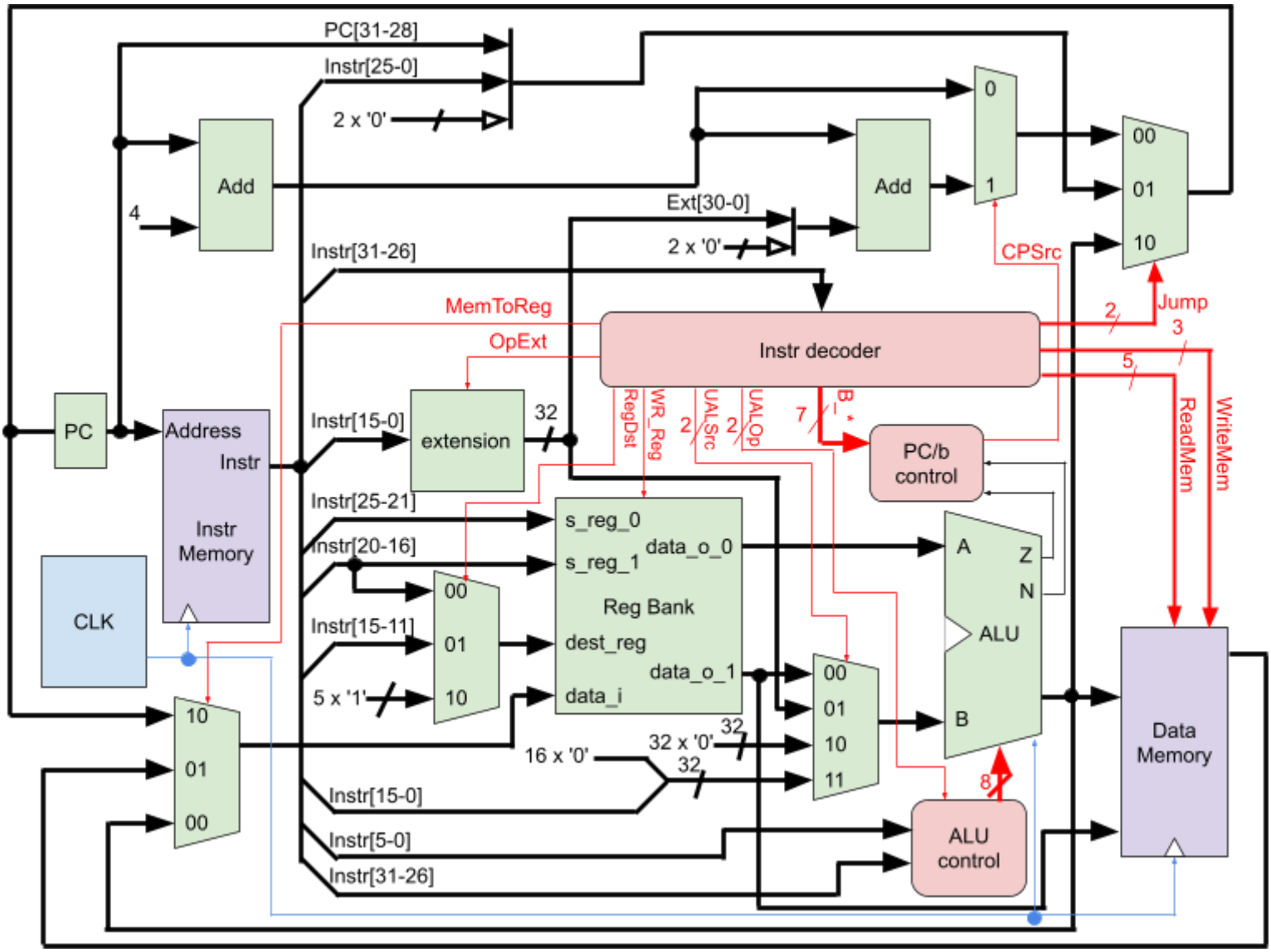
TP CSM - M2 SIAME

L'objectif de ces 6 séances de TP est de réaliser en VHDL un microprocesseur MIPS basé sur une version simplifiée du R3000 (une partie des instructions du jeu MIPS n'est pas implémentée). Vous commencerez par réaliser les composants nécessaires à l'implémentation du chemin de données (en vert sur le schéma), puis vous réaliserez les mémoires (en violet) et le chemin de contrôle (en rouge).



