Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI 1re – Thème 5 Année **2021-22**

Programmation en assembleur

VisUAL

Afin de programmer en assembleur, nous allons utiliser lo logiciel `VisUAL` qui est un simulateur de processeur ARM.
La page de présentation du logiciel se trouve ici :
https://salmanarif.bitbucket.io/visual/index.html
La liste des commandes supportées est ici :
https://salmanarif.bitbucket.io/visual/supported_instructions.html

Premiers programmes

Chacun des exercices devra être fait dans un fichier différent : `ex1.visual`, `ex2.visual`...
Vous pouvez aussi donner un nom plus explicite.

Exercice 1 : Recopier le programme de l'exercice 1 de la feuille sur l'assembleur et vérifier le résultat obtenu.

```
        MOV     R0, #10
        MOV     R1, #3
        ADD     R2, R0, R1
        ADD     R2, R0, R2
```

Exercice 2 : Recopier le programme que vous avez écrit pour l'exercice 2 de la feuille sur l'assembleur et vérifier le résultat obtenu.

Exercice 3 :

1) Recopier le programme de l'exercice 3 de la feuille sur l'assembleur et vérifier le résultat obtenu. Vous pouvez utiliser le bouton "Step Forwards" pour exécuter les instructions une par une. Il est aussi possible de placer un point d'arrêt en cliquant juste à droite du numéro d'une ligne. Cela met un point rouge au début de cette ligne. Lors de l'exécution, le programme fera toujours une pause à cet endroit pour que vous puissiez voir la valeur des différents registres.

```
        MOV     R0, #10
        MOV     R1, #3
        CMP     R0, R1
        BGE     label2
label1  SUB     R2, R1, R0
        END
label2  SUB     R2, R0, R1
        END
```

2) Modifier le programme pour passer par le label `label1`.

Exercice 4 : Recopier le programme que vous avez écrit pour l'exercice 4 de la feuille sur l'assembleur et vérifier le résultat obtenu.

Exercice 5 : Recopier le premier exemple de la feuille sur l'assembleur.

```
init    MOV     R0, #4      ; R0 = 4
        MOV     R1, #5      ; R1 = 5
        MOV     R2, #0      ; R2 = 0, le résultat
boucle  CMP     R1, #0      ; Si R1 = 0, on a fini
        BEQ     fin         ; On passe à la fin si R1 = 0
        ADD     R2, R0, R2  ; Sinon, R2 = R2 + R0
        SUB     R1, R1, #1  ; R1 = R1 - 1
        B       boucle      ; On recommence
fin     END                 ; Fin du programme
```

Exercice 6 : Recopier le programme que vous avez écrit pour l'exercice 6 de la feuille sur l'assembleur et le tester.

Exercice 7 : Recopier le programme de l'exercice 6 de la feuille sur l'assembleur et vérifier le résultat obtenu.

```
init    MOV     R0, #16
        MOV     R1, #3
        MOV     R2, #0
boucle  CMP     R0, R1
        BLT     fin
        SUB     R0, R0, R1
        ADD     R2, R2, #1
        B       boucle
fin     END
```

Des programmes un peu plus complexes

Le programme de l'exercice précédent permet de faire une division euclidienne. Nous allons maintenant faire un programme permettant de diviser par 2.
Nous allons utiliser l'algorithme ci-contre, où a est le nombre à diviser par 2.

$b \leftarrow 0$
Tant que $a > b$ **:**
  $a \leftarrow a - 1$
  $b \leftarrow b + 1$

Exercice 8 : Écrire un programme en assembleur réalisant cet algorithme. Vous pourrez utiliser `R0` pour a et `R1` pour b.

En fait, la division par 2 en binaire est très simple. Il suffit d'enlever le bit de droite. Pour cela, on peut utiliser l'instruction "`LSR dest, op1, op2`" qui met dans `dest` la valeur de `op1` où les `op2` derniers bits ont été enlevés. Ainsi "`LSR R0, R0, #1`" divise `R0` par 2.

Exercice 9 : Écrire un programme en assembleur qui triple la valeur de `R0`. Le résultat doit se trouver dans `R0`.

Exercice 10 : Écrire un programme en assembleur qui calcule dans `R1` la durée de vol de la suite de Syracuse obtenue en partant de `R1`.

Exercice 11 : Modifier le programme précédent pour qu'il stocke dans `R2` la valeur maximale obtenue dans la suite de Syracuse en partant de `R0`.

Exercice 12 : Écrire un programme en assembleur qui détermine si `R0` est divisible par `R1`. Si c'est le cas, mettre 1 dans `R2`, sinon mettre 0.

Exercice 13 : Écrire un programme en assembleur qui calcule le plus grand entier n tel que $n^2 \leq$ R0. Vous pouvez pour cela stocker la valeur de n dans R1 et de n^2 dans R2. On rappelle aussi que $(n+1)^2 = n^2 + n + n + 1$.

Pour aller plus loin

Dans tous les programmes précédents, nous n'utilisons que les registres. Mais si nous voulons stocker une liste de valeurs, il faut utiliser la mémoire vive. Dans l'exemple du cours, nous avons vu que nous pouvons associer une adresse à chaque variable. Mais pour une liste, on doit mettre à jour l'adresse utilisée au fur et à mesure de l'exécution.
Le programme suivant va stocker les valeurs de 10 à 1 à partir de l'adresse 100_{16}, c'est-à-dire 256.

```
        MOV     R12, #0x100     ; adresse mémoire de départ
        MOV     R0, #10         ; valeur de départ
boucle  STR     R0, [R12]       ; on sauve R0 à l'adresse R12
        ADD     R12, R12, #4    ; on passe à l'adresse suivante
        SUB     R0, R0, #1
        CMP     R0, #0          ; on arrete quand on arrive à 0
        BGT     boucle
        END
```

Les adresses mémoires vont de 4 en 4 puisque chaque adresse peut contenir 4 octets, c'est-à-dire 32 bits.
Le langage permet à la fois de sauvegarder la valeur à l'adresse donnée, mais aussi de calculer la prochaine adresse. C'est ce que fait le programme suivant.

```
        MOV     R12, #0x100
        MOV     R0, #10
boucle  STR     R0, [R12],#4 ; sauvegarde puis ajoute 4 à R12
        SUB     R0, R0, #1
        CMP     R0, #0
        BGT     boucle
        END
```

Exercice 14 : En vous inspirant des exemples ci-dessus, modifier le programme sur la suite de Syracuse pour sauvegarder les valeurs successives.

Exercice 15 : Écrire un programme en assembleur qui détermine si le nombre R0 est premier. Vous pourrez utiliser les programmes que vous avez précédemment écrits.

Exercice 16 : Écrire un programme en assembleur qui met en mémoire la liste des nombres premiers inférieurs à n, où n est un nombre stocké dans un registre de votre choix.