BỘ GIÁO DỤC & ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHỆ
THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ
--- oOo ---

GIÁO TRÌNH

VI ĐIỀU KHIỂN

Tác giả: ThS. PHẠM HÙNG KIM KHÁNH

03/2008

Sưu tầm bởi: www.daihoc.com.vn

LỜI NÓI ĐẦU

Giáo trình Vi điều khiển được biên soạn nhằm cung cấp cho sinh viên kiến thức về họ vi điều khiển MCS-51, cách thức lập trình điều khiển, nạp chương trình và thiết kế phần cứng điều khiển thiết bị.

Giáo trình được sử dụng cho khóa học 45 tiết dành cho sinh viên hệ đại học Khoa Điện Điện tử trường Đại học Kỹ thuật Công nghệ TPHCM.

Bố cục giáo trình gồm 4 chương dựa theo đề cương môn học Kỹ thuật Vi điều khiển dành cho sinh viên ngành Điện Tử Viễn Thông:

Chương 1. Tổng quan về họ vi điều khiển MCS-51

Chương 2. Lập trình hợp ngữ

Chương 3. Các hoạt động của họ vi điều khiển MCS-51

Chương 4. Các ứng dụng

Phụ lục 1: Tóm tắt tập lệnh

Phụ lục 2: Mô tả tập lệnh

PHẠM HÙNG KIM KHÁNH

Sưu tầm bởi: www.daihoc.com.vn

MỤC LỤC

Sưu tầm bởi: www.daihoc.com.vn

Sưu tầm bởi: www.daihoc.com.vn

Sưu tầm bởi: www.daihoc.com.vn

Chương 1: TỔNG QUAN VỀ VI ĐIỀU KHIỂN MCS-51

Chương này giới thiệu tổng quan về họ vi điều khiển MCS-51(chủ yếu trên AT89C51): cấu trúc phần cứng, sơ đồ chân, các thanh ghi, đặc tính lập trình và các đặc tính về điện.

1. Giới thiệu

Họ vi điều khiển MCS-51 do Intel sản xuất đầu tiên vào năm 1980 là các IC thiết kế cho các ứng dụng hướng điều khiển. Các IC này chính là một hệ thống vi xử lý hoàn chỉnh bao gồm các các thành phần của hệ vi xử lý: CPU, bộ nhớ, các mạch giao tiếp, điều khiển ngắt.

MCS-51 là họ vi điều khiển sử dụng cơ chế CISC (Complex Instruction Set Computer), có độ dài và thời gian thực thi của các lệnh khác nhau. Tập lệnh cung cấp cho MCS-51 có các lệnh dùng cho điều khiển xuất / nhập tác động đến từng bit.

MCS-51 bao gồm nhiều vi điều khiển khác nhau, bộ vi điều khiển đầu tiên là 8051 có 4KB ROM, 128 byte RAM và 8031, không có ROM nội, phải sử dụng bộ nhớ ngoài. Sau này, các nhà sản xuất khác như Siemens, Fujitsu, … cũng được cấp phép làm nhà cung cấp thứ hai.

MCS-51 bao gồm nhiều phiên bản khác nhau, mỗi phiên bản sau tăng thêm một số thanh ghi điều khiển hoạt động của MCS-51.

2. Vi điều khiển AT89C51

AT89C51 là vi điều khiển do Atmel sản xuất, chế tạo theo công nghệ CMOS có các đặc tính như sau:

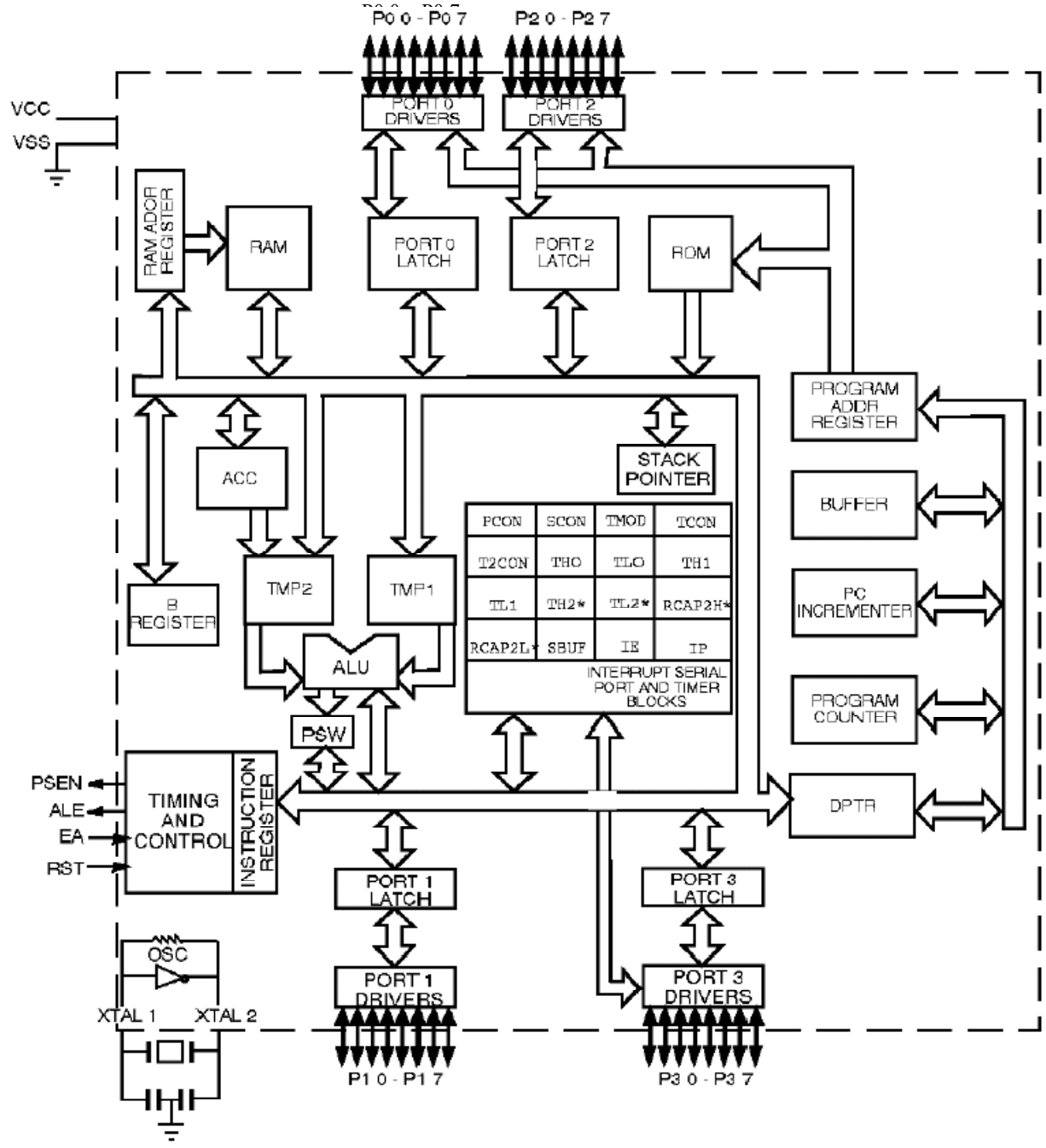
- 4 KB PEROM (Flash Programmable and Erasable Read Only Memory), có khả năng tới 1000 chu kỳ ghi xoá
- Tần số hoạt động từ: 0Hz đến 24 MHz
- 3 mức khóa bộ nhớ lập trình
- 128 Byte RAM nội.
- 4 Port xuất /nhập I/O 8 bit.
- 2 bộ Timer/counter 16 Bit.
- 6 nguồn ngắt.
- Giao tiếp nối tiếp điều khiển bằng phần cứng.
- 64 KB vùng nhớ mã ngoài
- 64 KB vùng nhớ dữ liệu ngoài.
- Cho phép xử lý bit.
- 210 vị trí nhớ có thể định vị bit.
- 4 chu kỳ máy (4 µs đối với thạch anh 12MHz) cho hoạt động nhân hoặc chia.

Sưu tầm bởi: www.daihoc.com.vn

- Có các chế độ nghỉ (Low-power Idle) và chế độ nguồn giảm (Power-down).

Ngoài ra, một số IC khác của họ MCS-51 có thêm bộ định thời thứ 3 và 256 byte RAM nội.

2.1. Sơ đồ

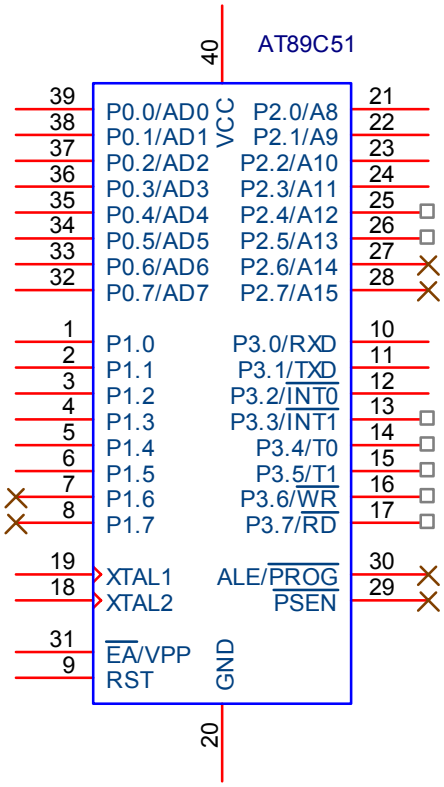


Note: (*)For Timer 2 only.

Hình 1.1 – Sơ đồ khối của AT89C51

Sưu tầm bởi: www.daihoc.com.vn

AT89C51 gồm có 40 chân, mô tả như sau:



Hình 1.2 – Sơ đồ chân của AT89C51

❖ **Port 0:**

Port 0 là port có 2 chức năng ở các chân 32 – 39 của AT89C51:

- Chức năng IO (xuất / nhập): dùng cho các thiết kế nhỏ. Tuy nhiên, khi dùng chức năng này thì Port 0 phải dùng thêm các điện trở kéo lên (pull-up), giá trị của điện trở phụ thuộc vào thành phần kết nối với Port.

  Khi dùng làm ngõ ra, Port 0 có thể kéo được 8 ngõ TTL.

  Khi dùng làm ngõ vào, Port 0 phải được set mức logic 1 trước đó.

- Chức năng địa chỉ / dữ liệu đa hợp: khi dùng các thiết kế lớn, đòi hỏi phải sử dụng bộ nhớ ngoài thì Port 0 vừa là bus dữ liệu (8 bit) vừa là bus địa chỉ (8 bit thấp).

Ngoài ra khi lập trình cho AT89C51, Port 0 còn dùng để nhận mã khi lập trình và xuất mà khi kiểm tra (quá trình kiểm tra đòi hỏi phải có điện trở kéo lên).

Sưu tầm bởi: www.daihoc.com.vn

❖ **Port 1:**

Port1 (chân 1 – 8) chỉ có một chức năng là IO, không dùng cho mục đích khác (chỉ trong 8032/8052/8952 thì dùng thêm P1.0 và P1.1 cho bộ định thời thứ 3). Tại Port 1 đã có điện trở kéo lên nên không cần thêm điện trở ngoài.

Port 1 có khả năng kéo được 4 ngõ TTL và còn dùng làm 8 bit địa chỉ thấp trong quá trình lập trình hay kiểm tra.

Khi dùng làm ngõ vào, Port 1 phải được set mức logic 1 trước đó.

❖ **Port 2:**

Port 2 (chân 21 – 28) là port có 2 chức năng:

- Chức năng IO (xuất / nhập): có khả năng kéo được 4 ngõ TTL.
- Chức năng địa chỉ: dùng làm 8 bit địa chỉ cao khi cần bộ nhớ ngoài có địa chỉ 16 bit. Khi đó, Port 2 không được dùng cho mục đích IO.

Khi dùng làm ngõ vào, Port 2 phải được set mức logic 1 trước đó.

Khi lập trình, Port 2 dùng làm 8 bit địa chỉ cao hay một số tín hiệu điều khiển.

❖ **Port 3:**

Port 3 (chân 10 – 17) là port có 2 chức năng:

- Chức năng IO: có khả năng kéo được 4 ngõ TTL.

Khi dùng làm ngõ vào, Port 3 phải được set mức logic 1 trước đó.

- Chức năng khác: mô tả như bảng 1.1

Bảng 1.1: Chức năng các chân của Port 3

Bit	Tên	Chức năng
P3.0	RxD	Ngõ vào port nối tiếp
P3.1	TxD	Ngõ ra port nối tiếp
P3.2	$\overline{\text{INT0}}$	Ngắt ngoài 0
P3.3	$\overline{\text{INT1}}$	Ngắt ngoài 1
P3.4	T0	Ngõ vào của bộ định thời 0
P3.5	T1	Ngõ vào của bộ định thời 1
P3.6	$\overline{\text{WR}}$	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	$\overline{\text{RD}}$	Tín hiệu điều khiển đọc từ bộ nhớ dữ liệu ngoài.

Sưu tầm bởi: www.daihoc.com.vn

❖ **Nguồn:**

Chân 40: VCC = 5V ± 20%

Chân 20: GND

❖ **$\overline{\text{PSEN}}$ (Program Store Enable):**

$\overline{\text{PSEN}}$ (chân 29) cho phép đọc bộ nhớ chương trình mở rộng đối với các ứng dụng sử dụng ROM ngoài, thường được nối đến chân $\overline{\text{OC}}$ (Output Control) của ROM để đọc các byte mã lệnh. $\overline{\text{PSEN}}$ **sẽ** ở mức logic 0 trong thời gian AT89C51 lấy lệnh.Trong quá trình này, $\overline{\text{PSEN}}$ sẽ tích cực 2 lần trong 1 chu kỳ máy.

Mã lệnh của chương trình được đọc từ ROM thông qua bus dữ liệu (Port0) và bus địa chỉ (Port0 + Port2).

Khi 8951 thi hành chương trình trong ROM nội, $\overline{\text{PSEN}}$ sẽ ở mức logic 1.

❖ **ALE/$\overline{\text{PROG}}$ (Address Latch Enable / Program):**

ALE/$\overline{\text{PROG}}$ (chân 30) cho phép tách các đường địa chỉ và dữ liệu tại Port 0 khi truy xuất bộ nhớ ngoài. ALE thường nối với chân Clock của IC chốt (74373, 74573).

Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động trên chip và có thể được dùng làm tín hiệu clock cho các phần khác của hệ thống. Xung này có thể cấm bằng cách set bit 0 của SFR tại địa chỉ 8Eh lên 1. Khi đó, ALE chỉ có tác dụng khi dùng lệnh MOVX hay MOVC. Ngoài ra, chân này còn được dùng làm ngõ vào xung lập trình cho ROM nội ($\overline{\text{PROG}}$).

❖ **$\overline{\text{EA}}$/VPP (External Access) :**

$\overline{\text{EA}}$ (chân 31) dùng để cho phép thực thi chương trình từ ROM ngoài. Khi nối chân 31 với Vcc, AT89C51 sẽ thực thi chương trình từ ROM nội (tối đa 8KB), ngược lại thì thực thi từ ROM ngoài (tối đa 64KB).

Ngoài ra, chân $\overline{\text{EA}}$ được lấy làm chân cấp nguồn 12V khi lập trình cho ROM.

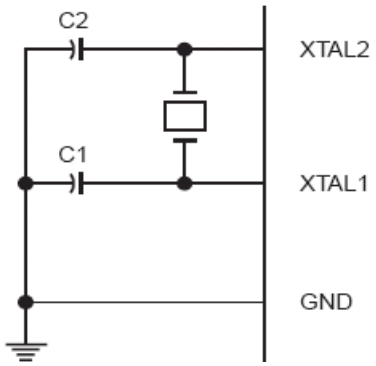
❖ **RST (Reset)**:

RST (chân 9) cho phép reset AT89C51 khi ngõ vào tín hiệu đưa lên mức 1 trong ít nhất là 2 chu kỳ máy.

❖ **X1,X2:**

Ngõ vào và ngõ ra bộ dao động, khi sử dụng có thể chỉ cần kết nối thêm thạch anh và các tụ như hình vẽ trong sơ đồ. Tần số thạch anh thường sử dụng cho AT89C51 là 12Mhz.

Sưu tầm bởi: www.daihoc.com.vn



Giá trị C_1, C_2 = 30 pF ± 10 pF

Hình 1.3 – Sơ đồ kết nối thạch anh

2.2. Định thì chu kỳ máy

Một chu kỳ máy bao gồm 6 trạng thái (12 xung clock). Một trạng thái bao gồm 2 phần ứng với 12 xung clock : Phase 1 và Phase 2. Như vậy, một chu kỳ máy bao gồm 12 xung clock được biểu diễn từ S1P1 đến S6P2 (State 1, Phase 1 → State 6, Phase 2). Chu kỳ lấy lệnh và thực thi lệnh mô tả như hình 1.4.

Tín hiệu chốt địa chỉ ALE tích cực 2 lần trong một chu kỳ máy (trong khoảng thời gian S1P2 đến S2P1 và từ S4P2 đến S5P1). Từ đó tần số xung tại chân ALE bằng 1/6 tần số thạch anh.

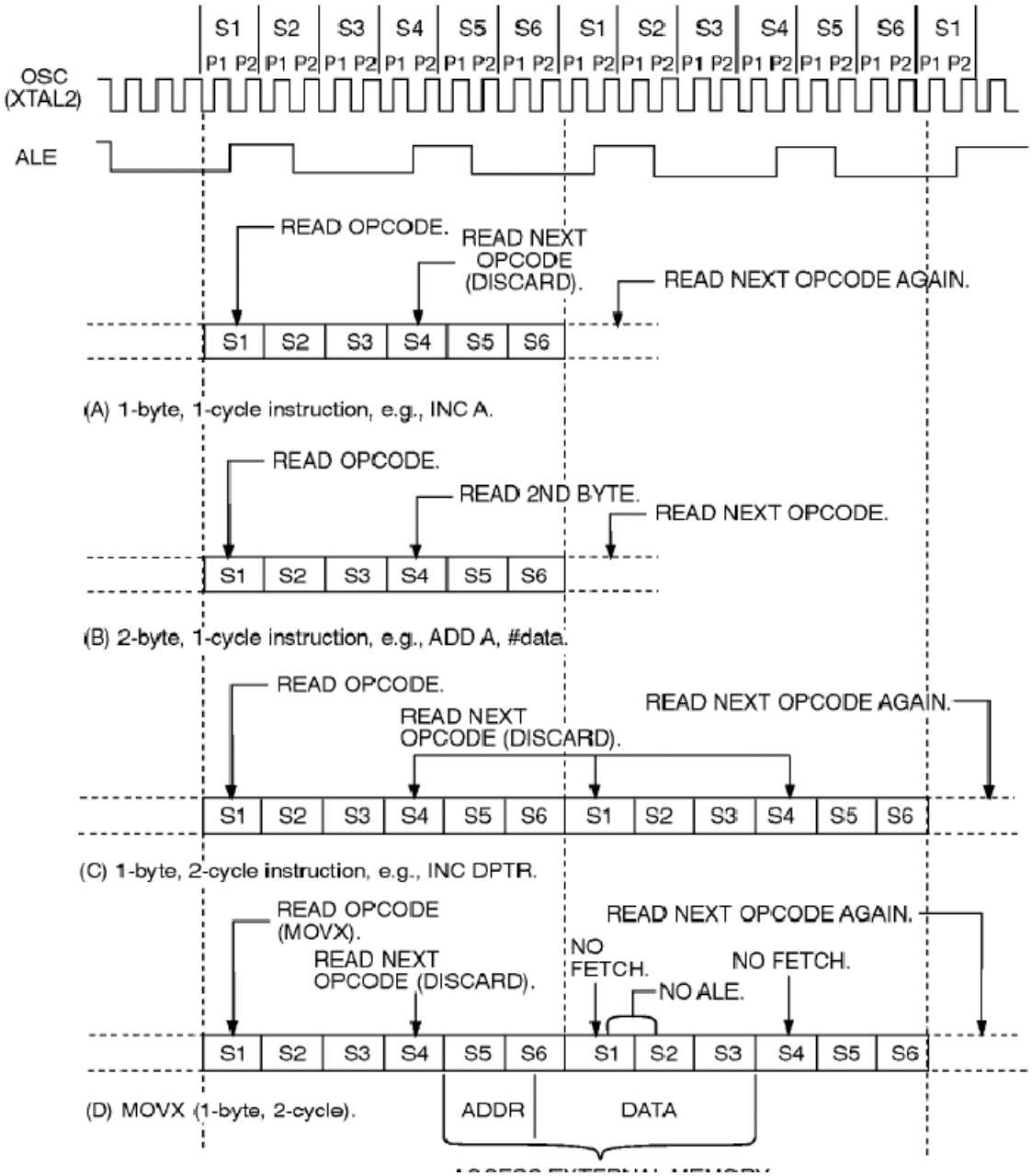
➢ Đối với các lệnh thực thi trong 1 chu kỳ:

- Lệnh 1 byte: được thực thi tại thời điểm S1P2 sau khi mã lệnh được chốt vào thanh ghi lệnh tại S1P1.
- Lệnh 2 byte: byte thứ 2 được đọc tại thời điểm S4 và sẽ được thực thi tại thời điểm S4.

➢ Đối với các lệnh thực thi trong 2 chu kỳ:

Quá trình lấy lệnh thực hiện tại thời điểm S1 của chu kỳ đầu tiên (byte mà lệnh 1). Nếu lệnh có nhiều hơn 1 byte thì sẽ được lấy ở các thời điểm tiếp theo giống như các lệnh thực thi trong 1 chu kỳ.

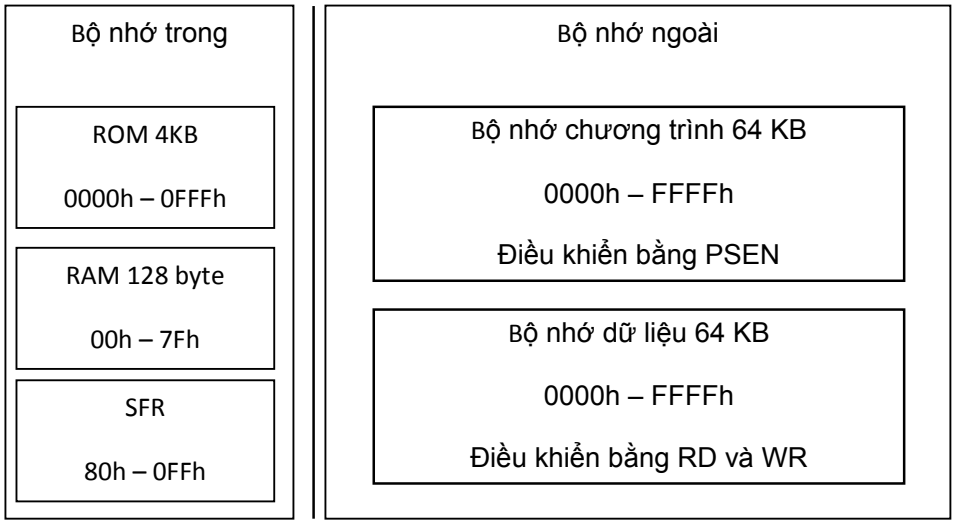
Sưu tầm bởi: www.daihoc.com.vn



Hình 1.4 – Chu kỳ lệnh

Sưu tầm bởi: www.daihoc.com.vn

2.3. Tổ chức bộ nhớ



Hình 1.5 - Các vùng nhớ trong AT89C51

Bộ nhớ của họ MCS-51 có thể chia thành 2 phần: bộ nhớ trong và bộ nhớ ngoài. Bộ nhớ trong bao gồm 4 KB ROM và 128 byte RAM (256 byte trong 8052). Các byte RAM có địa chỉ từ 00h – 7Fh và các thanh ghi chức năng đặc biệt (SFR) có địa chỉ từ 80h – 0FFh có thể truy xuất trực tiếp. Đối với 8052, 128 byte RAM cao (địa chỉ từ 80h – 0FFh) không thể truy xuất trực tiếp mà chỉ có thể truy xuất gián tiếp (xem thêm trong phần tập lệnh).

Bộ nhớ ngoài bao gồm bộ nhớ chương trình (điều khiển đọc bằng tín hiệu $\overline{\text{PSEN}}$) và bộ nhớ dữ liệu (điều khiển bằng tín hiệu $\overline{\text{RD}}$ hay $\overline{\text{WR}}$ để cho phép đọc hay ghi dữ liệu). Do số đường địa chỉ của MCS-51 là 16 bit (Port 0 chứa 8 bit thấp và Port 2 chứa 8 bit cao) nên bộ nhớ ngoài có thể giải mã tối đa là 64KB.

2.3.1. Tổ chức bộ nhớ trong

Bộ nhớ trong của MCS-51 gồm ROM và RAM. RAM bao gồm nhiều vùng có mục đích khác nhau: vùng RAM đa dụng (địa chỉ byte từ 30h – 7Fh và có thêm vùng 80h – 0FFh ứng với 8052), vùng có thể địa chỉ hóa từng bit (địa chỉ byte từ 20h – 2Fh, gồm 128 bit được định địa chỉ bit từ 00h – 7Fh), các bank thanh ghi (từ 00h – 1Fh) và các thanh ghi chức năng đặc biệt (từ 80h – 0FFh).

Sưu tầm bởi: www.daihoc.com.vn

❖ Các thanh ghi chức năng đặc biệt (SFR – Special Function Registers):

Bảng 1.2 – Các thanh ghi chức năng đặc biệt

Địa chỉ byte	Có thể định địa chỉ bit	Không định địa chỉ bit						
F8h								
F0h	B							
E8h								
E0h	ACC							
D8h								
D0h	PSW							
C8h	(T2CON)		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		
C0h								
B8h	IP	SADEN						
B0h	P3							
A8h	IE	SADDR						
A0h	P2							
98h	SCON	SBUF	BRL	BDRCON				
90h	P1							
88h	TCON	TMOD	TL0	TH0	TL1	TH1	AUXR	CKCON
80h	P0	SP	DPL	DPH				PCON

Các thanh ghi có thể định địa chỉ bit sẽ có địa chỉ bit bắt đầu và địa chỉ byte trùng nhau. Ví dụ như: thanh ghi P0 có địa chỉ byte là 80h và có địa chỉ bit bắt đầu từ 80h (ứng với P0.0) đến 87h (ứng với P0.7). Chức năng các thanh ghi này sẽ mô tả trong phần sau.

Sưu tầm bởi: www.daihoc.com.vn

❖ **RAM nội**: chia thành các vùng phân biệt: vùng RAM đa dụng (30h – 7Fh), vùng RAM có thể định địa chỉ bit (20h – 2Fh) và các bank thanh ghi (00h – 1Fh).

<table>
<tr><th>Địa chỉ byte</th><th colspan="8">Địa chỉ bit</th><th>Chức năng</th></tr>
<tr><td>7F</td><td colspan="8"></td><td rowspan="2">Vùng RAM đa dụng</td></tr>
<tr><td>30</td><td colspan="8"></td></tr>
<tr><td>2F</td><td>7F</td><td>7E</td><td>7D</td><td>7C</td><td>7B</td><td>7A</td><td>79</td><td>78</td><td rowspan="14">Vùng có thể định địa chỉ bit</td></tr>
<tr><td>2E</td><td>77</td><td>76</td><td>75</td><td>74</td><td>73</td><td>72</td><td>71</td><td>70</td></tr>
<tr><td>2D</td><td>6F</td><td>6E</td><td>6D</td><td>6C</td><td>6B</td><td>6A</td><td>69</td><td>68</td></tr>
<tr><td>2C</td><td>67</td><td>66</td><td>65</td><td>64</td><td>63</td><td>62</td><td>61</td><td>60</td></tr>
<tr><td>2B</td><td>5F</td><td>5E</td><td>5D</td><td>5C</td><td>5B</td><td>5A</td><td>59</td><td>58</td></tr>
<tr><td>2A</td><td>57</td><td>56</td><td>55</td><td>54</td><td>53</td><td>52</td><td>51</td><td>50</td></tr>
<tr><td>29</td><td>4F</td><td>4E</td><td>4D</td><td>4C</td><td>4B</td><td>4A</td><td>49</td><td>48</td></tr>
<tr><td>28</td><td>47</td><td>46</td><td>45</td><td>44</td><td>43</td><td>42</td><td>41</td><td>40</td></tr>
<tr><td>27</td><td>3F</td><td>3E</td><td>3D</td><td>3C</td><td>3B</td><td>3A</td><td>39</td><td>38</td></tr>
<tr><td>26</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td></tr>
<tr><td>25</td><td>2F</td><td>2E</td><td>2D</td><td>2C</td><td>2B</td><td>2A</td><td>29</td><td>28</td></tr>
<tr><td>24</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td></tr>
<tr><td>23</td><td>1F</td><td>1E</td><td>1D</td><td>1C</td><td>1B</td><td>1A</td><td>19</td><td>18</td></tr>
<tr><td>22</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td></tr>
<tr><td>21</td><td>0F</td><td>0E</td><td>0D</td><td>0C</td><td>0B</td><td>0A</td><td>09</td><td>08</td></tr>
<tr><td>20</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td>1F
18</td><td colspan="8">Bank 3</td><td rowspan="4">Các bank thanh ghi</td></tr>
<tr><td>17
10</td><td colspan="8">Bank 2</td></tr>
<tr><td>1F
08</td><td colspan="8">Bank 1</td></tr>
<tr><td>07
00</td><td colspan="8">Bank thanh ghi 0 (mặc định cho R0-R7)</td></tr>
</table>

Hình 1.6 – Sơ đồ phân bố RAM nội

➢ *RAM đa dụng:*

RAM đa dụng có 80 byte từ địa chỉ 30h – 7Fh có thể truy xuất mỗi lần 8 bit bằng cách dùng chế độ địa chỉ trực tiếp hay gián tiếp.

Các vùng địa chỉ thấp từ 00h – 2Fh cũng có thể sử dụng cho mục đích như trên ngoài các chức năng đề cập như phần sau.

➢ *RAM có thể định địa chỉ bit:*

Vùng địa chỉ từ 20h – 2Fh gồm 16 byte (= 128 bit) có thể thực hiện giống như vùng RAM đa dụng (mỗi lần 8 bit) hay thực hiện truy xuất mỗi lần 1 bit bằng các lệnh

Sưu tầm bởi: www.daihoc.com.vn

xử lý bit. Vùng RAM này có các địa chỉ bit bắt đầu tại giá trị 00h và kết thúc tại 7Fh. Như vậy, địa chỉ bắt đầu 20h (gồm 8 bit) có địa chỉ bit từ 00h – 07h; địa chỉ kết thúc 2Fh có địa chỉ bit từ 78h – Fh.

