BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH

VI XỬ LÝ

BIÊN SOẠN: PHẠM QUANG TRÍ

TP. HỒ CHÍ MINH
6 - 2006

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

CHƯƠNG 1

GIỚI THIỆU CHUNG VỀ BỘ VI XỬ LÝ

CHƯƠNG 1
GIỚI THIỆU CHUNG VỀ BỘ VI XỬ LÝ

I. SỰ PHÁT TRIỂN CỦA CÁC BỘ VI XỬ LÝ:

1. Thế hệ 1 (*1971 - 1973*):

Đặc điểm chung của các vi xử lý thế hệ này:
- Bus dữ liệu: 4 bit.
- Bus địa chỉ: 12 bit.
- Công nghệ chế tạo: PMOS.
- Tốc độ thực hiện lệnh: 10 – 60 µs/lệnh với f_{CLOCK} = 0,1 – 0,8 MHz.

Một số bộ vi xử lý đặc trưng cho thế hệ này: 4040 (*Intel*), PPS-4 (*Rockwell International*), …

2. Thế hệ 2 (*1974 - 1977*):

Đặc điểm chung của các vi xử lý thế hệ này:
- Bus dữ liệu: 8 bit.
- Bus địa chỉ: 16 bit.
- Công nghệ chế tạo: NMOS hoặc CMOS.
- Tốc độ thực hiện lệnh: 1 – 8 µs/lệnh với f_{CLOCK} = 1 – 5 MHz.

Một số bộ vi xử lý đặc trưng cho thế hệ này: 6502 (*Mos Technology*), 6800/6809 (*Motorola*), 8080/8085 (*Intel*), Z80 (*Zilog*), …

3. Thế hệ 3 (*1978 - 1982*):

Đặc điểm chung của các vi xử lý thế hệ này:
- Bus dữ liệu: 16 bit.
- Bus địa chỉ: 20 - 24 bit.
- Công nghệ chế tạo: HMOS.
- Tốc độ thực hiện lệnh: 0,1 – 1 µs/lệnh với f_{CLOCK} = 5 – 10 MHz.

Một số bộ vi xử lý đặc trưng cho thế hệ này: 68000 / 68010 (Motorola), 8086 / 80186 / 80286 (Intel), …

4. Thế hệ 4 (*1983 - nay*):

Đặc điểm chung của các vi xử lý thế hệ này:
- Bus dữ liệu: 32 - 64 bit.
- Bus địa chỉ: 32 bit.
- Công nghệ chế tạo: HCMOS.
- Tốc độ thực hiện lệnh: 0,01 – 0,1 µs với f_{CLOCK} = 20 – 100 MHz.

Một số bộ vi xử lý đặc trưng cho thế hệ này: 68020 / 68030 / 68040 / 68060 (*Motorola*), 80386 / 80486 / Pentium (*Intel*), …

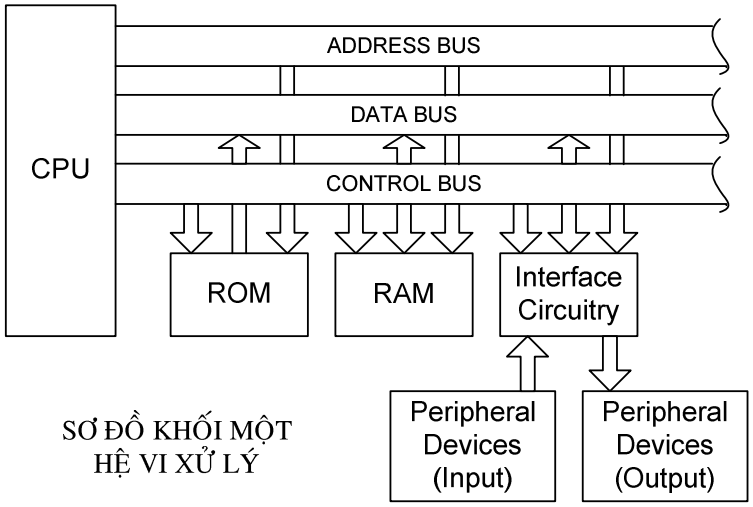
II. SƠ ĐỒ KHỐI CỦA MỘT HỆ VI XỬ LÝ:

Định nghĩa hệ vi xử lý:
- Khả năng được lập trình để thao tác trên các dữ liệu mà không cần sự can thiệp của con người.

- Khả năng lưu trữ và phục hồi dữ liệu.

Tổng quát, hệ vi xử lý gồm:

- Phần cứng (*Hardware*): các thiết bị ngoại vi → để giao tiếp với con người.
- Phần mềm (*Software*): chương trình → để xử lý dữ liệu.



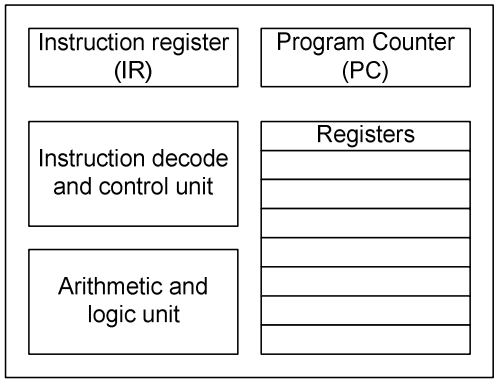
SƠ ĐỒ KHỐI MỘT HỆ VI XỬ LÝ

CPU (*Central Processing Unit*): đơn vị xử lý trung tâm.
RAM (*Random Access Memory*): bộ nhớ truy xuất ngẫu nhiên.
ROM (*Read Only Memory*): bộ nhớ chỉ đọc.
Interface Circuitry: mạch điện giao tiếp.
Peripheral Devices (*Input*): các thiết bị ngoại vi (*thiết bị nhập*).
Peripheral Devices (*Output*): các thiết bị ngoại vi (*thiết bị xuất*).
Address bus: bus địa chỉ.
Data bus: bus dữ liệu.
Control bus: bus điều khiển.

III. ĐƠN VỊ XỬ LÝ TRUNG TÂM:

CPU đóng vai trò chủ đạo trong hệ vi xử lý, nó quản lý tất cả các hoạt động của hệ và thực hiện tất cả các thao tác trên dữ liệu.

CPU là một vi mạch điện tử có độ tích hợp cao. Khi hoạt động, CPU ***đọc mã lệnh*** được ghi dưới dạng các bit 0 và bit 1 từ bộ nhớ, sau đó nó sẽ thực hiện ***giải mã*** các lệnh này thành dãy các xung điều khiển tương ứng với các thao tác trong lệnh để điều khiển các khối khác ***thực hiện*** từng bước các thao tác đó và từ đó tạo ra các xung điều khiển cho toàn hệ.

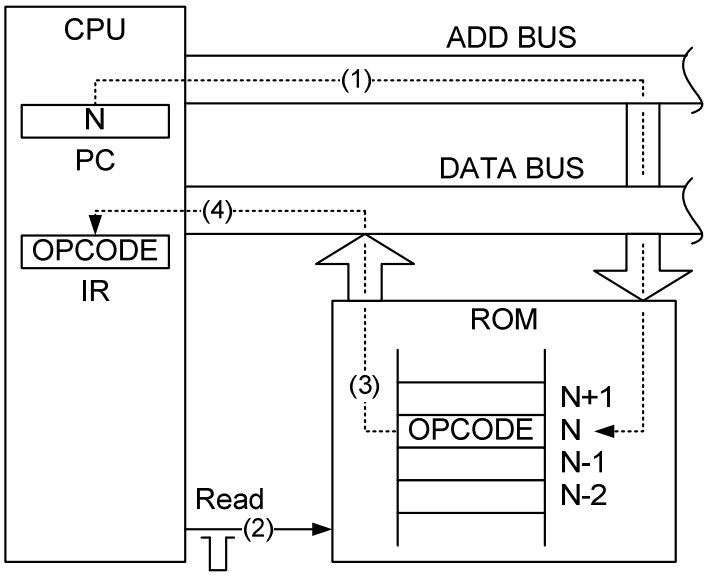


SƠ ĐỒ KHỐI ĐƠN GIẢN CỦA MỘT CPU

IR (*Instruction Register*): thanh ghi lệnh.
PC (*Program Counter / Instruction Pointer*): bộ đếm chương trình / con trỏ lệnh.
Instruction decode and control unit: đơn vị giải mã lệnh và điều khiển.
ALU (*Arithmetic and Logic Unit*): đơn vị số học và logic.
Registers: các thanh ghi.

Tóm lại, khi hoạt động CPU sẽ thực hiện liên tục 2 thao tác: ***tìm nạp lệnh*** và ***giải mã – thực hiện lệnh***.

- Thao tác ***tìm nạp lệnh***:
  - Nội dung của thanh ghi PC được CPU đưa lên bus địa chỉ (1).
  - Tín hiệu điều khiển đọc (*Read*) chuyển sang trạng thái tích cực (2).
  - Mã lệnh (*Opcode*) từ bộ nhớ được đưa lên bus dữ liệu (3).
  - Mã lệnh được chuyển vào trong thanh ghi IR trong CPU (4).
  - Nội dung của thanh ghi PC tăng lên một đơn vị để chuẩn bị tìm nạp lệnh kế tiếp từ bộ nhớ.



HOẠT ĐỘNG CỦA BUS CHO CHU KỲ TÌM NẠP LỆNH

- Thao tác ***giải mã – thực hiện lệnh***:
  - Mã lệnh từ thanh ghi IR được đưa vào đơn vị giải mã lệnh và điều khiển.
  - Đơn vị giải mã lệnh và điều khiển sẽ thực hiện giải mã opcode và tạo ra các tín hiệu để điều khiển việc xuất nhập dữ liệu giữa ALU và các thanh ghi (*Registers*).
  - Căn cứ trên các tín hiệu điều khiển này, ALU thực hiện các thao tác đã được xác định.

Một chuỗi các lệnh (*Opcode*) kết hợp lại với nhau để thực hiện một công việc có ý nghĩa được gọi là chương trình (*Program*) hay phần mềm (*Software*).

IV. BỘ NHỚ BÁN DẪN:

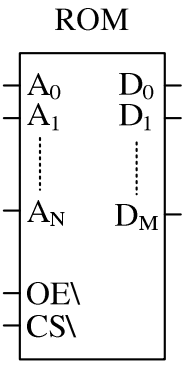
Bộ nhớ bán dẫn là một bộ phận khác rất quan trọng của hệ vi xử lý, các chương trình và dữ liệu đều được lưu giữ trong bộ nhớ.

Bộ nhớ bán dẫn trong hệ vi xử lý gồm:

- ROM: bộ nhớ chương trình → lưu giữ chương trình điều khiển hoạt động của toàn hệ thống.
- RAM: bộ nhớ dữ liệu → lưu giữ dữ liệu, một phần chương trình điều khiển hệ thống, các ứng dụng và kết quả tính toán.

Sơ lược về cấu trúc và phân loại ROM – RAM:

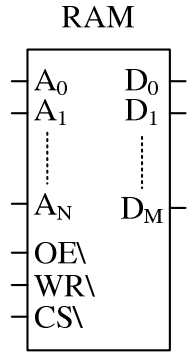
- ROM (*Read Only Memory*): bộ nhớ chỉ đọc, thông tin trong ROM sẽ không bị mất đi ngay cả khi nguồn điện cung cấp cho ROM không còn.
  - Cấu trúc ROM:



A_0 .. A_N: các chân địa chỉ (Address)
D_0 .. D_M: các chân dữ liệu (Data)
OE\ (Output Enable): ngõ vào cho phép xuất data
CS\ (Chip Select): ngõ vào cho phép ROM hoạt động

  - Phân loại một số loại ROM:
    - MROM (*Mask ROM*): ROM mặt nạ.
    - PROM (*Programmable ROM*): ROM lập trình được.
    - EPROM (*Eraseable PROM*): ROM lập trình và xóa được.
      - UV-EPROM (*Ultra Violet EPROM*): ROM xóa bằng tia cực tím.
      - EEPROM (*Electric EPROM*): ROM lập trình và xóa bằng tín hiệu điện.
      - Flash ROM: ROM lập trình và xóa bằng tín hiệu điện.

- RAM (*Random Access Memory*): bộ nhớ truy xuất ngẫu nhiên (*bộ nhớ ghi đọc*), thông tin trong RAM sẽ bị mất đi khi nguồn điện cung cấp cho RAM không còn..
  - Cấu trúc RAM:



A_0 .. A_N: các chân địa chỉ (Address)
D_0 .. D_M: các chân dữ liệu (Data)
OE\ (Output Enable): ngõ vào cho phép xuất dữ liệu
WR\ (Write): ngõ vào cho phép ghi dữ liệu
CS\ (Chip Select): ngõ vào cho phép RAM hoạt động

  - Phân loại một số loại RAM:
    - DRAM (*Dynamic RAM*): RAM động
    - SRAM (*Static RAM*): RAM tĩnh

Cách xác định dung lượng bộ nhớ bán dẫn 8 bit sử dụng cho chip vi điều khiển 8051 như sau:

- Dựa vào số lượng chân địa chỉ:

Dung lượng = 2^N, với N là số đường địa chỉ của bộ nhớ.

Ví dụ: Bộ nhớ bán dẫn 8 bit có 10 đường địa chỉ. Cho biết dung lượng của bộ nhớ là bao nhiêu?

N = 10 → Dung lượng = 2^{10} = **1024** = **1 KB**

- Dựa vào mã số của bộ nhớ:

Mã số: **XX YYYY**	**XX**: loại bộ nhớ 27: UV-EPROM 28: EEPROM 61,62: SRAM 40,41: DRAM **YYYY**: dung lượng bộ nhớ
Dung lượng = YYYY (Kbit) hoặc **Dung lượng = YYYY / 8 (KB)**	

Ví dụ: Bộ nhớ có mã số 27256, dung lượng của bộ nhớ là bao nhiêu ?

27 → Bộ nhớ **UV-EPROM**

256 → Dung lượng = **256 (Kbit)** = **32 (KB)**

V. CÁC THIẾT BỊ NGOẠI VI (CÁC THIẾT BỊ XUẤT NHẬP):

Mạch điện giao tiếp (*Interface Circuitry*) và các thiết bị xuất nhập hay thiết bị ngoại vi (*Peripheral Devices*) tạo ra khả năng giao tiếp giữa hệ vi xử lý với thế giới bên ngoài. Bộ phận giao tiếp giữa bus hệ thống của hệ vi xử lý với các thế giới bên ngoài thường được gọi là cổng (*Port*). Như vậy tùy theo từng loại thiết bị giao tiếp mà ta có các cổng nhập (*Input*) để lấy thông tin từ ngoài vào hệ và các cổng xuất (*Output*) để đưa thông tin từ trong hệ ra ngoài.

Tổng quát, ta có 3 loại thiết bị xuất nhập sau:

- Thiết bị lưu trữ lớn: băng từ, đĩa từ, đĩa quang, …
- Thiết bị giao tiếp với con người: màn hình, bàn phím, máy in, …
- Thiết bị điều khiển / kiểm tra: các bộ kích thích, các bộ cảm biến, …

VI. HỆ THỐNG BUS:

Bus là tập hợp các đường dây mang thông tin có cùng chức năng. Việc truy xuất thông tin tới một mạch điện xung quanh CPU thì nó sử dụng 3 loại bus: bus địa chỉ, bus dữ liệu và bus điều khiển. CPU sử dụng hệ thống bus này để thực hiện các thao tác đọc (*READ*) và ghi (*WRITE*) thông tin giữa CPU với bộ nhớ hoặc các thiết bị ngoại vi.

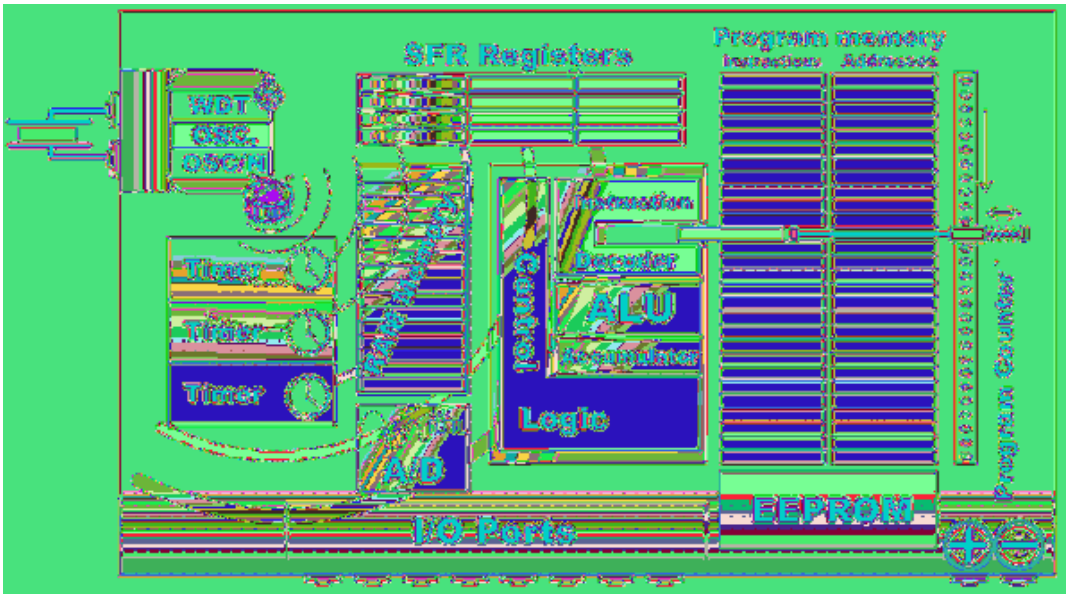
- Bus địa chỉ (*Address bus*):
  - Để chuyển tải thông tin của các bit địa chỉ.
  - Là loại bus 1 chiều (*CPU → MEM hay I/O*).
  - Để xác định bộ nhớ hoặc thiết bị ngoại vi mà CPU cần trao đổi thông tin.
  - Để xác định dung lượng bộ nhớ hoặc ngoại vi mà CPU có khả năng truy xuất.
- Bus dữ liệu (*Data bus*):
  - Để chuyển tải thông tin của các bit dữ liệu.
  - Là loại bus 2 chiều (*CPU ↔ MEM hay I/O*).
  - Để xác định số bit dữ liệu mà CPU có khả năng xử lý cùng một lúc.
- Bus điều khiển (*Control bus*):
  - Để chuyển tải thông tin của các bit điều khiển (mỗi đường dây là một tín hiệu điều khiển khác nhau).
  - Là loại bus 1 chiều (*CPU → MEM-I/O hoặc MEM-I/O → CPU*).
  - Để điều khiển các khối khác trong hệ và nhận tín hiệu điều khiển từ các khối đó để phối hợp hoạt động.

VII. VI XỬ LÝ – VI ĐIỀU KHIỂN:

Để phân biệt bộ vi xử lý và bộ vi điều khiển ta có thể dựa trên các yếu tố như sau:

Yếu tố phân loại		**Vi xử lý (*Microprocessor*)**	**Vi điều khiển (*Microcontroller*)**
Cấu trúc phần cứng (*Hardware architecture*)	CPU	X	X
	ROM		X
	RAM		X
	Mạch giao tiếp nối tiếp		X
	Mạch giao tiếp song song		X
	Mạch điều khiển ngắt		X
	Các mạch điều khiển khác		X
Các ứngdụng (*Applications*)	Ứng dụng lớn, tính toán phức tạp	X	
	Ứng dụng nhỏ, tính toán đơn giản		X
Các đặc trưng của tập lệnh (*Instruction set feature*)	Các kiểu định địa chỉ	Nhiều	Ít
	Độ dài từ dữ liệu xử lý	Byte, Word, Double word, …	Bit, Byte

VIII. MINH HỌA KIẾN TRÚC CỦA MỘT HỆ VI ĐIỀU KHIỂN:



WDT (*Watch-Dog Timer*): Bộ định thời Watch-Dog.
OSC., **OSC/N** (*Oscillator*): Bộ dao động (N: hệ số chia tần).
Timer: Bộ định thời.
A/D (*Analog/Digital*): Bộ biến đổi tín hiệu tương tự/số.
SFR Registers (*Special Function Register*): Các thanh ghi chức năng đặc biệt.
RAM Memory: Bộ nhớ dữ liệu.

Program Memory: Bộ nhớ chương trình.
EEPROM: Bộ nhớ EEPROM.
I/O Ports: Các port xuất/nhập.
Instruction Decoder: Bộ giải mã lệnh.
ALU: Đơn vị logic và số học.
Accumulator: Thanh ghi tích lũy.
Control Logic: Điều khiển logic.
Program Counter: Bộ đếm chương trình.
Instructions/Addresses: Các lệnh / địa chỉ.

IX. LỰA CHỌN BỘ VI ĐIỀU KHIỂN KHI THIẾT KẾ:

Có bốn họ vi điều khiển thông dụng trên thị trường hiện nay là: 68xxx của Motorola, 80xxx của Intel, Z8xx của Zilog và PIC16xxx của Microchip Technology. Mỗi loại vi điều khiển trên đều có một tập lệnh và thanh ghi riêng nên chúng không tương thích lẫn nhau. Vậy khi ta tiến hành thiết kế một hệ thống sử dụng vi điều khiển thì ta cần dựa trên những tiêu chuẩn nào? Có ba tiêu chẩn chính:

- ***Tiêu chuẩn thứ nhất là:*** Đáp ứng yêu cầu tính toán một cách hiệu quả và kinh tế. Do vậy, trước tiên ta cần phải xem xét bộ vi điều khiển 8 bit, 16 bit hay 32 bit là thích hợp nhất. Một số tham số kỹ thuật cần được cân nhắc khi chọn lựa là:
  - ***Tốc độ:*** tốc độ lớn nhất mà vi điều khiển hỗ trợ là bao nhiêu.
  - ***Kiểu IC:*** là kiểu 40 chân DIP, QFP hay là kiểu đóng vỏ khác (*DIP: vỏ dạng hai hàng chân, QFP: vỏ vuông dẹt*). Kiểu đóng vỏ rất quan trọng khi có yêu cầu về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.
  - ***Công suất tiêu thụ:*** là một tiêu chuẩn cần đặc biệt lưu ý nếu sản phẩm dùng pin hoặc điện áp lưới.
  - Dung lượng bộ nhớ ROM và RAM tích hợp sẵn trên chip.
  - Số chân vào/ra và bộ định thời trên chip.
  - Khả năng dễ dàng nâng cao hiệu suất hoặc giãm công suất tiêu thụ.
  - Giá thành trên một đơn vị khi mua số lượng lớn. Vì đây là vấn đề có ảnh hưởng đến giá thành cuối cùng của sản phẩm.
- ***Tiêu chuẩn thứ hai là:*** Có sẵn các công cụ phát triển phần mềm, chẳng hạn như các chương trình mô phỏng, trình biên dịch, trình hợp dịch và gỡ rối.
- ***Tiêu chuẩn thứ ba là:*** Khả năng đáp ứng về số lượng ở hiện tại cũng như ở tương lai. Đối với một số nhà thiết kế thì tiêu chuẩn này thậm chí còn quan trọng hơn cả hai tiêu chuẩn trên.

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

CHƯƠNG 2

PHẦN CỨNG CHIP VI ĐIỀU KHIỂN 8051

CHƯƠNG 2
PHẦN CỨNG CHIP VI ĐIỀU KHIỂN 8051

I. TỔNG QUÁT:

1. Giới thiệu chung:

MCS-51 là họ vi điều khiển của hãng Intel. Vi mạch tổng quát của họ MCS-51 là chip 8051. Chip 8051 có một số đặc trưng cơ bản sau:

- Bộ nhớ chương trình bên trong: 4 KB (*ROM*).
- Bộ nhớ dữ liệu bên trong: 128 byte (*RAM*).
- Bộ nhớ chương trình bên ngoài: 64 KB (*ROM*).
- Bộ nhớ dữ liệu bên ngoài: 64 KB (*RAM*).
- 4 port xuất nhập (*I/O port*) 8 bit.
- 2 bộ định thời 16 bit.
- Mạch giao tiếp nối tiếp.
- Bộ xử lý bit (*thao tác trên các bit riêng lẻ*).
- 210 vị trí nhớ được định địa chỉ, mỗi vị trí 1 bit.
- Nhân / Chia trong 4 μs.

Ngoài ra, trong họ MCS-51 còn có một số chip vi điều khiển khác có cấu trúc tương đương như:

Chip	ROM trong	RAM trong	Bộ định thời
8031	0 KB	128 byte	2
8032	0 KB	256 byte	3
8051	4 KB PROM	128 byte	2
8052	8 KB PROM	256 byte	3
8751	4 KB UV-EPROM	128 byte	2
8752	8 KB UV-EPROM	256 byte	3
8951	4 KB FLASH ROM	128 byte	2
8952	8 KB FLASH ROM	256 byte	3

2. Các phiên bản của chip vi điều khiển 8051:

2.1 Bộ vi điều khiển 8031:

8031 là một phiên bản khác của họ 8051. Chip này thường được coi là 8051 không có ROM trên chip. Để có thể dùng được chip này cần phải bổ sung thêm ROM ngoài chứa chương trình cần thiết cho 8031. 8051 có chương trình được chứa ở ROM trên chip bị giới hạn đến 4KB, còn ROM ngoài của 8031 thì có thể lên đến 64KB. Tuy nhiên, để có thể truy cập hết bộ nhớ ROM ngoài thì cần dùng thêm hai cổng (*Port 0 và Port 2*) , do vậy chỉ còn lại có hai cổng (*Port 1 và Port 3*) để sử dụng. Nhằm khắc phục vấn đề này, chúng ta có thể bổ sung thêm cổng vào/ra cho 8031.

2.2 Bộ vi điều khiển 8052:

8052 là một phiên bản của họ 8051. 8052 có tất cả các thông số kỹ thuật của 8051, ngoài ra còn có thêm 128 byte RAM, 4KB ROM và một bộ định thời nữa. Như vậy, 8052 có tổng cộng 256 byte RAM, 8KB ROM và ba bộ định thời.

Đặc tính kỹ thuật	8031	8051	8052
ROM trên chip (KB)	0	4	8
RAM trên chip (byte)	128	128	256
Bộ định thời	2	2	3
Chân vào/ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	5	5	6

Như bảng thông số trên ta thấy 8051 là một trường hợp riêng của 8052. Mọi chương trình viết cho 8051 đều có thể chạy được trên 8052 nhưng điều ngược lại có thể là không đúng.

2.3 Bộ vi điều khiển 8751:

Chip 8751 chỉ có 4KB bộ nhớ UV-EPROM trên chip. Để sử dụng chip này cần phải có thiết bị lập trình PROM và thiết bị xoá UV-EPROM. Do ROM trên chip của 8751 là UV-EPROM, nên cần phải mất khoảng 20 phút để xoá 8751 trước khi được lập trình. Vì đây là quá trình mất nhiều thời gian nên nhiều nhà sản xuất đã cho ra phiên bản Flash ROM và UV-RAM.

2.4 Bộ vi điều khiển AT8951 của Atmel Corporation:

AT8951 là phiên bản 8051 có ROM trên chip là bộ nhớ Flash. Phiên bản này rất thích hợp cho các ứng dụng nhanh vì bộ nhớ Flash có thể được xóa trong vài giây. Dĩ nhiên là để dùng AT8951 cần phải có thiết bị lập trình PROM hỗ trợ bộ nhớ Flash nhưng không cần đến thiết bị xóa ROM vì bộ nhớ Flash được xóa bằng thiết bị lập trình PROM. Để tiện sử dụng, hiện nay hãng Atmel đang nghiên cứu một phiên bản của AT8951 có thể được lập trình qua cổng COM của máy tính PC và như vậy sẽ không cần đến thiết bị lập trình PROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt	Vcc	Số chân IC
AT89C51	4KB	128	32	2	5	5V	40
AT89LV51	4KB	128	32	2	5	3V	40
AT89C1051	1KB	64	15	1	3	3V	20
AT89C2051	2KB	128	15	2	5	3V	20
AT89C52	8KB	256	32	3	6	5V	40
AT89LV52	8KB	256	32	3	6	3V	40

2.5 Bộ vi điều khiển DS5000 của Dallas Semiconductor:

Một phiên bản phổ biến khác nữa của 8051 là DS5000 của hãng Dallas Semiconductor. Bộ nhớ ROM trên chip của DS5000 là NV-RAM. DS5000 có khả năng nạp chương trình vào ROM trên chip trong khi nó vẫn ở trong hệ thống mà không cần phải lấy ra. Cách thực hiện là dùng qua cổng COM

của máy tính PC. Đây là một điểm mạnh rất được ưa chuộng. Ngoài ra, NV-RAM còn có ưu việt là cho phép thay đổi nội dung RAM theo từng byte mà không cần phải xóa hết trước khi lập trình như bộ nhớ EPROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt	Vcc	Số chân IC
DS5000-8	8KB	128	32	2	6	5V	40
DS5000-32	32KB	128	32	2	6	5V	40
DS5000T-8	8KB	128	32	2	6	5V	40
DS5000T-32	32KB	128	32	2	6	5V	40

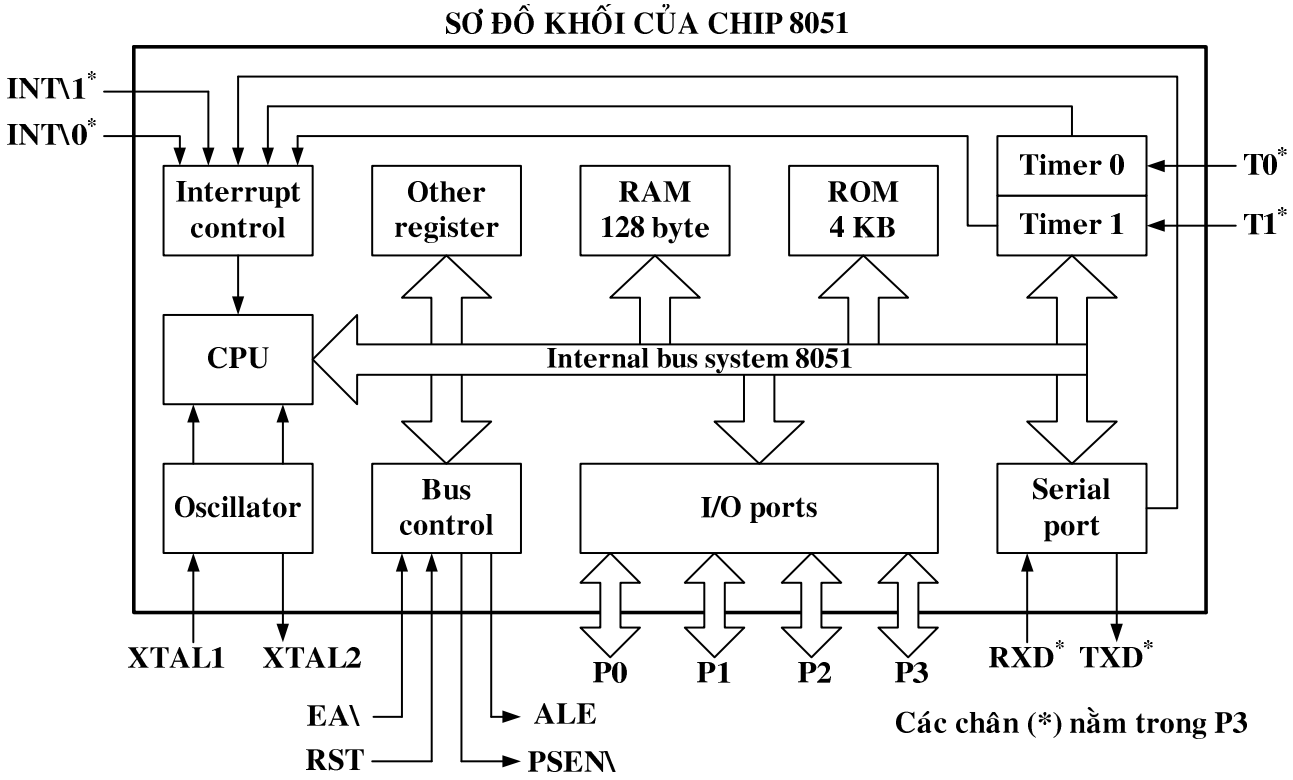
Điểm đặc biệt là các chip có chữ "T" theo sau ký hiệu "5000" có nghĩa là chip đó có thiết kế thêm một đồng hồ thời gian thực (*RTC: Real Time Clock*) bên trong. Lưu ý đồng hồ thời gian thực RTC hoàn toàn khác với bộ định thời Timer. RTC tạo và lưu giữ thời gian của ngày (*giờ/phút/giây*) và ngày tháng (*ngày/tháng/năm*) trên thực tế ngay cả khi không có nguồn cung cấp.

2.6 Bộ vi điều khiển P89V51xx của Philips Corporation:

Đây là một phiên bản cải tiến sử dụng CPU là bộ vi điều khiển 80C51 với nhiều tính năng vượt trội: dung lượng ROM/RAM trên chip rất lớn, 3 Timer 16 bit + 1 Watch-dog Timer, 2 thanh ghi DPTR, 8 nguồn ngắt, PWM (*Pulse Width Modulator*), SPI (*Serial Peripheral Interface*) và đặc biệt là bộ nhớ chương trình trên chip có tính năng ISP (*In-System Programming*) và IAP (*In-Application Programming*),…

II. CÁC CHÂN CỦA CHIP 8051:

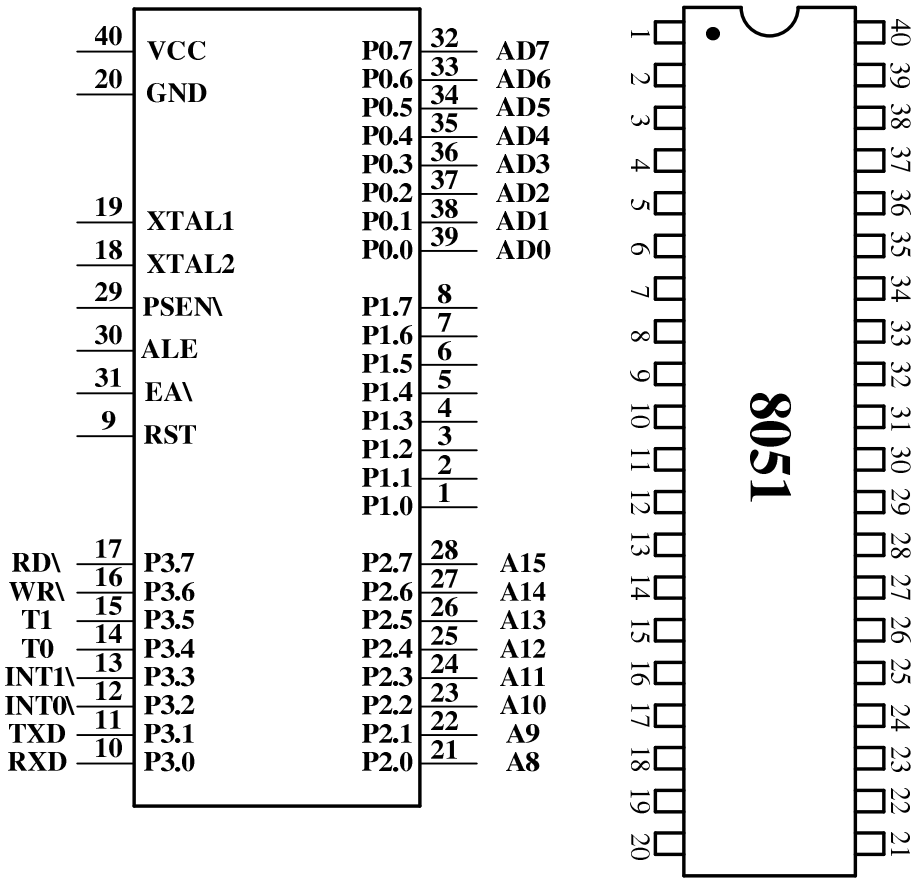
1. Sơ đồ khối và chức năng các khối của chip 8051:



- CPU (***Central Processing Unit***): Đơn vị xử lý trung tâm → tính toán và điều khiển quá trình hoạt động của hệ thống.
- OSC (***Oscillator***): Mạch dao động → tạo tín hiệu xung clock cung cấp cho các khối trong chip hoạt động.
- Interrupt control: Điều khiển ngắt → nhận tín hiệu ngắt từ bên ngoài (*INT0\, INT1*), từ bộ định thời (*Timer 0, Timer 1*) và từ cổng nối tiếp (*Serial port*), lần lượt đưa các tín hiệu ngắt này đến CPU để xử lý.
- Other registers: Các thanh ghi khác → lưu trữ dữ liệu của các port xuất/nhập, trạng thái làm việc của các khối trong chip trong suốt quá trình hoạt động của hệ thống.
- RAM (***Random Access Memory***): Bộ nhớ dữ liệu trong chip → lưu trữ các dữ liệu.
- ROM (***Read Only Memory***): Bộ nhớ chương trình trong chip → lưu trữ chương trình hoạt động của chip.
- I/O ports (***In/Out ports***): Các port xuất/nhập → điều khiển việc xuất nhập dữ liệu dưới dạng song song giữa trong và ngoài chip thông qua các port P0, P1, P2, P3.
- Serial port: Port nối tiếp → điều khiển việc xuất nhập dữ liệu dưới dạng nối tiếp giữa trong và ngoài chip thông qua các chân TxD, RxD.
- Timer 0, Timer 1: Bộ định thời 0, 1 → dùng để định thời gian hoặc đếm sự kiện (*đếm xung*) thông qua các chân T0, T1.
- Bus control: Điều khiển bus → điều khiển hoạt động của hệ thống bus và việc di chuyển thông tin trên hệ thống bus.
- Bus system: Hệ thống bus → liên kết các khối trong chip lại với nhau.

2. Sơ đồ chân và chức năng các chân của chip 8051:

SƠ ĐỒ CHÂN CỦA CHIP 8051



2.1. Port 0:

- Port 0 (*P0.0 – P0.7*) có số chân từ 32 – 39.
- Port 0 có hai chức năng:
  - Port xuất nhập dữ liệu (*P0.0 - P0.7*) → không sử dụng bộ nhớ ngoài.
  - Bus địa chỉ byte thấp và bus dữ liệu đa hợp (*AD0 – AD7*) → có sử dụng bộ nhớ ngoài.

✓ ***Lưu ý: Khi Port 0 đóng vai trò là port xuất nhập dữ liệu thì phải sử dụng các điện trở kéo lên bên ngoài***.

- Ở chế độ mặc định (*khi reset*) thì các chân Port 0 (*P0.0 - P0.7*) được cấu hình là ***port xuất*** dữ liệu. Muốn các chân Port 0 làm ***port nhập*** dữ liệu thì cần phải lập trình lại, bằng cách ghi mức logic cao (*mức 1*) đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port (*vấn đề này được trình bày ở phần kế tiếp*).
- Khi lập trình cho ROM trong chip thì Port 0 đóng vai trò là ngõ vào của dữ liệu (*D0 – D7*) *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.

2.2. Port 1:

- Port 1 (*P1.0 – P1.7*) có số chân từ 1 – 8.
- Port 1 có một chức năng:
  - Port xuất nhập dữ liệu (*P1.0 – P1.7*) → sử dụng hoặc không sử dụng bộ nhớ ngoài.
- Ở chế độ mặc định (*khi reset*) thì các chân Port 1 (*P1.0 – P1.7*) được cấu hình là ***port xuất*** dữ liệu. Muốn các chân Port 1 làm ***port nhập*** dữ liệu thì cần phải lập trình lại, bằng cách ghi mức logic cao (*mức 1*) đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port (*vấn đề này được trình bày ở phần kế tiếp*).
- Khi lập trình cho ROM trong chip thì Port 1 đóng vai trò là ngõ vào của địa chỉ byte thấp (*A0 – A7*) *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.

2.3. Port 2:

- Port 2 (*P2.0 – P2.7*) có số chân từ 21 – 28.
- Port 2 có hai chức năng:
  - Port xuất nhập dữ liệu (*P2.0 – P2.7*) → không sử dụng bộ nhớ ngoài.
  - Bus địa chỉ byte cao (*A8 – A15*) → có sử dụng bộ nhớ ngoài.
- Ở chế độ mặc định (*khi reset*) thì các chân Port 2 (*P2.0 – P2.7*) được cấu hình là ***port xuất*** dữ liệu. Muốn các chân Port 2 làm ***port nhập*** dữ liệu thì cần phải lập trình lại, bằng cách ghi mức logic cao (*mức 1*) đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port (*vấn đề này được trình bày ở phần kế tiếp*).
- Khi lập trình cho ROM trong chip thì Port 2 đóng vai trò là ngõ vào của địa chỉ byte cao (*A8 – A11*) và các tín hiệu điều khiển *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.

2.4. Port 3:

- Port 3 (*P3.0 – P3.7*) có số chân từ 10 – 17.
- Port 3 có hai chức năng:
  - Port xuất nhập dữ liệu (*P3.0 – P3.7*) → không sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.
  - Các tín hiệu điều khiển → có sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.
- Ở chế độ mặc định (*khi reset*) thì các chân Port 3 (*P3.0 – P3.7*) được cấu hình là ***port xuất*** dữ liệu. Muốn các chân Port 3 làm ***port nhập*** dữ liệu thì cần phải lập trình lại, bằng cách ghi mức logic cao (*mức 1*) đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port (*vấn đề này được trình bày ở phần kế tiếp*).

- Khi lập trình cho ROM trong chip thì Port 3 đóng vai trò là ngõ vào của các tín hiệu điều khiển *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.
- Chức năng của các chân Port 3:

Bit	Tên	Địa chỉ bit	Chức năng
P3.0	RxD	B0H	Chân nhận dữ liệu của port nối tiếp.
P3.1	TxD	B1H	Chân phát dữ liệu của port nối tiếp.
P3.2	INT0\	B2H	Ngõ vào ngắt ngoài 0.
P3.3	INT1\	B3H	Ngõ vào ngắt ngoài 1.
P3.4	T0	B4H	Ngõ vào của bộ định thời/đếm 0.
P3.5	T1	B5H	Ngõ vào của bộ định thời/đếm 1.
P3.6	WR\	B6H	Điều khiển ghi vào RAM ngoài.
P3.7	RD\	B7H	Điều khiển đọc từ RAM ngoài.

2.5. Chân PSEN\:

- PSEN (***Program Store Enable***): cho phép bộ nhớ chương trình, chân số 29.
- Chức năng:
  - Là tín hiệu cho phép truy xuất (***đọc***) bộ nhớ chương trình (*ROM*) ngoài.
  - Là tín hiệu xuất, tích cực mức thấp.

  PSEN\ = 0 → trong thời gian CPU tìm - nạp lệnh từ ROM ngoài.

  PSEN\ = 1 → CPU sử dụng ROM trong (*không sử dụng ROM ngoài*).
- Khi sử dụng bộ nhớ chương trình bên ngoài, chân PSEN\ thường được nối với chân OE\ của ROM ngoài để cho phép CPU đọc mã lệnh từ ROM ngoài.

2.6. Chân ALE:

- ALE (***Address Latch Enable***): cho phép chốt địa chỉ, chân số 30.
- Chức năng:
  - Là tín hiệu cho phép chốt địa chỉ để thực hiện việc giải đa hợp cho bus địa chỉ byte thấp và bus dữ liệu đa hợp (*AD0 – AD7*).
  - Là tín hiệu xuất, tích cực mức cao.

  ALE = 0 → trong thời gian bus AD0 - AD7 đóng vai trò là bus D0 - D7.

  ALE = 1 → trong thời gian bus AD0 - AD7 đóng vai trò là bus A0 - A7.
- Khi lập trình cho ROM trong chip thì chân ALE đóng vai trò là ngõ vào của xung lập trình (*PGM*) *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.

✓ ***Lưu ý:*** $\boxed{f_{ALE} = \frac{f_{OSC}}{6}}$ → có thể dùng làm xung clock cho các mạch khác.

f_{ALE} (MHz): tần số xung tại chân ALE.

f_{OSC} (MHz): tần số dao động trên chip (tần số thạch anh).

- Khi lệnh lấy dữ liệu từ RAM ngoài (*MOVX*) được thực hiện thì một xung ALE bị bỏ qua *(xem giản đồ trang 38-39 sách "Họ vi điều khiển 8051")*.

2.7. Chân EA\:

- EA (***External Access***): truy xuất ngoài, chân số 31.
- Chức năng:
  - Là tín hiệu cho phép truy xuất (***sử dụng***) bộ nhớ chương trình (*ROM*) ngoài.

- Là tín hiệu nhập, tích cực mức thấp.

EA\ = 0 → Chip 8051 sử dụng chương trình của ROM ngoài.

EA\ = 1 → Chip 8051 sử dụng chương trình của ROM trong.

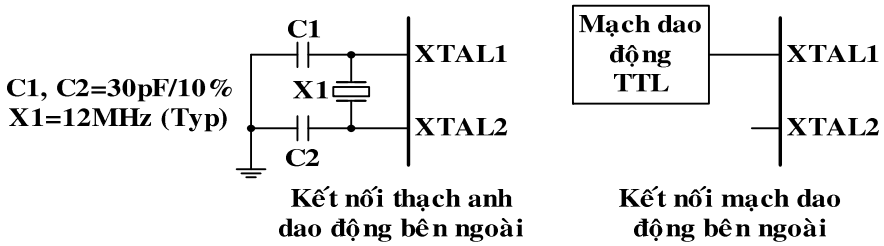
- Khi lập trình cho ROM trong chip thì chân EA đóng vai trò là ngõ vào của điện áp lập trình (*Vpp = 12V – 12,5V cho họ 89xx; 21V cho họ 80xx, 87xx*) *(xem sách "Họ vi điều khiển 8051" trang 333-352)*.

✓ ***Lưu ý:*** Chân EA\ phải được nối lên Vcc (*nếu sử dụng chương trình của ROM trong*) hoặc nối xuống GND (*nếu sử dụng chương trình của ROM ngoài*), không bao giờ được phép bỏ trống chân này.

2.8. Chân XTAL1, XTAL2:

- XTAL (*Crystal*): tinh thể thạch anh, chân số 18-19.
- Chức năng:
  - Dùng để nối với thạch anh hoặc mạch dao động tạo xung clock bên ngoài, cung cấp tín hiệu xung clock cho chip hoạt động.
  - XTAL1 → ngõ vào mạch tạo xung clock trong chip.
  - XTAL2 → ngõ ra mạch tạo xung clock trong chip.

✓ ***Lưu ý:*** $\mathbf{f_{TYP} = 12MHz}$ $\mathbf{f_{TYP}}$ (MHz): tần số danh định.



2.9. Chân RST:

- RST (***Reset***): thiết lập lại, chân số 9.
- Chức năng:
  - Là tín hiệu cho phép thiết lặp (*đặt*) lại trạng thái ban đầu cho hệ thống.
  - Là tín hiệu nhập, tích cực mức cao.

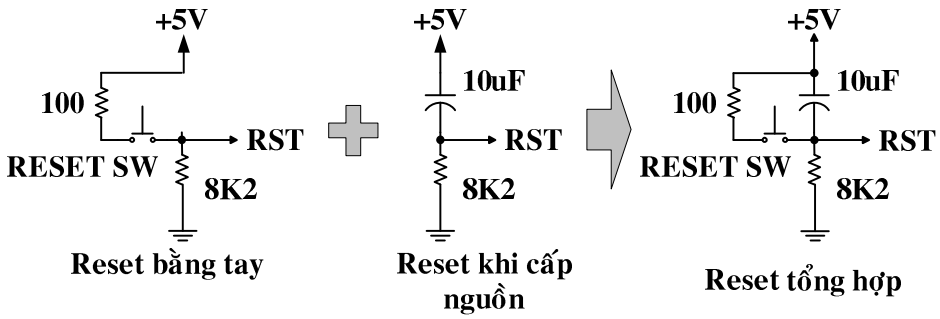
RST = 0 → Chip 8051 hoạt động bình thường.

RST = 1 → Chip 8051 được thiết lặp lại trạng thái ban đầu.

✓ ***Lưu ý:*** $t_{Reset} \geq 2 \times T_{Machine}$ $T_{Machine} = \frac{12}{f_{OSC}}$

$\mathbf{t_{RESET}}$ (µs): thời gian reset. $\mathbf{f_{OSC}}$ (MHz): tần số thạch anh.

$\mathbf{T_{MACHINE}}$ (µs): chu kỳ máy.



Ví dụ: Xác định chu kỳ máy và thời gian reset tương ứng cho từng trường hợp f_{OSC} = 11,0592MHz, f_{OSC} = 12MHz và f_{OSC} = 16MHz.

Giải

- f_{OSC} = 11,0592MHz → $T_{MACHINE}$ = 1,085µs và $t_{RESET} \geq$ 2,17µs.
- f_{OSC} = 12MHz → $T_{MACHINE}$ = 1µs và $t_{RESET} \geq$ 2µs.
- f_{OSC} = 16MHz → $T_{MACHINE}$ = 0,75µs và $t_{RESET} \geq$ 1,5µs

2.10. Chân Vcc, GND:

- Vcc, GND: nguồn cấp điện, chân số 40 và 20.
- Chức năng:
  - Cung cấp nguồn điện cho chip 8051 hoạt động.
  - Vcc = **+5V ± 10%** và GND = **0V**.

III. CẤU TRÚC CÁC PORT XUẤT NHẬP CHIP 8051:

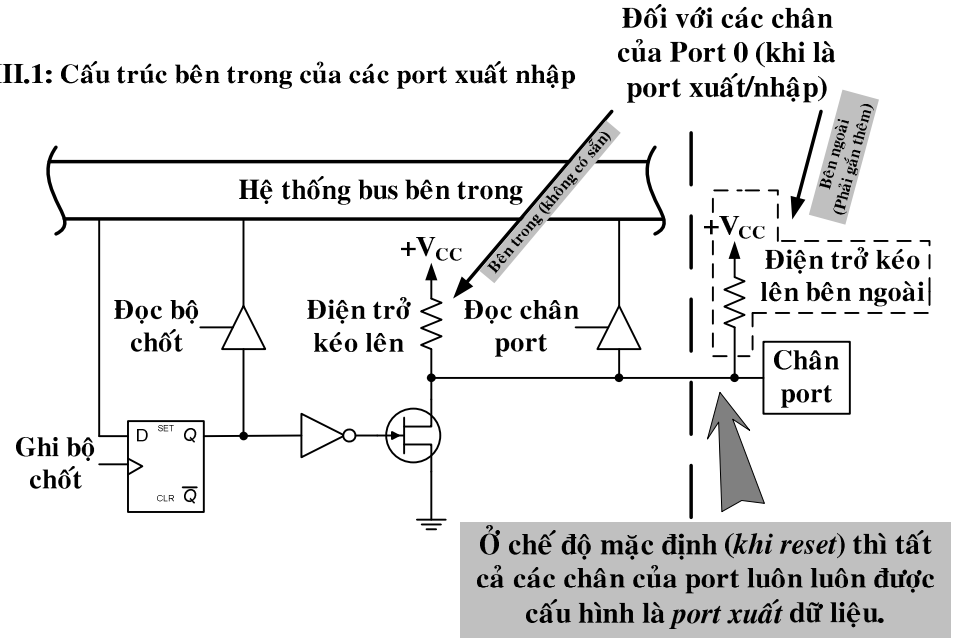
Khả năng fanout (số lượng tải đầu ra) của các từng chân port chip 8051 là:

- Port 0: 8 tải TTL.
- Port 1: 4 tải TTL.
- Port 2: 4 tải TTL.
- Port 3: 4 tải TTL.

✓ ***Lưu ý:***

- Khi ***Port 0 đóng vai trò là port xuất nhập*** thì sẽ không có điện trở kéo lên bên trong → ***do đó người sử dụng cần thêm vào điện trở kéo lên bên ngoài*** *(xem Hình III.1).*

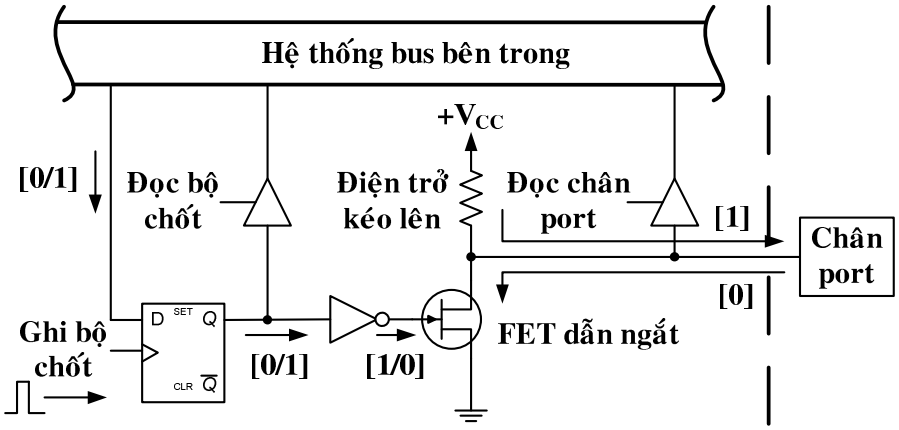
Hình III.1: Cấu trúc bên trong của các port xuất nhập



- Ở chế độ mặc định (*khi reset*) thì tất cả các chân của các port (*P0 – P3*) được cấu hình là ***port xuất*** dữ liệu.
- Muốn các chân port của chip 8015 làm ***port nhập*** dữ liệu thì ta cần phải được lập trình lại, bằng cách ghi mức logic cao (mức 1) đến tất cả các bit (*các chân*) của port trước khi bắt đầu nhập dữ liệu từ port (*vấn đề này được trình bày ở phần kế tiếp*).

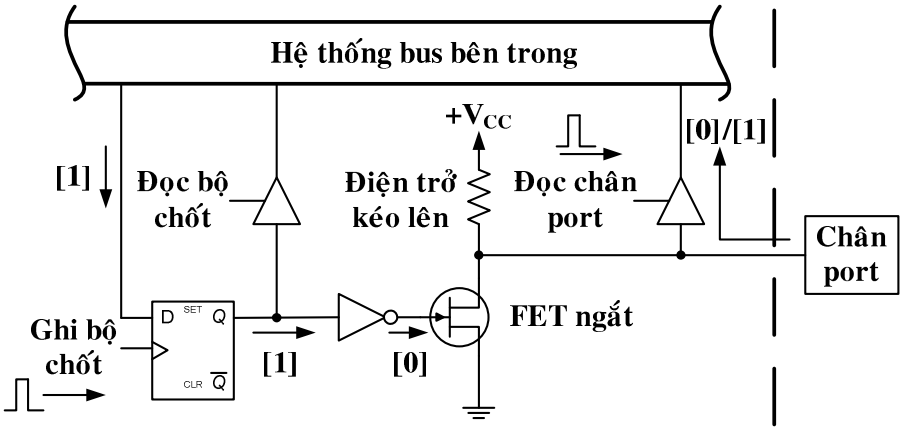
- Các chân trong cùng một port không nhất thiết phải có cùng kiểu cấu hình (*port xuất hoặc port nhập*). Nghĩa là trong cùng một port có thể có chân dùng để nhập dữ liệu, có thể có chân dùng để xuất dữ liệu. Điều này là tùy thuộc vào nhu cầu và mục đích của người lập trình.

Quá trình ghi chân port (*xuất dữ liệu ra chân port*).



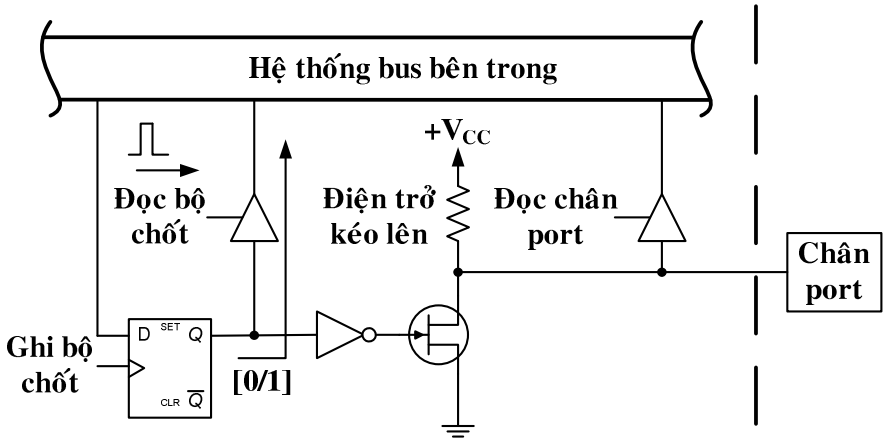
Hình III.3: Thao tác ghi chân port

Quá trình đọc chân port (*nhập dữ liệu từ chân port*).



Hình III.2: Thao tác đọc chân port

Quá trình đọc bộ chốt (*kiểm tra dữ liệu tại chân port*).



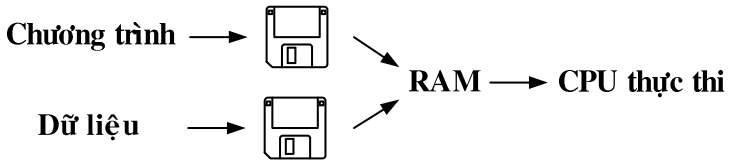
Hình III.4: Thao tác đọc bộ chốt

✓ ***Lưu ý:*** Việc đọc dữ liệu của bất kỳ một port nào có thể cho ta hai giá trị khác nhau tùy thuộc vào lệnh mà ta sử dụng để đọc dữ liệu từ port (*xem thêm trong phần tập lệnh*). Xảy ra hiện tượng không mong muốn này là do quá trình đọc dữ liệu của chip 8051 gồm hai quá trình khác nhau: *quá trình đọc chân port* và *quá trình đọc bộ chốt*.

- ***Quá trình đọc chân port:*** Khi ta sử dụng các lệnh ***MOV, ADD,…*** Dữ liệu nhận được sau khi thực hiện quá trình đọc là ***dữ liệu hiện tại ở các chân port***.
- ***Quá trình đọc bộ chốt:*** Khi ta sử dụng các lệnh ***ANL, ORL, XRL, CPL, INC, DEC, DJNZ, JBC, CLR bit, SETB bit, MOV bit.*** Dữ liệu nhận được sau khi thực hiện quá trình đọc là ***dữ liệu hiện tại ở các bộ chốt*** *(là các dữ liệu đã được ghi ra port tại thời điểm trước đó bởi quá trình ghi chân port)*, chứ không phải là dữ liệu hiện tại ở các chân port. Cho nên, nếu tại thời điểm thực hiện quá trình đọc mà dữ liệu tại các chân port có bị thay đổi đi chăng nữa thì dữ liệu đọc về cũng không được cập nhật.

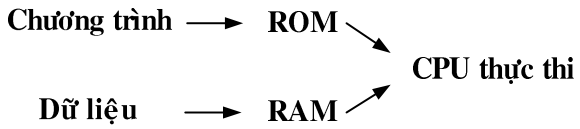
IV. TỔ CHỨC BỘ NHỚ CỦA CHIP 8051:

- Bộ vi xử lý → có không gian bộ nhớ chung cho dữ liệu và chương trình.



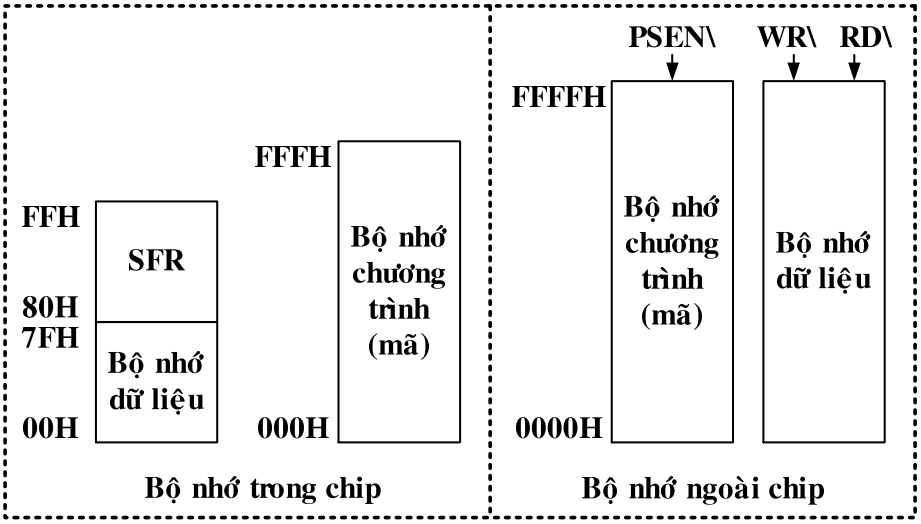
→ chương trình và dữ liệu nằm chung trên RAM trước khi đưa vào CPU để thực thi.

- Bộ vi điều khiển → có không gian bộ nhớ riêng cho dữ liệu và chương trình.

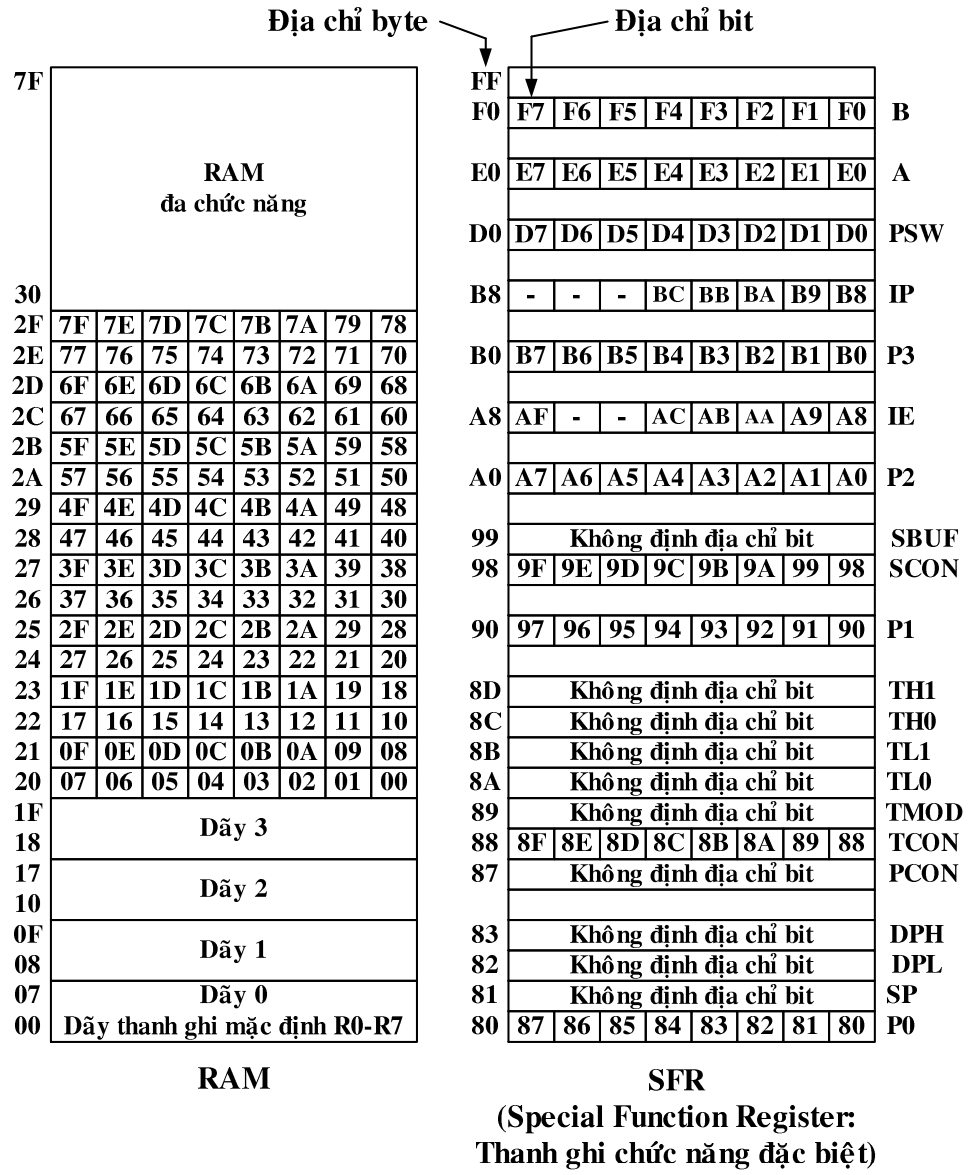


→ chương trình và dữ liệu nằm riêng trên ROM và RAM trước khi đưa vào CPU để thực thi.

- Tổ chức bộ nhớ của chip 8051:

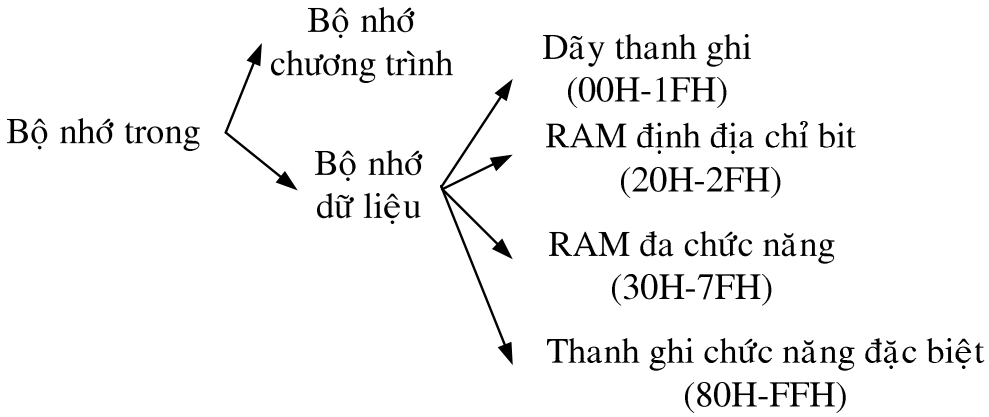


Không gian bộ nhớ của chip 8051



Bộ nhớ dữ liệu trên chip 8051

1. **Bộ nhớ trong:**



1.1. Bộ nhớ chương trình (*ROM*):

- Dùng để lưu trữ chương trình điều khiển cho chip 8051 hoạt động.
- Chip 8051 có 4 KB ROM trong, địa chỉ truy xuất: 000H – FFFH.

1.2. Bộ nhớ dữ liệu (*RAM*):

- Dùng để lưu trữ các dữ liệu và tham số.
- Chip 8051 có 128 byte RAM trong, địa chỉ truy xuất: 00H – 7FH.
- RAM trong của chip 8051 được chia ra:

  - RAM đa chức năng:

RAM đa chức năng → Gồm: 80 byte RAM
→ Địa chỉ truy xuất: 30H – 7FH
→ Kiểu truy xuất dữ liệu → Trực tiếp
→ Gián tiếp

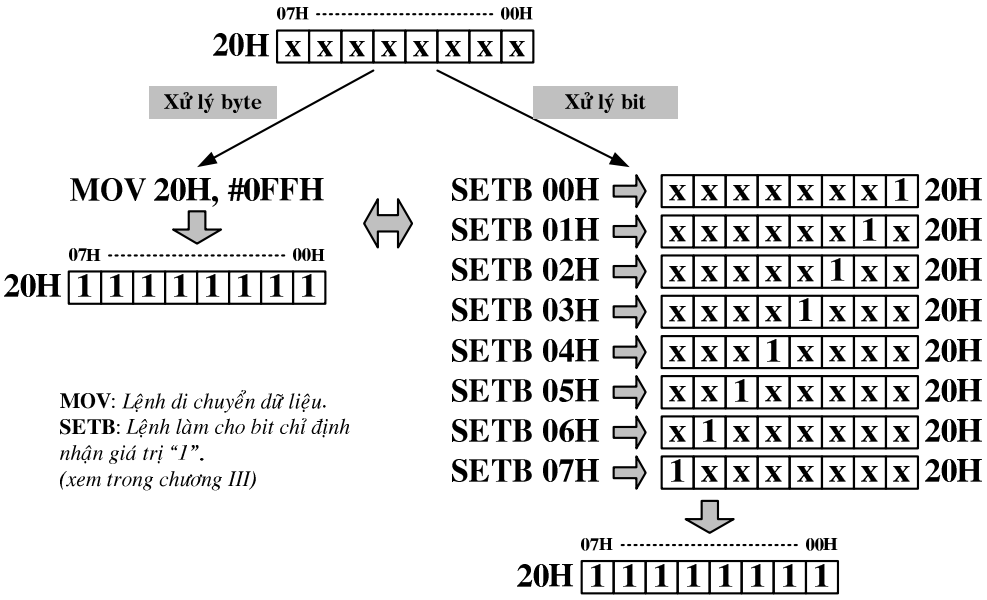
  - RAM định địa chỉ bit:

→ cho phép xử lý từng bit dữ liệu riêng lẻ mà không ảnh hưởng đến các bit khác trong cả byte.

RAM định địa chỉ bit → Gồm: 128 byte RAM
→ Địa chỉ truy xuất: 20H – 2FH
→ Kiểu truy xuất dữ liệu → Trực tiếp
→ Gián tiếp
→ **Theo từng bit**

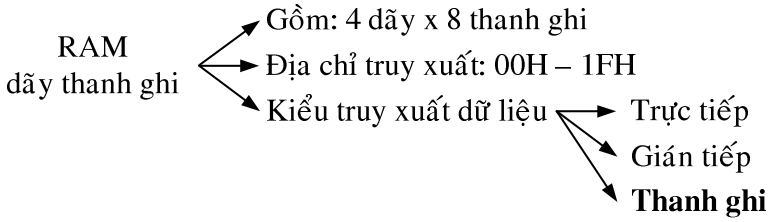
✓ ***Lưu ý:*** Nếu trong chương trình không sử dụng các bit trong vùng RAM định địa chỉ bit này, ta có thể sử dụng vùng nhớ 20H – 2FH cho các mục đích khác của ta. Ngược lại, ta phải viết chương trình cẩn thận khi sử dụng vùng nhớ 20H – 2FH vì nếu sơ suất ta có thể ghi dữ liệu đè lên các bit đã được sử dụng.

✓ <u>*Ví dụ:*</u> Viết lệnh làm cho 8 bit trong ô nhớ có địa chỉ 20H thuộc RAM nội có giá trị là 1 (*xét trường hợp địa chỉ byte và địa chỉ bit*).



- Các dãy thanh ghi:

→ cho phép truy xuất dữ liệu nhanh, lệnh truy xuất đơn giản và ngắn gọn.



Bảng số liệu dưới đây minh họa địa chỉ của các ô nhớ trong một dãy và các ký hiệu thanh ghi R0 – R7 được gán cho từng ô nhớ trong dãy tích cực.

	Dãy 0	**Dãy 1**	**Dãy 2**	**Dãy 3**
R0	00H	08H	10H	18H
R1	01H	09H	11H	19H
R2	02H	0AH	12H	1AH
R3	03H	0BH	13H	1BH
R4	04H	0CH	14H	1CH
R5	05H	0DH	15H	1DH
R6	06H	0EH	16H	1EH
R7	07H	0FH	17H	1FH

Địa chỉ của các thanh ghi (R0 – R7) tương ứng với dãy thanh ghi tích cực.

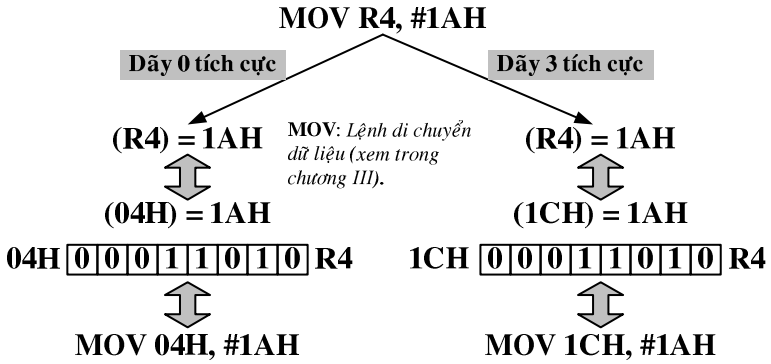
✓ **Lưu ý:**

- Ở chế độ mặc định thì dãy thanh ghi tích cực (đang được sử dụng) là **dãy 0** và các thanh ghi trong dãy lần lượt có tên là **R0 - R7**. Có thể thay đổi dãy tích cực bằng cách thay đổi các bit chọn dãy thanh ghi RS1 và RS0 trong thanh ghi PSW *(xem phần thanh ghi PSW)*.
- Nếu chương trình của ta chỉ sử dụng dãy thanh ghi đầu tiên (*dãy 0*) thì ta có thể sử dụng vùng nhớ 08H – 1FH cho các mục đích khác của ta. Nhưng nếu trong chương trình có sử dụng các dãy thanh ghi (*dãy 1, 2 hoặc 3*) thì phải rất cẩn thận khi sử dụng vùng nhớ từ 1FH trở xuống vì nếu sơ suất ta có thể ghi dữ liệu đè lên các thanh ghi R0 – R7 của ta.

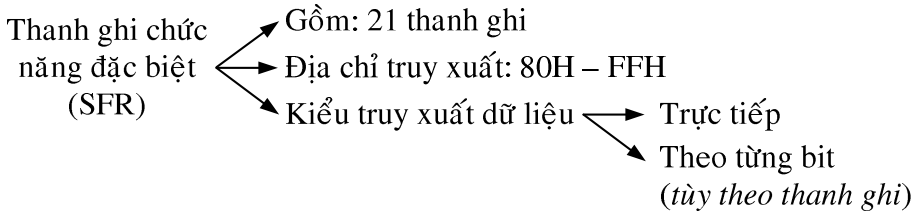
✓ *Ví dụ 1:* Quan hệ giữa ký hiệu thanh ghi R4 với các ô nhớ có địa chỉ tương ứng trong dãy thanh ghi tích cực?

- Nếu dãy 0 tích cực: Thanh ghi **R4** ⇔ Ô nhớ **04H** RAM nội.
- Nếu dãy 1 tích cực: Thanh ghi **R4** ⇔ Ô nhớ **0CH** RAM nội.
- Nếu dãy 2 tích cực: Thanh ghi **R4** ⇔ Ô nhớ **14H** RAM nội.
- Nếu dãy 3 tích cực: Thanh ghi **R4** ⇔ Ô nhớ **1CH** RAM nội.

✓ *Ví dụ 2:* Khi chip 8051 thực hiện lệnh **MOV R4, #1AH** thì giá trị "1AH" sẽ được nạp vào trong ô nhớ có địa chỉ là bao nhiêu thuộc RAM nội. Xét tương ứng cho từng trường hợp dãy thanh ghi tích cực là *Dãy 0* và *Dãy 3*?



1.3. Thanh ghi chức năng đặc biệt (*SFR*):



✓ **Lưu ý:**

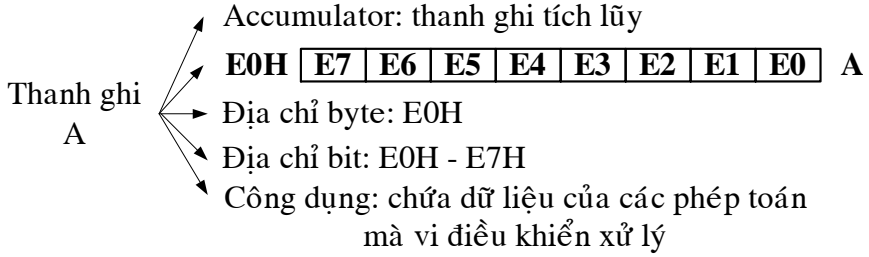
- Không được phép đọc hay ghi dữ liệu vào các địa chỉ SFR mà nó chưa được đăng ký (*nghĩa là các địa chỉ SFR chưa được đặt tên*). Vì việc đọc hay ghi dữ liệu vào các nơi này có thể làm phát sinh những hoạt động không mong muốn và đó có thể là nguyên nhân làm cho chương trình của ta không tương thích với các phiên bản sau của chip MCS-51 (*có thể ở các phiên bản đó các địa chỉ SFR này được sử dụng cho một vài mục đích khác*).
- **Chỉ được truy xuất các SFR bằng kiểu định địa chỉ trực tiếp (*tuyệt đối không sử dụng kiểu định địa chỉ gián tiếp trong trường hợp này*).**

✓ *Ví dụ:* Cho biết trước (R0)=90H. Viết lệnh dùng để xuất (ghi) giá trị 5AH ra Port1 như sau (*xem giải thích lệnh trong "**Chương 3: Tập lệnh của 8051.**"*):

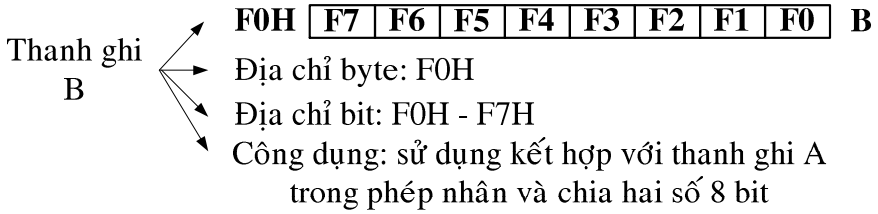
- Sử dụng kiểu định địa chỉ trực tiếp:
  MOV P1, #5AH hoặc **MOV 90H, #5AH**
- Sử dụng kiểu định địa chỉ gián tiếp:
  MOV @R0, #5AH ⇨ SAI

⇨ Điều này không hợp lệ đối với chip 8051 vì phương pháp định địa chỉ gián tiếp như trên chỉ sử dụng cho vùng nhớ RAM nội. Trong khi đó RAM nội của chip 8051 chỉ có 128 byte (*00H – 7FH*), cho nên khi thực hiện lệnh này nó sẽ trả về kết quả không xác định. (*Lưu ý: nếu ta dùng phiên bản chip 8052 thì sẽ tránh được điều này*).

1.3.1. Thanh ghi A:



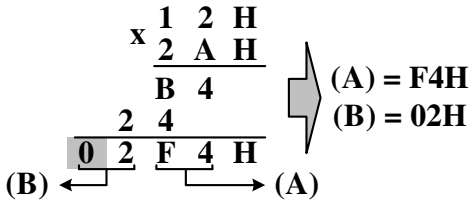
1.3.2. Thanh ghi B:



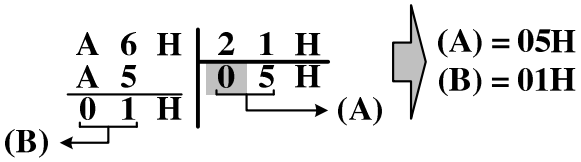
- Phép nhân 2 số 8 bit không dấu → kết quả là số 16 bit.
  - Byte cao → chứa vào thanh ghi B.
  - Byte thấp → chứa vào thanh ghi A.
- Phép chia 2 số 8 bit → thương số và số dư là số 8 bit.

- Thương số → chứa vào thanh ghi A.
- Số dư → chứa vào thanh ghi B.

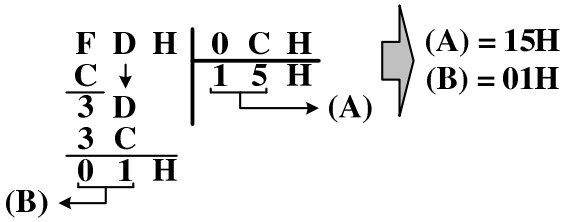
✓ *Ví dụ:* Thực hiện phép tính 12H x 2AH. Hỏi (A)=?, (B)=?



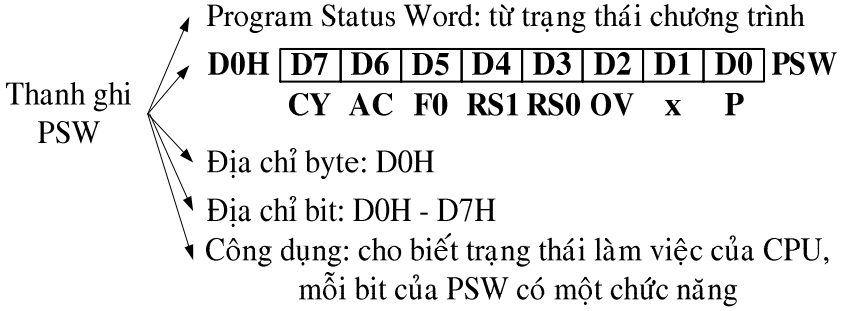
✓ *Ví dụ:* Thực hiện phép tính A6H : 21H. Hỏi (A)=?, (B)=?



✓ *Ví dụ:* Thực hiện phép tính FDH : 0CH. Hỏi (A)=?, (B)=?



1.3.3. Thanh ghi từ PSW:



- Cờ CY (*Carry Flag*): cờ nhớ → báo có nhớ/mượn tại bit 7.
  - CY = 0: nếu không có nhớ từ bit 7 hoặc không có mượn cho bit 7.
  - CY = 1: nếu có nhớ từ bit 7 hoặc có mượn cho bit 7.
- Cờ AC (*Auxiliary Carry*): cờ nhớ phụ → báo có nhớ/mượn tại bit 3.
  - AC = 0: nếu không có nhớ từ bit 3 hoặc không có mượn cho bit 3.
  - AC = 1: nếu có nhớ từ bit 3 hoặc có mượn cho bit 3.
- Cờ F0 (*Flag 0*): cờ zero → có nhiều mục đích dành cho các ứng dụng khác nhau của người lập trình (*dự trữ cho các phiên bản chip trong tương lai*).

- Bit RS0, RS1 (*Register Select*): bit chọn dãy thanh ghi → cho phép xác định dãy thanh ghi tích cực (*hay dãy thanh ghi mà các thanh ghi có tên là R0-R7*).

RS1	RS0	Dãy thanh ghi	R0 → R7
0	0	Dãy 0	00H → 07H
0	1	Dãy 1	08H → 0FH
1	0	Dãy 2	10H → 17H
1	1	Dãy 3	18H → 1FH

- Cờ OV (*Overflow*): cờ tràn → báo kết quả tính toán của phép toán số học (*phép toán có dấu*) có nằm trong khoảng từ -128 đến +127 hay không.
  - OV = 0: nếu -128 ≤ kết quả ≤ +127.
  - OV = 1: nếu kết quả < -128 hoặc kết quả > +127. Nói cách khác là: Đối với *phép cộng* thì OV=1 nếu có nhớ từ bit 7 nhưng không có nhớ từ bit 6 hoặc nếu có nhớ từ bit 6 nhưng không có nhớ từ bit 7. Đối với *phép trừ* thì OV=1 nếu có mượn cho bit 7 nhưng không có mượn cho bit 6 hoặc nếu có mượn bit 6 nhưng không có mượn bit 7.

Biểu diễn toán hạng 8 bit có dấu.

DEC	BIN	HEX
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
.....	.....	.....
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
.....	.....	.....
+125	0111 1101	7D
+126	0111 1110	7E
+127	0111 1111	7F

Bảng liệt kê các số 8 bit có dấu.

- Cờ P (Parity): cờ chẵn lẻ → báo số chữ số 1 trong thanh ghi A là số chẵn hay số lẻ (*trong chip 8051 sử dụng chế độ parity chẵn*).
  - P = 0: nếu số chữ số 1 trong thanh ghi A là số chẵn (*parity chẵn*).
  - P = 1: nếu số chữ số 1 trong thanh ghi A là số lẻ (*parity chẵn*).

✓ *Ví dụ:* Minh họa cách 8051 biểu diễn số -5.

Giải

Các bước thực hiện:

B1:	**0000 0101**	Biểu diễn số 5 dạng nhị phân 8 bit.
B2:	**1111 1010**	Lấy bù 1.
B3:	**1111 1011**	Lấy bù 2.

Vậy số FBH là biểu diễn số có dấu dạng bù 2 của số -5.

✓ *Ví dụ:* Minh họa cách 8051 biểu diễn số -34H.

Các bước thực hiện:

B1:	**0011 0100**	Biểu diễn số 34H dạng nhị phân 8 bit.
B2:	**1100 1011**	Lấy bù 1.
B3:	**1100 1100**	Lấy bù 2.

Vậy số CCH là biểu diễn số có dấu dạng bù 2 của số -34H.

✓ *Ví dụ:* Minh họa cách 8051 biểu diễn số -128.

Các bước thực hiện:

B1:	**1000 0000**	Biểu diễn số -128 dạng nhị phân 8 bit.
B2:	**0111 1111**	Lấy bù 1.
B3:	**1000 0000**	Lấy bù 2.

Vậy số 80H là biểu diễn số có dấu dạng bù 2 của số -128.

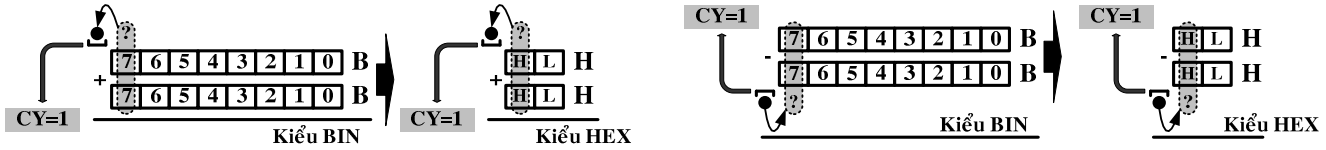
✓ *Ví dụ:* Minh họa trạng thái hoạt động của các cờ CY, AC, OV và P khi thực hiện phép cộng/trừ số học hai giá trị với nhau.

