

Numéro d'anonymat (donné sur votre étiquette)

--

**Examen Substitution 2021 – 2022**  
**Architecture des ordinateurs 1 – LU3IN029**  
**Durée : 1h30**

**Documents autorisés :** Aucun document ni machine électronique n'est autorisé à l'exception du mémento MIPS.

**Répondre directement sur le sujet. Ne pas désagrafer les feuilles.**

Le barème indiqué pour chaque question n'est donné qu'à titre indicatif tout comme le barème total qui sera ramené à une note sur 20 points. Le poids relatif des exercices et des questions par contre ne changera pas.

Le sujet comporte 18 pages et est composé de 3 exercices indépendants.

- Exercice 1 - 7 points : Rotation d'un mot binaire – (p. 2)
- Exercice 2 - 23 points : Fonction récursive : fréquence des chiffres – (p. 5)
- Exercice 3 - 15 points : Architecture et programmation système – (p. 13)

## Exercice 1 : Rotation d'un mot binaire – 7 points

On souhaite implémenter un circuit qui, étant donné un mot binaire  $A = a_3a_2a_1a_0$  de 4 bits et une valeur  $v$  comprise entre 0 et 3 encodée sur deux bits  $i_1i_0$ , réalise la rotation à gauche de  $v$  bits de  $A$ . Ce circuit produit le mot binaire  $S = s_3s_2s_1s_0$ .

La rotation à gauche de 0 bit produit le mot d'entrée en sortie.

La rotation à gauche de 1 bit décale les bits vers la gauche de 1 position et met le bit de poids fort  $a_3$  sur le bit de poids faible de  $S$ , soit  $s_0$ .

La rotation à gauche de 2 bits décale les bits vers la gauche de 2 positions et met respectivement  $a_3$  et  $a_2$ , les deux bits de poids forts de  $A$ , sur  $s_1$  et  $s_0$ , les deux bits de poids faible de  $S$ .

### Question 1.1 : 3 points

Remplir la table de vérité ci-dessous afin de donner, pour chacun des bits  $s_i$  de la sortie  $S$ , sa valeur en fonction de la valeur de  $i_1$  et  $i_0$ .

$i_1$	$i_0$	$s_3$	$s_2$	$s_1$	$s_0$

**Solution:**

$i_1$	$i_0$	$s_3$	$s_2$	$s_1$	$s_0$
0	0	$a_3$	$a_2$	$a_1$	$a_0$
0	1	$a_2$	$a_1$	$a_0$	$a_3$
1	0	$a_1$	$a_0$	$a_3$	$a_2$
1	1	$a_0$	$a_3$	$a_2$	$a_1$

À partir de la table de vérité donnée à la question précédente, déduire une expression algébrique (booléenne) pour les bits de sortie  $s_3$  et  $s_0$ .

**Solution:**

$$s_3 = a_3.\overline{i_1}.\overline{i_0} + a_2.\overline{i_1}.i_0 + a_1.i_1.\overline{i_0} + a_0.i_1.i_0$$

$$s_0 = a_0.\overline{i_1}.\overline{i_0} + a_3.\overline{i_1}.i_0 + a_2.i_1.\overline{i_0} + a_1.i_1.i_0$$

