```
-----------
| Données |
-----------

.byte : 1 octet (équiv. en C : char)
.half (half-word) : 2 octets (équiv. en C : short)
.word : 4 octets (équiv. en C : int)

.float : 4 octets, nombre flottant simple précision (équiv. en C : float)
.double : 8 octets, nombre flottant double précision (équiv. en C : double)

.asciiz : chaîne de caractères, terminée par le caractère nul (équiv. en C : char*)
.ascii : chaîne de caractères

On utilise la notation hexadécimale pour représenter la valeur d'un byte, half-word
ou word. Chaque chiffre hexadécimal représente 4 bits. Par exemple :

0x8e : représente la valeur [1000 1110] (byte)
                            \--/ \--/
                             8    e

0x5ca0 : représente la valeur [0101 1100 1010 0000] (half-word)
                              \--/ \--/ \--/ \--/
                               5    c    a    0

0x8f01a0cd : représente la valeur [1000 1111 0000 0001 1010 0000 1100 1101] (word)
                                   \--/ \--/ \--/ \--/ \--/ \--/ \--/ \--/
                                    8    f    0    1    a    0    c    d

Note : MIPS utilise le complément à deux pour représenter les entiers négatifs.

-------------------------------------
| Convention d'appel d'une fonction |
-------------------------------------

  Pour appeler une fonction, on utilise les instructions jal (appeler une fonction)
et jr (retourner) :

    jal fonction    écrit l'adresse de l'instruction suivante (PC+4) dans le
                    registre $ra et saute à l'instruction qui se trouve à
                    l'étiquette fonction

    jr $ra          branchement : saute à l'instruction dont l'adresse se trouve
                    dans le registre $ra

  Si la fonction prend des arguments : Les 4 premiers arguments sont passés dans
les registres $a0-$a3. Si la fonction prend plus de 4 arguments, les arguments
restants sont passés dans la pile (voir ci-dessous).

  Si la fonction retourne un résultat, ceci est passé dans le registre $v0 (ou bien
dans la paire de registres $v0,$v1 si besoin de retourner plus de 32 bits).

  Une fonction a le droit d'utiliser les registres $t0-$t9 et les laisser dans un
état arbitraire.
==> Corollaire : Si j'appelle une fonction et que j'ai besoin de retrouver les
valeurs des registres $t0-$t9, je suis obligé de les sauvegarder avant d'appeler la
fonction, et les restaurer une fois que la fonction a retourné.
De même pour les registres $a0-$a3 et $v0-$v1.

  Par contre, une fonction qui utilise les registres $s0-$s7 est obligée de les
laisser dans leur état d'origine.
==> Corollaire 1 : Une fonction qui utilise $s0-$s7 est obligée de les sauvegarder
au tout début et de les restaurer juste avant de retourner.
==> Corollaire 2 : Si j'appelle une fonction, je peux m'attendre à ce que les
registres $s0-$s7 restent inchangés.

  Si une fonction f1 appelle une autre fonction f2, alors f1 est obligée de
sauvegarder sa propre adresse de retour (registre $ra) : le contenu de $ra sera
```

remplacé au moment d'exécution de l'instruction jal f2.

La pile

La pile réside dans ce segment et le registre $sp contient l'adresse du dernier mot (word) de la pile. Pour empiler un nouveau mot, on décrémente $sp (la pile grossit vers les adresses décroissantes) et ensuite on écrit le mot dans ($sp) :

```
-----------------------------------------------------
|   sub $sp, $sp, 4     # allouer 4 octets = 1 mot |
|   sw $t0, ($sp)       # empiler $t0              |
-----------------------------------------------------
```

Pour dépiler, on applique la procédure inverse :

```
-----------------------------------------------------------
|   lw $t1, ($sp)       # lire le dernier mot dans $t1 |
|   add $sp, $sp, 4     # désallouer 4 octets          |
-----------------------------------------------------------
```

Trame de pile d'un appel de fonction

Afin de mettre en œuvre tout cela de manière cohérente et dans l'intérêt de minimiser le risque d'erreur, on utilise la convention suivante pour la trame de pile d'un appel de fonction :

1) Si la fonction prend plus de 4 arguments : l'appelant écrit les arguments, au-delà des 4 premiers, en dessous de $sp (dans l'ordre), sans modifier $sp.

