

EXERCICE A : Langage D'assemblage MIPS (5 points)**Numéro :**

On considère le programme C suivant réalisant l'inversion des éléments d'un tableau d'entiers. Le tableau est statique et l'inversion se fait en place.

On s'intéresse à l'écriture de ce programme en langage d'assemblage MIPS. On adoptera la convention d'appel de fonction décrite dans le manuel « langage d'assemblage » *en optimisant le passage des arguments par registre*.

```
#include <stdio.h>
int tab[10] = {0,1,2,3,4,5,6,7,8,9};

void print_tab(int t[], int taille){
    int i;
    for(i=0;i<taille;i++)
        printf("%d ",t[i]);
    printf("\n");
}

void swap(int *t1, int *t2){
    int x = *t1;
    *t1 = *t2;
    *t2 = x;
}

void reverse(int i, int j, int t[]){
    for(;i<j;i++,j--)
        swap(&t[i],&t[j]);
}

void main(){

    reverse(0,9,tab);
    print_tab(tab,10);
}
```

A1) Complétez les instructions MIPS du programme principal. Il y a exactement le nombre de lignes nécessaires.

```
main:
    addiu $29, $29, -12
    _____
    _____
    jal _____
    _____
    _____
    addiu $29, $29, 12
    ori $2, $0, 10
    syscall
```

A2) Complétez le code assembleur MIPS de la fonction `reverse`.

```

reverse:
    # na = 2, nv = 0, nr = 3, $4:i, $5:j, $6:t

    addiu $29, $29, _____
    sw $31, _____ ($29)
    sw $16, _____ ($29)
    sw $17, _____ ($29)
    sw $18, _____ ($29)

    # recopie de i et j dans $16 et $17
    _____ $16, $4, $0
    _____ $17, $5, $0

loop: slt $18, $16, $17
    _____ $18, $0, _____
    #recuperation de &t[i] et &t[j] dans $4 et $5
    _____ $4, $16, _____
    addu $4, _____, $4
    _____ $5, $17, _____
    addu $5, _____, $5
    jal swap

    _____
    _____
    j _____
fin_reverse:
    lw $18, _____ ($29)
    lw $17, _____ ($29)
    lw $16, _____ ($29)
    lw $31, _____ ($29)
    addiu $29, $29, _____
    jr $31

```

A3) Écrivez l'intégralité du code de la fonction `swap`. Vous utiliserez impérativement les registres contenant les arguments ainsi que les registres auxiliaires \$16 et \$17 à l'exclusion de tout autre registre auxiliaire.

`swap:`

```

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

```

EXERCICE B : Caches de 1er niveau (5 points)**Numéro :**

Le but de cet exercice est d'analyser le remplissage d'un cache à correspondance directe, puis d'évaluer le nombre de cycles nécessaires à l'exécution d'une fonction C, en tenant compte des caractéristiques des caches et du placement des données en mémoire. La fonction réalise la somme pondérée des éléments d'un tableau.

```

int C[256];
int V[256];

int prodmat(int *coef, int *val, int nbval)
{
    int i, res = 0;
    for (i=0; i < nbval; i++) {
        res = res + (*coef) * (*val);
        coef++;
        val++;
    }
    return res;
}

A prodmat:
B    li    $3,0
C    li    $2,0
D    blez  $6,fin
E loop:
F    lw    $7,0($4)
G    lw    $8,0($5)
H    addiu $3,$3,1
I    mult  $8,$7
J    addiu $4,$4,4
K    addiu $5,$5,4
L    mflo  $7
M    addu  $2,$2,$7
N    bne   $3,$6,loop
O    nop
P fin:
Q    jr    $31
R    nop

```

On considère des caches L1 d'instructions et de données de type « write-through » à correspondance directe, d'une capacité totale de 256 octets chacun. La ligne de cache a une longueur de 16 octets pour le cache instructions et 16 octets pour le cache de données.

B1) Donnez le nombre de cases d'un cache (une case permet de stocker une ligne) et le nombre des bits respectifs des champs « offset », « index » et « étiquette » de l'adresse.

--	--

B2) On suppose que la fonction **prodmat** n'a encore jamais été exécutée. Son adresse est **0x10010018**. Placez toutes les instructions de la fonction **prodmat** dans les cases du cache d'instructions. Le tableau ci-dessous est un extrait du cache d'instructions, car celui-ci contient plus de 5 cases. Vous mettez donc en face de chaque case la valeur de l'index de la case occupée. Afin de simplifier le remplissage, vous mettez les lettres correspondantes aux instructions (par exemple **F** pour l'instruction « lw \$7, 0(\$4) ». Donnez le nombre total de miss instruction **nbmiss_inst**, pour charger le code de cette fonction dans le cache:

index	Offset C	Offset 8	Offset 4	Offset 0

B3) La fonction **prodmatt** est appliquée sur les deux variables globales C et V, lesquelles sont deux tableaux de 256 entiers. Les tableaux C et V se suivent dans la mémoire (V est après C) et le tableau C est à l'adresse 0x10010100. En supposant que les tableaux C et V n'ont encore jamais été lus, donnez, **en le justifiant**, le nombre de miss data :

- **nbmiss_data_1** : après **le premier tour** de boucle.
- **nbmiss_data_2** : après **le deuxième** tour de boucle.
- **nbmiss_data** : après **nbval** tours de boucle.

