Nom :	Prénom :
N° Étudiant :	Groupe de TD :

Partiel 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. Le barème total est lui aussi donné à titre indicatif : 40 points.

Le partiel est composé d'exercices indépendants.

Exercice 1 : Arithmétique – 3 points

On considère le mot binaire sur 8 bits $m_1 = \texttt{0b10100011}$.

Question 1.1 : 1 points

On interprète le mot m_1 par un entier naturel.

1. Quelle est la valeur décimale $n_1 \in \mathbb{N}$ de ce mot lorsqu'il est interprété par un entier naturel ? Justifier la réponse.

Solution:

$n_1 = 2^7 + 2^5 + 2^1 + 2^0 = 128 + 32 + 2 + 1 = 163$

2. Donner le mot binaire m_2 de 16 bits représentant le même entier naturel n_1.

Solution:

$m_2 = 0000\,0000\,1010\,0011$

3. À partir de la valeur décimale de $r_1 = n_1 + n_1$, justifier combien de bits au minimum il faut pour effectuer sur l'additionneur l'opération $n_1 + n_1$ et obtenir le résultat r_1.

Solution:
$n_1 + n_1 = 163 + 163 = 326 \notin [0, 255]$ donc il faudra 9 bits pour représenter l'entier naturel 163 (car $326 \in [0, 2^9 - 1] = [0, 511]$)

Question 1.2 : 1 points

On interprète le mot m_1 par un entier relatif.

1. Quelle est la valeur décimale $n_2 \in \mathbb{Z}$ de ce mot lorsqu'il est interprété par un entier relatif (représenté en complément à 2 sur 8 bits) ? Justifier la réponse.

Solution:
$n_2 = -2^7 + 2^5 + 2^1 + 2^0 = -128 + 32 + 2 + 1 = -93$

2. Donner le mot binaire m_3 de 16 bits représentant le même entier relatif n_2.

Solution:
$m_3 = 1111\,1111\,1010\,0011$

3. À partir de la valeur décimale de $r_2 = n_2 + n_2$, justifier combien de bits au minimum il faut pour effectuer sur l'additionneur l'opération $n_2 + n_2$ et obtenir le résultat r_2.

Solution:
$n_2 + n_2 = -93 - 93 = -186 \notin [-128, 127]$ donc il faudra 9 bits pour représenter l'entier relatif -186 (car $-186 \in [-2^8, 2^8 - 1] = [-256, 255]$)

Question 1.3 : 1 points

Réaliser l'addition des 2 mots binaires de 8 bits suivants, puis, en observant les retenues, indiquez s'il y a un dépassement de capacité si on considère les opérandes comme des entiers naturels et si on considère les opérandes comme des entiers relatifs. Justifier vos réponses.

$$\begin{array}{cccccccccc} & 1 & 0 & 1 & 0 & & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 0 & 0 & & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

Solution:

$$\begin{array}{ccccccccccc} (1) & (0) & (0) & (0) & (0) & & (0) & (1) & (1) & & \\ & 1 & 0 & 1 & 0 & & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 0 & 0 & & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 & 0 & & 1 & 1 & 1 & 0 \end{array}$$

Si les opérandes sont des entiers naturels, on observe la dernière retenue qui vaut 1 et il y a donc un dépassement. Si les opérandes sont des entiers relatifs, on observe les deux dernières retenues qui valent respectivement 1 et 0 et il y a donc un dépassement car 1 xor $0 = 1$.

Exercice 2 : Circuits – 5 points

On souhaite réaliser un circuit permettant d'évaluer les réponses données par un étudiant à **une** question d'un QCM (questionnaire à choix multiples).

Trois réponses sont proposées par question et l'étudiant peut choisir 0, 1, 2 ou 3 réponses parmi elles.

Question 2.1 : 1 points

Pour chaque réponse proposée r, représentée sur 1 bit :

- l'étudiant peut choisir de sélectionner la réponse et dans ce cas de positionner r à 1, ou de ne pas sélectionner la réponse et dans ce cas de positionner r à 0 ;
- on dispose d'une entrée s indiquant si la réponse est correcte (et dans ce cas $s = 1$) ou non (et dans ce cas $s = 0$)

A partir des deux bits r et s on souhaite calculer deux sorties b (le bonus sur 1 bit) et m (le malus sur 1 bit) telles que :

- $b = 1$ si et seulement si la réponse r est sélectionnée par l'étudiant et la solution s indique que la réponse est correcte
- $m = 1$ si et seulement si l'étudiant a sélectionné la réponse r mais que s indique que cette réponse n'est pas correcte.

