Sous programmes en assembleur 68hc11 pour maquettes 68hc11

J. Weiss
Équipe ETSCM
SUPÉLEC Campus de Rennes

1 Introduction

Ce document présente des programmes et des sous-programmes permettant l'exploitation des diverses fonctionnalités de la maquette 68hc11 (origine : ERP562 et 565), telles que :

Affichage : LCD de caractères ASCII (afficheur de 4 lignes de 20 caractères)
LED 7 segments de 2 digits
LED 3 mm (LED 1, LED 2 et LED3)
Saisie : clavier 16 touches
boutons poussoirs (2)
Gestion d'interruptions temps réel (horloge)

2 Organisation de la maquette

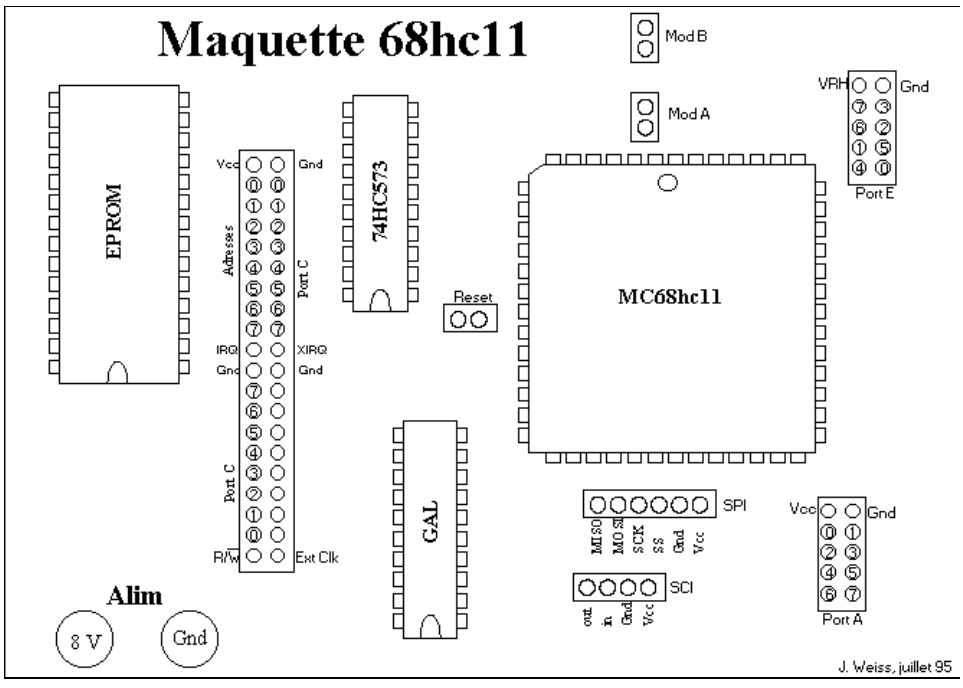
2.1 Implantation de la carte mère

La carte mère comprend le microcontrôleur, un décodeur, un tampon d'adresse ainsi qu'un emplacement pour une mémoire EEPROM de type 27C256.

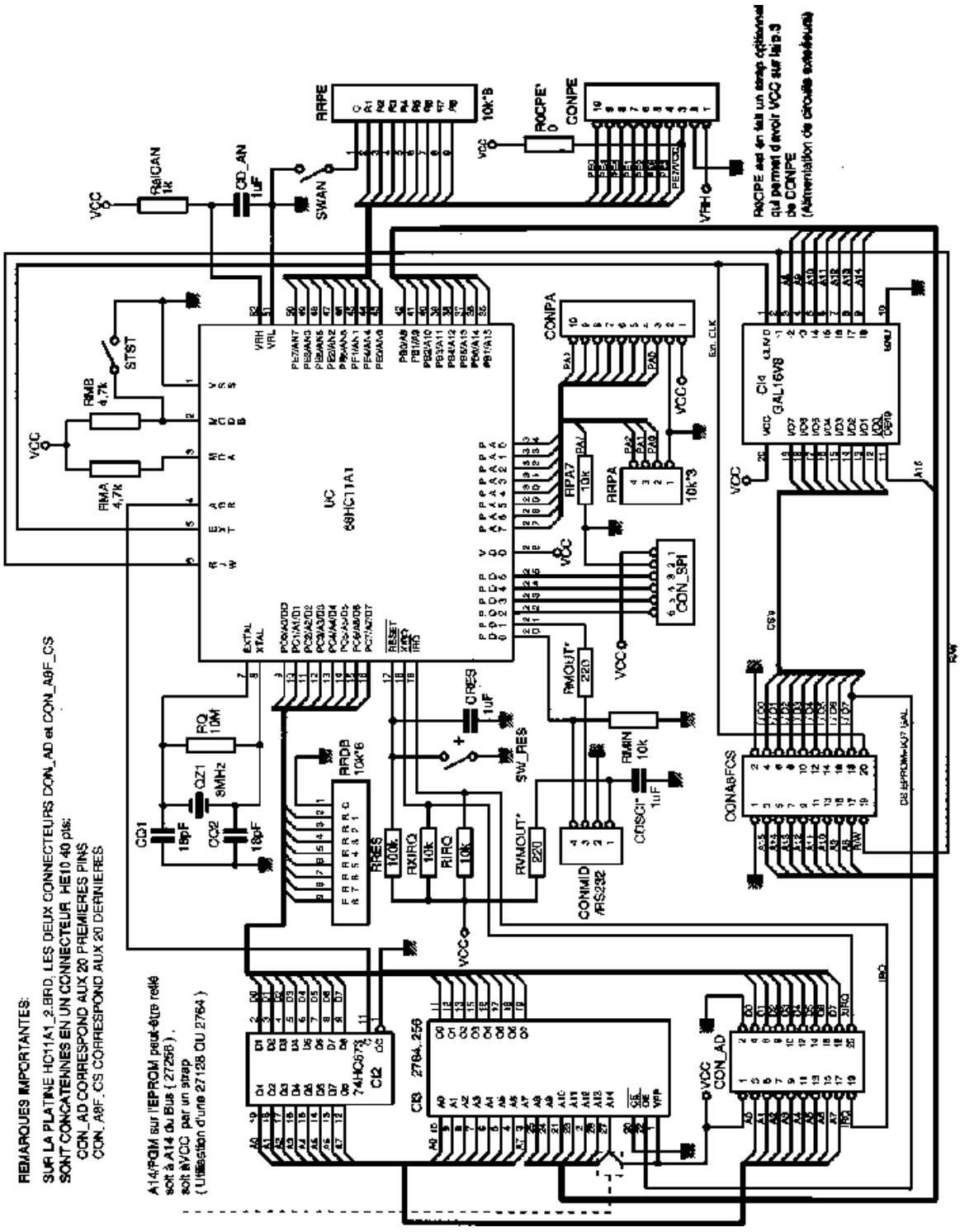
En mode *bootstrap* et en mode *normal*, les connecteurs de la carte permettent de disposer de tous les ports de sortie du microcontrôleur; en mode *Étendu* le connecteur central de la carte permet la connexion d'une carte comprenant une extension mémoire et un module d'E/S parallèles (2*8 bits); le mode de fonctionnement du microcontrôleur est déterminé par la position des cavaliers ModA et ModB au moment du Reset :

Mode *BootStrap* : les 2 cavaliers doivent être en place
Mode *Normal* : seul le cavalier ModB doit être en place
Mode *Étendu* : aucun des cavaliers ne doit être installé

L'implantation de la carte avec les différents brochages est donnée par la figure suivante :



2.2 Schéma électrique de la carte mère



2.3 espace d'adressage

Le microcontrôleur est capable d'adresser 64 kOctets (adresse sur 16 bits) en mémoire, il peut s'agir de RAM, de ROM (masquée, EPROM ou EEPROM) ou de registres de configuration; les ressources peuvent être internes ou externes, en cas de conflit (plusieurs entités à la même adresse), les ressources internes sont prioritaires. En mode *Bootstrap*, et en mode *Normal*, il n'est possible d'exploiter que les ressources internes.

2.3.1 Ressources internes

Type	68hc11A1	68hc811E2	Taille
Registres	$1000-$103F		64 registres
RAM	$0000-$00FF		256 Octets
ROM Boot	$BF40-$BFFF		192 Octets
ROM Interne	$E000-$FFFF	-	8kOctets/-
EEPROM	$B600-$B7FF	$F800-$FFFF	512 Octets/ 2kOctets

2.3.2 Mode Étendu

En mode *Étendu* (uniquement pour 68hc11A1), la table de l'espace adressable définie par la carte est donnée par le tableau ci-après :

Adresse Hexa	A15-A12 A11-A8 A7-A4 A3-A0	**CONTENU**	
$0000-$00FF	%0000 0000 0000 0000	256 OCTETS RAM	
$0100	%0000 0001 0000 0000		
$1000-$103F	%0001 0000 0000 0000	64 REGISTRES	- PORTS
$1040	%0001 0000 0100 0000		- REGISTRES DE DIR. DES PORTS
$2000	%0010 0000 0000 0000	RAM Externe : 8k Octets	- REGISTRES DATA/CONTROL TIMER
$3000	%0011 0000 0000 0000		- REGISTRES DATA/CONTROL SPI
$4000	%0100 0000 0000 0000		- REGISTRES DATA/CONTROL SCI
$5000	%0101 0000 0000 0000		- REGISTRES DATA/CONTROL A/D
$6000	%0110 0000 0000 0000		- REGISTRES DE CONFIGURATION
$7000	%0111 0000 0000 0000		
$8000	%1000 0000 0000 0000		
$9000	%1001 0000 0000 0000		
$A000	%1010 0000 0000 0000		
$B000-$B5FF	%1010 0000 1000 0000		
$B600-$B7FF	%1011 0000 0000 0000	512 OCTETS EEPROM	
$B800	%1011 1000 0000 0000		
$C000	%1100 0000 0000 0000	EPROM PROGRAMME	- ITs SCI ($FFD6, $FFD7)
$D000	%1101 0000 0000 0000	EPROM PROGRAMME	- IT SPI ($FFD8, $FFD9)
$E000	%1110 0000 0000 0000	EPROM PROGRAMME	- ITs TIMER
$F000	%1111 0000 0000 0000	EPROM PROGRAMME	- ITs EXTERIEURES
$FFC0	%1111 1111 1100 0000	VECTEURS IT & RESET	**- RESET ($FFFE, $FFFF)**

2.4 Vecteurs d'interruption

Les adresses et la gestion des vecteurs d'interruption dépendent du type de processeur et du mode de fonctionnement :

2.4.1 Mode bootstrap

En mode *Bootstrap*, le vecteur d'initialisation (Reset) pointe vers la ROM boot (adresse $BF40), ce qui lance une routine (Bootloader) de téléchargement d'un programme de 256 Octets (exactement !) en RAM; l'exécution de ce programme démarre après le chargement du 256ème octet, il peut s'agir, par exemple, du "Talker" utilisé par le moniteur (PCBug11).

