# ARMv7

## Cortex M0

Si3 - Groupe 1

Antoine STEYER
Axel AIELLO
Gabriela CAVALCANTE DA SILVA
Raquel LOPES DE OLIVEIRA
Thomas JALABERT
Loïc ROSE

9 janvier 2016

# Schedule

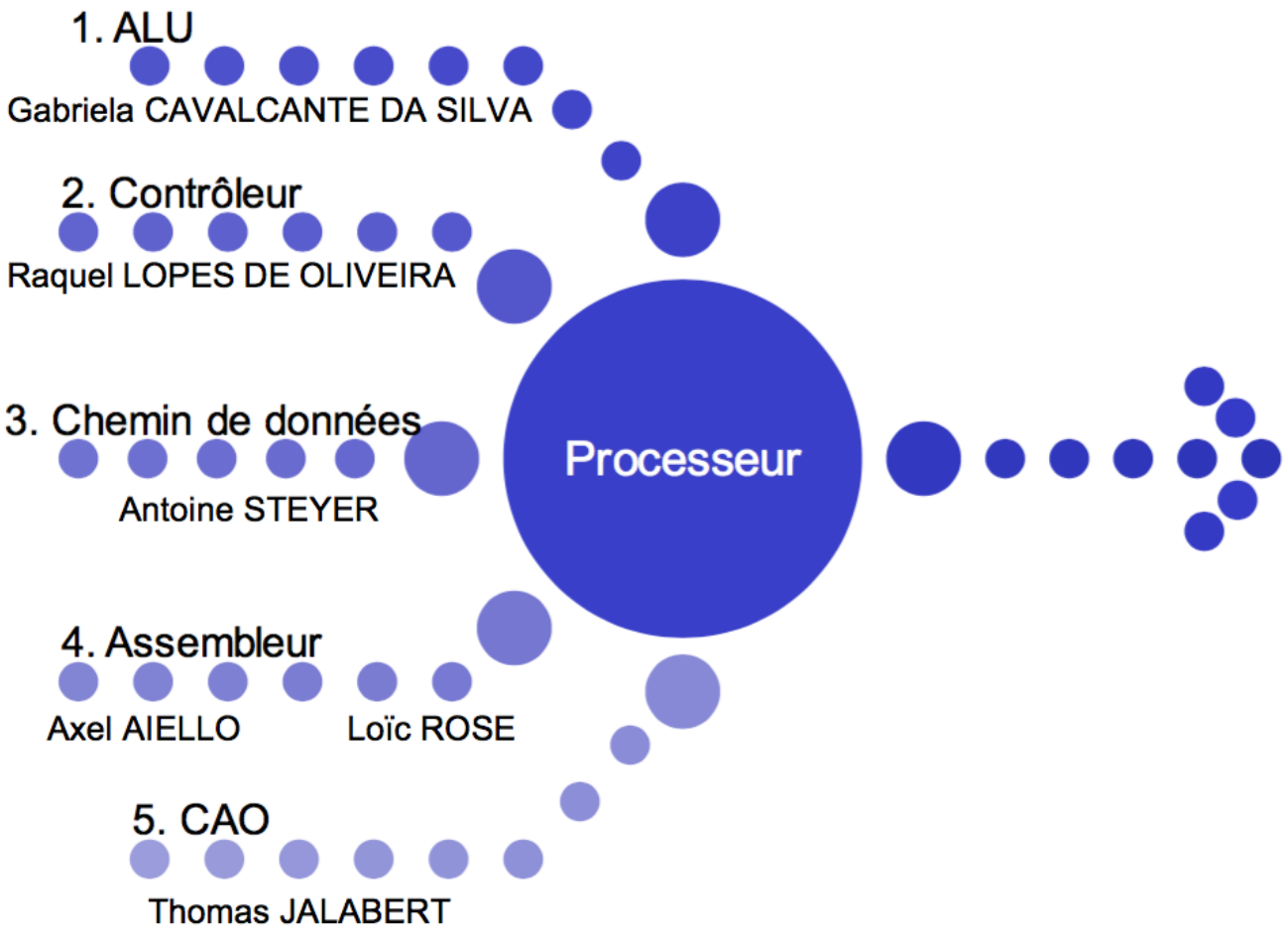



