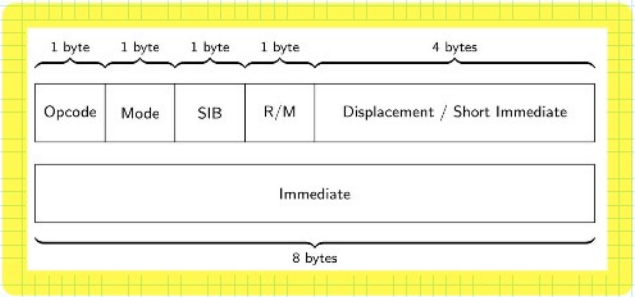
FORMATO ISTRUZIONI MACCHINA

domenica 13 novembre 2022 13:31

La codifica delle ISTRUZIONI NON utilizza sempre lo stesso numero di bit. Le ISTRUZIONI hanno un formato variabile:



= 16 BYTE

Quando vado a prelevare dalla memoria un certo INSIEME di BYTE per capire quale ISTRUZIONE devo andare ad eseguire, mi serve qualche cosa che mi indichi quali sono e quanti sono i BYTE che devo INTERPRETARE per eseguire la mia ISTRUZIONE.
Questo è uno dei motivi per il quale noi organizziamo le NOSTRE ISTRUZIONI IN CLASSI; (→ tipologie di operandi)
Perché ci aspettiamo che ISTRUZIONI simili utilizzino gli stessi operandi, e questo serve alla CU, individuando la classe, quanti BYTE servono per INTERPRETARE quella specifica ISTRUZIONE.

QUESTO CONCETTO LO RITROVIAMO NEL CODICE OPERATIVO OPCODE.

- **OPCODE**: CONTIENE UN'INFORMAZIONE SULLA CLASSE. QUINDI LA CU, LEGGENDO I BIT che SONO MEMORIZZATI NEL PRIMO BYTE NELLA CODIFICA NELLE ISTRUZIONI, È IN GRADO DI CAPIRE COME DEVE INTERPRETARE I BYTE SUCCESSIVI.
- **MODE**: DESCRIVE LA modalità operativa di una determinata ISTRUZIONE appartenente AD UNA CLASSE.
- **SIB**: Scala Indice Byte, vedremo come verranno utilizzati questi 3 parametri, PER ANDARE A RECUPERARE I dati dalla memoria.
- **R/M**: Register OR Memory, descrive delle INFORMAZIONI SU gli operandi di TIPO REGISTRO O gli operandi di TIPO memoria, coinvolti da un ISTRUZIONE;

Dopodiché abbiamo 4 BYTE e 8 BYTE successivi, che ci servono per RAPPRESENTARE delle Costanti o degli SPIAZZAMENTI IN memoria.

ANALIZZIAMO BENE UNO PER UNO ...

OPCODE 1 BYTE.

Class	Type

Sono due campi ciascuno di 4 bit. I 4 BIT PIÙ SIGNIFICATIVI IDENTIFICANO LA CLASSE!
Se io utilizzo 4 BIT di classe posso rappresentare fino a 16 classi distinte (2^4).
LA CLASSE IDENTIFICA LA FAMIGLIA DI ISTRUZIONI A CUI APPARTIENE L'ISTRUZIONE specifica, che abbiamo appena prelevato dalla memoria;
PER ESEMPIO le ISTRUZIONI ARITMETICHE [CLASSE 2] AVRANNO SCRITTO ALL'INTERNO di questi 4 bit 0010.

FRA TUTTE LE ISTRUZIONI di quella classe, devo capire quale è la specifica ISTRUZIONE RAPPRESENTATA IN QUESTO CASO, E I 4 BIT MENO significativi identificano la precisa ISTRUZIONE.

Se ho una ADD IN CLASS ci sarà SCRITTO 0010, e IN TYPE UN codice NUMERICO che identifica l'ISTRUZIONE di somma; Se vado a fare la SOTTRAZIONE, avrò la stessa classe MA UN TIPO differente!

IL MIO PROCESSORE leggendo questi 8 bit PUÒ capire quali sono le MICROOPERAZIONI che PERMETTONO di eseguire questa specifica ISTRUZIONE.

