Lo stack

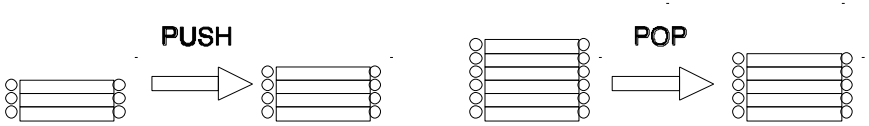
Lo stack è un'area localizzata all'interno della memoria centrale del computer che viene utilizzata per:

⇒ salvare temporaneamente il valore contenuto nei registri;
⇒ poter effettuare le chiamate ai sottoprogrammi (ad es. subroutine in Assembler o funzioni in linguaggio C).

Le locazioni di memoria che costituiscono tale area sono accessibili secondo le regole definite da una struttura dati astratta denominata anch'essa "stack", termine che in italiano può essere tradotto con "pila" o "catasta". Tale struttura dati adotta una politica di gestione detta LIFO (Last In First Out).

La politica di gestione LIFO è la stessa che si adotta quando si lava una pila di piatti: il primo piatto lavato è il primo ad essere posto sul tavolo e quindi l'ultimo ad essere asciugato; viceversa l'ultimo piatto ad essere lavato risulta il primo della pila che si è formata e quindi il primo ad essere asciugato. Quindi, in una struttura di tipo LIFO l'ultimo elemento ad essere introdotto nella struttura stessa (nel nostro esempio: la pila di piatti) è il primo ad essere estratto.

Le operazioni che si possono effettuare sullo stack sono due: l'inserzione di un elemento (PUSH) e l'estrazione di un elemento dalla struttura (POP). Entrambe le operazioni devono essere effettuate sulla cima (o "testa") dello stack: la PUSH inserisce un nuovo elemento sulla cima dello stack, mentre la POP estrae l'elemento presente sulla testa. Non è possibile effettuare alcuna operazione se non sulla testa dello stack.



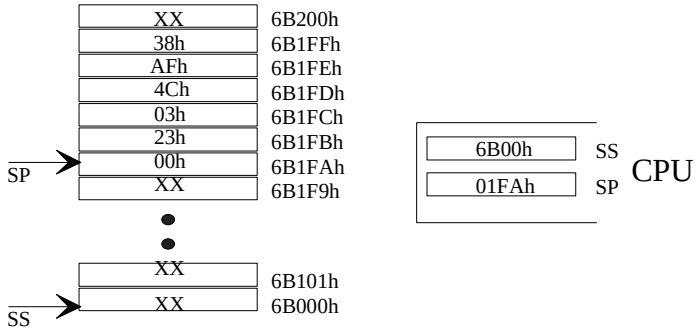
Le operazioni dello stack .

Evidentemente non è possibile estrarre un elemento da uno stack vuoto: quando si tenta di effettuare una POP da uno stack che non presenta elementi al suo interno si incorre in un errore di *STACK UNDERFLOW*. D'altra parte, normalmente allo stack viene assegnata una dimensione massima: quando si cerca di effettuare una PUSH su uno stack il cui numero di elementi è pari alla dimensione massima della struttura, si induce un errore di *STACK OVERFLOW*.

In un sistema gestito da un microprocessore Intel 8086, viene dedicato allo stack un segmento in memoria centrale (*STACK SEGMENT*) che può avere una dimensione massima di 65536 bytes (64 Kbytes); ferma restando la dimensione massima, il numero di bytes che vengono riservati allo stack all'interno di un'applicazione Assembler viene deciso dal programmatore, che deve tenere conto delle esigenze dell'applicazione stessa.

La locazione elementare dello stack è la word: ciò significa che è possibile effettuare una PUSH o una POP di un registro o di una locazione di memoria a 16 bit e **non** di un registro o di una locazione di memoria a 8 bit. La gestione dello stack avviene essenzialmente tramite due registri interni della CPU: il registro SS (*STACK SEGMENT*) e il registro SP (*STACK POINTER*).

