Tutorial 01.03

Gửi đến: Đoàn Hiệp, Doãn Minh Đăng, picvietnam@googlegroups.com

Nội dung: BÀI 1: PIC16F877A TỪ DỄ TỚI KHÓ

MICROSOFT WORD

Tóm tắt:

Tutorial post lên luồng "PIC16F877A TỪ DỄ TỚI KHÓ" thuộc chuyên mục "CƠ BẢN VỀ VI ĐIỀU KHIỂN VÀ PIC". Bài đầu tiên bao gồm nội dung sau:

Ứng dụng đơn giản nhất dành cho vi điều khiển PIC16F877A, đó là xuất dữ liệu ra một port nào đó của vi điều khiển. Các bước tiến hành bao gồm:

_ Bước 1: Xây dựng mạch test.

_ Bước 2: Xây dựng chương trình.

_ Bước 3: Nhận xét và kết luận.

Một số đặc điểm về các port điều khiển của vi điều khiển PIC16F877A.

Chương trình và sơ đồ nguyên lí mạch test đi kèm.

1. Điều khiển các port I/O

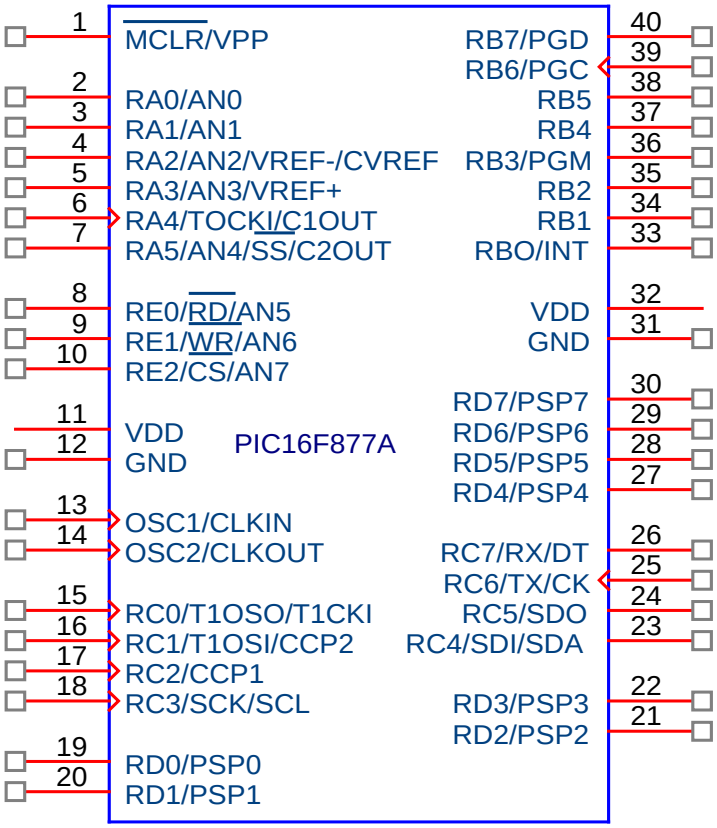
Đây là một trong những ứng dụng đơn giản nhất giúp ta làm quen với vi điều khiển. Trong ứng dụng này ta sẽ xuất một giá trị nào đó ra một PORT của vi điều khiển, chẳng hạn như PORTB. Giá trị đưa ra PORTB sẽ được kiểm tra bằng cách gắn các LED vào các chân I/O của PORT đó.

1.1. Xây dựng mạch test cho ứng dụng

Trước tiên ta cùng xây dựng mạch test cho ứng dụng này. Ngoại trừ vi điều khiển PIC16F877A, các thành phần còn lại trong mạch đều rất thông dụng và dễ dàng tìm thấy trên thị trường, do đó hãy thi công mạch test này để ta có thể xem xét các hiệu ứng cụ thể của vi điều khiển một cách trực quan và nghiêm túc, vì sau bài này, các bạn sẽ thấy rằng ta không thể ngồi một chỗ đọc sách hay tài liệu mà có thể lường trước được hết những hiệu ứng mà vi điều khiển tạo ra, thậm chí là các hiệu ứng từ ứng dụng đơn giản nhất này.

