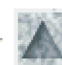
CHAPITRE N°10 : Sous-programme

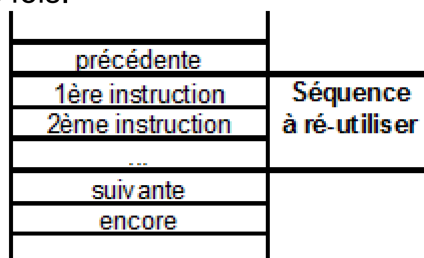
PLAN :

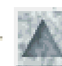
I) Définition p1
II) Appel d'un sous-programme
1) Défnition p2
2) Fonctions p2
3) Appels consécutifs p2
4) Appels imbriqués p3
5) En résumé p4
III) Programme d'appel
1) Instructions générales p4
2) JSR p4
3) Fonctionnement p5
IV) Retour de sous-programme
1) Définition et instructions générales p5
2) RTS p5
3) Fonctionnement p6
V) Mode d'appel
1) Mode direct (appel à adresse fixe) p6
2) Appel de méthode p7
VI) Paramètres p8
VII) Fonctions p8
VIII) Stack frame
1) Généralités p8
2) Exemple de fram pour un appel p8
3) Exemple de fram d'exécution p9
4) Exemple de fram juste avant le retour p10



I) Définition

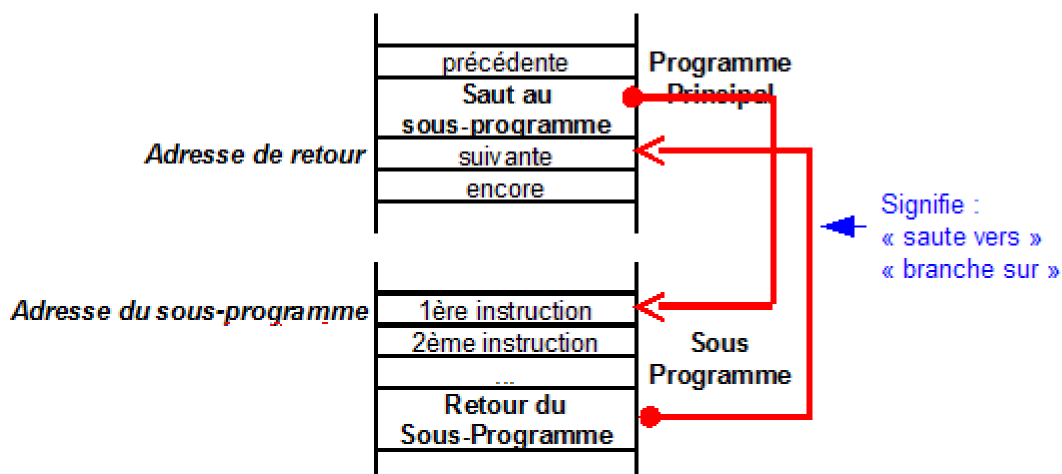
Un sous-programme ("subroutine" ou "subprogram") est une séquence d'instructions que l'on veut pouvoir utiliser plusieurs fois.
Structure d'un sous-programme :



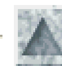


II) Appel d'un sous-programme

1) Définition



Le saut au sous-programme ("Jump to SubRoutine") est souvent appelé : Appel au sous-programme ("Call").



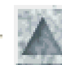
---

2) Fonctions

Les fonctions-membres ou méthodes des objets des langages orientés objet (e.g. JAVA, C++, C#, Eiffel ...) sont implémentées par des sous-programmes.
Utilisation :

