Date : 29 Mai, 2021
Groupe 72
Antoine SILVIN
Joshua COHEN-DUMANI

I. Introduction

a. Contexte

Dans le contexte du cours de Microcontrôleurs 2021, chaque groupe d'étudiants a dû développer un projet d'application de la carte STK-300, utilisant le langage assembleur. Ce rapport a pour but de décrire et présenter l'application développée par le groupe 72, composé d'Antoine Silvin (Sciper : 311390) et de Joshua Cohen-Dumani (Sciper : 311105).

b. Description générale

L'idée de base de ce projet est d'essayer de développer une version réduite de l'ordinateur de bord d'une voiture. Cet ordinateur simplifié a en effet trois fonctionnalités principales : un thermomètre, un radar de recul (faisant usage d'un capteur de distance et d'un buzzer), et un moyen de stockage de l'historique de la température en mémoire puis écriture sur un terminal. Les deux valeurs obtenues avec les capteurs sont présentées à l'utilisateur à l'aide de l'affichage LCD.

L'utilisateur peut interagir avec le microcontrôleur, à l'aide de boutons permettant d'enclencher ou non certaines fonctions, ou d'afficher des valeurs de la température sur le terminal d'un ordinateur connecté.

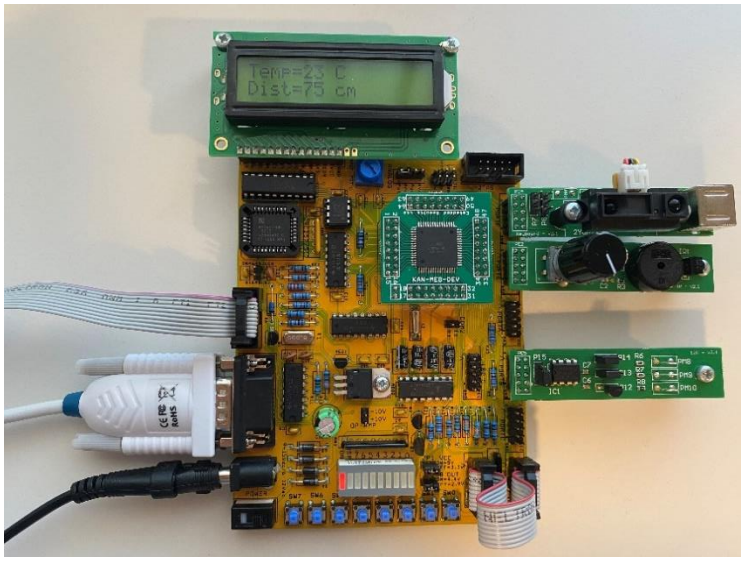

Figure 1: La carte STK-300 avec périphériques intégrés

Nous avons conscience que si ce système venait à être implémenté, d'autres versions ou modèles des périphériques seraient utilisés. Par exemple un capteur de distance avec une plus grande plage admissible, ou un affichage intégré dans l'interface utilisateur de la voiture. Cependant nous avons estimé que ceci n'était pas un problème dans le cadre de ce projet, et nous nous sommes basés purement sur la fonction du périphérique.

II. Mode d'emploi

Radar de recul

La fonction de radar de recul est effectuée à l'aide de trois périphériques : le capteur de distance, le buzzer, et un affichage LCD. Le capteur de distance est placé sur le port F et le buzzer sur le port E. Le capteur de distance, lorsqu'il est allumé, envoie en tout temps la distance qu'il capte. L'affichage sur le LCD est le suivant : Dans la plage du capteur (entre 10 et 80cm) le LCD affiche la distance à l'unité près en [cm]. Lorsque la distance est inférieure à 10cm, le LCD n'est plus fiable car la tension analogique envoyée par le capteur de distance correspond à des valeurs qui se trouvent ailleurs dans la LUT. Lorsque la distance est supérieure à 80cm, le LCD affiche 1000 indiquant que on est loin et en dehors de la plage de mesure.

Le buzzer est par défaut éteint, et peut être allumé à l'aide du bouton 0 (noté SW0 sur le hardware). La fréquence émise par celui-ci sera inversement proportionnelle à la distance, devenant de plus en plus aigu plus l'objet se rapproche de notre microcontrôleur. Le capteur de distance peut être complètement éteint à l'aide du bouton 1 (SW1). Il est recommandé d'éteindre le buzzer lorsque le capteur de distance est éteint, car il jouera simplement la dernière note qu'il avait en mémoire. Cependant nous avons décidé de le garder opérationnel pour rester proche d'une voiture réelle qui émet un son dès que la voiture est très proche d'un obstacle, même si le moteur est éteint (si son ordinateur de bord est allumé bien entendu).

Thermomètre

Le capteur de température, placé sur le port B, envoie en tout temps la valeur actuelle de la température à l'AVR, qui l'affiche à l'aide d'un LCD. Il est possible d'éteindre le capteur de température à l'aide du bouton 2 (SW2). La lecture de température possède 4 chiffres significatifs. Lorsque ce module est éteint, l'affichage est figé. Il recommence à s'actualiser après réactivation.