B4) La durée d'un tour de boucle de la fonction **prodmatt** est **d_tour**=10 cycles. Le coût d'un miss instruction ou d'un miss de donnée est **cout_miss**=10 cycles. On suppose que le processeur exécute 1 instruction par cycle. Vous négligerez les instructions qui précèdent et qui suivent la boucle. Donnez une expression littérale de la durée d'exécution en fonction de la durée d'un tour de boucle (**d_tour**), du nombre de miss total (**nbmiss_inst** et **nbmiss_data**), du coût du miss (**cout_miss**) et du nombre de tour (**nbval**). Calculez la durée estimée de l'exécution de la fonction avec les valeurs numériques du problème.

B5) Le tableau V est maintenant décalé de 16 octets. Autrement dit, la nouvelle adresse de V[0] est à l'ancienne adresse + 16. Recalculez le nombre de MISS de données après **nbval** tours de boucles : **nbmiss_data_new**. Recalculez la durée estimée de l'exécution de la fonction.

EXERCICE C : GIET (5 points)**Numéro :**

Dans le questionnaire à choix multiple suivant, chaque bonne réponse donne ½ point, chaque mauvaise réponse coûte ¼ de point. Il a qu'une réponse valide par question.

C1) Combien d'appels système différents sont traités par le GIET étudié en cours et TD ?

- 0x80000180
- 31
- 16
- 32

C2) Pour quelle raison la fonction de commutation de contexte est-elle toujours écrite en assembleur dans un système d'exploitation ?

- Parce qu'elle est spécifique au processeur qui fait tourner le système d'exploitation
- Parce qu'elle s'exécute en mode superviseur
- Parce qu'elle manipule les registres de l'ICU
- Parce qu'elle est spécifique au système d'exploitation et indépendante du processeur

C3) On suppose que le PC du MIPS32 contient 0x400200, et qu'à cette adresse se trouve une instruction add générant une exception de type dépassement de capacité (overflow). Juste après l'exécution de cette instruction add, quel est le contenu du registre EPC ?

- EPC contient 0x400204
- EPC contient 0x80000180
- EPC contient 0x400200
- EPC contient l'adresse _exc_handler

C4) Dans le GIET, quelle étiquette assembleur marque le début du code assembleur permettant de gérer spécifiquement les appels système ?

- _sys_vector
- _sys_handler
- _sys_enable
- _sys_demux

C5) Lorsque le processeur exécute plusieurs tâches en parallèle par multiplexage temporel, pourquoi le code de boot doit-il obligatoirement initialiser le registre \$31 de toutes les tâches (à part la première à s'exécuter) ?

- \$31 doit contenir l'adresse d'entrée du GIET.
- \$31 doit contenir l'adresse de retour au GIET.
- \$31 doit contenir une adresse de début de nouvelle tâche sélectionnée valide.
- Ce n'est pas du tout obligatoire, on peut laisser \$31 non initialisé.

C6) Comment un périphérique signale t'il qu'il a terminé le traitement qu'on lui a demandé?

- En générant une exception.
- En générant une interruption.
- Grâce à un appel système.
- C'est au logiciel applicatif en mode utilisateur de vérifier que le traitement est terminé.

C7) Lorsque le processeur exécute plusieurs tâches en pseudo-parallélisme, dans quel segment mémoire sont stockés les contextes des tâches ?

- kcode
- kunpdata
- kdata
- stack

C8) Parmi les actions réalisées par la fonction système `_fb_write()`, quel est celle qui active le composant DMA (lancement d'un transfert) pour la fonction?

- La lecture du registre protégé \$12
- L'écriture dans le registre `DMA_LEN`
- L'écriture dans le registre `DMA_SRC`
- L'écriture dans le registre `DMA_DST`

C9) A quoi servent les verrous associés aux périphériques IOC et DMA ?