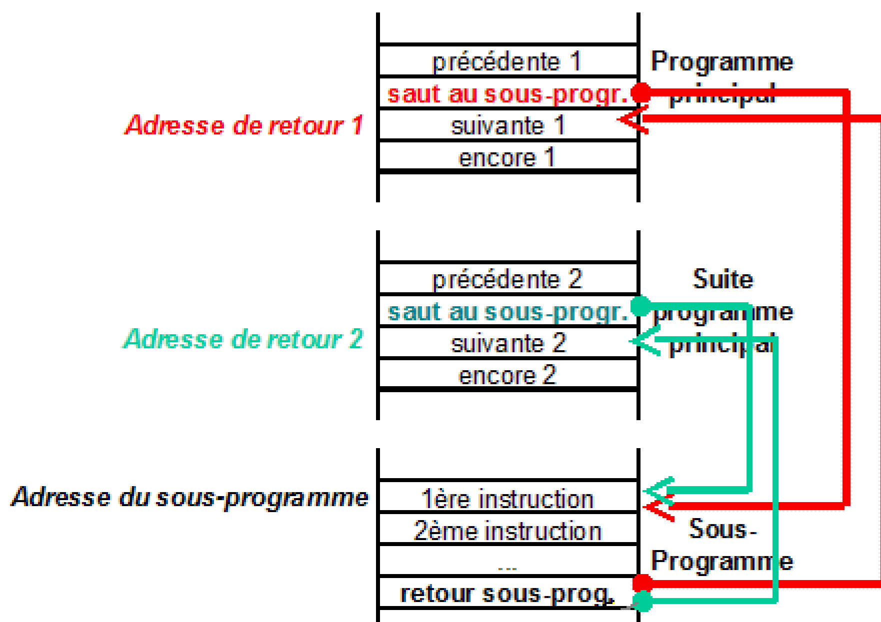
```
toto.boost() ;
```

appelle le sous-programme "boost" de l'objet "toto".



---

3) Appels consécutifs

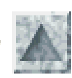


Pour retourner à l'instruction suivant l'appel dans le programme principal, il faut avoir d'abord sauvegardé son adresse. Elle s'appelle "adresse de retour" :

Mémorisation retour :

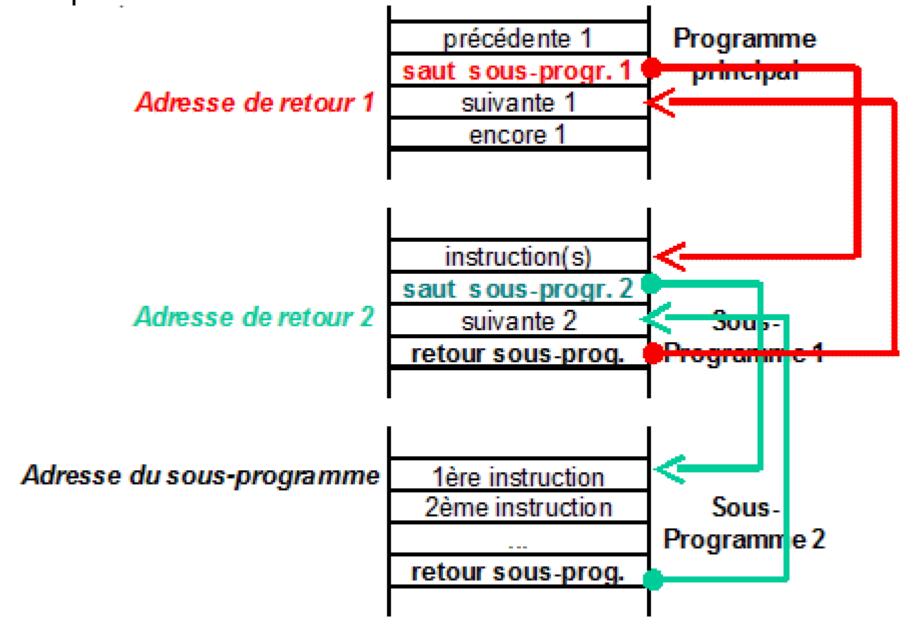
Adresse de retour 2

(c'est la sauvegarde de l'adresse de retour dans un registre ou une case mémoire fixe)

