CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 2000 Computer Systems
Lecture 9

Review – Last Lecture

- This lecture we're going to talk about:
  - History of Architecture
  - Memory and I/O
  - The HACK machine

What we’re doing now

- The CPU and basic computers
- The last part of hardware (at least in lectures).
- As with other parts of this course we will give a basic outline and you will work at filling in the gaps in worksheets tutes and pracs.

What we have seen so far

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

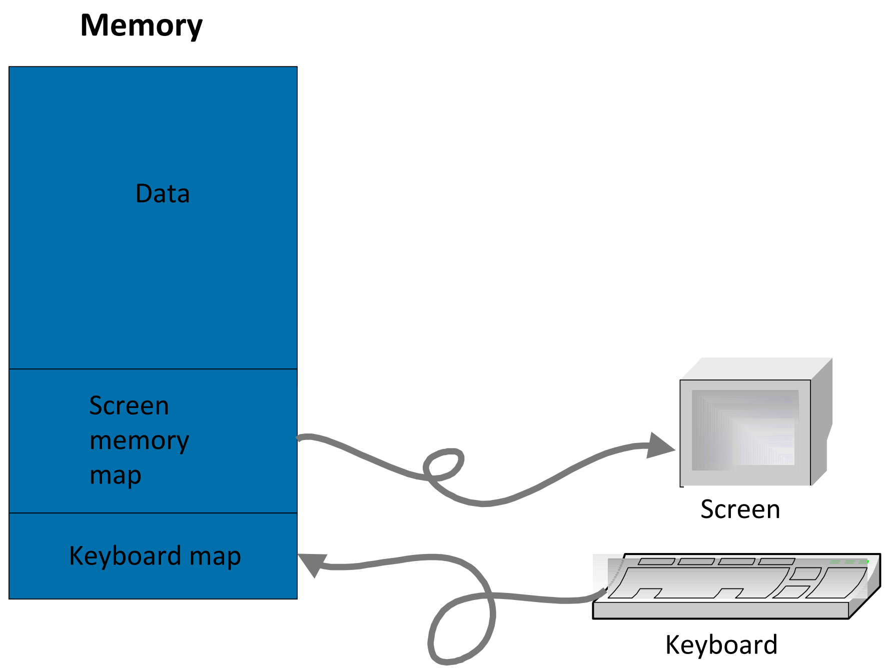
- Memory
- CPU
- Computer

The last parts!

Things to remember

- Every wire has a signal
  - For all intents and purposes this will be a 0 (false) or a 1 (true)
- Every chip will produce a predictable response to its inputs.
  - Input includes the clock on clocked (memory chips)
- Memory chips will only change if the load signal is 1 (true)
- Output devices are mapped into ordinary read/write memory.
- Here in terms of circuit design we:
  - Perform all operations
  - Connect to all outputs
  - Select what to use later

Memory Mapping – Conceptual view



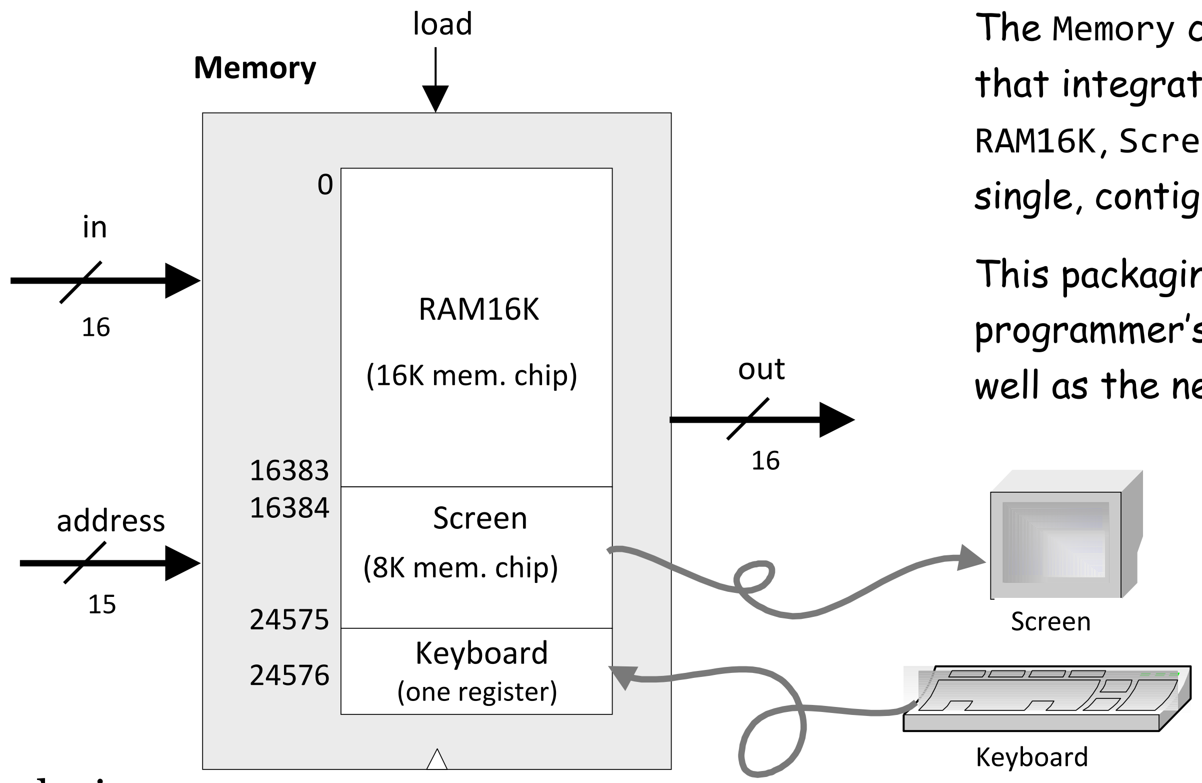
From the program’s point of view our RAM is just 32K of read/write memory

