

CHƯƠNG 6

HOẠT ĐỘNG NGẮT

(*INTERRUPT*)

I. MỞ ĐẦU:

1 CPU CHỈ THỰC THI ĐƯỢC 1 LỆNH TẠI MỘT THỜI ĐIỂM.

Ngắt (*Interrupt*) là việc xảy ra một điều kiện (*một sự kiện*) làm cho chương trình đang thực thi (*chương trình chính*) bị tạm dừng để quay sang thực thi một chương trình khác (*chương trình xử lý ngắt*) rồi sau đó quay trở về để thực thi tiếp chương trình đang bị tạm dừng. Các ngắt đóng vai trò quan trọng trong việc thiết kế và hiện thực các ứng dụng của bộ vi điều khiển. Các ngắt cho phép hệ thống đáp ứng một sự kiện theo cách không đồng bộ và xử lý sự kiện trong khi một chương trình khác đang thực thi. Một hệ thống được điều khiển bởi ngắt cho ta **ảo tưởng** nhiều công việc đang được vi xử lý thực hiện đồng thời.

CPU dĩ nhiên không thể thực thi nhiều hơn một lệnh ở một thời điểm nhưng CPU có thể tạm ngưng việc thực thi một chương trình để thực thi một chương trình khác rồi sau đó quay về thực thi tiếp tục chương trình đang bị tạm ngưng, điều này thì tương tự như việc CPU rời khỏi chương trình gọi để thực thi chương trình con bị gọi để rồi sau đó quay trở về chương trình gọi.

Cần phải phân biệt sự giống và khác nhau giữa “**ngắt**” và “**gọi chương trình con**”:

- **Giống nhau:**

Khi xảy ra điều kiện tương ứng thì CPU sẽ *tạm dừng* chương trình chính đang thực thi để thực thi một chương trình khác (*chương trình con / chương trình xử lý ngắt*) rồi sau đó (*sau khi xử lý xong chương trình con / chương trình xử lý ngắt*) thì CPU sẽ quay về để thực thi tiếp tục chương trình chính đang bị tạm dừng.

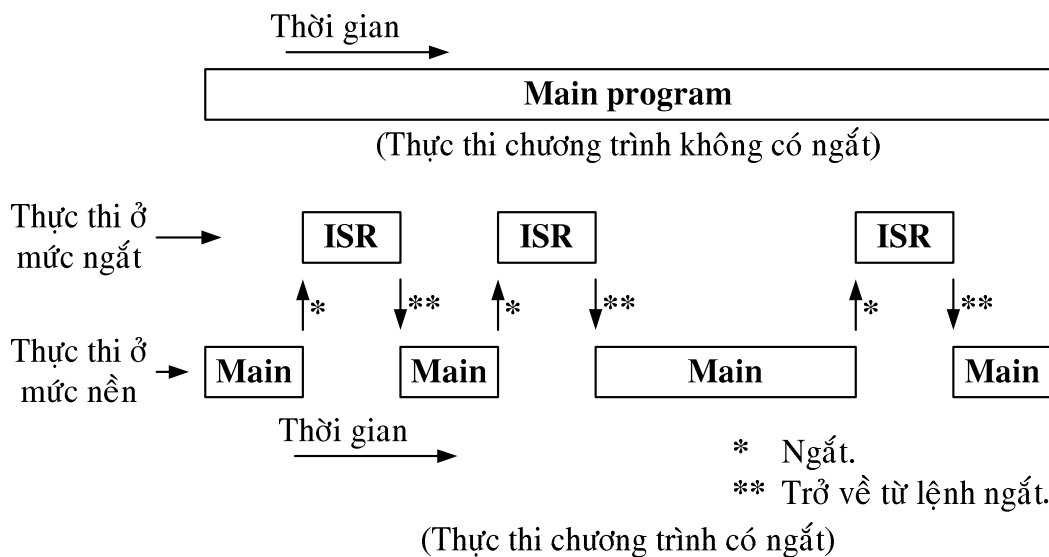
- **Khác nhau:**

	Ngắt	Chương trình con
Thời điểm xảy ra sự kiện	Không biết trước (<i>hay xảy ra không đồng bộ với chương trình chính</i>).	Biết trước (<i>hay xảy ra đồng bộ với chương trình chính</i>).
Nguyên nhân dẫn đến sự kiện	Do các tín hiệu điều khiển từ Timer, Serial port và bên ngoài chip.	Do lệnh gọi chương trình con (<i>ACALL, LCALL</i>).

Chương trình xử lý ngắt (*tức là chương trình mà CPU phải thực hiện khi có một ngắt xảy đến*) được gọi là trình phục vụ ngắt ISR (*ISR: Interrupt Service Routine*) hay trình quản lý ngắt (*Interrupt Handler*). ISR được thực thi nhằm đáp ứng một ngắt và trong trường hợp tổng quát thực hiện việc xuất nhập đối với một thiết bị. Khi một ngắt xuất hiện, việc thực thi chương trình chính tạm thời bị dừng lại và CPU thực thi việc rẽ nhánh đến trình phục vụ ngắt ISR. CPU sẽ thực thi ISR để thực hiện một công việc và kết thúc việc thực hiện công việc này khi gặp lệnh “quay về từ trình phục vụ ngắt” (*lệnh RETI*), sau đó chương trình chính tiếp tục được thực thi tại nơi bị tạm dừng. Ta có thể nói chương trình

chính được thực thi ở mức nền (Base level), còn ISR được thực thi ở mức ngắt (Interrupt level).

Biểu diễn việc thực thi chương trình có ngắt và không có ngắt:



Một ví dụ về ngắt điển hình là nhập thông số điều khiển sử dụng bàn phím. Ta hãy khảo sát một ứng dụng của lò viba. Chương trình chính có thể điều khiển thành phần công suất của lò để thực hiện **việc nấu nướng**. Tuy nhiên trong khi đang nấu, hệ thống phải đáp ứng **việc nhập số liệu** bằng tay trên cửa lò (chẳng hạn như ta muốn yêu cầu rút ngắn bớt hay kéo dài thêm thời gian nấu), điều này có thể xảy ra tại bất cứ thời điểm nào trong quá trình nấu.

Trường hợp ta không sử dụng ngắt: Như ta đã biết, một hệ thống chỉ có thể thực thi một công việc tại một thời điểm. Cho nên khi hệ thống đang thực thi việc nấu nướng thì nó không thể thực thi việc đáp ứng nhập số liệu khi nó xảy ra và ngược lại. Vì thế trong trường hợp này hệ thống phải thực hiện cho xong việc nấu nướng rồi mới thực hiện tiếp việc đáp ứng nhập số liệu (điều này vô lý vì khi đã nấu nướng xong thì cần gì phải điều chỉnh thời gian nữa) hoặc ngược lại hệ thống phải thực hiện cho xong việc đáp ứng nhập số liệu rồi mới thực hiện tiếp việc nấu nướng (điều này cũng vô lý vì không thể biết trước được việc nhập số liệu xảy ra lúc nào, cho nên quá trình hệ thống chờ đợi việc nhập số liệu sẽ trở nên vô nghĩa).

Trường hợp ta sử dụng ngắt: Ta nhận thấy rằng việc nấu nướng là việc diễn ra liên tục từ đầu đến cuối, còn việc đáp ứng nhập số liệu chỉ xảy ra khi ta nhấn bàn phím (không xác định được thời điểm xảy ra). Vì thế, ta phân cấp cho chương trình chính (mức nền) sẽ điều khiển thành phần công suất của lò để thực hiện **việc nấu nướng**, còn việc đáp ứng nhập số liệu sẽ do ngắt điều khiển (mức ngắt). Bình thường thì lò thực hiện việc nấu nướng như đã xác định, khi người sử dụng nhấn bàn phím thì một tín hiệu ngắt được tạo ra và chương trình chính sẽ bị tạm thời dừng lại. ISR được thực thi để đọc mã phím và thay đổi các điều kiện nấu tương ứng, sau đó kết thúc bằng cách chuyển điều khiển trở về chương trình chính. Chương trình chính được thực thi tiếp từ nơi tạm dừng.

Điều quan trọng trong ví dụ nêu trên là việc nhập bàn phím xuất hiện không đồng bộ nghĩa là xuất hiện ở các khoảng thời không báo trước hoặc được điều khiển bởi phần mềm đang được thực thi trong hệ thống. **Đó là một ngắt.**

II. PHƯƠNG PHÁP PHỤC VỤ THIẾT BỊ:

Một bộ vi điều khiển có thể phục vụ một hoặc nhiều thiết bị. Có hai phương pháp phục vụ thiết bị là: phương pháp ngắt (*Interrupt*) và phương pháp thăm dò (*Polling*).

Ở phương pháp ngắt, mỗi khi có một thiết bị cần được phục vụ thì thiết bị sẽ báo cho bộ vi điều khiển bằng cách gửi đến đó một tín hiệu ngắt. Khi nhận được tín hiệu này, bộ vi điều khiển sẽ ngừng mọi công việc đang thực hiện để chuyển sang phục vụ cho thiết bị này.

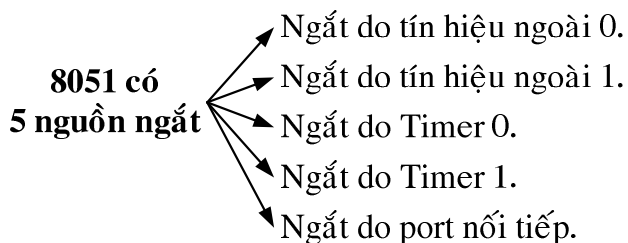
Ở phương pháp thăm dò, bộ vi điều khiển liên tục kiểm tra tình trạng của một thiết bị và khi điều kiện được đáp ứng thì nó sẽ tiến hành phục vụ cho thiết bị này. Sau đó, bộ vi điều khiển chuyển sang kiểm tra trạng thái của thiết bị kế tiếp cho đến khi tất cả thiết bị đều được phục vụ.

Điểm mạnh của phương pháp ngắt là một bộ vi điều khiển có thể phục vụ được nhiều thiết bị, nhưng dĩ nhiên là không cùng một thời điểm. Mỗi thiết bị có thể được bộ vi điều khiển phục vụ dựa theo mức ưu tiên được gán. Ở phương pháp thăm dò, thì không thể gán mức ưu tiên cho thiết bị được vì bộ vi điều khiển tiến hành kiểm tra các thiết bị theo kiểu hỏi vòng một cách lần lượt qua từng thiết bị. Ngoài ra, phương pháp ngắt cho phép bộ vi điều khiển che hoặc bỏ qua một yêu cầu phục vụ của thiết bị, điều mà phương pháp thăm dò không thể thực hiện. Tuy nhiên, lý do chính mà phương pháp ngắt được ưa chuộng hơn là vì phương pháp thăm dò lãng phí đáng kể thời gian của bộ vi điều khiển do phải hỏi dò từng thiết bị, ngay cả khi chúng không cần được phục vụ.

