INF401

TP3et4 « Codage des données »

Shaghayegh HAJMOHAMMADKASHI , Kimia KHADEMLOU– MIN3

3.1 Déclaration de données en langage d'assemblage :

Chaque ligne comporte une adresse puis un certain nombre d'octets et enfin leur correspondance sous forme d'un caractère (quand cela a un sens).

Dans quelle base les informations sont-elles affichées ? « Les informations sont affichées en hexadécimal. »

Combien d'octets sont-ils représentés sur chaque ligne ? « Chaque ligne représente généralement 16 octets, bien que cela puisse varier en fonction de la largeur de la console ou du formatage spécifique utilisé par l'outil arm-eabi-objdump. »

Donnez pour chacun des octets affichés la correspondance avec les valeurs d´déclarées dans la zone data ?

Pour déterminer la correspondance entre les octets affichés et les valeurs déclarées dans la zone de données, nous devons examiner chaque déclaration dans le fichier assembleur d'origine et la comparer aux octets affichés dans le fichier objet.

Comment est codée la chaîne de caractères, en particulier comment est représentée la fin de cette chaîne ?

- Pour le caractère 'A' déclaré comme « aa »: .byte 65, on devrait s'attendre à voir l'octet « 41 » en hexadécimal (ASCII pour 'A').
- Pour l'entier 15 déclaré comme « oo »: .byte 15, on devrait s'attendre à voir l'octet « 0F » en hexadécimal.
- Pour la chaîne "bonjour" déclarée comme « cc »: .asciz "bonjour", chaque caractère de la chaîne sera représenté en ASCII et suivi d'un octet « null (00) » pour marquer la fin de la chaîne.
- Pour la paire de valeurs <'B', 3> déclarée comme « rr »: .byte 66, .byte 3, on devrait s'attendre à voir les octets « 42 » et « 03 » en hexadécimal.
- Pour le tableau d'entiers [0x1122, 0x3456, 0xfafd] déclaré comme « T »: .hword 0x1122, .hword 0x3456, .hword 0xfafd, chaque entier de 16 bits sera représenté sur deux octets dans l'ordre big-endian.
- Pour l'entier 65 déclaré comme « xx »: .byte 65, on devrait s'attendre à voir l'octet « 41 » en hexadécimal.

La fin de la chaîne de caractères est codée par un octet null (00), ce qui indique la fin de la chaîne lorsqu'elle est lue en tant que chaîne ASCII.

Compilez donnees2.s. Quelle est maintenant la représentation de chacun des entiers de la zone data modifiée ?

```
.data  @ Début de la zone de données

    aa:     .byte 'A'     @ Caractère 'A'
    oo:     .word 15      @ Entier 15 sur 32 bits
    cc:     .asciz "bonjour"   @ Chaîne de caractères "bonjour"
    rr:     .byte 'B'     @ Caractère 'B'
            .word 3       @ Entier 3 sur 32 bits
    T:      .word 0x1122  @ Tableau d'entiers sur 32 bits
            .word 0x3456
            .word 0xfafd
    xx:     .word 65      @ Entier 65 sur 32 bits

@end_data  @ Fin de la zone de données
```

Dans la représentation hexadécimale fournie par la sortie de la commande arm-eabi-objdump, chaque octet est affiché en paires de chiffres hexadécimaux. Voici comment interpréter la représentation de chacun des entiers de la zone de données modifiée :

1) Caractère 'A' (aa) :

- Représentation hexadécimale : 41
- Ce caractère est représenté par l'octet 41 en hexadécimal.

2) Entier 15 (oo) :

- Représentation hexadécimale : 00 00 00 0F
- Cet entier est représenté sur 32 bits (4 octets) avec la valeur 0x0F placée à l'octet de poids le plus faible (le plus à droite) en format big-endian, ce qui signifie que les octets sont ordonnés du plus significatif au moins significatif.

3) Chaîne de caractères "bonjour" (cc) :

- Représentation hexadécimale : 62 6F 6E 6A 6F 75 72 00 42
- Chaque caractère ASCII de la chaîne est représenté par son code hexadécimal correspondant.

