1 dicembre 2016

HD → MU → MP — architettura pipeline (RISC) / a singolo bus interno (CISC) { codifica, exe, fetch } hardware

prestazioni ↓

quali istruzioni di un CISC devono essere incorporate in un RISC? → legge di Amdahl ottimizzo le istruzioni di uso più frequente in un programma

Considereremo software CISC e un processore INTEL 8086.

INTEL 8086 → esecuzione concorrente di più programmi { - parallelizzazione di fetch ed esecuzione; - memoria suddivisa in segmenti }

ne analizzeremo il MODELLO di PROGRAMMAZIONE (cioè quanto dell'architettura è visibile al programmatore assembler).

dimensione della memoria: 1MB
bus indirizzi: 20 bit
bus dati: 16 bit

Inoltre è Byte-Addressable: è cioè possibile indirizzare il singolo byte o 1 word (2 byte).

MUL op → c'è un operatore (il registro AX) è implicito.
Prende il contenuto del registro AX, lo moltiplica per il dato specificato da op: il risultato va in una coppia di registri

(DX, AX) ← AX (16 bit) × op (16 bit)
(DX, AX): 32 bit

Es. In generale, per evitare problemi di overflow, il risultato di A (n bit) × B (n bit) sta in 2n bit per evitare overflow.

Nel processore ci sono vari tipi di registri.

GENERAL PURPOSE REGISTERS

	0 – 7	8 – 15
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

AX - Accumulator
BX - Base → è l'unico a poter contenere un indirizzo
CX - Counter
DX - Data → utilizzato per il trasferimento dati

I registri possono essere utilizzati per i dati o come contatore, ma esistono istruzioni particolari che utilizzano

i registri nel modo scritto e destro

Come si implementa un ciclo nell'8086?

```
    MOV  CX, n;
CICLO: _____
       _____
         :
       _____
    LOOP CICLO; equivale { DEC CX ↔ SUB CX, 1
                         { JNZ CKLO
```

Il registro CX è inizializzato a n (valore di memoria o dato immediato), che rappresenta il numero di istruzioni
Istr. di TRASFERIMENTO DATI

CICLO è rappresentato come un salto relativo (numero in C_2 che rappresenta il Δ fra l'istruzione successiva e quella corrispondente all'etichetta CICLO). → non è l'indirizzo di CICLO

L'ASSEMBLATORE svolge il medesimo compito del COMPILATORE, ma a basso livello: prende le istr. nel linguaggio che sto scrivendo e le trasforma in istr. in linguaggio macchina.

I registri possono essere utilizzati anche a metà.

	AH	AL
① AX	11110000	10101011

Nel processore le istruzioni logiche sono eseguite più velocemente delle istruzioni aritmetiche

XOR AH, AH; equivale a MOV AH, 0

↓
fa lo XOR bit a bit

Dopo l'esecuzione di ①

	AH	AL
AX	00000000	10101111

AND AL, 3A (h) qualche che è scritto in esadecimale

↓ ↘
0011 1010

```
  1 0 1 0 1 1 1 1
+ 0 0 1 1 1 0 1 0
-----------------
  1 1 0 0 1 0 0 0
```

Supponiamo di voler aggiungere un 1 al penultimo bit di AH

OR AH, 02 (h); maschera da usare
↑ 0000 0010 esadecimale

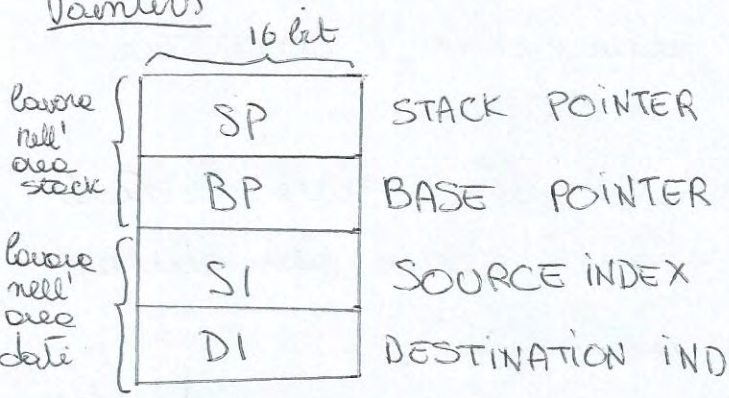
equivale
↓
ORAH, 0000 0010 (B) — indica che è in binario

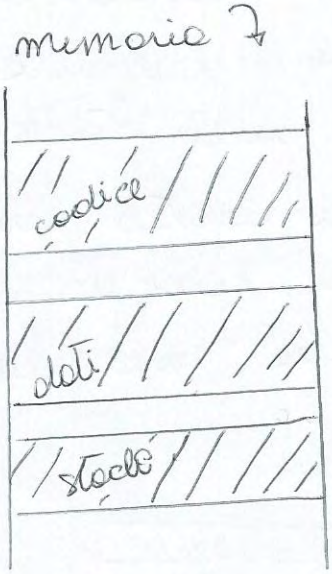
Il linguaggio assembler non è CASE SENSITIVE: non distingue tra

