Rapport du projet « Microprocesseur »

Bruno Kauffmann Thanh Trung Nguyen
Quang Cuong Pham

12 juin 2005

1 Architecture

1.1 Organisation générale (Fig. 1)

Le Microprocesseur (μp) que nous avons construit est un μp à 8 bits : toutes les instructions et toutes les données sont ainsi codées sur 8 bits.

Il y a 4 grandes entités :

- La **Mémoire Programme**
- L'**Unité Arithmétique et Logique** (ALU)
- La **Mémoire Vive** (RAM)
- Le **bloc de Contrôle**

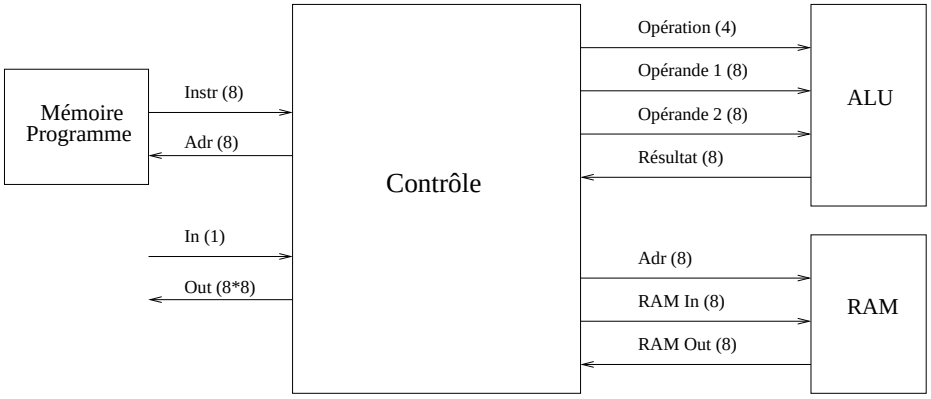


Fig. 1 – Organisation générale

1.2 Le bloc de Contrôle (Fig. 2)

Ce bloc comprend un **Décodeur**, un **banc de registres** comprenant 8 registres numérotés de 0 à 7, deux registres spéciaux : l'**accumulateur** et le **compteur de programme** (PC), ainsi que quelques mini-boîtes de contrôle.

1.2.1 Le Décodeur (Fig. 3)

La tâche du Décodeur est de lire l'instruction et de sortir des signaux de commande en fonction de cette instruction. Comme nous avons précisé, chaque instruction est codée sur 8 bits : les 5 premiers bits codent l'instruction à proprement parler, les 3 derniers représentent soit un numéro de registre ($R0 \ldots R7$), soit une constante ($0 \ldots 7$).

1.2.2 Le jeu d'instructions

N°	Instr.	Reg/Cst	Sémantique	Commentaire
00	AND	Reg	$Acc \leftarrow Acc \wedge Reg$	
01	ORR	Reg	$Acc \leftarrow Acc \vee Reg$	
02	XOR	Reg	$Acc \leftarrow Acc \oplus Reg$	
03	NOT		$Acc \leftarrow \neg Acc$	
04				
05	ADD	Reg	$Acc \leftarrow Acc + Reg$	
06	SUB	Reg	$Acc \leftarrow Acc - Reg$	
07	INC		$Acc \leftarrow Acc + 1$	
08	LSL	Reg	$Acc \leftarrow Acc \ll Reg$	
09	LSR	Reg	$Acc \leftarrow Acc \gg Reg$	
10				
11				
12				
13				
14				
15				
16	LRE	Reg	$Acc \leftarrow Reg$	Lecture registre
17	SRE	Reg	$Reg \leftarrow Acc$	Écriture registre
18	LRA	Reg	$Acc \leftarrow RAM[Reg]$	Lecture RAM
19	SRA	Reg	$RAM[Reg] \leftarrow Acc$	Écriture RAM
20	LIO		$Acc \leftarrow In; In \leftarrow 0$	Lecture quartz
21	SIO	Cst	$Out[Cst] \leftarrow Acc$	Écriture sortie
22	LCS	Cst	$Acc \leftarrow Cst$	Chargement d'une cst
23				
24	JMP		$PC \leftarrow Acc$	Saut inconditionnel
25	JPC	Reg	$Reg = 0 \Rightarrow PC \leftarrow Acc$	Saut conditionnel
26				
27				
28				
29				
30				
31				

1.2.3 Le banc de registres (Fig. 4)

Le banc de registre contient 8 registres de 8 bits chacun. Le mode d'adressage à une adresse impose que les registres sont utilisés principalement en mode lecture, seule l'instruction `SRE` accède aux registres en mode écriture en recopiant la valeur de l'accumulateur dans le registre précisé.

