Ventilateur à déclenchement automatique

Projet de Microcontrôleur: Rapport Technique

DE GUERRE Rodrigue,
DIEZ LEBOFFE Andrea

I. INTRODUCTION

La tâche principale du projet de microcontrôleur est de réaliser à partir de la carte STK300 un système embarqué au choix en utilisant obligatoirement la télécommande infrarouge IR RC5.

Le choix que nous avons fait consiste à développer un ventilateur qui s'allume en fonction d'une certaine température. On allume/éteint le ventilateur grâce à la télécommande et la température mesurée par la capteur de chaleur est affichée dans l'écran.

II. DESCRIPTION TECHNIQUE

Table 2.1: Périphériques

Périphériques	Utilisation
Affichage LCD 2x16	Affichage de la température
Capteur de température DS18B20	Détection de la température extérieure
Interface IR NEC	Permet de réceptionner les commandes provenants de la télécommande
Micro servo SG90	Permet la rotation du ventilateur
IR Remote Control	Permet de choisir la température de consigne et d'éteindre le ventilateur

Table 2.2: Interruptions

Interruptions	Rôle
remmote_INT (INT7)	Charge une valeur arbitraire sur r28 pour activer la vérification des interruptions dans d'autres modules.

Table 2.3: Ports utilisés

Ports utilisés	Périphériques connectées
Port B (carte STK300): tous les PINS	Capteur de température
Port D (carte STK300)	Moteur pas à pas
PORT E (carte STK300): PIN7	Capteur infrarouge

