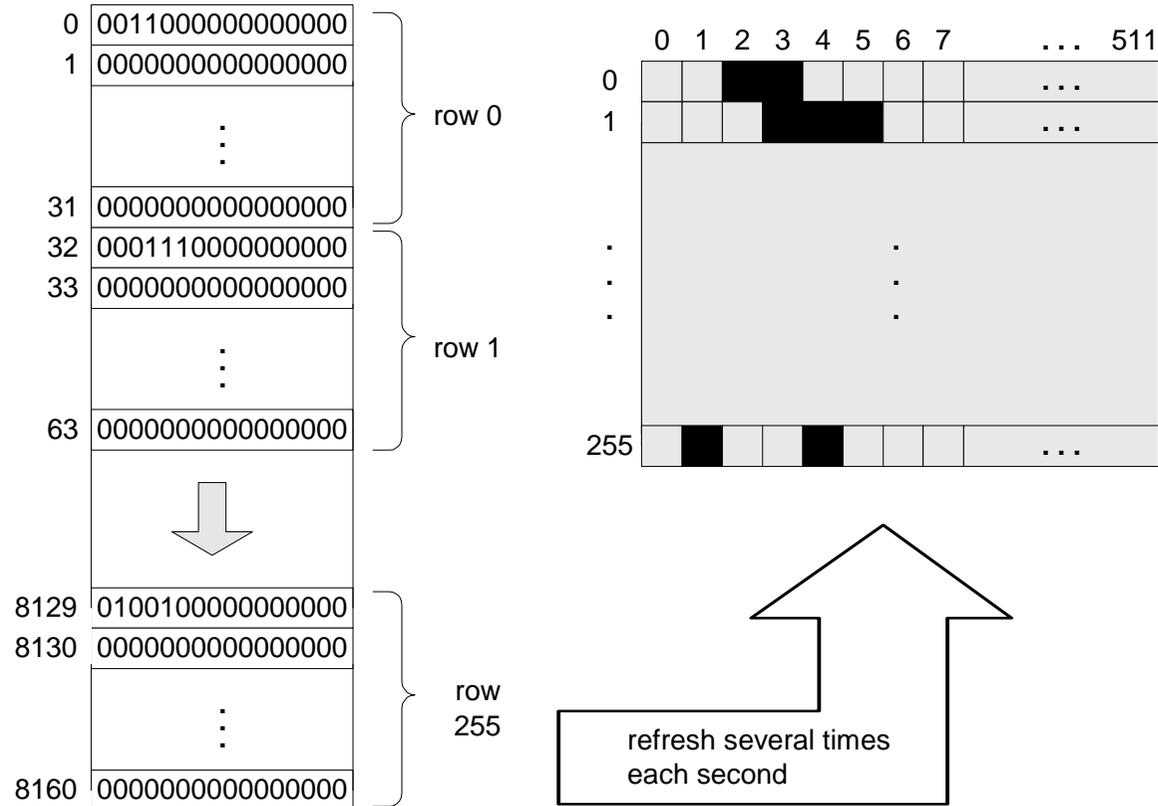|	
0	0011000000000000
1	0000000000000000
	.
31	0000000000000000
32	0001110000000000
33	0000000000000000
	.
63	0000000000000000

> row 0

> row 1

Screen

8129	0100100000000000
8130	0000000000000000
	.
8160	0000000000000000

> row 255

refresh several times each second
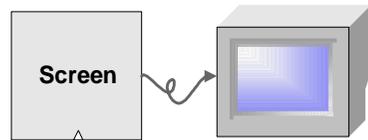
How to set the (row,col) pixel of the screen to black or to white:

❑ Low-level (machine language):   Set the `col%16` bit of the word found at `Screen[row*32+col/16]` to `1` or to `0` (`col/16` is integer division)

❑ High-level: Use the OS command `drawPixel(row,col)` (effects the same operation, discussed later in the course, when we'll write the OS).

