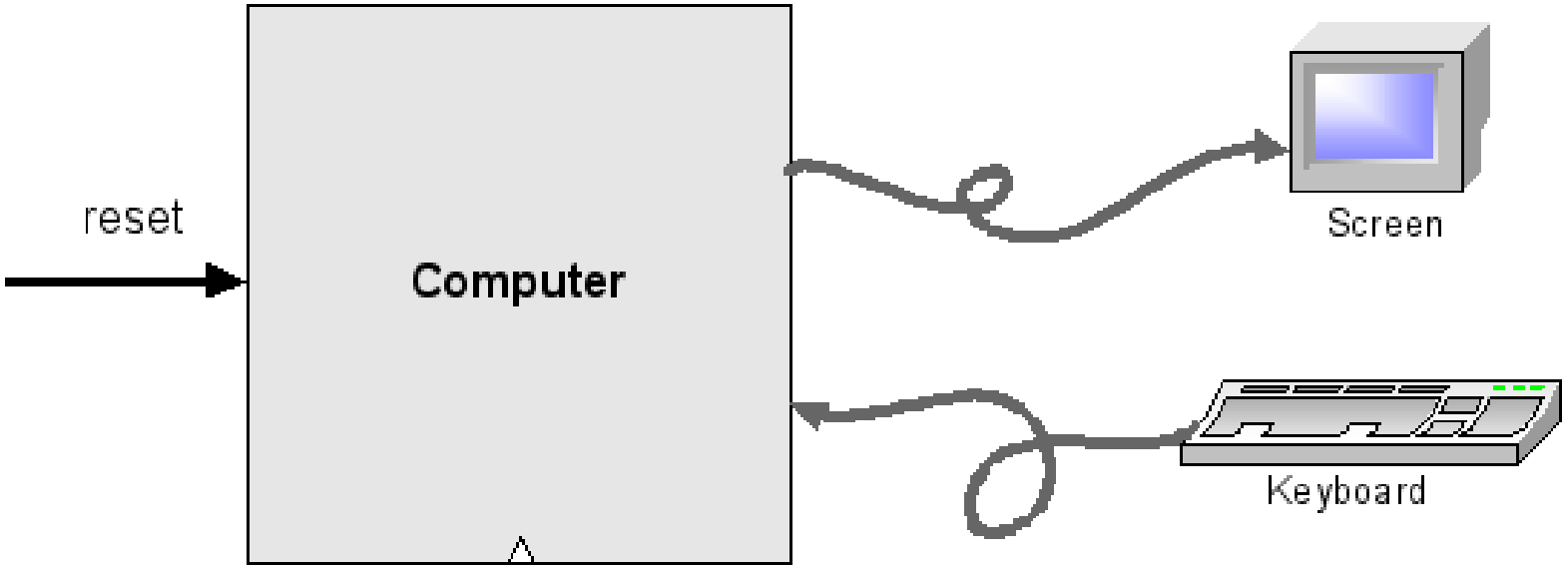
Realizzazione del Computer Hack



Prof. Ivan Lanese

A che punto siamo

Circuiti logici

- Nand
- Not
- And
- Or
- Xor
- Mux
- Dmux
- Not16
- And16
- Or16
- Mux16
- Or8Way
- Mux4Way16
- Mux8Way16
- DMux4Way
- DMux8Way

fatto

Circuiti aritmetici

- HalfAdder
- FullAdder
- Add16
- Inc16
- ALU

fatto

Circuiti sequenziali

- DFF
- Bit
- Register
- RAM8
- RAM64
- RAM512
- RAM4K
- RAM16K
- PC

fatto

Realizzazione del Computer

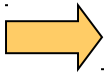
- Memory
- CPU
- Computer

DA FARE

Il computer Hack

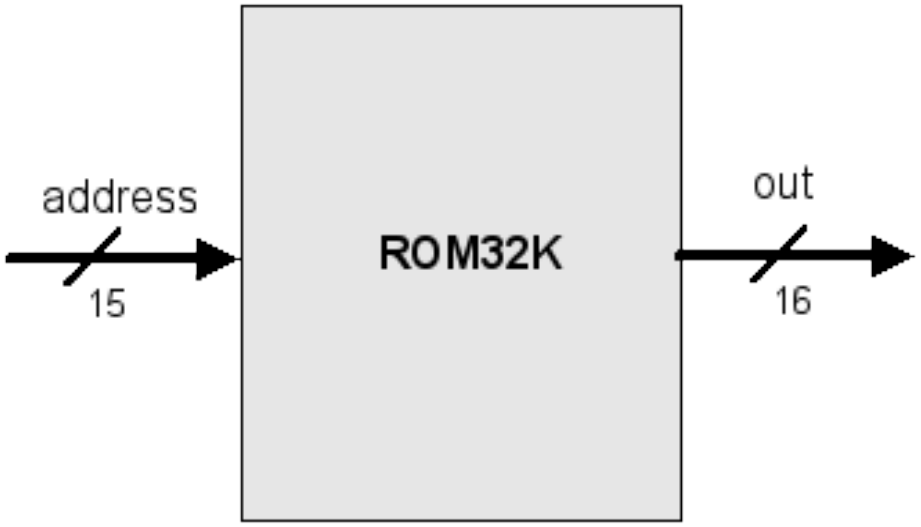
- Una architettura di tipo Von Neumann a 16-bit
  - però con separazione fra memoria dati e memoria programma (per poter caricare contemporaneamente dati e istruzioni)
- Schermo bianco e nero composto da 256 righe e 512 colonne
- Tastiera
- Progettato per eseguire programmi scritti nel linguaggio macchina Hack (già studiato)
- Di facile realizzazione a partire dall'insieme dei circuiti logici già costruiti fino ad ora

