POLYTECHNIQUE
MONTRÉAL

INF1600

Architecture des micro-ordinateurs

Travail Pratique 1 : Périphériques et architecture

Soumis par :

Chowdhury, Rasel – 2143023

Fellah, Rayan -2147523

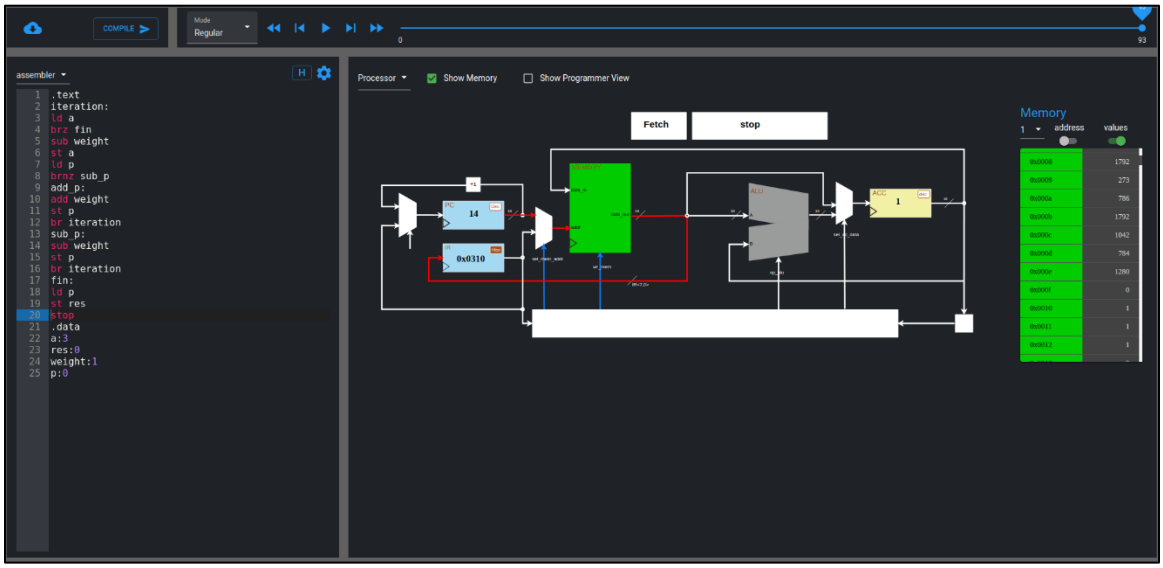
Section 4; Groupe 4

Partie 2.1 :

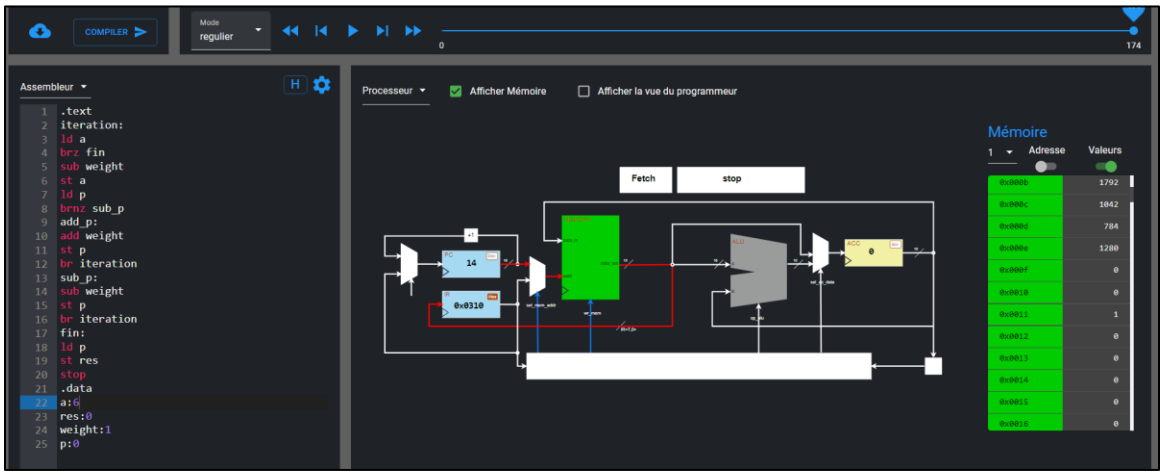
0. La valeur de << a >> dans notre rapport est de 3.
   Formule : = 3 + abs ((Matricule_1 + Matricule_2) % 13)
   = 3 + abs ((2143023 + 2147523) % 13)
   = 3+0
   = 3

