Architettura degli Elaboratori
Brutta

Andrea Malvezzi

05 novembre, 2024

Contents

1 ISA

L'ISA è l'interfaccia tra HW e SW e costituisce l'insieme delle istruzioni detto "instruction set" della CPU. Cambia da architettura a architettura e noi studieremo la versione HACK.

2 L'architettura HACK

L'architettura HACK non segue né la filosofia CISC né quella RISC ed esegue, essendo molto semplice e ridotta, ogni istruzione in un singolo ciclo di clock.
La RAM contiene i dati per il programma, la ROM contiene il programma stesso. Sono entrambe a 16 bit.
Si usano prevalenetemente due registri interni al processore, **D** e **A**. Inoltre si utilizza una "variabile" **M**, contenente il registro di memoria RAM attualmente puntato da A. Questo indirizzo di RAM puntato da A si indica con RAM[A] oppure MEM[A].
Oltre a questi due registri si usa anche il **Program Counter** (**PC**), ovvero il registro contenente l'address della prossima istruzione da eseguire. Si indica con ROM[PC].

2.1 Differenza tra A ed M

Il registro A è capace di contenere anche Costanti e valori simili (ecco come mai son più veloci), mentre il registro M...

3 Circuito spiegato

Le instruction vengono dalla ROM e vanno date in input ai registri A o M. Alla ALU entreranno sempre solamente due input: x (il registro D, i Data) e y (il registro A di Address oppure M di Memory).

4 Tipi di istruzione

Esistono due tipi di istruzione:

- A-instruction: istruzioni che lavorano sulla memoria (caricare valori su A);
- C-instruction (da C che è il bit di controllo): istruzioni che eseguono un'operazione (quindi che usano l'ALU) prelevando i due operandi da A, D, M oppure usando le costanti 0, 1 e -1. Questo risultato può essere poi salvato nei registri A, D, M o in alcune loro combinazioni.

Alcuni esempi:
comp = 0, 1, -1, D, A, !D, !A, ... Usa l'ALU;
dest = M, D, MD, A, AM, AD, AMD o nullo (in questo caso si omette l'istruzione). Salva un dato;
jump = JGT (Jump Greater Than), JEQ (Jump Equal), JGE, JLT, JNE, JLE, JMP (finto check, jumpa e basta) o nullo (si omette l'struzione). Salta ad un altro indirizzo di memoria a seguito di un check (è un *if* **sempre rispetto allo zero**);

4.1 Esempio

$$D = D + 1; JLT$$

Questo codice aumenta di 1 il valore del registro D, poi controlla che questa operazione ammonti a qualcosa minore di n, dove, essendo n omesso, si intende come pari a 0. In caso questo check risulti corretto, effettua un jump, ovvero cambia l'indirizzo nel PC e prosegue con l'istruzione successiva.

5 Le A-instructions

In forma generale si ha:
@value, dove value corrisponde al valore che vogliamo salvare in A.

5.1 Esempi

@17 // A = 17
D = A // D = Valore di A = 17 (uso una C-instruction)

@17 // A = 17
D = M // D = RAM[17] (uso una C-instruction)

6 Esercizi con A & C instructions

6.1 Assegna il valore zero a D

Se si parlasse di assegnare un valore ad A, useremmo la sintassi @value. Ma trattandosi del registro D, devo usare una C-instruction:
D = 0

6.2 Assegna il valore zero ad A

Trattandosi del registro A, posso usare una A-instruction:
@0

6.3 Assegna il valore tre a D

Non posso chiedere un valore non binario alla ALU di base, quindi devo prima assegnarlo ad A e poi prenderlo e metterlo in D, usando una A-instruction e poi una C-instruction:
@3
D = A

6.4 Assegna il valore uno sia ad A che a D

Ci sono due modi per eseguire questa operazione, concatenando una A-instruction e una C-instruction, oppure direttamente con una C-instruction (trattandosi di un valore binario):
Approcio con A e C instructions
@1
D = A
Approcio con singola C-instruction
AD = 1

6.5 Assegna a D il valore 3 e aggiungici 3

Non posso fare una somma tra numeri non binari in maniera diretta, quindi devo salvare i valori in due registri e sommare questi due (vedi 6.3):
@3
D = M
D = D + A

7 Esercizi più complessi

7.1 D = 10 - RAM[5]

@10
D = A
@5
D = D - M

7.2 RAM[0] = RAM[0]++

Approcio con 3 istruzioni:
@0
D = M
M = D + 1
Ottimizzato:
@0
M = M + 1

7.3 RAM[0] = RAM[0] + 2

@0
D = M
@2
M = D + A

7.4 D = RAM[RAM[0]]

@0
A = M
D = M
// AD = M non darebbe lo stesso risultato in quanto metterebbe CONTEMPORANEAMENTE sia A che D al valore corrente di M

7.5 RAM[RAM[0]] = RAM[0]

@0
A = M
M = A
Oppure:
@0
D = M
A = M
M = D

7.6 RAM[2] = RAM[0] + RAM[1]

@0
D = M
@1
D = D + M // D = RAM[0] + RAM[1]
M = D

8 Esercizi con A, C e JMP instructions

8.1 Se RAM[0] > 0 JUMP a RAM[1]

@0
D = M
@1
A = M // Il codice salta sempre al valore di A, quindi setto A a RAM[1]
D; JGT

8.2 D = D + 1, se D = 0 JMP all'istruzione 3

@3
D = D + 1; JEQ

8.3 RAM[0] = RAM[5], se RAM[5] <> 0 JMP istruzione 3

@5
D = M
@0
M = D
@3
D; JNE

9 Dichiarazione di etichetta

Un'etichetta è un bookmark che prende il valore dell'istruzione a seguire. Ad esempio:
@2
D = A
(BACK)
@4
D = M
In questo esempio (BACK) prende il valore della riga a cui corrisponde, quindi ROM[2] (da @4 in giù). Possiamo poi riutilizzare questa riga nel nostro programma richiamando @BACK:
@2
D = A
(BACK)
@4
D = M
@BACK
D; JGE

10 Il costrutto if-then-else

Si può ricreare il costrutto if-then-else tramite JUMP, C-instruction ed etichette:
D = 1 // o qualunque dato da testare
@ CASE_FALSE
D; JLE
... condizione vera ...
@END
0; JMP
(CASE_FALSE)
... condizione falsa ...
(END)
... resto del codice ...

In questo codice, richiamiamo @CASE_FALSE dopo aver settato i nostri dati. Questa operazione imposterà A al valore della riga in cui è presente la definizione dell'omonima etichetta.

Poi, controlliamo che la condizione che vogliamo verificare sia **false**: qualora questo accadesse, JMPiamo al valore di A, quindi al blocco di codice corrispondente al nostro *else* case.
A seguito del JMP, mettiamo il nostro blocco di codice da eseguire nel caso in cui la condizione imposta nel codice sia falsa (quindi la condizione che vogliamo verificare nella realtà sia effettivamente vera). Così facendo questo codice verrà eseguito nel caso in cui il JMP non venga effettuato.
Al termine del codice corrispondente al nostro *if* case, richiamiamo l'etichetta END, ovvero il codice seguente all'if-then-else, per poi effettuare un salto incondizionato (saltiamo in qualunque caso). Questo ci permettà di saltare il blocco di codice corrispondente all'*else* case.

11 Il costrutto while