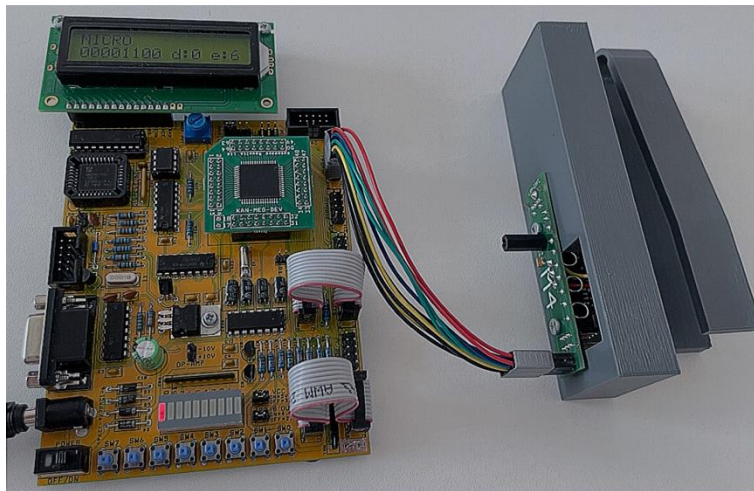


Rapport de Projet

Lecteur de Morse



1. Introduction

Alors que le code morse international est aujourd'hui un moyen de communication dépassé, son utilisation n'en n'est pas pour autant terminée. C'est ce que nous allons mettre en œuvre dans ce projet. L'objectif est de permettre aux personnes muettes (mutisme) de pouvoir s'exprimer en société à l'aide d'un petit appareil portable, capable de décrypter le code morse tapé par la personne.

Pour réaliser ce projet, nous avons convenu comme objectifs de réaliser une application :

- Possédant une implémentation utile pour certaines personnes au quotidien ;
- Respectant les conventions internationales du morse ;
- Nécessitant aucune connaissance préalable en lien avec le développement du système mais uniquement des bases en morse.

En plus de ces objectifs cibles, la donnée du projet nous a imposé certaines contraintes qu'il nous a fallu respecter qui consistaient en un programme :

- Diversifié en tâches ;
- Possédant des sous-routines et macros réutilisables et détaillées ;
- Qui puisse répondre en temps réel à l'utilisateur.






En ce qui concerne la mise en œuvre de ce projet, plusieurs périphériques se sont avérés indispensables à savoir :

- L'affichage LCD 2x16 (Hitachi44780 U 2x16 LCD) qui joue le rôle d'interface graphique et de traducteur de morse ;
- La télécommande infrarouge IR Remote Control Vivanco UR Z2, qui sert à parcourir le menu et changer la vitesse d'écriture du morse (prescaler)
- Détecteur de distance (SHARP G P2Y 0A21)
- Encodeur angulaire

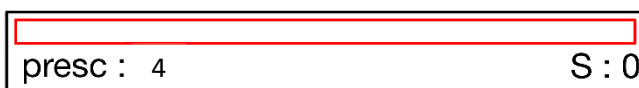
Pour développer notre traducteur de morse, nous avons utilisé le logiciel Atmel Studio 7.0 pour écrire notre programme en assembleur, ainsi que le logiciel AVRISP-U afin de charger le code désassemblé sur notre carte.

2. Mode d'emploi

Pour interagir avec le traducteur de morse, l'utilisateur a besoin du téléscripteur imprimé en 3D et de la télécommande (boutons : +, -, chanel up et chanel down, et mute).

	Remote control functionality	Sent value
	Increment prescaler	0x10
	Decrement prescaler	0x11
	Enter Menu	0x20
	Exit Menu	0x21
	Clear message from display	0x0d

A l'allumage de la carte, nous arrivons dans le menu (cf 1), et 2 données sont affichées sur le LCD. A gauche la valeur du prescaler mise à 5 par défaut, sous la forme «presc : 4». Et à droite la valeur du décalage du message sous la forme «s : 0». Le prescaler se règle à l'aide du bouton + et – de la télécommande et le décalage avec l'encodeur. Pour rentrer dans le mode lecture de morse (cf 2), il suffit de presser le bouton chanel down. Puis s'affiche 2 autres données. A gauche la valeur courante de la variable nb_dot sous la forme «d : 0», et à droite la valeur courante de la variable nb_empty sous la forme «e : 7». Puis en tapant sur la membrane, des temps court pour des points et long pour des traits (temps relatif au prescaler), il vous est possible de rentrer votre message en morse qui s'affichera en lettre au LCD. Si votre message dépasse les 16 caractères (espace compris), il vous est possible de le visualiser en entier. Il faut pour cela revenir dans le menu en pressant le bouton chanel up, pour tourner l'encodeur dans le sens anti-horaire pour faire défiler votre message de droite à gauche. Il vous est possible de clear le message affiché en pressant le bouton mute.



Ecran d'affichage du menu (1)



Ecran d'affichage mode lecture (2)

3. Description technique

1. Phases de traduction : Réalisation de points techniques

Quand le programme est dans le menu, il commence à décoder le morse en continu. Le programme suit le décodage Morse international, avec un intervalle de temps défini dans le menu par le prescaler. Nous allons brièvement rappeler le fonctionnement de l'écriture d'un message en morse ci-dessous, pour plus de détails sur la compréhension du morse vous retrouverez en annexe une page Wikipédia sur laquelle nous nous sommes basés pour faire notre algorithme de lecture de morse.

1.1) Fonctionnement du morse:

Le morse se base sur la combinaison unique de signaux court, et long:

- Un signal court à une durée de 1 unité de temps

- Un signal long a une durée de 3 unités de temps

Pour écrire une lettre en langage Morse il faut effectuer une combinaison unique de signaux court et long.

- 1 unité de temps permet de passer au prochain signal
- 3 unités de temps sont nécessaire pour passer a la prochaine lettre
- 7 unités de temps sont nécessaire pour passer au prochain mot

1.2) Lecture de combinaisons de signaux par le programme

Pour notre programme, une lecture de distance inférieure à une distance limite permet de générer un signal court (passer sa main devant le capteur de distance)

Avec cette convention d'écriture, notre programme pourra décoder les signaux envoyés par l'utilisateur au moyen du téléscripateur et en afficher le message décodé sur le LCD.

Pour la lecture, notre programme se base sur l'analyse du nombre de signaux court reçu et du nombre d'unités de temps écoulé entre chaque signaux

1.3) Decodage du signal

Une fois la combinaison de signaux décodés, notre programme doit pouvoir déterminer à quelle lettre de l'alphabet cette combinaison correspond. Pour ce faire, nous utilisons une méthode ingénieuse qui va permettre d'aller lire automatiquement le caractère ASCII correspondant à la combinaison en quelques étapes. C'est un moyen rapide qui nous évite de comparer la combinaison décodée à chaque combinaison possible. La méthode est la suivante :

Si l'on considère qu'un signal court correspond à un zéro, et un long à un 1, une combinaison de signaux court et long peut être interprétée comme un nombre binaire:

Exemple: `._.` ("C") --> `0b1010` = 10

Étant donné qu'un nombre binaire se lit de droite à gauche, et non de gauche à droite, il est nécessaire de "flip" la combinaison pour pouvoir bien décoder chaque combinaison correctement. Ce flip n'est que nécessaire ici pour le rapport, car le programme stocke déjà chaque signal de droite à gauche.

Cependant, cette représentation donne les mêmes nombres binaires pour plusieurs lettres: `."` (E), `.."` (I), `"..."` (S), `"...."` (H) représentent tous le chiffre 0. Pour résoudre ce problème, nous pouvons simplement comparer la taille des signaux: le signal `."` a une taille de 2, le signal `"..."` a lui une taille de 4. De cette manière, le nombre binaire correspondant à la combinaison de signaux et la longueur de la combinaison sont deux dimensions qui nous permettront d'identifier chaque lettre de manière très efficace.

Il nous suffit donc de ranger les caractères ASCII correspondants à chaque lettre dans un tableau a deux dimensions, où la longueur de la combinaison Morse d'une lettre et sa représentation en binaire correspond, respectivement, à sa ligne et a sa colonne dans le tableau.

