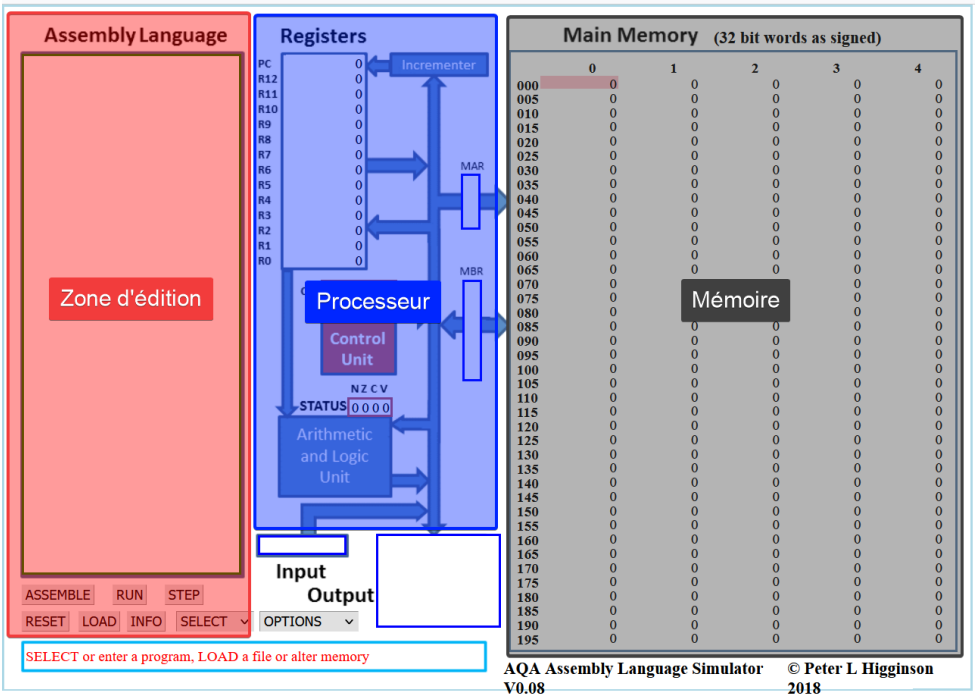
TP : Le simulateur AQA

Nous allons utiliser un simulateur d'architecture de Von Neumann, réalisé par Peter Higginson, professeur d'informatique anglais aujourd'hui à la retraite, pour préparer des étudiants anglais à leur examen de science informatique. AQA est le nom de l'Assessment and Qualifications Alliance, organisme de certification britannique. On peut l'exécuter en ligne sur :

http://www.peterhigginson.co.uk/AQA/

L'interface reprend les éléments de l'architecture de von Neumann :

- Dans la zone centrale, on trouve le processeur composé de l'unité de contrôle, de l'unité arithmétique et logique, et de 13 registres nommés R0 à R12. Les flèches bleues représentent les bus. Le registre PC contient l'adresse mémoire de l'instruction en cours d'exécution, le bloc Control Unit contient l'instruction machine en cours d'exécution codée en hexadécimal.
- Dans la zone à droite se trouve la mémoire : elle contient les instructions et les données sous la forme de mots de 32 bits écrits par défaut en base 10 avec signe. On peut, via le bouton OPTIONS, choisir un affichage en entier non signé en base 10 ou en binaire. La mémoire comporte 200 adresses.
- Dana la zone de gauche se trouve la zone d'édition du programme en assembleur.



Exercice 1 : découverte du simulateur

1. Dans le menu OPTIONS, choisir l'affichage en binaire.
2. Dans la zone d'édition, saisir le programme suivant, puis cliquer sur « Submit ».

```
MOV R0,#42
STR R0,150
HALT
```

3. L'assembleur a traduit les trois instructions en instructions machines et a placé ces instructions dans les adresses mémoires 000, 001 et 002.

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

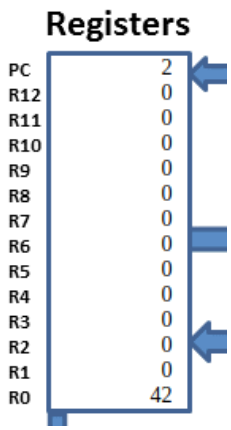
Nous pouvons donc maintenant affirmer que :

- 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 R0,150 ;
- l'instruction machine 11101111 00000000 00000000 00000000 correspond au code assembleur HALT

Au passage, pour l'instruction machine 11100011 10100000 00000000 00101010, vous pouvez remarquer que l'octet le plus à droite, 00101010_2, est bien égale à 42_{10} !

