ENTRÉES/SORTIES NUMÉRIQUES POUR ORIC

EXERCICES APPLIQUÉS

Fred_72

Version du 05/09/2024

Sommaire

Introduction....1
Brochage du connecteur E/S....1
Registre de contrôle (#3FD)....2
Contrôle des ports....3
Exercice 1 (Led)....4
  Modification....5
    Question....5
Exercice 2 (Led et BP)....6
  Modification....8
    Question....9
Exercice 3 (Led et LDR)....10
    Question....10
  Modification....11
    Question....11
Exercice 4 (Chenillard à Leds)....12
  Modifications....12
Exercice 5 (7 segments)....13
  Codage des chiffres....14
Exercice 6 (7 segments + 4511)....16
Exercice 7 (7 segments + Clavier)....17
  Codage des lettres....18
  Question....20
Exercice 8 (SR04)....21
Exercice 9 (LCD)....24
  Réglage du contraste....24
Exercice 10 (SERVOMOTEUR)....27
  Version ORIC1....29
Exercice 11 (MATRICE DE LEDS)....30

Introduction

Ce document propose de vous initier à l'utilisation des entrées-sorties numériques avec votre ordinateur ORIC 1 ou ORIC ATMOS.

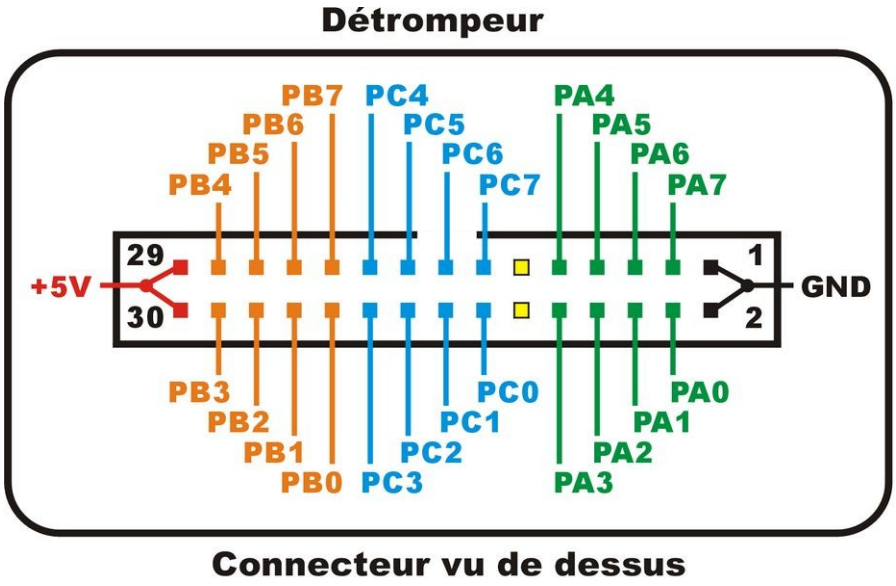
Vous serez guidé au fil des exemples d'application tout d'abord en BASIC puis progressivement en ASSEMBLEUR en commençant par des programmes simples puis de plus en plus compliqués vous permettant d'envisager des applications domotiques pilotées par votre ordinateur ORIC.

Ce document utilise l'extension 24 Entrées/Sorties pour ORIC présentée sur le forum system-cfg. Cette extension ajoute 24 E/S numériques [0V ou 5V] réparties sur 3 ports de 8 bits.

La correction des principaux exercices est dans le fichier « Exercice.dsk » associé.

Brochage du connecteur E/S

La **Figure 1** rappelle le brochage du connecteur IDC30 de cette extension.



1 : GND	2 : GND
3 : PA7	4 : PA0
5 : PA6	6 : PA1
7 : PA5	8 : PA2
9 : PA4	10 : PA3
11 : NC.	12 : NC.
13 : PC7	14 : PC0
15 : PC6	16 : PC1
17 : PC5	18 : PC2
19 : PC4	20 : PC3
21 : PB7	22 : PB0
23 : PB6	24 : PB1
25 : PB5	26 : PB2
27 : PB4	28 : PB3
29 : +5V	30 : +5V

Figure 1: Brochage du connecteur d'E/S

Programmation

L'extension est accessible dans la mémoire principale de l'ORIC en page 3 [#300-#3FF] aux adresses #3FA à #3FD. Le circuit 8255 dispose de 4 registres accessibles aux adresses suivantes :

Adresse (hexadécimal)	Registre
#3FA	PORT A
#3FB	PORT B
#3FC	PORT C
#3FD	CONTROL

Tableau 1: Adresses des registres du circuit 8255

Registre de contrôle (#3FD)

Le registre de contrôle permet de définir le sens de fonctionnement des ports (entrée ou sortie). Il doit être défini au début du programme avant toute action sur les registres des ports.

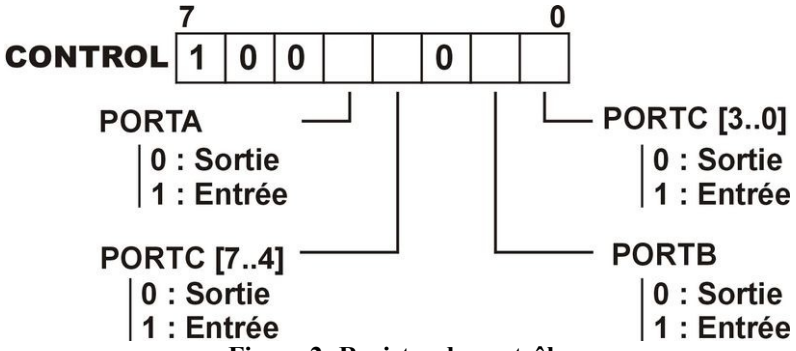


Figure 2: Registre de contrôle

Le port C est coupé en deux blocs de 4 bits, le sens de fonctionnement des 4 bits de poids faibles et des bits de poids forts peut être défini indépendamment. Les autres ports peuvent seulement être tout en entrée ou tout en sortie.

Le **Tableau 2** donne les valeurs du registre de configuration selon la direction des différents ports.

PORT A	PORT B	PORT C [7..4]	PORT C [3..0]	CONFIG (Hexa)	CONFIG (Décimal)
E	E	E	E	9B	155
E	E	E	S	9A	154
E	E	S	E	93	147
E	E	S	S	92	149
E	S	E	E	99	153
E	S	E	S	98	152
E	S	S	E	91	148
E	S	S	S	90	147
S	E	E	E	8B	139
S	E	E	S	8A	138
S	E	S	E	83	131
S	E	S	S	82	130
S	S	E	E	89	137
S	S	E	S	88	136
S	S	S	E	81	129
S	S	S	S	80	128

Tableau 2: Les différentes configurations possibles

Contrôle des ports

Le contrôle des ports d'entrées-sorties se fait à l'aide du code binaire traduisant les états '0' ou '1' de chaque bit de l'octet.

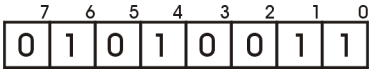


Figure 3: Octet en binaire

Chaque case (8 pour former un octet) correspond à un bit et le numéro placé au dessus indique sa position dans l'octet, c'est à dire sa pondération (2^n). Ainsi PA0 correspond au bit 0 du PORTA et PA7 au bit 7. Pour former une valeur numérique décimale plus compréhensible, il suffit d'ajouter toutes les valeurs des bits à 1 associées à leur pondération.

Valeur= PA0. 2^0 +PA1. 2^1 +PA2. 2^2 +PA3. 2^3 +PA4. 2^4 +PA5. 2^5 +PA6 2^6 +PA7. 2^7 , soit en valeur numérique : Valeur= PA0. 1 +PA1. 2 +PA2. 4 +PA3. 8 +PA4. 16 +PA5. 32 +PA6 64 +PA7. 128
Ce qui donne une plage de variation de 0 (tous les bits à 0) à 255 (tous les bits à 1).

Ainsi la valeur représentée sur la **Figure 3** est : 1.1+1.2+1.16+1.64= 83 en décimal.

Exercice 1 (Led)

But : Faire clignoter deux leds

Les 2 leds sont connectées sur les broches PA0 et PA1 du PORTA selon les branchements de la **Figure 4** (la patte la plus longue de la led est le +). Une résistance de limitation de 220 ohms est câblée en série avec chaque led. Elle est indispensable pour ne pas endommager les leds.

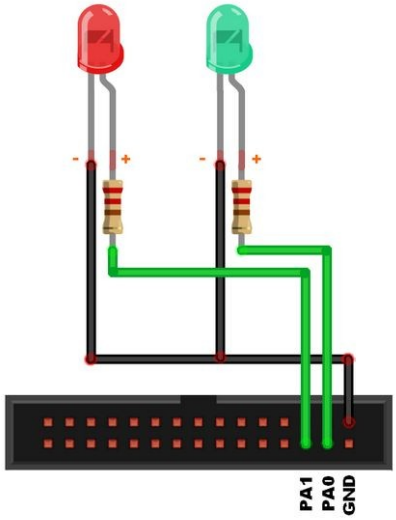


Figure 4: Leds clignotantes

Ce Programme BASIC permet de faire clignoter les 2 leds simultanément.

```
10 CLS
20 PRINT "COMMANDE DES LEDS DU PORT A"
30 POKE #3FD,139
40 POKE #3FA,3
50 GOSUB 100
60 POKE #3FA,0
70 GOSUB 100
80 GOTO 40

100 REM TEMPORISATION
110 FOR I=1 TO 400
120 NEXT I
130 RETURN
```

La ligne 30 place le PORTA en sortie. Les PORTB et PORTC sont laissés en entrée. La valeur 139 est obtenue à l'aide du **Tableau 2**.
La ligne 40 met les bits PA0 et PA1 à 1, les autres (PA2 à PA7) sont à 0. Cela allume les 2 leds.
La ligne 60 met tous les bits du PORTA à 0 et éteint les 2 leds.