4) Paire de valeurs <B, 3> (rr) :

- Représentation hexadécimale : 42 00 00 00 03
- Le caractère 'B' est représenté par l'octet 42, suivi de l'entier 3 encodé sur 32 bits avec la valeur 0x03 placée à l'octet de poids le plus faible en format big-endian.

5) Tableau d'entiers [0x1122, 0x3456, 0xfafd] (T) :

- Représentation hexadécimale : 11 22 00 00 34 56 00 00 FA FD 00 00
- Chaque entier est représenté sur 32 bits (4 octets) en format big-endian.

6) Entier 65 (xx) :

- Représentation hexadécimale : 00 00 00 41
- Cet entier est représenté sur 32 bits avec la valeur 0x41 placée à l'octet de poids le plus faible en format big-endian.

En résumé, chaque entier dans la zone de données modifiée est représenté sur 32 bits en utilisant le format big-endian, où les octets sont ordonnés du plus significatif au moins significatif.

3.2 Accès à la mémoire : échange mémoire/registres :

3.2.1 Lecture d'un mot de 32 bits :

Exécutez ce programme : arm-eabi-run accesmem. Notez les valeurs affichées. Que représente chacune d'elle ?

Lorsqu'on exécute le programme `accesmem` en utilisant la commande `arm-eabi-run accesmem`, on obtient les valeurs suivantes affichées :

1. `0001f440`
2. `0000010a`

Maintenant, on explique ce que représentent ces valeurs :

```
@ accesmem.s
      .data
aa:  .word 24
xx:  .word 266
bb:  .word 42
```

Ligne 1 est un commentaire indiquant le nom du fichier (accesmem.s).

Section .data où les données sont déclarées.

aa, xx, et bb sont des étiquettes pour les mots de 32 bits déclarés avec les valeurs 24, 266 et 42 respectivement.

```
LDR r5, LD_xx
LDR r6, [r5]
```

Chargement de l'adresse xx dans le registre r5, puis chargement du contenu de la mémoire à cette adresse dans le registre r6.

```
@ impression du contenu de r5
   MOV r1, r5
   BL EcrHexa32

@ impression du contenu de r6
   MOV r1, r6
   BL EcrHexa32
```

Chargement de l'adresse xx dans le registre r5, puis chargement du contenu de la mémoire à cette adresse dans le registre r6.

Affichage du contenu des registres r5 et r6 en appelant une fonction EcrHexa32 qui affiche les valeurs en hexadécimal.

La sortie qu'on obtient semble être des valeurs en hexadécimal représentant les contenus des registres r5 et r6. Voici comment interpréter ces valeurs :

1. 0001f440 : Cette valeur correspond au contenu du registre r5 après l'instruction LDR r5, =xx. Elle représente l'adresse mémoire où l'entier 266 est stocké.

2. 0000010a : Cette valeur correspond au contenu du registre r6 après l'instruction LDR r6, [r5]. Elle représente le contenu de la mémoire à l'adresse 266 (la valeur stockée à l'adresse xx).

Si on convertit ces valeurs en décimal, on obtiendra :

1. 0001f440 en hexadécimal est équivalent à 128064 en décimal, ce qui est l'adresse mémoire où l'entier 266 est stocké.

2. 0000010a en hexadécimal est équivalent à 266 en décimal, qui est la valeur stockée à l'adresse xx.

Donc, le programme charge l'adresse de xx dans r5, puis charge la valeur stockée à cette adresse dans r6, et enfin imprime ces valeurs en hexadécimal.