- *Các bank thanh ghi:*

Vùng địa chỉ từ 00h – 1Fh được chia thành 4 bank thanh ghi: bank 0 từ 00h – 07h, bank 1 từ 08h – 0Fh, bank 2 từ 10h – 17h và bank 3 từ 18h – 1Fh. Các bank thanh ghi này được đại diện bằng các thanh ghi từ R0 đến R7. Sau khi khởi động hệ thống thì bank thanh ghi được sử dụng là bank 0.

Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 đến R7. Việc thay đổi bank thanh ghi có thể thực hiện thông qua thanh ghi từ trạng thái chương trình (PSW).

Các bank thanh ghi này cũng có thể truy xuất bình thường như vùng RAM đa dụng đã nói ở trên.

2.3.2. Tổ chức bộ nhớ ngoài

MCS-51 có bộ nhớ theo cấu trúc Harvard: phân biệt bộ nhớ chương trình và dữ liệu. Chương trình và dữ liệu có thể chứa bên trong nhưng vẫn có thể kết nối với 64KB chương trình và 64KB dữ liệu. Bộ nhớ chương trình được truy xuất thông qua chân $\overline{PSEN}$ còn bộ nhớ dữ liệu đươc truy xuất thông qua chân $\overline{WR}$ hay $\overline{RD}$.

Lưu ý rằng việc truy xuất bộ nhớ chương trình luôn luôn sử dụng địa chỉ 16 bit còn bộ nhớ dữ liệu có thể là 8 bit hay 16 bit tuỳ theo câu lệnh sử dụng. Khi dùng bộ nhớ dữ liệu 8 bit thì có thể dùng Port 2 như là Port I/O thông thường còn khi dùng ở chế độ 16 bit thì Port 2 chỉ dùng làm các bit địa chỉ cao.

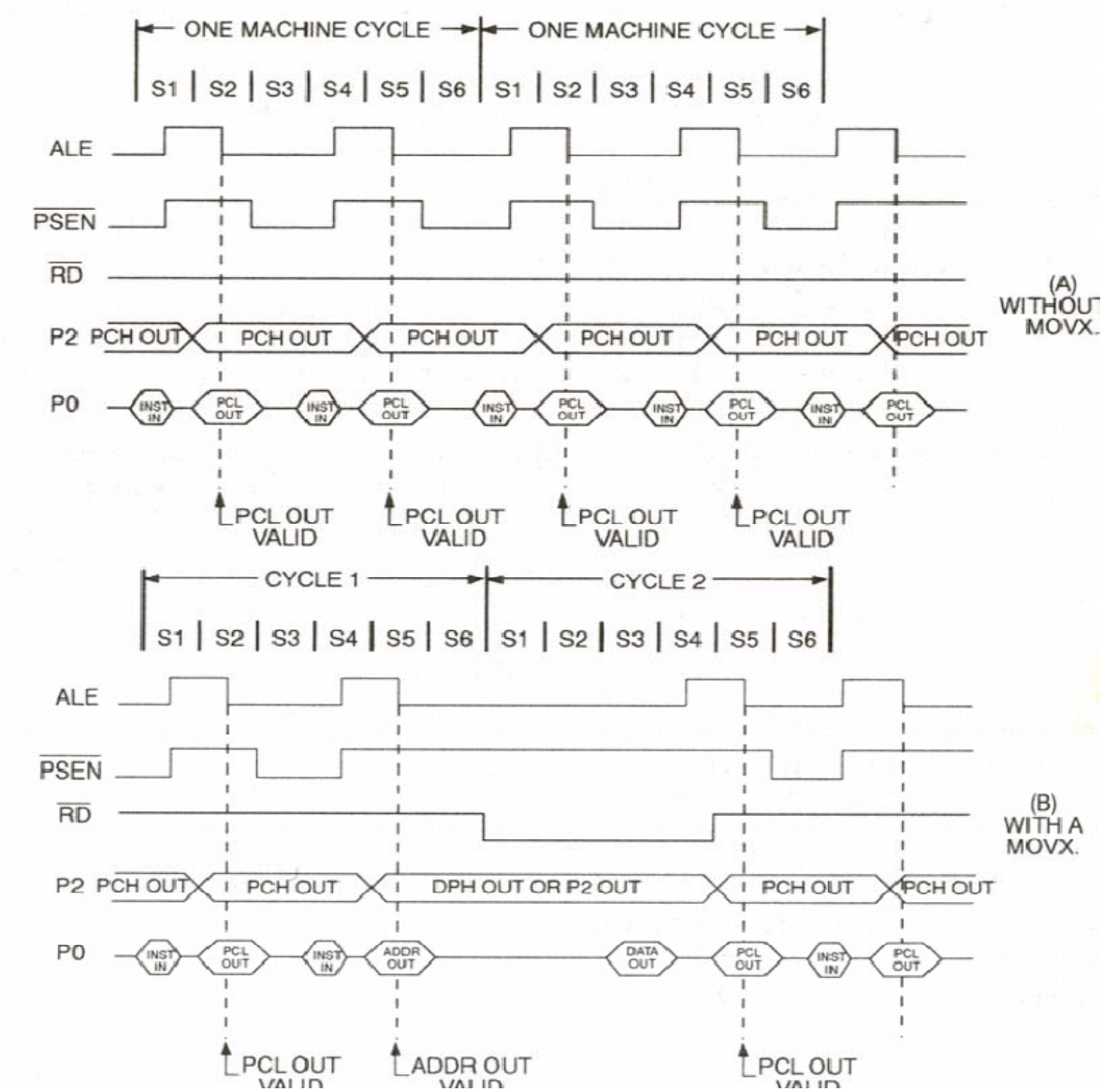
Port 0 được dùng làm địa chỉ thấp/ dữ liệu đa hợp. Tín hiệu ALE để tách byte địa chỉ và đưa vào bộ chốt ngoài.

Trong chu kỳ ghi, byte dữ liệu sẽ tồn tại ở Port 0 vừa trước khi $\overline{WR}$ tích cực và được giữ cho đến khi $\overline{WR}$ không tích cực.Trong chu kỳ đọc, byte nhận được chấp nhận vừa trước khi $\overline{RD}$ không tích cực.

Bộ nhớ chương trình ngoài được xử lý 1 trong 2 điều kiện sau:

- Tín hiệu $\overline{EA}$ tích cực (= 0).
- Giá trị của bộ đếm chương trình (PC – Program Counter) lớn hơn kích thước bộ nhớ.

Sưu tầm bởi: www.daihoc.com.vn



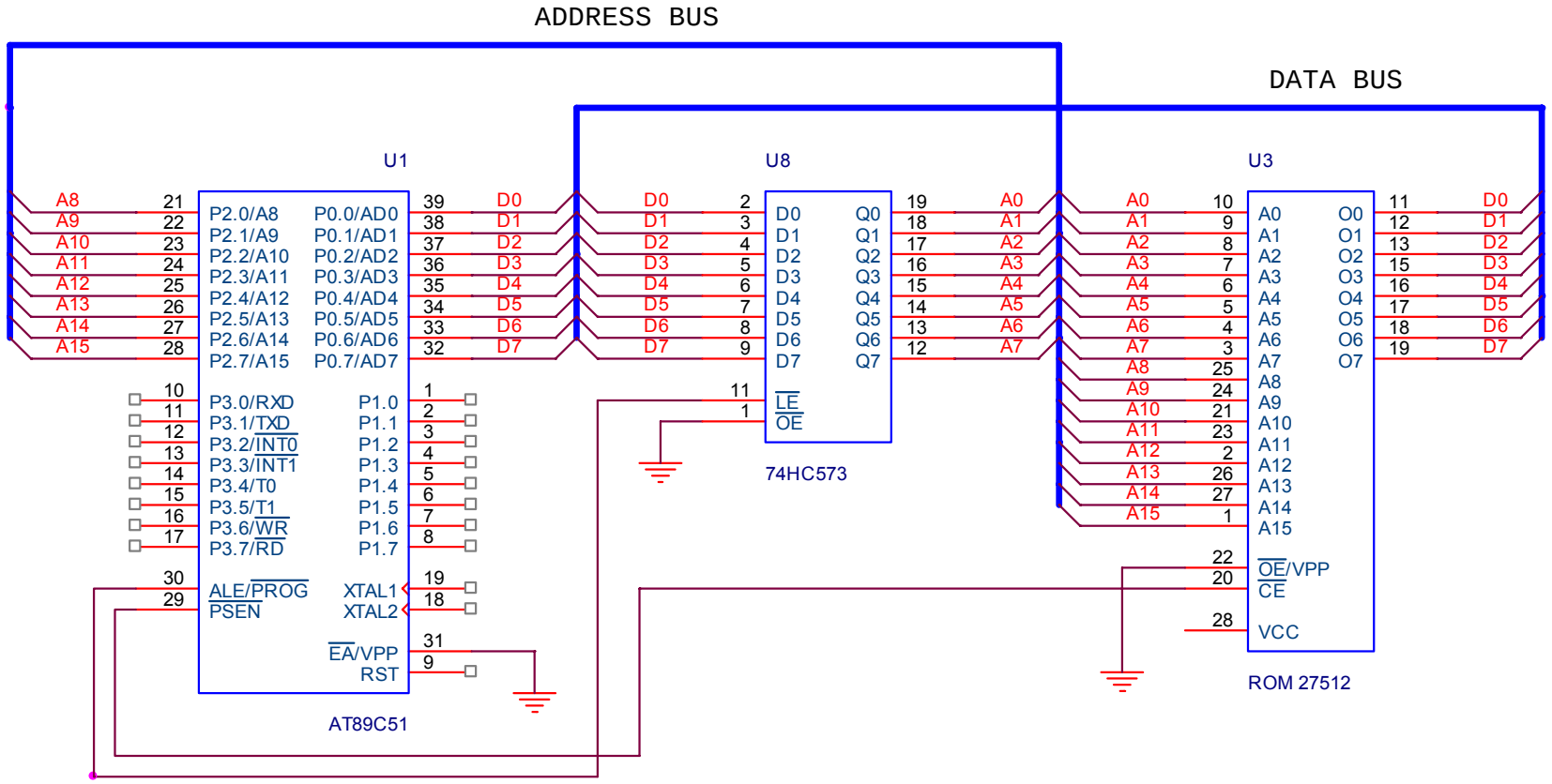
PCH: Program Counter High – PCL: Program Counter Low

DPH: Data Pointer High – DPL: Data Pointer Low

Hình 1.7 – Thực thi bộ nhớ chương trình ngoài

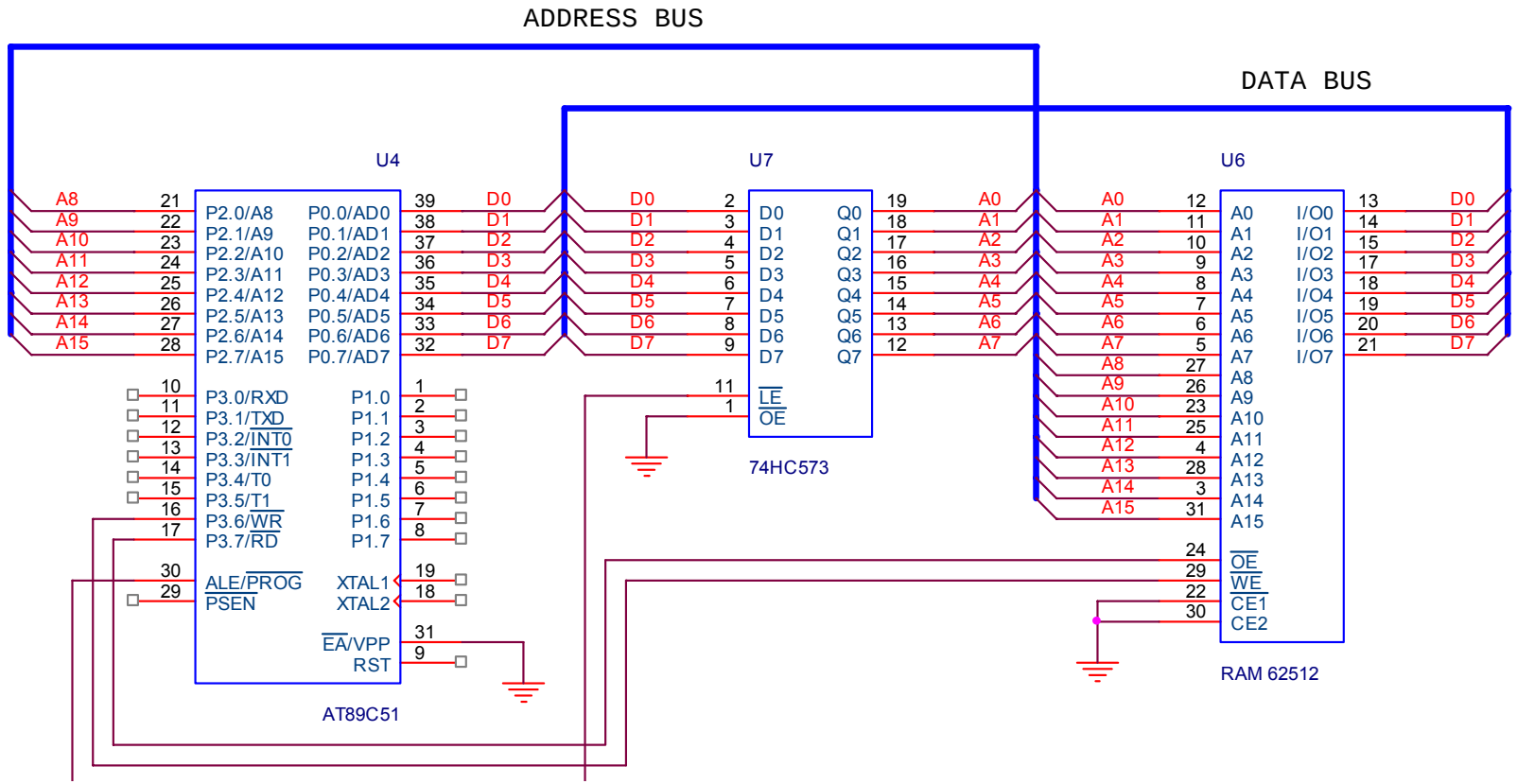
Sưu tầm bởi: www.daihoc.com.vn

Kết nối phần cứng khi thiết kế bộ nhớ ngoài mô tả như sau:



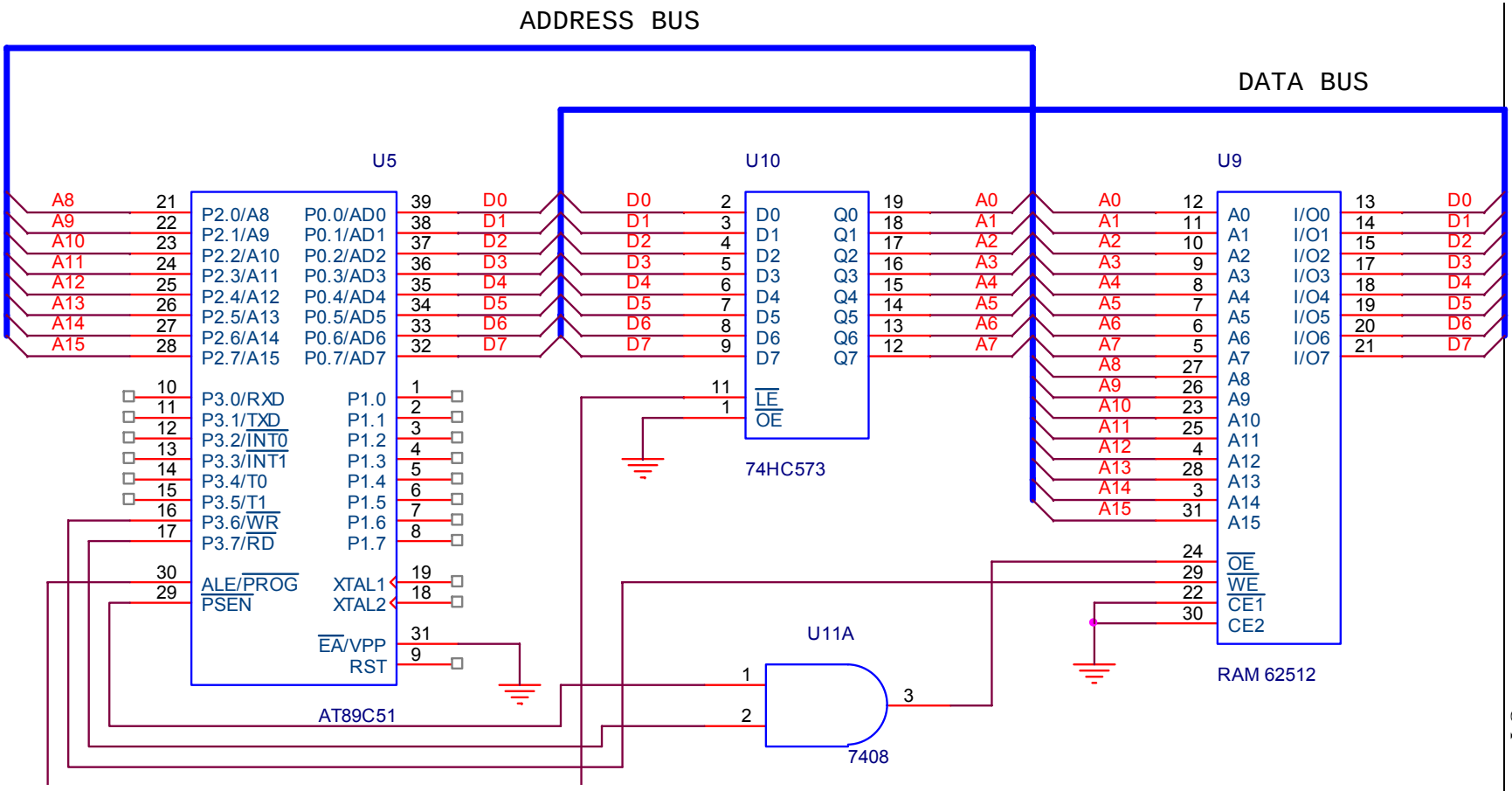
Hình 1.8 – Giao tiếp bộ nhớ chương trình ngoài

Sưu tầm bởi: www.daihoc.com.vn



Hình 1.9 – Giao tiếp bộ nhớ dữ liệu ngoài

Sưu tầm bởi: www.daihoc.com.vn



Hình 1.10 – Giao tiếp bộ nhớ chương trình và dữ liệu ngoài dùng chung

Sưu tầm bởi: www.daihoc.com.vn

❖ **Bộ nhớ chương trình ngoài:**

Quá trình thực thi lệnh khi dùng bộ nhớ chương trình ngoài có thể mô tả như hình 1.7. Trong quá trình này, Port 0 và Port 2 không còn là các Port xuất nhập mà chứa địa chỉ và dữ liệu. Sơ đồ kết nối với bộ nhớ chương trình ngoài mô tả như hình 1.8.

Trong một chu kỳ máy, tín hiệu ALE tích cực 2 lần. Lần thứ nhất cho phép 74HC573 mở cổng chốt địa chỉ byte thấp, khi ALE xuống 0 thì byte thấp và byte cao của bộ đếm chương trình đều có nhưng ROM chưa xuất vì $\overline{\text{PSEN}}$ chưa tích cực, khi tín hiệu ALE lên 1 trở lại thì Port 0 đã có dữ liệu là mã lệnh. ALE tích cực lần thứ hai được giải thích tương tự và byte 2 được đọc từ bộ nhớ chương trình. Nếu lệnh đang thực thi là lệnh 1 byte thì CPU chỉ đọc Opcode, còn byte thứ hai bỏ qua.

❖ **Bộ nhớ dữ liệu ngoài:**

Bộ nhớ dữ liệu ngoài được truy xuất bằng lệnh MOVX thông qua các thanh ghi xác định địa chỉ DPTR (16 bit) hay R0, R1 (8 bit). Sơ đồ kết nối với bộ nhớ dữ liệu ngoài mô tả như hình 1.9.

Quá trình thực hiện đọc hay ghi dữ liệu được cho phép bằng tín hiệu $\overline{\text{RD}}$ hay $\overline{\text{WR}}$ (chân P3.7 và P3.6).

❖ **Bộ nhớ chương trình và dữ liệu dùng chung:**

Trong các ứng dụng phát triển phần mềm xây dựng dựa trên AT89C51, ROM sẽ được lập trình nhiều lần nên dễ làm hư hỏng ROM. Một giải pháp đặt ra là sử dụng RAM để chứa các chương trình tạm thời. Khi đó, RAM vừa là bộ nhớ chương trình vừa là bộ nhớ dữ liệu. Yêu cầu này có thể thực hiện bằng cách kết hợp chân $\overline{\text{RD}}$ và chân $\overline{\text{PSEN}}$ thông qua cổng AND. Khi thực hiện đọc mà lệnh, chân $\overline{\text{PSEN}}$ tích cực cho phép đọc từ RAM và khi đọc dữ liệu, chân $\overline{\text{RD}}$ sẽ tích cực. Sơ đồ kêt nối mô tả như hình 1.10.

2.3.3. Giải mã địa chỉ

Trong các ứng dụng dựa trên AT89C51, ngoài giao tiếp bộ nhớ dỡ liệu, vi điều khiển còn thực hiện giao tiếp với các thiết bị khác như bàn phím, led, động cơ, … Các thiết bị này có thể giao tiếp trực tiếp thông qua các Port. Tuy nhiên, khi số lượng các thiết bị lớn, các Port sẽ không đủ để thực hiện điều khiển. Giải pháp đưa ra là xem các thiết bị này giống như bộ nhớ dữ liệu. Khi đó, cần phải thực hiện quá trình giải mã địa chỉ để phân biệt các thiết bị ngoại vi khác nhau.

Quá trình giải mã địa chỉ thường được thực hiện thông qua các IC giải mã như 74139 (2 -> 4), 74138 (3 -> 8), 74154 (4 -> 16). Ngõ ra của các IC giải mã sẽ được đưa tới chân chọn chip của RAM hay bộ đệm khi điều khiển ngoại vi.

Sưu tầm bởi: www.daihoc.com.vn

2.4. Các thanh ghi chức năng đặc biệt (SFR – Special Function Registers)

2.4.1. Thanh ghi tích luỹ (Accumulator)

Thanh ghi tích luỹ là thanh ghi sử dụng nhiều nhất trong AT89C51, được ký hiệu trong câu lệnh là **A**. Ngoài ra, trong các lệnh xử lý bit, thanh ghi tích luỹ được ký hiệu là **ACC**.

Thanh ghi tích luỹ có thể truy xuất trực tiếp thông qua địa chỉ E0h (byte) hay truy xuất từng bit thông qua địa chỉ bit từ E0h đến E7h.

VD: Câu lệnh:

```
MOV  A,#1
MOV  0E0h,#1
```

có cùng kết quả.

Hay:

```
SETB ACC.4
SETB 0E4h
```

cũng tương tự.

2.4.2. Thanh ghi B

Thanh ghi B dùng cho các phép toán nhân, chia và có thể dùng như một thanh ghi tạm, chứa các kết quả trung gian.

Thanh ghi B có địa chỉ byte F0h và địa chỉ bit từ F0h – F7h có thể truy xuất giống như thanh ghi A.

2.4.3. Thanh ghi từ trạng thái chương trình (PSW - Program Status Word)

Thanh ghi từ trạng thái chương trình PSW nằm tại địa chỉ D0h và có các địa chỉ bit từ D0h – D7h, bao gồm 7 bit (1 bit không sử dụng) có các chức năng như sau:

Bảng 1.3 – Chức năng các bit trong thanh ghi PSW

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Chức năng	CY	AC	F0	RS1	RS0	OV	-	P

CY (Carry): cờ nhớ, thường được dùng cho các lệnh toán học (C = 1 khi có nhớ trong phép cộng hay mượn trong phép trừ)

AC (Auxiliary Carry): cờ nhớ phụ (thường dùng cho các phép toán BCD).

F0 (Flag 0): được sử dụng tuỳ theo yêu cầu của người sử dụng.

Sưu tầm bởi: www.daihoc.com.vn

RS1, RS0: dùng để chọn bank thanh ghi sử dụng. Khi reset hệ thống, bank 0 sẽ được sử dụng.

Bảng 1.4 – Chọn bank thanh ghi

RS1	RS0	Bank thanh ghi
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

OV (Overflow): cờ tràn. Cờ OV = 1 khi có hiện tượng tràn số học xảy ra (dùng cho số nguyên có dấu).

P (Parity): kiểm tra parity (chẵn). Cờ P = 1 khi tổng số bit 1 trong thanh ghi A là số lẻ (nghĩa là tổng số bit 1 của thanh ghi A cộng thêm cờ P là số chẵn). Ví dụ như: A = 10101010b có tổng cộng 4 bit 1 nên P = 0. Cờ P thường được dùng để kiểm tra lỗi truyền dữ liệu.

2.4.4. Thanh ghi con trỏ stack (SP – Stack Pointer)

Con trỏ stack SP nằm tại địa chỉ 81h và không cho phép định địa chỉ bit. SP dùng để chỉ đến đỉnh của stack. Stack là một dạng bộ nhớ lưu trữ dạng LIFO (Last In First Out) thường dùng lưu trữ địa chỉ trả về khi gọi một chương trình con. Ngoài ra, stack còn dùng như bộ nhớ tạm để lưu lại và khôi phục các giá trị cần thiết.

Đối với AT89C51, stack được chứa trong RAM nội (128 byte đối với 8031/8051 hay 256 byte đối với 8032/8052). Mặc định khi khởi động, giá trị của SP là 07h, nghĩa là stack bắt đầu từ địa chỉ 08h (do hoạt động lưu giá trị vào stack yêu cầu phải tăng nội dung thanh ghi SP trước khi lưu). Như vậy, nếu không gán giá trị cho thanh ghi SP thì không được sử dụng các bank thanh ghi 1, 2, 3 vì có thể làm sai dữ liệu.

Đối với các ứng dụng thông thường không cần dùng nhiều đến stack, có thể không cần khởi động SP mà dùng giá trị mặc định là 07h. Tuy nhiên, nếu cần, ta có thể xác định lại vùng stack cho MCS-51.

2.4.5. Con trỏ dữ liệu DPTR (Data Pointer)

Con trỏ dữ liệu DPTR là thanh ghi 16 bit bao gồm 2 thanh ghi 8 bit: DPH (High) nằm tại địa chỉ 83h và DPL (Low) nằm tại địa chỉ 82h. Các thanh ghi này không cho phép định địa chỉ bit. DPTR được dùng khi truy xuất đến bộ nhớ có địa chỉ 16 bit.

2.4.6. Các thanh ghi port

Các thanh ghi P0 tại địa chỉ 80h, P1 tại địa chỉ 90h, P2, tại địa chỉ A0h, P3 tại địa chỉ B0h là các thanh ghi chốt cho 4 port xuất / nhập (Port 0, 1, 2, 3). Tất cả các thanh ghi này đều cho phép định địa chỉ bit trong đó địa chỉ bit của P0 từ 80h – 87h, P1 từ 90h – 97h, P2 từ A0h – A7h, P3 từ B0h – B7h. Các địa chỉ bit này có thể thay thế bằng toán tử •. Ví dụ như: 2 lệnh sau là tương đương:

Sưu tầm bởi: www.daihoc.com.vn

```
SETB P0.0

SETB 80h
```

2.4.7. Thanh ghi port nối tiếp (SBUF - Serial Data Buffer)

Thanh ghi port nối tiếp tại địa chỉ 99h thực chất bao gồm 2 thanh ghi: thanh ghi nhận và thanh ghi truyền. Nếu dữ liệu đưa tới SBUF thì đó là thanh ghi truyền, nếu dữ liệu đươc đọc từ SBUF thì đó là thanh ghi nhận. Các thanh ghi này không cho phép định địa chỉ bit.

2.4.8. Các thanh ghi định thời (Timer Register)

Các cặp thanh ghi (TH0, TL0), (TH1, TL1) và (TH2, TL2) là các thanh ghi dùng cho các bộ định thời 0, 1 và 2 trong đó bộ định thời 2 chỉ có trong 8032/8052. Ngoài ra, đối với họ 8032/8052 còn có thêm cặp thanh ghi (RCAP2L, RCAP2H) sử dụng cho bộ định thời 2 (sẽ thảo luận trong phần hoạt động định thời).

2.4.9. Các thanh ghi điều khiển

Bao gồm các thanh ghi IP (Interrupt Priority), IE (Interrupt Enable), TMOD (Timer Mode), TCON (Timer Control), T2CON (Timer 2 Control), SCON (Serial port control) và PCON (Power control).

- Thanh ghi IP tại địa chỉ B8h cho phép chọn mức ưu tiên ngắt khi có 2 ngắt xảy ra đông thời. IP cho phép định địa chỉ bit từ B8h – BFh.
- Thanh ghi IE tại địa chỉ A8h cho phép hay cấm các ngắt. IE có địa chỉ bit từ A8h – AFh.
- Thanh ghi TMOD tại địa chỉ 89h dùng để chọn chế độ hoạt động cho các bộ định thời (0, 1) và không cho phép định địa chỉ bit.
- Thanh ghi TCON tại địa chỉ 88h điều khiển hoạt động của bộ định thời và ngắt. TCON có địa chỉ bit từ 88h – 8Fh.
- Thanh ghi T2CON tại địa chỉ C8h điều khiển hoạt động của bộ định thời 2. T2CON có địa chỉ bit từ C8h – CFh.
- Thanh ghi SCON tại địa chỉ 98h điều khiển hoạt động của port nối tiếp. SCON có địa chỉ bit từ 98h – 9Fh.

Các thanh ghi đã nói ở trên sẽ được thảo luận thêm ở các phần sau.

Sưu tầm bởi: www.daihoc.com.vn

❖ Thanh ghi điều khiển nguồn PCON

Thanh ghi PCON tại địa chỉ 87h không cho phép định địa chỉ bit bao gồm các bit như sau:

Bảng 1.5 – Chức năng các bit trong thanh ghi PCON

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Chức năng	SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL

SMOD1 (Serial Mode 1): = 1 cho phép tăng gấp đôi tốc độ port nối tiếp trong chế độ 1, 2 và 3.