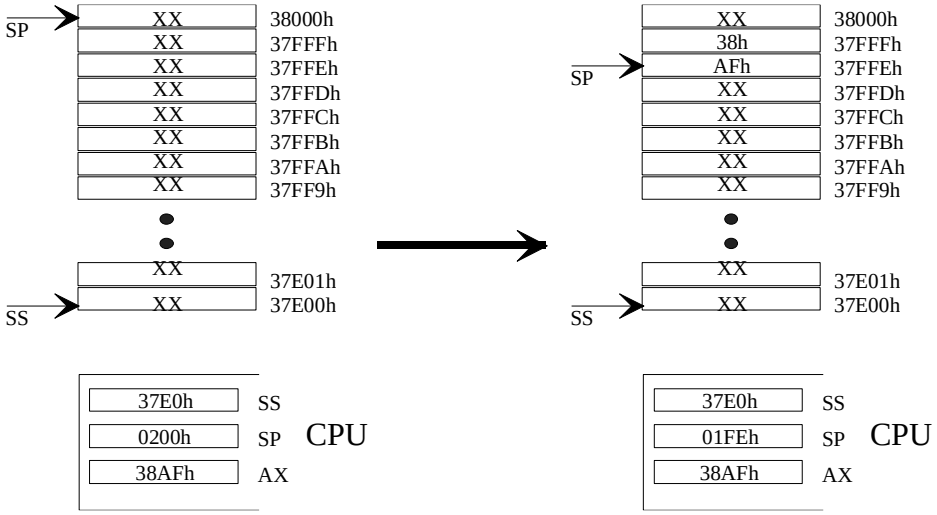
Il registro SS contiene l'indirizzo di inizio del segmento Stack diviso per 16 (obbligatoriamente il segmento Stack deve iniziare da un indirizzo divisibile per 16); il registro SP viene inizializzato alla dimensione dello stack decisa dal programmatore e contiene l'offset dell'ultima word salvata sullo stack (la ***testa*** dello stack), come mostrato dalla figura nell'esempio.



Esempio di localizzazione dello stack in main memory e registri della CPU coinvolti..

Vediamo come vengono eseguite la PUSH e la POP in un sistema gestito da un microprocessore 8086.

Eseguiamo dapprima la PUSH del registro AX, supponendo che esso contenga il valore 38AFH e che lo stack sia vuoto.



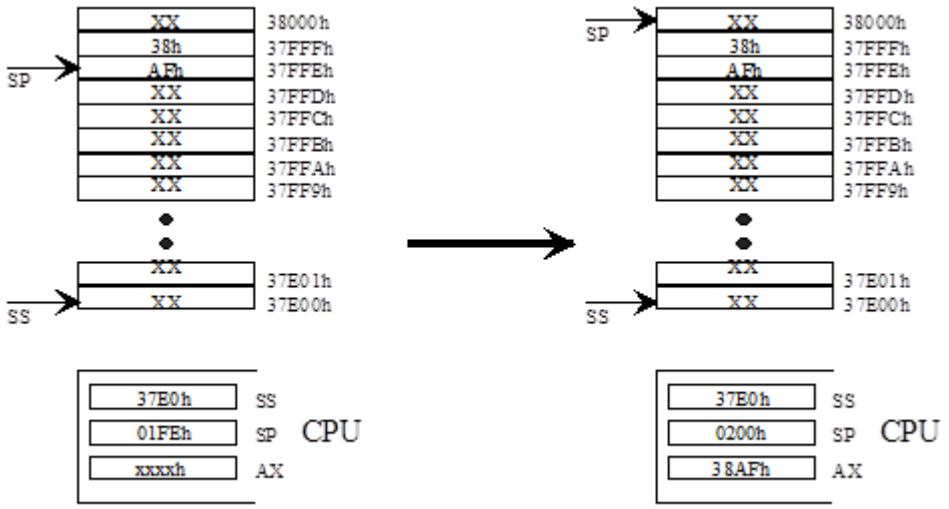
Prima della PUSH AX

Dopo la PUSH AX

L'esecuzione della PUSH implica le seguenti azioni elementari:

1. Il valore contenuto in SP viene decrementato di due; se il nuovo valore di SP è negativo, viene generata una condizione di errore corrispondente a Stack Overflow;
2. Se non è stata generata una condizione di errore, nella word in memoria centrale il cui indirizzo assoluto è determinato dal contenuto di SS moltiplicato per 16 più il contenuto di SP, viene caricato il valore contenuto in AX.