3.2.2 Lecture de mots de tailles différentes :

```
@ accesmem2.s
.data
D1: .word 266
D2: .hword 42
D3: .byte 12

.text
.global main
main:
    @ Chargement de l'adresse D1 en r3
    LDR r3, LD_D1
    @ Chargement de la valeur en mémoire à l'adresse D1 dans r4
    LDR r4, [r3]
    @ Affichage de l'adresse de D1
    MOV r1, r3
    BL EcrHexa32
    @ Affichage de la valeur lue à l'adresse D1
    MOV r1, r4
    BL EcrNdecimal32

    @ Chargement de l'adresse D2 en r5
    LDR r5, LD_D2
    @ Chargement de la valeur en mémoire à l'adresse D2 dans r6
    LDRH r6, [r5]
    @ Affichage de l'adresse de D2
    MOV r1, r5
    BL EcrHexa32
    @ Affichage de la valeur lue à l'adresse D2
    MOV r1, r6
```

```
    BL EcrNdecimal16

    @ Chargement de l'adresse D3 en r7
    LDR r7, LD_D3
    @ Chargement de la valeur en mémoire à l'adresse D3 dans r8
    LDRB r8, [r7]
    @ Affichage de l'adresse de D3
    MOV r1, r7
    BL EcrHexa32
    @ Affichage de la valeur lue à l'adresse D3
    MOV r1, r8
    BL EcrNdecimal8

    @ Fin du programme
    B fin

fin: B exit
LD_D1: .word D1
LD_D2: .word D2
LD_D3: .word D3
```

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run accesmem2
0001f474
266
0001f478
42
0001f47a
12
```

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run accesmem2
0001f474
0000010a
0001f478
0000002a
0001f47a
0000000c
```

ça c'est pour le moment qu'on a mis tous EcrHexa32.

Adresse 0001f474 - Valeur 0000010a :

La valeur 0000010a est en hexadécimal, ce qui équivaut à 266 en décimal.

Ainsi, l'adresse 0001f474 est l'endroit où est stockée la valeur 266 dans la mémoire.

Adresse 0001f478 - Valeur 0000002a :

La valeur 0000002a est en hexadécimal, ce qui équivaut à 42 en décimal.

Par conséquent, l'adresse 0001f478 est l'emplacement mémoire où est stockée la valeur 42.

Adresse 0001f47a - Valeur 0000000c :

La valeur 0000000c est en hexadécimal, ce qui équivaut à 12 en décimal.

L'adresse 0001f47a est donc l'endroit où est stockée la valeur 12 dans la mémoire.

3.2.3 Ecriture en mémoire :

Il semble que le programme fonctionne comme prévu. Voici ce qui se passe :

- Avant de stocker la valeur -10 dans DW, la valeur stockée à cette adresse est 0, ce qui est affiché en hexadécimal comme 00000000 et en décimal comme 0.
- Après avoir stocké -10 dans DW, la nouvelle valeur est -10, ce qui est affiché en hexadécimal comme fffffff6 (comme les nombres sont stockés en complément à deux, -10 en hexadécimal est 0xFFFFFFF6) et en décimal comme -10.

Le programme semble donc fonctionner comme prévu, stockant -10 à l'adresse DW et affichant correctement les valeurs avant et après la modification.

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-gcc -c ecrmem.s
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-gcc -o ecrmem ecrmem.o es.o
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run ecrmem
00000000
0
fffffff6
-10
```

Modifiez le programme ecrmem.s pour faire le même genre de travail mais avec un mot de 16 bits rangé à l'adresse DH et un mot de 8 bits rangé `a l'adresse DB.

Dans ce programme :

1. J'ai utilisé .hword pour déclarer un mot de 16 bits à l'adresse DH et .byte pour déclarer un mot de 8 bits à l'adresse DB.
2. Pour accéder à ces adresses dans le code, j'ai utilisé les directives LD_DH et LD_DB.
3. J'ai utilisé les instructions LDRH pour charger un mot de 16 bits à partir de l'adresse DH et LDRB pour charger un mot de 8 bits à partir de l'adresse DB.
4. Les valeurs chargées sont ensuite affichées en hexadécimal et en décimal.

