Algorigramme / Microcontroleur: Compte-rendu du TP

1 Premier exemple:

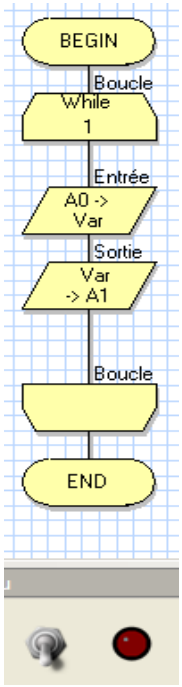
Commande d'une Led par un interrupteur à 2 positions

Algorithme:

Tant que 1

Lire : Var **<-** PORTA-0

Ecrire : Var -> PORTA-1



2 Mise en oeuvre du « ... Jusqu'à »

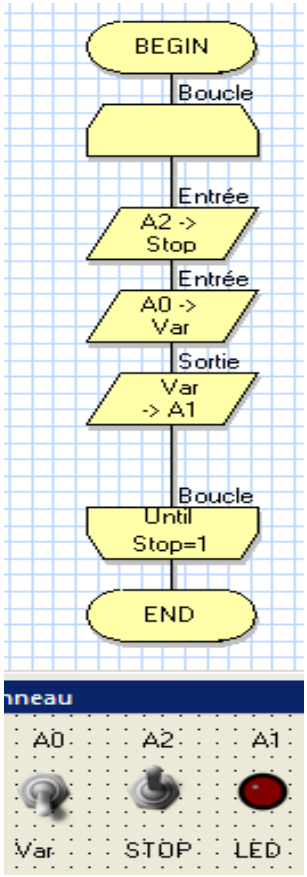
On **ajoute** au programme précédent un interrupteur qui servira à l'arrêt du programme:

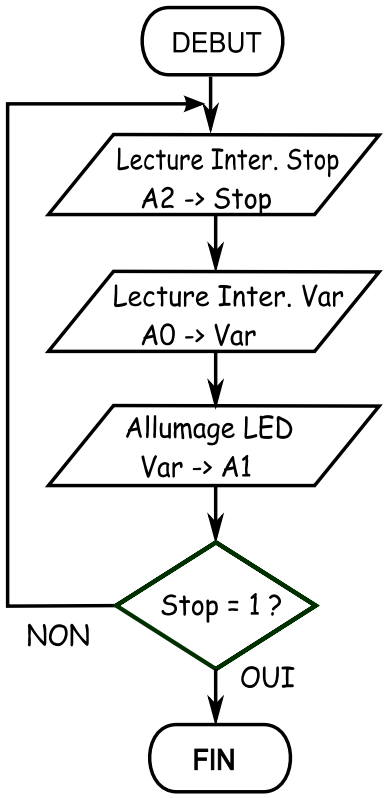
interrupteur Low = fonctionnement du programme

interrupteur High => arret

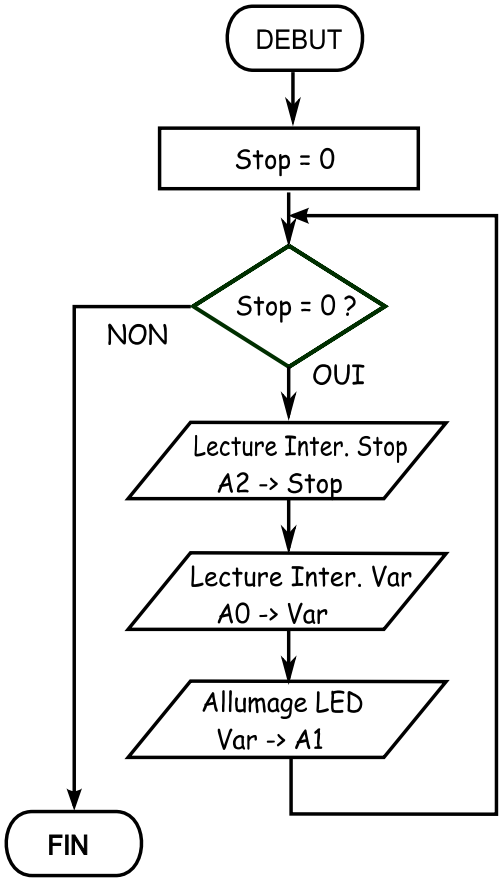
Une deuxième variable STOP est Nécessaire.

Algorigramme « normalisé » :





3 *Mise en oeuvre du « Tant que ... »*

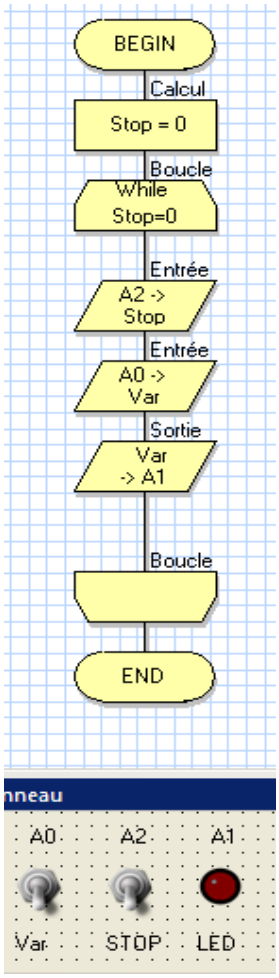


Le « test » intervient en début de boucle.

La qustion a été inversée par rapport à la structure jusqu'à : Tant que Stop n'a pas été activé, il faut exécuter la boucle.

Il faut initialiser la variable Stop avant le test.

En effet si Stop n'et pas égal à 0 en début de programme, la boucle ne sera jamais exécutée.



.

4 *Mise en oeuvre du « Si ... alors »*

On conserve le programme précédent (ou celui du chapitre2) et on veut réaliser **un compteur décimal.** L'affichage s'effectuera sur un afficheur 7 segments.

Cette fois le 1er interrupteur ne servira pas à allumer la Led, mais à incrémenter le compteur (compteur = compteur + 1) lorsqu'il passera de la position Basse à la position Haute.

Il faudra donc une nouvelle variable nommée « compteur » qui conservera la valeur du compteur.

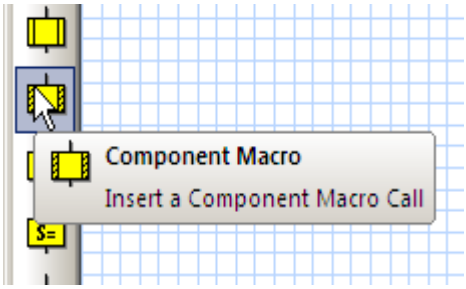
En comptage, nous voulons la suite 0 1 2 . . . . 9 0 1 2 . . etc .

Noter que si l'afficheur doit afficher un chiffre au delà de 9, il affiche 8 …ce que nous ne voulons pas.

Attention: il faut attendre que l'interrupteur soit remis en position Basse avant d'effectuer une nouvelle incrémentation … autrement le compteur va défiler à toute vitesse lorsque l'interrupteur sera en position Haute

4.1 Utilisation de l'afficheur :

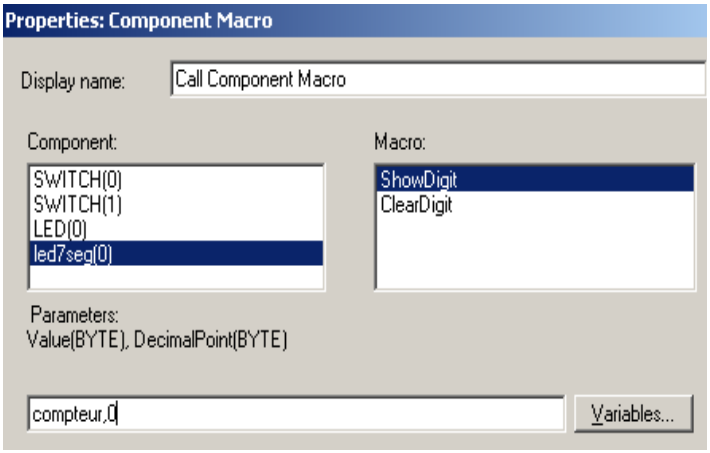
Un sous programme de gestion de l'afficheur est fourni « tout prêt ».



On le trouve dans la « boite à outils commande », il s'appelle « Component Macro »:

Le double clic ouvre la fenêtre des propriétés :

Si on sélectionne le composant Afficheur, on voit que deux méthodes sont utilisables :



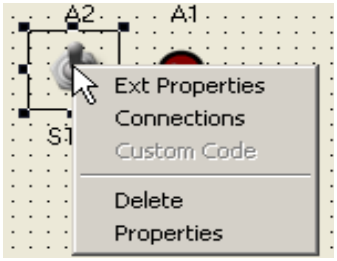
ClearDigit : permet d'effacer l'afficheur. Utile si on a plusieurs afficheurs pour un nombre à plusieurs chiffres.

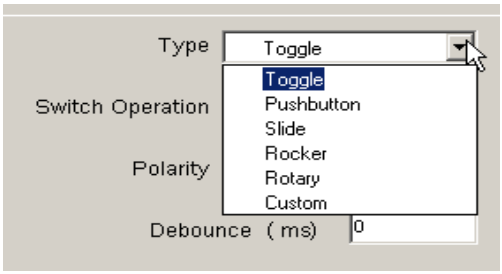
ShowDigit : que nous associerons avec la variable à afficher : ici « compteur ». Le zéro après la virgule est affecté au point décimal (nécessaire).

Dailleurs l'indication « Parameters : Value (Byte) , DecimalPoint(Byte) » indique bien que cette méthode attend DEUX paramètres.

4.2 Modification des interrupteurs

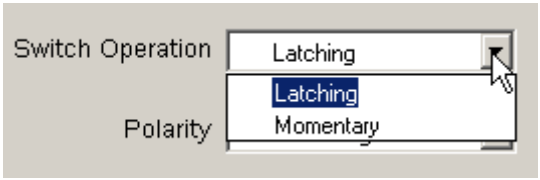
Un click droit sur l'élément ouvre un menu où l'on trouve une rubrique « Ext. Properties » (propriétés étendues).





Le type par défaut est le type « Toggle » à deux positions

Pour changer les interrupteurs en « boutons poussoir » il faut sélectionner « Pusbutton ».



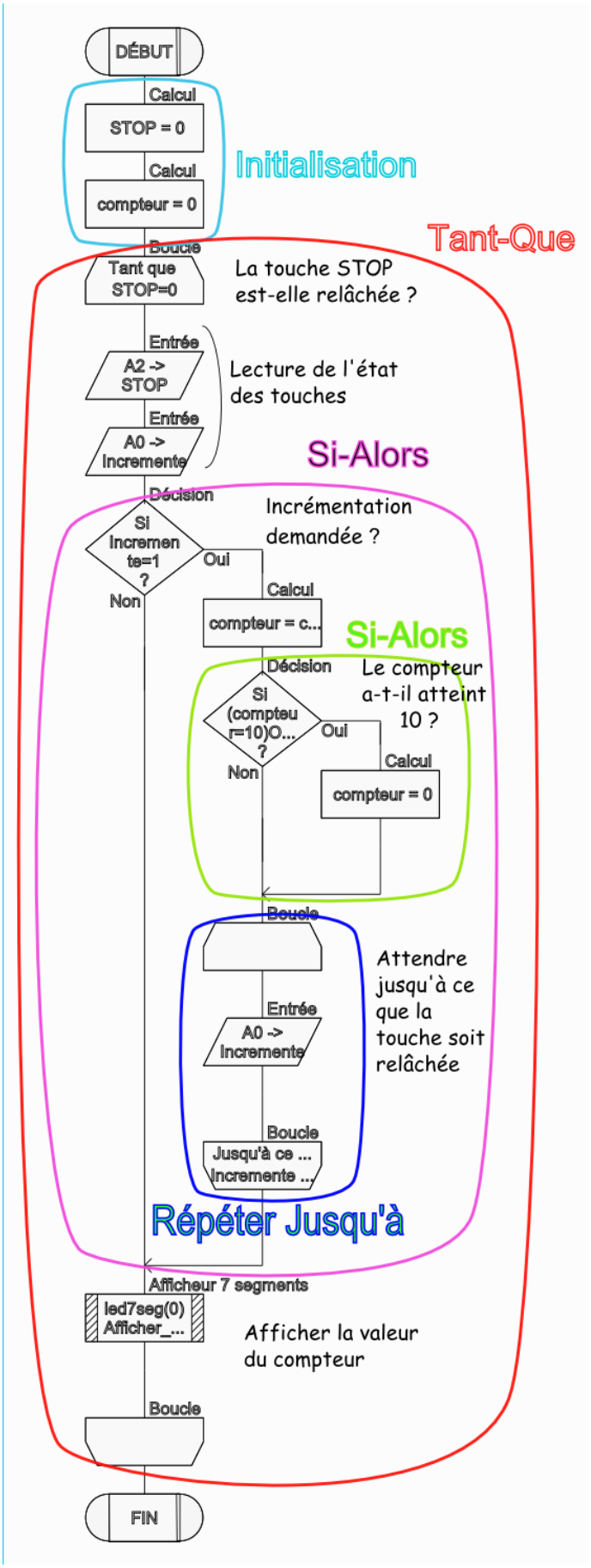
Il y a deux types de boutons poussoirs :

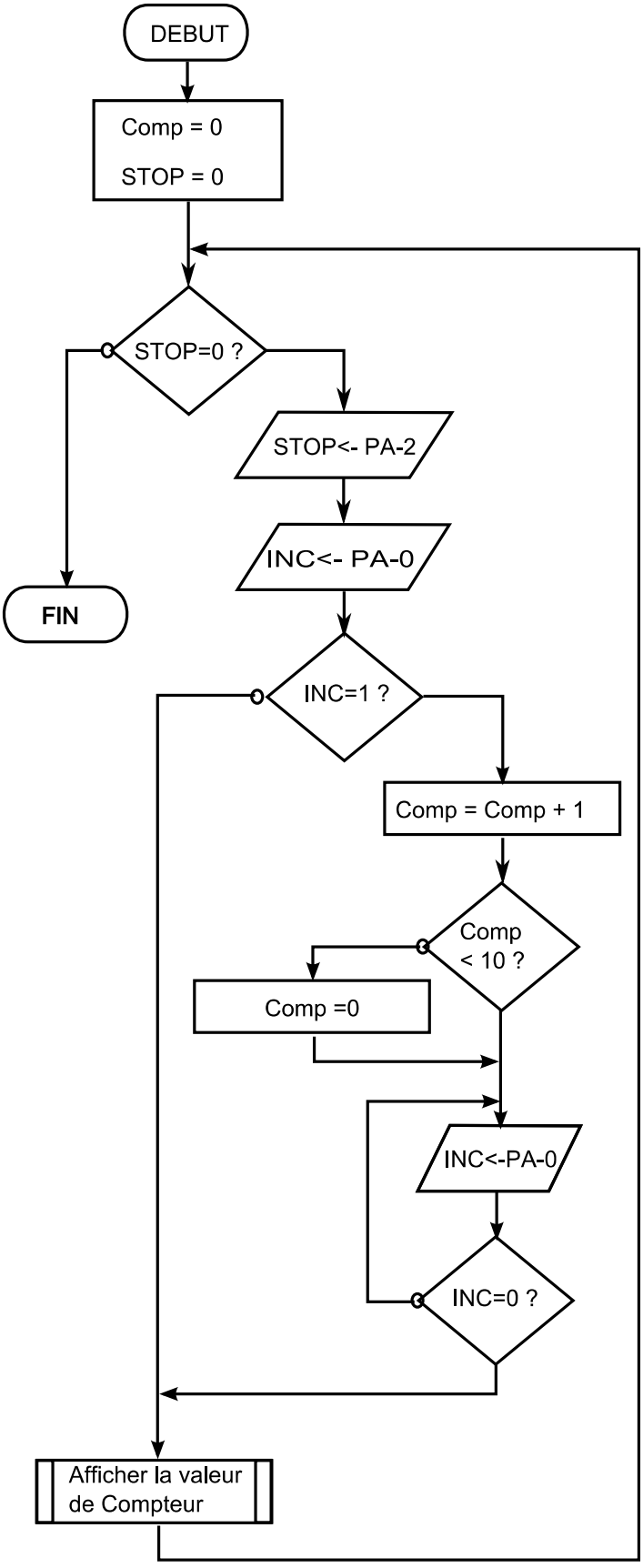
« Latching » : (bi-stable). Et

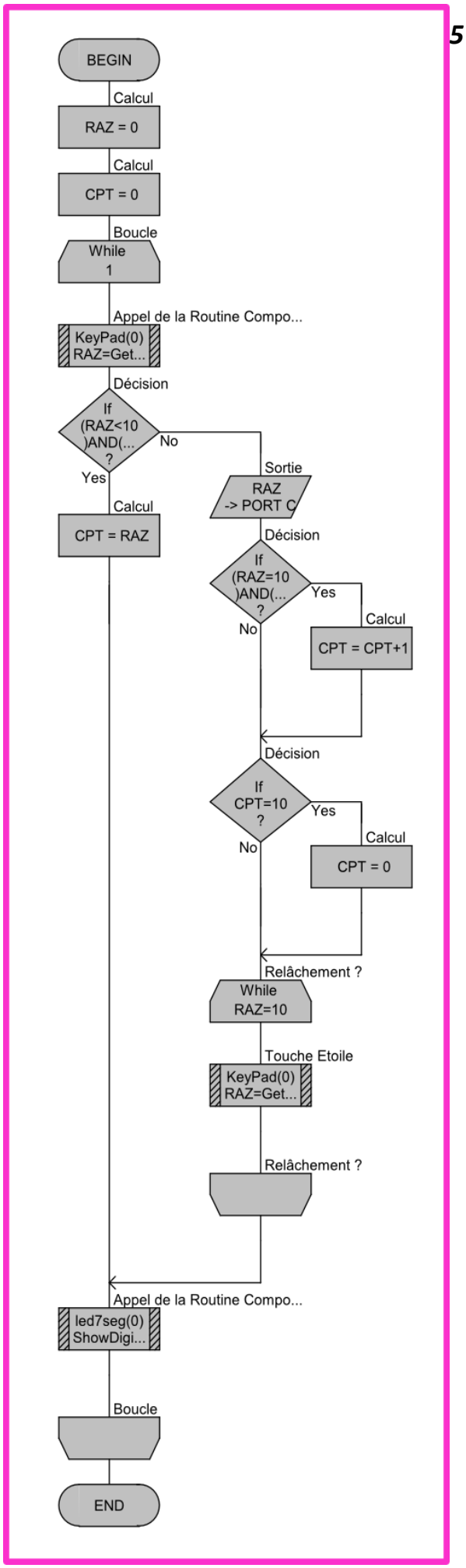
« Momentary » : actif uniquement si on appui dessus (monostable)

4.3 Algorigrammes du programme

Ci dessous un exemple du programme Flowcode et l'algorigramme correspondant







5 *Utilisation du clavier 12 touches*

On conserve l'essentiel du programme précédent. Le clavier (Keypad) servira à fournir le chiffre à faire apparaître sur l'afficheur.

5.1 Utilisation du clavier

On utilisera une « routine composant » pour lire le clavier, avec la méthode « GetKeypadNumber » qui retourne un nombre de la valeur de la touche appuyée.

La touche « * » renvoie la valeur 10 et « # » renvoie 11.

5.2 Algorithme à réaliser:

Répéter

lire la valeur renvoyer par le clavier

si la valeur est comprise entre 0 et 9 **Alors** afficher cette valeur

Si la valeur La touche est égale à « * » **Alors** incrémenter l'afficheur

Si la valeur de l'afficheur atteint 10 **Alors** la changer en 0

fin répéter

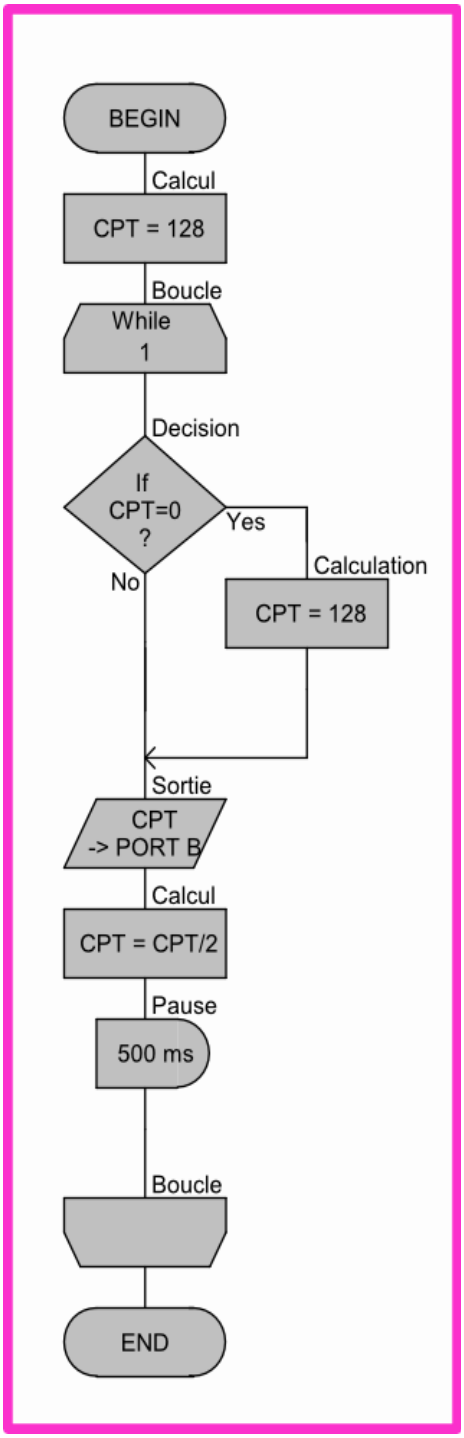
Remarque: comme l'afficheur ne peut pas afficher des valeurs supérieures à 9, pour « débugger » on pourra utiliser le « led array » qui comporte 8 leds et peut donc afficher jusqu'à 255.

5.3 Ecrire le programme et vérifier son fonctionnement correct.

6 « Chenillard à 8 leds »

6.1 *Une Led allumée qui « circule » indéfiniment de gauche à droite au rythme de 0,5s.*

On utilisera donc la variable « CPT» qui évoluera de 128 à 0 par divisions par 2 successives.
Ci-contre, deux versions du même algorithme avec deux programmes différents.



Algorithme de droite :

Début

Répéter

128 ->CPT

Tant-que (CPT différent de 0)

CPT → Port B (Leds)

CPT/2 → CPT

Pause 0,5s

Fin Tant-que

Fin Répéter

FIN

Algorithme de Gauche :

Début

Répéter

128->CPT

Si CPT=0 **alors** CPT=128

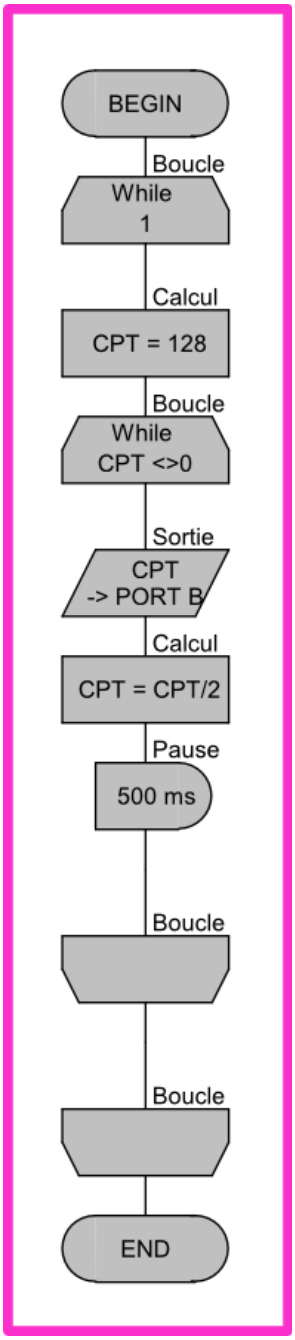
CPT->PortB (Leds)

