Plusieurs fois même opération ? ⇝ routines

Routine : bloc de code accessible

- Appel par saut (`JSR`) donc modif de PC — Étiquette
- Retour (`RET`) à instruction suivant l'appel — Mémorisation (R_7)

`JSR` : depuis le principal mémo retour dans R_7 + modif PC

```
     ...
     JSR sub     ; R7 <- PC ; PC <- PC + SEXT(PCoffset11)
     ...
```

`RET` : depuis le bloc retour au principal

```
sub: ...
     RET         ; PC <- R7
```



```
        .ORIG x3000
        LEA R0,string   ; Initialisation du pointeur R0
        AND R1,R1,0     ; Le compteur R1 est initialisé à 0
loop:   LDR R2,R0,0     ; Chargement dans R2 du caractere pointé par R0
        BRz end         ; Test de sortie de boucle
        ADD R0,R0,1     ; Incrémentation du pointeur
        ADD R1,R1,1     ; Incrémentation du compteur
        BR loop
end:    ST R1,res
        HALT
; Chaîne constante
string: .STRINGZ "Hello"
res:    .BLKW #1
        .END
```

On veut :

- Calcul de la longueur (chaîne pointée par R_0)
- Résultat rangé dans R_0


```
        .ORIG x3000
        LEA R0,string   ; Initialisation du pointeur R0
        JSR length      ; saut vers la routine
        ST  R0,res
        HALT
; données
string: .STRINGZ "Hello"
res:    .BLKW #1
; Routine pour calculer la longueur d'une chaîne (terminée par '\0')
; Entrée : R0 adresse de la chaîne
; Sortie : R0 longueur de la chaîne
length: AND R1, R1, #0
loop:   LDR R2, R0, #0
        BRz end
        ADD R0, R0, #1
        ADD R1, R1, #1
        BR  loop
end:    ADD R0, R1, #0
        RET
```


Appel à routine dans routine ?

```
; programme principal
      ...
      JSR sub1
      ...

; routine sub1
sub1: ...
      JSR sub2
      ...
      RET ; Problème :  ici R7 ne contient pas l'adresse de retour
          ; vers sub1 !  (puisque sub2 a été appelée juste avant...)

; routine sub2
sub2: ...
      RET
```

Appel à routine dans routine ? R_7 change! ⇝ pile

Pile : structure de mémorisation adresses et registres

- Sommet (dernier ajouté) pointé par R_6 (fond : `stackend`)
- Push ⇝ nouveau sommet ⇝ modif de R_6 + mémo

  Ici croissance vers adresses inférieures
- Pop ⇝ lecture + nouveau sommet ⇝ modif de R_6

Installation :

- De la place (pas trop, pas trop peu...) `.BLKW`
- Un fond `.BLKW`
- Initialisation (adresses de tout ce petit monde)

```
          .ORIG x3000
; Principal
main:     LD R6,spinit  ; initialisation de R6
          ...
          HALT          ; limite de l'exécution

; Gestion pile
spinit:   .FILL stackend ; déclaration et initialisation
          .BLKW #15
stackend: .BLKW #1      ; stackend : adresse du fond de la pile

          .END          ; limite de l'assemblage
```


Utilisation :

- Push : **Avant appel à sous-routine**
  1. Bouger le pointeur de sommet
  2. Stocker le registre à sauver

```
ADD R6, R6, #-1
STR Rx, R6, #0
```

- Pop : **Après retour de sous-routine**
  1. Récupérer la donnée au sommet
  2. Bouger le pointeur de sommet

```
LDR Rx, R6, #0
ADD R6, R6, #1
```