Eseguiamo ora la POP sul registro AX.



Prima della POP AX

Dopo la POP AX

L'esecuzione della POP implica le seguenti azioni elementari:

1. Se il valore contenuto in SP è uguale al valore della dimensione massima dello stack stabilita dal programmatore, viene generata una condizione di errore corrispondente a Stack Underflow. Se non è stata generata tale condizione di errore, nel registro AX viene caricato il valore contenuto nella word in memoria centrale il cui indirizzo assoluto è determinato dal contenuto di SS moltiplicato per 16 più il contenuto di SP.
2. Il valore contenuto in SP viene incrementato di due.

Occorre notare che il valore "estratto" mediante la POP è ancora presente in memoria centrale, ma non è più accessibile, in quanto SP tornerà a "puntare" a quella word con una successiva PUSH, che però avrà l'effetto di "ricoprire" quel valore.

Oltre a consentire il salvataggio temporaneo del valore contenuto nei registri interni della CPU o in altre locazioni della memoria centrale, lo stack ricopre un ruolo fondamentale perchè permette ai programmatori di richiamare sottoprogrammi (subroutine o funzioni).

Una subroutine è costituita da una sequenza di istruzioni che vengono identificate con un nome e che può essere richiamata da un punto qualsiasi del programma. Quando viene richiamata una subroutine, la CPU interrompe

l'esecuzione sequenziale del programma e va ad eseguire la prima istruzione del sottoprogramma; quando tutte le istruzioni della subroutine sono state eseguite, l'esecuzione del programma riprende dalla istruzione immediatamente successiva al richiamo della subroutine.

Lo stack ha la funzione di memorizzare l'indirizzo di ritorno, cioè l'indirizzo dell'istruzione che dovrà essere eseguita immediatamente dopo l'ultima istruzione della subroutine, che nel caso del linguaggio Assembler è sempre un'istruzione RET.

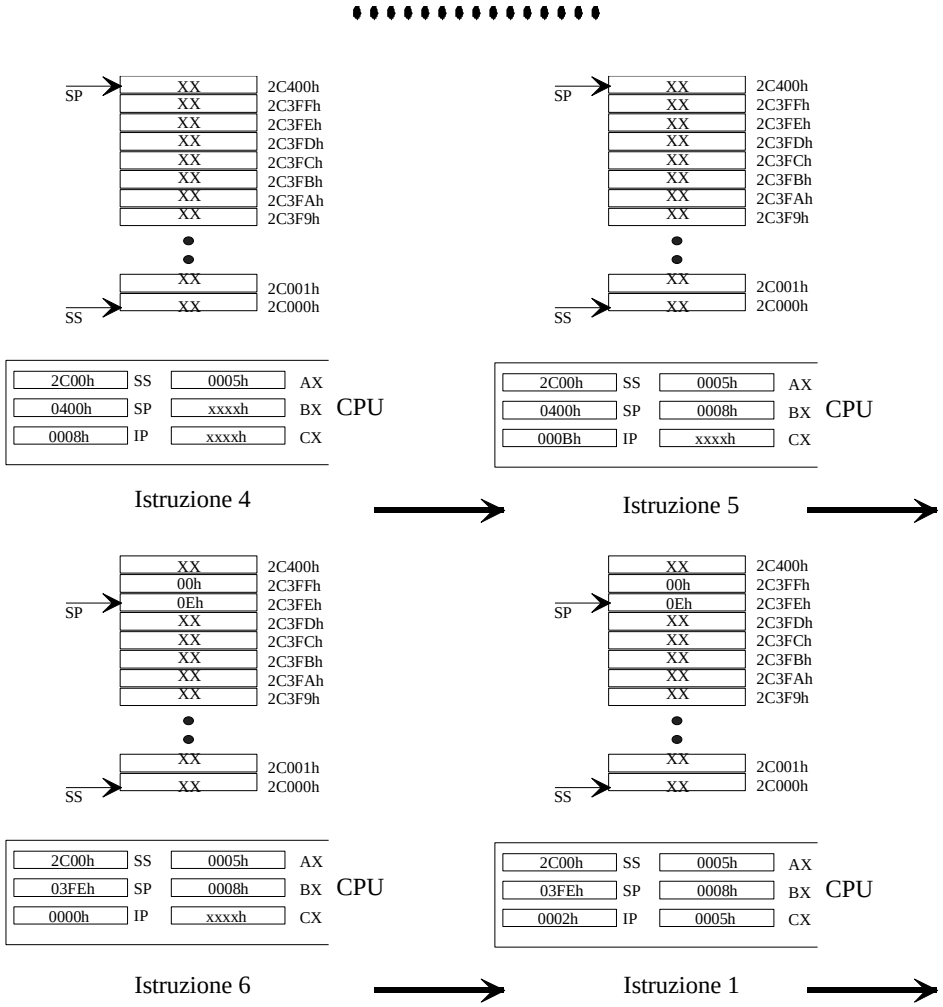
Osserviamo il comportamento dello stack e di alcuni registri della CPU (tra cui il registro IP - Instruction Pointer) sul seguente programma che richiama una subroutine la cui funzione è di calcolare la somma del contenuto dei registri AX e BX e di porre il risultato nel registro CX; le quantità numeriche sulla sinistra rappresentano l'offset della corrispondente istruzione rispetto all'inizio del programma.

