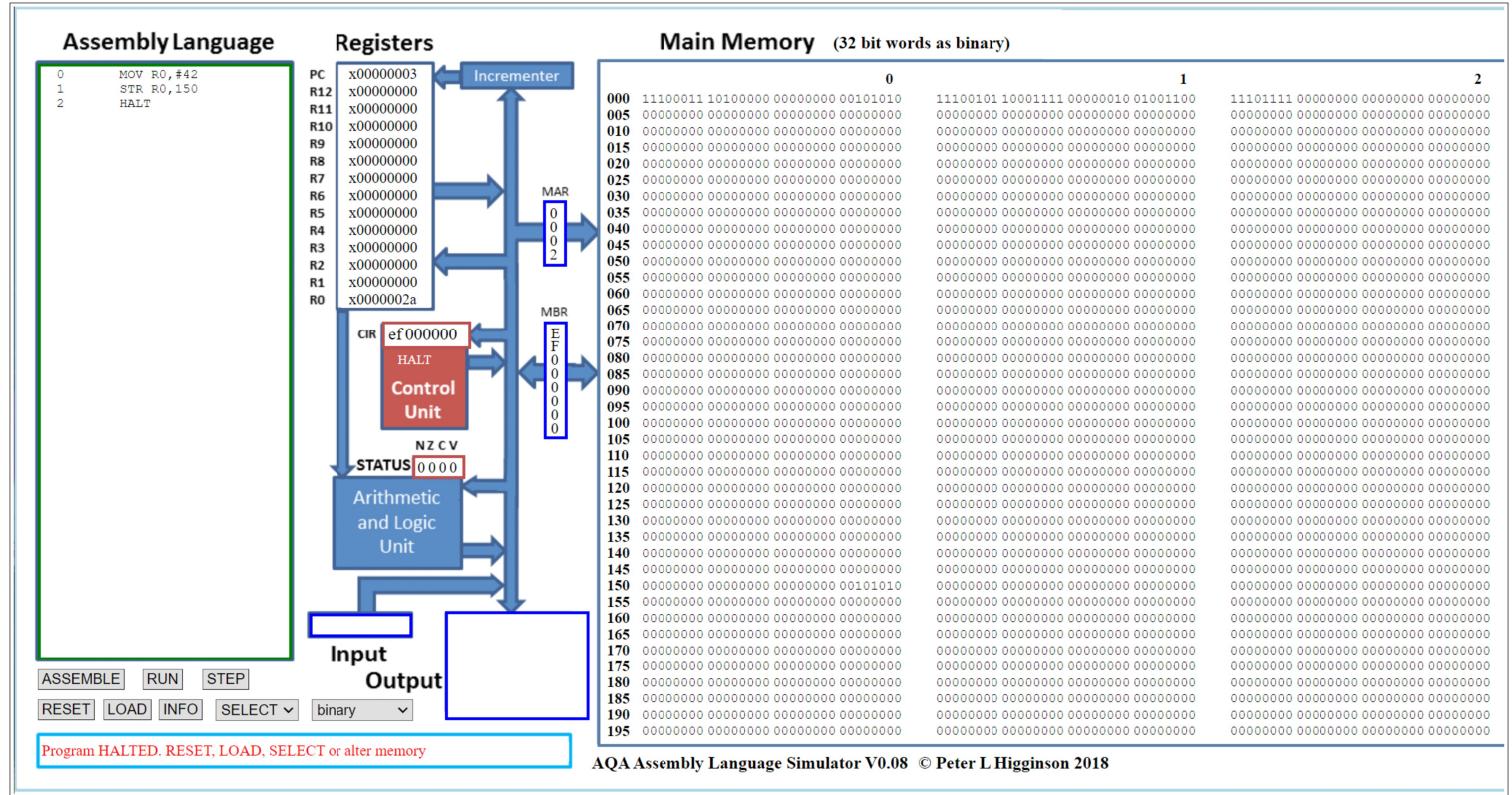


TD - Modèle de Von Neumann - Assembleur ARM

Affichage en binaire des instructions machines :



