

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

Partiel 2023 – 2024
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 17 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 : 50 points.

Le partiel est composé d'exercices indépendants.

- Exercice 1 - 8 points : Additions et débordements – (p. 1)
- Exercice 2 - 14 points : Implantation et exécution d'un programme décrit en assembleur – (p. 3)
- Exercice 3 - 15 points : Programmation assembleur – (p. 8)
- Exercice 4 - 13 points : Autour de l'additionneur – (p. 12)

Exercice 1 : Additions et débordements – 8 points

Soit un additionneur matériel sur 8 bits, muni de deux entrées A et B sur 8 bits, d'une sortie résultat S sur 8 bits.

Question 1.1 : 8 points

Dans le tableau suivant, chaque colonne représente une opération d'addition sur les entrées A et B. Compléter ce tableau, colonne par colonne, en trouvant des valeurs des mots binaires A et B pour obtenir des dépassements dans les cas indiqués (entiers naturels et entiers relatifs en codage complément à 2), et calculer la sortie S correspondante. Donner enfin les interprétations de A, B et S en entiers naturels et en entiers relatifs.

A				
B				
S				
Dépassement sur entiers naturels	Non	Non	Oui	Oui
Dépassement sur entiers relatifs	Non	Oui	Non	Oui
Interprétation de A en naturel				
Interprétation de B en naturel				
Interprétation de S en naturel				
Interprétation de A en relatif				
Interprétation de B en relatif				
Interprétation de S en relatif				

Solution:

A	0b00000000	0b01111111	0b11000000	0b10000000
B	0b00000000	0b00000001	0b11000000	0b10000000
S	0b00000000	0b10000000	0b10000000	0b00000000
Dépassement sur entiers naturels	Non	Non	Oui	Oui
Dépassement sur entiers relatifs	Non	Oui	Non	Oui
Interprétation de A en naturel	0	127	192	128
Interprétation de B en naturel	0	1	192	128
Interprétation de S en naturel	0	128	128	0
Interprétation de A en relatif	0	127	-64	-128
Interprétation de B en relatif	0	1	-64	-128
Interprétation de S en relatif	0	-128	-128	0

Exercice 2 : Implantation et exécution d'un programme décrit en assembleur – 14 points

On considère le programme écrit en assembleur MIPS suivant :

```
.data
    z1 : .word 1, 0xAABB
    z2 : .asciiz "abcd"
    z3 : .byte 0x6F, 0x6B, 0x00
    z4 : .half 1, 0xAABB

.text
    lui $8, 0x1001
    lw $10, 0($8)
    lw $11, 4($8)

    addu $12, $10, $11
    sw $12, 0($8)

    lh $20, 18($8)
    lhu $21, 18($8)

    slt $22, $21, $20      # $22 = 1 si $21 < $20
    bgtz $22, neg

    addiu $4, $8, 13
    j suite
neg :   addiu $4, $8, 8
suite : ori $2, $0, 4      # affichage
        syscall

    ori $2, $0, 10        # fin de programme
    syscall
```

Question 2.1 : 5 points

Décrivez le contenu du segment de données associé à ce programme, en complétant le tableau suivant (vue octet et vue mot).

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

Solution:

Adresse (de mot)	Vue par octet				Vue par mot
	+ 0	+ 1	+ 2	+3	
0x10010000	0x01	0x00	0x00	0x00	0x00000001
0x10010004	0xBB	0xAA	0x00	0x00	0x0000AABB
0x10010008	0x61	0x62	0x63	0x64	0x64636261
0x1001000c	0x00	0x6F	0x6B	0x00	0x006B6F00
0x10010010	0x01	0x00	0xBB	0xAA	0xAABB0001
0x10010014					

Question 2.2 : 3 points

Donnez le contenu en hexadécimal des registres \$10, \$11, \$20, \$21 et \$22 après l'exécution de ce programme.

Solution:

```
$8 = 0x10010000
$10 : 0x00000001
$11 : 0x0000AABB
$20 : 0xFFFFAABB
$21 : 0x0000AABB
$22 : 0x00000000
```

Question 2.3 : 1 point

Lors de l'exécution de l'instruction `bgt z`, le saut sera-t-il pris ? Justifiez votre réponse.