Le schéma de câblage se trouve en Annexe, **Figure A**

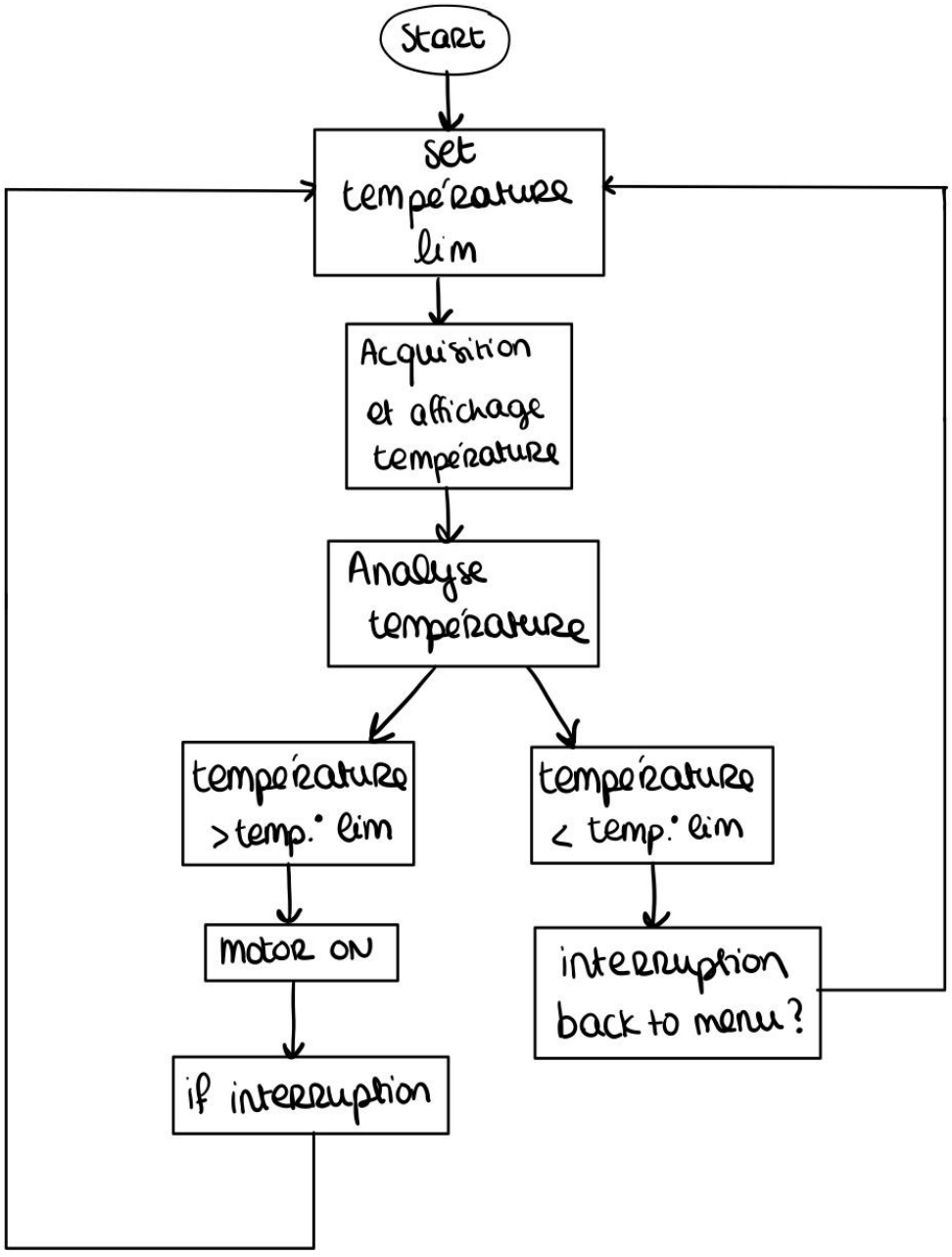
III. MISE EN PLACE

1. Brancher les périphériques sur la carte comme indiqué en Annexe
2. Allumer le dispositif

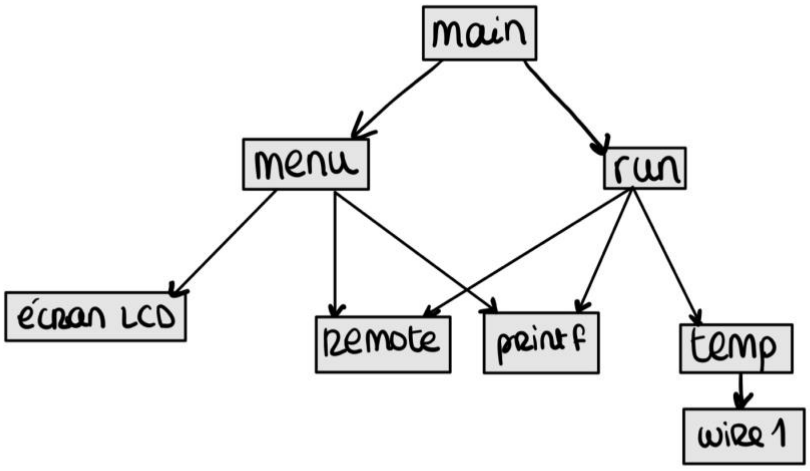
IV. MODE D'EMPLOI

- Lors du lancement du programme, l'utilisateur voit apparaître un menu lui proposant de choisir la température de consigne (température limite au-delà de laquelle le ventilateur se déclenche). Pour ce faire, il dispose de deux boutons ('+' et '-' qui chacun respectivement incrémente et décrémente de 0.5°C la température limite).
- Une fois la température choisie, l'utilisateur doit appuyer sur le bouton 'AV' afin de valider son choix
- Une fois le choix validé, le programme rentre dans le mode de comparaison de la température, et active le ventilateur si la température ambiante dépasse celle de consigne.
- Une fois le ventilateur allumé, c'est à l'utilisateur de l'éteindre en appuyant sur un bouton de la télécommande. Il revient alors au début du programme et peut ainsi choisir une nouvelle température limite.
- Lors de la comparaison de température, si l'utilisateur souhaite changer la température limite alors que le ventilateur ne s'est pas allumé (s'il s'est par exemple trompé de température limite) il peut revenir au menu en appuyant deux fois sur le bouton 'guide'.

V. FONCTIONNEMENT DU PROGRAMME



VI. ARBRE DE HIÉRARCHIE DES MODULES DU PROGRAMME



VII. PRÉSENTATION DES MODULES / ACCÈS AUX PÉRIPHÉRIQUES

Capteur de température DS18B20: “temp.asm”

Le capteur de températures est un thermomètre digital avec une interface 1-wire. Pour que les 2 bytes de température ne soient pas aléatoires, une conversion de température doit être faite après que le pulse de reset ait été émis. Une lecture du LSB et du MSB (stocké dans la mémoire temporaire scratchpad, cf Annexe Figure 1). La partie située après la virgule décimale est ignorée.

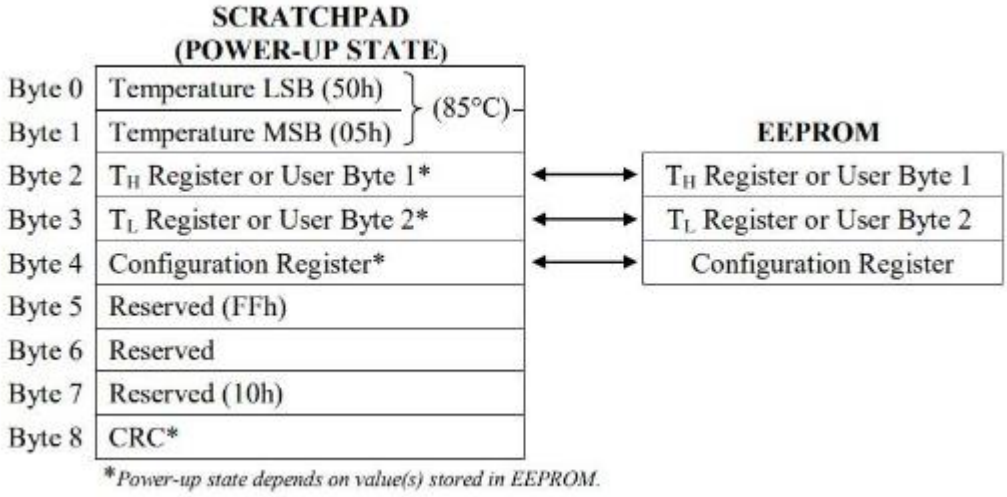


Figure 1: Organisation de la mémoire du capteur de température DS18B20

Module Moteur pas à pas: “run.asm”

Le JST X27 est un moteur pas à pas et le mouvement de rotation est généré par un circuit composé de 2 bobines, un stator et un rotor avec un aimant permanent bipolaire. Les étapes sont exécutées selon la séquence d'impulsions appliquée. Il peut amener de petites charges sur des rotations permanentes ou à une position angulaire précise.

Fichier d’interface “wire1.asm”

Le protocole 1-wire a pour particularité de n’utiliser qu’un seul fil pour communiquer et procurer une alimentation à des modules périphériques multiples. Ce fichier contient les routines réalisant les initialisations, la remise à zéro et l’écriture/ lecture de l’interface 1-wire, par software (cf Annexe Figure 2).

