



EDF of Silicium Projet PARM

Présentation de l'équipe

EDF of Silicium

**UAL
Assembleur**

Lydia Baraukova

**Banc de registre
Controlleur
Chemin de données**

Aldric Ducreux

**Assembleur
Chemin de données**

Sylvain Masia

Controlleur

Julien Molinier

Sommaire

01

Présentation du processeur

02

Présentation des composants

03

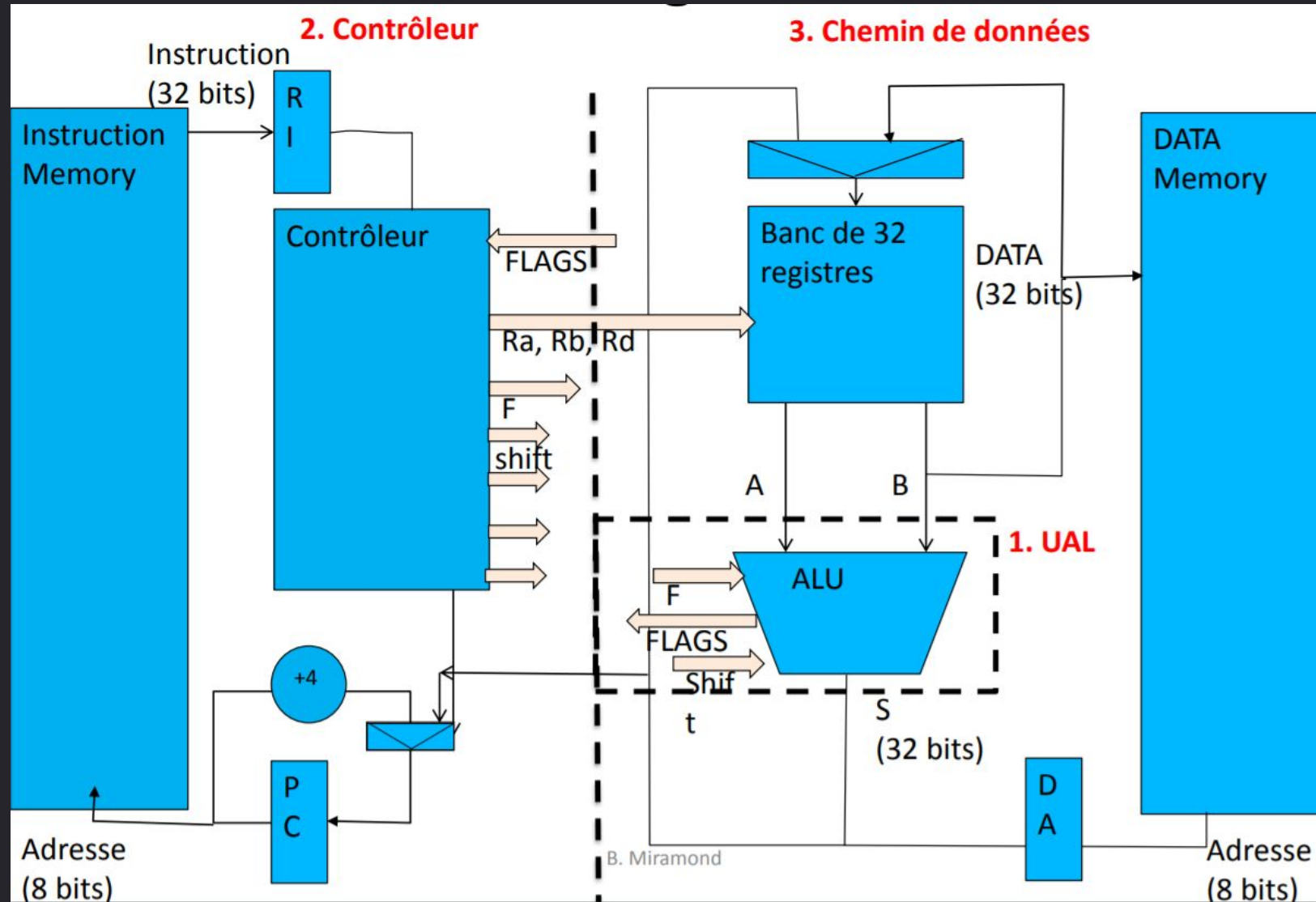
Analyse des résultats et de la conception



