Travaux Dirigés sur SIMCAL

SSP

IENAC 23 (AVI&SITA)

Présentation des TDs :

L'objet de ces travaux dirigés est l'illustration du cours de **Structure des Systèmes à Processeurs**.
On s'appliquera donc à comprendre le **fonctionnement interne** d'un calculateur générique simulé par SIMCAL.
On ne s'attardera pas sur les problèmes de programmation.

TD n°1

Prise en main de SIMCAL :

Dans un premier temps nous allons écrire nous même dans la mémoire centrale les Codes Opération en hexadécimal (ainsi que leurs opérandes si nécessaire) correspondants au code source d'un programme.

Pour illustrer cela commençons par un programme simpliste **(attention de bien respecter les adresses en mémoire centrale !)** :

Adresses	**Code Opération (+opérandes)**	Source		
0000	**A213**		JMP	Debut
0014	**B003**	Debut :	LDR	R0,0x003
0015	**FE00**		NOT	R0
0016	**FE10**	Boucle :	INC	R0
0017	**A1FE**		JMP	Boucle
0018	**FEAC**	Fin :	HLT	

Rappel : pour écrire en mémoire centrale il suffit de pointer à la souris sur la case à modifier

Une fois le code entré (à l'identique) :

- regarder dans la liste des codes machine SIMCAL (dernière page du petit fascicule) à quoi correspondent les codes que nous avons entré,
- comprendre le but du programme (pas trop complexe !),
- en observant IP, faire un RESET,
- exécuter la première boucle du programme en mode « PAS à PAS », (il faut cliquer sur l'horloge avec la souris pour simuler des tops d'horloge) en observant

les registres IP, R0, le décodeur, les flags et la mémoire centrale.

- faire un nouveau reset, puis exécuter à présent en mode « Instruction » (plus rapide) pour voir évoluer les flags au bout de quelques tours de boucles.

Questions :

- Quelle est la valeur d'IP pendant le chargement du Code Opération JMP dans le décodeur ?
- Quelle est la valeur d'IP après exécution Code Opération JMP
- Quelle était la valeur de l'opérande du JMP ? Pourquoi ?
- Qu'apporterait le remplacement de la commande JMP par un JNZ ?

Nous allons profiter à présent de l'assembleur intégré à SIMCAL.

- Taper dans l'éditeur Notepadd++ le programme suivant :

```
;Programme demo td1


val_debut equ   0                   ; directive

          org   0                   ; positionnement au début de la mémoire
          jmp   debut               ; saut au début du programme pricipal

          org 8
taille:   cst 10              ; déclaration de la constante "taille"
tableau:  var   10?           ; déclaration de la variable "tableau" (10 cases)

          org   20                  ; début du programme principal
debut:    ldr   r0,[taille]         ; recup cste taille
          ldr   r1,r0               ; taille dans r1 (servira de compteur)
          ldr   r0,val_debut        ; recup val de remplissage
          ldr   r2,r0               ; val de remplissage dans r2
          ldr   r0,tableau          ; recup adresse de début de tableau dans r0
          ldr   r4,r0               ; adresse de début de tableau dans r4
bcle:     str   [r4],r2             ; r2 dans la case mémoire pointée par r4
          inc   r2                  ; incrémentation de la val de remplissage
          inc   r4                  ; incrémentation de l'adresse dans le tableau
          dec   r1                  ; décrémentation du compteur (taille)
          jnz   bcle                ; saut tant que le compteur n'est pas à 0
fin:      hlt                       ; fin
```

- Le sauvegarder au format : .asm

- assembler votre programme .asm à l'aide de SIMCAL (Quand il n'y a plus d'erreur (en recopiant ;)), vous obtenez un .sim et un .lst)

- ouvrir le .sim dans SIMCAL et le .lst dans l'éditeur « Geany » : comparez le .lst avec ce qui se trouve maintenant en mémoire centrale de SIMCAL

- exécuter en mode automatique ce programme dans SIMCAL (oui il fallait faire un reset!)

- observer maintenant la zone de la mémoire centrale contenant la variable « tableau ». Vous comprenez l'évolution ?

- écrire l'organigramme du programme :

- pour « rigoler » on va modifier le fichier source (.asm) en positionnant la constante « taille » à 30. Sauvegardez, assemblez, chargez, exécutez ce nouveau programme. Que s'est t-il passé ? Comparez la mémoire centrale (après exécution du programme) au fichier .lst !
- Mettre un point d'arrêt sur les adresses du début de programme et sur l'adresse suivant (en 0014 et en 0015 à priori).
- Faire un reset puis relancer le programme (sans l'avoir rechargé en mémoire). Expliquez ce que vous observez (au niveau des points d'arrêts et derrière) ?!
- Qu'a causé cette constante trop grande ?

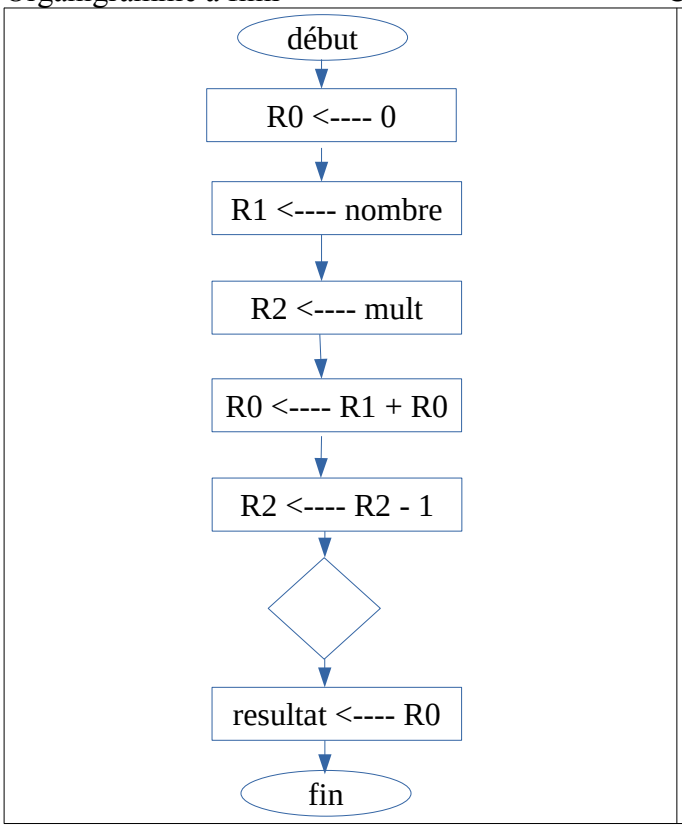
TD n°2

Sous-Programme et Pile

- **Réalisation d'une multiplication à partir d'une boucle d'addition :**

Ecrire un programme qui effectue la multiplication d'une constante « nombre » (fixée à 3) par une autre constante « mult » (fixée à 4), puis range le résultat dans une variable « résultat ».
L'organigramme est quasi fini (manque le test et la destination de la boucle).

Organigramme à finir

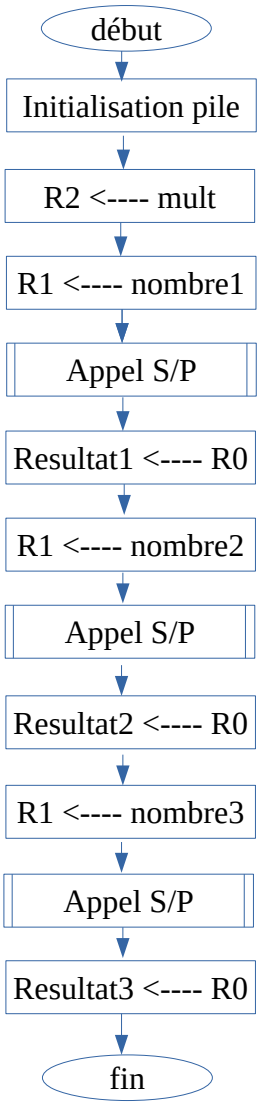


Code source

- **A présent on va transformer ce programme en sous-programme afin d'effectuer trois multiplications, avec les contraintes suivantes :**
  - pour le sous-programme :
    - paramètre d'entrée : le registre r1 pour la variable à multiplier, le registre r2 pour le multiplicateur (pour simplifier multiplicateur > 0),
    - paramètre de sortie : registre r0,
    - les registres autres que r0 (paramètre de sortie) ne doivent pas être modifiés (en sortie) par le sous-programme (utilisation de PUSH/POP si nécessaire),
  - les constantes à multiplier s'appellent nombre1(fixée à 1), nombre2 (fixée à 2), nombre3 (fixée à 3),
  - le multiplicateur en constante : mult (fixée à 4),
  - les résultats des trois multiplications : resultats1, resultats2, resultats3,
  - qui dit sous-programme sous-entend utilisation d'une pile (qu'on n'oublie pas de déclarer et « initialiser »)
  - utiliser les PUSH/POP de façon réfléchie dans le sous-programme
  - répartition en mémoire de la façon suivante :

Début du code (jump)
Zone de constantes et variables
Zone de pile
Zone du sous-programme
Zone du programme principal

Organigramme du programme principal



Code source

Exécuter votre programme en mettant des points d'arrêts judicieux pour observer la pile et SP ; puis finir en mode automatique pour aller observer vos variables dans la mémoire centrale (et vérifier ainsi que votre programme fonctionne.... ou pas !)

Contenu de la zone constantes/variables de votre mémoire à la fin du programme :

Nom de la cste/var	Adresse en mémoire	Valeur

Expliciter le contenu de votre pile à la fin du programme :

Adresse	Valeur	commentaires

Quelle est la valeur de SP à la fin du programme ?

La valeur d'IP ?

TD n°3

Utilisation des périphériques / Tests de mots d'état

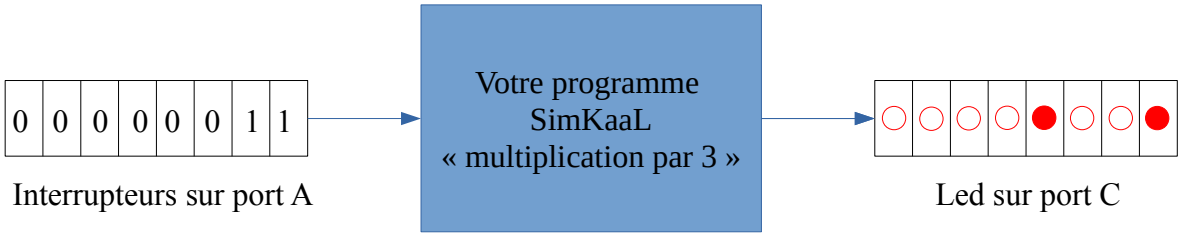
- **E/S parallèles :**

A partir de ce programme, qui ressemble fort à celui que vous avez réalisé au début du TD2 (donc vous pouvez reprendre le vôtre !)....

```
org 0
jmp     debut

org 10
mult:           CST 4
nombre:         CST 3
resultat:       VAR ?

org 20
debut:          ldr     r0,0
                ldr     r1,[nombre]
                ldr     r2,[mult]

bcle:           add     r0,r1r0
                dec     r2
                jnz     bcle
                str     [resultat],r0
                hlt
```

… on va conserver la constante « mult », remplacer la constante « nombre » par une variable (alimentée par le périphérique d'E/S parallèles (interrupteurs sur le Port A)), et utiliser la variable « resultat » pour allumer des led sur le port C. De plus on va faire tourner le programme en boucle infinie (inconditionnelle) afin de pouvoir faire plusieurs opérations de multiplications à la suite.
On modifie donc ce programme pour le rendre interactif par l'intermédiaire du port parallèle !
A vous de jouer !!!

On vient de réaliser une superbe calculatrice de ce type :

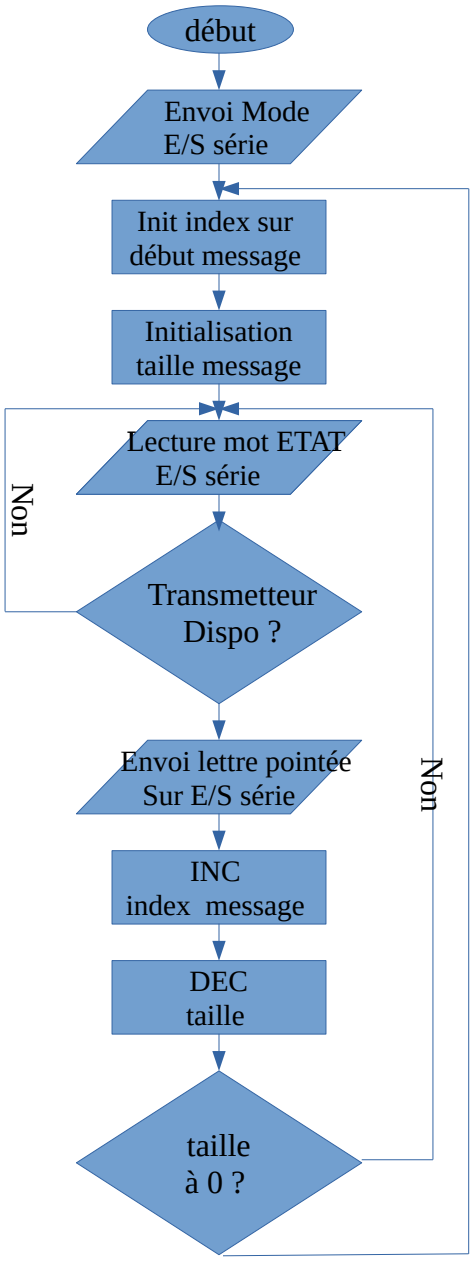


Remarque : ne pas oublier de configurer le circuit d'E/S parallèles (cela fait partie des initialisations de début de programme).
Exceptionnellement il n'est pas utile de faire un organigramme, ce ne serait qu'un enrichissement de celui du TD2 dont il est issu !

- **E/S série et test de mot d'état :**

Le but de ce nouveau programme est d'afficher à l'écran de SIMCAL le message « SIMCAL IENAC » en boucle. Pour cela, quelques petites suggestions (si nécessaire) :

- utiliser une constante de type tableau contenant le message « SIMCAL IENAC » (en assembleur on peut déclarer cela ainsi : *message : CST ' SIMCAL IENAC '*), il suffit ensuite d'incrémenter un index sur ce tableau, permettant de récupérer son contenu en mémoire centrale, pour l'envoyer vers le périphérique écran, via la sortie E/S série
- initialiser le périphérique d'E/S série en mode 2 (ce qui correspond à l'option transmission uniquement (voir fascicule)),
- avant d'envoyer vers la sortie E/S série, il faut vérifier que le mot d'état annonce que le « transmetteur est disponible » (fameux test de mot d'état sur le bit TV),
- le programme tourne en boucle infinie pour afficher en permanence ce message,
- Que ce serait-il passé si on n'avait pas utilisé le test de mot d'état ? Demander aux instructeurs éventuellement !

Organigramme	Code source assembleur
	*;td 3-2 affichage message avec test mot d'état* *SModeEtat EQU 0xFF5* *SRxTx EQU 0xFF4* *org 0 ; première instruction* *org 10 ; zone de données* *message: CST 'SIMCAL IENAC-'* *taille: CST* *org 30 ; votre programme* *hlt*

- Observer en mémoire centrale les codes ASCII composants le message à afficher
- Citer dans ce votre programme un exemple d'adressage en :
  - mode immédiat :
  - mode direct :
  - mode implicite :
  - mode indirect par registre
- Ecrire l'organigramme d'un programme permettant de lire en boucle des lettres tapées au clavier de l'entrée du périphérique E/S série, puis ranger ces lettres en mémoire centrale dans un tableau de 10 cases. Si vous avez du temps écrire au moins la boucle de test du mot d'état « récepteur plein RP » , au mieux l'intégralité du programme....

TD n°4

Sous-programme d'interruption :

- **Interruption manuelle :**

Ecrire un programme qui, en boucle sans fin, incrémente de 0 à FFh une variable dont la valeur est envoyée sur le port C du port parallèle.
Ce programme pourra être interrompu par le bouton « IT manuelle » qui réinitialisera la variable à 0.

L'organisation en mémoire centrale sera la suivante :

Début du code (jump)
Table des vecteurs d'interruption
Zone de constantes et variables
Zone de pile
Zone du sous-programme d'interruption
Zone du programme principal

Pour éviter de perdre du temps inutilement le programme principal vous est donné page suivante, il vous reste seulement à :

- comprendre le programme principal,
- vérifier l'organisation en mémoire centrale
- écrire le sous-programme d'interruption,
- réserver la table des vecteurs
- initialiser la table des vecteurs
- en clair compléter les zones grisées
- et à tester tout ça !!!!

...

- que pensez vous de l'utilisation (généralement conseillée) de PUSH/POP dans le sous-programme d'interruption ?

```
;tp 4-1 ; interruption manuelle

PicuMask        EQU             0xFF8
PicuEOI         EQU             0xFF9
PPMode          EQU             0xFF3
PPPortC         EQU             0xFF2
masque          EQU             0x05   ; pour que seule l'It manuelle soit autorisée
mode            EQU             04

;première instruction
org 0
jmp       init

;Table des Vecteurs
org                                     ; adresse en TdV concernant l'interruption IR2 (It Manuelle)
ad_SPIt:

;données
org 10
valeur:   VAR ?             ; notre variable à afficher et incrémenter

;pile
org 20
pile:           VAR 10?
fond_pile:      VAR ?

;programme
org 40

;sous-programmes  d'interruption

It_man:

;programme principal
init:           ldr       r0,0                  ;CLI
                ldr       fl,r0
                ldr r0,fond_pile                ;init SP
                ldr sp,r0
                                                ;init TdV

                ldr       r0,1                  ;init Port Parallèle
                str [PPMode],r0
                ldr       r0,masque             ;init PICU (masque)
                str [PicuMask],r0
                ldr       r0,1                  ;STI
                ldr       fl,r0

                ldr r0,0xff                     ; valeur max de  valeur
                ldr       r1,r0                 ; stockée dans R1
                ldr r0,0                ; init valeur à zero
                str [valeur],r0

bcle:           ldr r0,[valeur]
                str [PPPortC],r0
                inc       r0
                str [valeur],r0
                sub r4,r0r1
                jnz bcle
                jmp bcle
                hlt
```