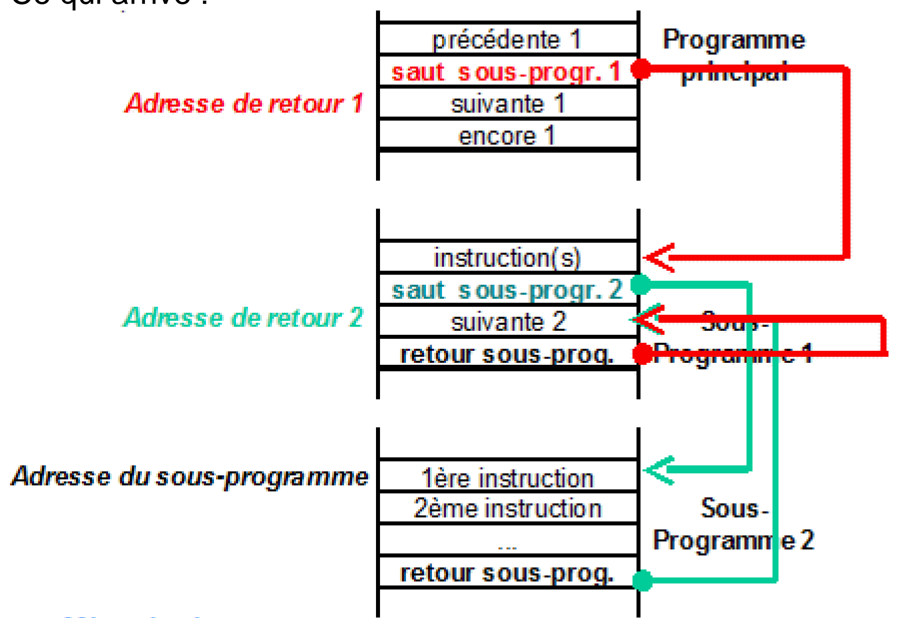


4) Appels imbriqués

Ce qu'on devrait avoir normalement :



Ce qui arrive :



Mémorisation retour :

Adresse de retour 2

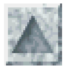
L'adresse de retour 1 est écrasée.

Solutions :
Il faut donc sauvegarder l'adresse de retour 1 dans une pile.

Ceci permet des sous-programmes ré-entrants (qui s'appellent eux-mêmes) et donc récursifs !
Mais la pile est limitée ...

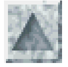
Autre solution :
liste chaînée (rare car pas adaptée ici : pas d'avantage, compliquée, volumineuse et moins

rapide).



5) En résumé

Un appel à sous-programme :
- empile l'adresse de retour
- saute à la 1ère instruction du sous-programme.



III) Programme d'appel

1) Instructions générales

Une instruction spéciale n'est pas nécessaire !

```
MPC R2 ;         // charge le contenu du PC dans R2
STW R2, -(SP) ;  // empile l'adresse de retour
JEA (R1) ;       // saute au sous-prog. pointé par R1
```

