# MSE352: Digital Logic & Microcontrollers

# **Lecture 4**

# **I/Os in Microcontroller 8051**

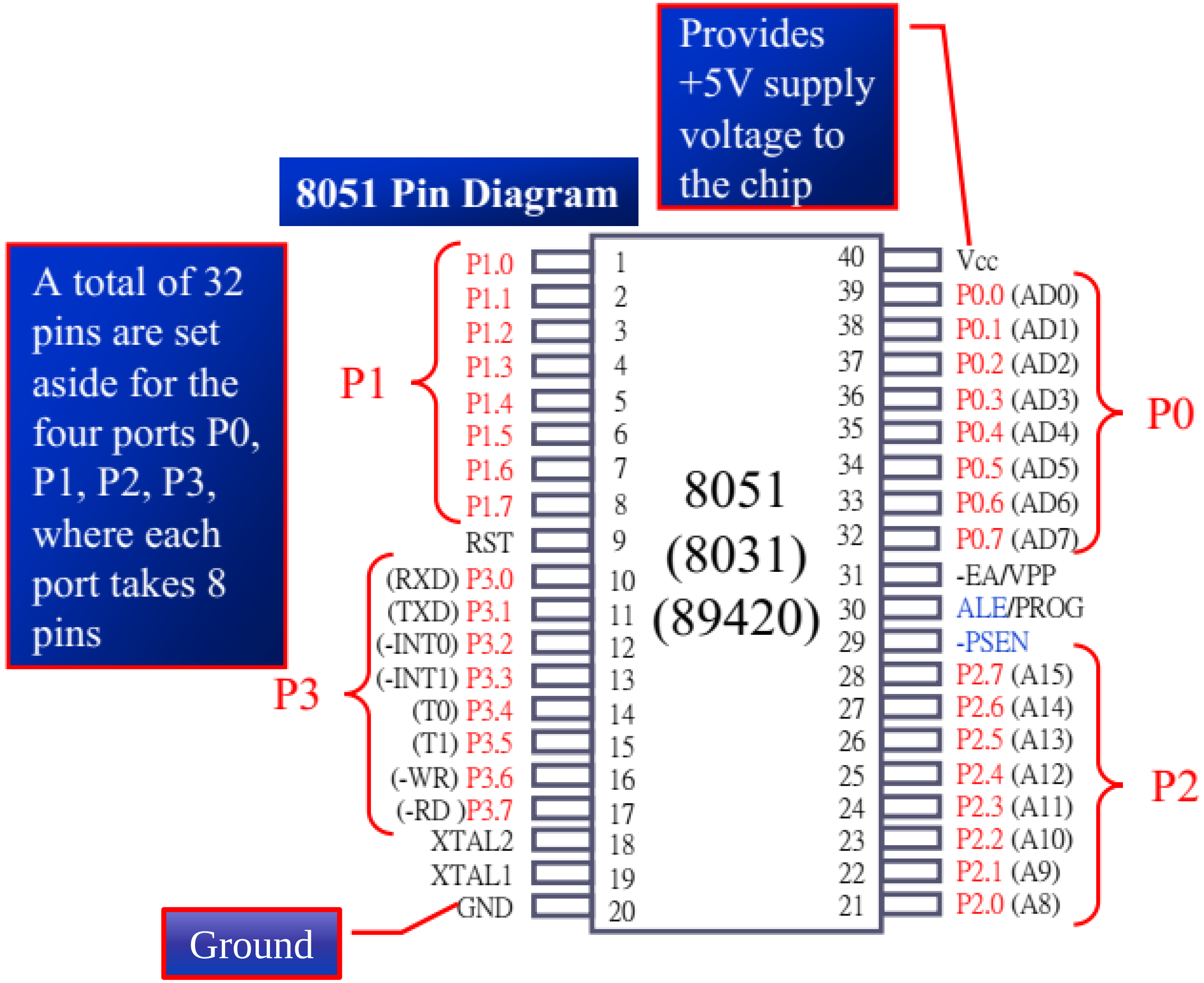

Mohammad Narimani, *Ph.D., P.Eng.*
*Lecturer*
School of Mechatronic Systems Engineering
Simon Fraser University