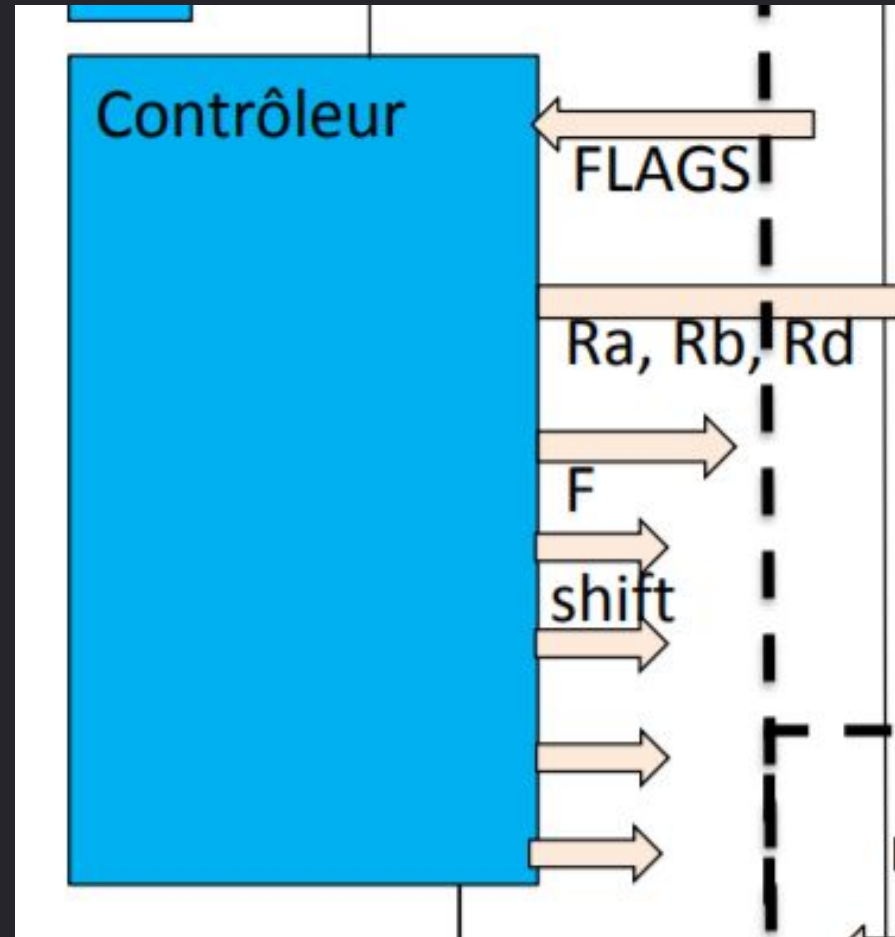
Présentation du processeur

01

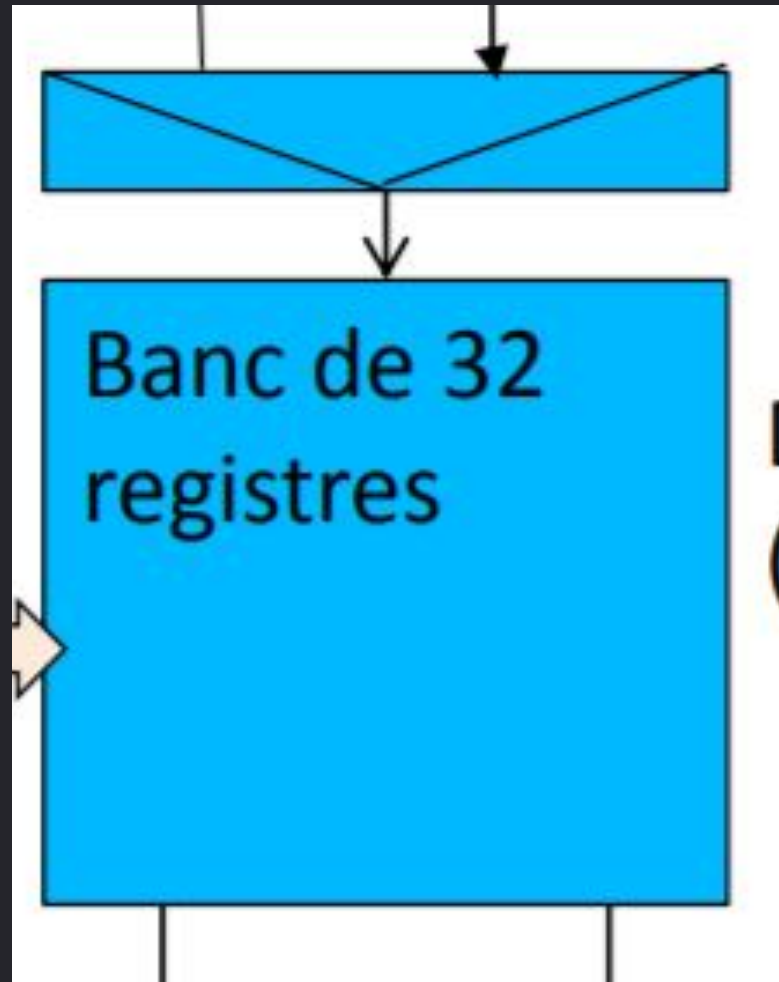
Processeur 32 bits



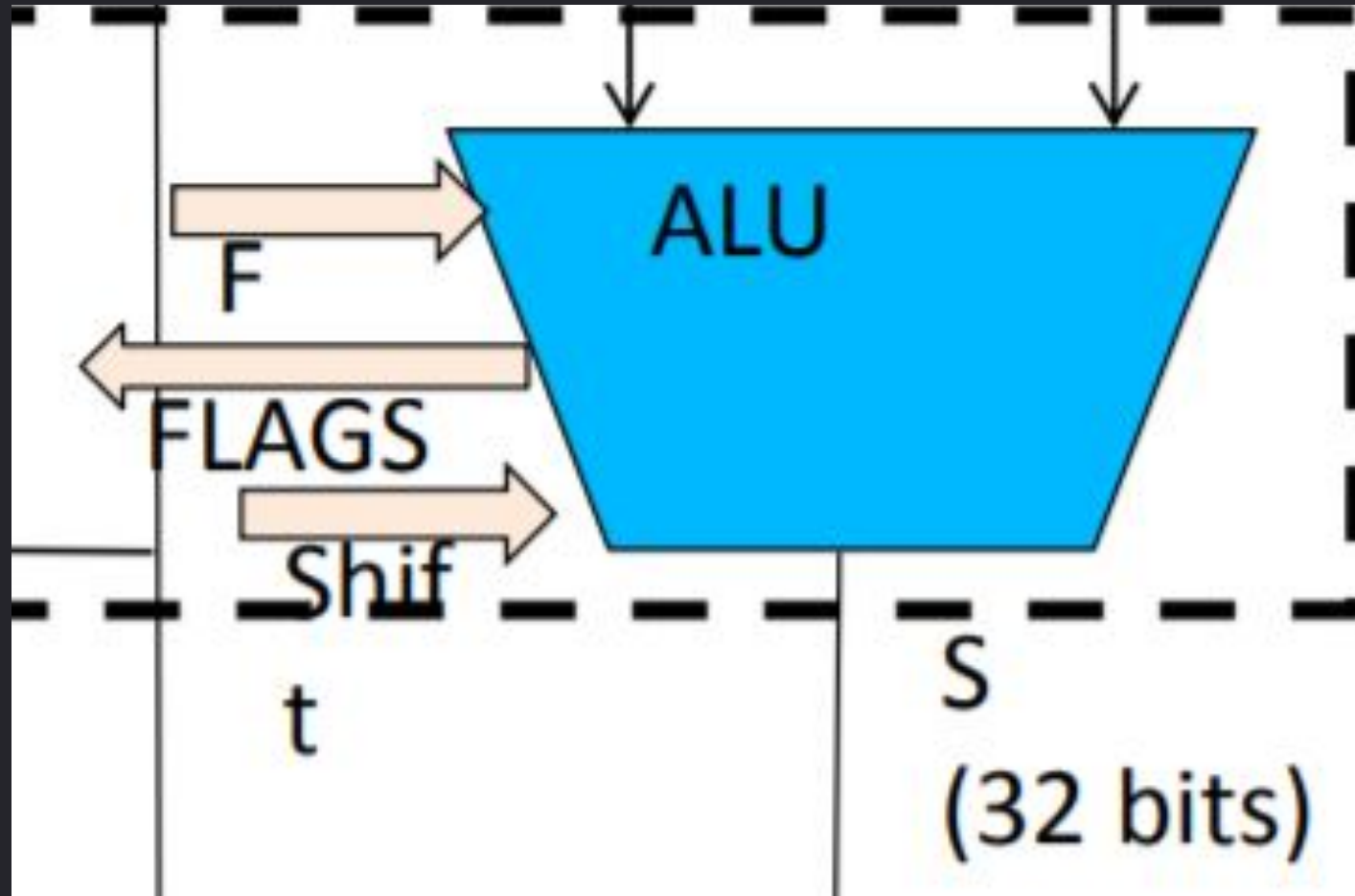
