Numéro d'anonymat (donné sur votre étiquette)

# Examen de substitution 2020 – 2021

Architecture des ordinateurs 1 – LU3IN029

Durée : 1h30

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

Le sujet comporte 20 pages. Ne pas désagrafer les feuilles. Répondre directement sur le sujet. 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.

L'examen est composé de 4 exercices indépendants.



## Exercice 1 : Arithmetique – 8 points

On considère un additionneur-soustracteur 12 bits : les entrées `A` et `B` sont les opérandes sur 12 bits, `cmd` indique l'opération à réaliser : 0 correspond à une addition, 1 à une soustraction. Le mot de 12 bits en sortie est nommé `S`. Il y a aussi les quatres drapeaux `OV`, `CF`, `ZF` et `SF` (identiques à ceux vu en TME).

### Question 1.1 : 2 points

Indiquer pour chacun des 4 drapeaux comment il peut être calculé et à quoi il sert.

**Solution**:

CF (*Carry Flag*) correspond à la valeur de la retenue sortante, permet de détecter des dépassements sur entiers naturels
OV (*Overflow*) correspond au xor des retenues des 2 derniers rangs, permet de détecter des dépassements sur entiers relatifs
ZF (*Zero Flag*) correspond à la négation du ou logique sur tous les bits de S, permet de détecter une sortie nulle ou non nulle
SF (*Sign Flag*) correspond à la valeur du bit de poids fort de S, permet de connaitre le signe du résultat

## Question 1.2 : 2 points

Donner la valeur de `S` (en hexadécimal), `OV`, `CF`, `ZF` et `SF` lorsque les entrées de l'additionneur-soustracteur sont `0xFFC` pour pour `A`, et `0x004` pour `B` et 0 pour la commande `cmd`.

**Solution**:

`S = 0x000, CF = 1, OV = 0, SF = 0, ZF = 1`

Si les entrées représentent des entiers naturels codés en base 2, est ce qu'il y a un dépassement de capacité ? Justifier.

**Solution**:

Oui, car `CF` vaut 1 ou parce que la somme dépasse $2^{12} - 1$.

Si les entrées réprésentent des entiers relatifs codés en complément à 2, est ce qu'il y a un dépassement de capacité ? Justifier.

**Solution**:

Non car `OV` vaut 0 ou parce que l'on additionne de 2 nombres de signes différents.

## Question 1.3 : 2 points

Donner deux mots binaires `m1` et `m2` de 12 bits tels qu'une addition sur ces deux mots engendre un dépassement de capacité s'il représentent des entiers relatifs (codés en complément à deux) mais n'engendre pas

de dépassement de capacité s'il représentent des entiers naturels (codés en base 2). Justifier votre réponse.

**Solution**:

OV = 1 donc les entiers relatifs doivent être de même signe
not(CF) donc entiers relatifs de valeur positive, et donc les quartets de poids fort sont $<= 7$
et comme OV =1 donc leur somme doit être $>= 8$.

Par exemple `0x400` pour `A` et `B` soit l'addition $2^{10} + 2^{10}$, le résultat vaut $2^{11}$ (S = `0x800`) qui n'est pas représentable sur 12 bits en complément à 2 ($2^{11} > 2^{11} - 1$, `0x800` s'interprète comme $-2^{11}$), mais est représentable sur 12 bits en base 2 ($2^{11} < 2^{12} - 1$).

## Question 1.4 : 2 points

On souhaite réaliser la soustraction entre les valeurs -4 et 2047 (2047 est la valeur à soustraire à -4). Quelles sont les valeurs (en hexadécimal pour les mots de plusieurs bits) à mettre en entrée de l'additionneur ? Quelles seront les valeurs de la sortie `S` et des drapeaux ? Justifier votre réponse.

Remarque : 2047 = $2^{11} - 1$

**Solution**:

`A` = `0xFFC`, `B` = `0x7FF`, `cmd` = 1.

