ADVANCED LCD DRIVER V1.0

1. The way this program coded!
2. About this LCD Driver.
3. Macro Manual

The way this program coded:

For Clarity, This program coded modular. Program classified in logical parts And each logical part has A related .asm file and a related .inc file, contains the related code.
For example all LCD routines placed in LCD.asm and all LCD definitions Placed in LCD.inc

*The Entry file for this Project is MAIN.asm so if you want to build the project Set the MAIN.asm as entry file.

*For all macros an underline '_' prefix attached to the macro name.
Like: _8reg2ram (loads one byte from sram to io registers.)

*For all constants two underlines '__' used as prefix .
Like: __LCD_Tick

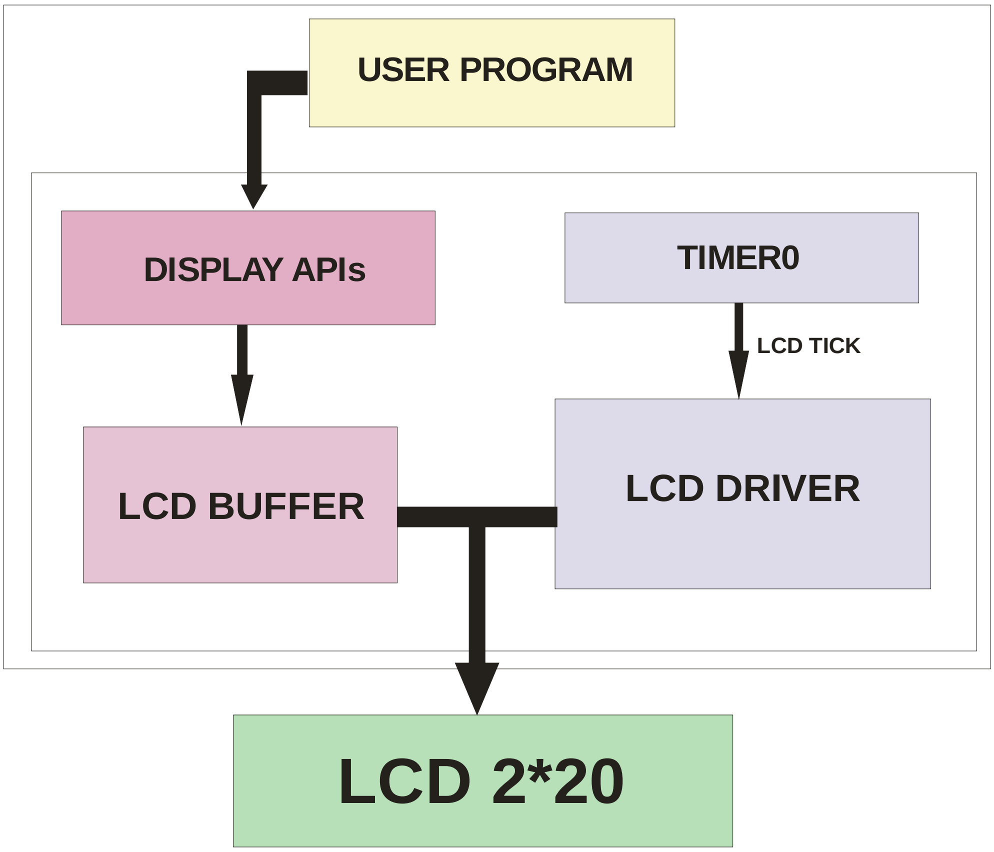
*all macros ,used in this program are placed in MACRO.asm.

About this LCD Driver :

In typical applications when we want to display a string off charcters on an LCD, We have to set the 1. lcd_address 2.wait until the LCD is busy, 3.load one charcter To LCD. For the whole string we do this sequence in one time. this sequence takes A lot of time (Apx. 800uS for 20 chars) .and it 's not good for time critical programs.

in Advance LCD Driver instead of doing the whole sequence at a time, the program generates An LCD_Tick. On each tick it sends A character to the LCD.
So displaing A 20char string takes 20*Time(lcd_tick) –in this application LCD_TICKS Generated each 1mS . -
Also because of buffering method of display the user can change the display contents so Fast without any need to know about lcd driver.
Some formatting functions provided to display strings and numbers on the LCD.
To demonstrate functionality of the driver A sample program -32bit chronometer- Placed in the soft timer routine.

*For more information on:
-LCDs refer to LCD manual . LCD_Manual.pdf
-Soft timers , search for SOFT TIMERS at avrfreaks.net .
-Delays , search for delays at avrfreaks.net.
-macro programming,search AVR studio's help for assembler directives.
- All mathematical routines are from atmel's app. note math32.



Macros Manual:

_INIT_STACK no param
Initializes the stack pointer at the top of sram.
Uses: ZH:ZL

_8RAM2REG @0=RAM_ADDRESS,@1=REGISTER
Loads one byte from sram to registers r0..r31

_8REG2RAM @0=RAM_ADDRESS,@1=REGISTER
Loads one byte from registers r0..r31 to sram

_16RAM2REG @0=RAM_ADDRESS,@1=REGISTER_High,@2=REGISTER_Low
Loads two bytes from sram to registers r0..r31
Uses: YH:YL

_16REG2RAM @0=RAM_ADDRESS,@1=REGISTER_High,@2=REGISTER_Low
Loads two bytes from registers r0..r31 to sram
Uses: YH:YL

_32RAM2REG @0=RAM_ADDRESS,@1=REGISTER_byte3,@2=REGISTER_byte2 ,@3=REGISTER_byte1,@4=REGISTER_byte0
Loads four bytes from sram to registers r0..r31
Uses: YH:YL

_16REG2RAM @0=RAM_ADDRESS,@1=REGISTER_byte3,@2=REGISTER_byte2 ,@3=REGISTER_byte1,@4=REGISTER_byte0
Loads four bytes from registers r0..r31 to sram
Uses: YH:YL

_IF_BIT_SET_CALL @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was set,will absolute-call to the routine.

_IF_BIT_SET_RCALL @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was set,will relative call to the routine.

_IF_BIT_SET_JUMP @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was set,will absolute-jump to the routine.

_IF_BIT_SET_RJUMP @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was set,will relative jump to the routine.

_IF_BIT_NOTSET_CALL @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was clear ,will absolute-call to the routine.

_IF_BIT_NOTSET_RCALL @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was clear ,will relative call to the routine.

_IF_BIT_NOTSET_JUMP @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was clear ,will absolute-jump to the routine.

_IF_BIT_NOTSET_RJUMP @0=Register,@1=bit_number,@2=Destination_address
Checks a single bit in a register and if it was clear ,will relative jump to the routine.

_SKIP_IF_BIT_SET @0=Register,@1=bit_number
Checks a single bit in a register and if it was set ,will skip next instruction.

_SKIP_IF_BIT_NOTSET @0=Register,@1=bit_number
Checks a single bit in a register and if it was clear ,will skip next instruction.

_SBR @0=Register,@1=Bit_number
SBR type macro for lower 16 registers.
@0= R0...R15 @1=0…7
Uses T flag in sreg.

_CBR @0=Register,@1=Bit_number
CBR type macro for lower 16 registers.
@0= R0...R15 @1=0…7
Uses T flag in sreg.

_WAIT_uS @0=CPU_frequency,@1=delay_duration
FOR-NEXT type loop , loops for delay_duration, has a resolution of micro-seconds.
Uses: R26

_WAIT_5uS @0=CPU_frequency,@1=delay_duration
FOR-NEXT type loop , loops for delay_duration, has a resolution of 5micro-seconds.
Uses: R26 , R25

_WAIT_10uS @0=CPU_frequency,@1=delay_duration
FOR-NEXT type loop , loops for delay_duration, has a resolution of 10micro-seconds.
Uses: R26 , R25

_WAIT_mS @0=CPU_frequency,@1=delay_duration

FOR-NEXT type loop , loops for delay_duration, has a resolution of mili-seconds.
Uses: R26 , R25

_LCD_DISPLAY_Msg

@0=starting address of A string. It uses the Z pointer as a pointer To the string placed in flash memory.

@1=line of LCD(1,2,3or 4) in this program __LCD_LINE1 and __LCD_LINE2 used as constants for this parameter.

@2= startinglocation of line. In this program 0 … 20.

@3=message terminator or message length. If parameter equals to __NULL=0xFF or __ZERO=0x00 the message considers as a Null_ended or zero_ended string, otherwise this parameter will Considers as message length. In this program 1… to 20.

