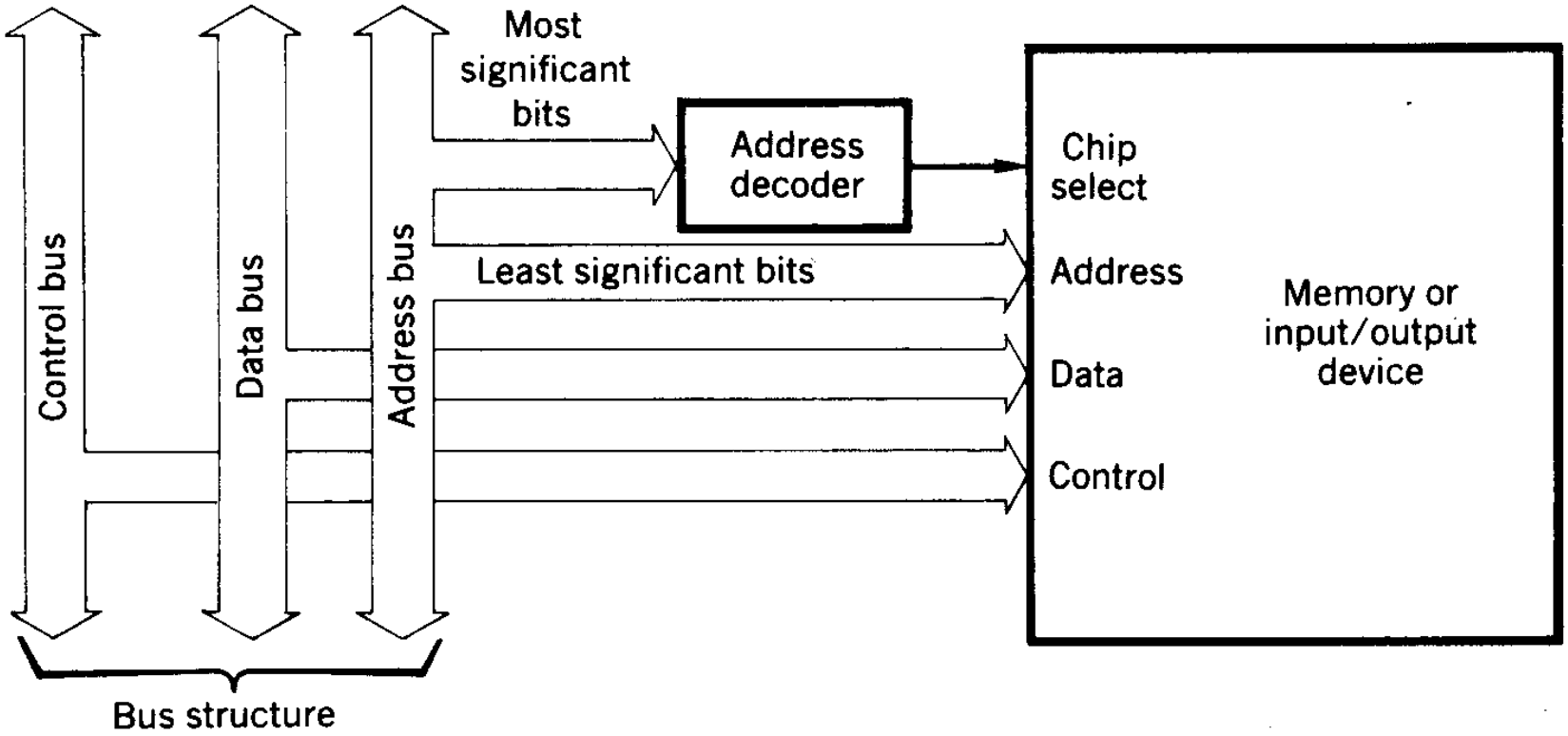
Chương 5
Thiết kế hệ vi xử lý

Nội dung

- Giải mã địa chỉ
- Giao tiếp bộ nhớ
- Giao tiếp với khóa (switch) và bàn phím
- Giao tiếp bộ hiển thị (Display)
  - Giao tiếp với LED
  - Giao tiếp với LCD
- Giao tiếp A/D-D/A

5.1 Giải mã địa chỉ

- Khi vi xử lý gửi một địa chỉ ra bus địa chỉ, thì thông tin này phải được chuyển thành lệnh cụ thể cho thiết bị cụ thể. **Giải mã địa chỉ** thực hiện tác vụ này. Nó sử dụng thông tin bus địa chỉ để xác định thiết bị nào sẽ được truy cập.



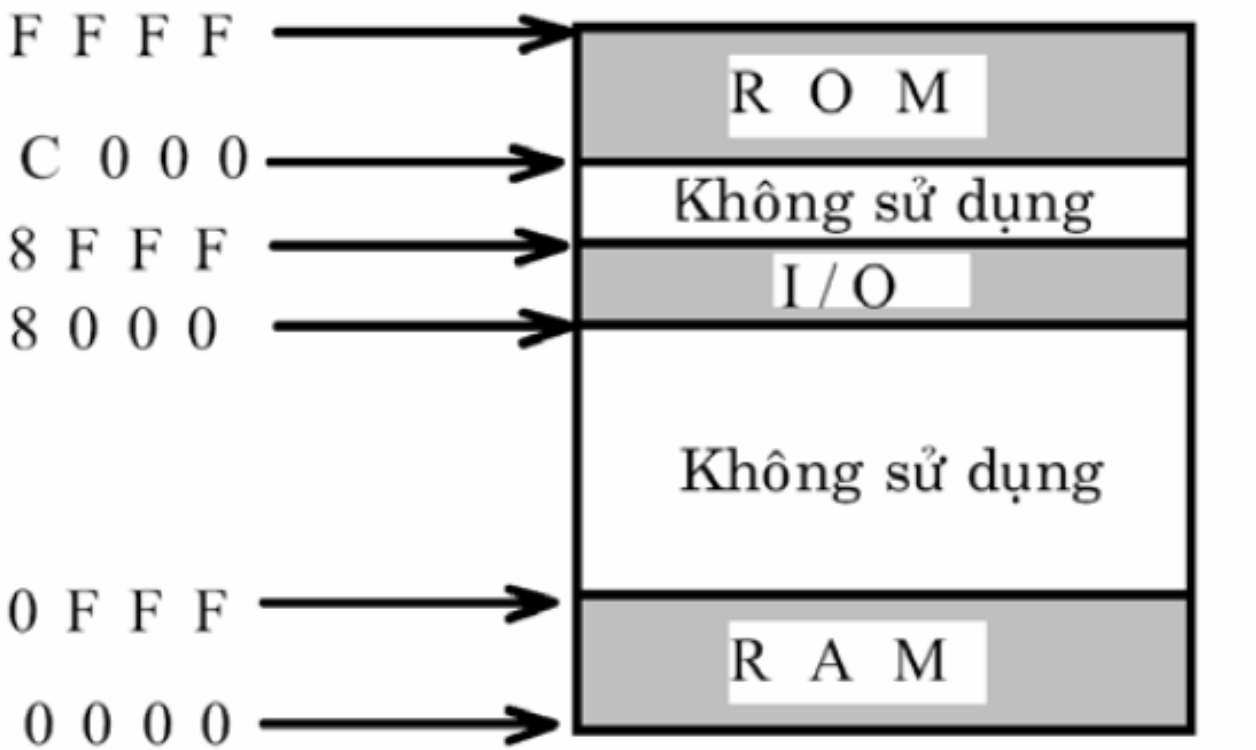
Phương pháp giải mã địa chỉ

Có 2 phương pháp giải mã địa chỉ:

1. Giải mã toàn phần (Full address decoding): Mỗi ngoại vi được gán cho một địa chỉ duy nhất. Tất cả các bit địa chỉ được dùng để định nghĩa vị trí được tham chiếu.
2. Giải mã một phần (Partial address decoding): Không phải tất cả các bit được dùng cho việc giải mã địa chỉ. Các ngoại vi có thể đáp ứng cho trên một địa chỉ. Phương pháp làm giảm độ phức tạp trong mạch giải mã địa chỉ. Thông thường các hệ thống nhỏ sử dụng giải mã một phần.

Thí dụ: TK mạch giải mã địa chỉ

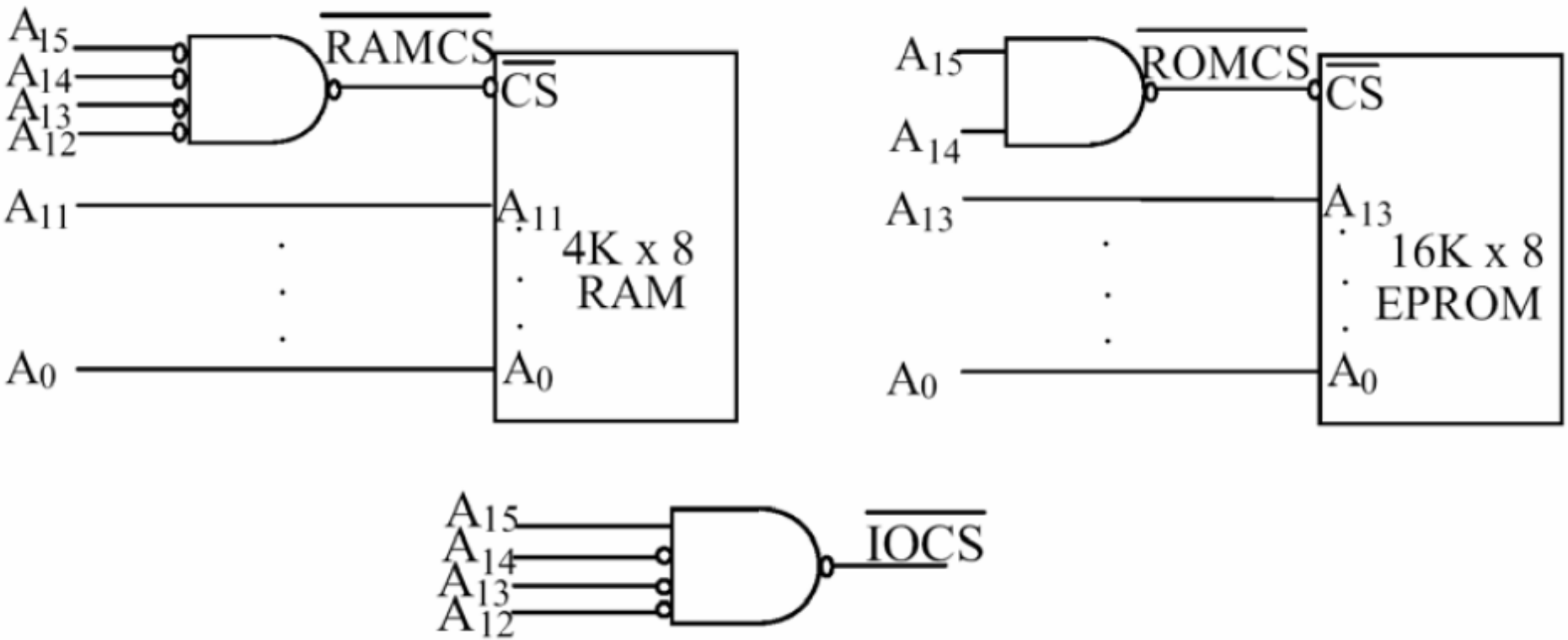
- Với bảng bộ nhớ sau, hãy thiết kế mạch giải mã toàn phần và mạch giải mã một phần cho các chip bộ nhớ (RAM 4KB và ROM 16KB) và I/O



Bảng bộ nhớ/IO của TD

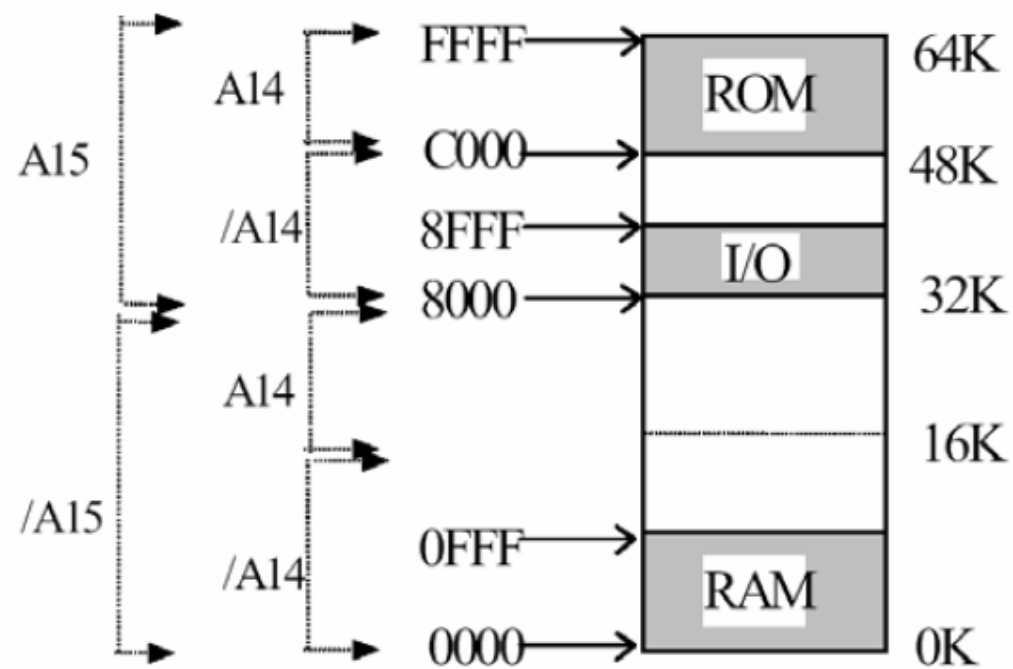
RAM	$A_{15}A_{14}A_{13}A_{12}$	$A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$
start	0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
end	0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1
	0 0 0 0	**x x x x x x x x x x x x**
I/O	$A_{15}A_{14}A_{13}A_{12}$	$A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$
start	1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
end	1 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1
	1 0 0 0	**x x x x x x x x x x x x**
ROM	$A_{15}A_{14}A_{13}A_{12}$	$A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$
start	1 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0
end	1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1
	1 1 x x	**x x x x x x x x x x x x**

Thí dụ giải mã địa chỉ toàn phần



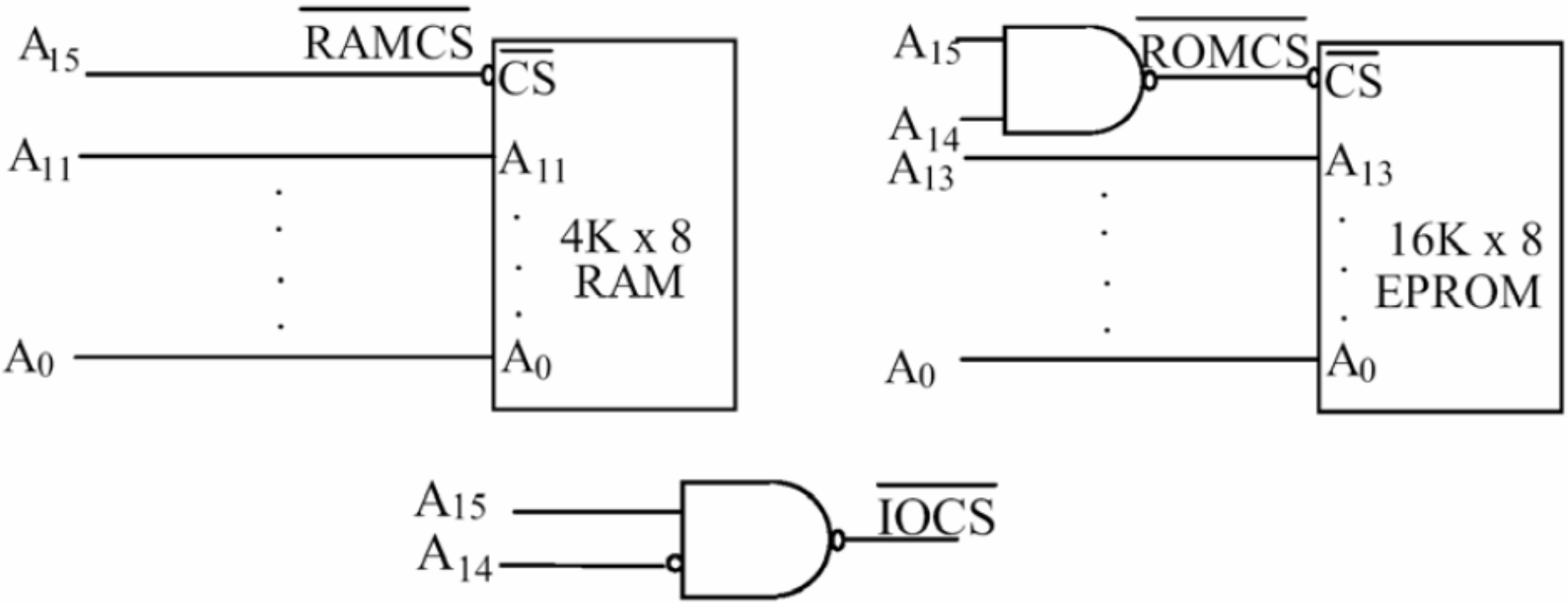
Kỹ thuật giải mã một phần

Ta có thể vẽ thêm vào bảng bộ nhớ như hình 5.4 để thấy rõ các bit địa chỉ ảnh hưởng đến tầm địa chỉ của bộ nhớ và I/O.



Hình 5.4 Bảng bộ nhớ với các bit địa chỉ xác định với bộ nhớ hay I/O.

Thí dụ giải mã địa chỉ một phần



kỹ thuật giải mã một phần dẫn đến:

1. Mạch giải mã đơn giản hơn.
2. Nhiều vùng địa chỉ hơn được cấp phát với tầm địa chỉ một phần. Thí dụ mạch giải mã cho RAM chỉ kiểm tra A15 và kết quả là toàn bộ tầm địa chỉ từ 0000H đến 7FFFH được cấp phát cho RAM.
3. Sự bất lợi của giải mã địa chỉ một phần là bất cứ khai triển nào thêm về bộ nhớ hoặc thiết bị I/O thì cần thiết kế lại mạch giải mã địa chỉ. Trái lại, với mạch giải mã toàn phần thì bất cứ sự thêm vào bộ nhớ hoặc thiết bị I/O, ta không cần sửa đổi mạch giải mã địa chỉ.

Hardware dùng cho mạch giải mã địa chỉ

- Cổng logic → mạch phức tạp
- Mạch giải mã (Decoder) – TD:74LS138,...
- Mạch so sánh (Comparator)
- ROM → thừa chức năng
- PLD: PLA, PAL,... → cho đáp ứng nhanh

5.2 Giao tiếp bộ nhớ

- Vi xử lý sử dụng bộ nhớ để lưu trữ các lệnh và dữ liệu trong khi thực thi chương trình. Do đó vi xử lý thực hiện nhiều tác vụ ghi/đọc với bộ nhớ trong khi thực thi chương trình.
- Mỗi chip bộ nhớ RAM hay ROM sẽ có một ngõ vào có tên là /CE (Chip Enable=cho phép chip [hoạt động]) hoặc /CS (Chip Select=Chọn chip), thông thường các chân này hoạt động logic tích cực thấp, nghĩa là chân này bằng mức 0 thì chip này được chọn.
- Vì trong mạch có nhiều thiết bị I/O và bộ nhớ, do đó cần phải có mạch giải mã địa chỉ để tạo ra các tín hiệu chọn chip.
- Bus điều khiển có các tín hiệu định thì (do vi xử lý cung cấp) để đồng bộ chuyển thông tin giữa vi xử lý và bộ nhớ hay thiết bị I/O. Tổng quát thì có 2 tín hiệu RD (Read) và WR (Write), hai tín hiệu này thông thường cũng hoạt động logic tích cực thấp. Ngoài ra tùy theo vi xử lý còn có thêm các tín hiệu khác như ALE, PSEN,...

Một số chân điều khiển bộ nhớ

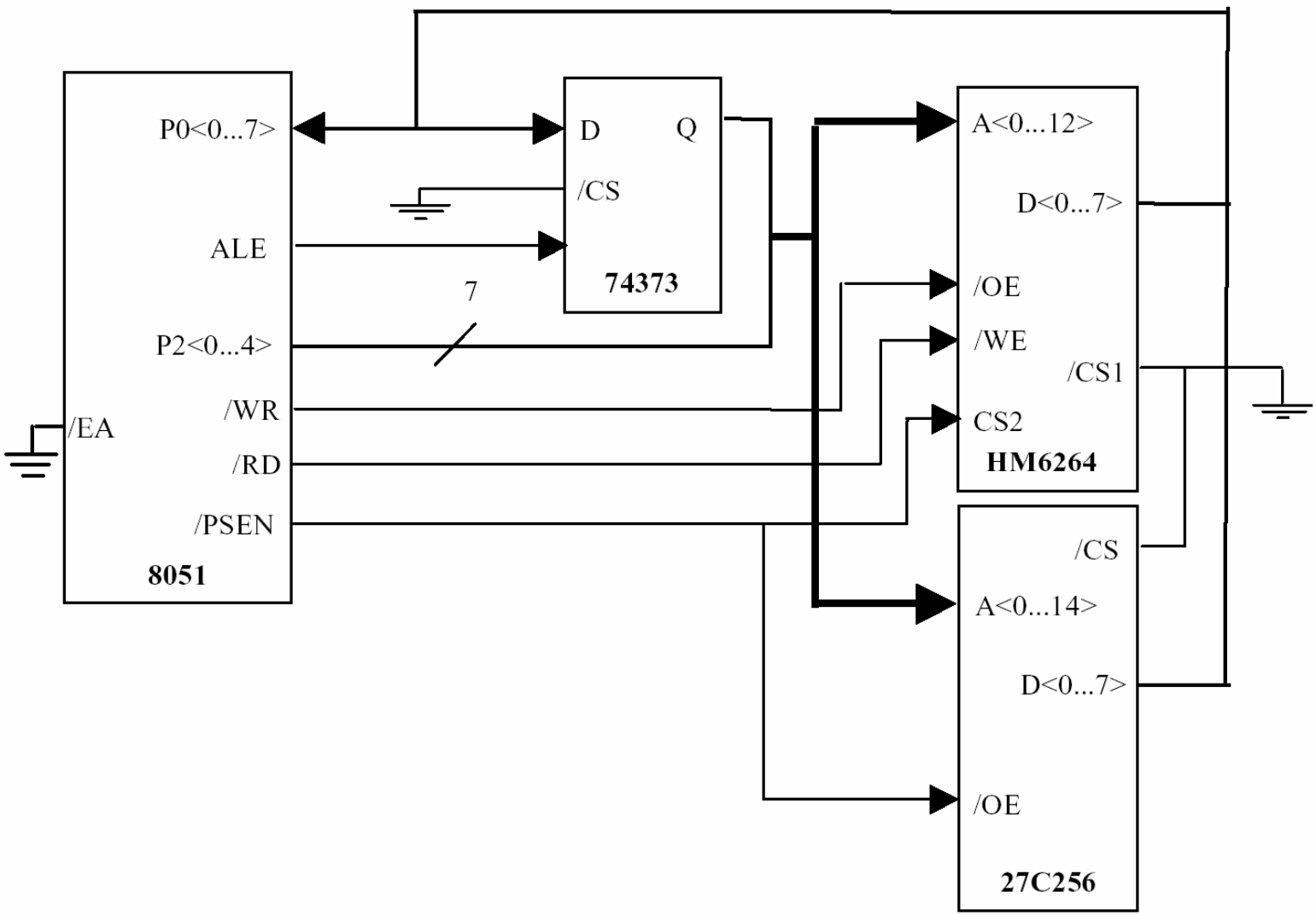
Bộ nhớ (và hầu hết các thiết bị ngoại vi) có các đường điều khiển đặc biệt để giao tiếp với vi xử lý

- **/CS hoặc /CE** (Chip Select hoặc Chip Enable)
  - được lái bởi mạch giải mã địa chỉ từ vi xử lý.
  - thường là tích cực thấp
  - khi được xác định, thì chip/ngoại vi được chọn.
- **/OE** (Output Enable=cho phép xuất) hay **/RD** (với RAM)
  - thường thấy trong các bộ nhớ
  - khi nó tích cực (thường là tích cực thấp) thì ngõ ra ở trạng thái hi-Z.
  - đôi khi nó được xem như /RD trong RAM.
- **/WR** (Write Enable=cho phép ghi)
  - được lái bởi /WR của vi xử lý

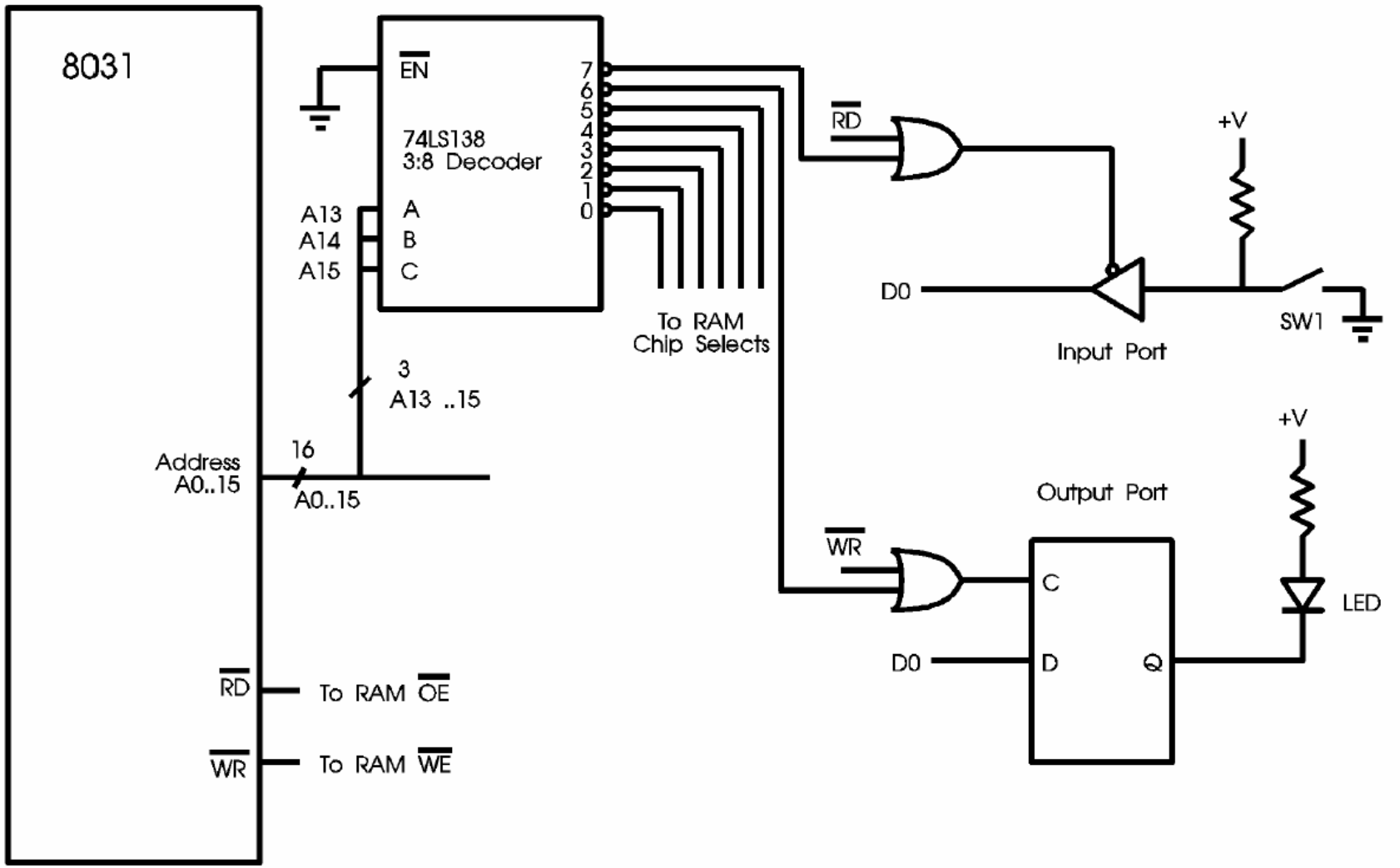
Qui tắc chung về thiết kế mạch giao tiếp bộ nhớ với 8051

1. Lập bảng bộ nhớ
   - Hệ thống và ứng dụng phụ thuộc
   - Giải pháp đơn giản là tách riêng 64KB bộ nhớ dữ liệu và 64KB bộ nhớ mã.
   - Để sử dụng các địa chỉ trên 64K thì sử dụng thêm các bit từ những cổng I/O không sử dụng làm các đường địa chỉ.
2. Chọn linh kiện bộ nhớ thích hợp (nếu không bị áp đặt sử dụng).
3. Sử dụng mạch giải mã địa chỉ (nếu cần) để tạo ra các tín hiệu /CE hay /CS cho các chip bộ nhớ.
4. Sử dụng đường /PSEN (từ 8051) cho bộ nhớ mã.
5. Sử dụng các đường /RD, /WR cho bộ nhớ dữ liệu.
6. Chân /EA=VCC (+5V) để sử dụng ROM trong 8051, cho /EA=GND (0V) để truy cập ROM ngoài.

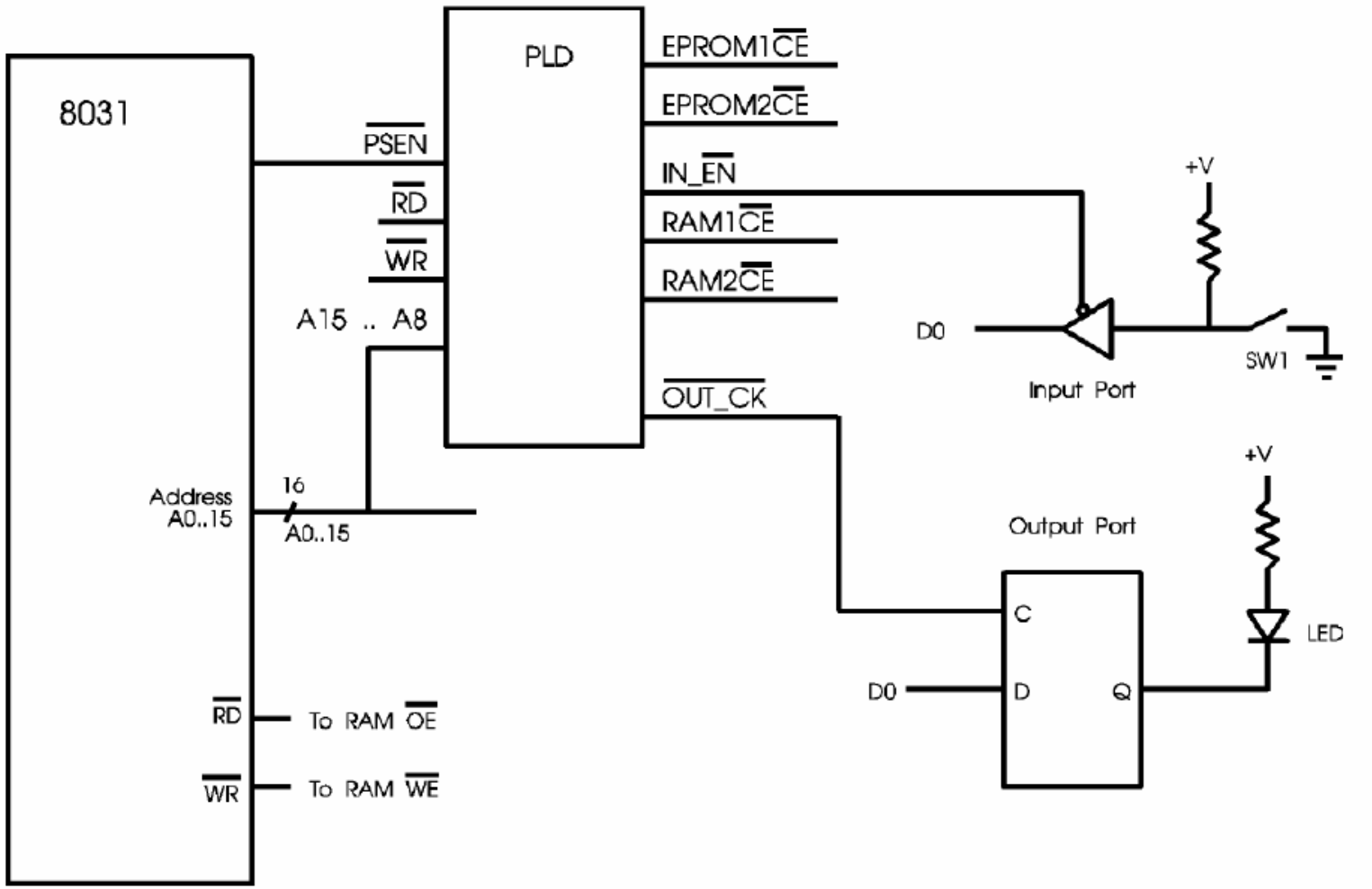
TD: Giao tiếp 8051 với RAM HM 6264 và ROM 27C256



TD: 8031/8051 giao tiếp với RAM ngoài và I/O



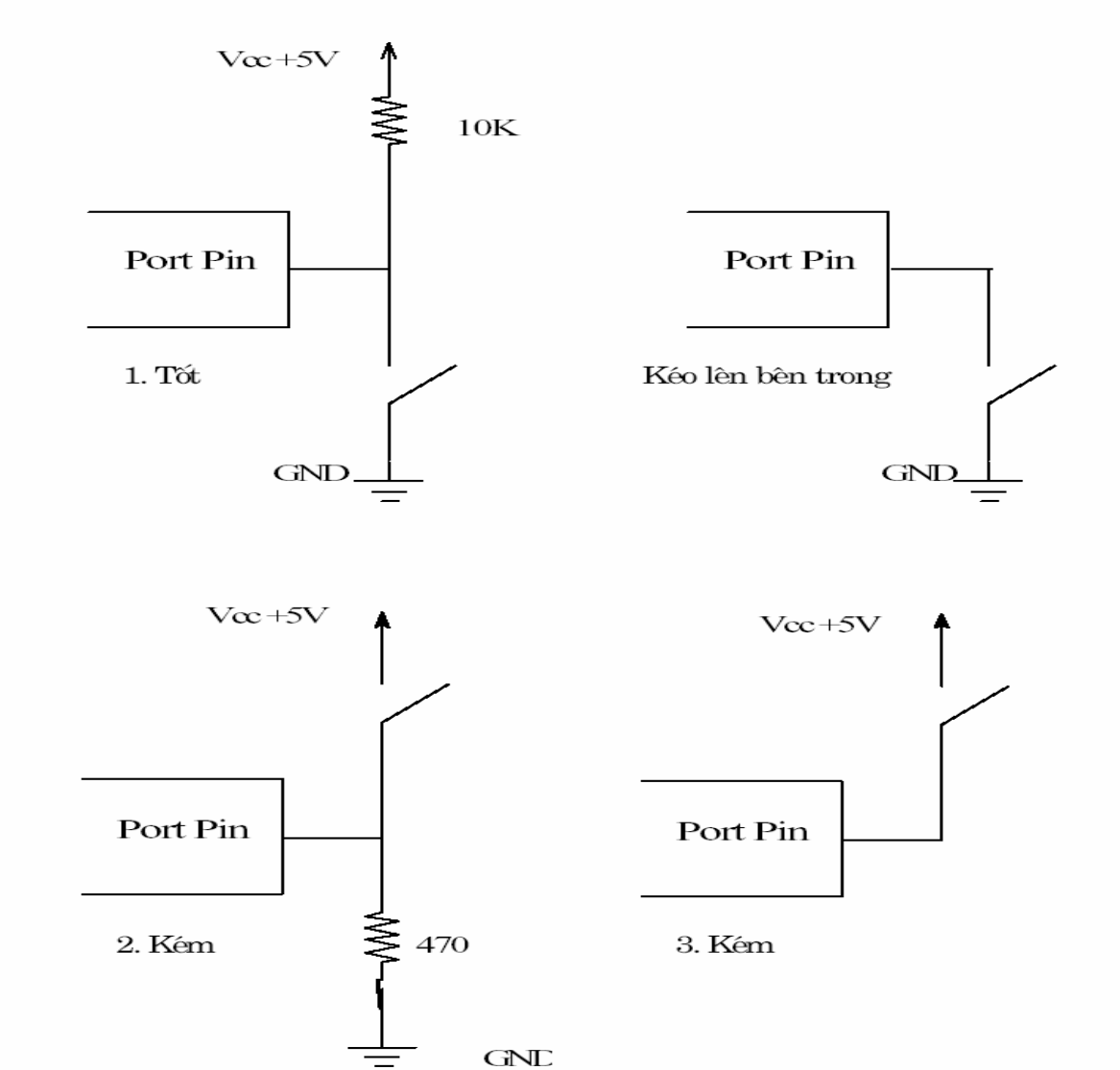
Giao tiếp bộ nhớ với mạch giải mã địa chỉ là PLD



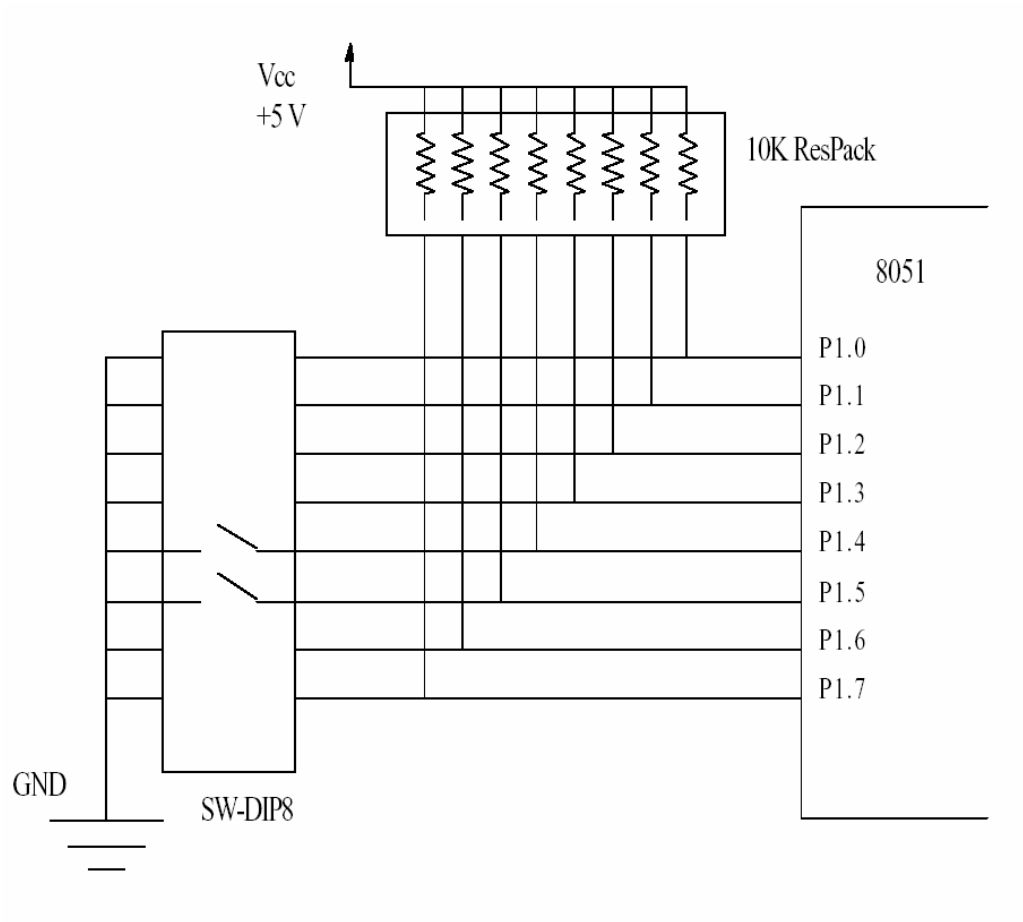
Hình 5.18 Giao tiếp bộ nhớ với mạch giải mã địa chỉ là PLD.

