Technical Note

Divide Routines

TN000601-0603

General Overview

Many microcontroller applications require arithmetic functions beyond the instructions directly supported by the device. The most common among these applications are simple multiply and divide routines for scaling and filtering purposes. The Z8 register file architecture provides a significant advantage for implementing efficient routines for these functions. This Technical Note shows efficient implementations of 16-bit by 8-bit and 16-bit by 16-bit unsigned divide routines. A companion Technical Note, Multiply Routines (TN0007), describes 8-bit by 8-bit and 16-bit by 16-bit multiply routines.

Discussion

The first software module, **divide_16_8** (see Divide 16 by 8 on page 2), illustrates an efficient algorithm for the division of a 16-bit unsigned value by an 8-bit unsigned value. This computation results in an 8-bit unsigned quotient and an 8-bit remainder. The second software module, **divide_16_16** (see Divide 16 by 16 on page 3), divides a 16-bit dividend by a 16-bit divisor. The result is a 16-bit quotient and a 16-bit remainder. Both routines use a similar algorithm.

In the **divide_16_8** routine, the dividend is repetitively shifted left (via the Rotate Left through Carry instruction). 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. This routine uses 33 bytes of code and 4 registers.

In the **divide_16_16** routine, both the 16-bit dividend and 16-bit remainder are shifted left as a single 32-bit quantity. If the high order bit shifted out is a 1, or if the resulting 16-bit remainder is less than or equal to the 16-bit divisor, the divisor is subtracted from the remainder. As the low order bits of the dividend are vacated by the shift left, the resulting partial quotient bits are rotated in.

This method allows the quotient and the low byte of the dividend to occupy the same byte, which saves register space, code, and execution time.

For the 16-by-8 divide routine, a check is performed before the division takes place to make sure the result fits into a single byte. A quick way to ensure this fit is by checking whether the divisor is greater than the MSB of the dividend. If the divisor is not greater, then the carry flag is set to indicate an error and the routine is exited. Otherwise, when the routine is completed, the carry flag is cleared. This routine uses 45 bytes of code and 7 registers.

The routines are implemented in a modular fashion such that it may be easily integrated into a target application. The registers used in the routines can be located in any register group by defining the value for the *math_group* equate.

The execution time of these divide routines depends on the values being used in the dividend, the divisor, and the operating frequency of the MCU. In the following 16-bit by 8-bit divide example, a Z8Plus device operating at a frequency of 10MHz takes 64µs to complete the divide function (including entry and exit overhead).

$$\frac{1234h}{56h} = 36h, \text{ Remainder } 10h$$

In the following 16-bit by 16-bit divide example, a Z8Plus device operating at a frequency of 10MHz takes 136µs to complete the divide function.

$$\frac{ABCDh}{1234h} = 0009h, \text{ Remainder } 07F9h$$

Sample Code

Divide 16 by 8

The following code illustrates the **divide_16_8** software module.

```
;*******************************************************************
; Divides a 16-bit value (dividend_hi, dividend_lo) by an 8-bit
; value (divisor).
; Result is returned in dividend_lo. Remainder is returned in dividend_hi.
; Carry flag is set on error.
;*******************************************************************
math_group  equ       00h       ;Defines the WRG for this routine

  segment   data
  align     16
div_len     ds        1
divisor     ds        1
dividend_hi ds        1         ;Dividend MSB and Remainder
dividend_lo ds        1         ;Dividend LSB and Quotient
;-------------------------------------------------------------------------
  segment   code

divide_16_8:
  push      rp                  ;Save the current register pointer
  srp       #math_group         ;and use our own working register group
  ld        .r(div_len),#8      ;divisor is 8 bits

  cp        .r(divisor),.r(dividend_hi);Check if result will fit in 8 bits
```

```
  jr        ugt,$loop8          ;Divisor must be greater than MSB of dividend
  scf                           ;Won't fit, set carry flag and return
  jr        $exit_divide_16_8
$loop8:
  rlc       .r(dividend_lo)     ;dividend (16 bit) * 2
  rlc       .r(dividend_hi)
  jr        c,$subtract8        ;Look for the carry out of the 16th bit
  cp        .r(divisor),.r(dividend_hi);or if divisor is greater than MSB of
dividend
  jr        ugt,$next8
$subtract8:
  sub       .r(dividend_hi),.r(divisor)
  scf                           ;Gets shifted into the result
$next8:
  djnz      .r(div_len),$loop8  ;No flags affected
  rlc       .r(dividend_lo)     ;Final shift out also clears carry flag
$exit_divide_16_8:
  pop       rp                  ;Restore register pointer
  ret
```

Divide 16 by 16

The following code illustrates the **divide_16_16** software module.

```
;*****************************************************************************
* Divides a 16-bit value (dividend_hi, dividend_lo) by another 16-bit
* value (divisor_hi, divisor_lo).
* Result is returned in dividend_hi and dividend_lo. The remainder is
* returned in remainder_hi and remainder_lo.
* Carry flag is set on error.
;*****************************************************************************
math_group  equ       00h       ;Defines the WRG for this routine

  segment   data
  align     16
divisor_hi  ds        1
divisor_lo  ds        1
dividend_hi ds        1         ;Dividend and Quotient
dividend_lo ds        1         ;Dividend and Quotient
remainder_hids        1
remainder_lods        1
div_len     ds        1

;-----------------------------------------------------------------------------
segmentcode

divide_16_16:
  push      rp                  ;Save the current register pointer
  srp       #math_group         ;and use our own working register group
```

```
  ld        .r(div_len),#16     ;divisor is 16 bits
  clr       .r(remainder_hi)    ;Remainder starts at 0
  clr       .r(remainder_lo)
  rcf
$loop16:
  rlc       .r(dividend_lo)
  rlc       .r(dividend_hi)
  rlc       .r(remainder_lo)
  rlc       .r(remainder_hi)
  jr        c,$subtract16       ;Look for the carry out of the 16th bit
  cp        .r(divisor_hi),.r(remainder_hi);or if divisor is greater than MSB
of dividend
  jr        ugt,$next16
  jr        ult,$subtract16
  cp        .r(divisor_lo),.r(remainder_lo)
  jr        ugt,$next16
$subtract16:
  sub       .r(remainder_lo),.r(divisor_lo);16-bit subtract
  sbc       .r(remainder_hi),.r(divisor_hi)
  scf                           ;Gets shifted into the result
$next16:
  djnz      .r(div_len),$loop16 ;No flags affected
  rlc       .r(dividend_lo)
  rlc       .r(dividend_hi)     ;Final shift out also clears carry flag
$exit_divide_16_16:
  pop       rp                  ;Restore register pointer
ret
```

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

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