Các ký hiệu:

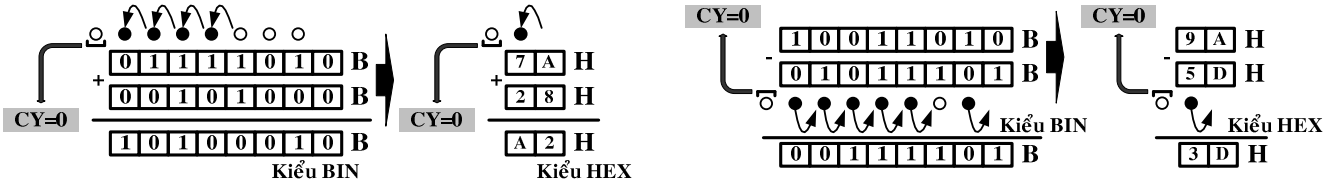
● Có nhớ / có mượn.
○ Không nhớ / không mượn.
? Tùy thuộc trạng thái trước đó.

- *Cờ nhớ (CY):*

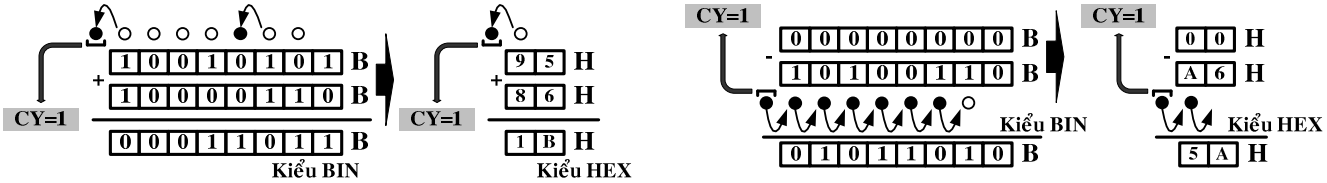
Minh họa hoạt động của cờ CY trong trường hợp CY = 1:



Xét cờ CY trong hai trường hợp "***7AH+28H***" và "***9AH-5DH***":

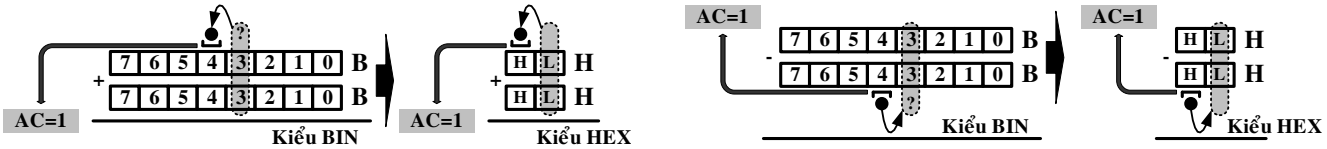


Xét cờ CY trong hai trường hợp "***95H+86H***" và "***00H-A6H***":

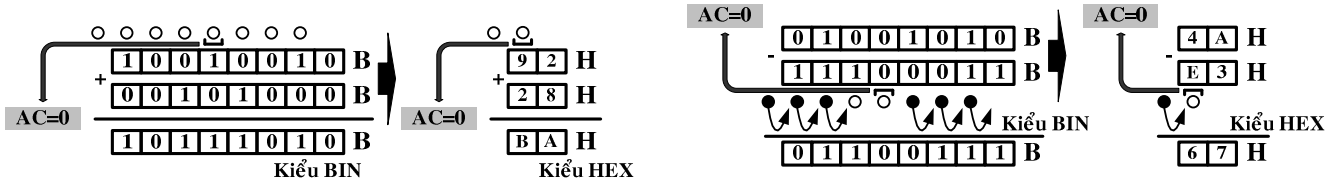


- *Cờ nhớ phụ (AC):*

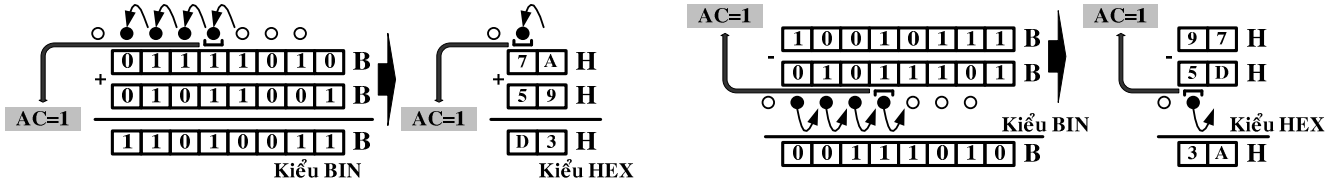
Minh họa hoạt động của cờ AC trong trường hợp AC = 1



Xét cờ AC trong hai trường hợp "***92H+28H***" và "***4AH-E3H***":

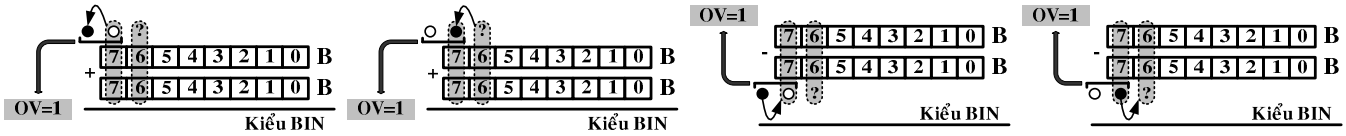


Xét cờ AC trong hai trường hợp "***7AH+59H***" và "***97H-5DH***":

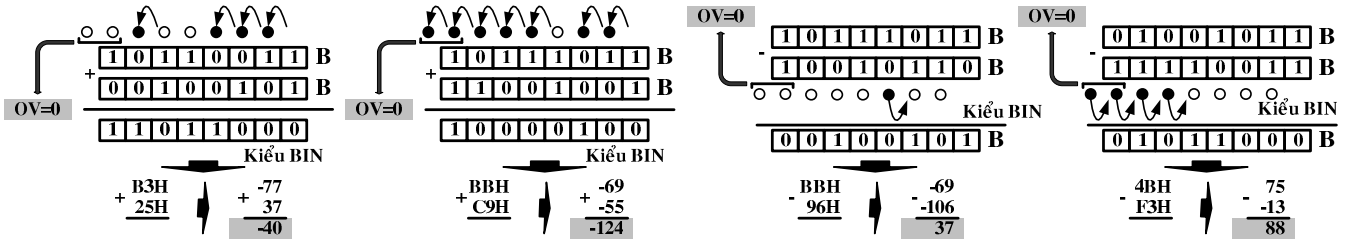


- *Cờ tràn (OV)*:

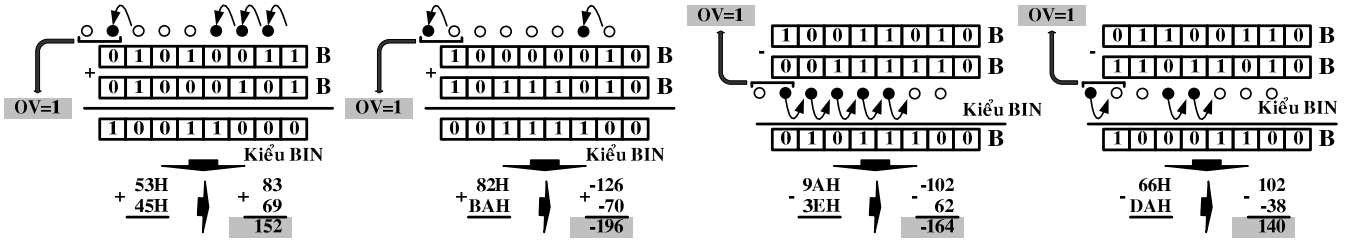
Minh họa hoạt động của cờ OV trong trường hợp OV = 1



Xét cờ OV trong các trường hợp "***B3H+25H***", "***BBH+C9H***", "***BBH-96H***" và "***4BH-F3H***":

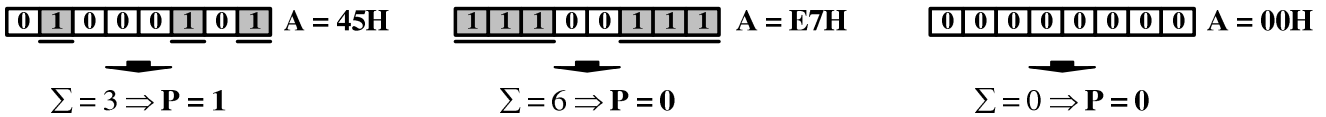


Xét cờ OV trong các trường hợp "***53H+45H***", "***82H+BAH***", "***9AH-3EH***" và "***66H-DAH***":



- *Cờ Parity (P):*

Xét cờ P trong các trường hợp "***(A)=45H***", "***(A)=E7H***", "***(A)=00H***":



✓ *Ví dụ:* Xác định nội dung các ô nhớ thuộc RAM nội của đoạn chương trình sau:

```
SETB PSW.3      ;Chọn dãy thanh ghi 1, RS0=1
CLR PSW.4       ;Chọn dãy thanh ghi 1, RS1=0
MOV R0, #10H    ;Nạp thanh ghi R0 giá trị 10H
MOV R1, #2AH    ;Nạp thanh ghi R1 giá trị 2AH
MOV R3, #3FH    ;Nạp thanh ghi R3 giá trị 3FH
MOV R5, #57H    ;Nạp thanh ghi R5 giá trị 57H
MOV R7, #0A9H   ;Nạp thanh ghi R7 giá trị A9H
```

MOV: *Lệnh di chuyển dữ liệu.*
SETB: *Lệnh làm cho bit chỉ định nhận giá trị "1".*
CLR: *Lệnh làm cho bit chỉ định nhận giá trị "0".*
(xem trong chương III)

Vì dãy tích cực là ***Dãy 1*** nên ta có:
Ô nhớ có địa chỉ 08H nhận giá trị 10H.
Ô nhớ có địa chỉ 09H nhận giá trị 2AH.
Ô nhớ có địa chỉ 0BH nhận giá trị 3FH.
Ô nhớ có địa chỉ 0DH nhận giá trị 57H.
Ô nhớ có địa chỉ 0FH nhận giá trị A9H.

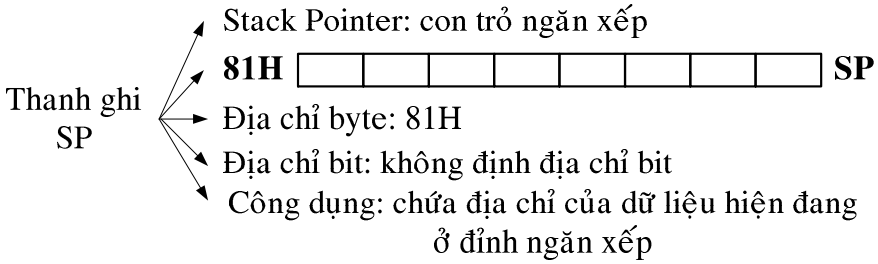
✓ *Ví dụ:* Xác định nội dung các ô nhớ thuộc RAM nội của đoạn chương trình sau:

```
CLR PSW.3       ;Chọn dãy thanh ghi 2, RS0=0
SETB PSW.4      ;Chọn dãy thanh ghi 2, RS1=1
MOV R0, #10H    ;Nạp thanh ghi R0 giá trị 10H
MOV R1, #2AH    ;Nạp thanh ghi R1 giá trị 2AH
MOV R3, #3FH    ;Nạp thanh ghi R3 giá trị 3FH
MOV R5, #57H    ;Nạp thanh ghi R5 giá trị 57H
MOV R7, #0A9H   ;Nạp thanh ghi R7 giá trị A9H
```

MOV: *Lệnh di chuyển dữ liệu.*
SETB: *Lệnh làm cho bit chỉ định nhận giá trị "1".*
CLR: *Lệnh làm cho bit chỉ định nhận giá trị "0".*
(xem trong chương III)

Vì dãy tích cực là ***Dãy 2*** nên ta có:
Ô nhớ có địa chỉ 10H nhận giá trị 10H.
Ô nhớ có địa chỉ 11H nhận giá trị 2AH.
Ô nhớ có địa chỉ 13H nhận giá trị 3FH.
Ô nhớ có địa chỉ 15H nhận giá trị 57H.
Ô nhớ có địa chỉ 17H nhận giá trị A9H.

1.3.4. Thanh ghi SP:



- Ngăn xếp là vùng nhớ dùng để lưu trữ tạm thời các dữ liệu.
- Đối với chip 8051 thì vùng nhớ được dùng để làm ngăn xếp được giữ trong RAM nội.
- Để sử dụng ngăn xếp thì ta phải khởi động thanh ghi SP (*nghĩa là nạp giá trị cho thanh ghi SP*) → vùng nhớ của ngăn xếp có *địa chỉ bắt đầu:* **(SP)+1** *và địa chỉ kết thúc:* **7FH**.
- Nếu không khởi động SP → vùng nhớ của ngăn xếp có *địa chỉ bắt đầu:* **08H** *và địa chỉ kết thúc:* **7FH** (*chế độ mặc định*).

✓ **Lưu ý:** Trong trường hợp không khởi động SP (*chế độ mặc định*) thì dãy thanh ghi 1 (*và có thể là dãy 2 và dãy 3*) sẽ không còn hợp lệ vì khi đó vùng nhớ này đã được sử dụng để làm ngăn xếp. Điều này có nghĩa là nếu ta sử dụng các dãy thanh ghi này và lưu trữ dữ liệu vào đó thì có khả năng sẽ bị mất do tác động cất dữ liệu vào ngăn xếp của các lệnh (*PUSH, ACALL, LCALL, ...*).

✓ *Ví dụ:* Hãy cho biết tầm địa chỉ của vùng nhớ ngăn xếp trong hai trường hợp sau: (SP)=5FH và (SP)=49H.

Theo qui định thì vùng nhớ của ngăn xếp có *địa chỉ bắt đầu:* **(SP)+1** *và địa chỉ kết thúc:* **7FH**.

- Trường hợp (SP)=5FH: tầm địa chỉ của vùng nhớ ngăn xếp là **60H - 7FH**.
- Trường hợp (SP)=49H: tầm địa chỉ của vùng nhớ ngăn xếp là **4AH - 7FH**.

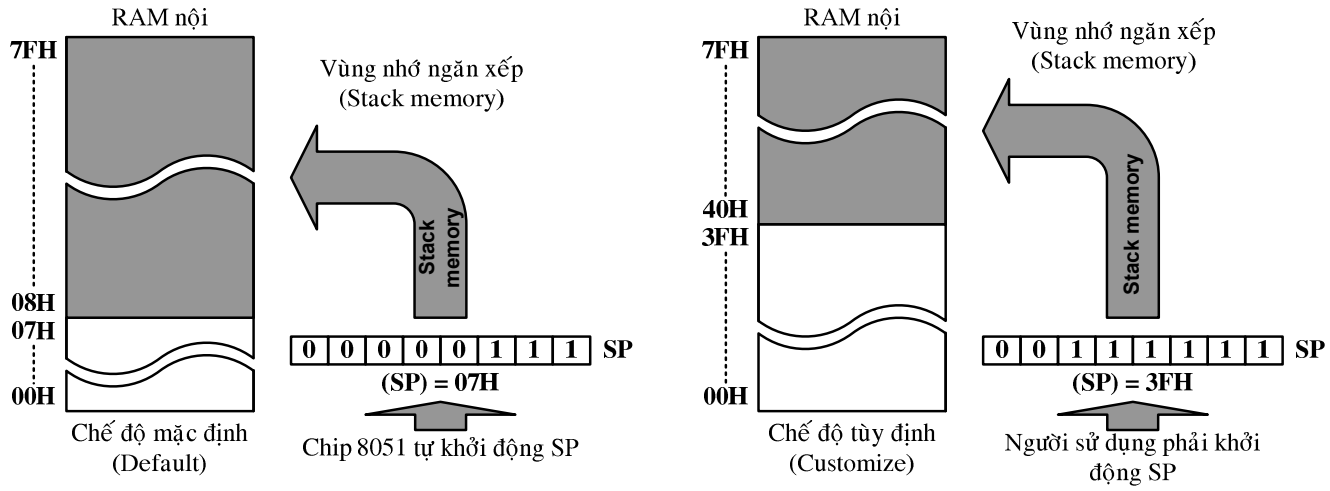
✓ *Ví dụ:* Hãy cho biết giá trị cần phải nạp cho thanh ghi SP để vùng nhớ ngăn xếp có tầm địa chỉ trong hai trường hợp sau: 62H – 7FH và 50H – 7FH.

Theo qui định thì vùng nhớ của ngăn xếp có *địa chỉ bắt đầu:* **(SP)+1** *và địa chỉ kết thúc:* **7FH**.

- Trường hợp 62H – 7FH: giá trị cần nạp cho thanh ghi SP là **61H**.
- Trường hợp 50H – 7FH: giá trị cần nạp cho thanh ghi SP là **4FH**.

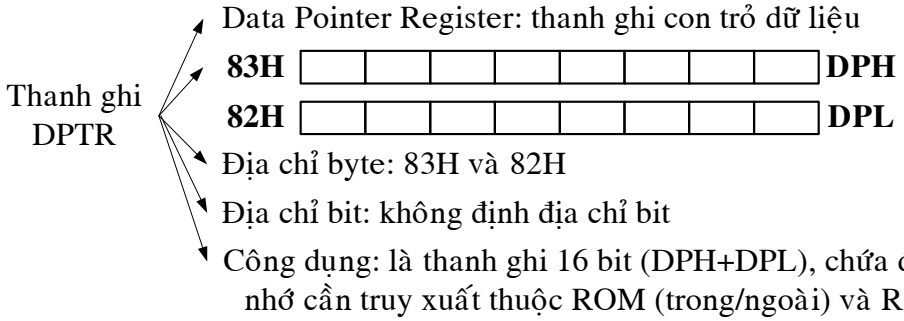
✓ *Ví dụ:* Minh họa vùng nhớ ngăn xếp trong trường hợp không khởi động SP (*chế độ mặc định*) và có khởi động SP (*với (SP) = 3FH*).

- Nếu người sử dụng không khởi động thanh ghi SP (*chế độ mặc định*) thì: (*xem hình bên dưới, phía trái*)



- Nếu người sử dụng muốn vùng nhớ ngăn xếp (chế độ tùy định) có tầm địa chỉ là 40H–7FH thì: (*xem hình bên trên, phía phải*)

1.3.5. Thanh ghi DPTR:



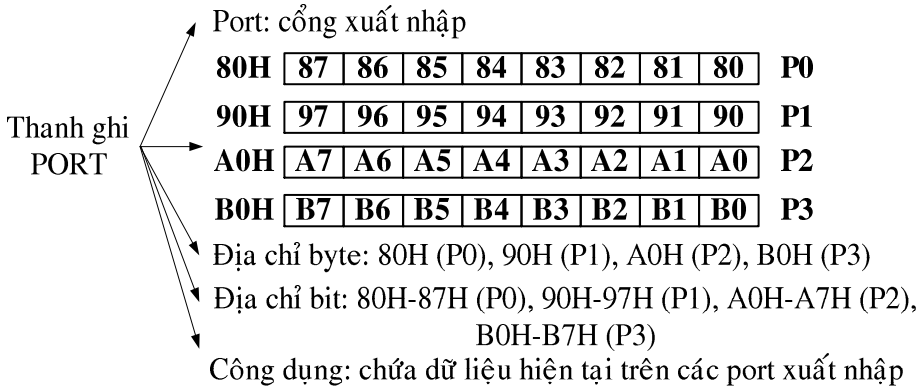
✓ *Ví dụ:* Khi ta muốn truy xuất (*ghi/đọc*) dữ liệu từ một ô nhớ thuộc RAM ngoài có địa chỉ là 0123H thì ta phải làm sao nạp được giá trị 0123H vào thanh ghi DPTR và sau đó thực hiện lệnh truy xuất **MOVX** (*xem giải thích lệnh trong "**Chương 3: Tập lệnh của 8051.**"*).

(DPTR) = 0123H ⇔ (DPH) = 01H và (DPL) = 23H

✓ *Ví dụ:* Khi ta muốn truy xuất (*đọc*) byte mã từ một ô nhớ thuộc ROM trong có địa chỉ là 0ABCH thì ta phải làm sao nạp được giá trị 0ABCH vào thanh ghi DPTR và sau đó thực hiện lệnh truy xuất **MOVC** (*xem giải thích lệnh trong "**Chương 3: Tập lệnh của 8051.**"*).

(DPTR) = 0ABCH ⇔ (DPH) = 0AH và (DPL) = BCH

1.3.6. Thanh ghi port xuất nhập:

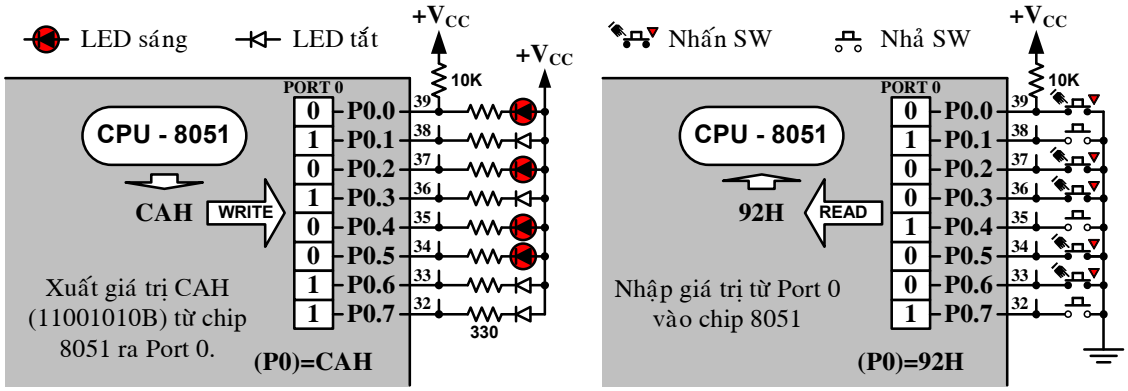


✓ **Lưu ý:**

- Trong trường hợp phần cứng có sử dụng ROM hoặc RAM bên ngoài thì ta không thể sử dụng Port 0 và Port 2 để xuất nhập dữ liệu. Vì khi đó chip 8051 sẽ sử dụng hai port này để xác định địa chỉ và dữ liệu cho bộ nhớ ngoài. Khi đó, ta chỉ có thể sử dụng Port 1 và Port 3 để xuất nhập dữ liệu.
- Ở chế độ mặc định (*khi reset*) thì tất cả các chân của các port (*P0 – P3*) được cấu hình là ***port xuất*** dữ liệu. Muốn các chân port của chip 8015 làm ***port nhập*** dữ liệu thì ta cần phải được lập trình lại, bằng cách ghi mức logic cao (*mức 1*) đến tất cả các bit (*các chân*) của port trước khi bắt đầu nhập dữ liệu từ port.

✓ *Ví dụ 1:* Hoạt động xuất (*ghi*) và nhập (*đọc*) dữ liệu tại các chân port (Port 0) của chip 8051 (*xem hình minh họa bên dưới*).

- *Hình phía trái:* Minh họa trạng thái hoạt động của port khi thực hiện lệnh xuất (*ghi*) dữ liệu ra Port 0 của chip 8051.
- *Hình phía phải:* Minh họa trạng thái hoạt động của port khi thực hiện lệnh nhập (*đọc*) dữ liệu từ Port 0 của chip 8051.



✓ *Ví dụ 2:* Đoạn chương trình dưới đây sẽ cấu hình cho Port 0 làm port nhập (*đọc*) dữ liệu. Sau đó liên tục đọc dữ liệu từ port này và gửi dữ liệu đó đến Port 1 (*xem giải thích lệnh trong "**Chương 3: Tập lệnh của 8051.**"*):

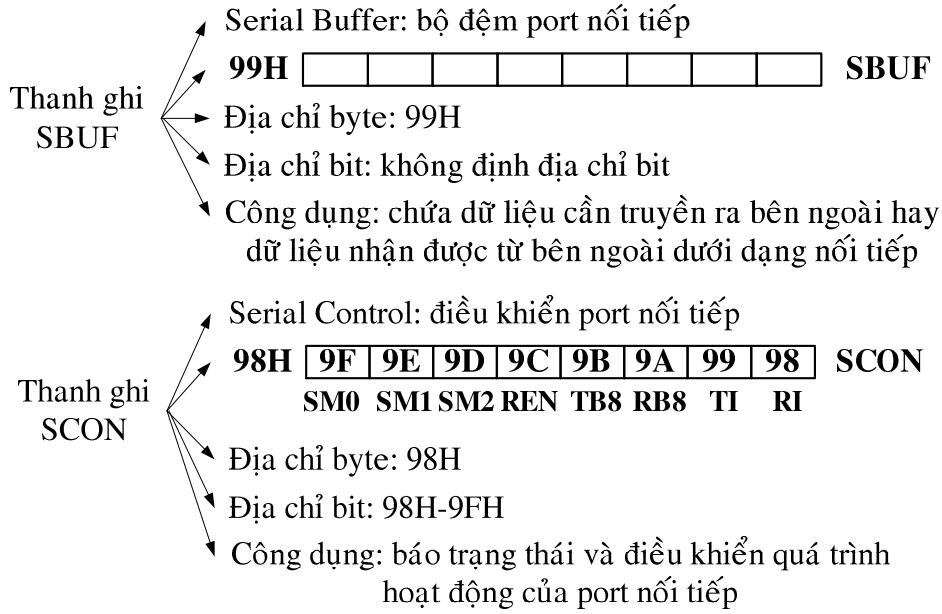
```
        MOV     P0, #0FFH       ;Cấu hình P0 làm port nhập bằng
                                ;cách ghi "1" vào tất cả các bit.
BACK:
        MOV     A, P0           ;Đọc dữ liệu từ P0.
        MOV     P1, A           ;Gửi dữ liệu đó ra P1.
        SJMP    BACK            ;Lặp lại.
```

✓ *Ví dụ 3:* Đoạn chương trình dưới đây sẽ thực hiện các thao tác sau (*xem giải thích lệnh trong "**Chương 3: Tập lệnh của 8051.**"*):

- Liên tục kiểm tra bit P1.2 cho đến khi bit này bằng 1.
- Khi P1.2 =1, hãy xuất (*ghi*) giá trị 45H ra P0.
- Gửi một xung mức cao tới P1.3.

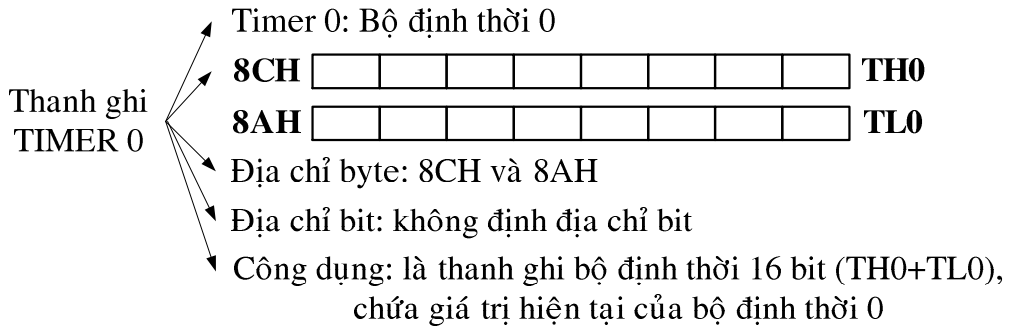
```
        SETB    P1.2            ;Cấu hình P1.2 làm ngõ vào.
        JNB     P1.2, $         ;Kiểm tra liên tục nếu P1.2 = 0.
        MOV     P0, #45H        ; Xuất giá trị 45H ra P0.
        SETB    P1.3            ;Đưa P1.3 lên cao rồi đưa P1.3
        CLR     P1.3            ;xuống thấp để tạo xung.
```

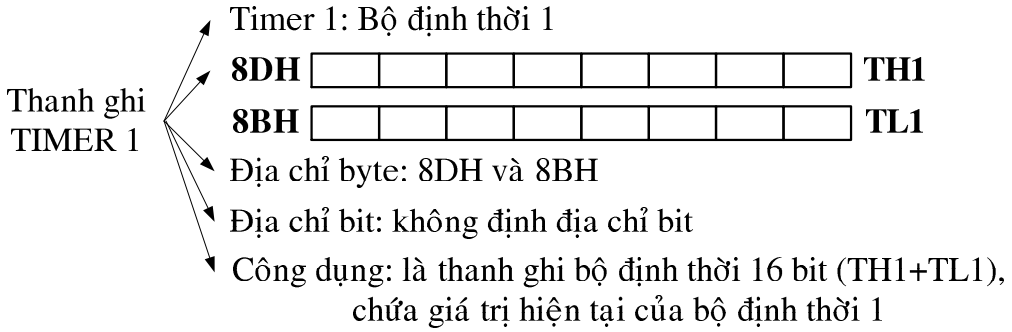
1.3.7. Thanh ghi port nối tiếp:

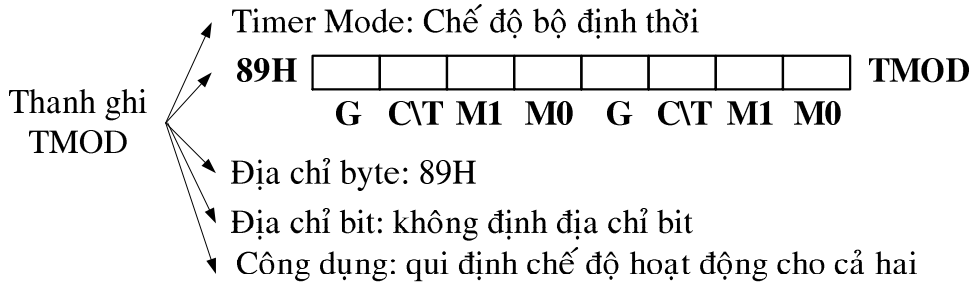


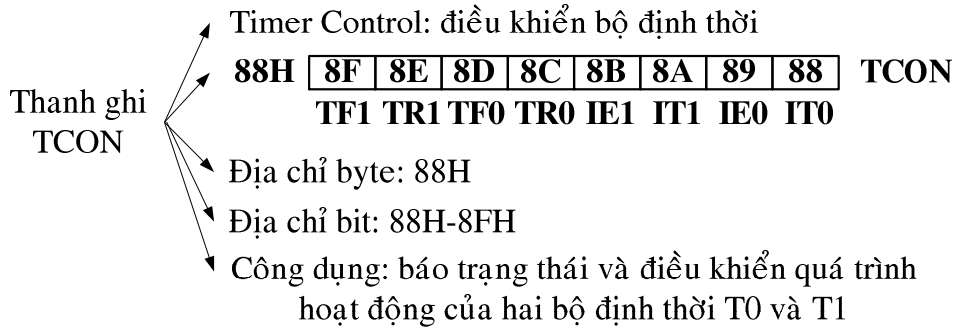
(*xem thêm trong **"Chương 5: Hoạt động của port nối tiếp."***)

1.3.8. Thanh ghi định thời:



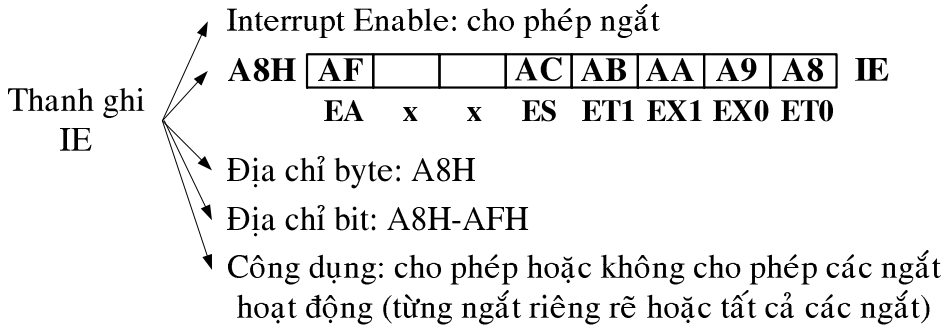


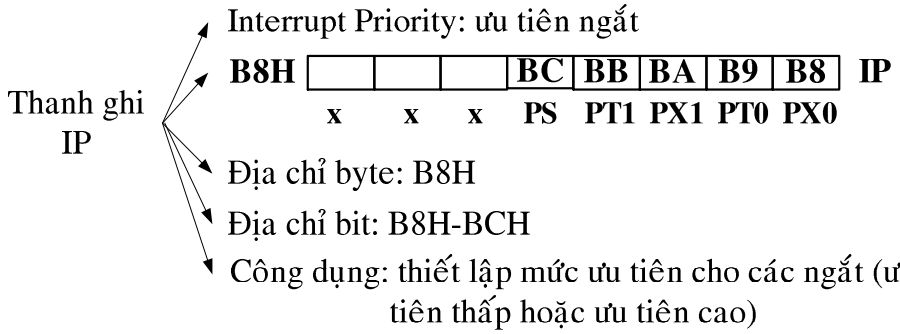




(*xem thêm trong* ***"Chương 4: Hoạt động của bộ định thời."***)

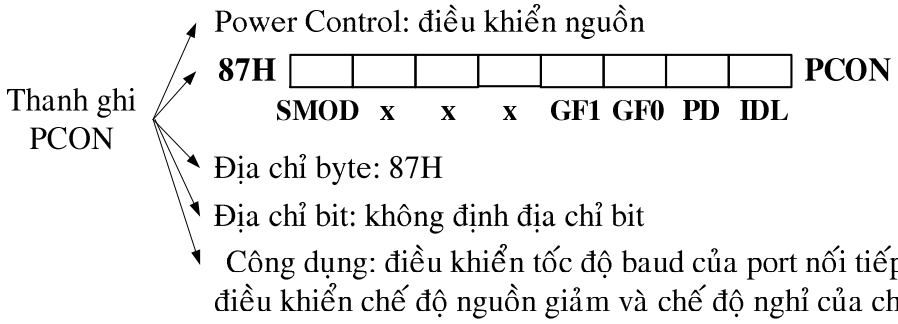
1.3.9. Thanh ghi ngắt:





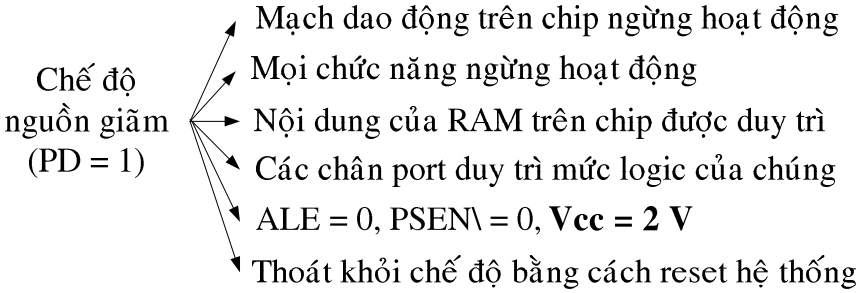
(*xem thêm trong* ***"Chương 6: Hoạt động ngắt."***)

1.3.10. Thanh ghi điều khiển nguồn:

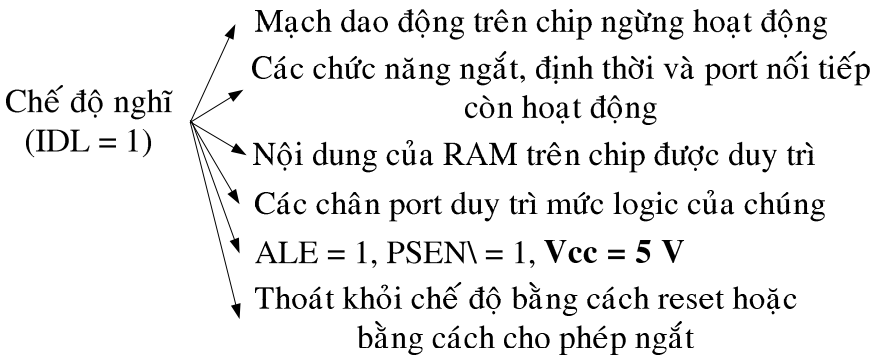


(*xem thêm trong* ***"Chương 5: Hoạt động của port nối tiếp."***)

- Bit SMOD (*Serial Mode*) → cho phép tăng gấp đôi tốc độ truyền dữ liệu nối tiếp (*tốc độ baud*) khi SMOD = 1.
- Bit GF1, GF0 (*General Function*) → cho phép người lập trình dùng với mục đích riêng (*dự trữ cho các phiên bản chip trong tương lai*).
- Bit PD (*Power Down*) → dùng để qui định chế độ nguồn giãm.
- Bit IDL (*Idle*) → dùng để qui định chế độ nghĩ.

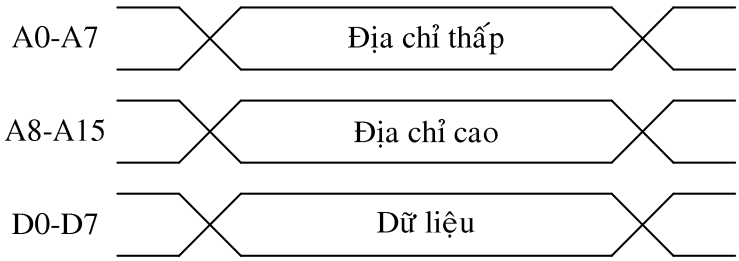


✓ **Lưu ý:** Hệ thống phải phục hồi Vcc = 5V trước khi thoát khỏi chế độ nguồn giãm.

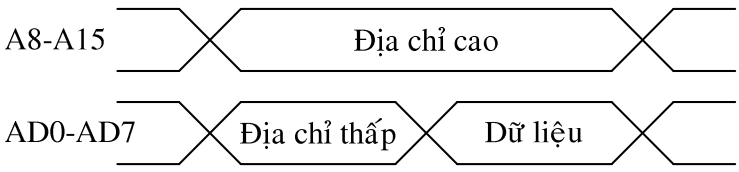


2. Bộ nhớ ngoài:

- Chip 8051 cho ta khả năng mở rộng:
  - Không gian bộ nhớ chương trình lên đến 64 KB.
  - Không gian bộ nhớ dữ liệu lên đến 64 KB.
- Khi sử dụng bộ nhớ ngoài:
  - Port 0 → bus địa chỉ byte thấp và bus dữ liệu đa hợp (*AD0-AD7*).
  - Port 2 → bus địa chỉ byte cao (*A8-A15*).
  - Port 3 → các tín hiệu điều khiển (*WR\, RD*).
- Sự khác nhau giữa đa hợp và không đa hợp bus địa chỉ và bus dữ liệu:



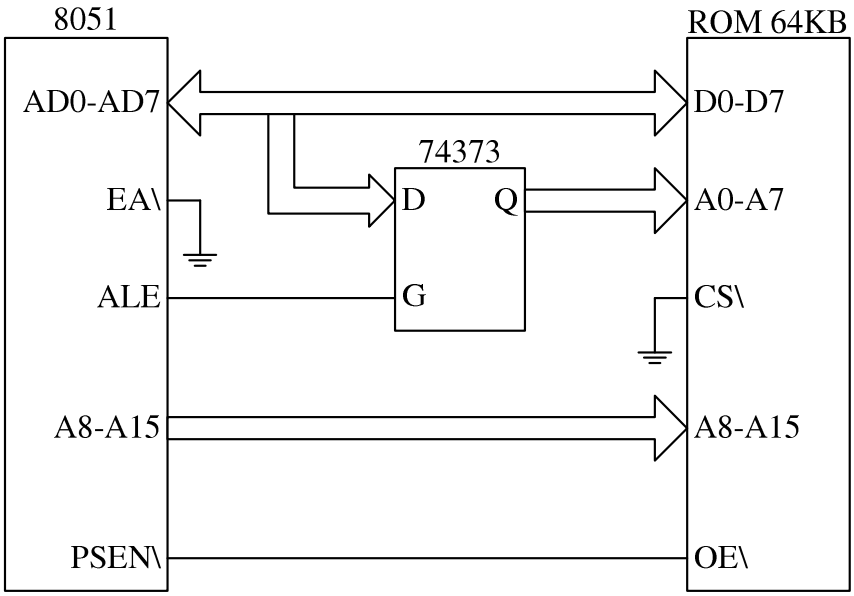
Không đa hợp (24 đường)



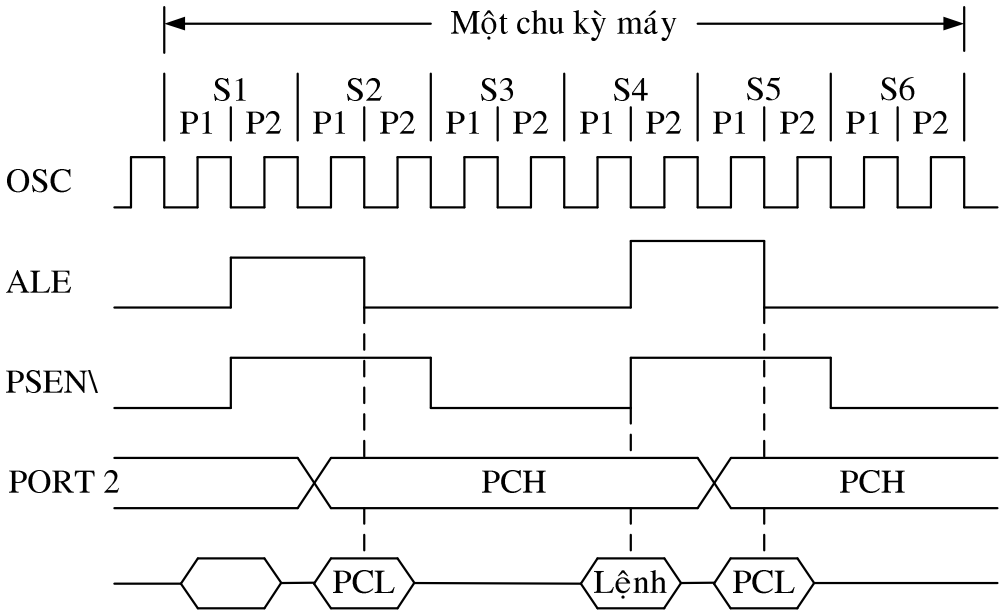
Đa hợp (16 đường)

→ nhằm làm giãm số lượng chân đưa ra ngoài chip → giãm kích thước của chip.

2.1. Kết nối và truy xuất bộ nhớ chương trình ngoài:

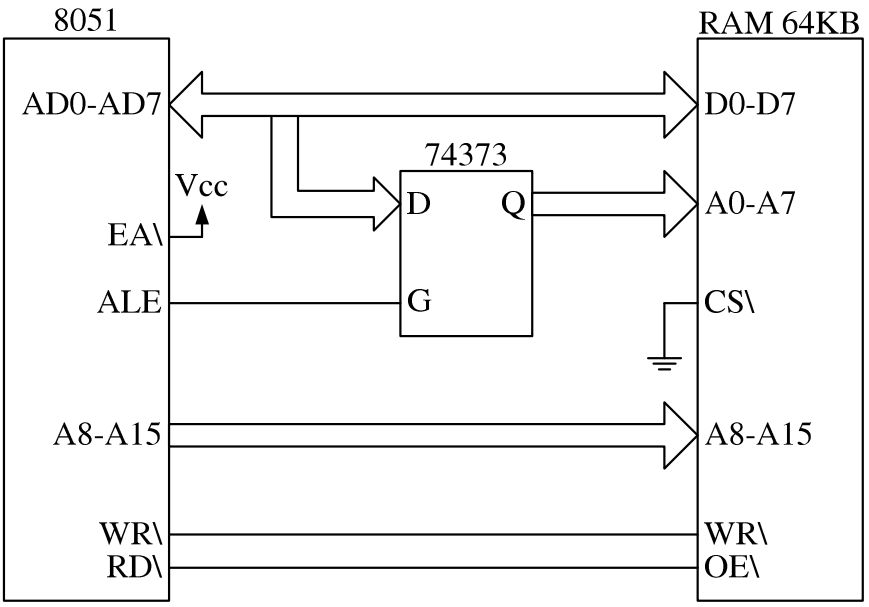


Truy xuất bộ nhớ chương trình bên ngoài

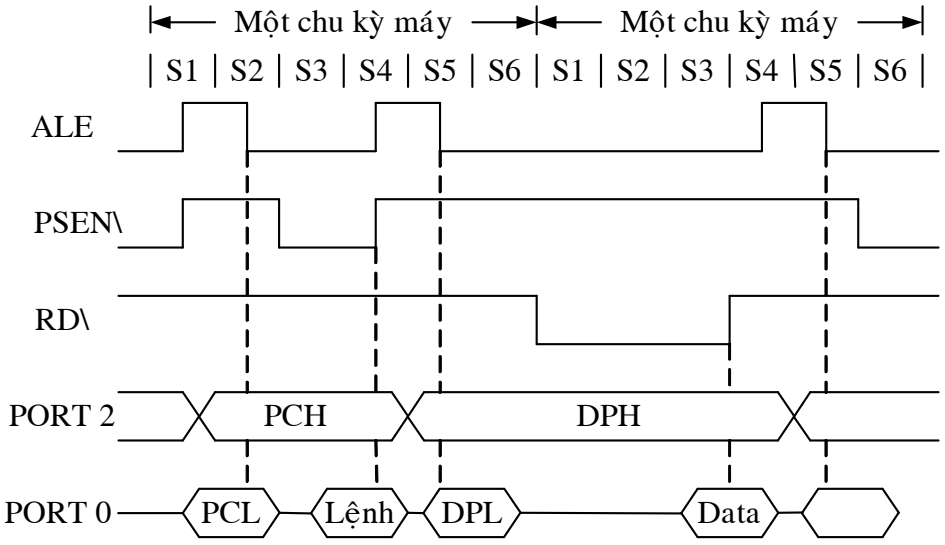


Giản đồ thời gian của chu kỳ tìm nạp lệnh ở bộ nhớ ngoài

2.2. Kết nối và truy xuất bộ nhớ dữ liệu ngoài:



Truy xuất bộ nhớ dữ liệu bên ngoài

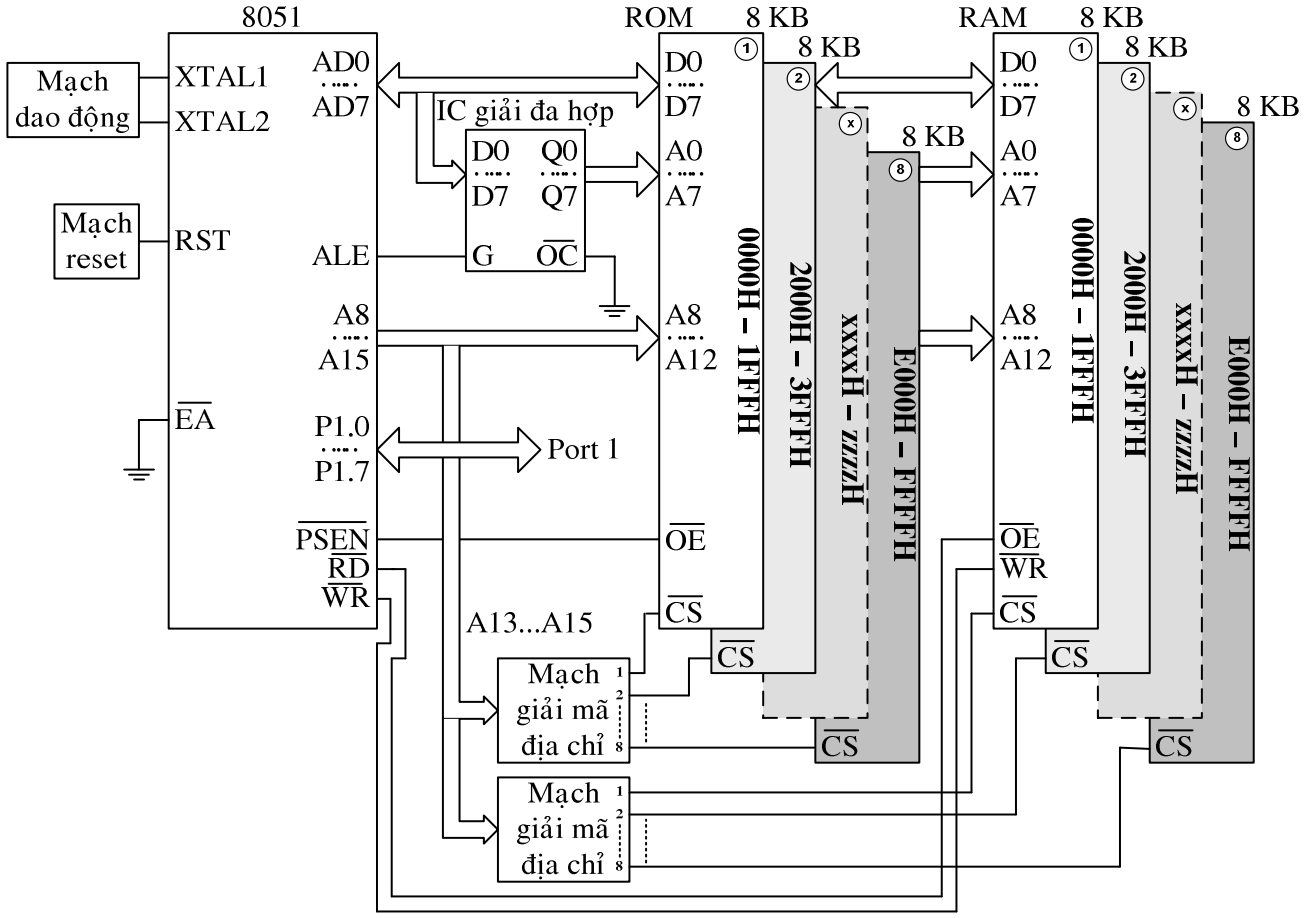


2.3. Giải mã địa chỉ:

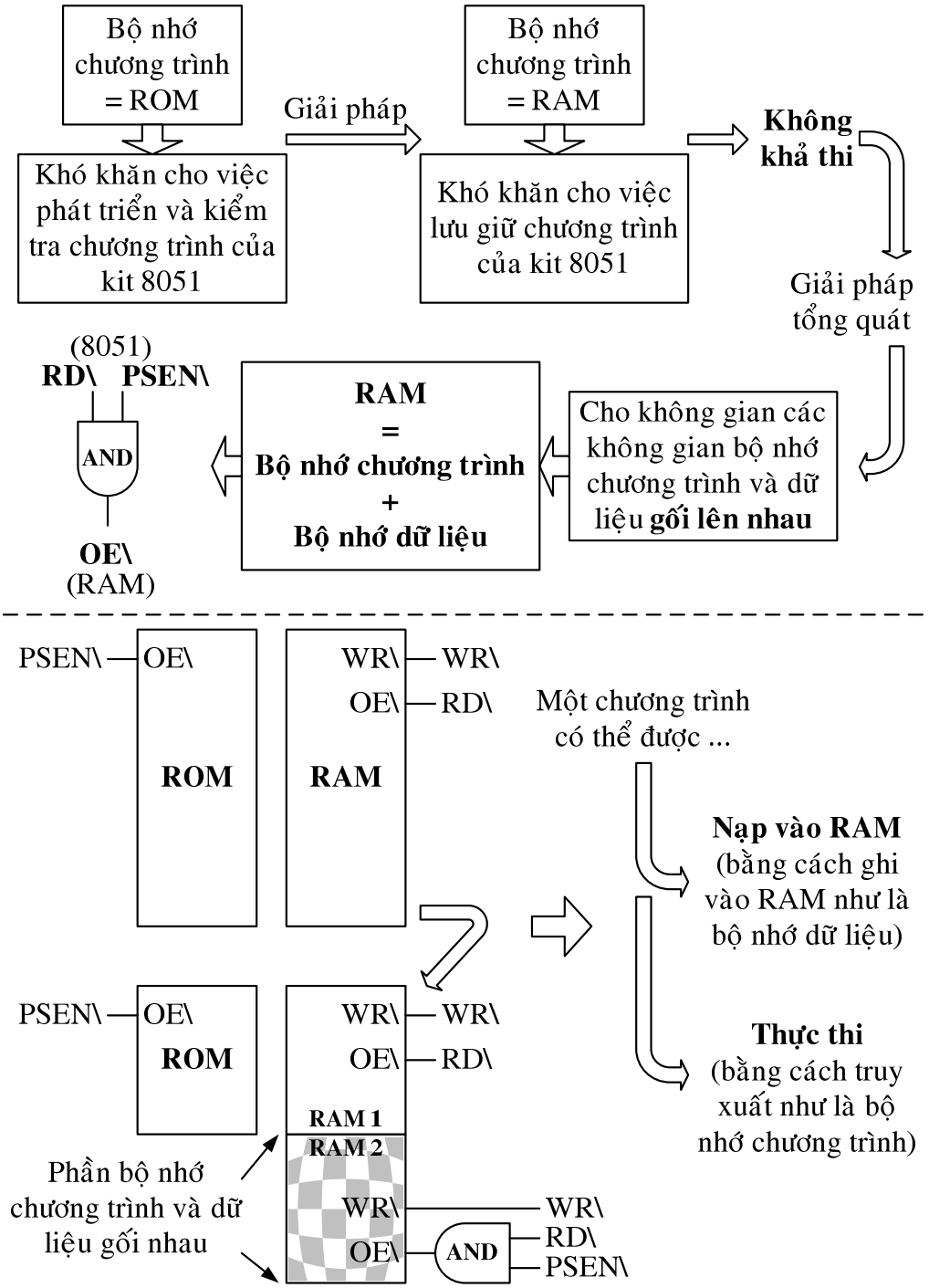
Nếu ta trường hợp ROM và RAM được kết hợp từ nhiều bộ nhớ có dung lượng nhỏ hoặc cả hai giao tiếp với chip 8051 thì ta cần phải giải mã địa chỉ. Việc giải mã địa chỉ này cũng cần cho hầu hết các bộ vi xử lý (*xem thêm trong "**Chương phụ lục 1: Giải mã địa chỉ.**"*).

Ví dụ nếu các ROM và RAM có dung lượng 8KB được sử dụng thì tầm địa chỉ mà chip 8051 quản lý (*0000H – FFFFH*) cần phải được giải mã thành từng đoạn 8 KB để chip có thể chọn từng IC nhớ trên các giới hạn 8KB tương ứng: IC1: 0000H – 1FFFH, IC2: 2000H – 3FFFH, …

IC chuyên dùng cho việc tạo tín hiệu giải mã là 74HC138, các ngõ ra của IC này lần lượt được nối với các ngõ vào chọn chip CS\ tương ứng của các IC nhớ để cho phép các IC nhớ hoạt động (*tại một thời điểm chỉ có một IC nhớ được phép hoạt động*). Cần lưu ý là do các đường cho phép IC nhớ hoạt động riêng lẽ cho từng loại (*PSEN\ cho bộ nhớ chương trình, RD\ và WR\ cho bộ nhớ dữ liệu*) nên 8051 có thể quản lý không gian nhớ lên đến 64KB cho ROM và 64KB cho RAM.



2.4. Các không gian nhớ chương trình và dữ liệu gối nhau:



RAM 1: đóng vai trò là bộ nhớ dữ liệu.
RAM 2: đóng vai trò là bộ nhớ chương trình + bộ nhớ dữ liệu.

V. HOẠT ĐỘNG RESET:

Chân RST = 0 → Chip 8051 hoạt động bình thường.
Chân RST = 1 → Chip 8051 được reset.

✓ **Lưu ý:**

- Để hoàn tất quá trình reset thì chân RST phải ở mức cao tối thiểu là 2 chu kỳ máy và sau đó chuyển xuống mức thấp.
- Nội dung của RAM trong chip không bị ảnh hưởng bởi hoạt động reset.
- Sau khi reset, việc thực thi chương trình luôn luôn bắt đầu ở vi trí đầu tiên trong bộ nhớ chương trình: ***Địa chỉ 0000H***.
- Trạng thái của các thanh ghi sau khi reset hệ thống:
  - Bộ đếm chương trình (*PC*) 0000H
  - Thanh ghi A 00H
  - Thanh ghi B 00H
  - Thanh ghi PSW 00H
  - Thanh ghi SP 07H
  - Thanh ghi DPTR 0000H
  - Port 0 – Port 3 FFH
  - Thanh ghi IP xxx00000B
  - Thanh ghi IE 0xx00000B
  - Các thanh ghi định thời 00H
  - Thanh ghi SCON 00H
  - Thanh ghi SBUF 00H
  - Thanh ghi PCON (*HMOS*) 0xxxxxxxB
  - Thanh ghi PCON (*CMOS*) 0xxx0000B

VI. PHẦN BÀI TẬP:

<u>*Bài 1:*</u> Sử dụng một vi mạch 74138 và các cổng cần thiết để thiết kế mạch giải mã địa chỉ tạo ra các tín hiệu chọn chip tương ứng các vùng địa chỉ sau:

Tín hiệu chọn chip	Vùng địa chỉ	Đặc tính truy xuất
$\overline{CS0}$	0000H - 3FFFH	$\overline{PSEN}$
$\overline{CS1}$	4000H - 7FFFH	$\overline{PSEN}$
$\overline{CS2}$	6000H - 7FFFH	$\overline{RD}$, $\overline{WR}$
$\overline{CS3}$	8000H - 87FFH	$\overline{RD}$
$\overline{CS4}$	8800H - 8FFFH	$\overline{WR}$

Bài 2: Sử dụng một vi mạch 74138 và các cổng cần thiết để thiết kế mạch giải mã địa chỉ tạo ra các tín hiệu chọn chip tương ứng các vùng địa chỉ sau:

Tín hiệu chọn chip	Vùng địa chỉ	Đặc tính truy xuất
$\overline{CS0}$	9800H - 9BFFH	$\overline{PSEN}$
$\overline{CS1}$	9800H - 9BFFH	$\overline{RD}$, $\overline{WR}$
$\overline{CS2}$	9C00H - 9DFFH	$\overline{RD}$, $\overline{WR}$
$\overline{CS3}$	9E00H - 9EFFH	$\overline{RD}$, $\overline{WR}$

Bài 3: Chỉ dùng một vi mạch 74138 (*không dùng thêm cổng*), thiết kế mạch giải mã địa chỉ tạo ra một tín hiệu chọn chip /CS tương ứng tầm địa chỉ F000H-F3FFH.

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

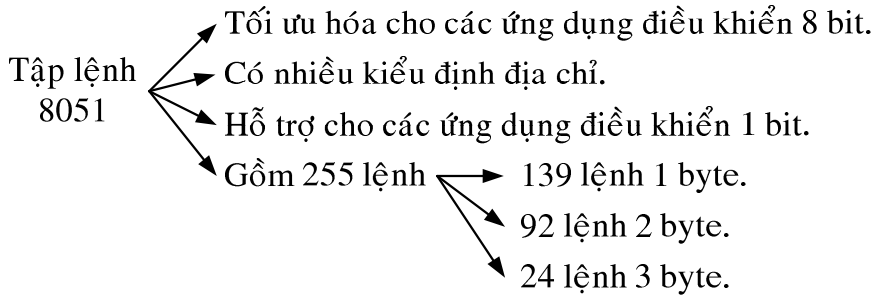
GIÁO TRÌNH VI XỬ LÝ

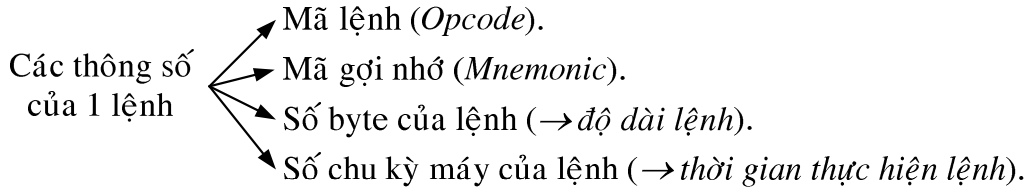
CHƯƠNG 3

TẬP LỆNH CỦA 8051

CHƯƠNG 3
TẬP LỆNH CỦA 8051

I. MỞ ĐẦU:



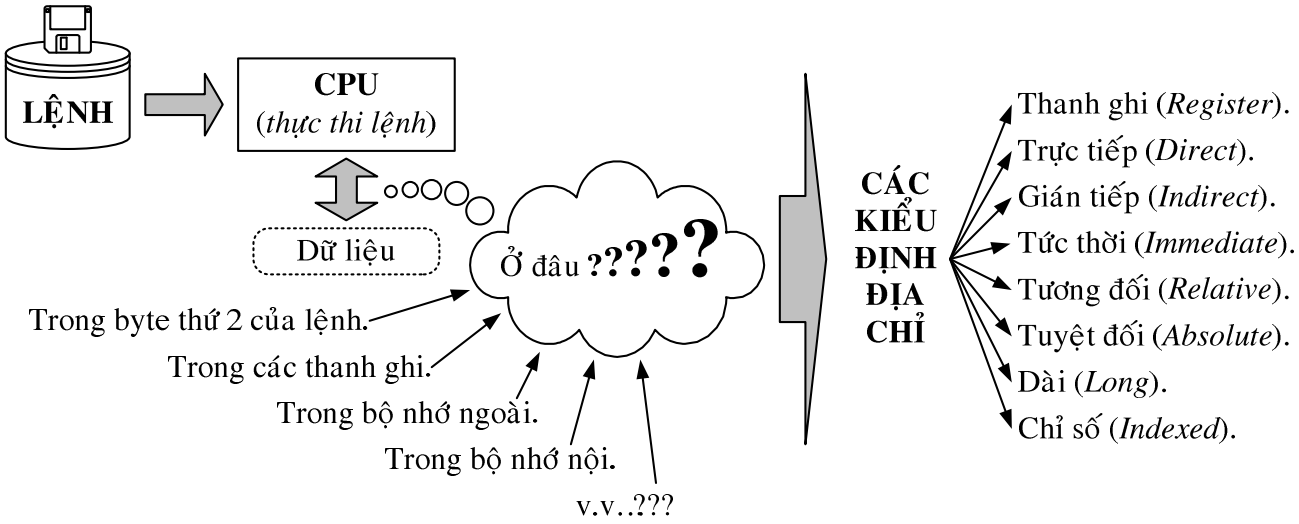


Khuông dạng tổng quát của một dòng lệnh:

[LABEL:] MNEMONIC [OPERAND][,OPERAND]… [;COMMENT]

- Nhãn (*Label*): biểu thị địa chỉ của dòng lệnh (*hoặc dữ liệu*) theo sau, được dùng trong trường toán hạng của lệnh nhảy, lệnh rẽ nhánh (*SJMP **AAA**; ACALL **BBB**; CJNE A, #35H, **LOOP**; JNB P3.1, **TEST_1**…*).
  - ✓ **Lưu ý về nhãn:**
    - Do người lập trình tự đặt (*không được trùng với từ khoá, mã gợi nhớ, chỉ dẫn, toán tử hoặc ký hiệu tiền định nghĩa*).
    - Bắt đầu bằng ký tự chữ, dấu chấm hỏi (?), dấu gạch dưới (_).
    - Dài tối đa 31 ký tự.
    - Kết thúc bằng dấu hai chấm (:).
- Mã gợi nhớ (*Mnemonic*): biểu diễn các mã của lệnh hoặc các chỉ dẫn của chương trình dịch hợp ngữ (*Mã gợi nhớ: **ADD**, **SUBB**, **INC**, …; Chỉ dẫn: **ORG**, **EQU**, **DB**, …*).
- Toán hạng (*Operand*): chứa địa chỉ hoặc dữ liệu mà lệnh sẽ sử dụng. Số lượng toán hạng trong một dòng lệnh phụ thuộc vào từng dòng lệnh (***RET** – không toán hạng, **INC A** – một toán hạng, **ADD A, R0** – hai toán hạng, **CJNE A, #12H, ABC** – ba toán hạng*).
  - ✓ **Lưu ý về toán hạng:** trong các lệnh có 2 toán hạng thì toán hạng đầu tiên còn được gọi là ***toán hạng đích*** (*Destination*), toán hạng thứ hai còn được gọi là ***toán hạng nguồn*** (*Source*).
- Chú thích (*Comment*): làm cho rõ nghĩa cho chương trình. Các chú thích phải nằm trên cùng một dòng và bắt đầu bằng dấu chấm phẩy (;). Các chú thích nếu nằm trên nhiều dòng thì mỗi dòng cũng phải bắt đầu bằng dấu chấm phẩy (;).
  - ✓ **Lưu ý:** Chi tiết về phần này xem thêm tại *"**Chương 7: Lập trình hợp ngữ**" trong sách "**Họ vi điều khiển – Tống Văn On**"*.

II. CÁC KIỂU ĐỊNH ĐỊA CHỈ (ADDRESSING MODE):



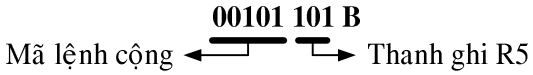
1. Định địa chỉ thanh ghi (*Register Addressing*):

- Được dùng để truy xuất dữ liệu trong các thanh ghi từ **R0** đến **R7**.
- Số byte của lệnh: 1 byte.
- Cấu trúc lệnh:

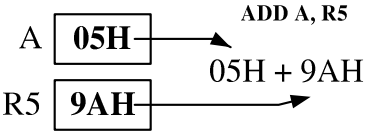
Opcode	n	n	n

- *Ví dụ:* **ADD A, R5** ⇒ Lệnh cộng nội dung thanh ghi A với nội dung thanh ghi R5. (*Giả sử: (A)=05H, (R5)=9AH*).

⇒ Mã lệnh:



⇒ Mô tả lệnh:



- Ngoài ra, một số trường hợp đặc biệt kiểu định địa chỉ này cũng dùng để truy xuất dữ liệu trong các thanh ghi như: thanh ghi chứa **A**, thanh ghi con trỏ dữ liệu **DPTR**, thanh ghi bộ đếm chương trình **PC**, cờ nhớ **C** và cặp thanh ghi **AB**.
- *Ví dụ:* **INC A** ⇨ Lệnh tăng nội dung thanh ghi A.
  INC DPTR ⇨ Lệnh tăng nội dung thanh ghi DPTR.

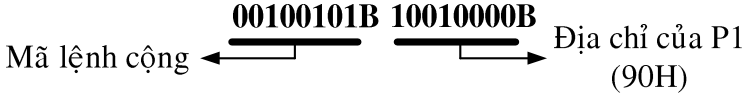
2. Định địa chỉ trực tiếp (*Direct Addressing*):

- Được dùng để truy xuất dữ liệu trong các ô nhớ (***00H - FFH***) hay trong các thanh ghi (***A, B, P0–P3, DPH, DPL,…***) của bộ nhớ bên trong chip.
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh:

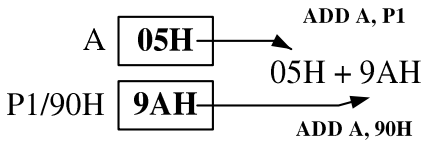
Opcode	Direct address

- *Ví dụ:* **ADD A, P1 ⇔ ADD A, 90H** ⇒ Lệnh cộng nội dung thanh ghi A với nội dung thanh ghi port 1 hay ô nhớ 90H. (*Giả sử: (A) = 05H, (P1) = (90H) = 9AH*).

⇒ Mã lệnh:

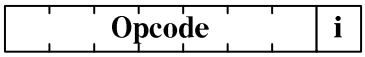


⇒ Mô tả lệnh:



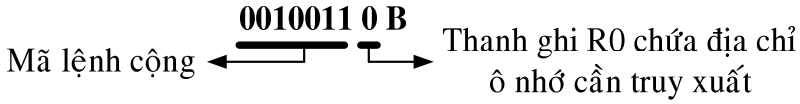
3. Định địa chỉ gián tiếp (*Indirect Addressing*):

- Được dùng để truy xuất dữ liệu trong các ô nhớ "gián tiếp" của bộ nhớ bên trong chip. Các thanh ghi **R0** và **R1** được dùng để chứa địa chỉ của các ô nhớ gián tiếp (*00H - FFH*) trong chip. Lưu ý rằng, trước các thanh ghi R0, R1 cần phải có dấu "**@**".
- Số byte của lệnh: 1 byte.
- Cấu trúc lệnh:



- *Ví dụ:* **ADD A, @R0** ⇒ Lệnh cộng nội dung thanh ghi A với nội dung ô nhớ có địa chỉ chứa trong thanh ghi R0. (*Giả sử: (A) = 05H, (R0) = 3BH, (3BH) = 9AH*).

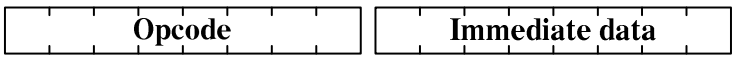
⇒ Mã lệnh:



⇒ Mô tả lệnh:

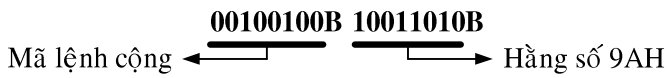
4. Định địa chỉ tức thời (*Immediate Addressing*):

- Được dùng để truy xuất một hằng số (***giá trị biết trước***) thay vì là một biến (***giá trị không biết trước***) như các kiểu định địa chỉ trên. Lưu ý rằng, trước dữ liệu tức thời cần phải có dấu "**#**". Chế độ định địa chỉ tức thời có thể dùng để nạp dữ liệu vào mọi ô nhớ và thanh ghi bất kỳ (*đối với thanh ghi 8 bit:* ***#00H - #0FFH****, đối với thanh ghi 16 bit:* ***#0000H - #0FFFH***).
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh:

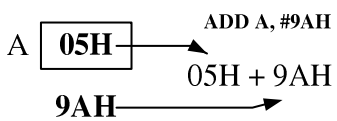


- *Ví dụ:* **ADD A, #9AH** ⇒ Lệnh cộng nội dung thanh ghi A với giá trị 9AH. (*Giả sử: (A) = 05H*).

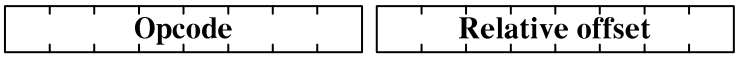
⇒ Mã lệnh:



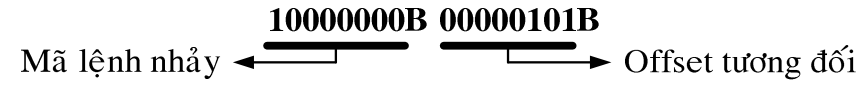
⇒ Mô tả lệnh:



5. Định địa chỉ tương đối (*Relative Addressing*):

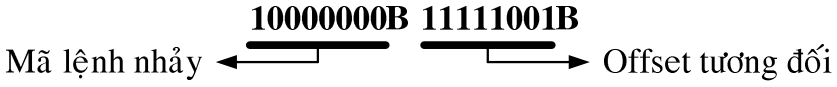
- Được sử dụng cho các lệnh nhảy.
- Địa chỉ tương đối (*hay offset*) là một giá trị **8 bit có dấu**.
- Tầm nhảy giới hạn là: **-128 byte … 127 byte** từ vị trí của lệnh tiếp theo sau lệnh nhảy.
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh: 

- *Ví dụ 1:* **SJMP AAA** ⇒ Lệnh nhảy đến nhãn AAA (*Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 0107H, lệnh SJMP nằm trong bộ nhớ tại địa chỉ 0100H và 0101H*).

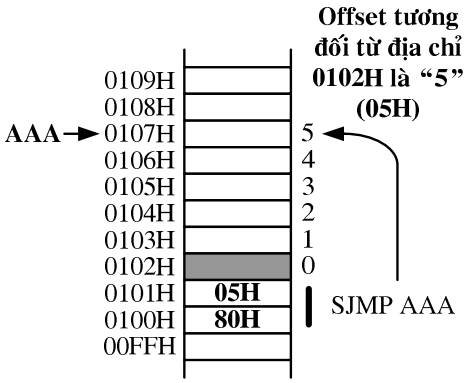
⇒ Mã lệnh: 

⇒ Mô tả lệnh: *xem hình 3.5.2.1*.

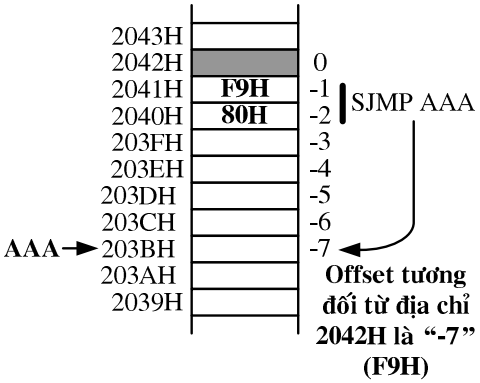
- *Ví dụ 2:* **SJMP AAA** ⇒ Lệnh nhảy đến nhãn AAA (*Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 203BH, lệnh SJMP nằm trong bộ nhớ tại địa chỉ 2040H và 2041H*).

⇒ Mã lệnh: 

⇒ Mô tả lệnh: *xem hình 3.2.5.2*.

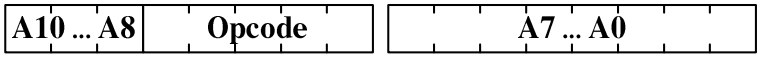


Hình 3.2.5.1

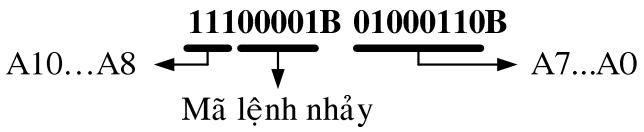


Hình 3.2.5.2

6. Định địa chỉ tuyệt đối (*Absolute Addressing*):

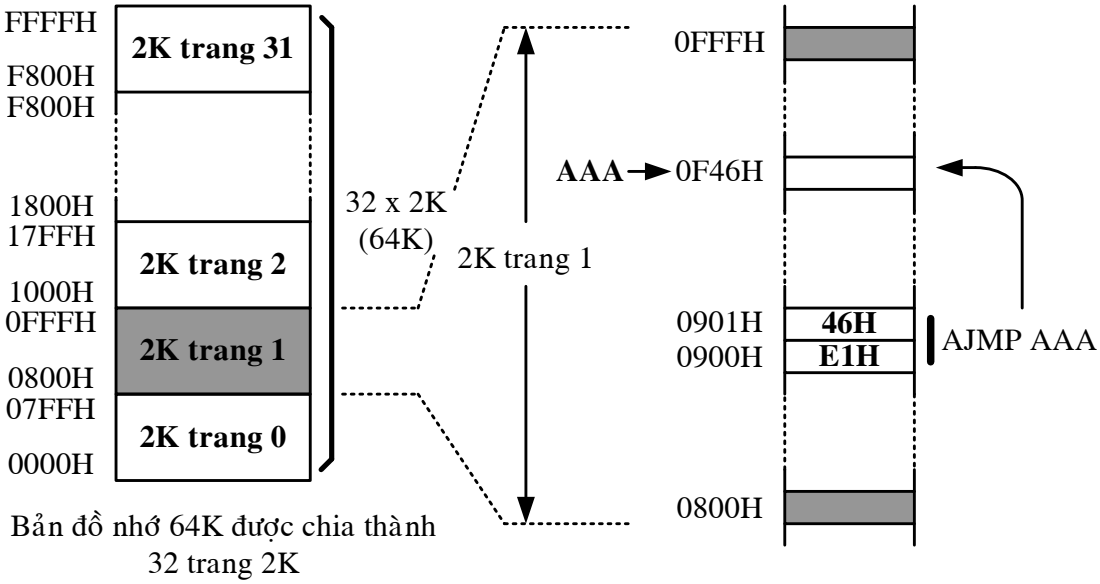
- Được sử dụng cho các lệnh ACALL và AJMP.
- Địa chỉ tuyệt đối là một giá trị **11 bit**.
- Tầm nhảy giới hạn là: **trong cùng trang 2K hiện hành** (*trang 2K chứa lệnh nhảy*).
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh: 

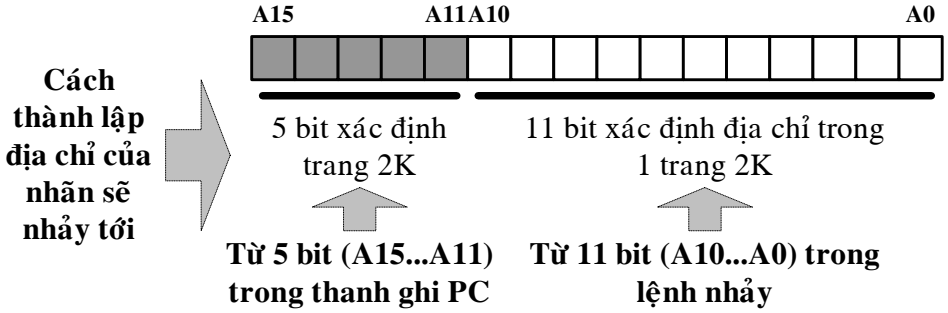
- *Ví dụ:* **AJMP AAA** ⇒ Lệnh nhảy đến nhãn AAA (*Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 0F46H, lệnh AJMP nằm trong bộ nhớ tại địa chỉ 0900H và 0901H*).



⇒ Mã lệnh:

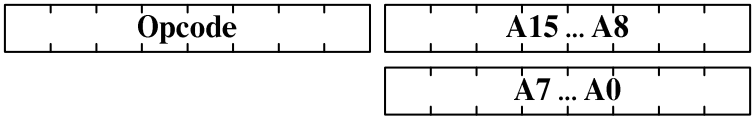
⇒ Mô tả lệnh:





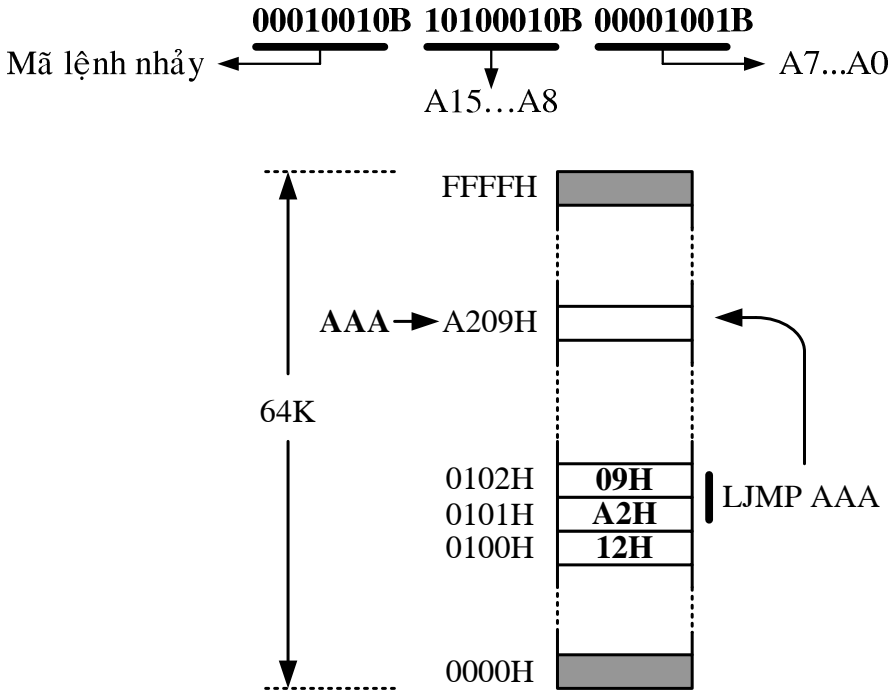
7. Định địa chỉ dài (*Long Addressing*):

- Được sử dụng cho các lệnh LCALL và LJMP.
- Địa chỉ dài là một giá trị **16 bit**.
- Tầm nhảy giới hạn là: **toàn bộ không gian nhớ 64K**.
- Số byte của lệnh: 3 byte.
- Cấu trúc lệnh:



- *Ví dụ:* **LJMP AAA** ⇒ Lệnh nhảy đến nhãn AAA (*Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ A209H, lệnh LJMP nằm trong bộ nhớ tại địa chỉ 0100H, 0101H và 0102H*).

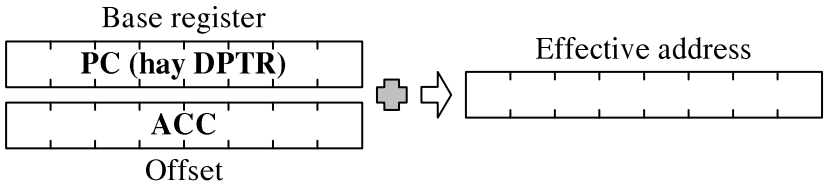
⇒ Mã lệnh:



⇒ Mô tả lệnh:

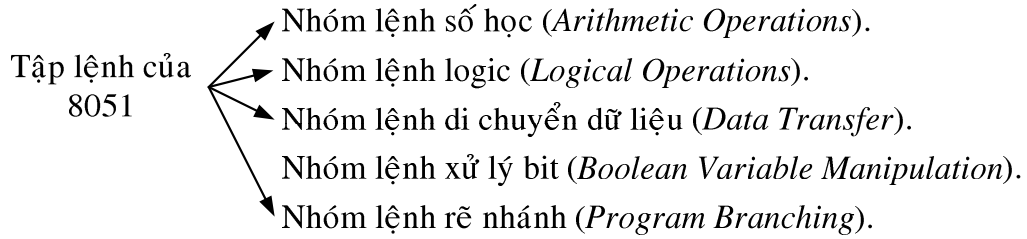
8. Định địa chỉ chỉ số (*Indexed Addressing*):

- Được dùng trong các ứng dụng cần tạo các bảng nhảy hay các bảng tìm kiếm. Kiểu định địa chỉ này dùng một thanh ghi nền (***PC*** *hay* ***DPTR***) kết hợp với một *offset* (***A***) để tạo thành dạng địa chỉ hiệu dụng cho lệnh.
- Số byte của lệnh: 1 byte.
- Cấu trúc lệnh:



- *Ví dụ:* **JMP @A+DPTR** ⇒ Lệnh nhảy gián tiếp.

III. TẬP LỆNH CỦA 8051 (*8051 INSTRUCTION SET*):



Một số ký hiệu dùng trong lệnh:

Rn	Địa chỉ thanh ghi sử dụng (*R0 – R7*).
direct	Địa chỉ trực tiếp của một byte trong RAM nội (*00H-FFH*)
@Ri	Địa chỉ gián tiếp sử dụng (*R0 hoặc R1*).
source	Toán hạng nguồn (*Rn, direct hoặc @Ri*).
dest	Toán hạng đích (*Rn, direct hoặc @Ri*).
#data	Hằng số 8 bit (*#00H - #0FFH*).
#data16	Hằng số 16 bit (*#0000H - #0FFFFH*).
bit	Địa chỉ trực tiếp của một bit (*địa chỉ bit*).
rel	*Offset* 8 bit có dấu.
addr11	Địa chỉ 11 bit.
addr16	Địa chỉ 16 bit.
←	Được thay thế bởi …
( )	Nội dung của …
(( ))	Nội dung được chứa bởi …
rrr	Thanh ghi của dãy thanh ghi (*000 = R0, 001 = R1, ..., 111 = R7*).
i	Địa chỉ gián tiếp sử dụng R0 (*i = 0*) hoặc R1 (*i = 1*).
dddddddd	Các bit dữ liệu.
aaaaaaaa	Các bit địa chỉ.
eeeeeeee	Địa chỉ tương đối.

Một số lưu ý khi lập trình bộ vi điều khiển 8051:

- Để thông báo đó là một giá trị tức thời thì cần phải đặt thêm ký hiệu "**#**" vào trước giá trị đó. Nếu không có ký hiệu "**#**" thì giá trị đó được hiểu là địa chỉ của ô nhớ.

```
MOV  A, #12H      ;Nạp giá trị 12H vào thanh ghi A.
MOV  A, 12H       ;Sao chép nội dung của ô nhớ có địa
                  ;chỉ 12H vào thanh ghi A.
```

Ở đây ta cũng nên lưu ý rằng nếu thiếu ký hiệu "#" thì lệnh trên cũng không gây ra lỗi trong quá trình biên dịch. Vì trình dịch hợp ngữ cho đó là một lệnh hợp lệ. Tuy nhiên, kết quả lập trình sẽ không đúng như ý muốn của người lập trình.

- Các giá trị tức thời nếu có thành phần chữ (*A, B, C, ..., F*) đứng đầu thì cần phải ***thêm số 0*** vào trước thành phần chữ và sau ký hiệu "#". Việc này để báo rằng thành phần chữ đó là một số HEX chứ không phải là một ký tự.

```
MOV  A, #BH       ;Thiếu số 0 → gây lỗi khi biên dịch.
MOV  A, #0BH      ;Thêm số 0 → đúng.
MOV  A, #F9H      ;Thiếu số 0 → gây lỗi khi biên dịch.
MOV  A, #0F9H     ;Thêm số 0 → đúng.
```

Ở đây ta cũng nên lưu ý rằng việc thiếu số 0 thêm vào này sẽ gây lỗi trong quá trình biên dịch đối với các chương trình biên dịch cũ. Ngày nay, một số phần mềm biên dịch đã hỗ trợ việc này. Điều này có nghĩa là ta có thể thêm hay không thêm số 0 vào thì đều không ảnh hưởng gì đến quá trình biên dịch (*không gây ra lỗi khi biên dịch*).

