Lilian Laporte
Dimitri Jacquemont

31/05/2021
BA4 G 96

Rapport de projet

Titre du projet : Thermostat pour moteur de voiture : “Thermotor”

Dans le cadre du cours de Microcontrôleurs (BA4), nous avons travaillé sur un projet autour de la carte STK-300. Pour ce projet, l’utilisation du **capteur de température 1-Wire** est obligatoire et nous utilisons également **l'affichage LCD**, le **servomoteur Futaba S3003,** les **boutons poussoirs** et le **buzzer.**

Description générale de l’application

Pour ce projet, nous avons mis au point un appareil permettant de contrôler la température d’un environnement en continu. Il peut être utilisé afin de contrôler des pales ou une valve pour laisser passer un flux d'air frais dans l’environnement désiré et de faire sonner une alarme si la température est trop élevée. Ce système peut donc être utilisé pour refroidir un moteur de voiture grâce à l’air frais extérieur qui rentre sous le cas lorsque la voiture roule.

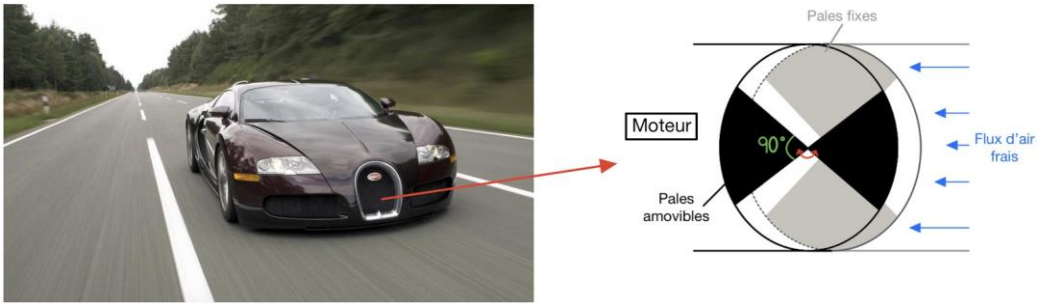
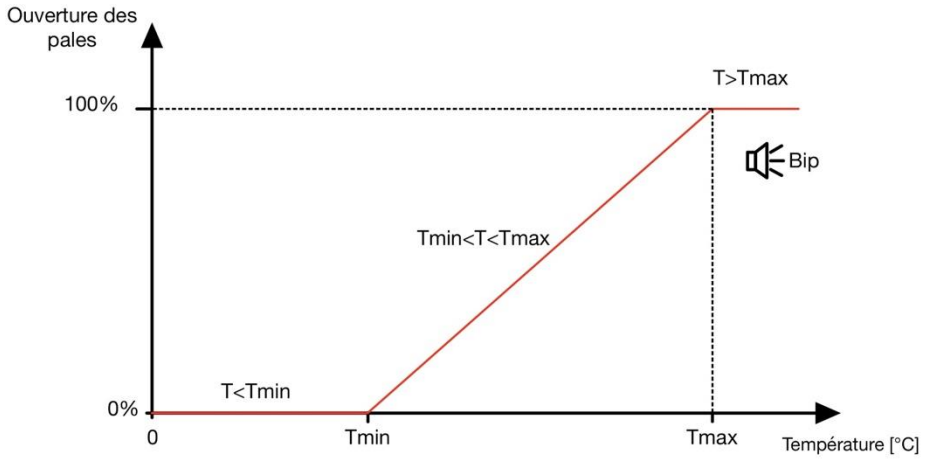


Schéma du fonctionement du Thermotor

L’utilisateur peut choisir une température minimum, qui correspond à la fermeture totale des pales, et une température maximum qui correspond à l’ouverture complète des pales. Si la température se trouve entre ces paramètres, l’ouverture des pales se fait linéairement. De plus, si $T>T_{max}$ alors une sonnerie d’alarme retentit pour prévenir que le moteur est trop chaud.



Ouverture des pales en fonction de T

Mode d'emploi

Connecter au port E le module M4 et brancher le moteur sur les pins P7 de ce module. Connecter également au port B le capteur de température et l'écran LCD aux pins situés en haut de la carte. Finalement, relier les boutons au port D avec des câbles plats puis allumer la carte avec le switch ON/OFF. Comme les ports A et C sont déjà occupés par le LCD et que le port F est seulement une entrée, nous ne pouvions pas connecter le buzzer en même temps que le moteur. Il faudra donc utiliser la carte avec le moteur et le buzzer séparément sur le port E et changer ces paramètres dans le code (lors d'une application réaliste l'utilisateur ne doit pas faire ceci mais c'est seulement avec notre carte qu'il faut changer le code) :

- ```
  149      OUTI    DDRE,0xff       ; configure portE to output
  220     MOTOR b1,b0         ;Activate it when the motor is connected
  234     MOTOR b1,b0         ;Activate it when the motor is connected
  279     MOTOR b1,b0         ;Activate it when the motor is connected
  ```
  Laisser ces lignes actives si le moteur est utilisé ou les mettre en commentaire si le buzzer est utilisé

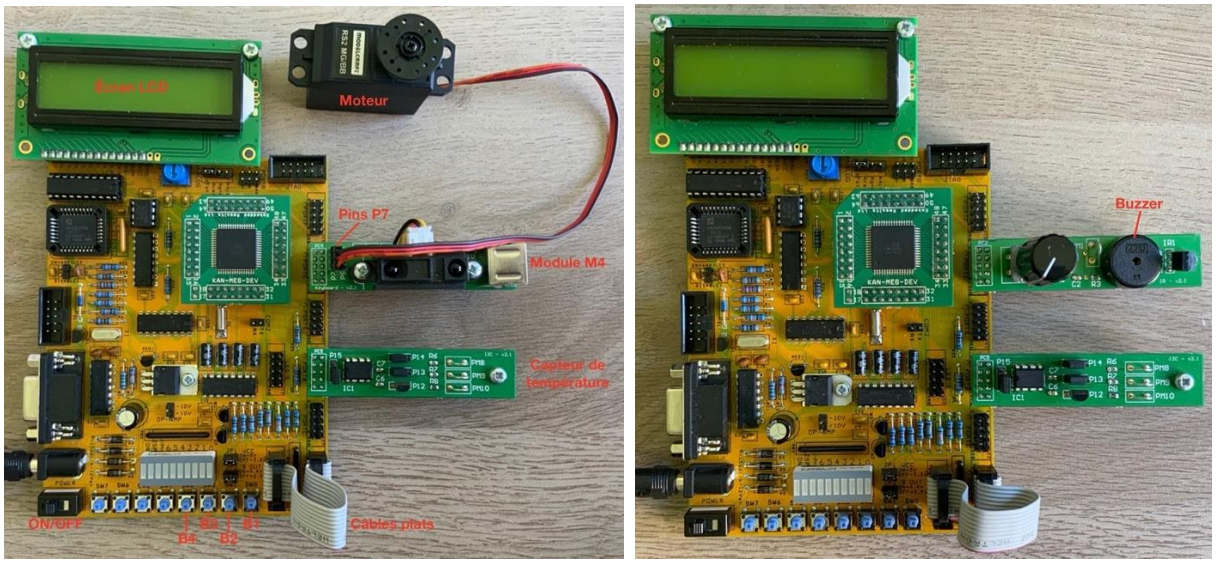
- ```
  150      ;sbi    DDRE,SPEAKER    ; make pin SPEAKER an output
  235     ;rcall alarm        ;Activate it when the speaker is connected
  ```
  Laisser ces lignes actives si le buzzer est utilisé ou les mettre en commentaire si le moteur est utilisé

Nom complet	Diminutif	Fonction
Écran LCD	LCD	Affiche les informations utiles à l'utilisateur
Moteur	-	Tourne à une position précise en fonction de la température
Capteur de température	Capteur	Récupère la température actuelle de la pièce
Buzzer	-	Émet du son lorsque $T>T_{max}$
switch ON/OFF	switch	Allume/Éteint la carte
Bouton 1	B1	Augmente T_{min}
Bouton 2	B2	Diminue T_{min}
Bouton 3	B3	Augmente T_{max}
Bouton 4	B4	Diminue T_{max}

Tableau récapituatif des périphériques

Au démarrage, le LCD affiche les valeurs de la température actuelle et de T_{min} et T_{max}.
- Pour augmenter T_{min}, appuyer sur B1 et pour le diminuer, appuyer sur B2.
- Pour augmenter T_{max}, appuyer sur B3 et pour le diminuer, appuyer sur B4.

Suite à ces modifications, le LCD et le moteur se mettent à jour continuellement.



Images de la carte avec les périphériques

Description technique de l'application et du matériel

Ports utilisés :

- **PORT E** : ce port est bidirectionnel donc le moteur peut être branché sur ce port (avec l'intermédiaire du module M4 dans notre cas) et recevoir les informations nécessaires. Le buzzer peut donc être également branché dessus.
- **PORT B** : ce port est également bidirectionnel et est prédéfini pour que le capteur de température s'y trouve. Nous l'avons donc branché ici.
- **PORT D** : une nouvelle fois ce port est bidirectionnel. De plus, les lignes d'interruptions INT0 à INT7 sont câblées sur ce port donc il est naturel de l'utiliser pour que les boutons provoquent des interruptions.

Les paramètres d'utilisation sont changés à l'aide des boutons B1 à B4. Ces boutons activent donc des interruptions asynchrones à flanc montant car l'utilisateur peut appuyer sur ceux-ci à n'importe quel moment. Une autre interruption est effectuée par le timer TCCR0 afin de mettre à jour la température actuelle de l'environnement.

Périphériques :

- **Capteur de température** : Nous utilisons seulement ce capteur pour recevoir la température, nous n'entrons aucun paramètre supplémentaire dans celui-ci. Comme la récupération de la température demande un temps considérable (750 ms dans notre cas), nous avons décidé d'implémenter cette partie du code dans une interruption. Comme notre système n'a pas besoin d'être réactif à la milliseconde près, cette interruption est déclenchée à l'aide d'un timer overflow qui se déclenche toutes les secondes.
- **Moteur** : il se positionne selon un angle entre 0° et 90° en fonction de la température. Comme dit précédemment, sa mise à jour n'a pas besoin d'être très précise dans le temps donc le code qui lui est associée se trouve dans le main.
- **Buzzer** : l'utilisation du buzzer dans notre projet est très brève et fait office d'alarme lorsque T_{max} est dépassé.
- **Affichage LCD** : les paramètres actuels sont affichés sur l'interface graphique comme montré ci-dessous. Nous y affichons la température actuelle ainsi que T_{min} et T_{max}.

Photo du LCD lorsqu'il est actif

Fonctionnement du programme

Récupération de la température

Comme expliqué précédemment, la mesure de la température par le capteur prend un certain temps et peut donc se faire dans une interruption afin de ne pas ralentir le reste du programme et donc de fonctionner en multitâche. En vue d'effectuer une interruption toute les seconde (pas nécessaire d'en faire plus souvent), nous avons sélectionné l'horloge allant à 32'768 Hz (AS = 1), ainsi qu'un prescaler à 128 (avec TCCR0), abaissant la fréquence à 256 Hz (voir table 11.1 p.189 du polycopié).

Une fois rentré dans l'interruption, nous utilisons la librairie *wire1.asm* dans l'intention de communiquer avec le capteur. Lors de la récupération, un pulse de reset est envoyé au capteur. Suite à cela, l'instruction de convertir la conversion de température d'analogique à numérique est envoyée. Une fois la valeur de la température reçue et lue, nous transférons la partie entière de la température dans le registre b3 car notre système n'a pas besoin d'être précis au dixième de degré. Finalement, nous utiliserons b3 pour toutes les futures opérations futures sur la température.

Affichages des différentes températures

Dans le but d'afficher les différentes informations sur le LCD, nous commençons par "nettoyer l'écran de ce qu'il y avait précédemment" et mettons le curseur en position initiale ; l'affichage est prêt à l'emploi. Nous procédons ensuite par l'affichage de chaînes de caractères et de variables à l'aide de la librairie *printf.asm* (choisie pour sa flexibilité d'utilisation). Nous avons choisi d'afficher en haut de l'écran la température actuelle, et en bas la valeur des variables T_{min} et T_{max}.

Calcul du pulse à envoyer au moteur

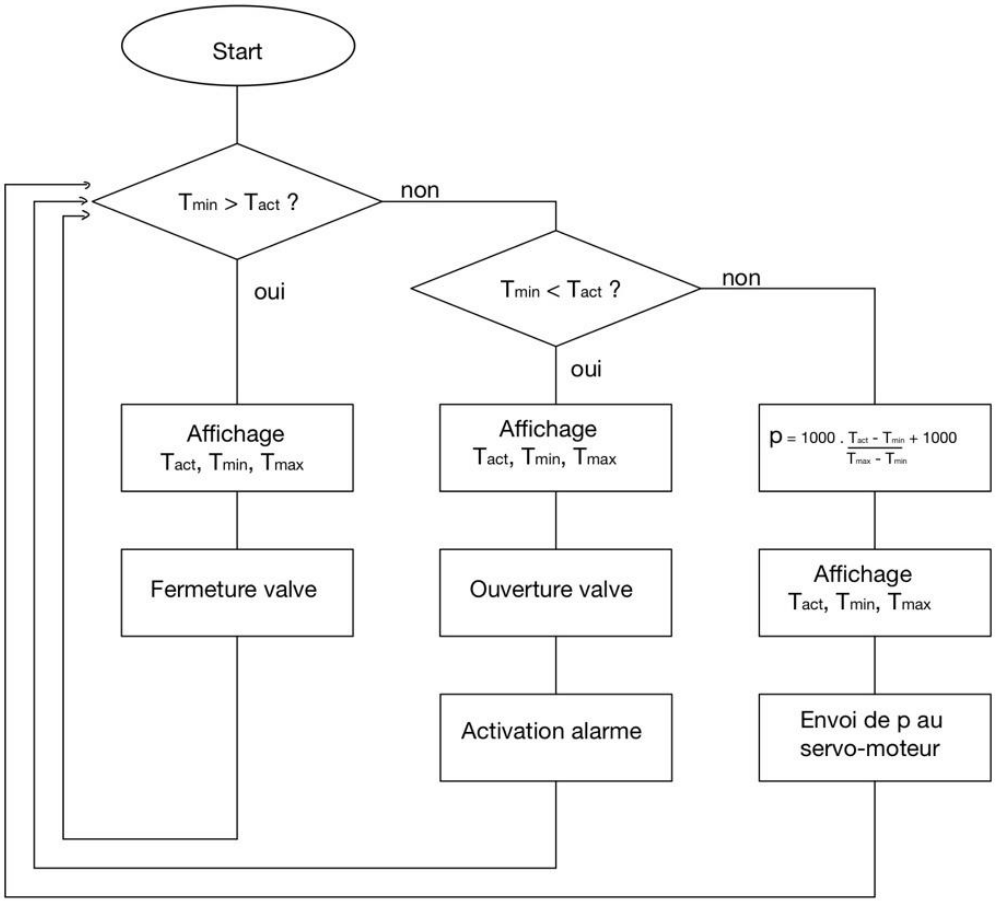
Pour effectuer les différents calculs de cette partie, nous avons choisi d'utiliser la librairie *math.asm*. La formule souhaitée pour la valeur du pulse est : $p = 1000 \cdot \frac{T_{act} - T_{min}}{T_{max} - T_{min}} + 1000$. L'ordre des opérations ayant un réel impact sur la précision du résultat, nous avons commencé par calculer la valeur de $T_{act} - T_{min}$ pour ensuite la multiplier par 1000 à l'aide de la routine *mul21*. Suite à cela le dénominateur $T_{max} - T_{min}$ est calculé, puis nous finissons le calcul par la division et nous utilisons donc la routine *div21* pour obtenir le second terme de p. Un offset de 1000 est nécessaire pour avoir une valeur entre 1000 et 2000 (temps du pulse haut) et donc pour faire fonctionner le moteur et nous utilisons donc la macro *ADDI2* provenant de la librairie macro.asm.

Envoie du pulse au moteur

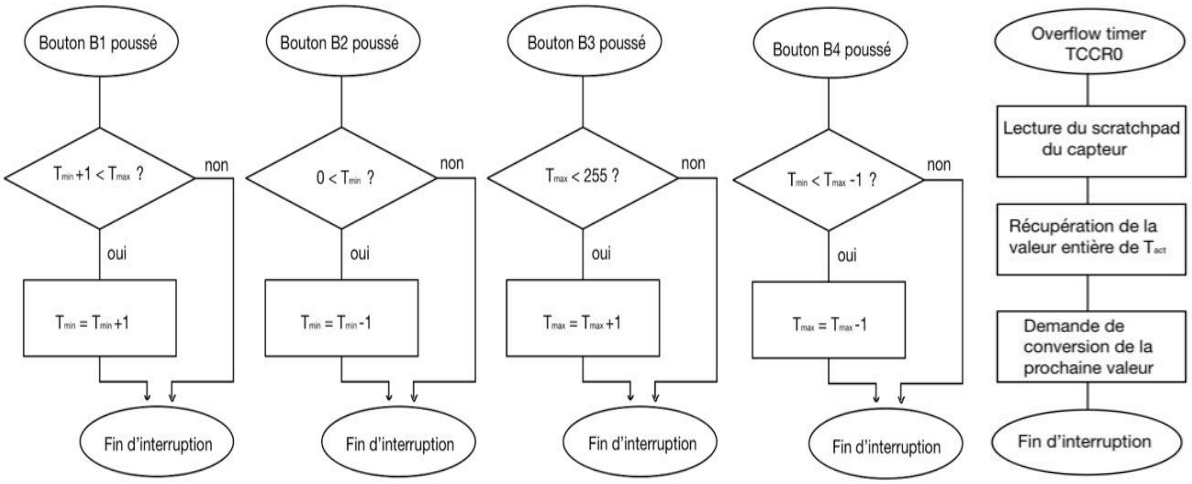
Le pin qui envoie le signal au moteur est d'abord mis à 0 durant 20ms puis est mis à 1 le temps que p décrémente jusqu'à 0. Le moteur ne peut donc pas recevoir plus d'un signal toutes les 20 ms.

Activation de l'alarme

Le code pour activer l'alarme se trouve dans la sous-routine *alarme*. Afin de l'activer, la sous-routine est appelée depuis la sous-routine où passe le PC lorsque T_{act} est supérieur à T_{max}. La sous-routine en question utilise la librairie *sound.asm*. Notre son est paramétré en posant a0 à 250 (ce qui correspond à une fréquence de 400 Hz) et b0 à 20 (ce qui correspond à une durée de son de 50 ms).



Organigramme du main



Organigrammes des interruptions avec les boutons

Organigramme de l'interruption pour le capteur

Présentation des modules

Notre code utilise les différents modules et différentes librairies mis à disposition par le professeur. Nous avons également divisé notre code en macros et sous-routines afin d'en avoir une meilleure compréhension.

- Modules disponibles sur moodle :
  - *defintions.asm* : permet d'utiliser certaines notations avec les registres comme a0,a1,...etc.
  - *macros.asm* : utile pour certaines macros de calcul comme *ADDI2* ou encore des macros plutôt pratiques comme *OUTI*.
  - *printf.asm* et *lcd.asm* : très pratique et très facile pour écrire sur le LCD avec par exemple la macro *PRINTF*.
  - *wire1.asm* : sert à communiquer en 1-Wire et donc nous a permis de communiquer avec le capteur de température.
  - *sound.asm* : permet d'émettre du son avec un code très compact en utilisant la sous-routine *sound*.
- Librairie :
  - *math.asm* : nous avons utilisé les sous-routines *mul21* et *div21* et qui permettent donc de faire un calcul compliqué en une unique ligne.
- Macros ; utilisées lorsqu'un code est écrit plusieurs fois et qu'il est préférable de ne pas rajouter des cycles d'horloge supplémentaires :
  - *MOTOR* : elle a été créée car le moteur est utilisé à 3 endroits différents. Elle prend en entrée un nombre sur 16 bits (soit 2 registres comme b0 et b1) et permet donc de fixer le moteur à une position précise.
  - *AFFICHER* : comme pour *MOTOR*, cette macro est utilisée dans 3 endroits différents. Elle prend en entrée T_{act}, T_{max} puis T_{min} et permet donc d'afficher ces valeurs comme expliqué précédemment.
- Sous-routines ; permettent de rendre compact le main et donc d'avoir une meilleure lisibilité du code :
  - *tmin* : affiche les paramètres sur le LCD et tourne le moteur à 0° lorsque $T_{min}>T_{act}$.
  - *tmax* : affiche les paramètres sur le LCD, tourne le moteur à 90° et active l'alarme lorsque $T_{act}>T_{max}$.
  - *conversion* : effectue le calcul $p = 1000 \cdot \frac{T_{act}-T_{min}}{T_{max}-T_{min}} + 1000$ puis affiche les paramètres sur le LCD et fait tourner le moteur en fonction de la valeur de p lorsque $T_{min}<T_{act}<T_{max}$

- alarm : Elle permet simplement de rendre le code compact

Description de détail de l'accès aux périphériques, références

- **Capteur de température 1-Wire**

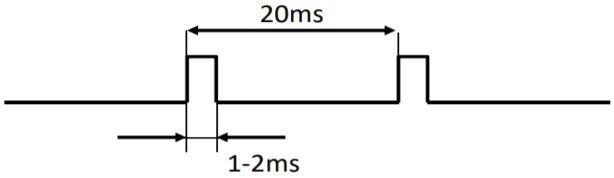
Ce capteur de température communique avec le microcontrôleur en 1-Wire, c'est-à-dire qu'il peut à la fois recevoir et envoyer des informations au capteur en utilisant la modulation par largeur d'impulsion de 1 à 60 us. La communication 1-Wire permet également au maître de communiquer avec plusieurs esclaves. Dans notre cas, le maître communique seulement avec un esclave qui est le capteur de température.

- **Affichage LCD Hitachi 44780**

L'écran LCD permet un affichage de 2x16 caractères. Il est possible d'écrire sur l'écran avec le module LCD mais le plus simple reste d'utiliser le module *PRINTF*. Ce dernier permet une communication vers différents périphériques tels que le LCD, l'UART, l'I2C. Les sous-routines qui lui sont associées permettent alors d'écrire sur le LCD avec une grande facilité et une bonne compréhension pour le lecteur.

- **Moteur servo Futaba S3003**

Les servomoteurs sont des moteurs qui ont pour fonction principale de maintenir une position fixe tout en pouvant supporter des charges élevées. Ils se différencient donc des moteurs pas à pas qui eux ne peuvent pas supporter de grosses charges. Ce moteur est de plus contrôlé en valeur angulaire ce qui change du servo SG90 qui est contrôlé en vitesse. Le contrôle du moteur se fait grâce à la modulation de la largeur d'impulsion (PWM : Pulse Width Modulation). On peut donc modifier la taille du signal haut entre 1 et 2 ms sur une période fixe de 20 ms comme montré ci-dessous :



Graphique de la modulation de largeur d'impulsion du moteur S3003.

- **Buzzer**

Le buzzer est un périphérique permettant de communiquer facilement avec l'utilisateur. La librairie *sound.asm* (plus particulièrement la routine *sound*) permet d'envoyer un signal rectangulaire paramétré en fréquence et en durée au haut-parleur piézo-électrique. Le contrôle du buzzer se fait grâce la modulation de la taille de la période (fréquence du son, contrôlé par a0) et le nombre de fois que la période du signal est répétée (contrôlé par b0). Une fois le signal reçu, ce dernier produira un beep.

Conclusion

Pour conclure, ce projet a été très intéressant du fait qu'on ait choisi notre propre système et que nous ayons obtenu un résultat concret et fonctionnel. Nous avons rencontré de nombreuses difficultés car sommes encore novices mais nous avons réussi à les surmonter avec persévérance. Ce projet nous a aider à mieux comprendre comment un microcontrôleur fonctionne.

Annexe

main.asm

```
;====================MACROS===========================
 .macro       MOTOR
       P0     PORTE,SERVO1  ;low
       WAIT_US 20000
       P1     PORTE,SERVO1
loop:
       SUBI2  @0,@1,0x1
       brne   loop
       P0     PORTE,SERVO1  ;high
       .endmacro

.macro AFFICHER
       rcall  lcd_home

       mov a0,@0
       PRINTF LCD
       .db    "T=",FDEC,a,"C ",CR,0  ;display temperature

       mov a0, @1
       mov b0, @2
       PRINTF LCD
       .db    LF,"Tmin=",FDEC,b," Tmax=",FDEC,a,"    ",0  ;display Tmin & Tmax
       .endmacro


.include "definitions.asm"
.include "macros.asm"

.org   0
       rjmp   reset
.org INT0addr              ;Button 0
       jmp ext_int0
.org INT1addr              ;Button 1
       jmp ext_int1
.org INT2addr              ;Button 2
       jmp ext_int2
.org INT3addr              ;Button 3
       jmp ext_int3
.org   OVF0addr            ; timer overflow 0 interrupt vector
       rjmp   overflow0
.org   0x30

; =============== interrupt service routines ================

ext_int0:                                ;Button 0
       in     _sreg,SREG

       ldi a0, 0b00000010
       mov a1, d1
       sub a1, a0                 ;Tmax-2

       mov a0, c3

       sub a1, a0                 ;Tmax-1-Tmin
       brsh infcorrect            ;Tmin<Tmax-1?

       out    SREG,_sreg
       reti
infcorrect:
```

```
	inc c3
	out	SREG,_sreg
	reti


ext_int1:                               ;Button 1
	in	_sreg,SREG

	ldi a0, 0b00000001
	mov a1, c3
	sub a1, a0
	ldi a0, 0b11111111
	sub a1, a0

	breq end2                       ;Tmin>0?
	dec c3

	out	SREG,_sreg
	reti
end2:
	out	SREG,_sreg
	reti


ext_int2:                               ;Button 2
	in	_sreg,SREG

	ldi a0, 0b00000001
	add a0, d1

	breq end1                       ;Tmax<255?
	inc d1

	out	SREG,_sreg
	reti
end1:
	out	SREG,_sreg
	reti


ext_int3:                               ;Button 3
	in	_sreg,SREG
	dec d1

	ldi a0, 0b00000001
	add a0, c3

	sub a0, d1
	brsh correctsup                 ;Tmin<Tmax-1?

	out	SREG,_sreg
	reti
correctsup:
	mov d1, c3
	ldi a0, 0b00000001
	add d1, a0

	out	SREG,_sreg
	reti

overflow0:
	in _sreg,SREG
```

```
        rcall  wire1_reset              ; send a reset pulse
        CA     wire1_write, skipROM
        CA     wire1_write, readScratchpad
        rcall  wire1_read               ; read temperature LSB
        MOVB b3,3,a0,7
        MOVB b3,2,a0,6
        MOVB b3,1,a0,5
        MOVB b3,0,a0,4
        clr a0
        rcall  wire1_read               ; read temperature MSB
        MOVB b3,6,a0,2
        MOVB b3,5,a0,1
        MOVB b3,4,a0,0                        ;b3=Val entière de la température

        rcall  wire1_reset              ; send a reset pulse
        CA     wire1_write, skipROM; skip ROM identification
        CA     wire1_write, convertT      ; initiate temp conversion
        out    SREG,_sreg
        reti

; =================================== initialisation (reset)
================================
reset:
        LDSP   RAMEND              ; set up stack pointer (SP)
        rcall  wire1_init          ; initialize 1-wire(R) interface

        OUTI   DDRE,0xff           ; configure portE to output
        ;sbi   DDRE,SPEAKER  ; make pin SPEAKER an output

        rcall  LCD_init            ; initialize the LCD

        ldi r16,0x00  ;configure portD as input
        out DDRD,r16

        OUTI   EIMSK,0b11001111
        OUTEI  EICRA,0b11111111
        OUTI   TIMSK,(1<<TOIE0)
        OUTI   ASSR, (1<<AS0)      ; clock from TOSC1 (external)
        OUTI   TCCR0,5                    ; CS0=1 CK/256

        ldi a2, 40
        mov d1,a2     ; Tmax
        ldi a2,25
        mov b3,a2     ; Tact
        ldi a2,2
        mov c3,a2     ; Tmin

        rcall  wire1_reset              ; send a reset pulse
        CA     wire1_write, skipROM; skip ROM identification
        CA     wire1_write, convertT      ; initiate temp conversion

        sei                                          ; set global interrupt
        rjmp main


.include "lcd.asm"              ; include the LCD routines
.include "printf.asm"           ; include formatted print routines
.include "wire1.asm"       ; include Dallas 1-wire(R) routines
.include "math.asm"             ; include math routines
.include "sound.asm"       ; include sound routines
```

```
main:
	in _sreg,SREG
	cli
	mov a0,c3
	sub a0, b3 ; T<Tmin ?

	brpl t_min

	mov a0,d1
	sub a0, b3 ; T>Tmax ?

	brcs t_sup

	out SREG,_sreg
	sei

	rcall conversion

	rjmp main

t_min:
	rcall tmin
	rjmp main

t_sup:
	rcall tsup
	rjmp main

; =======SOUS-ROUTINES=======

; ====TMIN====
tmin:
	AFFICHER b3,d1,c3

	ldi b0, 0b11101000
	ldi b1, 0b00000011  ;b=paramters for the motor
	MOTOR b1,b0               ;Activate it when the motor is connected

	out SREG,_sreg
	sei
	rjmp main

	ret

; ====TSUP====
tsup:
	AFFICHER b3,d1,c3

	ldi b0, 0b11010000
	ldi b1, 0b00000111  ;b=paramters for the motor
	MOTOR b1,b0               ;Activate it when the motor is connected
	;rcall alarm        ;Activate it when the speaker is connected

	out SREG,_sreg
	sei

	ret

;====CONVERSION====
conversion :
	in _sreg,SREG
	cli
```

```
        mov b0, b3 ; b0 = Tact
        mov a0, c3 ; a0 = Tmin

        sub b0, a0 ; b0 = Tact-Tmin

        clr c0
        clr c1

        ldi a0, 0b11101000
        ldi a1, 0b00000011

        rcall mul21   ;c = 1000*(Tact-Tmin)

        mov a0, c0
        mov a1, c1

        mov b0, d1 ; b0 = Tmax
        mov b1, c3 ; b1 = Tmin

        sub b0, b1 ; b0 = Tmax - Tmin

        rcall div21 ; c = 1000*(Tact-Tmin)/(Tmax-Tmin)

        mov d2,c1
        mov b2,c0

        AFFICHER b3,d1,c3

        mov b0,b2
        mov b1,d2

        ADDI2  b1,b0,1000    ; add an offset of 1000
        MOTOR b1,b0                ;Activate it when the motor is connected

        out SREG,_sreg
        sei

        ret

; ====ALARM====
alarm:
        ldi a0,250    ;parameter for the routine sound
        ldi b0,20     ;parameter for the routine sound
        rcall sound
        ret
```

definitions.asm

```
; file:      definitions.asm   target ATmega128L-4MHz-STK300
; purpose library, definition of addresses and constants
; 20171114 A.S.

; === definitions  ===
.nolist                    ; do not include in listing
.set   clock  = 4000000

.def   char   = r0   ; character (ASCII)
.def   _sreg  = r1   ; saves the status during interrupts
.def   _u     = r2   ; saves working reg u during interrupt
.def   u      = r3   ; scratch register (macros, routines)

.def   e0     = r4   ; temporary reg for PRINTF
.def   e1     = r5

.equ   c      = 8
.def   c0     = r8   ; 8-byte register c
.def   c1     = r9
.def   c2     = r10
.def   c3     = r11

.equ   d      = 12   ; 4-byte register d (overlapping with c)
.def   d0     = r12
.def   d1     = r13
.def   d2     = r14
.def   d3     = r15

.def   w      = r16  ; working register for macros
.def   _w     = r17  ; working register for interrupts

.equ   a      = 18
.def   a0     = r18  ; 4-byte register a
.def   a1     = r19
.def   a2     = r20
.def   a3     = r21

.equ   b      = 22
.def   b0     = r22  ; 4-byte register b
.def   b1     = r23
.def   b2     = r24
.def   b3     = r25

.equ   px     = 26   ; pointer x
.equ   py     = 28   ; pointer y
.equ   pz     = 30   ; pointer z

; === ASCII codes
.equ   BEL    =0x07  ; bell
.equ   HT     =0x09  ; horizontal tab
.equ   TAB    =0x09  ; tab
.equ   LF     =0x0a  ; line feed
.equ   VT     =0x0b  ; vertical tab
.equ   FF     =0x0c  ; form feed
.equ   CR     =0x0d  ; carriage return
.equ   SPACE  =0x20  ; space code
.equ   DEL    =0x7f  ; delete
.equ   BS     =0x08  ; back space

; === STK-300 ===
.equ   LED    = PORTB       ; LEDs on STK-300
```

```
.equ   BUTTON = PIND ; buttons on the STK-300

; === module M2 (encoder/speaker/IR remote) ===
.equ   SPEAKER       = 2    ; piezo speaker
.equ   ENCOD_A       = 4    ; angular encoder A
.equ   ENCOD_B       = 5    ; angular encoder B
.equ   ENCOD_I       = 6    ; angular encoder button
.equ   IR     = 7    ; IR module for PCM remote control system

; === module M5 (I2C/1Wire) ===
.equ   SCL    = 0    ; I2C serial clock
.equ   SDA    = 1    ; I2C serial data
.equ   DQ     = 5    ; Dallas 1Wire
                        ; master transmitter status codes, Table 88
.equ   I2CMT_START = 0x08      ; start
.equ   I2CMT_REPSTART = 0x10   ; repeated start
.equ   I2CMT_SLA_ACK= 0x18     ; slave ack
.equ   I2CMT_SLA_NOACK = 0x20  ; slave no ack
.equ   I2CMT_DATA_ACK = 0x28   ; data write, ack
.equ   I2CMT_DATA_NOACK = 0x30 ; data write, no ack
                        ; master receiver status codes, Table 89
.equ   I2CMR_SLA_ACK = 0x40 ; slave address ack
.equ   I2CMR_SLA_NACK       = 0x48 ; slave address no ack
.equ   I2CMR_DATA_ACK = 0x50      ; master data ack
.equ   I2CMR_DATA_NACK= 0x58      ; master data no ack

; === module M4 (Keyboard/Sharp/Servo) ===
.equ   KB_CLK = 0    ; PC-AT keyboard clock line
.equ   KB_DAT = 1    ; PC-AT keyboard data line
.equ   GP2_CLK       = 2    ; Sharp GP2D02 distance measuring sensor
.equ   GP2_DAT       = 3    ; Sharp GP2D02 distance measuring sensor
.equ   GP2_AVAL = 3; Shart GP2Y0A21 distance measuring sensor
.equ   SERVO1 = 4    ; Futaba position servo

; === module M3 (potentiometer/BNC) ===
.equ   POT    = 0    ; potentiometer
.equ   BNC1   = 2    ; BNC input
.equ   BNC2   = 4    ; BNC input
.list
```

macros.asm

```
; file:       macros.asm   target ATmega128L-4MHz-STK300
; purpose library, general-purpose macros
; author (c) R.Holzer (adapted MICRO210/EE208 A.Schmid)
; v2019.01 20180820 AxS

; ==============
;      pointers
; ==============

; --- loading an immediate into a pointer XYZ,SP ---
.macro       LDIX   ; sram
       ldi    xl, low(@0)
       ldi    xh,high(@0)
       .endmacro
.macro       LDIY   ; sram
       ldi    yl, low(@0)
       ldi    yh,high(@0)
       .endmacro
.macro       LDIZ   ; sram
       ldi    zl, low(@0)
       ldi    zh,high(@0)

       .endmacro
.macro LDZD   ; sram, reg   ; sram+reg -> Z
       mov    zl,@1
       clr    zh
       subi   zl, low(-@0)
       sbci   zh,high(-@0)
       .endmacro
.macro LDSP   ; sram
       ldi    r16, low(@0)
       out    spl,r16
       ldi    r16,high(@0)
       out    sph,r16
       .endmacro

; --- load/store SRAM addr into pointer XYZ ---
.macro       LDSX   ; sram
       lds    xl,@0
       lds    xh,@0+1
       .endmacro
.macro       LDSY   ; sram
       lds    yl,@0
       lds    yh,@0+1
       .endmacro
.macro       LDSZ   ; sram
       lds    zl,@0
       lds    zh,@0+1
       .endmacro
.macro       STSX   ; sram
       sts    @0,  xl
       sts    @0+1,xh
       .endmacro
.macro       STSY   ; sram
       sts    @0,  yl
       sts    @0+1,yh
       .endmacro
.macro       STSZ   ; sram
       sts    @0,  zl
       sts    @0+1,zh
       .endmacro
```

```
; --- push/pop pointer XYZ ---
.macro PUSHX                ; push X
       push   xl
       push   xh
       .endmacro
.macro POPX                 ; pop X
       pop    xh
       pop    xl
       .endmacro

.macro PUSHY                ; push Y
       push   yl
       push   yh
       .endmacro
.macro POPY                 ; pop Y
       pop    yh
       pop    yl
       .endmacro

.macro PUSHZ                ; push Z
       push   zl
       push   zh
       .endmacro
.macro POPZ                 ; pop Z
       pop    zh
       pop    zl
       .endmacro

; --- multiply/divide Z ---
.macro MUL2Z                ; multiply Z by 2
       lsl    zl
       rol    zh
       .endmacro
.macro DIV2Z                ; divide Z by 2
       lsr    zh
       ror    zl
       .endmacro

; --- add register to pointer XYZ ---
.macro ADDX   ;reg          ; x <- y+reg
       add    xl,@0
       brcc   PC+2
       subi   xh,-1         ; add carry
       .endmacro
.macro ADDY   ;reg          ; y <- y+reg
       add    yl,@0
       brcc   PC+2
       subi   yh,-1         ; add carry
       .endmacro
.macro ADDZ   ;reg          ; z <- z+reg
       add    zl,@0
       brcc   PC+2
       subi   zh,-1         ; add carry
       .endmacro

; ====================
;      miscellaneous
; ====================

; --- output/store (regular I/O space) immediate value ---
.macro OUTI   ; port,k      output immediate value to port
```

```
        ldi     w,@1
        out     @0,w
        .endmacro

; --- output/store (extended I/O space) immediate value ---
.macro OUTEI  ; port,k    output immediate value to port
        ldi     w,@1
        sts     @0,w
        .endmacro

; --- add immediate value ---
.macro ADDI
        subi    @0,-@1
        .endmacro
.macro ADCI
        sbci    @0,-@1
        .endmacro

; --- inc/dec with range limitation ---
.macro INC_LIM        ; reg,limit
        cpi     @0,@1
        brlo    PC+3
        ldi     @0,@1
        rjmp    PC+2
        inc     @0
        .endmacro

.macro DEC_LIM        ; reg,limit
        cpi     @0,@1
        breq    PC+5
        brlo    PC+3
        dec     @0
        rjmp    PC+2
        ldi     @0,@1
        .endmacro

; --- inc/dec with cyclic range ---
.macro INC_CYC        ; reg,low,high
        cpi     @0,@2
        brsh    _low    ; reg>=high then reg=low
        cpi     @0,@1
        brlo    _low    ; reg< low  then reg=low
        inc     @0
        rjmp    _done
_low:   ldi     @0,@1
_done:
    .endmacro

.macro DEC_CYC        ; reg,low,high
        cpi     @0,@1
        breq    _high   ; reg=low then reg=high
        brlo    _high   ; reg<low then reg=high
        dec     @0
        cpi     @0,@2
        brsh    _high   ; reg>=high then high
        rjmp    _done
_high:  ldi     @0,@2
_done:
        .endmacro

.macro INCDEC ;port,b1,b2,reg,low,high
        sbic    @0,@1
```

```
        rjmp    PC+6

        cpi     @3,@5
        brlo    PC+3
        ldi     @3,@4
        rjmp    PC+2
        inc     @3

        sbic    @0,@2
        rjmp    PC+7

        cpi     @3,@4
        breq    PC+5
        brlo    PC+3
        dec     @3
        rjmp    PC+2
        ldi     @3,@5
        .endmacro

; --- wait loops ---
; wait 10...196608 cycles
.macro WAIT_C ; k
        ldi     w,  low((@0-7)/3)
        mov     u,w                     ; u=LSB
        ldi     w,high((@0-7)/3)+1      ; w=MSB
        dec     u
        brne    PC-1
        dec     u
        dec     w
        brne    PC-4
        .endmacro

; wait micro-seconds (us)
; us = x*3*1000'000/clock) ==> x=us*clock/3000'000
.macro WAIT_US ; k
        ldi     w, low((clock/1000*@0/3000)-1)
        mov     u,w
        ldi     w,high((clock/1000*@0/3000)-1)+1 ; set up: 3 cyles
        dec     u
        brne    PC-1            ; inner loop: 3 cycles
        dec     u               ; adjustment for outer loop
        dec     w
        brne    PC-4
        .endmacro

; wait mili-seconds (ms)
.macro WAIT_MS ; k
        ldi     w, low(@0)
        mov     u,w             ; u = LSB
        ldi     w,high(@0)+1    ; w = MSB
wait_ms:
        push    w               ; wait 1000 usec
        push    u
        ldi     w, low((clock/3000)-5)
        mov     u,w
        ldi     w,high((clock/3000)-5)+1
        dec     u
        brne    PC-1            ; inner loop: 3 cycles
        dec     u               ; adjustment for outer loop
        dec     w
        brne    PC-4
        pop     u
```

```
        pop     w

        dec     u
        brne    wait_ms
        dec     w
        brne    wait_ms
        .endmacro

; --- conditional jumps/calls ---
.macro JC0                  ; jump if carry=0
        brcs    PC+2
        rjmp    @0
        .endmacro
.macro JC1                  ; jump if carry=1
        brcc    PC+2
        rjmp    @0
        .endmacro

.macro JK     ; reg,k,addr  ; jump if reg=k
        cpi     @0,@1
        breq    @2
        .endmacro
.macro _JK    ; reg,k,addr  ; jump if reg=k
        cpi     @0,@1
        brne    PC+2
        rjmp    @2
        .endmacro
.macro JNK    ; reg,k,addr  ; jump if not(reg=k)
        cpi     @0,@1
        brne    @2
        .endmacro

.macro CK     ; reg,k,addr  ; call if reg=k
        cpi     @0,@1
        brne    PC+2
        rcall   @2
        .endmacro
.macro CNK    ; reg,k,addr  ; call if not(reg=k)
        cpi     @0,@1
        breq    PC+2
        rcall   @2
        .endmacro

.macro JSK    ; sram,k,addr ; jump if sram=k
        lds     w,@0
        cpi     w,@1
        breq    @2
        .endmacro
.macro JSNK   ; sram,k,addr ; jump if not(sram=k)
        lds     w,@0
        cpi     w,@1
        brne    @2
        .endmacro

; --- loops ---
.macro DJNZ   ; reg,addr    ; decr and jump if not zero
        dec     @0
        brne    @1
        .endmacro
.macro DJNK   ; reg,k,addr  ; decr and jump if not k
        dec     @0
        cpi     @0,@1
```

```
        brne   @2
        .endmacro

.macro IJNZ   ; reg,addr    ; inc and jump if not zero
        inc    @0
        brne   @1
        .endmacro
.macro IJNK   ; reg,k,addr  ; inc and jump if not k
        inc    @0
        cpi    @0,@1
        brne   @2
        .endmacro
.macro _IJNK  ; reg,k,addr  ; inc and jump if not k
        inc    @0
        ldi    w,@1
        cp     @0,w
        brne   @2
        .endmacro

.macro ISJNK  ; sram,k,addr ; inc sram and jump if not k
        lds    w,@0
        inc    w
        sts    @0,w
        cpi    w,@1
        brne   @2
        .endmacro
.macro _ISJNK ; sram,k,addr ; inc sram and jump if not k
        lds    w,@0
        inc    w
        sts    @0,w
        cpi    w,@1
        breq   PC+2
        rjmp   @2
        .endmacro

.macro DSJNK  ; sram,k,addr ; dec sram and jump if not k
        lds    w,@0
        dec    w
        sts    @0,w
        cpi    w,@1
        brne   @2
        .endmacro

; --- table lookup ---
.macro LOOKUP ;reg, index,tbl
        push   ZL
        push   ZH
        mov    zl,@1         ; move index into z
        clr    zh
        subi   zl, low(-2*@2)      ; add base address of table
        sbci   zh,high(-2*@2)
        lpm                  ; load program memory (into r0)
        mov    @0,r0
        pop    ZH
        pop    ZL
        .endmacro

.macro LOOKUP2       ;r1,r0, index,tbl
        mov    zl,@2         ; move index into z
        clr    zh
        lsl    zl            ; multiply by 2
        rol    zh
```

```
        subi    zl, low(-2*@3)      ; add base address of table
        sbci    zh,high(-2*@3)
        lpm                 ; get LSB byte
        mov     w,r0        ; temporary store LSB in w
        adiw    zl,1        ; increment Z
        lpm                 ; get MSB byte
        mov     @0,r0       ; mov MSB to res1
        mov     @1,w        ; mov LSB to res0
        .endmacro

.macro LOOKUP4      ;r3,r2,r1,r0, index,tbl
        mov     zl,@4       ; move index into z
        clr     zh
        lsl     zl          ; multiply by 2
        rol     zh
        lsl     zl          ; multiply by 2
        rol     zh
        subi    zl, low(-2*@5)      ; add base address of table
        sbci    zh,high(-2*@5)
        lpm
        mov     @1,r0       ; load high word LSB
        adiw    zl,1
        lpm
        mov     @0,r0       ; load high word MSB
        adiw    zl,1
        lpm
        mov     @3,r0       ; load low word LSB
        adiw    zl,1
        lpm
        mov     @2,r0       ; load low word MSB
        .endmacro

.macro LOOKDOWN ;reg,index,tbl
        ldi     ZL, low(2*@2) ; load table address
        ldi     ZH,high(2*@2)
        clr     @1
loop:   lpm
        cp      r0,@0
        breq    found
        inc     @1
        adiw    ZL,1
        tst     r0
        breq    notfound
        rjmp    loop
notfound:
        ldi     @1,-1
found:
        .endmacro

; --- branch table ---
.macro C_TBL  ; reg,tbl
        ldi     ZL, low(2*@1)
        ldi     ZH,high(2*@1)
        lsl     @0
        add     ZL,@0
        brcc    PC+2
        inc     ZH
        lpm
        push    r0
        lpm
        mov     zh,r0
        pop     zl
```

```
        icall
        .endmacro
.macro J_TBL  ; reg,tbl
        ldi     ZL, low(2*@1)
        ldi     ZH,high(2*@1)
        lsl     @0
        add     ZL,@0
        brcc    PC+2
        inc     ZH
        lpm
        push    r0
        lpm
        mov     zh,r0
        pop     zl
        ijmp
        .endmacro

.macro BRANCH ; reg         ; branching using the stack
        ldi     w, low(tbl)
        add     w,@0
        push    w
        ldi     w,high(tbl)
        brcc    PC+2
        inc     w
        push    w
        ret
tbl:
        .endmacro

; --- multiply/division ---
.macro DIV2   ; reg
        lsr     @0
        .endmacro
.macro DIV4   ; reg
        lsr     @0
        lsr     @0
        .endmacro
.macro DIV8   ; reg
        lsr     @0
        lsr     @0
        lsr     @0
        .endmacro

.macro MUL2   ; reg
        lsl     @0
        .endmacro
.macro MUL4   ; reg
        lsl     @0
        lsl     @0
        .endmacro
.macro MUL8   ; reg
        lsl     @0
        lsl     @0
        lsl     @0
        .endmacro

; ====================================
;       extending existing instructios
; ====================================

; --- immediate ops with r0..r15 ---
.macro _ADDI
```

```
        ldi     w,@1
        add     @0,w
        .endmacro
.macro _ADCI
        ldi     w,@1
        adc     @0,w
        .endmacro
.macro _SUBI
        ldi     w,@1
        sub     @0,w
        .endmacro
.macro _SBCI
        ldi     w,@1
        sbc     @0,w
        .endmacro
.macro _ANDI
        ldi     w,@1
        and     @0,w
        .endmacro
.macro _ORI
        ldi     w,@1
        or      @0,w
        .endmacro
.macro _EORI
        ldi     w,@1
        eor     @0,w
        .endmacro
.macro _SBR
        ldi     w,@1
        or      @0,w
        .endmacro
.macro _CBR
        ldi     w,~@1
        and     @0,w
        .endmacro
.macro _CPI
        ldi     w,@1
        cp      @0,w
        .endmacro
.macro _LDI
        ldi     w,@1
        mov     @0,w
        .endmacro

; --- bit access for port p32..p63 ---
.macro _SBI
        in      w,@0
        ori     w,1<<@1
        out     @0,w
        .endmacro
.macro _CBI
        in      w,@0
        andi    w,~(1<<@1)
        out     @0,w
        .endmacro

; --- extending branch distance to +/-2k ---
.macro _BREQ
        brne    PC+2
        rjmp    @0
        .endmacro
.macro _BRNE
```

```
        breq    PC+2
        rjmp    @0
        .endmacro
.macro _BRCS
        brcc    PC+2
        rjmp    @0
        .endmacro
.macro _BRCC
        brcs    PC+2
        rjmp    @0
        .endmacro
.macro _BRSH
        brlo    PC+2
        rjmp    @0
        .endmacro
.macro _BRLO
        brsh    PC+2
        rjmp    @0
        .endmacro
.macro _BRMI
        brpl    PC+2
        rjmp    @0
        .endmacro
.macro _BRPL
        brmi    PC+2
        rjmp    @0
        .endmacro
.macro _BRGE
        brlt    PC+2
        rjmp    @0
        .endmacro
.macro _BRLT
        brge    PC+2
        rjmp    @0
        .endmacro
.macro _BRHS
        brhc    PC+2
        rjmp    @0
        .endmacro
.macro _BRHC
        brhs    PC+2
        rjmp    @0
        .endmacro
.macro _BRTS
        brtc    PC+2
        rjmp    @0
        .endmacro
.macro _BRTC
        brts    PC+2
        rjmp    @0
        .endmacro
.macro _BRVS
        brvc    PC+2
        rjmp    @0
        .endmacro
.macro _BRVC
        brvs    PC+2
        rjmp    @0
        .endmacro
.macro _BRIE
        brid    PC+2
        rjmp    @0
```

```
        .endmacro
.macro _BRID
        brie    PC+2
        rjmp    @0
        .endmacro

; ====================
;       bit operations
; ====================

; --- moving bits ---
.macro MOVB    ; reg1,b1, reg2,b2   ; reg1,bit1 <- reg2,bit2
        bst     @2,@3
        bld     @0,@1
        .endmacro
.macro OUTB    ; port1,b1, reg2,b2  ; port1,bit1 <- reg2,bit2
        sbrs    @2,@3
        cbi     @0,@1
        sbrc    @2,@3
        sbi     @0,@1
        .endmacro
.macro INB     ; reg1,b1, port2,b2  ; reg1,bit1 <- port2,bit2
        sbis    @2,@3
        cbr     @0,1<<@1
        sbic    @2,@3
        sbr     @0,1<<@1
        .endmacro

.macro Z2C                          ; zero to carry
        sec
        breq    PC+2    ; (Z=1)
        clc
        .endmacro
.macro Z2INVC                       ; zero to inverse carry
        sec
        brne    PC+2    ; (Z=0)
        clc
        .endmacro

.macro C2Z                          ; carry to zero
        sez
        brcs    PC+2    ; (C=1)
        clz
        .endmacro

.macro B2C     ; reg,b                    ; bit to carry
        sbrc    @0,@1
        sec
        sbrs    @0,@1
        clc
        .endmacro
.macro C2B     ; reg,b                    ; carry to bit
        brcc    PC+2
        sbr     @0,(1<<@1)
        brcs    PC+2
        cbr     @0,(1<<@1)
        .endmacro
.macro P2C     ; port,b             ; port to carry
        sbic    @0,@1
        sec
        sbis    @0,@1
        clc
```

```
        .endmacro
.macro C2P      ; port,b            ; carry to port
        brcc    PC+2
        sbi     @0,@1
        brcs    PC+2
        cbi     @0,@1
        .endmacro

; --- inverting bits ---
.macro INVB     ; reg,bit           ; inverse reg,bit
        ldi     w,(1<<@1)
        eor     @0,w
        .endmacro
.macro INVP     ; port,bit          ; inverse port,bit
        sbis    @0,@1
        rjmp    PC+3
        cbi     @0,@1
        rjmp    PC+2
        sbi     @0,@1
        .endmacro
.macro INVC                         ; inverse carry
        brcs    PC+3
        sec
        rjmp    PC+2
        clc
        .endmacro

; --- setting a single bit ---
.macro SETBIT ; reg(0..7)
; in    reg (0..7)
; out   reg with bit (0..7) set to 1.
; 0=00000001
; 1=00000010
; ...
; 7=10000000
        mov     w,@0
        clr     @0
        inc     @0
        andi    w,0b111
        breq    PC+4
        lsl     @0
        dec     w
        brne    PC-2
        .endmacro

; --- logical operations with masks ---
.macro MOVMSK ; reg1,reg2,mask      ; reg1 <- reg2 (mask)
        ldi     w,~@2
        and     @0,w
        ldi     w,@2
        and     @1,w
        or      @0,@1
        .endmacro
.macro ANDMSK ; reg1,reg2,mask      ; reg1 <- ret 1 AND reg2 (mask)
        mov     w,@1
        ori     w,~@2
        and     @0,w
        .endmacro
.macro ORMSK  ; reg1,reg2,mask      ; reg1 <- ret 1 AND reg2 (mask)
        mov     w,@1
        andi    w,@2
        or      @0,w
```

```
        .endmacro

; --- logical operations on bits ---
.macro ANDB   ; r1,b1, r2,b2, r3,b3      ; reg1,b1 <- reg2,b2 AND reg3,b3
        set
        sbrs   @4,@5
        clt
        sbrs   @2,@3
        clt
        bld    @0,@1
        .endmacro
.macro ORB    ; r1,b1, r2,b2, r3,b3      ; reg1.b1 <- reg2.b2 OR reg3.b3
        clt
        sbrc   @4,@5
        set
        sbrc   @2,@3
        set
        bld    @0,@1
        .endmacro
.macro EORB   ; r1,b1, r2,b2, r3,b3      ; reg1.b1 <- reg2.b2 XOR reg3.b3
        sbrc   @4,@5
        rjmp   f1
f0:     bst    @2,@3
        rjmp   PC+4
f1:     set
        sbrc   @0,@1
        clt
        bld    @0,@0
        .endmacro

; --- operations based on register bits ---
.macro FB0    ; reg,bit           ; bit=0
        cbr    @0,1<<@1
        .endmacro
.macro FB1    ; reg,bit           ; bit=1
        sbr    @0,1<<@1
        .endmacro
.macro _FB0   ; reg,bit           ; bit=0
        ldi    w,~(1<<@1)
        and    @0,w
        .endmacro
.macro _FB1   ; reg,bit           ; bit=1
        ldi    w,1<<@1
        or     @0,w
        .endmacro
.macro SB0    ; reg,bit,addr            ; skip if bit=0
        sbrc   @0,@1
        .endmacro
.macro SB1    ; reg,bit,addr            ; skip if bit=1
        sbrs   @0,@1
        .endmacro
.macro JB0    ; reg,bit,addr            ; jump if bit=0
        sbrs   @0,@1
        rjmp   @2
        .endmacro
.macro JB1    ; reg,bit,addr            ; jump if bit=1
        sbrc   @0,@1
        rjmp   @2
        .endmacro
.macro CB0    ; reg,bit,addr            ; call if bit=0
        sbrs   @0,@1
        rcall  @2
```

```
        .endmacro
.macro CB1    ; reg,bit,addr            ; call if bit=1
        sbrc   @0,@1
        rcall  @2
        .endmacro
.macro WB0    ; reg,bit           ; wait if bit=0
        sbrs   @0,@1
        rjmp   PC-1
        .endmacro
.macro WB1    ; reg,bit           ; wait if bit=1
        sbrc   @0,@1
        rjmp   PC-1
        .endmacro
.macro RB0    ; reg,bit           ; return if bit=0
        sbrs   @0,@1
        ret
        .endmacro
.macro RB1    ; reg,bit           ; return if bit=1
        sbrc   @0,@1
        ret
        .endmacro

; wait if bit=0 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro WB0T   ; reg,bit,timeout,addr
        ldi    w,@2+1
        dec    w      ; 1 cyc
        breq   @3     ; 1 cyc
        sbrs   @0,@1  ; 1 cyc
        rjmp   PC-3   ; 2 cyc = 5 cycles
        .endmacro

; wait if bit=1 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro WB1T   ; reg,bit,timeout,addr
        ldi    w,@2+1
        dec    w      ; 1 cyc
        breq   @3     ; 1 cyc
        sbrc   @0,@1  ; 1 cyc
        rjmp   PC-3   ; 2 cyc = 5 cycles
        .endmacro

; --- operations based on port bits ---
.macro P0     ; port,bit          ; port=0
        cbi    @0,@1
        .endmacro
.macro P1     ; port,bit          ; port=1
        sbi    @0,@1
        .endmacro
.macro SP0    ; port,bit          ; skip if port=0
        sbic   @0,@1
        .endmacro
.macro SP1    ; port,bit          ; skip if port=1
        sbis   @0,@1
        .endmacro
.macro JP0    ; port,bit,addr           ; jump if port=0
        sbis   @0,@1
        rjmp   @2
        .endmacro
.macro JP1    ; port,bit,addr           ; jump if port=1
        sbic   @0,@1
        rjmp   @2
```

```
        .endmacro
.macro CP0    ; port,bit,addr            ; call if port=0
        sbis   @0,@1
        rcall  @2
        .endmacro
.macro CP1    ; port,bit,addr            ; call if port=1
        sbic   @0,@1
        rcall  @2
        .endmacro
.macro WP0    ; port,bit          ; wait if port=0
        sbis   @0,@1
        rjmp   PC-1
        .endmacro
.macro WP1    ; port,bit          ; wait if port=1
        sbic   @0,@1
        rjmp   PC-1
        .endmacro
.macro RP0    ; port,bit          ; return if port=0
        sbis   @0,@1
        ret
        .endmacro
.macro RP1    ; port,bit          ; return if port=1
        sbic   @0,@1
        ret
        .endmacro

; wait if port=0 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro WP0T   ; port,bit,timeout,addr
        ldi    w,@2+1
        dec    w      ; 1 cyc
        breq   @3     ; 1 cyc
        sbis   @0,@1  ; 1 cyc
        rjmp   PC-3   ; 2 cyc = 5 cycles
        .endmacro

; wait if port=1 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro WP1T   ; port,bit,timeout,addr
        ldi    w,@2+1
        dec    w      ; 1 cyc
        breq   @3     ; 1 cyc
        sbic   @0,@1  ; 1 cyc
        rjmp   PC-3   ; 2 cyc = 5 cycles
        .endmacro

; ============================
;      multi-byte operations
; ============================

.macro SWAP4                ; swap 2 variables
        mov    w ,@0
        mov    @0,@4
        mov    @4,w
        mov    w ,@1
        mov    @1,@5
        mov    @5,w
        mov    w ,@2
        mov    @2,@6
        mov    @6,w
        mov    w ,@3
        mov    @3,@7
```

```
        mov     @7,w
        .endmacro
.macro SWAP3
        mov     w ,@0
        mov     @0,@3
        mov     @3,w
        mov     w ,@1
        mov     @1,@4
        mov     @4,w
        mov     w ,@2
        mov     @2,@5
        mov     @5,w
        .endmacro
.macro SWAP2
        mov     w ,@0
        mov     @0,@2
        mov     @2,w
        mov     w ,@1
        mov     @1,@3
        mov     @3,w
        .endmacro
.macro SWAP1
        mov     w ,@0
        mov     @0,@1
        mov     @1,w
        .endmacro

.macro LDX4   ;r..r0        ; load from (x+)
        ld      @3,x+
        ld      @2,x+
        ld      @1,x+
        ld      @0,x+
        .endmacro
.macro LDX3   ;r..r0
        ld      @2,x+
        ld      @1,x+
        ld      @0,x+
        .endmacro
.macro LDX2   ;r..r0
        ld      @1,x+
        ld      @0,x+
        .endmacro

.macro LDY4   ;r..r0        ; load from (y+)
        ld      @3,y+
        ld      @2,y+
        ld      @1,y+
        ld      @0,y+
        .endmacro
.macro LDY3   ;r..r0
        ld      @2,y+
        ld      @1,y+
        ld      @0,y+
        .endmacro
.macro LDY2   ;r..r0
        ld      @1,y+
        ld      @0,y+
        .endmacro

.macro LDZ4   ;r..r0        ; load from (z+)
        ld      @3,z+
        ld      @2,z+
```

```
        ld      @1,z+
        ld      @0,z+
        .endmacro
.macro LDZ3   ;r..r0
        ld      @2,z+
        ld      @1,z+
        ld      @0,z+
        .endmacro
.macro LDZ2   ;r..r0
        ld      @1,z+
        ld      @0,z+
        .endmacro

.macro STX4   ;r..r0        ; store to (x+)
        st      x+,@3
        st      x+,@2
        st      x+,@1
        st      x+,@0
        .endmacro
.macro STX3   ;r..r0
        st      x+,@2
        st      x+,@1
        st      x+,@0
        .endmacro
.macro STX2   ;r..r0
        st      x+,@1
        st      x+,@0
        .endmacro

.macro STY4   ;r..r0        ; store to (y+)
        st      y+,@3
        st      y+,@2
        st      y+,@1
        st      y+,@0
        .endmacro
.macro STY3   ;r..r0
        st      y+,@2
        st      y+,@1
        st      y+,@0
        .endmacro
.macro STY2   ;r..r0
        st      y+,@1
        st      y+,@0
        .endmacro

.macro STZ4   ;r..r0        ; store to (z+)
        st      z+,@3
        st      z+,@2
        st      z+,@1
        st      z+,@0
        .endmacro
.macro STZ3   ;r..r0
        st      z+,@2
        st      z+,@1
        st      z+,@0
        .endmacro
.macro STZ2   ;r..r0
        st      z+,@1
        st      z+,@0
        .endmacro

.macro STI4   ;addr,k             ; store immediate
```

```
        ldi     w,  low(@1)
        sts     @0+0,w
        ldi     w, high(@1)
        sts     @0+1,w
        ldi     w,byte3(@1)
        sts     @0+2,w
        ldi     w,byte4(@1)
        sts     @0+3,w
        .endmacro
.macro STI3    ;addr,k
        ldi     w,  low(@1)
        sts     @0+0,w
        ldi     w, high(@1)
        sts     @0+1,w
        ldi     w,byte3(@1)
        sts     @0+2,w
        .endmacro
.macro STI2    ;addr,k
        ldi     w,  low(@1)
        sts     @0+0,w
        ldi     w, high(@1)
        sts     @0+1,w
        .endmacro
.macro STI     ;addr,k
        ldi     w,@1
        sts     @0,w
        .endmacro

.macro INC4                     ; increment
        ldi     w,0xff
        sub     @3,w
        sbc     @2,w
        sbc     @1,w
        sbc     @0,w
        .endmacro
.macro INC3
        ldi     w,0xff
        sub     @2,w
        sbc     @1,w
        sbc     @0,w
        .endmacro
.macro INC2
        ldi     w,0xff
        sub     @1,w
        sbc     @0,w
        .endmacro

.macro DEC4                     ; decrement
        ldi     w,0xff
        add     @3,w
        adc     @2,w
        adc     @1,w
        adc     @0,w
        .endmacro
.macro DEC3
        ldi     w,0xff
        add     @2,w
        adc     @1,w
        adc     @0,w
        .endmacro
.macro DEC2
        ldi     w,0xff
```

```
        add     @1,w
        adc     @0,w
        .endmacro

.macro CLR9                 ; clear (also clears the carry)
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        clr     @4
        clr     @5
        clr     @6
        clr     @7
        clr     @8
        .endmacro
.macro CLR8
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        clr     @4
        clr     @5
        clr     @6
        clr     @7
        .endmacro
.macro CLR7
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        clr     @4
        clr     @5
        clr     @6
        .endmacro
.macro CLR6
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        clr     @4
        clr     @5
        .endmacro
.macro CLR5
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        clr     @4
        .endmacro
.macro CLR4
        sub     @0,@0
        clr     @1
        clr     @2
        clr     @3
        .endmacro
.macro CLR3
        sub     @0,@0
        clr     @1
        clr     @2
        .endmacro
.macro CLR2
        sub     @0,@0
```

```
        clr     @1
        .endmacro

.macro COM4                 ; one's complement
        com     @0
        com     @1
        com     @2
        com     @3
        .endmacro
.macro COM3
        com     @0
        com     @1
        com     @2
        .endmacro
.macro COM2
        com     @0
        com     @1
        .endmacro

.macro NEG4                 ; negation (two's complement)
        com     @0
        com     @1
        com     @2
        com     @3
        ldi     w,0xff
        sub     @3,w
        sbc     @2,w
        sbc     @1,w
        sbc     @0,w
        .endmacro
.macro NEG3
        com     @0
        com     @1
        com     @2
        ldi     w,0xff
        sub     @2,w
        sbc     @1,w
        sbc     @0,w
        .endmacro
.macro NEG2
        com     @0
        com     @1
        ldi     w,0xff
        sub     @1,w
        sbc     @0,w
        .endmacro

.macro LDI4   ; r..r0, k    ; load immediate
        ldi     @3,  low(@4)
        ldi     @2, high(@4)
        ldi     @1,byte3(@4)
        ldi     @0,byte4(@4)
        .endmacro
.macro LDI3
        ldi     @2,  low(@3)
        ldi     @1, high(@3)
        ldi     @0,byte3(@3)
        .endmacro
.macro LDI2
        ldi     @1,  low(@2)
        ldi     @0, high(@2)
        .endmacro
```

```
.macro LDS4                ; load direct from SRAM
       lds    @3,@4
       lds    @2,@4+1
       lds    @1,@4+2
       lds    @0,@4+3
       .endmacro
.macro LDS3
       lds    @2,@3
       lds    @1,@3+1
       lds    @0,@3+2
       .endmacro
.macro LDS2
       lds    @1,@2
       lds    @0,@2+1
       .endmacro

.macro STS4                ; store direct to SRAM
       sts    @0+0,@4
       sts    @0+1,@3
       sts    @0+2,@2
       sts    @0+3,@1
       .endmacro
.macro STS3
       sts    @0+0,@3
       sts    @0+1,@2
       sts    @0+2,@1
       .endmacro
.macro STS2
       sts    @0+0,@2
       sts    @0+1,@1
       .endmacro

.macro STDZ4  ; d, r3,r2,r1,r0
       std    z+@0+0,@4
       std    z+@0+1,@3
       std    z+@0+2,@2
       std    z+@0+3,@1
       .endmacro
.macro STDZ3  ; d, r2,r1,r0
       std    z+@0+0,@3
       std    z+@0+1,@2
       std    z+@0+2,@1
       .endmacro
.macro STDZ2  ; d, r1,r0
       std    z+@0+0,@2
       std    z+@0+1,@1
       .endmacro

.macro LPM4                ; load program memory
       lpm
       mov    @3,r0
       adiw   zl,1
       lpm
       mov    @2,r0
       adiw   zl,1
       lpm
       mov    @1,r0
       adiw   zl,1
       lpm
       mov    @0,r0
       adiw   zl,1
```

```
        .endmacro
.macro LPM3
        lpm
        mov     @2,r0
        adiw    zl,1
        lpm
        mov     @1,r0
        adiw    zl,1
        lpm
        mov     @0,r0
        adiw    zl,1
        .endmacro
.macro LPM2
        lpm
        mov     @1,r0
        adiw    zl,1
        lpm
        mov     @0,r0
        adiw    zl,1
        .endmacro
.macro LPM1
        lpm
        mov     @0,r0
        adiw    zl,1
        .endmacro

.macro MOV4                     ; move between registers
        mov     @3,@7
        mov     @2,@6
        mov     @1,@5
        mov     @0,@4
        .endmacro
.macro MOV3
        mov     @2,@5
        mov     @1,@4
        mov     @0,@3
        .endmacro
.macro MOV2
        mov     @1,@3
        mov     @0,@2
        .endmacro

.macro ADD4                     ; add
        add     @3,@7
        adc     @2,@6
        adc     @1,@5
        adc     @0,@4
        .endmacro
.macro ADD3
        add     @2,@5
        adc     @1,@4
        adc     @0,@3
        .endmacro
.macro ADD2
        add     @1,@3
        adc     @0,@2
        .endmacro

.macro SUB4                     ; subtract
        sub     @3,@7
        sbc     @2,@6
        sbc     @1,@5
```

```
        sbc     @0,@4
        .endmacro
.macro SUB3
        sub     @2,@5
        sbc     @1,@4
        sbc     @0,@3
        .endmacro
.macro SUB2
        sub     @1,@3
        sbc     @0,@2
        .endmacro

.macro CP4                      ; compare
        cp      @3,@7
        cpc     @2,@6
        cpc     @1,@5
        cpc     @0,@4
        .endmacro
.macro CP3
        cp      @2,@5
        cpc     @1,@4
        cpc     @0,@3
        .endmacro
.macro CP2
        cp      @1,@3
        cpc     @0,@2
        .endmacro

.macro TST4                     ; test
        clr     w
        cp      @3,w
        cpc     @2,w
        cpc     @1,w
        cpc     @0,w
        .endmacro
.macro TST3
        clr     w
        cp      @2,w
        cpc     @1,w
        cpc     @0,w
        .endmacro
.macro TST2
        clr     w
        cp      @1,w
        cpc     @0,w
        .endmacro

.macro ADDI4                    ; add immediate
        subi    @3,  low(-@4)
        sbci    @2, high(-@4)
        sbci    @1,byte3(-@4)
        sbci    @0,byte4(-@4)
        .endmacro
.macro ADDI3
        subi    @2,  low(-@3)
        sbci    @1, high(-@3)
        sbci    @0,byte3(-@3)
        .endmacro
.macro ADDI2
        subi    @1,  low(-@2)
        sbci    @0, high(-@2)
        .endmacro
```

```
.macro SUBI4                ; subtract immediate
       subi   @3,  low(@4)
       sbci   @2, high(@4)
       sbci   @1,byte3(@4)
       sbci   @0,byte4(@4)
       .endmacro
.macro SUBI3
       subi   @2,  low(@3)
       sbci   @1, high(@3)
       sbci   @0,byte3(@3)
       .endmacro
.macro SUBI2
       subi   @1,  low(@2)
       sbci   @0, high(@2)
       .endmacro

.macro LSL5                 ; logical shift left
       lsl    @4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro LSL4
       lsl    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro LSL3
       lsl    @2
       rol    @1
       rol    @0
       .endmacro
.macro LSL2
       lsl    @1
       rol    @0
       .endmacro

.macro LSR4                 ; logical shift right
       lsr    @0
       ror    @1
       ror    @2
       ror    @3
       .endmacro
.macro LSR3
       lsr    @0
       ror    @1
       ror    @2
       .endmacro
.macro LSR2
       lsr    @0
       ror    @1
       .endmacro

.macro ASR4                 ; arithmetic shift right
       asr    @0
       ror    @1
       ror    @2
       ror    @3
       .endmacro
```

```
.macro ASR3
       asr    @0
       ror    @1
       ror    @2
       .endmacro
.macro ASR2
       asr    @0
       ror    @1
       .endmacro

.macro ROL8                ; rotate left through carry
       rol    @7
       rol    @6
       rol    @5
       rol    @4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL7
       rol    @6
       rol    @5
       rol    @4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL6
       rol    @5
       rol    @4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL5
       rol    @4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL4
       rol    @3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL3
       rol    @2
       rol    @1
       rol    @0
       .endmacro
.macro ROL2
       rol    @1
       rol    @0
       .endmacro

.macro ROR8                ; rotate right through carry
       ror    @0
```

```
        ror     @1
        ror     @2
        ror     @3
        ror     @4
        ror     @5
        ror     @6
        ror     @7
        .endmacro
.macro ROR7
        ror     @0
        ror     @1
        ror     @2
        ror     @3
        ror     @4
        ror     @5
        ror     @6
        .endmacro
.macro ROR6
        ror     @0
        ror     @1
        ror     @2
        ror     @3
        ror     @4
        ror     @5
        .endmacro
.macro ROR5
        ror     @0
        ror     @1
        ror     @2
        ror     @3
        ror     @4
        .endmacro
.macro ROR4
        ror     @0
        ror     @1
        ror     @2
        ror     @3
        .endmacro
.macro ROR3
        ror     @0
        ror     @1
        ror     @2
        .endmacro
.macro ROR2
        ror     @0
        ror     @1
        .endmacro

.macro PUSH2
        push    @0
        push    @1
        .endmacro
.macro POP2
        pop     @1
        pop     @0
        .endmacro

.macro PUSH3
        push    @0
        push    @1
        push    @2
        .endmacro
```

```
.macro POP3
	pop	@2
	pop	@1
	pop	@0
	.endmacro

.macro PUSH4
	push	@0
	push	@1
	push	@2
	push	@3
	.endmacro
.macro POP4
	pop	@3
	pop	@2
	pop	@1
	pop	@0
	.endmacro

.macro PUSH5
	push	@0
	push	@1
	push	@2
	push	@3
	push	@4
	.endmacro
.macro POP5
	pop	@4
	pop	@3
	pop	@2
	pop	@1
	pop	@0
	.endmacro

; --- SRAM operations ---
.macro INCS4	; sram		; increment SRAM 4-byte variable
	lds	w,@0
	inc	w
	sts	@0,w
	brne	end
	lds	w,@0+1
	inc	w
	sts	@0+1,w
	brne	end
	lds	w,@0+2
	inc	w
	sts	@0+2,w
	brne	end
	lds	w,@0+3
	inc	w
	sts	@0+3,w
end:
	.endmacro

.macro INCS3	; sram		; increment SRAM 3-byte variable
	lds	w,@0
	inc	w
	sts	@0,w
	brne	end
	lds	w,@0+1
	inc	w
	sts	@0+1,w
```

```
        brne    end
        lds     w,@0+2
        inc     w
        sts     @0+2,w
end:
        .endmacro

.macro INCS2  ; sram        ; increment SRAM 2-byte variable
        lds     w,@0
        inc     w
        sts     @0,w
        brne    end
        lds     w,@0+1
        inc     w
        sts     @0+1,w
end:
        .endmacro

.macro INCS   ; sram        ; increment SRAM 1-byte variable
        lds     w,@0
        inc     w
        sts     @0,w
        .endmacro

.macro DECS4  ; sram        ; decrement SRAM 4-byte variable
        ldi     w,1
        lds     u,@0
        sub     u,w
        sts     @0,u
        clr     w
        lds     u,@0+1
        sbc     u,w
        sts     @0+1,u
        lds     u,@0+2
        sbc     u,w
        sts     @0+2,u
        lds     u,@0+3
        sbc     u,w
        sts     @0+3,u
        .endmacro
.macro DECS3  ; sram        ; decrement SRAM 3-byte variable
        ldi     w,1
        lds     u,@0
        sub     u,w
        sts     @0,u
        clr     w
        lds     u,@0+1
        sbc     u,w
        sts     @0+1,u
        lds     u,@0+2
        sbc     u,w
        sts     @0+2,u
        .endmacro
.macro DECS2  ; sram        ; decrement SRAM 2-byte variable
        ldi     w,1
        lds     u,@0
        sub     u,w
        sts     @0,u
        clr     w
        lds     u,@0+1
        sbc     u,w
        sts     @0+1,u
```

```
        .endmacro
.macro DECS    ; sram        ; decrement
        lds     w,@0
        dec     w
        sts     @0,w
        .endmacro

.macro MOVS4   ; addr0,addr1 ; [addr0] <-- [addr1]
        lds     w,@1
        sts     @0,w
        lds     w,@1+1
        sts     @0+1,w
        lds     w,@1+2
        sts     @0+2,w
        lds     w,@3+1
        sts     @0+3,w
        .endmacro
.macro MOVS3   ; addr0,addr1 ; [addr0] <-- [addr1]
        lds     w,@1
        sts     @0,w
        lds     w,@1+1
        sts     @0+1,w
        lds     w,@1+2
        sts     @0+2,w
        .endmacro
.macro MOVS2   ; addr0,addr1 ; [addr0] <-- [addr1]
        lds     w,@1
        sts     @0,w
        lds     w,@1+1
        sts     @0+1,w
        .endmacro
.macro MOVS    ; addr0,addr1 ; [addr0] <-- [addr1]
        lds     w,@1
        sts     @0,w
        .endmacro

.macro SEXT    ; reg1,reg0   ; sign extend
        clr     @0
        sbrc    @1,7
        dec     @0
        .endmacro

; ====================================
;       Jump/Call with constant arguments
; ====================================

; --- calls with arguments a,b,XYZ ---
.macro CX      ; subroutine,x
        ldi     xl, low(@1)
        ldi     xh,high(@1)
        rcall   @0
        .endmacro
.macro CXY     ; subroutine,x,y
        ldi     xl, low(@1)
        ldi     xh,high(@1)
        ldi     yl, low(@2)
        ldi     yh,high(@2)
        rcall   @0
        .endmacro
.macro CXZ     ; subroutine,x,z
        ldi     xl, low(@1)
        ldi     xh,high(@1)
```

```
        ldi     zl, low(@2)
        ldi     zh,high(@2)
        rcall   @0
        .endmacro
.macro CXYZ     ; subroutine,x,y,z
        ldi     xl, low(@1)
        ldi     xh,high(@1)
        ldi     yl, low(@2)
        ldi     yh,high(@2)
        ldi     zl, low(@3)
        ldi     zh,high(@3)
        rcall   @0
        .endmacro
.macro CW       ; subroutine,w
        ldi     w, @1
        rcall   @0
        .endmacro
.macro CA       ; subroutine,a
        ldi     a0, @1
        rcall   @0
        .endmacro
.macro CAB      ; subroutine,a,b
        ldi     a0, @1
        ldi     b0, @2
        rcall   @0
        .endmacro

; --- jump with arguments w,a,b ---
.macro JW       ; subroutine,w
        ldi     w, @1
        rjmp    @0
        .endmacro
.macro JA       ; subroutine,a
        ldi     a0, @1
        rjmp    @0
        .endmacro
.macro JAB      ; subroutine,a,b
        ldi     a0, @1
        ldi     b0, @2
        rjmp    @0
        .endmacro
.list
```

lcd.asm

```
; 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
```

printf.asm

```
; file printf.asm   target ATmega128L-4MHz-STK300
; purpose library, formatted output generation
; author (c) R.Holzer (adapted MICRO210/EE208 A.Schmid)
; v2019.02 20180821 AxS 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,01-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
; author (c) Raphael Holzer
;
; 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)
; author (c) Raphael Holzer, EPFL
;
; 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
```

wire1.asm

```
; 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
```

math.asm

```
; file math.asm   target ATmega128L-4MHz-STK300
; purpose library, mathematical routines
; copyright R.Holzer

; === unsigned multiplication (c=a*b) ===

mul11: clr    c1                 ; clear upper half of result c
       mov    c0,b0              ; place b in lower half of c
       lsr    c0                 ; shift LSB (of b) into carry
       ldi    w,8                ; load bit counter
_m11:  brcc   PC+2               ; skip addition if carry=0
       add    c1,a0              ; add a to upper half of c
       ROR2   c1,c0              ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m11             ; Decrement and Jump if bit-count Not Zero
       ret

mul21: CLR2   c2,c1              ; clear upper half of result c
       mov    c0,b0              ; place b in lower half of c
       lsr    c0                 ; shift LSB (of b) into carry
       ldi    w,8                ; load bit counter
_m21:  brcc   PC+3               ; skip addition if carry=0
       ADD2   c2,c1, a1,a0       ; add a to upper half of c
       ROR3   c2,c1,c0           ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m21             ; Decrement and Jump if bit-count Not Zero
       ret

mul22: CLR2   c3,c2              ; clear upper half of result c
       MOV2   c1,c0, b1,b0       ; place b in lower half of c
       LSR2   c1,c0              ; shift LSB (of b) into carry
       ldi    w,16               ; load bit counter
_m22:  brcc   PC+3               ; skip addition if carry=0
       ADD2   c3,c2, a1,a0       ; add a to upper half of c
       ROR4   c3,c2,c1,c0        ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m22             ; Decrement and Jump if bit-count Not Zero
       ret

mul31: CLR3   c3,c2,c1           ; clear upper half of result c
       mov    c0,b0              ; place b in lower half of c
       lsr    c0                 ; shift LSB (of b) into carry
       ldi    w,8                ; load bit counter
_m31:  brcc   PC+4               ; skip addition if carry=0
       ADD3   c3,c2,c1, a2,a1,a0  ; add a to upper half of c
       ROR4   c3,c2,c1,c0        ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m31             ; Decrement and Jump if bit-count Not Zero
       ret

mul32: CLR3   d0,c3,c2           ; clear upper half of result c
       MOV2   c1,c0, b1,b0       ; place b in lower half of c
       LSR2   c1,c0              ; shift LSB (of b) into carry
       ldi    w,16               ; load bit counter
_m32:  brcc   PC+4               ; skip addition if carry=0
       ADD3   d0,c3,c2, a2,a1,a0  ; add a to upper half of c
       ROR5   d0,c3,c2,c1,c0           ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m32             ; Decrement and Jump if bit-count Not Zero
       ret

mul33: CLR3   d1,d0,c3           ; clear upper half of result c
       MOV3   c2,c1,c0, b2,b1,b0  ; place b in lower half of c
       LSR3   c2,c1,c0           ; shift LSB (of b) into carry
       ldi    w,24               ; load bit counter
_m33:  brcc   PC+4               ; skip addition if carry=0
```

```
       ADD3   d1,d0,c3, a2,a1,a0  ; add a to upper half of c
       ROR6   d1,d0,c3,c2,c1,c0   ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m33              ; Decrement and Jump if bit-count Not Zero
       ret

mul41: CLR4   d0,c3,c2,c1         ; clear upper half of result c
       mov    c0,b0               ; place b in lower half of c
       lsr    c0                  ; shift LSB (of b) into carry
       ldi    w,8                 ; load bit counter
_m41:  brcc   PC+5                ; skip addition if carry=0
       ADD4   d0,c3,c2,c1, a3,a2,a1,a0; add a to upper half of c
       ROR5   d0,c3,c2,c1,c0             ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m41              ; Decrement and Jump if bit-count Not Zero
       ret

mul42: CLR4   d1,d0,c3,c2         ; clear upper half of result c
       MOV2   c1,c0, b1,b0              ; place b in lower half of c
       LSR2   c1,c0               ; shift LSB (of b) into carry
       ldi    w,16                ; load bit counter
_m42:  brcc   PC+5                ; skip addition if carry=0
       ADD4   d1,d0,c3,c2, a3,a2,a1,a0; add a to upper half of c
       ROR6   d1,d0,c3,c2,c1,c0   ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m42              ; Decrement and Jump if bit-count Not Zero
       ret

mul43: CLR4   d2,d1,d0,c3         ; clear upper half of result c
       MOV3   c2,c1,c0, b2,b1,b0  ; place b in lower half of c
       LSR3   c2,c1,c0            ; shift LSB (of b) into carry
       ldi    w,24                ; load bit counter
_m43:  brcc   PC+5                ; skip addition if carry=0
       ADD4   d2,d1,d0,c3, a3,a2,a1,a0; add a to upper half of c
       ROR7   d2,d1,d0,c3,c2,c1,c0; shift-right c, LSB (of b) into carry
       DJNZ   w,_m43              ; Decrement and Jump if bit-count Not Zero
       ret

mul44: CLR4   d3,d2,d1,d0         ; clear upper half of result c
       MOV4   c3,c2,c1,c0, b3,b2,b1,b0; place b in lower half of c
       LSR4   c3,c2,c1,c0         ; shift LSB (of b) into carry
       ldi    w,32                ; load bit counter
_m44:  brcc   PC+5                ; skip addition if carry=0
       ADD4   d3,d2,d1,d0, a3,a2,a1,a0; add a to upper half of c
       ROR8   d3,d2,d1,d0,c3,c2,c1,c0    ; shift-right c, LSB (of b) into carry
       DJNZ   w,_m44              ; Decrement and Jump if bit-count Not Zero
       ret

; === signed multiplication ===
mul11s: rcall mul11
       sbrc   a0,7
       sub    c1,b0
       sbrc   b0,7
       sub    c1,a0
       ret

mul22s: rcall mul22
       sbrs   a1,7
       rjmp   PC+3
       SUB2   c3,c2, b1,b0
       sbrs   b1,7
       rjmp   PC+3
       SUB2   c3,c2, a1,a0
       ret
```

```
mul33s: rcall mul33
        sbrs   a2,7
        rjmp   PC+4
        SUB3   d1,d0,c3, b2,b1,b0
        sbrs   b2,7
        rjmp   PC+4
        SUB3   d1,d0,c3, a2,a1,a0
        ret

mul44s: rcall mul44
        sbrs   a3,7
        rjmp   PC+5
        SUB4   d3,d2,d1,d0, b3,b2,b1,b0
        sbrs   b3,7
        rjmp   PC+5
        SUB4   d3,d2,d1,d0, a3,a2,a1,a0
        ret

; === unsigned division c=a/b ===
div11: mov    c0,a0               ; c will contain the result
       clr    d0                  ; d will contain the remainder
       ldi    w,8                 ; load bit counter
_d11:  ROL2   d0,c0               ; 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,_d11              ; Decrement and Jump if bit-count Not Zero
       rol    c0                  ; last shift (C into result c)
       com    c0                  ; complement result
       ret

div21: MOV2   c1,c0, a1,a0        ; c will contain the result
       clr    d0                  ; d will contain the remainder
       ldi    w,16                ; load bit counter
_d21:  ROL3   d0,c1,c0            ; 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,_d21              ; Decrement and Jump if bit-count Not Zero
       ROL2   c1,c0               ; last shift (carry into result c)
       COM2   c1,c0               ; complement result
       ret

div22: MOV2   c1,c0, a1,a0        ; c will contain the result
       CLR2   d1,d0               ; d will contain the remainder
       ldi    w,16                ; load bit counter
_d22:  ROL4   d1,d0,c1,c0         ; shift carry into result c
       SUB2   d1,d0, b1,b0        ; subtract b from remainder
       brcc   PC+3
       ADD2   d1,d0, b1,b0        ; restore if remainder became negative
       DJNZ   w,_d22              ; Decrement and Jump if bit-count Not Zero
       ROL2   c1,c0               ; last shift (carry into result c)
       COM2   c1,c0               ; complement result
       ret

div31: MOV3   c2,c1,c0, a2,a1,a0  ; c will contain the result
       clr    d0                  ; d will contain the remainder
       ldi    w,24                ; load bit counter
_d31:  ROL4   d0,c2,c1,c0         ; 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,_d31              ; Decrement and Jump if bit-count Not Zero
       ROL3   c2,c1,c0            ; last shift (carry into result c)
       COM3   c2,c1,c0            ; complement result
       ret

div32: MOV3   c2,c1,c0, a2,a1,a0  ; c will contain the result
       CLR2   d1,d0               ; d will contain the remainder
       ldi    w,24                ; load bit counter
_d32:  ROL5   d1,d0,c2,c1,c0            ; shift carry into result c
       SUB2   d1,d0, b1,b0        ; subtract b from remainder
       brcc   PC+3
       ADD2   d1,d0, b1,b0        ; restore if remainder became negative
       DJNZ   w,_d32              ; Decrement and Jump if bit-count Not Zero
       ROL3   c2,c1,c0            ; last shift (carry into result c)
       COM3   c2,c1,c0            ; complement result
       ret

div33: MOV3   c2,c1,c0, a2,a1,a0  ; c will contain the result
       CLR3   d2,d1,d0            ; d will contain the remainder
       ldi    w,24                ; load bit counter
_d33:  ROL6   d2,d1,d0,c2,c1,c0   ; shift carry into result c
       SUB3   d2,d1,d0, b2,b1,b0  ; subtract b from remainder
       brcc   PC+4
       ADD3   d2,d1,d0, b2,b1,b0  ; restore if remainder became negative
       DJNZ   w,_d33              ; Decrement and Jump if bit-count Not Zero
       ROL3   c2,c1,c0            ; last shift (carry into result c)
       COM3   c2,c1,c0            ; complement result
       ret

div41: MOV4   c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
       clr    d0                  ; d will contain the remainder
       ldi    w,32                ; load bit counter
_d41:  ROL5   d0,c3,c2,c1,c0            ; 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   c3,c2,c1,c0         ; last shift (carry into result c)
       COM4   c3,c2,c1,c0         ; complement result
       ret

div42: MOV4   c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
       CLR2   d1,d0               ; d will contain the remainder
       ldi    w,32                ; load bit counter
_d42:  ROL6   d1,d0,c3,c2,c1,c0   ; shift carry into result c
       SUB2   d1,d0, b1,b0        ; subtract b from remainder
       brcc   PC+3
       ADD2   d1,d0, b1,b0        ; restore if remainder became negative
       DJNZ   w,_d42              ; Decrement and Jump if bit-count Not Zero
       ROL4   c3,c2,c1,c0         ; last shift (carry into result c)
       COM4   c3,c2,c1,c0         ; complement result
       ret

div43: MOV4   c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
       CLR3   d2,d1,d0            ; d will contain the remainder
       ldi    w,32                ; load bit counter
_d43:  ROL7   d2,d1,d0,c3,c2,c1,c0; shift carry into result c
       SUB3   d2,d1,d0, b2,b1,b0  ; subtract b from remainder
       brcc   PC+4
       ADD3   d2,d1,d0, b2,b1,b0  ; restore if remainder became negative
       DJNZ   w,_d43              ; Decrement and Jump if bit-count Not Zero
       ROL4   c3,c2,c1,c0         ; last shift (carry into result c)
```

```
       COM4   c3,c2,c1,c0          ; complement result
       ret

div44: MOV4   c3,c2,c1,c0, a3,a2,a1,a0; c will contain the result
       CLR4   d3,d2,d1,d0          ; d will contain the remainder
       ldi    w,32                 ; load bit counter
_d44:  ROL8   d3,d2,d1,d0,c3,c2,c1,c0    ; shift carry into result c
       SUB4   d3,d2,d1,d0, b3,b2,b1,b0; subtract b from remainder
       brcc   PC+5
       ADD4   d3,d2,d1,d0, b3,b2,b1,b0; restore if remainder became negative
       DJNZ   w,_d44               ; Decrement and Jump if bit-count Not Zero
       ROL4   c3,c2,c1,c0          ; last shift (carry into result c)
       COM4   c3,c2,c1,c0          ; complement result
       ret

; === signed division ===
div33s:       push   u
       mov    u,a2
       eor    u,b2
       sbrs   a2,7
       rjmp   d33a
       NEG3   a2,a1,a0
d33a:  sbrs   b2,7
       rjmp   d33b
       NEG3   b2,b1,b0
d33b:  rcall  div33
       sbrs   u,7
       rjmp   d33c
       NEG3   c2,c1,c0
d33c:  pop    u
       ret
```

sound.asm

```
; 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
       tst    a0
       breq   sound_off     ; if a0=0 then no sound
sound1:
       mov    w,a0
       rcall  wait9us             ; 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  wait9us
       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
```