SMOD0 (Serial Mode 0): cho phép chọn bit SM0 hay FE trong thanh ghi SCON (= 1 chọn bit FE).

POF (Power-off Flag): dùng để nhận dạng loại reset. POF = 1 khi mở nguồn. Do đó, để xác định loại reset, cần phải xoá bit POF trước đó.

GF1, GF0 (General purpose Flag): các bit cờ dành cho người sử dụng.

PD (Power Down): được xoá bằng phần cứng khi hoạt động reset xảy ra. Khi bit PD = 1 thì vi điều khiển sẽ chuyển sang chế độ nguồn giảm. Trong chế độ này:

- Chỉ có thể thoát khỏi chế độ nguồn giảm bằng cách reset.
- Nội dung RAM và mức logic trên các port được duy trì.
- Mạch dao động bên trong và các chức năng khác ngừng hoạt động.
- Chân ALE và $\overline{\text{PSEN}}$ ở mức thấp.
- Yêu cầu Vcc phải có điện áp ít nhất là 2V và phục hồi Vcc = 5V ít nhất 10 chu kỳ trước khi chân RESET xuống mức thấp lần nữa.

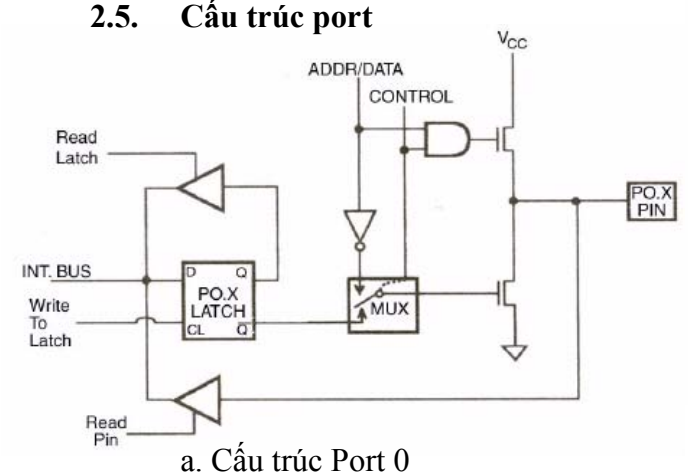
IDL (Idle): được xoá bằng phần cứng khi hoạt động reset hay có ngắt xảy ra. Khi bit IDL = 1 thì vi điều khiển sẽ chuyển sang chế độ nghỉ. Trong chế độ này:

- Chỉ có thể thoát khỏi chế độ nguồn giảm bằng cách reset hay có ngắt xảy ra.
- Trạng thái hiện hành của vi điều khiển được duy trì và nội dung các thanh ghi không đổi.
- Mạch dao động bên trong không gởi được tín hiệu đến CPU.
- Chân ALE và $\overline{\text{PSEN}}$ ở mức cao.

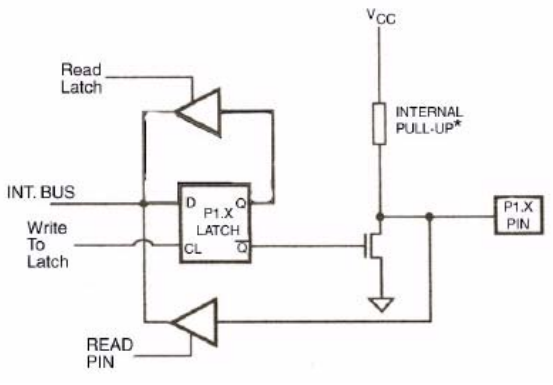
Lưu ý rằng các bit điều khiển PD và IDL có tác dụng chính trong tất cả các IC họ MSC-51 nhưng chỉ có thể thực hiện được trong các phiên bản CMOS.

Sưu tầm bởi: www.daihoc.com.vn

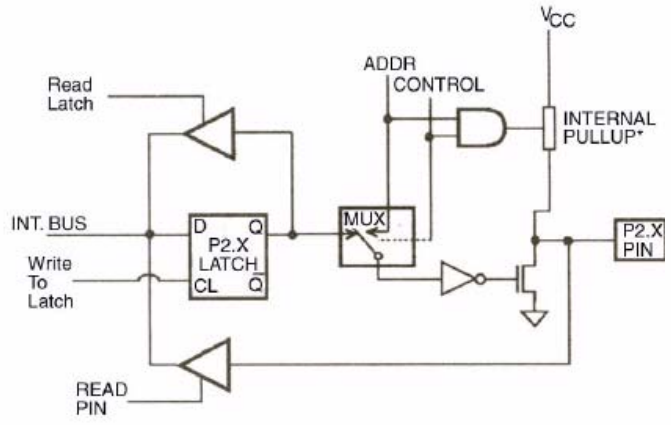
2.5. Cấu trúc port



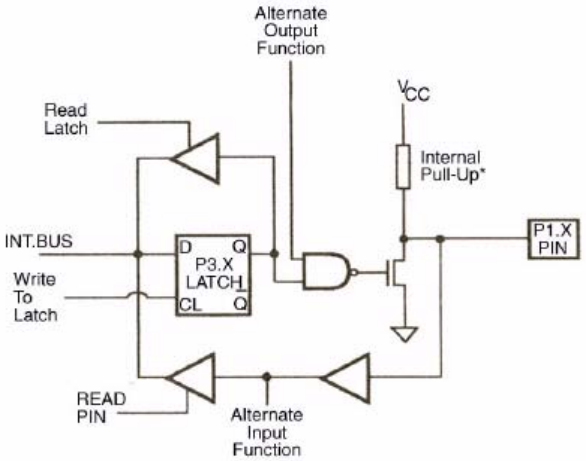
a. Cấu trúc Port 0



b. Cấu trúc Port 1



c. Cấu trúc Port 2



d. Cấu trúc Port 3

Hình 1.11 – Cấu trúc các Port của AT89C51

Cấu trúc các Port mô tả như hình vẽ, mỗi port có một bộ chốt (SFR từ P0 đến P3), một bộ đệm vào và bộ lái ngõ ra.

❖ **Port 0:**

- Khi dùng ở chế độ IO: FET kéo lên tắt (do không có các tín hiệu ADDR và CONTROL) nên ngõ ra Port 0 hở mạch. Như vậy, khi thiết kế Port 0 làm việc ở chế độ IO, cần phải có các điện trở kéo lên. Trong chế độ này, mỗi chân của Port 0 khi dùng làm ngõ ra có thể kéo tối đa 8 ngõ TTL (xem thêm phần sink / source trong 2.7).

Khi ghi mức logic 1 ra Port 0, ngõ ra $\overline{Q}$ của bộ chốt (latch) ở mức 0 nên FET tắt, ngõ ra Port 0 nối lên Vcc thông qua FET và có thể kéo xuống mức 0 khi kết nối với tín hiệu ngoài. Khi ghi mức logic 0 ra Port 0, ngõ ra $\overline{Q}$ của bộ chốt ở mức 1 nên FET dẫn, ngõ ra Port 0 được nối với GND nên luôn ở mức 0 bất kể ngõ vào. Do đó, để đọc dữ liệu tại Port 0 thì cần phải set bit tương ứng.

Sưu tầm bởi: www.daihoc.com.vn

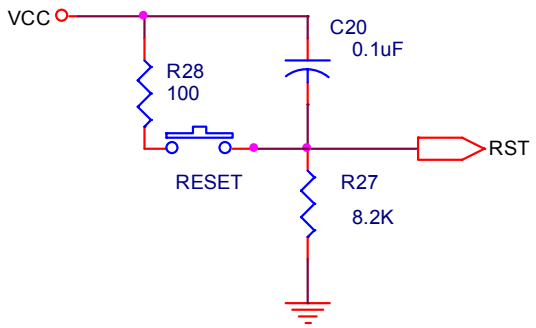
- Khi dùng ở chế độ địa chỉ / dữ liệu: FET đóng vai trò như điện trở kéo lên nên không cần thiết kế thêm các điện trở ngoài.

❖ **Port 1, 2, 3:**

Không dùng FET mà dùng điện trở kéo lên nên khi thiết kế không cần thiết phải thêm các điện trở ngoài. Khi dùng ở chế độ IO, cách thức hoạt động giống như Port 0 (nghĩa là trước khi đọc dữ liệu thì cần phải set bit tương ứng). Port 1, 2, 3 có khả năng sink / source dòng cho 4 ngõ TTL.

2.6. Hoạt động Reset

Để thực hiện reset, cần phải tác động mức cao tại chân RST (chân 9) của AT89C51 ít nhất 2 chu kỳ máy. Sơ đồ mạch reset có thể mô tả như sau:



Hình 1.12 – Sơ đồ mạch reset của AT89C51

Sau khi reset, nội dung của RAM nội không thay đổi và các thanh ghi thay đổi về giá trị mặc định như sau:

Bảng 1.6 - Giá trị mặc định của các thanh ghi khi reset

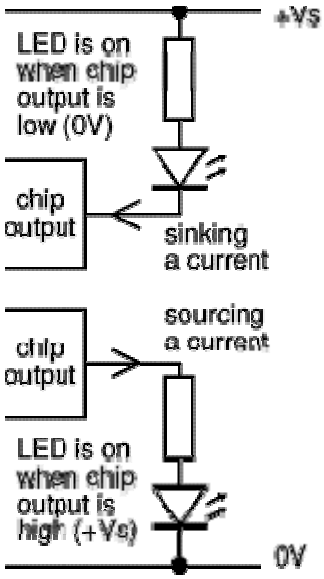
Thanh ghi	Nội dung
Đếm chương trình PC	0000h
A, B, PSW, SCON, SBUF	00h
SP	07h
DPTR	0000h
Port 0 đến port 3	FFh
IP	XXX0 0000b
IE	0X0X 0000b
Các thanh ghi định thời	00h
PCON (HMOS)	0XXX XXXXb
PCON (CMOS)	0XXX 0000b

Sưu tầm bởi: www.daihoc.com.vn

2.7. Các vấn đề khác

2.7.1. Dòng sink và source

Dòng điện sink và source là một phần quan trọng khi thiết kế các mạch điện tử. Sự khác nhau của chúng được mô tả như hình 1.13.



Hình 1.13 – Khác nhau giữa dòng sink và source

Trong AT89C51, Port 0 có dòng sink của mỗi chân tương đương với 8 ngõ TTL còn các Port khác có dòng sink /source tương đương với 4 ngõ TTL.

2.7.2. Lập trình cho AT89C51

2.7.2.1. Các chế độ khoá bộ nhớ chương trình

Bảng 1.7 – Các chế độ khoá chương trình

Chế độ	Lập trình các bit khoá			Mô tả
	LB1	LB2	LB3	
1	U	U	U	Không khoá
2	P	U	U	Không cho phép lệnh MOVC tại bộ nhớ chương trình ngoài, chân $\overline{EA}$ được lấy mẫu và chốt khi reset, không cho phép lập trình.
3	P	P	U	Giống chế độ 2 và không cho phép kiểm tra.
4	P	P	P	Giống chế độ 3 và không cho phép thực thi ngoài.

Trong AT89C51, có 3 bit khoá (LB – lock bit) có thể được lập trình (P – programmed) hay không (U – unprogrammed) cho phép chọn các chế độ khoá khác nhau (bảng 1.7).

Sưu tầm bởi: www.daihoc.com.vn

2.7.2.2. Lập trình

Khi AT89C51 ở trạng thái xoá, tất cả các ô nhớ thường là 0FFh và có thể được lập trình. Điện áp lập trình có thể là 5V hay 12V tuỳ theo loại IC. Điện áp lập trình xác định bằng ký hiệu trên chip hay các byte nhận dạng khi đã xoá chip (xem bảng 1.8).

Bảng 1.8 – Nhận dạng điện áp lập trình

	V_{pp} = 12V	V_{pp} = 5V
Ký hiệu	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Byte nhận dạng	(30h) = 1Eh (31h) = 51h (32h) = 0FFh	(30h) = 1Eh (31h) = 51h (32h) = 05h

Lưu ý rằng AT89C51 được lập trình theo từng byte nên phải thực hiện xoá tất cả chip trước khi lập trình.

Quá trình lập trình cho AT89C51 được thực hiện theo các bước sau:

- Bước 1: Đặt giá trị địa chỉ lên đường địa chỉ.
- Bước 2: Đặt dữ liệu lên đường dữ liệu.
- Bước 3: Đặt các tín hiệu điều khiển tương ứng (xem bảng 1.9).
- Bước 4: Đặt chân $\overline{EA}$/VPP lên điện áp 12V (nếu sử dụng điện áp lập trình 12V).
- Bước 5: Tạo một xung tại chân ALE/$\overline{PROG}$ (xem bảng 1.9). Thường chu kỳ ghi 1 byte không vượt quá 1.5 ms. Sau đó thay đổi địa chỉ và lặp lại bước 1 cho đến khi kết thúc dữ liệu cần lập trình.

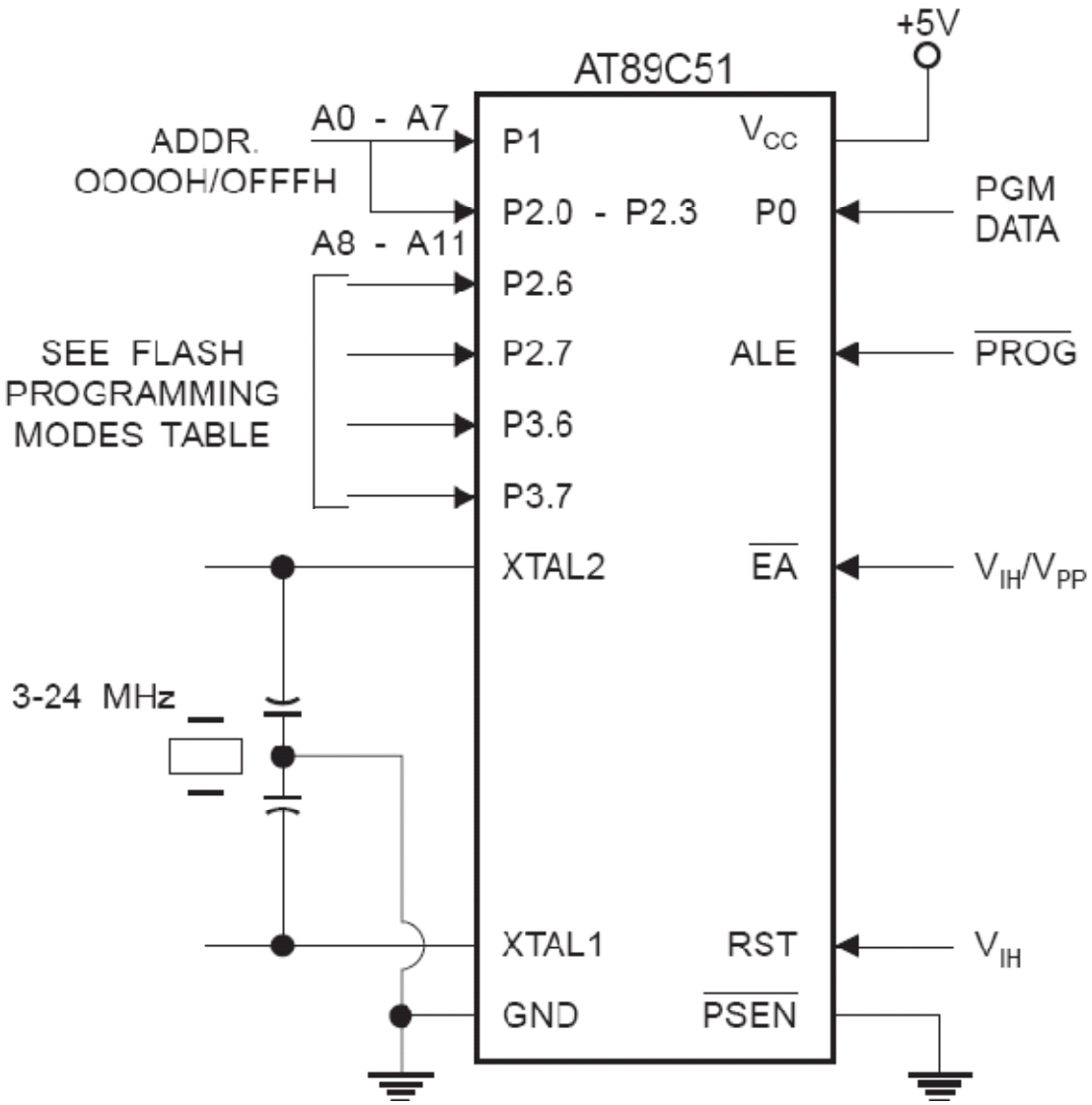
Bảng 1.9 – Các tín hiệu điều khiển lập trình

Chế độ		RST	$\overline{PSEN}$	$\overline{PROG}$	V_{pp}	P2.6	P2.7	P3.6	P3.7
Ghi mã		H	L	⎍	H/12V	L	H	H	H
Đọc mã		H	L	H	H	L	L	H	H
Ghi lock bit	LB1	H	L	⎍	H/12V	H	H	H	H
	LB2	H	L	⎍	H/12V	H	H	L	L
	LB3	H	L	⎍	H/12V	H	L	H	L
Xoá chip		H	L	⎍	H/12V	H	L	L	L
Đọc byte nhận dạng		H	L	H	H	L	L	L	L

Lưu ý rằng các xung $\overline{PROG}$ đòi hỏi thời gian không vượt quá 1.5 ms, chỉ có chế độ xoá chip cần xung 10ms.

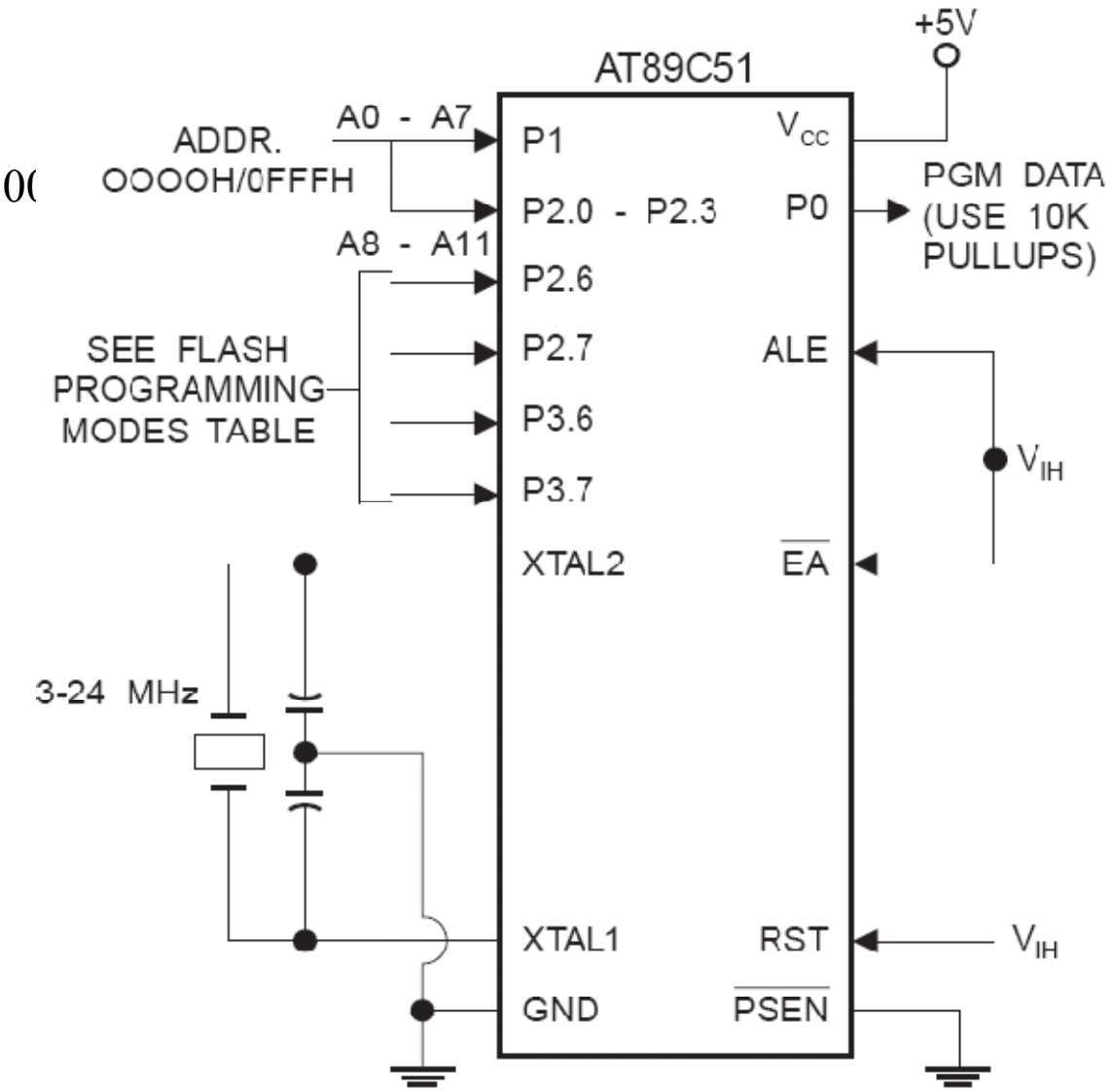
Sơ đồ mạch lập trình và kiểm tra cho AT89C51 mô tả như hình 1.14 và 1.15.

Sưu tầm bởi: www.daihoc.com.vn



Hình 1.14 – Sơ đồ mạch lập trình cho AT89C51

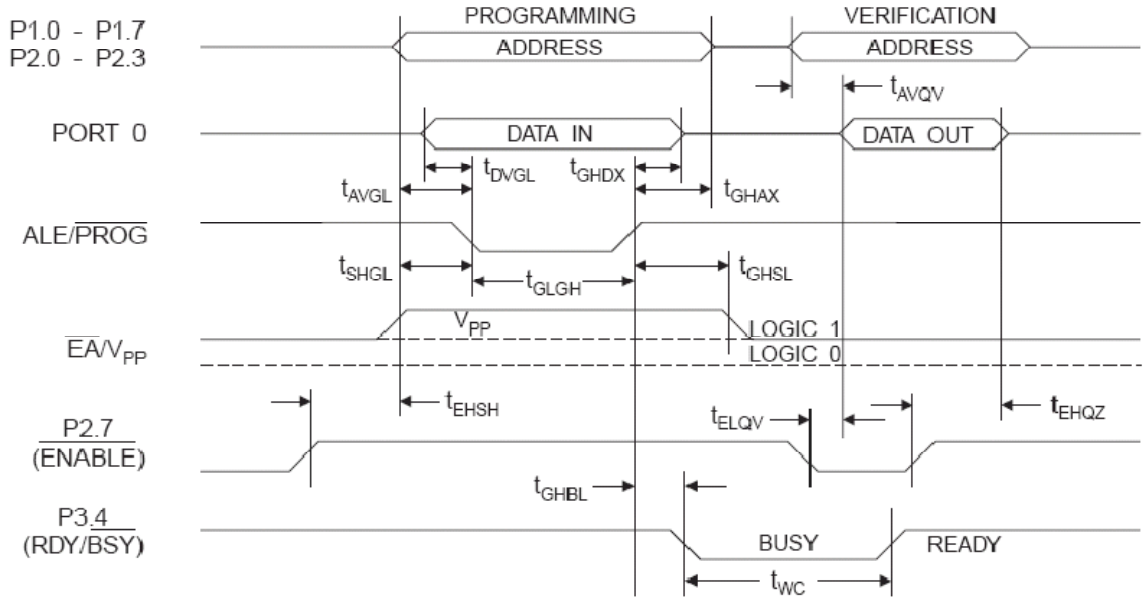
Sưu tầm bởi: www.daihoc.com.vn



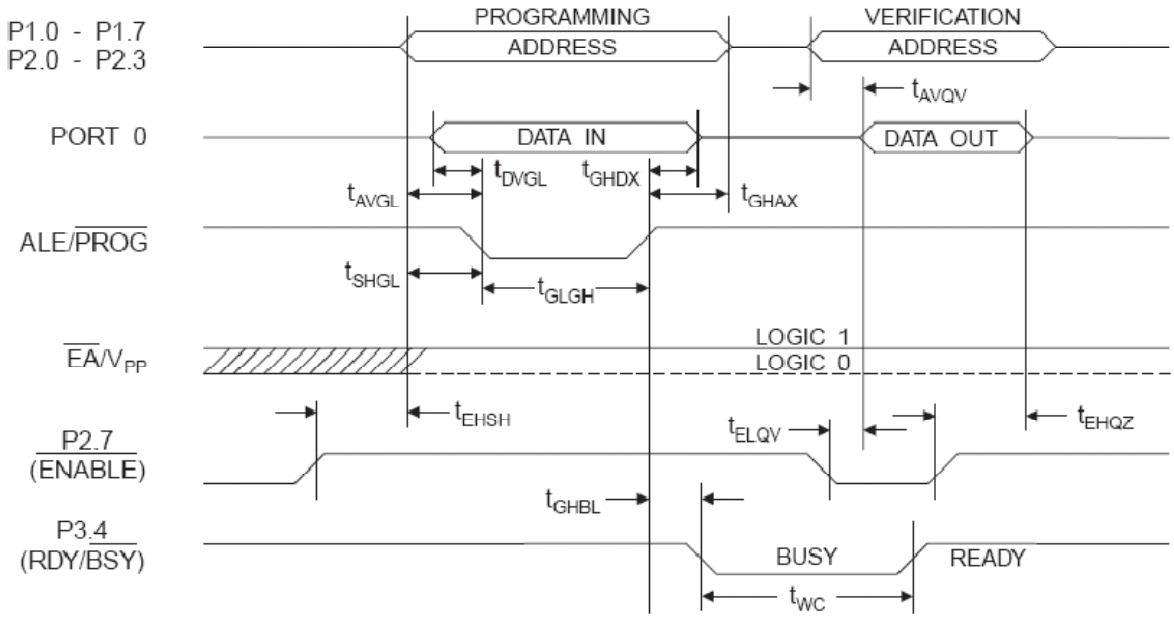
Hình 1.15 – Sơ đồ mạch kiểm tra cho AT89C51

Sưu tầm bởi: www.daihoc.com.vn

Các dạng tín hiệu dùng để lập trình cho AT89C51 được mô tả như hình 1.16 và 1.17.



Hình 1.16 – Dạng sóng lập trình ở điện áp 12V



Hình 1.17 - Dạng sóng lập trình ở điện áp 5V

Sưu tầm bởi: www.daihoc.com.vn

Khi lập trình, các thông số về thời gian và điện áp được mô tả như bảng 1.10.

Bảng 1.10 – Các đặc tính lập trình và kiểm tra
T = 0 – 70^{0}C, V_{CC} = 5V ± 10%

Ký hiệu	Mô tả	Min	Max	Đơn vị
V_{PP} (1)	Điện áp lập trình	11.5	12.5	V
I_{PP} (1)	Dòng điện lập trình		1.0	mA
$1/t_{CLCL}$	Tần số thạch anh	3	24	MHz
t_{AVGL}	Khoảng thời gian từ lúc địa chỉ ổn định cho đến khi có thể tạo xung $\overline{PROG}$ (xuống mức thấp)	$48t_{CLCL}$		
t_{GHAX}	Khoảng thời gian giữ lại địa chỉ sau khi chân $\overline{PROG}$ lên mức cao	$48t_{CLCL}$		
t_{DVGL}	Khoảng thời gian từ lúc dữ liệu ổn định cho đến khi có thể tạo xung $\overline{PROG}$ (xuống mức thấp)	$48t_{CLCL}$		
t_{GHDX}	Khoảng thời gian giữ lại dữ liệu sau khi chân $\overline{PROG}$ lên mức cao	$48t_{CLCL}$		
t_{EHSH}	Khoảng thời gian từ lúc P2.7 (ENABLE) lên mức cao đến khi V_{pp} chuyển đến giá trị điện áp lập trình (5V/12V)	$48t_{CLCL}$		
t_{SHGL}	Khoảng thời gian từ lúc V_{pp} chuyển lên giá trị điện áp lập trình đến khi chân $\overline{PROG}$ xuống mức thấp	10		µs
t_{GHSL} (1)	Khoảng thời gian từ lúc chân $\overline{PROG}$ lên mức cao đến khi V_{pp} chuyển xuống giá trị điện áp thấp	10		µs
t_{GLGH}	Độ rộng xung lập trình	1	110	µs
t_{AVQV} (2)	Khoảng thời gian từ lúc đưa địa chỉ cho đến lúc có thể đọc dữ liệu		$48t_{CLCL}$	
t_{ELQV} (2)	Khoảng thời gian từ lúc chân P2.7 (ENABLE) xuống mức thấp đến khi có thể đọc dữ liệu		$48t_{CLCL}$	
t_{EHQZ} (2)	Khoảng thời gian từ lúc chân P2.7 (ENABLE) lên mức cao đến khi thả nổi đường dữ liệu	0	$48t_{CLCL}$	
t_{GHBL}	Khoảng thời gian từ lúc chân $\overline{PROG}$ lên mức cao đến khi chân P3.4 (BUSY) xuống mức thấp		1.0	µs
t_{WC}	Chu kỳ ghi byte		2.0	ms

(1) Chỉ dùng cho điện áp lập trình 12V

(2) Dùng cho chế độ kiểm tra

(Tham khảo thêm một mạch lập trình cho AT89C51 tại Phụ lục 3)

Sưu tầm bởi: www.daihoc.com.vn

2.7.3. Các đặc tính của AT89C51

2.7.3.1. Đặc tính DC

Bảng 1.11 – Đặc tính DC của AT89C51
T = - 40 – 85^{0}C; V_{CC} = 5V ± 20%

Ký hiệu	Mô tả	Điều kiện	Min	Max	Đơn vị
V_{IL}	Điện áp ngõ vào mức thấp	Trừ $\overline{EA}$	-0.5	0.2 V_{CC} - 0.1	V
V_{IL1}	Điện áp ngõ vào mức thấp	$\overline{EA}$	-0.5	0.2 V_{CC} - 0.3	V
V_{IH}	Điện áp ngõ vào mức cao	Trừ XTAL1, RST	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Điện áp ngõ vào mức cao	XTAL1, RST	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Điện áp ngõ ra mức thấp (1) (Port 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Điện áp ngõ ra mức thấp (1) (Port 0,ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Điện áp ngõ ra mức cao (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Điện áp ngõ ra mức cao (Port 0 trong chế độ địa chỉ dữ liệu đa hợp)	I_{OH} = -800 µA V_{CC} = 5V ± 10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Dòng ngõ vào mức 0 (Port 1,2,3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Dòng điện xảy ra khi chuyển mức logic từ 1 xuống 0 (P1, 2, 3)	V_{IN} = 2V, V_{CC} = 5V ± 10%		-650	µA
I_{LI}	Dòng điện ngõ vào	0.45 < V_{IN} < V_{CC}		±10	µA
R_{RST}	Điện trở kéo xuống tại ngõ Reset		50	300	KΩ
C_{IO}	Điện dung tại các chân	Tần số = 1 MHz T_A = 25°C		10	pF
I_{CC}	Dòng tối thiểu của nguồn cung cấp	Chế độ thường 12 MHz		20	mA
		Chế độ nghỉ 12 MHz		5	mA
	Chế độ nguồn giảm (2)	V_{CC} = 6V		100	µA
		V_{CC} = 3V		40	µA

(1) Ở chế độ thường, I_{OL} xác định như sau:

- I_{OLmax} tại mỗi chân là 10 mA.

