Présentation sommaire du 68HC11E9

I) Introduction

Contrairement aux microprocesseurs, les microcontrôleurs comportent en plus de leur unité de calcul de la mémoire et des circuits périphériques.

Le microcontrôleur 68HC11 est développé par Motorola. Il existe en plusieurs modèles possédant des caractéristiques un peu différentes. La version que l'on utilise s'appelle 68HC11E9.

Pour tout renseignements complémentaires, voir le manuel complet utilisateur du 68HC11E9

II) Registres

Le 68HC11 possède deux types de registre :

- des registres à usage général :
  - sur 8 bits : A et B, appelée accumulateurs ; ils sont utilisés pour les transferts de données entre la mémoire et l'unité de calcul ;
  - sur 16 bits : X et Y, appelés registres d'index car ils peuvent être utilisés pour l'adressage indexé ; il existe également le registre SP appelé pointeur de pile (voir plus bas) et le registre PC (Program Counter) ;
  -
- des registres-système, dont chaque bit a une signification bien particulière, qui servent à configurer le 68HC11 et utiliser leurs possibilités.

<table>
<tr><td>7 A 0</td><td>7 B 0</td></tr>
<tr><td colspan="2">15 D 0</td></tr>
<tr><td colspan="2">15 X 0</td></tr>
<tr><td colspan="2">15 Y 0</td></tr>
<tr><td colspan="2">15 SP 0</td></tr>
<tr><td colspan="2">15 PC 0</td></tr>
</table>

Les accumulateurs A, B, D

A et B sont des registres utilisés pour des transferts 8 bits. Ils peuvent être regroupés en un seul registre D, A constituant les poids forts et B les poids faibles. L'accumulateur D est donc d'une capacité de 16 bits, est utilisé pour des transferts de données supérieures à 8 bits.

Les registres d'index

Les registres X et Y sont des registres d'adresse : ils servent à stocker des adresses, ce sont des pointeurs sur la mémoire. Ils sont également appelés registres d'index, ou d'indexation.

Ils sont utilisés pour l'accès aux périphériques et à leurs registres de configuration. Avant usage on y charge une adresse représentant un périphérique, ou bien une adresse mémoire afin d'y mettre ou récupérer une donnée.

Le pointeur de pile SP

Lors de l'exécution d'une routine d'interruption ou d'un sous-programme, certains registres système du 68HC11 sont sauvegardés automatiquement dans un endroit de la mémoire RAM appelée pile. Cet endroit est pointé par le registre 16 bits SP, appelé pointeur de pile.

Les registres-utilisateur peuvent également être sauvegardés dans la pile et récupérés plus tard, par programmation, par les instruction `push` et `pul`.

Le compteur ordinal PC (Program Counter)

Celui-ci en lecture seule et contient en permanence l'adresse de la prochaine instruction du programme.

Le registre CCR (Code condition register)

Ce dernier est indispensable, il permet de vérifier l'état de tous les drapeaux du µC, ils sont très importants pour toutes les instructions de tests qui sont basées sur l'usage de ces drapeaux.

III Ports d'entrée-sortie

a) Caractéristiques communes

Le 68HC11 possède 5 ports d'entrée-sortie appelés port A, B, C, D, E.

Le port A comporte des bits en entrées , d'autres en sorties et d'autres en entrée sortie.

Le port D est bidirectionnel et nécessite le paramétrage de son registre DDRD afin de spécifier la direction des données sur ces ports. Le port B fonctionne qu'en sortie et le port E ne fonctionne qu'en entrée. Le port C est bidirectionnel peut être programmé en entrée/sortie.

Le tableau ci-dessous donne le nom du port, sa configuration et le paramètre de configuration s'il existe (pour les modes de fonctionnement circuit seul et bootstrap)

Ports	Entrée seule (unidirectionnel)	Sortie seule (unidirectionnel)	Entrées/sorties (bidirectionnel)	registre de direction (si E/S)
A			X	DDRA, PACTL(bits 3 et 7)
B		x		
C			X	DDRC, PORTCL
D			x	DDRD
E	x			

Les registres de direction servent à définir si le port est en sortie ou en entrée. Ce registre n'existe donc que dans les cas ou le port est bidirectionnel. Dans ces registres si un bit est à 1, le bit correspondant du port correspondant est configuré en sortie, si le bit est à 0 il est en entrée :

Bit d'un port DDRx	Bit du port x correspondant
0	entrée
1	sortie

b) Détail des différents ports

Port A

Le port A un est port d'usage général d'entrée/sortie 8 broches de données (PA0 à PA7).

Les bits PA0 PA1 et PA2 sont en entrée. Les bits PA4 PA5 PA6 sont en sortie les bits 3 et 7 du PORT A peuvent être programmés en entrée ou en sortie par les bits DDRA7 et DDRA3 du registre PACTL. Le port A offre en plus la possibilité de communiquer avec le timer.

Port B

Le port B ne sera pas utilisé pour ce projet. En mode circuit seul et bootstrap, le port B est en sortie. En mode étendu ou spécial test , les bits du port B contiennent les adresses de poids forts.

Port C

Le port C ne sera pas utilisé pour ce projet. En mode circuit seul et bootstrap, le port C est en entrée /sortie. Après un RESET les bits sont par défaut en entrée.LE registre de direction de données DDRC permet de mettre les bits en entére ou en sortie En mode étendu ou spécial test , les bits du port C contiennent les adresses de poids faibles.

Port D

Le port D est un port bidirectionnel d'usage général de 6 bits. il est associé à un registre de données et à un registre de direction des données DDRD[0,5]. Les 6 lignes peuvent être utilisées pour gérer l'interface de communication série SCI ou l'interface série périphérique SPI.

Port E

Le port E est un port 8 bits de communication unidirectionnel, en entrée seulement. Il peut servir d'interface d'acquisition analogique pour l'utilisation du convertisseur analogique-numérique (CAN) intégré au 68HC11. Les broches inutilisées du port E sur le convertisseur peuvent être utilisées comme entrées logiques.

Une conversion analogique-numérique s'obtient en positionnant un certain bit d'un registre de gestion de la conversion ; on attend alors qu'un autre bit indique que la conversion est terminée.

V) Les circuits périphériques

a) Le timer

Permet de générer des bases de temps précises. Il peut fonctionner par interruptions, c'est à dire qu'une interruption peut être générée à intervalles de temps réguliers, programmables. La

durée maximale entre deux interruptions successives est de l'ordre de 0,5 s. On peut obtenir des durées plus importantes par programmation, en utilisant des boucles par exemple.

b) L'interface de communication série asynchrone (SCI)

Elle permet des échanges d'information avec la carte, notamment avec le protocole RS-232. Le fonctionnement de cette interface sera détaillée à la séance 2 du projet.

c) L'interface de communication série synchrone (SPI)

Cet interface est un sous système indépendant de communication qui permet de communiquer de façon synchrone avec des synthétiseurs de fréquence , des écrans LCD, des convertisseurs analogique numériques ou d'autres microprocesseurs. En plus des données, il y a transmission d'un signal d'horloge.

d) Le CAN

Il possède 8 entrées analogiques connectées au port E. Ces entrées sont multiplexées, c'est à dire qu'elles ne peuvent pas être utilisées toutes en même temps. La conversion est effectuée sur 8 bits.

Voir également le paragraphe décrivant le port E, plus haut.

VI) Programmation

IV.1) Structure des programmes

Les programmes peuvent être écrits indifféremment avec des majuscules ou des minuscules.

En général, la structure des programmes est la suivante :

- déclaration des constantes

exemple :

```
adprog equ $2000          ;adresse de début du programme
```

- déclaration des registres-système

exemple :

```
porta   equ   $1000       ;adresse du port A
```

- déclaration des variables

exemple :

```
      org    $0000        ;positionnement en début de RAM
tab   dsm    10           ;réservation de 10 emplacements-mémoire (=octets)
```

- début et fin de programme :

exemple :

```
   org    startbase    ;positionnement du PC (Program Counter)
   …                   ;code du programme + saut à routines
                       ;+ déclaration des routines

   end                 ;fin du programme
```

Exemple de programme mettant à 1 les sorties 0 et 1 du port B :

```
PORTB equ     $1004
      org     $2000     ;
      ldaa    #$03      ;chargement de l'accumulateur A avec la valeur voulue
      staa    PORTB     ;copie de cette valeur sur le port
      end               ;fin du programme
```

IV.2) Langage de programmation : assembleur

Le langage est composé de directives et d'instructions d'assemblage :

- Les **directives** d'assemblage sont interprétées lors de l'assemblage
- Les **instructions** sont exécutées après lancement du programme

Les programmes se composent d'un programme principal et de routines, ou sous-routines. Ces dernières sont équivalentes aux fonctions de langage C par exemple : on les appelle par une instruction de saut (voir liste de quelques instructions ci-dessous), et on en sort par l'instruction de fin de sous-routine.

a) Syntaxe du langage

Ce paragraphe donne quelques directives et instructions, parmi les plus courantes.

Directives d'assemblage

EQU Assigne un symbole, une valeur ou une étiquette.
Exemple :

```
REGBASE EQU $1000
```

RMB
Réserve des emplacements (octets) dans la mémoire.
Exemple :

```
      ORG $0000
EX1   RMB 1       ; reserve 1 octet à l'adresse $0000.
EX2   RMB 2       ; reserve 2 octets aux adresses $0001 et $0002.
```

FCB Permet la définition d'une constante sur 1 octet. Si on veut assigner un tableau, il faut séparer les valeurs par des virgules.
Exemple :

```
      ORG $0000
TABLE FCB $7A,$F1       ;met les valeurs $7A, $F1 respectivement aux
```

```
                    ;adresses $0000 et $0001 dans la RAM
```

FDB Comme FCB mais sur un double (2 octets)
Exemple :

```
      ORG $0000
TABLE FDB $0102, $F1F2
```

FCC Réserve d'emplacements mémoire pour des caractères ASCII.
Exemple :

```
Message FCC 'Bonjour'
```

ORG ORiGin
Exemple :

```
      ORG $FE00    ;pointeur de programme en début de RAM programme (mode bootstrap)
```

RTS ReTurn from Subroutin : retour de sous-routine

END fin de programmer

Instructions d'assemblage

LDAA, LDAB
Exemple : ldaa #$05 ; mettre dans l'accu a la valeur 05 en héxadécimal

LDX, LDY
Exemple : ldx $1000 ; mettre dans x l'adresse $1000 en héxadécimal

STAA, STAB
Exemple : staa PortA ; copie la valeur contenue dans accu a dans le PORTB

STX, STY
Exemple : stx $2002 ; copie la valeur contenue dans accu x dans la mémoire $2000

INCA, INCB INCrement A, B : incrémentation de A, B

DECA, DECB DECrement A, B : décrémentation A, B

DEX, DEY DEcrement X, Y

INX, INY Increment X, Y

CLRA, CLRB Mise à zéro de accu A, accuB

Instructions de saut

BNE Branch if Not Equal : saut, si le résultat de l'opération précédente est différent de zéro, à l'adresse spécifiée en argument.

BEQ Branch if Equal : idem BNE mais si résultat de l'opération précédente nul.

BRA BRAnch always : branchement inconditionnel

BSR Branch to SubRoutine : saut à sous-routine. Saut court : la différence entre l'adresse de la sous-routine et le point de départ ne doit pas dépasser 256 octets

JSR Jump to SubRoutine : idem BSR mais saut long.

BRCLR BRanch if Bit CLeaR : saut si le bit du registre système donné en argument, accédé par un masque également donné en argument, est à 1

BRSET Idem BRCLR mais si bit à 1.

Opérations avec la pile

LDS LoaD Stack pointeur : chargement du pointeur de pile

PSHX, PSHY PuSH X, PuSH Y : sauvegarde sur la pile des registres X ou Y. Utile dans le cas où l'on souhaite utiliser ces registres, en restituant leur valeur initiale après cette utilisation.

PULX, PULY PULl X, PULl Y : opération inverse de PSHX et PSHY

Gestion des interruptions

SEI : Interdit toute interruption masquable, met à 1 le bit I du CCR qui interdit toute interruption masquable

CLI : Met à 0 le bit I du CCR : autorise les interruptions masquables

RTI : Retour d'interruption avec restauration du contexte à la fin du sous programme d'interruption

WAIT : Attente d'interruption après sauvegarde du contexte, le 68HC11 attend une interruption. Nous avons préféré dans les programmes d'exemple utiliser une boucle infinie pour attendre une interruption.

b) Modes d'adressage

Inhérent

Exemple :

```
CLRA  ; mettre l'accu A à zero.
ABA         ;(accu A+accu B)=> accu A
```

Immédiat

La donnée est une constante #.
Exemple :

```
ANDA #%01011000     ;ET logique entre l'accu. A et la valeur argument
```

Direct

L'adresse de la donnée se fait sur 1 octet. $00 à $FF.
Exemple :

```
LDAA $01      ; charge le contenu de la mémoire d'adr. $01 dans l'acc. A.
```

Etendu

L'adresse de la donnée se fait sur 2 octets. (16 bits). Exemple :

```
STAB $1000 ;stocke B à l'adresse $1000
```

Indexé

Le code est suivi d'une déplacement par rapport à une adresse contenue dans le registre X ou Y.

```
LDAA 2,Y      ;stocke la donnée contenue à l'adresse Y+2 dans l'accu A
LDAA 0,X
LDAA X        ;stocke la donnée contenue à l'adresse X dans A.
```

Relatif

Appliqué aux branchements.

```
CLRB          ;B est à 0
```

```
CONT INC B     ;B=B+1
CMPB #100      ;compare B avec la valeur décimale
BNE CONT       ;si B .100, il branche à CONT
```

Bit

Mode position bit

```
BCLR PORT B %01100001 ; positionne un 0 sur les bits 0, 5 et 6 du port B
```

Mode test de bit et branchement

```
BRCLR PORTB $80 BOUCLE    ;regarde si le bit 7 du port B est à zéro
                          ;($80=%10000000)
                          ;si vrai, elle va à BOUCLE,
                          ;sinon elle continue le programme
```