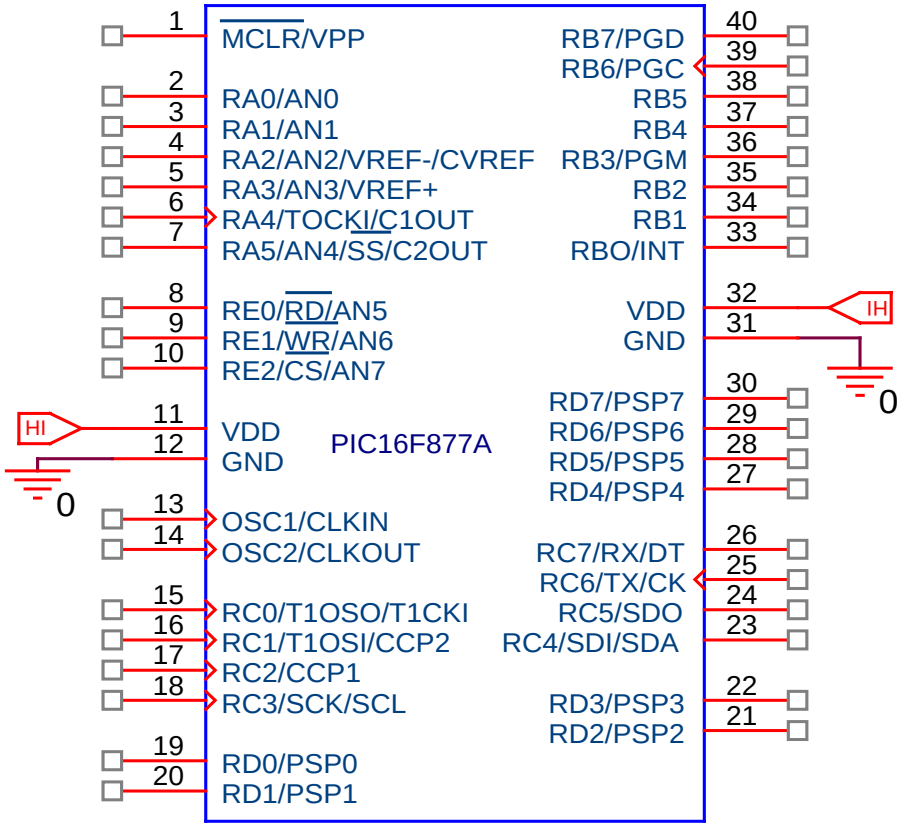
Do đây là bài đầu tiên, cho nên các bước tiến hành sẽ rất nghiêm túc và thận trọng. Nào, bắt đầu!

Ta có vi điều khiển PIC16F877A với sơ đồ chân như sau:



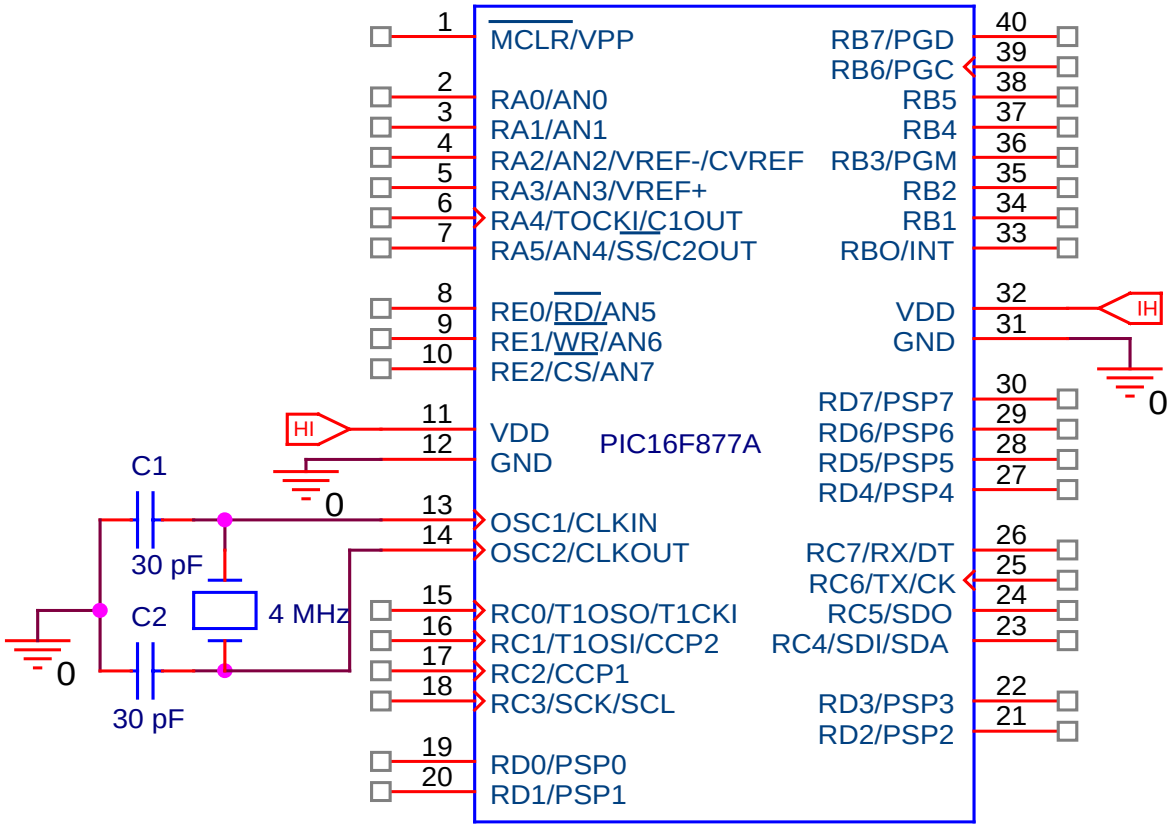
Hình 1.1 Vi điều khiển PIC16F877A.