1. Les structures de contrôle utilisées au sein de ce programme sont << la structure de contrôle conditionnelle >> (dû au commande << brnz >> et << brz >>) et << la structure de contrôle itérative >> (dû au commande << brnz >> et << brz >>) . En effet, les comportements de deux commandes, soit le << if >> et le << while >> sont répétées à travers le programme en entier, expliquant pourquoi les deux structures de contrôle qui y apparaissent sont << la structure de contrôle conditionnelle >> et << la structure de contrôle itérative >>.

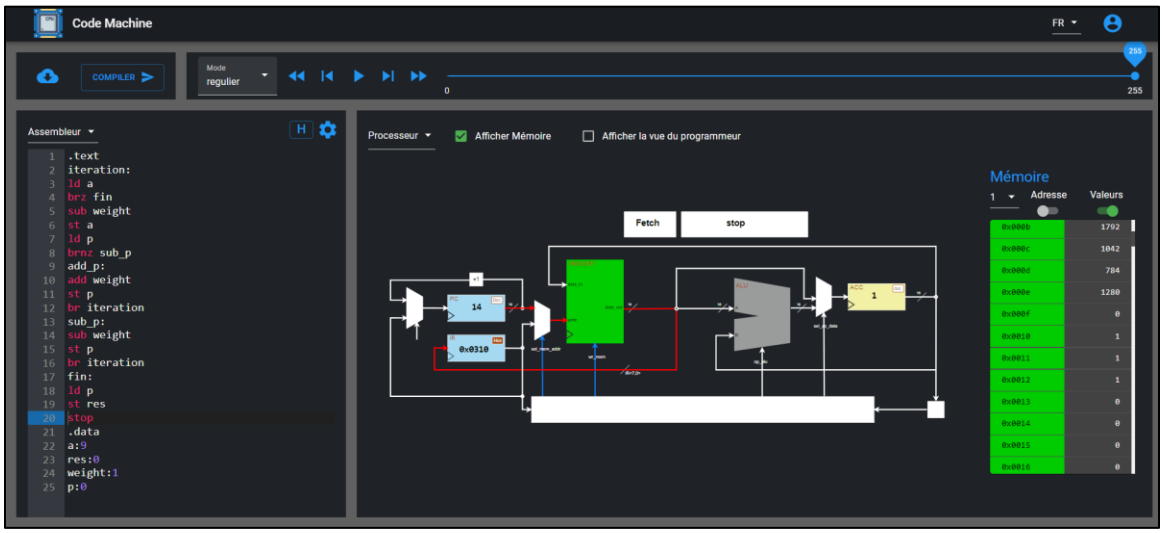
2. Le contenu en mémoire à l'adresse 0x0010 à la fin de notre programme est de 1. Celle-ci correspond à la valeur de << res >> que l'on stock dans le mémoire.



3. Lorsqu'on rentre un nombre pair, la valeur de << res >> est égal à 0 à la fin du programme. Dans le cas contraire, lorsqu'on rentre un nombre impair, la valeur de << res >> est égal à 1.



<< a >> (6) est pair. << res >> à l'adresse 0x0010 vaut 0.