La SRAM n'ayant pas ces aspects de ligne et de colonne, mais contenant uniquement des adresses consécutives, il nous faudra juste définir deux constantes: `TABLE_START` et `TABLE_WIDTH`. Ces valeurs représentent le début du tableau en mémoire et sa largeur, respectivement, pour pouvoir placer ce tableau dans la SRAM.

Voici comment notre programme pourra, de manière très rapide, convertir une combinaison de signaux court et long en un caractère ASCII et l'afficher sur le LCD

1.4) Exemple

Déroulement pour decodage du signal `._.`:

- Représentation binaire: `._.` -> Flip -> `._.` = `0b0101` = 5
- Longueur de la combinaison: 4
- Le caractère ASCII correspondant se situe donc à la ligne 4, colonne 5
- Adresse contenant le caractère correspondant: `TABLE_START + (4-1)*TABLE_WIDTH + 5`

En remplaçant les valeurs: `TABLE_START = 0x0130` et `TABLE_WIDTH = 0x0010` on trouve, l'adresse `0x130 + 3*0x10 + 5 = 0x165`. En seulement 3 opérations nous avons pu décoder le signal `._.` en une adresse, `0x0165` où se trouvera le caractère correspondant à cette combinaison (ici "C" = `0x43`) ce qui nous permet bien de décoder du morse.

1.5) Nom des variables équivalentes dans le code:

TABLE_START: adresse du début du tableau dans la SRAM, Constant

TABLE_WIDTH: nombre d'adresses dans une ligne du tableau, constant

Ltr_col: variable sur 1 byte, contient la représentation binaire de la combinaison du signal lu

Ltr_col_bit: variable sur 1 byte, contient le bit de ltr_col à changer lors de la lecture du prochain signal, correspond à la longueur de la combinaison

Nb_dot: variable sur 1 byte, compteur pour le nombre de signal court, utilisé pour détecter un signal long

Nb_empty: variable sur 1 byte, compteur pour le nombre d'espacement entre deux signaux, utilisé pour détecter le passage entre deux lettres ou deux mots

II. Description technique du matériel

Le montage technique du matériel (après avoir chargé le programme sur l'appareil) consiste à assembler les périphériques sur la carte STK-300 :

- L'écran LCD se relie directement au port LCD prévu à cet effet. Ce dernier a pour objectif d'afficher des données reçues depuis le microcontrôleur (réglés en output).
- La mise sous tension du système s'effectue en reliant la carte au réseau électrique domestique (230V – 50Hz). Les périphériques sont alimentés électriquement directement depuis la carte qui gère l'alimentation en électricité par des ports prévus à cet effet.
- En ce qui concerne la télécommande, elle agit de manière extérieure à la carte émettant des données sous le format RC5 qui sont détectées par le capteur infrarouge du module M2 qui à son tour communique ces données avec la carte à travers le PIN 7 du PORT E.
- Pour le capteur de distance, il est connecté au PORT F par l'intermédiaire de 10 câbles raccordeurs. Par ailleurs, les LEDs intégrées à la carte servent seulement à indiquer que le programme a bien été chargé sur la carte et n'ont donc pas d'utilité directe dans le cadre de notre projet.
- L'encodeur angulaire est situé sur le PORT E et est rendu effectif lors de la mise sous tension de la carte.

III. Interruptions

2.1) INT7

Nous utilisons la télécommande afin de naviguer à travers les multiples états de notre programme de manière asynchrone, indépendamment de la tâche en cours. Pour ce faire, nous devons utiliser une interruption externe. Le pin IR de la télécommande étant à l'état 1 au repos, il nous faut configurer le registre EICRB sur un flanc descendant afin de pouvoir déclencher une interruption lors de la réception d'un signal de la télécommande.

Ainsi, le fait de presser un bouton sur la télécommande va envoyer un signal qui va être reçu par le capteur IR du module M5, qui va lui mettre le PIN7 du PORTE à 0 qui correspond au pin pour INT7 et va donc déclencher une interruption sur flanc descendant, afin de décoder la commande envoyée et effectuer la tâche appropriée.

2.2) Timer 0

Comme la lecture du morse se fait de manière synchrone, nous devons utiliser un timer afin de lire les signaux à des intervalles précis. Nous avons donc choisi le tim0ovf, qui génère une interruption à un intervalle défini dans le prescaler (TCCR0), que l'on pourra modifier et redéfinir dans le menu. L'utilisateur peut choisir la valeur à mettre dans TCCR0, allant de 1 à 7, à l'aide de la télécommande, et permettra donc de définir la vitesse de lecture/écriture en morse.

La sous routine de service de l'interruption contient l'algorithme qui décode progressivement le signal en sa représentation binaire, en utilisant les variables ltr_col, ltr_col_bit, nb_dot, nb_empty et d'autres.

IV. Fonctionnement du programme

3.1) Top Down

La partie principale du programme se situe dans le fichier main.asm, c'est ce code qui est set as entry file dans notre projet. À l'exécution du programme nous allons en premier lieu dans le reset, qui va initialiser les

périphériques (directions des ports), appeler des sous routines d'initialisations (pour nos périphériques et nos variables), voir section Description des modules pour plus de détails sur l'accès au périphériques.

Un point important de la structure de notre code est l'utilisation d'un Status Register personnalisé que l'on a nommé FREG. Le FREG (Flag Register) est une variable sur 1 byte stocké en SRAM qui contient 8 flag de contrôle pour détecter plusieurs changements d'état distincts lors de l'exécution du programme, les 8 flag et leur bit correspondant sont représenté sur la Figure 3.1 ci-dessous.

Ensuite, le programme va dans le label menu, qui va lire en boucle les valeurs de l'encodeur et les signaux de la télécommande (voir section sur l'accès au périphérique), et modifier les valeurs correspondantes dans le menu (message scroll et valeur de prescaler) puis lire le flag MENU du FREG pour savoir quand sortir du menu.

Une fois sorti du menu, le programme va dans le label reading, qui va, selon la distance lue par le capteur de distance (voir la section sur l'accès au périphériques), modifier le flag DETECTED. Puis il va lire la valeur des lettres décodées en SRAM et les afficher sur le LCD afin que l'utilisateur puisse lire le message qu'il a écrit. Dans cette partie le programme va également afficher deux variables de contrôle: nb_dot et nb_empty qui servent à aider l'utilisateur à vérifier si son signal est bien décodé.

3.2) Zone Mémoire Reservée

Le tableau ci-dessous représente la mémoire SRAM par ranger de 16 adresses (pour des raisons visuelles). On peut y voir exactement à quelles adresses sont stockées toutes nos variables de contrôle. Nous utilisons donc le banc d'adresses allant de 0x0100 à 0x0170 (0x0170 exclu) qui peut donc être visualisé comme un tableau de 7 lignes de 16 adresses. La première ligne (les premières 16 adresses) sont utilisées pour stocker nos variables de contrôle. Les prochaines 32 adresses (lignes 2 et 3 du tableau) contiennent les lettres décodées formant le message écrit par l'utilisateur (qui seront affichées sur le LCD). Finalement, les dernières 64 adresses correspondent à la table de correspondance qui contient la valeur ASCII de chaque lettre (expliqué dans la section sur le décodage du signal).

