Université Sétif 1
Faculté des Sciences
Département d'Informatique

2-ième année(année2019/20)
Architecture des Ordinateurs
Enseignant Kara Abdelaziz

Solution TP 1
(Conception d'un processeur rudimentaire)

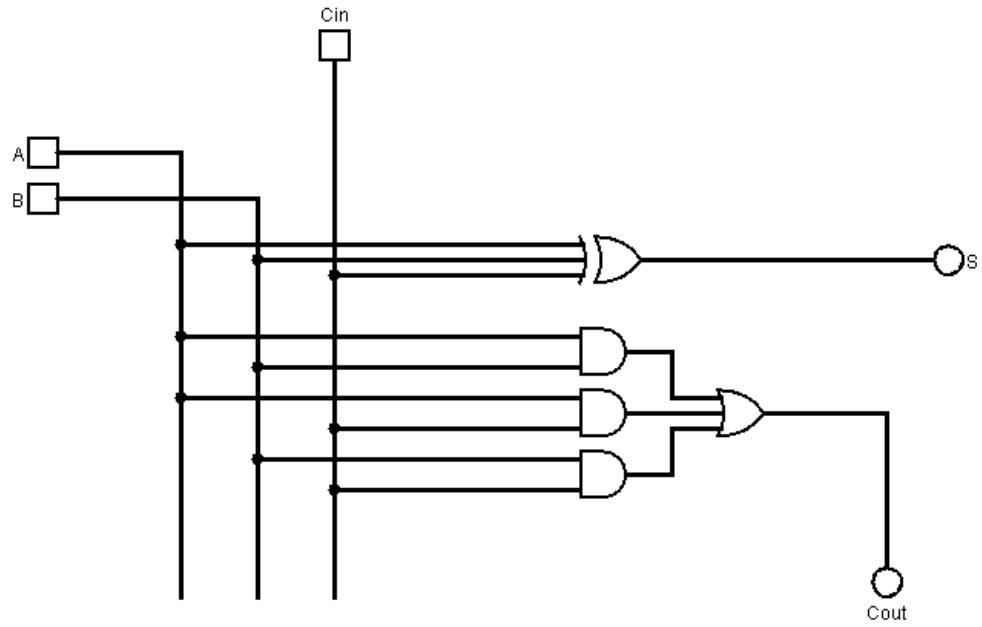
Exercice 01 :

les 3 étapes pour construire un processeur :

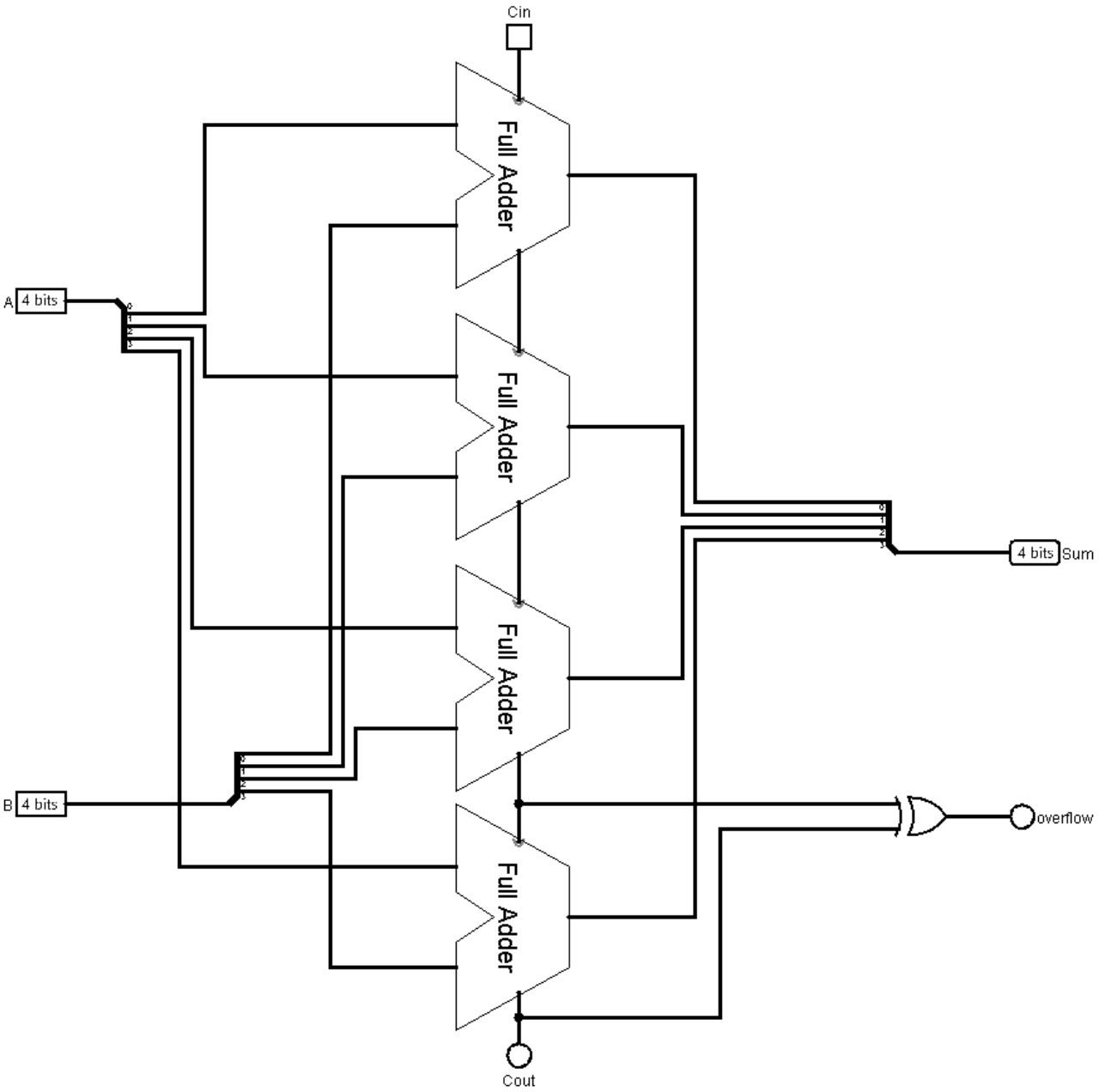
- **conception de l'UAL**
- **conception du Datapath**
- **conception de l'UCC**

1. La conception de l'UAL : pour construire une UAL on a besoin des composants suivants.

Le Full-Adder :

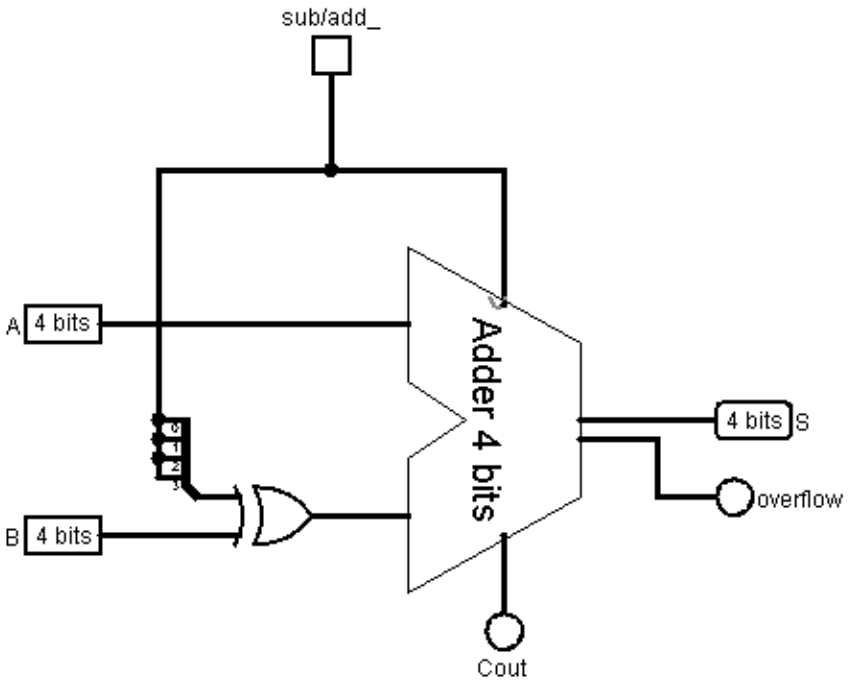


L'Adder (4 bits) :



Remarque : le bit du overflow est calculé par une porte XOR sur l'entrée de la retenue et la sortie de la retenue du Full-Adder du dernier bit.