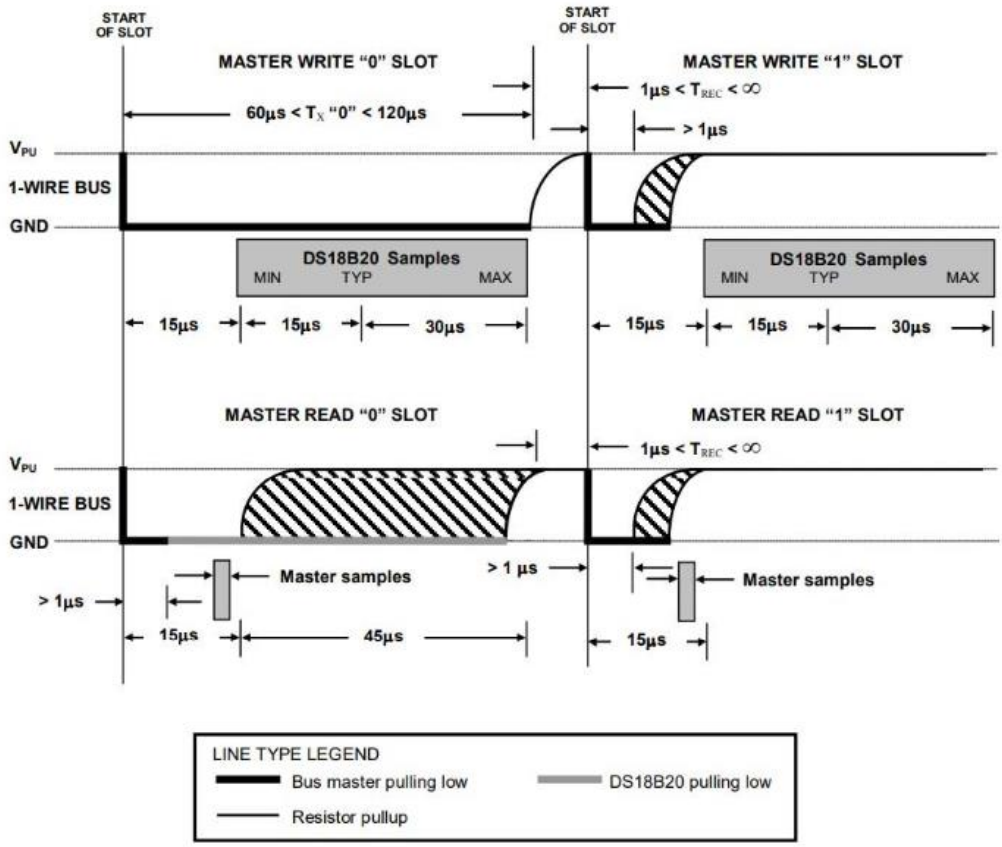


Figure 2: Diagrammes temporels du protocole 1-wire pour l'écriture (schéma du haut) et la lecture (schéma du bas)

Module LCD: "lcd.asm" et "printf.asm"

L'affichage sur le LCD est alphanumérique et se fait sur 2 lignes de 16 caractères. Il permet d'interagir avec l'utilisateur. A la mise sous tension, le module LCD exécute une initialisation à la première ligne d'affichage et place le curseur à la position 0 (home).

Module IR Remote Control "remote.asm"

La télécommande infrarouge émet au format de données RC5. La norme RC5 utilise un codage bi-phase (voir figure 3) la fréquence porteuse fixée à 36 kHz.

La transmission d'un mot de données commence par deux bits de départ suivis d'un bit de basculement. Le bit de basculement change de valeur à chaque

nouvel appui sur une touche. Les cinq bits d'adresse représentent l'adresse de l'appareil à contrôler. Les six bits de commande contiennent les informations à transmettre. Chaque bit du mot de données se compose d'une demi-période de bit sans transmission et d'une demi-période de bit avec une rafale de 32 impulsions à 36 kHz.

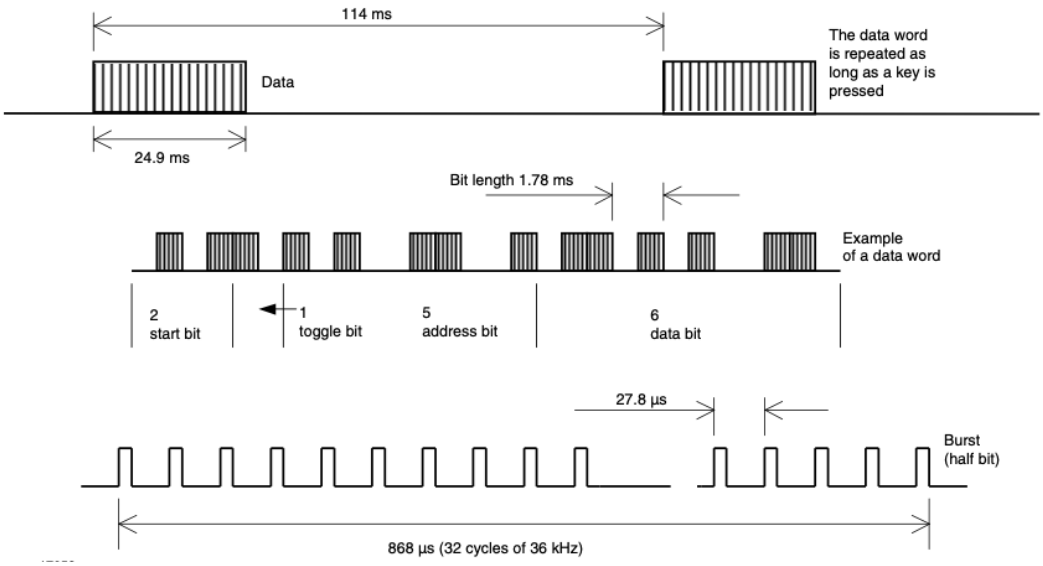


Figure 3: RC5 transmission code

VIII. AMÉLIORATION POSSIBLE ET REMARQUES

Lors du développement de notre projet, nous avions prévu un fonctionnement plus fluide et faisant plus de sens sur certains points, mais nous avons malheureusement été limité par les caractéristiques matérielles de la carte.