01 Processeur 32 bits



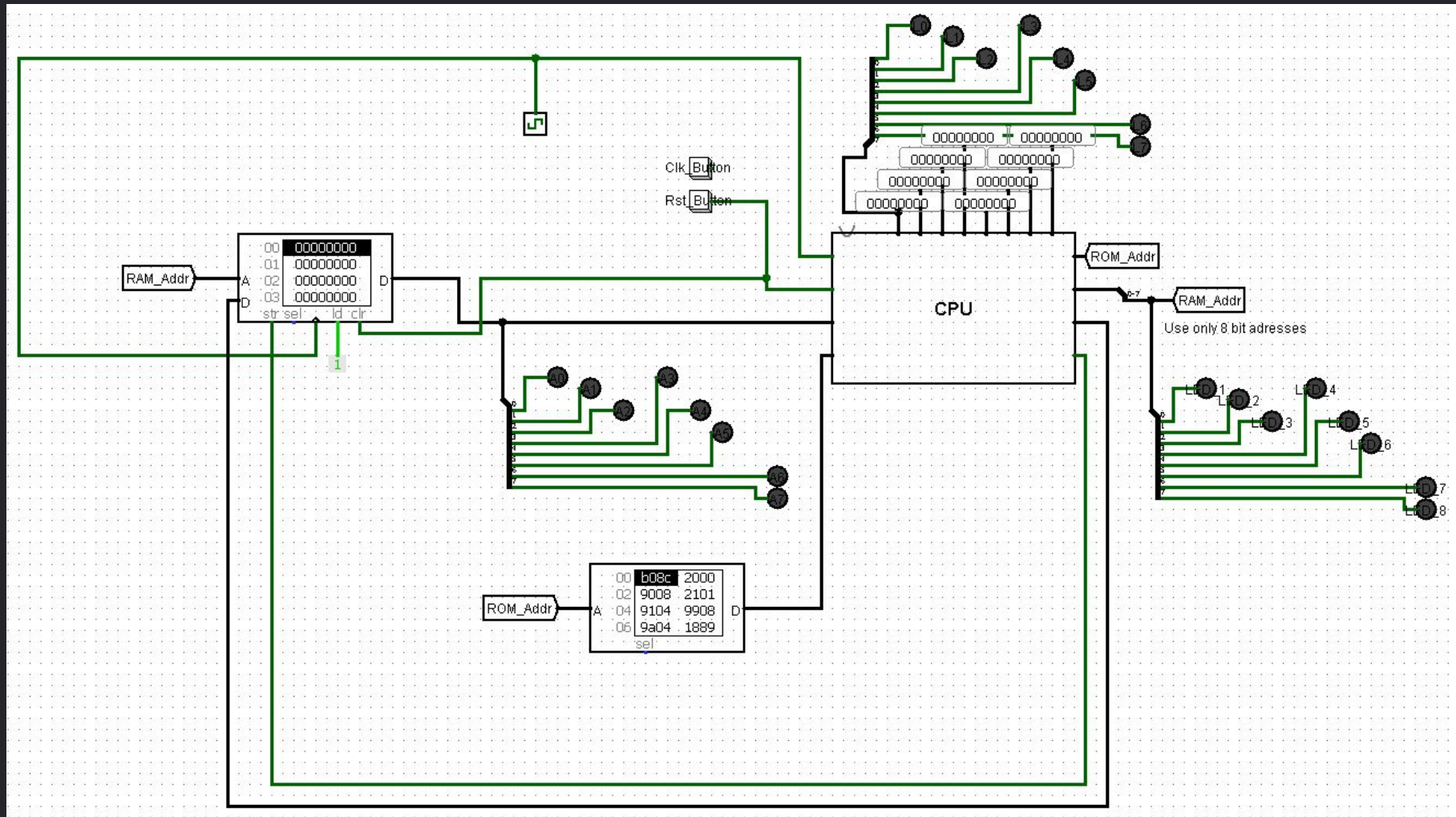
01 Processeur 32 bits




01 Processeur 32 bits



01 Processeur 32 bits

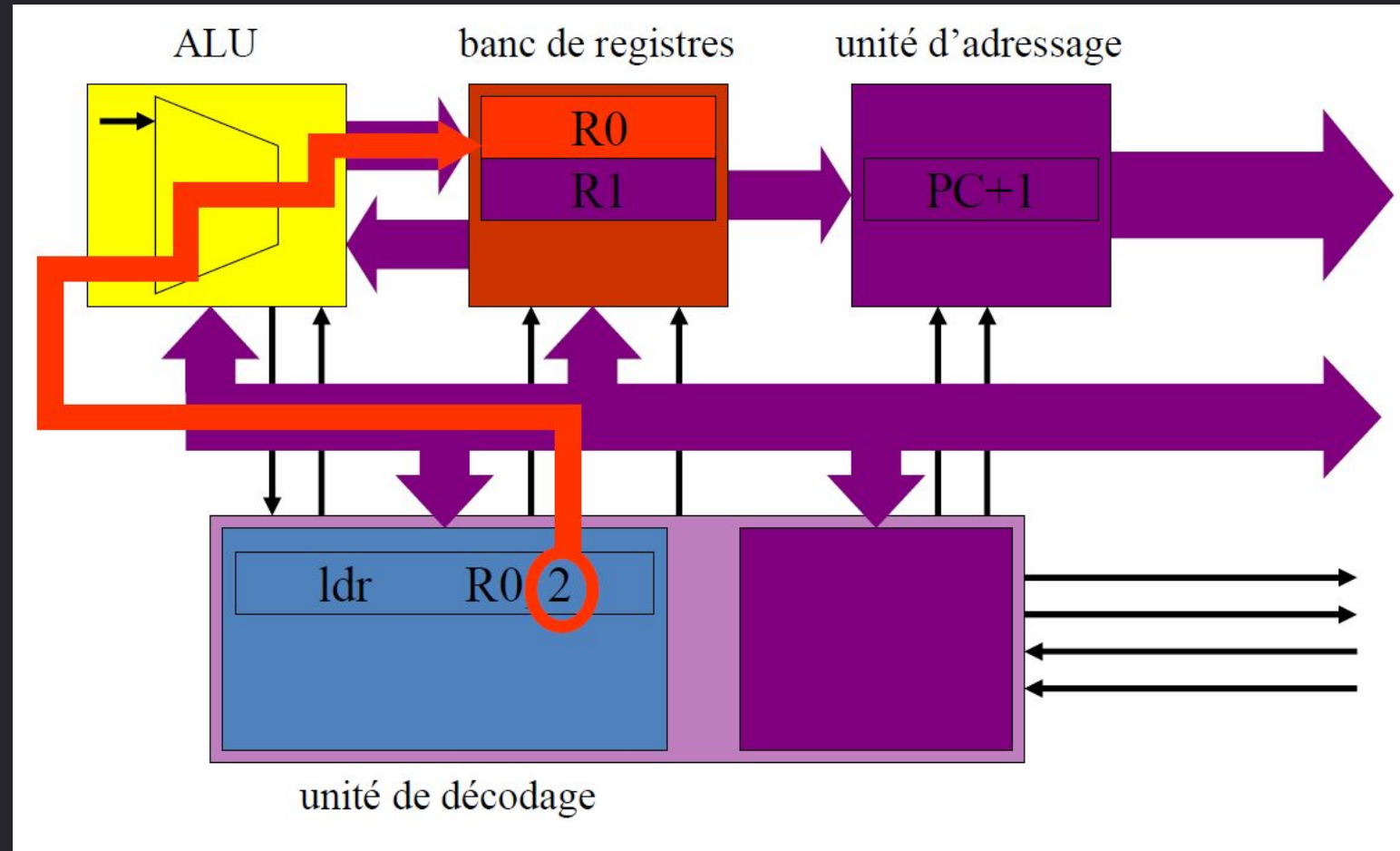


Three teal-colored abstract shapes, consisting of two overlapping circles and a smaller circle, are positioned in the upper left quadrant of the image.