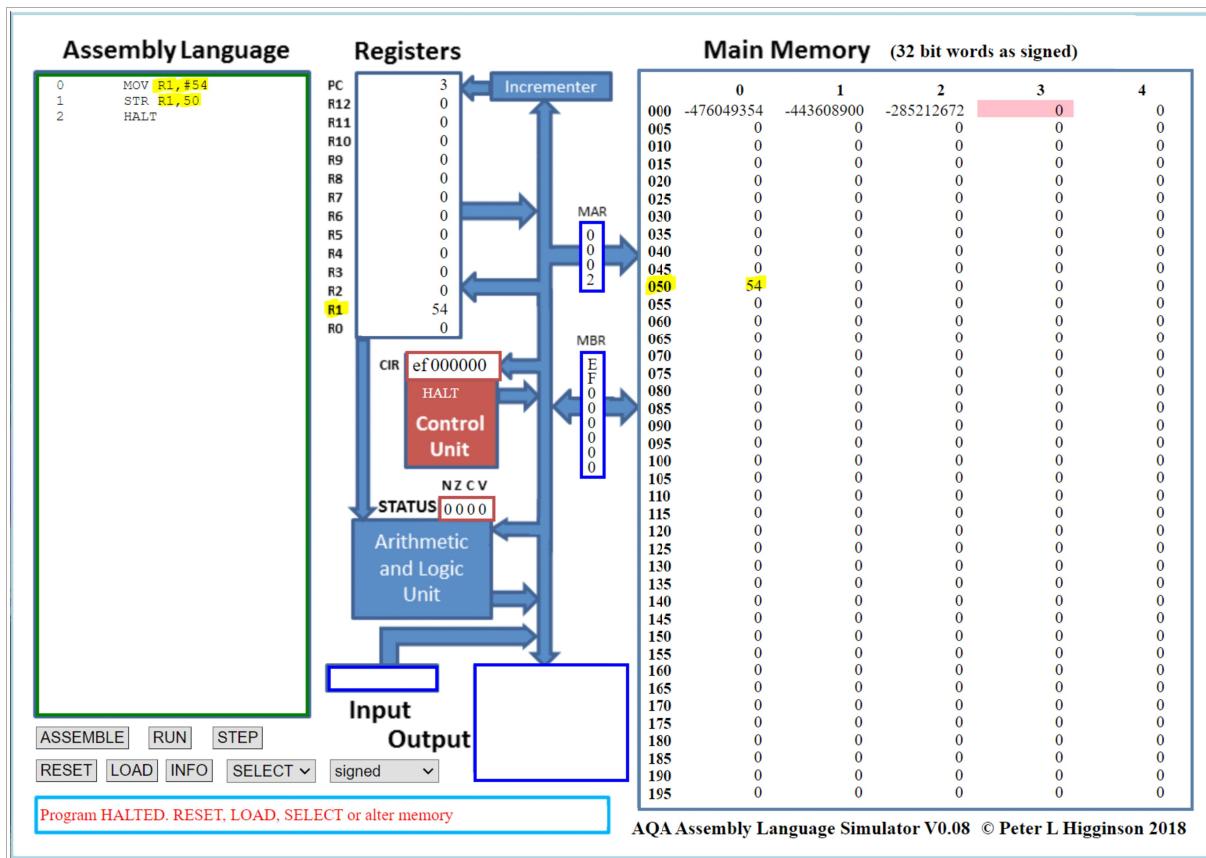
Comme on peut le voir ci-dessus les instructions machines : "MOV R0,#42" ; "STR R0,150" ; "HALT" ont leurs emplacements mémoires respectifs en binaire (000, 001, 002).

On remarque aussi que l'instruction « store » a bel et bien stocké la valeur du registre 0 (soit #42) dans l'emplacement mémoire 150.

Nous pouvons donc maintenant affirmer que :

- l'instruction machine "11100011 10100000 00000000 00101010" correspond au code assembleur "MOV R0,#42"
- l'instruction machine "11100101 10001111 00000010 01001100" correspond au code assembleur "STR R0,150"
- l'instruction machine "11101111 00000000 00000000 00000000" correspond au code assembleur "HALT"

Modification du programme précédent afin d'obtenir la valeur 54 à l'adresse mémoire 50 :



Assembly Language :