Figure: Component Schema

- Réaliser les blocs d'opérateurs arithmétiques et logiques
- Générer les Flags

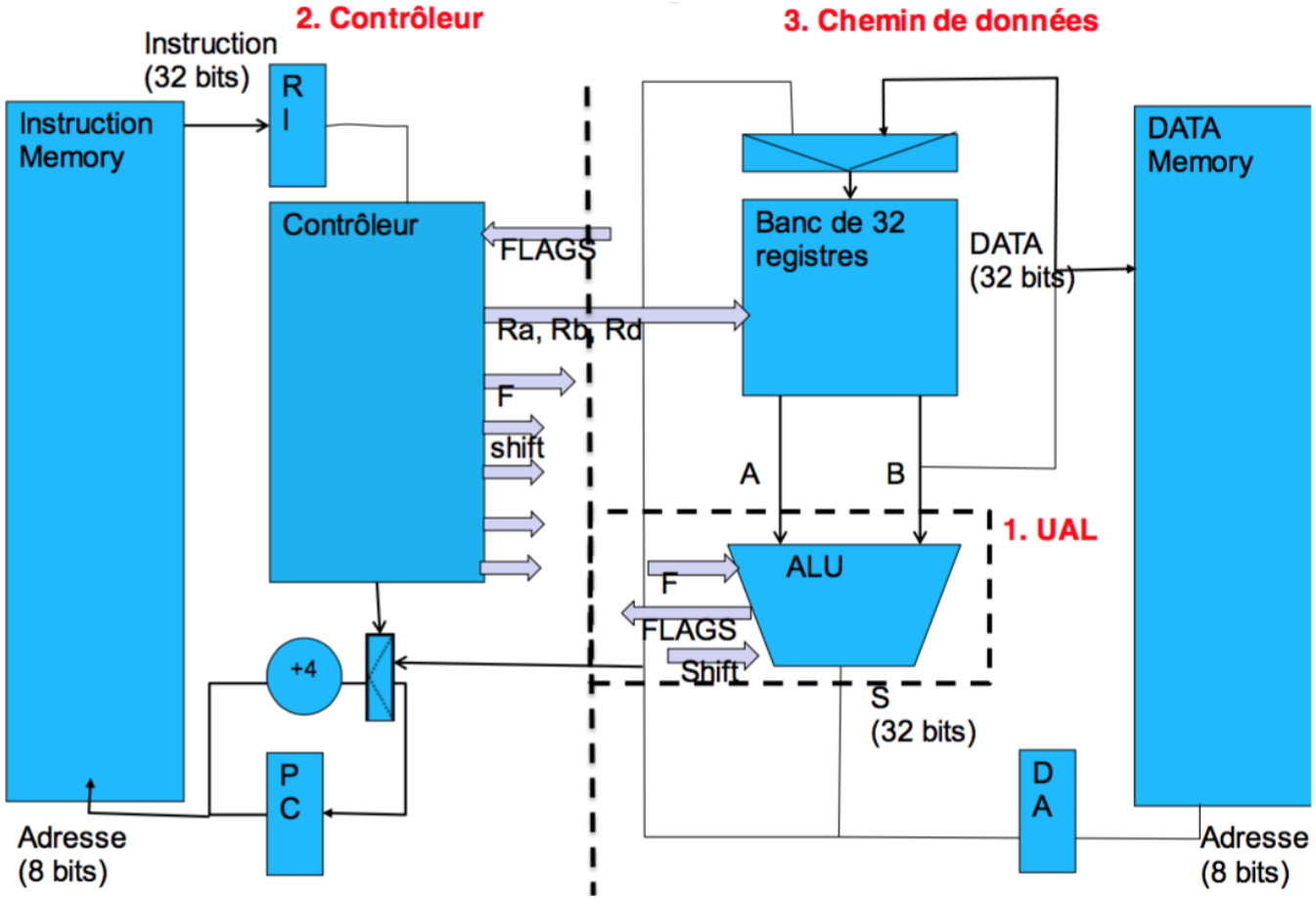



Figure: Architecture générale

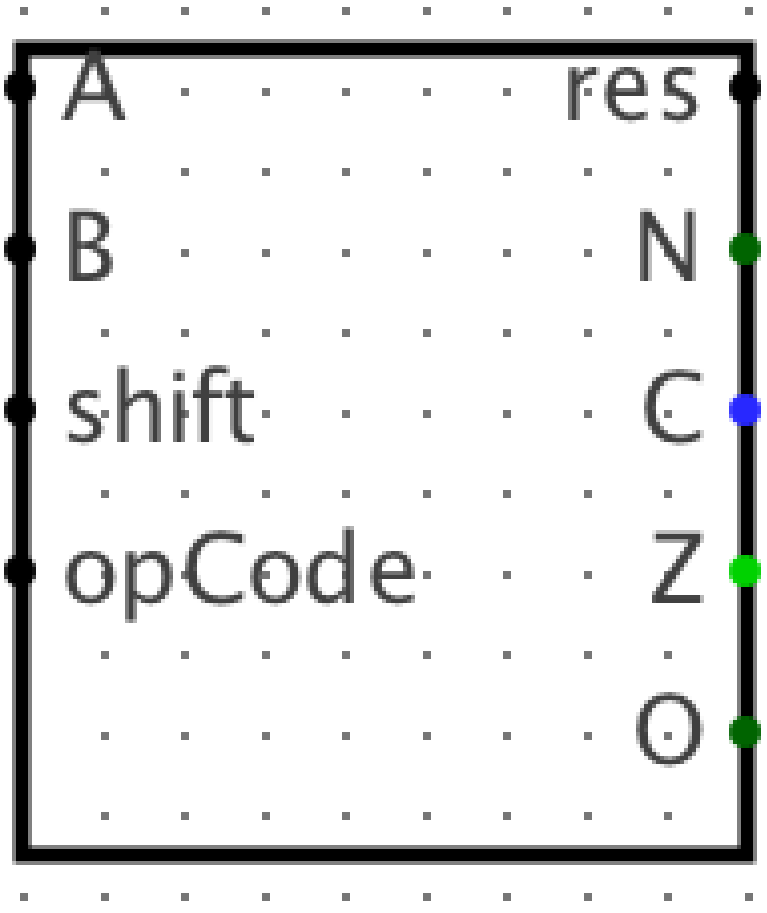