1.2.4 L'accumulateur (Fig. 5)

L'accumulateur est un registre à 8 bits qui contient l'opérande et le résultat de presque toutes les instructions. Son entrée est commandée par la **Boîte-Acc** qui elle-même reçoit ses signaux du Décodeur.

1.2.5 Le compteur de programme (Fig. 6)

Le PC est un registre à 8 bits qui contient l'adresse de l'instruction qu'il faut exécuter au prochain cycle. Sa sortie est donc connectée sur l'entrée de la Mémoire Programme. Son entrée est commandée par la **Boîte-PC**.

1.3 Les autres blocs

1.3.1 La Mémoire Programme

Elle contient le code du programme que l'on veut exécuter à l'aide du μp. Ce bloc possède une entrée à 8 bits (adresse de l'instruction qu'on veut exécuter) et une sortie à 8 bits (l'instruction en question).

La Mémoire Programme est simulée en soft, pour des raisons d'encombrement.

1.3.2 L'ALU (Fig. 7)

Ce bloc est chargé des opérations arithmétiques et logiques. Il dispose de 3 entrées : une à 4 bits codant l'opération à faire (parmi AND, ORR, XOR, NOT, ADD, SUB, LSL, LSR, INC) et deux à 8 bits qui contiennent les opérandes éventuels. Il possède aussi une sortie de 8 bits contenant le résultat de l'opération.

La figure 7 montre un mini-ALU à 1 bit dont les entrées/sorties sont :

- Une entrée à 3 bits codant l'opération (parmi AND, ORR, XOR, NOT, ADD, SUB)
- Une entrée opérandes à 2 bits, 1 bit par opérande
- Une entrée report de la mini-ALU précédente à 1 bit
- Une sortie résultat à 1 bit
- Une sortie report à 1 bit

Pour traiter les opérations AND, ORR, XOR, NOT, ADD et SUB nous connectons 8 mini-ALU en série. Les opérations LSL, LSR et INC ont un comportement global donc nécessitent un traitement à part.

1.3.3 La RAM

Elle contient des données qui sont créées et utilisées pendant l'exécution du programme. Deux entrées à 8 bits chacune contenant l'adresse de lecture/écriture et le mot à écrire, une sortie à 8 bits contenant le mot lu.

La RAM est simulée en soft, pour des raisons d'encombrement.

1.3.4 L'interface

Nous disposons d'une entrée `In` à 1 bit et de huit sorties `Out[0...7]` à 8 bits. Le fonctionnement de l'interface sera expliqué dans la section suivante.

L'interface est simulée en soft, pour des raisons d'encombrement.

2 Simulation d'une montre

2.1 Stratégie employée

Nous supposons qu'un quartz de fréquence 1024 Hz est connecté à l'entrée `In` du μp. Cela veut dire que tous les 1/1024 s, un certain mini-registre dénommé `In` à 1 bit (accessible à notre μp en lecture/écriture) voit sa valeur passer à 1 (ou y rester) sous l'impulsion du quartz. Nous supposons deplus que la fréquence de l'horloge du μp est très supérieur à 1024 Hz (par exemple égale à $256 \times 1024 = 262144$ Hz), si bien que nous pouvons faire toutes les mises à jour nécessaires sans être « pris de court » par le quartz.

Cela étant, nous disposons d'une instruction `LIO` qui charge la valeur du mini-registre dans l'accumulateur et remet (éventuellement) la valeur du mini-registre à 0. Ainsi la stratégie est la suivant : nous traitons toutes les mises à jour nécessaires en moins de 256 intructions, puis nous appelons `LIO` répétitivement jusqu'à ce qu'nous lisions un 1, auquel cas nous incrémentons un certain compteur, puis nous recommençons à faire des mises à jour, etc.

