

# **Architecture Matérielle et Logicielle des Ordinateurs**

## **Examen de juin 2019**

### **Session 2**

L'examen comporte 4 parties A, B, C et D indépendantes notées sur 5 points chacune.

Les documents sont autorisés.

Vous devez rédiger vos réponses dans les cadres réservés pour chaque partie.  
(Aucune feuille volante ne sera acceptée)

**Commencez par recopier votre numéro d'anonymat sur chacune des 4 feuilles de ce document. (Les quatre parties sont corrigées séparément).**

Répondez aux questions posées en faisant des phrases complètes et compréhensibles (sujet, verbe, complément...)



**EXERCICE A : Assembleur (5 pts)****N° Anonymat :**

On considère la fonction **Fib** qui calcule la valeur du terme de rang **n** dans la suite de Fibonacci. Nous vous rappelons que le terme de rang **n** de la suite de Fibonacci est la somme des termes de rang **n-1** et **n-2** (on suppose que le terme de rang 0 est 0 et que le terme de rang 1 est 1).

Cette fonction peut s'écrire de manière récursive de la façon suivante :

```
unsigned int Fib (unsigned int n)
{
    if (n < 2)
        return (n) ;

    return (Fib (n-1) + Fib (n-2)) ;
}
```

Le code assembleur suivant (à compléter) a été obtenu par la compilation du code C de la fonction **Fib**. Ce code utilise le registre persistant \$16 pour sauvegarder la valeur de retour de l'appel **Fib (n-1)**

```
.text
.globl Fib
.func Fib

Fib :      /* Partie 1 à compléter dans la question A3 */      /* Prologue */
          ori      $2 , $4 , 0
          sltiu   $7 , $4 , 2
          bne     $7 , $0 , endif

          /* Partie 2 à compléter dans la question A4 */
          jal     Fib

          /* Partie 3 à compléter dans la question A5 */
          jal     Fib
          addiu   $2 , $2 , $16

endif :   /* Partie 4 à compléter dans la question A6 */      /* Epilogue */
          jr      $31

.endfunc
```

**A1)** (0.5 pt) Combien de places faut-il réserver sur la pile dans le prologue de cette fonction ? Justifiez votre réponse.

---

**A2)** (0.5 pt) Dites pourquoi le registre \$4 doit être sauvegardé sur la pile avant l'appel à **Fib**

**A3)** (1 pt) Complétez la Partie 1 manquante du code

**A4)** (1 pt) Complétez la Partie 2 manquante du code

**A5)** (1 pt) Complétez la Partie 3 manquante du code

**A6)** (1 pt) Complétez la Partie 4 manquante du code

**EXERCICE B : Mémoire cache (5 pts)****N° Anonymat :**

On considère un cache de données de premier niveau write-through à correspondance directe, d'une capacité totale de 8 Kio. La ligne de cache a une largeur de 32 octets (8 mots de 32 bits). Les adresses sont sur 32 bits.

Le but de l'exercice est d'analyser le remplissage d'un cache de données L1, puis d'estimer le nombre de cycles nécessaires à l'exécution d'un programme. On suppose que le cache de données est initialement vide.

On considère la fonction suivante :

```
int32_t X[N];
...
void shift(int32_t X[N]) {
    for (register int32_t i = N - 1; i > 0; i -= 1) {
        X[i] = X[i - 1];
    }
    X[0] = 0;
}
```

Dans tout l'exercice, on suppose que le tableau global X est implanté en mémoire à l'adresse 0x1001 7000, et qu'il est passé à la fonction shift, et que N vaut 1024 (#define N 1024).

**Rappels** : le mot clé "register" est une directive passée au compilateur pour qu'il place la variable *i* dans un registre plutôt que sur la pile ; la variable *i* reste donc en registre durant toute la durée d'exécution de la fonction et ni sa lecture ni son écriture ne provoquent d'accès au cache de données. Un int32\_t est codé sur 4 octets.