**Question 1.2 : 4 points**

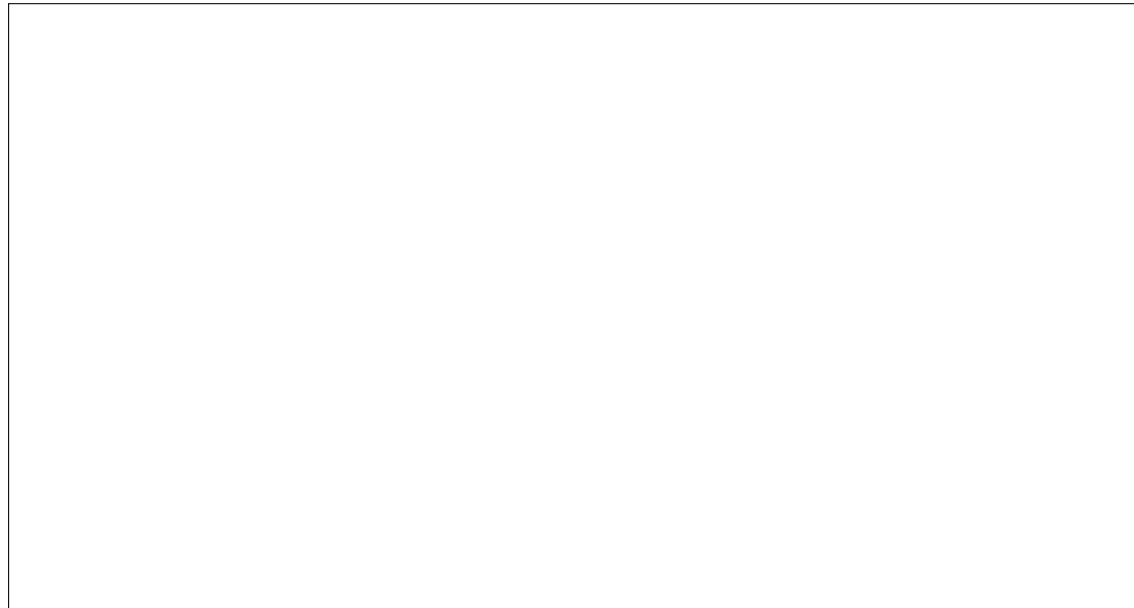
On dispose d'un multiplexeur 2 vers 1 que l'on note  $\text{MUX2}(e_0, e_1, c)$ . Ce circuit a 2 entrées  $e_0$  et  $e_1$  sur 1 bit ainsi qu'une commande  $c$  de sélection sur un bit. Il a pour sortie la valeur de l'entrée  $e_0$  lorsque  $c$  vaut 0 et  $e_1$  lorsque  $c$  vaut 1.

Donner une expression algébrique (booléenne) de  $\text{MUX2}(e_0, e_1, c)$ .

**Solution:**

$$\text{MUX2}(e_0, e_1, c) = \overline{c}.e_0 + c.e_1$$

Déduire pour chacune des expressions données pour les bits  $s_3$  et  $s_0$ , une expression composée uniquement de multiplexeur(s)  $\text{MUX2}$  en justifiant votre réponse pas à pas.



**Solution:**

$$\begin{aligned}s_3 &= a_3.\overline{i_1}.\overline{i_0} + a_2.\overline{i_1}.i_0 + a_1.i_1.\overline{i_0} + a_0.i_1.i_0 \\&= \overline{i_1}.(a_3.\overline{i_0} + a_2.i_0) + i_1.(a_1.\overline{i_0} + a_0.i_0) \\&= \overline{i_1}.\text{MUX2}(a_3, a_2, i_0) + i_1.\text{MUX2}(a_1, a_0, i_0) \\&= \text{MUX2}(\text{MUX2}(a_3, a_2, i_0), \text{MUX2}(a_1, a_0, i_0), i_1)\end{aligned}$$

$$s_0 = \text{MUX2}(\text{MUX2}(a_0, a_3, i_0), \text{MUX2}(a_2, a_1, i_0), i_1)$$

## Exercice 2 : Fonction récursive : fréquence des chiffres – 23 points

On considère la fonction C donnée ci-dessous.

```
unsigned int frequence_chiffre_rec(unsigned int n, unsigned char t[]){
    unsigned int res;

    if (n >= 10){
        res = frequence_chiffre_rec(n / 10, t);
        t[n % 10] = t[n % 10] + 1;
        return res + 1;
    }
    else {
        t[n] = t[n] + 1;
        return 1;
    }
}
```

### Question 2.1 : 13 points

Donner le code assembleur correspondant à la fonction `frequence_chiffre_rec`. Le code doit contenir des commentaires faisant le lien avec le code source. Toute allocation en pile **doit être justifiée** sous forme de commentaires (il faut justifier la taille allouée).