Uses:

Z pointer as a pointer to the array of charactrs.
Y pointer uses to trace the current position of display buffer.
R16 , R17

* LOAD_TERMINATEDstr_FLASH2LCDBUFFER and
LOAD_CHARstr_FLASH2LCDBUFFER
Routines are called inside this macro.

_LCD_DISPLAY_NUM

@0=length of the source binary number. It accepts __16bit =0, __24bit=1 and __32bit=2 for 16,24 and 32 bit numbers

@1=sign , use __SIGNED or __UNSIGNED constants. If you use __SIGNED param for a negative number a neg. sign '-' will add As a prefix to your number.

@2=line of lcd . like display message routine.

@3=starting location of display. Like display message routine.

@4=number of digits at the left side of decimal point.

@5= MSB of binary number

@6

@7

@8= LSB of binary number

For 16bit numbers use only @5 and @6
For 24bit numbers use only @5 , @6 and @7
For 32bit numbers use @5,@6,@7 and @8

Uses:

R16,R17,R18,R19,R20,R21,R22,R23,R24 YH:YL AND ZH:ZL

* These routines are called from this macro

BCD2LCDBUFFER() loads converted number to the lcd buffer.	
BIN2BCD16()	for 16 bit bin to pBCD conversion
BIN3BCD16()	for 24 bit bin to pBCD conversion
BIN4BCD16()	for 32 bit bin to pBCD conversion
NEG16	
NEG24	
NEG32	

```
.CSEG

.ORG    $00
    RJMP SYSTEM_INI
.ORG    $02
    RETI
.ORG    $04
    RETI
.ORG    $06
    RETI
.ORG    $08
    RETI
.ORG    $0A
    RETI
.ORG    $0C
    RETI
.ORG    $0E
    RETI
.ORG    $10
    RETI
.ORG    $12
    RJMP ISR_TOV0
.ORG    $14
    RETI
.ORG    $16
    RETI
.ORG    $18
    RETI
.ORG    $1A
    RETI
.ORG    $1C
    RETI
.ORG    $1E
    RETI
.ORG    $20
    RETI
.ORG    $22
    RETI
.ORG    $24
    RETI
.ORG    $26
    RETI
.ORG    $28
    RETI






.NOLIST
.INCLUDE  "M16DEF.INC"
.LIST
.INCLUDE  "MAIN.INC"
.INCLUDE  "MACRO.INC"
.INCLUDE  "DOC.INC"
.INCLUDE  "TIMER.INC"
```

```
.INCLUDE  "LCD.INC"
.INCLUDE  "SRAM.INC"


.INCLUDE  "INT_VECTOR.ASM"
.INCLUDE  "SYSINI.ASM"
.INCLUDE  "TIMER.ASM"
.INCLUDE  "LCD.ASM"
.INCLUDE  "MATH.ASM"

MAIN:

     _IF_BIT_SET_RCALL        STIMER_FLAGS,_STIMER1_CM,ON_STIMER1

     _IF_BIT_SET_RCALL        SYSTEM_FLAGS,__LCD_TICK,LOAD_CHAR_BUFFER2LCD

     RJMP MAIN






.CSEG


SYSTEM_INI:


    _INIT_STACK  ;INITIATING STACK AT THE TOP OF SRAM

; FOR PIN MAPPING REFER TO DOC.INC
;I/O PORTS CONFIGURATIONS
;PORT A        LCD DATA BUS.ALL OUTPUTS
     LDI       TEMP,$FF
     OUT       DDRA,TEMP
     LDI       TEMP,0
     OUT       PORTA,TEMP
;PORT B        LCD PINS
     LDI       TEMP,0B11100000
     OUT       DDRB,TEMP
     LDI       TEMP,0B00011111
     OUT       PORTB,TEMP
;PORT C
     LDI       TEMP,0B00000000
     OUT       DDRC,TEMP
     LDI       TEMP,0B11111111
     OUT       PORTC,TEMP
;PORT D
     LDI  TEMP,0B00000000
     OUT  DDRD,TEMP
     LDI      TEMP,0B11111111
     OUT  PORTD,TEMP


     RCALL     LCD_INIT

; CLEARING SRAM     (FIRST 255BYTES)
```

```
     LDI         TEMP,$0
     CLR         YH
     CLR         ZH
     LDI         YL,(DATARAM_START)
     LDI         ZL,(DATARAM_END)
CLEAR_DATARAM:
     ST          Y+,TEMP
     CP          YL,ZL
     BRNE        CLEAR_DATARAM

;CLEAR R0-R25
     CLR         R0
     CLR         R29
     LDI         R28,1
     LDI         R25,24
__CLEAR_REGISTERS:
     ST          Y+,R0
     DEC         R25
     BRNE        __CLEAR_REGISTERS
     CLR         R31
     CLR         R30
     CLR         R29
     CLR          R28
     CLR          R27
     CLR          R26


;**********************************************************
;    THIS PART IS FOR TESTING LCD ROUTINES.
;    TO TEST EACH FUNCTION DISABLE OTHER ROUTINES,
;    ALSO DISABLE THE CHRONOMETER PART PLACED IN SOFT-TIMER#1
;    ROUTINE.
;    TO UNDERSTAND FUNCTION-TYPE MACROS REFER TO
;    MACROS MANUAL(README.PDF)
;**********************************************************



;    _LCD_DISPLAY_MSG    str_MESSAGE1,__LCD_LINE1,0,20
;    _LCD_DISPLAY_MSG    str_MESSAGE3,__LCD_LINE2,0,__NULL
;    _LCD_DISPLAY_MSG    str_MESSAGE4,__LCD_LINE1,0,__ZERO


;    16BIT NUMBER
;==========================================================
;    LDI       R16,LOW(12345)
;    LDI       R17,HIGH(12345)
;    MOV       R3,R17
;    MOV       R2,R16
;    _LCD_DISPLAY_NUM         __16BIT,__UNSIGNED,__LCD_LINE1,0,4,R3,R2


;    24BIT NUMBER
;==========================================================
;    LDI       R16,LOW(12345678)
;    LDI       R17,HIGH(12345678)
;    LDI       R18,BYTE3(12345678)
;    MOV       R4,R18
;    MOV       R3,R17
;    MOV       R2,R16
;    _LCD_DISPLAY_NUM         __24BIT,__UNSIGNED,__LCD_LINE1,0,4,R4,R3,R2

;    32BIT NUMBER
;==========================================================
;    LDI       R16,LOW(-123456789)
;    LDI       R17,HIGH(-123456789)
;    LDI       R18,BYTE3(-123456789)
;    LDI       R19,BYTE4(-123456789)
;    MOV       R5,R19
;    MOV       R4,R18
;    MOV       R3,R17
;    MOV       R2,R16
;    _LCD_DISPLAY_NUM         __32BIT,__SIGNED,__LCD_LINE1,0,1,R5,R4,R3,R2


;**********************************************************
;
;**********************************************************

;    INITIATING THE 32BIT CHRONOMETER
;***DISABLE THIS PART IF YOU WANT TO TEST OTHER FUNCTIONS***
;==========================================================
;    ACTIVATING SOFT-TIMER#1 TO GENERATE 100mS TICKS FOR
;    THE CHRONOMETER PART.
     LDI       TEMP,(100-1)
     _8REG2RAM       RAM_STIMER1_PV,TEMP
     _SBR      STIMER_FLAGS,_STIMER1_EN
;    STARTING VALUE FOR CHRONOMETER=0
     CLR       R16
     _32REG2RAM      RAM_CHRONOMETER,R16,R16,R16,R16
     _LCD_DISPLAY_MSG     str_CHRONO_MSG,__LCD_LINE1,0,__NULL
     _LCD_DISPLAY_MSG     str_CHRONO_LINE,__LCD_LINE2,0,__NULL



; TIMER0 SETTINGS
     LDI  TEMP,3
     OUT  TCCR0,TEMP   ; TIMER0 PRESCALER CLKSRC/64=>1/16000000*64*250=1mS
                            ; TIMER0 WILL START TO COUNTING FROM WHEN YOU SET
                            ; THE VALUE OF TCCR0.
     LDI  TEMP,(0XFF-250+1)
     OUT  TCNT0,TEMP   ;*LOADS TIMER0 FOR 1 ms
     LDI       TEMP,(1<<TOIE0)
     OUT  TIMSK,TEMP
     SEI       ;GLOBAL INTERRUPT FLAG SETS.

     RJMP MAIN






.CSEG
;***********************************************************************
```

```
;* TIMER0 INTERRUPT HANDLER ROUTINE
;***********************************************************************
ISR_TOV0:
     IN       R1,SREG
     PUSH     TEMP
     LDI      TEMP,(255-250+1)
     OUT      TCNT0,TEMP
     PUSH     R17
     PUSH     R31
     PUSH     R30
CHECK_STIMER1:
     _IF_BIT_NOTSET_RJUMP    STIMER_FLAGS,_STIMER1_EN,END_OF_ISR_TOV0
     _8RAM2REG          RAM_STIMER1_CV,R16
     _8RAM2REG          RAM_STIMER1_PV,R17
     CP       R16,R17
     BRLO     INCREASE_STIMER1
     CLR       R16
     _SBR      STIMER_FLAGS,_STIMER1_CM
     RJMP SAVE_STIMER1
INCREASE_STIMER1:
;================
     INC      R16
SAVE_STIMER1:
;==============
     _8REG2RAM       RAM_STIMER1_CV,R16
END_OF_ISR_TOV0:
     _SBR      SYSTEM_FLAGS,__LCD_TICK

     POP      R30
     POP      R31
     POP      R17
     POP       TEMP
     OUT      SREG,R1
     RETI





;***********************************************************
;    SOFT TIMER#1
;***********************************************************
ON_STIMER1:
     _CBR      STIMER_FLAGS,(_STIMER1_CM)

     _32RAM2REG       RAM_CHRONOMETER,R5,R4,R3,R2
     LDI       R20,1
     ADD       R2,R20
     CLR       R20
     ADC       R3,R20
     ADC       R4,R20
     ADC       R5,R20
     _32REG2RAM       RAM_CHRONOMETER,R5,R4,R3,R2

     _LCD_DISPLAY_NUM         __32BIT,__UNSIGNED,__LCD_LINE2,2,9,R5,R4,R3,R2

END_OF_ON_STIMER1:
     RET
```

```
;**********************************************************
;    LCD INITIALIZATION ROUTINE (8BIT MODE)
;    FOR MORE INFORMATION REFER TO LCD DATASHEETS.
;    ALL CONSTANTS & VARIABLES DEFINED IN **LCD.INC** FILE.
;**********************************************************

LCD_INIT:

;    _WAIT_mS  CPU_FREQUENCY,15
;    INSTEAD OF USING A 15mS DELAY ,SET THE STARTUP TIMING (CKSEL=1110,SUT=10)
;    START-UP TIME:258CK+64mS
     LDI       LCD_REGISTER,__LCD_INIT_CODE
     RCALL     LCD_WRITE_INST

     _WAIT_mS    CPU_FREQUENCY,5

     LDI       LCD_REGISTER,__LCD_INIT_CODE
     RCALL     LCD_WRITE_INST

     _WAIT_mS    CPU_FREQUENCY,1

     LDI       LCD_REGISTER,__LCD_INIT_CODE
     RCALL     LCD_WRITE_INST

     RCALL     LOOP_IF_LCD_BUSY
     LDI       LCD_REGISTER,(__LCD_8BIT_INTERFACE | __LCD_2LINE | __LCD_5x8_MATRIX)
     RCALL     LCD_WRITE_INST

     RCALL     LOOP_IF_LCD_BUSY
     LDI       LCD_REGISTER,__LCD_OFF
     RCALL     LCD_WRITE_INST

     RCALL     LOOP_IF_LCD_BUSY
     LDI       LCD_REGISTER,__LCD_CLEAR
     RCALL     LCD_WRITE_INST

     RCALL     LOOP_IF_LCD_BUSY
     LDI       LCD_REGISTER,__LCD_ON ;| __LCD_SHOW_BLINK
     RCALL     LCD_WRITE_INST

     RCALL     CHAR_GENERATOR

     RET
;**********************************************************
;    FUNCTION: LCD_WRITE_INST
;    USED TO SET THE LCD ADDRESS...
;    REGISTER USED: R18 AS LCD REGISTER
;
;**********************************************************
LCD_WRITE_INST:
     CLI
     OUT       PORTA,LCD_REGISTER
     CBI       PORTB,__LCD_RW
     CBI       PORTB,__LCD_DC
     SBI       PORTB,__LCD_STROBE
     NOP
     NOP
     NOP
     CBI       PORTB,__LCD_STROBE
     SEI
     RET
;**********************************************************
;    FUNCTION: LCD_WRITE_DATA
;    USED TO SEND DATA TO LCD.
;    REGISTER USED: R18 AS LCD REGISTER
;
;**********************************************************
LCD_WRITE_DATA:
     CLI
     OUT       PORTA,LCD_REGISTER
     CBI       PORTB,__LCD_RW
     SBI       PORTB,__LCD_DC
     SBI       PORTB,__LCD_STROBE
     NOP
     NOP
     NOP
     CBI       PORTB,__LCD_STROBE
     SEI
     RET
;**********************************************************
;    FUNCTION: LOOP_IF_LCD_BUSY
;    USED TO CHECK THE STATUS OF BUSY FLAG.
;    REGISTER USED: R16 AS TEMP
;
;**********************************************************
LOOP_IF_LCD_BUSY:
     CLR       TEMP
     OUT       PORTA,TEMP
     OUT       DDRA,TEMP
BUSY_LOOP:
     CLI
     SBI       PORTB,__LCD_RW
     CBI       PORTB,__LCD_DC
     SBI       PORTB,__LCD_STROBE
     NOP
     NOP
     IN        TEMP,PINA
     SEI
     NOP
     ANDI TEMP,0B10000000 ;BIT No.7 IS FOR BUSY FLAG
     BRNE BUSY_LOOP
     CBI       PORTB,__LCD_STROBE
     SER       TEMP
     OUT       DDRA,TEMP
     RET

;**********************************************************
;    FUNCTION: LOAD_TERMINATEDstr_FLASH2LCDBUFFER
;
;    USED TO LOAD NULL_TERMINATED OR ZERO_TERMINATED
;    STRINGS TO DISPLAY BUFFER.YOU CAN ADD OTHER TERMINATORS
;    TO YOUR CODE.
;    REGISTER USED: R16,R17,YH:YL,ZH:ZL
;    CYCLES: FOR A 20CHAR STRING IT TAKES 189CYCLES
;              WITHOUT RET.AT 16MHz IT TAKES~12uS
;    NOTE: THIS FUNCTION USED BY _LCD_DISPLAY_MSG MACRO.
;**********************************************************
LOAD_TERMINATEDstr_FLASH2LCDBUFFER:
     ADD      YL,R16     ; BUFFER POINTER
     CLR      R16
     ADC      YH,R16
trmstr_LOAD_char:
     LPM       R16,Z+
     CP        R16,R17
     BREQ END_OF_trmstr
     ST       Y+,TEMP
     RJMP     trmstr_LOAD_char
END_OF_trmstr:
     RET
;**********************************************************
;    FUNCTION: LOAD_CHARstr_FLASH2LCDBUFFER
;
;    USED TO LOAD PART OF ARRAY OF CHARS TO DISPLAY
;    BUFFER.YOU HAVE TO SPECIFY THE LENTH OF ARRAY.
;    REGISTER USED: R16,R17,YH:YL,ZH:ZL
;    CYCLES: FOR A 20CHAR ARRAY IT TAKES 182CYCLES
;              WITHOUT RET.AT 16MHz IT TAKES~12uS
;    NOTE: THIS FUNCTION USED BY _LCD_DISPLAY_MSG MACRO.
;**********************************************************
LOAD_CHARstr_FLASH2LCDBUFFER:
     ADD      YL,R16     ; BUFFER POINTER
     CLR      R16
     ADC      YH,R16
charstr_LOAD_char:
     LPM       R16,Z+
     DEC       R17
     ST       Y+,TEMP
     BREQ END_OF_charstr
     RJMP     charstr_LOAD_char
END_OF_charstr:
     RET
;**********************************************************
;    FUNCTION: BCD2LCDBUFFER
;
;    USED TO FORMAT A PACKED-BCD NUMBER AND THEN LOAD
;    IT TO DISPLAY BUFFER.
;    NUMERICAL FORMATING INCLUDES ADDING NEGETIVE SIGN
;    AND DECIMAL POINT TO THE NUMBER.ALSO IT CONVERTS THE
;    NUMERIC VALUE TO ASCII VALUE TO DISPLAY IT ON THE LCD.
;
;    REGISTER USED: R16,R17,R18,R19,YH:YL,ZH:ZL
;    CYCLES: FOR A 10DIGIT SIGNED NUMBER IT TAKES 130CYCLES
;              WITHOUT RET.AT 16MHz IT TAKES~8uS
;    NOTE: THIS FUNCTION USED BY _LCD_DISPLAY_NUM MACRO.
;**********************************************************
BCD2LCDBUFFER:
     ADD      YL,R16     ; BUFFER POINTER
     CLR      R16
     ADC      YH,R16
     BLD       R17,7
     LDI       R16,'-'
     _SKIP_IF_BIT_NOTSET      R17,7
     ST        Y+,R16
     CBR       R17,(1<<7)
NUM2BUF_LOOP:
     LD        R18,Z
     ANDI R18,0XF0
     SWAP R18
     SUBI R18,-0X30
     ST        Y+,R18
     DEC       R17
     BRNE NUM2BUF_HiNIB
     LDI       R16,'.'
     ST        Y+,R16
NUM2BUF_HiNIB:
     LD        R18,Z
     ANDI R18,0X0F
     SUBI R18,-0X30
     ST        Y+,R18
     DEC       R17
     BRNE NUM2BUF_LowNIB
     LDI       R16,'.'
     ST        Y+,R16
NUM2BUF_LowNIB:
     SBIW ZH:ZL,1
     DEC       R19
     BRNE NUM2BUF_LOOP
END_OF_BCD:
     RET

;**********************************************************
;    FUNCTION: LOAD_CHAR_BUFFER2LCD
;
;    USED TO SEND A SINGLE BYTE FROM DISPLAY-BUFFER TO
;    LCD.THIS FUNCTION CALLS BY EACH LCD-TICK AT 1mS AND
;    AFTER 40mS IT SCANS THE ENTIRE LCD WINDOW(FOR 2*40 LCD)
;    AND THEN IT STARTS FROM FIRST POSITION OF LCD WINDOW.
;    YOU CAN USE OTHER TICKS BUT 1mS IS GOOD ENOUGH.

;    REGISTER USED: R16(TEMP),R17,R18,YH:YL
;    CYCLES: IT TAKES 120CYCLES
;              WITHOUT RET.AT 16MHz IT TAKES~8uS
;    NOTE:
;**********************************************************
LOAD_CHAR_BUFFER2LCD:
     LDI      YH,HIGH(RAM_DISPLAY_BUFFER_LINE1)
     LDI      YL,LOW(RAM_DISPLAY_BUFFER_LINE1)
     LDS      R17,RAM_LCD_POINTER
     ADD      YL,R17
     CLR      TEMP
     ADC      YH,TEMP
LCD_POINTER_AT_LINE1:
     CPI      R17,20
     BRSH     LCD_POINTER_AT_LINE2
     LDI      LCD_REGISTER,__LCD_LINE1_ADR
     ADD      LCD_REGISTER,R17
     CALL LOOP_IF_LCD_BUSY
     CALL LCD_WRITE_INST
```

```
        LD       LCD_REGISTER,Y
        RCALL    LOOP_IF_LCD_BUSY
        RCALL    LCD_WRITE_DATA
        INC      R17
        RJMP     SAVE_DISP_POINTER
LCD_POINTER_AT_LINE2:
        CPI      R17,40
        BREQ     RESET_DISP_POINTER
        LDI      LCD_REGISTER,(__LCD_LINE2_ADR-20)
        ADD      LCD_REGISTER,R17
        CALL LOOP_IF_LCD_BUSY
        CALL LCD_WRITE_INST
        LD       LCD_REGISTER,Y
        RCALL    LOOP_IF_LCD_BUSY
        RCALL    LCD_WRITE_DATA
        INC      R17
        RJMP     SAVE_DISP_POINTER
RESET_DISP_POINTER:
        LDI      R17,0
SAVE_DISP_POINTER:
        STS      RAM_LCD_POINTER,R17
        _CBR     SYSTEM_FLAGS,__LCD_TICK
        RET



;*******************************************************
;       FUNCTION: CHAR_GENERATOR
;
