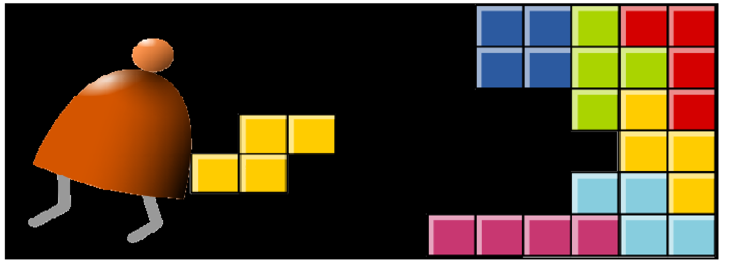
Computer Architecture

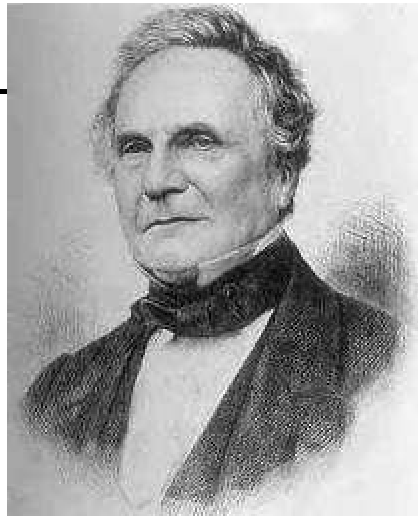


Building a Modern Computer From First Principles

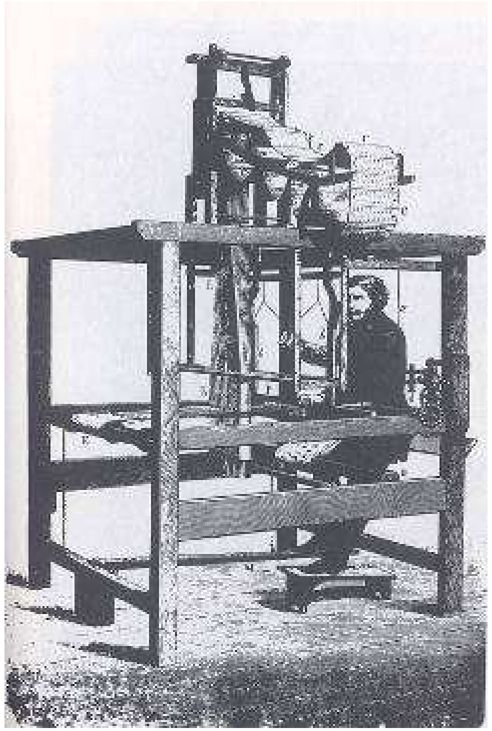
www.nand2tetris.org

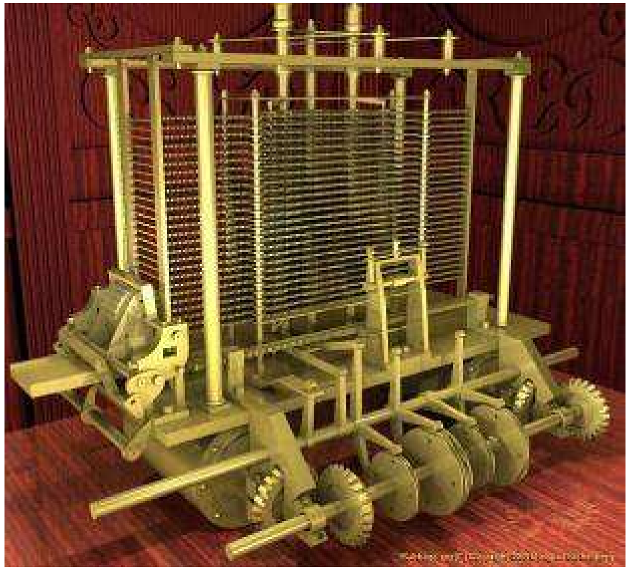
Babbage's Analytical Engine (1835)

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



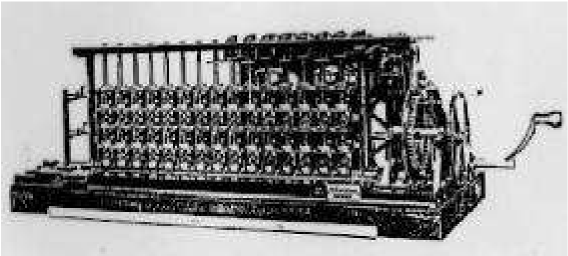
Charles Babbage (1791-1871)



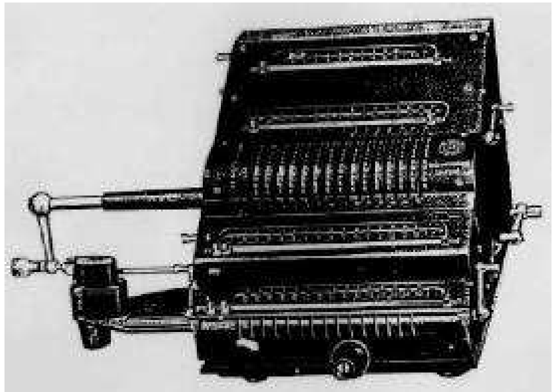


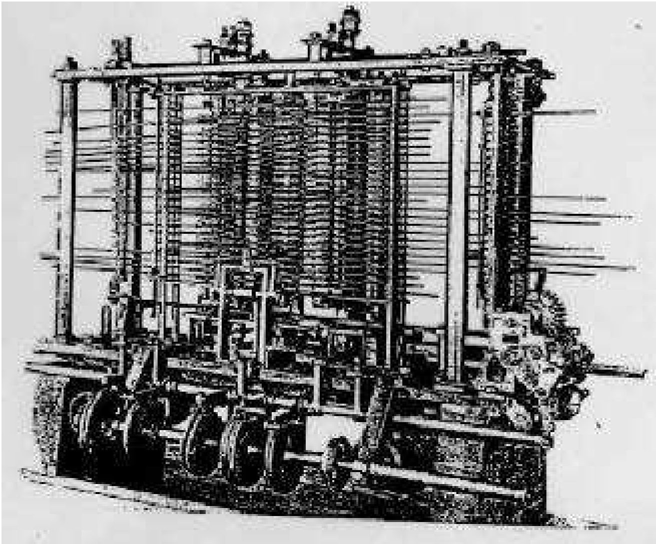
Other early computers and “computer scientists”