En particulier, les points que nous aurions aimé améliorer sont les suivants :

- Pouvoir effectuer deux boucles en même temps afin de pouvoir continuer à comparer la température en même temps que le ventilateur tourne (il aurait sans doute fallu un processeur multi-thread afin de pouvoir gérer l'activation du ventilateur sur le main thread et effectuer la comparaison de température en background, pour ensuite faire passer l'interruption sur le main thread).
- Une autre amélioration que nous aurions aimé implémenter aurait été de pouvoir détecter quel bouton de la télécommande a été pressé lors des interruptions afin de pouvoir effectuer des actions différentes.

Malheureusement, cela n'est physiquement pas possible. Nous avons donc implémenté une solution s'en rapprochant au maximum, mais qui revient à devoir appuyer deux fois sur un bouton donné.

- Enfin, nous avons aussi été limités au niveau des ports de la carte. En effet, nous nous sommes rendu compte lors du développement que les ports A et C ne fonctionnaient pas correctement avec l'utilisation simultanée de l'écran LCD et du PORTD.

IX. REFÉRENCES

1. Datasheet du capteur de températures: DS18B20
2. Datasheet du Stepper motor x27
3. Datasheet du 1-wire
4. Datasheet du IR et IR receiver

X. LISTE DES ANNEXES

1. Schéma de câblage pour notre projet
2. Table de registres utilisés dans les différents modules
3. Code source

PLAN DE CABLAGE

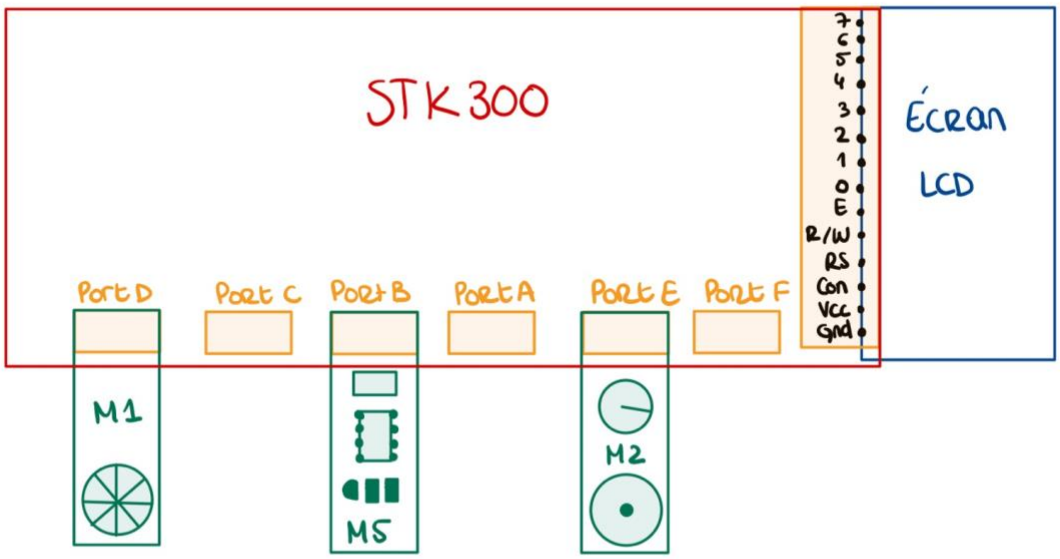


Table des registres utilisés dans les différents modules

MENU	Utilisation dans le code
a2 = r20	Registre qui stocke le LSByte de la température limite initiale choisie
a3 = r21	Registre qui stocke le MSByte de la température limite initiale choisie
b0 = r22	Copie du registre a2

b1 = r23	Copie du registre a3

REMOTE	Utilisation dans le code
b0 = r22	Registre qui stocke la valeur du signal de la télécommande
b1 = r23	Registre qui stocke la valeur du constructeur/modèle de la télécommande
b2 = r24	Compteur des 14 bits du signal RC5

RUN	Utilisation dans le code
r28	Registre stockant une valeur arbitraire attribuée lors d'une interruption
a0 = r18	Registre qui stocke le LSByte de la température ambiante mesurée
a1 = r19	Registre qui stocke le MSByte de la température ambiante mesurée
b0 = r22	Registre qui stocke le LSByte de la température limite choisie
b1 = r23	Registre qui stocke le MSByte de la température limite choisie

TEMP	Utilisation dans le code
a0 = r18	Registre qui permet la lecture des deux Bytes de la température et stocke le LSByte de la température ambiante mesurée
a1 = r19	Registre qui stocke le MSByte de la température ambiante mesurée
c0 = r8	Registre qui permet un stockage temporaire du LSByte de la température