;       USED TO GENERATE SPECIAL CHARACTERS (MAXIMUM 8 CHARS)
;       AND LOAD THEM TO CG_RAM OF LCD.

;       REGISTER USED: R17,R18(LCD_REGISTER),ZH:ZL
;       NOTE:
;*******************************************************
CHAR_GENERATOR:
        LDI      ZH,HIGH(USER_CHAR*2)
        LDI      ZL,LOW(USER_CHAR*2)
        LDI       R17,__LCD_CGA
CG_LOOP:
        MOV       LCD_REGISTER,R17
        RCALL     LOOP_IF_LCD_BUSY
        RCALL     LCD_WRITE_INST
        LPM       LCD_REGISTER,Z+
        RCALL     LOOP_IF_LCD_BUSY
        RCALL     LCD_WRITE_DATA
        INC       R17
        CPI       R17,128
        BRLO CG_LOOP
        RET

USER_CHAR:
.DB       0X00,0X00,0X00,0X00,0X00,0X00,0X00,0X00
.DB       0X00,0X18,0X1C,0X1C,0X0E,0X06,0X02,0X01
.DB       0X00,0X03,0X07,0X07,0X0E,0X0C,0X08,0X10
.DB       0X01,0X02,0X06,0X0E,0X1C,0X1C,0X18,0X00
.DB       0X10,0X08,0X0C,0X0E,0X07,0X07,0X03,0X00
.DB       0X1C,0X14,0X1C,0X00,0X00,0X00,0X00,0X00
.DB       0X07,0X03,0X13,0X1F,0X1F,0X13,0X03,0X07
```

```
.DB      0X1C,0X18,0X19,0X1F,0X1F,0X19,0X18,0X1C


str_MESSAGE1:
.DB  "**  HELLO WORLD!! **",0XFF
str_MESSAGE2:
.DB "THIS IS A TEST PROG.",0XFF
str_MESSAGE3:
.DB "*NULL ENDED STRING.*",0XFF
str_MESSAGE4:
.DB "*ZERO ENDED STRING.*",0X00
str_CHRONO_LINE:
.DB  "<<            Sec.>>",0XFF
str_CHRONO_MSG:
.DB  "*32BIT CHRONOMETER!*",0XFF
```

```
;**** A P P L I C A T I O N   N O T E   A V R ? ? ? ************************
;*
;* Title:      32-bit Arithmetic Routines with Macrolibrary
;* Project:         Math32
;* Version:         2.3
;* Last updated:    2003.09.15
;* Create Date:     1999.10.25
;* Target MCU:      AT90S8515 (as well as others AVR uC)
;*             (C) ATMEL Corporation (mailto:avr@atmel.com)
;* Originator:      (C) 1999-2003 Andre Birua (mailto:birua@hotmail.com)
;*             This Application Note absolutely free in use anybody
;* INTERPRETATION
;* This package of assembler subprograms is developed for integer arithmetic
;* with tracing of sign bit in 32 bits calculations and data reloads.
;* It is based on microcontroller register file to the maximum.
;* In real users projects available abundant digit capacity allows to avoid
;* overflow and reduces inaccuracy of rounding errors in chain calculations.
;* Included macro definitions will increase readability of assembler source
;* at bit by bit and multibyte data operations inside AVR software model
;*
;* DESCRIPTION
;* This Application Note lists:
;*   i) Math32 subroutines for the following:
;*   Add/Subtract/Multiply/Divide/Complement 32 bits operands,
;*   Binary 16 & 24 bits operand to/back BCD conversion,
;*   Binary 32 bits operand to BCD conversion,
;*   Initialization of data memory on a pattern,
;*   Load/Store group of registers from/to data space;
;*  ii) macro definitions call mathematical and data transfer subroutines;
;* iii) useful general macroinstructions for the AVR 8-Bit RISC family
;*
;* "ADD32"      Add without Carry       Rd32 = Rd32 + Rr32
;* "SUB32"      Subtract without Carry  Rd32 = Rd32 - Rr32
;* "MUL32"      Multiply Unsigned       Rd64 = Rd32 * Rr32
;* "DIV32"      Divide Unsigned         Rd32 = Rd32 / Rr32 (Rd64)
;* "COM32"      One's Complement        Rd32 = 0xffffffff - Rd32
;* "NEG32"      Two's Complement        Rd32 = 0x00000000 - Rd32
;* "BCD2bin"    BCD to Binary 16        Rd16 = Rd24|Rr24
;* "BCD3bin"    BCD to Binary 24        Rd24 = Rd32|Rr32
;* "Bin2BCD"    Binary 16 to BCD        Rd24 = Rd16|Rr16
;* "Bin3BCD"    Binary 24 to BCD        Rd32 = Rd24|Rr24
;* "Bin4BCD"    Binary 32 to BCD        Rd40 = Rd32|Rr32 || hwrd(Rr32)&Rd16
;* "MathMem"    Init Data Memory        (MA) = 0x00|0xff
;* "MathLoad"   Load Registers          Rd32|Rr32 = (MA)
;* "MathSave"   Store Registers         (MA) = Rd32|Rd64
;*
;* Rd64: destination registers (8) in the register file
;* Rd32: destination (and source) registers (4) in the register file
;* Rr32: source registers (4) in the register file
;* (MA): address for access to variable in the internal memory (SRAM)
;* Note: Math32 use high registers, r0 and lower 512 bytes of data space,
;*    so Rd64=r20:r27, Rd32=r20:r23, Rd24=r20:r22, Rd16=r20:r21,
;*    Rd40=r20:r24, Rr32=r16:r19, Rr24=r16:r18, Rr16=r16:r17, MA=0:511
;*
;* Number of words & cycles (Min|Max)        c o m m e n t s
;* "ADD32"      6    4|5    Size of Add32sign
;* "SUB32"     16    6|15   Size of Sub32sign
;* "MUL32"     24  460|556  Size of Mul32b, based on AVR200 16x16 unsigned
;* "DIV32"     28  528|688  Size of Div32b, based on AVR200 16/16 unsigned
;* "COM32"      5    4|4    Part of Sub32
;* "NEG32"      9    8|8    Part of Sub32
;* "BCD2bin"   26   86|89   Equivalent of AVR204, but smaller & quicker
;* "BCD3bin"   43   38|402  Different from BCD2bin translation algorithm
;* "Bin2BCD"   22   19|177  Equivalent of AVR204, but smaller & much faster
;* "Bin3BCD"   21   36|366  In the form of preamble for Bin2BCD
;* "Bin3BCD"   40   36|333  All-sufficient expansion of Bin2BCD
;* "Bin4BCD"   37  515|671  Based on AVR204 16-bit Bin to BCD conversion
;* "Bin4BCD"   48  874|878  All-sufficient transform instead of pre-Bin4BCD
;* "MathMem"   10    7|645  Size of MathMemLimit, max cycle for 128 bytes
;* "MathLoad"  15   41|46   Size and max cycle for Rr32 load
;* "MathSave"  14   13|78   Size and max cycle for Rd64 save
;* In total:  350 words     Usually +7 cycles: rcall & ret
;*
;* All routines are Code Size` optimized implementations and debugged with
;* macrocode for AVR macro assembler version 1.30 (Jan 27 1999 01:30:00) &
;*          AVR32 macro assembler version 1.30 (Sep    8 1999 01:30:00).
;*    However, AVR32 macro assembler version 1.54 (Nov 14 2001 14:05:48) &
;*          AVR32 macro assembler version 1.56 (May    6 2002 14:54:01)
;* generate dummy warnings: Register already defined by the .DEF directive
;* (command option for disable this kind of warning as yet is absent...)
;*                CheckIt with AVR Studio !
;* NOTE
;* ` Bin4BCD transformations has partial loop optimization for speed-up
;* While using Math32, it is important to consider the allocation of the
;* microcontroller resources available for the program. It is required:
;* - to use r0,r16..r31 with Math32;
;* - to allocate variables used in calculation in the bottom of the memory;
;* - to use T flag as a sign bit (input, output and temporary),
;*   if you need to operate negative numbers or up-down overflow error
;*
;* VERSION
;* 1.0 Original version (in use starting with 1999.12.22)
;* 1.1 Fixed precedence bugs if macroparameter is an assembler expression
;* 1.2 Modify CBF & SBF & IBF macrocalls
;* 1.3 Full modification mathematical and data transfer macronotation
;* 1.4 Optimaze for speed and code size Mul32 & Div32 & BCD2bin & Bin2BCD
;* 2.0 Version for publication (added description, note and demo sections)
;* 2.1 Updated Bin2BCD, added Bin4BCD conversion & XCH macrocall
;* 2.2 Added functionally closed modifications of Bin3&4BCD translation
;* 2.3 Added BCD3bin conversion, normalize the comment of Bin3&4BCD
;*
;* DEMO
;* section below is a sample of macrocalls and not an ordinary Math32 usage
;*
;***************************************************************************
;*
;* Bin2BCD == 16-bit Binary to BCD conversion
;*
;* fbinL:fbinH >>>  tBCD0:tBCD1:tBCD2
;*     hex               dec
;*   r16r17    >>>   r20r21r22
;*
;***************************************************************************
.def fbinL    =r16 ; binary value Low byte
.def fbinH    =r17 ; binary value High byte