Bây giờ ta hãy cấp nguồn cho vi điều khiển hoạt động, và câu hỏi đặt ra là cấp nguồn như thế nào? Tất nhiên, nguồn cung cấp sẽ là nguồn 5V, vấn đề ở đây là, vi điều khiển PIC16F877A có đến hai chân cấp nguồn V_{CC} và hai chân GND. Các bạn có cảm thấy bối rối và thắc mắc là tại sao lại có đến 4 chân cấp nguồn như vậy không? Và sau đây là câu trả lời, ta phải cấp nguồn vào tất cả các chân nguồn trên, như vậy thì vi điều khiển mới hoạt động được. Và mạch nguyên lí sau khi cấp nguồn như sau:



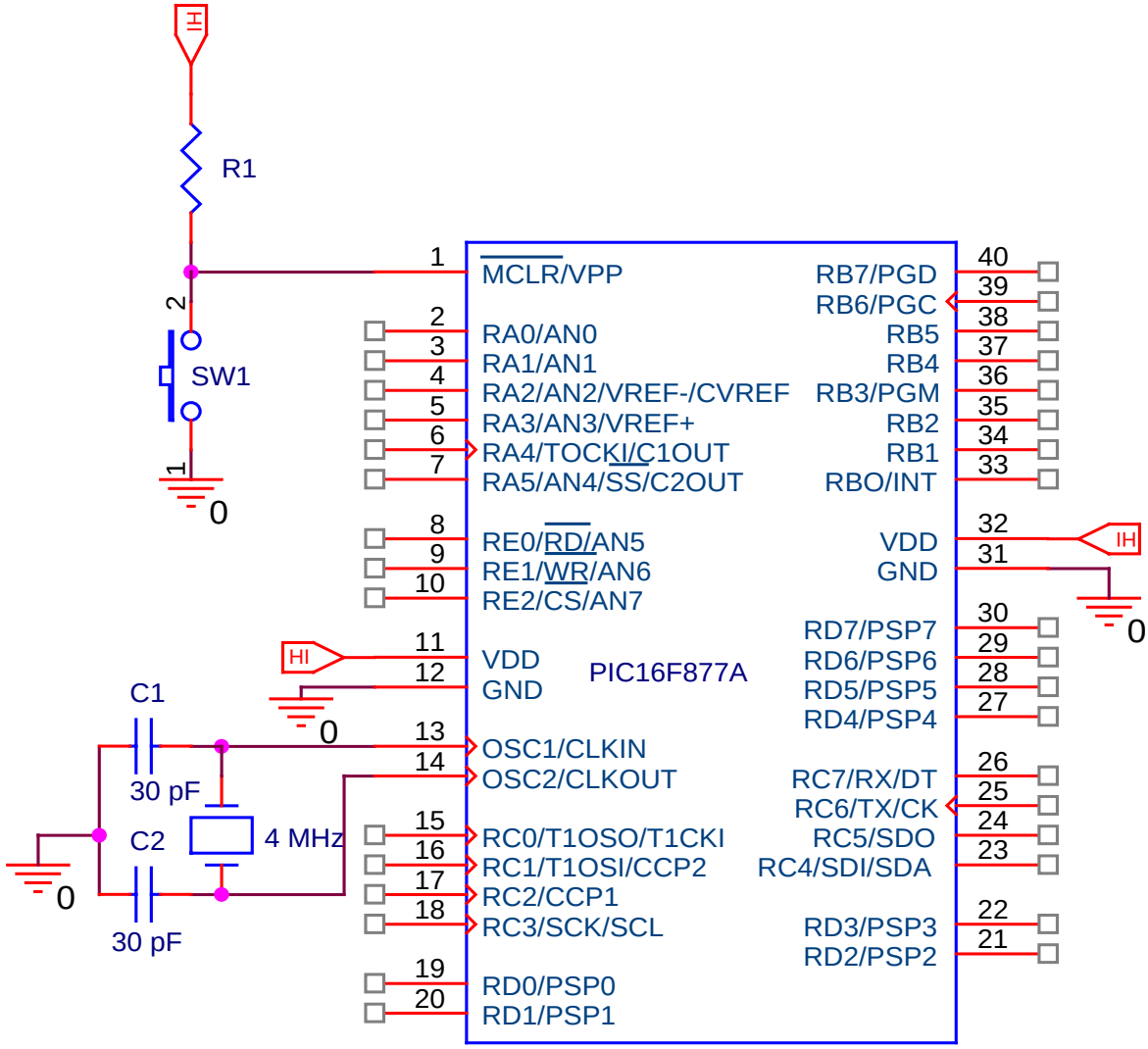
Hình 1.2 Vi điều khiển PIC16F877A sau khi cấp nguồn.

Tiếp theo, ngoài nguồn cung cấp, ta cần phải cung cấp xung hoạt động cho vi điều khiển. PIC16F877A và các vi điều khiển nói chung cho phép nhiều cách cung cấp xung hoạt động khác nhau. Ở đây ta sẽ dùng thạch anh làm nguồn xung, và công việc của ta là gắn thạch anh vào hai chân 13 và 14 của vi điều khiển. Tuy nhiên các bạn cũng biết rằng, các xung dao động do thạch anh tạo ra cũng không thực sự ổn định một cách tuyệt đối, và cách khắc phục là gắn thêm các tụ lọc vào thạch anh. Như vậy, cần phải gắn các tụ như thế nào và giá trị bao nhiêu? Câu trả lời nằm trong cái datasheet. Các bạn lật cái datasheet PIC16F87xA do Microchip cung cấp ra. Trang 145, hình 4.1 hướng dẫn cách gắn các tụ C1, C2 vào thạch anh, và trang 146, bảng 14-2 hướng dẫn cách chọn giá trị cho tụ. Ỡ đây la dùng thạch anh 4 MHz nên tụ C1 và C2 sẽ có giá trị 15 pF. Một điểm đáng chú ý nữa là chất lượng thạch anh tại thị trường Việt Nam không thực sự tốt, cho nên để tăng sự ổn định, ta sẽ dùng tụ 30 pF. Xong! Và sau đây là mạch nguyên lí sau khi gắn thêm thạch anh:



Hình 1.3 PIC16F877A sau khi cấp nguồn và thêm thạch anh.

