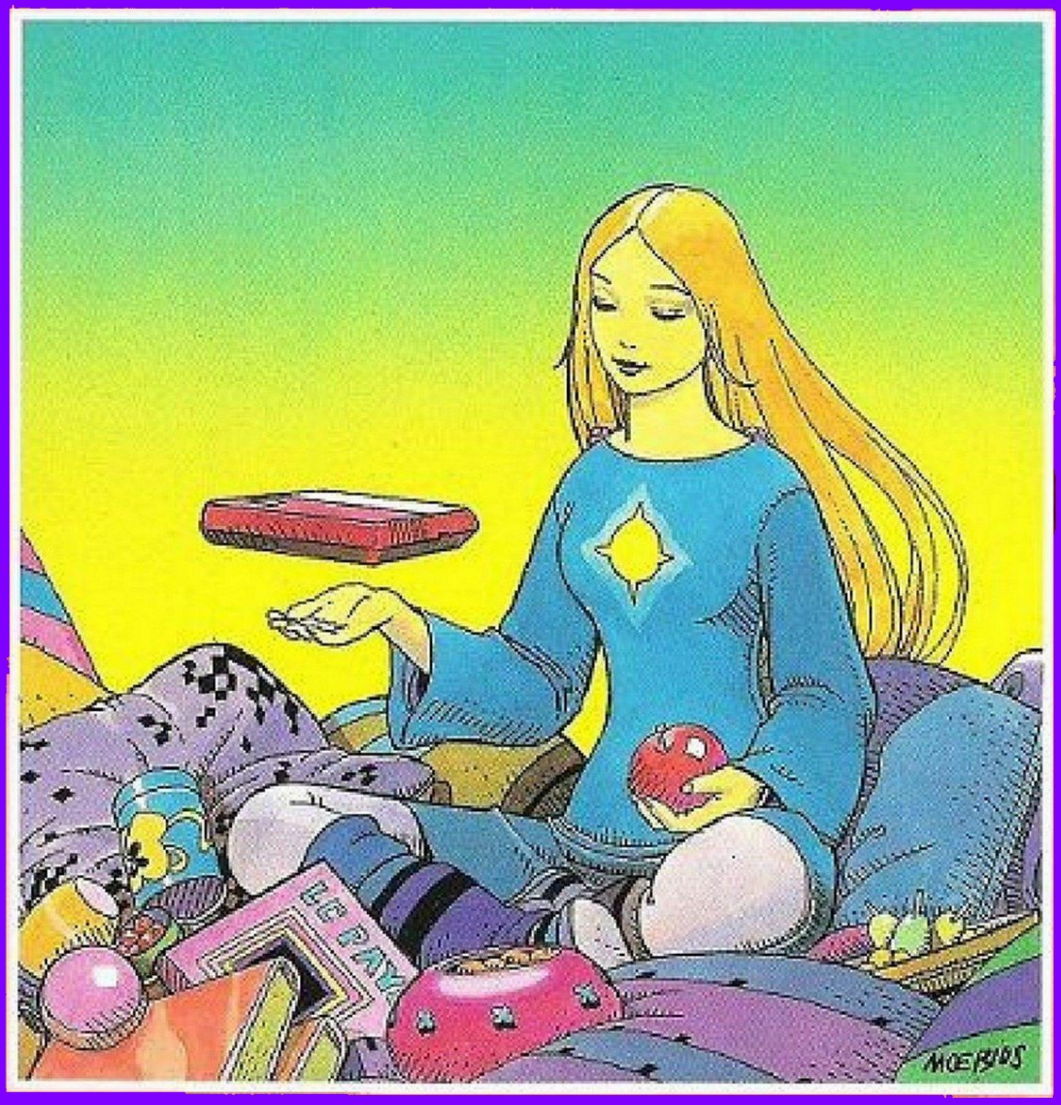
ALICE

13 EXERCICES AVEC LA CARTE 24 E/S



Fred_72 & 6502_man

Sommaire

Introduction

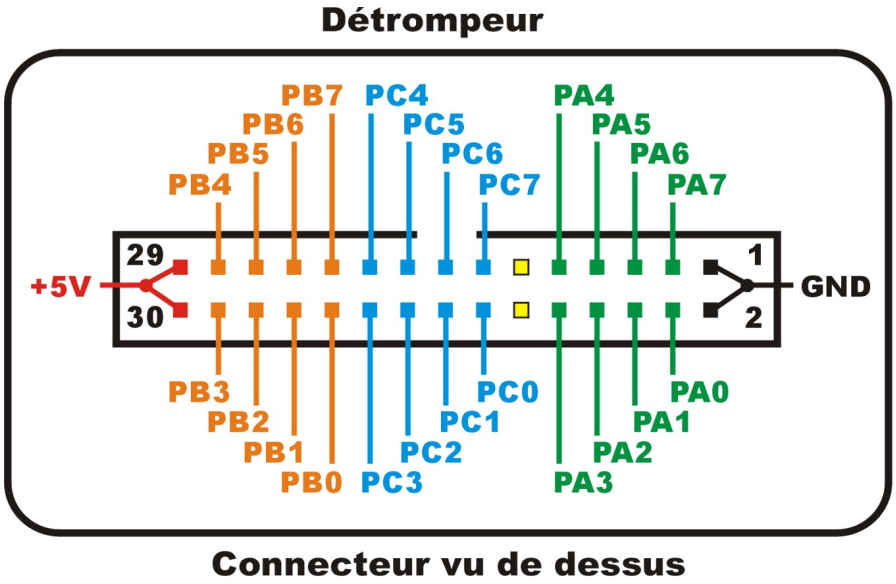
Ce document propose de vous initier à l'utilisation des entrées-sorties numériques avec votre ordinateur Alice 32 ou 90.

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 Alice.

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

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

La programmation se fait à l'aide des commandes POKE et PEEK du BASIC ou bien en langage machine avec les instructions LDAA et STAA. Le circuit 8255 dispose de 4 registres accessibles aux adresses suivantes :

Adresse (décimal)	Adresse (Hexadécimal)	Registre
48992	BF60	PORT A
48993	BF61	PORT B
48994	BF62	PORT C
48995	BF63	CONTROL

Tableau 1: Adresses des registres du circuit 8255

Registre de contrôle

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.

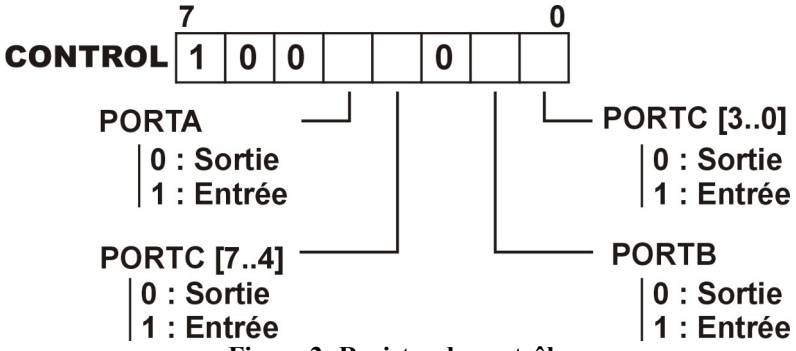


Figure 2: Registre de contrôle

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.

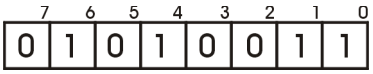


Figure 3: Octet en binaire

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.

Pour écrire sur un port, il suffit de calculer la valeur décimale à partir de l'image binaire que l'on souhaite envoyer puis d'utiliser l'instruction POKE avec l'adresse du PORT cible.
Par exemple POKE 48992,85 envoie la valeur 85 sur le port (en binaire 01010101) . Cette valeur provoque la mise à 1 des bits impairs et à 0 les bits pairs du PORTA

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.

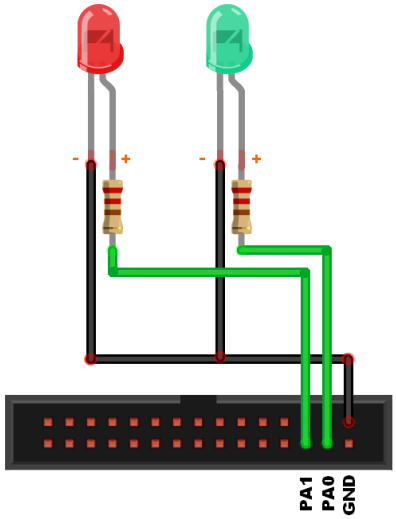


Figure 4: Leds clignotantes

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

```
5 CLS
10 PRINT "COMMANDE DES LEDS DU PORT A"
30 POKE 48995,139
40 POKE 48992,3
50 GOSUB 100
60 POKE 48992,0
70 GOSUB 100
80 GOTO 40

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

L'instruction POKE 48995,139 de 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**.

L’instruction POKE 48992,3 de la ligne 40 met les bits PA0 et PA1 à 1, les autres (PA2 à PA7) sont à 0. Cela allume les 2 leds.
L’instruction POKE 48992,0 de la ligne 60 met tous les bits du PORTA à 0. Cela 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 48992,1
60 POKE 48992,2
```

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)

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.

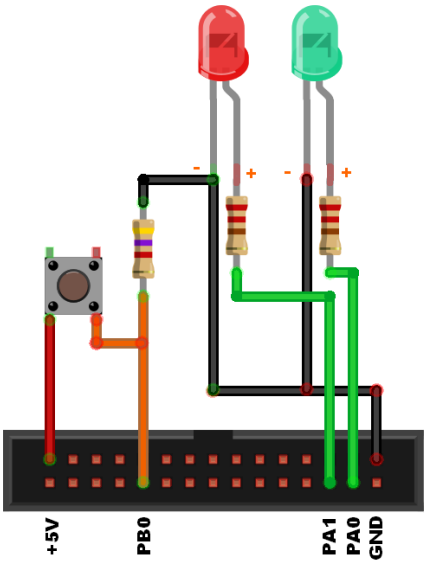


Figure 7: Commande par BP

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 rouge lorsque le bouton poussoir est appuyé et la led verte lorsqu'il est relâché.

```
5 CLS
10 PRINT "COMMANDE PAR BP DES LEDS DU PORT A"
30 POKE 48995,139
40 IF (PEEK(48993) AND 1)<>0 THEN GOTO 70
50 POKE 48992,2
60 GOTO 40
70 POKE 48992,1
80 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 rouge ou la seule led verte.
La lecture fait appel à l'instruction PEEK 48993 qui permet de lire l'état des 8 bits du PORTB. Cependant, 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 :

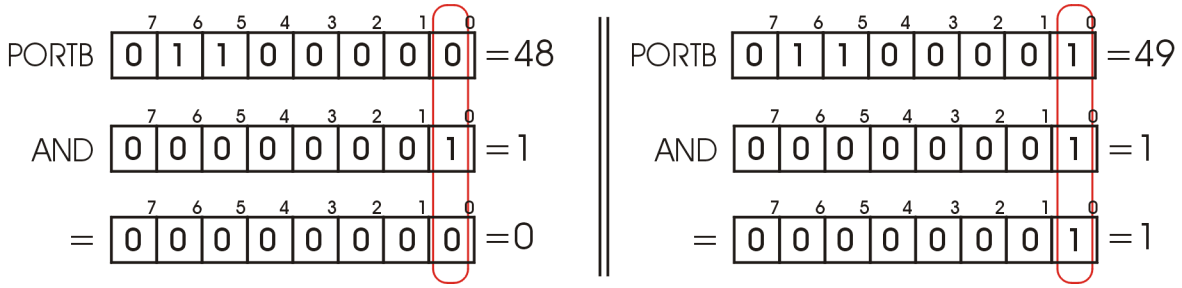


Figure 8: Filtrage avec un AND

Sur cette figure, les bits 5 et 6 sont à 1, ce qui correspond à des états indésirables ou bien 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 appuyé) soit 1 (BP relâché).

Le programme fait cette opération puis oriente le programme vers la ligne 50 ou 70 en fonction de l'état logique du bouton poussoir.

Modification 1

On ajoute un 2ème bouton poussoir connecté à PB1.

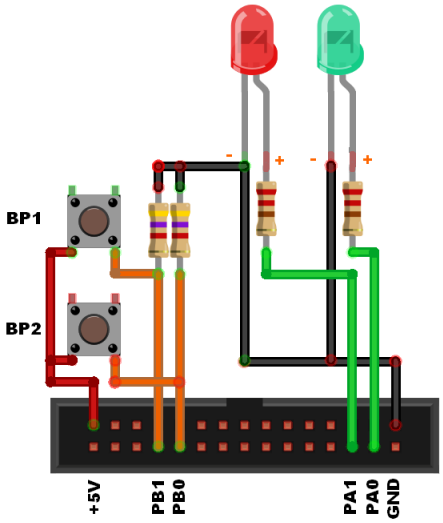


Figure 9: Commande par 2 BPs

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.