Để làm rõ hơn vấn đề này, chúng ta cần xem lại các ví dụ về lập trình bộ định thời đã được trình bày trong chương 4. Trong đó có lệnh **JNB TF1, \$** được sử dụng để chờ đợi cho đến khi bộ định thời tràn ($TF=1$). Ở các ví dụ này, trong khi chờ đợi chờ $TF=1$ thì bộ vi điều khiển không thể làm được công việc gì khác, điều này dẫn đến việc lãng phí thời gian. Cũng với bộ định thời này, nếu ta dùng phương pháp ngắt thì bộ vi điều khiển có thể thực hiện một số công việc nào đó trong khi đang chờ đợi chờ $TF=1$. Khi chờ $TF=1$ thì bộ vi điều khiển sẽ bị ngắt cho dù nó đang làm việc gì đi chăng nữa, điều này sẽ không làm cho bộ vi điều khiển bị lãng phí thời gian một cách vô nghĩa.

III. TỔ CHỨC NGẮT CỦA 8051:

1. Các nguồn ngắt:



Lưu ý:

- Khi ta **reset hệ thống** thì tất cả các ngắt đều bị cấm hoạt động.
- Các nguồn ngắt này được cho phép hoặc cấm hoạt động bằng lệnh do người lập trình thiết lập cho từng ngắt.
- Việc xử lý các ngắt được thực hiện qua 2 sơ đồ:
 - Sơ đồ ưu tiên ngắt → có thể thay đổi được và do người lập trình thiết lập.
 - Sơ đồ chuỗi vòng → cố định, không thay đổi được.

⇒ Hai sơ đồ này giúp CPU giải quyết các vấn đề liên quan đến ngắt như: *hai hay nhiều ngắt xảy ra đồng thời hoặc một ngắt xảy ra trong khi một ngắt khác đang được thực thi.*

Các cờ ngắt của chip 8051:

Loại ngắt	Cờ ngắt	Vị trí của bit trong các thanh ghi
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer 1	TF1	TCON.7
Ngắt Timer 0	TF0	TCON.5
Ngắt port nối tiếp	RI	SCON.0
Ngắt port nối tiếp	TI	SCON.1

Lưu ý:

- Một ngắt xảy ra thì cờ ngắt tương ứng sẽ được set bằng 1.
- Khi ISR của ngắt được thực thi thì cờ ngắt tương ứng sẽ tự động bị xóa về 0 bằng phần cứng (ngoại trừ cờ ngắt RI và TI phải được xóa về 0 bằng phần mềm).
- Đối với ngắt ngoài sẽ có hai cách kích hoạt để tạo ra một tín hiệu ngắt: ngắt ngoài kích hoạt khi có **mức thấp** và ngắt ngoài kích hoạt khi có **cạnh âm** tại chân INT0\ hoặc INT1\.

2. Qui định việc chọn loại kích hoạt cho ngắt ngoài:

Việc chọn lựa loại kích hoạt cho các ngắt ngoài, thuộc *loại kích hoạt cạnh* hay thuộc *loại kích hoạt mức*, thì được lập trình thông qua các bit IT0 và IT1 của thanh ghi TCON.

IT0 = 0 → Ngắt ngoài 0 được kích khởi bởi việc phát hiện **mức thấp** tại chân INT0\.

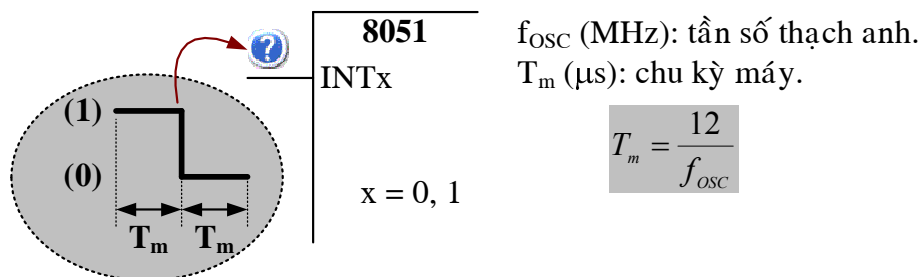
IT0 = 1 → Ngắt ngoài 0 được kích khởi bởi việc phát hiện **cạnh âm** tại chân INT0\.

IT1 = 0 → Ngắt ngoài 1 được kích khởi bởi việc phát hiện **mức thấp** tại chân INT1\.

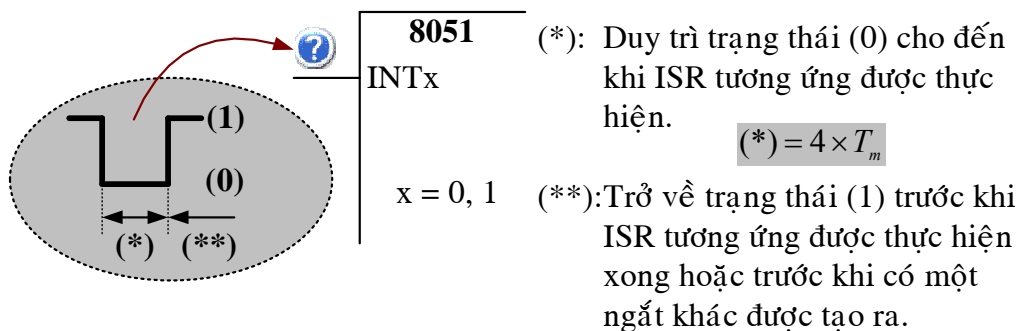
IT1 = 1 → Ngắt ngoài 1 được kích khởi bởi việc phát hiện **cạnh âm** tại chân INT1\.

Lưu ý: Khi tạo tín hiệu ngắt tại chân INT0\ hoặc INT1\ ta cần phải chú ý đến thời gian duy trì tác động của tín hiệu ngắt.

- Đối với loại ngắt kích hoạt cạnh âm (*thời gian tối thiểu*):



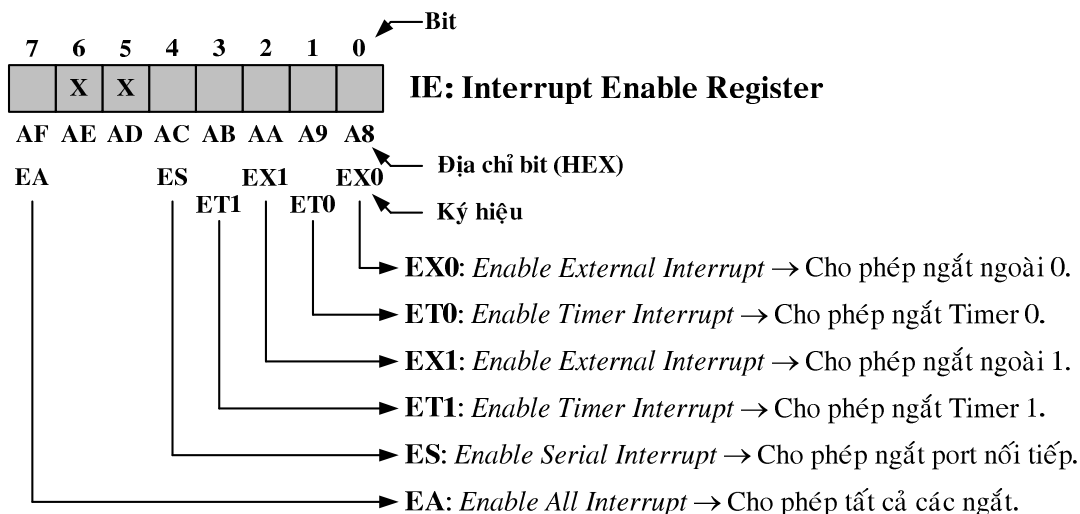
- Đối với loại ngắt kích hoạt mức thấp (*thời gian tối đa*):



3. Thanh ghi cho phép ngắt (IE):

Thanh ghi cho phép ngắt (IE: *Interrupt Enable*): chứa các bit dùng để cho phép hoặc cấm các ngắt hoạt động.

Cấu trúc của thanh ghi IE:



Lưu ý: Cho phép khi bit = 1 Cấm khi bit = 0

Hai điều kiện để một ngắt được phép hoạt động là:

- Bit EA = 1.
- Bit ngắt tương ứng = 1.

Ví dụ: Để ngắt của Timer 1 được phép hoạt động ta dùng lệnh:

```
SETB ET1
SETB EA
```

hoặc

```
MOV IE, #10001000B
```

Mặc dù cả cách trên đều cho ta một kết quả như nhau sau khi hệ thống được thiết lập lại trạng thái ban đầu (*reset hệ thống*). Tuy nhiên trong khi chương trình đang hoạt động thì ảnh hưởng của hai cách này có khác nhau vì cách thứ hai ghi lên thanh ghi IE.

Cách thứ nhất, sử dụng hai lệnh SETB nên chỉ ảnh hưởng đến 2 bit cần tác động mà không gây ảnh hưởng đến 5 bit còn lại của thanh ghi IE. Trong khi đó, cách thứ hai chỉ sử dụng lệnh MOV nên sẽ làm cho 5 bit còn lại này bit xóa mất. Tốt nhất ta nên khởi động thanh ghi IE bằng lệnh MOV ở đầu chương trình ngay sau khi hệ thống được thiết lập lại. Việc cho phép hoặc không cho phép các ngắt trong chương trình nên sử dụng các lệnh SETB hoặc CLR để tránh ảnh hưởng đến các bit khác trong thanh ghi IE.

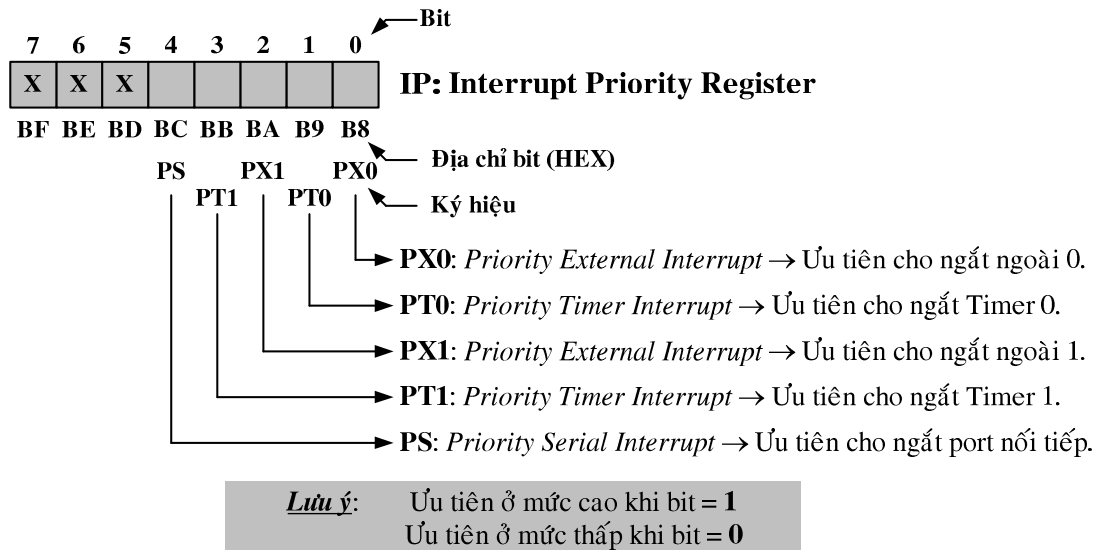
4. Thanh ghi ưu tiên ngắt (IP):

Khái niệm ưu tiên ngắt giúp 8051 giải quyết vấn đề **hai tín hiệu ngắt xuất hiện đồng thời** và vấn đề **một tín hiệu ngắt xuất hiện trong khi một ngắt khác đang được thực thi**.