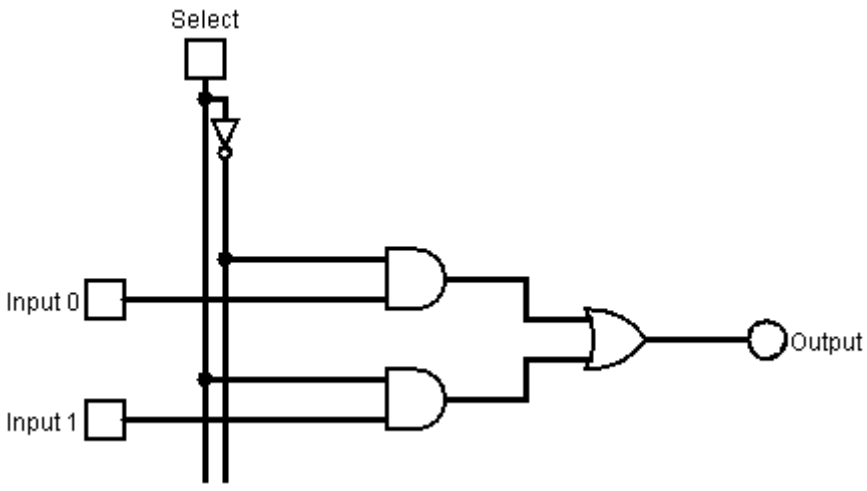
L'Additionneur/soustracteur (4 bits) :



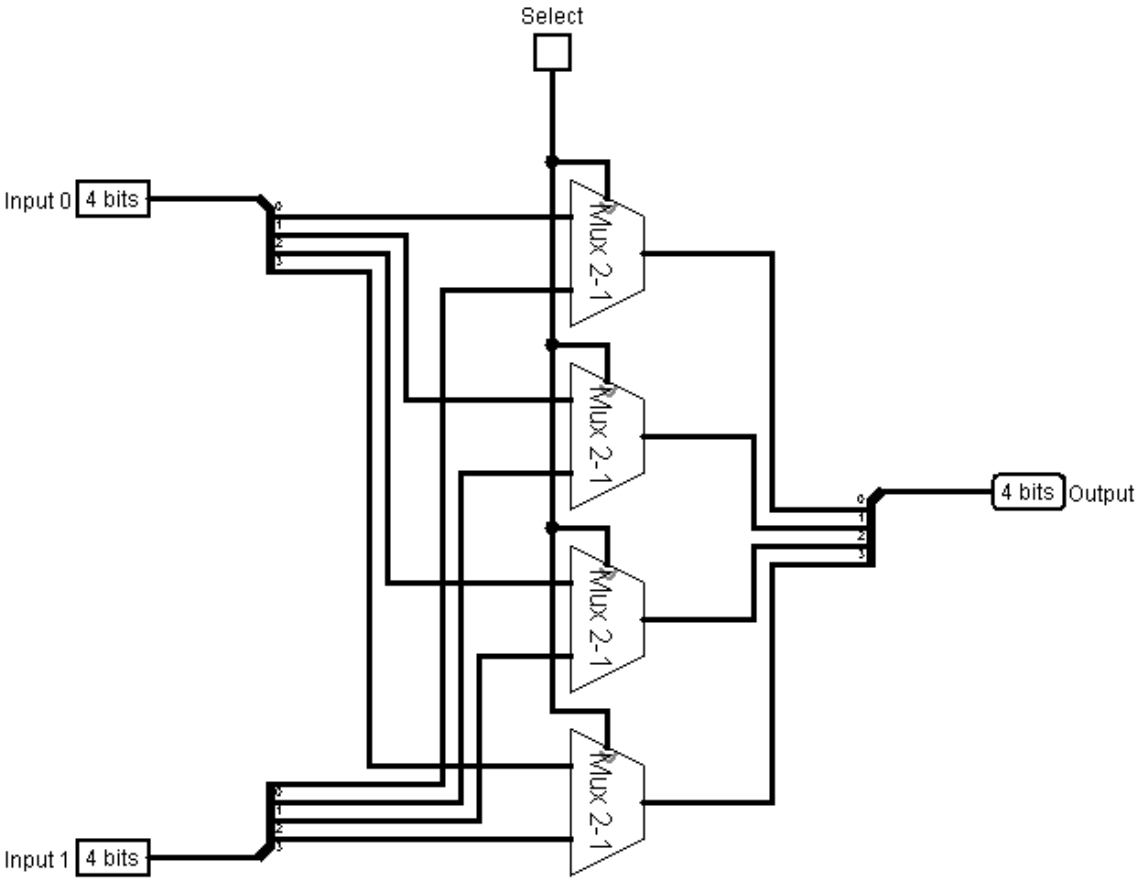
Remarque : La commande en haut permet de choisir entre l'addition et la soustraction, `add_` signifie $\overline{add}$ donc le 0 pour la commande fait l'addition, et 1 fait la soustraction parce qu'il n'y a pas de bar sur `sub`.

Les Multiplixeurs : tout au long de la construction du processeur on aurait besoin d'utiliser plusieurs types de multiplexeurs, pour l'instant on aurait besoin des multiplexeurs Mux 2-1, Mux 2-1 (4 bits), Mux 4-1 et Mux 4-1 (4 bits).

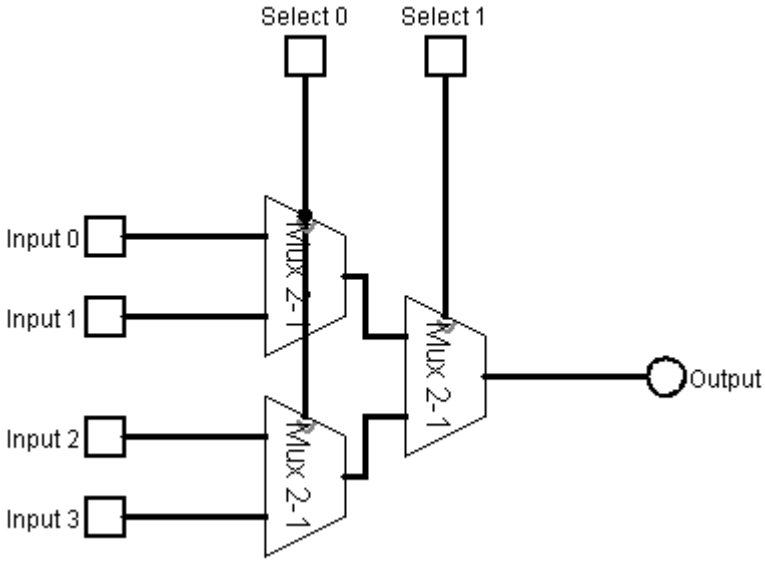
Mux 2-1 :



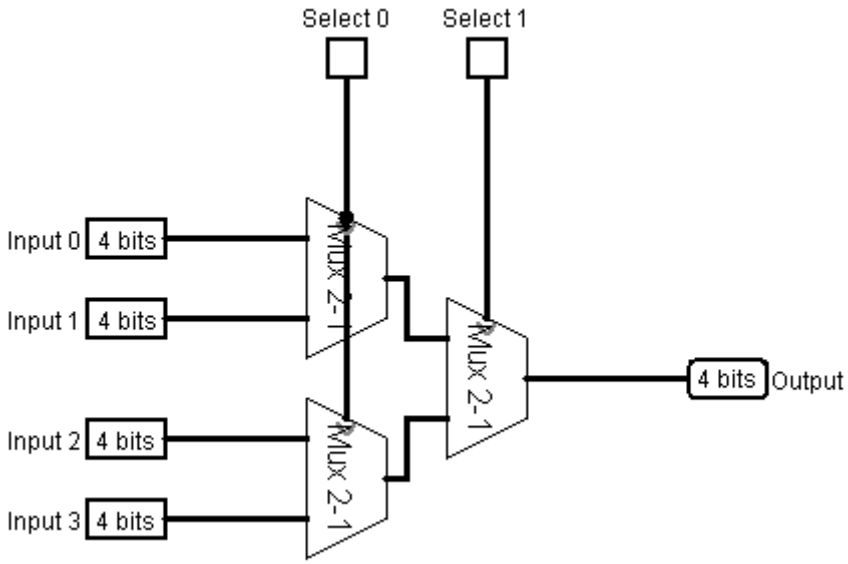
Mux 2-1 (4 bits) :