Blaise Pascal
1623-1662

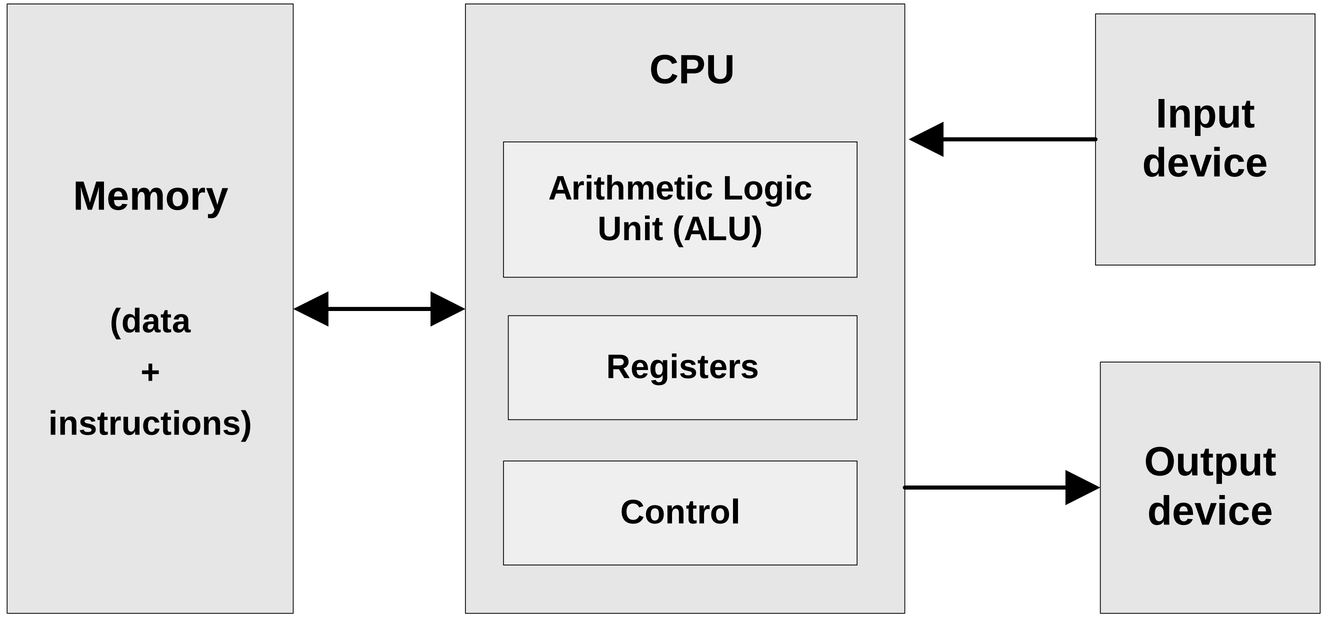


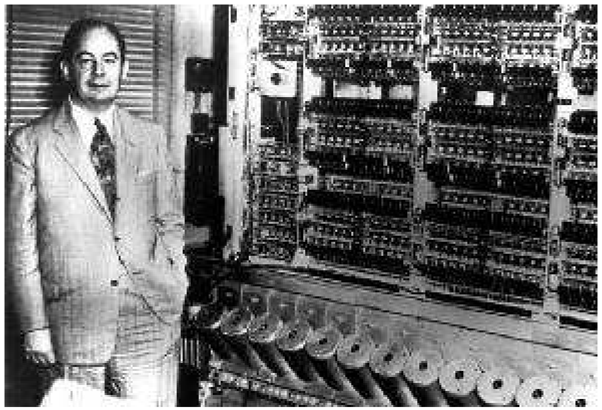
Gottfried Leibniz
1646-1716





Von Neumann machine (c. 1940)



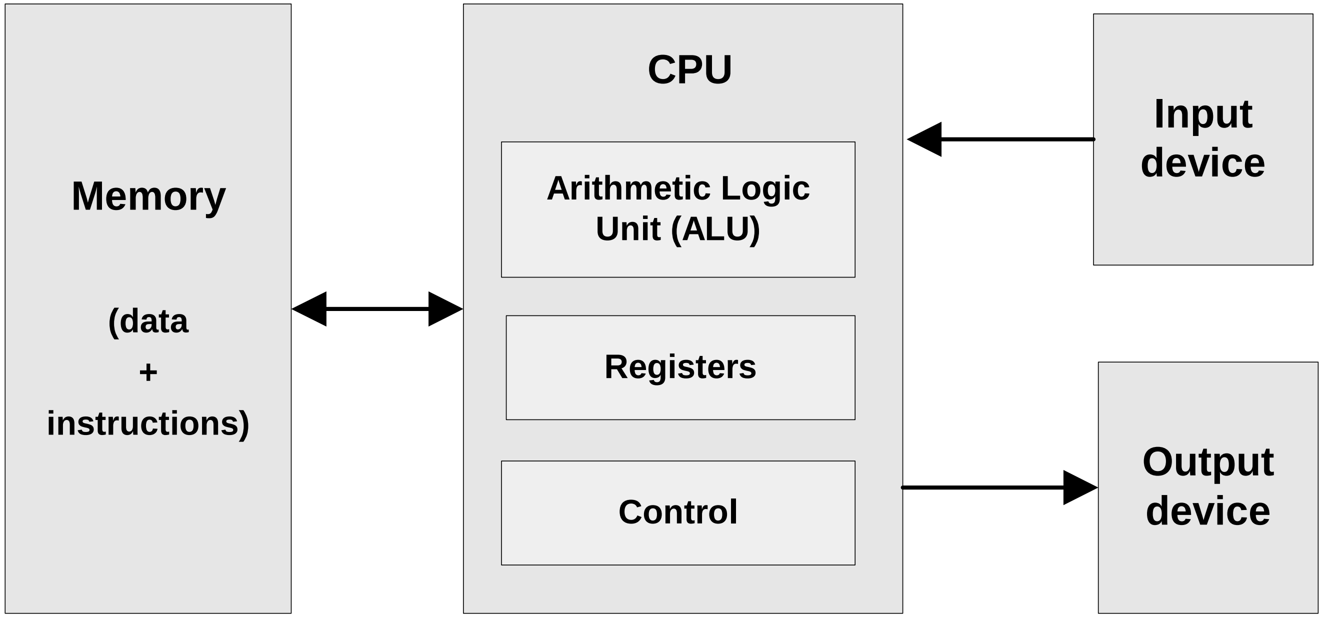


Stored program concept!