Solution:

non le saut ne sera pas pris car \$22 = 0 (\$21 représente un nombre positif alors que \$20 représente un nombre négatif, donc la condition \$21 < \$20 n'est pas réalisée).

Question 2.4 : 1 point

Quel sera le message affiché par le programme ?

Solution:

Le programme affichera "ok" (affichage des octets à partir de l'adresse 0x1001000D, interprétés comme des codes ascii).

L'implantation du segment de code est partiellement donnée ci-dessous :

Adresse (de mot)	Contenu du mot	Instruction desassemblée
0x00400000	0x3C081001	lui \$8, 0x1001
0x00400004	0x8D0A0000	lw \$10, 0(\$8)
0x00400008	0x8D0B0004	lw \$11, 4(\$8)
0x0040000C	0x014B6021	addu \$12, \$10, \$11
0x00400010		sw \$12, 0(\$8)
0x00400014		lh \$20, 18(\$8)
0x00400018		lhu \$21, 18(\$8)
0x0040001C		slt \$22, \$21, \$20
0x00400020	0x1EC00002	bgtz \$22, 0x00000002
0x00400024		addiu, \$4, \$8, 0x0D
0x00400028		j 0x0040030
0x0040002C		addiu \$4, \$8, 8
0x00400030		ori \$2, \$0, 4
0x00400034	0x0000000C	syscall
0x00400038	0x3402000A	ori \$2, \$0, 10
0x0040003C	0x0000000C	syscall
0x00400040	0x????????	<hors du programme>
0x00400044		

Question 2.5 : 2 points

Donnez le codage en hexadécimal de l'instruction `slt $22, $21, $20`, implantée à l'adresse 0x0040001C.

Solution:

L'instruction `slt $22, $21, $20` est codée selon le format R, comme suit :

La valeur du CODOP est SPECIAL soit `0b000000` et FUNC vaut `0b101010`

La valeur de Rs est 21 soit `0b10101`

La valeur de Rt et 20 soit `0b10100`

La valeur de Rd est 22 soit `0b10110`

La valeur de Sh est 0 soit `0b000000`

Le codage binaire est donc :

OPCOD	RS	RT	RD	SH	FUNC
000000	10101	10100	10110	00000	101010.

En regroupant par quartet : `0000 0010 1011 0100 1011 0000 0010 1010`

En hexadécimal, cela donne : `0x02B4B02A`

Question 2.6 : 2 points

Le codage hexadécimal associé à l'instruction `bgtz $22, neg` est `0x1EC00002`, implantée à l'adresse `0x00400020`. Expliquez le codage de cette instruction, et notamment la valeur du champ immédiat.

Solution:

L'instruction `bgtz $22, neg` est codée selon le format I, donc le champ immédiat est sur 16 bits.

Lorsque le saut est pris, l'exécution de cette instruction modifie le compteur de programme PC de telle sorte que : $PC_{\text{cible}} = PC + 4 + (Imm_{16} * 4)$

avec $PC = 0x00400020$

et $PC_{\text{cible}} = 0x0040002C$ (adresse de l'instruction cible du saut, soit l'instruction `addi $4, $8, 8`).

ainsi, $(Imm_{16} * 4) = PC_{\text{cible}} - PC - 4 = 0x0040002C - 0x00400020 - 4 = 8$ et $Imm_{16} = 8/4 = 2$ sur 16 bits, soit `0x0002`.

Exercice 3 : Programmation assembleur – 15 points

Soit le code C suivant, qui étant donné un tableau d'entiers relatifs, comprenant uniquement des valeurs positives ou nulles, et terminé par la valeur -1, 1) parcourt les éléments du tableau et compte le nombre d'éléments compris dans l'intervalle [20..40], et 2) affiche ce nombre.

```
int tab[] = {10, 5, 23, 18, 12, 17, 36, 99, 120, 17, -1};
int c = 0;

void main() {
    int i = 0;

    while (tab[i] != -1) {
        if ((tab[i] >= 20) && (tab[i] <= 40)) {
            c = c + 1;
        }
        i += 1;
    }
    printf("%d", c);      // affichage entier

    exit();
}
```