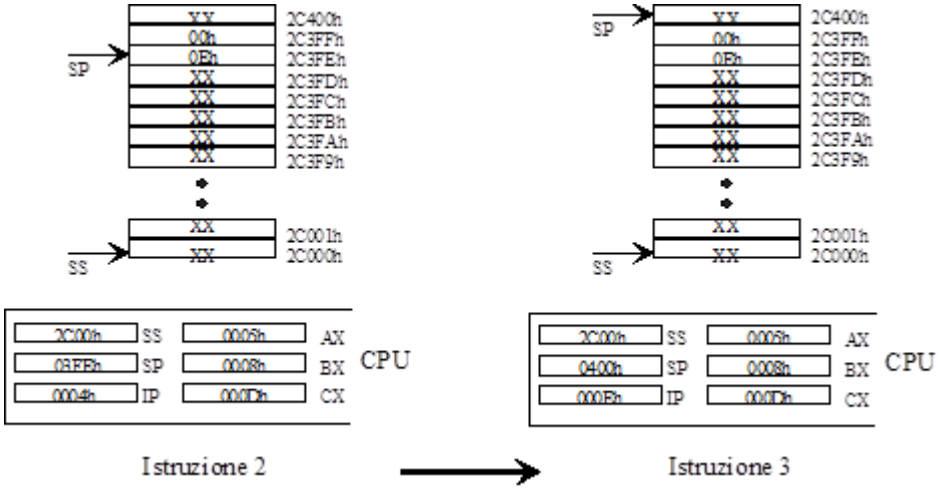
ASSEMBLY-->

All'inizio di ogni riga (ogni riga corrisponde ad un'istruzione) ha una serie di cifre, che definisce l'offset (distanza misurata in byte) dall'istruzione all'inizio del codice

```
                .model small
                .code
0000    somma   proc  near
0000      mov  cx,ax          ; 1
0002      add  cx,bx          ; 2
0004      ret                 ; 3
0005    somma   endp
0005    inizio:
0005      mov  ax,5           ; 4
0008      mov  bx,8           ; 5
000B      call somma          ; 6
000E      mov  ax,4c00h       ; 7
0011      int  21h
        end inizio
```