CPT/2->CPT

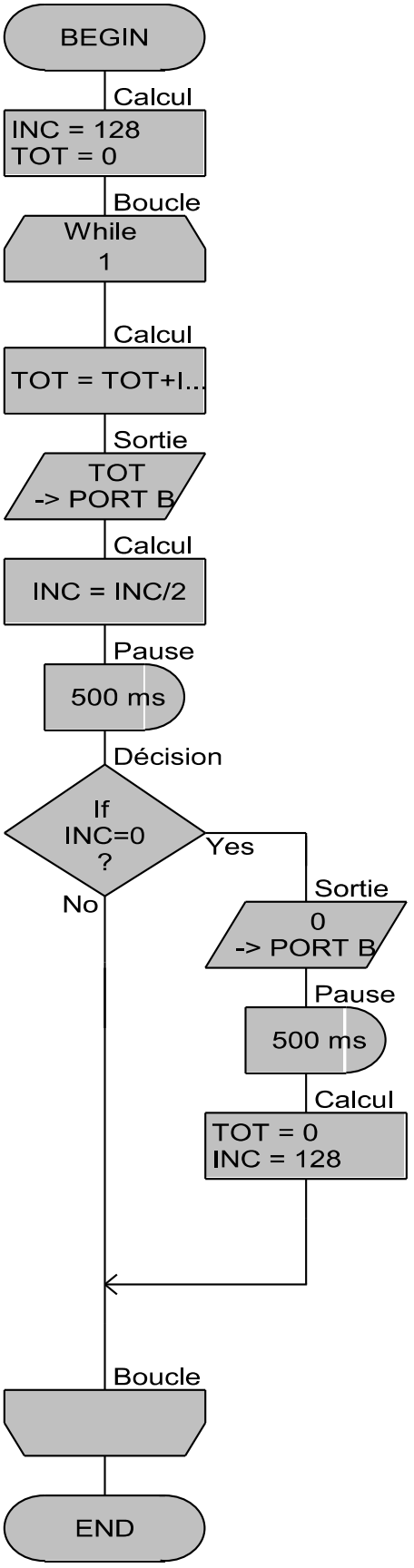
Pause 0,5s

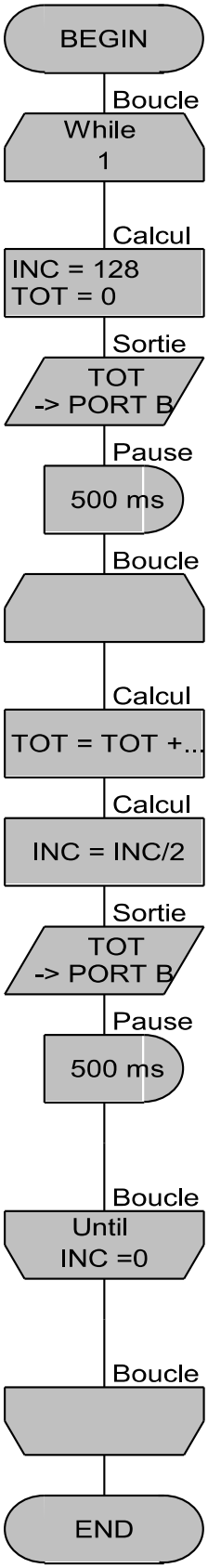
Fin Répéter

FIN



6.2 Modification de la version précédente : c'est un « ruban» qui « nait » à gauche et qui s'allonge jusquà la 8éme LED. Lorsque le ruban est complet, il s'éteint au bout de 0,5s





On utilise donc deux variables :

«CPT» qui évoluera de 128 à 0 et

«TOT» qui sera augmentée de «CPT» à chaque tour de boucle.

C'est «TOT » qui sera utilisée par l 'affichage et évoluera donc de 128 à 255, puis passera à zéro avent de reprendre à 128.

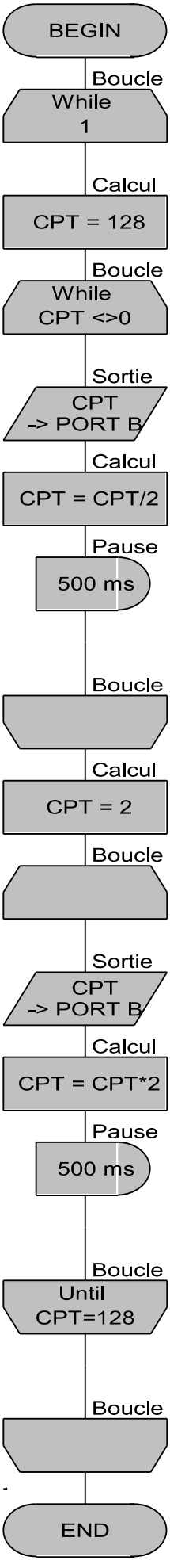
Valeurs successives de TOT :

128,	soit 10000000
128+64	soit 11000000
128+64+32	soit 11100000
etc.	
jusqu'à 255	soit 11111111
Puis 0	soit 00000000
et retour à 128	

Difficulté : obtenir les 9 états que comporte une séquence, sachant que le «remplissage» des 8 bits comprte 8 étapes.

Il faut donc ajouter un Affichage + tempo de 0,5s dans l'algorigramme.

Ci-contre deux versions du même programme.



6.3 Dérivée de la version 1 :

lorsque le point lumineux arrive «à droite», il repart en sens inverse.

<<<<<<< *Programme de Gauche*

6.4 Dérivée de la version 1 :

Deux groupes de 8 LEDS. Un point lumineux sur chaque groupe. Ils partent des extrémités opposées et se croisent au milieu.

Programme de droite >>>>>>>>>>>