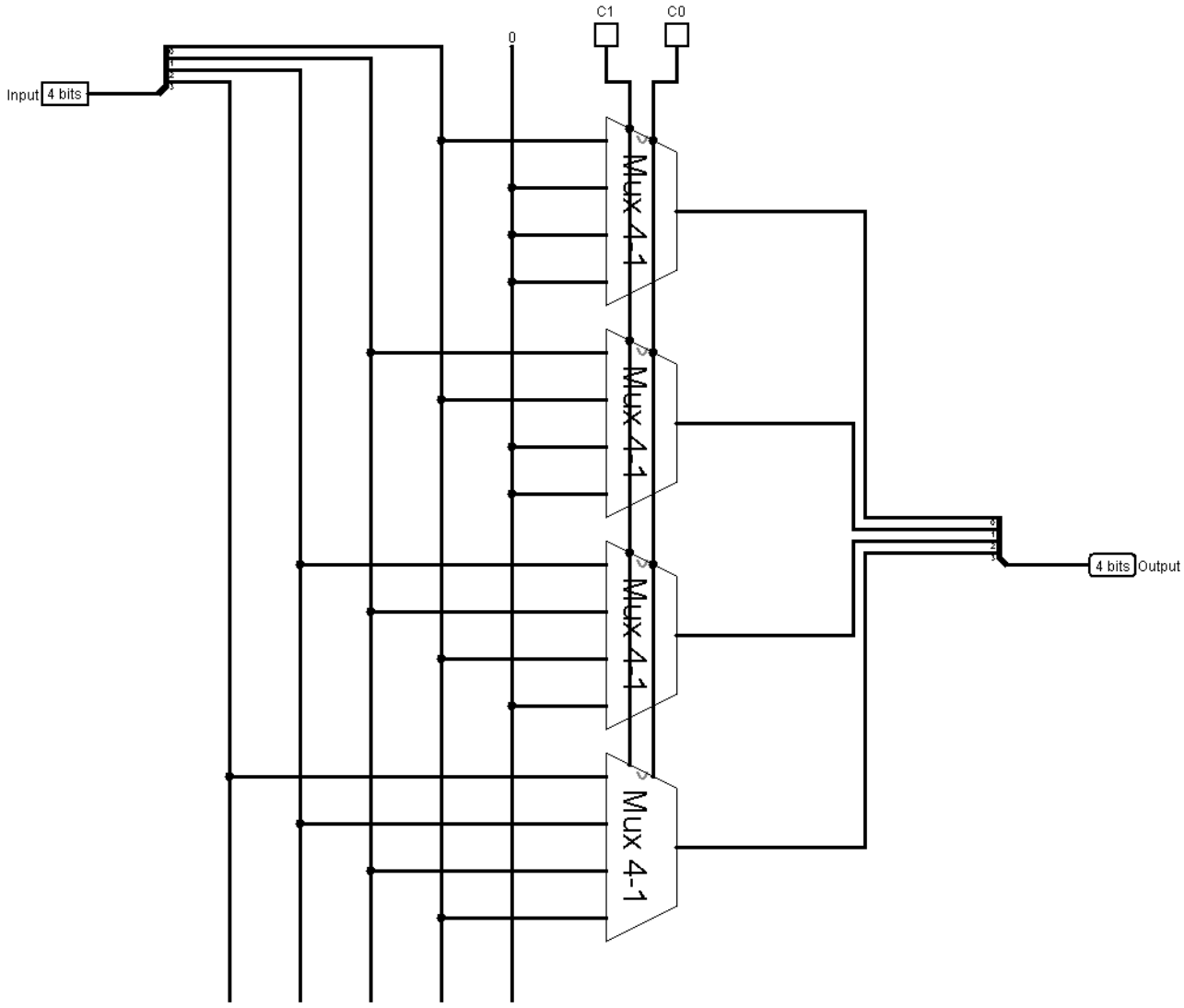
Mux 4-1 :



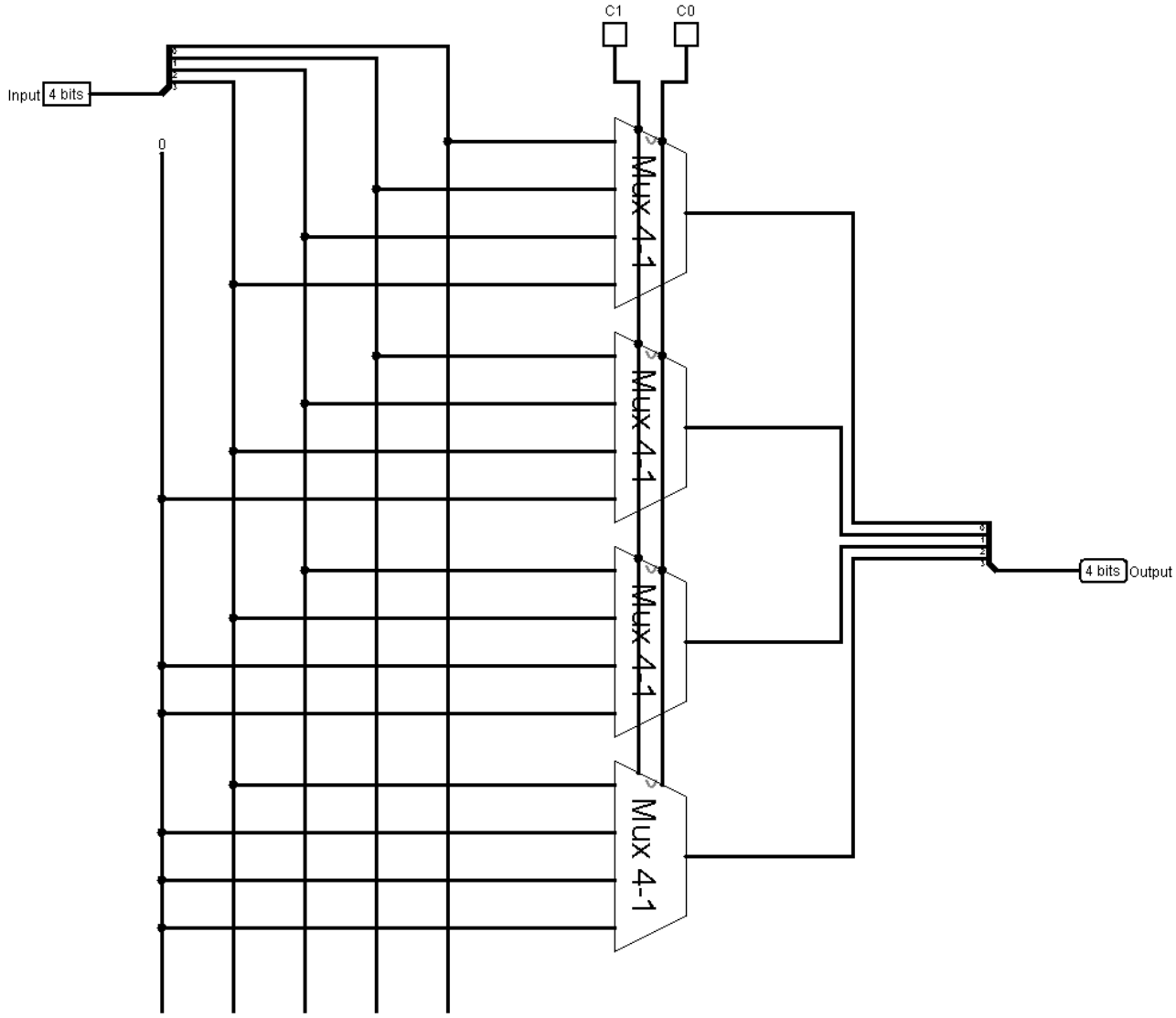
Mux 4-1 (4 bits) :



Shifter Left (4 bits) :



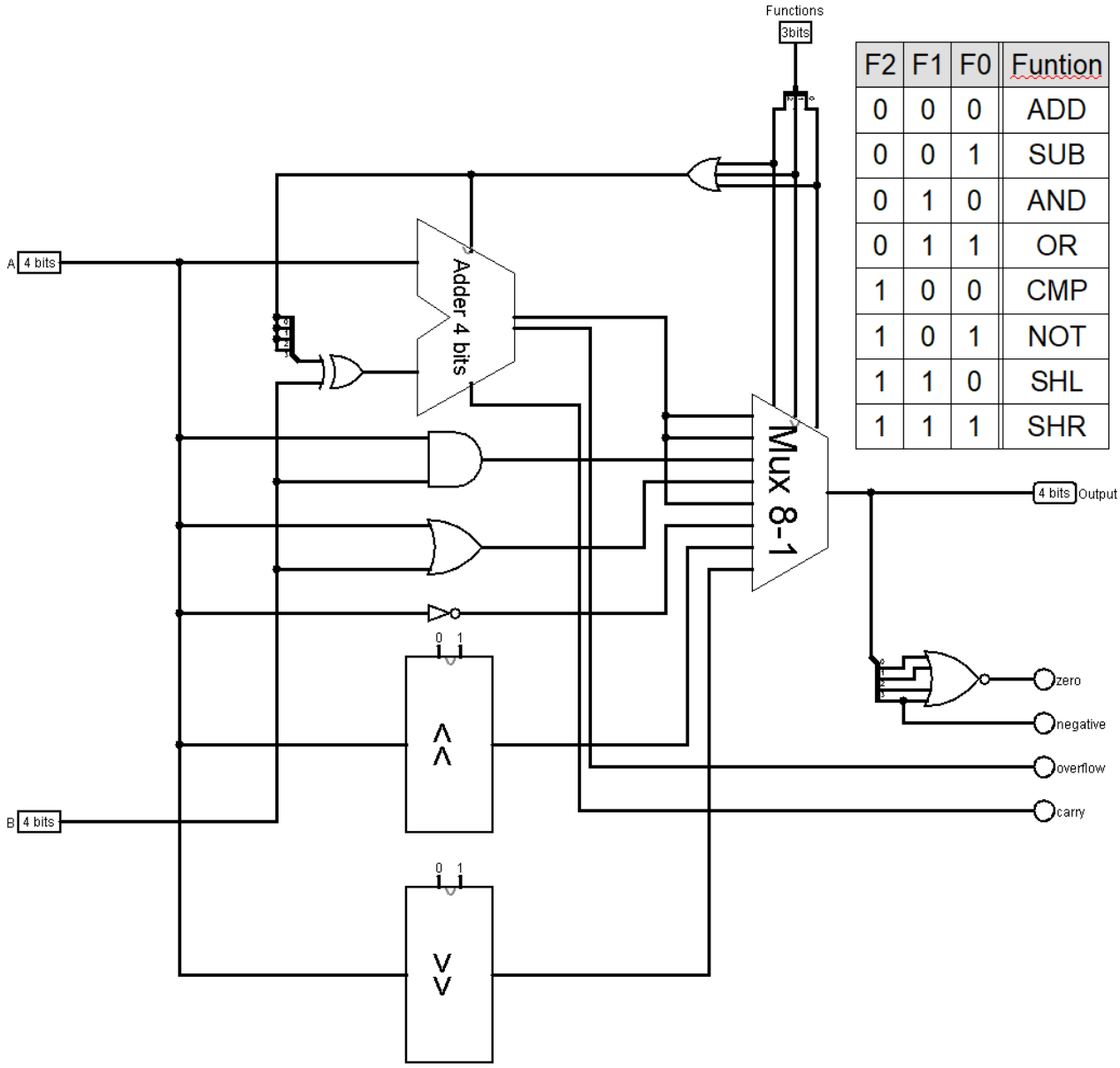
Shifter Right (4 bits) :