Piano di costruzione del computer Hack



- Memoria istruzioni
- Memoria dati:
  - Dati
  - Schermo
  - Tastiera
- CPU
- Computer

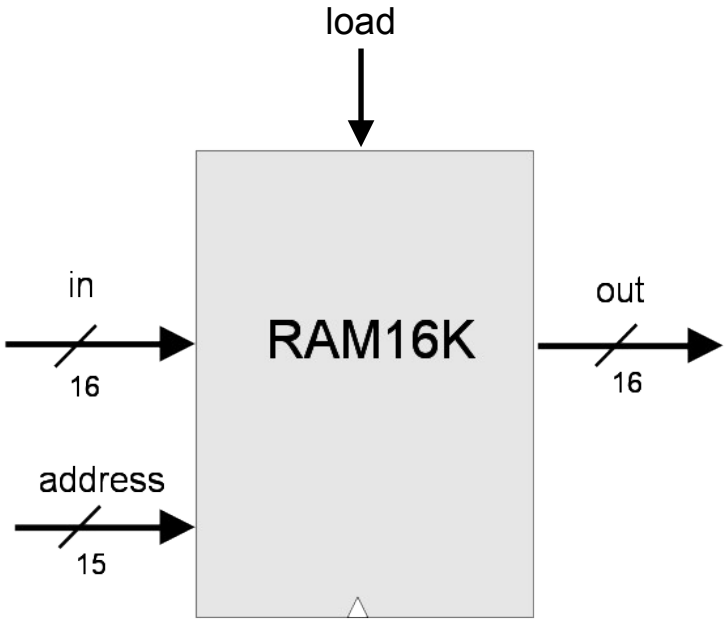
Memoria per le istruzioni



Funzione:

- La ROM (builtin chip) è precaricata con un programma scritto nel linguaggio macchina Hack
- La ROM emette un numero a 16-bit: `out = ROM32K[address]`
- Tale numero è interpretato come la prossima istruzione da eseguire

Memoria per i dati



Logica di lettura/scrittura:

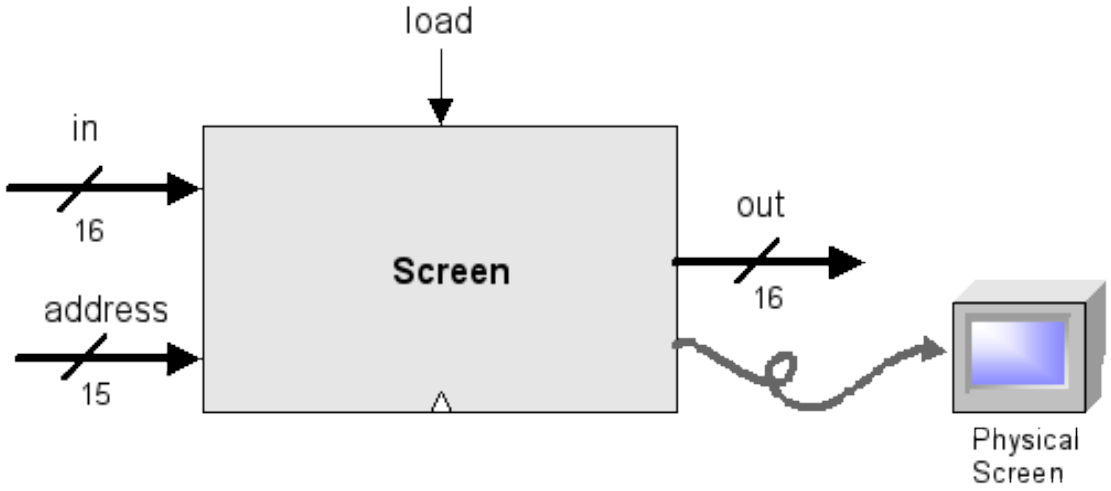
Per leggere RAM[k]: metti k sul bus address
leggi out

Per scrivere RAM[k]=x: metti k sul bus address
metti x sul bus in
metti 1 su load
attendi un ciclo di clock

Piano di costruzione del computer Hack

- ✓ Memoria istruzioni
- Memoria dati:
  - ✓ Dati
  - ⇨ Schermo
  - Tastiera
- CPU
- Computer

Schermo



Lo schermo ha una funzionalità tipo RAM:

- Logica di lettura:

  `out = Screen[address]`

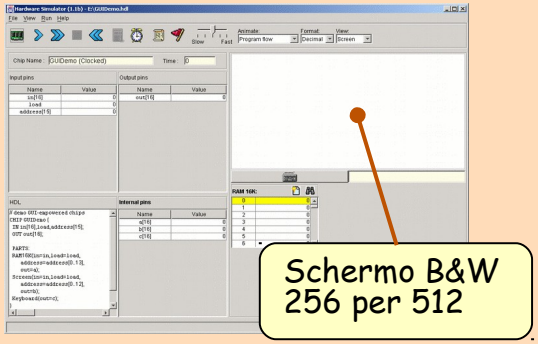
- Logica di scrittura:

  `if load then Screen[address] = in`

Legame fra Screen e monitor

Esiste un legame fra ogni pixel del monitor e dei corrispondenti bit all'interno di Screen

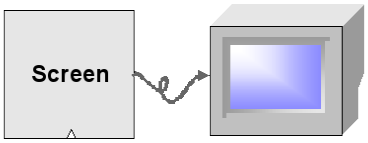
Schermo simulato:



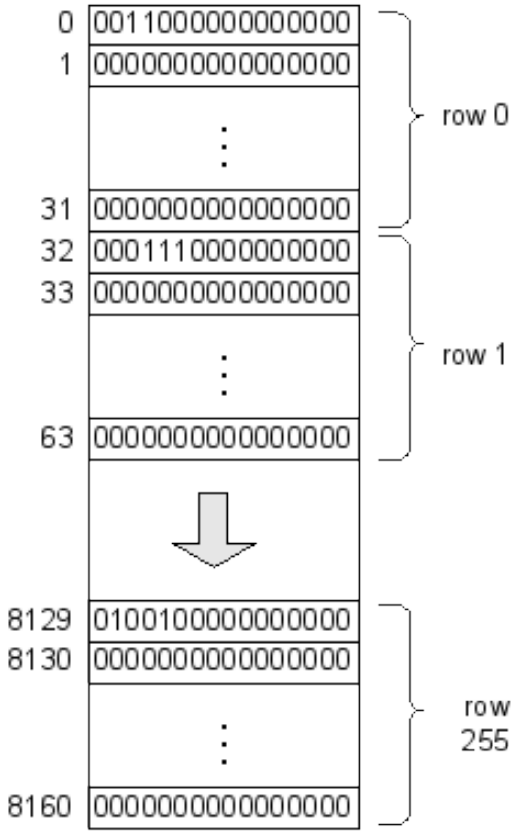
Quando il chip built-in `Screen.hdl` viene caricato nell'hardware simulator, viene aperta una finestra che simula lo schermo; viene frequentemente "refreshed" per rispettare la relativa mappa in memoria

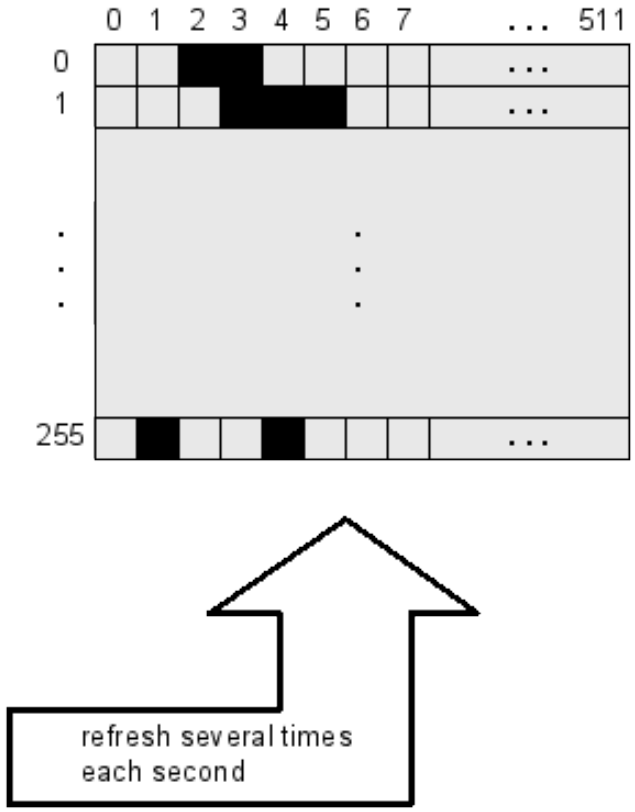
Mappa dello Schermo sulla memoria “Screen”

“Screen” contiene 8K parole da 16 bit



un bit a 1 indica che il corrispondente pixel deve essere nero,
0 indica invece bianco





Impostare il pixel `(row,col)` dello schermo per essere nero o bianco:

- Imposta il bit `col%16` della parola alla locazione `Screen[row*32+col/16]` a 1 (nero) o `0` (bianco)
  (`col/16` è la divisione intera)