Le code de la fonction peut être **optimisé** en particulier les variables locales peuvent être optimisées en registre.

**Solution:**

```

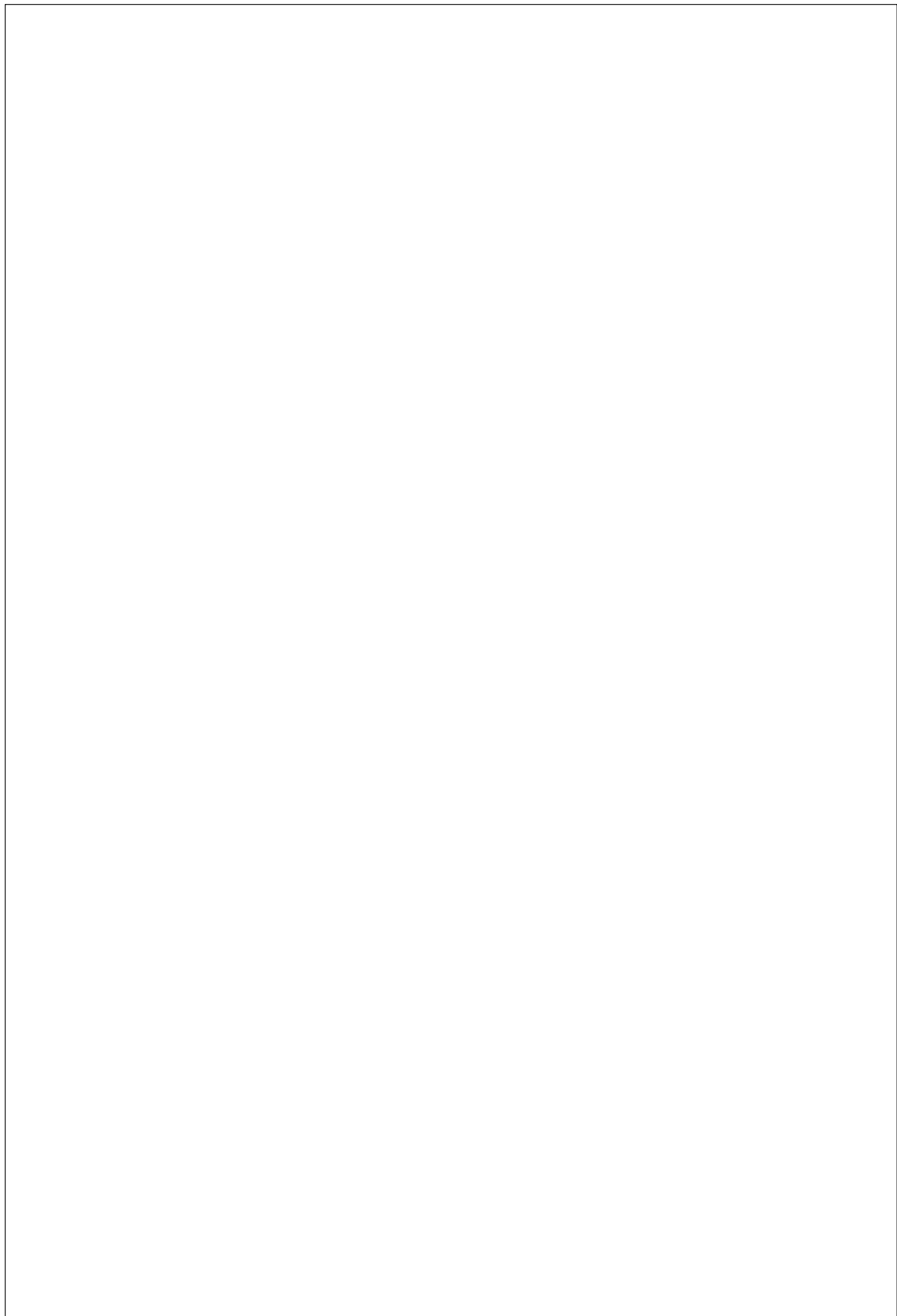
sw    $5, 20($29)    # sauvegarde de t

ori   $10, $0, 10
slt   $11, $4, $10   # $11 = 1 si n < 10
bne   $11, $0, fin_non_rec
# cas recursif
divu  $4, $10        # division par 10
mflo  $4              # n / 10
jal   frequence_chiffre_rec

addiu $2, $2, 1      # res + 1
lw     $4, 16($29)    # relecture n
lw     $5, 20($29)    # relecture t
ori    $10, $0, 10
divu   $4, $10        # division n par 10
mfhi   $9             # n % 10
addu   $9, $9, $5      # adresse de tab[n % 10]
lbu    $11, 0($9)     # lecture tab[n % 10]
addiu  $11, $11, 1     # incrementation
sb     $11, 0($9)      # tab[n%10] <- tab[n%10] + 1
j      epilogue
fin_non_rec:
ori    $2, $0, 1      # valeur de retour vaut 1
addu   $12, $4, $5     # adresse de tab[n]
lbu    $13, 0($12)     # lecture tab[n]
addiu  $13, $13, 1     # incrementation
sb     $13, 0($12)     # tab[n] <- tab[n] + 1

epilogue:
lw     $31, 12($29)
addiu  $29, $29, 16
jr     $31