SFU SIMON FRASER UNIVERSITY SURREY

# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# I/O ports



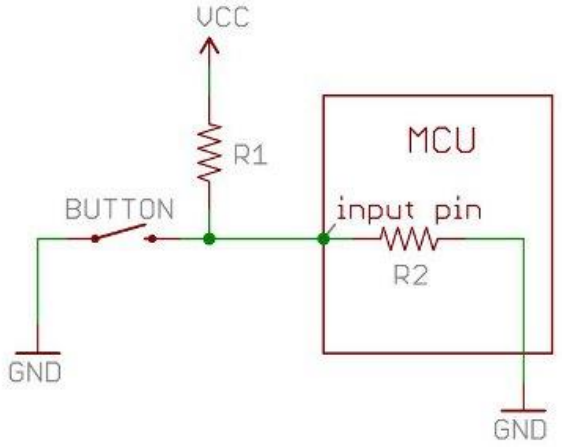
8051 Pin Diagram
Provides +5V supply voltage to the chip
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins
Ground
P1
P3
P0
P2
8051
(8031)
(89420)
P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
(-INT0) P3.2 12
(-INT1) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
(-WR) P3.6 16
(-RD )P3.7 17
XTAL2 18
XTAL1 19
GND 20
40 Vcc
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 -EA/VPP
30 ALE/PROG
29 -PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

# I/O ports

- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins
- All the ports upon RESET are configured as input, ready to be used as input ports
  - When the first 0 is written to a port, it becomes an output
  - To reconfigure it as an input, a 1 must be sent to the port
    - To use any of these ports as an input port, it must be programmed
- In 8051, P0 does not have internal pull-up resistors, while P1, P2 and P3 have.

# I/O ports

- Pull-up resistors
  - Consider we have an MCU with an input pin. If there is nothing connected to that pin, it would be difficult to say what the value of this input pin is (high or low). In the other words it is floating. Therefore, connecting a pull-up resistor, we can determine the state of this pin.
  - Based on pressing the button in the picture, there are two states for the input pin:
    - Button pressed → Low (0)
    - Button released → High (1)

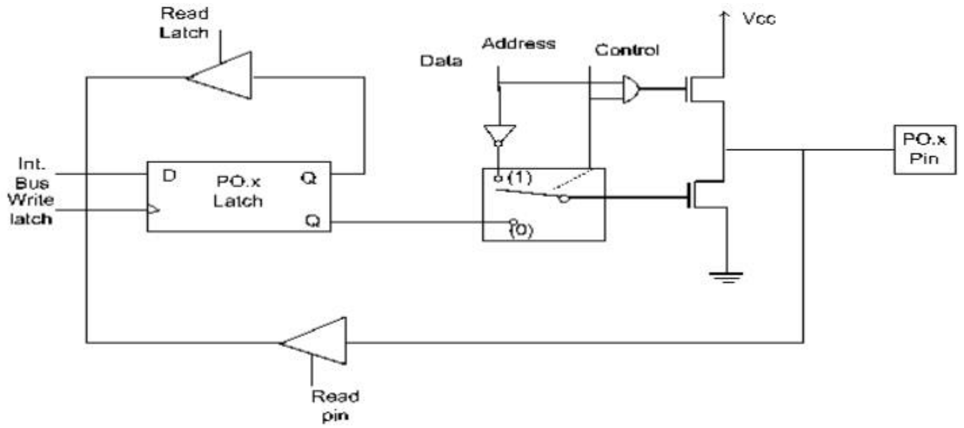


# I/O ports

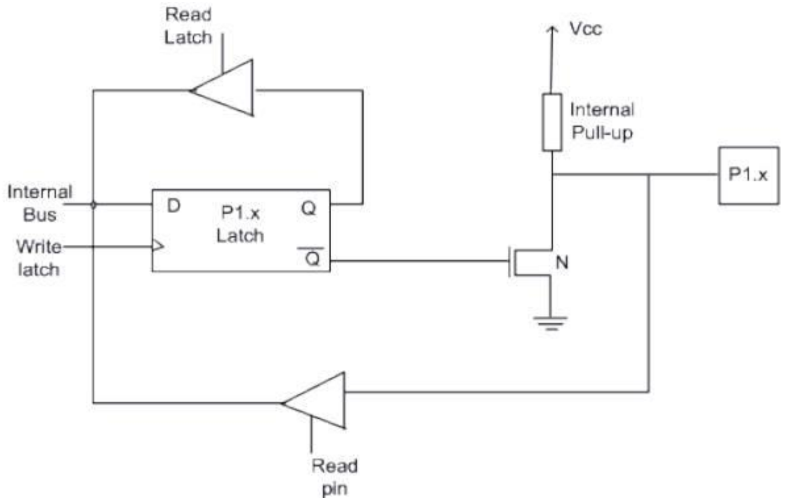
- What is the value of Pull-up Resistor (R1)?
  - There are two considerations for choosing R1:
    - When the button is pressed there should not be high current to ground leading to high power dissipation.
    - The value for R1 should be in the range of R1< 1/10*|Impedance (R2)|. In this way we make sure that when the button is not pressed, the larger amount of voltage drops on R2, to make sure that the input pin voltage is large enough to read it as high. Usually the range of R2 is 100k-1MΩ.
- Internal Pull-ups: Many MCUs have internal pull-ups. For example, ports P1, P2 and P3 in 8051 have internal pull-up resistors.