Keyboard

Keyboard

out

16
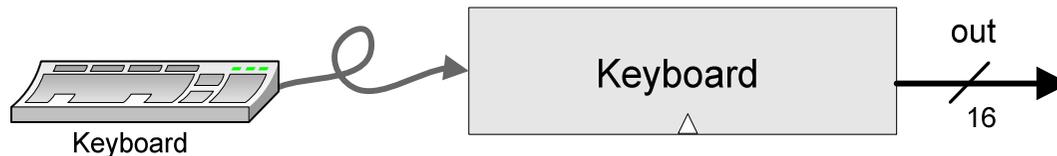
Keyboard chip: a single 16-bit register

Input: scan-code (16-bit value) of the currently
pressed key, or 0 if no key is pressed

Output: same

Special keys:

Key pressed	Keyboard output
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134

Key pressed	Keyboard output
end	135
page up	136
page down	137
insert	138
delete	139
esc	140
f1-f12	141-152

Simulated keyboard:

The simulated keyboard enabler button

The keyboard is implemented as a built-in `Keyboard.hdl` chip. When this java chip is loaded into the simulator, it connects to the regular keyboard and pipes the scan-code of the currently pressed key to the keyboard memory map.

How to read the keyboard:

❑ Low-level (hardware): probe the contents of the `Keyboard` chip

❑ High-level: use the OS command `keyPressed()`
(effects the same operation, discussed later in the course, when we'll write the OS).

Lecture / construction plan

✓ ■ Instruction memory

➡ ■ Memory:

✓ ❑ Data memory

✓ ❑ Screen

✓ ❑ Keyboard

■ CPU

■ Computer

Memory: conceptual / programmer's view
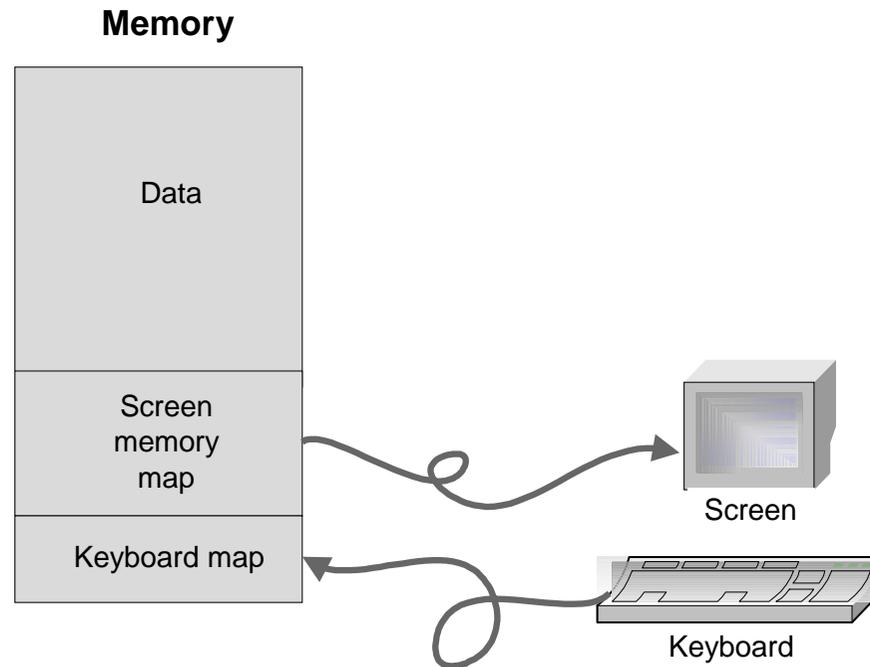
Memory

Using the memory:

❑ To record or recall values (e.g. variables, objects, arrays), use the first 16K words of the memory

❑ To write to the screen (or read the screen), use the next 8K words of the memory

❑ To read which key is currently pressed, use the next word of the memory.

Memory: physical implementation

load

Memory

0

in

16

RAM16K

(16K mem. chip)

out

16

16383

16384

address

Screen

(8K mem. chip)