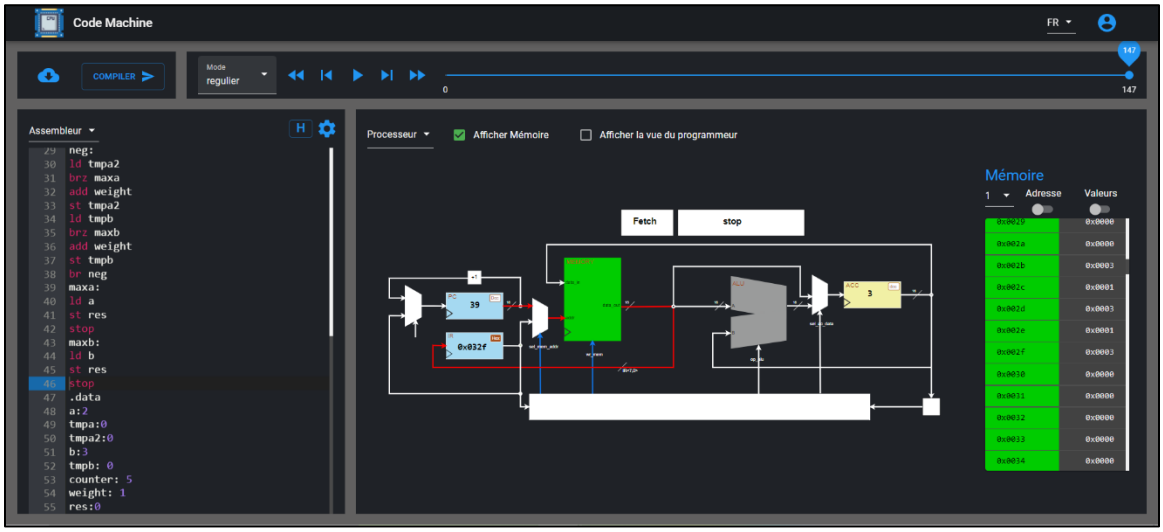


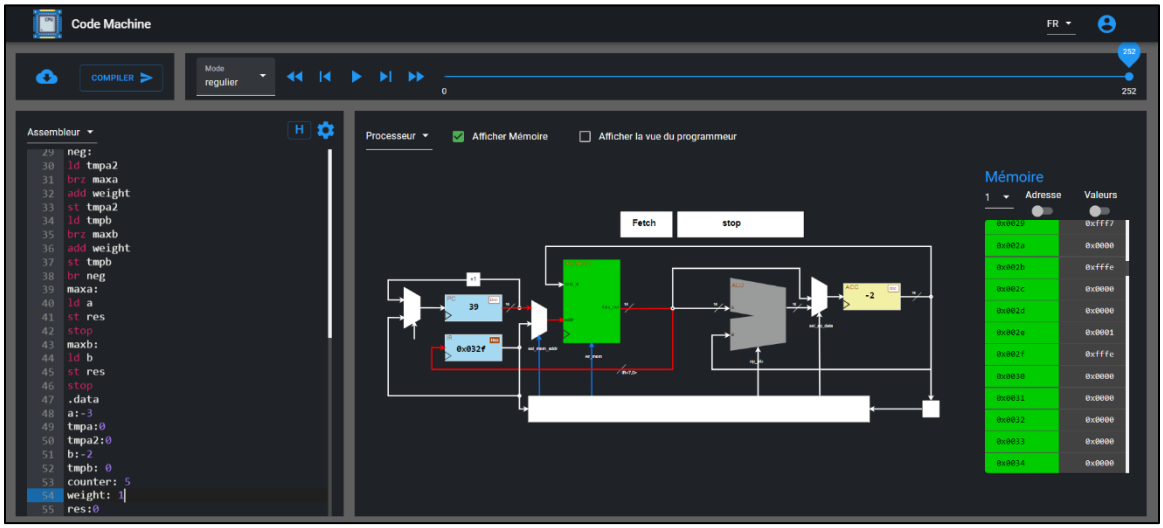
<< a >> (9) est impair. << res >> à l'adresse 0x0010 vaut 1.