# I/O ports

- P0 and P1 structure




Open drain Port: P0

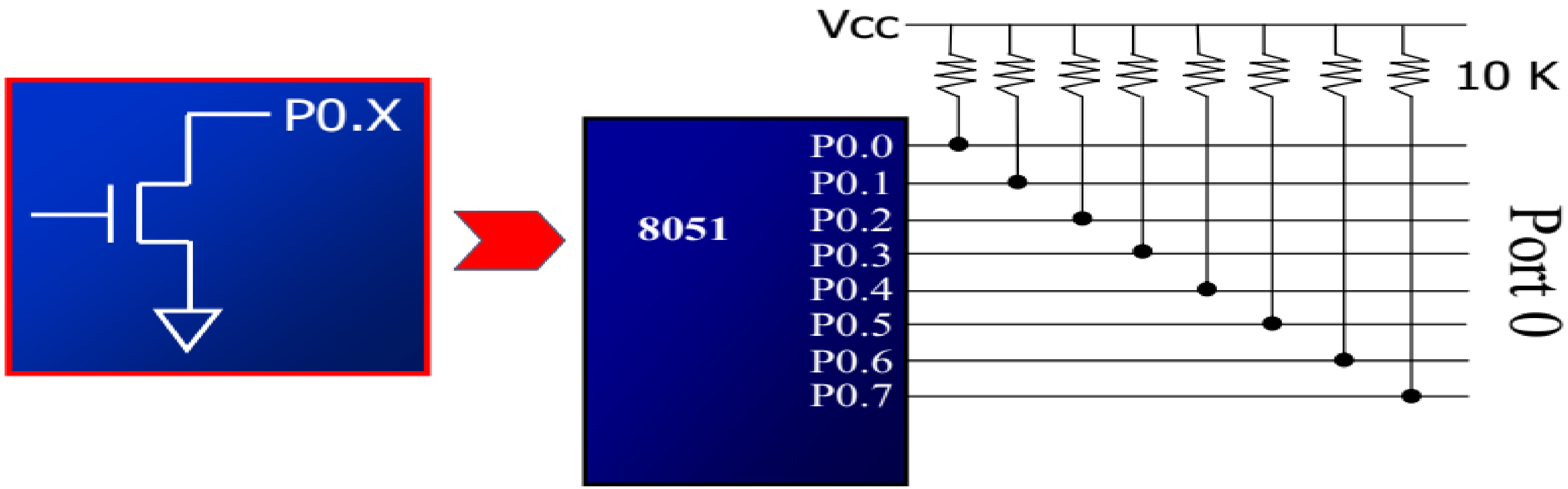



Port with internal pull-up resistor: P1 (P2 and P3 are similar)

- For more detailed information about pins look at 8051 datasheet, page 4 or https://www.youtube.com/watch?v=t9bN-2nFNEs

# I/O ports

- Pull-up resistors for P0 are used for input or output, where each pin must be connected externally to a 10K ohm pull-up resistor
  - This is due to the fact that P0 is an open drain, unlike P1, P2, and P3
    - Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips



# I/O ports

The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK:   MOV     A,#55H
        MOV     P0,A
        ACALL   DELAY
        MOV     A,#0AAH
        MOV     P0,A
        ACALL   DELAY
        SJMP    BACK
```

# I/O ports

- In order to make port 0 an input, the port must be programmed by writing 1 to all the bits

Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1

```
        MOV     A,#0FFH         ;A=FF hex
        MOV     P0,A            ;make P0 an i/p port
                                ;by writing it all 1s
BACK:   MOV     A,P0            ;get data from P0
        MOV     P1,A            ;send it to port 1
        SJMP    BACK            ;keep doing it
```

# I/O ports

- Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data
  - When connecting an 8051/31 to an external memory, port 0 provides both address and data

# I/O ports

- Port 1 can be used as input or output
  - In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally
  - Upon reset, port 1 is configured as an input port

The following code will continuously send out to port 1 the alternating value 55H and AAH

```
        MOV     A,#55H
BACK:   MOV     P1,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

# I/O ports

- To make port 1 an input port, it must be programmed as such by writing 1 to all its bits

Port 1 is configured first as an input port by writing 1s to it, then data is received from that port and saved in R7 and R5

```
        MOV     A,#0FFH      ;A=FF hex
        MOV     P1,A         ;make P1 an input port
                             ;by writing it all 1s
        MOV     A,P1         ;get data from P1
        MOV     R7,A         ;save it to in reg R7
        ACALL   DELAY        ;wait
        MOV     A,P1         ;another data from P1
        MOV     R5,A         ;save it to in reg R5
```

