AVR ASM INTRODUCTION

Search this site

AVR ASSEMBLER TUTOR >

2c. LOGARITHMS

BASIC AVR MATH III v1.2

LOGARITHMS & e

by RetroDan@GMail.Com

CONTENTS:

The advantage of using integer math is the speed as versus the much slower speed and large size of floating-point math, however answers can be subject to larger rounding errors with integer math.

1. LOG BASE 2 OF A SINGLE-BYTE NUMBER

The integer value of logarithm base two of a number is useful in determining the number of bits required to store the number. The routine works by shifting the number left until a one falls out. The routine yields an error of about 1% and uses 10 to 40 clock cycles.

```
.DEF  ANSF = R0              ;Fractional Part of Answ
.DEF   ANS = R1              ;Integer Part of Answer
```

```
.DEF     A = R16              ;Original Value
.DEF     N = R20              ;Counter/Index

        LDI A,100             ;Load Original Value
        LDI  N,7              ;Initial log2 Value
        PUSH A                ;Store Original Value
LOOP:   LSL A                 ;Shift Left until a one
         BRCS FINI            ;
SKIP:   DEC N                 ;
         BRNE LOOP            ;
FINI:   MOV ANS,N             ;Store the Integer Part
        MOV ANSF,A            ;Store Fractional Part
        POP A                 ;Restore Original Value
```

2. LOG BASE TWO OF A SIXTEEN-BIT NUMBER

We can expand the previous routine to handle 16-bit numbers. The routine uses 20 to 100 clock cycles and should have an error of about 0.1%.

```
.DEF  ANSF = R0               ;Fractional Part of Answ
.DEF   ANS = R1               ;Integer Part of Answer
.DEF    AL = R16              ;Original Value
.DEF    AH = R17              ;
.DEF     N = R20              ;Counter/Index

        LDI AL,LOW(10000)     ;Load Original Value
        LDI AH,HIGH(10000)    ;
        LDI  N,15             ;Initial log2 Value
        PUSH AL               ;Store Original Value
        PUSH AH               ;
LOOP:   LSL AL                ;Shift Left until a one
        ROL AH                ;
         BRCS FINI            ;
SKIP:   DEC N                 ;
         BRNE LOOP            ;
FINI:   MOV ANS,N             ;Store the Integer Part
        MOV ANSF,AH           ;Store Fractional Part
        POP AH                ;Restore Original Value
```

```
        POP AL                ;
```

3. MULTIPLYING A SINGLE-BYTE NUMBER BY e

The value of e is 2.718281828. To estimate this with integer math we use the ratio 87/32 which is 2.71875 yielding an error of 0.017%. We first multiply our number by 87 then shift the result five times to the right for a division of 32. The routine is about 16 bytes long and takes about 28 clock cycles.

```
.DEF  ANSL = R0                ;Answer Low Byte
.DEF  ANSH = R1                ;Answer High Byte
.DEF     A = R16               ;Original Value
.DEF     B = R18               ;Workspace
.DEF     N = R20               ;Counter

        LDI A,100              ;Load Multiplier into A
        LDI B,87               ;Load Est of e = 87/32
        MUL A,B                ;
        LDI N,5                ;32 = 2^5
LOOP:   LSR ANSH               ;Divide result by 32
        ROR ANSL               ;
        DEC N                  ;
         BRNE LOOP             ;
```