Figure: ALU Component

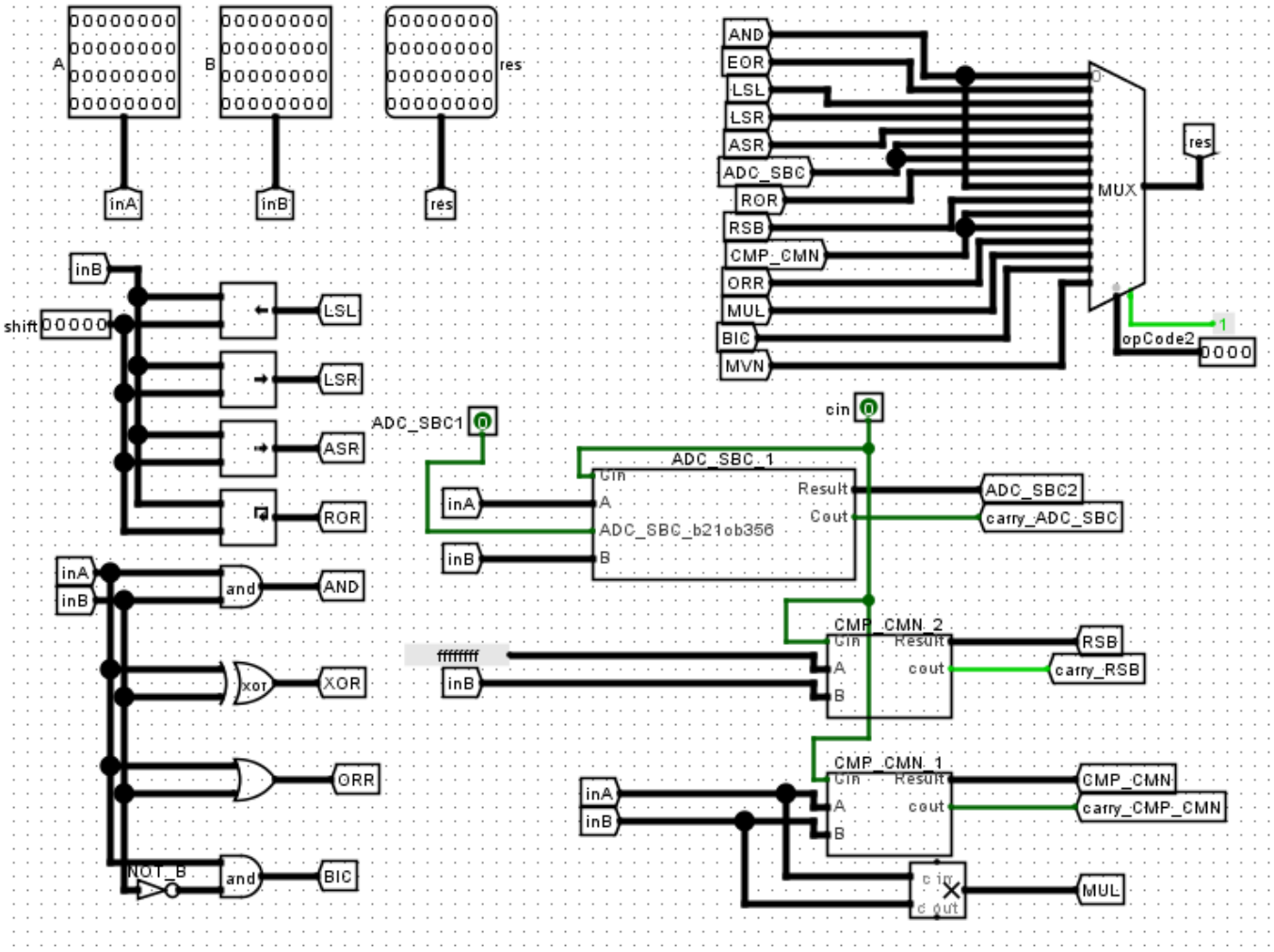



Figure: Detail de fonctionnement

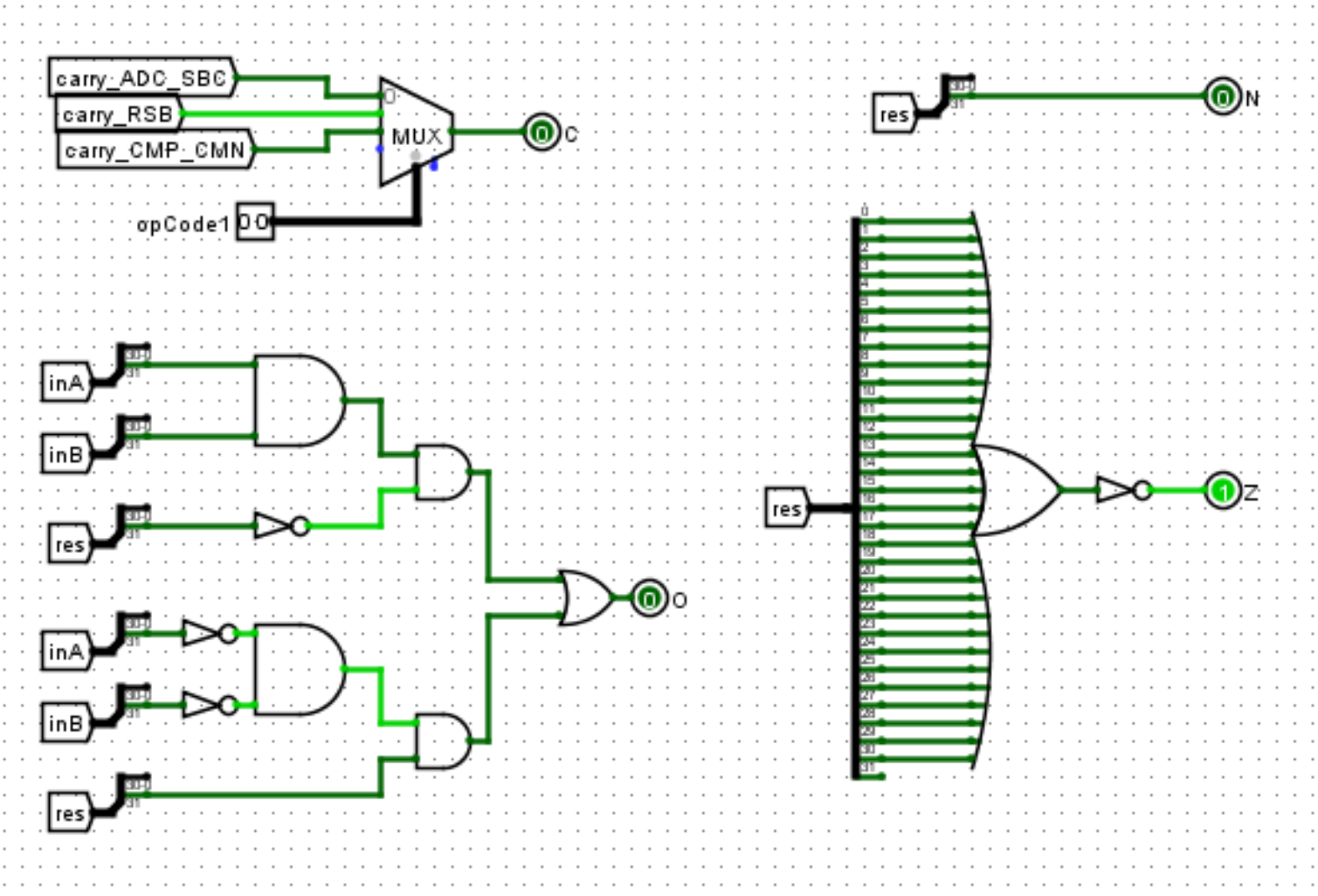
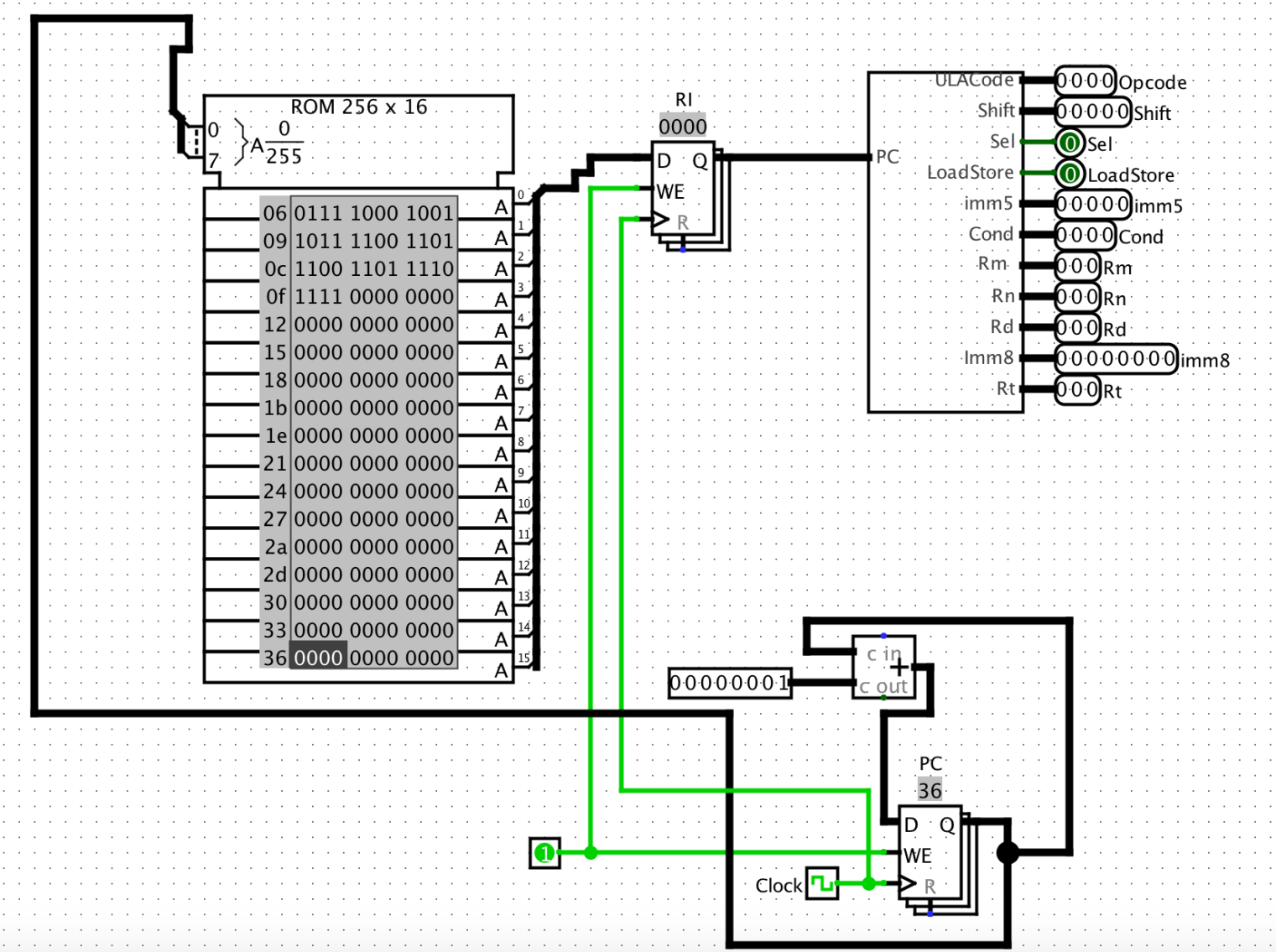
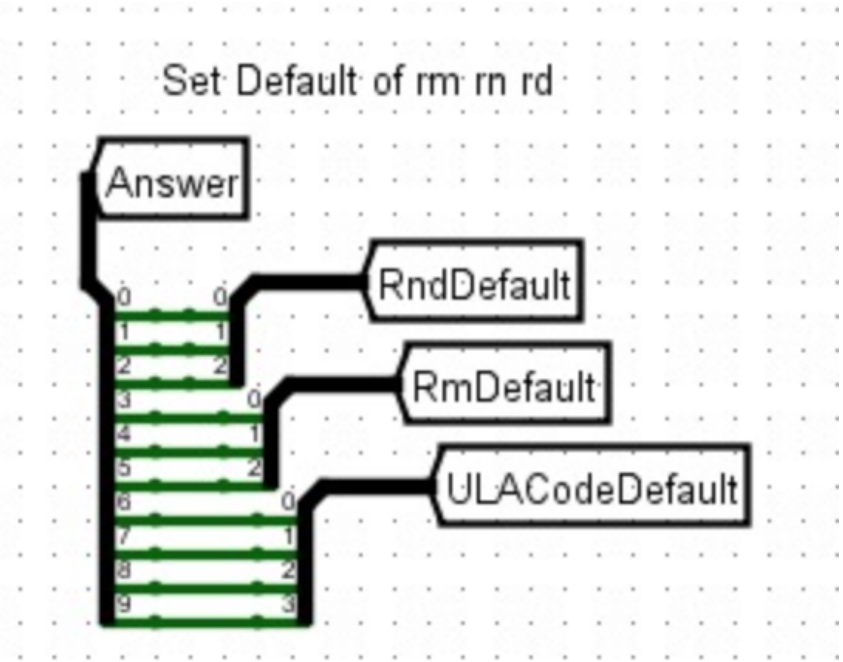