Envoi de données au terminal

Il est également possible de visualiser les valeurs de la température enregistrées dans la mémoire EEPROM (valeurs enregistrées toutes les secondes depuis l'allumage du microcontrôleur) en appuyant le bouton 3 (SW3). Ces valeurs sont envoyées avec l'UART vers un terminal et peut être visualisé en utilisant un logiciel comme PuTTy ou RealTerm. Les paramètres sont :

Speed (Baud)	9600
Data Bits	8
Stop Bits	1
Parity	None

Flow control	None
Local Echo (Terminal)	Forced ON

III. Description technique

a. Périphériques utilisés,

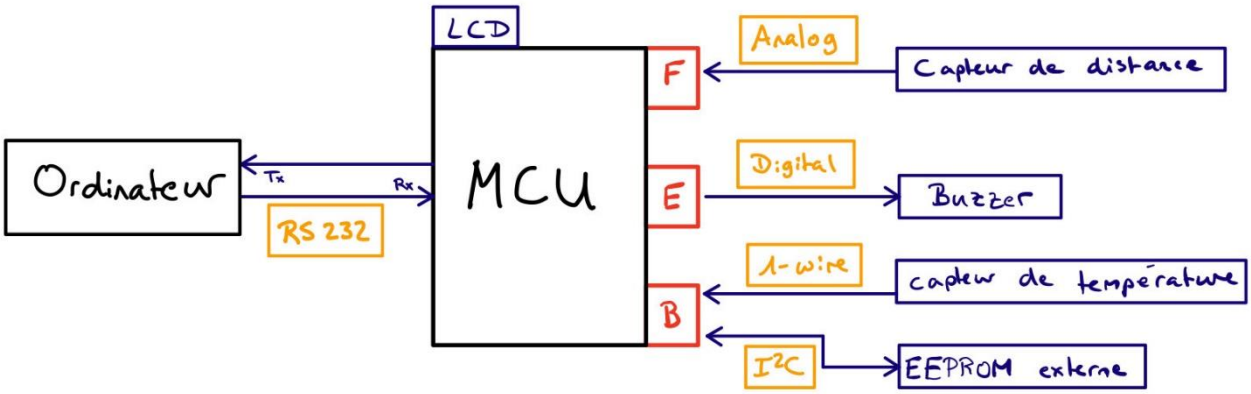

Figure 2: Schéma externe de notre application

Capteur de Température

Nous utilisons tout d'abord le capteur de température fourni, qui utilise le protocole 1-wire. La valeur obtenue par celui-ci est ensuite utilisée par deux autres modules : la mémoire externe EEPROM, ainsi que l'affichage LCD.

L'information est tout d'abord obtenue à l'aide du 1-wire , puis convertie en une donnée que l'on peut ensuite interpréter. La valeur est définie sur 12 bits et cette conversion dure 750 ms. Les valeurs (LSB et MSB) sont copiées des registres ***a*** en mémoire SRAM interne aux adresses $104 et $106 sous les labels *temperature0* et *temperature1* pour être réutilisés plus tard, pour l'affichage par exemple. La valeur de la température en binaire est ensuite séparés en deux chiffres : chiffres des dizaines stockée en mémoire données à l'adresse $114 sous le label *memory_data_1* et le chiffre des unités à l'adresse $116 sous le label *memory_data_2*. Pour séparer en deux chiffres on procède de la façon suivante : On cherche d'abord le chiffre des dizaines en comparant la valeur avec 10 puis 20 puis 30 et ainsi de suite… Ensuite, on multiplie la valeur des dizaines par dix et on la soustrait pour obtenir le chiffre des unités. L'affichage sur le LCD se fait un peu plus tard, on conserve 2 chiffres après la virgule pour le LCD grâce à un stockage temporaire en SRAM aux adresses *temperature0* et *temperature1* (comme mentionné ci-dessus).

Buzzer

Le buzzer fonctionne à l'aide du timer/compteur 2 (TCR2). Nous utilisons ce timer, ainsi que le registre OCR2, afin de générer un signal carré à une fréquence audible, proportionnelle à la distance obtenue. Le timer est utilisé en mode *output compare* , ainsi nous pouvons régler la fréquence en changeant la valeur dans OCR2. Le prescaler est réglé de sorte à avoir une fréquence audible. La valeur de la distance en [cm] est mise dans OCR2 donc la période d'interruption est proportionnelle à la distance. A chaque interruption, on inverse la tension sur le Buzzer (0 ou 5V). Ainsi, plus on est proche, plus le signal est aigu.

Capteur de distance

Nous utilisons le capteur de distance Sharp GP2Y0A21, qui est un capteur analogique. Nous avons donc dû convertir la valeur enregistrée, qui correspond à des volts, à une distance en centimètre. Nous avons donc créé un court programme python qui va (par interpolation de la courbe donnée dans la Datasheet) créer un graphique reliant ces deux valeurs. À partir de ce graphique nous pouvons extraire une liste de points que nous pouvons mettre dans une look-up table dans notre code principal. Ceci nous permet d'avoir une valeur de tension pour chaque centimètre dans la plage admissible du capteur. Ce programme est fourni en annexe.

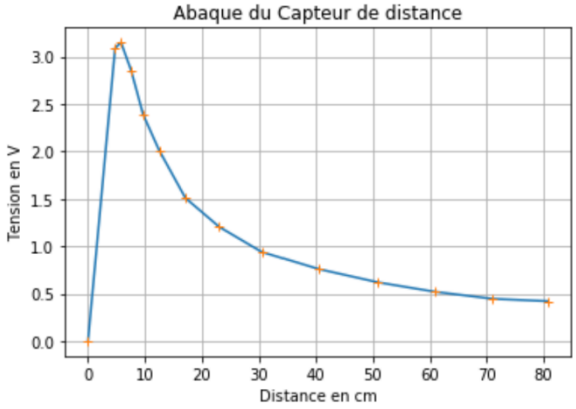

Figure 3: Graphique en sortie de interpolation.py

Nous avons placé le capteur de distance sur le port F, celui-ci étant connecté à un multiplexer le reliant au CAN qui va convertir la tension obtenue par le capteur en valeur numérique que l'on peut ensuite interpréter à l'aide de la look-up table.