Io posso utilizzare varie TIPOLOGIE di REGISTRI, posso utilizzare REGISTRI A 32, 16, 64 BIT, e quindi Bisogna portare l'INFORMAZIONE che sto, AD ESEMPIO, utilizzando un operando di TIPO REGISTRO, e il REGISTRO che sto usando devo considerarlo a 32 bit.
Questa operazione va codificata.
Devo specificare quali sono gli operandi della mia ISTRUZIONE;
PER questo ci viene IN AIUTO il campo Mode:

MODE 1 BYTE

SS	DS	DI	Mem
2 bit	2 bit	2 bit	2 bit

I due bit più significativi, **SS** sono due bit che descrivono la codifica della dimensione dell'operando: BYTE, WORD, longword e Quadword

DS, è la dimensione dell'operando destinazione, perché possono esistere istruzioni che hanno dimensione diversa per operando sorgente e operando destinazione;

Posso codificare una costante dentro la mia istruzione:

`ADDb $1, %al` Somma 1 al contenuto di al

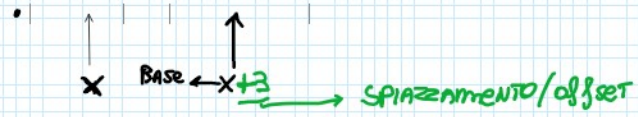
- `$1`: specifico la costante
- `%al`: che è un registro! → 3 bit logici (6)

IN **DI**, IL BIT **I** INDICA SE C'È O MENO LA PRESENZA DI UNA COSTANTE, SE È ZERO NON STO USANDO NESSUNA COSTANTE, SE VALE 1 SIGNIFICA CHE ALL'INTERNO DI UN'ISTRUZIONE STO USANDO UNA COSTANTE.

MA PERCHÈ NON VIENE SALVATO IL FATTO SE LA COSTANTE È UNA SORGENTE O UNA DESTINAZIONE? PERCHÈ NON HA SENSO SCRIVERE SU UNA COSTANTE!
LA COSTANTE STA SOLO NELLA SORGENTE.

IL BIT **D** È UN FLAG CHE MI PERMETTE DI SPECIFICARE SE ESISTE UNO SPIAZZAMENTO;

Cos'è uno spiazzamento?
Ogni cella di memoria è identificata da un indirizzo. Quando vado a scrivere in memoria io, non necessariamente dovrò scrivere nelle mie istruzioni l'indirizzo completo. Potrò specificare un indirizzo relativo a partire da qualche altro coso.



Se avrò un BIT memorizzato da qualche parte in memoria, posso accedere ad un altro indirizzo A PARTIRE da X più 3 byte: X+3.
Mi spiazzo a partire da un indirizzo di BASE.

IL FLAG D MI DICE SE È PRESENTE O MENO UNO SPIAZZAMENTO DI MEMORIA ALL'INTERNO della mia ISTRUZIONE.

GLI ULTIMI DUE BIT, **MEM**, MI IDENTIFICANO quale dei due operandi (sorgente o destinazione) RAPPRESENTA UN INDIRIZZO IN MEMORIA ANZICHÈ UN REGISTRO;

`ADDb $1, i` (i: iteratore) Sommare 1 al contenuto dell'area di memoria i.

IN questo caso l'operando di destinazione sarà un indirizzo di memoria e tengo traccia di questa informazione.
IL CAMPO MEM FA questo.

Campo	Valore	Significato
SS	00	La sorgente è un byte
	01	La sorgente è una word
	10	La sorgente è una longword
	11	La sorgente è una quadword
DS	00	La destinazione è un byte
	01	La destinazione è una word
	10	La destinazione è una longword
	11	La destinazione è una quadword
DI	00	Spiazzamento non utilizzato, immediato non presente
	01	Immediato presente
	10	Spiazzamento utilizzato
	11	Spiazzamento utilizzato, immediato presente
Mem	00	Sia la sorgente che la destinazione sono registri
	01	La sorgente è un registro, la destinazione è in memoria
	10	La sorgente è in memoria, la destinazione è un registro
	11	Condizione impossibile (genera un'eccezione a runtime)

`ADDb %al, i` → %al: REGISTRO SORGENTE; i: MEMORIA DESTINAZIONE

`ADDb i, %al` → i: MEM; %al: REGISTRO

NON SI PUÒ COPIARE UN DATO da MEMORIA a MEMORIA.
Si PASSA PER la CPU, Si CARICA il valore dalla memoria al REGISTRO, e si SCRIVE CON

UN dato da MEMORIA a MEMORIA.
Si PASSA PER la CPU,
si CARICA il valore dalla memoria al REGISTRO, e si SCRIVE con un'altra ISTRUZIONE, dal REGISTRO alla memoria.

