EXERCICE A : Programmation assembleur (5 points) **Numéro d'anonymat :**

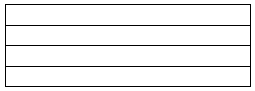
On se propose d'écrire une fonction qui prend un tableau "tab" de "size" entiers en argument et qui trouve l'entier qui a le plus grand nombre de bits à 1. Par exemple si tab est défini par :
`Int tab[3] = {0, 4, 7};`
Le nombre de bits à 1 de l'entier 0, c'est 0, le nombre de bit à 1 de l'entier 4, c'est 1, le nombre de bits à 1 de l'entier 7, c'est 3. Le nombre qui a le plus grand nombre de bits à 1 est donc 7.

```
1  int nbbits (int n) {
2      int i, res;
3      res = 0;
4      for( i = 32; i != 0; i--) {
5            res = res + (n & 1);
6            n = n >> 1;
7      }
8      return res;
9  }
10 int val_nbbits (int * tab, int size)
{
11     int max, val, i;
12     val = tab[0];
13     max = nbbits(tab[0]);
14     for(i = 1; i != size; i++) {
15           if (max < nbbits(tab[i])) {
16                 max = nbbits(tab[i]);
17                 val = tab[i];
18           }
19     }
20     return val;
21 }
```

```
22 int tab[3] = {14, 29, 1027};
23 main() {
24     int val;
25     val = val_nbbits(tab, 3);
26     printf("%d", max);
27     return 0;
28 }
```

A1) Écrire en assembleur l'allocation du tableau "`tab`" (ligne 22) dans la section des données globales.

A2) Représenter l'état de la pile avant l'exécution de la fonction "`val_nbbits`" (dans la fonction main). Indiquez clairement ce que pointe $29 (les adresses petites sont en bas, les adresses grandes sont en haut). Justifiez votre réponse.



A3) En supposant que, dans la fonction **`val_nbbits`**, les variables `tab, size, max, val` et `i` utilisent respectivement les registres `$16, $17, $18, $19` et `$20`. Le code de la fonction utilisera les registres $8, $9 et $10 pour les calculs intermédiaires. Combien de place (en octets) doit-on réserver dans la pile pour le contexte de la fonction val_nbbits. Justifiez votre réponse.

A4) écrivez le prologue de la fonction `val_nbbits()`

A5) Écrivez le code des lignes 12 et 13. On suppose que les registres `$16, $17, $18, $19` et `$20` n'ont pas encore été initialisés.

A6) La fonction `nbbits()` est une fonction terminale. En supposant que les variables `i` et `res` utilisent respectivement les registres `$8` et `$9`, écrivez le code de la fonction `nbbits()`.

Exercice B : Mémoires Cache **Numéro :**

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 nécessaire à l'exécution d'une fonction. On suppose que le cache de données est initialement vide.
On considère la fonction suivante :

```
int32_t X[N];
int32_t Y[N];
...
void inv_scan(int32_t X[N], int32_t Y[N]) {
    Y[N - 1] = X[N - 1];
    for (register int32_t i = N - 2; i >= 0; i -= 1) {
        Y[i] = X[i] + Y[i + 1];
    }
}
```

Dans tout l'exercice, on suppose que les tableaux globaux X et Y sont implantés en mémoire respectivement aux adresses 0x1001 0000 et 0x1001 1000, que ces tableaux sont passés à la fonction inv_scan, et que N vaut 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 point). Donner le nombre de bits des champs offset, index et étiquette d'une adresse.

B2 (1 point). Quels sont les éléments de X qui peuvent occuper les mots du cache suivants : case d'index 51, mot 7 ; et case d'index 128, mot 0 ? (une case contient une ligne)

B3 (1 point). Représenter dans le schéma ci-dessous la ou les case(s) valide(s) du cache après 1 itération, en précisant les index (seules 3 cases sont représentées), et en mettant dans les parties « Mot <x> » les éléments de X ou Y présents (exemple : X[12] ou Y[7]).
Note : dans le code assembleur produit, la lecture de X[i] a lieu avant celle de Y[i + 1].

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

B4 (1 point). Calculer, en le justifiant, le nombre de miss de données rencontrés lors de l'exécution de cette fonction.

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

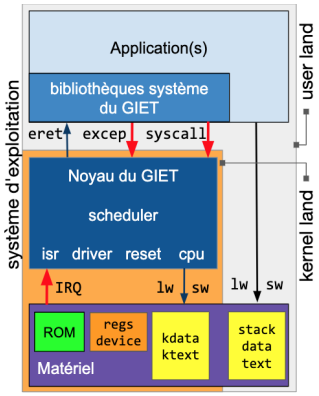
B6 (0.5 point). Peut-on simplement réduire le nombre de miss rencontrés lors de l'exécution ? Si oui, comment ? Si non, pourquoi ?