L'adresse de début et de fin définissant la zone de mémoire réservée est définie dans le fichier variables control.asm

La définition des labels pour chaque variables est définie dans le fichier variables_definition.asm

3.3) Constantes de calibrations

Nous utilisons plusieurs constantes définies dans plusieurs fichiers: au début du main sont définies les constantes tels que MENU_REFRESH_RATE_MS, qui définit la vitesse de lecture des input de l'utilisateur lorsqu'il est dans le menu, ou encore INIT_PRESCALER qui définit la valeur initiale du prescaler. On y trouve également les constantes du morse, les valeurs des signaux pour chaque boutons de la télécommande, ainsi que les bits correspondant à chaque flag du FREG. Le fichier table.asm contient lui toutes les constantes lié au tableau de correspondance en SRAM tel que TABLE_START et TABLE_WIDTH (voir section décodage du signal)

4. Présentation des modules

Le schéma ci-dessous représente les différents modules et leurs fonctionnalités dans le projet

Module		Contenu
1. Definitions.asm	:	définitions des variables de base de l'ATMEGA-128
2. Encoder.asm	:	sous routines d'accès à l'encoder angulaire
3. Lcd.asm	:	sous routines d'accès au LCD
4. Macros.asm	:	macros à usage générique
5. Main.asm	:	code principale du programme, définitions de constantes
6. Printf.asm	:	sous routines d'accès au lcd
7. Remote.asm	:	sous routines d'accès à la télécommande
8. Sharp.asm	:	sous routines d'accès au capteur de distance
9. Table.asm	:	définition de la table de correspondance en SRAM et de ses constantes

- | | | |
|------------------------------|---|---|
| 10. Variables_control.asm | : | sous routines de contrôle des variables |
| 11. Variables_definition.asm | : | définitions des variables en SRAM |

5. Accès aux périphériques

4.1) Liste des périphériques et driver utilisé

Pour ce programme, nous utilisons en tous les périphériques suivant:

PORT

- L'affichage LCD, directement branché sur la carte ATMEGA-128
- La télécommande, connecté sur le PORTE
- L'encodeur angulaire connecté sur le PORTE
- Le détecteur de distance connecté sur le PORTF
- Les 8 panneaux de LED connecté sur le PORTB

Chaque périphérique est accédé à travers leur driver respectif:

DRIVER(S)

- L'affichage LCD lcd.asm printf.asm
- La télécommande, remote.asm
- L'encodeur angulaire encoder.asm
- Le détecteur de distance sharp.asm

Les fichiers lcd.asm, printf.asm, remote.asm, encoder.asm, sharp.asm ont été pris des TP vu en semestre ou du livre du cours. Seuls deux sous routines ont été ajoutées au fichier encoder.asm, qui sont les deux sous routines appelées respectivement lors d'une rotation vers la gauche et la droite de l'encodeur.

4.2) Accès aux périphériques par les sous routines


Ainsi, on appelle les sous routines des fichiers suivant pour accéder au peripheriques:

	Driver	Description
1. LCD_init	lcd.asm	initialise le LCD
2. LCD_clear	lcd.asm	clear le LCD
3. LCD_home	lcd.asm	ramener le curseur à l'origine
4. PRINTF	printf.asm	print sur le lcd
5. encoder_init	encoder.asm	initialiser l'encodeur angulaire
6. read_encoder	encoder.asm	lire la rotation de l'encodeur
7. read_sharp	sharp.asm	lire la distance du sensor SHARP
8. read_remote	remote.asm	lire le signal de la télécommande

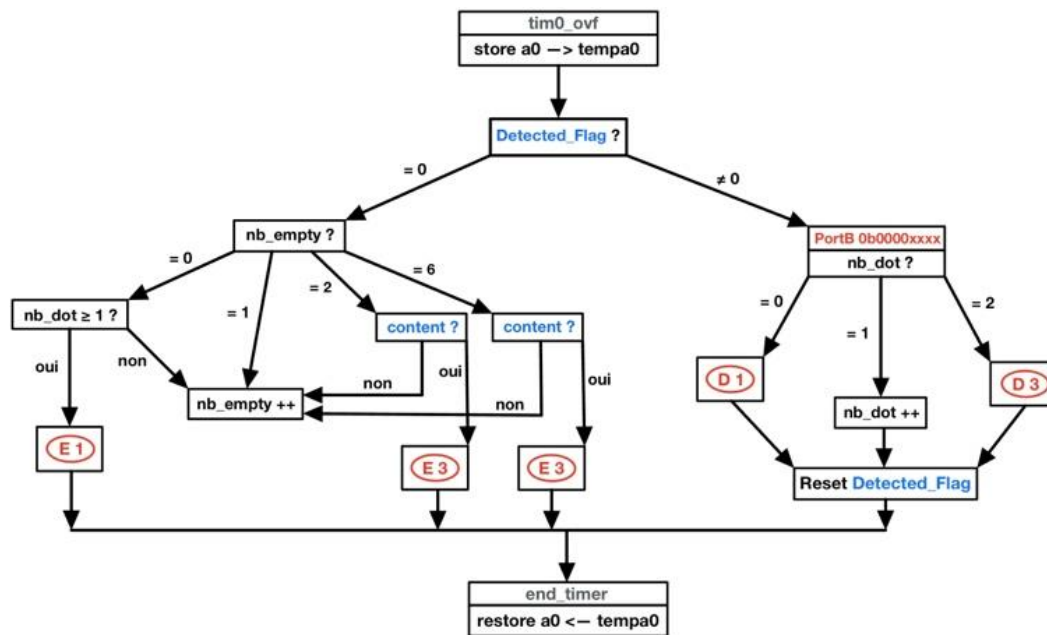
6. Références

- Schmid A. & Holzer R. (2022), Microcontrôleurs : Théorie et pratique de l'AVR. EPFL PRESS/ Presses polytechniques et universitaires romandes
- 8-bit Atmel Microcontroller with 128KBytes In-System Programmable Flash (2011). Atmel Corporation© (PDF)
- 8-bit Atmel Microcontroller with 128KBytes In-System Programmable Flash : Summary (2011). Atmel Corporation© (PDF)
- Cottenceau B. (2019), Carte ARDUINO UNO Microcontrôleur ATMega328, Polytech Angers (PDF)
- AVR Instruction Set Manual (2016). Atmel Corporation© (PDF)
- STK300 AVR Starter Kit Manual (2013). Atmel Corporation© & Kanda© (PDF)
- Atmel Studio User Guide (2016). Atmel Corporation© (PDF)

7. Annexes

Input signal															
Output LCD															C
nb_dot	1	2	3	0	1	0	1	2	3	0	1	0	0	0	0
nb_empty	0	0	0	1	0	1	0	0	0	1	0	1	2	3	4

Exemple de l'écriture de la lettre C



Pseudo-code de la routine de service d'interruption de TIM0OVF

X	Clear message
S	Save message
-	Prescaler Decrement
+	Prescaler Increment
C	Content
U	Update
D	Detected
M	Menu

Flag du FREG

	nb_dot	nb_empty	Ltr_col_bit
E1	nb_dot = 0	nb_empty ++	Ltr_col_bit ++
E3	—	nb_empty ++	Ltr_col_bit = 0
D1	nb_dot ++	nb_empty = 0	—
D3	nb_dot ++	—	—

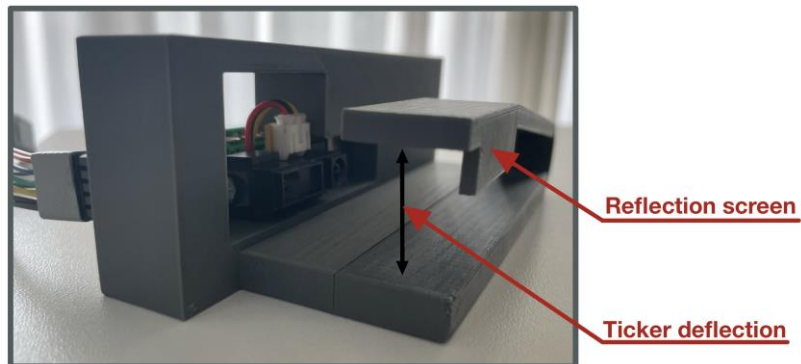
Modifications des variables de décodage en fonction du signal

7	6	5	4	3	2	1	0
X	S	—	+	C	U	D	M

Position des Flags dans le FREG

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x100	F reg	Mod presc	nb_empty	nb_dot	ltr col	ltr col bit	current letter		menu letter		scroll left	scroll right	print counter	remote command	enc old	temp0
0x110	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
0x120	L16	L17	L18	L19	L20	L21	L22	L23	L24	L25	L26	L27	L28	L29	L30	L31
0x130	E	T														
0x140	I	N	A	M												
0x150	S	D	R	G	U	K	W	O								
0x160	H	B	L	Z	F	C	P		V	X		Q		Y	J	

Répartition de nos variables dans la SRAM



Réalisation en impression 3D de la membrane jouant le rôle de téléscripteur

Vous trouverez ci-dessous l'intégralité du code source en langage assembleur :

```

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

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

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

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