```
5 CLS
10 PRINT "COMMANDE PAR 2 BOUTONS DES LEDS DU PORT A"
30 POKE 48995,139
40 A=1
50 IF (PEEK(48993) AND 1)<>0 THEN A=0
60 B=2
70 IF (PEEK(48993) AND 2)<>0 THEN B=0
80 POKE 48992, A OR B
90 GOTO 40
```

La lecture fait toujours appel à l'instruction PEEK associée à l'opérateur AND, mais cette fois les valeurs 1 et 2 sont utilisées.

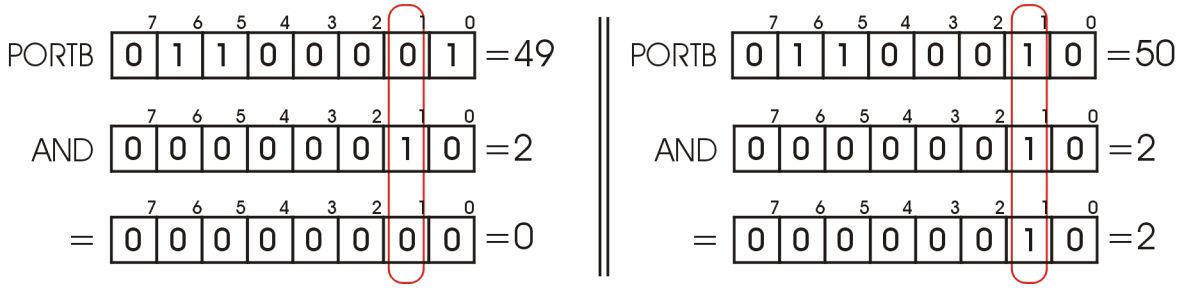


Figure 10: Filtrage du 2ème bouton poussoir

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 :

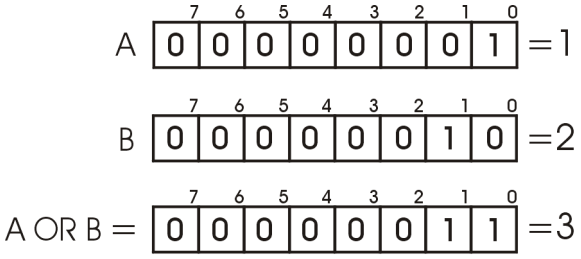


Figure 11: Association des variables A et B du programme

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 ?

Modification 2

On retire les boutons poussoirs et on met à la place une photorésistance (LDR) associée à une résistance de 15 kilo-ohms. La photorésistance est un composant dont la résistance varie selon la lumière. Si le composant est exposé à une lumière vive, sa résistance est faible. Si le composant est maintenue dans l'obscurité, sa résistance est grande.

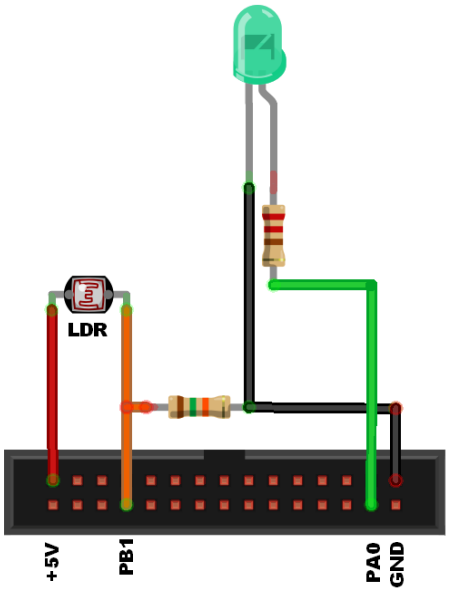


Figure 12: Photorésistance et Led

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 pour obtenir le fonctionnement désiré ?

Modification 3

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

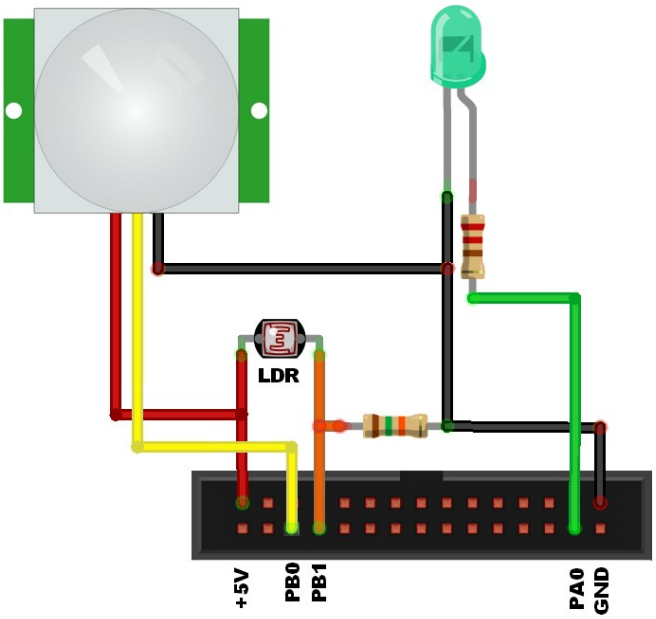


Figure 13: Interrupteur crépusculaire 2

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 10s. Comment faut-il modifier le programme pour obtenir le fonctionnement désiré ?

Exercice 3 (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.

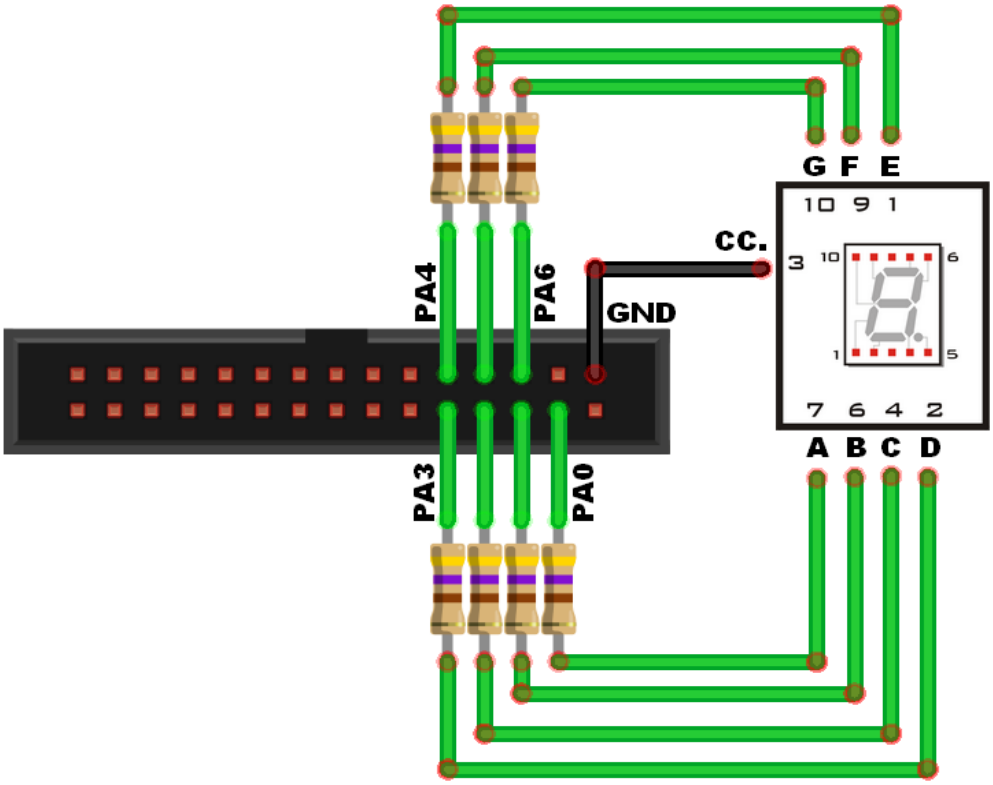


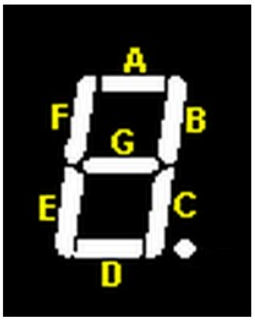
Figure 14: Commande d'un afficheur 7 segments

Codage des chiffres

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

L'envoi d'une valeur numérique de 0 à 9 sur le PORTA ne suffit pas à afficher un chiffre. Il faut utiliser un codeur 7 segments qui peut prendre la forme d'un composant (CD4511 par exemple) ou un morceau de programme qui convertit la valeur numérique en une série de '1' et de '0' conduisant à l'affichage des segments pour former le chiffre souhaité.

C'est cette 2ème solution qui est utilisée ici. 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.

```
5 CLS
10 PRINT "COMPTAGE SUR UN AFFICHEUR"
20 DIM A(10)
30 FOR V=0 TO 9
40 READ A(V)
50 NEXT V
55 DATA 63,6,91,79,102,109,125,7,127,111

60 POKE 48995,139
70 FOR N=0 TO 9
```

```
80 POKE 48992, A(N)
90 GOSUB 200
100 NEXT N
110 GOTO 70

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

La 1ère partie du programme permet de remplir les cases du tableau A avec les 10 valeurs permettant de visualiser des chiffres compréhensibles sur l'afficheur 7 segments.

La ligne 60 permet de mettre le PORTA en sortie. Les PORTB et PORTC sont laissés en entrée.

Les lignes 70 et 100 mettent en place une boucle de comptage FOR-NEXT durant laquelle N parcourt les valeurs 0 à 9.

Enfin la ligne 80 envoie sur le PORTA la valeur correspondant au chiffre N en utilisant le tableau pour transformer cette valeur numérique en code '7 segments'.

Modification 1

Un 2ème afficheur 7 segments est ajouté. Il est branché sur le PORTB. Cela va permettre maintenant de compter de 0 à 99.

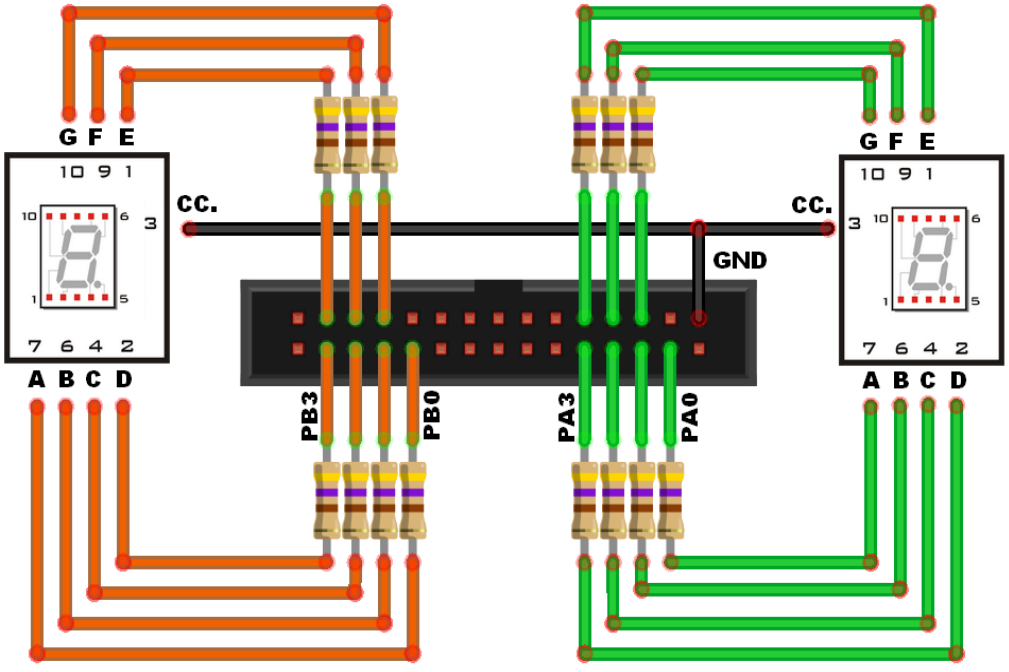


Figure 15: Commande de 2 afficheurs 7 segments

Le programme BASIC est modifié ainsi :

```
60 POKE 48995,137
70 FOR N=0 TO 99
75 D=INT(N/10)
80 U=N-10*D
85 POKE 48992, A(U)
90 POKE 48993, A(D)
100 GOSUB 200
110 NEXT N
120 GOTO 70
```

La ligne 60 est modifiée afin de mettre les PORTA et PORTB en sortie, le PORTC inutilisé reste en entrée.