Sưu tầm bởi: www.daihoc.com.vn

- I_{OLmax} tại mỗi port 8 bit: 26 mA cho Port 0 và 15 mA cho Port 1,2,3.
- I_{OLmax} tại tất cả các ngõ vào: 71 mA.

Nếu I_{OL} không thoả mãn các điều kiện trên, điện áp V_{OL} có thể sẽ lớn hơn giá trị trong bảng 1.11

(2) Điện áp V_{cc} tối thiểu trong chế độ nguồn giảm là 2V.

2.7.3.2. Đặc tính AC

Bảng 1.12 – Đặc tính AC của AT89C51

Ký hiệu	Mô tả	Thạch anh 12 MHz		Thạch anh 16 - 24 MHz		Đơn vị
		Min	Max	Min	Max	
$1/t_{CLCL}$	Tần số thạch anh			0	24	MHz
t_{LHLL}	Độ rộng xung ALE	127		$2t_{CLCL}$-40		ns
t_{AVLL}	Khoảng thời gian từ lúc địa chỉ ổn định đến khi ALE xuống mức thấp	43		t_{CLCL}-13		ns
t_{LLAX}	Khoảng thời gian giữ lại địa chỉ sau khi ALE xuống mức thấp	48		t_{CLCL}-20		ns
t_{LLIV}	Khoảng thời gian từ lúc ALE xuống mức thấp đến khi mã lệnh vào hợp lệ		233		$4t_{CLCL}$-65	ns
t_{LLPL}	Khoảng thời gian từ lúc ALE xuống mức thấp đến khi $\overline{PSEN}$ xuống mức thấp	43		t_{CLCL}-13		ns
t_{PLPH}	Độ rộng xung $\overline{PSEN}$	205		$3t_{CLCL}$-20		ns
t_{PLIV}	Khoảng thời gian từ lúc $\overline{PSEN}$ xuống mức thấp đến khi mã lệnh vào hợp lệ		145		$3t_{CLCL}$-45	ns
t_{PXIX}	Khoảng thời gian giữ lại mã lệnh sau tín hiệu $\overline{PSEN}$	0		0		ns
t_{AVIV}	Khoảng thời gian từ lúc đặt địa chỉ đến khi mã lệnh vào hợp lệ		312		$5t_{CLCL}$-55	ns
t_{PXIZ}	Khoảng thời gian thả nổi ngõ vào mã lệnh sau tín hiệu $\overline{PSEN}$				t_{CLCL}-10	ns

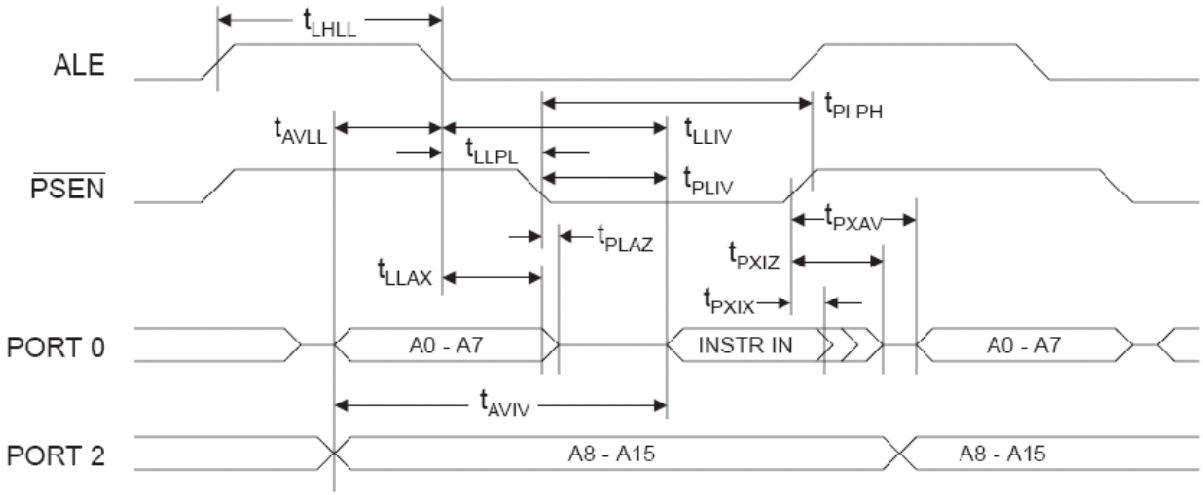
Sưu tầm bởi: www.daihoc.com.vn

t_{PXAV}	Khoảng thời gian từ tín hiệu $\overline{PSEN}$ đến khi địa chỉ hợp lệ	75		t_{CLCL}-8		ns
t_{PLAZ}	Khoảng thời gian từ lúc $\overline{PSEN}$ xuống mức thấp đến khi thả nổi địa chỉ		10		10	ns
t_{RLRH}	Độ rộng xung $\overline{RD}$	400		$6t_{CLCL}$-100		ns
t_{WLWH}	Độ rộng xung $\overline{WR}$	400		$6t_{CLCL}$-100		ns
t_{RLDV}	Khoảng thời gian từ lúc $\overline{RD}$ xuống mức thấp đến khi dữ liệu vào hợp lệ		252		$5t_{CLCL}$-90	ns
t_{RHDX}	Khoảng thời gian giữ lại dữ liệu sau tín hiệu $\overline{RD}$	0		0		
t_{RHDZ}	Khoảng thời gian thả nổi dữ liệu sau tín hiệu $\overline{RD}$		97		$2t_{CLCL}$-28	ns
t_{LLDV}	Khoảng thời gian từ lúc ALE xuống mức thấp đến khi dữ liệu hợp lệ		517		$8t_{CLCL}$-150	ns
t_{AVDV}	Khoảng thời gian từ lúc đặt địa chỉ đến khi dữ liệu hợp lệ		585		$9t_{CLCL}$-165	ns
t_{LLWL}	Khoảng thời gian từ lúc ALE xuống mức thấp đến khi $\overline{RD}$ hay $\overline{WR}$ xuống mức thấp	200	300	$3t_{CLCL}$-50	$3t_{CLCL}$+50	ns
t_{AVWL}	Khoảng thời gian từ lúc đặt địa chỉ đến khi $\overline{RD}$ hay $\overline{WR}$ xuống mức thấp	203		$4t_{CLCL}$-75		ns
t_{QVWX}	Khoảng thời gian từ lúc dữ liệu hợp lệ đến khi $\overline{WR}$ chuyển mức logic	23		t_{CLCL}-20		ns
t_{QVWH}	Khoảng thời gian từ lúc dữ liệu hợp lệ đến khi $\overline{WR}$ lên mức cao	433		$7t_{CLCL}$-120		ns
t_{WHQX}	Khoảng thời gian giữ	33		t_{CLCL}-20		ns

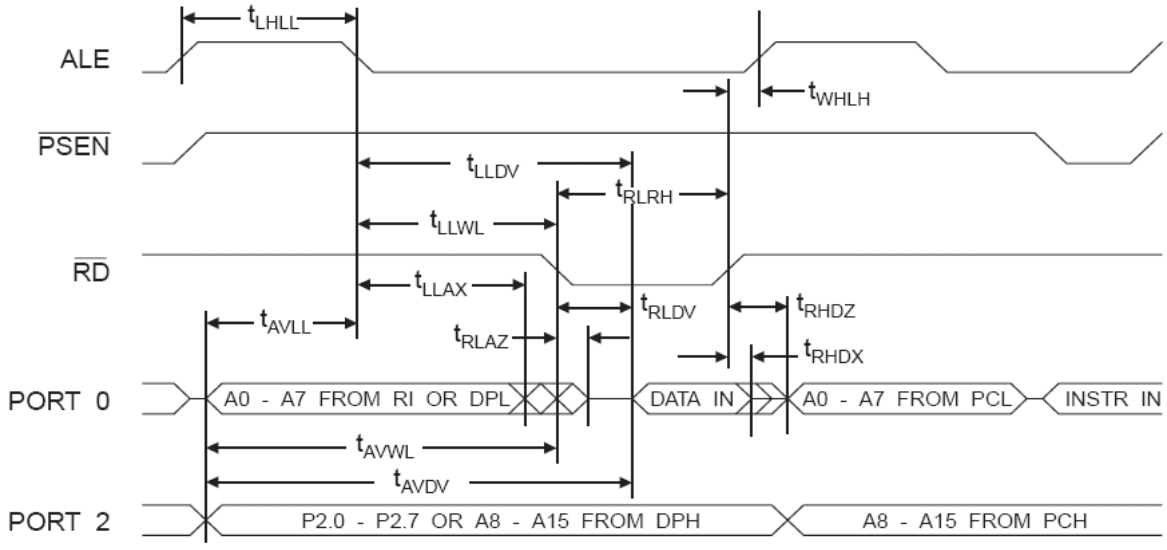
Sưu tầm bởi: www.daihoc.com.vn

	lại dữ liệu sau tín hiệu $\overline{WR}$					
t_{RLAZ}	Khoảng thời gian từ lúc $\overline{RD}$ xuống mức thấp đến khi thả nổi địa chỉ		0		0	ns
t_{WHLH}	Khoảng thời gian từ lúc $\overline{RD}$ hay $\overline{WR}$ lên mức cao đến khi ALE lên mức cao	43	123	t_{CLCL}-20	t_{CLCL}+25	ns ns ns ns ns ns ns

Các đặc tính AC được mô tả trong các hình vẽ sau:

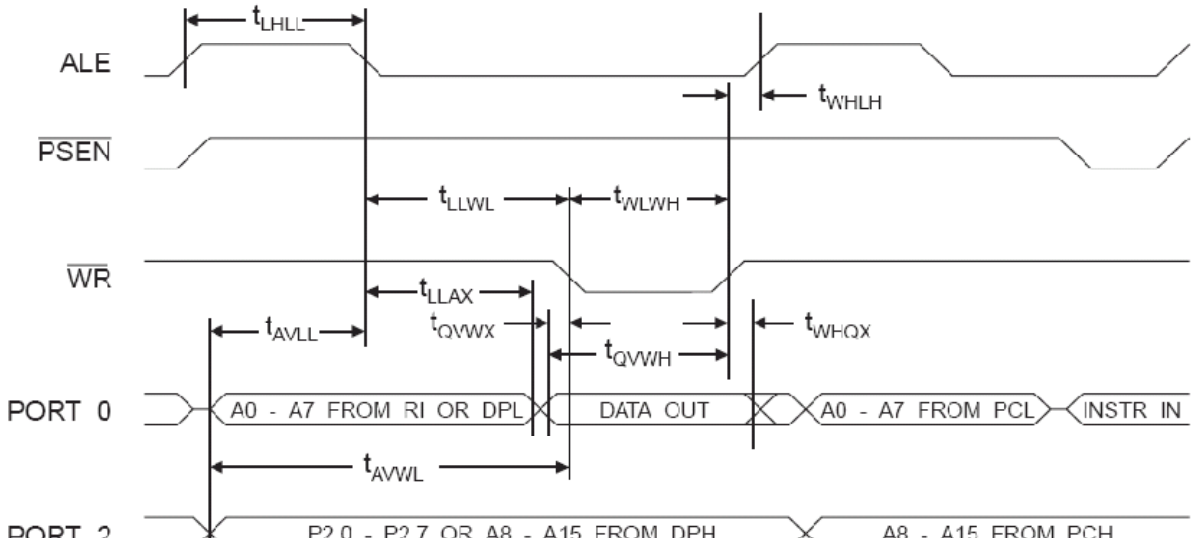


Hình 1.18 – Chu kỳ đọc bộ nhớ chương trình ngoài



Hình 1.19 – Chu kỳ đọc bộ nhớ dữ liệu ngoài

Sưu tầm bởi: www.daihoc.com.vn



Hình 1.20 – Chu kỳ ghi dữ liệu bộ nhớ ngoài

Sưu tầm bởi: www.daihoc.com.vn

BÀI TẬP CHƯƠNG 1

1. Giải thích tại sao thường phải có điện trở kéo lên (pull-up) tại Port 0? Trường hợp nào không cần sử dụng điện trở này?

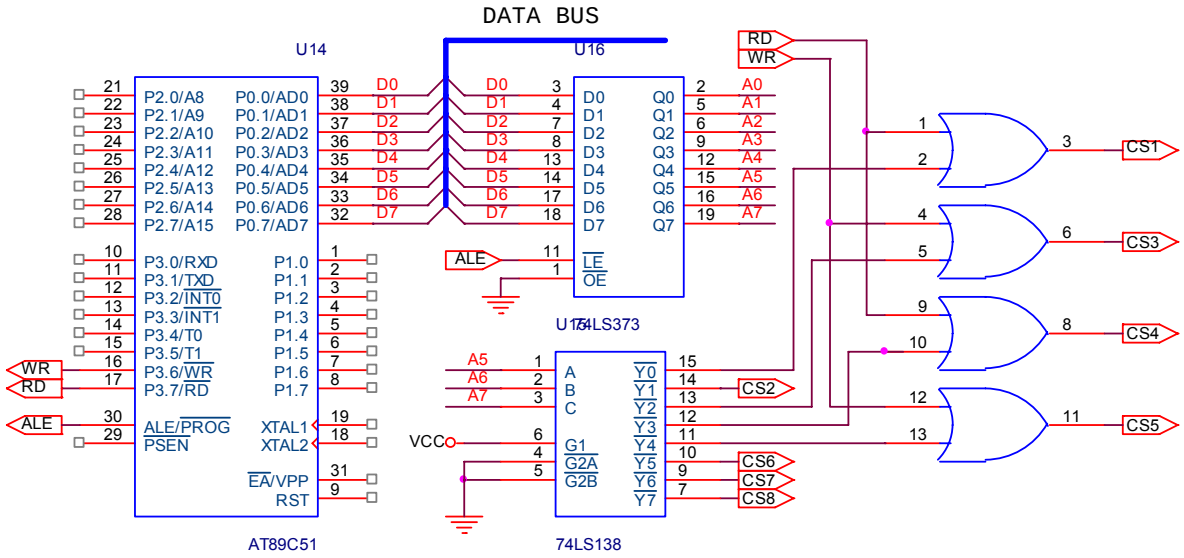
2. Thiết kế mạch giải mã địa chỉ dùng 74LS138 cho 1 RAM 8 KB, 1 RAM 4KB và 1 ROM 16 KB.

3. Cho bản đồ bộ nhớ sau:

Bộ nhớ	Địa chỉ
RAM1	1000h – 1FFFh
RAM2	3800h – 3FFFh
ROM	8000h – 9FFFh

   Lập bản đồ bộ nhớ đầy đủ và thiết kế mạch giải mã địa chỉ theo bản đồ trên.

4. Cho mạch như hình vẽ. Xác định địa chỉ các chân CS. Cho biết chân nào dùng để đọc, chân nào dùng để ghi.



Sưu tầm bởi: www.daihoc.com.vn

Chương 2: LẬP TRÌNH HỢP NGỮ TRÊN VI ĐIỀU KHIỂN MCS-51

Chương này giới thiệu cách thức lập trình trên MCS-51 cũng như giải thích hoạt động của các lệnh sử dụng cho họ MCS-51.

Các ký hiệu cần chú ý:

Rn : các thanh ghi từ R0 – R7 (bank thanh ghi hiện hành)
Ri : các thanh ghi từ R0 – R1 (bank thanh ghi hiện hành)
@Rn : định địa chỉ gián tiếp 8 bit dùng thanh ghi Rn
@DPTR : định địa chỉ gián tiếp 16 bit dùng thanh ghi DPTR
direct : định địa chỉ trực tiếp RAM nội (00h – 7Fh) hay SFR (80h – FFh)
(direct) : nội dung của bộ nhớ tại địa chỉ direct
#data8 : giá trị tức thời 8 bit
#data16 : giá trị tức thời 16 bit
bit : địa chỉ bit của các ô nhớ có thể định địa chỉ bit (00h – 7Fh đối với địa chỉ bit và 20h – 2Fh đối với địa chỉ byte)

1. Các phương pháp định địa chỉ

❖ Định địa chỉ trực tiếp

Định địa chỉ trực tiếp chỉ dùng cho các thanh ghi chức năng đặc biệt và RAM nội của 8951. Giá trị địa chỉ trực tiếp 8 bit được thêm vào phía sau mã lệnh. Nếu địa chỉ trực tiếp từ 00h – 7Fh thì đó là RAM nội của 8951 (128 byte), còn địa chỉ từ 80h – FFh là địa chỉ các thanh ghi chức năng đặc biệt (xem bảng 1.2, chương 1).

Các lệnh sau có kiểu định địa chỉ trực tiếp:

```
MOV A, P0
MOV A, 30h
```

Lệnh đầu tiên chuyển nội dung từ Port 0 vào thanh ghi A. Khi biên dịch, chương trình sẽ thay thế từ gợi nhớ P0 bằng địa chỉ trực tiếp của Port 0 (80h) và đưa vào byte 2 của mã lệnh. Lệnh thứ hai chuyển nội dung của RAM nội có địa chỉ 30h vào thanh ghi A.

❖ Định địa chỉ gián tiếp

Định địa chỉ gián tiếp có thể dùng cho cả RAM nội và RAM ngoại. Trong chế độ này, địa chỉ của RAM xác định thông qua một thanh ghi (R0, R1, SP cho địa chỉ 8 bit và DPTR cho địa chỉ 16 bit). Các lệnh sau có kiểu địa chỉ gián tiếp:

```
MOV A, @R0
```


Sưu tầm bởi: www.daihoc.com.vn

```
MOVX A, @DPTR
```

Lệnh đầu tiên chuyển nội dung cúa RAM nội có địa chỉ chứa trong thanh ghi R0 vào thanh ghi A (giả sử R0 = 30h thì chuyển nội dung của ô nhớ 30h). Lệnh thứ hai chuyển nội dung RAM ngoại vào thanh ghi A (địa chỉ RAM chứa trong DPTR).

❖ **Định địa chỉ thanh ghi**

Các thanh ghi từ R0 – R7 có thể truy xuất bằng cách định địa chỉ trực tiếp hay gián tiếp như trên. Ngoài ra, các thanh ghi này còn có thể truy xuất bằng cách dùng 3 bit trong mã lệnh để chọn 1 trong 8 thanh ghi (8 thanh ghi này có địa chỉ trực tiếp thay đổi tuỳ theo bank thanh ghi đang sử dụng).

❖ **Định địa chỉ tức thời**

Giá trị của một hằng số có thể đưa trực tiếp vào mã lệnh của chương trình. Trong hợp ngữ, hằng số được xác định bằng cách sử dụng dấu #.

Lệnh:

```
MOV A, #10h
```

có chế độ địa chỉ tức thời.

❖ **Định địa chỉ chỉ số**

Quá trình định địa chỉ chỉ số chỉ có thể dùng cho bộ nhớ chương trình, được dùng để đọc dữ liệu trong các bảng tìm kiếm. Chế độ này thường dùng một thanh ghi nền 16 bit (PC hay DPTR) để chỉ vị trí của bảng và thanh ghi A chỉ vị trí của các phần tử trong bảng.

2. Các vấn đề liên quan khi lập trình hợp ngữ

2.1. Cú pháp lệnh

Một lệnh trong chương trình hợp ngữ có dạng như sau:

Nhãn	**Lệnh**	**Toán hạng**	**Chú thích**
A:	MOV	A, #10h	; Đưa giá trị 10h vào thanh ghi A
LED	EQU	30h	; Định nghĩa ô nhớ chứa mã led
On_Led	BIT	00h	; Cờ trạng thái led

Trường nhãn định nghĩa các ký hiệu (có thể là địa chỉ trong chương trình, các hằng dữ liệu, tên đoạn hay các cấu trúc lập trình). Trường nhãn không bắt đầu bằng số và không trùng với các từ khoá có sẵn.

Trường lệnh chứa các từ gợi nhớ cho các lệnh của MCS-51 hay các lệnh giả dùng cho chương trình dịch.

Sưu tầm bởi: www.daihoc.com.vn

Trường toán hạng chứa các thông số liên quan đến lệnh đang sử dụng.

Trường chú thích dùng để ghi chú trong chương trình hợp ngữ. Trường này phải được bắt đầu bằng dấu **;** và chương trình dịch sẽ bỏ qua các từ đặt sau dấu **;**.

Lưu ý rằng các chương trình dịch không phân biệt chữ hoa và chữ thường.

2.2. Khai báo dữ liệu

- Khi khai báo hằng số, chữ **h** cuối cùng xác định hằng số là số thập lục phân; chữ **b** cuối cùng xác định số nhị phân và chữ **d** cuối (hay không có) xác định số thập phân. Lưu ý rằng đối với số thập lục phân, khi bắt đầu bằng chữ A $\rightarrow$ F thì phải thêm số 0 vào phía trước.

Ví dụ:

1010b ; Số nhị phân

1010h ; Số thập lục phân

1010 ; Số thập phân

0F0h ; Số thập lục phân nhưng bắt đầu bằng chữ F nên phải thêm vào phía trước số 0.

- Khi dùng dấu # phía trước một con số, đó chính là dữ liệu tức thời còn nếu không dùng dấu # thì đó là địa chỉ của ô nhớ. Lưu ý rằng khi dùng RAM nội thì chỉ dùng địa chỉ từ 00 – 7Fh còn vùng địa chỉ từ 80h – 0FFh dùng cho các thanh ghi chức năng đặc biệt. Đối với họ 89x52, RAM nội có 256 byte thì các byte địa chỉ cao (từ 80h – 0FFh) không thể truy xuất trực tiếp mà phải truy xuất gián tiếp.

Ví dụ:

```
MOV A,30h      ; Chuyển nội dung ô nhớ 30h vào A
MOV A,#30h     ; Chuyển giá trị 30h vào A
MOV A,80h      ; Chuyển nội dung Port 0 vào A (80h là
               ; địa chỉ Port 0
MOV R0,#80h    ; Chuyển nội dung ô nhớ 80h vào A (chỉ
MOV A,@R0      ; dùng cho họ 89x52)
```

- Để định nghĩa trước một vùng nhớ trong bộ nhớ chương trình, có thể dùng các chỉ dẫn **DB** (define byte – định nghĩa 1 byte) hay **DW** (define word – định nghĩa 2 byte).

Ví dụ: Định nghĩa trước dữ liệu cho led như sau:

Sưu tầm bởi: www.daihoc.com.vn

```
Led: DB   01h,02h,04h,08h,10h,20h,40h,80h
```

Đoạn chương trình này xác định tại nhãn Led có chứa các giá trị lần lượt từ 01h đến 80h. Nếu nhãn Led đặt tại địa chỉ 100h thì giá trị tương ứng như sau:

Địa chỉ	Giá trị
100h	01h
101h	02h
102h	04h
103h	08h
104h	10h
105h	20h
106h	40h
107h	80h

- Để dễ nhớ và dễ hiểu khi lập trình, các chương trình dịch cho phép dùng các ký tự thay thế cho các ô nhớ bằng các lệnh giả EQU, BIT.

Ví dụ:

```
LED  EQU  30h
ON_LED   BIT  00h
```

Giả sử chương trình hợp ngữ có các lệnh sau:

```
MOV  A,LED
SETB ON_LED
```

Khi biên dịch, chương trình dịch sẽ tự động chuyển thành dạng lệnh sau:

```
MOV  A,30h
SETB 00h
```

2.3. Các toán tử

❖ Các toán tử số học:

Bao gồm các toán tử +, -, *, /, **mod**.

Ví dụ: Các lệnh sau tương đương:

```
MOV A,#12h          MOV A,#10h + 2h
MOV A,#21 mod 2     MOV A,#1
MOV A,#12/4         MOV A,#3
```


Sưu tầm bởi: www.daihoc.com.vn

❖ **Các toán tử logic:**

Bao gồm các toán tử: **OR**, **AND**, **NOT**, **XOR**.

Ví dụ: Các lệnh sau tương đương:

```
MOV A,#01h          MOV A,#03h AND 91h
MOV A,#-5           MOV A,#NOT 5
MOV A,#24h          MOV A,#20h OR 04h
```

❖ **Các toán tử quan hệ:**

Bao gồm các toán tử: **EQ** (=), **NE** (<>), **LT** (<), **LE** (<=), **GT** (>), **GE** (>=). Lưu ý rằng khí sử dụng các toán tử quan hệ, chỉ có 2 kết quả: sai (= 0) hay đúng (= FFh hay FFFFh tuỳ theo kết quả là 8 bit hay 16 bit).

Ví dụ: Các lệnh sau tương đương:

```
MOV A,#00h          MOV A,#5 EQ 6
MOV A,#0FFh         MOV A,#7 < 9
MOV DPTR,#0FFFFh    MOV DPTR,#5 NE 6
```

❖ **Các toán tử khác:**

Bao gồm các toán tử: **SHR** (dịch phải), **SHL** (dịch trái), **HIGH** (byte cao), **LOW** (byte thấp), **(**, **)**.

Ví dụ: Các lệnh sau tương đương:

```
MOV A,#06h          MOV A,#03h SHL 1
MOV A,#01h          MOV A,#HIGH 0123h
MOV A,#02h          MOV A,#LOW 0102h
```

2.4. Cấu trúc chương trình

- Cấu trúc chương trình hợp ngữ cơ bản mô tả như sau:

```
ORG 0000h       ; Đặt lệnh LJMP main tại địa chỉ
LJMP main       ; 0000h (địa chỉ bắt đầu khi
                ; reset AT89C51)
ORG 0030h       ; Vùng địa chỉ 0003h – 002Fh
Main:           ; dùng để chứa các chương trình
                ; phục vụ ngắt
```


Sưu tầm bởi: www.daihoc.com.vn

```
…
CALL Subname
…
;---------------
Subname:
…
…
RET
END       ; kết thúc chương trình
```

Các lệnh giả ORG cho biết lệnh phía sau đặt tại vị trí nào trong chương trình. Lưu ý rằng khi khởi động, chương trình trong AT89C51 sẽ được thực thi tại địa chỉ 0000h nên thông thường tại địa chỉ này sẽ có lệnh ***LJMP main*** để xác định chương trình chính sẽ bắt đầu tại nhãn main.

Các dấu **;** xác định đây là một chú thích, chương trình dịch sẽ bỏ qua tất cả các phần nằm sau dấu **;**.

Các địa chỉ từ 0003h – 002Fh phục vụ cho mục đích xử lý ngắt nên không sử dụng. Tuy nhiên, nếu chương trình không cần xử lý ngắt thì cũng có thể sử dụng luôn vùng địa chỉ này.

- Khi thực hiện soạn thảo chương trình hợp ngữ, có thể dùng bất kỳ chương trình soạn thảo không định dạng (như NotePad, Norton Commander, …) và thường lưu file với phần mở rộng .asm, .a51 (tuỳ theo chương trình dịch).
- Sau khi soạn thảo, dùng một chương trình dịch để chuyển từ file văn bản thành file .hex (có thể dùng sim51.exe, oh.exe). Ngoài ra, có nhiều chương trình soạn thảo bao gồm cả chương trình dịch bên trong (xem thêm phần phụ lục).
- Khi dịch ra file .hex, dùng một mạch nạp để nạp file .hex vào AT89C51 (xem thêm phụ lục).

Sưu tầm bởi: www.daihoc.com.vn

3. Tập lệnh

3.1. Nhóm lệnh chuyển dữ liệu

3.1.1. RAM nội

Các lệnh trong nhóm lệnh chuyển dữ liệu trong RAM nội mô tả như bảng sau:

Bảng 2.1 – Các lệnh chuyển dữ liệu trong RAM nội

Lệnh	Hoạt động	Chế độ địa chỉ				Chu kỳ thực thi
		Tức thời	Trực tiếp	Gián tiếp	Thanh ghi	
MOV A,(byte)	A = (byte)	x	x	x	x	1
MOV (byte),A	(byte) = A		x	x	x	1
MOV (byte1),(byte2)	(byte1) = (byte2)	x	x	x	x	2
MOV DPTR,#data16	DPTR = data16	x				2
PUSH (byte)	SP = SP + 1 [SP] = (byte)		x			2
POP (byte)	(byte) = [SP] SP = SP – 1		x			2
XCH A,(byte)	Chuyển đổi dữ liệu giữa ACC và (byte)		x	x	x	1
XCHD A,@Ri	Chuyển đổi 4 bit thấp giữa ACC và @Ri			x		1

❖ **Lệnh MOV (Move):**

Di chuyển dữ liệu giữa các thanh ghi và bộ nhớ trong đó 128 byte RAM có địa chỉ từ 80h – FFh (chỉ có trong 8x52) chỉ có thể truy xuất bằng cách định địa chỉ gián tiếp. Các dạng của lệnh MOV như sau:

```
MOV A, Rn       ; Chuyển nội dung thanh ghi Rn vào thanh ghi A
MOV Rn, A       ; Chuyển nội dung thanh ghi A vào thanh ghi Rn
MOV A, direct   ; Chuyển nội dung ô nhớ trực tiếp vào thanh ghi A
MOV direct, A   ; Chuyển nội dung thanh ghi A vào ô nhớ trực tiếp
MOV A,@Ri       ; Chuyển nội dung của ô nhớ có địa chỉ chứa trong Ri vào A
MOV @Ri,A       ; Chuyển nội dung củaA vào ô nhớ có địa chỉ chứa trong Ri
```


Sưu tầm bởi: www.daihoc.com.vn

MOV A, #data8 ; Chuyển giá trị 8 bit vào A
MOV Rn, direct; Chuyển nội dung ô nhớ trực tiếp vào thanh ghi Rn
MOV direct, Rn ; Chuyển nội dung thanh ghi Rn vào ô nhớ trực tiếp
MOV Rn, #data8; Chuyển giá trị 8 bit vào Rn
MOV direct, direct; Chuyển nội dung giữa 2 ô nhớ trực tiếp
MOV direct, @Ri; Chuyển nội dung của ô nhớ có địa chỉ chứa trong Ri vào ô nhớ trực tiếp
MOV @Ri, direct; Chuyển nội dung của ô nhớ trực tiếp vào ô nhớ có địa chỉ chứa trong Ri
MOV direct, #data8; Chuyển giá trị 8 bit vào ô nhớ trực tiếp
MOV @Ri, #data8; Chuyển giá trị 8 bit vào ô nhớ có địa chỉ chứa trong Ri
MOV C, bit ; Chuyển giá trị 1 bit vào cờ C
MOV bit, C ; Chuyển giá trị cờ C vào 1 bit
MOV DPTR, #data16 ; Chuyển giá trị tức thời 16 bit vào thanh ghi DPTR

Trong lệnh MOV, khi sử dụng địa chỉ trực tiếp từ 80h – FFh thì có thể thay bằng các từ gợi nhớ của các thanh ghi chức năng đặc biệt.

