The Hack microarchitecture
COMSM1302 Overview of Computer Architecture

John Lapinskas, University of Bristol

The Hack microarchitecture

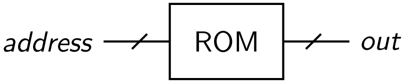
This week, your main assignment is to build a Hack computer in Logisim. From next week, we'll fully move on from hardware to software, and in particular to the process of translating high-level languages into assembly.

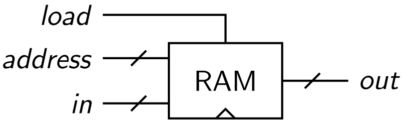
The main focus here is on the computer itself — it's not too hard to see where Hack's memory-mapped I/O would fit in, but this is non-examinable.

You've already built all the components you need, and for convenience we've loaded model versions into the skeleton Logisim file.

We do recommend you use the Logisim built-in ROM, RAM and registers rather than the ones you designed, to make it easier to view and edit the contents in Logisim and actually test the CPU!

Components: Memory



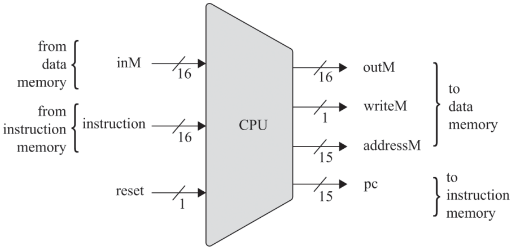


- 64KB.
- 15-bit address space.
- 16-bit words.
- $\mathit{out} = \mathrm{ROM}[\mathit{address}]$ unclocked.
- Stores program to be executed.

- 64KB.[1]
- 15-bit address space.[1]
- 16-bit words.
- $\mathit{out} = \mathrm{RAM}[\mathit{address}]$ unclocked.
- Stores data.
- On each clock tick, if $\mathit{load} = 1$, update RAM[*address*] to *in*.

[1] Normally Hack would have 32KB RAM in a 14-bit address space, but here we use 64KB. Logisim can't simulate a screen easily so we just store the pixels in RAM[0x4000]–RAM[0x5xFFFF], using real memory instead of memory-mapping.

Components: The CPU



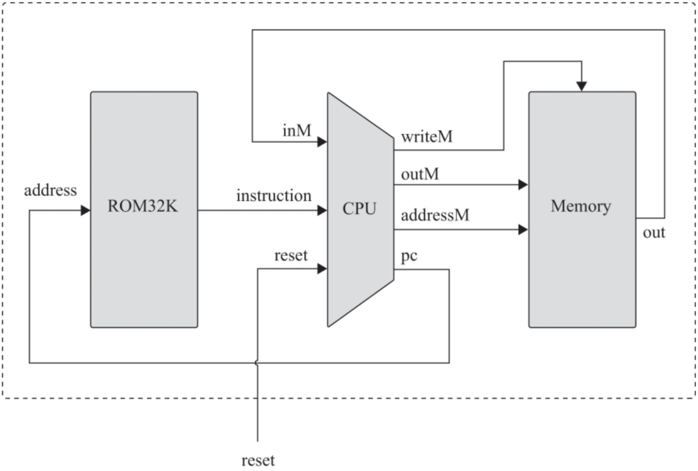
Source: Nisan and Schocken

The CPU should follow the fetch-execute cycle as discussed earlier in the unit.

After each clock tick:

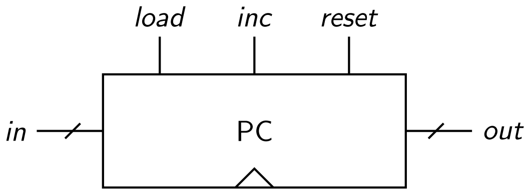
- The CPU should execute the *instruction* input, ROM[*pc*], as machine code.
- *inM* should contain the value of *M* (loaded from RAM).
- *pc* should contain the value of the program counter.
- *addressM* should contain the value of *A*.
- If the CPU is writing to RAM, *writeM* should be 1 and *outM* should be the value being written. Otherwise, *writeM* should be 0.
- If *reset* is high, the PC should be set to zero. (No need to reset *A* or *D*.)