Présentation des composants

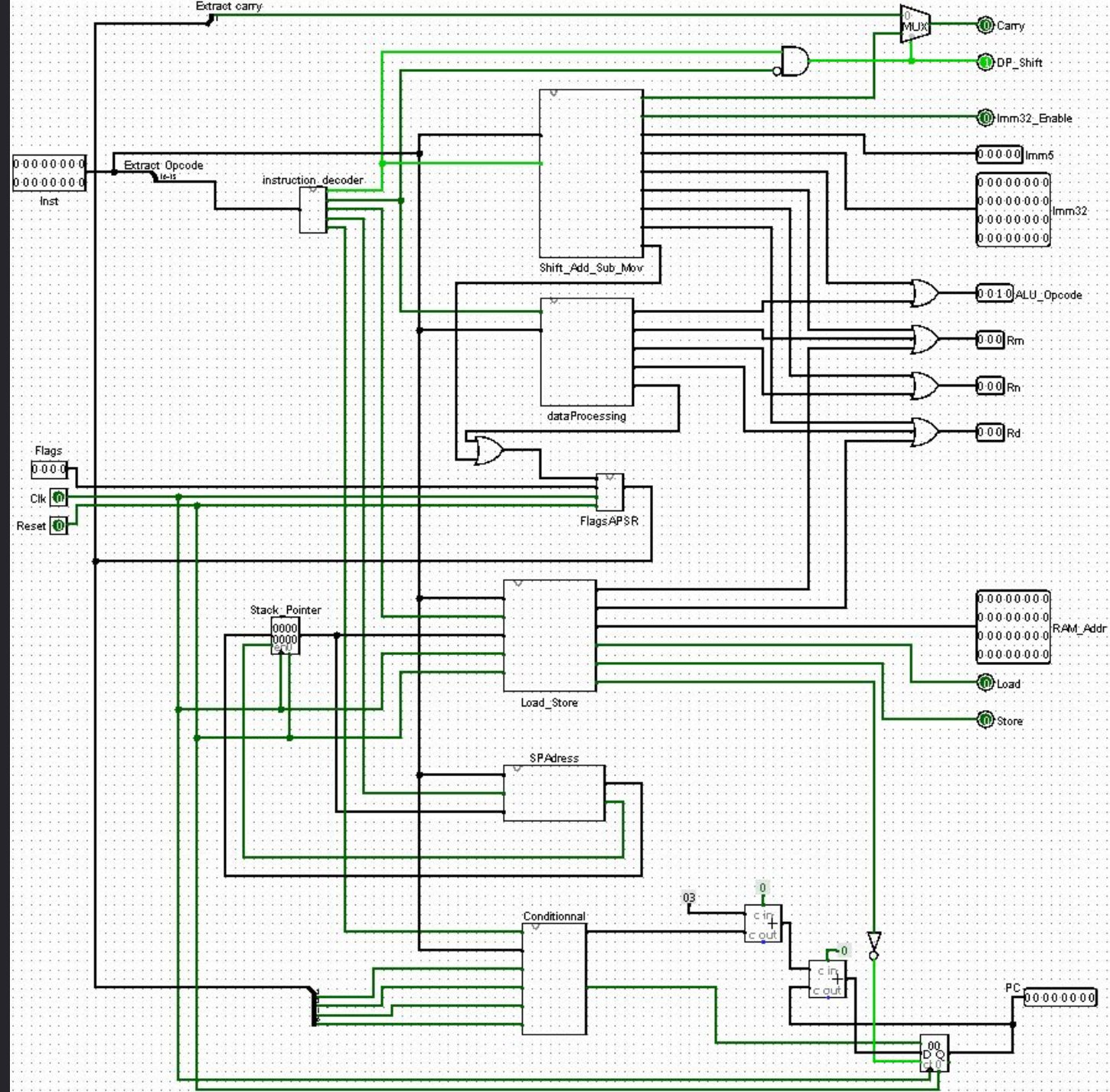
01

Contrôleur



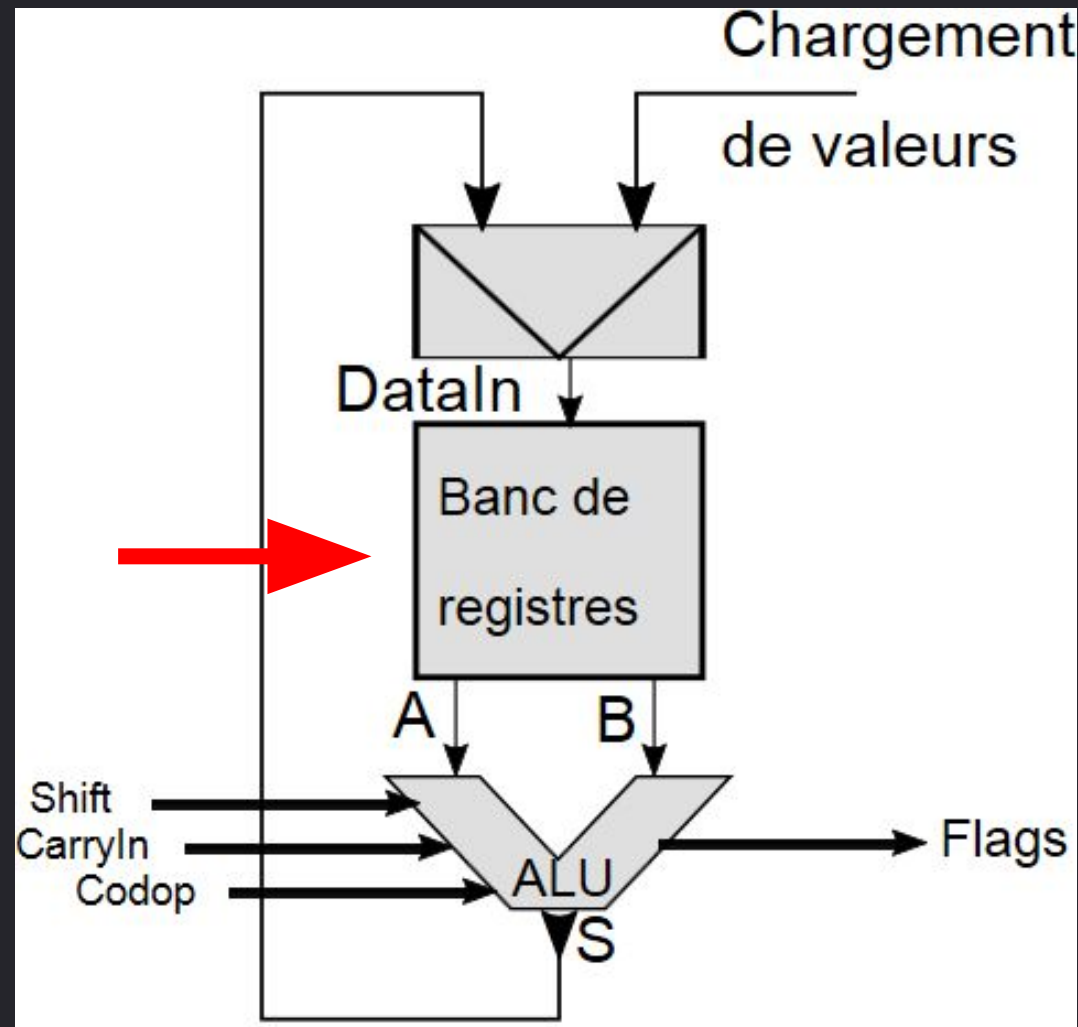
01

Contrôleur