Nous disposons également d'une instruction `SIO Cst` qui écrit la valeur de l'accumulateur sur la sortie numéro `Cst`.

2.2 Simulation

Nous avons conçu une net-list qui implémente le μp décrit dans ce rapport (*cf.* l'annexe A), et nous avons simulé cette net-list à l'aide d'un simulateur de circuit écrit en Ocaml (*cf.* l'annexe B). Enfin, nous avons testé le μp ainsi simulé sur le programme de la montre (*cf.* l'annexe C).

A La net-list du μp

- Le Décodeur : `decodeur.net`
- La Boîte-Acc : `boite_acc.net`
- La Boîte-PC : `boite_pc.net`
- Le Banc de Registres : `boite_regs.net`
- La Sortie : `boite_out.net`
- L'ALU : `alu.net`
- Le μp lui-même : `microproc.net`

B Le simulateur de circuit

- Le parseur du langage d'assemblage : `parser.mli, parser.ml`
- Le simulateur : *cf.* le rapport du simulateur de Bruno Kauffmann

C Le programme simulant la montre en langage d'assemblage

Le programme `montre.swx` est écrit dans le langage d'assemblage propre à notre μp. Tous les 262144 cycles, il met à jour les sorties Out[0...5] de telle manière que ces sorties représentent le temps écoulé depuis le démarrage du programme sous le format hhmmss.

D Les figures

Références

[1] J. Vuillemin, *Théorie des technologies de l'information*, 2002.

[2] J. Duprat, F. de Denechin, *TD : Un microprocesseur RISC 16 bits*, 2001.

[3] E. Sanchez, *Processor Architecture*, 2002.

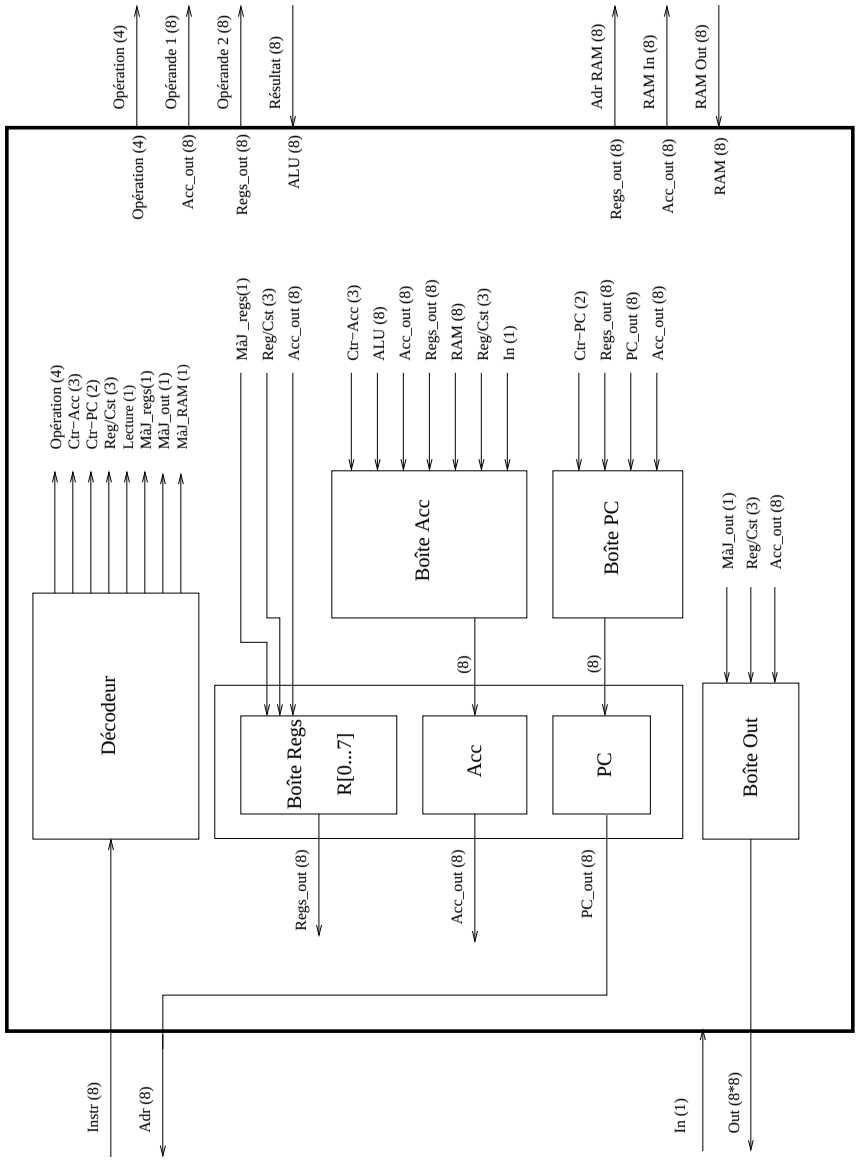


FIG. 2 – Le Bloc de Contrôle

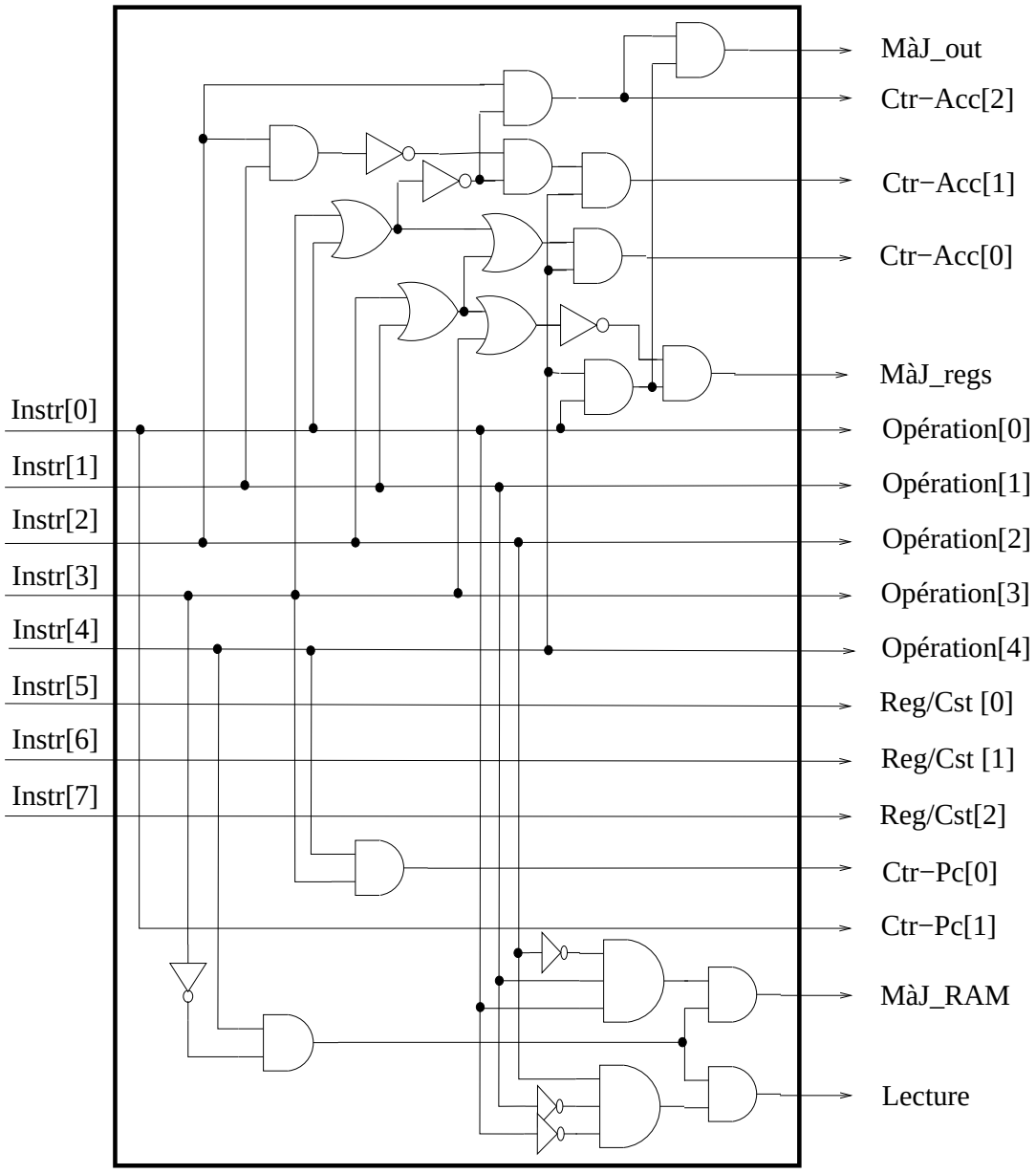


FIG. 3 – Le Décodeur