15

24575

24576

Keyboard

(one register)
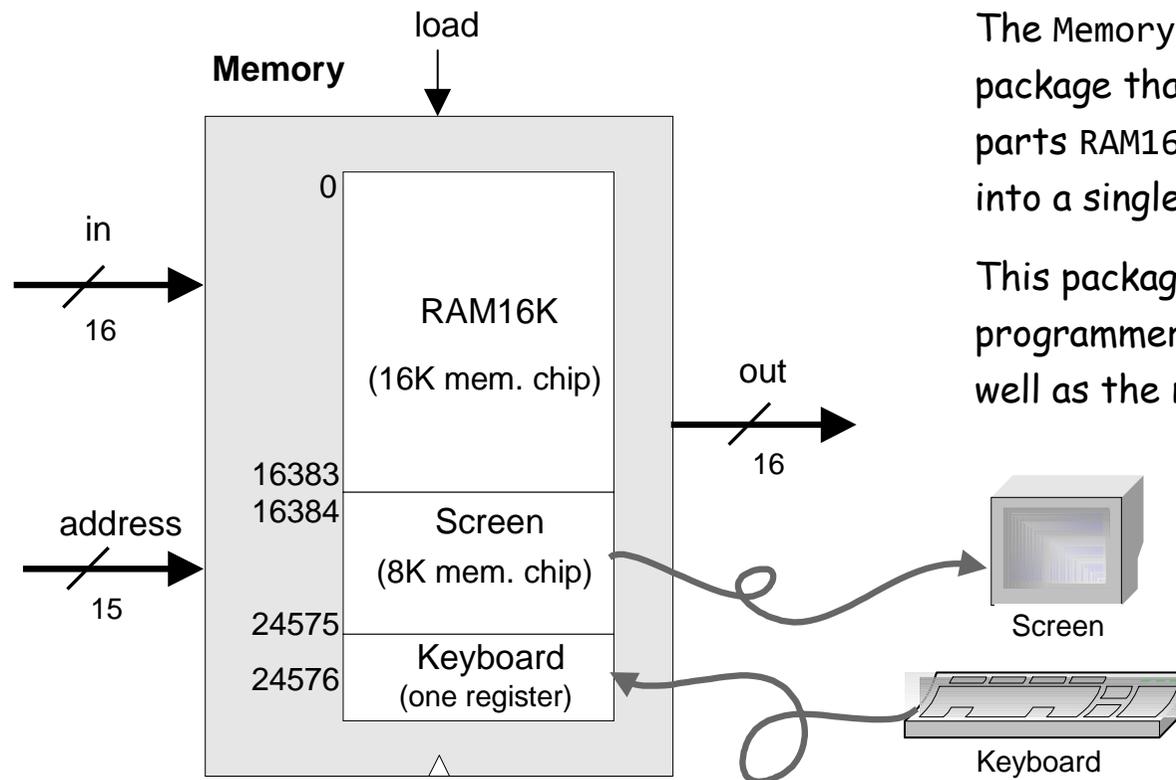
Screen

Keyboard

The Memory chip is essentially a package that integrates the three chip-parts RAM16K, Screen, and Keyboard into a single, contiguous address space.

This packaging effects the programmer's view of the memory, as well as the necessary I/O side-effects.

Access logic:

❑ Access to any address from 0 to 16,383 results in accessing the RAM16K chip-part

❑ Access to any address from 16,384 to 24,575 results in accessing the Screen chip-part

❑ Access to address 24,576 results in accessing the keyboard chip-part

❑ Access to any other address is invalid.

Lecture / construction plan

✓ ■ **Instruction memory**

✓ ■ **Memory:**

    ✓ ❑ **Data memory**

    ✓ ❑ **Screen**

    ✓ ❑ **Keyboard**

➡ ■ **CPU**

■ **Computer**

A pledge to patience ...

"At times ... the fragments that I lay out for your inspection may seem not to fit well together, as if they were stray pieces from separate puzzles. In such cases, I would counsel patience. There are moments when a large enough fragment can become a low wall, a second fragment another wall to be raised at a right angle to the first. A few struts and beams later, and we may made ourselves a rough foundation ... But it can consume the better part of a chapter to build such a foundation; and as we do so the fragment that we are examining may seem unconnected to the larger whole. Only when we step back can we see that we have been assembling something that can stand in the wind."

From: Sailing the Wind Dark Sea (Thomas Cahill)