```



## Question 2.2 : 5 points

On considère le programme principal donné ci-dessous avec les déclarations de données globales.

```
unsigned int nb = 1234;
unsigned char tab_chiffre[10]; /* tableau non initialisé de 10 entiers non signés codés sur 1 octet */

void main() {

    unsigned int nb_ch;

    nb_ch = frequence_chiffre_rec(nb, tab_chiffre);

    printf("%d", nb); /* affichage de nb */

    printf("%d", nb_ch); /* affichage du nombre de chiffres */

    exit();
}
```

Donner le code assembleur correspondant à ces déclarations et au `main`. Le code doit contenir des commentaires faisant le lien avec le code source. Toute allocation en pile **doit être justifiée** sous forme de commentaires (il faut justifier la taille allouée).

Le code de la fonction peut être **optimisé** en particulier les variables locales peuvent être optimisées en registre.



### **Solution:**

```
.data
nb: .word 1234
tab_chiffre:
    .space 10 # alloue 10 octets initialisés par défaut à 0

.text
main:
    addiu $29, $29, -12 # nv = 1 + na = 2

    lui    $8, 0x1001
    lw     $4, 0($8)    # nb = 1er argument
    lui    $5, 0x1001
    ori    $5, $5, 4    # tab_chiffre = 2eme argument
    jal    frequence_chiffre_rec

    ori    $16, $2, 0    # sauvegarde du resultat (nb_ch) dans un registre persistant

    lui    $8, 0x1001
    lw     $4, 0($8)    # lecture nb en memoire
    ori    $2, $0, 1    # affichage nb
    syscall

    ori    $4, $16, 0    # affichage de nb_ch
    ori    $2, $0, 1
    syscall
    addiu  $29, $29, 12
    ori    $2, $0, 10
    syscall

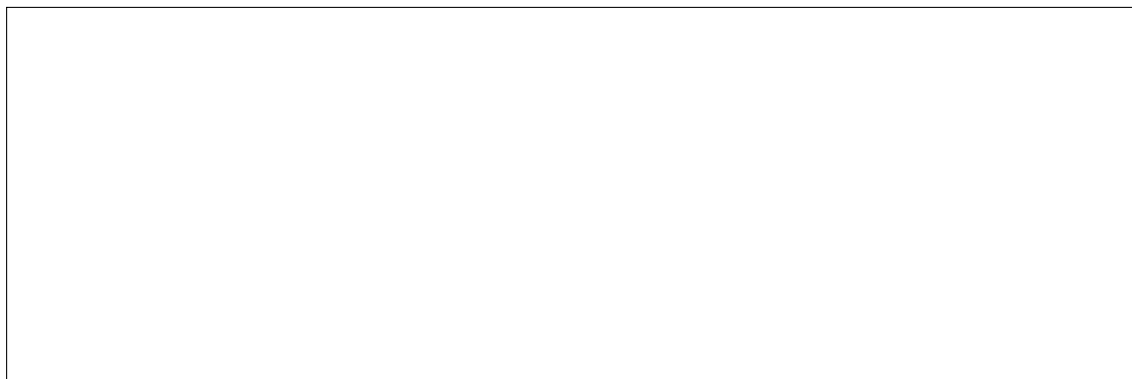
frequence_chiffre_rec:
    addiu  $29, $29, -16 # nv = 1 + na = 2 + nr = 0 + $31
    sw     $31, 12($29)

    sw     $4, 16($29)    # sauvegarde de n
```