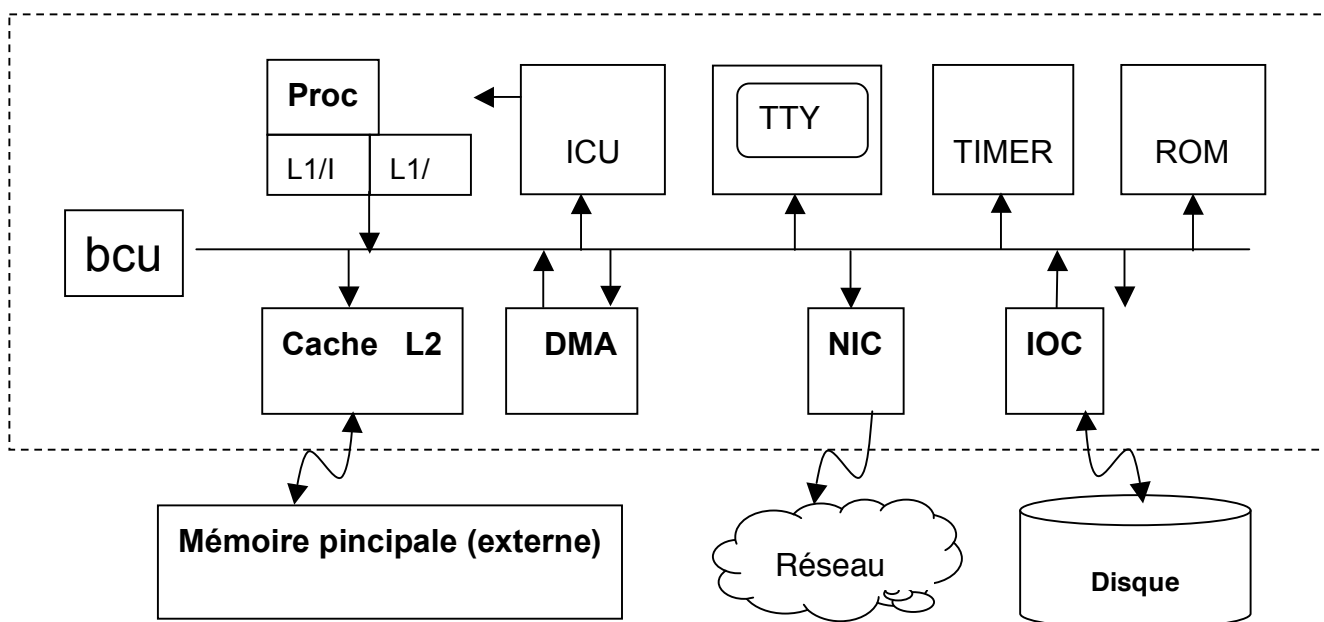
- A empêcher l'accès direct aux périphériques par le code utilisateur.
- A garantir que les exceptions sont bien prioritaires par rapport aux interruptions.
- A vérifier les droits d'accès à l'espace adressable.
- A permettre l'utilisation des périphériques par plusieurs tâches logicielles indépendantes.

C10) Dans le GIET, que représente le tableau `_exception_vector`?

- La table des pointeurs vers les fonctions gérant les différentes causes d'appel au GIET.
- La table des pointeurs vers les fonctions gérant les différents types d'exception.
- La table des pointeurs vers les fonctions gérant les interruptions.
- La table des pointeurs vers les fonctions gérant les appels système.

EXERCICE D : Bus système (5 points)**Numéro :**

On s'intéresse dans cet exercice à un serveur de « *Video on Demand* » dont l'architecture est décrite ci-dessous. Les vidéos sont stockées sur le disque et sont transmises sur le réseau par un contrôleur réseau (Network Interface Controller). Celui-ci se comporte comme une cible multi-canaux, contenant un ensemble de N tampons adressables $FIFO[i]$ dans lesquels le système doit écrire pour transmettre les flux vidéo. Les N tampons $FIFO[i]$ définissent N canaux indépendants, permettant de transmettre N vidéos simultanément. On cherche à évaluer le nombre maximal de vidéos qui peuvent être transmises simultanément, en faisant l'hypothèse que c'est la bande passante du bus système (et non la bande passante du contrôleur réseau) qui est le facteur limitant. Le bus système fonctionne à une fréquence de 100 MHz, et possède une largeur de 32 bits.



Pour chaque vidéo, le transfert se fait en deux temps :

- phase 1 : le contrôleur de disque transfère des blocs de 512 octets du disque vers N tampons intermédiaires $BUF[i]$ en mémoire (un tampon $BUF[i]$ par vidéo transférée).
- phase 2 : Pour chaque vidéo en cours de transfert, le contrôleur DMA lit le contenu du tampon $BUF[i]$ et le re-écrit dans le tampon $FIFO[i]$.

Les vidéos, compressées suivant le format MPEG, ont un encombrement de 600 Moctets, pour une durée de 1 heure quarante minutes. On rappelle que la fréquence vidéo est de 25 images par seconde. (une nouvelle image doit être affichée toutes les 40 ms).

D1) Qu'est-ce qu'un composant maître ? Combien y a-t-il de composants maîtres sur ce bus, et quels sont-ils ? Combien y a-t-il de composants cibles, et quels sont-ils ? Comment est désignée la cible d'une transaction ?

D2) Du côté serveur, de combien de cycles du bus dispose-t-on entre deux images consécutives d'une même vidéo ? Quelle est la bande passante maximale du bus (en nombre d'octets par cycle) ?

D3) Quel est le nombre total d'images d'une séquence vidéo de 1H 40' ? Quel est l'encombrement moyen d'une image (en nombre d'octets) ?

D4) Pour chaque image affichée, combien y a-t-il de transferts de cette image sur le bus du serveur ? Quel est le nombre de cycles minimal d'occupation du bus pour transférer une image complète depuis le disque jusqu'au contrôleur réseau ?

D5) Donnez en fonction des résultats précédents une borne pour le nombre maximal de vidéos qui peuvent être transmises simultanément par le serveur.