

Numéro d'anonymat (donné sur votre étiquette)

Examen 2023 – 2024
Architecture des ordinateurs 1 – LU3IN029
Durée : 2h00

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.

L'examen est composé de 5 exercices indépendants.

- Exercice 1 - 8 points : Circuit Booléen - Distance de Hamming – (p. 1)
- Exercice 2 - 6 points : Instructions assembleur et petit programme – (p. 4)
- Exercice 3 - 6 points : Implantation de données et instructions en mémoire – (p. 6)
- Exercice 4 - 16 points : Programmation Assembleur – (p. 9)
- Exercice 5 - 15 points : Architecture et programmation système – (p. 9)

Exercice 1 : Circuit Booléen - Distance de Hamming – 8 points

La distance de Hamming DH de deux mots de n bits est un nombre qui est égal au nombre de bits qui diffèrent entre les 2 mots. En d'autres termes, il s'agit du nombre de bits à 1 du résultat du xor des 2 mots. Par exemple sur 8 bits, $DH(0xF0, 0xA3) = 4$.

La table de vérité suivante définit les valeurs des 2 sorties s_1, s_0 , qui encodent l'entier définissant la distance de Hamming des 2 mots (sur deux bits) a_1a_0 et b_1b_0 .

Question 1.1 : 2 points

Complétez les tables de vérité des sorties s_1 et s_0 .

a_1	a_0	b_1	b_0	s_1	s_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Solution:

a_1	a_0	b_1	b_0	s_1	s_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	0

Question 1.2 : 2 points

Donner les formes normales disjonctives de s_0 et s_1 .

Solution:

$$s_0 = \overline{a_1} \cdot \overline{a_0} \cdot \overline{b_1} \cdot b_0 + \overline{a_1} \cdot \overline{a_0} \cdot b_1 \cdot \overline{b_0} + \overline{a_1} \cdot a_0 \cdot \overline{b_1} \cdot \overline{b_0} + \overline{a_1} \cdot a_0 \cdot b_1 \cdot b_0 + a_1 \cdot \overline{a_0} \cdot \overline{b_1} \cdot \overline{b_0} + a_1 \cdot \overline{a_0} \cdot b_1 \cdot b_0 + a_1 \cdot a_0 \cdot \overline{b_1} \cdot b_0 + a_1 \cdot a_0 \cdot b_1 \cdot \overline{b_0}$$

$$s_1 = \overline{a_1} \cdot a_0 \cdot b_1 \cdot \overline{b_0} + a_1 \cdot \overline{a_0} \cdot \overline{b_1} \cdot b_0 + a_1 \cdot a_0 \cdot \overline{b_1} \cdot \overline{b_0} + \overline{a_1} \cdot \overline{a_0} \cdot b_1 \cdot b_0$$

Question 1.3 : 2 points

Montrez que $s_0 = (a_0 \oplus b_0) \oplus (a_1 \oplus b_1)$. Vous procéderez par factorisation de la forme normale disjonctive de s_0 et ferez apparaître des formes XOR (\oplus) et de XNOR ($\overline{\oplus}$).

Solution:

$$s_0 = \overline{a_1} \cdot \overline{a_0} \cdot \overline{b_1} \cdot b_0 + \overline{a_1} \cdot \overline{a_0} \cdot b_1 \cdot \overline{b_0} + \overline{a_1} \cdot a_0 \cdot \overline{b_1} \cdot \overline{b_0} + \overline{a_1} \cdot a_0 \cdot b_1 \cdot b_0 + a_1 \cdot \overline{a_0} \cdot \overline{b_1} \cdot \overline{b_0} + a_1 \cdot \overline{a_0} \cdot b_1 \cdot b_0 + a_1 \cdot a_0 \cdot \overline{b_1} \cdot b_0 + a_1 \cdot a_0 \cdot b_1 \cdot \overline{b_0}$$

$$s_0 = \overline{a_0} \cdot \overline{b_0} \cdot (\overline{a_1} \cdot b_1 + a_1 \cdot \overline{b_1}) + \overline{a_0} \cdot b_0 \cdot (\overline{a_1} \cdot \overline{b_1} + a_1 \cdot b_1) + a_0 \cdot \overline{b_0} \cdot (\overline{a_1} \cdot \overline{b_1} + a_1 \cdot b_1) + a_0 \cdot b_0 \cdot (\overline{a_1} \cdot b_1 + a_1 \cdot \overline{b_1})$$

$$s_0 = \overline{a_0} \cdot \overline{b_0} \cdot (a_1 \oplus b_1) + \overline{a_0} \cdot b_0 \cdot (\overline{a_1} \oplus \overline{b_1}) + a_0 \cdot \overline{b_0} \cdot (\overline{a_1} \oplus b_1) + a_0 \cdot b_0 \cdot (a_1 \oplus b_1)$$

$$s_0 = (\overline{a_0} \cdot \overline{b_0} + a_0 \cdot b_0) \cdot (a_1 \oplus b_1) + (\overline{a_0} \cdot b_0 + a_0 \cdot \overline{b_0}) \cdot (\overline{a_1} \oplus b_1)$$

$$s_0 = (\overline{a_0} \oplus b_0) \cdot (a_1 \oplus b_1) + (a_0 \oplus b_0) \cdot (\overline{a_1} \oplus b_1)$$

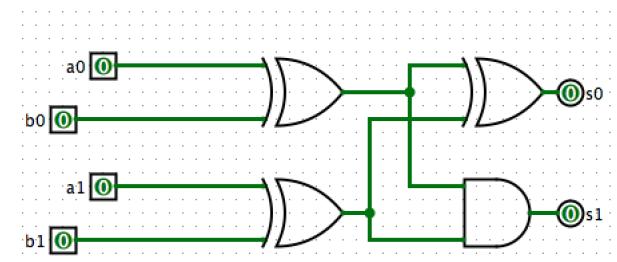
$$s_0 = (a_0 \oplus b_0) \oplus (a_1 \oplus b_1)$$

Pas demandé, mais par un même raisonnement :
 $s_1 = (a_0 \oplus b_0).(a_1 \oplus b_1)$

Question 1.4 : 2 points

On suppose de plus que $s_1 = (a_0 \oplus b_0).(a_1 \oplus b_1)$. Dessinez le schéma correspondant au circuit sous forme de portes logiques AND, OR, XOR, XNOR à deux entrées.

Solution:



Exercice 2 : Instructions assemeleur et petit programme – 6 points

Question 2.1 : 3 points

Compléter en hexadécimal (sur 32 bits) les valeurs contenues dans les registres après l'exécution des instructions suivantes :

```
.text
    lui    $8,    0x8000      # $8 = _____
    sra    $9,    $8, 16       # $9 = _____
```

```

srl $10, $9, 1      # $10 = _____
sll $11, $10, 12    # $11 = _____
ori $12, $11, 0xBABA # $12 = _____
andi $13, $12, 0xF0F0 # $13 = _____

```

Solution:

```
.data
lui $8, 0x8000      # $8 = 0x80000000
sra $9, $8, 16       # $9 = 0xFFFF8000
srl $10, $9, 1        # $10 = 0x7FFFC000
sll $11, $10, 12      # $11 = 0xFC000000
ori $12, $11, 0xBABA # $12 = 0xFC00BABA
andi $13, $12, 0xF0F0 # $13 = 0x0000B0B0
```

Question 2.2 : 3 points

Soit 0x03020100 le contenu du registre \$8. Écrire un programme assembleur MIPS, utilisant UNIQUEMENT des instructions arithmétiques et logiques, qui met dans \$8 la valeur 0x00020103, c'est-à-dire qui permute l'octet de poids faible avec l'octet de poids fort dans le mot. Par exemple, si initialement \$8 = 0xAB1278FF, en sortie \$8 vaudra 0xFF1278AB. On pourra utiliser les registres \$9 à \$15 pour les calculs intermédiaires.

Solution:

```
.data
x : .word 0xAB1278FF

.text
lui $8, 0x1001
lw $8, 0($8)

ori $9, $0, 0xFF # masque pour sélectionner un octet
and $10, $8, $9 # $10 : octet0
sll $9, $9, 24
and $11, $8, $9 # $11 : octet3

lui $9, 0x00FF
ori $9, $9, 0xFF00
and $8, $8, $9 # $8 : octet2 et octet3 préservés, octet0 et octet3 mis à 0

srl $11, $11, 24 # place l'octet3 en poids faible
sll $10, $10, 24 # place l'octet0 en poids fort
or $9, $10, $11 # $9 : octet0 en poids fort et octet3 en poids faible, 00 milieu
or $8, $8, $9
```

Exercice 3 : Implantation de données et instructions en mémoire – 6 points

On considère l'implantation des instructions suivantes :

Adresse (de mot)	Vue par mot
0x00400000	0x3C081001
0x00400004	0x8D0A0000
0x00400008	0x8D0B0004
0x0040000C	0x014B6021

Question 3.1 : 3 points

Décodez les deux mots binaires implantés aux adresses 0x00400004 et 0x0040000C : détaillez le contenu de chaque champs et sa signification.

Solution:

à l'adresse 0x00400004, instruction 0x8D0A0000 :

- soit en binaire : 0b1000 1101 0000 1010 0000 0000 0000 0000
- CODOP : 0b100011 : mnémonique : LW, donc format I : l'instruction est de la forme lw Rt, Imm(Rs)
- Rs : 0b01000 → \$8
- Rt = 0b01010 → \$10
- Imm16 : 0b 0000 0000 0000 0000 → offset = 0

Il s'agit de l'instruction lw \$10, 0(\$8).

à l'adresse 0x0040000C, instruction 0x014B6021 :

- soit en binaire : 0b0000 0001 0100 1011 0110 0000 0010 0001
- CODOP : 0b000000 : code : SPECIAL → FUNC : 0b100001, donc mnémonique ADDU :
- l'instruction est de la forme addu Rd, Rs, Rt
- Rs : 0b01010 → \$10
- Rt = 0b01011 → \$11
- Rd : 0b01100 → \$12

Il s'agit de l'instruction addu \$12, \$10, \$11.

Question 3.2 : 3 points

Complétez l'implantation des données décrite dans le tableau ci-dessous, qui a été créée par les directives assembleurs suivantes :

```
.data
    a : .half 2, 10
    b : .asciiz "msg"
    c : .byte 0xA, 0xB, 0xC
    d : .word 10, 20, 30
    e : .byte 'o','k'
```

Adresse (de mot)	Vue par octet				Vue par mot
	+ 0	+ 1	+ 2	+3	
0x10010000					
0x10010004					
0x10010008					
0x1001000c					
0x10010010					
0x10010014					
0x10010018					
0x1001001c					

Solution:

Adresse (de mot)	Vue par octet				Vue par mot
	+ 0	+ 1	+ 2	+3	
0x10010000	0x02	0x00	0xA	0x00	0x000A0002
0x10010004	0x6D	0x73	0x67	0x00	0x0067736D
0x10010008	0x0A	0x0B	0x0C	0x00	0x000C0B0A
0x1001000c	0x0A	0x00	0x00	0x00	0x0000000A
0x10010010	0x14	0x00	0x00	0x00	0x00000014
0x10010014	0x1E	0x00	0x00	0x00	0x0000001E
0x10010018	0x6F	0x6B	0x00	0x00	0x00006B6F
0x1001001c					

Exercice 4 : Programmation assembleur – 16 points

On considère le programme C suivant. Il demande la saisie d'une chaîne de caractères au clavier, puis compte le nombre de voyelles de cette chaîne de caractères, puis affiche ce nombre. Tous les caractères ont un code ASCII sur un octet.

```
char str[20];
char set[] = "aeiouy";
int cpt = 0;

int est_voyelle( char c, char set[]){
    int j = 0;

    while( set[j] != 0x00) {
        if ( c == set[j] ) {
            return 1;
        }
        j = j+1;
    }
    return 0;
}

void main(){
    int i = 0, res;

    scanf("%s",str);           //appel système de saisie de chaîne de caractères
    while ( str[i]!=0x00 ) {
        res = est_voyelle(str[i],set);
        if ( res ) {
            cpt = cpt + 1;
        }
        i = i+1;
    }
    printf("%d",cpt);

    exit();
}
```

Question 4.1 : 2 points

Complétez le schéma de la pile ci-dessous, vue après le prologue de la fonction `est_voyelle`.

Adresse (de mot)	contenu (mot)
0xFFFFFFF8	<fond de pile>
0xFFFFFFF4	
0xFFFFFFF0	
0xFFFFFEC	
0xFFFFFE8	
0xFFFFFE4	
0xFFFFFE0	

Solution:

Adresse (de mot)	contenu (mot)
0xFFFFFFF8	<fond de pile>
0xFFFFFFF4	main :res
0xFFFFFFF0	main :i
0xFFFFFEC	main :arg1
0xFFFFFE8	main :arg0
0xFFFFFE4	voy : \$31
0xFFFFFE0	voy :j

et \$29 = 0xFFFFFE4.

Question 4.2 : 6 points

Donner le code assembleur correspondant à la fonction `est_voyelle`. Toute allocation en pile doit être assortie d'un commentaire expliquant la taille allouée. Vous devez respecter les conventions d'allocation en pile et d'appel de fonction. Vous pouvez optimiser votre code, en précisant le rôle des registres que vous utilisez, et vous devez commenter votre code.

Solution:

```
est_voyelle :    # retourne 1 si le caractère passé dans $4 est dans l'ensemble implanté à
partir de l'adresse $5
    addiu $29, $29, -8      # $31 + 1 var loc j, initialisée à 0
    sw $31, 4($29)
    sw $0, 0($29)

    sw $5, 12($29)        # svg $5, modifié dans la boucle (et stable dans le main)

boucle2 :
    lb $10, 0($5)
    beq $10, $0, sortie_ko # tous les éléments parcourus sans avoir trouvé str[i]

    # cmp str[i] et set[j]
    beq $4, $10, sortie_ok

    # itération suivante
```

```
addiu $5, $5, 1
j boucle2

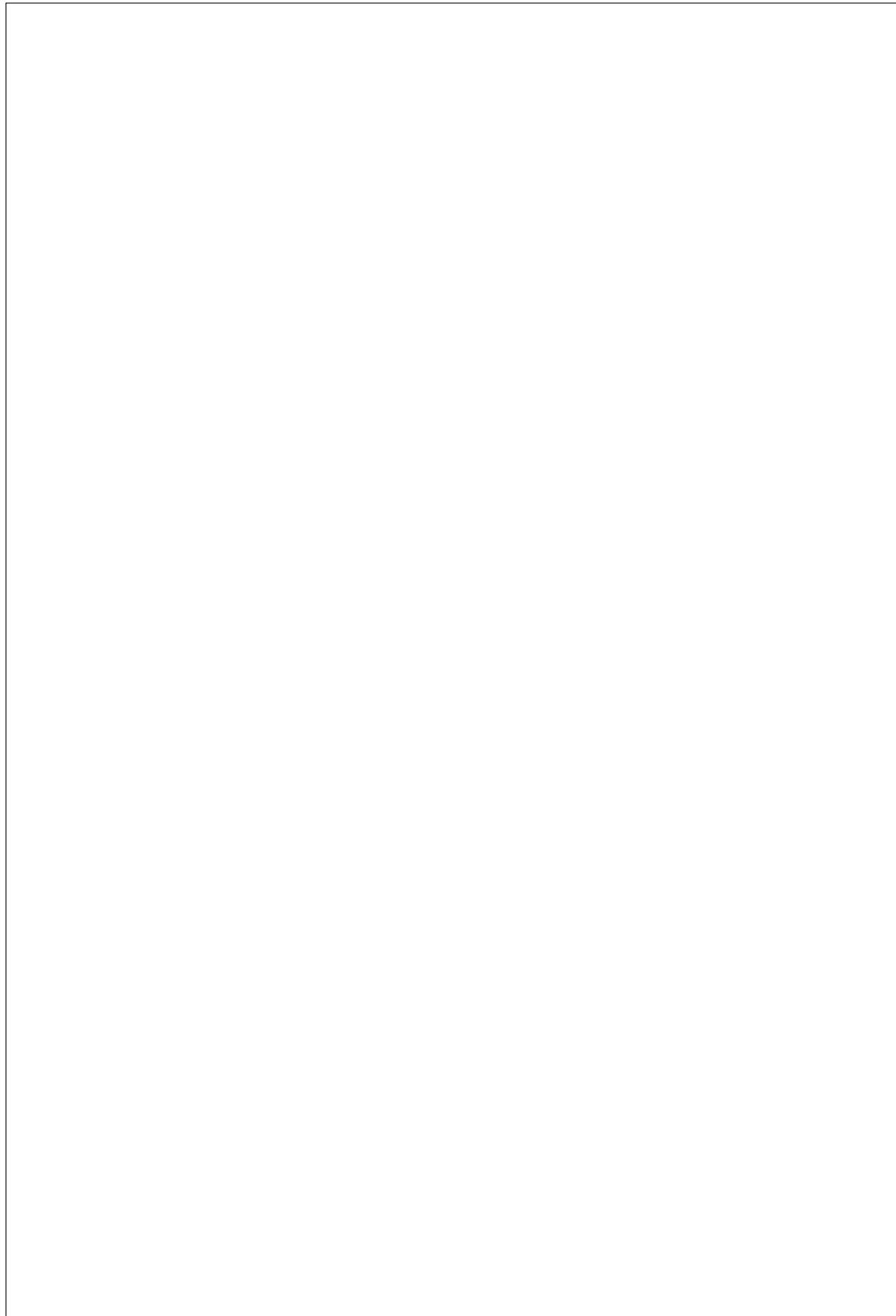
sortie_ko:
    xor $2, $2, $2
    j epilogue

sortie_ok:
    ori $2, $0, 1

epilogue:
    lw $5,12($29)
    lw $31, 4($29)
    addiu $29, $29, 8
    jr $31
```

Question 4.3 : 8 points

Donner le code assembleur correspondant aux déclarations globales et au programme principal. Toute allocation en pile doit être assortie d'un commentaire expliquant la taille allouée. Vous devez respecter les conventions d'allocation en pile et d'appel de fonction. Vous pouvez optimiser votre code, en précisant le rôle des registres que vous utilisez.



Solution:

```
.data
    str    : .space 20
    set   : .ascii "aeiouy"
    .align 5
    cpt    : .word 0

.text
main:
    # pile : 1 var locale i, initialisée à 0 et res, + 2 arg
    addiu $29, $29, -16
    sw $0, 12($29)

    # saisie chaîne, terminée par les caractères 0x0A puis 0x00
    ori $2, $0, 8
    lui $4, 0x1001
    ori $5, $0, 20
    syscall

    lui $18, 0x1001  # &str[0]
    lw $19, 4($29)   # i
    lui $5, 0x1001
    addiu $5, $5, 20 # &set
    lw $15, 32($18)  # cpt

    or $16, $0, $5   # svg de $5 ds registre persistant

boucle:
    # tq str[i] != 0x00
    add $20, $19, $18  # & str[i]
    lb $4, 0($20) # str[i]
    beq $4, $0, apres_boucle

    #appel res = est_voyelle(str[i], set Accent)
    or $5, $0, $16  # restauration de $5 (si modifié dans it précédente de est_voyelle)
    jal est_voyelle  # appel est Accentue

    # if res then ... else ...
    beq $2, $0, fin_boucle # saut si res = 0 (caractère courant est non accentué)

then:
    # cpt = cpt+1
    addiu $15, $15, 1

fin_boucle:
    # i <- i+1 et retour début de boucle
    addiu $19, $19, 1
    j boucle

apres_boucle :
    # affiche cpt
    or $4, $15, $0
    ori $2, $0, 1
    syscall

    # pile et terminaison
    addiu $29, $29, +16
    ori $2, $0, 10
    syscall
```