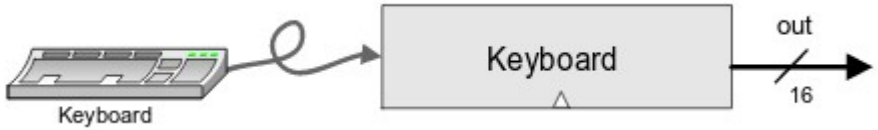
Esempio di uso dello schermo in linguaggio Hack

- Disegna sullo schermo un rettangolo largo 16 pixel e alto R[0] pixel

```
   @0
   D=M     //D=RAM[0]
   @INFINITE_LOOP
   D;JLE  // if D>0 then
   @counter
   M=D     //  counter=D
   @SCREEN
   D=A
   @address
   M=D     //  address=SCREEN
(LOOP)     //  repeat
   @address
   A=M     //   A=address
   M=-1    //   M[A]=111...1
   @address
   D=M     //   D=address
   @32
   D=D+A   //   D += 32
```

```
   @address
   M=D     //   address=D
   @counter
   MD=M-1 //   counter--
   @LOOP
   D;JGT  //  until counter<=0
(INFINITE_LOOP)
   @INFINITE_LOOP
   0;JMP
```

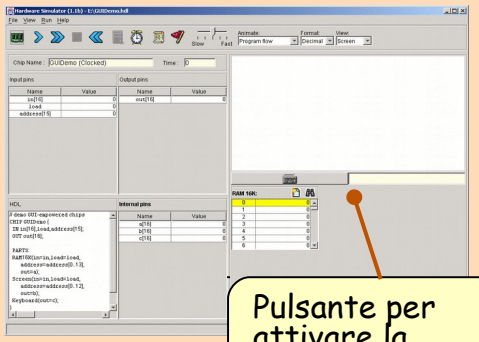
Tastiera



Chip "Keyboard": un unico registro da 16-bit

Out: codifica ASCII (estesa a 16-bit) del tasto correntemente premuto, oppure 0 se non si preme alcun tasto (esistono alcuni tasti speciali, vedi sotto)

Tastiera simulata:



Esiste un chip built-in `Keyboard.hdl`. Quando viene caricato nel simulatore, si collega alla tastiera fisica e collega il codice del tasto correntemente premuto alla mappa in memoria implementata dal chip `Keyboard`

Tasti speciali:

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

Come leggere dalla tastiera:

- Leggi il contenuto del chip `Keyboard`

Esempio di uso della tastiera in linguaggio Hack

- Colora lo schermo di nero se si preme un tasto

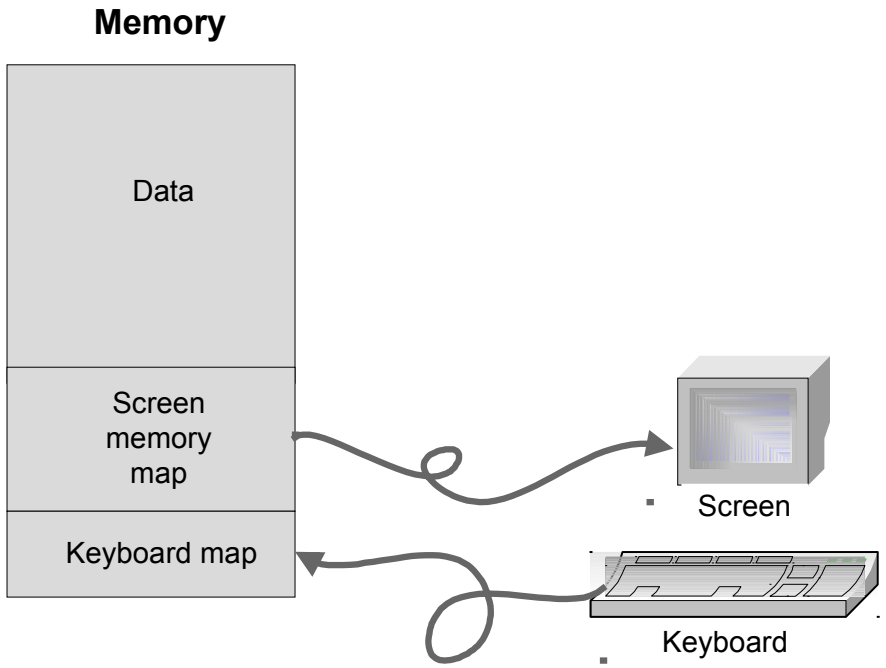
```
(START) // while(true)
   @SCREEN
   D=A
   @i
   M=D   //   i = SCREEN
(LOOP)   //   while(true)
   @i
   D=M
   @24575    //(last map word)
   D=A-D   //    D=24575-i
   @START
   D;JLT   //   if D<0 exit
   @KBD
   D=M
   @BLACKEN
   D;JNE //jmp if key pressed
```

```
   @i      //     M[i]=000...0
   D=M     //           (white)
   A=D
   M=0
   @CONTINUE
   0;JMP
(BLACKEN)//    else
   @i      //     M[i]=111...1
   D=M     //           (black)
   A=D
   M=-1
(CONTINUE)
   @i      //   endif
   M=M+1   //   i++
   @LOOP
   0;JMP   //   end_while
           // end_while
```