- Trong lệnh, các giá trị tức thời hay địa chỉ của ô nhớ có thể được biểu diễn dưới bất kỳ dạng nào BIN (*nhị phân*), DEC (*thập phân*) hay HEX (*thập lục phân*).
  - *Địa chỉ ô nhớ:* các câu lệnh sau đây là tương đương nhau:

```
MOV   A, 64H          ;Sao chép nội dung của ô nhớ có địa
                      ;chỉ 64H vào thanh ghi A.
MOV   A, 100          ;Sao chép nội dung của ô nhớ có địa
                      ;chỉ 64H vào thanh ghi A.
MOV   A, 01100100B    ;Sao chép nội dung của ô nhớ có địa
                      ;chỉ 64H vào thanh ghi A.
```

  - *Giá trị tức thời:* các câu lệnh sau đây là tương đương nhau:

```
MOV   A, #0C9H        ;Nạp giá trị C9H vào thanh ghi A.
MOV   A, #201         ;Nạp giá trị C9H vào thanh ghi A.
MOV   A, #11001001B   ;Nạp giá trị C9H vào thanh ghi A.
```

Lưu ý các hậu tố đi kèm tương ứng cho từng dạng: **B** – dạng BIN (*nhị phân*), **H** – dạng HEX (*thập lục phân*), **D** hoặc **không có hậu tố** – dạng DEC (*thập phân*).

- Chuyển một giá trị tức thời hay địa chỉ của ô nhớ lớn hơn khả năng chứa của một thanh ghi thì sẽ gây ra lỗi (***00H-FFH****: cho thanh ghi hoặc ô nhớ 8 bit;* ***0000H-FFFFH****: cho thanh ghi 16 bit - DPTR*).

```
MOV   A, #123H        ;Không hợp lệ vì 123H > FFH.
MOV   A, #214         ;Hợp lệ vì 214 (D6H) < FFH (255).
MOV   A, #0F2H        ;Hợp lệ vì F2H < FFH.
MOV   A, 123H         ;Không hợp lệ vì 123H > FFH.
MOV   A, 200          ;Hợp lệ vì 200 (C8H) < FFH (255).
MOV   DPTR, #123H     ;Hợp lệ vì 123H < FFFFH (16 bit).
```

1. Nhóm lệnh số học:

1.1. Lệnh ADD A, <src-byte>:

- *Chức năng:* Cộng (*Add*).
- *Mô tả:* ADD cộng nội dung của thanh ghi A (***A***) với nội dung của một byte có địa chỉ được chỉ ra trong lệnh (***src-byte***) và đặt kết quả vào thanh ghi A. *Các cờ bị ảnh hưởng*.
  - Cờ **CY = 1** nếu có số nhớ từ bit 7. Ngược lại **CY = 0**.
  - Cờ **AC = 1** nếu có số nhớ từ bit 3. Ngược lại **AC = 0**.
  - Cờ **OV = 1** nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc nếu có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6. Ngược lại **OV = 0**.
  - Khi cộng hai số nguyên không dấu và có dấu:
    - Số không dấu: CY = 1 ⇔ Phép toán có nhớ.
    - Số có dấu: CY = 1 ⇔ Số dương = Số âm + Số âm.
      ⇔ Số âm = Số dương + Số dương.

- *Các dạng lệnh:*

ADD A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00101rrr
Hoạt động	(A) ← (A) + (Rn)

ADD A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	00100101 aaaaaaaa
Hoạt động	(A) ← (A) + (direct)

ADD A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0010011i
Hoạt động	(A) ← (A) + ((Ri))

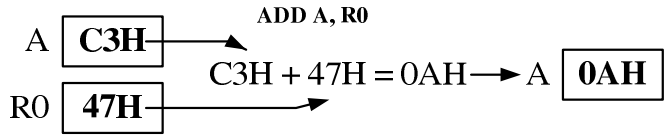
ADD A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	00100100 dddddddd
Hoạt động	(A) ← (A) + #data

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=47H, (P1)=(90H)=AAH, (47H)=D2H.

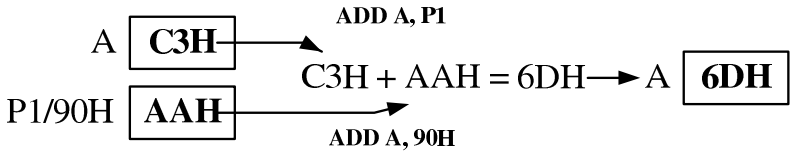
Sau khi thực thi lệnh **ADD A, R0** thì:

(A)=0AH, CY=1, AC=0, OV=0



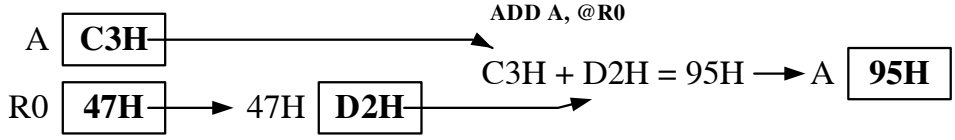
Sau khi thực thi lệnh **ADD A, 90H** hay **ADD A, P1** thì:

(A)=6DH, CY=1, AC=0, OV=1



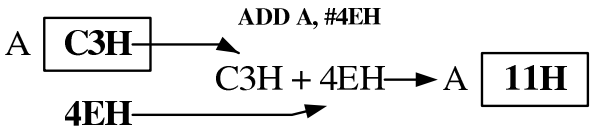
Sau khi thực thi lệnh **ADD A, @R0** thì:

(A)=95H, CY=1, AC=0, OV=0



Sau khi thực thi lệnh **ADD A, #4EH** thì:

(A)=11H, CY=1, AC=1, OV=0



1.2. ADDC A, <src-byte>

- *Chức năng:* Cộng có cờ nhớ (*Add with Carry*).
- *Mô tả:* ADDC cộng đồng thời nội dung của thanh ghi A (***A***) với nội dung của byte có địa chỉ được chỉ ra trong lệnh (***src-byte***) và cờ nhớ (***CY***), đặt kết quả vào thanh ghi A. *Các cờ bị ảnh hưởng*.
  - Cờ **CY = 1** nếu có số nhớ từ bit 7. Ngược lại **CY = 0**.
  - Cờ **AC = 1** nếu có số nhớ từ bit 3. Ngược lại **AC = 0**.
  - Cờ **OV = 1** nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc nếu có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6. Ngược lại **OV = 0**.
  - Khi cộng hai số nguyên không dấu và có dấu:
    - Số không dấu: CY = 1 ⇔ Phép toán có nhớ.
    - Số có dấu: CY = 1 ⇔ Số dương = Số âm + Số âm.
      ⇔ Số âm = Số dương + Số dương.
- *Các dạng lệnh:*

ADDC A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00110rrr
Hoạt động	(A) ← (A) + (C) + (Rn)

ADDC A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	00110101 aaaaaaaa
Hoạt động	(A) ← (A) + (C) + (direct)

ADDC A,@Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0011011i
Hoạt động	(A) ← (A) + (C) + ((Ri))

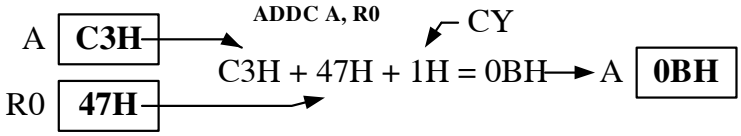
ADDC A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	00110100 dddddddd
Hoạt động	(A) ← (A) + (C) + # data

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=47H, (P1)=(90H)=AAH, (47H)=D2H và cờ CY=1.

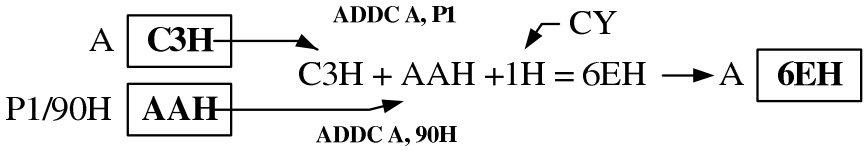
Sau khi thực thi lệnh **ADDC A, R0** thì:

(A)=0BH, CY=1, AC=0, OV=0



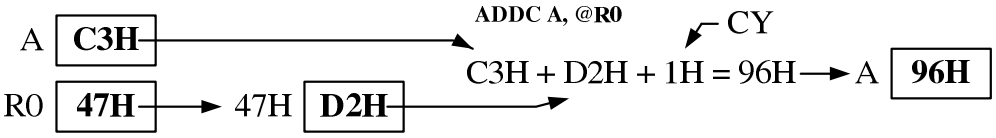
Sau khi thực thi lệnh **ADDC A, 90H** hay **ADDC A, P1** thì:

(A)=6DH, CY=1, AC=0, OV=1



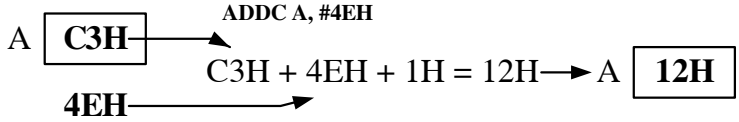
Sau khi thực thi lệnh **ADDC A, @R0** thì:

(A)=96H, CY=1, AC=0, OV=0



Sau khi thực thi lệnh **ADDC A, #4EH** thì:

(A)=11H, CY=1, AC=1, OV=0



1.3. SUBB A, <src-byte>

- *Chức năng:* Trừ có số mượn (*Subtract with Borrow*).
- *Mô tả:* SUBB trừ nội dung của thanh ghi A (***A***) với nội dung của byte có địa chỉ được chỉ ra trong lệnh (***src-byte***) cùng với cờ nhớ và cất kết quả vào thanh ghi A. *Các cờ bị ảnh hưởng*.
    - Cờ **CY = 1** nếu có số mượn cho bit 7. Ngược lại **CY = 0**.
    - Cờ **AC = 1** nếu có số mượn cho bit 3. Ngược lại **AC = 0**.
    - Cờ **OV = 1** nếu có số mượn cho bit 6 nhưng không có số mượn cho bit 7 hoặc nếu có số mượn cho bit 7 nhưng không có số mượn cho bit 6. Ngược lại **OV = 0**.
    - Khi cộng hai số nguyên không dấu và có dấu:
        - Số không dấu: CY = 1 ⇔ Phép toán có mượn.
        - Số có dấu: CY = 1 ⇔ Số dương = Số âm - Số dương.
          ⇔ Số âm = Số dương - Số âm.

- *Các dạng lệnh:*

SUBB A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	10011rrr
Hoạt động	(A) ← (A) – (C) – (Rn)

SUBB A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	10010101 aaaaaaaa
Hoạt động	(A) ← (A) – (C) – (direct)

SUBB A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	1001011i
Hoạt động	(A) ← (A) – (C) – ((Ri))

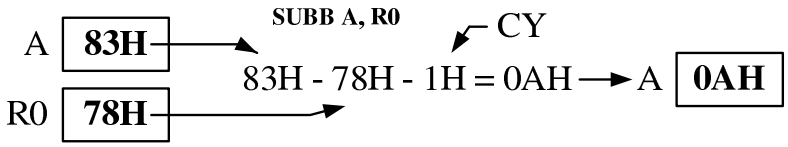
SUBB A, #data

Số byte	1
Số chu kỳ	1
Mã đối tượng	100110100 dddddddd
Hoạt động	(A) ← (A) – (C) – #data

- *Ví dụ:* Cho biết trước (A)=83H, (R0)=78H, (P1)=(90H)=AAH, (78H)=C5H và cờ CY=1.

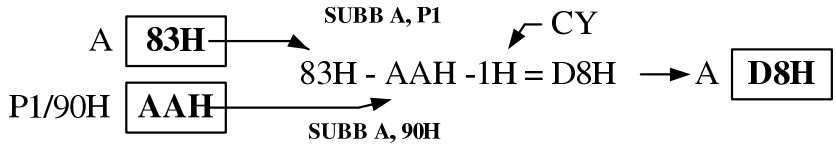
Sau khi thực thi lệnh **SUBB A, R0** thì:

(A)=0AH, CY=0, AC=1, OV=1



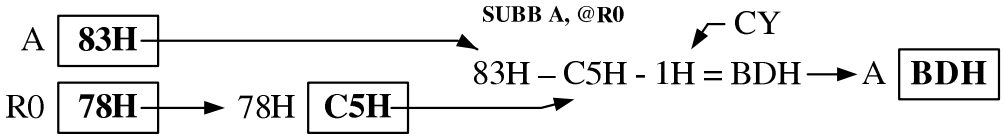
Sau khi thực thi lệnh **SUBB A, 90H** hay **SUBB A, P1** thì:

(A)=D8H, CY=1, AC=1, OV=0



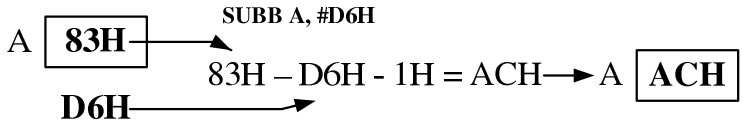
Sau khi thực thi lệnh **SUBB A, @R0** thì:

(A)=BDH, CY=1, AC=1, OV=0



Sau khi thực thi lệnh **SUBB A, #D6H** thì:

(A)=ACH, CY=1, AC=1, OV=0



1.4. INC byte

- *Chức năng:* Tăng thêm 1 (*Increment*).
- *Mô tả:* Tăng nội dung của byte có địa chỉ được chỉ ra trong lệnh (***byte***) thêm 1. *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.
- *Các dạng lệnh:*

INC A

Số byte	1
Số chu kỳ	1
Mã đối tượng	00000100
Hoạt động	(A) ← (A) + 1

INC Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00001rrr
Hoạt động	(Rn) ← (Rn) + 1

INC direct

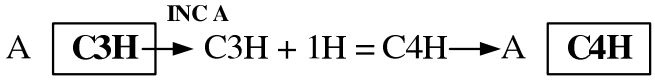
Số byte	2
Số chu kỳ	1
Mã đối tượng	00000101 aaaaaaaa
Hoạt động	(direct) ← (direct) + 1

INC @Ri

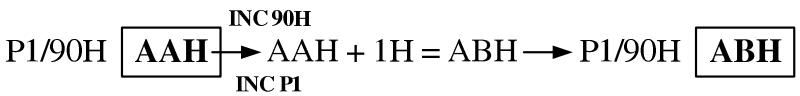
Số byte	1
Số chu kỳ	1
Mã đối tượng	0000011i
Hoạt động	((Ri)) ← ((Ri)) + 1

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=69H, (P1)=(90H)=AAH, (69H)=7FH.

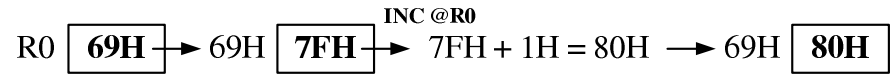
Sau khi thực thi lệnh **INC A** thì: (A)=C4H



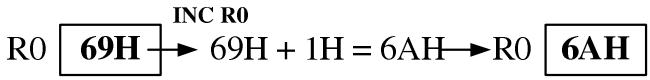
Sau khi thực thi lệnh **INC 90H** hay **INC P1** thì: (P1)=(90H)=ABH



Sau khi thực thi lệnh **INC @R0** thì: (@R0)=(69H)=80H



Sau khi thực thi lệnh **INC R0** thì: R0=6AH



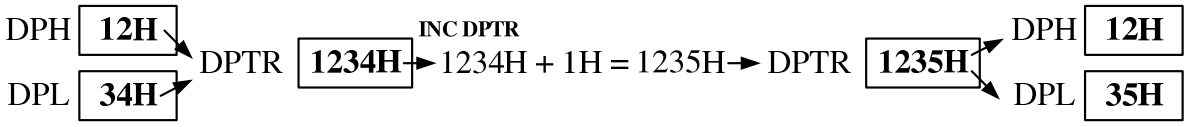
1.5. INC DPTR

- *Chức năng:* Tăng con trỏ dữ liệu (*Increment Data Pointer*).
- *Mô tả:* Tăng nội dung của thanh ghi con trỏ dữ liệu 16-bit thêm 1. Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	2
Mã đối tượng	10100011
Hoạt động	(DPTR) ← (DPTR) + 1

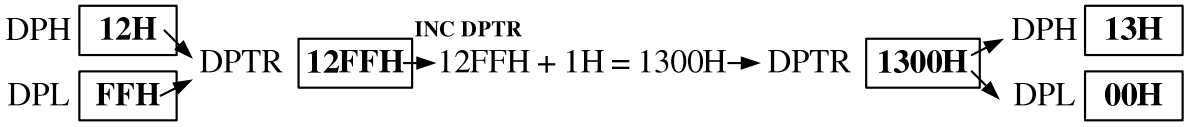
- *Ví dụ 1:* Cho biết trước (DPTR)=1234H.

Sau khi thực thi lệnh **INC DPTR** thì:
(DPTR)=1235H với (DPH)=12H và (DPL)=35H



- *Ví dụ 2:* Cho biết trước (DPH)=12H và (DPL)=FFH.

Sau khi thực thi lệnh **INC DPTR** thì:
(DPTR)=1300H với (DPH)=13H và (DPL)=00H



- *Lưu ý:* Không có lệnh giãm nội dung của DPTR (***DEC DPTR***). Nếu muốn giãm nội dung của DPTR ta phải viết một đoạn chương trình con để thực hiện điều này. Chương trình con được minh họa như sau:

```
DEC_DPTR:                       ;Chương trình con giãm DPTR.
        PUSH  ACC               ;Cất tạm giá trị ACC.
        DEC   DPL               ;Giãm byte thấp của DPTR.
        MOV   A, DPL            ;So sánh byte thấp của DPTR
        CJNE  A,#0FFH, SKIP     ;với FFH.
        DEC   DPH               ;Giãm byte cao của DPTR.
SKIP:
        POP   ACC               ;Phục hồi giá trị ACC.
        RET
```

1.6. DEC byte

- *Chức năng:* Giãm bớt 1 (*Decrement*).
- *Mô tả:* Giãm nội dung của byte có địa chỉ được chỉ ra trong lệnh (***byte***) bớt 1. *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.
- *Các dạng lệnh:*

DEC A

Số byte	1
Số chu kỳ	1
Mã đối tượng	00010100
Hoạt động	(A) ← (A) – 1

DEC Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00011rrr
Hoạt động	(Rn) ← (Rn) – 1

DEC direct

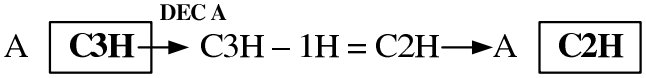
Số byte	2
Số chu kỳ	1
Mã đối tượng	00010101 aaaaaaaa
Hoạt động	(direct) ← (direct) – 1

DEC @Ri

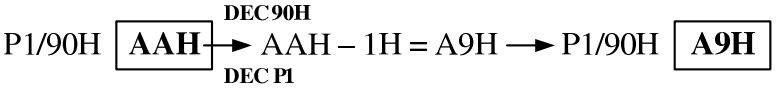
Số byte	1
Số chu kỳ	1
Mã đối tượng	0001011i
Hoạt động	((Ri)) ← ((Ri)) – 1

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=60H, (P1)=(90H)=AAH, (60H)=7AH.

Sau khi thực thi lệnh **DEC A** thì: (A)=C2H

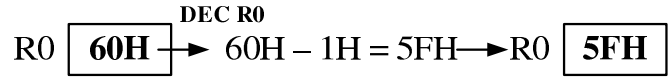


Sau khi thực thi lệnh **DEC 90H** hay **DEC P1** thì: (P1)=(90H)=A9H



Sau khi thực thi lệnh **DEC @R0** thì: (@R0)=(60H)=79H

Sau khi thực thi lệnh **DEC R0** thì: R0=5FH



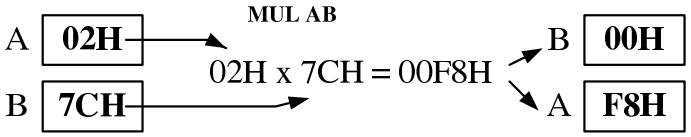
1.7. MUL AB

- *Chức năng:* Nhân (*Multiply*).
- *Mô tả:* MUL AB nhân các số nguyên không dấu 8-bit chứa trong thanh ghi A và thanh ghi B. Tích số là một giá trị 16 bit, **byte thấp** (*8 bit thấp*) được cất trong thanh ghi A còn **byte cao** (*8 bit cao*) được cất trong thanh ghi B.
  Nếu tích số lớn hơn 255 (0FFH) thì cờ tràn OV=1. Cờ nhớ CY luôn luôn bị xóa.

Số byte	1
Số chu kỳ	4
Mã đối tượng	10100100
Hoạt động	(B) ← HIGH BYTE OF (A) × (B) (A) ← LOW BYTE OF (A) × (B)

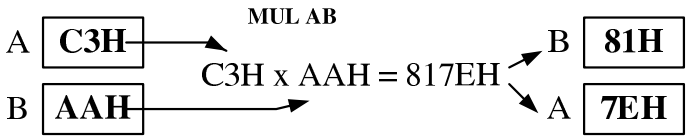
- *Ví dụ 1:* Cho biết trước (A)=02H, (B)=7CH.

Sau khi thực thi lệnh **MUL AB** thì: (B)= 00H, (A)= F8H, CY=0, OV=0.



- *Ví dụ 2:* Cho biết trước (A)=C3H, (B)=AAH.

Sau khi thực thi lệnh **MUL AB** thì: (B)= 81H, (A)= 7EH, CY=0, OV=1.



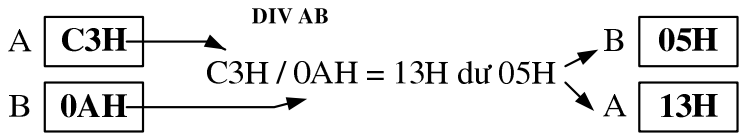
1.8. DIV AB

- *Chức năng:* Chia (*Divide*).
- *Mô tả:* DIV AB chia số nguyên không dấu 8-bit chứa trong thanh ghi A cho số nguyên không dấu 8-bit chứa trong thanh ghi B. **Thương số** được cất trong thanh ghi A còn **số dư** được cất trong thanh ghi B. *Cờ CY và cờ OV bị xoá.*
  Nếu ban đầu B chứa 00H, giá trị trả về trong thanh ghi A và thanh ghi B không được xác định và cờ OV=1. Cờ CY được xóa trong mọi trường hợp.

Số byte	1
Số chu kỳ	4
Mã đối tượng	10000100
Hoạt động	(A) ← QUOTIENT OF (A) / (B) (B) ← REMAINDER OF (A) / (B)

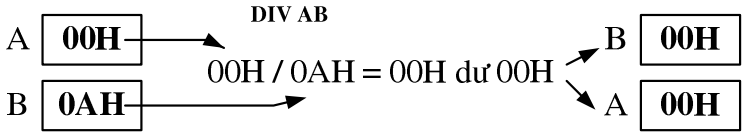
- *Ví dụ 1:* Cho biết trước (A)=C3H, (B)=0AH.

Sau khi thực thi lệnh **DIV AB** thì: (B)= 05H, (A)= 13H, CY=0, OV=0.



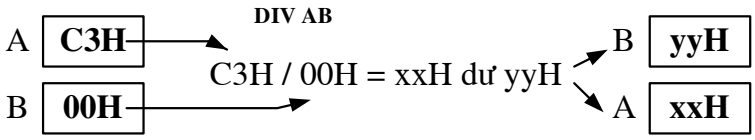
- *Ví dụ 2:* Cho biết trước (A)=00H, (B)=0AH.

Sau khi thực thi lệnh **DIV AB** thì: (B)= 00H, (A)= 00H, CY=0, OV=0.



- *Ví dụ 3:* Cho biết trước (A)=C3H, **(B)=00H**.

Sau khi thực thi lệnh **DIV AB** thì: (B)= yyH, (A)= xxH, CY=0, **OV=1**.



1.9. DA A

- *Chức năng:* Hiệu chỉnh thập phân nội dung của thanh ghi A đối với phép cộng (*Decimal-adjust Accumulator for Addition*)

- *Mô tả:* DA A hiệu chỉnh giá trị 8-bit trong thanh ghi A (*giá trị này là kết quả phép cộng hai toán hạng có dạng BCD - gói trước đó*) để tạo ra hai digit 4 bit. **Phép cộng được thực hiện bởi lệnh ADD hoặc ADDC, lệnh DA A không áp dụng cho phép trừ SUBB)**.
  Nếu cờ AC = 1 hoặc nếu 4 bit thấp của thanh ghi A có giá trị > "9" (xxxx1010 – xxxx1111), thì "6"được cộng với nội dung của thanh ghi A để tạo ra số BCD ở 4 bit thấp. Sau khi cộng, cờ CY = 1 nếu có số nhớ từ 4 bit thấp chuyển đến tất cả 4 bit cao.
  *Nếu cờ CY = 1 hoặc nếu 4 bit cao của thanh ghi A có giá trị > "9" (1010xxxx – 1111xxxx), thì "6"được cộng với 4 bit cao để tạo ra số BCD ở 4 bit cao. Sau khi cộng cờ CY = 1 nếu có số nhớ từ 4 bit cao nhưng cờ CY không bị xóa. Vậy thì cờ CY chỉ ra rằng tổng của 2 toán hạng BCD ban đầu lớn hơn 99. **Cờ OV không bị ảnh hưởng**.*
  Tất cả sự kiện trên chỉ xảy ra trong một chu kỳ máy. Lệnh này thực hiện phép biến đổi thập phân bằng cách cộng 00H, 06H, 60H hay 66H với nội dung của thanh ghi A tùy thuộc vào nội dung ban đầu của thanh ghi A và các điều kiện của từ trạng thái chương trình PSW.

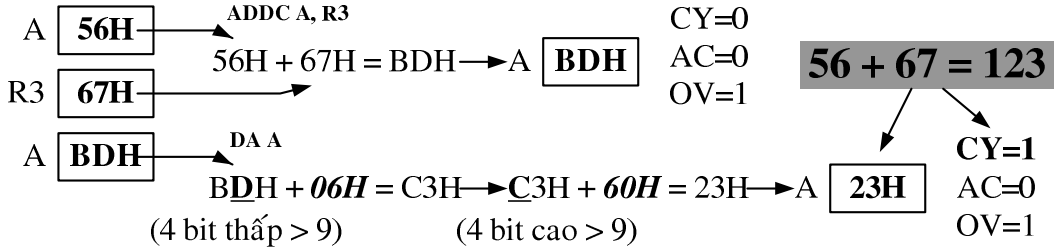
- *Lưu ý:* DA A không thể đơn giản biến đổi số hex trong thanh ghi A thành số dạng BCD, DA A cũng không áp dụng cho phép trừ thập phân.

Số byte	1
Số chu kỳ	1
Mã đối tượng	11010100

Hoạt động Giả sử nội dung của thanh ghi A là BCD
IF [[(A3 – A0) > 9] OR [(AC) = 1]]
THEN (A3 – A0) ← (A3 – A0) + 6
AND
IF [[(A7 – A4) > 9] OR [(C) = 1]]
THEN (A7 – A4) ← (A7 – A4) + 6

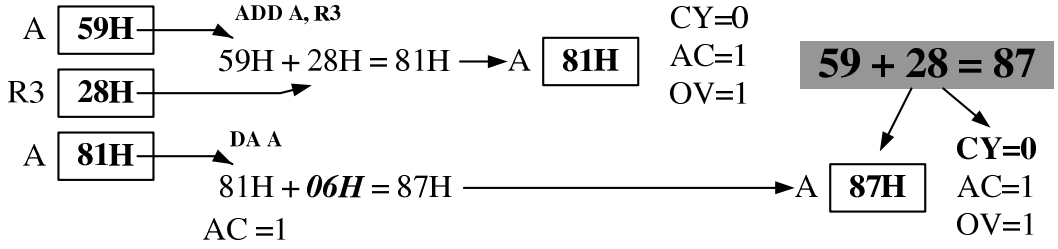
- *Ví dụ 1:* Cho biết trước (A)=56H → biểu diễn BCD của số 56
(R3)=67H → biểu diễn BCD của số 67

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A
thì: cờ CY=1 và (A)=23 → biểu diễn BCD của số 123 (56+67)



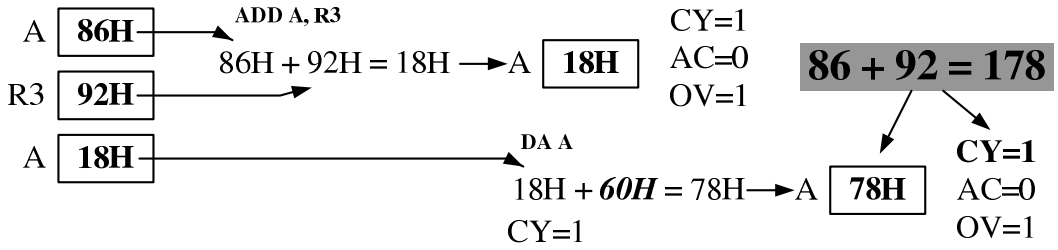
- *Ví dụ 2:* Cho biết trước (A)=59H → biểu diễn BCD của số 59
(R3)=28H → biểu diễn BCD của số 28

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A
thì: cờ CY=0 và (A)=87 → biểu diễn BCD của số 87 (59+28)



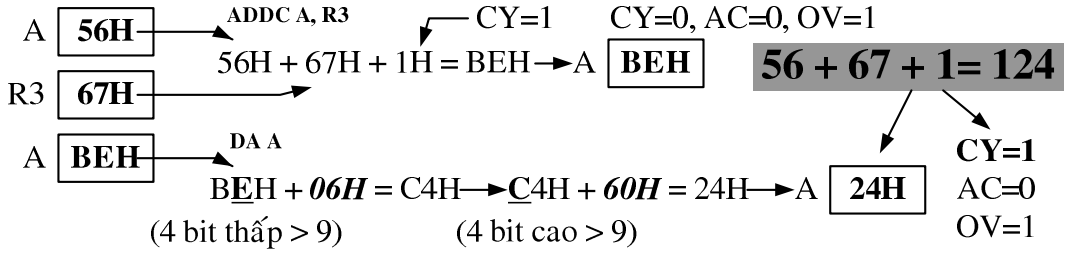
- *Ví dụ 3:* Cho biết trước (A)=86H → biểu diễn BCD của số 86
(R3)=92H → biểu diễn BCD của số 92

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A
thì: cờ CY=1 và (A)=78 → biểu diễn BCD của số 178 (86+92)



- *Ví dụ 4:* Cho biết trước (A)=56H → biểu diễn BCD của số 56
  (R3)=67H → biểu diễn BCD của số 67
  cờ CY=1

Sau khi thực thi chuỗi lệnh: **ADDC A, R3**
DA A
thì: cờ CY=1 và (A)=24 → biểu diễn BCD của số 124 (56+67+1)



- *Lưu ý:* Các giá trị BCD có thể được tăng thêm 1 đơn vị hoặc giãm đi 1 đơn vị bằng cách ***cộng với 01H (khi tăng)*** hoặc ***cộng với 99H (khi giãm)***.

  - *Ví dụ 1:* Giả sử cho (A)=39H → biểu diễn BCD của số 39.

Sau khi thực thi chuỗi lệnh: **ADD A, #01H**
DA A
thì: cờ CY=0 và (A)=40H → biểu diễn BCD của số 40.

  - *Ví dụ 2:* Giả sử cho (A)=30H → biểu diễn BCD của số 30.

Sau khi thực thi chuỗi lệnh: **ADD A, #99H**
DA A
thì: cờ CY=1 và (A)=29H → biểu diễn BCD của số 29.

2. Nhóm lệnh logic:

Bảng trạng thái của các phép toán logic
AND – OR – XOR – CPL

A	B		A AND B	A OR B	A XOR B	CPL A
0	0		0	0	0	1
0	1		0	1	1	1
1	0		0	1	1	0
1	1		1	1	0	0

2.1. ANL <dest-byte>, <src-byte>

- *Chức năng:* AND hai toán hạng (*Logical-AND*).
- *Mô tả:* ANL thực hiện phép toán AND từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (***dest-byte***). *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.

- *Các dạng lệnh:*

ANL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01011rrr
Hoạt động	(A) ← (A) AND (Rn)

ANL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01010101 aaaaaaaa
Hoạt động	(A) ← (A) AND (direct)

ANL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0101011i
Hoạt động	(A) ← (A) AND ((Ri))

ANL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01010100 dddddddd
Hoạt động	(A) ← (A) AND #data

ANL direct, A

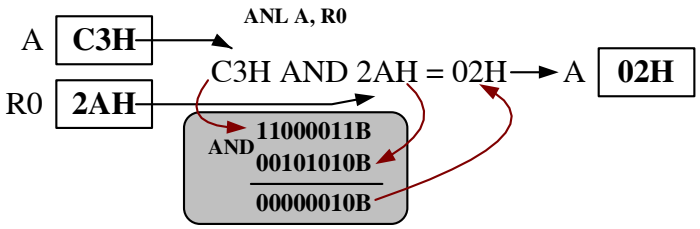
Số byte	2
Số chu kỳ	1
Mã đối tượng	01010010 aaaaaaaa
Hoạt động	(direct) ← (direct) AND (A)

ANL direct, #data

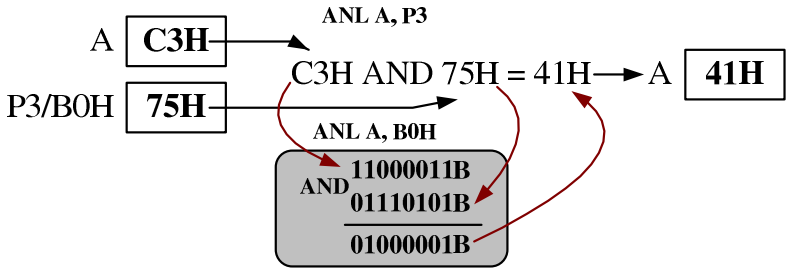
Số byte	3
Số chu kỳ	2
Mã đối tượng	01010011 aaaaaaaa dddddddd
Hoạt động	(direct) ← (direct) AND #data

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=2AH, (P3)=(B0H)=75H, (2AH)=55H.

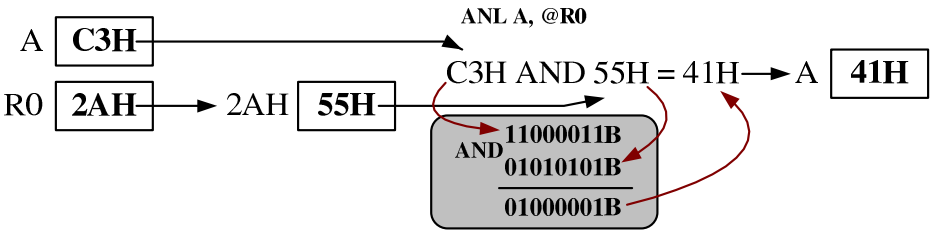
Sau khi thực thi lệnh **ANL A, R0** thì: (A)=02H



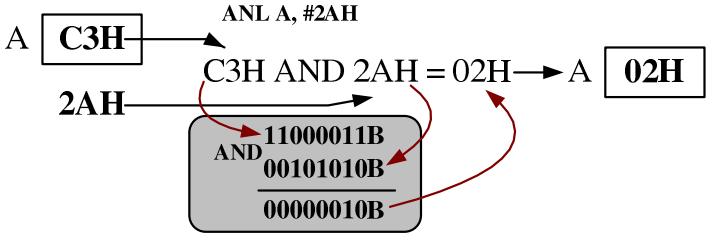
Sau khi thực thi lệnh **ANL A, B0H** hay **ANL A, P3** thì: (A)=41H



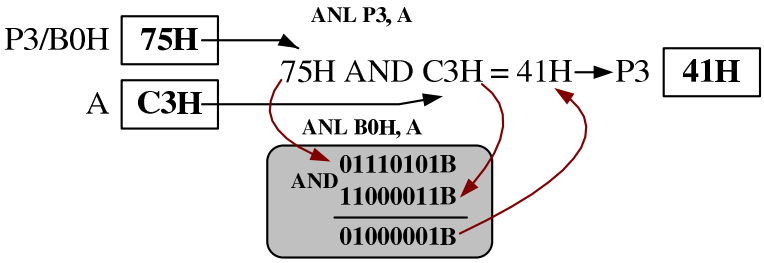
Sau khi thực thi lệnh **ANL A, @R0** thì: (A)=41H



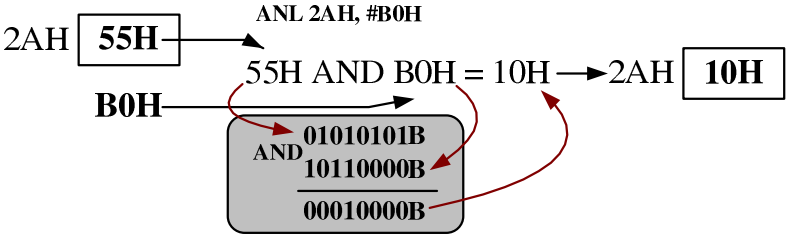
Sau khi thực thi lệnh **ANL A, #2AH** thì: (A)=02H



Sau khi thực thi lệnh **ANL B0H, A** hay **ANL P3, A** thì: (P3)=41H



Sau khi thực thi lệnh **ANL 2AH, #B0H** thì: (2AH)=10H



2.2. ORL <dest-byte>, <src-byte>

- *Chức năng:* OR logic hai toán hạng (*Logical-OR*).
- *Mô tả:* ORL thực hiện phép toán OR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (***dest-byte***). *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.
- *Các dạng lệnh:*

ORL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01001rrr
Hoạt động	(A) ← (A) OR (Rn)

ORL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01000101 aaaaaaaa
Hoạt động	(A) ← (A) OR (direct)

ORL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0100011i
Hoạt động	(A) ← (A) OR ((Ri))

ORL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01000100 dddddddd
Hoạt động	(A) ← (A) OR #data

ORL direct, A

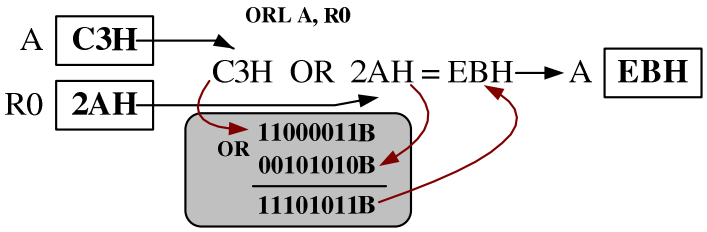
Số byte	2
Số chu kỳ	1
Mã đối tượng	01000010 aaaaaaaa
Hoạt động	(direct) ← (direct) OR (A)

ORL direct, #data

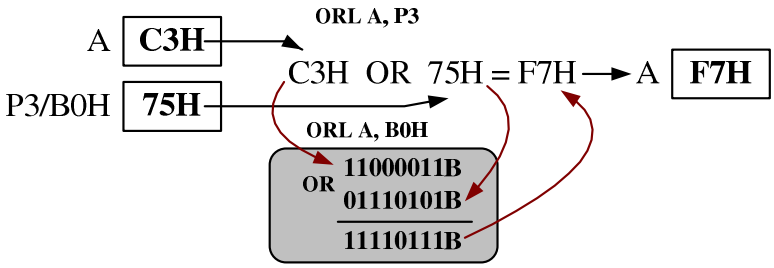
Số byte	3
Số chu kỳ	2
Mã đối tượng	01000011 aaaaaaaa dddddddd
Hoạt động	(direct) ← (direct) OR #data

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=2AH, (P3)=(B0)=75H, (2AH)=55H.

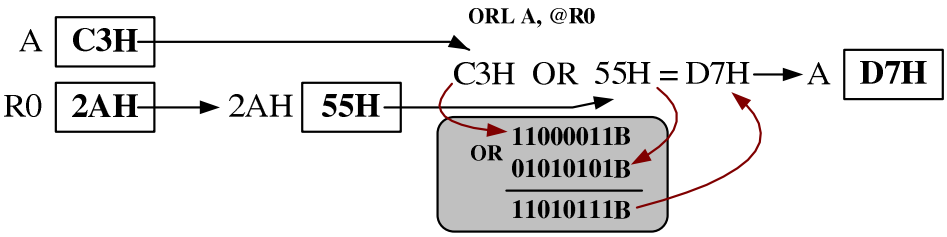
Sau khi thực thi lệnh **ORL A, R0** thì: (A)=EBH



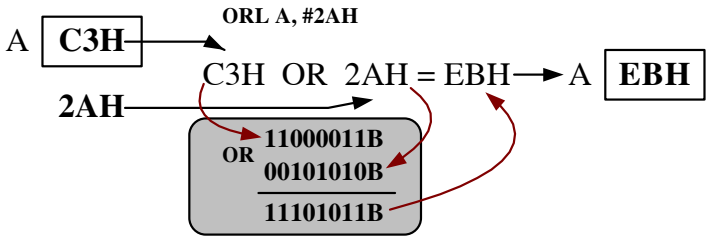
Sau khi thực thi lệnh **ORL A, B0H** hay **ORL A, P3** thì: (A)=F7H



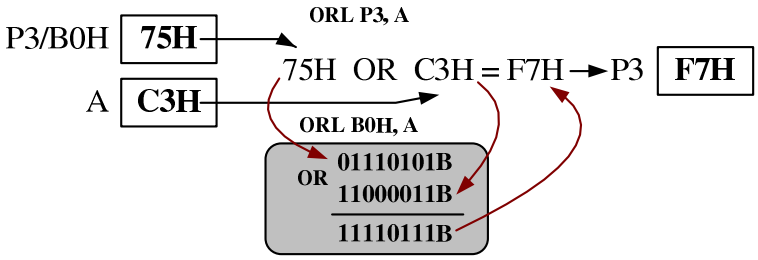
Sau khi thực thi lệnh **ORL A, @R0** thì: (A)=D7H



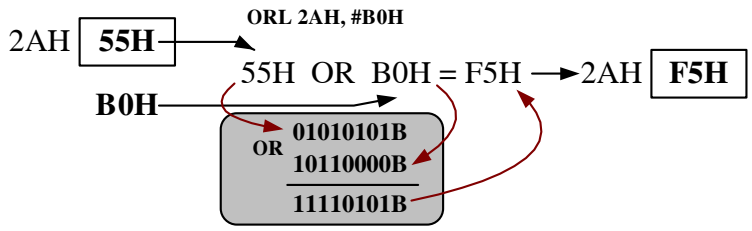
Sau khi thực thi lệnh **ORL A, #2AH** thì: (A)=EBH



Sau khi thực thi lệnh **ORL B0H, A** hay **ORL P3, A** thì: (P3)=F7H



Sau khi thực thi lệnh **ORL 2AH, #B0H** thì: (2AH)=F5H



2.3. XRL <dest-byte>, <src-byte>

- *Chức năng:* XOR logic hai toán hạng (*Logical Exclusive-OR*).
- *Mô tả:* XRL thực hiện phép toán XOR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (***dest-byte***). *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.
- *Các dạng lệnh:*

XRL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01101rrr
Hoạt động	(A) ← (A) ⊕ (Rn)

XRL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01100101 aaaaaaaa
Hoạt động	(A) ← (A) ⊕ (direct)

XRL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0110011i
Hoạt động	(A) ← (A) ⊕ ((Ri))

XRL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01100100 dddddddd
Hoạt động	(A) ← (A) ⊕ #data

XRL direct, A

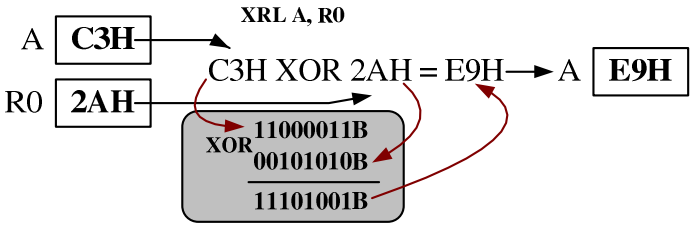
Số byte	2
Số chu kỳ	1
Mã đối tượng	01100010 aaaaaaaa
Hoạt động	(direct) ← (direct) ⊕ (A)

XRL direct, #data

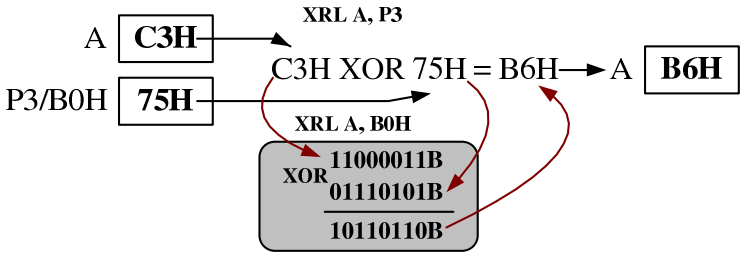
Số byte	3
Số chu kỳ	2
Mã đối tượng	01100011 aaaaaaaa dddddddd
Hoạt động	(direct) ← (direct) ⊕ #data

- *Ví dụ:* Cho biết trước (A)=C3H, (R0)=2AH, (P3)=(B0)=75H, (2AH)=55H.

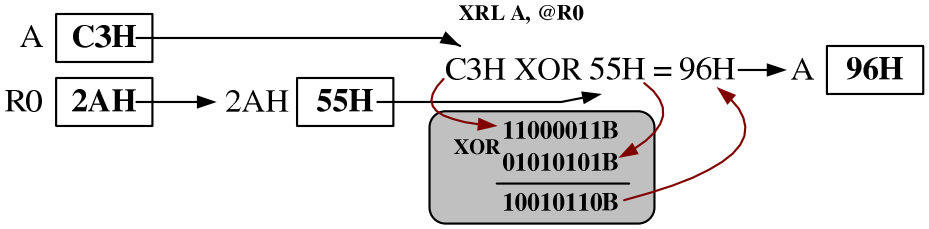
Sau khi thực thi lệnh **XRL A, R0** thì: (A)=E9H



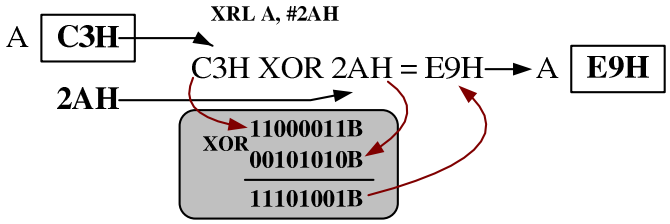
Sau khi thực thi lệnh **XRL A, B0H** hay **XRL A, P3** thì: (A)=B6H



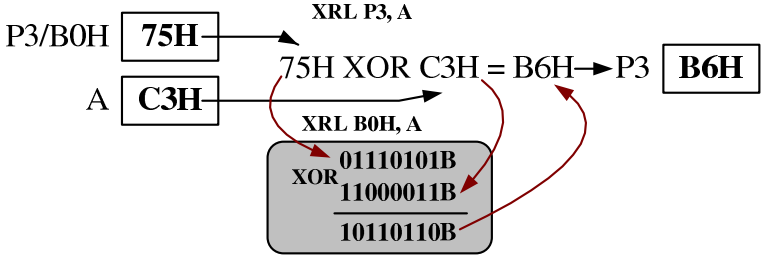
Sau khi thực thi lệnh **XRL A, @R0** thì: (A)=96H



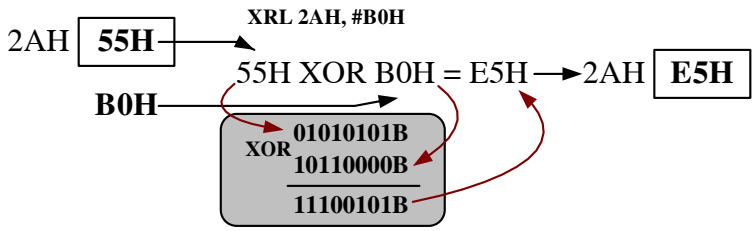
Sau khi thực thi lệnh **XRL A, #2AH** thì: (A)=E9H



Sau khi thực thi lệnh **XRL B0H, A** hay **XRL P3, A** thì: (P3)=B6H



Sau khi thực thi lệnh **XRL 2AH, #B0H** thì: (2AH)=E5H



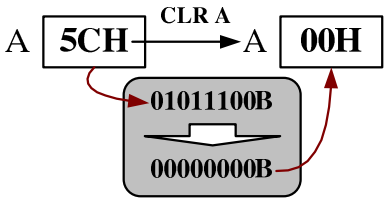
2.4. CLR A

- *Chức năng:* Xóa thanh ghi A (*Clear Acc*).
- *Mô tả:* Thanh ghi A bị xóa (*tất cả các bit đều bằng 0*). *Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

Số byte	1
Số chu kỳ	1
Mã đối tượng	11100100
Hoạt động	(A) ← 0

- *Ví dụ:* Cho biết trước (A)=5CH.

Sau khi thực thi lệnh **CLR A** thì: (A)=00H



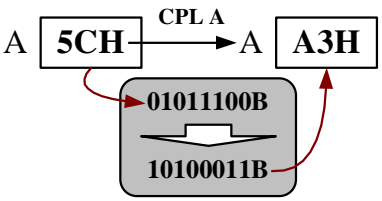
2.5. CPL A

- *Chức năng:* Lấy bù nội dung thanh ghi A (*Complement Acc*).
- *Mô tả:* Mỗi một bit của thanh ghi A được lấy bù logic (*bù 1: các bit 1 được thì đổi thành bit 0 và các bit 0 được thì đổi thành bit 1*). *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	11110100
Hoạt động	(A) ← NOT(A)

- *Ví dụ:* Cho biết trước (A)=5CH.

Sau khi thực thi lệnh **CPL A** thì: (A)=A3H



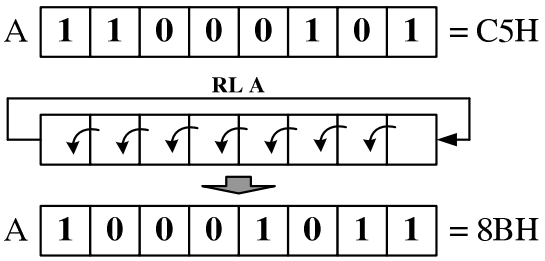
2.6. RL A

- *Chức năng:* Quay trái thanh ghi A (*Rotate Acc Left*).
- *Mô tả:* 8 bit trong thanh ghi A được quay trái 1 bit. Bit 7 được quay đến vị trí của bit 0. *Các cờ không bị ảnh hưởng*.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00100011
Hoạt động	(An+1) ← (An), n = 0 – 6 (A0) ← A7

- *Ví dụ:* Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **RL A** thì: (A)=8BH



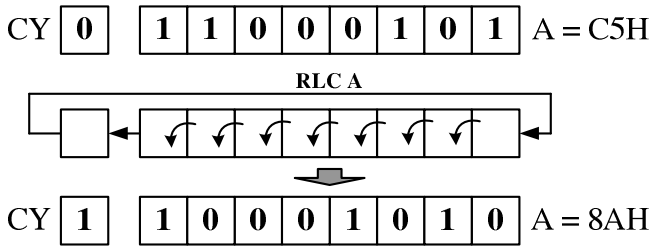
2.7. RLC A

- *Chức năng:* Quay trái thanh ghi A cùng với cờ nhớ.
- *Mô tả:* 8 bit trong thanh ghi A và cờ nhớ cùng được quay trái 1 bit. Bit 7 được di chuyển đến cờ CY và trạng thái ban đầu của cờ CY được đưa đến vị trí của bit 0. *Các cờ khác không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	00110011
Hoạt động	(An+1) ← (An), n = 0 – 6 (A0) ← (C), (C) ← A7

- *Ví dụ:* Cho biết trước (A)=C5H và cờ CY=0.

Sau khi thực thi lệnh **RLC A** thì: (A)=8AH và cờ CY=1



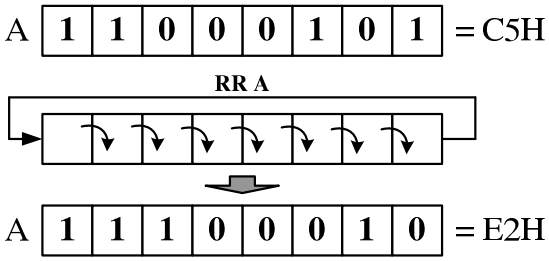
2.8. RR A

- *Chức năng:* Quay phải thanh ghi A (*Rotate Acc Right*).
- *Mô tả:* 8 bit trong thanh ghi A được quay phải 1 bit. Bit 0 được quay đến vị trí của bit 7. *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	00000011
Hoạt động	(An) ← (An+1), n = 0 – 6 (A7) ← A0

- *Ví dụ:* Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **RR A** thì: (A)=E2H



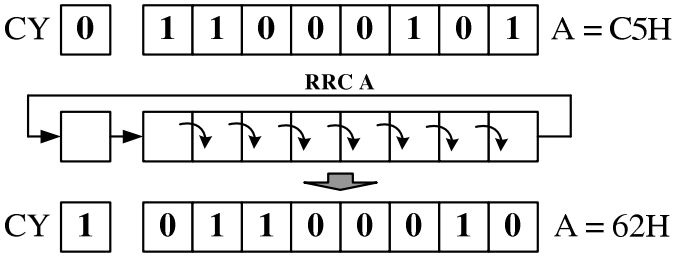
2.9. RRC A

- *Chức năng:* Quay phải thanh ghi A cùng với cờ nhớ.
- *Mô tả:* 8 bit trong thanh ghi A và cờ nhớ cùng được quay phải 1 bit. Bit 0 được di chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa đến vị trí của bit 7. *Các cờ khác không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	00010011
Hoạt động	(An) ← (An+1), n = 0 – 6 (A7) ← (C) (B) ← A0

- *Ví dụ:* Cho biết trước (A)=C5H và cờ CY=0.

Sau khi thực thi lệnh **RRC A** thì: (A)=62H và cờ CY=1



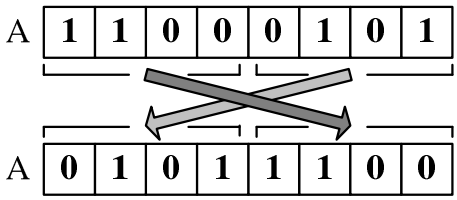
2.10. SWAP A

- *Chức năng:* Tráo đổi nội dung hai nửa thấp và cao của thanh ghi A (*Swap nibble*).
- *Mô tả:* SWAP A tráo đổi nội dung hai nửa thấp và cao của thanh ghi A (*các bit từ 3 đến 0 ⇔ các bit từ 7 đến 4*). Thao tác này còn có thể được hiểu như là quay thanh ghi A đi 4 bit. *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	11000100
Hoạt động	(A3 – A0) ↔ (A7 – A4)

- *Ví dụ:* Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **SWAP A** thì: (A)=5CH



- *Lưu ý:* Lệnh này có thể được dùng để chuyển đổi giá trị nhị phân trong thanh ghi A (*giá trị này nhỏ hơn 100*) thành số BCD như sau:

```
MOV     B, #10      ;Chia giá trị cho 10 để tách ra
DIV     AB          ;(A)=digit chục – (B)=digit đơn vị.
SWAP    A           ;Đưa digit chục lên nữa cao của ACC.
ADD     A, B        ;Thêm digit đơn vị vào nữa thấp.
```

3. Nhóm lệnh di chuyển dữ liệu:

3.1. MOV <dest-byte>, <src-byte>

- *Chức năng:* Di chuyển nội dung của toán hạng nguồn (***src-byte***) đến toán hạng đích (***dest-byte***).
- *Mô tả:* Nội dung của byte được chỉ ra bởi toán hạng thứ hai được sao chép vào vị trí được xác định bởi toán hạng thứ nhất. Byte nguồn không bị ảnh hưởng. *Các thanh ghi khác và các cờ không bị ảnh hưởng*.
- *Các dạng lệnh:*

MOV A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	11101rrr
Hoạt động	(A) ← (Rn)

MOV A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	11100101 aaaaaaaa
Hoạt động	(A) ← (direct)

Lưu ý: *MOV A, ACC là lệnh không hợp lệ.*

MOV A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	1110011i
Hoạt động	(A) ← ((Ri))

MOV A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01110100 dddddddd
Hoạt động	(A) ← #data

MOV Rn, A

Số byte	1
Số chu kỳ	1
Mã đối tượng	11111rrr
Hoạt động	(Rn) ← (A)

MOV Rn, direct

Số byte	2
Số chu kỳ	2
Mã đối tượng	10101rrr aaaaaaaa
Hoạt động	(Rn) ← (direct)

MOV Rn, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01111rrr dddddddd
Hoạt động	(Rn) ← #data

MOV direct, A

Số byte	2
Số chu kỳ	1
Mã đối tượng	11110101 aaaaaaaa
Hoạt động	(direct) ← (A)

MOV direct, Rn

Số byte	2
Số chu kỳ	2
Mã đối tượng	10001rrr aaaaaaaa
Hoạt động	(direct) ← (Rn)

MOV direct, direct

Số byte	3
Số chu kỳ	2
Mã đối tượng	10000101 aaaaaaaa aaaaaaaa
Hoạt động	(direct) ← (direct)

***Lưu ý**: byte 2 chứa địa chỉ nguồn, byte 3 chứa địa chỉ đích.*

MOV direct, @Ri

Số byte	2
Số chu kỳ	2
Mã đối tượng	1000011i aaaaaaaa
Hoạt động	(direct) ← ((Ri))

MOV direct, #data

Số byte	3
Số chu kỳ	2
Mã đối tượng	01110101 aaaaaaaa dddddddd
Hoạt động	(direct) ← #data

MOV @Ri, A

Số byte	1
Số chu kỳ	1
Mã đối tượng	1111011i
Hoạt động	((Ri)) ← (A)

MOV @Ri, direct

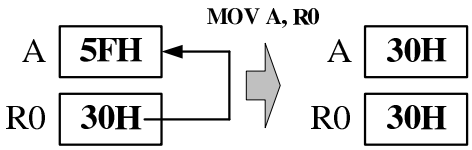
Số byte	2
Số chu kỳ	2
Mã đối tượng	1010011i aaaaaaaa
Hoạt động	((Ri)) ← (direct)

MOV @Ri, #data

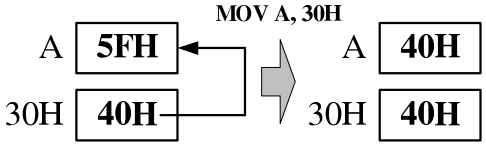
Số byte	2
Số chu kỳ	1
Mã đối tượng	0111011i dddddddd
Hoạt động	((Ri)) ← #data

- *Ví dụ:* Cho biết trước (A)=5FH, (R0)=30H, (30H)=40H, (P1)=CAH.

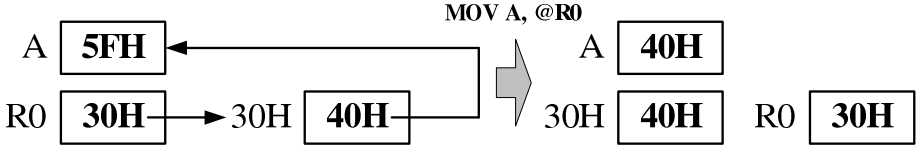
Sau khi thực thi lệnh **MOV A, R0** thì: (A)=30H, (R0)=30H



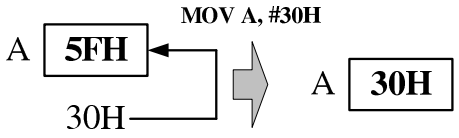
Sau khi thực thi lệnh **MOV A, 30H** thì: (A)=40H, (30H)=40H



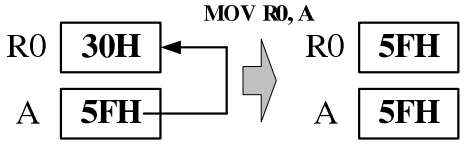
Sau khi thực thi lệnh **MOV A, @R0** thì: (A)=40H, (R0)=30H, (30H)=40H



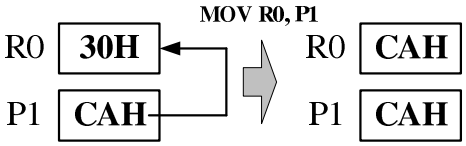
Sau khi thực thi lệnh **MOV A, #30H** thì: (A)=30H



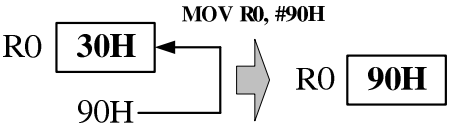
Sau khi thực thi lệnh **MOV R0, A** thì: (A)=5FH, (R0)=5FH



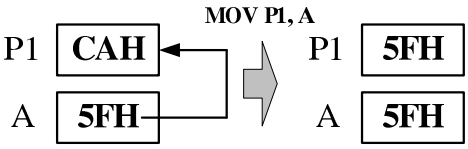
Sau khi thực thi lệnh **MOV R0, P1** thì: (R0)=CAH, (P1)=CAH



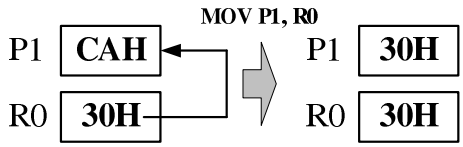
Sau khi thực thi lệnh **MOV R0, #90H** thì: (R0)=90H



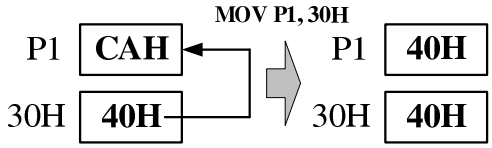
Sau khi thực thi lệnh **MOV P1, A** thì: (A)=5FH, (P1)=5FH



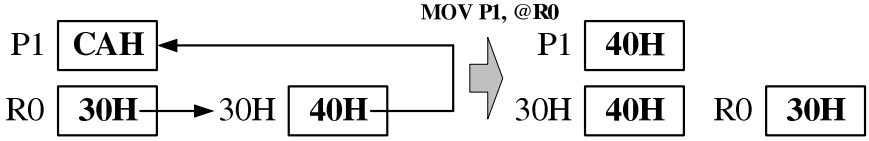
Sau khi thực thi lệnh **MOV P1, R0** thì: (R0)=30H, (P1)=30H



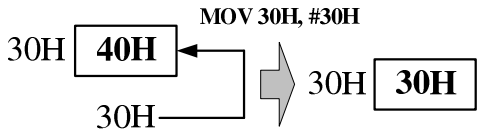
Sau khi thực thi lệnh **MOV P1, 30H** thì: (30H)=40H, (P1)=40H



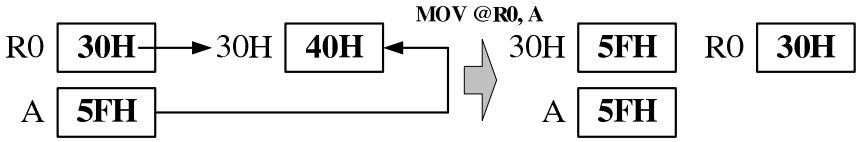
Sau khi thực thi lệnh **MOV P1, @R0** thì: (R0)=30H, (30H)=40H, (P1)=40H



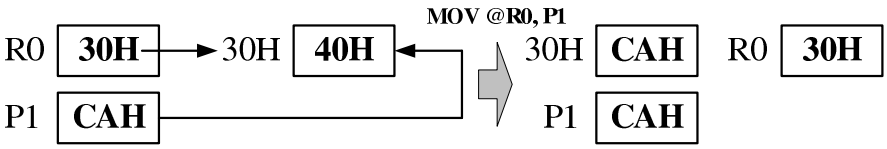
Sau khi thực thi lệnh **MOV 30H, #30H** thì: (30H)=30H



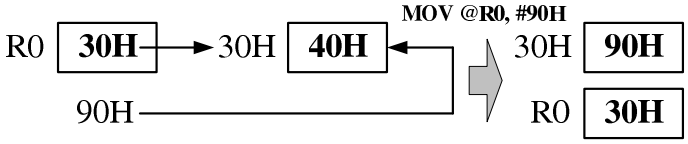
Sau khi thực thi lệnh **MOV @R0, A** thì: (A)=5FH, (30H)=5FH, (R0)=30H



Sau khi thực thi lệnh **MOV @R0, P1**: (30H)=CAH, (P1)=CAH, (R0)=30H



Sau khi thực thi lệnh **MOV @R0, #90H** thì: (30H)=90H, (R0)=30H



3.2. MOVC A, @A+ <base-reg>

- *Chức năng:* Di chuyển byte mã hoặc byte hằng số.
- *Mô tả:* MOVC nạp cho thanh ghi A byte mã hoặc byte hằng số từ bộ nhớ chương trình. Địa chỉ của byte được tìm nạp là tổng của giá trị 8 bit không dấu ban đầu chứa trong thanh ghi A với nội dung của thanh ghi nền 16 bit (*thanh ghi nền có thể là con trỏ dữ liệu hoặc PC*). Trong trường hợp sau, thanh ghi PC được tăng để chỉ đến địa chỉ của lệnh tiếp theo trước khi được cộng với nội dung của thanh ghi A, các thanh ghi nền không bị thay đổi. Phép cộng bit thứ 16 do số nhớ từ 8 bit thấp có thể truyền qua các bit cao. *Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

MOVC A, @A+DPTR

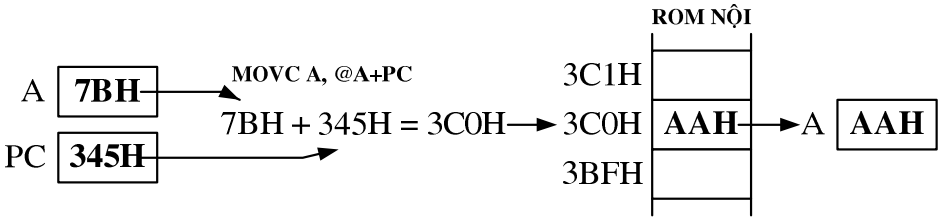
Số byte	1
Số chu kỳ	2
Mã đối tượng	10010011
Hoạt động	(A) ← ((A)+(DPTR))

MOVC A, @A+PC

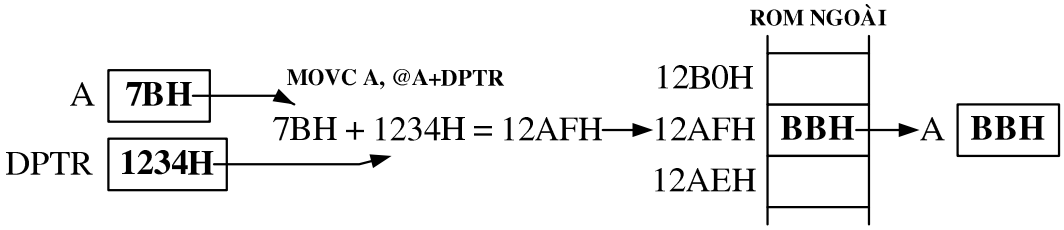
Số byte	1
Số chu kỳ	2
Mã đối tượng	10000011
Hoạt động	(A) ← ((A)+(PC))

- *Ví dụ 1:* Cho biết trước (A)=7BH, (PC)=345H, (DPTR)=1234H. Ô nhớ ROM nội (3C0H)=AAH. Ô nhớ ROM ngoài (12AFH)=BBH.

Sau khi thực thi lệnh **MOVC A, @A+PC** thì: (A)=AAH



Sau khi thực thi lệnh **MOVC A, @A+DPTR** thì: (A)=BBH



- *Ví dụ 2:* Cho chuỗi lệnh sau:

```
        MOVC    A, @A+PC            ;(@A+PC)=([A]+[PC])
        SJMP    $                   ;Độ dài lệnh là 2 byte.
DULIEU:
        DB      66H,77H,88H
        DB      99H,0AAH
        DB      0BBH
```

Sau khi thực thi chuỗi lệnh thì:

- ⇨ (A) = 66H nếu trước khi thực thi lệnh ta có (A) = 02H.
- ⇨ (A) = 77H nếu trước khi thực thi lệnh ta có (A) = 03H.
- ⇨ (A) = 88H nếu trước khi thực thi lệnh ta có (A) = 04H.
- ⇨ (A) = 99H nếu trước khi thực thi lệnh ta có (A) = 05H.
- ⇨ (A) = AAH nếu trước khi thực thi lệnh ta có (A) = 06H.
- ⇨ (A) = BBH nếu trước khi thực thi lệnh ta có (A) = 07H.

- *Ví dụ 3:* Cho chuỗi lệnh sau:

```
        MOV     DPTR, #CODEDISP
        MOVC    A, @A+DPTR          ;(@A+DPTR)=([A]+[DPTR])
        SJMP    $                   ;Độ dài lệnh là 2 byte.
CODEDISP:
        DB      48H,5AH,6BH,0A9H,0F5H,90H
```

Sau khi thực thi chuỗi lệnh thì:

- ⇨ (A) = 48H nếu trước khi thực thi lệnh ta có (A) = 00H.
- ⇨ (A) = 5AH nếu trước khi thực thi lệnh ta có (A) = 01H.
- ⇨ (A) = 6BH nếu trước khi thực thi lệnh ta có (A) = 02H.
- ⇨ (A) = A9H nếu trước khi thực thi lệnh ta có (A) = 03H.
- ⇨ (A) = F5H nếu trước khi thực thi lệnh ta có (A) = 04H.
- ⇨ (A) = 90H nếu trước khi thực thi lệnh ta có (A) = 05H.

3.3. MOVX <dest-byte>, <src-byte>

- *Chức năng:* Di chuyển ở bộ nhớ ngoài (*Move External*).
- *Mô tả:* MOVX chuyển dữ liệu giữa thanh ghi A với nội dung của một byte trong bộ nhớ dữ liệu ngoài (*ta dùng ký hiệu X nối tiếp với MOV*). Các lệnh này được chia làm 2 loại, hai loại này khác nhau ở chổ chúng cung cấp **địa chỉ gián tiếp 8 bit hay 16 bit cho bộ nhớ dữ liệu ngoài** có dung lượng **256 byte hay 64 KB**.

  Với ***loại thứ nhất***, sử dụng thanh ghi R0 hoặc R1 để lưu giữ địa chỉ của dữ liệu cần truy xuất thuộc RAM ngoài. Loại này dùng trong trường hợp bộ nhớ RAM có dung lượng nhỏ (*tối đa là 256 byte*). **Port 1 & Port 2 là port xuất/nhập dữ liệu.**

  Với ***loại thứ hai***, sử dụng thanh ghi DPTR để lưu giữ địa chỉ của dữ liệu cần truy xuất thuộc RAM ngoài. Loại này dùng trong trường hợp bộ nhớ RAM có dung lượng lớn (*tối đa là 64 KB*). **Port 1 là port xuất/nhập dữ liệu.**

  Trong nhiều tình huống ta có thể trộn hai loại trên của lệnh MOVX. Một dãy RAM lớn với các đường địa chỉ cao được điều khiển bởi P2 có thể được định địa chỉ thông qua con trỏ dữ liệu hoặc với mã để xuất ra các bit địa chỉ cao đến P2 được tiếp theo bởi một lệnh MOVX sử dụng R0 hoặc R1.
- *Các dạng lệnh:*

MOVX A, @Ri

Số byte	1
Số chu kỳ	2
Mã đối tượng	11100011i
Hoạt động	(A) ← ((Ri))

MOVX A, @DPTR

Số byte	1
Số chu kỳ	2
Mã đối tượng	11100000
Hoạt động	(A) ← ((DPTR))

MOVX @Ri, A

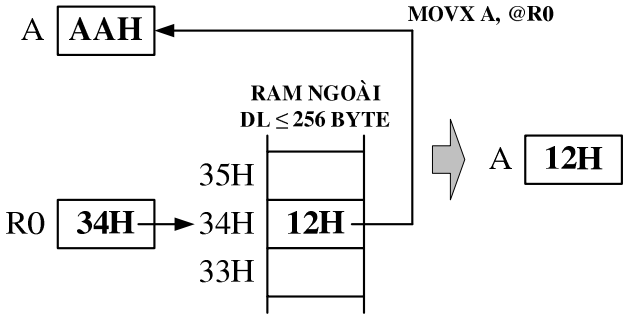
Số byte	1
Số chu kỳ	2
Mã đối tượng	11110011
Hoạt động	((Ri)) ← (A)

MOVX @DPTR, A

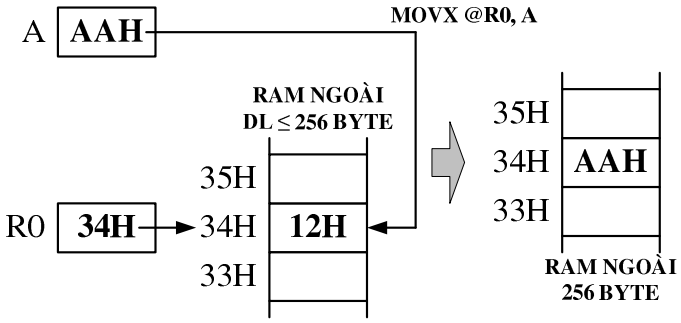
Số byte	1
Số chu kỳ	2
Mã đối tượng	11110000
Hoạt động	((DPTR)) ← (A)

- *Ví dụ:* Cho biết trước (A)=AAH, (DPTR)=1234H, (R0)=34H. Ô nhớ RAM ngoài 256 byte: (34H)=12H & 64 KB: (1234H)=7FH

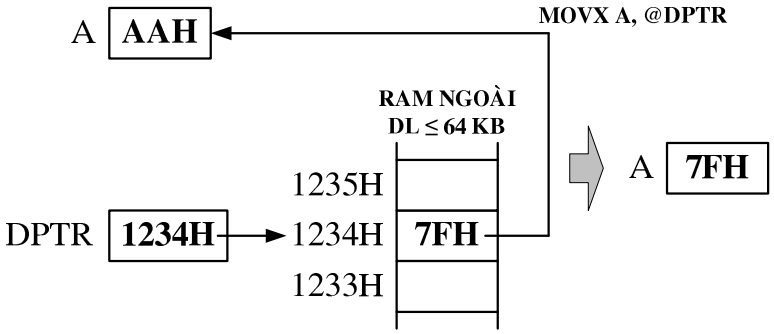
Sau khi thực thi lệnh **MOVX A, @R0** thì: (A)=12H



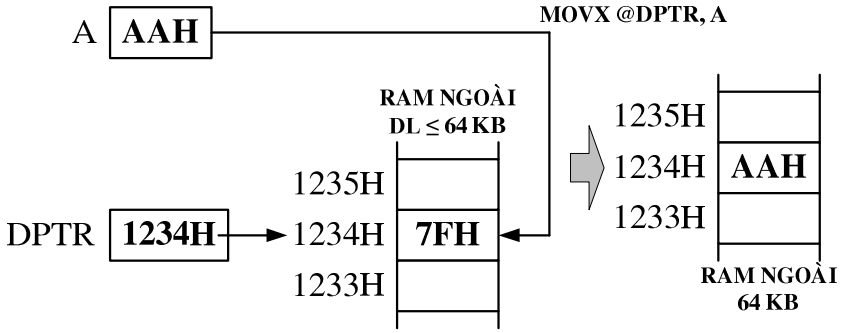
Sau khi thực thi lệnh **MOVX @R0, A** thì: ô nhớ RAM ngoài (34H)=AAH



Sau khi thực thi lệnh **MOVX A, @DPTR** thì: (A)=7FH



Sau khi thực thi lệnh **MOVX @DPTR, A** thì: (1234H)=AAH



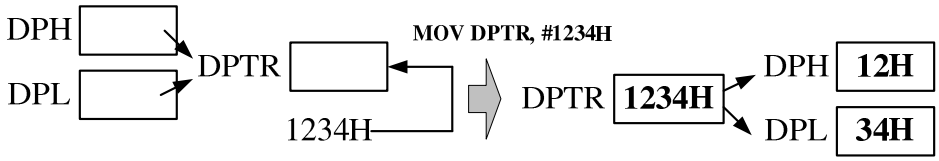
3.4 MOV DPTR, #data16

- *Chức năng:* Nạp hằng số 16-bit cho con trỏ dữ liệu DPTR.
- *Mô tả:* Con trỏ dữ liệu được nạp bởi hằng số 16-bit chỉ ra trong lệnh. Hằng số 16-bit được đặt ở byte 2 và byte 3 của lệnh. Byte 2 là byte cao được nạp cho DPH còn byte 3 là byte thấp được nạp cho DPL. *Các cờ không bị ảnh hưởng.*

Số byte	3
Số chu kỳ	2
Mã đối tượng	10010000 dddddddd dddddddd
Hoạt động	(DPTR) ← #data16

- *Ví dụ:*

Sau khi thực thi lệnh **MOV DPTR, #1234H** thì:



3.5. PUSH direct

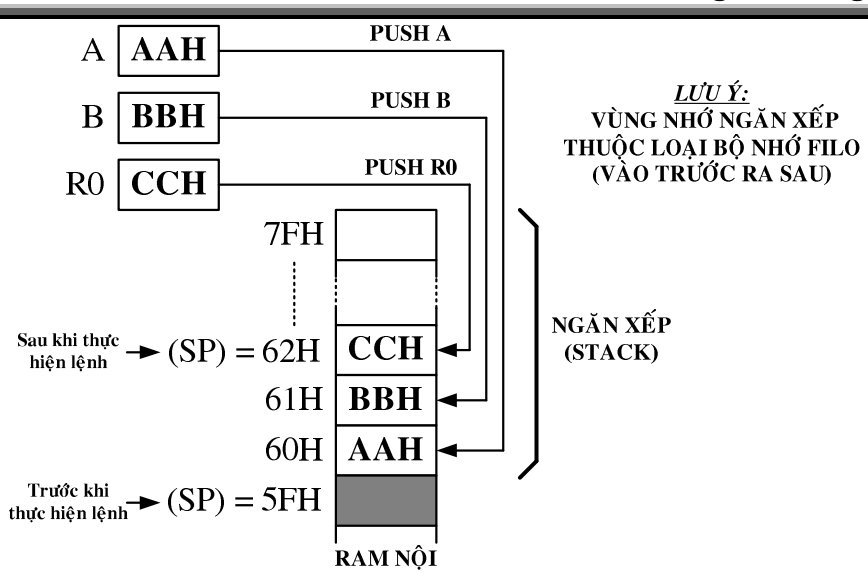
- *Chức năng:* Cất vào ngăn xếp (*stack).*
- *Mô tả:* Con trỏ *stack* được tăng bởi 1. Nội dung của toán hạng được chỉ ra trong lệnh sau đó được sao chép vào RAM nội tại địa chỉ được trỏ đến bởi con trỏ stack. *Các cờ không bị ảnh hưởng.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	11000000 aaaaaaaa
Hoạt động	(SP) ← (SP) + 1 ((SP)) ← (direct)

- *Ví dụ:* Cất tạm thời nội dung của thanh ghi A, B và R0 vào ngăn xếp. Cho biết trước (A)=AAH, (B)=BBH, (R0)=CCH, (SP)=5FH.

Sau khi thực thi chuỗi lệnh:
PUSH A
PUSH B
PUSH 00H

Thì: (SP)=62H, (60H)=AAH, (61H)=BBH, (62H)=CCH



3.6. POP direct

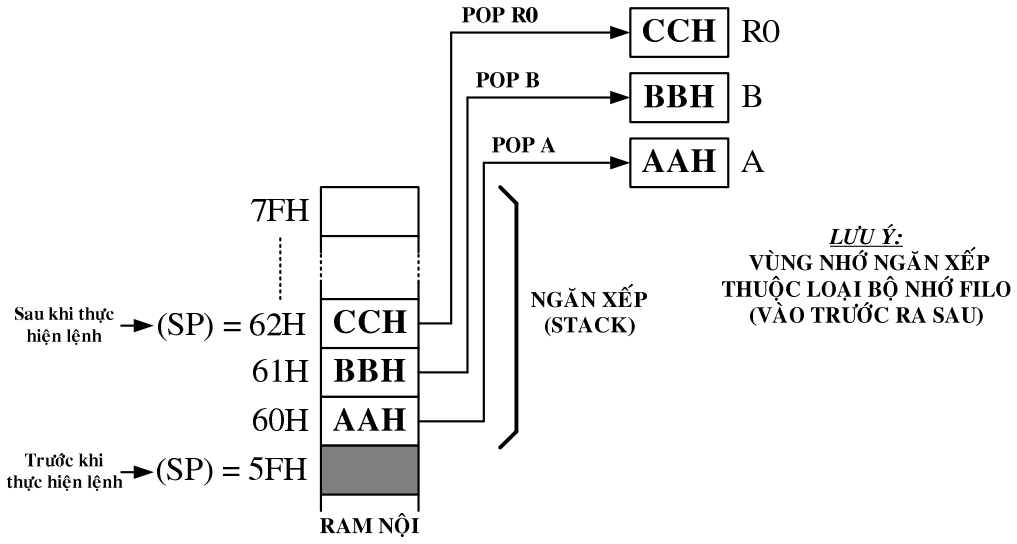
- *Chức năng:* Lấy ra từ ngăn xếp (*stack).*
- *Mô tả:* Nội dung của vùng RAM nội được định địa chỉ bởi con trỏ stack SP được đọc và nội dung con trỏ stack được giảm bởi 1. Giá trị đọc được sau đó được chuyển đến byte được định địa chỉ trực tiếp chỉ ra trong lệnh. *Các cờ không bị ảnh hưởng.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	11010000 aaaaaaaa
Hoạt động	(direct) ← ((SP))
	(SP) ← (SP) – 1

- *Ví dụ:* Lấy lại nội dung của thanh ghi A, B và R0 đã cất vào ngăn xếp lúc đầu (ví dụ trên).

Sau khi thực thi chuỗi lệnh: **POP 00H**
POP B
POP A

Thì: (SP)=5FH, (R0)=CCH, (B)=BBH, (A)=AAH



3.7. XCH A, <byte>

- *Chức năng:* Tráo đổi nội dung của thanh ghi A với nội dung của một byte
- *Mô tả:* XCH nạp cho thanh ghi A nội dung của byte chỉ ra trong lệnh, đồng thời ghi nội dung ban đầu của thanh ghi A cho byte vừa nêu trên. Toán hạng nguồn đồng thời là toán hạng đích và ngược lại.
- *Các dạng lệnh:*

XCH A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	11001rrr
Hoạt động	(A) ↔ (Rn)

XCH A, direct

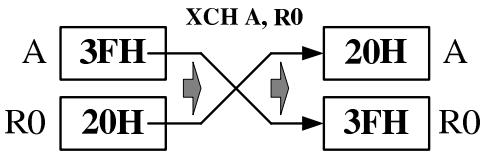
Số byte	2
Số chu kỳ	1
Mã đối tượng	11000101 aaaaaaaa
Hoạt *động*	(A) ↔ (direct)

XCH A, @Ri

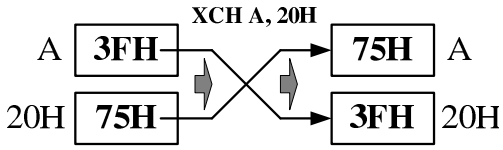
Số byte	1
Số chu kỳ	1
Mã đối tượng	1100011i
Hoạt động	(A) ↔ ((Ri))

- *Ví dụ:* Cho biết trước (A)=3FH, (R0)=20H, (20H)=75H.

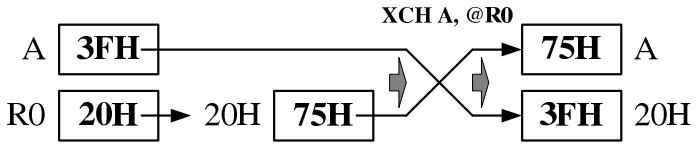
Sau khi thực thi lệnh **XCH A, R0** thì: (A)=20H, (R0)=3FH



Sau khi thực thi lệnh **XCH A, 20H** thì: (A)=75H, (20H)=3FH



Sau khi thực thi lệnh **XCH A, @R0** thì: (A)=75H, (20H)=3FH, (R0)=20H



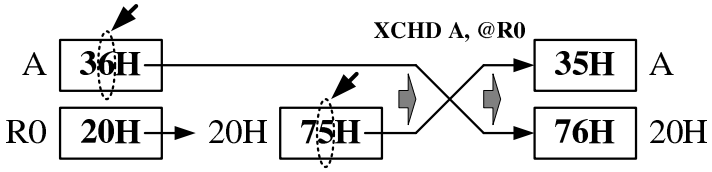
3.8. XCHD A, @Ri

- *Chức năng:* Tráo đổi digit (*Exchange Digit*).
- *Mô tả:* XCHD tráo đổi nội dung nửa thấp của thanh ghi A (*biểu diễn một digit số hex hoặc BCD*) với nội dung nửa thấp của một byte trong RAM nội, byte này được định địa chỉ gián tiếp bởi thanh ghi chỉ ra trong lệnh. *Nửa cao của các thanh ghi vừa nêu trên không bị ảnh hưởng và các cờ cũng không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	110101i
Hoạt động	(A3 – A0) ↔ (Ri3 – Ri0)

- *Ví dụ:* Cho biết trước (A)=36H, (R0)=20H, (20H)=75H.