Figure: Detail de fonctionnement

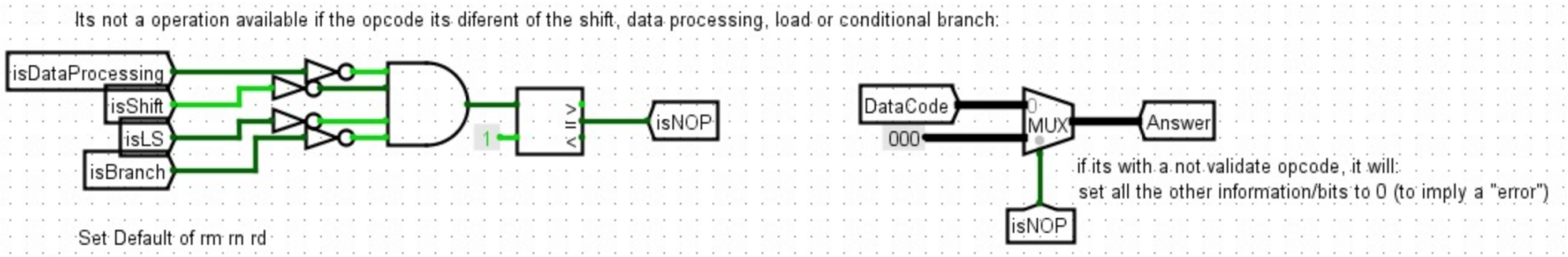
# Contrôleur
## Main



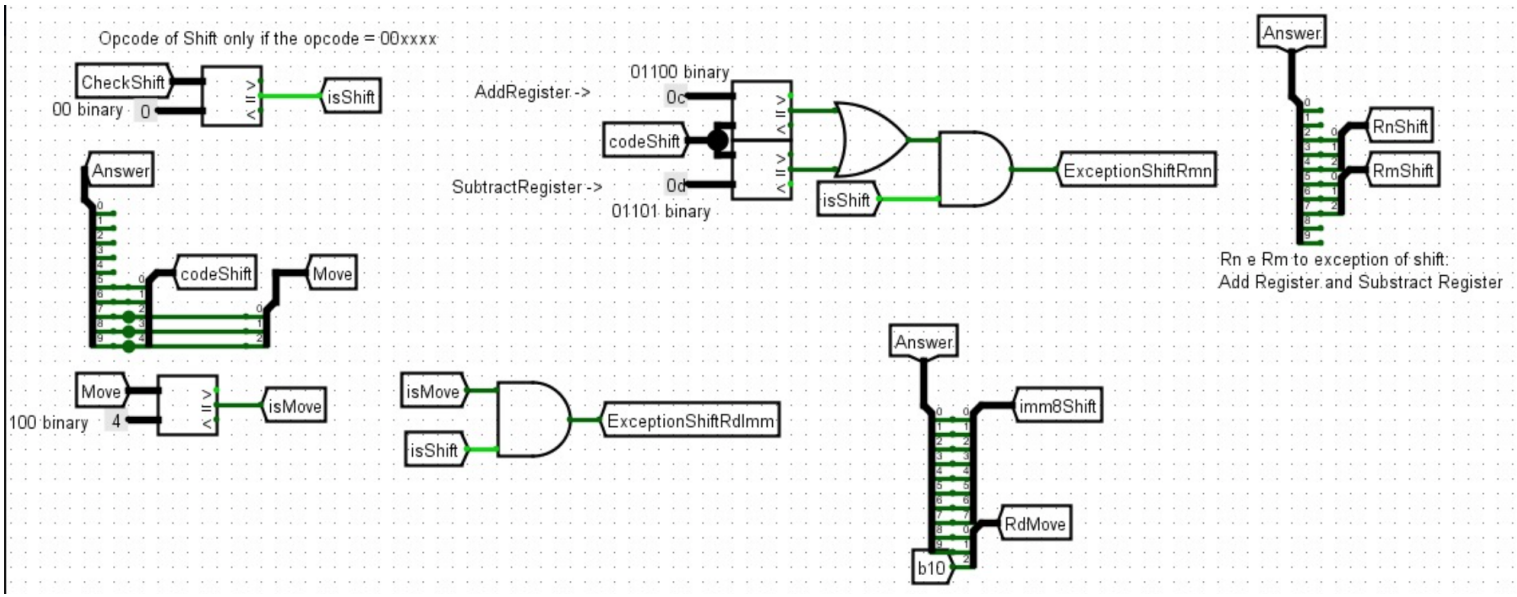
Tunels were used a lot to make the visualisation and verifications easier. In the image below it's possible see the output of Rm, Rn, Rd and opcode of the majority of instructions, so they were seted as the default ones.