5.5 Giao tiếp với khóa (switch) và bàn phím

Công tắc/khóa (hay phím đơn) ở các chân cổng I/O



Sử dụng DIP switch ở cổng I/O

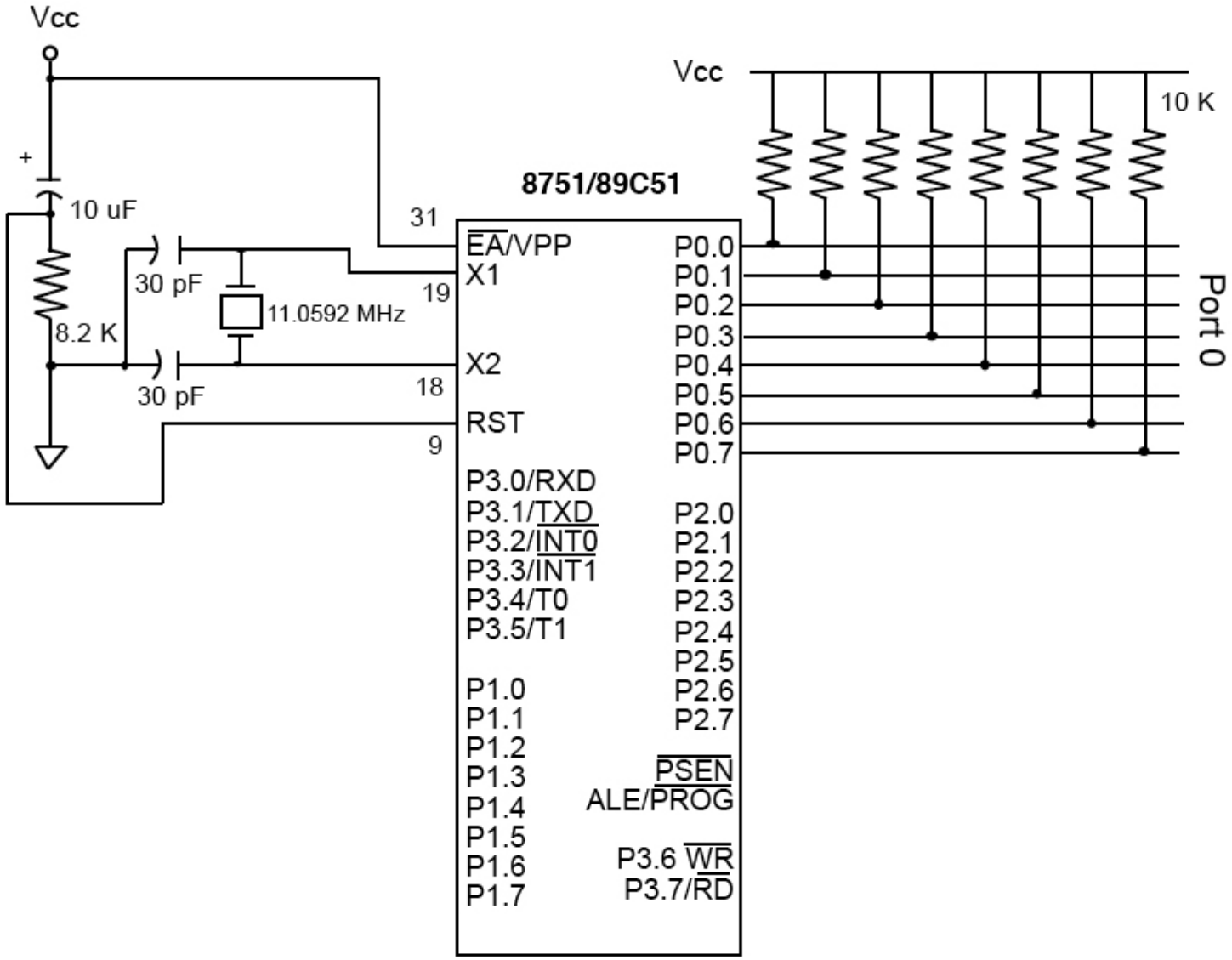


Đoạn chương trình đọc dữ liệu vào:

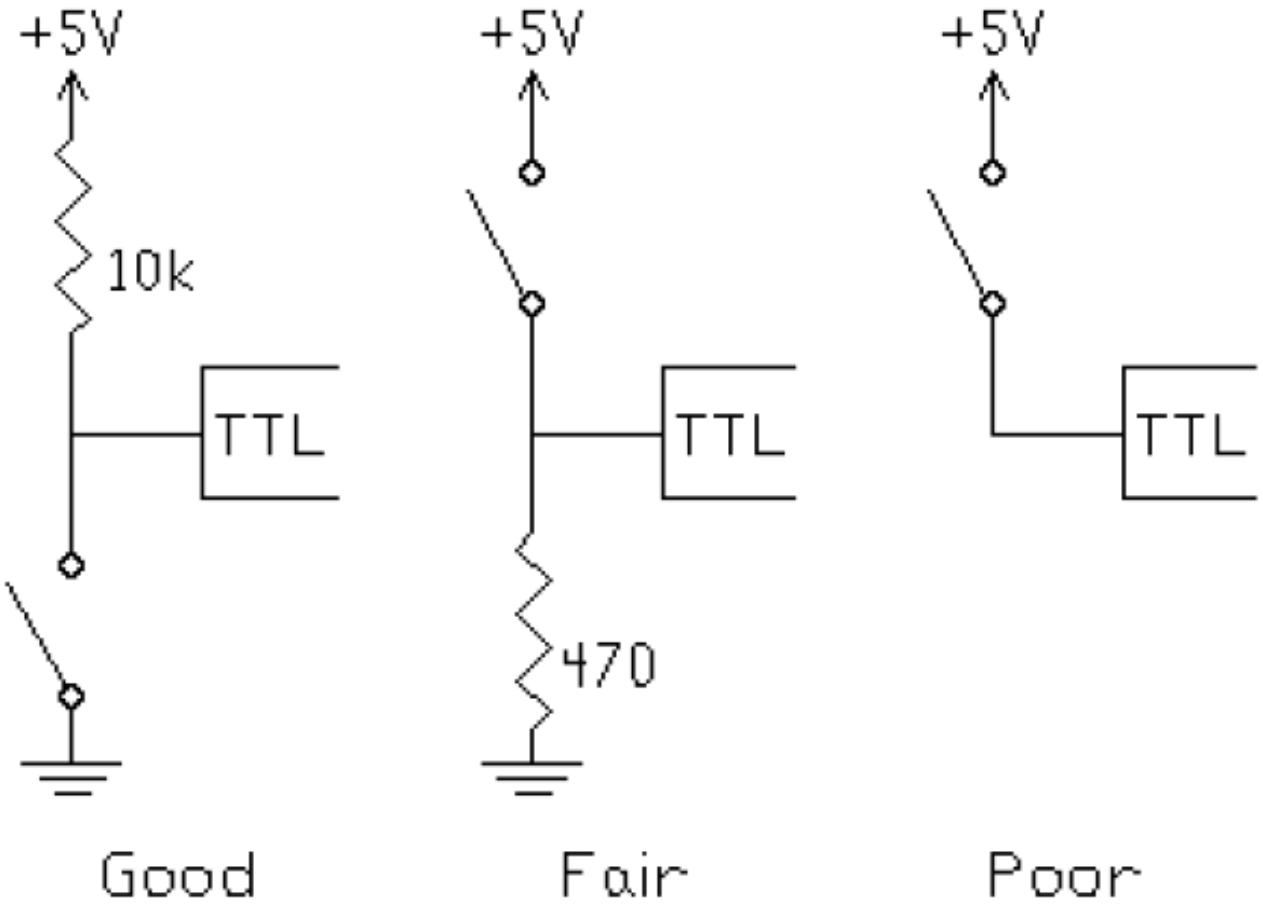
mov **P1,#0FFH** ; đặt cấu hình nhập cho cổng P1

mov **A, P1**

Kết nối tối thiểu cho các hệ thống dùng 89C51/52



SWITCH ON I/O PORTS (1/2)



SWITCH ON I/O PORTS (2/2)

•**Good Circuit**

It is always best connecting the switch to ground with a pull-up resistor as shown in the "Good" circuit. When the switch is open, the 10k resistor supplies very small current needed for logic 1. When it is closed, the port pin is short to ground. The voltage is 0V and all the sinking current requirement is met, so it is logic 0. The 10k resistor will pass 0.5 mA (5 Volt/10k ohm). Thus the circuits waste very little current in either state. The drawback is that the closure of switch gives logic 0 and people like to think of a switch closure gives logic 1. But this is not a matter because it is easy to handle in software.

•**Fair circuit**

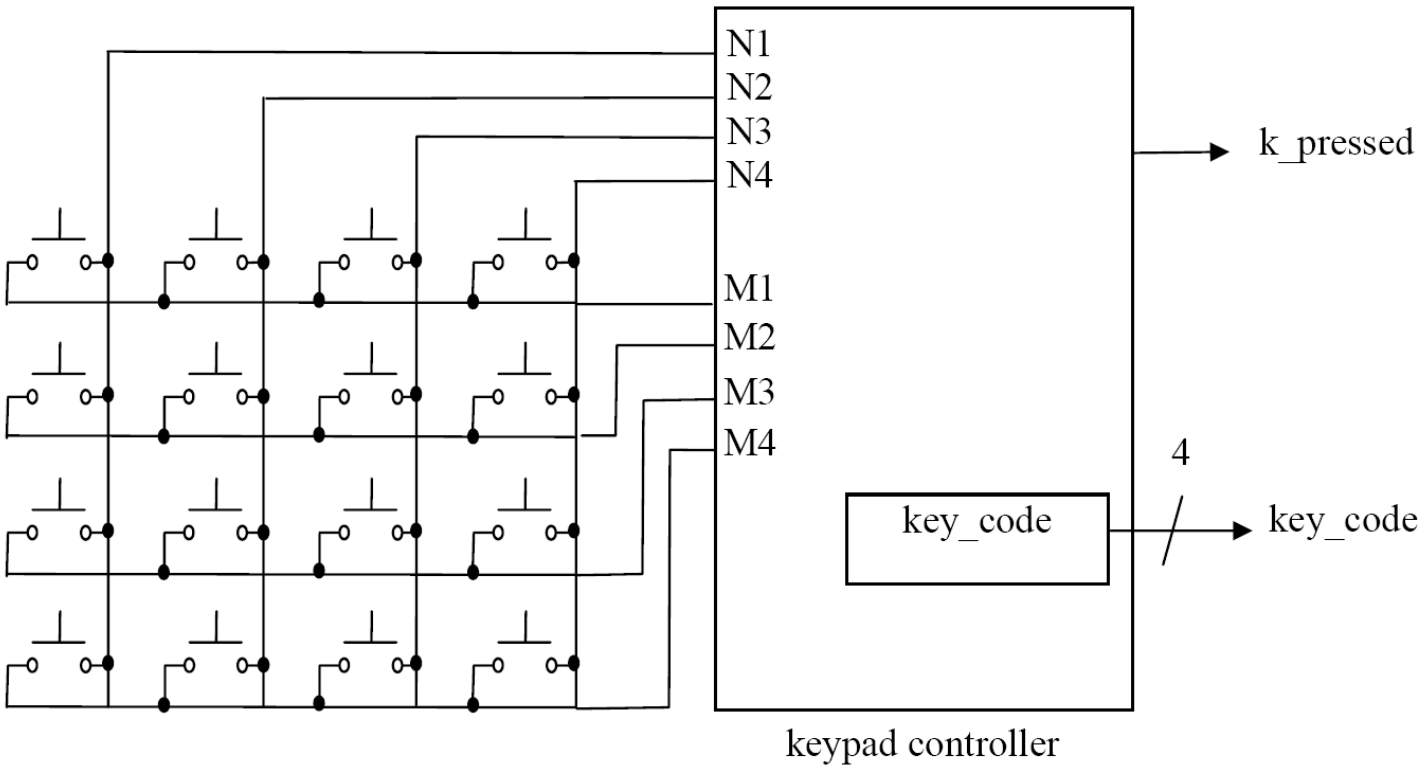
The "Fair" circuit requires that the pull-down resistor be very small. Otherwise, the pin will rise above 0.9V when the resistor passes the 1.6mA sinking current. When the switch is closed, the circuit waste a large current since virtually no current flows into the pin. The only advantage is that a switch closure gives logic 1.

•**Poor circuit**

In the "Poor" circuit, the logic 1 is stable when the switch is closed. But when the switch is open, the input floats to a noise-sensitive high rather than a low. An open TTL pin is usually read as logic 1 but the pin may picks up noise like an antenna.

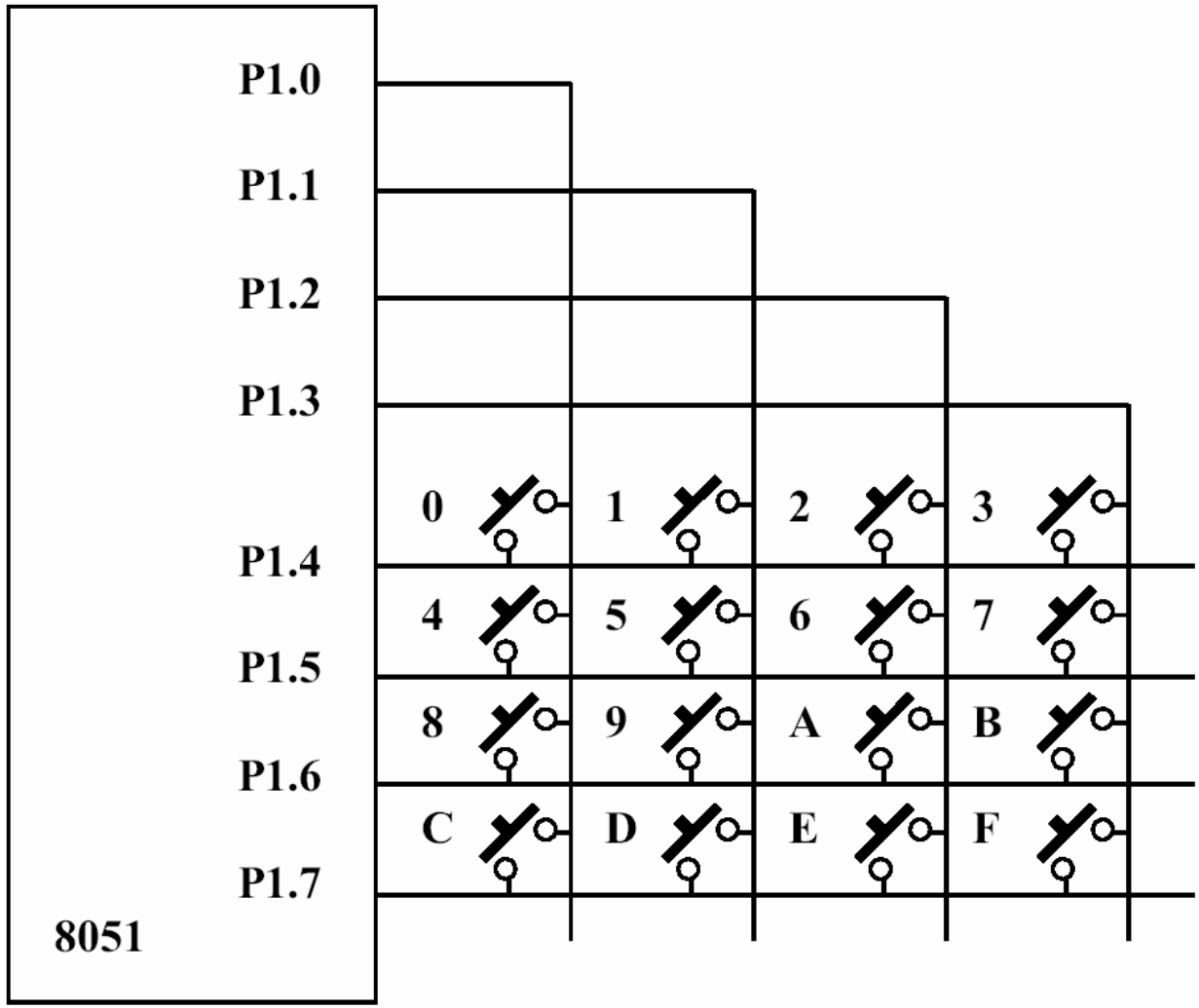
To conclude, driving a TTL input should always consider current sinking (pulling input to 0V).

Keypad controller



N=4, M=4

Giao tiếp với bàn phím hex

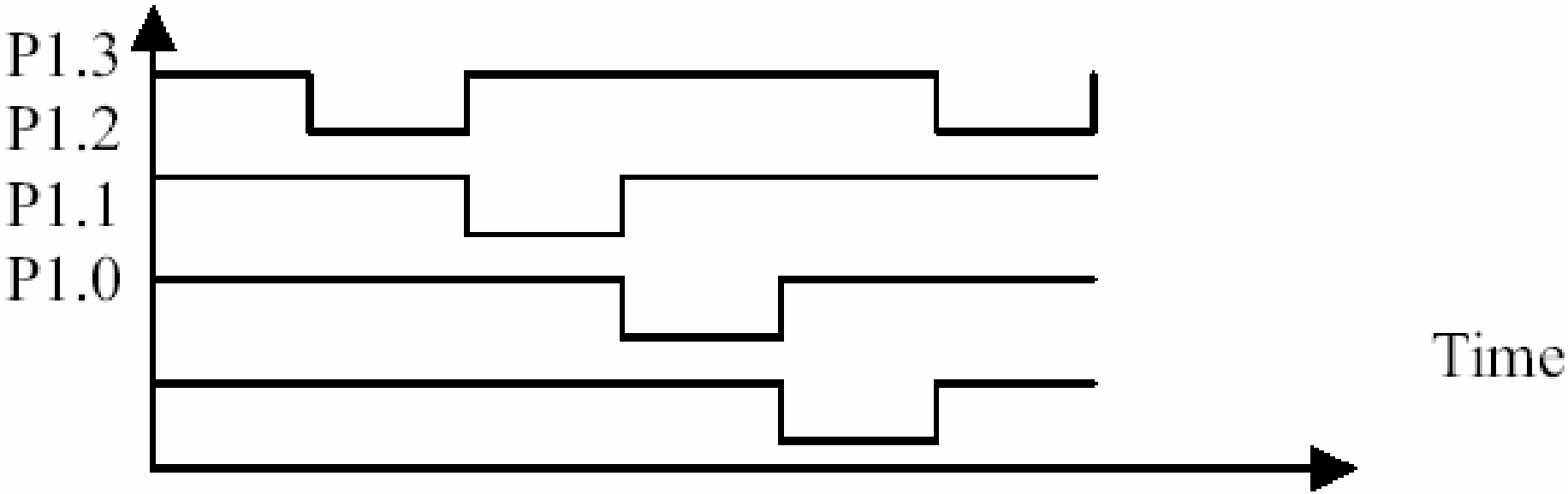


Giao tiếp với bàn phím hex

Để đọc toàn bộ bàn phím, mỗi hàng lần lượt được đọc vào như sau:

1. Dữ liệu 0111 được ghi ra các bit cổng xuất (P1.0 đến P1.3).
2. Các bit cổng nhập (P1.4 đến P1.7) được đọc vào. Nếu không có phím nào được nhấn trên hàng đó thì giá trị đọc vào sẽ là 1111. Nếu có bất cứ phím nào được nhận trên hàng đó thì sẽ có 0 ở bit tương ứng.
3. Dữ liệu 1011 được ghi vào cổng xuất bằng cách dịch 0 vào cột kế; và cổng nhập được đọc vào.
4. Dữ liệu 1101 được ghi vào cổng xuất; và cổng nhập được đọc vào.
5. Dữ liệu 1110 được ghi vào cổng xuất; và cổng nhập được đọc vào.
6. Chu trình này được lặp đi lặp lại vô tận bằng cách quay về bước 1 ở trên.

Giao tiếp với bàn phím hex



Dạng sóng ở các bit cổng xuất P1.0 đến P1.3

Chương trình quét bàn phím

```
; Chương trình quét bàn phím
        org 2000H
start:  LJMP    init            ; nhảy đến chương trình chính
scan:   MOV     P1, #0F0H       ; kiểm tra với các phím nhấn
        MOV     A, P1                   ; bằng cách quan sát P1
        CJNE    A, #0F0H, scan
scanner:
        MOV     A, #0FEH        ; khuôn mẫu bit bắt đầu (4 bit thấp là 1110)
1up:    MOV     R0, A                   ; cất khuôn mẫu xuất
        MOV     P1, A                   ; mẫu xuất
        MOV     A, P1                   ; đọc cổng
        MOV     R1, A                   ; cất phím đọc được
        ORL     A, #0FH         ; đặt các bit ở cột lên 1
        CJNE    A, #0FFH, cnvrt         ; nhảy đến cnvrt nếu phím được nhấn
        MOV     A, R0                   ; lấy lại các bit lái cột
        RL      A                       ; xoay bit lái cột sang trái
        CJNE    A, #0EFH, 1up           ; nhảy đến quét cột kế
        SJMP    scanner         ; bắt đầu đợt quét mới
```

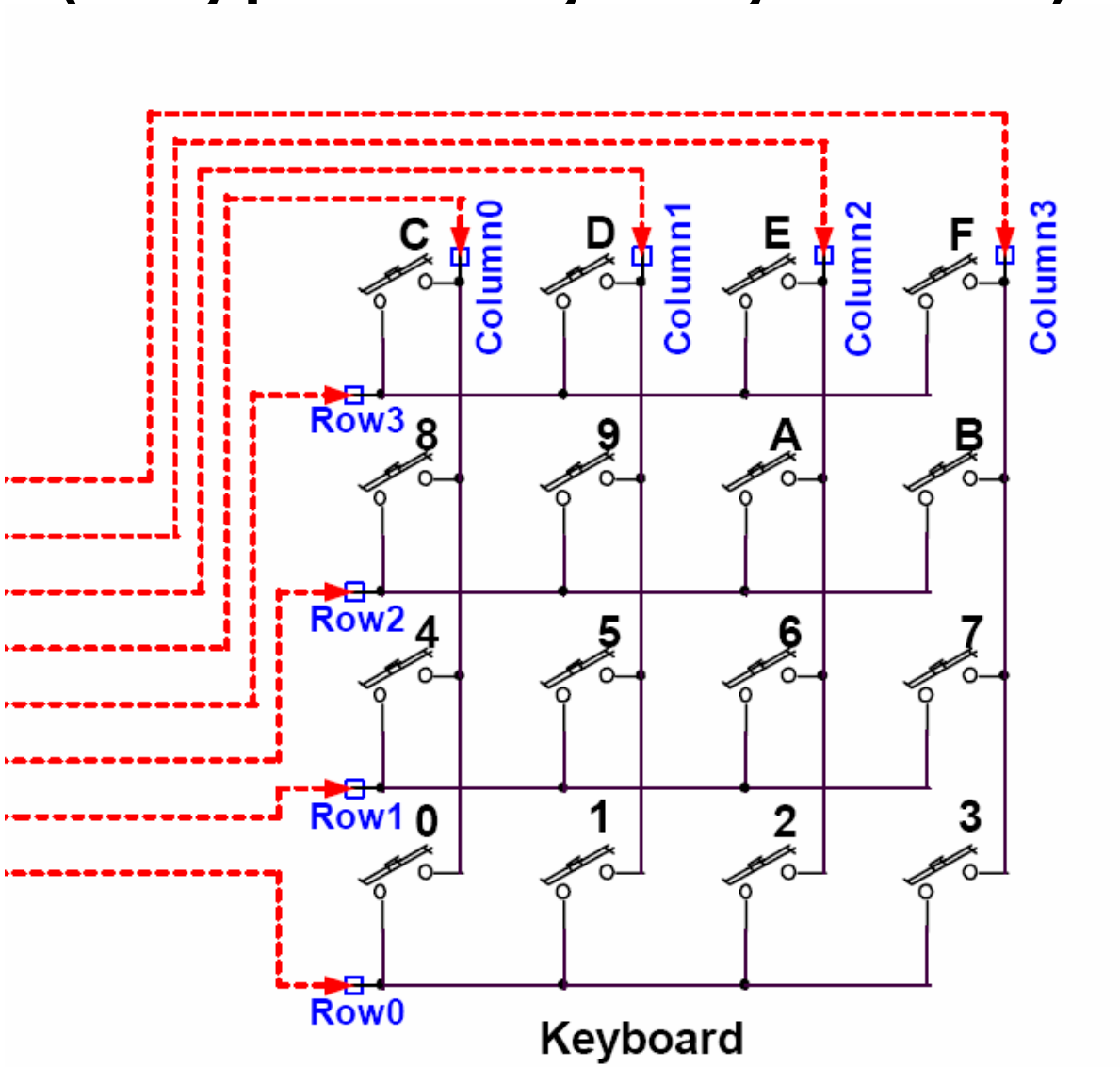
Chương trình quét bàn phím

```
cnvrt:
    MOV     A, R1                   ; khôi phục phím đọc được
    MOV     R3, #0                  ; xóa bộ đếm bảng
clup:  JNB  ACC.0, cnvrt2           ; xong với số đếm chính
    RR      A                       ; xoay giá trị
    MOV     R2, A                   ; cất giá trị được xoay
    MOV     A, R3                   ; tăng bộ đếm thêm 1
    ADD     A, #4                   ; cộng 4 cho mỗi cột
    MOV     R3, A                   ; cất số đếm
    MOV     A, R2                   ; lấy lại giá trị xoay
    SJMP    c1up
cnvrt2: MOV A, R1                   ; khôi phục phím đọc được.
    SWAP    A                       ; hoán đổi 2 nửa byte
c1up2: JNB  ACC.0, xlat; nhảy đến dịch phím đọc được
    RR      A                       ; xoay để tìm bit 0
    INC     R3                      ; tăng số đếm thêm 1
    SJMP    c1up2                   ; nhảy đến kiểm tra bit kế
xlat:  MOV  A, R3                   ; lấy số đếm
    MOV     DPTR, #keytab ; chỉ đến bảng dịch
    MOVC    A, @A+DPTR              ; lấy ký tự được dịch
    RET
```

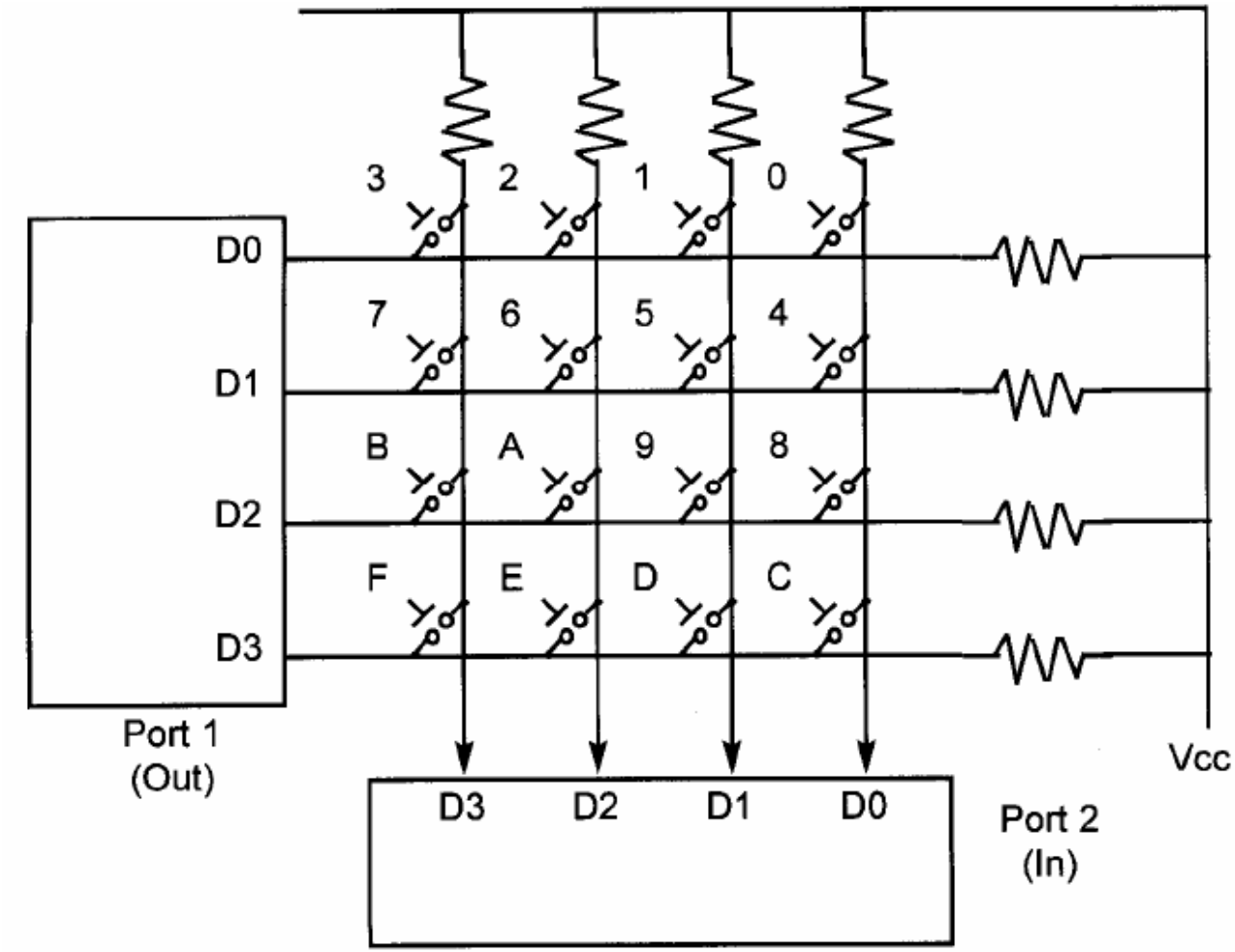
```
DSPLY: RET              ; trình con display giả (cụ thể sẽ đựoc viết đầy đủ)
init:  ACALL   scan     ; lấy phím nhấn
       MOV     R0, A    ; cất ký tự
       CLR     C        ; xóa cờ nhớ để trừ
       SUBB    A, #0AH ; kiểm tra xem có > 10 ?
       JNC     letter   ; nhảy nếu > 10
       MOV     A, R0    ; lấy lại ký tự
       ORL     A, #30H; đổi thành số ASCII
       SJMP    dply     ; nhảy đến hiển thị ký tự
letter:  MOV   A, R0    ; lấy lại ký tự
       ADD     A, #55   ; điều chỉnh cho ASCII
dply:  LCALL  DSPLY; hiển thị ký tự
         SJMP  init     ; làm lại
```

Chú ý là 3 dòng tại vòng lặp “scan” kiểm tra xem có phải tất cả các phím nhấn đã được nhả ra trước khi bắt đầu chu trình quét.

Một dạng bàn phím (keypad hay keyboard)



Interfacing to the Keyboard



– keyboard is organized in a matrix of rows and columns
– the key press is scanned and identified by microcontroller

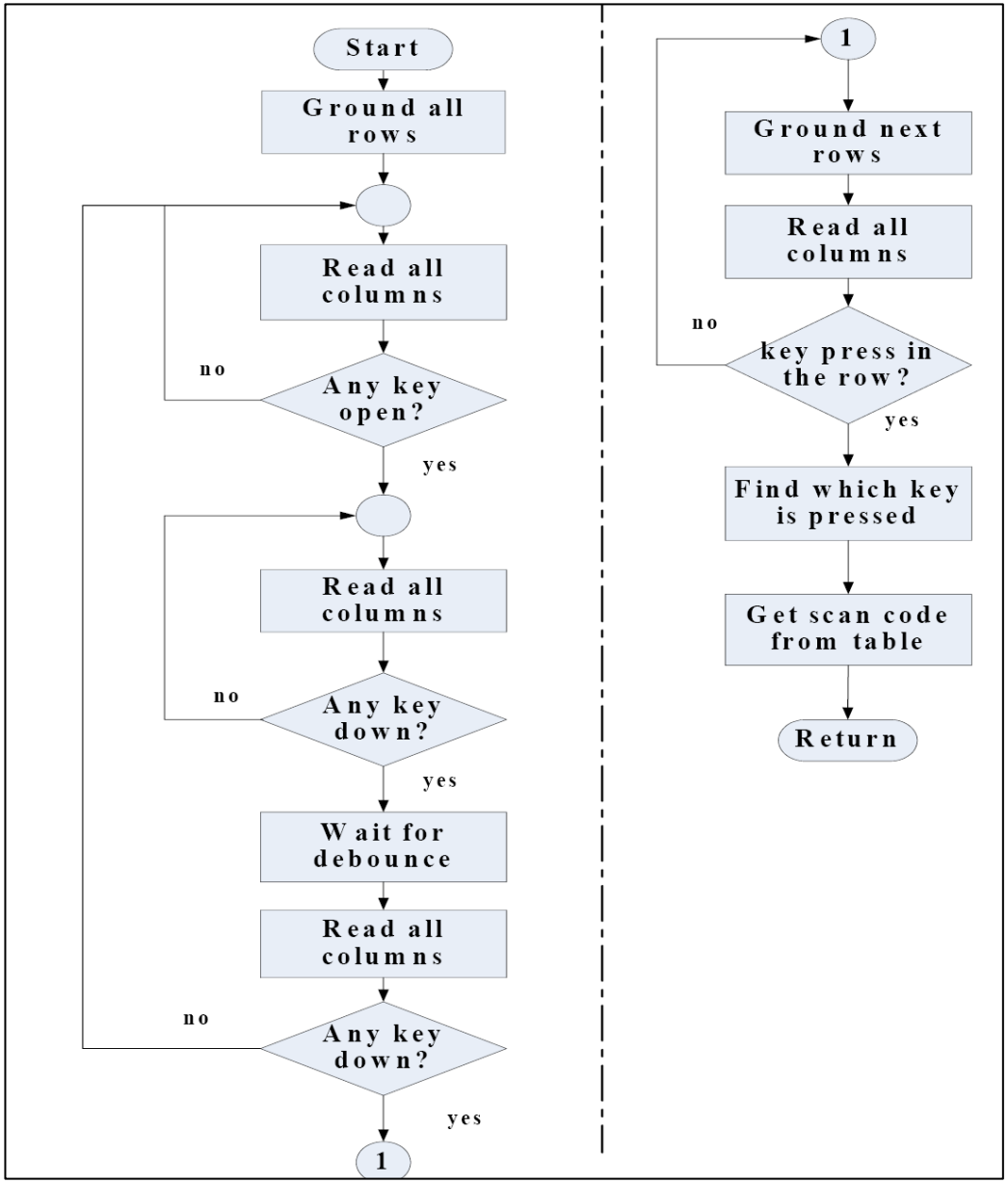
Interfacing to the Keyboard

- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed
- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns
  - If the data read from columns is D3 –D0 = 1111, no key has been pressed and the process continues till key press is detected
  - If one of the column bits has a zero, this means that a key press has occurred
    - For example, if D3 –D0 = 1101, this means that a key in the D1 column has been pressed
    - After detecting a key press, microcontroller will go through the process of identifying the key

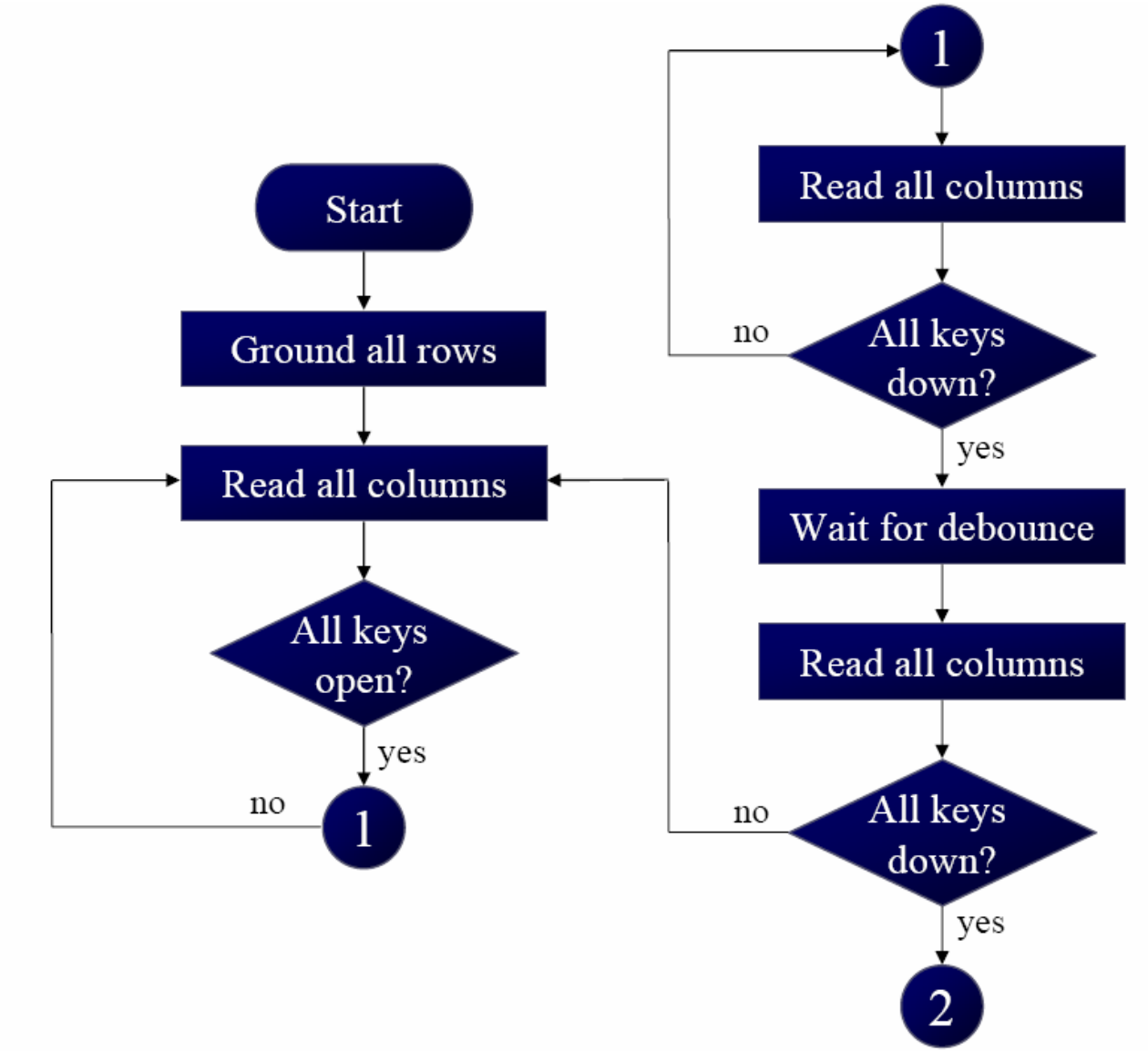
Interfacing to the Keyboard

- **Starting with the top row, the microcontroller grounds it by providing a low to row D0 only**
  - It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row
- **It grounds the next row, reads the columns, and checks for any zero**
  - This process continues until the row is identified
- **After identification of the row in which the key has been pressed**
  - Find out which column the pressed key belongs to

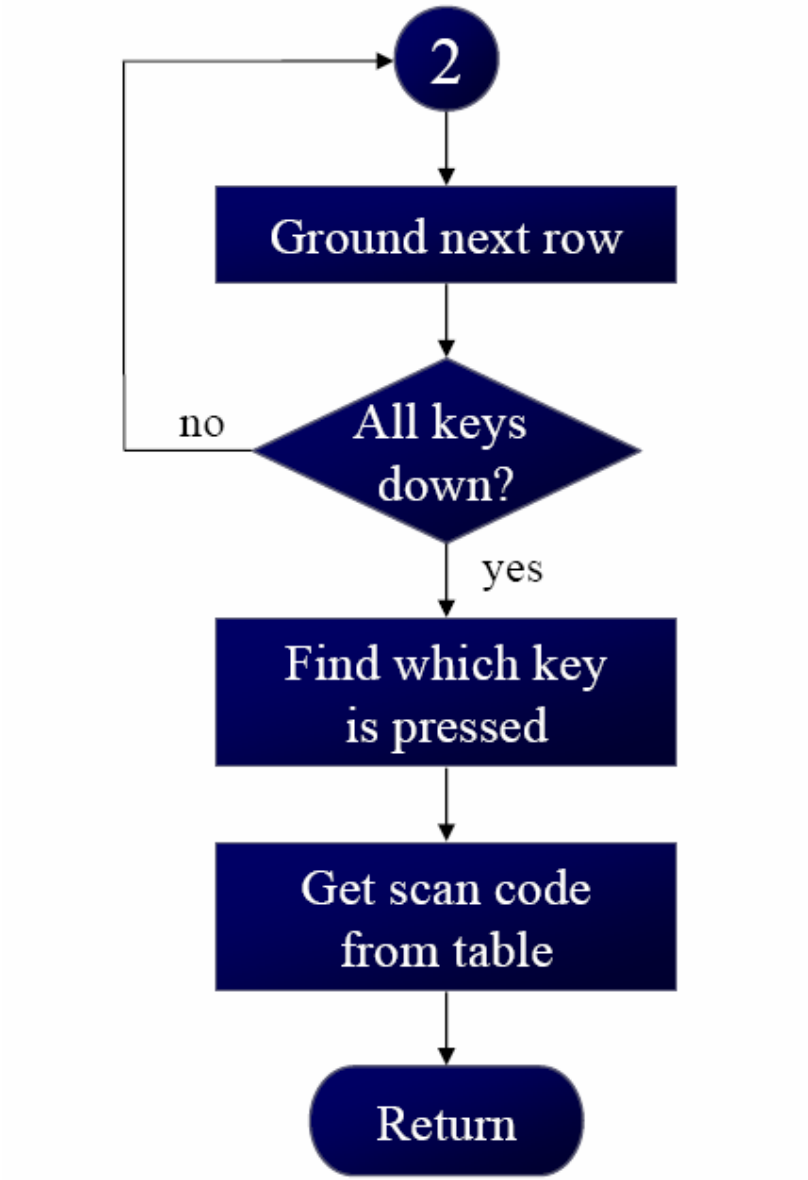
Interfacing to the Keyboard



Interfacing to the Keyboard



Interfacing to the Keyboard



Interfacing to the Keyboard

```
;Keyboard subroutine. This program sends the ASCII code
;for pressed key to P0.1
;P1.0-P1.3 connected to rows P2.0-P2.3 connected to columns
     MOV  P2,#0FFH  ;make P2 an input port
K1:  MOV P1,#0      ;ground all rows at once
     MOV A,P2       ;read all col. (ensure all keys open)
     ANL A,00001111B     ;masked unused bits
     CJNE A,#00001111B,K1     ;check til all keys released
K2:  ACALL DELAY               ;call 20 msec delay
     MOV A,P2                  ;see if any key is pressed
     ANL A,#00001111B          ;mask unused bits
     CJNE A,#00001111B,OVER    ;key pressed, await closure
     SJMP K2                   ;check il key pressed
OVER: ACALL DELAY              ;wait 20 msec debounce time
     MOV A,P2                  ;check key closure
     ANL A,#00001111B          ;mask unused bits
     CJNE A,#00001111B,OVER1   ;key pressed, find row
     SJMP K2                   ;if none, keep polling
```

Interfacing to the Keyboard

```
OVER1: MOV P1,#11111110B        ;ground row 0
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_0  ;key row 0, find the col.
       MOV P1,#11111101B        ;ground row 1
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_1  ;keyrow 1, find the col.
       MOV P1,#11111011B        ;ground row 2
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_2  ;key row 2, find the col.
       MOV P1,#11110111B        ;ground row 3
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_3  ;keyrow 3, find the col.
       LJMP K2                  ;if none, false input, repeat
```

Interfacing to the Keyboard

```
ROW_0: MOV DPTR,#KCODE0        ;set DPTR=start of row 0
       SJMP FIND               ;find col. key belongs to
ROW_1: MOV DPTR,#KCODE1        ;set DPTR=start of row 1
       SJMP FIND               ;find col. key belongs to
ROW_2: MOV DPTR,#KCODE2        ;set DPTR=start of row 2
       SJMP FIND               ;find col. key belongs to
ROW_3: MOV DPTR,#KCODE3        ;set DPTR=start of row 3
FIND:  RRC A                   ;see if any CY bit low
       JNC MATCH               ;if zero, get the ASCII code
       INC DPTR                ;point to next col. address
       SJMP FIND               ;keep searching
MATCH: CLR A                   ;set A=0 (match is found)
       MOVC A,@A+DPTR          ;get ASCII code from table
       MOV  P0,A               ;display pressed key
       LJMP  K1
;ASCII LOOK-UP TABLE FOR EACH ROW
      ORG     300H
KCODE0: DB     '0','1','2','3'          ;ROW 0
KCODE1: DB     '4','5','6','7'          ;ROW 1
KCODE2: DB     '8','9','A','B'          ;ROW 2
KCODE3: DB     'C','D','E','F'          ;ROW 3
      END
```

Interfacing to the Keyboard

Last program for detection and identification of key activation goes through the following stages:

1.To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high

- When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed

Interfacing to the Keyboard

2. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it

- Remember that the output latches connected to rows still have their initial zeros (provided in stage 1), making them grounded
- After the key press detection, it waits 20 ms for the bounce and then scans the columns again
  - (a) it ensures that the first key press detection was not an erroneous one due a spike noise
  - (b) the key press. If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press

Interfacing to the Keyboard

3.To detect which row key press belongs to, it grounds one row at a time, reading the columns each time

- If it finds that all columns are high, this means that the key press cannot belong to that row
  - –Therefore, it grounds the next row and continues until it finds the row the key press belongs to
- Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row

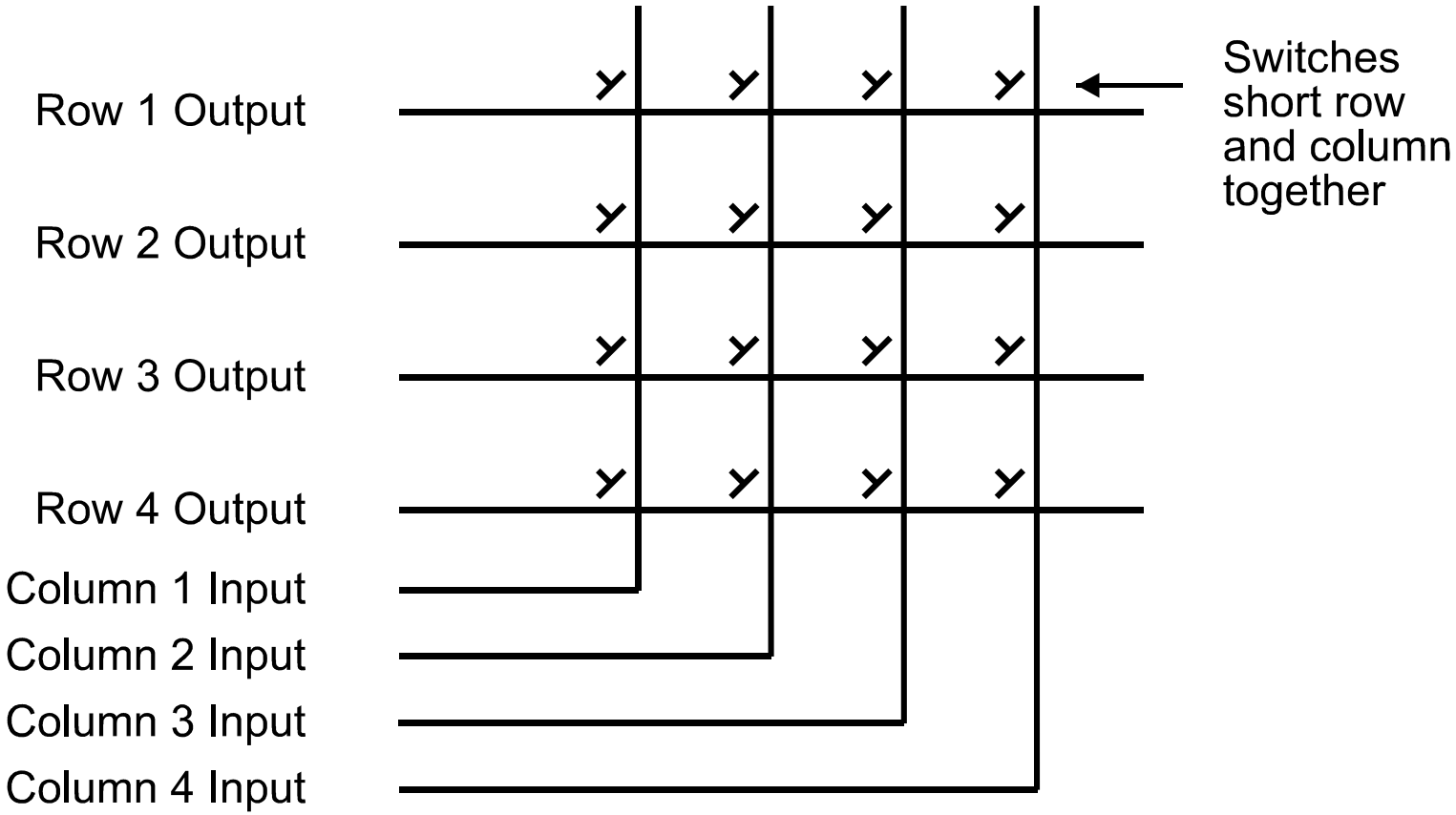
4.To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low

- Upon finding the zero, it pulls out the ASCII code for that key from the look-up table
- otherwise, it increments the pointer to point to the next element of the look-up table

Switch Matrix

- Switches organized as Row/Column
- Switch Shorts row line to column line
- Walking zero on columns to activate one column at a time
- Check for low level on row inputs to determine which key in this column is pressed

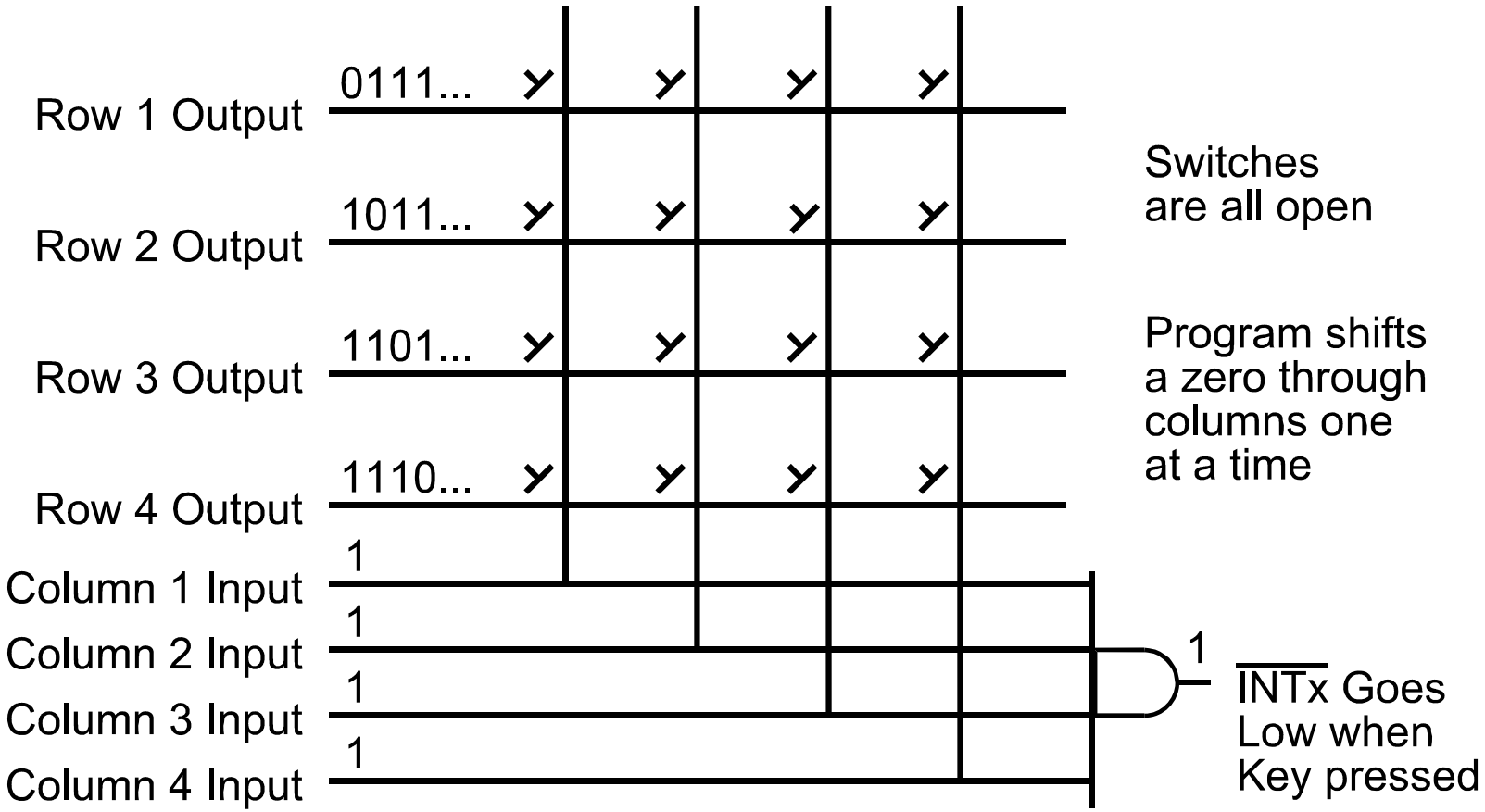
Simple Switch Matrix



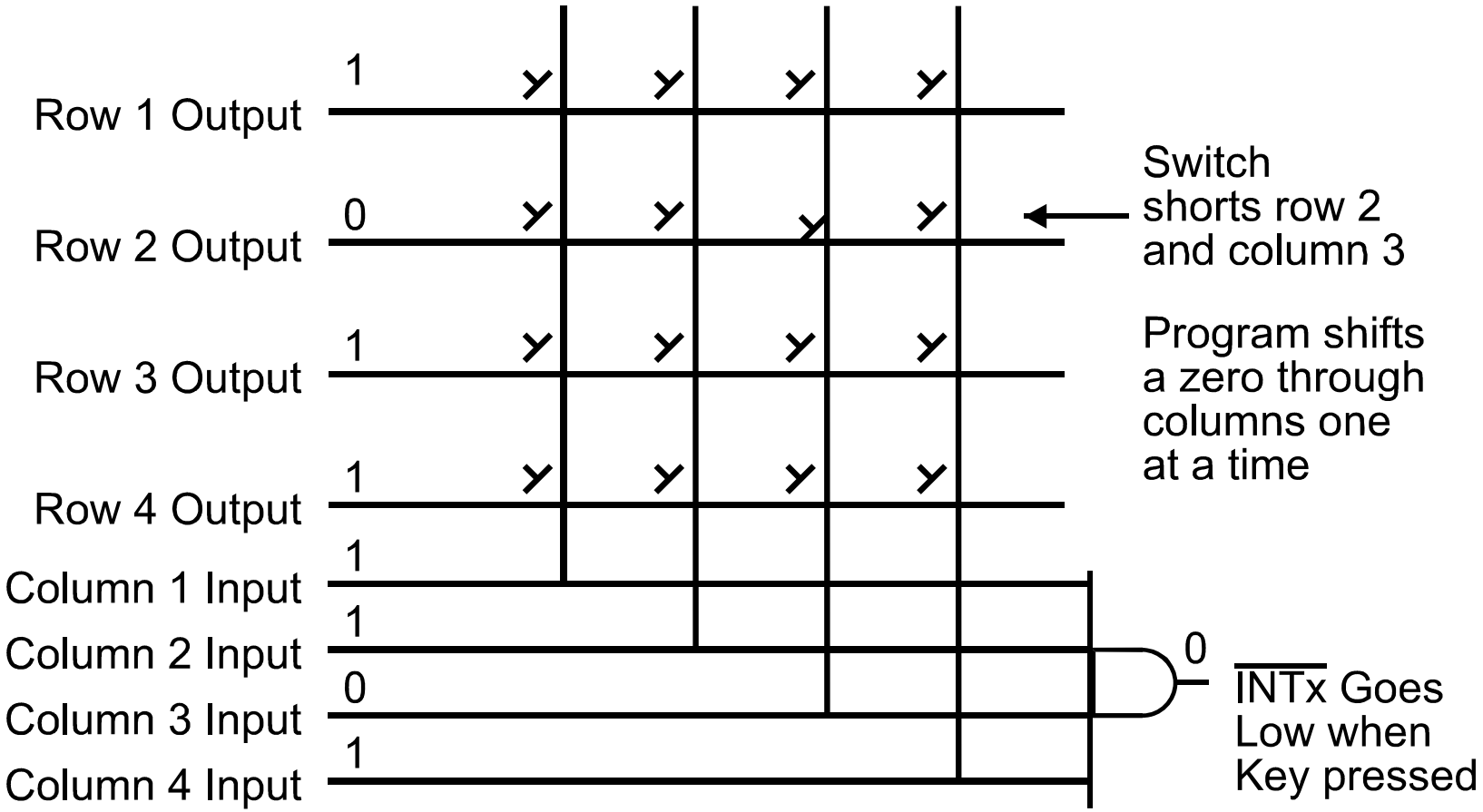
Switch Matrix

- Switches organized in Rows/Columns
- Switch Shorts row line to column line
- Walking zero on columns to activate one column at a time
- Check for low level on row inputs to determine which key is pressed
- Multiplexed row and column

Multiplexing a Switch Matrix



Switch Matrix Key Pressed

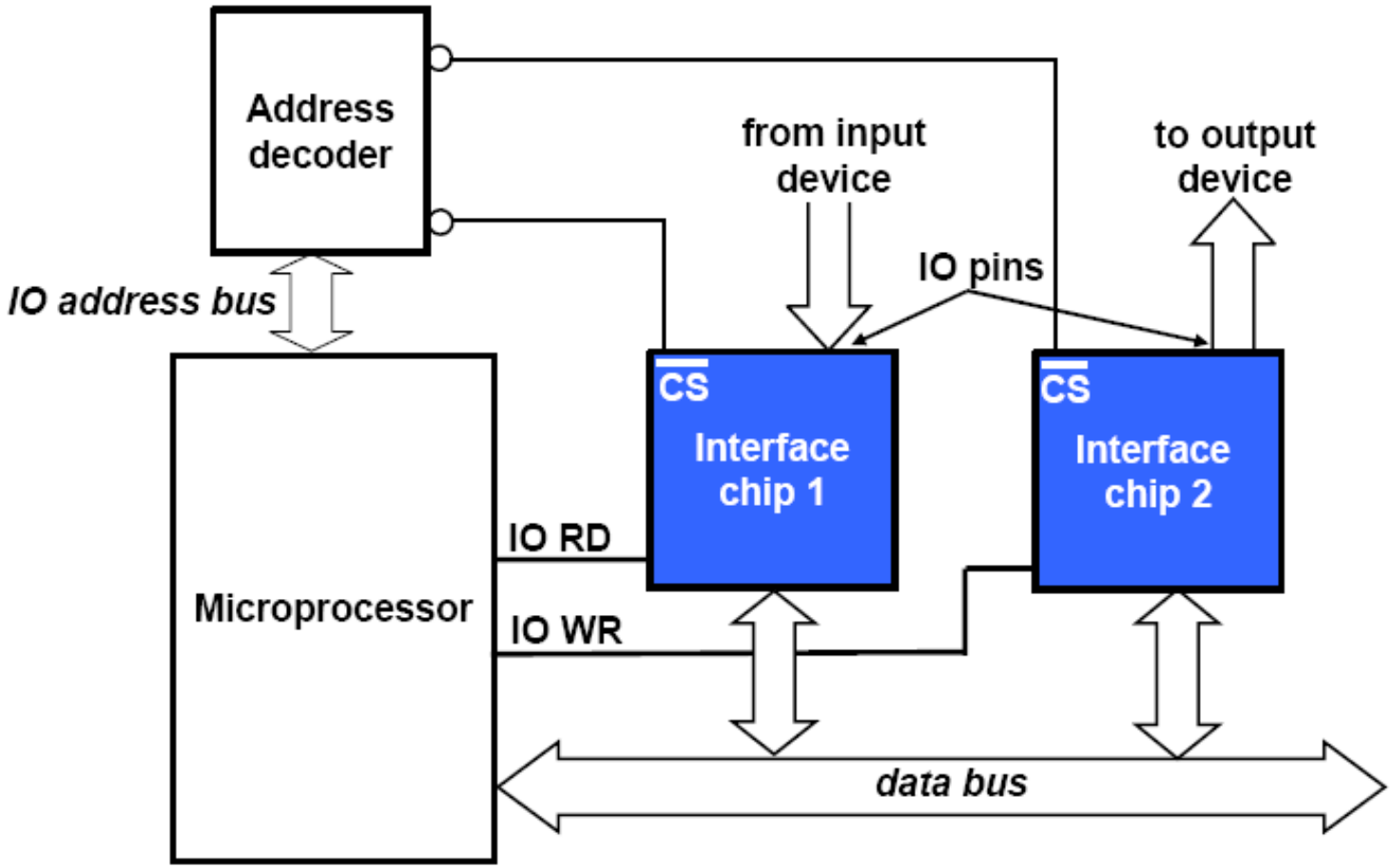


Phần xuất/nhập số với VXL

Giao tiếp với các thiết bị I/O

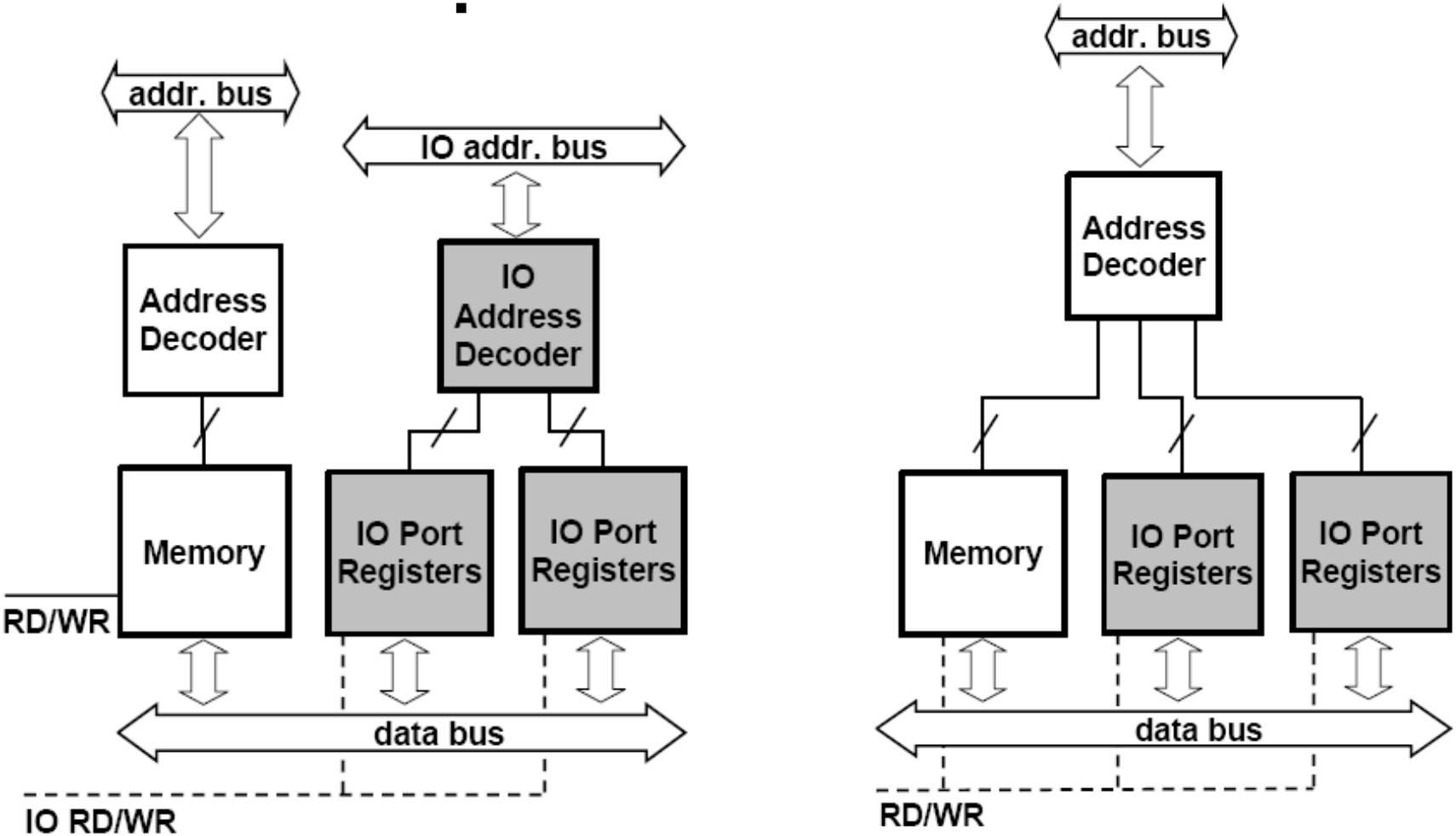
- Truyền dữ liệu được đồng bộ hóa bằng các mạch giao tiếp đgl các cổng IO (nhập-xuất).
  - Một cổng IO có thể được dùng cho nhiều thiết bị khác nhau.
  - Tất cả các cổng IO có thể dùng chung bus.
- Mạch giao tiếp có thể bao gồm
  - Các thanh ghi dữ liệu: dữ liệu được gửi đi hay nhận về
  - Các thanh ghid điều khiển: để chọn kiểu tác vụ IO
  - Các thanh ghi trạng thái: trạng thái của tác vụ IO
- Với cách nhìn VXL thì các thuật ngữ “cổng IO” và “thiết bị IO” đôi khi được dùng cho nhau.

Chip giao tiếp IO



Bus địa chỉ IO có thể trùng hoặc khác bus địa chỉ dùng cho bộ nhớ chính.

Nhắc lại: Có 2 kiểu ánh xa IO

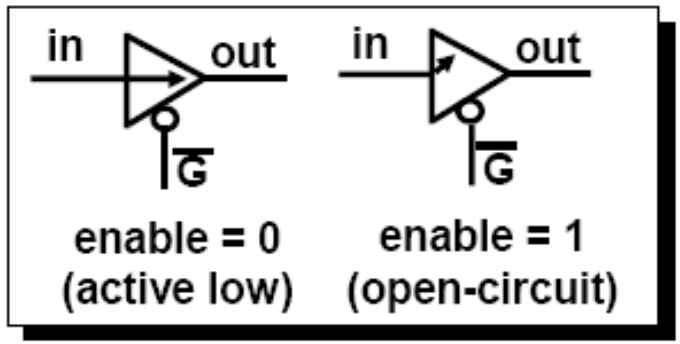


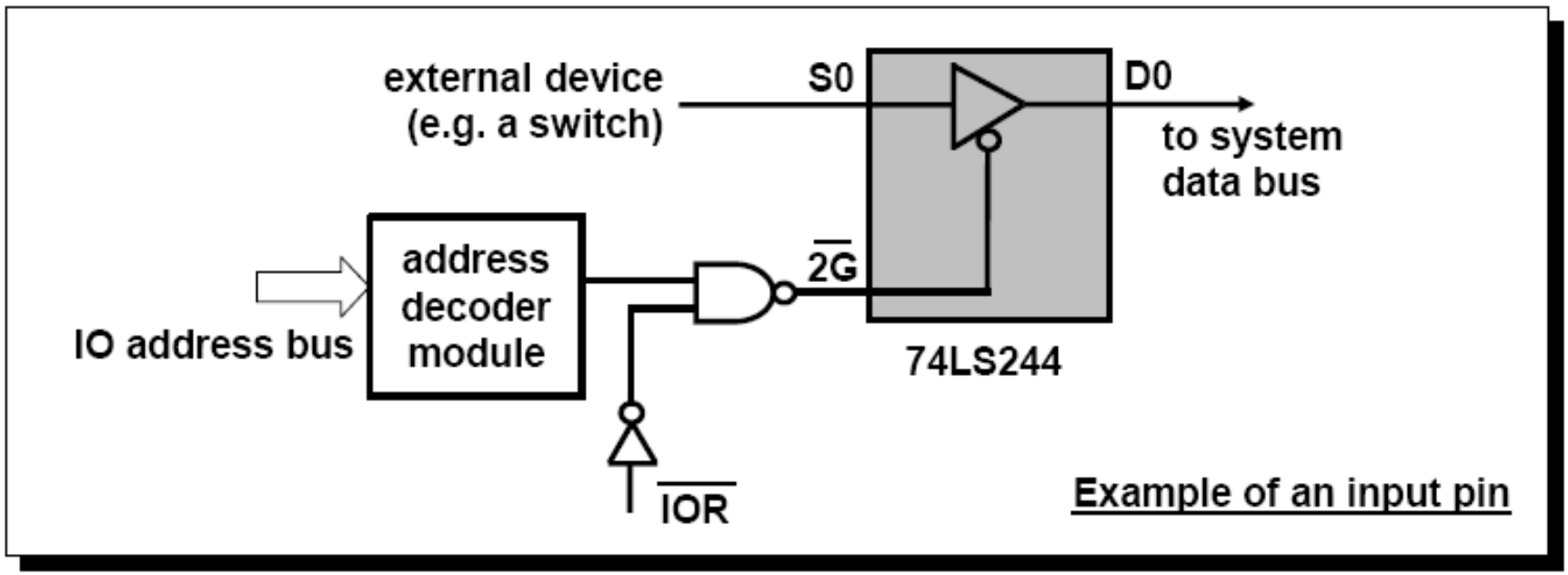
(a) IO-mapped input/output (b) Memory-mapped input/output

Các kiểu cổng IO

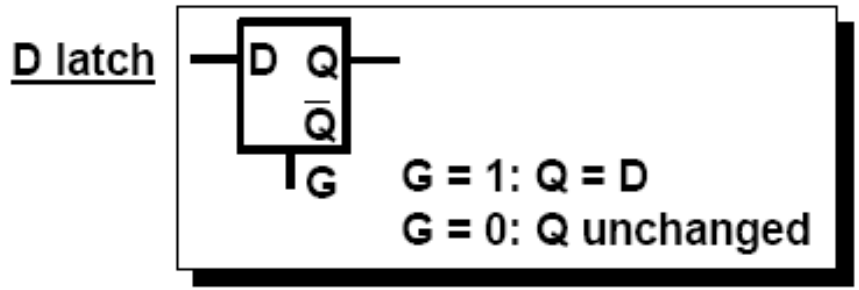
- Một cổng IO có thể có nhiều chân IO; mỗi chân tương ứng với 1 bit dữ liệu.
- Có 3 kiểu chân IO:
  - Các chân chỉ nhập
  - Các chân chỉ xuất
  - Các chân 2 chiều
- Một cổng IO có thể chứa nhiều chân có các kiểu hỗn hợp.

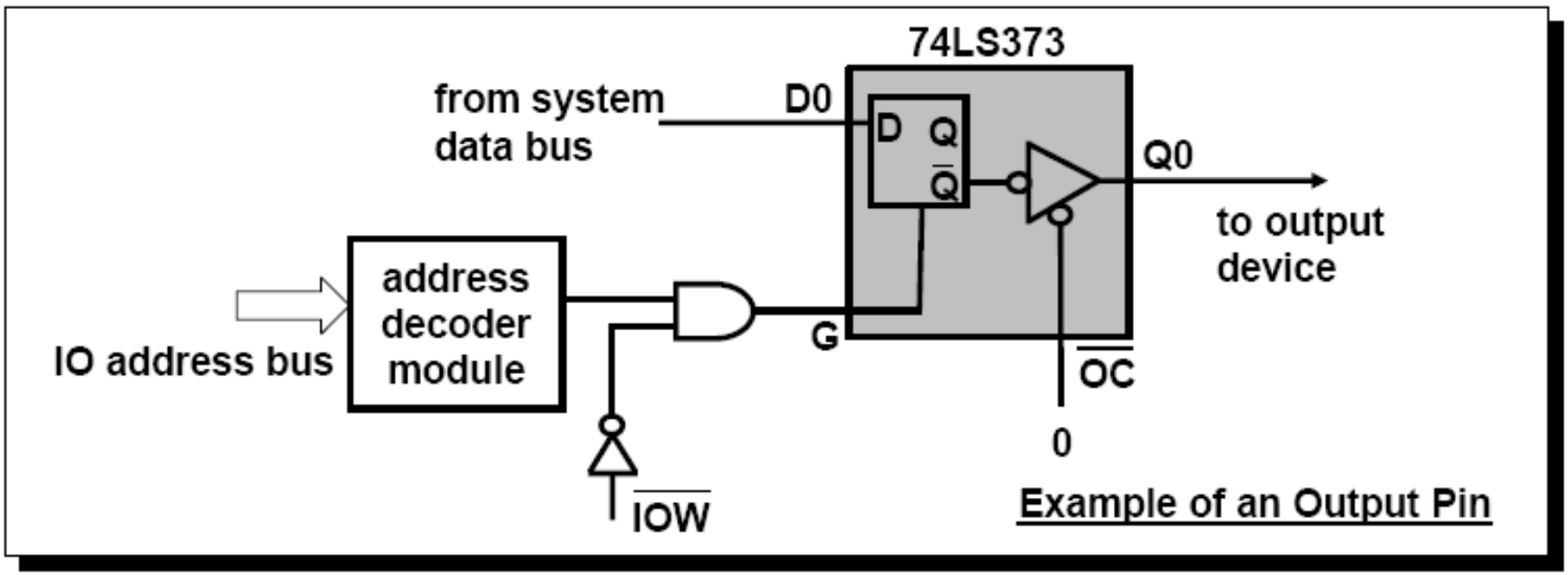
Các chân nhập (dùng đệm 3 trạng thái)



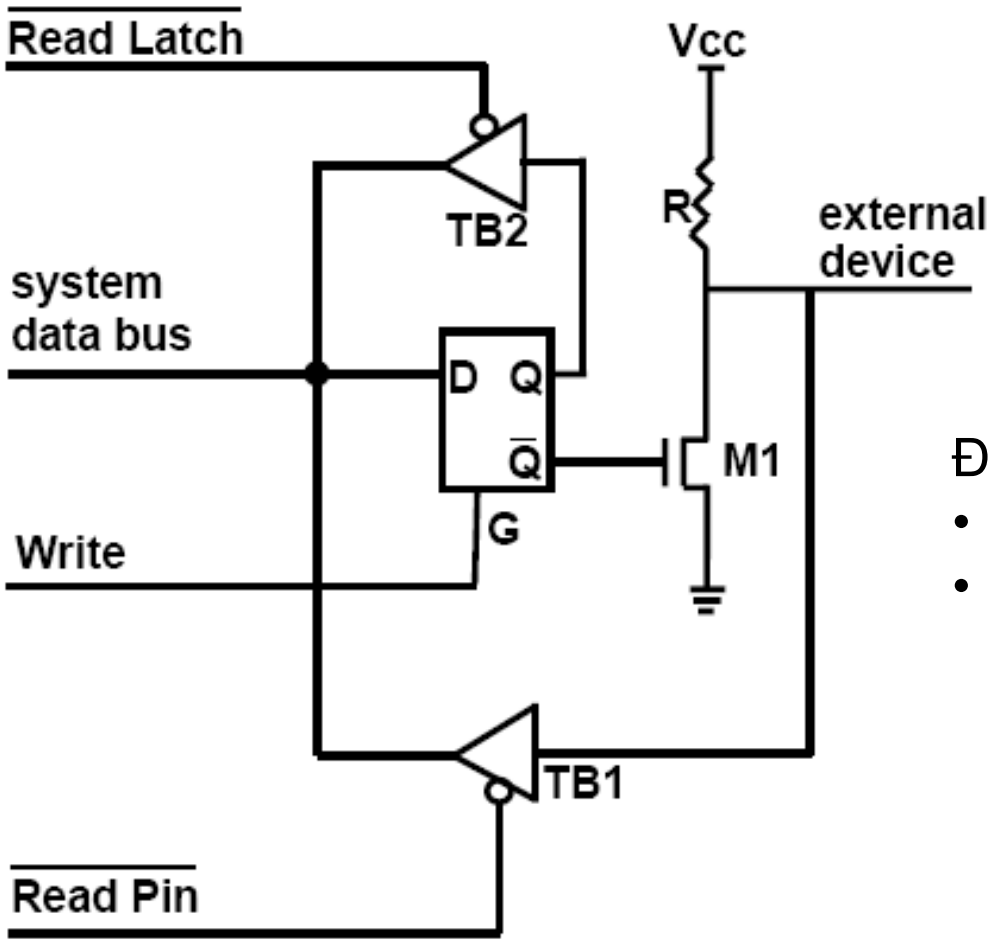


Các chân xuất (dùng mạch chốt)





Các chân 2 chiều



Example of a quasi-bidirectional pin

Đặt cấu hình:

- Nhập: ghi D=1 để M1 OFF
- Xuất: ghi D=0 để M1 ON

Bouncing Contacts

- Push-button switches, toggle switches, and electromechanical relays all have one thing in common: contacts.
- Metal contacts make and break the circuit and carry the current in switches and relays. Because they are metal, contacts have mass.
- Since at least one of the contacts is movable, it has springiness.
- Since contacts are designed to open and close quickly, there is little resistance (damping) to their movement

Bouncing

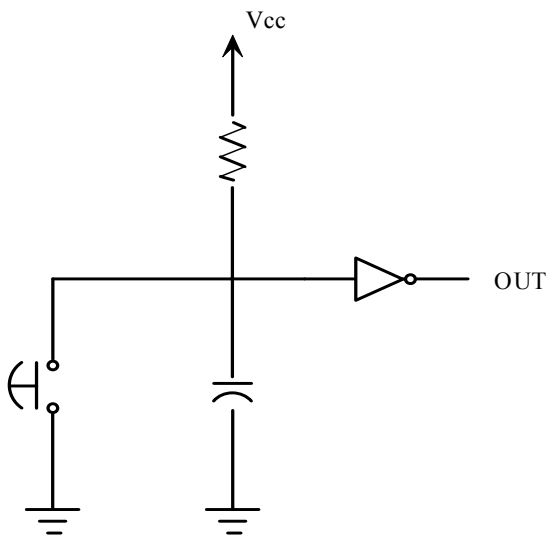
- Because the moving contacts have mass and springiness with low damping they will be "bouncy" as they make and break.
- That is, when a normally open (N.O.) pair of contacts is closed, the contacts will come together and bounce off each other several times before finally coming to rest in a closed position.
- The effect is called "contact bounce" or, in a switch, "switch bounce”.

Why is it a problem?

- If such a switch is used as a source to an edge-triggered input such as INT0, then the MCS-51 will think that there were several “events” and respond several times.
- The bouncing of the switch may last for several milliseconds.
  - Given that the MCS-51 operates at microsecond speed, a short ISR may execute several times in response to the above described bounciness

Hardware Solution

- The simplest hardware solution uses an **RC time constant** to suppress the bounce. The time constant has to be larger than the switch bounce and is typically **0.1 seconds**.
- As long as capacitor voltage does not exceed a threshold value, the output signal will be continued to be recognized as a logic 1.
- The buffer after the switch produces a sharp high-to-low transition.



Hardware Solution

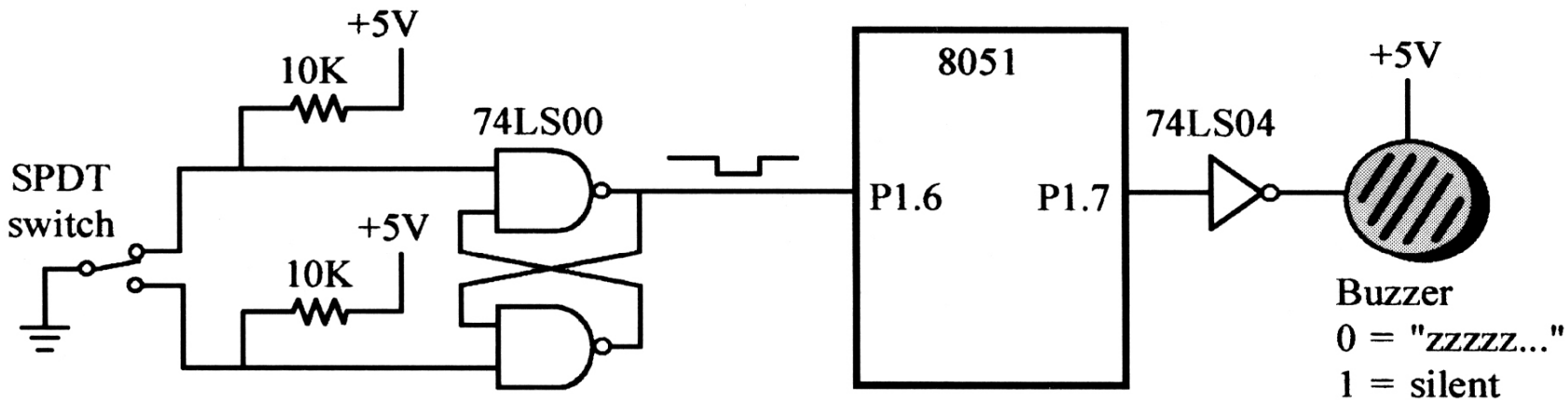


FIGURE 4–7
Buzzer example

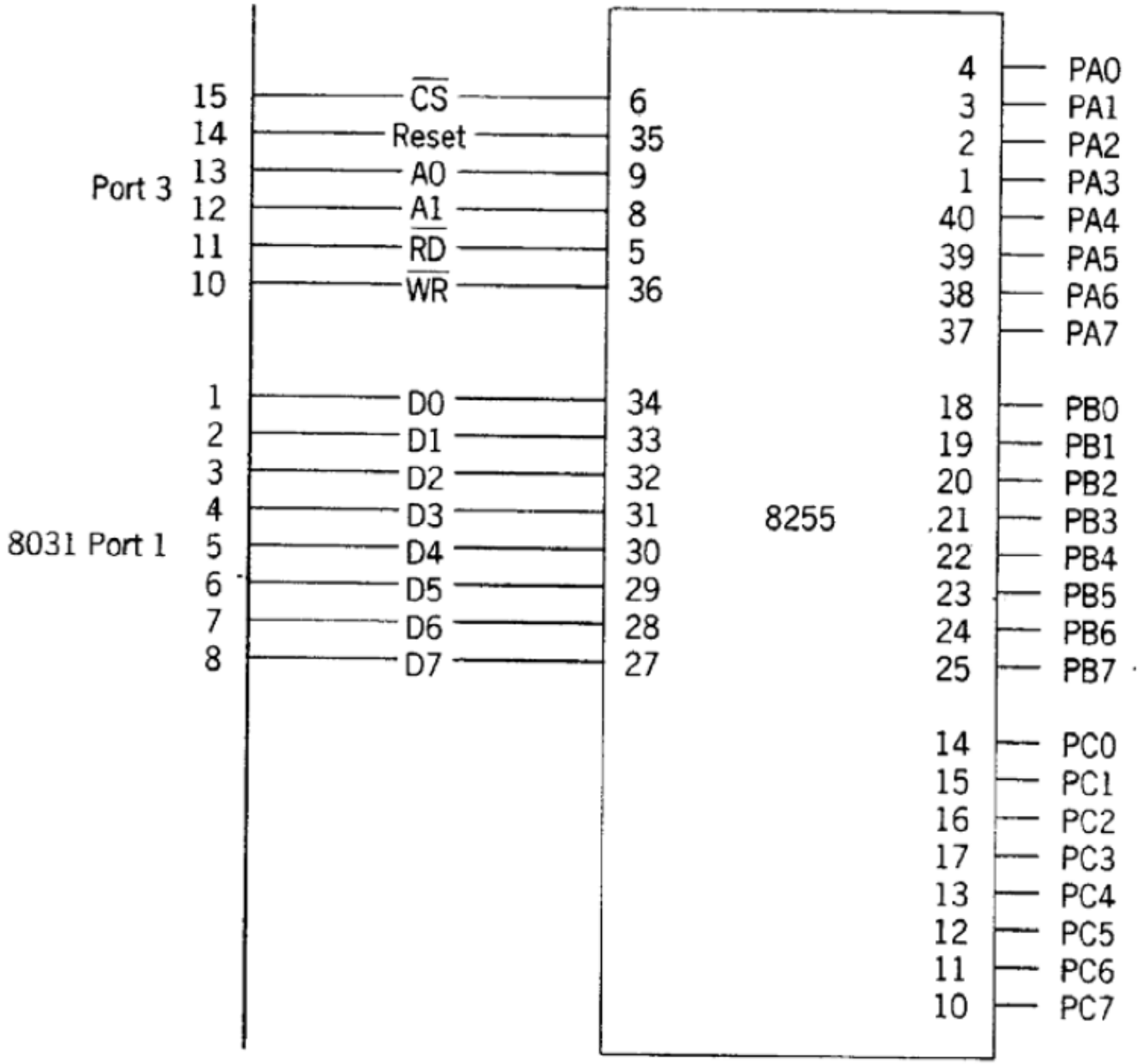
Software Solution

- It is also possible to counter the bouncing problem using software.
- The easies way is the **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

Mở rộng xuất nhập song song với 8255 PPI

Các IC hỗ trợ I/O cho các vi xử lý 8086/8088 và các MPU/MCU tương thích.

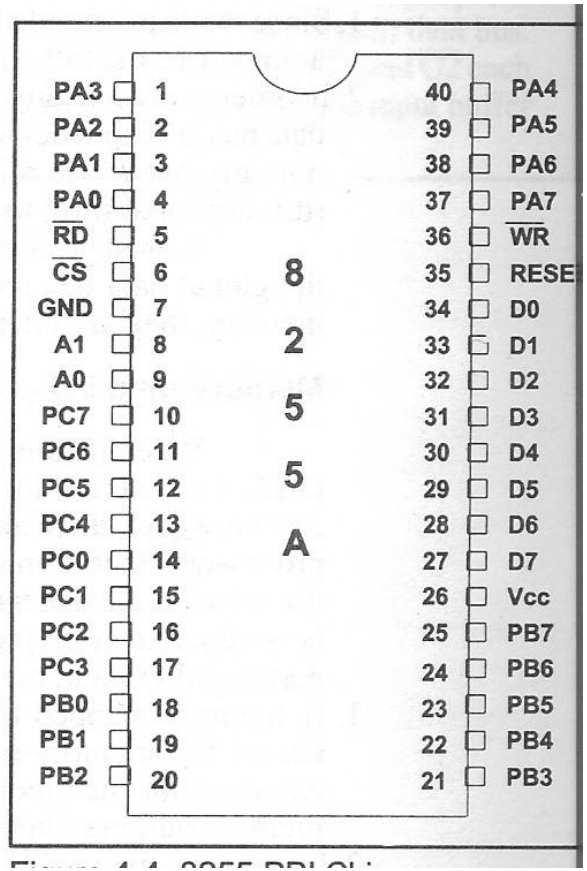
Số hiệu IC	**Mô tả**
8155/56	RAM với I/O và timer
8185	RAM 1KB
8755/8255	EPROM/ROM với I/O
8231	Đơn vị xử lý số học
8237	Bộ điều khiển DMA lập trình được
8251	Giao tiếp nối tiếp lập trình được
8254	Mạch định thì khoảng thời gian lập trình được
8255	Giao tiếp ngoại vi lập trình được
8256	UART đa chức năng
8259A	Bộ điều khiển ngắt lập trình được
8272	Bộ điều khiển đĩa mềm mật độ đơn/kép
8275	Bộ điều khiển CRT lập trình được
8279	Giao tiếp hiển thị bàn phím lập trình được
8295	Bộ điều khiển máy in ma trận điểm
82720	Bộ điều khiển hiển thị đồ họa



Hình 5.38 Mở rộng I/O song song của 8031/8051 bằng 8255.

8255

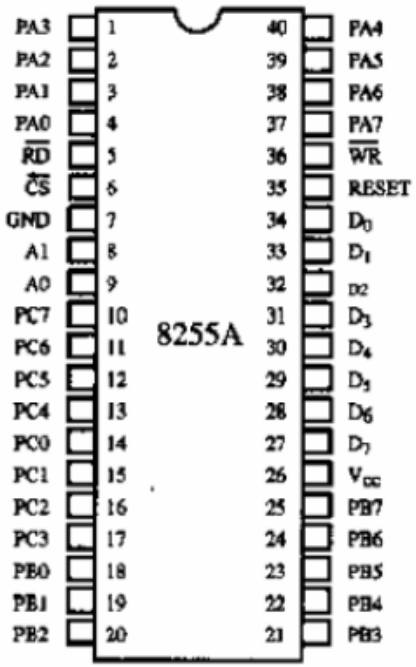
- 8051 has limited number of I/O ports
- one solution is to add parallel interface chip(s)
- 8255 is a Programmable Peripheral Interface PPI
- Add it to 8051 to expand number of parallel ports
- 8051 I/O port does not have handshaking capability
- 8255 can add handshaking capability to 8051



8255

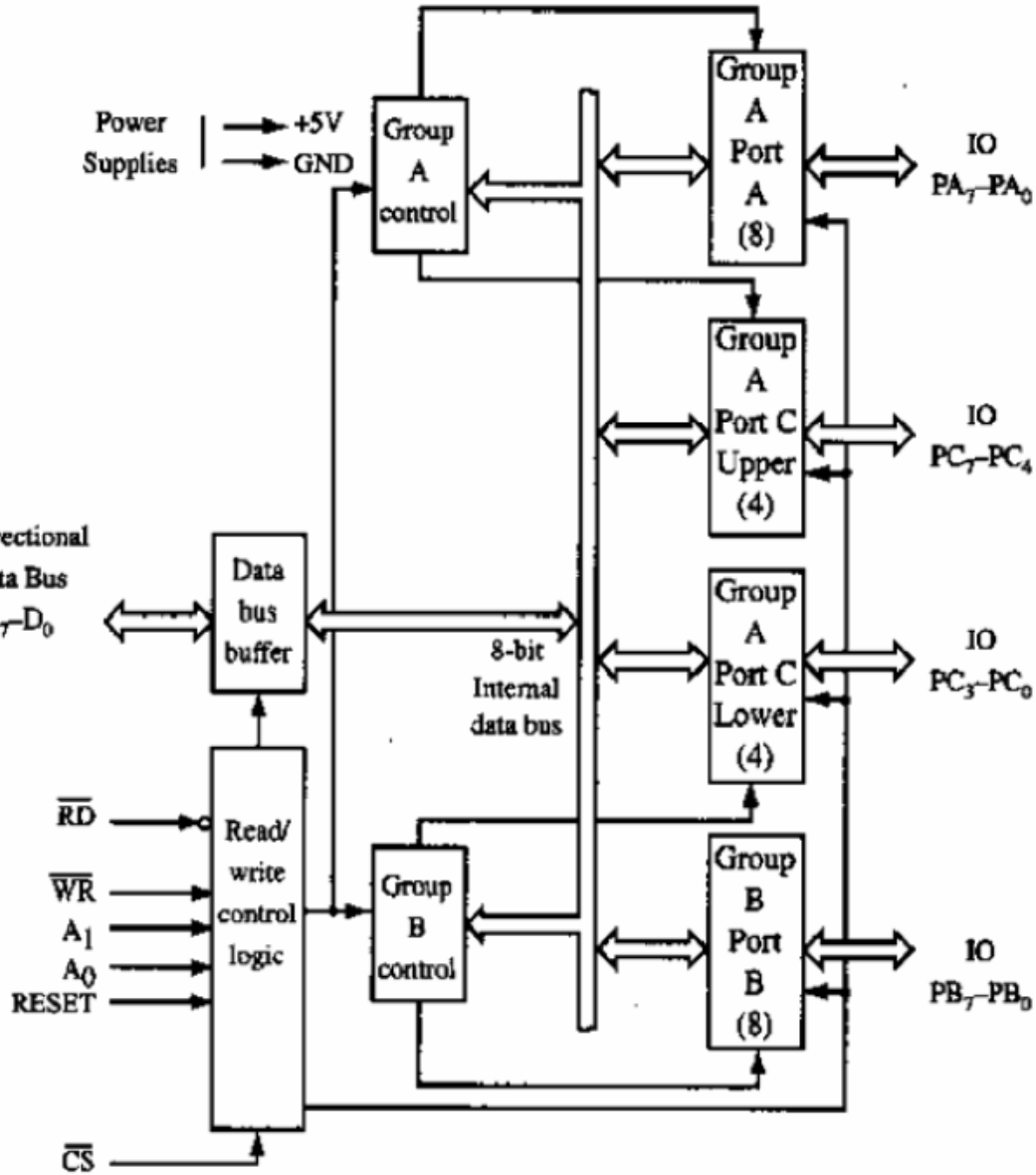
- Programmable Peripheral Interface (PPI)
  - Has 3 8_bit ports A, B and C
  - Port C can be used as two 4 bit ports CL and Ch
  - Two address lines A0, A1 and a Chip select CS
  - 8255 can be configured by writing a control-word in CR register

CS*	A1	A0	Selects:
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	x	x	8255 is not selected



Pin Names

D_7–D_0	Data Bus (Bidirectional)
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0, A1	Port Address
PA7–PA0	Port A (bit)
PB7–PB0	Port B (bit)
PC7–PC0	Port C (bit)
V_{cc}	+5V
GND	0V



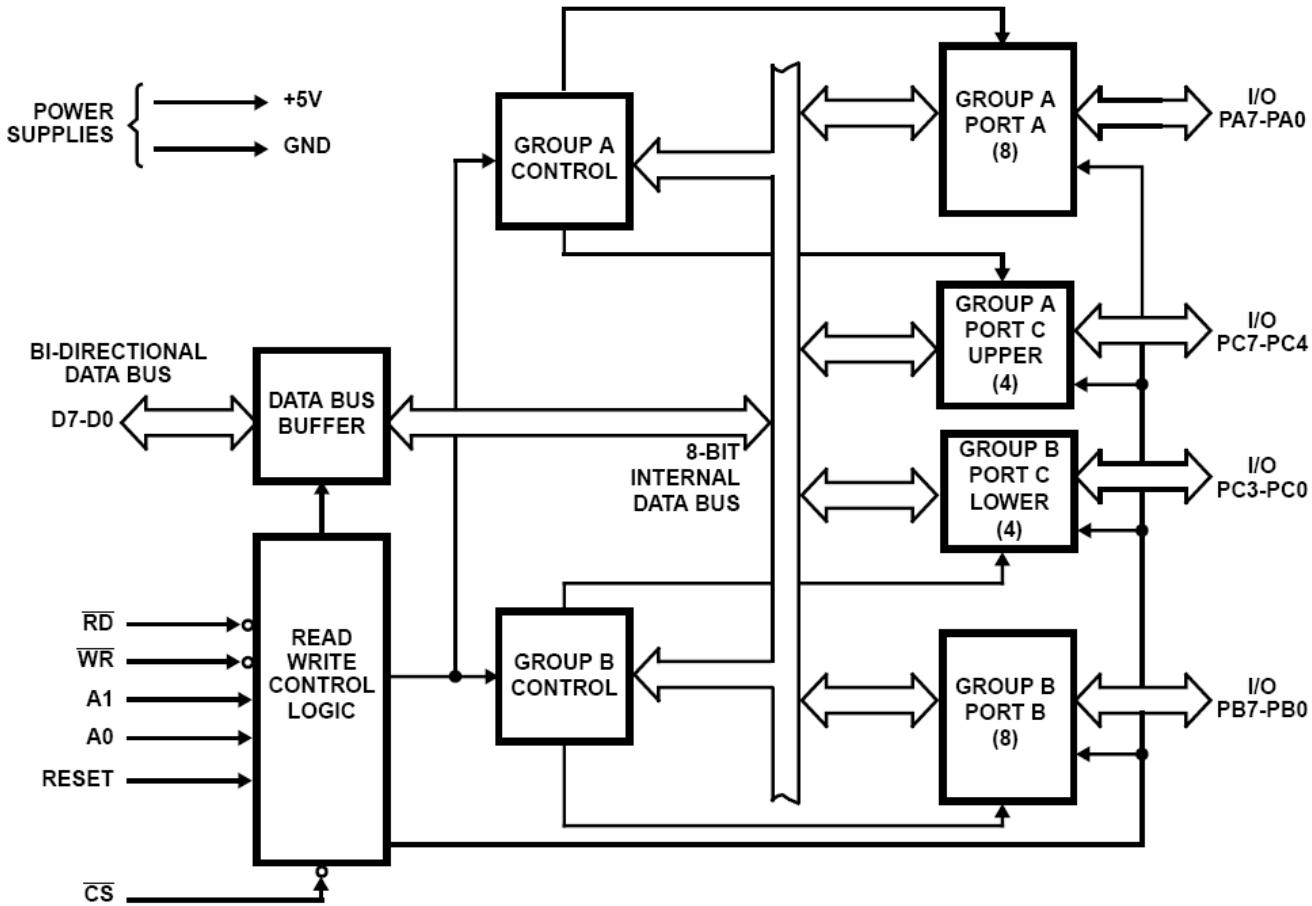
Hình 5.39 PPI 8255.

Bảng 5.3 Mô tả các chân của PPI 8255

Ký hiệu	Số chân	Kiểu	Mô tả
V_{CC}	26		V_{CC}: Chân cấp nguồn +5V. Ta nên gắn thêm 1 tụ 0.1µF giữa chân 26 và 7 để chống nhiễu do đột biến nguồn.
GND	7		GROUND
D0-D7	27-34	I/O	DATA BUS: Các đường bus dữ liệu là các chân 2 chiều 3 trạng thái được nối vào bu sdữ liệu hệ thống.
RESET	35	I	RESET: Mức cao ở ngõ vào này xóa thanh ghi điều khiển và tất cả các cổng (A, B, C) bị reset về chế độ nhập.
/CS	6	I	CHIP SELECT: Ngõ vào chọn chip này hoạt động tích cực thấp, được dùng để cho phép 8255 kết hợp với bus dữ liệu để truyền thông với CPU.
/RD	5	I	READ: Đây là tín hiệu điều khiển (ngõ vào tích cực thấp) mà CPU dùng để đọc thông tin trạng thái hoặc dữ liệu qua bus dữ liệu.
/WR	36	I	WRITE: Đây là tín hiệu điều khiển (ngõ vào tích cực thấp) mà CPU dùng để nạp (ghi) các "từ điều khiển" và dữ liệu vào 8255.
A0-A1	8, 9	I	ADDRESS: Đây là các ngõ vào địa chỉ, cùng với các ngõ vào /R và /WR, để điều khiển 1 trong 3 cổng hoặc thanh ghi "từ điều khiển". A0 và A1 thường được nối vào các bit thấp nhất của bus địa chỉ A0, A1.
PA0-PA7	1-4, 37-40	I/O	PORTA: cổng nhập và xuất 8 bit có giữ (có chốt).
PB0-PB7	18-25	I/O	PORT B: cổng nhập và xuất 8 bit có giữ (có chốt).
PC0-PC7	10-17	I/O	PORT C: cổng nhập và xuất 8 bit có giữ (có chốt).

8255

Functional Diagram



8255 – Logic điều khiển đọc/ghi

Phần điều khiển có 6 đường:

/RD (Read): Tín hiệu điều khiển này cho phép "Đọc". Khi tín hiệu này ở mức thấp, CPU đọc dữ liệu từ cổng I/O được chọn của 8255A.

/WR(Xóa): Tín hiệu điều khiển này cho phép "Ghi". Khi tín hiệu này ở mức thấp, CPU ghi vào cổng I/O được chọn của 8255A.

RESET (Xóa): Đây là tín hiệu tích cực mức cao; nó xóa thanh ghi điều khiển và đặt tất cả các cổng trong chế độ nhập.

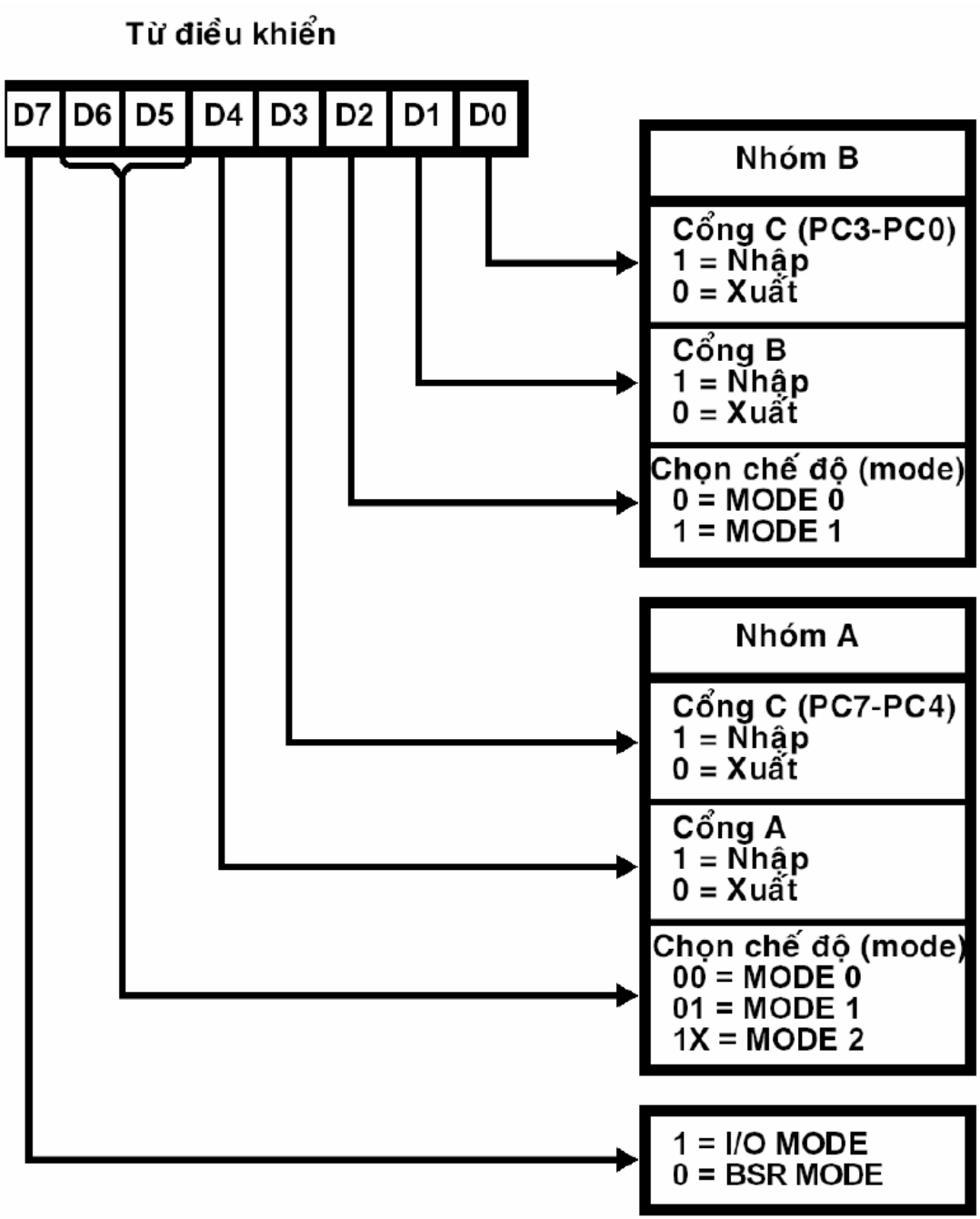
/CS, A0 và A1: Đây là các tín hiệu chọn thiết bị. /CS được nối vào địa chỉ được giải mãa, và A0 và A1 thường được nối vào các đường địa chỉ A0 và A1 tương ứng.

Tín hiệu /CS là tín hiệu chọn chip chính, và A0 và A1 xác định 1 trong cácc ổng I/O hay thanh ghi điều khiển như sau:

/CS	A1	A0	Chọn
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	Thanh ghi điều khiển
1	X	X	8255A không được chọn

Bảng 5.4 Hoạt động cơ bản của 8255.

A1	A0	RD	WR	CS	INPUT OPERATION (READ) Hoạt động nhập (Đọc)
0	0	0	1	0	Cổng A → Bus dữ liệu
0	1	0	1	0	Cổng B → Bus dữ liệu
1	0	0	1	0	Cổng C → Bus dữ liệu
1	1	0	1	0	Từ điều khiển → Bus dữ liệu
					OUTPUT OPERATION(WRITE) **Hoạt động xuất (Ghi)**
0	0	1	0	0	Bus dữ liệu → Cổng A
0	1	1	0	0	Bus dữ liệu → Cổng B
1	0	1	0	0	Bus dữ liệu → Cổng C
1	1	1	0	0	Bus dữ liệu → Điều khiển
					DISABLE FUNCTION **(Chức năng cấm)**
X	X	X	X	1	Bus dữ liệu → trạng thái thứ 3
X	X	1	1	0	Bus dữ liệu → trạng thái thứ 3



Hình 5.41 Dạng "từ điều khiển" của 8255 cho chế độ I/O.

8255

Các cổng A, B và C

8255 chứa 3 cổng 8 bit (A, B và C). Tất cả các cổng này có thể được đặt cấu hình bằng phần mềm, nhưng mỗi cổng có các đặc tính riêng của nó.

Cổng A: Bộ đệm/chốt xuất dữ liệu 8 bit và mạch chốt nhập dữ liệu 8 bit.

Cổng B: Bộ đệm/chốt nhập/xuất dữ liệu 8 bit và bộ đệm nhập dữ liệu 8 bit.

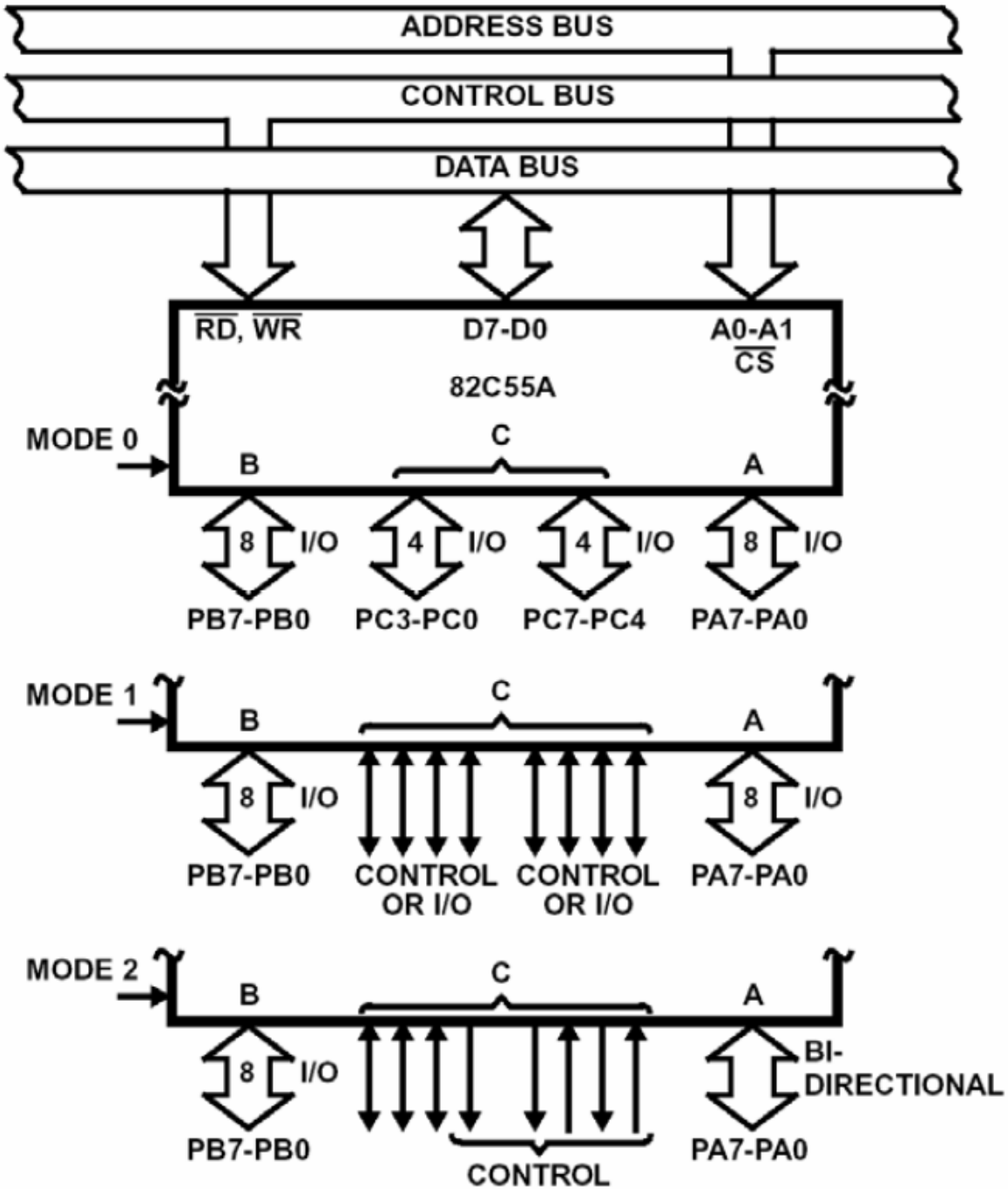
Cổng C: Bộ đệm/chốt xuất dữ liệu 8 bit và bộ đệm nhập dữ liệu 8 bit (không có chốt đối với nhập). Cổng này có thể được chia thành 2 cổng 4 bit theo chế độ làm việc. Mỗi cổng 4 bit chứa mạch chốt 4 bit và có thể dùng làm ngõ ra tín hiệu điều khiển và các ngõ vào tín hiệu trạng thái cùng với các cổng A và B.

Chọn chế độ làm việc

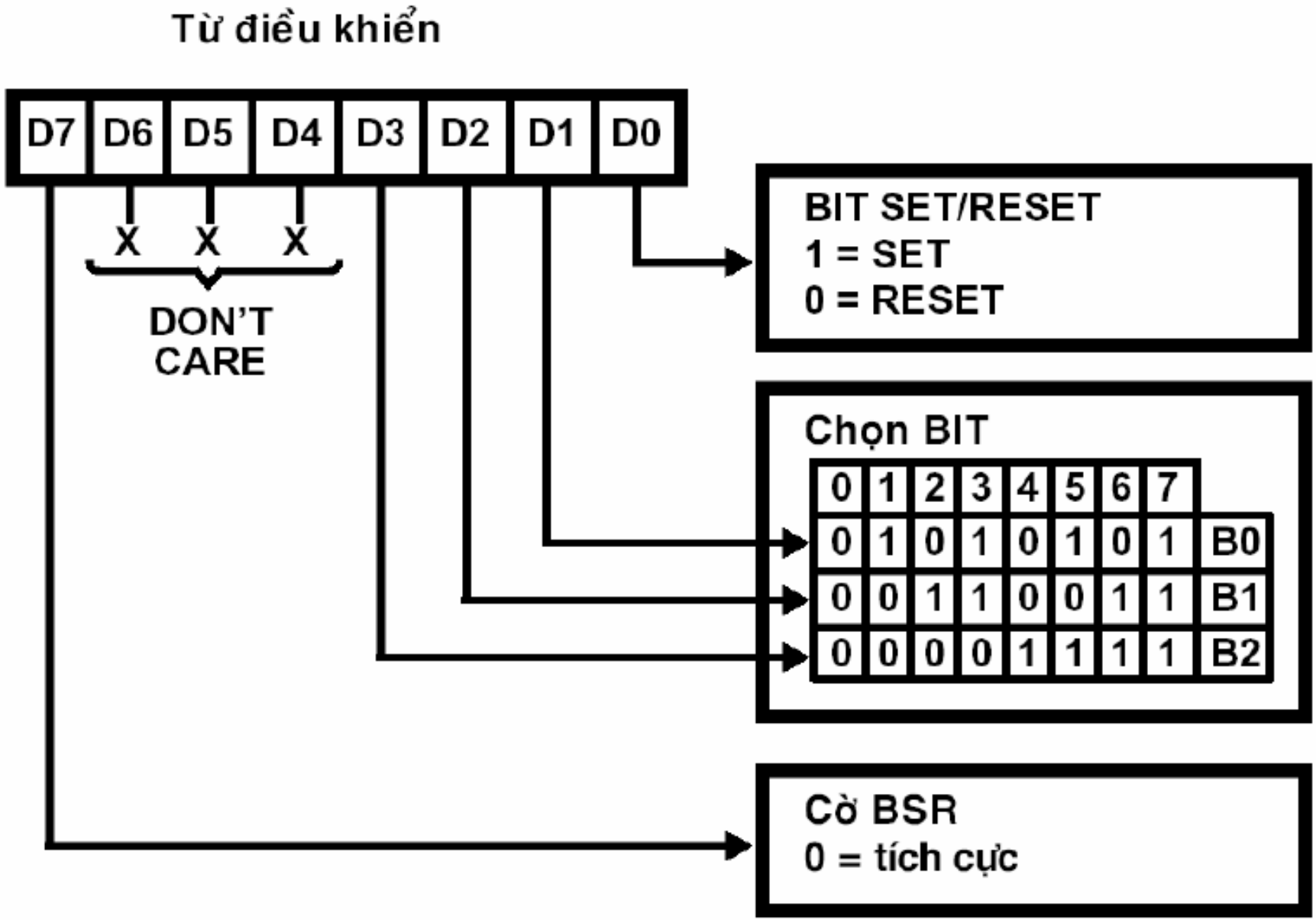
Có chế độ làm việc cơ bản mà có thể chọn được bằng phần mềm hệ thống:

- Mode 0: Nhập/Xuất cơ bản (Basic I/O)
- Mode 1: Nhập/Xuất có lấy mẫu (Strobed I/O)
- Mode 2: Bus 2 chiều.

Khi ngõ vào reset ở mức cao, tất cả cổng sẽ được đặt vào chế độ nhập với tất cả 14 đường cổng được giữ ở mức logic 1 bằng các thiết bị giữ bus bên trong. Sau khi hết reset, 8255 vẫn duy trì ở chế độ nhập, không cần sự khởi tạo nào cả.



Hình 5.42 Các định nghĩa chế độ cơ bản và giao tiếp bus.



Hình 5.43 Dạng BSR.

8255

Các chức năng điều khiển ngắt

Khi 8255 được lập trình để hoạt động ở chế độ 1 hoặc 2, các tín hiệu điều khiển được cung cấp để có thể được sử dụng như các ngõ vào yêu cầu ngắt với CPU. Các tín hiệu yêu cầu ngắt, được sinh ra từ cổng C, có thể bị cấm hay cho phép bằng cách đặt hay xóa fliflop INTE liên hệ, bằng cách dùng chức năng BSR của cổng C.

Chức năng này để cho người lập trình cho phép hoặc cấm ngắt CPU bằng thiết bị I/O cụ thể mà không ảnh hưởng bất cứ thiết bị nào khác trong cấu trúc ngắt.

Định nghĩa flipflop INTE:

(Bit Set) thì INTE=1–Cho phép ngắt.

(Bit Reset) thì INTE=0–Cấm ngắt.

Chú ý là tất cả các flipflop mặt nạ bị reset tự động trong khi chọn chế độ và Reset thiết bị.

Thủ tục chung để truyền thông các ngoại vi qua 8255, cần có 3 bước:

1. Xác định các địa chỉ của các cổng A, B, và C và của thanh ghi điều khiển theo logic chọn chip (/CS) và các đường địa chỉ A_0 và A_1.
2. Ghi từ điều khiển vào thanh ghi điều khiển.
3. Ghi các lệnh I/O để truyền thông với ngoại vi qua các cổng A, B, và C.

8255

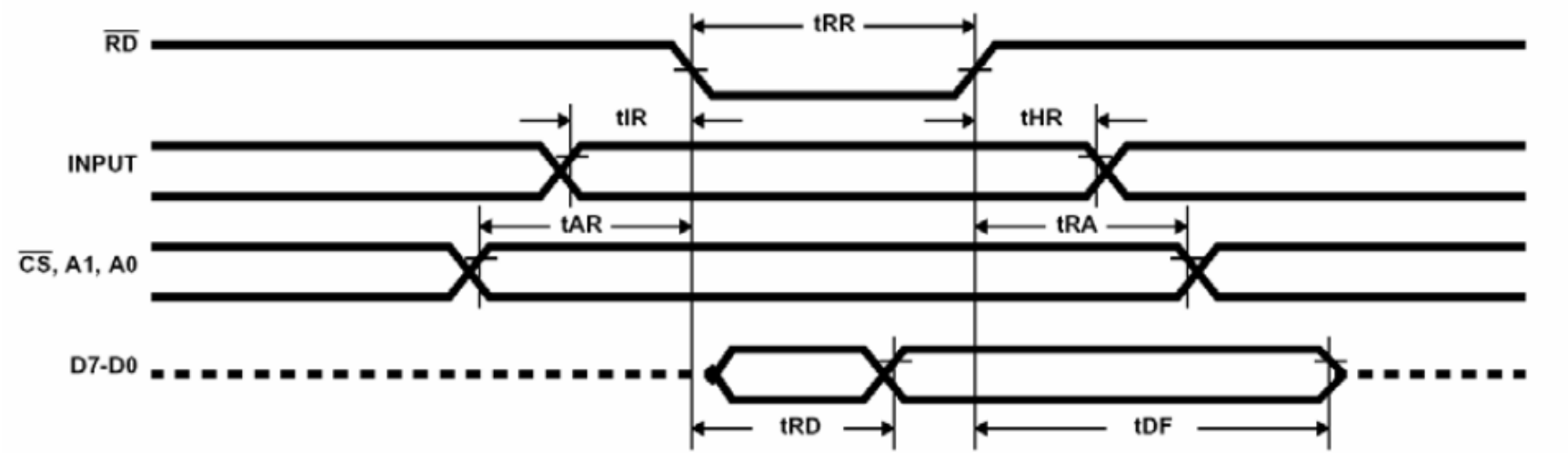
PPI 8255 với Mode 0 (Nhập hay Xuất đơn giản)

Trong chế độ này, mỗi cổng (hay nửa cổng trong trường hợp C) có thể lập trình để làm việc như cổng nhập hay xuất. Các đặc tính I/O trong "chế độ 0" như sau:

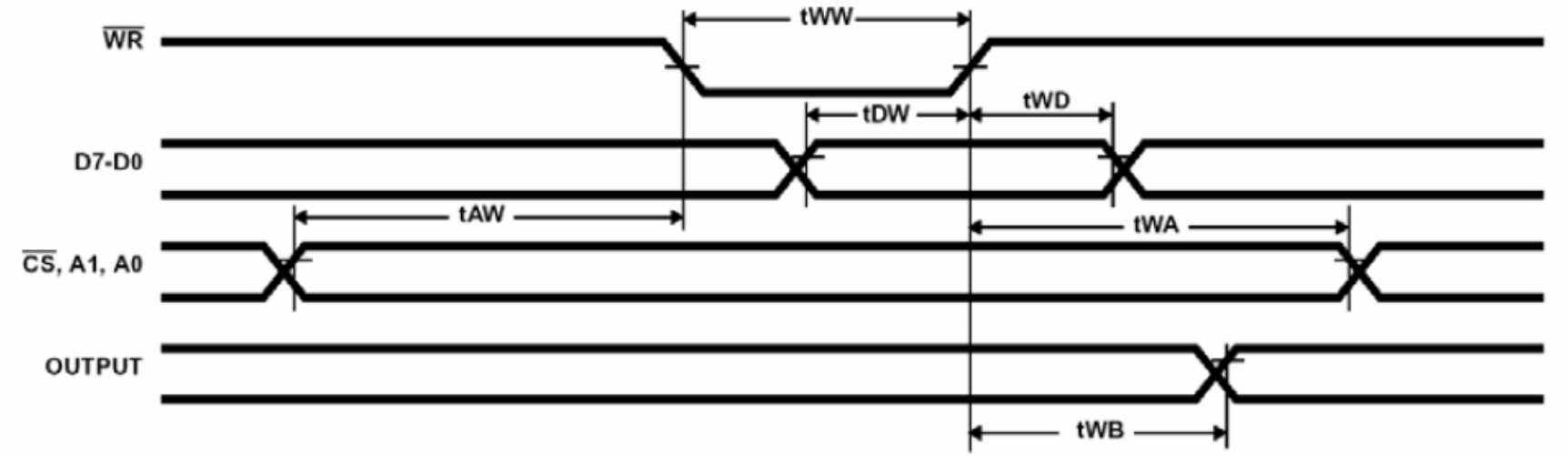
1. Hai cổng 8 bit (A và B) và hai cổng 4 bit (C_U và C_L)
2. Các ngõ ra được chốt.
3. Các ngõ vào không được chốt.
4. Các cổng không có khả năng bắt tay hay ngắt,
5. Có thể có 16 kiểu cấu hình I/O như trong bảng 5.5.

8255

Giản đồ định thì với chế độ 0



Hình 5.44 Mode 0 với nhập cơ bản



Hình 5.45 Mode 0 với xuất cơ bản

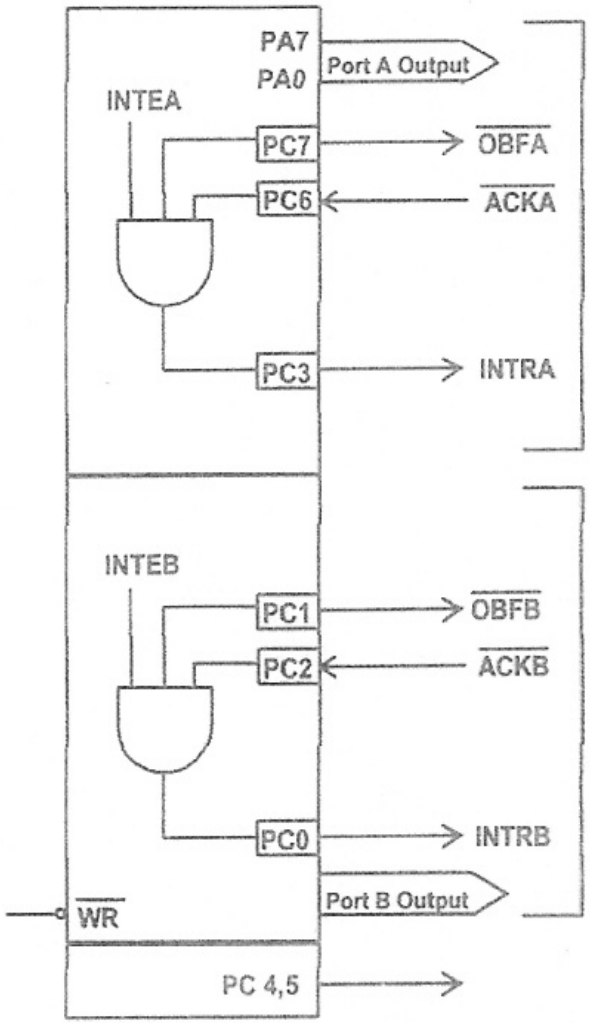
Mode 0

- Provides simple input and output operations for each of the three ports.
  - No “handshaking” is required, data is simply written to or read from a specified port.
  - Two 8-bit ports and two 4-bit ports.
  - Any port can be input or output.
  - Outputs are latched.
  - Inputs are not latched

Mode 1

- Mode 1 Basic functional Definitions:
  - Two Groups (Group A and Group B).
  - Each group has one 8-bit data port and one 4-bit control/data port.
  - The 8-bit data port can be either input or output. Both inputs and outputs are latched.
  - The 4-bit port is used for control and status of the 8-bit data port.

8255 mode 1 (output)



Port A with

Handshake Signals

Control Word -- Mode 1 Output

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	1/0	1	0	x

I/O Mode

Port A Mode 1

Port A Output

Port B Output

Port B Mode 1

PC 4,5
1 = Input
0 = Output

Port B with

Handshake Signals

Status Word -- Mode 1 Output

D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBFA}$	INTEA	I/O	I/O	INTRA	INTEB	$\overline{OBFB}$	INTRB

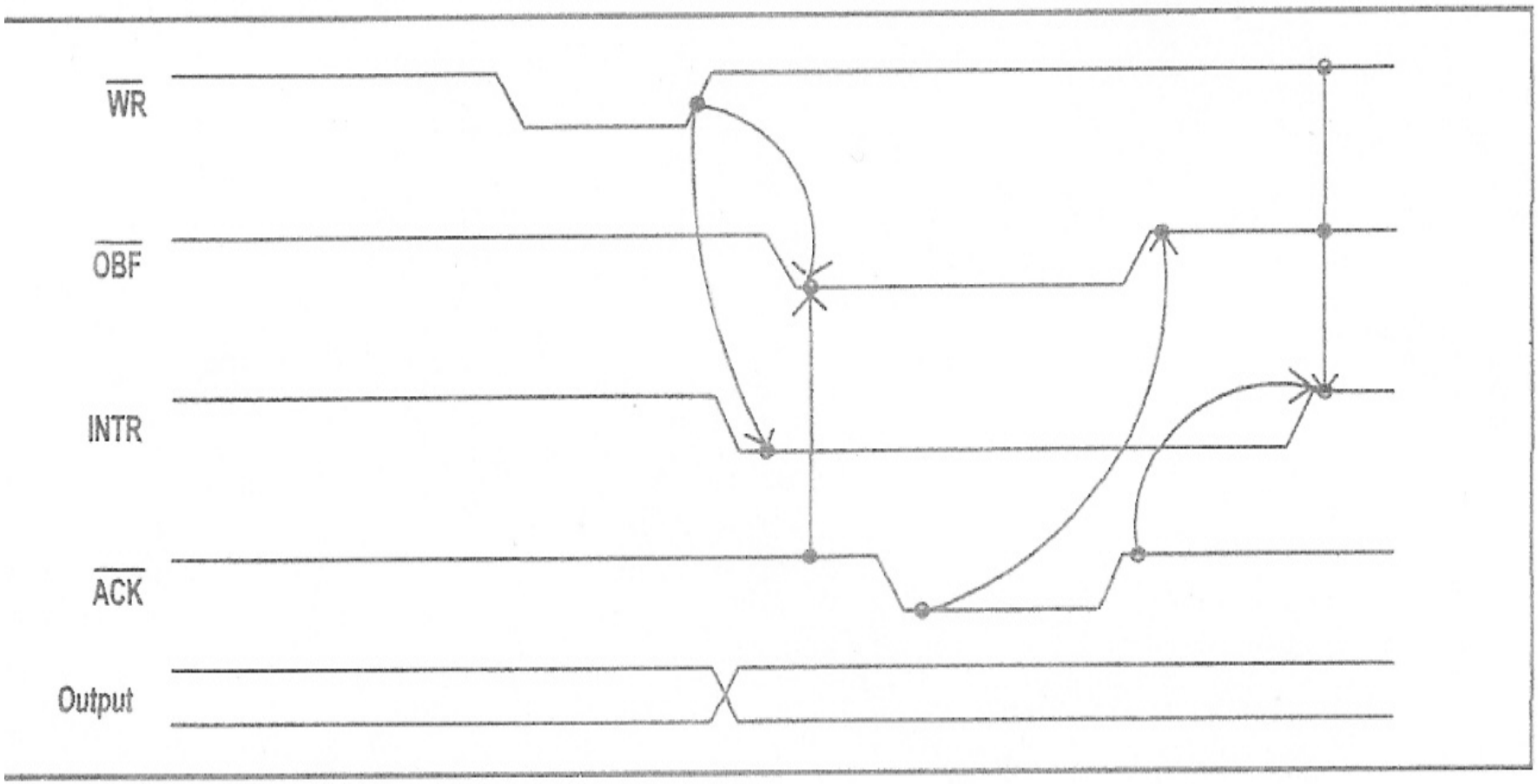
INTEA is controlled by PC6 in BSR mode.

INTEB is controlled by PC2 in BSR mode.

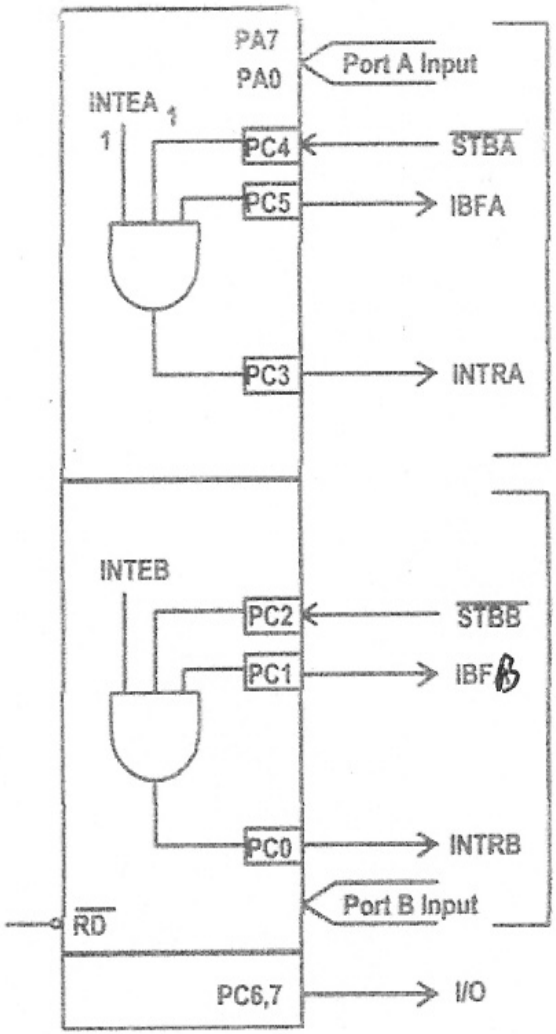
Mode 1 – Control Signals

- Output Control Signal Definition
  - OBF (Output Buffer Full F/F). (C7 for A, C1 for B)
    - The OBF output will go “low” to indicate that the CPU has written data out to the specified port.
      - A signal to the device that there is data to be read.
  - ACK (Acknowledge Input). (C6 for A, C2 for B)
    - A “low” on this input informs the 8255 that the data from Port A or Port B has been accepted.
      - A response from the peripheral device indicating that it has read the data.
  - INTR (Interrupt Request). (C3 for A, C0 for B)
    - A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU.

Timing diagram for mode1(output)



8255 mode 1 (input)



Port A with

Handshake Signals

Control Word -- Mode 1 Input

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1/0	1	1	x

I/O Mode

Port A Mode 1

Port A Input

Port B Input

Port B Mode 1

PC 6,7

1 = Input
0 = Output

Port B with

Handshake Signals

Status Word -- Mode 1 Input

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB

INTEA is controlled by PC4 in BSR mode.

INTEB is controlled by PC2 in BSR mode.

Mode 1 – Control Signals

- Input Control Signal Definition
  - STB (Strobe Input). (C4 for A, C2 for B)
    - A “low” on this input loads data into the input latch.
  - IBF (Input Buffer Full F/F) (C5 for A, C1 for B)
    - A “high” on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement from the 8255 to the device.
  - INTR (Interrupt Request) (C3 for A, C0 for B)
    - A “high” on this output can be used to interrupt the CPU when an input device is requesting service.

Timing diagram for mode1(input)

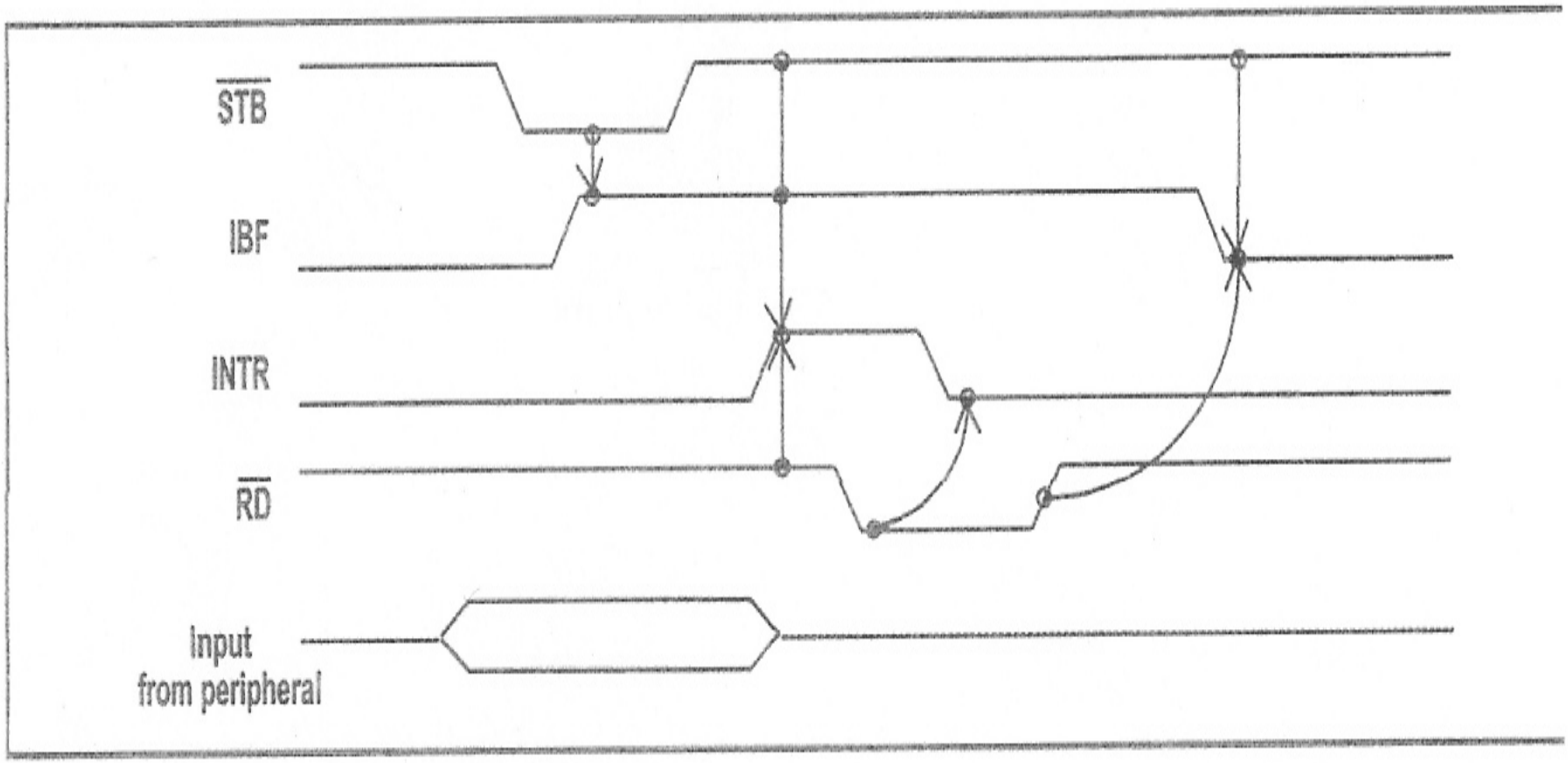


Figure 4-13. Timing Diagram for Mode 1 Input

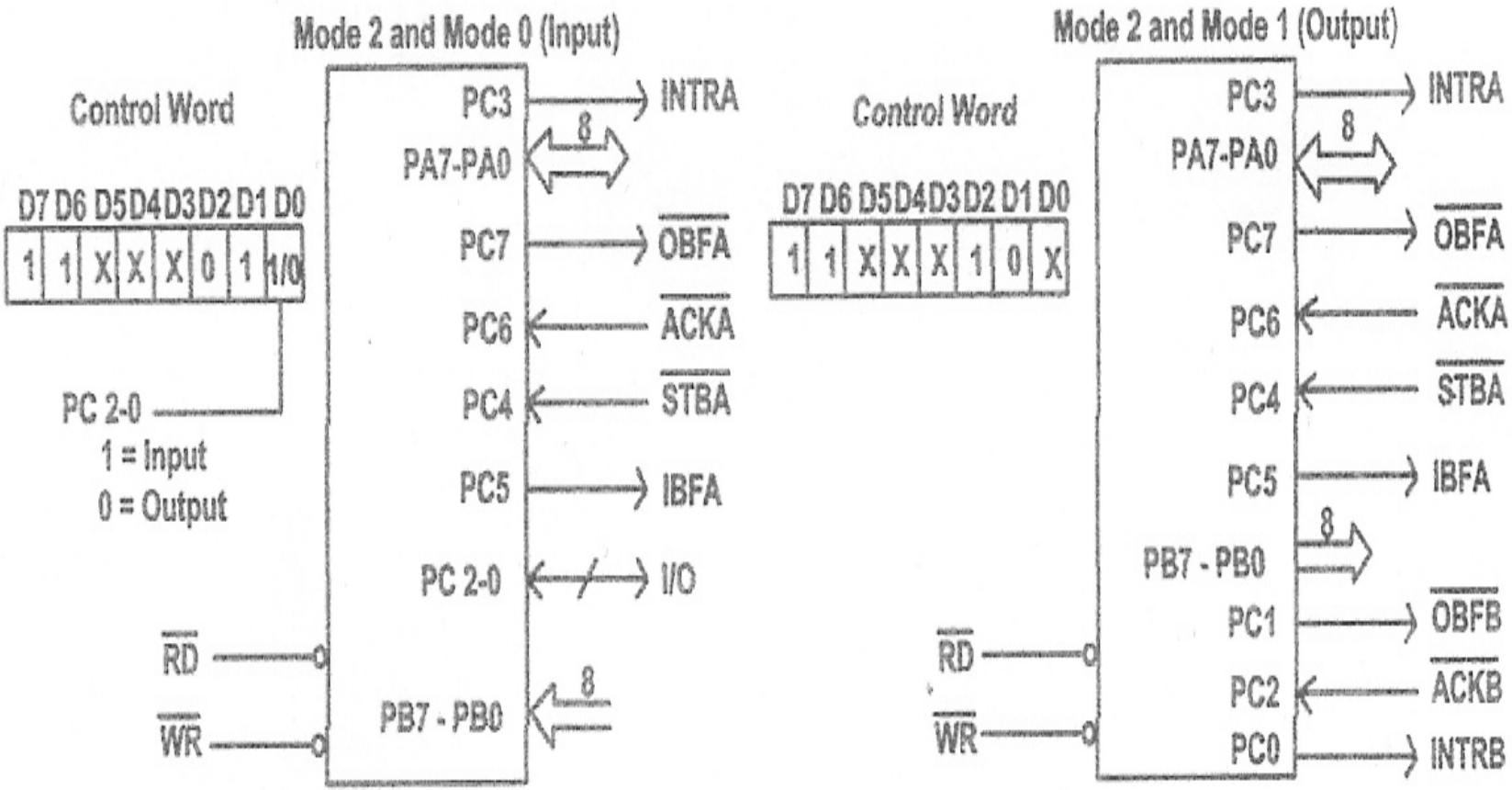
Mode 2 - Strobed Bidirectional Bus I/O

- MODE 2 Basic Functional Definitions:
  - Used in Group A only.
  - One 8-bit, bi-directional bus port (Port A) and a 5-bit control port (Port C).
  - Both inputs and outputs are latched.
  - The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Mode 2

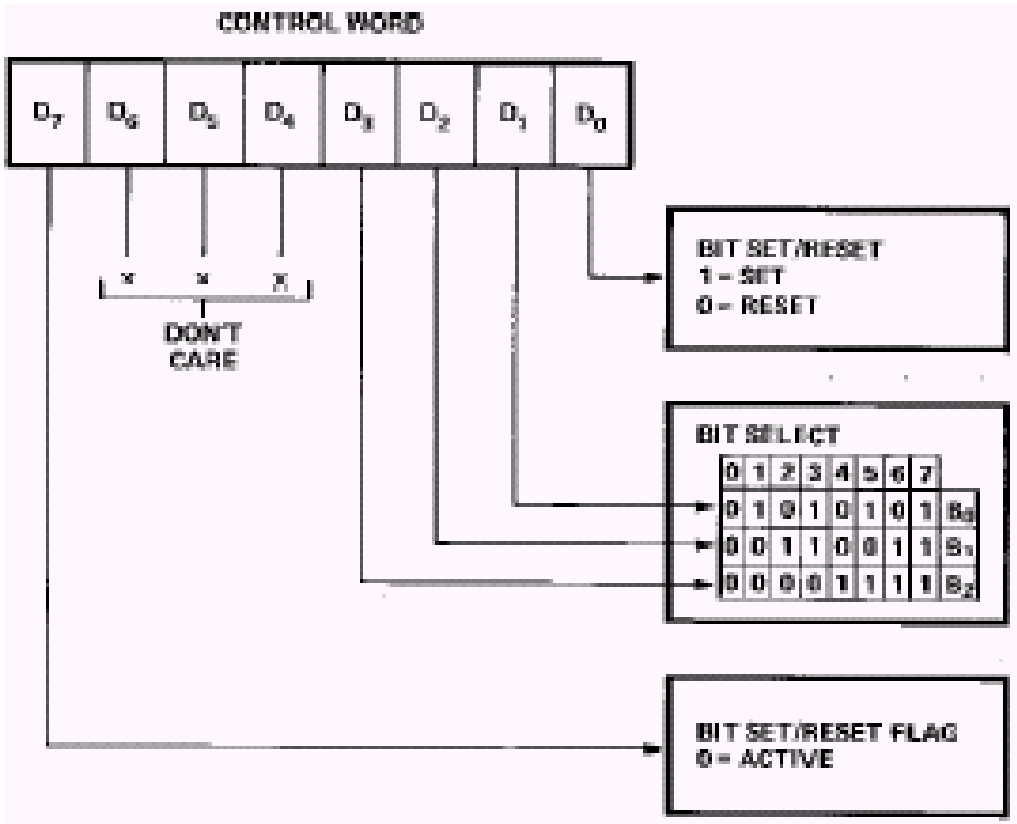
- **Output Operations**
  - OBF (Output Buffer Full). The OBF output will go low to indicate that the CPU has written data out to port A.
  - ACK (Acknowledge). A low on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high impedance state.
- **Input Operations**
  - STB (Strobe Input). A low on this input loads data into the input latch.
  - IBF (Input Buffer Full F/F). A high on this output indicates that data has been loaded into the input latch.

Pin	Function
PC7	/OBF
PC6	/ACK
PC5	IBF
PC4	/STB
PC3	INTR
PC2	I/O
PC1	I/O
PC0	I/O


Mode 2 and Mode 0 (Input)
Control Word
D7 D6 D5 D4 D3 D2 D1 D0
1 1 X X X 0 1 1/0
PC 2-0
1 = Input
0 = Output
PC3
INTRA
8
PA7-PA0
PC7
OBFA
PC6
ACKA
PC4
STBA
PC5
IBFA
PC 2-0
I/O
RD
WR
PB7 - PB0
8
Mode 2 and Mode 1 (Output)
Control Word
D7 D6 D5 D4 D3 D2 D1 D0
1 1 X X X 1 0 X
PC3
INTRA
8
PA7-PA0
PC7
OBFA
PC6
ACKA
PC4
STBA
PC5
IBFA
8
PB7 - PB0
PC1
OBFB
PC2
ACKB
PC0
INTRB
RD
WR

BSR Mode

- If used in BSR mode, then the bits of port C can be set or reset individually

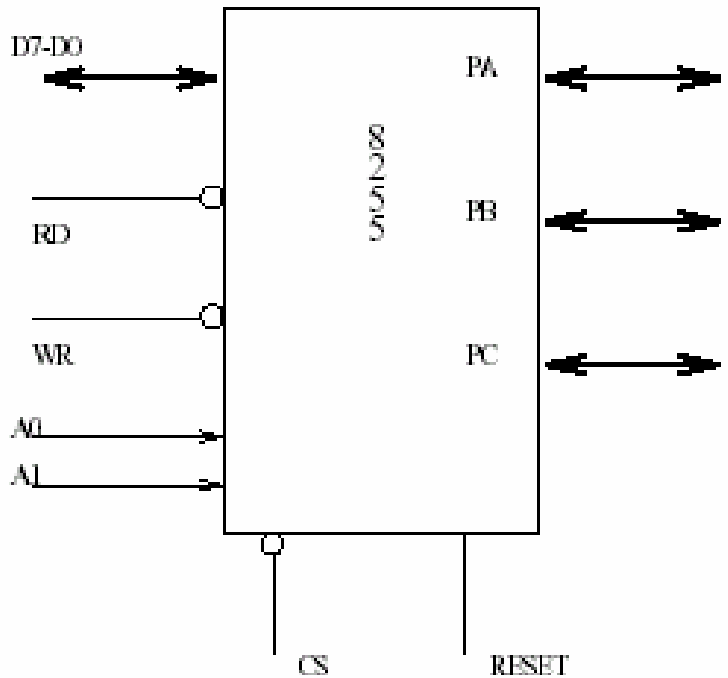


BSR Mode example

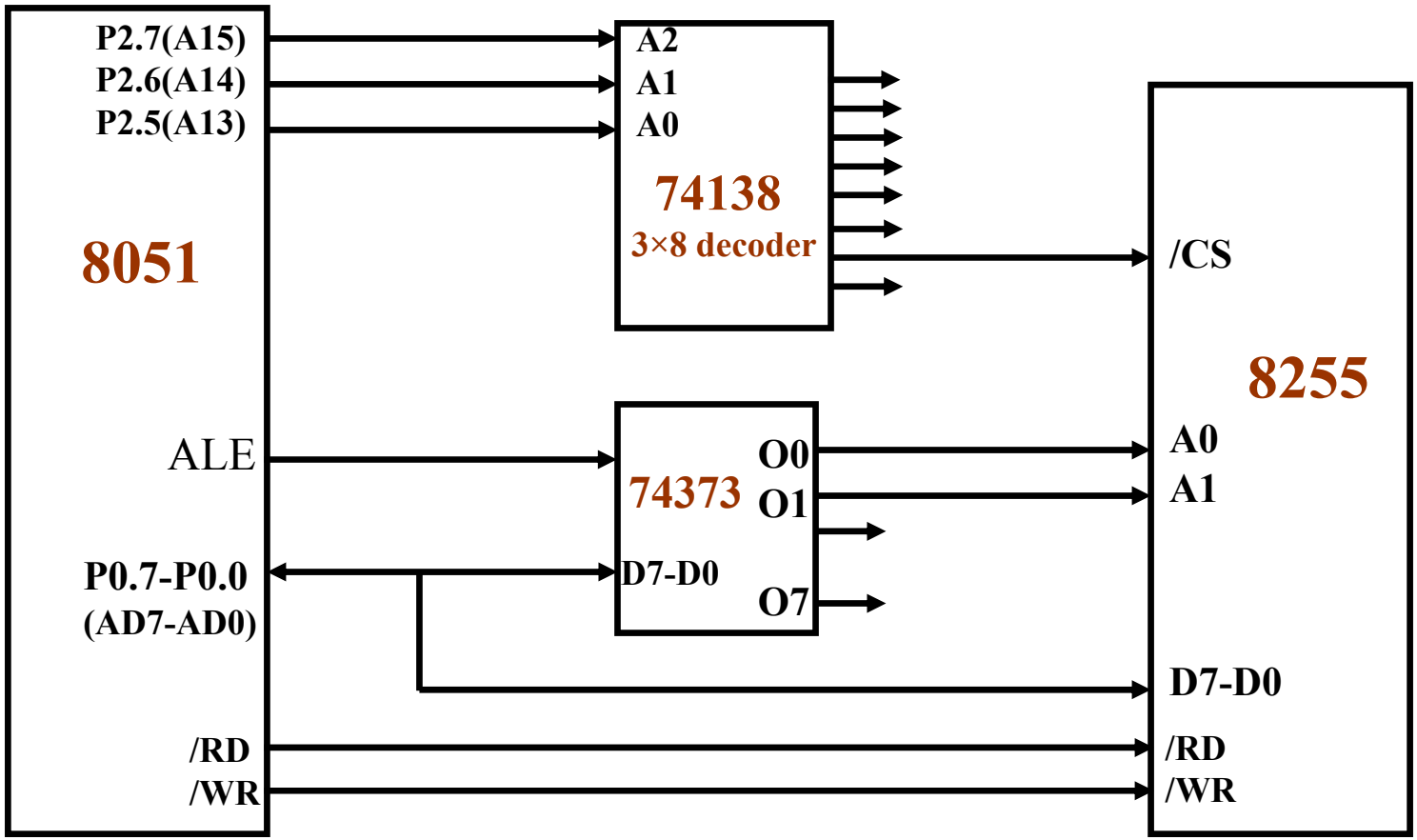
```
    Move dptr, 0093h
Up: Move a, 09h  ;set pc4
    Movx @dptr,a
    Acall delay
    Mov  a,08h ;clr pc4
    Movx @dptr,a
    Acall delay
    Sjmp up
```

Interfacing 8255 with 8051

- CS is used to interface 8255 with 8051
- If CS is generated from lets say Address lines A15:A12 as follows,
  A15:A13 = 110
- Address of 8255 is 110 xxxxx xxxx xx00b
- Base address of 8255 is
  - 1100 0000 0000 0000b=C000H
- Address of the registers
  - A = C000H
  - B = C001H
  - C = C002H
  - CR = C003H



Interfacing 8255 with 8051



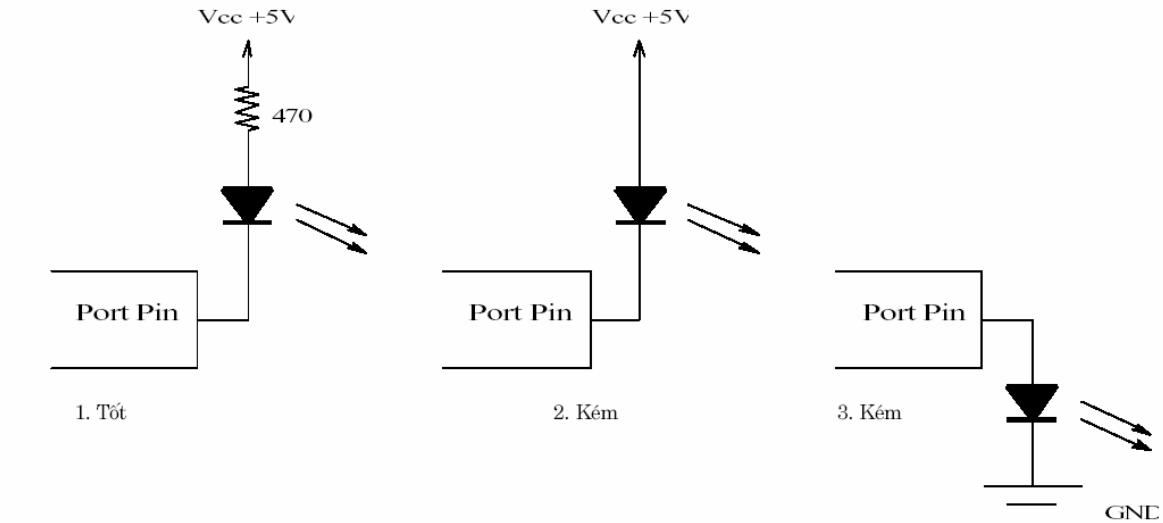
8255 Usage: Simple Example

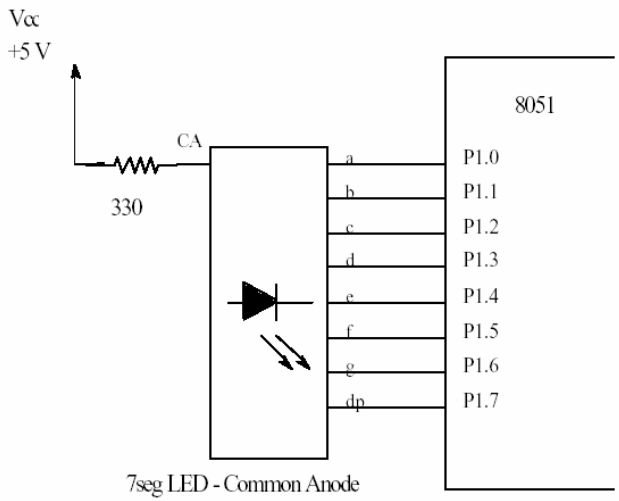
- 8255 memory mapped to 8051 at address C000H base
  - A = C000H, B = C001H, C = C002H, CR = C003H
- Control word for all ports as outputs in mode0
  - CR : 1000 0000b = 80H

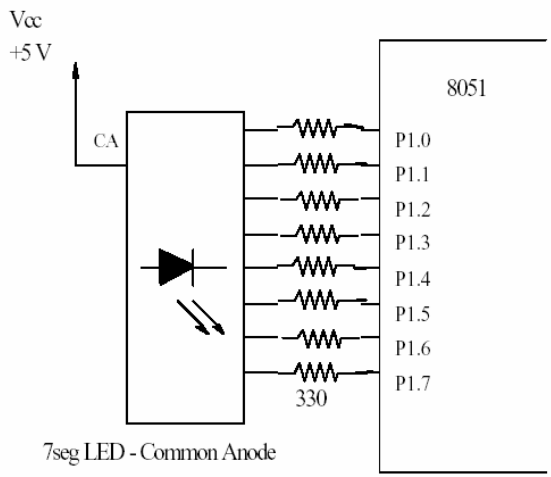
```
test:       mov   A, #80H       ; control word
      mov   DPTR, #C003H        ; address of CR
      movx  @DPTR, A      ; write control word
      mov   A, #55h       ; will try to write 55 and AA
                           ;                  alternatively
repeat:mov   DPTR,#C000H        ; address 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)
```

5.6 Giao tiếp bộ hiển thị (Display)
5.6.1 Giao tiếp với LED

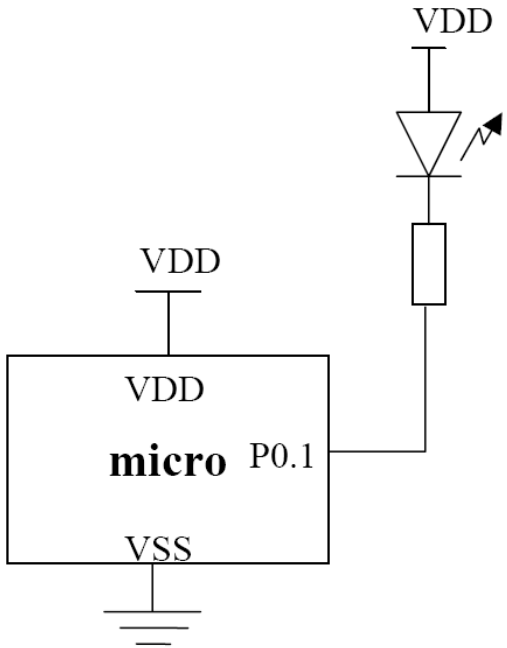
Giao tiếp với LED



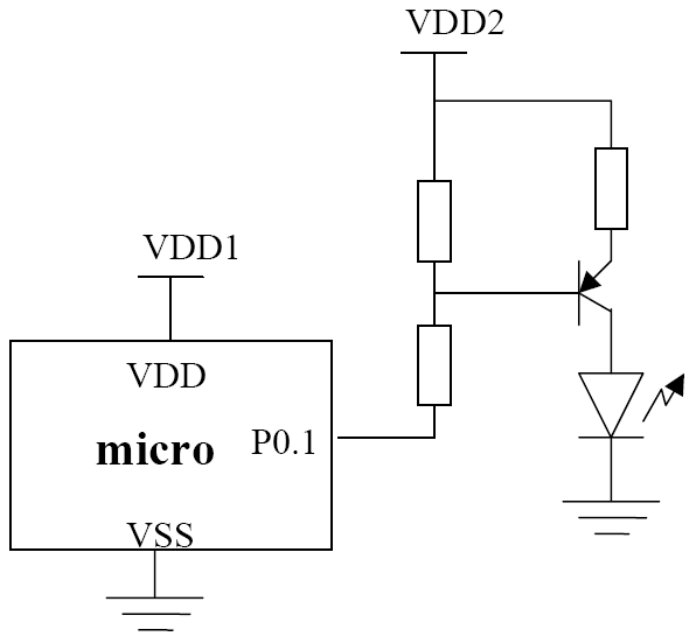




Giao tiếp với LED đơn

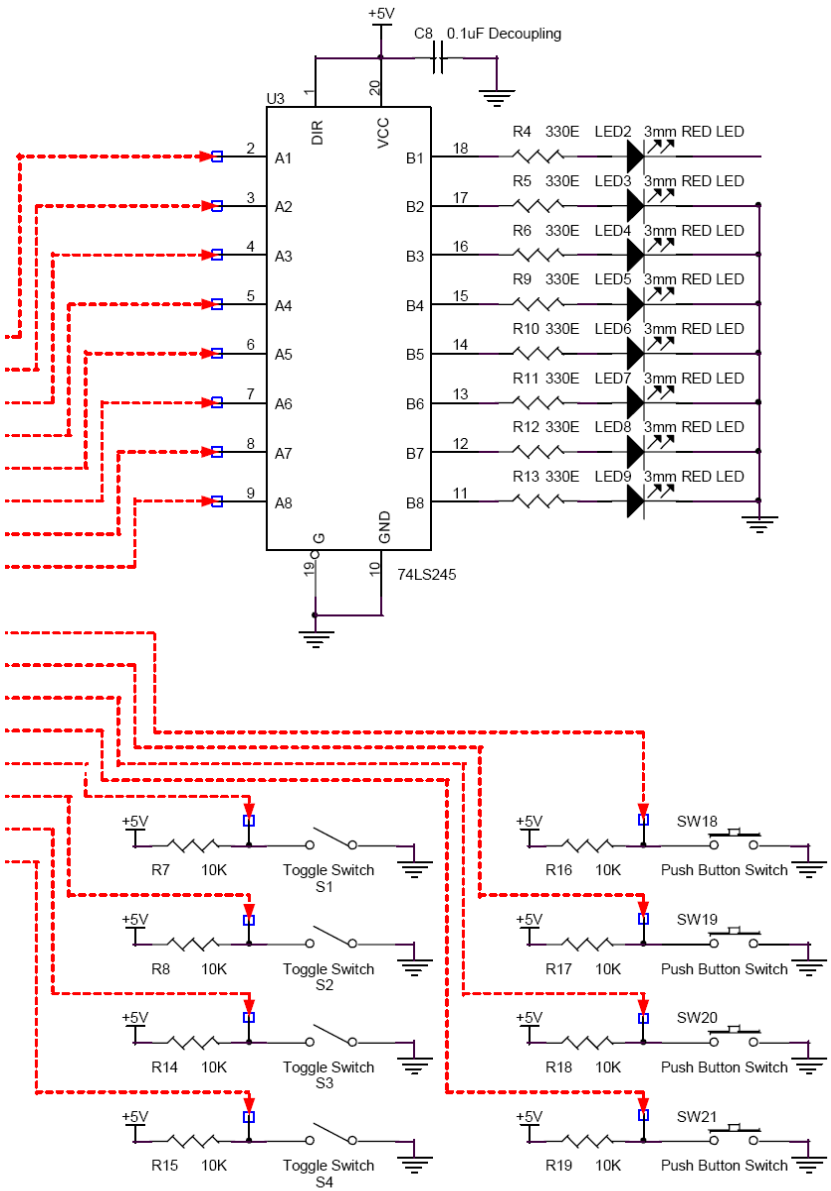


Direct hardware interface
20mA max sink current
Configure as quasi-bidir or
Push-pull

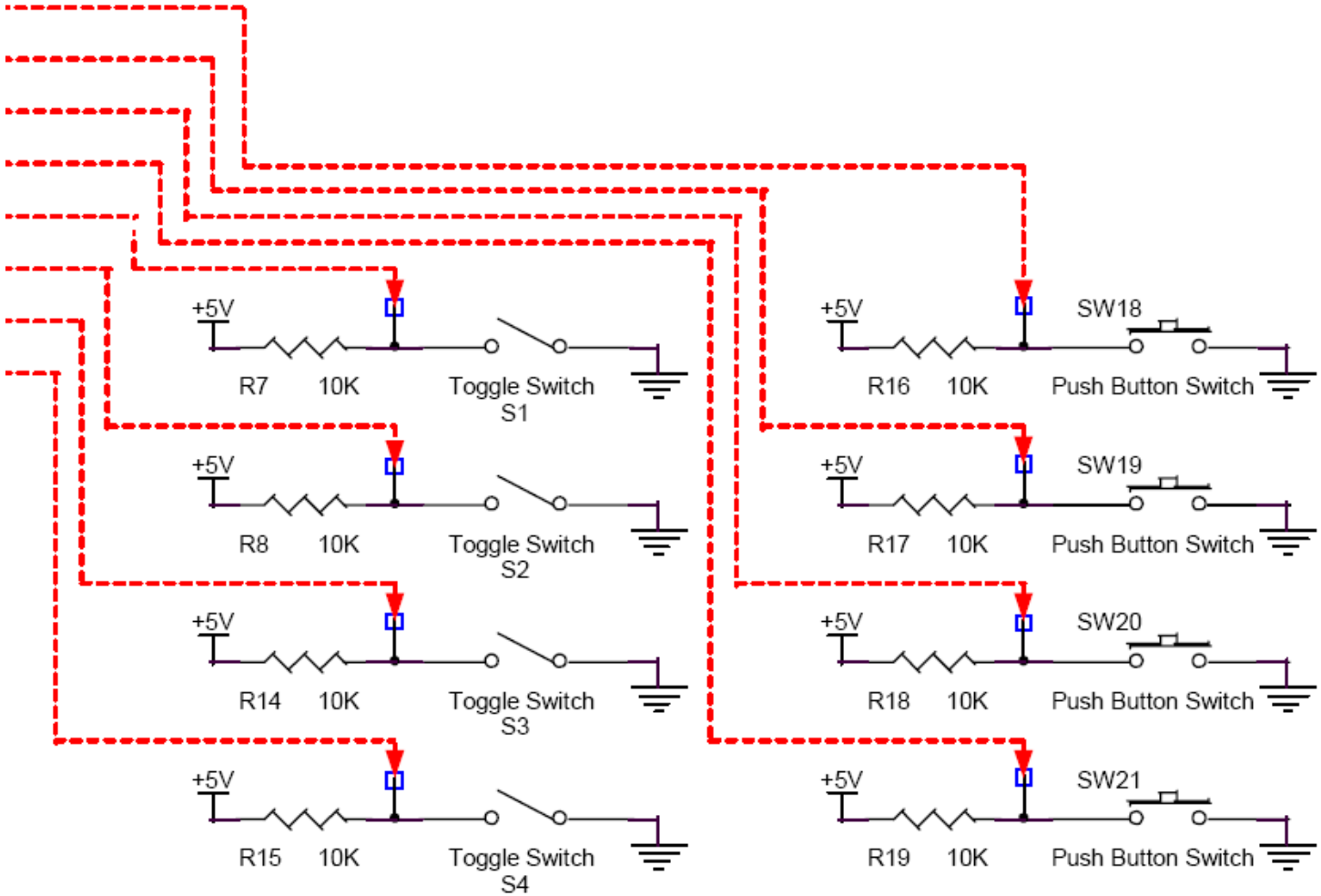


Interfacing to 5V hardware
Configure as open-drain
5V supply drives load

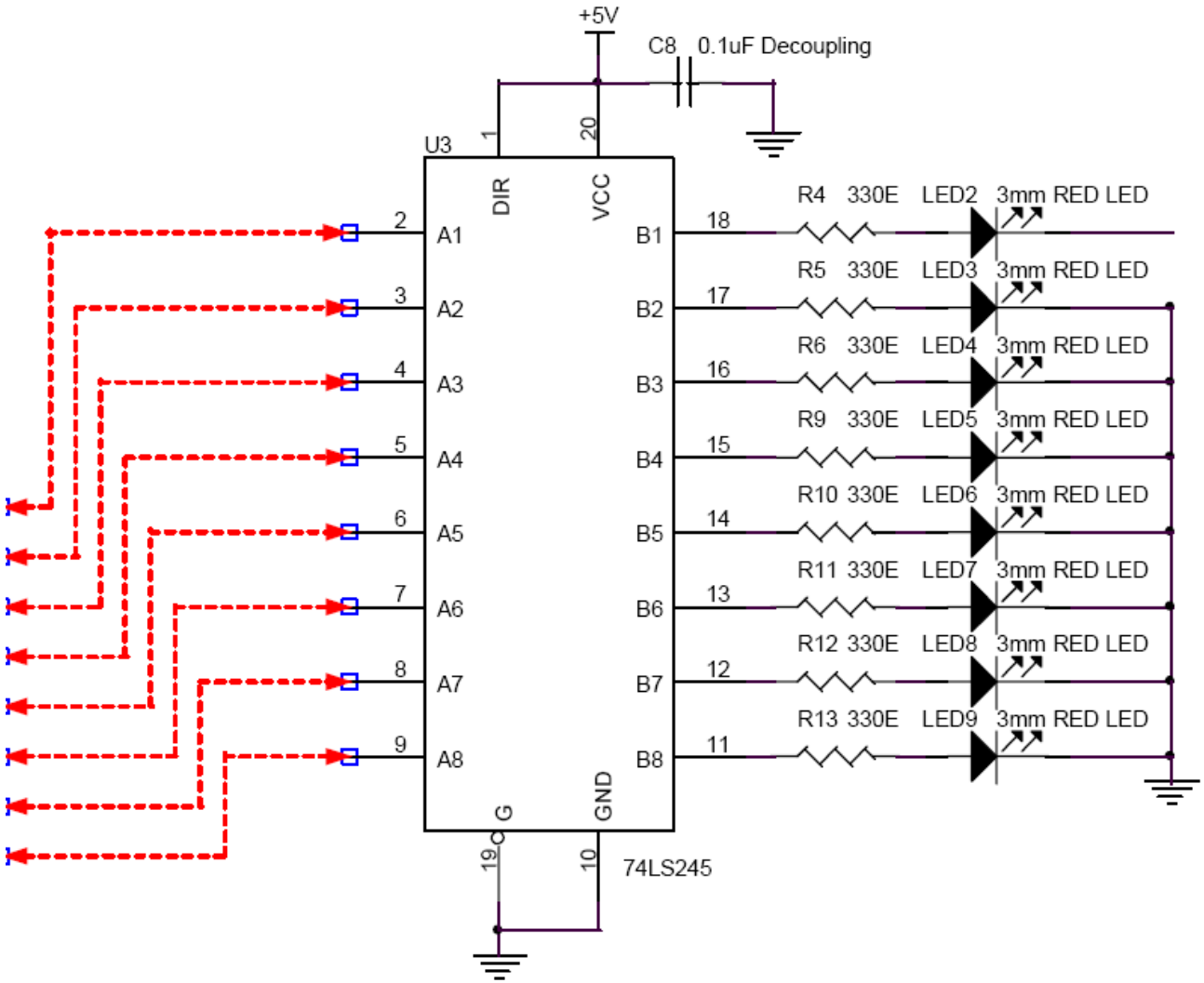
TD: Mạch cho nhập từ phím nhấn và xuất ra LED đơn



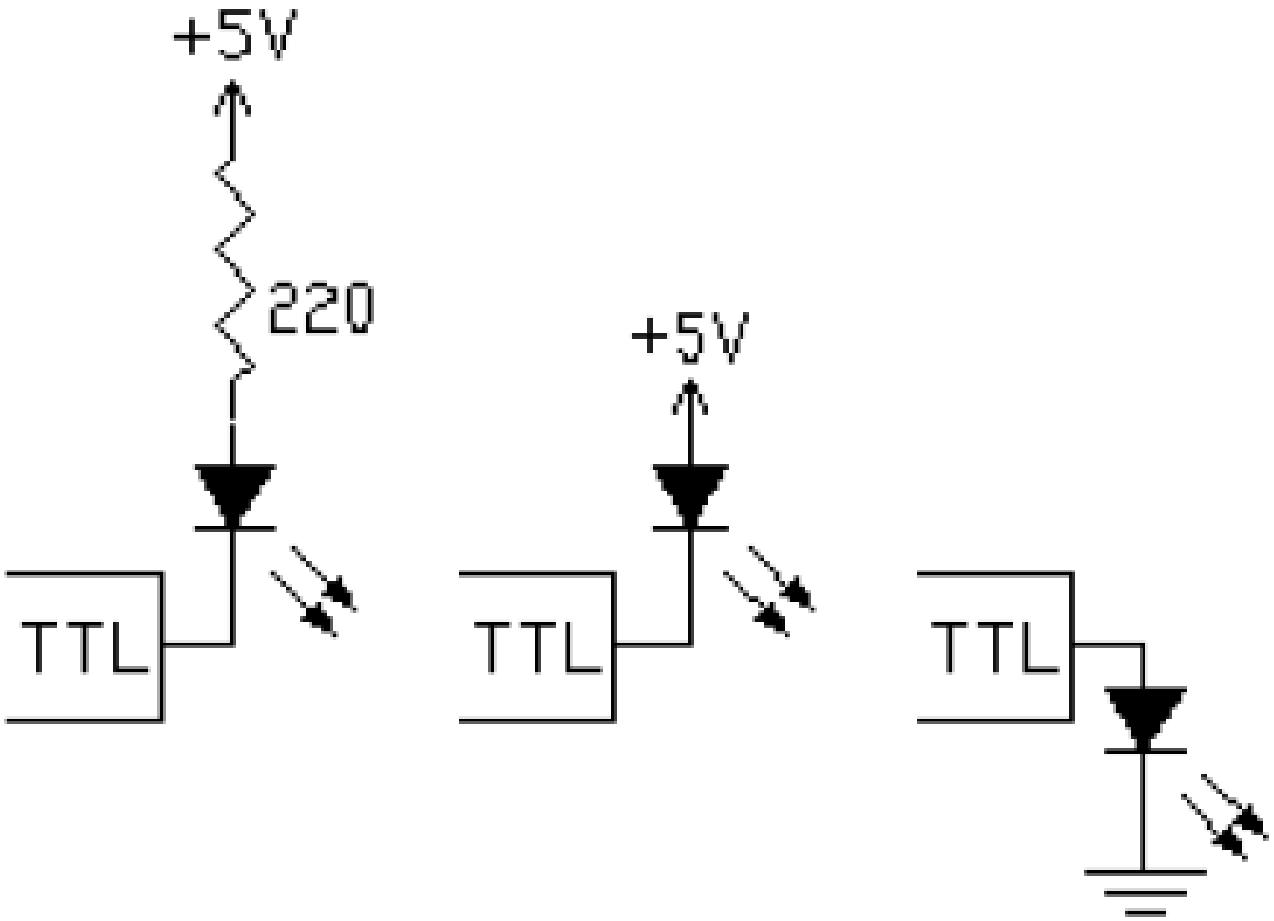
Mạch nhập từ phím nhấn



Mạch xuất ra LED đơn



LED ON I/O PORTS (1/2)



LED ON I/O PORTS (2/2)

Since TTL outputs is designed to feed multiple TTL inputs, they are good at current sinking but poor at current sourcing. The Standard TTL can sink up to 16mA and source 250uA. The LS logic family can sink 8mA and source 100uA. The 8051 port pin can sink 1.6mA (3.2mA for port 0) and source 60uA. Therefore, if you drive significant current, try to arrange your circuits to use current sinking. Unlike diodes, Light-emitting diodes have a forward voltage drop from 1.7 to 2.5 volts and most of them flow a forward current 20mA.

Poor circuit

since the TTL output can't source above 1mA so the LED will be very dim.

Fair circuit

The LED will conduct heavily at about 2V and the extra 3V has to be dropped in the TTL circuitry. This causes high power dissipation in the TTL or the LED fails.

Good circuit

The resistor limits the current. The resistance can be calculated by assuming its voltage is about 2.5V and the TTL output is 0.9V. For 2.2V LED, 1.9V is across the resistor so the 220ohm would limit the current to 8.6mA (1.9/220). For 1.7V LED, 2.4V is across the resistor so it would limit the current to 10.9mA (2.4/220). The resistor should not less than 100 Ohm or the LED would fail.

Example 1

- **Connection** -Switch -P1.0, LED - P2.0
- **Condition** - Turn on LED when switch is pressed.
- **Solution:**

```
            SETB P2.0  ; LED OFF
            SETB P1.0 ; input pin.
LOOP:   JB     P1.0, LOOP  ; not grounded then
                                ; stay in loop
            CLR P2.0      ; To clear pin P0.0 when
                                ; P1.0 is at 0 v
```

- **Schematic Circuit?**

Ex: RELAY ON I/O PORT (2CO Relay)

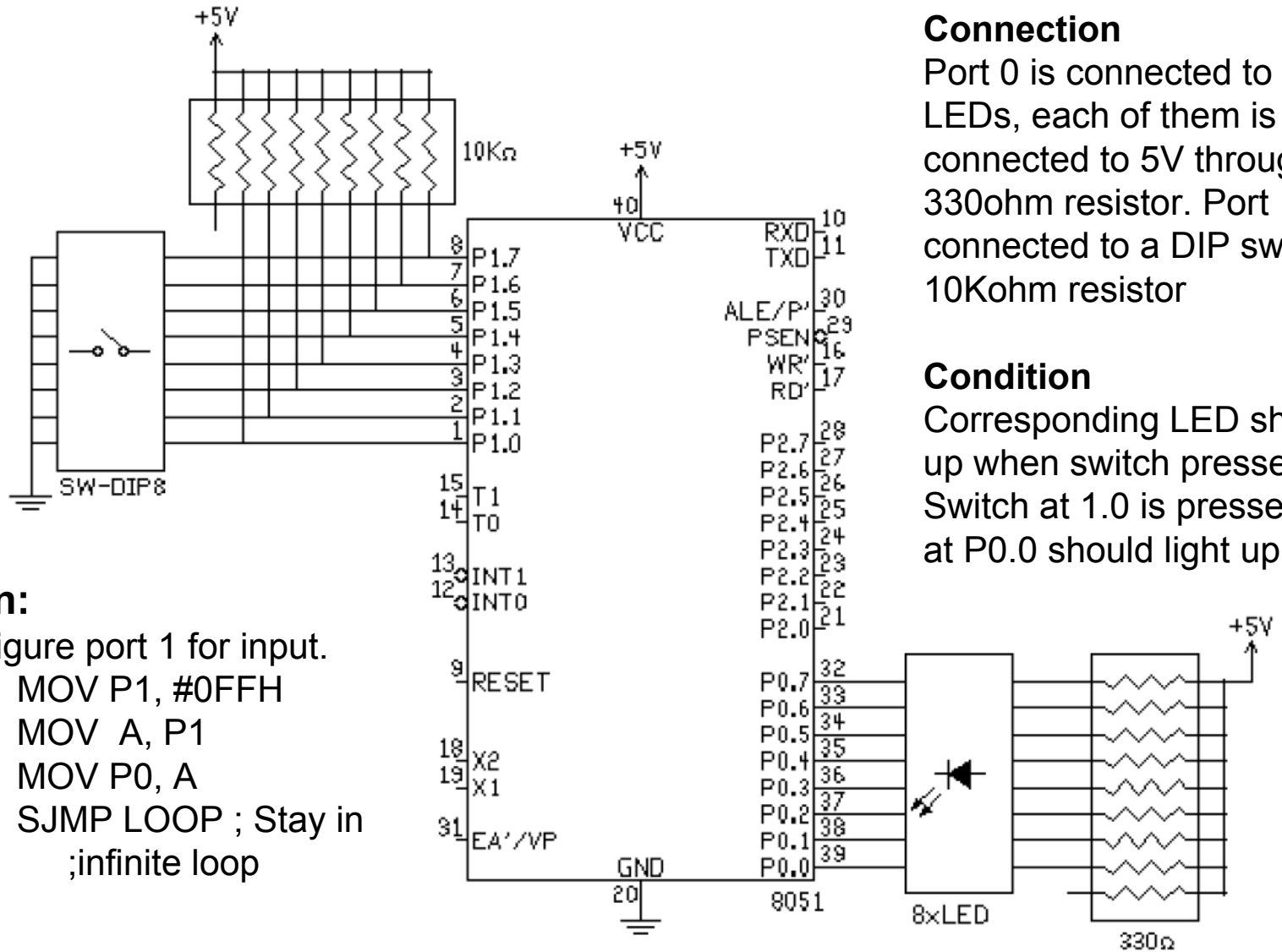
In A, NPN transistor (say a BC337 or BC338) is being used to control a relay with a 5 V coil. Series base resistor R1 is used to set the base current for Q1, so that the transistor is driven into saturation (fully turned on) when the relay is to be energized. That way, the transistor will have minimal voltage drop, and hence dissipate very little power as well as delivering most of the 5V to the relay coil.

How do work out the value of R1?.

Let us say RLY1 needs 50mA of coil current to pull in and hold reliably, and has a resistance of 24 Ohms so it draws this current from 5V. Our BC337/338 transistor will need enough base current to make sure it remains saturated at this collector current level. To work this out, we simply make sure that the base current is greater than this collector current divided by the transistors minimum DC current gain hFE. So as the BC337/338 has a minimum hFE of 100 (at 100mA), we'll need to provide it with at least 50mA/100 = 0.5mA of base current.

In practice, you give it roughly double this value, say 1mA of base current, just to make sure it does saturate. So if your resistance will be TTL Logic High Voltage (Min) /1ma (1K approx)

Example 2 – Switch and LED



Connection

Port 0 is connected to eight LEDs, each of them is connected to 5V through a 330ohm resistor. Port 1 is connected to a DIP switch and a 10Kohm resistor

Condition

Corresponding LED should light up when switch pressed , i.e. if Switch at 1.0 is pressed -> LED at P0.0 should light up.

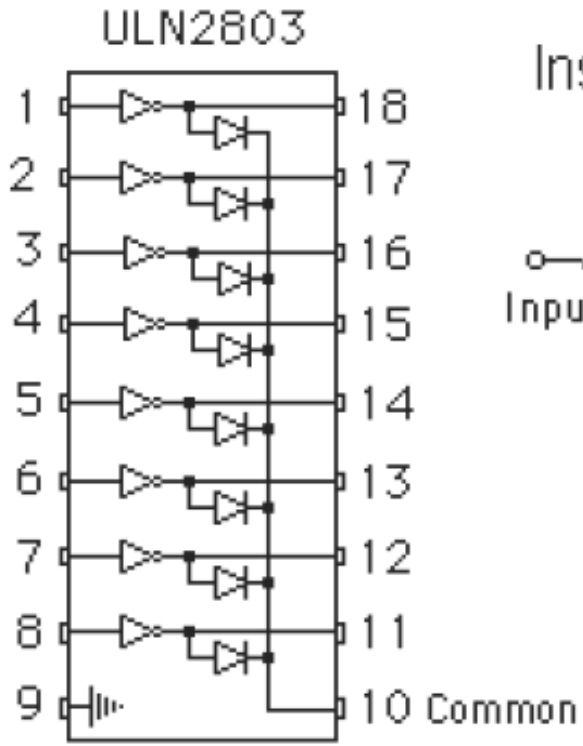
Solution:

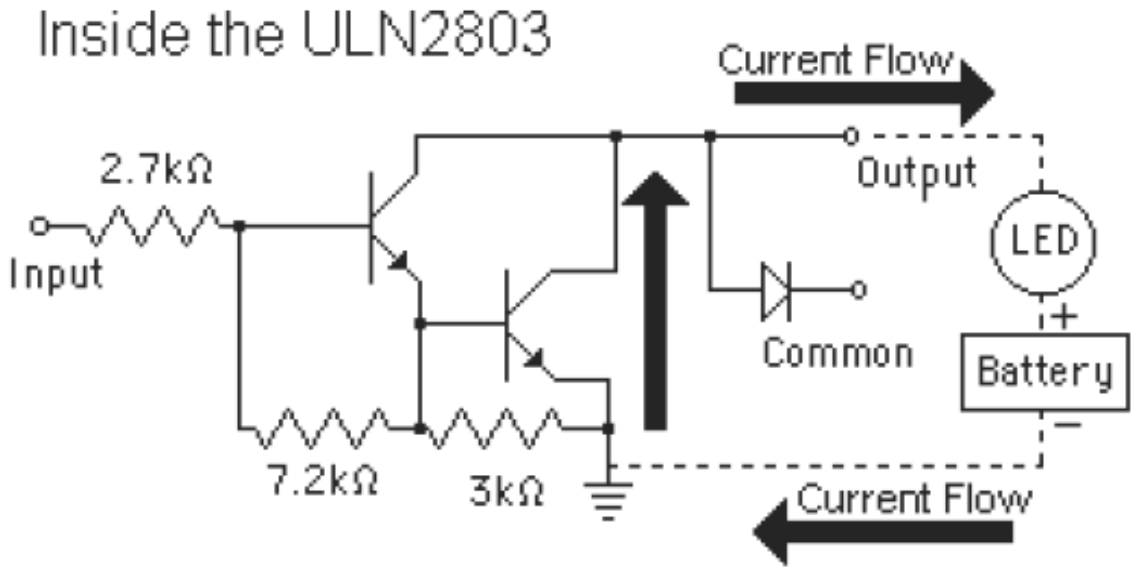
```
; To configure port 1 for input.
          MOV P1, #0FFH
LOOP:     MOV  A, P1
          MOV P0, A
          SJMP LOOP ; Stay in
              ;infinite loop
```

Using ULN (Driver IC)

Another option for driving relays would be to use a high-voltage, high-current, Darlington array driver IC such as the ULN2803. The ULN2803 can directly interface to the data outputs of the 8051 pins, and provides much higher drive-current. The ULN2803 also has internal diode protection that eliminates the need for the fly-back diode as shown in the above relay driver schematics. You can connect 8 relay using this IC.

So ULN is better choice if you have more than 3 relay. (Simple design of circuit & PCB as well !)





From 1995 Dick Smith Catalogue

7 Segment Display

INTRODUCTION

For the seven segment display you can use the LT-541 or LSD5061-11 chip (etc…). Each of the segments of the display is connected to a pin on the 8051 (the schematic shows how to do this). In order to light up a segment on the the pin must be set to 0V. To turn a segment off the corresponding pin must be set to 5V. This is simply done by setting the pins on the 8051 to '1' or '0'.

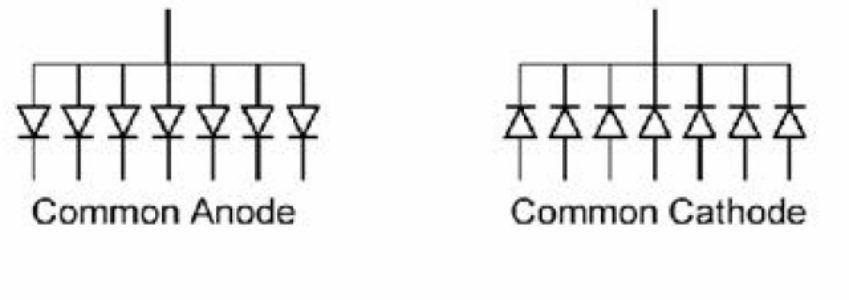
LED displays are

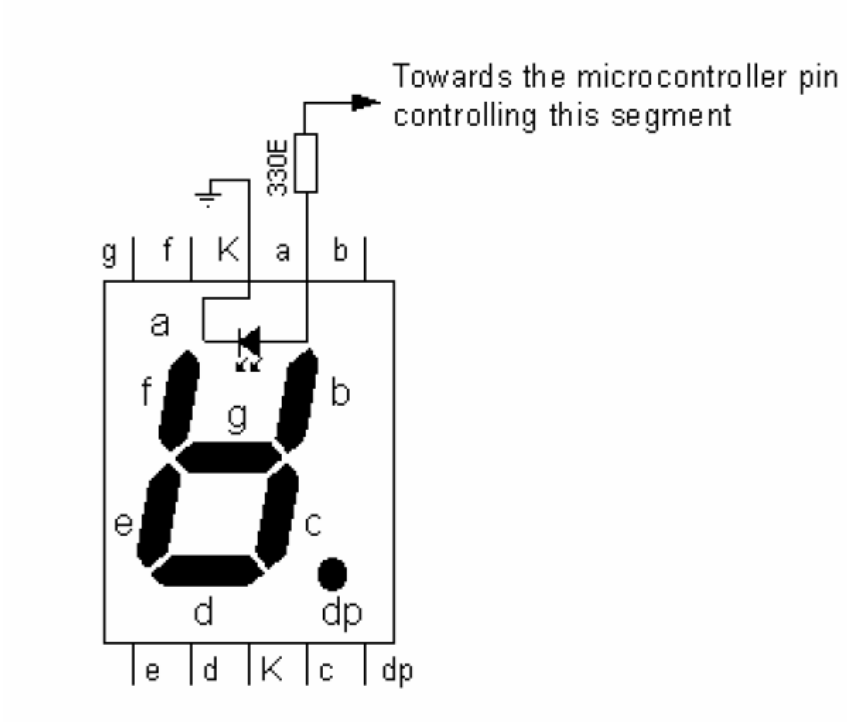
- Power-hungry (10ma per LED)
- Pin-hungry (8 pins per 7-seg display)

But they are cheaper than LCD display

7-SEG Display are available in two types: Common anode (CA) & common cathode (CC), but command anode display are most suitable for interfacing with 8051 since 8051 port pins can sink current better than sourcing it.

7 Segment Display

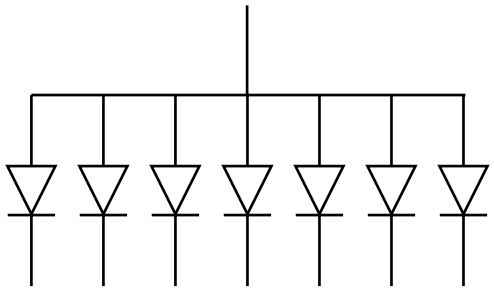




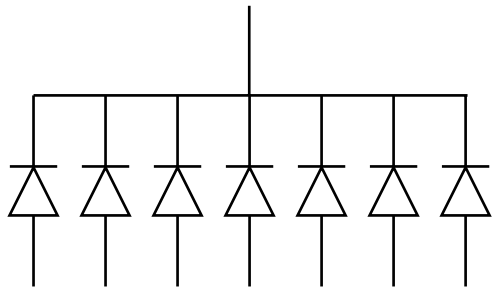
dp = decimal point

The 7-segment Display (Cont.)

- 7-segment displays come in 2 configurations:



Common Anode (CA)

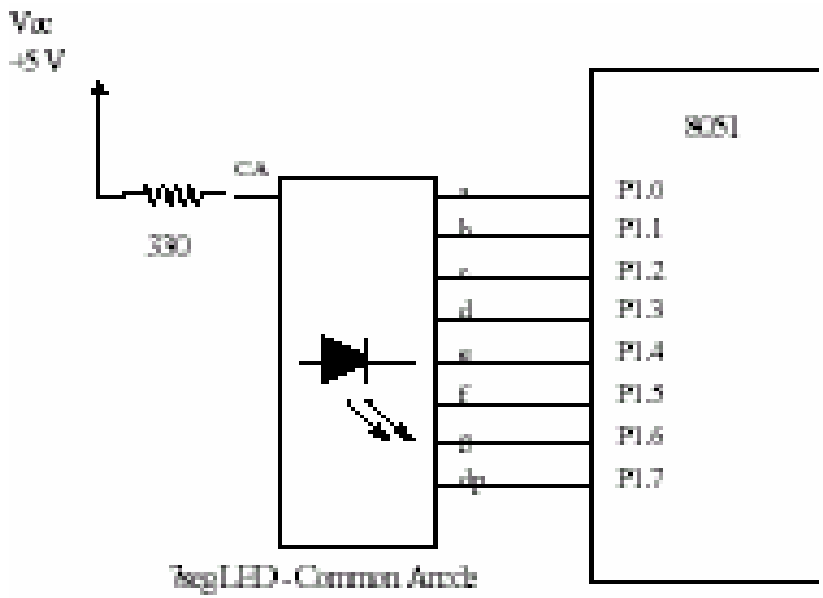


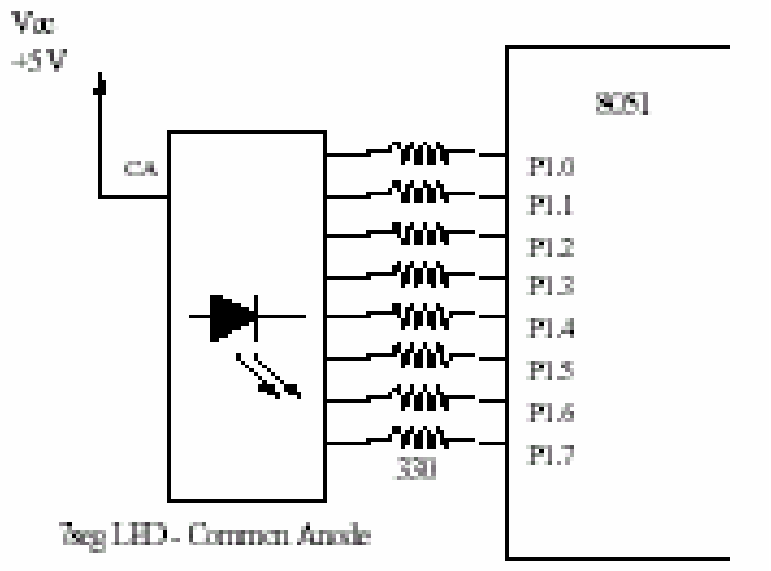
Common Cathode (CC)

- As we have seen, it would be preferable to connect the cathode of each diode to the output pin.
- Therefore, the common anode variety would be better for our interfacing needs.

Interfacing a 7-segment display

- Also, as seen with interfacing the LED, a resistor will be needed to control the current flowing through the diode.
  - This leaves two possibilities:





  - Case 2 would be more appropriate as 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

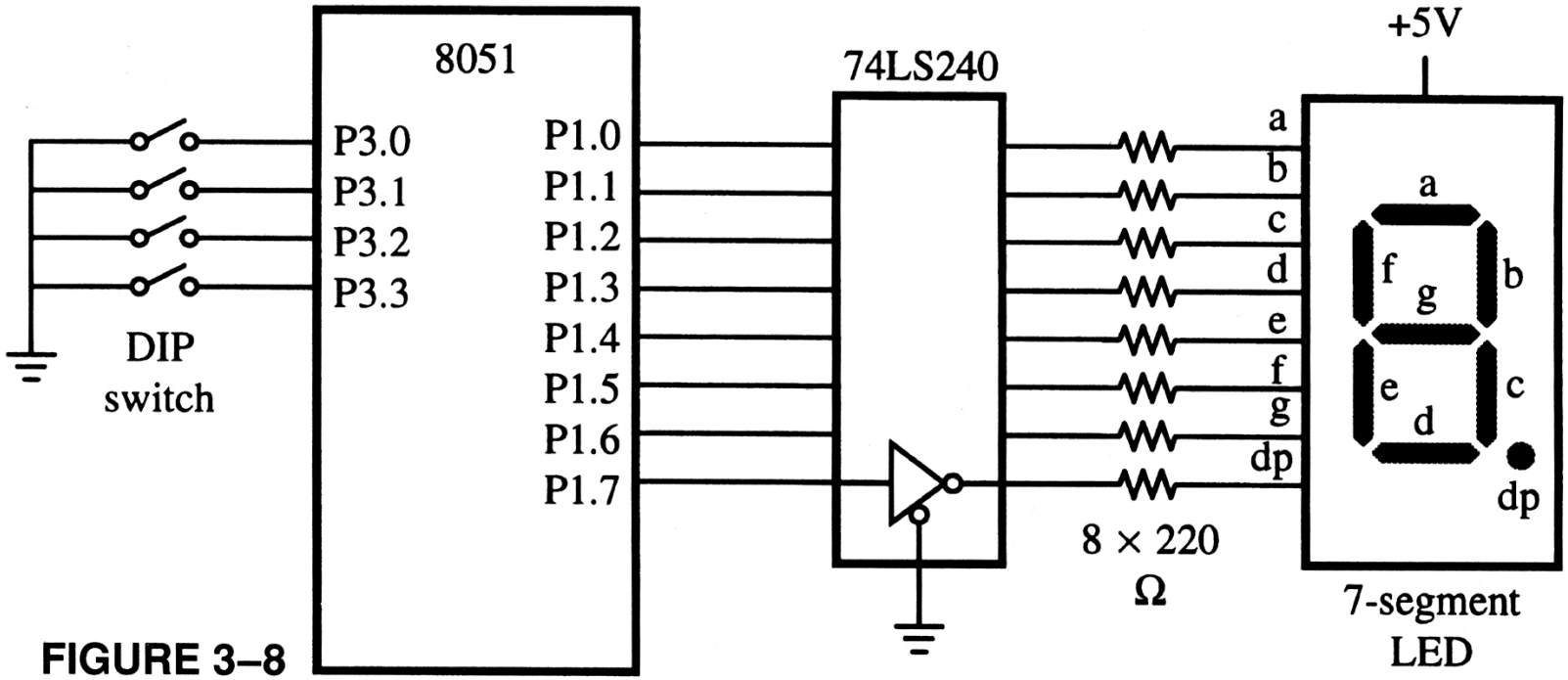


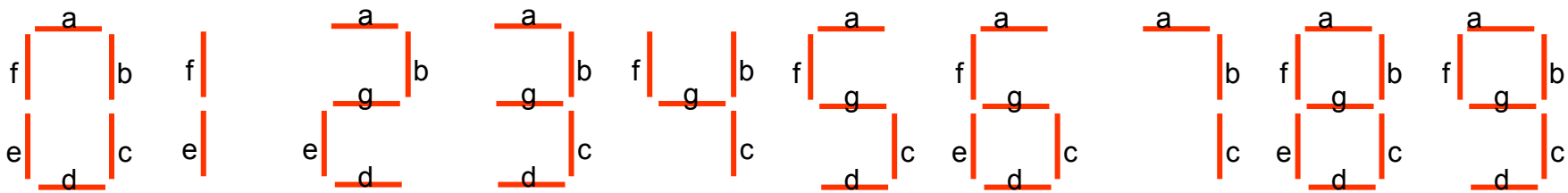
FIGURE 3–8
Interface to a DIP switch and 7-segment LED

BCD to 7-Seg lookup table

```
        mov   p3,#0fh
        mov   a,p3
        anl   a,0fh
get_code: mov   DPTR, #7s_tab
        movc  A, @A+DPRT
        mov   p1,a

7s_tab:   db 3fh,30h,5bh,4fh,66h
        db 6dh,7dh,07h,7fh,6fh
        END
```

BCD	p g f e d c b a 7_seg	hex
0000	0 0 1 1 1 1 1 1	3f
0001	0 0 1 1 0 0 0 0	30
0010	0 1 0 1 1 0 1 1	5b
0011	0 1 0 0 1 1 1 1	4f
0100	0 1 1 0 0 1 1 0	66
0101	0 1 1 0 1 1 0 1	6d
0110	0 1 1 1 1 1 0 1	7d
0111	0 0 0 0 0 1 1 1	07
1000	0 1 1 1 1 1 1 1	7f
1001	0 1 1 0 1 1 1 1	6f



Creating Digit Pattern with 7-segment LED Display

For displaying Digit say 7 we need to light segments: a ,b, c. Since we are using Common anode display , to do so we have to to provide Logic 0 (0 v) at anode of these segments.
So need to clear pins: P1.0 ,P1.1,P1.2. that is 1 1 1 1 1 0 0 0 ➔ F8h .

Connection

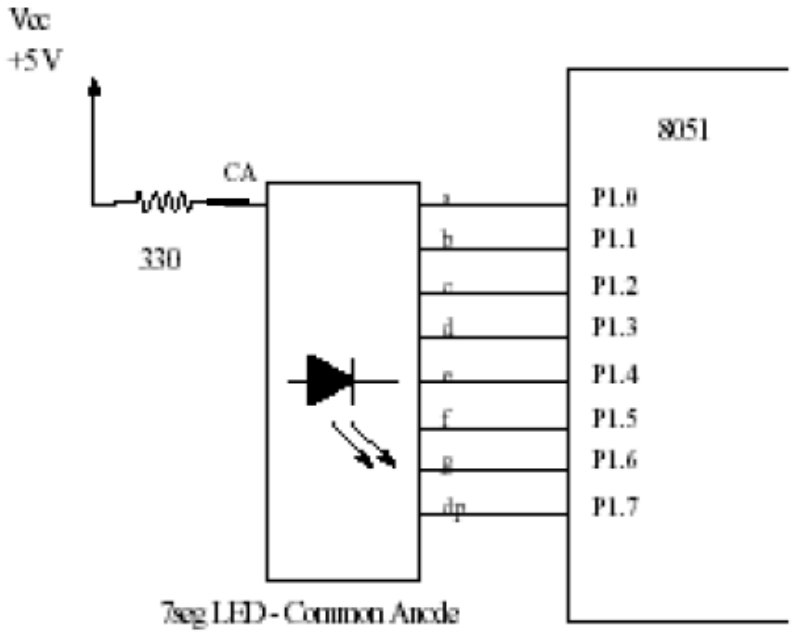
Segment number	8051 pin number
a	P1.0
b	P1.1
c	P1.2
d	P1.3
e	P1.4
f	P1.5
g	p1.6
h(dp)	P1.7

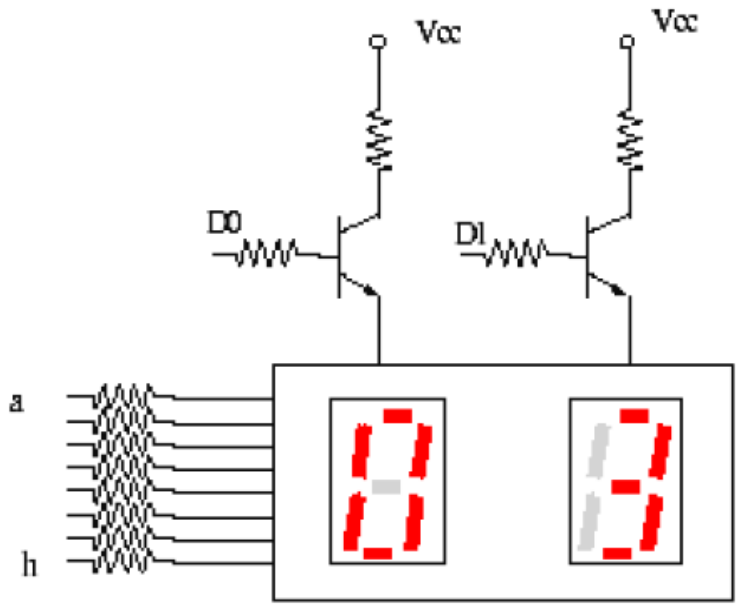
Hex Code

Digit	Seg. h	Seg. g	Seg. f	Seg. e	Seg. d	Seg. c	Seg. b	Seg. a	HEX
0	1	1	0	0	0	0	0	0	C0
1	1	0	0	0	0	1	1	0	06
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99

You can also do this for some characters like A ,E .. but not for D or B because it will be same as that of 0 & 8 . So this is one of limitation of 7-seg display.
Since we can Enable only one 7-seg display at a time ,we need to scan these display at fast rate .The scanning frequency should be high enough to be flicker-free. At least 30HZ .Therefore – time one digit is ON is 1/30 seconds

INTERFACING TO LED DISPLAY (1/2)





Common Anode display

Note that we are using Common Anode display. so the common Anode pin is tied to 5v .The cathode pins are connected to port 1 through 330 Ohm resistance (current limiting).

INTERFACING TO LED DISPLAY (2/2)

Connection: a:h to port p1.0:p1.7 , D0:D1 to p3.0:p3.1.

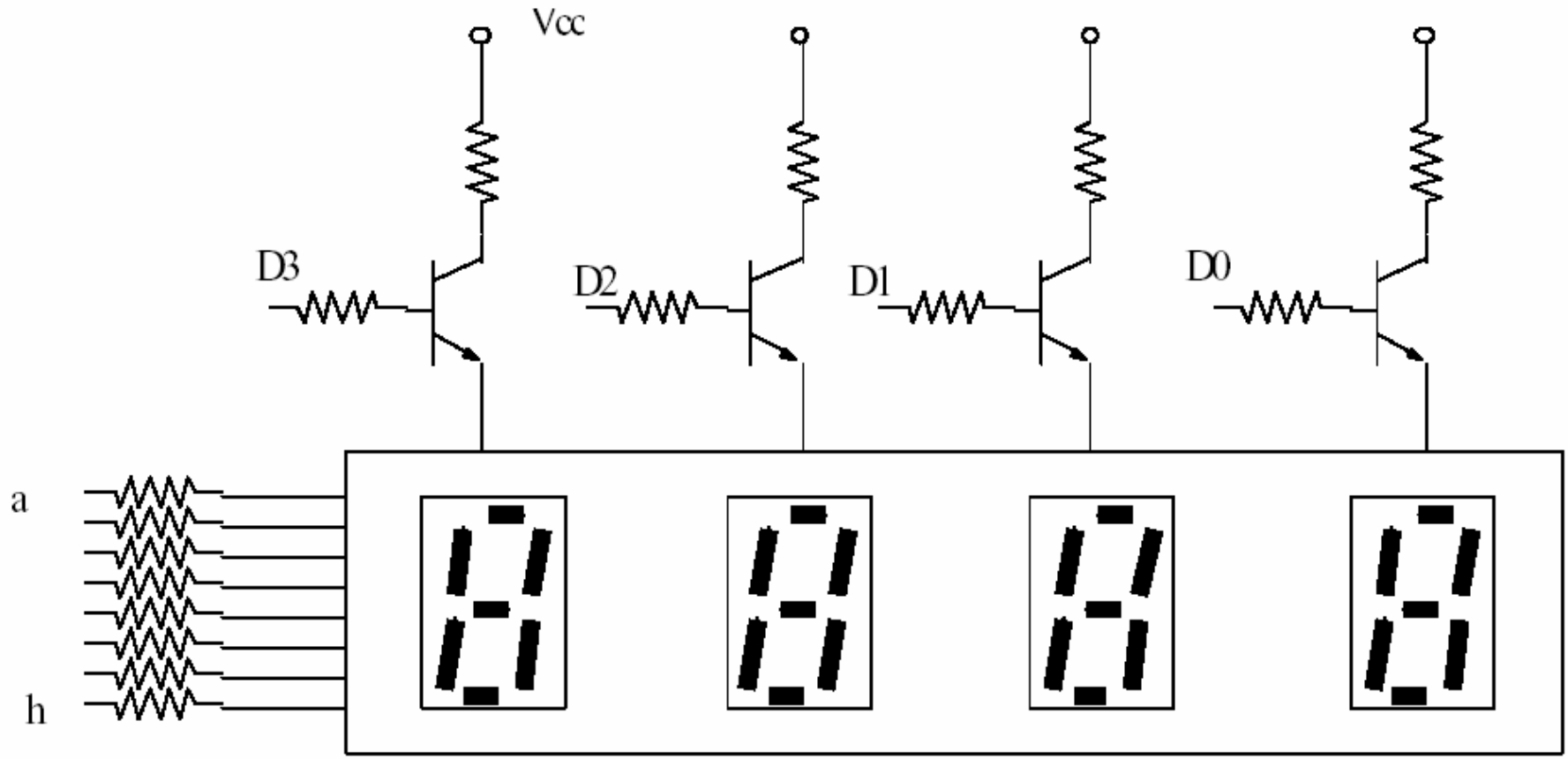
To Display: Consider example of vending machine where we want to display number of soft drink bottles on display entered by customer. Suppose he enter 3 (03) bottles then we will use lookup table to see DIGIT PATTERN of these keys.

So DIGI[1]=C0 (hex code for '0') &
DIGI[2]=B0(hex code for '3').

Algorithm

start : Disable [D0:D1]
again : Enable D0
[a:h] - pattern for Digit1
Delay
Disable D0.
Enable D1
[a:h] - pattern for Digit2
Delay
Goto **again**

Hiển thị quét LED với 4 LED 7 đoạn

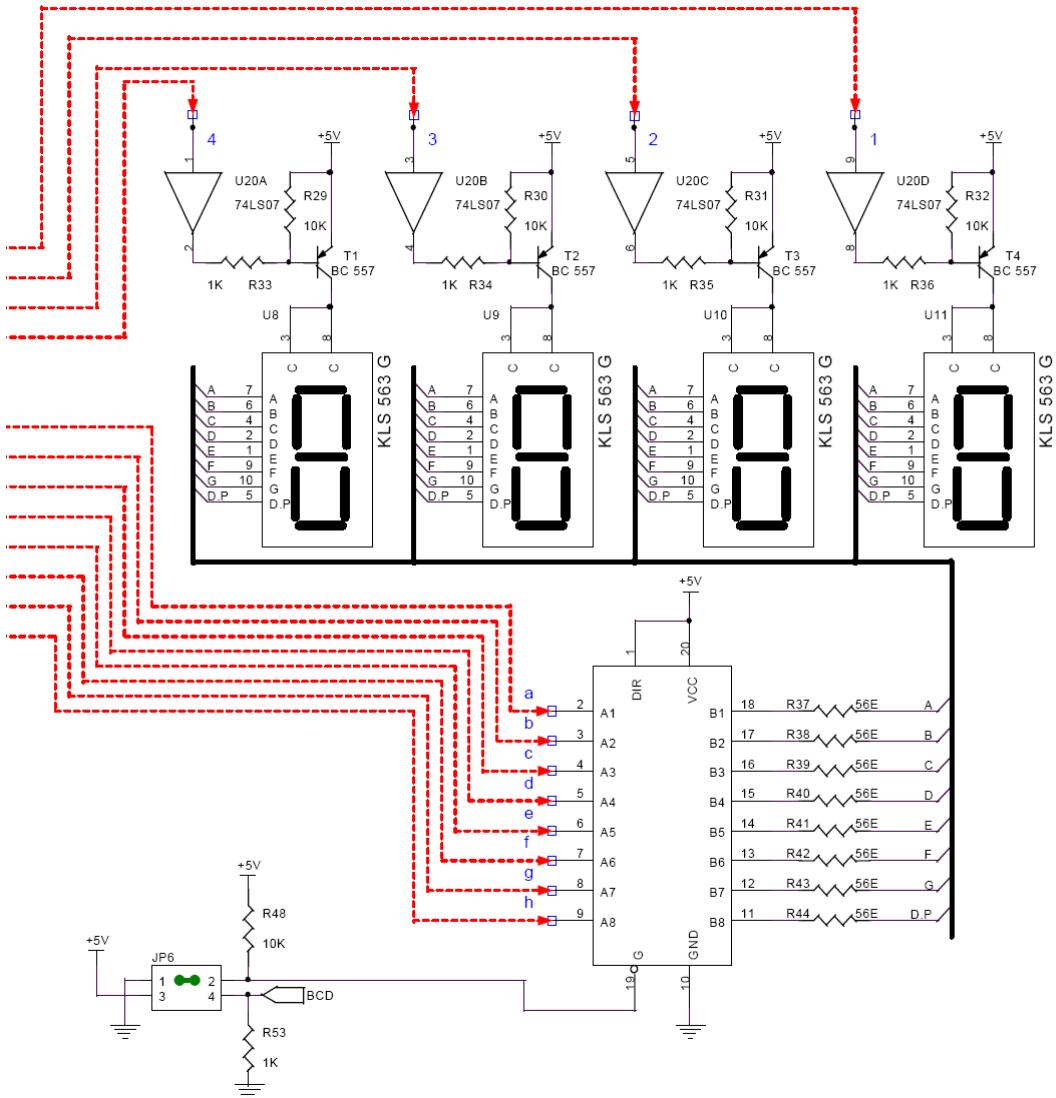


Chú ý với hiển thị dồn kênh

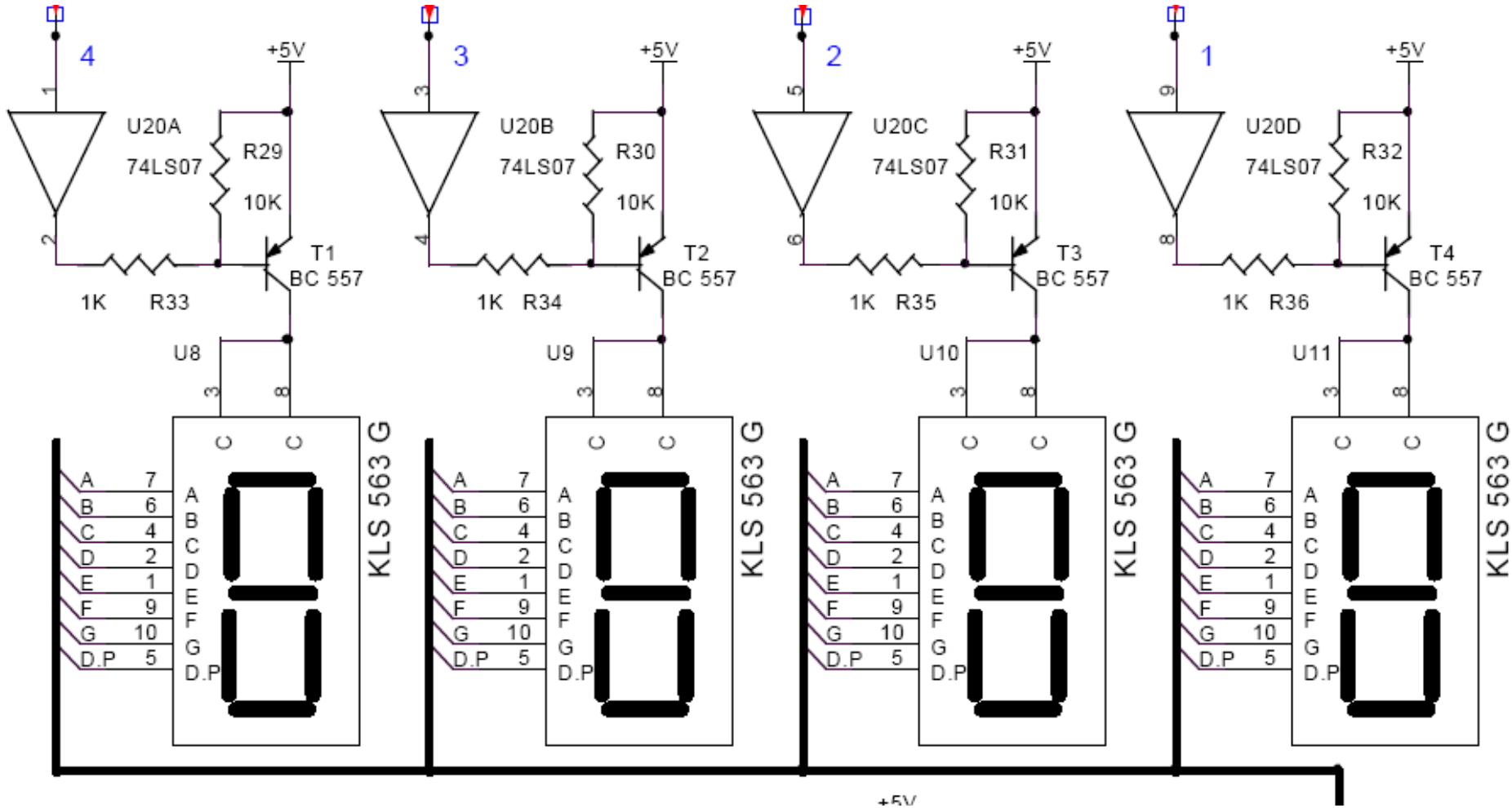
Với hiển thị LED dồn kênh:

- Ở mỗi thời điểm chỉ có một hiển thị LED 7 đoạn được cho phép (qua các khóa điện tử BJT).
- Các ngõ vào a-h nối chung với nhau cho tất cả các LED 7 đoạn.
- Tổng số chân cổng cần sử dụng là 8 + số ký số (digit), với thí dụ hình 5.80 thì tổng số chân là 8 + 4 = 12.
- Tần số quét phải đủ cao để tránh tình trạng thấy LED nhấp nháy:
    - tối thiểu 40Hz
    - thời gian cho 1 digit sáng là 1/40 giây.
    - tần số quét cao hơn thì sẽ giảm sự nhấp nháy

TD: Mạch hiển thị LED 7 đoạn của **www.MightyMicons.com**

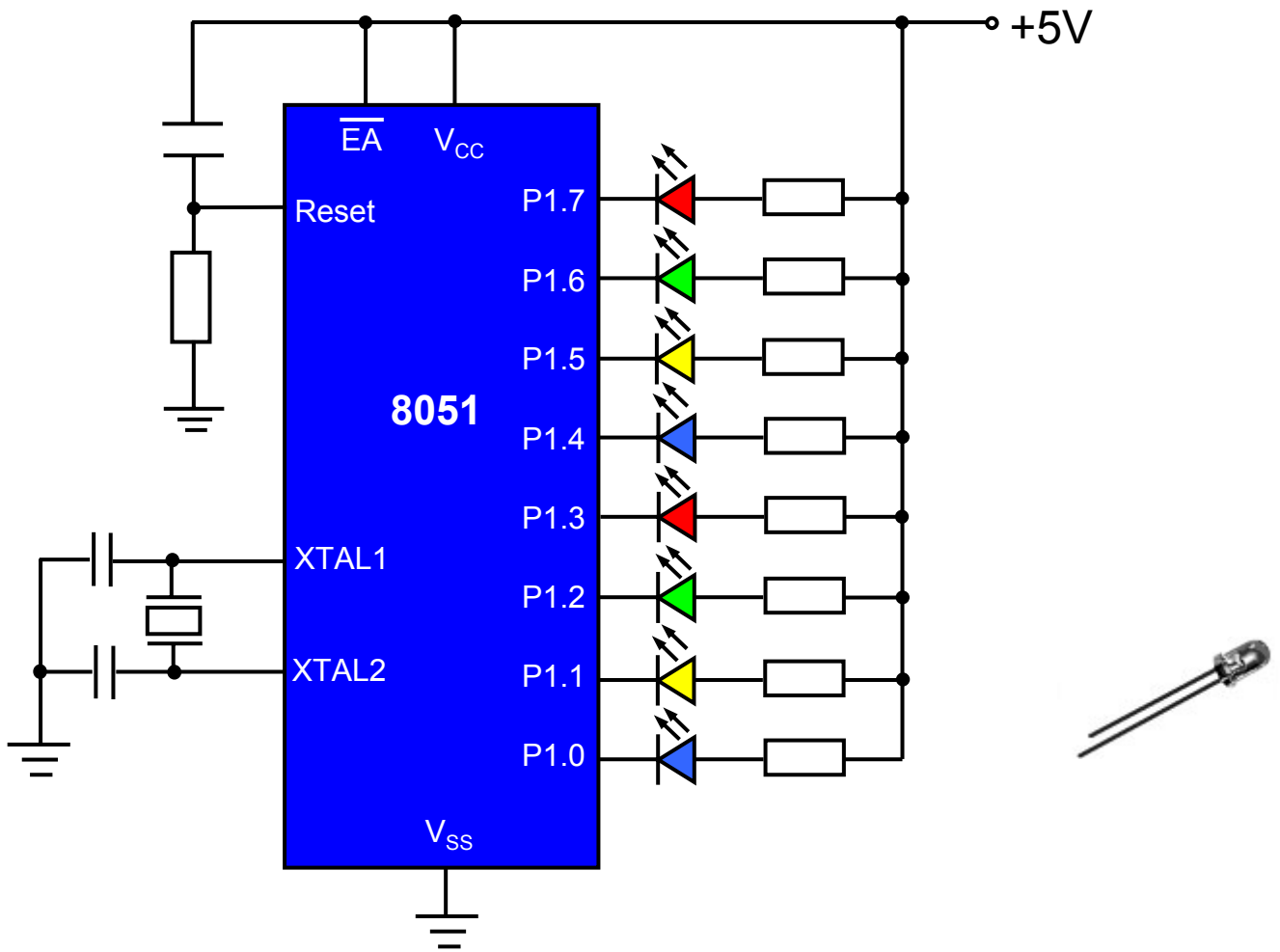


Mạch này dùng LED 7 đoạn loại nào? (CA hay CC)

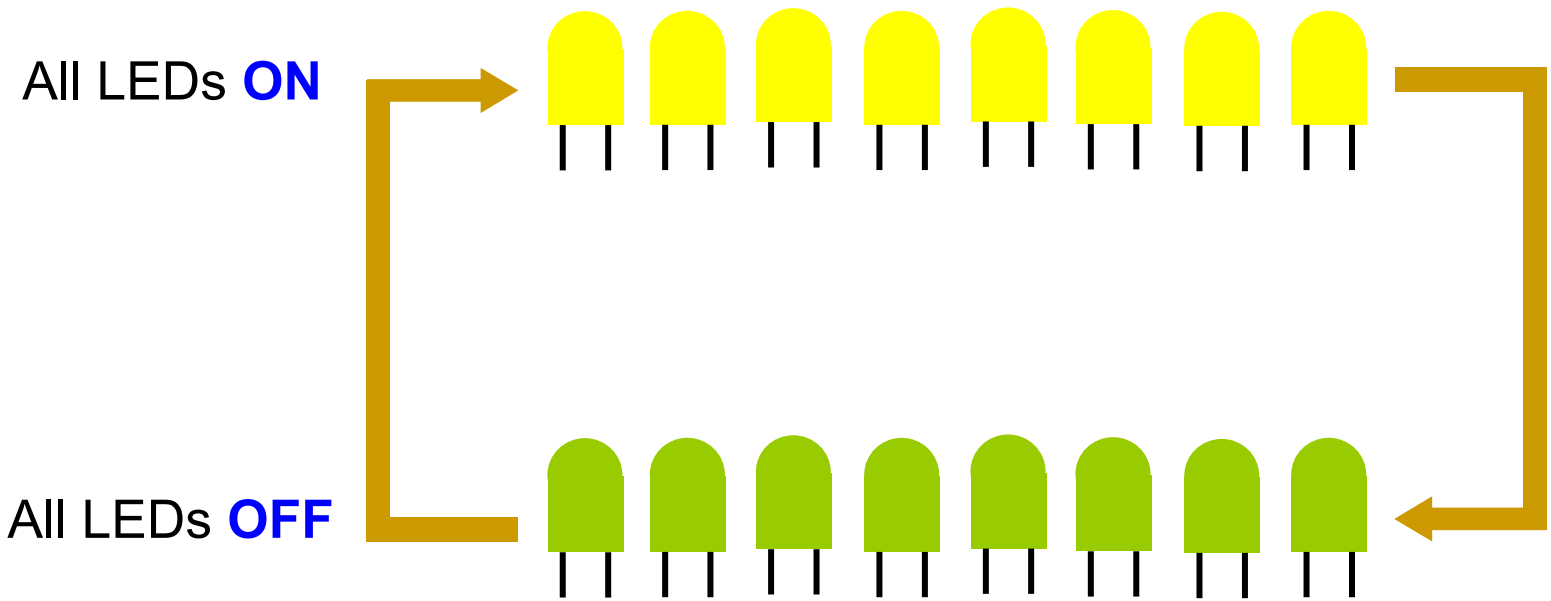


Một số thí dụ với LED và phím nhấn

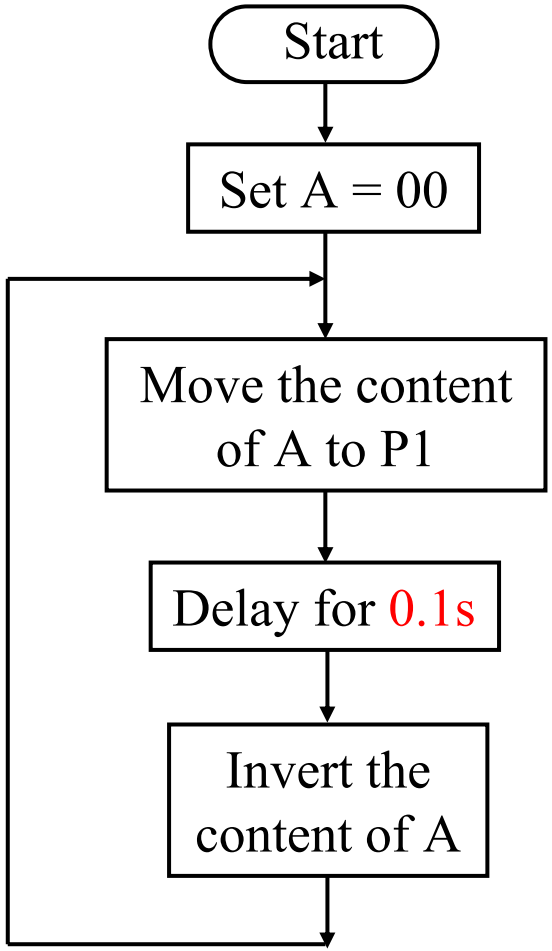
Basic Output Techniques with LEDs



Example 3: Light-up LEDs



Program Listing for Example 3

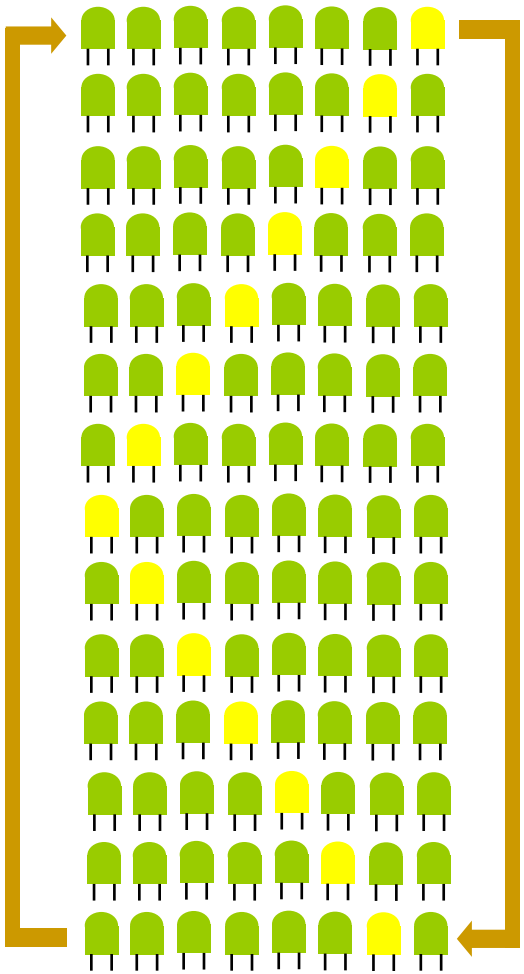



```
        ORG     0000H
        CLR     A
LOOP:   MOV     P1, A
        CPL     A
        ACALL   DELAY
        AJMP    LOOP
DELAY:  MOV     R6, #250
DL1:    MOV     R7, #200
DL2:    DJNZ    R7, DL2
        DJNZ    R6, DL1
        RET
        END
```

Assume 12MHz clock, determine the delay time.

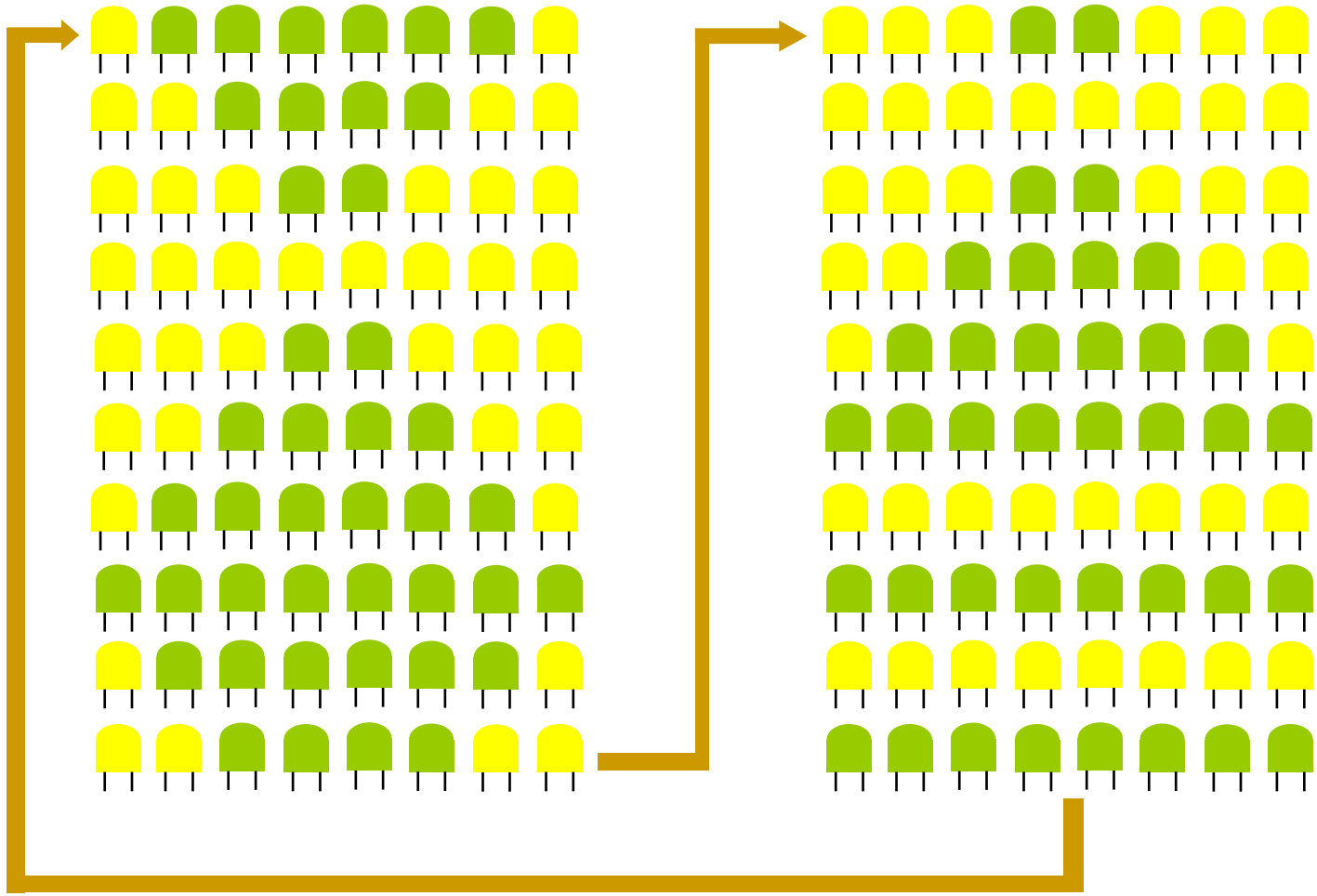
Time delay, Tex1 = 1 + [(1+ 200 * 2) + 2] * 250 + 2 = 100,753 MachineCycle

Example 4: Lighting Sequence



```
        ORG     0000H
START:  MOV     R1, #07H
        MOV     A, #11111110B
LEFT:   MOV     P1, A
        ACALL   DELAY
        RL      A
        DJNZ    R1, LEFT
        ;
        MOV     R1, #07H
        MOV     A, #01111111B
RIGHT:  MOV     P1, A
        ACALL   DELAY
        RR      A
        DJNZ    R1, RIGHT
        AJMP    START
        ;
DELAY:  ..........
```

Example 5: Use a Look-up Table

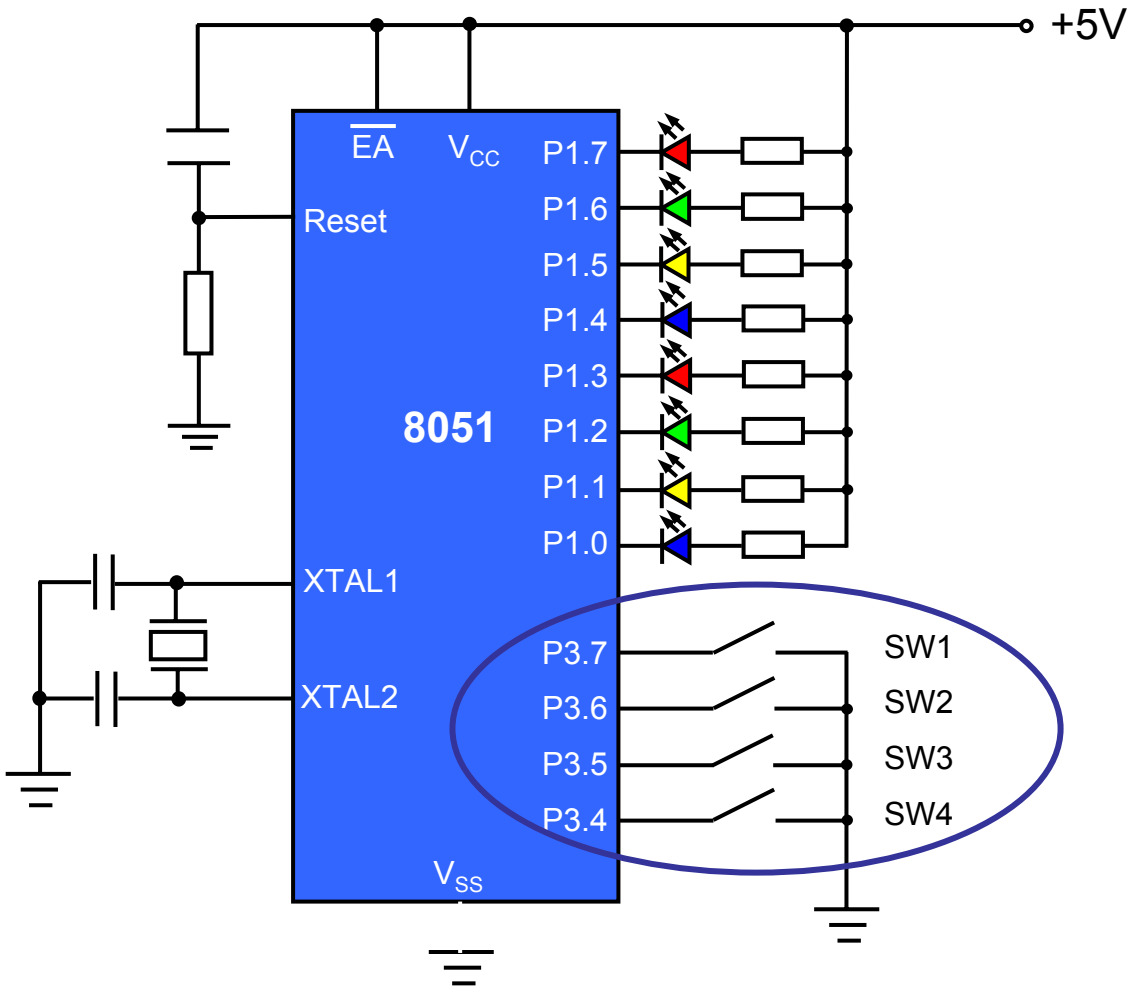


Program Listing of Example 5

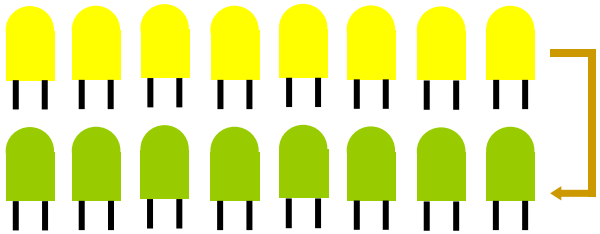
```
                ORG         0000H
START:          MOV         R0, #OK–LUT+1     ; length of table
                MOV         DPTR, #LUT        ; code start address
                MOV         R1, #00H
LOOP:           MOV         A, R1
                MOVC        A, @A+DPTR
                MOV         P1, A
                ACALL       DELAY
                INC         R1                ; point to next data of table
                DJNZ        R0, LOOP          ; finish ?
                AJMP        START
                ;
DELAY:
;  do not modify the value in register R0, & R1
;  DELAY =  1 + (100751 + 2 ) * 2 + 2 = 201,509 Machine Cycle !
                MOV         R5, #2
DL1:            MOV         R6, #250
DL2:            MOV         R7, #200
DL3:            DJNZ        R7, DL3
                DJNZ        R6, DL2
                DJNZ        R5, DL1
                RET
                ;
```

```
LUT: DB         01111110B
     DB         00111100B
     DB         00011000B
     DB         00000000B
     DB         00011000B
     DB         00111100B
     DB         01111110B
     DB         11111111B
     ;
     DB         01111110B
     DB         00111100B
     DB         00011000B
     DB         00000000B
     DB         00011000B
     DB         00111100B
     DB         01111110B
     DB         11111111B
     ;
     DB         00000000B
     DB         11111111B
     DB         00000000B
OK:  DB         11111111B
     END
```

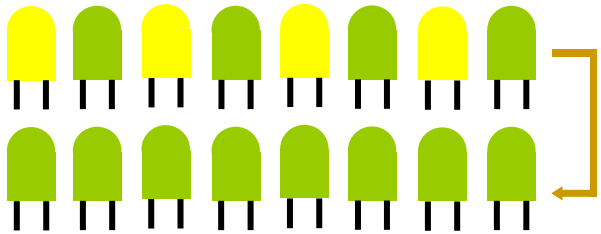
Example 6: Basic Input Technique



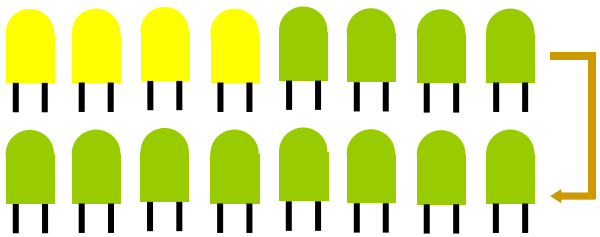
Example 6



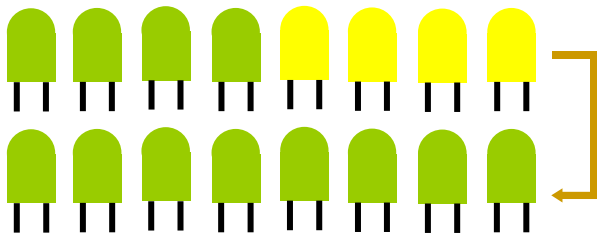
When SW1 Closed



When SW2 Closed



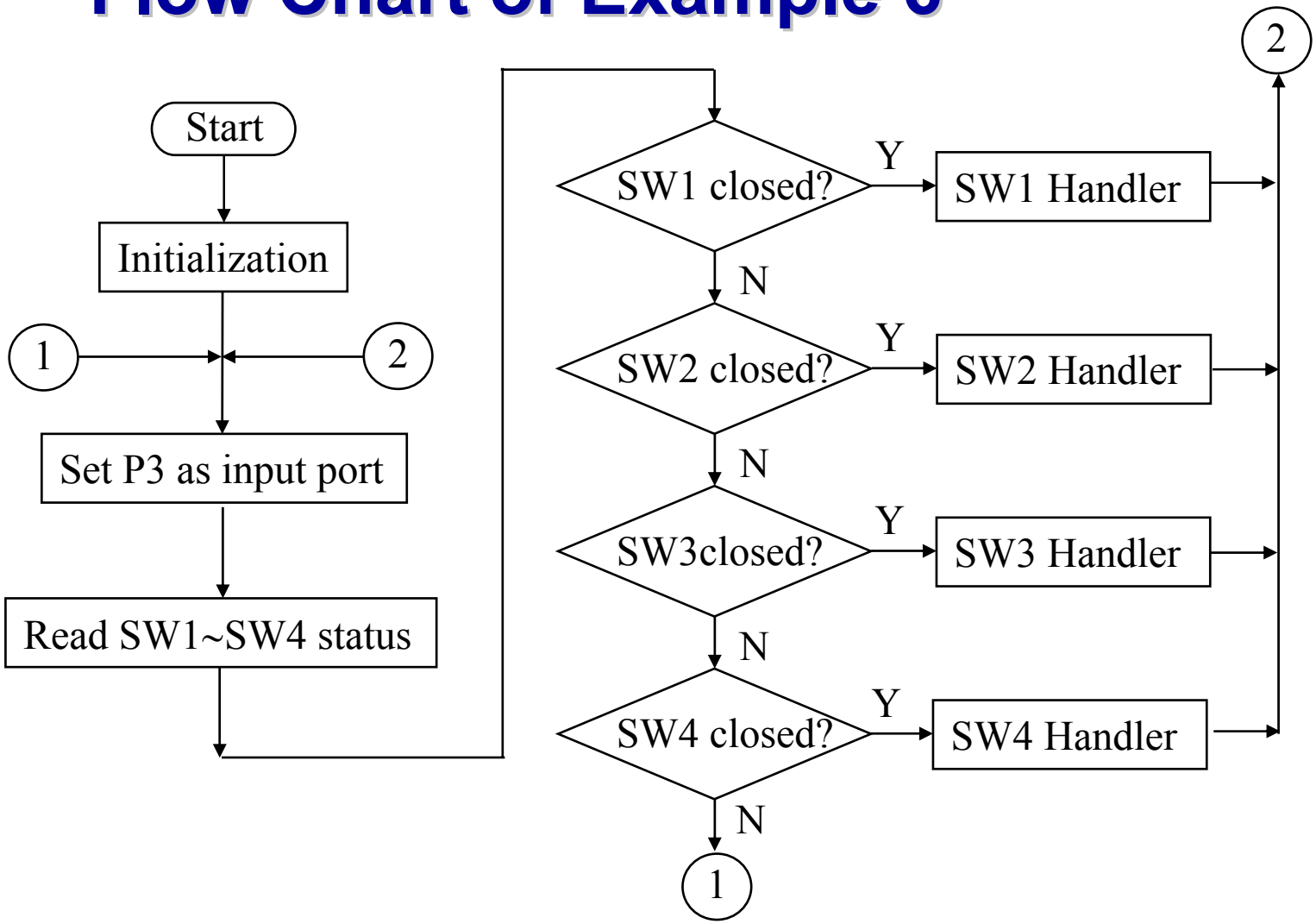
When SW3 Closed



When SW4 Closed

Priority: SW1 → SW2 → SW3 → SW4

Flow Chart of Example 6

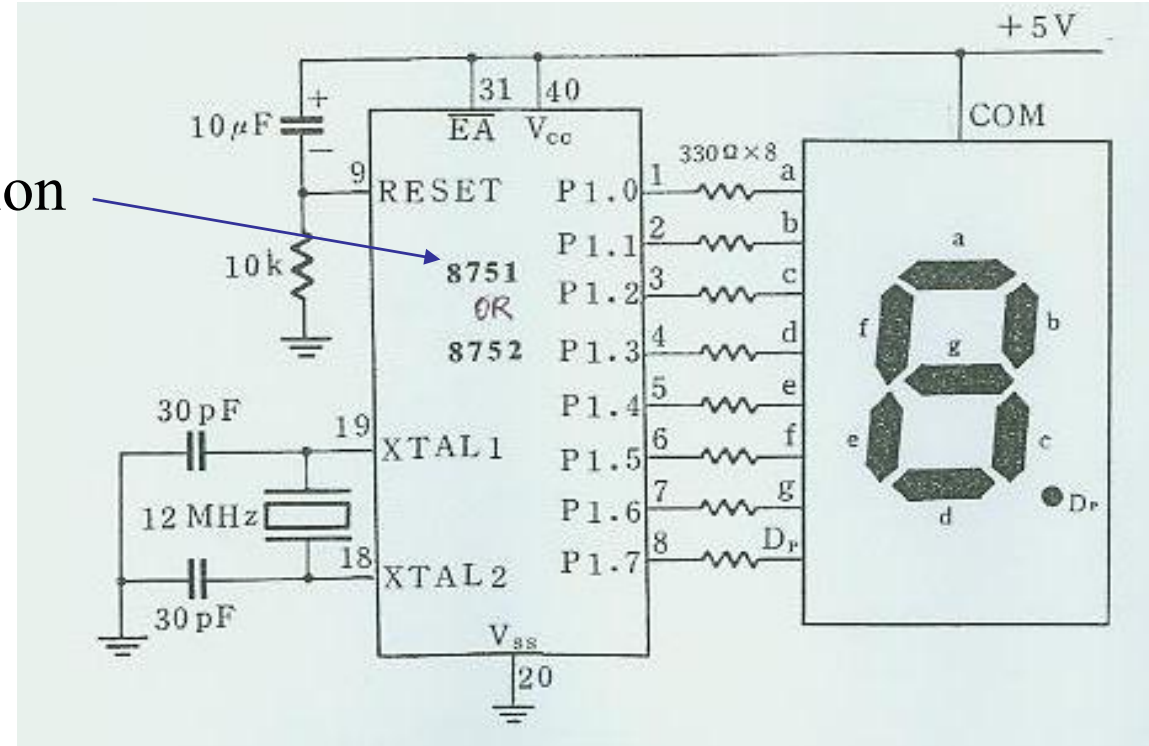


Program Listing of Example 6

```
                ORG         0000H
                MOV         R1, #00000000B
                MOV         R2, #01010101B
                MOV         R3, #00001111B
                MOV         R4, #11110000B
                ;
TEST:           ORL         P3, #0FFH
 ; P3 is configured as input port!
                JNB         P3.7, CASE1
                JNB         P3.6, CASE2
                JNB         P3.5, CASE3
                JNB         P3.4, CASE4
                AJMP        TEST
                ;
CASE1:          MOV         A, R1
                MOV         P1, A
                ACALL       DELAY
                XRL         A, #11111111B
                MOV         P1, A
                AJMP        TEST
                ;
CASE2:          MOV         A, R2
                MOV         P1, A
                ACALL       DELAY
                XRL         A, #10101010B
                MOV         P1, A
                AJMP        TEST
CASE3:          MOV         A, R3
                MOV         P1, A
                ACALL       DELAY
                XRL         A, #11110000B
                MOV         P1, A
                AJMP        TEST
CASE4:          MOV         A, R4
                MOV         P1, A
                ACALL       DELAY
                XRL         A, #00001111B
                MOV         P1, A
                AJMP        TEST
DELAY:          ……..
;inside DELAY, don't modify A, R1, R2, R3 & R4.
                RET
                END
```

Ex7: 7-Segment LED Numeric Display

Eprom Version
Of 8051



R3 is used as a counter, write a 8051 assembly language program using look-up table method, to display the value in R3 to a 7-segment display

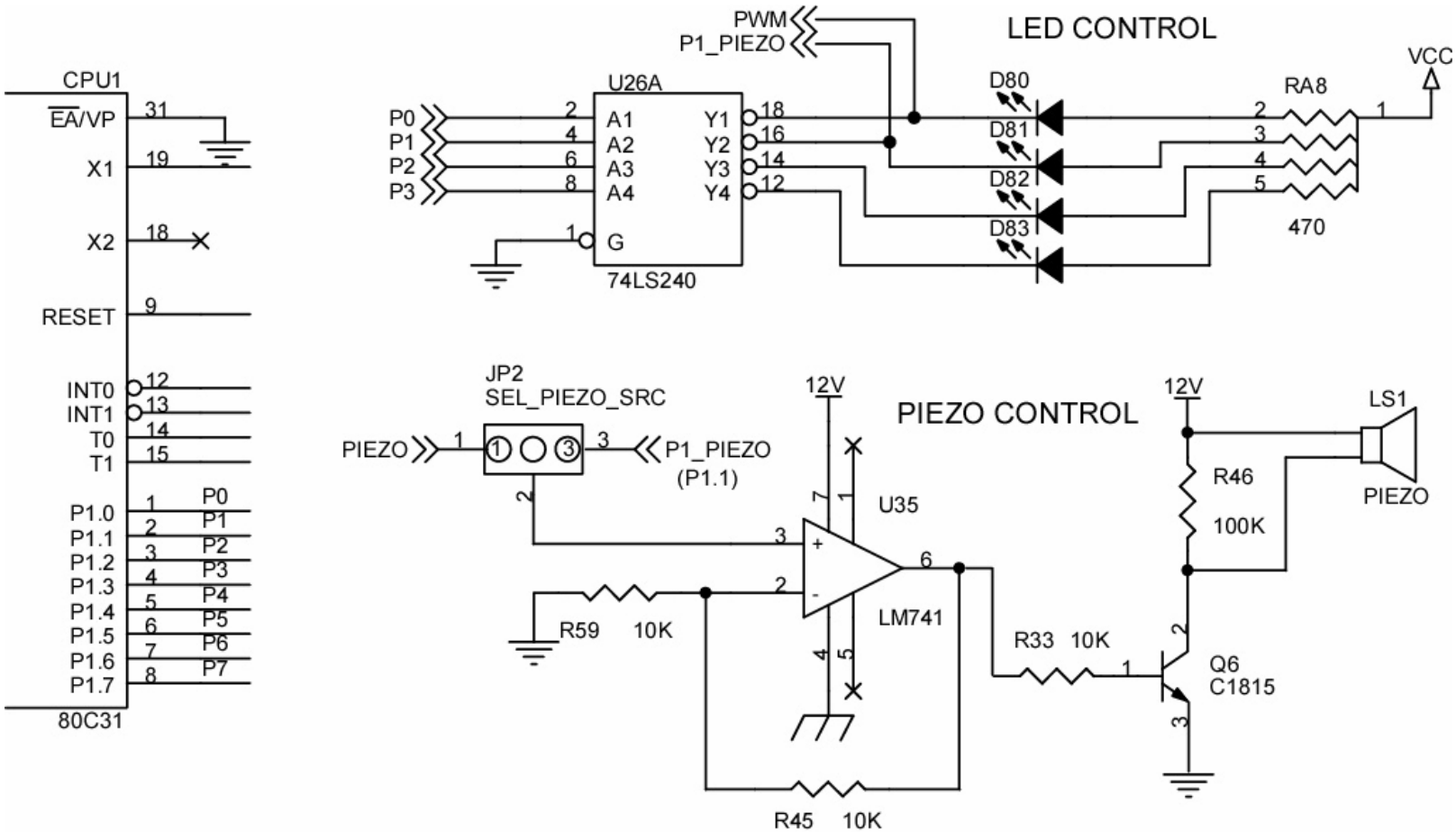
Program Listing of Example 7

```
                ORG         0000H
                MOV         R3, #00H
LOOP:           MOV         DPTR, #TABLE
                MOV         A, R3
                MOVC        A, @A+DPTR
;
; Display numbers on 7-segment display
                MOV         P1, A
                ACALL       DELAY
;
; Increase R3 by 1 and loop back
                MOV         A, R3
                ADD         A, #1
                DA          A
                ANL         A, #0FH
; take the lower nibble only
; now A has value between 0 to 9
                MOV         R3, A     ;update R3
                AJMP        LOOP
                ;
DELAY:          ……..
                RET
```

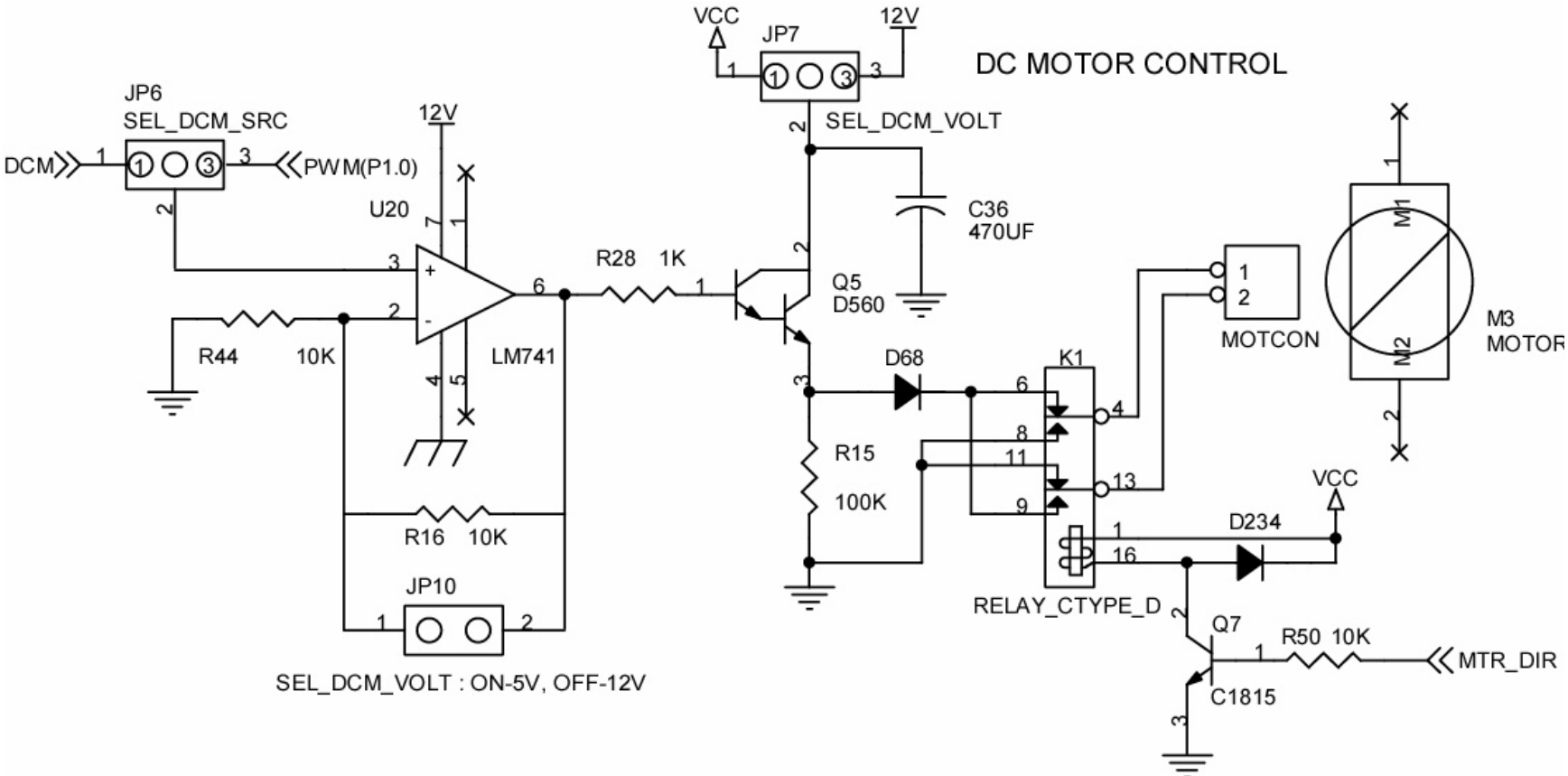
```
TABLE:          DB          11000000B   ; 0
                DB          11111001B   ; 1
                DB          10100100B   ; 2
                DB          10110000B   ; 3
                DB          10011001B   ; 4
                DB          10010010B   ; 5
                DB          10000010B   ; 6
                DB          11111000B   ; 7
                DB          10000000B   ; 8
                DB          10010000B   ; 9
                .
                ;
                END
```

Một số thí dụ các mạch giao tiếp IO

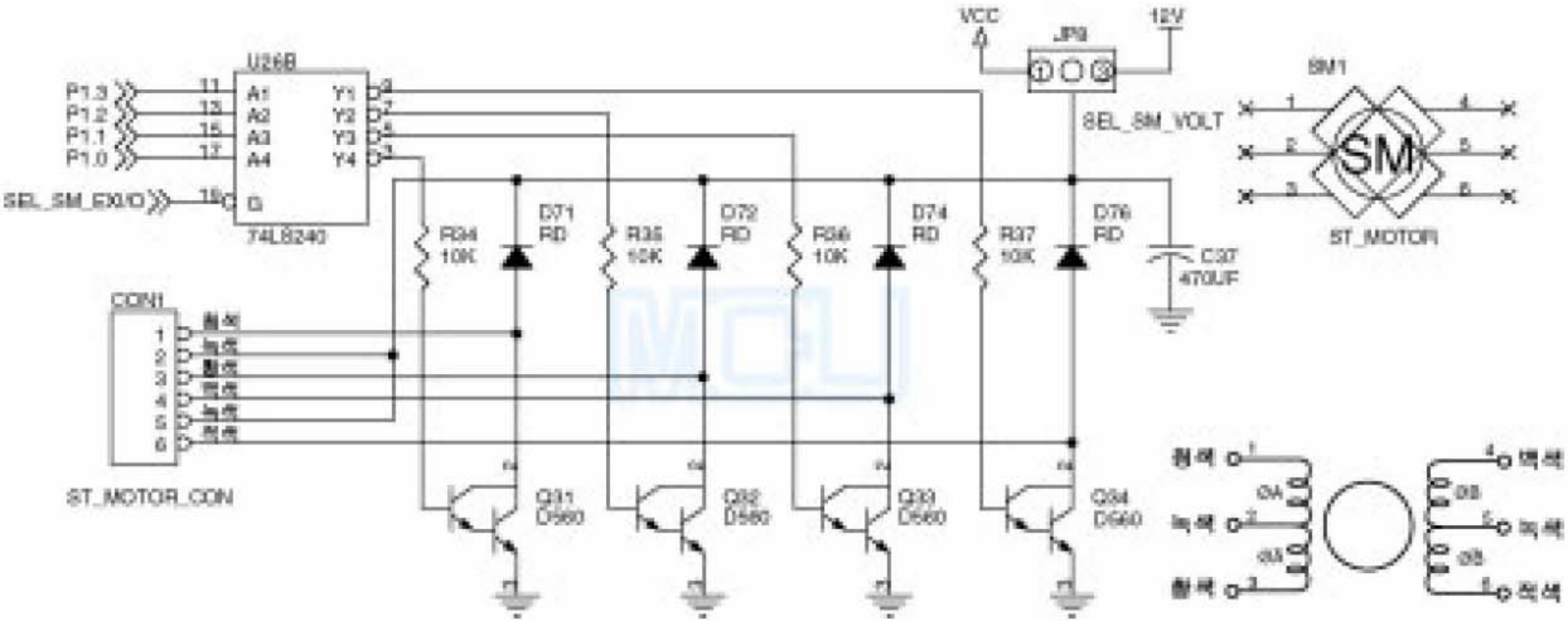
8031/8051 với LED và loa



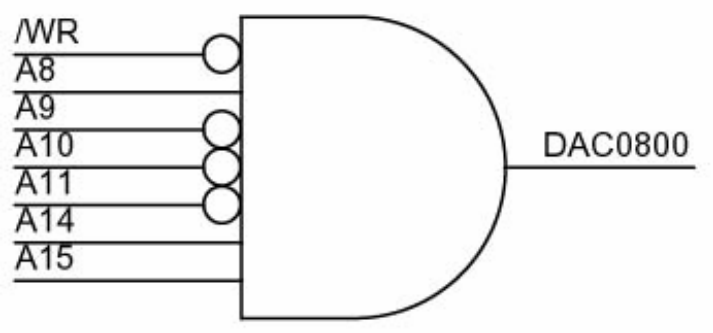
Giao tiếp với động cơ DC

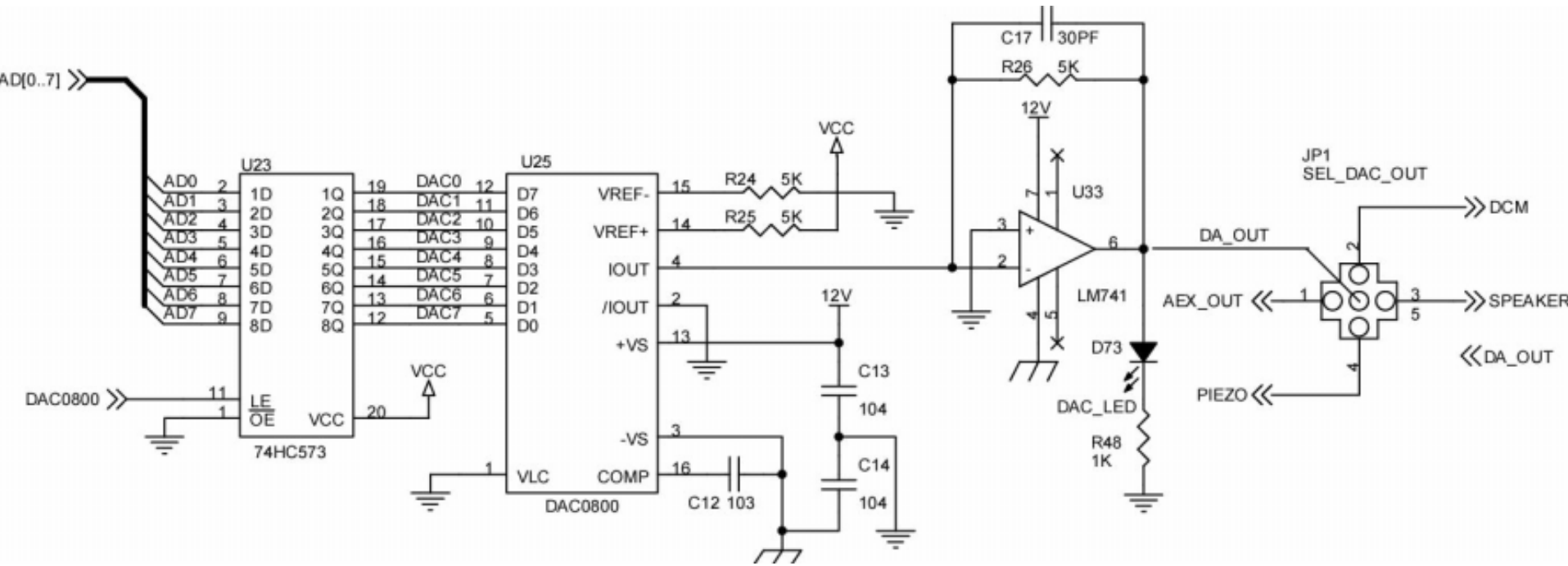


Giao tiếp với động cơ bước

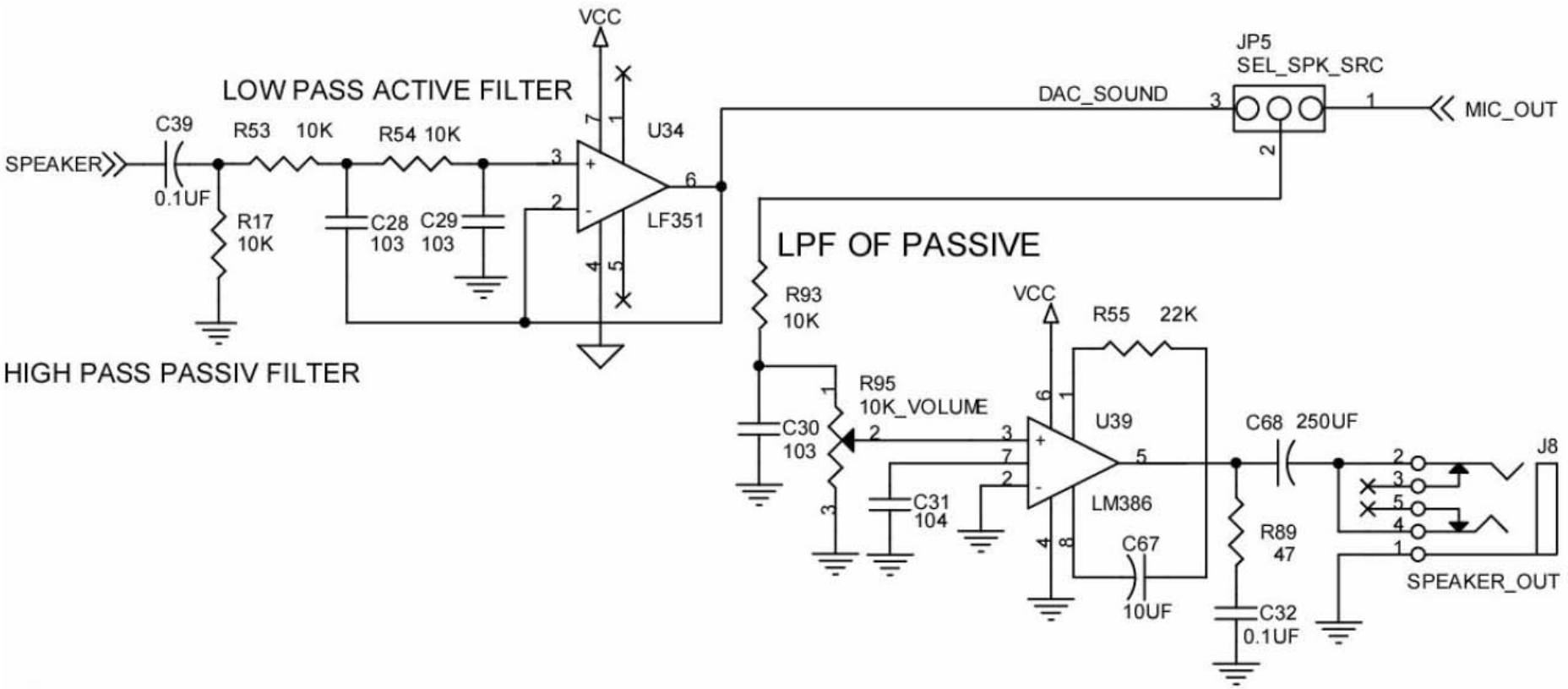


Giao tiếp với DAC





Mạch loa đi kèm với DAC



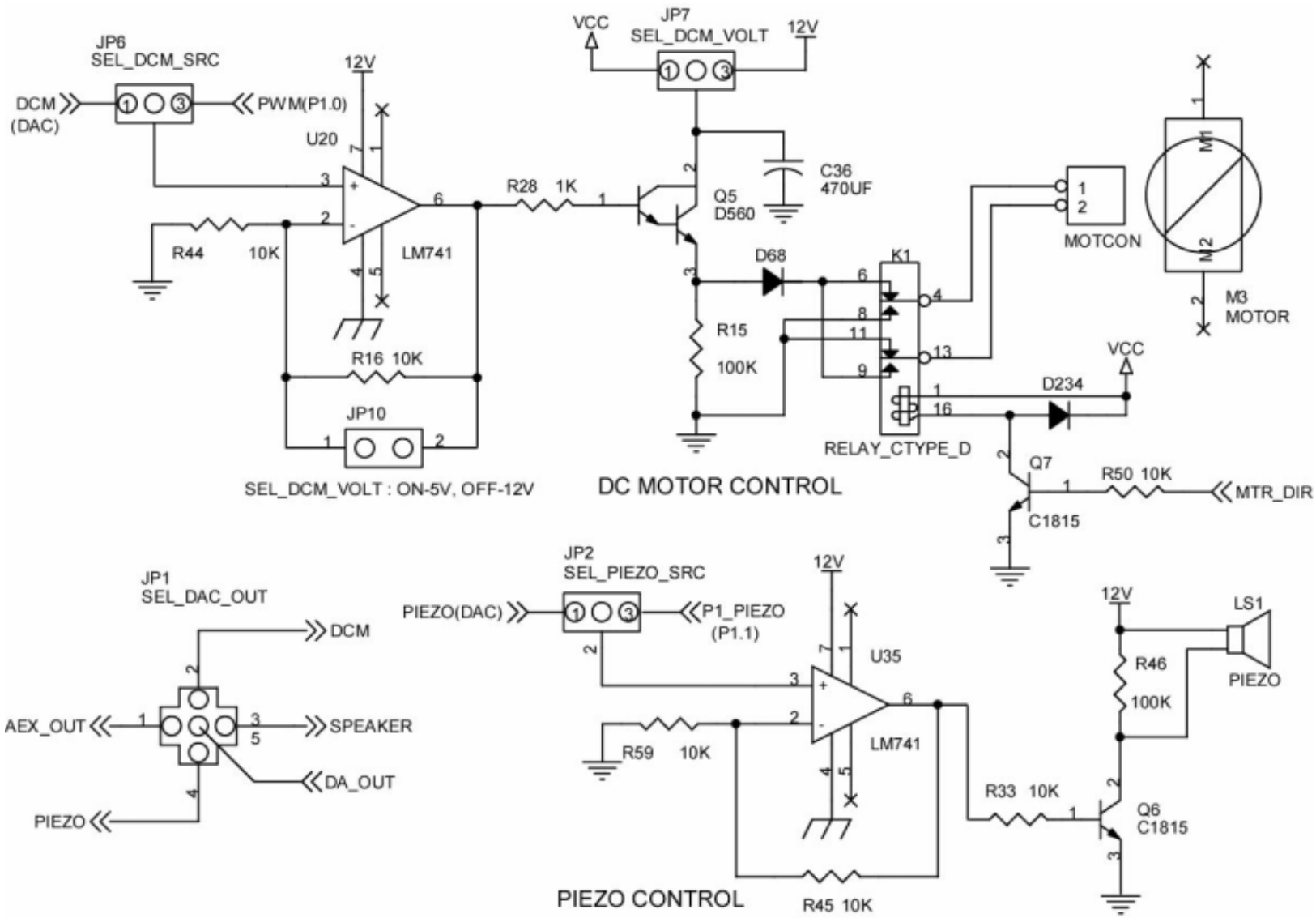
- **Low Pass Filter**

  $fc = 1/(2\pi RC) = 1/(2\pi \times 10 \times 10 \times 0.01 \times 10^{-6}) = 1592[Hz]$

- **High Pass Filter**

  $fc = 1/(2\pi RC) = 1/(2\pi \times 10 \times 10 \times 0.1 \times 10^{-6}) = 159.2[Hz]$

Mạch điều khiển động cơ DC đi kèm DAC



Giao tiếp với ADC