Note that Hack is a <u>word</u>-addressable machine

<u>Using the memory:</u>

- ❑ 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 Mapping – Detailed View



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.

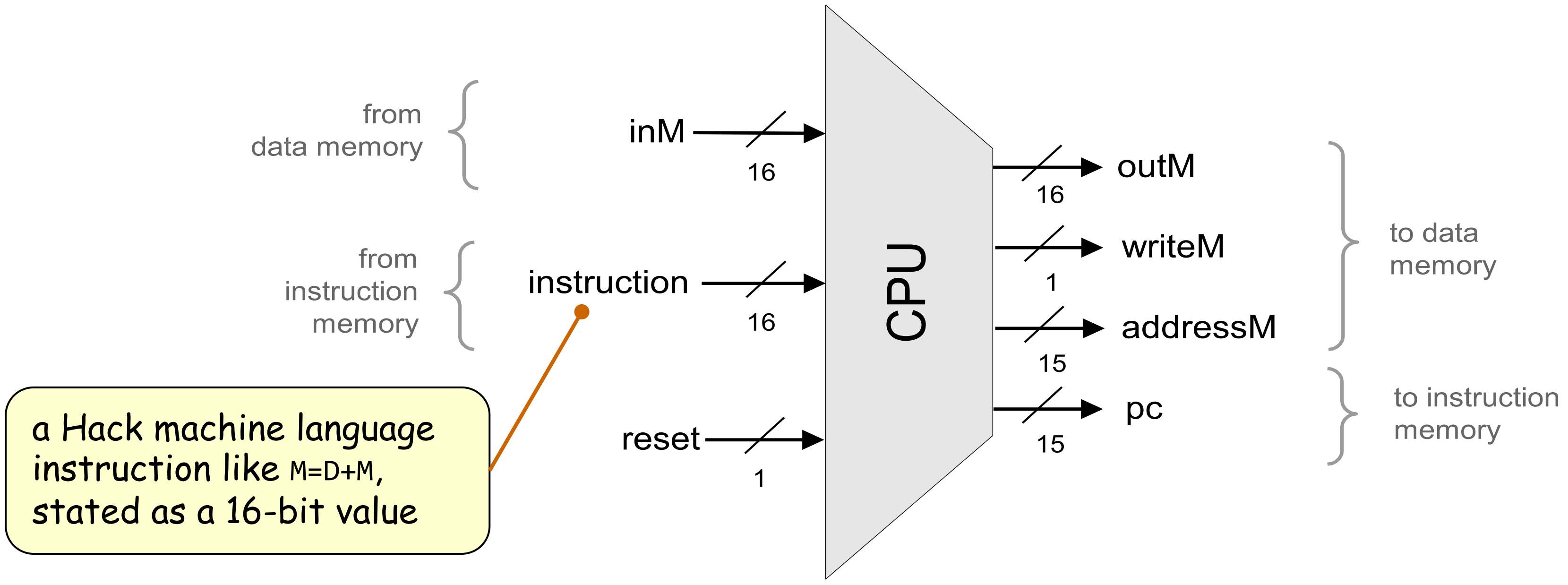
Access logic:

- 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 Plan

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

CPU – From the Outside



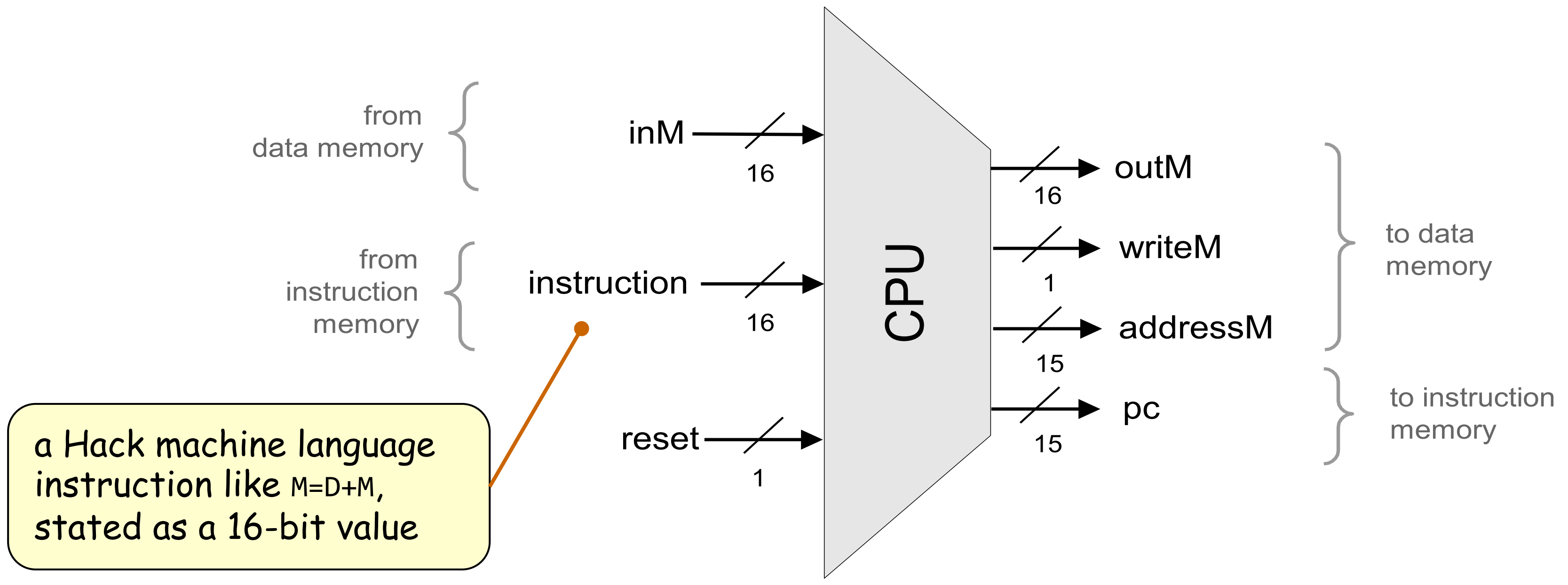
CPU internal components (invisible in this chip diagram): `ALU` and 3 registers: `A`, `D`, `PC`

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



<u>CPU internal components </u>(invisible in this chip diagram): `ALU` and 3 registers: `A`, `D`, `PC`

<u>CPU fetch logic:</u>

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

<u>If `reset==0`:</u> 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`

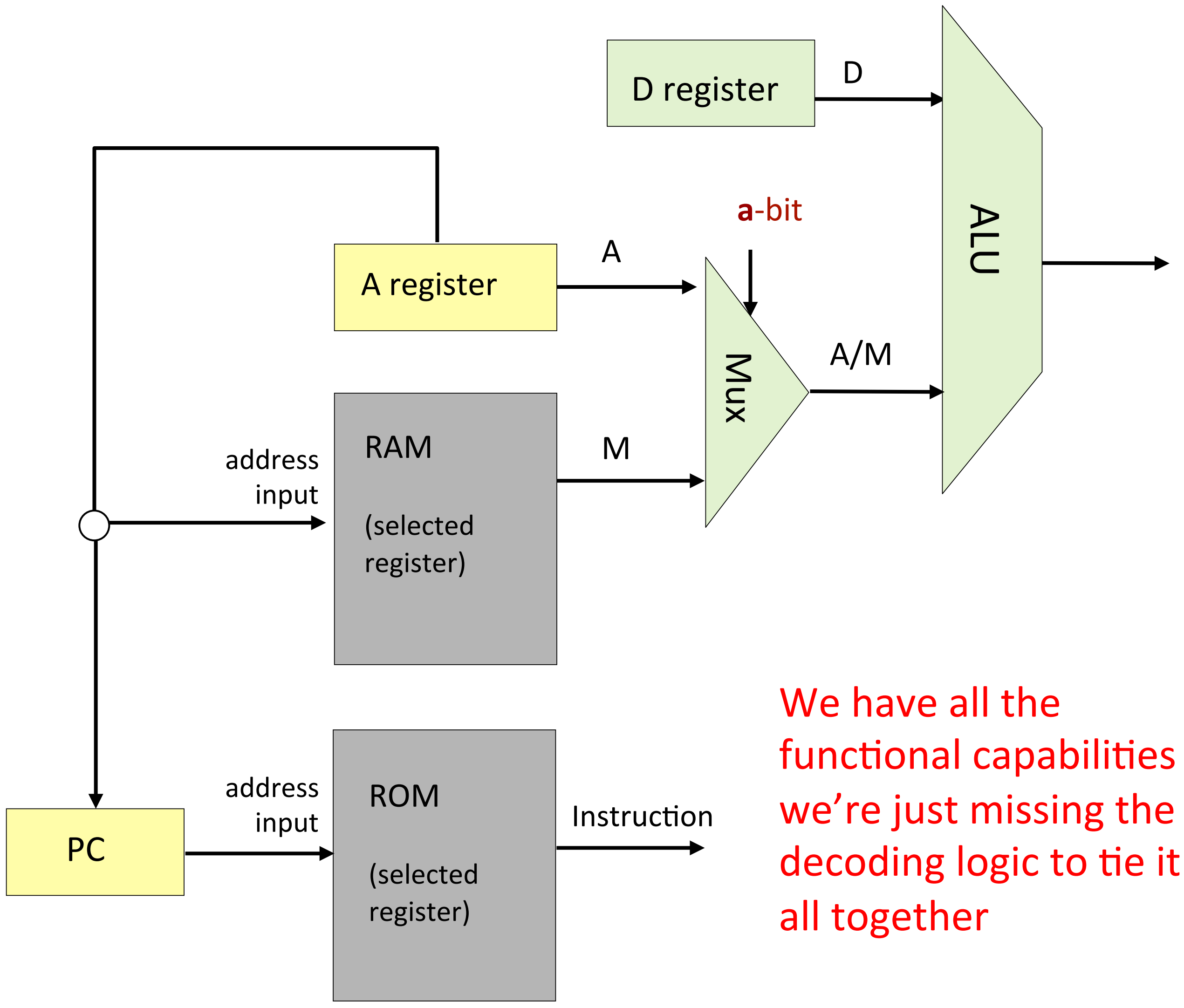
<u>If `reset==1`:</u> the `PC` is set to `0`. (restarting the computer)