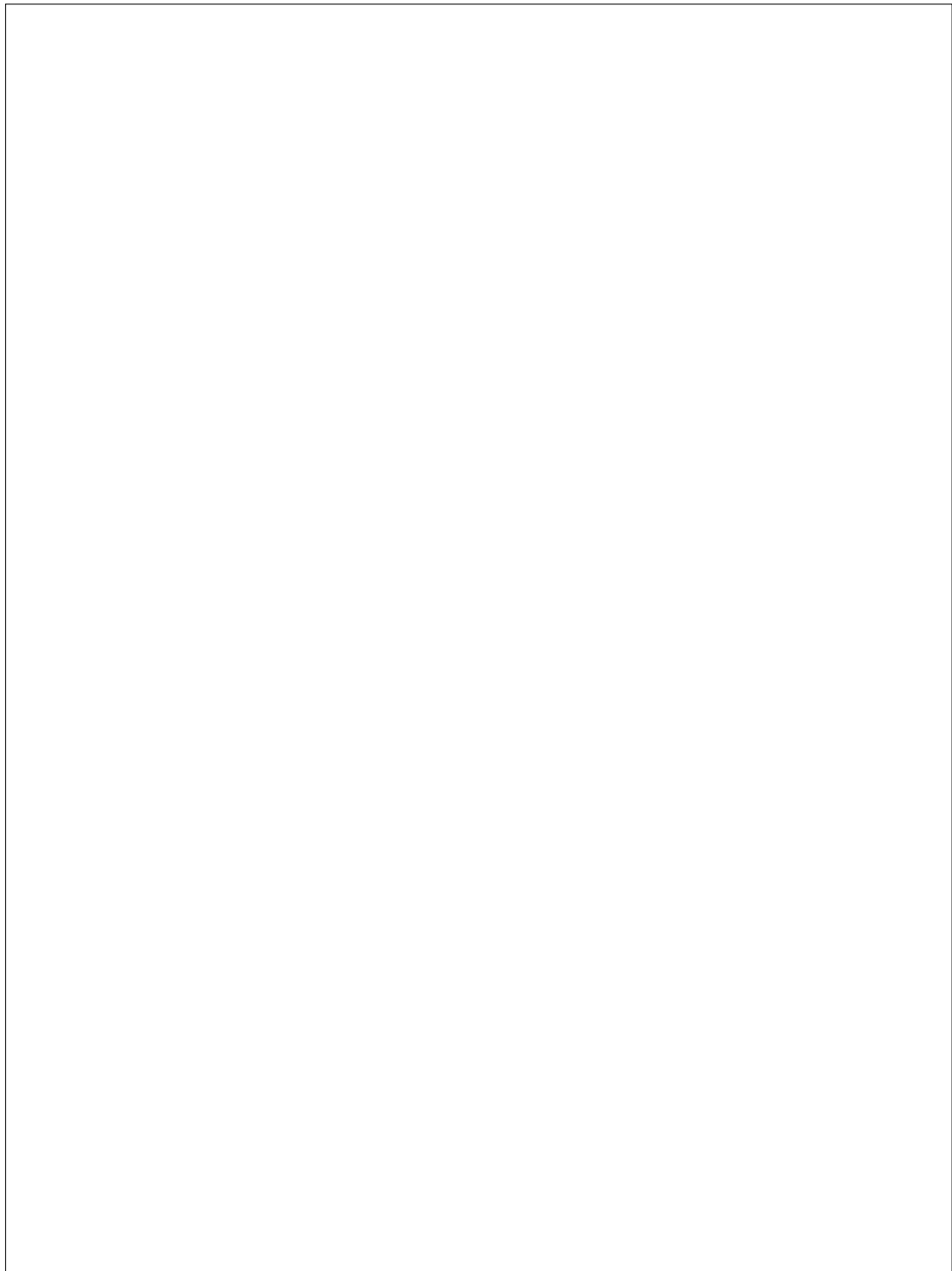
L'exécution du programme précédent produit :