CPU

from
data memory { inM → 16

from
instruction
memory { instruction → 16

a Hack machine language
instruction like M=D+M,
stated as a 16-bit value

reset → 1

CPU

→ outM
16

→ writeM
1

→ addressM
15

} to data
memory

→ pc
15
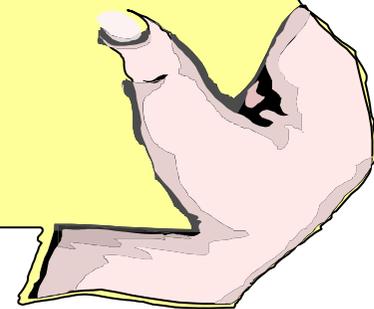
} to instruction
memory
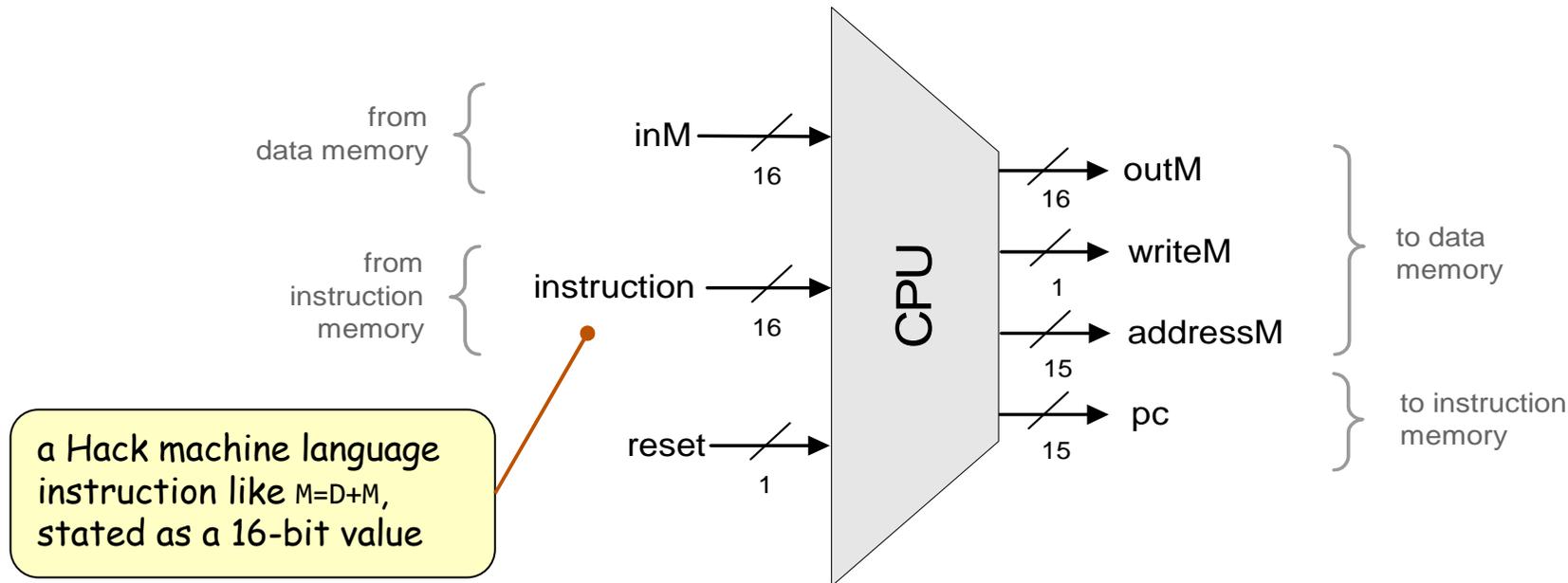
CPU internal components (invisible in this chip diagram): ALU and 3 registers: A, D, PC

CPU execute logic:

The CPU executes the instruction according to the Hack language specification:

❑ The D and A values, if they appear in the instruction, are read from (or written to) the respective CPU-resident registers

❑ The M value, if there is one in the instruction's RHS, is read from inM

❑ If the instruction's LHS includes M, then the ALU output is placed in outM, the value of the CPU-resident A register is placed in addressM, and writeM is asserted.

