APPLICATION NOTE

ST7 MATH UTILITY ROUTINES

by Microcontroller Division Application Team

INTRODUCTION

The goal of this application note is to present the following mathematical routines:

- division of two 8-bit numbers
- multiplication of two 16-bit numbers
- division of one 32 bit number by a 16-bit one (result stored into a word)
- addition of two 16-bit numbers
- subtraction of two 16-bit numbers
- test if a 16 bit number value is within a predefined range
- binary to decimal conversion

In this application, the MCU used is a ST72251.

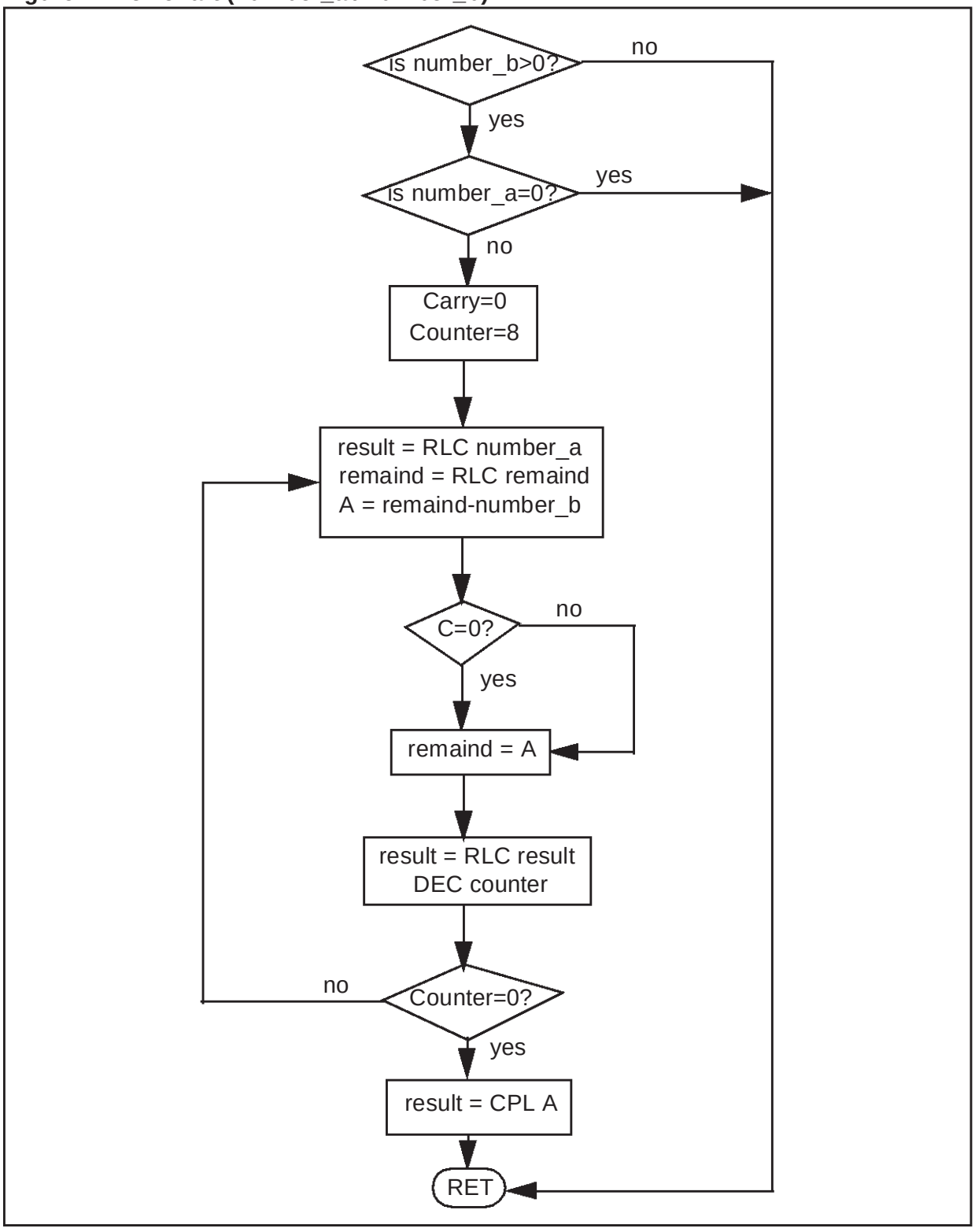
1 DIVISION OF TWO BYTES

The flowchart in Figure 1 describes the routine dividing two 8 bit numbers. These two numbers have to be positive and greater than 0. The division is the following: number_a / number_b.

The COUNTER variable contains the shift register, the RESULT variable contains the result and REMAIND contains the remainder.

Note: RLC = Rotate Left Logical through Carry and CPL = Logical 1-Complement.

Figure 1. Flowchart (number_a / number_b)



