# Rapport Travaux Pratique n°4
# Assembleur

Zacharia Benkirane, Thomas Petazzoni

5 janvier 2003

## 1 Présentation

Il s'agit dans ce TP de se familiariser avec la chaîne de développement pour 68HC11.

## 2 Etude d'un programme assembleur

### 2.1 Le programme Cheni.s

| | | | |
|---|---|---|---|
| | `PORTA = $1000` | | |
| | `DDRA = $1001` | | |
| | `.area text` | | Début de la zone de code |
| `Init :` | `lds #$7FFF` | Immédiat | Initialisation de l'adresse de la pile (en haut de la mémoire) |
| | `cli` | | Autorisation des interruptions (pour le débuggeur) |
| | `ldaa #$FF` | Immédiat | On charge le registre A avec la valeur 0xFF (tous les bits à 1) |
| | `staa DDRA` | Étendu | On écrit la valeur du registre A (0xFF) sur à l'adresse DDRA, ce qui a pour effet de configurer tous les bits du Port A en sortie. |
| `Main :` | `ldaa #1` | Immédiat | On charge la valeur du registre A avec la valeur 1 |
| `Boucle :` | `cmpa #$80` | Immédiat | On compare la valeur du registre A avec 0x80. Si c'est égal on positionne le bit $Z$ du registre d'état à 1. |
| | `bne else` | | Saute au label *else* si $Z = 0$. |

| | | | |
|---|---|---|---|
| | `ldaa #1` | Immédiat | Charge A avec la valeur 1. |
| | `bra finsi` | | Saut inconditionnel au label *finsi*. |
| `else :` | `lsla` | | Décalage vers la gauche du registre A (multiplication par 2). |
| `finsi :` | `staa PORTA` | Étendu | On copie la valeur du registre A à l'adresse PORTA, c'est à dire qu'on envoie cette valeur sur le Port A. |
| | `bsr Tempo` | | Appel de la sous-routine *Tempo* |
| | `bra Boucle` | | Branchement inconditionnel au label *Boucle* |
| `Tempo :` | `pshx` | | On empile la valeur de X (sauvegarde de la valeur de X). |
| | `ldx #$8000` | Immédiat | On charge le registre X avec la valeur 0x8000. |
| `Wait :` | `dex` | | Décrémente le registre X de 1. |
| | `bne Wait` | | Tant que X n'est pas égal à 0, on repart au label *Wait*. |
| | `pulx` | | On dépile le sommet de la pile dans X (restauration de la valeur de X). |
| | `rts` | | On sort de la sous-routine, et on retourne là où on a été appellé. |

**Commentaires :**

Le programme initialise une pile, en haut de la mémoire. Ensuite, il configure le Port A, de manière à ce que tous les bits puissent être utilisés en sortie. Il charge ensuite la valeur 1 dans A (au niveau de *Main* pour la première boucle, à la troisième instruction de *Boucle* les autres fois), puis tant que A n'est pas égal à 0x80, il décale A vers la gauche, envoie la valeur de A sur le port A, fait une petite temporisation via la sous routine *Tempo*, et boucle ainsi à l'infini. Quand A est égal à 0x80, alors on recharge A avec la valeur 1, et on repart.

Les sorties sont donc successivement : 0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80.

La sous routine *Tempo* est très simple : elle sauvegarde la valeur du registre X sur la pile. Puis elle charge ce registre avec la valeur 0x8000, et décrémente ce registre jusqu'à ce qu'il atteigne 0. La valeur du registre X est alors restaurée (en dépilant), puis on retourne de la sous-routine grâce à l'instruction `rts`.

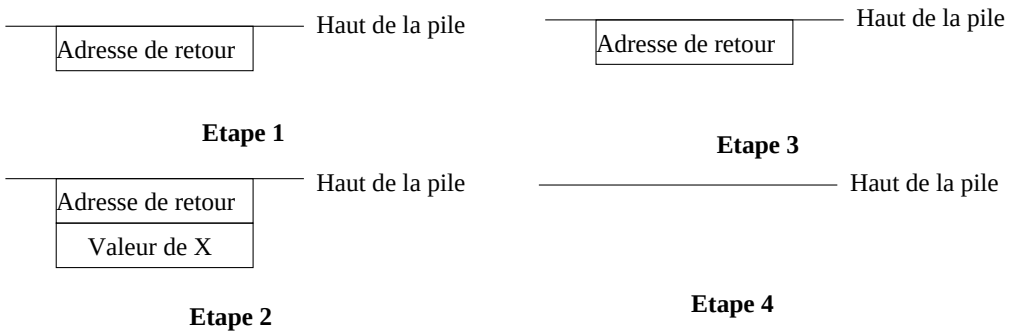
## 2.2 Fonctionnement de la pile

La pile est stockée dans la mémoire. Le registre *SP* indique quelle est l'adresse du bas de la pile. A chaque fois qu'on empile une valeur, *SP* est décrémenté, et

a chaque fois qu'on dépile une valeur, *SP* est incrémenté. La pile part donc des adresses hautes pour aller vers les adresses basses.

Par exemple `pshx` va empiler la valeur du registre X, tandis que `pulx` va dépiler une valeur de la pile, et la placer dans le registre X. L'instruction `lds` permet de spécifier l'adresse du haut de la pile.

Lors de l'appel à la sous-routine *Tempo*, via l'instruction `bsr`, l'adresse de retour est empilée (étape 1). Cette adresse correspond à l'adresse de l'instruction située directement après le `bsr`. Ensuite, l'éxécution se poursuit au niveau de *Tempo*. La valeur du registre X est alors empilée (étape 2). La boucle s'éxécute, puis on dépile la valeur en haut de la pile dans X (étape 3). Ensuite l'instruction `rts` dépile une valeur de la pile pour connaître l'adresse de retour (étape 4).




## 2.3 Réécriture du programme en langage C

Le programme est assez simple à convertir en C. Une simple boucle infinie qui multiplie la valeur de *PORTA* par 2 à chaque itération, et qui la réinitialisé à 1 lorsque l'on a atteint 0x80. Une fonction *tempo* sert à faire une temporisation.

```
/* Inclusion du fichier de description du processeur : ports PORTA et DDRA
   */
include <hc11.h>

/* Fonction tempo, definie plus loin */
void tempo(int cnt);

int main(void)
{
   /* On active les interruptions */
   asm("cli");
   /* On configure tous les bits de PORTA en sortie */
   DDRA=0xFF;
   /* On initialise la valeur de PORTA */
   PORTA=0x1;
```

```
    /* Boucle infinie */
    while(1)
    {
        /* Si on est arrive a 128, on repart a 1 */
        if (PORTA==0x80) PORTA=0x1;
        /* Sinon, on multiplie par 2 a chaque fois */
        else PORTA = PORTA * 2;
        /* Petite attente */
        tempo(0x8000);
    }
    return 0;
}

/* Fonction permettant d'attendre un certain temps */
void tempo(int cnt)
{
  for ( ; cnt != 0; cnt--) ;
}
```

# 3 Ecriture d'un programme en assembleur

## 3.1 Sans inversion

Cette première version du programme est très simple. On définit tout d'abord quelques valeurs (*porta, portc, ddra*). Au début du programme, on initialise une pile, on active les interruptions, puis on configure le port A en sortie (`ldaa #$FF, staa ddra`). Enfin le corps du programme est une boucle infinie qui lit la valeur sur le port C dans le registre A (`ldaa portc`), puis écrit cette valeur sur le port A (`staa porta`).

```
porta=$1000
portc=$1061
ddra=$1001

        .area text



Init :  lds #$7fff
        cli
        ldaa #$ff
        staa ddra
```

```
Main:   ldaa portc
        staa porta
        bra  Main
```

## 3.2 Avec inversion

Cette seconde version est très similaire à la précédente, sauf au niveau de la boucle principale. La valeur du port C est chargée dans le registre A (`ldaa portc`), puis la sous-routine *Inverse* est appelée, pour procéder à l'inversion. Après l'éxécution de cette procédure, la valeur inversée se trouve dans le registre B, qu'on écrit sur le port A (`stab porta`).

La sous-routine *Inverse* est assez subtile : on décale à droite le registre A, pour faire tomber le bit de poids faible dans le bit de retenue. Ensuite, on ajoute 0 + le bit de retenue au registre B, qui est décalée vers la gauche à chaque itération pour faire remonter les bits de poids faible de A vers les bits de poids fort de B. Cette opération est réalisée 8 fois, pour les 8 bits. Le comptage s'effectue grâce au registre, chargé initialement avec 8, puis décrémenté à chaque itération.

```
porta=$1000
portc=$1061
ddra=$1001

        .area text

Init :
        ;  Initialisation  de la  pile
        lds #$7fff
        ; Activation des interruptions
        cli
        ; Configuration du PORTA en ecriture
        ldaa #$ff
        staa ddra

Main:
        ; On charge la valeur du PORTC dans le registreA
        ldaa portc
        bsr Inverse
        ; On reporte cette valeur sur le port A
        stab porta
        bra  Main

Inverse :       ldx #$08
```

```
Boucle:     lslb
            lsra
            adcb #$00
            dex
            bne Boucle
            rts
```