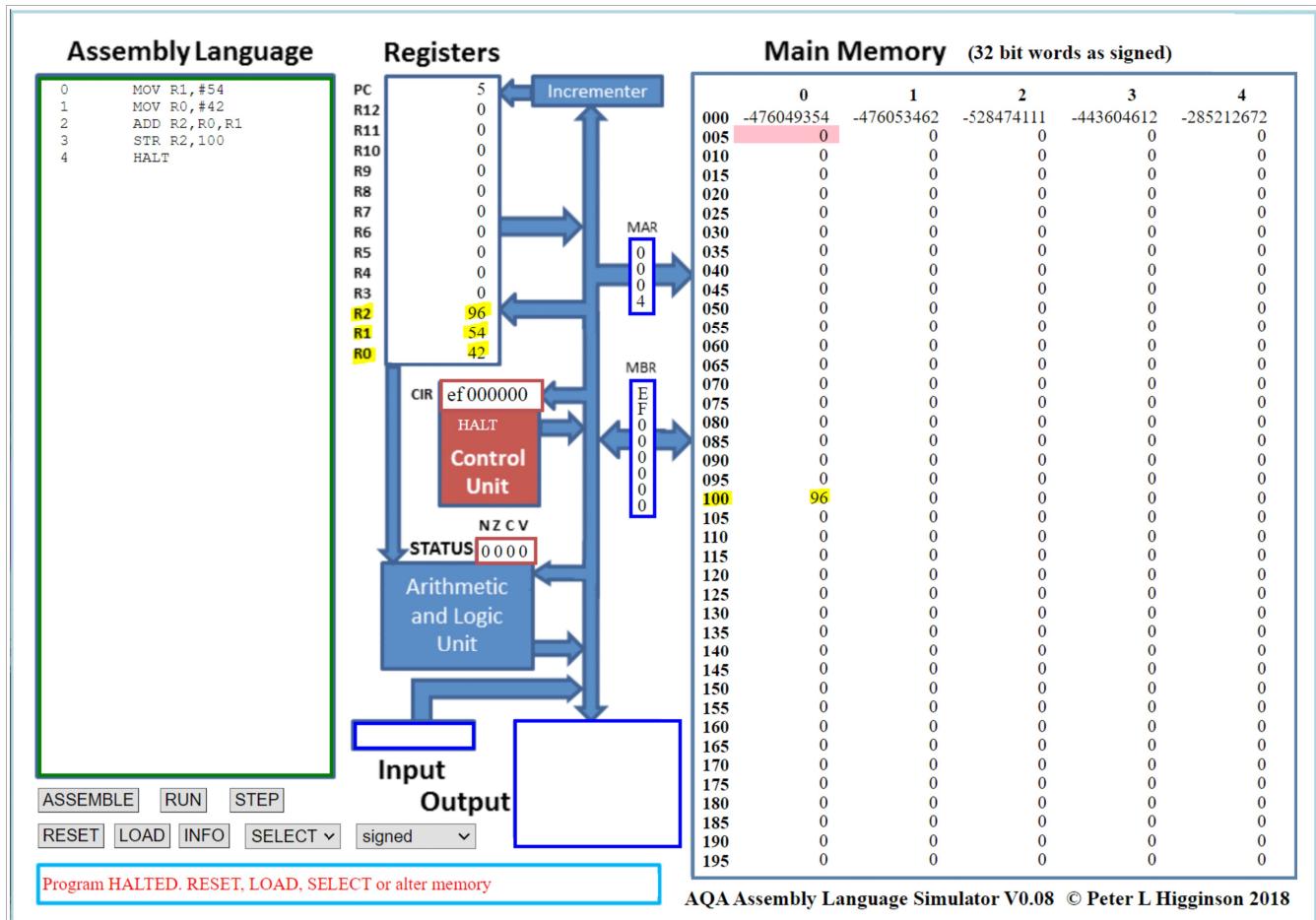
```
MOV R1,#50
STR R1,50
HALT
```

Interprétation du code :

MOV R1,#50	Place le nombre 50 dans le registre R1
STR R1,50	Enregistre la valeur du registre R1 dans la mémoire qui a pour adresse « 50 »
HALT	Arrête l'exécution du programme

Grâce à ce code j'obtiens bien à la fin de l'exécution, la valeur 54 à l'adresse mémoire 050 comme le démontre la démonstration graphique juste au-dessus.

Programme calculant la somme de 42 et 54 afin de stocker le résultat à l'adresse mémoire 100 :



Assembly Language :

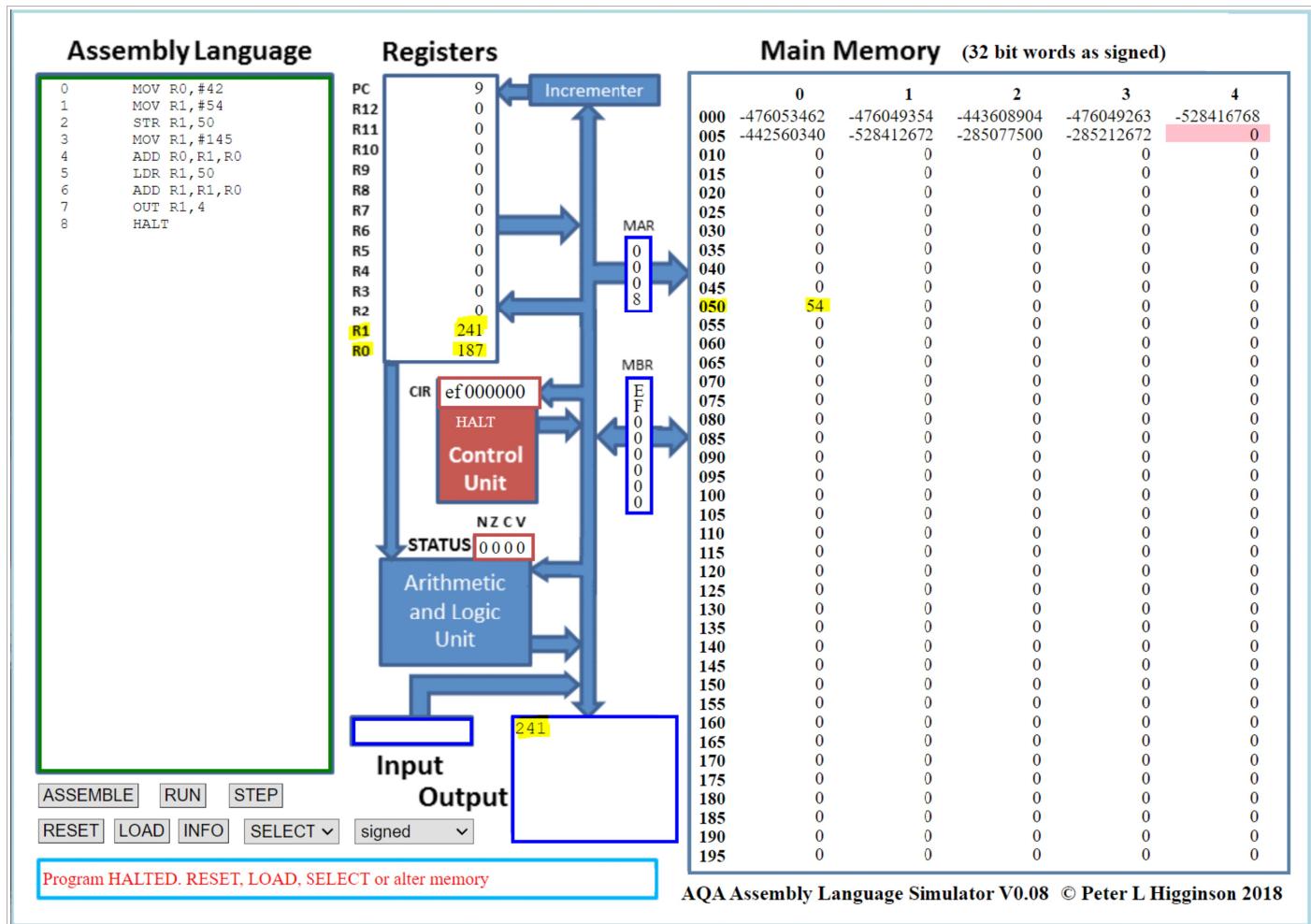
```

MOV R1,#54
MOV R0,#42
ADD R2,R0,R1
STR R2,100
HALT
    
```

Interprétation du code :