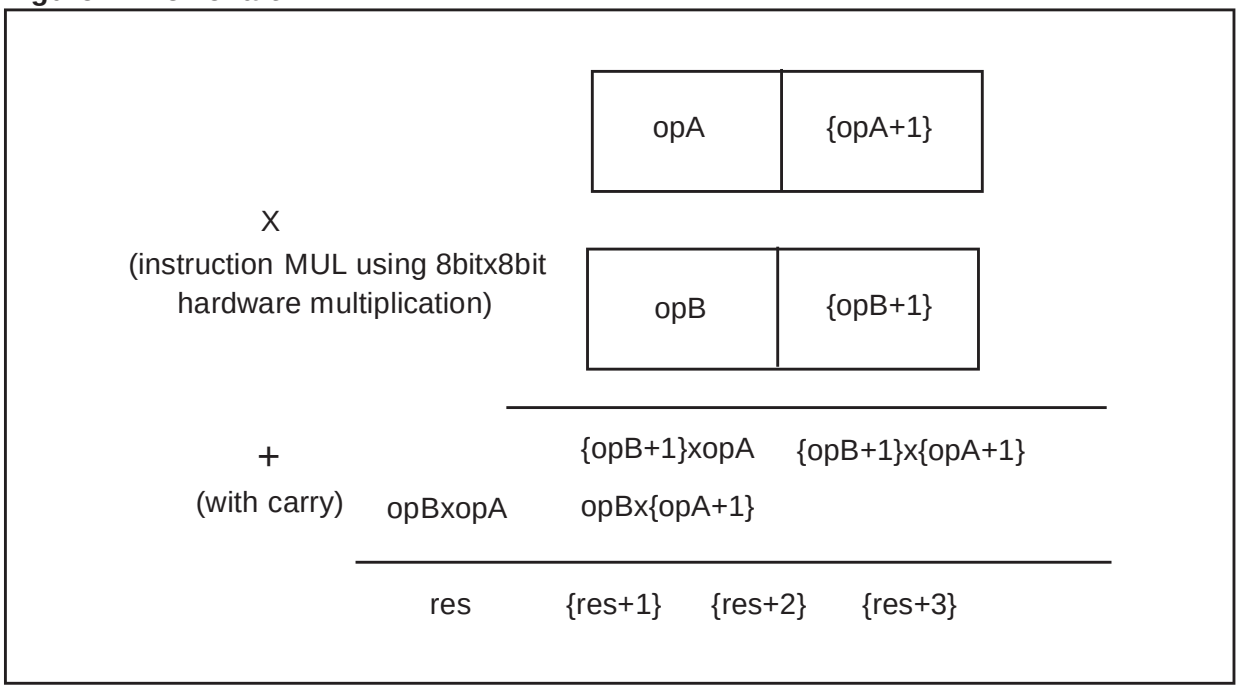
2 MULTIPLICATION OF WORDS

This routine describes the multiplication of two 16-bit numbers. Operations used are multiplication and addition with carry.

The RES variable contains the result of the multiplication (four 8 bit parts).

Note: In the flowchart Figure 2: opA means MSB of the OPERAND_A, opB means MSB of the OPERAND_b, {opA+1} LSB of the OPERAND_A and {opB+1} LSB of the OPERAND_b.

Figure 2. Flowchart



3 LONG BY WORD DIVISION

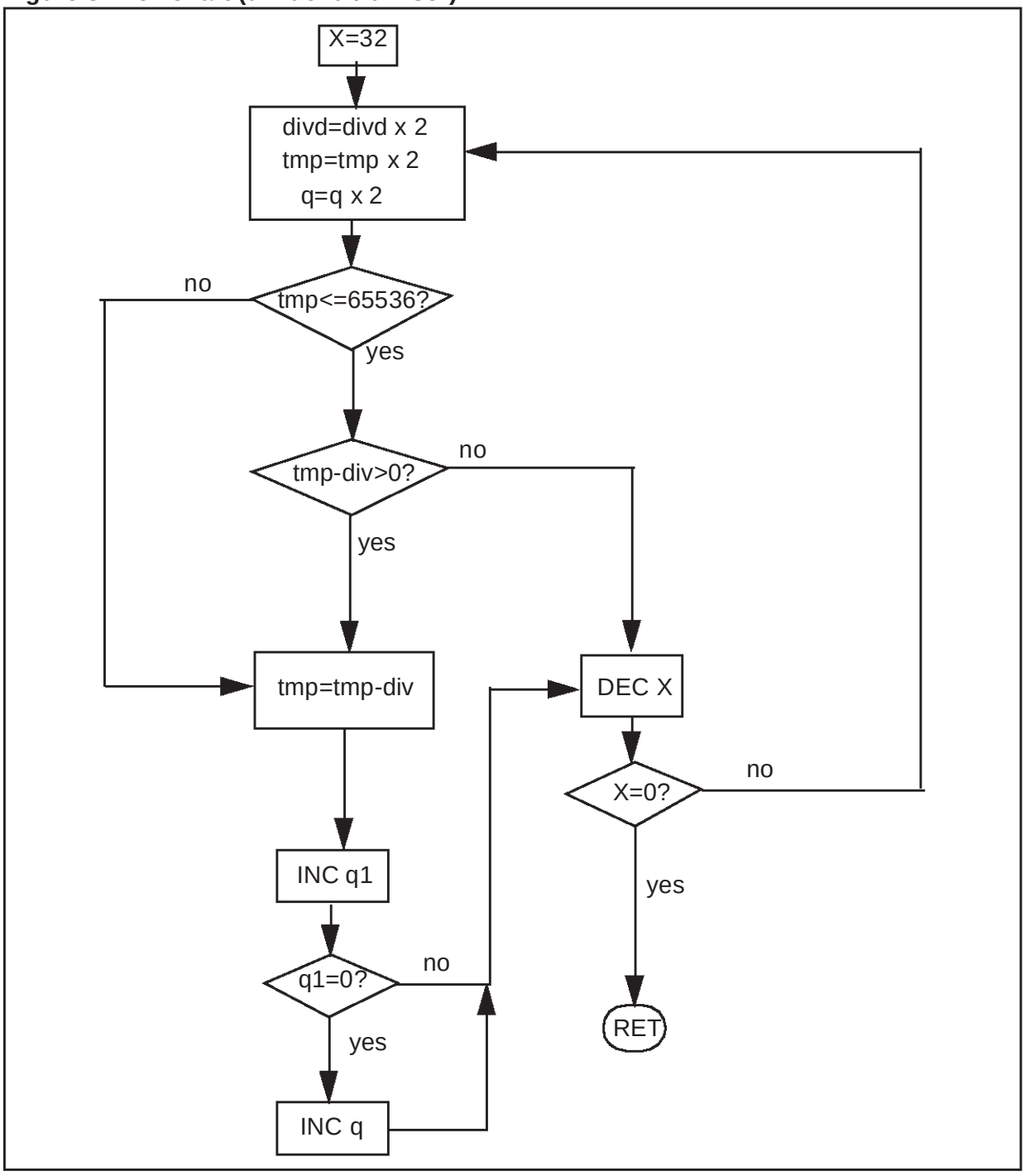
This routine describes a division of a 32-bit number by a 16-bit one.

The result is saved in two 8-bit variables. The result is stored In 16 bits so it's only valid for small numbers.

DIVIDEND variable contains the dividend (32 bit), DIVISOR the divisor (16 bits), TEMPQUOT the quotient tempory value (32 bits) and QUOTIENT the result (tempquot brought to 16 bits).

In the following flowchart divd means dividend, div means divisor, tmp means tempquot and qx means {quotient+x}.

Figure 3. Flowchart (dividend / divisor)



4 ADDITION OF TWO WORDS

This very simple routine adds two 16-bit numbers. The result is saved in two 8 bits registers. The carry flag indicates any overflow.

The LSB parts of the two words are treated first (they are added). Then, MSB parts are added with an ADC instruction to take into account any overflow.

Please refer to the corresponding software for more details.

5 SUBSTRACTION OF TWO WORDS

The method is exactly the same that the addition one.

LSB parts are subtracted first and then the MSB parts are substracted with an SBC instruction to take into account any overflow.

Please refer to the corresponding software for more details.

6 CHECK MIN/MAX

This routine tests if a 16-bit numeric is within a predefined range.

