OPERANDI

lunedì 21 novembre 2022 20:54

queste convenzioni vengono usate per rappresentare gli operandi delle istruzioni:

VIENE UTILIZZATO:

- **B**, un operando è un registro general purpose, un indirizzo di memoria, o un valore immediato.
  Se sto utilizzando un valore immediato, la sua posizione dipende dalla possibile presenza dello spiazzamento e dalla sua dimensione.
- **E**, un operando è un registro di uso generale, o un indirizzo di memoria.
  In caso di un indirizzo di memoria, qualsiasi combinazione della modalità di indirizzamento è lecita.
- **G**, l'operando è un registro di uso generale;
- **K**, l'operando è una costante numerica non segnata di valore fino a $2^{32}-1$
- **M**, l'operando è una locazione di memoria, codificata come uno spiazzamento a partire dal contenuto del registro RIP dopo l'esecuzione della fase di fetch.

Come viene attuata la variazione di flusso di controllo nel nostro programma;

```
IF (x>0) THEN
   .... (*) |K
ELSE
   ....
```

Tutte le istruzioni del nostro programma in memoria, devono essere contigue.
Qui però ho una variazione di flusso di controllo

Se la condizione non è verificata, io non devo andare a codificare l'istruzione successiva, devo andare ad eseguire l'istruzione che sta nell'else.

Devo avere la possibilità di alterare il flusso di controllo in maniera esplicita perchè magari voglio fare salti.

RIP, viene automaticamente aggiornato, DURANTE LA FASE DI FETCH, per puntare subito all'istruzione successiva,
quando io ho eseguito il controllo, il mio RIP sta puntando alle istruzioni che stanno qui (*).
E se la condizione non è verificata non deve puntare lì.

DEVO MODIFICARE IL CONTENUTO DI RIP.

IN REALTÀ possiamo rappresentare il tutto come una "quantità" che deve essere sommata al registro RIP.

```
IF (x<0) THEN
   RIP = RIP + K
```

↳ la quantità di byte che passa nel blocco IF soddisfatto! (offset)

Rappresento la distanza di questo salto con un offset
Ma l'esecuzione di questa somma avviene dopo la fase di fetch.
Perché è la fase di fetch stessa che incrementa RIP.

RIP è eseguita da quella somma avvenuta dopo la fase di fetch.
Perché è la fase di fetch stessa che incrementa RIP.
K non è uno spiazzamento a partire dall'inizio della mia ISTRUZIONE, ma è uno spiazzamento a partire dalla fine dell'ISTRUZIONE.
Perché la dimensione dell'ISTRUZIONE è già stata sommata a RIP IN fase di fetch.

$$k = K$$

- **Imm K** = L'operando è un dato immediato di K cifre binarie!