Ví dụ: lệnh MOV A, 80h có thể thay thế bằng lệnh MOV A, P0 (xem thêm bảng 1.2, chương 1).

Khi lệnh MOV thực hiện truy xuất bit, các bit có thể là địa chỉ trực tiếp (từ 00h – 7Fh) hay các từ gợi nhớ đã được định nghĩa. Các bit được định nghĩa trước mô tả như sau:

Bảng 2.2 – Các bit được định nghĩa trước trong 8951

Thanh ghi	Từ gợi nhớ	Địa chỉ bit	Thanh ghi	Từ gợi nhớ	Địa chỉ bit
A	ACC.0 – ACC.7	E0h – E7h	B	B.0 – B.7	F0h – F7h
PSW	CY hay C	D7h	SCON	SM0	9Fh
	AC	D6h		SM1	9Eh
	F0	D5h		SM2	9Dh
	RS1	D4h		REN	9Ch
	RS0	D3h		TB8	9Bh
	OV	D2h		RB8	9Ah
	P	D0h		TI	99h
				RI	98h
Các thanh ghi Port	P0.0 – P0.7	80h – 87h	IP	PS	BCh
	P1.0 – P1.7	90h – 97h		PX1	BBh
	P2.0 – P2.7	A0h – A7h		PT1	BAh
	P3.0 – P3.7	B0h – B7h		PX0	B9h
				PT0	B8h

Sưu tầm bởi: www.daihoc.com.vn

IE	EA	AFh	TCON	TF1	8Fh
	ES	ACh		TR0	8Eh
	EX1	ABh		TF0	8Dh
	ET1	AAh		TR0	8Ch
	EX0	A9h		IE1	8Bh
	ET0	A8h		IT1	8Ah
				IE0	89h
				IT0	88h

Ví dụ: Lệnh MOV C, P0.0 có thể thay bằng lệnh MOV C, 80h.

❖ **Lệnh PUSH / POP:**

Các lệnh này cho phép cất hay lấy nội dung của stack. Khi thực hiện lệnh PUSH, nội dung thanh ghi SP tăng lên 1 và cất byte vào stack. Khi thực hiện lệnh POP, byte được lấy ra từ stack và sau đó giảm SP 1 giá trị. Lưu ý rằng khi sử dụng 8951, do bộ nhớ nội chỉ có 128 byte (00h – 7Fh) nên giá trị của SP không được vượt quá 7Fh (nếu vượt qua thì dữ liệu sẽ bị mất khi dùng lệnh PUSH và dữ liệu không xác định khi dùng lệnh POP). Còn đối với 8x52, do RAM nội là 256 byte nên không có hiện tượng này.

Các dạng của lệnh PUSH / POP:

PUSH direct ; Cất vào stack
POP direct ; Lấy dữ liệu từ stack

Lưu ý rằng lệnh PUSH và POP chỉ dùng cho địa chỉ trực tiếp nên không thể thực hiện lệnh PUSH Rn do thanh ghi Rn có 4 địa chỉ khác nhau tuỳ theo bank thanh ghi sử dụng.

Xét thanh ghi R0: 4 địa chỉ của R0 ứng với 4 bank là 00h, 08h, 10h, 18h. Mặc định khi reset, bank 0 được sử dụng nên các thanh ghi Rn có địa chỉ từ 00h – 07h. Khi đó thay vì dùng lệnh PUSH R0, ta có thể thay bằng lệnh PUSH 00h.

❖ **Lệnh XCH / XCHD (Exchange / Exchange Digit):**

Lệnh XCH / XCHD dùng để hoán chuyển 8 bit / 4 bit thấp của thanh ghi A với các thanh ghi khác hay bộ nhớ (lệnh XCHD chỉ dùng cho bộ nhớ nội định địa chỉ gián tiếp). Các dạng lệnh như sau:

XCH A,(byte) ; Hoán chuyển 8 bit
XCHD A,@Ri ; Hoán chuyển 4 bit thấp

Ví dụ: Xét đoạn lệnh:

```
MOV A, #30h   ; A = 30h
```


Sưu tầm bởi: www.daihoc.com.vn

```
MOV R0, #54h  ; R0 = 54h
MOV 30h, #20h ;  Ô  nhớ  30h  chứa  giá  trị  20h  hay
              ;(30h) = 20h
XCH A, R0     ; Hoán chuyển giữa A và R0 → A = 54h
              ; và R0 = 30h
XCHD A, @R0   ; Chuyển 4 bit thấp giữa A và ô nhớ
              ; R0 = 30h → @R0: nội dung ô nhớ 30h → 20h
              ;Chuyển 4 bit thấp→ A = 50h và (30h) = 24h
```

3.1.2. RAM ngoại

Các lệnh trong nhóm lệnh chuyển dữ liệu trong RAM ngoại mô tả như sau:

Bảng 2.3 – Các lệnh chuyển dữ liệu trong RAM ngoại

Lệnh	Hoạt động	Chu kỳ thực thi
MOVX A, @Ri	Đọc nội dung từ RAM ngoại tại địa chỉ Ri	2
MOVX @Ri, A	Ghi vào RAM ngoại tại địa chỉ Ri	2
MOVX A, @DPTR	Đọc nội dung từ RAM ngoại tại địa chỉ DPTR	2
MOVX @DPTR, A	Ghi vào RAM ngoại tại địa chỉ DPTR	2

(MOVX : Move eXternal)

Đối với các lệnh đọc / ghi dữ liệu của RAM ngoại, chỉ cho phép thực hiện định địa chỉ gián tiếp. Khi địa chỉ RAM là 8 bit thì dùng thanh ghi R0 hay R1 còn nếu là địa chỉ 16 bit thì phải dùng thanh ghi DPTR. Lưu ý rằng khi dùng địa chỉ 8 bit thì các bit địa chỉ cao không sử dụng nên Port 2 có thể sử dụng cho mục đích khác nhưng nếu dùng địa chỉ 16 bit thì Port 2 chỉ có nhiệm vụ là xuất 8 bit địa chỉ cao.

Khi thực hiện lệnh đọc từ RAM ngoại, chân $\overline{RD}$ sẽ xuống mức thấp còn khi thực hiện lệnh ghi, chân $\overline{WR}$ xuống mức thấp.

3.1.3. Bảng tìm kiếm

Các lệnh trong nhóm lệnh tìm kiếm dữ liệu trong bảng mô tả như sau:

Bảng 2.4 – Các lệnh tìm kiếm dữ liệu

Lệnh	Hoạt động	Chu kỳ thực thi
MOVC A, @A + DPTR	Đọc nội dung bộ nhớ chương trình tại địa chỉ A + DPTR	2
MOVC A, @A +PC	Đọc nội dung bộ nhớ chương trình tại địa chỉ A + PC	2

(MOVC: Move Code)

Sưu tầm bởi: www.daihoc.com.vn

Các lệnh này cho phép tìm kiếm dữ liệu đã định nghĩa sẵn trong bộ nhớ chương trình (nếu bộ nhớ chương trình là ROM ngoại thì tín hiệu đọc là $\overline{PSEN}$). Các thanh ghi DPTR hay PC (Program Counter: bộ đếm chương trình – xác định địa chỉ của lệnh kế tiếp sẽ thực hiện) chứa vị trí nền của các bảng tìm kiếm còn thanh ghi A chứa vị trí của phần tử (thông thường kích thước 1 phần tử trong bảng tìm kiếm là 1 byte).

Ví dụ: Lấy phần tử thứ 2 trong bảng LED_7S:

```
MOV A, #2          ; Phần tử thứ 2
MOV DPTR, #LED_7S  ; Địa chỉ nền của bảng tìm kiếm
MOVC A, @A + DPTR  ; Đọc nội dung phần tử
……
LED_7S: DB data8, data8, data8, data8, … ; Nội dung bảng tìm kiếm có thể đặt tuỳ ý trong bộ nhớ chương trình
```

Để sử dụng thanh ghi PC tìm kiếm dữ liệu, quá trinh tìm kiếm phải thực hiện thông qua chương trình con và bảng phải được đặt ngay sau chương trình con.

Ví dụ: Lấy phần tử thứ 2 trong bảng LED_7S:

```
MOV A, #2          ; Phần tử thứ 2
CALL Read_Led7s
……
Read_Led7s:
MOVC A, @A+PC
RET
LED_7S: DB 0, data8, data8, data8, data8, … ; Nội dung bảng tìm kiếm
```

Lưu ý rằng trong đoạn lệnh trên, khi thực hiện lệnh MOVC, thanh ghi PC sẽ chỉ đến lệnh kế tiếp là lệnh RET chứ không phải bảng LED_7S. Do đó, bảng tìm kiếm trong trường hợp này sẽ không có phần tử 0 mà bắt đầu tại phần tử 1. Để chương trình giống như cách thực hiện dùng DPTR, cần phải thay đổi chương trình con như sau:

Ví dụ: Lấy phần tử thứ 2 trong bảng LED_7S:

```
MOV A, #2          ; Phần tử thứ 2
CALL Read_Led7s
……
Read_Led7s:
INC A                  ; Tăng nội dung A lên 1 để hiệu chỉnh vị trí bảng
MOVC A, @A+PC
RET
```


Sưu tầm bởi: www.daihoc.com.vn

```
LED_7S: DB data8, data8, data8, data8, … ; Nội dung bảng tìm kiếm
```

3.2. Nhóm lệnh xử lý bit

Họ MCS-51 chứa một bộ xử lý bit hoàn chỉnh. RAM nội có 128 bit có thể xử lý bit và các thanh ghi chức năng đặc biệt có thể hỗ trợ lên tới 128 bit (các bit trong SFR xem tại bảng 2.2). Các địa chỉ bit từ 00h – 7Fh nằm trong RAM nội còn các địa chỉ từ 80h – FFh nằm trong SFR.

Các lệnh trong nhóm lệnh logic mô tả như trong bảng sau:

Bảng 2.5 – Các lệnh logic

Lệnh	Hoạt động	Chu kỳ thực thi
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	Bit = C	2
CLR C	C = 0	1
CLR bit	Bit = 0	1
SETB C	C = 1	1
SETB bit	Bit = 1	1
CPL C	C = NOT C	1
CPL bit	Bit = NOT bit	1
JC rel	Nhảy đến nhãn rel nếu C = 1	2
JNC rel	Nhảy đến nhãn rel nếu C = 0	2
JB bit,rel	Nhảy đến nhãn rel nếu bit = 1	2
JNB bit,rel	Nhảy đến nhãn rel nếu bit = 0	2
JBC bit,rel	Nhảy đến nhãn rel nếu bit = 1 và sau đó xoá bit	2

ANL: And logic; ORL: Or logic; CLR: Clear; CPL: Complement

Bit: các bit trong RAM nội từ 00h – 7Fh hay trong SFR theo bảng 2.2

Rel: địa chỉ tương đối (cho phép trong vùng từ -128 ÷ 127 byte trong bộ nhớ chương trình)

Ví dụ: Chuyển từ bit 00h vào P1.0

```
MOV C, 00h    ; Chuyển bit 00h vào cờ Carry
MOV P1.0, C   ; Chuyển cờ Carry vào P1.0
```

Lưu ý rằng trong tập lệnh logic không có lệnh XOR mà phải thực hiện bằng phần mềm, cụ thể như sau:

Thực hiện lệnh C = C XRL bit:

Sưu tầm bởi: www.daihoc.com.vn

```
JNB bit, next
CPL C
Next:
```

Ngoài ra, các lệnh nhảy trên đều dùng địa chỉ tương đối, nghĩa là chỉ cho phép trong vùng từ -128 ÷ 127 byte. Nếu cần nhảy đến dịa chỉ xa hơn thì phải dùng các lệnh nhảy khác, như mô tả trong phần sau.

3.3. Nhóm lệnh chuyển điều khiển

Nhóm lệnh chuyển điều khiển bao gồm các lệnh nhảy, các lệnh liên quan đến chương trình con, mô tả như sau:

Bảng **2.6** – Các lệnh chuyển điều khiển

Lệnh	Hoạt động	Chu kỳ thực thi
JMP addr	Nhảy tới nhãn addr	2
JMP @A+DPTR	Nhảy tới địa chỉ A + DPTR	2
CALL addr	Gọi chương trình con tại địa chỉ addr	2
RET	Trở về từ chương trình con	2
RETI	Trở về từ chương trình con phục vụ ngắt	2
NOP	Không làm gì cả	1

JMP: Jump

RET: Return

RETI: Return from Interrupt

NOP: No Operation

Lệnh	Hoạt động	Chế độ địa chỉ				Chu kỳ thực thi
		Tức thời	Trực tiếp	Gián tiếp	Thanh ghi	
JZ rel	Nhảy đến nhãn rel nếu A = 0	Chỉ dùng cho thanh ghi A				2
JNZ rel	Nhảy đến nhãn rel nếu A ≠ 0	Chỉ dùng cho thanh ghi A				2
DJNZ (byte),rel	(byte) = (byte) - 1 Nếu (byte) ≠ 0 thì nhảy đến nhãn rel		x		x	2
CJNE A,(byte),rel	Nhảy đến nhãn rel nếu A ≠ (byte)	x	x			2
CJNE (byte), #data8,rel	Nhảy đến nhãn rel nếu (byte) ≠ data8			x	x	2

JZ: Jump if Zero; JNZ: Jump if Not Zero

DJNZ: Decrement and Jump if Not Zero

Sưu tầm bởi: www.daihoc.com.vn

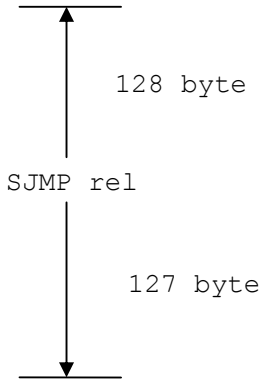
CJNE: Compare and Jump if Not Equal

❖ **Lệnh JMP (Jump):**

Lệnh JMP bao gồm 3 lệnh: LJMP (Long jump), AJMP (Absolute jump) và SJMP (Short jump) cho phép nhảy đến một vị trí bất kỳ trong chương trình.

Lệnh LJMP có kích thước 3 byte trong đó 1 byte mã lệnh và 2 byte chứa địa chỉ nhãn nên phạm vi biểu diễn địa chỉ là 64K (2 byte = 16 bit → phạm vi biểu diễn $2^{16} = 2^6 \times 2^{10} = 64K$). Do đó lệnh LJMP có thể thực hiện nhảy đến bất kỳ vị trí nào trong chương trình và địa chỉ sử dụng trong lệnh LJMP là địa chỉ tuyệt đối.

Lệnh SJMP có kích thước 2 byte trong đó có 1 byte mã lệnh và 1 byte địa chỉ nên phạm vi biểu diễn địa chỉ là 256 byte. Trong lệnh này, địa chỉ sử dụng không phải là địa chỉ tuyệt đối mà là địa chỉ tương đối (khoảng nhảy tính từ vị trí bắt đầu lệnh). Do byte địa chỉ sử dụng phương pháp bù 2 nên phạm vi biểu diễn từ -128 ÷ + 127, nghĩa là phạm vi nhảy của lệnh SJMP chỉ trong phạm vi từ - 128 đến 127 byte. Phạm vi thực hiện mô tả như hình vẽ.

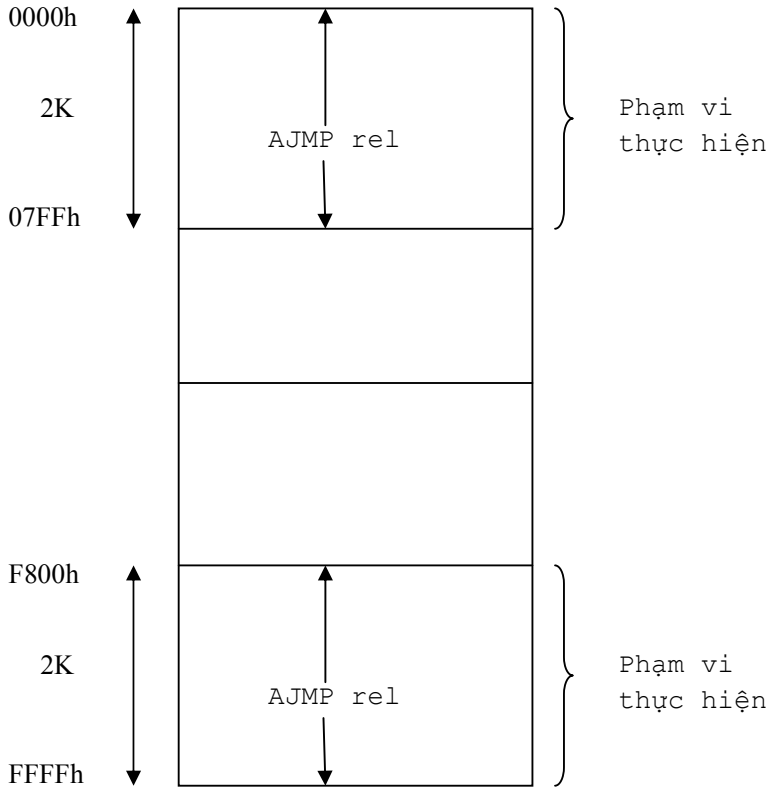


Hình 2.1 – Phạm vi thực hiện của lệnh SJMP

Lệnh AJMP có kích thước 2 byte trong đó địa chỉ chứa trong 11 bit nên phạm vi biểu diễn địa chỉ là 2^{11} (2K). Trong khi đó, vùng địa chỉ tối đa của MCS-51 là 64K nên khi thực hiện lệnh AJMP, 64K chương trình phải chia thành từng vùng 2K (tổng cộng 32 vùng) và lệnh AJMP chỉ có thể thực hiện trong một vùng.

Tuy nhiên, khi lập trình cho MCS-51, thông thường các chương trình dịch đều cho phép sử dụng lệnh **JMP** thay thế cho 3 lệnh trên. Khi biên dịch, chương trình dịch sẽ tự động thay thế bằng các lệnh thích hợp.

Sưu tầm bởi: www.daihoc.com.vn



Hình 2.2 – Phạm vi thực hiện của lệnh AJMP

Lệnh JMP @A + DPTR cho phép chọn các vị trí nhảy khác nhau tuỳ theo giá trị trong thanh ghi A. Địa chỉ nhảy đến chính là tổng giá trị của thanh ghi A và DPTR.

Ví dụ:

```
MOV DPTR, # JUMP_TABLE ; Địa chỉ bảng nhảy
MOV A, INDEX_NUMBER    ; Vị trí nhảy
MOV B, #3              ; x3 do lệnh LJMP
MUL AB                 ; có kích thước 3
JMP @ A + DPTR
.....
JUMP_TABLE:
LJMP LABEL0            ; Vị trí nhảy 0
LJMP LABEL1            ; Vị trí nhảy 1
LJMP LABEL2            ; Vị trí nhảy 2
LJMP LABEL3            ; Vị trí nhảy 3
LJMP LABEL4            ; Vị trí nhảy 4
```


Sưu tầm bởi: www.daihoc.com.vn

❖ **Lệnh CALL, RET, RETI:**

Lệnh CALL dùng để gọi chương trình con, bao gồm 2 lệnh: ACALL (Absolute Call) và LCALL (Long Call). Vị trí có thể gọi lệnh CALL giống như đã xét trong lệnh JMP. Khi lập trình, thông thường các chương trình dịch cũng cho phép thay thế duy nhất bằng lệnh CALL và khi biên dịch, lệnh CALL sẽ được thay thế bằng lệnh ACALL hay LCALL tuỳ theo vị trí gọi lệnh. Lưu ý rằng khi thực hiện lệnh CALL thì trong chương trình con phải kết thúc bằng lệnh RET.

Ngoài ra, khi sử dụng các chương trình con phục vụ ngắt, khi kết thúc phải dùng lệnh RETI. Lệnh RETI và lệnh RET chỉ khác nhau ở chỗ lệnh RETI báo cho hệ thống điều khiển ngắt biết rằng quá trình xử lý ngắt đã thực hiện xong.

❖ **Lệnh JZ, JNZ:**

Lệnh JZ và JNZ dùng để kiểm tra nội dung của thanh ghi A. Lệnh JZ nhảy khi A = 0 và JNZ nhảy khi A ≠ 0. Lưu ý rằng phạm vi nhảy chỉ cho phép trong khoảng từ -128 ÷ 127 byte (giống như khi sử dụng lệnh SJMP).

❖ **Lệnh DJNZ:**

Lệnh DJNZ thường được dùng để tạo vòng lặp. Số lần lặp được chuyển vào thanh ghi đếm ở đầu vòng lặp (thanh ghi đếm có thể dùng bất kỳ thanh ghi nào hay là bộ nhớ).

Ví dụ:

```
     MOV R7, #10 ; Lặp 10 lần
LOOP:
     ……
     ……
     DJNZ R7, LOOP
```

❖ **Lệnh CJNE:**

Lệnh CJNE dùng để so sánh 2 giá trị với nhau, khi 2 giá trị này khác nhau thì sẽ thực hiện lệnh nhảy. Lưu ý rằng trong tập lệnh của MCS-51 không có lệnh lớn hơn hay nhỏ hơn nên chỉ có thể thực hiện các lệnh này bằng cách kết hợp lệnh CJNE và nội dung của cờ Carry.

Trong lệnh CJNE, nếu byte đầu tiên nhỏ hơn byte thứ hai thì CF = 1. Ngược lại (byte đầu tiên lớn hơn hay bằng byte thứ hai) thì CF = 0.

Ví dụ: Kiểm tra nội dung của thanh ghi A, nếu A nhỏ hơn 10 thì xuất giá trị trong thanh ghi A ra Port 1. Ngược lại thì xuất giá trị 10 ra Port 1.

Sưu tầm bởi: www.daihoc.com.vn

```
     CJNE A,#10,Khacnhau; So sánh A với 10
     JMP  Xuat10        ; Nếu A = 10 thì xuất giá trị 10
Khacnhau:
     JC   XuatA         ; Nếu CF = 1 (A < 10) thì xuất nội
Xuat10:                 ; dung trong A ra P1
     MOV  P1,#10
     SJMP Tiep
XuatA:
     MOV  P1,A
Tiep:
```

3.4. Nhóm lệnh logic

Nhóm lệnh logic bao gồm các lệnh liên quan đến xử lý logic theo từng byte, mô tả như sau:

Bảng 2.7 – Các lệnh logic

Lệnh	**Hoạt động**	**Chế độ địa chỉ**				**Chu kỳ thực thi**
		Tức thời	Trực tiếp	Gián tiếp	Thanh ghi	
ANL A,(byte)	A = A AND (byte)	x	x	x	x	1
ANL (byte),A	(byte)=(byte) AND A		x			1
ANL (byte),#data8	(byte)=(byte)AND data8		x			2
ORL A,(byte)	A = A OR (byte)	x	x	x	x	1
ORL (byte),A	(byte)=(byte) OR A		x			1
ORL (byte),#data8	(byte)=(byte) OR data8		x			2
XRL A,(byte)	A = A XOR (byte)	x	x	x	x	1
XRL (byte),A	(byte)=(byte) XOR A		x			1
XRL (byte),#data8	(byte)=(byte) XOR data8		x			2
CLR A	A = 0	Chỉ dùng cho thanh ghi A				1
CPL A	A = NOT A	Chỉ dùng cho thanh ghi A				1
RR A	Quay phải thanh ghi A 1 bit	Chỉ dùng cho thanh ghi A				1
RLC A	Quay phải thanh ghi A và CF 1 bit	Chỉ dùng cho thanh ghi A				1
RL A	Quay trái thanh ghi A 1 bit	Chỉ dùng cho thanh ghi A				1

Sưu tầm bởi: www.daihoc.com.vn

RLC A	Quay trái thanh ghi A và CF 1 bit	Chỉ dùng cho thanh ghi A	1
SWAP A	Đổi vị trí nibble cao và thấp của ACC	Chỉ dùng cho thanh ghi A	1

RL: Rotate Left, RLC: Rotate Left through Carry

RR: Rotate Right; RRC: Rotate Right through Carry

❖ **Lệnh ANL, ORL, XRL:**

Các lệnh logic này thực hiện giống như trong các lệnh xử lý bit nhưng thực hiện trên 8 bit của các thanh ghi hay bộ nhớ. Lệnh XRL còn được dùng để đảo tất cả các bit như sau:

```
XRL P0, #0FFh
```

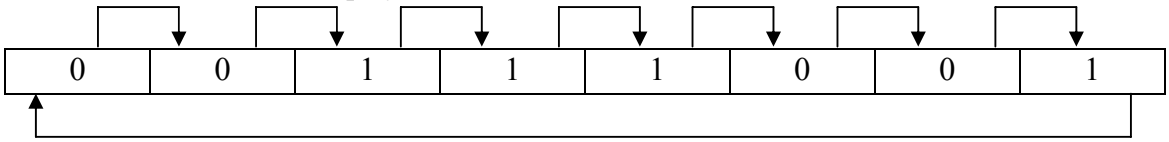
❖ **Lệnh RR, RRC, RL, RLC:**

Các lệnh này dùng để quay phải hay quay trái thanh ghi A 1 bit.

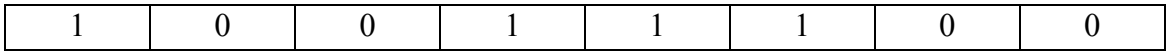
Ví dụ: Giả sử thanh ghi A = 39h (0011 1001b), CF = 1. Nội dung thanh ghi A sau khi thực hiện các lệnh quay tương ứng như sau:

RR A:

Trước khi quay:



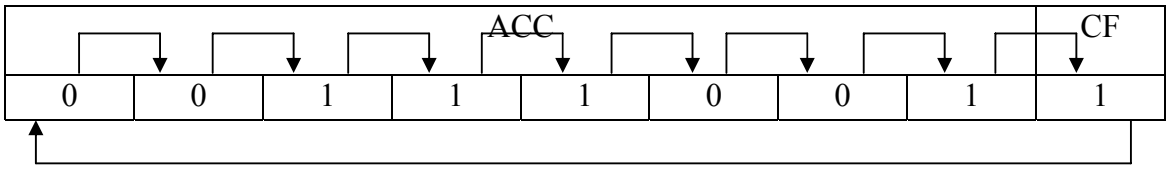
Sau khi quay:



RL A: A = 0111 0010b (72h)

RRC A:

Trước khi quay:



Sau khi quay:

ACC								CF
1	0	0	1	1	1	0	0	1

Sưu tầm bởi: www.daihoc.com.vn

RLC A: A = 0111 0011b (73h); CF = 0

❖ **Lệnh SWAP:**

Lệnh SWAP A dùng để hoán chuyển nội dung 2 nibble trong thanh ghi A.

VÍ Dụ: Nếu nội dung thanh ghi A = 39h thì sau khi thực hiện lệnh SWAP A, nội dung thanh ghi A là 93h.

3.5. Nhóm lệnh số học

Các lệnh trong nhóm lệnh số học mô tả như trong bảng sau:

Bảng 2.8 – Các lệnh số học

Lệnh	**Hoạt động**	**Chế độ địa chỉ**				**Chu kỳ thực thi**
		Tức thời	Trực tiếp	Gián tiếp	Thanh ghi	
ADD A,(byte)	A = A + (byte)	x	x	x	x	1
ADDC A,(byte)	A = A + (byte) + C	x	x	x	x	1
SUBB A,(byte)	A = A - (byte) - C	x	x	x	x	1
INC A	A = A + 1	Chỉ dùng cho thanh ghi tích luỹ ACC				1
INC (byte)	(byte) = (byte) + 1		x	x	x	1
INC DPTR	DPTR = DPTR + 1	Chỉ dùng cho thanh ghi con trỏ lệnh DPTR				2
DEC A	A = A - 1	Chỉ dùng cho thanh ghi tích luỹ ACC				1
DEC (byte)	(byte) = (byte) - 1		x	x	x	1
MUL AB	B_A = B x A	Chỉ dùng cho thanh ghi tích luỹ ACC và thanh ghi B				4
DIV AB	A = A div B B = A mod B	Chỉ dùng cho thanh ghi tích luỹ ACC và thanh ghi B				4
DA A	Hiệu chỉnh trên số BCD	Chỉ dùng cho thanh ghi tích luỹ ACC				1

❖ **Lệnh ADD:**

Thực hiện cộng giữa thanh ghi tích luỹ A và một toán hạng khác. Lệnh ADD ảnh hưởng đến các cờ Carry (C), Overflow (OV) và Auxiliary (AC).

Sưu tầm bởi: www.daihoc.com.vn

Lệnh ADD có 4 chế độ địa chỉ khác nhau:

- `ADD A, #30h` ; định địa chỉ tức thời (A = A + 30h)
- `ADD A, 30h` ; định địa chỉ trực tiếp (A = A + [30h] trong đó [30h] là giá trị của RAM nội có địa chỉ 30h)
- `ADD A, @R0` ; định địa chỉ gián tiếp (A = A + [R0] trong đó [30h] là giá trị của RAM nội có địa chỉ chứa trong thanh ghi R0)

```
MOV R0,#30h    ; R0 = 30h
ADD A,@R0      ; A = A + [R0] = A + [30h] (cộng nội dung của thanh ghi ACC với RAM nội có địa chỉ 30h)
```

- `ADD A,R0` ; định địa chỉ thanh ghi (A = A + R0)

❖ **Lệnh ADDC, SUBB:**

Thực hiện cộng hay trừ nội dung của thanh ghi A với một toán hạng khác trong đó có dùng thêm cờ Carry. Lệnh ADDC và SUBB ảnh hưởng đến các cờ C, OV và AC.

