EXERCICE A : Programmation en assembleur (5 points)

Numéro d'anonymat:

A1) Quelles sont les 3 grandes parties qui composent une fonction F écrite en assembleur ?

Soit le programme C suivant :

```
int fonca(int a)
{
     int la1,la2;

     la1=a+1;
     la2=a+2;
     return (la1+la2)/2;
}

int foncb(int b1, int b2)
{
     int lb1;

     lb1=b1*2 + b2;
     return lb1;
}

int foncc()
{
     int lc1, lc2, lc3, lc4, lc5;

     lc1=1; lc2=2; lc3=3;

     lc4=fonca(lc1);
     lc5=foncb(lc2,lc3);
     return 0;
}
```

Dans le tableau suivant, **que vous devez remplir**, **nv** représente le nombre de variables locales de la fonction considérée, **na** représente le nombre maximum d'arguments des fonctions appelées par la fonction considérée, **nr** représente le nombre de registres persistants à sauvegarder pour la fonction considérée, en incluant $31. La colonne « Première instruction du prologue » doit contenir la première instruction de la fonction, faisant partie du prologue.

A2) A partir du code C précédent, remplir le tableau suivant

	nv	**na**	**Nr**	**Première instruction du prologue**
fonca()				
foncb()				
foncc()				

A3) On s'intéresse à la fonction fonca(int a). On suppose que $4 contient la valeur de l'argument a, et que l'on utilise respectivement $5 et $6 pour les variables locales la1 et la2. Ecrire le code assembleur correspondant au corps de fonction de fonca(int a). La division entière par 2 sera signée et effectuée de manière optimisée. Le résultat sera mis dans $2. Le code sera commenté.

A4) Dessiner la pile une fois que la première instruction du prologue de foncc() a été effectuée. En particulier, préciser les zones de la pile où vont être stockés les registres persistants, les variables locales et enfin les arguments des fonctions appelées par foncc(),.

EXERCICE B : Caches de Ier niveau (5 points)

Numéro d'anonymat:

Le but de cet exercice est de mesurer le nombre de cycles nécessaires à l'exécution du programme C ci-dessous en tenant compte des effets de cache. Ce programme implémente une partie de l'algorithme de « tri fusion », en fusionnant deux tableaux préalablement triés (A et B) pour créer un seul tableau (C).

```
int A[1024], B[1024], C[1024];
int main () {
      register int i =0, j=0, k=0;
      while ((i < 1024) && (j<1024)) {
            register int var_a, var_b;
            var_a = (i < 1024) ? A[i] : var_a;
            var_b = (j < 1024) ? B[j] : var_b;
            if (var_a < var_b) {
                  C[k] = var_a;
                  i++;
            } else {
                  C[k] = var_b;
                  j++;
      }
      k++;
}
```

On considère un cache de données L1, suivant une stratégie de correspondance directe, d'une capacité totale de 4 ko. Chaque ligne de ce cache a une taille de 64 octets (16 mots de 32 bits). On suppose que le cache de données est initialement vide (tous ses bits de validité sont à 0).

On suppose que le tableau B est initialisé avec les nombres de 1 à 1024, le tableau A avec les nombres de 1025 à 2048. Exemple : A[0]=1025, A[1]=1026, B[0]=1, B[1]=2, ...

Les données du programme sont stockées de façon contigüe dans la mémoire, dans l'ordre de leur déclaration. Le premier élément A[0] est rangé à l'adresse 0x00001000.

Le mot clé « register », utilisé dans le programme C, est une directive passée au compilateur pour qu'il place les variables i, j, k, var_a et var_b dans des registres plutôt que sur la pile du programme. Ces variables sont donc contenues dans des registres durant toute la durée d'exécution du programme et leur lecture ne provoque pas d'accès au cache de données. Les adresses sont sur 32 bits et chaque adresse référence un octet en mémoire. On rappelle que 1 ko = 1024 octets = 2^{10} octets = 0x400 octets.

B1) Donnez les adresses de base en mémoire des deux tableaux B et C.

B2) Donnez le nombre de cases de ce cache (une case permet de stocker une ligne) et le nombre de bits respectifs des champs « offset », « index » et « étiquette » de l'adresse.

B3) Donnez l'état de ce cache de données après le premier tour de boucle en remplissant le tableau ci-dessous. Le champ « index » contient l'index de case du cache. Pour faciliter la compréhension, le champ « étiquette » contiendra l'adresse complète (sur 32 bits) du mot d'indice 0 de la ligne de cache correspondante. Le champ « data_i » contient le mot d'indice « i » de la ligne de cache. On utilisera la notation hexadécimale, précédée de 0x pour l'étiquette et les données.

index	valid	étiquette	data_15	data_14	...	data_1	data_0

B4) Donnez le nombre de MISS sur le cache de données pour les 16 premières itérations de la boucle ; pour la 17ième itération ; de la 18ième itération à la fin de l'exécution. Quel est le nombre total de MISS pour l'exécution complète de la boucle ?

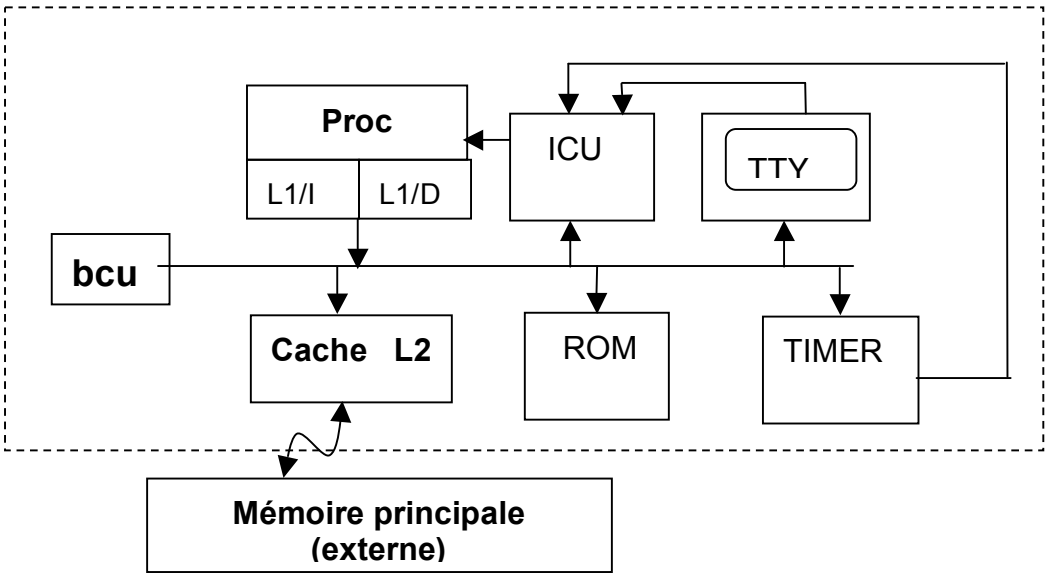
B5) La compilation de cette boucle génère une séquence de 11 instructions en assembleur MIPS 32 (on ne considère pas la déclaration des variables ni leur initialisation). Grâce à la technique de pipeline, le CPI (nombre de cycles par instruction) avec un système mémoire parfait est de 1 cycle par instruction. Le coût d'un MISS est de 25 cycles. On considère que le cache instruction se comporte comme un cache parfait (0 MISS). Le nombre total de MISS de données pour l'exécution complète de la boucle est NMISS (calculé à la question précédente).

- Donnez le temps total (en nombre de cycles horloge) nécessaire à l'exécution des 1024 itérations de la boucle.
- Quel est le CPI réel (indiquez deux chiffres après la virgule) ?

EXERCICE C : Mécanisme d'interruption (5points)

Numéro d'anonymat :

On considère l'architecture matérielle ALMO-MONO (utilisée en TP). Cette architecture contient un seul processeur (avec ses caches de 1er niveau), un timer, un terminal écran/clavier de type TTY, une ROM contenant le « code de boot », un cache de 2e niveau permettant d'accéder à la mémoire externe, et un concentrateur d'interruptions ICU.



Le processeur exécute un programme dont le but est d'afficher sur l'écran TTY les caractères saisis au clavier. L'algorithme de ce programme est résumé ci-dessous :

```
char      to_print ;
while (1)
{
    if (tty_get_irq(& to_print))    tty_printf ("\t%c\n", to_print);
}
```

Ce programme utilise le mécanisme d'interruption, plutôt qu'une attente active : le terminal envoie une interruption lorsqu'un caractère est saisi au clavier.

C1) Que doit on faire dans la phase d'initialisation du système (code de boot) pour que le mécanisme d'interruption fonctionne correctement ? On ne demande aucune valeur numérique, seulement les principes et les noms de variables/registres/structures utilisés.

C2) Que fait l'appel système tty_get_irq() ? On décrira précisément la suite des appels de fonctions déclenchée par l'exécution de cet appel système, en précisant ce que fait chaque fonction, jusqu'au retour de l'appel système.

C3) L'utilisateur appuie sur une touche du clavier, ce qui active l'interruption provenant du contrôleur TTY. Décrivez précisément ce qui se passe dans la machine entre l'activation de l'interruption et la reprise d'exécution du programme interrompu.

Exercice D : Code Système (5 points)

Numéro d'anonymat:

Cochez une seule réponse pour chaque question :
réponse juste 0,5 point ; réponse fausse -0,25 point ; pas de réponse 0 point.

D1) Le composant icu de la plateforme de l'exercice C est connecté aux lignes d'interruption de la manière suivante : le tty est sur l'entrée irq_in[1] de l'ICU, le timer sur l'entrée irq_in[0], comment faut-il programmer le masque de l'ICU pour démasquer les deux interruptions?

☐ 0x00000011

☐ 0x00000000000000000000000000000011

☐ 0x00000033

☐ 3

D2) Un « appel système » est

☐ Une instruction privilégiée exécutable en mode système pour accéder à certains registres.

☐ Une fonction exécutée en mode utilisateur qui provoque l'exécution de code système.

☐ Une fonction qui s'exécute en mode système.

☐ Un type particulier d'interruption géré par le GIET.

D3) Laquelle de ces actions ne nécessite pas que le processeur soit en mode superviseur

☐ exécuter l'instruction assembleur syscall

☐ exécuter l'instruction assembleur eret

☐ masquer les interruptions

☐ accéder à l'adresse 0x80000000

D4) Qu'est-ce que le taux de MISS d'un cache ?

☐ C'est l'accélération observée d'un programme grâce à son cache.

☐ C'est le rapport entre le nombre de MISS et le nombre de transactions sur le bus.

☐ C'est le rapport entre le nombre de requêtes qui font MISS et le nombre total de requêtes.

☐ C'est le rapport entre le nombre de MISS et le nombre de cycles exécutés.

D5) Dans le GIET, les interruptions sont masquées pendant l'exécution des ISR car :

☐ Une ISR est par principe in-interruptible.

☐ Il ne faut pas perdre la cohérence de la pile.

☐ Le code système est par principe in-interruptible.

☐ Les ISRs utilisées sont très courtes, donc on peut se le permettre.

D6) Registres privilégiés

☐ HI / LO sont des registres privilégiés.

☐ $26 et $27 sont des registres privilégiés.

☐ $29 (le pointeur de pile) est un registre privilégié.

☐ Ni HI, ni LO, ni $26, ni $27, ni $29 ne sont privilégiés.

D7) Parallélisme de tâche par multiplexage temporel

☐ On ne parle de parallélisme de tâches que sur les machines multiprocesseurs.

☐ Les tâches partagent la même pile d'exécution.

☐ Les taches utilisent chacune leur propre pile d'exécution.

☐ Sur une machine multiprocesseur, la communication entre tâches utilise le DMA.

D8) A quoi sert la table des pages dans le mécanisme de mémoire virtuelle ?

☐ Connaître à l'avance la quantité de mémoire utilisée par un processus.

☐ Retrouver le numéro de page physique connaissant le numéro de page virtuelle.

☐ Retrouver le numéro de page virtuelle connaissant le numéro de page physique.

☐ Accélérer le calcul de traduction d'adresse.

D9) A quoi sert le fichier ldscript ?

☐ C'est une description de l'ensemble des outils utilisés dans la chaîne de compilation croisée.

☐ C'est un fichier qui définit les adresses de base des différents segments.

☐ C'est un fichier qui contient la définition des tables de page.

☐ C'est un script qui appelle l'éditeur de lien ld.

D10) à propos de l'instruction mfc0

☐ Elle permet de lire le contenu des registres protégés, lorsque le processeur est en mode user

☐ Comme mfhi et mflo, elle permet de récupérer le contenu du registre c0.

☐ C'est l'instruction de masquage Booléen des registres systèmes.

☐ C'est une instruction privilégiée pour lire le contenu des registres systèmes.