Ensuite l'appelant exécute l'instruction jal ; on rentre donc dans la fonction appelée.

2) Au tout début, l'appelé décrémente $sp pour allouer sufisamment d'octets pour accueillir les arguments de pile (ceux après le 4ème argument), la valeur de $ra, et les valeurs locales qui seront empilées par la suite.

3) Avant de retourner, l'appelé écrit dans $ra sa valeur originale, remet $sp à sa valeur avant l'appel de la fonction, et finit par exécuter l'instruction jr $ra. (sans oublier d'écrire dans $v0 la valeur de retour)

Exemple

Soit une fonction f qui prend 5 arguments entiers (5 words) et retourne un entier, et qui a besoin d'utiliser localement les registres $s0, $s1 et $s2. L'appelant appelle f avec 5 arguments initialement stockés dans les registres $t0-$t4 : (pseudocode)

```
-----------------------------------          Etat de la pile avant l'appel
|   $a0 := $t0                    |
|   $a1 := $t1                    |                  |         |
|   $a2 := $t2                    |                  |---------| $sp
|   $a3 := $t3                    |                  |   $t4   |
|   écrire $t4 dans -4($sp)       |                  |---------| -4
|   jal f                         |                  |         |
|   récupérer le résultat dans $v0 |                 |---------| -8
-----------------------------------
```

Dans sa trame de pile, l'appelé a besoin de stocker 5 éléments (le 5e argument, $ra, et les trois registres $s0-$s2). Donc il commence par décrémenter $sp par 5*4 = 20. Voici le pseudocode complet :

```
-------------------------------------------------------------  Etat de la pile
| f:                                                        |  avant le calcul
|   $sp := $sp - 20                                         |    |         |
|   écrire $ra dans 12($sp)                                 |    |---------| 20
|   # Sauvegarde des valeurs $s0, $s1, $s2 de l'appelante   |    |   $t4   |
|   écrire $s0 dans 8($sp)                                  |    |---------| 16
|   écrire $s1 dans 4($sp)                                  |    |   $ra   |
|   écrire $s2 dans 0($sp)                                  |    |---------| 12
|   # le 5e argument se trouve dans 16($sp)                 |    |   $s0   |
|                                                           |    |---------| 8
|   ....... effectuer le calcul en utilisant $s0, $s1 et $s2|    |   $s1   |
|            et écrire le résultat dans $v0                 |    |---------| 4
|                                                           |    |   $s2   |
| f_return:                                                 |    |---------| $sp
|   lire 12($sp) dans $ra                                   |    |         |
|   # Restauration de $s0, $s1, $s2                         |    |---------| -4
|   lire 8($sp) dans $s0                                    |
|   lire 4($sp) dans $s1                                    |
|   lire 0($sp) dans $s2                                    |
|   $sp := $sp + 20                                         |
|   jr $ra                                                  |
-------------------------------------------------------------
```

Observez que l'appelant construit la première partie de la trame de pile en y mettant le 5e argument, sans pour autant modifier $sp. C'est la fonction appelée qui se charge de décrémenter $sp et de construire le reste de la trame de pile.

Si la fonction f a besoin de plus de 8 variables locales : Dans ce cas, les registres $s0-$s7 ne sont pas suffisants et la fonction utilisera la pile pour stocker les variables au-delà de 8.

Remarque : Le registre $fp (frame pointer) est parfois utilisé pour stocker l'adresse du premier mot de la trame de pile actuelle, dans le cas où la taille de la trame de pile est calculée dynamiquement. On n'utilisera pas $fp.