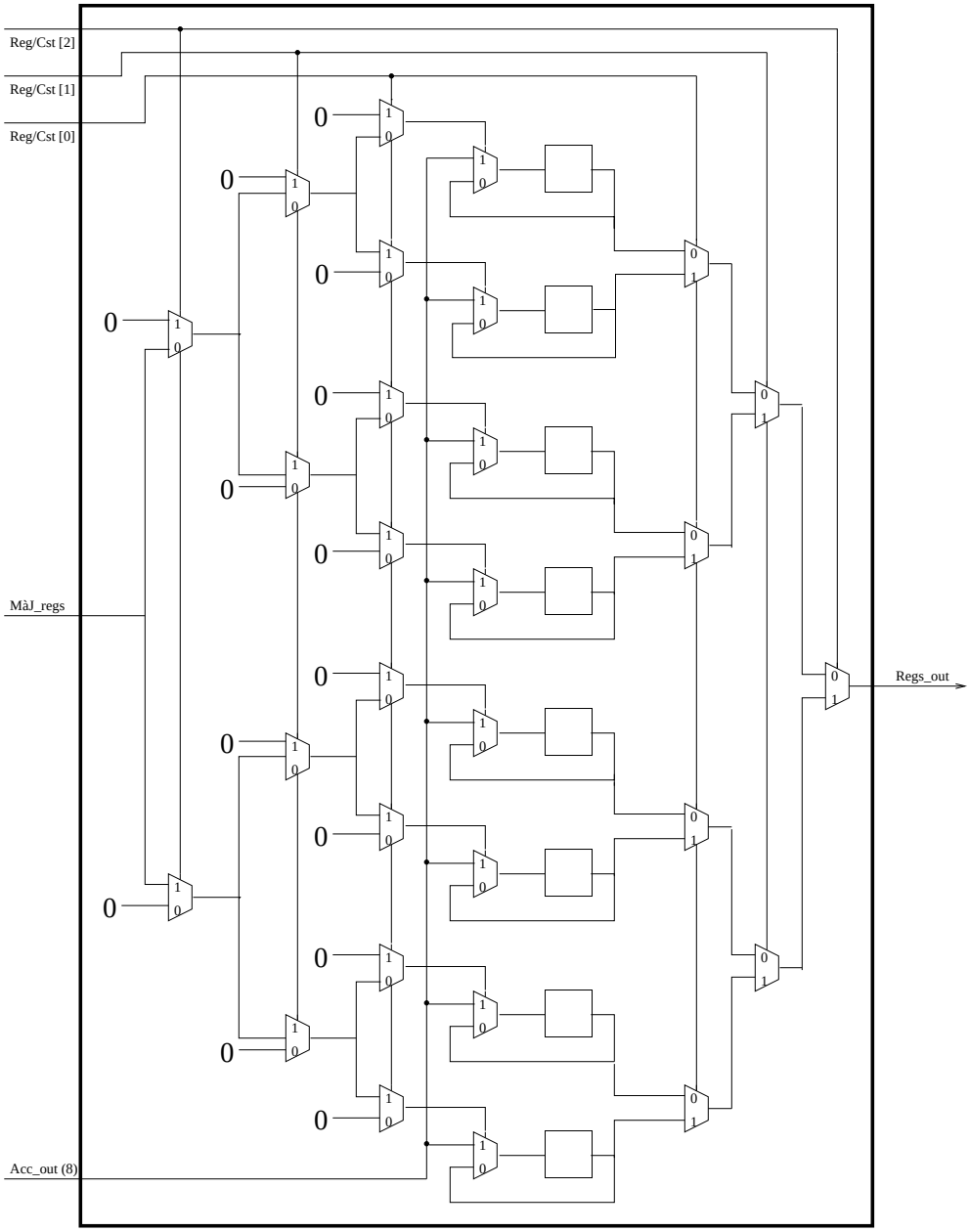
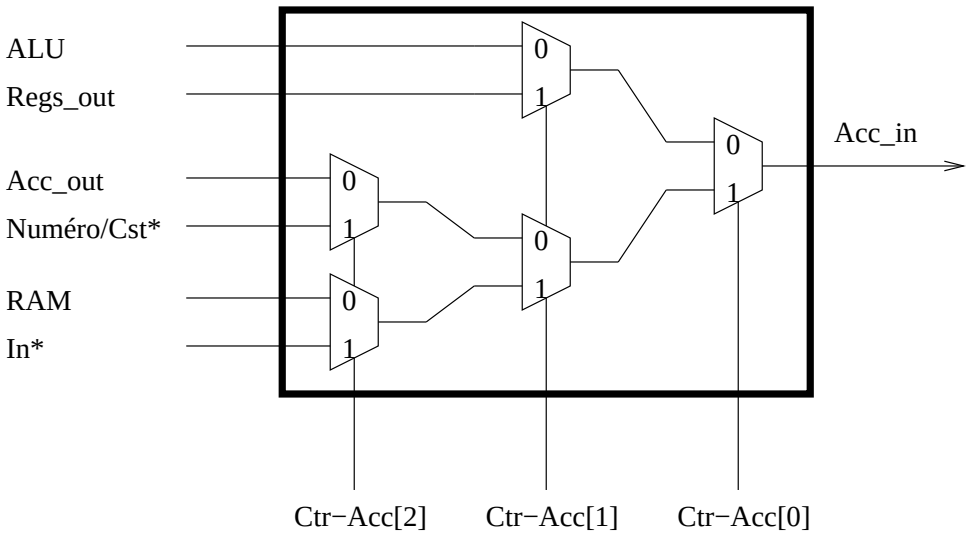