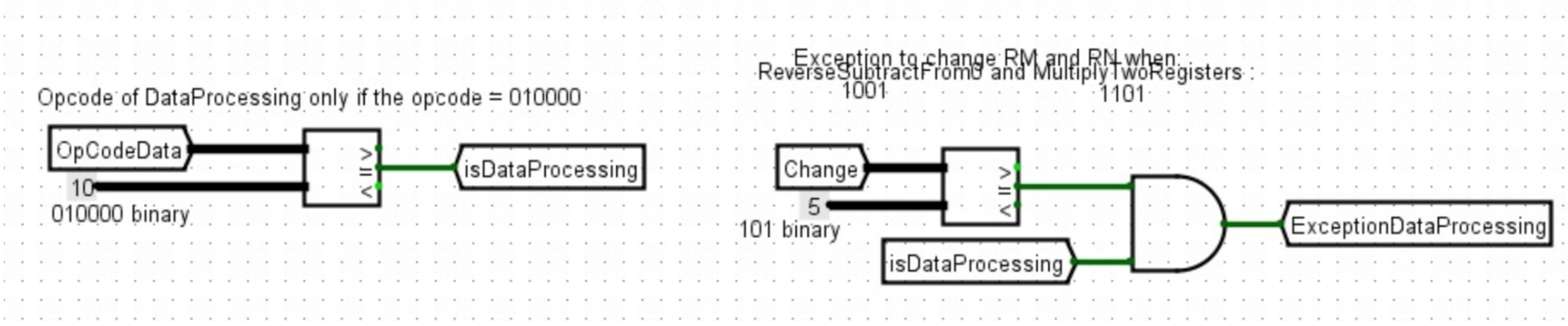
It was made a verification that the opcode it's not one of the four instructions : shift, data processing, load or conditional branch ; he changes the others bits to O. This will "imply a 'error'".



For the Shift's instructions the RM is always the bits 543(same as the default). But there is 3 instructions that this can change : The add register and subtract register has as RM the 876 and the RN as 543.The other situation its the move where the RD will not be the default (210), it will 10 9 8 and it will be send the imm8.

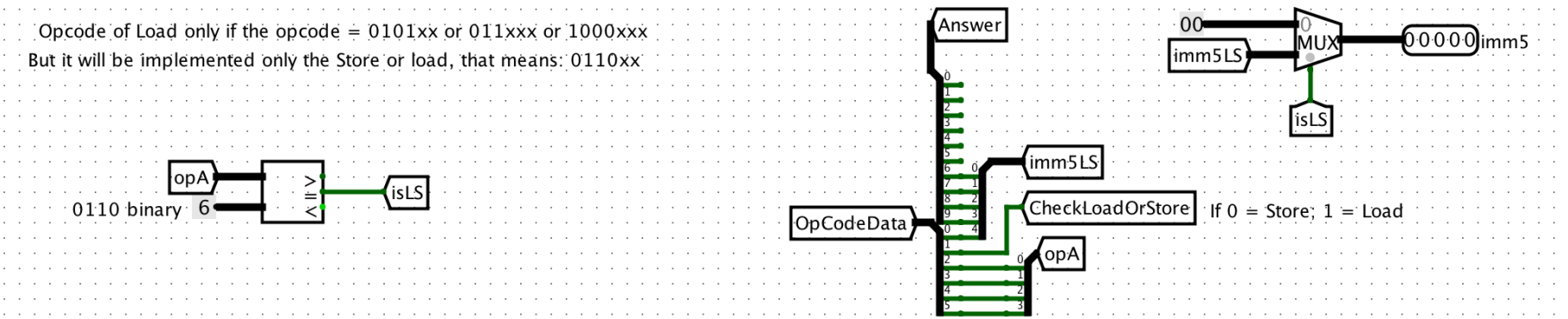


For the Data Processing the only situation the RM and RN changed its to reverse subtract from 0 (1001) and Multiply two registers (1101), that means if the b9 = 1 ; b7 = 0 and b6 = 1 (they are took from the tunel *change* for verification).