```
.data
DW: .word 0
DH: .hword 20  @ Modification : Utilisation de .hword pour un mot de 16 bits
DB: .byte 1   @ Modification : Utilisation de .byte pour un mot de 8 bits

.text
.global main
main:
    @ Affichage de la valeur Ã  l'adresse DW
    LDR r0, LD_DW
    LDR r1, [r0]
    BL EcrHexa32
    BL EcrZdecimal32

    @ Modification de la valeur Ã  l'adresse DW
    MOV r4, #-10
    STR r4, [r0]

    @ Affichage de la nouvelle valeur Ã  l'adresse DW
    LDR r0, LD_DW
    LDR r1, [r0]
    BL EcrHexa32
    BL EcrZdecimal32

    @ Affichage de la valeur Ã  l'adresse DH
    LDR r0, LD_DH
    LDRH r1, [r0]  @ Utilisation de LDRH pour charger un mot de 16 bits
    BL EcrHexa16
    BL EcrZdecimal32

    @ Affichage de la valeur Ã  l'adresse DB
    LDR r0, LD_DB
    LDRB r1, [r0]  @ Utilisation de LDRB pour charger un mot de 8 bits
    BL EcrHexa8
    BL EcrZdecimal32

fin: B exit

LD_DW: .word DW
LD_DH: .word DH
LD_DB: .word DB
```

4.1 Un premier programme en langage d'assemblage :

Modifiez ce programme pour qu'il affiche la chaîne "BONJOUR" sur une ligne puis la chaine "au revoir..." sur la ligne suivante.

Modifiez le programme en ajoutant une suite d'instructions qui transforme chaque caractère majuscule en minuscule.

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run caracteres
BONJOUR
bonjour
au revoir...
```

```
.data
cc: @ ne pas modifier cette partie
    .byte 0x42
    .byte 0x4f
    .byte 0x4e
    .byte 0x4a
    .byte 0x4f
    .byte 0x55
    .byte 0x52
    .byte 0x00     @ code de fin de chaine
    @ la suite pourra Ã²tre modifiÃ©e
    .word 12
    .word 0x11223344

cd: .asciz "au revoir..."

.text
.global main
main:
    @ impression de la chaine de caractere d'adresse cc
    ldr r1, LD_cc
    bl EcrChaine

    @ modification de la chaine d'adresse cc
    ldr r1, LD_cc
    ldrb r2, [r1]   @ Charger le premier caractÃ¨re 'B'
    add r2, r2, #32  @ Convertir en minuscule
    strb r2, [r1]   @ Sauvegarder le premier caractÃ¨re modifiÃ©

    ldr r1, LD_cc
    ldrb r2, [r1, #1]   @ Charger le deuxiÃ¨me caractÃ¨re 'O'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #1]   @ Sauvegarder le deuxiÃ¨me caractÃ¨re modifiÃ©
I
    ldr r1, LD_cc
    ldrb r2, [r1, #2]   @ Charger le troisiÃ¨me caractÃ¨re 'N'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #2]   @ Sauvegarder le troisiÃ¨me caractÃ¨re modifiÃ©

    ldr r1, LD_cc
    ldrb r2, [r1, #3]   @ Charger le quatriÃ¨me caractÃ¨re 'J'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #3]   @ Sauvegarder le quatriÃ¨me caractÃ¨re modifiÃ©
```

```
    ldr r1, LD_cc
    ldrb r2, [r1, #4]   @ Charger le cinquiÃ¨me caractÃ¨re 'O'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #4]   @ Sauvegarder le cinquiÃ¨me caractÃ¨re modifiÃ©

    ldr r1, LD_cc
    ldrb r2, [r1, #5]   @ Charger le sixiÃ¨me caractÃ¨re 'U'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #5]   @ Sauvegarder le sixiÃ¨me caractÃ¨re modifiÃ©

    ldr r1, LD_cc
    ldrb r2, [r1, #6]   @ Charger le septiÃ¨me caractÃ¨re 'R'
    add r2, r2, #32      @ Convertir en minuscule
    strb r2, [r1, #6]   @ Sauvegarder le septiÃ¨me caractÃ¨re modifiÃ©

    @impression de la chaine cc modifiee
    ldr r1 , LD_cc
    bl EcrChaine
    @ impression de la chaine au revoir
    ldr r1, LD_cd
    bl EcrChaine



    fin: B exit  @ terminaison immÃ©diate du processus (plus tard on saura faire mieux)

LD_cc: .word cc
LD_cd: .word cd
```

