3TC31 (ex. INF107)

Examen intermédiaire (première partie)

2025–2026

Nom : .....

Prénom : .....

Instructions
— Durée : 30 minutes
— Document autorisé : 1 feuille A4 recto-verso uniquement
— Dispositifs électroniques (calculatrice, ordinateur...) interdits
— Vous pouvez répondre en français ou en anglais
— Ce sujet contient quelques questions de cours et deux exercices indépendants
— **Répondez directement sur le sujet**

Questions de cours (4 points / 5 minutes)

Question 1 (2 points)

Quelle est la valeur, en décimal, du nombre représenté en complément à 2 sur 4 bits par la valeur 1111 ?

Réponse :

Question 2 (2 points)

Expliquez le fonctionnement d'une bascule D (entrée `D`, sortie `Q`, horloge `clk`).

Réponse :

Exercice 1 : Logique combinatoire (6 points / 10 minutes)

Nous cherchons à construire un opérateur prenant en entrée deux nombres A et B **non signés** représentés chacun sur n bits, et ayant une sortie S sur n bits telle que $S = A$ si $A \geqslant B$, sinon $S = B$ (autrement dit, cet opérateur renvoie la plus grande des deux entrées).

Pour comparer deux nombres non signés A et B (et donc renvoyer le plus grand des deux), il faut tout d'abord comparer leurs bits de poids fort (A_{n-1} et B_{n-1}). S'ils sont différents, on peut déterminer le nombre le plus grand. S'ils sont identiques, il faut répeter l'opération sur les bits suivants A_{n-2} et B_{n-2} et ainsi de suite.

Comme pour l'addition à propagation de retenue vue en cours, il est donc possible de réaliser cet opérateur en chaînant n opérateurs élémentaires (cf. figure 1). L'opérateur élémentaire numéro i (avec $0 \leqslant i < n$) prend en entrée les bits i de A et B et produit le bit i de S.

L'opérateur élémentaire numéro i reçoit en plus en entrée depuis l'opérateur $i + 1$ les signaux :
— $E_{A>B}$: 1 si la comparaison des bits de poids forts (de $n - 1$ à $i + 1$) a déjà permis de déterminer que $A > B$, 0 dans le cas contraire
— $E_{A<B}$: 1 si la comparaison des bits de poids forts (de $n - 1$ à $i + 1$) a déjà permis de déterminer que $A < B$, 0 dans le cas contraire

Enfin, l'opérateur élémentaire numéro i produit les sorties suivantes à destination de l'opérateur élémentaire $i-1$:

— $S_{A>B}$: 1 si la comparaison des bits de poids forts (de $n-1$ à i) a déjà permis de déterminer que $A > B$, 0 dans le cas contraire

— $S_{A<B}$: 1 si la comparaison des bits de poids forts (de $n-1$ à i) a déjà permis de déterminer que $A < B$, 0 dans le cas contraire

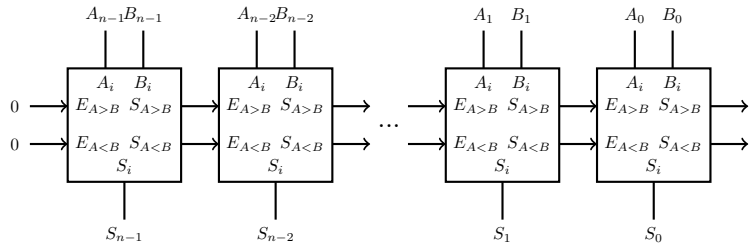

FIGURE 1 – Opérateur complet à partir des opérateurs élémentaires

Question 3 (3 points)

Complétez la table de vérité pour l'opérateur élémentaire sur 1 bit. Comme le cas $E_{A>B} = E_{A<B} = 1$ n'est pas censé arriver, on ne le représentera pas dans le tableau (vous pourrez par la suite donner les valeurs que vous voulez aux sorties dans ce cas).

$E_{A>B}$	$E_{A<B}$	A_i	B_i	S_i	$S_{A>B}$	$S_{A<B}$
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
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			

Question 4 (3 points)

Donnez les équations logiques des sorties S_i, $S_{A>B}$ et $S_{A<B}$ en fonction des entrées $E_{A>B}$, $E_{A<B}$, A_i et B_i (et des opérateurs logiques élémentaires vus en cours : $+$ pour le OU logique, $\cdot$ pour le ET logique, $\oplus$ pour le OU exclusif, $\overline{e}$ pour le complément de e).

$S_i =$

$S_{A>B} =$

$S_{A<B} =$

Exercice 2 : Processeur RISC-V (10 points / 15 minutes)

La figure 2 rappelle le chemin de données (*data path*) de l'implémentation vue en cours du jeu d'instructions de base RISC-V permettant de traiter les instructions registre-vers-registre,

les instructions immédiates, les chargements (*loads*) et stockages (*stores*). Pour simplifier, la partie traitement des instructions de saut n'est pas représentée.

Vous trouverez également en annexe (voir dernière page du sujet) la description de quelques instructions.

Question 5 (8 points)

On suppose que la mémoire d'instructions (*instruction memory*) contient deux instructions :

Adresse	Instruction	Code machine
0	`addi x3, x2, 2`	`0x00210193`
4	`sw x4, 2(x3)`	`0x0041A123`

On suppose que la mémoire de données (*data memory*) contient les mots de 32 bits suivants, représentés en décimal :

Adresse	Donnée
0	10
4	20
8	40
12	80

Enfin, on suppose que les registres contiennent les valeurs suivantes, représentées en décimal :

Registre	Valeur
0	0
1	4
2	8
3	12
4	16

Compléter les deux premières lignes du tableau de la page 5 avec les valeurs des différents signaux internes du processeur pendant l'exécution des deux instructions. Vous pouvez utiliser les symboles +, −… pour représenter l'opération effectuée par l'ALU (`op`). Si la valeur d'un signal n'a pas d'impact (et que vous en êtes sûr), vous pouvez inscrire `x`. Si vous faites une erreur, n'hésitez pas à barrer proprement et à utiliser les lignes supplémentaires du tableau.

