Compte rendu du TP de conception et implémentation MIPS

Omar ROMDHANI
Omar.Romdhani@ensi-uma.tn

Janvier 2019

Résumé

Ce document est le compte-rendu du TP d'implémentation d'un CPU 8-bit basé sur l'architecture MIPS.

1 Introduction

L'objective du TP est d'implémenter un CPU MIPS 8-bit.
Pour cela on va réduire le nombre des registres pour 8 registres, zero, at, v0, v1, a0, a1, a2, a3.
L'UAL est à 8-bit par contre le registre d'instruction sera sur 16 bits.

2 Jeu d'instruction

Comme il a été mentionné, chaque instruction comporte 16 bits distribués comme suit :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CodeOP			rs			rt			rd						

- ☐ Les 3 bits du poids le plus faible sont réservés pour le 'CodeOP' $[15-13]$
- ☐ les 3 bits suivants sont réservés pour le numéro du registre 'rs' $[12-10]$
- ☐ les 3 bits suivants sont réservés pour le numéro du registre 'rt' $[9-7]$
- ☐ les 3 bits suivants sont réservés pour le numéro du registre 'rd' $[6-3]$
- ☐ les 4 bits restants dépendent de l'instruction :
  - — Pour une instruction de type R seuls les 3 bits du poids le plus faibles sont utilisés pour donner le numéro de la fonction.
  - — Pour une instruction de type I les bits $[6-0]$ sont pour la valeur immédiate.

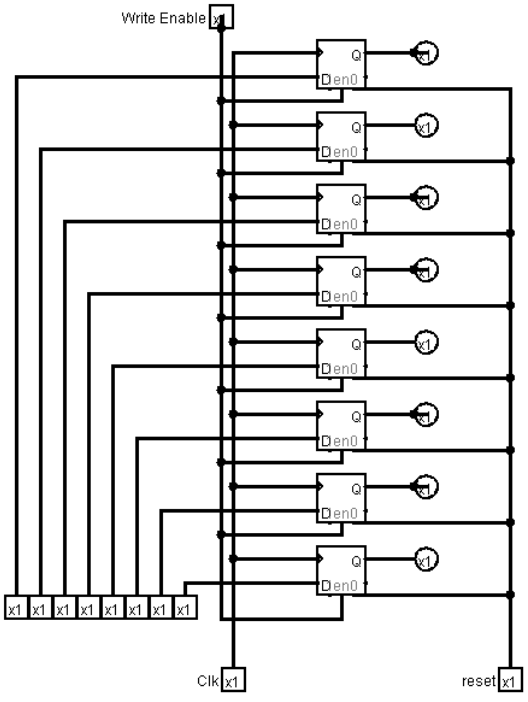
Puisque l'instruction est sur 16 bits, le nombre d'instructions implémentées est réduit.

Les instructions implémentées sont :

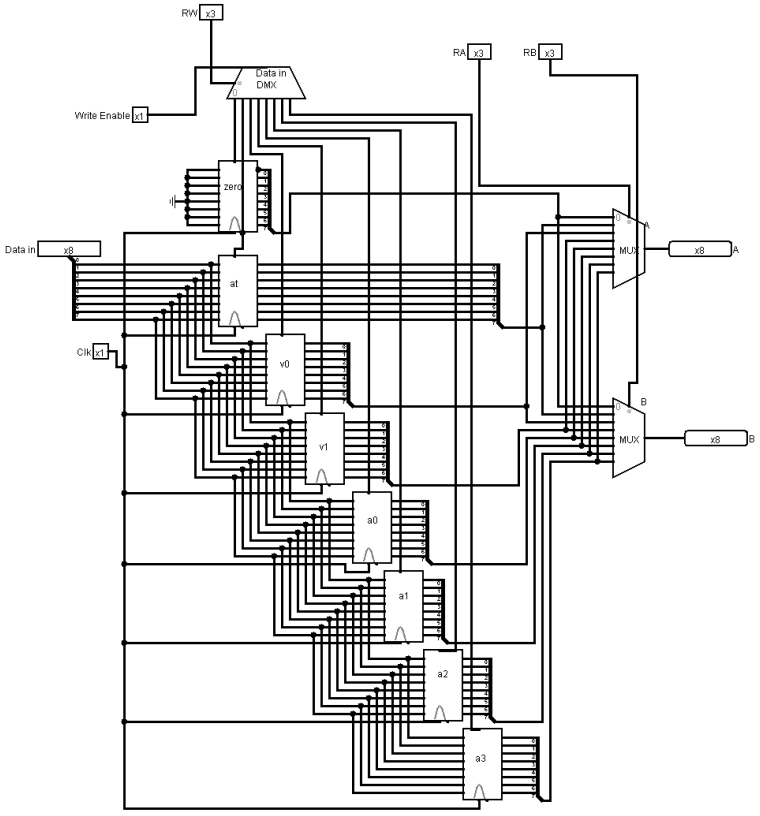
- ☐ **add**, **sub**, **or**, **and**, **stl** (type R).
- ☐ **adi**, **subi**, **ori**, **andi**, **beg**, **lw**, **sw** (type I).

3 Banc de registres

Chaque registre est l'association de 8 bascules D.



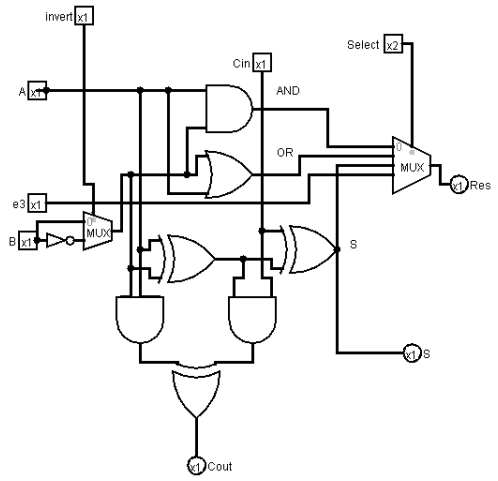
Le banc de registre est la connextion de 8 registres