A partir de la ligne 75, la variable D contient le chiffre des dizaines tandis que la variable U contient le chiffre des unités. Ces 2 valeurs sont envoyées, après codage 7 segments, sur les ports A et B.

Modification 2

On ajoute un bouton poussoir et une résistance de **4.7 kilo-ohms** connectés sur PC0.

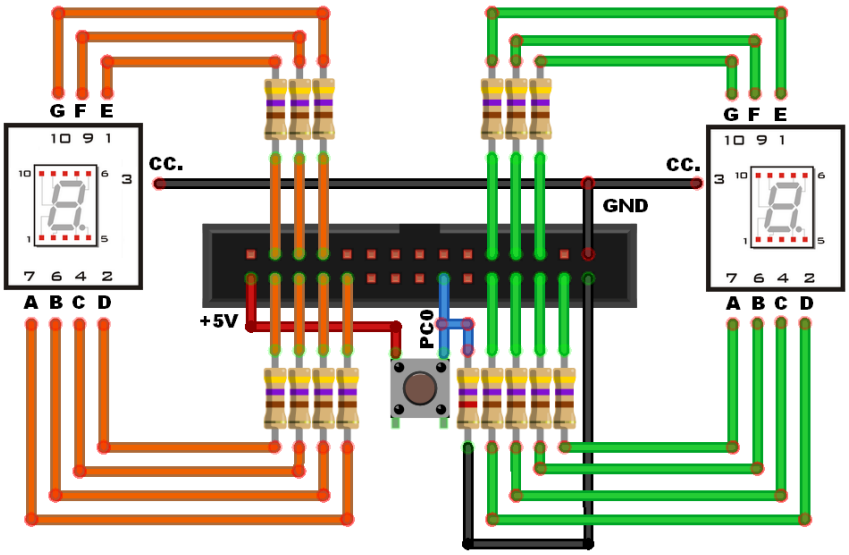


Figure 16: Commande de 2 afficheurs pilotés par un bouton poussoir

On souhaite pouvoir changer le sens de comptage en appuyant sur le bouton poussoir. Lorsque le bouton poussoir est relâché, le programme fonctionne comme précédemment (comptage) mais lorsque le bouton poussoir est appuyé, les afficheurs décomptent.

Question

Comment faut-il modifier le programme pour changer le sens de comptage selon l'état du bouton poussoir ?

Exercice 4 (7 segments-ASM)

But : Commander 2 afficheurs 7 segments

Le but de cet exercice entièrement en assembleur est d'afficher sur un des 2 afficheurs, le caractère frappé au clavier. Le choix de l'afficheur se fait en appuyant au clavier la combinaison SHIFT+1 ou SHIFT+2.

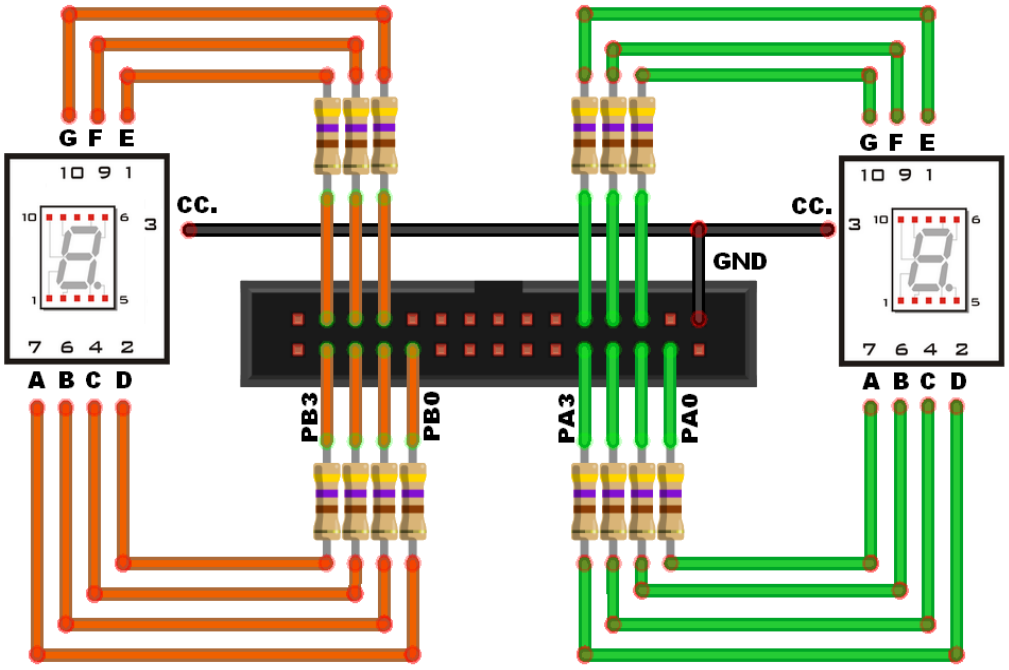
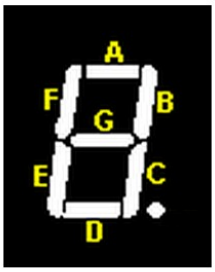


Figure 17: Commande de 2 afficheurs 7 segments

La 1ère opération consiste à définir les codes des différents caractères sur un afficheur 7 segments. Pour cet exercice, les codes seront traduits en hexadécimal plus approprié pour un codage en assembleur. Tous les caractères alphanumériques ne peuvent pas être visualisés sur un afficheur 7 segments. Pour les caractères qui ne sont pas dans le tableau, l'afficheur reste éteint (soit la valeur 0).

Codage des caractères



Caractère	G	F	E	D	C	B	A	Valeur hexadécimale
0	0	1	1	1	1	1	1	3F
1	0	0	0	0	1	1	0	06
2	1	0	1	1	0	1	1	5B
3	1	0	0	1	1	1	1	4F
4	1	1	0	0	1	1	0	66
5	1	1	0	1	1	0	1	6D
6	1	1	1	1	1	0	1	7D
7	0	0	0	0	1	1	1	07
8	1	1	1	1	1	1	1	7F
9	1	1	0	1	1	1	1	6F
A	1	1	1	0	1	1	1	77
B	1	1	1	1	1	0	0	7C
C	0	1	1	1	0	0	1	39
D	1	0	1	1	1	1	0	5E
E	1	1	1	1	0	0	1	79
F	1	1	1	0	0	0	1	71
H	1	1	1	0	1	1	0	76
I	0	1	1	0	0	0	0	30
J	0	0	0	1	1	1	0	0E
L	0	1	1	1	0	0	0	38
N	1	0	1	0	1	0	0	54
O	1	0	1	1	1	0	0	5C
P	1	1	1	0	0	1	1	73
S	1	1	0	1	1	0	1	6D
T	1	1	1	0	0	0	0	70
U	0	0	1	1	1	0	0	1C
Y	1	1	0	1	1	1	0	6E
°	1	1	0	0	0	1	1	63

Tableau 6: Codes pour afficher les caractères sur un afficheur 7 segments

Le programme est à éditer en mémoire à partir de l'adresse $5000

```
        .ORG $5000

        JSR $FBD6           ; efface l'écran
        LDD #$0000
        STD $3280           ; LDD = x,y du curseur
        LDX #Text7segs-1
        JSR $E7A8           ; Affiche du texte

        LDAA #$89
        STAA $BF63          ; PORTA et PORTB en sortie, PORT C en entrée
        LDAA #$3F           ; AFFICHE '0' sur l'afficheur droit
        STAA $BF60
        LDAA #$06           ; AFFICHE '1' sur l'afficheur gauche
        STAA $BF61

        LDX #$BF20
        STX WorkingArea

; attente touche
WaitKey
        JSR $F883           ; A = code ascii de la touche enfoncée
                            ; Z = 0 aucun touche
        BEQ WaitKey
        CMPA #$21           ; afficheur 1
        BNE Continue2
        LDX #$BF60
        STX WorkingArea
        JMP WaitKey
Continue2
        CMPA #$22           ; afficheur 2
        BNE Continue3
        LDX #$BF61
        STX WorkingArea
        JMP WaitKey
Continue3
        PSHA
        JSR $F9C6           ; Affiche le caractère contenu dans A
        PULB
        LDX #CodageSegments
        SUBB #$30
        ABX
        LDAA 0,X

        LDX WorkingArea
        STAA ,X
        JMP WaitKey

Text7segs
        .DB "CONTROLE D'AFFICHEURS 7 SEGMENTS"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "TAPEZ UN CHIFFRE OU UNE LETTRE :"
        .DB 13
        .DB "SHIFT ET 1 OU 2 POUR CHOISIR L'AFFICHEUR"
```

```
        .DB 13
        .DB 0

CodageSegments
                .DB $3F,$06,$5B,$4F,$66,$6D,$7D,$07,$7F,$6F
                .DB $00,$00,$00,$00,$00,$00,$00,$77,$7C,$39
                .DB $3E,$79,$71,$00,$76,$06,$1E,$00,$38,$00
                .DB $00,$5C,$73,$00,$00,$5D,$70,$3E,$5E,$00
                .DB $00,$00,$00,$63
WorkingArea
                .DB 0,0

                .END
```

La 1ère partie du programme permet d'afficher du texte sur l'écran puis place les chiffre 0 et 1 sur les afficheurs.

Le programme attend ensuite l'appui sur une touche. Si l'utilisateur appuie sur la combinaison "SHIFT+1" ou "SHIFT+2", cela change l'adresse stockée dans "Working Area" afin de changer la destination (PORTA ou PORTB). Si c'est une autre touche qui est appuyée, le caractère correspondant s'affiche sur le dernier afficheur sélectionné.

La table de codage est placée à la fin du programme, elle reprend les valeurs définies dans le **Tableau 6**. Elle est suivie par une zone de 2 octets permettant de stocker l'adresse du port en cours d'utilisation.

Question

Comment faut-il modifier le système pour ajouter un 3ème afficheur sur le PORTC et le commander par la combinaison "SHIFT+3" ?

Exercice 5 (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.

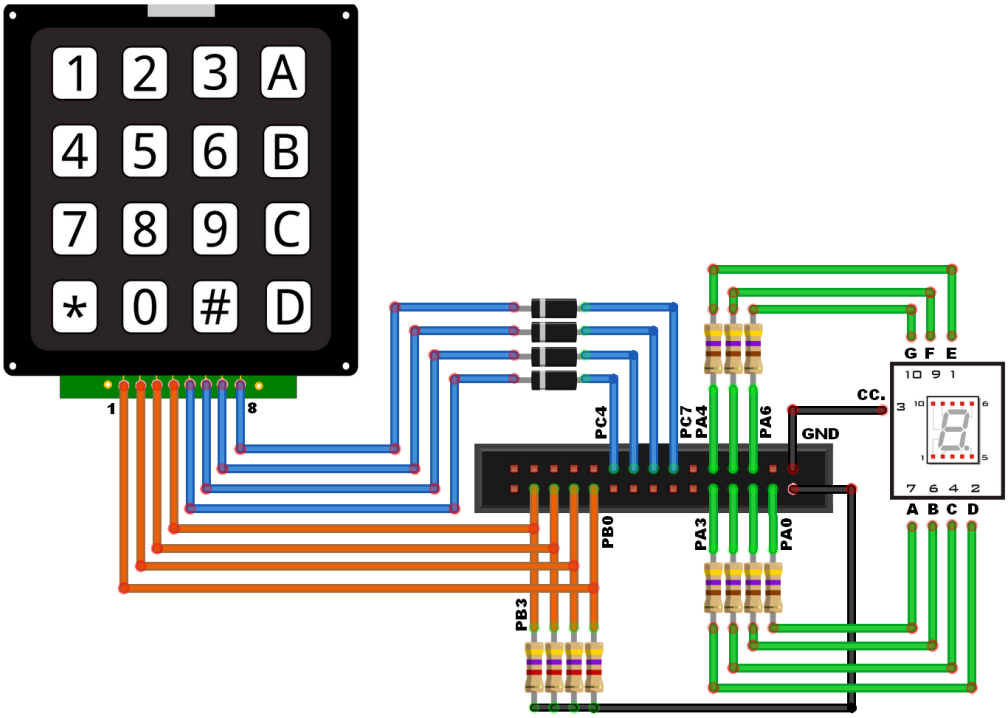


Figure 18: Clavier et afficheur 7 segments

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.