Figure 5: Allumage/Extinction des leds

La temporisation appelée par les lignes 50 et 70 permet de ralentir le rythme du clignotement.

Modification

Le programme précédent fait clignoter les leds simultanément. En modifiant les valeurs écrites sur le PORTA (lignes 40 et 60), il est possible de faire clignoter les leds alternativement.

Les nouvelles lignes sont :

```
40 POKE #3FA,1
60 POKE #3FA,2
```

Cela correspond aux nouvelles valeurs binaires suivantes :

Figure 6: Clignotement alterné des LEDs

Question

Comment faut-il modifier le programme pour accélérer ou ralentir le rythme du clignotement ?

Exercice 2 (Led et BP)

But : Commander l'allumage des leds avec un bouton poussoir

Comme l'exercice précédent, les 2 leds sont connectées sur les broches PA0 et PA1 du PORTA selon les branchements de la **Figure 7**. Elles sont accompagnées d'un bouton poussoir et d'une résistance de **4.7 kilo-ohms** connectés sur la broche PB0.

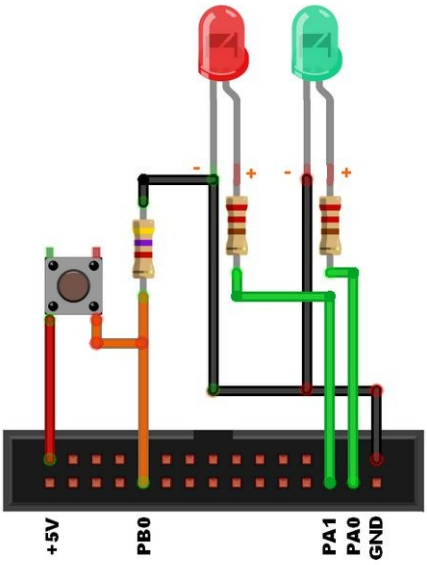


Figure 7: Commande par BP

La résistance de **4.7 kilo-ohms** permet de maintenir un état '0' lorsque le bouton poussoir est relâché. Sans cette résistance, la tension sur la broche PB0 est flottante ce qui se traduit par une lecture aléatoire de '1' et de '0' sur cette entrée.

Ce programme BASIC permet d'allumer la led verte lorsque le bouton poussoir est appuyé et la led rouge lorsqu'il est relâché.

```
10 CLS
20 PRINT "COMMANDE PAR BP DES LEDS DU PORT A"
30 POKE #3FD,139
40 A=2
50 IF (PEEK(#3FB) AND 1) <>0 THEN A=1
60 POKE #3FA,A
70 GOTO 40
```

L'écriture des valeurs vers le PORTA se fait de la même façon que précédemment avec les valeurs 1 et 2 permettant d'allumer la seule led verte ou la seule led rouge.
La lecture fait appel à la commande PEEK(#3FB) permettant au BASIC de récupérer la valeur lue sur le PORTB.
Le bouton poussoir n'est raccordé qu'à la broche PB0, il convient donc de filtrer la valeur lue pour n'avoir que l'information relative à l'état logique du bouton poussoir (bit 0).

Pour cela le programme utilise l'opérateur AND (ET logique). Cet opérateur binaire agit de la façon suivante (A et B sont les valeurs d'entrées, S est la sortie) :

S=A AND B

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Tableau 3: Opérateur AND

Pour obtenir un 1 logique, il faut que les 2 entrées A et B soient à 1.

Dans le programme, la même opération est effectuée pour chaque bit de l'octet. Cela conduit à :

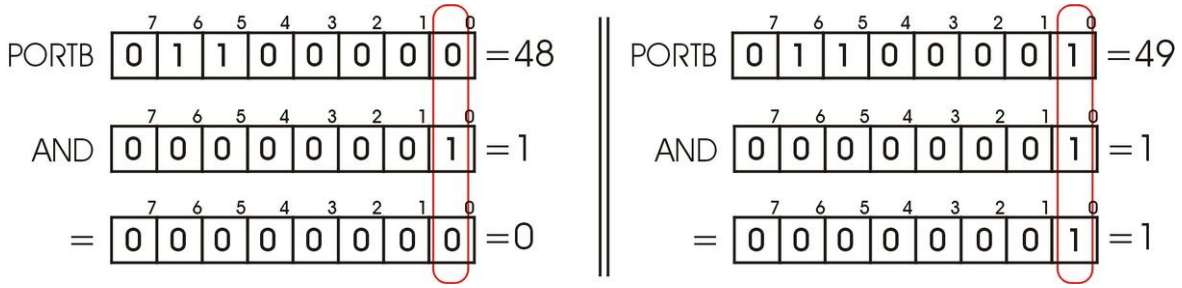


Figure 8: Filtrage avec un AND

Sur cette figure, les bits 5 et 6 sont à 1 traduisant des états indésirables ou issus d'autres boutons poussoirs. Le masque associé à l'opérateur AND vaut 1 ce qui permet de ne sélectionner que le bit PB0 du PORTB. Il en ressort une valeur numérique qui vaut soit 0 (BP relâché) soit 1 (BP appuyé).

Le programme fait cette opération en ligne 50 puis modifie ou non la valeur contenue dans A en fonction de l'état logique du bouton poussoir. Il ne reste plus qu'à écrire la valeur A sur le PORTA pour allumer la led correspondante.

Modification

On ajoute un 2ème bouton poussoir connecté à PB1.

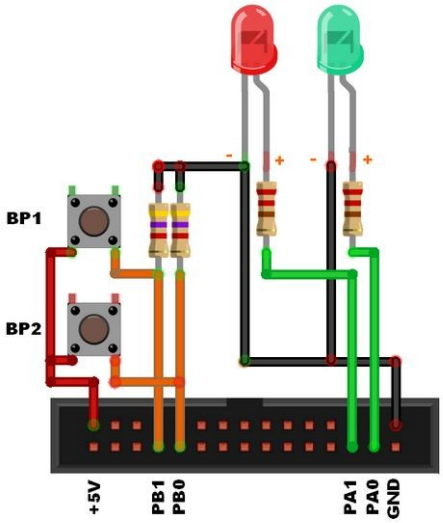


Figure 9: Commande par 2 BPs

Ce nouveau programme BASIC permet d'allumer la led rouge lorsque le bouton poussoir BP1 est appuyé et la led verte lorsque le bouton poussoir BP2 est appuyé. Enfin, les 2 leds sont allumées si les 2 boutons poussoirs sont appuyés simultanément.

```
10 CLS
20 PRINT "COMMANDE PAR 2 BPS DES LEDS DU PORT A"
30 POKE #3FD,139
40 A=0
50 IF (PEEK(#3FB) AND 1) <>0 THEN A=1
60 B=0
70 IF (PEEK(#3FB) AND 2) <>0 THEN B=2
80 POKE #3FA, A OR B
90 GOTO 40
```

La lecture fait toujours appel à la routine READ suivie de la commande PEEK associée à l'opérateur AND, mais cette fois les valeurs 1 et 2 sont utilisées pour tester les 2 boutons poussoirs.

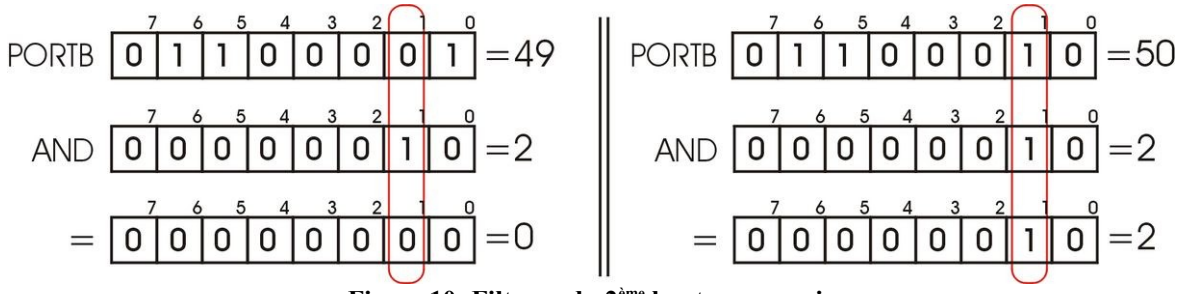


Figure 10: Filtrage du 2ème bouton poussoir

Pour BP2, le résultat de l'opération vaut 0 ou 2. C'est la raison pour laquelle le programme utilise le test <> 0 plutôt qu'un test d'égalité.

Le programme utilise 2 variables A et B pour avoir 2 valeurs de commande du PORTA. La modification du port se fait par une écriture unique qui va combiner les 2 variables pour former une seule valeur finale à l'aide de l'opérateur OR (OU logique).

Cet opérateur binaire agit de la façon suivante (A et B sont les valeurs d'entrées, S est la sortie) :

S=A OR B

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Tableau 4: Opérateur OR

Pour obtenir un 1 logique, il suffit qu'une des 2 entrées (A ou B) soit à 1.

