

La mémoire principale

M. Dubacq

S1D 2009

1 Alignement de données

Objectif : *Comprendre la déclaration d'un segment de données, comprendre les problèmes d'alignement de données.*

Soit le segment de données suivant sur une machine MIPS 32 bits :

```
.data
part0: .word 0xFE084312
part1: .word 25,32,65,-30
part2: .ascii "paracetamol"
part3: .ascii "totoro"
part4: .asciiz "paracetamol"
part5: .asciiz "dino2"
part6: .half 28
part7: .word 42
part8: .space 15
part9: .space 40 # tableau 10 entiers
```

La déclaration `.asciiz` déclare une chaîne de caractère avec un caractère nul à la fin, contrairement à la déclaration `.ascii`.

1. Sur une machine 32 bits, rappelez la taille prise en mémoire par une donnée de type `word`, `half`, `byte`. Est-ce que ceci changerait sur une machine 64 bits ? *Un word est de 4 octets, un half de 2 octets, un byte est évidemment d'un octet. Les deux premiers sont susceptibles de changer dans un processeur 64 bits.*
2. Sachant que l'adresse de `part0` est `0x10010000`, dire quelle est l'adresse de `part1`, `part2`, `part3`, `part4`. *part1 vaut part0+4, soit 0x10010004 (je ne répéterai plus le 0x100100 dans la suite). part2 est 16 octets plus loin, donc 0x...14. part3 est 11 octets plus loin (.ascii est la chaîne de caractères sans caractère nul terminal), et part4 encore six octets plus loin, soit 0x...1f et 0x...25.*
3. On s'intéresse à la représentation du premier nombre (à l'adresse `part0`). Quel octet trouve-t-on à l'adresse `0x10010001` si on est sur un processeur *little-endian* ? Et sur un processeur *big-endian* ? Expliquez en faisant une représentation de la mémoire entre l'adresse `part0` et l'adresse `part1` dans les deux cas. *Dans le premier cas (little endian, petit-boutien), les octets de poids faibles sont dans les adresses basses : 0x12 à l'adresse 0x10010000 puis 0x43 à l'adresse 0x10010001 puis 0x08 puis 0xFF. Dans l'autre cas (big endian, grand-boutien), on a 0xFF à l'adresse 0x10010000 puis 0x08 à l'adresse 0x10010001 puis 0x43 puis 0x12.*
On pourra leur rappeler les pages du cours correspondantes au cas où.
4. Donnez ensuite l'adresse de `part5`. Expliquez la différence avec `part3`. *.asciiz est la chaîne de caractères avec zero terminal, donc 12 octets plus loin, pas 11 : 0x...31.*
5. Donnez l'adresse de `part6`. Expliquez pourquoi. *La donnée est placée automatiquement sur une adresse multiple de 2 (puisque la donnée est un demi-mot, de taille 2 octets). Donc au lieu d'être 0x...37, c'est 0x...38 (on perd un octet).*
6. Donnez de même l'adresse de `part7`. *Même chose, mais avec un multiple de 4 : au lieu de 0x3a, on a 0x3c.*
7. Concluez en donnant l'adresse de `part8` et de `part9`. Que va-t-il se passer lorsqu'on essaiera de faire `Load R1,part9` ? *part8 sera décalé de 64 octets (0x40) et part9 de 79 (0x4f). Si on essaie de faire Load (sous-entendu : LoadWord) à l'adresse part9, on essaiera de charger une donnée de taille 4 depuis une adresse non-multiple de 4 : on obtiendra un beau Unaligned address in inst/data fetch : 0x1001004f, ce qui veut dire « erreur d'alignement ». Le processeur ne pourra pas traiter l'instruction.*

2 Overclocking et mémoire

Objectif : *Comprendre les rapports numériques entre la mémoire et le processeur.*

L'*overclocking* est une technique qui consiste en l'accélération de l'horloge d'un processeur pour obtenir des performances meilleures. Toutefois, la cadence du bus de données, du northbridge et de la mémoire sont toutes déterminées à partir de la cadence du processeur en divisant par un entier (le multiplicateur). Cet entier est, sur certaines cartes-mères, réglable.