.def tBCD0    =r20 ; BCD value digits 0 and 1
.def tBCD1    =r21 ; BCD value digits 2 and 3
.def tBCD2    =r22 ; BCD value digit 4 (MSD is lowermost nibble)

Bin2BCD20:  mov  r16,r20   ;for compatibility with Math32
          mov  r17,r21   ;
Bin2BCD16:  ldi  tBCD2,0xff  ;initialize digit 4
binbcd_4:   inc  tBCD2        ;
          subi fbinL,low(10000);subiw fbin,10000
          sbci fbinH,high(10000)
          brcc binbcd_4     ;
          ldi  tBCD1,0x9f  ;initialize digits 3 and 2
binbcd_3:   subi tBCD1,0x10  ;
          subi fbinL,low(-1000);addiw fbin,1000
          sbci fbinH,high(-1000)
          brcs binbcd_3     ;
binbcd_2:   inc  tBCD1        ;
          subi fbinL,low(100)   ;subiw fbin,100
          sbci fbinH,high(100) ;
          brcc binbcd_2     ;
          ldi  tBCD0,0xa0  ;initialize digits 1 and 0
binbcd_1:   subi tBCD0,0x10  ;
          subi fbinL,-10    ;addi fbin,10
          brcs binbcd_1     ;
          add  tBCD0,fbinL ;LSD
binbcd_ret: ret             ;
.equ Bin2BCD=Bin2BCD20 ;default registers BIN to BCD call

;***************************************************************************
;*
;* Bin4BCD == 32-bit Binary to BCD conversion     [ together with Bin2BCD ]
;*
;* fbin0:fbin1:fbin2:fbin3  >>>  tBCD0:tBCD1:tBCD2:tBCD3:tBCD4
;*        hex                     dec
;*    r18r19r20r21      >>>   r20r21r22r23r24
;*
;***************************************************************************
.def fbin0    =r18 ; binary value byte 0 (LSB)
.def fbin1    =r19 ; binary value byte 1
```

```
.def fbin2    =r20 ; binary value byte 2
.def fbin3    =r21 ; binary value byte 3 (MSB)
.def tBCD0    =r20 ; BCD value digits 0 and 1 (same as fbin2)
.def tBCD1    =r21 ; BCD value digits 2 and 3 (same as fbin3)
.def tBCD2    =r22 ; BCD value digits 4 and 5
.def tBCD3    =r23 ; BCD value digits 6 and 7
.def tBCD4    =r24 ; BCD value digits 8 and 9 (MSD)

Bin4BCD:       rcall     Bin2BCD20    ;
          clr  tBCD3          ;initial highest bytes of result
          ldi  tBCD4,0xfe     ;
binbcd_loop:        subi tBCD0,-0x33 ;add 0x33 to digits 1 and 0
          sbrs tBCD0,3        ;if bit 3 clear
          subi tBCD0,0x03     ;    sub 3
          sbrs tBCD0,7        ;if bit 7 clear
          subi tBCD0,0x30     ;    sub $30
          subi tBCD1,-0x33    ;add 0x33 to digits 3 and 2
          sbrs tBCD1,3        ;if bit 3 clear
          subi tBCD1,0x03     ;    sub 3
          sbrs tBCD1,7        ;if bit 7 clear
          subi tBCD1,0x30     ;    sub $30
          subi tBCD2,-0x33    ;add 0x33 to digits 5 and 4
          sbrs tBCD2,3        ;if bit 3 clear
          subi tBCD2,0x03     ;    sub 3
          sbrs tBCD2,7        ;if bit 7 clear
          subi tBCD2,0x30     ;    sub $30
          lsl  fbin0          ;
          rol  fbin1          ;shift lower word
          rol  tBCD0          ;through all bytes
          rol  tBCD1          ;
          rol  tBCD2          ;
          rol  tBCD3          ;
          rol  tBCD4          ;
          brmi binbcd_loop    ;7 shifts w/o correction of MSD
          rol  fbinH          ;since Bin2BCD fbinH = 0xff
          brcc binbcd_ret     ;  so as to do 16_lsl in total
          subi tBCD3,-0x33    ;add 0x33 to digits 7 and 6
          sbrs tBCD3,3        ;if bit 3 clear
          subi tBCD3,0x03     ;    sub 3
          sbrs tBCD3,7        ;if bit 7 clear
          subi tBCD3,0x30     ;    sub $30
          subi tBCD4,-0x03    ;add 0x03 to digit 8 only
          sbrs tBCD4,3        ;if bit 3 clear
          subi tBCD4,0x03     ;    sub 3
          rjmp binbcd_loop    ;

;***************************************************************************
;*
;* Bin4BCD == 32-bit Binary to BCD conversion
;*
;* fbin0:fbin1:fbin2:fbin3  >>>  tBCD0:tBCD1:tBCD2:tBCD3:tBCD4
;*        hex                    dec
;*    r16r17r18r19      >>>    r20r21r22r23r24
;*
;***************************************************************************
.def fbin0    =r16 ; binary value byte 0 (LSB)
.def fbin1    =r17 ; binary value byte 1
.def fbin2    =r18 ; binary value byte 2
.def fbin3    =r19 ; binary value byte 3 (MSB)
```

```
.def tBCD0    =r20 ; BCD value digits 0 and 1
.def tBCD1    =r21 ; BCD value digits 2 and 3
.def tBCD2    =r22 ; BCD value digits 4 and 5
.def tBCD3    =r23 ; BCD value digits 6 and 7
.def tBCD4    =r24 ; BCD value digits 8 and 9 (MSD)

Bin4BCD20:  mov  r16,r20   ;for compatibility with Math32
          mov  r17,r21   ;
          mov  r18,r22   ;
          mov  r19,r23   ;
Bin4BCD16:  clr  tBCD0        ;initial result (5 bytes)
          clr  tBCD1        ;    & shift
          clr  tBCD2        ;           loop
          ldi  tBCD3,0xfe   ;            counter
          ldi  tBCD4,0xff   ;                too
          rjmp binbcd_jump ;for speed-up and skip of MSD corr
binbcd_876: subi tBCD4,-0x03 ;add 0x03 to digit 8 only
          sbrs tBCD4,3      ;if bit 3 clear
          subi tBCD4,0x03   ;    sub 3
          subi tBCD3,-0x33 ;add 0x33 to digits 7 and 6
          sbrs tBCD3,3      ;if bit 3 clear
          subi tBCD3,0x03   ;    sub 3
          sbrs tBCD3,7      ;if bit 7 clear
          subi tBCD3,0x30   ;    sub $30
binbcd_54:  subi tBCD2,-0x33 ;add 0x33 to digits 5 and 4
          sbrs tBCD2,3      ;if bit 3 clear
          subi tBCD2,0x03   ;    sub 3
          sbrs tBCD2,7      ;if bit 7 clear
          subi tBCD2,0x30   ;    sub $30
binbcd_3210:    subi tBCD1,-0x33 ;add 0x33 to digits 3 and 2
          sbrs tBCD1,3      ;if bit 3 clear
          subi tBCD1,0x03   ;    sub 3
          sbrs tBCD1,7      ;if bit 7 clear
          subi tBCD1,0x30   ;    sub $30
          subi tBCD0,-0x33 ;add 0x33 to digits 1 and 0
          sbrs tBCD0,3      ;if bit 3 clear
          subi tBCD0,0x03   ;    sub 3
          sbrs tBCD0,7      ;if bit 7 clear
          subi tBCD0,0x30   ;    sub $30
binbcd_jump:    lsl  fbin0        ;
          rol  fbin1        ;
          rol  fbin2        ;
          rol  fbin3        ;shift input value
          rol  tBCD0        ;through all bytes
          rol  tBCD1        ;
          rol  tBCD2        ;
          rol  tBCD3        ;
          rol  tBCD4        ;
          brcs binbcd_3210 ;16_lsl w/o correction of dig_87654
          inc  fbin0        ;
          brpl binbcd_54    ;+7_lsl w/o correction of dig_876
          sbrs fbin2,0      ;
          rjmp binbcd_876   ;32_lsl in total (fbin = 0x1ffff)
          ret          ;

;***************************************************************************
;*
;* Bin3BCD == 24-bit Binary to BCD conversion     [ together with Bin2BCD ]
;*
```

```
;* fbin0:fbin1:fbin2  >>>  tBCD0:tBCD1:tBCD2:tBCD3
;*     hex                     dec
;*     r16r17r18      >>>     r20r21r22r23
;*
;***************************************************************************
.def fbin0    =r16 ; binary value byte 0 (LSB)
.def fbin1    =r17 ; binary value byte 1
.def fbin2    =r18 ; binary value byte 2 (MSB)
.def tBCD0    =r20 ; BCD value digits 0 and 1
.def tBCD1    =r21 ; BCD value digits 2 and 3
.def tBCD2    =r22 ; BCD value digits 4 and 5
.def tBCD3    =r23 ; BCD value digits 6 and 7 (MSD)

Bin3BCD:     ldi  tBCD3,0xff       ;initialize digits 7 and 6
binbcd_7:    inc  tBCD3            ;
          subi fbin0,byte1(10000*100)  ;subit fbin,1000000
          sbci fbin1,byte2(10000*100)  ;
          sbci fbin2,byte3(10000*100)  ;
          brcc binbcd_7          ;
          subi tBCD3,-6          ; delete decimal correction
          sbrs tBCD3,4           ; if NUMBER<10000000 always