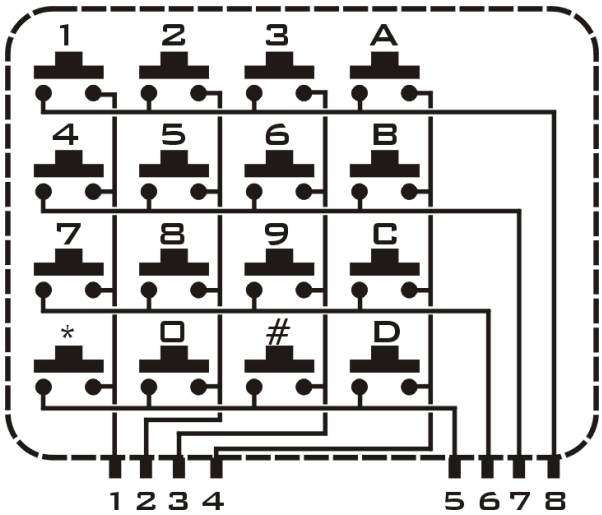


Figure 19: Matrice du clavier 16 touches

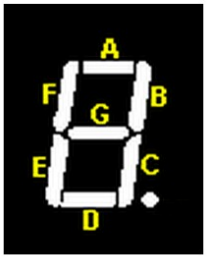
Pour lire ce clavier, il faut disposer de 4 entrées et 4 sorties. Le principe consiste à alimenter une seule rangée à la fois puis à lire les 4 colonnes. La lecture du clavier nécessite donc 4 cycles. Il ne reste plus ensuite qu'à identifier la touche pressée.

Sur le schéma, il y a des résistances de tirage à '0' et des diodes. Les résistance 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 lors de **l'exercice 3** mais il reste les lettres 'A' à 'F'. Le **Tableau 7** donne les nouvelles valeurs associées à ces 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 7: Codage des caractères supplémentaires

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

```
5 CLS
10 PRINT "LECTURE CLAVIER 16 TOUCHES"
20 DIM A(16),C1(5),C2(5),C3(5),C4(5)
30 FOR V=0 TO 15
40 READ A(V)
50 NEXT V
55 DATA 63,6,91,79,102,109,125,7,127,111,119,124,57,94,121,113
60 FOR V=1 TO 4
70 READ C1(V),C2(V),C3(V),C4(V)
80 NEXT V
85 DATA 1,4,7,14,2,5,8,0,3,6,9,15,10,11,12,13

100 POKE 48995,131
110 GOSUB 200
120 IF KB<>255 THEN POKE 48992,A(KB)
125 IF KB=255 THEN POKE 48992,0
130 GOTO 110

200 REM LECTURE CLAVIER
210 POKE 48994,16
220 R1=PEEK(48992) AND 15
230 POKE 48994,32
240 R2=PEEK(48992) AND 15
250 POKE 48994,64
260 R3=PEEK(48992) AND 15
270 POKE 48994,128
280 R4=PEEK(48992) 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
```

La 1ère partie du programme 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 partie 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 ensuite 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 6 (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.

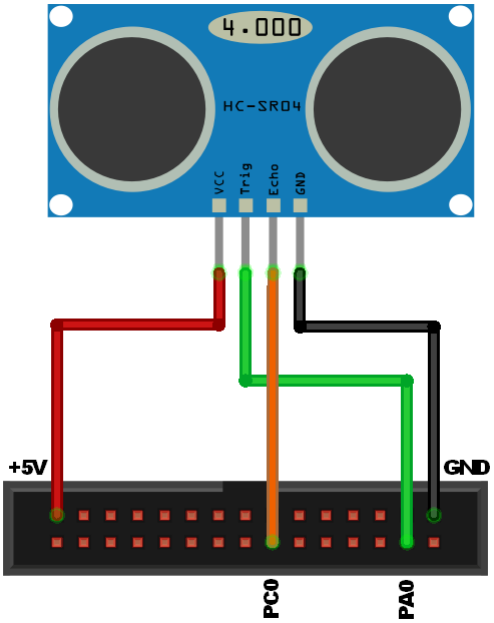


Figure 20: Mesure de distance

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).

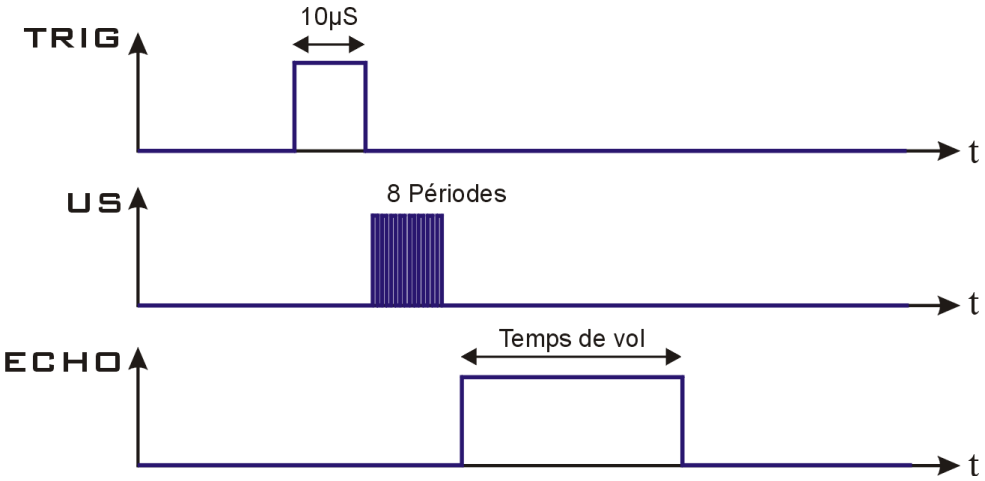


Figure 21: SR04 - Principe de la mesure

La mesure commence lorsque le module SR04 reçoit une impulsion de 10µs. La commande et la mesure nécessitent des temps très brefs, aussi cette partie sera assurée par un programme en assembleur. Le programme est donc constitué d'une partie en BASIC et d'une partie en assembleur.

```
5 CLS
10 FOR A=0 TO 91
20 READ V
30 POKE 17670+A,V
40 NEXT A
50 DATA 206,0,0,255,69,0,255,69,2,255,69,4,134,1,183,191
60 DATA 96,134,1,134,1,134,0,134,0,183,191,96,182,191
70 DATA 98,132,1,39,249,12,182,69,5,137,1,183,69,5,182,69,4
75 DATA 137,0,183,69,4,182,69,3,137,0,183,69,3,182,69,2,137
80 DATA 0,183,69,2,182,69,1,137,0,183,69,1,182,69,0,137,0,183
85 DATA 69,0,182,191,98,132,1,38,200,57

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 POKE 48995,139
140 EXEC 17670
150 D=PEEK(17668)*256+PEEK(17669)
160 PRINT "DISTANCE  : ",D
170 GOTO 140
```

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

```
        .ORG $4500
WorkingArea
        .DB 0,0,0,0,0,0

Start
        LDX #$0000
        STX WorkingArea
        STX WorkingArea+2
        STX WorkingArea+4

        LDAA #$01
        STAA $BF60          ; envoie une impulsion sur le trigger

                            ; il faut une durée minimal de 10µs
        LDAA #$01
        LDAA #$01
        LDAA #$00           ; vérifié à l'oscillo     => env. 12us

        LDAA #$00
        STAA $BF60          ; met à zéro le trigger

Attendhaut                  ; attente du signal à l'état haut
        LDAA $BF62          ; lecture PORTC
        ANDA #$01
        BEQ Attendhaut
```

```
Distance                ; compte le temps
        CLC
        LDAA WorkingArea+5
        ADCA #$01
        STAA WorkingArea+5
        LDAA WorkingArea+4
        ADCA #$00
        STAA WorkingArea+4
        LDAA WorkingArea+3
        ADCA #$00
        STAA WorkingArea+3
        LDAA WorkingArea+2
        ADCA #$00
        STAA WorkingArea+2
        LDAA WorkingArea+1
        ADCA #$00
        STAA WorkingArea+1
        LDAA WorkingArea+0
        ADCA #$00
        STAA WorkingArea+0

        LDAA $BF62          ; attente du signal à l'état bas
        ANDA #$01
        BNE Distance
        RTS

        .END
```

Exercice 7 (SPEAKER)

But : Commander un haut parleur pour produire des sons

Cet exercice permet d'émettre un son dans un haut parleur à partir d'une sortie numérique. Il utilise un amplificateur audio afin d'amplifier le signal pour l'envoyer vers un petit haut-parleur. En effet, le signal issu de la carte d'entrées-sorties n'est pas assez puissant pour alimenter directement un haut-parleur.

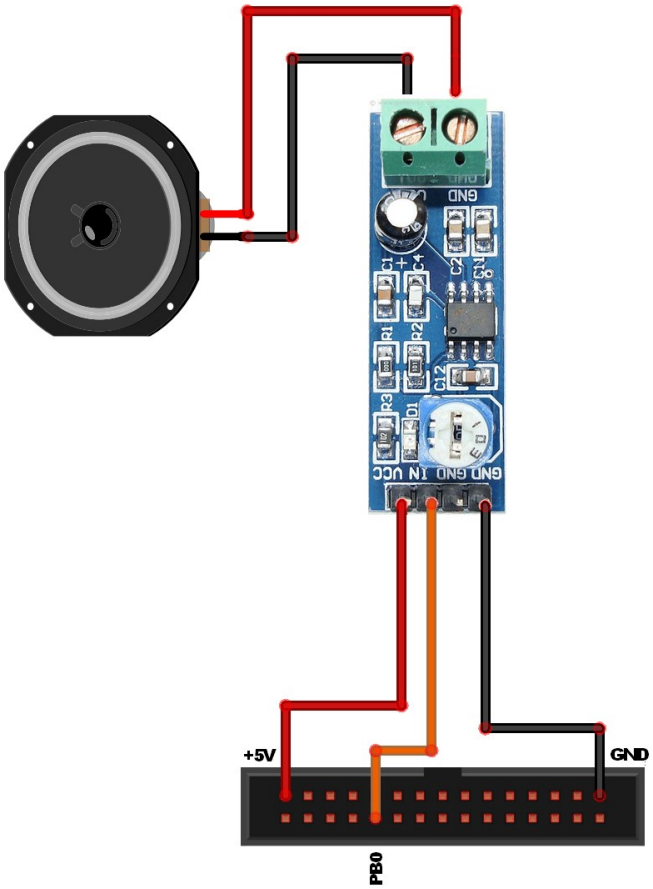


Figure 22: Génération d'un son

La période du signal de sortie étant très faible (quelques millisecondes), le programme est entièrement écrit en assembleur.