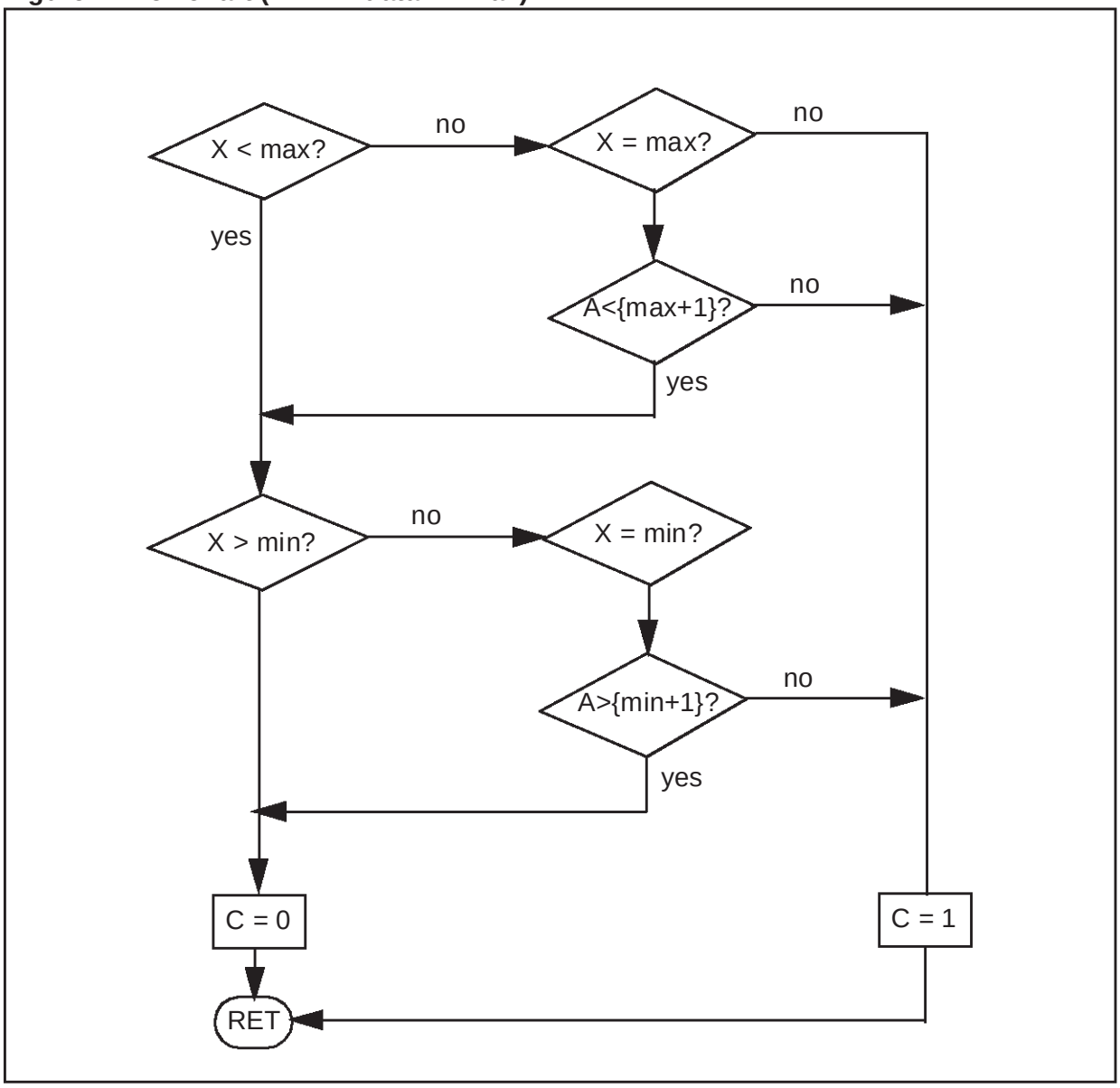
The data variable contain the number to test, min the minimum value and max the maximum one.

The C flag is updated according to the result. If C = 0 at the end of the routine, the test is OK (the value is between min and max) and if C=1, the test has failed.

In the following flowchart, max and min are MSB parts and {max+1} and {min+1} are LSB parts of max and min variables.

Please refer to the corresponding software for more details.

Figure 4. Flowchart (min <= data <= max)



7 CHECK RANGE FOR A WORD

This routine is similar to the previous one. It tests if a 16-bit number value is within a predefined range: MEDIAN-DELTA<=DATA<=MEDIAN+DELTA.

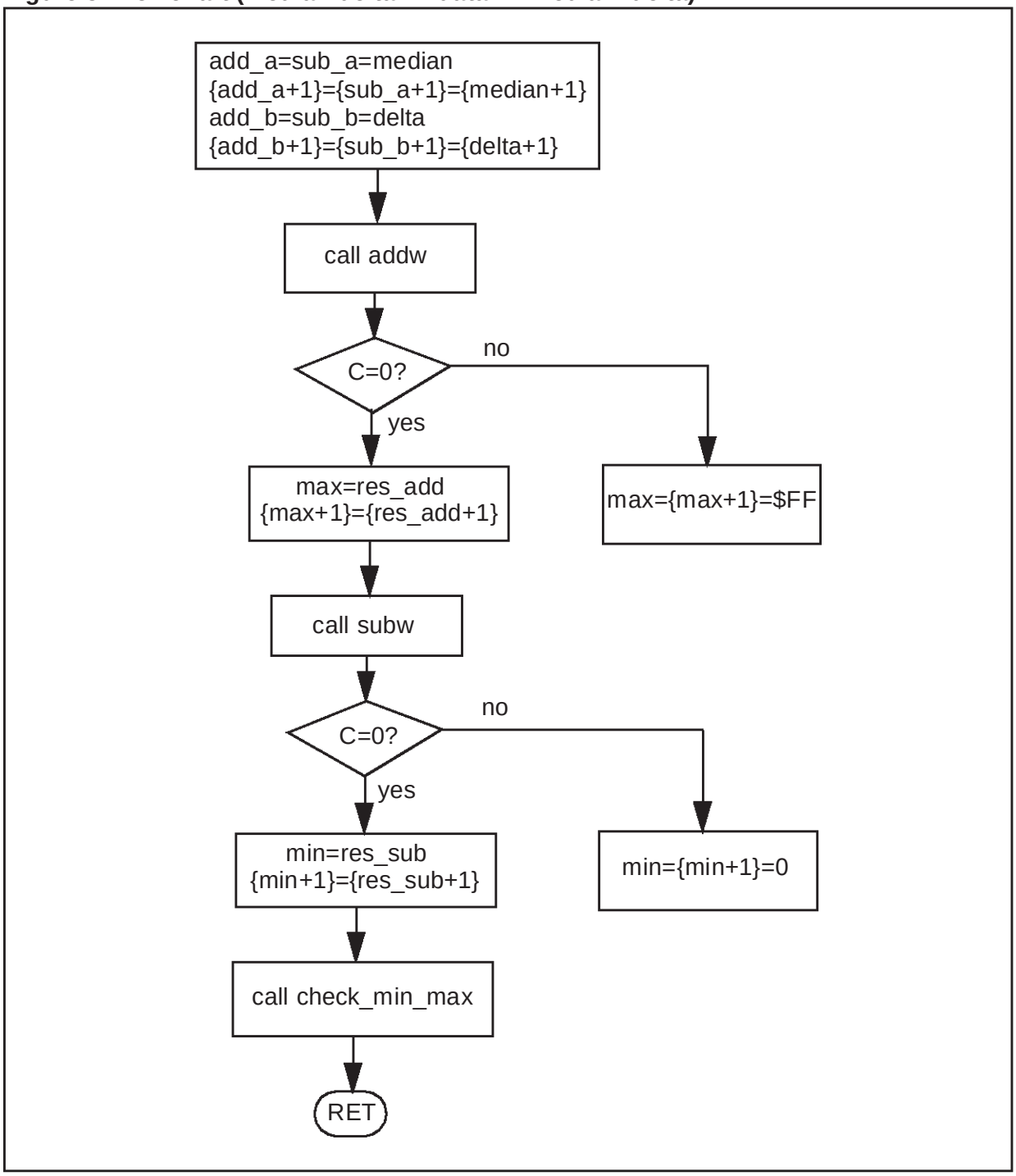
The data register contains the number to test, Median the median value and Delta the desired approximation.