4. Si le processeur est sur 32 bits et qu'il n'utilise pas le bit le plus significatif, alors le résultat maximal du programme serait de 2^32-1.

Partie 2.2 :

Pour pouvoir déterminer la valeur maximale entre deux nombres positifs, nous avons itérer à travers une boucle en soustrayant 1 unité à chacune des valeur à chaque itération. Le nombre qui arriverait à 0 en premier serait alors la valeur minimale, et on sauvegarde alors l'autre valeur dans le <<< res >>. Dans le cas où nous aurions deux valeurs négatives, nous additionnons alors 1 unité à chacune des valeurs, et c'est la première arrivée à 0 qui serait la valeur maximale. Pour déterminer s'il faut aller dans l'étiquette << pos >> (dans laquelle on compare deux valeurs positives) ou << neg >> (dans laquelle on compare deux valeurs négatives), il a fallu déterminer un compteur que nous avons initialiser à 99 (valeur maximale possible) et le décrémenter d'une unité à chaque itération, à la même fréquence que nous décrémentions une des deux valeurs entrées. Si le compteur arrivait à 0 en premier, les valeurs sont donc négatives; sinon, les valeurs sont positives.





Voici notre code au complet :

```
.text
ld a
st tmpa
ld a
st tmpa2
ld b
st tmpb
br posouneg
posouneg:
ld tmpa
brz pos
sub weight
st tmpa
ld counter
brz neg
sub weight
st counter
br posouneg
pos:
ld tmpa2
brz maxb
sub weight
st tmpa2
ld tmpb
brz maxa
sub weight
st tmpb
br pos
neg:
```

```
ld tmpa2
brz maxa
add weight
st tmpa2
ld tmpb
brz maxb
add weight
st tmpb
br neg
maxa:
ld a
st res
stop
maxb:
ld b
st res
stop

.data
a:2
tmpa:0
tmpa2:0
b:3
tmpb: 0
counter: 5
weight: 1
res:0
```

Partie 3.1:

0. Notre valeur de data dans le rapport est de 42.

   Formule = 3 + | (2+1+4+7+5+2+3) – (2+1+4+3+0+2+3) |

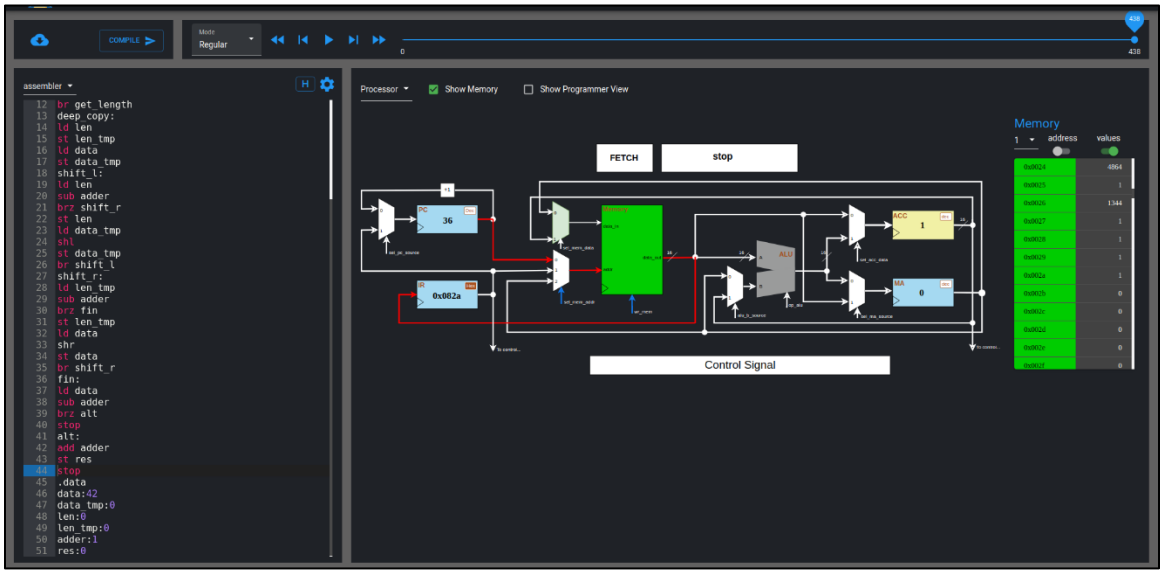
   = 3 + | (24) + (15) |

   = 3 + (39)

   = 42

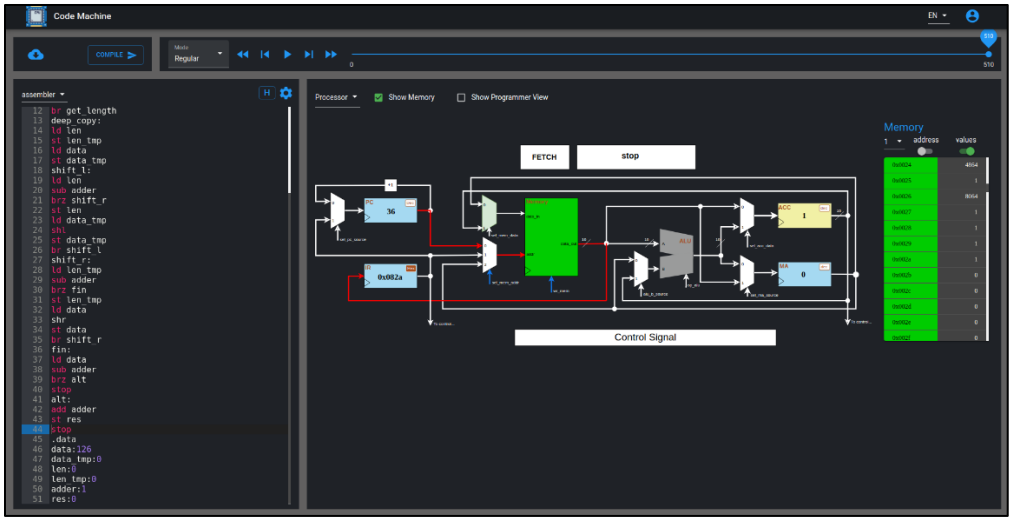
1. Les structure de contrôle au sein de ce programme sont la structure de contrôle itérative (elle agit comme un << while et un do while >>) et la structure de contrôle conditionnelle (elle agit comme un << if … else >>).