Le programme est à éditer en mémoire à partir de l'adresse $5000

```
        .ORG $5000

        JSR $FBD6           ; efface l'écran
        LDD #$0000
        STD $3280           ; LDD = x,y du curseur
        LDX #TextSPEAKER-1
        JSR $E7A8

        LDAA #$8B
        STAA $BF63          ; PORTA en sortie, PORTB et PORTC en entrée
        LDAA #$00
        STAA $BF60          ; PORTA=0

; attente touche
WaitKey
        JSR $F883           ; A = code ascii de la touche enfoncée
                            ; Z = 0 aucun touche
        BEQ WaitKey

        STAA WorkingArea+2
        RORA
        STAA WorkingArea+3
        LDX #$10
        STX WorkingArea

; TEMPO DIFFÉRENTE SELON LA TOUCHE
; A CONTIENT LE CODE ASCII DE LA TOUCHE (SERT A FAIRE VARIER LE SON)

BoucleSon
        LDAB #$01           ; émet un son sur le SPEAKER
        STAB $BF60
        JSR TEMPO

        LDAB #$00           ; coupe le son sur le SPEAKER
        STAB $BF60
        JSR TEMPOB

        LDX WorkingArea
        DEX
        STX WorkingArea
        BNE BoucleSon

        JMP WaitKey

TEMPO
        LDAA WorkingArea+2
TimingZ
        LDX #$0010
Timing
        DEX
        BNE Timing
        DECA
        BNE TimingZ
        RTS
```

```
TEMPOB
        LDAA WorkingArea+3
TimingZB
        LDX #$0010
TimingB
        DEX
        BNE TimingB
        DECA
        BNE TimingZB
        RTS


TextSPEAKER
        .DB "UTILISATION D'UN SPEAKER"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "APPUYEZ UNE TOUCHE POUR EMETTRE UN SON :"
        .DB 13
        .DB "DIFFERENT SELON LA TOUCHE."
        .DB 13,13
        .DB 0

WorkingArea
        .DB 00,00,00,00

        .END
```

Exercice 8 (SR04 + SPEAKER)

But : Réaliser un détecteur de proximité

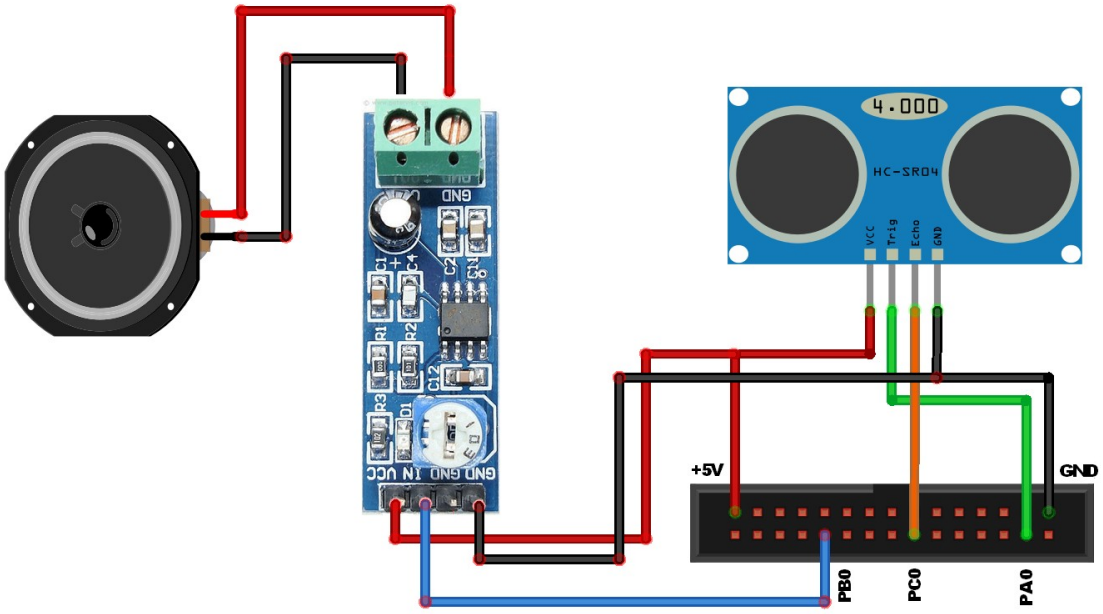


Figure 23: Détecteur de proximité

En combinant les programmes des 2 exercices précédents, il est possible de réaliser un radar de proximité. La hauteur du son dans le haut-parleur va dépendre de la distance de l'obstacle.

Le programme est à éditer en mémoire à partir de l'adresse $5000

```
OUTTX.EQU $E7A8          ; LDX = adresse chaîne (terminé par null)
OUTCA.EQU $F9C6          ; LDAA = code ascii
SCANK.EQU $F883          ; <= LDAA = code ascii de la touche
                         ; enfoncée Z = 0 aucun touche
CLS         .EQU $FBD6          ; efface l'écran
CURSOR      .EQU $3280          ; LDD = y,x du curseur

        .ORG $5000

        LDAA #$89        ; PORTA, PORTB en sortie, PORTC en entrée
        STAA $BF63

        LDAA #$00        ; AUCUN SON DU BUZZER
        STAA $BF61

        JSR CLS
Start
        LDD #$0000
        STD CURSOR
        LDX #TextRadar-1
        JSR OUTTX
```

```
ReStart
        LDX #$0000
        STX WorkingArea
        STX WorkingArea+2
        STX WorkingArea+4

        LDAA #$FF
        STAA $BF60          ; envoie une impulsion sur le trigger

        LDAA #$FF           ; il faut une durée minimal de 10µs
        LDAA #$FF           ; vérifie à l'oscillo     => env. 12us
        LDAA #$00
        LDAA #$00
        STAA $BF60          ; met à zéro le trigger

Attendhaut                  ; attente du signal à l'état haut
        LDAA $BF62
        ANDA #$01
        BEQ Attendhaut

Distance
        CLC
        LDAA WorkingArea+5
        ADCA #$01
        STAA WorkingArea+5
        LDAA WorkingArea+4
        ADCA #$00
        STAA WorkingArea+4
        LDAA WorkingArea+3
        ADCA #$00
        STAA WorkingArea+3
        LDAA WorkingArea+2
        ADCA #$00
        STAA WorkingArea+2
        LDAA WorkingArea+1
        ADCA #$00
        STAA WorkingArea+1
        LDAA WorkingArea+0
        ADCA #$00
        STAA WorkingArea+0

        LDAA $BF62          ; attente du signal à l'état bas
        ANDA #$01
        BNE Distance

        LDD #$0A00
        STD CURSOR
        LDX #TextDistance-1
        JSR OUTTX

        LDAA WorkingArea+3
        JSR HPRINT
        LDAA WorkingArea+4
        JSR HPRINT
        LDAA WorkingArea+5
        JSR HPRINT
```

```
        LDAB #$10
Son
        LDAA #$FF           ; EMET UN SON  BUZZER
        STAA $BF61          ; connecté au PORT B

        LDAA WorkingArea+5

Tempo
        DECA
        BNE Tempo

        LDAA #$00           ; AUCUN SON DU BUZZER
        STAA $BF61          ; connecté au PORT B

        LDAA WorkingArea+5
Tempo2
        DECA
        BNE Tempo2

        DECB
        BNE Son

FIN
        JMP ReStart


HPRINT:
        PSHA                ; print  HEXA number
        LSRA
        LSRA
        LSRA
        LSRA
        JSR Hex2Asc
        JSR OUTCA
        PULA
        JSR Hex2Asc
        JSR OUTCA
        RTS

Hex2Asc:
        ANDA #$0F           ; converti hexa en ascii
        CMPA #$0A
        BPL LetAsc
        ADDA #$30
        RTS
LetAsc
        ADDA #$30+07
        RTS


TextRadar
        .DB "CONTROLE D'UN SONAR TYPE ARDUINO"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "POINTEZ LE SONAR VERS LE PLAFOND"
        .DB 13
```

```
        .DB "PLACEZ VOTRE MAIN AU DESSUS"
        .DB 13
        .DB "DU RADAR POUR FAIRE VARIER LE"
        .DB 13
        .DB "SON EN FONCTION DE LA DISTANCE"
        .DB 13
        .DB 0

TextDistance
        .DB "DISTANCE EN CM (HEXA) :"
        .DB 0

WorkingArea
        .DB 00,00,00,00,00,00

        .END
```

Exercice 9 (LCD)

But : Piloter un afficheur LCD

Cet exercice présente la méthode pour contrôler un afficheur LCD (2 lignes de 16 caractères) à partir de votre ordinateur Alice. 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.

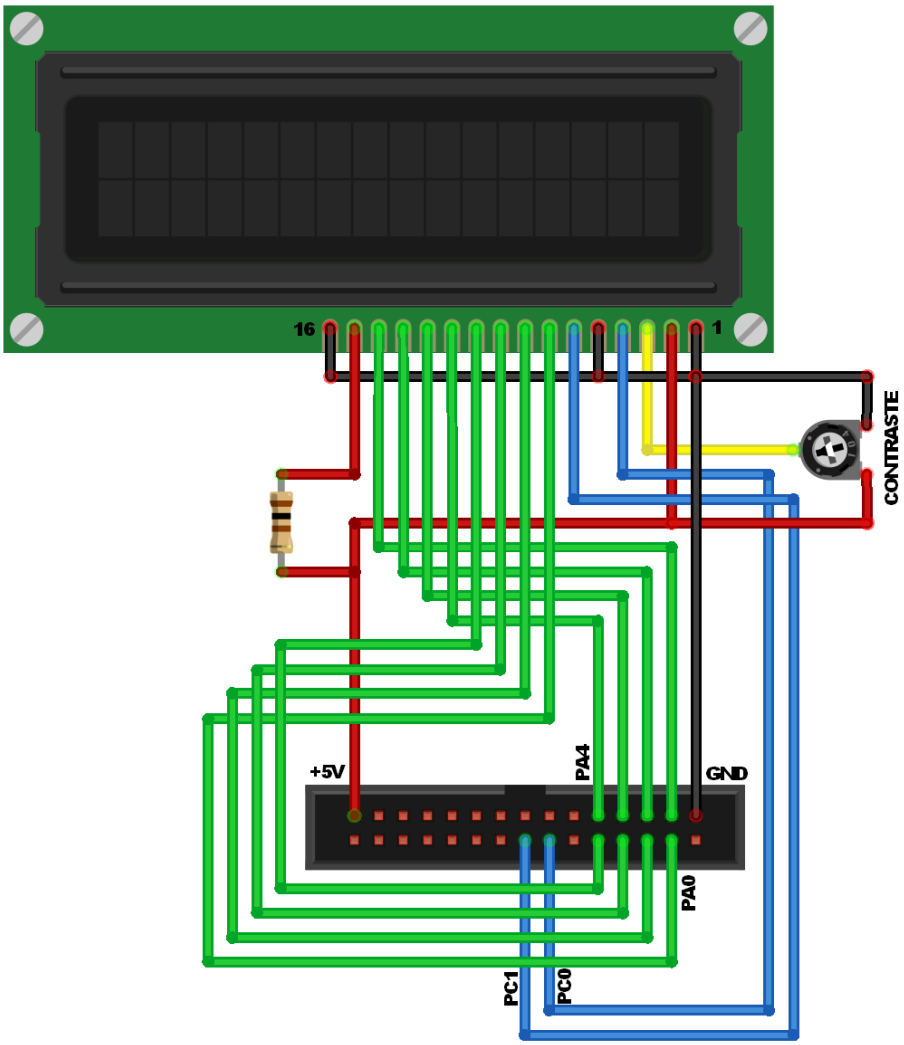


Figure 24: Contrôle d'un afficheur LCD

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, 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 entrée.

Le programme sera composé d'une partie en BASIC et d'une partie en assembleur. Le code machine sera entré en mémoire par le programme BASIC. L'appel aux 2 routines assembleurs se fait à l'aide de l'instruction EXEC.

```
5 CLS
10 PRINT "CONTROLE D'UN AFFICHEUR LCD"
20 PRINT
30 POKE 48995,138
40 FOR I=0 TO 35
50 READ N
60 POKE 17664+I,N
65 NEXT I
70 DATA 134,00,183,191,96,134,0,183,191,98,138,2,183,191,98
75 DATA 134,0,183,191,98,206,1,44,9,38,253,57,134,0,183,191
80 DATA 96,134,1,32,227

100 GOSUB 300
110 A$="BONJOUR LE MONDE"
120 B$="MATRA ALICE"
125 REM AFFICHAGE LIGNE1
130 POKE 17665,1
135 EXEC 17664
140 T$=A$
150 GOSUB 200
155 REM AFFICHAGE LIGNE2
160 POKE 17665,192
165 EXEC 17664
170 T$=B$
180 GOSUB 200
190 END

200 REM ENVOI UNE CHAINE DE CARACTERES
205 FOR I=1 TO LEN (T$)
210 POKE 17692,ASC(MID$(T$,I,1))
220 EXEC 17691
230 NEXT I
240 RETURN

300 REM INIT AFFICHEUR
310 POKE 17665,56
320 EXEC 17664
330 POKE 17665,12
340 EXEC 17664
350 RETURN
```

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

```
            .ORG $4500

Send_Com                ; Envoi d'une commande (@17664)
            LDAA #0     ; la valeur est modifiée par POKE  17665
            STAA $BF60
            LDAA #0

Continue
            STAA $BF62
            ORAA #2
            STAA $BF62
            LDAA #0
            STAA $BF62

            LDX #$012C
Tempo
            DEX
            BNE Tempo
            RTS

Send_DATA               ; Envoi d'un caractère (@17691)
            LDAA #0     ; la valeur est modifiée par POKE 17692
            STAA $BF60
            LDAA #1
            BRA Continue

            .END
```

La 1ère partie du programme (lignes 5 à 80) place le code machine en mémoire et met les PORTA et PORTC en sortie. Le PORTB inutilisé reste en entrée.