How the CPU fits in



Source: Nisan and Schocken

CPU components: The program counter



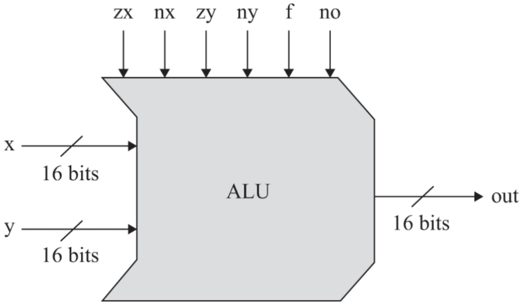
At most one of the *load*, *inc* or *reset* inputs should be high at once.
in is a 16-bit input, and *out* is a 16-bit output.

On each clock cycle:

- If $reset = 1$, set $out \leftarrow 0$.
- If $inc = 1$, set $out \leftarrow out + 1$.
- If $load = 1$, set $out \leftarrow in$.

(A normal register is the same, but without the *inc* or *reset* inputs.)

CPU components: The ALU



Source: Nisan and Schocken (with minor adjustment)

The ALU is unclocked. It should compute *out* from *x*, *y*, *zx*, *nx*, *zy*, *ny*, *f* and *no* as shown in the table on the next slide.

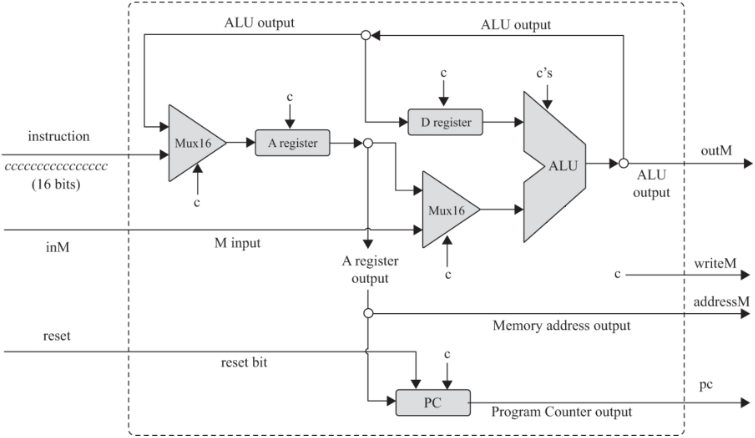
To evaluate the `jump` part of a C-instruction, you will need another sub-circuit to check whether *out* is positive, negative, or zero.

CPU components: The ALU

zx	*nx*	*zy*	*ny*	*f*	*no*	*out*
1	0	1	0	1	0	`0`
1	1	1	1	1	1	`1`
1	1	1	0	1	0	`-1`
0	0	1	1	0	0	`x`
1	1	0	0	0	0	`y`
0	0	1	1	0	1	`!x`
1	1	0	0	0	1	`!y`
0	0	1	1	1	1	`-x`
1	1	0	0	1	1	`-y`
0	1	1	1	1	1	`x+1`
1	1	0	1	1	1	`y+1`
0	0	1	1	1	0	`x-1`
1	1	0	0	1	0	`y-1`
0	0	0	0	1	0	`x+y`
0	1	0	0	1	1	`x-y`
0	0	0	1	1	1	`y-x`
0	0	0	0	0	0	`x&y`
0	1	0	1	0	1	`x|y`

Notice these inputs match $\mathtt{c_1c_2c_3c_4c_5c_6}$ of `comp` in a C-instruction...

Building the CPU



Source: Nisan and Schocken (with minor adjustment)

This is only one possible implementation!