Ngắt ưu tiên mức cao → Ngắt ưu tiên mức thấp

Thanh ghi ưu tiên ngắt (IP: *Interrupt Priority*): chứa các bit dùng để thiết lập mức độ ưu tiên (*mức cao hay mức thấp*) cho từng ngắt riêng rẽ.

Cấu trúc của thanh ghi IP:



Khi hệ thống được thiết lập lại trạng thái ban đầu thì tất cả các ngắt đều sẽ được mặc định ở mức ưu tiên thấp. Ý tưởng “các mức ưu tiên” cho phép một trình phục vụ ngắt được tạm dừng bởi một ngắt khác nếu ngắt mới này có mức ưu tiên cao hơn mức ưu tiên của ngắt hiện đang được phục vụ. Điều này hoàn toàn hợp lý đối với 8051 vì ta chỉ có hai mức ưu tiên. Nếu có ngắt có mức ưu tiên cao xuất hiện, trình phục vụ ngắt cho ngắt có mức ưu tiên thấp phải tạm dừng (*nghĩa là bị ngắt*). Ta không thể tạm dừng một chương trình phục vụ ngắt có mức ưu tiên cao.

Chương trình chính do được thực thi ở mức nền và không được kết hợp với một ngắt nào nên luôn luôn bị ngắt bởi các ngắt cho dù các ngắt có mức ưu tiên thấp hay mức ưu tiên cao. Nếu có hai ngắt với mức ưu tiên ngắt khác nhau xuất hiện đồng thời, ngắt có mức ưu tiên cao sẽ được phục vụ trước.

5. Thứ tự chuỗi vòng ngắt (Interrupt Polling Sequence):

Khái niệm chuỗi vòng giúp 8051 giải quyết vấn đề **hai hay nhiều tín hiệu ngắt có mức ưu tiên giống nhau xuất hiện đồng thời**.

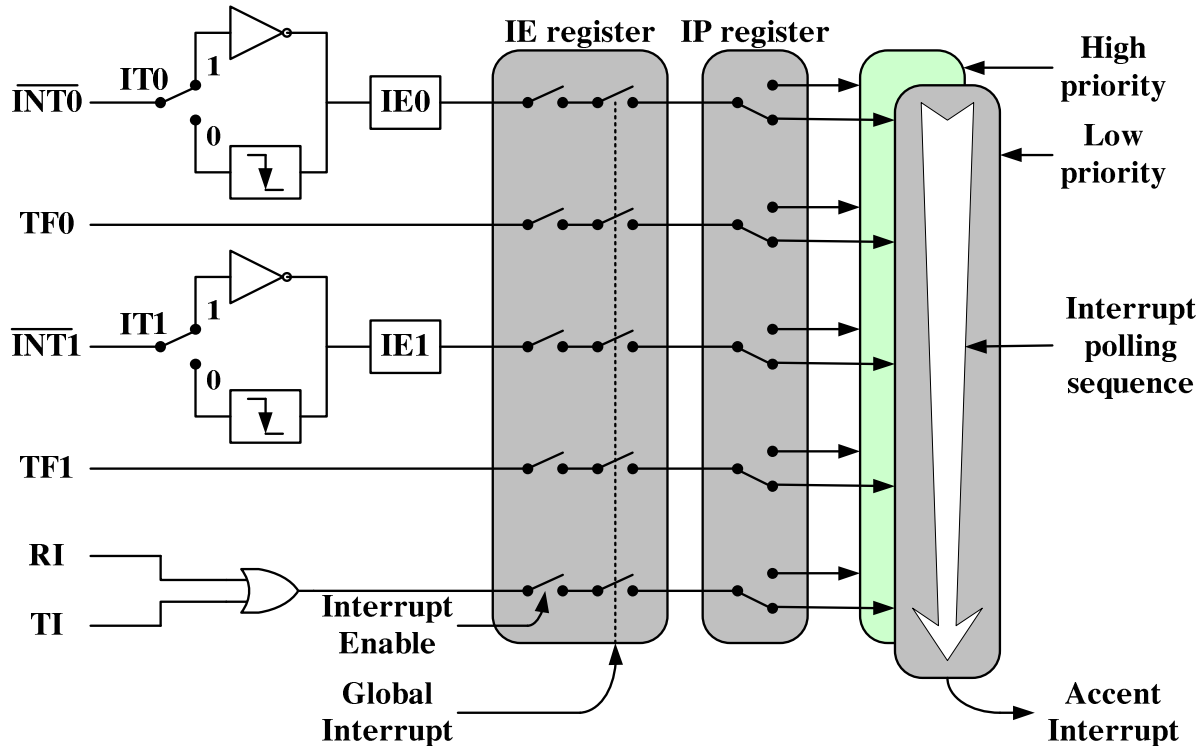
Chuỗi vòng này sẽ là (*được sắp xếp theo thứ tự từ thấp đến cao*):

**Ngắt ngoài 0 → Ngắt Timer 0 → Ngắt ngoài 1 →
Ngắt Timer 1 → Ngắt port nối tiếp → Ngắt Timer 2 (chỉ có ở 8052)**

Hình dưới đây minh họa 5 nguyên nhân ngắt, cơ chế cho phép riêng rẽ và toàn cục, chuỗi vòng và các mức ưu tiên. Trạng thái của tất cả các nguyên nhân ngắt được thể hiện thông qua các bit cờ tương ứng trong các thanh ghi chức năng đặc biệt có liên quan. Dĩ nhiên nếu một ngắt nào đó không được phép, nguyên nhân ngắt tương ứng không thể tạo ra một ngắt nhưng phần mềm vẫn có thể kiểm tra cờ ngắt đó. Lấy thí dụ bộ định thời và port nối tiếp trong hai chương trước sử dụng các cờ ngắt một cách rộng rãi dù không có ngắt tương ứng xảy ra, nghĩa là không sử dụng các ngắt.

Ngắt do port nối tiếp là kết quả OR của cờ ngắt khi thu RI (*cờ ngắt thu*) và cờ ngắt khi phát TI (*cờ ngắt phát*).

Cấu trúc ngắt của 8051:



IE register: thanh ghi IE.

IP register: thanh ghi IP.

High priority interrupt: ngắt ưu tiên cao.

Low priority interrupt: ngắt ưu tiên thấp.

Interrupt polling sequence: thứ tự chuỗi vòng ngắt.

Interrupt enable: cho phép ngắt.

Global enable: cho phép toàn cục.

IV. XỬ LÝ NGẮT VÀ CÁC VECTO NGẮT:

1. Qui trình xử lý ngắt:

Các thao tác sẽ xảy ra khi có một ngắt xuất hiện và nó được CPU chấp nhận:

- Hoàn tất thực thi lệnh tại thời điểm đó và dừng chương trình chính.
- Giá trị của thanh ghi PC được cất vào stack.
- Trạng thái của ngắt tại thời điểm đó được lưu giữ lại.
- Các ngắt được giữ lại ở mức ngắt.
- Địa chỉ của ISR của ngắt tương ứng được nạp vào thanh ghi PC.
- ISR của ngắt tương ứng được thực thi.

(ISR thực thi xong khi gặp lệnh RETI).

- Giá trị trong stack (của PC cũ) được phục hồi lại vào thanh ghi PC.
- Trạng thái các ngắt được phục hồi lại.
- Chương trình chính tiếp tục được thực thi tại chỗ bị tạm dừng.

ISR được thực thi để đáp ứng công việc của ngắt. Việc thực thi ISR kết thúc khi gặp lệnh RET (trở về từ một trình phục vụ ngắt). Lệnh này lấy lại giá trị cũ của bộ đếm chương trình PC từ stack và phục hồi trạng thái của ngắt cũ. Việc thực thi chương trình chính được tiếp tục ở nơi bị tạm ngưng.

2. Các vector ngắt:

Khi một ngắt được chấp nhận, giá trị được nạp cho bộ đếm chương trình PC được gọi là vector ngắt. Vector ngắt là địa chỉ bắt đầu của chương trình phục vụ ngắt (ISR) của ngắt tương ứng.

Vector reset hệ thống cũng được xem như là một ngắt: chương trình chính bị ngắt và bộ đếm chương trình PC được nạp giá trị mới.

Khi một trình phục vụ ngắt được trở đến, cờ gây ra ngắt sẽ tự động bị xóa về 0 bởi phần cứng. Các ngoại lệ bao gồm các cờ RI và TI đối với các ngắt do port nối tiếp, các nguyên nhân ngắt thuộc loại này do có hai khả năng tạo ra ngắt nên trong thực tế CPU không xóa cờ ngắt.

Bảng qui định địa chỉ bắt đầu của các ISR (bảng vector ngắt):

Loại ngắt	Cờ ngắt	Địa chỉ của vectơ ngắt
Reset hệ thống	→ RST	→ 0000H
Ngắt ngoài 0	→ IE0	→ 0003H
Ngắt Timer 0	→ IT0	→ 000BH
Ngắt ngoài 1	→ IE1	→ 0013H
Ngắt Timer 1	→ IT1	→ 001BH
Ngắt port nối tiếp	→ RI hoặc TI	→ 0023H