Dans le programme, cette opération est utilisée avec les variables A et B. Cela conduit à :

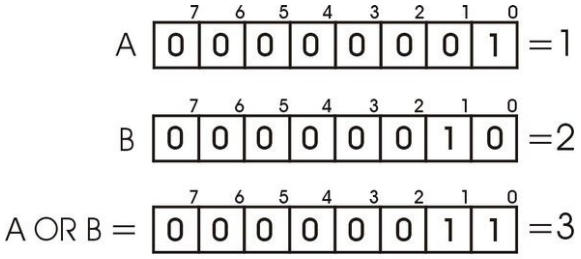


Figure 11: Association des variables A et B du programme

Remarque : Il est possible d'utiliser l'opérateur addition (+) pour arriver au même résultat dans le cas de variables distinctes (c'est à dire qui n'utilise pas les mêmes bits). C'est le cas ici car A utilise le bit 0 tandis que B utilise le bit 1 (c'est pour cela que dans le programme il est écrit B=2).

Question

Comment faut-il modifier le programme si BP2 est connecté sur la broche PB4 ?

Exercice 3 (Led et LDR)

But : Commander l'allumage de la led selon la luminosité

La led est associée à une photorésistance (LDR). La photorésistance est un composant dont la résistance varie selon la lumière. Une résistance de 15 kilo-ohms forme un diviseur de tension avec la photorésistance afin d'obtenir une tension dépendant de la luminosité. Si le composant est exposé à une lumière vive, sa résistance est faible et la tension en PB1 est élevée (proche de 5V). Si le composant est maintenu dans l'obscurité, sa résistance est grande et la tension en PB1 est faible (proche de 0V). L'entrée PB1 ne peut pas directement mesurer une tension (il faudrait un convertisseur analogique/numérique pour cela) mais elle réagit à partir d'un certain niveau (environ 2V) au dessus duquel elle voit un niveau '1' sinon elle voit un niveau '0'.

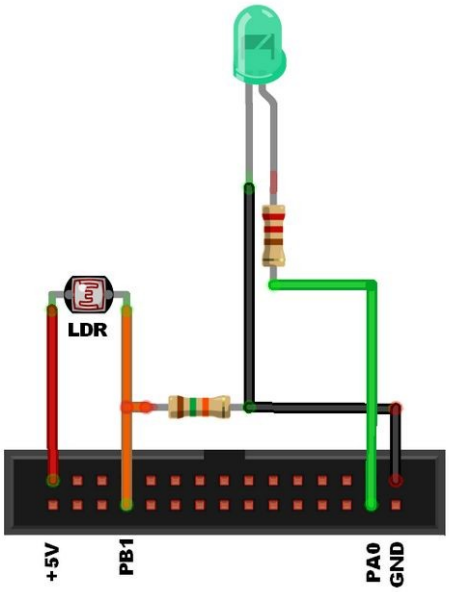


Figure 12: Photorésistance et Led

Question

Le but est ici de créer un interrupteur crépusculaire. Le jour (lorsque le capteur est éclairé), la led est éteinte et la nuit (lorsque le capteur est dans l'obscurité), la led est allumée. Comment faut-il modifier le programme de l'exercice précédent pour obtenir le fonctionnement désiré ?

Modification

On ajoute à présent un détecteur de mouvement (PIR) sur l'entrée PB0.

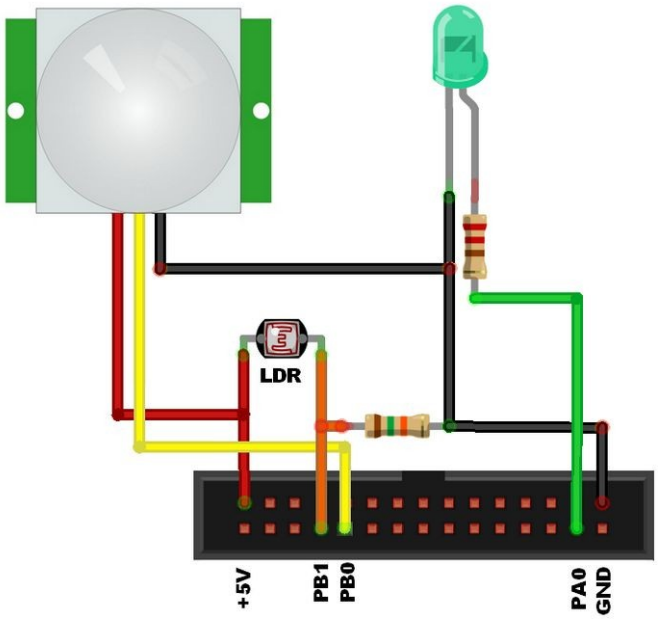


Figure 13: Interrupteur crépusculaire 2

Question

Le but est maintenant de créer un interrupteur crépusculaire à détection de mouvement. Le jour, lorsque le capteur est éclairé, la led est éteinte et la nuit, lorsque le capteur est dans l'obscurité et qu'un mouvement est détecté, la led s'allume pendant 10 secondes. Comment faut-il modifier le programme pour obtenir le fonctionnement désiré ?

Exercice 4 (Chenillard à Leds)

But : Faire un chenillard avec 8 leds

Les 8 leds sont connectées sur les broches PA0 à PA7 du PORTA selon les branchements de la **Figure 14**. Une seule led est allumée à la fois dans une séquence de défilement de la droite vers la gauche ou de la gauche vers la droite.

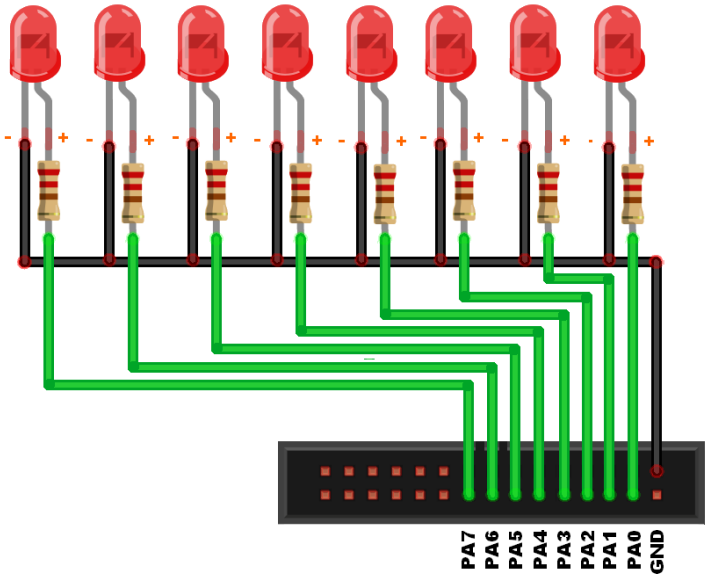


Figure 14: Chenillard à 8 leds

Ce programme BASIC permet de faire défiler une led allumée parmi les 8 en la faisant glisser de la droite vers la gauche au rythme de la temporisation.

```
10 CLS
20 PRINT "CHENILLARD SUR LE PORT A"
30 POKE #3FD,139
40 A=1
50 FOR K=1 TO 8
60 POKE #3FA, A
70 GOSUB 200
80 A=A*2
90 NEXT K
95 GOTO 40

200 REM TEMPORISATION
210 FOR I=1 TO 200
220 NEXT I
230 RETURN
```

Modifications

Comment faut-il modifier le programme pour obtenir un défilement de la gauche vers la droite ?

Comment faut-il modifier le programme pour obtenir un défilement dans les 2 sens (style K2000) ?

Exercice 5 (7 segments)

But : Commander un afficheur 7 segments

Cet exercice fait appel à un afficheur 7 segments pour afficher les chiffres de 0 à 9. Dans cet exercice l'afficheur utilisé est un HS-5161A à cathodes communes. Il est bien sûr possible d'utiliser n'importe quel afficheur à cathodes communes mais il faudra adapter le câblage selon son brochage.

Le terme cathodes communes signifie que la cathode (le -) des leds constituant les 7 segments de l'afficheur sont raccordées ensemble à l'intérieur de l'afficheur. Cela permet de réduire le nombre de broches nécessaires à seulement 9 pour 7 segments + le point décimal.

Pour allumer un segment, il faut mettre à '1' (+5V) la broche concernée. Comme pour les exercices précédents, le courant dans les leds doit être limité par une résistance de **470 ohms** en série avec chaque led.