2. Le contenu en mémoire à l'adresse 0x002A à la fin de l'exécution du programme est :

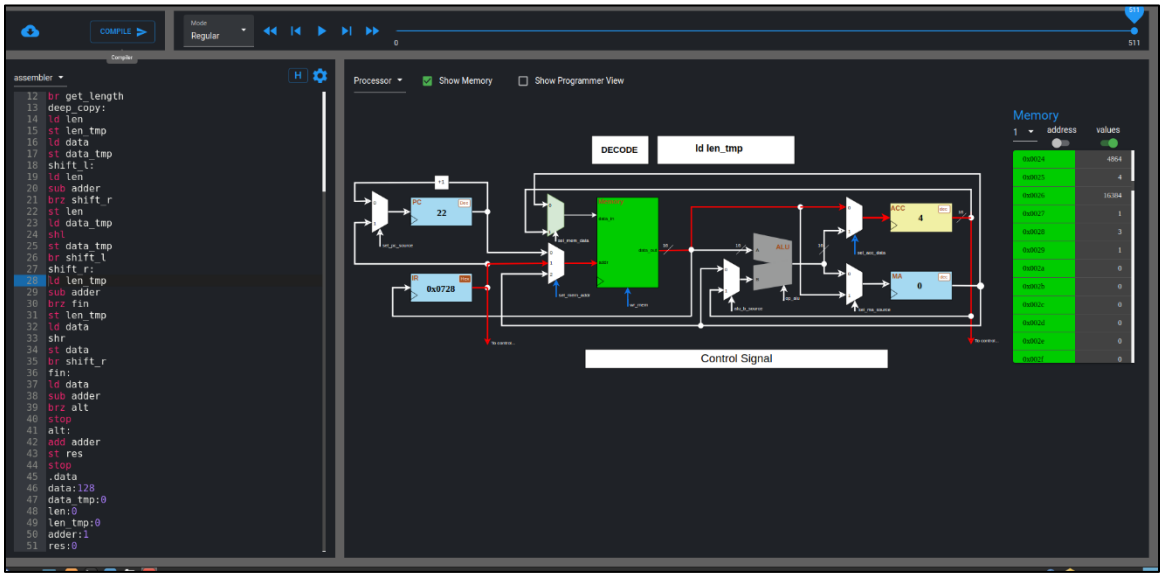


3. En gros, lorsque la valeur de la donnée saisit est supérieur a 2^7-1, << res >> retourne un 0. Dans le cas contraire, lorsque la donnée saisit est inférieur à 2^7, << res >> retourne 1. Cela est démontré dans la photo du haut.

4. Le code C se te trouvera dans un document à part intitulé << **CODE3_TP1_1600** >>.



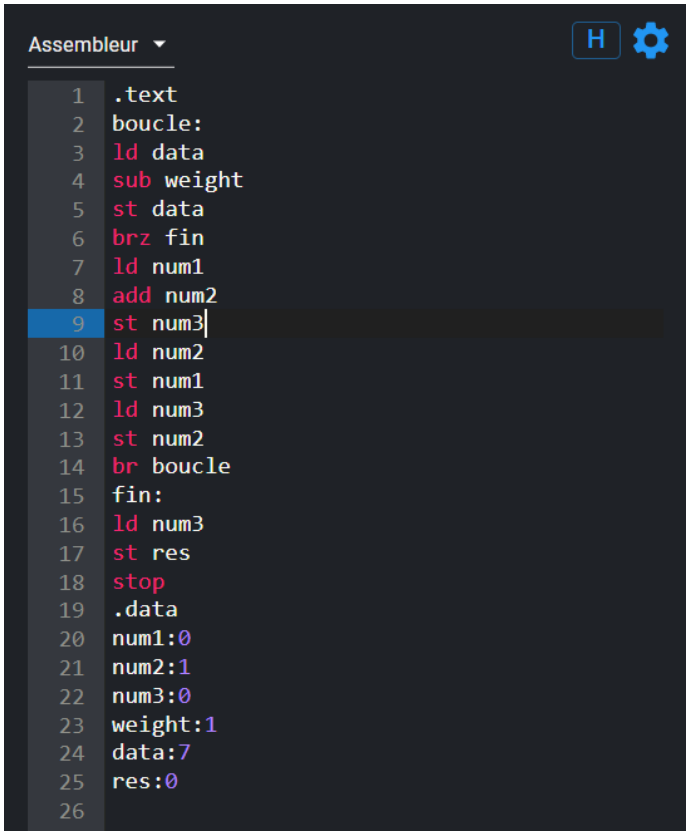
Calcul avec data=126 < 127, «res» à l'adresse 0x002A donne 1 à la fin du programme.



Calcul avec data=128 > 127, «res» à l'adresse 0x002A donne 0 à la fin du programme.