Comme soustraire c'est additionner l'opposé, le résultat de `0xFFC - 0x7FF` est le même que celui de `0xFFC + 0x801` qui vaut `0x7FD`

> On a donc `S = 0x7FD`, `CF = 1`, `OV = 1`, `ZF = 0`, `SF = 0`
>
> Remarque : le résultat correct vaut -2051 ce qui n'est pas représentable sur 12 bits car l'intervalle de représention est [-$2^{11}$, $2^{11} - 1$]. On a bien OV qui vaut 1 et un résultat positif alors que cela revient à additionner deux nombres négatifs.

# Exercice 2 : Programmation assembleur – 9 points

On considère le programme C ci-dessous. La fonction `echange(str, index1, index2)` intervertit les caractères de la chaine `str` en position `index1` et `index2` si les indices `index1` et `index2` sont inférieurs strictement à la longueur de la chaine `str`.

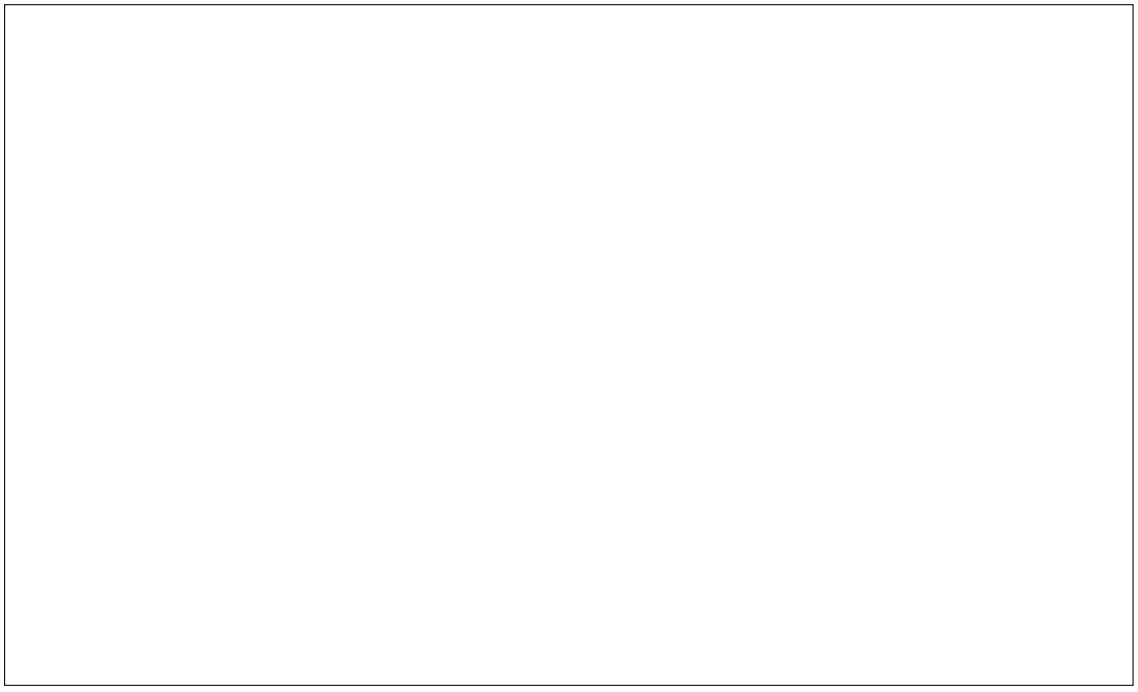
```c
void echange(char * str, int index1, int index2);

void main(){
  char chaine[8] = "bonjour";

  int i = 0;
  int j = 3;

  echange(chaine, i, j);

  printf("%s", chaine);

  return;
}
```

## Question 2.1 : 6 points

Donner le code assembleur correspondant uniquement au programme principal (`main`) **sans optimisation**. Le code de la fonction `echange` n'est pas demandé.
**Attention** : votre code devra être écrit lisiblement et contenir des commentaires pour justifier le nombre d'octets alloués sur la pile ainsi que des commentaires pour expliquer les différentes instructions de votre code.
**Rappel** : on peut indiquer un caractère comme opérande immédiat d'une instruction arithmétique ou logique, par exemple `ori $8, $0, 'a'`