La look-up table a la forme suivante :

130 , 80, 131, 79 , … , 645 , 12, 686, 11, 728, 10 , 1024, 0

Avec en noir la valeur de la tension entre 0 et 3.3V échelonnée de 0 à 1024 et en bleu la distance en [cm] associée. Le programme parcourt la table et compare la valeur reçue du CAN aux valeurs en noir. Il continue tant que la valeur du CAN est plus grande que la valeur en noir. Une fois arrivé, il ne reste plus qu'à afficher la valeur en bleu suivante qui est la distance correspondante.

Affichage LCD

L'affichage sur le LCD s'effectue dans la sous-routine « distance ». On check tout d'abord si le module de distance est allumé grace à la valeur stockée en mémoire données à l'adresse $100 sous

le label *capt_dist_enable*. Si non, on passe à l'affichage directement en affichant les valeurs obtenues par le capteur de température. Nous affichons les 2 bytes de *temperature0* et *temperature1* stockée en SRAM, dans lesquels est stocké la valeur à 4 chiffres significatifs de la température. Si le capteur de distance est *enabled*, on passe à la sous-routine « dist », qui va convertir la valeur obtenue par l'ADC du port F en [cm] en parcourant la look-up table. La valeur en [cm] de la distance et la valeur de la temperature en [°C] sont affichées grâce aux macros disponibles dans la librairie printf.asm.

Mémoire externe (EEPROM)

Chaque seconde, la valeur de la température est enregistrée, à 2 chiffres significatifs près, dans la mémoire externe EEPROM. Comme expliqué ci-dessus, la valeur entière de la température courante est stockée aux adresses *memory_data_1* et *memory_data_2* par la sous-routine « temp ». Ainsi, quand la sous routine « write_memory_i2c_with_data » est appelée elle récupère ces valeurs pour les stocker en mémoire EEPROM externe par liaison I2C.

On initialise la communication pas I2C puis on écrit l'adresse à laquelle on veut écrire en mémoire EEPROM (grâce à « i2c_write »). Cette adresse est stockée MSB à l'adresse $110 sous le label *memory_adress_MSB* et LSB à l'adresse $112 sous le label *memory_adress_LSB.* Ensuite, on récupère la valeur dans *memory_data_1* et on l'écrit. Nous appliquons la même procédure pour la valeur dans *memory_data_2*. Après ceci, on écrit deux fois le caractère '' ~ '' que l'on a choisi arbitrairement comme caractère *end of file* puis on cesse la communication I2C. L'adresse d'écriture, soit la valeur dans *memory_adress_MSB* et *memory_adress_LSB,* est ensuite incrémenté deux fois pour que la prochaine écriture vienne remplacer les caractères '' ~ '' qu'elle réécrira plus loin.

Lorsque le bouton SW3 est appuyé, la sous-routine « write_terminal » est appelée. Celle-ci extrait d'abord la première valeur contenue dans l'EEPROM à l'aide de la sous-routine « i2c_read », puis l'envoie par le protocole RS232 sur le terminal. L'adresse de lecture dans la mémoire EEPROM est auto post-incrémenté. On répète ce protocole jusqu'à ce que l'on atteigne le caractère '' ~ '' après quoi on cesse la communication.

Durant l'écriture en mémoire EEPROM, les interruptions sont désactivées pour éviter d'appeler l'interruption écriture alors que le programme est en train d'écrire la mémoire et n'a pas encore réécrit les caractère *end_of_file.*

b. Interruptions

Les interruptions dans notre programme se font sur les flancs descendants. Nous utilisons les boutons 0 à 3, donc INT0-INT3, qui sont asynchrone. Les interruptions sont possibles à n'importe quel moment pendant l'utilisation du microcontrôleur, sauf lorsque le programme est en train d'écrire la valeur de la température dans l'EEPROM.

Les boutons 0, 1, et 2 fonctionnent de manière analogue. Le bouton 0 peut enable ou disable le buzzer, le bouton 1 le l'affichage et la conversion de la distance, le bouton 2 la mesure de la température. Ceci se fait à l'aide de booléens : *dist_sound*, *capt_dist_enable*, et *temp_measure_enable* pour les boutons 0, 1 et 2 respectivement. Ces booléens sont stockés en mémoire donnée.