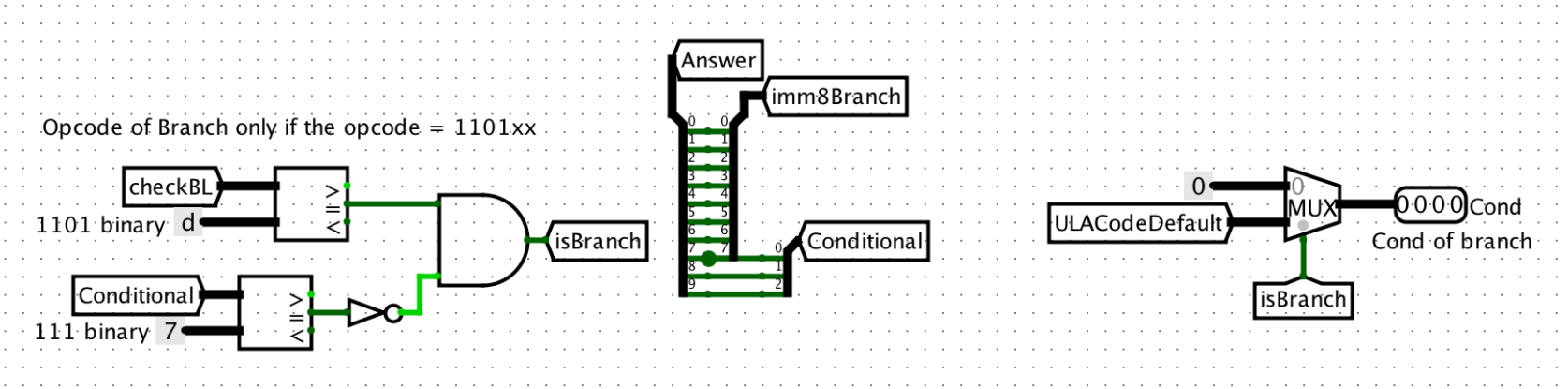


# Contrôleur
## Tunel as flags - Load/Store

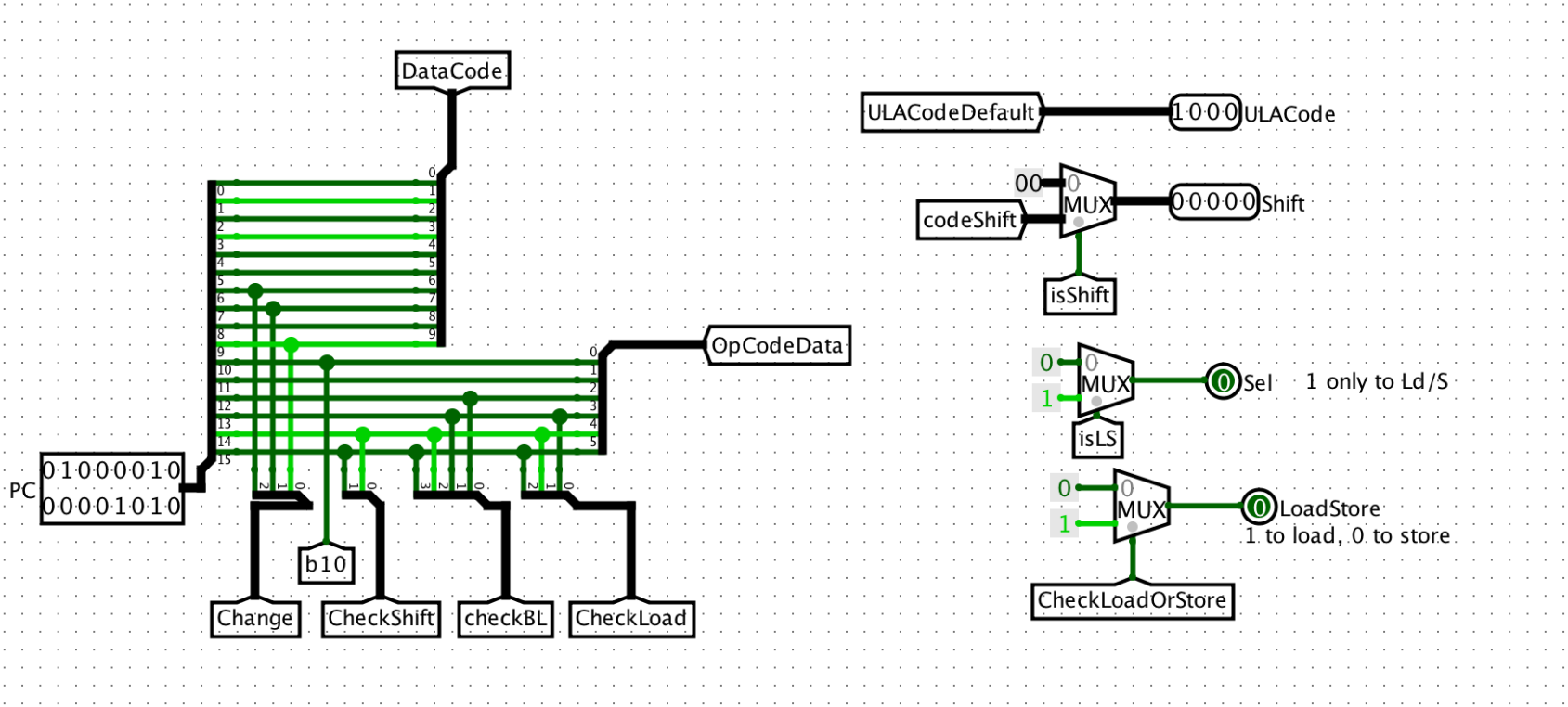
For the loadStore the tunel *isLS* it's only 1 when the verifications of opA is 1.The Rt its the same of RD (default), the RN it's the RM(default) and its created a imm5.



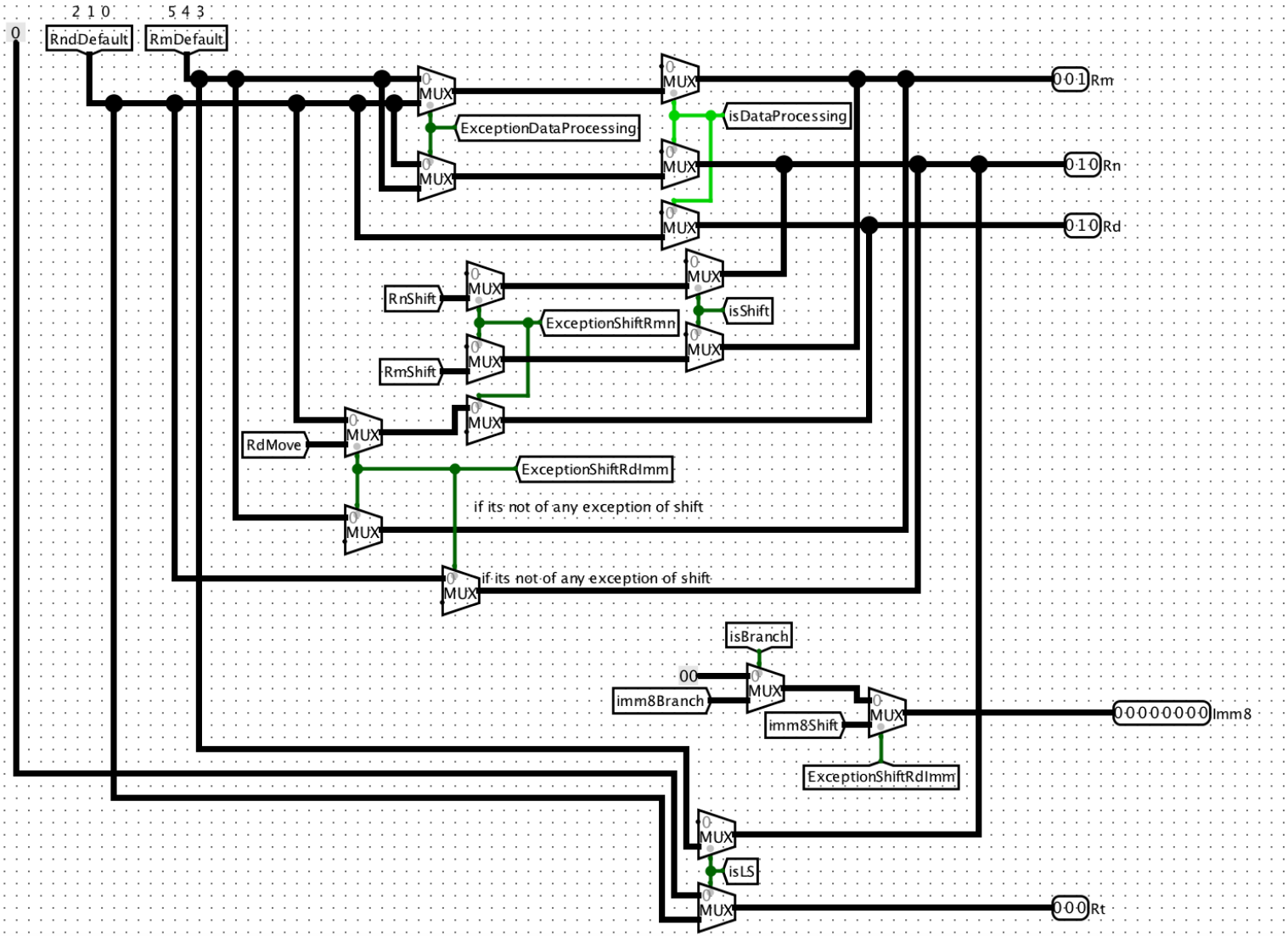
Only one instruction of branch had to be developed : conditional branch. So the tunel isBranch is only 1 when b15=1, b14=1,b13=0,b12=1,b11 !=1,b10 !=1 and b9 !=1.



Set of outputs of the controler after verifications.



Set of outputs of the controler after verifications.



