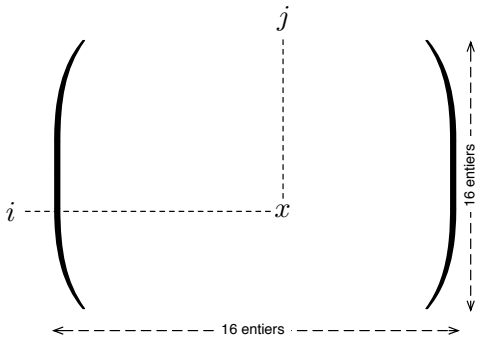
LI312 - Examen 2008

Benjamin BARON

1 Caches

Soit un processeur MIPS associé à un cache de données à correspondance directe. Ce cache possède une capacité totale de stockage de 1 Ko (= 1 024 octets). On rappelle que toutes les adresses émises par le processeur sont des adresses octets, et les adresses sont codées sur 32 bits. On ne s'intéresse dans cet exercice qu'au cache de données, et chaque ligne du cache de données possède une longueur de 64 octets.

On considère une matrice carrée A de 16×16 entiers : pour tout couple d'entiers (i, j), $0 \leqslant i, j \leqslant 15$, $A[i, j]$ désigne l'entier de la ligne i et la colonne j. La matrice est stockée en mémoire à l'adresse $A =$ `0x00000 17C00`. Chaque entier est représenté par un mot de 32 bits, l'entier $A[i, j]$ est ainsi stocké à l'adresse $A + (16 \times i + j) \times 4$.



Question 1.1. On rappelle qu'une case du cache est un emplacement pouvant contenir une ligne de cache complète.

Nombre NbC de cases du cache : $NbC = {}^{1024}/_{64} = {}^{2^{10}}/_{2^6} = 2^4 = 16$.

Une ligne de cache complète peut alors contenir ${}^{64}/_{4} = 16$ mots de 32 bits.

Le nombre de bits de l'*index* est donc égal à $\log_2(16) = 4$. L'index désigne une case dans le cache.

Le nombre de bits de l'*offset* est égal à $\log_2(64) = 6$. L'offset désigne un octet dans une case du cache.

Question 1.2. La matrice A s'étend de l'adresse

`0001 0111 1100 0000 0000` (`0x1 7C00`)

à l'adresse

`0001 0111 1111 1111 1100` (`0x1 7FFC`)

La seule adresse pouvant occuper le mot 0 (offset = 0) de la ligne 0 (index = 0) du cache est alors le mot $A[0, 0]$ placé à l'adresse `0x0001 7C00`.

D'une manière générale, les éléments $A[c, k]$ de la matrice A occuperont le mot k (offset = k) de la case c (index = c) du cache.

Question 1.3. On souhaite exécuter le code suivant (on suppose la variable S initialisée au préalable) :

```
for(j = 0; j < N; j++) {
    for(i = 0; i < N; i++) {
        S += A[i,j];
    }
}
```

Remplir le tableau ci-après pour la valeur $N = 3$ en indiquant pour les 9 sommes effectuées, la valeur du couple (i, j) et si l'accès au cache de données pour lire l'entier $A[i, j]$ est un succès ou un échec.

Somme	Valeur de (i, j)	Succès / échec
1	(0, 0)	Echec (remplissage de la ligne (0, 0) . . . (0, 15))
2	(1, 0)	Echec (remplissage de la ligne (1, 0) . . . (1, 15))
3	(2, 0)	Echec (remplissage de la ligne (2, 0) . . . (2, 15))
4	(0, 1)	Succès
5	(1, 1)	Succès
6	(2, 1)	Succès
7	(0, 2)	Succès
8	(1, 2)	Succès
9	(2, 2)	Succès

Taux de MISS obtenu : $3/9 = 1/3$.

Question 1.4. On considère maintenant le code suivant équivalent au précédent mais en permutant i et j :

```
for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++)
        S += A[i,j];
    }
```

Remplir le tableau suivant pour la valeur $N = 3$ en indiquant pour les 9 sommes effectuées, la valeur du couple (i, j) et si les accès au cache de données sont un succès ou un échec.

Somme	Valeur de (i, j)	Succès / échec
1	(0, 0)	Echec (remplissage de la ligne (0, 0) . . . (0, 15))
2	(0, 1)	Succès
3	(0, 2)	Succès
4	(1, 0)	Echec (remplissage de la ligne (1, 0) . . . (1, 15))
5	(1, 1)	Succès
6	(1, 2)	Succès
7	(2, 0)	Echec (remplissage de la ligne (2, 0) . . . (2, 15))
8	(2, 1)	Succès
9	(2, 2)	Succès

Taux de MISS obtenu : $3/9 = 1/3$.