Sau khi thực thi lệnh **XCHD A, @R0** thì: (A)=35H, (20H)=76H



4. Nhóm lệnh xử lý bit:

4.1. CLR bit

- *Chức năng:* Xóa bit.
- *Mô tả:* Bit được chỉ ra trong lệnh được xóa. *Các cờ không bị ảnh hưởng. CLR có thể thao tác trên cờ nhớ và trên một bit bất kỳ được định địa chỉ bit.*
- *Các dạng lệnh:*

CLR C

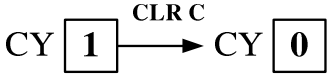
Số byte	1
Số chu kỳ	1
Mã đối tượng	11000011
Hoạt động	(C) ← 0

CLR bit

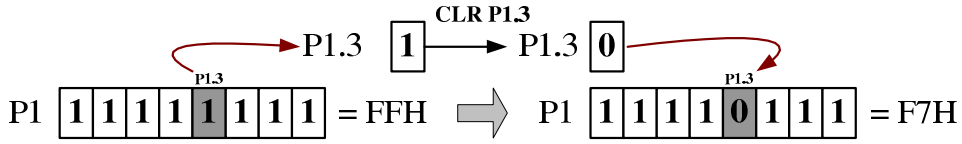
Số byte	*2*	
Số chu kỳ	*1*	
Mã đối tượng	*11000010*	*bbbbbbbb*
Hoạt động	*(bit) ← 0*	

- *Ví dụ:* Cho biết trước cờ CY=1, (P1)=FFH.

Sau khi thực thi lệnh **CLR C** thì: cờ CY=0



Sau khi thực thi lệnh **CLR P1.3** thì: P1.3=0 tức làm (P1)=F7H



4.2. CPL bit

- *Chức năng:* Lấy bù bit (*Complex*).
- *Mô tả:* Bit được chỉ ra trong lệnh được lấy bù. Một bit có giá trị 1 được đổi thành 0 và ngược lại. *Các cờ không bị ảnh hưởng. CPL có thể thao tác trên cờ nhớ và trên một bit bất kỳ được định địa chỉ bit.*
- *Lưu ý:* Khi lệnh này được dùng để làm thay đổi giá trị của một chân xuất (*bất kỳ chân nào của một port*) thì giá trị được dùng làm dữ liệu ban đầu của chân được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ chân nhập.
- *Các dạng lệnh:*

CPL C

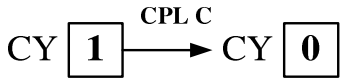
Số byte	1
Số chu kỳ	1
Mã đối tượng	10110011
Hoạt động	(C) ← NOT(C)

CPL bit

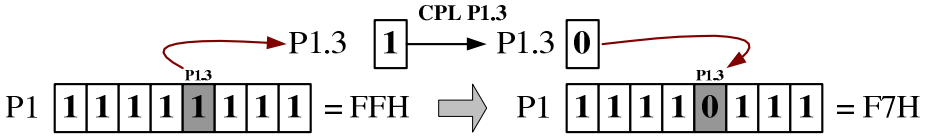
Số byte	*2*
Số chu kỳ	*1*
Mã đối tượng	*10110010 bbbbbbbb*
Hoạt động	*(bit) ← NOT(bit)*

- *Ví dụ:* Cho biết trước cờ CY=1, (P1)=FFH.

Sau khi thực thi lệnh **CPL C** thì: cờ CY=0



Sau khi thực thi lệnh **CPL P1.3** thì: P1.3=0 tức làm (P1)=F7H



4.3. SETB <bit>

- *Chức năng:* *Set* bit bằng 1 (*Set Bit*).
- *Mô tả:* SETB *set* bit được chỉ ra trong lệnh bằng 1. *SETB có thể thao tác trên cờ nhớ hoặc các bit bất kỳ được định địa chỉ bit. Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

SETB C

Số byte	1
Số chu kỳ	1
Mã đối tượng	11010011
Hoạt động	(C) ← 1

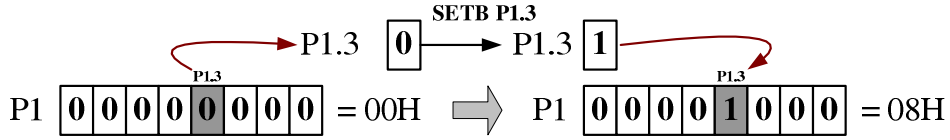
SETB bit

Số byte	*2*	
Số chu kỳ	*1*	
Mã đối tượng	*11010010*	*bbbbbbbb*
Hoạt động	*(bit) ← 1*	

- *Ví dụ:* Cho biết trước cờ CY=0, (P1)=00H.

Sau khi thực thi lệnh **SETB C** thì: cờ CY=1

Sau khi thực thi lệnh **SETB P1.3** thì: P1.3=1 tức làm (P1)=08H



4.4. ANL C, <src-bit>

- *Chức năng:* AND logic hai bit.
- *Mô tả:* Nếu giá trị của bit nguồn là 0 thì lệnh này sẽ xóa CY và ngược lại nếu giá trị của bit nguồn là 1 thì lệnh này giữ nguyên giá trị hiện hành của cờ CY. Dấu gạch chéo (/) đặt trước toán hạng trong chương trình hợp ngữ chỉ ra rằng bit nguồn được lấy bù trước khi AND với CY nhưng ***giá trị của bit nguồn không bị thay đổi*** bởi thao tác lấy bù này. *Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

ANL C, bit

Số byte	2	
Số chu kỳ	2	
Mã đối tượng	10000010	bbbbbbbb
Hoạt động	(C) ← (C) AND (bit)	

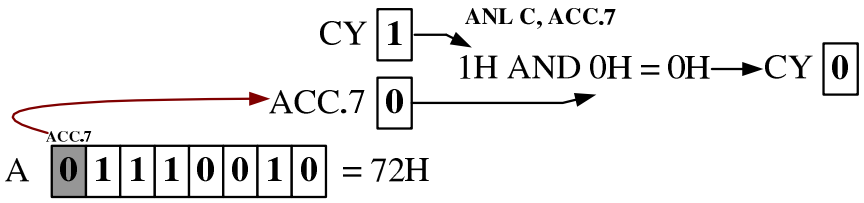
ANL C, /bit

Số byte	*2*	
Số chu kỳ	*2*	
Mã đối tượng	*10110000*	*bbbbbbbb*

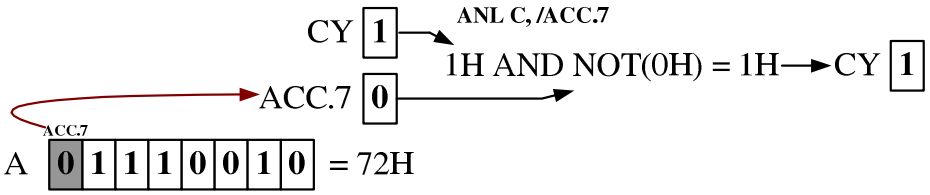
Hoạt động *(C) ← (C) AND NOT(*bit)

- *Ví dụ:* Cho biết trước (A)=72H, cờ CY=1.

Sau khi thực thi lệnh **ANL C, ACC.7** thì: cờ CY=0, (A)=72H



Sau khi thực thi lệnh **ANL C, /ACC.7** thì: cờ CY=1, (A)=72H



4.5. ORL C, <src-bit>

- *Chức năng:* OR logic hai bit.
- *Mô tả:* Nếu giá trị của bit nguồn là 1 thì phép toán sẽ *set* cờ CY=1 và ngược lại nếu giá trị của bit nguồn là 0 thì phép toán giữ nguyên giá trị hiện hành của cờ CY. Dấu gạch chéo / đặt trước toán hạng trong chương trình hợp ngữ chỉ ra rằng bit nguồn được lấy bù trước khi OR logic với cờ nhớ nhưng ***giá trị của bit nguồn không bị thay đổi*** bởi thao tác lấy bù. *Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

ORL C, bit

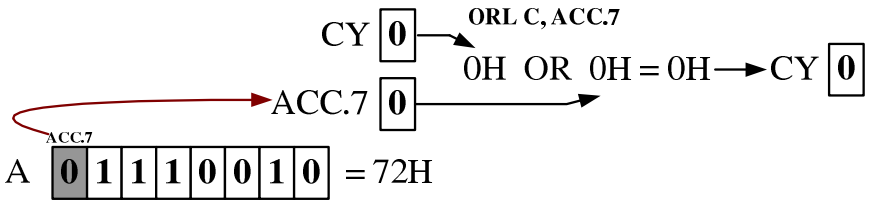
Số byte	2
Số chu kỳ	2
Mã đối tượng	01110010 bbbbbbbb
Hoạt động	(C) ← (C) OR (bit)

ORL C, /bit

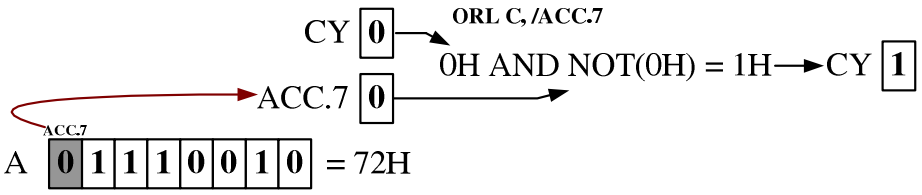
Số byte	*2*
Số chu kỳ	*2*
Mã đối tượng	*10100000 bbbbbbbb*
Hoạt động	(C) ← (C) OR NOT(bit)

- *Ví dụ:* Cho biết trước(A)=72H, cờ CY=0.

Sau khi thực thi lệnh **ORL C, ACC.7** thì: cờ CY=0, (A)=72H



Sau khi thực thi lệnh **ORL C, /ACC.7** thì: cờ CY=1, (A)=72H



- *Lưu ý:* Các lệnh trên bao gồm lệnh ANL và ORL nhưng không tồn tại lệnh XRL. Cho nên nếu ta cần XOR hai bit , BIT1 và BIT2, và kết quả được cất vào trong cờ nhớ (*CY*) thì ta sử dụng đoạn lệnh sau đây:

```
……….
MOV     C, BIT1          ;Nạp BIT1 vào cờ nhớ.
JNB     BIT2, SKIP       ;BIT2=0 thì C = BIT1.
CPL     C                ;BIT2=1 thì C\ = BIT1\.
SKIP:
……..                     ;BIT1 XOR BIT2
```

4.6. MOV <dest-bit>, <src-bit>

- *Chức năng:* Di chuyển bit nguồn (***src-bit***) đến bit đích (***dest-bit***).
- *Mô tả:* Nội dung của bit được chỉ ra bởi toán hạng thứ hai được sao chép vào vị trí được xác định bởi toán hạng thứ nhất. Một trong hai toán hạng phải là cờ nhớ và toán hạng còn lại có thể là bit bất kỳ được định địa chỉ bit. Bit nguồn không bị ảnh hưởng. *Các thanh ghi khác và các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

MOV C, bit

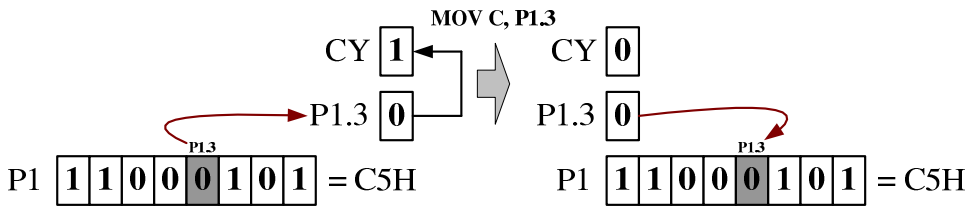
Số byte	2	
Số chu kỳ	1	
Mã đối tượng	10100010	bbbbbbbb
Hoạt động	(C) ← (bit)	

MOV bit, C

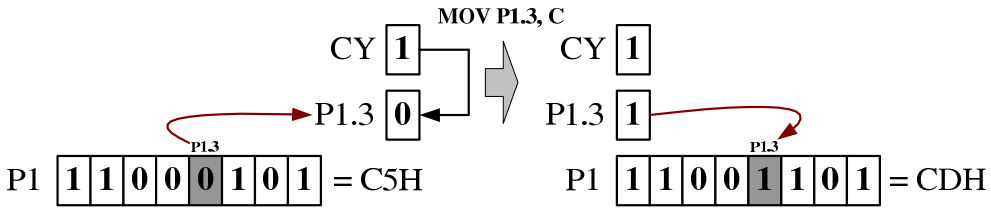
Số byte	*2*	
Số chu kỳ	*2*	
Mã đối tượng	*10010010*	*bbbbbbbb*
Hoạt động	*(bit) ← (C)*	

- *Ví dụ:* Cho biết trước cờ CY=1, (P1)=C5H.

Sau khi thực thi lệnh **MOV C, P1.3** thì: cờ CY=0, (P1)=C5H



Sau khi thực thi lệnh **MOV P1.3, C** thì: cờ CY=1, (P1)=CDH

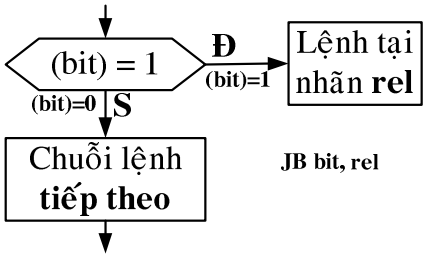


4.7. JB bit, rel

- *Chức năng:* Nhảy nếu bit bằng 1.
- *Mô tả:* Nếu bit chỉ ra trong lệnh bằng 1 thì nhảy đến địa chỉ được chỉ ra trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng, bit được kiểm tra sẽ không bị thay đổi.*

Số byte	3
Số chu kỳ	2
Mã đối tượng	00100000 bbbbbbbb eeeeeeee
Hoạt động	(PC) ← (PC) + 3 IF (bit) = 1 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JB bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ 1:* Cho biết trước (A)=56H, (P1)=CAH.

Sau khi thực thi chuỗi lệnh:

```
JB  P1.2, AAA
JB  ACC.2, BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (P1)=CAH và (A)=56H.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
      MOV     A, #0FFH
      MOV     P0, #9CH          ; 9CH = 10011100B
      JB      P0.7, AAA
      MOV     A, #00H
AAA:
      ……..
```

Sau khi thực thi chuỗi lệnh thì (A)=FFH, (P0)=9CH (*chương trình thực hiện lệnh nhảy **JB P0.7, AAA***).

- *Ví dụ 3:* Cho chuỗi lệnh:

```
        MOV     A, #0FFH
        MOV     P0, #9CH            ; 9CH = 10011100B
        JB      P0.5, AAA
        MOV     A, #00H
AAA:
        ……
```

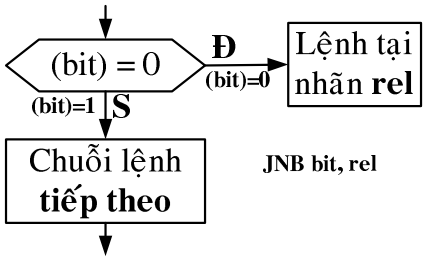
Sau khi thực thi chuỗi lệnh thì (A)=00H, (P0)=9CH (*chương trình không thực hiện lệnh nhảy* ***JB P0.5, AAA***).

4.8. JNB bit, rel

- *Chức năng:* Nhảy nếu bit bằng 0.
- *Mô tả:* Nếu bit chỉ ra trong lệnh bằng 0 thì nhảy đến địa chỉ được chỉ ra trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng.*

Số byte	3
Số chu kỳ	2
Mã đối tượng	00110000 bbbbbbbb eeeeeeee
Hoạt động	(PC) ← (PC) + 3 IF (bit) = 0 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JNB bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ 1:* Cho biết trước (A)=56H, (P1)=CAH.

Sau khi thực thi chuỗi lệnh:

```
        JNB P1.3, AAA
        JNB ACC.3, BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=56H và (P1)=CAH.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
        MOV     P1, #10H
        MOV     A, #6BH          ; 6BH = 01101011B
        JNB     ACC.5, AAA
        MOV     P1, #01H
AAA:
        ……
```

Sau khi thực thi chuỗi lệnh thì (A)=6BH, (P1)=01H (*chương trình không thực hiện lệnh nhảy **JNB ACC.5, AAA***).

- *Ví dụ 3:* Cho chuỗi lệnh:

```
        MOV     P1, #10H
        MOV     E0H, #6BH        ; 6BH = 01101011B
        JNB     E2H, AAA
        MOV     P1, #01H
AAA:
        ……
```

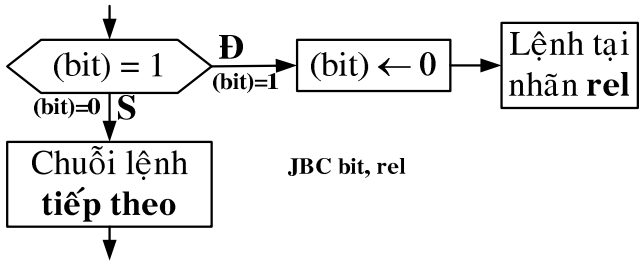
Sau khi thực thi chuỗi lệnh thì (A)=(E0H)=6BH, (P1)=10H (*chương trình thực hiện lệnh nhảy **JNB E2H, AAA***).

4.9. JBC bit, rel

- *Chức năng:* Nhảy nếu bit bằng 1 và xóa bit (*làm cho bit = 0*).
- *Mô tả:* Nếu bit được chỉ ra trong lệnh bằng 1 thì xóa bit này và rẽ nhánh đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Bit sẽ không được xóa nếu bit này đã là 0. *Các cờ không bị ảnh hưởng.*

Số byte	3
Số chu kỳ	2
Mã đối tượng	00010000 bbbbbbbb eeeeeeee
Hoạt động	(PC) ← (PC) + 3 IF (bit) = 1 THEN (bit) ← 0 (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JBC bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
  Khi lệnh này được dùng để làm thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.

- *Ví dụ 1:* Cho biết trước (A)=56H.

Sau khi thực thi chuỗi lệnh:

```
JBC ACC.3, AAA
JBC ACC.2, BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB và **(A)=52H**.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
        MOV   A, #76H
        MOV   P3, #9CH          ; 9CH = 10011100B
        JBC   P3.2, AAA
        MOV   A, #67H
AAA:
        ……….
```

Sau khi thực thi chuỗi lệnh thì (A)=76H, **(P3)=98H** (*chương trình thực hiện lệnh nhảy **JBC P3.2, AAA***).

- *Ví dụ 3:* Cho chuỗi lệnh:

```
        MOV   A, #76H
        MOV   B0H, #9CH         ; 9CH = 10011100B
        JBC   B1H, AAA
        MOV   A, #67H
AAA:
        ……….
```

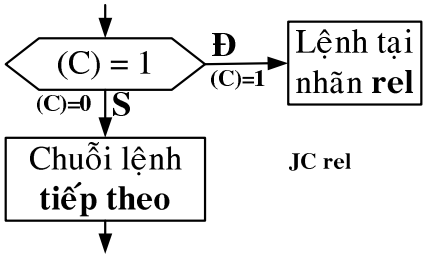
Sau khi thực thi chuỗi lệnh thì (A)=67H, **(P3)=(B0H)=9CH** (*chương trình không thực hiện lệnh nhảy **JBC B1H, AAA***).

4.10. JC rel

- *Chức năng:* Nhảy nếu cờ CY = 1.
- *Mô tả:* Nếu cờ CY = 1 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	01000000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (C) = 1 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JC rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ 1:* Cho biết trước (A)=7FH, cờ CY=0.

Sau khi thực thi chuỗi lệnh:

```
JC    AAA
ADD   A,#0F7H
JC    BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=76H, cờ CY=1.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
      MOV    A, #9AH
      MOV    R0, #76H
      ADD    A, R0
      JC     AAA
      MOV    R0, #67H
AAA:
      ……
```

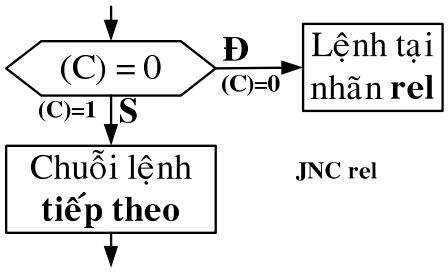
Sau khi thực thi chuỗi lệnh thì (A)=10H, (R0)=76H và cờ CY=1 (*chương trình thực hiện lệnh nhảy **JC AAA***).

4.11. JNC rel

- *Chức năng:* Nhảy nếu cờ CY = 0.
- *Mô tả:* Nếu cờ CY = 0 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	01010000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (C) = 0 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JNC rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.

- *Ví dụ 1:* Cho biết trước (A)=7FH, cờ CY=1.

Sau khi thực thi chuỗi lệnh:

```
JNC   AAA
ADD   A,#26H
JNC   BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=A5H, cờ CY=0.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
      MOV     A, #0B9H
      MOV     R1, #52H
      ADD     A, R1
      JNC     AAA
      MOV     R1, #25H
AAA:
      ……..……
```

Sau khi thực thi chuỗi lệnh thì (A)=0BH, (R1)=25H và cờ CY=1 (*chương trình không thực hiện lệnh nhảy **JNC AAA***).

5. Nhóm lệnh rẽ nhánh:

5.1. ACALL addr11

- *Chức năng:* Gọi đến địa chỉ tuyệt đối (*Absolute Call*)
- *Mô tả:* ACALL gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong lệnh (*xem thêm giải thích về chương trình con ở phần 5.3*). Chú ý rằng, **chương trình con được gọi phải được bắt đầu trong cùng khối 2K của bộ nhớ chương trình với byte đầu tiên của lệnh theo sau lệnh ACALL**. *Các cờ không bị ảnh hưởng.*

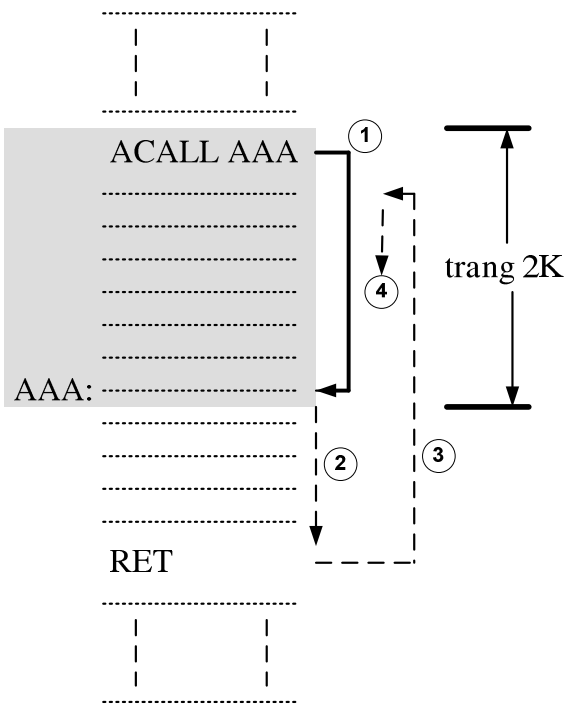
Số byte	2	
Số chu kỳ	2	
Mã đối tượng	aaa1001	aaaaaaaa

Lưu ý: aaa = A10–A8 và aaaaaaaa = A7–A0

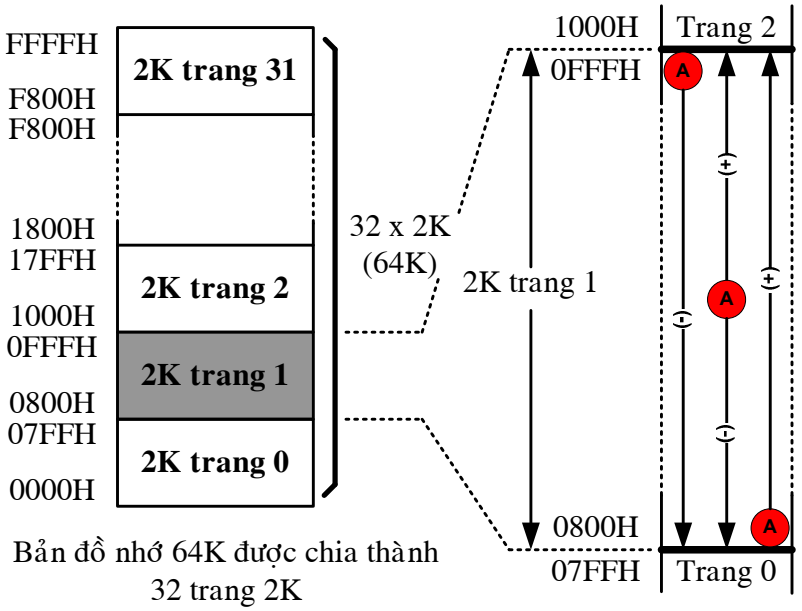
Hoạt động

(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC7–PC0)
(SP) ← (SP) + 1
((SP)) ← (PC15–PC8)
(PC10–PC0) ← địa chỉ trang

- *Mô tả:*



- *Lưu ý:* Do không gian bộ nhớ chương trình tối đa là 64KB cho nên ta có 32 trang (*khối*) và mỗi trang bắt đầu ở địa chỉ là biên của 2KB (*như là: 0000H, 0800H, 1000H, 1800H, ..., F800H*).



Bản đồ nhớ 64K được chia thành 32 trang 2K

- *Ví dụ 1:* Cho biết trước (SP)=07H, lệnh ACALL ở vị trí 0123H và nhãn AAA ở vị trí 0345H của bộ nhớ chương trình. Lệnh kế tiếp lệnh ACALL ở vị trí 0125H.

Sau khi thực thi lệnh **ACALL AAA** thì: (SP)=09H, (PC)=0345H và 2 ô nhớ của RAM nội (08H)=23H, (09H)=01H.

- *Ví dụ 2:* Cho biết trước (SP)=07H, lệnh ACALL ở vị trí 0800H và nhãn AAA ở vị trí 0700H của bộ nhớ chương trình.

Không thể thực thi lệnh **ACALL AAA** (lỗi lập trình) vì lệnh ACALL và nhãn AAA không nằm trong cùng một trang (lệnh ACALL thuộc trang 1, nhãn AAA thuộc trang 0).

5.2. LCALL addr16

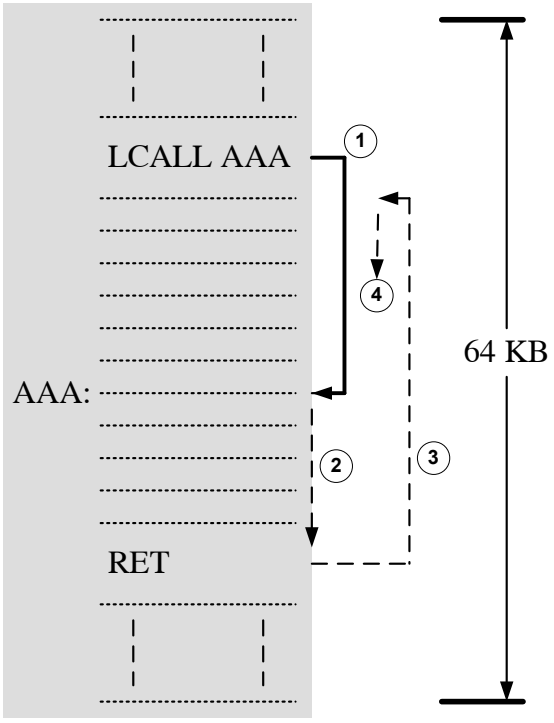
- *Chức năng:* Gọi một chương trình con (*Long Call*).
- *Mô tả:* LCALL gọi một chương trình con với địa chỉ bắt đầu chương trình con được chỉ ra trong lệnh (*xem thêm giải thích về chương trình con ở phần 5.3*). Chú ý rằng, **chương trình con có thể bắt đầu ở bất cứ nơi nào trong không gian bộ nhớ chương trình 64KB**. *Các cờ không bị ảnh hưởng.*

  Số byte 3
  Số chu kỳ 2
  Mã đối tượng 00010010 aaaaaaaa aaaaaaaa

  Lưu ý: byte 2 chứa các bit địa chỉ từ A15 – A8
  byte 3 chứa các bit địa chỉ từ A7 – A0.

  Hoạt động
  (PC) ← (PC) + 3
  (SP) ← (SP) + 1
  (SP) ← (PC7 – PC0)
  (SP) ← (SP) + 1
  (SP) ← (PC15 – PC8)
  (PC) ← addr15 – addr0

- *Mô tả:*



- *Ví dụ:* Cho biết trước (SP)=07H, lệnh LCALL ở vị trí 0123H và nhãn AAA ở vị trí 1234H của bộ nhớ chương trình. Lệnh kế tiếp lệnh ACALL ở vị trí 0126H.

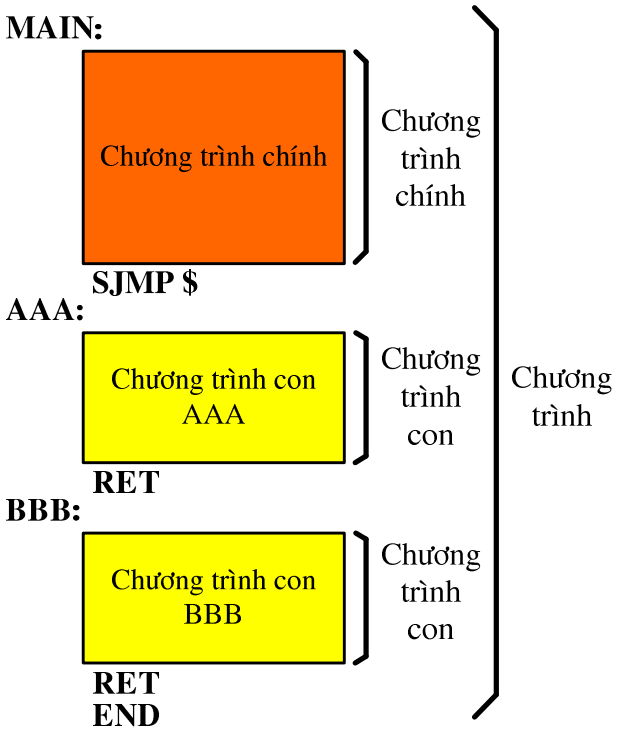
Sau khi thực thi lệnh **LCALL AAA** thì: (SP)=09H, (PC)=1234H và 2 ô nhớ của RAM nội (08H)=26H, (09H)=01H.

5.3. RET

- *Chức năng:* Trở về từ chương trình con (*Return*).
- *Mô tả:* RET lấy lại các byte cao và byte thấp của PC từ *stack*, giảm con trỏ *stack* bởi 2. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ chứa trong PC, trong trường hợp tổng quát là lệnh ngay sau lệnh ACALL hoặc LCALL. *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	2
Mã đối tượng	00100010
Hoạt động	(PC15 – PC8) ← ((SP)) (SP) ← (SP) – 1 (PC7 – PC0) ← ((SP)) (SP) ← (SP) – 1

- *Lưu ý:* Chương trình con có một số qui định sau:
  - Là một chuỗi gồm nhiều lệnh được kết hợp lại với nhau để thực hiện một công việc nào đó.
  - Bắt đầu bằng một **NHÃN**, do người lập trình tự đặt ra (*nhãn này chính là tên của chương trình con*).
  - Kết thúc bằng lệnh **RET**.
  - Được đặt ở cuối chương trình, phía trên chỉ dẫn kết thúc chương trình **END**.
  - Giữa chương trình con và chương trình chính phải được cách ly bằng lệnh:
    - ✓ **SJMP $**.
    - ✓ **SJMP MAIN** (*MAIN: là nhãn bất kỳ thuộc chương trình chính*).
  - Chương trình con có thể được gọi ra nhiều lần tại bất kỳ thời điểm nào, tùy thuộc vào người lập trình yêu cầu (*thông qua các lệnh gọi ACALL, LCALL và các tín hiệu ngắt*).
  - Trong một chương trình có thể có một hoặc nhiều chương trình con.



- *Ví dụ 1:* Dựa vào *Ví dụ 1* của phần 5.1. Cho biết trước (SP)=09H; ô nhớ RAM nội (08H)=25H và (09H)=01H .

Sau khi thực thi lệnh **RET** thì: (SP)=07H và chương trình được tiếp tục với lệnh tại địa chỉ 0125H (*vị trí của lệnh kế tiếp lệnh ACALL*).

- *Ví dụ 2:* Dựa vào *Ví dụ* của phần 5.2. Cho biết trước (SP)=09H; ô nhớ RAM nội (08H)=26H và (09H)=01H .

Sau khi thực thi lệnh **RET** thì: (SP)=07H và chương trình được tiếp tục với lệnh tại địa chỉ 0126H (*vị trí của lệnh kế tiếp lệnh LCALL*).

5.4. RETI

- *Chức năng:* Trở về từ chương trình con phục vụ ngắt.
- *Mô tả:* RETI lấy lại các byte cao và byte thấp của PC từ *stack*, phục hồi logic ngắt để có thể nhận các ngắt khác có cùng ưu tiên ngắt với ngắt vừa xử lý. Con trỏ *stack* được giảm bởi 2. Không có thanh ghi nào khác bị ảnh hưởng; *PSW không được tự động phục hồi trở lại trạng thái trước khi xử lý ngắt*. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ chứa trong PC, trong trường hợp tổng quát là lệnh ngay sau điểm mà yêu cầu ngắt được phát hiện. *Nếu có một ngắt có ưu tiên ngắt thấp hơn hoặc cùng ưu tiên ngắt được treo khi lệnh RETI được thực thi, một lệnh được thực thi trước khi ngắt đang treo được xử lý.*

Số byte	1
Số chu kỳ	2
Mã đối tượng	00110010
Hoạt động	(PC15 – PC8) ← ((SP))
	(SP) ← (SP) – 1
	(PC7 – PC0) ← ((SP))
	(SP) ← (SP) – 1

- *Lưu ý:* Lệnh **RETI** trả điều khiển về chương trình gọi từ một ISR (*ISR: Interrupt Service Routine: chương trình con phục vụ ngắt*). Điểm khác nhau giữa RETI và RET là RETI có báo hiệu cho hệ thống điều khiển ngắt rằng quá trình xử lý ngắt đã xong. Nếu trường hợp không có một ngắt nào được duy trì trong thời gian RETI thực thi thì lệnh RETI sẽ hoạt động như lệnh RET.
- *Ví dụ:* Cho biết trước (SP)=0BH; ô nhớ RAM nội (0AH)=23H và (0BH)=01H. một tín hiệu ngắt được phát hiện trong lệnh ở địa chỉ 0123H đang thực thi.

Sau khi thực thi lệnh **RETI** thì: (SP)=09H và chương trình được tiếp tục với lệnh tại địa chỉ 0123H.

5.5. AJMP addr11

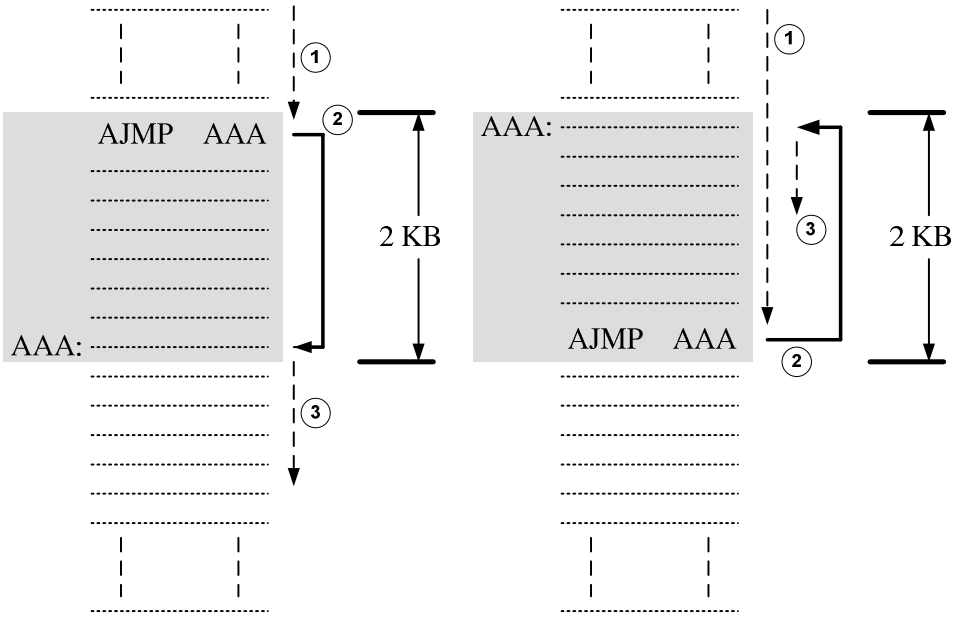
- *Chức năng:* Nhảy đến địa chỉ tuyệt đối (*Absolute Jump*).
- *Mô tả:* AJMP chuyển việc thực thi chương trình đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **đích nhảy đến phải ở trong cùng khối 2K của bộ nhớ chương trình với byte đầu tiên của lệnh theo sau lệnh AJMP**.

Số byte	2	
Số chu kỳ	2	
Mã đối tượng	aaa00001	aaaaaaaa

Ghi chú: aaa = A10 – A8 và aaaaaaaa = A7 – A0

Hoạt động	(PC) ← (PC) + 2
	(PC10 – PC0) ← địa chỉ trang

- *Mô tả:*



- *Ví dụ:* Cho biết trước lệnh AJMP ở vị trí 0345H và nhãn AAA ở vị trí 0123H của bộ nhớ chương trình.

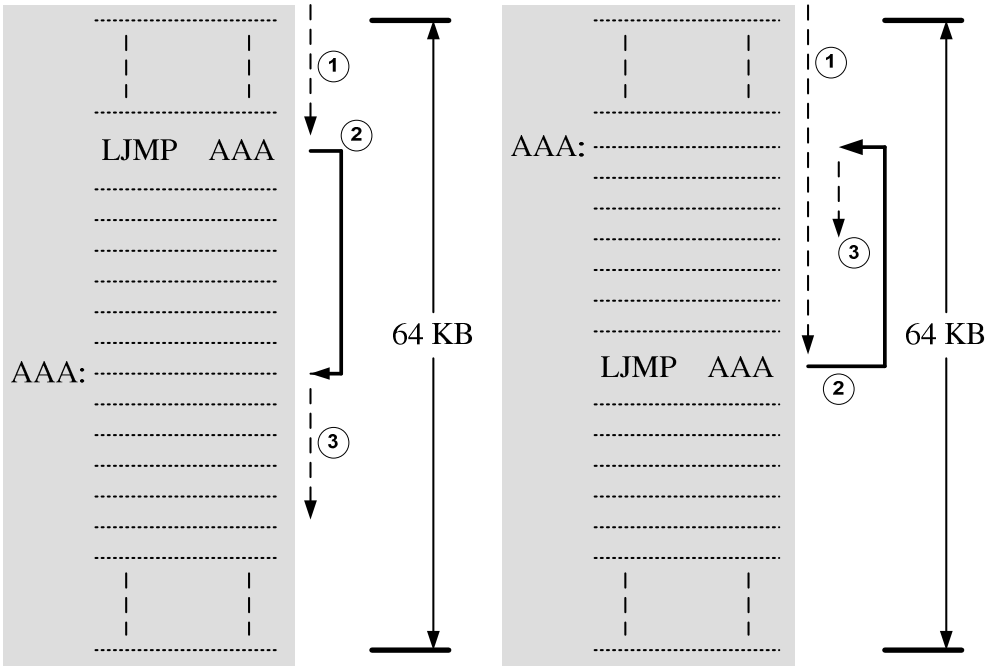
Sau khi thực thi lệnh **AJMP AAA** thì: (PC)=0123H.

5.6. LJMP addr16

- *Chức năng:* Nhảy dài (Long Jump).
- *Mô tả:* LJMP tạo ra một rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **địa chỉ đích có thể ở bất cứ nơi nào trong không gian địa chỉ của bộ nhớ chương trình 64KB**. *Các cờ không bị ảnh hưởng.*

Số byte	3
Số chu kỳ	2
Mã đối tượng	00010010 aaaaaaaa aaaaaaaa
Lưu ý:	byte 2 chứa các bit địa chỉ từ A15–A8 byte 3 chứa các bit địa chỉ từ A7–A0
Hoạt động	(PC) ← addr15 – addr0

- *Mô tả:*



- *Ví dụ:* Cho biết trước lệnh LJMP ở vị trí 0123H và nhãn AAA ở vị trí 1234H của bộ nhớ chương trình.

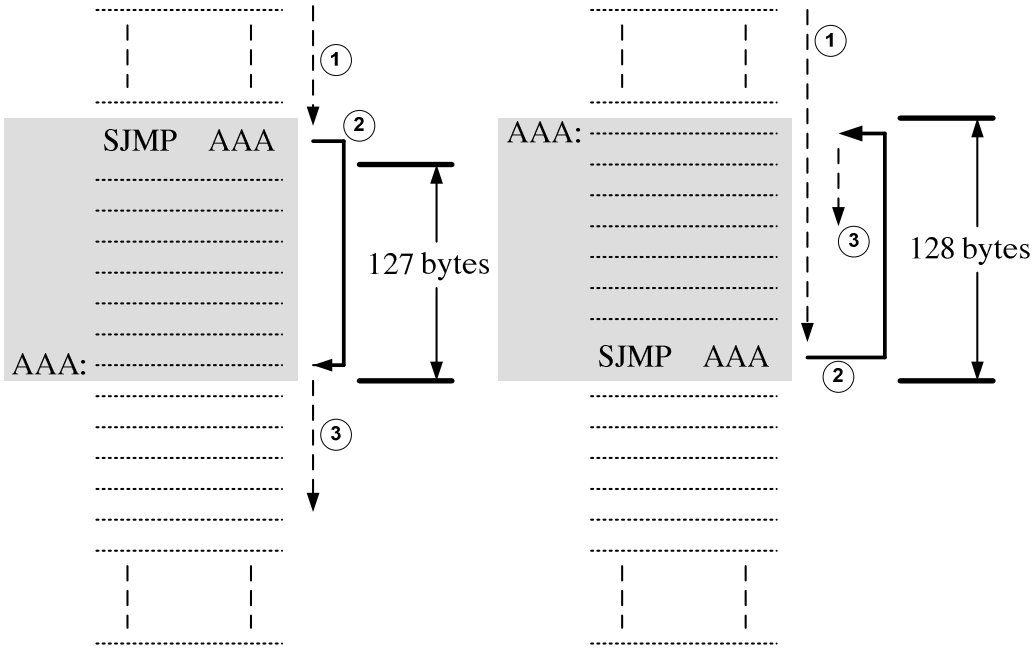
Sau khi thực thi lệnh **LJMP AAA** thì: (PC)=1234H.

5.7. SJMP rel

- *Chức năng:* Nhảy ngắn (*Short Jump*).
- *Mô tả:* Điều khiển chương trình rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **tầm nhảy cho phép là 128 byte trước lệnh và 127 byte sau lệnh**.

Số byte	2
Số chu kỳ	2
Mã đối tượng	10000000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2
	(PC) ← (PC) + byte_2

- *Mô tả:*



- *Lưu ý:* Lệnh **SJMP $** là một lệnh vòng lặp vô tận (lệnh nhảy tại chỗ), thường được sử dụng khi cần kết thúc một chương trình điều khiển của chip 8051 (*lưu ý rằng các tín hiệu ngắt vẫn hoạt động bình thường, vì thế đây chính là phương pháp duy nhất để thoát khỏi vòng lặp vô tận này)*.
- *Ví dụ:* Cho biết trước lệnh SJMP nằm tại địa chỉ 0100H và nhãn AAA nằm tại địa chỉ 0123H trong bộ nhớ chương trình.

Sau khi thực thi lệnh **SJMP AAA** thì: (PC)=0123H.

5.8. JMP @A+DPTR

- *Chức năng:* Nhảy gián tiếp.
- *Mô tả:* Cộng giá trị 8-bit không dấu chứa trong thanh ghi A với con trỏ 16-bit và nạp kết quả tổng cho bộ đếm chương trình PC. Đây chính là địa chỉ của lệnh kế tiếp được tìm nạp. *Cả hai, thanh ghi A và con trỏ dữ liệu đều không bị thay đổi. Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	2
Mã đối tượng	01110011
Hoạt động	(PC) ← (PC) + (A) + (DPTR)

- *Ví dụ:* Cho biết trước (A)=00H, 02H, 04H, 06H.

Sau khi thực thi chuỗi lệnh:

```
            MOV   DPTR, #JMP_TBL
            JMP   @A+DPTR
JMP_TBL:
            AJMP  AAA    ;Độ dài lệnh là 2 byte.
            AJMP  BBB    ;Độ dài lệnh là 2 byte.
            AJMP  CCC    ;Độ dài lệnh là 2 byte.
            AJMP  DDD    ;Độ dài lệnh là 2 byte.
```

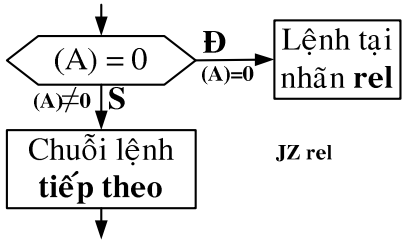
thì: nếu (A)=00H, chương trình được tiếp tục với lệnh tại nhãn AAA.
nếu (A)=02H, chương trình được tiếp tục với lệnh tại nhãn BBB.
nếu (A)=04H, chương trình được tiếp tục với lệnh tại nhãn CCC.
nếu (A)=06H, chương trình được tiếp tục với lệnh tại nhãn DDD.

5.9. JZ rel

- *Chức năng:* Nhảy nếu nội dung thanh ghi A bằng 0.
- *Mô tả:* Nếu tất cả các bit của thanh ghi A đều bằng 0 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng. Nội dung thanh ghi A không bị thay đổi.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	01100000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (A) = 0 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JZ rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ:* Cho biết trước (A)=01H.

Sau khi thực thi chuỗi lệnh:

```
JZ   AAA
DEC  A
JZ   BBB
```

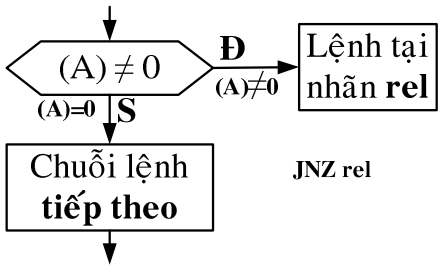
thì chương trình được tiếp tục với lệnh tại nhãn BBB.

5.10. JNZ rel

- *Chức năng:* Nhảy nếu nội dung thanh ghi A khác 0.
- *Mô tả:* Nếu thanh ghi A có một bit bất kỳ bằng 1 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng. Nội dung thanh ghi A không bị thay đổi.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	01110000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (A) < > 0 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JNZ rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ:* Cho biết trước (A)=00H.

Sau khi thực thi chuỗi lệnh:

```
JNZ  AAA
INC  A
JNZ  BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB.

5.11. CJNE <dest-byte>, <src-byte>, rel

- *Chức năng:* So sánh và nhảy nếu không bằng.
- *Mô tả:* CJNE so sánh giá trị của 2 toán hạng (*src-byte*) và (*dest-byte*) rồi rẽ nhánh đến địa chỉ được chỉ ra trong lệnh nếu các giá trị của 2 toán hạng này không bằng nhau. *Cờ CY = 1 nếu giá trị nguyên không dấu của (dest- byte) nhỏ hơn giá trị nguyên không dấu của (src-byte) và ngược lại CY = 0. Không có toán hạng nào trong 2 toán hạng bị ảnh hưởng.*
- *Các dạng lệnh:*

CJNE A, direct, rel

Số byte	3
Số chu kỳ	2
Mã đối tượng	10110101 aaaaaaaa eeeeeeee

Hoạt động	(PC) ← (PC) + 3
	IF (A) < > (direct) THEN
	(PC) ← (PC) + địa chỉ tương đối
	IF (A) < (direct) THEN
	(C) ← 1
	ELSE
	(C) ← 0

CJNE A, #data, rel

Số byte	3
Số chu kỳ	2
Mã đối tượng	10110100 dddddddd eeeeeeee

Hoạt động	(PC) ← (PC) + 3
	IF (A) < > #data THEN
	(PC) ← (PC) + địa chỉ tương đối
	IF (A) < #data THEN
	(C) ← 1
	ELSE
	(C) ← 0

CJNE Rn, #data, rel

Số byte	3
Số chu kỳ	2
Mã đối tượng	10111rrr dddddddd eeeeeeee

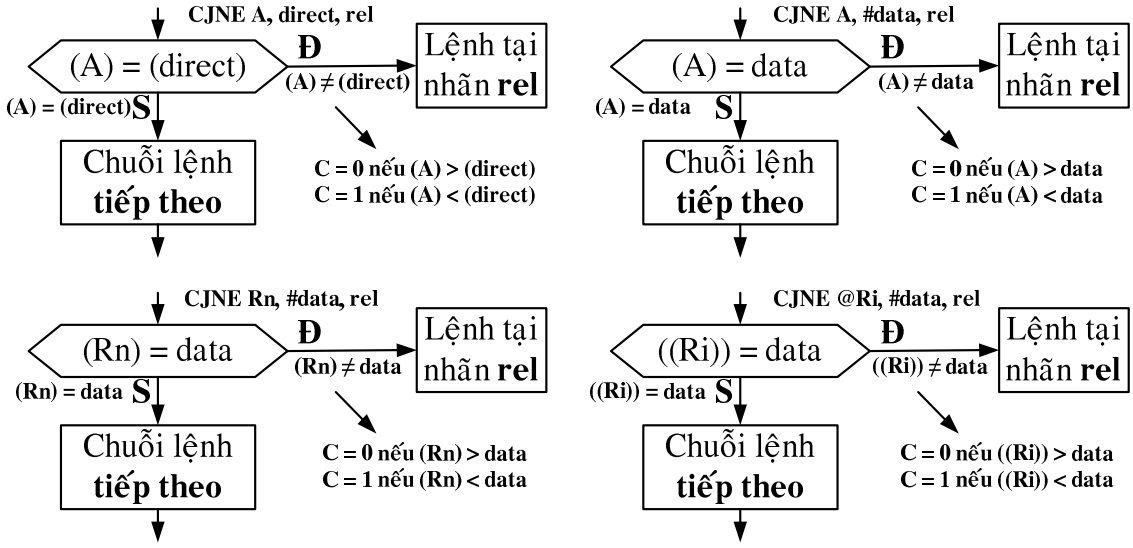
Hoạt động	(PC) ← (PC) + 3
	IF (Rn) < > #data THEN
	(PC) ← (PC) + địa chỉ tương đối
	IF (Rn) < #data THEN
	(C)← 1
	ELSE
	(C) ← 0

CJNE @Ri, #data, rel

Số byte	3
Số chu kỳ	2
Mã đối tượng	1011011i ddddddddd eeeeeeee

Hoạt động	(PC) ← (PC) + 3
	IF ((Ri)) < > #data THEN
	(PC) ← (PC) + địa chỉ tương đối
	IF ((Ri)) < #data THEN
	(C)← 1
	ELSE
	(C) ← 0

- *Lưu đồ:*



- *Ví dụ 1:* Cho biết trước (A)=34H, (01H)=(R1)=34H, (34H)=B9H.

Sau khi thực thi chuỗi lệnh:

```
CJNE   A, 01H, AAA
CJNE   @R1, #9BH, BBB
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB và cờ C=0.

- *Ví dụ 2:* Cho chuỗi lệnh:

```
        CLR     C
        MOV     A, #40H
        MOV     40H, #0B1H
        CJNE    A, 40H, AAA
        MOV     40H, #1BH
AAA:
        ADDC    A, 40H
```

Sau khi thực thi chuỗi lệnh thì (A)=F2H, (40H)=B1H, CY=0.

- *Ví dụ 3:* Ứng dụng cho phép so sánh lớn hơn hay nhỏ hơn. Giả sử:
  - Khi ta muốn nhảy đến nhãn BIG nếu (A) ≥ #20H, thì các lệnh sau được sử dụng:

```
……
CJNE    A, #20H, $+3    ;So sánh (A) với con số 20H.
JNC     BIG             ;Nhảy đến BIG nếu (A) ≥ #20H.
……
```

  - Khi ta muốn nhảy đến nhãn SMALL nếu (A) < #20H, thì các lệnh sau được sử dụng:

```
……
CJNE    A, #20H, $+3    ;So sánh (A) với con số 20H.
JC      SMALL           ;Nhảy đến SMALL nếu (A) < #20H.
……
```

- *Lưu ý:* Ký hiệu $ là một ký hiệu của trình dịch hợp ngữ, biểu thị địa chỉ của lệnh hiện hành (*vì CJNE có độ dài lệnh là 3 byte nên $+3 sẽ là địa chỉ của lệnh tiếp theo CJNE, tức là lệnh JC/JNC*).

5.12. DJNZ <byte>, <rel-addr>

- *Chức năng:* Giảm và nhảy nếu byte khác 0.
- *Mô tả:* DJNZ giảm byte chỉ ra bởi toán hạng đầu trong lệnh và rẽ nhánh đến địa chỉ được chỉ ra bởi toán hạng thứ hai trong lệnh nếu kết quả sau khi giảm khác 0. *Nếu giá trị ban đầu của byte là 00H ta sẽ có tràn sang 0FFH. Các cờ không bị ảnh hưởng.*
- *Các dạng lệnh:*

DJNZ Rn, rel

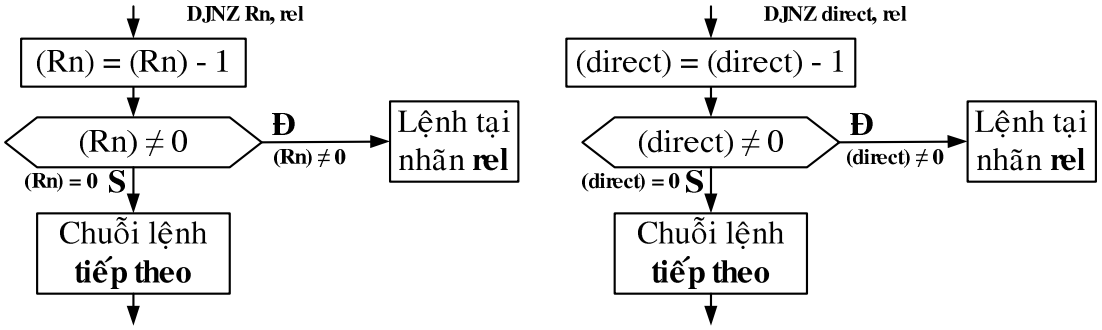
Số byte	2
Số chu kỳ	2
Mã đối tượng	11011rrr eeeeeeee
Hoạt động	(PC) ← (PC) + 2 (Rn) ← (Rn) – 1 IF (Rn) < > 0 THEN (PC) ← (PC) + byte_2

DJNZ direct, rel

Số byte	*3*
Số chu kỳ	*2*
Mã đối tượng	*11010101 aaaaaaaa eeeeeeee*
Hoạt động	*(PC) ← (PC) + 2* *(direct) ← (direct) – 1* *IF (direct) < > 0 THEN* *(PC) ← (PC) + byte_2*

- *Lưu ý:* Khi lệnh này được dùng để làm thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.

- *Lưu đồ:*



- *Lưu ý:* Khi lệnh DJNZ được thực hiện thì giá trị trong thanh ghi Rn được giãm (-1) trước khi đem ra so sánh với giá trị "0".
- *Ví dụ 1:* Cho chuỗi lệnh sau:

```
        MOV   A, #50
        MOV   R2, #8
AAA:    ADD   A, #1
        DJNZ  R2, AAA
```

Sau khi thực thi chuỗi lệnh thì (A)=58, (R2)=0.

- *Ví dụ 2:* Cho chuỗi lệnh sau:

```
        MOV   A, #50
        MOV   R3, #56
AAA:    ADD   A, #1
        DJNZ  R3, AAA
```

- Sau khi thực thi chuỗi lệnh thì (A)=106, (R1)=0.
- *Ví dụ 3:* Cho chuỗi lệnh sau:

```
        MOV   A, #50
        MOV   R1, #0
AAA:    ADD   A, #1
        DJNZ  R1, AAA
```

Sau khi thực thi chuỗi lệnh thì (A)=50, (R1)=0.

- *Lưu ý rằng:* Qua ba ví dụ về lệnh DJNZ ở trên cho ta một nhận xét như sau:
  - Nếu (Rn)=**8** ⇨ Lệnh ADD được thực hiện **8 lần**.
  - Nếu (Rn)=**56** ⇨ Lệnh ADD được thực hiện **56 lần**.
  - Nếu (Rn)=**0** ⇨ Lệnh ADD được thực hiện **256 lần**.

5.13. NOP

- *Chức năng:* Không làm gì (*No Operation*).
- *Mô tả:* Việc thực thi chương trình tiếp tục với lệnh tiếp theo. *Không có thanh ghi hay cờ nào bị ảnh hưởng.*

Số byte	*1*
Số chu kỳ	*1*
Mã đối tượng	*00000000*
Hoạt động	*(PC) ← (PC) + 1*

Danh sách các lệnh làm ảnh hưởng tới các cờ CY, AC và OV

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Ghi chú: **X** là cờ tương ứng bị ảnh hưởng.
0 là cờ tương ứng bằng 0. **1** là cờ tương ứng bằng 1.

IV. CÁC VÍ DỤ ỨNG DỤNG VỀ TẬP LỆNH:

- *Ví dụ 1:* Viết đoạn lệnh để xóa thanh ghi A và sau đó cộng 9 vào thanh ghi A 10 lần. Sau khi hoàn tất thì cất giá trị trong thanh ghi A vào thanh ghi R7.

Giải

```
        MOV     A, #0           ;Xoá ACC, A = 0.
        MOV     R0, #10         ;Nạp số lần lặp, R0 = 10.
BACK:
        ADD     A, #9           ;Cộng thêm 9 vào ACC.
        DJNZ    R0, BACK        ;Kiểm tra số lần lặp lại, 10 lần.
        MOV     R7, A           ;Cất ACC vào thanh ghi R7.
```

- *Ví dụ 2:* Viết đoạn lệnh để nạp vào thanh ghi A với giá trị FFH và sau đó lấy bù thanh ghi A 500 lần.

Giải

```
        MOV     A, #0FFH        ;Nạp A = FFH.
        MOV     R0, #10         ;Nạp số lần lặp 1, R0 = 10.
LOOP:
        MOV     R1, #50         ;Nạp số lần lặp 2, R1 = 50.
BACK:
        CPL     A               ;Lấy bù ACC.
        DJNZ    R1, BACK        ;Kiểm tra số lần lặp 2 (vòng trong), 50 lần.
        DJNZ    R0, LOOP        ;Kiểm tra số lần lặp 1(vòng ngoài), 10 lần.
```

- *Ví dụ 3:* Viết đoạn lệnh để xác định xem R0 có chứa giá trị 0 hay không? Nếu không thì nạp vào R0 giá trị FFH.

Giải

```
    MOV     A, R0          ;Chuyển nội dung R5 vào A.
    JNZ     NEXT           ;Nhảy đến NEXT nếu A ? 0 (thoát ra).
    MOV     R0, #0FFH      ;Nạp  R0 = FFH nếu A = 0.
NEXT:
```

- *Ví dụ 4:* Viết đoạn lệnh để tìm tổng của 79H, F5H và E2H. Ghi byte thấp của tổng vào R0 và byte cao của tổng vào R1.

Giải

```
    MOV     A, #0          ;Xoá ACC, A = 0.
    MOV     R1, #0         ;Xoá R1, R1 = 0.
    ADD     A, #79H        ;Cộng thêm 79H (A = 79H, CY = 0).
    JNC     NO_CY1         ;Nhảy nếu CY = 0.
    INC     R1             ;Nếu CY = 1 thì tăng R1.
NO_CY1:
    ADD     A, #0F5H       ;Cộng thêm F5H (A = 6EH, CY =1).
    JNC     NO_CY2         ;Nhảy nếu CY = 0.
    INC     R1             ;Nếu CY = 1 thì tăng R1.
NO_CY2:
    ADD     A, #0E2H       ;Cộng thêm E2H (A = 50H, CY = 1).
    JNC     NO_CY3         ;Nhảy nếu CY = 0.
    INC     R1             ;Nếu CY = 1 thì tăng R1.
NO_CY3:
    MOV     R0, A          ;R0 = 50H (byte thấp), R1 = 02H (byte cao).
```

- *Ví dụ 5:* Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện các lệnh sau:

```
MOV  R3, #55H      DEC  R3        DJNZ  R2, AAA
LJMP  AAA          SJMP  AAA     NOP
MUL  AB
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11{,}0592MHz} = 1{,}085(\mu s)$$

Lệnh:	***Số chu kỳ máy:***	***Thời gian thực hiện:***
MOV R3, #55H	1	t = 1 x 1,085 (µs) = 1,085 (µs)
DEC R3	1	t = 1 x 1,085 (µs) = 1,085 (µs)
DJNZ R2, AAA	2	t = 2 x 1,085 (µs) = 2,17 (µs)
LJMP AAA	2	t = 2 x 1,085 (µs) = 2,17 (µs)
SJMP AAA	2	t = 2 x 1,085 (µs) = 2,17 (µs)
NOP	1	t = 1 x 1,085 (µs) = 1,085 (µs)
MUL AB	4	t = 4 x 1,085 (µs) = 4,34 (µs)

- *Ví dụ 6:* Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

```
MOV  A, #55H
MOV  P1, A
CPL  A
END
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11{,}0592MHz} = 1{,}085(\mu s)$$

Đoạn lệnh:	***Số chu kỳ máy:***
MOV A, #55H	1
MOV P1, A	1
CPL A	1
END	

Tổng thời gian thực hiện đoạn lệnh trên là:

t = (1 + 1 + 1) x 1,085 (µs) = 3,255 (µs)

- *Ví dụ 7:* Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

```
DELAY:
      MOV  R3, #200
      DJNZ R3, $
      RET
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11{,}0592MHz} = 1{,}085(\mu s)$$

Đoạn lệnh:	***Số chu kỳ máy:***
DELAY:	
MOV R3, #200	1
DJNZ R3, $	2
RET	1

Tổng thời gian thực hiện đoạn lệnh trên là:

t = [1 + (2 x 200) + 1] x 1,085 (µs) = 436,17 (µs)

- *Ví dụ 8:* Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592 MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11{,}0592MHz} = 1{,}085(\mu s)$$

Đoạn lệnh:	***Số chu kỳ máy:***
DELAY:	
MOV R3, #250	1
HERE:	
NOP	1
NOP	1
NOP	1
NOP	1
DJNZ R3, HERE	2
RET	1

Tổng thời gian thực hiện đoạn lệnh trên là:

t = [1 + ((1 + 1 + 1 + 1 + 2) x 250) + 1] x 1,085 (µs) = 1629,67 (µs)

- *Ví dụ 9:* Viết đoạn lệnh để c?t giá trị của thanh ghi R5, R6 và A vào ngăn xếp. Sau đó lấy ra và cho lần lượt vào các thanh ghi R2, R3 và B tương ứng.

Giải

```
PUSH    05H     ;Cất R5 vào ngăn xếp.
PUSH    06H     ;Cất R6 vào ngăn xếp.
PUSH    0E0H    ;Cất ACC vào ngăn xếp.
POP     0F0H    ;Lấy từ ngăn xếp cho vào B, (B) = (A).
POP     03H     ;Lấy từ ngăn xếp cho vào R3, (R3) = (R6).
POP     02H     ;Lấy từ ngăn xếp cho vào R2, (R2) = (R5).
```

- *Ví dụ 10:* Viết đoạn lệnh để chuyển giá trị trong ô nhớ có địa chỉ 55H vào các ô nhớ RAM tại địa chỉ từ 40H – 44H, sử dụng:
  - Chế độ định địa chỉ trực tiếp.
  - Chế độ định địa chỉ gián tiếp (không dùng vòng lặp).
  - Chế độ định địa chỉ gián tiếp (dùng vòng lặp).

Giải

Chế độ định địa chỉ trực tiếp:

```
MOV     A, #55H     ;Nạp giá trị 55H vào thanh ghi A.
MOV     40H, A      ;Sao nội dung của A vào ô nhớ 40H.
MOV     41H, A      ;Sao nội dung của A vào ô nhớ 41H.
MOV     42H, A      ;Sao nội dung của A vào ô nhớ 42H.
MOV     43H, A      ;Sao nội dung của A vào ô nhớ 43H.
MOV     44H, A      ;Sao nội dung của A vào ô nhớ 44H.
```

- ***Chế độ định địa chỉ gián tiếp (không dùng vòng lặp):***

```
MOV     A, #55H     ;Nạp giá trị 55H vào thanh ghi A.
MOV     R0, #40H    ;Nạp địa chỉ bắt đầu, R0 = 40H.
MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC     R0          ;Tăng con trỏ, R0 = 41H.
MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC     R0          ;Tăng con trỏ, R0 = 42H.
MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC     R0          ;Tăng con trỏ, R0 = 43H.
MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC     R0          ;Tăng con trỏ, R0 = 44H.
MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
```

- ***Chế độ định địa chỉ gián tiếp (dùng vòng lặp):***

```
        MOV     A, #55H     ;Nạp giá trị 55H vào thanh ghi A.
        MOV     R0, #40H    ;Nạp địa chỉ bắt đầu, R0 = 40H.
        MOV     R1, #5      ;Nạp số lần lặp lại, R1 = 5.
LOOP:
        MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
        INC     R0          ;Tăng con trỏ.
        DJNZ    R1, LOOP    ;Kiểm tra số lần lặp cho đến khi số lần = 0.
```

- *Ví dụ 11:* Viết đoạn lệnh để xóa 16 ô nhớ RAM nội có địa chỉ bắt đầu từ 60H.

Giải

```
        CLR     A           ;Xoá ACC, A = 0.
        MOV     R0, #60H    ;Nạp địa chỉ bắt đầu, R0 = 60H.
        MOV     R1, #16     ;Nạp số lần lặp lại, R1 = 16.
LOOP:
        MOV     @R0, A      ;Sao nội dung A vào ô nhớ do R0 trỏ đến.
        INC     R0          ;Tăng con trỏ.
        DJNZ    R1, LOOP    ;Kiểm tra số lần lặp cho đến khi số lần = 0.
```

- *Ví dụ 12:* Viết đoạn lệnh để chuyển một khối dữ liệu gồm 10 byte từ vị trí ô nhớ RAM nội bắt đầu tại địa chỉ 35H đến các vị trí ô nhớ RAM nội bắt đầu tại địa chỉ 60H.

Giải

```
        MOV     R0, #35H    ;Nạp địa chỉ bắt đầu (nguồn), R0 = 35H.
        MOV     R1, #60H    ;Nạp địa chỉ bắt đầu (đích), R1 = 60H.
        MOV     R2, #10     ;Nạp số lần lặp lại, R2 = 5.
LOOP:
        MOV     A, @R0      ;Sao nội dung ô nhớ do R0 trỏ đến vào A.
        MOV     @R1, A      ;Sao nội dung A vào ô nhớ do R1 trỏ đến.
        INC     R0          ;Tăng con trỏ (nguồn).
        INC     R1          ;Tăng con trỏ (đích).
        DJNZ    R2, LOOP    ;Kiểm tra số lần lặp cho đến khi số lần = 0.
```

- *Ví dụ 13:* Giả sử chữ "DHCN" được lưu trong ROM nội tại vùng nhớ có địa chỉ bắt đầu là 200H. Hãy phân tích cách chương trình sau đây hoạt động và xác định xem chữ "DHCN" sẽ được lưu vào đâu sau khi chương trình hoàn tất?

```
        ORG     0000H           ;Địa chỉ lưu chương trình trong ROM.
        MOV     DPTR, #200H     ;Nạp con trỏ vùng dữ liệu, DPTR=200H.
        CLR     A               ;Xoá ACC, A = 0.
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) = 200H trỏ đến đưa vào A.
        MOV     R0, A           ;Cất dữ liệu đó vào R0.
        INC     DPTR            ;Tăng con trỏ, DPTR=201H.
        CLR     A               ;Xoá ACC, A = 0.
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) = 201H trỏ đến đưa vào A.
        MOV     R1, A           ;Cất dữ liệu đó vào R1.
        INC     DPTR            ;Tăng con trỏ, DPTR=202H.
        CLR     A               ;Xoá ACC, A = 0.
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) = 202H trỏ đến đưa vào A.
        MOV     R2, A           ;Cất dữ liệu đó vào R2.
        INC     DPTR            ;Tăng con trỏ, DPTR=203H.
        CLR     A               ;Xoá ACC, A = 0.
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) = 203H trỏ đến đưa vào A.
        MOV     R3, A           ;Cất dữ liệu đó vào R3.
        SJMP    $               ;Dừng chương trình.
        ORG     200H            ;Địa chỉ lưu chương trình trong ROM.
MYDATA:
        DB      "DHCN"          ;Khai báo dữ liệu.
        END                     ;Kết thúc chương trình.
```

Giải

Theo chương trình trên, các ô nhớ của bộ nhớ chương trình (ROM) có địa chỉ 200H - 203H chứa các nội dung sau: ***(200H) = 44H = 'D', (201H) = 48H = 'H', (202H) = 43H = 'C', (203H) = 4EH = 'N'***.

Đầu tiên với (DPTR) = 200H và (A) = 0. Lệnh ***MOVC A, @A+DPTR*** chuyển nội dung của ô nhớ có địa chỉ 200H (A + DPTR = 0 + 200H) trong ROM vào A. Thanh ghi A lúc này sẽ chứa giá trị 44H là mã ASCII của ký tự "D". Ký tự này được cất vào thanh ghi R0.

Tiếp theo, DPTR được tăng lên (DPTR = 201H) và A được xoá (A = 0) để lấy nội dung của vị trí nhớ kế tiếp trong ROM có địa chỉ là 201H (A + DPTR = 0 + 200H) vào A. Thanh ghi A lúc này sẽ chứa giá trị 48H là mã ASCII của ký tự "H". Ký tự này được cất vào thanh ghi R1.

Quá trình diễn ra tương tự như vậy, sau khi hoàn tất chương trình ta có (R0) = 44H, (R1) = 48H, (R2) = 43H, (R3) = 4EH là mã ASCII của các ký tự "D", "H", "C" và "N".

- *Ví dụ 14:* Giả sử chữ "TP.HCM" được lưu trong ROM nội tại vùng nhớ có địa chỉ bắt đầu là 200H. Hãy viết chương trình để chuyển các byte dữ liệu này vào các ô nhớ RAM nội bắt đầu từ địa chỉ 40H.

Giải

Phương pháp sử dụng bộ đếm dữ liệu:

```
        ORG     0000H           ;Địa chỉ lưu chương trình trong ROM.
        MOV     DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
        MOV     R0, #40H        ;Nạp địa chỉ bắt đầu chứa trong RAM.
        MOV     R1, #6          ;Nạp giá trị bộ đếm (số lượng ký tự).
LOOP:
        CLR     A               ;Xoá ACC, A = 0
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) trỏ đến đưa vào A.
        MOV     @R0, A          ;Cất vào ô nhớ RAM do R0 trỏ đến.
        INC     DPTR            ;Tăng con trỏ dữ liệu.
        INC     R0              ;Tăng địa chỉ vùng RAM.
        DJNZ    R1, LOOP        ;Lặp lại cho đến khi bộ đếm  = 0.
        SJMP    $               ;Dừng chương trình.
        ORG     200H            ;Địa chỉ lưu chương trình trong ROM.
MYDATA:
        DB      "TP.HCM"        ;Khai báo dữ liệu.
        END                     ;Kết thúc chương trình.
```

Phương pháp sử dụng ký tự NULL để kết thúc chuỗi:

```
        ORG     0000H           ;Địa chỉ lưu chương trình trong ROM.
        MOV     DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
        MOV     R0, #40H        ;Nạp địa chỉ bắt đầu chứa trong RAM.
LOOP:
        CLR     A               ;Xoá ACC, A = 0
        MOVC    A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                                ;(A+DPTR) trỏ đến đưa vào A.
        JZ      EXIT            ;Thoát ra nếu có ký tự NULL.
        MOV     @R0, A          ;Cất vào ô nhớ RAM do R0 trỏ đến.
        INC     DPTR            ;Tăng con trỏ dữ liệu.
        INC     R0              ;Tăng địa chỉ vùng RAM.
        SJMP    LOOP            ;Lặp lại.
EXIT:
        SJMP    $               ;Dừng chương trình.
        ORG     200H            ;Địa chỉ lưu chương trình trong ROM.
MYDATA:
        DB      "TP.HCM", 0     ;Khai báo dữ liệu, có ký tự NULL.
        END                     ;Kết thúc chương trình.
```

- *Ví dụ 15:* Viết chương trình để lấy giá trị x từ P1 và liên tục gửi giá trị x^2 đến P2.

Giải

```
        ORG     0000H           ;Địa chỉ lưu chương trình trong ROM.
        MOV     DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
        MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.
LOOP:
        MOV     A, P1           ;Lấy số liệu x từ P1.
        MOVC    A, @A+DPTR      ;Lấy giá trị x^2 từ vùng dữ liệu.
        MOV     P2, A           ;Xuất giá trị x^2 ra P2.
        SJMP    LOOP            ;Lặp lại.
        ORG     200H            ;Địa chỉ lưu chương trình trong ROM.
MYDATA:                         ;Vùng dữ liệu tương ứng từ 0^2 - 9^2.
        DB      0,1,4,9,16,25,36,49,64,81
        END
```

- *Ví dụ 16:* Viết đoạn lệnh tính tổng các giá trị của các ô nhớ RAM nội có địa chỉ 40H – 44H. Kết quả byte thấp cất vào thanh ghi A và byte cao cất vào thanh ghi B.

Giải

```
        MOV     R0, #40H        ;Nạp địa chỉ bắt đầu chứa trong RAM.
        MOV     R1, #5          ;Nạp giá trị bộ đếm (số lượng ô nhớ).
        CLR     A               ;Xoá ACC, A = 0.
        MOV     B, A            ;Xoá thanh ghi B, B = 0
LOOP:
        ADD     A, @R0          ;Cộng nội dung ô nhớ do R0 trỏ đến vào A.
        JNC     NO_CY           ;Nhảy nếu không có nhớ, CY = 0.
        INC     B               ;Tăng thanh ghi B nếu có nhớ, CY = 1.
NO_CY:
        INC     R0              ;Tăng con trỏ đến ô nhớ kế tiếp.
        DJNZ    R1, LOOP        ;Lặp lại cho đến khi bộ đếm = 0.
```

- *Ví dụ 17:* Viết đoạn lệnh tính tổng hai số 16 bit là 3CE7H và 3B8DH. Cất kết quả vào R7 (byte cao) và R6 (byte thấp).

Giải

```
        CLR     C               ;Xoá cờ nhớ, CY = 0.
        MOV     A, #0E7H        ;Nạp byte thấp 1 vào A, A = E7H.
        ADD     A, #8DH         ;Cộng byte thấp 2 vào A, A = 74H, CY = 1.
        MOV     R6, A           ;Lưu byte thấp của tổng vào R6.
        MOV     A, #3CH         ;Nạp byte cao 1 vào A, A = 3CH.
        ADDC    A, #3BH         ;Cộng có nhớ byte cao 2 vào A, A = 78H.
        MOV     R7, A           ;Lưu byte cao của tổng vào R7.
```

- *Ví dụ 18:* Viết đoạn lệnh tính tổng của 5 dữ liệu BCD được lưu trong RAM nội tại địa chỉ bắt đầu từ 40H như sau: (40H) = 71H, (41H) = 11H, (42H) = 65H, (43H) = 59H, (44H) = 37H. Kết quả được lưu vào thanh ghi R7 (byte cao) và thanh ghi A (byte thấp) dưới dạng số BCD.

Giải

```
        MOV     R0, #40H        ;Nạp địa chỉ bắt đầu chứa trong RAM.
        MOV     R1, #5          ;Nạp giá trị bộ đếm (số lượng ô nhớ).
        CLR     A               ;Xoá ACC, A = 0.
        MOV     R7, A           ;Xoá thanh ghi R7, R7 = 0
LOOP:
        ADD     A, @R0          ;Cộng nội dung ô nhớ do R0 trỏ đến vào A.
        DA      A               ;Hiệu chỉnh thành số BCD.
        JNC     NO_CY           ;Nhảy nếu không có nhớ, CY = 0.
        INC     R7              ;Tăng thanh ghi R7 nếu có nhớ, CY = 1.
NO_CY:
        INC     R0              ;Tăng con trỏ đến ô nhớ kế tiếp.
        DJNZ    R1, LOOP        ;Lặp lại cho đến khi bộ đếm = 0.
```

- *Ví dụ 19:* Viết đoạn lệnh để nhận dữ liệu dưới dạng số HEX trong phạm vi 00H – FFH từ Port 1 và chuyển đổi về dạng thập phân. Lưu các số vào các thanh ghi R7 (LSB), R6 và R5 (MSB).

Giải

```
        MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.
        MOV     A, P1           ;Đọc dữ liệu từ P1.
        MOV     B, #10          ;Nạp B = 10.
        DIV     AB              ;Chia cho 10, tách lấy số cao/số thấp.
        MOV     R7, B           ;Lưu giá trị hàng đơn vị vào R7.
        MOV     B, #10          ;Nạp B = 10.
        DIV     AB              ;Chia cho 10, tách lấy số cao/số thấp.
        MOV     R6, B           ;Lưu giá trị hàng chục vào R6.
        MOV     R5, A           ;Lưu giá trị hàng trăm vào R5.
```

- *Ví dụ 20:* Viết đoạn lệnh đọc và kiểm tra Port 1 xem có chứa giá trị 45H hay không? Nếu (P1) = 45H thì xuất giá trị 99H ra Port 2, ngược lại thì thoát khỏi đoạn lệnh.

Giải

```
        MOV     P2, #00H        ;Xoá P2, P2 = 0.
        MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.
        MOV     R0, #45H        ;Nạp R0 = 45H, giá trị cần kiểm tra.
        MOV     A, P1           ;Đọc dữ liệu từ P1.
        XRL     A, R0           ;Kiểm tra dữ liệu có bằng 45H, nếu bằng thì
        JNZ     EXIT            ;A = 0, không bằng thì A ? 0.
        MOV     P2, #99H        ;Nạp P2 = 99H nếu P1 = 45H (A = 0).
EXIT:
```

- *Ví dụ 21:* Viết đoạn lệnh để lấy bù 2 giá trị chứa trong thanh ghi R0.

Giải

```
        MOV     A, R0           ;Nạp dữ liệu cần lấy bù vào A.
        CPL     A               ;Lấy bù 1.
        ADD     A, #1           ;Cộng thêm 1 để được bù 2.
```

- *Ví dụ 22:* Viết đoạn lệnh kiểm tra thanh ghi A xem có chứa giá trị 99H hay không? Nếu (A) = 99H thì nạp giá trị FFH vào thanh ghi R1, ngược lại thì nạp giá trị 00H vào thanh ghi R1.

Giải

```
        MOV     R1, #0          ;Nạp R0 = 00H.
        CJNE    A, #99H, NEXT   ;So sánh A với giá trị 99H, nếu không
                                ;bằng thì nhảy đến NEXT.
        MOV     R1, #0FFH       ;Nạp R0 = FFH  nếu A = 99H.
NEXT:
```

- *Ví dụ 23:* Giả sử một cảm biến nhiệt được nối tới ngõ vào P1. Hãy viết đoạn lệnh đọc nhiệt độ và so sánh với giá trị 75. Dựa vào kết quả kiểm tra để đặt giá trị nhiệt độ vào các thanh ghi như sau:

Nếu t = 75^{O}C thì (A) = 75.
Nếu t < 75^{O}C thì (R1) = t.
Nếu t > 75^{O}C thì (R2) = t.

Giải

```
        MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.
        MOV     A, P1           ;Đọc dữ liệu (t) từ P1.
        CJNE    A, #75, OVER    ;So sánh dữ liệu (t) với giá trị 75, nếu
                                ;không bằng thì nhảy đến OVER.
        SJMP    EXIT            ;Nếu A = 75 thì thoát chương trình.
OVER:                           ;Trường hợp khi dữ liệu (t) khác 75.
        JNC     NEXT            ;Nhảy đến NEXT nếu dữ liệu (t) > 75, C=0.
        MOV     R1, A           ;Nếu dữ liệu (t) < 75 thì nạp dữ liệu vào
        SJMP    EXIT            ;R1 (R1 = A = t) và thoát chương trình.
NEXT:                           ;Trường hợp khi dữ liệu (t) > 75.
        MOV     R2, A           ;Nạp dữ liệu vào R2 (R2 = A = t).
EXIT:
```

- *Ví dụ 24:* Viết đoạn lệnh liên tục kiểm tra giá trị tại Port 1 nếu giá trị này khác 63H. Nếu (P1) = 63H thì thoát đoạn lệnh không kiểm tra nữa.

Giải

```
        MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.
HERE:
        MOV     A, P1           ;Đọc dữ liệu từ P1.
        CJNE    A, #63H, HERE   ;Duy trì kiểm tra đến khi P1 = 63H.
```

- *Ví dụ 25:* Giả sử các ô nhớ trong RAM nội có địa chỉ từ 40H – 44H chứa nhiệt độ của các ngày được chỉ ra dưới đây. Hãy viết đoạn lệnh kiểm tra xem có giá trị nào bằng 65 không? Nếu có thì đặt địa chỉ của ô nhớ đó vào R4, ngược lại thì đặt R4 = 0.

(40H) = 76, (41H) = 79, (42H) = 69, (43H) = 65, (44H) = 64

Giải

```
        MOV     R4, #0          ;Xoá R4 = 0.
        MOV     R0, #40H        ;Nạp địa chỉ bắt đầu chứa trong RAM.
        MOV     R1, #5          ;Nạp giá trị bộ đếm (số lượng ô nhớ).
        MOV     A, #65          ;Nạp giá trị cần tìm vào A, A = 65.
BACK:
        CJNE    A, @R0, NEXT    ;So sánh dữ liệu do R0 trỏ đến với 65.
        MOV     R4, R0          ;Nếu bằng thì lưu địa chỉ ô nhớ đó vào R4.
        SJMP    EXIT            ;Thoát chương trình.
NEXT:                           ;Trường hợp dữ liệu do R0 trỏ đến khác 65.
        INC     R0              ;Tăng con trỏ đến ô nhớ kế tiếp.
        DJNZ    R2, BACK        ;Lặp lại cho đến khi bộ đếm bằng 0.
EXIT:
```

- *Ví dụ 26:* Viết đoạn lệnh tìm tổng các chữ số 1 trong thanh ghi R0.

Giải

```
        MOV     R1, #0          ;Xoá R1 = 0, lưu số chữ số 1.
        MOV     R2, #8          ;Nạp giá trị bộ đếm (số bit kiểm tra).
        MOV     A, R0           ;Nạp dữ liệu cần kiểm tra từ R0 vào A.
LOOP:
        RLC     A               ;Xoay trái, đưa bit cần kiểm tra vào cờ CY.
        JNC     NEXT            ;Kiểm tra cờ CY.
        INC     R1              ;Tăng giá trị của R1 nếu cờ CY = 1.
NEXT:
        DJNZ    R2, LOOP        ;Lặp lại cho đến khi bộ đếm bằng 0.
```

- *Ví dụ 27:* Viết đoạn lệnh để chuyển đổi số BCD nén chứa trong thanh ghi A thành hai số ASCII và chứa hai số này trong thanh ghi R2 và R3.

Giải

```
        MOV     A, #29H         ;Nạp A = 29H, mã BCD nén của số 29.
        MOV     R2, A           ;Lưu lại số BCD cần chyển đổi trong R2.
        ANL     A, #0FH         ;Xoá (che) 4 bit cao của số BCD (A = 09H).
        ORL     A, #30H         ;Chuyển thành mã ASCII (A = 39H).
        MOV     R3, A           ;Cất vào R3 (R3 = 39H – mã ASCII của 9).
        MOV     A, R2           ;Lấy lại số BCD lúc ban đầu (A = 29H).
        ANL     A, #0F0H        ;Xoá (che) 4 bit thấp của số BCD (A=20H).
        SWAP    A               ;Hoán chyển vị trí 4 bit cao vẫp bit thấp.
        ORL     A, #30H         ;Chuyển thành mã ASCII (A = 32H).
        MOV     R2, A           ;Cất vào R2 (R2 = 32H – mã ASCII của 2).
```

- *Ví dụ 28:* Giả sử bit P2.3 là một đầu vào và biểu diễn nhiệt độ của một lò sấy. Nếu P2.3 = 1 có nghĩa là lò quá nóng. Hãy liên tục kiểm tra bit này, nếu nhiệt độ quá cao thì hãy gửi một xung mức cao đến P1.5 để bật còi báo hiệu.

Giải