VAR
var
Var

Nel processore sono presenti alcuni registri

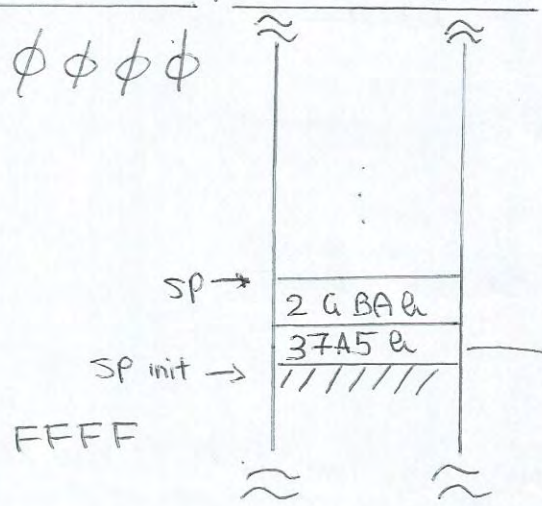
Pointers





I registri pointers non sono splittabili.

Stack nel processore 8086



All'inizio lo stack pointer punta alla prima locazione non dello stack.
Generalmente SP punta all'ultima locazione scritta

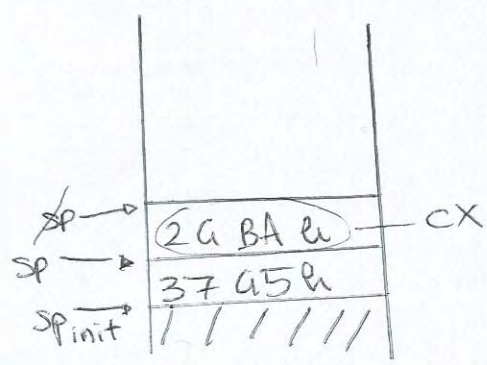
BOTTOM OF STACK locazione a partire dalla quale si inizia a scrivere

scrittura nello stack PUSH — push op; stack ← op
prelevare un dato dallo stack POP — pop op; stack → op

Non si può fare il push di un dato immediato

```
mov AX, 37A5 h
PUSH AX
POP CX → POP scrive il contenuto puntato da SP nella destinazione (CX) e poi incrementa SP di 2
```

il valore 2CBA h non viene cancellato, ma verrà sovrascritto al push successivo (fig 2)



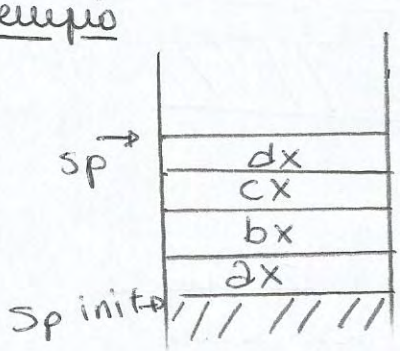
push op ↔
```
SUB SP, 2
MOV [SP], op
```

Proprietà dello stack

1) cresce verso l'alto
2) SP punta all'ultima locazione scritta
3) si possono trasferire solo parole (16 bit)
4) PUSH decrementa SP di 2 e poi scrive l'operatore sorgente nello stack.

BP è un registro "ausiliario": serve ad utilizzare lo stack come se fosse un'area RAM qualsiasi (serve ad andare a prendere "quel che ci pare" in memoria).

esempio

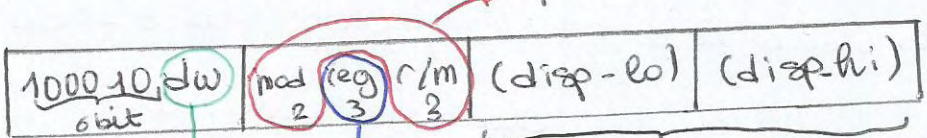



```
push dx
push bx
push cx
push dx
MOV BP, SP
ADD BP, 6
MOV dx, [BP]
```

Supponiamo di voler spostare il contenuto del registro dx nel registro dx

es. Le operazioni `MOV dx, [Si+2]` `MOV dx, [BP+6]` sono lecite ma non si può fare [SP].

Consideriamo le operazioni del tipo MOV reg/mem to/from reg (c'è almeno un registro)



d,w sono bit che completano l'informazione sui modi di indirizzamento

Specifica il registro

dipendono dal modo di indirizzamento

d dice se il registro è sorgente (0) o destinazione (1)

w indica se il trasferimento è per una word (1) o solo per un byte (0) → BYTE ADDRESSABLE

esempi

1) MOV dx, [Si + 2] codifica in macchina

	mod	reg	rm	DB →
10001011	01	010	100	00000010
8 B		5	4	0 2

codifica in esadecimale

DB → scostamento +2

quest'istruzione richiede 3 byte

2) AND Var, 37A5 h

6 byte

DI6 diretto di memoria

Come mai le istruzioni sono lunghe 20 bit anche se di base sono lunghe 2 byte?

