4.4 Unsigned Divide Operation

The PIC18 MCU does not provide any divide instruction. Therefore, a divide operation must be synthesized by other instructions. A simple but popular divide algorithm in use today is the *repeated subtraction method.* This method performs unsigned divide operation. The hardware required for implementing the repeated subtraction method is shown in Figure 4.1.

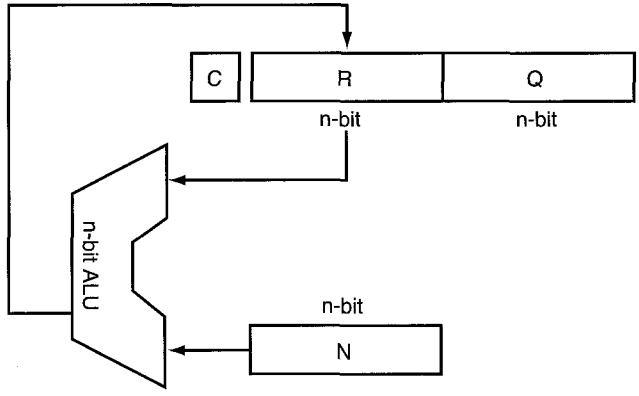

Figure 4.1 ■ Division hardware

Before performing the repeated subtraction operation, one needs to load 0, the dividend, and the divisor into registers R, Q, and N, respectively. The carry flag is used to indicate whether the subtraction result is negative. The ALU can perform n-bit unsigned addition and subtraction operations. The repeated subtraction method consists of n steps. Each division step consists of three parts:

Step 1
Shift the register pair (R, Q) one place to the left.

Step 2
Subtract the contents of N from R and put the result back to R if the result is positive.

Step 3
If the result of Step 2 is negative, then set the least significant bit of Q to 0. Otherwise, set the least significant bit of Q to 1.

Example 4.3

Write a program to divide an 8-bit number into another 8-bit number.

Solution: The following program is a direct translation of the paper-and-pencil division algorithm:

```
            #include <p18F8720.inc>
lp_cnt      set         0x00
rem         set         0x01        ; register to hold remainder
quo         set         0x02        ; register to hold quotient
dsr         set         0x03        ; register to hold divisor
dvd         set         0x04        ; register to hold dividend
dd          equ         0xf5        ; value used as dividend
```

```
dr          equ         0x11            ; value used as divisor
            org         0x00
            goto        start
            org         0x08
            retfie
            org         0x18
            retfie
start       movlw       dd
            movwf       quo,A           ; initialize Q register in Figure 4.1.
            movlw       dr
            movwf       dsr,A           ; initialize N register in Figure 4.1
            clrf        rem,A           ; initialize R register in Figure 4.1
            movlw       0x08
            movwf       lp_cnt          ; initialize loop count to 8
loop        bcf         STATUS,C,A      ; clear the C flag
            rlcf        quo,F,A         ; rotate (R, Q) pair to the left one place
            rlcf        rem,F,A         ; "
            movf        dsr, W,A
            subwf       rem,W,A         ; subtract and leave the difference in WREG
            btfss       STATUS,C,A      ; skip if carry is 1
            goto        negative
            bsf         quo,0,A         ; set the least significant bit of Q1 to 1
            mowvf       rem,A           ; place the difference in rem
            goto        next
negative    bcf         quo,0,A         ; set the quotient bit to 0
next        decfsz      lp_cnt,F,A      ; decrement the loop count and skip if zero
            goto        loop
            nop
            end
```

Example 4.4

Write a program to divide an unsigned 16-bit number into another unsigned 16-bit number.

Solution: The assembly program for the 16-bit unsigned division is as follows:

```
            #include <p18F8720.inc>
lp_cnt      set         0x00            ; loop count
temp        set         0x01            ; temporary storage
dsr         set         0x04            ; divisor
quo         set         0x06            ; quotient
rem         set         0x08            ; remainder
dd_h        equ         0x68            ; high byte of dividend test number
dd_l        equ         0x20            ; low byte of "
dr_h        equ         0x01            ; high byte of divisor test number
dr_l        equ         0x48            ; low byte of divisor test number
            org         0x00
            goto        start
            org         0x08
            retfie
```

```
        org     0x18
        retfie
start   movlw   dd_h        ; initialize Q register in Figure 4.1
        movwf   quo+1,A     ; "
        movlw   dd_l        ; "
        movwf   quo,A       ; "
        movlw   dr_h        ; initialize N register in Figure 4.1
        movwf   dsr+1,A     ; "
        movlw   dr_l        ; "
        movwf   dsr,A       ; "
        clrf    rem,A       ; initialize R register in Figure 4.1 to 0
        clrf    rem+1,A     ; "
        movlw   D'16'
        movwf   lp_cnt,A    ; initialize loop count to 16
loop    bcf     STATUS,C,A  ; clear the C flag
        rlcf    quo,F,A     ; rotate (R, Q) pair to the left one place
        rlcf    quo+1,F,A   ; "
        rlcf    rem,F,A     ; "
        rlcf    rem+1,F,A   ; "
        movf    dsr,W,A
        subwf   rem,W,A
        movwf   temp,A      ; save the low byte of the difference
        movf    dsr+1,W,A
        subwfb  rem+1,W,A
        btfss   STATUS,C    ; skip if carry is 1
        goto    less
        bsf     quo,o,A     ; set the quotient bit to 1
        movwf   rem+1,A     ; place the difference in R register
        movff   temp,rem    ; "
        goto    next
less    bcf     quo,0,A     ; set the quotient bit to 0
next    decfsz  lp_cnt,F,A  ; decrement the loop count and skip is zero
        goto    loop
        nop
        end
```