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 f(registers)
- Write the ALU output to selected registers
- As a side-effect of this operation,
  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

- 16-bit Von Neumann platform
- *Instruction memory* and *data memory* are physically separate
- Screen: 512 by 256 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

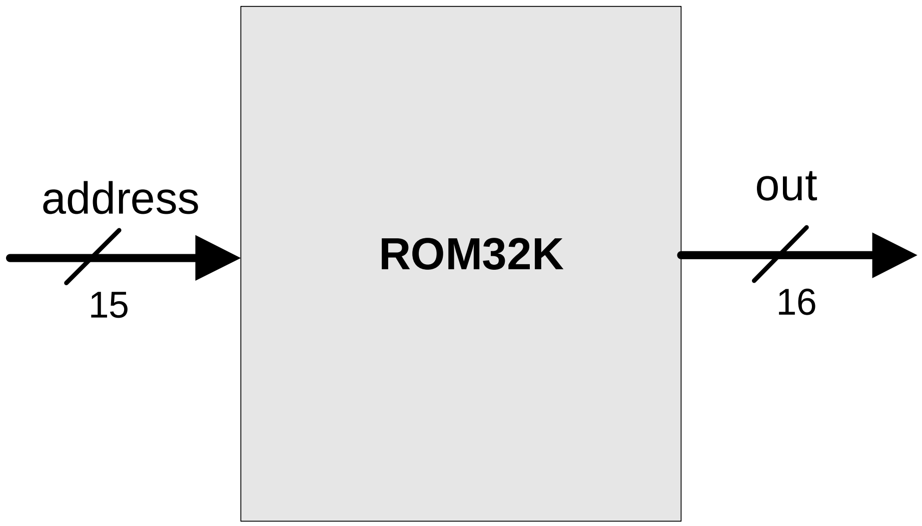
<u>Main parts of the Hack computer:</u>

- Instruction memory (ROM)
- Memory (RAM):
  - Data memory
  - Screen
  - Keyboard
- CPU
- Computer (the logic that holds everything together).

Lecture plan

- Instruction memory
- Memory:
  - Data memory
  - Screen
  - Keyboard
- CPU
- Computer

Instruction memory



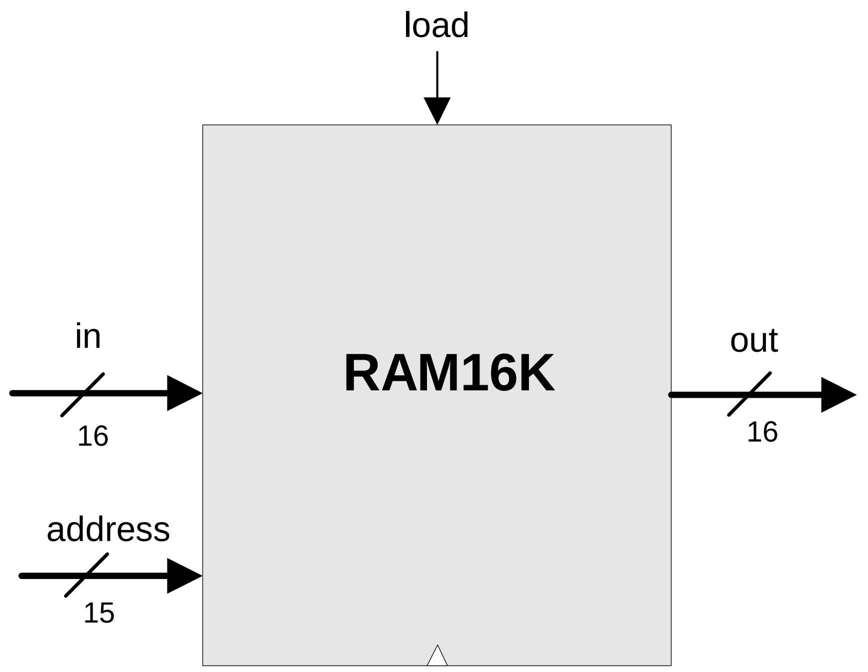
Function:

- Pre-loaded with a machine language program
- Always emits a 16-bit number:

out = ROM32K[address]

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

Data memory



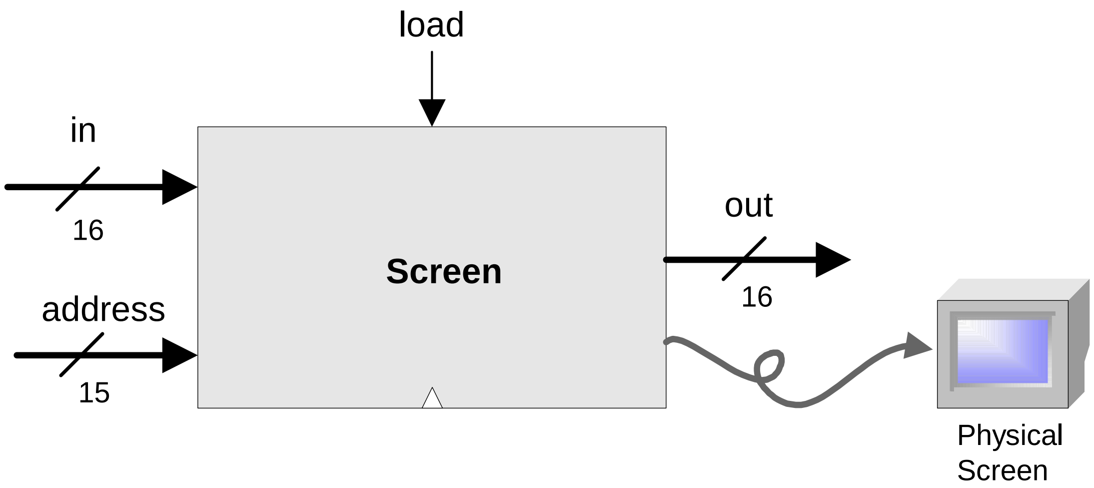
Reading/writing logic

- Low level: Set **address**, **in**, **load** ; probe **out**
- High level (OS): **out = peek(address)**
  poke(address,in).

