binary division with remainder

```
                         Arithmetic Logic Unit (ALU 9+8 Bit)
  ____________________________________________________________________________________
      bit 1                  bit
      sign extension         123456789        fta
                                              000         control bits fta
                                         00000000 ALS     shifting pointer
           operand 1     AL1 000000000  00000000 ALD     dividend shift register
           operand 1     AL2 000000000  00000000 ALN     negative divisor
           carryover     ALC 000000000  00000000 ALP     positive divisor
       remainder by DIV  ALR 000000000  00000000 ARL     result at DIV
      (result by add)        9 bit      8 bit       (1+16bit result low byte)
  _____________________________________________________________________________________
```

sequence control with control bits: f (first loop)   t (two)   a (add)

f: First total run in bit comparison of dividend in AL1 and divisor in AL2 (f=1).
   The dividend is shifted to the right into the ALD auxiliary register of the dividend if required.
   In the shifting pointer register ALS, a 1 is inserted from the left.

t: In the first subsequence the bits are compared from the left, possibly shifted.
   This means that the subsequent bits in the second subprocess must also be checked
   to determine the size, t=1 is set.

a: Solution bit 1 is appended to the quotient, then addition of the negative divisor (current remainder).
   A bit from the auxiliary register ALD of the dividend is shifted.
   This is only done if there is still a 1 in the shifting pointer ALS above it on the left.
   The addition (add) is done with the 9 bit registers.

the division algorithm - for positive operands -

```
-------------------------------------------------------------------------------------------
    set ALN negative divisor          // to be calculated in advance, 0 is to be intercepted beforehand
    set ALP divisor                   // 1 is to be intercepted before, x\1 = x remainder 0
    set ALS 00000000                  // initialize shifting pointer
    set ALD 00000000                  // initialize dividend shift register
    set ARL 00000000                  // initialize result (quotient)
    set ALf 1                         // preset first total pass (outer loop)
    set ALt 0                         // preset no second partial process (partial loop)
    set ALa 0                         // preset no addition of the negative divisor
    set AL1 0+dividend                // with sign extension 0
    set AL2 0+divisor                 // with sign extension 0
    // ----------------------------------------------------------------------------------
    // main loop, abort only with break, place partial dividend over divisor
    loop
      bit = 3     // check bits AL1, AL2 from the left, without sign extension bit 1 and sign bit 2 (0 positive)
      // first partial loop, abort only with break
      loop
        b1 = get(AL1,bit)                               // read out current bit
        b2 = get(AL2,bit)                               // read out current bit
        bs = getleft(ALS,1)                             // left bit shift counter  (bs=1, shift left possible)
        // ---------------
        if b1=0 ∧ b2=0
          if bit=9
            // at bit=9 division 0\0 would not be defined, to be intercepted before!
          else
            increment bit                               // pass leading zeros
          endif // bit
        endif
        // ---------------
```

```
  // ---------------
  if b1=0 ∧ b2=1
    if bs=1    // Shift possible, shift left, shift along ALS (observe AL1 9 bit)
      set AL1 getright(AL1,8)+getleft(ALD,1)         //   << AL1_ALD(1)  (Do not increase the bit counter!)
      set ALD getright(ALD,7)+0                      //   << ALD(2..8)_0
      set ALS getright(ALS,7)+0                      //   shift the shift counter to the left, append 0
      set ARL getright(ARL,7)+0                      //   append solution bit 0 (for shift position left)
    else       //  no more left shifting possible, dividend < divisor
      set ALt 0                                      // not a second loop,
      set ALa 0                                      // no addition negative divisor
      set ALR AL1                                    // current remainder from AL1 to ALR
      set ARL getright(ARL,7)+0                      // append solution bit 0
      break
    endif // bs
  endif
  // ---------------
  if b1=1 ∧ b2=0
    if bit=9
      // at bit=9 division 1\0 would not be defined, to be intercepted before!
    else
      if ALf=1     // only on first pass in outer loop: left position dividend to divisor
        set ALD getright(AL1,1)+getleft(ALD,7)       //   >> AL1(9)_ALD
        set AL1 0 + getleft(AL1,8)                   //   >> 0_AL1(1..8)
        set ALS 1 + getleft(ALS,7)                   // fill shift counter with 1 from left
        increment bit
      else
        set ALt 0                                    // no second loop
        set ALa 1                                    // addition negative divisor
        set ARL getright(ARL,7)+1                    // append solution bit 1
        break
      endif // ALf
    endif // bit
  endif
  // ---------------
  if b1=1 ∧ b2=1
    if bit=9
      // at bit=9 division 1\1 = 1 remainder 0 division by 1 would have to be intercepted before!
    else
      set ALt 1                                      // second loop (since 1 found over 1)
      increment bit                                  //        ... from following bit
      break
    endif // bit
  endif
  //----------------
end_loop // first partial loop
//***************************
```

```
if ALt=1                     //  <---  possibly run through subloop two
//****************************
  loop  //  second partial loop, abort only with break, check further bits to the right
    b1 = get(AL1,bit)                                // read out current bit
    b2 = get(AL2,bit)                                // read out current bit
    bs = getleft(ALS)                                // Shift counter left bit (1 shift possible)
    // ---------------
    if b1=0 ∧ b2=0
      if bit=9
        if bs=1              // only interim result  // still possible to shift
          set ALa 1                                  // addition negative divisor
        else //   letzte Teilrechnung
          set ALa 0                                  // kno addition negative divisor
          set ALR 000000000                          // remainder 0 to ALR
        endif // bs
        set ARL getright(ARL,7)+1                    // append solution bit
        break
      else
         increment bit
      endif // bit
    endif
    // ---------------
    if b1=0 ∧ b2=1
      if bs=1     // shift left possible, shift ALS along, (note AL1 9 bit)
        set AL1 getright(AL1,8)+getleft(ALD,1)       //   << AL1_ALD(1) (Do not increase the bit counter!)
        set ALD getright(ALD,7)+0                    //   << ALD(2..8)_0
        set ALS getright(ALS,7)+0                    // shift counter: shift to the left, append 0
        set ARL getright(ARL,7)+0                    // append solution bit 0 (for: shift digit left)
        set ARL getright(ARL,7)+1                    // append solution bit 1 (for 'division')
        set ALa 1                                    // addition negative divisor, position reached
        break
      else
        set ALa 0                                    // no shifting possible, no addition negative divisor
        set ARL getright(ARL,7)+0                    // append solution bit 0
        set ALR AL1                                  // Remainder to ALR
        break
      endif // bs
    endif
    // ---------------
    if b1=1 ∧ b2=0
      set ARL getright(ARL,7)+1                      // append solution bit 1
      set ALa 1                                      // addition negative divisor, position reached
      break                                          // dividend greater than divisor
    endif
    // ---------------
```

```
        // ---------------
        if b1=1 ∧ b2=1
          if bit=9
            if bs=1              // only interim result  // still possible to shift
              set ALa 1                                  // addition negative divisor
            else //   letzte Teilrechnung
              set ALa 0                                  // no addition negative divisor
              set ALR 000000000                          // remainder zero to ALR
            endif // bs
            set ARL getright(ARL,7)+1                    // append solution bit
            break
          else
            increment bit
          endif // bit
        endif
        // ---------------
      end_loop // second partial loop
      // ****************************
    endif  // ALt
    // ==================================================
    //
    if ALa=1                     //  <---  possible addition of negative divisor
      //
      set AL1 getright(AL1,8)                            // sign extension occurs with addition
      set AL2 ALN                                        // negative divisor  to AL2 for addition
      // Perform addition in the ALU, the sign bit of AL1 and AL2 is duplicated in advance.
      //
      add   //  <-- Addition algorithm add_ALU_9 (AL1+AL2) in 9 bit arithmetic unit with sign extension.
      //
      bs = getleft(ALS,1)                                // shift counter left bit
      if bs=1                    // shift counter left bit is 1, bit of the dividend shift possible
        set AL1 ALR                                      // difference in ALR is remainder, set again in AL1
        set AL2 0+ALP                                    // Set positive divisor with sign extension in AL2
        //                         ... another bit of the dividend - postpone
        set AL1 getright(AL1,8)+getleft(ALD,1)           //   << AL1_ALD(1)  (Do not increase the bit counter!)
        set ALD getright(ALD,7)+0                        //   << ALD(2..8)_0
        set ALS getright(ALS,7)+0                        // shift counter: shift left, append 0
      else
        break                                            // ready, remainder in ALR
      endif // bs
    else //  no addition negative divisor
      break                                              // remainder in ALR
    endif // ALa
    //==================================================
    if  ALf=1
      set ALf 0          // indicator 0 for further passes of the outer loop after first pass
    endif // ALf
  end_loop // outer loop
  //
  //Remainder in ALR, quotient in ARL
```

____

```
// NOTES:      Restriction of the number range -128 ... 127 decimal,
//             -128 is not possible as an operand and leads to an overflow when forming the two's complement.
//             The decimal numbers 0 and 1 as divisor are to be intercepted before.
```