4 Conception de l'UAL

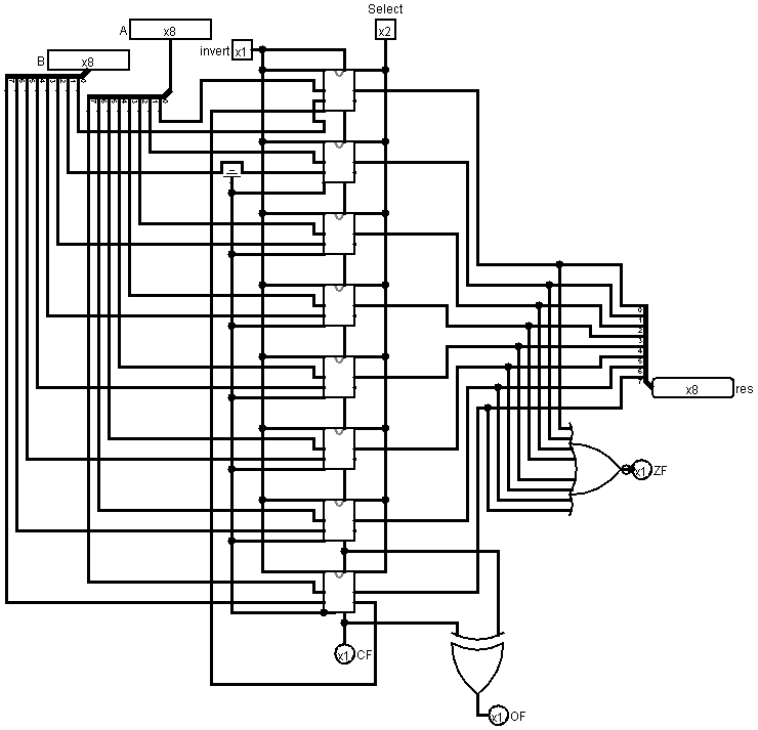
4.1 Conception de l'UAL 1-bit

L'UAL 1-bit comporte le 'ET' le 'OU' et un additionneur exactement comme le UAL vu en cours.



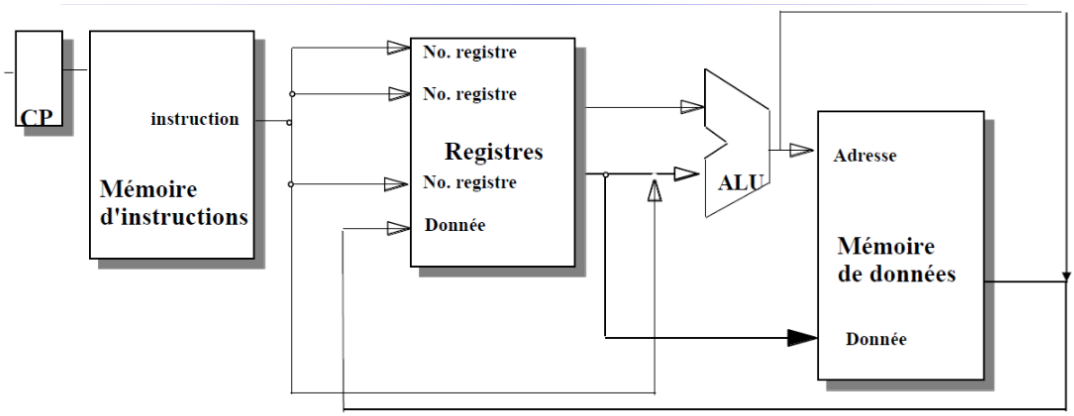
4.2 UAL 8-bit

Elle est obtenue en connectant 8 UAL de 1 bit les unes aux autres.



5 Chemin de données

Une vue abstraite du spis ensemble MIPS :

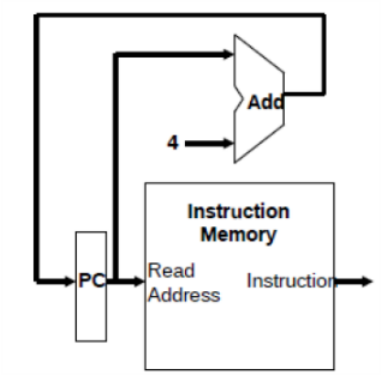


Le Fetch

Le recherche d'instruction se fait en deux étapes :

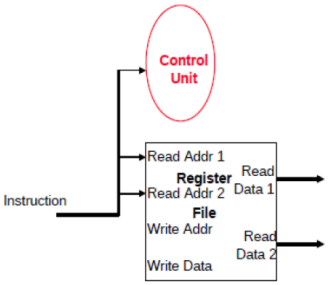
1. Lire l'instruction du registre d'instructions.

2. Mettre à jour le registre PC



Le Decode

Décoder consiste à envoyer l'opcode **(IR[15-13])** à l'unité de contrôle et les signaux de lecture de registres (signaux d'adresse de registres contenu dans l'instruction).



Exécution

- ☐ Instructions de type R(add, sub, or, and, stl)
- ☐ Instructions de type I(adi, subi, ori, andi, beg, lw, sw)

PS : L'instruction J (jump) n'est pas implémentée.

5.1 Instruction de type R

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
000			rs			rt			rd				func		

Le opcode pour les instructions de type R est 000.

Le code func pour chaque instruction est comme suit :

- ☐ add : 000
- ☐ sub : 0$\underline{01}$
- ☐ and : 011
- ☐ or : 010
- ☐ slt : 1$\underline{01}$

Exemple :

```
add $2,$1,$1
```

opcode rs rt rd func

⇒ 000 001 001 010 0 000

5.2 Instruction de type I

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode			rs			rt			immediate						

les Opcodes pour les instructions sont les suivants :

- ☐ addi : 100
- ☐ subi : 1$\underline{01}$
- ☐ andi : 111
- ☐ ori : 110
- ☐ beq : 0$\underline{01}$
- ☐ lw : 010
- ☐ sw : 011