**B1)** (0.5 pt) Donner le nombre de bits des champs offset, index et étiquette d'une adresse.

offset :  
index :  
étiquette :

**B2)** (1 pt) Donner, pour les 3 premiers miss : la valeur de *i*, l'élément de X provoquant le miss (ex : X[12]), l'index de la case du cache correspondant, et l'offset du mot correspondant.

	<i>i</i>	Élément de X	Index	Offset
1er Miss				
2e Miss				
3e Miss				

**B3)** (1 pt) Représenter dans le schéma ci-dessous la ou les case(s) valide(s) du cache après 10 itérations, en précisant les index (seules 3 cases sont représentées), et en mettant dans les parties « Mot <x> » les éléments de X présents (exemple : X[12]).

---

Index	TAG	Mot 7	Mot 6	Mot 5	Mot 4	Mot 3	Mot 2	Mot 1	Mot 0
.									
.									
.									

**B4)** (0.5 pt) Calculer, en le justifiant, le nombre de miss de données rencontrés lors de l'exécution de cette fonction.

.

**B5)** (0.5 pt) Donner le taux de miss sur le cache de données pour cette fonction.

.

**B6)** (0.5 pt) Peut-on simplement réduire le nombre de miss rencontrés lors de l'exécution ? Si oui, comment ? Sinon, pourquoi ?

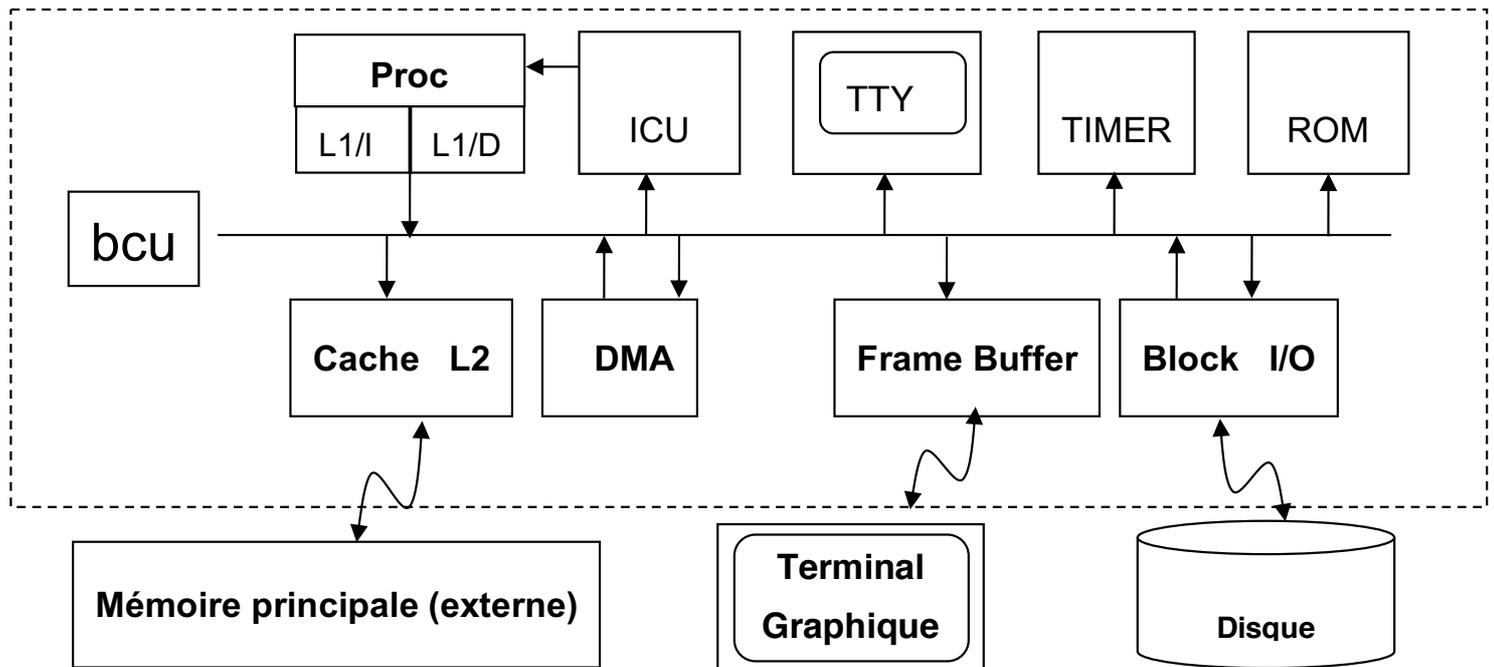
.