Bây giờ là mạch reset cho vi điều khiển sử dụng chế độ reset từ chân MCLR của vi điều khiển (chân số 1). Ta đã biết vi điều khiển sẽ được reset khi chân MCLR chuyển từ mức logic 1 xuống mức logic 0 và ta sử dụng một công tắc cơ khí để thực hiện viêc chuyển đổi đó, như vậy ta mới có thể tác động cho vi điều khiển reset bằng tay. Lại một câu hỏi nữa, phải thiết kế mạch như thế nào để thực hiện được công việc đó? Dễ thôi, ta có thể thiết kế như hình 1.4. Bình thường công tắc hở, chân MCLR của vi điều khiển mang mức logic 1 (vì được nối với nguồn qua điện trở hạn dòng R1). Điện trở R1 phải có giá trị nhỏ hơn 40K để bảo đảm điện áp cung cấp cho vi điều khiển. Khi ấn công tắc, chân MCLR được nối với GND nên mang mức logic 0, khi đó vi điều khiển sẽ được reset.



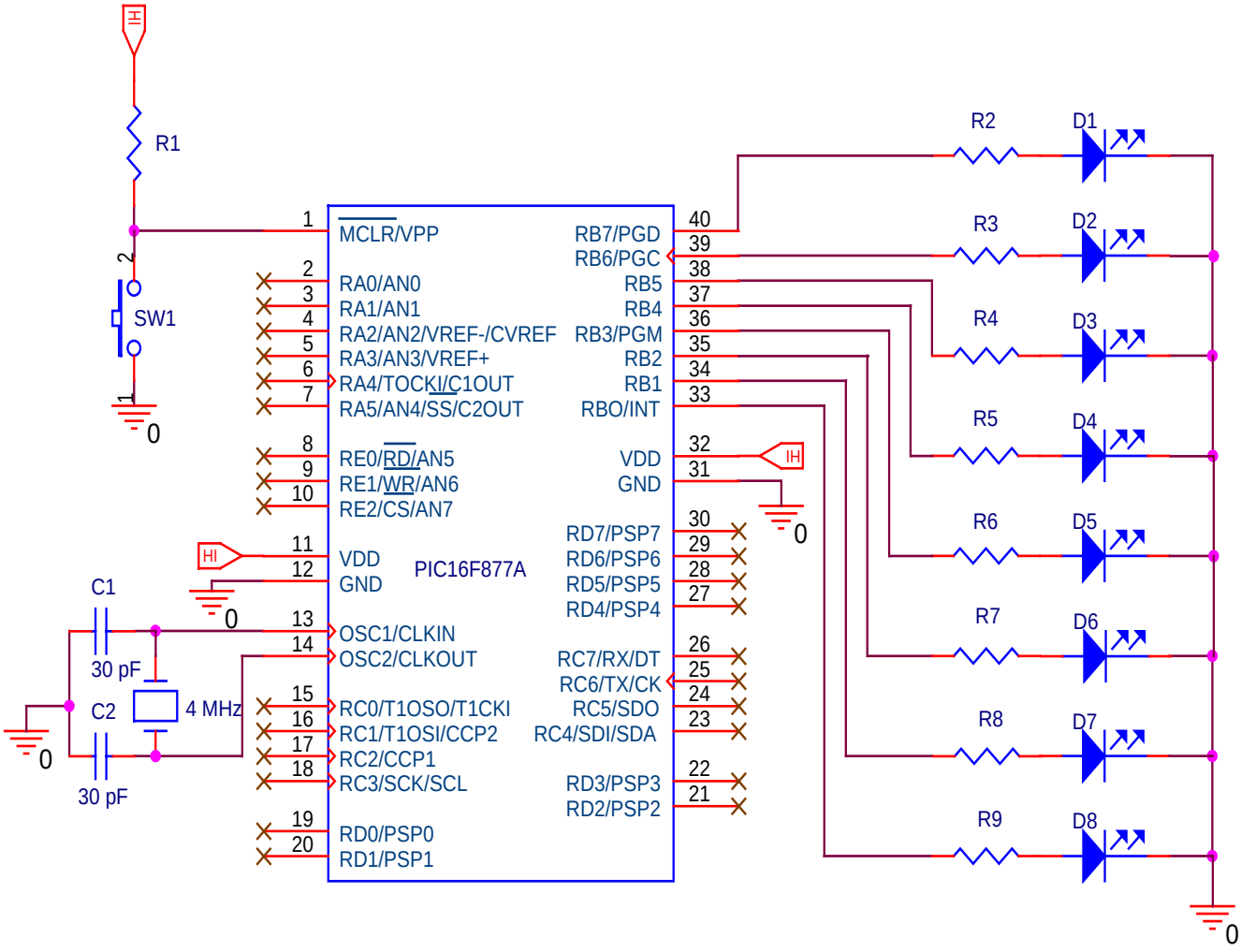
Hình 1.4 PIC16F877A sau khi tiếp tục thêm vào mạch seset.

Đến đây xem như ta đã hoàn thành những thành phần cơ bản nhất cho một mạch test dành cho vi điều khiển PIC16F877A. Ta tiếp tục phát triển các thành phần tiếp theo để test các port của vi điều khiển.

