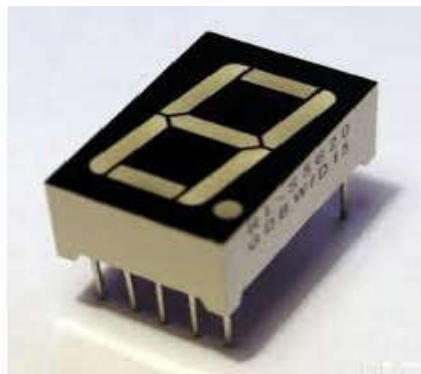


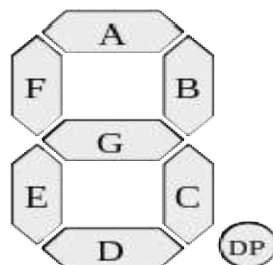
LAB 7

Le but du LAB : comment décoder un afficheur BCD7SEG à travers un LUT (LOOK-UP TABLE), en plus comment programmer un LUT par l'assembleur de MID-RANGE.

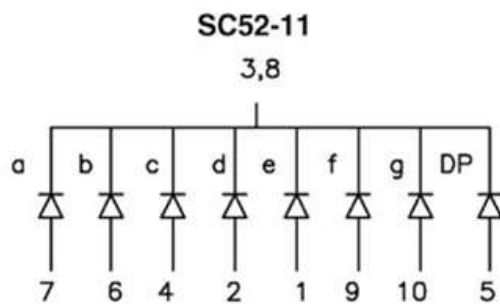
Avant de commencer, il faut étudier d'abord le composant électronique l'afficheur BCD7SEG (Binary Coded Decimal 7 Segments). La figure suivante présente un afficheur BCD7SEG :



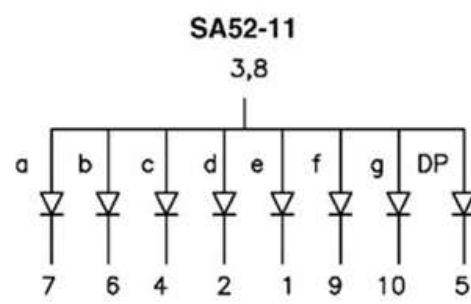
Un afficheur BCD7SEG est un ensemble de LED notée A, B, C, D, E, F, G, DP comme indique la figure suivante :



Il existe de types, les afficheurs à anode commune et les afficheurs à cathode commune :



Afficheur 7 segments
cathode commune CC



Afficheur 7 segments
anode commune AC

Comment afficher les chiffres de 0 jusqu'à 9 sur ce genre d'afficheur ?

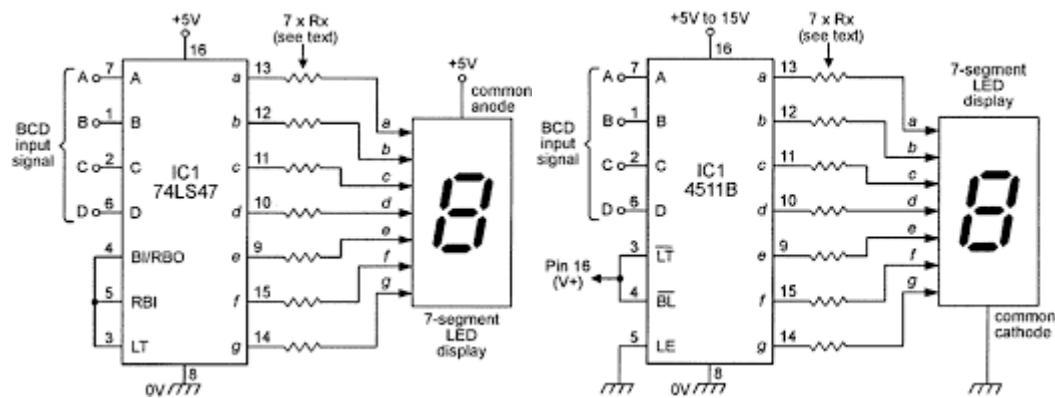
Tout simplement, on allume les segments (LED) correspondants pour afficher le chiffre désiré.

Truth Table for 7 Segment Decoder

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

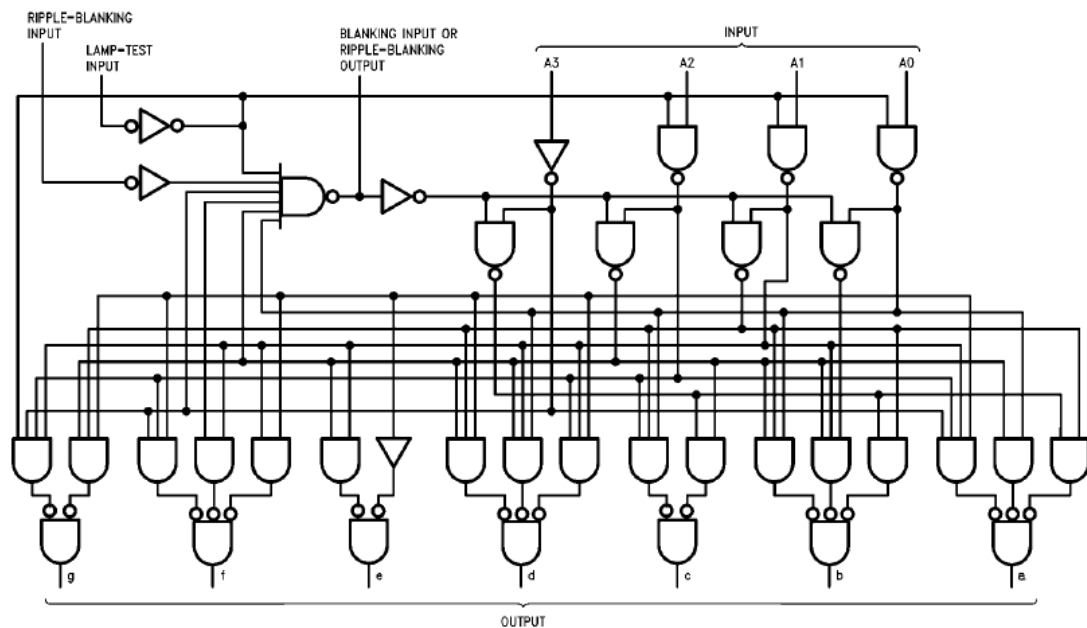
Dans le monde des circuits logiques de la série TTL 74 et la série CMOS 4x, on trouve quelques circuits intégrés spécialisés en décodage de ce genre d'afficheur comme : 7448, 7447, 7446 en TTL et 4543, 4511, 4056 en CMOS.

Les figures suivantes montrent comment connecter le décodeur avec d'afficheur :



Que remarquez-vous ?

D'après la table de vérité de notre décodeur, on peut par l'aide de la table de karnaugh de sortir les différentes équations logiques de chaque segment. Le logigramme suivant présente la logique interne du circuit 74LS48



Tout le monde rappelle bien le LAB3, qu'on a pris avec, comment traduire un circuit logique en algorithme puis transformer en assembleur !

La logique interne de 74LS48 donne une image sur le calcul menstuel à faire par notre μC pour arriver décoder un chiffre !!!

Est-ce qu'il y a un autre moyen ou issu pour ne pas baigner dans la merde de calcul ?

Absolument, oui ! C'est à travers les **LUT** (LOOK-UP TABLE) ! Une **table de correspondance** (aussi appelé en anglais ***par Look-up Table***) est un terme informatique et électronique désignant une liste d'association de valeurs. Elle se comporte sur le même modèle qu'une **table de vérité** désignant sa sortie de manière unique en fonction de ses entrées et du contenu de la table. Il s'agit d'une **structure de données** stockée en mémoire, employée pour **remplacer un calcul par une opération plus simple, c'est la lecture d'une case mémoire d'un tableau**. Le gain de vitesse peut être significatif, car rechercher une valeur en mémoire est souvent plus rapide qu'effectuer un calcul important.

En programmation, un **Look-up Table** est un **tableau de constantes** situé dans la mémoire, donc nécessite le **mode d'adressage indirect** pour consulter (lire) ses éléments.

Très bien !

Le mode d'adressage indirect ! C'est pour lire/écrire des données volatiles dans l'espace DATA_I/O de notre MID-RANGE et le LUT d'après la définition est un tableau de constantes !

C'est quoi cette histoire ?

C'est de gaspillage de l'espace DATA_I/O pour créer un tableau de constantes qui ne change jamais durant l'exécution du code ! **Comment faire ?**

La logique me dit que l'espace CODE présente l'espace idéal pour mettre notre LUT. **Mais, comment déclarer un LUT dedans ?**

On a posé la même question à MICROCHIP, et cette dernière m'a répondu comme suit :

Soit le problème suivant : faire le programme qui calcul et indique sur une LED relié à RB0 la fonction majorité des pattes RA0, RA1 et RA2

La solution : on commence d'abord par la table de vérité de la fonction majorité de 3 bits.

Index	Inputs			Output
	A	B	C	MAJ
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Donc, MAJ = {0, 0, 0, 1, 0, 1, 1, 1} est un tableau de 8 constantes.

Algorithme

```
; Déclaration de MAJ
MAJ = {0, 0, 0, 1, 0, 1, 1, 1};

While (True)
    // Lecture des bits RA0, RA1 et RA2 seulement
    Index ← PORTA and B'00000111'; // masqué les bits non utiles
    // Lecture de la valeur correspondante à [RA2, RA1, RA0]
    // avec mise à jour de la patte RB0
    RB0 ← MAJ[Index];
End While
```

L'assembleur :

```
        ; INIT PCLATH BY 0 (SELECT PAGE0)
        CLRF PCLATH
        ; READ PORTA
        MOVF PORTA, W
        ; MASK BITS NOT USED AND SAVE ONLY RA2, RA1, RA0
        ANDLW B'00000111'
        ; READ MAJ TABLE
        CALL MAJ_LUT
        ; UPDATE RB0
        MOVWF PORTB
        GOTO $-4

;=====
; MAJ LUT SUBROUTINE
;=====
MAJ_LUT: ADDWF PCL, F           ; (ADDR : Z)
        RETLW 0x00 ; index : 0   ; (ADDR : Z+1)
        RETLW 0x00 ; index : 1   ; (ADDR : Z+2)
        RETLW 0x00 ; index : 2   ; (ADDR : Z+3)
        RETLW 0x01 ; index : 3   ; (ADDR : Z+4)
        RETLW 0x00 ; index : 4   ; (ADDR : Z+5)
        RETLW 0x01 ; index : 5   ; (ADDR : Z+6)
        RETLW 0x01 ; index : 6   ; (ADDR : Z+7)
        RETLW 0x01 ; index : 7   ; (ADDR : Z+8)
```

Analyser le code.

On suppose que les pattes RA0, RA1 et RA2 sont charger par {1, 1, 0}, donc :

Après « MOVF PORTA, W » le W charger par « XXXXX011 »

Après « ANDLW B'00000111' » le W charger par « 00000**011** », cette valeur correspond à l'index de la valeur 3.

Au moment d'appel par « CALL MAJ_LUT » :

Après « ADDWF PCL, F » {si PCL est la destination on va voir l'exécution de deux opérations}

- $PCL \leftarrow PCL_{NEXT} + W$
- $PCH \leftarrow PCLATH$

Donc, **un saut dynamique**.

Si on suppose que l'adresse de l'instruction « ADDWF PCL, F » est « Z », l'instruction suivante sera « Z+1 » et ainsi de suite.

Donc, « ADDWF PCL, F » présente un saut vers l'adresse $PC = [PCH : PCL] = [PCLATH : PCL_{NEXT} + W] = [0 : Z+1 + 3] = [0 : Z + 4] = Z + 4$.

« Z+4 » présente l'adresse de l'instruction « ADDWF PCL, F » plus 4 donc l'adresse de l'instruction en surbrillance jaune dans le code au-dessus, c'est « RETLW 0x01 ».

Après « RETLW 0x01 » le MID-RANGE revient au code appelant par une valeur de « 0x01 » dans le W.

Après « MOVWF PORTB » : la patte RB0 sera charger par la valeur correspondante à la fonction majorité de la combinaison « 011 ».

Et le cycle recommence par « GOTO \$-4 ».

A vous mes chers étudiants : analyser le code si RA2, RA1, RA0 = 100. (On principe la fonction majorité de cette combinaison donne 0, **vérifier**).

Question pour champion : quelle est la taille de LUT maximale qu'on peut déclarer par cette technique ?

Question pour champion : quelle est l'adresse qu'on peut avec déclarer un LUT de taille maximale ?

Vous remarquez que, le saut dynamique dépend directement par le contenu de registre W à travers la relation suivante : $PCL \leftarrow PCL_{NEXT} + W$, cette dernière est une opération sur 8bits, donc on va voir 256 valeur possible (min : 0, max : 255).

Si on prend l'exemple de la fonction majorité, pour déclarer son LUT, on a écrit un code de 9 instructions, de l'adresse Z jusqu'à l'adresse Z+8. Par analogie, l'adresse de la première instruction de LUT_{MAX} est Z et la dernière adresse est Z+255.

Donc, la taille max de LUT est : 255.

Et pour l'adresse qu'on peut avec déclarer ce LUT ; on peut distinguer facilement à partir de l'adresse de la dernière instruction $Z+255 = MAX_{8Bits} = 255$ s'implique que $Z=0$. Le $Z=0$, c'est pour PCL, donc l'adresse qu'il faut utiliser pour déclarer un LUT de 255 valeurs est un multiple de 256 {256, 512,...} ou à chaque 1/8 d'une page code ($2K/256 = 8$).

```
LUT:    ORG 0x200 ; adresse multiple de 0x100 (D'256')
        ADDWF PCL, F
        RETLW V1
        RETLW V2
        RETLW V3
        ...
        RETLW V255
```

Il existe une autre syntaxe proposé par Microchip pour éviter la séquence de RETLW par ce qu'elle est fastidieuse. Cette syntaxe se base sur l'emploi de la directive « **DT** », c'est l'acronyme de « **DEFINE TABLE** ».

```
LUT:    ADDWF PCL, F
        RETLW 0x01
        RETLW 0x20
        RETLW 0x3F
        RETLW 0xEA
```

```
LUT:    ADDWF PCL, F
        DT  0x01, 0x20, 0x3F, 0xEA
```

Après compilation, le compilateur MPASM remplace automatiquement la table défini par DT en séquence de RETLW.

=====

Maintenant, on revient à notre LAB :

Il me demande de faire un compteur libre M10 incrémente par une fréquence de 2Hz (500ms) avec l'affichage se fait sur un afficheur BCD7SEG-AC puis BCD7SEG-CC.

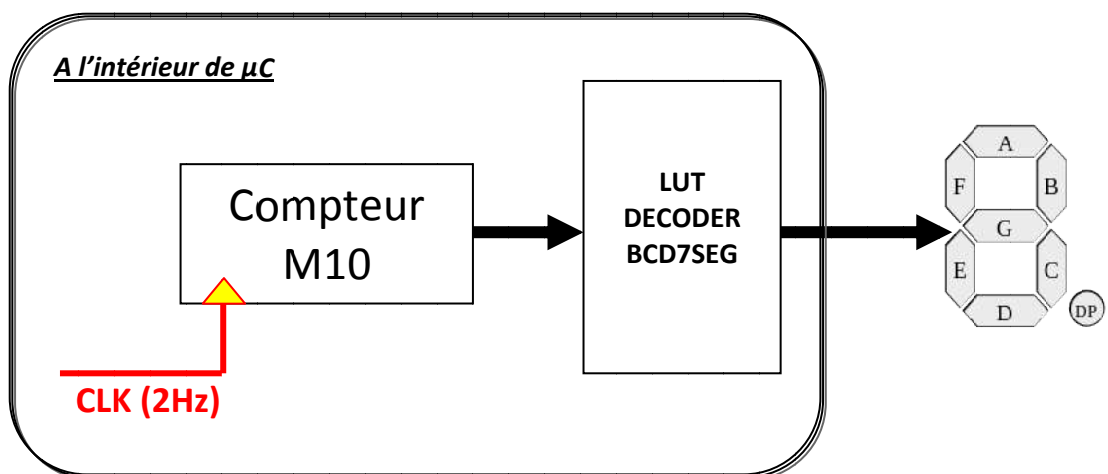
Quel est le cycle réaliser par un compteur M10 ?

