

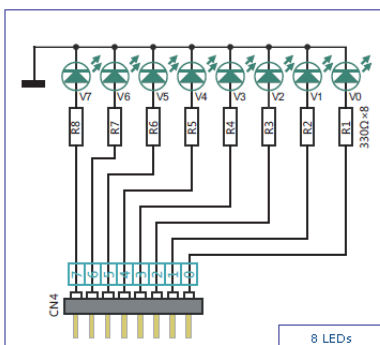
Chapter 3

PIC18 I/O Programming

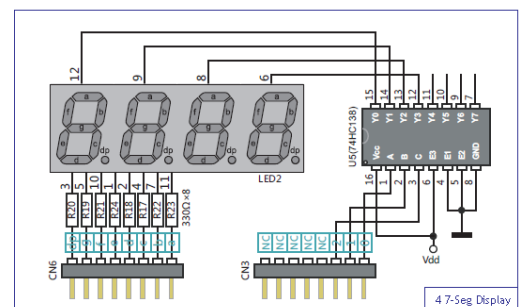
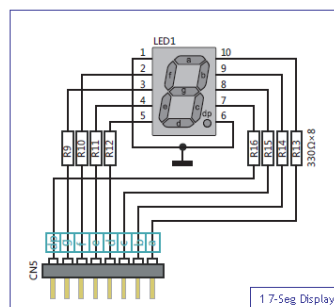
1

PIC18 Parallel Ports

- I/O ports are available in PIC18 to interact, monitor and control peripherals.



LED



**7-Segment LED
to display digits**

2

PIC18 Parallel Ports

- The number of ports in the PIC18 family depends on the number of pins on the chip.
- PIC18F4520 has five ports: Port A-E
- Different ports have different number of pins:
 - Port A, B, C and D: 8 pins
 - Port E: 3 pins

3

PIC18 Parallel Ports

- The behaviour of each port is controlled by two special function registers (SFRs):
 - PORTx – indicates the voltage levels on the pins of the device
 - TRISx – data direction register
- e.g., For Port B, we have PORTB and TRISB

4

TRISx and PORTx SFRs

- Each of the Ports A-E can be used for input and output.
- TRISx SFR is used for designating the direction of a port
 - 0 for output (e.g., Controlling LED state)
 - 1 for input (e.g., Key scan)
- e.g., To output data to Port B:
 1. Write 0s to TRISB SFR
 2. Write data to PORTB SFR

5

TRISx and PORTx SFRs

- e.g., output the hex value 0x26 to Port C

```
clrf TRISC
movlw 0x26
movwf PORTC
```
- e.g., Read the current value of Port D into WREG register

```
setf TRISD
movf PORTD, W
```

6

I/O ports and bit-addressability

- Often need to access individual bit of the port instead of the entire 8 bits.
- PIC18 provides instructions that alter individual bits without altering the rest of the bits in the port.
- Most common bit-oriented instructions:

Instructions

bsf fileReg, bit

bcf fileReg, bit

btg fileReg, bit

btfsc fileReg, bit

btfss fileReg, bit

Function

Bit Set fileReg

Bit Clear fileReg

Bit Toggle fileReg

Bit test fileReg, skip if clear

Bit test fileReg, skip if set

7

bsf, bcf and btg

- `bsf fileReg, bit`
 - e.g., `bsf PortB, 5` sets Bit 5 of Port B to be 1
- `bcf fileReg, bit`
 - e.g., `bcf PortB, 5` sets Bit 5 of Port B to be 0.
- `btg fileReg, bit`
 - e.g., `btg PortB, 5` toggles Bit 5 of Port B (i.e., sets it to 1 if the current value is 0 & vice versa)

8

Checking the state of an I/O port

- `btfs` (bit test file, skip if clear) and `btfss` (bit test file, skip if set) are *conditional skipping* instructions (Sec. 3.3) used to make branching decision based on the status of a given bit.
- e.g., `btfs PORTD, 2` skips the next instruction if Bit 2 of Port D equals 0.
- e.g., `btfss PORTD, 2` skips the next instruction if Bit 2 of Port D equals 1.

9

Example: `btfss`

- e.g., Write a program to (a) keep monitoring RB2 bit until it becomes high (b) When RB2 becomes high, write 45H to Port C

```
bsf TRISB, 2; set RB2 as input
clrf TRISC; set Port C as output
movlw 0x45
```

```
Again: btfss PORTB, 2
      bra Again
      movwf PORTC
```

10

Example: btfss

- e.g., Write a program to check RB2.
 - If RB2 = 0, send the letter 'N' to Port D
 - If RB2 = 1, send the letter 'Y' to Port D

```
        bsf TRISB, 2; set RB2 as input
        clrf TRISD; set Port D as output
Again:  btfss PORTB, 2
        bra Over
        movlw A'Y'
        movwf PORTD
        bra Again
Over:   movlw A'N'
        movwf PORTD
        bra Again
```