PORTB của vi điều khiển sẽ được test đầu tiên. Mục đích của mạch test là kiểm tra xem các giá trị ta xuất ra port bằng chương trình có đúng hay không, và để phát hiện được các giá trị đó một cách trực quan, ta sử dụng 8 LED gắn vào 8 chân trong PORTB của vi điều khiển. Khi ta xuất giá trị mang mức logic 1 ra môt chân nào đó trong PORTB của vi điều khiển, LED tương ứng gắn với chân đó sẽ sáng lên (do lúc này điện áp ở chân của vi điều khiển là 5V) và ngược lại, nếu giá trị xuất ra mang mức logic 0 thì LED sẽ không sáng (do lúc này điện áp ở chân của vi điều khiển là 0V). Tuy nhiên, ta cần **chú ý** đến một điểm quan trọng nữa, đó là để LED sáng bình thường, điện áp đặt vào hai đầu của LED vào khoảng 1,8V đến 2,2V, trong khi điện áp tại chân I/O của vi điều khiển khi ta xuất ra mức logic 1 sẽ

là 5V. Do đó cần mắc thêm điện trở hạn dòng cho LED (có thể dùng điện trở có giá trị 0.33 K).

Dựa vào các điểm đã phân tích ở trên ta có thể xây dựng được mạch nguyên lí hoàn chỉnh cho ứng dụng test PORTB như sau:



Hình 1.5 Mạch nguyên lí hoàn chỉnh cho ứng dụng test PORTB.

Như vậy đến đây ta đã hoàn tất việc thiết kế phần cứng cho ứng dụng. Trong phần tiếp theo ta tiếp tục bàn đến việc viết chương trình cho ứng dụng trên.

1.2. Xây dựng chương trình xuất dữ liệu ra PORTB vi điều khiển PIC16F877A

Trước tiên, để viết được chương trình, ta cần tìm hiểu một số đặc điểm về cấu trúc của vi điều khiển PIC16F877A và cú pháp của một số lệnh sử dụng trong chương trình.

1.2.1 Một số đặc điểm về cấu trúc PORTB vi điều khiển PIC16F877A.

Ta cần chú ý đến các điểm sau:

- PORTB của vi điều khiển PIC16F877A cũng như các port điều khiển khác đều cho phép truyền nhận dữ liệu theo hai hướng, có nghĩa là ta được phép đọc và xuất dữ liệu ra port điều khiển. Hướng truyền nhận được thiết lập bằng cách đưa giá trị thích hợp vào **thanh ghi TRISB**. Mỗi bit trong thanh ghi điều khiển hướng xuất/nhập cho một chân của port (bit 7 của thanh ghi TRISB điều khiển chân RB7, bit 6 của thanh ghi TRISB điều khiển chân RB6, ...). Nếu một bit trong thanh ghi TRISB mang **mức logic 0** thì vi điều khiển sẽ hiểu rằng chân điều khiển bởi bit đó là chân **xuất dữ liệu** và ngược lại, nếu một bit trong thanh ghi TRISB mang **mức logic 1** thì vi điều khiển sẽ hiểu rằng chân điều khiển bởi bit đó là chân **nhập dữ liệu**. Ví dụ, ta muốn thiết lập chân **RB3, RB2, RB1, RB0** của PORTB là **nhập**, chân **PB7, RB6, RB5, RB4** của PORTB là **xuất**, khi đó giá trị tương ứng đưa vào thanh ghi TRISB sẽ là **'00001111'**.
- Dữ liệu nhập vào hay xuất ra PORTB sẽ được chứa trong **thanh ghi PORTB**. Ví dụ, giả sử như tất cả các chân của PORTB đều là chân xuất dữ liệu, khi đó muốn đưa tất cả các chân của PORTB lên mức logic 1, ta chỉ việc đưa vào thanh ghi PORTB giá trị '11111111'. Nếu tất cả các chân trong PORTB đều là chân nhập dữ liệu, muốn biết được trạng thái mức logic của từng chân ta chỉ việc đọc giá trị của thanh ghi PORTB.
- Trong cấu trúc bộ nhớ dữ liệu của PIC16F877A, **thanh ghi PORTB** nằm ở **BANK 0**, còn **thanh ghi TRISB** nằm ở **BANK 1**. Ta đã biết muốn truy xuất giá trị của một thanh ghi nào đó trong bộ nhớ dữ liệu của vi điều khiển PIC, trước tiên cần chọn BANK dữ liệu chứa thanh ghi đó, và việc chọn BANK dữ liệu được điều khiển bởi hai bit **RP1:RP0** của **thanh ghi STATUS**. Cụ thể như sau:

  RP1:RP0 = 00 chọn BANK 0.

  RP1:RP0 = 01 chọn BANK 1.

  RP1:RP0 = 10 chọn BANK 2.

  RP1:RP0 = 11 chọn BANK 3.

Các đặc điểm này sẽ là cơ sở cho việc hình thành chương trình xuất dữ liệu ra PORTB của vi điều khiển PIC16F877A.

1.2.2 Các lệnh sử dụng cho chương trình

Phần này sẽ đề cập đến các lệnh sử dụng trong chương trình xuất dữ liệu ra PORTB của vi điều khiển PIC16F877A. Ta cần sử dụng các lệnh sau:

- **Lệnh BSF**

  Cú pháp: BSF thanhghi,bit

  (tham số "bit" mang giá trị từ 0 đến 7).

  Chức năng: lệnh này dùng để đưa bit có số thứ tự chứa trong tham số "bit" của thanh ghi chứa trong tham số "thanhghi" lên mức logic 1.

