Signed Math on the Z8 MCU

The Z8 Flag Register and Signed Math Instructions

©1999 ZiLOG, Inc.

Table of Contents

Introduction 3
The Application 4
Schematic 7
Sample Code 8
Appendix A 15
Appendix B 16
References 17
Information Integrity 18
Document Disclaimer 18

Acknowledgments

Project Lead Engineer
Steve Narum

System and Code Development:
Steve Narum

 ©1999 ZiLOG, Inc.

Introduction

The Z8 MCU's architecture provides ease-of-use for signed math functions. The CPU's flag register and signed mathematical instructions allow a programmer to work easily with signed values of nearly any length.

In other architectures, there is little or no provision for signed math, programmers must work around this limitation. For example, numbers can be manipulated as unsigned values, with the actual sign stored as a software flag. The flow of the program is then modified according to the sign. This technique adds significant overhead in both execution speed and program memory size. Often the application requires a more expensive processor that has more ROM or hardware math-execution units.

The ability to recognize signed data makes the Z8 MCU valuable for many functions. In a positioning motor control application, the difference between the current and requested positions requires both a magnitude and a sign to indicate the direction the motor should be turning.

©1999 ZiLOG, Inc.

The Application

In this application note, a simple motor-control algorithm is demonstrated using one of the lowest-cost ZiLOG chips—the Z86E02. This small processor has a bare minimum of OTP code memory and only the basic set of peripherals. See the Application Note titled A DC Motor Controller Using the ZiLOG Z86E06 MCU, for a discussion of motor control basics.

The motor control application uses a PWM output to drive the winding of a motor either clockwise or counter-clockwise. The load's absolute position is encoded by a rheostat or similar device that provides a DC voltage output proportional to the position. This voltage is read by an external analog-to-digital converter, and the Z8 uses a microwire interface to read an 8-bit position value. An external microwire input allows the external system to request an exact position for the motor. There are three limit switches that stop the motor if any are closed.

For each cycle of the software, the present position of the motor is read from the ADC and a positional error value is calculated. This error value requires at least nine bits to store the signed value. The requested position and the actual position could be on either end of the range, and require a magnitude of eight bits. The error could be in either direction from the requested position (either positive or negative), and requires one sign bit. This situation causes problems for many processors that cannot provide a simple way of executing signed math. Some processors cannot execute signed math on more than eight bits (7 magnitude bits, plus 1 sign bit). Others do not support signed math at all.

The Z8 processor allows a designer to store the error as a 16-bit signed number and apply mathematical functions to it with ease. Addition and subtraction are provided in the ADD and SUB instructions and are extended to multiple byte values by the ADC and SBC instructions. The sign bits are automatically handled by the Arithmetic Logic Unit (ALU). An example from the software is shown below.

```
CLR    ERR_HI
LD     ERR_LO,POSR
SUB    ERR_LO,POSA
SBC    ERR_HI,#%00
```

Two 8-bit unsigned numbers, one from the ADC and the other input externally, are subtracted into a 16-bit signed result. Nothing more is required. The Z8's ALU automatically sets the sign flag after the subtraction to the sign of the result. If necessary, a jump statement can follow immediately to perform a different task, depending on the direction of the error. No compare or test instruction is necessary.

Addition works equally well. Both addition and subtraction can be extended to more bytes by adding more ADC or SBC instructions, one per byte.

Multiplication by successive addition is also a common function in software that deals with positioning controls. The basic technique of multiplying two 8-bit unsigned values into an unsigned 16-bit result is shown in Appendix A. The code segment below demonstrates how this idea can be extended to multiplication of a signed 16-bit number (`ERR_HI`, `ERR_LO`) by an unsigned 8-bit scalar `RES_LO` into a signed 24-bit result (`RES_HI`, `RES_MD`, `RES_LO`).

©1999 ZiLOG, Inc.

```
MULT_8x16s:    LD     PULSE_CNT,#%08
               CLR    RES_HI
               CLR    RES_MD
               RRC    RES_LO
MULT_LOOP:     JR     NC,MULT_SHIFT
               ADD    RES_MD,ERR_LO
               ADC    RES_HI,ERR_HI
MULT_SHIFT:    SRA    RES_HI
               RRC    RES_MD
               RRC    RES_LO
               DJNZ   PULSE_CNT,MULT_LOOP
               RET
```

After exiting this multiplication loop, the flags reflect the resulting value's sign. Subsequently, decisions can be made without requiring a redundant test.

The Z8 performs signed division by two equally well, and requires only two bytes for the SRA instruction. By chaining the SRA into RRC instructions on the lower bytes, longer values can be divided easily. Following is an example from the code:

```
           LD     PULSE_CNT,#%04
DIV_LOOP:  SRA    RES_HI
           RRC    RES_MD
           RRC    RES_LO
           DJNZ   PULSE_CNT,DIV_LOOP
```

This code segment demonstrates a 24-bit signed value being divided by 16. When the ALU sets the flags on the last RRC, the DJNZ instruction does not modify the flags. This condition allows the programmer to use the flags in a jump instruction without testing first. Care must be taken, however, since the flags are the result of the most recent operation. The Z flag, for example, would indicate that the lowest byte is 0 after the division; it does not indicate that the entire 24-bit value is 0.

Also, if rounding is necessary, the carry flag can be treated as a half bit. The last RRC instruction shifts a bit into the carry flag only if the prior value was odd. A simple construct automatically rounds the result to the nearest integer value, as in the following example:

```
           .
           .
           .
           RRC RES_LO
           DJNZ   PULSE-CNT, DIV_LOOP
ROUND:     ADC    RES_LO,#0
           ADC    RES_MD,#0
           ADC    RES_HI,#0
```

Performing division on signed numbers by a number that is not a multiple of two is not as straightforward. Unfortunately, the task is complex enough that it becomes simpler to change the sign, perform an unsigned divide, then restore the sign. Appendix B describes a 16 x 8 divider that can be extended to longer values if necessary.

Figure 1 is a schematic of the Z86E02 microwire-controlled motor controller. Following Figure 1 is a signed math demonstration code for the Z86E02 operating at 8 MHz.

Schematic

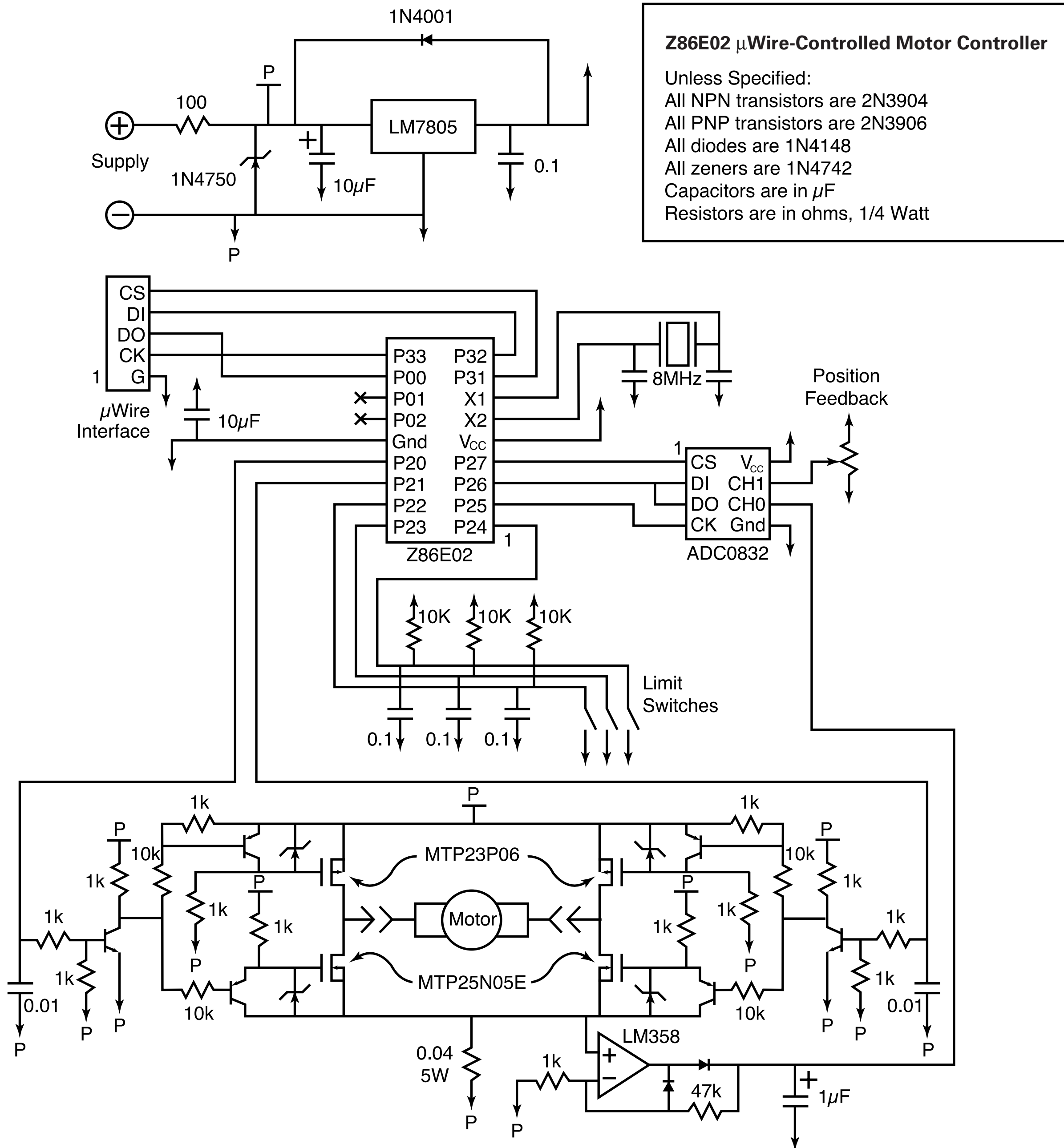

Figure 1. Z86E02 µWire-Based Motor Controller

©1999 ZiLOG, Inc.

Sample Code

```
;
; Signed math demonstration code on the Z86E02 @ 8MHz
;


.INCLUDE     MACROS.H       ; Include some useful macros

; Macros include: WDT, SETBIT, CLRBIT, JNZ

MATH_GROUP    .EQU     %10
BIT_CNT       .EQU     R0     ; uWire bit counter
PULSE_CNT     .EQU     R1     ; PWM pulse & math counter
ERR_HI        .EQU     R2     ; Error high byte
ERR_LO        .EQU     R3     ; Error low byte
ERR_WORD      .EQU     RR2    ; Error as a word
OERR_HI       .EQU     R4     ; Old error high byte
OERR_LO       .EQU     R5     ; Old error low byte
RES_HIW       .EQU     RR6    ; Math result, high word
RES_HI        .EQU     R6     ; Math result, high byte
RES_MD        .EQU     R7     ; Math result, mid byte
RES_LO        .EQU     R8     ; Math result, low byte
ACCUM         .EQU     R9     ; Error accumulator
POSR          .EQU     R10    ; Requested position
POSA          .EQU     R11    ; Actual position
HI_TIME       .EQU     R12    ; PWM high time
LO_TIME       .EQU     R13    ; PWM low time
; (spare)     .EQU     R14
STATUS        .EQU     R15    ; Status flags register

; Bit masks for STATUS register

MOVING        .EQU     00100000B ; Moving/stopped flag
PWM           .EQU     01000000B ; PWM high/low state flag

; Port pin bit masks

WINDINGS      .EQU     00000011B ; Motor winding outputs (P20,1)
SWITCHES      .EQU     00011100B ; Limit switch inputs (P22,3,4)
AD_CLK        .EQU     00100000B ; ADC clock (P25)
AD_DATA       .EQU     01000000B ; ADC data (P26)
AD_CS         .EQU     10000000B ; ADC chip select (P27)

MWXMIT        .EQU     00000001B ; uWire transmit (P00)
MWCS          .EQU     00000010B ; uWire chip select (P31)
MWREC         .EQU     00000100B ; uWire receive (P32)
MWCLK         .EQU     00001000B ; uWire clock (P33)
```

```
; P2M bit masks

NOMOVE       .EQU     00011111B ; Disable both directions
MOVECW       .EQU     00011110B ; Enable CW winding
MOVECCW      .EQU     00011101B ; Enable CCW winding
DATAIN       .EQU     01011111B ; Recv data on AD_DATA
DATAOUT      .EQU     00011111B ; Send data on AD_DATA

; Constants

KP           .EQU     %30   ; Proportional gain constant
KI           .EQU     %02   ; Integral gain constant
KD           .EQU     %20   ; Differential gain constant


; Macro definitions

READ_POS     .MACRO         ; Get position from ADC
                            ; Macro READ_POS uses the uWire interface to read
an
                            ; position value from
                            ; an external ADC.  The returned value is an 8 bit
                            ; unsigned, POSA.
    LD       BIT_CNT  #%04  ; 3 header bits + start ADC
    CLRBIT   P2,AD_CS       ; Enable uWire ADC
    SETBIT   P2,AD_DATA     ; Send 3 1's (start, read, ch1)

STARTBITS:   DJNZ     BIT_CNT  SENDBIT     ; Check bit counter
             LD       P2M      #DATAIN     ; Make P26 input on last
SENDBIT:     INC      BIT_CNT              ; Fix bit counter
             SETBIT   P2,AD_CLK            ; Raise clock
             NOP                           ; Wait
             CLRBIT   P2,AD_CLK            ; Drop clock
             DJNZ     BIT_CNT  STARTBITS   ; Next bit

             LD       BIT_CNT  #%08        ; Shift in 8 bits (MSB first)
GETBIT:      RCF                           ; Assume zero
             SETBIT   P2,AD_CLK            ; Raise clock
             TM       P2       #AD_DATA    ; Test data input
             JR       Z        RECV_ZERO
             SCF                           ; Nope, it's a one
RECV_ZERO:   CLRBIT   P2,AD_CLK            ; Drop clock
             RLC      POSA                 ; Rotate bit into place
             DJNZ     BIT_CNT  GETBIT      ; Next bit

             SETBIT   P2,AD_CS             ; Deselect the ADC
             LD       P2M      #DATAOUT    ; P26 back to output
             .ENDM
```

```
; Calculation macros

C_ERROR         .MACRO    ; Calculate positional error
                CLR       ERR_HI                ; Stuff 8 bit unsigned into
                LD        ERR_LO   POSR         ; 16 bit signed
                SUB       ERR_LO   POSA         ; Find difference (POSR - POSA)
                SBC       ERR_HI   #%00
                .ENDM

C_PROP          .MACRO                          ; Calc proportional part
                LD        RES_LO   #KP          ; Put gain in multiplier
                CALL      MULT_8x16s            ; ERR is the multiplicand
                CALL      DIV_LIMIT             ; Divide by 16 and limit to +/-100
                LD        HI_TIME  RES_MD       ; Stuff signed 24 bits
                LD        LO_TIME  RES_LO       ; into signed 16
                .ENDM

C_INT           .MACRO                          ; Calc integral part
                OR        ERR_LO   ERR_LO       ; (Possible values are FF02h to
00FFh)
                JR        Z        ADDINT       ; Don't change ACCUM if ERR = 0
                OR        ERR_HI   ERR_HI       ; Is ERR + or -?
                JR        MI       MNS_1
PLS_1:          CP        ACCUM    #100         ; ACCUM >= 100?
                JR        GE       ADDINT
                ADD       ACCUM    #KI          ; No, ACCUM += KI
                JR        ADDINT
MNS_1:          CP        ACCUM    #-100        ; ACCUM <= -100?
                JR        LE       ADDINT
                SUB       ACCUM    #KI          ; No, ACCUM -= KI
ADDINT:         ADD       LO_TIME  ACCUM        ; Add ACCUM into the sum
                ADC       HI_TIME  #%00
                .ENDM

C_DIFF          .MACRO    ; Calc differential part
                LD        RES_HI   ERR_HI       ; Save ERR
                LD        RES_LO   ERR_LO
                SUB       ERR_LO   OERR_LO      ; Subtract (ERR - OERR)
                SBC       ERR_HI   OERR_HI
                LD        OERR_LO  RES_LO       ; Update OERR
                LD        OERR_HI  RES_HI
                LD        RES_LO   #KD          ; Load multiplier
                CALL      MULT_8x16s            ; Take deltaERR * KD
                CALL      DIV_LIMIT             ; Divide by 16 and limit to +/-100
                ADD       RES_LO   LO_TIME      ; And add the sum into it
                ADC       RES_MD   HI_TIME      ; (RES used in SET_PWM)
                .ENDM
```

```
SET_PWM       .MACRO                            ; Set up PWM and enable windings
              CALL     LIMIT_100                ; Limit sum to +/-100
              JR       PL       POS_SUM         ; RES + or - ?
NEG_SUM:      COM      RES_LO                   ; Change sign
              INC      RES_LO
              LD       P2M      #MOVECCW        ; Enable CCW winding
              JR       CALC_PWM
POS_SUM:      LD       P2M      #MOVECW         ; Enable CW winding
CALC_PWM:     LD       HI_TIME  RES_LO          ; PWM high time = RES
              LD       LO_TIME  #100
              SUB      LO_TIME  RES_LO          ; PWM low time = (100 - RES)
              JR       NZ       CHECK_HI
              INC      LO_TIME                  ; If zero, add one
CHECK_HI:     INC      HI_TIME
              DJNZ     HI_TIME  PWM_DONE        ; Zero?
              INC      HI_TIME                  ; If zero, add one
PWM_DONE:
              .ENDM


; Interrupt vector table

              .ORG     %00

              JR       Init                     ; Reset the part
; Sacrifice IRQ0 to cause a reset if PC wraps to 0000h
;             .WORD    Init                     ; IRQ0 (P32f)
              .WORD    Init                     ; IRQ1 (P33f)
              .WORD    MICROWIRE                ; IRQ2 (P31f)
              .WORD    Init                     ; IRQ3 (P31r)
              .WORD    Init                     ; IRQ4 (T0)
              .WORD    T1_SERVICE               ; IRQ5 (T1)

              .ORG     %0C

Init:         WDT
              LD       SPL      #%40            ; Stack at end of C02's RAM
              LD       SPH      #%3F            ; Use SPH as pointer to clear RAM
              SRP      #%F0
CLR_REGS:     CLR      @R14                     ; Clear a register
              DJNZ     R14      CLR_REGS

              LD       P01M     #%04            ; P0 is an output
              LD       P2M      #NOMOVE         ; P20-4 = in, P25-7 = out
              LD       P3M      #%01            ; P3 = digital, P2 = push-pull

              LD       PRE1     #%06            ; T1 int clk, single shot, div by 1
              CLR      TMR                      ; Disable both timers
```

```
             LD        IPR       #%01         ; IRQ5 > IRQ2
             CLR       IMR
             EI
             OR        IMR       #00100100B   ; T1 and P31 only

             SRP       #MATH_GROUP

MAIN:        WDT
             TCM       P2        #SWITCHES    ; Limit switch closed?
             JR        NZ        MAIN         ; Wait until it's opened

STOPPED:     OR        POSR      POSR         ; Was stop requested?
             JR        Z         MAIN         ; Wait for new request

MOVIN:       TM        STATUS    #MOVING      ; Already moving?
             JR        NZ        MAIN         ; Wait until done

CALC_MOVE:   DI                               ; No uWire IRQ during READ_POS
             READ_POS                         ; Get current position from ADC
             C_ERROR                          ; Calculate positional error
             EI                               ; Recv OK while doing the math
             C_PROP                           ; Calc proportional part
             C_INT                            ; Calc integral part
             C_DIFF                           ; Calc differential part
             SET_PWM                          ; Set up PWM and enable windings

MOVE:        SETBIT    STATUS,MOVING          ; Flag that we're moving
             LD        PULSE_CNT #100         ; (dec)  100 pulses per move
             OR        IRQ       #%10         ; Force IRQ4
             JP        MAIN


; Math subroutines

MULT_8x16s:                                   ; Multiply 8 bit unsigned by 16 bit
                                              ; signed.
                                              ; 24 bit signed result.  8 bit
                                              ; destroyed.
             LD        PULSE_CNT #%08         ; Loop 8 times
             CLR       RES_HI                 ; Start with clear sum
             CLR       RES_MD
             RRC       RES_LO                 ; Rotate 1st bit into C, save C
MULT_LOOP:   JR        NC        MULT_SHIFT
             ADD       RES_MD    ERR_LO       ; Add in ERR * 256
             ADC       RES_HI    ERR_HI
MULT_SHIFT:  SRA       RES_HI                 ; Divide result by 2
             RRC       RES_MD
             RRC       RES_LO
             DJNZ      PULSE_CNT MULT_LOOP
```

```
            RET


DIV_LIMIT:                                  ; Divide signed 24 bit by 16 and
                                            ; limit result to +/- 100.
                                            ; Returns a 16 bit signed with
RES_HI
                                            ; containing garbage.
            LD      PULSE_CNT #%04          ; Four shifts = div by 16
DIV_LOOP:   SRA     RES_HI                  ; Divide by 2
            RRC     RES_MD
            RRC     RES_LO
            DJNZ    PULSE_CNT DIV_LOOP
LIMIT_100:  INCW    RES_HIW                 ; See if upper word is FFh
            JR      Z       RES_NEG
            DECW    RES_HIW                 ; See if upper word was 00h
            JR      Z       RES_POS
LIMIT_LO:   LD      RES_LO  #100            ; Limit to 100
            CLR     RES_MD
            OR      RES_HI  RES_HI          ; Should it be + or - ?
            JR      PL      LIMIT_DONE
            COM     RES_MD
            COM     RES_LO                  ; Fix the sign
            INC     RES_LO
LIMIT_DONE: RET
RES_NEG:    DECW    RES_HIW                 ; Restore sign
            OR      RES_LO  RES_LO          ; See if LO byte is negative
            JR      PL      LIMIT_LO        ; If not, actual value is too nega-
tive
            CP      RES_LO  #-100           ; If so, test magnitude
            JR      LT      LIMIT_LO
            JR      LIMIT_EXIT              ; It's OK, exit
RES_POS:    CP      RES_LO  #100            ; Can use LO as unsigned, check mag
            JR      UGT     LIMIT_LO        ; It's OK, exit
LIMIT_EXIT: OR      RES_HI  RES_HI          ; Fix the sign flag.  CP goofs it
up.
            RET


; Interrupt service routines

T1_SERVICE: ; Software PWM timer.
            TM      STATUS  #PWM            ; Which half are we on?
            JR      Z       LO_PULSE
HI_PULSE:   SETBIT  P2,WINDINGS             ; Make windings high
            SETBIT  STATUS,PWM              ; Flag we're doing high
            LD      T1      HI_TIME         ; Load timer
            AND     TMR     #%0C            ; And start
            IRET
```

```
LO_PULSE:     CLRBIT   P2,WINDINGS          ; Make windings low
              CLRBIT   STATUS,PWM           ; Flag we're doing low
              DJNZ     PULSE_CNT FIRE_LOW   ; Done?
END_MOVE:     LD       P2M       #NOMOVE    ; Stop motor
              CLRBIT   STATUS,MOVING        ; Flag stopped
              IRET
FIRE_LOW:     LD       T1        LO_TIME    ; Load timer
              AND      TMR       #%0C       ; And start
              IRET


MICROWIRE:                                  ; Read microwire input.  WDT
catches
                                            ; timeout
              WDT                           ; Start with a fresh WDT count
              LD       BIT_CNT   #%08       ; Get eight bits
WAIT_DN:      TM       P3        #MWCLK     ; Wait for low uWire clock
              JR       NZ        WAIT_DN
              SETBIT   P0,MWXMIT            ; Assert ready
WAIT_UP:      TM       P3        #MWCLK     ; Wait for high uWire clock
              JR       Z         WAIT_UP
              RCF                           ; Assume zero
              TM       P3        #MWREC     ; Get data
              JR       Z         NOT_ONE
              SCF                           ; It's a one
NOT_ONE:      RLC      POSR                 ; Rotate it into position
              DJNZ     BIT_CNT   WAIT_DN    ; Last bit?
              CLRBIT   P0,MWXMIT            ; Deassert uWire ready
              INC      POSR
              DJNZ     POSR      NOT_ZERO   ; See if POSR = 0
              LD       PULSE_CNT #%01       ; If so, stop the motor next T0
NOT_ZERO:     IRET
```

 ©1999 ZiLOG, Inc.

Appendix A

The following module illustrates an efficient algorithm for the multiplication of two unsigned 8-bit values, resulting in a 16-bit product. The algorithm repetitively shifts the multiplicand right (using RRC), the result being a shift of the low-order bit into the carry flag. If a 1 is shifted out, the multiplier is added to the high-order byte of the partial product. As the high-order bits of the multiplicand are vacated by the shift, the resulting partial-product bits are rotated in. Thus, the multiplicand and the low byte of the product occupy the same byte. As a result, there is a savings of register space, code, and execution time.

```
; ARITH MODULE
; CONSTANT
; MULTIPLIER = R1
; PRODUCT_LO = R3
; PRODUCT_HI = R2
; COUNT = R0
; GLOBAL
; MULTIPLICATION PROCEDURE
; *******************************
; Purpose = To perform an 8-bit by 8-bit unsigned binary
;           multiplication.
;
; Input =   R1 =   multiplier
;           R3 =   multiplicand
;
; Output =  RR2 =  product
;           R0 =   destroyed
; *******************************
ENTRY:
      ld        COUNT,#09H              ;8 bits+1
      clr       PRODUCT_HI              ;Init High result byte
      rcf                               ;CARRY = 0
LOOP: rrc       PRODUCT_HI
      rrc       PRODUCT_LO
      jr        NC,NEXT
      add       PRODUCT_HI,MULTIPLIER
NEXT: djnz      COUNT,LOOP
      ret
```

Appendix B

The following module illustrates an efficient algorithm for the division of a 16-bit unsigned value by an 8-bit unsigned value, resulting in an 8-bit unsigned quotient. The algorithm repetitively shifts the dividend left (using RLC). If the high-order bit shifted out is a 1, or if the resulting high-order dividend byte is greater than or equal to the divisor, the divisor is subtracted from the high byte of the dividend. As the low-order bits of the dividend are vacated by the shift left, the resulting partial-quotient bits are rotated in. Thus, the quotient and the low byte of the dividend occupy the same byte. As a result, there is a savings of register space, code, and execution time.

```
; ARITH MODULE
; CONSTANT
; COUNT = R0
; DIVISOR = R1
; DIVIDEND_HI = R2
; DIVIDEND_LO = R3
; GLOBAL
; DIVIDE PROCEDURE
; *******************************
; Purpose = To perform a 16-bit by 8-bit unsigned binary division.
;
; Input =   R1 = 8-bit divisor
;           RR2 = 16-bit dividend
;
; Output =  R3 = 8-bit quotient
;           R2 = 8-bit remainder
;           Carry flag = 1 if overflow
;                      = 0 if no overflow
*******************************
ENTRY:
      ld        COUNT,#08H            ;Loop counter
                                      ;Check if result fits in
                                      ;8 bits
      cp        DIVISOR,DIVIDEND_HI
      jr        UGT,LOOP              ;CARRY = 0 FOR RLC)
                                      ;Won't fit. Overflow
      scf                             ;CARRY = 1
      ret
LOOP:                                 ;Result fits. Go ahead
                                      ;with division
      rlc       DIVIDEND_LO           ;DIVIDEND * 2
      rlc       DIVIDEND_HI
      jr        c,SUBT
      cp        DIVISOR,DIVIDEND_HI
      jr        UGT,NEXT              ;CARRY = 0
SUBT: sub       DIVIDENT_HI,DIVISOR
      scf                             ;To be shifted into result
NEXT:           djnz  COUNT,LOOP      ;No flags affected
      ret
```

References

1. Steven Frank, *Intelligent Remote Positioner (Motor Control)*, Microchip Technology, Inc. AN531, DS00531C, 1994.
2. ZiLOG, *ZiLOG Z8 Microcontroller User's Manual*, UM95Z800103, 1995.
3. ZiLOG, *The Z8 Application Note Handbook*, DB96Z8X0100, 1996.
4. ZiLOG, *A DC Motor Controller Using the ZiLOG Z86E06 MCU*, AP96DZ80500, 1996.

 ©1999 ZiLOG, Inc.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

© 1999 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES. OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses are conveyed, implicitly or otherwise, by this document under any intellectual property rights.

ZiLOG, Inc.
910 East Hamilton Avenue, Suite 110
Campbell, CA 95008
Telephone (408) 558-8500
FAX (408) 558-8300
Internet: http://www.zilog.com