Chapitre 5 : Les sous Programmes

I. Introduction :

Les sous programmes jouent en assembleur le role des procédures et des fonctions dans le langage haut niveau, afin de facilité des programmes ecrits en assembleur et d'optimiser leurs codes.

II. Appel de sous-programmes :

Pour éviter la répétition d'une même séquence d'instructions plusieurs fois dans un programme, on rédige la séquence une seule fois en lui attribuant un nom (au choix) et on l'appelle lorsqu'on en a besoin.

Le programme appelant est le **programme principal**. La séquence appelée est un **sous-programme** ou **procédure.**

Ecriture d'un sous-programme :

```
nom_sp   PROC
           ⋮   ← instructions du sous-programme
         ret   ← instruction de retour au programme principal
nom_sp   ENDP
```

Un sous programme est définie dans le segment code juste avant l'étiquette « main » ou après la fonction retour DOS.

Remarque : une procédure peut être de type **NEAR** si elle se trouve dans le même segment ou de type **FAR** si elle se trouve dans un autre segment.

Exemple : ss_prog1 **PROC NEAR**

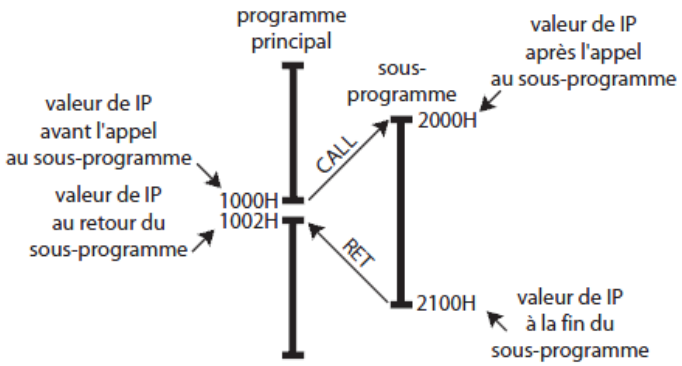
Ss_prog2 **PROC FAR**

Appel d'un sous-programme par le programme principal : **CALL** procédure

```
     ⋮    ← instructions précédant l'appel au sous-programme
call nom_sp ← appel au sous-programme
     ⋮    ← instructions exécutées après le retour au programme principal
```

Lors de l'exécution de l'instruction CALL, le pointeur d'instruction IP est chargé avec l'adresse de la première instruction du sous-programme.

Lors du retour au programme appelant, l'instruction suivant le CALL doit être exécutée, c'est-à-dire que IP doit être rechargé avec l'adresse de cette instruction.



Avant de charger IP avec l'adresse du sous-programme, l'adresse de retour au programme principal, c'est-à-dire le contenu de IP, est sauvegardée dans une zone mémoire particulière appelée **pile**. Lors de l'exécution de l'instruction **RET**, cette adresse est récupéré à partir de la **pile** et rechargée dans IP, ainsi le programme appelant peut se poursuivre.

III. Fonctionnement de la pile :

La pile est une zone mémoire dont le processeur se sert pour stocker des données temporaire comme l'adresse de retour dans le cas d'appel des sous programme ou pour échanger les données entre programme et sous programme.

La pile est une zone mémoire fonctionnant en mode LIFO (Last In First Out : dernier entre, premier sorti).

Deux opérations sont possibles sur la pile :

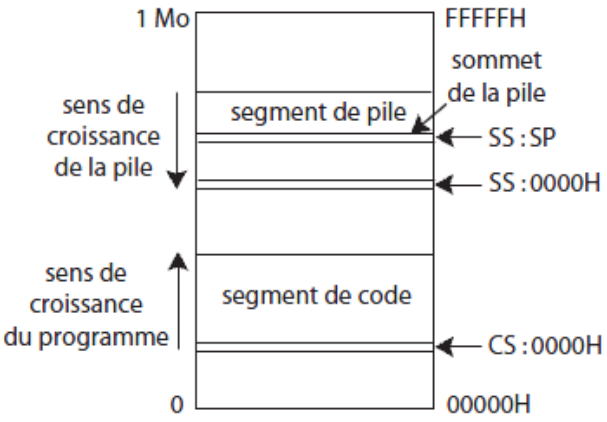
- empiler une donnée : placer la donnée au sommet de la pile ;
- dépiler une donnée : lire la donnée se trouvant au sommet de la pile.