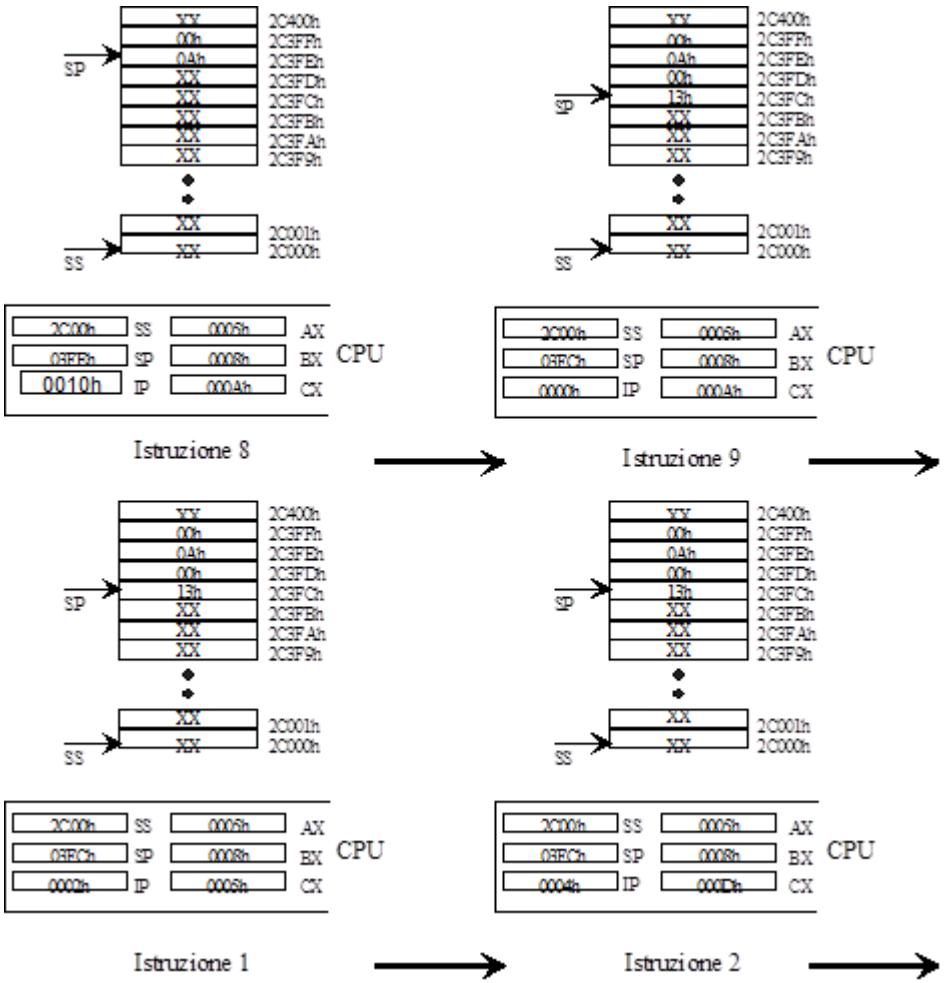
; n --> sono i commenti

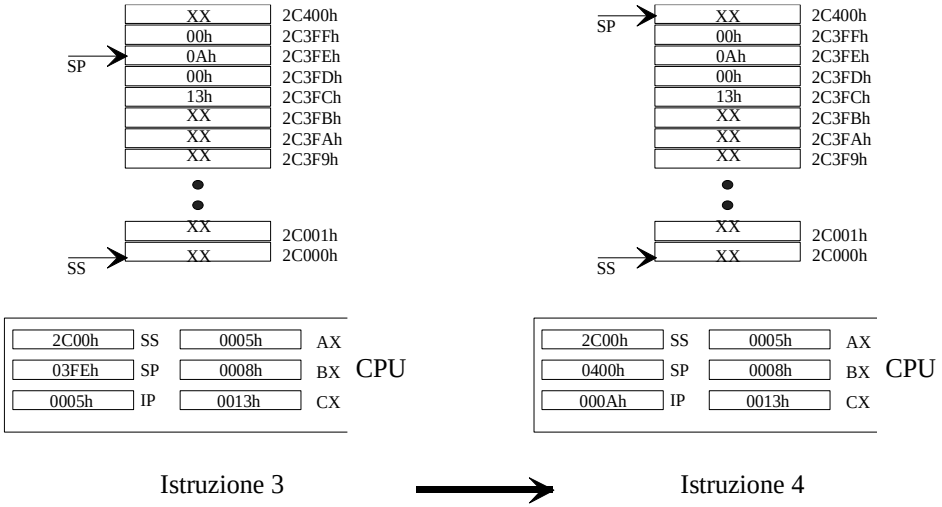




Come si può osservare dal codice precedente, è ovviamente possibile all'interno di uno stesso programma utilizzare tutte le istruzioni che implicano l'uso dello stack (PUSH-POP, CALL-RET). Vediamo però cosa accade se non vengono utilizzate correttamente:

```
                .model small
                .stack 0400h
                .code
0000    somma   proc near
0000      mov  cx,ax          ; 1
0002      add  cx,bx          ; 2
0004      pop  cx             ; 3
0005      ret                 ; 4
0006    somma   endp
0006    inizio:
0006      mov  ax,5           ; 5
0009      mov  bx,8           ; 6
000C      mov  cx,10          ; 7
000F      push cx             ; 8
0010      call somma          ; 9
0013      mov  ax,4c00h       ; 10
0016      int  21h
                end inizio
```





