```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; port_definitions.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Detta är en lista på olika portdefinitioner och
; adresser till de olika twi-expanders på DAvid-kortet.
;
; Bör inkluderas i början av main.asm.
;
;;;;;;;;;;;;;;;;

#ifndef _PORT_DEF_
#define _PORT_DEF_

;;;;;;;;;;;;;;;;
;   Adresser
;;;;;;;;;;;;;;;;

        .equ    ADDR_LCD    = $20       ; IC1
        .equ    ADDR_LEFT8  = $24       ; IC2
        .equ    ADDR_RIGHT8 = $25       ; IC3
        .equ    ADDR_ROTLED = $26       ; IC4
        .equ    ADDR_SWITCH = $27       ; IC5
        ;.equ   SLA_W       = (ADDR_RIGHT8 << 1) | 0    ; $4A 0b01001010
        ;.equ   SLA_R       = (ADDR_RIGHT8 << 1) | 1    ; $4B 0b01001011

;;;;;;;;;;;;;;;;
;   Arduino pins
;;;;;;;;;;;;;;;;

        .equ    IR_RX       = PB0
        .equ    SPEAKER     = PB1       ; Piezo-högtalaren (även IR-TX)

        .equ    MATRIX_LATCH= PB2       ; SPI SS
        .equ    MOSI        = PB3       ; SPI
        .equ    MISO        = PB4       ; SPI   (Även RGB-remsan)
        .equ    SPI_CLK     = PB5       ; SPI


        .equ    JOY_R_H     = PC0       ; Höger joystick x-led
        .equ    JOY_R_V     = PC1       ; Höger joystick y-led
        .equ    JOY_L_H     = PC2       ; Vänster joystick x-led
        .equ    JOY_L_V     = PC3       ; Vänster joystick y-led
```

```
        .equ    SDA         = PC4       ; TWI
        .equ    SCL         = PC5       ; TWI


        .equ    SW_R        = PD0       ; Tryckknapp R
        .equ    SW_L        = PD1       ; Tryckknapp L

        .equ    RTC_CLK     = PD2       ; Realtidsklocka

        .equ    SW_ROT      = PD3       ; Tryckknapp vred

        .equ    D_LED_R     = PD4       ; LED på kortet, skiljt från LED_R ovan
knapp R (IC4)
        .equ    D_LED_L     = PD5       ; Se ovan

        .equ    ROT_B       = PD6       ; Rotary B
        .equ    ROT_A       = PD7       ; Rotary A, undersök vilken av dessa som
är höger/vänster

;;;;;;;;;;;;;;;;
;   I/O-expanders bit-definitioner
;;;;;;;;;;;;;;;;


    ;:: IC1, LCD ::
        .equ    LCD_RS      = 0
        .equ    LCD_RW      = 1
        .equ    LCD_E       = 2
        .equ    LCD_BL      = 3
        .equ    LCD_D4      = 4
        .equ    LCD_D5      = 5
        .equ    LCD_D6      = 6
        .equ    LCD_D7      = 7


    ;:: IC4, LEDS ::
        .equ    LED_ROT0    = 0
        .equ    LED_ROT1    = 1
        .equ    LED_L1      = 2
        .equ    LED_L       = 3
        .equ    LED_L2      = 4
        .equ    LED_R1      = 5
        .equ    LED_R       = 6
        .equ    LED_R2      = 7


    ;:: IC5, SWITCH ::
        .equ    SW_R1       = 0
        .equ    SW_R2       = 1
        .equ    SW_L1       = 2
        .equ    SW_L2       = 3
        .equ    JOY_R_SEL   = 4
        .equ    JOY_L_SEL   = 5
```

```
#endif /* _PORT_DEF_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; memory.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Detta är en lista på minnesallokeringar i processorns SRAM.
;
; Bör inkluderas i början av main.asm.
;
;;;;;;;;;;;;;;;;

#ifndef _MEM_
#define _MEM_

;;;;;;;;;;;;;;;;
;   Minne
;;;;;;;;;;;;;;;;

        .dseg
        .org    SRAM_START



LED_STATUS:
        .byte   1

RIGHT8_VAL:
        .byte   1
LEFT8_VAL:
        .byte   1


LCD_PORT:
        .byte   1
LINE:
        .byte   17
LCD_LINE1:
        .byte   17
LCD_LINE2:
        .byte   17
```

```
CURSOR:
        .byte   1                       ; Cursor-position


NOTE_LENGTH:
NOTE_LENGTH_LOW:
        .byte   1
NOTE_LENGTH_HIGH:
        .byte   1



DAMATRIX_MEM: ; Vidominnet
                                        ; En rad är Matris 1s GBR följt av matris
2s GBR
DA_ROW1:
        .byte   6
DA_ROW2:
        .byte   6
DA_ROW3:
        .byte   6
DA_ROW4:
        .byte   6
DA_ROW5:
        .byte   6
DA_ROW6:
        .byte   6
DA_ROW7:
        .byte   6
DA_ROW8:
        .byte   6


GAMEBOARD:

GB_ROW1:
        .byte   16
GB_ROW2:
        .byte   16
GB_ROW3:
        .byte   16
GB_ROW4:
        .byte   16
GB_ROW5:
        .byte   16
GB_ROW6:
        .byte   16
GB_ROW7:
        .byte   16
GB_ROW8:
        .byte   16


COMPONENT_TABLE:
```

```
    PADDLE1:
        ;x,y
        .byte 2
    PADDLE2:
        ;x,y
        .byte 2
    BALL:
        ;x,y,riktning
        .byte 3

PLAYER2_SCORED:
        .byte 1
PLAYER1_SCORED:
        .byte 1
P1_SCORE:
        .byte 1
P2_SCORE:
        .byte 1
PLAYER_WIN:
        .byte 1

COUNTER_UPDATE:
        .byte   1

        .cseg


#endif /* _MEM_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; main.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



        .INCLUDE "port_definitions.asm"

        .org    $0000
        jmp     COLD
        .org    OC2Aaddr
        jmp     SPEAKER_TIMER_INT
        .org    OC1Aaddr
        jmp     TIMER1_INT

        .org    INT_VECTORS_SIZE
```

```
        .INCLUDE "memory.asm"
        .INCLUDE "twi.asm"
        .INCLUDE "switches.asm"
        .INCLUDE "7seg.asm"
        .INCLUDE "led.asm"
        .INCLUDE "lcd.asm"
        .INCLUDE "DAmatrix.asm"
        .INCLUDE "gameengine.asm"
        .INCLUDE "speaker.asm"

        .equ    N       = $64


;;;;;;;;;;;;;;;;
;   Uppstart
;;;;;;;;;;;;;;;;


COLD:
    ; Initiera stackpekaren
        ldi     r16, HIGH(RAMEND)
        out     SPH, r16
        ldi     r16, LOW(RAMEND)
        out     SPL, r16


    ; Kör initieringar av hårdvara och minne
        call    INIT_TWI
        call    LINE_INIT
        call    LCD_INIT
        call    SPI_MasterInit
        call    TIMER1_INIT
        call    TIMER2_INIT
        call    DA_MEM_INIT
        call    DA_PRINT_MEM


    ; Skriv ut välkomstmeddelande
        ldi     ZH, HIGH(WELCOME_MSG*2)
        ldi     ZL, LOW(WELCOME_MSG*2)
        call    LCD_FLASH_PRINT
        ldi     r16, $02
        call    DELAY_S

                ; Kör MAIN, eller välj ett testprogram från tests.asm
        jmp     MAIN

;;;;;;;;;;;;;;;;
;   Huvudprogram
;;;;;;;;;;;;;;;;

MAIN:
    ; Skriv ut "redo"-meddelandet
```

```
        ldi     ZH, HIGH(READY_MSG*2)
        ldi     ZL, LOW(READY_MSG*2)
        call    LCD_FLASH_PRINT

    ; Kolla knapptryckningar för att starta spelet
        clz
        call    RQ
        ;brne   MAIN    ; Avkommentera denna om vi vill att båda spelarna måste
trycka "redo"
        call    LQ
        brne    MAIN

    ; Töm LCDn på "redo"-meddelandet
        ;call   LINE_INIT
        ;call   LINE_PRINT
        call    LCD_ERASE

    ; Starta spelet
        call    GAME_INIT
        call    PONG
        rjmp    MAIN




;;;;;;;;;;;;;;;;
;   Subrutiner
;;;;;;;;;;;;;;;;

;:: V�nterutiner ::

WAIT:
        push    r16
        ldi     r16, $34
W1:
        dec     r16
        brne    W1
        pop     r16
        ret

    ; ---------------

DELAY_S:                        ; Vänteloop som varar i antal sekunder, angivet
som argument i r16.