11

Example: btfsc

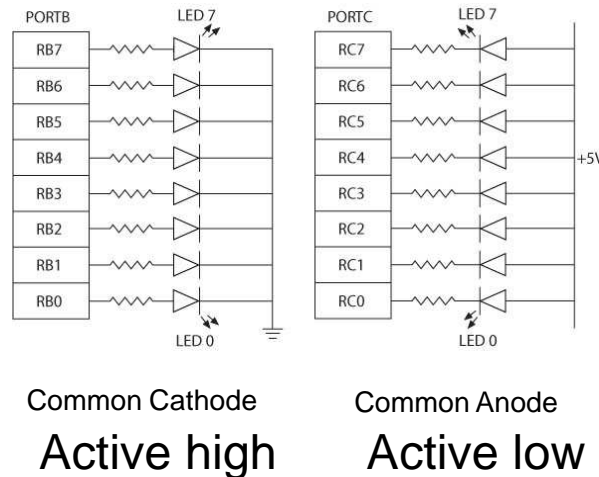
- e.g., Perform the same function using btfsc

```
        bsf TRISB, 2; set RB2 as input
        clrf TRISD; set Port D as output
Again:  btfsc PORTB, 2
        bra Over
        movlw A'N'
        movwf PORTD
        bra Again
Over:   movlw A'Y'
        movwf PORTD
        bra Again
```

12

Interfacing with LED

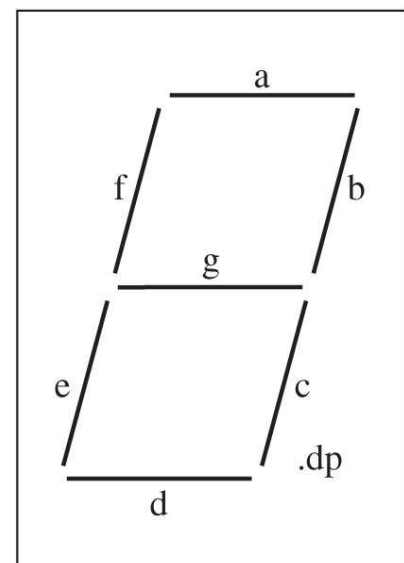
- Two ways of connecting LEDs to I/O ports:
 - **Common Cathode**: LED cathodes are grounded and logic 1 from the I/O port turns on the LEDs.
 - **Common Anode**: LED anodes are connected to the power supply and logic 0 from the I/O port turns on the LEDs.



13

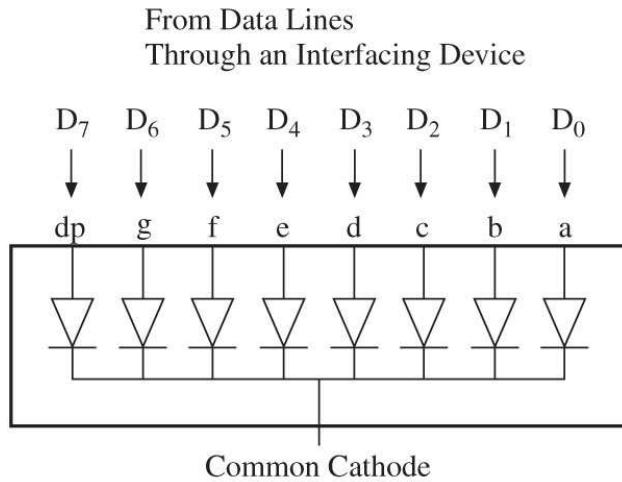
Interfacing with 7-Segment LED

- Used to display decimal numbers (0 through 9) and a few alphabets
- A group of eight LEDs physically mounted in the shape of the number eight plus a decimal point
- Each LED is called a segment and labeled as 'a' through 'g'.



14

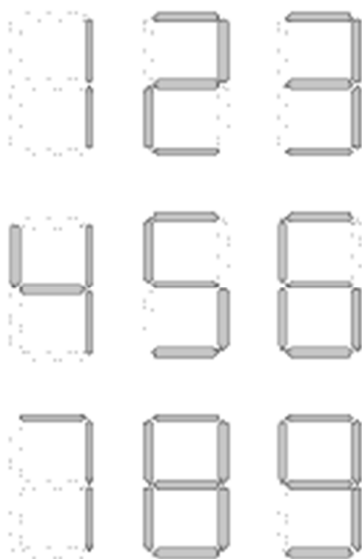
Interfacing with 7-Segment LED



- In a common cathode seven-segment LED
 - All cathodes are connected together to ground and the anodes are connected to data lines
- Logic 1 turns on a segment.
- Example: To display digit 1, all segments except b and c should be turned off.
- Byte 00000110 = 06 will display digit 1.