Code Source

```
; file: main.asm       target: ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
; Purpose: Main entry point of the program

; ==== Top-level includes ===
.include "m128def.inc"
.include "definitions.asm"
.include "macros.asm"

; ==== Interrupt vector table ====

.org 0x0000
   rjmp reset

.org INT7addr
   rjmp remote_INT

; ==== Device reset ====
reset:
   LDSP     RAMEND                    ; load stack pointer SP
   OUTI     EIMSK,(1<<7)              ; enable INT7
   OUTI     EICRB,(0<<ISC71)+(1<<ISC70)    ; Interupt on any edge (rising or falling)
   OUTI     DDRD,0x0f                 ; make motor port output
   sei                                ; set global interrupt
   rcall    LCD_init                  ; initialize LCD
   rcall    wire1_init                ; initialize 1-wire(R) interface
   rjmp     main                      ; jump to main

; ====== Programm Constants ======
.equ     T1       =  1778            ; bit period T1 = 1778 usec
.equ     T_MOT    =  1000            ; waiting period in micro-seconds
.equ     port_mot =  PORTD           ; port to which motor is connected
.equ     menuGuide =  0b11011101
```

```
.equ       increaseVol =   0b11101111
.equ       decreaseVol =   0b11101110
.equ       AV_confirm  =   0b11000111

; ==== Interruptions routines ====

remote_INT:
    ldi        r28,2                    ; load arbitrary value to r28 to enable interrupts check in other modules
    reti

; ==== Modules Imports ====
.include "drivers/lcd.asm"              ; include LCD driver routines
.include "drivers/wire1.asm"            ; include Dallas 1-wire(R) routines
.include "lib/printf.asm"               ; include formatted printing routines

.include "menu.asm"
.include "run.asm"
.include "remote.asm"
.include "temp.asm"


; ===== Main Entry Point =====
main:
    cli                             ; interrupts are already disabled locally in the modules but this allows to clearly identify it
    clr        r28                    ; And adds 'symmetry' with the below sei
    rcall      menu
    rcall      LCD_clear
    sei
    jmp        run
```

```
; file: menu.asm        target: ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
```

```
; Purpose: Selection of the limit temperature

menu:
   rcall      LCD_clear
   rcall      LCD_home
   PRINTF     LCD
   .db "Choix de TempLim",CR,0

   WAIT_MS    2000
   ldi        a2,0b10110000      ; set intinal TempLim value to 27°C (a2=TempLim LSByte)
   ldi        a3,0b00000001      ; TempLim MSByte
   MOV2       b0,b1, a2,a3
   rcall      LCD_clear
   rcall      LCD_home
   PRINTF     LCD
   .db "TempLim=",FFRAC2+FSIGN,b,4,$42,"C ",CR,0
_get_ir_value:
   rcall      remote             ; ir_detect
   WAIT_MS    100                ; Wait 100ms for calibration purposes (enables the user to inc/dec value by 0.5°C)

_confirm:
   cpi        b0,AV_confirm      ; check if temp is validated (button 'AV' is pressed)
   breq       _temp_chosen       ; ret
_clear_CFlag:
   in         w,SREG
   andi       w,0b11111110
   out        SREG,w
_inc:
   cpi        b0,increaseVol     ; check if button '+' is pressed
   brne       _dec
   ldi        r16,0b00001000
   add        a2,r16             ; add '0.5'
   brcc       _display_temp
   cpi        a3,0b00000011      ; high temp lim is set to 48°C
   brsh       PC+3
```

```
    ldi         r16,0b00000000
    adc         a3,r16
    rjmp        _display_temp
    ldi         a2,0b11110000       ; Set max value to 48 or 48.5 ?   ; 0b11111000
    rjmp        _display_temp
_dec:
    cpi         b0,decreaseVol      ; check if button '-' is pressed
    brne        _display_temp
    cpi         a2,0b00000000       ; check low byte limit ==> sould never go below '0b00001000'
    brne        PC+6                ; branch to sub '0.5' to low byte
    cpi         a3,0b00000000       ; check high byte low limit
    breq        _display_temp
    dec         a3                  ; dec high byte  <=>  subi a3,0b00000001
    ldi         a2,0b11111000       ; set low byte to max value
    rjmp        _display_temp
    subi        a2,0b00001000       ; dec low byte by '0.5'

_display_temp:
    MOV2        b0,b1, a2,a3
    rcall       LCD_clear
    rcall       LCD_home
    PRINTF      LCD
    .db "TempLim=",FFRAC2+FSIGN,b,4,$42,"C ",CR,0
    rjmp        _get_ir_value
_temp_chosen:
    MOV2        b0,b1, a2,a3
    rcall       LCD_clear
    rcall       LCD_home
    PRINTF      LCD
    .db "ChosenTemp=",FFRAC2+FSIGN,b,4,$42,"C ",CR,0    ; not enough space for 'chosen temp lim' ; consider a \n
    WAIT_MS     2000
    ret
```

```
; file: run.asm       target: ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
; Purpose: Temperature comparison and activation of the motor

run:
    cli
    clr     r28
temp:
    rcall   read_temp

_displayTemp:
    PRINTF  LCD
    .db "Temp=", FFRAC2+FSIGN, a, 4, $42, "C ", LF      ;, 0, 0
    .db CR, CR,"Limit=", FFRAC2+FSIGN, b, 4, $42, "C ",CR,0

_test_temp:
    cp      a1,b1               ; compare high byte (MSByte)
    brlo    PC+3
    cp      a0,b0               ; compare low byte (LSByte)
    brsh    mot_loop
    clr     r28
    sei

_back_to_menu:
    WAIT_MS 1000
    cpi     r28,2
    brne    PC+5
    rcall   remote
    cpi     b0,menuGuide            ; check if back to menu asked
    brne    PC+2                ; reg b0 ok to use because overriden by new temp in call read_temp
    rjmp    main
    rjmp    run

mot_loop:
    sei
```

```
    MOTOR     0b0100                ; output motor patterns
    MOTOR     0b1110
    MOTOR     0b1010
    MOTOR     0b1011
    MOTOR     0b0001
    MOTOR     0b0101
    cpi       r28,2
    breq      PC+2
    rjmp      mot_loop
    rjmp      main                  ; Could change for rjmp 'run' // rjmp 'main' enables you to change the limit temp

wait:
    WAIT_US   T_MOT                 ; wait routine called in macro MOTOR (cf macros.asm)
    ret
```

```
; file temp.asm   target ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
; Purpose: Dallas 1-wire(R) temperature sensor routine
; ==> reads temperature from sensor and stores it in reg a0,a1
;
read_temp:
    rcall     wire1_reset
    CA        wire1_write, skipROM
    CA        wire1_write, convertT
    WAIT_MS   750                   ; wait time for the chip to be accessed while the temp is being converted

    rcall     lcd_home
    rcall     wire1_reset
    CA        wire1_write, skipROM
    CA        wire1_write, readScratchpad
    rcall     wire1_read            ; reads temp LSByte and stores result in a0
    mov       c0, a0
    rcall     wire1_read            ; reads temp MSByte
```

```
    mov     a1, a0              ; MSByte in a1
    mov     a0, c0              ; LSByte in a0
    ret
```

```
; file: remote.asm      target: ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
; Purpose: Remote controll signal processing

remote:
    cli                     ; diable interrupts to enable remote controll
    clr     r28             ; clear 'interrupt check value'
ir_detet:
    CLR2    b1,b0           ; clear 2-byte register
    ldi     b2,14           ; load bit-counter
    WP1     PINE,IR         ; Wait if Pin=1
    WAIT_US (T1/4)          ; wait a quarter period

menu_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,menu_loop    ; Decrement and Jump if Not Zero

    com     b0              ; complement b0
    ret
```

```
; file: macros.asm   target ATmega128L-4MHz-STK300
; Authors: Andrea Diez Leboffe, Rodrigue de Guerre
; Purpose: library, general-purpose macros

.macro  MOTOR
    ldi w,@0
    out port_mot,w          ; output motor pin pattern
    rcall   wait            ; wait period
.endmacro

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

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

    .endmacro
.macro  LDZD    ; sram, reg ; sram+reg -> Z
    mov zl,@1
```

```
    clr zh
    subi    zl, low(-@0)
    sbci    zh,high(-@0)
    .endmacro
.macro  LDSP    ; sram
    ldi r16, low(@0)
    out spl,r16
    ldi r16,high(@0)
    out sph,r16
    .endmacro

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