MOV R1,#54	Place le nombre 54 dans le registre R1
MOV R0,#42	Place le nombre 42 dans le registre R0
ADD R2,R0,R1	Additionne la valeur du registre R0 avec celle du registre R1 puis place le résultat dans le registre R2.
STR R2,100	Enregistre la valeur du registre R2 dans la mémoire qui a pour adresse « 100 »
HALT	Arrête l'exécution du programme

Programme additionnant trois nombres afin de stocker le résultat à l'adresse mémoire 100 (en utilisant seulement 2 registres) :



Assembly Language :

```

MOV R0,#42
MOV R1,#54
STR R1,50
MOV R1,#145
ADD R0,R1,R0
LDR R1,50
ADD R1,R1,R0
OUT R1,4
HALT

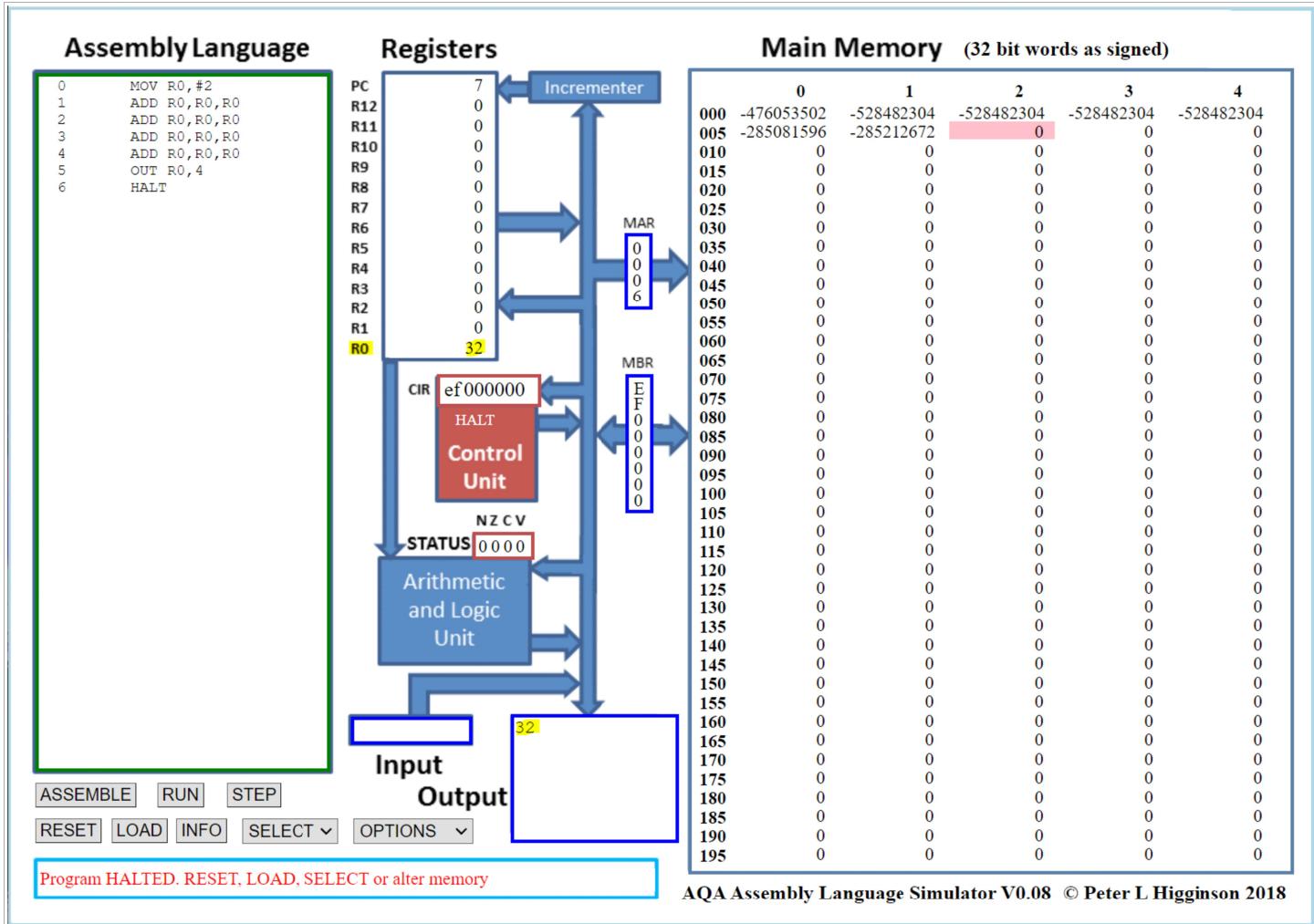
```

Interprétation du code :

MOV R0, #42	Place le nombre 42 dans le registre R0
MOV R1, #54	Place le nombre 54 dans le registre R1
STR R1, 50	Enregistre la valeur du registre R1 dans la mémoire qui a pour adresse « 50 »
MOV R1, #145	Place le nombre 145 dans le registre R1
ADD R0, R1, R0	Additionne la valeur du registre R0 avec celle du registre R1 puis place le résultat dans le registre R0.
LDR R1, 50	Charge la valeur enregistrée dans l'adresse mémoire « 50 » dans le registre R1.
ADD R1, R1, R0	Additionne la valeur du registre R0 avec celle du registre R1 puis place le résultat dans le registre R1.
OUT R1, 4	Retourne en sortie le nombre du registre R1.
HALT	Arrête l'exécution du programme

* On peut remarquer que je n'utilise seulement que deux registres.