❖ **Lệnh MUL:**

Nhân nội dung của thanh ghi A với thanh ghi B. Lệnh MUL ảnh hưởng đến cờ OV và xoá cờ C (C = 0).

Ví dụ:

```
MOV  A,#50     ; 50 x 25 = 1250 → 04E2h
MOV  B,#25     ; byte cao = 04h, byte thấp = E2h
MUL  AB        ; B = 04h, A = E2h
```

❖ **Lệnh DIV:**

Chia nội dung của thanh ghi A cho thanh ghi B. Lệnh DIV ảnh hưởng đến cờ OV và xoá cờ C (C = 0).

Ví dụ:

```
MOV  A,#250    ; 250 / 40 = 6 dư 10
MOV  B,#40     ;
DIV  AB        ; B = 0Ah (10), A = 06h
```

❖ **Lệnh DA A:**

Hiệu chỉnh nội dung thanh ghi A sau khi thực hiện các phép toán liên quan đến số BCD. Quá trình thực hiện lệnh DA A mô tả như sau:

Sưu tầm bởi: www.daihoc.com.vn

- Nếu A[3-0] > 9 hay AC = 1 thì A[3-0] = A[3-0] + 6
- Nếu A[7-4] > 9 hay C = 1 thì A[7-4] = A[7-4] + 6

Lệnh DA A cũng ảnh hưởng đến cờ C.

Sưu tầm bởi: www.daihoc.com.vn

BÀI TẬP CHƯƠNG 2

1. Xác định giá trị của các biểu thức sau:

a. (10 SHL 2) OR (1000 1000b)

b. (5*2 – 10 SHR 1) AND (11h)

c. HIGH(10000)

d. LOW(-30000)

2. Viết đoạn chương trình đọc nội dung của ô nhớ 30h. Nếu giá trị đọc lớn hơn hay bằng 10 thì xuất 10 ra P0, ngược lại thì xuất giá trị vừa đọc ra P0.

3. Viết đoạn chương trình xuất các giá trị trong ô nhớ 30h – 3Fh ra P1 (giữa các lần xuất có thời gian trì hoãn).

4. Viết đoạn chương trình theo yêu cầu sau:

   - Đọc dữ liệu từ P1 (10 lần) và lưu giá trị đọc mỗi lần vào ô nhớ 30h – 39h (mỗi lần đọc có trì hoãn một khoảng thời gian).
   - Tìm giá trị lớn nhất trong các ô nhớ 30h – 39h, lưu vào ô nhớ 3Ah và xuất giá trị này ra P2.
   - Kiểm tra nội dung ô nhớ 3Ah, nếu = 0 thì quay lại đầu chương trình, ngược lại thì xuất giá trị này ra P3.

5. Viết đoạn chương trình theo yêu cầu:

   - B1: Kiểm tra bit P3.0:

P3.0	**Thực hiện**
= 0	Đến bước 2
= 1	Đến bước 3

   - B2: Đọc dữ liệu từ P2, đảo tất cả các bit và xuất ra P0. Sau đó quay lại bước 1.
   - B3: xuất nội dung tại ô nhớ 30h ra P1 và quay lại bước 1

Sưu tầm bởi: www.daihoc.com.vn

Chương 3: CÁC HOẠT ĐỘNG CỦA VI ĐIỀU KHIỂN MCS-51

Chương này giới thiệu về các hoạt động đặc trưng của họ vi điều khiển MCS-51: định thời, cổng nối tiếp, ngắt và các cách thức để điều khiển các hoạt động này.

1. Hoạt động định thời (Timer / Counter)

1.1. Giới thiệu

AT89C51 có 2 bộ định thời 16 bit có thể hoạt động ở các chế độ khác nhau và có khả năng định thời hay đếm sự kiện (Timer 0 và Timer 1). Khi hoạt động định thời (timer), bộ Timer / Counter sẽ nhận xung đếm từ dao động nội còn khi đếm sự kiện (counter), bộ Timer / Counter nhận xung đếm từ bên ngoài. Bộ Timer / Counter bên trong AT89C51 là các bộ đếm lên 8 bit hay 16 bit tuỳ theo chế độ hoạt động. Mỗi bộ Timer / Counter có 4 chế độ hoạt động khác nhau và được dùng để:

- Đếm sự kiện tại các chân T0 (chân 14) hay T1 (chân 15).
- Chờ một khoảng thời gian.
- Tạo tốc độ cho port nối tiếp.

Quá trình điều khiển hoạt động của Timer / Counter được thực hiện thông qua các thanh ghi sau:

Bảng 3.1 – Các thanh ghi điều khiển hoạt động Timer / Counter

Thanh ghi	Địa chỉ byte	Địa chỉ bit
TCON	88h	88h – 8Fh
TMOD	89h	Không
TL0	90h	Không
TL1	91h	Không
TH0	92h	Không
TH1	93h	Không

Ngoài ra, trong họ 8x52 còn có thêm bộ định thời thứ 3 (Timer 2).

1.2. Hoạt động Timer / Counter

Hoạt động cơ bản của Timer / Counter gồm có các thanh ghi timer THx và TLx (x = 0, 1) mắc liên tầng tạo thành dạng thanh ghi 16 bit. Khi set bit TRx trong thanh ghi TCON (xem thêm phần 1.3), timer tương ứng sẽ hoạt động và giá trị trong thanh ghi TLx tăng lên 1 sau mỗi xung đếm. Khi TLx tràn (thay đổi từ 255 → 0), giá trị của THx tăng lên 1. Khi THx tràn, cờ tràn tương ứng TFx (trong thanh ghi TCON) sẽ được đưa lên mức 1.

Sưu tầm bởi: www.daihoc.com.vn

Tuỳ theo nội dung của bit C/$\overline{T}$ (xem thêm thanh ghi TMOD, phần 1.3), xung đếm có thể lấy từ dao động nội (C/$\overline{T}$ = 0) hay từ các chân Tx bên ngoài (C/$\overline{T}$ = 1). Lưu ý rằng phải xoá bit TRx khi thay đổi chế độ hoạt động của Timer.

Khi xung đếm lấy từ dao động nội, tốc độ đếm = $f_{OSC}/12$ hay $f_{OSC}/2$ trong chế độ X2(nghĩa là nếu f_{OSC} = 12 MHz thì tốc độ xung đếm là 1 MHz hay cứ 1 µs thì có 1 xung đếm trong chế dộ chuẩn) hay tốc độ đếm = $f_{PER}/6$ (f_{PER}: tần số xung ngoại vi – peripheral clock).

Khi lấy xung đếm từ bên ngoài (các chân Tx),bộ đếm sẽ tăng lên 1 khi ngõ vào Tx ở mức 1 trong 1 chu kỳ và xuống mức 0 trong chu kỳ kế tiếp. Do đó, tần số xung tối đa tại các chân Tx là $f_{OSC}/24$ trong chế độ thường hay $f_{OSC}/12$ trong chế độ X2 (=$f_{PER}/12$).

1.3. Các thanh ghi điều khiển hoạt động

1.3.1. Thanh ghi điều khiển timer (TCON – Timer/Counter Control Register)

TCON chứa các bit trạng thái và các bit điều khiển cho Timer 1, Timer 0.

Bảng 3.2 – Nội dung thanh ghi TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	Ký hiệu	Địa chỉ	Mô tả
TCON.7	TF1	8Fh	Cờ báo tràn timer 1 (Timer 1 overflow Flag). Được xoá bởi phần cứng khi chuyển đến chương trình con xử lý ngắt hay xoá bằng phần mềm. Đặt bằng phần cứng khi Timer 1 tràn
TCON.6	TR1	8Eh	Điều khiển Timer 1 chạy (Timer 1 Run Control Bit). Cho phép Timer 1 hoạt động (= 1) hay ngừng (= 0).
TCON.5	TF0	8Dh	Timer 0 overflow Flag
TCON.4	TR0	8Ch	Timer 0 Run Control Bit
TCON.3	IE1	8Bh	Dùng cho ngắt ngoài 0 và 1 (sẽ xét trong phần 3 – xử lý ngắt)
TCON.2	IT1	8Ah	
TCON.1	IE0	89h	
TCON.0	IT0	88h	

Giá trị khi reset: TCON = 00h

Sưu tầm bởi: www.daihoc.com.vn

1.3.2. Thanh ghi chế độ timer (TMOD – Timer/Counter Mode)

Thanh ghi TMOD chứa hai nhóm 4 bit dùng để đặt chế độ làm việc cho Timer 0, và Timer 1. Lưu ý rằng khi lập trình cho AT89C51, thông thường thanh ghi TMOD chỉ được gán một lần ở đầu chương trình.

Bảng 3.3 – Nội dung thanh ghi TMOD

GATE1	C/$\overline{T}$1	M11	M01	GATE0	C/$\overline{T}$0	M10	M00

<table>
<tr><th>Bit</th><th>Tên</th><th>Timer</th><th>Mô tả</th><th>Timer</th></tr>
<tr><td>7</td><td>GATE1</td><td>1</td><td>Timer 1 Gating Control Bit
GATE = 0: timer hoạt động bình thường
GATE = 1: timer chỉ hoạt động khi chân $\overline{INT1}$ = 1</td><td rowspan="5">Dùng cho Timer 1</td></tr>
<tr><td>6</td><td>C/T1</td><td>1</td><td>Timer 1 Timer/Counter Select Bit
= 1: đếm bằng xung ngoài tại chân T1 (chân 15)
= 0: đếm bằng xung dao động bên trong</td></tr>
<tr><td>5</td><td>M11</td><td>1</td><td rowspan="2">Timer 1 Mode Select Bit
<table>
<tr><th>M11</th><th>M01</th><th>Chế độ</th></tr>
<tr><td>0</td><td>0</td><td>13 bit</td></tr>
<tr><td>0</td><td>1</td><td>8 bit tự động nạp lại</td></tr>
<tr><td>1</td><td>0</td><td>16 bit</td></tr>
<tr><td>1</td><td>1</td><td>Không dùng Timer 1</td></tr>
</table></td></tr>
<tr><td>4</td><td>M01</td><td>1</td></tr>
<tr><td>3</td><td>GATE0</td><td>0</td><td>Timer 0 Gating Control Bit</td><td rowspan="5">Dùng cho Timer 0</td></tr>
<tr><td>2</td><td>C/T0</td><td>0</td><td>Timer 0 Timer/Counter Select Bit</td></tr>
<tr><td>1</td><td>M10</td><td>0</td><td rowspan="2">Timer 0 Mode Select Bit
Các chế độ giống như timer 1 trong đó chế độ 3 dùng TH0 và TL0 làm 2 giá trị đếm của timer 0 và timer 1 (xem thêm phần 1.4)</td></tr>
<tr><td>0</td><td>M00</td><td>0</td></tr>
</table>

Giá trị khi reset: TMOD = 00h

Ngoài ra, Timer còn các thanh ghi chứa giá trị đếm: TH0, TL0 (Timer 0) và TH1, TL1 (Timer 1), mỗi thanh ghi có kích thước 8 bit. Giá trị các thanh ghi này khi reset cũng là 00h.

1.4. Các chế độ hoạt động

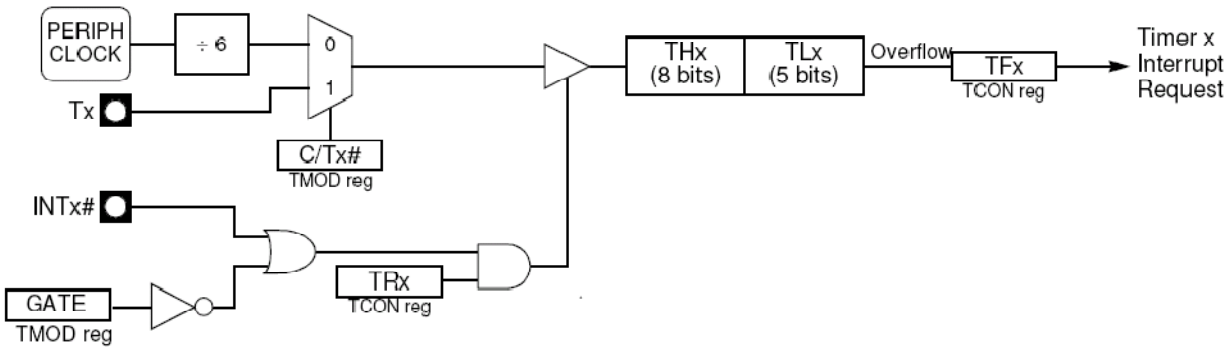
Các chế độ của timer được xác định bằng 4 bit trong thanh ghi TMOD, trong đó 4 bit thấp điều khiển timer 0 và 4 bit cao điều khiển timer 1, mô tả như sau:

Sưu tầm bởi: www.daihoc.com.vn

1.4.1. Chế độ 0

Chế độ 0 là chế độ 13 bit bao gồm 8 bit của thanh ghi THx và 5 bit của thanh ghi TLx còn 3 bit cao của thanh ghi TLx không sử dụng. Mỗi lần có xung đếm, giá trị trong thanh ghi 13 bit tăng lên 1. Khi giá trị này thay đổi từ 1 1111 1111 1111b đến 0 thì bộ đếm tràn làm cho TFx được đặt lên mức 1.

Do chế độ 0 sử dụng 13 bit nên giá trị đếm tối đa là 2^{13} = 8192. Chế độ này được cung cấp nhằm mục đích tạo khả năng tương thích với 8048 và thường không được sử dụng hiện nay.

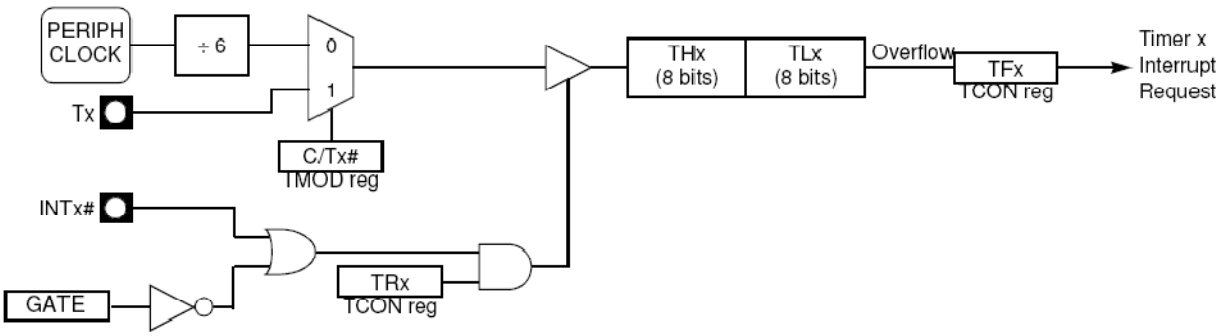


Hình 3.1 – Chế độ 0 của Timer/Counter

1.4.2. Chế độ 1

Chế độ 1 giống như chế độ 0 nhưng sử dụng 16 bit bao gồm 8 bit của THx và 8 bit của TLx nên giá trị đếm tối đa là 2^{16} = 65536. Như vậy, chế độ 0 và chế độ 1 giống nhau nhưng chỉ khác ở số bit đếm nên thông thường chế độ 0 không sử dụng mà chỉ dùng chế độ 1.

Khi bộ đếm tràn (giá trị trong cặp thanh ghi THx_TLx thay đổi từ 1111 1111 1111 1111b đến 0), cờ tràn TFx được set lên mức 1. Lưu ý rằng, khi timer tràn, giá trị của các thanh ghi đếm là 0 (THx = 0 và TLx = 0) nên nếu muốn timer hoạt động tiếp thì phải nạp lại giá trị cho các thanh ghi THx và TLx.



Hình 3.2 – Chế độ 1 của Timer/Counter

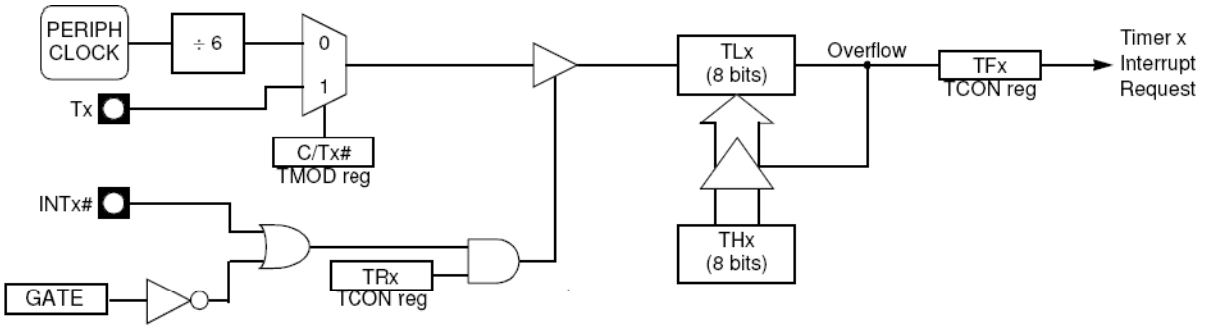
Sưu tầm bởi: www.daihoc.com.vn

1.4.3. Chế độ 2

Chế độ 2 là chế độ 8 bit trong đó sử dụng thanh ghi TLx để chứa giá trị đếm còn thanh ghi THx chứa giá trị nạp lại (do đó chế độ này được gọi là chế độ tự động nạp lại – autoreload).

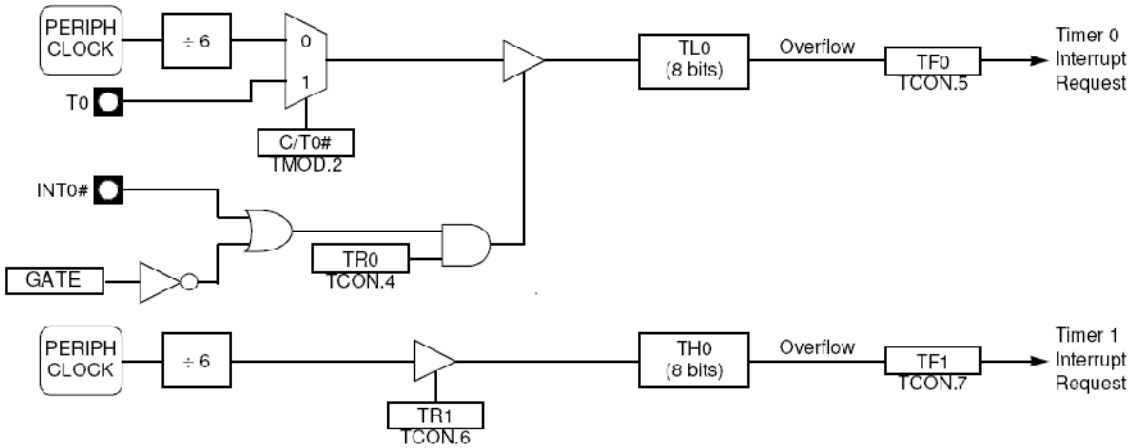
Trong chế độ 2, mỗi khi giá trị trong thanh ghi TLx thay đổi từ 1111 1111b đến 0 thì cờ TFx được set lên mức 1 đồng thời giá trị trong thanh ghi THx được chuyển vào thanh ghi TLx. Như vậy, giá trị đếm trong TLx và THx chỉ được nạp một lần khi khởi động timer (có thể không cần nạp cho TLx nhưng khi đó chu kỳ hoạt động đầu tiên của timer sẽ sai).

Chế độ 2 sử dụng 8 bit đếm trong thanh ghi TLx nên giá trị đếm tối đa là 2^8 = 256.



Hình 3.3 – Chế độ 2 của Timer/Counter

1.4.4. Chế độ 3



Hình 3.4 – Chế độ 3 của Timer/Counter

Chế độ 3 sử dụng các thanh ghi TL0 và TH0 như các bộ định thời độc lập trong đó TL0 điều khiển bằng các thanh ghi của timer 0 và TH0 điều khiển bằng các thanh ghi của tỉmer 1. Khi TL0 chuyển từ giá trị 1111 1111b đến 0 thì TF0 được đặt lên mức 1 còn TH0 chuyển từ 1111 1111b đến 0 thì TF1 được đặt lên mức 1. Lưu ý rằng trong chế độ 3 (chỉ có trong Timer 0), Timer 1 không tác động đến cờ TF1 nên thường được

Sưu tầm bởi: www.daihoc.com.vn

dùng để tạo tốc độ baud cho port nối tiếp (xem thêm phần 2 – cổng nối tiếp) hay dùng cho mục đích khác.

Chế độ này chỉ cho phép tác động đến cờ tràn TF1 thông qua xung đếm của dao động nội mà không đếm bằng dao động ngoài tại chân T1 đồng thời bit GATE1 (TMOD.7) không tác động đến quá trình đếm tại TH0.

1.5. Timer 2

Timer 2 là bộ định thời 16 bit (chỉ có trong họ 8x52). Giá trị đếm của timer 2 chứa trong các thanh ghi TH2 và TL2. Giống như timer 0 và timer1, timer 2 cũng hoạt động như bộ định thời (timer) hay đếm sự kiện (counter). Chế độ định thời đếm bằng dao động nội, chế độ đếm sự kiện đếm bằng xung ngoài tại chân T2 (P1.0) và chọn chế độ bằng bit $C/\overline{T}2$ của thanh ghi T2CON. Các thanh ghi điều khiển timer 2 bao gồm: T2CON, T2MOD, RCAP2H, RCAP2L, TH2 và TL2.

Timer 2 có 3 chế độ hoạt động: capture (giữ), autoreload (tự động nạp lại) và tạo tốc độ baud (chọn chế độ trong thanh ghi T2CON). Các bit chọn chế độ được mô tả như bảng 3.4.

Bảng 3.4 – Chọn chế độ trong Timer 2

RCLK	TCLK	$CP/\overline{RL}2$	TR2	Chế độ
0	0	0	1	Tự động nạp lại 16 bit
0	0	1	1	Giữ 16 bit
X	1	X	1	Tạo tốc độ baud
1	X	X	1	
X	X	X	0	Ngưng

1.5.1. Các thanh ghi điều khiển Timer 2

❖ **Thanh ghi T2CON:**

Bảng 3.5 – Nội dung thanh ghi T2CON

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T}2$	$CP/\overline{RL}2$

Bit	Tên	Mô tả
7	TF2	Timer 2 overflow Flag TF2 không được tác động khi RCLK hay TCLK = 1. TF2 phải được xoá bằng phần mềm và được đặt bằng phần cứng khi Timer tràn
6	EXF2	Timer 2 External Flag Được đặt khi EXEN2 = 1 và xảy ra chế độ nạp lại hay giữ do có cạnh âm tại chân T2EX (P1.1) (chuyển từ 1 xuống 0). Khi EXF2 = 1 và cho phép ngắt tại Timer 2 thì chương trình sẽ chuyển đến chương trình phục vụ ngắt của Timer 2. EXF2 phải được xoá bằng phần mềm

Sưu tầm bởi: www.daihoc.com.vn

5	RCLK	Receive Clock Bit (chỉ dùng cho port nối tiếp ở chế độ 1 và 3) RCLK = 0: dùng timer 1 làm xung clock thu cho port nối tiếp RCLK = 1: dùng timer 2 làm xung clock thu cho port nối tiếp
4	TCLK	Transmit Clock Bit Giống như RCLK nhưng dùng cho xung clock phát
3	EXEN2	Timer 2 External Enable Bit = 0: bỏ qua tác động tại chân T2EX (P1.1) = 1: xảy ra chế độ nạp lại hay giữ do có cạnh âm tại chân T2EX (P1.1) (chuyển từ 1 xuống 0)
2	TR2	Timer 2 Run Control Bit = 0: cấm timer 2 = 1: chạy timer 2
1	$C/\overline{T}2$	Timer / Counter 2 Select Bit = 0: định thời (đếm bằng dao động nội) = 1: đếm sự kiện (đếm bằng xung tại T2 (P1.0))
0	$CP/\overline{RL}2$	Timer 2 Capture / Reload Bit Nếu RCLK = 1 hay TCLK = 1: bỏ qua Nếu RCLK = 0 và TCLK = 0: chọn chế độ giữ (= 1) hay nạp lại (= 0) khi xuất hiện xung âm tại T2EX (P1.1) và EXEN2 = 1

Giá trị khi reset: T2CON = 00h, T2CON cho phép định vị bit

❖ **Thanh ghi T2MOD:**

Bảng 3.6 – Nội dung thanh ghi T2MOD

-	-	-	-	-	-	T2OE	DCEN

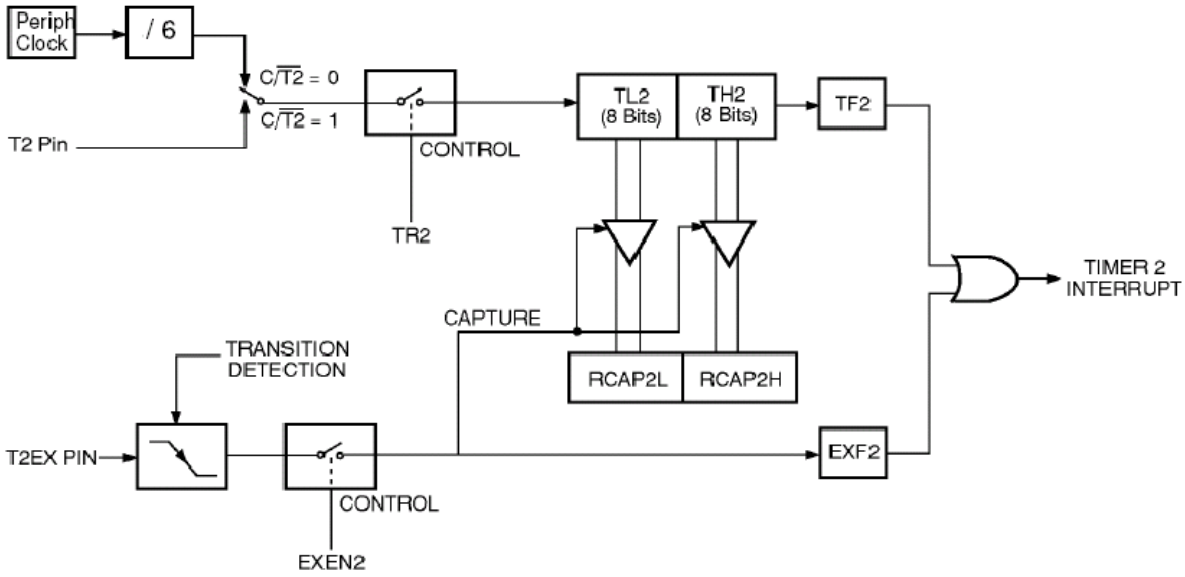
Bit	Tên	Mô tả
7	-	
6	-	
5	-	
4	-	
3	-	
2	-	
1	T2OE	Timer 2 Output Enable Bit = 0: T2 (P1.0) là ngõ vào clock hay I/O port = 1: T2 là ngõ ra clock
0	DCEN	Down Counter Enable Bit = 0: cấm timer 2 là bộ đếm lên / xuống = 1: cho phép timer 2 là bộ đếm lên / xuống

Sưu tầm bởi: www.daihoc.com.vn

Giá trị khi reset: T2MOD = xxxx xx00b, MOD không cho phép định vị bit

Các thanh ghi TH2, TL2, RCAP2H và RCAP2L không cho phép định vị bit và giá trị khi reset là 00h. Các chế độ hoạt động của Timer 2 mô tả trong phần sau.

1.5.2. Chế độ capture



Hình 3.5 – Chế độ giữ của Timer 2

Chế độ giữ của Timer 2 có 2 trường hợp xảy ra:

- Nếu EXEN2 = 0: Timer 2 hoạt động giống như Timer 0 và 1, nghĩa là khi giá trị đếm tràn (TH2_TL2 thay đổi từ FFFFh đến 0) thì cờ tràn TF2 được đặt lên mức 1 và tạo ngắt tại Timer 2 (nếu cho phép ngắt).
- Nếu EXEN2 = 1: vẫn hoạt động như trên nhưng thêm một tính chất nữa là: khi xuất hiện cạnh âm tại chân T2EX (P1.1), giá trị hiện tại của TH2 và TL2 được chuyển vào cặp thanh ghi RCAP2H, RCAP2L (quá trình giữ (capture) xảy ra); đồng thời, bit EXF2 = 1 (sẽ tạo ngắt nếu cho phép ngắt tại Timer 2).

1.5.3. Chế độ tự động nạp lại

Chế độ tự động nạp lại cũng có 2 trường hợp giống như chế độ giữ:

- Nếu EXEN2 = 0: khi Timer tràn, cờ tràn TF2 được đặt lên 1 và nạp lại giá trị cho TH2, TL2 (từ cặp thanh ghi RCAP2H, RCAP2L) đồng thời tạo ngắt tại timer 2 nếu cho phép ngắt.
- Nếu EXEN2 = 1: hoạt động giống như trên nhưng khi có xung âm tại chân T2EX thì cũng nạp lại giá trị cho TH2, TL2 và đặt cờ EXF2 lên 1.

