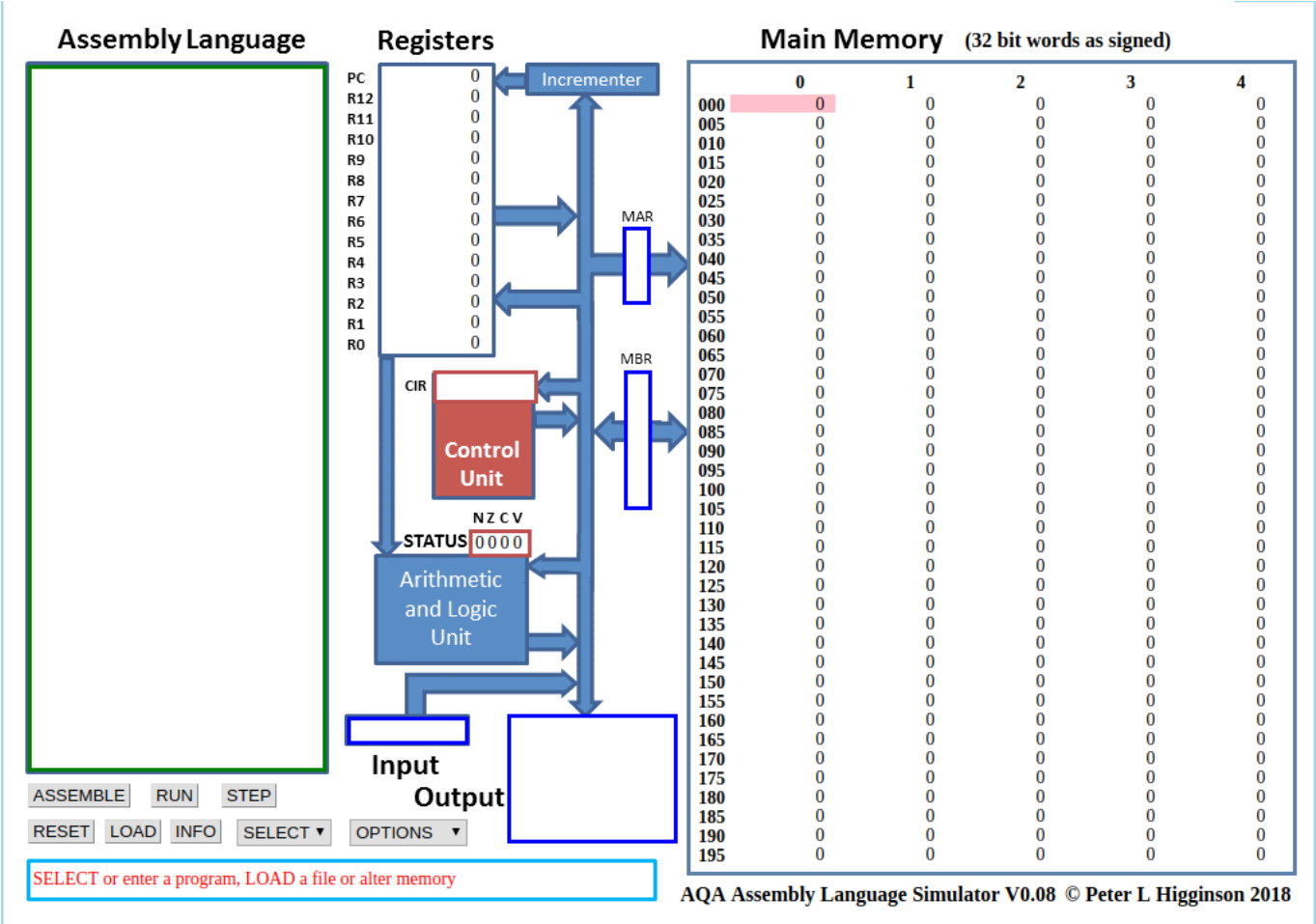
Un simulateur de CPU

Afin de faire le lien entre le cours intitulé "Modèle d'architecture de von Neumann" et celui sur l'assembleur, nous allons utiliser un simulateur développé par Peter L. Higginson.
Ce simulateur est basé sur une architecture de von Neumann. Nous allons donc trouver dans ce simulateur :

- une RAM
- un CPU
- des bus (transmission de l'information)
- des entrées (le langage assembleur)

La version en ligne de ce simulateur que vous allez utiliser est disponible ici : http://www.peterhigginson.co.uk/AQA/

Allez à cette adresse, et vous devriez obtenir ceci:



On distingue déjà les différentes parties du simulateur :

- à droite, on trouve les cases mémoires de la mémoire vive ("main memory")
- au centre, on trouve le microprocesseur avec ses différents composants.
- à gauche on trouve les entrées ("Assembly Language") : c'est dans cette zone que nous allons saisir nos programmes en assembleur

Revenons rapidement sur les parties RAM et CPU :

La RAM

Par défaut le contenu des différentes cellules de la mémoire est en base 10 (entier signé), mais d'autres options sont possibles : base 10 (entier non-signé, "unsigned"), base 16 ("hex"), base 2 ("binary").
On accède à ces options à l'aide du bouton "OPTIONS" situé en bas dans la partie gauche du simulateur.