**Problems :**

- A lot of slides (but just to explain better)
- It doesn't receive the flag of ALU
- Didn't received the QMC to use for simplify the project.
- Too much MUX (this is a problem ? )

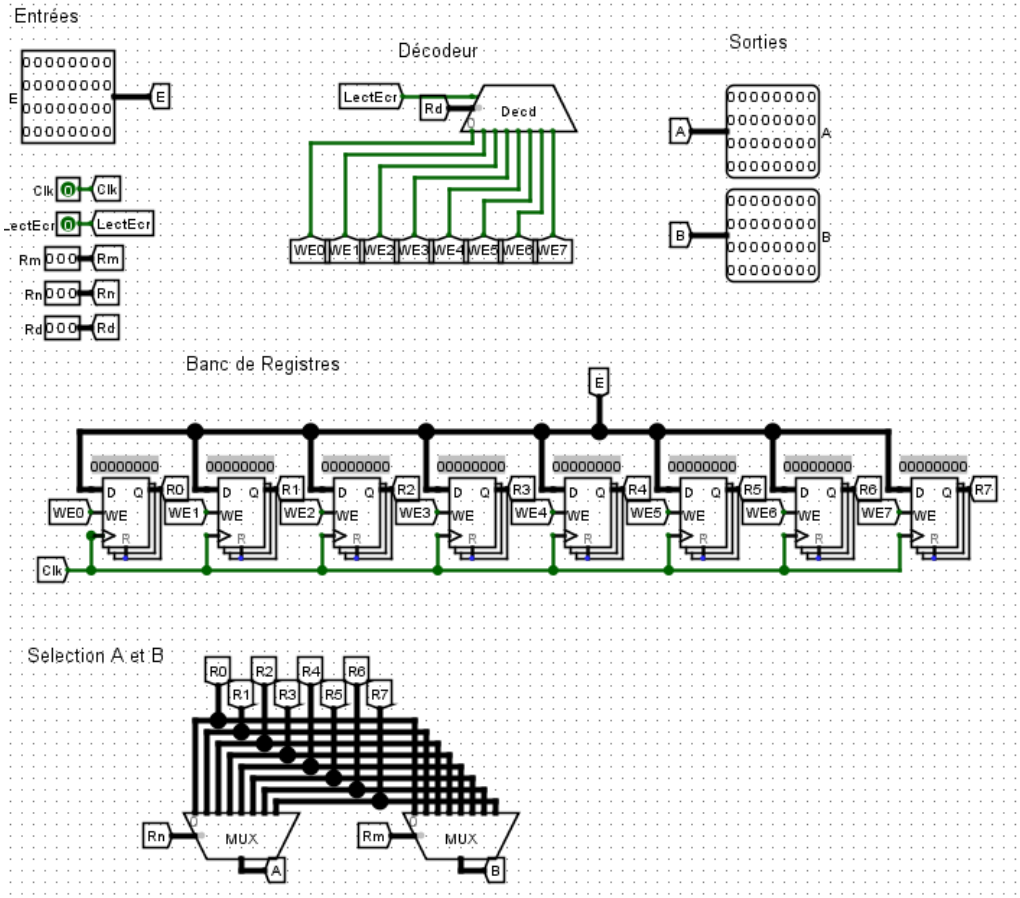



Figure: Banc de registre

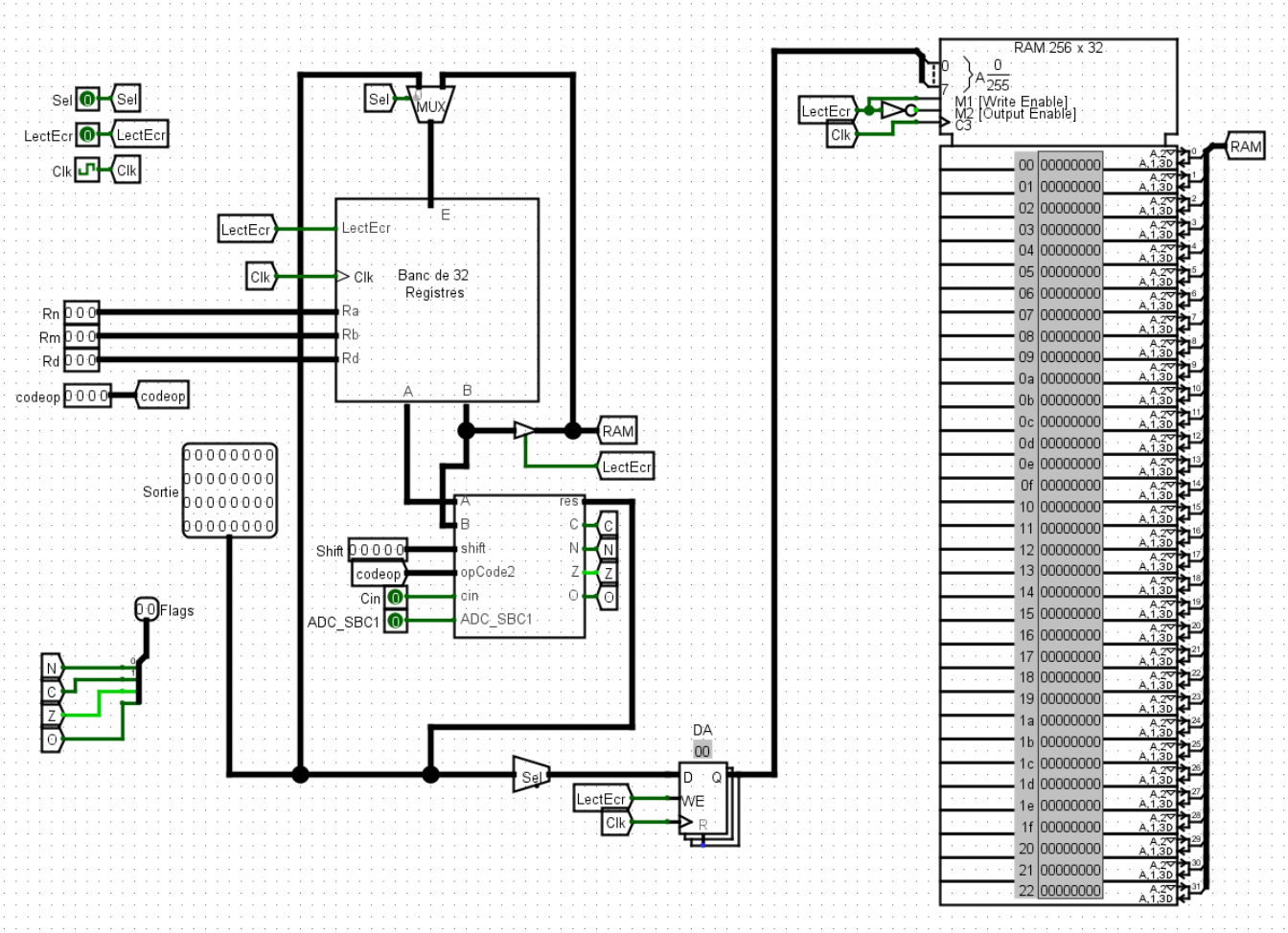



Figure: Chemin de données

Pourquoi avoir choisi Java ?

- JAVA
  - Maitrisé par toute l'équipe
  - Utilisé en ce moment même sur d'autres projets
  - Développement relativement long
- Python
  - Langage non maitrisé par l'ensemble du groupe
  - Script similaire publié par l'Université de Berkley.
  - Rapidité d'implémentation

JAVA semblait une bonne approche pour concevoir le parser. Finalement la réalisation en Python s'avère plus simple. Et seul le parser en Python est fonctionnel.