Organizzazione di memoria

MOV AX, 12 3A h

Motorola 68'000 — Big Endian

Intel 8086 — Little Endian

← 2 byte →	
12 h	3A h
3A h	12 h

conversioni

Come si civa effettivamente l'indirizzo in memoria

Segment Registers

CS (code segment)	→ lavora sul codice
DS (data seg.)	lavorano sui dati
ES (extra-data segment)	
SS (stack segment)	→ lavora sullo stack

16 bit

PA indirizzo fisico (20 bit)

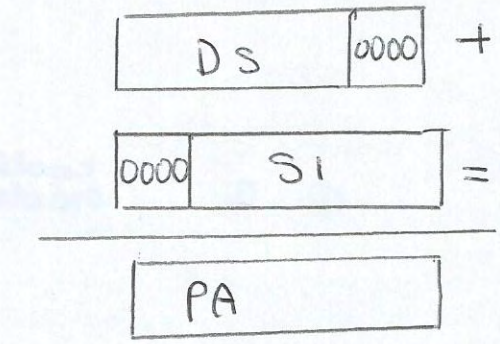
SR segment register 16 bit

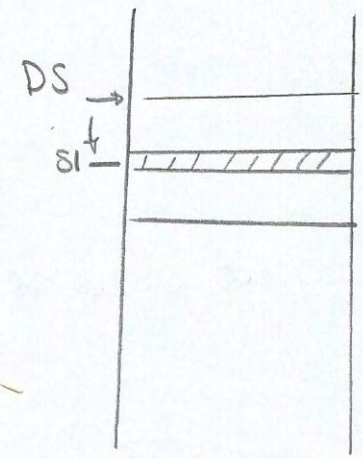
EA effective address 16 bit } 32 bit

EA → indirizzo efficace

PA = SR : EA (è costituito da)

Si lavora con offset a 16 bit, con a 0 a destra del SI e a 0 a sinistra di DS





PC = program counter (IP) : CS

IP = registro a 16 bit

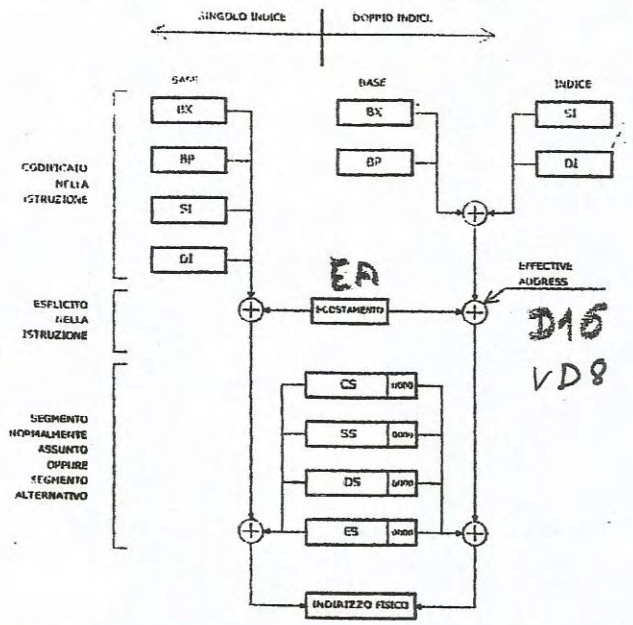


Figura 14.7 - Metodo di calcolo dell'indirizzo di memoria. Si noti che lo scostamento può essere, a seconda dei casi, di 8 o di 16 bit. L'indirizzo effettivo *Effective Address (EA)* è quello calcolato a monte del contributo del registro di segmento, in base al contenuto dell'istruzione. L'indirizzo fisico (lineare) è ottenuto sommando ad EA l'indirizzo di partenza del segmento, cioè il contenuto del registro di segmento moltiplicato per 16.

Instruction type	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6
AND immediate to register/memory	1000000*w*	mod 100 r/m	(disp-lo)	(disp-hi)	data	data if $w = 1$
JE	01110100	IP-inc				
MOV register/memory to/from register	100010*dw*	mod reg r/m	(disp-lo)	(disp-hi)		
PUSH register	01010reg					

	mod					
r/m	00	01	10	11		reg
000	BX+SI	BX+SI+D8	BX+SI+D16	AL	AX	000
001	BX+DI	BX+DI+D8	BX+DI+D16	CL	CX	001
010	BP+SI	BP+SI+D8	BP+SI+D16	DL	DX	010
011	BP+DI	BP+DI+D8	BP+DI+D16	BL	BX	011
100	SI	SI+D8	SI+D16	AH	SP	100
101	DI	DI+D8	DI+D16	CH	BP	101
110	D16	BP+D8	BP+D16	DH	SI	110
111	BX	BX+D8	BX+D16	BH	DI	111
				w=0	*w=1*	

3086 : Codifica di alcune istruzioni e modi di indirizzamento.