Auteurs :

EL QABLI Rim ***340997***
FROTTIER Zoé ***341115***

Le 30 mai 2023

READTEA

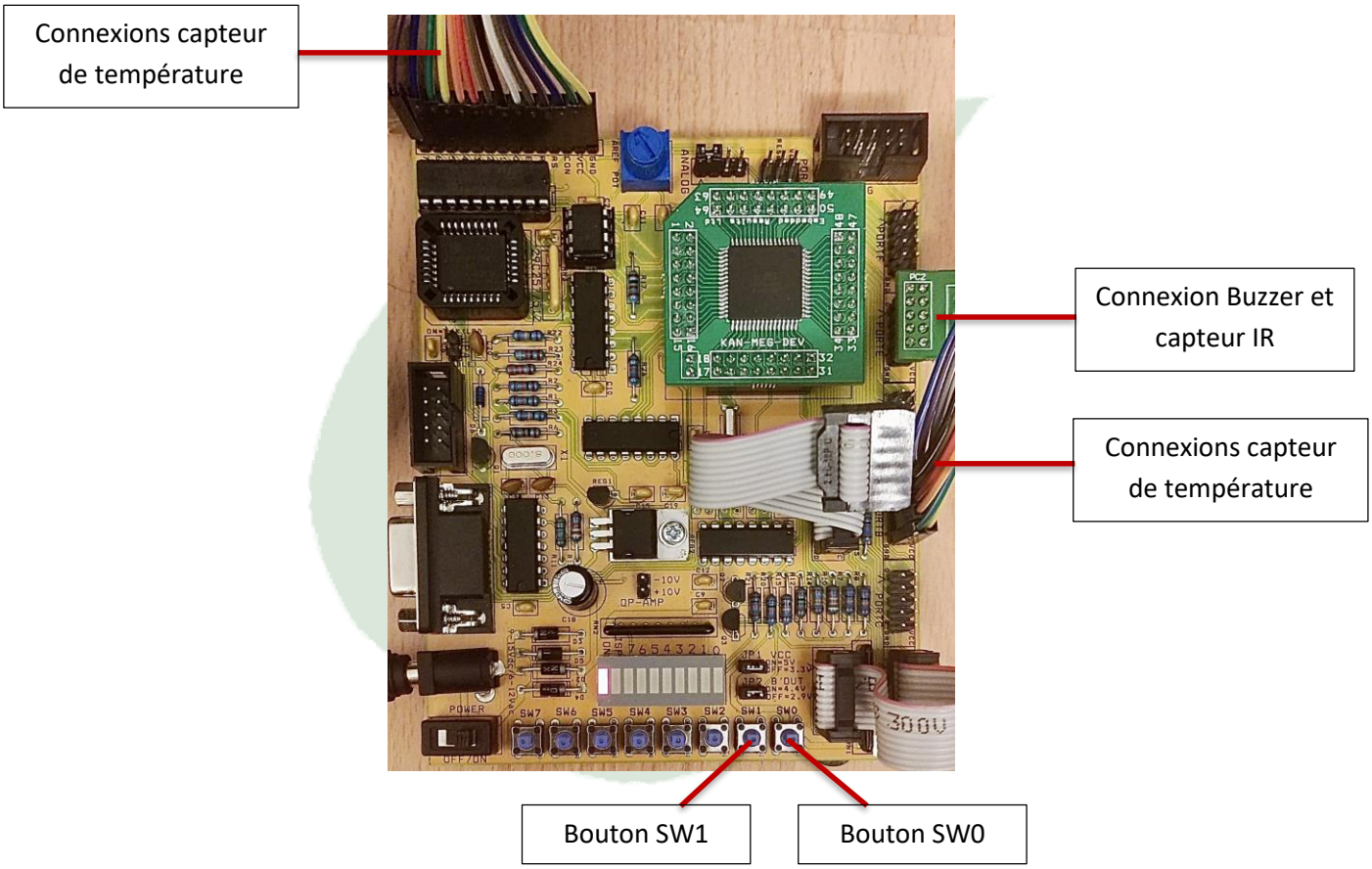
1. Introduction

Ce rapport a pour but de discuter notre projet dans le cadre du cours de Microcontrôleurs pour la section de Microtechnique.
Notre application s'agit d'une machine à boisson, qui permet à l'utilisateur de fixer la température à laquelle il souhaite déguster son thé (ou toute autre boisson chaude). En effet, dans le cas d'une température trop chaude, l'utilisateur définit une température plus faible, et lorsque l'air ambiant aura refroidit le thé à la température voulue, notre mécanisme le fera savoir par une mélodie, ce qui permet à l'utilisateur de vaquer à ses occupations tout en évitant un potentiel oubli.

2. Mode d'emploi

1) Après avoir allumé le mécanisme via le switch, entrer la température cible voulue via la télécommande IR Remote Control Vivanco UR Z2. A ce moment-là, la température du thé s'affiche temps réel en dessus de la température cible sur le périphérique d'affichage LCD 2x16 (Hitachi44780U 2x16 LCD). Afin de modifier la valeur de la température cible, presser le 1er bouton (SW0) et suivre la même démarche que précédemment.

2) Lorsque le thé aura été suffisamment refroidi par la température ambiante et donc aura atteint la température cible, une mélodie sera déclenchée afin de prévenir l'utilisateur. Celui-ci peut alors communiquer au mécanisme qu'il a bien saisi que le thé est à la température désirée, et peut alors interrompre la mélodie en pressant le 2nd bouton (SW1). La mélodie s'arrête alors et le mécanisme revient à son état initial ou l'utilisateur peut choisir à nouveau une température cible pour son thé selon le mode d'emploi précité.



3. Description technique de l'application et du matériel

Afin de mettre au point notre application, nous avons choisi d'utiliser les périphériques suivants:

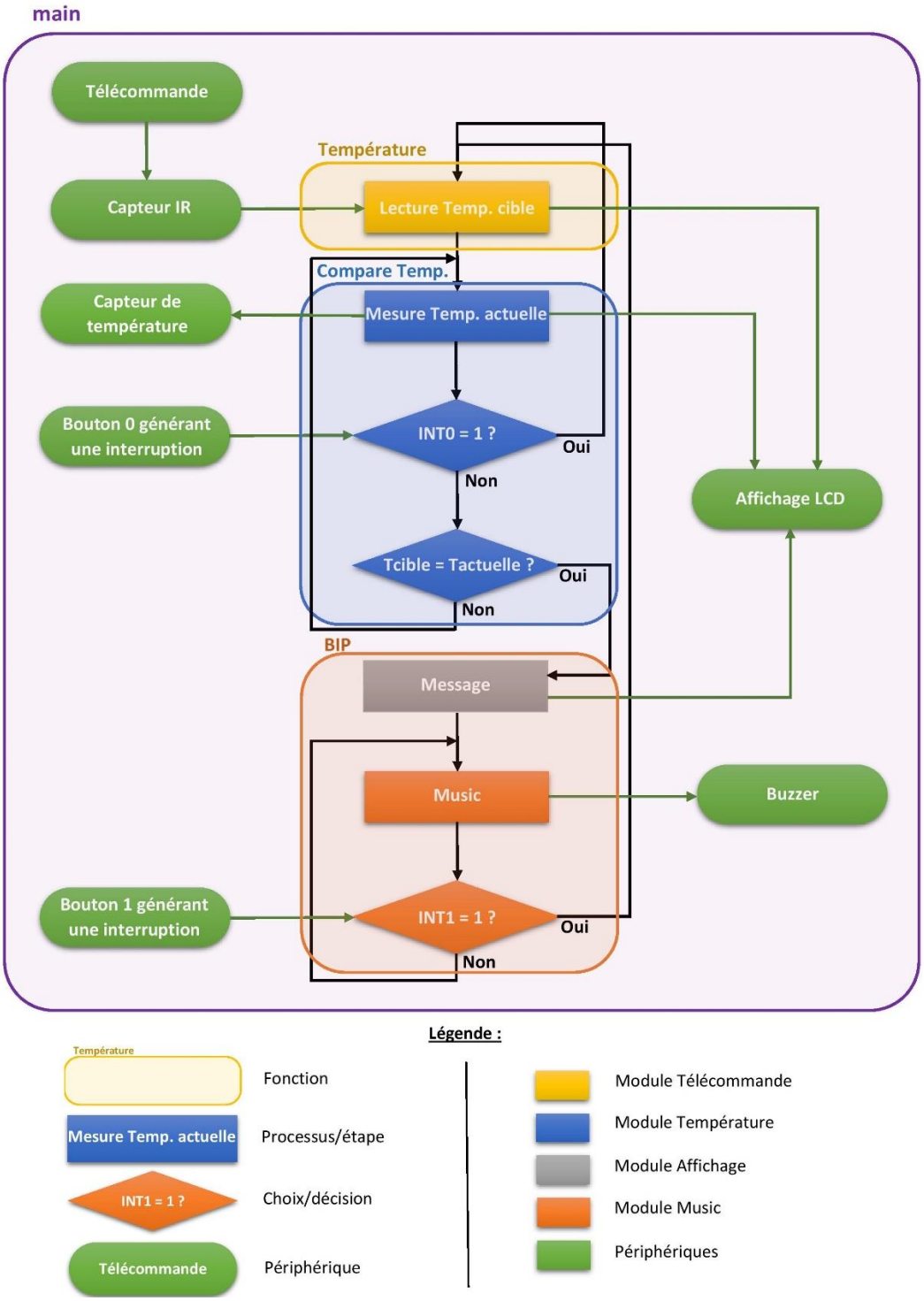
Périphérique	Port
Télécommande IR-RC5 Remote Control Vivanco UR Z2 (RC5) et capteur infra-rouge	Port E
Périphérique d'affichage LCD Hitachi44780U 2x16	Port LCD
Capteur de température (1-wire)	Port B
Un buzzer piézo-électrique	Port E
Boutons poussoir (SW0 et SW1)	Port D sur INT0 et INT1

Les différentes interfaces d'acquisition qui constituent notre mécanisme sont la télécommande, le capteur de température, ainsi que les deux boutons-poussoirs, tandis que les interfaces d'affichage sont le périphérique d'affichage LCD et le buzzer piezo-électrique.

Afin de permettre à l'utilisateur d'interagir avec notre mécanisme, nous avons mis en place deux interruptions qui fonctionnent respectivement via les boutons poussoirs (cf. image précédente). Ces deux boutons servent à gérer respectivement les fonctionnalités suivantes : permettre à l'utilisateur de changer la température cible et de communiquer au mécanisme qu'il a bien saisi que le thé est à la température désirée, interrompant ainsi la mélodie jouée initialement.

4. Fonctionnement du programme

Architecture et fonctionnement algorithmique de notre mécanisme



➢ Interruptions :

Nous utilisons les interruptions INT0 et INT1 sur le port D qui sont reliés respectivement aux boutons SW0 et SW1.

- INT0 : Lorsque le bouton SW0 est appuyé, le registre a3 est mis à 1. Cette interruption permet de revenir au programme principale (main) lorsqu'une température cible à été choisi et que l'on souhaite la modifier.
- INT1 : Lorsque le bouton SW1 est appuyé, le registre b3 est mis à 1. Cette interruption permet d'arrêter la musique et de revenir au programme principal (main).

➢ Algorithmique de notre mecanisme :

Lorsque l'on allume le système, le reset s'exécute et initialise le pointeur de la pile et les périphériques (LCD, capteur de température (1-wire)). On met la pin speaker du port E en sortie et on active les interruptions. Puis le main s'exécute, la fonction affiche le message principal "Readtea" ainsi que la demande de la température cible. Ensuite on appelle le module télécommande qui va permettre d'enregistrer la valeur saisie par l'utilisateur. La valeur de la température cible est lue depuis le port E. On appelle une fonction d'affichage pour afficher la température cible. Ensuite la fonction loopCompareTemp est appelée. Elle va appeler la température qui va lire la température sur le port B et appeler une fonction d'affichage. A la fin de cette fonction on regarde si le registre a3 est passé à 1 ce qui signifie qu'il y a eu une interruption sur INT0. Si a3=1, le programme retourne au main sinon il continue. Puis dans la fonction loopCompareTemp on compare la température cible à la température actuelle. Tant qu'elles ne sont pas égales, la fonction loopCompareTemp va se répéter. Lorsqu'elles sont égales, le programme saute à la fonction bip qui appelle la fonction d'affichage qui affiche le message "c'est prêt" et appelle la musique qui change le bit speaker du port E pour faire un son. Dans la fonction play de music, on regarde si le registre b3 est passé à 1 ce qui correspond à l'interruption de INT1 et qui permet d'arrêter la musique et de revenir au programme principal. S'il n'y a pas d'interruption, la musique continue de jouer en boucle.

➢ Mémoire :

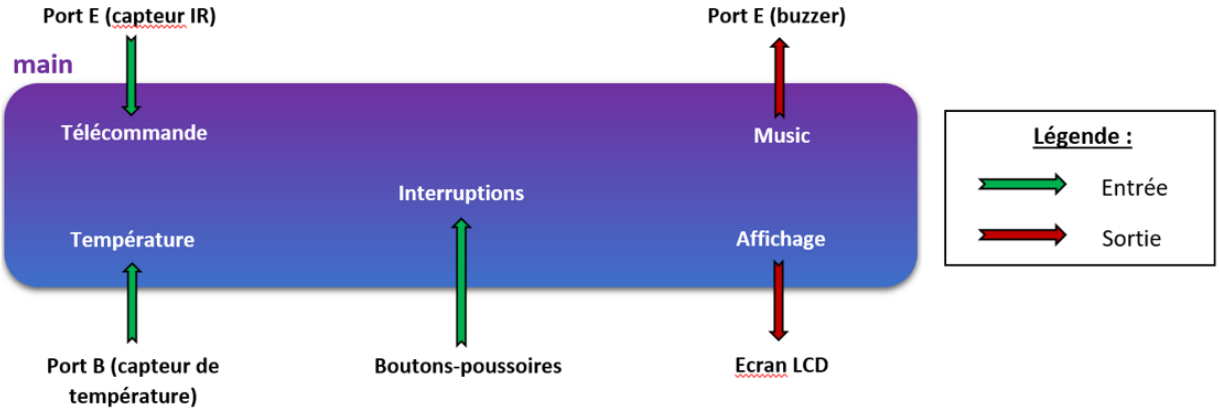
Nous utilisons les registres de l'AVR pour stocker les variables et la mémoire programme pour stocker les chaînes de caractères et la musique.

5. Présentation des modules du code

Le programme est décomposé en plusieurs modules principaux ainsi que de librairies. Les librairies utilisées sont ldc.asm, sound.asm, wire1.asm, printf.asm, definition.asm, macro.asm. Les modules principaux regroupent les fonctions qui utilisent un périphérique et sont répartis comme suit :

Fichier .asm	Périphérique utilisé
Télécommande	Télécommande et capteur infra-rouge
Température	Capteur de température
Music	Buzzer
Affichage	Ecran LCD

Utilisation des ports par les différents modules et interruptions au sein de notre mécanisme



➢ Le module Télécommande comprend la sous-routine telecommande qui permet de lire, décoder et stocker la valeur de la température saisie par l'utilisateur.

➢ Le module Température comprend la sous-routine température et la boucle loopCompareTemp. La sous-routine température lit, décode et stocke la valeur de la température actuelle. Elle fait appel à une fonction d'affichage pour afficher la température en temps réel. La boucle loopCompareTemp appelle la routine température, compare la valeur de la température actuelle à la température cible. Lorsque la température cible et la température actuelle sont égal, il y a un branchement à la fonction bip qui se trouve dans le module main. Ce module inclus le fichier "wire1.asm" qui contient les sous-routines pour la communication 1-wire.

➢ La fonction bip appelle la routine affiche_msg_pret et la routine music.

➢ Le module Affichage contient les sous-routines liées à l'affichage. Elles sont :

- affiche_Tcible pour afficher la température cible
- affiche_msg_pret qui affiche le message "c'est prêt" lorsque la température cible et égale à la température actuelle
- affiche_msg_principal qui affiche le nom du système et la demande de la température cible.
- affiche_Tact qui affiche la température cible une fois qu'elle a été saisie.

Ce module contient les fichiers "lcd.asm" et "printf.asm" qui contiennent les sous-routines et les macros pour afficher des messages sur l'écran lcd.

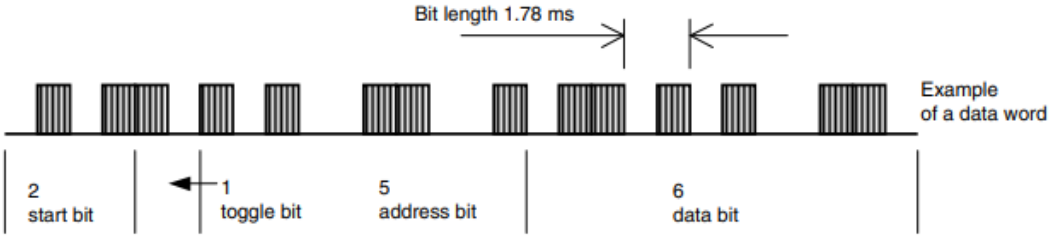
➢ Le module music comprend la look-up table avec les notes de la mélodie et la fonction music qui initialise le pointeur z et la fonction play qui permet de jouer les notes les unes après les autres.

➢ Le module main continent les fichiers "macros.asm" et "definitions.asm" ainsi que tous les autres modules : Telecommande.asm","Temperature.asm","Affichage.asm","music.asm. Il contient la table d'interruption, le reset, le programme main et la fonction bip.
La fonction main appelle différentes fonctions des différents modules : affiche_msg_principal, telecommande, afficheTcible, loopCompareTemp. La fonction bip appelle les fonctions affiche_msg_pret et music. La table d'interruption comprend les interruptions asynchrone INT0 et INT1.

6. Description de détail de l'accès au périphérique

➢ Télécommande :

Le périphérique de la télécommande utilise le protocole RC5 qui est asynchrone. Il utilise 14 bit encodé au format Manchester. La donnée qui correspond à la touche appuyée est codée sur les 6 derniers bits.



Pour notre projet, nous devons entrer une température à l'aide de la télécommande. Ainsi, appuyer sur deux touches pour former le nombre de la température.

Le code Télécommande.asm, le premier chiffre est lu et décodé puis stocké dans le registre a2. Il s'agit du chiffre des dizaines de la température. Ensuite le deuxième chiffre est lu et décodé et est stocké dans b0. Puis nous multiplions le chiffre des dizaines par 10 et ajoutons le chiffre des unités pour convertir les deux chiffres en un nombre et le stocker dans un unique registre c2.
Ainsi l'utilisateur entre la valeur de la température souhaitée à l'aide de la télécommande et cette valeur se retrouve stocké dans le registre c2.

➢ Capteur de Température :

Le capteur de température communique en 1-wire. Il utilise une seule ligne physique pour réaliser la communication entre le microcontrôleur et le périphérique. Nous avons utilisé la librairie wire1.asm.

Dans le code Temperature.asm, on lance la mesure en appelant les sous-routines de wire1.asm puis on stocke le résultat dans a1 et a0. La température est donnée en forme fractionnaire avec quatre bits pour la partie décimale et 12 pour la partie entière.

Pour notre application, la température va de 0 à 100 degrés. Nous avons donc décidé de ne garder que la partie entière et de stocker le résultat dans un seul registre. Nous avons fait deux masques pour récupérer les bits de poids faible du registre a1 et les bits de poids fort du registre a0. On a utilisé la fonction swap pour mettre les bits de poids faibles de a1 à la place des bits de poids fort et les bits de poids fort de a0 à la place des bits de poids faibles. Nous avons additionné les deux registres et stocker le résultat final dans le registre a0.

➢ Controlleur de LCD Hitachi 44780 :

Le périphérique LCD dispose de 14 lignes pour l'alimentation et le transfert de données. Le microcontrôleur et le Lcd communiquent le bus d'adresse, de données et de contrôle du système. Nous avons utilisé les librairies lcd.asm et printf.asm pour utiliser l'écran LCD.

Nous utilisons les routines lcd_clear, lcd_home, lcd_lf de lcd.asm pour effacer et positionner le curseur l'emplacement des messages à afficher.

Toutes les sous-routines relatives à l'affichage sont regroupée dans le fichier Affichage.asm.

Nous avons utilisé la macro PRINTF pour afficher de manière simple des chaînes de caractères ainsi que les valeurs des registres. Les valeurs des registres sont affichées en décimal en utilisant FDEC.

- **Buzzer :**

Le buzzer piézoélectrique est utilisé pour générer une mélodie.

Pour cela nous utilisons le fichier sound.asm qui comporte les notes. Pour faire la mélodie nous avons fait une look-up table avec les notes de la mélodie qui se suivent. La look-up table est stocké en mémoire programme. On initialise le pointeur z pour à l'adresse de la première note puis on incrémente le pointeur z pour avancer dans la look-up table.

La sous-routine sound utilise les bytes a0, a1, b0 et b1. Les registres a0, b0 et a1 sont utilisés pour le temps de la note et a0 contient la fréquence de la note voulue. La pin SPEAKER du port E est mise à 1 et 0 à la fréquence de la note.

7. Annexes : code source de l'application

Veuillez retrouver le code source de notre application ci-après.

```
/* Projet de Microcontrôleur Readtea
    main.asm
    340997  Rim El Qabli
    341115  Frottier Zoé
*/
.include "macros.asm"
.include "definitions.asm"

; === Table d'interruption  ===
.org    0
    jmp reset
    jmp ext_int0
    jmp ext_int1

ext_int0:                       ;   Interruption retour menu principal
    ldi a3,1
    rcall LCD_clear
    reti

ext_int1:                       ; Interruption stop music et retour menu
  principal
    ldi b3,1
    reti

reset:
    LDSP        RAMEND
    rcall       wire1_init              ; initialise l'interface 1-wire(R)
    rcall       LCD_init                ; initialise le lcd
    OUTI        EIMSK,0b00000011        ; active les interruptions sur les
      bouttons SW0 ET SW1
    sbi         DDRE,SPEAKER            ; met la pin SPEAKER en sortie
    sei                                 ; autorise les interruptions
    rjmp        main                    ; saute au main


.include "Telecommande.asm"
.include "Temperature.asm"
.include "music.asm"
.include "Affichage.asm"


main:
    ldi         a3,0                    ; initialise le registre pour
      l'interruption de retour au menu principal
    rcall       affiche_msg_principal
    rcall       telecommande            ;appelle la sous-routine qui recoit les
      chiffres de la télécommande , les décode et stocke le resultat dans c2
    rcall       afficheTcible           ;appelle la sous-routine qui affiche la
      temperature cible saisie avec la télécommande
    rcall       loopCompareTemp         ;appelle la sous-routine qui compare la
      temperature actuelle et la temperature cible
    rjmp        main
```

```
bip:            ;réponse du microcontroleur lorsque Tcible=Tact
    rcall       affiche_msg_pret
    rcall       music
```

```
/*
 * Telecomande.asm
  Module Télécommande : fonction telecommande qui permet de lire la temperature
    saisie depuis la télécommande et stocke le resultat.
  340997    Rim El Qabli
  341115    Frottier Zoé
 */

 .equ           T1 = 1760           ;periode

telecommande:                       ;Lecture et decodage chiffre des dizaines
        CLR2        b1,b0           ;efface les registre b0,b1
        ldi         b2,14           ;initialise un compteur de bits
        WP1         PINE,IR         ;Attend si PINE=1
        WAIT_US     (T1/4)

loop1:
        P2C     PINE,IR
        ROL2        b1,b0
        WAIT_US     (T1-4)
        DJNZ        b2,loop1
        mov a2,b0                   ; stocke le chiffre des dizaines dans a2

        WAIT_MS     500             ;Attend entre la saisie des deux chiffres

        CLR2        b1,b0           ;Lecture et décodage du chiffres des unitées
        ldi         b2,14
        WP1         PINE,IR
        WAIT_US     (T1/4)

loop:
        P2C         PINE,IR
        ROL2        b1,b0
        WAIT_US     (T1-4)
        DJNZ        b2,loop

                                    ;stocke le deuxieme chiffre dans b0
        mov         a1,b0           ;Convertit les deux chiffres en un nombre
        ldi         r16,9
        mov         r17,a2
mul_loop:
        add         r17,a2
        dec         r16
        brne        mul_loop
        add         a1,r17
        mov         c2,a1           ;stocke la temperature cible dans c2
    ret
```

```
/*
 * Temperature.asm
    Module Temperature : fonction temperature pour lire , decoder et stocker la
      temperature depuis le capteur de temperature
    340997  Rim El Qabli
    341115  Frottier Zoé

 */
 .include "wire1.asm"                   ; inclus les routines communication 1-
   wire

 temperature:
    ldi         b3,0                    ;intialise le registre pour le retour
      menu principal ( modifié avec l'interruption)
    rcall       wire1_reset
    CA  wire1_write, skipROM
    CA  wire1_write, convertT
    WAIT_MS     750


    rcall       wire1_reset             ;mesure la température
    CA          wire1_write, skipROM
    CA          wire1_write, readScratchpad
    rcall       wire1_read

    mov         c0,a0
    rcall       wire1_read
    mov         a1,a0
    mov         a0,c0                   ; temperature stocké dans a0 et a1

    andi        a0,0b11110000           ;Masquage pour récuperer la partie
      entière de la temperature et stocker le resultat dans a0
    swap        a0
    andi        a1,0b00001111
    swap        a1
    add         a0,a1

    rcall       affiche_Tact            ;affiche la temperature actuelle sur le
      LCD
    cpi         a3,1                    ;verifie si il y a eu une interruption
      pour retourner au menu principal ( boutton SW0)
    brne        PC+2
    rjmp        main                    ;retour au menu si interruption (a3=1)
ret



loopCompareTemp:
    rcall   temperature                 ;mesure la temperature
    cp c2, a0;                          ;compare la temperature cible (dans c2)
      avec la temperature actuelle (dans a0)
    breq bip2                           ;saute à bip2
    rjmp loopCompareTemp
```

```
bip2:   rjmp bip                        ;saute à bip qui affiche le message et
  saute à la musique
```

```
/*
 * Affichage.asm
   Module Affichage : fonctions qui affichent des messages sur l'écran lcd
     340997 Rim El Qabli
     341115 Frottier Zoé
 */

 .include "lcd.asm"                ; inclus les routines LCD
 .include "printf.asm"             ; inclus les routines d'affichage

 afficheTcible:
    rcall       LCD_clear          ;efface le contenu du lcd
    rcall       LCD_lf             ;met le curseur sur la deuxieme ligne
    PRINTF      LCD
.db "Temp cible=",FDEC,10,"C",0
ret

affiche_msg_pret:
    rcall       LCD_clear
    PRINTF      LCD
.db "**!c'est pret!**",0

    rcall       LCD_lf
    PRINTF      LCD
.db "****************",0
ret

affiche_msg_principal:
rcall       LCD_home
    PRINTF      LCD
.db "****READTEA****",0
rcall       LCD_lf                 ;met le curseur sur la deuxieme ligne
    PRINTF LCD
.db "Temp cible=",0
ret

affiche_Tact:                      ;affiche la temperature mesuré par le
  capteur de temperature
    rcall       LCD_home
    PRINTF  LCD
.db "Temp act=",FDEC,18,"C",0
ret

efface_lcd:
    rcall LCD_clear
    ret
```

```
/*
 * music.asm
 * Module "music" , look up table avec la mélodie et fonction music et play pour jouer les notes.
 *


 */



.include "sound.asm"                    ; inclus les routines pour la musique et les labels pour les notes

 ;Notes de la musique
musique:
    .db si2,1,si2,fam2,1,fam2,si2,1,si2,fam2,1,fam2,1,fam2,so2,1
    .db so2,1,so2,la2,so2,fam2,1,fam2,1,fam2,mi2,1,mi2,1,mi2,1
    .db mi2,re2,1,re2,1,re2,1,re2,dom2,1,dom2,re2,dom2,1,si,1
    .db 0 ;fin de la musique

music:
    ldi     zl, low(2*musique)          ;initialise le pointeur z au debut de la musique
    ldi     zh,high(2*musique)
play:
    cpi     b3,1                        ;regarde si le boutton SW1=1,           interrupetion pour arreter la musique
    breq    end1
    lpm                                 ;charge la note à jouer dans r0
    adiw    zl,1                        ;incremente le poointeur z
    tst     r0                          ;test si c'est la fin de la musique
    breq    end                         ;saute à end si fin de la musique
    mov     a0,r0                       ;met la note r0 dans a0
    ldi     b0,100
    rcall   sound                       ;appelle la sous routine sound de       sound.asm
    rjmp    play

end:
    rjmp music                          ;recommence la musique

end1:
    rcall efface_lcd                    ;efface le lcd
    jmp main                            ;retour au menu principal
```

```
; file  lcd.asm   target ATmega128L-4MHz-STK300
; purpose  LCD HD44780U library
; ATmega 128 and Atmel Studio 7.0 compliant

; === definitions ===
.equ    LCD_IR  = 0x8000    ; address LCD instruction reg
.equ    LCD_DR  = 0xc000    ; address LCD data register

; === subroutines ===
LCD_wr_ir:
; in    w (byte to write to LCD IR)
    lds u, LCD_IR       ; read IR to check busy flag  (bit7)
    JB1 u,7,LCD_wr_ir   ; Jump if Bit=1 (still busy)
    rcall   lcd_4us     ; delay to increment DRAM addr counter
    sts LCD_IR, w       ; store w in IR
    ret

lcd_4us:
    rcall   lcd_2us     ; recursive call
lcd_2us:
    nop                 ; rcall(3) + nop(1) + ret(4) = 8 cycles (2us)
    ret

LCD:
LCD_putc:
    JK  a0,CR,LCD_cr    ; Jump if a0=CR
    JK  a0,LF,LCD_lf    ; Jump if a0=LF
LCD_wr_dr:
; in    a0 (byte to write to LCD DR)
    lds w, LCD_IR       ; read IR to check busy flag  (bit7)
    JB1 w,7,LCD_wr_dr   ; Jump if Bit=1 (still busy)
    rcall   lcd_4us     ; delay to increment DRAM addr counter
    sts LCD_DR, a0      ; store a0 in DR
    ret

LCD_clear:      JW  LCD_wr_ir, 0b00000001       ; clear display
LCD_home:       JW  LCD_wr_ir, 0b00000010       ; return home
LCD_cursor_left:    JW  LCD_wr_ir, 0b00010000   ; move cursor to left
LCD_cursor_right:   JW  LCD_wr_ir, 0b00010100   ; move cursor to right
LCD_display_left:   JW  LCD_wr_ir, 0b00011000   ; shifts display to left
LCD_display_right:  JW  LCD_wr_ir, 0b00011100   ; shifts display to right
LCD_blink_on:       JW  LCD_wr_ir, 0b00001101   ; Display=1,Cursor=0,Blink=1
LCD_blink_off:      JW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0
LCD_cursor_on:      JW  LCD_wr_ir, 0b00001110   ; Display=1,Cursor=1,Blink=0
LCD_cursor_off:     JW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0

LCD_init:
    in  w,MCUCR                 ; enable access to ext. SRAM
    sbr w,(1<<SRE)+(1<<SRW10)
    out MCUCR,w
    CW  LCD_wr_ir, 0b00000001   ; clear display
    CW  LCD_wr_ir, 0b00000110   ; entry mode set (Inc=1, Shift=0)
```

```
    CW  LCD_wr_ir, 0b00001100   ; Display=1,Cursor=0,Blink=0
    CW  LCD_wr_ir, 0b00111000   ; 8bits=1, 2lines=1, 5x8dots=0
    ret

LCD_pos:
; in    a0 = position (0x00..0x0f first line, 0x40..0x4f second line)
    mov w,a0
    ori w,0b10000000
    rjmp    LCD_wr_ir

LCD_cr:
; moving the cursor to the beginning of the line (carriage return)
    lds w, LCD_IR           ; read IR to check busy flag  (bit7)
    JB1 w,7,LCD_cr          ; Jump if Bit=1 (still busy)
    andi    w,0b01000000    ; keep bit6 (begin of line 1/2)
    ori w,0b10000000        ; write address command
    rcall   lcd_4us         ; delay to increment DRAM addr counter
    sts LCD_IR,w            ; store in IR
    ret

LCD_lf:
; moving the cursor to the beginning of the line 2 (line feed)
    push    a0              ; safeguard a0
    ldi a0,$40              ; load position $40 (begin of line 2)
    rcall   LCD_pos         ; set cursor position
    pop a0                  ; restore a0
    ret
```

```
; file  printf.asm   target ATmega128L-4MHz-STK300
; purpose library, formatted output generation
; v2019.02 20180821 supports SRAM input from 0x0260
;                   through 0x02ff that should be reserved

; === description ===
;
; The program "printf" interprets and prints formatted strings.
; The special formatting characters regognized are:
;
; FDEC  decimal number
; FHEX  hexadecimal number
; FBIN  binary number
; FFRAC fixed fraction number
; FCHAR single ASCII character
; FSTR  zero-terminated ASCII string
;
; The special formatting characters are distinguished from normal
; ASCII characters by having their bit7 set to 1.
;
; Signification of bit fields:
;
; b     bytes       1..4 b bytes        2
; s     sign        0(unsigned), 1(signed)  1
; i     integer digits
; e     base        2,,36           5
; dp    dec. point  0..32           5
; $if   i=integer digits,  0=all digits,  1..15 digits
;       f=fraction digits, 0=no fraction, 1..15 digits
;
; Formatting characters must be followed by an SRAM address (0..ff)
; that determines the origin of variables that must be printed (if any)
; FBIN, sram
; FHEX, sram
; FDEC, sram
; FCHAR,sram
; FSTR, sram
;
; The address 'sram' is a 1-byte constant. It addresses
;    0..1f  registers r0..r31,
;   20..3f  i/o ports, (need to be addressed with an offset of $20)
;   0x0260..0x02ff  SRAM
; Variables can be located into register and I/0s, and can also
; be stored into data SRAM at locations 0x0200 through 0x02ff. Any
; sram address higher than 0x0060 is assumed to be at (0x0260+address)
; from automatic address detection in _printf_formatted: and subsequent
; assignment to xh; xl keeps its value. Consequently, variables that are
; to be stored into SRAM and further printed by fprint must reside at
; 0x0200 up to 0x02ff, and must be addressed using a label. Usage: see
; file string1.asm, for example.

; The FFRAC formatting character must be followed by
;   ONE sram address and
```

```
;   TWO more formatting characters
; FFRAC,sram,dp,$if

; dp    decimal point position, 0=right, 32=left
; $if   format i.f, i=integer digits, f=fraction digits

; The special formatting characters use the following coding
;
; FDEC  11bb'iiis   i=0 all digits, i=1-7 digits
; FBIN  101i'iiis   i=0 8 digits,   i=1-7 digits
; FHEX  1001'iiis   i=0 8 digits,   i=1-7 digits
; FFRAC 1000'1bbs
; FCHAR 1000'0100
; FSTR  1000'0101
; FREP  1000'0110
; FFUNC 1000'0111
;   1000'0010
;   1000'0011
; FESC  1000'0000

; examples
; formatting string         printing
; "a=",FDEC,a,0             1-byte variable a, unsigned decimal
; "a=",FDEC2,a,0            2-byte variable a (a1,a0), unsigend
; "a=",FDEC|FSIGN,a,0       1-byte variable 1, signed decimal
; "n=",FBIN,PIND+$20,0      i/o port, binary, notice offset of $20
; "f=",FFRAC4|FSIGN,a,16,$88,0  4-byte signed fixed-point fraction
;               dec.point at 16, 8 int.digits, 8 frac.digits
; "f=",FFRAC2,a,16,$18,0        2-byte unsigned fixed-point fraction
;               dec.point at 16, 1 int.digits, 8 frac.digits
; "a=",FDEC|FDIG5|FSIGN,a,0 1-byte variable, 5-digit, decimal, signed
; "a=",FDEC|FDIG5,a,0       1-byte variable, 5-digit, decimal, unsigned

; === registers modified ===
; e0,e1 used to transmit address of putc routine
; zh,zl used as pointer to prog-memory

; === constants ==============================================

.equ    FDEC    = 0b11000000    ; 1-byte variable
.equ    FDEC2   = 0b11010000    ; 2-byte variable
.equ    FDEC3   = 0b11100000    ; 3-byte variable
.equ    FDEC4   = 0b11110000    ; 4-byte variable

.equ    FBIN    = 0b10100000
.equ    FHEX    = 0b10010100    ; 1-byte variable
.equ    FHEX2   = 0b10011000    ; 2-byte variable
.equ    FHEX3   = 0b10011100    ; 3-byte variable
.equ    FHEX4   = 0b10010000    ; 4-byte variable

.equ    FFRAC   = 0b10001000    ; 1-byte variable
.equ    FFRAC2  = 0b10001010    ; 2-byte variable
.equ    FFRAC3  = 0b10001100    ; 3-byte variable
```

```
.equ    FFRAC4  = 0b10001110    ; 4-byte variable

.equ    FCHAR   = 0b10000100
.equ    FSTR    = 0b10000101

.equ    FSIGN   = 0b00000001

.equ    FDIG1   = 1<<1
.equ    FDIG2   = 2<<1
.equ    FDIG3   = 3<<1
.equ    FDIG4   = 4<<1
.equ    FDIG5   = 5<<1
.equ    FDIG6   = 6<<1
.equ    FDIG7   = 7<<1

; ===macro ===================================================

.macro  PRINTF          ; putc function (UART, LCD...)
    ldi w, low(@0)      ; address of "putc" in e1:d0
    mov e0,w
    ldi w,high(@0)
    mov e1,w
    rcall   _printf
    .endmacro

; mod   y,z


; === routines ================================================

_printf:
    POPZ            ; z points to begin of "string"
    MUL2Z           ; multiply Z by two, (word ptr -> byte ptr)
    PUSHX

_printf_read:
    lpm             ; places prog_mem(Z) into r0 (=c)
    adiw    zl,1    ; increment pointer Z
    tst r0          ; test for ZERO (=end of string)
    breq    _printf_end ; char=0 indicates end of ascii string
    brmi    _printf_formatted ; bit7=1 indicates formatting character
    mov w,r0
    rcall   _putw   ; display the character
    rjmp    _printf_read    ; read next character in the string

_printf_end:
    adiw    zl,1    ; point to the next character
    DIV2Z           ; divide by 2 (byte ptr -> word ptr)
    POPX
    ijmp            ; return to instruction after "string"

_printf_formatted:
```

```
; FDEC  11bb'iiis
; FBIN  101i'iiis
; FHEX  1001'iiis
; FFRAC 1000'1bbs
; FCHAR 1000'0100
; FSTR  1000'0101

    bst r0,0        ; store sign in T
    mov w,r0        ; store formatting character in w
    lpm
    mov xl,r0       ; load x-pointer with SRAM address
    cpi xl,0x60
    brlo rio_space
dataram_space:      ; variable originates from SRAM memory
    ldi xh,0x02     ;>addresses are limited to 0x0260 through 0x02ff
    rjmp space_detect_end   ;>that enables automatic detection of the origin
rio_space:          ; variable originates from reg or I/O space
    clr xh          ; clear high-byte, addresses are 0x0000 through 0x003f
      (0x005f)
space_detect_end:
    adiw    zl,1    ; increment pointer Z

;   JB1 w,6,_putdec
;   JB1 w,5,_putbin
;   JB1 w,4,_puthex
;   JB1 w,3,_putfrac
    JK  w,FCHAR,_putchar
    JK  w,FSTR ,_putstr
    rjmp    _putnum

    rjmp    _printf_read

; === putc (put character) ================================
; in    w   character to put
;   e1,e0   address of output routine (UART, LCD putc)
_putw:
    PUSH3   a0,zh,zl
    MOV3    a0,zh,zl, w,e1,e0
    icall           ; indirect call to "putc"
    POP3    a0,zh,zl
    ret

; === putchar (put character) =============================
; in    x   pointer to character to put
_putchar:
    ld  w,x
    rcall   _putw
    rjmp    _printf_read

; === putstr (put string) =================================
; in    x   pointer to ascii string
;   b3,b2   address of output routine (UART, LCD putc)
_putstr:
```

```
    ld  w,x+
    tst w
    brne    PC+2
    rjmp    _printf_read
    rcall   _putw
    rjmp    _putstr

; === putnum (dec/bin/hex/frac) ============================
; in    x   pointer to SRAM variable to print
;   r0  formatting character

_putnum:
    PUSH4   a3,a2,a1,a0 ; safeguard a
    PUSH4   b3,b2,b1,b0 ; safeguard b
    LDX4    a3,a2,a1,a0 ; load operand to print into a

; FDEC  11bb'iiis
; FBIN  101i'iiis
; FHEX  1001'iiis
; FRACT 1000'1bbs

    JB1 w,6,_putdec
    JB1 w,5,_putbin
    JB1 w,4,_puthex
    JB1 w,3,_putfrac

; FDEC  11bb'iiis
_putdec:
    ldi b0,10       ; b0 = base (10)

    mov b1,w
    lsr b1
    andi    b1,0b111
    swap    b1      ; b1 = format 0iii'0000 (integer digits)
    ldi b2,0        ; b2 = dec. point position = 0 (right)

    mov b3,w
    swap    b3
    andi    b3,0b11
    inc b3          ; b3 = number of bytes (1..4)
    rjmp    _getnum ; get number of digits (iii)

; FBIN  101i'iiis   addr
_putbin:
    ldi b0,2        ; b0 = base (2)
    ldi b3,4        ; b3 = number of bytes (4)
    rjmp    _getdig ; get number of digits (iii)

; FHEX  1001'iiis   addr
_puthex:
    ldi b0,16       ; b0 = base (16)
    ldi b3,4        ; b3 = number of bytes (4)
    rjmp    _getdig
```

```
_getdig:
    mov b1,w
    lsr b1
    andi    b1,0b111
    brne    PC+2
    ldi b1,8        ; if b1=0 then 8-digits
    swap    b1      ; b1 = format 0iii'0000 (integer digits)
    ldi b2, 0       ; b2 = dec. point position = 0 (right)
    rjmp    _getnum

; FFRAC 1000'1bbs   addr     00dd'dddd,     iiii'ffff

_putfrac:
    ldi b0,10       ; base=10
    lpm
    mov b2,r0       ; load dec.point position
    adiw    zl,1    ; increment char pointer
    lpm
    mov b1,r0       ; load ii.ff format
    adiw    zl,1    ; increment char pointer

    mov b3,w
    asr b3
    andi    b3,0b11
    inc b3          ; b3 = number of bytes (1..4)

    rjmp    _getnum

_getnum:
; in    a   4-byte variable
;   b3  number of bytes (1..4)
;   T   sign, 0=unsigned, 1=signed

    JK  b3,4,_printf_4b
    JK  b3,3,_printf_3b
    JK  b3,2,_printf_2b

_printf_1b:         ; sign extension
    clr a1
    brtc    PC+3    ; T=1 sign extension
    sbrc    a0,7
    ldi a1,0xff
_printf_2b:
    clr a2
    brtc    PC+3    ; T=1 sign extension
    sbrc    a1,7
    ldi a2,0xff
_printf_3b:
    clr a3
    brtc    PC+3    ; T=1 sign extension
    sbrc    a2,7
    ldi a3,0xff
```

```
_printf_4b:

    rcall   _ftoa       ; float to ascii
    POP4    b3,b2,b1,b0 ; restore b
    POP4    a3,a2,a1,a0 ; restore a

    rjmp    _printf_read

; ================================================
; func  ftoa
; converts a fixed-point fractional number to an ascii string
;
; in    a3-a0   variable to print
;   b0  base, 2 to 36, but usually decimal (10)
;   b1  number of digits to print ii.ff
;   b2  position of the decimal point (0=right, 32=left)
;   T   sign (T=0 unsiged, T=1 signed)

_ftoa:
    push    d0
    PUSH4   c3,c2,c1,c0 ; c = fraction part, a = integer part
    CLR4    c3,c2,c1,c0 ; clear fraction part

    brtc    _ftoa_plus  ; if T=0 then unsigned
    clt
    tst a3              ; if MSb(a)=1 then a=-a
    brpl    _ftoa_plus
    set                 ; T=1 (minus)
    tst b1
    breq    PC+2        ; if b1=0 the print ALL digits
    subi    b1,0x10     ; decrease int digits
    NEG4    a3,a2,a1,a0 ; negate a
_ftoa_plus:
    tst b2              ; b0=0 (only integer part)
    breq    _ftoa_int
_ftoa_shift:
    ASR4    a3,a2,a1,a0 ; a = integer part
    ROR4    c3,c2,c1,c0 ; c = fraction part
    DJNZ    b2,_ftoa_shift
_ftoa_int:
    push    b1          ; ii.ff (ii=int digits)
    swap    b1
    andi    b1,0x0f

    ldi w,'.'           ; push decimal point
    push    w
_ftoa_int1:
    rcall   _div41      ; int=int/10
    mov w,d0            ; d=reminder
    rcall   _hex2asc
    push    w           ; push rem(int/10)
    TST4    a3,a2,a1,a0 ; (int/10)=?
    breq    _ftoa_space ; (int/10)=0 then finished
```

```
    tst b1
    breq    _ftoa_int1  ; if b1=0 then print ALL int-digits
    DJNZ    b1,_ftoa_int1
    rjmp    _ftoa_sign
_ftoa_space:
    tst b1              ; if b1=0 then print ALL int-digits
    breq    _ftoa_sign
    dec b1
    breq    _ftoa_sign
    ldi w,' '           ; write spaces
    rcall   _putw
    rjmp    _ftoa_space
_ftoa_sign:
    brtc    PC+3        ; if T=1 then write 'minus'
    ldi w,'-'
    rcall   _putw
_ftoa_int3:
    pop w
    cpi w,'.'
    breq    PC+3
    rcall   _putw
    rjmp    _ftoa_int3

    pop b1              ; ii.ff (ff=frac digits)
    andi    b1,0x0f
    tst b1
    breq    _ftoa_end
_ftoa_point:
    rcall   _putw       ; write decimal point
    MOV4    a3,a2,a1,a0, c3,c2,c1,c0
_ftoa_frac:
    rcall   _mul41      ; d.frac=10*frac
    mov w,d0
    rcall   _hex2asc
    rcall   _putw
    DJNZ    b1,_ftoa_frac
_ftoa_end:
    POP4    c3,c2,c1,c0
    pop d0
    ret

; === hexadecimal to ascii ===
; in    w
_hex2asc:
    cpi w,10
    brsh    PC+3
    addi    w,'0'
    ret
    addi    w,('a'-10)
    ret

; === multiply 4byte*1byte ===
; funct mul41
```

```
; multiplies a3-a0 (4-byte) by b0 (1-byte)
;
; in    a3..a0  multiplicand (argument to multiply)
;   b0  multiplier
; out   a3..a0  result
;   d0  result MSB (byte 4)
;
_mul41: clr d0          ; clear byte4 of result
    ldi w,32            ; load bit counter
__m41:  clc             ; clear carry
    sbrc    a0,0        ; skip addition if LSB=0
    add d0,b0           ; add b to MSB of a
    ROR5    d0,a3,a2,a1,a0  ; shift-right c, LSB (of b) into carry
    DJNZ    w,__m41     ; Decrement and Jump if bit-count Not Zero
    ret

; === divide 4byte/1byte ===
; func div41
; in    a0..a3  divident (argument to divide)
;   b0  divider
; out   a0..a3  result
;   d0  reminder
;
_div41: clr d0          ; d will contain the remainder
    ldi w,32            ; load bit counter
__d41:  ROL5    d0,a3,a2,a1,a0  ; shift carry into result c
    sub d0, b0          ; subtract b from remainder
    brcc    PC+2
    add d0, b0          ; restore if remainder became negative
    DJNZ    w,__d41     ; Decrement and Jump if bit-count Not Zero
    ROL4    a3,a2,a1,a0 ; last shift (carry into result c)
    COM4    a3,a2,a1,a0 ; complement result
    ret
```

```
/*
    Librarie sound
    340997  Rim El Qabli
    341115  Frottier Zoé
 */
; file: sound.asm   target ATmega128L-4MHz-STK300
; purpose library, sound generation

sound:
; in    a0  period of oscillation (in 10us)
;   b0  duration of sound (in 2.5ms)

    mov b1,b0           ; duration high byte = b
    clr b0              ; duration  low byte = 0
    clr a1              ; period high byte = a
    cpi a0,1
    breq    sound_off   ;Si a0=1 ne fait pas de son
sound1:
    mov w,a0
    rcall   wait8us     ; 9us
    nop                 ; 0.25us
    dec w               ; 0.25us
    brne    PC-3        ; 0.50us    total = 10us
    INVP    PORTE,SPEAKER   ; invert piezo output
    sub b0,a0           ; decrement duration low  byte
    sbc b1,a1           ; decrement duration high byte
    brcc    sound1      ; continue if duration>0
    ret

sound_off:
    ldi a0,1
    rcall   wait4us
    sub b0,a0           ; decrement duration low  byte
    sbc b1,a1           ; decrement duration high byte
    brcc    PC-3        ; continue if duration>0
    ret

; === wait routines ===

wait9us:rjmp    PC+1        ; waiting 2 cycles
    rjmp    PC+1            ; waiting 2 cylces
wait8us:rcall   wait4us     ; recursive call with "falling through"
wait4us:rcall   wait2us
wait2us:nop
    ret     ; rcall(4), nop(1), ret(3) = 8cycl. (=2us)

; === calculation of the musical scale ===

; period (10us) = 100'000/freq(Hz)
.equ    do  = 100000/517    ; (517 Hz)
.equ    dom = do*944/1000   ; do major
.equ    re  = do*891/1000
.equ    rem = do*841/1000   ; re major
```

```
.equ    mi  = do*794/1000
.equ    fa  = do*749/1000
.equ    fam = do*707/1000   ; fa major
.equ    so  = do*667/1000
.equ    som = do*630/1000   ; so major
.equ    la  = do*595/1000
.equ    lam = do*561/1000   ; la major
.equ    si  = do*530/1000

.equ    do2 = do/2
.equ    dom2    = dom/2
.equ    re2 = re/2
.equ    rem2    = rem/2
.equ    mi2 = mi/2
.equ    fa2 = fa/2
.equ    fam2    = fam/2
.equ    so2 = so/2
.equ    som2    = som/2
.equ    la2 = la/2
.equ    lam2    = lam/2
.equ    si2 = si/2

.equ    do3 = do/4
.equ    dom3    = dom/4
.equ    re3 = re/4
.equ    rem3    = rem/4
.equ    mi3 = mi/4
.equ    fa3 = fa/4
.equ    fam3    = fam/4
.equ    so3 = so/4
.equ    som3    = som/4
.equ    la3 = la/4
.equ    lam3    = lam/4
.equ    si3 = si/4
```

```
; file  wire1.asm   target ATmega128L-4MHz-STK300
; purpose Dallas 1-wire(R) interface library

; === definitions ===
.equ    DQ_port = PORTB
.equ    DQ_pin  = DQ

.equ    DS18B20     = 0x28

.equ    readROM     = 0x33
.equ    matchROM    = 0x55
.equ    skipROM     = 0xcc
.equ    searchROM   = 0xf0
.equ    alarmSearch = 0xec

.equ    writeScratchpad = 0x4e
.equ    readScratchpad  = 0xbe
.equ    copyScratchpad  = 0x48
.equ    convertT    = 0x44
.equ    recallE2    = 0xb8
.equ    readPowerSupply = 0xb4

; === routines ===

.macro  WIRE1   ; t0,t1,t2
    sbi DQ_port-1,DQ_pin    ; pull DQ low (DDR=1 output)
    ldi w,(@0+1)/2
    rcall   wire1_wait      ; wait low time (t0)
    cbi DQ_port-1,DQ_pin    ; release DQ (DDR=0 input)
    ldi w,(@1+1)/2
    rcall   wire1_wait      ; wait high time (t1)
    in  w,DQ_port-2         ; sample line (PINx=PORTx-2)
    bst w,DQ_pin            ; store result in T
    ldi w,(@2+1)/2
    rcall   wire1_wait      ; wait separation time (t2)
    ret
    .endmacro

wire1_wait:
    dec w                   ; loop time 2usec
    nop
    nop
    nop
    nop
    nop
    brne    wire1_wait
    ret

wire1_init:
    cbi DQ_port,  DQ_pin    ; PORT=0 low (for pull-down)
    cbi DQ_port-1,DQ_pin    ; DDR=0 (input hi Z)
    ret
```

```
wire1_reset:    WIRE1   480,70,410
wire1_write0:   WIRE1   56,4,1
wire1_write1:   WIRE1   1,59,1
wire1_read1:    WIRE1   1,14,45

wire1_write:
    push    a1
    ldi a1,8
    ror a0

    brcc    PC+3                ; if C=1 then wire1, else wire0
    rcall   wire1_write1
    rjmp    PC+2
    rcall   wire1_write0

    DJNZ    a1,wire1_write+2    ; dec and jump if not zero
    pop a1
    ret

wire1_read:
    push    a1
    ldi a1,8
    ror a0
    rcall   wire1_read1         ; returns result in T
    bld a0,7                    ; move T to MSb
    DJNZ    a1,wire1_read+2     ; dec and jump if not zero
    pop a1
    ret

wire1_crc:
    ldi w,0b00011001
    ldi a2,8
crc1:   ror a0
    brcc    PC+2
    eor a1,w
    bst a1,0
    ror a1
    bld a1,7
    DJNZ    a2,crc1
    ret
```