❒ — À faire vous-même **1** —

À l'aide du bouton "OPTIONS", passez à l'affichage en binaire.
Comme vous pouvez 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 aussi une adresse (de 000 à 199), ces adresses sont codées en base 10 et sont visibles sur la première ligne en haut et sur la première colonne à gauche.
Repassez à un affichage en base 10 (bouton "OPTION"->"signed")

Le CPU

Dans la partie centrale du simulateur, nous allons trouver en allant du haut vers le bas :

- le bloc "registre" ("Registers") : nous avons 13 registres (R0 à R12) + 1 registre (PC) qui contient l'adresse mémoire de l'instruction en court d'exécution
- le bloc "unité de commande" ("Control Unit") qui contient l'instruction machine en cours d'exécution (au format hexadécimal)
- le bloc "unité arithmétique et logique" ("Arithmetic and Logic Unit")

Nous ne nous intéresserons pas aux autres composants de la partie CPU mais vous en avez sans doute entendu parler dans vos recherches de la semaine dernière.

Programmer en assembleur

Comme déjà dit plus haut, la partie de gauche permet de saisir des programmes en assembleur. Les instructions de cet assembleur sont légèrement différentes des instructions vues dans le cours et ce pour souligner que chaque assembleur est différent.

❐ — À faire vous-même **2** —

Dans la partie "Assembly Language" saisissez les lignes de codes suivantes (ne lancez pas encore votre programme) :

```
MOV R0,#42
STR R0,199
HALT
```

La première ligne stocke **le nombre 42** (#42) dans le registre 0 **(R0)**.
La seconde ligne envoie le contenu du registre 0 **(R0)** à l'adresse 199 de la RAM.

Une fois la saisie terminée, cliquez sur le bouton "submit". Vous devriez voir apparaitre des nombres "étranges" dans les cellules mémoires d'adresses 000, 001 et 002 :

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) et la troisième à l'adresse 002 (elle correspond à "HALT" en assembleur) Pour avoir une idée des véritables instructions machines, repassez à un affichage en binaire (bouton "OPTION"->"binary"). Vous devriez obtenir 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

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 R0,199"
- l'instruction machine "11101111 00000000 00000000 00000000" correspond au code assembleur "HALT"

❐ — À faire vous-même **3** —

Dans les exercices sur l'assembleur, nous avons vu que les instructions binaires étaient souvent découpées en deux parties : une partie instruction et une partie contenu.

Nous allons essayer de décoder ces parties dans le cas de notre simulateur.

- Passez en représentation hexadécimal (hex). Souvenez-vous que l'hexa est le regroupement de 4 bits.
- Notez les adresses hexadécimales qui apparaissent dans les cases mémoire 0, 1 et 2.
- Changez votre programme en assembleur en utilisant maintenant le registre R1 au lieu de R0. Notez à nouveau les adresses hexadécimales qui apparaissent dans les cases mémoire 0, 1 et 2.
- À nouveau, changez votre programme en assembleur en utilisant maintenant le registre R10 au lieu de R1. Notez les changements dans la RAM.

A partir de vos tests, répondez aux questions ci-dessous :

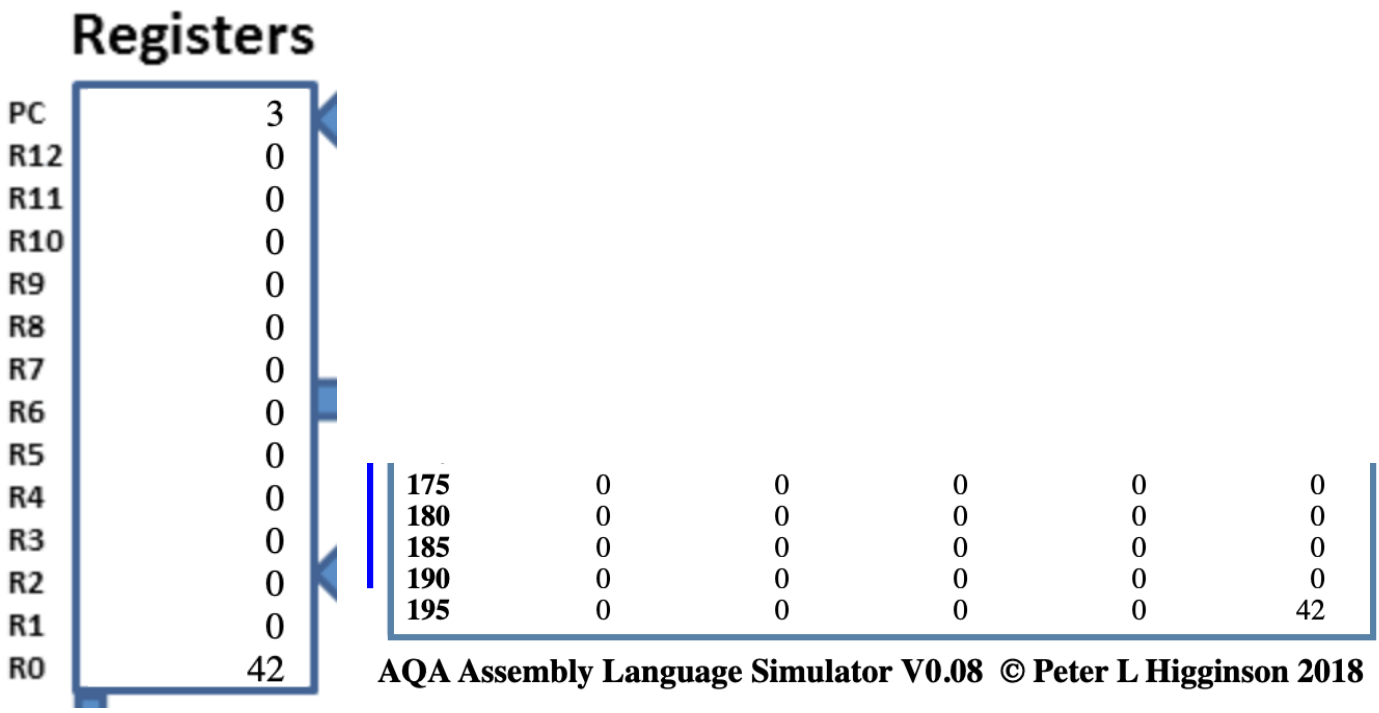
- Sur quel nombre de bits le contenu peut-il être stocké ? En déduire le nombre maximum que l'on peut stocker dans cet architecture.
- Sur combien de bits les registres sont-ils stocké ?
- Quel est l'instruction binaire correspondant à MOV ? Sur combien de bits est-elle stockée ?

❐ — À faire vous-même **4** —

Pour exécuter notre programme, il suffit maintenant de cliquer sur le bouton "RUN".

Vous allez voir le CPU effectuer vos opérations en direct grâce à de petites animations. Cela va très vite, régler donc immédiatement la vitesse de simulation à l'aide des boutons "<<" et ">>". Un appui sur le bouton "STOP" met en pause la simulation, si vous rappuyez une deuxième fois sur ce même bouton "STOP", la simulation reprend là où elle s'était arrêtée.

Une fois la simulation terminée, vous pouvez constater que la cellule mémoire d'adresse 199, 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". Si vous remettez votre mémoire à 0, il faudra cliquer sur le bouton "ASSEMBLE" avant de pouvoir exécuter de nouveau votre programme.

❐ — À faire vous-même **5** —

Modifiez le programme précédent pour qu'à la fin de l'exécution on trouve le nombre 54 à l'adresse mémoire 50. On utilisera le registre R1 à la place du registre R0. Testez vos modifications en exécutant la simulation.

Le simulateur prend en charge les instructions complexes (CMP, BNE = vérification de non-égalité etc.). Allez voir les informations sur ces instructions en cliquant sur le bouton INFO. Il existe des "labels" que nous avons appelé des ADDR (adresse) dans les exercices sur l'assembleur.

❒ — À faire vous-même **6** —

Saisissez et testez le programme suivant :

```
  MOV R0, #4
  STR R0,30
  MOV R0, #8
  STR R0,75
  LDR R0,30
  CMP R0, #10
  BNE else
  MOV R0, #9
  STR R0,75
  B endif
else:
  LDR R0,30
  ADD R0, R0, #1
  STR R0,30
endif:
  MOV R0, #6
  STR R0,23
  HALT
```

Que semble faire ce programme ?

❒ — À faire vous-même **7** —

À partir du programme ci-dessus, et testez le programme suivant :

```
  MOV R0, #4
  STR R0,30
  MOV R0, #8
  STR R0,75
  LDR R0,30
  CMP R0, #10
  BNE else
  MOV R0, #9
  STR R0,75
  B endif
else:
  LDR R0,30
  ADD R0, R0, #1
```

```
   STR R0,30
endif:
   MOV R0, #6
   STR R0,23
   HALT
```

Que semble faire ce programme ?

❐ — À faire vous-même **7** —

Voici un programme Python :

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

Écrivez et testez un programme en assembleur équivalent au programme ci-dessus.

❐ — À faire vous-même **8** —

Reprenez l'exercice 4 du cours sur l'assembleur et écrivez le programme Assembleur permettant de faire la multiplication d'un entier a par un entier b (tous deux non nuls).