**B7)** (0.5 pt) La compilation de ce code a produit une boucle de 7 instructions pour la boucle (qui met donc 7 cycles à s'exécuter en présence d'un système mémoire parfait). On néglige l'effet des miss sur le cache d'instructions. Calculer le nombre de cycles total pour l'exécution de la fonction, et le nombre moyen de cycles par itération de la boucle si un miss de données coûte 18 cycles.

.

**EXERCICE C : Bus système (5 pts)****N° Anonymat :**

On considère l'architecture matérielle utilisée en TP. Cette architecture contient un seul processeur (avec ses caches de 1<sup>er</sup> niveau), un timer, un terminal écran/clavier de type TTY, un contrôleur graphique (« frame buffer ») permettant d'afficher des images, une ROM contenant le « code de boot », un cache de 2<sup>e</sup> niveau permettant d'accéder à la mémoire externe, et deux périphériques possédant une capacité d'adressage de la mémoire : le contrôleur DMA permet de transférer des données d'un tampon mémoire vers un autre. Le contrôleur I/O permet de transférer des données entre le disque et un tampon mémoire.



Des images sont stockées sur le disque, et on cherche à afficher une séquence d'image en respectant la cadence vidéo, qui est de 25 images par seconde (une nouvelle image doit être affichée toutes les 40 ms). L'affichage d'une image nécessite trois étapes :

- **Phase Load** : chargement de l'image depuis le disque vers un premier tampon mémoire appelé buf\_in. Ce transfert est réalisé par le contrôleur I/O.
- **Phase Modif** : le processeur lit l'image stockée dans buf\_in, la modifie, et recopie l'image modifiée dans un second tampon mémoire buf\_out.
- **Phase Display** : affichage de l'image par copie du tampon buf\_out vers la mémoire vidéo. Ces transferts (lecture puis écriture) sont réalisés par le DMA.

Le but général de l'exercice est de déterminer la taille maximale d'une image pour garantir la cadence d'une image toutes les 40ms, en faisant l'hypothèse que le facteur limitant est la bande passante du bus (mesurée en nombre d'octets par cycle). On suppose que l'image est codée en « niveaux de gris », et que chaque pixel est codé sur 8 bits. On notera N le nombre total de pixels d'une image : Par exemple, une image de 400 lignes de 600 pixels a une taille de  $N = 240\,000$  pixels.

On suppose que cette architecture est celle d'un système embarqué (tel qu'un téléphone mobile), qui fonctionne à 10 MHz. Cette fréquence est assez basse, pour limiter la

---

consommation d'énergie, et augmenter la durée de vie de la batterie. La largeur du bus est de 32 bits, ce qui signifie qu'on peut transférer sur le bus au plus un mot de 32 bits par cycle.

**C1)** (1.5 pts) 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 ?

**C2)** (1.5 pts) De combien de cycles dispose-t-on entre deux affichages d'image ? Quelle est la bande passante maximale du bus (en nombre d'octets par cycle)? Quel est le temps minimal nécessaire (mesuré en nombre de cycles) pour transférer une image de N pixels entre deux composants matériels ?

**C3)** (2 pts) Pour chaque image affichée, combien y a-t-il de transferts de cette image sur le bus ? En déduire la taille maximale d'une image, en faisant l'hypothèse (optimiste) que le bus ne sert qu'à transférer les pixels d'une image d'un composant vers un autre.

---

**Exercice D : GIET et mémoire virtuelle (5 pts) N° d'anonymat:**

**ATTENTION :** Chaque question peut avoir une ou plusieurs réponses.  
Cocher toutes les réponses justes donne 0,5 point.  
Chaque réponse fausse retire 0,25 point.  
L'absence de réponse vaut 0 point.

