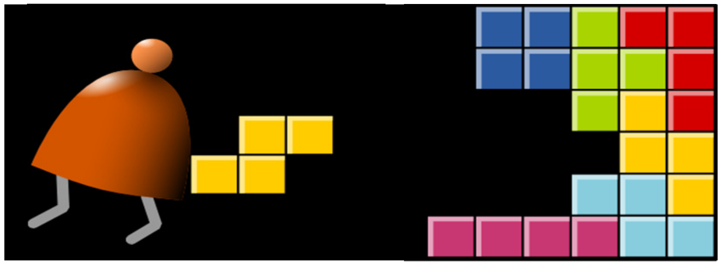
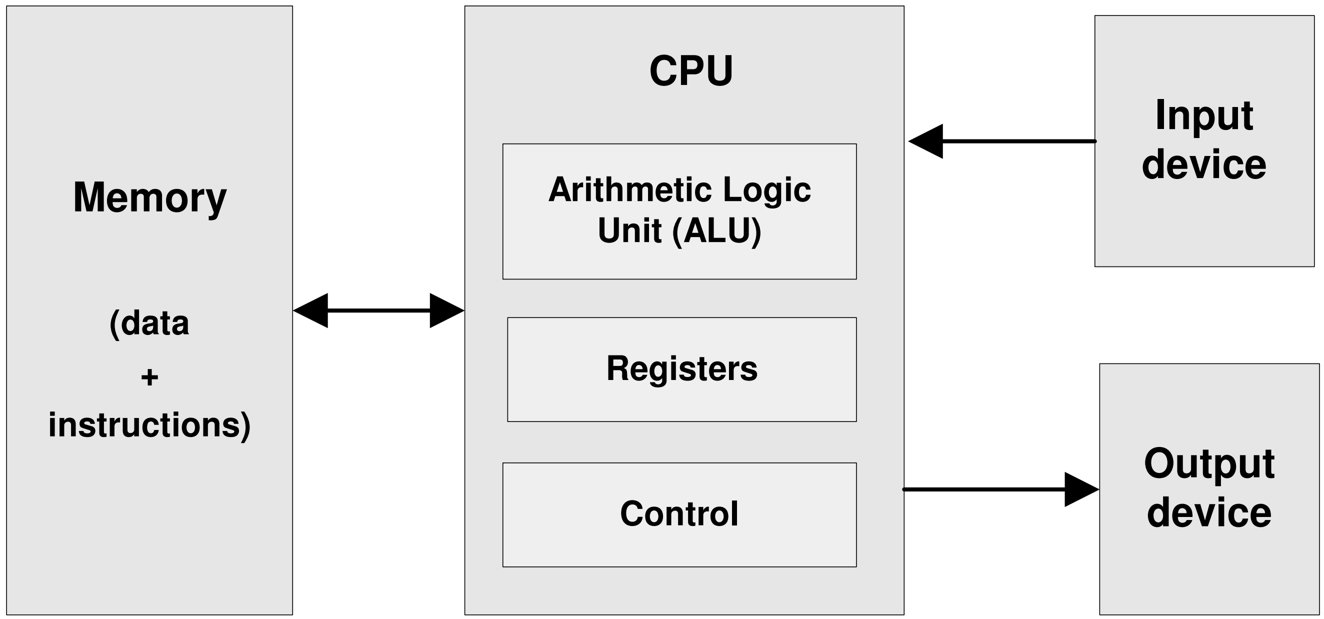
Computer Architecture

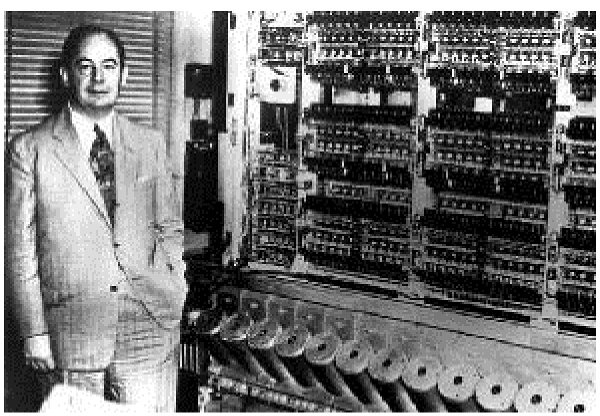


Building a Modern Computer From First Principles

www.nand2tetris.org

Von Neumann machine (circa 1940)