```
JNB     P2.3, $     ;Duy trì kiểm tra đầu vào.
SETB    P1.5        ;Khi P2.3 = 1, tạo mức cao cho P1.5 rồi
CLR     P1.5        ;tạo mức thấp cho P1.5, phát một xung.
```

- *Ví dụ 29:* Viết đoạn lệnh kiểm tra xem thanh ghi A có chứa một số chẵn hay không? Nếu có thì chia cho 2, nếu không thì tăng lên 1 để chẵn hoá số đó rồi chia nó cho 2.

Giải

```
        MOV     B, #2       ;Gán số chia, B = 2.
        JNB     ACC.0, OK   ;Kiểm tra nếu là số chẵn thì nhảy đến OK.
        INC     A           ;Chẵn hoá nếu là số lẻ.
OK:
        DIV     AB          ;Chia A cho 2.
```

V. PHẦN BÀI TẬP:

- ***Truy xuất RAM nội (theo 2 cách: định địa chỉ ô nhớ trực tiếp và gián tiếp):***

Bài 1: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 30H của RAM nội.

Bài 2: Viết đoạn lệnh xóa nội dung ô nhớ 31H của RAM nội.

Bài 3: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 32H của RAM nội.

Bài 4: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 33H của RAM nội vào thanh ghi A.

Bài 5: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 34H của RAM nội vào ô nhớ 35H của RAM nội.

Bài 6: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi R4 vào ô nhớ 36H của RAM nội.

Bài 7: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 37H của RAM nội vào thanh ghi R5.

Bài 8: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào thanh ghi R1.

Bài 9: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi R2 vào thanh ghi A.

Bài 10: Viết đoạn lệnh ghi (*chuyển*) giá trị ABH vào thanh ghi A.

Bài 11: Viết đoạn lệnh ghi (*chuyển*) giá trị CDH vào thanh ghi R3.

- ***Truy xuất RAM ngoài:***

Bài 1: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 30H của RAM ngoài (*RAM ngoài có dung lượng ≤ 256 byte*).

Bài 2: Viết đoạn lệnh xóa ô nhớ 31H của RAM ngoài (*RAM ngoài có dung lượng ≤ 256 byte*).

Bài 3: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 32H của RAM ngoài vào thanh ghi A (*RAM ngoài có dung lượng ≤ 256 byte*).

Bài 4: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 33H của RAM ngoài (*RAM ngoài có dung lượng ≤ 256 byte*).

Bài 5: Viết đoạn lệnh chuyển dữ liệu ô nhớ 34H của RAM ngoài vào ô nhớ 35H của RAM ngoài (*RAM ngoài có dung lượng ≤ 256 byte*).

Bài 6: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 1230H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 7: Viết đoạn lệnh xóa ô nhớ 1231H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 8: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 1232H của RAM ngoài vào thanh ghi A (*RAM ngoài có dung lượng > 256 byte*).

Bài 9: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 1233H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 10: Viết đoạn lệnh chuyển dữ liệu ô nhớ 1234H của RAM ngoài vào ô nhớ 1235H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

- ***Truy xuất Port:***

Bài 1: Viết đoạn lệnh xuất (*ghi*) giá trị 0FH ra Port 1.

Bài 2: Viết đoạn lệnh xuất (*ghi*) giá trị F0H ra Port 2.

Bài 3: Viết đoạn lệnh xuất (*ghi*) nội dung thanh ghi A ra Port 1.

Bài 4: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào thanh ghi A.

Bài 5: Viết đoạn lệnh nhập (*đọc*) từ Port 1 và xuất ra Port 2.

Bài 6: Viết đoạn lệnh xuất (*ghi*) nội dung ô nhớ 37H của RAM nội ra Port 3.

Bài 7: Viết đoạn lệnh nhập (*đọc*) từ Port 2 vào ô nhớ 38H của RAM nội.

Bài 8: Viết đoạn lệnh xuất mức 1 (*mức logic cao*) ra chân P1.0

Bài 9: Viết đoạn lệnh xuất mức 0 (*mức logic thấp*) ra chân P1.1

- ***Truy xuất RAM nội, RAM ngoài và Port:***

Bài 1: Viết đoạn lệnh chuyển ô nhớ 40H (*RAM nội*) vào ô nhớ 2000H (*RAM ngoài*).

Bài 2: Viết đoạn lệnh chuyển nội dung ô nhớ 2001H (*RAM ngoài*) vào ô nhớ 41H (*RAM nội*).

Bài 3: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào ô nhớ 42H (RAM nội).

Bài 4: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào ô nhớ 2002H (RAM ngoài).

Bài 5: Viết đoạn lệnh xuất (*ghi*) nội dung ô nhớ 43H (*RAM nội*) ra Port 1.

Bài 6: Viết đoạn lệnh xuất (ghi) nội dung ô nhớ 2003H (*RAM ngoài*) ra Port 1.

- ***Sử dụng vòng lặp:***

Bài 1: Viết đoạn lệnh xóa 20 ô nhớ RAM nội có địa chỉ bắt đầu là 30H.

Bài 2: Viết đoạn lệnh xóa các ô nhớ RAM nội từ địa chỉ 20H đến 7FH.

Bài 3: Viết đoạn lệnh xóa 250 ô nhớ RAM ngoài có địa chỉ bắt đầu là 4000H.

Bài 4: Viết đoạn lệnh xóa 2500 ô nhớ RAM ngoài có địa chỉ bắt đầu là 4000H.

Bài 5: Viết đoạn lệnh xóa các ô nhớ RAM ngoài từ địa chỉ 2000H đến 205FH.

Bài 6: Viết đoạn lệnh xóa các ô nhớ RAM ngoài từ địa chỉ 2000H đến 3FFFH.

Bài 7: Viết đoạn lệnh xóa toàn bộ RAM ngoài có dung lượng 8KB, biết rằng địa chỉ đầu là 2000H.

Bài 8: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM nội có địa chỉ đầu là 30H đến vùng RAM nội có địa chỉ đầu là 40H.

Bài 9: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 100 byte trong RAM ngoài có địa chỉ đầu là 2000H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 10: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 1000 byte trong RAM ngoài có địa chỉ đầu là 2000H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 11: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM nội có địa chỉ đầu là 30H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 12: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM ngoài có địa chỉ đầu là 5F00H đến vùng RAM nội có địa chỉ đầu là 40H.

Bài 13: Cho một chuỗi dữ liệu gồm 20 byte liên tiếp trong RAM nội, bắt đầu từ địa chỉ 20H. Hãy viết đoạn lệnh lần lượt xuất các dữ liệu này ra Port 1.

Bài 14: Giả sử Port 1 được nối đến một thiết bị phát dữ liệu (ví dụ như 8 nút nhấn). Hãy viết đoạn lệnh nhận liên tiếp 10 byte dữ liệu từ thiết bị phát này và ghi vào 10 ô nhớ (RAM nội) liên tiếp bắt đầu từ ô nhớ 50H.

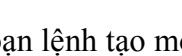
- ***Tạo trễ:***

Bài 1: Viết chương trình con delay 100s, biết rằng f_{OSC} dùng trong hệ thống là:
  a. 6 MHz.
  b. 11,0592 MHz.
  c. 12 MHz.
  d. 24 MHz

Bài 2: Viết chương trình con delay 100ms, biết rằng f_{OSC} dùng trong hệ thống là:
  a. 6 MHz.
  b. 11,0592 MHz.
  c. 12 MHz.
  d. 24 MHz

Bài 3: Viết chương trình con delay 1s, biết rằng f_{OSC} dùng trong hệ thống là:
  a. 6 MHz.
  b. 11,0592 MHz.
  c. 12 MHz.
  d. 24 MHz

- ***Tạo xung:***

Bài 1: Viết đoạn lệnh tạo một xung dương () tại chân P1.0 với độ rộng xung 1ms, biết rằng f_{OSC} =12 MHz.

Bài 2: Viết đoạn lệnh tạo chuỗi xung vuông có f = 100 KHz tại chân P1.1 (f_{OSC} *=12 MHz*).

Bài 3: Viết đoạn lệnh tạo chuỗi xung vuông có f = 100 KHz và có chu kỳ làm việc D = 40% tại chân P1.2 (f_{OSC} *=12 MHz*).

Bài 4: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 KHz tại chân P1.3 (f_{OSC} *=24 MHz*).

Bài 5: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 KHz và có chu kỳ làm việc D = 30% tại chân P1.3 (f_{OSC} *=24 MHz*).

Bài 6: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 Hz tại chân P1.4 (f_{OSC} *=12 MHz*).

Bài 7: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 Hz và có chu kỳ làm việc D = 25% tại chân P1.5 (f_{OSC} *=11,0592 MHz*).

- ***Các phép toán:***

Bài 1: Cho một chuỗi số 8 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H. Hãy viết chương trình con cộng chuỗi số này và ghi kết quả vào ô nhớ 2FH trong RAM nội (*giả sử kết quả nhỏ hơn hoặc bằng 255*).

Bài 2: Cho một chuỗi số 8 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H. Hãy viết chương trình con cộng chuỗi số này và ghi kết quả vào hai ô nhớ 2EH:2FH trong RAM nội (*ô nhớ 2EH chứa byte cao của kết quả và ô nhớ 2FH chứa byte thấp của kết quả*).

Bài 3: Cho một chuỗi số 16 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H theo nguyên tắc ô nhớ có địa chỉ nhỏ hơn chứa byte cao và ô nhớ có địa chỉ lớn hơn chứa byte thấp. (*Ví dụ: byte cao của số 16 bit đầu tiên được cất tại ô nhớ 30H và byte thấp của số 16 bit đầu tiên được cất tại ô nhớ 31H; byte cao của số 16 bit thứ hai được cất tại ô nhớ 32H và byte thấp của số 16 bit thứ hai được cất tại ô nhớ 33H*). Hãy viết chương trình con cộng chuỗi số này và cất kết quả vào hai ô nhớ 2EH:2FH trong RAM nội.

Bài 4: Tương tự như các bài 1, 2, 3 nhưng thực hiện đối với phép trừ.

Bài 5: Viết chương trình con lấy bù 2 số 16 bit chứa trong hai thanh ghi R2:R3.

- ***So sánh:***

Bài 1: Cho hai số 8 bit, số thứ nhất chứa trong ô nhớ 30H, số thứ hai chứa trong ô nhớ 31H. Viết chương trình con so sánh hai số này. Nếu số thứ nhất lớn hơn hoặc bằng số thứ hai thì set cờ F0, nếu ngược lại thì xóa cờ F0.

Bài 2: Cho hai số 16 bit, số thứ nhất chứa trong hai ô nhớ 30H:31H, số thứ hai chứa trong hai ô nhớ 32H:33H. Viết chương trình con so sánh hai số này. Nếu số thứ nhất lớn hơn hoặc bằng số thứ hai thì set cờ F0, nếu ngược lại thì xóa cờ F0.

Bài 3: Cho một chuỗi ký tự dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh xuất các ký tự in hoa có trong chuỗi này ra Port 1. Biết rằng mã ASCII của ký tự in hoa là từ 65H (*chữ A*) đến 90H (*chữ Z*).

Bài 4: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (*có mã ASCII là 0DH*) và ghi cả ký tự này vào RAM.

Bài 5: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (*có mã ASCII là 0DH*) và không ghi ký tự này vào RAM.

Bài 6: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (*có mã ASCII là 0DH*) và không ghi ký tự này vào RAM mà thay bằng ký tự NULL (*có mã ASCII là 00H*).

Bài 7: Cho một chuỗi ký tự dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh đổi các ký tự in hoa có trong chuỗi này thành ký tự thường. Biết rằng mã ASCII của ký tự thường bằng mã ASCII của ký tự in hoa cộng thêm 32H.

Bài 8: Cho một chuỗi ký tự số dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh đổi các ký tự số này thành mã BCD. Biết rằng mã ASCII của các ký tự số là từ 30H (*số 0*) đến 39H (*số 9*).

- ***Sử dụng lệnh nhảy có điều kiện:***

Bài 1: Cho một chuỗi dữ liệu dưới dạng số có dấu trong RAM ngoài, dài 100 byte, bắt đầu từ địa chỉ 0100H. Viết đoạn lệnh lần lượt xuất các dữ liệu trong chuỗi ra Port 1 nếu là số dương (*xem số 0 là dương*) và xuất ra Port 2 nếu là số âm.

Bài 2: Cho một chuỗi dữ liệu dưới dạng số có dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và kết thúc bằng số 0. Viết đoạn lệnh lần lượt xuất các dữ liệu trong chuỗi ra Port 1 nếu là số dương và xuất ra Port 2 nếu là số âm.

Bài 3: Cho một chuỗi dữ liệu dưới dạng số không dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và độ dài chuỗi là nội dung ô nhớ 00FFH. Viết đoạn lệnh đếm số số chẵn (*chia hết cho 2*) có trong chuỗi và cất vào ô nhớ 00FEH.

Bài 4: Cho một chuỗi dữ liệu dưới dạng số không dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và độ dài chuỗi là nội dung ô nhớ 00FFH. Viết đoạn lệnh ghi các số chẵn (*xem số 0 là số chẵn*) có trong chuỗi vào RAM nội bắt đầu từ địa chỉ 30H cho đến khi gặp số lẻ thì dừng.

Bài 5: Viết chương trình con có nhiệm vụ lấy một byte từ một chuỗi data gồm 20 byte cất trong RAM ngoài bắt đầu từ địa chỉ 2000H và xuất ra Port1. Mỗi lần gọi chương trình con chỉ xuất một byte, lần gọi kế thì xuất byte kế tiếp, lần gọi thứ 21 thì lại xuất byte đầu, ...

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

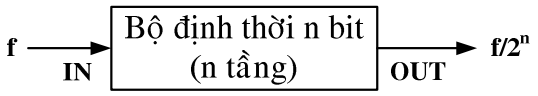
CHƯƠNG 4

HOẠT ĐỘNG CỦA BỘ ĐỊNH THỜI (TIMER)

CHƯƠNG 4
HOẠT ĐỘNG CỦA BỘ ĐỊNH THỜI (*TIMER*)

I. MỞ ĐẦU:

Bộ định thời (TIMER)
- Là chuỗi các FF (mỗi FF là 1 mạch chia 2).
- Ngõ vào: nhận tín hiệu xung clock từ nguồn xung.
- Ngõ ra: truyền tín hiệu xung clock cho FF báo tràn (cờ tràn).



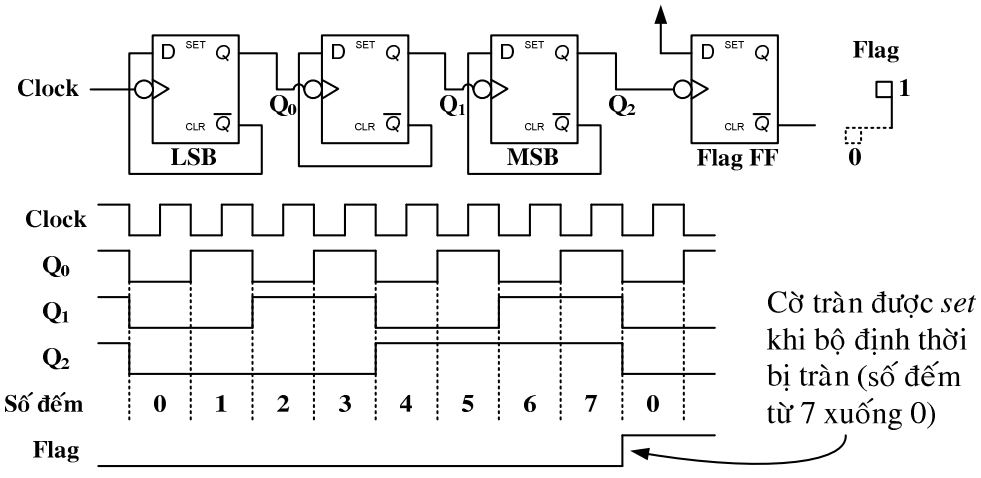
f: tần số xung ngõ vào.
$f/2^n$: tần số xung ngõ ra.
n: số lượng FF trong bộ định thời.

- **Tần số:** tần số xung ngõ ra bằng tần số xung ngõ vào chia cho 2^N.
- **Giá trị:** giá trị nhị phân trong các FF của bộ định thời là số đếm của các xung clock tại ngõ vào từ khi bộ định thời bắt đầu đếm.
- **Tràn:** xảy ra hiện tượng tràn (*cờ tràn = 1*) khi số đếm chuyển từ giá trị lớn nhất xuống giá trị nhỏ nhất của bộ định thời.

Ví dụ: Bộ định thời 16 bit (*chứa 16 FF bên trong*).

- Tần số: $f_{OUT} = \frac{f_{IN}}{2^{16}} = \frac{f_{IN}}{65536}$
- Giá trị: số đếm nằm trong khoảng 0 (*0000H*) → 65535 (*FFFFH*).
- Tràn: cờ tràn bằng 1 khi số đếm từ FFFFH chuyển xuống 0000H.

Hình minh họa đơn giản hoạt động của bộ định thời 3 bit:



Hình 4.1.1: Bộ định thời 3 bit. (a) Sơ đồ logic. (b) Giản đồ thời gian.

Hoạt động của một bộ định thời 3 bit đơn giản được minh họa trong hình trên. Mỗi một tầng là D FF kích khởi cạnh âm hoạt động như một mạch chia 2 do ta nối ngõ ra $\overline{Q}$ với ngõ vào D. Flipflop cờ (*Flag FF*) là một mạch chốt D được set bằng 1 bởi tầng cuối của bộ định thời. Giản đồ thời gian cho

thấy tầng thứ nhất (Q_0) chia 2 tần số xung clock, tầng thứ hai (Q_1) chia 4 tần số xung clock, … Số đếm được ghi ở dạng thập phân và được kiểm tra dễ dàng bằng cách khảo sát trạng thái của 3 flipflop. Ví dụ, số đếm là 4 xuất hiện khi $\mathbf{Q_2 = 1, Q_1 = 0, Q_0 = 0}$. Các flipflop ở trên là các flipflop tác động cạnh âm (*nghĩa là trạng thái của các flipflop sẽ thay đổi theo cạnh âm của xung clock*). Khi số đếm tràn từ 111 xuống 000, ngõ ra Q_2 có cạnh âm làm cho trạng thái của flipflop cờ đổi từ 0 lên 1 *(ngõ vào D của flipflop này luôn luôn ở logic 1)*.

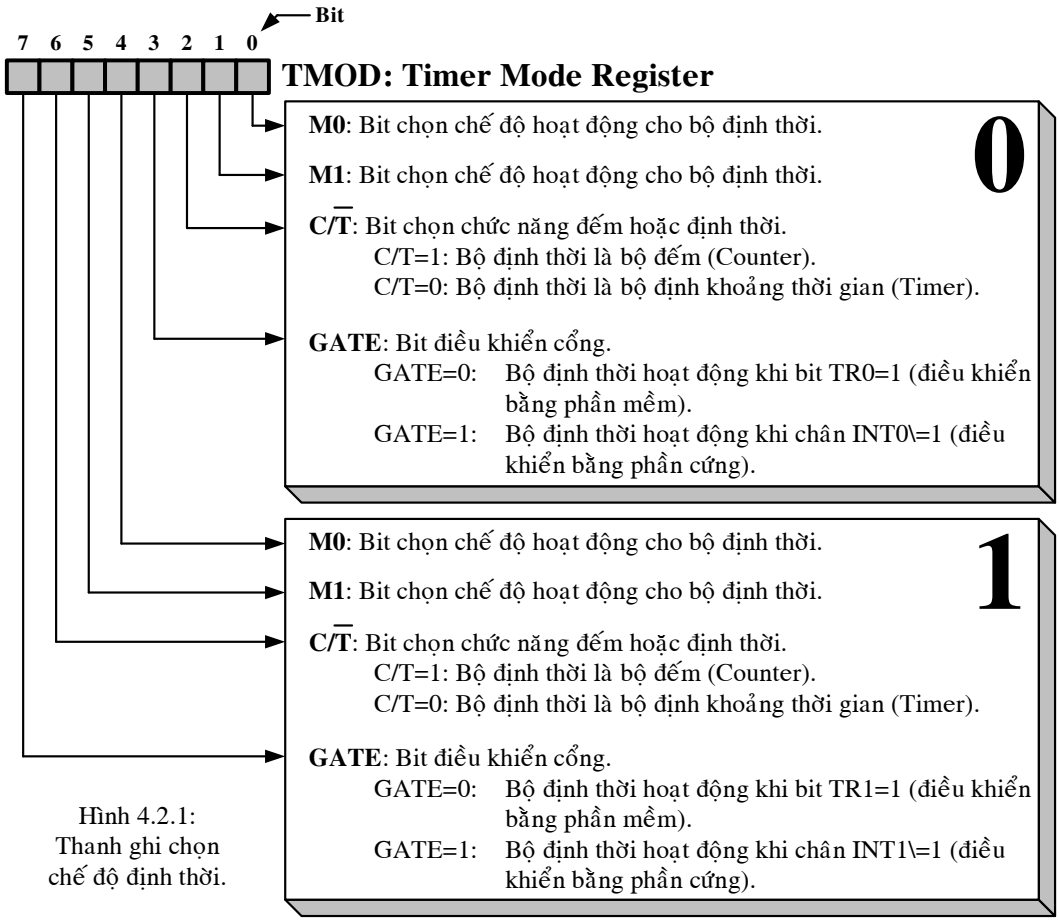
Chip 8051 có 2 bộ định thời
- Dùng để định một khoảng thời gian.
- Dùng để đếm sự kiện (đếm số xung).
- Dùng để tạo tốc độ baud cho port nối tiếp.

- ***Ứng dụng định thời gian (TIMER):*** bộ định thời được lập trình sao cho sẽ tràn sau một khoảng thời gian đã qui định và khi đó cờ tràn của bộ định thời sẽ bằng 1.
- ***Ứng dụng đếm sự kiện (COUNTER):*** để xác định số lần xuất hiện của một kích thích từ bên ngoài tới một chân của chip 8051 (*kích thích là sự chuyển trạng thái từ 1 xuống 0*).
- ***Ứng dụng tạo tốc độ baud cho port nối tiếp:*** xem thêm trong chương ***"Chương 5: Hoạt động port nối tiếp.***".

Thanh ghi chức năng đặc biệt của bộ định thời
- **TCON:** Điều khiển bộ định thời.
- **TMOD:** Chọn chế độ cho bộ định thời.
- **TL0:** Byte thấp của bộ định thời 0.
- **TL1:** Byte thấp của bộ định thời 1.
- **TH0:** Byte cao của bộ định thời 0.
- **TH1:** Byte cao của bộ định thời 1.

II. THANH GHI CHẾ ĐỘ ĐỊNH THỜI (*TMOD*):

- Thanh ghi TMOD (*Timer Mode Register*) chứa các bit dùng để thiết lập chế độ hoạt động cho bộ định thời 0 và bộ định thời 1.
- Thanh ghi TMOD được nạp giá trị một lần tại thời điểm bắt đầu của chương trình để qui định chế độ hoạt động của các bộ định thời.
- Cấu trúc thanh ghi TMOD:



Hình 4.2.1:
Thanh ghi chọn
chế độ định thời.

- Các chế độ hoạt động của bộ định thời:

M1	M0		Chế độ		Mô tả
0	0	→	0	→	Chế độ định thời 13 bit.
0	1	→	1	→	Chế độ định thời 16 bit.
1	0	→	2	→	Chế độ định thời 8 bit tự động nạp lại.
1	1	→	3	→	Chế độ định thời chia xẻ.

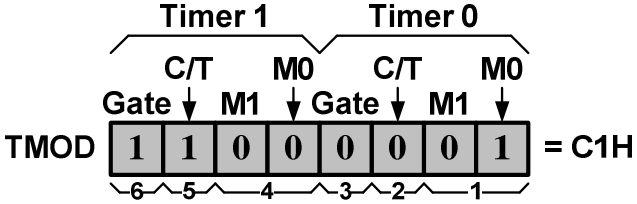
Hình 4.2.2: Các chế độ định thời.

- *Ví dụ 1:* Cho biết giá trị cần nạp cho thanh ghi TMOD để
  - Timer 0: là bộ định thời gian 16 bit, được điều khiển bằng phần mềm (*bit TR0*).
  - Timer 1: là bộ đếm xung 13 bit, được điều khiển bằng phần cứng (*chân* $\overline{INT}1$).

Giải

Phân tích:

(1): Chế độ 16 bit. ⇨ M1 = 0, M0 = 1.
(2): Bộ định thời gian. ⇨ $C/\overline{T}$ = 0.
(3): Điều khiển bằng phần mềm. ⇨ GATE = 0.
(4): Chế độ 13 bit. ⇨ M1 = 0, M0 = 0.
(5): Bộ đếm xung. ⇨ $C/\overline{T}$ = 1.
(6): Điều khiển bằng phần cứng. ⇨ GATE = 1.



Từ đó ta có: **(TMOD) = 11000001B = C1H**.

- *Ví dụ 2:* Cho biết giá trị cần nạp cho thanh ghi TMOD để
  - Timer 0: không sử dụng.
  - Timer 1: là bộ định thời gian 8 bit tự nạp lại, được điều khiển bằng phần mềm (*bit TR1*).

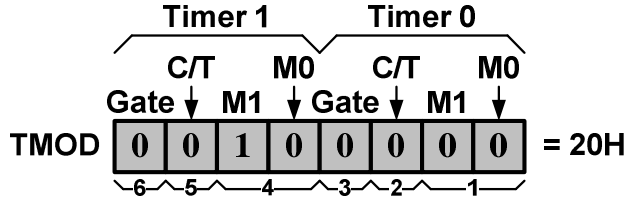
Giải

Phân tích:

(1): Không sử dụng. ⇨ M1 = 0, M0 = 0.
(2): Không sử dụng. ⇨ $C/\overline{T}$ = 0.
(3): Không sử dụng. ⇨ GATE = 0.

Do Timer 0 không sử dụng, nên ta có thiết lập nó ở bất cứ chế độ nào. Thông thường để dễ dàng ta nên cho: GATE=0, $C/\overline{T}$ *= 0, M1 = 0 và M0 = 0.*

(4): Chế độ 8 bit tự động nạp lại. ⇨ M1 = 1, M0 = 0.
(5): Bộ định thời gian. ⇨ $C/\overline{T}$ = 0.
(6): Điều khiển bằng phần mềm. ⇨ GATE = 0.

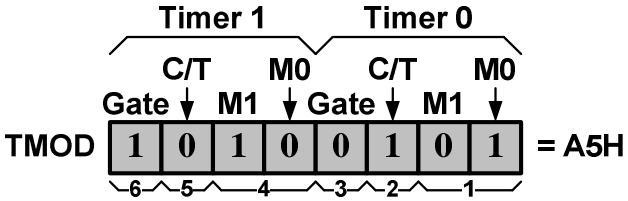


Từ đó ta có: **(TMOD) = 00100000B = 20H**.

- *Ví dụ 3:* Cho biết (TMOD) = A5H. Hãy cho biết chế độ hoạt động của các Timer 0 và Timer 1.

Giải

Ta có: **(TMOD) = A5H = 10100101B**.



Giải thích:

M1 = 1, M0 = 0.	⇨	(4): Chế độ 8 bit tự động nạp lại.
$C/\overline{T}$ = 0.	⇨	(5): Bộ định thời gian.
GATE = 1.	⇨	(6): Điều khiển bằng phần cứng.
M1 = 0, M0 = 1.	⇨	(1): Chế độ 16 bit.
$C/\overline{T}$ = 1.	⇨	(2): Bộ đếm xung.
GATE = 0.	⇨	(3): Điều khiển bằng phần mềm.

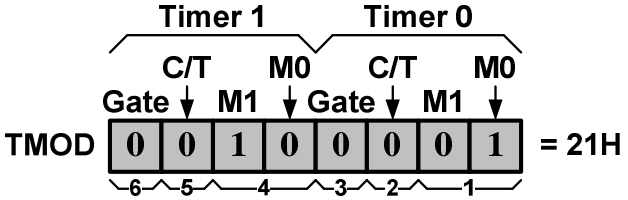
Từ đó ta có:

- Timer 0: là bộ đếm xung 16 bit, được điều khiển bằng phần mềm (*bit TR0*).
- Timer 1: là bộ định thời gian 8 bit tự nạp lại, được điều khiển bằng phần cứng (*chân* $\overline{INT1}$).

- *Ví dụ 4:* Cho biết (TMOD) = 21H. Hãy cho biết chế độ hoạt động của các Timer 0 và Timer 1.

Giải

Ta có: **(TMOD) = 21H = 00100001B**.



Giải thích:

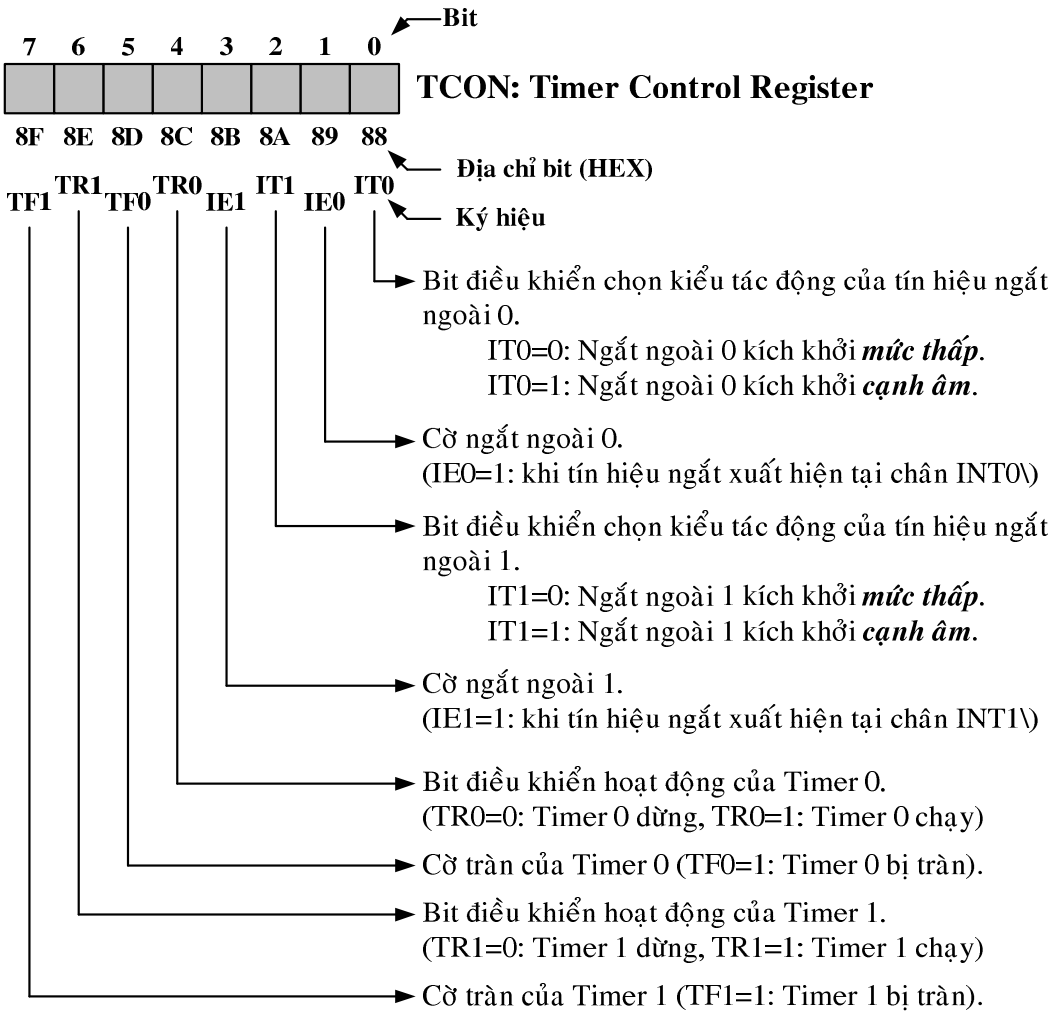
M1 = 1, M0 = 0.	⇨	(4): Chế độ 8 bit tự động nạp lại.
$C/\overline{T}$ = 0.	⇨	(5): Bộ định thời gian.
GATE = 0.	⇨	(6): Điều khiển bằng phần mềm.
M1 = 0, M0 = 1.	⇨	(1): Chế độ 16 bit.
$C/\overline{T}$ = 0.	⇨	(2): Bộ định thời gian.
GATE = 0.	⇨	(3): Điều khiển bằng phần mềm.

Từ đó ta có:

- Timer 0: là bộ định thời gian 16 bit, được điều khiển bằng phần mềm (*bit TR0*).
- Timer 1: là bộ định thời gian 8 bit tự nạp lại, được điều khiển bằng phần mềm (*bit TR1*).

III. THANH GHI ĐIỀU KHIỂN ĐỊNH THỜI (*TCON*):

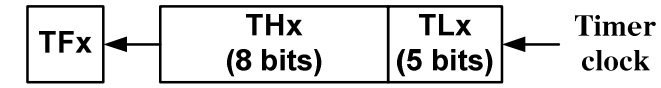
- Thanh ghi TCON (*Timer Control Register*) chứa các bit dùng để điều khiển và báo trạng thái của bộ định thời 0 và bộ định thời 1.
- Cấu trúc thanh ghi TCON:



- *Lưu ý:* Các bit **IT0, IT1, IE0, IE1** không dùng để điều khiển các bộ định thời. Các bit này được dùng để phát hiện và khởi động các ngắt ngoài. Việc thảo luận các bit này sẽ được trình bày trong ***"Chương 6: Hoạt động ngắt."***.

IV. CÁC CHẾ ĐỘ ĐỊNH THỜI VÀ CỜ TRÀN:

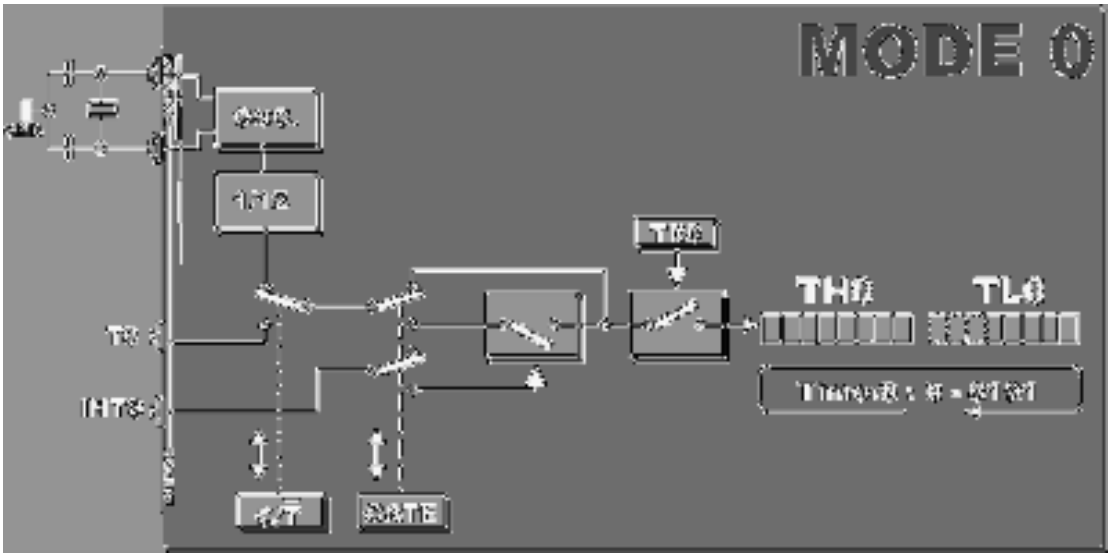
1. Chế độ định thời 13 bit (*Chế độ 0*):



x = 0, 1: Bộ định thời 0, 1.
Timer clock: Xung clock cho bộ định thời.
Overflow flag: Cờ tràn.

Chế độ 0 (*Mode 0*):

- Chế độ định thời 13 bit.
- Sử dụng 8 bit của thanh ghi THx và 5 bit thấp của thanh ghi TLx để tạo ra bộ định thời.
- ***Số đếm:*** 0000H → 1FFFH nghĩa là từ 0 → 8191. ***Thời gian định thời:*** từ $1.T_{Timer} \rightarrow 2^{13}.T_{Timer}$ nghĩa là từ $1.T_{Timer} \rightarrow 8192.T_{Timer}$.
- Thanh ghi THx và TLx chứa giá trị của bộ định thời.
- Khi có xung clock, bộ định thời *bắt đầu đếm lên từ giá trị chứa trong THx/TLx*.
- Xảy ra tràn (*cờ tràn TFx=1*) khi *số đếm chuyển từ 1FFFH sang 0000H* và việc đếm sẽ *tiếp tục đếm lên từ giá trị 0000H*.



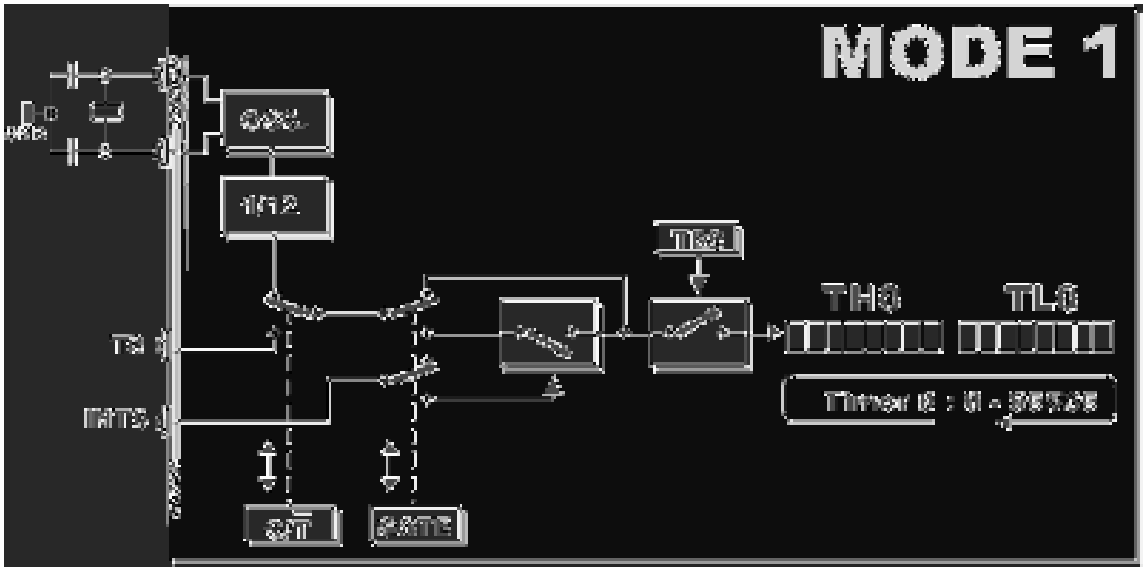
Kiến trúc của Timer 0 ở chế độ 0 (Mode 0).

2. Chế độ định thời 16 bit (*Chế độ 1*):

x = 0, 1: Bộ định thời 0, 1.
Timer clock: Xung clock cho bộ định thời.
Overflow flag: Cờ tràn.

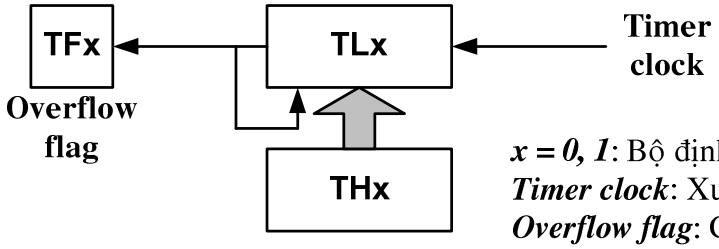
Chế độ 1 (*Mode 1*):

- Chế độ định thời 16 bit.
- Sử dụng thanh ghi THx và TLx để tạo ra bộ định thời.
- ***Số đếm:*** 0000H → FFFFH nghĩa là từ 0 → 65535. ***Thời gian định thời:*** từ $1.T_{Timer} \rightarrow 2^{16}.T_{Timer}$ nghĩa là từ $1.T_{Timer} \rightarrow 65536.T_{Timer}$.
- Thanh ghi THx và TLx chứa giá trị của bộ định thời.
- Khi có xung clock, bộ định thời *bắt đầu đếm lên từ giá trị chứa trong THx/TLx*.
- Xảy ra tràn (*cờ tràn TFx=1*) khi *số đếm chuyển từ FFFFH sang 0000H* và việc đếm sẽ *tiếp tục đếm lên từ giá trị 0000H*.



Kiến trúc của Timer 0 ở chế độ 1 (Mode 1).

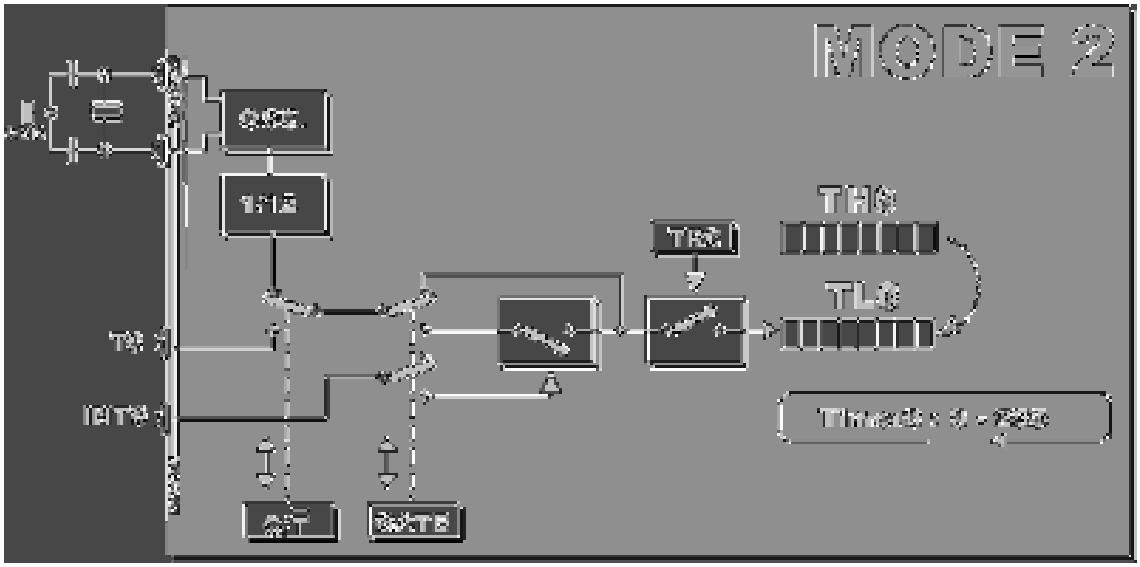
3. Chế độ định thời 8 bit tự nạp lại (*Chế độ 2*):



x = 0, 1: Bộ định thời 0, 1.
Timer clock: Xung clock cho bộ định thời.
Overflow flag: Cờ tràn.

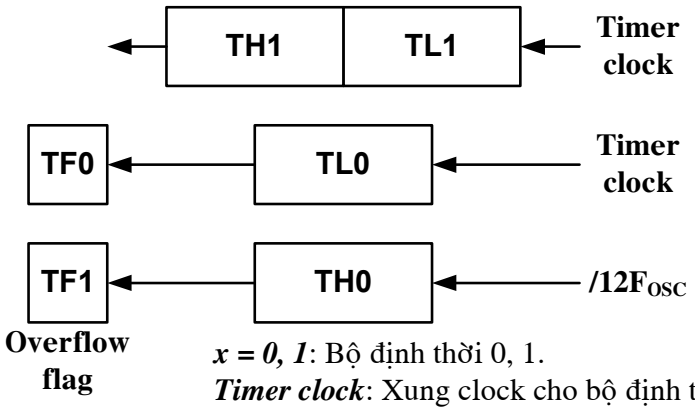
Chế độ 2 (*Mode 2*):

- Chế độ định thời 8 bit tự nạp lại.
- Sử dụng thanh ghi TLx để tạo ra bộ định thời.
- ***Số đếm:*** 00H → FFH nghĩa là từ 0 → 255. ***Thời gian định thời:*** từ $1.T_{Timer} \rightarrow 2^8.T_{Timer}$ nghĩa là từ $1.T_{Timer} \rightarrow 256.T_{Timer}$.
- Thanh ghi TLx chứa giá trị của bộ định thời và thanh ghi THx chứa giá trị sẽ được dùng để nạp lại cho bộ định thời.
- Khi có xung clock, bộ định thời *bắt đầu đếm lên từ giá trị chứa trong TLx (THx không thay đổi giá trị)*.
- Xảy ra tràn (*cờ tràn TFx=1*) khi *số đếm chuyển từ FFH sang 00H,* đồng thời *giá trị trong THx sẽ được nạp vào TLx* và việc đếm sẽ *tiếp tục đếm lên từ giá trị chứa trong thanh ghi TLx (giá trị này bằng với giá trị của THx)*.



Kiến trúc của Timer 0 ở chế độ 2 (Mode 2).

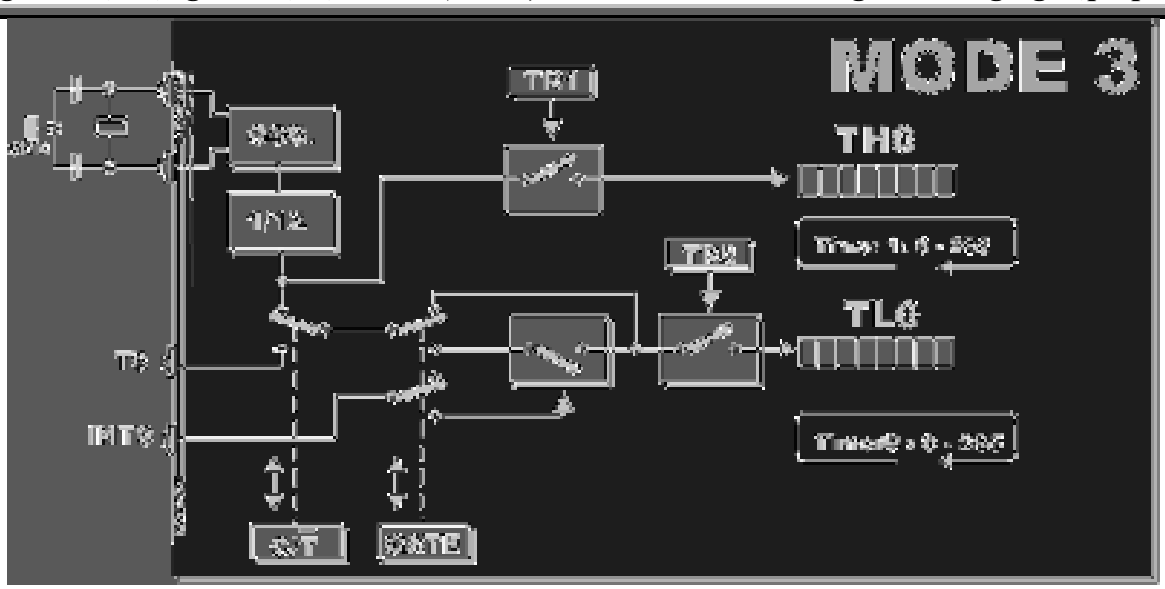
4. Chế độ định thời chia xẻ (*Chế độ 3*):



x = 0, 1: Bộ định thời 0, 1.
Timer clock: Xung clock cho bộ định thời.
Overflow flag: Cờ tràn.

Chế độ 3 (*Mode 3*) là:

- Chế độ định thời chia xẻ.
- Bộ định thời 0 được chia ra:
    - Bộ định thời 8 bit thứ I:
        - Sử dụng thanh ghi **TL0** để tạo ra bộ định thời.
        - ***Số đếm:*** 00H → FFH nghĩa là từ 0 → 255. ***Thời gian định thời:*** từ $1.T_{Timer}$ → $2^8.T_{Timer}$ nghĩa là từ $1.T_{Timer}$ → $256.T_{Timer}$.
        - Thanh ghi TL0 chứa giá trị của bộ định thời.
        - Khi có xung clock, bộ định thời *bắt đầu đếm lên từ giá trị chứa trong TL0.*
        - Xảy ra tràn (***cờ tràn TF0=1***) khi *số đếm chuyển từ FFH sang 00H* và việc đếm sẽ *tiếp tục đếm lên từ giá trị 00H.*
    - Bộ định thời 8 bit thứ II:
        - Sử dụng thanh ghi **TH0** để tạo ra bộ định thời.
        - ***Số đếm:*** 00H → FFH nghĩa là từ 0 → 255. ***Thời gian định thời:*** từ $1.T_{Timer}$ → $2^8.T_{Timer}$ nghĩa là từ $1.T_{Timer}$ → $256.T_{Timer}$.
        - Thanh ghi TH0 chứa giá trị của bộ định thời.
        - Khi có xung clock, bộ định thời *bắt đầu đếm lên từ giá trị chứa trong TH0.*
        - Xảy ra tràn (***cờ tràn TF1=1***) khi *số đếm chuyển từ FFH sang 00H* và việc đếm sẽ *tiếp tục đếm lên từ giá trị 00H.*
- Bộ định thời 1:
    - Là bộ định thời 16 bit.
    - Không hoạt động ở chế độ 3 nhưng có thể hoạt động các chế độ khác (chế độ 0, 1, 2).
    - Không có cờ báo tràn như các bộ định thời khác.



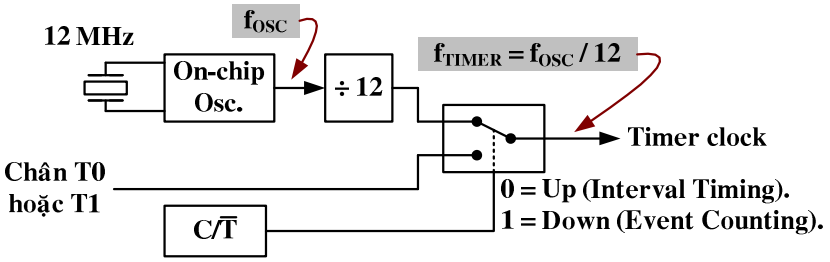
Kiến trúc của Timer 0 ở chế độ 3 (Mode 3).

V. NGUỒN XUNG CLOCK CHO BỘ ĐỊNH THỜI:

Nguồn xung cho bộ định thời được tạo ra từ:

- *Mạch dao động trên chip* → dùng cho tính năng định thời gian.
- *Xung kích thích bên ngoài* → dùng cho tính năng đếm sự kiện.

1. Trường hợp định thời gian:



Nguồn xung clock

Timer clock: Xung clock định thời.
On-chip Osc: Bộ dao động trên chip.
Up (Interval Timing): Vị trí trên (định thời gian).
Down (Event Counting): Vị trí dưới (đếm sự kiện).

Nếu **C/T=0** thì:

- Bộ định thời được dùng để định thời gian (*Timer*).
- Nguồn xung clock định thời được lấy từ mạch dao động trên chip.

Lưu ý:

- Tần số xung clock cung cấp cho bộ định thời bằng 1/12 tần số của mạch dao động trên chip 8051.
- ***Thời gian định thời*** là khoảng thời gian được tính từ lúc bộ định thời bắt đầu đếm lên (*từ giá trị chứa trong các thanh ghi THx/TLx*) cho đến lúc bộ định thời bắt đầu tràn (*thời gian này phụ thuộc vào giá trị ban đầu được nạp cho các thanh ghi THx và TLx*).

- *Ví dụ:* Tìm tần số xung clock và chu kỳ của bộ định thời đối với trường hợp các hệ thống vi điều khiển xây dựng trên chip 8051 với tần số thạch anh như sau: 11,0592 MHz, 12 MHz và 16 MHz.

Giải

Gọi f_{TIMER} là tần số xung clock của bộ định thời, f_{OSC} là tần số xung clock của thạch anh. Theo như trên đã trình bày, ta có:

$$f_{OSC} = 11{,}0592(MHz) \rightarrow f_{TIMER} = \frac{f_{OSC}}{12} = \frac{11{,}0592(MHz)}{12} = 921{,}6(KHz)$$

$$T_{TIMER} = \frac{1}{f_{TIMER}} = \frac{1}{921{,}6(KHz)} = 1{,}085(\mu s)$$

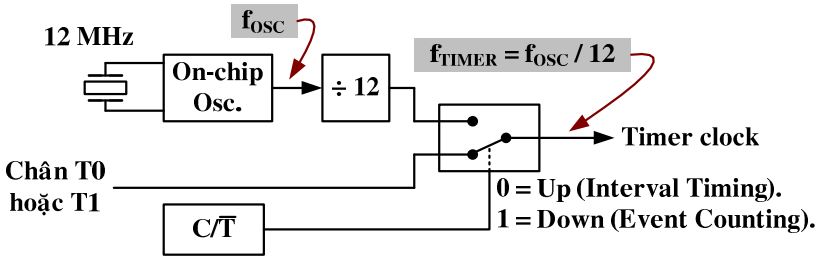
$$f_{OSC} = 12(MHz) \rightarrow f_{TIMER} = \frac{f_{OSC}}{12} = \frac{12(MHz)}{12} = 1(MHz)$$

$$T_{TIMER} = \frac{1}{f_{TIMER}} = \frac{1}{1(MHz)} = 1(\mu s)$$

$$f_{OSC} = 16(MHz) \rightarrow f_{TIMER} = \frac{f_{OSC}}{12} = \frac{16(MHz)}{12} = 1{,}333(MHz)$$

$$T_{TIMER} = \frac{1}{f_{TIMER}} = \frac{1}{1{,}333(MHz)} = 0{,}75(\mu s)$$

2. Trường hợp đếm sự kiện:



Nguồn xung clock

Timer clock: Xung clock định thời.
On-chip Osc: Bộ dao động trên chip.
Up (Interval Timing): Vị trí trên (định thời gian).
Down (Event Counting): Vị trí dưới (đếm sự kiện).

Nếu **C/T=1** thì:

- Bộ định thời được dùng để đếm sự kiện (*Counter*).
- Nguồn xung clock định thời được lấy từ xung kích thích bên ngoài tại hai chân T0 và T1 của chip 8051.

Lưu ý:

- Tần số kích thích ***tối đa cho phép*** tại chân T0 và T1:

$$f_{T0,T1(MAX)} = \frac{f_{TIMER}}{2}$$

f_{TIMER}: tần số xung clock định thời.

$f_{T0,T1(MAX)}$: tần số kích thích tối đa cho phép tại T0 và T1.

- *Ví dụ:* Tính tần số kích thích tối đa cho phép tại chân T0 và T1 đối với trường hợp các hệ thống vi điều khiển xây dựng trên chip 8051 với tần số thạch anh như sau: 11,0592 MHz, 12 MHz và 16 MHz.

Giải

$$f_{OSC} = 11{,}0592(MHz) \rightarrow f_{T0,T1(MAX)} = \frac{f_{TIMER}}{2} = \frac{921{,}6(KHz)}{2} = 460{,}8(KHz)$$

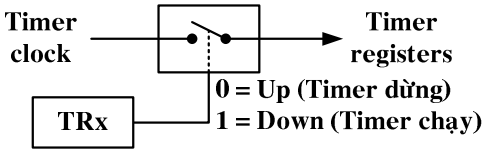
$$f_{OSC} = 12(MHz) \rightarrow f_{T0,T1(MAX)} = \frac{f_{TIMER}}{2} = \frac{1(MHz)}{2} = 500(KHz)$$

$$f_{OSC} = 16(MHz) \rightarrow f_{T0,T1(MAX)} = \frac{f_{TIMER}}{2} = \frac{1{,}333(MHz)}{2} = 666{,}5(KHz)$$

- ***Số lượng sự kiện (số xung)*** mà bộ định thời đếm được sẽ được chứa trong các thanh ghi THx/TLx, giá trị trong các thanh ghi này sẽ tăng theo mỗi xung kích thích bên ngoài tại T0 và T1 của chip 8051.

- ***Một kích thích*** được gọi là một sự kiện (một xung) khi xảy ra ***sự chuyển trạng thái từ 1 xuống 0*** ở chân T0 hoặc T1.

VI. KHỞI ĐỘNG, DỪNG VÀ ĐIỀU KHIỂN CÁC BỘ ĐỊNH THỜI:



Bắt đầu và dừng các bộ định thời.

- ***Cách 1:*** (*thường được dùng để định thời gian*).

Điều kiện sử dụng: bit GATE = 0

(*Phương pháp điều khiển bằng phần mềm*)

⇒ Bộ định thời x chạy khi bit TRx = 1.

⇒ Bộ định thời x dừng khi bit TRx = 0.

Ví dụ: Để khởi động bộ định thời 0 ta dùng lệnh: **SETB TR0**
Để dừng bộ định thời 0 ta dùng lệnh: **CLR TR0**

- ***Cách 2:*** (*thường được dùng để đo độ rộng xung tại chân INTx*).

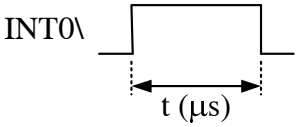
Điều kiện sử dụng: bit GATE = 1 và bit /TRx = 1

(*Phương pháp điều khiển bằng phần cứng*)

⇒ Bộ định thời x chạy khi chân INTx* = 1.

⇒ Bộ định thời x dừng khi chân INTx* = 0.

Ví dụ: Đo độ rộng xung (*tính bằng μs*) tại chân INT0, với f_{OSC} = 12MHz.



Giải

Ta khởi động bộ định thời 0 như sau:

- Chế độ định thời 16 bit (*chế độ 1*).
- Giá trị trong TH0/TL0 là 0000H.
- GATE = 1 và TR0 = 1 (*điều khiển hoạt động của Timer 0 bằng phần cứng, tức điều khiển bằng tín hiệu tại chân INT0*).

Khi INT0\ ⇒ Bộ định thời 0 được mở cổng và nhận xung clock 1 MHz

⇓

Bộ định thời bắt đầu đếm lên từ giá trị chứa trong TH0/TL0

⇓

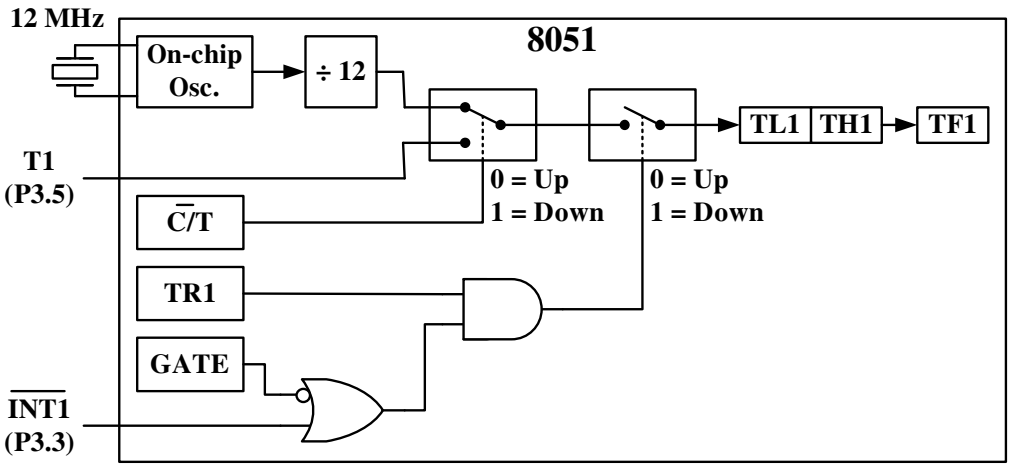
Khi INT0\ ⇒ Bộ định thời 0 bị khoá cổng và không nhận xung clock 1 MHz

⇓

Bộ định thời dừng

⇒ **Độ rộng xung (*tính bằng μs*) = Số đếm chứa trong TH0/TL0.**

Hình minh họa Timer 1 hoạt động ở chế độ 1 (*Timer 16 bit*):



Hoạt động ở chế độ 1 của Timer 1

VII. KHỞI ĐỘNG VÀ TRUY XUẤT THANH GHI ĐỊNH THỜI:

Trước khi các bộ định thời hoạt động cần phải:

- Qui định chế độ của bộ định thời ⇒ ***thanh ghi TMOD***.
- Qui định điểm bắt đầu đếm của bộ định thời (*khoảng thời gian định thời*) ⇒ ***thanh ghi THx/TLx***.

o *Ví dụ 1:* Khởi động bộ định thời 1 hoạt động ở *chế độ 16 bit*, xung clock được lấy từ mạch dao động trên chip (*nghĩa là bộ định được dùng để định thời một khoảng thời gian*), được khởi động bằng bit TR1 (*điều khiển bằng phần mềm*).

Giải

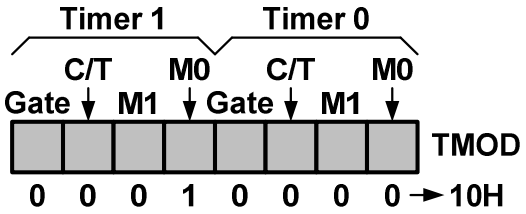
Ta dùng lệnh:

MOV TMOD, #10H

hoặc

MOV TMOD, #00010000B

Giải thích:



- GATE = 0 → điều khiển bằng phần mềm (*bit TR1*).
- C/T = 0 → sử dụng mạch dao động trên chip (*dùng để định một khoảng thời gian*).
- M1 = 0, M1 = 1 → TIMER1 hoạt động ở chế độ 1 (*chế độ định thời 16 bit*).

o *Ví dụ 2:* Dùng bộ định thời 1 ở ví dụ trên để định một khoảng thời gian là 100 µs. Giả sử vi điều khiển sử dụng thạch anh 12 MHz.

Giải

Ta dùng lệnh:

MOV TL1, #9CH
MOV TH1, #0FFH

hoặc

MOV TL1, #LOW(-100)
MOV TH1, #HIGH(-100)

Giải thích:

$$f_{OSC} = 12(MHz) \rightarrow f_{TIMER} = \frac{f_{OSC}}{12} = \frac{12}{12} = 1(MHz)$$

$$\rightarrow T_{TIMER} = \frac{1}{f_{TIMER}} = \frac{1}{1(MHz)} = 1(\mu s)$$

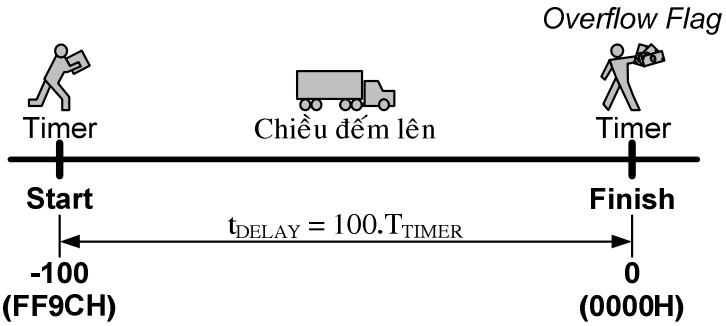
Trong đó: f_{OSC}: tần số thạch anh.
f_{TIMER}: tần số xung clock định thời.
T_{TIMER}: chu kỳ xung clock định thời.

⇨ Vậy cứ mỗi 1µs (tức là sau mỗi chu kỳ của xung clock định thời) thì bộ định thời sẽ tăng giá trị một lần.

Mà ta đã biết: *thời gian định thời là khoảng thời gian được tính từ lúc bộ định thời bắt đầu đếm lên cho đến lúc bộ định thời bắt đầu tràn*.

⇨ Vậy để bộ định thời tràn sẽ tràn sau khoảng thời gian 100µs thì ta phải khởi động bộ định thời tại thời điểm cách điểm tràn (*theo chiều âm – vì bộ định thời chỉ đếm lên*) 100 chu kỳ xung clock định thời.

⇨ Vì điểm tràn có giá trị là 0 cho nên giá trị cần nạp cho các thanh ghi TH1/TL1 là **-100** (*hay **FF9CH***).



Tổng quát, ta có ***công thức tính giá trị cần nạp cho bộ định thời*** để có thời gian định thời như mong muốn là:

$$N = -\frac{f_{OSC}}{12} t_{DELAY}$$

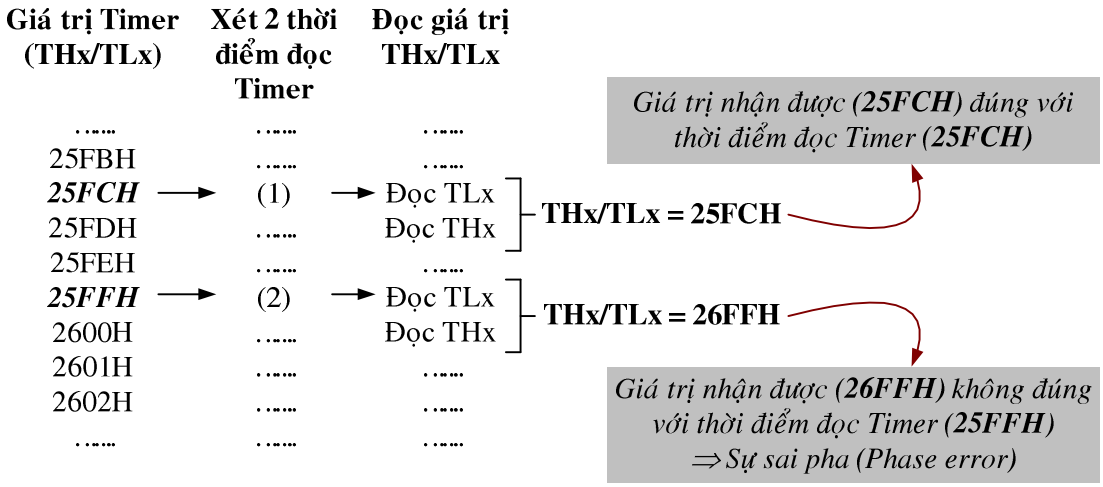
Trong đó: N: giá trị cần nạp cho bộ định thời.
f_{OSC} (MHz): tần số thạch anh.
t_{DELAY} (µs): thời gian cần định thời.

2. Truy xuất giá trị của bộ định thời đang hoạt động:

Trong các ứng dụng thực tế, ta cần phải đọc giá trị (*nội dung*) chứa trong các thanh ghi định thời THx/TLx trong khi bộ định thời vẫn đang hoạt động. Do giá trị của bộ định thời được chứa trong cả hai thanh ghi THx/TLx. Cho nên ta phải đọc hai thanh ghi này bằng hai dòng lệnh liên tiếp nhau (*do không có lệnh nào có thể đọc đồng thời cả hai thanh ghi định thời này*). Một sự sai pha (*phase error*) có thể xuất hiện nếu có sự tràn từ byte thấp chuyển sang byte cao giữa hai lần đọc và do vậy ta không thể đọc đúng được giá trị cần đọc.

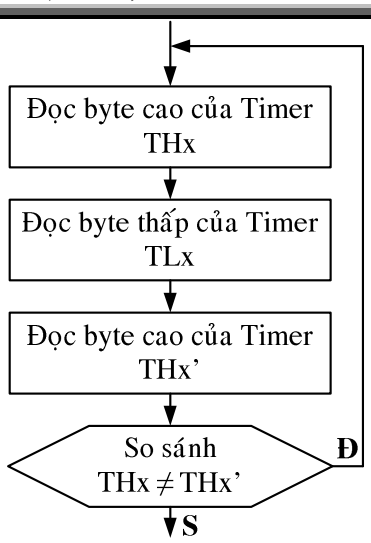
- *Ví dụ:* Minh họa về sự sai pha (*phase error*) có thể xuất hiện nếu có sự tràn từ byte thấp chuyển sang byte cao giữa hai lần đọc giá trị làm cho ta không thể đọc đúng được giá trị cần đọc của THx/TLx trong khi bộ định thời đang hoạt động.

Giải



Minh họa trường hợp đọc giá trị chứa trong các thanh ghi định thời đang hoạt động.

Giải pháp đưa ra là trước tiên ta phải đọc byte cao, kế đến đọc byte thấp và rồi đọc byte thấp lần nữa. Nếu byte cao thay đổi giá trị, ta lặp lại thao tác đọc vừa nêu. Lưu đồ giải thuật dùng để đọc chính xác giá trị (*nội dung*) chứa trong các thanh ghi định thời THx/TLx của *bộ định thời đang hoạt động:*



- *Ví dụ:* Đọc nội dung của các thanh ghi TH1/TL1 trong khi bộ định thời 1 đang hoạt động. Nội dung sau khi đọc của thanh ghi TH1 chứa trong R7, của thanh ghi TL1 chứa trong R6.

```
AGAIN:  MOV A, TH1
        MOV R6, TL1
        CJNE A, TH1, AGAIN
        MOV R7, A
```

VIII. CÁC KHOẢNG THỜI GIAN ĐỊNH THỜI:

Khảo sát trường hợp 8051 dùng thạch anh ***12 MHz***:

- Khoảng thời gian định thời ***ngắn nhất (μs)***: **1**
- Khoảng thời gian định thời ***dài nhất (μs)***:
  - **≈ 10** ⇒ Dùng các lệnh.
  - **≤ 256** ⇒ Dùng bộ định thời 8 bit tự động nạp lại.
  - **≤ 65536** ⇒ Dùng bộ định thời 16 bit.
  - **Không giới hạn** ⇒ Dùng bộ định thời 16 bit + các vòng lặp.

Khảo sát trường hợp tổng quát:

- Khoảng thời gian định thời ***ngắn nhất***: $\mathbf{1.T_{TIMER}}$
- Khoảng thời gian định thời ***dài nhất***:
  - $\mathbf{\approx 10.T_{TIMER}}$ ⇒ Dùng các lệnh.
  - $\mathbf{\leq 256.T_{TIMER}}$ ⇒ Dùng bộ định thời 8 bit tự động nạp lại.
  - $\mathbf{\leq 65536.T_{TIMER}}$ ⇒ Dùng bộ định thời 16 bit.
  - **Không giới hạn** ⇒ Dùng bộ định thời 16 bit + các vòng lặp.

với $\boxed{T_{TIMER} = \frac{12}{f_{OSC}}}$ T_{TIMER}(μs): chu kỳ xung clock định thời.

f_{OSC} (MHz): tần số thạch anh.

IX. CÁC BƯỚC CƠ BẢN KHỞI ĐỘNG TIMER VÀ COUNTER:

1. Các bước cơ bản để khởi động Timer:

- Chọn chế độ hoạt động cho Timer, cho bit GATE=0 và C/T=0:

  MOV TMOD, #...(1)…
- Chọn giá trị thích hợp (*khoảng thời gian định thời*) cho Timer:

  MOV THx, #...(2)…

  MOV TLx, #...(3)…
- Cho Timer chạy:

  SETB TRx
- Kiểm tra cờ báo tràn (*kiểm tra đủ thời gian định thời*):

  JNB TFx, $

  hoặc

  WAIT: JNB TFx, WAIT
- Xóa cờ báo tràn (*chuẩn bị cho lần định thời tiếp theo*):

  CLR TFx
- Dừng Timer (*sau khi đã hoàn tất quá trình định thời*):

  CLR TRx

Lưu ý:

- **x**: Số thứ tự của Timer sử dụng.
- **(1)**: Giá trị này phụ thuộc vào Timer được chọn và chế độ hoạt động của Timer đó.
- **(2), (3)**: Giá trị này phụ thuộc vào khoảng thời gian cần định thời. Cũng cần lưu ý thêm, việc chọn giá trị cho không phải lúc nào ta cũng phải chọn giá trị cho cả hai thanh ghi này mà nó tùy thuộc vào từng chế hoạt động của Timer (*Mode 0: THx/TLx, Mode 1: THx/TLx, Mode 2: THx, Mode 3: THx hoặc TLx*).
- Các giá trị trên phải thoả mãn điều kiện sau:

  - ***Chế độ 8 bit:*** giá trị trong khoảng từ **-255** đến **-1** (*tương ứng từ* $255.T_{TIMER}$ *đến* $1.T_{TIMER}$).

    Ví dụ: **MOV TH1, #(-255)** → định thời $255.T_{TIMER}$
  - ***Chế độ 13 bit:*** giá trị trong khoảng từ **-8191** đến **-1** (*tương ứng từ* $8191.T_{TIMER}$ *đến* $1.T_{TIMER}$).

    Ví dụ: **MOV TL1, #LOW(-8000)** → định thời $8000.T_{TIMER}$

    MOV TH1, #HIGH(-8000)
  - ***Chế độ 16 bit:*** giá trị trong khoảng từ **-65535** đến **-1** (*tương ứng từ* $65535.T_{TIMER}$ *đến* $1.T_{TIMER}$).

    Ví dụ: **MOV TL1, #LOW(-10000)** → định thời $10000.T_{TIMER}$

    MOV TH1, #HIGH(-10000)

Trường hợp đặc biệt nếu giá trị (*N*) nạp vào thanh ghi THx/TLx là ***giá trị 0*** thì thời gian định thời sẽ là lớn nhất cho từng chế độ.

- Chế độ 8 bit: $\mathbf{N = 0 \rightarrow t_{DELAY} = 256.T_{TIMER}}$.
- Chế độ 13 bit: $\mathbf{N = 0 \rightarrow t_{DELAY} = 8192.T_{TIMER}}$.
- Chế độ 16 bit: $\mathbf{N = 0 \rightarrow t_{DELAY} = 65536.T_{TIMER}}$.

2. Các bước cơ bản để khởi động Counter:

- Chọn chế độ hoạt động cho Counter, cho bit GATE=0 và C/T=1:

MOV TMOD, #...(1)…

- Xoá các giá trị chứa trong thanh ghi THx và TLx (*nghĩa là cho số xung ban đầu bằng 0*):

MOV THx, #00H
MOV TLx, #00H

- Cho Counter chạy:

SETB TRx

- Kiểm tra cờ báo tràn (*kiểm tra số đếm bị tràn*) để xử lý trường hợp số đếm bị tràn.
- Xóa cờ báo tràn (*sau khi đã xử lý cho trường hợp số đếm bị tràn*):

CLR TFx

- Dừng Counter (*sau khi đã hoàn tất quá trình đếm xung*):

CLR TRx

- Đọc số xung đếm được trong thanh ghi **THx** và **TLx**.

Lưu ý:

- **x**: Số thứ tự của Counter sử dụng.
- **(1)**: Giá trị này phụ thuộc vào Counter được chọn và chế độ hoạt động của Counter. Giá trị này phải thoả mãn điều kiện sau:
  - ***Chế độ 8 bit:*** số lượng xung tối đa mà Counter đếm được từ **0** đến **255**.
  - ***Chế độ 13 bit:*** số lượng xung tối đa mà Counter đếm được từ **0** đến **8191**.
  - ***Chế độ 16 bit:*** số lượng xung tối đa mà Counter đếm được từ **0** đến **65535**.

Trong quá trình đọc số xung đếm được chứa trong các thanh ghi THx/TLx ta phải chú ý đến trường hợp Counter bị tràn. Vì khi đó giá trị trong thanh ghi THx/TLx (*nơi chứa số xung đếm được*) sẽ trở về **0**. Cho nên nếu ta không có biện pháp xử lý cho trường hợp này thì kết quả là số xung mà ta nhận được sẽ bị sai. Vì thế, nếu ta giả sử ban đầu Counter được khởi động với giá trị là 0 thì cứ mỗi lần Counter bị tràn thì ta phải cộng thêm vào số xung đọc về **256 xung** (*trường hợp 8 bit*) hoặc **8192 xung** (*trường hợp 13 bit*) hoặc **65536 xung** (*trường hợp 16 bit*).

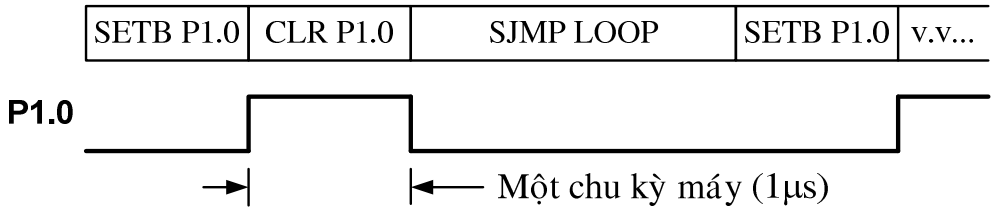
X. CÁC VÍ DỤ MINH HỌA:

1. Ví dụ 1: (*Tạo dạng xung*)

Viết chương trình tạo dạng xung tuần hoàn trên chân P1.0 có tần số cao nhất có thể có. Tần số và chu kỳ nhiệm vụ của dạng xung này là bao nhiêu?

Giải

```
        ORG 8100H
LOOP:   SETB P1.0       ;1 chu kỳ máy
        CLR P1.0        ;1 chu kỳ máy
        SJMP LOOP       ;2 chu kỳ máy
        END
```



- Chu kỳ xung: 4 μs → Tần số xung: 250 KHz.
- Thời gian mức cao: 1 μs.
- Thời gian mức thấp: 3 μs.
- Chu kỳ nhiệm vụ: 25%.

2. Ví dụ 2: (*Tạo thời gian trễ*)

Viết chương trình con tạo thời gian trễ t_{Delay}, sử dụng phương pháp dùng các lệnh (không dùng Timer). Biết rằng tần số thạch anh là 12 MHz.

a. Thời gian trễ t_{Delay} = 100 μs.

b. Thời gian trễ t_{Delay} = 10 ms.

c. Thời gian trễ t_{Delay} = 1 s.

Giải

- ***Phương pháp:***

Phương pháp thực hiện là sử dụng lệnh vòng lặp DJNZ để tạo thời gia trễ t_{Delay} như mong muốn. Ví dụ mẫu:

```
DELAY:                          ;t_Delay = 10 x 20 x 30 x 2.T_Timer
        MOV R0, #10
BBB:
        MOV R1, #20
AAA:
        MOV R2, #30
        DJNZ R2, $              ;lệnh 2.T_Timer
        DJNZ R1, AAA
        DJNZ R0, BBB
        RET
```

Tổng quát, ta có công thức tính thời gian trễ t_{Delay} như sau:

$$t_{Delay} = [Rn] \times [Rm] \times ... \times [Rv] \times 2.\frac{12}{f_{Osc}} \quad (1)$$

Trong đó: **t_{Delay}** (μs): thời gian trễ.

f_{Osc} (MHz): tần số thạch anh.

T_{Timer} (μs): chu kỳ Timer $\left(T_{Timer} = \frac{12}{f_{Osc}} \right)$.

[Rn], [Rm], …, [Rv]: giá trị của các vòng lặp (*số lần lặp lại lệnh DJNZ*).

Lưu ý: Giá trị của các vòng lặp phải thỏa điều kiện **0 ≤ [Rn] ≤ 255**. Đặc biệt nếu chọn [Rn] = 0 thì điều này sẽ tương đương với trường hợp ta chọn [Rn] = 256 (*tương tự cho các [Rm], ..., [Rv] khác*).

- ***Tính toán:*** *Tìm giá trị cần nạp cho các thanh ghi vòng lặp:*

Ta có: $t_{Delay} = [Rn] \times [Rm] \times ... \times [Rv] \times 2.\frac{12}{f_{Osc}}$

Với t_{Delay} = 100 μs, f_{Osc} = 12 MHz thì ta chọn:

[Rn] = 50

$$\Leftrightarrow t_{Delay} = 50 \times 2.\frac{12}{12} = 100\mu s$$

Với t_{Delay} = 10 ms, f_{Osc} = 12 MHz thì ta chọn:

[Rn] = 20 và **[Rm] = 250**

$$\Leftrightarrow t_{Delay} = 20 \times 250 \times 2.\frac{12}{12} = 10000\mu s$$

Với t_{Delay} = 1 s, f_{Osc} = 12 MHz thì ta chọn:

[Rn] = 10, **[Rm] = 200** và **[Ro] = 250**

$$\Leftrightarrow t_{Delay} = 10 \times 200 \times 250 \times 2.\frac{12}{12} = 1000000\mu s$$

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

Với t_{Delay} = 100 μs:

```
DELAY:                  ;t_Delay = 50 x 2.T_Timer
        MOV R0, #50     ;lệnh 1.T_Timer
        DJNZ R0, $      ;lệnh 2.T_Timer
        RET             ;lệnh 2.T_Timer
```

Với t_{Delay} = 10 ms:

```
DELAY:                  ;t_Delay = 20 x 250 x 2.T_Timer
        MOV R0, #20     ;lệnh 1.T_Timer
AAA:
        MOV R1, #250    ;lệnh 1.T_Timer
        DJNZ R1, $      ;lệnh 2.T_Timer
        DJNZ R0, AAA    ;lệnh 2.T_Timer
        RET             ;lệnh 2.T_Timer
```

Với t_{Delay} = 1 s:

```
DELAY:                  ;t_Delay = 10 x 200 x 250 x 2.T_Timer
        MOV R0, #10     ;lệnh 1.T_Timer
BBB:
        MOV R1, #200    ;lệnh 1.T_Timer
AAA:
        MOV R2, #250    ;lệnh 1.T_Timer
        DJNZ R2, $      ;lệnh 2.T_Timer
        DJNZ R1, AAA    ;lệnh 2.T_Timer
        DJNZ R0, BBB    ;lệnh 2.T_Timer
        RET             ;lệnh 2.T_Timer
```

- ***Lưu ý về độ chính xác của t_{Delay}:*** Khi sử dụng phương pháp tạo thời gian trễ như trên (*phương pháp dùng lệnh, không dùng Timer*) thì việc định thời gian thường có một sai số nhất định. Vì ở đây, để đơn giản trong việc tính toán mà ta đã bỏ qua không tính đến thời gian cần thiết để thực hiện từng lệnh trong chương trình, chỉ quan tâm đến thời gian thực hiện của lệnh **DJNZ** ($2T_{Timer}$) và số lần thực hiện của nó. Công thức trình bày ở trên (1) chỉ là công thức tính thời gian t_{Delay} có độ chính xác tương đối, muốn tính thời gian t_{Delay} chính xác thì ta cần phải tính tổng thời gian thực hiện (*nghĩa là tính số chu kỳ máy hay số chu kỳ Timer*) của tất cả các lệnh có trong chương trình. Độ chính xác của chương trình tạo thời gian trễ theo phương pháp tính tương đối này phụ thuộc vào *số lần lặp lại và số vòng lặp*.

Với t_{Delay} = 100 μs, f_{Osc} = 12 MHz thì t_{Delay} chính xác là:

$$t_{Delay(cx)} = 1.T_{Timer} + 2.T_{Timer} \times 50 + 2.T_{Timer} = 103.T_{Timer}$$

Với t_{Delay} = 10 ms, f_{Osc} = 12 MHz thì t_{Delay} chính xác là:

$$t_{Delay(cx)} = 1.T_{Timer} + \left(1.T_{Timer} + 2.T_{Timer} \times 250 + 2.T_{Timer}\right) \times 20 + 2.T_{Timer} = 10065.T_{Timer}$$

Với t_{Delay} = 1 s, f_{Osc} = 12 MHz thì t_{Delay} chính xác là:

$$t_{Delay(cx)} = 1.T_{Timer} + \left(1.T_{Timer} + \left(1.T_{Timer} + 2.T_{Timer} \times 250 + 2.T_{Timer}\right) \times 200 + 2.T_{Timer}\right) \times 10 + 2.T_{Timer}$$
$$= 1006033.T_{Timer}$$

3. Ví dụ 3: (*Tạo thời gian trễ*)

Viết chương trình con tạo thời gian trễ 100 μs dùng Timer 0. Biết rằng tần số thạch anh là 12 MHz.

Giải

- ***Tính toán:*** *Tìm giá trị cần nạp cho bộ định thời và chế độ hoạt động của bộ định thời này:*

Theo đề bài ta có: $t_{Delay} = 100(\mu s)$ và $f_{Osc} = 12(MHz)$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 100.10^{-6} = -100$$

Vậy: **N = -100** hoặc **N = 9CH**.

Ta có: $t_{Delay} = 100(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} \le 256.T_{Timer}$ (hay N nằm trong khoảng từ -255 đến -1) nên ta có thể chọn Timer ở **chế độ 1** (*chế độ 16 bit*) hoặc **chế độ 2** (*chế độ 8 bit tự động nạp lại*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

⇒ Chương trình con hoàn chỉnh khi sử dụng Timer 0 ở chế độ 2:

```
DELAY:
        MOV TMOD, #02H
        MOV TH0, #(-100)    hoặc    MOV TH0, #9CH
        SETB TR0
        JNB TF0, $
        CLR TF0
        CLR TR0
        RET
```

⇒ Chương trình con hoàn chỉnh khi sử dụng Timer 0 ở chế độ 1:

```
DELAY:
        MOV TMOD, #01H
        MOV TH0, #HIGH(-100)    hoặc    MOV TH0, #0FFH
        MOV TL0, #LOW(-100)     hoặc    MOV TL0, #9CH
        SETB TR0
        JNB TF0, $
        CLR TF0
        CLR TR0
        RET
```

- ***Độ chính xác (xét về mặt thời gian) của chương trình:*** Khi sử dụng phương pháp tạo thời gian trễ như trên (*phương pháp dùng Timer*) thì việc định thời gian cũng xuất hiện một sai số. Vì ở đây, để đơn giản trong việc tính toán mà ta đã bỏ qua không tính đến thời gian cần thiết để thực hiện từng lệnh trong chương trình, chỉ quan tâm đến giá trị cần phải nạp cho bộ định thời sao cho Timer định được khoảng thời gian mà ta yêu cầu. Độ chính xác của chương trình định thời khi sử dụng phương pháp này không phụ thuộc vào *giá trị cần nạp cho Timer (N)* mà nó chỉ phụ thuộc vào *số lượng lệnh sử dụng trong chương trình*.

Với dạng thứ nhất (*chế độ 2*) thì t_{Delay} chính xác là: $t_{Delay(cx)} = 11.T_{Timer} + t_{Delay} = 111(\mu s)$

Với dạng thứ hai (*chế độ 1*) thì t_{Delay} chính xác là: $t_{Delay(cx)} = 13.T_{Timer} + t_{Delay} = 113(\mu s)$

4. Ví dụ 4: (*Tạo thời gian trễ*)

Viết chương trình con tạo thời gian trễ 10 ms dùng Timer 1. Biết rằng tần số thạch anh là 12 MHz.

Giải

- ***Tính toán:*** *Tìm giá trị cần nạp cho bộ định thời và chế độ hoạt động của bộ định thời này:*

Theo đề bài ta có: $t_{Delay} = 10(ms)$ và $f_{Osc} = 12(MHz)$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 10.10^{-3} = -10000$$

Vậy: **N = -10000** hoặc **N = D8F0H**.

Ta có: $t_{Delay} = 10(ms) = 10000(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} \le 65536.T_{Timer}$ (hay N nằm trong khoảng từ -65535 đến -1) nên ta chọn Timer ở **chế độ 1** (*chế độ 16 bit*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

⇒ Chương trình con hoàn chỉnh khi sử dụng Timer 1 ở chế độ 1:

```
DELAY:
        MOV TMOD, #10H
        MOV TH1, #HIGH(-10000)  hoặc  MOV TH1, #0D8H
        MOV TL1, #LOW(-10000)   hoặc  MOV TL1, #0F0H
        SETB TR1
        JNB TF1, $
        CLR TF1
        CLR TR1
        RET
```

- ***Độ chính xác (xét về mặt thời gian) của chương trình:***

Với ví dụ trên (*chế độ 1*) thì t_{Delay} chính xác là: $t_{Delay(cx)} = 13.T_{Timer} + t_{Delay} = 10013(\mu s)$

5. Ví dụ 5: (*Tạo thời gian trễ*)

Viết chương trình con tạo thời gian trễ 1s dùng Timer 1. Biết rằng tần số thạch anh là 12 MHz.

Giải

- ***Tính toán:*** *Tìm giá trị cần nạp cho bộ định thời và chế độ hoạt động của bộ định thời này:*

Theo đề bài ta có:

$$t_{Delay} = 1(s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 1 = -1000000$$

Vậy: **N = -1000000** (giá trị quá lớn không thể nạp trực tiếp vào các thanh ghi THx/TLx).

Ta có: $t_{Delay} = 1(s) = 1000000(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} > 65536.T_{Timer}$ nên ta chọn phải Timer ở **chế độ 1** (*chế độ 16 bit*) kết hợp với **các thanh ghi** để tạo vòng lặp.

Gọi: N' là giá trị cần nạp cho các thanh ghi định thời.

[Rn] là giá trị cần nạp cho thanh ghi kết hợp (*vòng lặp*).

$\Rightarrow$ **N = [Rn] x N'**

Ta tự chọn: **N' = -10000 $\rightarrow$ [Rn] = 100**

- *Lưu ý:*
  - **N'** được chọn sao cho phù hợp với qui định chọn giá trị cần nạp cho các thanh ghi định thời ở chế độ 1.
  - **[Rn] ≤ 255**, đặc biệt nếu chọn [Rn] = 0 thì điều này sẽ tương đương với trường hợp ta chọn [Rn] = 256.

$\Rightarrow$ Giá trị cần nạp cho các thanh ghi định thời là **-10000** và giá trị cần nạp cho thanh ghi kết hợp là **100**.

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

$\Rightarrow$ Chương trình con hoàn chỉnh khi sử dụng Timer 1 ở chế độ 1:

```
DELAY:
        PUSH 00H
        MOV TMOD, #10H
        MOV R0, #100
AAA:
        MOV TH1, #HIGH(-10000)  hoặc  MOV TH1, #0D8H
        MOV TL1, #LOW(-10000)   hoặc  MOV TL1, #0F0H
        SETB TR1
        JNB TF1, $
        CLR TF1
        CLR TR1
        DJNZ R0, AAA
        POP 00H
        RET
```

- ***Độ chính xác (xét về mặt thời gian) của chương trình:*** Trường hợp này cũng tương tự như các trường hợp định thời sử dụng Timer đã nêu ở các ví dụ trên. Tuy nhiên ở đây, độ chính xác của chương trình định thời khi sử dụng phương pháp này không phụ thuộc vào *giá trị cần nạp cho Timer (N)* mà nó phụ thuộc vào *số lượng lệnh sử dụng trong chương trình, số lần lặp lại và số vòng lặp* .

Với ví dụ trên (*chế độ 1 + vòng lặp*) thì t_{Delay} chính xác là:

$$t_{Delay(cx)} = 5.T_{Timer} + \left(11.T_{Timer} + t_{Delay(Timer)}\right) \times 100 + 4.T_{Timer}$$
$$= 1001109(\mu s) = 1{,}001109(s)$$

Với $t_{Delay(Timer)}$: thời gian định thời của Timer (*10000 μs*).

6. Ví dụ 6: (*Tạo thời gian trễ*)

Viết chương trình con tạo thời gian trễ 60s dùng Timer 0. Biết rằng tần số thạch anh là 12 MHz.

Giải

- ***Tính toán:*** *Tìm giá trị cần nạp cho bộ định thời và chế độ hoạt động của bộ định thời này:*

Theo đề bài ta có:

$$t_{Delay} = 60(s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 60 = -60000000$$

Vậy: **N = -60000000** (giá trị quá lớn không thể nạp trực tiếp vào các thanh ghi THx/TLx).

Ta có: $t_{Delay} = 60(s) = 60000000(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} > 65536.T_{Timer}$ nên ta chọn phải Timer ở **chế độ 1** (*chế độ 16 bit*) kết hợp với **các thanh ghi** để tạo vòng lặp.

Gọi: N' là giá trị cần nạp cho các thanh ghi định thời.
[Rn] là giá trị cần nạp cho thanh ghi kết hợp (*vòng lặp 1*).
[Rm] là giá trị cần nạp cho thanh ghi kết hợp (*vòng lặp 2*).

$\Rightarrow$ **N = [Rn] x [Rm] x N'**

Ta tự chọn: **N' = -10000 $\rightarrow$ [Rm] = 100 $\rightarrow$ [Rn] = 60**

- *Lưu ý:*
  - **N'** được chọn sao cho phù hợp với qui định chọn giá trị cần nạp cho các thanh ghi định thời ở chế độ 1.
  - **[Rn]**, **[Rm] ≤ 255**, đặc biệt nếu chọn [Rn], [Rm] = 0 thì điều này sẽ tương đương với trường hợp ta chọn [Rn], [Rm] = 256.

$\Rightarrow$ Giá trị cần nạp cho các thanh ghi định thời là **-10000** và giá trị cần nạp cho các thanh ghi kết hợp là **60** (*cho vòng lặp 1*), **100** (*cho vòng lặp 2*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

⇒ Chương trình con hoàn chỉnh khi sử dụng Timer 0 ở chế độ 1:

```
DELAY:
        PUSH 00H
        PUSH 01H
        MOV TMOD, #01H
        MOV R0, #60
AAA:
        MOV R1, #100
BBB:
        MOV TH0, #HIGH(-10000)   hoặc   MOV TH0, #0D8H
        MOV TL0, #LOW(-10000)    hoặc   MOV TL0, #0F0H
        SETB TR0
        JNB TF0, $
        CLR TF0
        CLR TR0
        DJNZ R1, BBB
        DJNZ R0, AAA
        POP 01H
        POP 00H
        RET
```

- ***Độ chính xác (xét về mặt thời gian) của chương trình:*** Trường hợp này cũng tương tự như các trường hợp định thời sử dụng Timer đã nêu ở các ví dụ trên. Tuy nhiên ở đây, độ chính xác của chương trình định thời khi sử dụng phương pháp này không phụ thuộc vào *giá trị cần nạp cho Timer (N)* mà nó phụ thuộc vào *số lượng lệnh sử dụng trong chương trình, số lần lặp lại và số vòng lặp* .

Với ví dụ trên (*chế độ 1 + vòng lặp*) thì t_{Delay} chính xác là:

$$t_{Delay(cx)} = 7.T_{Timer} + \left(1.T_{Timer} + \left(11.T_{Timer} + t_{Delay(Timer)}\right)\times 100 + 2.T_{Timer}\right)\times 60 + 6.T_{Timer}$$
$$= 60066193(\mu s) = 60{,}066193(s)$$

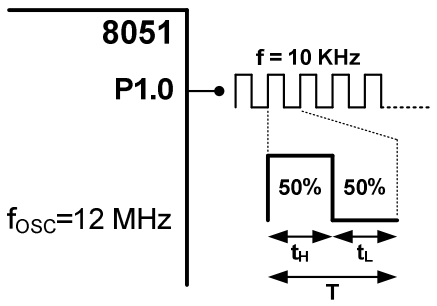
Với $t_{Delay(Timer)}$: thời gian định thời của Timer (*10000 μs*).

7. Ví dụ 7: (*Tạo sóng vuông*)

Viết chương trình tạo sóng vuông có tần số 10 KHz ở ngõ ra P1.0 và có chu kỳ làm việc D=50%. Biết rằng tần số thạch anh là 12 MHz và sử dụng bộ định thời 0.

Giải

- ***Tính toán:***



Theo đề bài, ta có chu kỳ làm việc D=50% cho nên:

$$t_H = 50\% \times T = 50\% \times \frac{1}{f} = 0{,}5 \times \frac{1}{10.10^3} = 5.10^{-5}(s) = 50(\mu s)$$

$\Rightarrow$ **t_H = 50 (µs)** và **t_L = 50 (µs)**.

Vậy ở đây ta phải dùng Timer 0 để tạo thời gian trễ 50(µs) cho thời gian sóng ở mức cao và 50(µs) cho thời gian sóng ở mức thấp.

Theo như trên, ta có:

$$t_{Delay} = 50(\mu s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 50.10^{-6} = -50$$

Vậy: **N = -50** hoặc **N = CEH**.

Ta có: $t_{Delay} = 50(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} \le 256.T_{Timer}$ (*hay N nằm trong khoảng từ -255 đến -1*) nên ta có thể chọn Timer ở **chế độ 1** (*chế độ 16 bit*) hoặc **chế độ 2** (*chế độ 8 bit tự động nạp lại*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

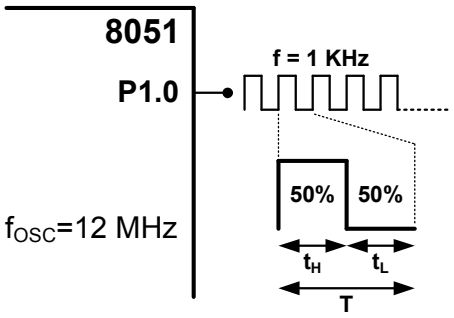
```
MAIN:
        SETB  P1.0
        ACALL DELAY50US
        CLR  P1.0
        ACALL DELAY50US
        SJMP  MAIN
DELAY50US:
        MOV TMOD , #02H
        MOV TH0, #(-50)   hoặc   MOV TH0, #0CEH
        SETB  TR0
        JNB  TF0, $
        CLR TR0
        CLR  TF0
        RET
        END
```

8. Ví dụ 8: (*Tạo sóng vuông*)

Viết chương trình tạo sóng vuông có tần số 1 KHz ở ngõ ra P1.0 và có chu kỳ làm việc D=50%. Biết rằng tần số thạch anh là 12 MHz và sử dụng bộ định thời 0.

Giải

- ***Tính toán:***



Theo đề bài, ta có chu kỳ làm việc D=50% cho nên:

$$t_H = 50\% \times T = 50\% \times \frac{1}{f} = 0{,}5 \times \frac{1}{1.10^3} = 5.10^{-4}(s) = 500(\mu s)$$

$\Rightarrow$ $\mathbf{t_H = 500\ (\mu s)}$ và $\mathbf{t_L = 500\ (\mu s)}$.

Vậy ở đây ta phải dùng Timer 0 để tạo thời gian trễ 500(μs) cho thời gian sóng ở mức cao và 500(μs) cho thời gian sóng ở mức thấp.

Theo như trên, ta có:

$$t_{Delay} = 500(\mu s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 500.10^{-6} = -500$$

Vậy: **N = -500** hoặc **N = FE0CH**.

Ta có: $t_{Delay} = 500(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} \le 65536.T_{Timer}$ (*hay N nằm trong khoảng từ -65535 đến -1*) nên ta chọn Timer ở **chế độ 1** (*chế độ 16 bit*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
MAIN:
        SETB  P1.0
        ACALL DELAY500US
        CLR  P1.0
        ACALL DELAY500US
        SJMP  MAIN
DELAY500US:
        MOV TMOD , #01H
        MOV TH0, #HIGH(-500)  hoặc  MOV TH0, #0FEH
        MOV TL0, #LOW(-500)   hoặc  MOV TL0, #0CH
```

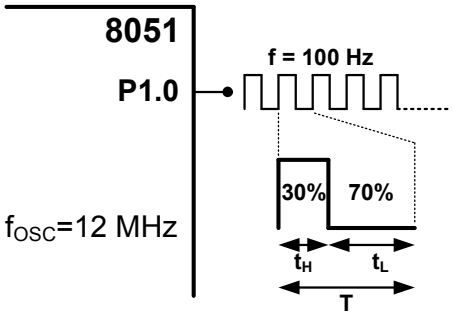
```
SETB  TR0
JNB  TF0, $
CLR TR0
CLR  TF0
RET
END
```

9. Ví dụ 9: (*Tạo sóng vuông*)

Viết chương trình tạo sóng vuông có tần số 100 Hz ở ngõ ra P1.0 và có chu kỳ làm việc D=30%. Biết rằng tần số thạch anh là 12 MHz và sử dụng bộ định thời 0.

Giải

- ***Tính toán:***



Theo đề bài, ta có chu kỳ làm việc D=30% cho nên:

$$t_H = 30\% \times T = 30\% \times \frac{1}{f} = 0{,}3 \times \frac{1}{100} = 3.10^{-3}(s) = 3000(\mu s)$$

$\Rightarrow$ $\mathbf{t_H = 3000\ (\mu s)}$ và $\mathbf{t_L = 7000\ (\mu s)}$.

Vậy ở đây ta phải dùng Timer 0 để tạo thời gian trễ 3000(μs) cho thời gian sóng ở mức cao và 7000(μs) cho thời gian sóng ở mức thấp.

Theo như trên, ta có (*xét trường hợp t_H*):

$$t_{Delay} = 3000(\mu s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 3000.10^{-6} = -3000$$

Vậy: **N = -3000** hoặc **N = F448H**.

Ta có: $t_{Delay} = 3000(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

Vì $t_{Delay} \le 65536.T_{Timer}$ (*hay N nằm trong khoảng từ -65535 đến -1*) nên ta chọn Timer ở **chế độ 1** (*chế độ 16 bit*).

Tương tự như trên, ta có (*xét trường hợp* t_L):

$$t_{Delay} = 7000(\mu s) \text{ và } f_{Osc} = 12(MHz)$$

Giá trị cần nạp cho bộ định thời được tính theo công thức:

$$N = -\frac{f_{Osc}}{12} \times t_{Delay} = -\frac{12.10^6}{12} \times 7000.10^{-6} = -7000$$

Vậy: **N = -7000** hoặc **N = E4A8H**.

Ta có: $t_{Delay} = 7000(\mu s)$

$$T_{Timer} = \frac{1}{f_{Timer}} = \frac{12}{f_{Osc}} = \frac{12}{12.10^6} = 10^{-6}(s) = 1(\mu s)$$

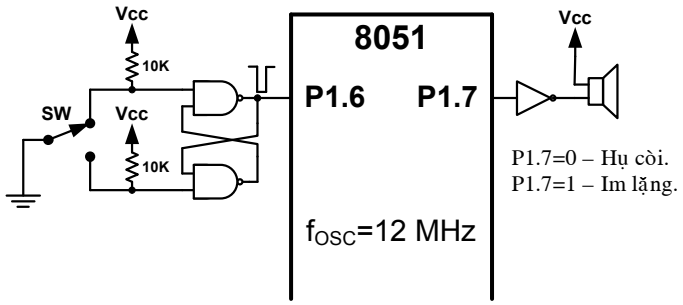
Vì $t_{Delay} \le 65536.T_{Timer}$ (*hay N nằm trong khoảng từ -65535 đến -1*) nên ta chọn Timer ở **chế độ 1** (*chế độ 16 bit*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
MAIN:
        SETB  P1.0
        ACALL DELAY3000US
        CLR  P1.0
        ACALL DELAY7000US
        SJMP  MAIN
DELAY3000US:
        MOV TMOD , #01H
        MOV TH0, #HIGH(-3000)  hoặc  MOV TH0, #0F4H
        MOV TL0, #LOW(-3000)   hoặc  MOV TL0, #48H
        SETB  TR0
        JNB  TF0, $
        CLR TR0
        CLR  TF0
        RET
DELAY7000US:
        MOV TMOD , #01H
        MOV TH0, #HIGH(-7000)  hoặc  MOV TH0, #0E4H
        MOV TL0, #LOW(-7000)   hoặc  MOV TL0, #0A8H
        SETB  TR0
        JNB  TF0, $
        CLR TR0
        CLR  TF0
        RET
        END
```

10. Ví dụ 10: (*Giao tiếp với thiết bị ngoại vi*)

Một còi được nối với chân P1.7 và một chuyển mạch (có chống dội) được nối với chân P1.6 của chip 8051 (xem trong hình vẽ). Viết chương trình điều khiển đọc mức logic cung cấp bởi chuyển mạch (khi chuyển mạch thay đổi từ vị trí trên xuống vị trí dưới thì một xung mức thấp được tạo ra tại chân P1.6) và hụ còi trong thời gian 1sec sau mỗi lần phát hiện sự chuyển trạng thái từ 1 xuống 0 tại chân P1.6.

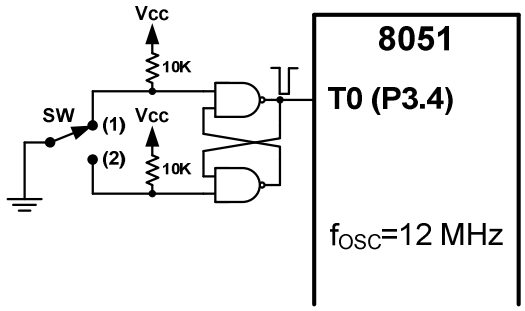


Giải

```
HUNDRED     EQU  100              ;Khai báo biến
COUNT       EQU  -10000
            ORG 0000H
MAIN:
            JNB P1.6, $          ;Chờ logic 1 ở ngõ vào P1.6.
            JB P1.6, $           ;Chờ logic 0 ở ngõ vào P1.6
            SETB P1.7            ;Còi hụ.
            ACALL DELAY          ;Thời gian 1 giây.
            CLR P1.7             ;Tắt còi.
            SJMP MAIN
DELAY:
            PUSH 00H
            MOV TMOD, #10H
            MOV R0, # HUNDRED
AAA:
            MOV TH1, #HIGH(COUNT)
            MOV TL1, #LOW(COUNT)
            SETB TR1
            JNB TF1, $
            CLR TF1
            CLR TR1
            DJNZ R0, AAA
            POP 00H
            RET
            END
```

11. Ví dụ 11: (*Đếm xung*)

Một chuyển mạch (có chống dội) được nối với chân T0 (P3.4) của chip 8051. Viết chương trình điều khiển đếm số lượng xung được tạo ra bởi chuyển mạch (khi chuyển mạch thay đổi từ vị trí (1) sang vị trí (2) thì một xung mức thấp được tạo ra tại chân T0). Số xung đếm được sẽ chứa trong RAM nội (dạng số HEX) tại các ô nhớ có địa chỉ bắt đầu tại 40H. Biết rằng số lượng xung tạo ra được khống chế nằm trong khoảng 0 – 255 xung.



Giải

- ***Tính toán:***

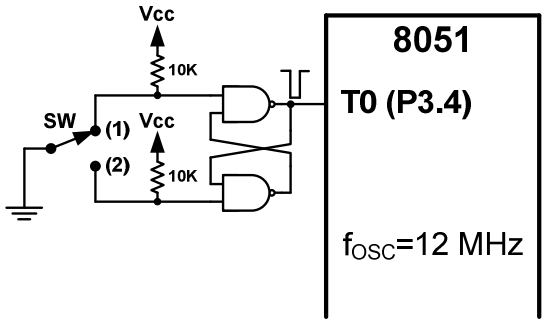
  Theo yêu cầu của đề bài:
  - Viết chương trình đếm xung → Cấu hình cho Timer 0 là một bộ đếm xung (*Counter*).
  - Số xung nằm trong khoảng 0 – 255 xung → Chọn chế độ 8 bit (*chế độ 2*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
MAIN:
        MOV     TMOD, #06H      ;Chế độ Counter 8 bit (chế độ 2).
        MOV     TH0, #00H       ;Giá trị ban đầu của bộ đếm.
        SETB    P3.4            ;Cấu hình P3.4 là ngõ vào.
        SETB    TR0             ;Khởi động bộ đếm.
LOOP:
        MOV     A, TL0          ;Đọc số xung từ bộ đếm.
        MOV     40H, A          ;Cất số xung đếm được vào ô 40H.
        JNB     TF0, LOOP       ;Tiếp tục quá trình đếm xung nếu
                                ;bộ đếm chưa bị tràn.
        CLR     TF0             ;Xoá cờ tràn.
        CLR     TR0             ;Dừng bộ đếm.
        END                     ;Kết thúc chương trình.
```

12. Ví dụ 12: (*Đếm xung*)

Một chuyển mạch (có chống dội) được nối với chân T0 (P3.4) của chip 8051. Viết chương trình điều khiển đếm số lượng xung được tạo ra bởi chuyển mạch (khi chuyển mạch thay đổi từ vị trí (1) sang vị trí (2) thì một xung mức thấp được tạo ra tại chân T0). Số xung đếm được sẽ chứa trong RAM nội (dạng số HEX) tại các ô nhớ có địa chỉ bắt đầu tại 50H. Biết rằng số lượng xung tạo ra được khống chế nằm trong khoảng 0 – 65535 xung.



Giải

- ***Tính toán:***

  Theo yêu cầu của đề bài:
  - Viết chương trình đếm xung → Cấu hình cho Timer 0 là một bộ đếm xung (*Counter*).
  - Số xung nằm trong khoảng 0 – 65535 xung → Chọn chế độ 16 bit (*chế độ 1*).

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
MAIN:
        MOV     TMOD, #05H      ;Chế độ Counter 16 bit (chế độ 1).
        MOV     TH0, #00H       ;Giá trị ban đầu của bộ đếm (cao).
        MOV     TL0, #00H       ;Giá trị ban đầu của bộ đếm (thấp).
        SETB    P3.4            ;Cấu hình P3.4 là ngõ vào.
        SETB    TR0             ;Khởi động bộ đếm.
LOOP:                           ;Đọc giá trị của bộ đếm đang hoạt động.
        MOV     A, TH0          ;Đọc số xung đếm được (phần cao).
        MOV     50H, TL0        ;Cất số xung đếm được (phần thấp).
        CJNE    A, TH0, LOOP    ;Đọc số xung đếm được (phần cao)
                                ;lần nữa để kiểm tra.
        MOV     51H, A          ;Cất số xung đếm được (phần cao).
        JNB     TF0, LOOP       ;Tiếp tục quá trình đếm xung nếu
                                ;bộ đếm chưa bị tràn.
        CLR     TF0             ;Xoá cờ tràn.
        CLR     TR0             ;Dừng bộ đếm.
        END                     ;Kết thúc chương trình.
```

XI. PHẦN BÀI TẬP:

- ***Tạo trễ:***

Bài 1: Viết chương trình con mang tên DELAY500US có nhiệm vụ tạo trễ 0,5ms dùng Timer. (f_{OSC}=*6MHz*).

Bài 2: Viết chương trình con mang tên DELAY10MS có nhiệm vụ tạo trễ 10ms dùng Timer. (f_{OSC}=*12MHz*).

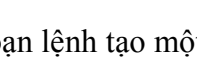
Bài 3: Viết chương trình con mang tên DELAY10S có nhiệm vụ tạo trễ 10s dùng Timer. (f_{OSC}=*12MHz*).

Bài 4: Viết chương trình con mang tên DELAY1S có nhiệm vụ tạo trễ 1s dùng Timer. (f_{OSC}=*11,0592MHz*).

Bài 5: Viết chương trình con delay 100s, biết rằng f_{OSC} dùng trong hệ thống là:
a. 6 MHz.
b. 11,0592 MHz.
c. 12 MHz.
d. 24 MHz

Bài 6: Viết chương trình con delay 100ms, biết rằng f_{OSC} dùng trong hệ thống là:
a. 6 MHz.
b. 11,0592 MHz.
c. 12 MHz.
d. 24 MHz

Bài 7: Viết chương trình con delay 1s, biết rằng f_{OSC} dùng trong hệ thống là:
a. 6 MHz.
b. 11,0592 MHz.
c. 12 MHz.
d. 24 MHz

Bài 8: Viết đoạn lệnh tạo một xung dương () tại chân P1.0 với độ rộng xung 1ms, biết rằng f_{OSC} =12 MHz.

- ***Tạo xung:***

Bài 1: Dùng chương trình con DELAY500US (*Bài 1 phần tạo trễ*) để viết đoạn lệnh tạo sóng vuông f=1KHz tại P1.0.

Bài 2: Dùng chương trình con DELAY10MS (*Bài 2 phần tạo trễ*) để viết đoạn lệnh tạo sóng vuông f=50Hz tại P1.1.

Bài 3: Dùng chương trình con DELAY500US (*Bài 1 phần tạo trễ*) để viết đoạn lệnh tạo sóng vuông f=500Hz (*D=25%*) tại P1.2.

Bài 4: Dùng chương trình con DELAY10MS (*Bài 2 phần tạo trễ*) để viết đoạn lệnh tạo sóng vuông f=20Hz (*D=20%*) tại P1.3.

Bài 5: Viết đoạn lệnh tạo chuỗi xung vuông có f = 100 KHz tại chân P1.1 (f_{OSC} =*12 MHz*).

Bài 6: Viết đoạn lệnh tạo chuỗi xung vuông có f = 100 KHz và có chu kỳ làm việc D = 40% tại chân P1.2 (f_{OSC} =*12 MHz*).

Bài 7: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 KHz tại chân P1.3 (f_{OSC} =*24 MHz*).

Bài 8: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 KHz và có chu kỳ làm việc D = 30% tại chân P1.3 (f_{OSC} *=24 MHz*).

Bài 9: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 Hz tại chân P1.4 (f_{OSC} *=12 MHz*).

Bài 10: Viết đoạn lệnh tạo chuỗi xung vuông có f = 10 Hz và có chu kỳ làm việc D = 25% tại chân P1.5 (f_{OSC} *=11,0592 MHz*).

Bài 11: Viết đoạn lệnh dùng Timer tạo sóng vuông f=500Hz tại P1.4. (f_{OSC}*=12MHz*).

Bài 12: Viết đoạn lệnh dùng Timer tạo sóng vuông f=20KHz tại P1.5. (f_{OSC}*=24MHz*).

Bài 13: Viết đoạn lệnh dùng Timer tạo 2 sóng vuông có cùng f=1KHz tại P1.6 và P1.7. Biết rằng sóng vuông tại P1.7 chậm pha hơn sóng vuông tại P1.6 là 100µs. (f_{OSC}*=12MHz*).

Bài 14: Viết đoạn lệnh dùng Timer điều khiển đèn giao thông tại một giao lộ. Cho biết rằng:

Đèn	***Bit điều khiển***	***Thời gian***
Xanh 1	P1.0	25s
Vàng 1	P1.1	3s
Đỏ 1	P1.2	
Xanh 2	P1.3	33s
Vàng 2	P1.4	3s
Đỏ 2	P1.5	

Đèn sáng khi bit điều khiển bằng 0 (f_{OSC}*=12MHz*).

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

CHƯƠNG 5

HOẠT ĐỘNG CỦA PORT NỐI TIẾP (SERIAL PORT)

CHƯƠNG 5
HOẠT ĐỘNG CỦA PORT NỐI TIẾP (*SERIAL PORT*)

I. MỞ ĐẦU:

Máy tính truyền dữ liệu theo hai phương pháp: truyền dữ liệu song song và truyền dữ liệu nối tiếp.

- ***Truyền song song:*** Sử dụng nhiều dây dẫn để truyền dữ liệu giữa các thiết bị có khoảng cách gần nhau (*khoảng vài mét*). Phương pháp này cho phép truyền dữ liệu với tốc độ cao nhờ sử dụng nhiều dây dẫn để truyền dữ liệu đồng thời nên tại một thời điểm có thể truyền được nhiều bit thông tin nhưng khoảng cách truyền thì có nhiều hạn chế.

- ***Truyền nối tiếp:*** Sử dụng một dây dẫn để truyền dữ liệu (*một dây phát đi và một dây thu về*) giữa các thiết bị có khoảng cách xa nhau (*khoảng vài trăm mét trở lên*). Phương pháp này sẽ truyền dữ liệu với tốc độ chậm hơn (*so với phương pháp truyền song song*) vì chỉ sử dụng một dây dẫn để truyền dữ liệu nên tại một thời điểm chỉ có thể truyền được một bit thông tin nhưng khoảng cách truyền thì không bị hạn chế như ở phương pháp song song.

Chip 8051 có một port nối tiếp (*serial port*) với các tính năng như sau:

Chức năng của port nối tiếp
- *Chuyển đổi từ **dữ liệu song song** thành **dữ liệu nối tiếp**.*
- *Chuyển đổi từ **dữ liệu nối tiếp** thành **dữ liệu song song**.*

Giao tiếp với port nối tiếp
- *Thông qua chân TxD (Transmit Data: chân phát dữ liệu).*
- *Thông qua chân RxD (Receive Data: chân nhận dữ liệu).*

Đặc trưng của port nối tiếp
- *Hoạt động song công (full duplex): có nghĩa là có khả năng thu/phát dữ liệu cùng lúc.*
- *Nhận tiếp một dữ liệu vào trong khi một dữ liệu đã được nhận và lưu giữ trong bộ đệm thu (bộ đệm thu là nơi chứa dữ liệu được nhận vào).*

- ***Lưu ý:*** Ở trường hợp đặc trưng thứ hai thì dữ liệu thứ nhất sẽ không bị mất nếu CPU đọc xong dữ liệu thứ nhất trước khi dữ liệu thứ hai được nhận đầy đủ.

Các thanh ghi chức năng đặc biệt của port nối tiếp:

Các thanh ghi chức năng của port nối tiếp
- SBUF (Serial Buffer Register): thanh ghi đệm của port nối tiếp.
- SCON (Serial Control Register): thanh ghi điều khiển port nối tiếp.

Đại lượng đặc trưng cho tốc độ truyền dữ liệu nhanh hay chậm là ***tốc độ baud (baud rate)*** hay còn gọi là tần số hoạt động của port nối tiếp có thể là giá trị cố định hay thay đổi tùy theo yêu cầu của người lập trình. Khi chế độ tốc độ baud thay đổi được sử dụng, bộ định thời 1 cung cấp xung clock tốc độ baud và ta phải lập trình sao cho phù hợp. Ở phiên bản chip 8031/8052, bộ định thời 2 cũng có thể được lập trình để cung cấp xung clock tốc độ baud.

II. THANH GHI ĐỆM PORT NỐI TIẾP (*SBUF*):

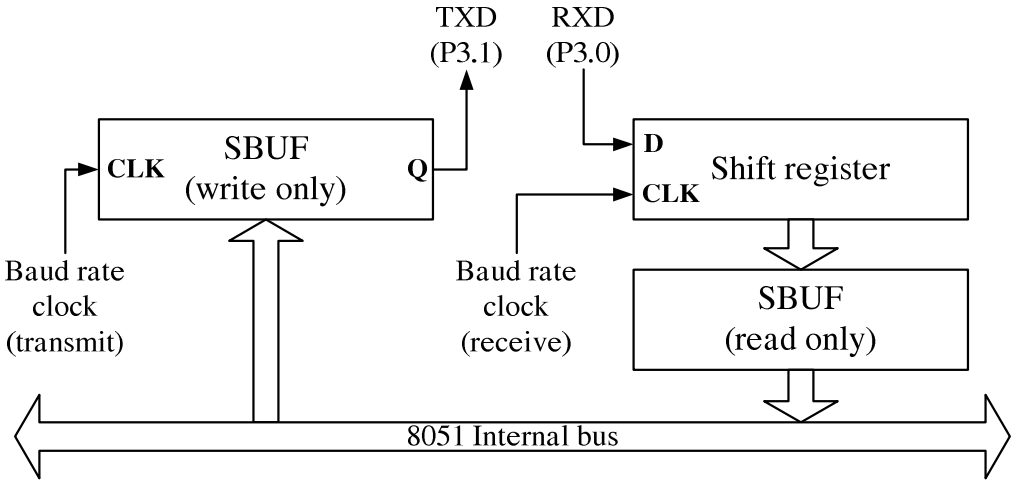
Thanh ghi SBUF (*Serial Buffer Register*): được dùng để lưu giữ dữ liệu cần phát đi và dữ liệu đã nhận được. Việc ghi dữ liệu vào thanh ghi SBUF sẽ nạp dữ liệu để phát đi và việc đọc dữ liệu từ thanh ghi SBUF sẽ truy xuất dữ liệu đã thu được.

Thanh ghi SBUF bao gồm 2 thanh ghi:

⇒ ***Thanh ghi phát (bộ đệm phát)***: dùng để lưu giữ dữ liệu cần phát đi.

⇒ ***Thanh ghi thu (bộ đệm thu)***: dùng để lưu giữ dữ liệu đã nhận được.

Cấu trúc của thanh ghi SBUF:



Write only: *chỉ ghi.*
Read only: *chỉ đọc.*
Shift register: *thanh ghi dịch bit.*
Baud rate clock (transmit): *xung clock tốc độ baud (phát).*
Baud rate clock (receive): *xung clock tốc độ baud (thu).*
8051 internal bus: *bus nội của 8051.*

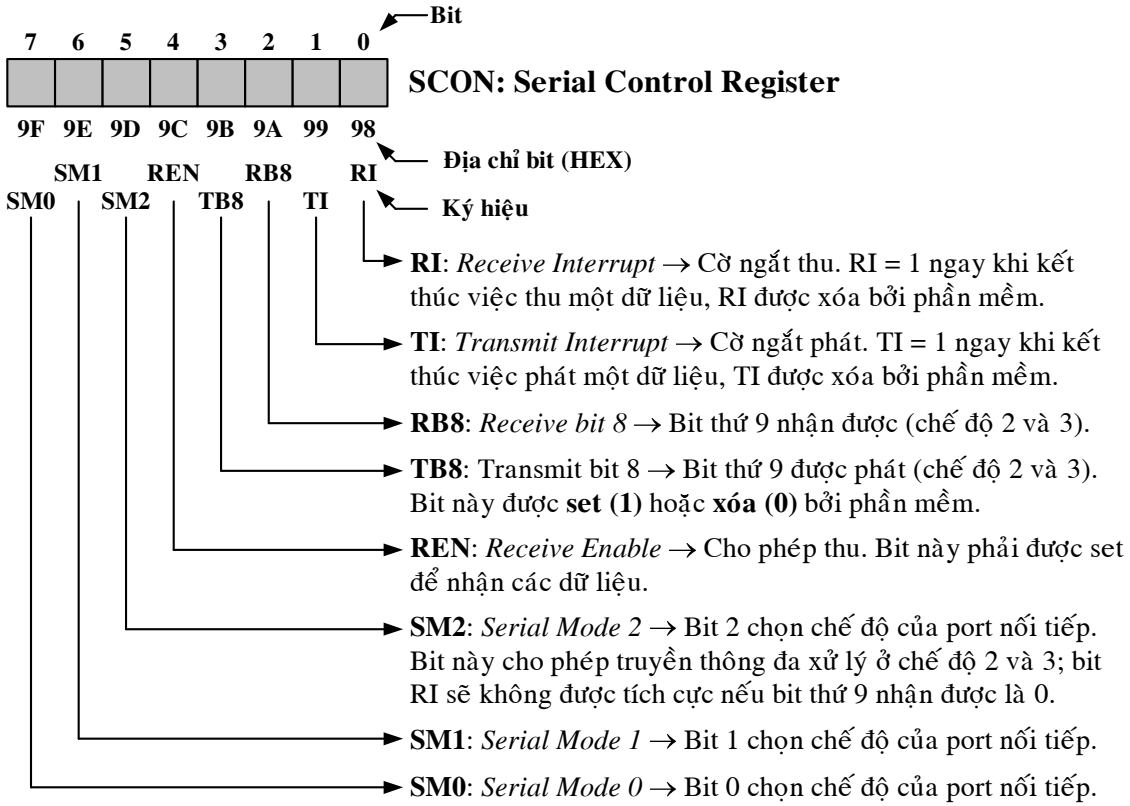
- *Ví dụ:* Các lệnh ghi dữ liệu vào SBUF và đọc dữ liệu từ SBUF.

```
MOV     SBUF, #45H      ;Phát giá trị 45H qua port nối tiếp.
MOV     SBUF, #"D"      ;Phát giá trị 44H qua port nối tiếp.
MOV     SBUF, A         ;Phát nội dung của A qua port nối tiếp.
MOV     A, SBUF         ;Đọc dữ liệu thu được từ port nối tiếp.
```

III. THANH GHI ĐIỀU KHIỂN PORT NỐI TIẾP (*SCON*):

Thanh ghi SCON (*Serial Control Register*): chứa các bit dùng để điều khiển chế độ hoạt động và báo trạng thái của port nối tiếp.

Cấu trúc của thanh ghi SCON:



Các chế độ của port nối tiếp:

SM0	SM1	Chế độ	Mô tả	Tốc độ baud
0	0	0	Thanh ghi dịch.	Cố định (f_{OSC} / 12).
0	1	1	UART 8 bit.	Thay đổi (thiết lập bởi Timer 1).
1	0	2	UART 9 bit.	Cố định (f_{OSC} / 32 hoặc f_{OSC} / 64).
1	1	3	UART 9 bit.	Thay đổi (thiết lập bởi Timer 1).

Trước khi sử dụng port nối tiếp cần phải:

Qui định chế độ của port nối tiếp **PHỤ THUỘC** Thanh ghi SCON

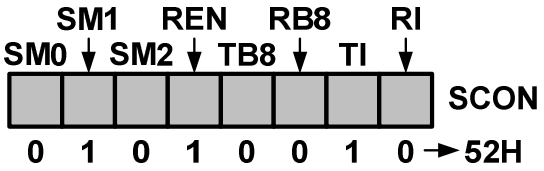
Ví dụ: *Khởi động port nối tiếp ở chế độ 1, cho phép port thu dữ liệu từ chân RxD và sẵn sàng phát dữ liệu từ chân TxD.*

Giải

Ta dùng lệnh:

MOV SCON, #52H

Giải thích:



SM0 = 0, SM1 = 1 → cho phép port hoạt động ở chế độ 1.
REN = 1 → cho phép port nối tiếp được phép thu dữ liệu.
TI = 1 → chuẩn bị port nối tiếp sẵn sàng phát dữ liệu qua chân TxD.
RI = 0 → chuẩn bị port nối tiếp sẵn sàng thu dữ liệu qua chân RxD.

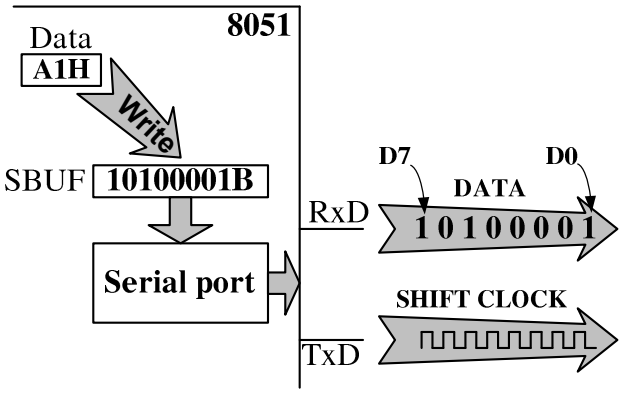
IV. CÁC CHẾ ĐỘ HOẠT ĐỘNG CỦA PORT NỐI TIẾP:

1. Chế độ 0 – Thanh ghi dịch 8 bit:

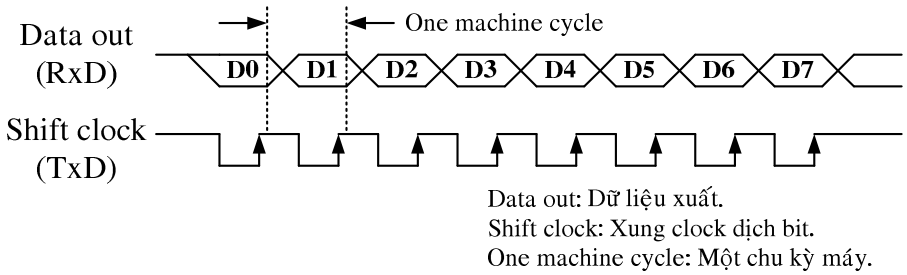
Chế độ 0
- Chế độ thanh ghi dịch 8 bit.
- Các bit SM0 = 0, SM1 = 0.
- ***Chân RxD***: thu (nhận) và phát (truyền) dữ liệu nối tiếp.
- ***Chân TxD***: xuất xung clock để dịch bit.
- Bit D0: luôn luôn được thu và phát đầu tiên.
- Tốc độ baud: không đổi, bằng f_{OSC} / 12.

Quá trình phát dữ liệu:

- ***Quá trình khởi động***: *Ghi dữ liệu cần phát vào SBUF* ⇒ ***Việc phát dữ liệu bắt đầu****: Dữ liệu từ SBUF được dịch ra chân RxD đồng thời với các xung clock dịch bit được gởi ra chân TxD (mỗi bit được truyền đi trên chân RxD trong 1 chu kỳ máy).*



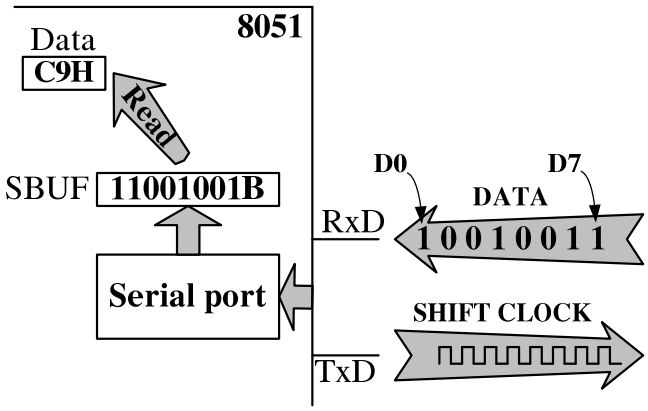
- Giản đồ thời gian phát dữ liệu:



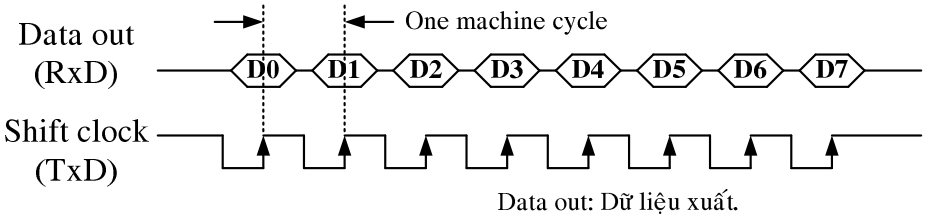
Data out: Dữ liệu xuất.
Shift clock: Xung clock dịch bit.
One machine cycle: Một chu kỳ máy.

Quá trình thu dữ liệu:

- ***Quá trình khởi động***: *Set bit cho phép thu (REN=1)* $\rightarrow$ *Xóa cờ ngắt thu (RI=0)* $\Rightarrow$ ***Việc thu dữ liệu bắt đầu***: *Các xung clock dịch bit được gởi ra chân TxD và dữ liệu từ thiết bị bên ngoài được dịch vào chân RxD bởi các xung clock dịch bit này (việc dịch dữ liệu vào chân RxD xảy ra ở cạnh lên của xung clock dịch bit).*

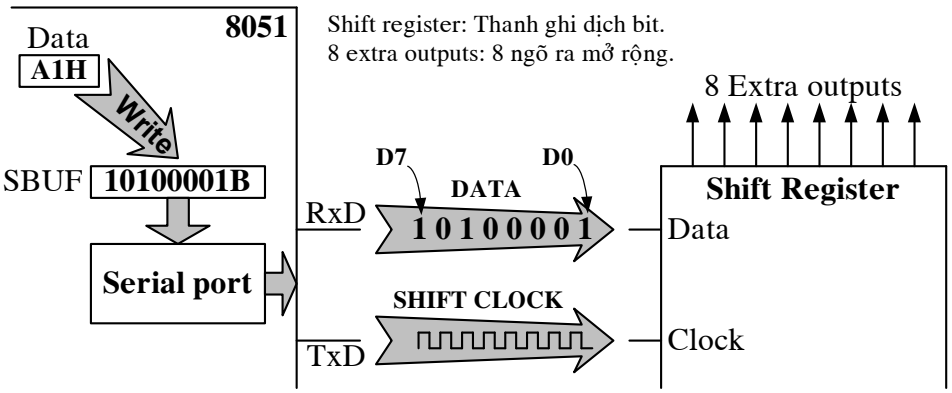


- Giản đồ thời gian thu dữ liệu:



Data out: Dữ liệu xuất.
Shift clock: Xung clock dịch bit.
One machine cycle: Một chu kỳ máy.

Ứng dụng: Một ứng dụng khả thi của chế độ 0 (chế độ thanh ghi dịch bit) là mở rộng thêm các ngõ ra cho chip 8051. Một vi mạch thanh ghi dịch nối tiếp – song song có thể được nối với các chân TxD và RxD của chip 8051 để cung cấp thêm 8 đường xuất (*xem hình vẽ bên dưới*). Các thanh ghi dịch bit khác có thể ghép *cascade* với thanh ghi dịch bit đầu tiên để mở rộng thêm nữa.



Chế độ thanh ghi dịch bit của port nối tiếp.

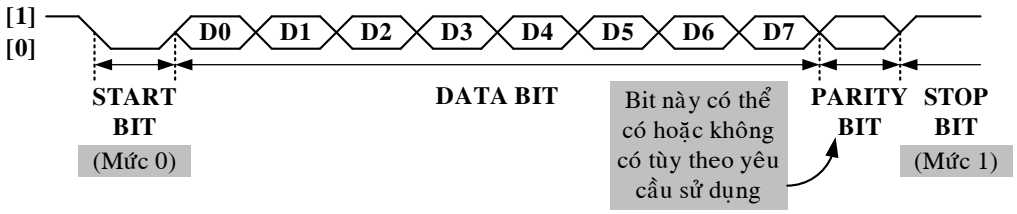
2. Chế độ 1 – UART 8 bit có tốc độ baud thay đổi:

Trong chế độ 1, port nối tiếp của 8051 hoạt động như một bộ thu phát không đồng bộ 8 bit có tốc độ baud thay đổi (*UART - Universal Asynchronous Receiver Transmitter*).

Nhiệm vụ UART:
- Biến đổi dữ liệu phát từ song song thành nối tiếp.
- Biến đổi dữ liệu thu từ nối tiếp thành song song.

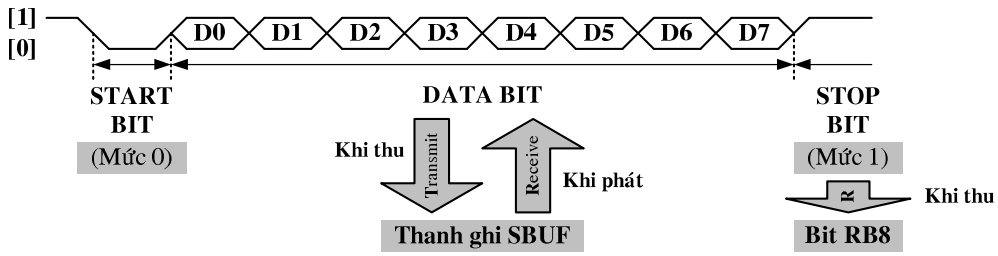
UART là một bộ thu phát dữ liệu nối tiếp với mỗi ký tự dữ liệu được đứng trước bởi một bit START (*logic 0*) và được đứng sau bởi một bit STOP (*logic 1*). Thỉnh thoảng, một bit chẵn lẽ (*Parity bit*) được chèn vào giữa bit dữ liệu sau cùng và bit stop. Hoạt động chủ yếu của UART là biến đổi dữ liệu phát từ song song thành nối tiếp và biến đổi dữ liệu thu từ nối tiếp thành song song.

Hình vẽ khuông dạng dữ liệu khi được sử dụng ở chế độ UART:



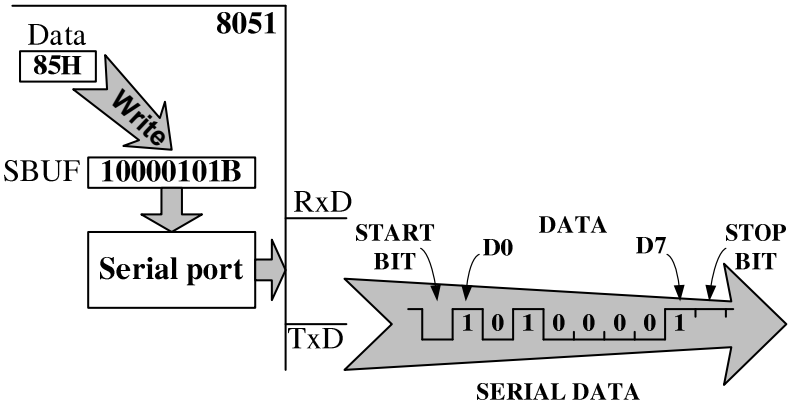
Chế độ 1
- Chế độ UART 8 bit.
- Các bit SM0 = 0, SM1 = 1.
- ***Chân RxD***: thu (nhận) dữ liệu nối tiếp.
- ***Chân TxD***: phát (truyền) dữ liệu nối tiếp.
- Dữ liệu được truyền kèm theo *Start bit* và *Stop bit*.
- Bit D0: luôn luôn được thu và phát đầu tiên.
- Tốc độ baud: thay đổi, được thiết lập bởi Timer 1.

Khuông dạng của một dữ liệu khi sử dụng chế độ UART 8 bit:



Quá trình phát dữ liệu:

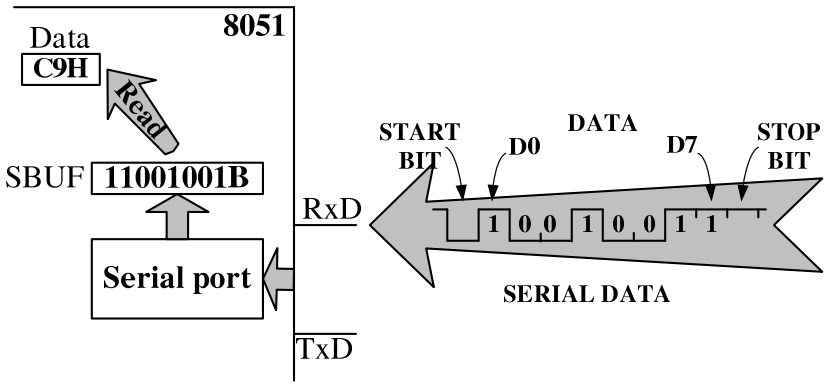
- ***Quá trình khởi động***: *Ghi dữ liệu cần phát vào SBUF* ⇒ ***Việc phát dữ liệu bắt đầu***: *Dữ liệu từ SBUF được dịch ra chân TxD (theo thứ tự:* ***Start bit → 8 bit data (D0 .. D7) → Stop bit****) → cờ TI=1.*



- Tốc độ baud: do người lập trình thiết lập và được qui định bởi tốc độ tràn của Timer 1.
- Thời gian của 1 bit trên đường truyền: bằng nghịch đảo của tốc độ baud (*1 / Baud rate*).
- Cờ ngắt phát TI = 1: khi bit stop được xuất hiện trên chân TxD.

Qúa trình thu dữ liệu:

- ***Quá trình khởi động***: *Một sự chuyển trạng thái từ mức 1 xuống mức 0 tại chân RxD (tức xuất hiện bit Start)* ⇒ ***Việc thu dữ liệu bắt đầu***: *8 bit dữ liệu được dịch vào trong SBUF (theo thứ tự: D0→D1→...→D7) → Stop bit (bit thứ 9) được đưa vào bit RB8 (thuộc thanh ghi SCON) → cờ RI=1.*



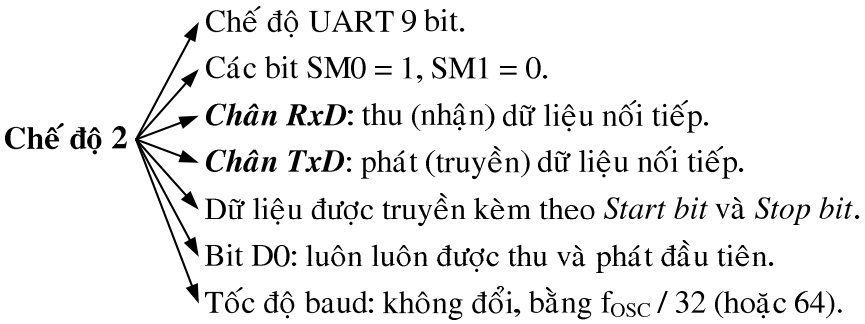
- Tốc độ baud: do người lập trình thiết lập và được qui định bởi tốc độ tràn của Timer 1.
- Hai điều kiện bắt buộc để thực hiện quá trình thu dữ liệu như trên:
  - **RI = 0** → Yêu cầu này có nghĩa là chip 8051 đã đọc xong dữ liệu trước đó và xoá cờ RI..
  - **(*SM2 = 1 và Stop bit = 1*) hoặc SM2 = 0** → chỉ áp dụng trong chế độ truyền thông đa xử lý. Yêu cầu này có nghĩa là không set cờ RI bằng 1 trong chế độ truyền thông đa xử lý khi bit dữ liệu thứ 9 là 0.
- Cờ ngắt thu RI = 1: khi 8 bit dữ liệu đã được nạp vào SBUF.

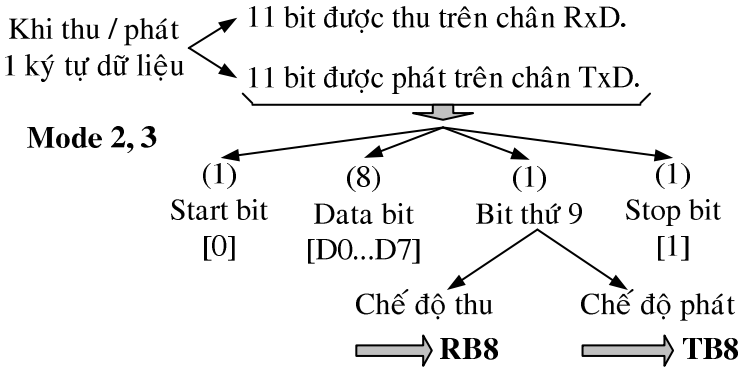
Lưu ý: Trường hợp các tín hiệu nhiễu xuất hiện trên đường truyền (*làm cho đường truyền xuất hiện mức thấp*) dẫn đến làm cho bộ thu nhận dạng sai, cho đó là sự xuất hiện của START bit (*logic 0*) và tiến hành thực hiện quá trình thu dữ liệu, từ đó dẫn đến kết quả nhận vào sẽ không đúng. Để tránh điều này xảy ra thì khi đường truyền có sự chuyển trạng thái từ 1 xuống 0, bộ thu yêu cầu mức 0 này phải

được duy trì trên đường truyền trong một khoảng thời gian xác định. Nếu không đảm bảo được như thế, bộ thu được giả sử rằng đã nhận được nhiễu thay vì nhận được START bit hợp lệ. Lúc đó bộ thu sẽ được thiết lập lại, quay về trạng thái nghỉ và chờ sự chuyển trạng thái từ 1 xuống 0 kế tiếp trên đường truyền.

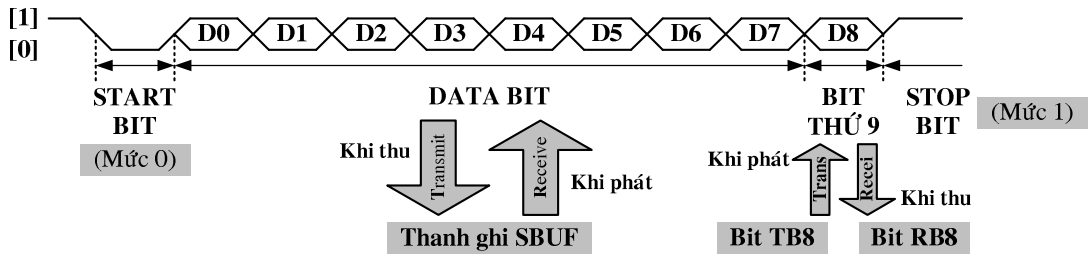
3. Chế độ 2 – UART 9 bit có tốc độ baud cố định:

(Tương tự như UART 8 bit, chỉ khác ở số bit dữ liệu là 9 bit)



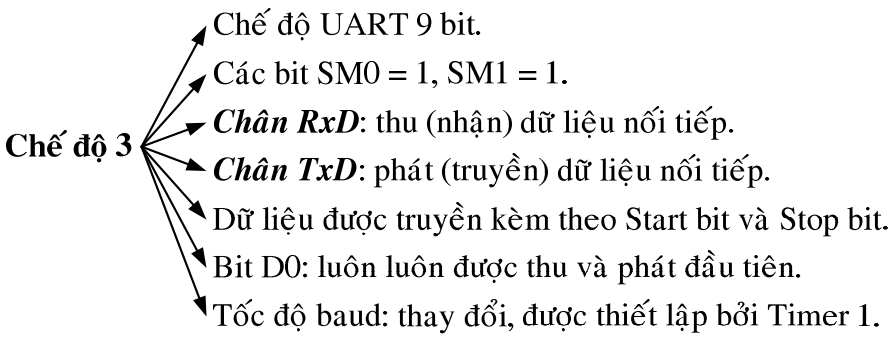


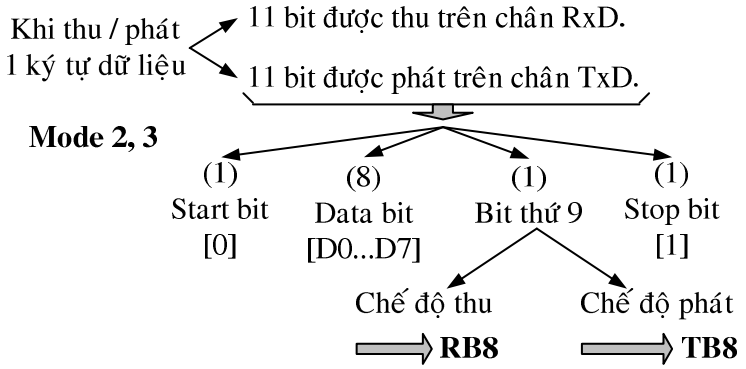
Khuông dạng của một dữ liệu khi sử dụng chế độ UART 9 bit:



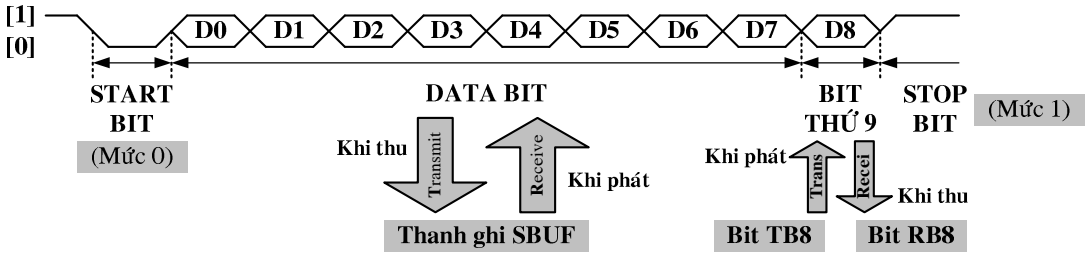
4. Chế độ 3 – UART 9 bit có tốc độ baud thay đổi:

(Tương tự như UART 9 bit, chỉ khác ở tốc độ baud có thể thay đổi)





Khuông dạng của một dữ liệu khi sử dụng chế độ UART 9 bit:



V. KHỞI ĐỘNG VÀ TRUY XUẤT CÁC THANH GHI:

1. Bit cho phép thu (*nhận*) dữ liệu (*REN: Receive Enable*):

- ***Công dụng:*** dùng để cho phép (*hoặc không cho phép*) nhận các ký tự dữ liệu.
  REN = 1: Cho phép nhận dữ liệu ⇒ lệnh thực hiện: **SETB REN**
  REN = 0: Không cho phép dữ liệu ⇒ lệnh thực hiện: **CLR REN**

2. Bit dữ liệu thứ 9:

- ***Công dụng:*** tùy thuộc vào đặc tính kỹ thuật của thiết bị nối tiếp mà có thể yêu cầu hoặc không yêu cầu bit dữ liệu thứ 9.

  Khi phát dữ liệu: bit dữ liệu thứ 9 ***phải được nạp vào bit TB8*** của SCON trước khi phát đi.
  Khi thu dữ liệu: bit dữ liệu thứ 9 ***sẽ được nạp vào bit RB8*** của SCON sau khi thu xong.

3. Bit kiểm tra chẵn / lẻ (*P: Parity*):

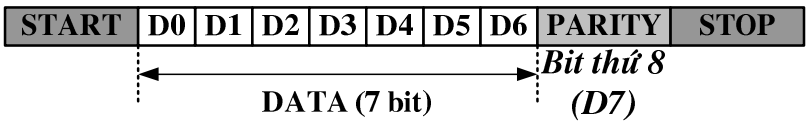
- ***Công dụng:*** Trong chip 8051 thì bit Parity được dùng để thiết lập việc **kiểm tra chẵn** cho 8 bit dữ liệu chứa trong thanh ghi A (*thường dùng để kiểm tra lỗi khi truyền dữ liệu*).

  P = 1 ⇒ Số lượng bit 1 trong thanh ghi A là số lẻ.
  P = 0 ⇒ Số lượng bit 1 trong thanh ghi A là số chẵn.

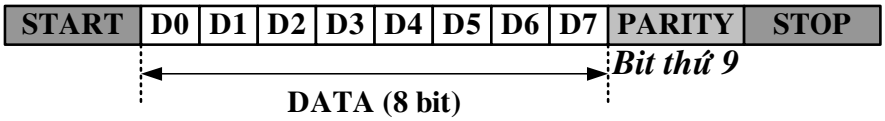
  hoặc

  Số lượng bit 1 trong thanh ghi A và bit P là một **số chẵn**.

***Ở chế độ 1* (*UART 8 bit*)** thì bit chẵn/lẻ do chip 8051 tạo ra có thể được thêm vào tại bit thứ 8 (*vị trí D7*) và khi đó ta chỉ có thể *truyền dữ liệu chỉ có 7 bit*.



Ở chế độ 2, 3 (UART 9 bit) thì bit chẵn/lẻ do chip 8051 tạo ra có thể được thêm vào tại bit thứ 9 (*nghĩa là thêm vào bit TB8 của SCON*) và khi đó ta có thể *truyền dữ liệu có 8 bit*.



Ví dụ: *Truyền dữ liệu (chế độ 2, 3 – UART 9 bit) chứa trong thanh ghi A thông qua port nối tiếp với yêu cầu truyền 8 bit dữ liệu + 1 bit kiểm tra chẵn (bit P).*

Chuỗi lệnh thực hiện:

```
MOV     C, P            ;Chuyển bit kiểm tra chẵn (bit P) vào TB8 và
MOV     TB8, C          ;bit này trở thành bit thứ 9.
MOV     SBUF, A         ;Truyền 8 bit dữ liệu trong A thông qua port.
```

Ví dụ: *Truyền dữ liệu (chế độ 2, 3 – UART 9 bit) chứa trong thanh ghi A thông qua port nối tiếp với yêu cầu truyền 8 bit dữ liệu + 1 bit kiểm tra lẻ (lấy bù bit P).*

Chuỗi lệnh thực hiện:

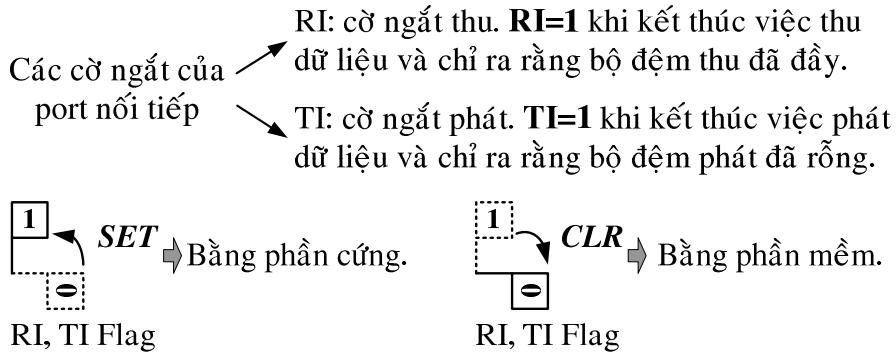
```
MOV     C, P            ;Biến đổi bit kiểm tra chẵn (bit P) thành bit
CPL     C               ;kiểm tra lẻ, chuyển bit kiểm tra lẻ vào TB8 và
MOV     TB8, C          ;bit này trở thành bit thứ 9.
MOV     SBUF, A         ;Truyền 8 bit dữ liệu trong A thông qua port.
```

Ví dụ: *Truyền dữ liệu (chế độ 1 – UART 8 bit) chứa trong thanh ghi A thông qua port nối tiếp với yêu cầu truyền 7 bit dữ liệu + 1 bit kiểm tra chẵn (bit P).*

Chuỗi lệnh thực hiện:

```
CLR     ACC.7           ;Xoá bit thứ 8 (D7) trong thanh ghi A.
MOV     C, P            ;Sao chép bit kiểm tra chẵn vào C.
MOV     ACC.7, C        ;Đặt bit kiểm tra chẵn vào bit thứ 8 trong A.
MOV     SBUF, A         ;Truyền 7 bit dữ liệu cộng bit kiểm tra chẵn.
```

4. Các cờ ngắt của port nối tiếp:

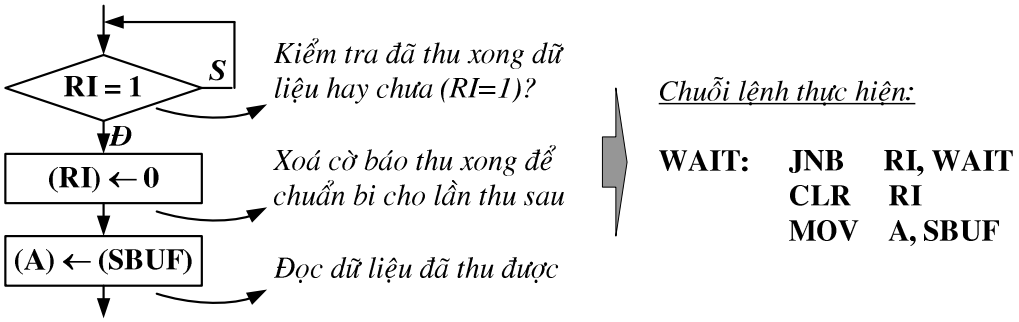


Từ phần trình bày trên đây, ta có thể thấy rằng:

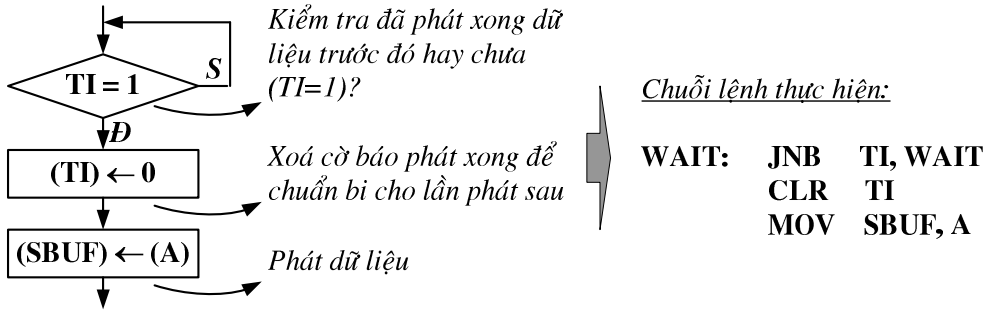
- Thông qua việc kiểm tra ***cờ ngắt TI*** có thể biết được chip 8051 đã sẵn sàng để truyền một byte dữ liệu hay chưa. Cần chú ý rằng, ở đây cờ TI được đặt (*TI = 1*) khi 8051 đã hoàn tất việc truyền một byte dữ liệu, còn được xoá (*TI=0*) thì phải do người lập trình thực hiện bằng lệnh (***CLR TI***). Cũng nên nhớ rằng, nếu ghi một byte vào thanh ghi SBUF trước khi cờ TI được đặt (*TI = 1*) thì sẽ có nguy cơ bị mất phần dữ liệu trước đó do chưa kịp truyền đi. Cờ TI có thể được kiểm tra bằng lệnh (***JNB TI,…***) hoặc sử dụng phương pháp ngắt (*sẽ được trình bày trong chương tiếp theo*).

- Thông qua việc kiểm tra ***cờ ngắt RI*** có thể biết được chip 8051 đã nhận xong một byte dữ liệu hay chưa. Cần chú ý rằng, ở đây cờ RI được đặt (*RI = 1*) khi 8051 đã hoàn tất việc nhận một byte dữ liệu, còn được xoá (*RI=0*) thì phải do người lập trình thực hiện bằng lệnh (***CLR RI***). Cũng nên nhớ rằng, nếu không tiến hành cất nội dung của thanh ghi SBUF vào nơi an toàn thì sẽ có nguy cơ bị mất dữ liệu vừa nhận được do dữ liệu tiếp theo được chuyển vào. Cờ RI có thể được kiểm tra bằng lệnh (***JNB RI,…***) hoặc sử dụng phương pháp ngắt (*sẽ được trình bày trong chương tiếp theo*).

Lưu đồ và đoạn lệnh để kiểm tra và thu một dữ liệu nối tiếp từ thiết bị bên ngoài vào chip 8051 (*chứa vào A*):

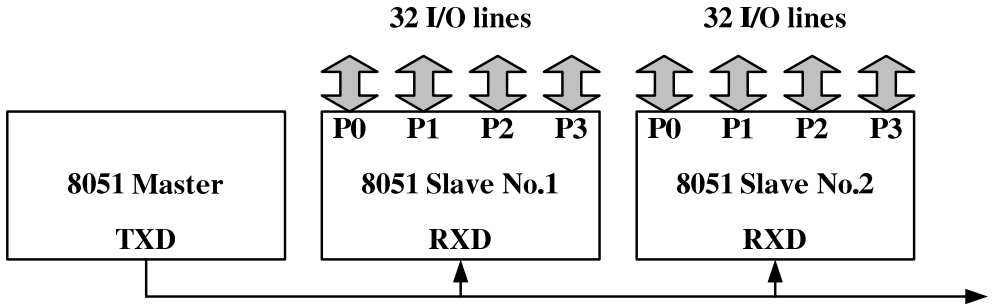


Lưu đồ và đoạn lệnh để kiểm tra và phát một dữ liệu nối tiếp từ chip 8051 (*chứa trong A*) ra thiết bị bên ngoài:



VI. TRUYỀN THÔNG ĐA XỬ LÝ:

Các chế độ 2 và 3 là các chế độ dự phòng cho việc truyền thông đa xử lý. Trong các chế độ này, 9 bit dữ liệu được thu và bit thứ 9 đưa đến RB8. Port có thể được lập trình sao cho khi bit stop được nhận, ngắt do port nối tiếp chỉ được tích cực nếu RB8=1. Đặc trưng này có thể được bằng cách set bit SM2 trong thanh ghi SCON bằng 1. Một ứng dụng cho điều này là một môi trường mạng sử dụng nhiều 8051 được sắp xếp theo mô hình chủ/tớ (*master/slave*) như hình dưới đây.

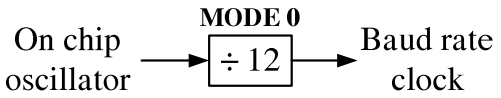


Khi bộ vi xử lý chủ (*master*) muốn truyền một khối dữ liệu đến một trong nhiều bộ xử lý tớ (*slave*), trước tiên bộ vi xử lý chủ phát đi một byte địa chỉ nhận dạng bộ vi xử lý tớ đích. Một byte địa chỉ khác với một byte dữ liệu ở chỗ bit thứ 9 là **1** (*đối với byte địa chỉ*) và là **0** (*đối với byte dữ liệu*). Một byte địa chỉ ngắt tất cả các bộ vi xử lý tớ để cho mỗi một bộ vi xử lý tớ có thể khảo sát byte nhận được để kiểm tra xem có phải là bộ vi xử lý tớ đang được định địa chỉ không. Bộ vi xử lý tớ được định địa chỉ sẽ xoá bit SM2 của mình và chuẩn bị nhận các byte dữ liệu theo sau. Các bộ vi xử lý tớ không được định địa chỉ có các bit SM2 của chúng được set bằng 1 và thực thi các công việc của riêng chúng, bỏ qua không nhận các byte dữ liệu. Các bộ vi xử lý này sẽ được ngắt lần nữa khi bộ vi xử lý chủ phát tiếp byte địa chỉ kế. Các sơ đồ cụ thể có thể được nêu ra sao cho một khi liên kết chủ tớ đã được thiết lập, bộ vi xử lý tớ cũng có thể phát đến bộ vi xử lý chủ. Mưu mẹo ở đây là không sử dụng bit dữ liệu thứ 9 sau khi liên kết vừa được thiết lập (*ngược lại các bộ vi xử lý tớ khác có thể được chọn một cách không cố ý*).

SM2 không ảnh hưởng đến chế độ 0, và trong chế độ 1 thì bit này có thể được dùng để kiểm tra sự hợp lệ của bit stop. Ở chế độ 1 thu, nếu SM2 = 1, ngắt thu sẽ không được tích cực trừ khi bit stop thu được là hợp lệ.

VII. TỐC ĐỘ BAUD CỦA PORT NỐI TIẾP:

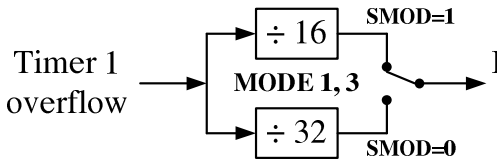
1. Tốc độ baud cho chế độ 0:



$$Baud\ rate = \frac{f_{osc}}{12}$$

On chip oscillator: mạch dao động trên chip.
Baud rate clock: xung clock tốc độ baud.
Baud rate: tốc độ baud.
f_{OSC}: tần số mạch dao động trên chip.
Timer 1 overflow rate: tốc độ tràn Timer 1.

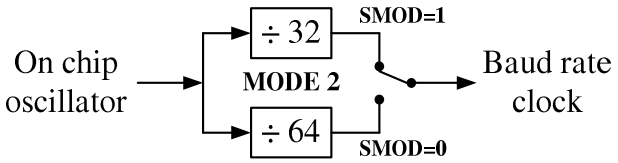
2. Tốc độ baud cho chế độ 1, 3:



SMOD = 1 $Baud\ rate = \frac{Timer\ I\ overflow\ rate}{16}$

SMOD = 0 $Baud\ rate = \frac{Timer\ I\ overflow\ rate}{32}$

3. Tốc độ baud cho chế độ 2:



$Baud\ rate = \frac{f_{osc}}{32}$ **SMOD = 1**

$Baud\ rate = \frac{f_{osc}}{64}$ **SMOD = 0**

- ***Lưu ý:***
  - Sau khi hệ thống *reset* thì bit SMOD = 0 (*chế độ mặc định*).
  - Vì thanh ghi PCON không được định địa chỉ từng bit, nên để tăng gấp đôi tốc độ baud (*tức làm cho SMOD=1*) ta phải thực hiện bằng những dòng lệnh sau:

```
MOV  A, PCON        ;Lấy giá trị từ thanh ghi PCON.
SETB  ACC.7         ;SMOD = 1.
MOV  PCON, A        ;Chuyển giá trị mới vào PCON.
```

4. Sử dụng Timer 1 làm xung clock tốc độ baud cho port nối tiếp:

Kỹ thuật tạo xung clock tốc độ baud bằng Timer 1:

1. Chọn chế độ cho Timer 1
- Chế độ 13 bit (Mode 0).
- Chế độ 16 bit (Mode 1).
- Chế độ 8 bit tự nạp lại (Mode 2) ⇒ ***thường dùng***.
- Chế độ 8 bit chia xẻ (Mode 3).

2. Nạp giá trị thích hợp vào thanh ghi TH1 để có tốc độ tràn đúng ⇒ tạo ra tốc độ baud cho port nối tiếp.

Về chọn chế độ: thường dùng Timer 1 ở chế độ 8 bit tự động nạp lại (*Mode 2*). Các tốc độ baud rất chậm có thể nhận được bằng cách sử dụng chế độ 16 bit (*Mode 1*).

- ***Ví dụ:*** *Khởi động thanh ghi TMOD để dùng T1 làm bộ tạo xung tốc độ baud (cho T1 hoạt động ở chế độ 2):*

MOV TMOD, #2xH x: dành cho Timer 0

Về chọn tốc độ baud:

Gọi M là giá trị cần nạp cho thanh ghi TH1 để có tốc độ baud như yêu cầu, ta có:

$$M = -\frac{f_{Timer}}{Timer\ 1\ overflow\ rate}$$

Mà: $f_{Timer} = \frac{f_{Timer}}{12}$ và $Baud\ rate = \frac{Timer\ 1\ overflow\ rate}{16\ (hoặc\ 32)}$

$$\Rightarrow \quad M = -\frac{f_{Osc}}{12} \times \frac{1}{Baud\ rate \times 16\ (hoặc\ 32)}$$

Vậy ta có:

$$M = -\frac{f_{Osc}}{192 \times Baud\ rate}, (SMOD = 1) \text{ hoặc } M = -\frac{f_{Osc}}{384 \times Baud\ rate}, (SMOD = 0)$$

Trong đó: f_{Osc} (Hz): tần số thạch anh.
Baud rate (bps): tốc độ baud của port nối tiếp.

- ***Ví dụ:*** *Tạo tốc độ baud là 1200 với trường hợp SMOD = 0 và chip 8051 dùng thạch anh 12 MHz.*

Gọi M là giá trị cần nạp cho thanh ghi TH1 để có tốc độ baud như yêu cầu, ta có:

$$M = -\frac{f_{Osc}}{384 \times Baud\ rate}, (SMOD = 0)$$

$$\Rightarrow M = -\frac{12.10^6}{384 \times 1200} = -26,0416 \cong -26 \ (làm\ tròn\ số).$$

Chuyển giá trị này vào thanh ghi TH1:

MOV TH1, #(-26) hoặc **MOV TH1, #E6H**

Chú ý: Do việc làm tròn cho nên sẽ có một sai số nhỏ trong việc xác định chính xác tốc độ baud. Cho nên để có tốc độ baud chính xác trong việc truyền dữ liệu thông qua port nối tiếp thì người ta thường dùng thạch anh dao động có tần số **11,0592 MHz** (*thay vì là 12 MHz*).

Ví dụ:

$$\Rightarrow M = -\frac{11{,}0592.10^6}{384\times 1200} = -24 \quad (\textit{không cần phải làm tròn số})$$

- *Bảng tính tốc độ baud cho port nối tiếp:*

Tốc độ baud	***Tần số thạch anh***	***SMOD***	***Giá trị nạp cho TH1*** (làm tròn)	***Giá trị nạp cho TH1*** (chưa làm tròn)	***Tốc độ baud thực tế***	***Sai số***
19200	12,000 MHz	1	-3 (FDH)	-3,25	20833	8,5%
9600	12,000 MHz	1	-7 (F9H)	-6,51	8928	7%
2400	12,000 MHz	0	-13 (F3H)	-13,02	2404	0,16%
1200	12,000 MHz	0	-26 (E6H)	-26,4	1202	0,16%
19200	11,0592 MHz	0	-15 (F1H)	-15	19200	0%
9600	11,0592 MHz	0	-3 (FDH)	-3	9600	0%
2400	11,0592 MHz	0	-12 (F4H)	-12	2400	0%
1200	11,0592 MHz	0	-24 (E8H)	-24	1200	0%

VIII. CÁC BƯỚC CƠ BẢN LẬP TRÌNH PORT NỐI TIẾP:

Trong các chế độ truyền dữ liệu nối tiếp của 8051 đã nêu trên thì trên thực tế sử dụng, để thực hiện việc thu và phát dữ liệu nối tiếp giữa chip 8051 với các thiết bị khác (*8051, máy tính, các thiết bị SPI,...*) thường người lập trình chỉ sử dụng hai chế độ sau : Mode 1 (*UART 8 bit có tốc độ baud thay đổi*) hoặc Mode 3 (*UART 9 bit có tốc độ baud thay đổi*). Còn hai chế độ còn lại thì rất ít sử dụng khi cần truyền dữ liệu nối tiếp. Cho nên ở đây chúng ta chỉ xem xét đến trình tự thực hiện việc lập trình (*bao gồm thao tác khởi động và điều khiển thu/phát dữ liệu*) để 8051 có thể truyền (*phát*) và nhận (*thu*) dữ liệu thông qua port nối tiếp theo hai chế độ UART nêu trên.

1. Lập trình 8051 truyền (*phát*) dữ liệu nối tiếp:

- Chọn chế độ hoạt động cho port nối tiếp:

MOV SCON, #...(1)…

- Chọn chế độ hoạt động Timer 1, cho bit GATE=0 và C/T=0:

MOV TMOD, #...(2)…

- Chọn giá trị thích hợp (*căn cứ vào tốc độ baud*) cho Timer 1:

MOV TH1, #...(3)…

- Cho Timer 1 chạy:

SETB TR1

- Kiểm tra xem đã phát xong toàn bộ dữ liệu trước đó hay chưa?

JNB TI, $

hoặc

WAIT: JNB TI, WAIT

- Xoá cờ ngắt phát TI (*chuẩn bị cho lần phát dữ liệu tiếp theo*):

CLR TI

- Ghi dữ liệu cần phát vào port nối tiếp để phát đi:

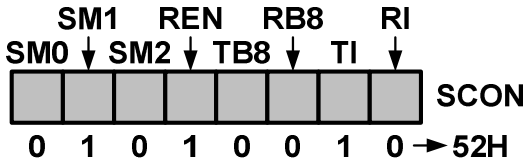
MOV SBUF, ...(4)…

- Quay trở lại bước 5 để phát một dữ liệu tiếp theo.

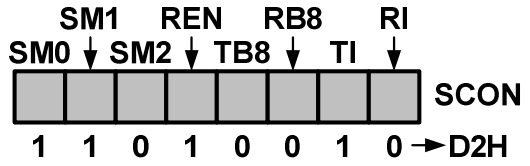
Lưu ý:

(1): Giá trị dùng để qui định chế độ hoạt động của port nối tiếp. Để đơn giản trong việc lập trình, ta có thể khởi động thanh ghi SCON theo như trình bày dưới đây:

Mode 1: **...(1)… = 52H**

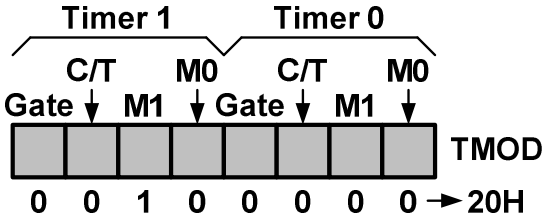


Mode 3: **...(1)… = D2H**



(2): Giá trị dùng để qui định chế độ hoạt động của Timer 1 (*dùng để tạo tốc độ baud cho việc truyền dữ liệu nối tiếp*). Để đơn giản trong việc lập trình, ta có thể khởi động thanh ghi TMOD theo như trình bày dưới đây (*chủ yếu ở đây ta chỉ cần sử dụng Timer 1 ở Mode 2 – Chế độ 8 bit tự động nạp lại*):

Mode 2: **...(2)… = 20H**



(3): Giá trị dùng để qui định tốc độ baud cho port nối tiếp. Giá trị này phụ thuộc vào tần số thạch anh, bit SMOD và tốc độ baud mà người lập trình mong muốn (*xem thêm "**Bảng tính tốc độ baud cho port nối tiếp**" như trên đã trình bày*).

...(3)… = M

$$M = -\frac{f_{Osc}}{192 \times Baud\ rate}, (SMOD = 1) \quad \text{hoặc} \quad M = -\frac{f_{Osc}}{384 \times Baud\ rate}, (SMOD = 0)$$

Trong đó: f_{Osc} (Hz): tần số thạch anh.

Baud rate (bps): tốc độ baud của port nối tiếp.

(4): Dữ liệu cần phát đi thông qua port nối tiếp. Dữ liệu này có thể là nội dung của một ô nhớ, thanh ghi hoặc một giá trị tức thời.

Nên nhớ rằng, nếu có yêu cầu thì bit gửi kèm theo (*ví dụ như bit Parity*) cần phải thêm vào trước khi tiến hành quá trình phát dữ liệu (*Mode 1: kèm thêm vào vị trí của bit D7, Mode 3: kèm thêm vào vị trí của bit TB8*).

2. Lập trình 8051 nhận (*thu*) dữ liệu nối tiếp:

- Chọn chế độ hoạt động cho port nối tiếp:

MOV SCON, #...(1)…

- Chọn chế độ hoạt động Timer 1, cho bit GATE=0 và C/T=0:

MOV TMOD, #...(2)…

- Chọn giá trị thích hợp (*căn cứ vào tốc độ baud*) cho Timer 1:

MOV TH1, #...(3)…

- Cho Timer 1 chạy:

SETB TR1

- Kiểm tra xem đã thu toàn bộ dữ liệu hay chưa?

JNB RI, $

hoặc

WAIT: JNB RI, WAIT

- Xoá cờ ngắt thu RI (*chuẩn bị cho lần thu dữ liệu tiếp theo*):

CLR RI

- Cất dữ liệu vừa thu được vào nơi an toàn (*tránh bị mất dữ liệu*):

MOV ...(4)…, SBUF

- Quay trở lại bước 5 để nhận một dữ liệu tiếp theo.

Lưu ý:

- **(1)**: Xem thêm "***Lập trình 8051 truyền (phát) dữ liệu nối tiếp***".
- **(2)**: Xem thêm "***Lập trình 8051 truyền (phát) dữ liệu nối tiếp***".
- **(3)**: Xem thêm "***Lập trình 8051 truyền (phát) dữ liệu nối tiếp***".
- **(4)**: Địa chỉ của một ô nhớ, thanh ghi mà dữ liệu thu được từ port nối tiếp sẽ lưu giữ vào trong đó.

Nên nhớ rằng, nếu có yêu cầu thì bit gửi kèm theo (*ví dụ như bit Parity*) cần phải được xử lý trước khi tiến hành việc cất dữ liệu thu được (*Mode 1: nằm tại vị trí của bit D7, Mode 3: nằm tại vị trí của bit RB8*).

IX. CÁC VÍ DỤ MINH HỌA:

1. Ví dụ 1: (*Chọn tốc độ baud*)

Chip 8051 sử dụng thạch anh 11,0592MHz. Hãy xác định giá trị cần nạp cho thanh ghi TH1 để có được các tốc độ baud: 9600, 2400, 1200 (nếu SMOD=0) và 19200 (nếu SMOD=1).

Giải

Xét trường hợp SMOD=0:

Gọi M là giá trị cần nạp cho thanh ghi TH1 để có tốc độ baud như yêu cầu, ta có:

$$M = -\frac{f_{Osc}}{384 \times Baud\ rate}$$

Baud rate = 9600

$$\Rightarrow M = -\frac{11,0592 \times 10^6}{384 \times 9600} = -3 \Rightarrow \textbf{(TH1) = -3} \text{ hay } \textbf{(TH1) = FDH}.$$

Baud rate = 2400

$$\Rightarrow M = -\frac{11{,}0592\times10^6}{384\times2400} = -12 \Rightarrow \textbf{(TH1) = -12} \text{ hay } \textbf{(TH1) = F4H}.$$

Baud rate = 1200

$$\Rightarrow M = -\frac{11{,}0592\times10^6}{384\times1200} = -24 \Rightarrow \textbf{(TH1) = -24} \text{ hay } \textbf{(TH1) = F8H}.$$

Xét trường hợp SMOD=1:

Gọi M là giá trị cần nạp cho thanh ghi TH1 để có tốc độ baud như yêu cầu, ta có:

$$M = -\frac{f_{Osc}}{192\times Baud\ rate}$$

Baud rate = 19200

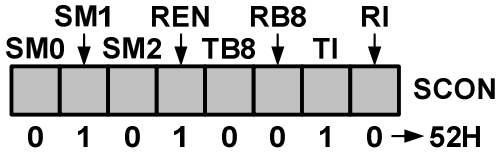
$$\Rightarrow M = -\frac{11{,}0592\times10^6}{192\times19200} = -3 \Rightarrow \textbf{(TH1) = -3} \text{ hay } \textbf{(TH1) = FDH}.$$

2. Ví dụ 2: (*Khởi động port nối tiếp*)

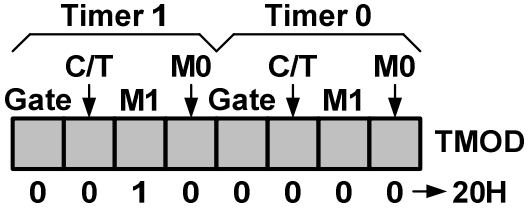
Viết một chuỗi lệnh để khởi động port nối tiếp sao cho port này hoạt động như một UART 8 bit với tốc độ baud là 2400, sử dụng Timer1 để cung cấp xung clock tốc độ baud. Chip 8051 sử dụng thạch anh 12MHz.

Giải

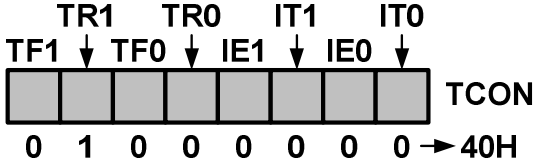
Để khởi động port nối tiếp có cấu hình như trên ta cần tác động đến các thanh ghi: SCON, TMOD, TCON và TH1.



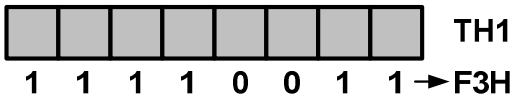
SM0 = 0, SM1 = 1, SM2 = 0 → chế độ UART 8 bit.
REN = 1 → cho phép port nối tiếp thu dữ liệu.
TI = 1 → cho phép port sẵn sàng phát dữ liệu (*bộ đệm phát rỗng*).
RI = 0 → cho phép port sẵn sàng thu dữ liệu (*bộ đệm thu rỗng*).



GATE = 0, C/T = 0, M1 = 1, M0 = 0 → Timer 1 ở chế độ định thời 8 bit tự nạp lại.



TR1 = 1→ cho phép Timer 1 hoạt động.



Gọi M là giá trị cần nạp cho thanh ghi TH1 để có tốc độ baud như yêu cầu, ta có:

$$M = -\frac{f_{Osc}}{384 \times Baud\ rate}(SMOD = 0)$$

$\Rightarrow M = -\frac{12.10^6}{384 \times 2400} = -13,02 \cong -13$ (làm tròn số).

Chuyển giá trị này vào thanh ghi TH1:

MOV TH1, #(-13) hoặc **MOV TH1, #F3H**

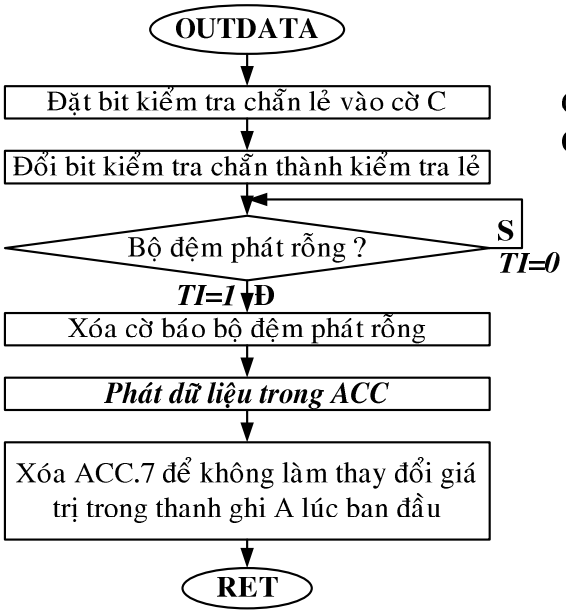
Chuỗi lệnh khởi động port nối tiếp được viết như sau:

```
MOV SCON, #52H
MOV TMOD, #20H
MOV TH1, #-13
SETB TR1
```

3. Ví dụ 3: (*Chương trình con phát (xuất) dữ liệu*)

Giả sử port nối tiếp đã được khởi động (như ở ví dụ 1). Hãy viết một chương trình con để phát dữ liệu (dạng 7 bit) chứa trong thanh ghi A ra port nối tiếp với bit kiểm tra lẻ là bit thứ 8. Chú ý rằng, việc trở về từ chương trình con này không được làm thay đổi nội dung thanh ghi A.

Giải



Chương trình được viết như sau:

```
OUTDATA:
        MOV     C, P
        CPL     C
        MOV     ACC.7, C
        JNB     TI, $
        CLR     TI
        MOV     SBUF, A
        CLR     ACC.7
        RET
```

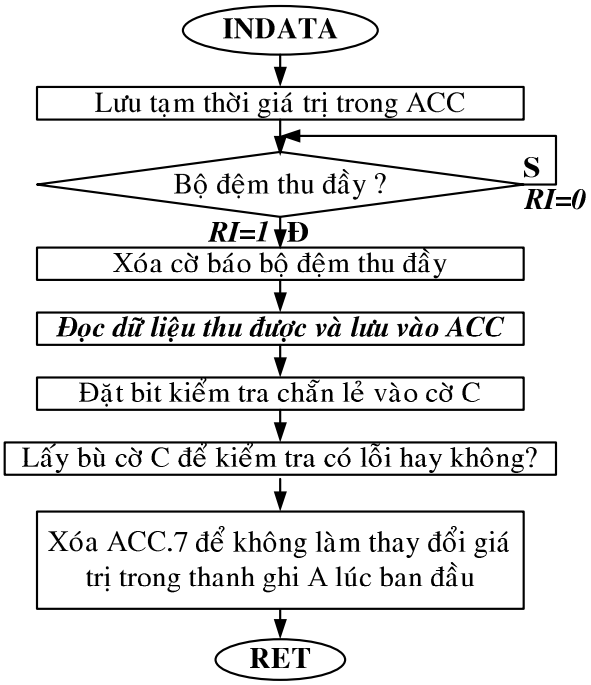
Ba lệnh đầu tiên đặt bit kiểm tra lẻ vào bit 7 của thanh ghi A (*ACC.7*). Do bit P trong thanh ghi PSW được thiết lập để kiểm tra chẵn cho giá trị trong thanh ghi A, cho nên bit này phải được lấy bù để trở thành bit kiểm tra lẻ trước khi đặt vào ACC.7. Lệnh JNB tạo ra một vòng lặp chờ để kiểm tra cờ ngắt phát TI cho đến khi cờ này được set bằng 1. Khi TI=1 (*do việc phát ký tự trước đó vừa kết thúc*), bit này sẽ được xóa và sau đó ký tự trong thanh ghi A được ghi vào bộ đệm của port nối tiếp SBUF và

việc phát ký tự được bắt đầu ở lần tràn kế của bộ đếm tạo xung clock cho port nối tiếp. Sau cùng bit ACC.7 sẽ được xóa để giá trị trả về giống như khi mã 7 bit được chuyển đến chương trình con.

4. Ví dụ 4: (*Chương trình con thu (nhận) dữ liệu*)

Giả sử port nối tiếp đã được khởi động (như ở ví dụ 1). Hãy viết một chương trình con để thu dữ liệu từ port nối tiếp và nạp giá trị thu được vào thanh ghi A (dữ liệu thu được có dạng 7 bit dữ liệu + 1 bit kiểm tra lẻ, tương tự dữ liệu từ ví dụ 2). Sử dụng bit thứ 8 thu được làm bit kiểm tra lẻ và set cờ nhớ C nếu có lỗi chẵn lẻ.

Giải



Chương trình được viết như sau:

```
INDATA:
        JNB     RI, $
        CLR     RI
        MOV     A, SBUF
        MOV     C, P
        CPL     C
        CLR     ACC.7
        RET
```

Chương trình con này bắt đầu bằng việc chờ cờ ngắt thu RI được set bằng 1 để chỉ ra rằng ký tự đã sẵn sàng trong bộ đệm thu SBUF (*để được đọc*). Khi RI=1, lệnh JNB chuyển điều khiển đến lệnh tiếp theo sau lệnh này. Cờ RI được xóa và mã trong SBUF được chứa vào thanh ghi A. Do bit P trong thanh ghi PSW được thiết lập để kiểm tra chẵn cho giá trị trong thanh ghi A. Cho nên bit này sẽ được set bằng 1 nếu nội dung thanh ghi A (*tức dữ liệu vừa thu được*) có chứa bit kiểm tra lẻ ở bit thứ 7 của thanh ghi này và ngược lại thì bit này sẽ được xoá bằng 0 thông qua lệnh CPL. Việc di chuyển bit P vào trong cờ nhớ CY sẽ làm cho CY=0 nếu không có lỗi hoặc CY=1 nếu có một lỗi chẵn lẻ . Sau cùng bit ACC.7 sẽ được xóa để đảm bảo rằng chỉ có mã 7 bit được trả về cho chương trình đã gọi.

5. Ví dụ 5: (*Truyền dữ liệu*)

Viết chương trình cho 8051 (f_{Osc}=11,0592MHz) để truyền liên tục một ký tự "A" thông qua port nối tiếp với tốc độ 4800 baud (Mode 1).

Giải

- ***Tính toán:*** *Dựa vào những công thức đã học, ta có:*

  Port nối tiếp (*Mode 1*) ⇒ **(SCON) = 52H**

  Timer 1 (*Mode 2*) ⇒ **(TMOD) = 20H**

  Baud rate = 4800 và f_{Osc}=11,0592MHz ⇒ **(TH1) = -6** với **(SMOD) = 0**

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
        MOV     SCON, #52H          ;UART 8 bit, tốc độ baud thay đổi.
        MOV     TMOD, #20H          ;Chế độ 8 bit tự nạp lại.
        MOV     TH1, #(-6)          ;Baud rate = 4800.
        SETB    TR1                 ;Khởi động Timer 1.
LOOP:
        JNB     TI, $               ;Kiểm tra phát dữ liệu trước hoàn tất?
        CLR     TI                  ;Xoá cờ TI, chuẩn bị cho lần phát kế tiếp.
        MOV     SBUF, #"A"          ;Phát ký tự "A".
        SJMP    LOOP                ;Lặp lại quá trình phát.
        END                         ;Kết thúc chương trình.
```

6. Ví dụ 6: (*Truyền dữ liệu*)

Viết chương trình cho 8051 (f_{Osc}=11,0592MHz) để truyền liên tục chuỗi ký tự "TTCNDT" thông qua port nối tiếp với tốc độ 19200 baud (Mode 1).

Giải

- ***Tính toán:***

Port nối tiếp (*Mode 1*) $\Rightarrow$ **(SCON) = 52H**

Timer 1 (*Mode 2*) $\Rightarrow$ **(TMOD) = 20H**

Baud rate = 19200 và f_{Osc}=11,0592MHz $\Rightarrow$ **(TH1) = -3** với **(SMOD) = 1**

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
        MOV     SCON, #52H          ;UART 8 bit, tốc độ baud thay đổi.
        MOV     TMOD, #20H          ;Chế độ 8 bit tự nạp lại.
        MOV     TH1, #(-3)          ;Baud rate = 19200.
        MOV     A, PCON             ;Lấy giá trị từ thanh ghi PCON.
        SETB    ACC.7               ;SMOD = 1.
        MOV     PCON, A             ;Chuyển giá trị mới vào PCON.
        SETB    TR1                 ;Khởi động Timer 1.
LOOP:
        MOV     DPTR, #MYDATA       ;Nạp con trỏ vùng dữ liệu.
NEXT:
        CLR     A                   ;Xoá ACC, A = 0
        MOVC    A, @A+DPTR          ;Lấy dữ liệu tại ô nhớ ROM do
                                    ;(A+DPTR) trỏ đến đưa vào A.
        JZ      EXIT                ;Thoát nếu là ký tự Null.
        JNB     TI, $               ;Kiểm tra phát dữ liệu trước hoàn tất?
        CLR     TI                  ;Xoá cờ TI, chuẩn bị phát tiếp.
        MOV     SBUF, A             ;Phát dữ liệu ra port nối tiếp.
        INC     DPTR                ;Tăng con trỏ dữ liệu.
        SJMP    NEXT                ;Lặp lại quá trình phát ký tự kế tiếp.
EXIT:
        SJMP    LOOP                ;Lặp lại từ đầu.
MYDATA:
        DB      "TTCNDT",0          ;Dữ liệu cần truyền đi, có ký tự Null.
        END                         ;Kết thúc chương trình.
```

8. Ví dụ 8: (*Nhận dữ liệu*)

Viết chương trình cho 8051 (f_{Osc}=11,0592MHz) để nhận liên tục các dữ liệu thông qua port nối tiếp với tốc độ 4800 baud (Mode 1) và gửi các dữ liệu nhận được đến P1.

Giải

- ***Tính toán:***

  Port nối tiếp (*Mode 1*) ⇒ **(SCON) = 52H**
  Timer 1 (*Mode 2*) ⇒ **(TMOD) = 20H**
  Baud rate = 4800 và f_{Osc}=11,0592MHz ⇒ **(TH1) = -6** với **(SMOD) = 0**

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
      MOV   SCON, #52H      ;UART 8 bit, tốc độ baud thay đổi.
      MOV   TMOD, #20H      ;Chế độ 8 bit tự nạp lại.
      MOV   TH1, #(-6)      ;Baud rate = 4800.
      SETB  TR1             ;Khởi động Timer 1.
LOOP:
      JNB   RI, $           ;Kiểm tra đã thu dữ liệu hoàn tất?
      CLR   RI              ;Xoá cờ RI, chuẩn bị cho lần thu kế tiếp.
      MOV   A, SBUF         ;Thu và cất dữ liệu vào ACC.
      MOV   P1, A           ;Gửi dữ liệu thu được ra P1.
      SJMP  LOOP            ;Lặp lại quá trình phát.
      END                   ;Kết thúc chương trình.
```

9. Ví dụ 9: (*Thu và phát dữ liệu*)

Cho port nối tiếp của 8051 (f_{Osc}=11,0592MHz) được nối với cổng COM của máy tính PC (giả sử rằng trên máy tính đã có sẵn chương trình để gửi và nhận dữ liệu nối tiếp thông qua cổng COM). Hãy viết chương trình cho 8051 để thực hiện các công việc sau:

- *Gửi câu thông báo "READY" đến máy tính.*
- *Liên tục nhận các dữ liệu nối tiếp từ máy tính gửi đến và chuyển các dữ liệu này ra P1 của 8051.*
- *Liên tục lấy các dữ liệu từ P2 của 8051 và phát các dữ liệu này đến máy tính thông qua port nối tiếp.*

Biết rằng 8051 truyền dữ liệu nối tiếp ở Mode 1 với tốc độ baud là 9600.

Giải

- ***Tính toán:***

  Port nối tiếp (*Mode 1*) ⇒ **(SCON) = 52H**
  Timer 1 (*Mode 2*) ⇒ **(TMOD) = 20H**
  Baud rate = 9600 và f_{Osc}=11,0592MHz ⇒ **(TH1) = -3** với **(SMOD) = 0**

- ***Chương trình:*** *Dựa vào những tính toán trên, ta có:*

```
      MOV   P2, #0FFH       ;Cấu hình P2 là cổng vào.
      MOV   SCON, #52H      ;UART 8 bit, tốc độ baud thay đổi.
      MOV   TMOD, #20H      ;Chế độ 8 bit tự nạp lại.
      MOV   TH1, #(-3)      ;Baud rate = 4800.
      SETB  TR1             ;Khởi động Timer 1.
      MOV   DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
NEXT:                       ;Phần phát câu thông báo
      CLR   A               ;Xoá ACC, A = 0
      MOVC  A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
```

```
                                ;(A+DPTR) trỏ đến đưa vào A.
    JZ          EXIT            ;Thoát nếu là ký tự Null.
    ACALL       OUTDATA         ;Phát dữ liệu, câu thông báo.
    INC         DPTR            ;Tăng con trỏ dữ liệu.
    SJMP        NEXT            ;Lặp lại quá trình phát ký tự kế tiếp.
EXIT:                           ;Phần phát/thu dữ liệu giữa 8051 và PC.
    ACALL       INDATA          ;Thu dữ liệu từ PC.
    MOV         P1, A           ;Chuyển dữ liệu này đến P1.
    MOV         A, P2           ;Lấy dữ liệu từ P2.
    ACALL       OUTDATA         ;Phát dữ liệu này đến PC.
    SJMP        EXIT            ;Lặp lại chu trình làm việc.
INDATA:                        ;Chương trình con thu dữ liệu.
    JNB         RI, $           ;Kiểm tra đã thu dữ liệu hoàn tất?
    CLR         RI              ;Xoá cờ RI, chuẩn bị cho lần thu tiếp.
    MOV         A, SBUF         ;Thu và cất dữ liệu vào ACC.
    RET
OUTDATA:    ;Chương trình con phát dữ liệu.
    JNB         TI, $           ;Kiểm tra phát dữ liệu trước hoàn tất?
    CLR         TI              ;Xoá cờ TI, chuẩn bị cho lần phát tiếp.
    MOV         SBUF, A         ;Phát dữ liệu chứa trong ACC.
    RET
MYDATA:
    DB          "READY",0       ;Câu thông báo, có ký tự Null.
    END                         ;Kết thúc chương trình.
```

X. PHẦN BÀI TẬP:

Bài 1: Viết đoạn lệnh đọc một chuỗi data chứa trong RAM nội từ địa chỉ 30H đến 50H và xuất ra một thiết bị (*ví dụ như màn hình tinh thể lỏng LCD*) được nối với port nối tiếp của 8051 (*chế độ UART 8 bit, 2400 baud*). Cho f_{OSC}=11,0592 MHz.

Bài 2: Viết đoạn lệnh nhận một chuỗi data từ một thiết bị ngoài (*ví dụ như máy đọc mã vạch*) nối với 8051 qua port nối tiếp (*chế độ UART 8 bit, 4800 baud*) và ghi data vào RAM nội từ địa chỉ 40H. Biết rằng chuỗi data gồm 20 byte và f_{OSC}=11,0592 MHz.

Bài 3: Viết đoạn lệnh lấy một chuỗi data chứa trong RAM ngoài bắt đầu từ địa chỉ 2000H và xuất ra một thiết bị được nối với port nối tiếp của 8051 (*chế độ UART 8 bit, 1200 baud*). Chuỗi kết thúc bởi ký tự EOT (*có mã ASCII là 04H*) và ký tự này cũng được xuất ra (f_{OSC}*=11,0592 MHz*).

Bài 4: Làm lại bài 3 nhưng không xuất ký tự EOT.

Bài 5: Viết đoạn lệnh nhận một chuỗi data từ một thiết bị ngoài nối với 8051 qua port nối tiếp (*chế độ UART 8 bit, 9600 baud*) và ghi data vào RAM ngoài bắt đầu từ địa chỉ 4000H. Chuỗi data bắt đầu bằng ký tự STX (02H) và kết thúc bằng ký tự ETX (03H). Không ghi hai ký tự này vào RAM. Cho f_{OSC}=11,0592 MHz.

Bài 6: Viết chương trình con mang tên XUAT có nhiệm vụ lấy một chuỗi data chứa trong RAM ngoài xuất ra port nối tiếp ở chế độ UART 9 bit. Bit thứ 9 là bit parity chẵn. Chuỗi data kết thúc bằng ký tự NULL (00H). đoạn lệnh gọi chương trình con XUAT sẽ đặt địa chỉ bắt đầu của chuỗi vào DPTR trước khi gọi chương trình con XUAT. Giả sử port nối tiếp đã được khởi động.

Bài 7: Viết chương trình con mang tên NHAP có nhiệm vụ nhập một chuỗi data gồm 30 byte từ port nối tiếp ở chế độ UART 9 bit, bit thứ 9 là bit parity lẻ. Nếu data nhận được không bị lỗi thì ghi

vào một vùng nhớ của RAM nội, nếu bị lỗi thì không ghi. đoạn lệnh gọi chương trình con NHAP sẽ đặt địa chỉ đầu của vùng nhớ vào thanh ghi R0 trước khi gọi chương trình con NHAP. Giả sử port nối tiếp đã được khởi động.

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH

KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

CHƯƠNG 6

HOẠT ĐỘNG NGẮT (INTERRUPT)

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.

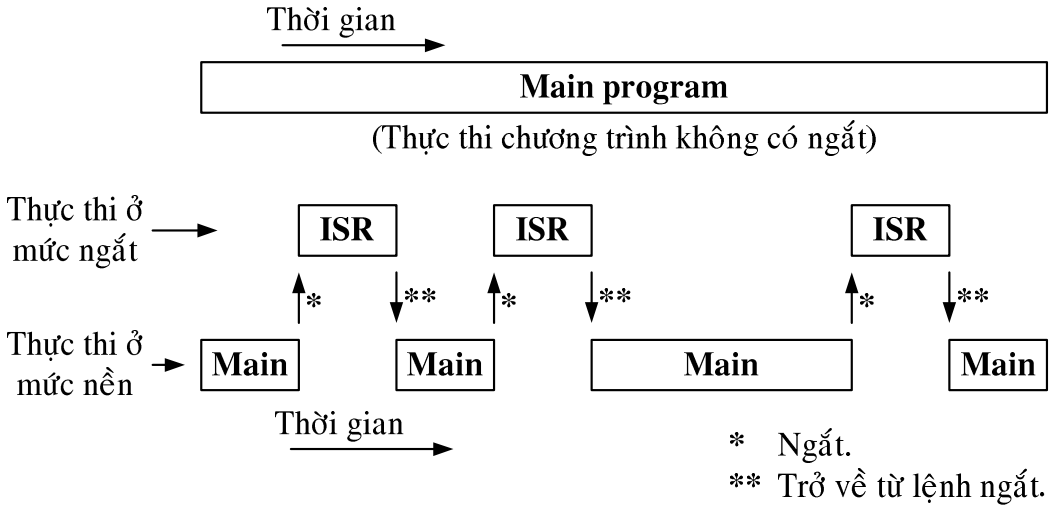
- ***Khác nhau:***

	Ngắt	**Chương trình con**
Thời điểm xảy ra sự kiện	Không biết trước (*hay xảy ra không đồng bộ với chương trình chính*).	Biết trước (*hay xảy ra đồng bộ với chương trình chính*).
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 (*ACALL, LCALL*).

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:



(Thực thi chương trình 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.

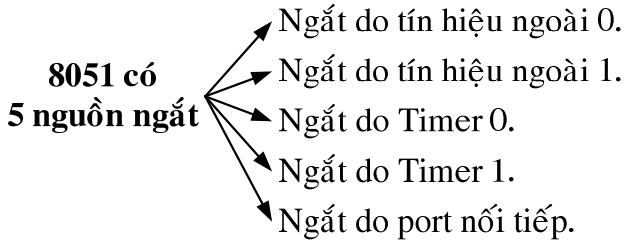
Ở phương pháp thăm dò, bộ vi điều khiển liên tục kiểm tra tình trạng của một thiết bị và khi điều kiện được đáp ứng thì nó sẽ tiến hành phục vụ cho thiết bị này. Sau đó, bộ vi điều khiển chuyển sang kiểm tra trạng thái của thiết bị kế tiếp cho đến khi tất cả thiết bị đều được phục vụ.

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 cờ 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 cờ TF=1. Khi cờ 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\.

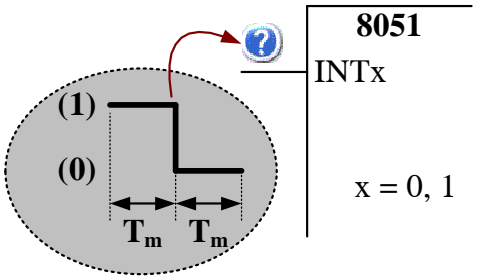
2. Qui định việc chọn loại kích hoạt cho ngắt ngoài:

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.

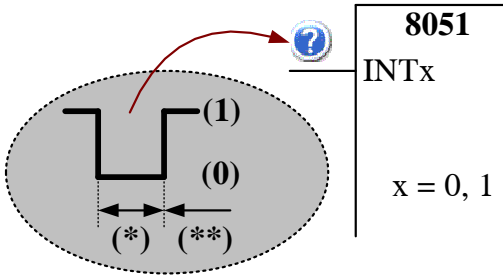
- Đối với loại ngắt kích hoạt cạnh âm (*thời gian tối thiểu*):



f_{OSC} (MHz): tần số thạch anh.
T_m (µs): chu kỳ máy.

$$T_m = \frac{12}{f_{OSC}}$$

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



(*): Duy trì trạng thái (0) cho đến khi ISR tương ứng được thực hiện.

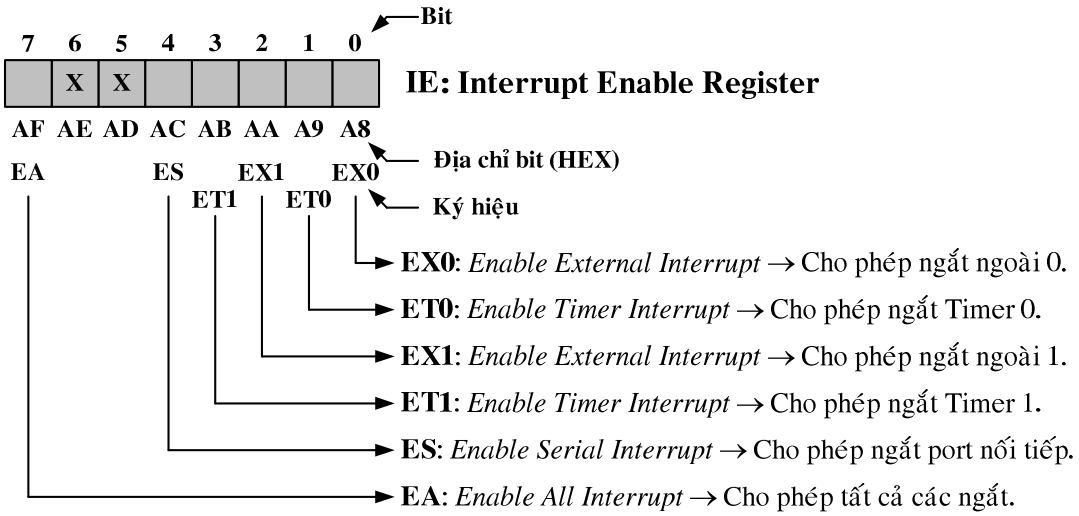
$$(*) = 4 \times T_m$$

(**):Trở về trạng thái (1) trước khi ISR tương ứng được thực hiện xong hoặc trước khi có một ngắt khác được tạo ra.

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.

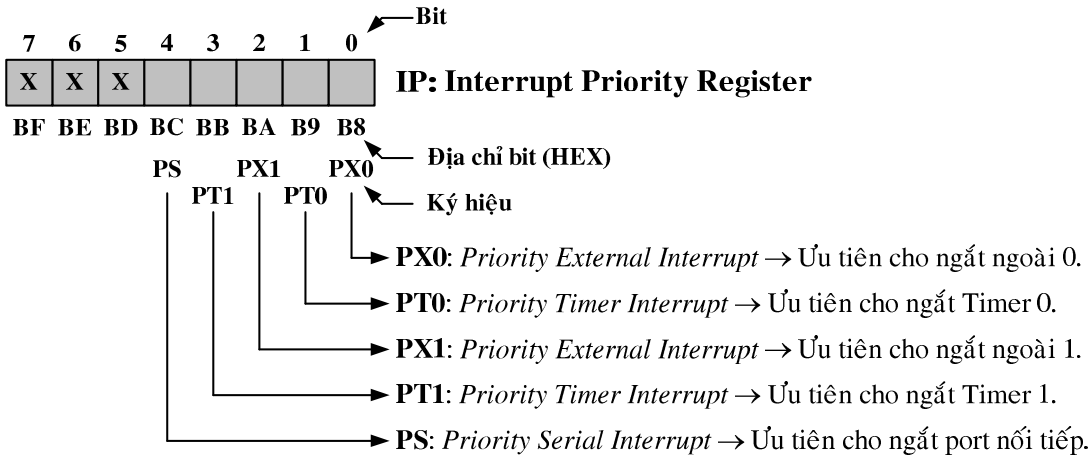
4. Thanh ghi ưu tiên ngắt (*IP*):

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:



Lưu ý: Ưu tiên ở mức cao khi bit = **1**
Ưu tiên ở mức thấp khi bit = **0**

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 phuc 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.***

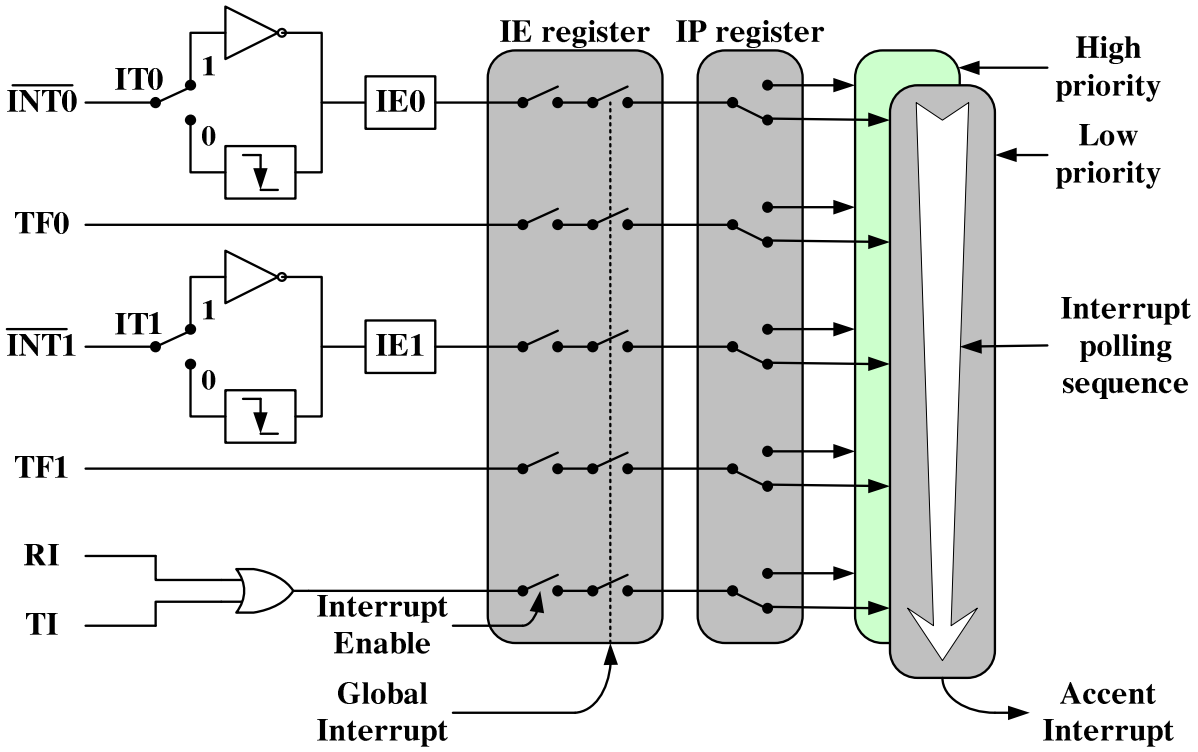
Chuỗi vòng này sẽ là (*được sắp xếp theo thứ tự từ thấp đến cao*):

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 VECTƠ 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 vectơ ngắt:

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

Vectơ 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 trỏ 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 vectơ 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:

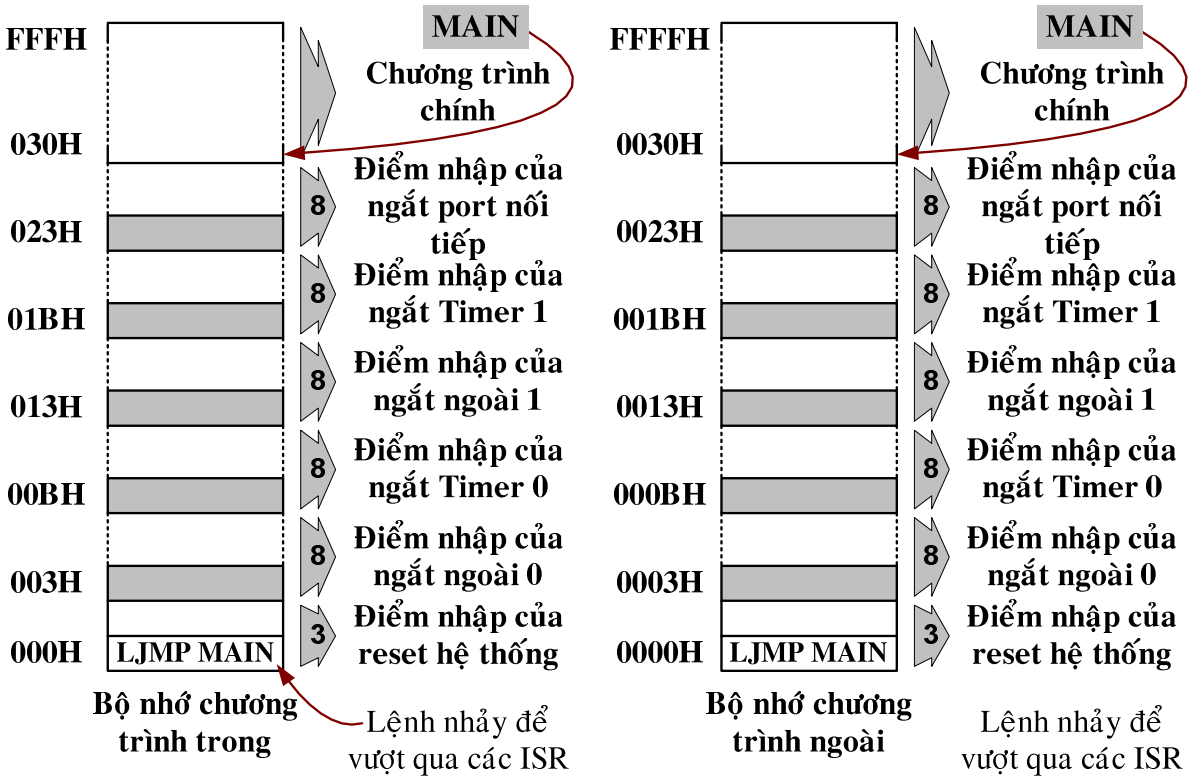
Khi ***thiết kế các chương trình không sử dụng ngắt*** thì ta sẽ gặp phải những trường hợp CPU hoàn toàn tiêu phí thời gian vào việc chờ đợi các tác nhân cần thiết xảy ra (*Ví dụ: sự tràn của cờ TF0, TF1; việc thu xong một dữ liệu và cờ RI=1; việc phát xong một ký tự và cờ TI=1; v.v...*) để sau đó mới 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 đến 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.
        ……
        ……
        ORG  0030H      ;Điểm nhập của chương trình chính.
MAIN:   ……              ;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.*

$\Rightarrow$ 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*) $\rightarrow$ ta phải chuyển ISR này đến một nơi khác trong bộ nhớ chương trình hoặc có thể viết lấn 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.
            ……
            ……
            SJMP $          ;Lệnh cách ly chương trình.
T0ISR:      ……              ;ISR của ngắt Timer 0.
            ……
            RETI            ;Kết thúc ISR của Timer 0.
EX1ISR:     ……              ;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 chổ – không làm gì cả*):

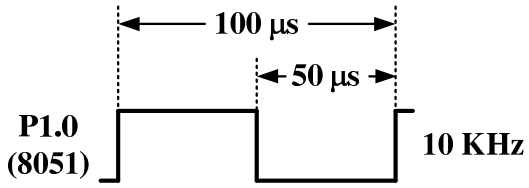
SJMP $

  - 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 chổ*). 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.
```

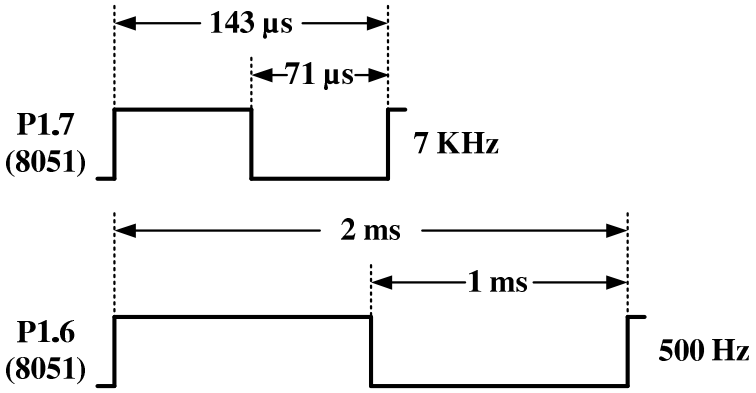
Lưu ý: Lệnh SJMP$ có thể được thay thế bằng một đoạn lệnh để thực thi những công việc khác. Việc thay thế này không ảnh hưởng gì đến việc tạo sóng vuông f=10KHz tại chân P1.0. Vì cứ sau mỗi 50µs thì những công việc đó sẽ bị tạm dừng (*ngắt Timer 0 xuất hiện*) để CPU thực hiện việc tạo sóng vuông rồi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

50µs do chương trình chính đang ở trong vòng lặp "không làm gì". Khi ngắt xuất hiện sau mỗi 50µs, 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 50µs.

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 trỏ 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} = 12MHz.*



Giải

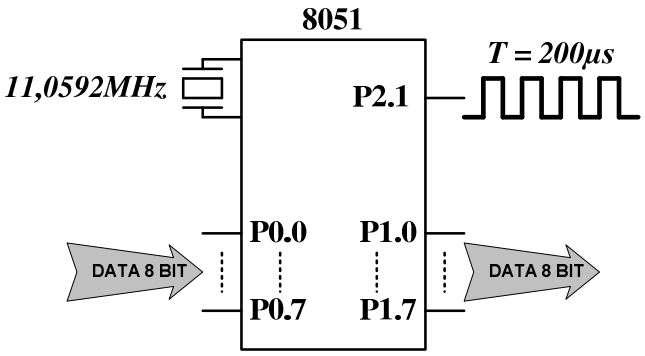
```
        ORG     0000H           ;Điểm nhập reset.
        LJMP    MAIN            ;Nhảy qua khỏi các vectơ ngắt.
        ORG     000BH           ;Điểm nhập ISR của Timer 0.
        LJMP    T0ISR           ;Lệnh nhảy đến ISR Timer 0.
        ORG     001BH           ;Điểm nhập ISR của Timer 1.
        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 71µs.
        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 = 7KHz và f = 500Hz tại chân P1.7 và chân P1.6. Vì cứ sau mỗi 71µs 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 7 KHz trên chân P1.7. Timer 1 hoạt động ở chế độ 1, được sử dụng để tạo ra dạng sóng 500 Hz 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.

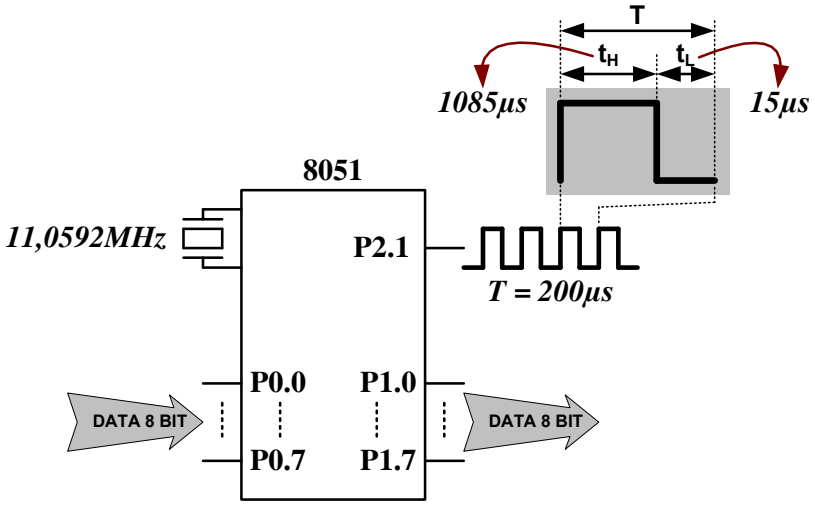
Ví dụ 3: *Viết chương trình liên tục nhận dữ liệu 8 bit ở cổng P0 và sau đó gửi dữ liệu này đến cổng P1. Trong thời gian này cần tạo ra trên chân P2.1 một sóng vuông có chu kỳ là 200µs. Sử dụng Timer 0 để tạo sóng vuông, f_{Osc} = 11,0592MHz.*



Giải

```
        ORG     0000H           ;Điểm nhập reset.
        LJMP    MAIN            ;Nhảy qua khỏi các vectơ 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 100µs (nữa 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 1085µs và thời gian sóng ở mức thấp là 15µs. Sử dụng Timer 1 để tạo sóng vuông,* f_{Osc}*=11,0592MHz.*



Giải

```
        ORG     0000H       ;Điểm nhập reset.
        LJMP    MAIN        ;Nhảy qua khỏi các vectơ 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 1085µs (cho nữa 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: 15µs),
        DJNZ    R2, $       ;tạo thời gian trễ (cho nữa chu kỳ sau).
        MOV     TL1, #18H   ;Định thời 1085µs (cho nữa 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 trỏ 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 trỏ 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}*=11,0592MHz.*

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     SCON, #42H  ;Chọn chế độ 1 cho port nối
                            ;tiếp, TI=1 để buộc có ngắt
                            ;và gởi 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 tộc 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, cờ 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 x 1,085µs = 8,68µs. Do vậy, với 8333µs dùng để truyền một dữ liệu mà chỉ có 8,68µs dành cho trình phục vu ngắt SPISR. Lệnh **SJMP $** thực thi trong khoảng thời gian 8333µs - 8,68µs = 8324,32µs 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, cấm 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} *= 11,0592MHz*

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, #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 cộng 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} *= 11,0592MHz*

Giải

```
    ORG     0000H       ;Điểm nhập reset.
    LJMP    MAIN        ;Nhảy qua khỏi các vectơ 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 (nữa 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 trỏ 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 $120^OC \pm 5^OC$.*

Giải

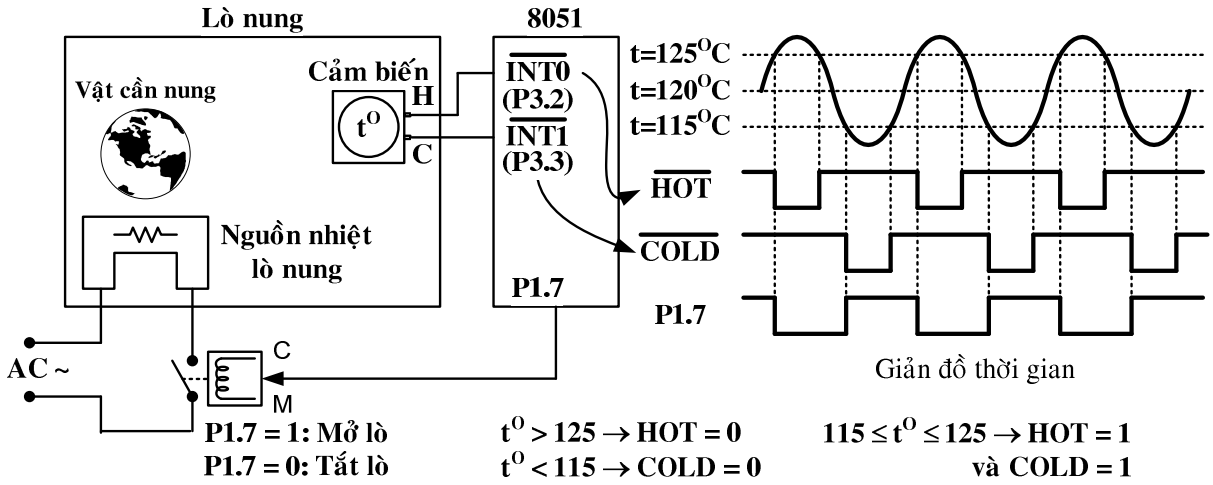
Giả sử ta dùng mạch giao tiếp như hình bên dưới. Cuộn dây điều khiển tắt/mở lò được nối với P1.7 sao cho:

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^OC và ***COLD\ = 0 nếu T < 115^OC***

Chương trình sẽ cho lò hoạt động (*mở lò*) khi T < 115^OC và cho lò ngưng hoạt động (*tắt lò*) khi T > 125^OC.



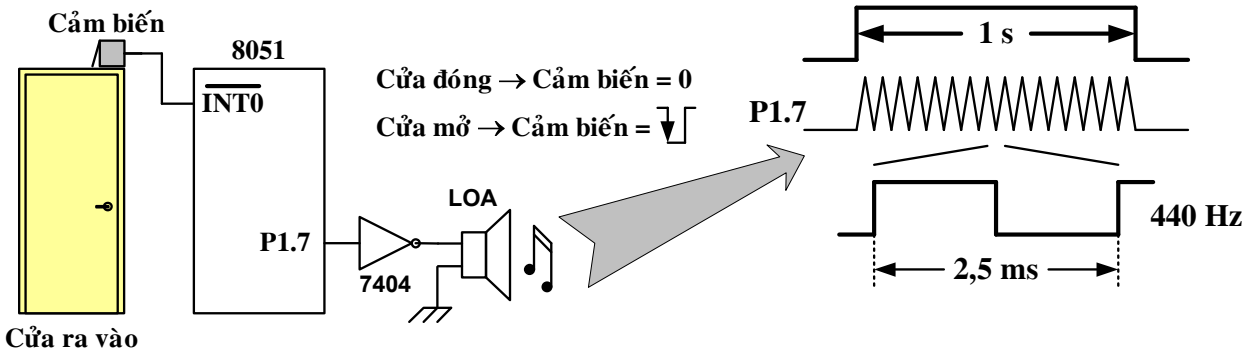

```
        ORG     0000H       ;Điểm nhập reset.
        LJMP    MAIN        ;Nhảy qua khỏi các vectơ ngắt.
        ORG     0003H       ;Điểm nhập ISR ngắt ngoài 0.
EX0ISR:
        CLR     P1.7        ;Tắt lò.
        RETI                ;Kết thúc ISR của ngắt ngoài 0.
        ORG     0013H       ;Điểm 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°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°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°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 vectơ 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 µs = 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 vectơ ngắt, chương trình chính và ba trình phục vụ ngắt. Các vị trí vectơ 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.

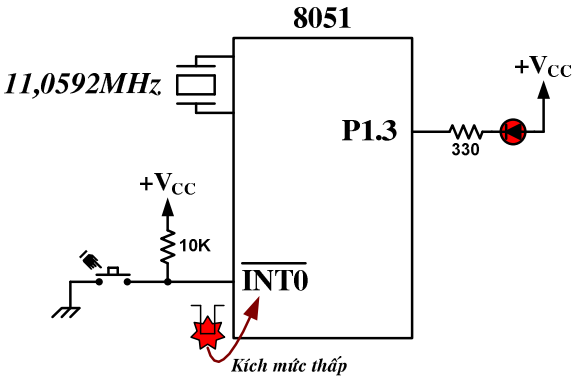
Lệnh ***MOV TMOD,#11H*** xác định chế độ hoạt động của cả hai bộ định thời là chế độ định thời 16 bit. Chỉ có ngắt ngoài 0 được phép bắt đầu (*lệnh **MOV IE,#81H***) do điều kiện cửa mở là điều kiện cần phải có trước khi một ngắt nào đó được chấp thuận. Cuối cùng lệnh ***SJMP $*** đặt chương trình vào vòng lặp "không làm gì". Khi điều kiện cửa mở được phát hiện (*bằng sự chuyển trạng tái 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 x 50000µs = 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 giãm 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 giãm 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.

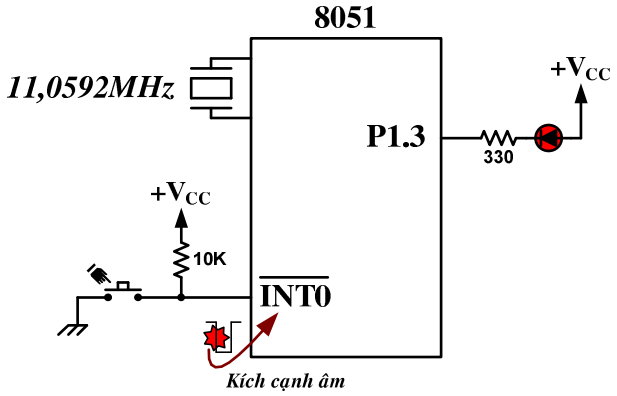
Ví dụ 3: *Ngắt ngoài kích khởi mức thấp - Cho mạch điện như hình vẽ. Chân INT0\ được nối với một công tắc bình thường ở mức cao, mỗi khi chân này có mức thấp (nhấn công tắc) thì điều khiển bật LED (bình thường thì LED tắt). Khi LED đã được bật thì phải sáng trong một khoảng thời gian (vài trặm µs) trước 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.TMachine.
        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 (nhảy tại chỗ).
        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 trặm 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 gởi 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}=11,059MHz.

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}=6MHz.

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.

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

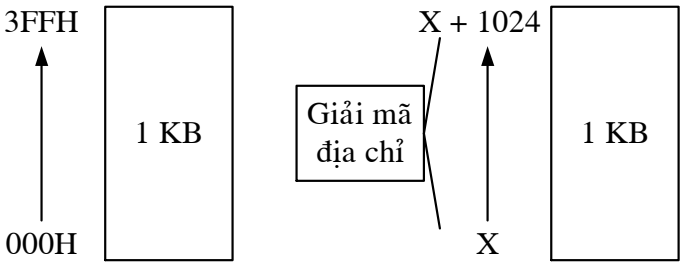
PHỤ LỤC 1

GIẢI MÃ ĐỊA CHỈ

PHỤ LỤC 1
GIẢI MÃ ĐỊA CHỈ

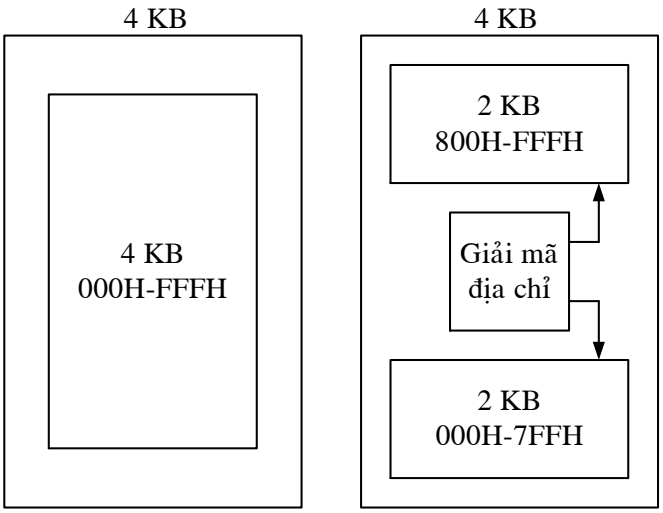
I. TỔNG QUÁT:

- *Trường hợp 1:* Khi cần xác định tầm địa chỉ hoạt động cho một bộ nhớ hoặc ngoại vi → cần có mạch giải mã địa chỉ.



Không dùng giải mã địa chỉ Sử dụng giải mã địa chỉ

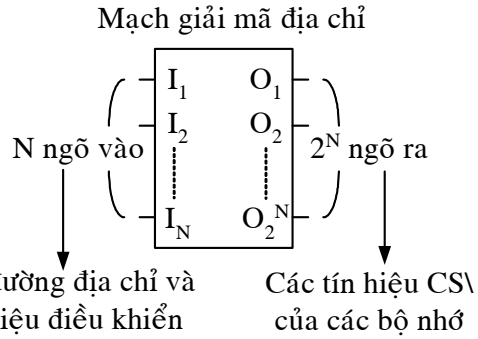
- *Trường hợp 2:* Khi bộ nhớ (ROM hoặc RAM) hoặc ngoại vi có dung lượng lớn được kết hợp từ nhiều bộ nhớ hoặc ngoại vi có dung lượng nhỏ lại với nhau → cần có mạch giải mã địa chỉ → nhằm xác định chính xác địa chỉ của từng bộ nhớ hoặc ngoại vi trên toàn bộ không gian nhớ.



Không dùng giải mã địa chỉ Sử dụng giải mã địa chỉ

II. PHƯƠNG PHÁP THIẾT KẾ MẠCH GIẢI MÃ ĐỊA CHỈ:

- Mạch giải mã địa chỉ :
  - Dùng cổng logic (AND, OR, NOT, …).
  - Dùng vi mạch giải mã (74LS138, 74LS154, …).



- Qui trình thiết kế mạch giải mã địa chỉ:

- ***Bước 1:*** Xác định số lượng vi mạch nhớ cần thiết để có được dung lượng bộ nhớ như yêu cầu.
- ***Bước 2:*** Xác định số đường địa chỉ cần thiết cho từng vi mạch nhớ.
- ***Bước 3:*** Lập bảng đồ nhớ để xác định chính xác vị trí (tầm địa chỉ) của từng vi mạch nhớ trong toàn bộ không gian nhớ.
- ***Bước 4:*** Chọn lựa phương án thiết kế mạch giải mã địa chỉ (dùng cổng logic hay dùng vi mạch giải mã).
- ***Bước 5:*** Thiết kế mạch giải mã theo phương án đã chọn.
  - Lập bảng trạng thái của mạch giải mã.
  - Nếu dùng cổng logic:
    - Dùng bìa K để đơn giản hóa trạng thái các ngõ ra → hàm số Boolean của các ngõ ra mạch giải mã địa chỉ.
    - Sử dụng các cổng logic để thiết kế dựa trên hàm Boolean.
  - Nếu dùng vi mạch giải mã:
    - So sánh bảng trạng thái của vi mạch giải mã với bảng trạng thái của mạch giải mã → thiết kế mạch.
- ***Bước 6:*** Kết nối mạch giải mã vào hệ thống.

1. Bài toán dạng 1:

Thiết kế 4 KB bộ nhớ dùng vi mạch 2732 sao cho có tầm địa chỉ hoạt động từ 000H → FFFH.

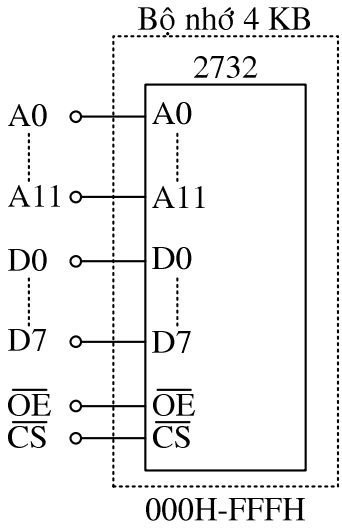
Giải

B1: Số vi mạch sử dụng: 1 vi mạch

B2: Số đường địa chỉ của vi mạch: 12 đường (A0-A11)

→ do chỉ có 1 vi mạch nhớ và địa chỉ hoạt động bắt đầu tại 000H ⇒ không cần dùng mạch giải mã địa chỉ.

B6: Sơ đồ kết nối:



2. Bài toán dạng 2:

Thiết kế 8 KB bộ nhớ dùng vi mạch 6264 sao cho có tầm địa chỉ hoạt động từ 3000H → 4FFFH.

Giải

B1: Số vi mạch sử dụng : 1 vi mạch

B2: Số đường địa chỉ của vi mạch: 13 đường (A0-A12)

→ do địa chỉ hoạt động bắt đầu tại 3000H (khác 0000H) ⇒ cần dùng mạch giải mã địa chỉ.

B3: Lập bảng đồ nhớ:

Phân vùng địa chỉ: IC-6264: 3000H-4FFFH

RAM 6264

A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000H	1
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4FFFH	

Giải mã địa chỉ

B4: Phương án thiết kế: dùng cổng logic

B5: Thiết kế mạch giải mã:

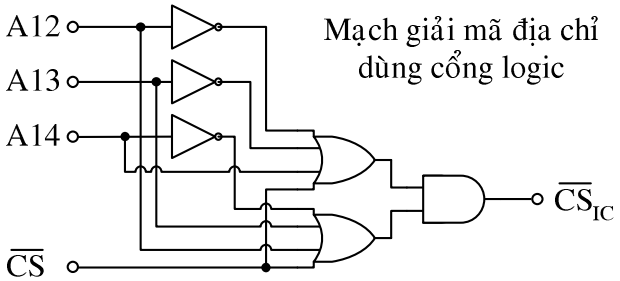
Dùng bìa Karnaugh đơn giản ta được:

$CS_{IC} = (A14+A13\backslash+A12\backslash+CS)(A14\backslash+A13+A12+CS)$

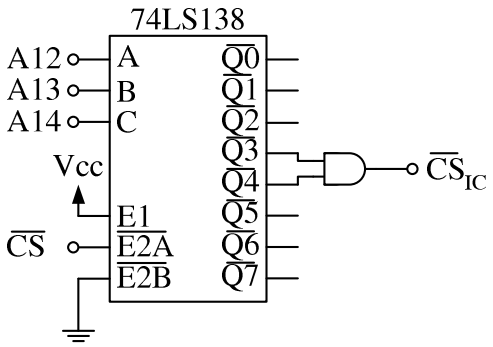
Bảng trạng thái

$\overline{CS}$	A14	A13	A12	$\overline{CS}_{IC}$
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	X	X	X	1

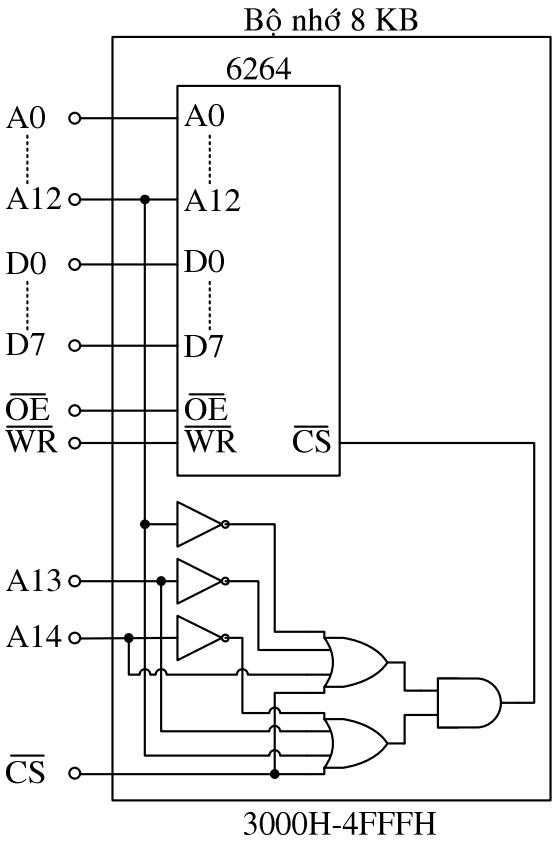
Mạch giải mã địa chỉ dùng cổng logic



Mạch giải mã địa chỉ dùng vi mạch



B6: Sơ đồ kết nối:



3. Bài toán dạng 3:

Thiết kế 32 KB bộ nhớ dùng vi mạch 2764 sao cho có tầm địa chỉ hoạt động từ 0000H → 7FFFH.

Giải

B1: Số vi mạch sử dụng : 4 vi mạch

B2: Số đường địa chỉ của vi mạch: 13 đường (A0-A12)

→ do sử dụng nhiều vi mạch nhớ (4 vi mạch) ⇒ cần dùng mạch giải mã địa chỉ.

B3: Lập bảng đồ nhớ:

Phân vùng địa chỉ:

IC1-2764: 0000H-1FFFH
IC2-2764: 2000H-3FFFH
IC3-2764: 4000H-5FFFH
IC4-2764: 6000H-7FFFH

ROM 2764

A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	1
...			...				...				...				...	
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	2
...			...				...				...				...	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	3
...			...				...				...				...	
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000H	4
...			...				...				...				...	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFH	

Giải mã địa chỉ

B4: Phương án thiết kế: dùng cổng logic

B5: Thiết kế mạch giải mã:

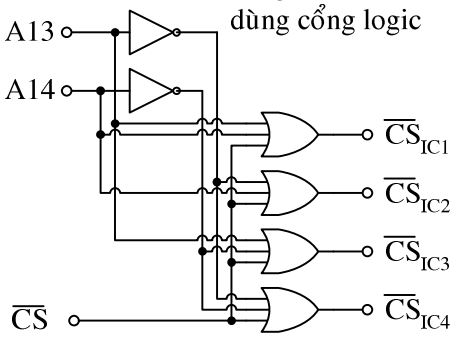
Bảng trạng thái

$\overline{CS}$	A14	A13	$\overline{CS}_{IC1}$	$\overline{CS}_{IC2}$	$\overline{CS}_{IC3}$	$\overline{CS}_{IC4}$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

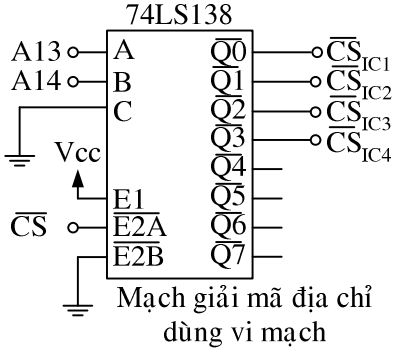
Dùng bìa Karnaugh đơn giản ta được:

$CS_{IC1} = A14+A13+CS$
$CS_{IC2} = A14+A13\+CS$
$CS_{IC3} = A14\+A13+CS$
$CS_{IC4} = A14\+A13\+CS$

Mạch giải mã địa chỉ dùng cổng logic

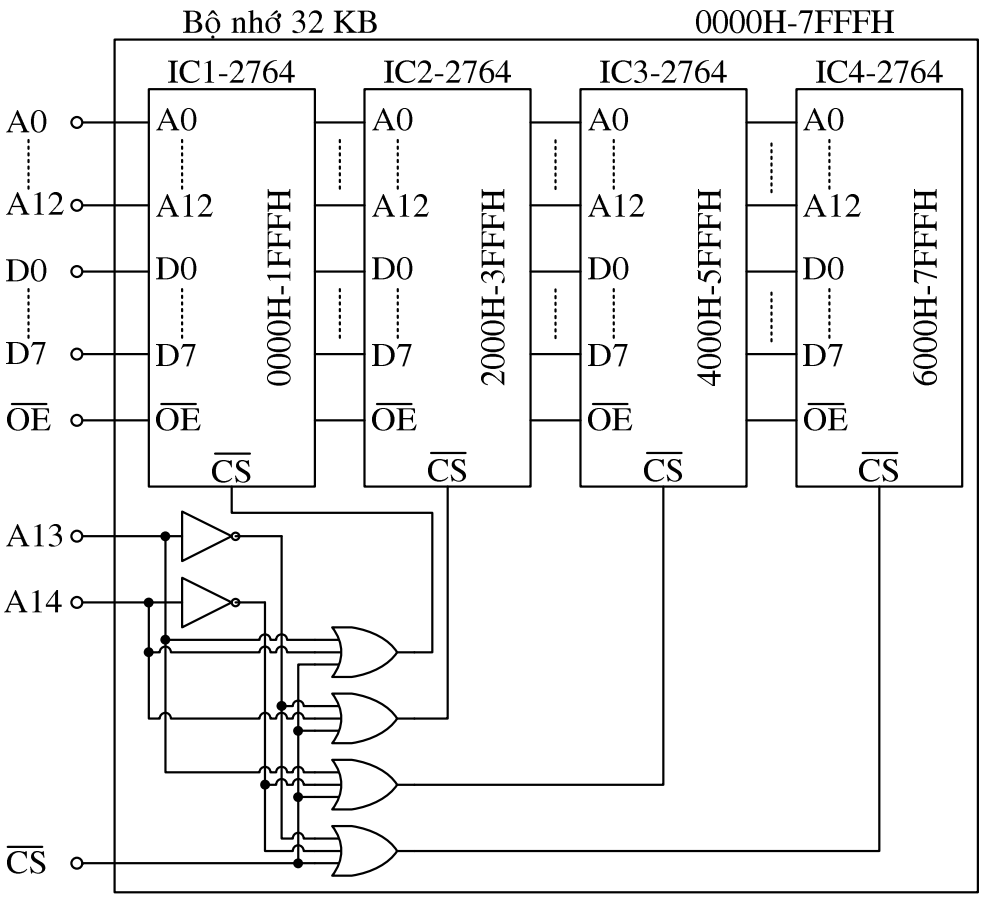


74LS138



Mạch giải mã địa chỉ dùng vi mạch

B6: Sơ đồ kết nối:



4. Bài toán dạng 4:

Thiết kế 48 KB bộ nhớ dùng vi mạch 62128 sao cho có tầm địa chỉ hoạt động từ 4000H → FFFFH.

Giải

B1: Số vi mạch sử dụng : 3 vi mạch

B2: Số đường địa chỉ của vi mạch: 14 đường (A0-A13)

→ do sử dụng nhiều vi mạch nhớ (3 vi mạch) và địa chỉ hoạt động bắt đầu tại 8000H ⇒ cần dùng mạch giải mã địa chỉ.

B3: Lập bảng đồ nhớ:

Phân vùng địa chỉ: IC1-62128: 4000H-7FFFH
IC2-62128: 8000H-BFFFH
IC3-62128: C000H-FFFFH

RAM 62128

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	1
...				...				...				...				...	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFH	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H	2
...				...				...				...				...	
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFFH	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000H	3
...				...				...				...				...	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH	

Giải mã địa chỉ

B4: Phương án thiết kế: dùng cổng logic

B5: Thiết kế mạch giải mã:

Bảng trạng thái

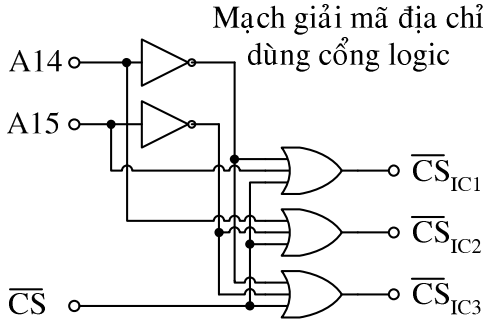
$\overline{CS}$	A15	A14	$\overline{CS}_{IC1}$	$\overline{CS}_{IC2}$	$\overline{CS}_{IC3}$
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	0
1	X	X	1	1	1

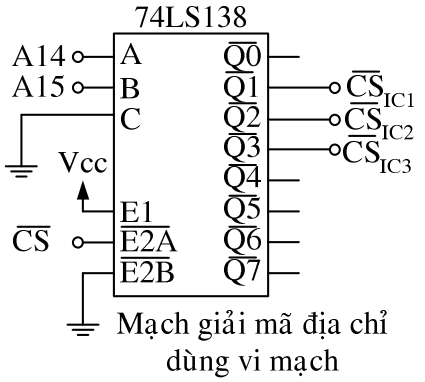
Dùng bìa Karnaugh đơn giản ta được:

$CS_{IC1} = A15+A14\backslash+CS$

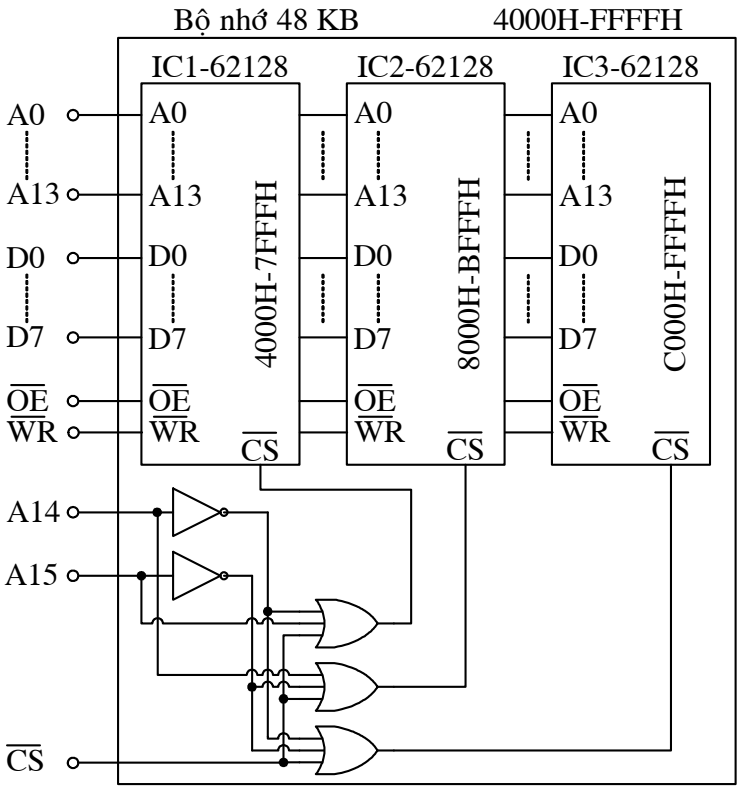
$CS_{IC2} = A15\backslash+A14+CS$

$CS_{IC3} = A15\backslash+A14\backslash+CS$





B6: Sơ đồ kết nối:



5. Bài toán dạng 5:

Thiết kế 64 KB bộ nhớ dùng vi mạch 27128 và 2764 sao cho có tầm địa chỉ hoạt động từ 0000H → FFFFH.

Giải

B1: Số vi mạch sử dụng : 3 vi mạch 27128 và 2 vi mạch 2764

B2: Số đường địa chỉ của vi mạch: 27128 có 14 đường (A0-A13)
2764 có 13 đường (A0-A12)

→ do sử dụng nhiều vi mạch nhớ (5 vi mạch) ⇒ cần dùng mạch giải mã địa chỉ.

B3: Lập bảng đồ nhớ:

Phân vùng địa chỉ:
IC1-2764: 0000H-1FFFH
IC2-2764: 2000H-3FFFH
IC3-27128: 4000H-7FFFH
IC4-27128: 8000H-BFFFH
IC5-27128: C000H-FFFFH

ROM 27128
ROM 2764

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	1
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	2
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	3
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFH	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H	4
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFFH	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000H	5
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH	

Giải mã địa chỉ

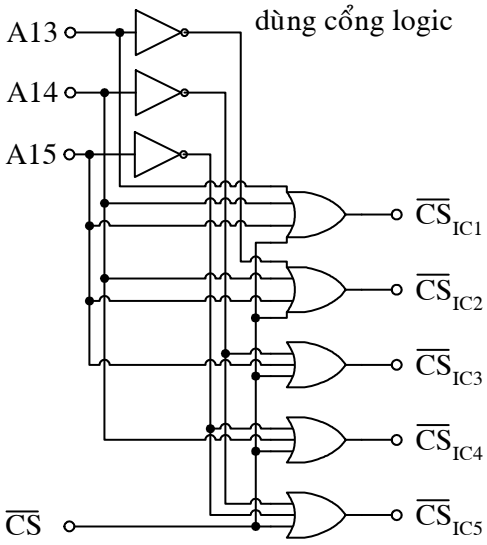
B4: Phương án thiết kế: dùng cổng logic

B5: Thiết kế mạch giải mã:

Bảng trạng thái

$\overline{CS}$	A15	A14	A13	$\overline{CS}_{IC1}$	$\overline{CS}_{IC2}$	$\overline{CS}_{IC3}$	$\overline{CS}_{IC4}$	$\overline{CS}_{IC5}$
0	0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	1	1
0	0	1	0	1	1	0	1	1
0	0	1	1	1	1	0	1	1
0	1	0	0	1	1	1	0	1
0	1	0	1	1	1	1	0	1
0	1	1	0	1	1	1	1	0
0	1	1	1	1	1	1	1	0
1	X	X	X	1	1	1	1	1

Mạch giải mã địa chỉ dùng cổng logic



Dùng bìa Karnaugh đơn giản ta được:

$CS_{IC1} = A15+A14+A13+CS$

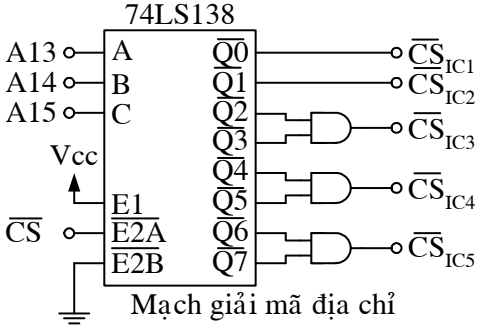
$CS_{IC2} = A15+A14+A13\+CS$

$CS_{IC3} = A15+A14\+CS$

$CS_{IC4} = A15\+A14+CS$

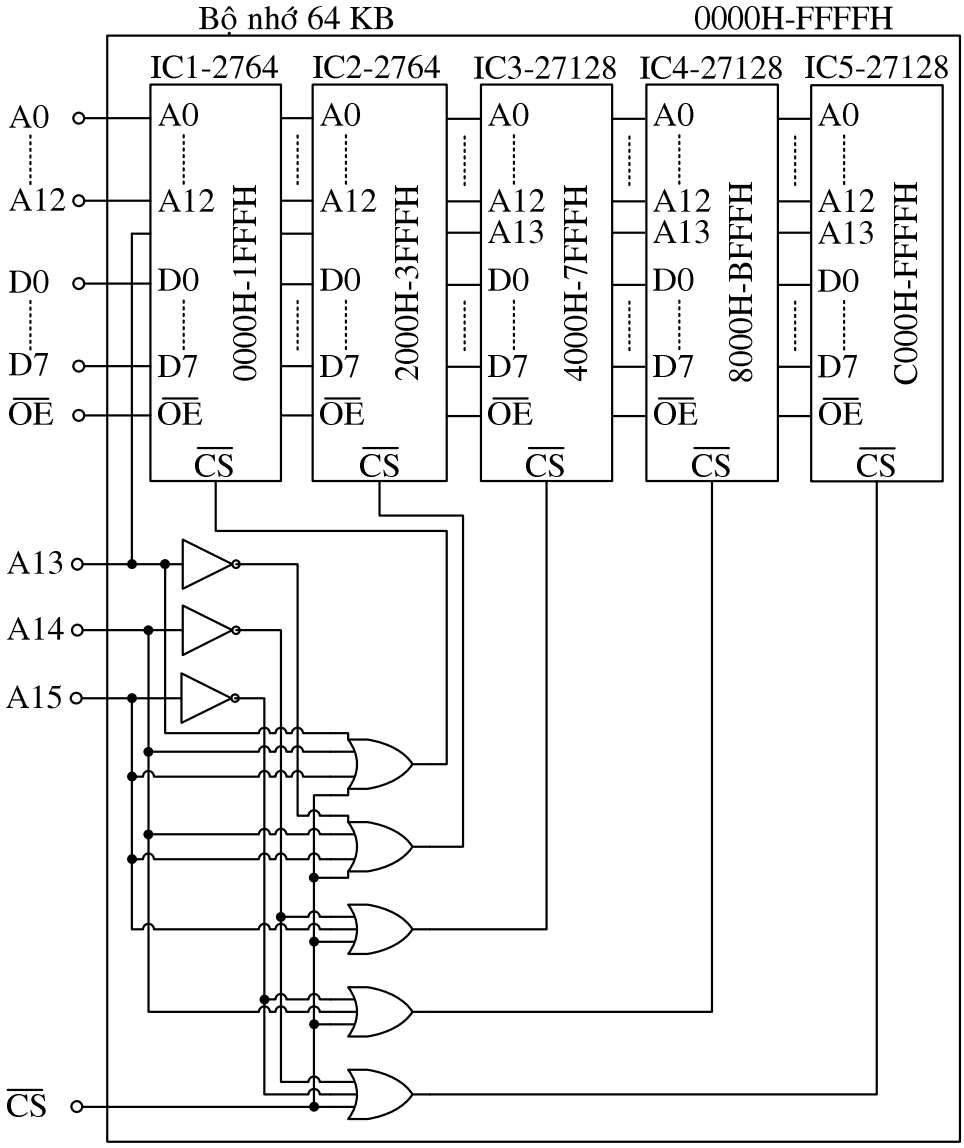
$CS_{IC5} = A15\+A14\+CS$

74LS138



Mạch giải mã địa chỉ dùng vi mạch

B6: Sơ đồ kết nối:



BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

PHỤ LỤC 2

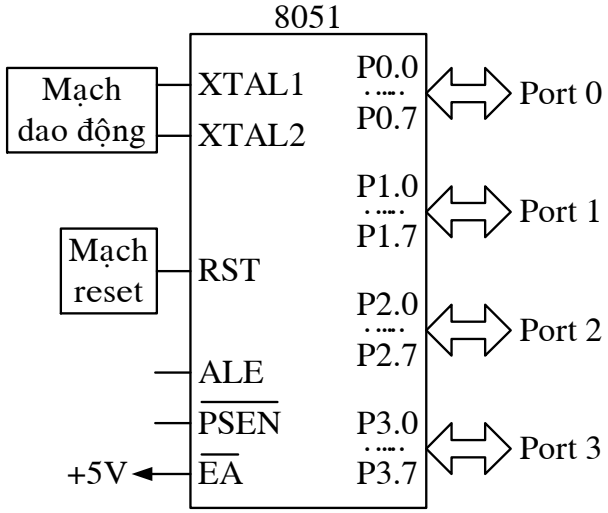
THIẾT KẾ KIT VI ĐIỀU KHIỂN 8051

PHỤ LỤC 2
THIẾT KẾ KIT VI ĐIỀU KHIỂN 8051

I. CẤU TRÚC TỔNG THỂ CỦA 1 KIT VI ĐIỀU KHIỂN:

1. Mini kit (*Bộ vi điều khiển sử dụng bộ nhớ trong*):

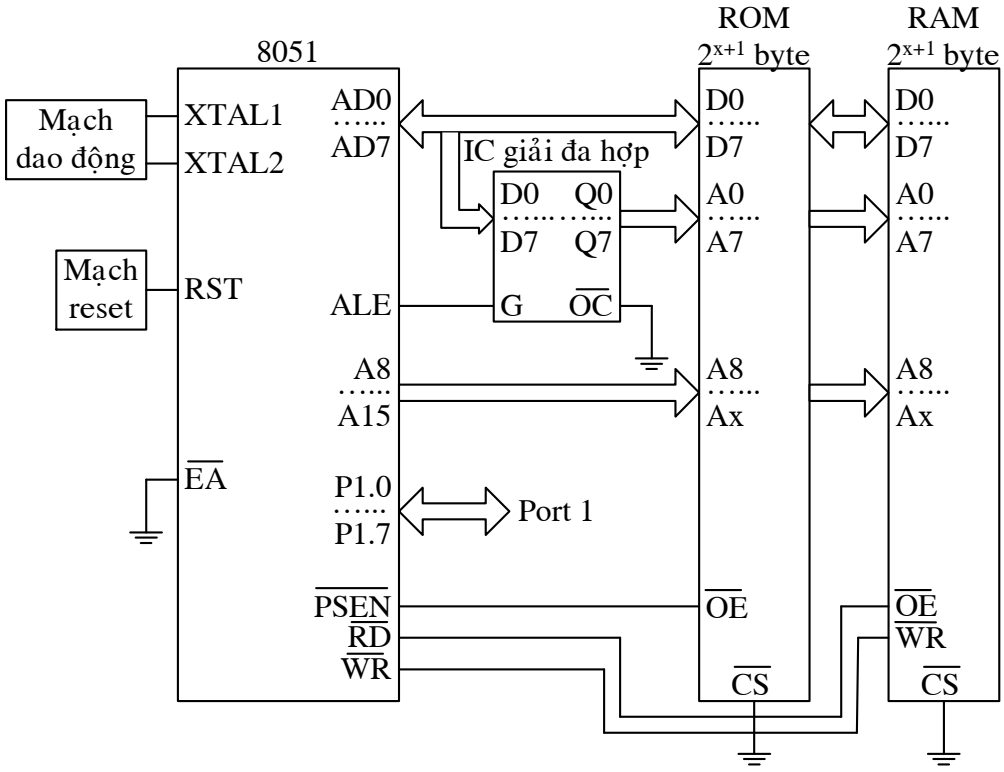
⇒ Cấu trúc đơn giản (*mini kit*) là cấu trúc thường được sử dụng để thiết kế hệ thống điều khiển dùng vi điều khiển 8051 → vì tận dụng được tất cả **4 port** để xuất nhập dữ liệu.