Lecture plan

- ✓ Instruction memory
- Memory:
  - ✓ Data memory
  - Screen
  - Keyboard
- CPU
- Computer

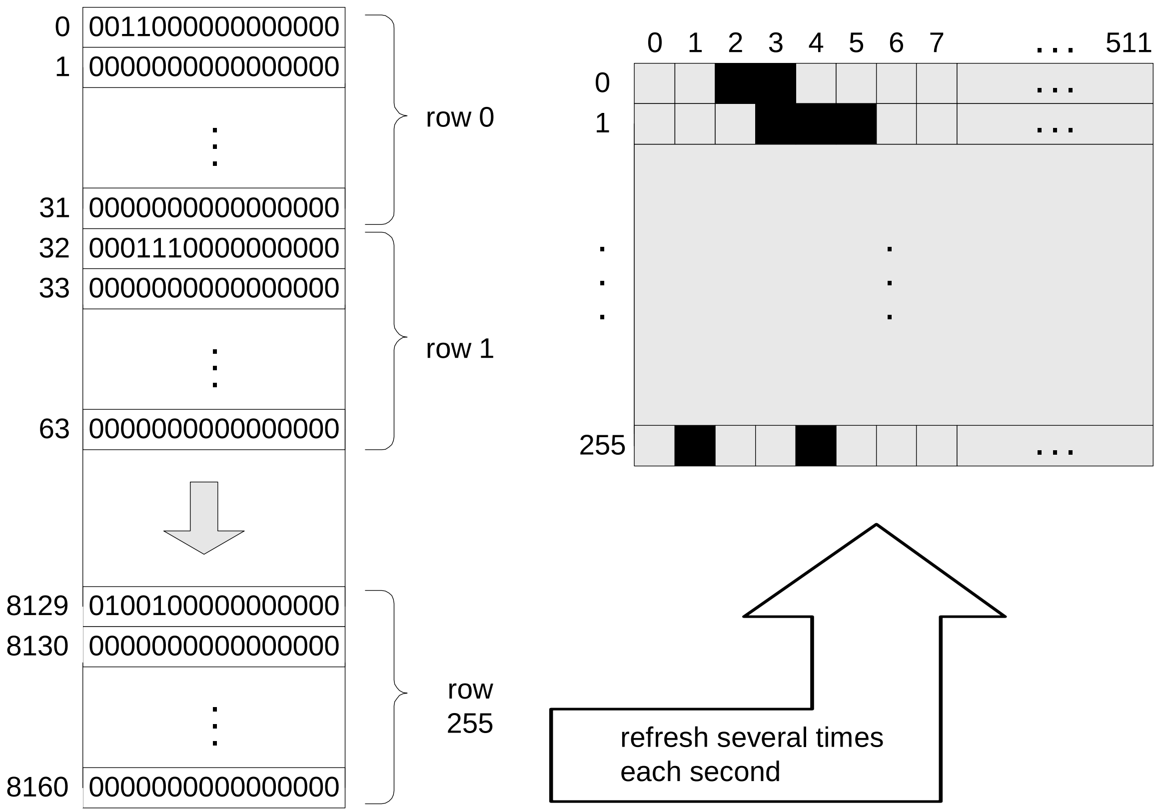
Screen

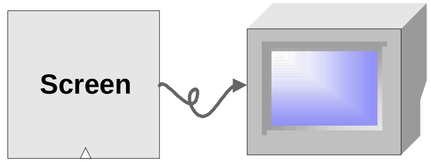


- Functions exactly like a 16-bit 8K RAM :
  - **`out = Screen[address]`**
  - **`If load then Screen[address] = in`**
- Side effect:
  continuously refreshes a 256 by 512 black-and-white screen

(The physical screen is simulated by the supplied hardware simulator;
The screen behavior is implemented by the built-in Screen.hdl chip).

Screen memory map

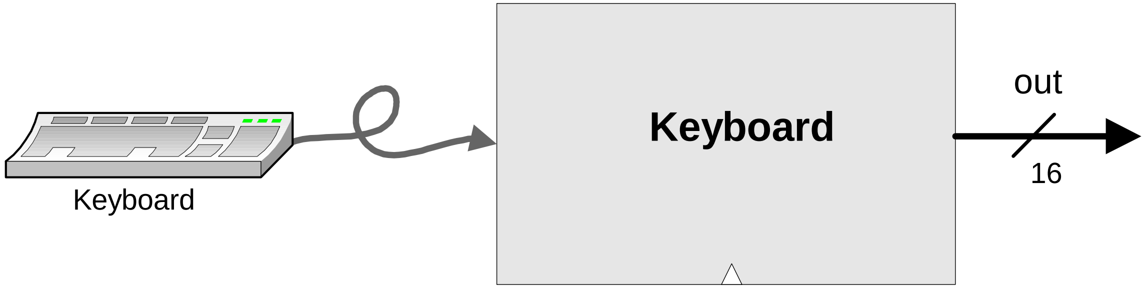




Writing pixel(x,y) to the screen:

- Low level: Set the **y%16** bit of the word found at **Screen[x*32+y/16]**
- High level (OS): **drawPixel(x,y)** (later).

Keyboard



Keyboard chip = 16-bit register

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

Output: Same

Special keys:

Key pressed	Keyboard output	Key pressed	Keyboard output
newline	128	end	135
backspace	129	page up	136
left arrow	130	page down	137
up arrow	131	insert	138
right arrow	132	delete	139
down arrow	133	esc	140
home	134	f1-f12	141-152