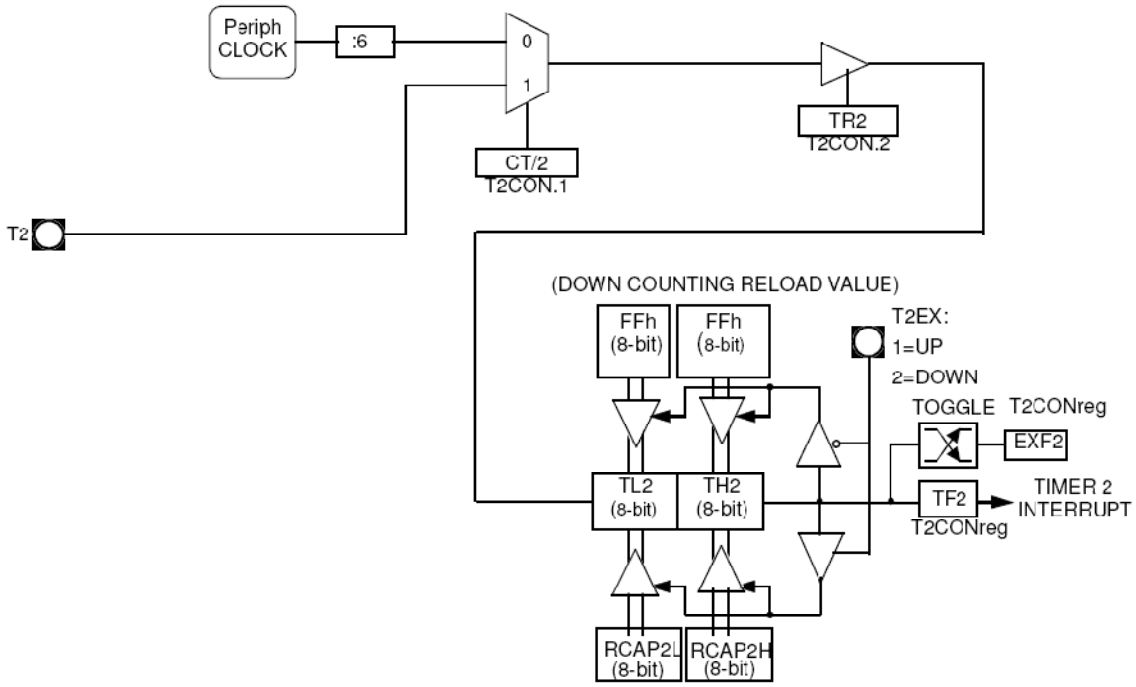
Sưu tầm bởi: www.daihoc.com.vn

Chế độ tự động nạp lại cũng cho phép thực hiện đếm lên hay xuống (điều khiển bằng bit DCEN trong thanh ghi T2MOD). Khi DCEN được đặt lên 1 và chân T2EX ở mức cao thì timer 2 sẽ đếm lên; còn nếu T2EX ở mức thấp thì timer 2 đếm xuống.

Khi đếm lên, tỉmer tràn tại giá trị đếm 0FFFFh. Khi tràn, cờ TF2 được đặt lên mức 1 và giá trị trong cặp thanh ghi RCAP2H, RCAP2L chuyển vào căp thanh ghi TH2, TL2.

Khi đếm xuống, timer tràn khi giá trị trong cặp thanh ghi TH2, TL2 bằng giá trị trong cặp thanh ghi RCAP2H, RCAP2L. Khi tràn, cờ TF2 được đặt lên 1 và giá trị 0FFFFh được nạp vào cặp thanh ghi TH2, TL2.

Trong chế độ này, khi timer tràn, giá trị trong cờ EXF2 sẽ chuyển mức và không tạo ngắt (có thể dùng thêm EXF2 để tạo giá trị đếm 17 bit).



Hình 3.6 – Chế độ tự động nạp lại

1.5.4. Chế độ tạo xung clock

Trong chế độ này, timer tạo ra một xung clock có chu kỳ bổn phận (duty cycle) 50%. Khi timer tràn, nội dung của thanh ghi RCAP2H, RCAP2L được nạp vào cặp thanh ghi TH2, TL2 và timer tiếp tục đếm. Tần số xung clock tại chân T2 được xác định theo công thức sau:

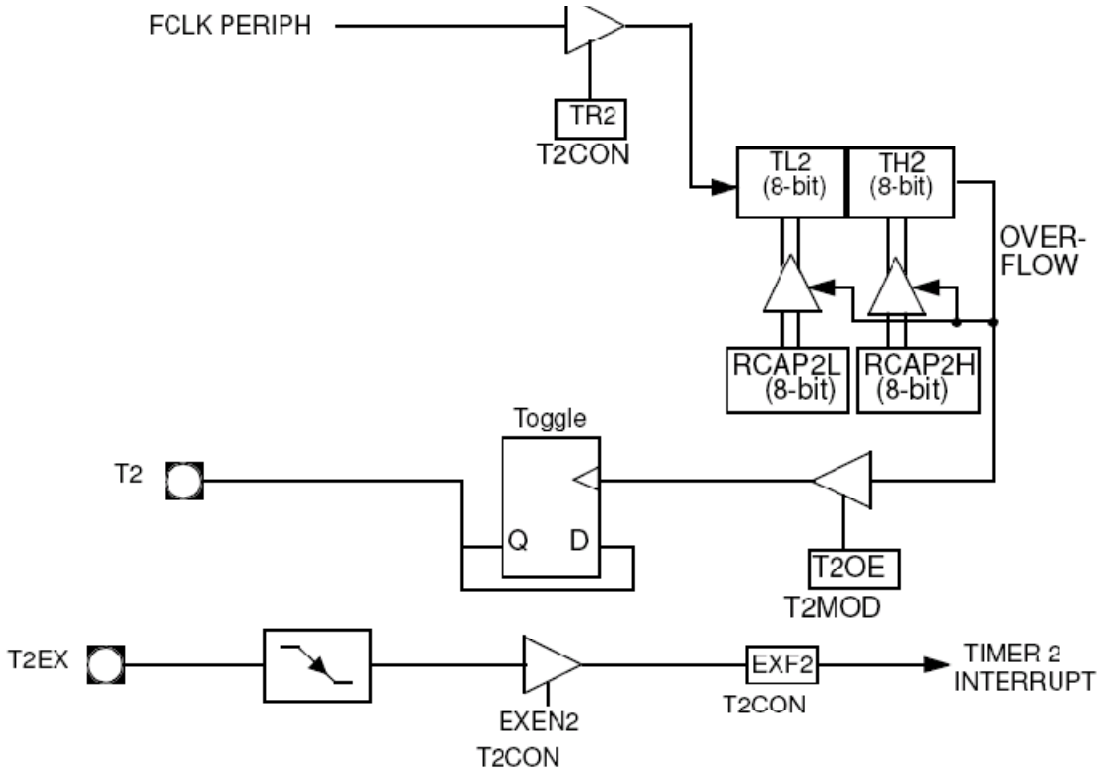
$$f = \frac{f_{OSC} x 2^{X2}}{2\left(65536 - \frac{RCAP2H}{RCAP2L}\right)}$$

Sưu tầm bởi: www.daihoc.com.vn

X2: bit nằm trong thanh ghi CKCON. Trong chế độ X2: $f_{OSC} = f_{thạch\ anh}$, ngược lại thì $f_{OSC} = f_{thạch\ anh}/2$.

Để timer 2 hoạt động ở chế độ tạo xung clock, cần thực hiện các bước sau:

- Đặt bit T2OE trong thanh ghi T2MOD = 1.
- Xoá bit $C/\overline{T}2$ trong thanh ghi T2CON = 0 (do chế độ này không cho phép đếm bằng dao động ngoài mà chỉ đếm bằng dao động nội).
- Xác định giá trị của cặp thanh ghi RCAP2H và RCAP2L theo tần số xung clock cần tạo.
- Khởi động giá trị cho cặp thanh ghi TH2, TL2 (có thể không cần thiết tuỳ theo ứng dụng).
- Đặt bit TR2 trong thanh ghi T2CON = 1 để cho phép timer chạy.



Hình 3.7 – Chế độ tạo xung clock

1.5.5. Chế độ tạo tốc độ baud

Khi các bit TCLK và RCLK trong thanh ghi T2CON được đặt lên mức 1, timer 2 sẽ dùng để tạo tốc độ baud cho cổng nối tiếp. Chế độ này cùng hoạt động như timer 0 và timer 1 (sẽ khảo sát cụ thể tại phần 2 – cổng nối tiếp).

Sưu tầm bởi: www.daihoc.com.vn

1.6. Các ví dụ

Để điều khiển hoạt động của timer, cần thực hiện:

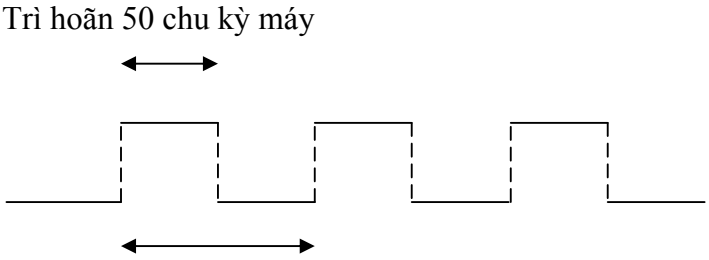
- Nạp giá trị cho thanh ghi TMOD để xác định chế độ hoạt động (thông thường chỉ dùng chế độ 1 – 16 bit và chế độ 2 – 8 bit tự động nạp lại).
- Nạp giá trị đếm trong các thanh ghi THx, TLx (thông thường sử dụng timer 0 và timer 1 nên quá trình đếm là đếm lên).
- Đặt các bit TR0, TR1 = 1 (cho phép timer hoạt động) hay xoá các bit này về 0 (cấm timer).
- Trong quá trình timer chạy, thực hiện kiểm tra các bit TF0, TF1 để xác định timer đã tràn hay chưa.
- Sau khi timer tràn, nếu thực hiện kiểm tra tràn bằng phần mềm (không dùng ngắt) thì phải thực hiện xoá TF0 hay TF1 để có thể tiếp tục hoạt động.

Ví dụ 1: Viết chương trình tạo sóng vuông tần số 10 KHz tại chân P1.0 dùng timer 0 (tần số thạch anh là f_{OSC} = 12MHz).

Giải

Do f_{OSC} = 12MHz nên chu kỳ máy = 1 µs.

f = 10 KHz → T = 1/f = 0.1 ms = 100 µs → một chu kỳ sóng vuông chiếm khoảng thời gian 100 chu kỳ máy → thời gian trì hoãn cần thiết là 50 chu kỳ máy.



Do giá trị đếm là 50 (ứng với 50 chu kỳ máy) nên chỉ cần dùng chế độ 8 bit (có thể đếm từ 1 đến 256) cho timer 0 (chế độ 2).

- Nội dung thanh ghi TMOD:

GATE1	C/T1	M11	M10	GATE0	C/T0	M01	M00
0	0	0	0	0	0	1	0
Timer 1 không dùng				Không dùng INT0	Đếm bằng dao động nội	Chế độ 8 bit	

TMOD = 0000 0010b (02h)

Sưu tầm bởi: www.daihoc.com.vn

- Giá trị đếm là 50 và do timer 0 đếm lên nên giá trị cần nạp cho TH0 là -50 (có thể không cần nạp cho TL0 nhưng lúc đó chu kỳ đầu tiên của xung sẽ sai).

Chương trình thực hiện như sau:

```
      MOV TMOD,#02h
      MOV TH0,#(-50)
      MOV TL0,#(-50)
      SETB TR0       ; Cho phép timer 0 chạy
Lap:
      JNB TF0,Lap    ; Nếu Timer chưa tràn thì chờ
      CLR TF0
      CPL P1.0       ; Đảo bit P1.0 để tạo xung vuông
      SJMP Lap
      END
```

Ví dụ 2: Viết chương trình tạo xung vuông tần số f = 1 KHz tại P1.1 dùng timer 1(tần số thạch anh là f_{OSC} = 12MHz).

Giải

Do f_{OSC} = 12MHz nên chu kỳ máy = 1 µs.

f = 1 KHz → T = 1/f = 1 ms = 1000 µs → một chu kỳ sóng vuông chiếm khoảng thời gian 1000 chu kỳ máy → thời gian trì hoãn cần thiết là 500 chu kỳ máy.

Giá trị đếm là 500 vượt quá phạm vi của chế độ 8 bit nên phải sử dụng timer 1 ở chế độ 16 bit (chế độ 1). Đối với chế độ 16 bit, do không có giá trị nạp lại nên mỗi khi timer tràn, cần phải nạp lại giá trị cho thanh ghi TH1 và TL1.

- Nội dung thanh ghi TMOD:

GATE1	C/T1	M11	M10	GATE0	C/T0	M01	M00
0	0	0	1	0	0	0	0
Không dùng INT1	Đếm bằng dao động nội	Chế độ 16 bit		Timer 0 không dùng			

TMOD = 0001 0000b (10h)

- Giá trị đếm là 500 nên giá trị cần nạp cho cặp thanh ghi TH0_TL0 là -500 (dùng các lệnh giả HIGH và LOW).

Chương trình thực hiện như sau:

```
      MOV TMOD,#10h
Batdau:
      MOV TH1,#HIGH(-500)
      MOV TL1,#LOW(-500)
      SETB TR1       ; Cho phép timer 1 chạy
```


Sưu tầm bởi: www.daihoc.com.vn

```
Lap:
     JNB TF1,Lap    ; Nếu Timer chưa tràn thì chờ
     CLR TF1
     CPL P1.1       ; Đảo bit P1.1 để tạo xung vuông
     CLR TR1
     SJMP Batdau    ; Quay lại nạp giá trị cho TH0_TL0
     END
```

Ví dụ 3: Viết chương trình tạo xung vuông tần số f = 10KHz tại P1.0 dùng timer 0 và xung vuông tần số f = 1 KHz tại P1.1 dùng timer 1.

Giải

Phân tích cho các thanh ghi giống như phần ví dụ 1 và 2 nhưng lưu ý rằng quá trình kiểm tra timer tràn sẽ khác: thực hiện kiểm tra timer 0, nếu chưa tràn thì kiểm tra timer 1 và kiểm tra tương tự cho timer 1.

Chương trình thực hiện như sau:

```
     MOV TMOD,#12h
     MOV TH1,#HIGH(-500)
     MOV TL1,#LOW(-500)
     MOV TH0,#(-50)
     MOV TL0,#(-50)
     SETB TR0
     SETB TR1
KtrT0:
     JNB TF0,KtrT1
     CLR TF0
     CPL P1.0
KtrT1:
     JNB  TF1,KtrT0
     CLR TF1
     CPL P1.1
     MOV TH1,#HIGH(-500)
     MOV TL1,#LOW(-500)
     SJMP KtrT0
     END
```

Lưu ý rằng, xung vuông tạo bằng cách như trên có thể không chính xác khi 2 timer tràn cùng lúc.

Ví dụ 4: Viết chương trình tạo xung vuông tần số f = 1 Hz tại P1.2 dùng timer1.

Giải

f = 1 Hz → T = 1/f = 1 s = 1 000 000 µs → một chu kỳ sóng vuông chiếm khoảng thời gian 500 000 chu kỳ máy → thời gian trì hoãn cần thiết là 500 000 chu kỳ máy.

Sưu tầm bởi: www.daihoc.com.vn

Giá trị đếm là 500 000, vượt quá khả năng của timer (tối đa chỉ đếm được 65536 chu kỳ) nên phải thực hiện tạo vòng lặp đếm nhiều lần cho đến khi đạt đến giá trị 500 000 (có thể đếm mỗi lần 50 000 và thực hiện vòng lặp 10 lần).

Chương trình thực hiện như sau:

```
     MOV TMOD,#10h
Batdau:
     MOV R7,#10     ; Lặp 10 lần
Lap:
     MOV TH1,#HIGH(-50000)
     MOV TL1,#LOW(-50000)
     SETB TR1
KtrT1:
     JNB  TF1,KtrT1
     CLR TF1
     CLR  TR0
     DJNZ R7,Lap    ; Nếu R7 ≠ 0 thì lặp lại

     CPL P1.2       ; Đảo bit để tạo xung

     SJMP Batdau
     END
```

Ví dụ 5: Viết chương trình con tạo thời gian trì hoãn 1s dùng timer 0.

Giải

Do chương trình yêu cầu tạo thời gian trì hoãn nên số chu kỳ đếm là 1 000 000.

Chương trình như sau:

```
     MOV TMOD,#01h
     ;--- CHƯƠNG TRÌNH CHÍNH
     ;---

Delay1s:
     MOV R7,#20     ; Lặp 20 lần
Lap:
     MOV TH0,#HIGH(-50000)   ; Mỗi lần trì hoãn 50 000 µs
     MOV TL0,#LOW(-50000)
     SETB TR0
Lap1:
     JNB TF0,Lap1
     CLR TF0
     CLR TR0
     DJNZ R7,Lap    ; Lặp đủ 20 lần thì thoát
     RET
```


Sưu tầm bởi: www.daihoc.com.vn

Lưu ý rằng khi viết chương trình trì hoãn như trên thì chương trình của AT89C51 xem như dừng lại, không làm gì cả (có thể giải quyết bằng cách sử dụng ngắt – xem thêm phần 3).

2. Cổng nối tiếp (Serial port)

Cổng nối tiếp trong 89C51 có khả năng hoạt động ở chế độ đồng bộ và bất đồng bộ dùng 2 chân TxD (P3.1) và RxD (P3.0). Chức năng của port nối tiếp là thực hiện chuyển đổi song song sang nối tiếp đối với dữ liệu xuất, và chuyển đổi nối tiếp sang song song đối với dữ liệu nhập.

Khi hoạt động ở chế độ truyền / nhận bất đồng bộ (UART – Universal Asynchronous Receiver / Transmitter), cổng nối tiếp có 3 chế độ song công (1, 2 và 3). Quá trình đọc / ghi cổng nối tiếp dùng thanh ghi SBUF (Serial Buffer), thực chất là 2 thanh ghi khác nhau: một thanh ghi truyền và một thanh ghi nhận.

Cổng nối tiếp có tất cả 4 chế độ khác nhau:

Chế độ 0: dữ liệu truyền / nhận thông qua chân RxD và xung clock dịch bit thông qua TxD với tốc độ baud bằng $f_{\text{thạch anh}}/12$.

Chế độ 1: truyền / nhận 10 bit: 1 bit start (luôn = 1), 8 bit dữ liệu và 1 bit stop (luôn = 0), tốc độ baud có thê thay đổi được và khi nhận, bit stop đưa vào RB8 của thanh ghi SCON.

Chế độ 2: truyền / nhận 11 bit: 1 bit start, 8 bit dữ liệu, bit thứ 9 và 1 bit stop. Khi truyền, bit 9 là bit TB8 và khi nhận, bit 9 là bit RB8 trong thanh ghi SCON. Tốc độ baud cố định là 1/32 hay 1/64 tần số thạch anh.

Chế độ 3: giống chế độ 2 nhưng tốc độ baud có thể thay đổi được.

Trong 4 chế độ trên, thường sử dụng chế độ 1 hay 3 để truyền dữ liệu. Trong trường hợp truyền dữ liệu giữa các vi điều khiển AT89C51 với nhau, có thể dùng chế độ 2. Ngoài ra, cổng nối tiếp còn có các chế độ nâng cao: kiểm tra lỗi khung và nhận dạng địa chỉ tự động.

Sưu tầm bởi: www.daihoc.com.vn

2.1. Các thanh ghi điều khiển hoạt động

2.1.1. Thanh ghi SCON (Serial port controller)

Bảng 3.7 – Nội dung thanh ghi SCON

FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit	Ký hiệu	Địa chỉ	Mô tả
SCON.7	FE	9Fh	Framing Error – kiểm tra lỗi khung Được đặt lên 1 khi phát hiện lỗi tại bit stop và phải xoá bằng phần mềm. Bit FE chỉ truy xuất được khi bit SMOD0 = 1 (trong thanh ghi PCON).
	SM0		Serial port Mode bit 0 - Xác định chế độ cho cổng nối tiếp
SCON.6	SM1	9Eh	Serial port Mode bit 1
SCON.5	SM2	9Dh	Serial port Mode bit 2 – Chế độ đa xử lý = 0: bình thường = 1: cho phép truyền thông đa xử lý trong chế độ 2 và 3
SCON.4	REN	9Ch	Reception Enable bit – Cho phép thu = 0: cấm thu = 1: cho phép thu tại cổng nối tiếp
SCON.3	TB8	9Bh	Transmitter Bit – Bit truyền thứ 9 trong chế độ 2 và 3
SCON.2	RB8	9Ah	Receiver Bit – Bit nhận thứ 9 trong chế độ 2 và 3. Trong chế độ 1, nếu SM2 = 0 thì RB8 = stop bit.
SCON.1	TI	99h	Transmit Interrupt flag – Cờ ngắt phát Được đặt bằng 1 khi kết thúc quá trình truyền và xoá bằng phần mềm.
SCON.0	RI	99h	Receive Interrupt flag – Cờ ngắt thu Được đặt bằng 1 khi nhận xong dữ liệu và xoá bằng phần mềm.

SM0	SM1	Mô tả	Tốc độ baud
0	0	Thanh ghi dịch	$f_{OSC}/12$
0	1	UART 8 bit	Thay đổi
1	0	UART 9 bit	$f_{OSC}/32$ hay $f_{OSC}/64$
1	1	UART 9 bit	Thay đổi

Giá trị khi reset: 00h, cho phép định địa chỉ bit

Sưu tầm bởi: www.daihoc.com.vn

2.1.2. Thanh ghi BDRCON (Baud Rate Control Register)

Bảng 3.8 – Nội dung thanh ghi BDRCON

-	-	-	BRR	TBCK	RBCK	SPD	SRC

Bit	Ký hiệu	Mô tả
7	-	
6	-	
5	-	
4	BRR	Baud Rate Run control bit – Cho phép hoạt động = 0: cấm bộ tạo tốc độ baud nội (internal baud rate generator) hoạt động = 1: cho phép
3	TBCK	Transmission Baud rate generator selection bit for UART – Chọn bộ tạo tốc độ baud truyền là bộ tạo tốc độ nội (= 1) hay bằng timer (= 0)
2	RBCK	Reception Baud rate generator selection bit for UART – Chọn bộ tạo tốc độ baud nhận là bộ tạo tốc độ nội (= 1) hay bằng timer (= 0)
1	SPD	Baud Rate Speed control bit for UART – Chọn tốc độ baud là nhanh (= 1) hay chậm (= 0)
0	SRC	Baud Rate Source select bit in Mode 0 for UART – Chọn tốc độ baud trong chế độ 0 từ dao động thạch anh (= 0) hay từ bộ tạo tốc độ baud nội (= 1)

Giá trị khi reset: 00h, không cho phép định địa chỉ bit

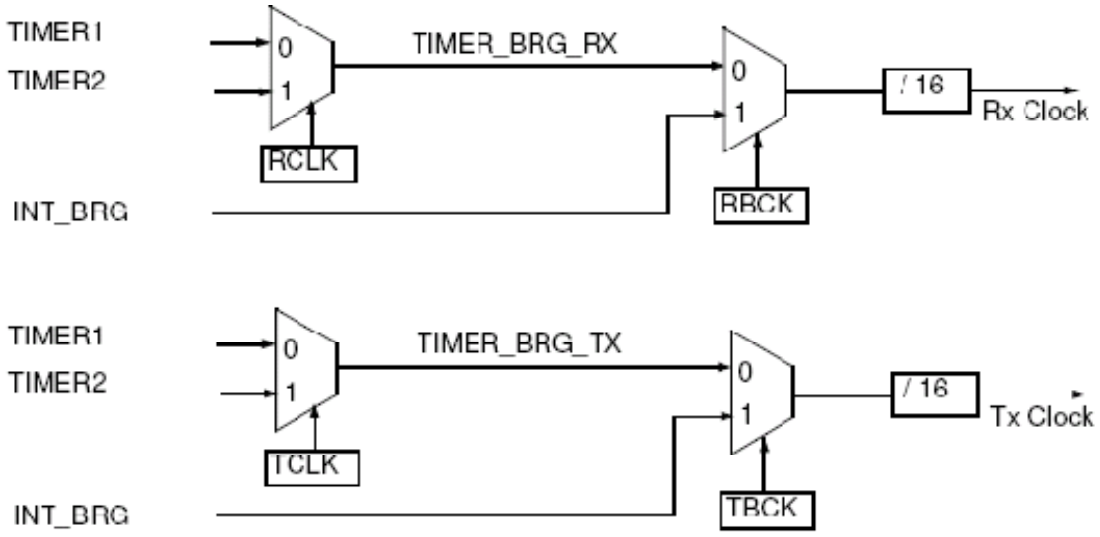
Ngoài ra còn có các thanh ghi SBUF (Serial Buffer), BRL (Baud Rate Reload), SADEN (Slave Address Mark), SADDR (Slave Address).

Lưu ý rằng các thanh ghi BDRCON, BRL, SADEN và SADDR chỉ có trong các phiên bản mới của MCS-51.

2.2. Tạo tốc độ baud

- Chế độ 0: tốc độ baud cố định = 1/12 tần số thạch anh.
- Chế độ 2: tốc độ baud = 1/32 tần số thạch anh khi SMOD = 1 hay 1/64 khi SMOD = 0 (SMOD: nằm trong thanh ghi PCON).
- Chế độ 1 và 3: tốc độ baud xác định bằng tốc độ tràn của timer 1. Trong họ 89x52, có thể dùng timer 2 để tạo tốc độ baud còn trong các phiên bản mới, có thể dùng bộ tạo tốc độ nội (INT_BRG – Internal Baud Rate Generator). Việc xác định nguồn tạo tốc độ baud mô tả như hình 3.8 và bảng 3.9.

Sưu tầm bởi: www.daihoc.com.vn



Hình 3.8 – Lựa chọn tốc độ baud

Bảng 3.9 – Lựa chọn tốc độ baud

TCLK	RCLK	TBCK	RBCK	Clock phát	Clock thu
0	0	0	0	Timer 1	Timer 1
1	0	0	0	Timer 2	Timer 1
0	1	0	0	Timer 1	Timer 2
1	1	0	0	Timer 2	Timer 2
X	0	1	0	INT_BRG	Timer 1
X	1	1	0	INT_BRG	Timer 2
0	X	0	1	Timer 1	INT_BRG
1	X	0	1	Timer 2	INT_BRG
X	X	1	1	INT_BRG	INT_BRG

2.2.1. Tạo tốc độ baud bằng Timer 1

Khi dùng timer 1 để tạo tốc độ baud, thông thường cần thiết lập timer 1 hoạt động ở chế độ 8 bit tự nạp lại và giá trị nạp ban đầu của timer 1 (chứa trong thanh ghi TH1) phụ thuộc vào tốc độ baud cần tạo theo công thức sau:

$$\text{Giá trị nạp} = -\frac{f_{OSC} \times 2^{SMOD}}{12 \times 32 \times baud\_rate}$$

Ví dụ: Giả sử tần số thạch anh là f_{OSC} = 11.0592 MHz, giá trị nạp khi tạo tốc độ baud 4800 bps là:

Sưu tầm bởi: www.daihoc.com.vn

Nếu SMOD = 0: giá trị nạp = $-\frac{11.0592\times10^6\times2^0}{12\times32\times4800}=-6\rightarrow$ TH1 = -6 hay TH1 = FAh

Nếu SMOD = 1: giá trị nạp = $-\frac{11.0592\times10^6\times2^1}{12\times32\times4800}=-12\rightarrow$ TH1 = -12 hay TH1 = F4h

Ví dụ: Giả sử tần số thạch anh là f_{OSC} = 12 MHz, giá trị nạp khi tạo tốc độ baud 4800 bps là:

Nếu SMOD = 0: giá trị nạp = $-\frac{12\times10^6\times2^0}{12\times32\times4800}=-6.51\rightarrow$ chọn giá trị nạp là -6 hay -7. Nếu chọn giá trị nạp = -6 thì tốc độ baud = 5208 bps còn nếu chọn -7 thì tốc độ baud là 4464 bps.

Nếu SMOD = 1: giá trị nạp = $-\frac{11.0592\times10^6\times2^1}{12\times32\times4800}=-13.02\rightarrow$ chọn giá trị nạp là -13 $\rightarrow$ tốc độ baud là 4807 bps. Như vậy, khi dùng tần số thạch anh là 12 MHz thì tốc độ baud sẽ có sai số $\rightarrow$ chỉ dùng khi kết nối nhiều vi điều khiển MCS-51 với nhau còn khi kết nối với các thiết bị khác (như máy tính chẳng hạn) thì nên sử dụng tần số thạch anh 11.0592 MHz.

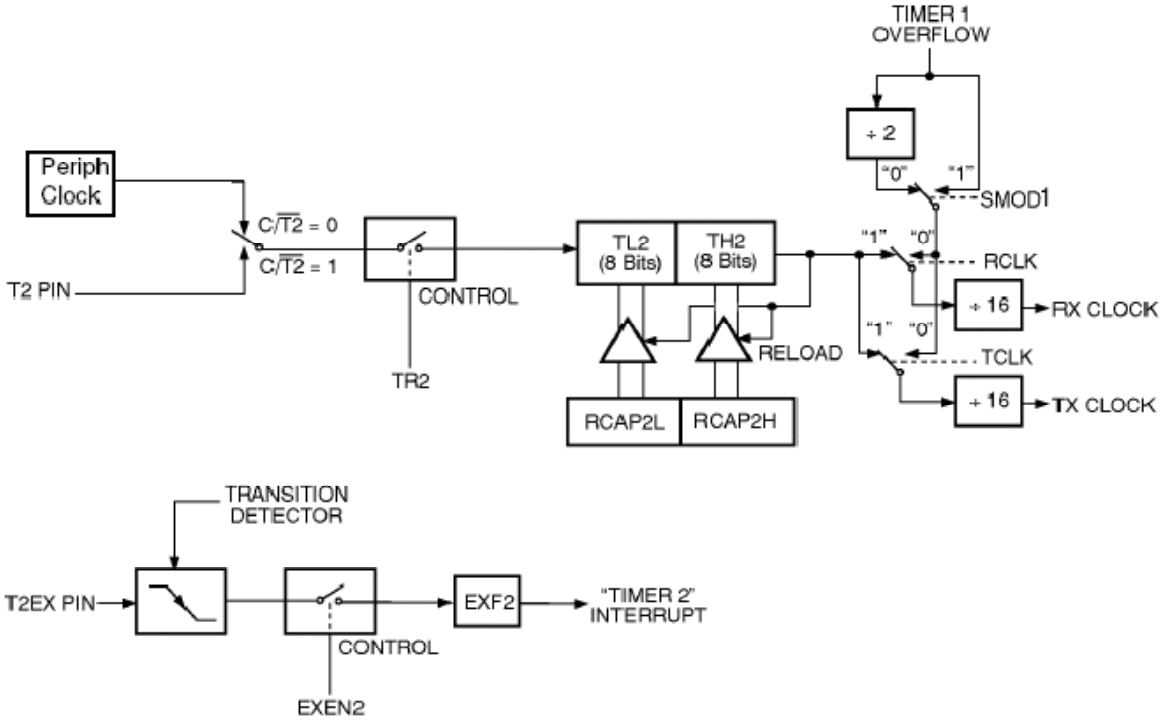
Các giá trị nạp thông dụng cho MCS-51 mô tả như sau:

Bảng 3.10 – Các giá trị nạp thông dụng

Tốc độ [bps]	f_{OSC}[MHz]	SMOD	Giá trị nạp	Tốc độ thực [bps]	Sai số
1200	11.059	0	-12	1200	0
4800	11.059	0	-6	4800	0
9600	11.059	0	-3	9600	0
1200	11.059	1	-24	1200	0
19200	11.059	1	-3	19200	0
1200	12	0	-26	1201.9	2.17%
2400	12	0	-13	2403.8	0.16%
4800	12	0	-6	5208.3	8.5%
9600	12	0	-3	10416.7	8.5%

Sưu tầm bởi: www.daihoc.com.vn

2.2.2. Tạo tốc độ baud bằng Timer 2



Hình 3.9 – Tạo tốc độ baud bằng timer 2

Timer 2 được dùng để tạo tốc độ baud khi đặt các bit TCLK, RCLK lên 1 (trong thanh ghi T2CON). Công thức liên quan giữa tốc độ baud và giá trị nạp như sau (lưu ý rằng giá trị nạp chứa trong cặp thanh ghi RCAP2H_RCAP2L):

$$\text{Giá trị nạp} = -\frac{f_{OSC}}{2 \times 16 \times \text{baud\_rate}}$$

Khi dùng Timer 2 để tạo tốc độ baud, xung clock thu và phát có thể tách riêng bằng cách chỉ dùng TCLK hay RCLK. Lúc đó, xung clock còn lại được xác định theo Timer 1. Ngoài ra, cũng có thể tạo ngắt cho Timer 2 bằng cách đặt bit EXEN2 = 1 và ngắt tạo ra khi xuất hiện cạnh âm tại chân T2EX.