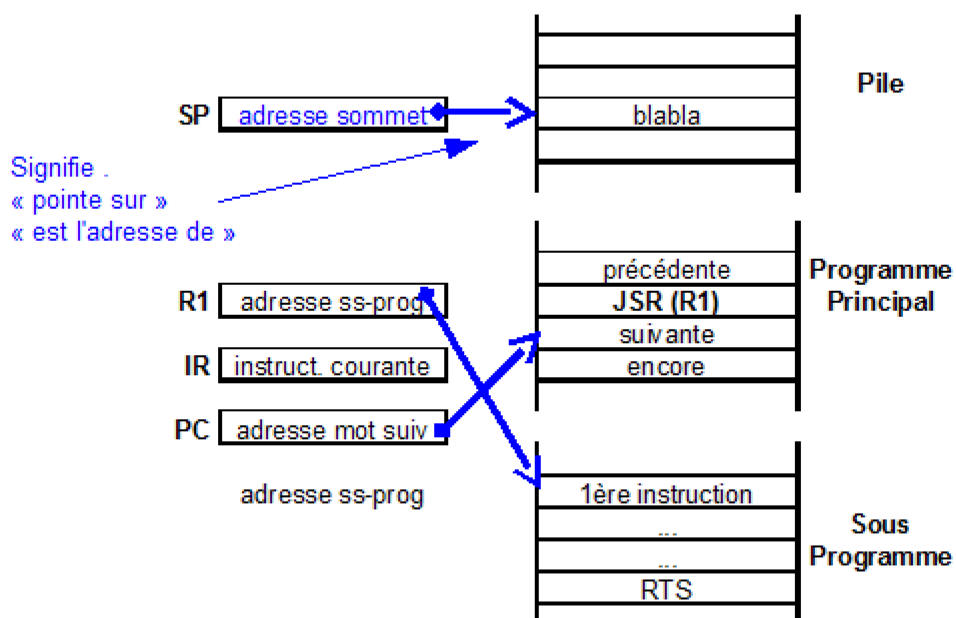
2) JSR

Le saut au sous-programme est effectué par l'instruction : "Jump to SubRoutine" (JSR).
L'opérande de l'instruction JSR est un sous-programme.
L'adresse de cet opérande est l'adresse du sous-programme, c'est à dire l'adresse de sa 1ère instruction.

MODE D'ADRESSAGE	NOTATION ASSEMBLEUR	ADRESSE DU SOUS-PROGRAMME	COMMENTAIRE
Basé	JSR *(R)*	R	*simple et générique*
Direct	JSR *@address*	address = IE = M[PC]	*adresse fixe*
Indexé	JSR *(R)disp*	R + disp = R + IE	*compliqué*
Indirect pré-indexé	JSR **(R)disp*	M[R + disp] = M[R + IE]	*très compliqué*
Basé pré-décrémenté	JSR *-(R)*	R-2	*peu cohérent ici*
Basé post-incrémenté	JSR *(R)*+	R	*peu cohérent ici*
Immédiat	-	PC	*incohérent ici*
Registre	-	-	*aberrant*

En conclusion :
JSR :
- empile l'adresse de retour
- saute à l'instruction de l'opérande

3) Fonctionnement



1) instructions initialisation (JSR) :
SP <- SP - 2 // pour obtenir l'adresse de retour
M[SP] <- adresse de retour // l'adresse de retour est contenu dans le PC, et on doit la stocker dans la pile.
Cette instruction devient : M[SP]<-PC
2) Ensuite, exécution du sous-programme :
l'instruction contenu dans IR est exécutée :
IR <- M[adresse ss-prog], qui devient IR <- M[R1]
PC <- adresse ss-prog # 2 // le PC pointe sur la deuxième instruction du sous-programme, qui devient : PC <- R1 # 2
Puis, on répète ce schéma jusqu'à la fin des instructions du sous-programme.

IV) Retour de sous-programme

1) Définition et instructions générales

Le retour de sous-programme :
- dépile l'adresse de retour ;
- saute à l'instruction à cette adresse.

Une instruction spéciale n'est pas nécessaire !

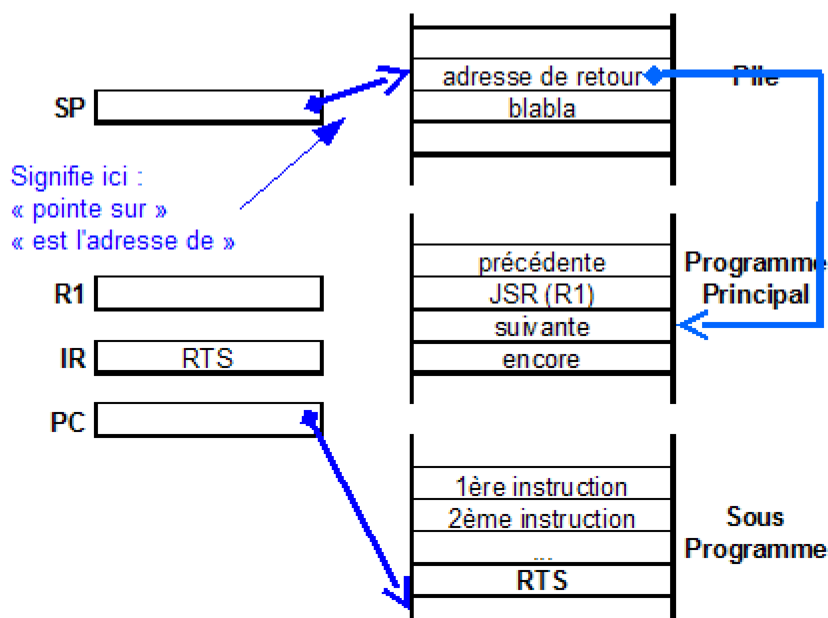
```
LDW R1,(SP)+ ; // dépile l'adresse de retour dans R1
JEA (R1) ;     // saute à l'instruction de retour
```