          subi tBCD3,6           ;
          ldi  tBCD2,0x9f        ;initialize digits 5 and 4
binbcd_6:    subi tBCD2,0x10       ;
          subi fbin0,byte1(-10000*10)  ;addit fbin,100000
          sbci fbin1,byte2(-10000*10)  ;
          sbci fbin2,byte3(-10000*10)  ;
          brcs binbcd_6          ;
binbcd_5:    inc  tBCD2            ;
          subi fbin0,byte1(10000)  ;subit fbin,10000
          sbci fbin1,byte2(10000)  ;
          sbci fbin2,byte3(10000)  ;
          brcc binbcd_5          ;
          rjmp binbcd_3-1        ;initialize digits 3 and 2

;***************************************************************************
;*
;* Bin3BCD == 24-bit Binary to BCD conversion
;*
;* fbin0:fbin1:fbin2  >>>  tBCD0:tBCD1:tBCD2:tBCD3
;*     hex                     dec
;*     r16r17r18      >>>     r20r21r22r23
;*
;***************************************************************************
.def fbin0    =r16 ; binary value byte 0 (LSB)
.def fbin1    =r17 ; binary value byte 1
.def fbin2    =r18 ; binary value byte 2 (MSB)
.def tBCD0    =r20 ; BCD value digits 0 and 1
.def tBCD1    =r21 ; BCD value digits 2 and 3
.def tBCD2    =r22 ; BCD value digits 4 and 5
.def tBCD3    =r23 ; BCD value digits 6 and 7 (MSD)

Bin3BCD20:   mov  r16,r20   ;for compatibility with Math32
          mov  r17,r21   ;
          mov  r18,r22   ;
Bin3BCD16:   ldi  tBCD3,0xfa       ;initialize digits 7 and 6
binbcd_107:  subi tBCD3,-0x10      ;
          subi fbin0,byte1(10000*1000) ;subit fbin,10^7
          sbci fbin1,byte2(10000*1000) ;
          sbci fbin2,byte3(10000*1000) ;
          brcc binbcd_107       ;
binbcd_106: dec  tBCD3            ;
          subi fbin0,byte1(-10000*100) ;addit fbin,10^6
          sbci fbin1,byte2(-10000*100) ;
          sbci fbin2,byte3(-10000*100) ;
          brcs binbcd_106       ;
          ldi  tBCD2,0xfa       ;initialize digits 5 and 4
binbcd_105: subi tBCD2,-0x10      ;
          subi fbin0,byte1(10000*10)   ;subit fbin,10^5
          sbci fbin1,byte2(10000*10)   ;
          sbci fbin2,byte3(10000*10)   ;
          brcc binbcd_105       ;
binbcd_104: dec  tBCD2            ;
          subi fbin0,byte1(-10000) ;addit fbin,10^4
          sbci fbin1,byte2(-10000) ;
          sbci fbin2,byte3(-10000) ;
          brcs binbcd_104       ;
          ldi  tBCD1,0xfa       ;initialize digits 3 and 2
binbcd_103: subi tBCD1,-0x10      ;
          subi fbin0,byte1(1000)   ;subiw fbin,10^3
          sbci fbin1,byte2(1000)   ;
          brcc binbcd_103       ;
binbcd_102: dec  tBCD1            ;
          subi fbin0,byte1(-100)   ;addiw fbin,10^2
          sbci fbin1,byte2(-100)   ;
          brcs binbcd_102       ;
          ldi  tBCD0,0xfa       ;initialize digits 1 and 0
binbcd_101: subi tBCD0,-0x10      ;
          subi fbin0,10         ;subi fbin,10^1
          brcc binbcd_101       ;
          add  tBCD0,fbin0      ;LSD
          ret              ;



;Neg32:        subi sub10,1   ;if result<0
;         sbci sub11,0   ;   neg result
;         sbci sub12,0   ;
;         sbci sub13,0   ;   (dec result)
;Com32:        com  sub10          ;    &
;         com  sub11          ;   (com result)
;         com  sub12          ;
;         com  sub13          ;   return set carry after com
;Return32u:    ret             ;
NEG16:
          SUBI R16,1
          SUBI R17,0
          COM      R16
          COM      R17
          RET
NEG24:
          SUBI R16,1
          SUBI R17,0
          SUBI R18,0
          COM      R16
          COM      R17
          COM      R18
          RET
NEG32:
```

```
          SUBI R16,1
          SUBI R17,0
          SUBI R18,0
          SUBI R19,0
          COM       R16
          COM       R17
          COM       R18
          COM       R19
          RET
```

```
;---------------------------------------------------------------------------------------------------
;    FILE NAME:     MACRO.INC
;
;    MACRO TYPE ROUTINES
;    CREATED BY OMID KOMPANI- FARASINA CO -  IRAN-TEHRAN
;    VERSION:2.0
;    LAST UPDATE: 1386-01-06       2007-MARCH-26
;    VERSION:1.0
;    LAST UPDATE: 1385-06-17       2006-SEP-08
;---------------------------------------------------------------------------------------------------

.CSEG

;***************************************************************************
;    INITIALIZING THE STACK POINTER AT THE TOP OF SRAM
;***************************************************************************
.MACRO    _INIT_STACK
     LDI       ZH,HIGH(RAMEND)
     OUT       SPH,ZH
     LDI       ZL,LOW(RAMEND)
     OUT       SPL,ZL
.ENDMACRO



;---------------------------------------------------------------------------------------------------
;
;    LOADING AND STORING FUNCTIONS
;
;---------------------------------------------------------------------------------------------------


;*************************************
;*  _LOAD_8BIT_RAM
;*************************************
.MACRO    _8RAM2REG
     LDS       @1,@0
.ENDMACRO

;*************************************
;*  _STORE_8BIT_RAM
;*************************************
.MACRO    _8REG2RAM
```

```
     STS       @0,@1
.ENDMACRO


;*************************************
;*  _LOAD_16BIT_RAM
;*************************************
.MACRO    _16RAM2REG
     LDI       YL,LOW(@0)
     LDI       YH,HIGH(@0)
     LD        @2,Y+
     LD        @1,Y
.ENDMACRO

;*************************************
;*  _STORE_16BIT_RAM
;*************************************
.MACRO    _16REG2RAM
     LDI       YL,LOW(@0)
     LDI       YH,HIGH(@0)
     ST        Y+,@2
     ST        Y+,@1
.ENDMACRO

;*************************************
;*  LOAD 32BIT FROM RAM TO REGISTER
;*************************************
.MACRO    _32RAM2REG
     LDI       YL,LOW(@0)
     LDI       YH,HIGH(@0)
     LD        @4,Y+
     LD        @3,Y+
     LD        @2,Y+
     LD        @1,Y
.ENDMACRO

;*************************************
;*  STORE 32BIT FROM REGISTER TO RAM
;*************************************
.MACRO    _32REG2RAM
     LDI       YL,LOW(@0)
     LDI       YH,HIGH(@0)
     ST        Y+,@4
     ST        Y+,@3
     ST        Y+,@2
     ST        Y,@1
.ENDMACRO




;-----------------------------------------------------------------------------------------------
;                                        - CONDITIONAL MACROS -
;-----------------------------------------------------------------------------------------------
;-----------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_SET_CALL
     SBRC     @0,@1
     CALL     @2
.ENDMACRO
```

```
;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_SET_RCALL
     SBRC     @0,@1
     RCALL     @2

.ENDMACRO
;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_SET_JUMP
     SBRC     @0,@1
     JMP      @2

.ENDMACRO

;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_SET_RJUMP
     SBRC     @0,@1
     RJMP     @2

.ENDMACRO

;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_NOTSET_CALL
     SBRS     @0,@1
     CALL     @2

.ENDMACRO

;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_NOTSET_RCALL
     SBRS     @0,@1
     RCALL    @2
.ENDMACRO

;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_NOTSET_JUMP
     SBRS     @0,@1
     JMP     @2
.ENDMACRO

;----------------------------------------------------------------------------------------------------
;
;----------------------------------------------------------------------------------------------------
.MACRO   _IF_BIT_NOTSET_RJUMP
     SBRS     @0,@1
     RJMP     @2
.ENDMACRO
```