Programme calculant 2^5 en utilisant qu'un seul registre et sans boucle :



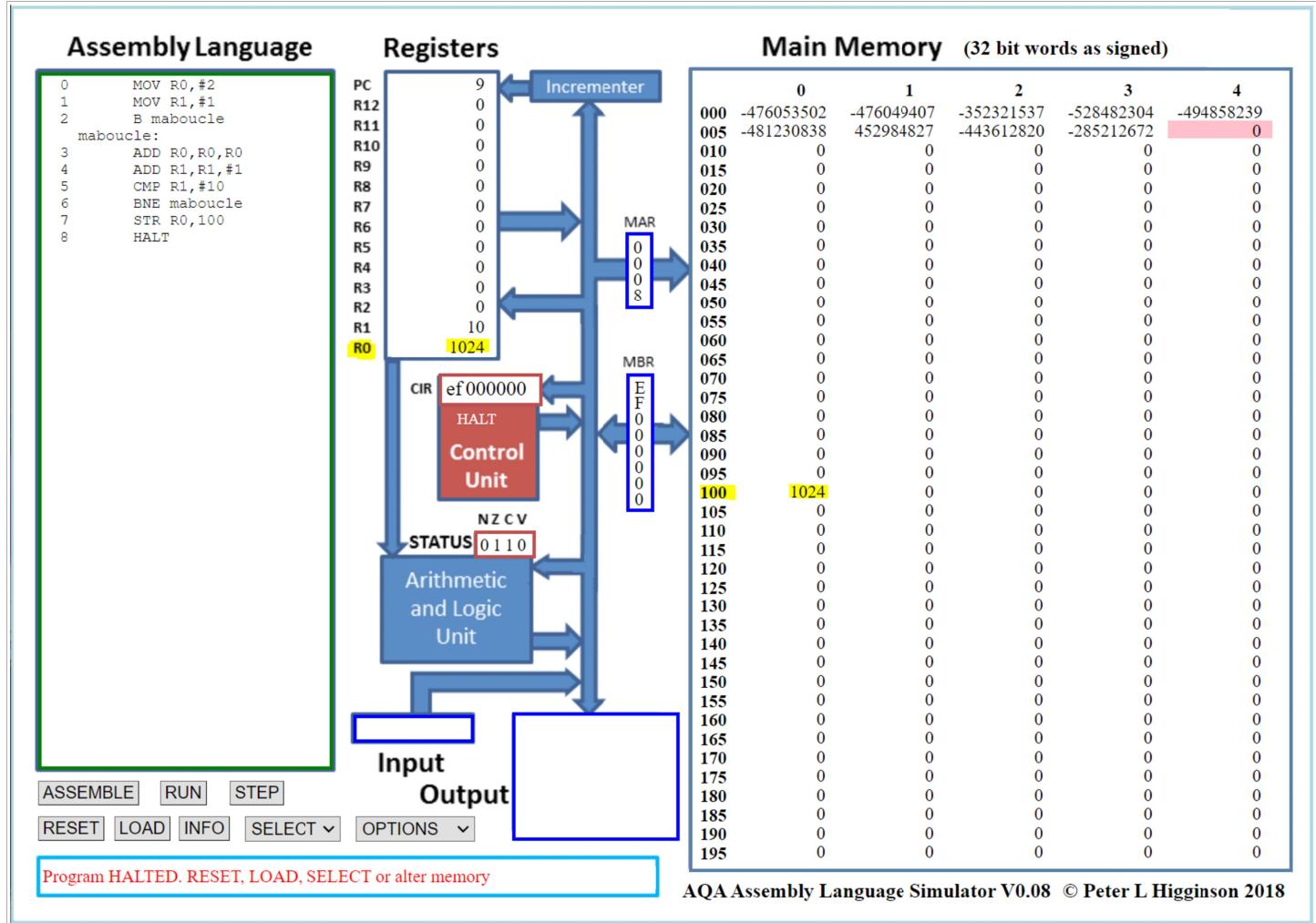
Assembly Language :

```

MOV R0, #2
ADD R0, R0, R0
ADD R0, R0, R0
ADD R0, R0, R0
ADD R0, R0, R0
OUT R0, 4
HALT
    
```

Sachant que faire l'opération 2^5 revient à faire : $2 \times 2 \times 2 \times 2 \times 2$, je procède donc de la manière suivante : $2+2=4$ $4+4=8$ $8+8=16$ $16+16=\underline{32}$ qui est bien égale à 2^5 . Alors je place le premier 2 dans le registre R0, auquel je vais additionner 4 fois de suite le résultat de l'addition précédente. Et enfin je retourne en sortie le résultat de ces additions successives qui est contenu toujours dans le même registre à savoir R0.

Programme calculant 2^{10} en utilisant qu'un seul registre et avec une boucle :



Assembly Language :

```

MOV R0,#2
MOV R1,#1
B maboucle
maboucle :
ADD R0,R0,R0
ADD R1,R1,#1
CMP R1,#10
BNE maboucle
STR R0,100
HALT
    
```

Interprétation du code :

MOV R0, #2	Place le nombre 2 dans le registre R0
MOV R1, #1	Place le nombre 1 dans le registre R1 (ce sera le compteur)
B maboucle	B signifie "Branch" et maboucle est un label. « B maboucle » renvoie l'exécution à maboucle:
maboucle:	Définition du label
ADD R0, R0, R0	Additionne la valeur du registre R0 avec celle du registre R0 puis place le résultat dans le registre R0. En clair, ici c'est l'instruction qui additionne plusieurs fois le nombre 2 comme dans l'exemple précédent.
ADD R1, R1, #1	Additionne la valeur du registre R1 avec une valeur 1 puis place le résultat dans le registre R1. En clair, ici on compte le nombre de fois que la boucle s'exécute pour qu'elle puisse s'arrêter à 10.
CMP R1, #10	Compare la valeur du registre R1 avec la valeur de 10. En clair, ici on vérifie si le compteur a atteint sa fin de course, c'est à dire 10 ici.
BNE maboucle	BNE signifie "Branch Not Equal", si la comparaison précédente n'est pas vraie alors cette instruction renvoie l'exécution tout au début de la boucle « maboucle » sinon l'exécution continue à l'étape suivante dans le code.
STR R0, 100	Enregistre la valeur du registre R0 dans la mémoire qui a pour adresse « 100 »
HALT	Arrête l'exécution du programme

