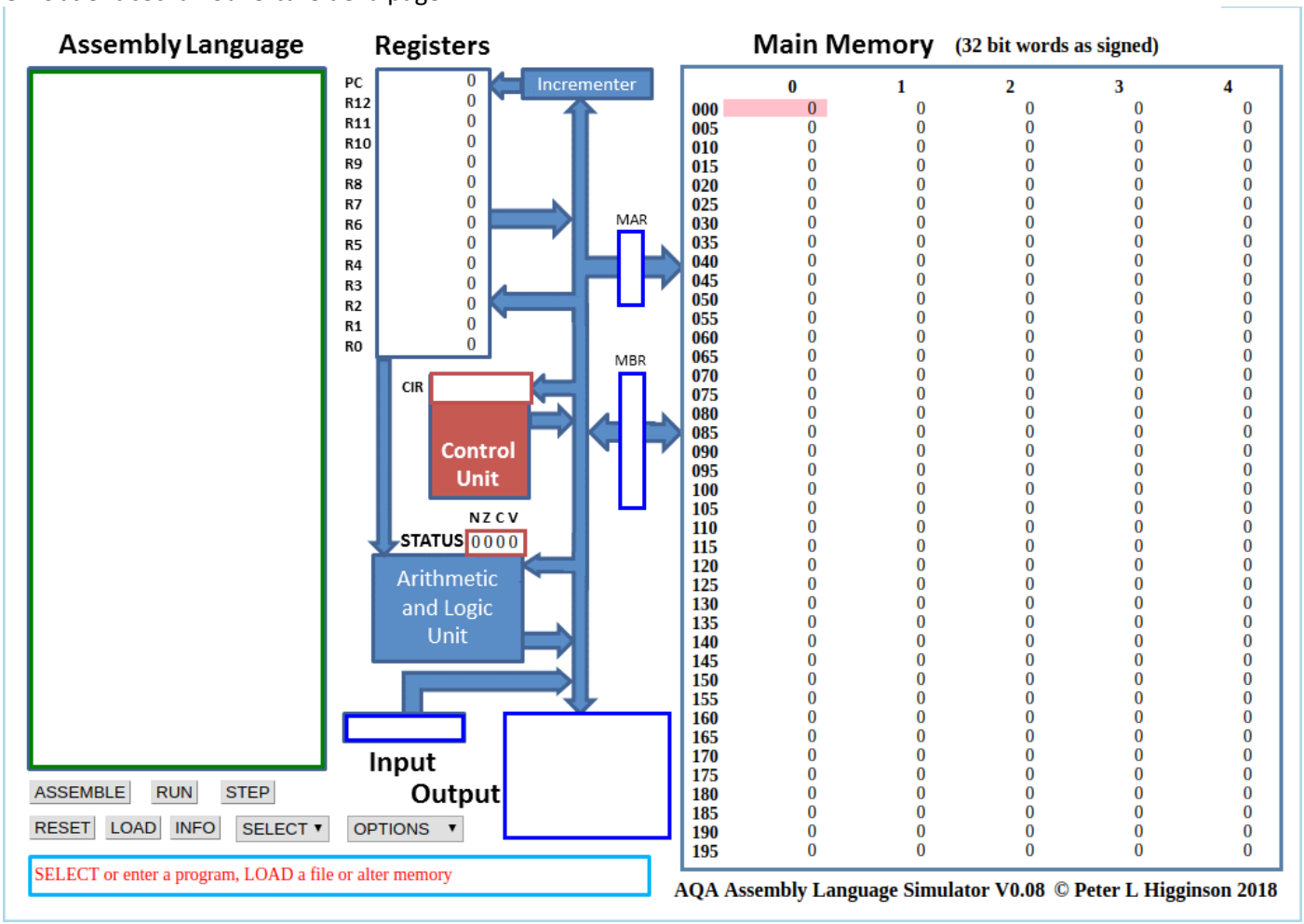
AES. Simulation. Processeur

Afin de mettre en pratique ce que a été étudié dans le cours "AES. Processeur", on va utiliser un simulateur développé par Peter L Higginson. Ce simulateur est basé sur une **architecture de von Neumann**. On va donc trouver dans ce simulateur :

- une RAM
- un CPU
- des entrées/sorties (input / output)

Une version en ligne de ce simulateur est disponible ici : http://www.peterhigginson.co.uk/AQA/
On obtient ceci à l'ouverture de la page :



Il est relativement facile de distinguer les différentes parties du simulateur :

- à droite, on trouve la **mémoire vive** (*main memory*),
- au centre, on trouve le **microprocesseur**,
- à gauche on trouve la zone d'édition (*Assembly Language*), c'est dans cette zone l'on saisit les programmes en assembleur,
- les entrées / sorties (*Input / Ouput*).

Pour ce qui est du CPU, il est composé ici de 12 registres et du registre PC (qui permet de pointer vers l'instruction à réaliser, il est incrémenté à chaque fois), de l'unité de calculs arithmétique et logique et l'unité de contrôle.

Question 1 :
À l'aide du bouton "OPTIONS", **passer** à un affichage en **binaire**.

Comme on peut le constater, chaque cellule de la mémoire comporte 32 bits (nous avons vu que classiquement une cellule de RAM comporte 8 bits). Chaque cellule de la mémoire possède une adresse (de 000 à 199), ces adresses sont codées en base 10.