- **Interruption clavier (E/S série) & interruption manuelle :**

Forts de cette première expérience dans le domaine des interruptions nous allons utiliser celles provenant du périphérique E/S série, plus précisément celles générées par le clavier branché sur celui-ci. Le rôle de notre programme sera donc de réceptionner (par interruption) les caractères frappés au clavier et de les afficher à l'écran. Ce programme tournera en boucle infinie et ne s'arrêtera que sur interruption manuelle.

A vous de jouer :

Notre **programme principal** comportera donc :

- ✗ une première partie concernant les initialisations divers :
  - pointeur de pile,
  - table des vecteurs (pour les 2 interruptions : manuelle et clavier),
  - PICU (masque autorisant les deux interruptions précédentes),
  - port E/S série (fonctionnement en réception, transmission, et par interruption),
  - acceptation des interruptions par le calculateur.
- ✗ Une seconde partie constituée d'une boucle infinie sans autre rôle que d'éviter que le programme s'arrête. Cette partie est en revanche interruptible et c'est là son unique intérêt.

Un **sous-programme d'interruption (nommé ITman**), lié à l'interruption manuelle, qui permet d'arrêter définitivement le programme.

Un **sous-programme d'interruption (nommé ITclav)**, lié au port d'E/S série, sur lequel on exploite l'interruption générée par la frappe au clavier, pour interrompre la boucle sans fin du programme principal. Son but est de récupérer le code ASCII de la touche frappée au clavier et de l'afficher à l'écran simulé (sans test de mot d'état, pour simplifier !)

Inutile de vous rappeler la partie déclaration... (variables, constantes, piles, table des vecteurs)

Organigramme du programme principal :

Organigramme du sous-programme d'interruption :

Votre programme :

; tp 4-2 Interruption E/S serie clavier, recup lettre et affiche à l'écran, remise à 0 par It_man, !

question subsidiaire : gérer le bit ER du mot d'état pour mettre l'accent sur les problèmes d'écrasement lorsque l'on tape trop vite au clavier...

TD n°5

Exercice de synthèse n°1 :

Ecrire un programme de gestion d'un parking à 9 places.

Consignes: Le programme, dans une boucle sans fin, affichera le nombre de places libres et détectera la sortie d'un véhicule.

Le nombre de places libres initial sera 9 (parking vide).

Le "capteur" de sortie sera simulé par un appui sur une touche clavier (géré par interruption) au passage d'un véhicule. Ce capteur ne sera pris en compte que si le parking n'est pas vide.

La sortie d'un véhicule mettra à jour le nombre de places disponibles.

L'entrée d'une voiture est soumise à l'ouverture d'une barrière qui sera symbolisée par 8 leds, toutes allumées (fermée) ou toutes éteintes (ouverte).

