AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR

- 1. AVR ASM BIT MANIPULATION
- 2a. BASIC ARITHMETIC
- **2b. BASIC MATH**
- 2c. LOGARITHMS
- 2z. INTEGER RATIOS for FASTER CODE
- 3a. USING THE ADC
- 3b. BUTTERFLY ADC
- 4a. USING THE EEPROM
- 4b. BUTTERFLY EEPROM
- 5. TIMER COUNTERS & PWM
- 6. BUTTERFLY LCD & JOYSTICK
- 7. BUTTERFLY SPI & AT45 DATAFLASH

Sitemap

AVR ASSEMBLER TUTOR >

2b. BASIC MATH

BASIC AVR ARITHMETIC v1.7

MULTIPLICATION, DIVISION, SQUARE & CUBE ROOT

by RetroDan@GMail.Com

CONTENTS:

1. MULTIPLIYING TWO SINGLE-BYTE NUMBERS WITH MUL COMMAND

If your AVR chip supports the multiplication command (MUL) then multiplying two eight-bit numbers is quite simple. MUL will work on all 32 registers R0 to R31 and leave the low-byte of the result in R0 and the high-byte in R1. The registers for multiplicand and multiplier remain unchanged. The routine takes about three cycles.

```
.DEF ANSL = R0            ;To hold low-byte of answer
.DEF ANSH = R1            ;To hold high-byte of answe
.DEF    A = R16           ;To hold multiplicand
.DEF    B = R18           ;To hold multiplier

        LDI A,42          ;Load multiplicand into A
        LDI B,10          ;Load multiplier into B
        MUL A,B           ;Multiply contents of A and
                          ;Result 420 left in ANSL an
```

2. MULTIPLYING A SINGLE-BYTE NUMBER BY POWER OF TWO

If our AVR does not support the hardware MUL command we will have to compute multiplications manually. If we need to multiply by a power of two such as 2,4,8 etc. The result can be achived by shifting bits to the left. Each shift to the left is a multiplication by two.

The Logical Shift Left (LSL) command is used on the lower byte because it will shift the contents one bit to the left, a zero is shifted into the lowest bit and the highest bit is shifted into the carry flag.

```
              10101010
Carry [1] <-- 01010100 <--0 (LSL)
```

We use the Rotate Left though Carry (ROL) command on the high byte because it will also shift contents one bit to the left, but it will shift the contents of the Carry Flag into the lowest bit.

```
              00000000
Carry [0] <-- 00000001 <--[1] Carry (ROL)
```

Every time we shift the multiplicand to the left we are multiplying it by two. So to multiply by eight we simply shift the multiplicand to the left three times. The routine takes about ten cycles.

```
.DEF ANSL = R0            ;To hold low-byte of answer
.DEF ANSH = R1            ;To hold high-byte of answe
.DEF   AL = R16           ;To hold low-byte of multip
.DEF   AH = R17           ;To hold high-byte of multi

        LDI AL,LOW(42)    ;Load multiplicand into AH:
        LDI AH,HIGH(42)   ;
MUL8:
        MOV ANSL,AL       ;Copy multiplicand into R1:
        MOV ANSH,AH       ;
        LSL ANSL          ;Multiply by 2
        ROL ANSH          ;Shift the Carry into R1
        LSL ANSL          ;Multiply by 2x2=4
        ROL ANSH          ;Shift the Carry into R1
        LSL ANSL          ;Multiply by 2x2x2=8
        ROL ANSH          ;Shift the Carry into R1
                          ;Result 42x8=336 left in AN
```

3. MANUALLY MULTIPLYING TWO SINGLE-BYTE NUMBERS

To do standard multiplication we examine how binary multiplication is achieved, we notice that when a digit in the multiplier is a one we add a shifted version of the multiplicand to our result. When the multiplier digit is a zero we need to add zero, which means we do nothing.

```
         00101010 = 42 multiplicand
        x00001010 = 10 multiplier
         --------
         00000000
        00101010
       00000000
      00101010
     00000000
    00000000
   00000000
  00000000
 ----------------
 0000000110100100 = 420 result
 ================
```

The routine below mimics the hardware multiply (MUL) by leaving the multiplicand and multiplier untouched, and the result appears in the register pair R1 and R0. It shifts the bits of the multiplier into the carry bit and uses the contents of the carry to add the multiplicand if it is a one or skip it if the carry is a zero. The routine takes about sixty cycles.

```
.DEF ANSL = R0            ;To hold low-byte of answer
.DEF ANSH = R1            ;To hold high-byte of answe
.DEF    A = R16           ;To hold multiplicand
.DEF    B = R18           ;To hold multiplier
.DEF    C = R20           ;To hold bit counter

        LDI A,42          ;Load multiplicand into A
        LDI B,10          ;Load multiplier into B
MUL8x8:
        LDI C,8           ;Load bit counter into C
        CLR ANSH          ;Clear high-byte of answer
        MOV ANSL,B        ;Copy multiplier into low-b
        LSR ANSL          ;Shift low-bit of multiplie
LOOP:    BRCC SKIP        ;If carry is zero then skip
        ADD ANSH,A        ;Add multiplicand to answer
SKIP:   ROR ANSH          ;Shift low-bit of high-byte
        ROR ANSL          ;of answer into low-byte
        DEC C             ;Decrement bit-counter
         BRNE LOOP        ;Check if done all eight bi
                          ;Result 420 left in ANSL an
```

4. MULTIPLYING TWO 16-BIT NUMBERS WITH THE MUL COMMAND

Multiplying two 16-bit numbers can result in a four-byte result. We use the hardware multiply (MUL) command to create all four cross products and add them to the 32-bit result. The MUL command leaves it results each time in R1:R0 which we then add into our result. The routine takes about twenty cycles.

```
.DEF ZERO = R2               ;To hold Zero
.DEF   AL = R16              ;To hold multiplicand
.DEF   AH = R17
```

```
.DEF   BL = R18             ;To hold multiplier
.DEF   BH = R19
.DEF ANS1 = R20             ;To hold 32 bit answer
.DEF ANS2 = R21
.DEF ANS3 = R22
.DEF ANS4 = R23

        LDI AL,LOW(42)      ;Load multiplicand into
        LDI AH,HIGH(42)     ;
        LDI BL,LOW(10)      ;Load multiplier into BH
        LDI BH,HIGH(10)     ;

MUL16x16:
        CLR ZERO            ;Set R2 to zero
        MUL AH,BH           ;Multiply high bytes AHx
        MOVW ANS4:ANS3,R1:R0 ;Move two-byte result in

        MUL AL,BL           ;Multiply low bytes ALxB
        MOVW ANS2:ANS1,R1:R0 ;Move two-byte result in

        MUL AH,BL           ;Multiply AHxBL
        ADD ANS2,R0         ;Add result to answer
        ADC ANS3,R1         ;
        ADC ANS4,ZERO       ;Add the Carry Bit

        MUL BH,AL           ;Multiply BHxAL
        ADD ANS2,R0         ;Add result to answer
        ADC ANS3,R1         ;
        ADC ANS4,ZERO       ;Add the Carry Bit
```

5. MULTIPLYING TWO 16-BIT NUMBERS MANUALLY

Multiplying Two-Byte numbers together can leave a result that is four-bytes wide (32-bits). With this routine we add the multiplicand to the high-bytes of our result for each one that appears in our 16-bit multiplier, then shift the result into the lower bytes of our result sixteen times, once for each bit of our multiplier. The routine takes about 180 cycles.

```
.DEF   AL = R16              ;To hold multiplicand
.DEF   AH = R17
.DEF   BL = R18              ;To hold multiplier
.DEF   BH = R19
.DEF ANS1 = R20              ;To hold 32 bit answer
.DEF ANS2 = R21
.DEF ANS3 = R22
.DEF ANS4 = R23
.DEF    C = R24              ;Bit Counter

        LDI AL,LOW(42)       ;Load multiplicand into
        LDI AH,HIGH(42)      ;
        LDI BL,LOW(10)       ;Load multiplier into BH
        LDI BH,HIGH(10)      ;

MUL16x16:
        CLR ANS3             ;Set high bytes of resul
        CLR ANS4             ;
        LDI C,16             ;Bit Counter
LOOP:   LSR BH               ;Shift Multiplier to the
        ROR BL               ;Shift lowest bit into C
         BRCC SKIP           ;If carry is zero skip a
        ADD ANS3,AL          ;Add Multiplicand into h
        ADC ANS4,AH          ;of the Result
SKIP:   ROR ANS4             ;Rotate high bytes of re
        ROR ANS3             ;the lower bytes
        ROR ANS2             ;
        ROR ANS1             ;
        DEC C                ;Check if all 16 bits ha
         BRNE LOOP           ;If not then loop back
```

6. MULTIPLYING TWO 32-BIT NUMBERS MANUALLY

The following routine will multiply two 32-bit numbers with a 64-Bit (8 Byte) result. The routine takes about 500 clock cycles.

```
 .DEF  ANS1 = R0          ;64-Bit Answer
 .DEF  ANS2 = R1          ;
 .DEF  ANS3 = R2          ;
```

```
 .DEF  ANS4 = R3          ;
 .DEF  ANS5 = R4          ;
 .DEF  ANS6 = R5          ;
 .DEF  ANS7 = R6          ;
 .DEF  ANS8 = R7          ;
 .DEF    A1 = R16         ;Multiplicand
 .DEF    A2 = R17         ;
 .DEF    A3 = R18         ;
 .DEF    A4 = R19         ;
 .DEF    B1 = R20         ;Multiplier
 .DEF    B2 = R21         ;
 .DEF    B3 = R22         ;
 .DEF    B4 = R23         ;
 .DEF     C = R24         ;Loop Counter

        LDI   A1,  LOW($FFFFFFFF)
        LDI   A2,BYTE2($FFFFFFFF)
        LDI   A3,BYTE3($FFFFFFFF)
        LDI   A4,BYTE4($FFFFFFFF)
        LDI   B1,  LOW($FFFFFFFF)
        LDI   B2,BYTE2($FFFFFFFF)
        LDI   B3,BYTE3($FFFFFFFF)
        LDI   B4,BYTE4($FFFFFFFF)
MUL3232:
        CLR   ANS1         ;Initialize Answer to zero
        CLR   ANS2         ;
        CLR   ANS3         ;
        CLR   ANS4         ;
        CLR   ANS5         ;
        CLR   ANS6         ;
        CLR   ANS7         ;
        SUB   ANS8,ANS8    ;Clear ANS8 and Carry Flag
        MOV   ANS1,B1      ;Copy Multiplier to Answer
        MOV   ANS2,B2      ;
        MOV   ANS3,B3      ;
        MOV   ANS4,B4      ;
        LDI   C,33         ;Set Loop Counter to 33
LOOP:
        ROR   ANS4         ;Shift Multiplier to right
        ROR   ANS3         ;
```

```
        ROR   ANS2         ;
        ROR   ANS1         ;
        DEC   C            ;Decrement Loop Counter
         BREQ DONE         ;Check if all bits process
          BRCC SKIP        ;If Carry Clear skip addit
        ADD   ANS5,A1      ;Add Multipicand into Answ
        ADC   ANS6,A2      ;
        ADC   ANS7,A3      ;
        ADC   ANS8,A4      ;
SKIP:
        ROR   ANS8         ;Shift high bytes of Answe
        ROR   ANS7         ;
        ROR   ANS6         ;
        ROR   ANS5         ;
         RJMP LOOP
DONE:
```

7. DIVIDING BY A POWER OF TWO

Since there is no hardware divide command, we will have to do it manually. If we need to divide by a power of two such as 2,4,8 etc. The result can be achieved by shifting bits to the right. Each shift to the right is a division by two.

The Logical Shift Right (LSR) command is used on the higher byte because it will shift the contents one bit to the right, a zero is shifted into the highest bit and the lowest bit is shifted into the carry flag.

```
              01010101
         0--> 00101010 -->[1] Carry
```

We use the Rotate Right though Carry (ROR) command on the low byte because it will also shift contents one bit to the right, but it will shift the contents of the Carry Flag into the highest bit.

```
              00000000
Carry [1] --> 10000000 -->[0] Carry (ROL)
```

Every time we shift the multiplicand to the right we are dividing it by two. So to divide by eight we simply shift the multiplicand to the right three times. The routine takes about ten cycles.

```
.DEF ANSL = R0            ;To hold low-byte of answer
.DEF ANSH = R1            ;To hold high-byte of answe
.DEF   AL = R16           ;To hold low-byte of multip
.DEF   AH = R17           ;To hold high-byte of multi

        LDI AL,LOW(416)   ;Load multiplicand into A
        LDI AH,HIGH(416)  ;
DIV8:
        MOVW ANSH:ANSL,AH:AL ;Copy multiplicand into
        LSR ANSH          ;Divide by 2
        ROR ANSL          ;Shift Carry Flag to R0
        LSR ANSH          ;Divide by 4 (2x2)
        ROR ANSL          ;Shift Carry Flag into R0
        LSR ANSH          ;Divide by 8 (2x2x2)
        ROR ANSL          ;Shift Carry Flag into R0
                          ;Result 416/8=52 left in AN
```

8. DIVIDING TWO SINGLE-BYTE NUMBERS

Just as multiplication can be achieved with shifting and addition. Dividing can be accomplished by shifting and subtraction. The routine below tries to repeatedly subtract the divisor. If the result is negative it reverses the process and shifts the dividend to the left to try again. The routine takes about 90 cycles.

```
.DEF  ANS = R0            ;To hold answer
.DEF  REM = R2            ;To hold remainder
.DEF    A = R16           ;To hold dividend
.DEF    B = R18           ;To hold divisor
.DEF    C = R20           ;Bit Counter

        LDI A,255         ;Load dividend into A
        LDI B,5           ;Load divisor into B
DIV88:
        LDI C,9           ;Load bit counter
        SUB REM,REM       ;Clear Remainder and Carry
        MOV ANS,A         ;Copy Dividend to Answer
LOOP:   ROL ANS           ;Shift the answer to the le
        DEC C             ;Decrement Counter
         BREQ DONE        ;Exit if eight bits done
```

```
        ROL REM          ;Shift the remainder to the
        SUB REM,B        ;Try to Subtract divisor fr
         BRCC SKIP       ;If the result was negative
        ADD REM,B        ;reverse the subtraction to
        CLC              ;Clear Carry Flag so zero s
         RJMP LOOP       ;Loop Back
SKIP:   SEC              ;Set Carry Flag to be shift
         RJMP LOOP
DONE:
```

9. DIVIDING TWO 16-BIT NUMBERS

The previous routine can be expanded to handle divide two-byte numbers in the range of zero to 65,535. The routine takes about 230 cycles.

```
.DEF ANSL = R0           ;To hold low-byte of answer
.DEF ANSH = R1           ;To hold high-byte of answe
.DEF REML = R2           ;To hold low-byte of remain
.DEF REMH = R3           ;To hold high-byte of remai
.DEF   AL = R16          ;To hold low-byte of divide
.DEF   AH = R17          ;To hold high-byte of divid
.DEF   BL = R18          ;To hold low-byte of diviso
.DEF   BH = R19          ;To hold high-byte of divis
.DEF    C = R20          ;Bit Counter

        LDI AL,LOW(420)   ;Load low-byte of dividend
        LDI AH,HIGH(420)  ;Load HIGH-byte of dividend
        LDI BL,LOW(10)    ;Load low-byte of divisor i
        LDI BH,HIGH(10)   ;Load high-byte of divisor
DIV1616:
        MOVW ANSH:ANSL,AH:AL ;Copy dividend into answ
        LDI C,17          ;Load bit counter
        SUB REML,REML     ;Clear Remainder and Carry
        CLR REMH          ;
LOOP:   ROL ANSL          ;Shift the answer to the le
        ROL ANSH          ;
        DEC C             ;Decrement Counter
         BREQ DONE        ;Exit if sixteen bits done
        ROL REML          ;Shift remainder to the lef
```

```
        ROL REMH            ;
        SUB REML,BL         ;Try to subtract divisor fr
        SBC REMH,BH
         BRCC SKIP          ;If the result was negative
        ADD REML,BL         ;reverse the subtraction to
        ADC REMH,BH         ;
        CLC                 ;Clear Carry Flag so zero s
         RJMP LOOP          ;Loop Back
SKIP:   SEC                 ;Set Carry Flag to be shift
         RJMP LOOP
DONE:
```

10. DIVIDING A 32-BIT NUMBER

The previous routine can be futher expanded to handle 32-bit number divided by a 16-bits number, This means numbers in the range of zero to 4,294,967,295 (4.3 billion) divided by numbers in the range zero to 65,535. The routine takes about 700 cycles.

```
.DEF ANS1 = R0              ;To hold low-byte of answer
.DEF ANS2 = R1              ;To hold second-byte of ansv
.DEF ANS3 = R2              ;To hold third-byte of answ
.DEF ANS4 = R3              ;To hold fourth-byte of ansv

.DEF REM1 = R4              ;To hold first-byte of rema
.DEF REM2 = R5              ;To hold second-byte of rem
.DEF REM3 = R6              ;To hold third-byte of rema
.DEF REM4 = R7              ;To hold fourth-byte of rem

.DEF ZERO = R8              ;To hold the value zero

.DEF   A1 = R16             ;To hold low-byte of divide
.DEF   A2 = R17             ;To hold second-byte of div
.DEF   A3 = R18             ;To hold third-byte of divi
.DEF   A4 = R19             ;To hold fourth-byte of div

.DEF   BL = R20             ;To hold low-byte of diviso
.DEF   BH = R21             ;To hold high-byte of divis

.DEF    C = R22             ;Bit Counter
```

```
        LDI A1,LOW(420)   ;Load low-byte of dividend
        LDI A2,BYTE2(420) ;Load second-byte of divide
        LDI A3,BYTE3(420) ;Load third-byte of dividen
        LDI A4,BYTE4(420) ;Load fourth-byte of divide

        LDI BL,LOW(10)    ;Load low-byte of divisor i
        LDI BH,HIGH(10)   ;Load high-byte of divisor

DIV3216:
        CLR ZERO
        MOVW ANS2:ANS1,A2:A1 ;Copy dividend into answ
        MOVW ANS4:ANS3,A4:A3 ;
        LDI C,33          ;Load bit counter
        SUB REM1,REM1     ;Clear Remainder and Carry
        CLR REM2          ;
        CLR REM3          ;
        CLR REM4          ;
LOOP:   ROL ANS1          ;Shift the answer to the le
        ROL ANS2          ;
        ROL ANS3          ;
        ROL ANS4          ;
        DEC C             ;Decrement Counter
         BREQ DONE        ;Exit if 32 bits done
        ROL REM1          ;Shift remainder to the lef
        ROL REM2          ;
        ROL REM3          ;
        ROL REM4          ;
        SUB REM1,BL       ;Try to subtract divisor fr
        SBC REM2,BH       ;
        SBC REM3,ZERO     ;
        SBC REM4,ZERO     ;
         BRCC SKIP        ;If the result was negative
        ADD REM1,BL       ;reverse the subtraction to
        ADC REM2,BH       ;
        ADC REM3,ZERO     ;
        ADC REM4,ZERO     ;
        CLC               ;Clear Carry Flag so zero s
         RJMP LOOP        ;Loop Back
SKIP:   SEC               ;Set Carry Flag to be shift
```

```
         RJMP LOOP
DONE:
```

11. SQUARE-ROOT OF A SINGLE-BYTE NUMBER

To compute the square of an eight-byte number we can take advantage of the fact that the sum of the odd numbers create square numbers:

```
    1         = 1^2 =  1
    1+3       = 2^2 =  4
    1+3+5     = 3^2 =  9
    1+3+5+7   = 4^2 = 16
    1+3+5+7+9 = 5^2 = 25
```

If we study the above table we notice that the square-root of the number equals the number of odd numbers we have summed. We simply create a routine that keeps track of the total of odd numbers that have been subtracted. The routine takes fifteen to one hunded cycles.

```
.DEF  ANS = R0             ;To hold answer
.DEF    A = R16            ;To hold the square
.DEF    B = R18            ;Sum, Work space

  LDI A,100                ;Load the square into A

SQRT:
LOOP:
  SUB A,B                  ;Subtract B from Square
   BRCS DONE               ;If bigger than sqaure we a
  INC ANS                  ;Increment the answer
  SUBI B,-2                ;Increment B by two
   RJMP LOOP
```

12. SQUARE-ROOT OF A 16-BIT NUMBER

We could expand the routine above to handle the square-root of a sixteen-bit number but the number of clock cycles can be as high as

3750. The routine below processes two bits at a time starting with the highest bits and also provides the remainder. It uses about 160 clock cycles.

```
 .DEF  ANSL = R0            ;Square Root (answer)
 .DEF  ANSH = R1            ;
 .DEF  REML = R2            ;Remainder
 .DEF  REMH = R3            ;
 .DEF    AL = R16           ;Square to take root (inpu
 .DEF    AH = R17           ;
 .DEF     C = R20           ;Loop Counter

        LDI   AL,LOW($FFFF)
        LDI   AH,HIGH($FFFF)
SQRT16:
        PUSH  AL            ;Save Square for later res
        PUSH  AH            ;
        CLR   REML          ;Initialize Remainder to z
        CLR   REMH          ;
        CLR   ANSL          ;Initialize Root to zero
        CLR   ANSH          ;
        LDI   C,8           ;Set Loop Counter to eight
LOOP:
        LSL   ANSL          ;Multiply Root by two
        ROL   ANSH          ;
        LSL   AL            ;Shift two high-bits of Sq
        ROL   AH            ;into Remainder
        ROL   REML          ;
        ROL   REMH          ;
        LSL   AL            ;Shift second high bit of
        ROL   AH            ;into Remainder
        ROL   REML          ;
        ROL   REMH          ;
        CP    ANSL,REML     ;Compare Root to Remainder
        CPC   ANSH,REMH     ;
         BRCC SKIP          ;If Remainder less or equa
        INC   ANSL          ;Increment Root
        SUB   REML,ANSL     ;Subtract Root from Remain
        SBC   REMH,ANSH     ;
        INC   ANSL          ;Increment Root
```

```
SKIP:
        DEC   C             ;Decrement Loop Counter
         BRNE LOOP          ;Check if all bits process
        LSR   ANSH          ;Divide Root by two
        ROR   ANSL          ;
        POP   AH            ;Restore Original Square
        POP   AL
   RJMP LOOP
```

13. SQUARE-ROOT OF A 32-BIT NUMBER

We can expand the previous routine to handle 32-bit numbers. It takes between 500 to 580 clock cycles to complete.

```
 .DEF  ANS1 = R0            ;Square Root (answer)
 .DEF  ANS2 = R1            ;
 .DEF  ANS3 = R2            ;
 .DEF  ANS4 = R3            ;
 .DEF  REM1 = R4            ;Remainder
 .DEF  REM2 = R5            ;
 .DEF  REM3 = R6            ;
 .DEF  REM4 = R7            ;
 .DEF    A1 = R16           ;Square (input)
 .DEF    A2 = R17           ;
 .DEF    A3 = R18           ;
 .DEF    A4 = R19           ;
 .DEF     C = R20           ;Loop Counter

        LDI   A1,  LOW($FFFFFFFF)
        LDI   A2,BYTE2($FFFFFFFF)
        LDI   A3,BYTE3($FFFFFFFF)
        LDI   A4,BYTE4($FFFFFFFF)
sqrt16:
        PUSH  A1            ;Save Square for later res
        PUSH  A2            ;
        PUSH  A3            ;
        PUSH  A4            ;
        CLR   REM1          ;Initialize Remainder to z
        CLR   REM2          ;
        CLR   REM3          ;
```

```
        CLR   REM4          ;
        CLR   ANS1          ;Initialize Root to zero
        CLR   ANS2          ;
        CLR   ANS3          ;
        CLR   ANS4          ;
        LDI   C,16          ;Set Loop Counter to sixte
LOOP:
        LSL   ANS1          ;Multiply Root by two
        ROL   ANS2          ;
        ROL   ANS3          ;
        ROL   ANS4          ;
        LSL   A1            ;Shift two high-bits of Sq
        ROL   A2            ;into Remainder
        ROL   A3            ;
        ROL   A4            ;
        ROL   REM1          ;
        ROL   REM2          ;
        ROL   REM3          ;
        ROL   REM3          ;
        LSL   A1            ;Shift second high bit of
        ROL   A2            ;into Remainder
        ROL   A3            ;
        ROL   A4            ;
        ROL   REM1          ;
        ROL   REM2          ;
        ROL   REM3          ;
        ROL   REM4          ;
        CP    ANS1,REM1     ;Compare Root to Remainder
        CPC   ANS2,REM2     ;
        CPC   ANS3,REM3     ;
        CPC   ANS4,REM4     ;
         BRCC SKIP          ;If Remainder less or equa
        INC   ANS1          ;Increment Root
        SUB   REM1,ANS1     ;Subtract Root from Remain
        SBC   REM2,ANS2     ;
        SBC   REM3,ANS3     ;
        SBC   REM4,ANS4     ;
        INC   ANS1          ;Increment Root
SKIP:
        DEC   C             ;Decrement Loop Counter
```

```
         BRNE LOOP          ;Check if all bits process
        LSR   ANS4          ;Divide Root by two
        ROR   ANS3          ;
        ROR   ANS2          ;
        ROR   ANS1          ;
        POP   A4            ;Restore Original Square
        POP   A3
        POP   A2
        POP   A1
```

14. CUBE ROOT OF A SINGLE-BYTE NUMBER

Since there are only seven possible cube roots of a single-byte number (0 to 6) this routine simply cycles through them until the correct root is found. The routine takes from 20 to 90 clock cycles.

```
.DEF   ANS = R0            ;Cube Root (answer)
.DEF   REM = R2            ;Remainder
.DEF     A = R16           ;Cube
.DEF     C = R17           ;Loop Counter

        LDI A,$FF          ;Load Cube into A
        CLR C              ;Start Loop Counter at Zero
LOOP:   MUL C,C            ;R0 = C^3
        MUL C,R0           ;
        CP  A,R0           ;Check if gone too far
         BRCS FINI         ;If so then Finish
        MOV REM,A          ;Calculate Remainder
        SUB REM,R0         ;
        INC C              ;Increment Loop Counter
        CPI C,7            ;Check if Done
         BRNE LOOP         ;Go back
FINI:   MOV ANS,C          ;Answer = Counter - 1
        DEC ANS
```

15. CUBE ROOT OF A 16-BIT NUMBER

The following routine finds the cube root by using two successive approximations, one high and one low and waits until they merge. The routine uses 150 to 250 clock cycles.

```
.DEF   ANS = R0              ;Answer (Cube Root)
.DEF  REML = R2              ;Remainder = A^3 - INT(C
.DEF  REMH = R3              ;
.DEF  CUB1 = R4              ;Answer from CUBE Routin
.DEF  CUB2 = R5              ;
.DEF  CUB3 = R6              ;
.DEF  TMP1 = R8              ;Temporary Workspace
.DEF  TMP2 = R9              ;
.DEF  ZERO = R10             ;To hold value Zero
.DEF   ONE = R11             ;To hold value One
.DEF   LES = R12             ;Low Estimate
.DEF   HES = R13             ;High Estimate
.DEF   AVG = R14             ;Average Low & High Esti

.DEF    AL = R16             ;Original Cube
.DEF    AH = R17             ;
.DEF    B  = R18             ;General Purpose

        LDI AL, LOW(1000)    ;Load Original Cube into
        LDI AH,HIGH(1000)    ;
        CLR REML             ;Clear Remainder
        CLR REMH             ;
        CLR ZERO             ;Set Zero
        CLR ONE              ;Set One
        INC ONE              ;
        CLR LES              ;Start Low Estimate at Z
        LDI B,42             ;Start High Estimate at
        MOV HES,B
LOOP:   MOV AVG,LES          ;AVG = (LowEst+HighEst+1
        ADD AVG,HES          ;
        ADC AVG,ONE          ;
        LSR AVG              ;
        MOV B,AVG            ;Calculate AVG^3
        RCALL CUBE           ;
        CP  CUB1,AL          ;Compare AVG^3 to Origin
        CPC CUB2,AH          ;
        CPC CUB3,ZERO        ;
         BRNE SKIP1          ;Check if AVG^3 = Origin
```

```
        MOV LES,AVG          ;AVG^3 = CUBE so Low-Est
         RJMP FINI2          ;
SKIP1:   BRCC ISHIGH         ;
        MOV LES,AVG          ;AVG^3 < CUBE so Low-Est
         RJMP SKIP2          ;
ISHIGH: MOV HES,AVG          ;AVG^3 > CUBE so High-Es
SKIP2:  MOV B,HES            ;B = HighEst - LowEst
        SUB B,LES            ;
         BREQ FINI           ;LowEst = HighEst so we
        CPI B,1              ;
         BRNE LOOP           ;If HighEst-LowEst > 1 t
FINI:
        MOV B,LES            ;Calculate Remainder
        RCALL CUBE           ;Remainder = Cube - LowE
        MOVW REMH:REML,AH:AL ;
        SUB REML,CUB1        ;
        SBC REMH,CUB2        ;
FINI2:  MOV ANS,LES          ;Store Result = LowEst
DONE:    RJMP DONE

;---------------------------;
; Calculates Cube of B      ;
; Results in CUB3:CUB2:CUB1  ;
;---------------------------;
CUBE:
        MUL B,B              ;Calc B*B
        MOVW TMP2:TMP1,R1:R0 ;
        MUL TMP1,B           ;Calc B*B*B
        MOVW CUB2:CUB1,R1:R0 ;
        MUL TMP2,B           ;
        ADD CUB2,R0          ;
        CLR CUB3             ;
        ADC CUB3,R1          ;
         RET
```

Comments

You do not have permission to add comments.