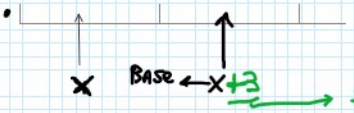
MEM ↓ ↳ REGISTRO

ORA CERCHIAMO di capire come i PROCESSORI Z64 MI PERMETTONO di identificare una delle celle in memoria:

MODALITÀ DI INDIRIZZAMENTO IN MEMORIA:

$$\left[\begin{Bmatrix}AX\\BX\\CX\\DX\\SP\\BP\\SI\\DI\\R8\\R9\\R10\\R11\\R12\\R13\\R14\\R15\end{Bmatrix}\right] + \left[\begin{Bmatrix}AX\\BX\\CX\\DX\\SP\\BP\\SI\\DI\\R8\\R9\\R10\\R11\\R12\\R13\\R14\\R15\end{Bmatrix} * \begin{Bmatrix}1\\2\\4\\8\end{Bmatrix}\right] + [spiazzamento]$$

Significa che quando io vado ad identificare un'area di memoria, il mio PROCESSORE farà queste somme e queste moltiplicazioni, PER andare a capire qual'è l'INDIRIZZO esatto IN CUI io voglio leggere o SCRIVERE.



L'INDIRIZZO di questa cella è X e l'INDIRIZZO dell'altra cella è X+3.

PER POTER specificare X+3 devo specificare lo SPIAZZAMENTO e devo specificare qual'è la BASE!

Noi ABBIAMO la POSSIBILITÀ di USARE un REGISTRO BASE, quindi posso SCRIVERE il valore della BASE all'INTERNO di un REGISTRO;
Poi, posso calcolare la locazione da cui voglio leggere o SCRIVERE, sommando al CONTENUTO del REGISTRO BASE un determinato spiazzamento!

[Se voglio SCRIVERE 1 all'INTERNO di X+3 posso fare così:]

```
Movq  $X, %rax      → SCRIVO X dentro rax (destinazione è un REGISTRO)
Addb  $1, 3(%rax)   → somma 1 a X+3 (destinazione è operando in MEMORIA)
```

è RAX quindi 64 BIT

SCRIVE la codifica BINARIA di X dentro il REGISTRO.

qui SOLO ORA che il CONTENUTO di RAX è X.

64 BIT perché STO parlando di un INDIRIZZO!

lo SPIAZZAMENTO è 3 A PARTIRE dal REGISTRO BASE

SPIAZZAMENTO = 3
BASE = X

LO TRATTO come costante. TRATTO un INDIRIZZO come costante.

SE AVESSI SCRITTO: `movq X, %rax` → INTERPRETA la codifica BINARIA di X come UN INDIRIZZO di MEMORIA, e VA a PRENDERE IL CONTENUTO di quell'AREA di MEMORIA

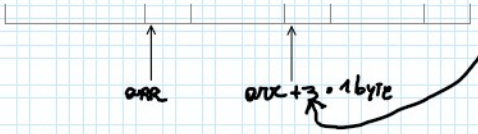
AREA di MEMORIA, LA SORGENTE È UN OPERANDO IN MEMORIA. [INDIRIZZO]

* DOPO VIENE eseguita la ADD!

leggo il valore CONTENUTO di quell'area di memoria.

qui l'INDIRIZZO è comunque un NUMERO, e quindi lo TRATTIAMO come costante.
STO SCRIVENDO la costante X dentro %rax, e dopo dico al PROCESSORE che questo NUMERO deve essere INTERPRETATO come INDIRIZZO;

CONCETTO DI ARRAY!



IL MIO VETTORE INCOMINCIA IN memoria da qualche parte.