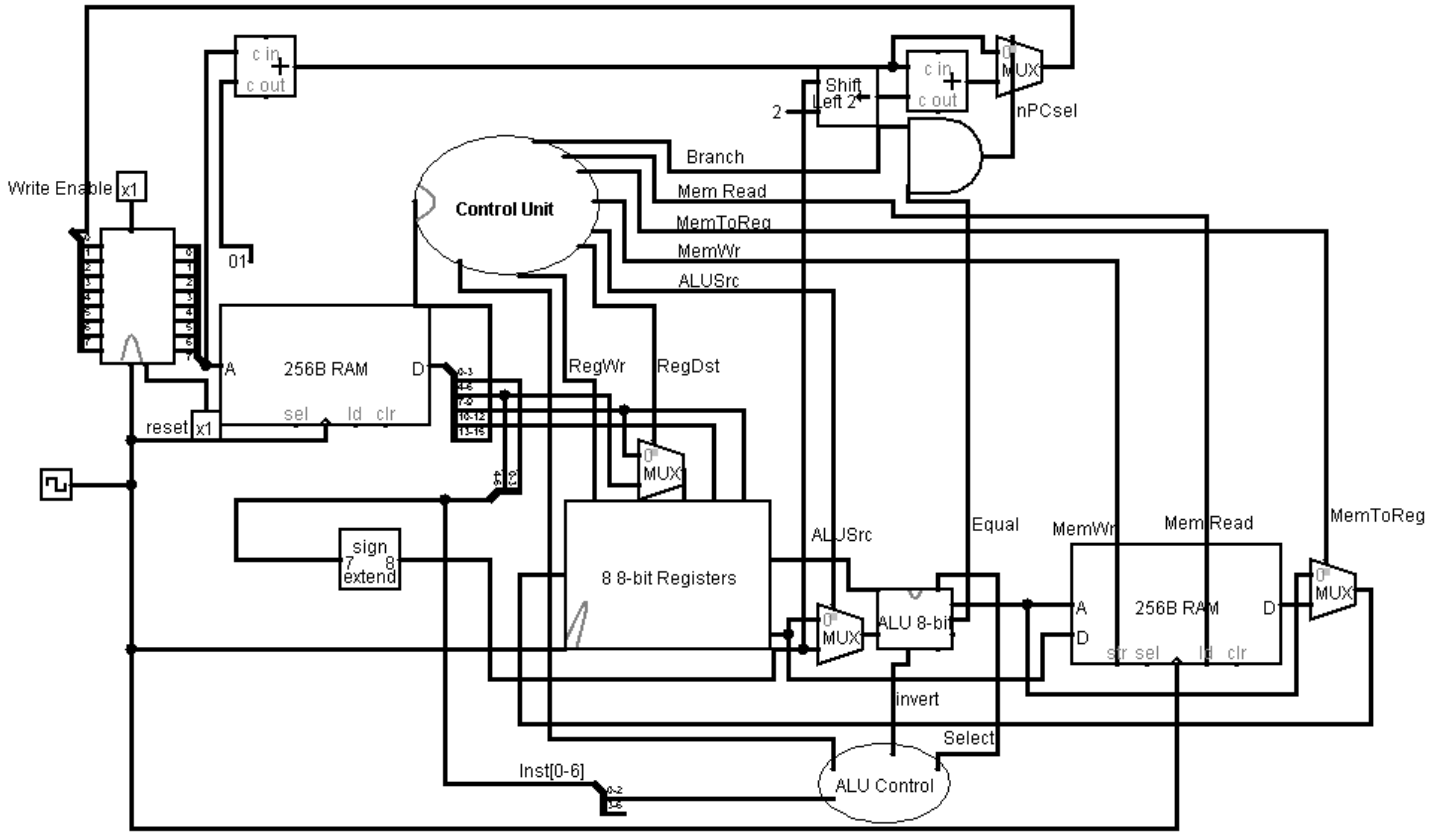
Exemple :

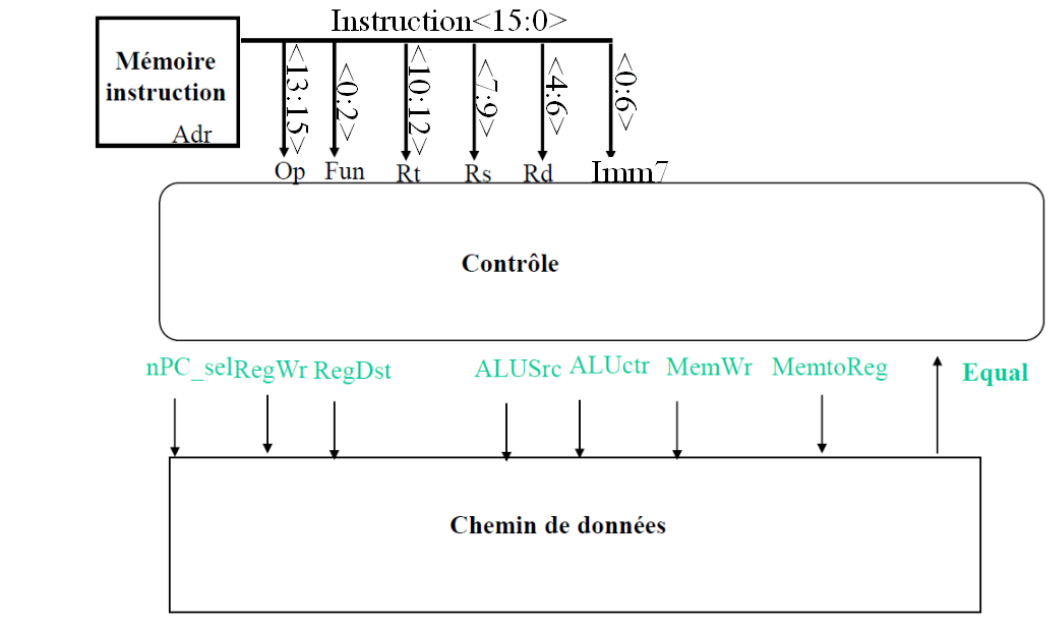
```
add $2,$1,4
```

opcode rs rt immédiate

$\Rightarrow$ 100 001 010 0000100

5.3 chemin de données





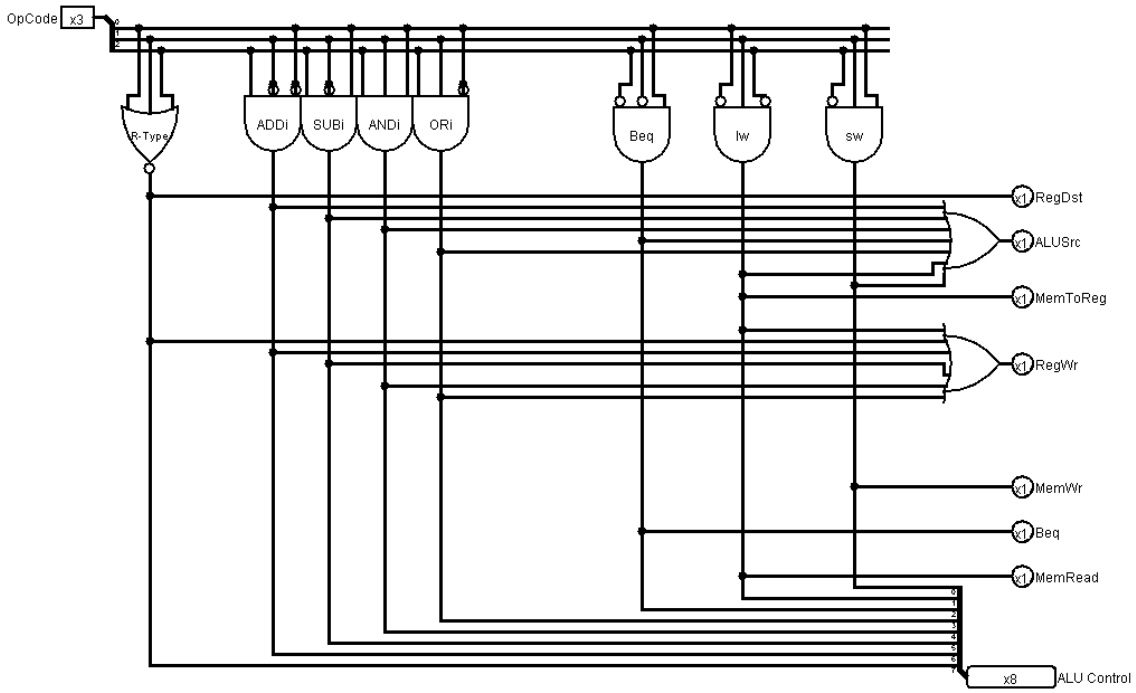
6 Unité de Contrôle

Table des Signaux de Contrôle :