2) RTS

Le retour du sous-programme est effectué par l'instruction : ”ReTurn from Subroutine" (RTS)

RTS :
- dépile l'adresse de retour
- saute à l'instruction suivant JSR

3) Fonctionnement



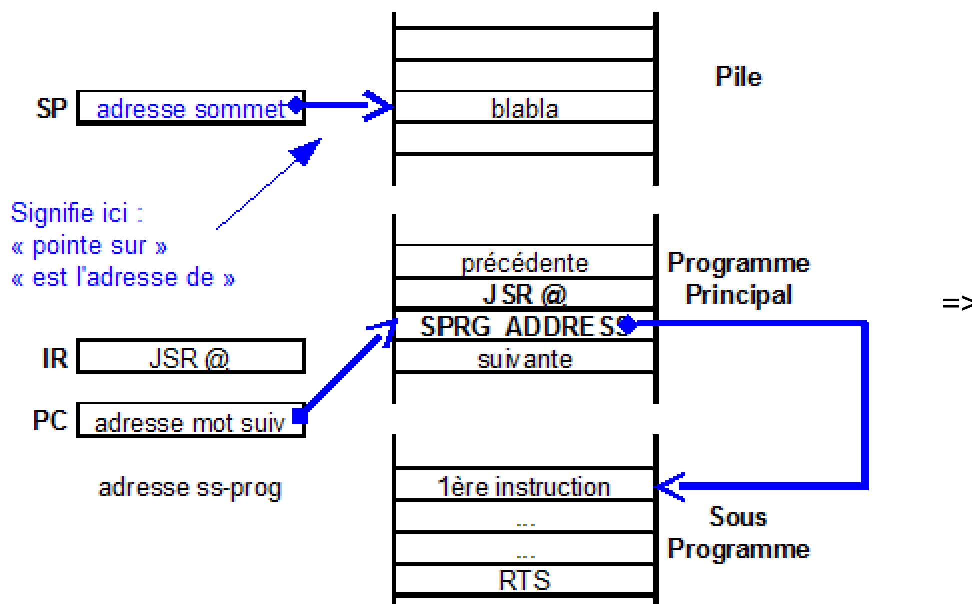
1) instructions initialisation :
PC <- adresse de retour //on veut retourner après JSR(R1), pour poursuivre. L'adresse est contenue dans le SP.
d'où l'instruction devient : PC <- M[SP]
SP <- SP # 2 // on incrémente SP pour obtenir la suite de la pile
2) Ensuite, préparation de l'instruction suivante :
IR <- M[PC] // on va exécuter l'instruction suivante
PC <- PC # 2 // on incrémente le PC pour passer à l'instruction suivante.

