Input/Output Ports and Interfacing LCD & Seven Segment Display
Basic I/O Concepts

• Peripherals such as LEDs and keypads are essential components of microcontroller-based systems

• Input devices
  • Provide digital information to an MPU
  • Examples: switch, keyboard, scanner, and digital camera

• Output devices
  • Receive digital information from an MPU
  • Examples: LED, seven-segment display, LCD, and printer

• Devices are interfaced to an MPU using I/O ports
I/O Interfacing
Interfacing and Addressing

• I/O ports
  • Buffers and latches on the MCU chip
    • Assigned binary addresses by decoding the address bus
  • Generally bidirectional
    • Internal data direction registers
  • To read binary data from an input peripheral
    • MPU places the address of an input port on the address bus
    • Enables the input port by asserting the RD signal
    • Reads data using the data bus
  • To write binary data to an output peripheral
    • MPU places the address of an output port on the address bus
    • Places data on data bus
    • Asserts the WR signal to enable the output port
PIC18F452/4520 I/O Ports

• MCU includes five I/O ports
  • PORTA, PORTB, PORTC, PORTD, PORTE
• Ports are multiplexed
  • Can be set up to perform various functions
• Each I/O port is associated with several SFRs
  • PORT
    • Functions as a latch or a buffer
  • TRIS
    • Data direction register
    • Logic 0 sets up the pin as an output
    • Logic 1 sets up the pin as an input
  • LAT
    • Output latch similar to PORT
PIC18F452/4520 I/O Ports

PORTA: Example of Multiple Fns
- Digital I/O: RA6-RA0
- Analog Input: AN0-AN4
- $V_{REF}^+$: A/D Reference Plus $V$  
- $V_{REF}^-$: A/D Reference Minus $V$  
- TOCK1: Timer0 Ext. Clock
- SS: SPI Slave Select Input  
- LVDIN: Low V Detect Input
PIC18F452/4520 I/O Ports

- PORTB
I/O Example

- Write instructions to set up pins RB7-RB4 of PORTB as inputs and pins RB3-RB0 as outputs

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW</td>
<td>0xF0</td>
<td>;Load B’11110000’ into WREG</td>
</tr>
<tr>
<td>MOVWF</td>
<td>TRISB</td>
<td>;Set PORTB TRIS Reg</td>
</tr>
</tbody>
</table>
Interfacing Output Peripherals

• Commonly used output peripherals in embedded systems
  • LEDs
  • Seven-Segment Displays
  • LCDs
• Two ways of connecting LEDs to I/O ports
  • Common Cathode
    • LED cathodes are grounded
    • Logic 1 from the I/O port turns on the LEDs
    • Current is supplied by the I/O port called current sourcing
  • Common Anode
    • LED anodes are connected to the power supply
    • Logic 0 from the I/O port turns on the LEDs
    • Current is received by the chip called current sinking
Interfacing Output Peripherals

Common Cathode

Common Anode
Seven-Segment Display

- Seven-segment Displays
  - Used to display BCD digits
    - 0 thru 9
  - A group of 7 LEDs physically mounted in the shape of the number eight
    - Plus a decimal point
  - Each LED is called a segment
    - ‘a’ through ‘g’
  - Two types
    - Common anode
    - Common cathode
Seven-Segment Display

- **Common Anode**
  - All anodes are connected together to a power supply
  - Cathodes are connected to data lines
- Logic 0 turns on a segment
- **Example: To display the digit 1**
  - All segments except b and c should be off
  - $1111001 = \text{F9}_H$
Seven-Segment Display

- **Common Cathode**
  - All cathodes are connected together to ground
  - 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 off
  - $00000110 = 06_{\text{H}}$
Reading from an I/O Port

- Read input switches on PORTB (RB7-RB4)
  - RB0 set HI (1)
  - Switches Open = LOW (0)
  - Switches Closed = HIGH (1)
- Display on PORTC

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW</td>
<td>0xF0</td>
<td>;Load B’11110000’ into WREG</td>
</tr>
<tr>
<td>MOVWF</td>
<td>TRISB</td>
<td>;Set PORTB TRIS Reg</td>
</tr>
<tr>
<td>CLRF</td>
<td>TRISC</td>
<td>;Set PORTC as Output</td>
</tr>
<tr>
<td>BSF</td>
<td>PORTB,0</td>
<td>;Set RB0 High</td>
</tr>
<tr>
<td>MOVF</td>
<td>PORTB,W</td>
<td>;Read PORTB</td>
</tr>
<tr>
<td>MOVWF</td>
<td>PORTC</td>
<td>;Display on PORTC</td>
</tr>
</tbody>
</table>
Internal Pull-Up Resistor

- Turning off the internal FET provides a pull-up resistor
- Bit7 (RBPU) in the INTCON2 register enables or disables the pull-up resistor
  - Instruction to Enable Pull Up Resistors:
    BCF INTCON2,7

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{RBPU} = \text{PORTB pull-up resistor enable bit} \]
\[ 0 = \text{Pull-up resistors are enabled} \]
\[ 1 = \text{Pull-up resistors are disabled} \]
Interfacing Push-Button Keys

• When a key is pressed (or released), mechanical metal contact bounces momentarily and can be read as multiple inputs

• Key debounce
  • Eliminating reading of one contact as multiple inputs
  • Hardware or Software

![Diagram of push-button interface]
Interfacing a Matrix Keypad

- **Hardware (PIC18 Simulator)**
  - 4 x 4 matrix keypad organized in the row and column format
  - Four columns are connected to the lower half of PORTB (RB0-RB3)
  - Four rows are connected to upper half of PORTB (RB4-RB7)
  - When a key is pressed, it makes a contact with the corresponding row and column
Interfacing a Matrix Keypad

- PIC18 Simulator Keypad Matrix
Interfacing a Matrix Keypad

- **Software**
  - To recognize and encode the key pressed
    - Set all the columns High by sending ones
    - Check for any key pressed (non-zero)
    - Set one column High at a time
      - Check all the rows in that column
    - Once a key is identified
      - Encode based on its position in the column
Interfacing LCD

- Problem statement
  - Interface a 2-line x 20 character LCD module with the built-in HD44780 controller to I/O ports of the PIC18 microcontroller.
  - Explain the control signals necessary to read from and write to the LCD.
  - Write a program to display ASCII characters.
Interfacing LCD

- **Hardware**
  - 20 x 2-line LCD display
    - Two lines with 20 characters per line
  - LCD has a display Data RAM
    - Stores data in 8-bit character code
  - Each register in Data RAM has its own address
    - Corresponds to its position on the line
      - Line 1 is $00_{\text{H}}$ to $13_{\text{H}}$
      - Line 2 is $40_{\text{H}}$ to $53_{\text{H}}$
Interfacing LCD

![Diagram of an HD 44780 Controller interfaced with a PIC18F 452 / 4520 microcontroller. The diagram shows signals for RD7 to RD0, RA3 to RA1, and connections to VDD, VO, VSS, and +5V.]
Interfacing LCD

• Driver HD44780
  • 8-bit data bus (RD7-RD0)
  • Three control signals
    • RS – Register Select (RA3)
    • R/W – Read/Write (RA2)
    • E – Enable (RA1)
  • Three power connections
    • Power, ground, and variable resistor to control brightness
Interfacing LCD

- Can be interfaced either in 8-bit mode or 4-bit mode
  - In 8-bit mode, all eight data lines are connected
  - In 4-bit mode, only four data lines are connected
    - Two transfers per character (or instruction) are needed
- Driver has two 8-bit internal registers
  - Instruction Register (IR) to write instructions to set up LCD
    - Table 9-3
  - Data Register (DR) to write data (ASCII characters)
Interfacing LCD

• LCD Operation
  • When the MPU writes an instruction to IR or data to DR, the controller:
    • Sets DB7 high indicating that the controller is busy
    • Sets DB7 low after the completion of the operation
  • The MPU should always check whether DB7 is low before sending an instruction or a data byte
Interfacing LCD

• Writing to or Reading from LCD (Table 9-4)
  • The MPU:
    • Asserts RS low to select IR
    • Asserts RS high to select DR
    • Reads from LCD by asserting the R/W signal high
    • Writes into LCD by asserting the R/W signal low
    • Asserts the E signal high and then low (toggles) to latch a data byte or an instruction
Interfacing LCD

• Software
  • To write into the LCD
    • Send the initial instructions to set up the LCD
      • 4-bit or 8-bit mode
    • Continue to check DB7 until it goes low
    • Write instructions to IR to set up LCD parameters
      • Number of display lines and cursor status
    • Write data to display a message
I/O devices (Peripherals)

• Examples: switches, LED, LCD, printers, keyboard, keypad
• **Interface** chips
  • are needed to resolve the speed problem
  • *synchronizes* data transfer between CPU and I/O device
• Connection of Interface and CPU
  • Data pins are connected to CPU data bus
  • I/O port pins are connected to I/O device
• CPU may be connected to **multiple** interface
• IO ports are simplest interface
I/O Interfacing

• Dedicated instructions for IO operations (Isolated I/O)

• same instruction for memory and IO (memory-mapped I/O)

• MCS-51 (8051) is memory mapped
Synchronization of CPU and interface chip

• To make sure that there are valid data in the interface
• two ways
  • **Polling** method: Read status bit - Simple method
  • **Interrupt** driven method: interface interrupts the CPU when it has new data - CPU executes the ISR
Synchronization of CPU and interface chip

• Output synchronization: two ways of doing this

1. Polling method
   • Interface chip uses a status bit to indicate that the data register is empty
   • CPU keeps checking status bit until it is set, and then writes data into interface chip

2. Interrupt driven method: interface chip interrupts the CPU when it data register is empty. CPU executes the ISR
8051 - Switch On I/O Ports

- **Case-1:**
  - Gives a logic 0 on switch close
  - Current is 0.5ma on switch close

- **Case-2:**
  - Gives a logic 1 on switch close
  - High current on switch close

- **Case-3:**
  - Can damage port if 0 is output
Simple input devices

- DIP switches usually have 8 switches
- Use the case-1 from previous page
- Sequence of instructions to read is:

\[
\begin{align*}
\text{MOV} & \quad \text{P1,} \#\text{FFH} \\
\text{MOV} & \quad \text{A,} \text{P1,}
\end{align*}
\]
Bouncing contacts

• Contact:
  • Push-button switches
  • Toggle switches
  • Electromechanical relays
• Make and break Contact normally open switch
• The effect is called "contact bounce" or, in a switch, "switch bounce".

• If used as edge-triggered input (as INT0), several interrupt is accorded
Hardware Solution

- An RC time constant to suppress the bounce
- The time constant has to be larger than the switch bounce
Hardware Solution

FIGURE 4–7
Buzzer example

Buzzer
0 = "zzzzz..."
1 = silent
Software Solution

- Read the new state of switch N time

- Wait-and-see technique
  - When the input drops
  - an "appropriate" delay is executed (10 ms)
  - then the value of the line is checked again to make sure the line has stopped bouncing
Interfacing a Keypad

16 keys arranged as a 4X4 matrix

- Place a 0 on R0 port
- Read C port
- If there is a 0 bit then the button at the column/row intersection has been pressed.
- Otherwise, try next row
- Repeat constantly
Interfacing a 7-segment display

- A **resistor** will be **needed** to control the current
- This leaves two possibilities:
  - Case 2 would be more appropriate
  - Case 1 will produce different **brightness** depending on the number of LEDs turned on.
Use of current buffer

- Interfacing to a DIP switch and 7-segment display
- Output a '1' to ON a segment
- We can use 74244 to common cathode 7_seg

![Diagram](image-url)
LCD Interfacing

• Liquid Crystal Displays (LCDs)
• cheap and easy way to display text
• Various configurations (1 line by 20 X char up to 8 lines X 80)
• Integrated controller
• The display has two register
  • command register
  • data register
• By RS you can select register
• Data lines (DB7-DB0) used to transfer data and commands
Alphanumeric LCD Interfacing

- **Pinout**
  - 8 data pins D7:D0
  - **RS**: Data or Command Register Select
  - **R/W**: Read or Write
  - **E**: Enable (Latch data)

- **RS – Register Select**
  - RS = 0 → Command Register
  - RS = 1 → Data Register

- **R/W = 0 → Write , R/W = 1 → Read**

- **E – Enable**
  - Used to latch the data present on the data pins.

- **D0 – D7**
  - Bi-directional data/command pins.
  - Alphanumeric characters are sent in ASCII format.
LCD Commands

- The LCD’s internal controller can accept several commands and modify the display accordingly. Such as:
  - Clear screen
  - Return home
  - Decrement/Increment cursor

- After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data.
  - We need to insert time delay between any two commands or data sent to LCD
# Pin Description

## Table 4-7: Pin Descriptions for LCD

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VSS</td>
<td>--</td>
<td>Ground</td>
</tr>
<tr>
<td>2</td>
<td>VCC</td>
<td>--</td>
<td>+5V power supply</td>
</tr>
<tr>
<td>3</td>
<td>VEE</td>
<td>--</td>
<td>Power supply source to control contrast</td>
</tr>
<tr>
<td>4</td>
<td>RS</td>
<td>I</td>
<td>Register select: RS=0 to select instruction command register, RS = 1 to select data register</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
<td>I</td>
<td>Read/write: R/W=0 for write, R/W=1 for read</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>I</td>
<td>Enable</td>
</tr>
<tr>
<td>7</td>
<td>DB0</td>
<td>I/O</td>
<td>The 8-bit data bus</td>
</tr>
<tr>
<td>8</td>
<td>DB1</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>9</td>
<td>DB2</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>10</td>
<td>DB3</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>11</td>
<td>DB4</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>12</td>
<td>DB5</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>13</td>
<td>DB6</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
<tr>
<td>14</td>
<td>DB7</td>
<td>I/O</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

Figure 4-34. Pin Positions for Various LCDs from Optrex
**Command Codes**

<table>
<thead>
<tr>
<th>Code (hex)</th>
<th>Command to LCD Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clear display screen</td>
</tr>
<tr>
<td>2</td>
<td>Return home</td>
</tr>
<tr>
<td>4</td>
<td>Decrement cursor (shift cursor to left)</td>
</tr>
<tr>
<td>6</td>
<td>Increment cursor (shift cursor to right)</td>
</tr>
<tr>
<td>5</td>
<td>Shift display right</td>
</tr>
<tr>
<td>7</td>
<td>Shift display left</td>
</tr>
<tr>
<td>8</td>
<td>Display off, cursor off</td>
</tr>
<tr>
<td>A</td>
<td>Display off, cursor on</td>
</tr>
<tr>
<td>C</td>
<td>Display on, cursor off</td>
</tr>
<tr>
<td>E</td>
<td>Display on, cursor on</td>
</tr>
<tr>
<td>F</td>
<td>Display on, cursor blinking</td>
</tr>
<tr>
<td>10</td>
<td>Shift cursor position to left</td>
</tr>
<tr>
<td>14</td>
<td>Shift cursor position to right</td>
</tr>
<tr>
<td>18</td>
<td>Shift the entire display to the left</td>
</tr>
<tr>
<td>1C</td>
<td>Shift the entire display to the right</td>
</tr>
<tr>
<td>C0</td>
<td>Force cursor to beginning of 2nd line</td>
</tr>
<tr>
<td>38</td>
<td>2 lines and 5x7 matrix</td>
</tr>
</tbody>
</table>

*Note: This table is extracted from Table 4-10.*
LCD Addressing

Figure 4-36. Cursor Addresses for Some LCDs

Table 4-9: LCD Addressing

<table>
<thead>
<tr>
<th></th>
<th>DB7</th>
<th>DB6</th>
<th>DB5</th>
<th>DB4</th>
<th>DB3</th>
<th>DB2</th>
<th>DB1</th>
<th>DB0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 1 (min)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Line 1 (max)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Line 2 (min)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Line 2 (max)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: All data is in hex.
**LCD Timing**

![Diagram of LCD Timing]

- **tPWH** = Enable pulse width = 450 ns (minimum)
- **tDSW** = Data set up time = 195 ns (minimum)
- **tH** = Data hold time = 10 ns (minimum)
- **tAS** = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)
- **tAH** = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

*Figure 4-37. LCD Timing*
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0</td>
<td>Clears entire display and sets DD RAM address 0 in address counter.</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Return Home</td>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Entry Mode Set</td>
<td>0 0 0 0 0 0 0 1 I/D S</td>
<td>Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Display ON/OFF Control</td>
<td>0 0 0 0 0 1 D C B</td>
<td>Sets ON/OFF of entire display (D), cursor ON/OFF (C), and blink of cursor position character (b).</td>
<td>40 µs</td>
</tr>
<tr>
<td>Cursor or Display Shift</td>
<td>0 0 0 0 0 1 S/C R/L</td>
<td>Moves cursor and shifts display without changing DD RAM contents.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Function Set</td>
<td>0 0 0 0 1 DL N F</td>
<td>Sets interface data length (DL), number of display lines (L) and character font (F).</td>
<td>40 µs</td>
</tr>
<tr>
<td>Set CG RAM Address</td>
<td>0 0 0 1 AGC</td>
<td>Sets CG RAM address. CG RAM data is sent and received after this setting.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Set DD RAM Address</td>
<td>0 0 1 ADD</td>
<td>Sets DD RAM address. DD RAM data is sent and received after this setting.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Read Busy Flag &amp; Address</td>
<td>0 1 BF AC</td>
<td>Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Write Data to CG or DD RAM</td>
<td>1 0 Write Data</td>
<td>Writes data into DD RAM or CG RAM.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Read Data from CG or DD RAM</td>
<td>1 1 Read Data</td>
<td>Reads data from DD RAM or CG RAM.</td>
<td>40 µs</td>
</tr>
</tbody>
</table>
Interfacing LCD with 8051

8051

P3.4

P3.5

P3.3

P1.7-P1.0

LM015

RW

E

RS

D7-D0
mov A, command
call cmd
delay
mov A, another_cmd
call cmd
delay
mov A, #'A'
call data
delay
mov A, #'B'
call data
delay
...

Command and Data Write Routines

data:mov P1, A
    ;A is ascii data
    setb P3.3
    ;RS=1 data
    clr P3.4
    ;RW=0 for write
    setb P3.5
    ;H->L pulse on E
    clr P3.5
ret

cmd:mov P1, A
    ;A has the cmd word
    clr P3.3
    ;RS=0 for cmd
    clr P3.4
    ;RW=0 for write
    setb P3.5
    ;H->L pulse on E
    clr P3.5
ret
Example

Figure 4-35. 8255-to-PC Interface Connection to LCD
8255 Usage: Simple Example

- 8255 memory mapped to 8051 at address C000H base
  - A = C000H, B = C001H, C = C002H, CR = C003H
- Control word for all ports as outputs in mode0
  - CR : 1000 0000b = 80H

```
test:
    mov    A, #80H ; control word
    mov    DPTR, #C003H ; address of CR
    movx   @DPTR, A ; write control word
    mov    A, #55h ; will try to write 55 and AA
    repeat:
        mov    DPTR, #C000H ; address of PA
        movx   @DPTR, A ; write 55H to PA
        inc    DPTR ; now DPTR points to PB
        movx   @DPTR, A ; write 55H to PB
        inc    DPTR ; now DPTR points to PC
        movx   @DPTR, A ; write 55H to PC
        cpl    A ; toggle A (55→AA, AA→55)
        acall  MY_DELAY ; small delay subroutine
        sjmp   repeat ; for (1)
```
Interfacing Keyboard and Display Devices

Topics Covered:

• Interface switches and keyboard to the 8051
• Interface LED displays to the 8051
• Overcome Keybounce and multiple key press problems
• Design a microcontroller based system with keyboard and display devices
• Interface and program the LCD controller
Some 8-bit Registers of the 8051
A: Accumulator
B: Used specially in MUL/DIV
R0-R7: GPRs

<table>
<thead>
<tr>
<th>Bit No.</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>CY</td>
<td>PSW</td>
</tr>
<tr>
<td>6</td>
<td>AC</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>F0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RS1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RS0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OV</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
</table>
| CY  | Carry Flag
     | Used by arithmetic and conditional branch instruction. |
| AC  | Auxiliary Carry Flag
     | Used by instructions which execute BCD operations. |
| F0  | General Purpose Flag |
| RS1 | Register Bank select control bits
     | These bits are used to select one of the four register banks. |
| RS0 | 

<table>
<thead>
<tr>
<th>RS1</th>
<th>RS0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Registerbank 0 at data address 00H - 07H selected</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Registerbank 1 at data address 08H - 0FH selected</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Registerbank 2 at data address 10H - 17H selected</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Registerbank 3 at data address 18H - 1FH selected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
</table>
| OV  | Overflow Flag
     | Used by arithmetic instruction. |
| F1  | General Purpose Flag |
| P   | Parity Flag
     | Always set/cleared by hardware to indicate an odd/even number of “one” bits in the accumulator. |
8051 Programming using Assembly
The MOV Instruction – Addressing Modes

**MOV dest,source** ; dest = source

- **MOV A,#72H** ; A=72H
- **MOV A, #'r'** ; A='r' OR 72H
- **MOV R4,#62H** ; R4=62H
- **MOV B,0F9H** ; B=the content of F9’th byte of RAM
- **MOV DPL,#34H**
- **MOV DPH,#76H**
- **MOV DPTR,#7634H**
- **MOV P1,A** ; mov A to port 1

**Note 1:**

MOV A,#72H ≠ MOV A,72H

After instruction “MOV A,72H” the content of 72’th byte of RAM will replace in Accumulator.

**8086**

- MOV AL,72H
- MOV AL,’r’
- MOV BX,72H
- MOV AL,[BX]

**8051**

- MOV A,#72H
- MOV A,’r’
- MOV A,72H

**Note 2:**

MOV A,R3 ≡ MOV A,3
Arithmetic Instructions

ADD A, Source ; A = A + SOURCE

ADD A, #6 ; A = A + 6
ADD A, R6 ; A = A + R6
ADD A, 6 ; A = A + [6]  or  A = A + R6
ADD A, 0F3H ; A = A + [0F3H]
Set and Clear Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETB bit</td>
<td>bit=1</td>
</tr>
<tr>
<td>CLR bit</td>
<td>bit=0</td>
</tr>
</tbody>
</table>

- SETB C ; CY=1
- SETB P0.0 ; bit 0 from port 0 =1
- SETB P3.7 ; bit 7 from port 3 =1
- SETB ACC.2 ; bit 2 from ACCUMULATOR =1
- SETB 05 ; set high D5 of RAM loc. 20h

Note:

CLR instruction is as same as SETB
i.e:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>; CY=0</td>
</tr>
</tbody>
</table>

But following instruction is only for CLR:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR A</td>
<td>; A=0</td>
</tr>
</tbody>
</table>
SUBB A, source ; A = A - source - CY

SETBC ; CY = 1

SUBB A, R5 ; A = A - R5 - 1

ADC A, source ; A = A + source + CY

SETBC ; CY = 1

ADC A, R5 ; A = A + R5 + 1
# 8051 Flag bits and the PSW register

- **PSW Register**

<table>
<thead>
<tr>
<th>CY</th>
<th>AC</th>
<th>F0</th>
<th>RS1</th>
<th>RS0</th>
<th>OV</th>
<th>--</th>
<th>P</th>
</tr>
</thead>
</table>

- **Carry flag**
- **Auxiliary carry flag**
- **Available to the user for general purpose**
- **Register Bank selector bit 1**
- **Register Bank selector bit 0**
- **Overflow flag**
- **User define bit**
- **Parity flag Set/Reset odd/even parity**

<table>
<thead>
<tr>
<th>RS1</th>
<th>RS0</th>
<th>Register Bank</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00H-07H</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>08H-0FH</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10H-17H</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>18H-1FH</td>
</tr>
</tbody>
</table>
Instructions that Affect Flag Bits:

<table>
<thead>
<tr>
<th>Instructions</th>
<th>CY</th>
<th>OV</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ADDC</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SUBB</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MUL</td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DA</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRC</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLC</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETB C</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANL C,bit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANL C,/bit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORL C,bit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV C,bit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CJNE</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: X can be 0 or 1
Example:
MOV   A,#88H
ADD   A,#93H

88         10001000
+93         +10010011
----         00011011
CY=1   AC=0   P=0

Example:
MOV   A,#38H
ADD   A,#2FH

38         00111000
+2F         +00101111
----         01100111
CY=0   AC=1   P=1

Example:
MOV   A,#9CH
ADD   A,#64H

9C         10011100
+64         +01100100
----         00000000
CY=1   AC=1   P=0
Addressing Modes

- Immediate
- Register
- Direct
- Register Indirect
- Indexed
Immediate Addressing Mode

MOV A,#65H
MOV A,#’A’
MOV R6,#65H
MOV DPTR,#2343H
MOV P1,#65H

Example :

<table>
<thead>
<tr>
<th></th>
<th>EQU</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

data1: db "Example"
Example

• Write the decimal value 4 on the SSD in the following figure. Switch the decimal point off.
Direct Addressing Mode

Although the entire of 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM loc. 30 – 7FH.

MOV    R0, 40H
MOV    56H, A
MOV    A, 4    ; ≡ MOV A, R4
MOV    6, 2    ; copy R2 to R6
             ; MOV R6,R2 is invalid !

SFR register and their address

MOV    0E0H, #66H    ; ≡ MOV A,#66H
MOV    0F0H, R2     ; ≡ MOV B, R2
MOV    80H,A        ; ≡ MOV P1,A
Register Indirect Addressing Mode

• In this mode, register is used as a pointer to the data.

MOV     A, @Ri ; move content of RAM loc. Where address is held by Ri into A
           (i=0 or 1)
MOV     @R1, B

In other word, the content of register R0 or R1 is sources or target in MOV, ADD and SUBB
instructions.
Example:
Write a program to copy a block of 10 bytes from RAM location starting at 37h to RAM
location starting at 59h.

Solution:
  MOV     R0, 37h ; source pointer
  MOV     R1, 59h ; dest pointer
  MOV     R2, 10 ; counter

L1:  MOV     A, @R0
     MOV     @R1, A
     INC     R0
     INC     R1
     DJNZ    R2, L1

jump
Hardware Structure of I/O Pin

- Each pin of I/O ports
  - Internal CPU bus: communicate with CPU
  - A D latch store the value of this pin
    - D latch is controlled by “Write to latch”
      - Write to latch = 1: write data into the D latch
  - 2 Tri-state buffer: "?
    - TB1: controlled by “Read pin”
      - Read pin = 1: really read the data present at the pin
    - TB2: controlled by “Read latch”
      - Read latch = 1: read value from internal latch
- A transistor M1 gate
  - Gate=0: open
  - Gate=1: close
Tri-state Buffer

Tri-state control (active high)

L     L
H     H
H     H

Low

High impedance (open-circuit)
Writing “1” to Output Pin P1.X

1. write a 1 to the pin

2. output pin is Vcc

- Internal CPU bus
- Write to latch
- Read latch
- Read pin

8051 IC
Writing “0” to Output Pin P1.X

1. write a 0 to the pin
2. output pin is ground

Read latch

Internal CPU bus
Write to latch

Read pin

8051 IC

P1.X pin output 0
Port 1 as Output （Write to a Port）

- Send data to Port 1:
  
  ```
  MOV A, #55H
  BACK: MOV P1, A
  ACALL DELAY
  CPL A
  SJMP BACK
  ```

- Let P1 toggle.
- You can write to P1 directly.
Reading Input v.s. Port Latch

- When reading ports, there are two possibilities:
  - Read the status of the input pin. (from \textit{external pin value})
    - MOV A, PX
    - JNB P2.1, TARGET ; jump if P2.1 is not set
    - JB P2.1, TARGET ; jump if P2.1 is set
    - Figures C-11, C-12
  - Read the \textit{internal latch} of the output port.
    - ANL P1, A ; P1 ← P1 AND A
    - ORL P1, A ; P1 ← P1 OR A
    - INC P1 ; increase P1
    - Figure C-17
    - Table C-6 Read-Modify-Write Instruction (or Table 8-5)
  - See Section 8.3
Reading “High” at Input Pin

1. Write a 1 to the pin MOV P1,#0FFH

2. MOV A,P1
   external pin=High

3. Read pin=1 Read latch=0
   Write to latch=1

8051 IC
Reading “Low” at Input Pin

1. Write a 1 to the pin
   MOV P1,#0FFH

2. MOV A,P1
   external pin=Low

3. Read pin=1 Read latch=0
   Write to latch=1

8051 IC
Port 1 as Input (Read from Port)

- In order to make P1 an input, the port must be programmed by writing 1 to all the bits.

```
MOV  A,#0FFH ;A=11111111B
MOV  P1,A ;make P1 an input port
BACK: MOV  A,P1 ;get data from P0
         MOV  P2,A ;send data to P2
         SJMP  BACK
```

- To be an input port, P0, P1, P2 and P3 have similar methods.
## Instructions For Reading an Input Port

- Following are instructions for reading external pins of ports:

<table>
<thead>
<tr>
<th>Mnemonics</th>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,PX</td>
<td>MOV A,P2</td>
<td>Bring into A the data at P2 pins</td>
</tr>
<tr>
<td>JNB PX.Y,..</td>
<td>JNB P2.1,TARGET</td>
<td>Jump if pin P2.1 is low</td>
</tr>
<tr>
<td>JB PX.Y,..</td>
<td>JB P1.3,TARGET</td>
<td>Jump if pin P1.3 is high</td>
</tr>
<tr>
<td>MOV C,PX.Y</td>
<td>MOV C,P2.4</td>
<td>Copy status of pin P2.4 to CY</td>
</tr>
</tbody>
</table>
Read-modify-write Feature

• Read-modify-write Instructions
  • Table C-6

• This feature combines 3 actions in a single instruction:
  1. CPU reads the latch of the port
  2. CPU perform the operation
  3. Modifying the latch
  4. Writing to the pin
  • Note that 8 pins of P1 work independently.
Port 1 as Input (Read from latch)

- Exclusive-or the Port 1:

```assembly
MOV P1,#55H ; P1 = 01010101
AGAIN: XOR P1,#0FFH ; complement
        ACALL DELAY
        SJMP AGAIN
```

- Note that the XOR of 55H and FFH gives AAH.
- XOR of AAH and FFH gives 55H.
- The instruction read the data in the latch (not from the pin).
- The instruction result will put into the latch and the pin.
Port 0 with Pull-Up Resistors

- Vcc
- Port 0
- 10 K
- Pull-Up Resistors
- P0.0
- P0.1
- P0.2
- P0.3
- P0.4
- P0.5
- P0.6
- P0.7
- DS5000
- 8751
- 8951

Vcc is connected to Port 0 through a series of pull-up resistors (10 KΩ).
8051 Programming Using C
Programming microcontrollers using high-level languages

- Most programs can be written exclusively using high-level code like ANSI C
- Extensions
  - To achieve low-level (Assembly) efficiency, extensions to high-level languages are required
- Restrictions
  - Depending on the compiler, some restrictions to the high-level language may apply
Keil C keywords

• **data/idata:**
  Description: The variable will be stored in internal data memory of controller.
  
  example:
  unsigned char data x;
  //or
  unsigned char idata y;

• **bdata:**
  Description: The variable will be stored in bit addressable memory of controller.
  
  example:
  • unsigned char bdata x;
  //each bit of the variable x can be accessed as follows
  x ^ 1 = 1; //1st bit of variable x is set
  x ^ 0 = 0; //0th bit of variable x is cleared

• **xdata:**
  Description: The variable will be stored in external RAM memory of controller.
  
  example:
  unsigned char xdata x;
Keil C keywords

- **sfr:**
  Description: sfr is used to define an 8-bit special function register from sfr memory.

  ```c
  example:
  sfr Port1 = 0x90;
  // Special function register with name Port1 defined at address 0x90
  ```

- **sfr16:**
  Description: This keyword is used to define a two sequential 8-bit registers in SFR memory.

  ```c
  example:
  sfr16 DPTR = 0x82;
  // 16-bit special function register starting at 0x82
  // DPL at 0x82, DPH at 0x83
  ```

- **using:**
  Description: This keyword is used to define register bank for a function. User can specify register bank 0 to 3.

  ```c
  example:
  void function () using 2{
  // code
  }
  // Function named "function" uses register bank 2 while executing its code
  ```

- **Interrupt:**
  Description: defines interrupt service routine

  ```c
  void External_Int0() interrupt 0{
  // code
  }
  ```
Data Converters

• Analog to Digital Converters (ADC)
  – Convert an analog quantity (voltage, current) into a digital code

• Digital to Analog Converters (DAC)
  – Convert a digital code into an analog quantity (voltage, current)

Dr. Konstantinos Tatas and Dr. Costas Kyriacou
Video (Analog - Digital)

- Pre-amplifier
- Amplifier Filters
- A/D
- Image enhancement and coding
- Modulator
- Analog
- Digital

Diagram showing a flow of operations from pre-amplifier to digital processing.
Temperature Recording by a Digital System

Sampling & quantization

Coding
Need for Data Converters

Digital processing and storage of physical quantities (sound, temperature, pressure etc) exploits the advantages of digital electronics

- Better and cheaper technology compared to the analog
- More reliable in terms of storage, transfer and processing
  - Not affected by noise
- Processing using programs (software)
  - Easy to change or upgrade the system
    - (e.g. Media Player 7 → Media Player 8 ή Real Player)
- Integration of different functions
  - (π.χ. Mobile = phone + watch + camera + games + email +
QUANTIZATION ERROR

- The difference between the true and quantized value of the analog signal
- Inevitable occurrence due to the finite resolution of the ADC
- The magnitude of the quantization error at each sampling instant is between zero and half of one LSB.
- Quantization error is modeled as noise (quantization noise)
Interfacing Switches

8051

DIP SWITCH

Vcc

R

P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
What is a Keyboard?

- Collection of keys interfaced to the microcontroller
- Arranged in the form of two dimensional matrix
- Matrix arrangement used for minimizing the number of port lines
- Junction of each row and column forms the key
Interfacing a Keyboard

-One key per port line

8051

P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7

+5V
Keys are organized in two-dimensional matrix to minimize the number of ports required for interfacing.
Interfacing a Keyboard

- Use of decoder further reduces the number of port lines required
Key Issues

• Key bounce can be overcome using Software/Hardware approach
• Keyboard Scanning
• Multiple Key Closure
  ▪ 2-key lockout
  ▪ 2-key rollover
• Minimize Hardware Requirement:
  ▪ Use of Keyboard Encoder
• Minimize Software Overhead
Interfacing a single LED

- Driver circuit to interface a single LED
Seven Segment LEDs

- Two types: Common cathode and common anode type
- Seven-segment LEDs can be conveniently used to display HEX characters
Multidigit Driver

• Features of Multidigit Driver
  - 8-segment driver output lines
  - 8-digit driver lines
  - 20 mA peak current
  - LEDs can withstand high peak current

• Sequencing operation:
  - Select data using digit address lines DA_{0-2}
  - Write data using ID_{0-3} and ID_{7} lines

• Three modes of operation:
  - HIGH: HEX, LOW: OFF, OPEN: CODED-HELP
Liquid Crystal Displays

• Key features:
  - Low Power Consumption
  - Voltage Controlled
  - Easy to read in bright light
  - Declining Cost
  - Ability to display Characters/Graphics
  - Intelligent controller and LCD display panels readily available
Liquid Crystal Displays

[Diagram showing connections and labels such as P1.0, P1.1, B0, B1, B2, B3, D0, D1, D2, D3, OSC, GND, Vcc, DATA IN, DIGIT SELECT, DIGIT 1, DIGIT 2, DIGIT 3, DIGIT 4, BACK PLANE, DISPLAY MODULE]
LCD Display Module

• LCD modules:
  - An LCD panel and small circuit board containing the controller chip
  - 14 – pin connections to microcontroller
  - HITACHI’S HD44780 controller can control up to 80 characters
  - Easy to program
  - 2 rows, 20/40 character in each row
  - Each character can be 5X8 or 5X11 matrix
LCD Display Module

- CG ROM stores segment pattern of 192 char.
- CG RAM stores segment patterns of 16 user-designed char.
- An 8-bit instruction reg.
- An 8-bit data reg.
- DD RAM stores up to 80 8-bit char. Codes
- 11 instructions clear display, return home
P1.0  ---  P1.2
    /    |
P1.1    |
    \

P1.0  ---  P1.2
    /    |
P1.1    |
P1.2    |