4. MULTIPLYING A SIXTEEN-BIT NUMBER BY e

To estimate the value of e = 2.718281828 we use the ratio 5567/2048 = 2.71826 which yields an error of 0.001%. The routine is about 40 bytes long and takes about 24 clock cycles.

```
.DEF ANS1 = R0                 ;Two Byte Answer
.DEF ANS2 = R1                 ;
.DEF ZERO = R10                ;To hold Zero
.DEF   AL = R16                ;To hold multiplicand
.DEF   AH = R17                ;
.DEF   BL = R18                ;To hold multiplier
.DEF   BH = R19                ;
.DEF WRK1 = R20                ;Workspace
```

```
.DEF WRK2 = R21             ;
.DEF WRK3 = R22             ;
.DEF WRK4 = R23             ;

        LDI AL,LOW(1000)     ;Load Original Value int
        LDI AH,HIGH(1000)    ;
        LDI BL,LOW(5567)     ;Load multiplier into BH
        LDI BH,HIGH(5567)    ;
MUL16x16:
        CLR ZERO             ;Set Zero
        MUL AH,BH            ;Multiply AH:AL by 5567
        MOVW WRK4:WRK3,R1:R0 ;
        MUL AL,BL            ;
        MOVW WRK2:WRK1,R1:R0 ;
        MUL AH,BL            ;
        ADD WRK2,R0          ;
        ADC WRK3,R1          ;
        ADC WRK4,ZERO        ;
        MUL BH,AL            ;
        ADD WRK2,R0          ;
        ADC WRK3,R1          ;
        ADC WRK4,ZERO        ;
        LSR WRK4             ;Ignore lower two bytes
        ROR WRK3             ;by 1024 and a shift mak
        MOV ANS2,WRK4        ;Store Answer
        MOV ANS1,WRK3        ;By ignoring the lower t
                             ;get a division by 65536
```

5. MULTIPLYING A SINGLE-BYTE NUMBER BY 1/e

The number 1/e = 0.367879441 can be estimated with the ratio 94/256 which is 0.36719 with an error of 0.19%. The routine is about 6 bytes long and takes about 2 clock cycles.

```
.DEF  ANSL = R0             ;Fractional Low Byte (Ig
.DEF   ANS = R1             ;Answer
.DEF     A = R16            ;Original Value
.DEF     B = R18            ;Workspace
.DEF     N = R20            ;Counter
```

```
        LDI A,100           ;Original Value
        LDI B,94            ;Load Est of e = 94/256
        MUL A,B             ;Multiply Original Value
```

6. MULTIPLYING A SIXTEEN-BIT NUMBER BY 1/e

The constant 1/e = 0.367879441 can be estimated by the ratio 24109/65536 = 0.36787 with an error of 0.001%. The routine is about 36 bytes long and takes about 20 clock cycles.

```
.DEF ANS1 = R0                  ;Two Byte Answer
.DEF ANS2 = R1
.DEF ZERO = R10                 ;To hold Zero
.DEF   AL = R16                 ;Original Value
.DEF   AH = R17                 ;
.DEF   BL = R18                 ;To hold multiplier
.DEF   BH = R19                 ;
.DEF WRK1 = R20                 ;Workspace
.DEF WRK2 = R21                 ;
.DEF WRK3 = R22                 ;
.DEF WRK4 = R23                 ;

        LDI AL,LOW(1000)     ;Load Original Value int
        LDI AH,HIGH(1000)    ;
        LDI BL,LOW(24109)    ;Load multiplier into BH
        LDI BH,HIGH(24109)   ;

MUL16x16:
        CLR ZERO             ;Set Zero
        MUL AH,BH            ;Multiply Original Value
        MOVW WRK4:WRK3,R1:R0 ;
        MUL AL,BL            ;
        MOVW WRK2:WRK1,R1:R0 ;
        MUL AH,BL            ;
        ADD WRK2,R0          ;
        ADC WRK3,R1          ;
        ADC WRK4,ZERO        ;
        MUL BH,AL            ;
```

```
        ADD WRK2,R0           ;
        ADC WRK3,R1           ;
        ADC WRK4,ZERO         ;
        MOV ANS2,WRK4         ;Store Answer
        MOV ANS1,WRK3         ;By ignoring the lower t
                              ;get a division by 65536
```

7. NATURAL LOGARITHM OF A SINGLE-BYTE NUMBER

Previously we created routines to calculate the log base two of a number. From the equations below we see that by multiplying the log2 of a number by 0.6931472 we can calculate the natural logarithm.

```
        ln(x) = log2(x)/log2(e)
              = log2(x) * 1/1.442695
              = log2(x) * 0.6931472
              = C * Log2(x) ; where C = 0.6931472
```

We will use the following ratios to estimate the constant above.

```
            C =   177/256   = 0.69141 Error 0.24%
```

The routine is about 40 bytes long and uses about 50 to 80 clock cycles the error should be less than 1%.

```
.DEF  ANSF = R2               ;Fractional Part of Answ
.DEF   ANS = R3               ;Integer Part of Answer
.DEF     A = R16              ;Original Value
.DEF     N = R20              ;Counter/Index

        LDI A,250             ;Load Original Value
        LDI  N,7              ;Initial log2 Value
        PUSH A                ;Store Original Value
LOOP:   LSL A                 ;Shift Left until a one
         BRCS FINI            ;
SKIP:   DEC N                 ;
         BRNE LOOP            ;
FINI:   MOV ANS,N             ;Store the Integer Part
        MOV ANSF,A            ;Store Fractional Part
        LDI A,177             ;Multiply by 177 then
```

```
MUL16x8:MUL ANSF,A            ;divide by 256 by ignori
        MOV ANSF,R1           ;lowest byte of the resu
        MUL ANS,A             ;
        CLR ANS               ;
        ADD ANSF,R0           ;
        ADC ANS,R1            ;
        POP A                 ;Restore Original Value
```

8. NATURAL LOGARITHM OF A SIXTEEN-BIT NUMBER

Previously we created routines to calculate the log base two of a number. From the equations below we see that by multiplying the log2 of a number by 0.6931472 we can calculate the natural logarithm.

```
        ln(x) = log2(x)/log2(e)
              = log2(x) * 1/1.442695
              = log2(x) * 0.6931472
              = C * Log2(x) ; where C = 0.6931472
```

We will use the following ratio to estimate the constant above.

```
            C = 45426/65536 = 0.69315 Error = 0.0004%
```

The routine is about 70 bytes long and takes about 40 to 120 clock cycles.

```
.DEF  ANSF = R2               ;Fractional Part of Answ
.DEF   ANS = R3               ;Integer Part of Answer
.DEF  TMP1 = R4               ;Temporary Workspace
.DEF  TMP2 = R5               ;
.DEF  TMP3 = R6               ;
.DEF  TMP4 = R7               ;
.DEF  ZERO = R10              ;Hold Zero
.DEF    AL = R16              ;Original Value
.DEF    AH = R17              ;
.DEF     N = R20              ;Counter/Index

        LDI AL,LOW(65000)     ;Load Original Value
        LDI AH,HIGH(65000)    ;
        LDI  N,15             ;Initial log2 Value
```

```
        PUSH AL             ;Store Original Value
        PUSH AH             ;
LOOP:   LSL AL              ;Shift Left until a one
        ROL AH              ;
         BRCS FINI          ;
SKIP:   DEC N               ;
         BRNE LOOP          ;
FINI:   MOV ANS,N           ;Store the Integer Part
        MOV ANSF,AH         ;Store Fractional Part
        LDI AL,LOW(45426)   ;Multiply by 45426
        LDI AH,HIGH(45426)  ;Then divide by 65536 by
MUL16x16:                   ;the lowest two bytes
        CLR ZERO
        MUL ANS,AH          ;
        MOVW TMP4:TMP3,R1:R0 ;
        MUL ANSF,AL         ;
        MOVW TMP2:TMP1,R1:R0 ;
        MUL ANS,AL          ;
        ADD TMP2,R0         ;
        ADC TMP3,R1         ;
        ADC TMP4,ZERO       ;
        MUL ANSF,AH         ;
        ADD TMP2,R0         ;
        ADC TMP3,R1         ;
        ADC TMP4,ZERO       ;
        MOVW ANS:ANSF,TMP4:TMP3 ;Store Answer
        POP AH              ;Restore Original Value
        POP AL              ;
```

9. LOG BASE 10 OF A SINGLE-BYTE NUMBER

Previously we created routines to calculate the log base two of a number. From the equations below we see that by multiplying the log2 of a number by 0.301030004 we can calculate the logarithm base 10.

```
     log10(x) = log2(x)/log2(10)
              = log2(x) * 1/3.321928
              = log2(x) * 0.301030004
              = C * Log2(x) ; where C = 0.0.301030004
```

We will use the following ratio to estimate the constant above.

```
C = 77/256 = 0.30078 Error = 0.08%
```

The routine is about 40 bytes long and takes about 20 to 50 clock cycles.

```
.DEF  ANSF = R2              ;Fractional Part of Answ
.DEF   ANS = R3              ;Integer Part of Answer
.DEF     A = R16             ;Original Value
.DEF     N = R20             ;Counter/Index

        LDI A,250            ;Load Original Value
        LDI  N,7             ;Initial log2 Value
        PUSH A               ;Store Original Value
LOOP:   LSL A                ;Shift Left until a one
         BRCS FINI           ;
SKIP:   DEC N                ;
         BRNE LOOP           ;
FINI:   MOV ANS,N            ;Store the Integer Part
        MOV ANSF,A           ;Store Fractional Part
        LDI A,77             ;Multiply by 77 then
MUL16x8:MUL ANSF,A           ;divide by 256 by ignori
        MOV ANSF,R1          ;lowest byte of the resu
        MUL ANS,A            ;
        CLR ANS              ;
        ADD ANSF,R0          ;
        ADC ANS,R1           ;
        POP A                ;Restore Original Value
```

10. LOG BASE 10 OF A SIXTEEN-BIT NUMBER

Previously we created routines to calculate the log base two of a number. From the equations below we see that by multiplying the log2 of a number by 0.301030004 we can calculate the logarithm base 10.

```
log10(x) = log2(x)/log2(10)
         = log2(x) * 1/3.321928
         = log2(x) * 0.301030004
         = C * Log2(x) ; where C = 0.0.301030004
```

We will use the following ratio to estimate the constant above.

```
C = 19728/65536 = 0.30103 Error = 0.00000
```

The routine is about 70 bytes long and takes about 40 to 125 clock cycles.

```
.DEF  ANSF = R2              ;Fractional Part of Answ
.DEF   ANS = R3              ;Integer Part of Answer
.DEF  TMP1 = R4              ;Workspace
.DEF  TMP2 = R5              ;
.DEF  TMP3 = R6              ;
.DEF  TMP4 = R7              ;
.DEF  ZERO = R10             ;Hold Zero
.DEF    AL = R16             ;Original Value
.DEF    AH = R17             ;
.DEF     N = R20             ;Counter/Index

        LDI AL,LOW(65000)    ;Load Original Value
        LDI AH,HIGH(65000)   ;
        LDI  N,15            ;Initial log2 Value
        PUSH AL              ;Store Original Value
        PUSH AH              ;
LOOP:   LSL AL               ;Shift Left until a one
        ROL AH               ;
         BRCS FINI           ;
SKIP:   DEC N                ;
         BRNE LOOP           ;
FINI:   MOV ANS,N            ;Store the Integer Part
        MOV ANSF,AH          ;Store Fractional Part
        LDI AL,LOW(19728)    ;Multiply by 19728
        LDI AH,HIGH(19728)   ;Then divide by 65536 by
MUL16x16:                    ;the two lowest bytes
        CLR ZERO             ;
        MUL ANS,AH           ;
        MOVW TMP4:TMP3,R1:R0 ;
        MUL ANSF,AL          ;
        MOVW TMP2:TMP1,R1:R0 ;
        MUL ANS,AL           ;
        ADD TMP2,R0          ;
        ADC TMP3,R1          ;
        ADC TMP4,ZERO        ;
        MUL ANSF,AH          ;
```

```
ADD TMP2,R0          ;
ADC TMP3,R1          ;
ADC TMP4,ZERO        ;
MOVW ANS:ANSF,TMP4:TMP3 ;Store Answer
POP AH               ;Restore Original Value
POP AL               ;
```

Comments