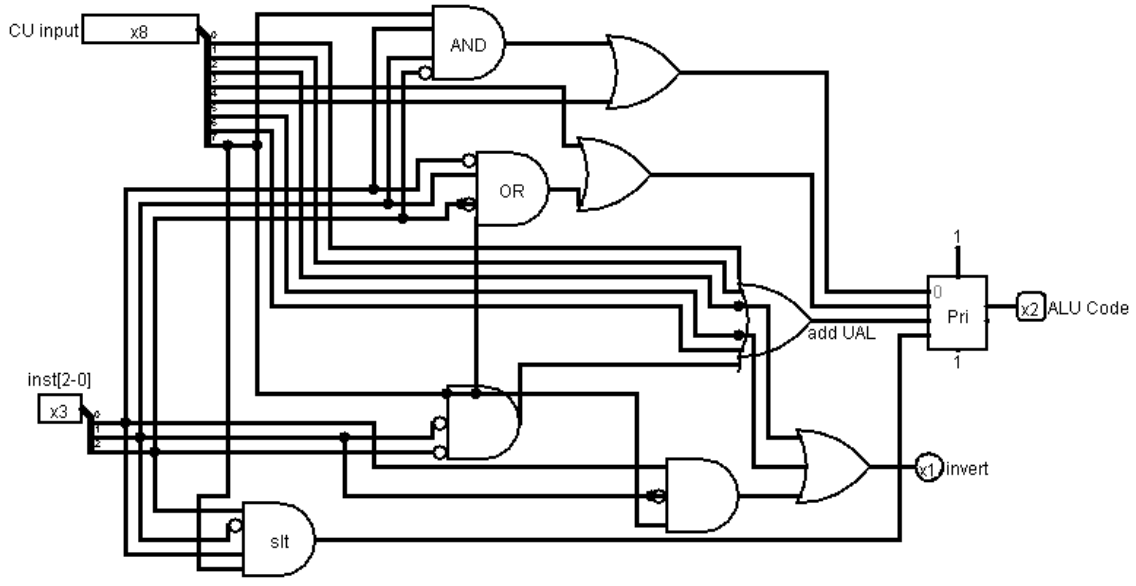
	add	sub	and	or	slt	addi	subi	ori	andi	lw	sw	beq
RegDst	1	1	1	1	1	0	0	0	0	0	x	x
ALUSrc	0	0	0	0	0	1	1	1	1	1	1	1
MemToReg	0	0	0	0	0	0	0	0	0	1	x	x
RegWr	1	1	1	1	1	1	1	1	1	1	0	0
MemWr	0	0	0	0	0	0	0	0	0	0	1	0
MemRead	0	0	0	0	0	0	0	0	0	1	0	0
Branch	0	0	0	0	0	0	0	0	0	0	0	1
invert	0	1	0	0	1	0	1	0	0	0	0	1
Select	2	2	0	1	3	2	2	1	0	2	2	2

PS :le signal 'select' est pour l'opération effectuée par le ALU

6.1 Unité de Contrôle



6.2 Contrôle de l'UAL



7 Test

Dans la section suivante, on test un simple code :

```
addi $1, $0, 4                ;100 000 001 0000100
add $2, $1, $1                ;000 001 001 010 0 000
slt $3, $1, $2                ;000 001 010 011 0 101
sw $3,100($0)                        ;011 000 011 1100100
lw $4, 100($0)                ;010 000 100 1100100
```

En Hexadecimal :