Les données ne peuvent etre empiler ou dépiler que par unité de mots. C'est-à-dire il est impossible de placer un octet seul sur la pile.

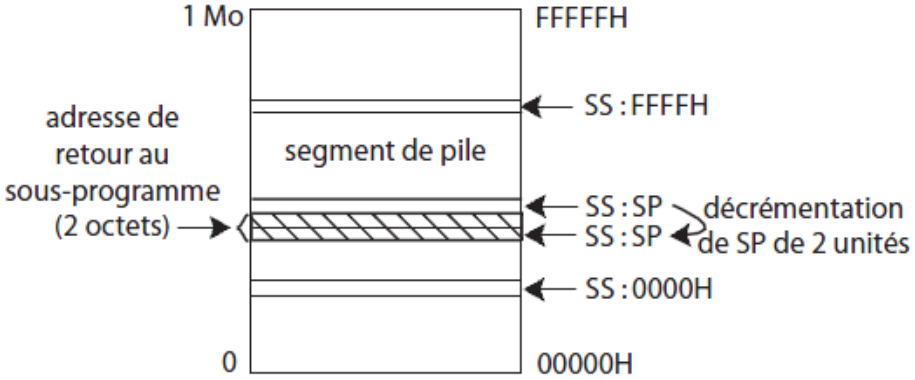
Le sommet de la pile est repéré par un registre appelé pointeur de pile (SP : Stack Pointer) qui contient l'adresse de la dernière donnée empilée.

La pile est définie dans le segment de pile dont l'adresse de départ est contenue dans le registre SS.

Remarque : la pile et le programme croissent en sens inverse pour diminuer le risque de collision entre le code et la pile dans le cas où celle-ci est placée dans le même segment que le code (SS = CS).



Lors de l'appel à un sous-programme, l'adresse de retour au programme appelant (contenu de IP) est empilée et le pointeur de pile SP est automatiquement décrémenté. Au retour du sous-programme, le pointeur d'instruction IP est rechargé avec la valeur contenue au sommet de la pile et SP est incrémenté.



La pile peut également servir à sauvegarder le contenu de registres qui ne sont pas automatiquement sauvegardés lors de l'appel à un sous programme :

- instruction d'empilage : **PUSH** opérande
- instruction de dépilage : **POP** opérande

où opérande est un registre ou une donnée sur 2 octets (on ne peut empiler que des mots de 16 bits). Exemple :

```
push ax         ; empilage du registre AX ...
push bx         ; ... du registre BX ...
push [1100H]    ; ... et de la case mémoire 1100H-1101H
     :
pop  [1100H]    ; dépilage dans l'ordre inverse de l'empilage
pop  bx
pop  ax
```

IV. Utilisation de la pile pour le passage des paramètres :

Pour transmettre des paramètres à une procédure, on peut les placer sur la pile avant l'appel de la procédure, puis celle-ci les récupère en effectuant un adressage basé de la pile en utilisant le registre BP.

