Architettura degli Elaboratori
Come implementare il computer HACK

Andrea Malvezzi

05 novembre, 2024

Contents

1 Premesse

Il computer HACK sfrutta l'architettura Von Neumann a 16-bit, con una feature di quella Harvard: la memoria dati viene separata da quella programma, in modo da poter caricare contemporaneamente dati e istruzioni.
Questo viene reso possibile dal bus dati che, mentre nelle architetture moderne svolge due compiti distinti e quindi non eseguibili in parallelo, qui ne svolge solamente uno. Conseguentemente per eseguire un'istruzione basta un ciclo di clock.

2 La memoria

2.1 La ROM

La ROM corrisponde ad un chip built-in che dato un address 15-bit codifica in binario un'istruzione da eseguire, quindi: out = ROM32K[$address$].

2.2 Chip memory

Il chip memory è composto da un totale di 3 componenti, per un totale di 24K indirizzi:

- la RAM da 16K;
- il chip **Screen** da 8K, usato per mappare lo schermo;
- un singolo registro **Keyboard** per leggere il tasto premuto;

Tuttavia, avendo in ingresso un address da 15 bit da convertire in binario, si sprecano molti indirizzi così facendo: $2^{15} = 32767$ meno i 24576 occupati dal chip memory, per un totale di 8191 indirizzi sprecati. Cosa accade a questi?

2.2.1 Il chip Screen

All'interno del chip Screen si ha una corrispondenza diretta tra il singolo bit e un pixel sullo schermo. In base al valore di questi bit, lo schermo viene costsantemente refreshato colorando di bianco (0) o di nero (1) ogni singolo pixel.
Per impostare il pixel in posizione (row, col) dello schermo ad un certo colore,

occorrerà quindi settare il bit *col*%16 della word all'indirizzo Screen[row*32 + col/16] a 1 oppure a 0.
Ma come mai questi numeri? Anzitutto, occorre specificare che nel computer HACK lo schermo è composto da 256 righe e 512 colonne.
Questo significa che in una singola riga ci saranno 512 caratteri, raggruppati in parole (che in informatica hanno spesso una grandezza di 16 bit), da cui si evince che in una singola riga si avranno 32 parole (512/16 = 32). Ed ecco spiegato il **row*32**.
Lo stesso ragionamento va applicato alle righe: avendo 256 righe, si avranno 16 (256/16 = 16) parole per colonna, da cui **col/16**.

2.2.2 Esempio di codice per disegnare un rettangolo largo 16 pixel e alto MEM[0] pixel

```
    @0              // A=0
    D=M             // D=RAM[0]

    @INFINITE_LOOP  // Preparo il salto
    D; JGE          // Se D>0

    @counter        // Variabile counter per iterare
    M=D             // RAM[counter]=RAM[0]

    @SCREEN         // Predefinita dal linguaggio
    D=A             // D=Indirizzo Screen
    @address        // Variabile address
    M=D             // RAM[address]=Indirizzo Screen

    (LOOP)
        @address    // Variabile address
        A=M         // A=Indirizzo Screen
        M=-1        // MEM[Indirizzo Screen]=-1

        @32         // A=32
        D=A         // D=32
        @address    // Variabile address
        M=M+D       // Address+=32

        @counter
```

```
26          MD=M-1      // Counter--, D per check sul jump
27
28          @LOOP
29          D;JGT       // Torno all'inizio
30
31      (END)           // Ciclo dummy finale
32          @END
33          0;JMP
```

2.2.3 Il chip Keyboard

Come annunciato precedentemente questo chip è composto da un solo registro e fornisce in output la codifica ASCII estesa del tasto premuto, oppure 0 se non si preme alcun tasto.
Il registro è read-only e vi sono alcuni tasti con codifiche particolari, come lo spazio.

3 Comportamento della CPU

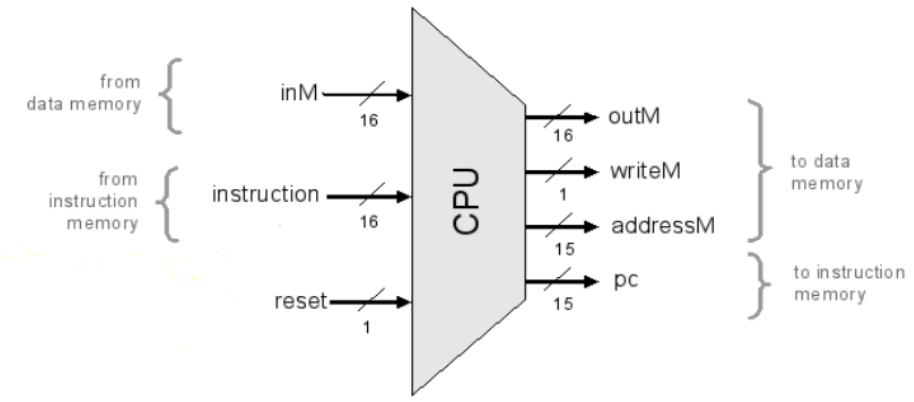

Figure 1: Schema logico CPU nell'architettura HACK.

La CPU è composta da 4 componenti: l'ALU e 3 registri, quali A, D, PC. Questa esegue istruzioni seguendo le specifiche del linguaggio HACK (nel nostro caso). Ovvero:

- i valori D ed A, se presenti, sono letti e/o scritti nei registri A ed M;
- il valore M, se presente nella right side dell'istruzione da eseguire, viene LETTO da inM (input-M);
- se invece il valore M risulta presente nella left side dell'istruzione da eseguire, allora l'output verrà scritto in outM, A prenderà il valore di addressM e writeM verrà alzato ad 1.

4 Codifica A-Instruction

In HACK le A-instruction sono codificate nella maniera seguente: il bit di segno (o modulo) è sempre pari a 0, mentre i 15 valori a seguire saranno cifre binarie.

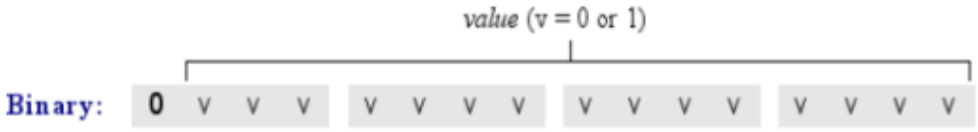

Figure 2: Codifica delle A-instruction nell'architettura HACK.

Tramite questo approcio si possono codificare tutti i valori non negativi codificabili in complemento a 2 con 16 bit (si ha effettivamente un bit in meno, quello modulo).

5 Codifica C-instruction

Nell'architettura HACK le C-instruction sono codificate nella maniera seguente: il modulo e le due cifre subito dopo di esso sono **sempre** pari ad 1, ed i valori seguenti sono posti ad 1 solamente quando alcune caratteristiche vengono rispettate.

Figure 3: Codifica delle C-instruction nell'architettura HACK.

E come possibile vedere dalla figura, il corpo effettivo dell'istruzione è composto da valori impostati a seconda della presenza di alcune direttive. Ma come sono decisi questi valori? tramite la seguente tabella:

(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		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

Figure 4: Ad esempio un'istruzione contenente un JGT avrebbe gli ultimi 3 bit (quelli di *jump*) pari a 001.