Partie 3.2

Voici une photo de notre code :

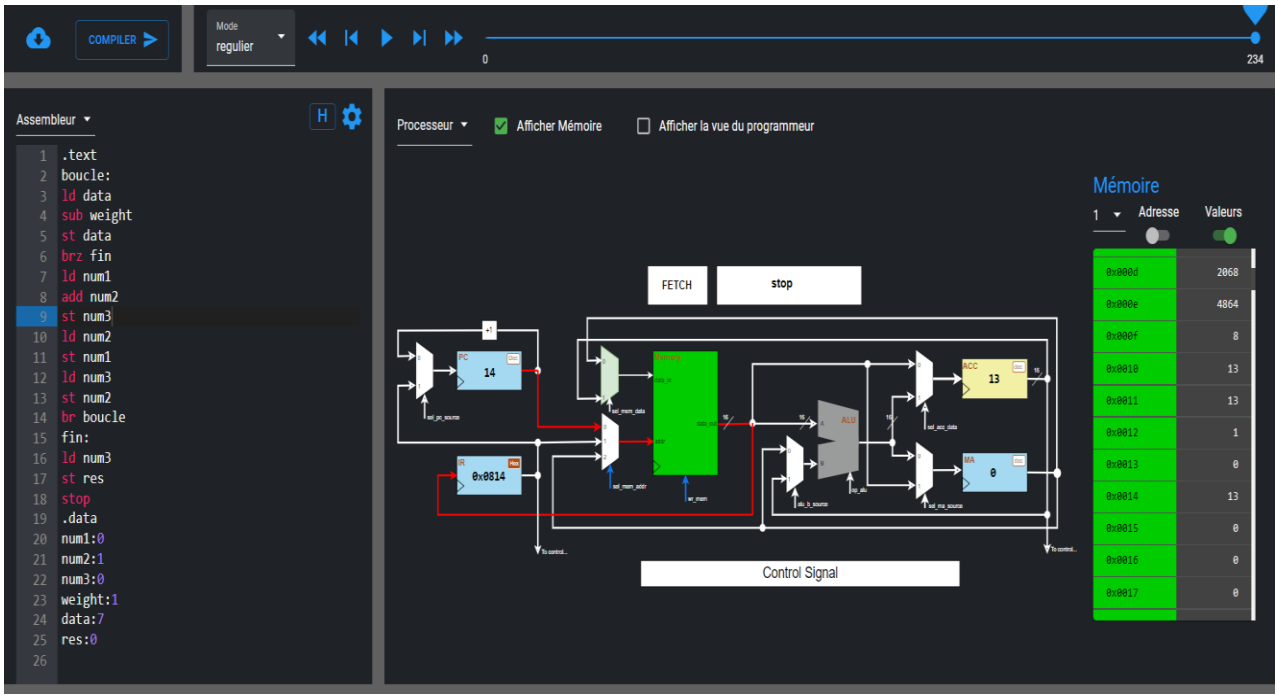


Le fonctionnement de ce programme est le suivant. Les deux premières valeurs de la suite de Fibonacci (0 et 1) sont assignées respectivement à num1 et num2. Num3 correspond à l'addition de num1 et num2. À chaque itération, la valeur précédente de num3 devient la valeur de num2, tandis que la valeur de num2 devient celle de num1. La variable data correspond à la position de la valeur recherchée dans la suite de Fibonacci (en comptant à partir de 0). Elle est décrémentée de 1 à chaque itération. Une fois qu'elle est de 0, num3 est stocké dans la variable res et le programme prend fin.

La valeur de << data >> dans notre rapport est de 7.

Formule : = 7 + abs ((Matricule_1 + Matricule_2) % 13)

= 7 + abs ((2143023 + 2147523) % 13)
= 7+0
= 7



La 7ème valeur de la suite de Fibonacci (en partant de 0) est 13. Cette valeur est stockée dans la variable «res», à l'adresse 0x0014.

Si on souhaite obtenir la nième valeur de la suite de Fibonacci, il suffit de modifier la valeur de data.