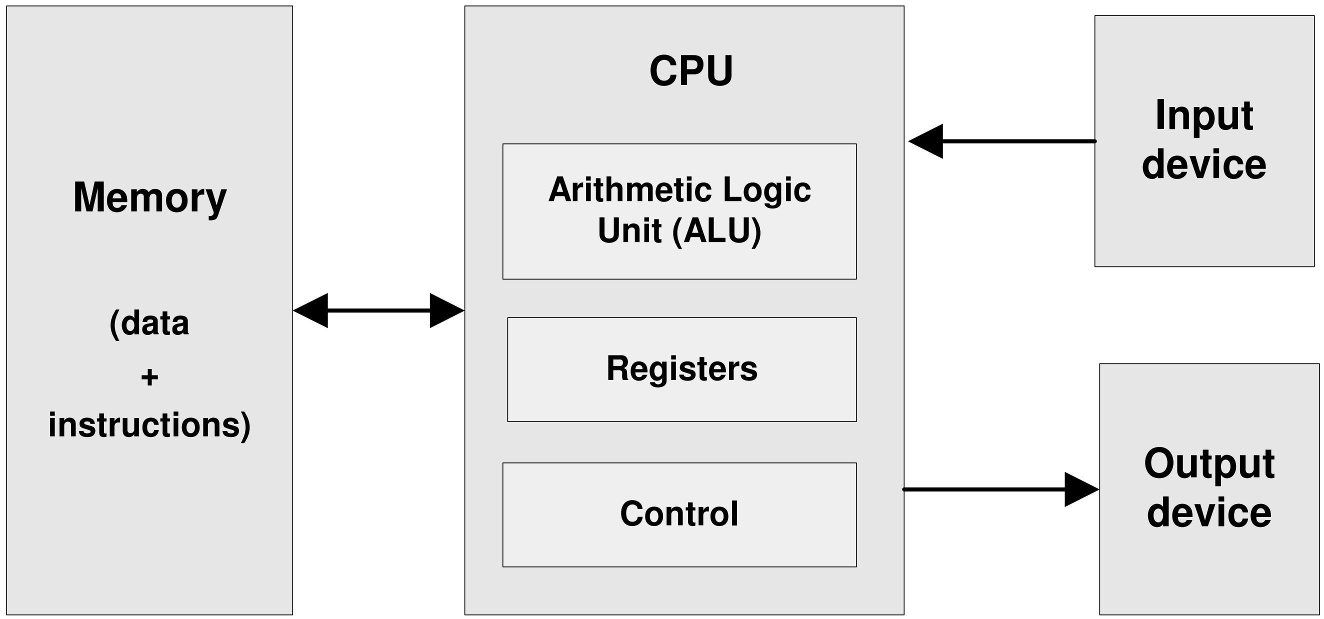


Stored program concept!

John Von Neumann (and others) ... made it possible

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

Processing logic: the instruction cycle



Executing the *current instruction* involves one or more of the following micro-tasks:

- Fetch the next instruction
- Decode the instruction
- Read source operands
- Perform ALU operation
- Store the result