Remarque : Le Shifteur a besoin de 2 bits de commandes pour pouvoir faire un décalage sur 0 bit, 1 bit, 2 bits ou 3 bits. Dans notre processeur on aurait besoin que du décalage de 1 bit, ainsi on va fixer les commandes dans l'UAL après des Shifter par des constantes.

L'UAL (4 bits) : L'assemblage final de l'UAL se fait en ajoutant quelques éléments représentée sur la figure en bas. On peut observer par exemple que les opérations et, ou, non logiques sont implémentées par les portes (4 bits) AND, OR, NOT. On observe aussi que la commande de l'additionneur/soustracteur est actionnée par une porte OR sur les 3 entrées de fonctions en sachant que ça ne retourne 0 que si ils sont à 000 (le code pour l'addition), les autres ce sont de la soustraction qu'on va utiliser 2 fois dans l'UAL, pour la soustraction et pour la comparaison. On a aussi besoin d'un multiplexeur 8-1 (4 bits) qu'on va implémenter juste après. Pour les bits de Flag, le bit zero est implémenté par une porte NOR, ça s'active que lorsque toutes ses entrées sont à 0. le bit negative est pris directement du dernier bit (bit de signe). Les bits de retenue et d'overflow sont pris de

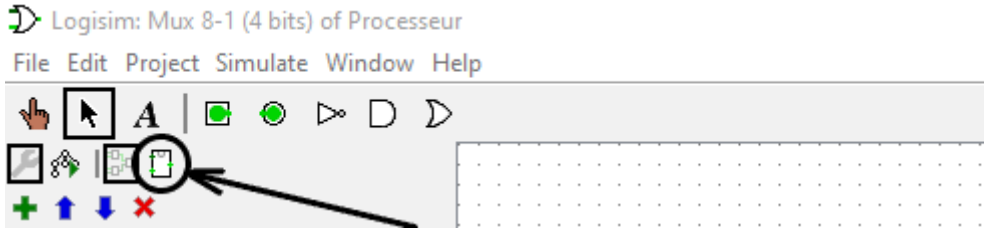
l'additionneur.



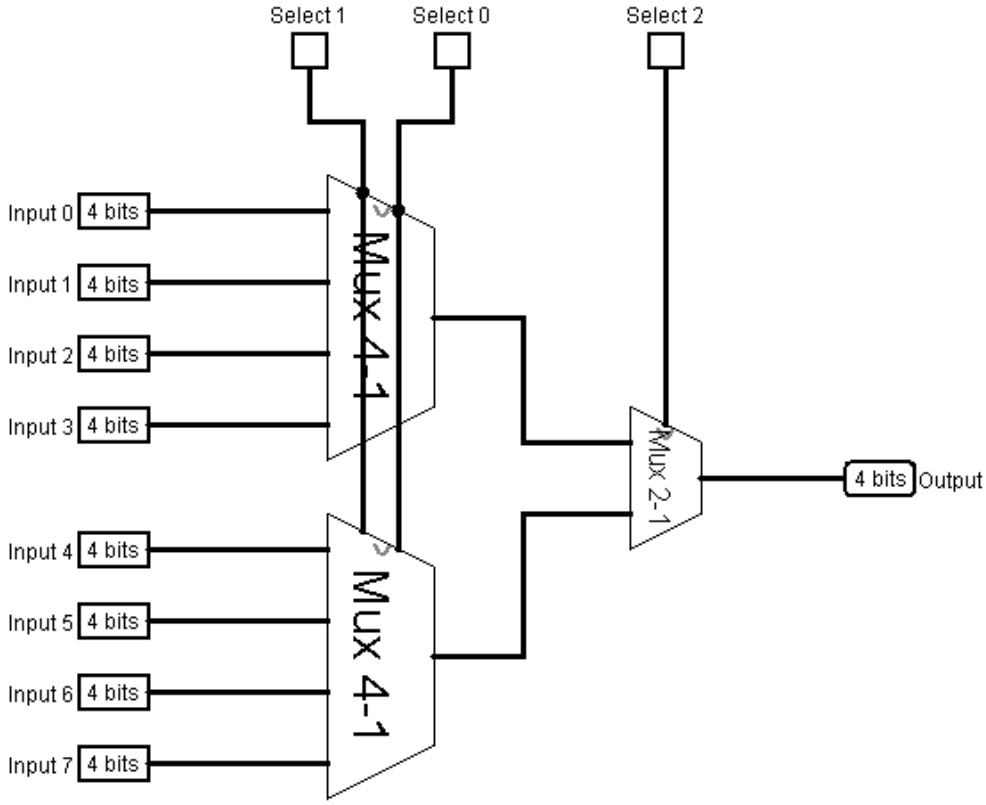
F2	F1	F0	Funtion
0	0	0	ADD
0	0	1	SUB
0	1	0	AND
0	1	1	OR
1	0	0	CMP
1	0	1	NOT
1	1	0	SHL
1	1	1	SHR

Remarque 1: Une erreur commune très répondue est de ne pas suivre le même ordre des bits de commande intérieur lorsque ils sont utilisés à l'extérieur.

Remarque 2: Il est possible de modifier la forme des circuit et la positions des portes à partir de l'icône illustré sur l'image en bas.

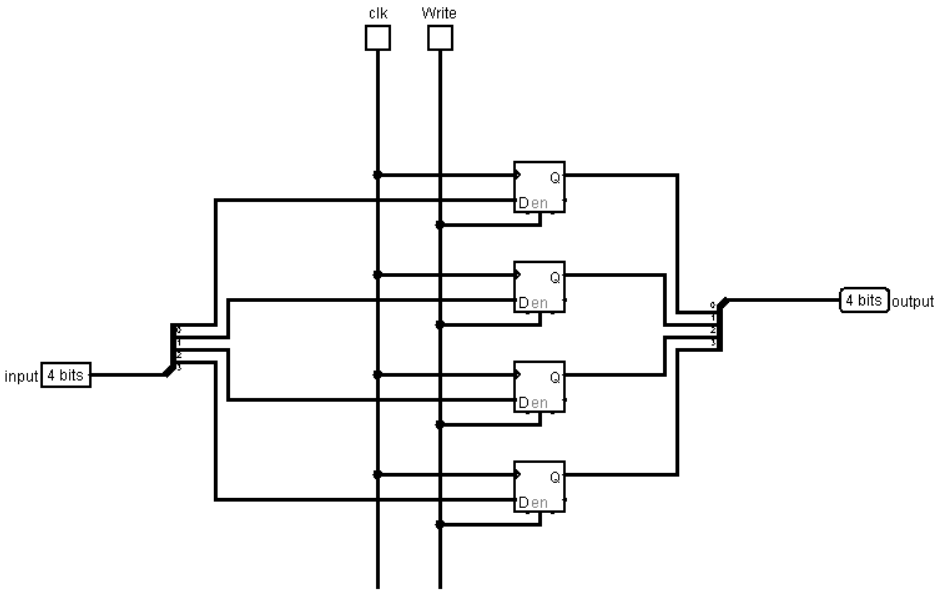


Mux 8-1 (4 bits) :

