University of British Columbia
Electrical and Computer Engineering
Digital Systems and Microcomputers
CPEN312

Lecture 18/19: Memory and Simple I/O in the 8051 Microcontroller

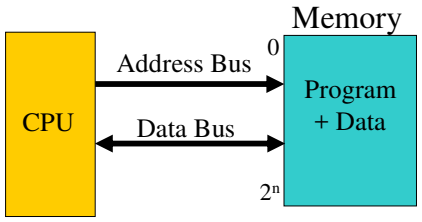
Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

April 4, 2017

Objectives

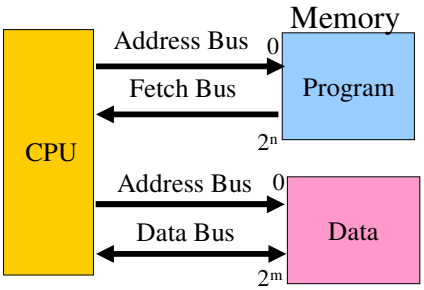
- Understand the differences and uses of bit, data, idata, xdata, and code memory in the 8051 microcontroller.
- Use the appropriate assembly instruction for each type of memory.
- Define variables using assembly code in the bit, data, idata, xdata, and code memory spaces.
- Perform digital I/O using the 8051.
- Use 7-segment displays to show numbers.
- Read push buttons.
- Understand and solve the problem of contact bounce.

Microcontroller Architectures



Von Neumann Architecture

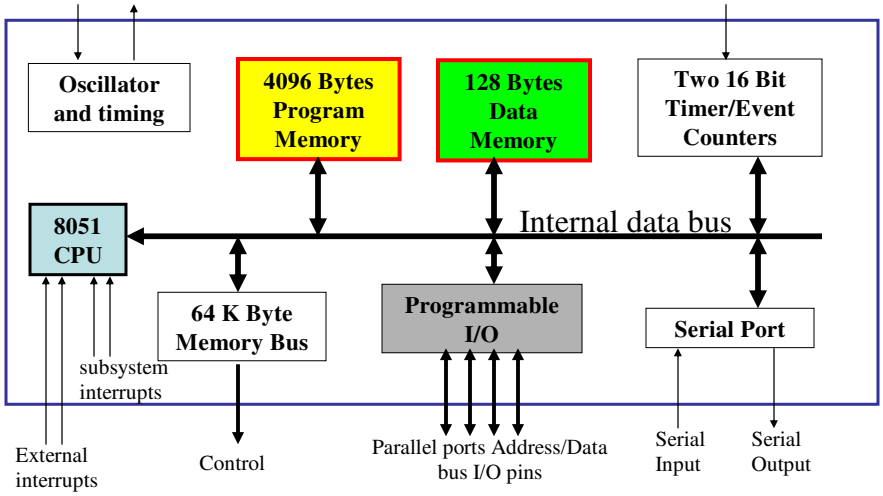
Same assembly instruction can access code and data.



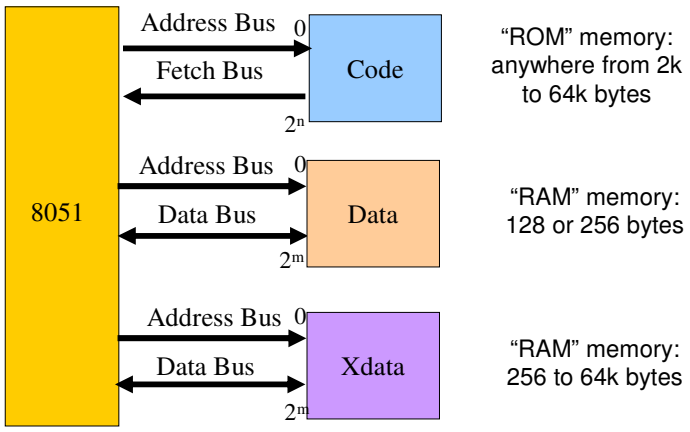
Harvard Architecture

Different assembly instructions to access code and data.

"Original" 8051 Microcontroller



"Modern" 8051 Microcontroller Memory



ROM Types

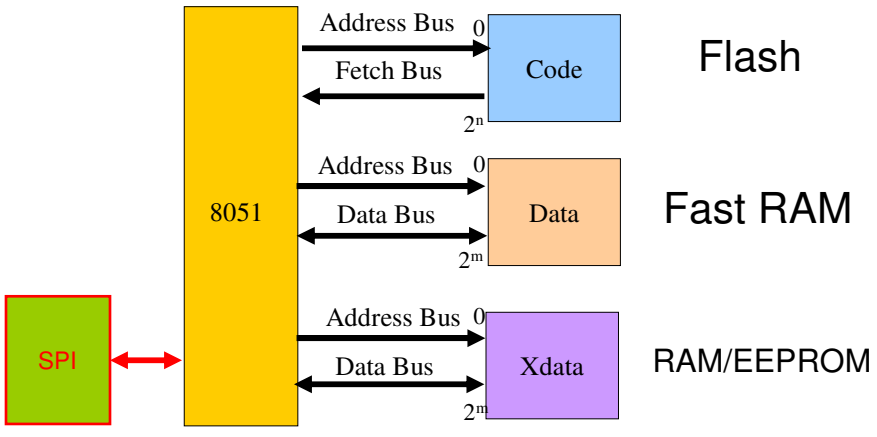
- ROM (Read Only Memory): can only be read by the microcontroller.
  - PROM: Non erasable.
  - EPROM: Erasable using ultraviolet light.
  - EEPROM: Erasable using electricity! Can be rewritten in a byte by byte basis. Good for non-volatile storage.
  - Flash: Erasable. Normally you'll need to erase a significant part of the memory before write anything to it. Cheap, inexpensive, fast, and reliable. DO NOT BUY MICROCONTROLLERS WITHOUT FLASH MEMORY.

RAM Types

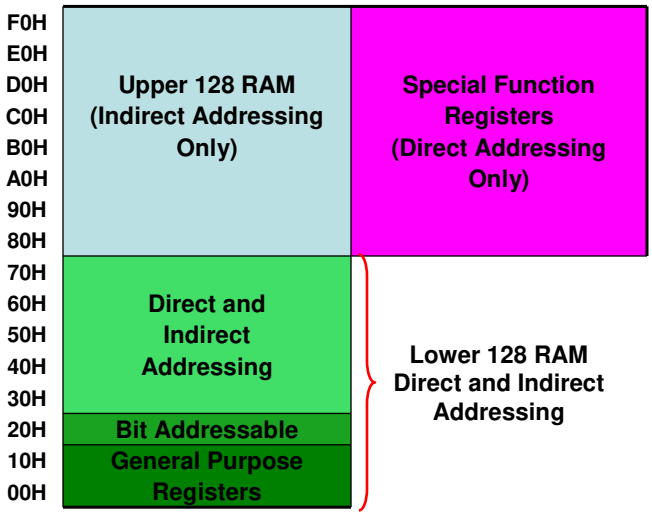
RAM (Random Access Memory):

- Dynamic: Ultra dense, but need to be refreshed periodically or the content will fade away. One memory cell is basically a capacitor! Not used very often with small/inexpensive microcontrollers.
- Static: Not as dense, but simpler to use. One memory cell is basically a flip-flop.

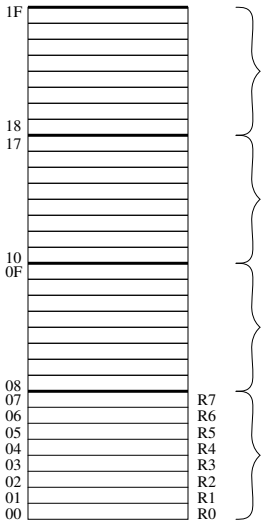
"Modern" 8051 Microcontroller Memory



On-Chip DATA Memory: RAM



General Purpose Registers



- **There are efficient MOV instructions that target the registers directly:**

  MOV R7, #55H ; takes 2 bytes

  MOV 07H, #55H ; takes 3 bytes

- **To change the register bank in use set/reset the RS0 and RS1 bits in PSW:**

RS1	RS0	Selects
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Bit Addressable Memory

2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

20h – 2Fh (16 locations X 8-bits = 128 bits)

Bit addressing:

setb 1Ah ; Two bytes

Byte addressing:

orl 23h, #04H ; Three bytes

Direct addressable memory

- If address < 80H: the target is general purpose RAM.
- If address ≥ 80H: the target is Special Function Registers

```
mov 30H, #0AAH ; Move number 0AAH to RAM 30H
mov 80H, #04H  ; Move number 04H to SFR 80H (P0)
mov P0, #04H   ; Move number 04H to SFR 80H (P0)
```

Bit Addressable SFRs

- If the address of a SFR ends in “0” or “8”, the bits of the SFR are bit addressable.

```
orl 80H, #04H ; Set bit 2 of SFR 80H (P0)
orl P0, #04H  ; Set bit 2 of SFR 80H (P0)
setb P0.2     ; Set bit 2 of SFR 80H (P0)
setb 82H      ; Set bit 2 of SFR 80H (P0)
orl DPH, #100 ; DPH is at address 83H, not bit addressable
```


Indirectly addressable memory

- All the data memory in the 8051 is indirectly addressable.
- The SFRs are not indirectly addressable.
- To indirectly access memory we use the indexing registers R0 and R1.

Indirectly addressable memory

Write a program that clears (set to zero) 32 consecutive bytes from internal RAM starting at address 70H.

```
    mov R0, #70H ; Starting address we want to clear
    mov R7, #32  ; How many bytes we want to clear
L1:
    mov @R0, #0  ; Set memory pointed by R0 to zero
    inc R0       ; Point to the next byte
    djnz R7, L1  ; Repeat as many times as needed
```


Expanded RAM

- In the original 8051 expanded (or external) RAM was only available as additional memory chips. Up to 64k!
- Newer 8051's may include some built in expanded RAM.
- Expanded RAM can only be accessed indirectly (via a pointer) using the MOVX instruction.
- The MOVX instruction uses either the DPTR (DPL, DPH), R0, or R1 as indexes.

Expanded RAM Access

Write a program that sets 50H bytes from expanded memory at address 80H to 255

```
    mov A, #0FFH        ; Value we want to set the memory to
    mov DPTR, #128      ; Starting address we want to set
    mov R0, #50H        ; How many bytes we want to set
X1:
    movx @DPTR, A       ; Set memory pointed by R0 to zero
    inc DPTR            ; Point to the next memory location
    djnz R0, X1         ; Repeat as many times as needed
```

Some 8051's derivatives have two DPTR registers to simplify memory access. Check the data sheet for 'dual data pointers'.

Remember: 80H=128, 0FFH=255

Code Memory

- Code memory can only be accessed indirectly using the MOVC instruction.
- The MOVC instruction uses either the DPTR (DPL, DPH), or the accumulator as indexes.
- Very useful when working with look-up tables!
- We can only READ from code memory.

Code Memory Access

Write a program to copy 8 bytes from code memory at address 1000H to data memory at address A0H

```
; Copy from code memory to data memory
    mov DPTR, #1000H    ; Source address in code memory
    mov R0, #A0H        ; Destination address in data memory
    mov R2, #08H        ; # of bytes we want to copy
C1:
    clr A
    movc A, @A+DPTR     ; Read from code memory
    mov @R0, A          ; Copy to data memory!
    inc DPTR            ; Point to next byte in code memory
    inc R0              ; Point to next byte in data memory
    djnz R2, C1         ; Repeat as many times as needed
```


Defining Variables in Assembly Source Code

- Use the segment directives (BSEG, DSEG, ISEG, XSEG, CSEG) to select bit, data, idata, xdata, or code memory.
- Use the origin directive override (AT) to set the beginning of the segment.
- USE the space allocation directives (DS, DBIT, DB, DW) to define the variables or their values.

Defining Variables in Assembly Source Code

```
; Defining bit variables
BSEG

ALARM: dbit 1
ONOFF: dbit 1

; Defining variables in data memory
DSEG at 30H

COUNT_HIGH: ds 1
COUNT_LOW:  ds 1
COUNT:      ds 2
BUFFER:     ds 16
```


Defining Variables in Assembly Source Code

```
; Defining xdata variables
XSEG at 80H

Sum:    ds 2
Adjust: ds 20

; Defining constants in code memory
CSEG at 1000H

Message: db 'Hello there…'
         db 0ah, 0dh ; ASCII code for nl/cr
         db 0
```

Internal 8051 I/O ports

- The standard 8051 has **FOUR** 8-bit bidirectional I/O ports.
- To use a pin on the original 8051 as an input you **MUST** first write a '1' to it.
- All pins of port 0 of the standard 8051 need a pull-up resistor (around 10k) before used as outputs.
- All standard I/O pins are bit and byte addressable. For example:

```
setb P1.0
clr P2.7
mov P0, #0C0H
```

- All I/O pins may serve multiple functions. For example P3.4 can also be used as the input for Timer 0.

Configuring the Ports for Input/Output

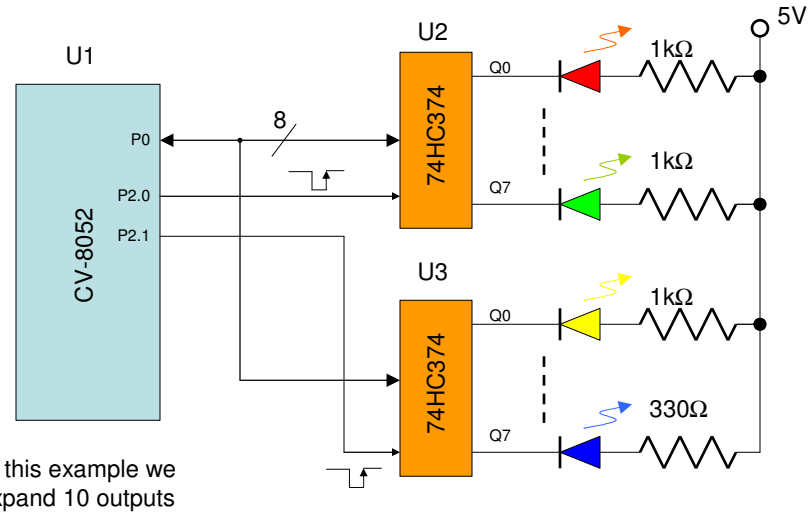
- Newer 8051 variants have ports that can handle more current, but they need to be configured first.
- For example, to configure the ports in the CV-8052 soft processor we use SFRs P0MOD, P1MOD, P2MOD, and P3MOD:

```
mov P0MOD, #00000010b ; Make P0.1 output
mov P1MOD, #0ffH ; All pins of P1 are now outputs
mov P2MOD, #00001111b ; P2.0 to P2.3 outputs, P2.4 to P2.7 inputs
```

Expanding the Internal Ports

- Many of new 8051 derivatives (small and cheap) do not have an external address/data bus.
- If you need more digital I/O there are basically two options:
  - Use SPI or I²C peripherals. Works great and nowadays you can get interesting SPI or I²C peripherals at very low cost!
  - Use digital logic to expand the available ports.

Expanding the Internal Ports

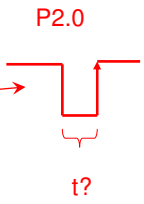


In this example we expand 10 outputs into 16 outputs!

Expanding the Internal Ports

```
; Configure P0 and P2
mov P0MOD, #0FFH
mov P2MOD, #03H

; Write R3 to U2
mov P0, R3
; Strobe the value into U2
clr P2.0
setb P2.0
; Write 55H to U3
mov P0, #55H
; Strobe the value into U3
clr P2.1
setb P2.1
```

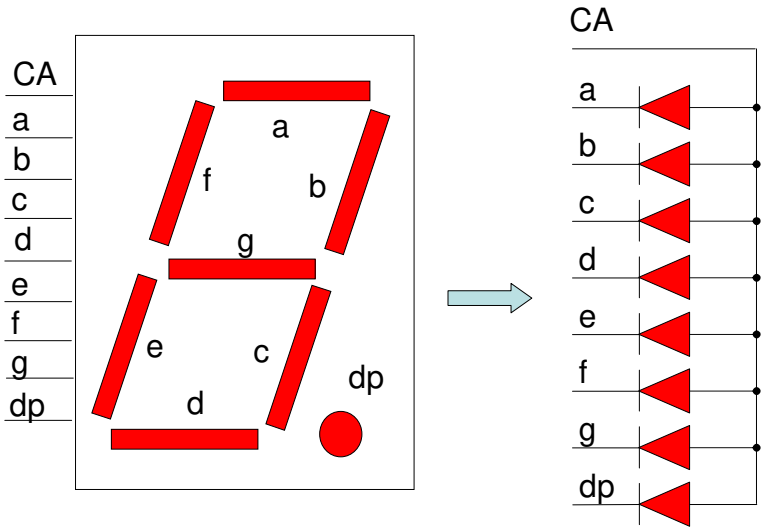


f=33.33MHz, 1 clock per cycle, two cycle instruction: 60ns

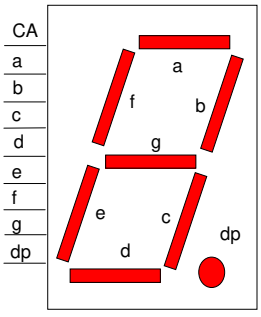
Example 1: Writing to Seven Segment LED displays

- Add two 7-segment LED displays to ports 0 and 3 of the CV-8052 microcontroller. The clock of the microcontroller has a frequency of 33.33MHz and it takes one clock period per machine cycle. Write a program to increment a counter approximately every second from 00 to 59 (in BCD). When the count reaches 60 reset it to zero and keep going.

7-Segment Display



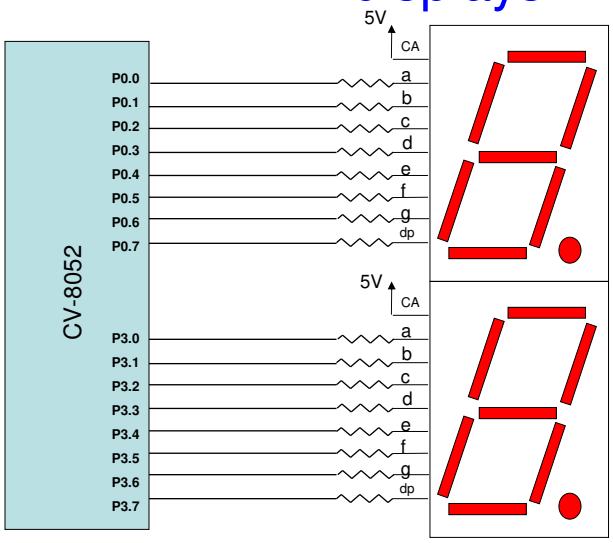
7-segment display look-up table



#	dp	g	f	e	d	c	b	a	HEX
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									

Zero is on!

Example 1: Writing to 7-Segment LED displays



Example 1: code

```
$MODDE0CV
org 0
   ljmp mycode

; Look-up table...
mytable:
    DB 0C0H, 0F9H, 0A4H, 0B0H, 099H
    DB 092H, 082H, 0F8H, 080H, 090H
Wait1Sec:
;33.33MHz, 1 clock per cycle: 0.030us
    mov R2, #180
L3: mov R1, #250
L2: mov R0, #250
L1: djnz R0, L1 ; 3 machine cycles-> 3*30ns*250=22.5us
    djnz R1, L2 ; 22.5us*250=5.625ms
    djnz R2, L3 ; 5.625ms*180=1s (approximately)
    ret
```

Example 1: code

```
mycode:
    mov SP, #7FH
    mov P0MOD, #0FFH
    mov P3MOD, #0FFH
    mov B, #0 ; counter
    mov dptr, #mytable
forever:
; Display MSB
    mov A, B
    swap A
    anl A, #0FH
    movc A, @A+dptr
    mov P0, A
; Display LSB
    mov A, B
    anl A, #0FH
    movc A, @A+dptr
    mov P3, A
```

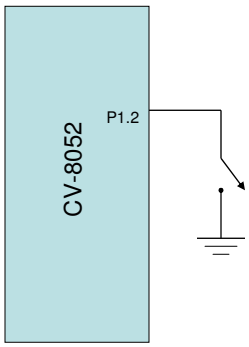

Example 1: code

```
; Do the one second delay
    lcall Wait1Sec
; Increment counter (register B) and convert to BCD
    mov A, B
    add A, #1
    da A
    mov B, A
; Check if count is 60
    cjne A, #060H, forever
    mov B, #0
    sjmp forever
END
```

Example 2: Reading Push Buttons

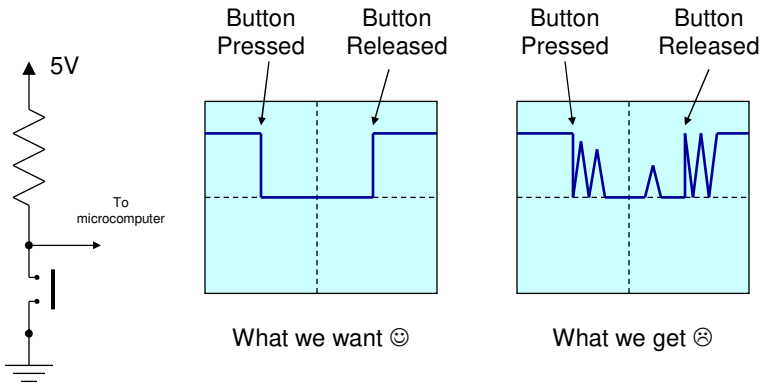
- Before using a pin for input we need to configure it:
  - Original 8051: Write '1' to the pin to be used as input.
  - Newer 8051s: configure the pin as input using designated SFRs.
- In the original 8051 any pin can be used as output or input. In newer 8051s some pins can be only input and/or outputs.
- In the original 8051 pins in the same port can be independently used as inputs or outputs. For example pin P0.0 can be used as input, while P0.1 can be used as output!

Example 2: Reading Push Buttons



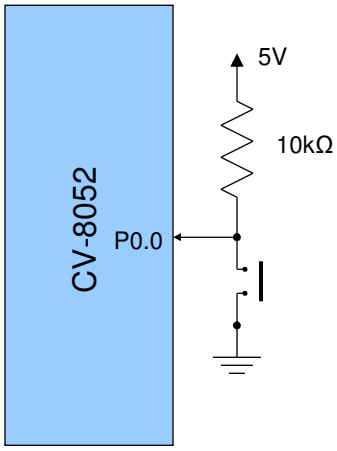

```
mov A, P1MOD
anl a, #11111011b ; P1.2 is input
mov P1MOD, a
.
.
.
.
jnb P1.2, ButtonPressed
jb P1.2, ButtonNotPressed
```

Problem: Contact Bounce



The time the contact bounces
can be as long as 50ms!

Software De-bouncing

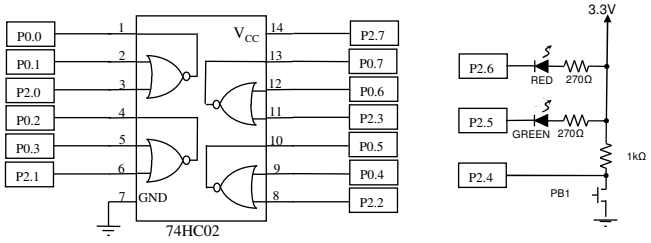



```
; After reset all ports are inputs
jb P0.0, not_pressed
lcall Wait50ms ; Wait and check again
jb P0.0, not_pressed
; Wait for the button to be released
L0: jnb P0.0, L0
sjmp pressed
```

This technique is called *wait-and-see*
in many textbooks

Example 3: Logic IC testing.

- The circuit in the figure below can be used to verify that a 74HC02 integrated circuit (IC) operates correctly. The 74HC02 IC consists of four 2-input NOR gates. Write a subroutine for the CV-8052 processor that tests the IC after push button PB1 is pressed and then released (no de-bouncing needed). Additionally, the subroutine should: configure the input and output pins, apply power to the IC, test all possible input/output combinations and either turn the green LED on if the IC passes all the tests or the red LED on if the IC fails any test. The subroutine should set the power pin as well as all of the IC inputs to zero before returning so that the IC can be removed safely from the circuit after the tests are completed.



Example 3: Logic IC testing.

- Pins that are outputs from the CV-8052, inputs to the IC:

  P0.1, P2.0, P0.3, P2.1, P2.2, P0.4, P2.3, P0.6, P2.5, P2.6, P2.7
- Pins that inputs to the CV-8052, outputs from the IC:

  P0.0, P0.2, P0.5, P0.7, P2.4

Example 3: Logic IC testing.

```
Test_NOR_IC:
    mov P0MOD, #01011010b
    mov P2MOD, #11101111b

    jb P2.4, $ ; Wait for PB1 press
    mov P2, #11100000B ; Power on, red LED off, green LED off
    jnb P2.4, $ ; Wait for PB1 release

    ; 0 nor 0 = 1
    clr P0.1
    clr P2.0
    jnb P0.0, error_NOR
    clr P0.3
    clr P2.1
    jnb P0.2, error_NOR
    clr P0.4
    clr P2.2
    jnb P0.5, error_NOR
    clr P2.3
    clr P0.6
    jnb P0.7, error_NOR
```


Example 3: Logic IC testing.

```
    ; 1 nor 0 = 0
    setb P0.1
    clr P2.0
    jb P0.0, error_NOR
    setb P0.3
    clr P2.1
    jb P0.2, error_NOR
    setb P0.4
    clr P2.2
    jb P0.5, error_NOR
    setb P2.3
    clr P0.6
    jb P0.7, error_NOR

    ; 0 nor 1 = 0
    clr P0.1
    setb P2.0
    jb P0.0, error_NOR
    clr P0.3
    setb P2.1
    jb P0.2, error_NOR
    clr P0.4
    setb P2.2
    jb P0.5, error_NOR
    clr P2.3
    setb P0.6
    jb P0.7, error_NOR
```

Example 3: Logic IC testing.

```
    ; 1 nor 1 = 0
    setb P0.1
    setb P2.0
    jb P0.0, error_NOR
    setb P0.3
    setb P2.1
    jb P0.2, error_NOR
    setb P0.4
    setb P2.2
    jb P0.5, error_NOR
    setb P2.3
    setb P0.6
    jb P0.7, error_NOR

    clr P2.5 ; Success, all tests pass
    sjmp done
error_NOR:
    clr P2.6 ; At least one test fails
done:
    anl P0, #10100101B ; Set all the IC inputs to zero
    anl P2, #11110000B ; Set all the IC inputs to zero
    clr P2.7 ; Turn power off
    ret
```


Exercises

- The CV-8052 soft processor has six 7-segment displays accessible through SFRs HEX0 to HEX5. Write an assembly clock program that displays the time of day as hours, minutes, and seconds.
- Modify the program from the previous exercise so that the hour, minute, and second displays can be set using push buttons KEY1, KEY2, and KEY3. De-bounce the push buttons!
- Add two extra 7-segment displays to the CV-8052 connected to ports P0 and P1. Write a program that displays the 8 digits of your student number using P1, P0, HEX5, HEX4, HEX3, HEX2, HEX1, and HEX0.