**Solution**:

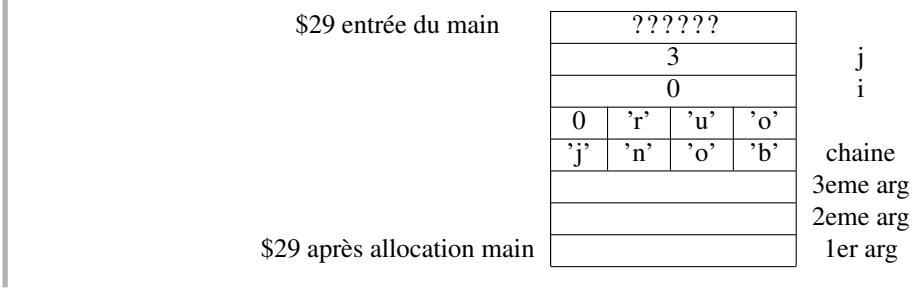
```
.data
.text
 # nv = 2 + 8/4 = 4 + na = 3 soit 7 mots
addiu $29, $29, -28
ori   $8, $0, 'b'
sb    $8, 12($29)
ori   $8, $0, 'o'
sb    $8, 13($29)
ori   $8, $0, 'n'
sb    $8, 14($29)
ori   $8, $0, 'j'
sb    $8, 15($29)
ori   $8, $0, 'o'
sb    $8, 16($29)
ori   $8, $0, 'u'
sb    $8, 17($29)
ori   $8, $0, 'r'
sb    $8, 18($29)
sb    $0, 19($29)
sw    $0, 20($29)  # i = 0
ori   $8, $0, 3
sw    $8, 24($29) # j = 3

addiu $4, $29, 12   # 1er param chaine
lw    $5, 20($29)
lw    $6, 24($29)
jal   echange

addiu $4, $29, 12
ori   $2, $0, 4
syscall

addiu $29, $29, 24
ori   $2, $0, 10
syscall
```



### Question 2.2 : 3 points

Dessiner l'état de la pile juste avant l'affichage de la chaine. Vous donnerez les valeurs connues contenues dans les emplacements alloués, en indiquant des valeurs numériques pour les variables entières, et des caractères pour les emplacements contenant des caractères. Vous indiquerez aussi à coté des emplacements à quoi ils correspondent.

**Solution**:




## Exercice 3 : Fonction récursive – 15 points

On considère le programme C suivant :

```
int size = 5;
int tab1[5];
int tab2[5];

int somme_cumul(int *t1, int *t2, int i, int n);

void main(){
  int res;
```

```c
  int val;
  int i = 0;
  /* remplissage de tab1 avec des valeurs positives lues au clavier */
  while (i < size){
    scanf("%d", &val);   /* lecture d'un entier mis dans val */
    if (val > 0) {
      tab1[i] = val;
      i++;
    }
  }
  res = somme_cumul(tab1, tab2, 0, size - 1);
  printf("%d", res);
  exit(0);
}
```

## Question 3.1 : 1 points

Donner le contenu de la section `.data` du programme assembleur correspondant au programme C donné ci-dessus. Vous indiquerez les adresses des variables en commentaire.

**Solution**:

```
.data
size:   .word 5     # 0x10010000
tab1:   .align 2    # 0x10010004
        .space 20
tab2:   .align 2     # 0x10010018
        .space 20
```

## Question 3.2 : 6 points

Donner le code correspondant au programme principal. Votre code devra être lisible et comporter des commentaires justifiant le nombre d'octets alloués sur la pile ainsi que des commentaires expliquant votre code.