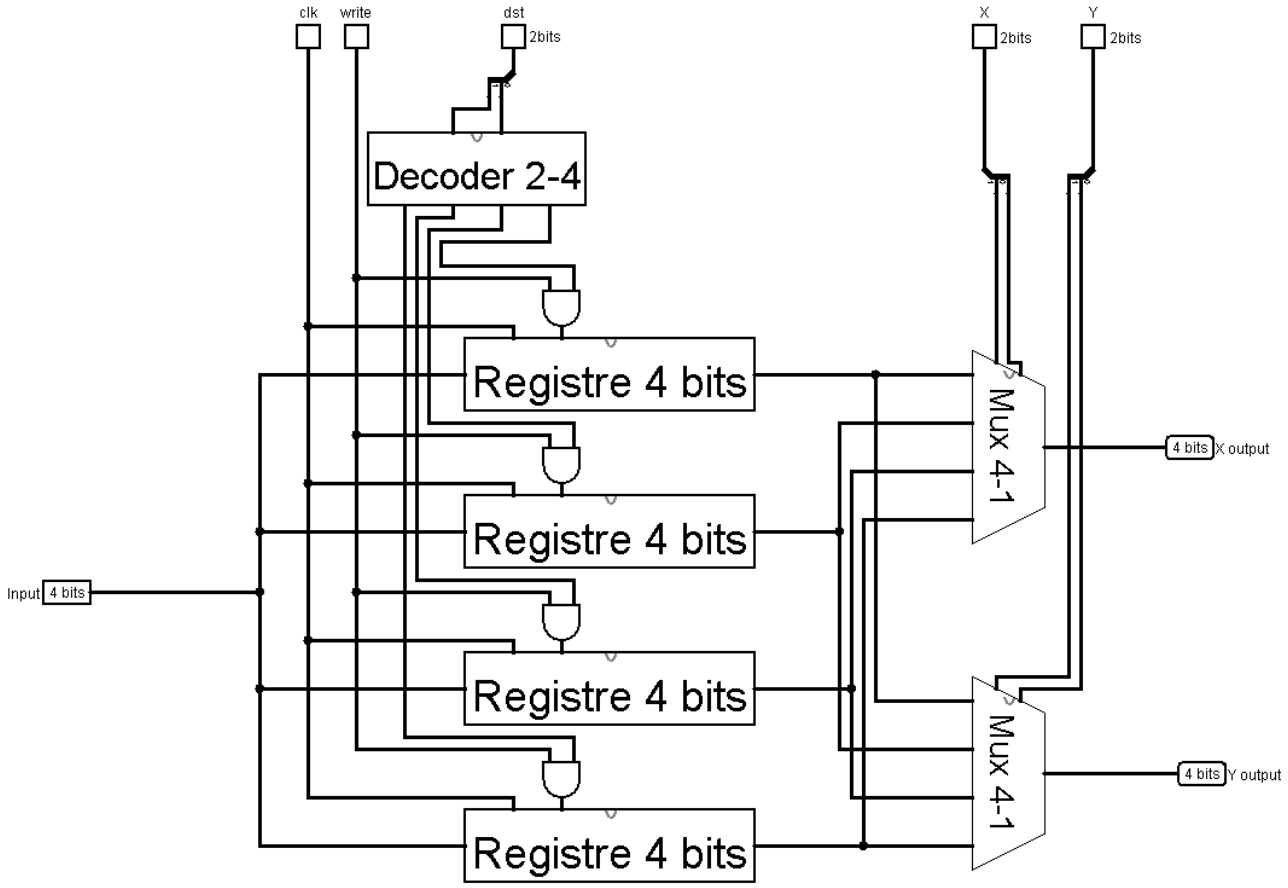


2. La conception du Datapath : En résumé, le Datapath contient plusieurs éléments différents, il contient un regster file (RF) de 4x4 bits, 3 registres de 4 bits (AR, DR, ST). Il contient un registre 12 bits (IR), de l'UAL, d'une ROM de programmes 16x12 bits de taille et une RAM de données de 16x4 bits, d'un compteur de 4 bits (PC) et de plusieurs multiplexeurs pour orienter le flux de données pour chaque instruction.

Register (4 bits) :

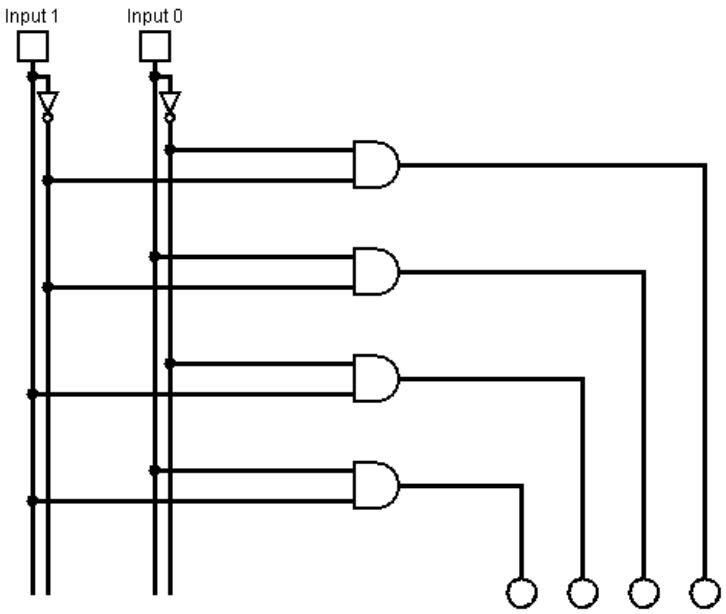


le RF :

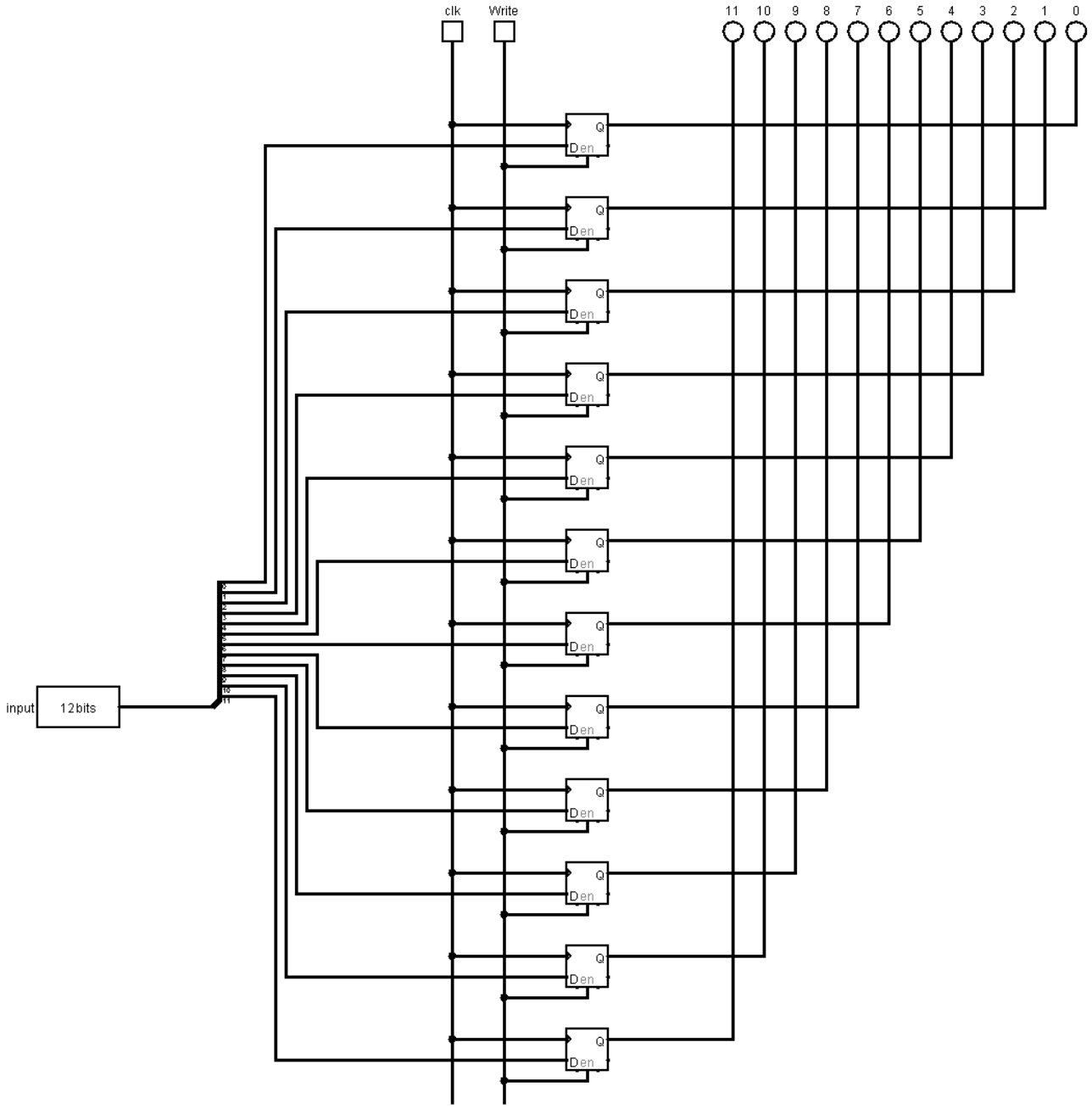


Remarque : Pour pouvoir écrire sur un registre on utilise un Decodeur 2-4, il va nous permettre de choisir sur 2 bits un registre parmi les 4 registres. On ajoute un autre signal de commande Write qui va commander l'écriture sur le registre désiré. Une porte AND est utilisée pour affirmer l'opération de l'écriture sur un seul registre.