The C flag is also updated according to the result of the test (C=1 means the test has failed).

This routine uses three previous routines: check-min-max, addw and subw.

Please refer to the corresponding software for more details.

Figure 5. Flowchart (median-delta <= data <= median+delta)



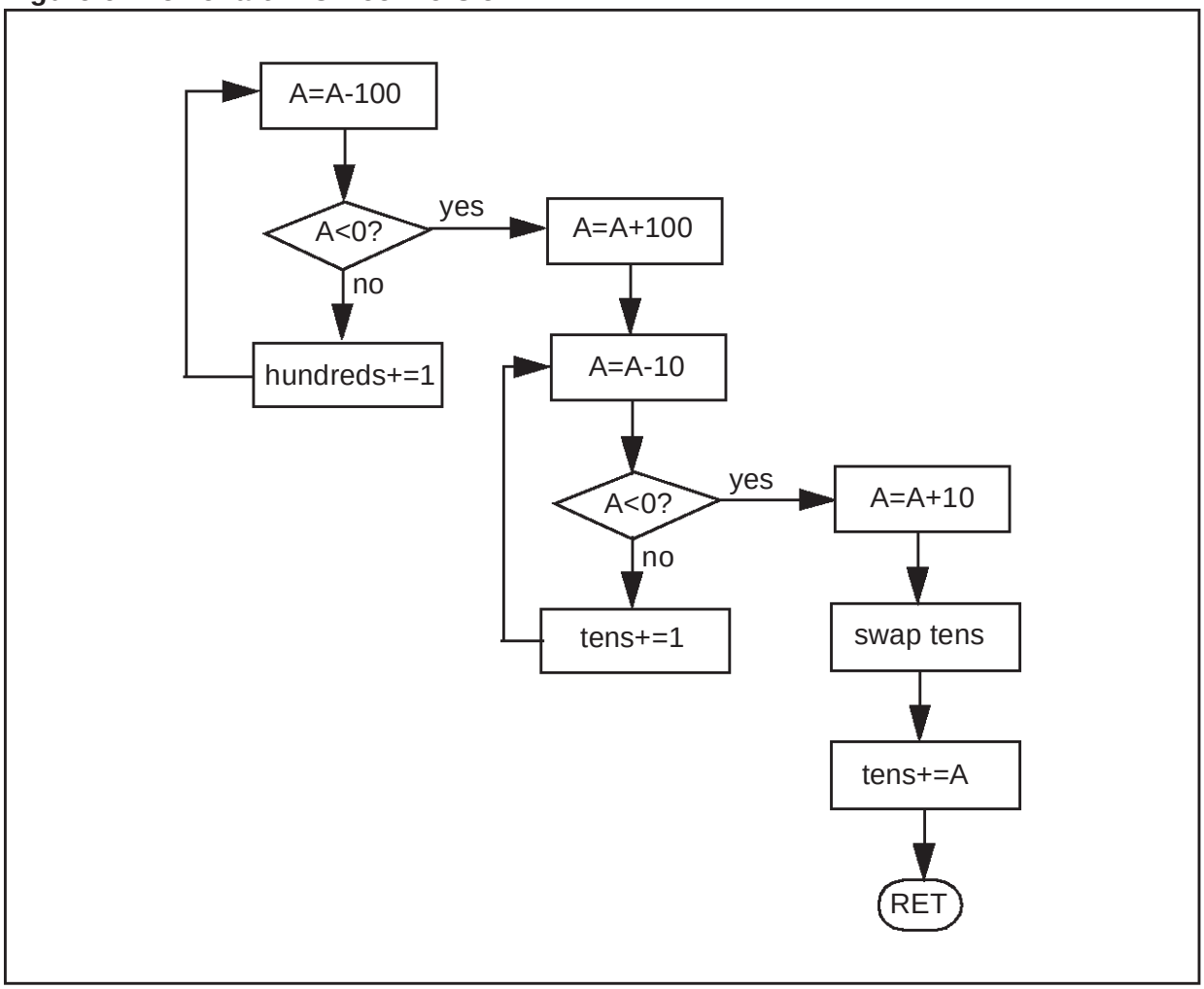
8 BINARY TO DECIMAL CONVERSION (BCD)

This routine performs a BCD conversion (Binary to decimal).

The value to convert is stored in the accumulator. The variable called Hundreds holds the number of hundreds of the converted value, the Tens one the number of tens and units.

Please refer to the corresponding software for more details.

Figure 6. Flowchart: BCD conversion



9 CPU CYCLES FOR EACH ROUTINE

For the ST7 family, fcpu max is 8MHz. Hence one CPU cycle is 0.125 µs.

Here follows a table with previous routines and corresponding numbers of CPU cycles.

