# Architettura degli Elaboratori
# ISA del processore HACK

Andrea Malvezzi

05 novembre, 2024

# Contents

# 1 Che cos'è l'ISA

L'ISA è l'interfaccia tra HW e SW. Costituisce il set di istruzioni con cui opera la CPU e di conseguenza, varia di architettura in architettura.

# 2 L'architettura HACK

L'architettura HACK non segue né la filosofia CISC né quella RISC e, data la sua semplicità, in essa ad ogni istruzione eseguita corrisponde un ciclo di clock. Qui si usano una RAM (per contenere i dati di un programma in esecuzione) e una ROM (per contenere il programma *stesso*) a 16 bit.
Si utilizzano prevalentemente tre registri di memoria: **A**, **D** ed **M**.
Il registro M è particolare, in quanto contiene il valore del registro di memoria RAM attualmente puntato da A. Questo si indica con RAM[A].
Oltre a questi due si usa inoltre il **Program Counter**, o **PC**. In esso è contenuto l'indirizzo della prossima istruzione da eseguire e si indica con ROM[PC].

# 3 Tipi di istruzione

Esistono due tipi di istruzioni:

- A-instruction: quelle che lavorano sulla memoria (che caricano valori in A);
- C-instruction: quelle che eseguono un instruzione (sfruttando l'ALU) prelevando gli operandi dai registri A,D ed M.

Le C-instruction solitamente seguono la forma generale dest = comp; jump, dove:

- **dest**: opzionale. Specifica dove memorizzare il risultato dell'operazione;
- **comp**: specifica l'operazione da eseguire;
- **jump**: opzionale. Specifica se, dopo aver eseguito dest = comp occorre fare un salto nel programma alla linea specificata dal valore del registro A (esegue PC = $A$). Funziona effettivamente come un *if* eseguito sempre rispetto allo zero.

## 3.1 Tutte le possibili operazioni

A seguire tutte le possibili operazioni eseguibili nell'architettura HACK.

comp = `0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A, A-D, D&A, D|A, M, !M, -M, M+1, M-1, D+M, D-M, M-D, D&M, D|M`

dest = `M, D, MD, A, AM, AD, AMD` o nullo (in questo caso dest viene omesso)

jump = `JGT, JEQ, JGE, JLT, JNE, JLE, JMP` o nullo (omesso)

Figure 1: Se un'operazione che si desidera eseguire non compare qui, allora occorre cambiare approcio.

### 3.1.1 Esempio di operazione

$$@2$$
$$\mathrm{D} = \mathrm{D} + 1; \mathrm{JLT}$$

Questa riga di codice aumenta di 1 il valore del registro D, poi controlla che questo valore sia strettamente minore di 0. Nel caso in cui questa condizione risultasse verificata, verrà effettuato un JUMP alla riga corrispondente al valore corrente di A, quindi, alla seconda.

# 4 Le etichette

Un'etichetta è un'istruzione composta di due parti: quella di dichiarazione e quella di chiamata.

## 4.1 Dichiarazione di etichetta

Un'etichetta fornisce un modo per non dover scrivere a mano (e quindi commettere errori) ogni A-instruction prima di effettuare un salto. Difatti, l'etichetta prende il valore del numero di riga in cui viene dichiarata, permettendo poi di saltare a quella eventuale riga qualora risultasse necessario.

### 4.1.1 Esempio di dichiarazione di etichetta

(MIA_ETICHETTA)
. . . codice . . .

Qui l'etichetta viene definita prima di un blocco di codice e prende il nome di MIA_ETICHETTA. Questa etichetta avrà quindi ora il valore della riga a cui è stata dichiarata (quella prima del blocco di codice).

## 4.2 Utilizzo dell'etichetta

Questa etichetta contiene ora il numero della riga in cui è stata dichiarata e, in quanto tale, fornisce un modo per saltare a tale riga mediante una istruzione JUMP.

