REGOLE ASSEMBLY

lunedì 8 giugno 2020 12:42

1) Per operare il confronto faccio la sottrazione fra due operandi
2) Per operare il salto devo introdurre una etichetta dove l'istruzione di salto porta quando non si verifica la condizione richiesta mentre devo mettere ciò che deve accadere se si verifica ciò che voglio subito dopo il jump in modo che venga eseguito se non si esegue il jump
3) La sottrazione SUB è un'operazione distruttiva di uno dei due operandi in quanto si va a ricopiare il risultato in uno dei due registri. Per ovviare a tale problema basta utilizzare l'operazione CMP (compare) che altera i flag senza però modificare uno dei due registri contenenti gli operandi
4) Una istruzione che non può mai essere eseguita è quella di operazione fra due parte della memoria, cioè da memoria in memoria
5) Le operazioni logiche vengono operate bit a bit
6) Quando voglio azzerare un registro basta fare l'XOR con se stesso
   XOR EAX, EAX
7) Quando conosco la lunghezza di una variabile la specifico nella parte SECTION .DATA. Quando non so quanto spazio occupererà un valore allora lo dichiaro nella SECTION .BSS
8) Quando si sposta un dato in una parte (registro) di un registro retrocompatibile che ha una dimensione maggiore del dato bisogna controllare che nella prima parte non ci siano altri dati. In tal caso, per ovviare al problema, si può utilizzare MOVSX o MOVZX per estendere i bit
9) In Assembly non è possibile verificare il singolo bit di una variabile per verificare il segno o la parità. Si può fare ciò attraverso lo SHIFT che sposta il bit nel carry flag dove posso utilizzare delle condizioni di verifica quali JC (CF==1) e JNC (CF==0)
10) Non posso utilizzare una variabile di lunghezza ottenuta col dollaro come dato per riservare dello sbazio nella sezione .bss
    //error: attempt to reserve non-constant quantity of BSS space
11) Quando dichiaro una costante scrivo equ
12) Quando dichiaro un valore inizializzato lo pongo in .data
13) Quando scrivo un dato non inizializzato lo pogno in .bss
14) Quando scrivo un dato non inizializzato devo usare reserve e non define
15) Quando uso l'operatore mul il primo registro è implicito -> è ad un solo operando
16) Quando uso lo shift devo specificare il numero di shift
17) Se non metto l'etichetta di start non funziona il codice
18) Se si vuole accedere al contenuto di una variabile bisogna meterla fra parentesi quadre
19) Se si vuole ottenere l'indirizzo di una variabile la si richiama col nome
20) Quando si sposta un indirizzo di un elemento indirizzo presente in un registro, poiché il registro è a 32 bit mentre l'elemento potrebbe essere più piccolo, per spostare l'elemento in un registro più piccolo si usa specificare la dimensione dell'elemento
    es. elemento dw 59
    mov EAX,elemento -> sposto l'indirizzo di elemento in eax
    mov cx, word[EAX] -> tale spostamento è possibile poiché sposto solo i bit che mi interessano
21) Bisogna ottimizzare lo spazio utilizzato in base alla dimensione dei dati che si usano
22) Quando opera la moltiplicazione e questa viene spostata nei registri EDX:EAX, in EAX si ha la parte meno significativa, in EDX quella più significativa
23) Nell'istruzione di mov non posso usare un registro non a 32 bit come base altrimenti mi da problemi
24) Una costante non rappresenta un indirizzo che contiene il valore ma rappresenta direttamente il valore
25) Quando lavoro con numeri con segno conviene sempre usare movsx per estendere il segno
26) Quando uso movsx o movzx devo specificare la dimensione del dato prelevato a destra. Infatti solo dei registri viene individuato automaticamente la dimensione mentre quando si usa mov speciale bisogna indicare quanti bit prelevare dalla memoria
27) Per mantenere il segno dei dati bisgona usare un registro più grande del dato in cui fare la copia tramite movsx così che si mantenga il segno
28) Ogni lettera è codificata su 8 bit e una stringa è definita come array di byte
29) Se devo copiare un immediato o una variabile indirizzo in memoria devo specificaren la grandezza
    es. MOV[VAR], word 2
    MOV[A], dword VAR

30) Nelle stringhe il codice 0 indica la fine della stringa, il codice 10 il ritorno a capo (indicabile direttamente come \n)
31) Una stringa si scrive in singolo apici
32) Bisogna controllare che il jump verifichi i flags modificati dalla operazione precedente altrimenti se ne considerasse altri ci potrebbero essere dei problemi
33) Se opero uno spostamento di un dato in un registro devo usare la parte di registro proporzionata al dato altrimenti copia male il dato, oppure usare movsx o movzx, poiché viene prelevato tanto dato quanto è la grandezza del registro o in caso se viene indicato

TIPI DI OPERANDI

1) INTEGER -> valore numerico con segno contenuto in un byte, una word o una doubleword
2) ORDINAL -> valore numerico senza segno contenuto in un byte, una word o una doubleword
3) NEAR POINTER -> indirizzo a 32 bit
4) STRING -> sequenza di byte, word o doubleword. Ciascuna stringa può contenere da 0 byte a 4 G
5) BIT FIELD -> sequenza di massimo 32 bit