Routine name	Description	Number of CPU cycles	Corresponding time
div_bxb	division of 2 bytes	372	46.5 µs
multiw	multiplication of 2 words	154	19.25
div_lxw	division of a word by a byte	1104	138 µs
addw	addition of 2 words	40	5 µs
subw	substraction of 2 words	40	5 µs
check_min_max	test if a value is within a predefined range	55	68.75 µs
check_range	test if a value is within a predefined range	239	29.875 µs
BtoD	Binary to decimal conversion	81	10.125 µs

10 SOFTWARE

The assembly code given below is guidance only. The complete software with all the files can be found in the software library.

```
st7/                    ; the first line is reserved
                      ; for specifying the instruction set
                       ; of the target processor

;*********************************************************************
; TITLE:            maths.ASM
; AUTHOR:           Microcontroller Division Applications Team
; DESCRIPTION:      Mathematics library for ST7 micros
;*********************************************************************

    TITLE  “maths.ASM”
                       ; this title will appear on each
                       ; page of the listing file
    MOTOROLA          ; this directive forces the Motorola
                       ; format for the assembly (default)
    #INCLUDE “st72251.inc”     ; include st72251 registers and memory mapping
file
    #INCLUDE “constant.inc”     ; include general constants file

;*********************************************************************
;    Variables, constants defined and referenced locally
;    You can define your own values for a local reference here
;*********************************************************************

;*********************************************************************
;    Public routines (defined here)
;*********************************************************************
; routines

;*********************************************************************
;    Extern routines  (defined elsewhere)
;*********************************************************************
; routines

;*********************************************************
;     MACROs SUB-ROUTINES LIBRARY SECTION
;*********************************************************

;    (must be placed here and not at the file’s end)
```

```
;*************************************************************
;    Program code
;*************************************************************

    WORDS        ; define subsequent addresses as words
                 ; meaning that all instructions are located
                 ; in the address field after 0FFh in the ST72251
                 ; memory mapping


  segment 'rom'

.main

    ld A,#$01        ; negative edge sensitive and low level
    ld MISCR,A       ; for IT, ST72251 in normal mode

;************************************
    ld A,#$B8
    ld number_a,A
    ld A,#$01
    ld number_b,A

    call div_bxb
;************************************

;************************************
    ld A,#$F3
    ld operand_a,A
    ld A,#$D3
    ld {operand_a+1},A
    ld A,#$FC
    ld operand_b,A
    ld A,#$C3
    ld {operand_b+1},A

    call multiw
;************************************

;************************************
    ld A,#$0E
    ld dividend,A
    ld A,#$DC
    ld {dividend+1},A
```

```
    ld A,#$BA
    ld {dividend+2},A
    ld A,#$98
    ld {dividend+3},A
    ld A,#$AB
    ld divisor,A
    ld A,#$CD
    ld {divisor+1},A

    call div_lxw
;************************************

;************************************
    ld A,#$FC
    ld add_a,A
    ld A,#$03
    ld {add_a+1},A
    ld A,#$F3
    ld add_b,A
    ld A,#$C3
    ld {add_b+1},A

    call addw
;************************************

;************************************
    ld A,#$FC
    ld sub_a,A
    ld A,#$03
    ld {sub_a+1},A
    ld A,#$F3
    ld sub_b,A
    ld A,#$C3
    ld {sub_b+1},A

    call subw
;************************************

;************************************
    ld A,#$CC
    ld data,A
    ld A,#$04
    ld {data+1},A
    ld A,#$00
    ld min,A
```

```
    ld A,#$C3
    ld {min+1},A
    ld A,#$CC
    ld max,A
    ld A,#$05
    ld {max+1},A

    CALL check_min_max
;************************************

;************************************
    ld A,#$00
    ld data,A
    ld A,#$e2
    ld {data+1},A
    ld A,#$00
    ld delta,A
    ld A,#$23
    ld {delta+1},A
    ld A,#$CC
    ld median,A
    ld A,#$05
    ld {median+1},A

    call check_range
;************************************

;************************************
    ld A,#$D4

    call BtoD
;************************************

    jp main


;    ******************************************
;    *                        *
;    * INTERRUPT SUB-ROUTINES LIBRARY SECTION *
;    *                        *
;    ******************************************

dummy  iret      ; Empty subroutine. Go back to main (iret instruction)
```

```
;    ****************************************
;    *                      *
;    * CALL SUB-ROUTINES LIBRARY SECTION    *
;    *                      *
;    ****************************************

;+--------------------------------------------------------------------------
+
;|              DIVISION A / B                |
;|                                            |
;| DATE :           21/11/96                   |
;| REVISION :         V01.00                    |
;|                                            |
;| SOFTWARE DESCRIPTION :  This routine divides two 8 bit numbers, A and |
;|              B, the result is saved into an 8 bit register. |
;|              A and B >= 0.                 |
;|                                            |
;| INPUT PARAMETERS :     NUMBER_A register contains the number A.    |
;|              NUMBER_B register contains the number B.    |
;|                                            |
;| INTERNAL PARAMETERS :   COUNTER register contains the shift counter.  |
;|                                            |
;|                                            |
;| OUTPUT PARAMETERS :    RESULT register contains the result.      |
;|              REMAIND register contains the remainder.    |
;|                                            |
;|  BYTE :          52 bytes                   |
;|                                            |
;| EXAMPLE :         ;***** program *****             |
;|                  ld A,#$E7h                |
;|                  ld number_a,A              |
;|               ld A,#$10h             |               LD NUMBER_B,10h
|
;|                  ld number_b,A              |
;|                  CALL DIVISION              |
;|                  - do...                  |
;|                  - do...                  |
;|              ;***** subroutine *****            |
;|                  . div_bxb                |
;|                  END                    |
;|                                            |
;+--------------------------------------------------------------------------
+
```

```
.div_bxb

; Initialize registers and save the context.

     push A          ; save the accumulator.
     clr result      ; clear the result register.
     clr remaind     ; clear the remainder register.

; Test A and B in order to enable the division subroutine.

     ld A,number_b       ; if B = 0 then the result = 0 ( not valid ).
     jrugt div0      ; else test A
end_div pop A            ; restore the accumulator.
     ret             ; and exit from sub routine.
div0   ld A,number_a ; if A = 0 then the result = 0.
     jreq end_div        ; else we can start the division subroutine

; Division subroutine.

     rcf             ; clear the carry flag.
     ld A,#$08       ; load the shift counter.
     ld counter,A
     ld A,number_a       ; \
     rlc A           ; | rotate left and save in result register.