CPU

from data memory

from instruction memory

a Hack machine language instruction like M=D+M, stated as a 16-bit value

inM 16

instruction 16

reset 1

CPU

outM 16

writeM 1
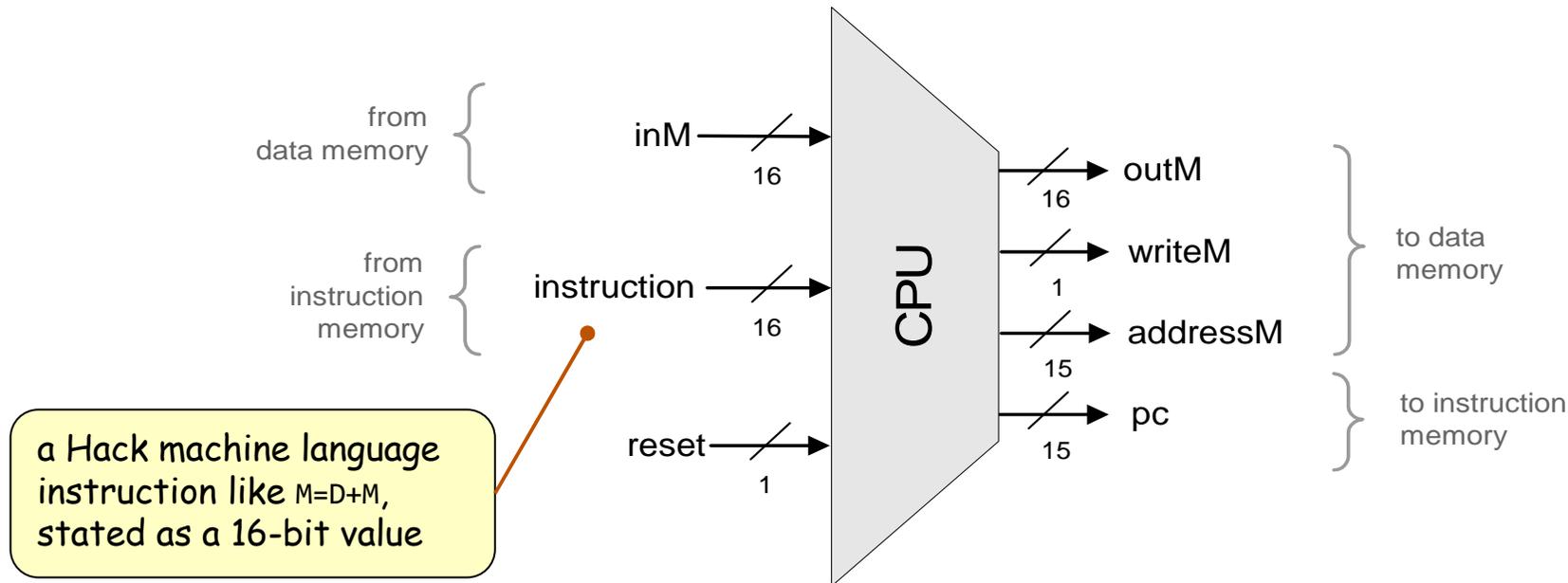
addressM 15

pc 15

to data memory

to instruction memory

CPU internal components (invisible in this chip diagram): ALU and 3 registers: A, D, PC

CPU fetch logic:

Recall that:

1. the instruction may include a jump directive (expressed as non-zero jump bits)
2. the ALU emits two control bits, indicating if the ALU output is zero or less than zero

If reset==0: the CPU uses this information (the jump bits and the ALU control bits) as follows:

If there should be a jump, the PC is set to the value of A; else, PC is set to PC+1

If reset==1: the PC is set to 0. (restarting the computer)

The *C*-instruction revisited

dest = comp; jump	*comp*				*dest*		*jump*		
binary: **1 1 1** a	c1 c2 c3 c4				c5 c6 d1 d2		d3 j1 j2 j3		

(when a=0) comp	c1	c2	c3	c4	c5	c6	(when a=1) comp
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

d1	d2	d3	Mnemonic	Destination (where to store the computed value)
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

CPU implementation
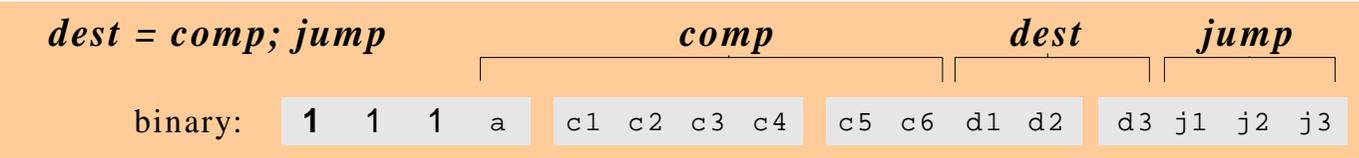
Chip diagram:

- Includes most of the CPU's execution logic

- The CPU's control logic is hinted: each circled "c" represents one or more control bits, taken from the instruction

