Implementazione di IJVM con Mic-1

Aniello Minutolo

Microistruzioni e notazione

Sommario

1. Microistruzioni e notazione
2. Il ciclo principale dell’interprete IJVM per Mic-1
3. Funzionamento dell’interprete IJVM per Mic-1

Microistruzioni e notazione

Definiamo una notazione semplificata denominata **MAL** (Micro Assembly Language) per codificare le istruzioni di IJVM in termini di microistruzioni di Mic-1

- in tale notazione, tutte le attività che si svolgono durante un unico ciclo di clock si specificano su una singola linea.

Ad esempio, supponiamo che in un ciclo vogliamo:

- incrementare il valore del registro SP;
- dare inizio a un'operazione di lettura;
- fare in modo che l'istruzione successiva sia quella nella locazione 122 della memoria di controllo.

SP = SP + 1; rd; goto 122

Mic-1: Micro Assembly Language

In Mic-1 è possibile scrivere su più registri in ciascun ciclo. Per codificare questa evenienza, in MAL possiamo scrivere:

SP = MDR = SP + 1

- Al contrario, solo un registro alla volta può essere inviato all'ALU attraverso il bus B. L'altro input dell'ALU (lato sinistro) può essere +1, 0, -1 oppure il registro H.
- Copiare un registro in un altro (es: SP = MDR) corrisponde semplicemente a impostare 0 sul canale sinistro dell'ALU.

Mic-1: Micro Assembly Language

- Per sommare due registri (uno dei due deve per forza essere H):

SP = MDR = SP + 1

La tabella riporta le operazioni legali dove:

- SOURCE può essere MDR, PC, MBR, MBRU (versione di MBR unsigned) SP, LV, CPP, TOS o OPC.
- DEST può essere MAR, MDR, PC, SP, LV, CPP, TOS, OPC o H. Sono concessi assegnamenti multipli.

DEST = H
DEST = SOURCE
DEST = H
DEST = SOURCE
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE - H
DEST = SOURCE – 1
DEST = - H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = -1

Mic-1: Micro Assembly Language

L'accesso in **lettura** o **scrittura** alla **memoria** attraverso MAR/MDR è rappresentato in MAL rispettivamente dalle sigle:

- `rd` per la lettura
- `wr` per la scrittura

L'operazione di fetch, che consiste nel prelievo di un codice operativo (opcode) a 1 byte dal flusso di istruzioni ISA (IJVM) mediante la porta PC/MBR, viene indicata in MAL con il termine:

- `fetch`

Mic-1: Micro Assembly Language

- Per convenzione, ogni riga di codice MAL che non include un **salto esplicito** viene tradotta in modo implicito, assegnando a **NEXT_ADDRESS** l'indirizzo della microistruzione successiva.
- Per effettuare un salto incondizionato, è sufficiente specificare alla fine della riga:

```
goto label
```

- In questo modo, il valore di **NEXT_ADDRESS** viene forzato all'indirizzo della label indicata.

Mic-1: Micro Assembly Language

Per implementare i salti condizionali è necessaria una notazione specifica; come noto, JAMN e JAMZ sfruttano i bit di stato N e Z.

Supponiamo, ad esempio, di voler effettuare un salto nel caso in cui il valore del registro TOS sia uguale a zero. In tal caso, dovremmo:

- far passare TOS attraverso la ALU, in modo che il bit Z venga aggiornato;
- attivare JAMZ (impostandolo a 1) per selezionare tra due possibili microistruzioni successive.

Queste due operazioni possono essere eseguite nello stesso ciclo di clock. In notazione MAL, ciò si esprime come:

```
Z = TOS; if (Z) goto L1; else goto L2
```

Mic-1: Micro Assembly Language

La notazione MAL per l'uso del bit JMPC è:

```
goto (MBR OR valore)
```

Questa sintassi indica al microassemblatore di utilizzare valore per NEXT_ADDRESS e di impostare il bit JMPC in modo che in MPC venga calcolato l’OR logico tra MBR e NEXT_ ADDRESS.