IV. Fonctionnement du programme

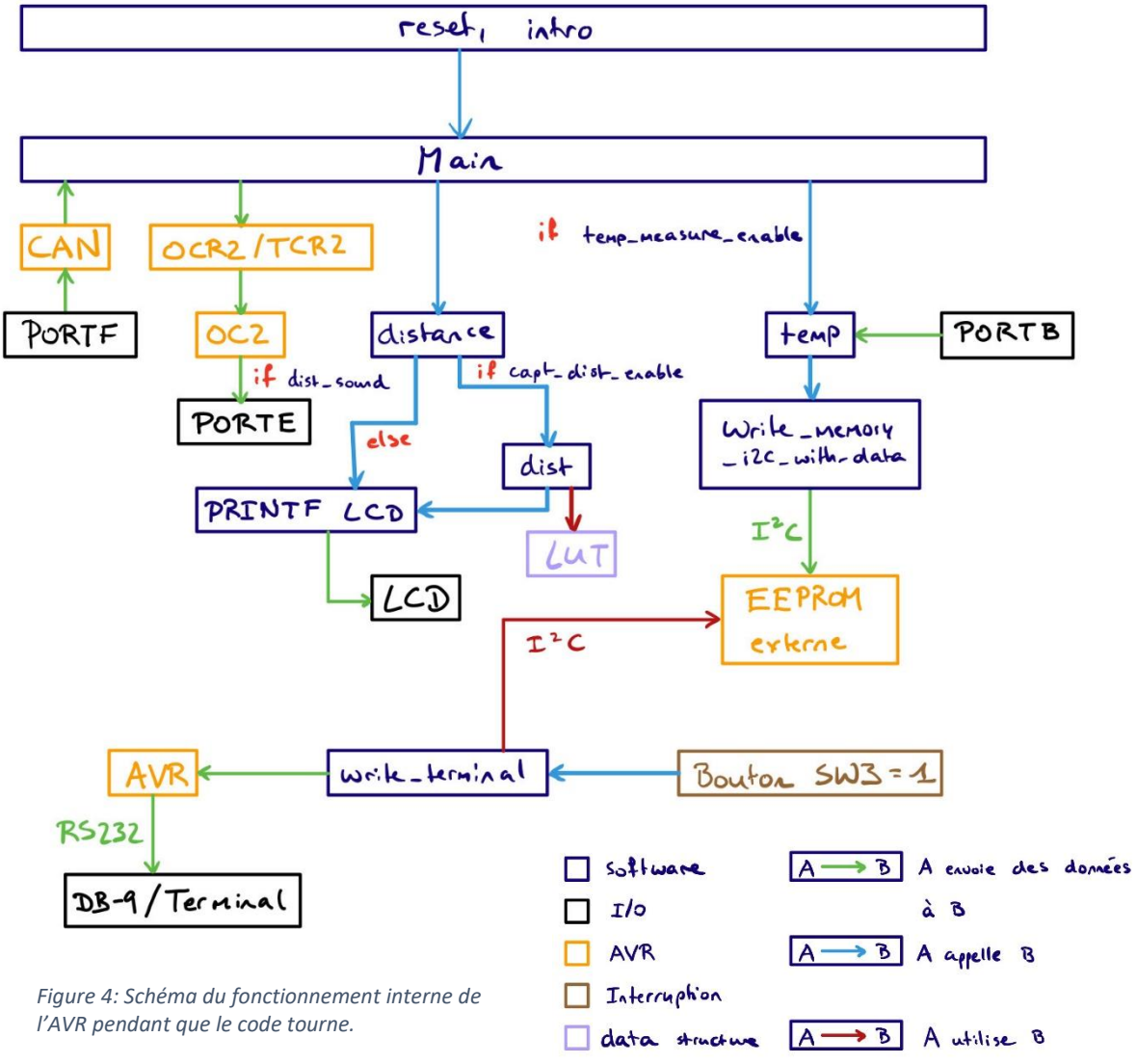

Figure 4: Schéma du fonctionnement interne de l'AVR pendant que le code tourne.

Fonctionnement du code

Lorsqu'on allume l'AVR, le programme appelle par défaut la fonction « *reset* », qui set les paramètres de fonctionnement du microcontrôleur. Reset appelle « *intro* », qui va preset la valeur des différentes variables utilisées, et autorise notamment les interruptions. Ensuite nous passons dans « *main* ».

Main commence par obtenir les valeurs de mesure du capteur de distance à travers l'ADC (CAN), et les enregistre dans des registres de passage **a0** et **a1**. Il envoie ensuite une valeur à OCR2 qui va set la fréquence du buzzer (valeur preset, initialement le port E est désactivé). Nous passons ensuite à la sous-routine distance, dont le fonctionnement est détaillé dans la section de l'affichage LCD, qui va afficher la température et, si le module est activé, convertir la valeur analogique dans les registres **a0** et **a1** en distance en [cm] à l'aide d'une LUT. On set par ailleurs la valeur qui va être envoyée à OCR2 à ce stade (après conversion). Ensuite, si le module est activé, « *main* » va appeler « *temp* », qui va en utilisant le protocole 1-wire obtenir et convertir la valeur de température. Cette température est ensuite stockée en mémoire externe EEPROM en utilisant le protocole i2c dans « *write_memory_i2c_with_data* ». Nous revenons ensuite à « *main* » qui « *loop* » à nouveau. Si le bouton SW3 est appuyé, « *write_terminal* » est appelé, qui va extraire les valeurs stockées dans l'EEPROM externe à l'aide du protocole i2c, puis envoyer ces données au terminal en utilisant le protocole RS232 (l'UART).

V. Annexes

Les annexes sont des PDFs concaténés avec l'ordre suivant :