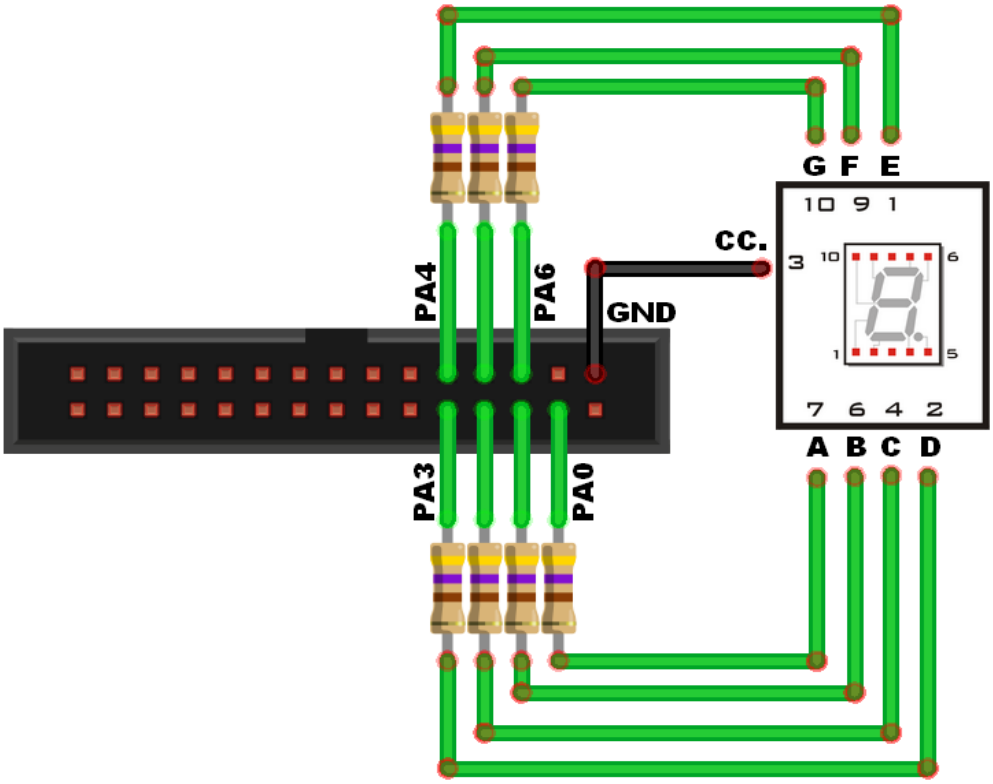
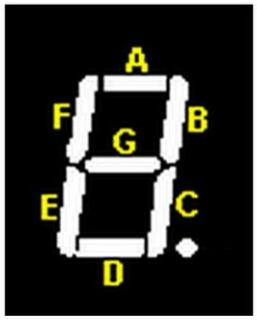


Figure 15: Commande d'un afficheur 7 segments

Codage des chiffres

L'afficheur est constitué de 7 segments (le point décimal n'est pas utilisé) placés judicieusement pour former un chiffre. Les segments sont nommés 'A' à 'G' selon la disposition de la figure.

Pour avoir un affichage cohérent, il faut coder l'information numérique (0 à 9) en une valeur permettant d'allumer les segments pour former un chiffre sur l'afficheur. Ce codage est réalisé par le programme à l'aide d'un tableau de 10 cases contenant les valeurs de commande de l'afficheur. Il suffit d'indexer le tableau avec la valeur numérique pour obtenir la valeur de commande de l'afficheur. Le **Tableau 5** donne le codage des chiffres de 0 à 9.



Chiffre	G	F	E	D	C	B	A	Valeur décimale
0	0	1	1	1	1	1	1	63
1	0	0	0	0	1	1	0	6
2	1	0	1	1	0	1	1	91
3	1	0	0	1	1	1	1	79
4	1	1	0	0	1	1	0	102
5	1	1	0	1	1	0	1	109
6	1	1	1	1	1	0	1	125
7	0	0	0	0	1	1	1	7
8	1	1	1	1	1	1	1	127
9	1	1	0	1	1	1	1	111

Tableau 5: Codage des chiffres sur un afficheur 7 segments

Ce Programme BASIC permet de compter de 0 à 9 sur l'afficheur.

```
10 CLS
15 GOSUB 1000
20 PRINT "COMPTAGE SUR UN AFFICHEUR"
30 POKE #3FD,139
40 FOR N=0 TO 9
50 POKE #3FA, A (N)
60 GOSUB 100
70 NEXT N
80 GOTO 40
```

```
100 REM TEMPORISATION
110 FOR I=1 TO 400
120 NEXT I
130 RETURN

1000 REM CHARGEMENT DES CODES 7 SEGMENTS
1010 DIM A(10)
1020 FOR I=0 TO 9
1030 READ A(I)
1040 NEXT I
1050 RETURN
1060 DATA 63,6,91,79,102,109,125,7,127,111
```

La 1ère partie du programme appelle le sous-programme placé à partir de la ligne 1000 pour remplir les cases du tableau A avec les 10 valeurs permettant de visualiser des chiffres compréhensibles sur l'afficheur 7 segments (valeurs placées dans la ligne DATA en 1060).

Les lignes 40 à 70 mettent en place une boucle de comptage FOR-NEXT durant laquelle N parcourt les valeurs 0 à 9.

La ligne 50 envoie sur le PORTA la valeur correspondant au chiffre N en utilisant le tableau A pour transformer cette valeur numérique en code '7 segments'.

Le sous-programme placé à partir de la ligne 100 est une temporisation permettant de ralentir le programme.

Exercice 6 (7 segments + 4511)

But : Commander un afficheur 7 segments avec un CI décodeur

Cet exercice fait toujours appel à un afficheur 7 segments pour afficher les chiffres de 0 à 9 mais cette fois utilise un circuit intégré décodeur HEF4511. L'utilisation de ce circuit permet de n'utiliser que 4 bits de commande et permet de se passer du programme de codage.

Il suffit d'envoyer la valeur numérique sur la partie basse du port (PA[0..3]) pour obtenir l'affichage du chiffre souhaité. Toutefois le CI décodeur ne peut afficher que les chiffres de 0 à 9.

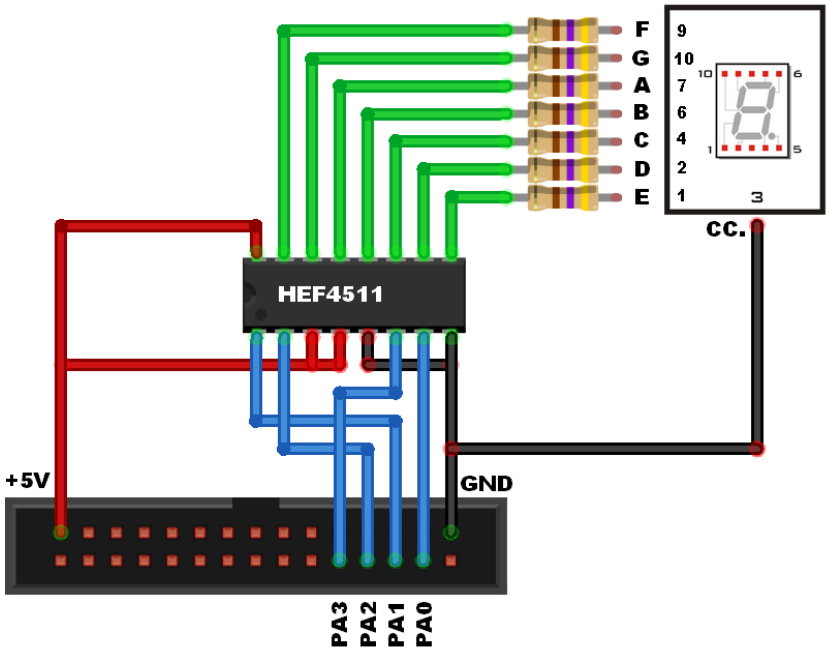


Figure 16: Commande d'un afficheur 7 segments avec un CI décodeur

Ce Programme BASIC permet de compter de 0 à 9 sur l'afficheur.

```
10 CLS
20 PRINT "COMPTAGE SUR UN AFFICHEUR"
30 POKE #3FD,139
40 FOR N=0 TO 9
50 POKE #3FA, N
60 GOSUB 100
70 NEXT N
80 GOTO 40

100 REM TEMPORISATION
110 FOR I=1 TO 400
120 NEXT I
130 RETURN
```

Exercice 7 (7 segments + Clavier)

But : Lire un clavier à matrice

Cet exercice fait appel à un afficheur 7 segments et un clavier à 16 touches. Le but de l'exercice est de visualiser la touche appuyée sur l'afficheur 7 segments.

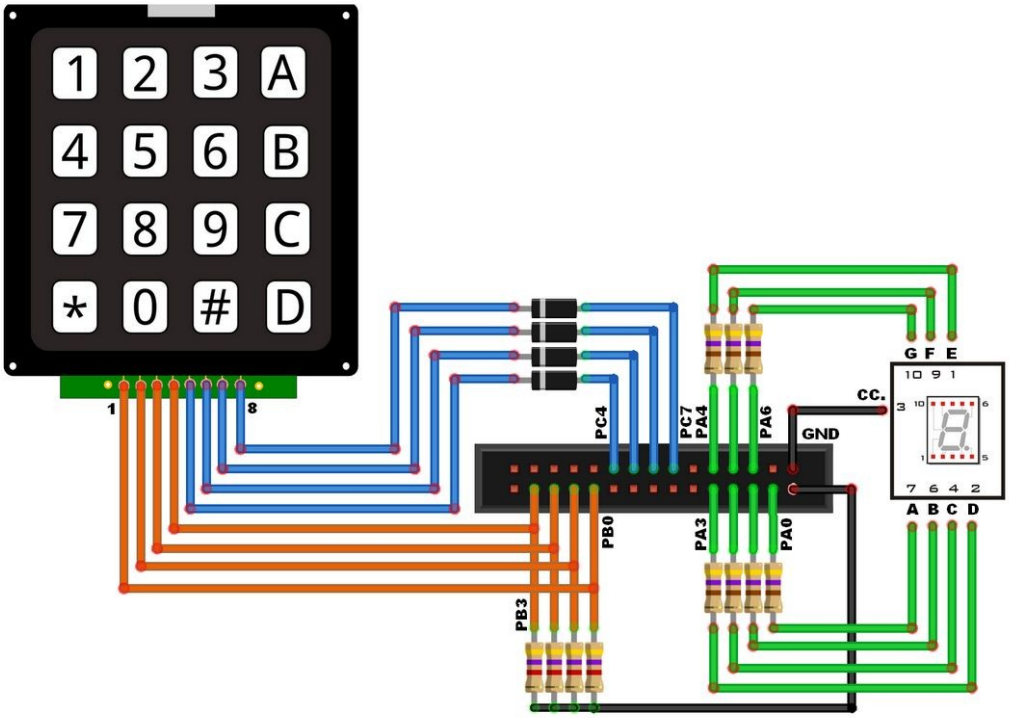


Figure 17: Clavier et afficheur 7 segments

Clavier

Le clavier est constitué de 16 touches réparties en 4 colonnes de 4 rangées (soit 4x4 touches). Cette façon de faire économise des fils mais nécessite une lecture active du clavier.