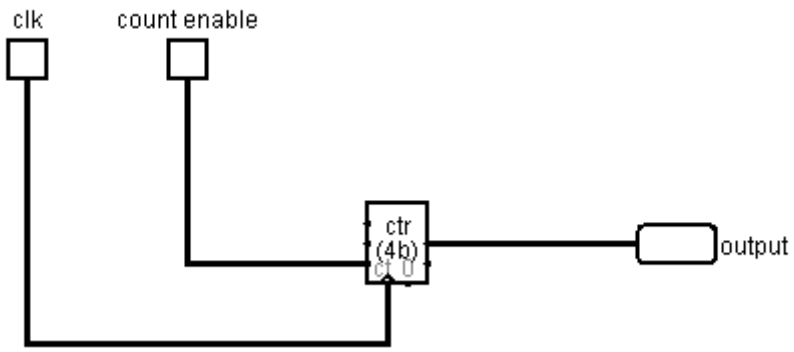
le Decoder 2-4 : il a été utilisé dans le RF



Register IR (12 bits) :



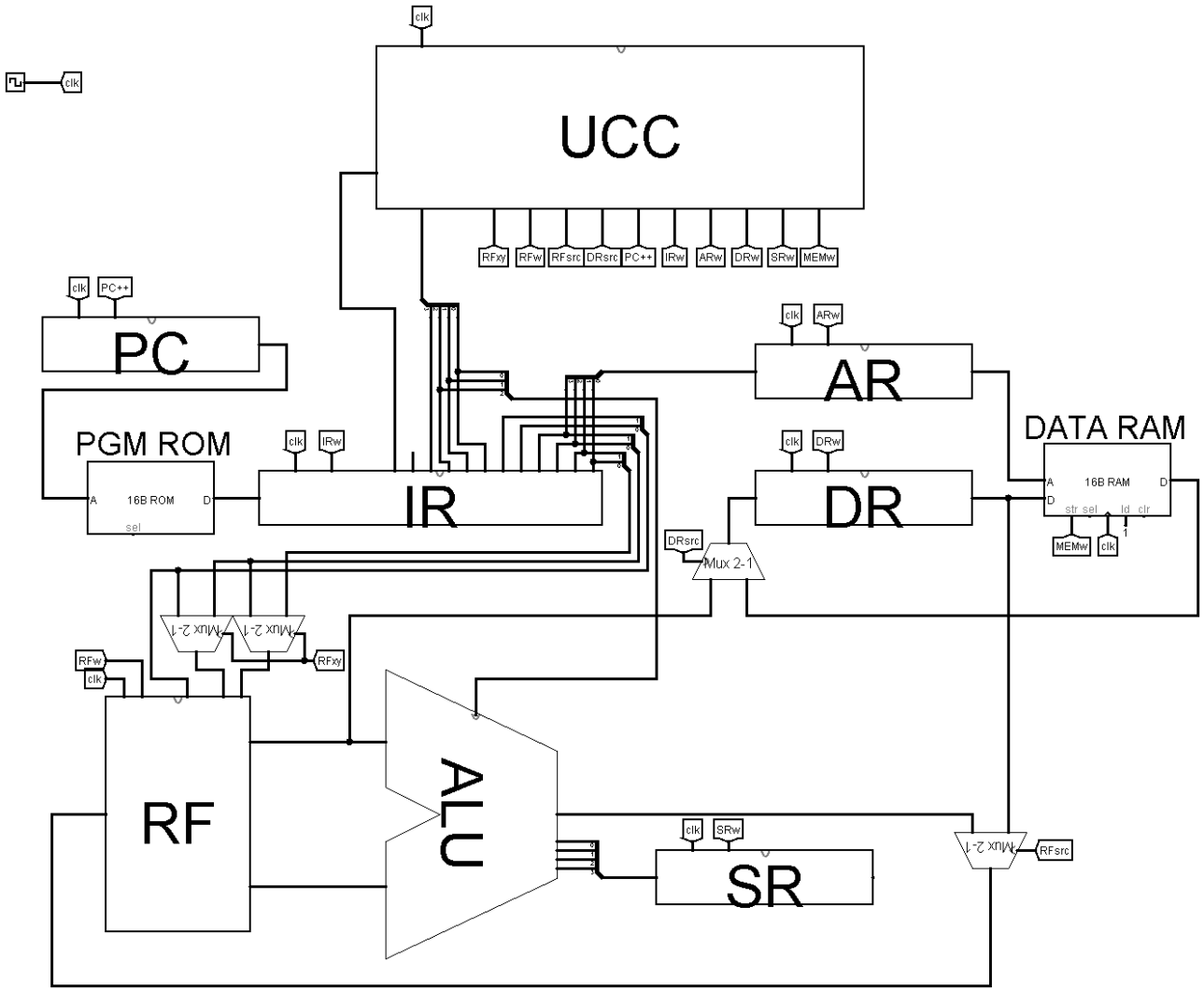
Le compteur PC (4 bits) :



Remarque 1: Les compteurs sont les composants qu'on devrait construire à partir des bibliothèques de Logisim, en raison que on a pas eu le temps les étudier.

Remarque 2: La commande `count enable` mise à 1 permet d'incrémenter le compteur par +1, et ainsi de passer à l'adresse suivante PC++.

Le Datapath : le datapath est construit en se basant sur les différents chemins que peut prendre une instruction. Par exemple on peut observer que AR prend des premiers 4 bits de l'instruction dans IR, ou encore que les 4 bits de l'OpCode dans IR sont directement transmis vers UCC pour le décodage de l'instruction, ainsi que le bit M du champ de format, les 3 premiers bits de l'OpCode sont aussi transmis à l'UAL pour l'opération à effectuer. On peut aussi remarquer que les 2 bits des registres r1, r2, r3 sont envoyés au RF et que des multiplexeurs vont choisir la valeur adéquate dans l'entrée X et Y de RF. Il existe aussi d'autre multiplexeurs pour faire la liaison de chemin entre la RAM et le RF.



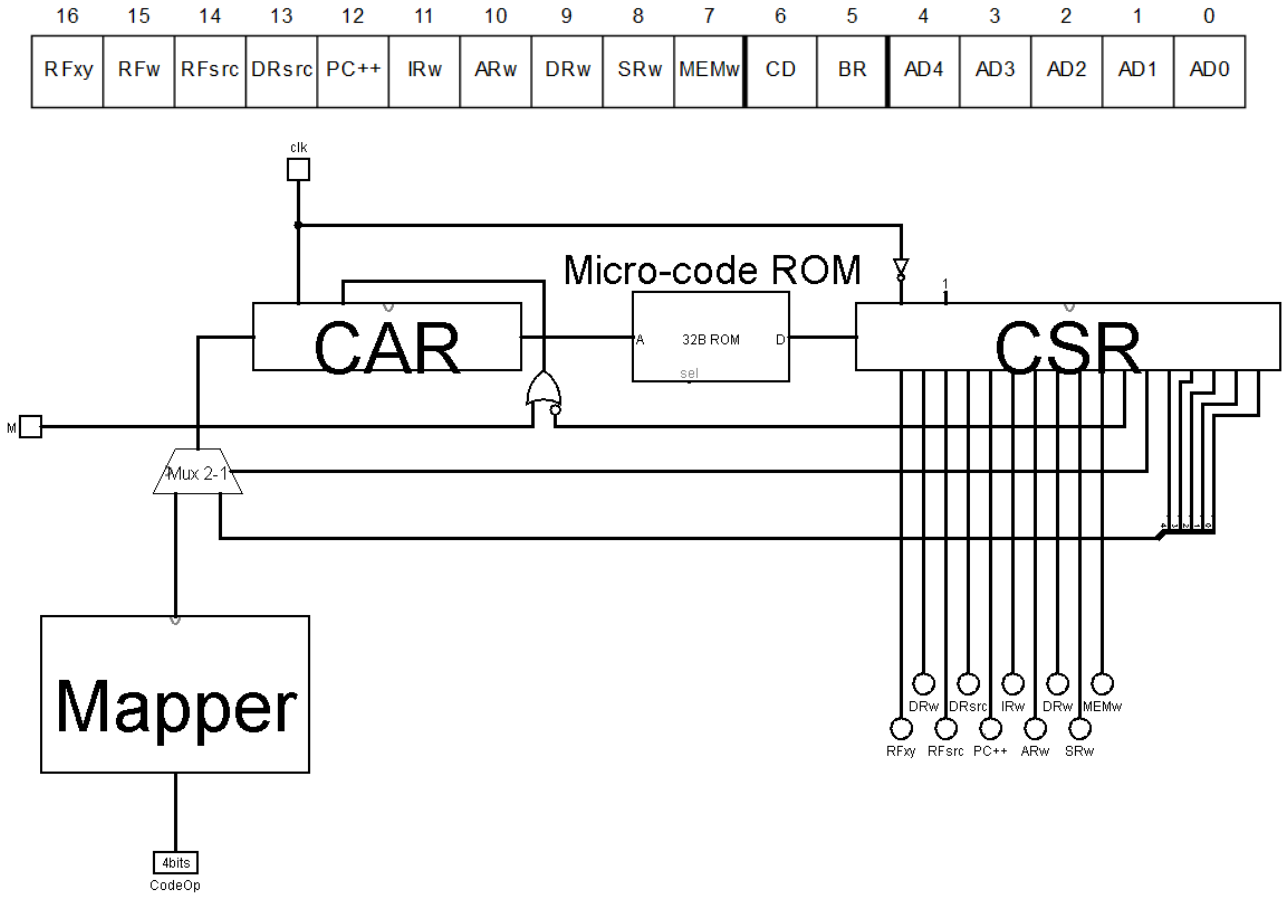
Remarque : La RAM doit avoir l'entrée séparée de la sortie, et ça c'est un paramètre à modifier dans ses options, c'est `separate load and strore` dans `Data interface`. Pour la commande de la RAM en vas implémenter que le signal d'écriture, la lecture est mise à 1.

La conception de l'UCC : L'unité de commande et de contrôle reçoit l'OpCode et le bit M de l'instruction de IR, et produit séquentiellement en suivant l'horloge les 10 signaux de commande vers le datapath.

La description des signaux de commande :

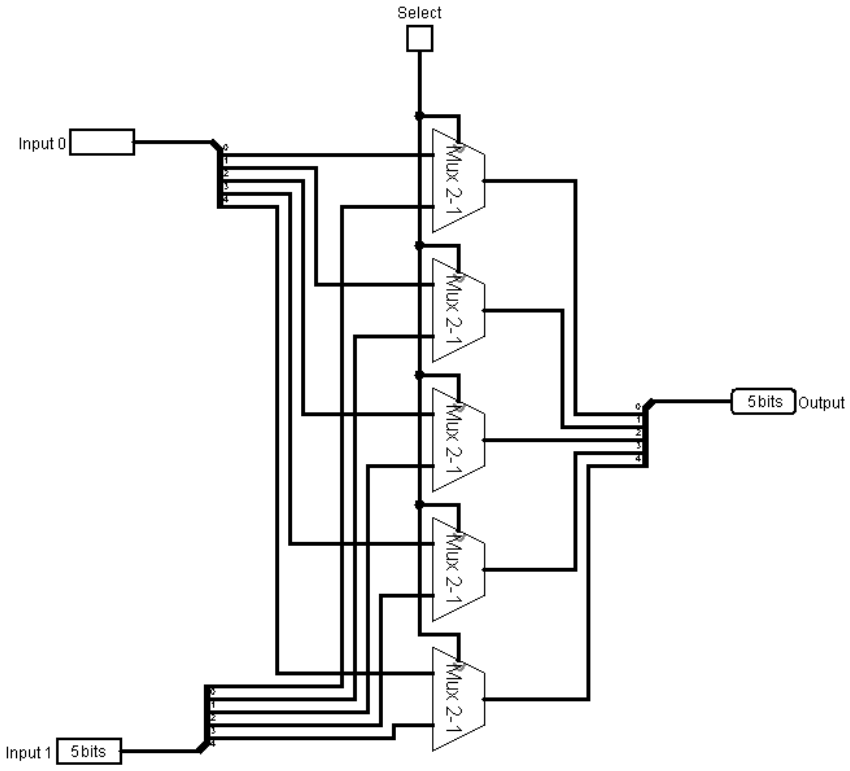
Signal	Description
RFxy	permet de choisir par 2 multiplexeurs entre r2,r3 et r1,r2 comme entrée de commande X,Y du RF.
RFw	la commande d'écriture sur RF dans le registre adressé dans l'entrée dst.
RFsrc	commande le multiplexeur qui choisit la source de RF entre l'ALU et DR.
DRsrc	commande le multiplexeur qui choisit la source de DR entre la RAM et la sortie X de RF.
PC++	incrémente le PC.
IRw	la commande d'écriture sur IR.
ARw	la commande d'écriture sur AR.
DRw	la commande d'écriture sur DR.
SRw	la commande d'écriture sur SR.
MEMw	la commande d'écriture sur la RAM.
clk	horloge

Le séquenceur se compose principalement de 2 registres CAR (Control Address Register) de 5 bits et CSR (Control Signal Register) de 17 bits et d'une ROM 32x17bits qui contient la liste les micro-instructions pour chaque instruction du processeur. Le registre CAR pointe sur la micro-instruction en cours d'exécution dans la ROM, le CSR quant à lui sauvegarde le code de la micro-instruction pour faire passer la liste des signaux de commande au datapath, le découpage de la micro-instruction est sur la figure en bas. Le séquenceur contient aussi un mappeur, il permet de décoder l'OpCode de l'instruction et de pointer directement sur l'adresse du début de la séquence de micro-code d'une l'instruction dans la ROM. 3 opérations de séquencement du prochain micro-code sont supportées par notre séquenceur, il peut chercher le micro-code dans l'adresse suivante CAR++, ou dans l'adresse du champ AD (les 5 premier bits dans CSR) ça faire un branchement, ou directement du mappeur, lors de la recherche d'une nouvelle instruction (Fetch). Ces 3 choix sont contrôlés par les 2 bits CD et BR dans CSR plus l'entrée M, qui vont influencer le multiplexeur et l'entrée Write de CAR pour exercer le séquencement adéquat.

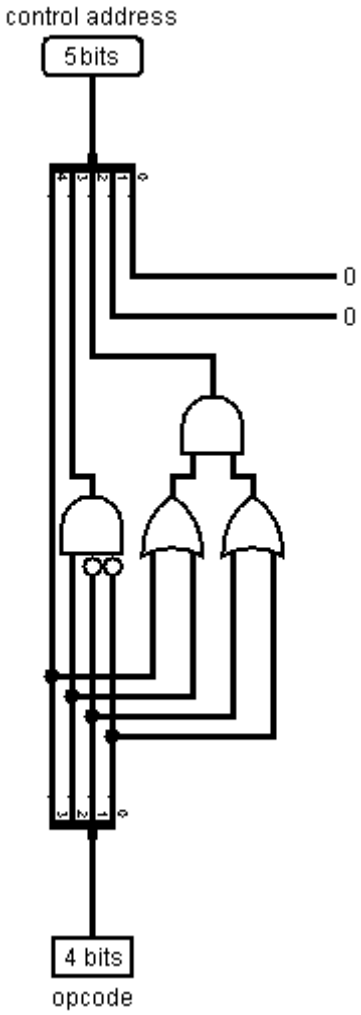


Remarque : Lors du démarrage du processeur, CAR doit impérativement contenir l'adresse 11000b (0x18), c'est l'adresse du début de la séquence de recherche de l'instruction (Fetch).

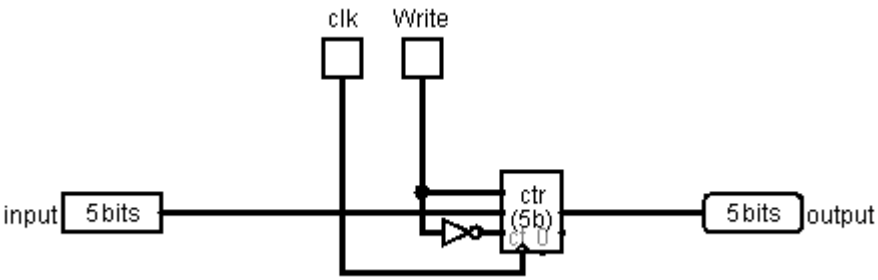
Le Multiplexeur 2-1 (5 bits) : on a encore besoin d'un multiplexeur.