**Question 2.3 : 5 points**

Dessiner l'arbre des appels engendrés par l'exécution du programme.



**Solution:**

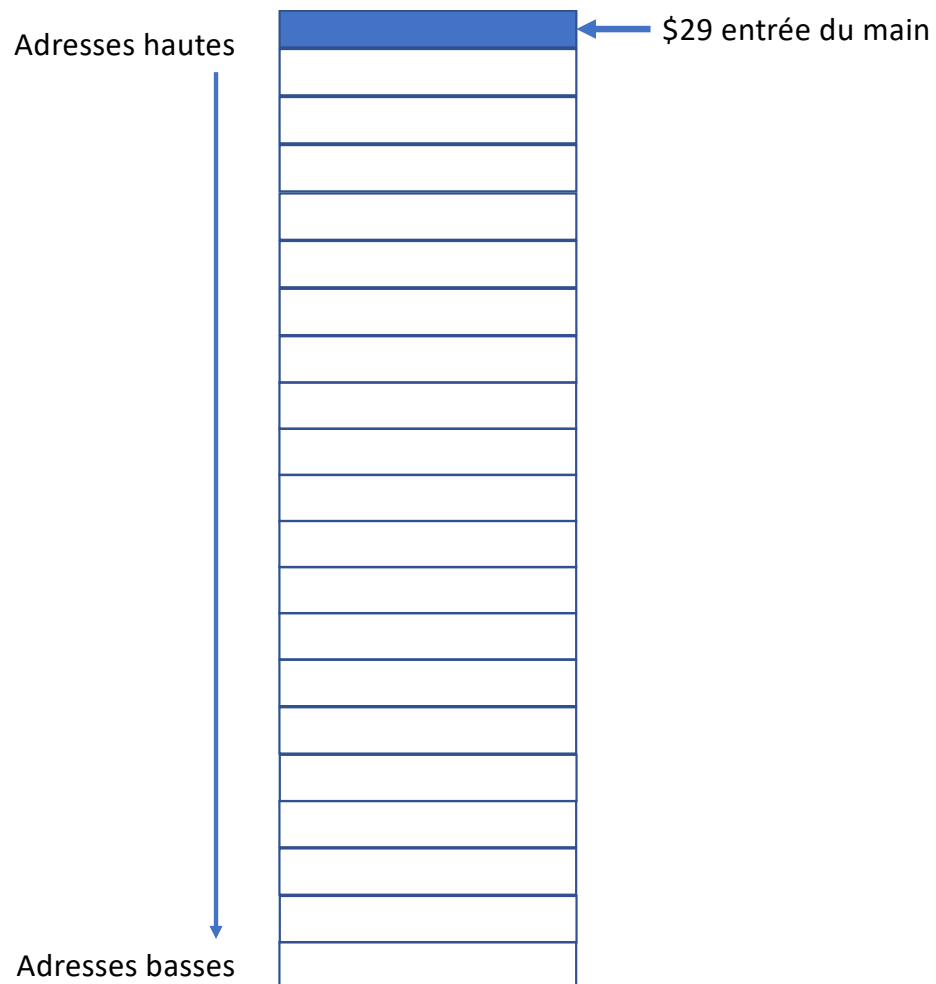
```

main
-> frequence_chiffre_rec(1234)
   -> frequence_chiffre_rec(123)
      -> frequence_chiffre_rec(12)
         -> frequence_chiffre_rec(1)