### 4.2.1 Esempio di utilizzo dell'etichetta

@MIA_ETICHETTA
0; JMP

Questo ci permette di assegnare ad A il valore della riga in cui l'etichetta è stata dichiarata, per poi eseguire un salto incondizionato (0 è sempre zero, quindi il JMP verrà sempre effettuato) alla riga corrispondente all'inizio del blocco di codice sottostante all'etichetta.

# 5 Costrutti della programmazione

## 5.1 If-then-else

Si può ricreare il costrutto if-then-else tramite JUMP, A/C-instruction ed etichette.

```
    // Costrutto if-then-else
    D=1             // Dato da testare

    @CASE_FALSE     // A=riga del caso false
    D;JLE           // Se D <= 0, JUMP al valore di A

    // Se salto, questo blocco non viene eseguito
    ...caso true...

    @END            // A=riga fine codice
    0;JMP           // Salto incondizionato

    (CASE_FALSE)    // Dichiaro l'etichetta
    ...caso false...

    (END)           // Dichiaro l'etichetta
    ...fine codice...
```

## 5.2 Ciclo while

Anche il ciclo while si può ricreare mediante JUMP, A/C-instruction ed etichette, seppure in maniera leggermente più complessa.

```
1     // Ciclo while
2     D=10              // Variabile per iterazione
3
4     (LOOP)           // Dichiaro inizio loop
5     D; JLE           // JMP se D<=0
6
7     ...codice...     // Se salto, questo viene saltato
8
9     @LOOP            // Al termine del codice, rifai
10    0;JMP            // Salto incondizionato
11
12    (EXIT)           // Dichiaro uscita loop
13    ...altro codice...
```

# 6 Cose a cui prestare attenzione

## 6.1 Fine dei programmi

Di base i programmi non finiscono, ma continuano ad incrementare il PC fino a che non si esaurisce la ROM. Per evitare questa behaviour, è buona pratica terminare un programma con un ciclo dummy che previene quindi di andare ad eseguire istruzioni scritte nelle celle della ROM successive.

```
// Ciclo dummy
...codice programma...

// Dichiaro un ciclo infinito
(END)       // Dichiaro fine programma
    @END    // A=Riga precedente
    0;JMP   // Salto incondizionato
```

## 6.2 Comparazioni tra valori diversi da 0

Come visto in precedenza, esistono solo comparazioni rispetto lo 0. Per ovviare a questa limitazione, si possono usare la somma e la sottrazione.

```
// A <= 10?
@2      // A=2
D=10    // D=10
D=D-A   // D=D-A, quindi D=10-2

@ELSE   // Preparo A per saltare all'else case
D;JLE   // Se D<=0, JUMP

...caso true...
@END    // Termina programma
0;JMP   // Salto incondizionato

(ELSE)  // Dichiaro else case
    ...caso false...

(END)   // Dichiaro ciclo dummy finale
    @END
    0;JMP
```

# 7 Le variabili

Anche in asm HACK si hanno le variabili, anche se funzionano in modo leggermente diverso. Qui si definiscono come A-instruction e si va a modificare il valore interno al loro registro.

```
1     // Esempio di uso delle variabili
2     @i          // Variabile i
3     M=1         // Registro di i pari ad 1 (RAM[i]=1)
4
5     (LOOP)      // Dichiaro inizio loop
6         @i      // uso i
7         D=M     // D=Valore registro di i
8         @10     // A=10
9         D=D-A   // D=D-A, quindi D=i-10
10
11        @END    // Preparo a terminare il programma
12        D;JGT   // Se D>0, JUMP
13
14        ...codice loop...
15        @i      // Uso i
16        M=M+1   // Aumento di 1 il valore di RAM[i]
17        ...codice loop...
18
19        @LOOP   // Preparo per ricominciare
20        0;JMP   // Salto incondizionato
21
22        (END)   // Dichiaro ciclo dummy finale
23            @END
24            0;JMP
25
26        // La variabile i serve ad iterare nel loop
27        // fino a quando non equivale ad 11 (i-10>10)
```

Esistono inoltre variabili pre-definite per accedere in maniera più esplicita alla memoria RAM, come @R$n$, dove $n$ va da 0 a 15.

