EXERCICE A : Programmation en assembleur (5 points) Numéro :

Dans cet exercice, on veut écrire l'algorithme de tri par sélection en langage d'assembleur du processeur MIPS 32. L'algorithme est codé en C dans la fonction sort() ci-dessous. Attention, l'algorithme itératif proposé ici est différent de l'algorithme récursif présenté en TD.

Le premier argument (a) de la fonction sort() est un pointeur vers le tableau d'entiers à trier. Le deuxième argument (size) est la taille du tableau.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 int a[] = {8, 6, 10, 3, 1, 2, 5, 4};
 4 int n = 8;
 5 void swap(int *a, int *b)
 6 {
 7      int temp;
 8      temp = *a;
 9      *a = *b;
10      *b = temp;
11 }
12 void sort(int a[], int size)
13 {
14      int i, m, max;
15      while (size > 1) {
16           m = 0;
17           max = a[m];
18           for (i = 1; i < size; i++) {
19                if (a[i] > max) {
20                     m = i;
21                     max = a[m];
22                }
23           }
24           if (m != size)
25                swap(&(a[m]), &(a[size - 1]));
26           size = size - 1;
27      }
28 }
29 int main()
30 {
31      int i;
32      sort(a, n);
33      for (i = 0; i < n; i++)
34           printf("%d ", a[i]);
35      exit(0);
36 }
```

A1) A l'entrée de la fonction swap, sur quoi pointe le pointeur de pile contenu dans $29 ?

Le registre $29 pointe dans la pile sur une case qui a été réservée pour contenir le premier paramètre de la fonction swap, c-a-d le paramètre a.

A2) Ecrire la fonction swap en langage assembleur. On supposera que l'argument a est dans le registre $4 et l'argument b dans le registre $5. On utilisera deux registres de travail $13 et $14. Attention : cette fonction swap ne possède pas les mêmes arguments que la fonction swap vue en TP !

```
swap:
      # prologue
      add $29, $29, -4          # optionnel
      # corps
      lw $13, 0($4)
      lw $14, 0($5)
      sw $14, 0($4)
      sw $13, 0($5)
      # epilogue
      add $29,$29,4             # optionnel si le pointeur de pile n'a pas été modifié
      jr  $31
```

A3) En supposant l'association variable – registre suivante : (m , $8) (i , $9) (a , $10) (size , $11) écrivez le code correspondant aux lignes 24 à 26

```
# ligne24 : if (m != size)
      beq     $8, $11, ligne26

#ligne25 : swap( &(a[m]) , &(a[size-1]))
      sll     $4, $8 2
      addu    $4, $10, $4
      addiu   $5, $11, -1
      sll     $5, $5, 2
      addu    $5, $10, $5
      jal     swap

ligne26 :  # size = size – 1
      addiu   $11, $11, -1
```

EXERCICE B : Caches de Ier niveau (5 points) Numéro :

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.