Une RAM est prévue pour fonctionner à 333 MHz. La vitesse du processeur qui va avec est de 3 GHz. En fait, cette RAM est capable de fonctionner sans problèmes jusqu'à 400 MHz, et le processeur peut être accéléré jusqu'à 10% sans défauts (il peut être accéléré uniquement par paliers de 100 MHz).

1. Expliquez pourquoi accélérer modérément la cadence du processeur peut amener à des meilleures performances, et pourquoi l'accélérer énormément peut ne pas fonctionner du tout. *Le processeur continue son cycle d'instruction en fonction de l'horloge, et donc l'exécute plus rapidement. Un cycle d'instruction plus rapide, c'est au total plus d'instructions par seconde. Mais pour que ça marche, il faut que les temps de réaction des circuits électroniques ne soient pas trop grands pour que les communications prévues lors de la conception du chemin de donnée aient le temps de se faire. Si on multiplie par 100 la fréquence, il y a de bonnes chances que les circuits combinatoires n'aient plus le temps de se stabiliser avant la fin du cycle d'horloge.*
2. Quel est le multiplicateur utilisé pour obtenir la cadence nominale? *Le multiplicateur est 3 GHz/333 MHz, soit de 9.*
3. Peut-on réaliser une accélération de la cadence du processeur qui ne crée aucun défaut sans changer le multiplicateur? Que deviendrait la vitesse de la mémoire si on le faisait? *Si on garde le multiplicateur de 9, on peut faire monter le processeur à 3300 MHz, et dans ce cas, la vitesse de la mémoire devient 366 MHz, soit une augmentation de 10% environ (aussi).*
4. Peut-on avoir une accélération du processeur et de la mémoire qui privilégie un peu plus la mémoire? *Si on garde le multiplicateur, non. Mais on peut changer le multiplicateur (par exemple mettre 8). On accélère alors le processeur de 200 MHz, ce qui donne un processeur à 3200 MHz, et une mémoire à 400 MHz. L'amélioration de la mémoire est alors de 20%, et celle du processeur de 7%. Selon qu'on ait souvent besoin de la mémoire ou pas, une solution ou l'autre pourra être un peu plus avantageuse (la deuxième l'est souvent, en pratique).*
5. Si on a un bus de données de largeur 32 bits, quel est le débit de la mémoire? Quel est son débit maximal? *Le nombre de cycles mémoire par seconde est de 333×10^6 cycles/seconde et donc le débit est de $32 \times 333 \times 10^6 = 10.67 \times 10^9$ bits/seconde (ou 10.67 Gb/s).*

3 Débogage en C

3.1 TP en salle machine

Objectif : *Savoir se servir d'un débogueur.*

1. En salle machine, connectez-vous sous Linux. Récupérez le programme `td09.c`. Le programme est disponible sur <http://www-info.iutv.univ-paris13.fr/jcdubacq/td09.c>
2. Ce programme comporte des erreurs. Essayez de les repérez, ne les corrigez pas. Le but de ce programme est de remplir un tableau avec des valeurs, puis de faire des modifications de ces valeurs (multiplier par deux, puis ajouter 1).
3. Compilez le programme, et exécutez-le avec le débogueur `ddd`. Le listing du programme doit s'afficher.
4. Commencez, avant de faire tourner le programme par insérer un point d'arrêt au début du `main`, un point d'arrêt dans la toute première boucle, un point d'arrêt dans la fonction `fact`.
5. Faites afficher la fenêtre de données de `ddd`. En faisant un clic droit, faites afficher toutes les variables du programme. Que se passe-t-il si on essaye d'afficher `NUM`? Que remarque-t-on quand on essaye d'afficher la variable `i` de la fonction `fact`? *NUM ne peut pas être affiché, car c'est une constante et non pas une variable. En fait, cette constante est remplacée directement dans le code source.*
6. Remarquez la différence d'affichage entre `b` et `c`. À quoi cela peut-il être dû? *La différence est due au fait que `b` est déclarée comme un tableau de taille statique, et non comme un pointeur. Même si c'est le même genre de valeur (comme on le verra par la suite), il est possible au débogueur de donner une sémantique différentes à ces données.*
7. Dans un terminal, testez le programme. Vérifiez qu'il ne fonctionne pas. *Boucle infinie, donc il faut l'arrêter par Control-C.*
8. Commencez l'exécution du programme (bouton `run` ou menu équivalent). Après le premier arrêt, utilisez le bouton `step` pour avancer pas à pas dans le programme. Regardez évoluer peu à peu les valeurs des variables. Que remarque-t-on sur la valeur affichée de `i` lorsqu'on rentre dans l'évaluation de la fonction `fact`? *Elle prend la valeur de l'intérieur de la fonction, et non plus celle du programme général.*
9. Quelle est la première erreur? Corrigez-là, recompilez (de l'intérieur du débogueur, avec le bouton `make`). *La première erreur est à l'intérieur de la fonction (`un = au lieu d'un !=`, sans doute une erreur de frappe?).*