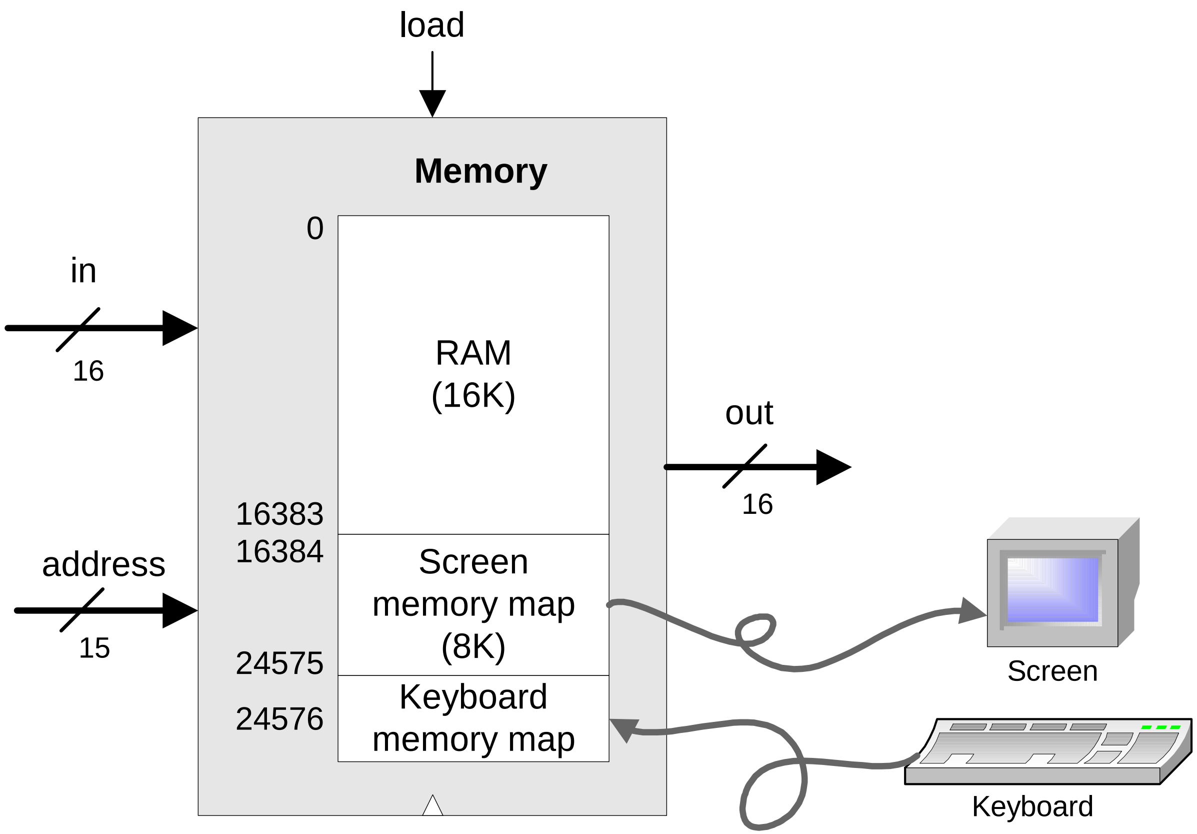
Reading the keyboard:

- Low level: probe the contents of the `Keyboard` register
- High level (OS): `keyPressed()` (later).

The Hack computer

- ✓ Instruction memory
- Memory:
  - ✓ Data memory
  - ✓ Screen
  - ✓ Keyboard
- CPU
- Computer

Memory



<u>Function:</u>

- Access to any address from 0 to 16,383 results in accessing the **RAM**
- Access to any address from 16,384 to 24,575 results in accessing the **Screen** memory map
- Access to address 24,576 results in accessing the **keyboard** memory map
- Access to any other address is invalid.

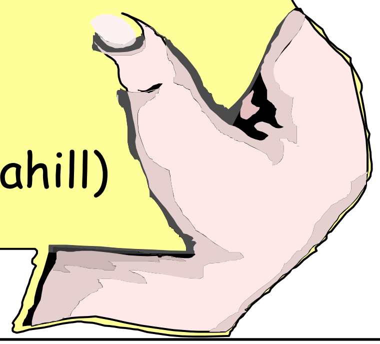
The Hack computer

- ✓ Instruction memory
- ✓ Memory:
  - ✓ Data memory
  - ✓ Screen
  - ✓ Keyboard
- CPU
- Computer