```

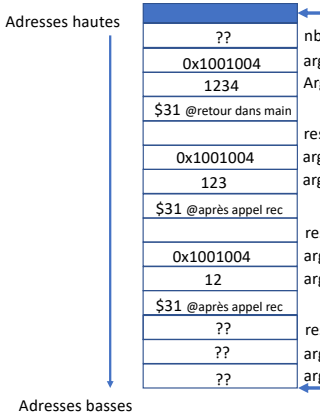
Entourer dans la réponse précédente le 3ème appel à la fonction `frequence_chiffre_rec`.  
 Sur le schéma ci-dessous indiquer la position du pointeur de pile à l'entrée de la fonction `frequence_chiffre_rec` lors de ce 3ème appel. Pour chaque emplacement (représentant un mot) alloué, indiquer :

- à droite de l'emplacement à quoi il correspond,
- dans l'emplacement son contenu (valeur décimale ou hexadécimale) s'il est connu, s'il est non significatif ou inconnu indiquer ??.



**Solution:**

Correspond à l'état de la pile à l'entrée de l'appel `frequence_chiffre_rec(12)`



## Exercice 3 : Architecture et programmation système – 15 points

Pour chaque question du QCM, vous avez 4 affirmations et vous devez dire, pour chacune, si elle est vraie ou fausse. Toutes les affirmations peuvent être vraies, ou fausses, ou un mélange de vraies et de fausses.

**Vous devez cocher 4 cases par question.** Pour chaque question, vous avez :

- 1 point si vous avez coché les 4 cases sans erreur.
- 0,5 point si vous avez 1 erreur ou coché seulement 3 cases.
- 0 point si vous avez commis 2 erreurs ou plus ou coché 2 cases ou moins.

### Question 3.1 : 8 points

#### 1. Affirmations sur l'espace d'adressage du MIPS

- (a) vrai ☐ ou faux ☐  
Les registres des contrôleurs de périphériques sont dans l'espace d'adressage du MIPS.
- (b) vrai ☐ ou faux ☐  
Les registres du processeur sont accessibles dans l'espace d'adressage du MIPS.
- (c) vrai ☐ ou faux ☐  
L'espace d'adressage, c'est l'ensemble des adresses que le MIPS peut produire.
- (d) vrai ☐ ou faux ☐  
Seul le mode kernel permet d'utiliser l'espace d'adressage MIPS.

**Solution:**

(a) vrai ; (b) faux ; (d) vrai ; (c) faux

#### 2. Affirmations sur l'architecture vue dans le module

- (a) vrai ☐ ou faux ☐  
Il est nécessaire d'avoir une ROM (mémoire persistante) pour contenir une partie du code.
- (b) vrai ☐ ou faux ☐  
Le noyau peut choisir les adresses de registres des contrôleurs des TTY.
- (c) vrai ☐ ou faux ☐  
Le code de démarrage du MIPS commence à partir de l'adresse 0x80000180.
- (d) vrai ☐ ou faux ☐  
Le nombre de TTY n'a pas d'impact sur le nombre et les adresses de registres du TTY0.

**Solution:**

(a) vrai ; (b) faux ; (c) faux ; (d) vrai

#### 3. Affirmations sur les modes d'exécution du MIPS

- (a) vrai ☐ ou faux ☐  
L'instruction `stmd` permet de changer le mode d'exécution du MIPS (user et kernel).
- (b) vrai ☐ ou faux ☐  
C'est le registre système `c0_cause` qui permet de définir le mode d'exécution du MIPS.
- (c) vrai ☐ ou faux ☐  
Il est impossible d'exécuter du code user en mode kernel.
- (d) vrai ☐ ou faux ☐  
L'instruction `eret` permet de passer du mode kernel au mode user.

