Exercices ASSEMBLEUR

I. ADDITION DE DEUX NOMBRES

Problème : on désire réaliser en assembleur l'équivalent de l'instruction : `a = b + c`

L'assembleur ne connaît pas la notion de variables ; il faut explicitement lui préciser quelle adresse mémoire on veut utiliser pour y stocker les données et les résultats de calcul. Par exemple, dans les exercices suivants :

- donnée `b` = adresse 20
- donnée `c` = adresse 21
- résultat `a` = adresse 22

(Remarque : on peut choisir bien entendu d'autres adresses pour les données (et le résultat), mais il faut s'assurer qu'elles ne viendront pas « empiéter » sur les instructions du programme (et oui, programme et données dans la même mémoire...) ; on a donc choisi ici de les mettre « suffisamment loin » pour ne pas risquer de rencontrer ce problème...Mais c'est un problème auquel devait veiller les premiers programmeurs !)

Programme proposé :

Adresse	Instruction	Contenu des adresses ou des registres				Commentaire
		20	**21**	**A**	**22**	
Avant exécution		23	31	0	0	
0	`SPEED 127`					
2	`COPYRA 20`					
4	`ADDRA 21`					
6	`COPYAR 22`					
8	`HALT`					

1. A l'aide du manuel, déterminer le rôle des différentes instructions.
2. Compléter le tableau avec la trace du programme à chaque pas d'exécution.
3. Faire vérifier votre travail, puis passer à la feuille de calcul où vous entrerez les instructions assembleur.
4. Programmer enfin le Digirule ; *en plus du programme, n'oubliez pas également de stocker les données 23 et 31 aux adresses 20 et 21*
5. Lancer l'exécution du programme, puis accéder à l'adresse 22 pour visualiser son contenu sur les LEDs de données : le calcul est-il correct ?
6. Modifier les données stockées aux adresses 20 et/ou 21 pour vérifier le bon fonctionnement du programme.

II. SOUSTRACTION DE DEUX NOMBRES

Problème : on désire réaliser en assembleur l'équivalent de l'instruction : `a = b - c`

Programme :

Adresse	Instruction	Contenu des adresses ou des registres				Commentaire
		20	**21**	**A**	**22**	
Avant exécution		23	31	0	0	
0	`SPEED 127`					
2	`COPYRA 20`					
4						
6	`COPYAR 22`					
8	`HALT`					

1. A l'aide du manuel, chercher par quelle instruction il faut remplacer celle à l'adresse 4.
2. Compléter le tableau avec la trace du programme à chaque pas d'exécution : quel résultat prévoit-on d'obtenir ?
3. Faire vérifier votre travail, puis passer à la feuille de calcul où vous entrerez les instructions assembleur.
4. Programmer enfin le Digirule.
5. Lancez l'exécution du programme, et afficher le contenu de l'adresse 22 ; vérifier la prévision faite à la question 2.

III. BOUCLE

Voila l'équivalent de la boucle `for … in range()` de Python.

Problème : afficher sur les LEDs de données les nombres binaires de 0 à 255 par pas de 1

```
for i in range(256) :
       print( i )
```

Programme :

Adresse	Instruction	Contenu des adresses ou des registres			Commentaire
		A	**255**	**15**	
Avant exécution		0	0	0	
0	SPEED 50				
2	COPYRA 15				
4	COPYAR 255				
6	INCRJZ 15				
8	JUMP 2				
10	HALT				

1. Quel est le rôle de la variable correspondant à l'adresse 15 ?
2. Indiquer les instructions correspondant à la boucle, et expliquer son fonctionnement.
3. Faire vérifier votre travail, puis passer à la feuille de calcul où vous entrerez les instructions assembleur.
4. Programmer enfin le Digirule, et vérifier la bonne exécution du programme.
5. Modifier le programme pour afficher les nombres de 255 à 0.

IV. MULTIPLICATION

Problème : on désire réaliser en assembleur l'équivalent de l'instruction : `a = b * c`

Pas d'instruction de multiplication dans le langage machine du Digirule, comme sur les premiers microprocesseurs (*les microprocesseurs actuels en disposent bien sûr*) : il va falloir faire autrement...

Principe : on prend par exemple `b = 25` et `c = 3`
→ `b * c = b + b + b`
→ le résultat du calcul est l'addition de b à lui-même, un nombre de fois égal à c
→ *il faut donc utiliser une **boucle**, dont le compteur ira de 1 à c, et dans laquelle on ajoutera b au résultat du tour de boucle précédent*

Programme proposé :

Adresse	Instruction	Contenu des adresses ou des registres				Commentaire
		20	**21**	**A**	**22**	
Avant exécution		25	3	0	0	
0	SPEED 50					
2	COPYRA 20					
4	ADDRA 22					
6	COPYAR 22					
8	DECRJZ					
10	21					
12	JUMP 2					
14	HALT					

1. Faire dans le tableau la trace du programme pour vérifier son bon fonctionnement.
2. Faire vérifier votre travail, puis passer à la feuille de calcul où vous entrerez les instructions assembleur.
3. Programmer enfin le Digirule.
4. Lancez l'exécution du programme, et afficher le contenu de l'adresse 22 : vérifier le résultat obtenu
5. Vérifiez le bon fonctionnement du programme avec d'autres données aux adresses 20 et 21.

V. IF.....ELIF....ELSE....

Les instructions conditionnelles existent en assembleur, et prennent la forme de saut conditionnel : si une condition est vérifiée, alors le PC « saute » à une adresse, sinon il continue « normalement »...Mais l'écriture des conditions est bien différente !

Problème : on veut coder l'équivalent du code Python suivant :

```
a = 156
b = 45

if ( a > b ) :
      print( a )
else :

       print( b )
```

Le programme étant plus long, les données ont donc du être « décalées » vers le haut de la mémoire :

- donnée `a` = adresse 25
- donnée `b` = adresse 26
- résultat = adresse 27

Programme proposé :

Adresse	Instruction	Contenu des adresses ou des registres				Commentaire
		25	26	A	27	
Avant exécution		156	45	0	0	
0	`SPEED 127`					
2	`COPYRA 25`					
4	`SUBLA 26`					
6	`BCRSC 1 252`					
9	`JUMP 16`					
11	`COPYRA 25`					
13	`COPYAR 27`					
15	`JUMP 21`					
17	`COPYRA 26`					
19	`COPYAR 27`					
21	`HALT`					

(*Noter que ce programme fait intervenir une instruction à deux opérandes à l'adresse 6*)

Remarque : le registre à l'adresse 252 est appelé STATUS : ses bits donnent des informations sur le résultat de l'opération ayant eu précédemment lieu :

- le bit 0 (= bit ZERO) est mis à 1 si l'opération précédente a donné un résultat nul
- le bit 1 (= bit CARRY) est mis à 1 si une retenue à eu lieu

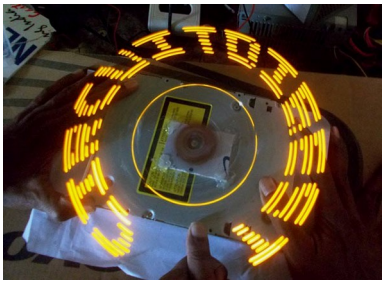
→ *c'est ce dernier bit que l'on exploite ici : en effet, si une soustraction donne un résultat positif, il n'y a pas de retenue ; si le résultat est négatif, il y a une retenue.*

1. Faire dans le tableau la trace du programme pour vérifier son bon fonctionnement.
2. Faire vérifier votre travail, puis passer à la feuille de calcul où vous entrerez les instructions assembleur.
3. Programmer enfin le Digirule.
4. Lancez l'exécution du programme, et afficher le contenu de l'adresse 27 : vérifier le résultat obtenu
5. Vérifiez le bon fonctionnement du programme avec d'autres données aux adresses 25 et 26 (`a < b`).

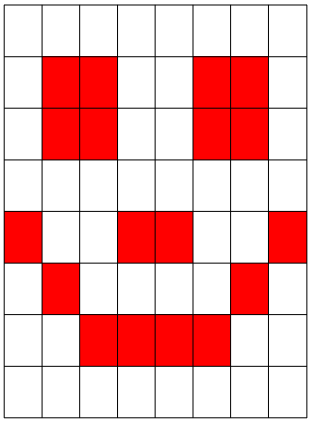
VI. ET POURTERMINER...

Persistance de la vision

On peut exploiter le phénomène de persistance de la vision pour faire du « light-paiting », en « secouant » de bas en haut le Digirule pendant qu'il affiche très rapidement les lignes composant une image les unes à la suite des autres : une image semble se former dans l'espace, *alors qu'une seule ligne n'en est affichée à la fois...*



A vous d'écrire le programme permettant de réaliser le light-painting de l'image ci-contre (ou une autre de votre choix...) :



Des jeux...

Des programmes sont enregistrés dans la mémoire du Digirule, notamment le célèbre « Kill the bit » ; vous pouvez le charger en tenant appuyé le bouton LOAD, et en sélectionnant le bouton D6.