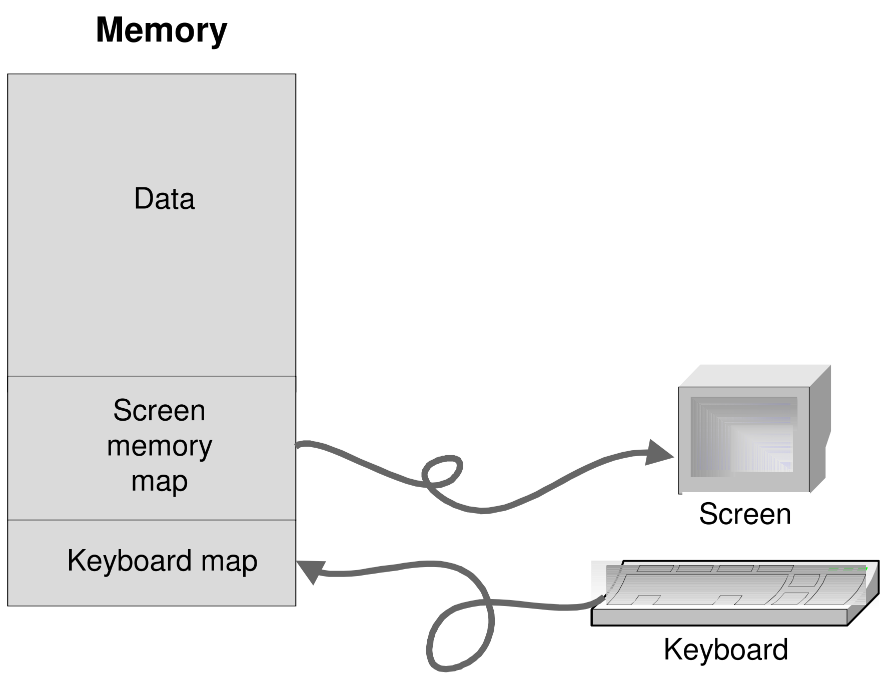
The Hack computer

- A 16-bit Von Neumann platform
- The *instruction memory* and the *data memory* are physically separate
- Screen: 512 columns by 256 rows, 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 (memory map)
  - Keyboard (memory map)
- CPU
- Computer (the logic that holds everything together).