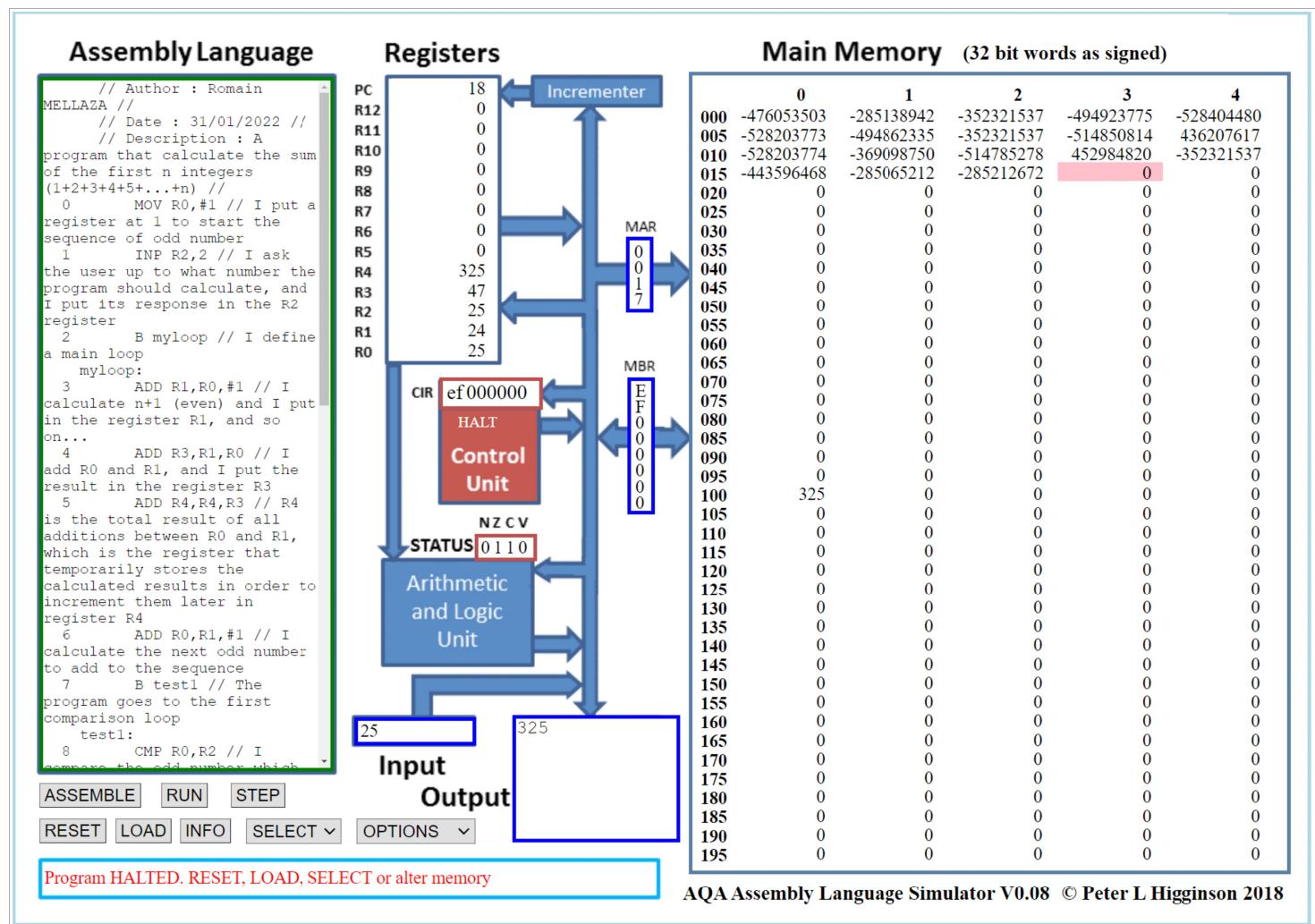
Sachant que faire l'opération 2^{10} revient à faire : $2 \times 2 \times 2$, je procède donc de la manière suivante :

$$2+2=4 \quad 4+4=8 \quad 8+8=16 \quad 16+16=32 \quad 32+32=64 \quad 64+64=128 \quad 128+128=256$$

$$256+256=512 \quad 512+512=\underline{\underline{1024}}$$

Tout d'abord je place le premier 2 dans le registre R0, auquel je vais additionner 9 fois de suite (**on pars de 1 et pas de 0**) le résultat de l'addition précédente. Et enfin je stocke le résultat de ces additions successives dans la mémoire qui a pour adresse « 100 ».