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

.MACRO    _SKIP_IF_BIT_NOTSET
     SBRC @0,@1
.ENDMACRO
;-------------------------------------------------------------------------------------------------
;
;-------------------------------------------------------------------------------------------------

.MACRO    _SKIP_IF_BIT_SET
     SBRS @0,@1
.ENDMACRO
;-------------------------------------------------------------------------------------------------
;
;-------------------------------------------------------------------------------------------------


.MACRO _CBR ; Register,Bit#
          CLT
          BLD       @0,@1
.ENDMACRO

.MACRO _SBR ; Register,Bit#
          SET
          BLD  @0,@1
.ENDMACRO



;*************************************************************************
;*************************************************************************
;   DELAY MACROS
;*************************************************************************
;*************************************************************************

;***********************************************************************
;MAKES A DELAY WITH uS TIME BASE
;***********************************************************************

.MACRO    _WAIT_uS

     LDI       R26,((@0/1000000)*@1/4)
WAIT_uS_LOOP1:
     DEC       R26
     NOP
     BRNE WAIT_uS_LOOP1

.ENDMACRO

;***********************************************************************
;MAKES A DELAY WITH 5uS TIME BASE
;***********************************************************************

.MACRO    _WAIT_5us

     LDI       R26,((@0/1000000)*@1/8)
```

```
WAIT_5uS_LOOP1:
    LDI     R25,9
WAIT_5uS_LOOP2:
    DEC     R25
    NOP
    BRNE WAIT_5uS_LOOP2
    DEC     R26
    NOP
    BRNE WAIT_5uS_LOOP1

.ENDMACRO

;***********************************************************************
;MAKES A DELAY WITH 10uS TIME BASE
;***********************************************************************

.MACRO   _WAIT_10us

    LDI     R26,((@0/1000000)*@1/8)
WAIT_10uS_LOOP1:
    LDI     R25,19
WAIT_10uS_LOOP2:
    DEC     R25
    NOP
    BRNE WAIT_10uS_LOOP2
    DEC     R26
    NOP
    BRNE WAIT_10uS_LOOP1

.ENDMACRO

;***********************************************************************
;MAKES A DELAY WITH mS TIME BASE
;***********************************************************************

.MACRO   _WAIT_mS

    LDI     R26,(@0/1000000)* @1
WAIT_mS_LOOP1:
    LDI     R25,249
WAIT_mS_LOOP2:
    DEC     R25
    NOP
    BRNE WAIT_mS_LOOP2
    DEC     R26
    NOP
    BRNE WAIT_mS_LOOP1

.ENDMACRO

;***********************************************************************
;DISPLAYS A MESSAGE PARTIAL OR FULL(NULL OR ZERO ENDED)
;***********************************************************************
.MACRO   _LCD_DISPLAY_MSG
    LDI     ZH,HIGH(@0*2)
    LDI     ZL,LOW(@0*2)
.IF @1==1
    LDI    YH,HIGH(RAM_DISPLAY_BUFFER_LINE1)
    LDI    YL,LOW(RAM_DISPLAY_BUFFER_LINE1)
```