15

Interfacing with 7-Segment LED



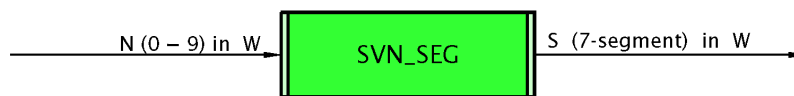
Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

16

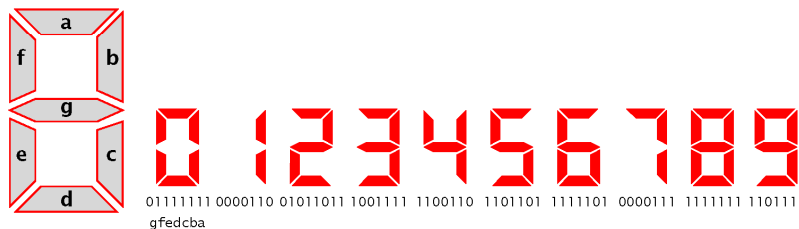
Applications: Mapping number to digit pattern to be displayed in LED

Use lookup table to implement a subroutine with the following input/output:

- Input: A number N ranging from 0 to 9 stored in WREG.
- Output: The 7-segment LED digit pattern corresponding to the input number.



(a) System view

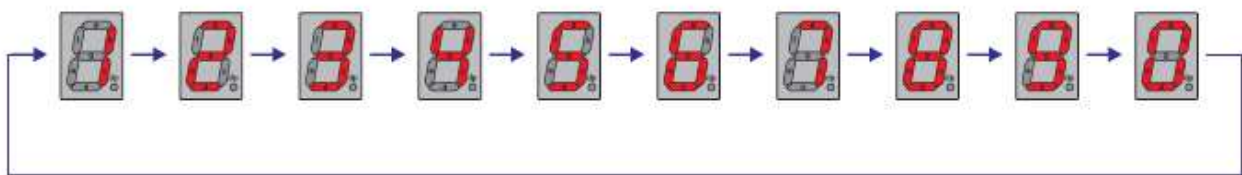


(b) The 7-segment font

Image courtesy of S. Katzen,
The essential PIC18
Microcontroller, Springer

17

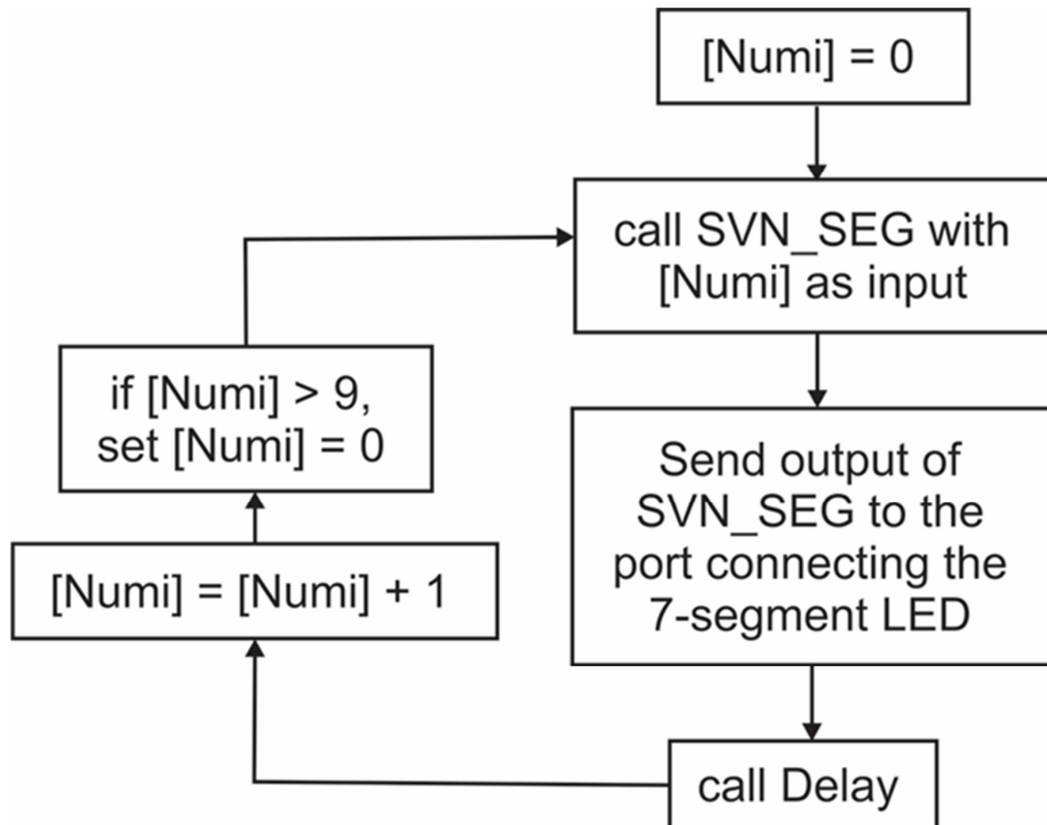
Lab 2 Task 1: Single digit 7-Segment LED