FIG. 4 – Le Banc de Registres



* Ces entrées ne sont pas à 8 bits, il convient donc de les compléter par des fils délivrant des zéros dans la réalisation finale.

FIG. 5 – La Boîte de Contrôle de l'Accumulateur

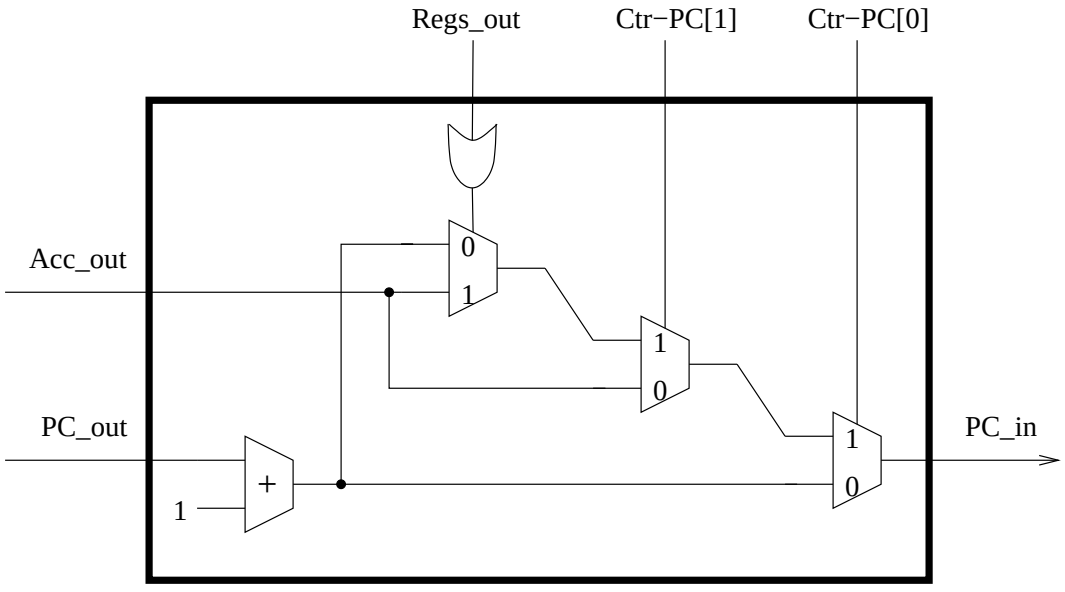


FIG. 6 – La Boîte de Contrôle du Compteur de Programme

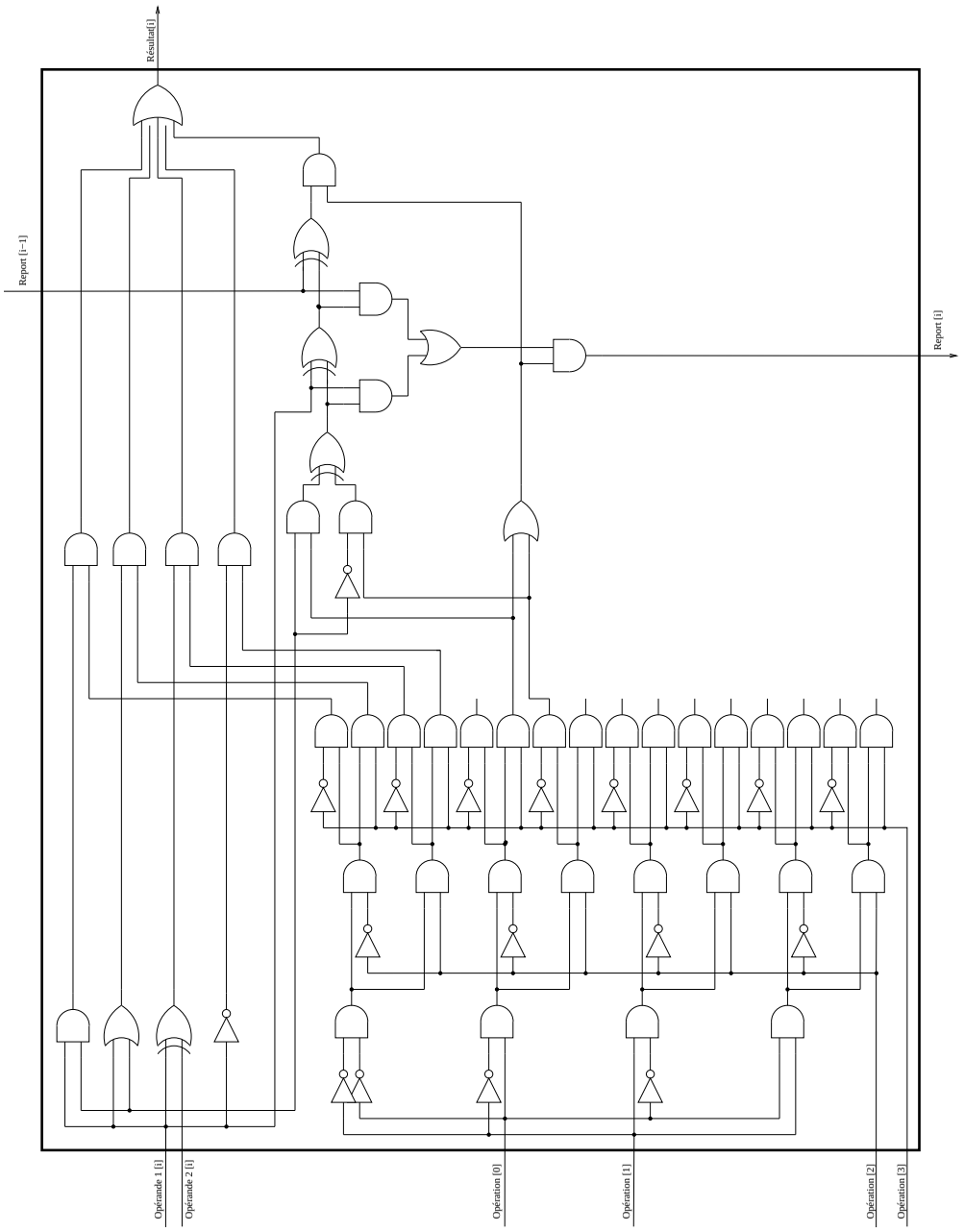


FIG. 7 – L'Unité Arithmétique et Logique (1 bit)