Istruzione 3 → Istruzione 4

L'esecuzione del programma, a questo punto, non prosegue correttamente, poichè nel registro IP, che indica l'offset della successiva istruzione da eseguire, è caricato un valore che non corrisponde ad alcuna istruzione del programma. L'errore è dovuto al fatto che la PUSH e la POP corrispondenti non si trovano entrambi o nel programma principale o all'interno della stessa subroutine.

Riassumendo, per una corretta gestione dello stack occorre osservare le seguenti regole:

1. Ad ogni PUSH deve corrispondere una e una sola POP e viceversa.
2. Il registro o l'area di memoria argomento di una PUSH devono comparire come argomento della POP corrispondente. Anche se questa regola potrebbe anche non comportare errori, è buona norma di programmazione attenersi ad essa.
3. La PUSH e la POP corrispondenti devono entrambe trovarsi o nel programma principale o all'interno della stessa subroutine.

È buona norma, quando si scrive una subroutine, salvare sullo stack il contenuto dei registri utilizzati dalla subroutine e ripristinare con le opportune POP il loro valore prima di far tornare il controllo al programma principale.

UTILIZZO AVANZATO DELLO STACK DA PARTE DI PROGRAMMI ASSEMBLER

Lo stack viene anche utilizzato per il passaggio di parametri alle subroutines.