4. Repasser à un affichage en base 10 unsigned afin de faciliter la lecture des données présentes en mémoire.
5. Pour exécuter notre programme, il suffit maintenant de cliquer sur le bouton "RUN". Vous allez voir le CPU "travailler" en direct grâce à de petites animations. Si cela va trop vite (ou trop doucement), vous pouvez régler la vitesse de simulation à l'aide des boutons "<<" et ">>". Un appui sur le bouton "STOP" met en pause la simulation, si vous appuyez une deuxième fois sur ce même bouton "STOP", la simulation reprend là où elle s'était arrêtée.
6. Une fois la simulation terminée, vous pouvez 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.



130	0	0
135	0	0
140	0	0
145	0	0
150	42	0
155	0	0

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.

7. Modifier 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. Tester vos modifications en exécutant la simulation.

Exercice 2

1. Saisir le programme ci-dessous, puis cliquer sur « Submit » :

```
MOV R0, #10
LDR R1, 10
ADD R2, R1, R0
STR R2, 11
HALT
```

2. Observer le contenu de la première ligne de la mémoire (adresses 000 à 004). À quoi correspondent ces nombres ? Donner l'instruction binaire qui correspond à l'instruction `ADD R2, R1, R0`.
3. Placer le nombre 12 à l'adresse mémoire 10.
4. Exécuter le programme pas à pas en cliquant sur STEP : à chaque clic sur STEP, une ligne du programme est exécutée.
   Décrire l'enchaînement d'opérations élémentaires lors de l'exécution des instructions de transfert de mémoire « MOV R0,#10 » puis « LDR R1,10 ». Observer l'évolution des registres PC (Compteur de programme), CIR (Registre d'instructions), MAR (adresse d'écriture/lecture en mémoire) et MBR (donnée à lire/écrire). Pour quelle(s) instruction(s), l'ALU est-elle sollicitée ?

Exercice 3

On considère le programme Python suivant :

```
a = 42   # valeur 42 à l'adresse 20 en mémoire centrale
b = 69   # valeur 69 à l'adresse 21 en mémoire centrale
a = a + b   # changement de valeur à l'adresse 20
b = a - b   # changement de valeur à l'adresse 21
a = a - b   # changement de valeur à l'adresse 20
```

1. Déterminer le contenu des variables a et b à la fin de l'exécution de ce programme Python.
2. Traduire ce programme en assembleur et le tester dans le simulateur.

En assembleur, les identifiants de variables sont remplacés par des adresses en mémoire centrale et les opérations arithmétiques ne sont effectuées que sur des registres, il faut donc d'abord transférer les opérandes de la mémoire centrale vers des registres.

Exercice 4

Dans cet exercice, vous allez utiliser des labels afin de coder en assembleur des structures conditionnelles.

On considère le programme suivant, écrit en assembleur :

```
//Lecture d'un entier dans Input et chargement dans le registre R0
INP R0, 2
//Comparaison du registre R0 avec le nombre 0
CMP R0, #0
//Branchement conditionnel sur l'étiquette else si R0 négatif
BLT else
MOV R1, R0
//Branchement inconditionnel sur l'étiquette fin
B fin
//étiquette else
else:
MOV R2, #0
SUB R1, R2, R0
//étiquette fin
fin:
//affichage du registre R1 dans Output
OUT R1, 4
HALT
```

1. Entrer ce programme dans le simulateur (il n'est pas nécessaire de taper les commentaires) et observer son exécution avec différentes valeurs entières signées en entrée. Conjecturer ce que fait ce programme.
2. Traduire ce programme en Python.

Exercice 5

Dans cet exercice, les labels seront utilisés pour coder une boucle en assembleur.

Voici un programme en Python :

```
x=0
while x<3:
    x=x+1
print(x)
```

Écrire et tester un programme en assembleur équivalent au programme ci-dessus.

Exercice 6

Traduire le programme Python suivant en assembleur et le tester dans le simulateur.

```
a = int(input()) #entier lu stocké dans le registre R0
b = int(input()) #entier lu stocké dans le registre R1
if a > b:
    m = a
else:
    m = b
#le maximum m de a et b est stocké dans le registre R2
#et en mémoire centrale à l'adresse 20
print(m)
```