Repasser à un affichage en **base 10** (bouton "OPTION"->"signed") pour faciliter la lecture

Question 2 :
Voici un petit code en assembleur :

```
MOV R0,#42

STR R0,150

HALT
```

Le **recopier** dans la zone dédiée. Une fois la saisie terminée, **cliquer** sur le bouton "submit". On doit voir apparaitre des nombres "étranges" dans les cellules mémoires d'adresses 000, 001 et 002 comme ci-dessous :

Main Memory (32 bit words as signed)

	0	1	2	3
000	-476053462	-443612596	-285212672	0
005	0	0	0	0

L'assembleur a fait son travail, il a converti les 3 lignes de notre programme en instructions machines.

- la première instruction machine est stockée à l'adresse mémoire 000 (elle correspond à "MOV R0,#42" en assembleur),
- la deuxième à l'adresse 001 (elle correspond à "STR R0,150" en assembleur),
- la troisième à l'adresse 002 (elle correspond à "HALT" en assembleur)

Remarque : pour avoir une idée des véritables instructions machines, il faut repasser à un affichage en binaire ((bouton "OPTION"->"binary")). On obtient alors ceci :

Main Memory (32 bit words as binary)

	0	1	2
000	11100011 10100000 00000000 00101010	11100101 10001111 00000010 01001100	11101111 00000000 00000000 00000000
005	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000

On peut analyser chacune des étapes du programme :

- 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".

Au passage, pour l'instruction machine "11100011 10100000 00000000 00101010", on remarque que l'octet le plus à droite, $(00101010)_2$, est bien égal à $(42)_{10}$!

Repasser à un affichage en **base 10** afin de faciliter la lecture des données présentes en mémoire.

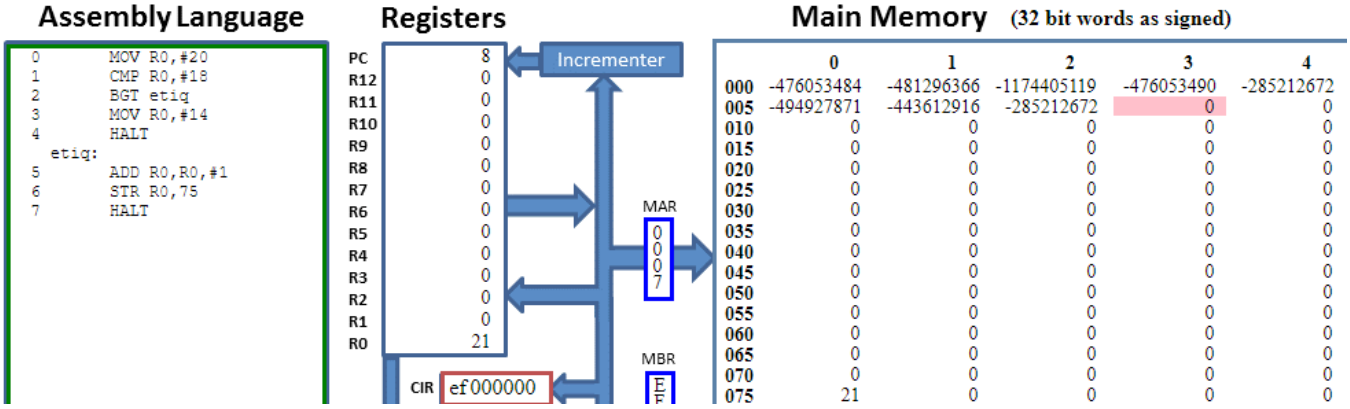
Question 3 :

Pour **exécuter** le programme, il suffit de cliquer sur le bouton « RUN ». On doit obtenir ceci à la fin de la simulation (voir à droite).

Une fois la simulation terminée, on peut constater que la cellule mémoire d'adresse 150, contient bien le nombre 42 (en base 10). Vous pouvez aussi constater que le registre R0 a bien stocké le nombre 42.

ATTENTION : pour **relancer** la simulation, il est nécessaire d'appuyer sur le bouton "RESET" afin de remettre les registres R0 à R12 à 0, ainsi que le registre PC (il faut que l'unité de commande pointe de nouveau sur l'instruction située à l'adresse mémoire 000).
La mémoire n'est pas modifiée par un appui sur le bouton "RESET", pour remettre la mémoire à 0, il faut cliquer sur le bouton "OPTIONS" et choisir "clr memory".
Pour remettre la mémoire à 0, il faudra cliquer sur le bouton "ASSEMBLE" avant de pouvoir exécuter de nouveau votre programme.

Question 4 :
Recopier l'exemple du cours, on doit obtenir ceci à la fin de la simulation :



Remarques importantes :

- S'il y a des **erreurs**, elles seront indiquées lors de **l'appui** sur le **bouton « SUBMIT »**.
- Il ne faut pas **mettre d'espace** lorsque l'on écrit le label « etiq: ».

Question 5 :
En utilisant les instructions en assembleur nécessaires (reprendre le cours), **calculer** le produit de 5 par 6.

Question 6 :
Traduire le programme suivant écrit en langage Python en langage assembleur.

```
x=0
while x<3:
    x=x+1
```

Question 7 :
Que font ces deux petits programmes ? **Expliquer**.

Assembly Language

```
0 MOV R0,#5
1 LSL R0,R0,#1
2 HALT
```

Assembly Language

```
0 MOV R0,#5
1 LSL R0,R0,#2
2 HALT
```