- The "decode" bar does not represent a chip, but rather indicates that the instruction bits are decoded somehow.
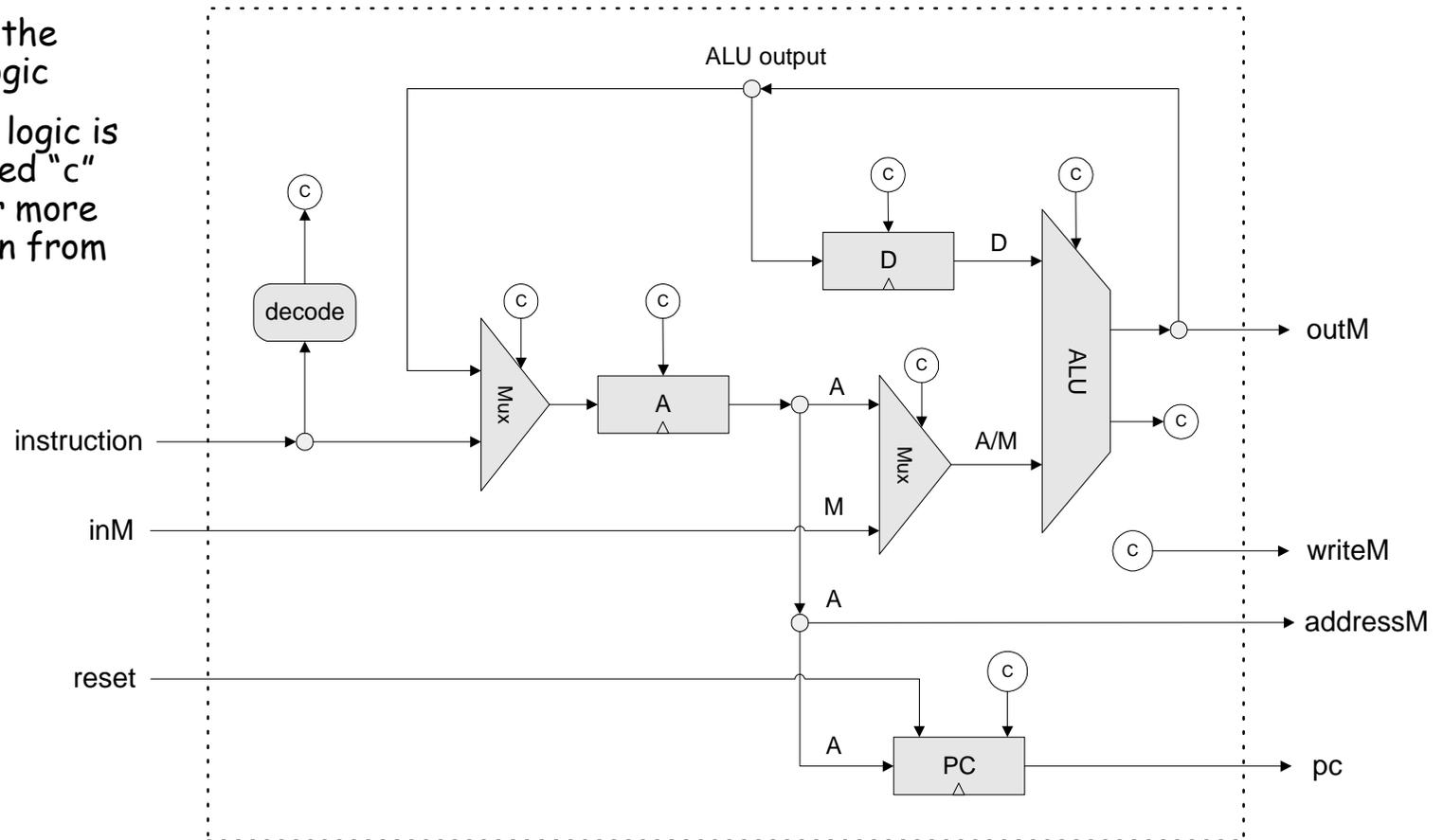
Cycle:

- Execute
- Fetch

Execute logic:

- Decode
- Execute

Fetch logic:

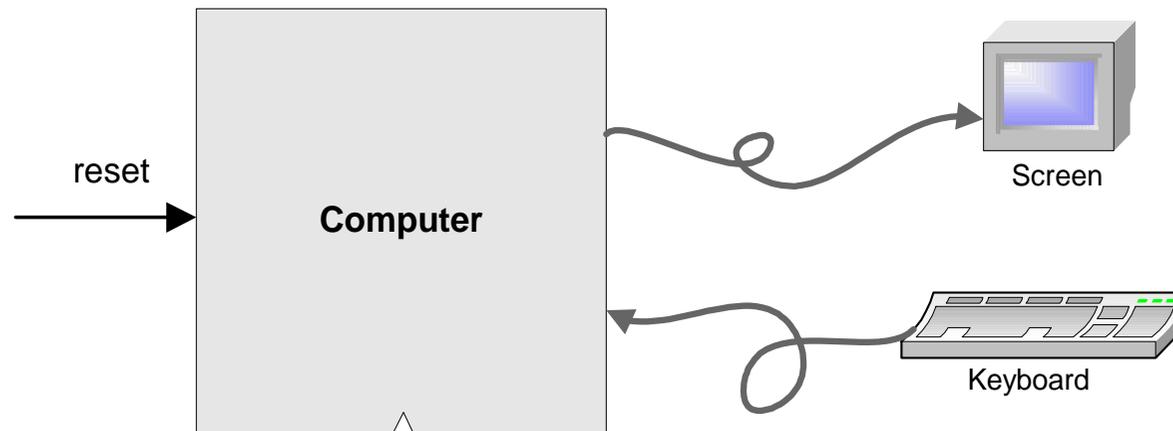If there should be a jump,
  set PC to A
else set PC to PC+1

Resetting the computer:

Set reset to 1,
then set it to 0.