```



```

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

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

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

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

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

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

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

; === STK-300 ===
.equ    LED      = PORTB    ; LEDs on STK-300
.equ    BUTTON   = PIND    ; buttons on the STK-300

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

; === module M5 (I2C/1Wire) ===
.equ    SCL       = 0    ; I2C serial clock
.equ    SDA       = 1    ; I2C serial data
.equ    DQ        = 5    ; Dallas 1Wire
                        ; master transmitter status codes, Table 88
.equ    I2CMT_START = 0x08    ; start
.equ    I2CMT_REPSTART = 0x10 ; repeated start
.equ    I2CMT_SLA_ACK = 0x18  ; slave ack
.equ    I2CMT_SLA_NOACK = 0x20 ; slave no ack

```

```

.equ  I2CMT_DATA_ACK = 0x28    ; data write, ack
.equ  I2CMT_DATA_NOACK = 0x30 ; data write, no ack
                                ; master receiver status codes, Table 89
.equ  I2CMR_SLA_ACK = 0x40    ; slave address ack
.equ  I2CMR_SLA_NACK      = 0x48 ; slave address no ack
.equ  I2CMR_DATA_ACK = 0x50    ; master data ack
.equ  I2CMR_DATA_NACK= 0x58    ; master data no ack

; === module M4 (Keyboard/Sharp/Servo) ===
.equ  KB_CLK = 0    ; PC-AT keyboard clock line
.equ  KB_DAT = 1    ; PC-AT keyboard data line
.equ  GP2_CLK      = 2    ; Sharp GP2D02 distance measuring sensor
.equ  GP2_DAT      = 3    ; Sharp GP2D02 distance measuring sensor
.equ  GP2_AVAL = 3; Shart GP2Y0A21 distance measuring sensor
.equ  SERV01 = 4    ; Futaba position servo

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

```

```

; file encoder.asm    target ATmega128L-4MHz-STK300
; purpose library angular encoder operation

; === definitions ===
.equ    ENCOD    = PORTE

encoder_left:
    ser w
    sts left_scroll, w
    clr w
    sts right_scroll, w
    ret

encoder_right:
    ser w
    sts right_scroll, w
    clr w
    sts left_scroll, w
    ret

encoder_init:
    in    w, ENCOD-1          ; make 3 lines input
    andi  w, 0b10001111
    out   ENCOD-1, w
    in    w, ENCOD            ; enable 3 internal pull-ups
    ori   w, 0b01110000
    out   ENCOD, w
    ret

read_encoder:
; a0, b0    if button=up    then increment/decrement a0
; a0, b0    if button=down then increment/decrement b0
; T         T=1 button press (transition up-down)
; Z         Z=1 button down change

    clt                                ; preclear T
    in     _w, ENCOD-2                ; read encoder port (_w=new)

    andi   _w, 0b01110000 ; mask encoder lines (A,B,I)
    lds    _u, enc_old                ; load previous value (_u=old)
    cp     _w, _u                      ; compare new<>old ?
    brne   PC+3
    clz
    ret                                ; if new=old then return (Z=0)
    sts    enc_old, _w                ; store encoder value for next time

    eor    _u, _w                     ; exclusive or detects transitions
    clz                                ; clear Z flag
    sbrc   _u, ENCOD_I                 ; transition on I (button)
    rjmp   encoder_button
    sbrc   _u, ENCOD_A                 ; return (no transition on I or A)
    ret

    sbrc   _w, ENCOD_I                 ; is the button up or down ?
    rjmp   i_down

i_up:
    sbrc   _w, ENCOD_A
    rjmp   a_rise

a_fall:
    rcall  encoder_right                ; if B=1 then increment
    sbrc   _w, ENCOD_B

```

```

        rcall encoder_left          ; if B=0 then decrement
        rjmp  i_up_done
a_rise:
        rcall encoder_right        ; if B=0 then increment
        sbrc  _w, ENCOD_B
        rcall encoder_left        ; if B=1 then decrement
i_up_done:
        clz                               ; clear Z
        ret

i_down:
        sbrc  _w, ENCOD_A
        rjmp  a_rise2
a_fall2:
        inc   b0                        ; if B=1 then increment
        sbrs  _w, ENCOD_B
        subi  b0, 2                    ; if B=0 then decrement
        rjmp  i_down_done
a_rise2:
        inc   b0                        ; if B=0 then increment
        sbrc  _w, ENCOD_B
        subi  b0, 2                    ; if B=1 then decrement
i_down_done:
        sez                               ; set Z
        ret

encoder_button:
        sbrc  _w, ENCOD_I
        rjmp  i_rise
i_fall:
        set                               ; set T=1 to indicate button press
        ret
i_rise:
        ret

.macro CYCLIC ;reg,lo,hi
        cpi   @0, @1-1
        brne  PC+2
        ldi   @0, @2
        cpi   @0, @2+1
        brne  PC+2
        ldi   @0, @1
.endmacro

```

```

; 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)
    JNB    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)
    JNB    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, 0b00000010      ; 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_1f:

```

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

```

```

; file:      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
; =====

;==== MY MACROS =====

;==== FREG =====
;==== FREG_SET
;====
macro FREG_SET
    lds _w, FREG          ; set FREG flag @0
    ori _w, (1<<@0)
    sts FREG, _w
.endmacro

;==== FREG_CLR
;====
macro FREG_CLR
    lds _w, FREG          ; clear FREG flag @0
    andi _w, ~(1<<@0)
    sts FREG, _w
.endmacro

;==== LETTERS =====
; SET CURRENT LETTER POINTER
;====
macro SET_CRNT_LTR
    ldi w, low(@0)
    sts current_ltr, w
    ldi w, high(@0)
    sts current_ltr+1, w
.endmacro

; SET PRINT LETTER POINTERS
;====
macro SET_MENU_PRNT_LTR
    ldi w, low(@0)
    sts menu_print_ltr, w
    ldi w, high(@0)
    sts menu_print_ltr+1, w
.endmacro

;====
macro SET_ALL_LTRS ; Write @0 in all @1 first letters and sets current_ltr to 100
    LDIZ 100
    ldi a0, @1
    clear_loop:
        st z+, @0
        dec a0
        cpi a0, 0
        brne clear_loop
    LDIZ 100
    STSZ current_ltr
.endmacro

;==== PRINT LETTERS LCD =====
; Print NBL_PRINT letters, addresses to print [ @0 --> @0 + NBL_PRINT ]
;====
macro LCD_PRNT_LETTERS
    clr w
    sts print_ctn, w
    rcall LCD_home
    LDSY @0

    print_loop:
        ld a0, y+

```

```

        PRINTF LCD
        .db FCHAR, 18, 0
        INCS print_ctn
        cpi w, NBL_PRINT
        brne print_loop

        PRINTF LCD
        .db LF, 0
    .endmacro

;==== READ/WRITE TABLE =====
.macro TABLE_WRITE_IN ;@0: table addr      reads value from w and increments it
    LDIZ @0
    st z, r16
    inc r16
.endmacro

; BRANCH IF NOT LAST LETTER
.macro BNLAST_LTR
    LDSZ current_ltr
    ldi a0, low(END_LETTERS)
    cp z1, a0                                ; check last for letter
    brne @0
    ldi a0, high(END_LETTERS)
    cp zh, a0
    brne @0
.endmacro
;=====

; --- loading an immediate into a pointer XYZ,SP ---
.macro          LDIX      ; sram
    ldi    x1, low(@0)
    ldi    xh,high(@0)
.endmacro

.macro          LDIY      ; sram
    ldi    y1, low(@0)
    ldi    yh,high(@0)
.endmacro

.macro          LDIZ      ; sram
    ldi    z1, low(@0)
    ldi    zh,high(@0)

    .endmacro

.macro LDZD      ; sram, reg    ; sram+reg -> Z
    mov    z1,@1
    clr    zh
    subi   z1, 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    x1,@0
    lds    xh,@0+1
    .endmacro

.macro          LDSY      ; sram
    lds    y1,@0

```



```

        lds    yh,@0+1
    .endmacro
.macro    LDSZ    ; sram
    lds    zl,@0
    lds    zh,@0+1
    .endmacro
.macro    STSX    ; sram
    sts    @0, x1
    sts    @0+1,xh
    .endmacro
.macro    STSY    ; sram
    sts    @0, y1
    sts    @0+1,yh
    .endmacro
.macro    STSZ    ; sram
    sts    @0, z1
    sts    @0+1,zh
    .endmacro

; --- push/pop pointer XYZ ---
.macro    PUSHX    ; push X
    push    x1
    push    xh
    .endmacro
.macro    POPX    ; pop X
    pop     xh
    pop     x1
    .endmacro

.macro    PUSHY    ; push Y
    push    y1
    push    yh
    .endmacro
.macro    POPY    ; pop Y
    pop     yh
    pop     y1
    .endmacro

.macro    PUSHZ    ; push Z
    push    z1
    push    zh
    .endmacro
.macro    POPZ    ; pop Z
    pop     zh
    pop     z1
    .endmacro

; --- multiply/divide Z ---
.macro    MUL2Z    ; multiply Z by 2
    lsl     z1
    rol     zh
    .endmacro
.macro    DIV2Z    ; divide Z by 2
    lsr     zh
    ror     z1
    .endmacro

; --- add register to pointer XYZ ---
.macro    ADDX    ; reg    ; x <- y+reg
    add     x1,@0
    brcc    PC+2
    subi    xh,-1    ; add carry
    .endmacro

```

```

    .macro ADDY      ;reg          ; y <- y+reg
        add        y1,@0
        brcc       PC+2
        subi       yh,-1          ; add carry
    .endmacro

    .macro ADDZ      ;reg          ; z <- z+reg
        add        z1,@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    z1, 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     z1,@2              ; move index into z
    clr     zh
    lsl     z1                  ; multiply by 2
    rol     zh
    subi    z1, 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    z1,1                ; 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     z1,@4              ; move index into z
    clr     zh
    lsl     z1                  ; multiply by 2
    rol     zh
    lsl     z1                  ; multiply by 2
    rol     zh
    subi    z1, low(-2*@5)      ; add base address of table
    sbci    zh,high(-2*@5)
    lpm
    mov     @1,r0                ; load high word LSB
    adiw    z1,1
    lpm
    mov     @0,r0                ; load high word MSB
    adiw    z1,1
    lpm
    mov     @3,r0                ; load low word LSB
    adiw    z1,1
    lpm
    mov     @2,r0                ; load low word MSB
    .endmacro

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

; --- branch table ---

```

```

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

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

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

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

.macro DIV4 ; reg
    lsr    @0
    lsr    @0
.endmacro

.macro DIV8 ; reg
    lsr    @0
    lsr    @0
    lsr    @0
.endmacro

.macro MUL2 ; reg
    lsl    @0
.endmacro

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

.macro MUL8 ; reg

```

```

        lsl    @0
        lsl    @0
        lsl    @0
    .endmacro

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

; --- immediate ops with r0..r15 ---
.macro _ADDI
    ldi    w,@1
    add    @0,w
.endmacro

.macro _ADCI
    ldi    w,@1
    adc    @0,w
.endmacro

.macro _SUBI
    ldi    w,@1
    sub    @0,w
.endmacro

.macro _SBCI
    ldi    w,@1
    sbc    @0,w
.endmacro

.macro _ANDI
    ldi    w,@1
    and    @0,w
.endmacro

.macro _ORI
    ldi    w,@1
    or     @0,w
.endmacro

.macro _EORI
    ldi    w,@1
    eor    @0,w
.endmacro

.macro _SBR
    ldi    w,@1
    or     @0,w
.endmacro

.macro _CBR
    ldi    w,~@1
    and    @0,w
.endmacro

.macro _CPI
    ldi    w,@1
    cp     @0,w
.endmacro

.macro _LDI
    ldi    w,@1
    mov    @0,w
.endmacro

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

.macro _CBI
    in     w,@0

```



```

andi    w, ~(1<<@1)
out     @0, w
.endmacro

```

; --- extending branch distance to +/-2k ---

```

.macro _BREQ
    brne    PC+2
    rjmp    @0
.endmacro

.macro _BRNE
    breq     PC+2
    rjmp     @0
.endmacro

.macro _BRCS
    brcc     PC+2
    rjmp     @0
.endmacro

.macro _BRCC
    brcs     PC+2
    rjmp     @0
.endmacro

.macro _BRSH
    brlo     PC+2
    rjmp     @0
.endmacro

.macro _BRLO
    brsh     PC+2
    rjmp     @0
.endmacro

.macro _BRMI
    brpl     PC+2
    rjmp     @0
.endmacro

.macro _BRPL
    brmi     PC+2
    rjmp     @0
.endmacro

.macro _BRGE
    brlt     PC+2
    rjmp     @0
.endmacro

.macro _BRLT
    brge     PC+2
    rjmp     @0
.endmacro

.macro _BRHS
    brhc     PC+2
    rjmp     @0
.endmacro

.macro _BRHC
    brhs     PC+2
    rjmp     @0
.endmacro

.macro _BRTS
    brtc     PC+2
    rjmp     @0
.endmacro

.macro _BRTC
    brts     PC+2
    rjmp     @0
.endmacro

.macro _BRVS
    brvc     PC+2

```

```

        rjmp    @0
    .endmacro

.macro _BRVC
    brvs    PC+2
    rjmp    @0
.endmacro

.macro _BRIE
    brid    PC+2
    rjmp    @0
.endmacro

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

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

; --- moving bits ---
.macro MOVB    ; reg1,b1, reg2,b2    ; reg1,bit1 <- reg2,bit2
    bst    @2,@3
    bld    @0,@1
.endmacro

.macro OUTB    ; port1,b1, reg2,b2    ; port1,bit1 <- reg2,bit2
    sbrs   @2,@3
    cbi    @0,@1
    sbrc   @2,@3
    sbi    @0,@1
.endmacro

.macro INB     ; reg1,b1, port2,b2    ; reg1,bit1 <- port2,bit2
    sbis   @2,@3
    cbr    @0,1<<@1
    sbic   @2,@3
    sbr    @0,1<<@1
.endmacro

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

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

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

.macro B2C    ; reg,b                ; bit to carry
    sbrc    @0,@1
    sec
    sbrs    @0,@1
    clc
.endmacro

.macro C2B    ; reg,b                ; carry to bit
    brcc    PC+2
    sbr     @0,(1<<@1)

```

```

        brcs    PC+2
        cbr     @0,(1<<@1)
    .endmacro

.macro P2C      ; port,b          ; port to carry
    sbic     @0,@1
    sec
    sbis     @0,@1
    clc
    .endmacro

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

; --- inverting bits ---
.macro INVB     ; reg,bit         ; inverse reg,bit
    ldi      w,(1<<@1)
    eor      @0,w
    .endmacro

.macro INVP     ; port,bit        ; inverse port,bit
    sbis     @0,@1
    rjmp     PC+3
    cbi      @0,@1
    rjmp     PC+2
    sbi      @0,@1
    .endmacro

.macro INVC     ; inverse carry
    brcs     PC+3
    sec
    rjmp     PC+2
    clc
    .endmacro

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

; --- logical operations with masks ---
.macro MOVMSK   ; reg1,reg2,mask  ; reg1 <- reg2 (mask)
    ldi      w,~@2
    and      @0,w
    ldi      w,@2
    and      @1,w
    or       @0,@1
    .endmacro

.macro ANDMSK   ; reg1,reg2,mask  ; reg1 <- reg1 AND reg2 (mask)
    mov      w,@1

```

```

        ori    w,~@2
        and    @0,w
    .endmacro

.macro ORMSK    ; reg1,reg2,mask    ; reg1 <- reg1 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   z1,1
        lpm
        mov    @2,r0
        adiw   z1,1
        lpm
        mov    @1,r0
        adiw   z1,1
        lpm
        mov    @0,r0
        adiw   z1,1
    .endmacro

    .macro LPM3

```

```

    lpm
    mov    @2,r0
    adiw   z1,1
    lpm
    mov    @1,r0
    adiw   z1,1
    lpm
    mov    @0,r0
    adiw   z1,1
    .endmacro