        push    r17
        mov     r17, r16
        ldi     r16, N
DELAY_S1:
        call    DELAY_N
        dec     r17
        brne    DELAY_S1
        pop     r17
        ret
```

```
    ; ---------------


DELAY_N:                        ; L�ngre v�nteloop, styrt av N som �r definerat
i b�rjan under "Data".
        push    r16
        ldi     r16, N
DELAY_N1:
        call    DELAY
        dec     r16
        brne    DELAY_N1
        pop     r16
        ret

    ; ---------------

DELAY:
                                ; V�nte-loop, upp till ~16 ms ($FFFF, h�r 10 ms
        push    r25
        push    r24
        ldi     r25, $63        ; $63C4 ger 160000 cykler f�r hela rutinen, i
princip exakt 10.0 ms
        ldi     r24, $C4
D1:
        adiw    r24, 1
        brne    D1
        pop     r24
        pop     r25
        ret



;;;;;;;;;;;;;;;;
;   Flashminnes-data
;;;;;;;;;;;;;;;;

        .INCLUDE "flash_messages.asm"

;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; gameengine.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
```

```
;   Beskrivning
;
;
;;;;;;;;;;;;;;;;

#ifndef _GAME_ENGINE_
#define _GAME_ENGINE_

;;;;;;;;;;;;;;;;
;       Timer
;;;;;;;;;;;;;;;;

        .equ    TIMER1_TICKS = 31250 - 1    ; 1/8 sekund @ 16/64 MHz

TIMER1_INIT:
        push    r16
        ldi     r16, (1<<WGM12)|(1<<CS11)|(1<<CS10) ; CTC, prescale 64
        sts     TCCR1B, r16
        ldi     r16, HIGH(TIMER1_TICKS)
        sts     OCR1AH, r16
        ldi     r16, LOW(TIMER1_TICKS)
        sts     OCR1AL, r16
        ldi     r16, (1<<OCIE1A)            ; allow to interrupt
        sts     TIMSK1, r16
        pop     r16
        ret

TIMER1_INT:
        push    r19
        push    r18
        push    r17
        push    r16
        in      r16, SREG
        push    r16
        call    UPDATE_BALL
        lds     r16, COUNTER_UPDATE
        inc     r16
        cpi     r16, $02
        brne    TIMER_DONE
        call    UPDATE_PADDLE1 ; vänstra planket
        call    UPDATE_PADDLE2 ; högra planket
        clr     r16
TIMER_DONE:
        sts     COUNTER_UPDATE, r16
        pop     r16
        out     SREG, r16
        pop     r16
        pop     r17
        pop     r18
        pop     r19
        reti


;;;;;;;;;;;;;;;;
```

```
;       Paddles
;;;;;;;;;;;;;;;;

UPDATE_PADDLE1:
        ; READ_JOY_L_V
        ; JOY i mitten 7F
        call    READ_JOY_L_V
        cpi     r16, $40            ; Gå ner
        brcs    INC_PADDLE1
        cpi     r16, $BF            ; Gå upp
        brcc    DEC_PADDLE1
        rjmp    RETURN_PADDLE1
DEC_PADDLE1:
        lds     r16, PADDLE1+1
        cpi     r16, $00
        breq    RETURN_PADDLE1
        dec     r16
        sts     PADDLE1+1, r16
        rjmp    RETURN_PADDLE1
INC_PADDLE1:
        lds     r16, PADDLE1+1
        cpi     r16, $06
        breq    RETURN_PADDLE1
        inc     r16
        sts     PADDLE1+1, r16
        rjmp    RETURN_PADDLE1
RETURN_PADDLE1:
        ret


UPDATE_PADDLE2:
        ; READ_JOY_R_V
        ; JOY i mitten 7E
        call    READ_JOY_R_V
        cpi     r16, $40            ; Gå ner
        brcs    INC_PADDLE2
        cpi     r16, $BF            ; Gå upp
        brcc    DEC_PADDLE2
        rjmp    RETURN_PADDLE2
DEC_PADDLE2:
        lds     r16, PADDLE2+1
        cpi     r16, $00
        breq    RETURN_PADDLE2
        dec     r16
        sts     PADDLE2+1, r16
        rjmp    RETURN_PADDLE2
INC_PADDLE2:
        lds     r16, PADDLE2+1
        cpi     r16, $06
        breq    RETURN_PADDLE2
        inc     r16
        sts     PADDLE2+1, r16
        rjmp    RETURN_PADDLE2
RETURN_PADDLE2:
```

```
        ret

LOAD_PADDLES:
        ldi     ZH, HIGH(PADDLE1)
        ldi     ZL, LOW(PADDLE1)
        call    LOAD_ONE_PADDLE
        ; Z = PADDLE2
        call    LOAD_ONE_PADDLE
        ret

LOAD_ONE_PADDLE:
        ldi     YH, HIGH(GAMEBOARD)
        ldi     YL, LOW(GAMEBOARD)
        ld      r16, Z+             ; Paddles X-koord
        add     YL, r16             ; Lägg till X koord som offset på gameboard-
pekaradressen
        brcc    NO_CARRY_X
        inc     YH
    NO_CARRY_X:
        ld      r16, Z+             ; Paddles Y-koord
        ldi     r17, 16
        mul     r16, r17

        add     YL, R0              ; Lägg till Y koord som offset på gameboard-
pekaradressen
        brcc    NO_CARRY_Y
        inc     YH
    NO_CARRY_Y:
                                    ; Här pekar Y på adress i gameboard som
motsvarar X,Y-koordinaterna
        ldi     r16, 'B'
        st      Y, r16
        adiw    YH:YL, 16
        st      Y, r16
        ret


INIT_PADDLES:
        ldi     ZH, HIGH(PADDLE1)
        ldi     ZL, LOW(PADDLE1)
        ldi     r16, 0
        st      Z+, r16
        ldi     r16, 3
        st      Z+, r16
        ldi     r16, 15
        st      Z+, r16
        ldi     r16, 3
        st      Z+, r16
        ret


;;;;;;;;;;;;;;;;
;       Ball
;;;;;;;;;;;;;;;;
```

```
/*INIT_BALL:
        ; Startposition
        ldi     ZH, HIGH(BALL)
        ldi     ZL, LOW(BALL)
        ldi     r16, 7
        st      Z+, r16
        ldi     r16, 4
        st      Z+, r16
        ldi     r16, $06
        st      Z, r16
        ret*/

INIT_BALL:
        ; Startposition
        ldi     ZH, HIGH(BALL)
        ldi     ZL, LOW(BALL)
        lds     r17, TCNT1L     ; pseudo-random-värde hämtat från räknare

        sbrc    r17, 2
        ldi     r16, $07
        sbrs    r17, 2
        ldi     r16, $08
        st      Z+, r16     ; X

        mov     r16, r17
        andi    r16, $07    ; Maska med 00000XXX, värde mellan 0-7
        st      Z+, r16     ; Y

        andi    r17, $0F
        cpi     r17, $04
        brlo    DIR_5
        cpi     r17, $08
        brlo    DIR_6
        cpi     r17, $0C
        brlo    DIR_9
DIR_10:
        ldi     r16, $0A
        rjmp    STORE_DIR
DIR_9:
        ldi     r16, $09
        rjmp    STORE_DIR
DIR_6:
        ldi     r16, $06
        rjmp    STORE_DIR
DIR_5:
        ldi     r16, $05
STORE_DIR:
        st      Z, r16      ; Riktning
        ret


LOAD_BALL:
        ldi     ZH, HIGH(BALL)
```

```
        ldi     ZL, LOW(BALL)
        ldi     YH, HIGH(GAMEBOARD)
        ldi     YL, LOW(GAMEBOARD)
        ld      r16, Z+             ; Ball X-koord
        add     YL, r16             ; Lägg till X koord som offset på gameboard-
pekaradressen
        brcc    BALL_NO_CARRY_X
        inc     YH
BALL_NO_CARRY_X:
        ld      r16, Z+             ; Ball Y-koord
        ldi     r17, 16
        mul     r16, r17

        add     YL, R0              ; Lägg till Y koord som offset på gameboard-
pekaradressen
        brcc    BALL_NO_CARRY_Y
        inc     YH
    BALL_NO_CARRY_Y:

        ldi     r16, 'R'
        st      Y, r16
        ret


UPDATE_BALL:

        call    CHECK_SCORING   ; ska inte update ball avbryta här om en bollen är
i vägg??