4.2 Alignements et ”petits bouts”:

4.2.1 Questions d’alignements:

Alignements1.s:

alignements1.s - /home/h/hajmohas/INF401/TP3et4/@im2ag-turing-01

File Edit Search Preferences Shell Macro Windows

```
x: .byte 0x01
y: .byte 0x02
z: .byte 0x04
a: .word 0x0A0B0C0D
b: .byte 0x08

.text
.global main
main:
    @ Lecture et impression de la valeur de x
    LDR r2, LD_x
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de y
    LDR r2, LD_y
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de z
    LDR r2, LD_z
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de a
    LDR r2, LD_a
    LDR r1, [r2]
    BL EcrHexa32

    @ Lecture et impression de la valeur de b
    LDR r2, LD_b
    LDR r1, [r2]
    BL EcrHexa8


fin:    B exit

LD_x:   .word   x
LD_y:   .word   y
LD_z:   .word   z
LD_a:   .word   a
LD_b:   .word   b
```

Que constatez-vous ?

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run alignements1
01
02
04
0402010d
08
```

Après avoir analysé les valeurs affichées et le problème d'alignement mentionné, voici ce qu’on constates :

- Les valeurs affichées pour x, y, et z semblent correctes et cohérentes avec les valeurs définies dans la section .data. Cela suggère que ces valeurs ont été correctement chargées et affichées.
- La valeur affichée pour « a » semble incorrecte. Cela indique qu'il y a eu un problème lors du chargement ou de l'affichage de la valeur de a. Ce problème peut être dû à un alignement incorrect des données.

- La valeur affichée pour b semble également correcte, ce qui suggère que le problème d'alignement a été résolu pour les données de 8 bits.
  Ainsi, pour résoudre le problème d'alignement, on a inséré la directive .balign 4 avant la déclaration de a, ce qui assure que a est aligné sur une adresse multiple de 4. Cela corrige l'erreur d'alignement pour les mots de 32 bits.

Alignements2.s:

alignements2.s - /home/h/hajmohas/INF401/TP3et4/@im2ag-turing-01

File Edit Search Preferences Shell Macro Windows

```
y: .byte 0x02
z: .byte 0x04
.balign 4  @ Pour aligner sur une adresse multiple de 4
a: .word 0x0A0B0C0D
b: .byte 0x08

.text
.global main
main:
    @ Lecture et impression de la valeur de x
    LDR r2, LD_x
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de y
    LDR r2, LD_y
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de z
    LDR r2, LD_z
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de a
    LDR r2, LD_a
    LDR r1, [r2]
    BL EcrHexa32

    @ Lecture et impression de la valeur de b
    LDR r2, LD_b
    LDR r1, [r2]
    BL EcrHexa8


fin:    B exit

LD_x:   .word   x
LD_y:   .word   y
LD_z:   .word   z
LD_a:   .word   a
LD_b:   .word   b
```

Vérifiez que le problème est résolu.

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run alignements2
01
02
04
0a0b0c0d
08
```

- La valeur de x est correctement affichée comme 01.
- La valeur de y est correctement affichée comme 02.
- La valeur de z est correctement affichée comme 04.
- La valeur de a est correctement affichée comme 0a0b0c0d.
- La valeur de b est correctement affichée comme 08.

Cela confirme que le problème d'alignement pour les mots de 32 bits a été résolu avec succès, et toutes les valeurs sont correctement affichées.

Ecrivez un programme dans lequel vous déclarez une valeur représentée sur 16 bits et dont l'adresse n'est pas un multiple de 2 (il suffit de placer un octet devant)

alignements3.s - /home/h/hajmohas/INF401/TP3et4/@im2ag-turing-01

File Edit Search Preferences Shell Macro Windows

```
.data
x: .byte 0x01
y: .byte 0x02
z: .byte 0x04
a: .hword 0xABCD
b: .byte 0x08

.text
.global main
main:
    @ Lecture et impression de la valeur de x
    LDR r2, LD_x
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de y
    LDR r2, LD_y
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de z
    LDR r2, LD_z
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de a
    LDR r2, LD_a
    LDR r1, [r2]
    BL EcrHexa16

    @ Lecture et impression de la valeur de b
    LDR r2, LD_b
    LDR r1, [r2]
    BL EcrHexa8


fin:    B exit

LD_x:   .word   x
LD_y:   .word   y
LD_z:   .word   z
LD_a:   .word   a
LD_b:   .word   b
```