V. THIẾT KẾ CÁC CHƯƠNG TRÌNH SỬ DỤNG NGẮT:

1. Tổng quan:

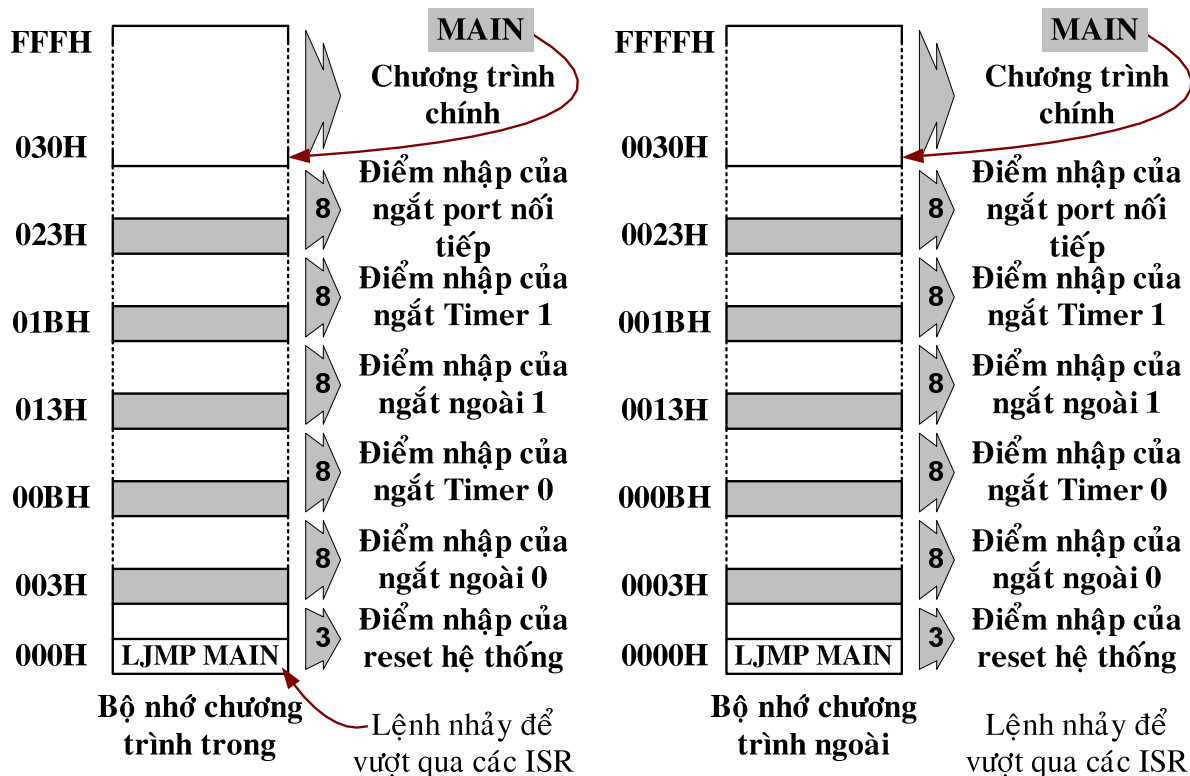
Khi **thiết kế các chương trình không sử dụng ngắt** thì ta sẽ gặp phải những trường hợp CPU hoàn toàn tiêu phí thời gian vào việc chờ đợi các tác nhân cần thiết xảy ra (Ví dụ: sự tràn của cờ TF0, TF1; việc thu xong một dữ liệu và cờ RI=1; việc phát xong một ký tự và cờ TI=1; v.v...) để sau đó mới tiếp tục thực hiện công việc.

⇒ Điều này không thích hợp cho các ứng dụng điều khiển đòi hỏi phải tác động qua lại với nhiều thiết bị cùng lúc.

Để giải quyết vấn đề trên ta cần **thiết kế các chương trình có sử dụng đến ngắt**.

⇒ Vì nó giúp cho CPU không tốn thời gian để chờ đợi tác nhân mà chỉ khi nào tác nhân xảy ra thì CPU mới thực hiện việc xử lý tác nhân đó, khoảng thời gian tác nhân không xảy ra thì CPU sẽ làm việc khác.

Tổ chức bộ nhớ khi sử dụng ngắt:



Khuông mẫu cho một chương trình có sử dụng ngắt:

```

ORG 0000H      ;Điểm nhập của reset hệ thống.
LJMP MAIN      ;Lệnh nhảy để vượt qua các ISR.
.....        ;Điểm nhập của các ISR.
.....
.....
MAIN:          ORG 0030H      ;Điểm nhập của chương trình chính.
.....          ;Chương trình chính bắt đầu.
.....
.....
END
    
```

2. Thiết kế các chương trình ISR kích thước nhỏ:

Điều kiện: Khi ISR có kích thước không quá 8 byte (kể cả lệnh RETI).

⇒ ISR phải được viết trong phạm vi điểm nhập tương ứng của nó trong bộ nhớ chương trình (xem phần tổ chức bộ nhớ khi sử dụng ngắt).

Lưu ý:

- Nếu chỉ có một nguyên nhân ngắt được sử dụng thì ISR của nó có thể được viết tràn sang điểm nhập của các ISR khác (nghĩa là ISR có kích thước lớn hơn 8 byte, nhưng phải nhỏ hơn 46 byte). Vì khi đó vùng nhớ của các ISR khác không được dùng đến nên ta có thể tận dụng để sử dụng cho ISR này.

- Nếu có nhiều nguyên nhân ngắt được sử dụng thì ta phải cẩn thận để đảm bảo cho các ISR được bắt đầu đúng vị trí mà không tràn sang ISR kế (nghĩa là ISR có kích thước không quá 8 byte).

Khuông mẫu chương trình: (*Ví dụ: dùng ngắt Timer0 và ngắt ngoài 1*)

```

ORG 0000H      ;Điểm nhập của reset hệ thống.
LJMP MAIN      ;Lệnh nhảy để vượt qua các ISR.
ORG 000BH      ;Điểm nhập cho ISR của Timer 0.
.....         ;ISR của Timer 0.
.....
RETI           ;Kết thúc ISR của Timer 0.
ORG 0013H      ;Điểm nhập cho ISR của ngắt ngoài 1.
.....         ;ISR của ngắt ngoài 1.
.....
RETI           ;Kết thúc ISR của ngắt ngoài 1.
ORG 0030H      ;Điểm nhập của chương trình chính.
MAIN:         ..... ;Chương trình chính bắt đầu.
.....
.....
END

```

3. Thiết kế các chương trình ISR kích thước lớn:

Điều kiện: Khi ISR có kích thước vượt quá 8 byte.

⇒ ISR không thể viết vào điểm nhập tương ứng của nó trong bộ nhớ chương trình (vì kích thước điểm nhập chỉ có 8 byte) → ta phải chuyển ISR này đến một nơi khác trong bộ nhớ chương trình hoặc có thể viết lần qua điểm nhập của ISR kế tiếp (nếu ISR đó không sử dụng).

Khuông mẫu chương trình: (*Ví dụ: dùng ngắt Timer0 và ngắt ngoài 1*)

```

ORG 0000H      ;Điểm nhập của reset hệ thống.
LJMP MAIN      ;Lệnh nhảy để vượt qua các ISR.
ORG 000BH      ;Điểm nhập cho ISR của Timer 0.
LJMP T0ISR     ;Lệnh nhảy đến ISR của Timer 0.
ORG 0013H      ;Điểm nhập cho ISR của ngắt ngoài 1.
LJMP EX1ISR    ;Lệnh nhảy đến ISR của ngắt ngoài 1.
ORG 0030H      ;Điểm nhập của chương trình chính.
MAIN:         ..... ;Chương trình chính bắt đầu.
.....
.....
T0ISR:       SJMP $      ;Lệnh cách ly chương trình.
.....         ;ISR của ngắt Timer 0.
.....
EX1ISR:      RETI       ;Kết thúc ISR của Timer 0.
.....         ;ISR của ngắt ngoài 1.
.....
RETI         ;Kết thúc ISR của ngắt ngoài 1.
END

```

• **Nhận xét tổng quát:**

○ Để đơn giản, các chương trình của chúng ta chỉ làm việc ở thời điểm bắt đầu. Chương trình chính khởi động port nối tiếp, bộ định thời và các thanh ghi ngắt sao cho thích hợp với yêu cầu đặt ra và rồi không làm gì cả. Công việc hoàn toàn được thực hiện bên trong các ISR. Sau các lệnh khởi động, chương trình chính chứa và thực hiện lệnh sau đây (*lệnh nhảy tại chỗ – không làm gì cả*):

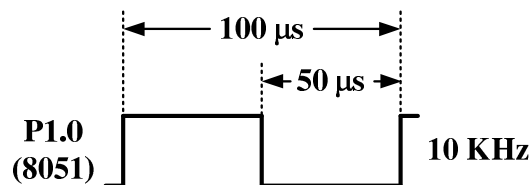
SJMP \$

○ Khi có một tín hiệu ngắt xuất hiện, chương trình chính tạm thời bị dừng lại trong khi ISR được thực thi. Lệnh RETI ở cuối của các ISR sẽ trả điều khiển về cho chương trình chính và chương trình chính tiếp tục không làm gì cả (*lệnh nhảy tại chỗ*). Điều này không có gì là không tự nhiên đối với chúng ta. Trong nhiều ứng dụng hướng điều khiển, phần lớn công việc được thực hiện trong trình phục vụ ngắt. Các ví dụ minh họa dưới đây sẽ cho ta thấy điều này.

VI. CÁC VÍ DỤ MINH HỌA:

1. Ví dụ minh họa xử lý ngắt Timer:

Ví dụ 1: *Viết chương trình sử dụng Timer 0 và các ngắt để tạo ra một sóng vuông có tần số 10 KHz trên chân P1.0, $f_{Osc} = 12MHz$.*



Giải

	ORG	0000H	;Điểm nhập reset.
	LJMP	MAIN	;Nhảy qua khỏi các vectơ ngắt.
	ORG	000BH	;Điểm nhập ISR của Timer 0.
T0ISR:			;ISR của Timer 0.
	CPL	P1.0	;Lấy bù.
	RETI		;Kết thúc ISR của Timer 0.
	ORG	0030H	;Điểm nhập chương trình chính.
MAIN:			;Chương trình chính bắt đầu.
	MOV	TMOD, #02H	;Chọn chế độ 2 cho Timer 0.
	MOV	TH0, #(-50)	;Định thời 50μs.
	SETB	TR0	;Cho Timer 0 hoạt động.
	MOV	IE, #82H	;Cho phép ngắt Timer 0.
	SJMP	\$	;Không làm gì (nhảy tại chỗ).
	END		;Kết thúc chương trình.

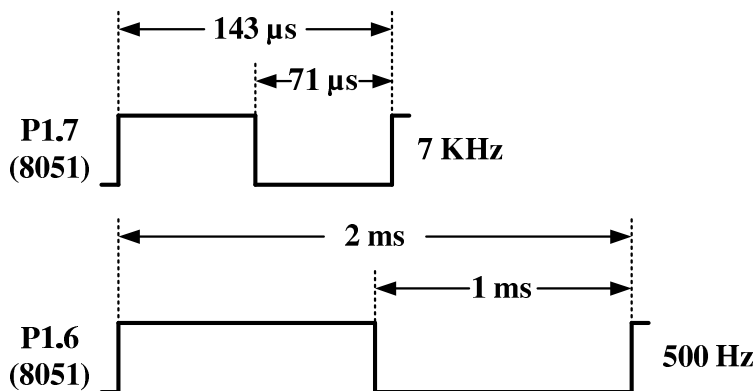
 **Lưu ý:** Lệnh SJMP\$ có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc tạo sóng vuông $f=10KHz$ tại chân P1.0. Vì cứ sau mỗi 50μs thì những công việc đó sẽ bị tạm dừng (*ngắt Timer 0 xuất hiện*) để CPU thực hiện việc tạo sóng vuông rồi quay về thực hiện tiếp những công việc đó.