        call    MOVE_BALL
        call    CHECK_PADDLE_COLLISION
        call    WALL_BOUNCE
        ret


CHECK_SCORING:
        lds     r16, (BALL)     ; bollens X
        cpi     r16, $00
        breq    SCORE2
        cpi     r16, $0F
        breq    SCORE1
        rjmp    CHECK_SCORING_DONE
SCORE2:
        ldi     r16, $01
        sts     PLAYER2_SCORED, r16
        rjmp    CHECK_SCORING_DONE
SCORE1:
        ldi     r16, $01
        sts     PLAYER1_SCORED, r16
        rjmp    CHECK_SCORING_DONE
CHECK_SCORING_DONE:
        ret

;Generell move
MOVE_BALL:
```

```
        lds     r16, (BALL)     ; bollens X
        lds     r17, (BALL+1)   ; bollens Y
        lds     r18, (BALL+2)   ; "riktning"
        sbrc    r18, 0          ; Xr+
        inc     r16
        sbrc    r18, 1          ; Xr-
        dec     r16
        sbrc    r18, 2          ; Yr+
        inc     r17
        sbrc    r18, 3          ; Yr-
        dec     r17
        sts     (BALL), r16
        sts     (BALL+1), r17
        ret


; Kolla om krock med Paddle
CHECK_PADDLE_COLLISION:
        lds     r16, (BALL)     ; bollens X
        lds     r17, (BALL+1)   ; bollens Y
        cpi     r16, $01
        brne    NO_POT_COLLISION_LEFT
        lds     r18, (PADDLE1+1)
        cp      r17, r18
        breq    COLLISION
        inc     r18
        cp      r17, r18
        breq    COLLISION
NO_POT_COLLISION_LEFT:
        cpi     r16, $0E
        brne    NO_COLLISION
        lds     r18, (PADDLE2+1)
        cp      r17, r18
        breq    COLLISION
        inc     r18
        cp      r17, r18
        breq    COLLISION
        rjmp    NO_COLLISION
COLLISION:
        call    PADDLE_BOUNCE
NO_COLLISION:
        ret


; Generell studs mot paddel
PADDLE_BOUNCE:
        lds     r16, (BALL)     ; bollens X
        lds     r18, (BALL+2)   ; "riktning"

        ldi     r19, $01
        cpi     r16, $01
        breq    PADDLE_BOUNCE_LEFT  ; vänster vägg, -1 på riktning
        cpi     r16, $0E
        breq    PADDLE_BOUNCE_RIGHT ; höger vägg, +1 på riktning
```

```
        rjmp    PADDLE_BOUNCE_DONE
PADDLE_BOUNCE_LEFT:
        sub     r18, r19
        sts     (BALL+2), r18
        rjmp    PADDLE_BOUNCE_DONE
PADDLE_BOUNCE_RIGHT:
        add     r18, r19
        sts     (BALL+2), r18
        rjmp    PADDLE_BOUNCE_DONE
PADDLE_BOUNCE_DONE:
        call    PLAY_NOTE_G
        ret


; Generell studs tak/golv
WALL_BOUNCE:
        lds     r17, (BALL+1)   ; bollens Y
        lds     r18, (BALL+2)   ; "riktning"

        ldi     r19, $04
        cpi     r17, $00
        breq    WALL_BOUNCE_TOP ; tak, -4 på riktning
        cpi     r17, $07
        breq    WALL_BOUNCE_BOT ; golv, +4 på riktning
        rjmp    WALL_BOUNCE_DONE
WALL_BOUNCE_TOP:
        sub     r18, r19
        sts     (BALL+2), r18
        call    PLAY_NOTE_B
        rjmp    WALL_BOUNCE_DONE
WALL_BOUNCE_BOT:
        add     r18, r19
        sts     (BALL+2), r18
        call    PLAY_NOTE_A
        rjmp    WALL_BOUNCE_DONE
WALL_BOUNCE_DONE:
        ret

;;;;;;;;;;;;;;;;
;       Gameboard
;;;;;;;;;;;;;;;;

CLEAR_GAMEBOARD:
        ldi     YH, HIGH(GAMEBOARD)
        ldi     YL, LOW(GAMEBOARD)
        clr     r16
        ldi     r17, $80            ; 8 rader x 16 bytes
CLEAR_GAMEBOARD_LOOP:
        st      Y+, r16
        dec     r17
        brne    CLEAR_GAMEBOARD_LOOP
        ret

;;;;;;;;;;;;;;;;
```

```
;       If scored
;;;;;;;;;;;;;;;;

PLAYER_SCORED: ; kontrollerar om spelaren och gjort mål och inc score:n
        lds     r16, PLAYER1_SCORED
        cpi     r16, 1
        breq    INC_P1_SCORE
        lds     r16, PLAYER2_SCORED
        cpi     r16, 1
        breq    INC_P2_SCORE
        rjmp    PLAYER_SCORED_DONE
INC_P1_SCORE:
        clr     r16
        sts     PLAYER1_SCORED, r16
        lds     r16, P1_SCORE
        inc     r16
        sts     P1_SCORE, r16
        cpi     r16, $05
        brne    P1_NO_WIN
        ldi     r17, $01
        sts     PLAYER_WIN, r17
P1_NO_WIN:
        call    LEFT8_WRITE
        call    INIT_BALL
        rjmp    PLAYER_SCORED_DONE
INC_P2_SCORE:
        clr     r16
        sts     PLAYER2_SCORED, r16
        lds     r16, P2_SCORE
        inc     r16
        sts     P2_SCORE, r16
        cpi     r16, $05
        brne    P2_NO_WIN
        ldi     r17, $01
        sts     PLAYER_WIN, r17
P2_NO_WIN:
        call    RIGHT8_WRITE
        call    INIT_BALL
PLAYER_SCORED_DONE:
        et

CHECK_WIN:
        lds     r16, PLAYER_WIN
        cpi     r16, $01
        ret

;;;;;;;;;;;;;;;;
;       GAMELOOP
;;;;;;;;;;;;;;;;


PONG:
        call    DELAY
        call    DELAY
```

```
        call    DELAY

        call    UPDATE
        call    DA_PRINT_MEM
        call    CHECK_WIN
        breq    WIN
        rjmp    PONG
WIN:
        cli
        call    PRINT_WIN_MSG
        ; call  FIREWORKS
        ; Spela ljud
        ldi     r16, $03
        call    DELAY_S     ; 3 sekunder
        call    LCD_ERASE
        ret

UPDATE:
        call    CLEAR_GAMEBOARD
        call    LOAD_PADDLES
        call    LOAD_BALL
        call    LOAD_DA_MEM
        call    PLAYER_SCORED
        ret

; Laddar kordinaterna till Gameboard

PRINT_WIN_MSG:
        lds     r16, P1_SCORE
        cpi     r16, $05
        brne    OTHER_PLAYER
        ldi     ZH, HIGH(P1_WINS_MSG*2)
        ldi     ZL, LOW(P1_WINS_MSG*2)
        rjmp    WIN_MSG_LOADED
    OTHER_PLAYER:
        ldi     ZH, HIGH(P2_WINS_MSG*2)
        ldi     ZL, LOW(P2_WINS_MSG*2)
WIN_MSG_LOADED:
        call    LCD_FLASH_PRINT
        ret

GAME_INIT:
        call    INIT_PADDLES
        call    INIT_BALL

        clr     r16         ; Cleara alla flaggor och räknare för spelet i minnet
        ldi     ZH, HIGH(PLAYER2_SCORED)
        ldi     ZL, LOW(PLAYER2_SCORED)
        st      Z+, r16
        st      Z+, r16
        st      Z+, r16
        st      Z+, r16
        st      Z+, r16
        st      Z+, r16
```

```
        call    LEFT8_WRITE
        clr     r16
        call    RIGHT8_WRITE

        sei

        ret


#endif /* _GAME_ENGINE_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; DAmatrix.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
;   Den här filen hanterar rutinerna för vår diplay bestående av två horisontellt
länkade DAmatrix.