Programme calculant la somme des « n » premiers entiers :



Assembly Language:

```
// Author      : Romain MELLAZA      //
// Date        : 31/01/2022          //
// Description : A program that calculate the sum of the first n
integers (1+2+3+4+5+...+n)      //

    MOV R0,#1           // I put a register at 1 to start the sequence
                          // of odd number
    INP R2,2            // I ask the user up to what number the
                          // program should calculate, and I put its
                          // response in the R2 register
    B myloop            // I define a main loop
myloop:
    ADD R1,R0,#1       // I calculate n+1 (even) and I put in the
                          // register R1, and so on...
    ADD R3,R1,R0       // I add R0 and R1, and I put the result in
                          // the register R3
    ADD R4,R4,R3       // R4 is the total result of all additions
                          // between R0 and R1, which is the register
                          // that temporarily stores the calculated
                          // results in order to increment them later
                          // in register R4.
    ADD R0,R1,#1       // I calculate the next odd number to add to
                          // the sequence.
    B test1             // The program goes to the first comparison
                          // loop.

test1:
    CMP R0,R2          // I compare the odd number which is in the
                          // current addition with the requested
                          // maximum, this comparison can only be true
                          // if the maximum is also odd.
    BNE test2           // If the comparison is not equal, then I move
                          // on to the next test which does exactly the
                          // same thing but for even numbers this time.
    ADD R4,R4,R2       // If the comparison is equal, then I add a
                          // step to the final result because my main
                          // loop does the additions 2 by 2.
    B final              // The program goes to the final loop

test2:
    CMP R1,R2          // I compare the even number which is in the
                          // current addition with the requested
                          // maximum, this comparison can only be true
                          // if the maximum is also even.
```

```

BNE myloop      // If the comparison is not equal, then the
                program returns to the main loop because
                this means that all the comparisons (even
                or odd) have concluded that the maximum
                has not yet been reached and that it is
                necessary to continue adding.
B final         // The program goes to the final loop
final:
STR R4,100      // I store the final result in memory 100
OUT R4,4         // I output the final result of the sequence
HALT            // I stop the execution of the program

```

J'ai écrit quelques commentaires en anglais pour expliquer mon processus, qui m'a permis de résoudre la problématique de l'énoncé.

Conversion d'un programme Python en langage Assembleur ARM :

Python Language :

```

x = 4
y = 8
if(x==y):
    y = x-4
else:
    y = x+y

```

Assembly Language :

```

// Author      : Romain MELLAZA          //
// Date        : 31/01/2022             //
// Description : A program that compares an x and y value, and
acts differently depending on the answer.      //

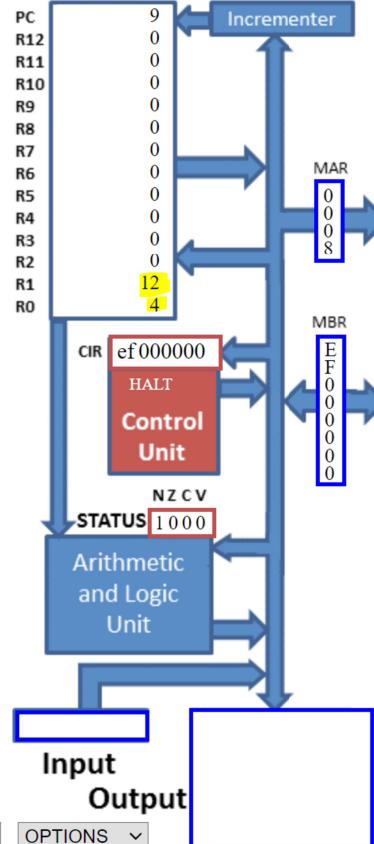
MOV R0,#4           //x
MOV R1,#8           //y
B if
if:
    CMP R0,R1
    BNE else
    SUB R1,R0,#4
    HALT
else:
    ADD R1,R0,R1
    HALT

```

Assembly Language

```
// Author : Romain
MELLAZA //
// Date : 31/01/2022 //
// Description : A
program that compares an x and
y value, and acts differently
depending on the answer. //
0    MOV R0,#4 //x
1    MOV R1,#8 //y
2    B if
if:
3    CMP R0,R1
4    BNE else
5    SUB R1,R0,#4
6    HALT
else:
7    ADD R1,R0,R1
8    HALT
```

Registers



Main Memory (32 bit words as signed)

	0	1	2	3	4
000	-476053500	-476049400	-352321537	-514850815	436207617
005	-499118076	-285212672	-528478207	-285212672	0
010	0	0	0	0	0
015	0	0	0	0	0
020	0	0	0	0	0
025	0	0	0	0	0
030	0	0	0	0	0
035	0	0	0	0	0
040	0	0	0	0	0
045	0	0	0	0	0
050	0	0	0	0	0
055	0	0	0	0	0
060	0	0	0	0	0
065	0	0	0	0	0
070	0	0	0	0	0
075	0	0	0	0	0
080	0	0	0	0	0
085	0	0	0	0	0
090	0	0	0	0	0
095	0	0	0	0	0
100	0	0	0	0	0
105	0	0	0	0	0
110	0	0	0	0	0
115	0	0	0	0	0
120	0	0	0	0	0
125	0	0	0	0	0
130	0	0	0	0	0
135	0	0	0	0	0
140	0	0	0	0	0
145	0	0	0	0	0
150	0	0	0	0	0
155	0	0	0	0	0
160	0	0	0	0	0
165	0	0	0	0	0
170	0	0	0	0	0
175	0	0	0	0	0
180	0	0	0	0	0
185	0	0	0	0	0
190	0	0	0	0	0
195	0	0	0	0	0