Ngay sau khi reset hệ thống, bộ đếm chương trình PC được nạp 0000H. Lệnh đầu tiên được thực thi là **LJMP MAIN**, lệnh này rẽ nhánh đến chương trình chính ở địa chỉ 0030H trong bộ nhớ chương trình. Ba lệnh đầu tiên của chương trình chính sẽ khởi động Timer 0 ở chế độ 8 bit tự động nạp lại (*Mode 2*), sao cho Timer 0 sẽ tràn sau mỗi 50μs. Lệnh **MOV IE, #82H** cho phép các ngắt do Timer0 tạo ra. Mỗi một lần tràn, Timer sẽ tạo ra một ngắt. Dĩ nhiên là lần tràn đầu tiên sẽ không xuất hiện sau

50 μ s do chương trình chính đang ở trong vòng lặp “không làm gì”. Khi ngắt xuất hiện sau mỗi 50 μ s, chương trình chính bị ngắt và ISR cho Timer0 được thực thi. Ở ví dụ trên ISR này chỉ đơn giản lấy bù bit của port và quay trở về chương trình chính nơi vòng lặp “không làm gì” được thực thi để chờ một ngắt mới sau mỗi 50 μ s.

Lưu ý là cờ tràn TF0 không cần được xóa bởi phần mềm do khi các ngắt được cho phép thì cờ này tự động được xóa bởi phần cứng khi CPU trở đến trình phục vụ ngắt.

Ví dụ 2: Viết chương trình sử dụng các ngắt để tạo đồng thời các dạng sóng vuông có tần số 7 KHz và 500 Hz tại các chân P1.7 và P1.6 (không quan tâm đến độ lệch pha của hai sóng này), $f_{Osc} = 12\text{MHz}$.



Giải

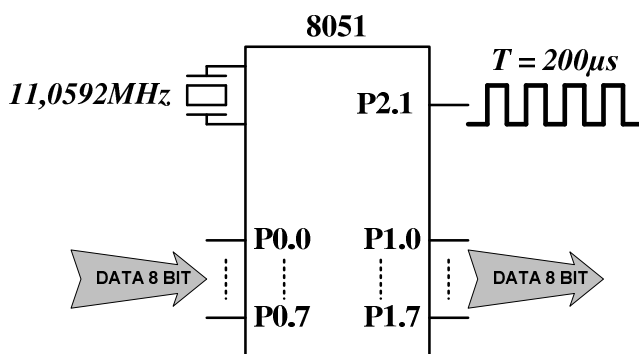
ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	000BH	;Điểm nhập ISR của Timer 0.
LJMP	T0ISR	;Lệnh nhảy đến ISR Timer 0.
ORG	001BH	;Điểm nhập ISR của Timer 1.
LJMP	T1ISR	;Lệnh nhảy đến ISR Timer 1.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	TMOD, #12H	;Chọn chế độ 2 cho Timer 0.
		;Chọn chế độ 1 cho Timer 1.
MOV	TH0, #-71	;Định thời 71 μ s.
SETB	TR0	;Cho Timer 0 hoạt động.
SETB	TF1	;Buộc Timer 1 ngắt.
MOV	IE, #8AH	;Cho phép các ngắt hoạt động.
SJMP	\$	;Không làm gì (nhảy tại chỗ).
T0ISR:		;ISR của ngắt Timer 0.
CPL	P1.7	;Lấy bù.
RETI		;Kết thúc ISR của Timer 0.
T1ISR:		;ISR của ngắt Timer 1.
CLR	TR1	;Dừng Timer 1.
MOV	TH1, #HIGH(-1000)	;Định thời 1ms.
MOV	TL1, #LOW(-1000)	
SETB	TR1	;Cho Timer 1 hoạt động.
CPL	P1.6	;Lấy bù.
RETI		;Kết thúc ISR của Timer 1.
END		;Kết thúc chương trình.

Lưu ý: Lệnh SJMP\$ có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc tạo sóng vuông $f = 7\text{KHz}$ và $f = 500\text{Hz}$ tại chân P1.7 và chân P1.6. Vì cứ sau mỗi $71\mu\text{s}$ và 1ms thì những công việc đó sẽ bị tạm dừng (ngắt Timer 0 và ngắt Timer 1 xuất hiện) để CPU thực hiện việc tạo sóng vuông rồi quay về thực hiện tiếp những công việc đó.

Việc tổ hợp các ngõ ra này rất khó tạo ra được trên một hệ thống không sử dụng điều khiển ngắt. Timer 0 hoạt động ở chế độ 2, được sử dụng để tạo ra dạng sóng 7KHz trên chân P1.7. Timer 1 hoạt động ở chế độ 1, được sử dụng để tạo ra dạng sóng 500Hz trên chân P1.6. Sở dĩ trong trường hợp này, Timer 1 phải được thiết lập để hoạt động ở chế độ 1 là do dạng sóng 500Hz yêu cầu thời gian mức cao và thời gian mức thấp là 1ms , chế độ 2 không sử dụng được trong trường hợp này.

Cũng cần chú ý là các thanh ghi TH1/TL1 không được khởi động ở đầu chương trình chính như trường hợp của thanh ghi TH0. Do TH1/TL1 phải được nạp lại sau mỗi lần bộ định thời tràn, TF1 được set bằng 1 trong chương trình chính bởi phần mềm (lệnh **SETB TF1**) được sử dụng để buộc phải có một ngắt ban đầu ngay trước khi các ngắt được cho phép. Điều này có hiệu quả cho việc bắt đầu dạng sóng 500Hz .

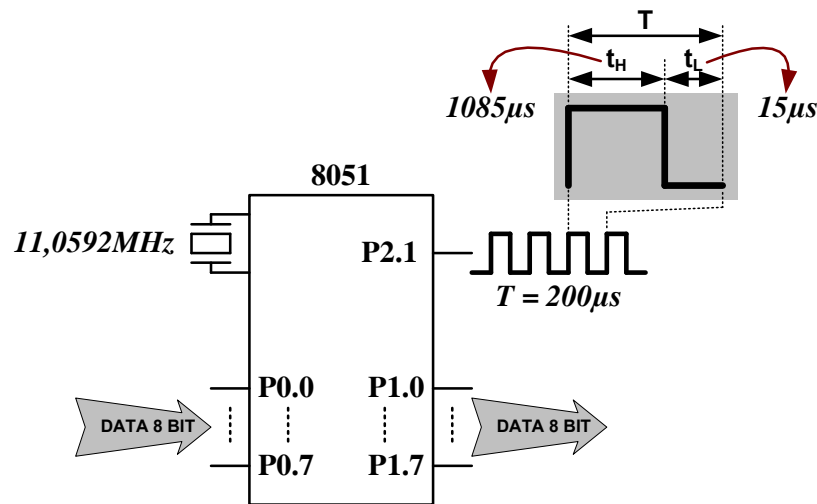
Ví dụ 3: Viết chương trình liên tục nhận dữ liệu 8 bit ở cổng P0 và sau đó gửi dữ liệu này đến cổng P1. Trong thời gian này cần tạo ra trên chân P2.1 một sóng vuông có chu kỳ là $200\mu\text{s}$. Sử dụng Timer 0 để tạo sóng vuông, $f_{\text{osc}} = 11,0592\text{MHz}$.



Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	000BH	;Điểm nhập ISR của Timer 0.
CPL	P2.1	;Đảo trạng thái chân P2.1, tạo xung.
RETI		;Kết thúc ISR của Timer 0.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	TMOD, #02H	;Chọn chế độ 2 cho Timer 0.
MOV	P0, #0FFH	;Cấu hình Port 0 là cổng vào.
MOV	TH0, #(-92)	;Định thời $100\mu\text{s}$ (nửa chu kỳ).
MOV	IE, #82H	;Cho phép ngắt Timer 0 hoạt động.
SETB	TR0	;Cho Timer 0 hoạt động.
BACK:		
MOV	A, P0	;Nhận dữ liệu từ P0.
MOV	P1, A	;Xuất dữ liệu vừa nhận được ra P1.
SJMP	BACK	
END		;Kết thúc chương trình.

Ví dụ 4: Viết chương trình liên tục nhận dữ liệu 8 bit ở cổng P0 và sau đó gửi dữ liệu này đến cổng P1. Trong thời gian này cần tạo ra trên chân P2.1 một sóng vuông với yêu cầu: thời gian sóng ở mức cao là $1085\mu s$ và thời gian sóng ở mức thấp là $15\mu s$. Sử dụng Timer 1 để tạo sóng vuông, $f_{Osc}=11,0592MHz$.



Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	001BH	;Điểm nhập ISR của Timer 1.
LJMP	T1ISR	;Lệnh nhảy đến ISR Timer 1.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	TMOD, #10H	;Chọn chế độ 1 cho Timer 1.
MOV	P0, #0FFH	;Cấu hình Port 0 là cổng vào.
MOV	TL1, #18H	;Định thời $1085\mu s$ (cho nửa chu kỳ đầu).
MOV	TH1, #0FCH	;
MOV	IE, #88H	;Cho phép ngắt Timer 1 hoạt động.
SETB	TR1	;Cho Timer 1 hoạt động.
BACK:		
MOV	A, P0	;Nhận dữ liệu từ P0.
MOV	P1, A	;Xuất dữ liệu vừa nhận được ra P1.
SJMP	BACK	;Lặp lại liên tục hai thao tác trên.
T1ISR:		;ISR của ngắt Timer 1.
CLR	TR1	;Dừng Timer 1.
CLR	P2.1	;Đảo trạng thái chân P2.1, tạo xung.
MOV	R2, #4	;Định thời $8T_{Machine}$ (tổng thời gian: $15\mu s$),
DJNZ	R2, \$	;tạo thời gian trễ (cho nửa chu kỳ sau).
MOV	TL1, #18H	;Định thời $1085\mu s$ (cho nửa chu kỳ đầu).
MOV	TH1, #0FCH	;
SETB	TR1	;Cho Timer 1 hoạt động.
SETB	P2.1	;Đảo trạng thái chân P2.1, tạo xung.
RETI		;Kết thúc ISR của Timer 1.
END		;Kết thúc chương trình.

2. Ví dụ minh họa xử lý ngắt port nối tiếp:

Các ngắt do port nối tiếp xuất hiện khi cờ ngắt phát TI hoặc cờ ngắt thu RI được set bằng 1. Một ngắt phát xuất hiện khi việc phát một ký tự đã ghi vào SBUF hoàn tất. Một ngắt thu xuất hiện khi một ký tự được thu nhận đầy đủ và đang ở trong SBUF để chờ được đọc. Như vậy, ngắt phát xảy ra khi bộ đệm phát SBUF rỗng, ngắt thu xảy ra khi bộ đệm thu SBUF đầy.

Các ngắt do port nối tiếp có khác với các ngắt do bộ định thời. Cờ gây ra ngắt ở port nối tiếp không được xóa bởi phần cứng khi CPU trở tới trình phục vụ ngắt. Lý do là vì ở đây ta có hai nguyên nhân tạo ra ngắt ở port nối tiếp, cụ thể là hai ngắt tạo ra bởi hai cờ TI và RI. Nguyên nhân ngắt phải được xác định trong trình phục vụ ngắt và cờ tạo ra ngắt được xóa bởi phần mềm. Cần nhắc lại là với các ngắt do bộ định thời, cờ tạo ra ngắt được xóa bởi phần cứng khi CPU trở tới trình phục vụ ngắt.