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run alignements3
01
02
04
01cd
08
```

La valeur de a est incorrecte car on utilise LDR pour charger une valeur sur 32 bits à partir de l'adresse de a. Cependant, la valeur de a est sur 16 bits, et son adresse n'est pas alignée sur un multiple de 2.

alignements4.s - /home/h/hajmohas/INF401/TP3et4/@im2ag-turing-01

File Edit Search Preferences Shell Macro Windows

```
.data
x: .byte 0x01
y: .byte 0x02
z: .byte 0x04
.balign 2
a: .hword 0xABCD
b: .byte 0x08

.text
.global main
main:
    @ Lecture et impression de la valeur de x
    LDR r2, LD_x
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de y
    LDR r2, LD_y
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de z
    LDR r2, LD_z
    LDR r1, [r2]
    BL EcrHexa8

    @ Lecture et impression de la valeur de a
    LDR r2, LD_a
    LDR r1, [r2]
    BL EcrHexa16

    @ Lecture et impression de la valeur de b
    LDR r2, LD_b
    LDR r1, [r2]
    BL EcrHexa8


fin:    B exit

LD_x:   .word   x
LD_y:   .word   y
LD_z:   .word   z
LD_a:   .word   a
```

```
hajmohas@im2ag-turing-01:[~/INF401/TP3et4]: arm-eabi-run alignements4
01
02
04
abcd
08
```

Le balign 2 est utilisé pour garantir que les variables 16 bits sont stockées à des adresses multiples de 2, ce qui peut améliorer les performances et la fiabilité du programme

4.2.2 Questions de ”petits bouts” :

Pour le fichier "accesmem":

```
1f438 00000000 18000000 0a010000 2a000000  ............*...
```

- aa est défini à l'adresse mémoire 0x1f43c avec la valeur hexadécimale 0x18000000, qui correspond à 24 en décimal.
- xx est défini à l'adresse 0x1f440 avec la valeur hexadécimale 0x0a010000, qui correspond à 266 en décimal.
- bb est défini à l'adresse 0x1f444 avec la valeur hexadécimale 0x2a000000, qui correspond à 42 en décimal.

Trois mots de 32 bits sont définis: aa, xx et bb.

aa contient la valeur décimale 24 (hexadécimal 18 00 00 00), xx contient 266 (hexadécimal 0a 01 00 00) et bb contient 42 (hexadécimal 2a 00 00 00).

Les mots sont stockés en "petits bouts", les bits de poids faibles aux adresses les plus basses.

Les adresses de aa, xx et bb sont espacées de 4 octets (32 bits) car chaque mot occupe 4 octets.

Pour le fichier "accesmem2":

```
1f470 00000000 0a010000 2a000c00 25630a00  ........*...%c..
```

- D1 est défini à l'adresse 0x1f474 avec la valeur hexadécimale 0x0a010000, qui correspond à 266 en décimal.
- D2 est défini à l'adresse 0x1f48 avec la valeur hexadécimale 0x2a00, qui correspond à 42 en décimal.
- D3 est défini à l'adresse 0x1f47a avec la valeur hexadécimale 0x0c, qui correspond à 12 en décimal.

Trois mots sont définis: D1, D2 et D3.

D1 est un mot de 4 octets contenant la valeur décimale 266 (hexadécimal 0a 01 00 00), D2 est un mot de 2 octets contenant 42 (hexadécimal 2a 00 00 00) et D3 est un mot d'1 octet contenant 12 (hexadécimal 0c 00).

Les mots sont stockés en "petits bouts".

Les adresses de D1, D2 et D3 sont espacées de manière cohérente avec leur taille:

D1 et D2 sont espacés de 4 octets car D1 occupe 4 octets.

D2 et D3 sont espacés de 2 octets car D2 occupe 2 octets.