**Solution**:

```
.text
# nv = 3 + na = 4 => 7 mots
      addiu  $29, $29, -28
      lui    $16,  0x1001
      lw     $17,  0($16)       # size (dans un registre persistant)
      ori    $16,  $16, 4       # &tab1[0] (dans un registre persistant)
      ori    $18,  $0, 0        # i (dans un registre persistant)

while:
      slt    $8,  $18, $17   # $11 = 1 si i < size
      beq    $8,  $0, suite
      ori    $2,  $0, 5
      syscall             # lecture d'un entier

      blez   $2,  while
      sll    $9,  $18, 2     # i * 4
      add    $9,  $9, $16    # adresse tab1[i]
      sw     $2,  0($9)      # tab1[i] = val
      addiu  $18, $18, 1     # i = i + 1
      j while
suite:
      lui    $4,  0x1001
      ori    $4,  $4, 4     # 1er arg = tab1
      lui    $5,  0x1001
      addiu  $5,  $5, 24    # 2eme arg = tab2
      ori    $6,  $0, 0     # 3eme arg = 0
      or     $7,  $0, $17   # 4eme arg = size - 1
      addiu  $7,  $7, -1
      jal    somme_cumul
      ori    $4,  $2, 0
      ori    $2,  $0, 1
      syscall

      addiu  $29, $29, 28
      ori    $2,  $0, 10
      syscall
```

Voici le code C de la fonction `somme_cumul`.

```
int somme_cumul(int *t1, int *t2, int i, int n){
  int res;
  if (i == n){
    t2[i] = t1[i];
    return t2[i];
  }
  res = somme_cumul(t1, t2, i+1, n);
  t2[i] = t1[i] + res;
  return t2[i];
}
```

## Question 3.3 : 7 points

Donner le code correspondant à la fonction `somme_cumul`.

**Solution**:

```
somme_cumul:
      # nv = 1, na = 4, nr = 0 + $31 => 6 mots
      addiu  $29, $29, -24
      sw     $31, 20($29)
      sw     $4,  24($29)          # sauvegarde t1
      sw     $5,  28($29)          # sauvegarde t2
      sw     $6,  32($29)          # sauvargarde i

      bne    $6,  $7, cas_rec
      sll    $6,  $6, 2
      addu   $8,  $4, $6  # &t1[i]
      lw     $8,  0($8)   # t1[i]
      addu   $9,  $5, $6   # &t2[i]
      sw     $8,  0($9)
      ori    $2,  $8, 0
      j      epilogue

cas_rec:
      addiu  $6, $6, 1      # i + 1
      jal    somme_cumul
      lw     $4,  24($29)   # lecture t1 sur la pile
      lw     $5,  28($29)   # lecture t2 sur la pile
      lw     $6,  32($29)   # lecture i sur la pile
      sll    $6,  $6, 2
      addu   $8,  $4, $6  # &t1[i]
      lw     $8,  0($8)   # t1[i]
      add    $8,  $8, $2  # t1[i] + res
      addu   $9,  $5, $6   # &t2[i]
      sw     $8,  0($9)
      ori    $2,  $8, 0
 epilogue:
      lw     $31, 20($29)
      addiu  $29, $29, 24
      jr     $31
```

**Solution**:

# Exercice 4 : Architecture et programmation système – 12 points

Cette partie du module d'architecture est évaluée par un QCM et un petit exercice de codage en C.

Pour chaque question du QCM, nous faisons 4 affirmations et vous devez dire, pour chacune, si elle est vraie ou fausse en cochant l'une ou l'autre des cases correspondantes.

**Attention** :

- Pour une affirmation, si vous ne cochez aucune case ou si vous cochez les deux cases, alors votre réponse est considérée comme une erreur.
- Les questions ne sont pas difficiles, mais vous devez prendre le temps de réfléchir avant de répondre.
- Toutes les affirmations peuvent être vraies, ou toutes peuvent être fausses, ou il peut y en avoir un mélange de vraies et de fausses affirmations.

Pour chaque question du QCM, le barème est le suivant :

- 1 point si vous n'avez commis aucune erreur.
- 0,5 point si vous avez commis une erreur.
- 0 point si vous avez commis 2 erreurs ou plus.