  Ví dụ: BSF PORTB,7

  (bit 7 của thanh ghi PORTB sau lệnh này sẽ mang mức logic 1).

- **Lệnh BCF**

  Cú pháp: BCF thanhghi,bit

  (tham số bit mang giá trị từ 0 đến 7)

  Chức năng: lệnh này dùng để đưa bit có số thứ tự chứa trong tham số "bit" của thanh ghi chứa trong tham số "thanhghi" về mức logic 0.

  Ví dụ: BCF PORTB,7

  (bit 7 của thanh ghi PORTB sau lệnh này sẽ mang mức logic 0).

- **Lệnh MOVLW**

  Cú pháp: MOVLW hangso

  (tham số "hangso" mang giá trị từ 0 đến 255)

  Chức năng: đưa giá trị của tham số "hangso" vào thanh ghi W. Ta có một số định dạng về tham số "hangso" như sau:

  Định đạng số hex: thêm kí tự "0x" vào trước tham số "hangso".

  Ví dụ: MOVLW 0x5F

  (đưa giá trị hex 5F vào thanh ghi W).

  Định dang số thập phân: thêm kí tự "d" vào trước tham số "hangso".

  Ví dụ: MOVLW d'15'

  (đưa giá trị thập phân 15 vào thanh ghi W).

  Định dạng số nhị phân: thêm kí tự "b" trước tham số "hangso"

  Ví dụ: MOVLW b'10101010'

  (đưa giá trị nhị phân 10101010 vào thanh ghi W).

- **Lệnh MOVWF**

  Cú pháp: MOVWF thanhghi

  Tác dụng: đưa giá trị chứa trong thanh ghi W vào thanh ghi được chỉ định bởi tham số "thanhghi".

  Ví dụ: MOVWF PORTB

  (đưa giá trị chứa trong thanh ghi W vào thanh ghi PORTB).

- **Lệnh CLRF**

  Cú pháp: CLRF thanhghi

  Tác dung: xóa thanh ghi được chỉ định bởi tham số "thanhghi".

  Ví dụ: CLRF PORTB

  (xóa thanh ghi PORTB).

- **Lệnh GOTO**

  Cú pháp: GOTO label

  Tác dụng: nhảy tới label được chỉ định bởi tham số "label".

  Ví dụ: GOTO next

  (nhảy tới label "next").

Ta thấy rằng trong tập lệnh của vi điều khiển PIC, không có lệnh nào cho phép đưa trực tiếp một giá trị nào đó vào một thanh ghi mà phải thông qua thanh ghi trung gian là thanh ghi W. Ví dụ, ta muốn đưa giá trị b'00000000' vào thanh ghi TRISB (thao tác này có nghĩa là thiết lập tất cả các chân của PORTB là chân xuất dữ liệu), ta có thể dùng hai lệnh như sau:

```
MOVLW   b'00000000'
MOVWF   TRISB
```

Tương tự ta có thể dùng "cặp lệnh" trên để đưa một giá trị 8 bit bất kì vào một thanh ghi bất kì trong bộ nhớ dữ liệu.

Tuy nhiên đây là một trường hợp đặc biệt. Việc đưa vào thanh ghi TRISB giá trị b'00000000' cũng đồng nghĩa với việc xóa thanh ghi TRISB, do đó ta có thể thay thế hai lệnh trên băng một lệnh duy nhất:

```
CLRF    TRISB
```

Đến đây ta đã có được các thông tin cần thiết cho việc viết chương trình điều khiển. Chương trình cụ thể được trình bày ở phần tiếp theo.