.macro LPM2
    lpm
    mov    @1,r0
    adiw   z1,1
    lpm
    mov    @0,r0
    adiw   z1,1
    .endmacro

.macro LPM1
    lpm
    mov    @0,r0
    adiw   z1,1
    .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
    nor    @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
    nor    @0
    nor    @1
    nor    @2
    nor    @3
    nor    @4
    nor    @5
    nor    @6
```



```

        nor    @7
    .endmacro
.macro ROR7
    nor    @0
    nor    @1
    nor    @2
    nor    @3
    nor    @4
    nor    @5
    nor    @6
    .endmacro
.macro ROR6
    nor    @0
    nor    @1
    nor    @2
    nor    @3
    nor    @4
    nor    @5
    .endmacro
.macro ROR5
    nor    @0
    nor    @1
    nor    @2
    nor    @3
    nor    @4
    .endmacro
.macro ROR4
    nor    @0
    nor    @1
    nor    @2
    nor    @3
    .endmacro
.macro ROR3
    nor    @0
    nor    @1
    nor    @2
    .endmacro
.macro ROR2
    nor    @0
    nor    @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 MOV3    ; 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 MOV2    ; addr0,addr1 ; [addr0] <-- [addr1]
    lds     w,@1
    sts     @0,w
    lds     w,@1+1
    sts     @0+1,w
.endmacro

.macro MOV     ; 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     x1, low(@1)
    ldi     xh,high(@1)
    rcall   @0
.endmacro

.macro CXY     ; subroutine,x,y
    ldi     x1, low(@1)
    ldi     xh,high(@1)
    ldi     y1, low(@2)
    ldi     yh,high(@2)
    rcall   @0
.endmacro

.macro CXZ     ; subroutine,x,z
    ldi     x1, low(@1)
    ldi     xh,high(@1)
    ldi     z1, low(@2)
    ldi     zh,high(@2)
    rcall   @0
.endmacro

.macro CXYZ    ; subroutine,x,y,z
    ldi     x1, low(@1)
    ldi     xh,high(@1)
    ldi     y1, low(@2)
    ldi     yh,high(@2)
    ldi     z1, 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

```

```

;
; main.asm
;
; Created: 19/05/2023 15:54:57
; Author : romai
;

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

;=== CONSTANTS =====
.equ MENU_REFRESH_RATE_MS = 10

.equ INIT_PRESCALER = 4
.equ MIN_PRESCALER = 1
.equ MAX_PRESCALER = 7
.equ EMPTY_BIT_LIMIT = 3

.equ DOT = 46
.equ DASH = 95

.equ MAX_SYMBOLS = 4
.equ NBL_PRINT = 0x10
.equ NBL_LETTERS = 0x20
.equ NB_SHORT = 1
.equ NB_DASH = 3
.equ NB_NEXT_SYMBOL = 1
.equ NB_NEXT_LETTER = 3
.equ NB_NEXT_WORD = 7

;=== REMOTE =====
.equ SIGNAL_POWER = 0x0C
.equ SIGNAL_MUTE = 0x0D

.equ SIGNAL_PLUS = 0x10
.equ SIGNAL_MINUS = 0x11

.equ SIGNAL_UP = 0x20
.equ SIGNAL_DOWN = 0x21

.equ SIGNAL_1 = 0x01
.equ SIGNAL_2 = 0x02
.equ SIGNAL_3 = 0x03

;=== FREG FLAGS =====
.equ MENU_FLAG = 0
.equ DETECTED_FLAG = 1
.equ UPDATE_FLAG = 2
.equ CONTENT_FLAG = 3
.equ INC_PRESC_FLAG = 4
.equ DEC_PRESC_FLAG = 5
.equ SAVE_MSG_FLAG = 6
.equ CLR_MSG_FLAG = 7

.cseg
.org 0
    jmp reset

.org INT7addr
    jmp int7_isr

.org OVF0addr
    jmp tim0_ovf

```

```

.org ADCCaddr
    jmp ADCCaddr_sra

.org 0x30

int7_isr:
    in _sreg, SREG

    rcall read_remote          ; reading remote command
    lds _w, remote_command

    check_signal_up:
    cpi _w, SIGNAL_UP
    brne check_signal_down
    FREG_SET MENU_FLAG        ; set menu flag
    rjmp int7_end

    check_signal_down:
    cpi _w, SIGNAL_DOWN
    brne check_signal_plus
    FREG_CLR MENU_FLAG        ; clear menu flag
    rjmp int7_end

    check_int7_exit:
    lds _w, FREG
    sbrs _w, MENU_FLAG
    rjmp int7_end

    check_signal_plus:
    cpi _w, SIGNAL_PLUS
    brne check_signal_min
    FREG_SET INC_PRESC_FLAG    ; set prescaler increment flag
    rjmp int7_end

    check_signal_min:
    cpi _w, SIGNAL_MINUS
    brne check_signal_mute
    FREG_SET DEC_PRESC_FLAG    ; set prescaler decrement flag
    rjmp int7_end

    check_signal_mute:
    cpi _w, SIGNAL_MUTE
    brne int7_end
    FREG_SET CLR_MSG_FLAG      ; set message clear flag

    int7_end:
    out SREG, _sreg
    reti

tim0_ovf:
    in _sreg, SREG              ; store SREG

    ; Set update flag
    FREG_SET UPDATE_FLAG

    ; save a0 in tmp0
    sts tmp0, a0

    lds _w, FREG                ; check detection
    sbrs _w, DETECTED_FLAG      ; detection->new_dot no_detection->new_empty
    rjmp new_dot

```

```

new_empty:
lds _w, nb_empty
cpi _w, NB_NEXT_SYMBOL - 1
brne check_next_letter

; 1 EMPTY
lds _w, nb_dot ; - check nb dot
cpi _w, 1
brsh PC+2
rjmp inc_nb_empty
clr _w ; - clear nb_dot
sts nb_dot, _w
lds _w, ltr_col_bit
cpi _w, MAX_SYMBOLS ; - check max symbols
brlo inc_ltr_col_bit
rjmp inc_nb_empty
inc_ltr_col_bit: ; - increment table column bit to modify
inc _w
sts ltr_col_bit, _w
rjmp inc_nb_empty

check_next_letter:
cpi _w, 1
brne PC+2
rjmp inc_nb_empty
cpi _w, NB_NEXT_LETTER - 1
breq empty_3
rjmp check_next_word
empty_3: ; 3 EMPTY
lds _w, FREG ; decode letter only if there is content
sbrs _w, CONTENT_FLAG
rjmp inc_nb_empty
lds _w, ltr_col_bit
dec _w
ldi a0, TABLE_WIDTH
mul _w, a0 ; table row = (letter col bit - 1) *
TABLE_WIDTH (result goes in r0)
mov _w, r0
lds a0, ltr_col ; table col = letter column
add _w, a0 ; table offset = table row + table column
ldi z1, low(TABLE_START)
ldi zh, high(TABLE_START)
add z1, _w ; add offset to table start
ld _w, z ; get table value

BNLAST_LTR store_ltr ; branch if not last letter
sts tmp0, _w ; store letter to write in tmp0
ldi _w, SPACE ; if so clear l0-15 and write in l00
SET_ALL_LTRS _w, NBL_PRINT