Piano di costruzione del computer Hack

- ✓ Memoria istruzioni
- ⇨ Memoria dati:
  - ✓ Dati
  - ✓ Schermo
  - ✓ Tastiera
- CPU
- Computer

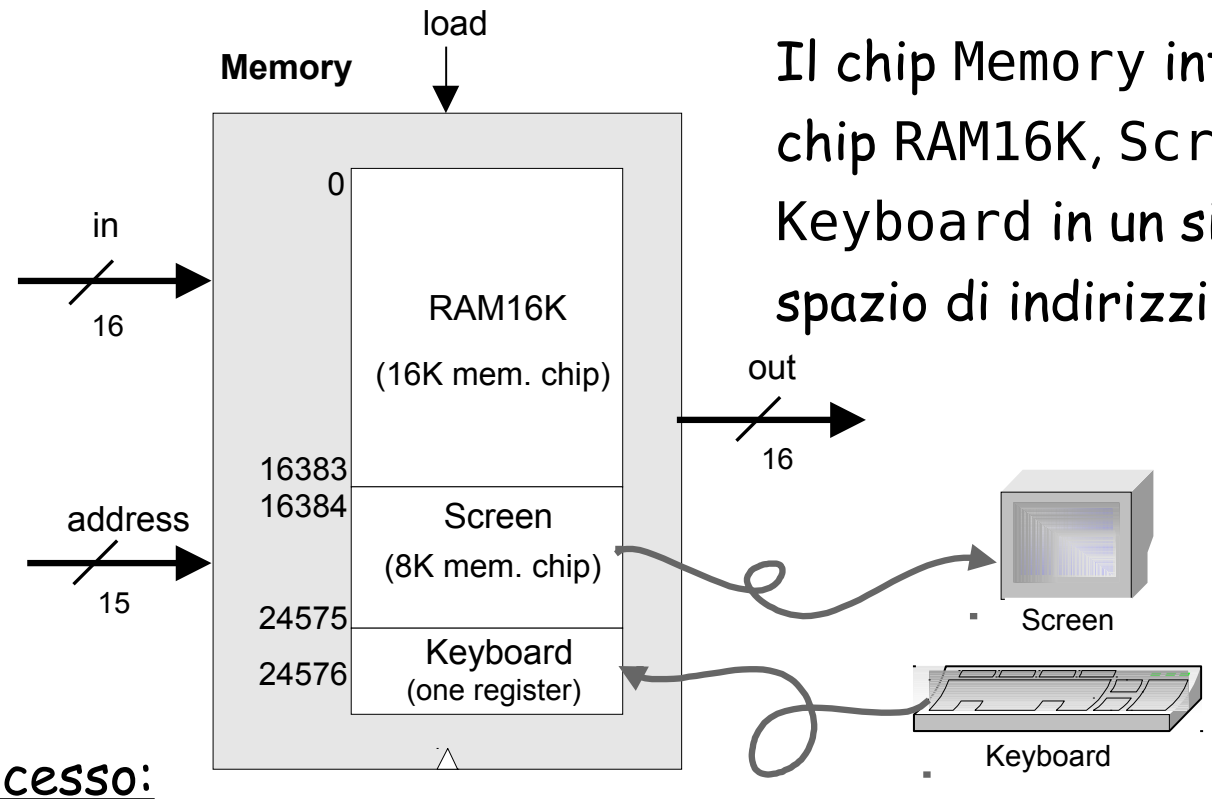
Struttura della memoria dati



Uso della memoria:

- La prima parte "Data" serve per memorizzare dati usati dal programma in esecuzione (usa le prime 16K parole di memoria)
- Per scrivere/leggere sulla mappa dello schermo nella parte "Screen" (usa le successive 8K parole di memoria)
- Per leggere il tasto premuto su tastiera (usa la successiva parola)

Memoria dati: implementazione



Il chip Memory integra i tre chip RAM16K, Screen, e Keyboard in un singolo spazio di indirizzi contigui