- Task: Write a loop to show digits 0-9 sequentially
- Before writing the loop, write 2 subroutines:
 - SVN_SEG: implement lookup table that maps number to digit pattern to be displayed in LED. Digit pattern is stored in WREG.
 - Delay: Each digit should stay displayed for a short time so that it is observable.
- 3 components of a loop
 - Initialization: [Numi] = 00 (number to be displayed)
 - Instructions to be repeated: (1) call SVN_SEG (2) Display digit pattern stored in WREG (3) call Delay (4) Increment [Numi]
 - This is an infinite loop and we do not need to track number of repetitions

18

Lab 2 Task 1: Single digit 7-Segment LED



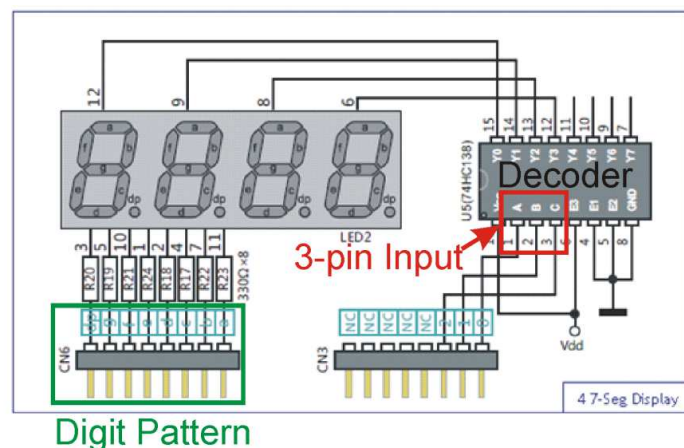
19

Interfacing with 4-digit 7-Segment LED

- Decoder selects the position where digit pattern is displayed.
- Use time multiplexing if we need to display all four digits.

Input 1: Digit Pattern

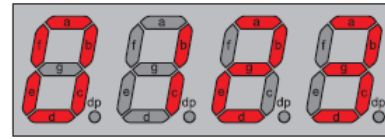
Input 2: Which position you want Input 1 to be displayed?



20

Lab 2 Task 2: 4-digit 7-Segment LED

Task: Display your group number of the 4-digit 7-segment LED.

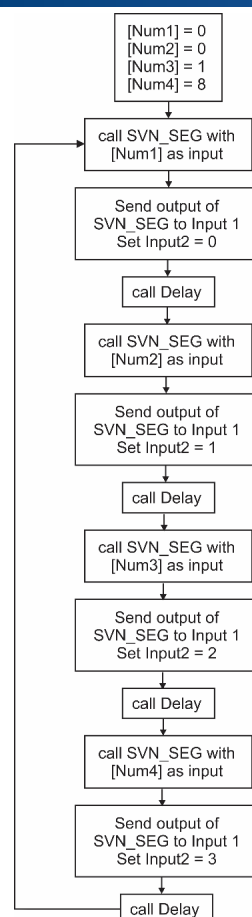


Algorithm

- Initialization
 - 4 file registers storing the 4 digits of your group number: Num1, Num2, ..., Num4
 - Loc: Which digit the number should be displayed?
- Loop: For the first digit, get the digit pattern corresponding to [Num1] by calling SVN_SEG → Digit pattern stored in [WREG]. Set Input1 = [WREG] and Input 2 = 0. Perform similar operations for the second, third and fourth digits.
- Repeat the loop infinitely.

21

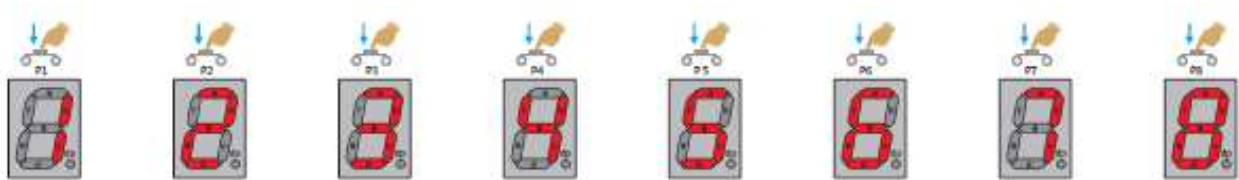
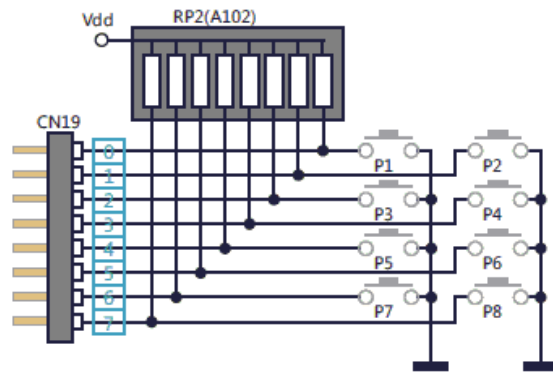
Lab 2 Task 2: 4-digit 7-Segment LED