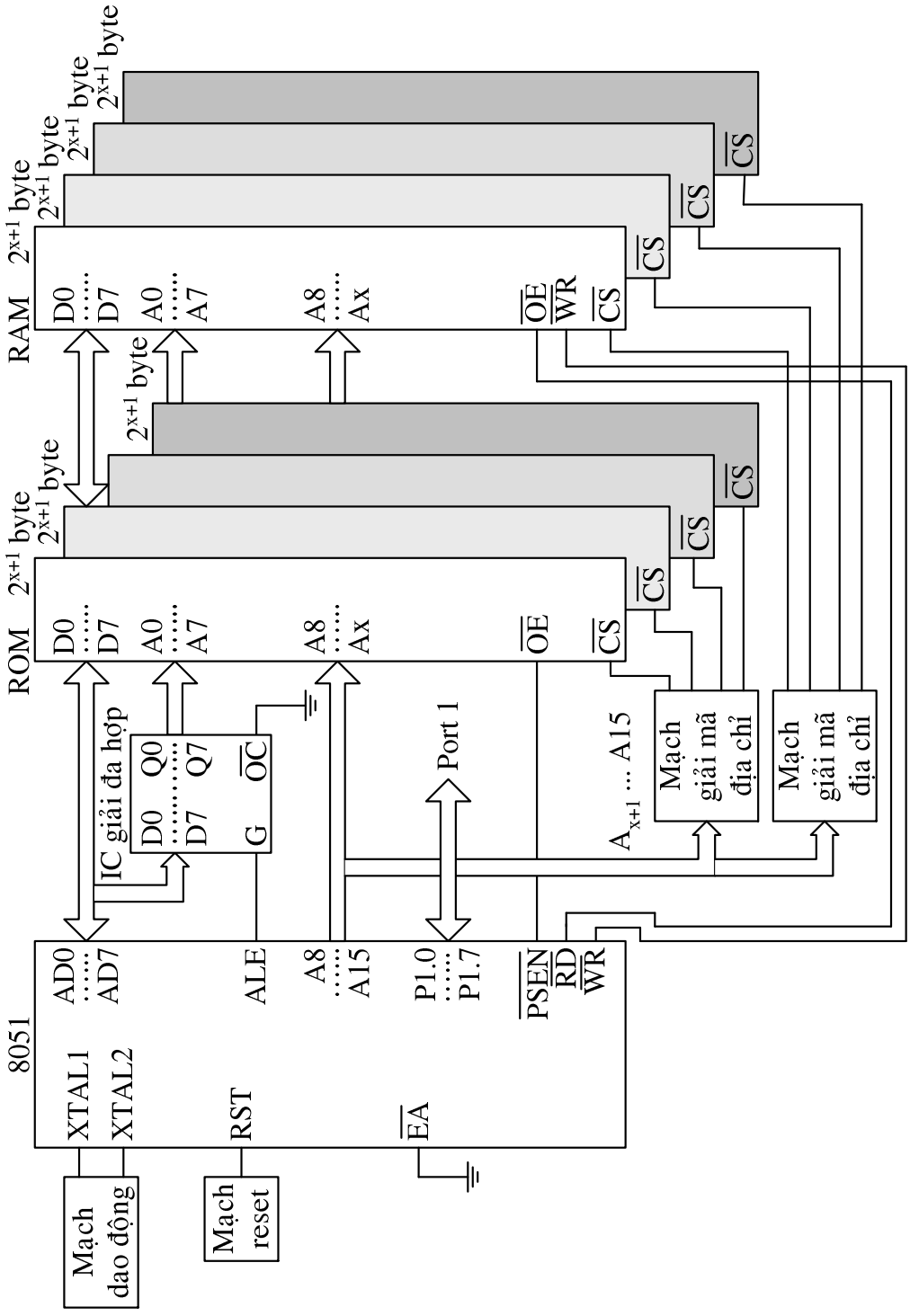
2. Kit mở rộng (*Bộ vi điều khiển sử dụng bộ nhớ ngoài*):

⇒ Cấu trúc mở rộng (*kit mở rộng*) là cấu trúc chỉ được sử dụng để thiết kế hệ thống điều khiển dùng vi điều khiển 8051 khi dung lượng của chương trình (*hoặc dữ liệu*) vượt quá khả năng chứa của bộ nhớ chương trình (*bộ nhớ dữ liệu*) bên trong chip → vì chỉ sử dụng được **1 port** để xuất nhập dữ liệu.

- **Dạng 1:**


8051
ROM
2x+1 byte
RAM
2x+1 byte
Mạch dao động
XTAL1
XTAL2
Mạch reset
RST
EA
AD0
AD7
ALE
A8
A15
P1.0
P1.7
Port 1
PSEN
RD
WR
IC giải đa hợp
D0
Q0
D7
Q7
G
OC
D0
D7
A0
A7
A8
Ax
OE
CS
D0
D7
A0
A7
A8
Ax
OE
WR
CS

- **Dạng 2:**


RAM
ROM
2^x+1 byte
D0 ..... D7
A0 ..... A7
A8 ..... Ax
OE
WR
CS
IC giải đa hợp
D0 Q0
D7 Q7
G OC
Port 1
A_x+1 ... A15
Mạch giải mã địa chỉ
8051
AD0 ..... AD7
ALE
A8 ..... A15
P1.0 ..... P1.7
PSEN
RD
WR
XTAL1
XTAL2
RST
EA
Mạch dao động
Mạch reset

II. PHÂN TÍCH SƠ ĐỒ NGUYÊN LÝ KIT VI ĐIỀU KHIỂN:

(xem hình vẽ được trình bày)

III. PHƯƠNG PHÁP THIẾT KẾ:

- ***Bước 1:*** Căn cứ vào yêu cầu thiết kế và **dung lượng của chương trình điều khiển** để chọn lựa cấu trúc đơn giản (*mini*) hay cấu trúc mở rộng.
- ***Bước 2:***

- Xác định số lượng bộ nhớ (*ROM, RAM*) và thiết bị ngoại vi để chọn lựa việc thiết kế mạch giải mã địa chỉ.
- Xác định tầm địa chỉ hoạt động của các bộ nhớ và thiết bị ngoại vi.

- ***Bước 3:*** Phác thảo sơ đồ khối của hệ thống kit.
- ***Bước 4:*** Thiết kế mạch giải mã địa chỉ cho kit.
- ***Bước 5:*** Thiết kế mạch giải đa hợp (*khi dùng bộ nhớ ngoài*), mạch reset, mạch dao động và các mạch hỗ trợ khác.
- ***Bước 6:*** Kết nối toàn bộ mạch thiết kế thành 1 kit vi điều khiển hoàn chỉnh theo sơ đồ khối.

Lưu ý: Trường hợp cấu trúc đơn giản (*mini*) thì chỉ cần quan tâm các bước sau: 1, 3, 5, 6.

IV. BÀI TẬP THIẾT KẾ:

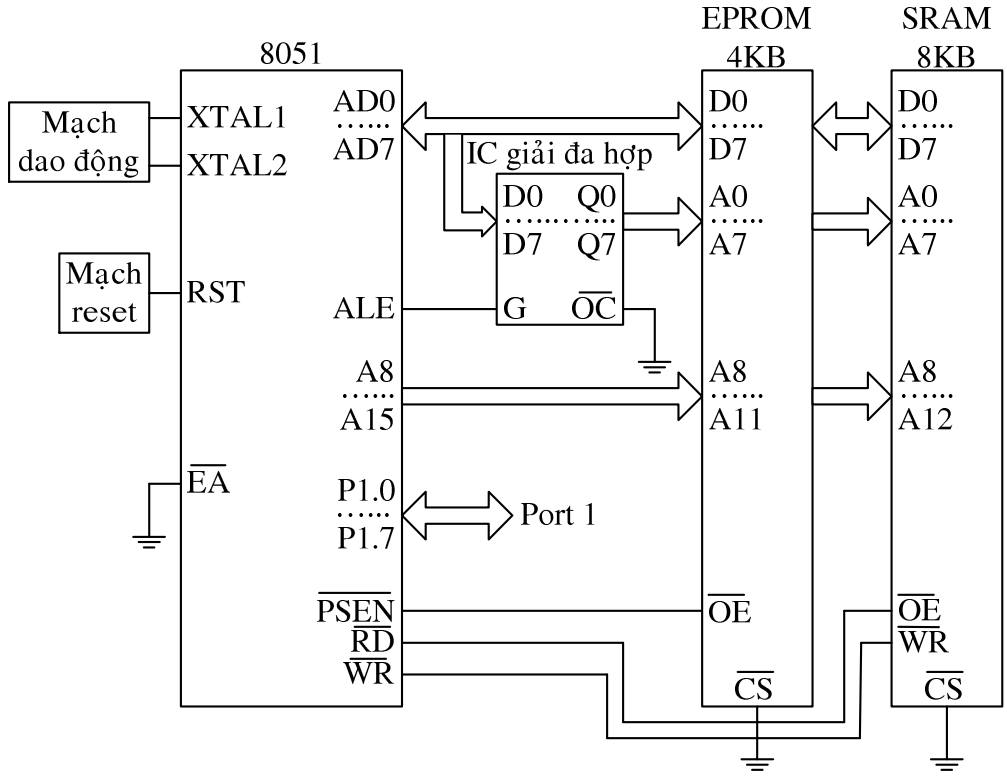
1. **Bài tập 1:**
   Thiết kế một kit vi điều khiển 8051 có các ngoại vi sau: 1 x EPROM 4KB, 1 x SRAM 8KB.

Giải

B1: Chọn lựa cấu trúc mở rộng để thiết kế kit.

B2: Số lượng vi mạch nhớ sử dụng: ROM – 1 x EPROM 4 KB
RAM – 1 x SRAM 8 KB
Phân vùng địa chỉ: EPROM 4KB: 0000H – 0FFFH
SRAM 8KB: 0000H – 1FFFH

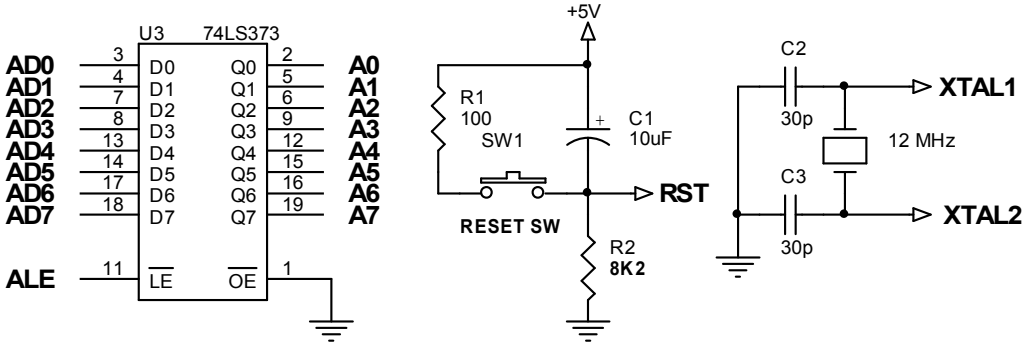
B3: Phác thảo sơ đồ khối của hệ thống:



B4: Thiết kế mạch giải mã địa chỉ:

→ không cần dùng mạch giải mã địa chỉ

B5: Thiết kế mạch giải đa hợp, mạch dao động và mạch reset:



Mạch giải đa hợp **Mạch reset** **Mạch dao động**

B6: Sơ đồ kết nối:

(xem hình vẽ TKK1 được trình bày dưới đây)

2. Bài tập 2:

Thiết kế một kit vi điều khiển 8051 có các ngoại vi sau: 1 x EPROM 27512, 3 x SRAM 6116.

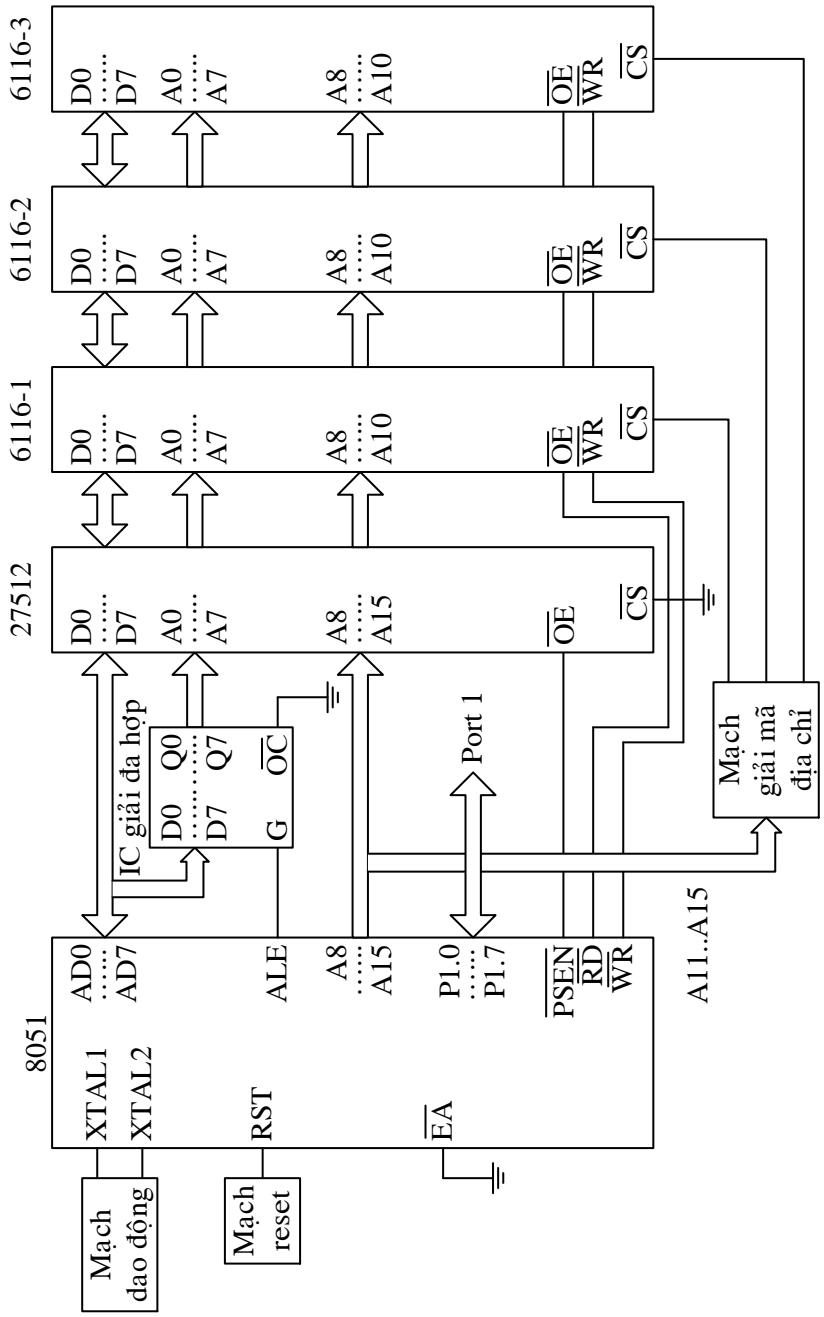
Giải

B1: Chọn lựa cấu trúc mở rộng để thiết kế kit.

B2: Số lượng vi mạch nhớ sử dụng: ROM – 1 x EPROM 27512 (64KB)
RAM – 3 x SRAM 6116 (2KB)

Phân vùng địa chỉ: 27512: 0000H – FFFFH
6116-1: 0000H – 07FFH
6116-2: 0800H – 0FFFH
6116-3: 1000H – 17FFH

B3: Phác thảo sơ đồ khối của hệ thống:



B4: Thiết kế mạch giải mã địa chỉ:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	1
.....				.....				.....				.....				.....	
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800H	2
.....				.....				.....				.....				.....	
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH	
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H	3
.....				.....				.....				.....				.....	
0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH	

6116

Giải mã địa chỉ

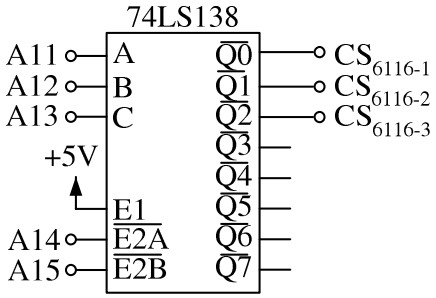
Bảng trạng thái

A15	A14	A13	A12	A11	$\overline{CS}_{6116-1}$	$\overline{CS}_{6116-2}$	$\overline{CS}_{6116-3}$
0	0	0	0	0	0	1	1
0	0	0	0	1	1	0	1
0	0	0	1	0	1	1	0

$CS_{6116-1} = A15+A14+A13+A12+A11$

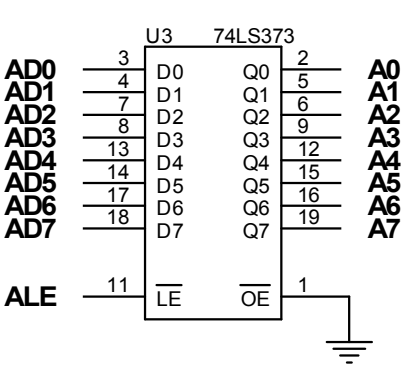
$CS_{6116-2} = A15+A14+A13+A12+A11\backslash$

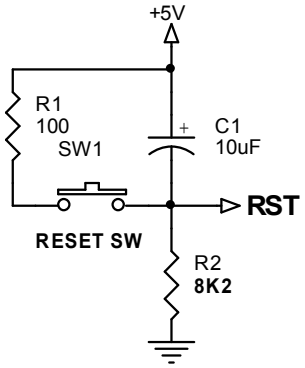
$CS_{6116-3} = A15+A14+A13+A12\backslash+A11$

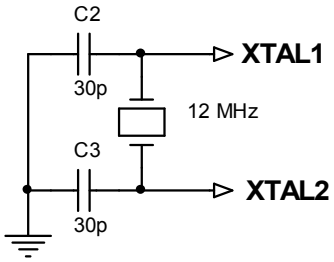


Mạch giải mã địa chỉ dùng vi mạch

B5: Thiết kế mạch giải đa hợp, mạch dao động và mạch reset:







Mạch giải đa hợp **Mạch reset** **Mạch dao động**

B6: Sơ đồ kết nối:

(xem hình vẽ TKK2 được trình bày dưới đây)

3. Bài tập 3:

Thiết kế một kit vi điều khiển 8051 có các ngoại vi sau: 1 x EPROM 8KB (0000H – 1FFFH), 1 x SRAM 8KB (2000H – 3FFFH).

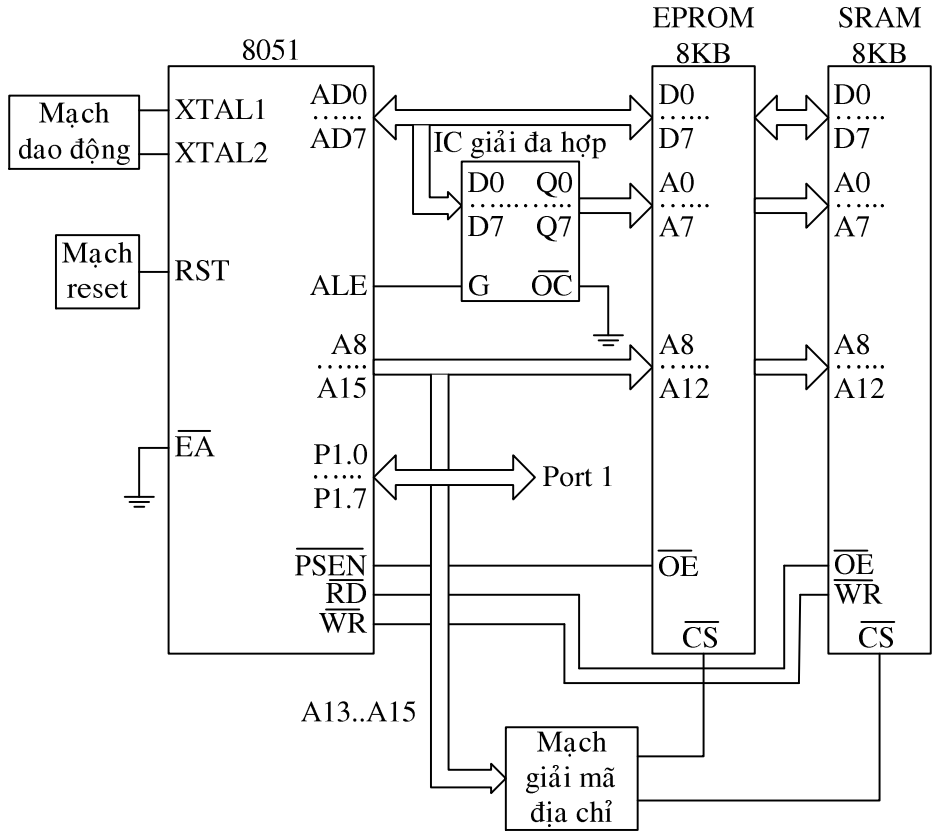
Giải

B1: Chọn lựa cấu trúc mở rộng để thiết kế kit.

B2: Số lượng vi mạch nhớ sử dụng: ROM – 1 x EPROM 8 KB
RAM – 1 x SRAM 8 KB

Phân vùng địa chỉ: EPROM 8KB: 0000H – 1FFFH
SRAM 8KB: 2000H – 3FFFH

B3: Phác thảo sơ đồ khối của hệ thống:



B4: Thiết kế mạch giải mã địa chỉ:

EPROM & SRAM

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ	IC
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM
.....				.....				.....				.....				.....	
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	17FFH	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	RAM
.....				.....				.....				.....				.....	
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	

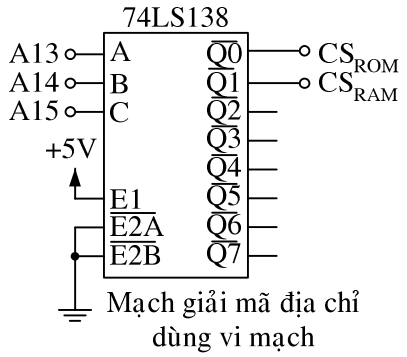
Giải mã địa chỉ

Bảng trạng thái

A15	A14	A13	$\overline{CS}_{ROM}$	$\overline{CS}_{RAM}$
0	0	0	0	1
0	0	1	1	0

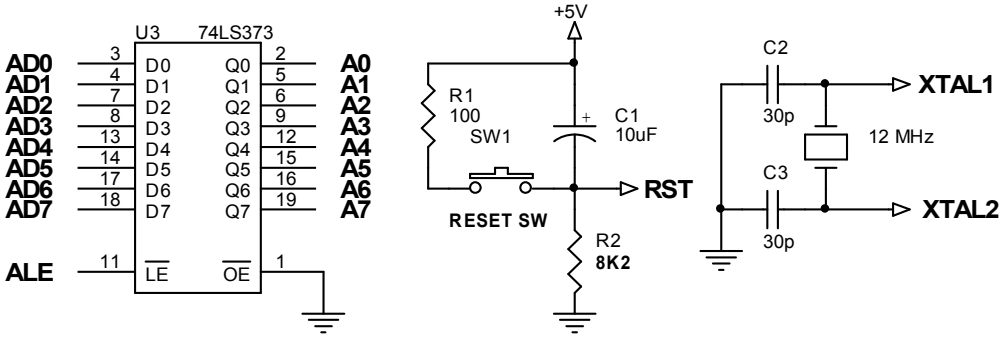
CS_{ROM} = A15+A14+A13

CS_{RAM} = A15+A14+A13\



Mạch giải mã địa chỉ dùng vi mạch

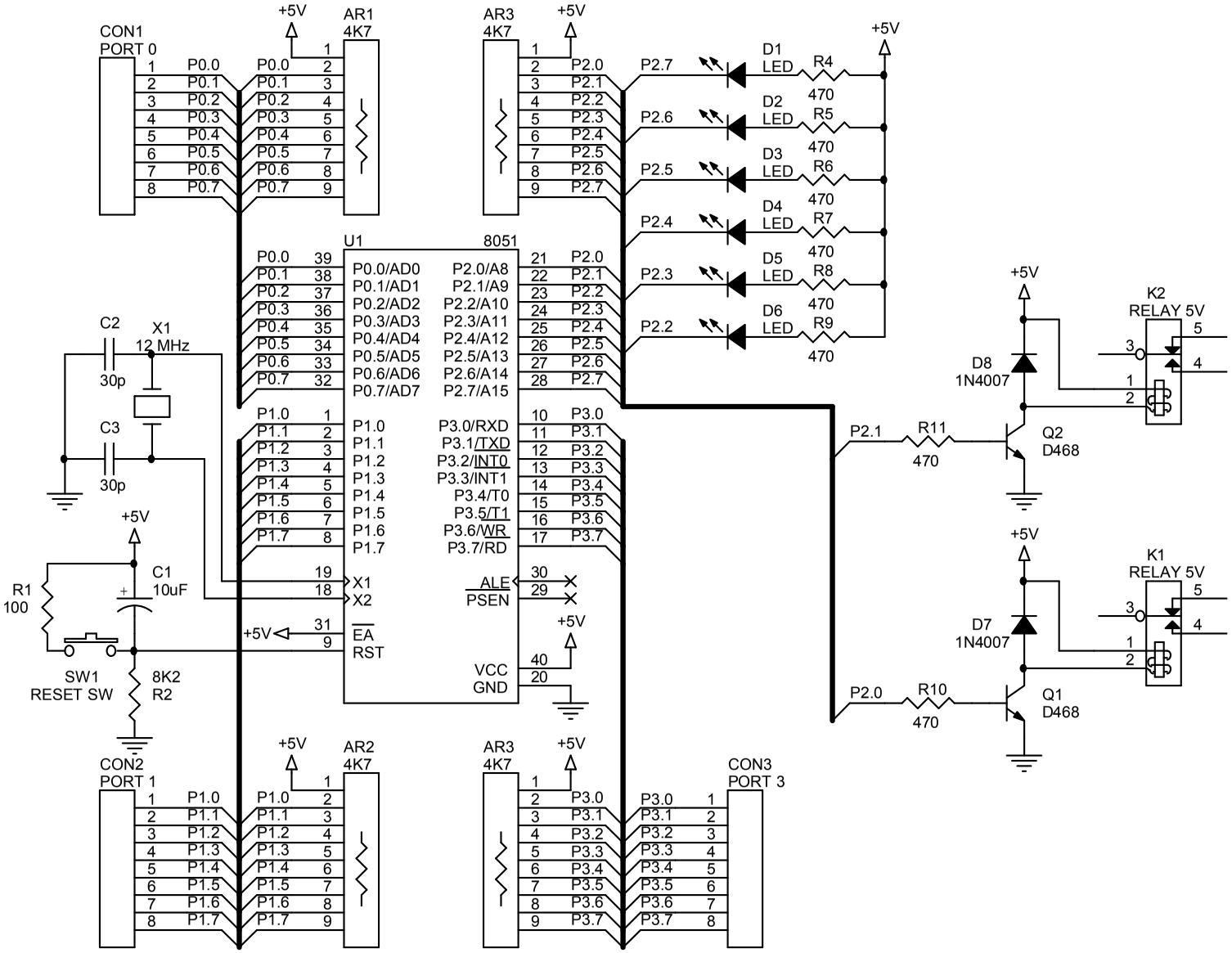
B5: Thiết kế mạch giải đa hợp, mạch dao động và mạch reset:

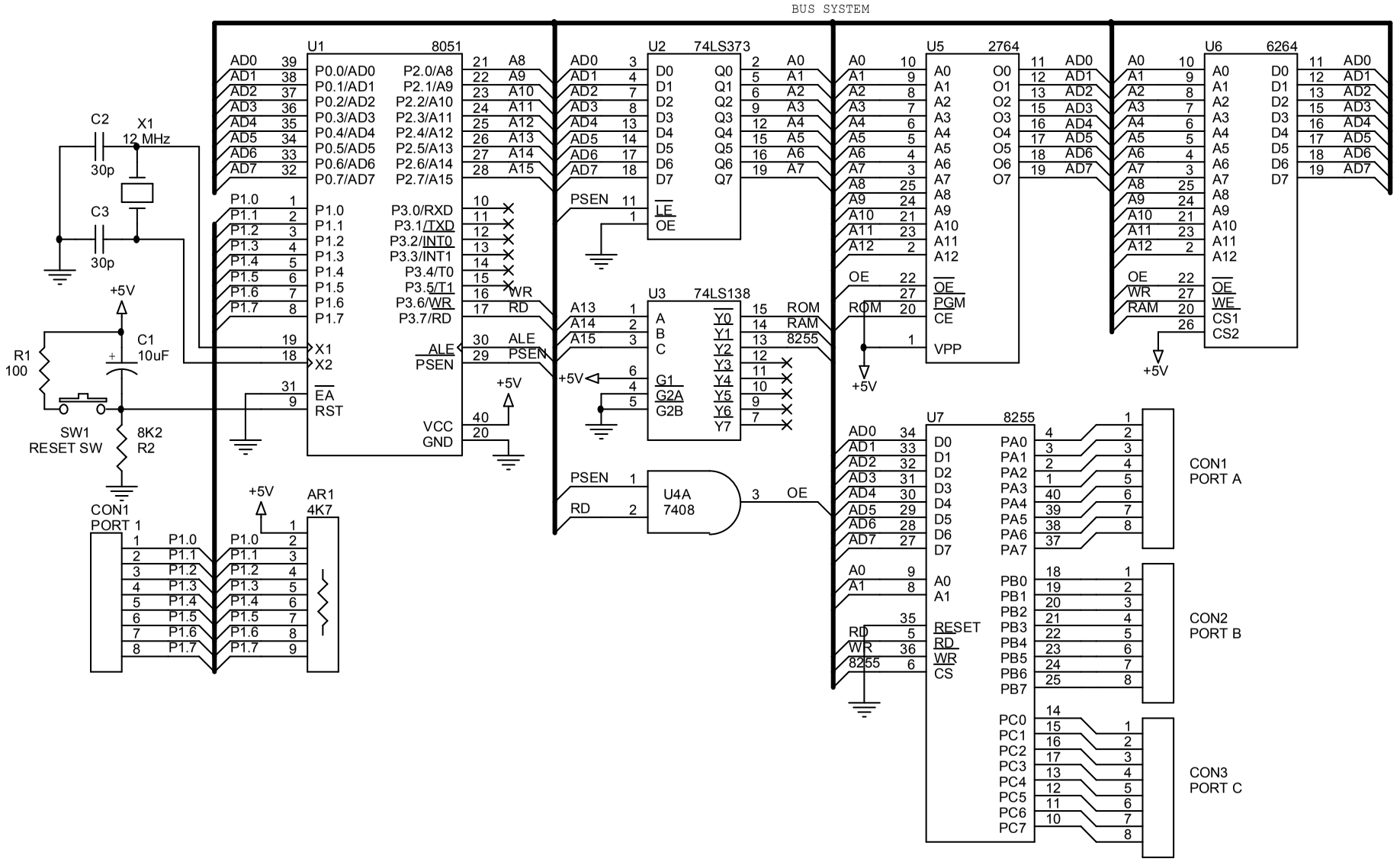


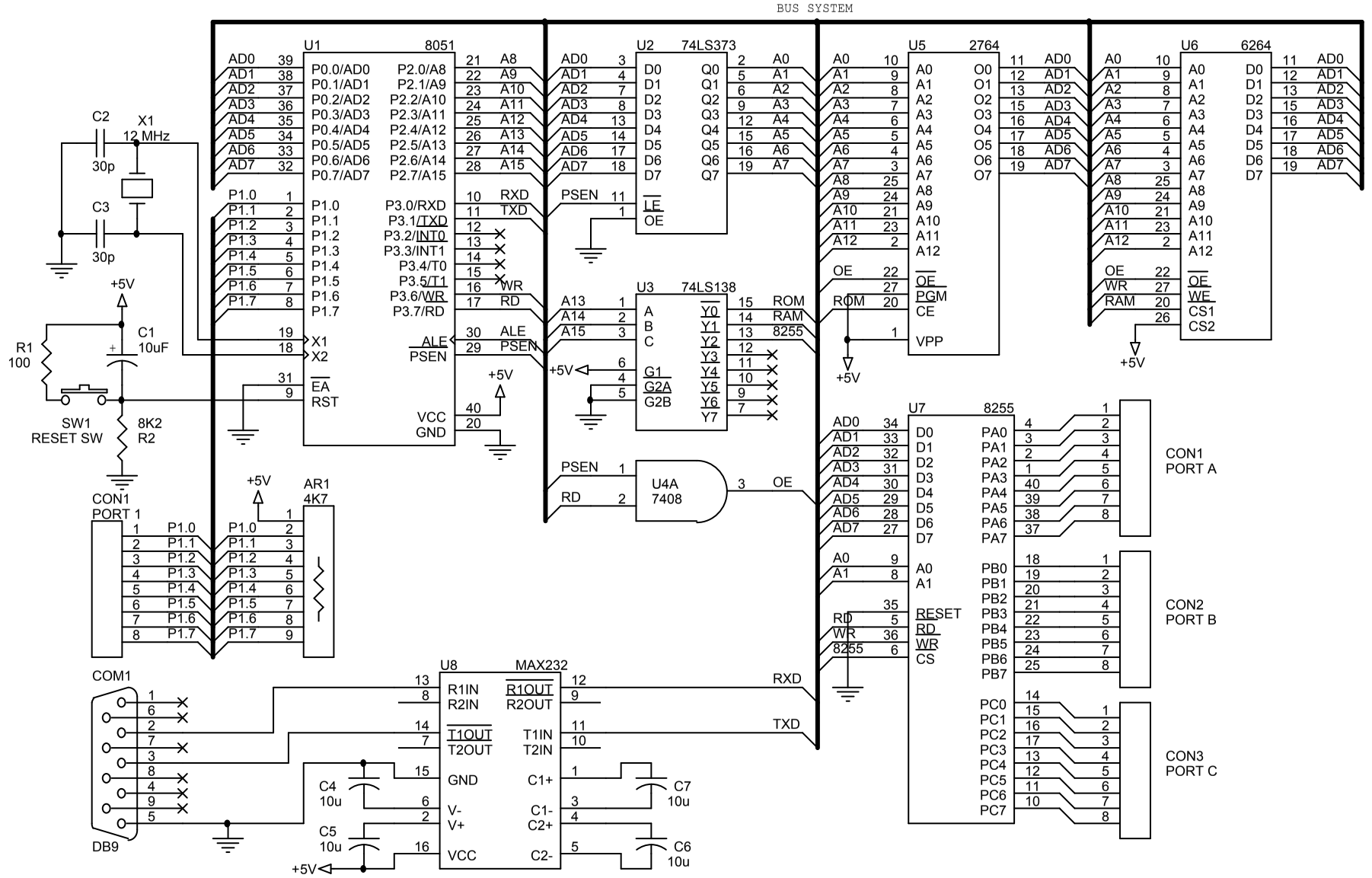
Mạch giải đa hợp **Mạch reset** **Mạch dao động**

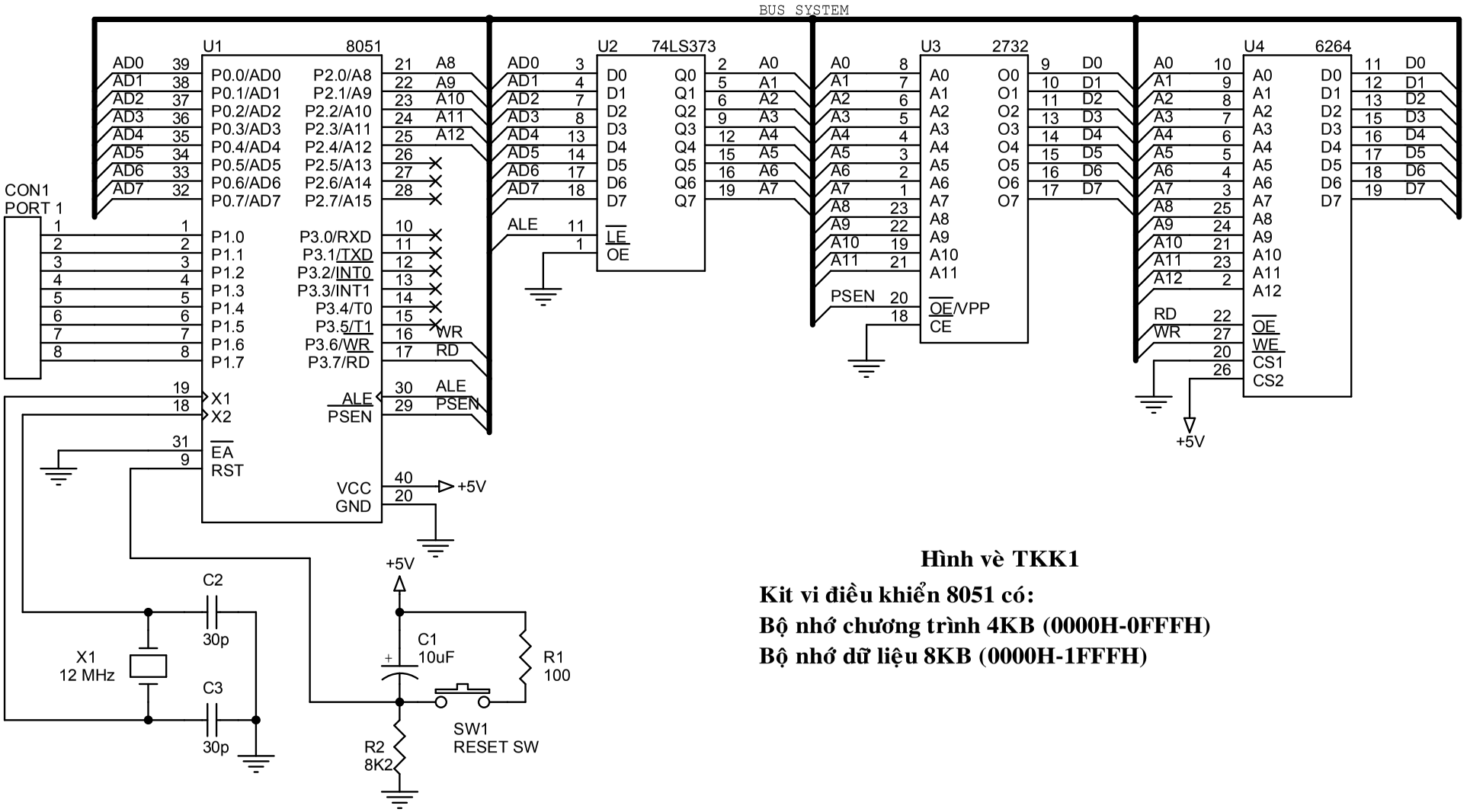
B6: Sơ đồ kết nối:

(xem hình vẽ TKK3 được trình bày dưới đây)


CON1
PORT 0
AR1
4K7
AR3
4K7
U1
8051
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
ALE
PSEN
EA
RST
VCC
GND
X1
X2
C2
30p
C3
30p
X1
12 MHz
C1
10uF
R1
100
SW1
RESET SW
8K2
R2
D1
LED
D2
LED
D3
LED
D4
LED
D5
LED
D6
LED
R4
R5
R6
R7
R8
R9
470
D8
1N4007
D7
1N4007
Q2
D468
Q1
D468
R11
R10
K2
RELAY 5V
K1
RELAY 5V
CON2
PORT 1
AR2
4K7
AR3
4K7
CON3
PORT 3
+5V


BUS SYSTEM
U1 8051
U2 74LS373
U3 74LS138
U4A 7408
U5 2764
U6 6264
U7 8255
X1 12 MHz
C2 30p
C3 30p
C1 10uF
R1 100
SW1 RESET SW
8K2 R2
AR1 4K7
CON1 PORT 1
CON1 PORT A
CON2 PORT B
CON3 PORT C
+5V


BUS SYSTEM
U1 8051
U2 74LS373
U3 74LS138
U4A 7408
U5 2764
U6 6264
U7 8255
U8 MAX232
X1 12 MHz
C1 10uF
C2 30p
C3 30p
R1 100
SW1 RESET SW
8K2 R2
AR1 4K7
CON1 PORT 1
CON1 PORT A
CON2 PORT B
CON3 PORT C
COM1
DB9
C4 10u
C5 10u
C6 10u
C7 10u
+5V
RXD
TXD

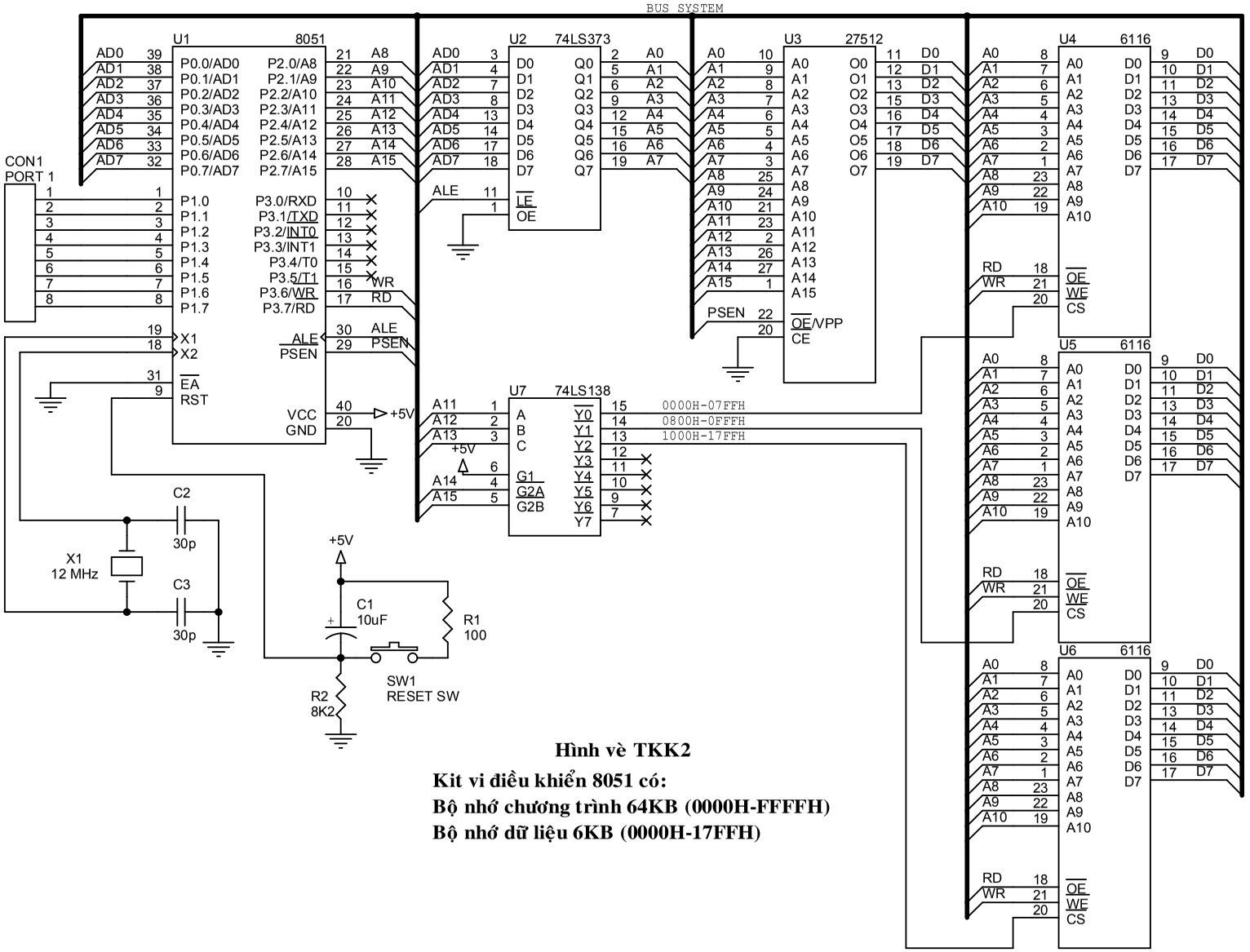


Hình vẽ TKK1

Kit vi điều khiển 8051 có:

Bộ nhớ chương trình 4KB (0000H-0FFFH)

Bộ nhớ dữ liệu 8KB (0000H-1FFFH)

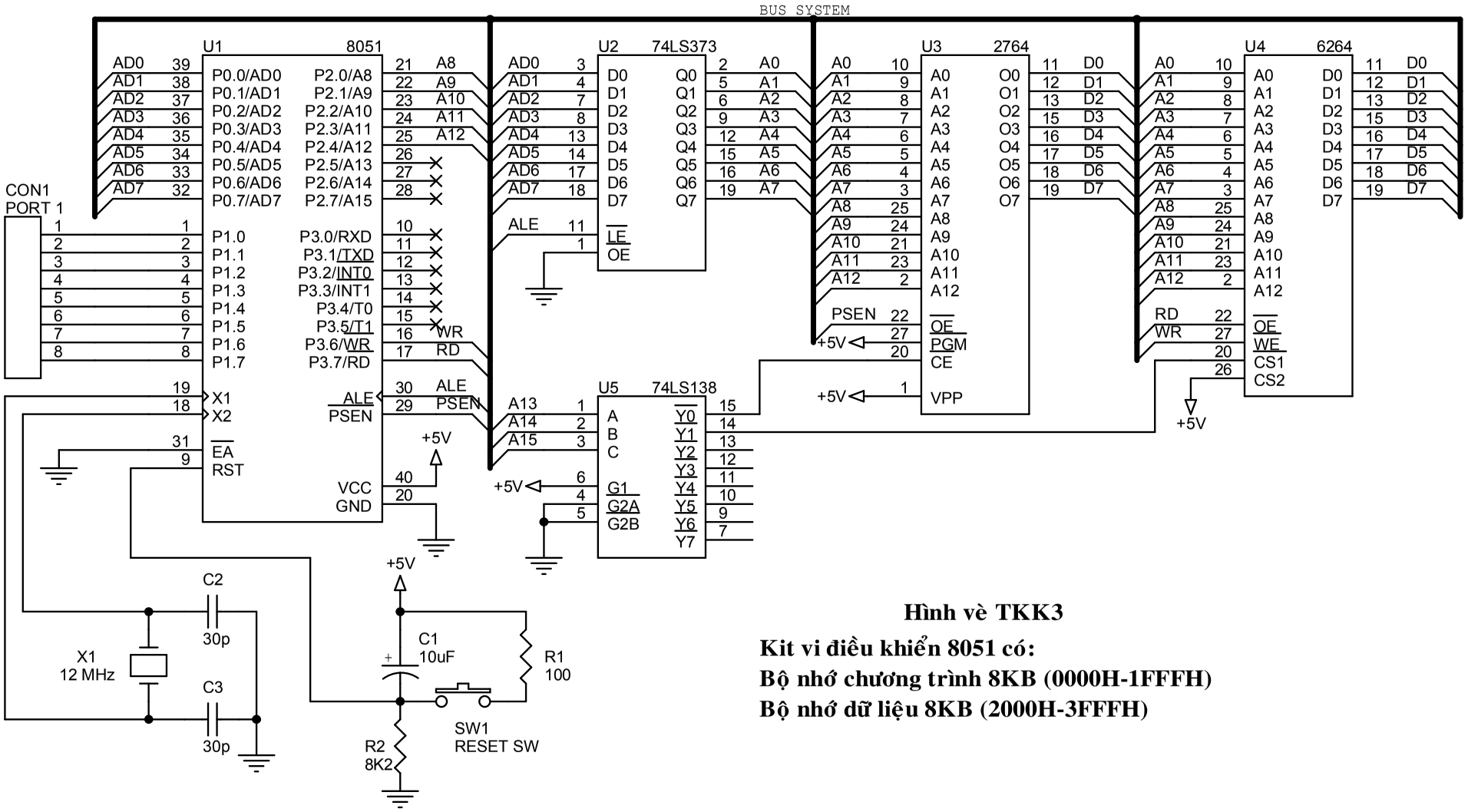


Hình vẽ TKK2

Kit vi điều khiển 8051 có:

Bộ nhớ chương trình 64KB (0000H-FFFFH)

Bộ nhớ dữ liệu 6KB (0000H-17FFH)



Hình vẽ TKK3

Kit vi điều khiển 8051 có:

Bộ nhớ chương trình 8KB (0000H-1FFFH)

Bộ nhớ dữ liệu 8KB (2000H-3FFFH)

BỘ CÔNG NGHIỆP
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ CÔNG NGHIỆP

GIÁO TRÌNH VI XỬ LÝ

PHỤ LỤC 3

THIẾT KẾ NGOẠI VI

PHỤ LỤC 3
THIẾT KẾ NGOẠI VI

I. TỔNG QUÁT:

Các thiết bị ngoại vi hay các thiết bị xuất nhập của bộ vi điều khiển cho ta đường truyền thông giữa bộ vi điều khiển và thế giới bên ngoài. Các thiết bị ngoại vi được kết nối trực tiếp hoặc gián tiếp với bộ vi điều khiển thông qua các cổng xuất-nhập (*các port*) của bộ vi điều khiển.

Các thiết bị ngoại vi luôn luôn chịu sự điều khiển (*nhận dữ liệu*) và sự kiểm tra (*cung cấp dữ liệu*) của bộ vi điều khiển. Từ đó tạo nên khả năng giao tiếp giữa vi điều khiển với thế giới bên ngoài.

Phân loại ngoại vi: ngoại vi điều khiển logic và ngoại vi điều khiển công suất.

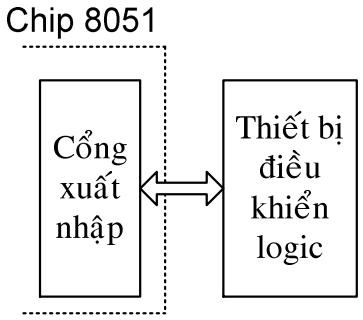
Chú ý: một số thông số kỹ thuật cần chú ý của chip vi điều khiển 8051 khi thiết kế ngoại vi:

V_{OL} (*Output Low Voltage*) = 0.45 V
V_{OH} (*Output High Voltage*) = 4.5 V
V_{IL} (*Input Low Voltage*) = 0 V … 1 V
V_{IH} (*Input High Voltage*) = 2V ... 5 V
V_{CC} (*Operating Voltage*) = 5 V … 6 V
$I_{O(MAX)}$ (*DC Output Current*) = 25 mA (*Port 0*)
$I_{O(MAX)}$ (*DC Output Current*) = 15 mA (*Port 1, 2, 3*)

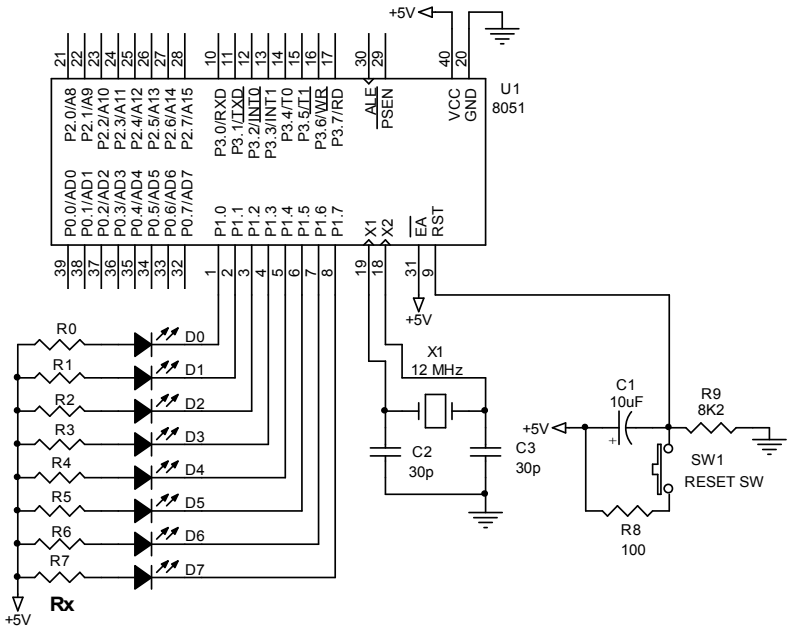
II. NGOẠI VI ĐIỀU KHIỂN LOGIC:

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

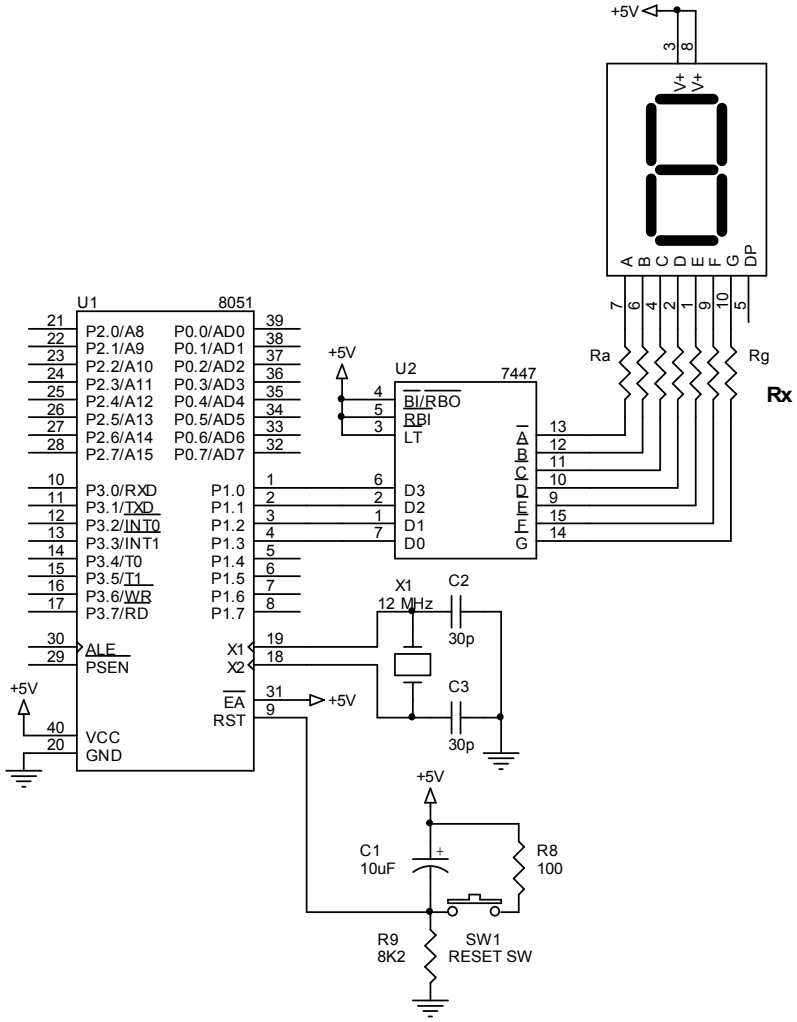
Ví dụ: các vi mạch số (*TTL, CMOS*), LED, các bộ chuyển đổi tín hiệu (*ADC, DAC*), bàn phím, …



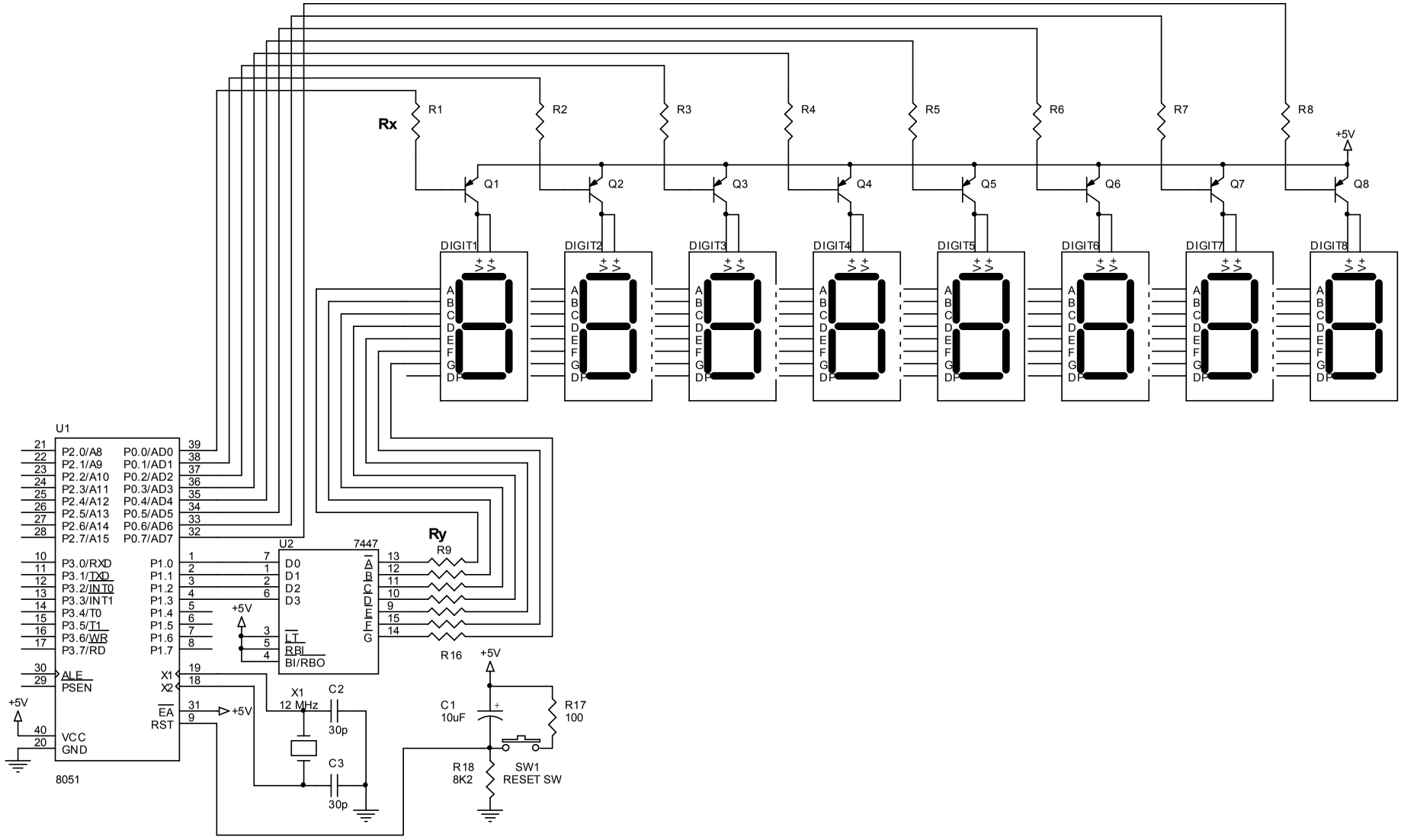
1. Giao tiếp với LED:



2. Giao tiếp với LED 7 đoạn (chế độ hiển thị tĩnh):



- *Ưu điểm:* cấu trúc và nguyên tắc điều khiển đơn giản.
- *Nhược điểm:* Số lượng Led 7 đoạn kết nối với vi điều khiển không nhiều và mỗi Led 7 đoạn phải có một vi mạch giải mã; tiêu tốn năng lượng rất nhiều (*vì muốn Led 7 đoạn sáng phải có dòng điện liên tục đi qua*).


Rx
R1
R2
R3
R4
R5
R6
R7
R8
+5V
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
DIGIT1
DIGIT2
DIGIT3
DIGIT4
DIGIT5
DIGIT6
DIGIT7
DIGIT8
A
B
C
D
E
F
G
DP
U1
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
ALE
PSEN
EA
RST
VCC
GND
X1
X2
8051
U2
7447
D0
D1
D2
D3
LT
RBI
BI/RBO
Ry
R9
X1
12 MHz
C2
30p
C3
30p
R16
C1
10uF
R17
100
R18
8K2
SW1
RESET SW

3. Giao tiếp với LED 7 đoạn (*chế độ hiển thị động*):

(Xem hình vẽ)

- *Ưu điểm:* Tiêu tốn năng lượng giãm rất nhiều so với chế độ tĩnh (*vì các Led 7 đoạn không được thắp sáng liên tục mà luân phiên sáng theo 1 chu kỳ*); số lượng Led 7 đoạn kết nối với vi điều khiển nhiều; chỉ cần 1 vi mạch giải mã cho tất cả các Led 7 đoạn.
- *Nhược điểm:* cấu trúc và nguyên tắc điều khiển phức tạp.

Chú ý:

- Tại một thời điểm chỉ có 1 Led sáng, các Led còn lại đều tắt.
- Để mắt người thấy cả con số (*8 digit*) sáng lên cùng một lúc tại 1 thời điểm:
  Số lần hiển thị hết 8 digit / 1 giây : 40 lần – 200 lần
  TNUMBER = 5 ms – 25 ms

4. Giao tiếp với vi mạch 8255:

Vi mạch 8255 là một vi mạch giao tiếp song song được lập trình. Được sử dụng để giao tiếp song song giữa vi điều khiển và thiết bị điều khiển bên ngoài.

Sơ đồ chân và chức năng các chân của vi mạch 8255

(xem hình vẽ)

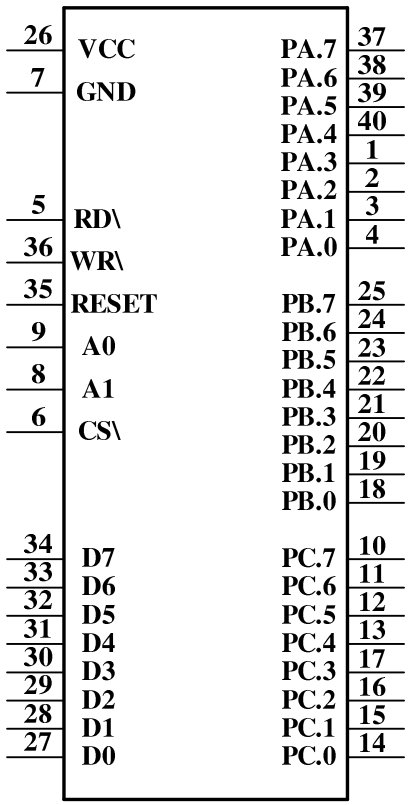
⇒ Để vi mạch 8255 hoạt động được thì trước khi sử dụng 8255 ta cần phải xác định cấu hình cho 8255 → khởi động 8255 → nạp cho CWR một giá trị cụ thể.

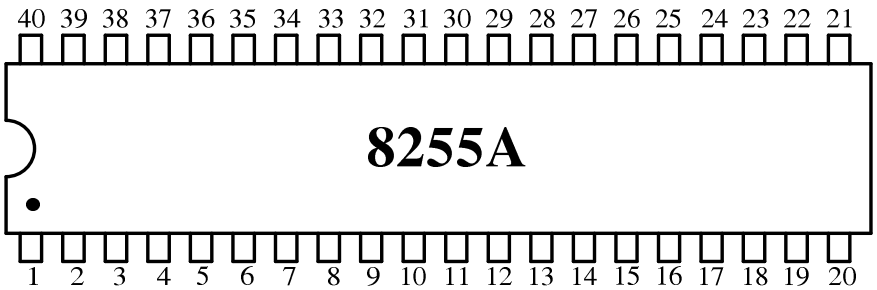
Cấu trúc từ điều khiển của 8255:

(xem hình vẽ)

Mode 0: chế độ port xuất-nhập dữ liệu cơ bản.
Mode 1: chế độ port xuất-nhập dữ liệu có chốt (*Strobed I/O*).
Mode 2: chế độ port xuất-nhập dữ liệu 2 chiều có chốt (*Strobed Bi-directional I/O*).

SƠ ĐỒ CHÂN CỦA IC 8255A





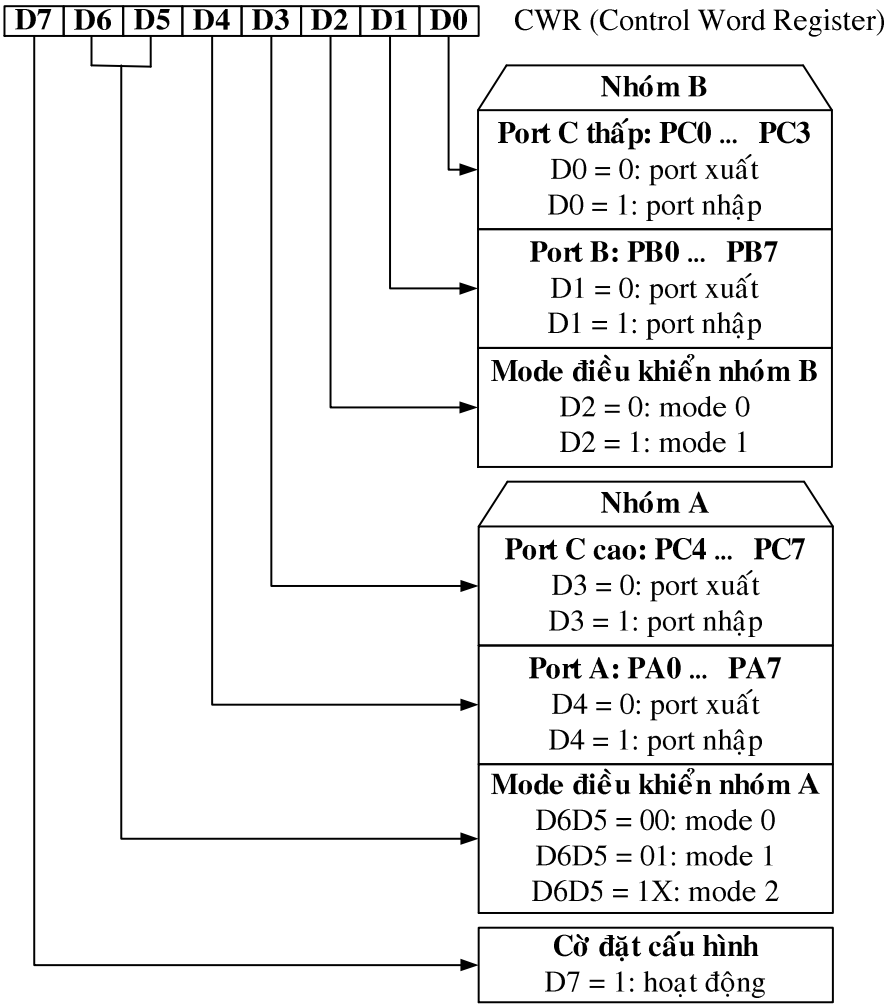
IC 8255 giao tiếp với vi điều khiển thông qua 3 bus:

- Bus dữ liệu: D0 – D7.
- Bus địa chỉ: A0 – A7.
- Bus điều khiển: RD\, WR\, CS\, RESET.

Chức năng các chân của IC giao tiếp ngoại vi 8255:

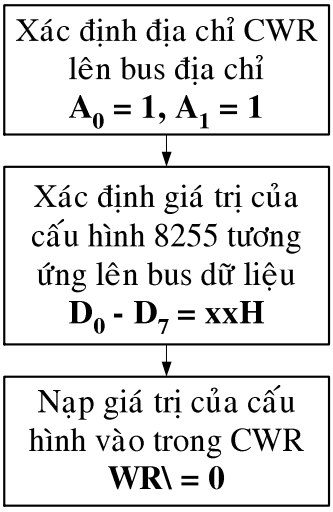
- **D0-D7**: các đường dữ liệu truyền thông tin về mã lệnh, trạng thái và dữ liệu giữa vi điều khiển và IC 8255.
- **PA0-PA7, PB0-PB7, PC0-PC7**: các cổng xuất-nhập dữ liệu giữa 8255 với các thiết bị ngoại vi. Các cổng này có thể là cổng xuất (Output) hay cổng nhập (Input) tùy thuộc vào lệnh điều khiển từ vi điều khiển đưa đến 8255, lệnh điều khiển được chứa trong thanh ghi từ điều khiển (CWR: Control Word Register) để định cấu hình cho các cổng xuất-nhập.
- **RESET**: thiết lập lại trạng thái ban đầu cho IC 8255.
- **CS**: dùng để lựa chọn IC 8255 khi vi điều khiển giao tiếp với nhiều IC 8255.
- **A0, A1**: dùng để chọn lựa các cổng xuất-nhập (PA, PB, PC) hoặc thanh ghi từ điều khiển (CWR) giao tiếp với vi điều khiển.

A1	A0	D0 – D7	Bus dữ liệu
0	0	PA0 – PA7	Port A
0	1	PB0 – PB7	Port B
1	0	PC0 – PC7	Port C
1	1	CWR0 – CWR7	Thanh ghi từ điều khiển



⇒ *Ví dụ:* Xác định giá trị nạp cho CWR để xác định cấu hình hoạt động của 8255 như sau:
Port A: xuất – Port B: xuất – Port C: xuất → CWR = 80H
Port A: nhập – Port B: xuất – Port C: nhập → CWR = 99H
Port A: xuất – Port B: nhập – Port CL: nhập, CH: xuất → CWR = 83H

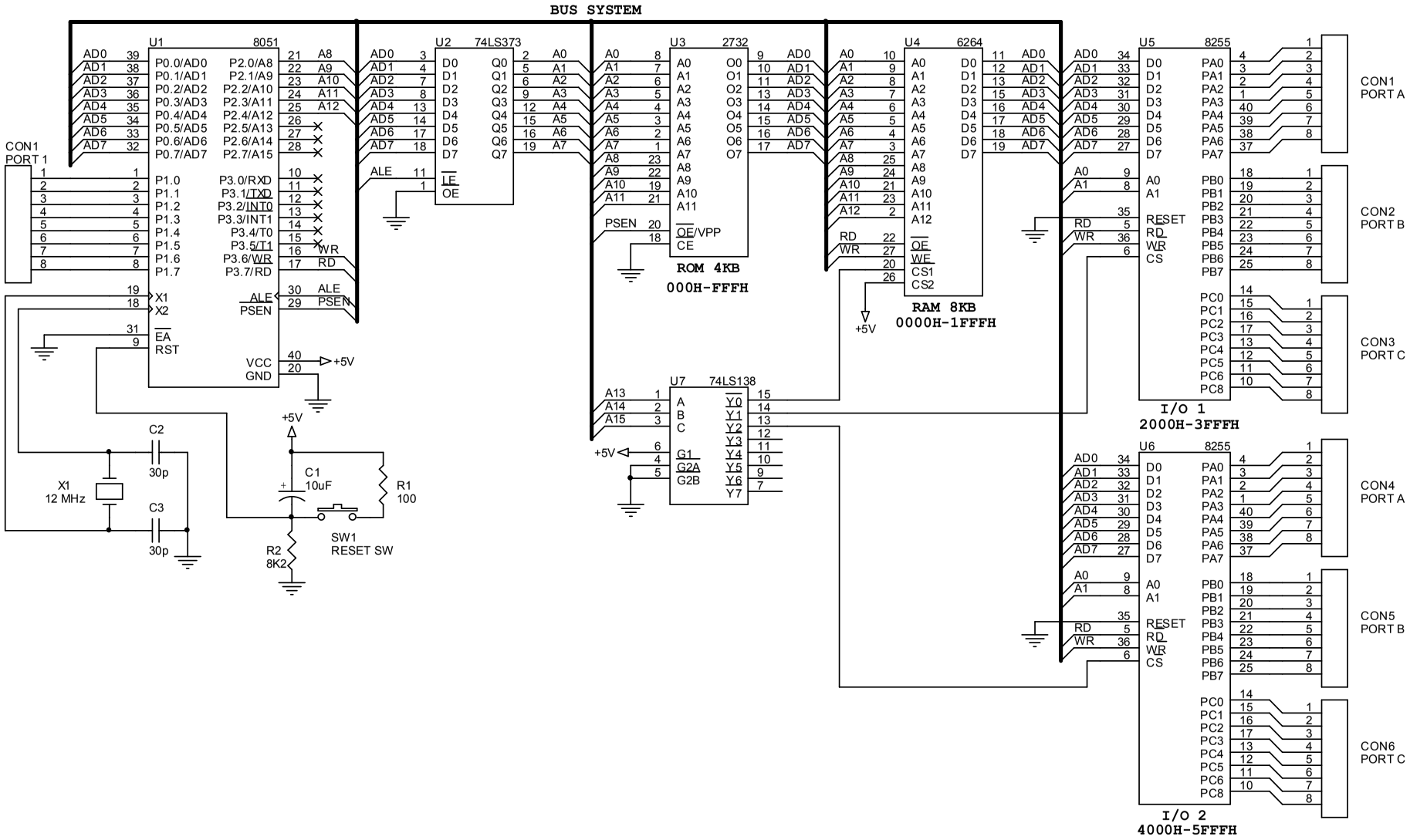
Qui trình khởi động vi mạch 8255 (nạp giá trị cấu hình 8255 cho CWR):



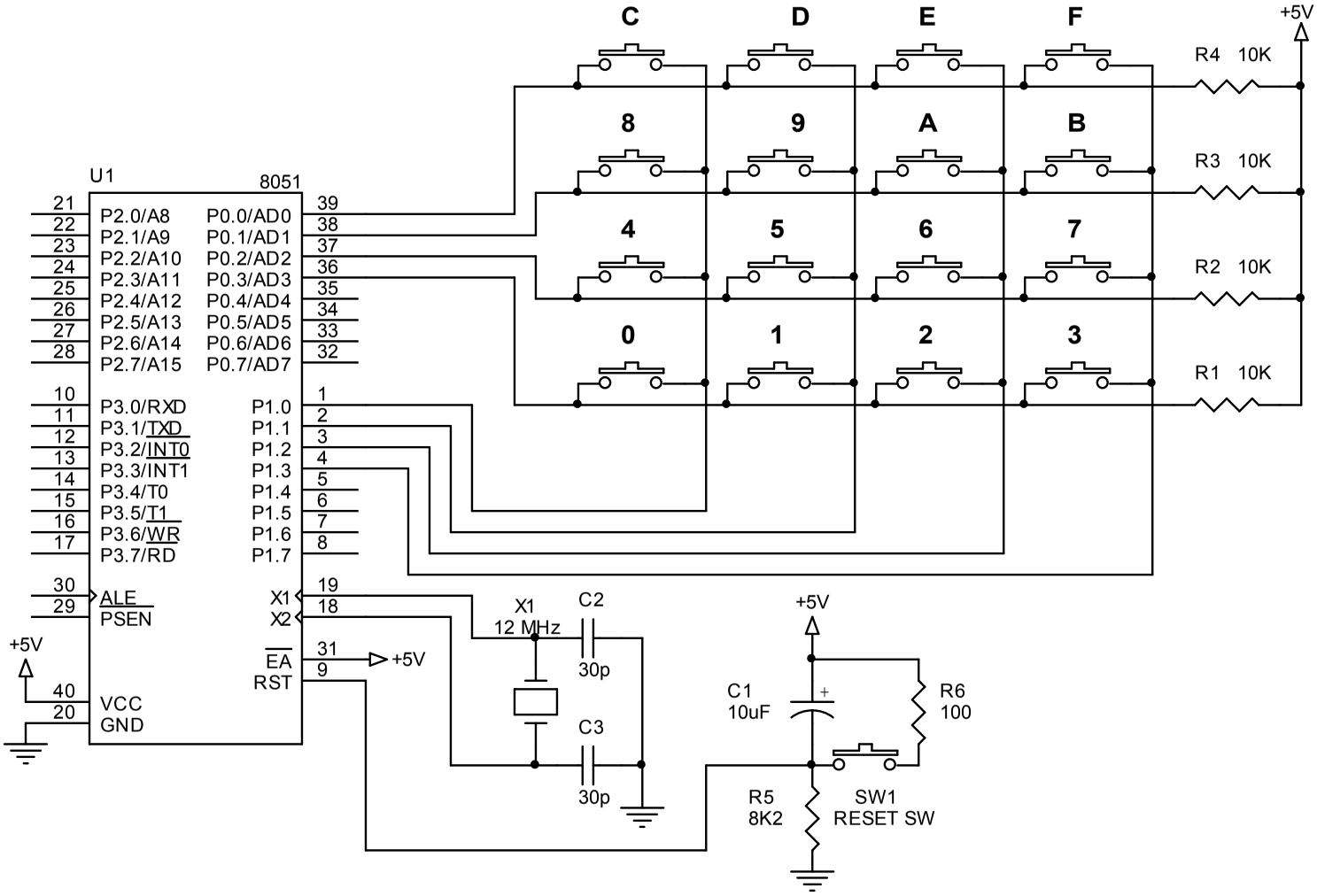
Bài tập: Thiết kế kit vi điều khiển có các ngoại vi sau: 1 EPROM 4KB, 1 SRAM 8 KB, 2 PPI 8255.

Bài tập: Thiết kế kit vi điều khiển có các ngoại vi sau: 1 EPROM 2764 (0000H-1FFFH), 1 SRAM 6264 (2000H-3FFFH), 2 PPI 8255 (A000H-BFFFH và E000H-FFFFH).

Kit vi điều khiển 8051 có 1 EPROM 4KB, 1 SRAM 8 KB, 2 PPI 8255



5. Giao tiếp với bàn phím:

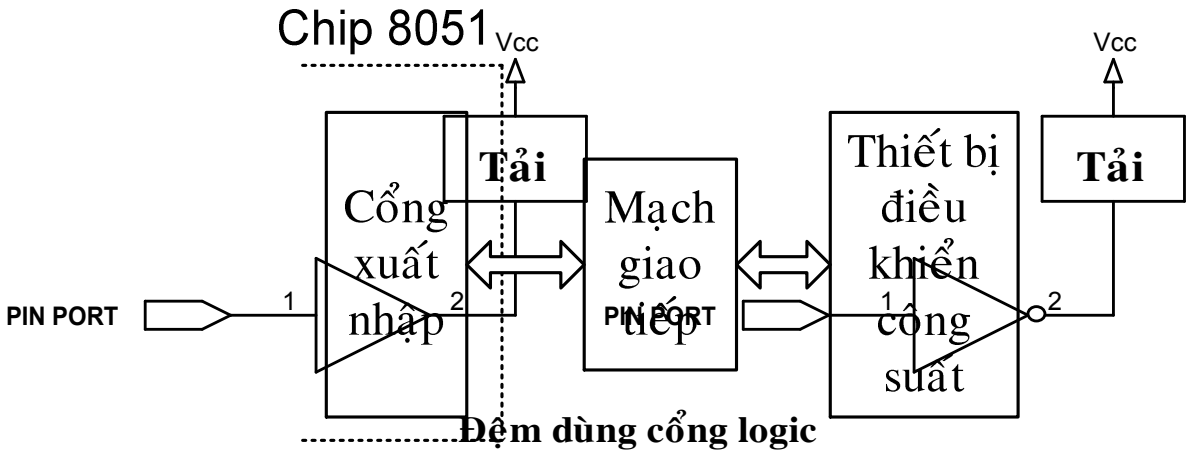


III. NGOẠI VI ĐIỀU KHIỂN CÔNG SUẤT:

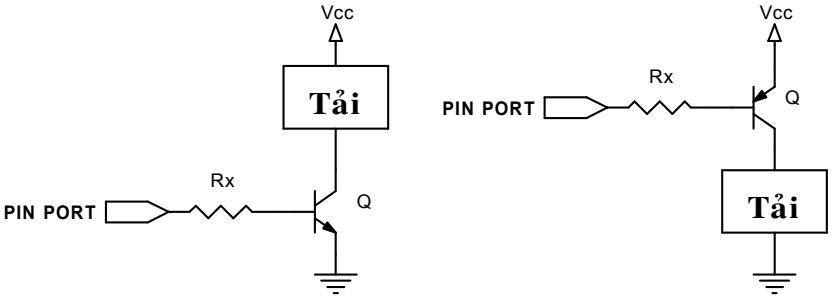
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Ví dụ: các dãy LED, bóng đèn, cuộn dây nam châm điện, relay, cuộn dây động cơ, …

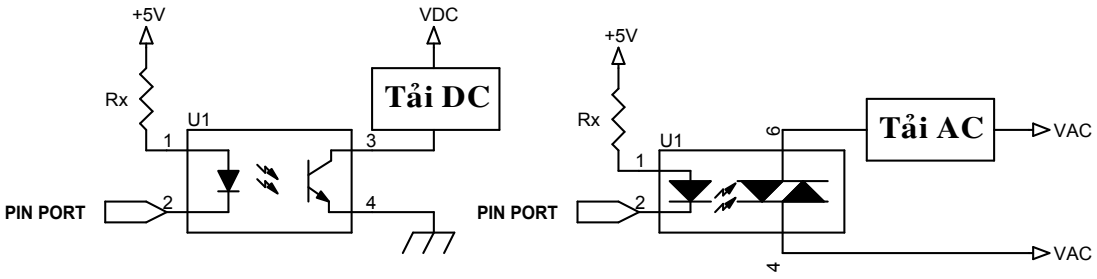
- **Một số kiểu mạch giao tiếp và thiết bị điều khiển công suất:**



Đệm dùng cổng logic
(cổng logic sử dụng loại OC: Output Collector)

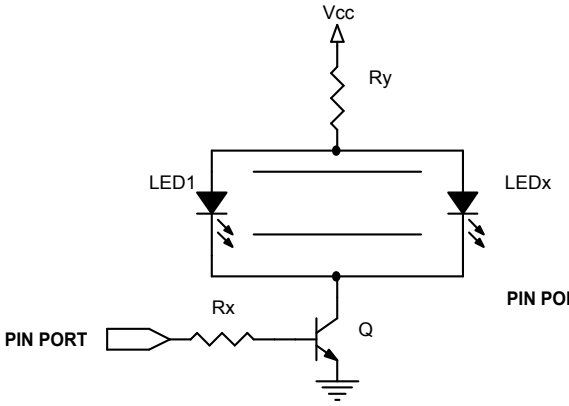


Đệm dùng transistor công suất
(transistor sử dụng loại có dòng Ic > 2 lần dòng tải)

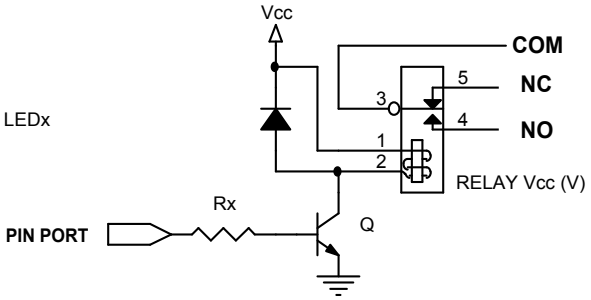


Giao tiếp tải DC cách ly

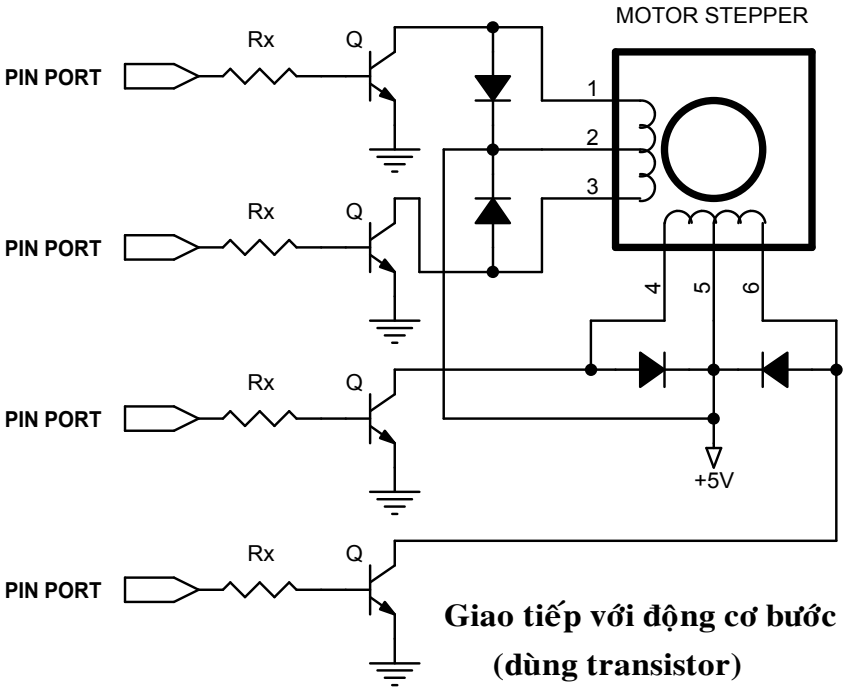
Giao tiếp tải AC cách ly



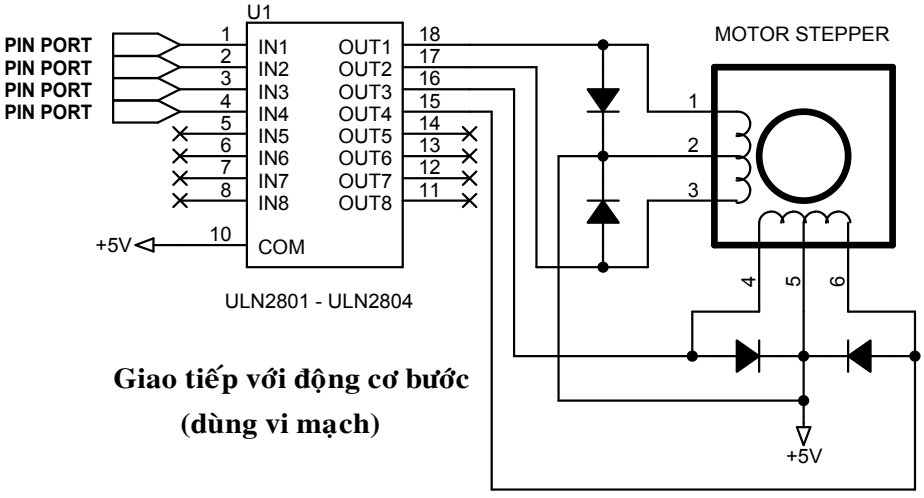
Giao tiếp dãy Led



Giao tiếp với relay



Giao tiếp với động cơ bước (dùng transistor)



Giao tiếp với động cơ bước (dùng vi mạch)