B7 (0.5 point). La compilation de ce code a produit une boucle de 11 instructions (qui met 11 cycles à s'exécuter en présence d'un système mémoire parfait), et des instructions avant la boucle qui mettent 4 cycles à s'exécuter. On néglige l'effet des miss sur le cache d'instructions. Calculer le nombre de cycles total pour l'exécution de la fonction si un miss de données coute 20 cycles (hors prologue et épilogue).

EXERCICE C : GIET (5 points) **Numéro d'anonymat :**

C1) (1 point) Le système d'exploitation GIET est composé d'une bibliothèque système et d'un noyau.
Que trouve-t-on dans les bibliothèques système ? Donnez deux exemples.

-



C2) (1 point) Le noyau gère 3 types d'événements : les appels système, les interruptions et les exceptions. Donnez deux exemples de chaque type de demande. De quoi est composé le vecteur de syscall ?

C3) (0.5 point) Le code du noyau s'exécute dans un mode d'exécution privilégié du processeur : le mode *kernel*. Que permet l'exécution en mode *kernel* qui est impossible en mode *user* (non-*kernel*) ? Est-ce que l'application peut accéder la mémoire sans passer par le noyau ?

C4) (1 point) Le scheduler (ordonnanceur) est le service du noyau en charge des changements de contexte des tâches. Dans le cas du GIET, quand ce service est-il appelé ? Que signifie changement de contexte des tâches ?

C5) (1 point) Sur une machine avec deux cœurs de calcul, chaque cœur dispose de deux caches (données et d'instructions) ayant une stratégie *write-through* sans cohérence. Il y a une perte de cohérence de cache lorsqu'un cache contient une ligne dont le contenu n'est plus valide. Expliquez en détails comment cela peut arriver.

C6) (0.5 point) Si on veut que deux tâches *user* partagent un même tableau en mémoire, les deux peuvent lire et écrire dans le tableau pour s'échanger des données. Puisque les caches L1 ne garantissent pas la cohérence matérielle de leur contenu, comment éviter le problème de cohérence ?

.

EXERCICE D Accès aux périphériques (5 points) **Numéro :**

0n s'intéresse dans cette partie aux opérations d'entrée/sortie permettant à un programme utilisateur d'accéder aux périphériques, tels qu'un terminal écran/clavier, un contrôleur réseau, ou un dispositif de stockage externe (disque magnétique ou clé USB).

D1) (1 point) Dans le cas du GIET, quels sont les trois arguments qu'un programme utilisateur doit transmettre au système d'exploitation lorsqu'il effectue un appel système pour demander un accès au disque en lecture ou en écriture ?

D2) (1 point) Expliquez pourquoi un programme utilisateur ne peut pas directement envoyer une commande d'entrée ou de sortie à un périphérique, sans passer par un appel système. Quelles sont les deux vérifications que doit effectuer le système d'exploitation avant de démarrer une opération d'entrée sortie de lecture ou d'écriture vers un fichier sur disque.

D3) (1 point) Comment le système d'exploitation déclenche-t-il une opération d'entrée/sortie sur un périphérique particulier ? Pour un type de périphérique donné, il existe évidemment de très nombreux fabricants. Deux contrôleurs de disque fournissent les mêmes services, mais utilisent des commandes différentes . Comment un système d'exploitation généraliste comme LINUX s'adapte-t-il à cette grande variété de périphériques matériels ?

D4) (1 point) Une opération d'entrée/sortie peut avoir une durée d'exécution très variable (quelques milliers à quelques millions de cycles), suivant le type et l'état courant du périphérique. L'exécution du programme demandeur est donc généralement suspendue par le système d'exploitation, en attendant la fin du transfert demandé, pour ne pas gaspiller inutilement du temps processeur. Comment le périphérique signale-t-il à l'OS qu'il a terminé le transfert ? Comment le programme utilisateur demandeur est-il re-activé ?

D5) (1 point) Puisque chaque périphérique matériel possède sa propre ligne d'interruption, le nombre de lignes d'interruption sortant des périphériques est beaucoup plus grand que le nombre d'interruptions entrant dans le processeur (certains processeurs n'ont qu'une seule ligne entrante). Lorsqu'un processeur est interrompu, et se branche au gestionnaire d'interruptions de l'OS, comment celui détermine-t-il quel périphérique est la source de l'interruption, et quelle ISR doit être exécutée ?