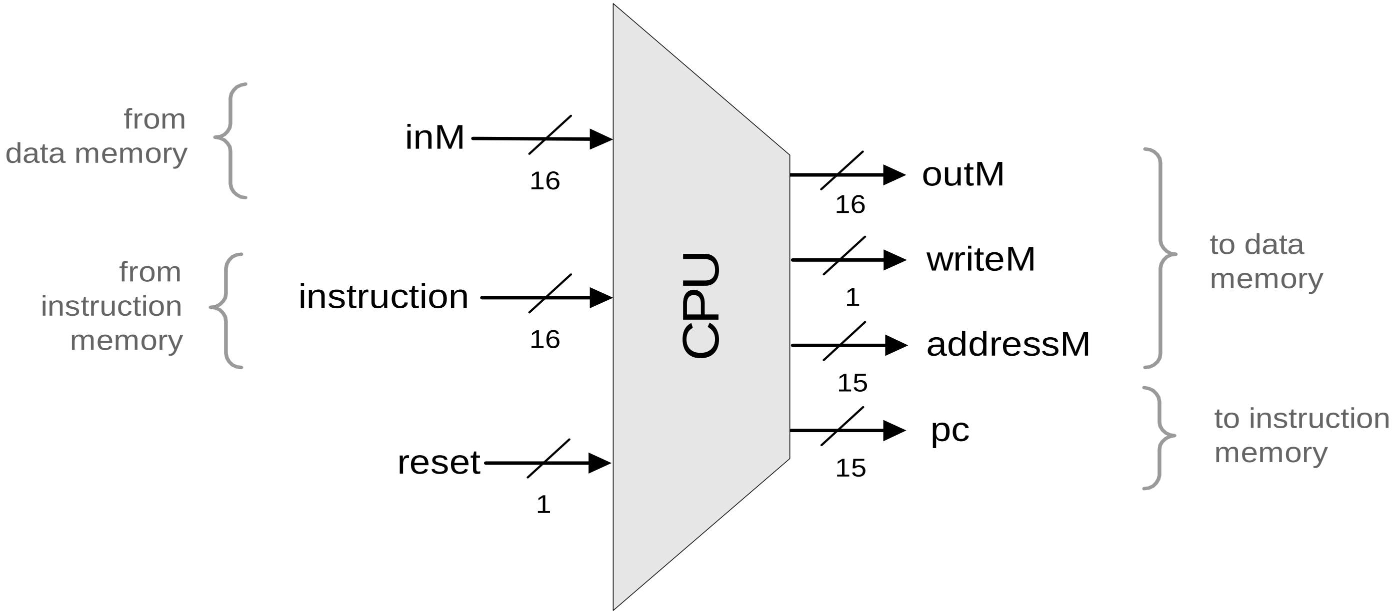
A pledge to patience

"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

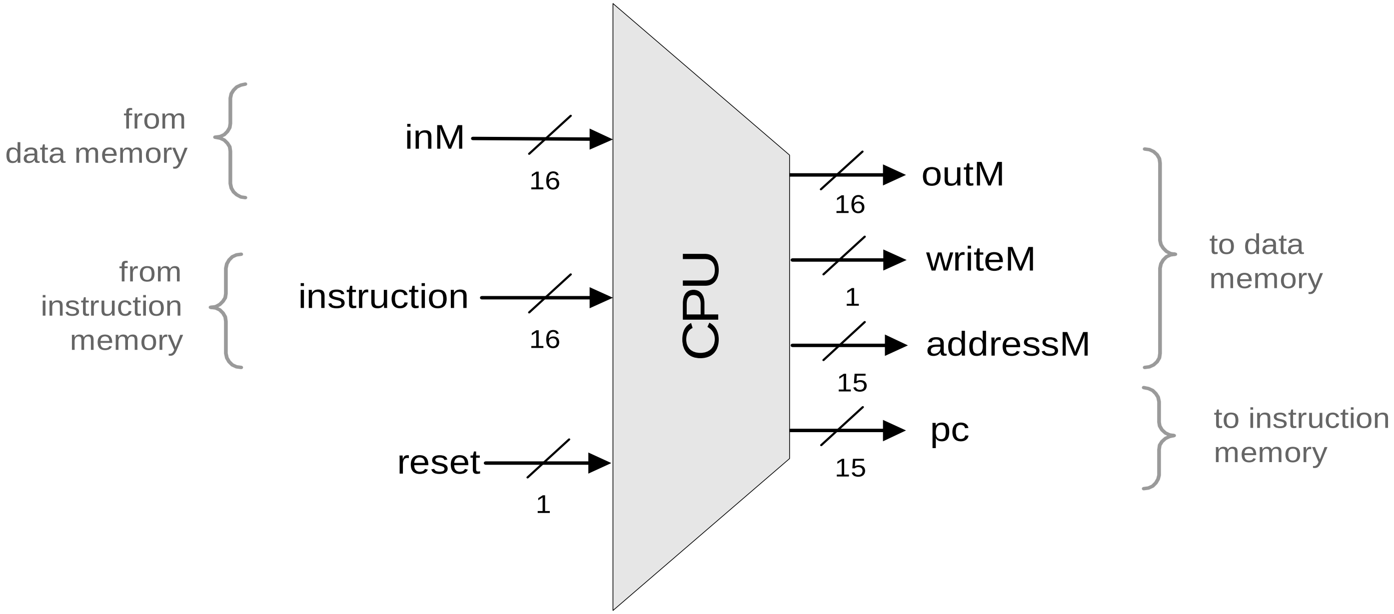


CPU internal components: **ALU** and **A**, **D**, **PC** registers

CPU Function: Executes the **instruction** according to the Hack language specification:

- The **M** value is read from **inM**
- The **D** and **A** values are read from (or written to) these CPU-resident registers
- If the instruction wants to write to **M** (e.g. **M=D**), then the **M** value is placed in **outM**, the value of the CPU-resident **A** register is placed in **addressM**, and **writeM** is asserted
- If **reset==1**, then **pc** is set to 0;
  Otherwise, **pc** is set to the address resulting from executing the current instruction.

CPU




```
CHIP CPU {
    IN  inM[16], instruction[16], reset;
    OUT outM[16], writeM, addressM[15], pc[15];
    PARTS:
    // Implementation missing
}
```

CPU implementation: next 3 slides.

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

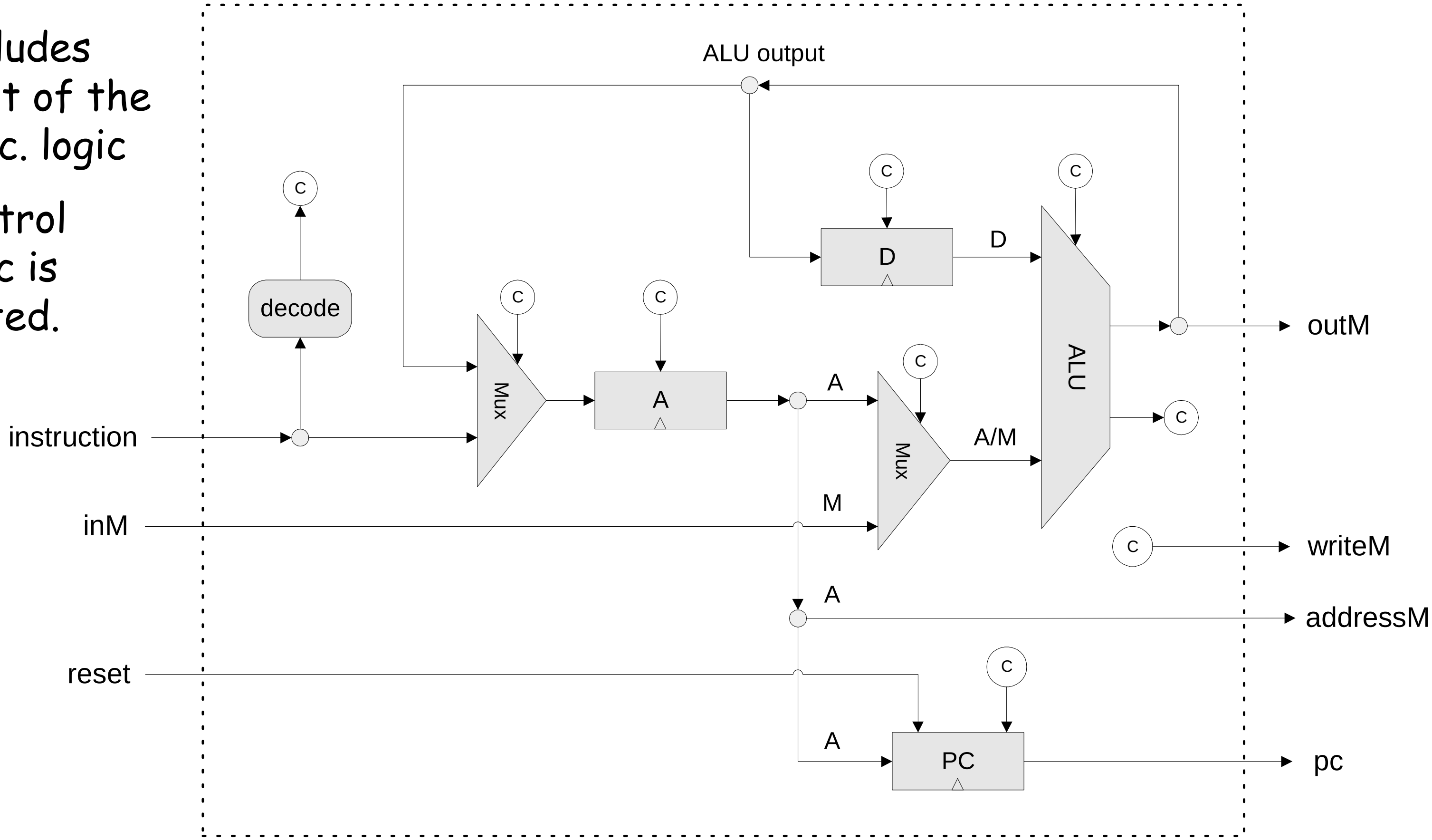
j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

