# PIC Architecture
# Instruction Set & Operations

# PICs-Instruction Set

- Have Covered Instruction Set Basics
  - Accumulator Architecture
  - Direct addressing
  - Indirect addressing
- Now lets look at the instructions

# MOVE instructions

- PIC spend a lot of time moving data around as data stored in memory
- movlw 20
  - Move the hex value H’20’ into W
  - To load a decimal must use the correct assembler directive - D’20’
- movlw -2
  - Loads B’1111 1110 into WREG

# More on MOVE

- Initialization of a variable
  - movlw B'11100000'
  - movwf TRIST
  - Initialize the PORTB data direction register
- Assembler MACRO
  - MOVLF B'11100000',TRISB
  - Assembled into the two instructions above

# The movff instruction

- movff    PORTB, PORTB_COPY
  - movff   - a two-word instruction
  - Thus source and destination have 12-bit addresses
    - Source – instruction bits + BSR
    - Destination – instruction 2nd byte
  - Moves data from a 1-byte source to a 1-byte destination.
  - For instruction memory as efficient as the regular move instruction.

# The movwf

- movwf f(,Banked) – Move WREG to f

- For storing back the result of an operation
- Does not affect status bits

# The movf instruction

- Move the value and set the status register bits appropriately
- Affects N and Z bits

# Move summary

- movlw  k   - load literal value
- movwf  MYVAR  - move value but do not affect status bits
- movff  V1,V2  - move data at source address to data at destination address
- movf  f,F/W  - move value and affect status bits
  - What does    movf  COUNT,F    do?

# Other move/load instructions

- lrsr 0,num1 - load FSR register
  - 1st argument is FSR register number
  - 2nd argument is value to load
- Saving a FSR register
  - movff FSR0L,FSR0L_TEMP
  - movff FSR0H,FRS0H_TEMP
- Loading a FSR register
  - movff FSR0L_TEMP,FSR0L
  - movff FRS0H_TEMP,FSR0H

# Load BSR register & other

- movlb   2
  - Load the value of 2 into the BSR register

- clrf  TEMP – Load 0x00 into variable TEMP
- setf  TEMP – Load 0xff into variable TEMP
- swapf  f      - swap nibbles of f

# Single operand instructions

- Single bit instructions
  - bsf PORTB,0 - Set the lsb of PORTB
  - bcf PORTB,1 - clear bit 1 of PORTB
  - btg PORTB,2 - toggle bit 2 of PORTB
- Rotate instructions illustrated on next slide
  - rlcf rlncf rrcf rrncf
  - cf rotate including carry bit
  - ncf rotate without carry bit

# Logical instructions

- andlw B’00001111’ And WREG with value
- andwf f,F/W - AND WREG with f putting result back in F or WREG
- iorlw k -Inclusive OR literal value and WREG
- iorwf f,F/W – inclusive OR WREG with f and put result into f or WREG
- xorlw k, xorwf f,F/W - Exclusive OR

# Arithmetic

- addlw k,  addwf  f,F/W
- addwfc  f,F/W  - Add WREG, F and carry bit
- daw – Decimal adjust sum of pack BCD
- subwf, sublw
- subwfb  f,F/W  - subtract with borrow

# Multiplicaiton

- mullw k - multiply WREG with literal value k putting result in PRODH:PRODL reg - WREG unaffected
- mullwf f(,Banked) - Multiply WREG with f putting results in PRODH:PRODL - both WREG and f remain unchanged

# Branches

- Needed for program flow control
- Tests on status register
  - bc, bnc, bz, bnz, bn, bnn, bov, bnov
  - Use the c, x, n, and ov bits of the status register
- bra – branch always

# Conditional Skip instructions

- Ten further instructions that test for a condition and skip over the next instruction if the condition is met.
  - Next instruction is typically a branch or rcall
  - Very useful at end of a loop

  - Loop code ….
  - decfsz  count,F    ;Decrement and skip if zero
  - bra    top_of_loop

# Skip instructions

- cpfseq f - skip if f = WREG
- cpfsgt f - skip if f > WREG
- cpfslt f - skip if f < WREG
- tstfsz t - Test f, skip if zero
- decfsz f,F/W - Decr f, res->WREG,skip if 0
- dcfsnz f,F/W - Decr f, res->WREG,skip not 0
- incfsz f,F/W – Incr f, res->WREG, skip if 0
- infsnz f,F/W – Incr f, res->WREG, skip not 0

# Other – Subroutine, interrupt

- rcall label  - call subroutine (within 512 instr)
- call label – call subroutine (anywhere)
- call  label, FAST  - call subroutine, copy state to shadow registers
- return – return form subroutine
- return  FAST  - return and restore from shadow registers
- return  k  - return and put value k in WREG

# cont

- retfie   - return from interrupt and re-enable
- retfie FAST – return, re-enable- restore
- push - Push addr of next instruction onto stack
- pop   - discard address on top of stack
- clrwdt  - clear watchdog timer
- sleep  - Go into standby mode
- reset   - software controlled reset
- nop

# Review: PIC instructions

- Logical operations
  - andlw/andwf
  - iorlw/iorwf
  - xorlw/xorwf
- Rotates
  - rrf
  - rlf
- Jumps/calls/return
  - goto
  - call
  - return/retlw/retfie
- Miscellaneous
  - nop
  - sleep/clrwdt

# Conditional Execution

STATUS bits:

none

- Conditional execution in PIC: skip next instruction if condition true
- Two general forms
  - Test bit and skip if bit clear/set
  - Increment/decrement register and skip if result is 0

```
btfsc    f, b        ;Test bit b of register f, where b=0 to 7, skip if clear
btfss    f, b        ;Test bit b of register f, where b=0 to 7, skip if set
decfsz   f, F(W)     ;decrement f, putting result in F or W, skip if zero
incfsz   f, F(W)     ;increment f, putting result in F or W, skip if zero
```

Examples:

```
btfsc   TEMP1, 0     ; Skip the next  instruction if bit 0 of TEMP1 equals 0
btfss   STATUS, C    ; Skip the next instruction if C==1
decfsz  TEMP1, F     ; Decrement TEMP1, skip if TEMP1==0
incfsz  TEMP1, W     ; W <- TEMP1+1 , skip if W==0 (TEMP1==0xFF)
                     ; Leave TEMP1 unchanged
```

# Example

- Show the values of all changed registers after each of the following sequences
  - What high-level operation does each perform?

(a)

```
        movf    a, W
        sublw   0xA
        btfsc   STATUS, Z
        goto    L1
        incf    b, W
        goto    L2
L1
        decf    b, W
L2
        movwf   a
```

(b)

```
        movf    NUM2, W
        subwf   NUM1, W
        btfss   STATUS, C
        goto    BL
        movf    NUM1, W
        goto    Done
BL
        movf    NUM2, W
Done
        movwf   MAX
```

# Example solution (part a)

```
        movf    a, W            → W = a
        sublw   0xA             → W = 10 – a
        btfsc   STATUS, Z       → Skip goto if result
                                   is non-zero
        goto    L1              → Goto L1 if result == 0
                                → Reach this point if
                                         result non-zero
        incf    b, W            → W = b + 1
        goto    L2
L1
        decf    b, W            → W = b - 1
L2
        movwf   a               → a = W → value depends
                                on what's executed before this
```

**High-level operation:**

**if ((10 – a) == 0)**

**a = b – 1**

**else**

**a = b + 1**

# Example solution (part b)

```
        movf    NUM2, W        → W = NUM2
        subwf   NUM1, W        → W = NUM1 – W
                               = NUM1 – NUM2
        btfss   STATUS, C  → Carry indicates
                               "above"
                               → if set, NUM1 > NUM2
        goto    BL
        movf    NUM1, W        → if (NUM1 >= NUM2)
                                        W = NUM1
        goto    Done           → Skip "below" section
BL
        movf    NUM2, W        → if (NUM1 < NUM2)
                                        W = NUM2
Done
        movwf   MAX
```

**<u>High-level operation:</u>**

**if (NUM1 < NUM2)**

**MAX = NUM2**

**else**

**MAX = NUM1**

# Working with 16-bit data

Assume a 16-bit counter, the upper byte of the counter is called COUNTH and the lower byte is called COUNTL.

## Decrement a 16-bit counter

```
        movf    COUNTL, F       ; Set Z if lower byte == 0
        btfsc   STATUS,  Z
        decf    COUNTH, F       ; if so, decrement COUNTH
        decf    COUNTL, F       ; in either case decrement COUNTL
```

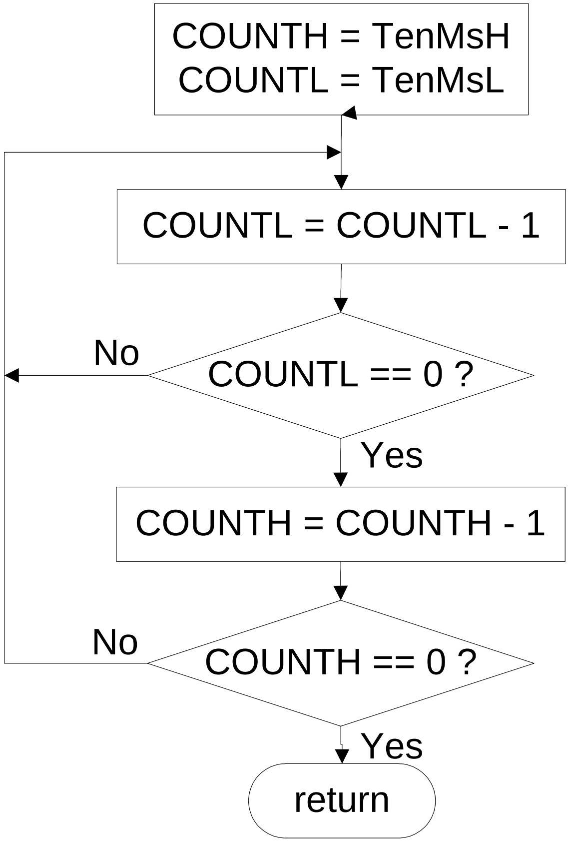
## Test a 16-bit variable for zero

```
        movf    COUNTL, F       ; Set Z if lower byte == 0
        btfsc   STATUS, Z       ; If not, then done testing
        movf    COUNTH, F       ; Set Z if upper byte == 0
        btfsc   STATUS, Z       ; if not, then done
        goto    BothZero        ; branch if 16-bit variable == 0
CarryOn
```

# A Delay Subroutine

```
; ******************************************************************************
; TenMs subroutine and its call inserts a delay of exactly ten milliseconds
; into the execution of code.
; It assumes a 4 MHz crystal clock. One instruction cycle = 4 * Tosc.
; TenMsH     equ  13        ; Initial value of TenMs Subroutine's counter
; TenMsL     equ  250
; COUNTH and COUNTL are two variables

TenMs
      nop                              ; one cycle
      movlw    TenMsH                  ; Initialize COUNT
      movwf    COUNTH
      movlw    TenMsL
      movwf    COUNTL
Ten_1
      decfsz   COUNTL,F                ; Inner loop
      goto     Ten_1
      decfsz   COUNTH,F                ; Outer loop
      goto     Ten_1
      return
```



# Applications

- Personal information products: Cell phone, pager, watch, pocket recorder, calculator
- Laptop components:  mouse, keyboard, modem, fax card, sound card, battery charger
- Home appliances:  door lock, alarm clock, thermostat, air conditioner, tv remote, hair dryer, VCR, small refrigerator, exercise equipment, washer/dryer, microwave oven
- Toys; video games, cars, dolls, etc.

# Summary

- Microprocessors and embedded controllers are a ubiquitous part of life today
- These devices come in a wide variety of configurations and designs
- Headhunters report that EEs familiar with µC, µP design are in the highest possible demand
- Feedback

# Assignment

- Illustrate PIC instruction set with example.