1. Lecture du fichier assembleur
2. Identification du opcode 1
3. Identification du opcode 2
4. Ecriture du fichier binaire.

```
AND  R2,R7
EOR  R1,R5
LSL  R5,R6
ADC  R4,R3
LSL  R1,R2,#23
```

Figure: Code assembleur

- ✓ Utilisation de logisim-evolution
- ✓ Utilisation de Quartus II (v13.0 sp1)
- ✓ Synthèse sur FPGA (DE2-BOARD)
- ✓ Validation de l'ALU

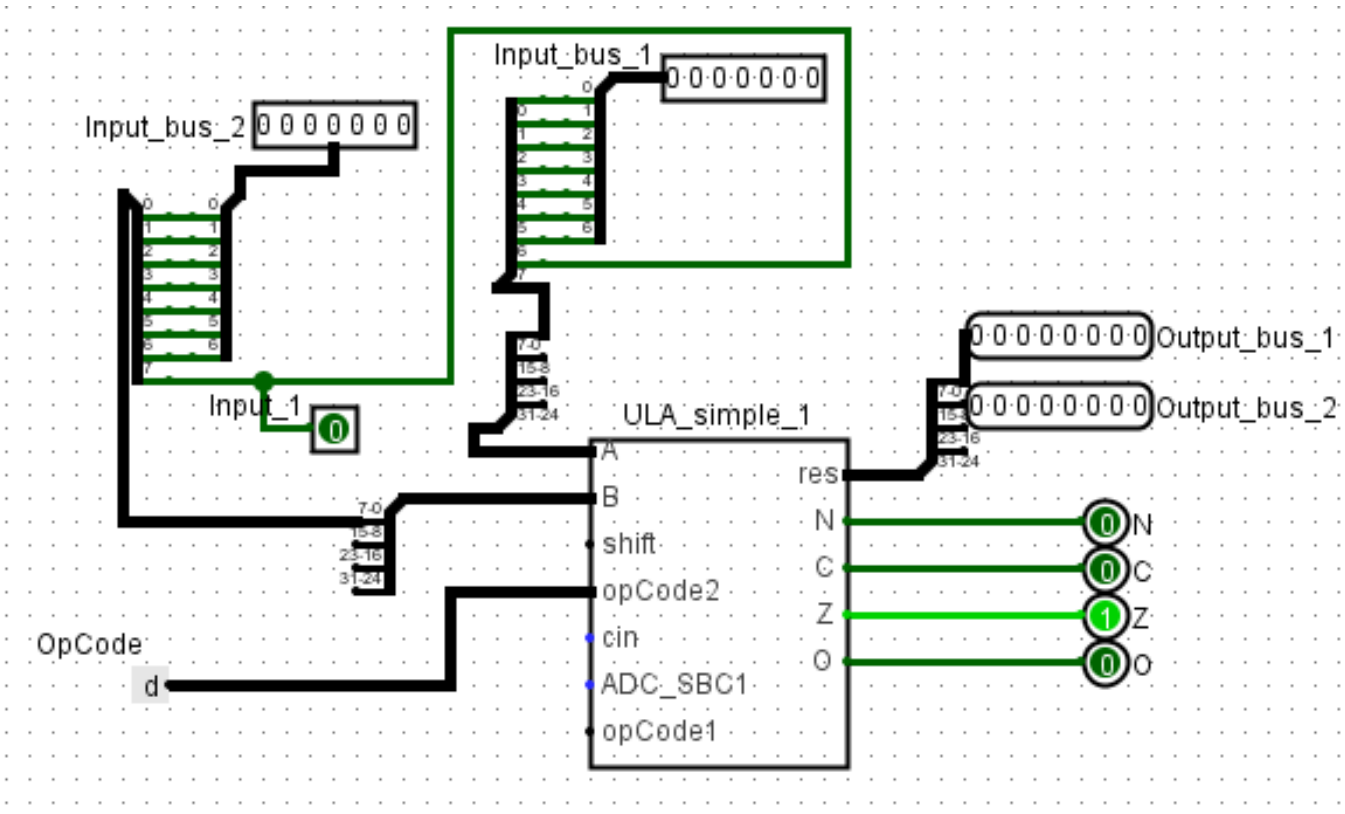



Figure: Multiplication avec l'ALU

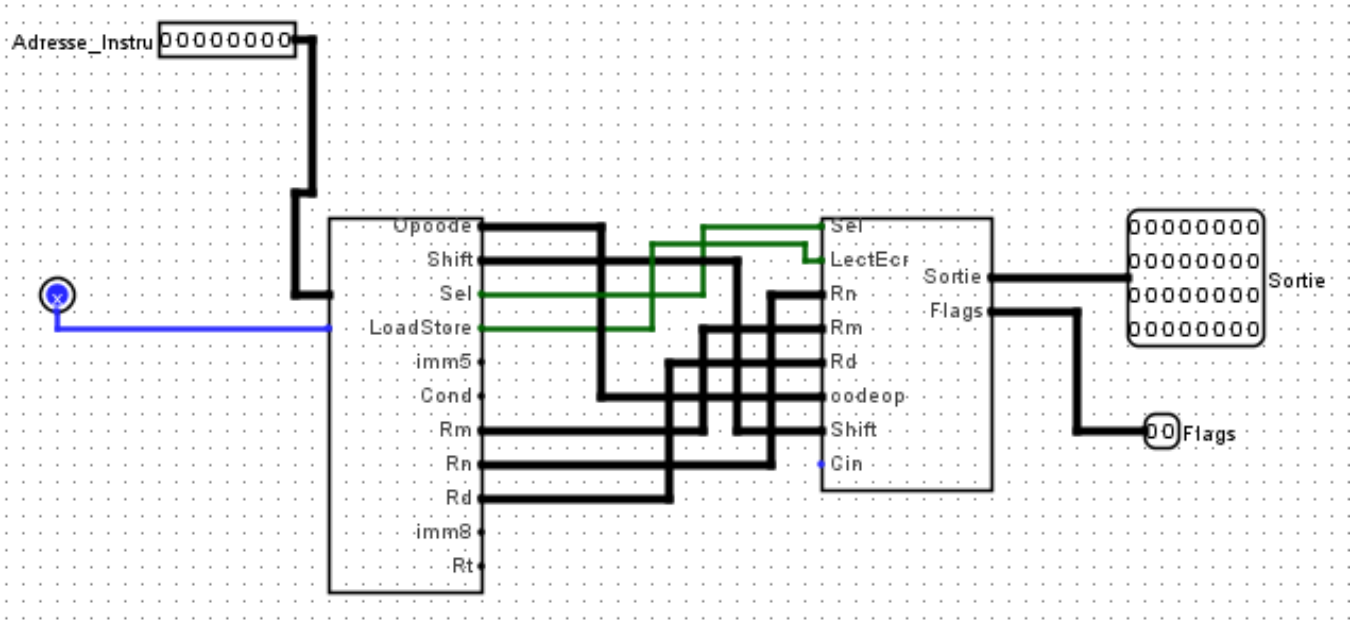



Figure: Fusion du contrôleur et du chemin de donnée

# Conclusion

**Resultats :**

- Extension de l'ULA
  - ✓ Implementation
  - ✓ Test pour le CAO
- Contrôleur-Mémoriel
  - ✓ Implementation sans le traitement dse Flags
  - ✗ Test pour le CAO
- Registres - MémoireD
  - ✓ Implementation
  - ✗ Test pour le CAO
- Assembleur/Désassembleur
  - ✓ Assembleur : Python
  - ✓ Désassembleur : Python
- Outils logiciels
  - ✓ Logisim evolution et Quartus
  - ✓ DE2 Board (xml and test)
  - ✗ Fusion partiellement réalisée
  - ✗ Simplification Quine-McCluskey

# References


- ARM®v7-M Architecture Reference Manual
  https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf