TP 2 : Programmation élémentaire en assembleur

Introduction

Dans ce TP, vous verrez :
- comment lire un port d'entrée en assembleur,
- comment utiliser des variables en mémoire,
- comment construire une boucle en assembleur.

Preambule : instruction MOVE

Pour transférer une valeur en mémoire sur un port, vous connaissez le couple d'instructions LDAA / STAA (voir TP 1). L'inconvénient de cette paire d'instruction est qu'elle utilise un accumulateur, bien qu'**aucun calcul** ne soit fait. Or, il arrive que cet accumulateur contienne une valeur à mémoriser. On utilise donc plutôt l'instruction MOV qui transfère une valeur d'un endroit à un autre, sans utiliser d'accumulateur. Cette instruction se décline en deux versions, MOVB pour *move byte* (1 octet), et MOVW pour *move word* (2 octets).

Par exemple, pour stocker la valeur '$ab' dans la case mémoire '$cdef', il faudra écrire : `movb #$ab,$cdef`. On préfèrera en général utiliser des symboles, qui améliorent la lisibilité du programme : `movb #VALEUR,ADRESSE`, ou `movb ADR1,ADR2`

1 Utilisation de l'instruction MOV

1. Créer un nouveau fichier, et l'enregistrer dans "z :\TP_II2" avec le nom "tp2_1.asm"
2. En analysant le schéma de la carte, quelle valeurs binaire faut-il écrire sur le PORTB pour allumer la DEL connectée sur PB0 : ____________
   pour allumer celle connectée sur PB7 : ____________
3. Ecrire un programme qui va alternativement allumer ces deux dels, avec un appel à la fonction TEMPO vue au TP1 :

```
        cli     ;autorisation d'interruption par Noice
        movb  #$ff,DDRB     ; port en sortie
LOOP    movb  #$    ,PORTB  ; quelle valeur ?
        movb  #$    ,PORTB  ; quelle valeur ?
fin     bra   LOOP
```

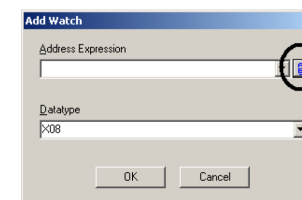
   Ne pas oublier la directive `org`, ainsi que l'inclusion du fichier de définition des registres d'E/S (voir TP 1).
4. Une fois le programme assemblé, sélectionner le fichier listing généré (tp2_1.lst) dans l'explorateur Windows, et faire clic-droit → «Générer un fichier de symboles». Cette commande va créer un fichier de "commandes Noice" (d'extension .noi), dans lequel seront inclus les définitions des symboles utilisés dans votre source. (note : cette manip sera à refaire **à chaque fois** pour chaque programme assembleur, à **chaque modification**).
5. Lancer Noice, charger le fichier .noi avec le bouton "Play Command File", et tester le programme en pas à pas.
6. Ajouter après chacune des deux instructions allumant les dels un appel à la fonction TEMPO vue au TP1, copier celle-ci dans votre source, et tester le programme.

2 Utilisation d'un port en entrée

2.1 Lecture d'un port en temps réel avec NoIce

La carte est dotée de deux boutons poussoirs (BP) qui sont connectés sur deux bits d'un port du microcontrôleur.

1. Chercher sur le schéma les deux BP : au repos, à quel niveau sont les lignes du port correspondantes : ____________
2. Sur quel port sont-ils connectés (voir document « Block Diagram ») : ____________
3. Lancer Noice, et activer la fenêtre *Watch* : cette fenêtre permet de montrer la valeur contenue à une adresse particulière.
4. Clic-droit → *Add Watch*, et cliquer sur le bouton encadré ci-dessous :



5. Dans la boite de dialogue *Select Symbol*, allez chercher le port en question, et validez. (Vous pouvez obtenir un classement alphabétique en cliquant dans l'en-tête de la colonne *Name*)
6. Puis, clic-droit → *Refresh Watches every X seconds*, et donner 1 seconde comme valeur. Cochez la case *Update While Target is Stopped* pour avoir la mise à jour même cible arrêtée.

7. Appuyer successivement sur les deux boutons poussoirs, et vérifier les valeurs affichées. Quelle est la valeur lue au repos sur le port : __________ Est-ce cohérent avec la valeur attendue ?

2.2 Lecture d'un port par programme et test des valeurs

Nous souhaitons avoir un programme qui lise l'état des boutons poussoirs, qui allume la DEL câblée sur PB7 si on appuie sur S2 et qui allume la del câblée sur PB0 si on appuie sur S3. Nous allons donc lire la valeur du port, en la plaçant dans un accumulateur, puis comparer cette valeur avec la valeur au repos : si identique, alors on recommence :

```
encore  ldaa   ______      ; ici , le nom du port
        cmpa   #____       ; ici , la valeur au repos
        beq    encore
```

Description :
– L'instruction `cmpa` effectue une comparaison, c'est à dire une soustraction, sans sauvegarde du résultat : seuls les bits d'état NZCV (dans le registre CC) sont positionnés à 1 ou 0, selon le résultat.
– L'instruction `beq` effectue un branchement conditionnel : elle teste le bit Z, et s'il est à 1 (c'est à dire ici, si le résultat de la soustraction a donné '0'), alors l'exécution se poursuit à l'étiquette indiquée. Sinon, l'exécution se poursuit à la ligne suivante.

Voici les principales instructions de branchement conditionnel :

Mnémonique	Nom	Branchement si :	est égal à :
BNE	Branch if Not Equal	Z	0
BEQ	Branch if EQual	Z	1
BPL	Branch if PLus	N	0
BMI	Branch if MInus	N	1
BCC	Branch if Carry Clear	C	0
BCS	Branch if Carry Set	C	1

Compléter le programme pour répondre au cahier des charges, et faites valider votre travail par l'enseignant.

3 Comptage des appuis sur le bouton poussoir

On souhaite écrire un programme qui affiche sur le *bargraph* de la carte la valeur binaire correspondant au nombre d'appui sur l'un des boutons poussoirs. Pour cela, nous aurons besoin d'une variable COMPTEUR, qui sera incrémentée à chaque appui.

1. Créer un nouveau fichier, et l'enregistrer dans z :\TP_II2 avec le nom TP2_2.asm
2. Déclarer une variable COMPTEUR sera déclarée dans une zone du ficher source séparée du programme (lire en annexe : "Utilisation d'une variable en RAM").

```
;------------------------------------------
; Programme
            org   $1000
            ...           ; ici , mon programme

;------------------------------------------
; Zone des variables
            org   $2000
COMPTEUR    DS.B  1
;------------------------------------------
```

Comme précédemment, nous détecterons l'appui sur le BP avec les lignes suivantes :

```
encore  ldaa   PTT
        cmpa   #__
        beq    encore
```

Il faut ensuite incrémenter le compteur avec l'instruction INC, et afficher sa valeur sur les dels :

```
        inc    COMPTEUR
        movb   COMPTEUR,PORTB
```

Puis, faire un branchement (instruction `bra`) à la ligne où l'on teste l'état du port.

3. Compléter le programme avec les instructions que vous connaissez, assembler, et tester en pas à pas (touche F9) **et** à vitesse réelle (bouton "Go").

Deux problèmes apparaissent :
– le compteur s'incrémente pendant toute la durée de l'appui,
– la valeur affichée est «inversée» (le commun des dels est au +VCC).

Pour ce dernier problème, il suffira de complémenter la valeur, avec l'instruction COM. On pourra écrire **à la place** des deux lignes ci-dessus :

```
        inc    COMPTEUR
        ldaa   COMPTEUR
        coma
        staa   PORTB
```

4. Pour le premier problème, il faut attendre le **relâchement** du bouton poussoir avant de reboucler au début du programme. Proposer une solution, et faire valider votre travail par l'enseignant.

4 Boucle en assembleur

Pour réaliser l'équivalent en assembleur d'une boucle 'for(;;)' en C, on réalise un comptage. En général, on utilisera un des deux accumulateurs comme compteur.

```
DEBUT    ldaa   #0
BOUCLE   ...          ; ici , la tâche
         ...
         inca         ; incrémentation A
         cmpa   #maxi    ; on compare
         bne    BOUCLE   ; si pas égal , on recommence...
         ...              ; sinon , on passe à la suite
```

La plupart du temps, il sera plus simple d'effectuer plutôt une décrémentation : on évite l'instruction de comparaison :

```
DEBUT    ldaa   #maxi
BOUCLE     ...        ; ici , la tâche
         deca         ; décrémentation A
         bne    BOUCLE   ; si pas à 0, on recommence
         ...              ; sinon , on passe à la suite
```

Application : Dans un nouveau fichier, sauvegardé comme "tp2_3.asm", écrire un programme qui fait clignoter 5 fois la del câblée sur PB0. Tester d'abord en pas à pas, puis insérer un appel au sous-programme de temporisation vu au TP1, et tester à vitesse normale.

5 Adressage indexé

Jusqu'à présent, vous avez vu 2 modes d'adressages :

- Le mode "Immédiat", indiqué par le '#' : la valeur donnée est la valeur à traiter.
  Exemple : `ldaa #$23` => Charge dans A la valeur (hexa) $23
- le mode "Etendu" : la valeur donnée est l'adresse de la valeur à traiter.
  Exemple : `ldaa $23` => Charge dans A la valeur qui se trouve à l'adresse $0023

Pour acceder à une table de valeurs (un "tableau" en C), il est nécessaire d'utiliser un troisième mode d'adressage, appelé "indexé" : l'adresse de la valeur à traiter est donnée par un registre index (registre X ou Y). En faisant varier l'index (incrémentation ou décrémentation), on pourra accéder à toutes les valeurs de la zone. L'index peut être considéré comme l'équivalent d'un pointeur en C.

Syntaxe : `ldaa 0,X` => Charge dans A la valeur qui se trouve à l'adresse indiquée par le contenu de X (Le '0' est un déplacement optionnel qu'on peut ajouter)

Il est impératif d'initialiser préalablement le registre X, par exemple sur la première adresse de la zone :
`ldx #ADR_BLOC` ou `ldx #$2345`

Remarque : on pourra utiliser avec l'adressage indexé pratiquement toutes les instructions d'accès à la mémoire (staa, cmpa, ...)

Exemple : pour lire les valeurs qui se trouvent de $2000 à $2010 (inclus), et les afficher sur le Port B, on écrira :

```
         ldx    #$2000
encore   ldaa   0,x
         staa   PORTB
         inx
         cpx    #$2011
         bne    encore
```

Application : Dans un nouveau fichier, sauvegardé comme "tp2_4.asm", écrire un programme qui permette de remplir une zone de mémoire avec une valeur donnée.

Par exemple, en définissant en tête du source les symboles suivants :

```
VALEUR   equ   45   (par exemple)
ADRESSE  equ   $2000
TAILLE   equ   50
```

alors à l'exécution, ce programme devra remplir les 50 octets à partir de l'adresse $2000 avec la valeur 45.

Ecrire le programme en utilisant les étiquettes ci-dessus pour spécifier les paramètres, et le faire valider par l'enseignant.

Annexe : utilisation d'une variable en RAM

On peut déclarer et utiliser des variables en assembleur, en les déclarant préalablement, tout comme en C. La seule différence est qu'il n'existe que **un seul type**, l'octet (byte), correspondant au type `char` en C.

La déclaration se fera avec la directive assembleur *Define Storage* (DS) qui permet de réserver en mémoire un certain nombre d'octets. Il faudra aussi spécifier manuellement avec la directive `org` à quelle adresse on désire stocker cette variable.

Par exemple, les lignes suivantes :

```
        org     $2000       ; localisation en mémoire
var1    DS.B    10          ; 10 Bytes
var2    DS.B    1           ; 1 Byte
```

réservent un bloc de 10 octets en mémoire à l'adresse $2000, et un bloc de 1 octet à l'adresse $200A

On pourra ensuite accéder au contenu de la variable de façon suivante :

```
ldaa    var2  ; lecture et chargement dans A
staa    var2  ; écriture de A en mémoire
```

On pourra aussi écrire :
`movb #33,var2` ⇒ stocke la valeur '33' dans la mémoire à l'adresse 'var2'

Ou aussi :
`movb var2,une_autre_variable` ⇒ transfère le contenu de 'var2' dans 'une_autre_variable'