# 8 Cose a cui prestare attenzione

## 8.1 Naming conventions

Generalmente le variabili hanno nomi in minuscolo, mentre le etichette in maiuscolo.

## 8.2 Nota sull'uso delle variabili

In termini pratici, i primi 16 indirizzi della memoria RAM (da @R0 a @R15) sono riservati per l'esecuzione del programma in esecuzione. Tutti gli indirizzi successivi saranno utilizzati da eventuali variabili, a scelta dell'assemblatore. Per questo motivo, qualora in un programma ci si ritrovasse a dover scrivere qualcosa come @16 oppure un qualunque valore superiore a 15, allora non si dovranno utilizzare variabili, in quanto queste rischierebbero di essere sovrascritte.

# 9 Esercizi

## 9.1 Esercizi introduttivi

### 9.1.1 D=0

```
D=0
```

### 9.1.2 A=1 e D=1

```
// Con due istruzioni separate:
@0      // A=0
D=0     // D=0. Anche D=A andrebbe bene

// Con la scrittura compatta:
AD=0    // Assegno CONTEMPORANEAMENTE A e D a 0
```

### 9.1.3 D=3

```
// Non esiste l'istruzione D=n con n diverso da 1
@3  // Prima assegno ad A il valore di 3

D=A // Poi assegno a D il valore di A
```

### 9.1.4 D=RAM[3]+3

```
@3      // Assegno ad A valore pari a 3
D=M     // D=RAM[3]
D=D+A   // D=D+A, quindi D=RAM[3]+3
```

### 9.1.5 D=3+4

```
@3      // A=3
D=A     // D=A, quindi D=3
@4      // A=4
D=D+A   // D=D+A, quindi D=3+4
```

### 9.1.6 RAM[3]=7

```
1    @7       // A=7
2    D=A      // D=7
3    @3       // A=3
4    M=D      // RAM[3]=7
```

### 9.1.7 D=RAM[A]+3

```
1    D=M      // Supponendo che A sia dichiarata a priori
2    @3       // A=3
3    D=D+A    // D=D+A, quindi D=RAM[A]+3
```

## 9.2 Esercizi significativi

### 9.2.1 D=D-3

```
1    @3       // A=3
2    D=D-A    // Supponendo che D sia dichiarata a priori
```

### 9.2.2 D=10-RAM[5]

```
1    @10      // A=10
2    D=A      // D=A, quindi D=10
3    @5       // A=5
4    D=D-M    // D=D-M, quindi D=10-RAM[5]
```

### 9.2.3 RAM[0]=RAM[0]+2

```
1    @2       // A=2
2    D=A      // D=A, quindi D=2
3    @0       // A=0
4    M=M+D    // M=M+D, quindi RAM[0]=RAM[0]+2
```

### 9.2.4 RAM[RAM[0]]=RAM[0]

```
    @0      // A=0
    AD=M    // A e D = M, quindi A e D = RAM[0]
    M=D     // M=D, quindi RAM[RAM[0]]=RAM[0]
    // M equivale a RAM[RAM[0]] in quanto A=RAM[0]
```

### 9.2.5 RAM[2]=RAM[0]+RAM[1]

```
    @0      // A=0
    D=M     // D=RAM[0]
    @1      // A=1
    D=D+M   // D=RAM[0]+RAM[1]
    @2      // A=2
    M=D     // M=D, quindi RAM[2]=RAM[0]+RAM[1]
    // Nessuno vieta di spezzettare il problema!
```

