Programmazione IJVM

Un primo esempio di programmazione è il seguente:

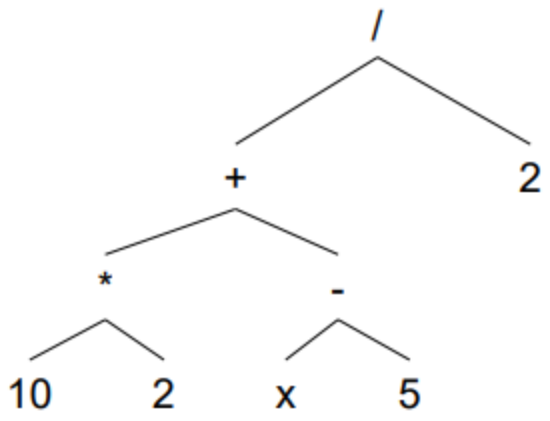
```
...
i = j + k;
if(i == 3) {
    k = 0;
}
else {
    j = j - 1;
}
...
```

Questo programma esegue una somma (**ILOAD**, **IADD**) assegna il risultato ad una variabile (**ISTORE**) e confronta due valori con un salto incondizionato (**IF_CMPEQ**). Esegue una sottrazione (**ISUB**) oppure assegna il valore costante a una variabile (**BITPUSH**, **ISTORE**).

```
Label   OPCODE      Operando    Commento
        ILOAD       j
        ILOAD       k
        IADD                    // j + k
        ISTORE      i           // i = j + k
        ILOAD       i
        BIPUSH      3
        IF_ICMPEQ   then        // if  (i==3) then ...
else:   ILOAD       j           // else
        BIPUSH      1
        ISUB                    // j -1
        ISTORE      j           // j = j -1
        GOTO        end_if
then:   BIPUSH      0
        ISTORE      k           // (then)... k = 0
end_if:
```

Notazione polacca inversa

Si costruisce un albero sintattico che rappresenta l'espressione. Si visita l'albero con ordine di operazione. Sull'operazione `(10 * 2 + (x - 5)) / 2`



Si visita l'albero in ordine differito.

Regole generali di traduzione

- var = constante → BIPUSH costante o LDC_W costante ISTORE var.
- var = espressione(es. x + 3) → ILOAD x; BIPUSH 3; IADD ISTORE var.
- atoi(byte) → BIPUSH byte BIPUSH 0x30 ISUB.
- i = i + 1 o i = i -1 → IINC i 1 o IINC i -1.

`atoi()` serve a trasformare una stringa nel carattere numerico. Da numerico ad ASCII solo con numeri non con lettere.

Codice C	Codice IJVM
while (var == espressione) { body }	Ciclo: calcolo della espressione sullo stack ILOAD var IF_ICMPEQ Body GOTO Continua Body: …..codice tra parentesi { body } GOTO Ciclo Continua:
while not(var == espressione) { body }	Ciclo: calcolo della espressione sullo stack ILOAD var IF_ICMPEQ Continua Body: …..codice tra parentesi { body } GOTO Ciclo Continua:

Codice C	Codice IJVM
do { body } while (var==espressione)	Ciclo: …..codice tra parentesi { body } calcolo della espressione sullo stack ILOAD var IF_ICMPEQ Ciclo … codice dopo il do-while
if (i==0) {ramo then} else {ramo else}	ILOAD i IFEQ ramothen ramoelse: codice ramo else GOTO dopoif ramothen: codice ramo then dopoif:

Codice C	Codice IJVM

if (i>=0) {ramo then}
else {ramo else}
 	
i>=0 equivale a !(i<0) ➔
if(i<0) {ramo else}
else {ramo then}
	ILOAD i
	IFLT ramoelse
	ramothen: codice ramo then
	GOTO dopoif
	ramoelse: codice ramo else
	dopoif:

if (i>0) {ramo then}
else {ramo else}
 	
i>0 equivale a !(i<0 or i==0) ➔
if(i<0) {ramo else}
else if (i==0) {ramo else}
else {ramo then}
	ILOAD i
	IFLT ramoelse
	ILOAD i
	IFEQ ramoelse
	ramothen: codice ramo then
	GOTO dopoif
	ramoelse: codice ramo else
	dopoif:

Codice C	Codice IJVM

if (i>0) {ramo then}
else {ramoelse}
 	
i>0 equivale a –i<0 ➔
if(0-i<0) {ramo then}
else {ramoelse}
	BIPUSH 0
	ILOAD i
	ISUB
	IFLT ramothen
	ramoelse: codice ramo else
	GOTO dopoif
	ramothen: codice ramo then
	dopoif:

Algoritmo di Horner

Da binario a decimale si legge il valore in cima allo stack, si moltiplica per un accumulatore. (metodo inefficiente)

Iniziamo con l'inizio dello stack sul numero 10110001 il primo ciclo avrà num = 0 e cont = 8. Dopo il primo ciclo num = 1 e cont = 7 questo fin quando cont è = 0. Il numero viene calcolato come: `num = num * 2 + atoi(cifra).`

```
        BIPUSH 0
        ISTORE num
        BIPUSH 8
        ISTORE cont
Ciclo:  ILOAD cont
        IFEQ  FINE
        BIPUSH 0x30
        ISUB
        ILOAD num
        DUP
        IADD
        IADD
        ISTORE num
        IINC cont -1
        GOTO Ciclo
Fine:   HALT
```

Per prima cosa si inizializzano la variabili prima num = 0 e poi cont = 8.

Dopo si controlla che cont non sia 0 altrimenti si salta alla label Fine.

Si controlla la cima dello stack con 0x30 e si sottrae con 0x31 che come risultato da 0x01 si carica num con il suo valore da carattere ad ASCII

Dopo di che si duplica il valore in num e si somma a se stesso

Si salva il valore su num e si decrementa cont.

Alla fine del ciclo si va nella label Fine.

Conta il numero di 1 in una stringa binaria

Per verificare se il primo numero è 1 basti pensare al controllo del numero negativo, se x è minore di 0 allora il primo numero è 1, per gli altri numeri basta shiftare la stringa di un valore dopo il confronto.

In MIC-1 non esiste lo shift a sinistra di un valore ma solo SLL8 quindi di 8 bit, possiamo ottenere lo stesso risultato semplicemente moltiplicando il valore di x per se stesso.

```
          BIPUSH 0
          ISTORE cont // cont = 0
          LDC_W valx
          DUP
          ISTORE x // x = n
Ciclo:
          DUP
          DUP
          IFEQ FINE    // while not(x==0){
          IFLT THEN    // if   (x<0)
          GOTO DOPO
 THEN:    IINC cont +1 // cont++;
 DOPO:    DUP
          IADD          // x*2
          DUP
          ISTORE x     // x = x*2;
          GOTO Ciclo    // } salta a inizio ciclo
FINE:     POP
          HALT          // fine
```

Come si struttura un programma

Un programma IJVM scritto nel formto simbolico JAS comprende:

- la dichiarazione delle costanti
- il programmara ovvero il main
- uno o più metodi

```
.costant
C1 10
```

```
C2 20
C3 30
.. ..
.end-costant

.main
    istruzioni varie
.end-main
```

Dentro al main posso dichiarare delle variabili locali

```
.main
    .var
    x
    y
    z
    .end-var
    .end-main
```

Si possono anche dichiarare dei metodi

```
.method nomeMetodo(par1, par2, ...)
    dichiarazione variabili
    istruzioni varie
.end-method
```

Ecco un esempio di programma in IJVM

```
.costant
VALX 0x70f0f0f
.end-costant

.main
    .var
    cont
    x
```

```
    .end-var
BIPUSH 0
ISTORE cont
LDC_W VALX
ISTORE x
Ciclo:
    ILOAD x
    IFEQ Fine
    ILOAD x
    IFLT Then
    GOTO Dopo
Then:
    IINC cont 1
Dopo:
    ILOAD x
    DUP
    IADD
    ISTORE x
    GOTO Ciclo
Fine:
    HALT
.end-main
```

Per stampare un istruzione

Per stampare un'istruzione si usa l'istruzione OUT il quale estrae un carattere dalla cima dello stack e lo visualizza sullo standard out della GUI.

Per prende in input un istruzione

Per prendere in input un carattere usiamo il comando IN che mette in cima allo stack il valore.

```
.main
.var
.end-var
```

```
leggiCarattere:
    IN
    DUP
    IFEQ bufferVuoto
    GOTO elaboraCarattere

bufferVuoto:
    POP
    GOTO leggiCarattere

elaboraCarattere:
    DUP
    BIPUSH 0x2E
    IF_ICMPEQ fine
    OUT
    GOTO leggiCarattere

fine:
    POP
    HALT
.end-main
```

Richiamo e utilizzo dei metodi

Per poter richiamare una funzione dobbiamo usare INVOKEVIRTUAL mentre per ritornare un valore IRETURN.

```
.method Prova(par1, par2, park)
Variabili locali
Codice
IRETURN
.end-method
```

Ogni volta che viene invocato un metodo si spostano LV e SP che puntano al nuovo record di attivazione.