ASSEMBLE RUN STEP
RESET LOAD INFO SELECT ▾ OPTIONS ▾

Program HALTED. RESET, LOAD, SELECT or alter memory

AQA Assembly Language Simulator V0.08 © Peter L Higginson 2018

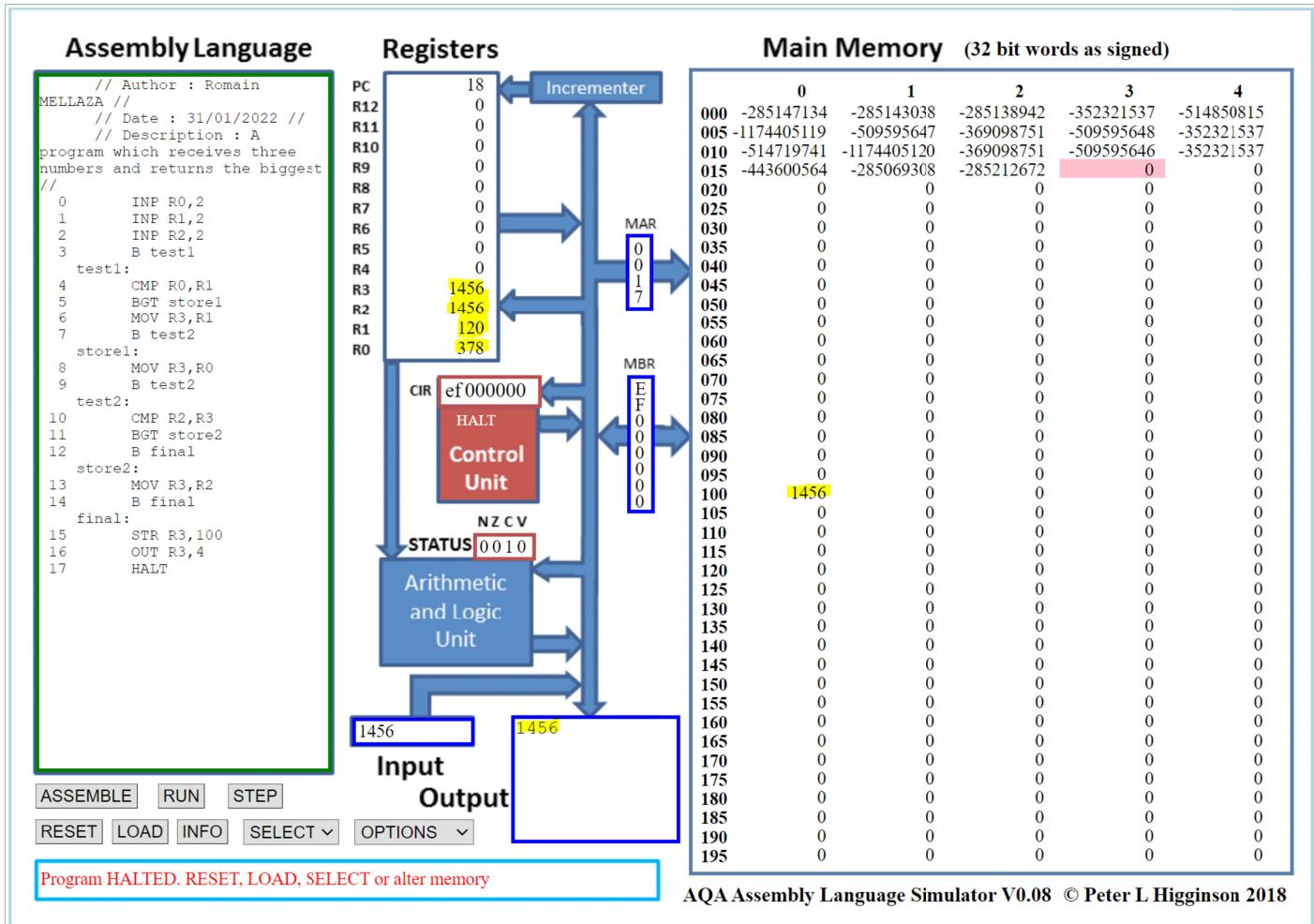
Comme on peut le voir dans cette simulation, à la fin de l'exécution je trouve $x=4$ et $y=12$. J'ai exécuté exactement le même code mais cette fois ci dans un interpréteur python et j'ai trouvé exactement le même résultat :

```
1  x = 4
2  y = 8
3  if(x==y):
4      y = x-4
5  else:
6      y = x+y
7
8  print("x =",x)
9  print("y =",y)
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL

x = 4
y = 12

Développement d'un programme qui stocke le plus grand de 3 nombres :



Mon programme utilise la logique suivante :

- En comparant d'abord R0 et R1 je ne garde que le plus grand des deux, puis en comparant ce nombre avec R2 (soit le troisième nombre) alors je suis sûr de trouver le plus grand des trois nombres en sortie de cette deuxième comparaison !

Pour la démonstration j'ai choisi trois nombres : 378 ; 120 ; 1456 et comme on peut le voir mon programme me ramène bien 1456, j'ai essayé tout les ordres possibles afin de m'assurer qu'à chaque fois c'est bien 1456 en résultat, et c'est bel et bien le cas !

Assembly Language :

```
// Author      : Romain MELLAZA          //
// Date        : 31/01/2022              //
// Description : A program which receives three numbers and
// returns the biggest.                  //

    INP R0,2
    INP R1,2
    INP R2,2
    B test1
test1:
    CMP R0,R1
    BGT store1
    MOV R3,R1
    B test2
store1:
    MOV R3,R0
    B test2
test2:
    CMP R2,R3
    BGT store2
    B final
store2:
    MOV R3,R2
    B final
final:
    STR R3,100
    OUT R3,4
    HALT
```

En complément de ce compte-rendu vous trouverez ci-joint dans le dossier, les programmes codés en assembleur ARM (.asm) afin de les tester par vous même !