0→1→2→3→4→5→6→7→8→9



Au moment du compteur arrive à 10, il revient à zéro et le cycle recommence encore une autre fois.

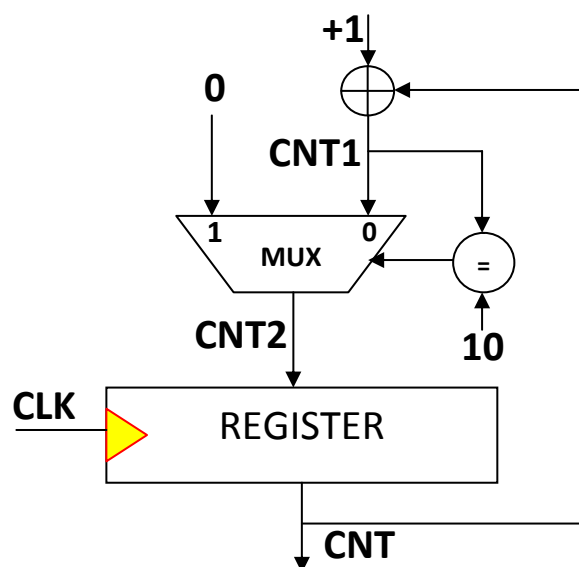
Synoptique du système à programmer :



En électronique, le compteur est un circuit logique séquentiel comme la bascule qu'on a émulée dans le LAB2, donc on va suivre les mêmes étapes pour programmer.

- Initialisation des variables de sortie et les **variables internes (au besoin)**.
- Boucler infiniment sur :
 - La mise à jour des sorties par le contenu des variables de sortie
 - La mise à jour des variables entrées par les entrées du système.
 - Réalisation du traitement entre les variables d'entrée et de sortie.

Le compteur, qu'est ce qu'il y a dedans ?



Comment transformer en algorithme ?

Algorithme :

```
; Initialisation
CNT ← 0 ;

While (True)
    ; Calcul des signaux internes
    CNT1 ← CNT+1;
    If (CNT1=10) then
        CNT2 ← 0;
    Else
        CNT2 ← CNT1;
    End If
    CNT ← CNT2;
    CALL DELAY(500ms);
End While
```

Comment modifier cet algorithme pour faire notre LAB ?

```

; Déclaration de LUT DECODER-BCD7SEG-CC
LUT = {7Eh,30h,6Dh,79h,33h,5Bh,5Fh,70h,7Fh,7Bh};

; Initialisation
CNT ← 0;

While (True)
    ; Mise à jour des sorties
    BCD7SEG ← LUT[CNT];
    ; Traitement du compteur
    CNT1 ← CNT+1;
    If (CNT1=10) then
        CNT2 ← 0;
    Else
        CNT2 ← CNT1;
    End If
    CNT ← CNT2;
    CALL DELAY(500ms);
End While

```

Attention : *au moment de traduire l’algorithme en assembleur, il faut ajouter la configuration du hardware.*

N.B : *ce n’est pas la peine de faire un délai de 500ms à un centième prêt, mais essayé de vous rapprocher vers le nombre de cycles désiré.*

Il existe une autre solution algorithmique plus optimale (moins de code et moins de variables).

```
; Déclaration de LUT DECODER-BCD7SEG-CC
LUT = {7Eh, 30h, 6Dh, 79h, 33h, 5Bh, 5Fh, 70h, 7Fh, 7Bh};

; Initialisation
CNT ← 0;

While (True)
    ; Mise à jour des sorties
    BCD7SEG ← LUT[CNT];
    ; Traitement du compteur
    CNT ← CNT+1;
    If (CNT=10) then
        CNT ← 0;
    End If
    CALL DELAY(500ms);
End While
```