Ví dụ 1: Viết chương trình sử dụng các ngắt để liên tục phát đi mã ASCII (có giá trị từ 20H đến 7EH) đến một thiết bị đầu cuối nối với 8051 qua port nối tiếp. Biết rằng $f_{osc} = 11,0592\text{MHz}$.

Giải

	ORG	0000H	;Điểm nhập reset.
	LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
	ORG	0023H	;Điểm nhập ISR port nối tiếp.
	LJMP	SPISR	;Lệnh nhảy đến ISR port nối tiếp.
	ORG	0030H	;Điểm nhập chương trình chính.
MAIN:			;Chương trình chính bắt đầu.
	MOV	SCON, #42H	;Chọn chế độ 1 cho port nối tiếp, TI=1 để buộc có ngắt và gọi ký tự đầu tiên.
	MOV	TMOD, #20H	;Chọn chế độ 2 cho Timer 1.
	MOV	TH1, #(-24)	;Tốc độ baud = 1200.
	SETB	TR1	;Cho Timer 1 hoạt động.
	MOV	A, #20H	;Nạp mã cho ký tự đầu tiên.
	MOV	IE, #90H	;Cho phép ngắt port nối tiếp.
	SJMP	\$	;Không làm gì (nhảy tại chỗ).
SPISR:			;ISR của ngắt port nối tiếp.
	CJNE	A, #7FH, SKIP	;Kiểm tra kết thúc bảng mã.
	MOV	A, #20H	;Trở lại ký tự đầu tiên.
SKIP:			
	MOV	SBUF, A	;Truyền ký tự ra port nối tiếp.
	INC	A	;Lấy mã của ký tự kế tiếp.
	CLR	TI	;Xóa cờ ngắt phát.
	RETI		;Kết thúc ISR của port nối tiếp.
	END		;Kết thúc chương trình.

 **Lưu ý:** Lệnh SJMP\$ có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc phát mã ASCII thông qua port nối tiếp. Vì cứ sau mỗi lần port nối tiếp truyền xong một ký tự thì những công việc đó sẽ bị tạm dừng (ngắt port nối tiếp xuất hiện) để CPU thực hiện việc kiểm tra ký tự kết thúc bảng mã và lấy mã của ký tự kế tiếp để tiếp tục phát đi rồi quay về thực hiện tiếp những công việc đó.

Bảng mã ASCII bao gồm 128 mã 7 bit (xem thêm trong phần phụ lục “Phụ lục 4 – Bảng mã ASCII”). Sau khi nhảy đến nhãn MAIN ở địa chỉ 0030H, ba lệnh đầu tiên dùng khởi động Timer 1 để cung cấp xung clock 1200 baud cho port nối tiếp, lệnh **MOV SCON, #42H** khởi động port nối tiếp ở

chế độ 1 (UART 8 bit có tốc độ baud thay đổi) và cho TI=1 để buộc tạo ra một ngắt trước khi các ngắt được cho phép hoạt động. Sau đó mã ASCII đầu tiên (20H) được nạp cho thanh ghi A và các ngắt do port nối tiếp được cho phép. Cuối cùng phần chính của chương trình đi vào vòng lặp “không làm gì” (tức là lệnh **SJMP \$**).

Trình phục vụ ngắt của port nối tiếp làm tất cả công việc một khi chương trình chính đã thiết lập các điều kiện ban đầu. Hai lệnh đầu tiên kiểm tra thanh ghi A và nếu mã ASCII đạt đến 7FH (nghĩa là mã vừa mới được phát đi là 7EH) thì thanh ghi A sẽ được thiết lập lại với nội dung là 20H. Sau đó mã ASCII được gửi đến bộ đệm của port nối tiếp (lệnh **MOV SBUF,A**) để được phát đi. Thực hiện việc tăng giá trị trong thanh ghi A để có mã kế tiếp, chờ phát được xóa (lệnh **CLR TI**) và trình phục vụ ngắt kết thúc (lệnh **RETI**). Điều khiển sẽ trả về chương trình chính và lệnh **SJMP \$** được thực thi cho đến khi TI lại được set bằng 1 cho lần phát dữ liệu kế tiếp.

Nếu ta so sánh tốc độ của CPU với tốc độ truyền dữ liệu, ta nhận thấy rằng lệnh **SJMP \$** được thực thi với phần trăm tỉ lệ thời gian rất lớn trong chương trình này. Phần trăm tỉ lệ này là bao nhiêu? Ở tốc độ 1200 baud, mỗi một bit được truyền đi trong một khoảng thời gian là 0,8333ms. Như vậy 8 bit dữ liệu cộng với 1 bit start, 1 bit stop (một lần truyền một dữ liệu gồm 10 bit) chiếm 8,333ms. Thời gian thực thi tệ nhất của trình phục vụ ngắt SPISR là tổng của số chu kỳ cho mỗi lệnh nhân với 1,085μs (tức $1T_{Machine}$), thời gian này tính được là $8 \times 1,085\mu s = 8,68\mu s$. Do vậy, với 8333μs dùng để truyền một dữ liệu mà chỉ có 8,68μs dành cho trình phục vụ ngắt SPISR. Lệnh **SJMP \$** thực thi trong khoảng thời gian $8333\mu s - 8,68\mu s = 8324,32\mu s$ tức là khoảng 99,9% thời gian. Do vậy ngắt cần được sử dụng để có thể loại bỏ được khoảng thời gian “không làm gì” này (chiếm đến 99,9% thời gian hoạt động của chương trình truyền dữ liệu này). Muốn vậy thì lệnh **SJMP \$** có thể được thay bởi các lệnh khác để thực hiện những công việc khác theo yêu cầu của ứng dụng. Các ngắt vẫn xuất hiện và các ký tự vẫn được phát từ port nối tiếp sau mỗi 8,333ms.

Ví dụ 2: Viết chương trình điều khiển 8051 đọc dữ liệu từ cổng P1 và ghi liên tục tới cổng P2. Đồng thời đưa một bản sao dữ liệu tới port nối tiếp để thực hiện việc truyền dữ liệu nối tiếp. Biết rằng tốc độ truyền là 9600 baud và $f_{Osc} = 11,0592MHz$

Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vectơ ngắt.
ORG	0023H	;Điểm nhập ISR port nối tiếp.
LJMP	SPISR	;Lệnh nhảy đến ISR port nối tiếp.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	P1, #0FFH	;Cấu hình Port 1 là cổng vào.
MOV	SCON, #42H	;Chọn chế độ 1 cho port nối tiếp, cảm thu.
MOV	TMOD, #20H	;Chọn chế độ 2 cho Timer 1.
MOV	TH1, #(-3)	;Tốc độ baud = 9600.
MOV	IE, #90H	;Cho phép ngắt port nối tiếp.
SETB	TR1	;Cho Timer 1 hoạt động.
BACK:		
MOV	A, P1	;Đọc dữ liệu từ P1.
MOV	P2, A	;Xuất dữ liệu nhận được ra P2.
SJMP	BACK	;Lặp lại các thao tác trên.
SPISR:		;ISR của ngắt port nối tiếp.
CLR	TI	;Xóa cờ ngắt phát TI.
MOV	SBUF, A	;Xuất dữ liệu nhận được ra port nối tiếp.

RETI ;Kết thúc ISR của port nối tiếp.
END ;Kết thúc chương trình.

Ví dụ 3: Viết chương trình điều khiển 8051 đọc dữ liệu từ cổng P1 và ghi liên tục tới cổng P2. Trong khi đó dữ liệu nhận được từ port nối tiếp thì được gửi đến cổng P0. Biết rằng tốc độ truyền là 9600 baud và $f_{Osc} = 11,0592\text{MHz}$

Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	0023H	;Điểm nhập ISR port nối tiếp.
LJMP	SPISR	;Lệnh nhảy đến ISR port nối tiếp.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	P1, #0FFH	;Cấu hình Port 1 là cổng vào.
MOV	SCON, #52H	;Chọn chế độ 1 cho port nối tiếp.
MOV	TMOD, #20H	;Chọn chế độ 2 cho Timer 1.
MOV	TH1, #(-3)	;Tốc độ baud = 9600.
MOV	IE, #90H	;Cho phép ngắt port nối tiếp.
SETB	TR1	;Cho Timer 1 hoạt động.
BACK:		
MOV	A, P1	;Đọc dữ liệu từ P1.
MOV	P2, A	;Xuất dữ liệu nhận được ra P2.
SJMP	BACK	;Lặp lại các thao tác trên.
SPISR:		;ISR của ngắt port nối tiếp.
CLR	RI	;Xóa cờ ngắt thu RI.
MOV	A, SBUF	;Nhận dữ liệu từ port nối tiếp.
MOV	P0, A	;Xuất dữ liệu nhận được ra P0.
RETI		;Kết thúc ISR của port nối tiếp.
END		;Kết thúc chương trình.

Ví dụ 4: Viết chương trình có sử dụng các ngắt để thực hiện các công việc sau:

- Nhận dữ liệu từ port nối tiếp, sau đó thì gửi dữ liệu này đến cổng P0.
- Nhận dữ liệu từ cổng P1, sau đó thì gửi dữ liệu này đến port nối tiếp và cổng P2.
- Sử dụng Timer 0 để tạo sóng vuông có tần số 5KHz tại P0.1.

Biết rằng tốc độ truyền là 4800 baud và $f_{Osc} = 11,0592\text{MHz}$

Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	000BH	;Điểm nhập ISR Timer 0.
CPL	P0.1	;Lấy bù chân P1.0, tạo xung.
RETI		;Kết thúc ISR của port nối tiếp.
ORG	0023H	;Điểm nhập ISR port nối tiếp.
LJMP	SPISR	;Lệnh nhảy đến ISR port nối tiếp.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
MOV	P1, #0FFH	;Cấu hình Port 1 là cổng vào.

MOV	SCON, #52H	;Chọn chế độ 1 cho port nối tiếp.
MOV	TMOD, #22H	;Chọn chế độ 2 cho Timer 0, Timer 1.
MOV	TH1, #(-6)	;Tốc độ baud = 4800.
MOV	TH0, #(-92)	;Định thời 100μs (nửa chu kỳ), f = 5KHz.
MOV	IE, #92H	;Cho phép ngắt port nối tiếp và Timer 0.
SETB	TR1	;Cho Timer 1 hoạt động.
SETB	TR0	;Cho Timer 0 hoạt động.
BACK:		
MOV	A, P1	;Đọc dữ liệu từ P1.
MOV	P2, A	;Xuất dữ liệu nhận được ra P2.
SJMP	BACK	;Lặp lại các thao tác trên.
SPISR:		
JB	TI, TRANS	;Kiểm tra thu xong (RI) hay phát xong (TI), nếu thu xong (RI = 1) thì:
CLR	RI	;Xóa cờ ngắt thu RI.
MOV	A, SBUF	;Nhận dữ liệu từ port nối tiếp.
MOV	P0, A	;Xuất dữ liệu nhận được ra P0.
RETI		;Kết thúc ISR của port nối tiếp.
TRANS:		;Nếu phát xong (TI = 1) thì:
CLR	TI	;Xóa cờ ngắt phát TI.
MOV	SBUF, A	;Xuất dữ liệu nhận được ra port nối tiếp.
RETI		;Kết thúc ISR của port nối tiếp.
END		;Kết thúc chương trình.