I. Registre 32 bits

En vous basant sur l'entité Reg du fichier sequential.vhd, écrivez l'architecture d'un registre 32 bits à déclenchement sur front montant.

Une fois le registre réalisé, testez-le en définissant un banc de test.

Port	Taille	I/O	Rôle
source	32 bits	Entrée	Bus d'entrée
wr	1 bit	Entrée	1 -> mise à jour 0 -> maintien de la valeur du registre
clk	1 bit	Entrée	Mise à jour sur front montant
output	32 bits	Sortie	Valeur du registre

II. Banc de registres

En vous basant sur l'entité RegisterBank du fichier sequential.vhd, écrivez l'architecture d'un banc de registres comprenant 32 registres de 32 bits chacun. Le registre R0 est câblé en dur sur la valeur 0 (il renvoie toujours 0 en lecture). Le banc permet de lire 2 registres en parallèle et d'écrire une donnée de 32 bits dans un registre. Vous pourrez utiliser le type bus_mux_array qui permet d'instancier des tableaux de signaux 32 bits. Vous n'êtes pas obligés de réutiliser le composant Reg défini à la question précédente, il sera utile plus tard dans le chemin de données pour stocker le Program Counter.

Une fois le banc de registres réalisé, testez-le en définissant un banc de test.

Port	Taille	I/O	Rôle
s_reg_0	5 bits	Entrée	Indice de registre à lire
s_reg_1	5 bits	Entrée	Indice de registre à lire
dest_reg	5 bits	Entrée	Indice du registre à écrire
data_i	32 bits	Entrée	Valeur à écrire dans le registre dest_reg
wr_reg	1 bit	Entrée	1 -> mise à jour du registre dest_reg 0 -> maintien de la valeur de tous les registres
clk	1 bit	Entrée	Mise à jour sur front montant
data_o_0	32 bits	Sortie	Valeur du registre s_reg_0
data_o_1	32 bits	Sortie	Valeur du registre s_reg_1

III. Multiplexeur 4 vers 1 générique

En vous basant sur l'entité Mux4_1 du fichier combi.vhd, écrivez l'architecture d'un multiplexeur 4->1 dont les 4 entrées (et la sortie) ont une taille paramétrable **size**.

Port	Taille	I/O	Rôle
reg0	size-1	Entrée	Opérande 0
reg1	size-1	Entrée	Opérande 1
reg2	size-1	Entrée	Opérande 2
reg3	size-1	Entrée	Opérande 3
c	2	Entrée	Sélecteur
reg_out	size-1	Sortie	Récupère l'opérande dont l'indice est codé par c

IV. Additionneur 32 bits simple

En vous basant sur l'entité Add du fichier combi.vhd, écrivez l'architecture flot de données d'un additionneur prenant deux entrées 32 bits non-signées.

Une fois l'additionneur réalisé, testez-le en définissant un banc de test.

Port	Taille	I/O	Rôle
reg1	32 bits	Entrée	Opérande 1
reg2	32 bits	Entrée	Opérande 2
reg3	32 bits	Sortie	Reg3 <- Reg1 + Reg2

V. Additionneur 32 bits avec retenues

En vous basant sur l'entité AddCarry du fichier combi.vhd, écrivez l'architecture d'un additionneur prenant 2 entrées 32 bits (signées ou non signées) et fournissant en sortie la somme des entrées ainsi que les bits de retenue des deux bits de poids fort (31 et 30). Ces bits de retenue seront utiles pour mettre à jour les bits de carry et d'overflow de l'ALU.
Pour réaliser cet additionneur, on commencera par implémenter un composant faisant l'addition complète 1 bit, et on instanciera 32 de ces additionneurs dans une approche structurelle.

Une fois l'additionneur réalisé, testez-le en définissant un banc de test.