Le sous-programme placé à partir de la ligne 200 assure l'envoi caractère par caractère de la chaîne de caractère T$ par le biais de la routine Send_DATA située à l'adresse mémoire 17691.

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.

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

Exercice 10 (MOTEUR)

But : Piloter un moteur à courant continu

Cet exercice présente la méthode pour piloter un moteur à courant continu en faisant varier sa vitesse. Les sorties de la carte 24 E/S ne sont pas capables de commander directement un moteur. Il est donc nécessaire d'utiliser un transistor de puissance, ici un MOSFET IRF530 (ou équivalent). Le moteur est un petit moteur à courant continu de 3V à 24V. Il faudra bien entendu adapter la tension d'alimentation externe à la tension de fonctionnement du moteur. Enfin une diode et un condensateur assurent la protection du transistor et la réduction des parasites.

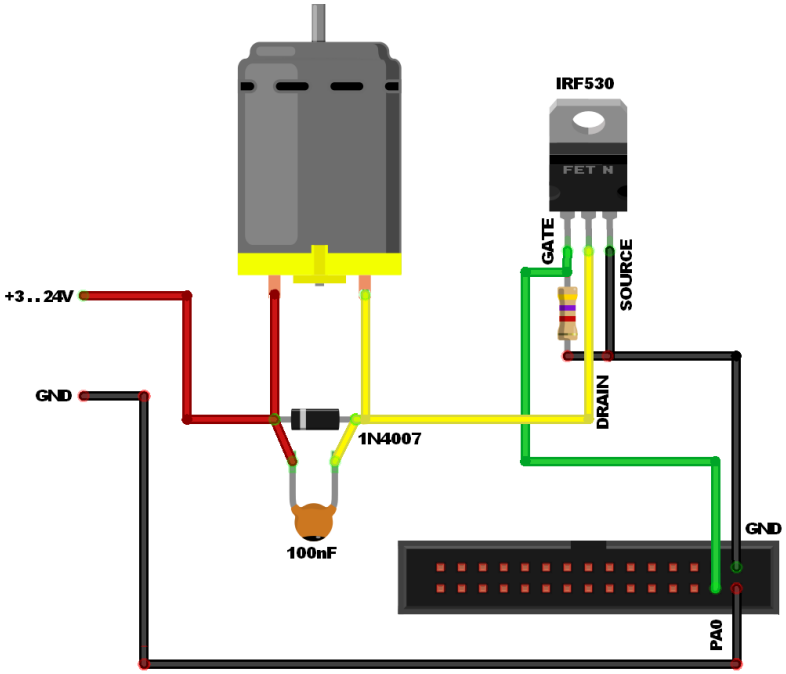


Figure 25: Contrôle d'un moteur à courant continu

Le contrôle du moteur nécessite de piloter rapidement la sortie PA0. Le programme est donc écrit intégralement en assembleur. Le moteur est contrôlé à l'aide des touches fléchées du clavier et la barre d'espace.

Le programme doit être implanté en mémoire à partie de l'adresse $5000.

```
OUTTX       .EQU $E7A8          ; LDX = adresse chaîne (terminée par null)
OUTCA       .EQU $F9C6          ; LDAA = code ascii
SCANK       .EQU $F883          ; <= LDAA = code ascii de la touche
                                ; enfoncée Z = 0 aucun touche
CURSOR      .EQU $3280          ; LDD = y,x du curseur

MOTOR_OFF_PIN     .EQU $00
MOTOR_ON_PIN      .EQU $01
PORTA       .EQU $BF60
```

```
        .ORG $5000

        LDAA #$8B               ; PORTA en sortie, PORTB et PORTC en entrée
        STAA $BF63

        LDAA #MOTOR_OFF_PIN
        STAA PORTA

        LDAA #$08
        STAA TimOn
        LDAA #$08
        STAA TimOff

        JSR $FBD6               ; efface écran
Start
        LDD #$0000
        STD CURSOR
        LDX #TextMotor-1
        JSR OUTTX

MoteurMove
        LDD #$0B09              ; mouvement du moteur
        STD CURSOR
        LDAA Marche
        CMPA #01
        BEQ MarcheOn
        LDX #TextOFF-1
        JSR OUTTX
        JMP FinCycle

MarcheOn
        LDX #TextON-1
        JSR OUTTX
        LDAA #MOTOR_ON_PIN
        STAA PORTA

        LDD #$0709
        STD CURSOR
        LDAA TimOn
        JSR HPRINT

        LDD #$0717
        STD CURSOR
        LDAA TimOff
        JSR HPRINT

        LDAB TimOn
        JSR Timing
        LDAA #MOTOR_OFF_PIN
        STAA PORTA
        LDAB TimOff
FinCycle
        JSR Timing
        JMP MoteurMove
```

```
Timing
        PSHB
        JSR SCANK
        BEQ PasDroite
        CMPA #$20
        BNE PASon
        LDAA Marche
        EORA #$01
        STAA Marche
PASon
        CMPA #$08
        BNE PasGauche
        DEC TimOn
PasGauche
        CMPA #$09
        BNE PasDroite
        INC TimOn
PasDroite
        CMPA #$0A
        BNE Pasbas
        DEC TimOff
Pasbas
        CMPA #$0B
        BNE PasHaut
        INC TimOff
PasHaut
        ;LDAB Tim
        PULB
Tempo
        LDX #$00FF
Tmpz
        DEX
        BNE Tmpz
        DECB
        BNE Tempo
        RTS

TextMotor
        .DB "CONTROLE D'UN MOTEUR BASSE TENSION"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "UTILISEZ LES FLECHES POUR MODIFIER"
        .DB 13
        .DB "LE CYCLE DU MOTEUR ON/OFF"
        .DB 13
        .DB "ESPACE= ARRETER OU DEMARRER LE MOTEUR"
        .DB 13,13
        .Db "DUREE ON:    DUREE OFF:"
        .DB 0

TextON
        .DB "MOTEUR ON "
        .DB 0
```

```
TextOFF
        .DB "MOTEUR OFF"
        .DB 0

HPRINT:
        PSHA                    ; print  HEXA number
        LSRA
        LSRA
        LSRA
        LSRA
        JSR Hex2Asc
        JSR OUTCA
        PULA
        JSR Hex2Asc
        JSR OUTCA
        RTS

Hex2Asc:
        ANDA #$0F               ; converti hexa en ascii
        CMPA #$0A
        BPL LetAsc
        ADDA #$30
        RTS
LetAsc
        ADDA #$30+07
        RTS

WorkingArea
        .DB 0,0,0,0,0,0

TimOn
        .DB 00
TimOff
        .DB 00

Marche
        .DB 01

        .END
```

Exercice 11 (MOTEUR PAS A PAS)

But : Piloter un moteur pas à pas (28BYJ-48)

Cet exercice présente la méthode pour piloter un moteur pas à pas avec la carte 24 E/S . La commande du moteur nécessite un driver et une alimentation externe (chargeur USB par exemple). En général ces moteurs pour Arduino sont vendus avec une carte driver équipée d'un ULN2003 (inverseurs de puissance à collecteurs ouverts).

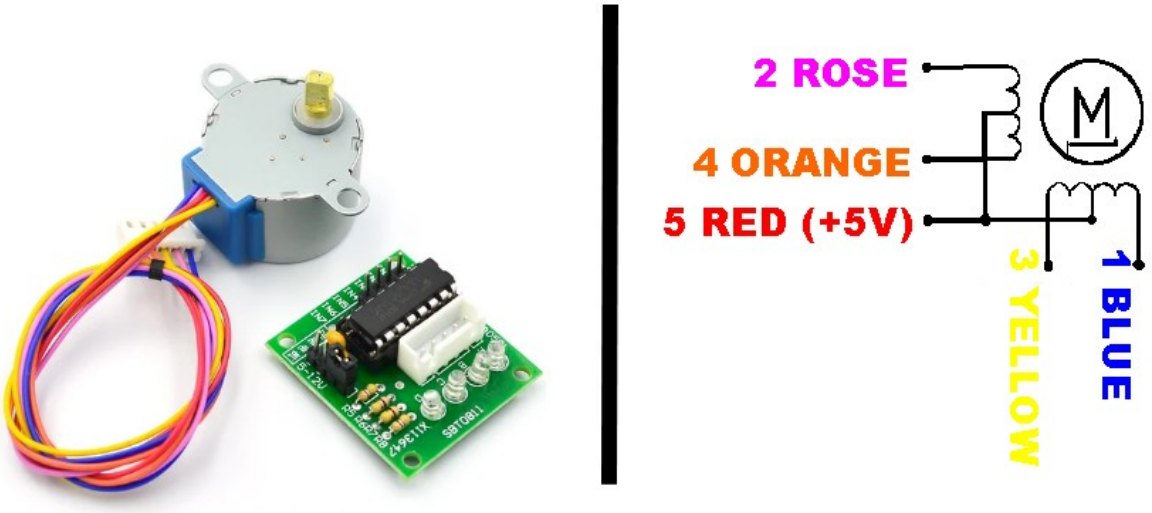


Figure 26: Moteur pas à pas et son driver

Fonctionnement du moteur

Le moteur **28BYJ-48** est un moteur unipolaire, il comporte donc 5 fils de connexions (4 demi-bobines et un fil commun). Ce moteur est équipé d'un réducteur si bien que l'axe de sortie nécessite 4096 pas pour faire un tour. Sa commande consiste à commander les 4 demi-bobines en respectant une séquence bien précise.

Le gros avantage d'un moteur pas à pas par rapport à un moteur à courant continu c'est qu'il est très facile de faire varier sa vitesse tout en connaissant précisément sa position angulaire. De même, lorsque le moteur est arrêté, il peut maintenir sa position en exerçant un couple résistant important. Mais en contrepartie, un moteur pas à pas ne tourne pas aussi vite qu'un moteur à courant continu et avance par petits sauts ce qui se traduit par des vibrations transmises à la structure porteuse.

Il existe 2 modes de commande des moteurs pas à pas : la commande en pas entiers et la commande en demi-pas. C'est cette dernière qui est mise en œuvre ici. Elle permet notamment d'augmenter la précision angulaire en doublant le nombre de pas du moteur.

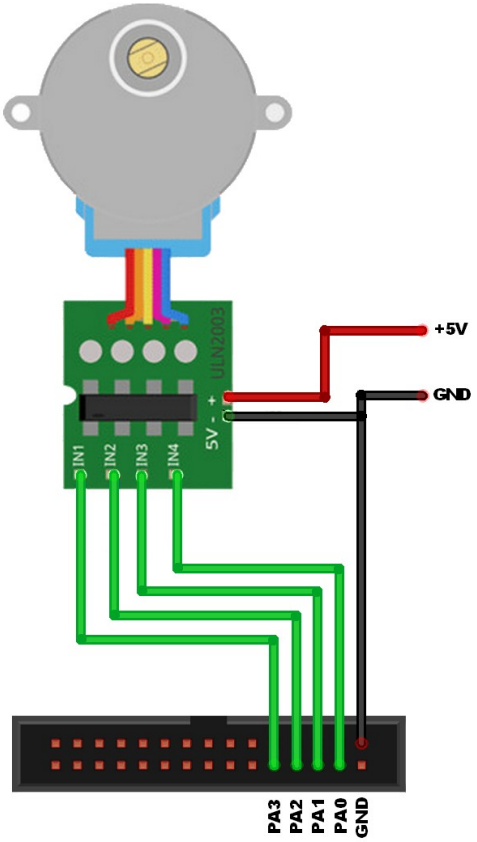


Figure 27: Branchement du moteur pas à pas

La carte de commande utilise un ULN2003, donc l'alimentation d'une bobine se fait en appliquant un '1' à l'entrée concernée.