Question 6 (2 points)

Écrivez le code assembleur équivalent à l'instruction C : `i = i + 1;`
Pour cette question, on supposera que :
— `i` est une variable entière sur 32 bits
— `i` est stockée dans la mémoire de données à l'adresse `20` (en base 10)
— Le reste du contenu de la mémoire de données est inconnu
— L'état initial des registres est inconnu (à l'exception de `r0` qui vaut toujours 0)
— Vous pouvez utiliser tous les registres (leurs valeurs initiales n'ont pas besoin d'être conservées)

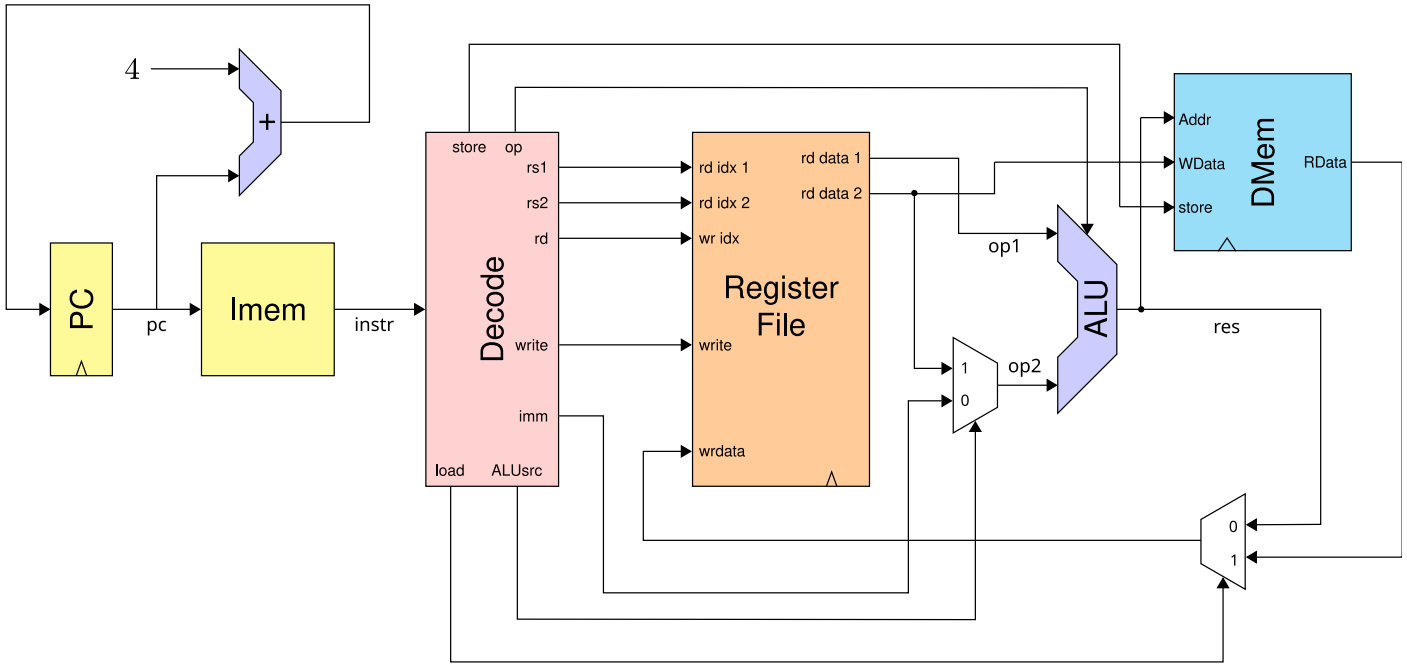

FIGURE 2 – RISC-V data path pour les instructions de type R, immédiates et mémoires

pc	instr	rs1	rs2	rd	op	ALUsrc	imm	op1	op2	res / Addr	WData	RData	write	load	store	wrdata
0	0x00210193															
4	0x0041A123															

ध

Annexe : Instructions RISC-V (liste non exhaustive)

add (addition)

Format `add rd, rs1, rs2`

Description Additionne le contenu du registre `rs2` au contenu du registre `rs1` et stocke le résultat dans le registre `rd`.

addi (immediate addition)

Format `addi rd, rs1, imm`

Description Additionne le contenu du registre `rs1` avec la valeur immédiate `imm` (étendue de manière signée sur 32 bits) et stocke le résultat dans le registre `rd`.

sub (subtraction)

Format `sub rd, rs1, rs2`

Description Soustrait le contenu du registre `rs2` au contenu du registre `rs1` et stocke le résultat dans le registre `rd`. stores the result in `rd`.

lw (load word)

Format `lw rd, offs(rs1)`

Description Charge une valeur de 32 bits depuis la mémoire et la stocke dans le registre `rd`. L'adresse de la donnée chargée est la somme du contenu du registre `rs1` et de la constante immédiate `offs`.

sw (store word)

Format `sw rs2, offs(rs1)`

Description Stocke la valeur du registre `rs2` en mémoire. L'adresse est la somme du contenu du registre `rs1` et de la constante immédiate `offs`.