On appelle Q le composant ayant r et s pour entrée et b et m pour sortie. Donner les expressions booléennes des sorties b et m en fonction de r et s.

Solution:

r	s	b	m
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

$b = r \cdot s$
$m = r \cdot \overline{s}$

Question 2.2 : 1 points

Rappeler la table de vérité d'un additionneur 1 bit (utilisé pour additionner 1 bit a avec 1 bit b) avec 1 bit c_{in} de retenue entrante et expliquer comment ce composant peut être utilisé pour donner la représentation binaire sur 2 bits d'un entier naturel correspondant au nombre de 1 présents dans une entrée sur 3 bits. On appelle A ce composant.

Solution:

a	b	c_{in}	c_{out}	s	nb bits à 1
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

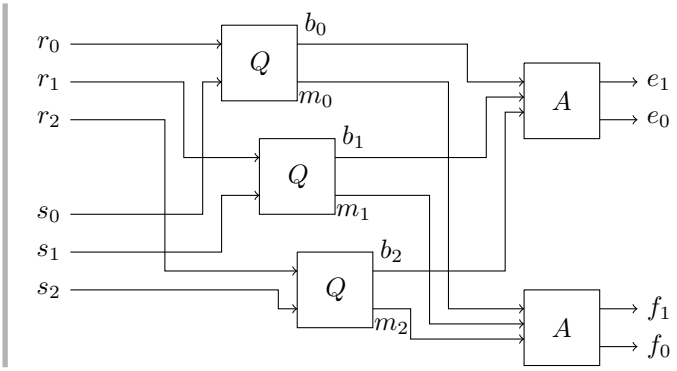
La dernière colonne indique l'entier naturel correspondant au nombre de 1 présents sur l'entrée abc_{in}. La représentation binaire de cet entier naturel sur 2 bits est donnée par $c_{out}s$.

Question 2.3 : 1 points

En utilisant uniquement des composants A et Q, construire un circuit permettant d'évaluer le résultat d'une question d'un QCM. Ce circuit :

- comprend 6 entrées :
  - r_2, r_1, r_0 indiquant quelles sont parmi les trois réponses proposées celles qui sont sélectionnées par l'étudiant
  - s_2, s_1, s_0 indiquant quelles sont parmi les trois réponses proposées celles qui sont correctes
- comprend 4 sorties :
  - e_1e_0 correspondant à la représentation binaire sur 2 bits du nombre de réponses correctes sélectionnées par l'étudiant (e_0 est le bit de poids faible)
  - f_1f_0 correspondant à la représentation binaire sur 2 bits du nombre de réponses incorrectes sélectionnées par l'étudiant (f_0 est le bit de poids faible)

Solution:



Question 2.4 : 1 points

Quels sont les couples (e_1e_0, f_1f_0) possibles en sortie du circuit de la question précédente ?

Solution:

Pour chaque question l'étudiant peut sélectionner 0, 1, 2 ou 3 réponses :

— s'il sélectionne 3 réponses, chacune d'elle peut être correcte ou incorrecte, et la somme du nombre de réponses correctes avec le nombre de réponses incorrectes vaut 3 :

$$\underbrace{(11,00)}_{(3,0)}, \underbrace{(10,01)}_{(2,1)}, \underbrace{(01,10)}_{(1,2)}, \underbrace{(00,11)}_{(0,3)}$$

— s'il sélectionne 2 réponses, chacune d'elle peut être correcte ou incorrecte, et la somme du nombre de réponses correctes avec le nombre de réponses incorrectes vaut 2 :

$$\underbrace{(10,00)}_{(2,0)}, \underbrace{(01,01)}_{(1,1)}, \underbrace{(00,10)}_{(0,2)}$$

— s'il sélectionne 1 réponse, elle peut être correcte ou incorrecte, et la somme du nombre de réponses correctes avec le nombre de réponses incorrectes vaut 1 :

$$\underbrace{(01,00)}_{(1,0)}, \underbrace{(00,01)}_{(0,1)}$$

— s'il ne sélectionne aucune réponse, on obtient $(00,00)$.

Question 2.5 : 1 points

On souhaite compléter le circuit de la question précédente pour qu'à partir de e_1e_0 et f_1f_0, une sortie h sur 1 bit indique si l'étudiant a sélectionné strictement plus de bonnes réponses que de mauvaises réponses. Il s'agit donc pour chacun des couples obtenus à la question précédente de comparer les entiers présents dans le couple. Déduire de la question précédente l'expression booléenne de h en fonction de e_1, e_0, f_1 et f_0.

Solution:

Il s'agit de construire un comparateur simplifié d'entiers naturels représentés sur 2 bits. Les seuls entiers à comparer sont ceux correspondant aux couples obtenus à la question précédente. La sortie h vaut 1 lorsque l'entier naturel représenté par e_1e_0 est strictement supérieur à l'entier naturel représenté par f_1f_0.

e_1	e_0	f_1	f_0	h
1	e_0	f_1	f_2	1
0	1	0	0	1

Parmi les couples à comparer possibles, seuls 4 correspondent à une sortie $h = 1$: $\underbrace{(11,00)}_{(3,0)}, \underbrace{(10,01)}_{(2,1)}, \underbrace{(10,00)}_{(2,0)}, \underbrace{(01,00)}_{(1,0)}$

En remarquant que les 3 premiers sont les seuls couples (parmi les couples possibles) où $e_1 = 1$, on obtient $h = e_1 + \left(\overline{e_1} \cdot e_0 \cdot \overline{f_1} \cdot \overline{f_0}\right)$ qui se simplifie en $h = e_1 + \left(e_0 \cdot \overline{f_1} \cdot \overline{f_0}\right)$.

Exercice 3 : Mémoire et instructions assembleur – 14 points

On considère qu'après chargement d'un programme assembleur MIPS le contenu du segment de données est le suivant :

	Vue par octet				Vue par mot
adresse (de mot)	+ 0	+ 1	+ 2	+3	
0x10010000					0x10010004
0x10010004					0x00313233
0x10010008	0xDD	0xCC	0xBB	0xAA	
0x1001000c					0x00FFFFFF
0x10010010	0x21	0x98	0x43	0xBA	
0x10010014	0x00	0x00	0x00	0x00	
0x10010018	0x00	0x00	0x00	0x00	

Question 3.1 : 3 points

1. Dans le tableau ci-dessus, indiquez le contenu mémoire de toutes les cases vides.

Solution:

	Vue par octet				Vue par mot
adresse (de mot)	+ 0	+ 1	+ 2	+3	
0x10010000	**0x04**	**0x00**	**0x01**	**0x10**	0X10010004
0x10010004	**0x33**	**0x32**	**0x31**	**0x30**	0x00313233
0x10010008	0xDD	0xCC	0xBB	0xAA	**0xAABBCCDD**
0x1001000c	**0xFF**	**0xFF**	**0xFF**	**0x00**	0x00FFFFFF
0x10010010	0x21	0x98	0x43	0xBA	**0xBA439821**
0x10010014	0x00	0x00	0x00	0x00	**0x00000000**
0x10010018	0x00	0x00	0x00	0x00	**0x00000000**

2. Donner un contenu possible de la section de données du programme assembleur MIPS permettant, après son assemblage et son chargement en mémoire, d'obtenir le contenu de la mémoire tel que donné ci-dessus.

Vous utiliserez au moins 5 directives d'allocation différentes.

Solution:

```
.data
    .word 0x10010004
    .asciiz "321"
    .byte 0xDD
    .byte 0xCC
    .half 0xAABB
    .word 0x00FFFFFF
    .word 0xBA439821
    .space 8
```

Question 3.2 : 1 points

Donner une suite d'instructions permettant de ranger dans les registres `$8` et `$9` les valeurs `0x00009821` et `0xFFFFBA43` uniquement à partir des données contenues dans la mémoire.

Solution:

```
.text     # non obligatoire

  lui $10, 0x1001
  lhu $8, 16($10)
  lh  $9, 18($10)
```

Question 3.3 : 5 points

On suppose que les registres `$8` et `$9` contiennent respectivement les valeurs `0x00009821` et `0xFFFFBA43`. Donner une suite d'instructions qui à partir de ces 2 registres permet d'obtenir dans le registre `$10` la valeur `0xBA984321`. Vos commentaires devront aider le lecteur à comprendre les différentes étapes de votre code et sa correction.

Seules des opérations de décalages ou logiques sont autorisées.

Solution:

```
#$8 contient 0x00009821 et $9 contient 0xFFFFBA43
andi $11, $8,  0xFF    # 0x21
andi $12, $8,  0xFF00  # 0x9800
sll  $12, $12, 4       # 0x980000
andi $13, $9,  0xFF    # 0x43
sll  $13, $11, 4       # 0x4300
andi $14, $9,  0xFF00  # 0xBA00
sll  $10, $14, 16      # 0xBA000000
ori  $10, $10, $13     # 0xBA004300
ori  $10, $10, $12     # 0xBA984300
ori  $10, $10, $11     # 0xBA984321
```

Question 3.4 : 5 points

On considère les instructions suivantes avec leurs adresses d'implantation.

Langage d'assemblage	Adresse d'implantation
`lui $8, 0x1001`	`0x00400000`
`lw $8, 0($8)`	`0x00400004`
`xor $10, $10, $10`	`0x00400008`
`etiq1:`	
`lb $9, 0($8)`	`0x0040000c`
`beq $9, $0, etiq2`	`0x00400010`
`andi $9, $9, 0x0F`	`0x00400014`
`addu $10, $10, $9`	`0x00400018`
`addiu $8, $8, 1`	`0x0040001c`
`j etiq1`	`0x00400020`
`etiq2:`	
`ori $4, $10, 0`	`0x00400024`
`ori $2, $0, 1`	`0x00400028`
`syscall`	`0x0040002c`
`ori $2, $0, 10`	`0x00400030`
`syscall`	`0x00400034`

Donner, en justifiant, le codage binaire des instructions `j etiq1` et `addu $10, $10, $9`. Donner pour chacune leur représentation en hexadécimal.

Solution:

Pour `addu $10, $10, $9`, le format est R et l'instruction génériquee est `addu Rd, Rs, Rt`. Donc `Rd` et `Rs` sont `$10`, `Rt` est `$9`. L'opcode est `SPECIAL` et vaut `0b000000`, le champ `FUNC` vaut `0b100001`. On obtient donc :

Opcod	Rs	Rt	Rd	Sh	Func
000000	01010	01001	01010	00000	100001

Le codage de cette instruction en hexadecimal est `0x01495021`.

Pour `j etiq1` au format J, avec l'opcod qui vaut `000010` et comme l'adresse cible est 0x0040000c, les bits 26 à 2 sont `00 0100 0000 0000 0000 0000 11`.

On obtient :

0pcod	Imd26
000010	00 0001 0000 0000 0000 0000 0011

Le codage de l'instruction en hexadécimal est `0x08100003`.

Qu'affiche le programme avant de terminer ? Que calcule-t-il ?

Solution:

Le programme affiche la valeur 6

Exercice 4 : Nombre de mots dans une phrase – 10 points

On souhaite écrire un programme assembleur MIPS permettant d'afficher le nombre de mots d'une phrase, saisie au clavier.

Par exemple, si la phrase saisie est : `" il fait beau et chaud."`, le programme affiche `5`.

Le décompte des mots est réalisé en identifiant les séparations des mots dans la phrase. La complexité du programme augmente en considérant différents séparateurs et caractères typographiques (espace, apostrophe, virgule, point, ...). On s'intéresse ici uniquement à la séparation de mots par le caractère espace.

Question 4.1 : 1 points

Donner le contenu de la section données du programme assembleur : il doit permettre d'allouer dans le segment de données une zone mémoire pour le stockage la phrase saisie au clavier. L'adresse de cette zone sera repérée par l'étiquette `phrase`. La zone mémoire allouée comprendra 256 octets.

Solution:

```
.data
phrase: .space 256
```

Question 4.2 : 2 points

Donner le contenu de la section de code du programme, qui (pour l'instant) contient un programme principal MIPS permettant la saisie au clavier d'une phrase et son stockage dans la zone mémoire d'étiquette `phrase`. Le programme comprend ensuite les instructions de terminaison.

Solution:

```
.text
# saisie chaine caracteres et stockage à l'adresse 0x10010000
lui $4, 0x1001
ori $5, $0, 256
ori $2, $0, 8
```

```
syscall


ori $2, $0, 10
syscall
```

On considère maintenant que la phrase saisie est *bien formée* : la phrase comprend au moins un mot et commence par un mot. Les mots sont séparés par **exactement** un caractère espace, et la phrase se termine par un caractère point, collé au dernier mot de la phrase.

Remarque : le codage ASCII du caractère espace est `0x20` et celui du caractère point est `0x2E`.

Question 4.3 : 7 points

Donner une suite d'instructions assembleur permettant de compter le nombre de mots de la phrase saisie, et l'affichage de ce nombre.
Remarque : le code demandé est celui à insérer entre la saisie de la phrase et la terminaison du programme donné en réponse à la question précédente, il ne s'agit donc pas du programme complet.
Important : vous commenterez les instructions de votre séquence afin d'expliciter soit le contenu/rôle des registres destinations, soit ce que réalisent les instructions.

Solution:

```
.text
 # saisie chaine caracteres et stockage à l'adresse 0x10010000
 lui $4, 0x1001
 ori $5, $0, 256
```

```
ori $2, $0, 8
syscall

#---- insertion ici ---
lui $4, 0x1001
xor $8, $8, $8
ori $9, $0, 0x2E # caractere point
ori $10, $0, 0x20 # caractere espace

dbtboucle :
lb $11,0($4)
beq $9, $11, finboucle      # caractere point rencontré -> sortie de boucle
bne $10,$11, finiteration   # caractere lu n'est pas espace : saut à fin d'iteration pour passer directe
addiu $8, $8, 1             # caractere lu est espace : incrémente le compteur de mots
finiteration :
addiu $4, $4, 1             # passe au caractere suivant
j dbtboucle



finboucle:
addiu $8, $8, 1             # compte le point final (si initialisation de $8 à 1 et non 0, pas la peine

or $4, $0, $8
ori $2, $0, 1
syscall
#---- fin insertion ---

ori $2, $0, 10
syscall
```

Exercice 5 : Maximum de 3 nombres – 8 points

Question 5.1 : 7 points

Soit le code C suivant, qui affiche le maximum de trois nombres (contenus dans des variables globales) en deux comparaisons et une affectation.

```
int a = 127;
int b = 274;
int c = 89;

void main() {
   int u = a;
   int v = b;
   int w = c;
   int r;
   if (u > v) {
      if (u > w) {
         r = u;
      }
      else {
         r = w;
      }
   }
   else {
      if (v > w) {
         r = v;
      }
      else {
         r = w;
      }
   }
```

```
    printf("%d", r);
    exit();
}
```

Écrivez le code assembleur correspondant au programme ci-dessus **sans optimisation**. On rappelle que sans optimisation des accès mémoire, chaque lecture (resp. écriture) d'une variable globale ou locale doit provoquer une lecture (resp. une écriture).

Vous commentez votre code assembleur afin d'indiquer ce que réalise chaque instruction (lien avec le code C ou convention MIPS)

Solution:

```
.data
  a: .word 127
  b: .word 274
  c: .word 89

.text
  # Prologue
  addiu $29, $29, -16
  # Corps
  lui   $8, 0x1001
  lw    $9, 0($8)           # lecture a
  sw    $9, 0($29)          # u <- a
  lw    $9, 4($8)           # lecture b
  sw    $9, 4($29)          # v <- b
  lw    $9, 8($8)           # lecture c
  sw    $9, 12($29)         # w <- c

  lw    $8, 0($29)          # lecture u
  lw    $9, 4($29)          # lecture v
  slt   $10, $9, $8         # v < u
  beq   $10, $0, _else_ext
  lw    $8, 0($29)          # lecture u
  lw    $9, 8($29)          # lecture w
  slt   $10, $9, $8         # w < u
  beq   $10, $0, _else_int_0
  lw    $8, 0($29)          # lecture u
  sw    $8, 12($29)         # r <- u
  j     _end_if_int_0
_else_int_0:
  lw    $8, 8($29)          # lecture w
  sw    $8, 12($29)         # r <- w
_end_if_int_0:
```

```
  j     _end_if_ext
_else_ext:
  lw    $8, 4($29)            # lecture v
  lw    $9, 8($29)            # lecture w
  slt   $10, $9, $8           # w < v
  beq   $10, $0, _else_int_1
  lw    $8, 4($29)            # lecture v
  sw    $8, 12($29)           # r <- v
  j     _end_if_int_1
_else_int_1:
  lw    $8, 8($29)            # lecture w
  sw    $8, 12($29)           # r <- w
_end_if_int_1:
_end_if_ext:
  # printf
  lw    $4, 12($29)
  ori   $2, $0, 1
  syscall
  # Epilogue
  addiu $29, $29, 16
  ori $2, $0, 10
  syscall
```