OPERAZIONI POSSIBILI:

1) da registro a registro
2) da memoria a registro
3) da registro a memoria
4) da immediato a registro
5) da immediato a memoria

NON si può spostare da memoria a memoria -> per fare ciò bisogna usare un registro d'appoggio in cui spostare uno dei valori

SIGNIFICATO ISTRUZIONI

ISTRUZIONI GENERICHE:

1) MOV A,B -> sposta il contenuto di B in A
2) MOVZX A,B -> sposta il contenuto del registro B (più piccolo) in A (più grande) estendendo i bit in 0
3) MOVSX A,B -> sposta il contenuto del registro B (più piccolo) in A (più grande) estendendo secondo il segno

4) ADD A,B -> somma A e B e sposta il risultato in A
5) SUB A,B -> opera la differenza B - A e salva il rusltato in A
6) CMP A,B -> opera il confronto fra A e B senza modificare i registri. Può essere seguito da un comando di JUMP
7) NEG A -> ottiene l'opposto di A in complemento a 2 (cioè A in segno opposto) sottraendo A da 0 e salvandolo in A
8) NOT A -> ottiene l'opposto di A in complemento a 1 (cioè inverte tutti i bit) sottraendo A da 0 e salvandolo in A
9) DIV A,B -> opera la divisione in numero naturale fra B e A e salva il risultato in A
10) IDIV A,B -> opera la divisione in numero con segno (complemento a 2) fra B e A e salva il risultato in A
11) DIV/IDIV A,B,C -> si hanno tre casi:
    ////////
12) MUL/IMUL A -> moltiplica A per
    - ◊ il contenuto di AL e salva il risultato in AX (se byte)
    - ◊ il contenuto di AX e salva il risultato in DX:AX (se word)
    - ◊ il contenuto di EAX e salva il risultato in EDX:EAX (se dword)
13) IMUL A,B -> moltiplica A per B e lo salva in A
14) IMUL A,B,N -> moltiplica B per l'immediato N e lo salva in A
15) MUL A,B -> opera la moltiplicazione in numero naturale fra B e A e salva il risultato in A
16) IMUL A,B -> opera la moltiplicazione in numero con segno (complemento a 2) fra B e A e salva il risultato in A

17) AND A,B -> opera l'and bit a bit fra A e B e salva in A
18) OR A,B -> opera l'or bit a bit fra A e B e salva in A
19) XOR A,B -> opera lo xor bit a bit fra A e B e salva in A
20) XCHG A,B -> scambia i contenuti dei due registri A e B

21) SAL/SHL A -> shifta verso sinistra di una posizione il valore di A
22) SAL/SHL A, n -> shifta verso sinistra di n posizioni il valore di A
23) SAL/SHL A, CL -> shifta verso sinistra di tot posizioni identificate dal numero degli ultimi 5 bit di CL il valore di A
24) SAR A -> shifta verso destra di una posizione il valore di A, mantenendo il segno
25) SHR A -> shifta verso destra di una posizione il valore di A, estendendo con zeri
26) ROR -> shift circolare a destra che pone il bit che si perde come bit più significativo e lo mette anche nel carry flag
27) ROL -> shift circolare a sinistra che pone il bit che si perde come bit meno significativo e lo mette anche nel carry flag

////// è utile utilizzare ROR e ROL quando l'operazione deve essere reversibile

1) LOOP etichetta -> incrementa un registro contatore, non altera i flags,se si verifica una specifica condizione allora salta al codice di "etichetta" e funziona fin quando non viene più richiamata dal codice

SALTI:

1) JMP etichetta -> istruzione di salto incondizionato (avviene senza condizioni) che va alla parte di codice di "etichetta"
2) JA/JNBE/JG/JNLE etichetta -> salta al codice di "etichetta" se il primo operando è maggiore del secondo
3) JAE/JNB/JGE/JNL etichetta -> salta al codice di "etichetta" se il primo operando è maggiore o uguale del secondo
4) JB/JNAE/JL/JNGE etichetta -> salta al codice di "etichetta" se il primo operando è minore del secondo
5) JBE/JNA/JLE/JNG etichetta -> salta al codice di "etichetta" se il primo operando è minore o uguale del secondo
   JE/JZ etichetta -> salta al codice "etichetta" se i due operadi sono uguali

Numeri considerati come naturali (senza segno)

STAMPA DI UNA STRINGA:

```
mov EDX, Lenght ; in Length si trova la lunghezza della stringa
mov ECX, Stringa ; in Stringa si trova la stringa da stampare
mov EAX, 4 ; codice di stampa
mov EBX, 1 ; codice dello schermo su cui stampare
int 0x80 ; codice di interruzione
```

CHIUSURA DI UN PROGRAMMA:

```
mov EAX, 0 ; comando di uscita
mov EBX, 1 ; valore di uscita se corretto
int 0x80 ; codice di interruzione
```