22

Lab 3 Task 1: 8-button keypad

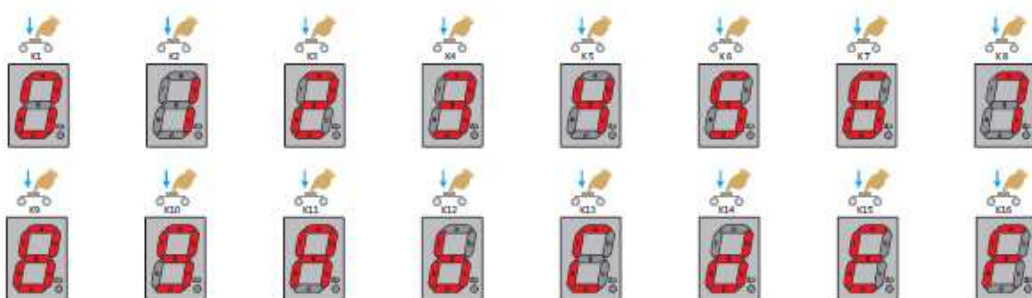
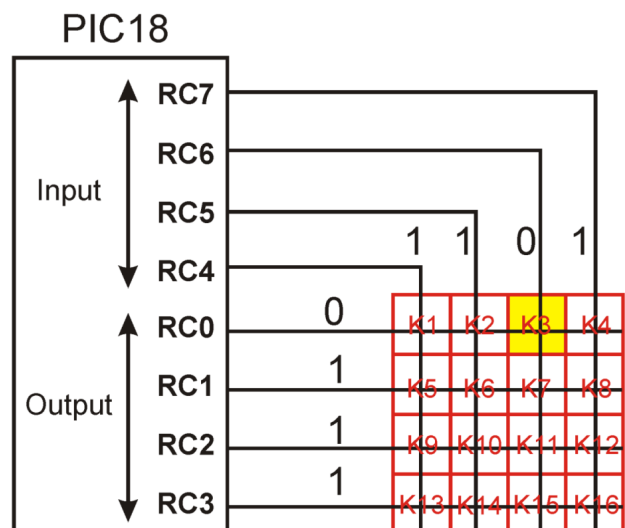
- Configuration of keypad: Pulled up (i.e., 1) when not pressed. 0 if pressed.
- Task: Show <X> when Button P<X> is pressed. <X>=1 to 8



23

Lab 3 Task 2: 4x4 Key Matrix

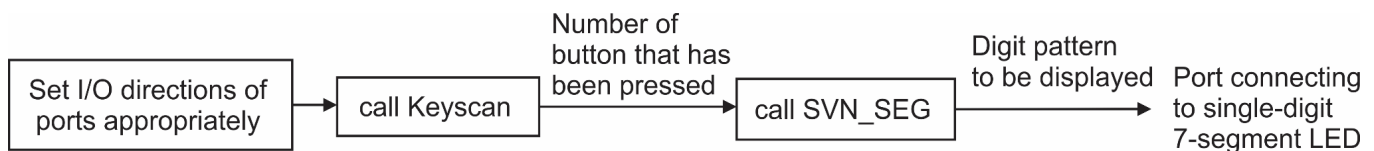
- Configuration of key matrix
 - Port C connects to keypad matrix
 - RC0-3 outputs, RC4-7 inputs
- Show <X> if Button K<X+1> is pressed where <X> = 0 to F.



24

Program Structure for Both Tasks

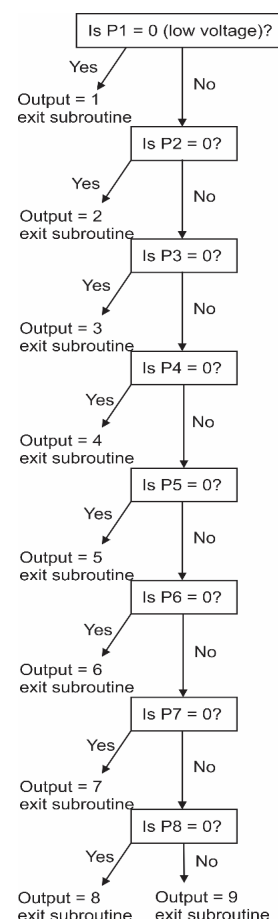
- Set I/O direction of appropriate ports.
- call Keyscan
 - Task 1: Output from 1 to 9, indicating the button that is pressed. 9 indicates no button is pressed.
 - Task 2: Output from 0 to 16, indicating the button that is pressed. 0 indicates that no button is pressed.
- call SVN_SEG
 - Input: Output from Keyscan
 - Output: Digit pattern corresponding to input value
 - Implemented using a lookup table
- Port connecting 1-digit 7-seg LED = Output from SVN_SEG



25

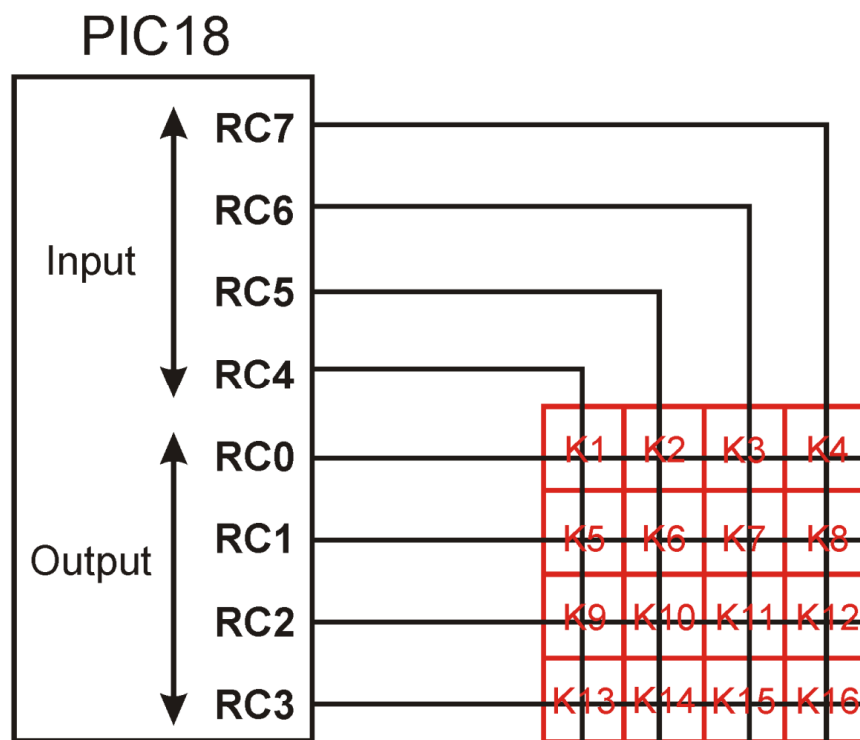
Keyscan subroutine for Task 1

- Scan P1 to P8 sequentially.
 - If a low voltage (i.e., logic 0) is detected, return the corresponding button number.
 - If a low voltage (i.e., logic 0) is not detected after scanning all buttons, return 9.



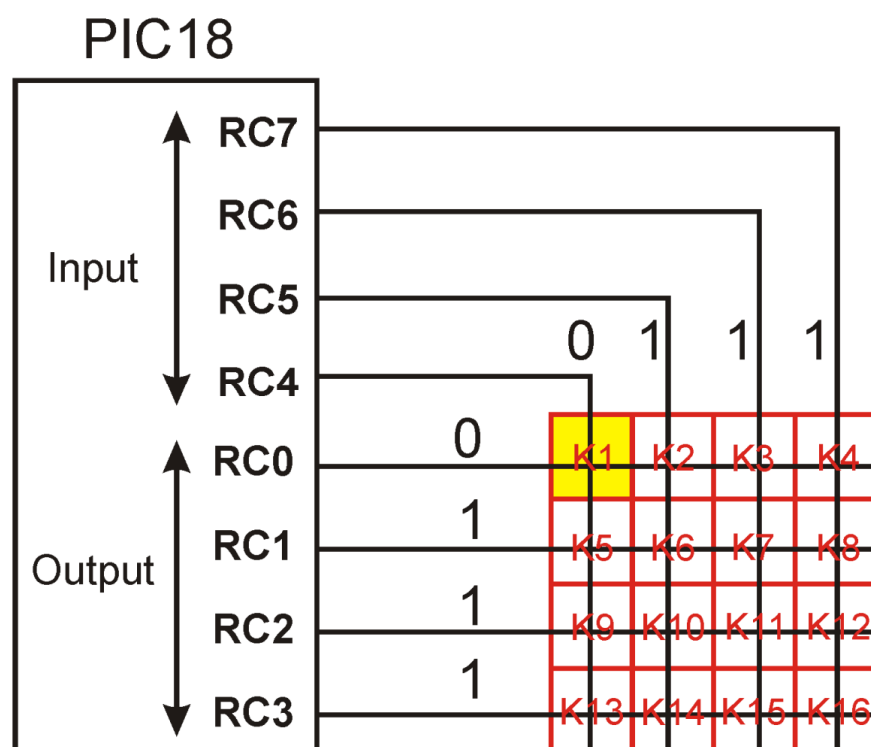
26

Task 2: Scanning 4x4 key matrix



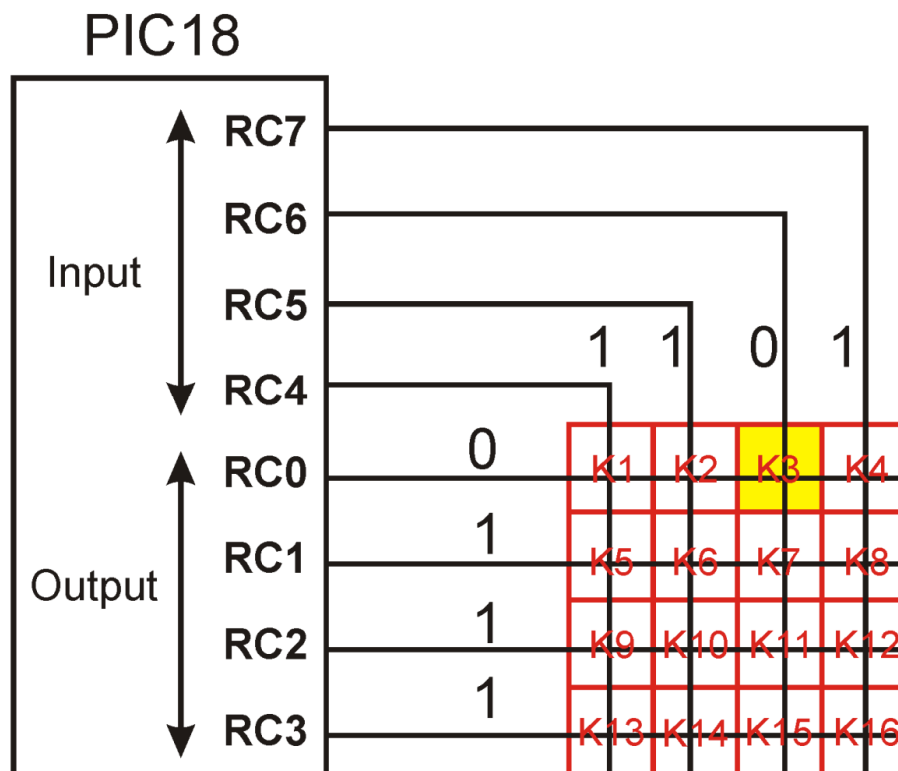
27

When K1 is pressed



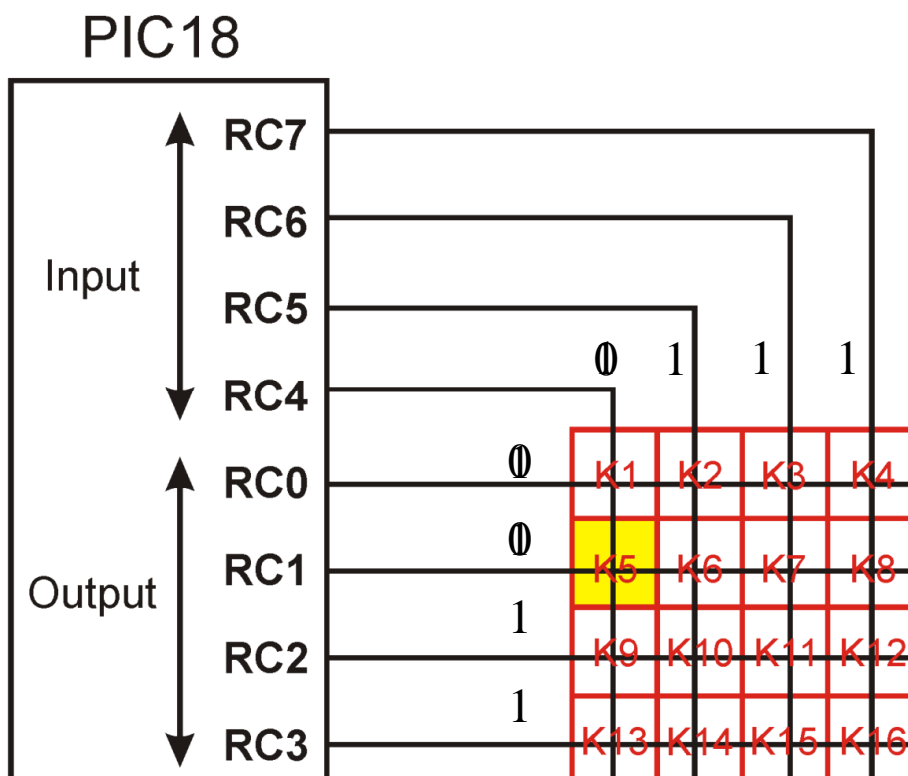
28

When K3 is pressed



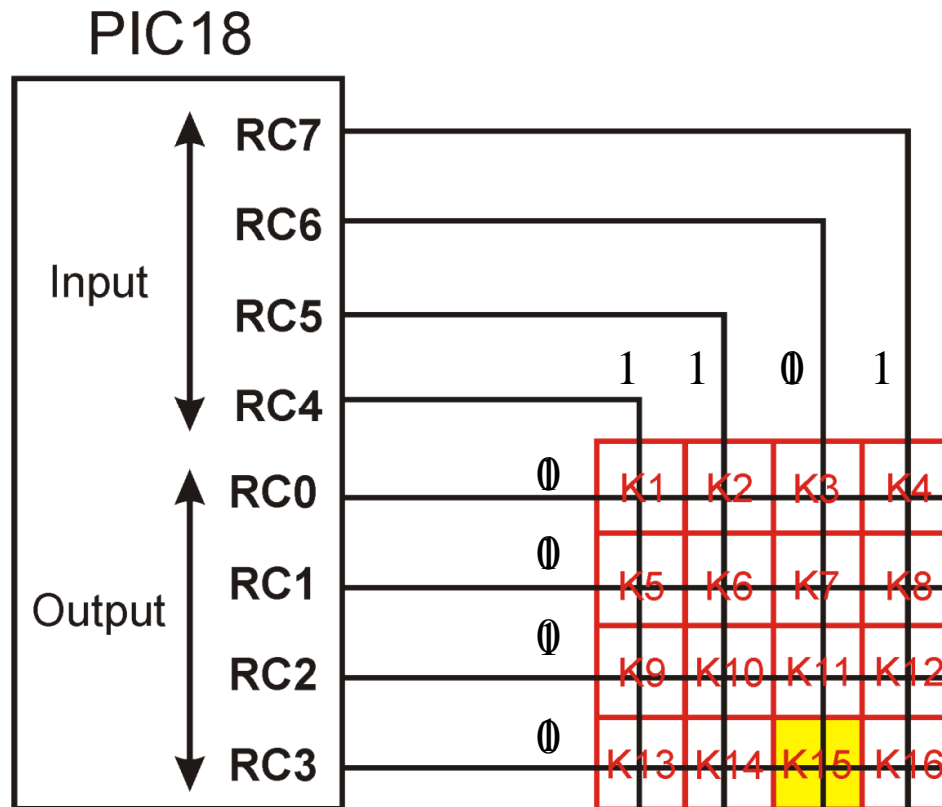
29

When K5 is pressed



30

When K15 is pressed



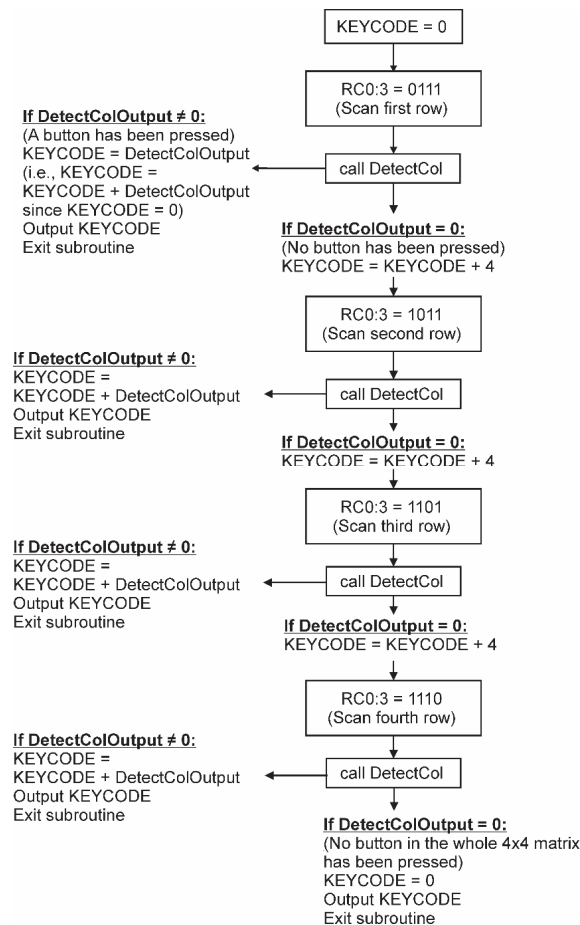
31

Keyscan subroutine for 4x4 key matrix

1. Initialize KEYCODE to 0.
2. Set RC0 to logic 0 and RC1:3 to logic 1
3. call DetectCol, read the states of RC4:7
 - 0111 → DetectColOutput = 1
 - 1011 → DetectColOutput = 2
 - 1101 → DetectColOutput = 3
 - 1110 → DetectColOutput = 4
 - 1111 → No button is pressed, DetectColOutput = 0
4. Two cases:
 - If DetectColOutput = 0, no button is pressed, scan next row. Add 4 to KEYCODE.
 - If DetectColOutput ≠ 0, a button has been pressed, return KEYCODE = KEYCODE + DetectColOutput
5. If no button is detected to be pressed after scanning four rows, KEYCODE = 0 and exit the subroutine.

32

Keyscan subroutine for 4x4 key matrix



33

You should be able to ...

- Code assembly language to use the ports for input or output
- Code I/O bit manipulation programs for the PIC
- Check the state of an I/O port and make branching decision based on it
- Interface with 1-digit/4-digit 7-segment LED and keypad matrix.
- Think about whether these components can be used in your mini-project

34