EECE.3170: Microprocessor Systems Design I

Fall 2016

Homework 9 Solution

1. *(40 points) You are given the following function containing a delay loop:*

```
Ten_1
    decfsz     COUNTL,F        ; Inner loop
    goto       Ten_1
    decfsz     COUNTH,F        ; Outer loop
    goto       Ten_1
    return
```

a. *(20 points) If COUNTL is initially 100, COUNTH is initially 10, the clock frequency is 500 kHz, and each instruction takes 4 clock cycles, how long does the whole delay loop take? You must show your work for full credit.*

Solution: The total amount of delay is the product of four factors: the clock cycle time, the number of cycles per instruction, the number of instructions per loop iteration, and the number of iterations. To determine each of these:

- Clock cycle time = 1 / (clock frequency) = 1 / 500 kHz = 2 x 10^{-6} sec = 2 μs
- Cycles per instruction is given as 4, which means each instruction takes 4 * 2 = 8 μs
- The number of instructions per iteration and number of loop iterations can be considered together to determine the total number of instructions:
  - For each decfsz/goto pair, the goto instruction is executed one fewer time than the decfsz because the goto is skipped the last time through the loop.
  - The number of iterations is based on the initial value of the variable being decremented:
    - For the outer loop, the decfsz executes 10 times and the goto executes 9.
    - The inner loop works differently on the first iteration than on all others:
      - 1st time: decfsz executes 100 times, goto executes 99
      - 2nd through 10th times: decfsz executes 256 times (since COUNTL is initially 0), goto executes 255.
  - The total number of instructions can be broken down as shown below, with the total instruction count for each instruction shown in red:

```
Ten_1
    decfsz     COUNTL,F   100 + 9 * 256 = 2404
    goto       Ten_1      99 + 9 * 255 = 2394
    decfsz     COUNTH,F   10
    goto       Ten_1      9
    return                1
               TOTAL:     2404 + 2394 + 10 + 9 + 1 = 4818
```

The total delay is therefore (4818 instructions) * (8 μs per instruction) = 38544 μs = **38.544 ms.**

b. *(20 points) What are the maximum and minimum possible delays this function can generate? What initial values would COUNTL and COUNTH have in each case?*

Solution: The maximum and minimum possible delays are based on the maximum and minimum possible number of loop iterations.

For the maximum delay, each loop would execute 256 times, since COUNTL and COUNTH are each 8-bit values and can therefore hold 256 different values (0-255). To obtain this maximum delay, **COUNTL and COUNTH are both initially 0.** We can use similar analysis to part (a) to find the total number of instructions executed:

```
Ten_1
    decfsz     COUNTL,F   256 * 256 = 65536
    goto       Ten_1      256 * 255 = 65280
    decfsz     COUNTH,F   256
    goto       Ten_1      255
    return                1
               TOTAL:     65536 + 65280 + 256 + 255 + 1 = 131328
```

The maximum delay is therefore (131328 instructions) * (8 µs per instruction) = 1,050,624 µs = **1.050624 seconds.**

For the minimum delay, each loop would exit in its first iteration, with each `decfsz` instruction skipping the `goto` instruction that immediately follows it. In order for the function to behave that way, **COUNTL and COUNTH must both initially be 1.**

The function would therefore execute a total of 3 instructions—the two decfsz instructions and the return instruction, since both gotos are skipped. That means the minimum possible delay is (3 instructions) * (8 µs per instruction) = **24 µs.**

2. *(40 points) You are given the following short PIC16F1829 assembly function:*

```
F: movf     PORTC, W
   andlw    B'00000001'
   addwf    PCL, F
   retlw    B'11110000'
   retlw    B'00111100'
   retlw    B'00001111'
   retlw    B'11111111'
```

a. *(20 points) How many possible return values does this function have? Give an example of a value stored in PORTC that would cause the function to return each of those possible values.*

Solution: The function has four `retlw` instructions, so it might appear to have four return values. As discussed using a similar example in class, adding a value to the program counter (PCL) chooses one of the following instructions by specifying the number of instructions to skip. In this case, the value added to PCL is the result of the `andlw` just before it. That instruction's result will be either 0 or 1, meaning this function has only **two return values**—the values returned by the first two `retlw` instructions (`B'11110000'` and `B'00111100'`).

0x00 is one example of a PORTC value that will cause the function to return `B'11110000'`. (Any value in which the least significant bit is 0 would work.)

0x01 is one example of a PORTC value that will cause the function to return `B'00111100'`. (Any value in which the least significant bit is 1 would work.)

b. *(10 points) Is it possible for the function to execute each of the 4 `retlw` instructions? If so, explain how, and if not, explain how you would modify the function to make each of those four instructions reachable.*

Solution: The answer above explains why it's not possible for the function to execute each of the four `retlw` instructions—the `andlw` instruction only generates 0 and 1 as possible values, making only the first two `retlw` instructions reachable.

To make all four instructions reachable, the `andlw` instruction must generate four different values between 0 and 3 so the addwf instruction functions as a jump that skips between 0 and 3 instructions. To obtain that result, **change the andlw instruction to: `andlw B'00000011'`**

c. *(30 points) Explain what effect each of the following pieces of code would have on I/O port A. Assume you are using the original version of the function F, not your (potentially) modified version from part (b).*

Note: Since we're using the original version of the function F, the only two values that function will return are `B'11110000'` and `B'00111100'`. The function return values then work as bitmasks for the logical operation that follows the function call.

i.
```
call    F
xorwf   LATA, F
```

Solution: XOR instructions will flip a group of bits depending on the bitmask used—any positions in which the bitmask holds a 1 will be flipped, while any positions in which the bitmask holds a 0 will remain the same. (For example, if port A holds B'10101010', XORing that value with B'11110000' will yield the result B'01011010'—the upper four bits are flipped, while the lower four bits stay the same.)

Therefore, this piece of code will flip either the upper four bits or middle four bits of port A.

ii.
```
call    F
iorwf   LATA, F
```

Solution: OR instructions will set a group of bits to 1 depending on the bitmask used—any positions in which the bitmask holds a 1 will be set to 1, while any positions in which the bitmask holds a 0 will remain the same. (For example, if port A holds B'10101010', ORing that value with B'11110000' will yield the result B'11111010'—the upper four bits are set to 1, while the lower four bits stay the same.)

Therefore, this piece of code will set either the upper four bits or middle four bits of port A to 1.

iii.
```
call    F
andwf   LATA, F
```

Solution: AND instructions will clear a group of bits (set them to 0) depending on the bitmask used—any positions in which the bitmask holds a 1 will remain the same, while any positions in which the bitmask holds a 0 will be cleared. (For example, if port A holds B'10101010', ANDing that value with B'11110000' will yield the result B'10100000'—the upper four bits stay the same, while the lower four bits are set to 0.)

Therefore, this piece of code will clear either the lower four bits of port A, or the two lowest and two highest bits of port A.