**Solution:**

(a) faux ; (b) faux ; (c) faux ; (d) vrai

#### 4. Affirmations sur la chaine de compilation

- (a) vrai [ ] ou faux [ ]  
L'édition de liens produit le fichier exécutable à partir des fichiers objets.
- (b) vrai [ ] ou faux [ ]  
L'édition de liens n'est pas obligatoire, c'est seulement si l'espace d'adressage n'est pas "standard".
- (c) vrai [ ] ou faux [ ]  
Le Makefile doit être exécuté quand le MIPS est en mode kernel.
- (d) vrai [ ] ou faux [ ]  
Le préprocesseur du langage C permet de transformer le code C en code assembleur.

##### Solution:

(a) vrai ; (b) faux ; (c) faux ; (d) faux

#### 5. Affirmations sur le fichier ldscript

- (a) vrai [ ] ou faux [ ]  
Il y a 1 espace d'adressage, mais 3 fichiers ldscript, 1 pour le noyau, 1 pour les bibliothèques et 1 pour l'application.
- (b) vrai [ ] ou faux [ ]  
Le fichier ldscript contient la description des régions de l'espace d'adressage occupées par la mémoire et la manière de les remplir avec les sections présentes dans les fichiers objet (.o).
- (c) vrai [ ] ou faux [ ]  
C'est dans le fichier ldscript que l'on décide de la taille de chaque pile d'exécution du programme.
- (d) vrai [ ] ou faux [ ]  
Les variables définies dans le fichier ldscript sont accessibles depuis le programme C.

##### Solution:

(a) faux ; (b) vrai ; (c) faux ; (d) vrai

#### 6. Affirmations sur le système d'exploitation

- (a) vrai [ ] ou faux [ ]  
Le noyau du système d'exploitation est compilé avec l'application pour donner un unique exécutable.
- (b) vrai [ ] ou faux [ ]  
Pour demander des services au noyau, on peut utiliser les instructions `syscall` ou `jaland kentry`.
- (c) vrai [ ] ou faux [ ]  
Le noyau du système d'exploitation a un seul point d'entrée pour l'utilisateur, quelle que soit la cause d'appel.
- (d) vrai [ ] ou faux [ ]  
Les fonctions système de la libc (appelées depuis l'application user) s'exécutent toutes en mode kernel.

##### Solution:

(a) faux ; (b) faux ; (c) vrai ; (d) faux

#### 7. Affirmations sur le passage de mode

- (a) vrai [ ] ou faux [ ]  
Il y a 5 adresses d'entrée du noyau : le boot, kentry et les syscalls, interruptions, exceptions.

- (b) vrai [ ] ou faux [ ]  
Lorsqu'une application user s'exécute, elle ne peut pas masquer les interruptions temporaire-  
ment.
- (c) vrai [ ] ou faux [ ]  
Le noyau utilise le registre `c0_sr` (status) pour connaître la cause d'appel (syscall, interrup-  
tion, exception)
- (d) vrai [ ] ou faux [ ]  
Les arguments des appels système sont donnés dans les registres \$4 à \$7.

**Solution:**

(a) faux ; (b) vrai ; (c) faux ; (d) vrai

**8. Affirmations les IRQ**

- (a) vrai [ ] ou faux [ ]  
Les IRQ sont masquées à l'entrée dans le noyau seulement pour les syscall.
- (b) vrai [ ] ou faux [ ]  
Le signal IRQ est levé par un périphérique pour prévenir d'un événement, mais il peut décider  
seul de le baisser si l'événement n'est plus à jour (s'il est devenu obsolète).
- (c) vrai [ ] ou faux [ ]  
Le vecteur d'interruption permet de connaître le numéro d'IRQ.
- (d) vrai [ ] ou faux [ ]  
Le signal IRQ qui sort d'un contrôleur de périphérique contient le numéro de périphérique.

**Solution:**

(a) faux ; (b) faux ; (c) faux ; (d) faux

### Question 3.2 : 2 points

Lors de l'exécution de l'instruction `syscall`, le MIPS est dérouté vers le noyau du système d'exploitation et les registres `PC`, `c0_EPC`, `c0_SR` et `c0_CAUSE` sont modifiés en même temps (l'ordre n'a pas d'importance). Dites quelles sont ces modifications pour chaque registre en les expliquant.