Remember…

- When building Computer Systems you have to think of how to build what you want with the tools that you have..
  - It's is the same for <u>any</u> language
- In this case you have all the hardware components listed so far
  - With the signals they take in and produce
- Plus a toolbox of gates and chips you can use
- How to build a CPU?

What already have

RAM and ROM are not part of CPU



We have all the functional capabilities we're just missing the decoding logic to tie it all together

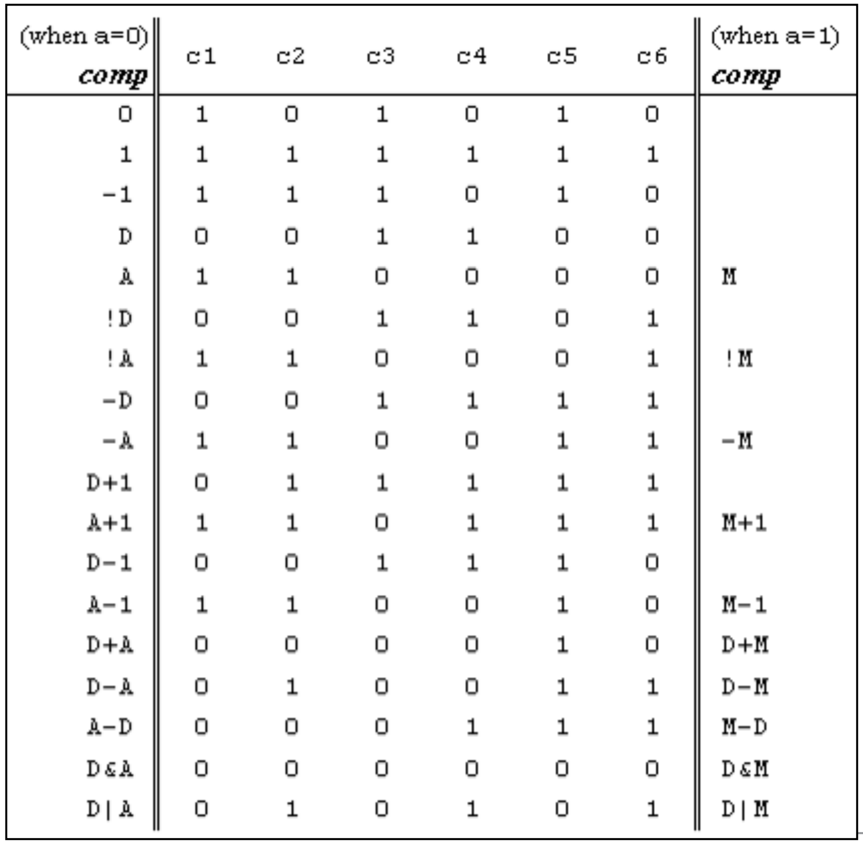
Think in terms of Instructions

- Recall in tutorial question 1 b) we designed circuitry to decode just an A-instruction.
  - This required a circuit that used the most significant bit (bit-15) to help activate the load wire on the A-Register
  - The circuit also had to feed the whole instruction to the A-Register
- Introducing the C-instruction requires extra machinery
  - For a start to this see the answer presented in tutes.