Le **Tableau 8** donne la séquence de commande des bobines du moteur pour obtenir une rotation de son axe dans le sens antihoraire.

Rotation sens antihoraire (un '1' alimente la bobine)									
Commande	Bobine alimentée	Pas 1	Pas 2	Pas 3	Pas 4	Pas 5	Pas 6	Pas 7	Pas 8
IN1	B1- (orange)	1	0	0	0	0	0	1	1
IN2	B2+ (jaune)	0	0	0	0	1	1	1	0
IN3	B1+ (rose)	0	0	1	1	1	0	0	0
IN4	B2- (bleu)	1	1	1	0	0	0	0	0
Valeur PORTA		9	1	3	2	6	4	12	8

Tableau 8: Commande du moteur en sens antihoraire

La dernière ligne du tableau donne les valeurs à écrire sur le PORTA pour commander la rotation du moteur.

De la même façon, le **Tableau 9** donne la séquence de commande des bobines du moteur pour obtenir une rotation de son axe dans le sens horaire.

Rotation sens horaire (un '1' alimente la bobine)									
Commande	Bobine alimentée	Pas 1	Pas 2	Pas 3	Pas 4	Pas 5	Pas 6	Pas 7	Pas 8
IN1	B1- (orange)	1	1	0	0	0	0	0	1
IN2	B2+ (jaune)	0	1	1	1	0	0	0	0
IN3	B1+ (rose)	0	0	0	1	1	1	0	0
IN4	B2- (bleu)	0	0	0	0	0	1	1	1
Valeur PORTA		8	12	4	6	2	3	1	9

Tableau 9: Commande du moteur en sens horaire

La dernière ligne du tableau donne les valeurs à écrire sur le PORTA pour commander la rotation du moteur.

Pour faire tourner le moteur, il suffit d'envoyer la succession des valeurs sur le PORTA. Cette commande peut se faire en BASIC ou en assembleur. L'envoi des valeurs ne doit pas se faire trop vite sinon le moteur n'aura pas le temps de réagir et perdra des pas. Cela se traduit par une rotation désordonnée du moteur. En BASIC ce n'est pas un problème car le langage est assez lent, mais en assembleur il faudra impérativement ajouter une boucle de temporisation pour éviter ce problème.

Le moteur est contrôlé à l'aide des touches fléchées du clavier et la barre d'espace.

Le programme doit être implanté en mémoire à partie de l'adresse $5000.

```
PORTA.EQU $BF60
PORTB.EQU $BF61
PORTC.EQU $BF62
CTRL        .EQU $BF63

OUTTX.EQU $E7A8          ; LDX = adresse chaine (terminé par null)
OUTCA.EQU $F9C6          ; LDAA = code ascii
SCANK.EQU $F883          ; <= LDAA = code ascii de la touche enfoncée Z = 0 aucun
touche
CLS         .EQU $FBD6          ; efface l'écran
CURSOR      .EQU $3280          ; LDD = y,x du curseur

        .ORG $5000

        LDAA #$80           ; place les ports A, B et C en sorties
        STAA CTRL
        LDAA #$00           ; met le moteur a l'arrêt
        STAA PORTA
        LDAA #00            ; Sens initiale horaire =0
        STAA Sens
```

```
        JSR CLS
Start
        LDD #$0000
        STD CURSOR
        LDX #TextStepper-1
        JSR OUTTX

Move
        JSR SCANK
        CMPA #8
        BNE PasGauche
        LDAA Sens
        EORA #01
        STAA Sens
PasGauche
        CMPA #9
        BNE PasDroite
        LDAA Sens
        EORA #01
        STAA Sens
PasDroite
        LDAA Sens
        CMPA #0
        BNE NotHor
        JSR Horaire
        JMP Move
NotHor
        JSR AntiHoraire
        JMP Move


; faire tourner le moteur dans le sens antihoraire
Horaire
        LDX #Datas
        LDAB #08
TourneH
        LDAA 0,X
        STAA PORTA
        INX
        PSHX
                JSR Tempo
        PULX
        DECB
        BNE TourneH
        RTS
```

```
; faire tourner le moteur dans le sens horaire
AntiHoraire
        LDX #Datas+7
        LDAB #08
TourneA
        LDAA 0,X
        STAA PORTA
        DEX
        PSHX
                JSR Tempo
        PULX
        DECB
        BNE TourneA
        RTS

Tempo                   ; ATTENTION ne pas reduire cette valeur
        LDX #$008F      ; valeur critique !!!!
Timing                  ; sous peine d'endommager votre moteur, ULN ....
        DEX
        BNE Timing
        RTS

TextStepper
        .DB "CONTROLE D'UN MOTEUR PAS A PAS"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "UTILISEZ LES TOUCHES GAUCHE ET"
        .DB 13
        .DB "DROITE POUR CHANGER LE SENS DE"
        .DB 13
        .DB "ROTATION DU MOTEUR."
        .DB 0

Datas
        .DB 8,12,4,5,2,3,1,9

Sens
        .DB 00
Step
        .DB 10

WorkingArea
        .DB 00,00,00,00,00,00

        .END
```

Ce 2ème programme permet de régler plus finement le mouvement du moteur en ajoutant la possibilité de régler ‘la vitesse’

```
PORTA.EQU $BF60
PORTB.EQU $BF61
PORTC.EQU $BF62
CTRL  .EQU $BF63


OUTTX.EQU $E7A8          ; LDX = adresse chaine (terminé par null)
OUTCA.EQU $F9C6          ; LDAA = code ascii
SCANK.EQU $F883          ; <= LDAA = code ascii de la touche enfoncée Z = 0 aucun
touche
CLS          .EQU $FBD6          ; efface ecran
CURSOR       .EQU $3280          ; LDD = y,x du curseur


        .ORG $4FF9
        .DB $01,$50,$00,$50,$00,$00,$00


        .ORG $5000

        LDAA #$80           ; place les ports A, B et C en sorties
        STAA CTRL

        LDAA #$00           ; met le moteur a l'arret
        STAA PORTA

        LDAA #00            ; Sens initiale horaire =0
        STAA Sens

        LDAA #10            ; pas initiale du moteur
        STAA Step


        JSR CLS
Start
        LDD #$0000
        STD CURSOR
        LDX #TextStepper-1
        JSR OUTTX



Move
        LDD #$0806
        STD CURSOR
        LDAA Step
        JSR HPRINT
        LDAA #$00           ; met le moteur a l'arret
        STAA PORTA
WaitKey
        JSR SCANK
        BEQ WaitKey
```

```
PORT2       .EQU $03
            LDAB PORT2          ; on lit l'état de shift
            ANDB #$02
            CMPB #$02
            BEQ Keying
            LDAB #$80
            ABA
Keying
        CMPA #8
        BNE PasGauche
            LDAA #$01
            STAA Sens
PasGauche
        CMPA #9
        BNE PasDroite
            LDAA #$00
            STAA Sens
PasDroite
        CMPA #$0B
        BNE PasHaut
                INC Step
                JMP Move

PasHaut
        CMPA #$0A
        BNE PasBas
                DEC Step
                JMP Move
PasBas
        CMPA #$8B
        BNE PasShaut
                LDAA Step
                ADDA #$10
                STAA Step
                JMP Move
PasShaut
        CMPA #$8A
        BNE PasSbas
                LDAA Step
                SUBA #$10
                STAA Step
                JMP Move
PasSbas

        LDAA Sens
        CMPA #00
        BNE NotHor
            JSR Horaire
            JMP Move
NotHor
        JSR AntiHoraire
        JMP Move


; faire tourner le moteur dans le sens antihoraire
Horaire
        LDAA Step
```

```
        STAA WorkingArea
PasPasHoraire
        LDX #Datas
        LDAB #08
TourneH
        LDAA 0,X
        STAA PORTA
        INX
        PSHX
                JSR Tempo
        PULX
        DECB
        BNE TourneH
        LDAA WorkingArea
        DECA
        STAA WorkingArea
        BNE PasPasHoraire
        RTS

; faire tourner le moteur dans le sens horaire
AntiHoraire
        LDAA Step
        STAA WorkingArea
PasPasAntiHoraire
        LDX #Datas+7
        LDAB #08
TourneA
        LDAA 0,X
        STAA PORTA
        DEX
        PSHX
                JSR Tempo
        PULX
        DECB
        BNE TourneA
        LDAA WorkingArea
        DECA
        STAA WorkingArea
        BNE PasPasAntiHoraire
        RTS


Tempo                   ; ATTENTION ne pas reduire cette valeur
        LDX #$008F      ; valeur critique !!!!
Timing
        DEX
        BNE Timing
        RTS




TextStepper
        .DB "CONTROL D'UN MOTEUR PAS A PAS"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
```

```
        .DB "UTILISEZ LES FLECHES GAUCHE ET"
        .DB 13
        .DB "DROITE POUR CHANGER LE SENS DE"
        .DB 13
        .DB "ROTATION DU MOTEUR."
        .DB 13
        .DB "HAUT ET BAS POUR CHANGER LE PAS."
        .DB 13
        .DB "SHIFT +HAUT ET BAS = PAS+10"
        .DB 13
        .DB "PAS :"
        .DB 0


Datas
        .DB 8,12,4,5,2,3,1,9

Sens
        .DB 00

Step
        .DB 10

WorkingArea
        .DB 00,00,00,00,00,00


;-------------------------------------
;       print  HEXA number
HPRINT:
                PSHA
        LSRA
        LSRA
        LSRA
        LSRA
        JSR Hex2Asc
        JSR OUTCA
                PULA
        JSR Hex2Asc
        JSR OUTCA
        RTS

;-------------------------------------
;       converti hexa en ascii
Hex2Asc:
        ANDA #$0F
        CMPA #$0A
        BPL LetAsc
        ADDA #$30
        RTS
LetAsc
        ADDA #$30+07
        RTS


        .END
```

Exercice 12 (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).

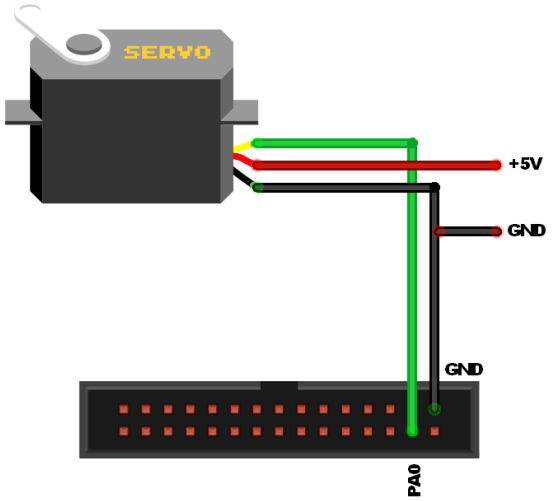


Figure 28: Commande d'un servomoteur

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.