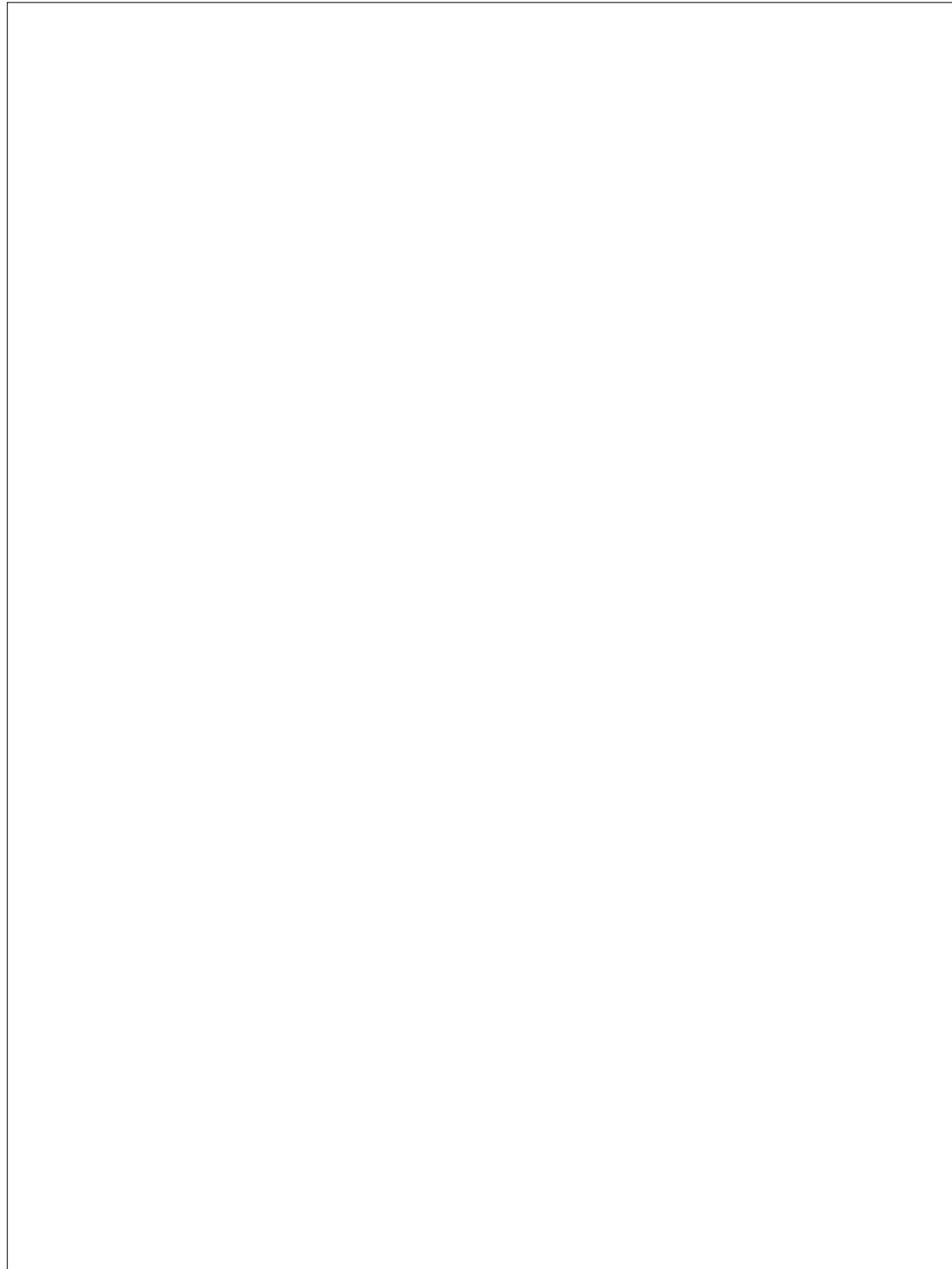
2

Question 3.1 : 15 points

Donner le code assembleur correspondant au programme ci-dessus, en respectant fidèlement la structure du programme C en assembleur (zones allouées, structure de boucle, tests internes, affichage, terminaison). Tous les lectures et écritures de variables (locales ou globales) dans le code source doivent se traduire en assembleur par un accès mémoire. Vous devez commenter un minimum votre code assembleur afin d'indiquer ce que réalise chaque instruction (lien avec le code C ou convention MIPS).