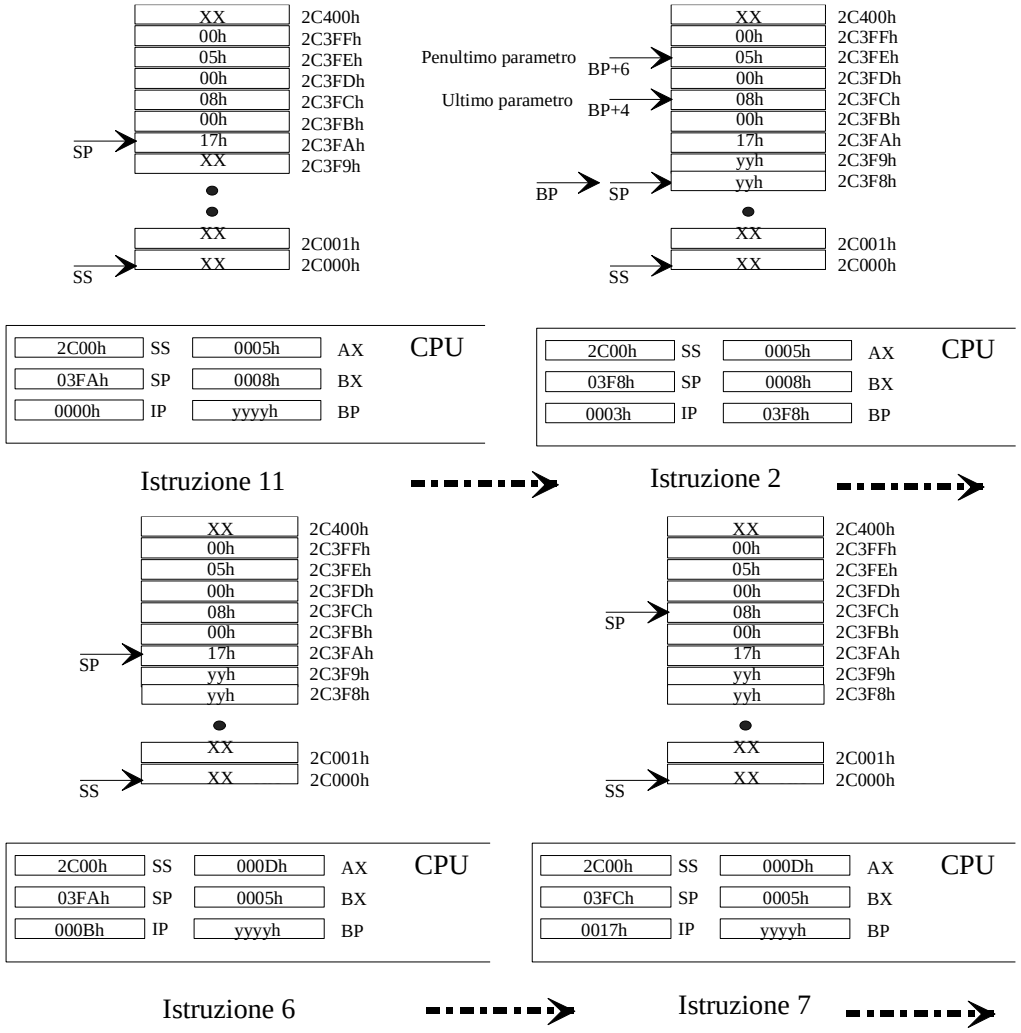
PASSAGGIO PARAMETRI AD UNA SUBROUTINE

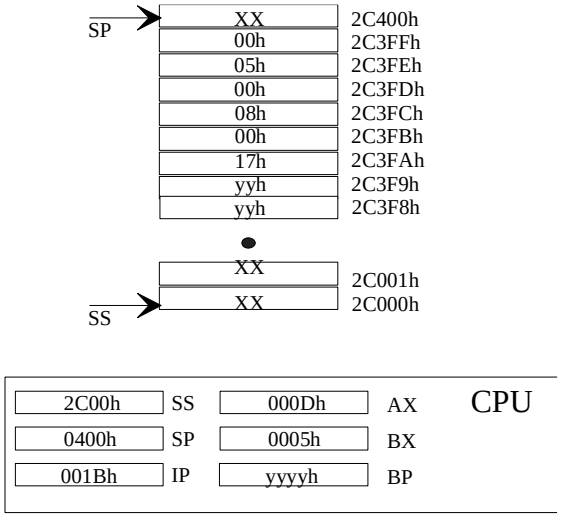
Il passaggio dei parametri ad una subroutine avviene utilizzando lo stack. Come noto il passaggio dei parametri può essere per *valore* o per *referenza*. Nel passaggio dei parametri per valore, attraverso lo stack viene passato alla subroutine il valore del dato. Quando un parametro viene passato per referenza, attraverso lo stack si passa alla subroutine un indirizzo di un'area di memoria (o meglio un offset) che contiene il dato parametro della subroutine; pertanto tutte le modifiche che vengono fatte su quel parametro cambiano il contenuto dell'area localizzata a quell'indirizzo. Osserviamo ora cosa significa passare un parametro per valore e un parametro per referenza e come interviene lo stack nel passaggio dei parametri.

PASSAGGIO PARAMETRI PER VALORE AD UNA SUBROUTINE

```
                        .model small
                        .stack 200h
                        .code
0000                    somma       proc  near
0000  55                        push  bp          ; 1
0001  8B EC                     mov   bp,sp       ; 2
0003  8B 46 04                  mov   ax,[bp+4]   ; 3
0006  8B 5E 06                  mov   bx,[bp+6]   ; 4
0009  03 C3                     add   ax,bx       ; 5
000B  5D                        pop   bp          ; 6
000C  C3                        ret               ; 7
000D                    somma       endp
000D                    inizio:
000D  B8 0005                   mov   ax,5        ; 8
0010  50                        push  ax          ; 9
0011  BB 0008                   mov   bx,8        ; 10
0014  53                        push  bx          ; 11
0015  E8 0000 R                 call  somma       ; 12
0018  83 C4 04                  add   sp,4        ; 13
001B  B8 4C00                   mov   ax,4c00h    ; 14
001E  CD 21                     int   21h         ; 15
                        end inizio
```

Questo semplice programma richiama una subroutine che somma due valori passati come parametro attraverso lo stack e restituisce il risultato nel registro AX. Come possiamo notare, i due valori che devono essere sommati sono valori espressi su 16 bit.





Istruzione 13

Si noti l'uso del registro BP. Il registro BP (Base Pointer) agisce nel segmento stack e pertanto l'offset in esso contenuto è calcolato rispetto all'inizio dello Stack segment. Lo scopo di tale registro è quello di "fissare" una locazione all'interno dello stack in modo tale da poter reperire senza difficoltà gli eventuali parametri della subroutine indipendentemente da successive variazioni del registro SP che possono avvenire a causa di PUSH o POP eseguite all'interno della subroutine stessa.