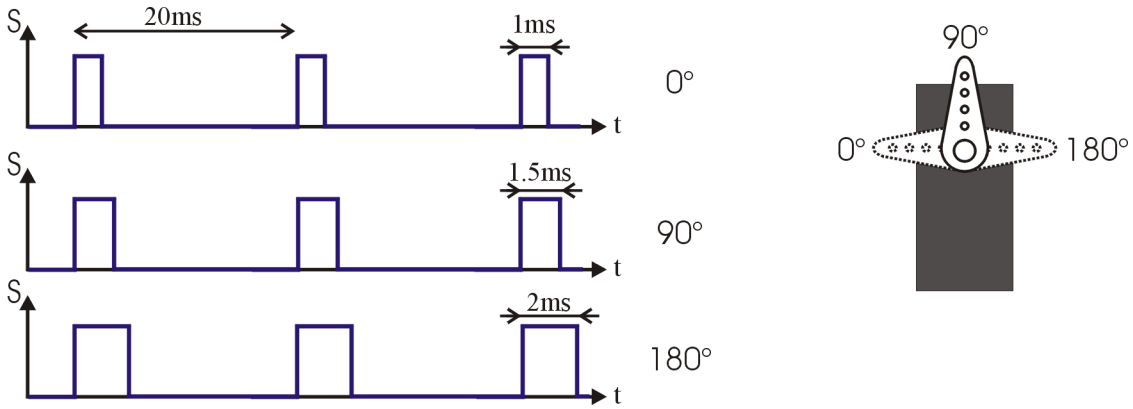


Figure 29: Commande d'un servomoteur analogique

Le programme doit être implanté en mémoire à partir de l'adresse $5000.

```
PORTA.EQU $BF60
PORTB.EQU $BF61
PORTC.EQU $BF62
CTRL        .EQU $BF63

OUTTX.EQU $E7A8         ; LDX = adresse chaîne (terminé par null)
OUTCA.EQU $F9C6         ; LDAA = code ascii
SCANK.EQU $F883         ; <= LDAA = code ascii de la touche
                        ; enfoncée Z = 0 aucun touche
CLS         .EQU $FBD6  ; efface l'écran
CURSOR      .EQU $3280  ; LDD = y,x du curseur

        .ORG $5000

        LDAA #$80       ; place les ports A, B et C en sorties
        STAA CTRL
        LDAA #$00       ; met le moteur a l'arret
        STAA PORTA
        LDAA #96        ; direction initiale
        STAA Direction
        LDAA #10        ; pas de déplacement initiale
        STAA STEP

        JSR CLS
Start
        LDD #$0000
        STD CURSOR
        LDX #TextServo-1
        JSR OUTTX

CommandServo
        LDD #$090B
        STD CURSOR
        LDAA Direction
        JSR HPRINT
        LDD #$0917
        STD CURSOR
        LDAA STEP
        JSR HPRINT

        JSR SCANK
        CMPA #08
        BNE PasGauche
        LDAA Direction
        ADDA STEP
        STAA Direction
        JMP FIN

PasGauche
        CMPA #09
        BNE PasDroite
        LDAA Direction
        SUBA STEP
        STAA Direction

PasDroite
```

```
        CMPA #'O'
        BNE PasO
        DEC STEP

PasO
        CMPA #'P'
        BNE PasP
        INC STEP
PasP
FIN
        JSR PulseServo
        JMP CommandServo

PulseServo                  ; 20 ms
        LDX #64*10                  ; 3µs
        STX WorkingArea             ; 4
        LDAA #$01                   ; 2
        STAA PORTA                  ; 3
        LDX Direction-1             ; 3
MS1
        MUL                         ; 10
        DEX                         ; 3
        BNE MS1                     ; 3
; total cycles =  16*X+8+
        LDAA #$00                   ; 2
        STAA PORTA                  ; 3
        LDX #WorkingArea            ; 3
Timing20ms
        ;MUL                        ; 10
        DEX                         ; 3
        BNE Timing20ms              ; 3
        RTS                         ; 5

;       print  HEXA number
HPRINT:
        PSHA
        LSRA
        LSRA
        LSRA
        LSRA
        JSR Hex2Asc
        JSR OUTCA
        PULA
        JSR Hex2Asc
        JSR OUTCA
        RTS

;       converti hexa en ascii
Hex2Asc:
        ANDA #$0F
        CMPA #$0A
        BPL LetAsc
        ADDA #$30
        RTS
LetAsc
        ADDA #$30+07
        RTS
```

```
TextServo
        .DB "CONTROLE D'UN SERVO MOTEUR"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13
        .DB "UTILISEZ LES TOUCHES GAUCHE ET"
        .DB 13
        .DB "DROITE POUR CONTROLER LE SERVO"
        .DB 13
        .DB "MOTEUR."
        .DB 13
        .DB "O ET P POUR CONTROLER LE PAS."
        .DB 13,13,13
        .DB "POSITION :       PAS :"
        .DB 0

Padding
        .DB 00
Direction
        .DB 00
STEP
        .DB 00

WorkingArea
        .DB 00,00,00,00,00,00
        .END
```

Exercice 13 (MATRICE DE LEDS)

But : Piloter une matrice de leds

Cet exercice présente la méthode pour pilote 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).

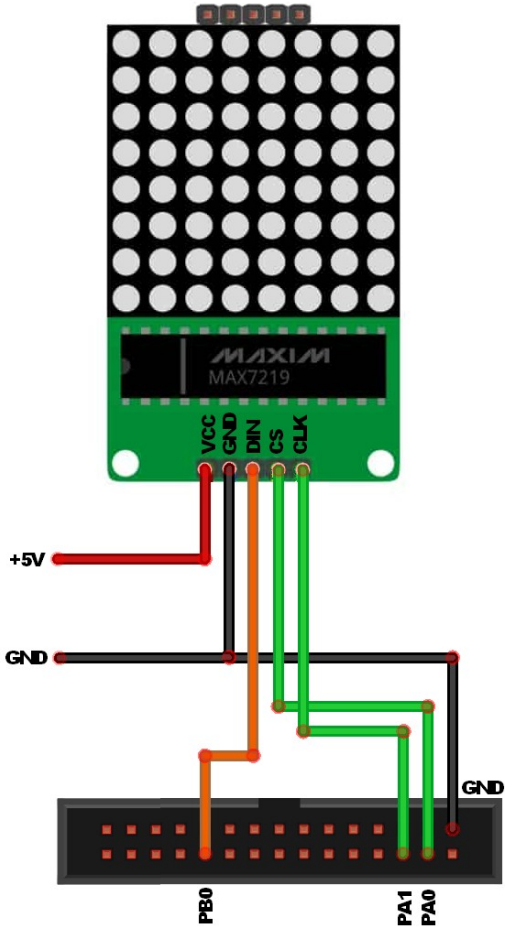


Figure 30: Contrôle d'une matrice de leds

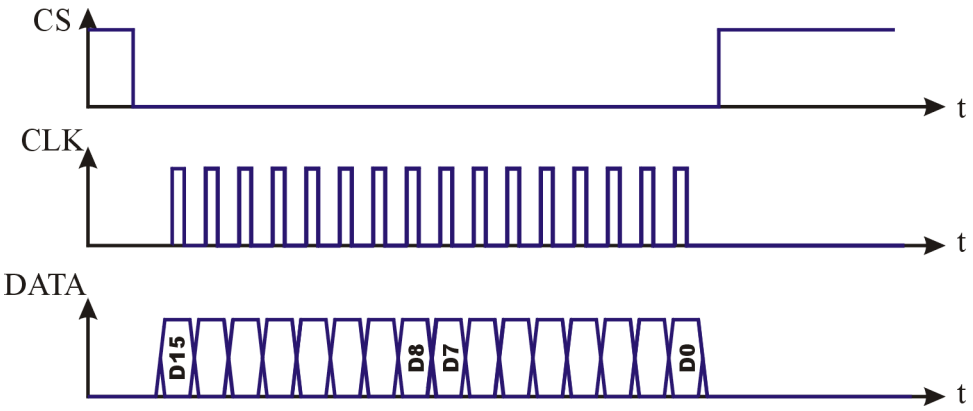


Figure 31: Signaux de contrôle de la matrice

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 montrent les signaux à utiliser. La communication se fait en série par blocs de 16 bits.

Le programme est à éditer en mémoire à partir de l‘adresse $5000.

```
CLK_HIGH_PIN    .EQU $01
CS_HIGH_PIN     .EQU $02
CS_LOW_PIN      .EQU $00
CLK_LOW_PIN     .EQU $00

PORTA           .EQU $BF60
PORTB           .EQU $BF61
PORTC           .EQU $BF62
CTRL            .EQU $BF63

        .ORG $5000

        LDAA #$89           ; PORTA et PORTB en sortie, PORTC en entrée
        STAA $BF63  ;
        LDAA #CS_HIGH_PIN+CLK_HIGH_PIN
        STAA PORTA
        JSR $FBD6           ; efface l’écran
Start
        LDD #$0000
        STD $3280           ; LDD = y,x du curseur
        LDX #TextMatrix-1
        JSR $E7A8           ; LDX = adresse chaîne


        LDX #DataInit       ; initialisation du MAX7219
Init7219
        LDAA #CS_LOW_PIN+CLK_HIGH_PIN
        STAA PORTA
        LDAA 0,X
        INX
        PSHX
        JSR Shifting8
        PULX
        LDAA 0,X
        INX
        PSHX
        JSR Shifting8
        LDAA #CS_HIGH_PIN+CLK_HIGH_PIN
        STAA PORTA
        PULX
        CPX #FinInit
        BNE Init7219

BoucleAff
        LDX #Font           ; affichage 6502MAN
        STX WorkingArea+2
NextCar
        LDAA #01
        STAA WorkingArea
Affichage
        LDX WorkingArea+2
```

```
        CPX #FinFont
        BEQ BoucleAff
        LDAA #CS_LOW_PIN+CLK_HIGH_PIN
        STAA PORTA
        LDAA WorkingArea
        JSR Shifting8
        LDX WorkingArea+2
        LDAA 0,X
        INX
        STX WorkingArea+2
        JSR Shifting8
        LDAA #CS_HIGH_PIN+CLK_HIGH_PIN
        STAA PORTA

        LDAA WorkingArea
        INCA
        STAA WorkingArea
        CMPA #$09
        BEQ PausingFrame
        BNE Affichage
PausingFrame
        JSR Wait
        JMP NextCar

Wait
        LDAB #$02
Tempz
        LDX #$FFFF
Tempo
        DEX
        BNE Tempo
        DECB
        BNE Tempz
        RTS


; envoi l'octet contenu dans A bit par bit
; en commençant par le MSB.
;
Shifting8                       ;  A = octets à envoyer
        PSHA
        LDX #$0008
Shift
        LDAA #CS_LOW_PIN+CLK_HIGH_PIN
        STAA PORTA
        PULA
        LDAB #$00
   ASLA
        ADCB #$00
   STAB PORTB
        PSHA
        LDAA #CS_LOW_PIN+CLK_LOW_PIN
        STAA PORTA
   DEX
        BNE Shift
        LDAA #CS_LOW_PIN+CLK_HIGH_PIN
        STAA PORTA
```

```
        PULA
        RTS

TextMatrix
        .DB "CONTROLE D'UNE MATRICE DE LEDS"
        .DB 13
        .DB "AVEC LA CARTE 24 ENTREES SORTIES"
        .DB 13,13,0

Font
        .DB $00,$FF,$00,$FF,$00,$FF,$00,$FF
        .DB $FF,$00,$FF,$00,$FF,$00,$FF,$00

        .DB $7E,$80,$80,$FC,$82,$82,$82,$7c   ;6
        .DB $7E,$80,$80,$FC,$02,$02,$02,$7C   ;5
        .DB $7E,$C1,$A1,$91,$89,$85,$83,$7E   ;0
        .DB $3E,$42,$04,$08,$10,$20,$40,$7E   ;2
        .DB $00,$C3,$A5,$99,$81,$81,$81,$81   ;M
        .DB $7E,$81,$81,$FF,$81,$81,$81,$81   ;A
        .DB $00,$C1,$A1,$A1,$91,$89,$85,$83   ;N
FinFont

DataInit .DB $09, $00, $0A, $01, $0B, $07, $0C, $01, $0F, $00
FinInit

; Decode-Mode Register  (Address (Hex) = X9)
;       No decode for all lines       = $00
;       ..............................

; Intensity Register (Address (Hex) = XA)
;       Minimum (01/32)               = $00
;       ..........................
;       Maximum (31/32)               = $0F

; Scan-Limit Register (Address (Hex) = XB)
;       Display line 1          = $00
;       Displey line 1 to 2     = $01
;       ..........................
;       Display line 1 to 8     = $07

; Shutdown Register (Address (Hex) = XC)
;       Shutdown mode                 = $00
;       Normal opération      = $01

; Display-Test Register (Address (Hex) = XF)
;       Normal opération      = $00
;       Display test Mode     = $01

WorkingArea
        .DB 0,0,0,0,0,0
        .END
```

Question

Comment faut-il modifier le programme pour piloter 3 matrices simultanément ?