2 Programmation assembleur

On désire manipuler une liste chaînée en mémoire dans la zone des variables globales initialisées. Chaque élément de cette liste est une structure contenant deux mots de 32 bits : un pointeur sur l'élément suivant (adresse sur 32 bits) et une valeur (entier sur 32 bits).

En C, on aurait :

```
struct liste {
    struct liste *NEXT;
    unisigned int VAL;
};
```

Question 2.1. Donner les directives d'assemblage permettant de réserver et d'initialiser dans le segment `.data` les 4 éléments de liste qui auront respectivement pour adresses : `elm1`, `elm2`, `elm3`, `elm4`, et pour valeurs : 23, 44, 52, 2.

```
.data
elm1: .word elm2, 23
elm2: .word elm3, 44
elm3: .word elm4, 52
elm4: .word 0, 2
```

Question 2.2. Convention d'utilisation de la pile.

Les trois fonctions de la pile sont :
- Transmission des arguments de la fonction appelante vers la fonction appelée ;
- Sauvegarde de la valeur des "registres persistants" ;
- Zone de travail : variables locales.

Il faut déplacer le pointeur de la pile dans le prologue d'une fonction :
`addiu $29, $29,` $-4 \times (nv + nr + \max(na_i))$
où :
- $na = \max(na_i, 4)$ où na_i représente ne nombre d'arguments des fonctions g_i appelées par f ;
- nv est le nombre de variables locales utilisées en C ;
- nr est le nombre de registres persistants utilisés + \$31.

La fonction renvoie la valeur de retour est conventionnellement écrite dans le registre \$2 par la fonction appelée.

Question 2.3. On veut écrire une fonction qui parcourt la liste et retourne la plus grande valeur de la liste. Cette fonction prend comme argument un pointeur sur le premier élément de la liste.

Le code C de la fonction est le suivant :

```
void pg(struct liste *pl) {
    unsigned int max = 0;
    while (pl != NULL) {
        if (pl->VAL > max)
            max = pl->VAL;
        pl = pl->NEXT;
    }
    return max;
}
```

On rappelle que la notation `pl->VAL` a la signification suivante : si `pl` est l'adresse de base de la structure, `pl->VAL` est la valeur dans le champ `VAL` de la structure.

Écrire cette fonction en assembleur MIPS R3000, en respectant les conventions d'utilisation de la pile. La variable locale `max` sera implantée dans un registre, et il n'est donc pas nécessaire de réserver de la place dans la pile pour cette variable. On écrira cette fonction en utilisant les registres suivants :
- \$9 : adresse de la structure ;
- \$10 : champ `VAL` de la structure ;

– $11 : registre de travail ;
– $2 : valeur de `max` (à retourner par la fonction)

```
pg:
#prologue
    addiu   $29,    $29,    -4      # seulement $31
    sw      $31,    0($29)

#corps
    xor     $2,     $2,     $2      # max = 0
    or      $9,     $4,     $0      # $9 = @structure

boucle:
    beq     $9,     $0,     fin     # Si pl == NULL, fin de la fonction
    lw      $10,    4($9)           # $10 = pl->VAL
    slt     $11,    $2,     $10     # Si max < pl->VAL alors $11 = 1
    beq     $11,    $0,     faux
    or      $2,     $10,    $0      # $ max = pl->VAL

faux:
    lw      $9,     0($9)           # pl = pl->NEXT
    j       boucle

fin:

#epilogue
    lw      $31,    0($29)
    addiu   $29,    $29,    4
    jr      $31
```

Question 2.4. Ecrire le programme `main()` qui appelle la fonction `pg()`, et imprime le résultat sur le terminal au moyen d'un appel système.

```
.text
    .globl main
#prologue
    addiu   $29,    $29,    -8      # $31 + 1 passage d'argument a la fonction pg
    sw      $31,    4($29)          # sauvegarde de $31

# Corps
    la      $4,     elm1            # $4 = @elm1
    jal     pg
# printf
    or      $4,     $2,     $0
    ori     $2,     $0,     1
    syscall

# Fonction terminale : inutile de retablir la pile
# exit
    ori     $2,     $0,     10
    syscall
```