3. Ví dụ minh họa xử lý ngắt ngoài:

Ngắt ngoài xảy ra khi có mức thấp hoặc có cạnh âm tác động lên chân INT0\ hoặc chân INT1\ của 8051. Khi một ngắt ngoài được tạo ra, cờ tạo ra ngắt (IE0 và IE1) được xóa bởi phần cứng khi CPU trở đến trình phục vụ ngắt nếu ngắt thuộc loại tác động cạnh âm, còn nếu ngắt thuộc loại tác động mức thấp thì nguyên nhân ngắt ngoài sẽ điều khiển mức của cờ thay vì là phần cứng trên chip.

Ví dụ 1: Viết chương trình sử dụng các ngắt để thiết kế bộ điều khiển lò nung sao cho nhiệt độ trong lò duy trì ở mức $120^{\circ}\text{C} \pm 5^{\circ}\text{C}$.

Giải

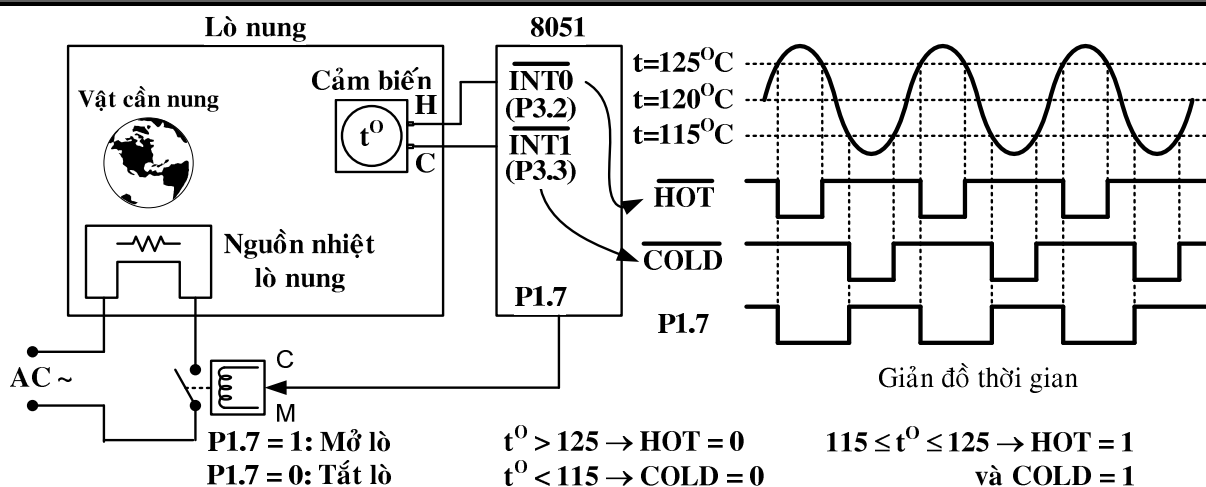
Giả sử ta dùng mạch giao tiếp như hình bên dưới. Cuộn dây điều khiển tắt/mở lò được nối với P1.7 sao cho:

P1.7 = 1 thì mở lò và P1.7 = 0 thì tắt lò

Bộ cảm biến nhiệt độ được nối với INT0\ và INT1\, cung cấp các tín hiệu HOT\ và COLD\ như sau:

HOT\ = 0 nếu $T > 125^{\circ}\text{C}$ và COLD\ = 0 nếu $T < 115^{\circ}\text{C}$

Chương trình sẽ cho lò hoạt động (mở lò) khi $T < 115^{\circ}\text{C}$ và cho lò ngưng hoạt động (tắt lò) khi $T > 125^{\circ}\text{C}$.



```

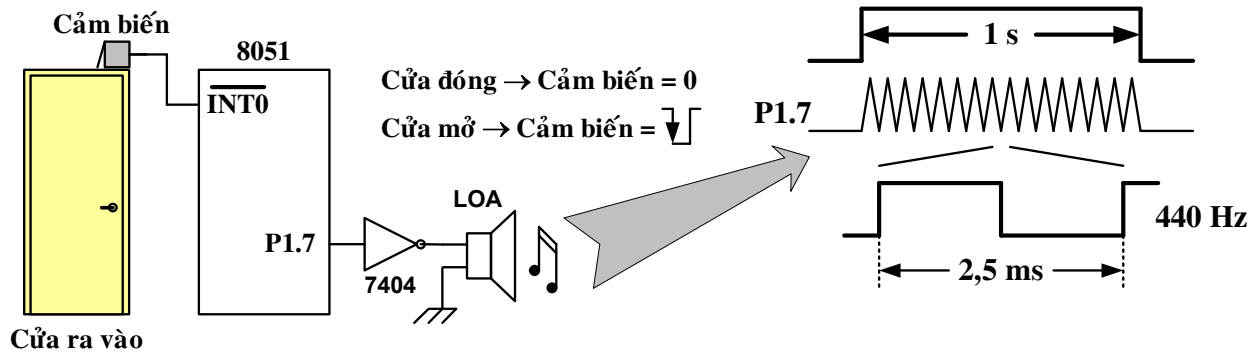
ORG    0000H    ;Điểm nhập reset.
LJMP   MAIN     ;Nhảy qua khỏi các vectơ ngắt.
ORG    0003H    ;Điểm nhập ISR ngắt ngoài 0.
EX0ISR:
CLR    P1.7     ;Tắt lò.
RETI    ;Kết thúc ISR của ngắt ngoài 0.
ORG    0013H    ;Điểm nhập ISR ngắt ngoài 1.
EX1ISR:
SETB   P1.7     ;Mở lò.
RETI    ;Kết thúc ISR của ngắt ngoài 1.
ORG    0030H    ;Điểm nhập chương trình chính.
MAIN:    ;Chương trình chính bắt đầu.
MOV     IE, #85H ;Cho phép ngắt ngoài 0 và 1.
SETB    IT0      ;Ngắt ngoài kích khởi cạnh âm.
SETB    IT1
SETB    P1.7     ;Mở lò.
JB      P3.2, SKIP ;Nếu  $t > 125^{\circ}\text{C}$ .
CLR     P1.7     ;Tắt lò.
SKIP:   SJMP    $ ;Không làm gì (nhảy tại chỗ).
END      ;Kết thúc chương trình.
    
```

Điều khiển tắt/mở lò tùy thuộc trạng thái nhiệt độ hiện tại của lò.

Lưu ý: Lệnh `SJMP $` có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc điều khiển hoạt động tắt mở lò theo nhiệt độ. Vì cứ sau mỗi lần có tín hiệu tác động từ cảm biến thì những công việc đó sẽ bị tạm dừng (ngắt ngoài 0 và 1 xuất hiện) để CPU thực hiện việc điều khiển tắt mở lò theo nhiệt độ qui định rồi quay về thực hiện tiếp những công việc đó.

Ba lệnh đầu tiên trong chương trình chính cho phép các ngắt ngoài và xác định các ngắt ngoài thuộc loại tác động cạnh âm. Do trạng thái hiện tại của các ngõ vào HOT\ và COLD\ chưa biết được nên ba lệnh tiếp theo sẽ điều khiển lò tắt/mở tùy thuộc vào trạng thái nhiệt độ hiện tại của lò. Trước tiên, lò được mở (lệnh `SETB P1.7`) và ngõ vào HOT được lấy mẫu (lệnh `JB P3.2, SKIP`). Nếu ngõ vào HOT ở mức cao, nghĩa là $T < 125^{\circ}\text{C}$, thì lệnh kế được bỏ qua và lò vẫn tiếp tục được mở. Ngược lại, nếu ngõ vào HOT ở mức thấp, nghĩa là $T > 125^{\circ}\text{C}$, thì lệnh kế tiếp `CLR P1.7` được thực thi để tắt lò trước khi đi vào vòng lặp “không làm gì”.

Ví dụ 2: Viết chương trình sử dụng các ngắt để thiết kế một hệ thống báo động tạo âm thanh 400 Hz trong vòng 1 giây (sử dụng một loa được nối với chân P1.7) mỗi khi bộ cảm biến đặt ở cửa (nối với chân INT0) tạo ra một sự chuyển trạng thái từ mức cao xuống mức thấp.



Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vector ngắt.
ORG	0003H	;Điểm nhập ISR ngắt ngoài 0.
LJMP	EX0ISR	;Lệnh nhảy đến ISR ngắt ngoài 0
ORG	000BH	;Điểm nhập ISR của Timer 0.
LJMP	T0ISR	;Lệnh nhảy đến ISR Timer 0.
ORG	001BH	;Điểm nhập ISR của Timer 1.
LJMP	T1ISR	;Lệnh nhảy đến ISR Timer 1.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
SETB	IT0	;Ngắt ngoài kích khởi cạnh âm.
MOV	TMOD, #11H	;Chọn chế độ 1 cho Timer 0, 1.
MOV	IE, #81H	;Cho phép ngắt ngoài 0.
SJMP	\$	;Không làm gì (nhảy tại chỗ).
EX0ISR:		;ISR của ngắt ngoài 0.
MOV	R7, #20	;20 lần x 50000 μ s = 1s.
SETB	TF0	;Buộc Timer 0 ngắt.
SETB	TF1	;Buộc Timer 1 ngắt.
SETB	ET0	;Cho phép ngắt Timer 0.
SETB	ET1	;Cho phép ngắt Timer 1.
RETI		;Kết thúc ISR của ngắt ngoài 0.
T0ISR:		;ISR của ngắt Timer 0.
CLR	TR0	;Dừng Timer 0.
DJNZ	R7, SKIP	;Kiểm tra đủ 20 lần (1 giây).
CLR	ET0	;Kết thúc phát âm nếu đủ.
CLR	ET1	
LJMP	EXIT	
SKIP:		
MOV	TH0, #HIGH(-50000)	;Định thời 0,05 s.
MOV	TL0, #LOW(-50000)	
SETB	TR0	;Cho Timer 0 hoạt động.
EXIT:		
RETI		;Kết thúc ISR của ngắt Timer 0.

T1ISR:		;ISR của ngắt Timer 1.
CLR	TR1	;Dừng Timer 1.
MOV	TH1, #HIGH(-1250)	;Định thời 1,25 ms.
MOV	TL1, #LOW(-1250)	
SETB	TR0	;Cho Timer 1 hoạt động.
RETI		;Kết thúc ISR của ngắt Timer 1.
END		;Kết thúc chương trình.

 **Lưu ý:** Lệnh **SJMP \$** có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc điều khiển hoạt động của loa phát ra âm thanh theo tín hiệu từ bộ cảm biến cửa mở. Vì cứ sau mỗi lần có tín hiệu tác động từ cảm biến thì những công việc đó sẽ bị tạm dừng (*ngắt ngoài 0 xuất hiện*) để CPU thực hiện việc điều khiển phát ra âm thanh tại loa trong một khoảng thời gian xác định rồi quay về thực hiện tiếp những công việc đó.