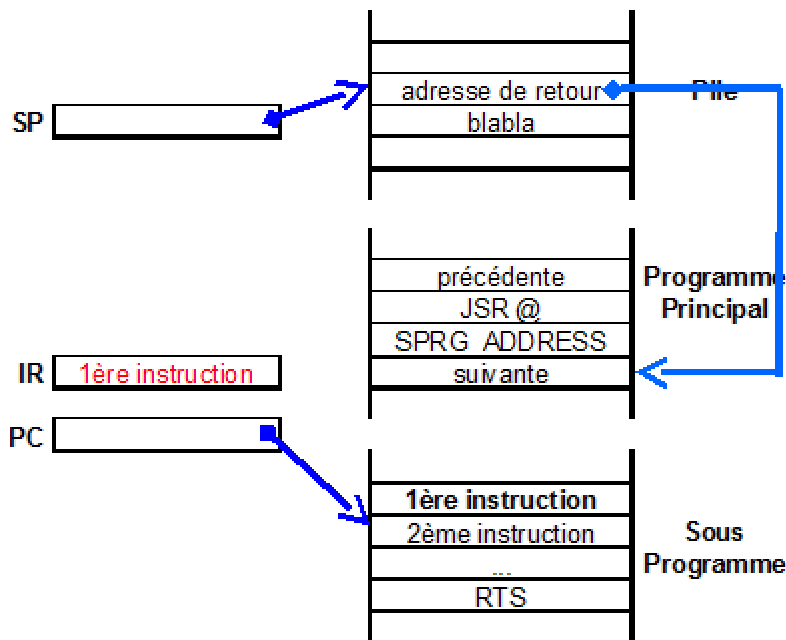
V) Mode d'appel

1) Mode direct (appel à adresse fixe)

Dans les langages non-objet, le sous-programme est souvent à une adresse fixe connue à l'avance.

```
JSR @SPRG_ADDRESS ;// appelle sprog d'adresse SPRG_ADDRESS
```





les 2 principales étapes de l'exécution de l'appel et du retour de sous-programme.

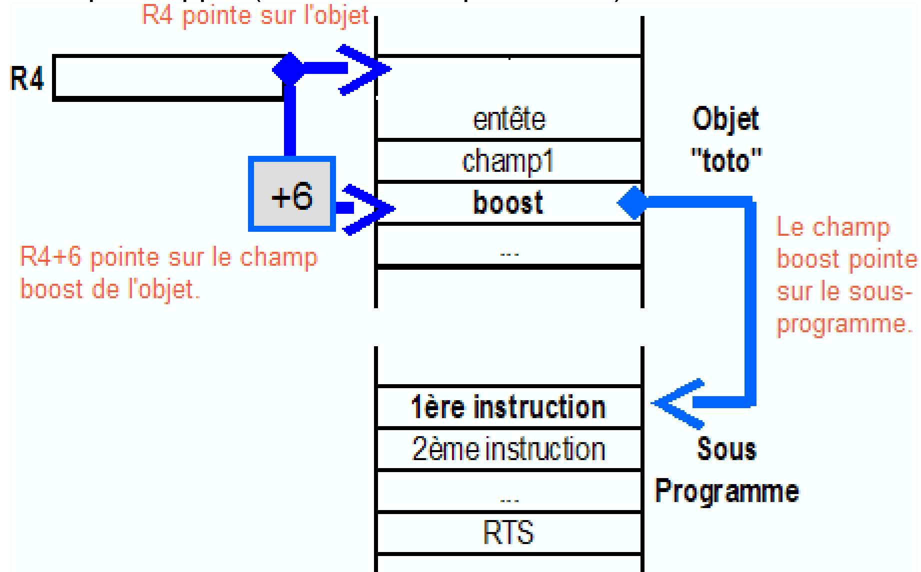
Le mode direct n'est pas nécessaire :

```
LDW R2, #SPRG_ADDRESS  ; // charge R2 avec SPRG_ADDRESS
JSR (R2) ;   // saute au sous-prog pointé par R2
```

2) Appel de méthode

Dans les langages à objet, l'adresse du sous-programme n'est connue qu'à l'exécution.
Le codage des objets varie d'un langage et d'une implémentation à l'autre.

exemple d'appel (mode indirect pré-indexé)

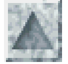



```
JSR *(R4)6 ; // appelle boost de l'objet pointé par R4
```

Sans mode indirect pré-indexé :
(le mode indirect pré-indexé n'est pas nécessaire)

```
LDW R2, R4 ;    // R2 pointe sur l'objet toto
ADQ 6, R2 ;     // R2 pointe sur le champ boost de toto
LDW R2, (R2) ; // R2 pointe sur le sous-programme
```

```
JSR (R2) ;      // saute au sous-programme boost de toto
```



VI) Paramètres

Le fonctionnement d'un sous-programme peut dépendre de paramètres ("arguments" ou "parameters") :

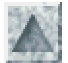
```
toto.boost(5, "piano") ;
```

(5 entier, et piano chaîne de caractères)
Le nombre de paramètres s'appelle arité.
Ils peuvent être représentés par :
- leur valeur
- leur adresse.