CPU implementation

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

Chip diagram:

- Includes most of the exec. logic
- Control logic is hinted.



Cycle:

- Fetch
- Execute

Execute logic:

- Decode
- Execute

Fetch logic:

If jump then set **PC** to **A**
else set **PC** to **PC+1**

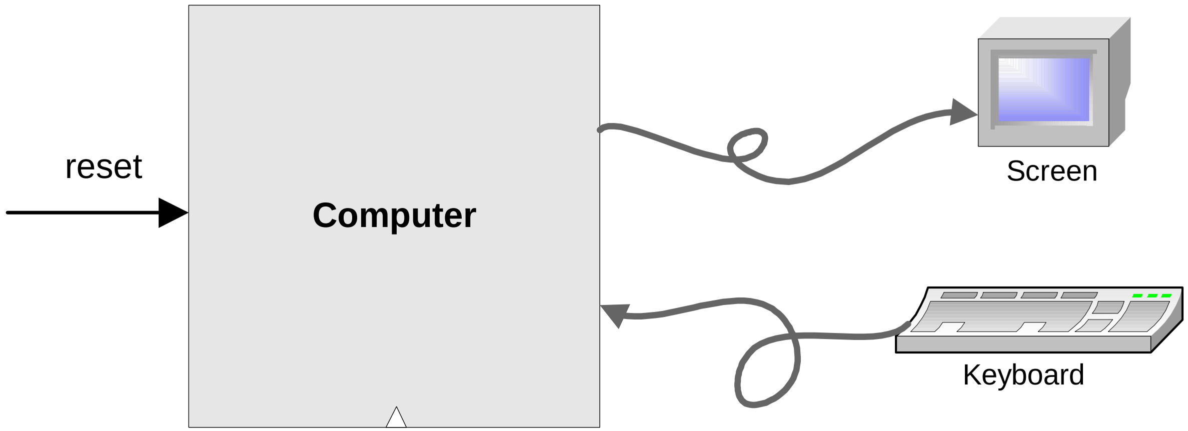
Resetting the computer:

Set **reset** to 1,
then set it to 0.

The Hack computer

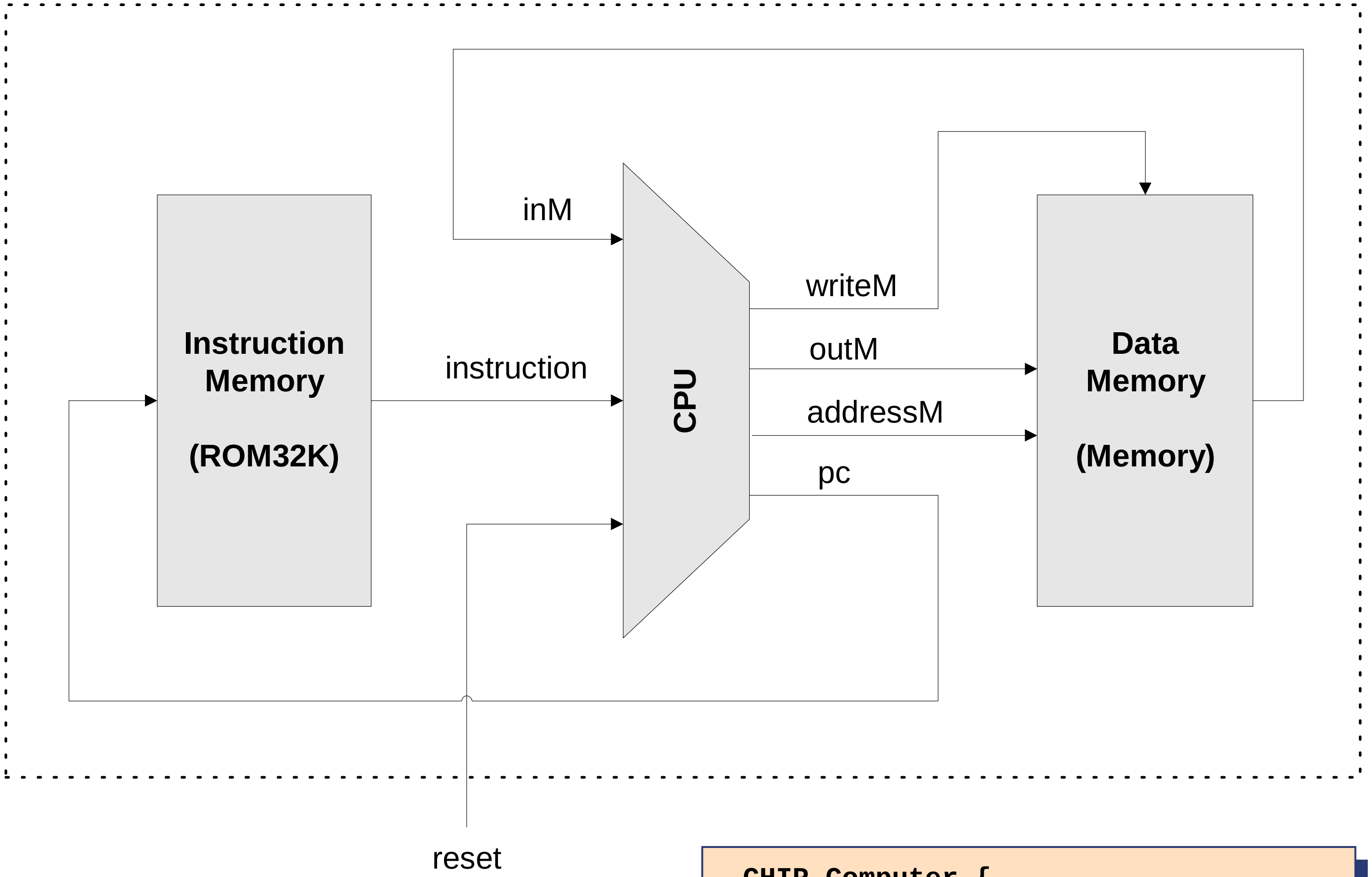
- ✓ Instruction memory
- ✓ Memory:
  - ✓ Data memory
  - ✓ Screen
  - ✓ Keyboard
- ✓ CPU
- Computer