Il est ensuite possible d'exécuter un programme à partir de l'EEPROM (adresse $B600 ou $F800); la gestion des vecteurs d'interruption est alors effectuée par des appels de sous-programmes placés en haut de la RAM (adresses $C4 à $FF). À titre d'exemple, le gestion d'une interruption RTI se fait en écrivant, à partir de l'adresse $EB le mnémonique du saut vers le sous-programme sur 3 Octets :

Gestion RTI : $EB : $7E (Code JMP) $EC-$EE : @ SP (2 octets).

Pour exécuter un programme en mode *Bootstrap* sous le contrôle de PCBug11, il faut placer les variables de programme et la pile dans plage d'adresses $B0-$C3, soit 20 octets; il est également possible d'exploiter l'espace des interruptions qui ne sont pas validées (plage $C4-$FF). Le tableau suivant donne la carte de la mémoire RAM en mode *Bootstrap* :

Adresses	Taille	Type	Source d'interruption	Masque	
	(Octets)			Global	Local
$0000-$00AF	175	Prog. (PCBug11)			
$00B0-$00C3	20	Dispo.			
$00C4-$00C6	3	Int. (PCBug11)	SCI : Liaison série asynchrone	I	
$00C7-$00C9	3	Int.	SPI : Liaison série synchrone	I	SPIE
$00CA-$00CC	3	Int.	Entrée du compteur	I	PAII
$00CD-$00CF	3	Int.	Débordement du compteur	I	PAOVI
$00D0-$00D2	3	Int.	Débordement du timer	I	TOI
$00D3-$00D5	3	Int.	OC5 : Comparateur 5	I	OC5I
$00D6-$00D8	3	Int.	OC4 : Comparateur 4	I	OC4I
$00D9-$00DB	3	Int.	OC3 : Comparateur 3	I	OC3I
$00DC-$00DE	3	Int.	OC2 : Comparateur 2	I	OC2I
$00DF-$00E1	3	Int.	OC1 : Comparateur 1	I	OC1I
$00E2-$00E4	3	Int.	IC3 : Entrée de capture 3	I	IC3I
$00E5-$00E7	3	Int.	IC2 : Entrée de capture 2	I	IC2I
$00E8-$00EA	3	Int.	IC1 : Entrée de capture 1	I	IC1I
$00EB-$00ED	3	Int.	RTI : Interruption temps réel	I	RTII
$00EE-$00F0	3	Int.	IRQ ou STRA	I	non/STAII
$00F1-$00F3	3	Int. (PCBug11)	XIRQ	X	
$00F4-$00F6	3	Int. (PCBug11)	SWI : interruption logicielle		
$00F7-$00F9	3	Int.	Code illegal		
$00FA-00FC	3	Int.	COP : Chien de garde		NOCOP
$00FD-$00FF	3	Int.	Défaut d'horloge		CME

2.4.2 Mode Normal

Le fonctionnement en mode *Normal* dépend du processeur employé :

68hc11A1 :

À l'initialisation, le processeur lance une routine en ROM (adresse $E000) qui va tester le bit de poids faible du port E, si celui-ci est à 1, alors on saute à l'adresse $B600 (départ de l'EEPROM), sinon le processeur continue l'exécution à partir de la ROM (programme de test dont on n'a pas la maîtrise).

Ainsi (bis repetita), pour lancer un programme à partir de l'EEPROM en mode *Normal* sur un microcontrôleur 68hc11A1, il faut placer le bit E0 (LSB port E) à un niveau logique 1.

La gestion des vecteurs d'interruption est alors la même qu'en mode *Bootstrap*, c'est-à-dire à partir de la RAM (plage $C4-$FF). La plage RAM disponible pour le programme est alors : $00-$C3 (195 Octets).

68hc811E2 :

Sur ce processeur, la zone EEPROM (plage : $F800-$FFFF) couvre la zone des vecteurs d'interruption (plage $FFD6-$FFFF), l'utilisateur est alors maître de la définition des appels à sous-programmes d'interruption (2 octets définissant l'adresse de la routine); les adresses de ces vecteurs sont celles définies pour le mode *Étendu*.

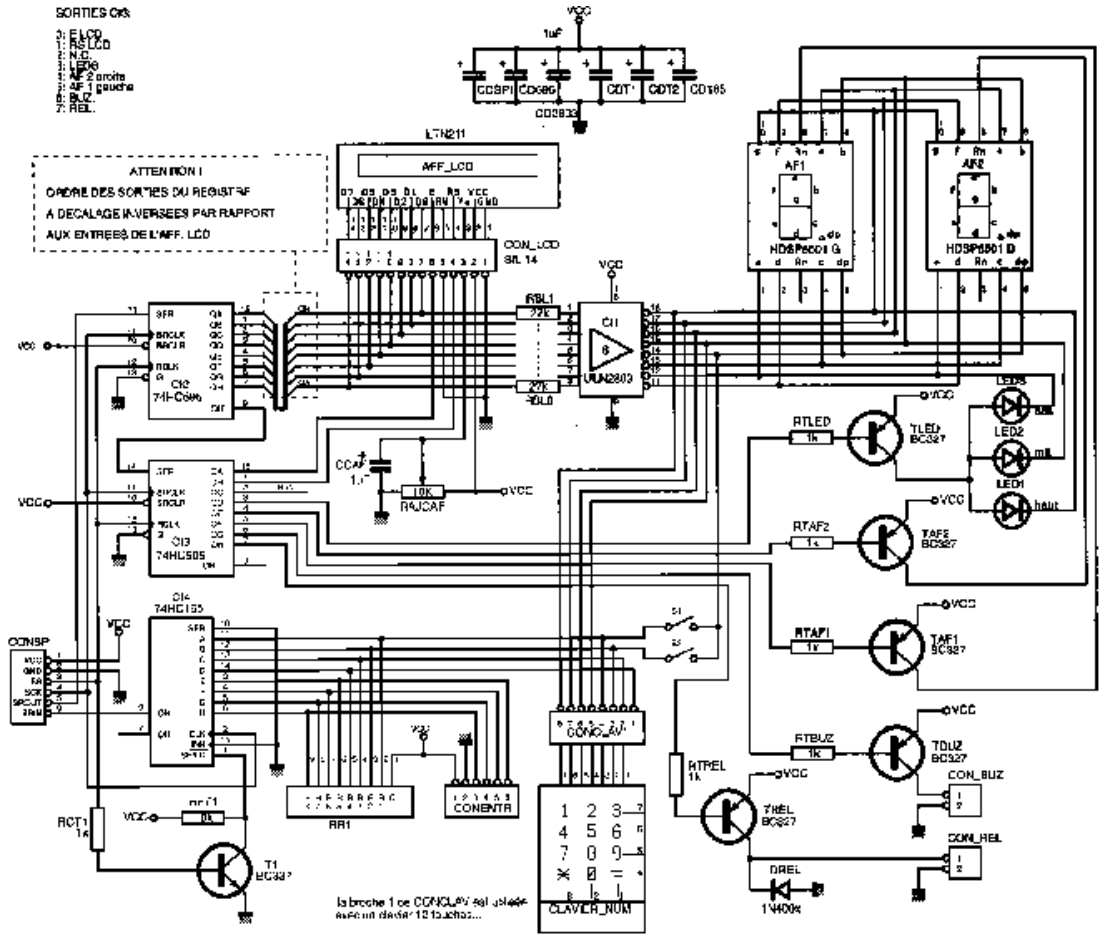
2.4.3 Mode Étendu

Dans ce mode, la gestion des vecteurs d'interruption est commune à tous le types de processeurs, ils sont définis par le tableau ci-après :

Adresses	Source d'interruption	Masque	
		Global	Local
$FFD6-$FFD7	SCI : Liaison série asynchrone	I	
$FFD8-$FFD9	SPI : Liaison série synchrone	I	SPIE
$FFDA-$FFDB	Entrée du compteur	I	PAII
$FFDC-$FFDD	Débordement du compteur	I	PAOVI
$FFDE-$FFDF	Débordement du timer	I	TOI
$FFE0-$FFE1	OC5 : Comparateur 5	I	OC5I
$FFE2-$FFE3	OC4 : Comparateur 4	I	OC4I
$FFE4-$FFE5	OC3 : Comparateur 3	I	OC3I
$FFE6-$FFE7	OC2 : Comparateur 2	I	OC2I
$FFE8-$FFE9	OC1 : Comparateur 1	I	OC1I
$FFEA-$FFEB	IC3 : Entrée de capture 3	I	IC3I
$FFEC-$FFED	IC2 : Entrée de capture 2	I	IC2I
$FFEE-$FFEF	IC1 : Entrée de capture 1	I	IC1I
$FFF0-$FFF1	RTI : Interruption temps réel	I	RTII
$FFF2-$FFF3	IRQ ou STRA	I	non/STAII
$FFF4-$FFF5	XIRQ	X	
$FFF6-$FFF7	SWI : interruption logicielle		
$FFF8-$FFF9	Code illegal		
$FFFA-$FFFB	COP : Chien de garde		NOCOP
$FFFC-$FFFD	Défaut d'horloge		CME
$FFFE-$FFFF	RESET		

2.5 Schéma électrique de la carte Clavier-Affichage

Cette carte sert d'interface homme-machine avec la carte mère, elle dispose d'un afficheur LCD de 4 lignes de 20 caractères, de 2 digits LED et de 3 LED 3 mm; la saisie peut se faire par un clavier de 16 touches et par 2 boutons poussoirs.



shéma de la carte clavier-affichage.

2.6 Transfert SPI

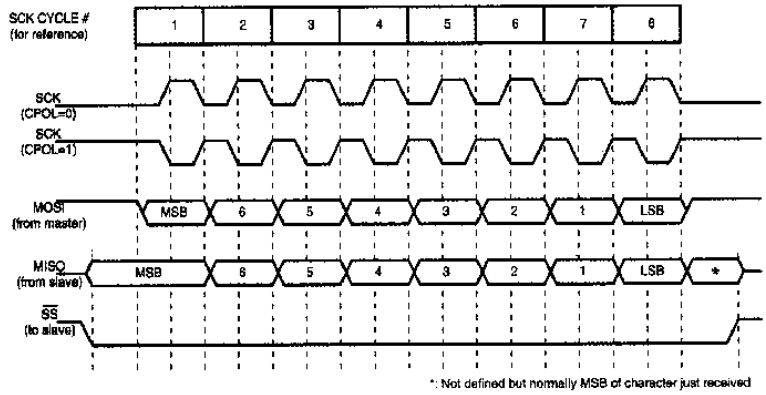
La particularité des maquettes est d'utiliser l'interface série synchrone (SPI) pour dialoguer entre la carte processeur et la carte clavier-affichage; les transferts sont organisés sous forme de 2 envois vers la carte Clavier-afficheur et d'une lecture.

```
*
* Premier mot envoyé :            Q7   Q6   Q5   Q4   Q3   Q2   Q1   Q0
*                                | R  | B  | D  | U  | L  | NC | RS | E  |
*                      Relais ____|    |    |    |    |         |    |
*                      Buzzer _________|    |    |    |         |    |
*                                           |    |    |         |    |
*   Afficheur 7 Seg.   Dizaines ____________|    |    |         |    |
*                      Unités _________________|    |         |    |
*                                                  |         |    |
*                      LED ________________________|         |    |
*                                                            |    |
*   Afficheur LCD      Register Select ______________________|    |
*                      Enable ___________________________________|


*
```

```
* Le deuxième mot envoyé correspond aux données
*
*      Afficheur 7 Seg. :          Q7   Q6   Q5   Q4   Q3   Q2   Q1   Q0
*                                | d  | e  | c  | dp | b  | a  | f  | g  |
*
*      Afficheur LCD :             Q7   Q6   Q5   Q4   Q3   Q2   Q1   Q0
*                                | q7 | q6 | q5 | q4 | q3 | q2 | q1 | q0 |
*
*
*                                  Q7   Q6   Q5   Q4   Q3   Q2   Q1   Q0
*           LED      :           |    | k3 |    |    | k2 |    |    | k1 |
*         Clavier    :           |    |    |    |    | r4 | r3 | r2 | r1 |
*      Interrupteurs :           |    |    |    | S  |    |    |    |    |
*
*
*    Commandes Afficheur LCD     RS (Register Select) : 0 -> Instr. Reg.
*                                                       1 -> Data Reg.
```

Le protocole SPI est indiqué par la figure suivante (N.B. : le signal SS doit être géré par logiciel) :



chronogrammes de l'interface SPI.

2.7 Registres internes du microcontrôleur

La liste de la plupart des registres de contrôle internes avec leur adresse (relativement à $1000) est donnée ci-après :

```
porta           equ 0           ; Port A (3 in, 5 out)
pioc            equ 2           ; Parallel I/O C Control Reg.
portc           equ 3           ; Port C
portb           equ 4           ; Port B (output only)
portcl          equ 5           ; Port C Control
ddrc            equ 7           ; Data Direction Reg Port C
portd           equ 8           ; Port D
ddrd            equ 9           ; Data Direction Reg Port D
porte           equ $A          ; Port E

cforc           equ $B          ; Compare Force Register
oc1m            equ $C          ; OC1 Action Mask Register
oc1d            equ $D          ; OC1 Action Data Register
tcnt            equ $E    ; (16 bits) : Timer Counter Register
tic1            equ $10   ; (16 bits) : Input Capture 1 Register
tic2            equ $12   ; (16 bits) : Input Capture 2 Register
tic3            equ $14   ; (16 bits) : Input Capture 3 Register
toc1            equ $16   ; (16 bits) : Output Compare 1 register
toc2            equ $18   ; (16 bits) : Output Compare 2 register
toc3            equ $1A   ; (16 bits) : Output Compare 3 register
toc4            equ $1C   ; (16 bits) : Output Compare 4 register
toc5            equ $1E   ; (16 bits) : Output Compare 5 register
tctl1           equ $20         ; Timer Control Register 1
tctl2           equ $21         ; Timer Control Register 2
tmsk1           equ $22         ; Timer Mask Register 1
tflg1           equ $23         ; Timer Flag Register 1
tmsk2           equ $24         ; Timer Mask Register 2
tflg2           equ $25         ; Timer Flag Register 2
pactl           equ $26         ; Pulse Accumulator Control Reg.

spcr            equ $28         ; SPI Control Register
spsr            equ $29         ; SPI Status Register
spdr            equ $2A         ; SPI Data Register
```

```
baud            equ $2B        ; SCI Baud Rate Control
sccr1           equ $2C        ; SCI Control Register 1
sccr2           equ $2D        ; SCI Control Register 2
scsr            equ $2E        ; SCI Status Register
scdr            equ $2F        ; SCI Data Register

adctl           equ $30        ; A/D Control/Status Register
adr1            equ $31        ; A/D Result Register 1
adr2            equ $32        ; A/D Result Register 2
adr3            equ $33        ; A/D Result Register 3
adr4            equ $34        ; A/D Result Register 4

bprot           equ $35        ; Block Protection
option          equ $39        ; System Configuration Options
coprst          equ $3A        ; Arm/Reset COP Timer circuitry
pprog           equ $3B        ; EEPROM Programming Register
hprio           equ $3C        ; Highest Priority Int. Reg.
config          equ $3f        ; COP, ROM & EEPROM enables
```

2.8 Constantes de la carte

```
*************************************************************
*              Constantes du programme                      *
*************************************************************

base_reg        equ $1000

** mots de commande (premier mot envoyé sur SPI)
RIEN            equ $FC  ; aucun affichage
LCD_on          equ $FF  ; affichage LCD
LCD_config      equ $FD  ; configuration de l'afficheur LCD
LED_on          equ $F6  ; sélection des LED
unites          equ $DE  ; sélection de l'afficheur des unités
dizaines        equ $EE  ; sélection de l'afficheur des dizaines
buzzer          equ $BE  ; sélection du buzzer
relais          equ $7E  ; sélection du relais
switches        equ $10  ; interrupteurs

** mots de donnée (deuxième mot envoyé sur SPI)
LED_k1          equ $1   ; cathode de la LED 1
LED_k2          equ $8   ; cathode de la LED 2
LED_k3          equ $40  ; cathode de la LED 3
```

3 Programmes assembleur

Les programmes et sous-programmes présentés ci-dessous ont pour but de fournir des utilitaires de développement pour les maquettes 68hc11; ces programmes sont présentés comme des modules élémentaires, les variables exploitées sont rapellées en début de listing (en commentaire); sauf mention contraire, l'utilisation des sous-programmes n'entraine de modification du contexte (accus, registres, ...). En plus des procédures d'initialisation, sont présentés les modules suivants :

Aff_msg(Y) : affiche la chaîne de caractères pointée par Y sur l'afficheur LCD)
variable modifiée : data_in

Aff_7seg(digit1,digit10) : affiche digit1 et digit10 sur les afficheurs 7 segments LED
variable modifiée : data_in

Aff_LED(A) : allume la diode LED définie par l'accumulateur A (LED_k#)

Lect_clavier() : effectue une lecture des touches du clavier et des interrupteurs
Valeur retournée : retour : touche : valeur du clavier (4 bits) ou $FF si rien
inter : état des interrupteurs (0 : S1, 1 : S2 et $FF : rien)

En annexe de ce document, sont fournis 2 programmes qui ont servi à bâtir ces modules utilitaires, ils ont exactement la même fonctionnalité (affichage d'un message et de l'heure) mais ils sont utilisés sur 2 processeurs différents :

LCD_Time.ASC : microcontrôleur 68hc811E2 en mode *Normal*
LCD_A1.asc : microcontrôleur 68hc11A1 en mode *Bootstrap* (PCBug11) et en mode *Normal*

3.1 Initialisation

3.1.1 Index, Pile et Vecteurs

3.1.1.1 Mode Bootstrap

```
*************************************************************
*                variables du programme                     *
*************************************************************
            ORG $00B0
variable1   RMB 1
...
*************************************************************
*                constantes du programme                    *
*************************************************************
JUMPEXT     equ $7E             ; mnémonique de l'instruction JMP
JRTI        equ $00EB           ; adresse du vecteur RTI
...
            ORG  $F800          ; cas d'un 68hc811e2 ($B600 pour 68hc11a1)
Debut       SEI                 ; set interrupt mask
            LDAA #JUMPEXT       ; mnémonique de l'instruction JMP
            LDX  #Rti           ; adresse du sous-programme d'IT (ici : RTI)
            STAA JRTI           ; ecriture de JMP Rti à partir de l'adresse $EB
            STX  JRTI+1         ;

.... autres définitions de vecteurs

            LDX  #base_reg      ; X <--- $1000 : base des registres
            LDS  #$E7           ; initialisation de la pile à $E7

.... autres initialisation

            CLI
.... départ du programme
```

3.1.1.2 Mode Normal

68hc11A1 : même procédure qu'en mode ***Boostrap*** mais les variables peuvent être placées à partir de $0.
68hc811E2 : même procédure qu'en mode ***Étendu***

3.1.1.3 Mode Étendu

```
************************************************************
*                variables du programme                    *
************************************************************
           ORG $0000               ; cas de la RAM interne ($2000 pour la RAM externe)
variable1        RMB 1
...

                 ORG  $FFF0          ; vecteur RTI (exemple)
                 FDB  Rti            ; adresse du sous-programme d'IT (ici : RTI)
                 ORG  $FFFE          ; vecteur RESET
                 FDB  Debut          ; début du programme
                 ORG  $F800          ; cas d'un 68hc811e2
                                     ;($B600 pour 68hc11a1 ou $C000 pour EPROM externe)

Debut            SEI                 ; set interrupt mask

.... autres initialisation

                 CLI

.... départ du programme
```

3.1.2 SPI

```
************************************************************
*                 initialisation SPI                       *
************************************************************
Init_SPI         BSET ddrd,x,#$38    ; SS, SCK, MOSI en sortie
                 LDAA #$51           ; Enable SPI avec SCK=125kHz, wired_or mode
                 STAA spcr,x
                 LDAA #$3F           ; on met les bits du port D à 1
                 STAA portd,x
```

3.1.3 Afficheur LCD

```
************************************************************
*                 initialisation LCD                       *
************************************************************
*Commande_LCD    RMB 1  ; mot de commande du LCD (0: commande; 2 : affichage)
*LCD_config      equ $FD  ; configuration de l'afficheur LCD

Init_LCD         LDAA #LCD_config     ; mode config LCD
                 STAA Commande_LCD
                 LDAA #$1             ; clear Display
                 JSR  Aff_LCD

** l'afficheur a besoin d'une tempo
Tempo               LDY #$1000           ; IY <- durée
T1               DEY                  ; IY <- IY-1
                 BNE T1

                 LDAA #$E             ; display =on
                 JSR  Aff_LCD
                 LDAA #$3C            ; N =2 lignes; F =  1
                 JSR  Aff_LCD
```

3.2 Affichage

3.2.1 Procédure d'appel d'affichage LCD

```
                LDY  #introduction
                JSR  Aff_msg
...
introduction    FCC ' On affiche ce texte\'
```

3.2.2 Préambule d'affichage LCD

```
*****************************************************************
* SP *           préparation de l'affichage LCD                 *
*****************************************************************
*Commande_LCD    RMB 1  ; mot de commande du LCD (0: commande; 2 : affichage)
Aff_LCD         PSHB
                LDAB Commande_LCD
                JSR  OUTSPI          ; on envoie la commande (E=1)
                ANDB #$FE            ; Disable
                JSR  OUTSPI          ; on renvoie la commande (E=0)
                PULB
                RTS
```

3.2.3 Affichage d'une chaîne de caractères sur LCD

```
*****************************************************************
* SP * envoi d'une chaine de caractères pointée par y finie par \ *
*****************************************************************
Aff_msg         PSHA
                PSHY
                LDAA #LCD_config       ; mode config LCD
                STAA Commande_LCD
                LDAA #$80              ; DD Ram =$00
                JSR  Aff_LCD
                LDAA #LCD_on           ; mode Affichage LCD
                STAA Commande_LCD
Envmot          LDAA 0,y
                CMPA #'\'
                BEQ  Envmot_fin        ; si \ alors retour
                JSR  Aff_LCD           ; si non envoi du caractère (0,y)
                INY                    ; IY <- IY + 1
                BRA  Envmot            ; on boucle
Envmot_fin      PULY
                PULA
                RTS
```

3.2.4 exemple : affichage de l'heure sur l'afficheur LCD

```
*************************************************************************
* SP *        Affichage de l'heure sur l'afficheur LCD                   *
*************************************************************************
*seconde          RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
*secondes         RMB 1    ; secondes de 0 à  59
*minutes          RMB 1    ; minutes de 0 à 59
*heures           RMB 1    ; heures de 0 à 23

Affhm           LDAA  #LCD_config     ; mode config LCD
                STAA  Commande_LCD
                LDAA  #$C7               ; DD Ram =$47
                JSR   Aff_LCD
                LDAA  #LCD_on                ; RS  = 1
                STAA  Commande_LCD
                LDAA  heures
                JSR   Aff_ascii
                LDAA  #':'
                JSR   Aff_LCD
                LDAA  minutes
                JSR   Aff_ascii
                LDAA  #':'
                JSR   Aff_LCD
                LDAA  secondes
                JSR   Aff_ascii
                RTS

** envoi en DECIMAL ASCII du nombre contenu dans A

Aff_ascii       PSHB                   ; sauvegarde de B
                JSR  Div_10            ; division par 10 de A
                STAA digit10
                STAB digit1
```

```
                ADDA #$30               ; A <- A + $30
                JSR  Aff_LCD            ; envoi du digit Quotient (dizaines)
                TBA                     ; A <- B
                ADDA #$30               ; B <- B + $30
                JSR  Aff_LCD            ; envoi du digit Reste (unités)
                PULB pulb               ; restauration de B
                RTS

** DIVISION PAR 10  ----------------------------
** donnee dans A -> A/10 ds A et reste ds B

Div_10          TAB                 ; B <-- A
                CLRA                ; A <- 0  : variable dans AccD
                LDX  #10            ; ix <- 10
                IDIV                ; Q:IX ; R:AccD <- AccD/IX
                PSHB                ; push B : Pile <- MSB du reste
                XGDX                ; Accd <-> IX
                TBA                 ; A <-- B
                PULB                ; pull B : B <- MSB du reste
                LDX  #base_reg      ; réinitialisation de IX ($1000)
                RTS                 ; return
```

3.2.5 Affichage de 2 digits LED 7 segments

```
*************************************************************************
* SP *        Affichage sur les afficheurs 7 segments LED                *
*                  digit1 correspond aux unités                         *
*                 digit10 correspond aux dizaines                       *
*************************************************************************
*digit1           RMB 1    ; digit des unités de l'afficheur LED
*digit10          RMB 1    ; digit des dizaines de l'afficheur LED

Aff_7seg        PSHY
                PSHB
                PSHA
                LDY  #Tab_7seg    ; on pointe sur la table de conversion
                LDAA c244         ; chargement du compteur d'horloge c244
                ASRA              ; on propage le LSB dans C
                BCC  Aff2         ; c244 est pair (C=0), on affiche les unités
                LDAB #dizaines    ; adresse du digit à afficher
                PSHB
                LDAB digit10      ; chargement du digit à afficher
                BRA  Fin_7seg

Aff2            LDAB #unites      ; adresse du digit à afficher
                PSHB
                LDAB digit1       ; chargement du digit à afficher

Fin_7seg        ABY               ;  Y <-- Y + B ( B :offset dans la table)
                LDAA 0,y          ; pointage sur la table
                PULB              ; on récupère l'adresse du digit
                JSR  OUTSPI       ; envoi du digit
                PULA
                PULB
                PULY
                RTS

*     segment :       dec.bafg
Tab_7seg        FCB #%11101110          ; chiffre 0
                FCB #%00101000          ; chiffre 1
                FCB #%11001101          ; chiffre 2
                FCB #%10101101          ; chiffre 3
                FCB #%00101011          ; chiffre 4
                FCB #%10100111          ; chiffre 5
                FCB #%11100111          ; chiffre 6
                FCB #%00101100          ; chiffre 7
                FCB #%11101111          ; chiffre 8
                FCB #%10101111          ; chiffre 9
                FCB #%01101111          ; chiffre A
                FCB #%11100011          ; chiffre b
                FCB #%11000001          ; chiffre c
                FCB #%11101001          ; chiffre d
                FCB #%11000111          ; chiffre E
                FCB #%01100111          ; chiffre F
```

3.2.6 Affichage sur les 3 diodes LED

```
*************************************************************************
* SP *                   Affichage  diodes LED                          *
*     Le numéro de la LED (LED_k#) se trouve dans l'accumulateur A      *
*************************************************************************
Aff_LED         PSHB
                LDAB #LED_on              ; mot de commande pour les diodes LED
                JSR OUTSPI
                PULB
                RTS
```

3.3 Transfert SPI

```
*************************************************************************
* SP *     Envoi de 2 octets par SPI et réception d'un octet            *
*            - Le mot de commande est dans l'accumulateur B             *
*            - Les données en émission  sont dans l'accumulateur A      *
*            - la donnée en réception est dans data_in                  *
*************************************************************************
*data_in         RMB 1    ; lecture sur SPI

OUTSPI          PSHA                    ; on empile la donnée
                PSHB                    ; on empile le mot de commande
                PSHY
                BCLR portd,x,#$20       ; met SS à 0 latche entrée et sortie registres
                STAB spdr,x             ; envoi du mot de commande

                LDY #08                 ; 8 bits à faire tourner
RBBMIR          ROLA                    ; les bits sortent à gauche par la retenue
                RORB                    ; et rentrent à droite par la carry
                DEY
                BNE RBBMIR
ATE20           BRCLR spsr,x,#$80 ATE20  ; attend fin transmission prec.
                LDAA  spdr,x             ; on récupère la donnée sur le SPI
                STAA  data_in           ; enregistre
                STAB  spdr,x                ; on envoie la donnée sur le SPI
ATEF2O          BRCLR spsr,x,#$80 ATEF2O ; attend fin transmission pour reset SS
                BSET  portd,x,#$20       ; met SS à 1 (delatche sorties registres)
                PULY
                PULB                     ; on restaure le mot de commande dans B
                PULA                     ; on restaure la donnée dans A
                RTS
```

3.4 Saisie du clavier

```
*****************************************************************************
* SP *             Lecture du clavier et des interrupteurs                  *
*            touche : lecture du clavier (4 bits ou $FF si rien)            *
*        inter : état des interrupteurs (0 : S1, 1 : S2 et $FF : rien)      *
*****************************************************************************
*touche           RMB 1    ; touche frappée sur le clavier (4 bits ou $FF si rien)
*inter            RMB 1    ; interrupteurs (0 : S1, 1 : S2 et $FF : rien)

Lect_clavier    PSHA
                PSHB
                PSHY
Init_lect       LDAA #$FF
                STAA inter
                STAA touche
                LDAA #1                 ; on pointe sur la 1ère rangée
                LDAB #RIEN              ; mot de commande nul
                BRA Lect_2
Lect_1          ASLA                    ; rangée suivante
                BEQ Fin_lect            ; fin du balayage des rangées
Lect_2          JSR OUTSPI              ; on stimule
                JSR OUTSPI              ; on réceptionne la donnée (4 bits LSB)
                BRSET data_in,#$0F,Lect_1 ; réponse vierge ?, si oui on boucle
                COM data_in             ; complémentation de la donnée
                CMPA #switches          ; adressage des interrupteurs ?
                BEQ Lect_int            ; il 'agit d'un interrupteur !
                JSR Lin_bin             ; conversion en binaire (résultat dans B)
                PSHB                    ; sauvegarde du résultat "rangée"
                LDAA data_in            ; réponse "colonne"
                JSR Lin_bin             ; conversion en binaire (résultat dans B)
                ASLB
                ASLB
                PULA                    ; récupération du résultat "rangée"
                ABA                     ; mise en forme de la réponse clavier
                TAB
                LDY #Tab_clavier
                ABY
                LDAA 0,Y
                STAA touche
Fin_lect        PULY
                PULB
                PULA
                RTS

Lect_int        LDAA data_in
                RORA
                STAA inter
                BRA Fin_lect

Lin_bin         LDAB #0             ; compteur à zéro
Lin_1           RORA                ; on décale
                BCS Fin_lin         ; fin si on a propagé un 1 dans C
                CMPB #3             ; fin des 4 bits ?
                BEQ Fin_lin         ; si oui alors fin
                INCB                ; incrémentation du compteur
                BRA Lin_1           ; on boucle
Fin_lin         RTS

* on lit : 147A2580369BFEDC
Tab_clavier     FCB #$1
                FCB #$4
                FCB #$7
                FCB #$A
                FCB #$2
                FCB #$5
                FCB #$8
                FCB #$0
                FCB #$3
                FCB #$6
                FCB #$9
                FCB #$B
                FCB #$F
                FCB #$E
                FCB #$D
                FCB #$C
```

3.5 Gestion des interruptions temps réel (RTI)

3.5.1 Initialisation et préparation RTI

```
*************************************************************
*                  Début du programme                       *
*************************************************************
Debut           SEI                     ; set interrupt mask
                LDX  #base_reg          ; X <--- $1000
                LDS  #$FF               ; initialisation de la pile à FF

                BSET pactl,x,#0         ; rti rate : 4,10 ms
                BSET tmsk2,x,#$40       ; rti Interrupt Enable
...
*************************************************************
*   Départ du programme  *
*************************************************************
                LDY  #introduction
                JSR  Aff_msg
...
B1              SEI
...
                JSR Lect_clavier
                JSR Aff_7seg
                CLI                   ; Clear interrupt Mask
                WAI                   ; Attente d'une interruption
                JMP  B1
```

3.5.2 Gestion de l'interruption RTI : exemple de calcul de l'heure

```
*************************************************************
* INT *          Gestion de l'interruption périodique        *
* RTI *                calcul de l'heure                     *
*************************************************************
*c244             RMB 1    ; compte 244 interuptions RTI avant d'obtenir 1s.
*bis              RMB 1    ; indique à ff que l'on a ajouté le reste
*seconde          RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
*secondes         RMB 1    ; secondes de 0 à  59
*minutes          RMB 1    ; minutes de 0 à 59
*heures           RMB 1    ; heures de 0 à 23

** toutes les 256 s, on charge c244 avec le reste (16) pour ajuster l'horloge

Rti             DEC c244
                BEQ Rti_0
                JMP Fin_rti
Rti_0           BSET  c244,#244        ; recharge le compteur
                BRSET bis,#$ff,Rti_1   ; bis indique que l'on a déjà re-
                                       * chargé c244 avec le reste
                INC  seconde           ; inc sauf si bis = $ff
                                       * chaque fois que seconde passe à 0
                BNE  Rti_2             ; on charge bis à ff et c244 à 16 (reste)
                                       * on ne compte pas cette "seconde"
Rti_01          LDAA #16               ; on charge le reste
                STAA c244
                BSET bis,#$ff
                JMP  Fin_rti

Rti_1           CLR  bis
Rti_2           JSR  Affhm
                LDAA #60
                INC  secondes           ; secondes de 0 à 59
                CMPA secondes
                BNE  Fin_rti
                CLR  secondes
                INC  minutes
                LDAA #60
                CMPA minutes
                BNE  Fin_rti            ; minutes
                CLR  minutes
                INC  heures
                LDAA heures
                CMPA #24
                BNE  Fin_rti            ; heures
                CLR  heures

Fin_rti         BCLR tflg2,x,#$BF       ; annule flag RTI
                RTI
```

4 Annexes

4.1 Annexe 1 : Boot ROM en $BF40 (mode bootstrap)

Programme Boot ROM, placé en $BF40 (longueur : 192 octets)

```
portd           equ 8           ; Port D
spcr            equ $28         ; SPI Control Register
baud            equ $2B
sccr2           equ $2D         ; SCI Control Register 2
scsr            equ $2E         ; SCI Status Register
scdr            equ $2F         ; SCI Data Register

        LDS #$FF                ; initialisation de la pile à $FF
        LDX #$1000              ; on place l'index à $1000 (base des registres)
        BSET spcr,X,#$20        ;
        LDAA #$A2               ;
        STAA baud,X             ;
        LDAA #$0C               ;
        STAA sccr2,X
        BSET sccr2,X,#$01
Att1    BRSET portd,X,#$01,Att1
        BCLR sccr2,X,#$01
Att2    BRCLR scsr,X,#$20,Att2
        LDAA scdr,X
        BNE  Cont1
        JMP  $B600
Cont1   CMPA #$55
        BEQ Cont3
        CMPA #$FF
        BEQ Cont2
        BSET baud,X,#$33
 Cont2  LDY #$0000
 Att3   BRCLR scsr,X,#$20, Att3
        LDAA scdr,X
        STAA $00,Y
        STAA scdr,X
        INY
        CPY #$0100
        BNE Att3
Cont3   JMP $0000


* contenu mémoire pour les vecteurs d'interruption
*
* $BFD6-$BFD7    $00C4    SCI
* $BFD8-$BFD9    $00C7    SPI

* $BFDA-$BFDB    $00CA    PAI  : Entrée compteur
* $BFDC-$BFDD    $00CD    PAOV : Débordement du compteur
* $BFDE-$BFDF    $00D0    TO   : Débordement du timer

* $BFE0-$BFE1    $00D3    OC5
* $BFE2-$BFE3    $00D6    OC4
* $BFE4-$BFE5    $00D9    OC3
* $BFE6-$BFE7    $00DC    OC2
* $BFE8-$BFE9    $00DF    OC1
* $BFEA-$BFEB    $00E2    IC3
* $BFEC-$BFED    $00E5    IC2
* $BFEE-$BFEF    $00E8    IC1
* $BFF0-$BFF1    $00EB    RTI
* $BFF2-$BFF3    $00EE    IRQ
* $BFF4-$BFF5    $00F1    XIRQ
* $BFF6-$BFF7    $00F4    SWI
* $BFFA-$BFFB    $00FA    COP
* $BFFE-$BFFF    $BF40    Reset
```

4.2 Annexe 2 : Boot ROM en $E000 (mode normal)

* Programme Boot ROM, placé en $E000

```
portd            equ 8          ; Port D
spcr             equ $28        ; SPI Control Register
baud             equ $2B
sccr2            equ $2D        ; SCI Control Register 2
scsr             equ $2E        ; SCI Status Register
scdr             equ $2F        ; SCI Data Register

        LDX  #$100A             ; chargement de l'index à $100A
Att2    BRCLR 0,X,#$01,Cont1    ; si port E(0) = 0, alors on continue en ROM
        jmp $B600               ; sinon on saute en EEPROM ($B600)
Cont1   LDAA #$93
        STAA option+Base_reg
        LDAA #$00
        STAA tmsk2+Base_reg
        LDAA #$00
        STAA $bprot+Base_reg
        LDS  #$65
        JSR  $E34B
        LDX  $047
        STX  $6F
        LDAA tctl1+Base_reg
        ORAA #$3
        STAA tctl1+Base_reg
        LDAA #$D0
        STAA $6E
        LDD  #$3F0D
        STD  $71
        JSR  $E1A8
        CLR  $A6
        INC  $A6
        CLR  $A8
        LDAA hprio+Base_reg
        ANDA #$20
        BEQ  $E07D
        LDAA #$3

...


* contenu mémoire pour les vecteurs d'interruption (mode normal)
* $FFD6-$FFD7    $00C4    SCI
* $FFD8-$FFD9    $00C7    SPI

* $FFDA-$FFDB    $00CA    PAI  : Entrée compteur
* $FFDC-$FFDD    $00CD    PAOV : Débordement du compteur
* $FFDE-$FFDF    $00D0    TO   : Débordement du timer

* $FFE0-$FFE1    $00D3    OC5
* $FFE2-$FFE3    $00D6    OC4
* $FFE4-$FFE5    $00D9    OC3
* $FFE6-$FFE7    $00DC    OC2
* $FFE8-$FFE9    $00DF    OC1
* $FFEA-$FFEB    $00E2    IC3
* $FFEC-$FFED    $00E5    IC2
* $FFEE-$FFEF    $00E8    IC1
* $FFF0-$FFF1    $00EB    RTI
* $FFF2-$FFF3    $00EE    IRQ
* $FFF4-$FFF5    $00F1    XIRQ
* $FFF6-$FFF7    $00F4    SWI
* $FFFA-$FFFB    $00FA    COP
* $FFFE-$FFFF    $E000    Reset
```

4.3 Annexe 3 : Talker PC Bug11 (talkJW.asc)

```
************************* TALKjw.ASC juillet 95 ************************
*   MCU resident, Interrupt driven Communication routines for 68HC11     *
*   monitor. Provides low level memory and stack read/write operations.  *
*                                                                        *
* CONSTANTES
TALKBASE  equ $0000
BOOTVECT  equ $00C4       ; adresse du vecteur SCI
STACK     equ $00EB       ; Départ de la pile
REGBASE   equ $1000       ; base des registres
*
JSCI      equ $00C4       ; adresse du vecteur SCI
JXIRQ     equ $00F1       ; adresse du vecteur XIRQ (pas utilisé)
JSWI      equ $00F4       ; adresse du vecteur SWI
JILLOP    equ $00F7       ; adresse du vecteur illegal Op Code (pas utilisé)
JCOP      equ $00FA       ; adresse du vecteur illegal Op Code (pas utilisé)
JMPEXT    equ $7E         ; mnémonique de l'instruction JMP
BRKCODE   equ $4A         ; signal d'arrêt vers l'IBM-PC
BRKACK    equ $4A         ; reconnaissance du signal d'arrêt par l'IBM-PC
*
* REGISTRES
BAUD      equ $2B
SCCR1     equ $2C
SCCR2     equ $2D
SCSR      equ $2E
SCDR      equ $2F
*
RDRF      equ $20
TDRE      equ $80
OR        equ $08
FE        equ $02
*
* PROGRAM
          org TALKBASE
TLKRSTART EQU *           ; allocation de la table des sauts des vecteurs d'interruption
          LDAA #JMPEXT
          LDY #NULLSRV
          LDX #BOOTVECT
SETVECT   EQU *
          STAA ,X
          INX
          STY ,X
          INX
          INX
          CPX #$100
          BNE SETVECT
          LDX #SCISRV
          STX JSCI+1
          LDX #TLKRSTART
          STX JILLOP+1
*
USERSTART EQU *
          LDS #STACK
          LDX #REGBASE
          CLR SCCR1,X
          LDD #$302C
          STAA BAUD,X       ; initialise SCI à 9600 baud, sans parité ni interruption
          STAB SCCR2,X      ; validation de tx & rx.
          LDAA #$40         ; validation des interruptions STOP et I, inhibition de XIRQ
          TAP
*
IDLE      JMP IDLE          ; attente d'une interruption SCI (provenant de l'IBM-PC)
* une commande RESET de l'IBM-PC change cette adresse de saut vers l'adressse de
* départ de programme utilisateur
*
SCISRV    EQU *             ; détection de l'interruptio
          LDAA SCSR+REGBASE ; on suppose qu'elle vient du SCI
          ANDA #RDRF
          BEQ SCISRV        ; sinon on boucle
*
RXSRV     EQU *             ; Traitement de la donnée reçue
          LDAA SCDR+REGBASE ; lecture de l'octet de commande & echo de celui-ci
          COMA              ; inversion des données
          BSR OUTSCI        ; réexpédition vers l'IBM-PC
          BPL INH1          ; Bit 7 à 1, alors on traite la commande
          BSR INSCI         ; sinon on lit l'octet de comptage dans ACCB.(0=256)
          XGDX              ; X <--> Accd : sauve la commande et l'octet de comptage dans X
          BSR INSCI         ; lecture de la partie haute de l'adresse
          TBA               ; AccA <-- AccB
```

```
          BSR INSCI         ; lecture de la partie haute de l'adresse
          XGDX              ; X <--> Accd : récupère la commande et l'octet de comptage dans X
          CMPA #$FE
          BNE RXSRV1        ; s'agit-il d'une lecture mémoire ?
*
TREADMEM  EQU *          ; Lecture mémoire
          LDAA ,X           ; lecture à l'adresse concernée
          BSR OUTSCI        ; envoi à l'IBM-PC
          TBA               ; AccA <-- AccB : sauve l'octet de comptage
          BSR INSCI         ; attente de l'echo
          TAB               ; AccB <-- AccA : récupère l'octet de comptage
          INX               ; incrémentation de l'adresse
          DECB              ; décrémentation de l'octet de comptage
          BNE TREADMEM      ; boucle tant qu'il y a des cases à lire
          RTI               ; attente d'une nouvelle interruption
*
RXSRV1    EQU *
          CMPA #$BE
          BNE RXSRVEX       ; s'agit-il d'une écriture mémoire ?
*
          TBA               ; AccA <-- AccB : sauve l'octet de comptage
TWRITMEM  EQU *         ; Écriture mémoire
          BSR INSCI         ; lecture de l'octet suivant dans AccB
          STAB ,X           ; lecture à l'adresse concernée
          LDAB ,X           ; vérification de l'écriture
          STAB SCDR+REGBASE ; echo vers l'IBM-PC
          INX               ; incrémentation de l'adresse
          DECA              ; décrémentation de l'octet de comptage
          BNE TWRITMEM      ; boucle tant qu'il y a des cases à lire
RXSRVEX   EQU *
NULLSRV   RTI               ; attente d'une nouvelle interruption
*
INSCI     EQU *        ; Lecture sur le port série (SCI)
          LDAB SCSR+REGBASE ; attente de RDRF=1
          BITB #(FE+OR)     ; s'agit-il d'un break
          BNE TLKRSTART     ; si oui, on redémarre le talker
          ANDB #RDRF
          BEQ INSCI
          LDAB SCDR+REGBASE ; Lecture de la donnée provenant de l'IBM-PC
          RTS               ; retour avec la donnée dans AccB
*
OUTSCI    EQU *          ; Lecture sur le port série (SCI); seul Y est modifié
          XGDY              ;  AccD <--> Y
OUTSCI1   LDAA SCSR+REGBASE
          BPL OUTSCI1       ; MS bit <> TDRE flag
          XGDY              ; AccD <--> Y
          STAA SCDR+REGBASE ; Important : on réactualise  CCR
          RTS
*
INH1      EQU *
          CMPA #$7E         ; s'agit-il d'une lecture des registres ?
          BNE INH2
*
INH1A     TSX               ; X <- SP     : on place SP dans X
          XGDX              ; AccD <--> X : puis dans AccD
          BSR OUTSCI        ; on envoie SP (MSB)
          TBA
          BSR OUTSCI        ; on envoie SP (LSB)
          TSX                X <- SP  : Récupère X (=stack pointer)
          LDAB #9           ; envoi dans la pile de 9 Octets
          BRA TREADMEM      ; i.e. CCR,ACCB,ACCA,IXH,IXL,IYH,IYL,PCH,PCL
*
INH2      EQU *
          CMPA #$3E         ; s'agit-il d'une lecture des registres ?
          BNE SWISRV1
*
          BSR INSCI         ; réception de SP (MSB) à partir de l'IBM-PC
          TBA
          BSR INSCI         ; réception de SP (LSB) à partir de l'IBM-PC
          XGDX              ; AccD <--> X
          TXS               ;  SP <-- X
          LDAA #9           ; réception dans la pile de 9 Octets
          BRA TWRITMEM
*
SWISRV    EQU *            ; traitement des interruptions logicielles : SWI
          LDAA #BRKCODE     ; on envoie un signal de break à l'IBM-PC
          BSR OUTSCI
SWIIDLE   CLI
          BRA SWIIDLE       ; on attend la réponse de l'IBM-PC (Ibit=0,Xbit=1)
*
SWISRV1   EQU *
```

```
            CMPA #BRKACK        ; acquitement du break par l'IBM-PC ?
            BNE RXSRVEX
            TSX                 X <- SP
            LDAB #9
            ABX                 X <-- AccB + X
            TXS                 SP <-- X
            LDD 7,X             ; on pointe sur le 7ème octet (PCH) de l'état des registres
            BSR OUTSCI          ; adresse de retour du point d'arrêt (MSB)
            TBA
            BSR OUTSCI          ; adresse de retour du point d'arrêt (LSB)

            LDD #SWIIDLE        ; on force une boucle "bidon" pour le retour du break
            STD 7,X             ; on pointe sur le 7ème octet (PCH) de l'état des registres
            BRA INH1A           ; renvoi de l'état des registres à l'IBM-PC
*
            END
```

4.4 Annexe 4 : Programmes exemples

4.4.1 LCD_Time.asc : utilisation d'un 68hc811E2 en mode normal

```
*                        LCD_Time
* programme en eeprom   pour 68HC811E2
* Affichage de l'heure sur matrice LCD
* début de programme : $f800
*
*           J. Weiss juillet 95
*
* Les données sont transférées par le port série synchrone (SPI) :
* les transferts sont organisés sous forme de 2 envois vers la
* carte Clavier-afficheur et d'une lecture.
* Le programme utilise les interruption temps réel (RTI) pour l'horloge.
*
ddrd            equ 9          ; Data Direction Reg Port D
portd           equ 8          ; Port D
spcr            equ $28        ; SPI Control Register
spsr            equ $29        ; SPI Status Register
spdr            equ $2A        ; SPI Data Register
tmsk2           equ $24        ; Timer Mask Register 2
tflg2           equ $25        ; Timer Flag Register 2

*************************************************************
*              Constantes du programme                      *
*************************************************************

base_reg        equ $1000

** mots de commande (premier mot envoyé sur SPI)
RIEN            equ $FC  ; aucun affichage
LCD_on          equ $FF  ; affichage LCD
LCD_config      equ $FD  ; configuration de l'afficheur LCD
LED_on          equ $F6  ; sélection des LED
unites          equ $DE  ; sélection de l'afficheur des unités
dizaines        equ $EE  ; sélection de l'afficheur des dizaines
buzzer          equ $BE  ; sélection du buzzer
relais          equ $7E  ; sélection du relais
switches        equ $10  ; interrupteurs

** mots de donnée (deuxième mot envoyé sur SPI)
LED_k1          equ $1   ; cathode de la LED 1
LED_k2          equ $8   ; cathode de la LED 2
LED_k3          equ $40  ; cathode de la LED 3

*************************************************************
*               variables du programme                      *
*************************************************************
                ORG $0000 ; on place les variables en bas de la RAM interne
c244            RMB 1    ; compte 244 interuptions RTI avant d'obtenir 1s.
bis             RMB 1    ; indique à ff que l'on a ajouté le reste
seconde         RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
secondes        RMB 1    ; secondes de 0 à  59
minutes         RMB 1    ; minutes de 0 à 59
heures          RMB 1    ; heures de 0 à 23
data_in         RMB 1    ; lecture sur SPI
Commande_LCD    RMB 1    ; mot de commande du LCD (0: commande; 2 : affichage)
digit1          RMB 1    ; digit des unités de l'afficheur LED
digit10         RMB 1    ; digit des dizaines de l'afficheur LED
touche          RMB 1    ; touche frappée sur le clavier (4 bits ou $FF si rien)
inter           RMB 1    ; interrupteurs (0 : S1, 1 : S2 et $FF : rien)
message         RMB 10   ; message à afficher

*************************************************************
*            initialisation des vecteurs                    *
*************************************************************
                ORG  $FFFE
                FDB  Rti            ; it périodique
                ORG  $FFFE
                FDB  Debut          ; début du programme
                ORG  $F800

*************************************************************
*                  Début du programme                       *
*************************************************************
Debut           SEI                 ; set interrupt mask
                LDX  #base_reg      ; X <--- $1000
                LDS  #$FF           ; initialisation de la pile à FF

Depart          BSET pactl,x,#0       ; rti rate : 4,10 ms
```

```
                BSET tmsk2,x,#$40     ; rti Interrupt Enable
                CLR  secondes
                CLR  minutes
                CLR  heures
                CLR  c244

*************************************************************
*                 initialisation SPI                        *
*************************************************************
Init_SPI        BSET ddrd,x,#$38    ; SS, SCK, MOSI en sortie
                LDAA #$51           ; Enable SPI avec SCK=125kHz, wired_or mode
                STAA spcr,x
                LDAA #$3F           ; on met les bits du port D à 1
                STAA portd,x

*************************************************************
*                 initialisation LCD                        *
*************************************************************
*Commande_LCD    RMB 1  ; mot de commande du LCD (0: commande; 2 : affichage)
*LCD_config      equ $FD  ; configuration de l'afficheur LCD

Init_LCD        LDAA #LCD_config     ; mode config LCD
                STAA Commande_LCD
                LDAA #$1             ; clear Display
                JSR  Aff_LCD

** l'afficheur a besoin d'une tempo
Tempo           LDY  #$1000          ; IY <- durée
T1              DEY                  ; IY <- IY-1
                BNE  T1

                LDAA #$E             ; display =on
                JSR  Aff_LCD
                LDAA #$3C            ; N =2 lignes; F =  1
                JSR  Aff_LCD

*************************************************************
*   Départ du programme  *
*************************************************************
                LDY  #introduction
                JSR  Aff_msg
                JSR  Affhm
B1              SEI
                TST  c244
                BNE  Go              ; on n'affiche que toutes les secondes
                JSR  Affhm

Go              JSR Lect_clavier
                JSR Aff_7seg
                CLI                  ; Clear interrupt Mask
                WAI                  ; Attente d'une interruption
                JMP  B1

introduction    FCC ' Bonjour, il est   \'

*****************************************************************
*****************************************************************
* SP * envoi d'une chaine de caractères pointée par y finie par \ *
*****************************************************************
Aff_msg         LDAA #LCD_config       ; mode config LCD
                STAA Commande_LCD
                LDAA #$80              ; DD Ram =$00
                JSR  Aff_LCD
                LDAA #LCD_on           ; mode Affichage LCD
                STAA Commande_LCD
Envmot          LDAA 0,y
                CMPA #'\'
                BEQ  Envmot_fin        ; si \ alors retour
                JSR  Aff_LCD           ; si non envoi du caractère (0,y)
                INY                    ; IY <- IY + 1
                BRA  Envmot            ; on boucle
Envmot_fin      RTS

*****************************************************************
* SP *          préparation de l'affichage LCD                  *
*****************************************************************
*Commande_LCD    RMB 1  ; mot de commande du LCD (0: commande; 2 : affichage)
Aff_LCD         PSHB
                LDAB Commande_LCD
                JSR  OUTSPI          ; on envoie la commande (E=1)
                ANDB #$FE            ; Disable
                JSR  OUTSPI          ; on renvoie la commande (E=0)
```

```
                PULB
                RTS

*****************************************************************
* SP *  Envoi de 2 octets par SPI et réception d'un octet       *
*       - Le mot de commande est dans l'accumulateur B          *
*       - Les données en émission  sont dans l'accumulateur A   *
*       - la donnée en réception est dans data_in               *
*****************************************************************
*data_in        RMB 1    ; lecture sur SPI

OUTSPI          PSHA              ; on empile la donnée
                PSHB              ; on empile le mot de commande
                PSHY
                BCLR portd,x,#$20 ; met SS à 0 latche entrée et sortie registres
                STAB spdr,x       ; envoi du mot de commande

                LDY #08         ; 8 bits à faire tourner
RBBMIR          ROLA            ; les bits sortent à gauche par la retenue
                RORB            ; et rentrent à droite par la carry
                DEY
                BNE RBBMIR
ATE20           BRCLR spsr,x,#$80 ATE20  ; attend fin transmission prec.
                LDAA  spdr,x             ; on récupère la donnée sur le SPI
                STAA  data_in            ; enregistre
                STAB  spdr,x             ; on envoie la donnée sur le SPI
ATEF2O          BRCLR spsr,x,#$80 ATEF2O ; attend fin transmission pour reset SS
                BSET  portd,x,#$20       ; met SS à 1 (delatche sorties registres)
                PULY
                PULB                     ; on restaure le mot de commande dans B
                PULA                     ; on restaure la donnée dans A
                RTS

*************************************************************************
* SP *        Affichage de l'heure sur l'afficheur LCD                   *
*************************************************************************
*seconde          RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
*secondes         RMB 1    ; secondes de 0 à  59
*minutes          RMB 1    ; minutes de 0 à 59
*heures           RMB 1    ; heures de 0 à 23

Affhm           LDAA  #LCD_config        ; mode config LCD
                STAA  Commande_LCD
                LDAA  #$C7               ; DD Ram =$47
                JSR   Aff_LCD
                LDAA  #LCD_on            ; RS  = 1
                STAA  Commande_LCD
                LDAA  heures
                JSR   Aff_ascii
                LDAA  #':'
                JSR   Aff_LCD
                LDAA  minutes
                JSR   Aff_ascii
                LDAA  #':'
                JSR   Aff_LCD
                LDAA  secondes
                JSR   Aff_ascii
                RTS

** envoi en DECIMAL ASCII du nombre contenu dans A

Aff_ascii       PSHB                  ; sauvegarde de B
                JSR  Div_10           ; division par 10 de A
                STAA digit10
                STAB digit1
                ADDA #$30             ; A <- A + $30
                JSR  Aff_LCD          ; envoi du digit Quotient (dizaines)
                TBA                   ; A <- B
                ADDA #$30             ; B <- B + $30
                JSR  Aff_LCD          ; envoi du digit Reste (unités)
                PULB pulb             ; restauration de B
                RTS

** DIVISION PAR 10  ----------------------------
** donnee dans A -> A/10 ds A et reste ds B

Div_10          TAB                ; B <-- A
                CLRA               ; A <- 0  : variable dans AccD
                LDX  #10           ; ix <- 10
                IDIV               ; Q:IX ; R:AccD <- AccD/IX
                PSHB               ; push B : Pile <- MSB du reste
                XGDX               ; Accd <-> IX
```

```
                TBA                 ; A <-- B
                PULB                ; pull B : B <- MSB du reste
                LDX  #base_reg      ; réinitialisation de IX ($1000)
                RTS                 ; return

*************************************************************************
* SP *        Affichage sur les afficheurs 7 segments LED                *
*                  digit1 correspond aux unités                         *
*                 digit10 correspond aux dizaines                       *
*************************************************************************
*digit1           RMB 1    ; digit des unités de l'afficheur LED
*digit10          RMB 1    ; digit des dizaines de l'afficheur LED

Aff_7seg        PSHY
                PSHB
                PSHA
                LDY  #Tab_7seg    ; on pointe sur la table de conversion
                LDAA c244         ; chargement du compteur d'horloge c244
                ASRA              ; on propage le LSB dans C
                BCC  Aff2         ; c244 est pair (C=0), on affiche les unités
                LDAB #dizaines    ; adresse du digit à afficher
                PSHB
                LDAB digit10      ; chargement du digit à afficher
                BRA  Fin_7seg

Aff2            LDAB #unites      ; adresse du digit à afficher
                PSHB
                LDAB digit1       ; chargement du digit à afficher

Fin_7seg        ABY               ;  Y <-- Y + B ( B :offset dans la table)
                LDAA 0,y          ; pointage sur la table
                PULB              ; on récupère l'adresse du digit
                JSR  OUTSPI       ; envoi du digit
                PULA
                PULB
                PULY
                RTS

*     segment :       dec.bafg
Tab_7seg        FCB #%11101110          ; chiffre 0
                FCB #%00101000          ; chiffre 1
                FCB #%11001101          ; chiffre 2
                FCB #%10101101          ; chiffre 3
                FCB #%00101011          ; chiffre 4
                FCB #%10100111          ; chiffre 5
                FCB #%11100111          ; chiffre 6
                FCB #%00101100          ; chiffre 7
                FCB #%11101111          ; chiffre 8
                FCB #%10101111          ; chiffre 9

*************************************************************************
* SP *            Lecture du clavier et des interrupteurs               *
*            touche : lecture du clavier (4 bits ou $FF si rien)        *
*        inter : état des interrupteurs (0 : S1, 1 : S2 et $FF : rien)  *
*************************************************************************
*touche           RMB 1    ; touche frappée sur le clavier (4 bits ou $FF si rien)
*inter            RMB 1    ; interrupteurs (0 : S1, 1 : S2 et $FF : rien)

Lect_clavier    PSHA
                PSHB
                PSHY
Init_lect       LDAA #$FF
                STAA inter
                STAA touche
                LDAA #1                 ; on pointe sur la 1ère rangée
                LDAB #RIEN              ; mot de commande nul
                BRA Lect_2
Lect_1          ASLA                    ; rangée suivante
                BEQ Fin_lect            ; fin du balayage des rangées
Lect_2          JSR OUTSPI              ; on stimule
                JSR OUTSPI              ; on réceptionne la donnée (4 bits LSB)
                BRSET data_in,#$0F,Lect_1 ; réponse vierge ?, si oui on boucle
                COM data_in             ; complémentation de la donnée
                CMPA #switches          ; adressage des interrupteurs ?
                BEQ Lect_int            ; il 'agit d'un interrupteur !
                JSR Lin_bin             ; conversion en binaire (résultat dans B)
                PSHB                    ; sauvegarde du résultat "rangée"
                LDAA data_in            ; réponse "colonne"
                JSR Lin_bin             ; conversion en binaire (résultat dans B)
                ASLB
                ASLB
                PULA                    ; récupération du résultat "rangée"
```

```
                ABA                     ; mise en forme de la réponse clavier
                TAB
                LDY #Tab_clavier
                ABY
                LDAA 0,Y
                STAA touche
Fin_lect        PULY
                PULB
                PULA
                RTS

Lect_int        LDAA data_in
                RORA
                STAA inter
                BRA Fin_lect

Lin_bin         LDAB #0             ; compteur à zéro
Lin_1           RORA                ; on décale
                BCS Fin_lin         ; fin si on a propagé un 1 dans C
                CMPB #3             ; fin des 4 bits ?
                BEQ Fin_lin         ; si oui alors fin
                INCB                ; incrémentation du compteur
                BRA Lin_1           ; on boucle
Fin_lin         RTS

* on lit : 147A2580369BFEDC
Tab_clavier     FCB #$1
                FCB #$4
                FCB #$7
                FCB #$A
                FCB #$2
                FCB #$5
                FCB #$8
                FCB #$0
                FCB #$3
                FCB #$6
                FCB #$9
                FCB #$B
                FCB #$F
                FCB #$E
                FCB #$D
                FCB #$C

*****************************************************************
* INT *          Gestion de l'interruption périodique           *
* RTI *                  calcul de l'heure                      *
*****************************************************************
*c244             RMB 1    ; compte 244 interuptions RTI avant d'obtenir 1s.
*bis              RMB 1    ; indique à ff que l'on a ajouté le reste
*seconde          RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
*secondes         RMB 1    ; secondes de 0 à  59
*minutes          RMB 1    ; minutes de 0 à 59
*heures           RMB 1    ; heures de 0 à 23

** toutes les 256 s, on charge c244 avec le reste (16) pour ajuster l'horloge

Rti              DEC c244
                 BEQ Rti_0
                 JMP Fin_rti
Rti_0            BSET  c244,#244        ; recharge le compteur
                 BRSET bis,#$ff,Rti_1   ; bis indique que l'on a déjà re-
                                        * chargé c244 avec le reste
                 INC  seconde           ; inc sauf si bis = $ff
                                        * chaque fois que seconde passe à 0
                 BNE  Rti_2             ; on charge bis à ff et c244 à 16 (reste)
                                        * on ne compte pas cette "seconde"
Rti_01           LDAA #16               ; on charge le reste
                 STAA c244
                 BSET bis,#$ff
                 JMP  Fin_rti

Rti_1            CLR  bis
Rti_2            JSR  Affhm
                 LDAA #60
                 INC  secondes           ; secondes de 0 à 59
                 CMPA secondes
                 BNE  Fin_rti
                 CLR  secondes
                 INC  minutes
                 LDAA #60
                 CMPA minutes
                 BNE  Fin_rti            ; minutes
```

```
                CLR  minutes
                INC  heures
                LDAA heures
                CMPA #24
                BNE  Fin_rti            ; heures
                CLR  heures

Fin_rti         BCLR tflg2,x,#$BF      ; annule flag RTI
                RTI

               END
```

4.4.2 LCD_A1.asc : utilisation d'un 68hc11A1 en mode bootstrap ou normal

```
*                         LCD_A1
* programme en eeprom   pour 68HC11A1
* Affichage de l'heure sur matrice LCD
* début de programme : $B600
*           J. Weiss juillet 95
*
* Les données sont transférées par le port série synchrone (SPI) :
* les transferts sont organisés sous forme de 2 envois vers la
* carte Clavier-afficheur et d'une lecture.
* Le programme utilise les interruption temps réel (RTI) pour l'horloge.
*

ddrd            equ 9          ; Data Direction Reg Port D
portd           equ 8          ; Port D
spcr            equ $28        ; SPI Control Register
spsr            equ $29        ; SPI Status Register
spdr            equ $2A        ; SPI Data Register
tmsk2           equ $24        ; Timer Mask Register 2
tflg2           equ $25        ; Timer Flag Register 2

************************************************************
*              Constantes du programme                     *
************************************************************

base_reg        equ $1000
JUMPEXT         equ $7E             ; mnémonique de l'instruction JMP
JRTI            equ $00EB           ; adresse du vecteur RTI

** mots de commande (premier mot envoyé sur SPI)
RIEN            equ $FC  ; aucun affichage
LCD_on          equ $FF  ; affichage LCD
LCD_config      equ $FD  ; configuration de l'afficheur LCD
LED_on          equ $F6  ; sélection des LED
unites          equ $DE  ; sélection de l'afficheur des unités
dizaines        equ $EE  ; sélection de l'afficheur des dizaines
buzzer          equ $BE  ; sélection du buzzer
relais          equ $7E  ; sélection du relais
switches        equ $10  ; interrupteurs

** mots de donnée (deuxième mot envoyé sur SPI)
LED_k1          equ $1   ; cathode de la LED 1
LED_k2          equ $8   ; cathode de la LED 2
LED_k3          equ $40  ; cathode de la LED 3

************************************************************
*              variables du programme                      *
************************************************************
                ORG $00B0 ; on place les variables en $B0
c244            RMB 1    ; compte 244 interuptions RTI avant d'obtenir 1s.
bis             RMB 1    ; indique à ff que l'on a ajouté le reste
seconde         RMB 1    ; nbre de secondes de 0 à 256 pour ajustement
secondes        RMB 1    ; secondes de 0 à  59
minutes         RMB 1    ; minutes de 0 à 59
heures          RMB 1    ; heures de 0 à 23
data_in         RMB 1    ; lecture sur SPI
Commande_LCD    RMB 1    ; mot de commande du LCD (0: commande; 2 : affichage)
digit1          RMB 1    ; digit des unités de l'afficheur LED
digit10         RMB 1    ; digit des dizaines de l'afficheur LED
touche          RMB 1    ; touche frappée sur le clavier (4 bits ou $FF si rien)
inter           RMB 1    ; interrupteurs (0 : S1, 1 : S2 et $FF : rien)
message         RMB 10   ; message à afficher

************************************************************
*             initialisation des vecteurs                  *
************************************************************
```

```
                ORG  $B600          ; adresse de l'EEPROM
Debut           SEI                ; set interrupt mask
                LDAA #JUMPEXT      ; mnémonique de l'instruction JMP
                LDX  #Rti          ; adresse du sous-programme d'IT (ici : RTI)
                STAA JRTI          ; ecriture de JMP Rti à partir de l'adresse $EB
                STX  JRTI+1        ;
                LDX  #base_reg     ; X <--- $1000
                LDS  #$EA          ; initialisation de la pile à $EA

Depart          BSET pactl,x,#0      ; rti rate : 4,10 ms
                BSET tmsk2,x,#$40    ; rti Interrupt Enable
```

...

La suite du programme est identique au programme précédent (LCD_time.asc) à partir de l'étiquette `Depart`.

4.5 Annexe 5 : Documentation sur l'afficheur LCD

Afficheur LCD Sanyo : 2 lignes de 24 Caractères

Commande	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear Display	0	0	0	0	0	0	0	0	0	1	effaçage de l'écran et retour du curseur à l'origine
Return Home	0	0	0	0	0	0	0	0	1	*	retour du curseur et du texte à l'origine
Shift Mode	0	0	0	0	0	0	0	1	I/D	S	sens de déplacement du curseur (et du texte); opération réalisée pendant un accès Données (R ou W)
Display ON/OFF	0	0	0	0		0	1	D	C	B	visualisation du texte et du curseur
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	déplacement du curseur et du texte
Function Set	0	0	0	0	1	DL	N	F	*	*	définition de la longueur des données, du nombre de lignes et de la police de caractères
Set CG RAM Address	0	0	0	1	ACG						définition de l'adresse du générateur de caractères; les données sont envoyées et reçues après cette commande
Set DD RAM address	0	0	1	ADD							définition de l'adresse de l'affichage; les données sont envoyées et reçues après cette commande
Read Busy Flag & Address	0	1	BF	AC							lecture du registre d'état et du compteur d'adresse
Write Data to CG or DD RAM	1	0	Write Data								écriture de données en RAM (CG ou DD)
Read Data From CG or DD RAM	1	1	Read Data								lecture de données en RAM (CG ou DD)

I/D = 1 : Increment		DD RAM : Display Data RAM
I/D = 0 : Decrement		CG RAM : Character Generator RAM
S = 1 : Accompanies display shift		ACG : CG RAM Address
S/C = 1 : Display shift		ADD : DD RAM Address
S/C = 0 : Cursor move		AC : Address Counter
R/L = 1 : Shift to the right		
R/L = 0 : Shift to the left		
DL = 1 : 8 bits	DL = 0 : 4 bits	
N = 1 : 2 lines	**N = 0 : 1 line**	
F = 1 : 5*10 Dots	**F = 0 : 5*7 Dots**	
BF = 1 : Internally operation		
BF = 0 : Can accept instruction		
D = 0 : Display OFF	D = 1 : Display ON	
C = 0 : Cursor OFF	C = 1 : Cursor ON	
B = 0 : Blink OFF	B = 1 : Blink ON	J. Weiss, Juin 94