8084

04a0

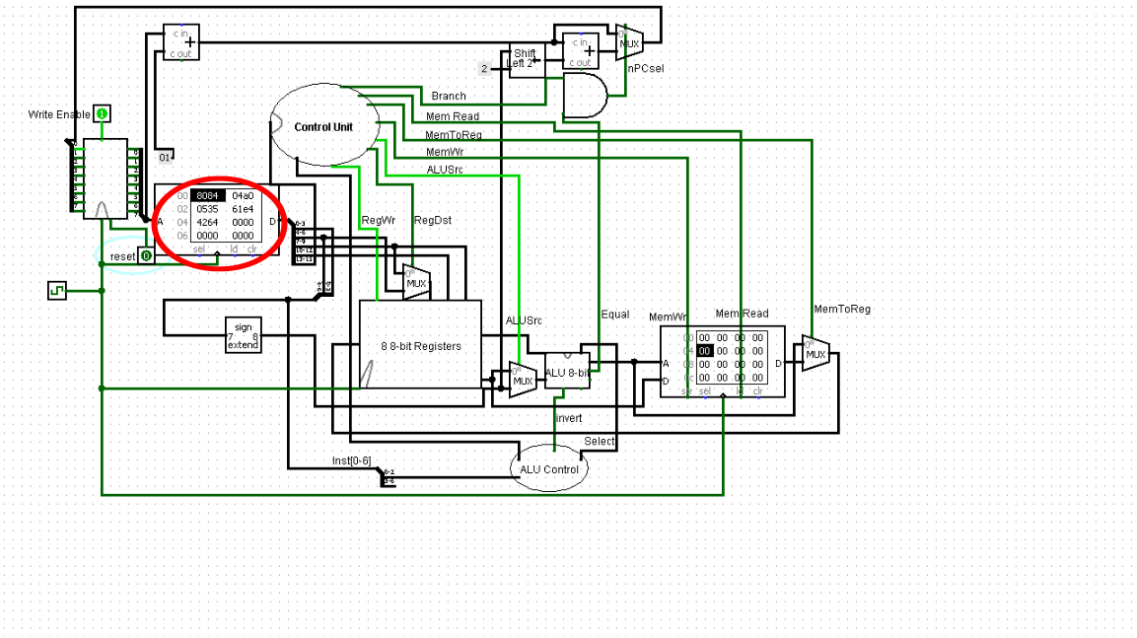
0535

61e4

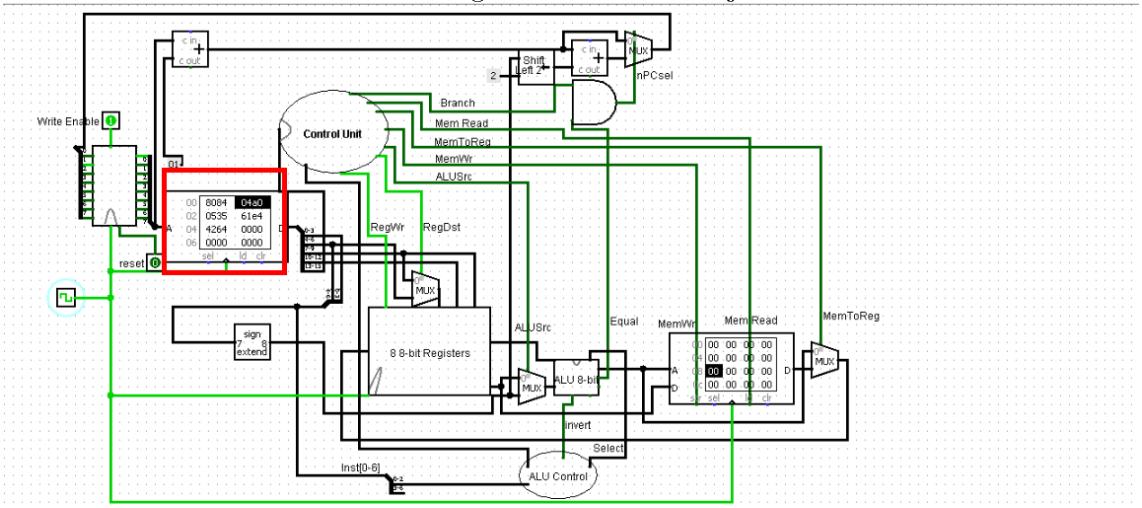
4264

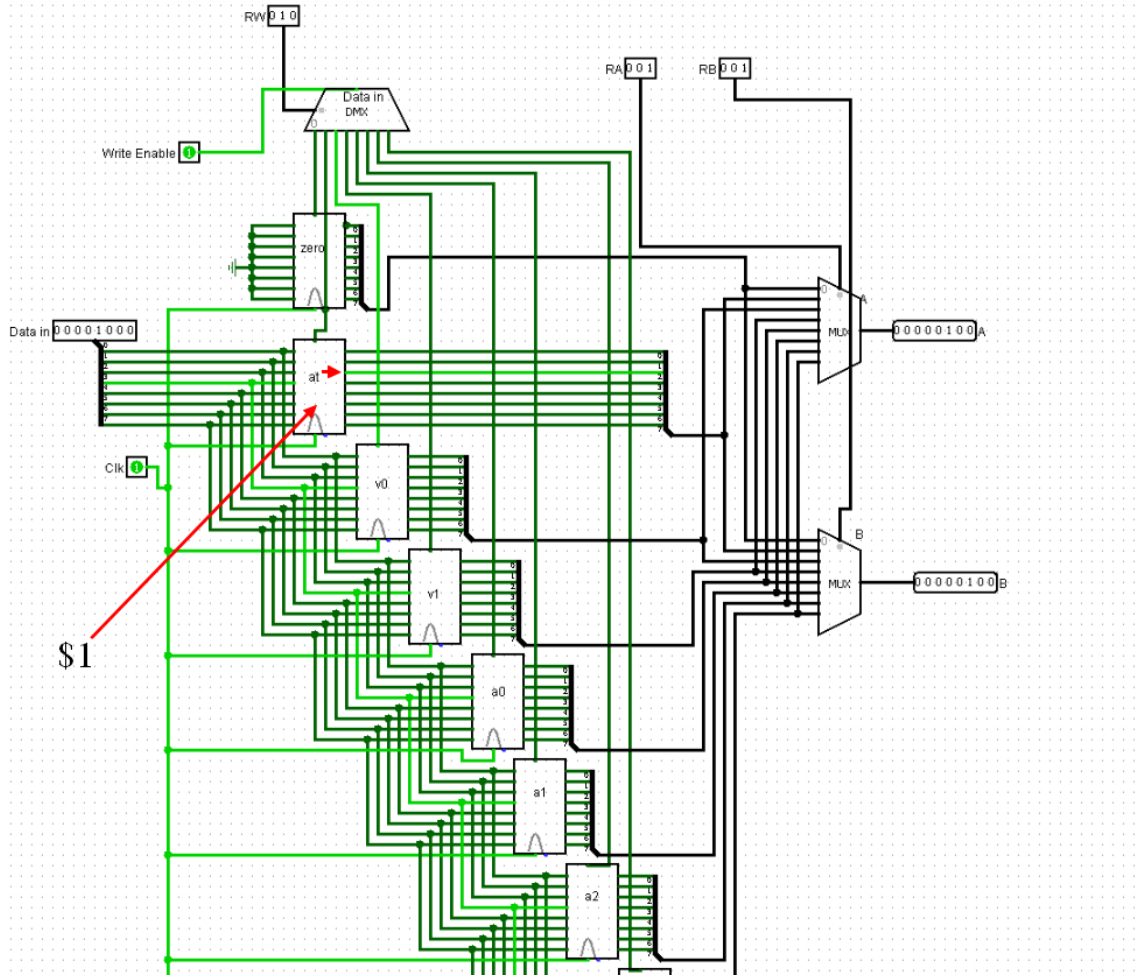
En suite des captures d'écran pour l'exécution :

PC est à zéro, la 1^{ere} instruction est 8084 (addi $1,$0,4)



Au front montant suivant le registre $1 est mis-à-jour.