Tout code illisible ou non commenté recevra la note 0.





Solution:

`.data`

```

tab: .word 10, 5, 23, 18, 12, 17, 36, 99, 120, 17, -1
c: .word 0

.text

main:
    addiu $29, $29, 4
    sw     $0, 0($29)      # i <- 0

    addiu $9, $0, -1
    addiu $11, $0, 40
    lui   $10, 0x1001      # &tab[0]
    lui   $12, 0x1001
    ori   $12, $12, 0x2c # &c

while:
    # while (tab[i] != -1)
    lw     $8, 0($29)      # i
    sll    $8, $8, 2       # i * 4
    addu   $8, $8, $10      # &tab[i]
    lw     $8, 0($8)       # tab[i]
    beq    $8, $9, fin_while

    # if
    lw     $8, 0($29)      # i
    sll    $8, $8, 2       # i * 4
    addu   $8, $8, $10      # &tab[i]
    lw     $8, 0($8)       # tab[i]
    slti   $8, $8, 20      # tab[i] < 20 ?
    bne    $8, $0, fin_if

    lw     $8, 0($29)      # i
    sll    $8, $8, 2       # i * 4
    addu   $8, $8, $10      # &tab[i]
    lw     $8, 0($8)       # tab[i]
    slt    $8, $11, $8      # 40 < tab[i] ?
    bne    $8, $0, fin_if

    # c = c + 1
    lw     $8, 0($12)      # c
    addiu  $8, $8, 1
    sw     $8, 0($12)      # c

fin_if:
    # i += 1
    lw     $8, 0($29)      # i
    addiu  $8, $8, 1
    sw     $8, 0($29)      # i

    j while

fin_while:
    lw     $4, 0($12)      # c
    ori    $2, $0, 1
    syscall

    addiu  $29, $29, 4

    ori    $2, $0, 10
    syscall

```

Exercice 4 : Autour de l'additionneur – 13 points

On considère le circuit logique réalisant l'addition de deux mots binaires sur 16 bits. Le circuit prend en entrée deux mots binaires $A = a_{15}..a_0$ et $B = b_{15}..b_0$; il produit le mot représentant la somme $S = s_{15}..s_0$ et la retenue sortante $cout_{15}$. Cet additionneur est réalisé par la mise en série de 16 additionneurs sommant des opérandes et une retenue entrante de taille 1 bit ; l'additionneur 1 bit de rang i somme les bits a_i , b_i et la retenue entrante cin_i ; il produit le bit de somme s_i et le bit de retenue $cout_i$, définis tels que : $s_i = a_i \oplus b_i \oplus cin_i$ et $cout_i = a_i.b_i + (a_i \oplus b_i).cin_i$ (\oplus désigne le OU exclusif).