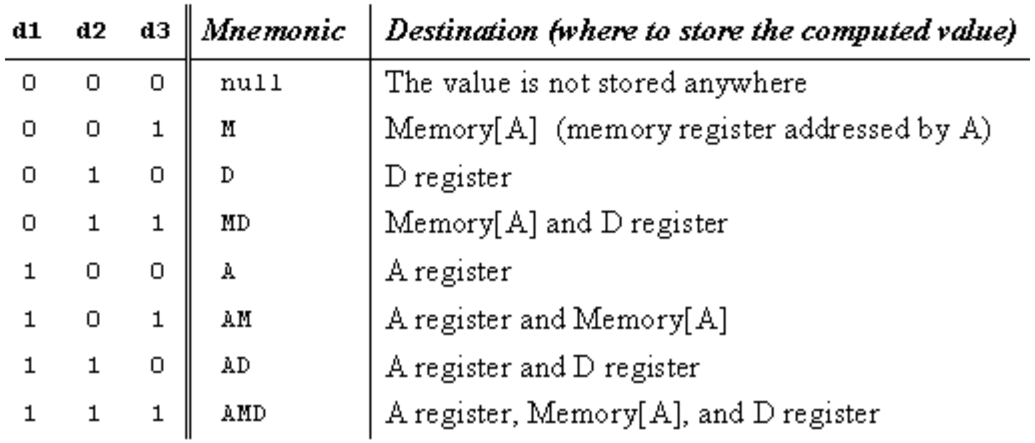
The *C*-instruction revisited

dest = comp; jump

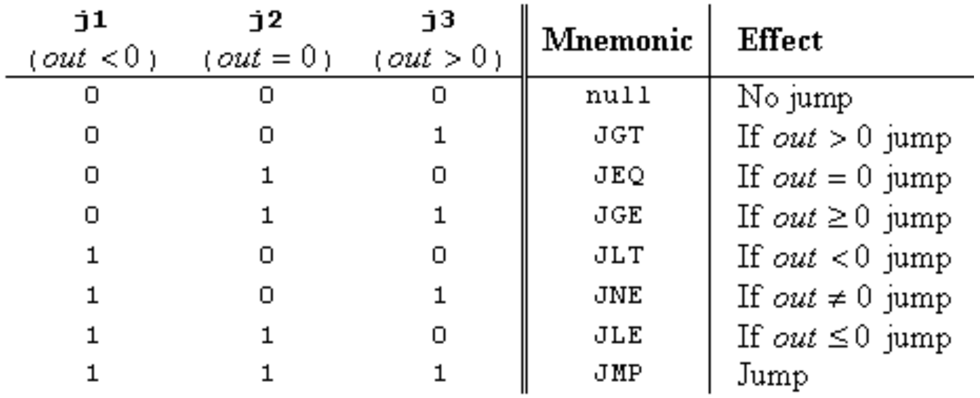
binary: $1\ 1\ 1\ \overbrace{a\ c1\ c2\ c3\ c4\ c5\ c6}^{comp}\ \overbrace{d1\ d2\ d3}^{dest}\ \overbrace{j1\ j2\ j3}^{jump}$



(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



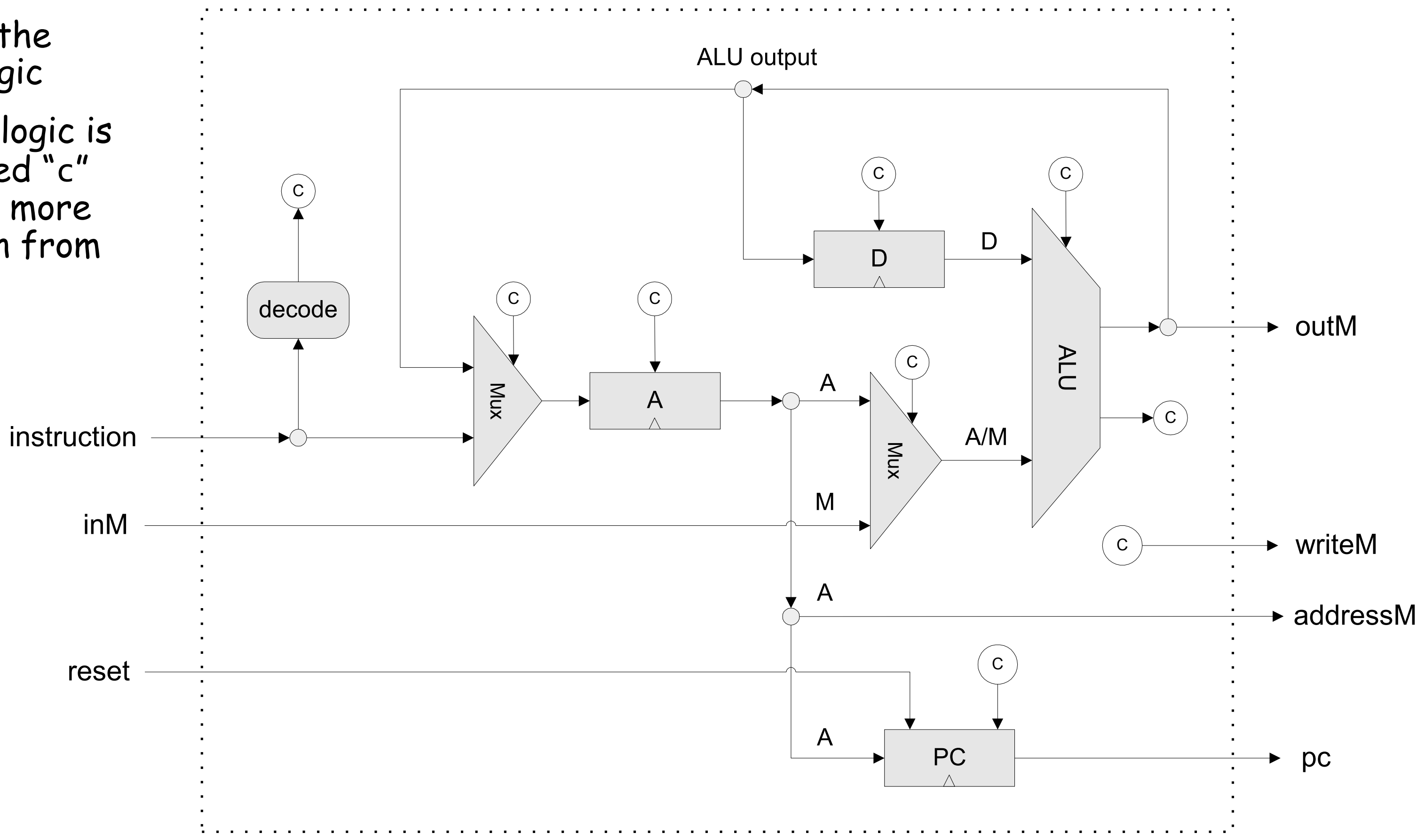
j1 ($out < 0$)	**j2** ($out = 0$)	**j3** ($out > 0$)	**Mnemonic**	**Effect**
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