;
;   Används främst genom utskrift mha "DA_PRINT_MEM" som skriver ut det befintliga
värdena i videominnet "DA_MEM" i memory.asm.
;
;   Den finns rutiner för att skriva ut stillbilder från tabeller i flash som
laddas till videominnet,
;   men det är tänkt att man ska sköta detta genom regelbunden överföring från
minnestabellen "GAMEBOARD" genom "LOAD_DA_MEM"
;
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Läs eventuellt in DDRB och or:a in istället vid init (för kompabilitet
med högtalaren)
;       * Lös bugg med blinkande lampor om få "pixlar" är tända
;       * Ändra flashutskriftrutin till att ta argument, så kan man mata in vilken
laddad tabell som helst. (Bra för vinstskärmar/animationer?)
;         (KLAR se GAMEBOARD_FROM_FLASH_ZPTR)
;
; ENDTODO
;;;;;;;;;;;;;;;;
```

```
#ifndef _DAMATRIX_
#define _DAMATRIX_


;;;;;;;;;;;;;;;;
;       SPI
;;;;;;;;;;;;;;;;

SPI_MasterInit:
; Set MOSI and SCK output, all others input
        ldi     r17, (1<<MATRIX_LATCH)|(1<<MOSI)|(1<<SPI_CLK)
        out     DDRB, r17
; Enable SPI, Master, set clock rate fck/4
        ldi     r17, (1<<SPE)|(1<<MSTR)|(0<<SPR1)|(0<<SPR0)
        out     SPCR, r17
        ret

SPI_OPEN:
    push    r16
    in      r16, PORTB
    cbr     r16, (1<<MATRIX_LATCH)
    out     PORTB, r16
    pop     r16
    ret

SPI_CLOSE:
        push    r16
        in      r16, PORTB
        sbr     r16, (1<<MATRIX_LATCH)
        out     PORTB, r16
        pop     r16
        ret



SPI_SEND_BYTE:
; Start transmission of data (r16)
        out     SPDR, r16
SPI_WAIT:
; Wait for transmission complete
        in      r16, SPSR
        sbrs    r16, SPIF
        rjmp    SPI_WAIT
        ret

;;;;;;;;;;;;;;;;
;   DAmatrix
;;;;;;;;;;;;;;;;


GAMEBOARD_FROM_FLASH:
        ldi     YH, HIGH(GAMEBOARD)
        ldi     YL, LOW(GAMEBOARD)
        ldi     ZH, HIGH(PIC_RAM*2)
```

```
        ldi     ZL, LOW(PIC_RAM*2)
        ldi     r17, $80            ; 8 rader x 16 bytes
GAMEBOARD_FROM_FLASH_LOOP:
        lpm     r16, Z+
        st      Y+, r16
        dec     r17
        brne    GAMEBOARD_FROM_FLASH_LOOP
        ret

DA_MEM_FLASH:
        ldi     YH, HIGH(DAMATRIX_MEM)
        ldi     YL, LOW(DAMATRIX_MEM)
        ldi     ZH, HIGH(PIC*2)
        ldi     ZL, LOW(PIC*2)
        ldi     r17, $30            ; 8 rader x 6 bytes
DA_MEM_FLASH_LOOP:
        lpm     r16, Z+
        st      Y+, r16
        st      Y+, r16
        st      Y+, r16
        dec     r17
        brne    DA_MEM_FLASH_LOOP
        ret

;;;;;;;;;;;;
; Animering
;;;;;;;;;;;;

GAMEBOARD_FROM_FLASH_ZPTR:          ; Tar tabell som argument i Z-pekaren
        ldi     YH, HIGH(GAMEBOARD)
        ldi     YL, LOW(GAMEBOARD)
        ldi     r17, $80            ; 8 rader x 16 bytes
GAMEBOARD_FROM_FLASH_ZPTR_LOOP:
        lpm     r16, Z+
        st      Y+, r16
        dec     r17
        brne    GAMEBOARD_FROM_FLASH_ZPTR_LOOP
        ret

FIREWORKS:
        ldi     ZH, HIGH(FW_ANIM1*2)
        ldi     ZL, LOW(FW_ANIM1*2)
        call    GAMEBOARD_FROM_FLASH_ZPTR
        call    LOAD_DA_MEM
        call    DA_PRINT_MEM
        call    DELAY_N
        ldi     ZH, HIGH(FW_ANIM2*2)
        ldi     ZL, LOW(FW_ANIM2*2)
        call    GAMEBOARD_FROM_FLASH_ZPTR
        call    LOAD_DA_MEM
        call    DA_PRINT_MEM
        call    DELAY_N
        ldi     ZH, HIGH(FW_ANIM3*2)
        ldi     ZL, LOW(FW_ANIM3*2)
```

```
        call    GAMEBOARD_FROM_FLASH_ZPTR
        call    LOAD_DA_MEM
        call    DA_PRINT_MEM
        call    DELAY_N
        ret

;;;;;;;;;;;;
; Gameboard
;;;;;;;;;;;;


; Laddar från Gameboard till videominne
LOAD_DA_MEM:
        push    YH
        push    YL
        push    ZH
        push    ZL
        ldi YH, HIGH(DAMATRIX_MEM)
    ldi YL, LOW(DAMATRIX_MEM)
        ldi ZH, HIGH(GAMEBOARD)
    ldi ZL, LOW(GAMEBOARD)

        ldi     r17, 16
LOAD_DA_MEM_LOOP: ; kör 16 gånger (2 displayer) * (8 rader)
        call    READ_GAMEBOARD_TO_DA
        dec     r17
        brne    LOAD_DA_MEM_LOOP

        pop     ZL
        pop     ZH
        pop     YL
        pop     YH
        ret

; läser en rad från en display och överför data till videominnet
READ_GAMEBOARD_TO_DA: ; x16
        push    r16
        push    r17
        push    r18
        push    r19
        push    r20

        clr     r17             ; till röd i videominne
        clr     r18             ; till blå
        clr     r19             ; till grön
        ldi     r20, $80
READ_GAMEBOARD_TO_DA_LOOP:
        ld      r16, Z+
        ; if(r16 = R,B,G)
        cpi     r16, 'R'
        breq    RED
        cpi     r16, 'B'
        breq    BLUE
        cpi     r16, 'G'
```

```
        breq    GREEN
        cpi     r16, 'W'
        breq    WHITE
        rjmp    END_IF

RED:
        or      r17, r20
        rjmp    END_IF
BLUE:
        or      r18, r20
        rjmp     END_IF
GREEN:
        or      r19, r20
        rjmp     END_IF
WHITE:
        or      r17, r20
        or      r18, r20
        or      r19, r20
        rjmp    END_IF
END_IF:
        lsr     r20
        brne    READ_GAMEBOARD_TO_DA_LOOP
        call    DA_TRANSFER_BYTE_TO_DA_MEM
        pop     r20
        pop     r19
        pop     r18
        pop     r17
        pop     r16
        ret

; överför en byte till videominnet
DA_TRANSFER_BYTE_TO_DA_MEM:
        st Y+, r17
        st Y+, r18
        st Y+, r19
        ret


;;;;;;;;;;;;
; Videominnet
;;;;;;;;;;;;

; initierar videominnet med 0:or
DA_MEM_INIT:
        ldi ZH, HIGH(DAMATRIX_MEM)
    ldi ZL, LOW(DAMATRIX_MEM)
        ldi     r17, $30            ; 8 rader x 6 bytes
DA_MEM_INIT_LOOP:
        ldi     r16, $00
        st      Z+, r16
        dec     r17
        brne    DA_MEM_INIT_LOOP
        ret
```

```
;;;;;;;;;;;;
; Print av videominnet
;;;;;;;;;;;;

DA_PRINT_MEM:
        ldi ZH, HIGH(DA_ROW8 + 6)
        ldi ZL, LOW(DA_ROW8 + 6)
        ldi     r17, $80 ;0b 1000 0000
        ;call    SPI_OPEN
DA_PRINT_MEM_LOOP:
        call    DA_PRINT_ROW
        brne    DA_PRINT_MEM_LOOP
        ;call    SPI_CLOSE
        ret
        ; $FF - (radnr)^2

DA_PRINT_ROW:

        ; Sätt rad
        mov     r18, r17
        com     r18
        call    SPI_OPEN
        call    DA_PRINT_ONE_DISPLAY
        call    DA_PRINT_ONE_DISPLAY
        call    SPI_CLOSE
        lsr     r17
        ret

DA_PRINT_ONE_DISPLAY:
        push    r17
        ldi     r17, $03
DA_PRINT_ONE_DISPLAY_LOOP:
        ld      r16, -Z
        call    SPI_SEND_BYTE
        dec     r17
        brne    DA_PRINT_ONE_DISPLAY_LOOP
        mov     r16, r18
        call    SPI_SEND_BYTE
        pop     r17
        ret


#endif /* _DAMATRIX_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; twi.asm
;
; TSIU51 -  Mikrodatorprojekt
```

```
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Modulen för hårdvarustött TWI-protokoll.
;
; Kräver un uppstart med INIT_TWI.
; Man kan sen använda TWI_SEND och TWI_READ för att skicka/ta emot 1 byte data.
;
; Båda använder r16 för indata/utdata, och kräver en adress laddad (7bits format)
i r17.
;
; Beroende av portdefinitionerna för SDL och SDA.
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Lägg till felkodshantering
;       * Interuppt-säkra
;       * Gör varianter för att skicka/ta emot mer än 1 byte vid varje
transaktion.
;       * Anpassa eventuellt namn till drivrutins-standard
;
; ENDTODO
;;;;;;;;;;;;;;;;

#ifndef _TWI_
#define _TWI_


        .equ    TWBR_PRESCALE   = 72
        ; Bestämmer överföringshastigheten (SCLs frekvens) enligt
        ; F_SCL = F_CPU / (16 + 2 * TWBR * 4^prescaler[TWPS0-1 i TWSR])
        ; Här 100 kHz


;;;;;;;;;;;;;;;;
;   INIT_TWI
;;;;;;;;;;;;;;;;

INIT_TWI:
        ldi     r16, TWBR_PRESCALE
        sts     TWBR, r16
        lds     r16, TWSR
        andi    r16, $FC
        sts     TWSR, r16
        ret

;;;;;;;;;;;;;;;;
```

```
;   TWI_SEND
;;;;;;;;;;;;;;;;

TWI_SEND:
        lsl     r17
        call    START
        call    TWI_WAIT
        call    SEND_ADDR
        call    TWI_WAIT
        call    WRITE_BYTE
        call    TWI_WAIT
        call    STOP
        ret

;;;;;;;;;;;;;;;;
;   TWI_READ
;;;;;;;;;;;;;;;;

TWI_READ:
        lsl     r17
        ori     r17, $01
        call    START
        call    TWI_WAIT
        call    SEND_ADDR
        call    TWI_WAIT
        call    READ_BYTE
        call    TWI_WAIT
        call    STOP
        ret

;;;;;;;;;;;;;;;;
;   Subrutiner
;;;;;;;;;;;;;;;;

START:
        push    r18
        ldi     r18, (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)
        sts     TWCR, r18
        pop     r18
        ret

    ; ---------------

STOP:
        push    r18
        ldi     r18, (1<<TWINT)|(1<<TWEN)|(1<<TWSTO)
        sts     TWCR, r18
        pop     r18
        ret

    ; ---------------

SEND_ADDR:
        push    r16
```

```
        mov     r16, r17
        call    WRITE_BYTE
        pop     r16
        ret

    ; ---------------

TWI_WAIT:
        push    r18
    TWI_WAIT_LOOP:
        lds     r18,TWCR
        sbrs    r18,TWINT
        rjmp    TWI_WAIT_LOOP
        pop     r18
        ret

    ; ---------------

WRITE_BYTE:
        push    r18
        sts     TWDR, r16
        ldi     r18, (1<<TWINT)|(1<<TWEN)
        sts     TWCR, r18
        pop     r18
        ret

    ; ---------------

READ_BYTE:
        push    r18
        ldi     r18, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)
        sts     TWCR, r18
        call    TWI_WAIT
        lds     r16, TWDR
        pop     r18
        ret

    ; ---------------

#endif /* _TWI_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; switches.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;
;
; Modulen för hantering av DAvid-kortets knappar
;
; Använder just nu query-anrop för enskilda knappar, sätter Z-flaggan vid
knapptryck.
;
;;;;;;;;;;;;;;;;

; 0b RY+ RY- RX+ RX- LY+ LY- LX+ LX-
; 0b 0 0

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Gör rutiner för joysticksen. (DELVIS KLAR)
;       * Avkoda AD-signaler från dom båda joysticksens två axlar på ett bättre
sätt (förslag: 1byte - 2bits/axel för båda joysticksen genom tabell-intervall)
;       * Skriv eventuellt alternativ rutin för att kolla samtliga knappar, eller
avkoda SWITCHES.
;
; ENDTODO
;;;;;;;;;;;;;;;;

#ifndef _SWITCH_
#define _SWITCH_


    ; "Neutralvärde för joystick axlar"
        .equ    VAL_JOY_R_H     = $83       ; Höger joystick x-led
        .equ    VAL_JOY_R_V     = $7E       ; Höger joystick y-led
        .equ    VAL_JOY_L_H     = $80       ; Vänster joystick x-led
        .equ    VAL_JOY_L_V     = $7F       ; Vänster joystick y-led


;;;;;;;;;;;;;;;;
;   Knapp-queries
;;;;;;;;;;;;;;;;

RQ:
        sbis    PIND, SW_R
        sez
        ret


R1Q:
        call    SWITCHES        ; Flytta eventuellt ut och g�r ett gemensamt call
f�r samtliga queries? Eller ska dom kunna kallas separat?
        clz                     ; SWITCHES verkar s�tta Z-flaggan i n�got steg,
s� denna beh�vs efter. Mer r�tt med en sbrc innan. Eller kanske v�nda p� sbrs
under f�r f�rre instruktioner?
        sbrs    r16, SW_R1      ; Kollar om bit SW_R1 (0) i r16 (h�mtat fr�n
switches) �r 0 och s�tter d� Z-flaggan
        sez
```

```
        ret



R2Q:
        call    SWITCHES
        clz
        sbrs    r16, SW_R2
        sez
        ret

LQ:
        sbis    PIND, SW_L
        sez
        ret

L1Q:
        call    SWITCHES
        clz
        sbrs    r16, SW_L1
        sez
        ret

L2Q:
        call    SWITCHES
        clz
        sbrs    r16, SW_L2
        sez
        ret

JOY_RQ:
        call    SWITCHES
        clz
        sbrs    r16, JOY_R_SEL
        sez
        ret

JOY_LQ:
        call    SWITCHES
        clz
        sbrs    r16, JOY_L_SEL
        sez
        ret

    ; ---------------

SWITCHES:
        ldi     r17, ADDR_SWITCH
        call    TWI_READ
        ret


;;;;;;;;;;;;;;;;
;   Joystick
```

```
;;;;;;;;;;;;;;;;



READ_JOY_R_H:
        ldi     r16, (1<<REFS0)|(1<<ADLAR)|JOY_R_H      ; Väljer kanal för AD-
omvandlaren, med 5v referens-spänning och left adjust (8 bitar)
        sts     ADMUX, r16
        call    ADC_READ8
        ret

READ_JOY_R_V:
        ldi     r16, (1<<REFS0)|(1<<ADLAR)|JOY_R_V      ; Väljer kanal för AD-
omvandlaren, med 5v referens-spänning och left adjust (8 bitar)
        sts     ADMUX, r16
        call    ADC_READ8
        ret

READ_JOY_L_H:
        ldi     r16, (1<<REFS0)|(1<<ADLAR)|JOY_L_H      ; Väljer kanal för AD-
omvandlaren, med 5v referens-spänning och left adjust (8 bitar)
        sts     ADMUX, r16
        call    ADC_READ8
        ret

READ_JOY_L_V:
        ldi     r16, (1<<REFS0)|(1<<ADLAR)|JOY_L_V      ; Väljer kanal för AD-
omvandlaren, med 5v referens-spänning och left adjust (8 bitar)
        sts     ADMUX, r16
        call    ADC_READ8
        ret

    ; ---------------

ADC_READ8:                              ; Returnerar ett värde mellan 0-255 från
vald ADC-kanal till r16
        ;ldi        r16, (1<<REFS0)|(1<<ADLAR)|PC0
        ;sts        ADMUX, r16
        ldi     r16, (1<<ADEN)|7        ; Sätt AD-enable och prescaler till 128
(=> 125 kHz)
        sts     ADCSRA, r16

ADC_CONVERT:                            ; Starta omvandling
        lds     r16, ADCSRA
        ori     r16, (1<<ADSC)
        sts     ADCSRA, r16
ADC_BUSY:                               ; Vänta tills omvandling är klar
        lds     r16, ADCSRA
        sbrc    r16, ADSC
        jmp     ADC_BUSY
        lds     r16, ADCH
        ret
```

```
    ; ---------------


#endif /* _SWITCH_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; speaker.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
;   Rutiner för att spela ljud på kortets pizeoelektriska högtalare.
;
;   Genererar med hjälp av interrupts oscillerande på/av signaler på PB1 för att
hålla ljudet igång vid
;   en viss frekvens en satt tid.
;
;   Anropas lämpligtvis med hjälp av "PLAY_NOTE_X".
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Sekvenser av flera ljud.
;
; ENDTODO;
;;;;;;;;;;;;;;;;

#ifndef _SPEAKER_
#define _SPEAKER_

        ; F(Speaker Timer)  = 500 000 Hz

        .equ        NOTE_G  = 80       ; = F(ST)/F(G)*2 ; F(G) = 3135.963 Hz
        .equ        NOTE_A  = 71       ; = F(ST)/F(A)*2 ; F(A) = 3520.000 Hz
        .equ        NOTE_B  = 63       ; = F(ST)/F(B)*2 ; F(B) = 3951.066 Hz


        ; Längder för tonerna om dom ska vara 0.05 sekunder
        .equ        NOTE_G_005_H    = $01
        .equ        NOTE_G_005_L    = $3A
        .equ        NOTE_A_005_H    = $01
        .equ        NOTE_A_005_L    = $60
        .equ        NOTE_B_005_H    = $01
```

```
        .equ        NOTE_B_005_L    = $8B


;:::::::::::::::
;       Titel
;:::::::::::::::


TIMER2_INIT:
        push    r16
                                            ; Set data direction till output för
Speaker pin
        ;ldi    r16, (1<<SPEAKER)
    ;out    DDRB, r16
        sbi     DDRB, SPEAKER

        ldi r16, (1<<WGM21) ; CTC
        sts TCCR2A, r16
        ldi r16, (0<<CS22)|(1<<CS21)|(1<<CS20)  ; prescale 32
        sts TCCR2B, r16
        pop     r16
        ret

ENABLE_SPEAKER_INT:
        ldi r16, (1<<OCIE2A)            ; allow to interrupt
    sts TIMSK2, r16
        ret

DISABLE_SPEAKER_INT:
        ldi r16, (0<<OCIE2A)            ; allow to interrupt
    sts TIMSK2, r16
        ret

PLAY_NOTE_G:
        ldi     r16, NOTE_G
        sts     OCR2A, r16
        ldi r16, NOTE_G_005_H
        sts     NOTE_LENGTH_HIGH, r16
        ldi     r16, NOTE_G_005_L
        sts     NOTE_LENGTH_LOW, r16
        call    ENABLE_SPEAKER_INT
        ret

PLAY_NOTE_A:
        ldi     r16, NOTE_A
        sts     OCR2A, r16
        ldi r16, NOTE_A_005_H
        sts     NOTE_LENGTH_HIGH, r16
        ldi     r16, NOTE_A_005_L
        sts     NOTE_LENGTH_LOW, r16
        call    ENABLE_SPEAKER_INT
        ret

PLAY_NOTE_B:
```

```
        ldi     r16, NOTE_B
        sts     OCR2A, r16
        ldi r16, NOTE_B_005_H
        sts     NOTE_LENGTH_HIGH, r16
        ldi     r16, NOTE_B_005_L
        sts     NOTE_LENGTH_LOW, r16
        call    ENABLE_SPEAKER_INT
        ret

STOP_SPEAKER:
        cbi PORTB, PB1
        call    DISABLE_SPEAKER_INT
        ret

SPEAKER_TIMER_INT:
        push    r16
        in  r16, SREG
        push    r16

        call    TOGGLE_SPEAKER
    call    CHECK_NOTE_LENGTH

        pop r16
        out SREG, r16
        pop r16
        reti


TOGGLE_SPEAKER:
        sbis    PORTB, PB1
        rjmp    TOGGLE_SPEAKER_ON
        rjmp    TOGGLE_SPEAKER_OFF
TOGGLE_SPEAKER_ON:
        sbi PORTB, PB1
        rjmp    TOGGLE_SPEAKER_DONE
TOGGLE_SPEAKER_OFF:
    cbi PORTB, PB1
TOGGLE_SPEAKER_DONE:
        ret


CHECK_NOTE_LENGTH:
        push    r25
        push    r24
        lds     r24, NOTE_LENGTH_LOW
        lds     r25, NOTE_LENGTH_HIGH
        sbiw    r25:r24, 1
        breq    STOP_NOTE
        sts     NOTE_LENGTH_LOW, r24
        sts     NOTE_LENGTH_HIGH, r25
        rjmp    CHECK_NOTE_DONE
STOP_NOTE:
    call    STOP_SPEAKER
CHECK_NOTE_DONE:
```

```
    pop     r24
    pop     r25
    ret

#endif /* _SPEAKER_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; lcd.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klasén, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Modulen för utskrifter på 2x16 LCDn på DAvid.
;
; Omgjord för överföring med TWI-protokoll.
; Används primärt genom att lagra meddelanden i LINE i SRAM och använda
LINE_PRINT,
; eller hårdkodat meddelande i .db, anropat med LCD_FLASH_PRINT och meddelandet
laddat i Z-pekaren
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Gör om rutinerna för att funka med TWI (KLAR)
;       * Stöd för att skriva till båda raderna
;       * Ändra så att LCD_FLASH_PRINT skriver från ett valbart meddelande i
flash, skickat som argument (KLAR)
;       * (Ev. samma för LINE, eller bygg på med bättre rutiner för att skriva
saker dit.)
;       * Integrera knappstyrning?
;
; ENDTODO
;;;;;;;;;;;;;;;;

#ifndef _LCD_
#define _LCD_

        ; Display kommandon
        .equ    FN_SET  = $28   ;0b 001(DL)NF--
        .equ    DISP_ON = $0C   ;0b 00001DCB
        .equ    LCD_CLR = $01   ;0b 00000001
        .equ    E_MODE  = $06   ;0b 000001(I/D)S
        .equ    C_HOME  = $02   ;0b 0000001-
```

```
MESSAGE:                    ; Max 16 tecken, sträng som skrivs ut med
LCD_FLASH_PRINT
        .db     "HELLO WORLD", $00

;;;;;;;;;;;;;;;;
;   Rutiner
;;;;;;;;;;;;;;;;

;:: Utskrifts-rutiner ::

LCD_PRINT_HEX:                          ; r16 som indata, skriver ut ett hex-värde
för en byte i ascii över två rutor på displayen
        push    r17
        ldi     XH, HIGH(LINE)
        ldi     XL, LOW(LINE)

        call    GET_HIGH_NIBBLE
        call    CONVERT_BIN_TO_HEX_ASCII
        st      X+, r17                 ; 16-talet i hex lagrat som ascii i LINE+0

        call    GET_LOW_NIBBLE
        call    CONVERT_BIN_TO_HEX_ASCII
        st      X+, r17                 ; Entalet i hex lagrat som ascii i LINE+1
        ldi     r17, $00                ;
        st      X+, r17                 ; Insert nullbyte

        call    LINE_PRINT
        pop     r17
        ret

GET_HIGH_NIBBLE:
        mov     r17, r16
        swap    r17
        andi    r17, $0F                ; Ta in data från r16 höga nibble och
maska bort resten.
        ret

GET_LOW_NIBBLE:
        mov     r17, r16
        andi    r17, $0F
        ret

CONVERT_BIN_TO_HEX_ASCII:
        cpi     r17, $0A
        brmi    BELOW_TEN
TEN_OR_ABOVE:
        subi    r17, $09
        ori     r17, $40
        rjmp    CONV_DONE
BELOW_TEN:
        ori     r17, $30
CONV_DONE:
        ret
```

```
    ; ---------------

LCD_FLASH_PRINT:
        ;ldi    ZH, HIGH(MESSAGE*2)
        ;ldi    ZL, LOW(MESSAGE*2)
        ldi     XH, HIGH(LINE)
        ldi     XL, LOW(LINE)
TRANSFER_CHAR:
        lpm     r16, Z+
        st      X+, r16
        cpi     r16, $00
        brne    TRANSFER_CHAR
        call    LINE_PRINT
        ret

    ; ---------------

LINE_PRINT:
        push    r16
        push    ZH
        push    ZL
        call    LCD_HOME
        ldi     ZH, HIGH(LINE)
        ldi     ZL, LOW(LINE)
        call    LCD_PRINT
        pop     ZL
        pop     ZH
        pop     r16
        ret

    ; ---------------

LCD_PRINT:
        ld      r16, Z+
        cpi     r16, $00
        breq    PRINT_DONE
        call    LCD_ASCII
        rjmp    LCD_PRINT
PRINT_DONE:
        ret

    ; ---------------

LCD_ASCII:
        push    r18
        lds     r18, LCD_PORT
        sbr     r18, (1<<LCD_RS)
        sts     LCD_PORT, r18
        call    LCD_WRITE8
        pop     r18
        ret

    ; ---------------
```

```
LCD_COMMAND:
        push    r18
        lds     r18, LCD_PORT
        cbr     r18, (1<<LCD_RS)
        sts     LCD_PORT, r18
        call    LCD_WRITE8
        pop     r18
        ret

    ; ---------------

LCD_WRITE8:
        call    LCD_WRITE4
        swap    r16
        call    LCD_WRITE4
        ret

LCD_WRITE4:
        lds     r19, LCD_PORT
        mov     r18, r16
        andi    r18, $F0
        andi    r19, $0F
        add     r19, r18
        sts     LCD_PORT, r19
        call    PULSE_E
        call    DELAY
        ret

PULSE_E:
        call    LCD_E_UP
        call    DELAY
        call    LCD_E_DOWN
        ret

LCD_E_UP:
        push    r16
        lds     r16, LCD_PORT
        sbr     r16, (1<<LCD_E)
        call    LCD_WRITE
        pop     r16
        ret

LCD_E_DOWN:
        push    r16
        lds     r16, LCD_PORT
        cbr     r16, (1<<LCD_E)
        call    LCD_WRITE
        pop     r16
        ret

LCD_WRITE:
        sts     LCD_PORT, r16
        ldi     r17, ADDR_LCD
```

```
        call    TWI_SEND
        ret

;:: Display-styrning ::

LCD_COL:                                ; Kommande för att sätta cursorn till
position i r16 (värde från $00-$0F, för andra raden $40-4F
        ori     r16, $80
        call    LCD_COMMAND
        ret

    ; ---------------

LCD_ERASE:
        push    r16
        ldi     r16, LCD_CLR
        call    LCD_COMMAND
        pop     r16
        ret

    ; ---------------

LCD_HOME:
        push    r16
        ldi     r16, C_HOME
        call    LCD_COMMAND
        pop     r16
        ret

    ; ---------------

LCD_BACKLIGHT_ON:
        push    r16
        lds     r16, LCD_PORT
        sbr     r16, (1<<LCD_BL)
        ;sts     LCD_PORT, r16   ; (Göra denna i LCD_WRITE?)
        call    LCD_WRITE       ; med tom indata? command?
        pop     r16
        ret

    ; ---------------

LCD_BACKLIGHT_OFF:
        push    r16
        lds     r16, LCD_PORT
        cbr     r16, (1<<LCD_BL)
        ;sts     LCD_PORT, r16
        call    LCD_WRITE       ; med tom indata? command?
        pop     r16
        ret

;:: Initieringsrutiner ::

LINE_INIT:                              ; Fyller hela LINE med mellanslag (" ",
```

```
$20) för intitiering av skärmutskriften. Möjliggör utskrift efter en tom ruta (jmf
nullbyte som avbryter utskrift)
        ldi     ZH, HIGH(LINE)
        ldi     ZL, LOW(LINE)
        ldi     r16, $20
        clr     r17
LINE_CYCLE:                             ; Laddar in $20 i LINE+X fram tills
LINE+16
        cpi     r17, $10
        breq    LINE_CYCLE_DONE
        st      Z+, r16
        inc     r17
        rjmp    LINE_CYCLE
LINE_CYCLE_DONE:
        clr     r16
        st      z, r16                  ; Avslutar med nullbyte i LINE+16
        ret

    ; ---------------

LCD_INIT:
        clr     r16
        sts     LCD_PORT, r16
                            ; Tänd bakgrundsljuset
        call    LCD_BACKLIGHT_ON
                            ; Vänta på att displayen ska vakna
        call    DELAY

                            ; Initiering av 4-bits mode på displayen
        ldi     r16, $30
        call    LCD_WRITE4
        call    LCD_WRITE4
        call    LCD_WRITE4
        ldi     r16, $20
        call    LCD_WRITE4
                            ; Konfigurering
                            ; 4-bit mode, 2 line, 5x8 font
        ldi     r16, FN_SET
        call    LCD_COMMAND
                            ; Display on, cursor on, cursor blink
        ldi     r16, DISP_ON
        call    LCD_COMMAND
                            ; Clear display
        ldi     r16, LCD_CLR
        call    LCD_COMMAND
                            ; Entry mode: Increment cursor, no shift
        ldi     r16, E_MODE
        call    LCD_COMMAND
        ret

    ; ---------------

LCD_PORT_INIT:
                            ; Initiering av portriktningar på arduinon
```

```
        ldi     r16, $07    ;
        out     DDRB, r16   ; PORTB, bit 0-2 (RS, E och BGLT) till output
        ldi     r16, $F0    ;
        out     DDRD, r16   ; PORTD, bit 4-7 till output
        ret

    ; ---------------

#endif /* _LCD_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; led.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Modulen för hantering av DAvid-kortets lysdioder.
; Lagrar lampornas av/på status i LED_STATUS i sram, se memory.asm.
;
; Behöver utökas
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Övriga LEDs förutom rotary encoderns LEDs
;
; ENDTODO
;;;;;;;;;;;;;;;;

#ifndef _LEDS_
#define _LEDS_


;;;;;;;;;;;;;;;;
;   LED-rutiner
;;;;;;;;;;;;;;;;

; ROTLED_RED:
;       ldi     r17, ADDR_ROTLED; << 1) | 0
;       ldi     r16, $01                        ; Obs omv�nt r�d/gr�n fr�n
h�rdvarubeskrivning. Maska eventuellt med en byte i SRAM om LED f�r L1/L/L2 osv
ska anv�ndas
;       call    TWI_SEND
```

```
;       ret

ROTLED_RED:
        ldi     r17, ADDR_ROTLED
        lds     r16, LED_STATUS
        cbr     r16, (1<<LED_ROT1)
        sts     LED_STATUS, r16
        call    TWI_SEND
        ret

ROTLED_GREEN:
        ldi     r17, ADDR_ROTLED
        lds     r16, LED_STATUS
        cbr     r16, (1<<LED_ROT0)
        sts     LED_STATUS, r16
        call    TWI_SEND
        ret

ROTLED_BOTH:
        ldi     r17, ADDR_ROTLED
        lds     r16, LED_STATUS
        cbr     r16, (1<<LED_ROT1)|(1<<LED_ROT0)
        sts     LED_STATUS, r16
        call    TWI_SEND
        ret

ROTLED_OFF:
        ldi     r17, ADDR_ROTLED
        lds     r16, LED_STATUS
        sbr     r16, (1<<LED_ROT1)|(1<<LED_ROT0)
        sts     LED_STATUS, r16
        call    TWI_SEND
        ret

    ; ---------------

#endif /* _LEDS_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; 7seg.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; Modulen för manipulering av DAvid-kortets båda 7-segments displayer.
```

```
;
; Använder en lookup-tabell för att översätta ett värde mellan 0 och F till 7-seg
format.
;
; Anropas med RIGHT8_WRITE/LEFT8_WRITE som tar ett inargument i r16 och skriver ut
den låga nibblen.
;
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
; TODO:
;       * Argument för att styra till punkten
;       * Gör en rutin för att skriva ut hel byte(i hex) till båda displayerna
(bra för felsökning?)
;
; ENDTODO
;;;;;;;;;;;;;;;;

#ifndef _7_SEG_
#define _7_SEG_

;;;;;;;;;;;;;;;;
;   Tabell
;;;;;;;;;;;;;;;;

TAB_7SEG:
        .db     $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F, $77, $7C, $39,
$5E, $79, $71
                                    ; LOOKUP-tabell f�r 0-F i 7-seg (pgfedcba),
p=0

;;;;;;;;;;;;;;;;
;   Rutiner
;;;;;;;;;;;;;;;;

RIGHT8_WRITE:                       ; Beh�ver indata i r16, kommer enbart kolla
l�g nibble
        andi    r16, $0F            ; 0000xxxx
        call    LOOKUP_7SEG
        ldi     r17, ADDR_RIGHT8
        call    TWI_SEND
        ret

    ; ---------------

LEFT8_WRITE:                        ; Beh�ver indata i r16, kommer enbart kolla
l�g nibble
        andi    r16, $0F
        call    LOOKUP_7SEG
        ldi     r17, ADDR_LEFT8
        call    TWI_SEND
        ret
```

```
    ; ---------------

LOOKUP_7SEG:                        ; Tar BIN/HEX v�rde i r16 (0-15) och
omvandlar till r�tt 7-seg symbol (utan punkt)
        push    ZL
        push    ZH
        ldi     ZH,HIGH(TAB_7SEG*2)
        ldi     ZL,LOW(TAB_7SEG*2)
        add     ZL,r16
        lpm     r16,Z
        pop     ZH
        pop     ZL
        ret

    ; ---------------

#endif /* _7_SEG_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; flash_messages.asm
;
; TSIU51 -  Mikrodatorprojekt
; Author :  Oskar Lundh, osklu130, Di1b
;           Johan Klas�n, johkl473, Di1b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;
;
;   Denna fil sammlar olika strängar eller hårdkodade minnessekvenser
;   som behöver lagras i flashminnet.
;
;;;;;;;;;;;;;;;;


#ifndef _FLASH_MESSAGES_
#define _FLASH_MESSAGES_


;;;;;;;;;;;;;;;;
;   LCD-meddelanden
;;;;;;;;;;;;;;;;

            ; OBS: Max 16 tecken, strängar som skrivs ut med LCD_FLASH_PRINT

WELCOME_MSG:
        .db     "WELCOME TO PONG!", $00
READY_MSG:
        .db     "HIT L/R TO START", $00
P1_POINT_MSG:
```

```
        .db     "PLAYER 1 SCORES!", $00
P2_POINT_MSG:
        .db     "PLAYER 1 SCORES!", $00
P1_WINS_MSG:
        .db     " PLAYER 1 WINS! ", $00
P2_WINS_MSG:
        .db     " PLAYER 2 WINS! ", $00

;;;;;;;;;;;;;;;;
;   Bilder till DA-matrix
;;;;;;;;;;;;;;;;

PIC:
        .db     0b00000000, 0b00000000
        .db     0b00000110, 0b01100000
        .db     0b00000110, 0b01100000
        .db     0b00000000, 0b00000000
        .db     0b00100000, 0b00000100
        .db     0b00011000, 0b00011000
        .db     0b00000111, 0b11100000
        .db     0b00000000, 0b00000000

PIC_RAM: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W",
"W", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W",
"W", " ", " "
        .db     " ", "W", "W", "W", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "

;;;;;;;;;;;;;;;;
;   Fireworks-animering
;;;;;;;;;;;;;;;;

FW_ANIM1: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
```

```
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "

FW_ANIM2: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "B",
"W", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
FW_ANIM3: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "B",
"W", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "B", " ",
"B", "W", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "B",
"w", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "

FW_ANIM4: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", " ", "B", "
", "W", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "B",
"B", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "B", "B", " ",
"B", "B", "W"
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "B",
"B", " ", " "
```

```
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", " ", "B", "
", "W", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "W", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "

FW_ANIM5: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", " ", "B", "
", "B", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "B", " ", " ", "
", "B", "B"
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", " ", "B", "
", "B", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "

FW_ANIM6: ; W = vit, R = röd, G = grön, B = blå
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", " ", " ", "
", " ", "B"
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "B", "
", " ", " "
        .db     " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", "
", " ", " "


#endif /* _FLASH_MESSAGES_ */
;;;;;;;;;;;;;;;;
;   End of file
;;;;;;;;;;;;;;;;
```