# I/O ports

- Port 2 can be used as input or output
  - Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally
  - Upon reset, port 2 is configured as an input port
- To make port 2 an input port, it must be programmed as such by writing 1 to all its bits
  - Port 2 is also designated as A8 – A15, indicating its dual function
  - Port 0 provides the lower 8 bits via A0 – A7

# I/O ports

- Port 3 can be used as input or output
  - Port 3 does not need any pull-up resistors
  - Port 3 is configured as an input port upon reset, this is not the only way it is most commonly used
- Port 3 has the additional function of providing some extremely important signals

| P3 Bit | Function | Pin |
|---|---|---|
| P3.0 | RxD | 10 |
| P3.1 | TxD | 11 |
| P3.2 | $\overline{\text{INT0}}$ | 12 |
| P3.3 | $\overline{\text{INT1}}$ | 13 |
| P3.4 | T0 | 14 |
| P3.5 | T1 | 15 |
| P3.6 | $\overline{\text{WR}}$ | 16 |
| P3.7 | $\overline{\text{RD}}$ | 17 |

Serial communications

External interrupts

Timers

Read/Write signals of external memories

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate function role

# I/O ports

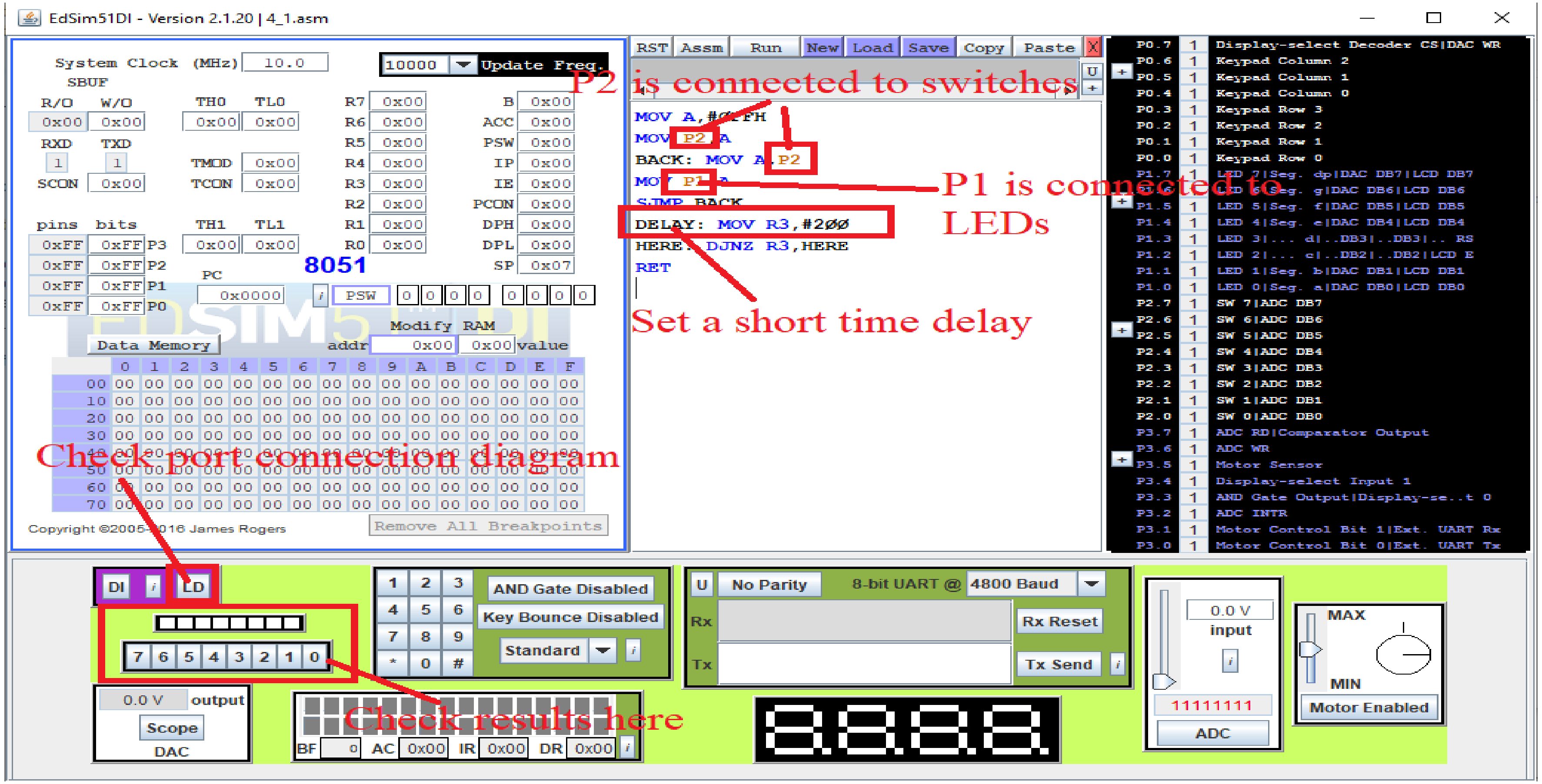
Write a program for the DS89C420 to toggle all the bits of P0, P1, and P2 every 1/4 of a second

```
        ORG     0
BACK:   MOV     A,#55H
        MOV     P0,A
        MOV     P1,A
        MOV     P2,A
        ACALL   QSDELAY      ;Quarter of a second
        MOV     A,#0AAH
        MOV     P0,A
        MOV     P1,A
        MOV     P2,A
        ACALL   QSDELAY
        SJMP    BACK
QSDELAY:
        MOV     R5,#11
H3:     MOV     R4,#248
H2:     MOV     R3,#255
H1:     DJNZ    R3,H1        ;4 MC for DS89C4x0
        DJNZ    R4,H2
        DJNZ    R5,H3
        RET
        END
```