Unsigned division program for numbers in other lengths (e.g., 32-bit by 32-bit) can be written in the same way and hence is left for you as an exercise.

4.5 Signed Divide Operation

The one complication for the signed division is that we must also set the sign of the remainder. The following equation must always hold for division:

Dividend = Quotient × Divisor + Remainder

Our common sense requires that the magnitude of the quotient be the same as long as the magnitudes of the dividends are the same and the magnitudes of the divisors are the same. We can determine the sign of the remainder on the basis of this principle. To illustrate, let's use (±35) ÷ (±6) as an example. The first situation is simple:

35 ÷ 6: Quotient = +5, Remainder = +5

If we change the sign of the dividend, the quotient must be changed as well:

-35 ÷ 6: Quotient = -5

Rewriting our basic formula to find the remainder,

Remainder = Dividend - Quotient × Divisor
= -35 - (-5 _ 6) = -35 + 30 = -5

If we change the sign of the divisor and keep the sign of dividend unchanged,

35 ÷ (-6): Quotient = -5
Remainder = 35 - (-5 × -6) = 35 - 30 = 5

If we change the signs of both the dividend and the divisor,

-35 ÷ -6: Quotient = 5
Remainder = -35 - (-5 × -6) = -35 + 30 = -5

From this discussion, we conclude that the correctly signed division algorithm negates the quotient if the signs of the operands are opposite and makes the sign of the nonzero remainder match the dividend.s

Example 4.5

Write a PIC18 program that performs the 8-bit signed divide operation. This program will leave the quotient and remainder in the data registers represented by **quo** and **rem,** respectively.

Solution: The following program implements the 8-bit signed divide operation described in this section:

```
            #include <p18F8720.inc>
sign        set         0x00
dvd         set         0x01            ; dividend
dsr         set         0x02            ; divisor
quo         set         0x03            ; quotient
rem         set         0x04            ; remainder
lp_cnt      set         0x05            ; loop count
dd          equ         0x82            ; testing number for dividend
dr          equ         0xf5            ; testing number for divisor
            org         0x00
            goto        start
            org         0x08
            retfie
            org         0x18
            retfie
start       bcf         sign,2,A        ; initialize the sign of quotient to positive
            bcf         sign,1,A        ; initialize the sign of dividend to positive
            bcf         sign,0,A        ; initialize the sign of divisor to positive
            movlw       dd
            movwf       dvd,A
            movlw       dr
            movwf       dsr,A
            btfss       dvd,7,A         ; check the sign of dividend
            goto        second
            btg         sign,2          ; change the sign of quotient
```

```
            bsf     sign,1      ; record the sign bit of the dividend
            negf    dvd,A       ; compute the magnitude of dividend
second      btfss   dsr,7,A     ; check the sign of the divisor
            goto    do_it
            btg     sign,2,A    ; change the sign of quotient
            bsf     sign,0,A    ; set the sign of the divisor
            negf    dsr,A       ; compute the magnitude of divisor
do_it       movf    dvd,W,A
            movwf   quo,A
            clrf    rem,A       ; initialize R register in Figure 4.1
            movlw   0x08
            movwf   lp_cnt,A    ; initialize loop count to 8
loop        bcf     STATUS,C,A  ; clear the C flag
            rlcf    quo,F,A     ; rotate (R, Q) pair to the left one place
            rlcf    rem,F,A     ; "
            movf    dsr,W,A
            subwf   rem,W,A     ; subtract and leave the difference in WREG
            btfss   STATUS,C,A  ; skip if carry is 1
            goto    negative
            bsf     quo,0,A     ; set the least significant bit of Q1 to 1
            movwf   rem,A       ; place the difference in R1
            goto    next
negative    bcf     quo,0,A     ; set the quotient bit to 0
next        decfsz  lp_cnt,F,A  ; decrement the loop count and skip if zero
            goto    loop
            btfss   sign,2,A    ; skip if sign of quotient is negative
            goto    check_re    ; "
            negf    quo,A
check_re    btfss   sign,1,A    ; skip if dividend is negative
            goto    ok_skip     ; "
            negf    rem,A
ok_skip     nop
            end
```

Example 4.6

Write a program to divide a signed 16-bit number into another 16-bit signed integer.