## 9.3 Esercizi con salti

### 9.3.1 Se RAM[0]>0, JUMP all'indirizzo in RAM[1]

```
    @0      // A=0
    D=M     // D=RAM[0]
    @1      // A=1
    A=M     // A=RAM[1]
    D;JGT   // Se RAM[0]>0, JUMP a RAM[1]
```

### 9.3.2 D=D+1, poi se D=0 JUMP a istruzione 3

```
    D=D+1   // Incrementa D
    @3      // A=3
    D;JEQ   // Se D=0, JUMP a valore di A (quindi 3)
```

### 9.3.3 RAM[0]=RAM[5] e se RAM[5]<>0 JUMP a istruzione 3

```
1     @5        // A=5
2     D=M       // D=RAM[5]
3     @0        // A=0
4     M=D       // RAM[0]=RAM[5]
5     @3        // A=3
6     D;JNE     // Se RAM[5] != 0, JUMP a valore di A
```

## 9.4 Esercizi con le etichette

### 9.4.1 Implementazione moltiplicazione

```
1     // Volendo eseguire RAM[2] = RAM[0] * RAM[1]:
2     @2            // A=2
3     M=0           // RAM[2]=0
4     @4            // A=4
5     M=1           // RAM[4]=1, contatore per loop
6
7     (LOOP)        // Inizio loop
8         @4        // A=4
9         D=M       // D=RAM[4], prendo contatore
10        @1        // A=1
11        D=D-M     // RAM[4]=RAM[4]-RAM[1]
12
13        @END      // Preparo a terminare
14        D;JGT     // Se contatore-RAM[1]>0, termina
15
16        @0        // A=0
17        D=M       // D=RAM[0], val da moltiplicare
18        @2        // A=2
19        M=M+D     // sommo RAM[0] a RAM[2]
20
21        @LOOP     // Preparo a rifare
22        0;JMP     // Salto incondizionato
23
24    (END)         // Ciclo dummy finale
25        @END
26        0;JMP
```

### 9.4.2 Implementazione moltiplicazione senza quarto registro

```
// Volendo eseguire RAM[2] = RAM[0] * RAM[1]:
@2          // A=2
M=0         // RAM[2]=0

(LOOP)      // Inizio loop
    @0      // A=0
    D=M     // D=RAM[0], val da moltiplicare
    @2      // A=2
    M=M+D   // sommo RAM[0] a RAM[2]

    @1
    DM=M-1
    @LOOP
    D;JGT   // Rifai se RAM[1]-1>0

...resto codice...
```

### 9.4.3 RAM[10]=max(RAM[9..0])

```
@9      // A=9
D=A     // D=9
@11     // A=11
M=D     // RAM[11]=9, contatore
@9      // A=9
D=M     // D=RAM[9], il primo valore equivale a max
@10     // A=10
M=D     // RAM[10]=RAM[9]

(LOOP)          // Dichiaro inizio loop
    @11     // A=11
    D=M     // D=RAM[11], contatore
    @END    // Preparo a terminare
    D;JEQ   // Se D equivale a 0, JUMP

    @11     // Rimetto A=11
    M=M-1   // RAM[11]--
    A=M     // A=RAM[11]
```

```
19          D=M      // D=RAM[A]
20          @10
21          D=M-D    // D=(Max)-(valore corrente)
22          @SET     // Preparo a cambiare max
23          D;JLT    // Se D<0, JUMP
24
25          @LOOP    // Preparo a rifare
26          0;JMP    // Salto incondizionato
27
28      (SET)
29          @11      // A=11
30          D=M      // D=RAM[11], contatore
31          A=D      // A=contatore
32          D=M      // D=(valore corrente)
33          @10      // A=10
34          M=D      // Cambio max
35
36          @LOOP    // Preparo a rientrare nel loop
37          0;JMP    // Salto incondizionato
38
39      (END)        // Ciclo dummy finale
40          @END
41          0;JMP
```

Di questo esercizio si trova una copia del codice eseguibile nell'emulatore nella cartella _esercizi.