Lecture / construction plan

✓ ■ **Instruction memory**

✓ ■ **Memory:**

       ❑ Data memory

       ❑ Screen

       ❑ Keyboard

✓ ■ CPU
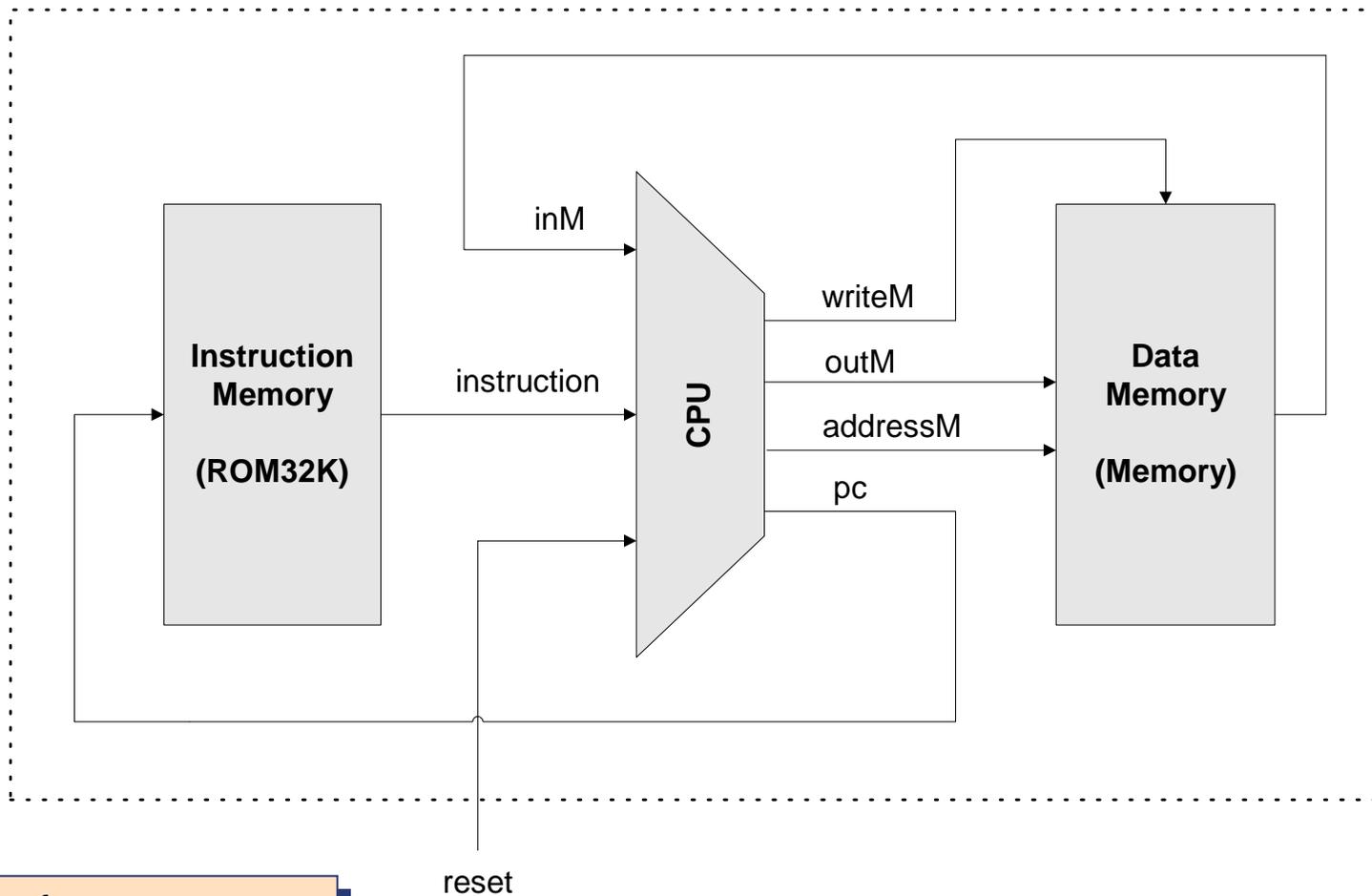
➡ ■ Computer

Computer-on-a-chip interface


```
Chip Name:  Computer  // Topmost chip in the Hack platform
Input:      reset
Function:   When reset is 0, the program stored in the
            computer's ROM executes. When reset is 1, the
            execution of the program restarts. Thus, to start a
            program's execution, reset must be pushed "up" (1)
            and "down" (0).

            From this point onward the user is at the mercy of
            the software. In particular, depending on the
            program's code, the screen may show some output and
            the user may be able to interact with the computer
            via the keyboard.
```

Computer-on-a-chip implementation


```
CHIP Computer {
    IN reset;
    PARTS:
    // implementation missing
}
```
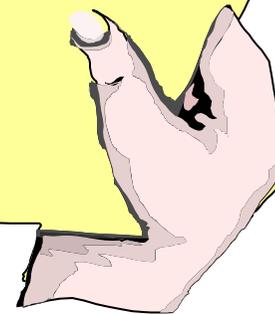
Implementation:
Simple, the chip-parts do all the hard work.

The spirit of things

We ascribe beauty to that which is
simple; which has no superfluous parts;
which exactly answers its end;
which stands related to all things;
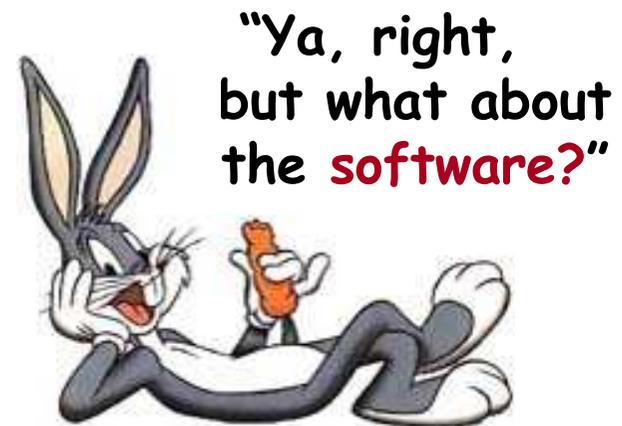which is the mean of many extremes.

(Ralph Waldo Emerson,
 1803-1882)

Lecture plan

✓ ■ Instruction memory

✓ ■ Memory:

   ❑ Data memory

   ❑ Screen

   ❑ Keyboard

✓ ■ CPU

✓ ■ Computer

"That's all folks!"

"Ya, right, but what about the **software?**"

Perspective: from here to a "real" computer

- Caching

- More I/O units

- Special-purpose processors (I/O, graphics, communications, …)

- Multi-core / parallelism

- Efficiency

- Energy consumption considerations

- And more …

Perspective: some issues we haven't discussed (among many)

- CISC / RISC (hardware / software trade-off)

- Hardware diversity: desktop, laptop, hand-held, game machines, …

- General-purpose vs. embedded computers

- Silicon compilers

- And more …