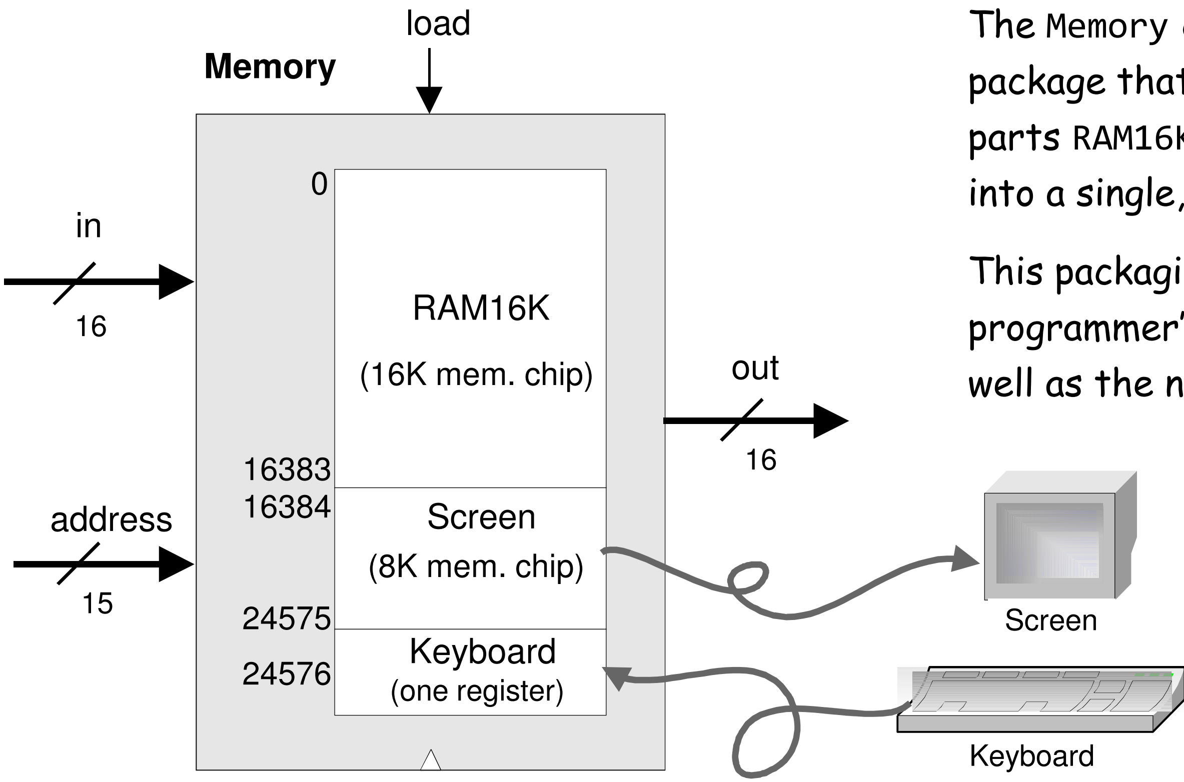
Memory: conceptual / programmer's view



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



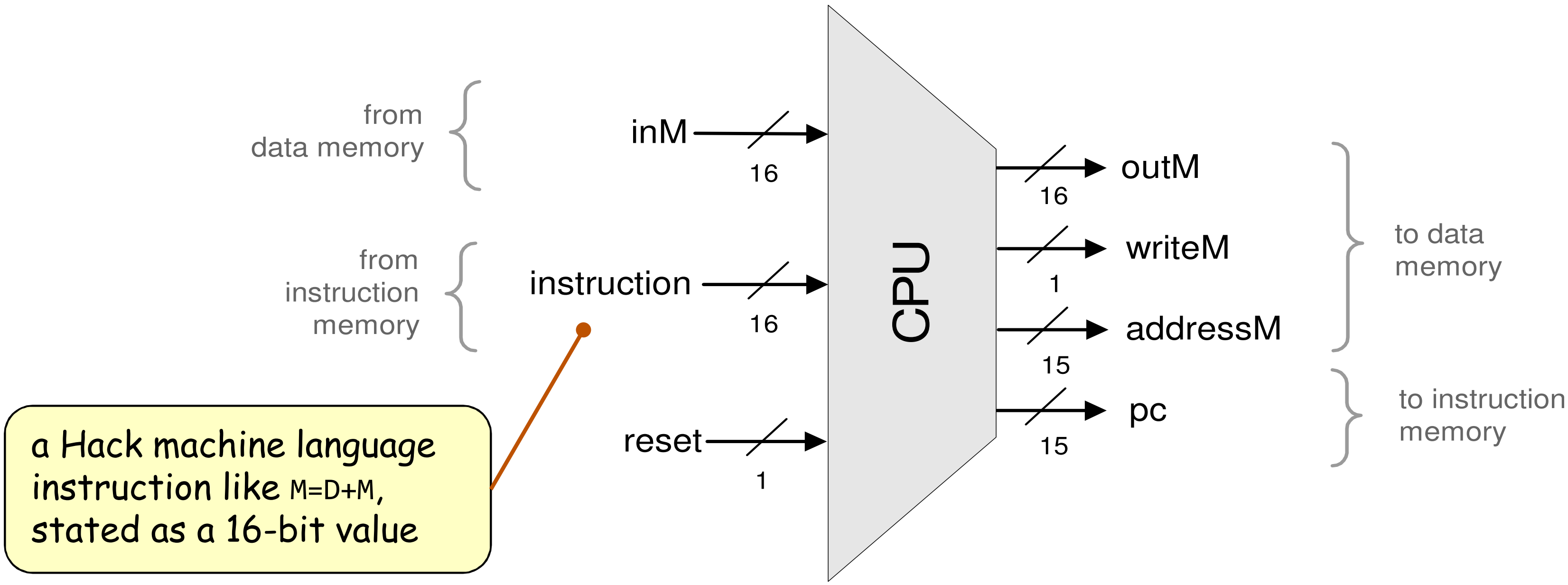
The Memory chip is essentially a package that integrates the three chip-parts RAM16K, Screen, and Keyboard into a single, contiguous address space.

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.