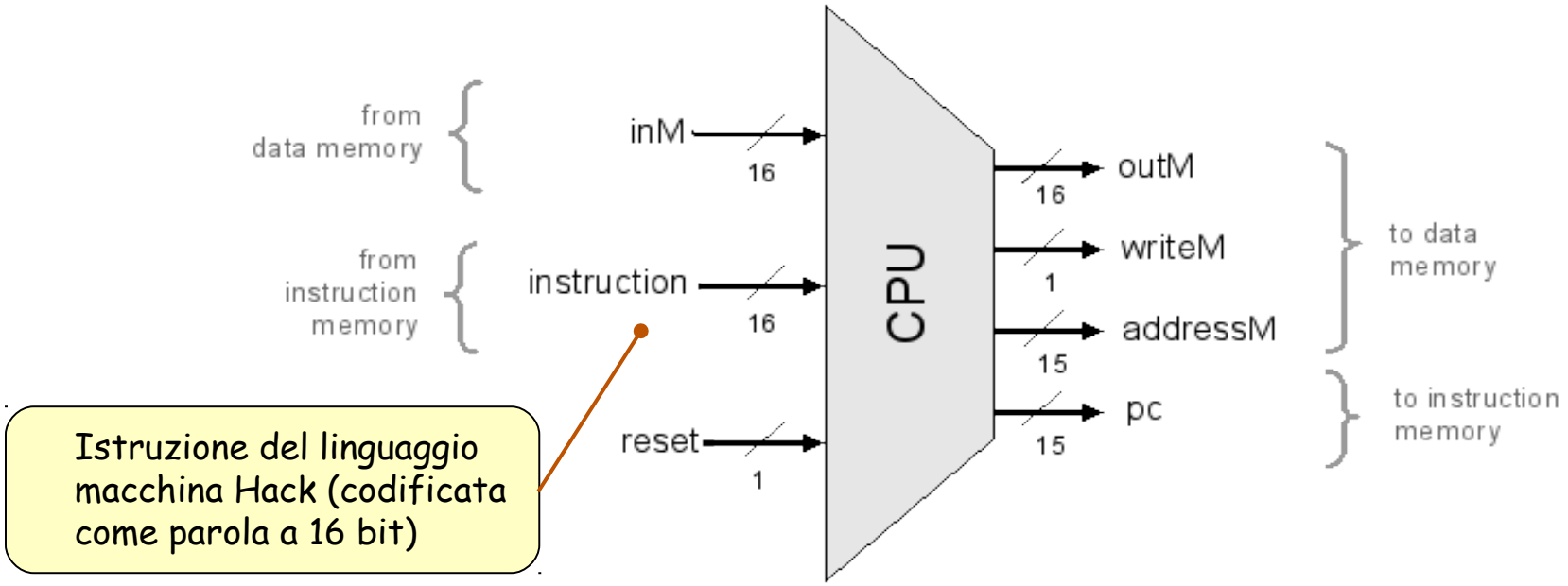
Logica di accesso:

- Dall'indirizzo 0 a 16383 si accede al chip RAM16K
- Dall'indirizzo 16384 a 24575 si accede al chip built-in Screen
- L'indirizzo 24576 accede al chip built-in Keyboard
- Ogni altro indirizzo non è valido

Piano di costruzione del computer Hack

- ✓ Memoria istruzioni
- ✓ Memoria dati:
  - ✓ Dati
  - ✓ Schermo
  - ✓ Tastiera
- ⇨ CPU
- Computer

Comportamento della CPU



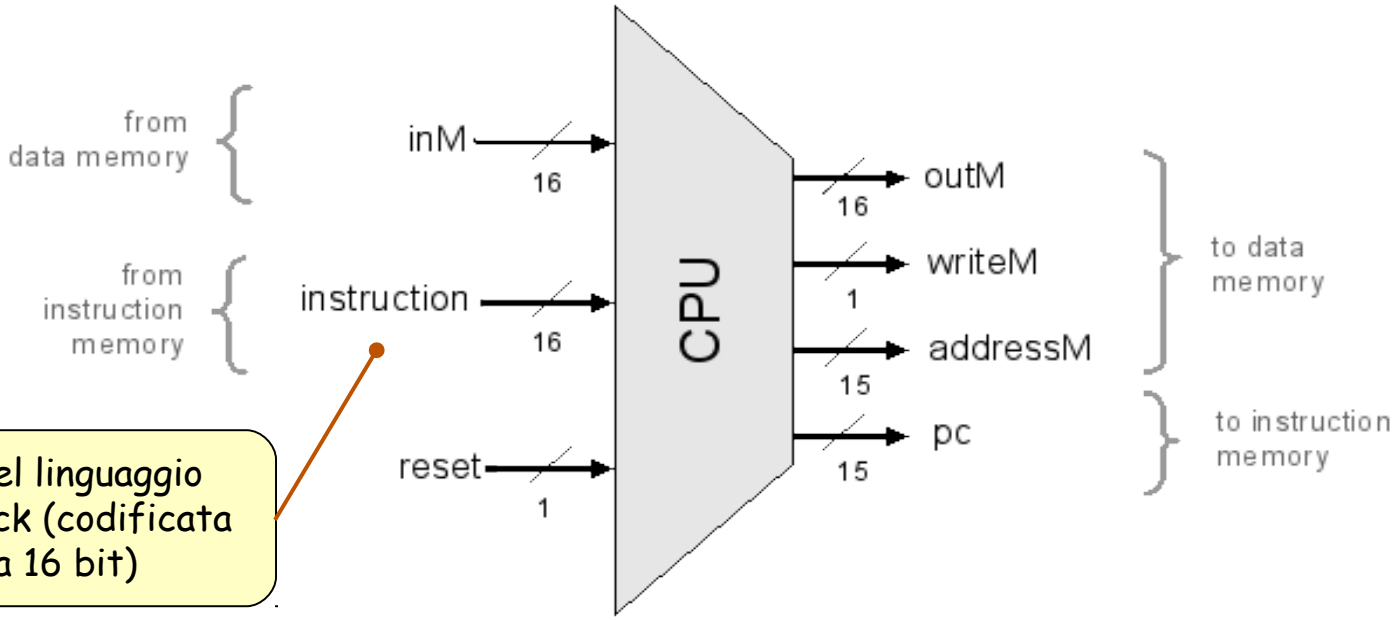
Componenti interni della CPU: ALU e 3 registri: A, D, PC