10. Refaites une exécution pas-à-pas. Vérifiez que la fonction renvoie maintenant la bonne valeur. Que constate-t-on dans l’affichage du tableau `b`? Trouvez la deuxième erreur. *Il s’agit d’une erreur dans l’affectation (`NUM` au lieu de `i`)*
11. Faites afficher *en partant de la zone de données* le contenu pointé par `c`. Quelle est la troisième erreur? Corrigez-la. *Il faut faire une affectation de `b` à `c`, et non à `*c`. En effet, `c` est un pointeur d’entiers, donc analogue à un tableau. L’erreur doit être manifeste dans une exécution pas à pas (la valeur de `c` ne doit pas changer, et le programme doit afficher quelque chose dans la partie du bas de la fenêtre (chez moi : `Program received signal EXC_BAD_ACCESS, Could not access memory. Reason : KERN_PROTECTION_FAILURE at address : 0x00000000`)).*
12. Quelle est l’adresse du premier élément du tableau `b`? *Il suffit de regarder la valeur de `c` juste après l’affectation de `b` à `c`.*
13. Surveillez pas-à-pas l’exécution de la deuxième boucle du programme (qui multiplie par 2). Que remarque-t-on sur le tableau `b`? Expliquez. *Les valeurs changent. Parce qu’on modifie des valeurs qui sont stockées en mémoire à l’adresse contenue dans `c` ou pas loin (rappel : `c[i]=*(c+i)`), et que `c` vaut `b`, on modifie donc la mémoire à l’endroit où est stocké le tableau `b`.*
14. Il serait plus simple de faire afficher `c` comme un tableau de 6 éléments. Utilisez la syntaxe donnée dans le cours pour le faire. Vérifiez que les valeurs sont changées aussi bien dans `c` que dans `b`. *Il faut taper `graph display c[0]@6`. Pour faire joli, on peut même rajouter (si `c` correspond à l’affichage numéro 3) ... `dependent on 3`.*
15. La dernière boucle utilise l’arithmétique des pointeurs : un pointeur (`c`) change de valeur et pointe tour à tour. Elle comporte une erreur. Trouvez-la et corrigez-la. *Lorsqu’on fait le test d’arrêt de la boucle, il faut comparer non pas `c` à `*d`, mais `c` à `d`. NB : cette question est plus difficile, mais ils ont le même problème en assembleur.*
16. Est-il correct d’affecter une valeur à `*d`? Pourquoi? *C’est possible, mais de la même façon qu’en changeant `c[i]`, on changeait dans le tableau l’élément `b[i]`, en écrivant au-delà de la fin du tableau (ce qu’est `d` puisque `d` pointe sur le `NUM+1`-ième élément du tableau), on risque d’écrire n’importe où dans la mémoire. Y compris dans une variable.*

3.2 Pour ceux qui finissent en avance

- Réécrivez le programme précédent en faisant de `b` non pas un tableau statique, mais un tableau alloué dynamiquement. N’oubliez pas de libérer l’espace mémoire à la fin du tableau.
- Est-il possible de lire dans ce tableau après avoir libéré la mémoire? *Oui. Mais il n’y a aucune garantie que les informations restent en place (voir ci-après).*
- Après avoir libéré la mémoire (à la fin du programme), créez un nouveau tableau d’entiers (de taille similaire mais pas identique). Que constate-t-on? *Normalement, la place mémoire est réattribuée. Mais pas toujours, parce que parfois le `free` est identique à ne rien faire.*