Il ciclo principale dell’interprete IJVM per Mic-1

Implementazione di IJVM con Mic-1

L’interprete IJVM per Mic-1 prevede, come tutti gli interpreti, un ciclo principale infinito che continuamente legge, decodifica ed esegue le istruzioni

- all’inizio di ogni istruzione viene eseguita la stessa sequenza di operazioni;
- è importante che la lunghezza di questa sequenza sia la più breve possibile;
- a partire dal momento in cui viene avviata la macchina, ogni volta che viene eseguita questa microistruzione l’opcode IJVM da eseguire si trova già all’interno in MBR.

Implementazione di IJVM con Mic-1

Il **ciclo principale** è costituito dalla sola riga **Main1** che:

- incrementa il Program Counter (PC = PC +1);
- fetch del prossimo byte (opcode seguente o operando istruzione corrente);
- salta all'indirizzo dell'istruzione presente in MBR:
  - MBR è un registro a 1 byte che mantiene in modo sequenziale i byte dello stream dell'istruzione provenienti dalla memoria per poterli interpretare.

Etichetta	Microistruzione	Commenti
Main1	PC = PC +1; fetch; goto (MBR)	MBR contiene già l'opcode dell'istruzione corrente; sarà dunque compito del microprogramma di ogni istruzione IJVM provvedere a ciò

Implementazione di IJVM con Mic-1

Etichetta	Microistruzione	Commenti
Main1	PC = PC +1; fetch; goto (MBR)	MBR contiene già l'opcode dell'istruzione corrente; sarà dunque compito del microprogramma di ogni istruzione IJVM provvedere a ciò

Il **ciclo principale** è costituito dalla sola riga **Main1** che:

- quello che fa la microistruzione è effettuare un salto nel microcodice per eseguire questa istruzione IJVM, oltre a iniziare il prelievo del byte che segue l'opcode;
- questo byte potrebbe essere un operando oppure l'opcode di una nuova microistruzione.

Il microprogramma che interpreta IJVM, eseguito su Mic-1, è un programma composto da 112 microistruzioni

Etichetta	Microistruzione	Commenti
Main1	PC = PC +1; fetch; goto (MB)	MBR contiene già l'opcode dell'istruzione corrente; sarà dunque compito del microprogramma di ogni istruzione IJVM provvedere a ciò
nop1	goto Main1	Non esegue nulla
Iadd1 iadd2 iadd3	MAR = SP = SP – 1; rd H = TOS MDR = TOS = MDR + H; wr; goto Main1	Legge la seconda parola in cima allo stack H = cima dello stack Somma le due parole in cima allo stack; scrive in cima allo stack
Isub1 isub2 isub3	MAR = SP = SP – 1; rd H = TOS MDR = TOS = MDR – H; wr; goto Main1	Legge la seconda parola in cima allo stack H = cima dello stack Esegue la sottrazione; scrive in cima allo stack
Iand1 Iand2 iand3	MAR = SP = SP – 1; rd H = TOS MDR = TOS = MDR AND H; wr; goto Main1	Legge la seconda parola in cima allo stack H = cima dello stack Esegue l'AND; scrive nella nuova cima dello stack
…	…	…

Microprogramma per Mic-1

All’inizio e alla fine di ogni istruzione, il registro TOS contiene il valore puntato da SP, la parola che si trova in cima allo stack

- questo valore è ridondante, in quanto la parola può essere letta in qualsiasi momento dalla memoria;
- ciononostante avendola a disposizione in un registro è spesso possibile risparmiare un riferimento alla memoria;
- per poche istruzioni il mantenimento di TOS implica un numero maggiore di operazioni di memoria.

Implementazione di IJVM con Mic-1

Le microistruzioni non vengono eseguite sequenzialmente, ma ciascuna di loro è obbligata a indicare esplicitamente il proprio successore:

- questo perché tutti gli indirizzi della memoria di controllo corrispondenti ai codici operativi devono essere riservati per la prima parola della corrispondente istruzione dell'interprete.