     ld result,A        ; /
div1   ld A,remaind        ; \
     rlc A           ; | rotate left the remainder register in
     ld remaind,A        ; / order to include the carry.
     sub A,number_b
     jrc div2         ; test if the subtraction is valid.
     ld remaind,A
div2   ld A,result         ; \
     rlc A           ; | rotate left the result register.
     ld result,A        ; /
     dec counter
     jreq div3
     jra div1
div3   cpl A           ; complement the result.
     ld result,A
     jra end_div
```

```
;+--------------------------------------------------------------------
+
;|              Multiplication A * B              |
;|                                      |
;| DATE :           21/11/96                    |
;| REVISION :         V01.00                     |
;|                                      |
;| SOFTWARE DESCRIPTION :   This routine multiplies two 16 bit numbers   |
;|              A and B, the result is saved into four 8-bit |
;|              registers (16x16= 32 bits)           |
;|              A and B >= 0.                  |
;|                                      |
;| INPUT PARAMETERS :     OPERAND_A registers contain the number A.    |
;|              OPERAND_B registers contain the number B.    |
;|                                      |
;|                                      |
;|                                      |
;| OUTPUT PARAMETERS :    res registers contain the result.       |
;|                                      |
;|                                      |
;| BYTE :           63 bytes                     |
;|                                      |
;| EXAMPLE :          ;***** program *****              |
;|                 ld A,#$F3                  |
;|                 ld operand_a,A               |
;|                 ld A,#$D3                  |
;|                 ld {operand_a+1},A             |
;|                 ld A,#$FC                  |
;|                 ld operand_b,A               |
;|                 ld A,#$C3                  |
;|                 ld {operand_b+1},A             |
;|                 CALL multiw                 |
;|                 - do...                   |
;|                 - do...                   |
;|              ;***** subroutine *****             |
;|                 . multiw                  |
;|                 END                     |
;|                                      |
;+--------------------------------------------------------------------
+

.multiw
    push A          ; save Accumulator in stack
    push X          ; save X register  in stack
```

```
ld X,operand_b      ; \
ld A,operand_a      ; | Multiplies MSB operand
mul X,A             ; /
ld res,X            ; and store in the 2 MSB result registers
ld {res+1},A

ld X,{operand_a+1}    ; \
ld A,{operand_b+1}    ; | Multiplies LSB operand
mul X,A             ; /
ld {res+2},X        ; and store in the 2 LSB result registers
ld {res+3},A

ld X,operand_a      ; \
ld A,{operand_b+1}    ; | Multiplies cross operands
mul X,A             ; /
add A,{res+2}       ; Add to previous result
ld {res+2},A
ld A,X
adc A,{res+1}
ld {res+1},A
ld A,res
adc A,#0
ld res,A

ld X,operand_b      ; \
ld A,{operand_a+1}    ; | Multiplies cross operands
mul X,A             ; /
add A,{res+2}       ; Add to previous result
ld {res+2},A
ld A,X
adc A,{res+1}
ld {res+1},A
ld A,res
adc A,#0
ld res,A

pop X               ; restore context before the CALL
pop A               ; restore context before the CALL
ret                 ; and go back to main program
```

```
;+--------------------------------------------------------------------------
+
;|              Long by Word division A/B            |
;|                                      |
;| DATE :           22/11/96                    |
;| REVISION :         V01.00                     |
;|                                      |
;| SOFTWARE DESCRIPTION :  This routine divides one 32-bit number A by  |
;|              a 16-bit number B. The result is saved in two |
;|              registers.                   |
;|              A and B >= 0.                  |
;|                                      |
;| INPUT PARAMETERS :    DIVIDEND registers contain the DIVIDEND (32 b).|
;|              DIVISOR registers contain the DIVISOR (16 b). |
;|                                      |
;| INTERNAL PARAMETERS :   TEMPQUOT registers contain the QUOTIENT    |
;|              temporary value (32 b).             |
;|                                      |
;| OUTPUT PARAMETERS :    QUOTIENT registers contain the result (16 b). |
;|              As the result is stored in 16 bits, this  |
;|              division is only valid for small numbers.
;            See use TEMPQUOT for the 32-bit result.|
;|                                      |
;| BYTE :          94 bytes                    |
;|                                      |
;| EXAMPLE :         ;***** program *****              |
;|                ld A,#$0E                 |
;|                ld dividend,A                |
;|                ld A,#$DC                 |
;|                ld {dividend+1},A              |
;|                ld A,#$BA                 |
;|                ld {dividend+2},A              |
;|                ld A,#$98                 |
;|                ld {dividend+3},A              |
;|                ld A,#$AB                 |
;|                ld divisor,A                |
;|                ld A,#$CD                 |
;|                ld {divisor+1},A              |
;|                CALL div_lxw                |
;|                - do...                  |
;|                - do...                  |
;|              ;***** subroutine *****             |
;|                . div_lxw                 |
;|                END                    |
;|                                      |
```

```
;+--------------------------------------------------------------------
.div_lxw

    push A          ; save Accumulator in stack
    push X          ; save X register in stack

    ld X,#32        ; Initialization process
    ld A,#0         ; We use the load instruction
    ld quotient,A   ; which is faster than the
    ld {quotient+1},A; clear instruction for
    ld tempquot,A   ; multiple short datas.
    ld {tempquot+1},A; For a smaller code size
    ld {tempquot+2},A; you'd better use the clear
    ld {tempquot+3},A; instruction

.execute
    sla {dividend+3}    ;Shift left dividend with 32 leading Zeros
    rlc {dividend+2}
    rlc {dividend+1}
    rlc dividend
    rlc {tempquot+3}
    rlc {tempquot+2}
    rlc {tempquot+1}
    rlc tempquot

    sla {quotient+1}     ; The result cannot be greater than 16 bits
    rlc quotient         ; so we can shift left the quotient

    ld A,tempquot        ; Test is left dividend is greater or equal
    or A,{tempquot+1}    ; to the divisor
    jrne dividendlsgreater

    ld A,{tempquot+2}
    cp A,divisor
    jrugt dividendlsgreater
    jrult nosubstract

    ld A,{tempquot+3}
    cp A,{divisor+1}
    jrult nosubstract

.dividendlsgreater       ; Substract divisor from left dividend
    ld A,{tempquot+3}
    sub A,{divisor+1}
    ld {tempquot+3},A
```

```
    ld A,{tempquot+2}
    sbc A,divisor
    ld {tempquot+2},A

    ld A,{tempquot+1}
    sbc A,#0
    ld {tempquot+1},A

    ld A,tempquot
    sbc A,#0
    ld tempquot,A

    inc {quotient+1}      ; The result cannot be greater than 16 bits
    jrne nosubstract      ; so we can increment the quotient
    inc quotient

.nosubstract
    dec X                 ; Decrement loop counter
    jrne execute          ; if X = 0 then exit else continue

    pop X                 ; restore contexte before the CALL
    pop A                 ; restore contexte before the CALL
    ret                   ; and go back to main program

;+--------------------------------------------------------------------------
;|              ADDITION A + B                 |
;|                                             |
;| DATE :           22/11/96                    |
;| REVISION :         V01.00                     |
;|                                             |
;| SOFTWARE DESCRIPTION :  This routine adds two 16 bit numbers, A and   |
;|              B, the result is saved in two 8 bits registers.|
;|              The carry flag indicates any overflow.     |
;|                                             |
;| INPUT PARAMETERS :    ADD_A registers contain the number A.      |
;|              ADD_B registers contain the number B.      |
;|                                             |
;| OUTPUT PARAMETERS :    RES_ADD register contains the result.      |
;|                                             |
;| BYTE :          17 bytes                     |
;|                                             |
;| EXAMPLE :         ;***** program *****              |
;|                 ld A,#$F3                 |
;|                 ld add_a,A                 |
```

```
;|                ld A,#$D3                |
;|                ld {add_a+1},A             |
;|                ld A,#$FC                |
;|                ld add_b,A                |
;|                ld A,#$C3                |
;|                ld {add_b+1},A             |
;|                 CALL addw               |
;|                 - do...                 |
;|                 - do...                 |
;|             ;***** subroutine *****           |
;|                 . addw                  |
;|                 END                     |
;|                                         |
;+-----------------------------------------------------------------------
.addw
    push A          ; save Accumulator in stack
    push X          ; save X register in stack

    ld A,{add_a+1}  ; get number A' LSB
    add A,{add_b+1} ; add number B' LSB
    ld {res_add+1},A; store LSB
    ld A,add_a      ; get number A' MSB
    adc A,add_b     ; add number B' MSB with LSB's carry
    ld res_add,A    ; store MSB

    pop X           ; restore context before the CALL
    pop A           ; restore context before the CALL
    ret             ; and go back to main program

;+-----------------------------------------------------------------------
;|             SUBSTRACTION A - B              |
;|                                         |
;| DATE :          22/11/96                |
;| REVISION :        V01.00                |
;|                                         |
;| SOFTWARE DESCRIPTION :  This routine substracts two 16 bit numbers, A |
;|             and B, the result is saved in two 8 bits    |
;|             registers. The carry flag indicates any     |
;|             overflow.                   |
;|                                         |
;| INPUT PARAMETERS :    sub_A registers contain the number A.     |
;|             sub_B registers contain the number B.     |
;|                                         |
;| OUTPUT PARAMETERS :   RES_sub register contain the result.      |
;|                                         |
```

```
;| BYTE :          17 bytes                    |
;|                                             |
;| EXAMPLE :        ;***** program *****       |
;|                 ld A,#$F3                   |
;|                 ld sub_a,A                  |
;|                 ld A,#$D3                   |
;|                 ld {sub_a+1},A              |
;|                 ld A,#$FC                   |
;|                 ld sub_b,A                  |
;|                 ld A,#$C3                   |
;|                 ld {sub_b+1},A              |
;|                  CALL subw                  |
;|                  - do...                    |
;|                  - do...                    |
;|              ;***** subroutine *****        |
;|                  .subw                      |
;|                  END                        |
;|                                             |
;+---------------------------------------------------------------------
.subw
    push A          ; save Accumulator in stack
    push X          ; save X register in stack

    ld A,{sub_a+1}  ; get number A' LSB
    sub A,{sub_b+1} ; sub number B' LSB
    ld {res_sub+1},A; store LSB
    ld A,sub_a      ; get number A' MSB
    sbc A,sub_b     ; sub number B' MSB with LSB's carry
    ld res_sub,A    ; store MSB

    pop X           ; restore context before the CALL
    pop A           ; restore context before the CALL
    ret             ; and go back to main program

;+---------------------------------------------------------------------
;|              CHECK MIN / MAX                |
;|                                             |
;| DATE :          22/11/96                    |
;| REVISION :        V01.00                    |
;|                                             |
;| SOFTWARE DESCRIPTION :  This routine tests if a 16-bit number |
;|              is within a predefined range.  |
;|                                             |
;|                MIN =< DATA =< MAX           |
;|                                             |
```

```
;| INPUT PARAMETERS :    DATA registers contain the number to test.   |
;|              MIN registers contain the minimum value.    |
;|              MAX registers contain the maximum value.    |
;[                                    |
;| OUTPUT PARAMETERS :    The C flag is updated according to the result.|
;|              C=1 means that the test has failed.       |
;|                                    |
;| BYTE :          32 bytes                    |
;|                                    |
;| EXAMPLE :         ;***** program *****              |
;|                 ld A,#$25                |
;|                 ld data,A                |
;|                 ld A,#$00                |
;|                 ld {data+1},A              |
;|                 ld A,#$00                |
;|                 ld min,A                 |
;|                 ld A,#$C3                |
;|                 ld {min+1},A               |
;|                 ld A,#$CC                |
;|                 ld max,A                 |
;|                 ld A,#$05                |
;|                 ld {max+1},A               |
;|                  CALL check_min_max           |
;|                  - do...                |
;|                  - do...                |
;|              ;***** subroutine *****            |
;|                  .check_min_max             |
;|                  END                   |
;|                                    |
;+--------------------------------------------------------------------------
.check_min_max

    push A          ; save Accumulator in stack
    push X          ; save X register  in stack

    ld X,data       ; get DATA MSB in X
    ld A,{data+1}   ; get DATA LSB in A

    cp X,max        ; Compare MSB with MAX
    jrugt out_of_range    ; if greater than exit
    jrne comp_min   ; else if equals compare LSB
    cp A,{max+1}
    jrugt out_of_range; LSB greater than exit

comp_min
```

```
    cp X,min        ; same thing with the LSB and the min value
    jrult out_of_range
    jrne in_range
    cp A,{min+1}
    jrult out_of_range

in_range
    rcf             ; Value in range so reset C flag
    jra exit        ; the value is within the two values

out_of_range
    scf             ; Value out of range so set C flag

exit
    pop X           ; restore contexte before the CALL
    pop A           ; restore contexte before the CALL
    ret             ; and go back to main program

;+--------------------------------------------------------------------------
;|              CHECK RANGE for a WORD              |
;|                                        |