Delay
$= 11 \times 248 \times 255 \times 4 \text{ MC} \times 90 \text{ ns}$
$= 250{,}430\ \mu s$

# I/O ports

- EdSim example: Setting I/O port and control LEDs with switches.






SFU SIMON FRASER UNIVERSITY SURREY

# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# Accessing entire 8 bits and each bit

The entire 8 bits of Port 1 are accessed

```
BACK:   MOV     A,#55H
        MOV     P1,A
        ACALL   DELAY
        MOV     A,#0AAH
        MOV     P1,A
        ACALL   DELAY
        SJMP    BACK
```

Rewrite the code in a more efficient manner by accessing the port directly without going through the accumulator

```
BACK:   MOV     P1,#55H
        ACALL   DELAY
        MOV     P1,#0AAH
        ACALL   DELAY
        SJMP    BACK
```

Another way of doing the same thing

```
        MOV     A,#55H
BACK:   MOV     P1,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

# Accessing entire 8 bits and each bit

- Sometimes we need to access only 1 or 2 bits of the port

```
BACK:   CPL     P1.2            ;complement P1.2
        ACALL   DELAY
        SJMP    BACK

;another variation of the above program
AGAIN:  SETB    P1.2            ;set only P1.2
        ACALL   DELAY
        CLR     P1.2            ;clear only P1.2
        ACALL   DELAY
        SJMP    AGAIN
```

| P0 | P1 | P2 | P3 | Port Bit |
|---|---|---|---|---|
| P0.0 | P1.0 | P2.0 | P3.0 | D0 |
| P0.1 | P1.1 | P2.1 | P3.1 | D1 |
| P0.2 | P1.2 | P2.2 | P3.2 | D2 |
| P0.3 | P1.3 | P2.3 | P3.3 | D3 |
| P0.4 | P1.4 | P2.4 | P3.4 | D4 |
| P0.5 | P1.5 | P2.5 | P3.5 | D5 |
| P0.6 | P1.6 | P2.6 | P3.6 | D6 |
| P0.7 | P1.7 | P2.7 | P3.7 | D7 |

# Accessing entire 8 bits and each bit

**Example 4-2**

Write the following programs.
Create a square wave of 50% duty cycle on bit 0 of port 1.

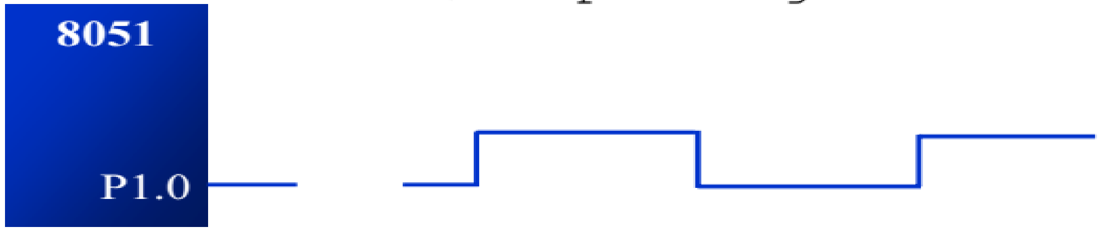
**Solution:**

The 50% duty cycle means that the "on" and "off" state (or the high and low portion of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

```
HERE:   SETB    P1.0    ;set to high bit 0 of port 1
        LCALL   DELAY   ;call the delay subroutine
        CLR     P1.0    ;P1.0=0
        LCALL   DELAY
        SJMP    HERE    ;keep doing it
```

Another way to write the above program is:

```
HERE:   CPL     P1.0    ;set to high bit 0 of port 1
        LCALL   DELAY   ;call the delay subroutine
        SJMP    HERE    ;keep doing it
```



# Accessing entire 8 bits and each bit

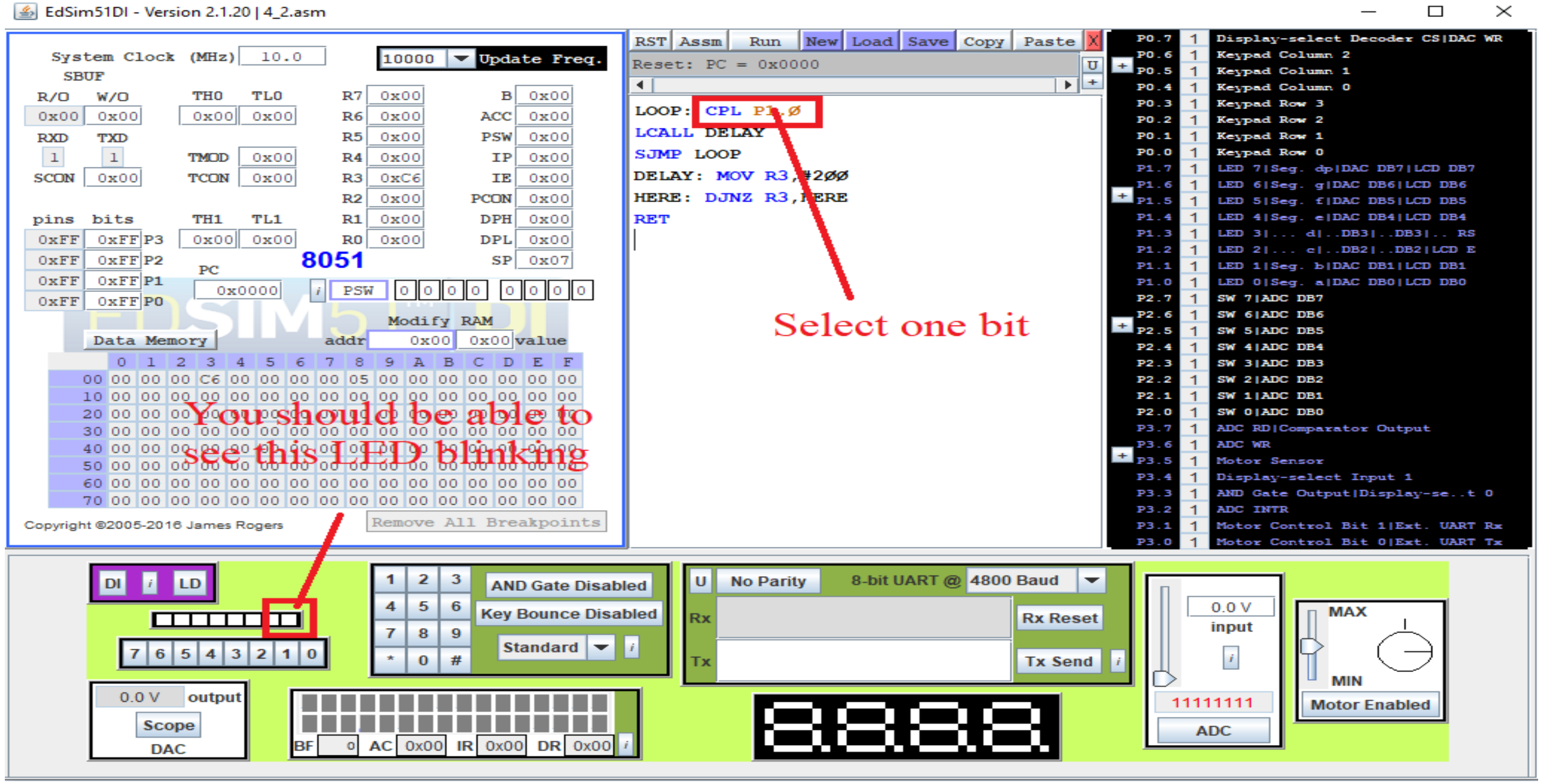
- Instructions that are used for signal-bit operations are as following

**Single-Bit Instructions**

| Instruction | Function |
|---|---|
| SETB bit | Set the bit (bit = 1) |
| CLR bit | Clear the bit (bit = 0) |
| CPL bit | Complement the bit (bit = NOT bit) |
| JB bit, target | Jump to target if bit = 1 (jump if bit) |
| JNB bit, target | Jump to target if bit = 0 (jump if no bit) |
| JBC bit, target | Jump to target if bit = 1, clear bit (jump if bit, then clear) |

# Accessing entire 8 bits and each bit

- EdSim example: select one bit and entire 8 bit




- Change first line to ‘LOOP: CPL P1’.

# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# Check an input bit

- The JNB and JB instructions are widely used single-bit operations
  - They allow you to monitor a bit and make a decision depending on whether it's 0 or 1
  - These two instructions can be used for any bits of I/O ports 0, 1, 2, and 3
    - Port 3 is typically not used for any I/O, either single-bit or byte-wise

**Instructions for Reading an Input Port**

| Mnemonic | Examples | Description |
|---|---|---|
| MOV A,PX | MOV A,P2 | Bring into A the data at P2 pins |
| JNB PX.Y, .. | JNB P2.1,TARGET | Jump if pin P2.1 is low |
| JB PX.Y, .. | JB P1.3,TARGET | Jump if pin P1.3 is high |
| MOV C,PX.Y | MOV C,P2.4 | Copy status of pin P2.4 to CY |

# Check an input bit

**Example 4-3**

Write a program to perform the following:
(a) Keep monitoring the P1.2 bit until it becomes high
(b) When P1.2 becomes high, write value 45H to port 0
(c) Send a high-to-low (H-to-L) pulse to P2.3

**Solution:**