LDSZ current_ltr
lds _w, tmp0
st z+, _w
STSZ current_ltr
rjmp reset_ltr

store_ltr:
st z+, _w
STSZ current_ltr ; store current letter address from pointer z

reset_ltr: ; reset table column bit letter row and letter
column
clr _w

```



```

sts ltr_col, _w
sts ltr_col_bit, _w

rjmp inc_nb_empty

check_next_word:
lds _w, FREG
sbrs _w, CONTENT_FLAG
rjmp inc_nb_empty
lds _w, nb_empty
cpi _w, NB_NEXT_WORD - 1
brlo inc_nb_empty
breq empty_7
rjmp end_timer
empty_7:                                     ; 7 EMPTY
BNLAST_LTR write_space
ldi _w, SPACE                               ; if last letter reset all letters
SET_ALL_LTRS _w, NBL_PRINT
rjmp inc_nb_empty

write_space:
LDSZ current_ltr
ldi _w, SPACE
st z+, _w
STSZ current_ltr

inc_nb_empty:
lds _w, nb_empty                           ; inc nb_empty
sbrs _w, EMPTY_BIT_LIMIT
inc _w
sts nb_empty, _w
rjmp end_timer

new_dot:
lds _w, nb_dot                             ; load nb_dot
cpi _w, 0                                   ; check first dot:
brne check_dash
; 1 DOT
FREG_SET CONTENT_FLAG

clr _w                                     ; - clear nb_empty
sts nb_empty, _w
rjmp inc_nb_dot

check_dash:
cpi _w, 1
breq inc_nb_dot
cpi _w, NB_DASH-1
brne reset_detected
; 3 DOT
lds _w, ltr_col_bit                         ; if 3rd dot set table column bit
cpi _w, 4
brlo PC+2
rjmp inc_nb_dot
ldi a0, 0x01
b_loop:
    cpi _w, 0
    breq end_loop
    lsl a0
    dec _w
    rjmp b_loop
end_loop:
lds _w, ltr_col

```

```

or _w, a0
sts ltr_col, _w

inc_nb_dot:                                ; increment nb_dot
lds _w, nb_dot
inc _w
sts nb_dot, _w

reset_detected:
FREG_CLR DETECTED_FLAG

end_timer:
; clear tmp values
clr _w
sts tmp0, _w

; restore register a
lds a0, tmp0
out SREG, _sreg                        ; restore SREG
reti

```

```

ADCCaddr_sra :
ldi r23,0x01
reti

```

```

#include "variables_definition.asm"
#include "variables_control.asm"
#include "encoder.asm"
#include "sharp.asm"
#include "remote.asm"
#include "table.asm"

```

```

reset :
LDSP    RAMEND
OUTI    DDRB,0xff

;turn off leds
OUTI    LED, 0xff

sei
rcall   LCD_init
rcall   encoder_init
OUTI    ADCSR,(1<<ADEN) + (1<<ADIE) + 6
OUTI    ADMUX,3

sei
;config INT7
OUTI    EIMSK, (1<<7)
in w, EICRB
ori w, (1<<ISC71)
andi w, ~(1<<ISC70)
out EICRB, w

;config timer0 overflow
OUTI    ASSR,(1<<AS0)
OUTI    TCCR0, INIT_PRESCALER

rcall   LCD_clear
rcall   LCD_home

rcall   variables_init

rjmp    menu

```

```

.include "lcd.asm"
.include "printf.asm"

clr_a:
    clr a0
    clr a1
    clr a2
    clr a3
    ret

menu:
    in w, TIMSK
    andi w, 0b11111110
    out TIMSK, w

    rcall LCD_clear
    rcall LCD_home

    ;set first letter to print to l00
    SET_MENU_PRNT_LTR l00

    ; check for message scroll flag from encoder
    check_encoder_scroll:
    OUTI LED, 0xff
    rcall read_encoder

    check_scroll_left:
    lds w, left_scroll
    cpi w, 0xff
    brne check_scroll_right
    ; check left limit
    LDSZ menu_print_ltr
    cpi zl, low(l16)
    brne mp_next
    cpi zh, high(l16)
    brne mp_next
    rjmp update_prescaler
    ; point menu print to next letter
    mp_next:
    OUTI LED, 0b00001111
    LDSZ menu_print_ltr
    adiw z, 1
    STSZ menu_print_ltr
    clr w
    sts left_scroll, w
    rjmp update_prescaler

    check_scroll_right:
    lds w, right_scroll
    cpi w, 0xff
    brne update_prescaler
    LDSZ menu_print_ltr
    cpi zl, low(l00)
    brne mp_prev
    cpi zh, high(l00)
    brne mp_prev
    rjmp update_prescaler
    ; point menu print to previous letter
    mp_prev:
    OUTI LED, 0b11110000
    LDSZ menu_print_ltr
    sbiw z, 1

```

```

STSZ menu_print_ltr
clr w
sts right_scroll, w

update_prescaler:
/*      Read prescaler change commands      */
lds a0, FREG

check_presc_inc:
sbrs a0, INC_PRESC_FLAG
rjmp check_presc_dec
lds w, mod_prescaler
cpi w, MAX_PRESCALER
brne inc_presc
FREG_CLR INC_PRESC_FLAG
rjmp check_msg_mute
inc_presc:
INCS mod_prescaler
FREG_CLR INC_PRESC_FLAG
rjmp set_new_presc

check_presc_dec:
sbrs a0, DEC_PRESC_FLAG
rjmp check_msg_mute
lds w, mod_prescaler
cpi w, MIN_PRESCALER
brne dec_presc
FREG_CLR DEC_PRESC_FLAG
rjmp check_msg_mute
dec_presc:
DECS mod_prescaler
FREG_CLR DEC_PRESC_FLAG

set_new_presc:
lds w, mod_prescaler
out TCCR0, w

check_msg_mute:
lds w, FREG
sbrs w, CLR_MSG_FLAG      ; check CLR_MESSAGE flag
rjmp display_menu        ; if flag not set jump to display menu
ldi w, SPACE
SET_ALL_LTRS w, NBL_LETTERS ; if flag set clear l00-l15
FREG_CLR CLR_MSG_FLAG      ; reset flag

display_menu:
LCD_PRNT_LETTERS menu_print_ltr

rcall clr_a
lds a0, mod_prescaler
LDIZ menu_print_ltr
ld a1, z
subi a1, 0x10

PRINTF LCD
.db CR, CR, "presc:", FDEC, a, "      s:", FDEC, 19, " ", 0

check_menu_exit:
WAIT_MS MENU_REFRESH_RATE_MS
lds w, FREG
sbrc w, MENU_FLAG
rjmp check_encoder_scroll

```

```

in w, TIMSK
ori w, (1<<TOIE0)
out TIMSK, w
clr w
sts nb_dot, w
sts nb_empty, w
sts ltr_col, w
sts ltr_col_bit, w
FREG_CLR CONTENT_FLAG

```

```

rcall LCD_clear
rcall LCD_home
rjmp reading

```

check_menu:

```

;check menu control variable
lds w, FREG
sbrc w, MENU_FLAG
rjmp menu
rjmp reading

```

reading:

```

rcall read_sharp ; reading distance sensor

```

print_morse:

```

rcall clr_a
lds a0, l00
lds a1, l01
lds a2, l02
lds a3, l03

```

```

;Only update LCD if UPDATE_FLAG set
lds w, FREG
sbrs w, UPDATE_FLAG
rjmp reading_end

```

display_letters:

```

LCD_PRNT_LETTERS menu_print_ltr

```

```

;display debug variables

```

display_debug:

```

rcall clr_a
lds a0, FREG
lds a1, nb_dot
lds a2, nb_empty
PRINTF LCD
.db "d:", FDEC, 19, " e:", FDEC, 20, 0

```

reset_update_flag:

```

FREG_CLR UPDATE_FLAG

```

reading_end:

```

rjmp check_menu

```

```

; 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/Os, 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 fprintf 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), unsigned
; "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    z1,1          ; 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    z1,1          ; 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     x1,r0         ; load x-pointer with SRAM address

```



```

        cpi    x1,0x60
        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   z1,1      ; 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,z1
        MOV3   a0,zh,z1, w,e1,e0
        icall  ; indirect call to "putc"
        POP3   a0,zh,z1
        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   z1,1         ; increment char pointer
    lpm
    mov    b1,r0        ; load ii.ff format
    adiw   z1,1         ; 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 unsigned, 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     a3           ; 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=remainder
        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      dividend (argument to divide)
;      b0      divider
; out  a0..a3      result
;      d0      remainder
;
_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

```

```

/*
 * remote.asm
 *
 * Created: 27-05-23 22:43:06
 * Author: romai
 */
; file ir_rc5.asm target ATmega128L-4MHz-STK300
; purpose IR sensor decoding RC5 format