## Question 4.1 : 7 points

1. **Propositions sur l'espace d'adressage du MIPS**
   (a) vrai [ ] ou faux [ ]
   L'espace d'adressage user n'est accessible que si le processeur est en mode user.
   (b) vrai [ ] ou faux [ ]
   L'expace d'adressage, c'est l'ensemble des adresses que le MIPS peut produire.
   (c) vrai [ ] ou faux [ ]
   Les registres des contôleurs de périphériques sont dans l'espace d'adressage du MIPS.
   (d) vrai [ ] ou faux [ ]
   Les registres du processeur sont accessibles dans l'espace d'adressage du MIPS.

   **Solution**:
   (a) faux ; (b) vrai ; (c) vrai ; (d) faux

2. **Propositions sur l'architecture vue dans le module**
   (a) vrai [ ] ou faux [ ]
   Les plages d'adresses utilisables dans l'espace d'adressage du processeur (pour la mémoire ou les contrôleurs de périphériques) sont choisies par le noyau.
   (b) vrai [ ] ou faux [ ]
   Les registres des contrôleurs de périphériques sont lus et écrits comme de la mémoire.
   (c) vrai [ ] ou faux [ ]
   Le code de démarrage du MIPS est dans le segment de code du noyau (là où se trouve, entre autre, le gestionnaire de syscall).
   (d) vrai [ ] ou faux [ ]
   Les adresses des registres de contrôle des terminaux TTY sont dans la pile du noyau.

   **Solution**:
   (a) faux ; (b) vrai ; (c) faux ; (d) faux

3. **Propositions sur les modes d'exécution du MIPS**
   (a) vrai [ ] ou faux [ ]
   L'exécution de l'instruction `eret` en mode user aura pour conséquence le passage en mode kernel.

(b) vrai [ ] ou faux [ ]
Le MIPS dispose de deux modes d'exécution (user et kernel) et il peut passer de l'un à l'autre grâce à l'instruction privilégiée `smod`.

(c) vrai [ ] ou faux [ ]
L'instruction `syscall` est une instruction privilégiée utilisable en mode kernel uniquement.

(d) vrai [ ] ou faux [ ]
L'existence d'un mode user pour les applications est indispensable, sans quoi l'utilisation du processeur serait impossible pour exécuter du code.

**Solution**:
(a) vrai ; (b) faux ; (c) faux ; (d) faux

4. **Propositions sur la chaine de compilation**

   (a) vrai [ ] ou faux [ ]
   Le préprocesseur produit du code binaire exécutable (non éditable par un éditeur de texte normal).

   (b) vrai [ ] ou faux [ ]
   L'édition de liens produit le binaire exécutable à partir des différents fichiers objets issus de la phase de compilation

   (c) vrai [ ] ou faux [ ]
   Le terme "*chaîne de compilation*" signifie "*compilateur*" (ici gcc).

   (d) vrai [ ] ou faux [ ]
   Un Makefile permet de construire un exécutable grâce à des règles contenant des commandes en langage shell.

   **Solution**:
   (a) faux ; (b) vrai ; (c) faux ; (d) vrai

5. **Propositions sur le fichier ldscript**

   (a) vrai [ ] ou faux [ ]
   Un fichier ldscript contient la description des régions de l'espace d'adressage occupé par la mémoire et la manière de les remplir avec les sections présentes dans les fichiers objet (.o).

   (b) vrai [ ] ou faux [ ]
   Le fichier ldscript est utilisé par gcc pendant l'étape de compilation du code C (.c) pour produire le code objet (.o)

   (c) vrai [ ] ou faux [ ]
   Les variables définies dans le fichier ldscript (définissant par exemple des adresses) sont accessibles depuis le programme C.

   (d) vrai [ ] ou faux [ ]
   Si un programme est composé d'un seul fichier C et que ce fichier n'a qu'une seule fonction, l'usage du ldscript n'est pas obligatoire.