Port	Taille	I/O	Rôle
A	32 bits	Entrée	Opérande 1
B	32 bits	Entrée	Opérande 2
cin	1 bit	Entrée	Retenue entrante
s	32 bits	Sortie	S <- A+B
c31	1 bit	Sortie	Retenue du niveau 31
c30	1 bit	Sortie	Retenue du niveau 30

VI. Barrel shifter

En vous basant sur l'entité BarrelShifter du fichier combi.vhd, écrivez l'architecture d'un barrel shifter réalisant à la fois un décalage à gauche (sur la sortie SL) et un décalage à droite (sur la sortie SR) d'un signal 32 bits en entrée. On décalera de la valeur codée sur le signal d'entrée ValDec. Une version simple en VHDL consiste à réaliser tous les décalages possibles à gauche et à droite, à les stocker dans des tableaux, puis à sélectionner le résultat du décalage souhaité.

Une fois le barrel shifter réalisé, testez-le en définissant un banc de test.

Port	Taille	I/O	Rôle
A	32 bits	Entrée	Opérande 1
ValDec	5 bits	Entrée	Nombre de décalages à effectuer
SR	32 bits	Sortie	A >> ValDec
SL	32 bits	Sortie	A << ValDec

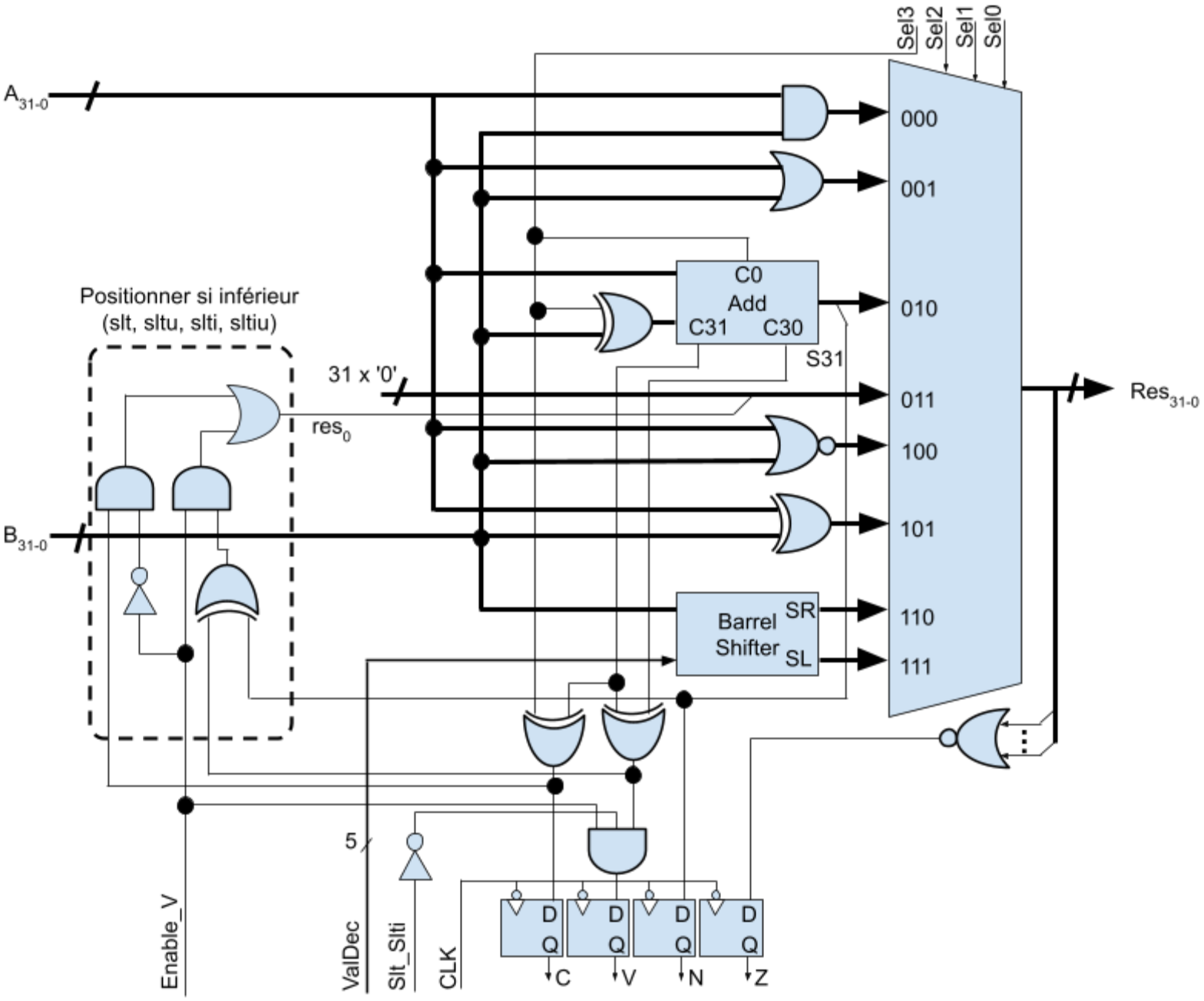
VII. ALU

En vous basant sur l'entité ALU du fichier combi.vhd, écrivez l'architecture correspondant à l'ALU du processeur, donnée dans le schéma ci-dessous.
Le bloc combinatoire de positionnement si inférieur (e.g. slt) est établi par la table de vérité suivante :

slt	Enable_V	C	V	N	res_0
signé	1	X	0	0	0
signé	1	X	0	1	1
signé	1	X	1	0	1
signé	1	X	1	1	0
non signé	0	0	X	X	0
non signé	0	1	X	X	1

Le bit res_0 est mis à un si l'opérande A est inférieure à l'opérande B, en utilisant une soustraction et en regardant les bits C, V et N du résultat.

res_0 = Enable_V.(N xor V) + /Enable_V.C



Les entrées de contrôle Enable_V, ValDec, Slt_Slti et Sel seront générées par des circuits que nous décrirons dans une séance prochaine, consacrée au chemin de contrôle du processeur.

Port	Taille	I/O	Rôle
A	32 bits	Entrée	Opérande 1
B	32 bits	Entrée	Opérande 2
sel	4 bits	Entrée	Signaux de contrôle
Enable_V	1 bit	Entrée	1-> opération signée