Exemple : soit une procédure effectuant la somme de deux nombres et retournant le résultat dans le registre AX :

- programme principal :

```
        mov  ax,200
        push ax         ; empilage du premier paramètre
        mov  ax,300
        push ax         ; empilage du deuxième paramètre
        call somme      ; appel de la procédure somme
```

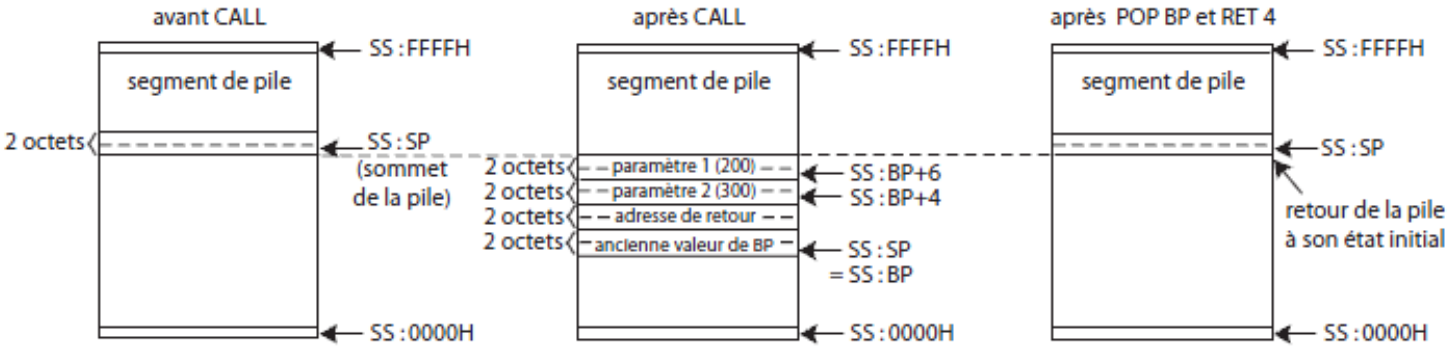
- procédure somme :

```
somme   proc
        push bp         ; sauvegarde de BP
        mov  bp,sp      ; faire pointer BP sur le sommet de la pile
        mov  ax,[bp+4]  ; récupération du deuxième paramètre
        add  ax,[bp+6]  ; addition au premier paramètre
        pop  bp         ; restauration de l'ancienne valeur de BP
        ret  4          ; retour et dépilage des paramètres
somme   endp
```

Remarque :

L'instruction **ret 4** permet de retourner au programme principal et d'incrémenter le pointeur de pile de 4 unités pour dépiler les paramètres afin de remettre la pile dans son état initial.

Etat de la pile :



Le registre Bp permet de lire des valeurs sur la pile sans la dépiler.

V. Les Macros :

Une suite d'instructions définie en Macro est reproduite à chaque occurrence du nom de la macro au moment de l'assemblage. Cependant, une séquence d'instructions, définie en tant que sous-programme elle sera exécutée par des opérations d'appel/retour (CALL et RET).

Une macro est définie avant la définition des segments (Pile, Données et Code). La syntax d'une définition de macro est la suivante:

NomMacro Macro « Liste de Paramêtres »

……

ENDM

Une macro peut désigner une simple séquence d'instructions. Dans ce cas, elle ne comprend pas de paramètres. Si on prend le cas des instructions permettant le retour au dos, le programmeur peut définir la macro DOSCALL suivante:

```
DOSCALL         Macro
Mov ah,4ch
Int 21h
ENDM
.............................
Code segment
Debut:
            .............................
DOSCALL     ; retour au Dos
Code ends
End debut
```

Une macro peut comporter des paramètres. A chaque référence de la Macro, les paramêtres effectifs remplaceront les occurrences des paramètres formels correspondants. On peut définir ainsi une Macro qui permet d'afficher une chaîne de caractères:

```
Affich   Macro   CH
  Mov ah,09h
  MOV DX, Offset CH
  Int 21h
ENDM
Data segment
            msg1 db '1: addition$'
            msg2 db 13,10,'2: soustraction$'
            msg3 db 13,10,'3: multiplication$'
Data ends
Code segment
            ASSUME CS :code, ds :data
      Debut:
            mov ax, data
            mov ds,ax
            affich msg1
            affich msg2
            affich msg3
            mov ah, 4ch
            int 21h
Code ends
      End debut
```