;| DATE :           22/11/96                    |
;| REVISION :         V01.00                     |
;|                                        |
;| SOFTWARE DESCRIPTION :  This routine tests if a 16-bit number |
;|              is within a predefined range          |
;|                                        |
;|                MEDIAN - DELTA =< DATA  =< MEDIAN + DELTA  |
;|                                        |
;| INPUT PARAMETERS :    DATA registers contain the number to test.   |
;|              MEDIAN registers contain the median value.   |
;|              DELTA registers contain the delta value to add |
;|              and subtract to the MEDIAN value.       |
;[                                        |
;| OUTPUT PARAMETERS :    The C flag is updated according to the result. |
;|              C=1 means that the test has failed.       |
;|                                        |
;|                                        |
;| NOTES:           This routines uses three previous sub routines.|
;|                                        |
;|              check_min_max                  |
;|              addw                        |
;|              subw                        |
;|                                        |
;|                                        |
```

```
;| BYTE :          66 bytes                      |
;|                                          |
;| EXAMPLE :         ;***** program *****              |
;|                  ld A,#$25                  |
;|                  ld data,A                  |
;|                  ld A,#$00                  |
;|                  ld {data+1},A                |
;|                  ld A,#$00                  |
;|                  ld delta,A                 |
;|                  ld A,#$23                  |
;|                  ld {delta+1},A               |
;|                  ld A,#$CC                  |
;|                  ld median,A                |
;|                  ld A,#$05                  |
;|                  ld {median+1},A              |
;|                   CALL check_range            |
;|                   - do...                  |
;|                   - do...                  |
;|               ;***** subroutine *****             |
;|                   .addw                    |
;|                   .subw                    |
;|                   .check_min_max             |
;|                   .check_range               |
;|                   END                     |
;+-------------------------------------------------------------------------
.check_range
    push X
    push A

    ld A,median     ; get MSB value and store it in
    ld add_a,A      ; add_a and sub_a for the words
    ld sub_a,A      ; subroutines (addw and subw)
    ld A,{median+1} ; Same thing for the LSB
    ld {add_a+1},A
    ld {sub_a+1},A

    ld A,delta      ; The second operand is for delta.
    ld add_b,A
    ld sub_b,A
    ld A,{delta+1}
    ld {add_b+1},A
    ld {sub_b+1},A

    call addw       ; Compute Median + delta
```

```
    jrnc no_ovfmax     ; test if an overflow occurred
    ld A,#$FF       ; if yes then the MAX value is set to FFFFh
    ld max,A        ; (saturation)
    ld {max+1},A

no_ovfmax
    ld A,res_add    ; else there is no overflow, then
    ld max,A        ; the computed value is the MAX value to keep.
    ld A,{res_add+1}
    ld {max+1},A

    call subw       ; Compute Median - delta

    jrnc no_ovfmin  ; test if an overflow occured
    clr A           ; if yes then the MIN value is set to 0000h
    ld min,A        ; (saturation)
    ld {min+1},A

no_ovfmin
    ld A,res_sub    ; else there is no overflow, then
    ld min,A        ; the computed value is the MIN value to keep.
    ld A,{res_sub+1}
    ld {min+1},A

    call check_min_max    ; Then we check if the value is within the range
                          ; set by max and min.

    pop A           ; restore context before the CALL
    pop X           ; restore context before the CALL
    ret             ; The result depends on the C flag.

;+--------------------------------------------------------------------------
;|              Binary to Decimal Conversion          |
;|                                       |
;| DATE :           25/11/96                    |
;| REVISION :         V01.00                     |
;|                                       |
;| SOFTWARE DESCRIPTION :  This routine performs a BCD (binary to decimal)|
;|              conversion.                    |
;|                                       |
;|                                       |
;| INPUT PARAMETERS :    Value (to convert) stored in the accumulator. |
;|                                       |
;|                                       |
;|                                       |
```

```
;| OUTPUT PARAMETERS :    Hundreds register holds the number of hundreds|
;|              Tens register holds the number of both tens   |
;|            (in the high nibble) and the number of units  |
|
;|              in the low nibble)               |
;| EXAMPLE:          If A=D4h (212d) the routine will output the  |
;|              following results:               |
;|                                      |
;|                Hundreds: 02  and   Tens: 12      |
;|                                 ||      |
;|                               tens|      |
;|                                 units    |
;|                                      |
;| BYTE :          31 bytes                     |
;|                                      |
;| EXAMPLE :         ;***** program *****              |
;|                 ld A,#$D4                |
;|                  CALL BtoD               |
;|                  - do...                 |
;|                  - do...                 |
;|              ;***** subroutine *****             |
;|                  .BtoB                   |
;|                  END                     |
;+-------------------------------------------------------------------------
.BtoD
    clr hundreds    ; clear registers used
    clr tens

hund   sub A,#100    ; A = A - 100
    jrc ten         ; Test if A < 100
    inc hundreds    ; No then hundreds = hundreds + 1
    jra hund        ; and loop
ten   add A,#100     ; Add 100 to set off last subtraction
temp   sub A,#10     ; A = A - 10
    jrc unit        ; Test if A < 10
    inc tens        ; No then tens = tens + 1
    jra temp        ; and loop
unit   add A,#10     ; Add 10 to set off last subtraction
    swap tens       ; swap nibbles (MSB nibble = TENS)
    OR A,tens       ; use mask to keep MSB nibble
    ld tens,A       ; store the tens/units value (LSB nibble = Units)
                    ; in one byte
    ret             ; End of sub routine
```

```
        segment 'vectit'


          DC.W  dummy    ;FFE0-FFE1h location
          DC.W  dummy    ;FFE2-FFE3h location
.i2c_it     DC.W  i2c_rt  ;FFE4-FFE5h location
          DC.W  dummy    ;FFE6-FFE7h location
          DC.W  dummy    ;FFE8-FFE9h location
          DC.W  dummy    ;FFEA-FFEBh location
          DC.W  dummy    ;FFEC-FFEDh location
.timb_it    DC.W  timb_rt;FFEE-FFEFh location
          DC.W  dummy    ;FFF0-FFF1h location
.tima_it    DC.W  tima_rt;FFF2-FFF3h location
.spi_it     DC.W  spi_rt  ;FFF4-FFF5h location
          DC.W  dummy    ;FFF6-FFF7h location
.ext1_it    DC.W  ext1_rt;FFF8-FFF9h location
.ext0_it    DC.W  ext0_rt;FFFA-FFFBh location
.softit   DC.W  sw_rt    ;FFFC-FFFDh location
.reset    DC.W  main     ;FFFE-FFFFh location
      ; This last line refers to the first line.
      ; It used by the compiler/linker to determine code zone
    END ; Be aware of the fact that the END directive should not
      ; stand on the left of the page like the labels's names.
```

"THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS."