16.317: Microprocessor-Based Systems I

Fall 2012

Exam 3 Solution

1. (20 points, 5 points per part) ***Multiple choice***

For each of the multiple choice questions below, clearly indicate your response by circling or underlining the single choice you think best answers the question.

a. When writing PIC 16F684 assembly code, which of the following multi-byte operations will use the carry bit (C) to transfer information from one byte to the next?

A. 16-bit addition

B. 32-bit AND

C. 32-bit left shift

D. 16-bit XOR

i. Only A ***2/5 points—got 1 of 2 operations correct***

ii. A and B

iii. A and C

iv. B and C

v. B and D

b. As discussed in class, many of the jump conditions supported by the 80386DX can be tested on the PIC 16F684 using a combination of the carry (C) and zero (Z) bits. Which of the following jump conditions **cannot** be tested using simply the C and Z bits?

i. Equality (i.e., previous compare operation shows that two values are equal)

ii. Less than (i.e., previous compare operation shows the first value is less than the second)

iii. Overflow (i.e., previous operation generated a result that cannot be represented in the number of bits available)

iv. Not zero (i.e., previous operation generates a non-zero result)

1 (continued)

c. Which of the following statements about shift and rotate operations using 16-bit values (and therefore multiple registers) on the PIC16F684 are **true**?

A. In a 16-bit left shift, the most significant byte should be shifted first to ensure that the correct data is transferred between bytes.

B. In a 16-bit rotate operation that does not include the carry, the initial value of the C bit should always be 0.

C. In a 16-bit arithmetic right shift, the initial value of the C bit should always be 1.

D. 16-bit rotate and shift operations with a shift amount of 1 can be done using a single PIC16F684 instruction.

i. Only C

ii. A and C

iii. B and D

iv. A and D

v. None of the above (i.e., A, B, C, and D are all false)

d. Which of the following statements most accurately describes how to handle 16-bit subtraction on the PIC16F684? Assume the subtraction in question is AX – BX, where AX is broken into bytes AL / AH, and BX is broken into bytes BL / BH.

i. Subtract AL – BL first, then subtract AH – BH. ***2/5—missing decrement based on C***

ii. Subtract AH – BH first, then subtract AL – BL.

iii. Subtract AL – BL first, then check the carry bit (C). If C is 1, decrement AH. Then, subtract AH – BH.

iv. Subtract AH – BH first, then check the carry bit (C). If C is 1, decrement AL. Then, subtract AL – BL. ***2/5—wrong order of bytes***

v. Stop worrying about how to do this on an 8-bit microcontroller and get yourself one that can handle a multi-byte operation in a single instruction.

(Note: (v) might be your favorite answer, it's probably not going to earn you any points.)

2. (40 points) ***PIC programming sequences***
Complete each program below by writing the appropriate PIC instruction into each of the blank spaces. The purpose of each instruction is described in a comment to the right of the blank.

a. (12 points)
This short program initializes an 8-bit counter, which is stored in register COUNT, to 0x12, then count down until it reaches 0.

```
        movlw   0x12            ; Set W = 0x12

        movwf   COUNT           ; Set COUNT = W = 0x12

L:      decfsz  COUNT, F        ; Decrement COUNT and
                                ;   skip next instruction if
                                ;   it's 0
        goto    L               ; Return to start of loop
```

b. (12 points)
This function returns a value that can be used to directly overwrite the current state of PORTC, which ensures that the following state transitions are made. Note that the lowest two bits of PORTC encode the state, the bits are shown from most significant to least significant, and the sequence repeats after 3 steps: 00 → 10 → 01 → 00

If the lowest two bits of PORTC match none of the states shown above, set those bits equal to 0. Note that the upper six bits of PORTC are unused and can therefore be overwritten with 0 values.

```
F:
        movf    PORTC, W        ; Read state of PORTC into W

        andlw   0x03            ; Clear all but lowest 2 bits of W

        addwf PCL, F            ; Add lowest 2 bits to PCL
                                ;   to pick instruction below
        retlw   b'00000010'     ; Go from 00 → 10

        retlw   b'00000000'     ; Go from 01 → 00

        retlw   b'00000001      ; Go from 10 → 01

        retlw   b'00000000'     ; Go from 11 → 00
```