Le Mapper :



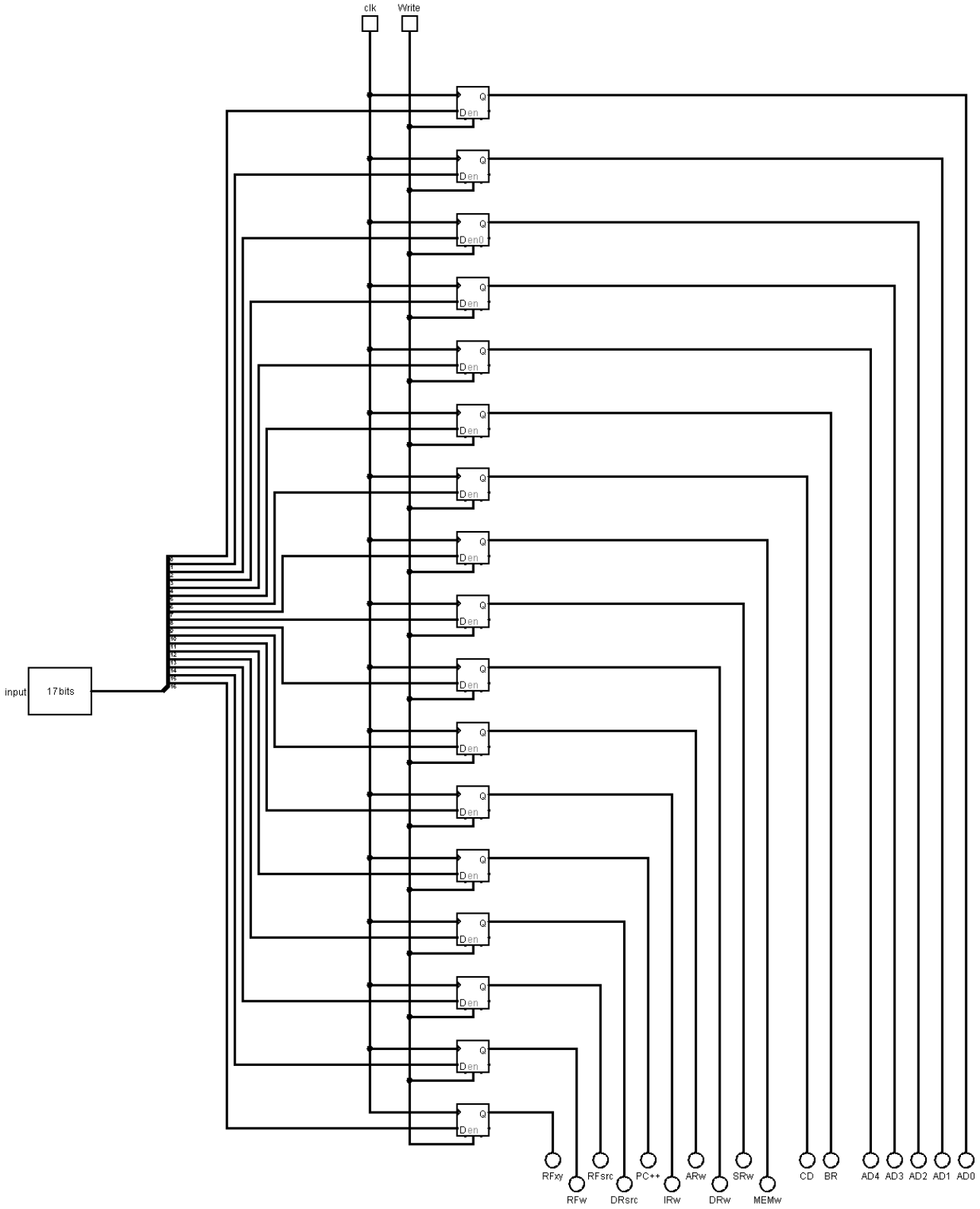
Le CAR :



Remarque 1: Le CAR aussi comme le PC est un compteur (counter en Anglais), on doit utiliser un compteur de la bibliothèque Logisim. La commande `Write` da CAR permet d'incrémenter +1 si elle est à 0, ou de charger une nouvelle adresse de l'entrée `input`.

Remarque 2 : le signal BR (BRanche) du séquenceur, s'il est mis à 1 il va indiquer au multiplexeur de choisir la valeur du mappeur, s'il est à 0 c'est en fonction de CD (ConDition), si CD est égale à 0 c'est la valeur de AD qui est passée au CAR, sinon c'est en fonction de M, si M est égale à 0 CAR s'incrémente de +1, sinon CAR reçoit AD.

Le CSR :



Remarque : Le CSR est déphasé d'un demi cycle d'horloge, c'est le NOT qui fait inverser le signal d'horloge pour le CSR, la raison est que les signaux de commande doivent rester stable lors du front montant d'horloge des composants du datapath. CSR fait réellement changer ses valeurs en provenance de la ROM lors du front descendant de l'horloge.

La table du micro-code dans la ROM :

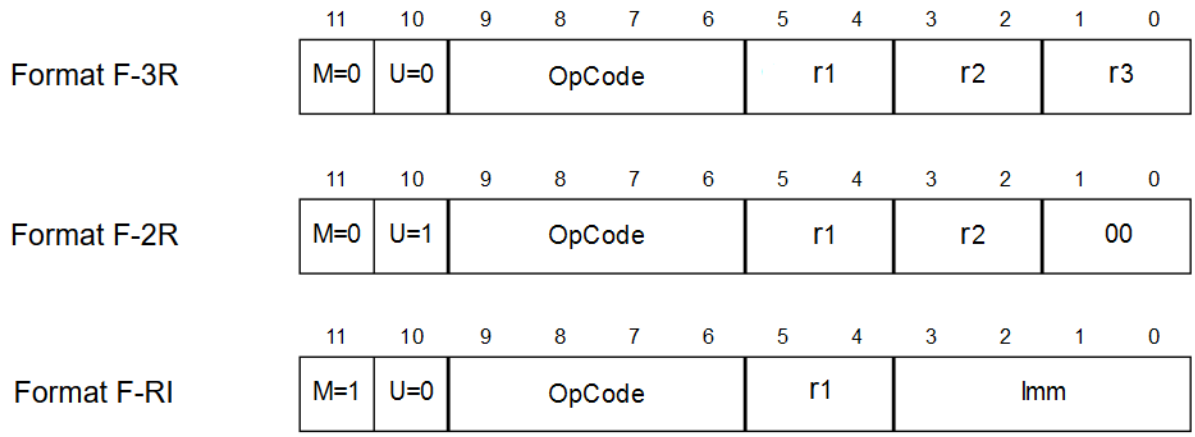
Instr	μ-inst	addr	Control Signals										seq ctr		AD	Hexadecimal
			RFxy	RFw	RFsrc	DRsrc	PC++	IRw	ARw	DRw	SRw	MEMw	CD	BR		
ADD SUB AND OR	ALU1	00000	0	1	1	-	0	0	0	0	0	0	0	0	11000	0x0C018
NOT SHL SHR	ALU2	00100	0	1	1	-	0	0	0	0	0	0	0	0	11000	0x0C018
CMP	CMP	01000	1	0	-	-	0	0	0	0	1	0	0	0	11000	0x10118
LOAD	DR←RAM	10000	-	0	-	1	0	0	0	1	0	0	0	0	10001	0x02211
	RF←DR	10001	-	1	0	-	0	0	0	0	0	0	0	0	11000	0x08018
STR	DR←RF	10100	1	0	-	0	0	0	0	1	0	0	0	0	10101	0x10215
	RAM←DR	10101	-	0	-	-	0	0	0	0	0	1	0	0	11000	0x00098
Fetch	IR←ROM	11000	-	0	-	-	0	1	0	0	0	0	0	0	11001	0x00819
	PC++	11001	-	0	-	-	1	0	0	0	0	0	0	0	11010	0x0101A
	seq ctr	11010	-	0	-	-	0	0	0	0	0	0	1	0	11100	0x0005C
	CAR←map	11011	-	0	-	-	0	0	0	0	0	0	-	1	-	0x00020
Direct	AR←IR[3,0]	11100	-	0	-	-	0	0	1	0	0	0	0	0	11011	0x0041B

Remarque : Le don't care (-) est considéré comme égale à 0.

L'Instructions Set (Jeu d'instructions) :

Instruction	OpCode
ADD	0000
SUB	0001
AND	0010
OR	0011
CMP	0100
NOT	0101
SHL	0110
SHR	0111
LOAD	1000
STR	1001

Format d'instruction :



Remarque 1: Les 2 derniers bits 10 et 11 sont le code du Format.

Remarque 2: M est un signal utilisé par l'UCC, ça lui permet de reconnaître les instructions d'accès à la mémoire RAM, lorsque M=1.

Le Mappage : Le mappeur (mapper en Anglais) est un circuit combinatoire qui permet à l'UCC de reconnaître l'adresse du début de la séquence de micro-code dans la ROM des micro-instructions à partir de l'OpCode de l'instruction.

Instr OpCode	Adresse
ALU1	00000
ALU2	00100
CMP	01000
LOAD	10000
STR	10100
Fetch	11000
Direct	11100

Remarque : Le micro-code de Direct est spécifique aux instructions d'accès à la mémoire (LOAD et STR), il est responsable de charger le registre AD avec l'adresse de la cellule mémoire RAM impliquée dans l'instruction.