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 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.

CPU implementation

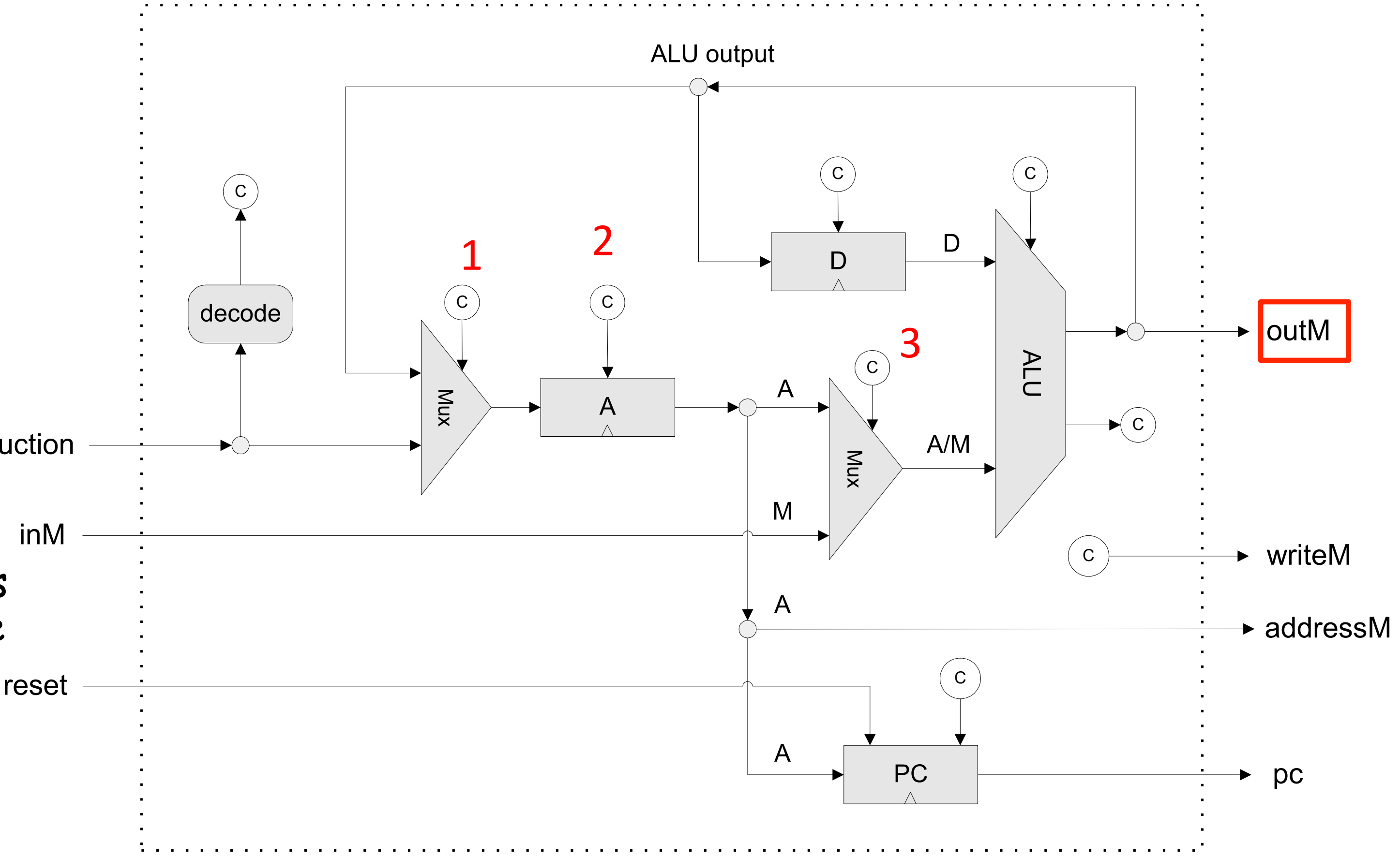
dest = comp; jump			
	comp	*dest*	*jump*
binary: 1 1 1 a	c1 c2 c3 c4	c5 c6 d1 d2	d3 j1 j2 j3

Class Exercise

For a C-instruction

Which bits of the instruction are used as © signals on wires 1, 2, and 3 of the diagram?

Why is there always a signal on outM when memory is not always a destination for the output?



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:

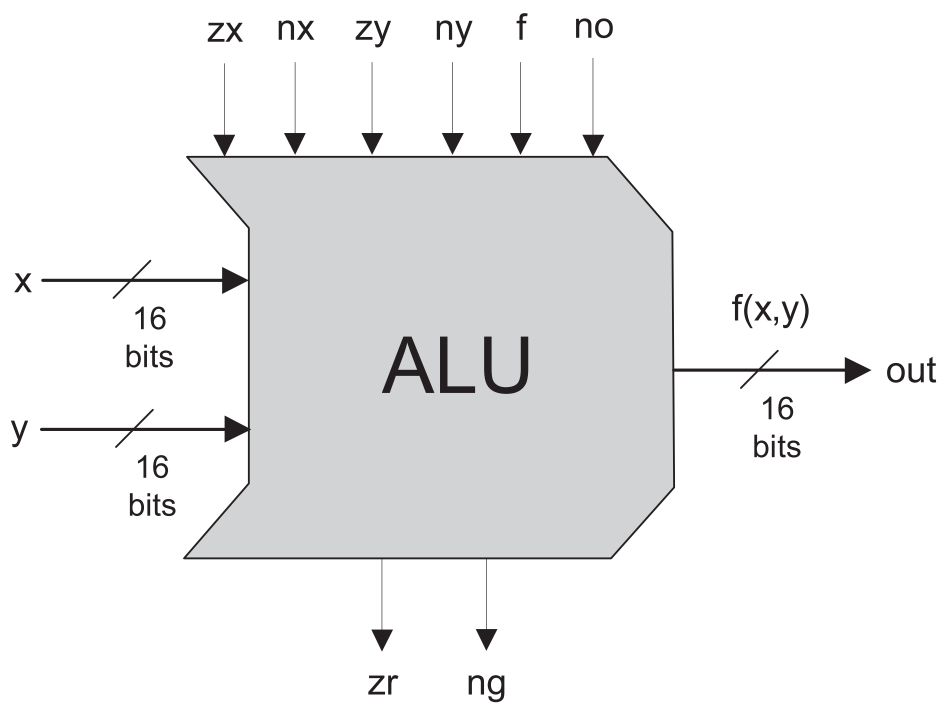
Set `reset` to `1`,
then set it to `0`.

Observations

- We can use individual bits from the C-instruction to control:
  - The ALU (multiple wires)
  - The Choice of Source (A,D,M)
  - The Choice of Destination (A,D,M)
  - Some Input signals to the PC to let it know if it should be updated with the value of A
    - Note that this decision also depends on some output status bits from the ALU..

Recall the ALU

These bits come from the C-Instruction

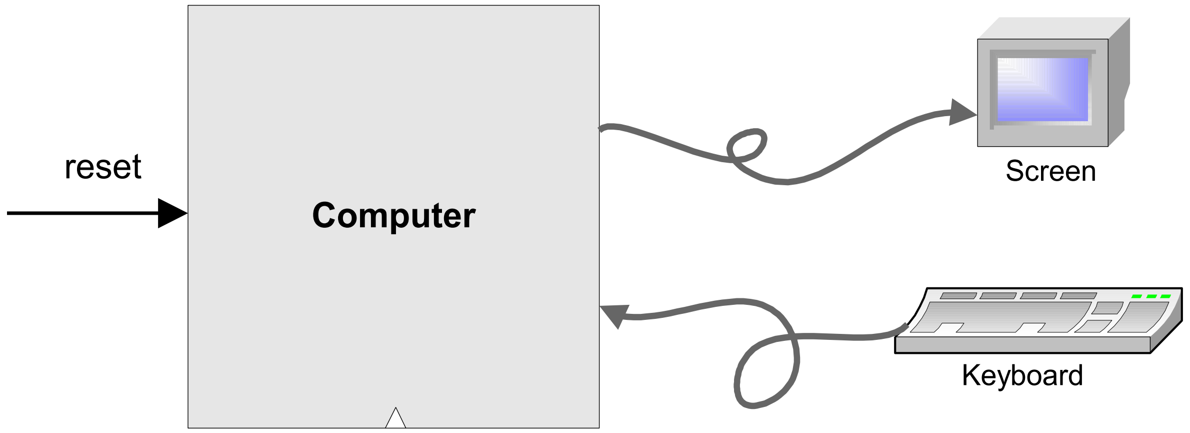


These bits are used to help drive conditional updates of the PC (conditional jumps)

Lecture / construction plan

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

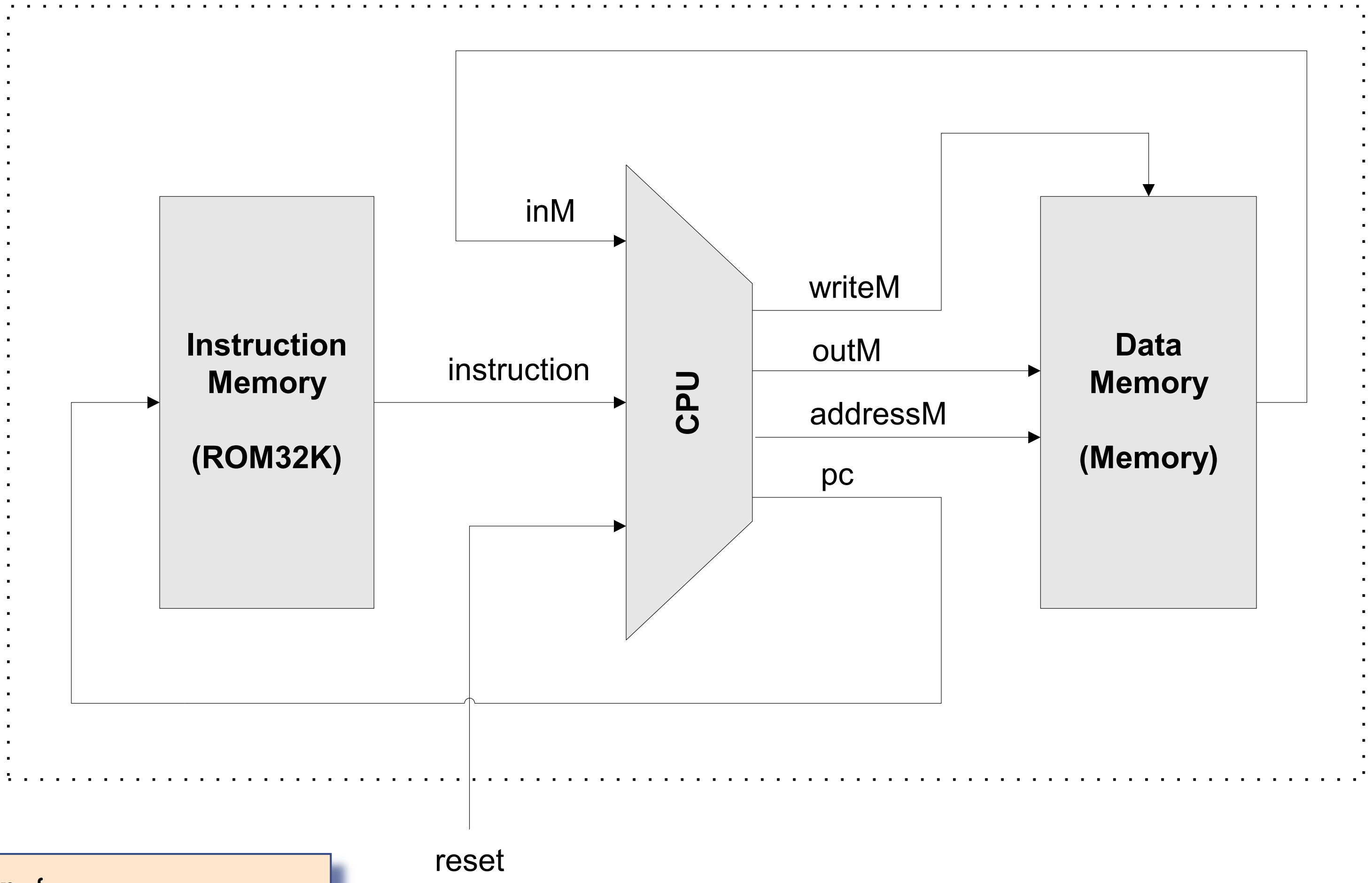
Computer-on-a-chip interface



```
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
}
```

<u>Implementation:</u>

Simple, the chip-parts do all the hard work.

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 …

Next..

- We're done with hardware for now!
- From now we talk about software
  - and how it interfaces with Hardware
- In the meantime
  - Keep working on assignment 2 and
  - Watch out for more quizzes.

Next week

- There is a lecture on Monday!
- There is a tutorial next week.
- You should read “Chapter 5” from the forums and continue working on assignment 2.
- Any questions? Ask on the forum or right now!

Next lecture

- You should read “Chapter 6” from the forums and keep working on at Assignment 2.
- Any questions? Ask on the forum or right now!