```
    .endmacro

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

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

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

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

```
.macro  DIV2Z           ; divide Z by 2
    lsr zh
    ror zl
    .endmacro

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

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

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

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

```
    sts @0,w
    .endmacro

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

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

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

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

```
    inc @0
    rjmp    _done
_low:   ldi @0,@1
_done:
    .endmacro

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

.macro  INCDEC  ;port,b1,b2,reg,low,high
    sbic    @0,@1
    rjmp    PC+6

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

    sbic    @0,@2
    rjmp    PC+7

    cpi @3,@4
    breq    PC+5
    brlo    PC+3
    dec @3
```

```
    rjmp    PC+2
    ldi @3,@5
    .endmacro

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

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

; wait mili-seconds (ms)
.macro  WAIT_MS ; k
    ldi w, low(@0)
    mov u,w     ; u = LSB
    ldi w,high(@0)+1    ; w = MSB
```

```
wait_ms:
    push    w           ; wait 1000 usec
    push    u
    ldi w, low((clock/3000)-5)
    mov u,w
    ldi w,high((clock/3000)-5)+1
    dec u
    brne    PC-1        ; inner loop: 3 cycles
    dec u               ; adjustment for outer loop
    dec w
    brne    PC-4
    pop u
    pop w

    dec u
    brne    wait_ms
    dec w
    brne    wait_ms
    .endmacro

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

.macro  JK  ; reg,k,addr    ; jump if reg=k
    cpi @0,@1
    breq    @2
    .endmacro
.macro  _JK ; reg,k,addr    ; jump if reg=k
```

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

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

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

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

```
    .endmacro
.macro  DJNK ; reg,k,addr   ; decr and jump if not k
    dec @0
    cpi @0,@1
    brne    @2
    .endmacro

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

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

```
    breq    PC+2
    rjmp    @2
    .endmacro

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

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

.macro  LOOKUP2 ;r1,r0, index,tbl
    mov zl,@2               ; move index into z
    clr zh
    lsl zl                  ; multiply by 2
    rol zh
    subi    zl, low(-2*@3)  ; add base address of table
    sbci    zh,high(-2*@3)
    lpm                     ; get LSB byte
    mov w,r0                ; temporary store LSB in w
```

```
    adiw    zl,1            ; increment Z
    lpm                     ; get MSB byte
    mov @0,r0               ; mov MSB to res1
    mov @1,w                ; mov LSB to res0
    .endmacro

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

.macro  LOOKDOWN ;reg,index,tbl
    ldi ZL, low(2*@2)       ; load table address
    ldi ZH,high(2*@2)
    clr @1
loop:   lpm
    cp  r0,@0
    breq    found
```

```
    inc @1
    adiw    ZL,1
    tst r0
    breq    notfound
    rjmp    loop
notfound:
    ldi @1,-1
found:
    .endmacro

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

```
    mov zh,r0
    pop zl
    ijmp
    .endmacro

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

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

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

```
    lsl @0
    lsl @0
    .endmacro
.macro  MUL8    ; reg
    lsl @0
    lsl @0
    lsl @0
    .endmacro

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

; --- immediate ops with r0..r15 ---
.macro  _ADDI
    ldi w,@1
    add @0,w
    .endmacro
.macro  _ADCI
    ldi w,@1
    adc @0,w
    .endmacro
.macro  _SUBI
    ldi w,@1
    sub @0,w
    .endmacro
.macro  _SBCI
    ldi w,@1
    sbc @0,w
    .endmacro
.macro  _ANDI
    ldi w,@1
    and @0,w
    .endmacro
.macro  _ORI
```

```
    ldi w,@1
    or  @0,w
    .endmacro
.macro  _EORI
    ldi w,@1
    eor @0,w
    .endmacro
.macro  _SBR
    ldi w,@1
    or  @0,w
    .endmacro
.macro  _CBR
    ldi w,~@1
    and @0,w
    .endmacro
.macro  _CPI
    ldi w,@1
    cp  @0,w
    .endmacro
.macro  _LDI
    ldi w,@1
    mov @0,w
    .endmacro