CPU



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

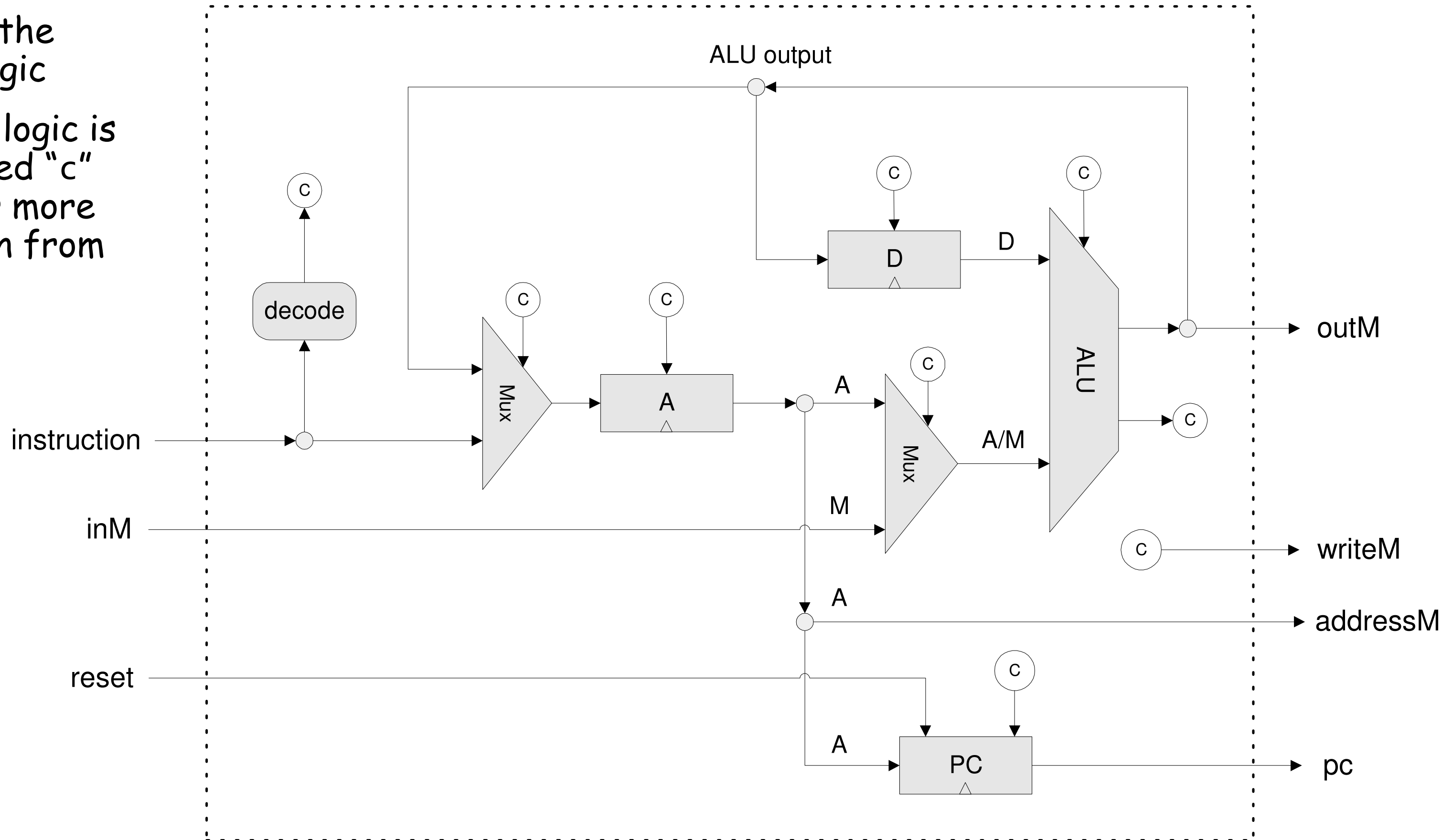
j1 (*out* < 0)	**j2** (*out* $= 0$)	**j3** (*out* > 0)	**Mnemonic**	**Effect**
0	0	0	null	No jump
0	0	1	JGT	If *out* > 0 jump
0	1	0	JEQ	If *out* $= 0$ jump
0	1	1	JGE	If *out* ≥ 0 jump
1	0	0	JLT	If *out* < 0 jump
1	0	1	JNE	If *out* $\neq 0$ jump
1	1	0	JLE	If *out* ≤ 0 jump
1	1	1	JMP	Jump

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 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.

Perspective: from here to a “real” computer

- Pipelining
- Caching
- More I/O units
- Special-purpose processors (I/O, graphics, communications, …)
- Multi-core / parallelism; GPUs
- 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, mobile, game machines, …
- General-purpose vs. embedded computers