Il Livello di ISA

A cura di:

Anna Antola Giuseppe Pozzi

DEI, PoliMI, Milano
anna.antola,giuseppe.pozzi@polimi.it

- versione dell'11 aprile 2003 -

Il livello di ISA

Il livello ISA

- I tipi di dato (Tanenbaum 5.2.1, 5.2.2, 5.2.5)
- Il formato delle istruzioni (Tanenbaum 5.3.5)
- Le modalità di indirizzamento istruzioni (Tanenbaum 5.4.1 - 5.4.6, 5.4.8, 5.4.13)

I tipi di dato

- Taglie privilegiate :
  - 1 bit: per codificare una variabile logica
  - 4 bit (nibble): per codificare in binario una cifra decimale
  - 8 bit (byte): per codificare in binario i caratteri alfanumerici e i segni tipografici
  - 16-32-64-(128).... perché multipli interi comuni di tutti i precedenti

Tipi di dato numerici

- A virgola fissa:
  - binari
    - interi con segno (signed) / interi senza segno (unsigned)
    - in complemento a 2 / in modulo e segno
  - decimali codificati in binario, cifra per cifra
- A virgola mobile:
  - binari in rappresentazione esponenziale (mantissa, esponente)

Tipi di dato numerici

- In ogni caso, l'ISA deve:
  - definire gli operatori utilizzabili;
  - definire procedure di approssimazione (arrotondamento/troncamento);
  - identificare i "risultati non rappresentabili e non approssimabili a numeri rappresentabili" (overflow).

Tipi di dato non numerici

- Caratteri alfanumerici
  - ASCII, UNICODE
- Booleani

- Indirizzi di memoria
  - Puntatori

Tipi di dati supportati da IJVM

Tipi di dati numerici per Java:

Tipo	**8 bit**	**16 bit**	**32 bit**	**64 bit**	**128 bit**
Intero con segno (complemento a 2)	X	X	X	X	
Intero senza segno					
Intero decimale codificato in binario					
Virgola mobile			X	X	

Tipi di dati *alfanumerici* per Java:

- caratteri UNICODE (16 bit)

Il formato delle istruzioni

- Definisce:
  - il codice operativo dell'istruzione
  - dove l'istruzione trova i suoi operandi (indirizzo e modalità di indirizzamento, memoria o registro)
  - dove l'istruzione lascia i suoi risultati (indirizzo e modalità di indirizzamento, memoria o registro)
- Inoltre, l'ISA deve sempre precisare eventuali "effetti collaterali" di ogni istruzione (ad esempio: se e come modifica i flag)

Formati istruzione tipici

Codice Operativo			
Codice Operativo	Indirizzo		
Codice Operativo	Indirizzo1	Indirizzo2	
Codice Operativo	Indirizzo1	Indirizzo2	Indirizzo3

- Se manca un indirizzo:
  - o l'istruzione non ha l'operando/risultato
  - o l'operando/risultato è in una posizione implicitamente determinata

Dimensione delle istruzioni vs. dimensione delle parole di memoria

Una istruzione per parola, con tutte le istruzioni di uguale lunghezza.

Più istruzioni per parola, con tutte le istruzioni di uguale lunghezza

Caso più generale: istruzioni di lunghezza diversa che possono essere contenute in più parole di memoria

Dimensione delle istruzioni vs. dimensione delle parole di memoria

Una parola di memoria		
Istruzione		
Istruzione	Istruzione	Istruzione
Istruzione		

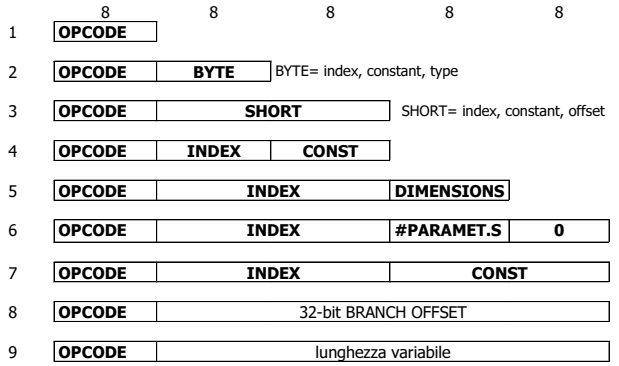
Dimensione delle istruzioni vs. dimensione delle parole di memoria

Massima flessibilità nella definizione delle istruzioni,

minimo spreco di spazio per il programma in memoria,

massima complessità dell'implementazione nella microarchitettura.

Formato delle istruzioni IJVM



Formato delle istruzioni IJVM

- Quasi tutte le istruzioni usano i formati tipo 1, 2 o 3:
  - OPCODE da un byte
  - Uno o due byte opzionali (per un indice, una costante, un tipo, un offset)
- In particolare, le più usuali hanno solo l'OPCODE (vedi achitettura a pila)

Formato delle istruzioni IJVM

- Alcune istruzioni di tipo 2 (2 byte) hanno le loro corrispondenti identiche di tipo 1 (1 byte), tramite indirizzamento implicito (cioè prefissato) delle variabili locali.
- Analogamente alcune istruzioni di tipo 3 (3 byte) hanno le loro corrispondenti identiche di tipo 2 (2 byte), tramite indirizzamento limitato nella constant pool (un byte solo di indirizzo: solo le prime 256 parole sono indirizzabili).
- I formati (tipi da 4 a 8) vengono usati per poche istruzioni; in particolare, il tipo 8 per salti a lunga distanza.

Codifica del campo indirizzi

- Il campo indirizzo rappresenta un **riferimento** all'operando (in memoria o in registro)
- Dipende:
  - dall' indirizzabilità della memoria (es. a byte, a parola di 2 – 4 – 8 byte)
  - dalla **modalità di indirizzamento**
- Si devono indirizzare:
  - dati
  - locazioni di arrivo di salti (scelta della prossima istruzione da eseguire)

Caratteristiche significative delle modalità di indirizzamento

- Prestazioni:
  - spazio occupato in memoria
  - velocità:
    - nel reperimento dell'istruzione completa
    - nella determinazione effettiva dell'indirizzo degli operandi e il loro reperimento

Caratteristiche significative delle modalità di indirizzamento

- Flessibilità d'uso:
  - effetti sulla rilocabilità del programma
  - rieseguibilità su dati diversi e rilocabilità dei dati
  - utilizzabilità in memorie ROM
  - adeguatezza a "scandire" particolari strutture dati (p.es. vettori)
  - adeguatezza a processore con: pipeline, multiple pipeline, esecuzione "fuori ordine", ecc.
  - adeguatezza al passaggio di dati tra procedure
  - ecc. ecc.

Modalita' di indirizzamento

- Indirizzamento immediato
- Indirizzamento diretto
- Indirizzamento di registri
- Indirizzamento indiretto tramite registri
- Indirizzamento con indice
- Indirizzamento tramite lo stack (v. JVM)

Modalita' di indirizzamento

- NOTA BENE: gli esempi utilizzati nella descrizione fanno riferimento a ipotetiche istruzioni ISA (in simbolico), con al più 2 operandi, nel formato

OPCODE	operando_1	operando_2

dove:operando_1 rappresenta sorgente_1 e destinazione;

operando_2 rappresenta sorgente_2

Indirizzamento immediato

- Il valore dell'operando è contenuto direttamente nell'istruzione

istruzione	*significato*
MOV R1, #0	0 => R1
ADD R2, #4	[R2]+4 => R2

NOTA BENE: [*nome registro*] indica il valore contenuto nel registro

- il simbolo # individua l'indirizzamento immediato
- il valore rappresentabile è limitato dalle dimensioni del campo indirizzo
- utile nel caso di costanti "piccole"