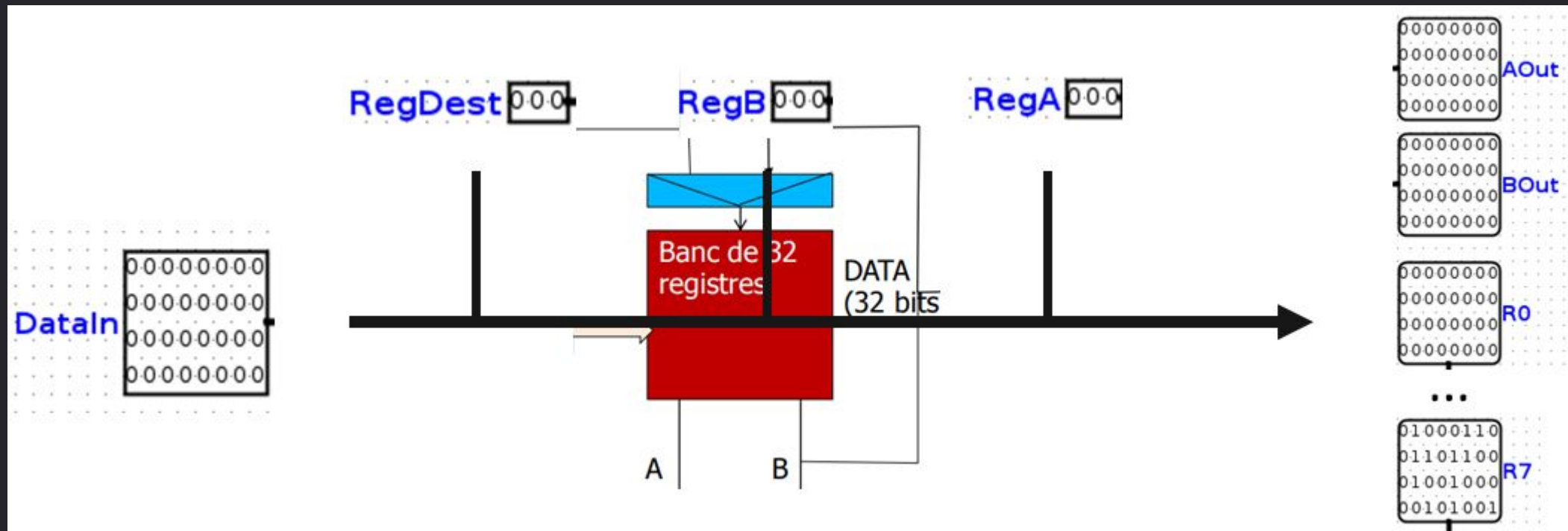


02

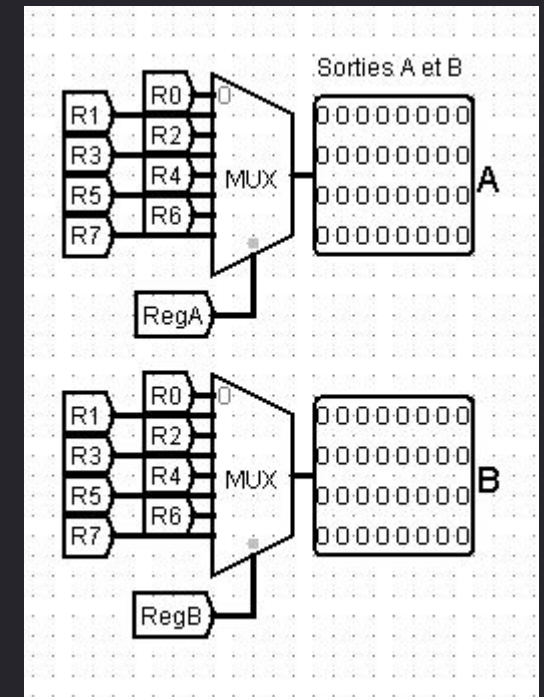
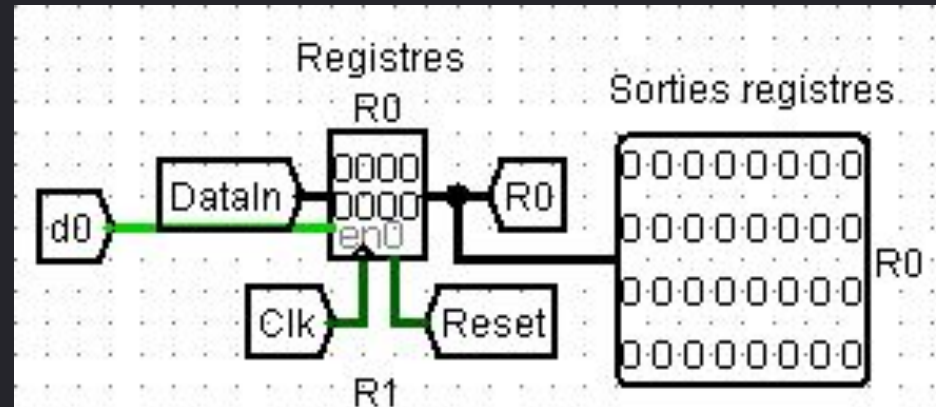
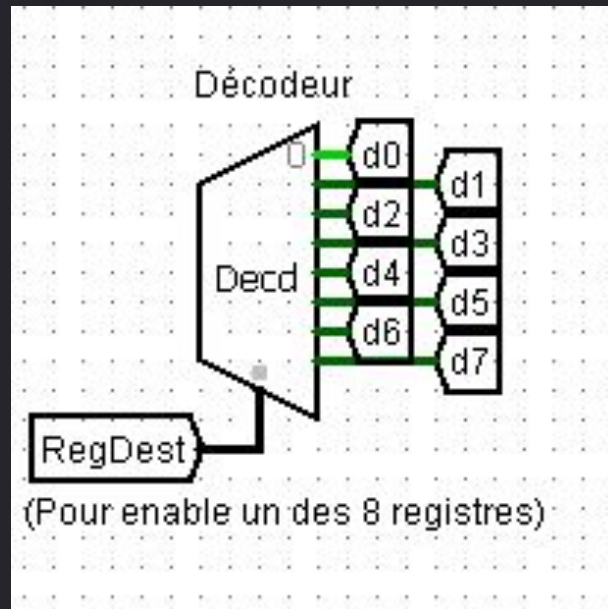
