A-instruction:

Symbolic: `@`*xxx* (*xxx* is a decimal value ranging from 0 to 32767, or a symbol bound to such a decimal value)

Binary: `0` *vvvvvvvvvvvvvvv* (*vv ... v* = 15-bit value of *xxx*)

C-instruction:

Symbolic: *dest* = *comp* ; *jump* (*comp* is mandatory. If *dest* is empty, the = is omitted; If *jump* is empty, the ; is omitted)

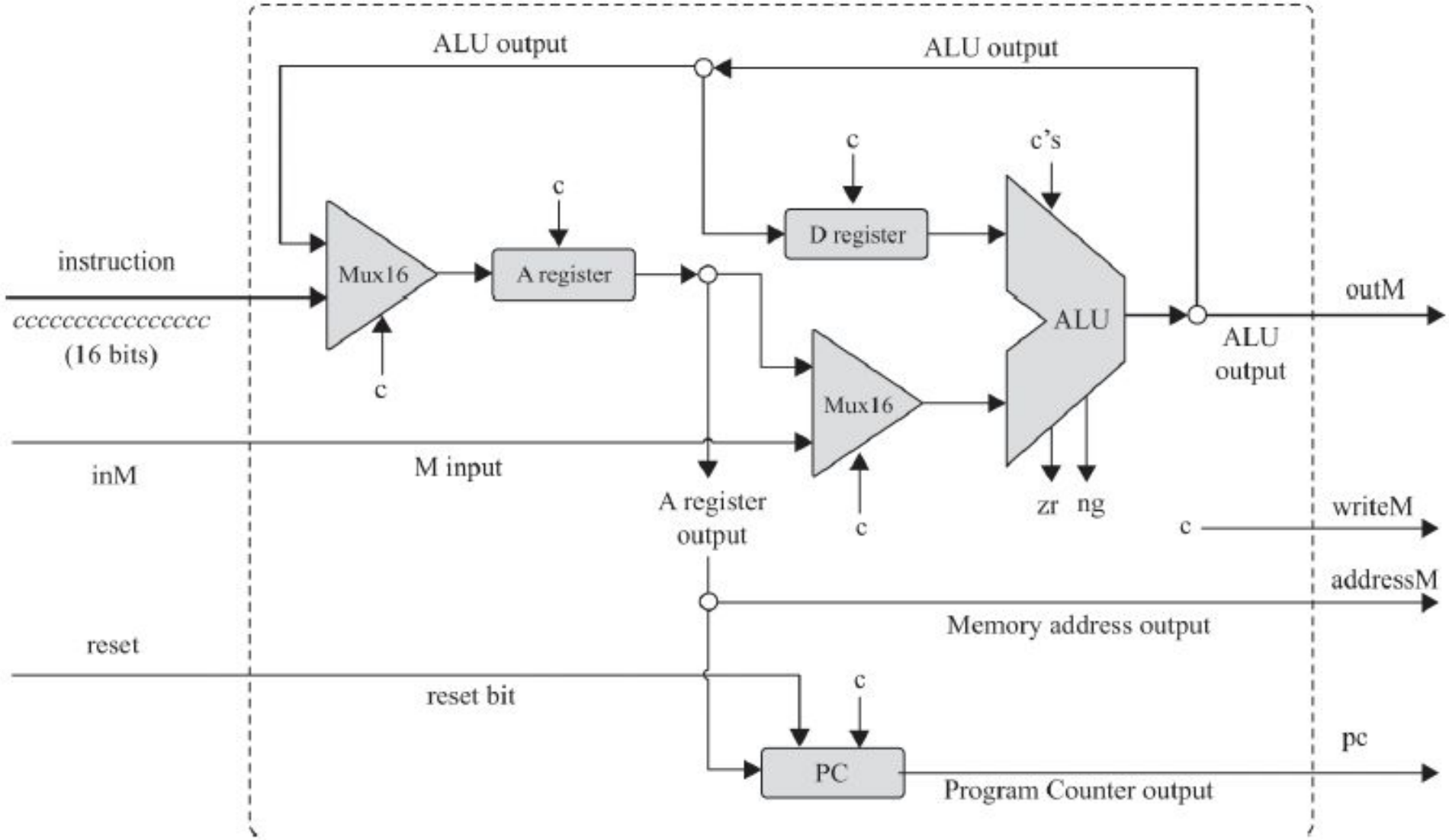
Binary: `111`*accccccdddjjj*

comp		*c*	*c*	*c*	*c*	*c*	*c*
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	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1

a == 0 *a* == 1

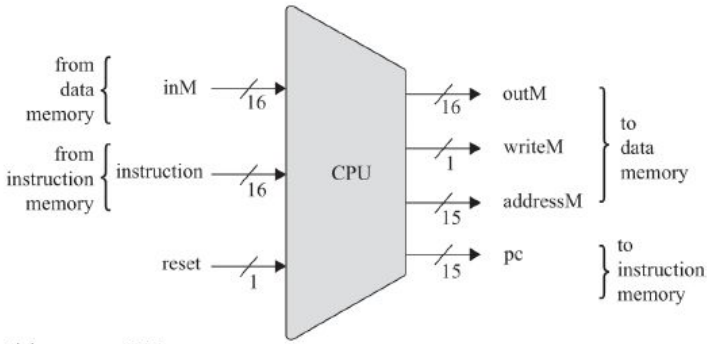
dest	*d*	*d*	*d*	Effect: store *comp* in:
null	0	0	0	the value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register (reg)
DM	0	1	1	D reg and RAM[A]
A	1	0	0	A reg
AM	1	0	1	A reg and RAM[A]
AD	1	1	0	A reg and D reg
ADM	1	1	1	A reg, D reg, and RAM[A]

jump	*j*	*j*	*j*	Effect:
null	0	0	0	no jump
JGT	0	0	1	if *comp* > 0 jump
JEQ	0	1	0	if *comp* = 0 jump
JGE	0	1	1	if *comp* ≥ 0 jump
JLT	1	0	0	if *comp* < 0 jump
JNE	1	0	1	if *comp* ≠ 0 jump
JLE	1	1	0	if *comp* ≤ 0 jump
JMP	1	1	1	unconditional jump


ALU output
ALU output
c
c's
c
D register
instruction
Mux16
A register
outM
cccccccccccccccc
(16 bits)
ALU
c
ALU
output
Mux16
inM
M input
A register
output
c
zr
ng
c
writeM
addressM
Memory address output
reset
reset bit
c
pc
PC
Program Counter output

5.2.2 Central Processing Unit

The Hack CPU interface is shown in figure 5.2. The CPU is designed to execute 16-bit instructions according to the Hack machine language specification presented in chapter 4. The CPU consists of an ALU, two registers named A and D, and a program counter named PC (these internal chip-parts are not seen outside the CPU). The CPU expects to be connected to an instruction memory, from which it fetches instructions for execution, and to a data memory, from which it can read, and into which it can write, data values. The inM input and the outM output hold the values referred to as M in the *C*-instruction syntax. The addressM output holds the address at which outM should be written.




```
Chip name: CPU
Input:
   instruction[16]   // Instruction to execute
   inM[16]           // The instruction's M input (contents of RAM[A])
   reset             // Signals whether to restart the program (if reset==1)
                     // or continue executing the program (if reset==0).
Output:
   outM[16]          // Written to RAM[addressM], the instruction's M output
   addressM[15]      // At which address to write?
   writeM            // Write to the memory?
   pc[15]            // Address of next instruction
```

Figure 5.2 The Hack Central Processing Unit (CPU) interface.

If the instruction input is an *A*-instruction, the CPU loads the 16-bit instruction value into the A register. If instruction is a *C*-instruction, then (i) the CPU causes the ALU to perform the computation specified by the instruction and (ii) the CPU causes this value to be stored in the subset of {A,D,M} destination registers specified by the instruction. If one of the destination registers is M, the CPU's outM output is set to the ALU output, and the CPU's writeM output is set to 1. Otherwise, writeM is set to 0, and any value may appear in outM.

As long as the reset input is 0, the CPU uses the ALU output and the jump bits of the current instruction to decide which instruction to fetch next. If reset is 1, the CPU sets pc to 0. Later in the chapter we'll connect the CPU's pc output to the address input of the instruction memory chip, causing the latter to emit the next instruction. This configuration will realize the fetch step of the fetch-execute cycle.

The CPU's outM and writeM outputs are realized by *combinational* logic; thus, they are affected instantaneously by the instruction's execution. The addressM and pc outputs are *clocked*: although they are affected by the instruction's execution, they commit to their new values only in the next time step.

5.2.3 Instruction Memory

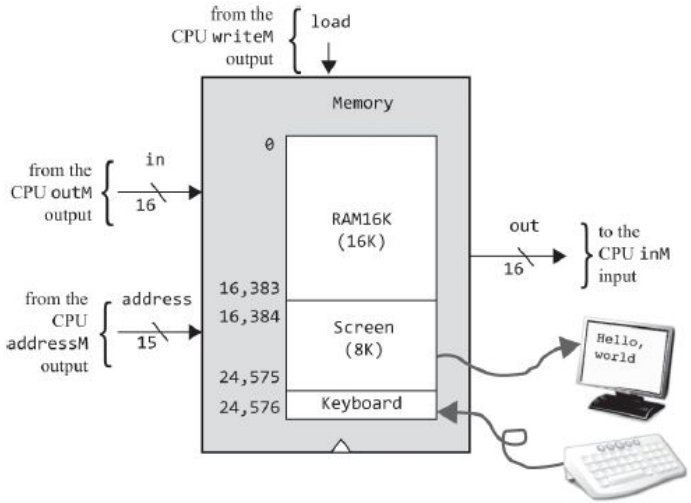
The Hack *instruction memory*, called ROM32K, is specified in figure 5.3.


```
Chip name: ROM32K
Input:     address[15]
Output:    out[16]
Function:  Emits the 16-bit value stored in the address selected by the
           address input. It is assumed that the chip is preloaded
           with a program written in the Hack machine language.
```

Figure 5.3 The Hack instruction memory interface.

5.2.5 Data Memory

The overall address space of the Hack *data memory* is realized by a chip named Memory. This chip is essentially a package of three 16-bit chip-parts: RAM16K (a RAM chip of 16K registers, serving as a general-purpose data store), Screen (a built-in RAM chip of 8K registers, acting as the screen memory map), and Keyboard (a built-in register chip, acting as the keyboard memory map). The complete specification is given in figure 5.6.




```
Chip name: Memory          // Data memory
Input:     in[16]          // What to write
           address[15]     // Where to read/write
           load            // Write-enable bit
Output:    out[16]         // Value at the given address
Function:
   The complete address space of the Hack computer's data memory.
   Only the top 16K+8K+1 words of the address space are used.
   Accessing an address in the range 0 - 16383 results in accessing RAM16K;
   Accessing an address in the range 16384 - 24575 results in accessing Screen;
   Accessing the address 24576 results in accessing Keyboard;
   Accessing any other address is invalid.
```

Figure 5.6 The Hack data memory interface. Note that the decimal values 16384 and 24576 are 4000 and 6000 in hexadecimal.

5.3.3 Computer

We have reached the end of our hardware journey. The topmost Computer chip can be realized using three chip-parts: the CPU, the data Memory chip, and the instruction memory chip, ROM32K. Figure 5.9 gives the details.

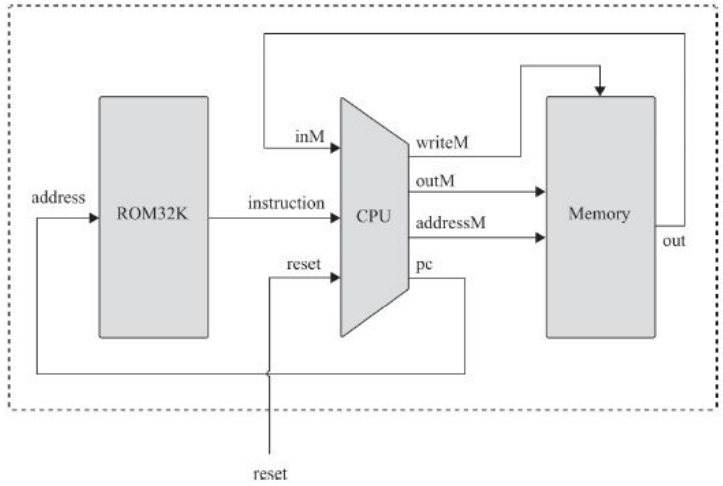


Figure 5.9 Proposed implementation of Computer, the topmost chip in the Hack platform.

The Computer implementation is designed to realize the following fetch-execute cycle: When the user asserts the reset input, the CPU's pc output emits 0, causing the instruction memory (ROM32K) to emit the first instruction in the program. The instruction will be executed by the CPU, and this execution may involve reading or writing a data memory register.