.equ T1 = 1800 ; bit period T1 = 1800 usec

read_remote:
    CLR2 b1,b0 ; clear 2-byte register
    ldi b2,14 ; load bit-counter
    WAIT_US (T1/4) ; wait a quarter period
read_loop:
    P2C PINE,IR ; move Pin to Carry (P2C)
    ROL2 b1,b0 ; roll carry into 2-byte reg
    WAIT_US (T1-4) ; wait bit period (- compensation)
    DJNZ b2,read_loop ; Decrement and Jump if Not Zero
    com b0 ; complement b0

    sts remote_command, b0 ; store command
    ret

```

```

/*
 * sharp.asm
 *
 * Created: 19/05/2023 16:13:56
 * Author: romai
 */

```

```

read_sharp:
    clr r23
    sbi ADCSR,ADSC
    WB0 r23,0
    in a0,ADCL
    in a1,ADCH
    cpi a1, 3
    brlo read_end
isthere:
    lds w, FREG
    ori w, (1<<DETECTED_FLAG)
    sts FREG, w
read_end:
    clr w

    ret

```

```

/*
 * table.asm
 *
 * Created: 27-05-23 20:16:11
 * Author: romai
 */
.equ TABLE_START = 0x130
.equ TABLE_WIDTH = 0x10

.equ END_LETTERS = TABLE_START

;===== [ ALPHABET TABLE ] =====
; 1 SYMBOL LETTERS (ROW 0)
.equ Eaddr = TABLE_START
.equ Taddr = TABLE_START + 1

; 2 SYMBOLS LETTERS (ROW 1)
.equ Iaddr = TABLE_START + TABLE_WIDTH
.equ Naddr = TABLE_START + TABLE_WIDTH + 1
.equ Aaddr = TABLE_START + TABLE_WIDTH + 2
.equ Maddr = TABLE_START + TABLE_WIDTH + 3

; 3 SYMBOLS LETTERS (ROW 2)
.equ Saddr = TABLE_START + (2*TABLE_WIDTH)
.equ Daddr = TABLE_START + (2*TABLE_WIDTH) + 1
.equ Raddr = TABLE_START + (2*TABLE_WIDTH) + 2
.equ Gaddr = TABLE_START + (2*TABLE_WIDTH) + 3
.equ Uaddr = TABLE_START + (2*TABLE_WIDTH) + 4
.equ Kaddr = TABLE_START + (2*TABLE_WIDTH) + 5
.equ Waddr = TABLE_START + (2*TABLE_WIDTH) + 6
.equ Oaddr = TABLE_START + (2*TABLE_WIDTH) + 7

; 4 SYMBOLS LETTERS (ROW 3)
.equ Haddr = TABLE_START + (3*TABLE_WIDTH)
.equ Baddr = TABLE_START + (3*TABLE_WIDTH) + 1
.equ Laddr = TABLE_START + (3*TABLE_WIDTH) + 2
.equ Zaddr = TABLE_START + (3*TABLE_WIDTH) + 3
.equ Faddr = TABLE_START + (3*TABLE_WIDTH) + 4
.equ Caddr = TABLE_START + (3*TABLE_WIDTH) + 5
.equ Paddr = TABLE_START + (3*TABLE_WIDTH) + 6

.equ Vaddr = TABLE_START + (3*TABLE_WIDTH) + 8
.equ Xaddr = TABLE_START + (3*TABLE_WIDTH) + 9

.equ Qaddr = TABLE_START + (3*TABLE_WIDTH) + 11

.equ Yaddr = TABLE_START + (3*TABLE_WIDTH) + 13
.equ Jaddr = TABLE_START + (3*TABLE_WIDTH) + 14

;===== [ ASCII ] =====
.equ ascii_top = 0x41

write_table:
    ldi r16, ascii_top
    TABLE_WRITE_IN Aaddr
    TABLE_WRITE_IN Baddr
    TABLE_WRITE_IN Caddr
    TABLE_WRITE_IN Daddr
    TABLE_WRITE_IN Eaddr
    TABLE_WRITE_IN Faddr
    TABLE_WRITE_IN Gaddr
    TABLE_WRITE_IN Haddr

```

```
TABLE_WRITE_IN Iaddr  
TABLE_WRITE_IN Jaddr  
TABLE_WRITE_IN Kaddr  
TABLE_WRITE_IN Laddr  
TABLE_WRITE_IN Maddr  
TABLE_WRITE_IN Naddr  
TABLE_WRITE_IN Oaddr  
TABLE_WRITE_IN Paddr  
TABLE_WRITE_IN Qaddr  
TABLE_WRITE_IN Raddr  
TABLE_WRITE_IN Saddr  
TABLE_WRITE_IN Taddr  
TABLE_WRITE_IN Uaddr  
TABLE_WRITE_IN Vaddr  
TABLE_WRITE_IN Waddr  
TABLE_WRITE_IN Xaddr  
TABLE_WRITE_IN Yaddr  
TABLE_WRITE_IN Zaddr  
ret
```



```

/*
 * variables_control.asm
 *
 * Created: 28-05-23 12:40:21
 * Author: romai
 */

;=== RESERVED MEMORY AREA IN SRAM ===
.equ RESERVED_MEMORY_AREA_START = 0x0100
.equ RESERVED_MEMORY_AREA_END   = 0x0170

.cseg
variables_init:
    ; CLEAR RESERVED MEMORY AREA (0x0100 - 0x0160)
    clr w
    LDIZ RESERVED_MEMORY_AREA_START          ; point z at sram start
(our table row 1)
    mclr_loop:                               ; memory
clear loop
    cpi z1, low(RESERVED_MEMORY_AREA_END)
    brne m_clear
    cpi zh, high(RESERVED_MEMORY_AREA_END)
    brne m_clear
    rjmp end_mclr_loop
    m_clear:
    st z+, w
    rjmp mclr_loop
end_mclr_loop:

; INIT LETTERS
ldi w, SPACE
LDIZ SRAM_START + TABLE_WIDTH          ; point z at table row 2
r2_loop:                                ; init write row2 loop
    cpi z1, low(131+1)
    brne r2_write
    cpi zh, high(131+1)
    brne r2_write
    rjmp end_r2_loop
    r2_write:
    st z+, w
    rjmp r2_loop
end_r2_loop:

; set prescaler
ldi w, INIT_PRESCALER
sts mod_prescaler, w

; set MENU and UPDATE Flag
ldi w, 0b00000101
sts FREG, w

; set current and print letters to 100
SET_CRNT_LTR 100
SET_MENU_PRNT_LTR 100

rcall write_table
ret

```

```

/*
* varuables.asm
*
* Created: 19/05/2023 17:29:51
* Author: romai
*/
.dseg
;=== FIRST TABLE ROW: 16 BYTES FOR CONTROL VARIABLES =====

;Flag Register
FREG:                .byte 1

;prescaler
mod_prescaler:       .byte 1

;morse variables
nb_empty:            .byte 1
nb_dot:               .byte 1

;letter index in table
ltr_col:              .byte 1
ltr_col_bit:         .byte 1

;variables for message print
current_ltr:          .byte 2
menu_print_ltr:       .byte 2
left_scroll:          .byte 1
right_scroll:         .byte 1
print_ctn:            .byte 1

;remote save variable
remote_command:       .byte 1

; encoder variables
enc_old:              .byte 1

;temp variables
tmp0:                 .byte 1

;=== SECOND TABLE ROW: MESSAGE LETTERS =====
l00:                  .byte 1
l01:                  .byte 1
l02:                  .byte 1
l03:                  .byte 1
l04:                  .byte 1
l05:                  .byte 1
l06:                  .byte 1
l07:                  .byte 1
l08:                  .byte 1
l09:                  .byte 1
l10:                  .byte 1
l11:                  .byte 1
l12:                  .byte 1
l13:                  .byte 1
l14:                  .byte 1
l15:                  .byte 1
l16:                  .byte 1
l17:                  .byte 1
l18:                  .byte 1
l19:                  .byte 1
l20:                  .byte 1
l21:                  .byte 1
l22:                  .byte 1

```

123:	.byte	1
124:	.byte	1
125:	.byte	1
126:	.byte	1
127:	.byte	1
128:	.byte	1
129:	.byte	1
130:	.byte	1
131:	.byte	1