ARR[3], voglio accedere al TERZO elemento del mio VETTORE;

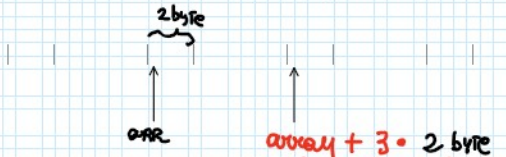
come realizzo in assembly un'istruzione di questo tipo? ARR[i]

```
MOVQ $3, %RCX
MOVQ $ARR, %rax
add $1, 3(%rax)   ≡   add $1, (%rax, %rcx, 1)
```

REG. BASE, offset, TAGLIA (perché ciascuna cella ha dim. 1 BYTE)

A PARTIRE dal REGISTRO RAX, UTILIZZANDO come SPIAZZAMENTO IL REGISTRO RCX.

Se CIASCUNA cella è due BYTE:



MA QUESTO LO FACCIAMO perché quando SCRIVEREMO CODICE ASSEMBLY CREEREMO ARRAY?

CAMBIA LA TAGLIA:

```
MOVQ $3, %RCX
MOVQ $ARR, %rax
add $1, 3(%rax)   ≡   add $1, (%rax, %rcx, 2)   ≡   addb $1, X(,%rcx,2)
```

INDICE o

REG. BASE, offset, SCALA (perché ciascuna cella ha dim. 1 BYTE)

$ arr

arr + 3·2

IL MIO PROCESSORE PER CALCOLARE L'INDIRIZZO di memoria andrebbe a fare questo calcolo: 3·2

• [MA TUTTO QUESTO DOVE STA? chiedevo ISTRUZIONI ASSEMBLY] •

I VALORI DI SCALA AMMESSI SONO: 1, 2, 4, 8.

Perché i TIPI PRIMITIVI gestiti dal PROCESSORE sono BYTE, WORD, long, QUADWORD;

QUINDI ESISTE IL BYTE SIB

SIB 1 BYTE

SCALA, INDICE, BASE!

7	6	5	4	3	0
B_p	I_p	Scale		Index Register	

↳ Di Mode

GLI ULTIMI DUE BIT, MOD, MI IDENTIFICANO quale dei due operandi (sorgente o destinazione) RAPPRESENTA UN INDIRIZZO IN MEMORIA ANZICHÉ UN REGISTRO;

B_p = STO USANDO UNA BASE NELLA MIA MODALITÀ DI INDIRIZZAMENTO;

I_p = STO USANDO UN INDICE; } quali sono passati come parametro;

addb $1, X(,%rcx,2) // LA BASE la esprimo come spiazzamento

arr = è un indirizzo in memoria, ed è il valore associato a quell'indirizzo!

[BIT dell'IR] → X + i·2

LA BASE posso non scriverla dunque!

[BIT dell'IR] → quell' indirizzo!
x + i·2

LA BASE ROSSO NON SCRIVERLA DUNQUE!

SCALE = 2 BIT PER RAPPRESENTARE 1, 2, 4, 8

Nel BANCO dei REGISTRI, un REGISTRO lo identifico con un codice numerico di 4 bit!
Qual'è il REGISTRO indice che devo andare ad usare?

INDEX REGISTER = 4 bit per il REGISTRO INDICE! (e se non c'è? o c'è un indirizzo?)

Mi MANCA IL REGISTRO BASE, IL REGISTRO SORGENTE e quello destinazione! `movq %rax, %rbx` (S, D)
che è il MODO PER CUI ESISTE IL CAMPO R/M.

R/M 1 BYTE

7 — 4	3 — 0
Source (Base) Register	Destination (Base) Register

4 BIT REGISTRO SORGENTE — 4 BIT REGISTRO DESTINAZIONE

Se uno dei due operandi è un operando in MEMORIA, non ha senso dire che sto rappresentando un REGISTRO SORGENTE, quella codifica viene usata come REGISTRO BASE. cosa scrivo?

```
movq $x, %rax
addb $1, 3(%rax)
```
↳ OP. IN MEMORIA perché ho le parentesi.

Posso avere fino a 3 REGISTRI in un'istruzione:

[SPOSTARE IL CONTENUTO di un REGISTRO in un'area di MEMORIA RAPPRESENTATA come BASE, INDICE e SCALA]

```
movq %RAX, (%rbx, %rcx, 8)
```
RAX: SORGENTE; rbx: BASE; rcx: INDICE; 8: SCALA