```
.ELIF @1==2
    LDI       YH,HIGH(RAM_DISPLAY_BUFFER_LINE2)
    LDI       YL,LOW(RAM_DISPLAY_BUFFER_LINE2)
.ENDIF
    LDI       R16,@2
    LDI        R17,@3
.IF @3==__NULL | @3==__ZERO
    RCALL     LOAD_TERMINATEDstr_FLASH2LCDBUFFER
.ELSE
    RCALL     LOAD_CHARstr_FLASH2LCDBUFFER
.ENDIF
.ENDMACRO
;*********************************************************************
;    DISPLAYS FORMATTED NUMBERS
;    16,24 OR 32 BIT NUMBERS WITH SIGN AND DECIMAL POINT
;
;*********************************************************************
.MACRO    _LCD_DISPLAY_NUM

.IF   @0==__16BIT
    MOV       fBINH,@5
    MOV       fBINL,@6
.IF @1==__SIGNED
    BST       fBINH,7
    SBRC fBINH,7
    RCALL     NEG16
.ENDIF
;    CALL BIN2BCD16 ; UNCOMMENT THIS LINE FOR ABSOLUTE CALLS
    RCALL     BIN2BCD16    ; UNCOMMENT THIS LINE FOR RELATIVE CALLS LOWER THAN +,-2K
.IF @2==1
    LDI       YH,HIGH(RAM_DISPLAY_BUFFER_LINE1)
    LDI       YL,LOW(RAM_DISPLAY_BUFFER_LINE1)
.ELIF @2==2
    LDI       YH,HIGH(RAM_DISPLAY_BUFFER_LINE2)
    LDI       YL,LOW(RAM_DISPLAY_BUFFER_LINE2)
.ENDIF
    LDI       R16,@3
    LDI        R17,@4
    LDI        R19,3
    LDI        ZL,22
    CLR        ZH
    RCALL     BCD2LCDBUFFER



.ELIF @0==__24BIT
    MOV       fBIN2,@5
    MOV       fBIN1,@6
    MOV       fBIN0,@7
.IF @1==__SIGNED
    BST       fBIN2,7
    SBRC fBIN2,7
    RCALL     NEG24
.ENDIF
;    CALL BIN3BCD16 ; UNCOMMENT THIS LINE FOR ABSOLUTE CALLS
    RCALL     BIN3BCD16    ; UNCOMMENT THIS LINE FOR RELATIVE CALLS LOWER THAN +,-2K
.IF @2==1
    LDI       YH,HIGH(RAM_DISPLAY_BUFFER_LINE1)
    LDI      YL,LOW(RAM_DISPLAY_BUFFER_LINE1)
.ELIF @2==2
    LDI      YH,HIGH(RAM_DISPLAY_BUFFER_LINE2)
    LDI      YL,LOW(RAM_DISPLAY_BUFFER_LINE2)
.ENDIF
    LDI      R16,@3
    LDI       R17,@4
    LDI       R19,4
    LDI       ZL,23
    CLR       ZH
    RCALL     BCD2LCDBUFFER

.ELIF @0==__32BIT
    MOV       fBIN3,@5
    MOV       fBIN2,@6
    MOV       fBIN1,@7
    MOV       fBIN0,@8
.IF @1==__SIGNED
    BST       fBIN3,7
    SBRC fBIN3,7
    RCALL     NEG32
.ENDIF
;    CALL BIN4BCD16 ; UNCOMMENT THIS LINE FOR ABSOLUTE CALLS
    RCALL     BIN4BCD16    ; UNCOMMENT THIS LINE FOR RELATIVE CALLS LOWER THAN +,-2K
.IF @2==1
    LDI      YH,HIGH(RAM_DISPLAY_BUFFER_LINE1)
    LDI      YL,LOW(RAM_DISPLAY_BUFFER_LINE1)
.ELIF @2==2
    LDI      YH,HIGH(RAM_DISPLAY_BUFFER_LINE2)
    LDI      YL,LOW(RAM_DISPLAY_BUFFER_LINE2)
.ENDIF
    LDI      R16,@3
    LDI       R17,@4
    LDI       R19,5
    LDI       ZL,24
    CLR       ZH
    RCALL     BCD2LCDBUFFER
.ENDIF

.ENDMACRO
```

```
.EQU CPU_FREQUENCY=16000000 ; USED BY DELAY ROUTINES
.EQU __LCD_TICK=0

.DEF TEMP=R16
.DEF SYSTEM_FLAGS=R7
```

```
.EQU DATARAM_START=$60
.EQU DATARAM_END=$FF

.DSEG

;*************************************************************************
;*************************************************************************
;   SOFT TIMERS SRAM ADDRESSES
;*************************************************************************
;*************************************************************************
RAM_STIMER1_CV: .BYTE   1
RAM_STIMER1_PV: .BYTE   1

RAM_DISPLAY_BUFFER_LINE1:   .BYTE   20
RAM_DISPLAY_BUFFER_LINE2:   .BYTE   20

RAM_LCD_POINTER:            .BYTE   2
RAM_CHRONOMETER:            .BYTE   4






.DEF STIMER_FLAGS=R10
;STIMER_FLAGS
.EQU    _STIMER1_EN=0
.EQU    _STIMER1_CM=3





.EQU __LCD_INIT_CODE=0B00110000
.EQU __LCD_8BIT_INTERFACE=0B00110000
.EQU __LCD_2LINE=0B00101000
.EQU __LCD_5x8_MATRIX=0B00100000
.EQU __LCD_CLEAR=0B00000001
.EQU __LCD_OFF=0B00001000
.EQU __LCD_ON=0B00001100
.EQU __LCD_SHOW_CURSOR=0B00000010
.EQU __LCD_SHOW_BLINK=0B00000001

.EQU __LCD_CGA=$40
.EQU    __LCD_LINE1_ADR=$80
.EQU    __LCD_LINE2_ADR=$C0
.EQU    __LCD_LINE1=1
.EQU    __LCD_LINE2=2
.EQU __NULL=0XFF
.EQU __ZERO=0X00
.EQU __16BIT=0
.EQU __24BIT=1
.EQU __32BIT=2
.EQU __UNSIGNED=0
.EQU __SIGNED=1
```

```
.EQU __LCD_DC=5
.EQU __LCD_RW=6
.EQU __LCD_STROBE=7

.DEF LCD_REGISTER=R18
```

```
;*************************************************************************************************
;*                                                                                               *
;*                                      - RELEASE INFO -                                         *
;*                                                                                               *
;*************************************************************************************************
;=================================================================================================
;                                            - GENERAL INFO -
;=================================================================================================
;
;   PROJECT      :  ADVENCED LCD DRIVER
;    SUB PROJECT     :   32 BIT CHRONOMETER WITH 100mS RESOLUTION
;   AUTHOR       :  OMID KOMPANI
;   BOARD DESIGN :  OMID KOMPANI
;   LAST UPDATE  :  1386-01-05          2007-03-25
;   VERSION      :  V1.0
;-------------------------------------------------------------------------------------------------
;   MCU          :  AVR ATMEGA16
;   CLOCK SOURCE :  EXTERNAL CRYSTAL 16.000 MHz OSCILATOR
;   LANGUAGE     :  ASSEMBLY
;   ASSEMBLER    :  AVR ASM 1
;   IDE          :  AVR STUDIO 4.10
;-------------------------------------------------------------------------------------------------
;=================================================================================================
;                                        - ABOUT THIS PROJECT -
;=================================================================================================
;
;=================================================================================================
;                           - HARDWARE & SOFTWARE -
;=================================================================================================
;
;   BOARD POWER  :  BIULTiN POWER SUPPLY
;                        AC 220V INPUT AND DC 5V OUTPUT
;-------------------------------------------------------------------------------------------------
;   PORTS        :  PORTA [0..7]: LCD DATA BUS
;                        PORTB
;                                5: LCD DATA/COMMAND SELECT(REGISTER SELECT)
;                                6: LCD READ/WRITE SELECT
;                                7: LCD STORBE

;                        PORTF [4..7]: JTAG CONNECTOR
;                                4: TCK
;                                5: TMS
;                                6: TDO
;                                7: TDI
;-------------------------------------------------------------------------------------------------
```