Se il programmatore ha l'accortezza di scrivere subito all'inizio della subroutine la sequenza di istruzioni PUSH BP - MOV BP,SP l'ultimo parametro che è stato caricato sullo stack sarà individuato da un offset pari alla somma del valore contenuto nel registro BP più 4; gli altri parametri saranno collocati nelle aree di memoria individuate dagli offset [BP+6], [BP+8] e così via.

Infatti, come si può vedere dalla simulazione precedente, sullo stack vengono caricati prima i parametri, poi l'indirizzo di ritorno della subroutine e quindi il "vecchio" valore di BP salvato sullo stack.

Quando la subroutine ha terminato la sua esecuzione occorre eliminare dallo stack i parametri. Per fare ciò si possono eseguire tante POP quanti sono i parametri; oppure, come nell'esempio precedente, semplicemente aggiornare

il registro SP aggiungendo allo stesso un valore pari al numero dei parametri * 2 (con 1 parametro ADD SP,2; con 2 ADD SP,4; con 3 ADD SP,6 e così via).

PASSAGGIO PARAMETRI PER REFERENZA AD UNA SUBROUTINE

Il seguente programma carica successivamente in ogni locazione di due vettori lunghi rispettivamente 10 e 20 byte l'offset della locazione stessa rispetto all'inizio del vettore cui appartiene. Per eseguire tale compito, il main richiama per due volte una subroutine cui passa come parametri la lunghezza del vettore (per valore) e l'offset del vettore sul quale effettuare il caricamento (per referenza).

```
                        .model small
                        .stack 200h
                        .data
0000  000A[ ?? ]        vett1 db 10 dup (?)
000A  0014[ ?? ]        vett2 db 20 dup (?)
                        .code
0000                    loadvett    proc  near
0000  55                            push  bp
0001  8B EC                         mov   bp,sp
0003  51                            push  cx
0004  8B 46 06                      mov   ax,[bp+6]
0007  8B 5E 04                      mov   bx,[bp+4]       ; istruzione 1
000A  B9 0000                       mov   cx,0
000D                    loopload:
000D  88 0F                         mov   [bx],cl         ; istruzione 2
000F  43                            inc   bx
0010  FE C1                         inc   cl
0012  3B C8                         cmp   cx,ax
0014  72 F7                         jb    loopload
0016  59                            pop   cx
0017  5D                            pop   bp
0018  C3                            ret
0019                    loadvett    endp
0019                    inizio:
0019  B8 ---- R                     mov   ax,@data
001C  8E D8                         mov   ds,ax
001E  B8 000A                       mov   ax,10           ; lunghezza 1.o vettore (1.o parametro)
0021  50                            push  ax
0022  BB 0000 R                     mov   bx,offset vett1 ; offset 1.o vettore (2.o parametro)
0025  53                            push  bx
0026  E8 0000 R                     call  loadvett
0029  83 C4 04                      add   sp,4
002C  B8 0014                       mov   ax,20           ; lunghezza 2.o vettore (1.o parametro)
002F  50                            push  ax
0030  BB 000A R                     mov   bx,offset vett2 ; offset 2.o vettore (2.o parametro)
0033  53                            push  bx
0034  E8 0000 R                     call  loadvett
0037  83 C4 04                      add   sp,4
003A  B8 4C00                       mov   ax,4c00h
003D  CD 21                         int   21h
                                    end inizio
```

Per quanto riguarda la gestione dello stack non cambia nulla rispetto al passaggio dei parametri per valore; il registro BP viene utilizzato nello stesso modo, i parametri sono reperiti senza differenze rispetto ai parametri per valore così come lo stack viene "svuotato" con la stessa tecnica.

Ciò che cambia è il contenuto del parametro. Come evidenziato dal passaggio del parametro e dalle istruzioni 1 e 2, nel parametro non viene caricato un "valore" ma un offset, quindi un indirizzo. Perciò è necessario che nella subroutine il parametro per referenza debba essere caricato nel registro base BX o nei registri indice SI e DI, in modo tale da poter far riferimento alle locazioni di memoria individuate da tali offset.