**D1) Le composant ICU de la plateforme de l'exercice C est connecté aux lignes d'interruption de la manière suivante : le TIMER est sur l'entrée irq\_in[8] de l'ICU, le TTY est sur l'entrée irq\_in[3], le DMA est sur l'entrée irq\_in[2] et l'IOC est sur l'entrée irq\_in[1]. Comment faut-il programmer le masque de l'ICU pour démasquer toutes les interruptions ?**

- 0x00000087
- 0x0000010E
- 8321
- 270

**D2) Si les lignes d'interruption du TTY et du TIMER de la question D1 se lèvent simultanément, quelle sera la valeur du registre ICU\_INDEX présent dans l'ICU**

- 3
- 0x108
- 8
- 264

**D3) Qu'est-ce que le vecteur d'interruption ?**

- C'est le numéro de la ligne d'interruption active.
- C'est l'état des lignes d'interruptions.
- C'est un tableau de pointeur de fonctions.
- C'est la structure qui définit quelles sont les interruptions à écouter.

**D4) Dites quelles instructions peuvent provoquer l'entrée dans le GIET**

- syscall
- lw \$2, (\$8)
- jr \$8
- ori \$2, \$3, \$4

**D5) Dans le GIET, les interruptions sont masquées pendant l'exécution des ISR (Interrupt Service Routine)**

- Pour éviter d'avoir à gérer une exception pendant l'exécution de l'ISR.
- Parce que les caches L1 sont de type write-through à correspondance directe.
- Parce que ça évite d'avoir à sauver le contexte du processeur (état de tous les registres).
- Parce que c'est plus simple à gérer et que la durée d'exécution d'une ISR est bornée et petite.

**D6) Le code assembleur ci-contre est l'entrée du GIET, dites quelles affirmations sont vraies.**

```
_giet:
    mfc0    $27 , $13
    la     $26 , _cause_vector
    andi   $27 , $27 , 0x3c
    addu   $26 , $26 , $27
    lw     $26 , ($26)
    jr     $26
```

- À l'entrée du GIET \$31 contient l'adresse de retour.
- \$26 et \$27 sont des registres privilégiés pour le kernel, leur valeur respective est sauvée automatiquement dans le coprocesseur 0 (là où se trouvent déjà les registres EPC, SR, CR, etc.)
- Le processeur n'est pas interruptible pendant l'exécution de cette séquence.
- Après l'exécution de la 1<sup>re</sup> instruction, le registre \$27 contient la cause d'entrée dans le GIET.

**D7) À quoi sert l'éditeur de lien ?**

- L'éditeur de lien place dans l'espace d'adressage les sections produites par le compilateur.
- C'est un éditeur de texte.
- C'est l'application qui produit l'exécutable.
- C'est l'autre nom du Makefile qui fait le lien entre les sources et les exécutables.

**D8) À propos de l'instruction eret, dites quelles affirmations sont vraies :**

- eret permet de sortir d'une fonction, c'est identique à jr \$31
- eret est une instruction privilégiée.
- eret modifie le registre de statut SR.
- eret est une macro-instruction qui réalise deux instructions.

**D9) À propos de la mémoire virtuelle, dites quelles affirmations sont vraies :**

- La mémoire virtuelle permet à plusieurs applications d'utiliser les mêmes adresses pour leur code et leurs données.
- Grâce à la mémoire virtuelle, il est possible d'augmenter le parallélisme réel d'exécution des programmes, pour éviter d'utiliser le temps partagé.
- La mémoire virtuelle permet d'empêcher une application d'accéder aux données d'une autre application.
- La mémoire virtuelle permet d'éviter l'usage du DMA pour le transfert de données entre un périphérique (device) et la mémoire physique.

**D10) À propos de la table des pages, dites quelles affirmations sont vraies :**

- Une table des pages permet de décrire le mapping de l'espace virtuel d'une application.
- L'acronyme (sigle) de la « Table des pages » est « TLB »
- Une table des pages est indexée par les numéros de pages physiques et contient les numéros de pages virtuelles.
- Une table des pages permet d'accélérer le calcul de traduction d'adresse.