a. Code source

1. Main.asm
2. definitions.asm
3. i2cx.asm
4. lcd.asm
5. printf.asm
6. wire1.asm
7. macros.asm

b. Autre

1. Interpolation.py

```
;
; Projet_Microcontroleurs_v1.asm
;
.include "macros.asm"
.include "definitions.asm"
;================================ "reservation" memoire données et definitions
=============================================
 .equ EEPROM = 0b10100000
 .equ R = 1
 .equ   baud    = 9600
 .equ capt_dist_enable = $100 ;Booléen
 .equ dist_sound = $102 ;Booléen
 .equ temperature0 = $104 ;LSB temperature
 .equ temperature1 = $106 ;MSB temperature
 .equ temp_measure_enable = $108 ;Booléen
 .equ memory_adress_MSB = $110 ;Adresse MSB pour I2C
 .equ memory_adress_LSB = $112 ;Adresse LSB pour I2C
 .equ memory_data_1 = $114 ;Unités
 .equ memory_data_2 = $116 ;Dizaines
 .equ frequence_buzzer = $118 ;Pas vraiement une fréquence mais valeur qui va
dans OCR2 et règle la fréquence du buzzer
;================================  interupt vector table
==============================
.org $00
        rjmp reset
;-------------------
.org $02
        rjmp button0
;-------------------
.org $04
        rjmp button1
;-------------------
.org $06
        rjmp button2
;-------------------
.org $08
        rjmp button3
;-------------------
.org OC2addr
        rjmp oc2
;-------------------
.org $30
;========================================== interupt routines
==================================
button0:
        in _sreg, SREG
        lds w, dist_sound
        com w
        sts dist_sound, w
        out SREG, _sreg
        reti
;-------------------
button1:
        in _sreg, SREG
```

```
        lds w, capt_dist_enable
        com w
        sts capt_dist_enable, w
        out SREG, _sreg
        reti
;-------------------
button2:
        in _sreg, SREG
        lds w,temp_measure_enable
        com w
        sts temp_measure_enable, w
        out SREG, _sreg
        reti
;-------------------
button3:
        in _sreg, SREG
        rcall write_terminal
        WAIT_MS 30
        out SREG, _sreg
        reti
; ================================== timer routine
==============================
oc2:
        in _sreg, SREG
        mov _u, u
        mov _w, r0
        lds r0, dist_sound
        sbrc r0, 0
        INVP PORTE, SPEAKER
        mov r0, _w
        mov u, _u
        out SREG, _sreg
        reti
; ===================================== reset
========================================
reset:
        LDSP RAMEND ;Load stack pointer
        OUTI DDRC, $ff  ;Leds output
        OUTI ADCSR, (1<<ADEN) + 6   ;Authorise CAN
        OUTI ADMUX, CAPT_DIST ;Multiplexeur du CAN sur le capteur de distance
        OUTI TCCR2, 0b00001100 ;Timer 2 output compare settings
        OUTI EIMSK, 0b00001111  ;Authorise les interupt sur les buttons 0 , 1 ,
2 et 3
        ldi r20, 0b10101010 ; Les interruptions sur les buttons 0 , 1 , 2 et 3
se font sur les flancs descendants
        sts EICRA, r20  ;Les interruptions sur les buttons 0 , 1 , 2 et 3 se
font sur les flancs descendants
        OUTI TIMSK, (1<<OCIE2) ;Authorise timer output compare
        OUTI    DDRE,0b00000010 + (1<<SPEAKER)  ; Make Tx (PE1) and SPEAKER an
output
        sbi     PORTE,PE1 ;Communication i2c
        in r16, SFIOR   ;Communication i2c
        ori r16, (1<<PUD)       ;Communication i2c
        out SFIOR, r16  ;Communication i2c
```

```
        rcall wire1_init
        rcall i2c_init
        rcall LCD_init
        rjmp intro
; ======================================== data
====================================
lut: ;Look-up-table pour le capteur de distance (précision de 1cm)
        .dw 0, 1000, 130, 80, 131, 79, 132, 78, 133, 77, 134, 76, 134, 75, 135,
74, 136, 73, 137, 72, 138, 71, 140, 70, 142, 69, 145, 68, 147, 67, 149, 66, 151,
65, 154, 64, 156, 63, 158, 62, 160, 61, 163, 60, 167, 59, 170, 58, 173, 57, 176,
56, 179, 55, 182, 54, 185, 53, 188, 52, 191, 51, 195, 50, 199, 49, 204, 48, 208,
47, 212, 46, 216, 45, 220, 44, 224, 43, 229, 42, 233, 41, 238, 40, 243, 39, 249,
38, 255, 37, 260, 36, 266, 35, 272, 34, 277, 33, 283, 32, 288, 31, 297, 30, 308,
29, 319, 28, 330, 27, 341, 26, 352, 25, 363, 24, 374, 23, 390, 22, 406, 21, 422,
20, 438, 19, 454, 18, 472, 17, 506, 16, 540, 15, 573, 14, 607, 13, 645, 12, 686,
11, 728, 10,800,0,1024 ;toutes les datas un nombre PAIRE
;==================================  include
==================================
.include "printf.asm"
.include "wire1.asm"
.include "i2cx.asm"
.include "lcd.asm"
;====================================== intro
========================================
intro:
        clr w ;preset
        sts capt_dist_enable, w ;preset
        sts dist_sound,w ;preset
        sts memory_adress_MSB, w ;preset
        sts memory_adress_LSB, w ;preset
        sts temp_measure_enable, w ;preset
        ldi b0, 20 ;preset
        out OCR2, b0  ;preset
        sts frequence_buzzer, b0 ; preset
        ldi _w, 18 ;preset
        sts temperature0, _w ;preset
        sts temperature1, _w ;preset
        sei    ;Authorise les interruptions
        rjmp main
;====================================== main
========================================
main:
        WAIT_MS 10 ;Fluidifie l'affichage LCD
;------------------
        sbi ADCSR, ADSC ;Initie la conversion du CAN
        WP1 ADCSR, ADSC ;Attends que la conversion démare
        in a0, ADCL ;Recupère le resultat du CAN
        in a1, ADCH ;Recupère le resultat du CAN
;------------------
        lds b0, frequence_buzzer ;Son proportionnel à la distance
        out OCR2, b0 ;Son proportionnel à la distance
;------------------
        ldi _w, 0b1000000 ;trick pour mesure de distance sans buggs
        mov e0, _w ;trick pour mesure de distance sans buggs
```

```
        JRO e0,7,distance ;trick pour mesure de distance sans buggs
        distance_fini: ;trick pour mesure de distance sans buggs
;-------------------
        lds r20, temp_measure_enable ;mesure de temprature ou pas
        cpi r20, 0 ;mesure de temperature ou pas
        breq suite ;mesure de temperature ou pas
        nop ;evite les buggs
        rjmp main
suite:
        rcall temp ;mesure de temperature ou pas
        temp_fini: ;mesure de temperature ou pas
        rjmp main
;============================================== sous routines
========================================
;------------------------------------- DISTANCE + affichage
-----------------------------------
distance:
        mov _u, r0
        lds r0, capt_dist_enable
        sbrs r0, 0
        rjmp dist
        lds a0, temperature0
        lds a1, temperature1
        rcall lcd_clear ;_____________ Affichage de la temperature seule
_____________
        PRINTF LCD
        .db     "Temp=",FFRAC2,a,4,$22," C",CR,0
        ldi b0, 255
        rjmp distance_fini
dist:    ;____________________ Calcul de la distance _____________________
        ldi zl, low(2*lut)-3
        ldi zh, high(2*lut)
        push zl
        push zh
repeat: ;_________ Parcours la lut (look-up-table) tant que la valeur de la
tensionn est inférieure à la valeur du capteur ___________
;do{
        pop zh
        pop zl
        ldi _w, 3
        ADDZ _w
        lpm
        mov c0, r0
        adiw zl, 1
        lpm
        mov c1, r0
        push zl
        push zh
        CP2 a1,a0,c1,c0
        brsh repeat       ;}while(mesure <= valeur_lut)
        pop zh  ; ; _____________________ Récupère la distance correspondant à la
tension de la lut _________________
        pop zl
        ldi a2, 3
```

```
        ldi a3, 0
        SUB2 zh,zl,a3,a2
        lpm
        mov c0, r0
        adiw zl, 1
        lpm
        mov c1, r0
        lds a1, temperature1
        lds a0, temperature0 ; _____________________ Affichage temperature +
distance _________________
        rcall lcd_clear
        PRINTF LCD
        .db     "Temp=",FFRAC2,a,4,$22," C",CR,0 ;affichage 4 C.S.
        rcall LCD_lf
        PRINTF LCD
        .db "Dist=",FDEC2,c," cm",CR,0  ;affichage
        inc c0
        mov b0, c0
        sts frequence_buzzer, b0
        rjmp distance_fini
;-------------------------------------- TEMPERATURE ---- "temp"
--------------------------------------
temp: ;_____________________________ Utilisation protocole 1-wire
_____________________________
        OUTI    PORTC, 0xff
        rcall   wire1_reset                     ; send a reset pulse
        CA              wire1_write, skipROM    ; skip ROM identification
        CA              wire1_write, convertT   ; initiate temp conversion
        WAIT_MS 750                                     ; wait 750 msec
        rcall   wire1_reset                     ; send a reset pulse
        CA              wire1_write, skipROM
        CA              wire1_write, readScratchpad
        rcall   wire1_read                      ; read temperature LSB
        mov             c0,a0
        rcall   wire1_read                      ; read temperature MSB
        mov             a1,a0
        mov             a0,c0
        sts temperature0, a0
        sts temperature1, a1
        ;_________________ temperature dans a1, a0 va etre transformée en stockage
en EEPROM et en memoire données _______________
        DIV2 a0
        DIV8 a0
        MUL8 a1
        MUL2 a1
        or a0, a1  ; On a ici dans a0 l'entier de la temperature
        ;_____________ Séparation des deux chiffres decimaux pour stockage en
EEPROM puis affichage Terminal _________________
        mov a2, a0 ;_____________ Cherche le chiffre des dizaines ___________
        ldi a1, 0
        cpi a0, 10
        brmi dizaines
        ldi a1, 1
        cpi a0, 20
```

```
        brmi dizaines
        ldi a1, 2
        cpi a0, 30
        brmi dizaines
        ldi a1, 3
        cpi a0, 40
        brmi dizaines
        ldi a1, 4
dizaines: ;____________________ Ecrit le chiffre des dizaines en mémoire
données_____________
        WAIT_US 10
        ldi a3, $30
        add a1, a3
        WAIT_US 10
        sts memory_data_2, a1
unites: ;____________________ Cherche le chiffre des unités _______________
        WAIT_US 10
        mov a0, a2
        WAIT_US 10
        lds a2, memory_data_2
        subi a2, $30

        mov a3, a2
        MUL8 a2
        add a2, a3
        add a2, a3

        sub a0, a2
        ldi a3, $30
        add a0, a3
        sts memory_data_1, a0 ;_____________ Ecrit le chiffre des unités en
memoire donnés _____________
        rcall write_memory_i2c_with_data   ;___________ Stockage en EEPROM
_____________
        write_memory_fini:
        rjmp temp_fini
;---------------------------------------------------- WRITE_TERMINAL
-------------------------------------
write_terminal:
        WAIT_MS 100
        CA i2c_start, EEPROM   ;_____________ Selectionne l'adresse de lecture au
debut de la memoire _________
        CA i2c_write, $00
        CA i2c_write, $00
        CA i2c_rep_start, EEPROM+R
loop_:
        WAIT_MS 100   ;_________________ Ecrit sur le terminal jusqu'a la
detection du caractère end of file ___________
;while(caractère /= ~){
        rcall i2c_read
        cpi a0, $7e ;caractère ~ indique un end of file
        breq fini
        rcall putc
        rcall i2c_ack
```

```
        rcall i2c_read
        rcall putc
        rcall i2c_ack

        ldi a0, $2C ;virgule pour séparer les chiffres sur le terminal
        rcall putc ;virgule
        rjmp loop_    ;} fin du while
fini:
        rcall i2c_ack
        ldi a0, $20 ;espace pour séparer les séries de donné sur le terminal
        rcall putc
        rcall i2c_read
        rcall i2c_no_ack
        rcall i2c_stop
        ret
;---------------------------------------- WRITE_I2C_MEMORY_WITH_DATA
----------------------------------------
write_memory_i2c_with_data:
        WAIT_MS 10
        CA i2c_start, EEPROM
        lds a0, memory_adress_MSB ;_________ Selectionne l'adresse d'ecriture
_____________
        rcall i2c_write
        lds a0, memory_adress_LSB
        rcall i2c_write
        cli   ;Desactive temporairement les interruption tant que les end of
file ne sont pas réécrits
        lds a0, memory_data_2 ; ________ Ecrit à l'adresse sélectionnée
_____________
        rcall i2c_write
        lds a0, memory_data_1
        rcall i2c_write
        rcall i2c_stop

        WAIT_MS 10
        lds  r0, memory_adress_LSB  ;__________ Incrémente l'adresse d'ecriture
_____________
        lds r1, memory_adress_MSB
        INC2 r1,r0
        INC2 r1,r0
        sts memory_adress_LSB, r0
        sts memory_adress_MSB, r1

        WAIT_MS 10
        CA i2c_start, EEPROM  ; _____________________ Ecrit ~~ deux fois le
caractère de end of file _____________
        lds a0, memory_adress_MSB
        rcall i2c_write
        lds a0, memory_adress_LSB
        rcall i2c_write
        ldi a0, $7e
        rcall i2c_write
        ldi a0, $7e
```

```
        rcall i2c_write
        sei ;Réactive les interruptions maintenant que les end of file ont été
réecrits
        rcall i2c_stop
        rjmp write_memory_fini

;========================= UART sous routines ~= Librairie uart1.asm
=========================
wait_bit:
        WAIT_C  (clock/baud)-18 ; subtract overhead
        ret
wait_1bit5:
        WAIT_C  (3*clock/2/baud)-18
        ret
putc:   cbi     PORTE,PE1       ; set start bit to 0
        rcall   wait_bit        ; wait 1-bit period (start bit)
        sec                     ; set carry
loop:   ror     a0              ; shift LSB into carry
        C2P     PORTE,PE1       ; carry to port
        rcall   wait_bit        ; wait 1-bit period (data bit)
        clc                     ; clear carry
        tst     a0              ; test c for zero
        brne    loop            ; loop back if not zero
        sbi     PORTE,PE1       ; set stop bit to 1
        rcall   wait_bit        ; wait 1-bit period (stop bit)
        ret
getc:   WP1     PINE,PE0        ; wait if pin Rx=1
        rcall   wait_1bit5      ; wait 1.5-bit period (start bit)
        ldi     a0,0x80         ; preload c
```

```
; 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     = PORTC ; 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 M2 (encoder/speaker/IR remote) ===
.equ    CAPT_DIST       = 3     ; capteur de distance


; === 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
```

```
; file  i2cx.asm   target ATmega128L-4MHz-STK300
; purpose extended I2C (400 k bit/s), software emulation

; === definitions ===
.equ    SDA_port= PORTB
.equ    SDA_pin = SDA
.equ    SCL_port= PORTB
.equ    SCL_pin = SCL
/*.equ  SDA_port= PORTD
.equ    SDA_pin = 1
.equ    SCL_port= PORTD
.equ    SCL_pin = 0*/

; === macros ===
; these macros control DDRx to simulate an open collector
; with external pull-up resistors

.macro  SCL0
        sbi     SCL_port-1,SCL_pin      ; pull SCL low (output, port=0)
        .endmacro
.macro  SCL1
        cbi     SCL_port-1,SCL_pin      ; release SCL (input, hi Z)
        .endmacro
.macro  SDA0
        sbi     SDA_port-1,SDA_pin      ; pull SDA low (output, port=0)
        .endmacro
.macro  SDA1
        cbi     SDA_port-1,SDA_pin      ; release SDA (input, hi Z)
        .endmacro

.macro  I2C_BIT_OUT     ;bit
        sbi     SCL_port-1,SCL_pin      ; pull SCL low (output, port=0)
        in      w,SDA_port-1            ; sample the SDA line
        bst     a0,@0                           ; store a0(bit) to T
        bld     w,SDA_pin                       ; load w(SDA) with T
        out     SDA_port-1,w            ; transfer bit_x to SDA
        cbi     SCL_port-1,SCL_pin      ; release SCL (input, hi Z)
        rjmp    PC+1                    ; wait 2 cyles
        .endmacro

.macro  I2C_BIT_IN      ;bit
        sbi     SCL_port-1,SCL_pin      ; DDRx=output   SCL=0
        cbi     SDA_port-1,SDA_pin      ; release SDA (input, hi Z)
        cbi     SCL_port-1,SCL_pin      ; DDRx=input    SCL=1
        nop                                     ; wait 1 cycle
        in      w,SDA_port-2            ; PINx=PORTx-2
        bst     w,SDA_pin                       ; store bit read in T
        bld     a0,@0                           ; load a0(bit) from T
        .endmacro

; === routines ===
i2c_init:
        cbi     SDA_port,  SDA_pin      ; PORTx=0 (for pull-down)
        cbi     SCL_port,  SCL_pin      ; PORTx=0 (for pull-down)
```

```
        SDA1                                    ; release SDA
        SCL1                                    ; release SCL
        ret

i2c_rep_start:
; in:   a0 (byte to transmit)
        SCL0
        SDA1
        SCL1
i2c_start:
; in:   a0 (byte to transmit)
        SDA0
i2c_write:
        com     a0                                      ; invert a0
        I2C_BIT_OUT 7
        I2C_BIT_OUT 6
        I2C_BIT_OUT 5
        I2C_BIT_OUT 4
        I2C_BIT_OUT 3
        I2C_BIT_OUT 2
        I2C_BIT_OUT 1
        I2C_BIT_OUT 0
        com     a0                                      ; restore a0
i2c_ack_in:
        SCL0
        SDA1                                    ; release SDA
        SCL1
        in      w,SDA_port-2            ; PINx=PORTx-2
        bst     w,SDA_pin                       ; store ACK into T
        ret

i2c_read:
; out:  a0 (byte read)
        I2C_BIT_IN 7
        I2C_BIT_IN 6
        I2C_BIT_IN 5
        I2C_BIT_IN 4
        I2C_BIT_IN 3
        I2C_BIT_IN 2
        I2C_BIT_IN 1
        I2C_BIT_IN 0
        ret

i2c_ack:
        SCL0
        SDA0
        SCL1
        ret

i2c_no_ack:
        SCL0
        SDA1
        SCL1
        ret
```

```
i2c_stop:
        SCL0
        SDA0
        SCL1
        SDA1                            ; release again
        ret
```

```
; 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

puts:
        lpm
        tst r0
        breq puts_done
        mov a0, r0
        rcall lcd_putc
        adiw zl, 1
        rjmp puts
puts_done: ret
```

```
; 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,0     1-byte variable, 5-digit, decimal, signed
; "a=",FDEC|FDIG5,a,0           1-byte variable, 5-digit, decimal, unsigned

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

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

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

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

.equ    FFRAC   = 0b10001000    ; 1-byte variable
.equ    FFRAC2  = 0b10001010    ; 2-byte variable
.equ    FFRAC3  = 0b10001100    ; 3-byte variable
.equ    FFRAC4  = 0b10001110    ; 4-byte variable
```

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

.equ    FSIGN   = 0b00000001

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

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

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

; mod   y,z


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

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

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

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

_printf_formatted:

; FDEC  11bb'iiis
; FBIN  101i'iiis
```

```
; FHEX  1001'iiis
; FFRAC 1000'1bbs
; FCHAR 1000'0100
; FSTR  1000'0101

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

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

        rjmp    _printf_read

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

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

; === putstr (put string) =================================
; in    x       pointer to ascii string
;       b3,b2   address of output routine (UART, LCD putc)
_putstr:
        ld      w,x+
        tst     w
        brne    PC+2
```

```
        rjmp    _printf_read
        rcall   _putw
        rjmp    _putstr

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

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

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

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

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

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

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

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

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

_getdig:
        mov     b1,w
        lsr     b1
```

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

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

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

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

        rjmp    _getnum

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

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

_printf_1b:                     ; sign extension
        clr     a1
        brtc    PC+3    ; T=1 sign extension
        sbrc    a0,7
        ldi     a1,0xff
_printf_2b:
        clr     a2
        brtc    PC+3    ; T=1 sign extension
        sbrc    a1,7
        ldi     a2,0xff
_printf_3b:
        clr     a3
        brtc    PC+3    ; T=1 sign extension
        sbrc    a2,7
        ldi     a3,0xff
_printf_4b:

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

```
        rjmp    _printf_read

; ============================================
; func  ftoa
; converts a fixed-point fractional number to an ascii string
; 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
```

```
; 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
```

```
; 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 JPO  ;jump if bit = 0
        sbis @0,@1
        rcall @2
        .endmacro

.macro JRO  ;jump if bit = 0
        sbrs @0,@1
        rcall @2
        .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
```

```python
import numpy as np
import matplotlib.pyplot as plt

#Valeurs prises sur l'image en pixels
X = np.array([106,138,145,157,171,190,221,261,311,378,448,515,583,648])
#Valeurs prises sur l'image en pixels
Y = np.array([412,37,30,66,122,168,229,266,298,320,337,349,358,361])

X = (X-106)*(80/(643-106))  #Conversion pixels en cm
Y = (412-Y)*(3/(412-48))    #Conversion pixels en V
x = X
y = Y

from scipy import interpolate   #interpolation linéaire
f = interpolate.interp1d(x, y)
y = f(x)
plt.plot(x,y) #afficher valeurs et interpolation
plt.plot(X,Y,'+')
plt.grid()
plt.title("Abaque du Capteur de distance")
plt.xlabel("Distance en cm")
plt.ylabel("Tension en V")
plt.show()

x = np.linspace(10,80,71)#extrait une valeur tous les cm de l'interpolation
y = f(x)*1024/3.3
nx =list(reversed(x))
ny =list(reversed(y))
look_up_table = [0,1000]
for k in range (len(nx)): #intercale tension sur 1024 et distance correspondante
    look_up_table.append(int(ny[k]))
    look_up_table.append(int(nx[k]))
look_up_table.append(int(1024))
look_up_table.append(int(0))

#affiche le resultat sous forme de liste pour le copier dans Atmel Studio
print(look_up_table)
```