   **Solution**:
   (a) vrai ; (b) faux ; (c) vrai ; (d) faux

6. **Propositions sur le système d'exploitation**

   (a) vrai [ ] ou faux [ ]
   Tout le code du système d'exploitation (libc et kernel) s'exécute toujours en mode kernel.

   (b) vrai [ ] ou faux [ ]
   Le noyau du système d'exploitation est écrit en assembleur.

   (c) vrai [ ] ou faux [ ]
   Le gestionnaire de syscall est dans la libc et pas dans le kernel.

   (d) vrai [ ] ou faux [ ]
   L'emplacement en mémoire du noyau du système d'exploitation est un choix du programmeur.

   **Solution**:
   (a) faux ; (b) faux ; (c) faux ; (d) faux

7. **Propositions sur le passage de mode**

   (a) vrai [ ] ou faux [ ]
   Il y a seulement deux points d'entrée dans le noyau : la fonction `kinit()` et la routine `kentry` à l'adresse 0x80000180.

(b) vrai [ ] ou faux [ ]
Lorsqu'une application de l'utilisateur s'exécute en mode user, il est possible de masquer les exceptions grâce au registre de status (`c0_sr`).

(c) vrai [ ] ou faux [ ]
La première chose que le noyau doit faire après une des trois causes d'appel (exécution de l'instruction syscall, interruption et exception) est d'analyser la cause d'appel.

(d) vrai [ ] ou faux [ ]
Les arguments des appels système sont donnés dans les registres $4 à $7.

**Solution**:
(a) vrai ; (b) faux ; (c) vrai ; (d) vrai

## Question 4.2 : 5 points

L'adresse du contrôleur de terminal TTY est `__tty_regs_maps=0xD0200000`, elle est définie dans le fichier '`kernel.ld`' et l'ordre des registres de contrôle est (par adresse croissante) :
`write`, `status`, `read`, `unused`.

- `write` est le registre de sortie vers l'écran.
  Chaque écriture est faite à l'emplacement du curseur, lequel avance automatiquement.
- `status` est le registre qui contient 0 lorsqu'aucune touche n'a été tapée au clavier,
  et autre chose que 0, lorsqu'un caractère est en attente de lecture dans le registre `read`.
- `read` est le registre qui contient le code ASCII de la touche tapée au clavier.
  Attention : il ne faut lire le registre `read` que si le registre `status` est différent de 0.

Pour déclarer les registres du contrôleur de TTY, vous utiliserez la déclaration d'un tableau :
Chaque case de ce tableau permet d'accéder l'un des registres décrit plus haut.

```
extern volatile int __tty_regs_maps[4];
```

Écrire une fonction en C `loopback()` qui lit en boucle les caractères depuis le clavier et les renvoie vers l'écran du terminal après avoir mis les lettres minuscules en majuscules. Cette fonction ne prend pas d'argument et ne rend rien. C'est donc une boucle sans fin qui attend qu'un caractère soit tapé, puis lit le caractère tapé, regarde si c'est une minuscule (de 'a' à 'z') pour la mettre en majuscule (de 'A' à 'Z') et enfin écrit le caractère sur l'écran.

Si vous tapez au clavier : `"Bonjour, nous sommes le 13 MARS"`
ce qui s'affiche est : `"BONJOUR, NOUS SOMMES LE 13 MARS"`

Le passage de minuscule à majuscule peut se faire de la manière suivante :
Si "`c`" est un caractère minuscule, alors "`(c + 'A' - 'a')`" est le caractère majuscule correspondant.

Vous devez commenter votre code.

Le prototype de la fonction est : `void loopback(void);`

**Solution**:

```
extern volatile int __tty_regs_maps[4];
void loopback(void)
{
    int c ;
    while ( 1 )
    {
        while ( __tty_regs_maps[1] == 0) ;
        c = __tty_regs_maps[2];
        if ( (c >= 'a') && (c <= 'z') )
            c = c + 'A' - 'a';
        __tty_regs_maps[0] = c ;
    }
}
```