```
        SETB P1.2          ;make P1.2 an input
        MOV  A,#45H        ;A=45H
AGAIN:  JNB  P1.2,AGAIN ; get out when P1.2=1
        MOV  P0,A          ;issue A to P0
        SETB P2.3          ;make P2.3 high
        CLR  P2.3          ;make P2.3 low for H-to-L
```

# Check an input bit

**Example 4-4**

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

**Solution:**

```
HERE:   JNB   P2.3,HERE  ;keep monitoring for high
        SETB  P1.5        ;set bit P1.5=1
        CLR   P1.5        ;make high-to-low
        SJMP  HERE        ;keep repeating
```

# Check an input bit

**Example 4-5**

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2

**Solution:**

```
        SETB P1.7          ;make P1.7 an input
AGAIN:  JB   P1.2,OVER     ;jump if P1.7=1
        MOV  P2,#'N'       ;SW=0, issue 'N' to P2
        SJMP AGAIN         ;keep monitoring
OVER:   MOV  P2,#'Y'       ;SW=1, issue 'Y' to P2
        SJMP AGAIN         ;keep monitoring
```

# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# Reading single bit into carry flag

**Example 4-6**

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2
Use the carry flag to check the switch status.

**Solution:**

```
        SETB P1.7          ;make P1.7 an input
AGAIN:  MOV  C,P1.2        ;read SW status into CF
        JC   OVER          ;jump if SW=1
        MOV  P2,#'N'       ;SW=0, issue 'N' to P2
        SJMP AGAIN         ;keep monitoring
OVER:   MOV  P2,#'Y'       ;SW=1, issue 'Y' to P2
        SJMP AGAIN         ;keep monitoring
```

# Reading single bit into carry flag

**Example 4-7**

A switch is connected to pin P1.0 and an LED to pin P2.7. Write a program to get the status of the switch and send it to the LED

**Solution:**

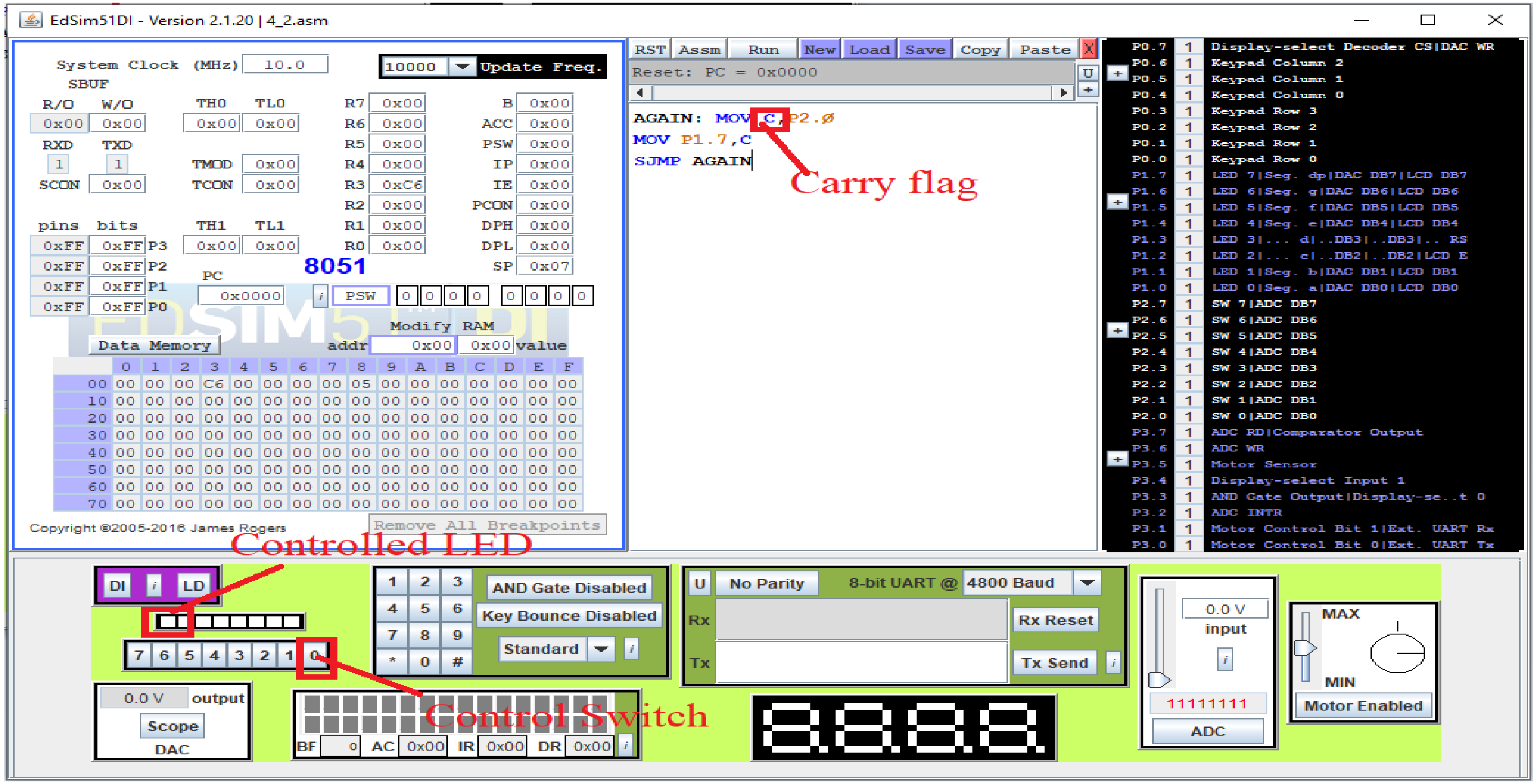
```
        SETB  P1.0          ;make P1.7 an input
AGAIN:  MOV   C,P1.0        ;read SW status into CF
        MOV   P2.7,C        ;send SW status to LED
        SJMP  AGAIN         ;keep repeating
```

The instruction 'MOV P2.7,P1.0' is wrong , since such an instruction does not exist

However 'MOV P2,P1' is a valid instruction

# Reading single bit into carry flag

- EdSim example: use carry flag to let switch control single LED



# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# Reading Latch for output port

- In reading a port
  - Some instructions read the status of port pins
  - Others read the status of an internal port latch
- Therefore, when reading ports there are two possibilities:
  - Read the status of the input pin
  - Read the internal latch of the output port
- Confusion between them is a major source of errors in 8051 programming
  - Read the internal latch of the output port

# Reading Latch for output port

- Some instructions read the contents of an internal port latch instead of reading the status of an external pin
  - For example, look at the **ANL P1, A** instruction and the sequence of actions is executed as follow

1. It reads the internal latch of the port and brings that data into the CPU
2. This data is ANDed with the contents of register A
3. The result is rewritten back to the port latch
4. The port pin data is changed and now has the same value as port latch

# Outline

- I/O ports
- Accessing entire 8 bits and each bit
- Check an input bit
- Reading single bit into carry flag
- Reading Latch for output port
- Read-modify-write feature

# Reading Latch for output port

- Read-Modify-Write
  - The instructions read the port latch normally read a value, perform an operation then rewrite it back to the port latch

**Instructions Reading a latch (Read-Modify-Write)**

| Mnemonics | Example |
|---|---|
| ANL PX | ANL P1,A |
| ORL PX | ORL P2,A |
| XRL PX | XRL P0,A |
| JBC PX.Y,TARGET | JBC P1.1,TARGET |
| CPL PX.Y | CPL P1.2 |
| INC PX | INC P1 |
| DEC PX | DEC P2 |
| DJNZ PX.Y,TARGET | DJNZ P1,TARGET |
| MOV PX.Y,C | MOV P1.2,C |
| CLR PX.Y | CLR P2.3 |
| SETB PX.Y | SETB P2.3 |

Note: x is 0, 1, 2, or 3 for P0 – P3

# Read-modify-write feature

- The ports in 8051 can be accessed by the Read-modify-write technique
- This feature saves many lines of code by combining in a single instruction all three actions
  1. Reading the port
  2. Modifying it
  3. Writing to the port

```
        MOV     P1,#55H   ;P1=01010101
AGAIN:  XRL     P1,#0FFH  ;EX-OR P1 with 1111 1111
        ACALL   DELAY
        SJMP    BACK
```