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

```
; --- extending branch distance to +/-2k ---
.macro  _BREQ
    brne    PC+2
    rjmp    @0
    .endmacro
.macro  _BRNE
    breq    PC+2
    rjmp    @0
    .endmacro
.macro  _BRCS
    brcc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRCC
    brcs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRSH
    brlo    PC+2
    rjmp    @0
    .endmacro
.macro  _BRLO
    brsh    PC+2
    rjmp    @0
    .endmacro
.macro  _BRMI
    brpl    PC+2
    rjmp    @0
    .endmacro
.macro  _BRPL
    brmi    PC+2
    rjmp    @0
    .endmacro
.macro  _BRGE
```

```
    brlt    PC+2
    rjmp    @0
    .endmacro
.macro  _BRLT
    brge    PC+2
    rjmp    @0
    .endmacro
.macro  _BRHS
    brhc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRHC
    brhs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRTS
    brtc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRTC
    brts    PC+2
    rjmp    @0
    .endmacro
.macro  _BRVS
    brvc    PC+2
    rjmp    @0
    .endmacro
.macro  _BRVC
    brvs    PC+2
    rjmp    @0
    .endmacro
.macro  _BRIE
    brid    PC+2
    rjmp    @0
    .endmacro
```

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

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

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

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

```
    clc
    .endmacro

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

.macro  B2C ; reg,b     ; bit to carry
    sbrc    @0,@1
    sec
    sbrs    @0,@1
    clc
    .endmacro
.macro  C2B ; reg,b     ; carry to bit
    brcc    PC+2
    sbr @0,(1<<@1)
    brcs    PC+2
    cbr @0,(1<<@1)
    .endmacro
.macro  P2C ; port,b    ; port to carry
    sbic    @0,@1
    sec
    sbis    @0,@1
    clc
    .endmacro
.macro  C2P ; port,b    ; carry to port
    brcc    PC+2
    sbi @0,@1
    brcs    PC+2
    cbi @0,@1
    .endmacro

; --- inverting bits ---
```

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

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

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

; --- logical operations on bits ---
.macro  ANDB    ; r1,b1, r2,b2, r3,b3   ; reg1,b1 <- reg2,b2 AND reg3,b3
    set
    sbrs    @4,@5
    clt
    sbrs    @2,@3
    clt
    bld @0,@1
    .endmacro
.macro  ORB ; r1,b1, r2,b2, r3,b3       ; reg1.b1 <- reg2.b2 OR reg3.b3
    clt
    sbrc    @4,@5
    set
    sbrc    @2,@3
    set
```

```
    bld @0,@1
    .endmacro
.macro  EORB    ; r1,b1, r2,b2, r3,b3   ; reg1.b1 <- reg2.b2 XOR reg3.b3
    sbrc    @4,@5
    rjmp    f1
f0: bst @2,@3
    rjmp    PC+4
f1: set
    sbrc    @0,@1
    clt
    bld @0,@0
    .endmacro

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

```
    sbrs    @0,@1
    rjmp    @2
    .endmacro
.macro  JB1 ; reg,bit,addr      ; jump if bit=1
    sbrc    @0,@1
    rjmp    @2
    .endmacro
.macro  CB0 ; reg,bit,addr      ; call if bit=0
    sbrs    @0,@1
    rcall   @2
    .endmacro
.macro  CB1 ; reg,bit,addr      ; call if bit=1
    sbrc    @0,@1
    rcall   @2
    .endmacro
.macro  WB0 ; reg,bit           ; wait if bit=0
    sbrs    @0,@1
    rjmp    PC-1
    .endmacro
.macro  WB1 ; reg,bit           ; wait if bit=1
    sbrc    @0,@1
    rjmp    PC-1
    .endmacro
.macro  RB0 ; reg,bit           ; return if bit=0
    sbrs    @0,@1
    ret
    .endmacro
.macro  RB1 ; reg,bit           ; return if bit=1
    sbrc    @0,@1
    ret
    .endmacro

; wait if bit=0 with timeout
; if timeout (in units of 5 cyc) then jump to addr
.macro  WB0T    ; reg,bit,timeout,addr
```

```
    ldi w,@2+1
    dec w          ; 1 cyc
    breq    @3     ; 1 cyc
    sbrs    @0,@1  ; 1 cyc
    rjmp    PC-3   ; 2 cyc = 5 cycles
    .endmacro

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

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

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

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

```
    rjmp    PC-3    ; 2 cyc = 5 cycles
    .endmacro

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

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

.macro  SWAP4           ; swap 2 variables
    mov w ,@0
    mov @0,@4
    mov @4,w
    mov w ,@1
    mov @1,@5
    mov @5,w
    mov w ,@2
    mov @2,@6
    mov @6,w
    mov w ,@3
    mov @3,@7
    mov @7,w
    .endmacro
.macro  SWAP3
    mov w ,@0
    mov @0,@3
    mov @3,w
```

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

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

```
    .endmacro

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

.macro  LDZ4    ;r..r0      ; load from (z+)
    ld  @3,z+
    ld  @2,z+
    ld  @1,z+
    ld  @0,z+
    .endmacro
.macro  LDZ3    ;r..r0
    ld  @2,z+
    ld  @1,z+
    ld  @0,z+
    .endmacro
.macro  LDZ2    ;r..r0
    ld  @1,z+
    ld  @0,z+
    .endmacro

.macro  STX4    ;r..r0      ; store to (x+)
```

```
   st  x+,@3
   st  x+,@2
   st  x+,@1
   st  x+,@0
   .endmacro
.macro  STX3    ;r..r0
   st  x+,@2
   st  x+,@1
   st  x+,@0
   .endmacro
.macro  STX2    ;r..r0
   st  x+,@1
   st  x+,@0
   .endmacro