```
int A[64],  B[64], C[64];
int main(){
  register int i, sum = 0;
  for (i=0; i<64; i++){
    C[i] = A[i] + B[i];
    sum = sum + C[i];
  }
  return 0;
}
```

On considère un cache de données L1, à correspondance directe, d'une capacité totale de 1Ko. Chaque ligne de ce cache a une taille de 64 octets (16 mots de 32 bits).

Les tableaux sont initialisés selon les formules suivantes :

A[i] = i exemple A[10] = 10

B[i] = 2 * i exemple B[12] = 24

C[i] = i+1 exemple C[20] = 21

On suppose que le cache de données est initialement vide (tous ses bits de validité sont à 0)

Les données du programme sont stockées de façon contiguë dans la mémoire, dans l'ordre de leur déclaration. Le premier élément A[0] du tableau A est à l'adresse 0x1000.

Le mot clé "register" est une directive passée au compilateur pour qu'il place les variables i et sur dans des registres plutôt que sur la pile du programme. Les variables i et sum sont dans des registres durant toute la durée d'exécution du programme et ni leur lecture ni leur écriture ne provoquent d'accès au cache de données.

Les adresses sont sur 32 bits, et chaque adresse référence un octet en mémoire.

On rappelle que 1Ko = 1024 octets = 2^{10} octets = 0x400 octets.

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

adresse A= 0x1000
adresse B = 0x1100
adresse C = 0x1200

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

-le nombre de cases 16 = 1024 / 64
-la taille du champ offset 6 = log2 (64)
-la taille du champ index 4 = log2 (16)
-la taille du champ étiquette 22 = 32 – 6 - 4

B3) Donner 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 la 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 concernée. Le champ data i contient le mot d'index i de la ligne de cache. On utilisera la notation hexadécimale, précédée de 0x pour les adresses et pour les données.

index	Valid	Etiquette	data 15	data 14	....	data 1	data 0		
0	1	0x1000		0x0F	0x0E			0x01	0x00
4	1	0x1100		0x1E	0x1C			0x02	0x00
8	1	0x1200		0x10	0x0F			0x02	0x00

A chaque itération, il faut aller lire trois entiers en mémoire : A[i], B[i], C[i], et donc 3 lignes de cache.

La ligne de cache contenant A[0] est rangée dans la case d'index 0

La ligne de cache contenant B[0] est rangée dans la case d'index 4

La ligne de cache contenant C[0] est rangée dans la case d'index 8, (et C[0] est modifié)

B4) Donner le nombre de MISS sur le cache données pour les 3 premières itérations de la boucle. Quel est le nombre total de MISS pour l'exécution complète de la boucle ?

On a 3 MISS pour la première itération. On a 0 MISS pour les 2 suivantes.

Comme on a 64 itérations, et qu'une ligne de cache contient 16 entiers, il faudra charger au total 4 lignes de cache pour chaque tableau. On aura au total 4*3 = 12 MISS

B5) La compilation de cette boucle génère une séquence de 20 instructions en assembleur MIPS 32 (on ne considère pas la déclaration des variables ni leur initialisation). Grâce à la technique de pipe-line, le CPI (nombre de cylce d'avec un système mémoire parfait est de 1 cycle par instruction. Le coût d'un MISS est de 10 cycles. On considère que le cache instruction se comporte comme un cache parfait (0 MISS). Donner le temps total (en nombre de cycles horloge) nécessaire à l'exécution des 64 itérations de la boucle. Quel est le CPI réel ?

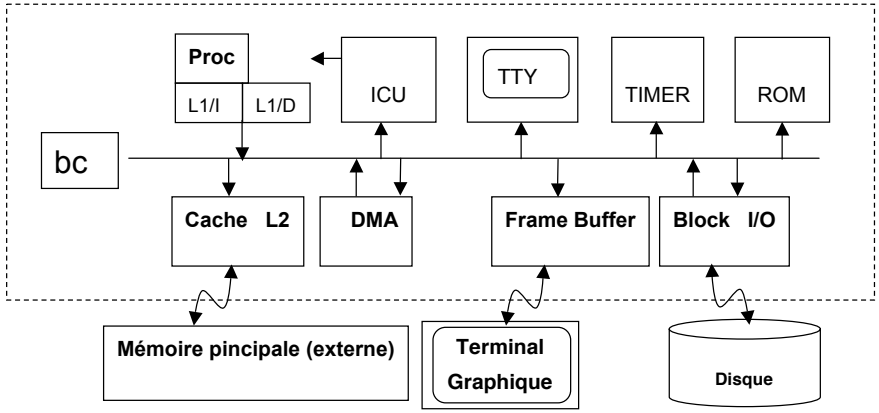
Il faut exécuter 20 * 64 instructions = 1280 cycles.

Comme on a 12 MISS, on a un surcoût de 120 cycles.

Le CPI effectif est de (1280 + 120) / 1280 = 1.09 cycle / instruction

EXERCICE C : Bus système (5 points) Numéro :

On considère l'architecture matérielle ALMO5 (utilisée en TP). Cette architecture contient un seul processeur (avec ses caches de 1er 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 2e 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.