2 (continued)

c. (16 points) The following short program repeatedly calls a function, ThreeVals, that uses the retlw instruction to return one of two values. The values have the following effects on the variable X:

- If the function returns 0, the program should clear X.
- If the function returns 1, the program should increment X.

```
L:      call    ThreeVals                   ; Call function ThreeVals

        andlw   0x01  (other instructions possible)   ; Test if W == 0; note that
                                                      ;   Z should be 1 if W == 0

        btfss   STATUS, Z                   ; Skip next instruction if Z == 1
                                            ;   and therefore W == 0

        goto    R1                          ; Handle case where W == 1

        clrf    X                           ; Clear x

        goto    L                           ; Return to start of loop

R1:     incf    X, F                        ; Increment x

        goto    L                           ; Return to start of loop
```

3. (40 points, 20 points per part) ***Complex operations***

For each of the following 80386 instructions, write a sequence of PIC 16F684 instructions that performs an equivalent operation. The operation is described in italics. **CHOOSE ANY TWO OF THE THREE PARTS** and fill in the space provided with appropriate code. **You may complete all three parts for up to 10 points of extra credit, but must clearly indicate which part is the extra one—I will assume it is part (c) if you mark none of them.**

Assume that variables with the same names are defined for all 8-bit 80386 registers (for example, "AL" and "BL"). If an operation uses a 16-bit register (e.g., AX), you can address each byte within that register (e.g. AH and AL). Also assume "TEMP" has been defined for cases where you may need an extra variable.

Note that shift or rotate operations should not be done by simply writing copies of the PIC rotate instructions. Use the shift amount provided as a literal value that will help determine the number of times you shift or rotate.

Finally, please note that you are not required to write comments describing each instruction. You are certainly welcome to do so if you feel it will make your solution clearer to the instructor.

a. SAR AX, 10 *(Shift 16-bit value AX right by 10 bits; maintain original sign)*

```
        movlw  10               ; W = 10
        movwf  TEMP             ; TEMP = W = 10
L:      bcf    STATUS, C        ; C = 0
        btfsc  AH, 7            ; Skip if MSB == 0
        bsf    STATUS, C        ; C = 1 if MSB == 1
                                ; C will hold copy of
                                ; MSB (keeping sign
                                ; intact)
        rrf    AH, F            ; Rotate AH right by 1
                                ; Bit rotated between bytes goes through C
        rrf    AL, F            ; Rotate AL right by 1
        decfsz TEMP, F          ; Decrement & test COUNT
        goto   L                ; Return to start of loop if
                                ;   COUNT != 0
```

3 (continued)

b. SETG AL *(AL = 0xFF if result of previous comparison is "greater than"; AL = 0x00 otherwise)*

Note that the "greater than" condition can be tested by checking if Z == 0 and C == 1

```
        clrw                ; W = 0
        btfsc  STATUS, Z    ; If Z == 0, check C
        goto   End          ; Otherwise, W remains 0
        btfsc  STATUS, C    ; If C == 0, W remains 0
        movlw  0xFF         ; Otherwise, Z == 0 and C == 1, so set W = 0xFF
End:    movwf  AL           ; Copy W to AL—will be either 0 or 0xFF
```

c. ADC AX, BX *(AX = AX + BX + C)*

Remember, this instruction takes the value of C before the addition starts and adds it in.

```
        btfsc  STATUS, C    ; If C == 0, skip next instruction
        incf   AL           ; If C == 1, increment one of the bytes (could be BL)

        btfsc  STATUS, C    ; Must now check result of increment—if increment
        incf   AH           ;   produced carry, increment one of the upper bytes
                            ; If there was no increment, C remains unchanged
                            ;   and you'll skip the second increment operation

        movf   BL, W        ; Next two instructions: add lowest bytes
        addwf  AL, F

        btfsc  STATUS, C    ; Check carry—if clear, skip to next bytes (AH/BH)
        incf   AH, F        ; If carry set, add 1 to AH (could be BH)

        movf   BH, W        ; Next two instructions: add AH/BH
        addwf  AH, F
```