Banc de registres



02 Banc de registres



02 Banc de registres



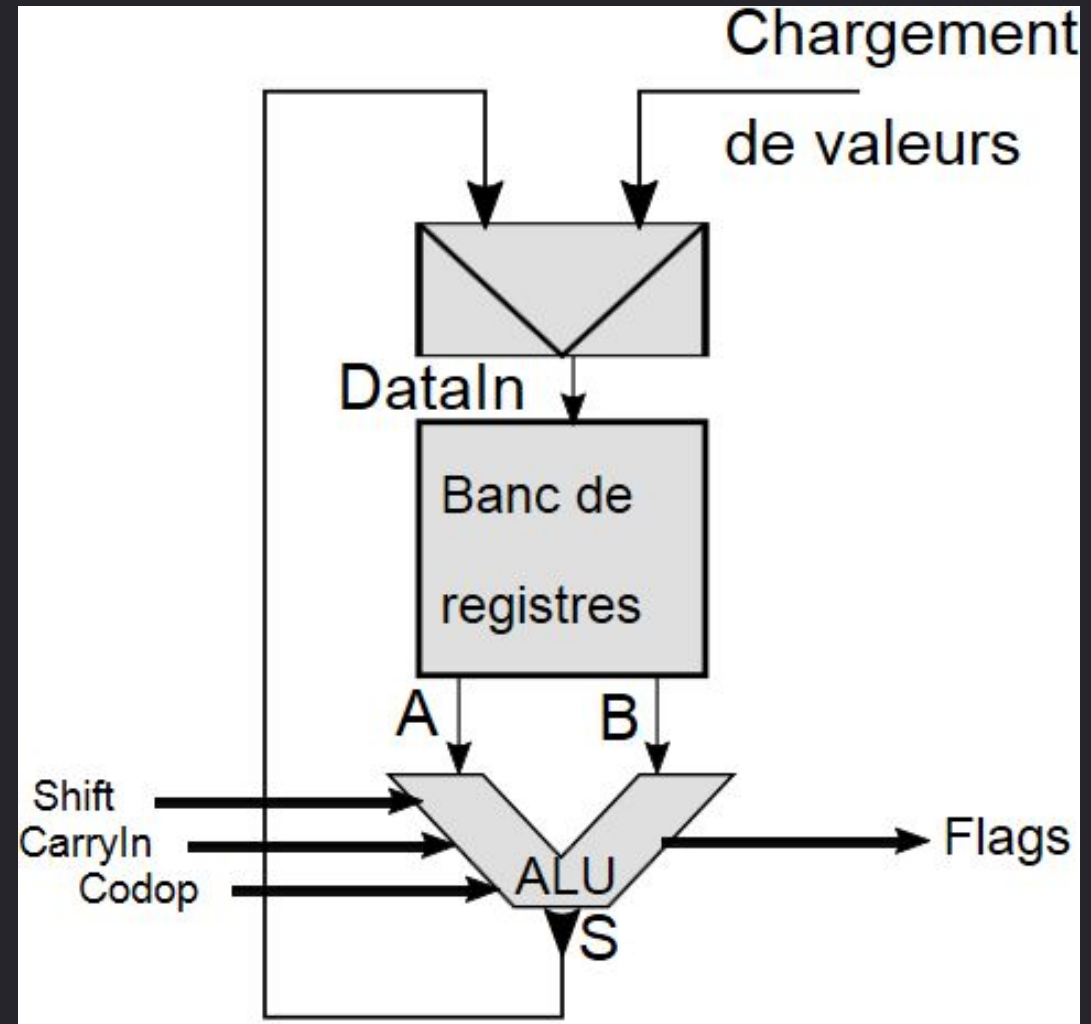
03

UAL (unité arithmétique et logique)