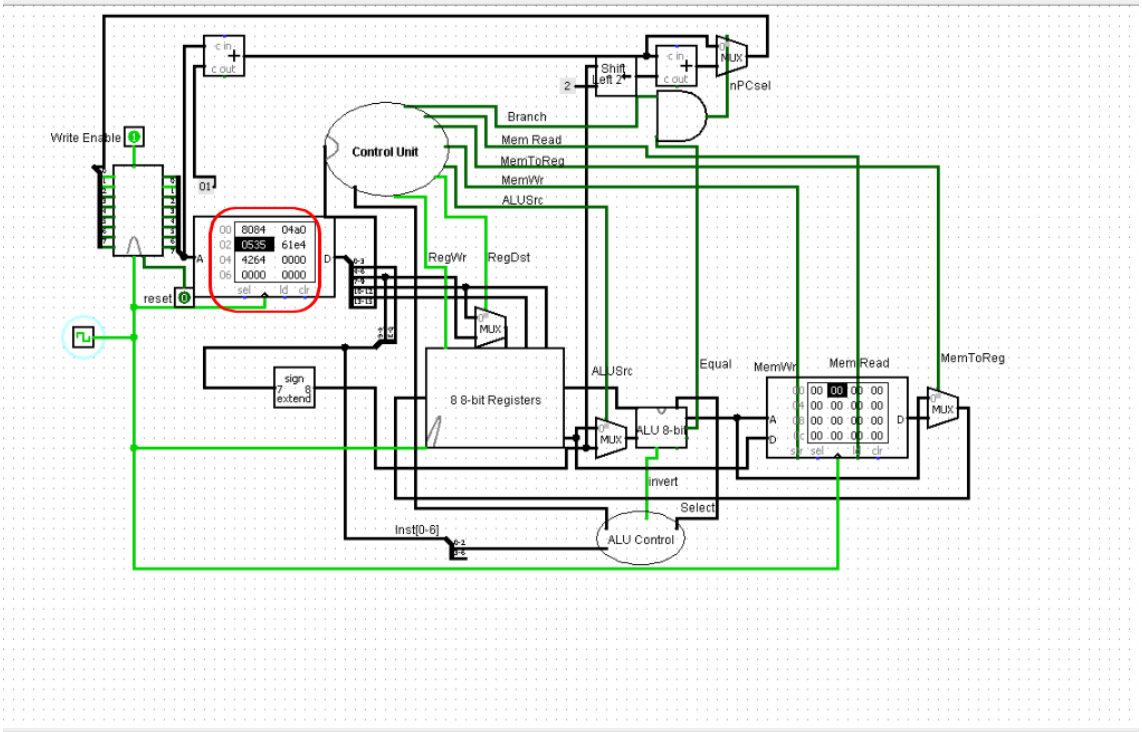


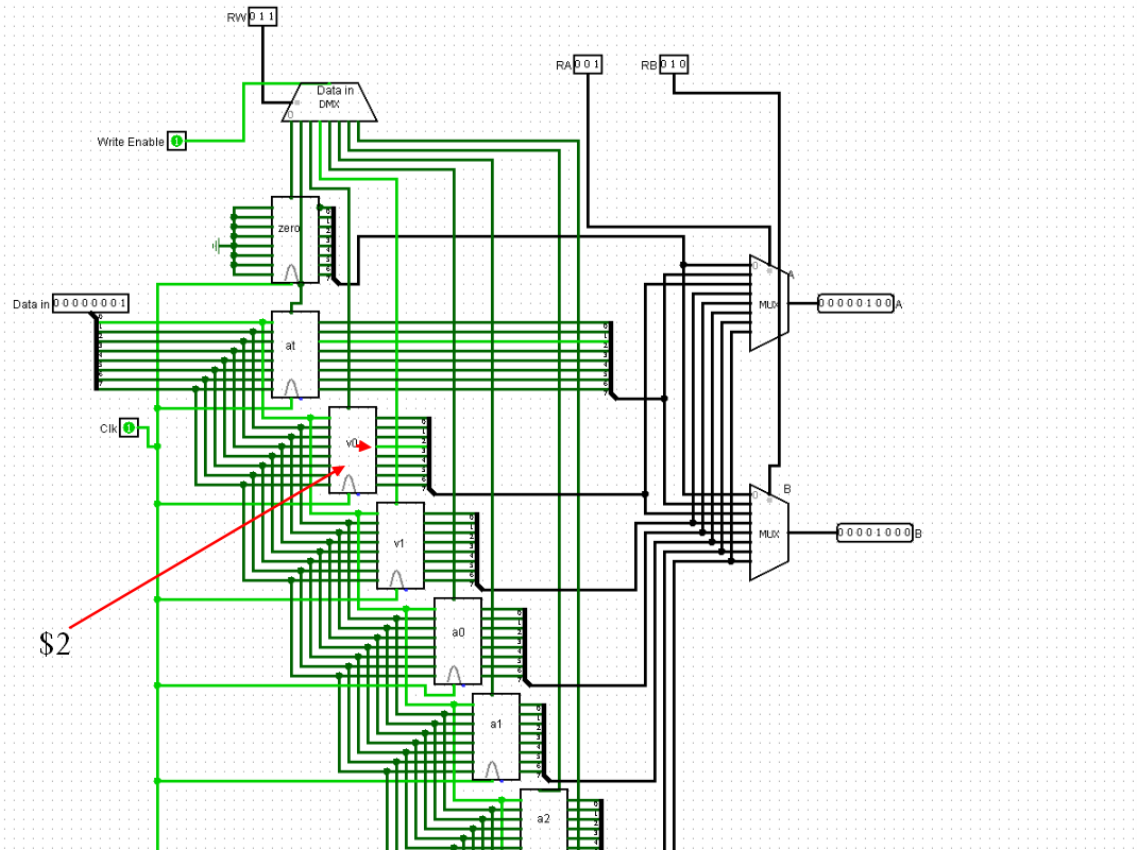


Maintenant, PC pointe sur la 2^{eme} case dans le Registre d'instruction IR.

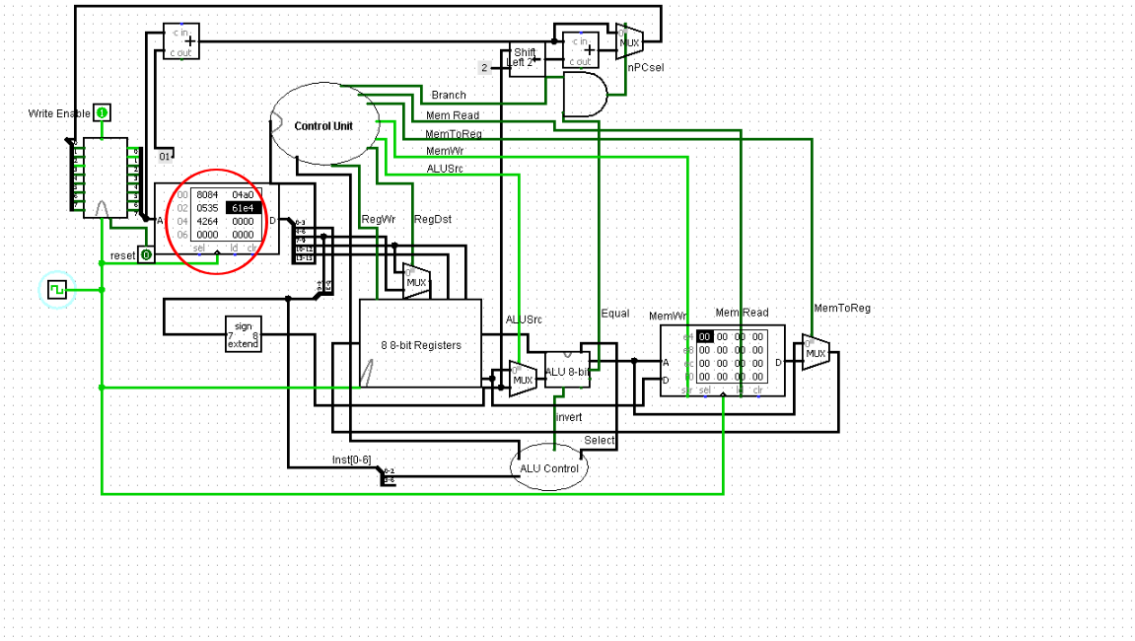
L'instruction est 04a0 (add $2,$1,$1).

Au front montant suivant, $2 \Leftarrow \$1 + \$1$.