Giải pháp cho ví dụ này là sử dụng ba ngắt: ngắt ngoài 0 (*bộ cảm biến cửa*), ngắt Timer 0 (*âm hiệu 400Hz*) và ngắt Timer 1 (*định thời 1s*).

Chương trình gồm năm phần phân biệt: vị trí các vector ngắt, chương trình chính và ba trình phục vụ ngắt. Các vị trí vector ngắt chứa các lệnh **LJMP** để chuyển điều khiển đến các trình phục vụ ngắt tương ứng. Chương trình chính bắt đầu ở địa chỉ 0030H chỉ chứa bốn lệnh. Lệnh **SETB IT0** cho phép ngõ vào ngắt ghép với bộ cảm biến cửa được kích khởi cạnh âm.

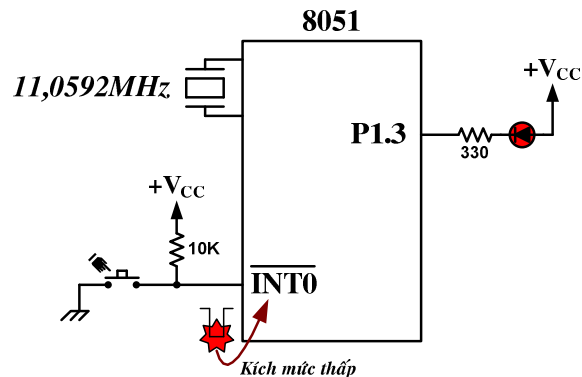
Lệnh **MOV TMOD, #11H** xác định chế độ hoạt động của cả hai bộ định thời là chế độ định thời 16 bit. Chỉ có ngắt ngoài 0 được phép bắt đầu (*lệnh MOV IE, #81H*) do điều kiện cửa mở là điều kiện cần phải có trước khi một ngắt nào đó được chấp thuận. Cuối cùng lệnh **SJMP \$** đặt chương trình vào vòng lặp “không làm gì”. Khi điều kiện cửa mở được phát hiện (*bằng sự chuyển trạng thái từ mức cao xuống mức thấp ở chân INT0*), ngắt ngoài 0 được tạo ra.

Trình phục vụ cho ngắt ngoài 0, EX0ISR, bắt đầu bằng việc nạp hằng số 20 cho R7, rồi set cờ tràn của cả hai bộ định thời bằng 1 để buộc các ngắt do bộ định thời xuất hiện. Tuy nhiên các ngắt do bộ định thời sẽ chỉ xuất hiện khi các bit tương ứng trong thanh ghi IE được cho phép. Hai lệnh kế tiếp **SETB ET0** và **SETB ET1** cho phép các ngắt do bộ định thời. Cuối cùng trình phục vụ cho ngắt ngoài 0, EX0ISR, kết thúc bằng lệnh **RETI** để trở về chương trình chính.

Bộ định thời 0 dùng để tạo ra khoảng thời gian định thời 1s và bộ định thời 1 dùng để tạo ra âm hiệu 400Hz. Sau khi trình phục vụ cho ngắt ngoài 0, EX0ISR, kết thúc thì các ngắt do bộ định thời lập tức được tạo ra (*và được chấp nhận sau khi thực thi một lệnh SJMP \$*). Dựa vào thứ tự trong chuỗi vòng nên ngắt do bộ định thời 0 sẽ được phục vụ trước tiên. Khoảng thời gian định thời 1s được tạo ra bằng cách lập trình để lặp lại 20 lần thời gian định thời 50000μs ($20 \times 50000\mu s = 1s$). Thanh ghi R7 hoạt động như là một bộ đếm.

Trình phục vụ ngắt T0ISR hoạt động như sau: trước tiên bộ định thời 0 được điều khiển ngừng hoạt động và R7 được giảm đi một đơn vị. Kế đến TH0/TL0 được nạp lại bởi giá trị (-50000), sau đó bộ định thời 0 được điều khiển hoạt động trở lại và trình phục vụ ngắt được kết thúc. Ở lần ngắt thứ 20, R7 được giảm xuống 0 (*1s đã trôi qua*). Các ngắt do cả hai bộ định thời được vô hiệu (*lệnh CLR ET0 và CLR ET1*) và trình phục vụ ngắt kết thúc. Không còn ngắt do bộ định thời tạo ra nữa cho đến khi điều kiện cửa mở xuất hiện một lần nữa. Âm hiệu 400Hz được lập trình bằng cách sử dụng ngắt do bộ định thời 1. Tần số 400Hz yêu cầu chu kỳ là 2500μs, trong đó có 1250μs ở mức cao và 1250μs ở mức thấp. Trình phục vụ ngắt cho bộ định thời 1 chỉ đơn giản nạp giá trị (-1250) cho TH1/TL1, sau đó lấy bù bit của port để kích loa và rồi kết thúc trình phục vụ ngắt.

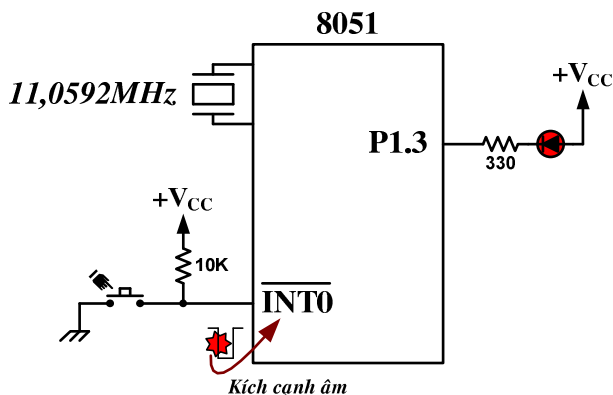
Ví dụ 3: Ngắt ngoài kích khởi mức thấp - Cho mạch điện như hình vẽ. Chân INT0\ được nối với một công tắc bình thường ở mức cao, mỗi khi chân này có mức thấp (nhấn công tắc) thì điều khiển bật LED (bình thường thì LED tắt). Khi LED đã được bật thì phải sáng trong một khoảng thời gian (vài trăm μs) trước khi tắt, khi công tắc được nhấn và giữ thì LED phải sáng liên tục.



Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vectơ ngắt.
ORG	0003H	;Điểm nhập ISR ngắt ngoài 0.
CLR	P1.3	;Bật LED.
MOV	R3, #255	;Thời gian LED duy trì trạng thái sáng là
DJNZ	R3, \$	;255.T _{Machine} .
SETB	P1.3	;Tắt LED.
RETI		;Kết thúc ISR của ngắt ngoài 0.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
CLR	TCON.0	;Ngắt ngoài 0 kích khởi mức thấp.
MOV	IE, #81H	;Cho phép ngắt ngoài 0.
SJMP	\$	;Không làm gì (<i>nhảy tại chỗ</i>).
END		;Kết thúc chương trình

Ví dụ 4: Ngắt ngoài kích khởi cạnh âm - Cho mạch điện như hình vẽ. Chân INT0\ được nối với một công tắc bình thường ở mức cao, mỗi khi chân này có sự chuyển trạng thái từ mức cao xuống mức thấp (nhấn công tắc) thì điều khiển bật LED (bình thường thì LED tắt). Khi LED đã được bật thì phải sáng trong một khoảng thời gian (vài trăm μs) trước khi tắt, khi công tắc được nhấn và giữ thì LED không được sáng liên tục.



Giải

ORG	0000H	;Điểm nhập reset.
LJMP	MAIN	;Nhảy qua khỏi các vectơ ngắt.
ORG	0003H	;Điểm nhập ISR ngắt ngoài 0.
CLR	P1.3	;Bật LED.
MOV	R3, #255	;Thời gian LED duy trì trạng thái sáng là
DJNZ	R3, \$	;255.T _{Machine} .
SETB	P1.3	;Tắt LED.
RETI		;Kết thúc ISR của ngắt ngoài 0.
ORG	0030H	;Điểm nhập chương trình chính.
MAIN:		;Chương trình chính bắt đầu.
SETB	TCON.0	;Ngắt ngoài 0 kích khởi cạnh âm.
MOV	IE, #81H	;Cho phép ngắt ngoài 0.
SJMP	\$	;Không làm gì (nhảy tại chỗ).
END		;Kết thúc chương trình

VII. PHẦN BÀI TẬP:

Bài 1: Viết đoạn lệnh dùng ngắt Timer để tạo sóng vuông $f=2KHz$ tại P1.7. ($f_{osc}=12MHz$).

Bài 2: Viết đoạn lệnh dùng ngắt Timer để tạo sóng vuông $f=200Hz$ tại P1.6. ($f_{osc}=12MHz$).

Bài 3: Viết đoạn lệnh dùng ngắt Timer để tạo đồng thời hai sóng vuông 1KHz và 50Hz tại P1.0 và P1.1. ($f_{osc}=6MHz$)

Bài 4: Viết đoạn lệnh lấy một chuỗi data chứa trong Ram ngoài bắt đầu từ địa chỉ 6200H đến địa chỉ 62FFH và xuất ra Port 1, mỗi lần xuất cách nhau 50ms. Sử dụng ngắt Timer. $f_{osc}=12MHz$.

Bài 5: Viết đoạn lệnh nhập data từ thiết bị ngoài kết nối với 8051 qua Port 1, mỗi lần nhập cách nhau 5s, data nhập về được ghi vào vùng Ram nội bắt đầu từ địa chỉ 50H đến địa chỉ 5FH. Biết rằng sau khi ghi vào ô nhớ cuối cùng thì trở lại ghi vào ô nhớ đầu. Sử dụng ngắt Timer. $f_{osc}=12MHz$.

Bài 6: Viết đoạn lệnh phát liên tục chuỗi số từ 0 đến 9 ra port nối tiếp theo chế độ UART 8 bit, 2400 baud. Sử dụng ngắt serial. $f_{osc}=12MHz$.

Bài 7: Viết đoạn lệnh chờ nhận data từ một thiết bị ngoài gửi đến 8051 qua port nối tiếp (chế độ UART 8 bit, 19200 baud). Nếu nhận được ký tự STX (02H) thì bật sáng LED, nếu nhận được ký tự ETX (03H) thì tắt LED, biết rằng LED được điều khiển bằng ngõ P1.3 (LED sáng khi bit điều khiển bằng 1). Sử dụng ngắt serial. $f_{OSC}=11,059\text{MHz}$.

Bài 8: Viết đoạn lệnh chờ nhận 1 xung cạnh xuống đưa vào chân /INT0 (P3.2), khi có xung thì nhập data từ Port 1 và phát ra port nối tiếp ở chế độ UART 9 bit 4800 baud, bit thứ 9 là bit parity lẻ. $f_{OSC}=6\text{MHz}$.

Bài 9: Viết đoạn lệnh đếm số xung đưa vào chân /INT1 (P3.3) và điều khiển relay thông qua chân P3.0 (relay đóng khi P3.0 bằng 1), cất số đếm vào ô nhớ 40H của Ram nội, nếu số đếm chưa đến 100 thì đóng relay, nếu số đếm đạt 100 thì ngắt relay.