```
I) Sous-multiples du mot et alignement

Réservation de place :

        .short  valeur          ! réserve et initialise 2 octets  --> short
        .byte   valeur          ! idem 1 octet . Note .byte 'a'   --> char

        .skip   nb_octets       ! réservation sans valeur initiale (0)
                                ! seul autorise dans bss

Contraintes d'alignement

        tout objet placé à une adresse multiple de sa taille
        &x modulo sizeof(x) = 0

        On laisse des trous (.skip) si besoin

Exemple :                4 * X     4*X + 1   4*X + 2   4*X + 3
                        -----------------------------------------
x:      .byte 0         |         |/////////|/////////|/////////|
        .skip 3         -----------------------------------------
y:      .word 6         |         .         .         .         |
                        -----------------------------------------
                         4 * (X+1) 4*(X+1)+1

        Pour éviter de compter à la main :
        .balign t       ! génère le bon nombre de .skip
                        ! t est la taille de l'objet stocké après

        Pour éviter de perdre trop de place : ordonner par taille

II) Structures

2.1) Déclaration

Une structure se déclare comme une liste de variables correspondant à ses
champs, avec les précautions d'alignement.

struct _s {                     @ SIZEOF_STRUCT_S = 16
   int x;                       @ DELTA_STRUCT_S_X =0
   char cx;                     @ DELTA_STRUCT_S_CX =4
   int y;                       @ DELTA_STRUCT_S_Y =8
   char cy;};                   @ DELTA_STRUCT_S_CY =12

struct _s s1 = {2, 'a', 3, 'b'};
struct _s s2, s3;

                .data                                .bss
s1:             .word   2               s2:     .skip   16
                .byte   'a'             s3:     .balign 4   @ redondant
                .balign 4                       .skip   16
                .word   3
                .byte   'b'

5.2) Accès

Comme pour une variable ordinaire, mais l'adresse des champs est déduite
de leur position par rapport au début de la structure.

    s2.y = 3                    @                   * &s2.y = 3
                                ldr     r10,= s2
                                mov     r9,#3
```



```
                                str     r9, [r10, #DELTA_STRUCT_S_Y]

III) Pointeurs (C) : déclaration

   Comme une variable ordinaire, un pointeur peut être stocké en mémoire
   ou dans un registre.

   On peut déclarer des pointeurs de n'importe quoi, y compris des pointeurs
   de fonctions et des pointeurs de pointeurs.

   La déclaration spécifie le type d'objet pointé pour :
   + vérification de cohérence de type :
                --> interdit var_float = * ptr_char
   + savoir combien d'octets lire ou écrire (application de * sur le pointeur)
   + ne change pas la taille du pointeur

   Tous les pointeurs ont la même taille : celle d'une adresse mémoire.

char c = 'a';
char c2 = 'c';
unsigned short s,s2;
register unsigned short r1, r2;

register short *rps,*rp2;       /* deux pointeurs stockés dans des registres */

char *pcarac;                   /* implicitement initialisé à NULL */
unsigned short *ps = &s;                /* avec initialisation */
        @    r1 : registre r1
        @    r2 : registre r2
        @   rps : registre r3  (choix arbitraire du registre)
        @   rp2 : registre r4  (choix arbitraire du registre)
        @   r9, r10 stockage temporaire

        .data
c:      .byte   'a'
c2:     .byte   0x63    @ ASCII (c) = 0x63

        .balign 4       @ ps qui suit est stocké sur 4 octets
ps:     .word   s

        .bss
        .balign 2
s:      .skip 2         @ pas de valeur initiale dabs bss --> skip
s2:     .skip 2

        .balign 4       @ pcarac qui suit est stocké sur 4 octets
pcarac: .skip   4

IV) Pointeurs : affectation

--> appliquer la même méthode que pour une variable ordinaire

Stockage dans un registre

    r1 = 0x1234                 ldr r1,= @x1234

    rps = &s                    ldr r3,=s
```

```
Stockage en mémoire

     c = 'b'                        mov   r9, #'b'  @ *&c = 'b'
                                    ldr   r10,= c
                                    str    r9, [r10]

     ps = &s2                       ldr   r9,= s2
                                    ldr  r10,= ps
                                    str  r9, [r10]

V) Pointeurs : utilisation ("déréférencement")

Pointeur stocké dans un registre

   r1 = *rps    /* r1 = s puisque rps contient &s */    ldrh  r1, [r3]
   *rps = r1     /* s = r1 /                            strh  r1, [r3]

Pointeur stocké en mémoire

   r1 = *ps   --> r1 = * *&ps   ldr     r10,= ps        @ r10 = &ps
                                ldr     r9, [r10]       @ r9 = *&ps
                                ldrh    r1, [r9]        @ r1 = * *&ps

/* ldrh : type unsigned short * et ldrsh : type short */

 *ps = r1     --> * *&ps = r1   ldr     r10,=ps         @ r10 = &ps
                                ldr     r9,  [r10]      @ r9  = *&ps
                                strh    r1, [r9]        @ * *&ps = r1

VI) Pointeur de structure

Acces à une structure via un pointeur : comme pour une variable ordinaire.

struct _s ptstruct;

ptstruct = &s2;
(*ptstruct).cy='u';   // réalise s2.cy = 'u'

La notation pointeur->cy est un raccourci pratique pour
(*pointeur.cy) :

   ldr  r0,=ptstruct         // r0=&pstruct
   ldr  r0,[r0]              // r0=*&ptstruct (r0=ptstruct)
   mov  r1,#'u'
   strb r1,[r0,#DELTA_STRUCT_S_CY]  // *(*&ptstruct).cy = 'u'
```