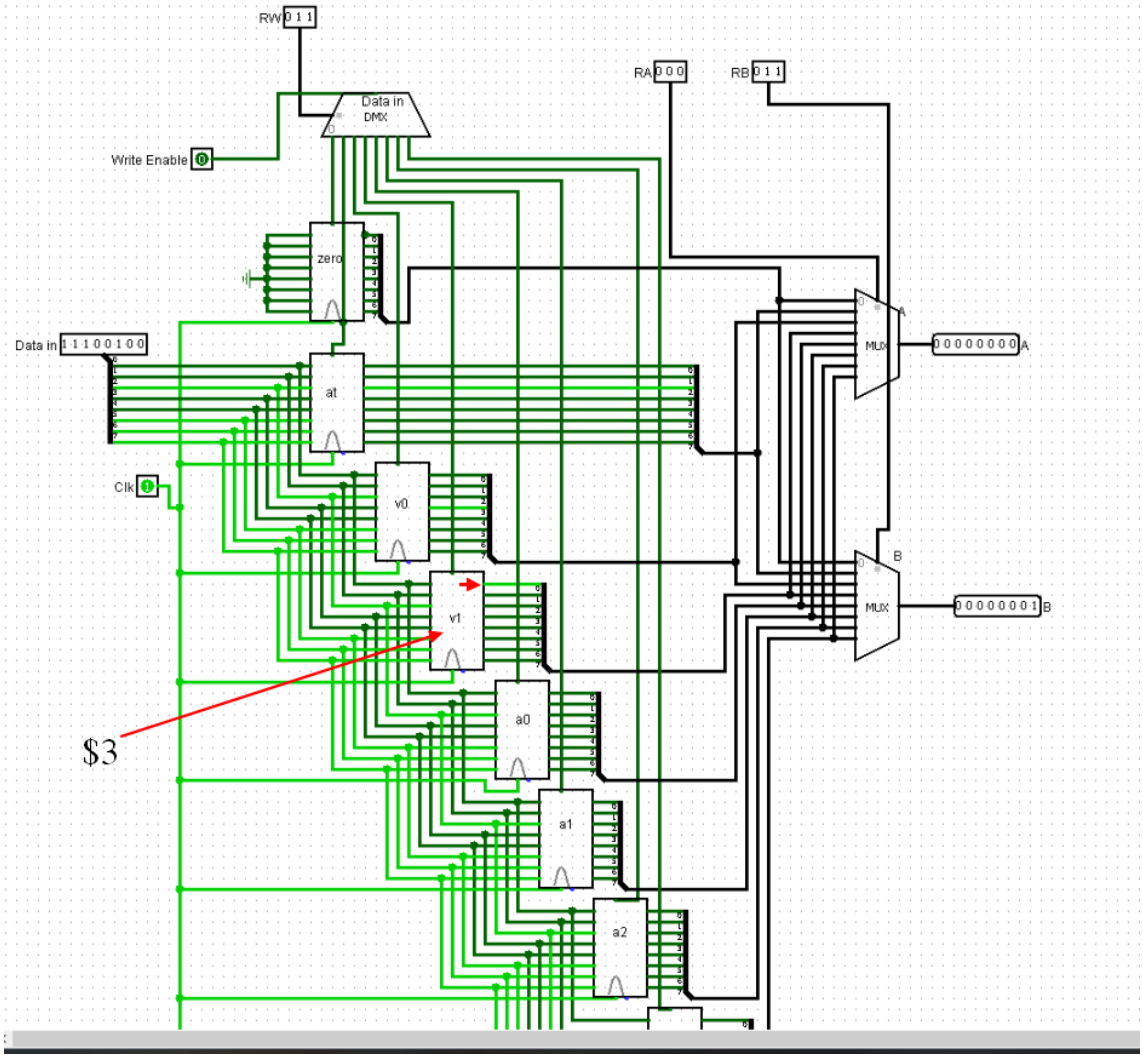




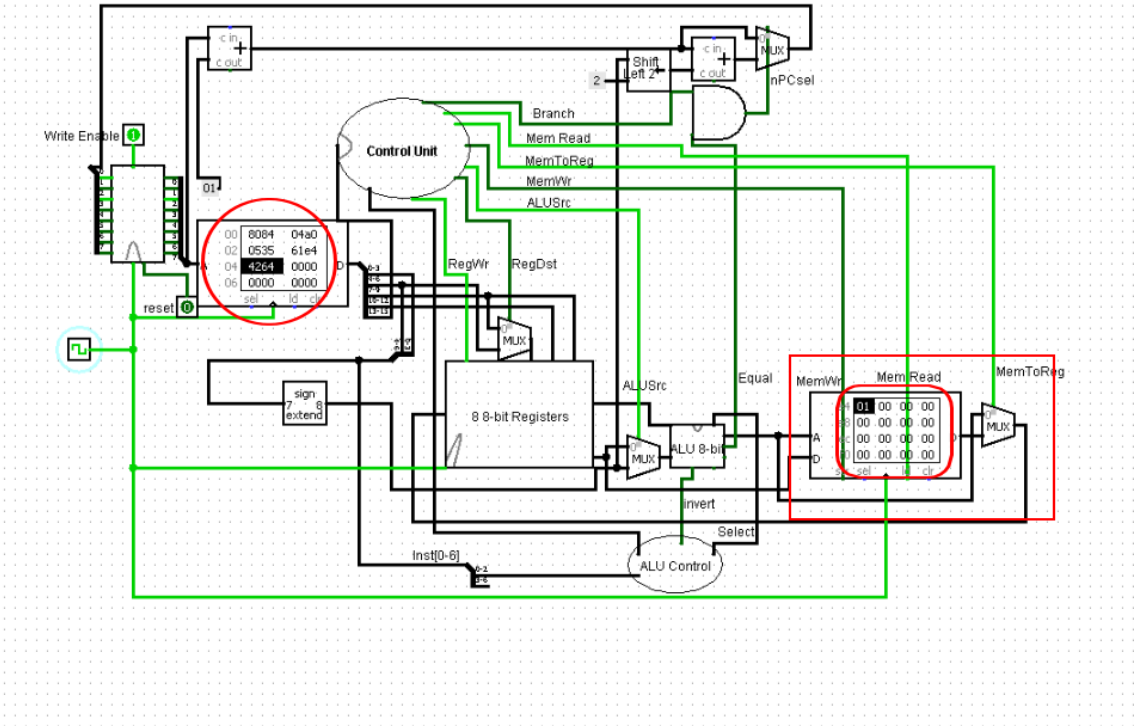
En ce moment le CPU est entrain d'exécuter la 3^{eme} instruction 0535 (slt \$3,\$1,\$2). De même, la valeur de \$3 ne sera mis-à-jour qu'après le front montant suivant.



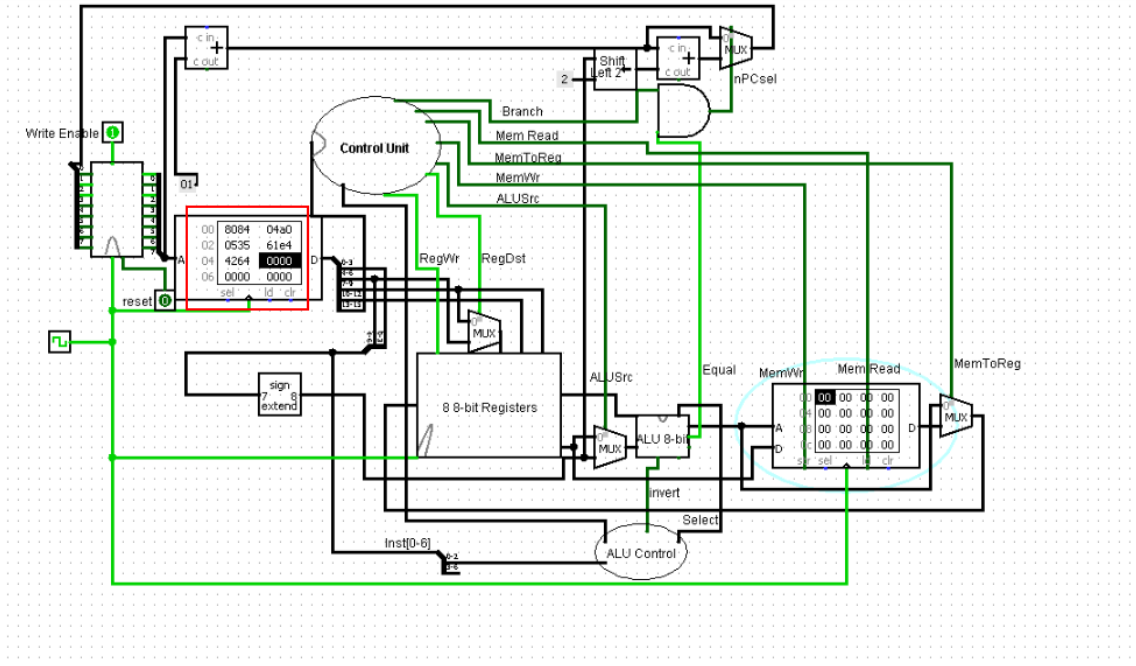
Et puisque \$1 $<$ \$2 , \$3 $\Leftarrow$ 1.


RW 0 1 1
RA 0 0 0
RB 0 1 1
Data in DMX
Write Enable
zero
Data in 1 1 1 0 0 1 0 0
at
Clk
v0
v1
a0
a1
a2
MUX
0 0 0 0 0 0 0 0 A
B
MUX
0 0 0 0 0 0 0 1 B
$3

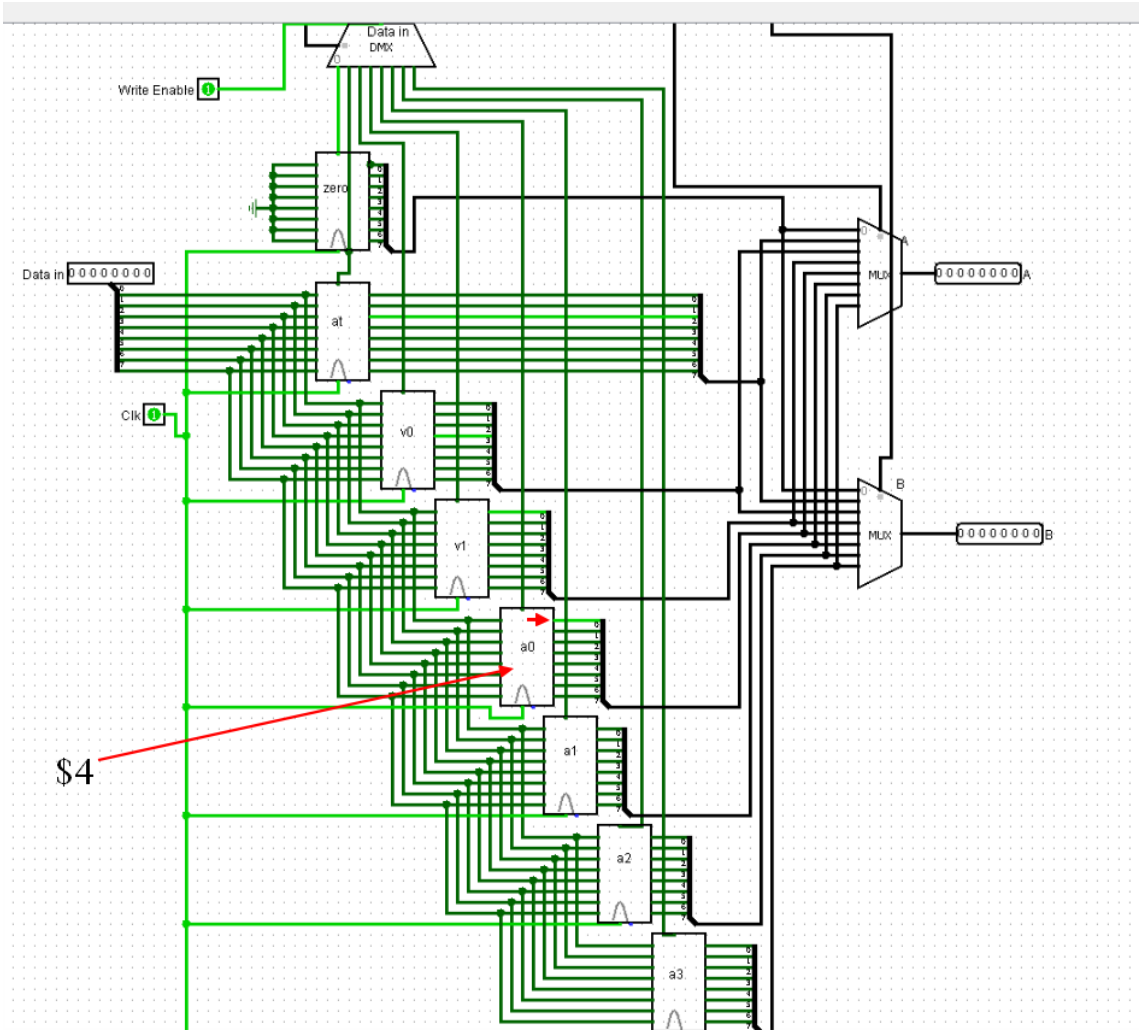
Ensuite l'instruction suivante est déja executée 61e4 (sw $3,100($0)).
On peut voir la valeur 1 dans la case mémoire d'adresse 'e4'.



La dernière instruction est maintenant exécutée.4264 (lw $4,100($0)).



Le registre $4 à la valeur 1 maintenant.


Data in
DMX
Write Enable
zero
Data in
00000000
at
Clk
v0
v1
a0
a1
a2
a3
$4
A
MUX
00000000
A
B
MUX
00000000
B