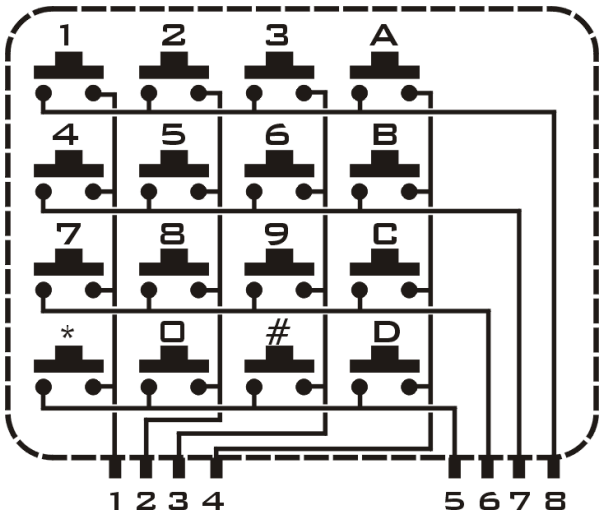


Figure 18: Matrice du clavier 16 touches

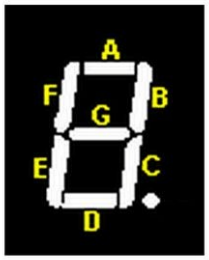
Pour lire le clavier, il faut disposer de 4 entrées et 4 sorties. Le principe consiste à alimenter une seule rangée à la fois puis à lire les 4 colonnes. La lecture du clavier nécessite donc 4 cycles. Il reste ensuite à identifier la touche pressée.

Sur le schéma, il y a des résistances de tirage à ‘0’ (4.7kilo-ohms) et des diodes (1N4148). Les résistances permettent de maintenir un état ‘0’ en l’absence d’action sur les touches du clavier (même principe que pour un bouton poussoir). Les diodes permettent de séparer les sorties. Si 2 touches sont appuyées simultanément, 2 sorties peuvent être en court-circuit (l’une est à ‘1’ et l’autre à ‘0’). Les diodes ne laissant passer le courant que dans un seul sens interdisent ce cas de figure. Elles ne sont pas nécessaires si l’utilisateur appuie sur une seule touche à la fois.

Sur le clavier, les caractères # et * sont remplacés par les lettres ‘E’ et ‘F’.

Codage des lettres

Les chiffres ont déjà été codés dans **l’exercice 5** mais il reste les lettres ‘A’ à ‘F’. Le **Tableau 6** donne les valeurs associées à ces nouveaux caractères.



Caractère	G	F	E	D	C	B	A	Valeur décimale
A	1	1	1	0	1	1	1	119
B	1	1	1	1	1	0	0	124
C	0	1	1	1	0	0	1	57
D	1	0	1	1	1	1	0	94
E	1	1	1	1	0	0	1	121
F	1	1	1	0	0	0	1	113

Tableau 6: Codage des caractères supplémentaires

Ce programme BASIC permet de lire le clavier et d'afficher le caractère de la touche pressée sur l'afficheur. Il est supposé que l'utilisateur appuie sur une seule touche à la fois.

```
10 CLS
15 GOSUB 1000
20 PRINT "LECTURE CLAVIER 16 TOUCHES"
30 POKE #3FD,131

100 REM PROGRAMME PRINCIPAL
110 GOSUB 200
115 PRINT @2,4;KB;"  "
120 IF KB<>255 THEN POKE #3FA,A(KB)
130 IF KB=255 THEN POKE #3FA,0
140 GOTO 110

200 REM LECTURE CLAVIER
210 POKE #3FC,16
220 R1=PEEK(#3FB) AND 15
230 POKE #3FC,32
240 R2=PEEK(#3FB) AND 15
250 POKE #3FC,64
260 R3=PEEK(#3FB) AND 15
270 POKE #3FC,128
280 R4=PEEK(#3FB) AND 15
290 B=0
300 IF R1+R2+R3+R4=0 THEN KB=255:GOTO 350
310 IF R1>0 THEN B=R1:GOSUB 400: KB=C1(B)
320 IF R2>0 THEN B=R2:GOSUB 400: KB=C2(B)
330 IF R3>0 THEN B=R3:GOSUB 400: KB=C3(B)
340 IF R4>0 THEN B=R4:GOSUB 400: KB=C4(B)
350 RETURN

400 REM CONVERSION
410 IF B=4 THEN B=3
420 IF B=8 THEN B=4
430 RETURN

1000 REM VARIABLES POUR LE CLAVIER
1010 DIM A(16),C1(5),C2(5),C3(5),C4(5)
1020 FOR V=0 TO 15
1030 READ A(V)
1040 NEXT V
1050 FOR V=1 TO 4
1060 READ C1(V),C2(V),C3(V),C4(V)
1070 NEXT V
1080 DATA 63,6,91,79,102,109,125,7,127,111,119,124,57,94,121,113
1090 DATA 1,4,7,14,2,5,8,15,3,6,9,0,10,11,12,13
1100 RETURN
```

Le sous programme à partir de la ligne 1000 initialise le tableau A avec les codes '7 segments' des caractères et les tableaux C1 à C4 avec les valeurs correspondant aux 4 colonnes du clavier.

Le sous-programme placé à partir de la ligne 200 permet de lire le clavier. Pour cela, chaque rangée est mise à '1' à tour de rôle puis les 4 bits correspondants aux 4 rangées sont lus et stockés dans les

variables R1 à R4. Les valeurs lues sont filtrées par un opérateur AND afin de ne garder que les 4 bits de poids faible.

Si aucune touche n'est appuyée, les variables R1 à R4 contiennent toutes la valeur 0.

Selon la touche appuyée les valeurs dans les variables R1 à R4 peuvent prendre les valeurs 1, 2, 4 ou 8. Cette valeur est ramenée à 1, 2, 3 ou 4 par le petit sous-programme commençant à la ligne 400 afin d'aligner cette valeur sur les cases des tableaux C1 à C4.

Question

Comment faut-il modifier le programme pour s'assurer que l'utilisateur n'appuie que sur une seule touche à la fois ?

Exercice 8 (SR04)

But : Mesurer la distance avec un radar à ultrasons SR04

Cet exercice permet de mesurer la distance d'un objet à l'aide d'un module SR04.

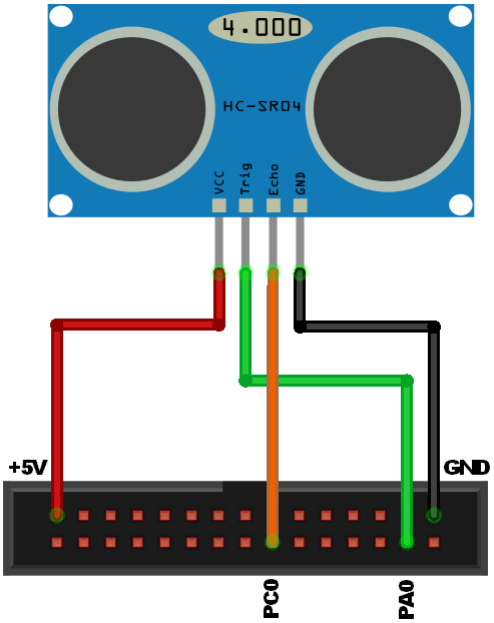


Figure 19: Mesure de distance

Principe de la mesure

A la réception de l'impulsion de commande TRIG, le module émet une salve d'impulsions sonores (ultrasons à 40kHz). L'onde sonore va se réfléchir sur l'obstacle puis revenir vers le module. La mesure du temps entre l'émission et la réception permet de déduire la distance séparant le module de l'obstacle. Le module ne retourne pas directement la distance mais donne une impulsion dont la durée correspond au temps de vol (c'est à dire le temps nécessaire pour faire l'aller-retour).

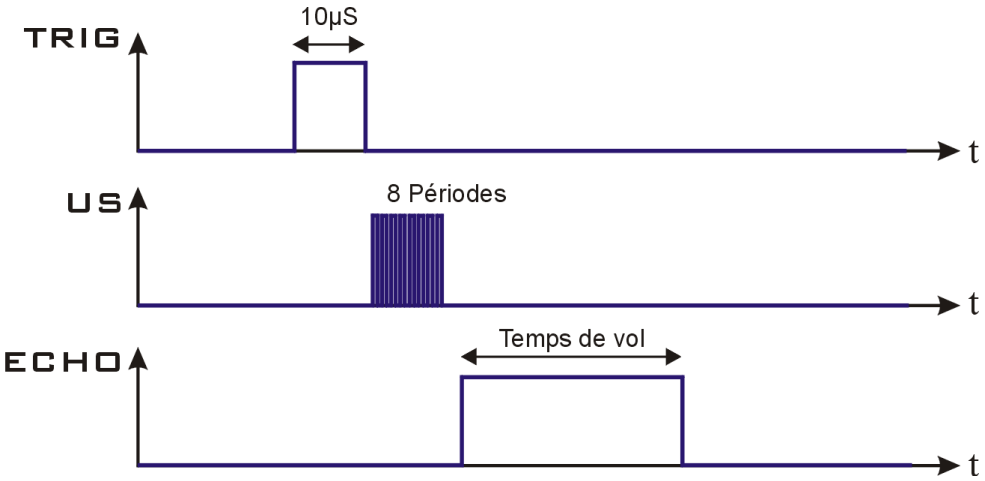


Figure 20: SR04 - Principe de la mesure

La commande et la mesure nécessitent des temps très brefs, donc cette partie est assurée par un programme assembleur appelé par le programme BASIC à l'aide de la commande CALL.

```
5 HIMEM #7FFF
10 CLS
20 FOR A=#8000 TO #8050
30 READ V
40 POKE A,V
50 NEXT A
60 DATA 169,139,141,253,3,120,169,1,141,250,3,169,0,169,0,169,0,169,0,169
70 DATA 0,141,250,3,162,0,232,240,50,173,252,3,41,1,240,246,169,0,141
80 DATA 0,129,141,1,129,141,2,129,173,0,129,24,105,1,141,0,129
90 DATA 173,1,129,105,0,141,1,129,173,2,129,105,0,141,2,129
95 DATA 173,252,3,41,1,208,224,88,96

100 PRINT "CONTROLE D'UN SONAR SR04"
110 PRINT "AVEC LA CARTE 24 ENTREES SORTIES"
115 PRINT
120 PRINT "PLACEZ VOTRE MAIN AU DESSUS DU RADAR"
130 CALL #8000
150 D=DEEK(#8100)
160 PRINT @8,8;"DISTANCE= ",D
170 GOTO 130
```

Le code assembleur contenu dans les lignes DATA est le suivant :

```
        .ORG $8000

; PORTA en sortie, PORTC en entrée
        LDA #$8B
        STA $3FD
        SEI

; envoie une impulsion sur le trigger (13µs)
        LDA #1
        STA $3FA
        LDA #0
        LDA #0
        LDA #0
        LDA #0
        LDA #0
        STA $3FA
        LDX #0
; attente du signal à l'état haut
WHA
        INX
        BEQ CDST2    ; Timeout

        LDA $3FC
        AND #1
        BEQ WHA

        LDA #0
        STA CMPT
        STA CMPT+1
        STA CMPT+2

; compte le temps
CDST
        LDA CMPT
        CLC
        ADC #1
        STA CMPT
        LDA CMPT+1
        ADC #0
        STA CMPT+1
        LDA CMPT+2
        ADC #0
        STA CMPT+2

; attente du signal à l'état bas
CDST1
        LDA $3FC
        AND #1
        BNE CDST
CDST2
        CLI
        RTS

        .ORG $8100
CMPT
        DB 0,0,0
```

Le programme commence par réserver la mémoire puis met en place le programme assembleur. Le programme principal situé à partir de la ligne 100 appelle le programme de mesure et affiche le résultat sur l'écran.

La mesure commence par l'envoi d'une impulsion sur la broche TRIG. Le programme attend ensuite que la sortie ECHO passe à '1'. Un compteur de timeout permet d'éviter de bloquer le programme si le module ne répond pas. Ensuite un compteur 24 bits compte le temps puis stocke le résultat en mémoire.

Les interruptions sont désactivées pendant la mesure afin de ne pas fausser le comptage. C'est le rôle des instructions SEI (interdit les interruptions) et CLI (autorise les interruptions) placées au début et à la fin du programme.

Exercice 9 (LCD)

But : Piloter un afficheur LCD

Cet exercice présente la méthode pour contrôler un afficheur LCD (2 lignes de 16 caractères). L'afficheur est connecté au PORTA. Le PORTC permet de piloter les signaux de contrôle de l'afficheur. Une résistance de 100 ohms est placée en série avec l'alimentation du rétroéclairage (broches 15 et 16) et une résistance ajustable de 5 kilo-ohms assure le réglage du contraste.

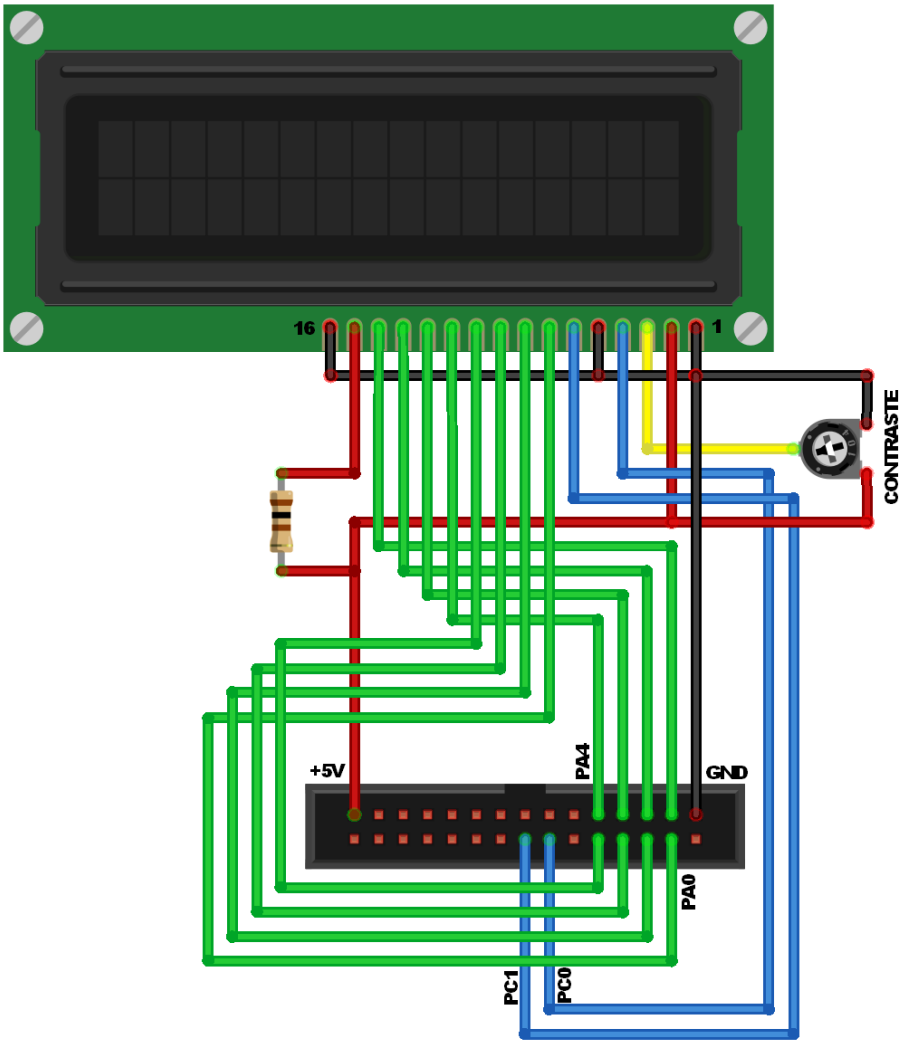


Figure 21: Contrôle d'un afficheur LCD

Réglage du contraste

A la mise sous tension, la ligne supérieure de l'afficheur doit être intégralement allumée tandis que la ligne inférieure doit être éteinte. Si rien n'est visible, il faut ajuster le contraste pour la faire apparaître en tournant la vis de réglage de la résistance variable. Après la séquence d'initialisation, cette ligne foncée disparaîtra et la luminosité sera homogène sur tout l'afficheur.

Les signaux de contrôle permettant de piloter l'afficheur sont : RS, R/W et E. Dans cette application, R/W est maintenu à 0 (écriture uniquement).

RS (PC0) permet de sélectionner le registre d'instructions (RS=0) ou le transfert des données vers l'affichage (RS=1).

E (PC1) est le signal d'activation. Les données envoyées sont validées par une impulsion sur cette broche.

Le programme est composé d'une partie en BASIC et d'une partie en assembleur s'occupant du transfert des commandes.

```
5 HIMEM #7FFF
10 CLS
20 PRINT "CONTROLE D'UN AFFICHEUR LCD"
30 GOSUB 1000

100 REM PROGRAMME PRINCIPAL
105 GOSUB 300
110 A$="BONJOUR LE MONDE"
120 B$="ORIC"
125 REM AFFICHAGE LIGNE1
130 POKE #8001,1
135 CALL #8000
140 T$=A$
150 GOSUB 200
155 REM AFFICHAGE LIGNE2
160 POKE #8001,192
165 CALL #8000
170 T$=B$
180 GOSUB 200
190 END

200 REM ENVOI D'UNE CHAINE DE CARACTERES
210 FOR I=1 TO LEN (T$)
220 POKE #8001,ASC(MID$(T$,I,1))
230 CALL #8023
240 NEXT I
250 RETURN

300 REM INIT AFFICHEUR
310 POKE #3FD,138
320 POKE #8001,56
330 CALL #8000
340 POKE #8001,12
350 CALL #8000
360 RETURN

1000 REM ROUTINES LM
1010 FOR I=#8000 TO #802D
1020 READ N
1030 POKE I,N
1040 NEXT I
1050 RETURN
1060 DATA 169,0,141,250,3,169,0,141,252,3,9,2,141,252,3,169,0,141,252,3
1070 DATA 162,255,234,234,202,208,251,162,255,234,234,202,208,251,96
1080 DATA 173,1,128,141,250,3,169,1,76,7,128
```

Le code assembleur contenu dans les lignes DATA est le suivant :

```
            .ORG $8000

; Envoi d'une commande
Send_Com
            LDA #0    ; valeur
            STA $3FA
            LDA #0

Continue
            STA $3FA
            ORA #2
            STA $3FA
            LDA #0
            STA $3FA

            LDX #$FF
Tempo
            NOP
            NOP
            DEX
            BNE Tempo

            LDX #$FF
Tempo1
            NOP
            NOP
            DEX
            BNE Tempo1
            RTS


; Envoi d'un caractère
Send_DATA                   ; $8023
            LDA $8001
            STA $3FA
            LDA #1
            JMP Continue
```

Le programme principal commence à la ligne 100. Il permet d'afficher un message sur les 2 lignes de l'afficheur.

Le sous-programme placé à partir de la ligne 200 assure l'envoi caractère par caractère de la chaîne de caractères T$ par le biais de la routine Send_DATA.

Le sous-programme placé à partir de la ligne 300 assure l'initialisation de l'afficheur en le plaçant en mode 8 bits, 2 lignes, affichage ON à l'aide de la routine Send_COM.

Les lignes 130-135 permettent d'effacer l'écran tandis que les lignes 160-165 placent le curseur sur la 2ème ligne.

Exercice 10 (SERVOMOTEUR)

But : Piloter un servomoteur

Cet exercice présente la méthode pour piloter un servomoteur en faisant varier sa position. La sortie 5V de la carte 24 E/S ne peut pas fournir assez de courant pour alimenter le servomoteur qui devra être alimenté en 5V par une source externe (chargeur USB par exemple).

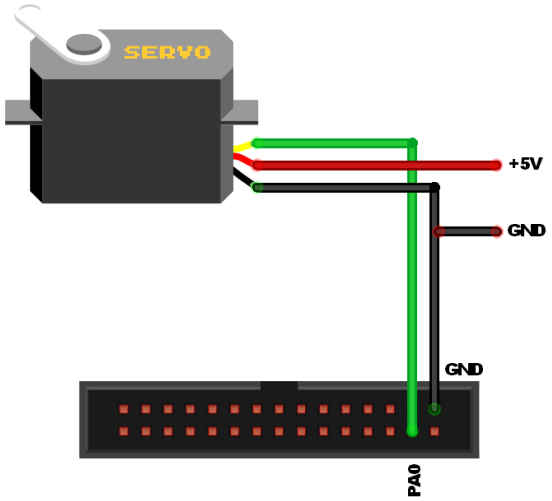


Figure 22: Commande d'un servomoteur

Le servomoteur est un dispositif électromécanique dont l'axe de sortie peut prendre n'importe quelle position sur un demi cercle (soit une variation totale de 180°). La commande de la position du servomoteur se fait à l'aide d'une impulsion de durée calibrée répétée au rythme de 50 fois par seconde (20ms). La durée de l'impulsion va de 1ms à 2ms. 1ms correspond à un angle de 0° tandis que 2ms correspond à un angle de 180°. La variation est linéaire sur toute la plage d'utilisation.

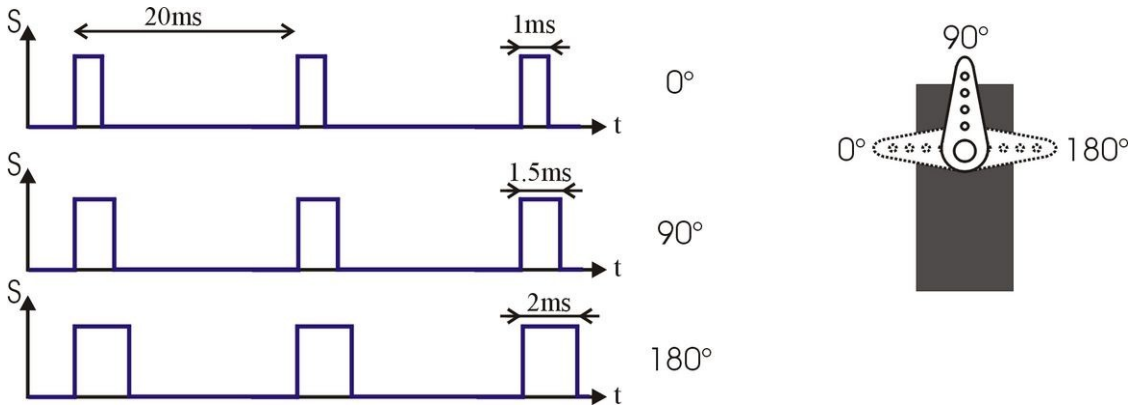


Figure 23: Commande d'un servomoteur analogique

La commande du servomoteur nécessite des impulsions courtes et précises à intervalle régulier. Le programme va donc utiliser les interruptions déclenchées toutes les 10ms par le VIA de l'Oric. Le vecteur d'interruption est situé dans la mémoire RAM aux adresses $245-246 pour l'ATMOS et $238-239 pour l'ORIC 1. Il sera détourné vers le programme de commande du servomoteur. Ainsi le programme sera exécuté en tâche de fond et le programme BASIC va simplement mettre à jour la

valeur de commande située à l'adresse #800D. Ce programme ne fonctionne que sur ORIC ATMOS. La version ORIC 1 est donnée sur la page suivante.

```
1 REM VERSION ORIC ATMOS
5 HIMEM #7FFF
10 CLS
15 GOSUB 1000
20 PRINT "CONTROLE D'UN SERVOMOTEUR"
30 POKE #3FD,139
40 CALL #8000

100 REM PROGRAMME PRINCIPAL
110 INPUT "VALEUR [0-255] :";V
120 POKE #800D,V
130 GOTO 110

1000 REM ROUTINES LM
1010 FOR I=#8000 TO #803A
1020 READ N
1030 POKE I,N
1040 NEXT I
1050 RETURN
1060 DATA 120,169,128,141,70,2,169,14,141,69,2,88,96,0,72,8,138,72,238
1070 DATA 58,128,173,58,128,41,1,240,18,169,1,141,250,3,162,100,202
1080 DATA 208,253,174,13,128,232, 202,234,208,252,169,0,141,250,3,104
1090 DATA 170,40,104,76,34,238,0
```

Le code assembleur contenu dans les lignes DATA est le suivant :

```
            .ORG $8000

; Détourne les interruptions
Set_INT
            SEI
            LDA # NewINT/256
            STA $246
            LDA # NewINT%256
            STA $245
            CLI
            RTS

VAL
            .BYTE 0

NewINT
            PHA
            PHP
            TXA
            PHA

; Pour exécuter le pgm 1/2
            INC FP
            LDA FP
            AND #1
            BEQ FIN

; PA0 à 1   (début de l'impulsion)
            LDA #1
            STA $3FA
            LDX #100        ; valeur#1ms
PTEMPO
            DEX
            BNE PTEMPO

            LDX  VAL        ; valeur angle
            INX
TEMPO
            NOP
            DEX
            BNE TEMPO

; PA0 à 0   (fin de l'impulsion)
FIN
            LDA #0
            STA $3FA
            PLA
            TAX
            PLP
            PLA

; saut routine INT
            JMP $EE22               ; ATMOS
FP
            .BYTE 0
```

La 1ère partie du programme assembleur détourne les interruptions vers la nouvelle routine permettant de générer l'impulsion de commande du servomoteur. Une fois en place, l'ORIC exécute cette routine à chaque appel d'interruption.

La routine commence par sauvegarder les différents registres A, P et X puis incrémente une variable FP. A chaque incrémentation, le bit 0 de FP change d'état, cela permet d'exécuter le programme générateur une fois sur deux, soit toutes les 20ms. La génération de l'impulsion commence par mettre à '1' la sortie PA0. Ensuite une 1ère temporisation de 1ms assure la partie fixe de l'impulsion puis une 2ème temporisation prend en compte la valeur de consigne VAL pour augmenter la durée de l'impulsion jusqu'à 2ms. Enfin la sortie PA0 est remise à '0' puis le programme continue avec la routine initiale de gestion des interruptions.

Remarque : Selon le modèle de servomoteur, il faut ajuster la plage de variation de la largeur de l'impulsion. Le programme est adapté à un servomoteur miniature de type SG90 (0.5 à 2ms).

Version ORIC1

Pour l'Oric 1, le programme précédent doit être modifié car le vecteur d'interruption est différent. Ces modifications ne concernent que la partie assembleur du programme.

Le code assembleur pour la version ORIC1 est le suivant :

```
            .ORG $8000

; Détourne les interruptions
Set_INT
            SEI
            LDA # NewINT/256
            STA $22A
            LDA # NewINT%256
            STA $229
            CLI
            RTS

VAL
            .byte 0

NewINT
            PHA
            PHP
            TXA
            PHA

; PA0 à 1    (début de l'impulsion)
            LDA #1
            STA $3FA

            LDX #197        ; valeur=1ms
PTEMPO
            DEX
            BNE PTEMPO

            LDX  VAL        ; valeur angle
            INX
TEMPO
            NOP
            DEX
            BNE TEMPO

; PA0 à 0    (fin de l'impulsion)
            LDA #0
            STA $3FA

            PLA
            TAX
            PLP
            PLA

; saut routine INT
            JMP $EC03       ; ORIC 1
```

Cela donne les lignes DATA suivantes :

```
1 REM VERSION ORIC 1
.....
1060 DATA 120,169,128,141,42,2,169,14,141,40,2,88,96,0,72,8,138,72,169,1
1070 DATA 141,250,3,162,197,202,208,253,174,13,128,232
1080 DATA 202,234,208,252,169,0,141,250,3,104,170,40,104,76,3,236
```

Exercice 11 (MATRICE DE LEDS)

But : Piloter une matrice de leds

Cet exercice présente la méthode pour piloter une matrice de 8x8 leds équipée du circuit MAX7219. Ce module consomme beaucoup de courant, il faut donc faire appel à une source d'alimentation externe (un chargeur USB par exemple).

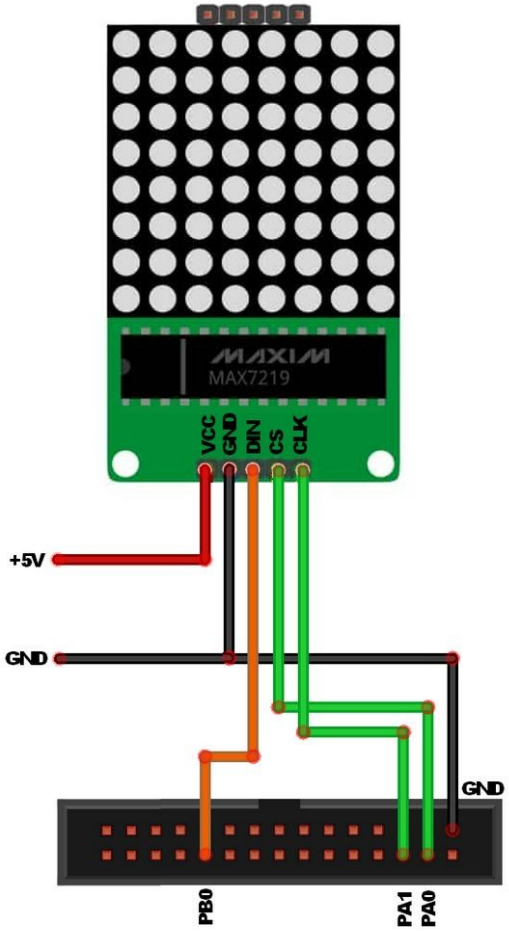


Figure 24: Contrôle d'une matrice de leds

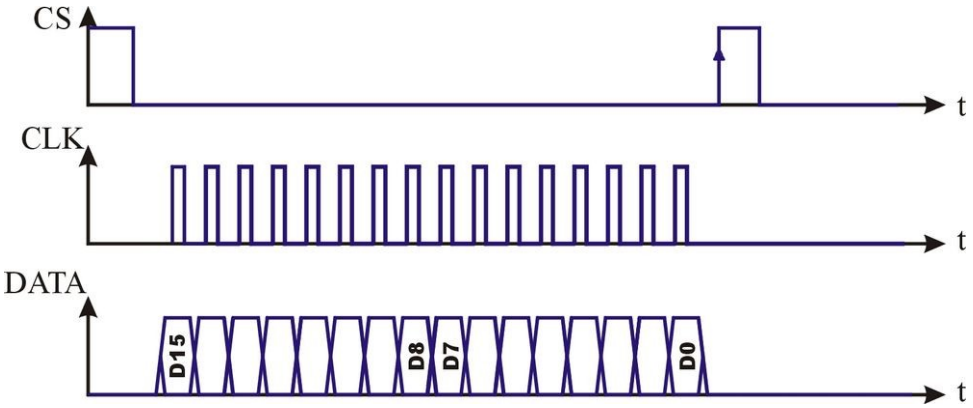


Figure 25: Signaux de contrôle de la matrice

Le contrôle de l'affichage est assuré par un circuit spécialisé (MAX7219) qui communique à l'aide de 3 fils avec la carte d'E/S. Les chronogrammes de la **Figure 25** montrent les signaux à utiliser. La communication se fait en série par blocs de 16 bits, le 1er octet désigne le registre destination tandis que le 2ème octet constitue la donnée.

Le but du programme est d'afficher 'HELLO' sur la matrice caractère par caractère. Chaque caractère va être codé puis stocké dans un tableau avant l'affichage.

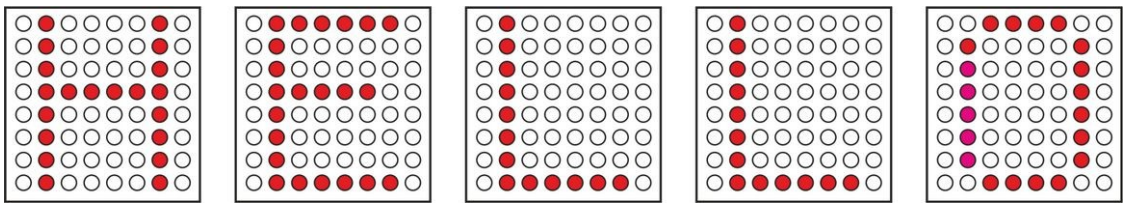


Figure 26: Message à visualiser

Le codage de chaque caractère se fait à l'aide de 8 octets (8x8 bits=64 bits soit 64 leds).

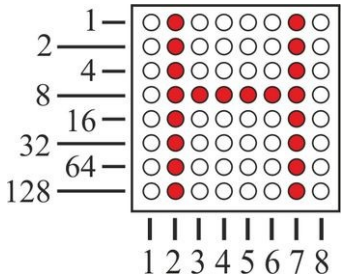


Figure 27: Codage d'un caractère

Chaque octet contrôle une colonne, chaque bit de l'octet est affecté à une led. La **Figure 27** indique le codage de la lettre 'H'. Les valeurs nécessaires sont donc : 0,255,8,8,8,8,255,0. Il suffit de procéder de la même façon pour coder les autres caractères.

Le programme est composé d'une partie en BASIC et d'une partie en assembleur.

```
5 HIMEM #7FFF
10 DIM C(40)
20 CLS
30 GOSUB 1000
40 PRINT "CONTROLE D'UNE MATRICE 8x8"
50 POKE #3FD,137
60 GOSUB 400
70 GOSUB 300

100 REM PROGRAMME PRINCIPAL
105 K=0
110 FOR J=0 TO 7
115 POKE #8062+J,C(K)
116 K=K+1
120 NEXT J
130 CALL #8033
140 GOSUB 200
145 GOSUB 400
150 IF K<39 THEN 110
160 GOSUB 200
170 GOSUB 200
180 GOTO 105
```

```
200 REM TEMPORISATION
210 FOR I=1 TO 500
220 NEXT I
230 RETURN

300 REM INITIALISATION
310 POKE #8060,9: POKE #8061,0: CALL #8000
320 POKE #8060,12: POKE #8061,1: CALL #8000
330 POKE #8060,10: POKE #8061,8: CALL #8000
340 POKE #8060,11: POKE #8061,7: CALL #8000
350 RETURN

400 REM CLS
410 FOR I=0 TO 7
420 POKE #8062+I,0
430 NEXT I
440 CALL #8033
450 RETURN

1000 REM ROUTINES LM
1010 FOR I=#8000 TO #804A
1020 READ N
1030 POKE I,N
1040 NEXT I
1060 DATA 169,0,141,250,3,173,96,128,32,26,128,173,97,128,32,26,128
1070 DATA 169,1,141,250,3,140,250,3,96,162,8,160,0,10,144,1,200
1080 DATA 140,251,3,160,2,140,250,3,160,0,140,250,3,202,208,234
1090 DATA 96,162,1,142,96,128,189,97,128,141,97,128,138,72,32,0
1095 DATA 128,104,170,232,224,9,208,235,96

1100 REM CARACTERES
1110 FOR I=0 TO 39
1120 READ C(I)
1130 NEXT I
1140 DATA 0,255,8,8,8,8,255,0,0,255,137,137,137,137,129
1150 DATA 0,0,255,128,128,128,128,128,0,0,255,128,128,128
1160 DATA 128,128,0,0,126,129,129,129,129,126,0
1170 RETURN
```

Le programme commence par réserver la mémoire puis appelle le sous programme situé à partir de la ligne 1000 pour mettre en place les routines en assembleur puis remplir le tableau avec les différents caractères du message.

Le programme appelle ensuite un sous-programme (ligne 400) permettant de vider l'affichage (CLS). Enfin le sous-programme situé à partir de la ligne 300 configure les différents registres du circuit intégré d'affichage (MAX7219).

Le programme principal envoie ensuite les valeurs codant les caractères par bloc de 8 en faisant une pause puis efface brièvement l'affichage entre chaque caractère. Une fois la séquence terminée, le programme fait une pause un peu plus longue puis recommence l'affichage.

Le code assembleur contenu dans les lignes DATA est le suivant :

```
            .ORG $8000

; Envoi d'une commande vers le MAX7219
Send_Com
            LDA #0
            STA $3FA
            LDA REG
            JSR Send8
            LDA DATA
            JSR Send8
            LDA #1
            STA $3FA
            STY $3FA
            RTS

; envoie 8 bits
Send8
            LDX #8
SC1
            LDY #0
            ASL A
            BCC SC2
            INY
SC2
            STY $3FB
            LDY #2
            STY $3FA
            LDY #0
            STY $3FA
            DEX
            BNE SC1
            RTS

; Envoi d'un bloc de 8 octets [REG  : 1-8]
Send_Bloc
            LDX #1
SB1
            STX REG
            LDA BLOC-1,X
            STA DATA
            TXA
            PHA
            JSR Send_Com
            PLA
            TAX
            INX
            CPX #9
            BNE SB1
            RTS

            .ORG $8060
REG
            .byte 0
DATA
            .byte 0
BLOC
            .byte 0,0,0,0,0,0,0,0
```