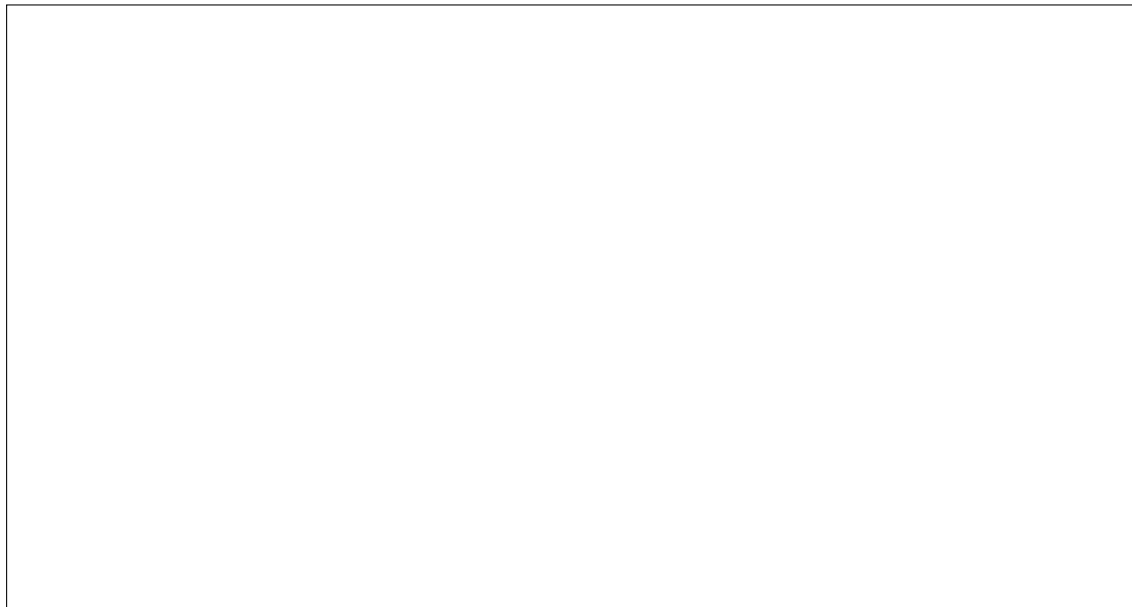
Le nombre de portes logiques traversées entre les entrées et les sorties du circuit conditionne le délai minimal de stabilisation des sorties. Le *chemin critique* d'un circuit est le chemin entre les entrées du circuit et une sortie qui comprend le plus grand nombre de portes logiques élémentaires. C'est ce chemin qui est la contrainte principale pour l'établissement de la valeur du signal de sortie.

Dans la suite, on cherche à déterminer le plus long chemin de l'additionneur 16 bits, puis à construire un additionneur plus rapide dans certains cas.

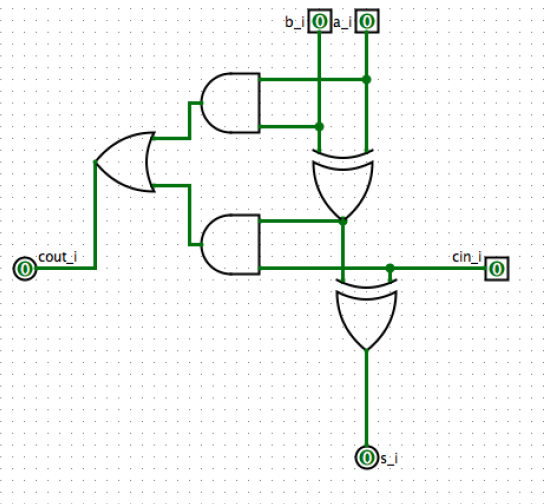
Dans tout cet exercice, les portes logiques élémentaires utilisées ont au maximum deux entrées (AND, OR, XOR, NOT).

Question 4.1 : 2 points

Représentez le schéma de l'additionneur 1 bit et déterminez le chemin critique de ce circuit.



Solution:



Le chemin critique est celui allant de a_i ou b_i vers $cout_i$, qui traverse 3 portes.

Question 4.2 : 1 point

Déterminez le chemin critique de l'additionneur 16 bits. Quel est le nombre de portes traversées sur ce chemin ?

Solution:

Le chemin allant de a_0 ou b_0 vers $cout_{15}$, qui traverse 2 portes pour les additionneurs des bits de rang 1 à 15 (chemin cin_i vers $cout_i$, et 3 portes pour l'additionneur de rang 0 (de a_0 ou b_0 vers $cout_0$, soit au total 33 portes.

On s'intéresse maintenant à la réalisation d'un additionneur plus rapide, dans lequel la propagation de retenue peut être anticipée, lorsque certaines conditions sont réalisées.

Question 4.3 : 1 point

Déterminez la valeur de $cout_i$ lorsque $a_i \neq b_i$. Montrez que si l'on somme deux mots de 8 bits $X = x_7..x_0$ et $Y = y_7..y_0$ tels que pour tout i : $x_i \neq y_i$, alors $cout_7 = cin_0$.

Solution:

Si $a_i \neq b_i$, alors $cout_i = cin_i$
 Si pour tout i , $x_i \neq y_i$, alors chaque rang de l'additionneur propage sa retenue entrante en sortie, et de proche en proche, $cout_7 = cin_0$.