Ces paramètres sont passés du programme principal au sous-programme par stockage préalable à l'appel :
- dans des registres (simple et rapide)
- sur la pile (nombre quasi-illimité)
- dans un descripteur en mémoire (rare : économise la pile mais compliqué).

VII) Fonctions

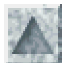
Un sous-programme qui retourne une information par la gauche s'appelle une fonction ("function").
Cette information est stockée préalablement au retour :
- dans un registre précis
- sur la pile.



VIII) Stack frame

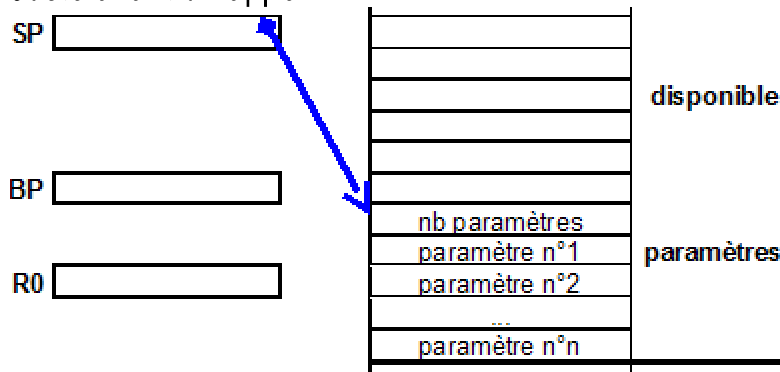
1) Généralités

Divers procédés sont utilisés pour échanger des informations entre programme et sous-programme.
Ils dépendent du langage évolué utilisé et de son implémentation sur chaque machine.
Ils sont basés sur la configuration des informations sur la pile à l'appel, pendant l'exécution et au retour.
La configuration de la pile s'appelle "stack frame" ou modèle de pile ou environnement.



2) Exemple de fram pour un appel

Juste avant un appel :



on empile des paramètres, on les comptes grâce à R0, et on empile le nbre de para sur la

pile.

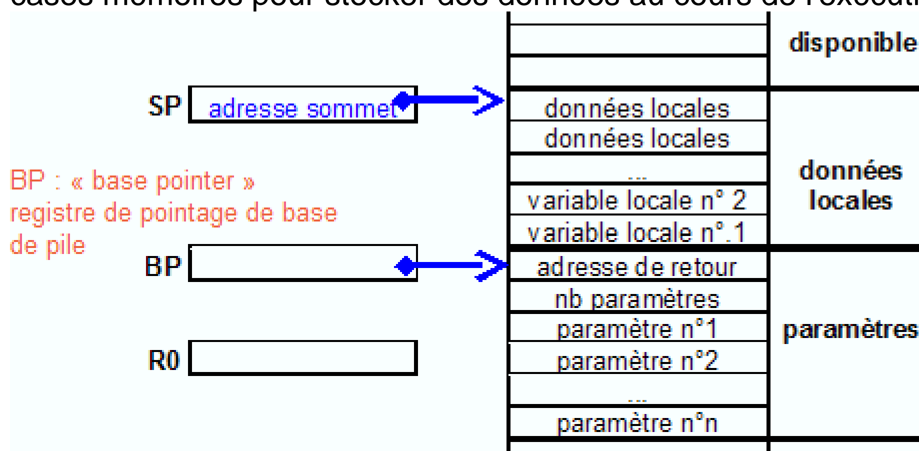
```
LDQ 0, R0     ; // initialise R0
...
dSTW Ri,-(SP) ; // empile un paramètre depuis Ri
ADQ 1, R0 ;     // compte le paramètre
...
STW R0,-(SP) ; // empile le nb de param.
```

