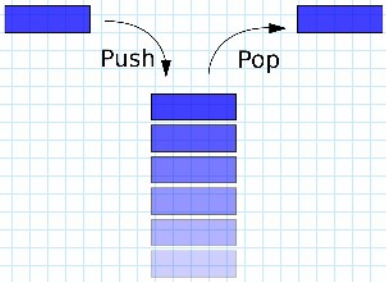
CLASSE 1 / STACK

lunedì 21 novembre 2022 22:37

ISTRUZIONI DI SPOSTAMENTO DATI

Ma per capire questa classe dobbiamo introdurre un altro concetto;
Ovvero il concetto di **STACK DI PROGRAMMA!**
IL PROCESSORE HA UN NUMERO LIMITATO DI REGISTRI.
NOI POSSIAMO DICHIARARE, OVVIAMENTE, PIÙ di 16 VARIABILI.
SE HO 16 REGISTRI E DICHIARO PIÙ DI 16 VARIABILI DOVE LE METTO LE VARIABILI AGGIUNTIVE?
MI SCRIVO IL CONTENUTO di una variabile sullo STACK.



- Un programma potrebbe avere bisogno di gestire tante variabili o variabili molto grandi
- È una struttura dati di tipo *LIFO* (Last-In First-Out): il primo elemento che può essere prelevato è l'ultimo ad essere stato memorizzato
- Si possono effettuare due operazioni su questa struttura dati:
  - *push*: viene inserito un elemento sulla sommità (*top*) della pila
  - *pop*: viene prelevato l'elemento affiorante (*top element*) dalla pila
- È possibile utilizzare un'area di memoria come "area di appoggio": lo stack (*pila*) di programma

IL NOSTRO PROCESSORE deve poterla gestire CON delle ISTRUZIONI dedicate!

NELLA NOSTRA architettura Zgu, tutto lo STACK è composto da Quadword.
NON SI PUÒ fare una push di UN singolo BYTE.

[8 byte]

UNO dei registri del NOSTRO PROCESSORE viene dedicato PER RICORDARCI, IN memoria, dove si TROVA l'ultimo elemento che abbiamo INSERITO.

- La cima dello stack è individuata dall'indirizzo memorizzato in un registro specifico chiamato SP (*Stack Pointer*)

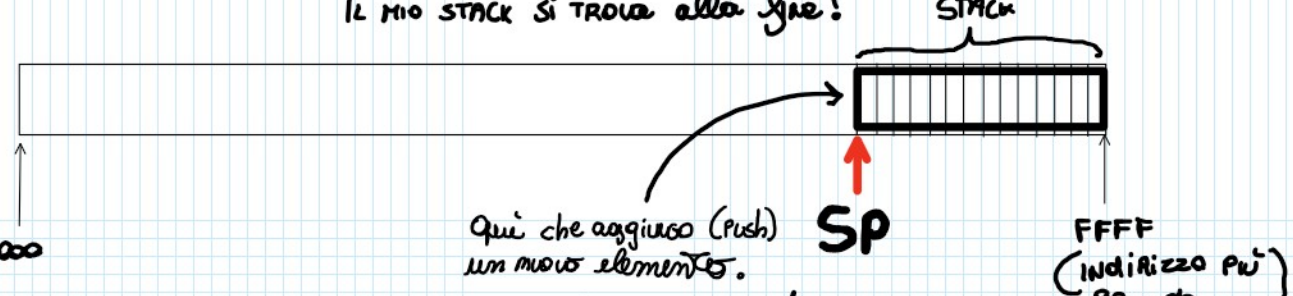
= Lo **stack pointer** (ESP) è, nelle architetture x86, un registro dedicato alla CPU che contiene l'indirizzo della locazione di memoria occupata dal top dello stack per permetterne le operazioni di push, che lo incrementerà, e di pop, che farà

OGNI VOLTA che aggiungo un elemento, LO STACK POINTER deve essere aggiornato.
OGNI VOLTA che tolgo un elemento, LO STACK POINTER DOVRÀ essere aggiornato.

SP È UN REGISTRO A 16 BIT DEL REGISTRO RSP;

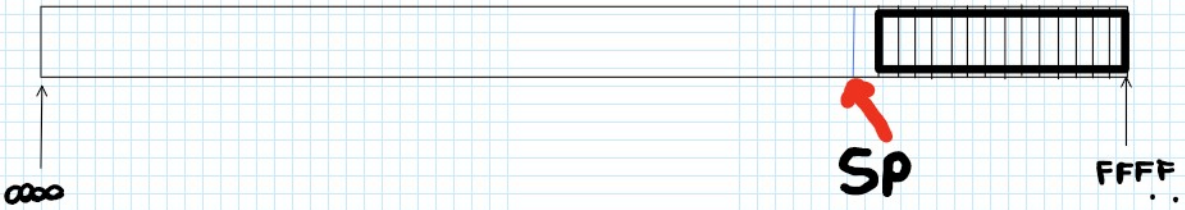
LO STACK NELLE ARCHITETTURE VIENE MEMORIZZATO AL CONTRARIO!