On réalise un circuit PROPAGATE_8, prenant en entrée 2 mots de 8 bits X et Y , et dont la sortie s est définie telle que :

- Si pour tout $i \in [0..7]$, $x_i \neq y_i$, alors $s = 1$
- Sinon : $s = 0$

Question 4.4 : 1 point

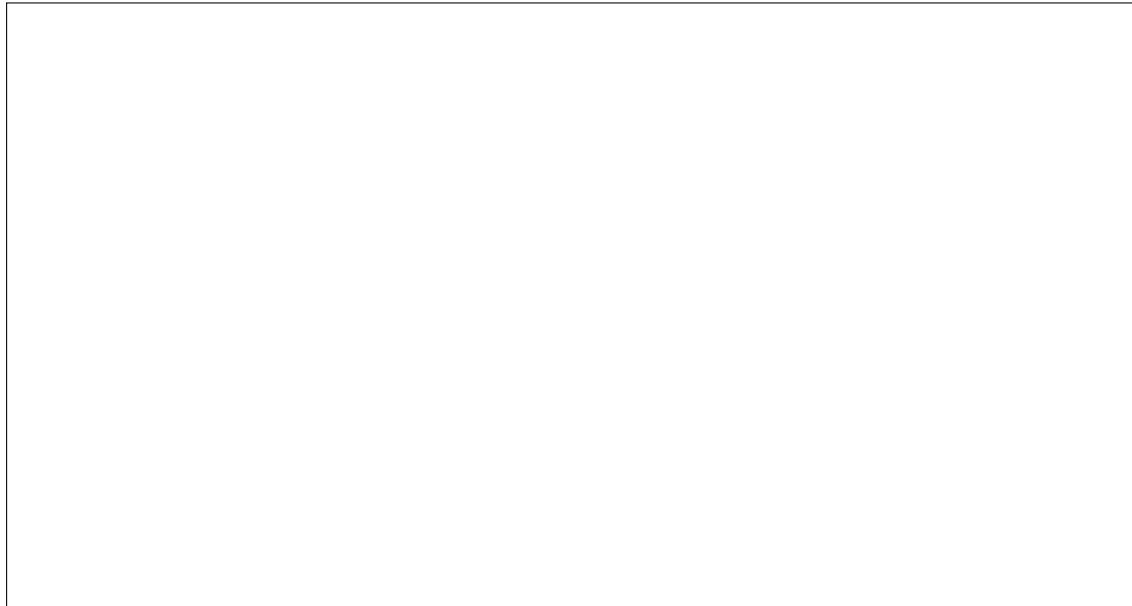
Donnez une expression définissant la valeur de la sortie s en fonction des entrées x_i et y_i pour tout $i \in [0..7]$, en faisant apparaître des opérations \oplus ("ou exclusif", XOR).

Solution:

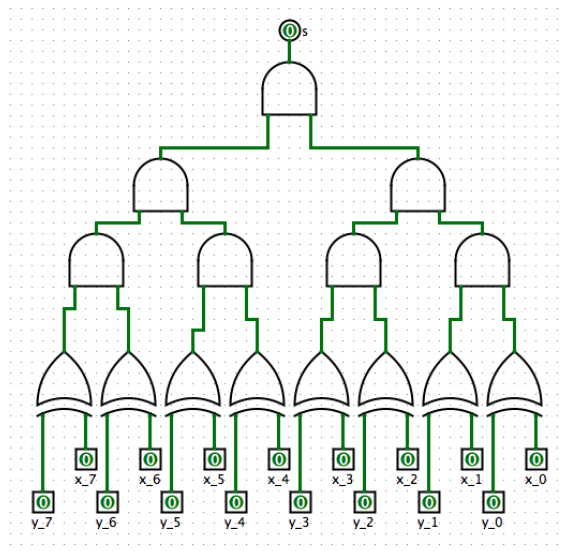
$$s = (x_7 \oplus y_7) \cdot (x_6 \oplus y_6) \cdot (x_5 \oplus y_5) \cdot (x_4 \oplus y_4) \cdot (x_3 \oplus y_3) \cdot (x_2 \oplus y_2) \cdot (x_1 \oplus y_1) \cdot (x_0 \oplus y_0)$$

Question 4.5 : 2 points

Réalisez le schéma du circuit PROPAGATE_8, comprenant uniquement des portes XOR à 2 entrées et des portes AND à deux entrées, *et minimisant la longueur des chemins entre les entrées et la sortie*. Quel est le nombre maximal de portes traversées entre les entrées et la sortie ?