Funzionamento dell’interprete IJVM per Mic-1

Funzionamento dell’interprete IJVM per Mic-1

Per vedere come funziona l’interprete, assumiamo che MBR contenga il valore 0x60 (opcode dell’istruzioni IADD di IJVM)

- sostituisce le due parole in cima allo stack con la loro somma.

Nel ciclo principale composto da una sola microistruzione compiamo tre azioni:

- incrementiamo PC, in modo che contenga l’indirizzo del primo byte dopo l’opcode;
- iniziamo a prelevare il byte successivo da portare all’interno di MBR;
- all’inizio di Main1 effettuiamo una diramazione verso l’indirizzo contenuto in MBR.

Etichetta	Microistruzione	Commenti
Main1	PC = PC +1; fetch; goto (MBR)	MBR contiene già l'opcode dell'istruzione corrente; sarà dunque compito del microprogramma di ogni istruzione IJVM provvedere a ciò

Funzionamento dell’interprete IJVM per Mic-1

Nel ciclo principale composto da una sola microistruzione compiamo tre azioni:

- incrementiamo PC, in modo che contenga l’indirizzo del primo byte dopo l’opcode;
- iniziamo a prelevare il byte successivo da portare all’interno di MBR
    - questo byte, prima o poi, si rivelerà necessario, o come operando per l’istruzione IJVM corrente oppure come opcode successivo (come nel caso dell’istruzione IADD, che non ha il byte per l’operando).

Funzionamento dell’interprete IJVM per Mic-1

All’inizio di Main1 effettuiamo una diramazione verso l’indirizzo contenuto in MBR:

- questo indirizzo, memorizzato nel registro dalla precedente microistruzione, equivale al valore numerico del codice opcode che in quel momento è in esecuzione;
- occorre prestare particolare attenzione al fatto che il valore che si sta iniziando a estrarre in questa microistruzione non gioca alcun ruolo nella diramazione.

Funzionamento dell’interprete IJVM per Mic-1

Il prelievo del byte successivo comincia in questo punto in modo che sia disponibile all’inizio della terza microistruzione

- in seguito potrebbe risultare necessario oppure no, ma conviene comunque far partire il prelievo, dato che la cosa non produce assolutamente alcun danno.

Funzionamento dell’interprete IJVM per Mic-1

Se il byte contenuto in MBR è composto da soli 0, allora identifica l’opcode di un’istruzione NOP

- in questo caso la successiva microistruzione viene prelevata dalla locazione 0 ed è quella etichettata con nop1;
- dato che questa istruzione non esegue alcunché essa effettua semplicemente un salto all’inizio del ciclo principale, dove viene ripetuta la sequenza utilizzando però il nuovo opcode memorizzato in MBR.

Funzionamento dell’interprete IJVM per Mic-1

Sottolineiamo che le microistruzioni non sono memorizzate consecutivamente all’interno della memoria di controllo

- la microistruzione `Main1` non si trova all’indirizzo 0 della memoria;
- all’indirizzo 0 deve essere memorizzata la microistruzione `nop1`.

Funzionamento dell’interprete IJVM per Mic-1

È compito del microassemblatore posizionare le microistruzioni negli indirizzi adatti e collegarle in sequenze di breve lunghezza usando il campo NEXT_ ADDRESS

- ogni sequenza inizia all’indirizzo corrispondente al valore numerico dell’*opcode* IJVM che interpreta (POP comincia per esempio in 0x57);
- il resto della sequenza può trovarsi in qualsiasi punto della memoria, e non necessariamente negli indirizzi immediatamente successivi.

Bibliografia

Libro di testo

- Andrew S. Tanenbaum e Todd Austin. Architettura dei calcolatori. Un approccio strutturale. 6/ED. Anno 2013. Pearson Italia spa. (Disponibile nella sezione “Biblioteca”)

Fonte argomenti e immagini

- Capitolo 4, Paragrafo 4.3, pp. 274-284