Comportamento della CPU:

La CPU esegue `instruction` seguendo la specifica del linguaggio Hack:

- I valori D e A, se presenti nell'istruzione, sono letti/scritti dai rispettivi registri
- Il valore M, se presente nella parte destra dell'istruzione, viene letto da `inM`
- Se la parte sinistra dell'istruzione contiene M, l'output della ALU viene posto su `outM`, il valore di A viene posto su `addressM`, e `writeM` viene settato

Comportamento della CPU



Istruzione del linguaggio macchina Hack (codificata come parola a 16 bit)

Politiche di caricamento delle istruzioni:

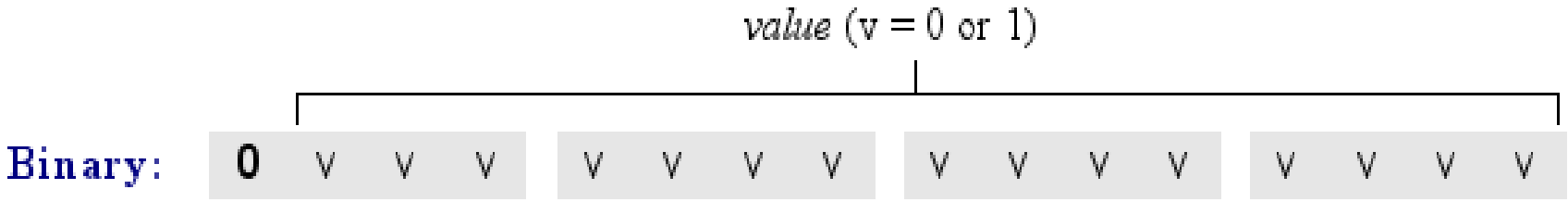
`instruction` può includere un salto (jump); nel caso di salto condizionatola condizione dipende dai due bit di controllo della ALU, indicanti se l'output è zero o negativo

Se reset==0: la CPU controlla se c'è il jump. In caso di jump si carica il contenuto di A in PC; altrimenti, si incrementa di 1 il registro PC

Se reset==1: si mette 0 in PC (si fa ripartire il computer)

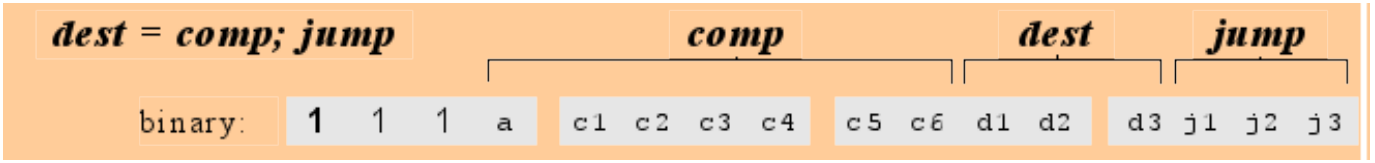
Codifica delle A-instruction

Symbolic: `@value` // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.



Il primo bit a 0 indica che si tratta di una A-instruction:
il numero naturale "value" viene codificato in binario dai successivi 15 bit
(tale codifica corrisponde alla codifica dei numeri non negativi che si possono codificare in complemento a 2 avendo a disposizione 16 bit)

Codifica delle C-instruction



(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

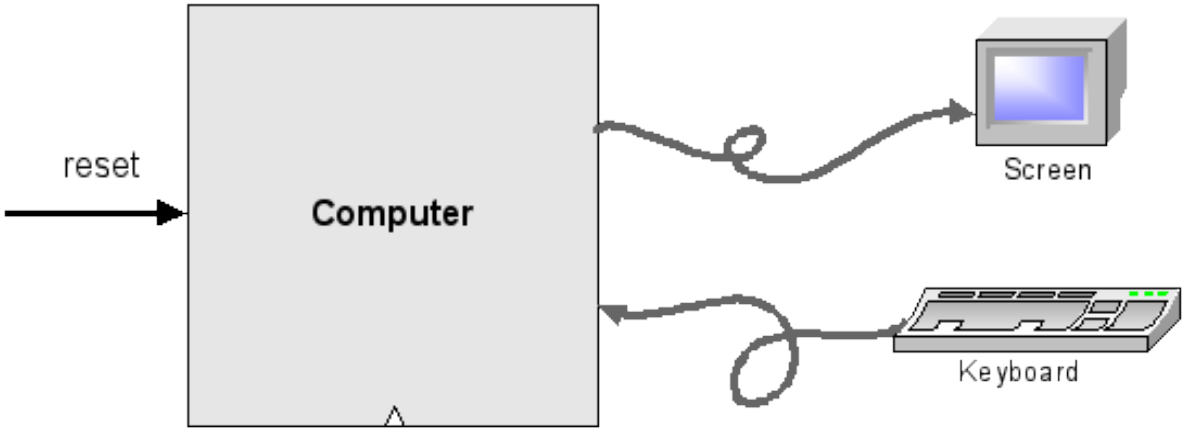
d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
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* ≠ 0 jump
1	1	0	JLE	If *out* ≤ 0 jump
1	1	1	JMP	Jump