16 opérations

Entrée :
code de l'opération,
A, B, shift, retenue
entrante

Sortie :
S et/ou drapeaux



04

Assembleur : Structure

Codé en Java

Classe principale : Converter (lit le code assembleur dans un fichier, le traduit en hexa et l'écrit dans un autre fichier)

Converter contient des classes internes et des enum qui correspondent à chaque type d'opération (ShiftAddSubMov, DataProcessing, LoadStore, ConditionalBranch, Miscellaneous) pour faciliter la conversion en binaire

Après la conversion en binaire on convertit en hexa

05

Assembleur

Tests unitaires
avec Junit4

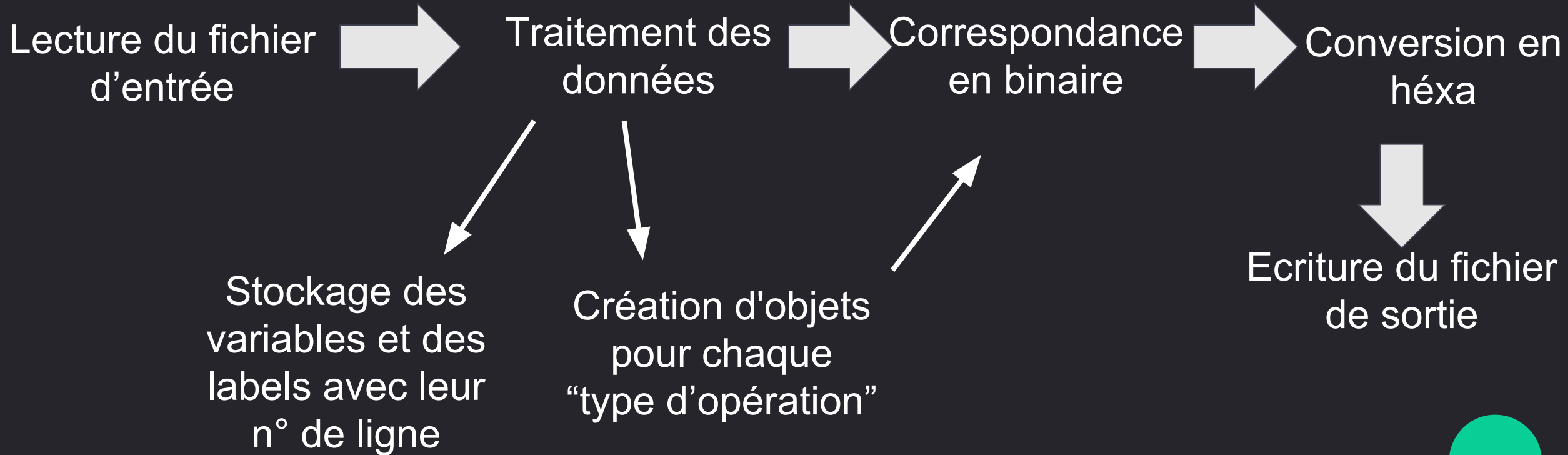
```
@Before
public void setupProgramWithBranches() {
    code = "=====
+ "; 1 programme de test : MAX = max(A, B)\n"
+ "=====
+ ".data\n"
+     "A: .word 0xf\n" // A = 15
+     "B: .word 0xff\n" // B = 255
+     "MAX: .word 0x0\n" // MAX = 0
+ ".end\n"
+ ".text\n"
+     "LDR R0,A\n" // R0 = A
+     "LDR R1,B\n" // R1 = B
+ "if:\n"
+     "CMP R0,R1\n" // comparer R0 à R1
+     "BMI else\n" // si R0 - R1 < 0, passer à else
+     "STR R0,MAX\n" // sinon, MAX = R0
+     "B endif\n" // passer à endif
+ "else:\n"
+     "STR R1,MAX\n" // si on est à else, MAX = R1
+ "endif:\n" // endif
+ ".end";
}

@Test
public void testProgramWithBranches() {
    String expected = "v2.0 raw\n"
        + "9800\n" // ldr 0 0
        + "9901\n" // ldr 1 1
        + "4288\n" // cmp 0 1
        + "d406\n" // bmi 6
        + "9002\n" // str 0 2
        + "de07\n" // b 7
        + "9102\n"; // str 1 2

    String result = "v2.0 raw\n" + new Converter().convert(code);
    assertEquals(expected, result);
}
```

04

Assembleur : Fonctionnement



A teal abstract graphic consisting of three overlapping circles of varying sizes, positioned in the upper left quadrant of the slide.

Analyse des résultats et de la conception

Analyse

Conception des modules :

Possibilité de modifier la structure des modules pour un meilleur coût énergétique et vitesse de calcul.

Résultats obtenus

L'assembleur fonctionne avec les tests unitaires.

Mais problème pour le fonctionnement général.

Processeur :

- Possibilité d'amélioration des composants
- Meilleure conception pour rentabilité
- Fonctionnement du projet incertain



Conclusion



Merci

Avez vous des questions ?

