EECE.3170: Microprocessor Systems Design I

Summer 2017

Homework 5 Solution

For each of the following complex operations, write a sequence of PIC 16F1829 instructions that performs an equivalent operation. Assume that X, Y, and Z are 16-bit values split into individual bytes as shown in the following cblock directive, which defines two additional variables you can use:

```
cblock 0x70
     XH, XL     ; High and low bytes of X
     YH, YL     ; High and low bytes of Y
     ZH, ZL     ; High and low bytes of Z
     TEMP       ; Temporary byte, if needed
endc
```

a. *Perform the 16-bit addition: X = Y + Z. Do not change Y or Z when performing this operation.*

Solution: First, we'll look at the inefficient method, which would work on any PIC microcontroller:

```
movf      YL, W        ; Copy YL to XL
movwf     XL
movf      YH, W        ; Copy YH to XH
movwf     XH
movf      ZL, W        ; Add low bytes
addwf     XL, F
btfsc     STATUS, C    ; Account for carry
incf      XH, F
movf      ZH, W        ; Add high bytes
addwf     XH, F
```

We can do this operation more efficiently by using the addwfc instruction found on microcontrollers like the PIC16F1829, which allows you to get rid of the extra instructions that "account for the carry:"

```
movf      YL, W        ; Copy YL to XL
movwf     XL
movf      YH, W        ; Copy YH to XH
movwf     XH
movf      ZL, W        ; Add low bytes
addwf     XL, F
movf      ZH, W        ; Add high bytes, including carry from low byte
addwfc    XH, F
```

b. *Perform the 16-bit subtraction: X = Y – Z. Do not change Y or Z when performing this operation.*

Solution: This operation is very similar to 16-bit addition, although you have to be more careful about what register is moved into the working register before the subtract instructions. First, the inefficient version—remember that C = 0 if a borrow occurs:

```
movf     YL, W        ; Copy YL to XL
movwf    XL
movf     YH, W        ; Copy YH to XH
movwf    XH
movf     ZL, W        ; Subtract low bytes
subwf    XL, F
btfss    STATUS, C    ; Account for borrow (C = 0 → "borrow" = 1)
decf     XH, F
movf     ZH, W        ; Subtract high bytes
subwf    XH, F
```

And the more efficient version that uses the "subtract with borrow" subwfb instruction:

```
movf     YL, W        ; Copy YL to XL
movwf    XL
movf     YH, W        ; Copy YH to XH
movwf    XH
movf     ZL, W        ; Subtract low bytes
subwf    XL, F
movf     ZH, W        ; Subtract high bytes, taking borrow into
subwfb   XH, F        ;   account
```

c. *Perform a 16-bit arithmetic right shift: X = Y >> ZL. (Note that, because the shift amount is no greater than 15, a single byte is sufficient to hold that value.) Do not change Y or ZL when performing this operation.*

Solution: Similarly to the last two problems, the first thing to be done is move the value to be shifted into the destination registers XH and XL. Once that's done, set up a loop with ZL iterations (we'll have to copy that value to another register so it's not changed) and do the shift. Remember, while the shift for the high byte can be an arithmetic shift, we need a rotate instruction to change the low byte so that the bit shifting between bytes is correctly accounted for.

```
        movf      YL, W       ; Copy YL to XL
        movwf     XL
        movf      YH, W       ; Copy YH to XH
        movwf     XH
        movf      ZL, W       ; Copy ZL to TEMP
        movwf     TEMP
L:      asrf      XH, F       ; Shift upper byte (C = bit to be shifted into XL)
        rrf       XL, F       ; Shift lower byte
        decfsz    TEMP, F     ; Decrement loop counter and return to start
        goto      L           ;    of loop if there are more iterations.
```

d. Given an 8-bit variable, YL, perform the multiplication:

*YL = YL * 10*

<u>Hint:</u> Note that multiplication by a constant amount can be broken into a series of shift and add operations. For example, in general:

- *X * 2 can be implemented by shifting X to the left by 1 (X << 1)*
- *X * 5 can be implemented as (X * 4) + X = (X << 2) + X*

<u>Solution:</u> Recognize that YL * 10 = (YL * 8) + (YL * 2) = (YL << 3) + YL + YL

```
        movf     YL, W         ; Copy original value of YL into TEMP
        movwf    TEMP
        movlw    3             ; Set W = 3—use as loop counter for left shift
L:      lslf     YL, F
        addlw    -1            ; Decrement loop counter
        btfss    STATUS, Z     ;   and exit once it reaches 0
        goto     L
        movf     TEMP, W       ; W = TEMP = original value of YL
                               ; At this point, YL = (original YL) << 3
                               ;    = (original YL) * 8
        addwf    YL, F         ; YL = (original YL) * 9
        addwf    YL, F         ; YL = (original YL) * 10
```

e. Given two 8-bit variables stored in XL and YL, copy the value of bit position YL within variable XL into the carry flag. For example:

- *If XL = 0x03 and YL = 0x00, set C to the value of bit 0 within XL.*
  - *Since XL = 0x03 = 0000 001<u>1</u>$_2$, C = 1*
- *If XL = 0xC2 and YL = 0x04, set C to the value of bit 4 within XL.*
  - *Since XL = 0xC2 = 110<u>0</u> 0011$_2$, C = 0*

Note that:

- *This operation is very similar to the bit test (BT) instruction in the x86 architecture.*
- *Since YL is not a constant, you cannot use the value of YL directly in any of the PIC bit test instructions (for example,* btfsc XL, YL *is <u>not</u> a valid instruction).*
- *Your code should not modify either XL or YL.*

<u>Solution</u>

```
        movlw   0x01        ; TEMP will hold bit mask used
        movwf   TEMP        ;       to isolate bit YL within XL
        movf    YL, W       ; Copy B to W—determines # of times to shift temp
L:      btfsc   STATUS, Z   ; Once W hits 0, end loop—bit mask is set
        goto    L2          ; Must test this first for case where YL == 0
        rlf     TEMP, F     ; TEMP will eventually be 1 << YL
        addlw   -1          ; Decrement W
        goto    L
L2:     bcf     STATUS, C   ; Clear C
        movf    TEMP, W     ; AND temp with XL to mask out all but bit YL
        andwf   XL, W
        btfss   STATUS, Z   ; If result is non-zero, set C bit; otherwise,
        bsf     STATUS, C   ;   leave as 0
```