Piano di costruzione del computer Hack

- ✓ Memoria istruzioni
- ✓ Memoria dati:
  - ✓ Dati
  - ✓ Schermo
  - ✓ Tastiera
- ✓ CPU
- → Computer

Descrizione dell'interfaccia del "Computer"



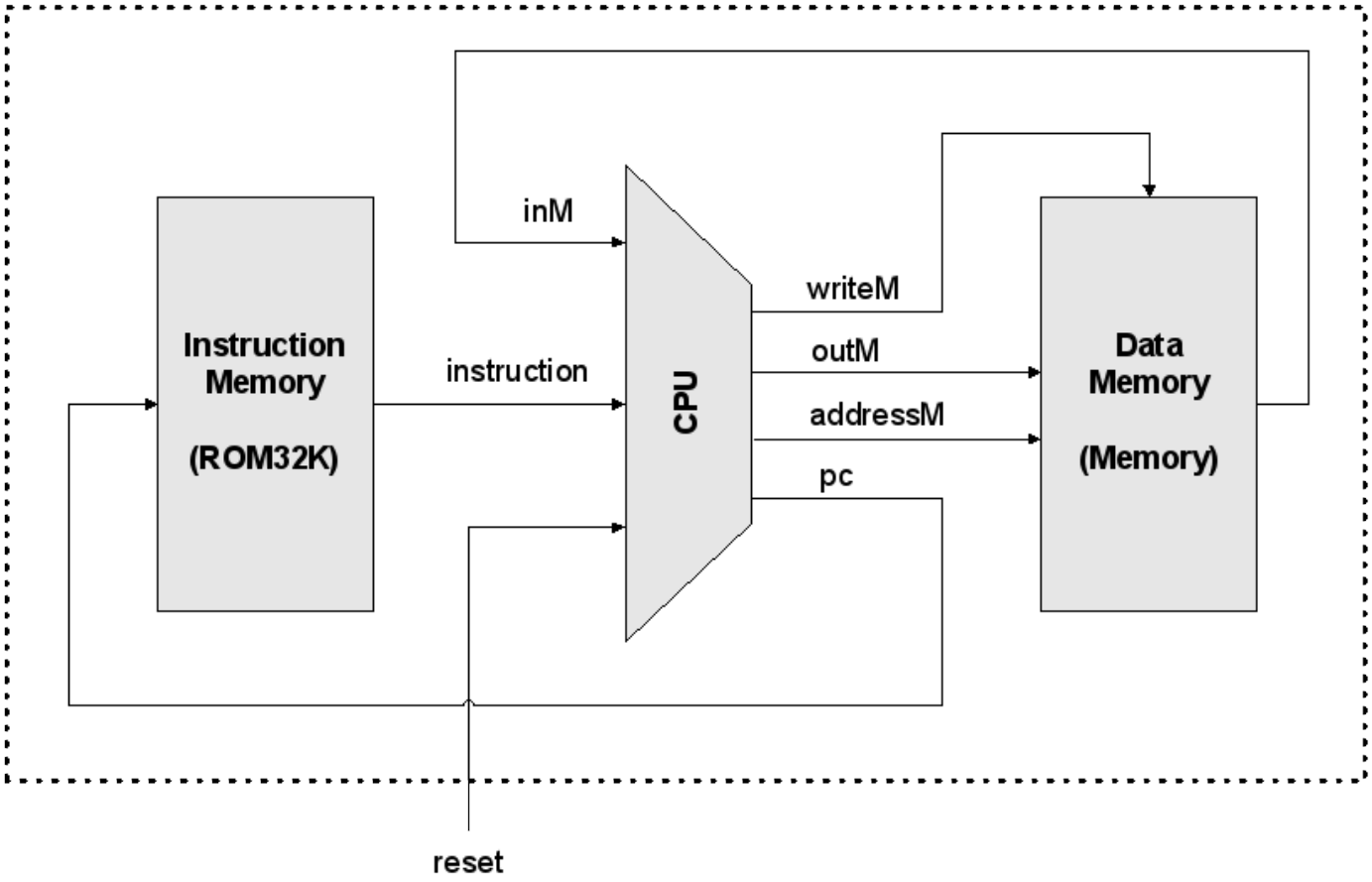
Chip Name: Computer // Topmost chip in the Hack platform

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.

Implementazione del “Computer”




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

Implementazione:

semplice combinazione dei chip già descritti