Indirizzamento diretto

- Nel campo operando è specificato l'**indirizzo di memoria** dell'operando.

istruzione	*significato*	
MOV R1, A	mem(A) => R1	*A è l'indirizzo simbolico*
ADD R1, B	[R1]+mem(B) => R1	*B è l'indirizzo simbolico*
MOV 1024, R1	[R1]=> mem(1024)	
BLT LOOP	salta se nell'ultimo confronto op_1 < op_2	*LOOP è l'indirizzo simbolico della destinazione di salto*

- L'indirizzo del dato non può cambiare durante l'esecuzione del programma (non può essere usato per referenziare parametri e variabili locali di un sottoprogramma)
- l'indirizzo esprimibile è funzione delle dimensioni del campo operando dell'istruzione

Indirizzamento di registri

- Nel campo operando è specificato **l'indirizzo di un registro** nel quale è contenuto il valore dell'operando.

istruzione	*significato*
OR R1, R2	[R1] or [R2] => R1
CMP R2, R3	confronta [R2] e [R3]

- compattezza delle istruzioni (n. limitato di registri e quindi bit per indirizzarli)
- non c'è accesso alla memoria per il movimento dei dati
- impone un uso *rigido* ed *esplicito* dei registri.
- solitamente richiede comunque ricopiature di dati tra memoria e registri.

Indirizzamento indiretto tramite registri

- Il campo operando contiene il riferimento ad un registro e il registro contiene l'indirizzo dell'operando (*puntatore*)

istruzione	*significato*
ADD R1, (R2)	[R1] + mem([R2]) => R1

- le due parentesi tonde **( )** individuano, in modo simbolico, l'indirizzamento indiretto tramite registro
- consente di far riferimento a memoria senza avere l'indirizzo completo nell'istruzione
- se varia il valore del registro, la stessa istruzione può essere usata per far riferimento a dati in memoria in posizioni diverse
- utile per la scansione di array

Indirizzamento con indice

- L'indirizzo di memoria dell'operando è specificato nell'istruzione tramite un offset costante (*base*) e il valore (variabile) di un registro, che funziona da registro indice

istruzione	*significato*	
MOV R4, A(R2)	mem(A + [R2]) => R4	*A è il valore simbolico della base*
ADD R4, B(R2)	[R4] + mem(B + [R2]) => R4	*B è il valore simbolico della base*

- Calcolo dell'indirizzo:

  ind. operando = offset + [reg]

Indirizzamento con indice

- La notazione base(reg) individua, in modo simbolico, l'indirizzamento;
- è la forma più usata per la scansione di array (il registro rappresenta l'indice degli elementi dell'array, l'offset costante rappresenta l'ind. iniziale dell'array).
- Può anche essere usato nel modo seguente (vedi JVM):
  - l'indirizzo di memoria dell'operando è specificato nell'istruzione tramite un offset variabile (*indice*) e il valore costante di un registro (*base*)
- Calcolo dell'indirizzo:
  - ind. operando = offset variabile+ [reg. base]

Modalità di indirizzamento in JVM

- In JVM ogni istruzione è associata ad una specifica modalità di indirizzamento
- JVM non ha registri visibili, cioè citabili in modo esplicito come operandi di una istruzione
- Le istruzioni aritmetico-logico hanno solo il campo OPCODE, e prevedono gli operandi già presenti nello stack in posizione opportuna (indirizzamento implicito nell'istruzione)
- Alcune istruzioni (ad es. BIPUSH) usano la modalità di indirizzamento immediato
- Le istruzioni di movimento dati da e in memoria (e altre ancora) usano la modalità di indirizzamento con indice con riferimento ad un registro implicito nel tipo di istruzione. La versione a indice utilizzata è quella in cui il registro implicito contiene la base, mentre l'indirizzo nell'istruzione rappresenta l'offset variabile (indice) rispetto alla base.