Computer-on-a-chip interface



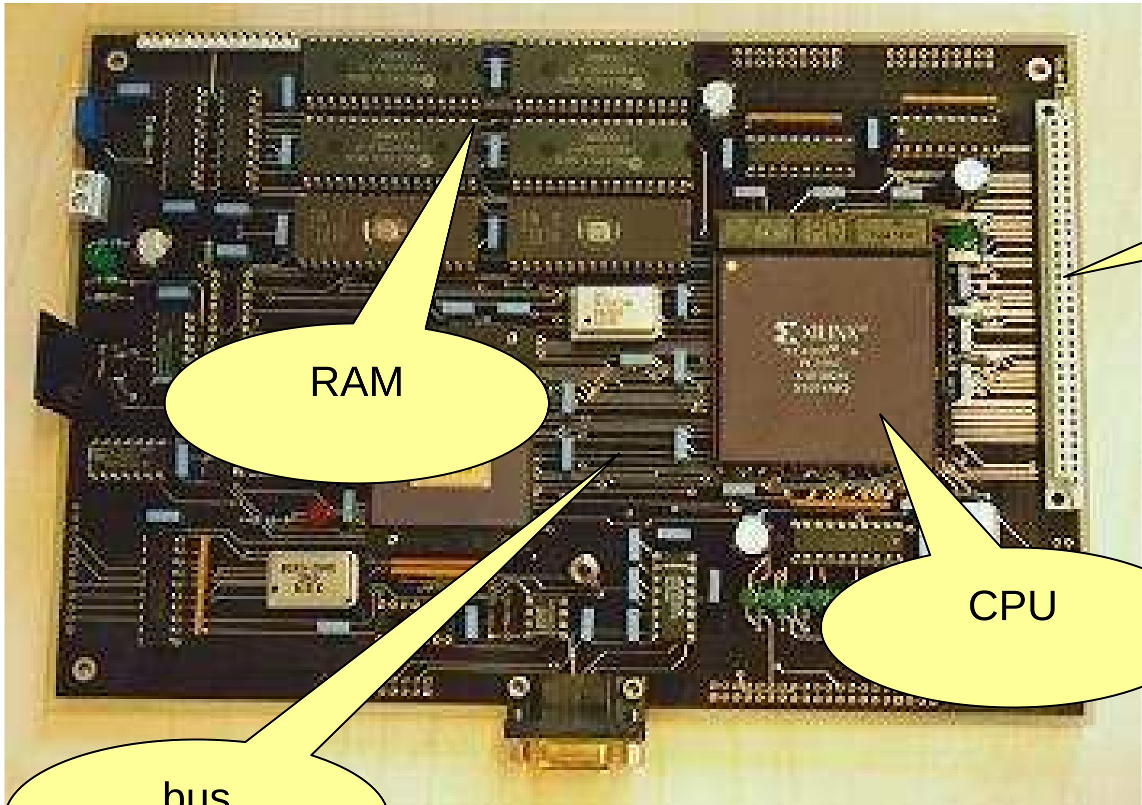

```
Chip Name: Computer  // Topmost chip in the Hack platform
Input:     reset
Function:  When reset is 0, the program stored in the
           computer's ROM executes. When reset is 1, the
           execution of the program restarts. Thus, to start a
           program's execution, reset must be pushed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

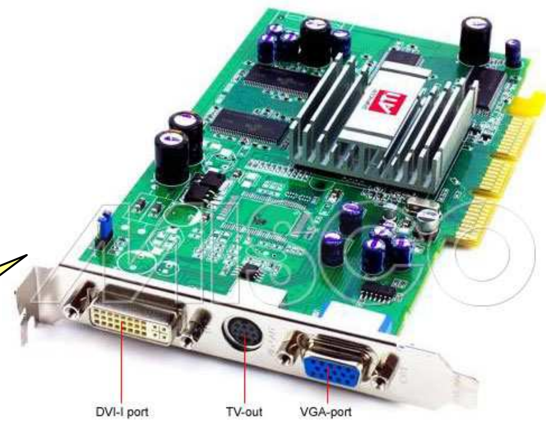



```
CHIP Computer {
    IN reset;
    PARTS:
    // implementation missing
}
```

How it actually looks (thank goodness for abstractions!)



<u>Confession:</u> actually, the parts labeling was made at random ... which helps emphasize the point: thank goodness for abstractions!



Lecture outline

- ✓ ■ Instruction memory
- ✓ ■ Memory:
  - ✓ ● Data memory
  - ✓ ● Screen
  - ✓ ● Keyboard
- ✓ ■ CPU
- ✓ ■ Computer

Perspective: from here to a "real" computer

- Caching
- More I/O units
- Special-purpose processors (I/O, graphics, communications, …)
- Efficiency
- And more ...

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

- CISC / RISC (HW/SW trade-off)
- Hardware diversity: desktop, laptop, hand-held, game machines, …
- General-purpose VS embedded computers
- Silicon compilers
- And more ...

The spirit of things