On suppose que des images sont stockées sur le disque, et qu'on cherche à afficher une séquence d'image en respectant la cadence vidéo. La fréquence vidéo est de 25 images par seconde (une nouvelle image doit être affichée toutes les 40 ms). Comme dans le TP18, 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, 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 25MHz. 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 au plus un mot de 32 bits par cycle.

C1) 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 ?

Un composant maître est un composant capable de démarrer une transaction en émettant une adresse vers une cible. La cible est désignée par les bits de poids fort de l'adresse, qui sont analysés (décodés) par le composant BCU. Dans cette architecture, il y a trois maîtres (processeur, contrôleur DMA, contrôleur I/O). Il y a 8 cibles (ROM, Cache L2, ICU, TTY, Timer, Frame Buffer, DMA, Contrôleur I/O)

C2) 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 ?

On a 25 millions de cycles par seconde. On a 25 images par seconde. On a donc 1 million de cycles entre deux images. Un mot de 32 bits contient 4 octets, et on peut transférer au plus un mot par cycle. La bande passante maximale est donc de 4 octets par cycle.

Un pixel représente un octet. Si on transfère 4 octets par cycle, il faut donc N/4 cycles

C3) 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.

On a 5 transferts :

- du contrôleur I/O vers le tampon buf_in en mémoire (c'est à dire vers le cache L2)
- du tampon buf_in vers le processeur (en pratique du cache L2 vers le cache L1)
- du processeur vers le tampon buf_out (en pratique du cache L1 vers le cache L2)
- du tampon buf_out (en pratique le cache L2) vers le contrôleur DMA
- du contrôleur DMA vers le Frame Buffer

Il faut donc tranférer 5 fois N octets, ce qui nécessite au moins 5*N/4 cycles.Comme on dispose de 1 millions de cycles, on obtient N = 4/5 M = 800 000 pixels.

Ceci est une borne maximale qui n'a aucune chance d'être atteinte : d'une part le bus est utilisé pour d'autres types de trafic (MISS sur le cache instruction, configuration des composants DMA et I/O contrôleur). D'autre part, la bande passante du bus ne peut être utilisée à 100%, car il y a systématiquement un cycle perdu entre deux transactions.

Exercice D : Interruptions et Mémoire virtuelle (5 points) Numéro :

Cochez une seule réponse pour chaque question

D1) On considère la plateforme matérielle de l'exercice 3. Combien faut-il connecter d'entrées d'interruption au contrôleur d'interruptions (ICU) ?

☐ 3

=> ☐ 4

☐ 5

☐ 8

D2) Pour une certaine application logicielle, on ne souhaite utiliser que deux types d'interruptions : l'interruption du contrôleur I/O est connectée sur l'entrée 6 de ICU, et l' interruption du terminal écran/clavier est connectée sur l'entrée 2. Quelle valeur faut-il écrire dans le registre de masque de l'ICU pour n'autoriser que ces 2 interruptions ?

☐ 0x01000100

=> ☐ 0x00000044

☐ 0x00000062

☐ 0xFFFFFFFF

D3) Une tâche s'exécutant sur le processeur est interrompue par une interruption. Donner la succession des événements qui se produisent dans le système (l'ordre est important).

☐ Acquittement interruption, Sauvegarde par le processeur du compteur ordinal dans EPC, Saut du processeur à l'adresse int_handler, Saut du processeur à l'adresse 0x80000180

☐ Saut du processeur à l'adresse 0x80000180, Sauvegarde par le processeur du compteur ordinal dans EPC, Saut du processeur à l'adresse int_handler, Acquittement interruption

☐ Saut du processeur à l'adresse int_handler, Acquittement interruption, Sauvegarde par le processeur du compteur ordinal dans EPC, Saut du processeur à l'adresse 0x80000180

=> ☐ Sauvegarde par le processeur du compteur ordinal dans EPC, Saut du processeur à l'adresse 0x80000180, Saut du processeur à l'adresse int_handler, Acquittement interruption

D4) Que fait l'instruction eret du processeur MIPS32?

