```
Stockage des variables

  Pour commencer, on ne s'intéresse qu'aux variables globales (externes aux
procédures).

I) Notations * et & du langage C

L'opérateur C unaire &yy signifie "adr_de" (premier octet de) yy.
L'opérateur unaire *aa représente Mem[aa] : contenant mémoire d'adresse aa.

Comment désigner le type C "pointeur de T" ?
(référence à/adresse d'une entité de type T")

Notation utilisée en C++ pour les références aux objets :
   & T   ptr;     /* ptr est du type obtenu par application de      */
                  /* l'opérateur référence (&) à une variable de type T */

Notation utilisée en C pour les pointeurs :
   (T *) ptr;     /* ou encore */
   T    *ptr;     /* ptr est d'un type tel que l'application de l'opérateur */
                  /* d'indirection * dessus retourne quelquechose de type T */

Aucune variable ou fonction ne peut avoir pour adresse NULL.
La constante NULL ne repère rien. Elle correspond en général à l'adresse 0.

II) Déclaration et stockage (réservation de place)

1.1) Dans registres du processeur

Rapide (identifiant court dans instruction, pas d'accès Mem),
mais nombre limité, un registre a un numéro mais pas d'adresse.

    ______ n'utiliser que dans le cadre LM pour spéficier type de stockage
   |       ne pas utiliser en programmation ordinaire
   v
register unsigned long a,b,c;   /* choix arbitraire : a:r7 b:r8 c:r9 */


2.1) en mémoire (sections data ou bss)

  plus lent (accès mémoire, identifiant sur 32/64 bits),
  nombre "illimité", autorise pointeurs sur variables

int x = 3;
int y = 6;
int z;

Traduction des déclarations en asm :

Chaque etiquette etc_var représente une adresse notée &var en C.
(etc_ pour & du C).

data : valeurs initiales quelconques
   (fichier exécutable contient valeur initiale de chaque octet)

bss  : valeurs initiales des variables obligatoirement 0
   (fichier exécutable ne précise que la taille de bss)

bss et data équivalents à optimisation de taille du fichier exécutable près

            .bss        @ implicite : valeurs initiales toujours à 0
```



```
etc_z:      .skip   4   @ réserve 4 octets, valinit implicite = 0

            .data       @ valeur initiales quelconques
                        @ contenues dans le fichier exécutable
etc_x:      .word   3   @ réserve 1 mot de 4 octets avec valinit 3
etc_y:      .word   6   @ idem                            valinit 6
 ^                  ^
 |                  |
Adresses/def Contenu initial
Etiquettes

Si lors d'une exécution data et bss débutent en 0x2000 et 0x3000, alors

&x en C et etc_x en asm synonymes de la constante adresse 0x00002000
&y en C et etc_y en asm synonymes de la constante adresse 0x00002004
&z en C et etc_z en asm synonymes de la constante adresse 0x00003000

Au début de l'exécution du programme
Mem[0x00002000] contient 3 sur 4 octets
Mem[0x00002004] contient 6 sur 4 octets
Mem[0x00003000] contient 0 sur 4 octets

En langage C :
   y signifie en réalité *&y  (emplacement mémoire dont l'adresse est
               &y, autrement dit Mem[0x2004])
   z signifie en réalité *&z  (emplacement mémoire dont l'adresse est
               &z, autrement dit Mem[0x3000])


III) Signification d'une affectation

    var = expression  (forme lvalue = rvalue)

   Les opérandes des opérateurs de l'expression à droite sont :
   * des constantes (qui seront incluses dans les instructions)
   * des contenants dont le contenu est consulté
      + registre (utilisables directement dans les instructions de calcul)
      + mémoire  (lire : copier le contenu dans un registre avant calcul)

   A gauche, il y a un contenant auquel on affecte un nouveau contenu
      + registre (champ dest des instructions de calcul)
      + mémoire (déposer résultat dans registre, recopier en mémoire : écrire)

3.1) Cas simple : variables dans des registres

@ a : r7  b : r8
C                   Traduction asm
                    Instr Dest opG opD
a = 4;              mov   r7,     #4
a = a - b;          sub   r7, r7, r8
a = b - 5;          sub   r7, r8, #5

      r7 ppnc                                 constante 4
      r7 ppnc  contenu_actuel(r7)   moins   contenu_actuel(r8)
      r7 ppnc  contenu_actuel(r8)   moins   constante 5
             ----->  prend pour nouveau contenu

# : constante immédiate (immédiatement accessible dans le code de l'instruction)

3.2) Cas général : variables en mémoire

   z = y - c;  s'écrit aussi  *&z = *&y - c;
```

```
   *   &z  =              *   &y  -                      c
   Mem[&z] ppnc contenu de Mem[&y] - contenu de registre r9

   Lire contenu de Mm[0x2004] --> copie  dans un registre tampon
   Ajouter contenu de tampon et registre r9 dans un registre
   Ecrire résultat présent dans registre --> Mem[0x3000]

    Première * à gauche d'une affectation : écriture
    Autres   *                            : lecture

IV) Variables en mémoire et instructions RISC

  Contraintes RISC :
  * seules load et store accèdent à la mémoire
  * les adresses utilisées par load et store de forme reg+reg ou reg+#cte_8

  ldr r1,[r2] + synonyme de ldr r1,[r2,#0]
              + signifie r1 ppnc contenu de Mem[r2] (lecture en mémoire)
              + correspond à r1 = *r2  (C)

  str r1,[r2] + synonyme de str r1,[r2,#0]
              + signifie Mem[r2] ppnc contenu de r1 (écriture en mémoire)
              + correspond à *r2 = r1  (C)

  ==> on décompose en un programme C équivalent pour traduire
  ==> les adresses et les contenus des variables transitent par des
  registres servant de variables temporaires.

register int  r1,  r2;      /* registres de type int     pour les contenus */
register int *r4, *r5;      /* registres de type (int *) pour les adresses */

Instructions

Rappel :       ldr  reg, =constante equivaut à peu pres à
               mov  reg, #constante, mais accepte une constante 32bits

r4 = &y;        ldr  r4, =etc_y    @ r4 = &y            --> r4 = 0x2004
r1 = *r4;       ldr  r1,[r4]       @ r1 = *&y  (r1 = y) --> r1 = 6
r2 = r1 - c ;   sub  r2, r1, r9    @ r2 = y - c         --> r2 = 6-c
r5 = &z;        ldr  r5, =etc_z    @ r5 = &z            --> r5 = 0x3000
*r5 = r2;       str  r2, [r5]      @ *&z = r2  (z = y-c) --> Mem[0x3000] = r2

               .ltorg               @ zone de stockage
                                    @ des constantes 0x2004, 0x3000

5  cycles de lecture : mots représentant les instructions
3  cycles de lecture demandés par les ldr :
   ctes 00002004 et 00003000 dans ltorg,
   contenu de y dans data
1 cycle d'écriture demandé par le str : contenu de z dans bss

Notes :

1)    On a besoin d'utiliser des registres comme temporaires

2)    On aurait pu n'utiliser que 2 registres temporaires :
      ldr  r1, =etc_y      @ ldr r1, = etc_y
      ldr  r1, [r1]        @ ldr r1, [r1]
      sub  r1, r1, r9
      ldr  r2, =etc_z
      str  r1, [r2]
```



```
3)    Toutes les etiquettes seront de la forme etc_qqchose : on supprime le
      etc_ a l'avenir. Mais une etiquette x dans data correspond a &x en C.
```