ValDec	5 bits	Entrée	Quantité de décalage
Slt	1 bit	Entrée	Contrôle du positionnement si inférieur (slt)
CLK	1 bit	Entrée	Mise à jour sur front descendant
Res	32 bits	Sortie	Résultat du calcul
N	1 bit	Sortie	1 -> résultat négatif
Z	1 bit	Sortie	1 -> résultat nul
C	1 bit	Sortie	Retenue sortante
V	1 bit	Sortie	Overflow

VIII. Module d'extension des immédiats

Un module est nécessaire pour étendre les immédiats (de longueur 16 bits dans les instructions) sur 32 bits.
Dans le format Immédiat des instructions MIPS, les 16 bits de poids faible de l'instruction représentent :

- un décalage signé pour les instructions de référence mémoire et de branchement
  => il faut donc étendre le bit de signe (b15)
- une constante signée dans le cas des instructions d'addition ou de positionnement si inférieur (e.g. slt) signées
  => il faut donc étendre le bit de signe (b15)
- une constante non signée pour toutes les autres instructions du format Immédiat
  => il faut étendre avec des '0'.

Lors de la spécification de la logique de contrôle (dans une séance prochaine), on générera un signal de contrôle ExtOp qui vaut 1 lorsque l'instruction décodée est signée et nécessite donc d'étendre le bit de signe (les deux premières catégories détaillées juste au-dessus) et 0 autrement (3ème catégorie et toutes instructions hors format Immédiat). Pour l'instant, on considère que ce signal est une entrée du module d'extension.
Pour résumer, lorsque le signal ExtOp est à 1, on étend les 16 bits de poids faible de l'instruction avec le bit de signe (b15), et lorsque ExtOp est à 0, on étend avec des zéros :

ExtOut			ExtOp
Extension		Valeur immédiate	
00000000	00000000	b_{15}…….b_8 b_7…….b_0	0
b_{15}…….b_{15}	b_{15}…….b_{15}	b_{15}…….b_8 b_7…….b_0	1

Port	Taille	I/O	Rôle
inst	32 bits	Entrée	Instruction en cours d'exécution
ExtOp	1 bit	Entrée	1 -> extension signée 0 -> extension non signée
ExtOut	32 bits	Sortie	Valeur immédiate étendue