SE PRENDO la memoria, che abbiamo detto essere un NASTRO lunghissimo:



sullo STACK di programma, ci aggiungo della quadword mentre sto eseguendo.
L'ultima quadword aggiunta viene puntata dallo STACK POINTER.

eseguendo.
L'ultima quadword aggiunta viene puntata dallo STACK POINTER.



SE PUSHO dentro lo STACK, IN REALTÀ LO STACK POINTER DECREMENTA! PERCHÈ VERSO SINISTRA È DECRESCENTE.

LO STACK CRESCE SE SP diminuisce.

QUINDI SIGNIFICA CHE LO STACK È RIBALTATO;

- Modificare il valore di SP significa perdere il riferimento alla cima dello stack, e quindi a tutto il suo contenuto
- Lo stack "cresce" se il valore contenuto in SP diminuisce, "decresce" se il valore contenuto in SP cresce
- Lo stack è posto in fondo alla memoria e cresce "all'indietro"

FATTA questa PREMESSA:

ISTRUZIONI DI SPOSTAMENTO DATI CLASSE 1

ovviamente qua ci va il suffisso, dell'estensione ✓✓✓

(?) PRESUNTO

contenuto dei flag!

Tipo	Mnemonico	Operandi	O S Z P C	Descrizione
0	mov	B, E	- - - - -	Fa una copia di B in E
1	movsX	E, G	- - - - -	Fa una copia di E in G con estensione del segno
2	movzX	E, G	- - - - -	Fa una copia di E in G con estensione dello zero
3	lea	E, G	- - - - -	Valuta la modalità di indirizzamento, salva il risultato in G
4	push	E	- - - - -	Copia il contenuto di E sulla cima dello stack
5	pop	E	- - - - -	Copia il contenuto della cima dello stack in E
6	pushf	– (OPERANDO IMPLICATO IL FLAGS)	- - - - -	Copia sulla cima dello stack il registro FLAGS
7	popf	–	- - - - -	Copia nel registro FLAGS il contenuto della cima dello stack
8	movs → STRING	[STRCPY IN C++]	- - - - -	Esegue una copia memoria-memoria
9	stos → STRING	–	- - - - -	Imposta una regione di memoria ad un dato valore

→ il numero è già rappresentato come C.A.2.

MOV

MOV l'operando sorgente può essere una costante, un REGISTRO o UN INDIRIZZO di memoria!
La destinazione è un REGISTRO o un'area di memoria.

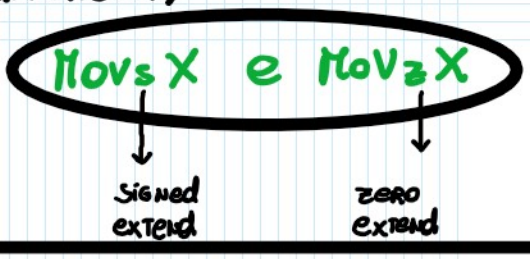
Voglio convertire dei dati da più grande a più piccolo, ma vogliamo fare anche l'opposto. Potrei decidere di fare anche l'opposto.

Che accortezza dobbiamo fare per questo tipo di conversione?

Se ho un numero a 8 bit di grandezza e lo voglio convertire nello <u>stesso</u> numero, però a 16 bit, posso aggiungere degli zero prima, FINTANTO CHE IL MIO NON È UN NUMERO NEGATIVO (-6), PERCHÈ IN C.A.2 INIZIANO TUTTI CON 1.

Quindi quando io faccio la conversione dalla taglia più piccola alla taglia più grande, devo dare un contesto a quel valore, ossia devo dire al processore se il valore che sto convertendo è INTERO SEGNATO O INTERO NON SEGNATO.
SE È UN INTERO NON SEGNATO E IL BIT PIÙ SIGNIFICATIVO È 1, quello è UN NUMERO GRANDE MA POSITIVO, quindi DEVO INSERIRE TUTTI ZERI.
SE È UN INTERO SEGNATO (NEGATIVO) DEVO INSERIRE TUTTI UNI, ALTRIMENTI STO CAMBIANDO IL SEGNO.