Solution: The following program will perform the signed 16-bit divide operation:

```
            #include <p18F8720.inc>
sign        set     0x00        ; keep track of the signs of dividend and divisor
dvd         set     0x02        ; dividend
dsr         set     0x04        ; divisor
quo         set     0x06        ; quotient
rem         set     0x08        ; remainder
lp_cnt      set     0x0A        ; loop count
temp        set     0x0B        ; temporary storage
dd_h        equ     0xD9        ; testing number for dividend
```

```
dd_l    equ     0xB8        ; "
dr_h    equ     0xFF        ; testing number for divisor
dr_l    equ     0x80        ; "
        org     0x00
        goto    start
        org     0x08
        retfie
        org     0x18
        retfie
start   bcf     sign,2,A    ; initialize the sign of quotient to positive
        bcf     sign,1,A    ; initialize the sign of dividend to positive
        bcf     sign,0,A    ; initialize the sign of divisor to positive
        movlw   dd_l        ; set up dividend
        movwf   dvd,A       ; "
        movlw   dd_h        ; "
        movwf   dvd+1,A     ; "
        movlw   dr_l        ; set up divisor
        movwf   dsr,A       ; "
        movlw   dr_h        ; "
        movwf   dsr+1,A     ; "
        btfss   dvd+1,7,A   ; check the sign of dividend
        goto    second
        btg     sign,2      ; change the sign of quotient
        bsf     sign,1      ; record the sign bit of the dividend
        comf    dvd,F,A     ; compute the magnitude of dividend
        comf    dvd+1,F,A   ; "
        incf    dvd,F,A     ; "
        movlw   0x00        ; "
        addwfc  dvd+1,F,A   ; "
second  btfss   dsr+1,7,A   ; check the sign of the divisor
        goto    do_it
        btg     sign,2,A    ; change the sign of quotient
        bsf     sign,0,A    ; set the sign of the divisor
        comf    dsr,F,A     ; compute the magnitude of divisor
        comf    dsr+1,F,A   ; "
        incf    dsr,F,A     ; "
        movlw   0x00        ; "
        addwfc  dsr+1,F,A   ; "
do_it   movff   dvd,quo     ; place dividend in Q register in Figure 4.1
        movff   dvd+1,quo+1
        clrf    rem,A       ; initialize R register in Figure 4.1
        clrf    rem+1,A     ; "
        movlw   D'16'
        movwf   lp_cnt,A    ; initialize loop count to 8
loop    bcf     STATUS,C,A  ; clear the C flag
        rlcf    quo,F,A     ; rotate (R, Q) pair to the left one place
        rlcf    quo+1,F,A   ; "
        rlcf    rem,F,A     ; "
        rlcf    rem+1,F,A   ; "
        movf    dsr,W,A     ; compute R-N and places the difference
```

```
            subwf   rem,W,A     ; in WREG and temp
            movwf   temp,A      ; "
            movf    dsr+1,W,A   ; "
            subwfb  rem+1,W,A   ; "
            btfss   STATUS,C,A  ; skip if carry is 1
            goto    negative
            bsf     quo,0,A     ; set the least significant bit of Q to 1
            movwf   rem+1,A     ; place the difference in R in Figure 4.1
            movff   temp,rem    ; "
            goto    next
negative    bcf     quo,0,A     ; set the quotient bit to 0
next        decfsz  lp_cnt,F,A  ; decrement the loop count and skip if zero
            goto    loop
            btfss   sign,2,A    ; skip if sign of quotient is negative
            bra     check_re    ; "
            comf    quo,F,A     ; complement the quotient
            comf    quo+1,F,A   ; "
            incf    quo,F,A     ; "
            movlw   0x00        ; "
            addwfc  quo+1,F,A
check_re    btfss   sign,1,A    ; skip if dividend is negative
            bra     ok_skip     ; "
            comf    rem,F,A     ; complement the remainder
            comf    rem+1,F,A   ; "
            incf    rem,F,A     ; "
            movlw   0x00        ; "
            addwfc  rem+1,F,A   ; "
ok_skip     nop
            end
```

Signed division program for numbers in other lengths (e.g., 32-bit by 32-bit division) can be written in the same manner and hence are left for you as an exercise. ▲

4.6 The Stack

A stack is a first-in-last-out (or last-in-first-out) data structure. To implement a stack, two things are needed:

1. A stack pointer that points to the top (or the byte immediately above the top) of the stack
2. A block of RAM of adequate size

The PIC18 MCU has no register designated as the stack pointer. However, the user can use one of the FSR registers as the stack pointer and use one or more banks of the data memory to implement the data stack. The stack implemented this way is called a *software stack*.

A stack can grow from a low address toward higher addresses or from a high address toward lower addresses. This text follows the convention used by the Microchip C18 compiler:

1. Use the FSR1 register as the stack pointer and set it to point to the next available byte on the stack as shown in Figure 4.2.
2. Grow the stack from a low address toward higher addresses.