Solution:



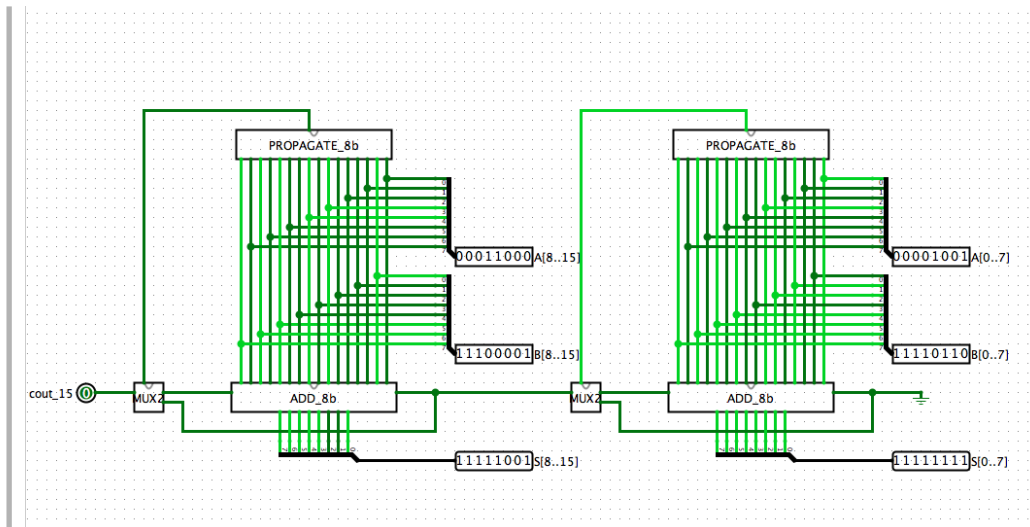
le chemin le plus long comprend 1 xor et 3 and, donc 4 portes (en fait, ils sont tous de même longueur).

On construit maintenant un additionneur 16 bits en mettant en série 2 additionneurs 8 bits, chacun étant muni d'un circuit PROPAGATE_8, qui permet de propager rapidement la retenue entrante en sortie lorsque les mots à sommer satisfont la condition pour tout $i \in [0..7], x_i \neq y_i$.

Le schéma suivant présente les deux étages de cet additionneur, avec les blocs ADD_8 et PROPAGATE_8 pour les bits de rangs [0..7] et de rangs [8..15]. L'interconnexion entre les étages [0..7] et [8..15] est à compléter. On appelle ADD_SKIP_16 ce circuit.

Question 4.6 : 4 points

Complétez le schéma du nouvel additionneur afin d'intégrer le mécanisme de propagation rapide de retenue entre les additionneurs de rangs [0..7] et [8..15], et vers l'additionneur suivant, en décrivant le contenu des



On utilise le circuit ADD_SKIP_16 pour sommer les mots $A = 0b\ 0000\ 1000\ 1110\ 0111$ et $B = 0b\ 0111\ 0111\ 0001\ 1000$,

Question 4.7 : 2 points

Déterminez les valeurs des sorties des circuits PROPAGATE_8 pour les rangs [0..7] et [8..15]. L'exécution de la somme a-t-elle été plus rapide que sur l'additionneur série initial ?

Solution:

les mots $A[0..7]$ et $B[0..7]$ ont bien des bits de même rangs complémentaires pour tous les rangs, donc $PROPAGATE_8\ [0..7] = 1$, et donc cin_8 sera positionnée par le circuit PROPAGATE (+ le multiplexeur), ce qui représente un chemin de $4 + 2 = 6$ portes au lieu de 16, donc le résultat final sera stabilisé plus rapidement que sans mécanisme de saut.