Juste après l'appel, on a la même chose, avec au-dessus du nbre de paramètres : l'adresse de retour.

```
JSR @adresse_du_sous_prog  // appelle le sous-prog.
```

3) Exemple de fram d'exécution

On ajoute un Base Pointer pour pointer dans les variables et paramètres, et on réserve des cases mémoires pour stocker des données au cours de l'exécution.




```
LDW BP,SP ;        // BP pointe sur adresse retour
ADQ #10, SP ;      // réserve 5 mots sur la pile
```

a) Accès à un paramètre

- mode indexé (déplacement > 0) :
BP pointe sur paramètre n°1, et on charge la valeur dans R0

```
LDW R0,(BP)4 ;  // charge le paramètre n°1 dans R0
```

- sans mode indexé :

```
LDW R0,BP ;    // charge BP dans R0
ADQ 4, R0;     // ajoute 4 à R0, R0 pointe sur param. 1
LDW R0, (R0); // charge paramètre n°1 dans R0
```

b) Accès à une variable

mode indexé (déplacement < 0) :

```
LDW R0,(BP)-4 ;  // charge la variable n°2 dans R0
```

c) Accès indirect

- mode indirect pré-indexé :
On va charger directement le paramètre 2 dans R0.

```
LDW R0,*(BP)6 ;  // charge M[M[BP+6]] dans R0
```

- sans mode indirect pré-indexé :

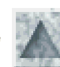
```
LDW R0,BP ;    // charge BP dans R0
ADQ 6, R0 ;    // ajoute 6 à R0; R0 pointe sur param.2
LDB R0, (R0); // charge octet n°0 dans R0
```

- mode indirect pré et post-indexé : (pré (+6) et post (+5) indexé)
le paramètre 2 pointe lui-même sur une chaîne de 6 caractères par exemple.
On charge le paramètre 2, puis le caractère 5 de ce paramètre.

```
LDW R0,(BP)6 ; // charge paramètre n°2 dans R0
LDB R0,(R0)5 ; // charge octet n°5 de la chaîne ds R0
```

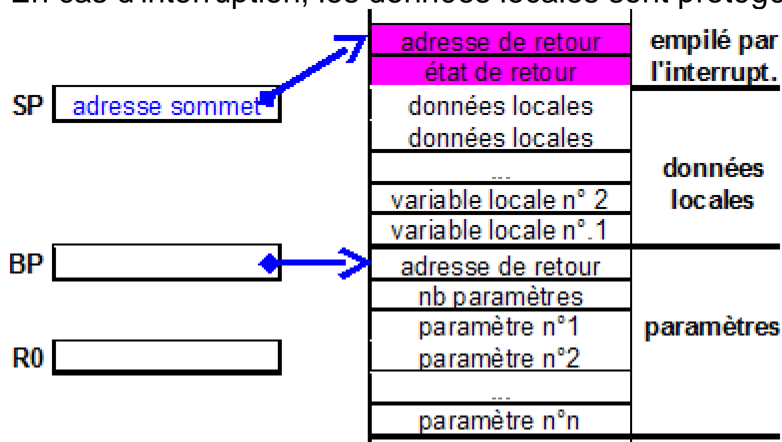
- sans mode indirect pré et post-indexé :

```
LDW R0,BP ;    // charge BP dans R0
ADQ 6, R0 ;    // ajoute 6 à R0; R0 pointe sur param.2
LDW R0, (R0); // charge paramètre n°2 dans R0
ADQ 5, R0 ;    // ajoute 5 à R0; R0 pointe sur 'n'
LDB R0, (R0); // charge octet n°5 de la chaîne ds R0
```



d) Pile juste après interruption

En cas d'interruption, les données locales sont protégées.



4) Exemple de fram juste avant le retour

On mémorise la valeur à retournée, et on dirige SP sur l'adresse de retour. Puis on sort du sous-programme.

```
LDW R0, ... ; // charge la valeur à retourner dans R0
LDW SP, BP  ; // SP pointe sur l'adresse de retour
RTS   // retourne du sous-programme
```