1.2.3 Chương trình test PORTB vi điều khiển PIC16F877A.

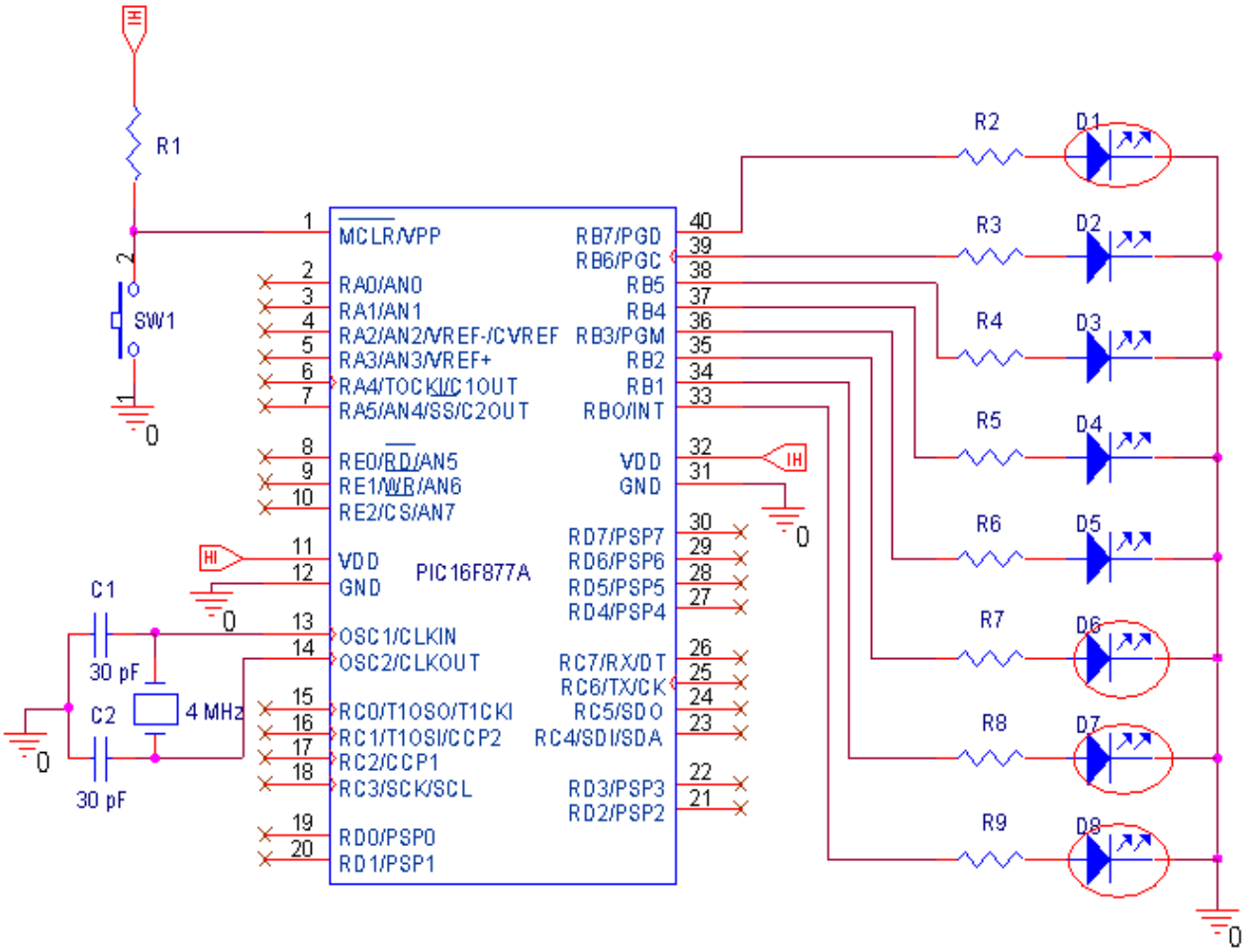
```
;-------------------------------------------------------------------------------
;       Ghi chú về chương trình
;-------------------------------------------------------------------------------
;       Chương trình 1.1
;       PORTBTEST.ASM
;       Chương trinh dùng để test PORTB của vi điều khiển PIC16F877A
;-------------------------------------------------------------------------------
;       Phần khai báo vi điều khiển
;-------------------------------------------------------------------------------
processor   16f877a             ; khai báo vi điều khiển sử dụng chương
                                ; trình này
include     <p16f877a.inc>      ; header file đính kèm
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON &
_XT_OSC & _WRT_OFF & _LVP_ON & _CPD_OFF
                                ; khai báo “configuration bit”
;-------------------------------------------------------------------------------
;       Chương trình bắt đầu tại đây
;-------------------------------------------------------------------------------
        ORG     0x000           ; địa chỉ bắt đầu chương trình
        GOTO    start
start
        BCF     STATUS,RP1
        BSF     STATUS,RP0      ; chọn BANK 1

        CLRF    TRISB           ; khởi tạo PORTB
                                ; PORTB là cổng xuất dữ liệu
        BCF     STATUS,RP0      ; chọn BANK 0
```

```
        MOVLW   b'10001111'     ; giá trị cần xuất ra PORTB
        MOVWF   PORTB           ; PORTB <- 8Fh
loop
        GOTO    loop            ; vòng lặp vô hạn để  dừng chương trình
        END                     ; kết thúc chương trình
```

1.3. Một số nhận định

Như vậy ta đã hoàn tất một ứng dụng dành cho PORTB của vi điều khiển PIC16F877A. Bây giờ hãy kiểm tra kết quả thực hiện của chương trình bằng cách sử dụng mạch test ta vừa thiết kế. Theo tính toán trên lí thuyết, do ta xuất ra PORTB giá trị b'10001111' nên các **LED** gắn vào các chân **RB7, RB3, RB2, RB1, RB0** sẽ **sáng**, còn các LED gắn vào các chân **RB6, RB5, RB4** sẽ **tắt**. Còn kết quả do vi điều khiển PIC tạo ra thì sao? Đây, các LED được khoanh tròn là các LED mà vi điều khiển PIC16F877A làm cho sáng lên:



Hình 1.6 Kết quả thự thi chương trình 1.1 của PIC16F877A.

Ta thấy có một điểm bất hợp lí ở đây. Đó là **LED** gắn vào **chân RB3** của vi điều khiển lại **không sáng**. Lí do tại sao? Đó là do chân RB3 của PIC16F877A còn có thêm một chức năng là chân nạp chương trình cho vi điều khiển ở chế độ nạp LVP (Low Voltage Programming). Khi chế độ nạp này được kích hoạt, chân RB3 của vi điều khiển sẽ không được hoạt động như chân I/O bình thường, cho nên mặc dù ta có xuất dữ liệu ra chân RB3 thì vi điều khiển vẫn không thể hiện tác động đó ra chân RB3. Vậy làm sao để khắc phục? Rất đơn giản, ta chỉ cần tắt chế độ nạp LVP bằng cách khai báo ... **&_LVP_OFF** .... khi khai báo các "Configuration bits" ở phần khai báo vi điều khiển.