**Solution:**

- $PC \leftarrow 0x80000180$   
Première adresse du noyau ou entrée du noyau ou encore adresse de `kentry`
- $EPC \leftarrow PC$  ou  $EPC \leftarrow$  adresse du `syscall`  
EPC contient l'adresse de l'instruction `syscall`.
- $c0\_SR.EXL \leftarrow 1$  ou juste  $EXL \leftarrow 1$   
Le MIPS passe en mode d'exception
- $c0\_CAUSE.XCODE \leftarrow 8$  ou juste  $XCODE \leftarrow 8$   
Le champ `XCODE` indique la cause d'appel, ici 8 est le code de `syscall`



### Question 3.3 : 5 points

L'écriture et la lecture dans les registres des contrôleurs de périphérique est normalement faite en langage C, mais pour cet examen, nous allons écrire la fonction `int tty_puts(int tty, char buf[])` en assembleur. L'adresse du contrôleur de TTY est `__tty_regs_maps=0xD0200000`, elle est définie dans le fichier `'kernel.ld'` et le label `__tty_regs_maps` est directement utilisable en assembleur. L'ordre des registres de contrôle est (par adresse croissante et pour chaque tty) : WRITE, STATUS, READ et UNUSED. Chacun de ces 4 registres fait 4 octets, mais seul leur octet de poids faible est utilisé. Les registres de chaque TTY se suivent, d'abord TTY0, puis TTY1, etc.

- WRITE est le registre de sortie vers l'écran.
- STATUS est le registre qui contient 0 lorsqu'aucune touche n'a été tapée au clavier,
- READ est le registre qui contient le code ASCII de la touche tapée au clavier.
- UNUSED n'est pas utilisé

La fonction `tty_puts(int tty, char buf[])` prend en argument le numéro du TTY et un pointeur sur un buffer de caractères, elle lit les caractères dans le buffer et les écrit sur le TTY, tant qu'il y a des caractères. À la fin, la fonction rend le nombre de caractères écrits.

Complétez le code assembleur suivant (dans le texte directement) et justifiez succinctement chaque réponse (de Q1 à Q5) dans le cadre qui suit (**une réponse juste non justifiée n'a pas tous les points**).

```
1:  tty_puts: # int tty_puts(int tty, char buf[])
2:          ori    $2,    $0,    0
3:          la     $8,    __tty_regs_maps
4:          sll    $4,    $4,    _____ --> Q1
5:          addu   $8,    $8,    $4
6:          j      tty_puts_cond
7:  tty_puts_loop:
8:          sw     $9,    _____($8) --> Q2
9:          addiu   $5,    $5,    _____ --> Q3
10:         addiu   $2,    $2,    1
11:  tty_puts_cond:
12:         lb      $9,    _____($5) --> Q4
13:         _____ $9,    $0,    tty_puts_loop --> Q5
14:         jr      $31
```

#### Solution:

```
1:  tty_puts: # int tty_puts(int tty, char buf[])
```

```

2:      ori    $2,    $0,    0
3:      la     $8,    __tty_regs_maps
4:      sll    $4,    $4,    4          --> Q1
5:      addu   $8,    $8,    $4
6:      j      tty_puts_cond
7: tty_puts_loop:
8:      sw     $9,    0_($8)          --> Q2
9:      addiu   $5,    $5,    1          --> Q3
10:     addiu   $2,    $2,    1
11: tty_gets_cond:
12:     lb      $9,    0($5)          --> Q4
13:     bne     $9,    $0,    tty_puts_loop --> Q5
14:     jr      $31

```

- Q1 : il faut multiplier par 16 pour chaque TTY, d'où le décalage à gauche de 4 bits
- Q2 : On écrit le caractère dans le registre WRITE
- Q3 : on se déplace d'un octet
- Q4 : on lit le caractère courant
- Q5 : on retourne en début de boucle si ce n'est pas la fin de chaîne.