☐ elle branche à l'adresse contenue dans le registre $31 et interdit les interruptions.

☐ elle branche à l'adresse contenue dans le registre $31 et autorise les interruptions.

☐ elle branche à l'adresse contenue dans le registre EPC et interdit les interruptions.

=> ☐ elle branche à l'adresse contenue dans le registre EPC et autorise les interruptions.

D5) Pourquoi les registres $26 et $27 du processeur ne doivent pas être utilisés par un programme utilisateur?

☐ Ces registres sont protégés. Un accès illégal à ces registres en mode utilisateur déclenche un départ en exception.

☐ Ces registres sont réservés au système d'exploitation et contiennent des informations importantes du système d'exploitation qui ne doivent pas être connues des utilisateurs.

=> ☐ Ces registres sont utilisés par le gestionnaire d'interruption, exceptions et trappes, et peuvent donc être modifiés à tout moment en cas d'interruption.

D6) Dans un système de mémoire virtuelle paginée, la MMU (memory management Unit) utilise des tables de pages indexées par le numéro de page virtuelle pour définir la correspondance entre les adresses virtuelles et les adresses physiques.

☐ La table de page est construite par le système d'exploitation et cette unique table est utilisée par tous les programmes des utilisateurs.

☐ Chaque programme utilisateur construit et utilise sa propre table de page.

=> ☐ La table de page d'un programme utilisateur est construite par le système d'exploitation, et il y a autant de tables de pages que de programmes utilisateurs.

D7) Pourquoi les tables de pages ont-elles souvent une organisation à plusieurs niveaux ?

☐ Une table de page à 2 niveaux se parcourt plus rapidement qu'une table de pages à 1 niveau et occupe moins de place en mémoire qu'une table de page à 1 niveau.

☐ Une table de pages à 2 niveaux se parcourt plus rapidement qu'une table de pages à 1 niveau mais occupe plus de place en mémoire qu'une table de page à 1 niveau.

=> ☐ Une table de page à 2 niveaux se parcourt moins rapidement qu'une table de pages à 1 niveau mais occupe moins de place en mémoire qu'une table de page à 1 niveau.

D8) La TLB (Translation Look-aside Buffer) est un petit cache contenant les plus récentes traductions (VPN ⇔ PPN). Que se passe-t-il (dans le cas général) en cas de MISS sur la TLB ?

☐ Le processeur est gelé pendant 10 cycles en attendant que la MMU aille rechercher la traduction (VPN⇔PPN) manquante en mémoire, puis le programme reprend son exécution.

=> ☐ Le processeur est gelé pendant une centaine de cycles, car la MMU doit faire plusieurs accès à la mémoire pour parcourir les tables de pages, puis le programme reprend son exécution.

☐ Le programme qui a fait un MISS TLB est systématiquement interrompu, et le système d'exploitation attribue le processeur à un autre programme, car un défaut de page nécessite un accès au disque, qui coûte plusieurs centaines de milliers de cycles.

D9) Lors d'une commutation de tâches dans un système possédant un mécanisme de mémoire virtuelle,

☐ le système ne s'occupe pas de la MMU

☐ le système doit modifier le registre PTPR contenant l'adresse de base de la table des pages

☐ le système se contente d'invalider les entrées « utilisateur » de la TLB

=> ☐ le système modifie le registre PTPR et invalide les entrées « utilisateur » de la TLB

D10) Le fichier ldscript (étudié en TD et TP) contient les directives de regroupement des différentes sections dans différents segments, et définit les adresses de base de ces segments mémoire.

☐ Il est utilisé par l'éditeur de liens et définit des adresses physiques.

=> ☐ Il est utilisé par l'éditeur de liens et définit des adresses virtuelles.

☐ Il est utilisé pour générer les fichiers .o et définit des adresses physiques.

☐ Il est utilisé pour générer les fichiers .o et définit des adresses virtuelles.