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

.macro  STZ4    ;r..r0      ; store to (z+)
   st  z+,@3
   st  z+,@2
   st  z+,@1
```

```
    st  z+,@0
    .endmacro
.macro  STZ3    ;r..r0
    st  z+,@2
    st  z+,@1
    st  z+,@0
    .endmacro
.macro  STZ2    ;r..r0
    st  z+,@1
    st  z+,@0
    .endmacro

.macro  STI4    ;addr,k     ; store immediate
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
    ldi w,byte3(@1)
    sts @0+2,w
    ldi w,byte4(@1)
    sts @0+3,w
    .endmacro
.macro  STI3    ;addr,k
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
    ldi w,byte3(@1)
    sts @0+2,w
    .endmacro
.macro  STI2    ;addr,k
    ldi w,  low(@1)
    sts @0+0,w
    ldi w, high(@1)
    sts @0+1,w
```

```
    .endmacro
.macro  STI ;addr,k
    ldi w,@1
    sts @0,w
    .endmacro

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

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

```
    adc @1,w
    adc @0,w
    .endmacro
.macro  DEC2
    ldi w,0xff
    add @1,w
    adc @0,w
    .endmacro

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

```
    clr @4
    clr @5
    clr @6
    .endmacro
.macro  CLR6
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    clr @5
    .endmacro
.macro  CLR5
    sub @0,@0
    clr @1
    clr @2
    clr @3
    clr @4
    .endmacro
.macro  CLR4
    sub @0,@0
    clr @1
    clr @2
    clr @3
    .endmacro
.macro  CLR3
    sub @0,@0
    clr @1
    clr @2
    .endmacro
.macro  CLR2
    sub @0,@0
    clr @1
    .endmacro
```

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

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

```
    .endmacro
.macro  NEG2
    com @0
    com @1
    ldi w,0xff
    sub @1,w
    sbc @0,w
    .endmacro

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

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

```
   .endmacro
.macro  LDS2
   lds @1,@2
   lds @0,@2+1
   .endmacro

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

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

```
    std z+@0+1,@1
    .endmacro

.macro  LPM4            ; load program memory
    lpm
    mov @3,r0
    adiw    zl,1
    lpm
    mov @2,r0
    adiw    zl,1
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro
.macro  LPM3
    lpm
    mov @2,r0
    adiw    zl,1
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
    .endmacro
.macro  LPM2
    lpm
    mov @1,r0
    adiw    zl,1
    lpm
    mov @0,r0
    adiw    zl,1
```

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

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

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

```
   add @1,@3
   adc @0,@2
   .endmacro

.macro  SUB4          ; subtract
   sub @3,@7
   sbc @2,@6
   sbc @1,@5
   sbc @0,@4
   .endmacro
.macro  SUB3
   sub @2,@5
   sbc @1,@4
   sbc @0,@3
   .endmacro
.macro  SUB2
   sub @1,@3
   sbc @0,@2
   .endmacro

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

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

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

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

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

```
.macro  LSL2
    lsl @1
    rol @0
    .endmacro

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

.macro  ASR4          ; arithmetic shift right
    asr @0
    ror @1
    ror @2
    ror @3
    .endmacro
.macro  ASR3
    asr @0
    ror @1
    ror @2
    .endmacro
.macro  ASR2
    asr @0
    ror @1
```

```
    .endmacro

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

```
    .endmacro
.macro  ROL4
    rol @3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL3
    rol @2
    rol @1
    rol @0
    .endmacro
.macro  ROL2
    rol @1
    rol @0
    .endmacro

.macro  ROR8            ; rotate right through carry
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    ror @5
    ror @6
    ror @7
    .endmacro
.macro  ROR7
    ror @0
    ror @1
    ror @2
    ror @3
    ror @4
    ror @5
    ror @6
```

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

.macro  PUSH2
    push    @0
    push    @1
```

```
    .endmacro
.macro  POP2
    pop @1
    pop @0
    .endmacro

.macro  PUSH3
    push    @0
    push    @1
    push    @2
    .endmacro
.macro  POP3
    pop @2
    pop @1
    pop @0
    .endmacro

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

.macro  PUSH5
    push    @0
    push    @1
    push    @2
    push    @3
```

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

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

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

```
    lds w,@0+1
    inc w
    sts @0+1,w
    brne    end
    lds w,@0+2
    inc w
    sts @0+2,w
end:
    .endmacro

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

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

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

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

```
    .endmacro

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

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

```
    .endmacro

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

; --- calls with arguments a,b,XYZ ---
.macro  CX  ; subroutine,x
    ldi xl, low(@1)
    ldi xh,high(@1)
    rcall   @0
    .endmacro
.macro  CXY ; subroutine,x,y
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi yl, low(@2)
    ldi yh,high(@2)
    rcall   @0
    .endmacro
.macro  CXZ ; subroutine,x,z
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi zl, low(@2)
    ldi zh,high(@2)
    rcall   @0
    .endmacro
.macro  CXYZ    ; subroutine,x,y,z
    ldi xl, low(@1)
    ldi xh,high(@1)
    ldi yl, low(@2)
    ldi yh,high(@2)
    ldi zl, low(@3)
    ldi zh,high(@3)
    rcall   @0
    .endmacro
```

```
.macro  CW  ; subroutine,w
    ldi w, @1
    rcall   @0
    .endmacro
.macro  CA  ; subroutine,a
    ldi a0, @1
    rcall   @0
    .endmacro
.macro  CAB ; subroutine,a,b
    ldi a0, @1
    ldi b0, @2
    rcall   @0
    .endmacro

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