Đến đây hẳn các bạn đã hiểu tại sao ta phải thi công mạch test và kiểm tra kết quả thực thi chương trình của vi điều khiển một cách trực quan, cho dù chương trình ứng dụng có đơn giản như thế nào đi nữa, có như vậy ta mới tìm hiểu sâu hơn được các khía cạnh, các vấn đề của vi điều khiển, đồng thời san bằng khoảng cách giữa lí thuyết và thực tế.

Bây giờ ta thử không xóa hết thanh ghi TRISB mà đưa vào thanh ghi TRISB một giá trị khác, chẳng hạn như giá trị b'10000000'. Các bạn đã biết phải làm thế nào rồi chứ? Chọn BANK 1 của bộ nhớ dữ liệu và sử dụng "cặp lệnh":

```
MOVLW   b'10000000'
MOVWF   TRISB
```

Kết quả thực thi chương trình là **LED** gắn vào **chân RB7** của PORTB **không sáng**. Lí do như sau, ta vừa khởi tạo chân RB7 của PORTB là chân nhập dữ liệu (bit 7 của thanh ghi TRISB mang giá trị logic 1), cho nên mặc dù ta có xuất dữ liệu ra chân RB7 (giá trị xuất ra vẫn là b'10001111') thì chân đó vẫn không xuất được dữ liệu ra bên ngoài, nhiệm vụ của chân RB7 lúc này chỉ là nhập dữ liệu từ bên ngoài. Như vậy tại một thời điểm, một chân I/O của vi điều khiển chỉ có thể thực hiên một trong hai nhiệm vụ xuất dữ liệu (Output) hoặc nhận dữ liệu (Input) tùy theo chức năng mà ta khởi tạo (đưa dữ kiệu thích hợp vào thanh ghi TRISB).

Tương tự ta có thể xuất ra PORTB một giá trị khác bằng cách thay đổi giá trị đưa vào thanh ghi PORTB.

Các port còn lại của vi điều khiển PIC16F877A cũng như các vi điều khiển PIC khác đều có cấu trúc tương tự, tức là có thanh ghi TRISx để điều khiển chức năng (Input hay Output) và thanh ghi PORTx chứa dữ liệu của port đó. Dựa vào đặc điểm này ta có thể viết chương trình điều khiển các port còn lại của PIC16F877A xuất dữ liệu ra bên ngoài theo cấu trúc như chương trình 1.1. Đây là thao tác nên thực hiện để kiểm tra lại các đặc tính của từng port trong vi điều khiển PIC16F877A, đồng thời giúp các bạn làm quen với cấu trúc chương trình cũng như cách viết chương trình dùng cho vi điều khiển PIC.

Sau đây là một vài đặc điểm về các port của vi điều khiển PIC16F877A mà các bạn nên chú ý:

- Chân **RA4** của PORTA là chân có **cực thu để hở,** cho nên khi test PORTA, ta cần gắn thêm điện trở kéo lên cho chân này để đảm bảo kết quả hiển thị ra LED.
- **PORTA và PORTE** mặc định khi khởi động là các chân I/O của tín hiệu analog, cho nên trước khi muốn sử dụng các chân này như các chân I/O bình thường ta cần tiến hành thêm một bước khởi tạo nữa. Tuy nhiên trong bài đầu tiên ta chỉ xuất dữ liệu ra LED nên chưa cần quan tâm đến bước khởi tạo này (kết quả hiển thị vẫn không có gì khác biệt so với tính toán trên lí thuyết), ta sẽ bàn kĩ đến các bước khởi tạo và đặc tính analog của các chân trong PORTA và PORTE trong các bài sau.
- Các chân trong PORTB, PORTC, PORTD cũng có các chức năng khác ngoài chức năng I/O, tuy nhiên khi khởi động các chân này được mặc định là các chân I/O bình thường nên trong quá trình test port ta không còn trở ngại gì nữa.

Đây là bài đầu tiên đơn giản nhất nhưng hẳn các bạn cũng đã nhận thấy rằng có rất nhiêu thông tin cần được xử lí đến nơi đến chốn một cách thực tế và với sự nghiêm túc cần thiết. Việc thực hiện thành công ứng dụng đơn giản nhưng cũng rất quan trọng này có thể xem là một "bước ngoặc" trong quá trình bạn làm quen với vi điều khiển PIC16F877A nói riêng và họ vi điều khiển PIC nói chung.

Bên cạnh đó các bạn cũng nên làm quen dần với cách ghi chú chương trình, công việc này có vẻ thừa thải nhưng tác dụng mà nó mang lại là rất tích cực. Thứ nhất, ta có thể dần định hướng cho mình một cấu trúc chương trình viết cho vi điều khiển PIC. Thứ hai, bạn có thể biết được mình vừa ra lệnh cho vi điều khiển PIC thực hiện công việc gì với lệnh mình vừa viết ra. Thứ ba, người khác khi dọc chương trình của bạn cũng cảm thấy thân thiện và dễ hiểu hơn.

Ứng dụng này phải được thực hiện thành công trước khi ta xây dựng các ứng dụng khác phức tạp hơn. Đây cũng là mục đích xây dựng của loạt bài "PIC16F877A từ dễ tới khó", ta sẽ bắt đầu từ ứng dụng đơn giản nhất này, và lấy nó làm cơ sở để xây dựng các ứng dụng ngày càng phức tạp hơn. Hy vọng các bạn có thể biết được thêm một số thông tin nào đó về vi điều khiển PIC16F877A sau bài đầu tiên này.

Hết bài 1!