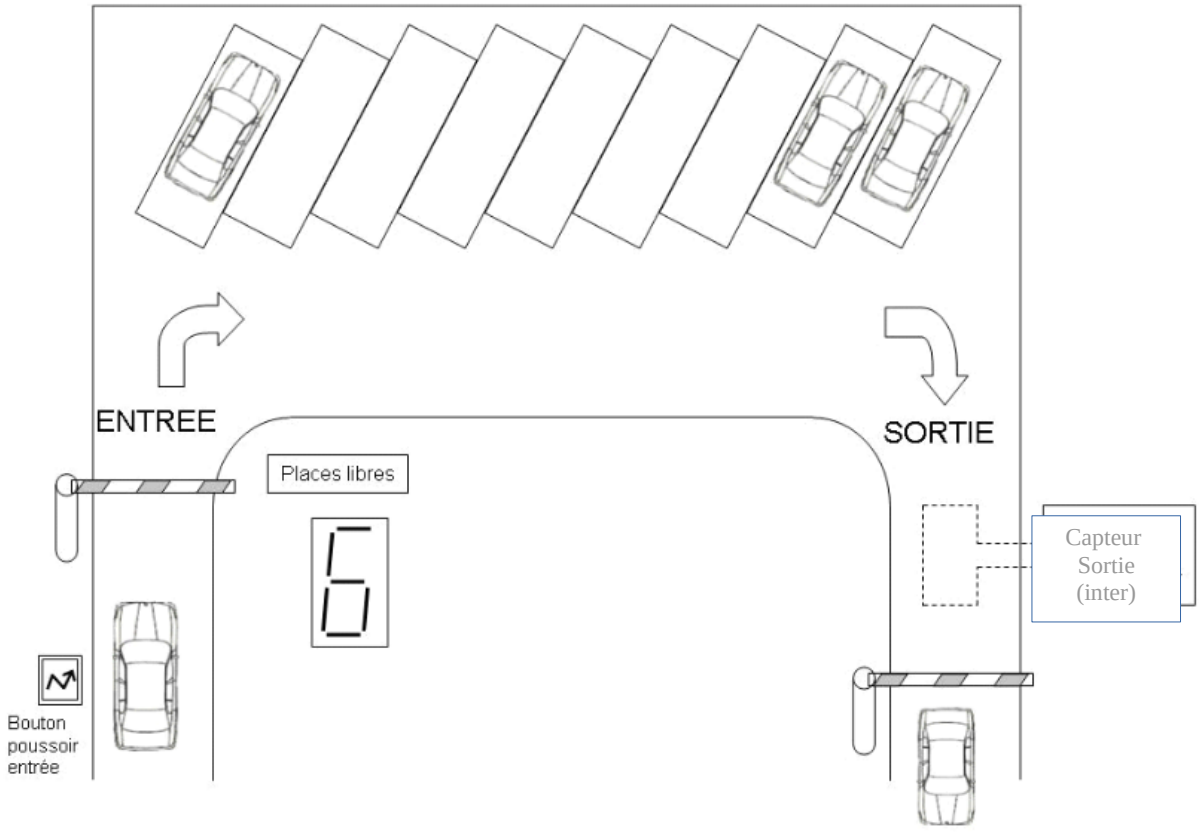
Par défaut la barrière sera fermée.

L'ouverture de la barrière sera demandée par le bouton poussoir (ITM) et ne sera réalisée que si le parking n'est pas complet.

La barrière restera ouverte 10 secondes (environ) puis se refermera.

Pour simplifier le traitement on supposera qu'à la fermeture de la barrière un véhicule est entré et le nombre de places libres sera mis à jour.

Bonus: En cas de parking plein, le programme affichera le message "COMPLET" sur l'écran



Exercice de synthèse n°2 :

Enfin un exercice, vous permettant de vous exprimer sans conseil ni consigne (seulement quelques suggestions) !
Le but à atteindre : écrire un programme qui marche de façon stable et pérenne, tout en respectant le cahier des charges ; ce programme devra réaliser un chenillard (des leds allumées qui se décalent sur une « guirlande » de leds) sur un port du périphérique E/S parallèle, à la vitesse d'environ 1 décalage toutes les 2-3 secondes (donc en utilisant le timer pour générer des interruptions). Les leds allumées correspondent à la consigne donnée par une entrée du même port E/S parallèle auquel on associe les interrupteurs. A chaque appui de l'interruption manuelle (ITM) le chenillard devra changer de sens. Pour arrêter ce programme on utilise l'entrée clavier (port E/S série par interruption), plus précisément la lettre « S » (comme Stop).

Suggestions du jour :

- les instructions ROL (ROtate Left) et ROR (ROtate Right),
- un code source commenté est l'assurance d'une considération attentive des instructeurs,
- des commentaires et un organigramme pour chaque partie, l'assurance d'une aide des instructeurs,
- on peut avancer par étapes afin de voir quelque(s) chose(s) fonctionner rapidement :
    - un chenillard immobile (en clair afficher en boucle des leds à partir d'une constante),
    - rajouter les interrupteurs pour choisir la forme du chenillard,
    - rajouter le timer pour faire décaler les leds (la chenille prend vie!),
    - rajouter l'ITM pour changer de sens,
    - rajouter l'IT clavier pour arrêter le programme avec la lettre « S »,
    - pour faire l'intéressant(e) une option utilisant les touches clavier « + » et « - » pourrait accélérer ou ralentir le chenillard....