COME FACCIAMO A DARE IL CONTESTO? DOBBIAMO USARE DELLE ISTRUZIONI DIFFERENTI;



Movb (%rax, %rcx, 2), %al

vado IN memoria, a quello INDIRIZZO e leggo IL dato.

Potremmo essere INTERESSATI a sapere quale è questo INDIRIZZO nel NOSTRO PROGRAMMA? CERTO!

Io POTREI DIRE al MIO PROCESSORE di calcolare questo INDIRIZZO sorgente e andare a SCRIVERE IL RISULTATO IN UN Registro.

lea

load Effective Address = lea

PER fare questo LA MOV diventa lea.

lea (%rax, %rcx, 2), %rax

chiaramente IL REGISTRO destinazione dovrà essere a 64 bit.

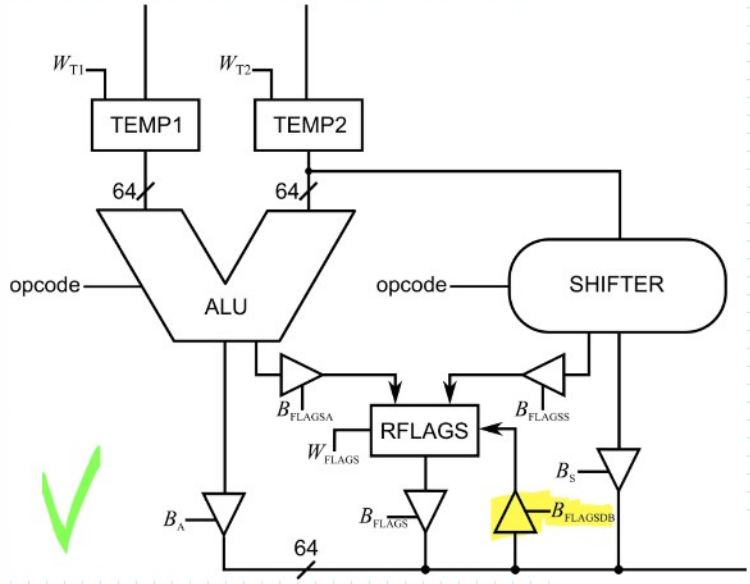
NON ACCEDE AL DATO IN MEMORIA;

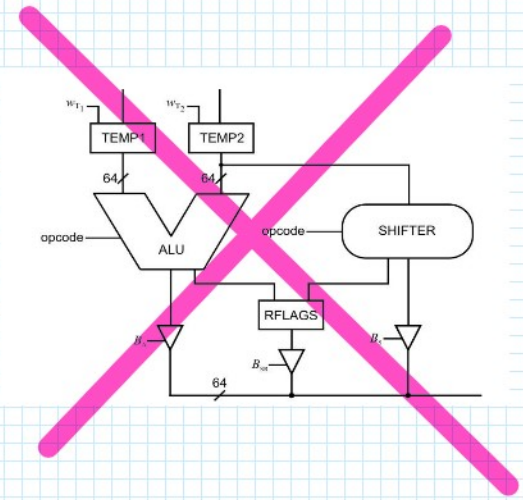
MA PER COME ABBIAMO definito LA NOSTRA ARCHITETTURA, LE POSSO ESEGUIRE TUTTE QUESTE ISTRUZIONI?

Guardiamo la 6 e la 7.

FIN ORA IL REGISTRO FLAGS POTEVA ESSERE SCRITTO SOLO DALLA ALU E DALLO SHIFTER.

DEVO collegare IL dataBUS INTERNO AL REGISTRO flags, mettendo un BUFFER -THREE STATE.





DATA BUS INTERNO
↓
I valori che arrivano dalla memoria corrono sul dataBUS INTERNO!

ESTENDIAMO LE ISTRUZIONI: MovsX e MovzX.

Istruzione (Accede al contenuto del Registro)	Tipo di conversione
`movsbw %al, %ax`	Estendi il segno da byte a word
`movsbl %al, %eax`	Estendi il segno da byte a longword
`movsbq %al, %rax`	Estendi il segno da byte a quadword
`movswl %ax, %eax`	Estendi il segno da word a longword
`movswq %ax, %rax`	Estendi il segno da word a quadword
`movslq %eax, %rax`	Estendi il segno da longword a quadword

X NON RAPPRESENTA UN SINGOLO SUFFISSO, MA 2.

- L'istruzione movzX supporta le stesse combinazioni di suffissi
- I nomi dei registri virtuali devono essere coerenti con i suffissi

- Solo da più piccola a più GRANDE;
- Perché da più grande a più piccola mi basta leggere meno byte quando faccio una mov.

Voce 010 1:12:30