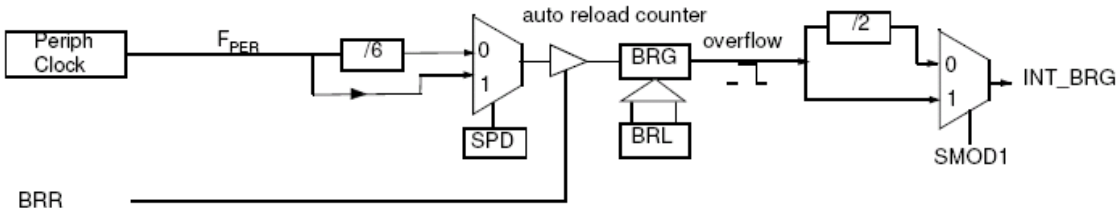
Ví dụ: Giả sử tần số thạch anh là f_{OSC} = 11.0592 MHz, giá trị nạp khi tạo tốc độ baud 4800 bps là:

$$\text{Giá trị nạp} = -\frac{11.0592 \times 10^6}{2 \times 16 \times 4800} = -72 \rightarrow \text{FFB8h}$$

$\rightarrow$ RCAP2H = FFh, RCAP2L = B8h

Sưu tầm bởi: www.daihoc.com.vn

2.2.3. Bộ tạo tốc độ baud nội (INT_BRG – Internal Baud Rate Generator)



Hình 3.10 – Bộ tạo tốc độ baud nội

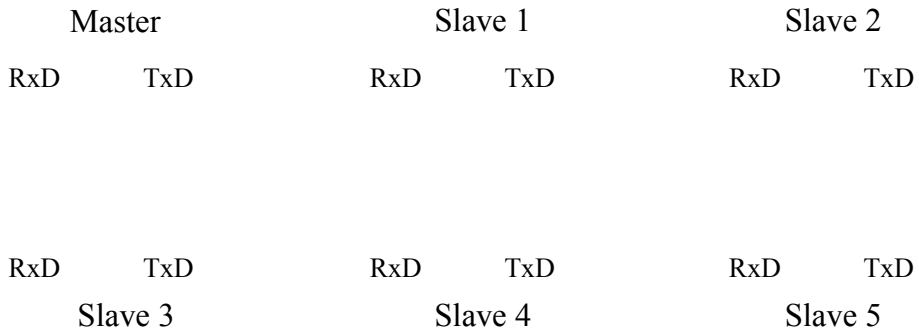
Giá trị nạp trong bộ tạo tốc độ nội chứa trong thanh ghi BRL và được xác định theo công thức sau:

$$\text{Giá trị nạp} = -\frac{f_{OSC} \times 2^{SMOD1}}{2 \times 32 \times 6^{1-SPD} \times baud\_rate}$$

Trong đó SMOD1 nằm trong thanh ghi PCON và SPD nằm trong thanh ghi BDRCON.

2.3. Truyền thông đa xử lý

Chế độ 2 và 3 của MCS-51 cho phép thực hiện kết nối nhiều vi điều khiển ở chế độ master – slave. Mô hình thực hiện của quá trình truyền thông mô tả như hình vẽ sau:



Hình 3.11 – Truyền thông đa xử lý

Quá trình truyền dữ liệu mô tả như sau:

- Khi khởi động, các vi điều khiển slave có bit SM2 = 1 (trong thanh ghi SCON) và hoạt động ở chế độ UART 9 bit. Như vậy, slave chỉ nhận được dữ liệu khi bit truyền thứ 9 (TB8 của master) là 1.

Sưu tầm bởi: www.daihoc.com.vn

- Mỗi slave được gán trước một địa chỉ. Khi cần trao đổi thông tin với slave nào, master sẽ gởi dữ liệu 9 bit gồm 8 bit địa chỉ của slave và bit 9 = 1. Dữ liệu này sẽ được tất cả các slave nhận về (do bit 9 = 1). Chương trình trong slave sẽ kiểm tra giá trị địa chỉ tương ứng, nếu trùng với địa chỉ đã cài đặt sẵn thì đảo bit SM2 (= 0), nếu khác thì bỏ qua.

- Tiếp tục, master sẽ gởi dữ liệu đến slave nhưng lúc này bit 9 = 0. Khi đó, chỉ có slave nào có bit SM2 = 0 mới nhận được dữ liệu.

- Sau khi truyền xong dữ liệu, master gởi lại 8 bit địa chỉ và bit 9 = 1. Slave nhận được sẽ đảo bit SM2 lần nữa để khôi phục trạng thái ban đầu.

Như vậy, trong quá trình truyền thông đa xử lý, có 2 loại thông tin gởi: byte địa chỉ nếu bit 9 = 1 và byte dữ liệu nếu bit 9 = 0.

2.4. Nhận dạng địa chỉ tự động

Trong các phiên bản mới của MCS-51, địa chỉ của các slave có thể nhận dạng bằng các thanh ghi SADDR và thanh ghi mặt nạ SADEN (các bit không quan tâm trong thanh ghi địa chỉ SADDR sẽ tương ứng với các bit 0 trong thanh ghi SADEN).

Xét hệ thống có 1 master và 3 slave:

Slave 1: SADDR = 1111 0001b, SADEN = 1111 1010b

1111 0001b
1111 1010b

1111 0x0xb

Slave 2: SADDR = 1111 0011b, SADEN = 1111 1001b

1111 0011b
1111 1001b

1111 0xx1b

Slave 3: SADDR = 1111 0001b, SADEN = 1111 1010b

1111 1011b
1111 0101b

1111 x0x1b

Sưu tầm bởi: www.daihoc.com.vn

Nếu chỉ cần gởi dữ liệu cho slave 1, địa chỉ cần sử dụng có bit 0 = 0 (do địa chỉ của slave 2 và slave 3 có bit 0 = 1 còn địa chỉ của slave 1 có bit 0 tuỳ ý), giả sử là 1111 0000b.

Nếu cần gởi cho slave 2 và slave 3 mà không gởi cho slave 1 thì địa chỉ cần dùng có bit 1 = 1 (do địa chỉ của slave 1 có bit 1 = 0 còn slave 2 và 3 thì tuỳ ý), giả sử như 1111 0011b.

❖ **Địa chỉ broadcast**

Địa chỉ broadcast tạo thành từ phép toán OR giữa các thanh ghi SADDR và SADEN trong đó các bit 0 xác định đó là các bit không quan tâm.

Giả sử SADDR = 0101 0000b và SADEN = 1111 1101b thì

```
        0101 0000b
OR      1111 1101b
-------------------
        1111 1101b
```

Địa chỉ broadcast là 1111 11x1b.

2.5. Kiểm tra lỗi khung

Chế độ kiểm tra lỗi khung chỉ có trong các chế độ 1, 2 và 3 được thực hiện bằng cách đặt bit SMOD0 lên 1 (trong thanh ghi PCON). Khi SMOD0 = 1, bộ thu sẽ kiểm tra bit stop mỗi khi có dữ liệu đến. Nếu bit stop không hợp lệ, bit FE sẽ được đặt lên 1 (trong thanh ghi SCON).

Phần mềm sau khi đọc byte dữ liệu sẽ kiểm tra bit FE để xác định có lỗi đường truyền hay không. Lưu ý rằng bit FE chỉ xoá bằng phần mềm hay khi reset hệ thống mà không bị xoá khi nhận bit stop hợp lệ.

2.6. Các ví dụ

Để điều khiển hoạt động của cổng nối tiếp, cần thực hiện các bước sau:

- Khởi động giá trị của thanh ghi SCON để xác định chế độ hoạt động.
- Chọn bộ tạo tốc độ baud (mặc định là timer 1) và xác định các thông số cần thiết theo tốc độ baud yêu cầu.
- Kiểm tra các bit TI và RI để xác định cho phép truyền hay nhận dữ liệu không.
- Nếu cần truyền dữ liệu thì kiểm tra TI và chuyển nội dung truyền vào thanh ghi SBUF.

Sưu tầm bởi: www.daihoc.com.vn

- Nếu cần nhận dữ liệu thì kiểm tra RI và đọc nội dung từ SBUF vào thanh ghi A.

Ví dụ 1: Khởi động cổng nối tiếp ở chế độ UART 8 bit với tốc độ baud 9600 bps, dùng timer 1 là bộ tạo tốc độ baud (giả sử tần số thạch anh là 11.0592 MHz).

Giải

- Nội dung thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
0	1	0	1	0	0	1	0
UART 8 bit		Không ở chế độ đa xử lý	Cho phép thu			Cho phép truyền	

SCON = 0101 0010b (52h)

- Nội dung thanh ghi TMOD:

GATE1	C/T1	M11	M10	GATE0	C/T0	M01	M00
0	0	1	0	0	0	0	0
Không dùng INT1	Đếm bằng dao động nội	Chế độ 8 bit		Timer 0 không dùng			

TMOD = 0010 0000b (20h)

- Giá trị đếm (theo bảng 3.10): TH1 = -3

Đoạn chương trình khởi động như sau:

```
MOV SCON,#52h
MOV TMOD, #20h
MOV TH1,#-3
SETB TR1
```

Ví dụ 2: Viết chương trình xuất liên tục các ký tự từ 'A' đến 'Z' ra cổng nối tiếp với tốc độ baud 4800 bps (giả sử tần số thạch anh là 11.0592 MHz).

Giải

Tốc độ = 4800 bps → giá trị đếm: TH1 = -6

Chương trình thực hiện như sau:

```
     MOV SCON,#52h
     MOV TMOD,#20h
     MOV TH1,#-6
     SETB TR1
Batdau:
     MOV A,#'A'
```


Sưu tầm bởi: www.daihoc.com.vn

```
Truyen:
     JNB TI,$ ; Nếu chưa cho phép truyền thì chờ
     CLR TI   ; Xoá TI để không cho phép truyền, sau khi
              ; truyền xong thì mới có thể truyền tiếp
     MOV SBUF,A    ; Truyền dữ liệu
     INC A         ; Qua ký tự kế
     CJNE A,#'Z'+1,Truyen; Nếu đã truyền xong từ 'A'
     SJMP Batdau        ; đến 'Z' thì lặp lại quá trình
```

Ví dụ 3: Viết chương trình nhận ký tự từ cổng nối tiếp với tốc độ baud 19200bps (giả sử tần số thạch anh là 11.0592 MHz).

Giải

Tốc độ = 1900 bps → giá trị đếm: TH1 = -3 và SMOD = 1

Chương trình thực hiện như sau:

```
     MOV SCON,#52h
     MOV TMOD,#20h
     MOV A,PCON    ; Gán bit SMOD = 1 (do PCON không cho
     SETB ACC.7    ; phép định địa chỉ bit nên phải thực
     MOV PCON,A    ; hiện thông qua thanh ghi A)

     MOV TH1,#-3
     SETB TR1
Nhan:
     JNB RI,$ ; Nếu chưa có ký tự đến thì chờ
     CLR RI   ; Xoá RI để không cho phép nhận, sau khi
              ; có ký tự tiếp theo thì mới nhận
     MOV A,SBUF    ; Nhận dữ liệu
     SJMP Nhan
```

Lưu ý rằng, đối với các ví dụ trên, khi truyền hay nhận dữ liệu thì MCS-51 phải chờ, không được thực hiện công việc khác. Vấn đề này có thể giải quyết bằng cách sử dụng ngắt (xem thêm phần 3).

3. Ngắt (Interrupt)

Ngắt là quá trình dừng chương trình đang thực thi để phục vụ cho một chương trình khác khi xảy ra một sự kiện. Chương trình xử lý sự kiện ngắt gọi là chương trình phục vụ ngắt (ISR – Interrupt Service Routine).

Sưu tầm bởi: www.daihoc.com.vn

Họ MCS-51 có tổng cộng 5 nguồn ngắt khác nhau (không kể reset cũng có thể xem như là một ngắt): ngắt ngoài 0, 1 (tại các chân $\overline{INT0}$, $\overline{INT1}$), timer 0, 1 (khi timer tương ứng tràn), cổng nối tiếp (khi có ký tự đến hay khi truyền ký tự đi). Đối với họ 89x52 sẽ có thêm ngắt timer 2.

3.1. Các thanh ghi điều khiển hoạt động

3.1.1. Thanh ghi IE (Interrupt Enable)

Bảng 3.11 – Nội dung thanh ghi IE

EA	-	ET2	ES	ET1	EX1	ET0	EX0

Bit	Ký hiệu	Địa chỉ	Mô tả
IE.7	EA	AFh	Enable All Cấm tất cả (= 0) hay cho phép ngắt
IE.6	-		
IE.5	ET2	ADh	Enable Timer 2 Cho phép ngắt tại timer 2 (= 1)
IE.4	ES	ACh	Enable serial port Cho phép ngắt tại cổng nối tiếp (= 1)
IE.3	ET1	ABh	Enable Timer 1 Cho phép ngắt tại timer 1 (= 1)
IE.2	EX1	AAh	Enable External interrupt 1 Cho phép ngắt tại ngắt ngoài 1 (= 1)
IE.1	ET0	A9h	Enable Timer 0 Cho phép ngắt tại timer 0 (= 1)
IE.0	EX0	A8h	Enable External interrupt 0 Cho phép ngắt tại ngắt ngoài 0 (= 1)

Giá trị khi reset: 00h, cho phép định địa chỉ bit

Thanh ghi IE cho phép một ngắt có xảy ra hay cấm ngắt (để cho phép cần dùng 2 bit: bit EA = 1 và bit cho phép tương ứng từng ngắt).

3.1.2. Thanh ghi IP (Interrupt Priority)

Bảng 3.12 – Nội dung thanh ghi IP

-	-	PT2	PS	PT1	PX1	PT0	PX0

Bit	Ký hiệu	Địa chỉ	Mô tả
IP.7	-		
IP.6	-		
IP.5	PT2	BDh	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại timer 2
IP.4	PS	BCh	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại cổng nối tiếp

Sưu tầm bởi: www.daihoc.com.vn

IP.3	PT1	BBh	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại timer 1
IP.2	PX1	BAh	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại ngắt ngoài 1
IP.1	PT0	B9h	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại timer 0
IP.0	PX0	B8h	Chọn mức ưu tiên cao (= 1) hay thấp (= 0) tại ngắt ngoài 0

Giá trị khi reset: 00h, cho phép định địa chỉ bit

Thanh ghi IP cho phép chọn mức ưu tiên cho các ngắt. Họ MCS-51 có 2 mức ưu tiên: mức cao và mức thấp. Quá trình xử lý ưu tiên ngắt mô tả như sau:

- Nếu 2 ngắt xảy ra đồng thời thì ngắt nào có mức ưu tiên cao hơn sẽ được phục vụ trước.
- Nếu 2 ngắt xảy ra đồng thời có cùng mức ưu tiên thì thứ tự ưu tiên thực hiện từ cao đến thấp như sau: ngắt ngoài 0 – timer 0 – ngắt ngoài 1 – timer 1 – cổng nối tiếp – timer 2.
- Nếu ISR của một ngắt có mức ưu tiên thấp đang chạy mà có ngắt khác xảy ra với mức ưu tiên cao thì ISR này sẽ tạm dừng để chạy ISR có mức ưu tiên cao (cũng có nghĩa là không thể dừng ISR có mức ưu tiên cao).

3.1.3. Thanh ghi TCON (Timer/Counter Control)

Bảng 3.13 – Nội dung thanh ghi TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	Ký hiệu	Địa chỉ	Mô tả
TCON.7	TF1	8Fh	Xem phần timer
TCON.6	TR1	8Eh	
TCON.5	TF0	8Dh	
TCON.4	TR0	8Ch	
TCON.3	IE1	8Bh	Cờ ngắt ngoài 1 Đặt bằng 1 khi phát hiện tác động ngắt tại $\overline{INT1}$ Xoá bằng phần mềm hay bằng phần cứng khi chuyển điều khiển đến ISR
TCON.2	IT1	8Ah	Interrupt 1 Type control bit = 0: ngắt ngoài 1 được tác động bằng mức logic 0 = 1: ngắt ngoài 1 được tác động bằng cạnh âm
TCON.1	IE0	89h	Dùng cho ngắt ngoài 0
TCON.0	IT0	88h	

Giá trị khi reset: TCON = 00h

Sưu tầm bởi: www.daihoc.com.vn

3.2. Xử lý ngắt

Để kiểm tra khi nào ngắt xảy ra, các cờ ngắt được lấy mẫu ở thời gian S5P2 của mỗi chu kỳ máy. Các điều kiện ngắt được hỏi vòng cho đến chu kỳ máy kế tiếp để xác định xem có ngắt xảy ra hay không. Khi có điều kiện ngắt, hệ thống ngắt sẽ tạo ra lệnh LCALL để gọi ISR tương ứng nhưng lệnh này sẽ không được thực hiện khi tồn tại một trong các điều kiện sau:

- Có một ngắt có mức ưu tiên bằng hay cao hơn đang được phục vụ.
- Chu kỳ hỏi vòng hiện tại không phải là chu kỳ cuối của một lệnh.
- Đang thực thi lệnh RETI hay bất kỳ lệnh nào có ảnh hưởng đến thanh ghi IE và IP.

Khi có ngắt xảy ra, các thao tác thực hiện lần lượt là:

- Hoàn tất lệnh hiện hành.
- Cất nội dung của thanh ghi PC vào stack.
- Lưu trạng thái của ngắt hiện hành.
- Đưa vào thanh ghi PC địa chỉ của ISR tương ứng.

Sau khi thực hiện xong ISR (kết thúc bằng lệnh RETI), thực hiện quá trình: khôi phục trạng thái ban đầu của ngắt và lấy địa chỉ từ stack đưa vào PC.

❖ Bảng vector ngắt

Khi xảy ra ngắt, thanh ghi PC sẽ được nạp giá trị tương ứng với các ngắt. Các giá trị này được gọi là vector ngắt, mô tả như sau:

Bảng 3.14 – Bảng vector ngắt

Nguyên nhân ngắt	**Địa chỉ**
Reset	0000h
Ngắt ngoài 0	0003h
Timer 0	000Bh
Ngắt ngoài 1	0013h
Timer 1	001Bh
Cổng nối tiếp	0023h
Timer 2	002Bh

Trong các nguyên nhân này, reset có thể được xem như một ngắt có vector ngắt là 0000h nhưng cách xử lý khi reset không giống như ngắt: khởi động tất cả các thanh ghi về giá trị mặc định và không lưu nội dung của PC vào stack.

Sưu tầm bởi: www.daihoc.com.vn

Theo bảng vector ngắt, ISR của ngắt ngoài 0 nằm từ địa chỉ 0003h đến 000Ah (chiếm tổng cộng 8 byte) nên khi sử dụng ISR có kích thước thấp hơn 9 byte thì có thể dùng trực tiếp tại địa chỉ 0003h (xem thêm phần sau). Tuy nhiên, nếu kích thước ISR lớn hơn thì phải dùng các lệnh nhảy tại các vector ngắt. Khi đó chương trình sẽ có cấu trúc như sau (tên của các ISR có thể thay đổi):

```
     ORG 0000h
     LJMP main
     ORG 0003h
     LJMP Int0_ISR
     ORG 000Bh
     LJMP Timer0_ISR
     ORG 0013h
     LJMP Int1_ISR
     ORG 001Bh
     LJMP Timer1_ISR
     ORG 0023h
     LJMP Serial_ISR
Main:
     ……
     ……
Int0_ISR:
     ……
     RETI
Timer0_ISR:
     ……
     RETI
Int1_ISR:
     ……
     RETI
Timer1_ISR:
     ……
     RETI
Serial_ISR:
     ……
     RETI
END
```

Lưu ý rằng nếu không sử dụng ngắt nào thì không cần phải khai báo ISR cho ngắt đó.

Sưu tầm bởi: www.daihoc.com.vn

3.3. Ngắt do bộ định thời

MCS-51 có 2 nguồn ngắt từ timer: timer 0 và timer 1 (đối với họ 89x52 còn có thêm timer 2). Khi timer hoạt động ở chế độ ngắt, chương trình vẫn hoạt động bình thường cho đến khi timer tràn thì mới chuyển đến vị trí của ISR (trong khi đó, khi timer hoạt động không sử dụng ngắt thì chương trình sẽ dừng lại – xem thêm phần ví dụ trong hoạt động định thời).

Các nguồn ngắt này cho phép hay cấm bằng các bit trong thanh ghi IE: EA, ET0, ET1 và chọn chế độ ưu tiên bằng các bit trong thanh ghi IP: PT0, PT1. Khi timer tràn, cờ TFx sẽ chuyển lên mức 1. Hệ thống ngắt khi phát hiện cờ TFx lên 1 sẽ chuyển đến ISR tương ứng và tự động xoá cờ TFx.

Quá trình điều khiển hoạt động bằng bộ định thời có sử dụng ngắt thực hiện như sau:

- Xác định chế độ hoạt động của bộ định thời.
- Nạp giá trị cho các thanh ghi THx, TLx.
- Cho phép ngắt tại các bộ định thời tương ứng (thanh ghi IE).
- Xác định mức ưu tiên (thanh ghi IP).
- Cho phép timer chạy bằng các bit TRx.
- Viết ISR cho timer tương ứng.

Ví dụ 1: Viết chương trình tạo sóng vuông tần số f = 5 KHz tại P1.0 dùng ngắt timer 1 (giả sử tần số thạch anh là 12 MHz).

Giải

f = 5 KHz → T = 200 µs (200 chu kỳ) → thời gian trì hoãn: 100 chu kỳ

Giá trị đếm = 100 → dùng chế độ 8 bit

TMOD = 0010 0000b (20h)

- Nội dung thanh ghi IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0
1	0	0	0	1	0	0	0

IE = 1000 1000b (88h)

Chương trình thực hiện như sau:

```
ORG 0000h
```


Sưu tầm bởi: www.daihoc.com.vn

```
     LJMP main
     ORG 001Bh
     CPL P1.0       ; đảo bit
     RETI           ; trở về chương trình chính từ ISR
Main:
     MOV TMOD,#20h
     MOV IE,#88h    ; Có thể thay thế bằng 2 lệnh sau:
                    ; SETB EA
                    ; SETB ET1
     MOV TH1,#(-100)
     MOV TL1,#(-100)
     SETB TR1
     SJMP $         ; Lặp tại chỗ, nghĩa là chương trình
                    ; không làm gì cả, chờ timer tràn (các
                    ; ứng dụng thực tế có thể xử lý các
                    ; công việc khác)
     END
```

Lưu ý rằng lệnh CPL P1.0 chiếm 2 byte, lệnh RETI chiếm 1 byte, tổng cộng ISR cho timer 1 là 3 byte không vượt quá 8 byte nên có thể đặt trực tiếp tại địa chỉ 001Bh.

Ví dụ 2: Viết chương trình tạo xung vuông tần số f = 10KHz tại P1.0 dùng ngắt timer 0 và xung vuông tần số f = 1 KHz tại P1.1 dùng ngắt timer 1.

Giải

Giá trị đếm cho timer 0: 50.

Giá trị đếm cho timer 1: 500.

→ timer 0: 8 bit, timer 1: 16 bit

TMOD = 0001 0010b (12h)

- Nội dung thanh ghi IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0
1	0	0	0	1	0	1	0

IE = 1000 1010b (8Ah)

Chương trình thực hiện như sau:

```
ORG 0000h
LJMP main
```


Sưu tầm bởi: www.daihoc.com.vn

```
      ORG 000Bh
      CPL P1.0
      RETI
      ORG 001Bh
      MOV TH1,#HIGH(-500)     ; 2 byte
      MOV TL1,#LOW(-500)      ; 2 byte
      CPL P1.1                ; 2 byte
      RETI                    ; 1 byte
Main:
      MOV TMOD,#12h
      MOV IE,#8Ah
      SETB TR0
      SETB TR1
      MOV TH1,#HIGH(-500)
      MOV TL1,#LOW(-500)
      MOV TH0,#(-50)
      MOV TL0,#(-50)
      SJMP $
      END
```

Trong ví dụ này, do timer 1 hoạt động ở chế độ 16 bit nên mỗi lần timer tràn phải thực hiện nạp lại giá trị cho timer 1.

Ví dụ 3: Viết chương trình dùng ngắt timer 0 sao cho cứ 1s thì tăng nội dung của các ô nhớ 30h, 31h, 32h theo quy luật đồng hồ (30h chứa giờ, 31h chứa phút, 32h chứa giây).

Giải

Yêu cầu chương trình trì hoãn là 1s trong khi timer 0 cho phép trì hoãn tối đa là 65536 µs → chọn giá trị đếm là 50000 và thực hiện lặp lại 20 lần (20 x 50000 = 1000000 µs = 1s).

TMOD = 0000 0001b (01h)

IE = 1000 0010b (82h)

Chương trình thực hiện như sau:

```
      Hour EQU 30h        ; Định nghĩa trước các ô nhớ
      Minute EQU 31h
      Second EQU 32h
      ORG 0000h
      LJMP main
```


Sưu tầm bởi: www.daihoc.com.vn

```
     ORG 0003h
     LJMP Timer0_ISR
Main:
     MOV TMOD,#01h
     MOV IE,#82h
     MOV TH0,#HIGH(-50000)
     MOV TL0,#LOW(-50000)
     MOV R7,#20              ; Lặp 20 lần
     SETB TR0
     SJMP $
Timer0_ISR:
     MOV TH0,#HIGH(-50000)
     MOV TL0,#LOW(-50000)
     DJNZ R7,exitTimer0      ; Nếu chưa đủ 20 lần thì thoát
     CALL Inc_clock          ; Tăng theo quy luật đồng hồ
     MOV R7,#20
exitTimer0:
     RETI
Inc_clock:
     INC Second              ; Tăng giây
     MOV A,Second
     CJNE A,#60,exitInc      ; Nếu giây < 60 thì thoát
     MOV Second,#0           ; Ngược lại thì gán giây = 0
     INC Minute              ; và tăng phút
     MOV A,Minute
     CJNE A,#60,exitInc      ; Nếu phút < 60 thi thoát
     MOV Minute,#0           ; Ngược lại thì gán phút = 0
     INC Hour                ; và tăng giờ
     MOV A,Hour
     CJNE A,#24,exitInc      ; Nếu giờ < 24 thì thoát
     MOV Hour,#0             ; Ngược lại thì gán giờ = 0
exitInc:
     RET
     END
```

3.4. Ngắt do cổng nối tiếp

MCS-51 có 2 nguồn ngắt do cổng nối tiếp: ngắt phát và ngắt thu. Hai nguồn ngắt này xác định bằng các bit RI, TI và dùng chung một địa chỉ ISR nên khi chuyển đến ISR, các cờ ngắt không tự động xoá bằng phần cứng mà phải thực hiện bằng phần mềm: kiểm tra nguyên nhân ngắt (RI hay TI) và xoá bit cờ tương ứng.

Sưu tầm bởi: www.daihoc.com.vn

Ví dụ: Viết chương trình khởi động cổng nối tiếp ở chế độ UART 8 bit với tốc độ truyền 4800 bps. Viết ISR cho cổng nối tiếp theo yêu cầu: truyền tuần tự các ký tự từ 'A' đến 'Z' ra cổng nối tiếp đồng thời mỗi lần có ký tự đến cổng nối tiếp thì nhận về và xuất ký tự nhận ra P0 (giả sử tần số thạch anh là 11.0592 MHz).

Giải

- Nội dung thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
0	1	0	1	0	0	0	0
UART 8 bit		Không ở chế độ đa xử lý	Cho phép thu			Không cho phép truyền	

SCON = 50h

- Nội dung thanh ghi TMOD:

GATE1	C/T1	M11	M10	GATE0	C/T0	M01	M00
0	0	1	0	0	0	0	0
Không dùng INT1	Đếm bằng dao động nội	Chế độ 8 bit		Timer 0 không dùng			

TMOD = 0010 0000b (20h)

- Giá trị đếm (theo bảng 3.10): TH1 = -6

- Nội dung thanh ghi IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0
1	0	0	1	0	0	0	0

IE = 1001 0000b (90h)

Chương trình thực hiện như sau:

```
     ORG 0000h
     LJMP main
     ORG 0023h      ; Địa chỉ ISR của cổng nối tiếp
     LJMP Serial_ISR
Main:
     MOV TMOD,#20h
     MOV TH1,#(-6)
     MOV TL1,#(-6) ; Tốc độ 4800 bps
     SETB TR1
     MOV R7,#'A'    ; Ký tự truyền đầu tiên
     MOV IE,#90h    ; Cho phép ngắt tại công nối tiếp
     SETB TI        ;Cho phép truyền
     SJMP $
```


Sưu tầm bởi: www.daihoc.com.vn

```
Serial_ISR:
     JNB RI,Transmit    ; Nếu không phải ngắt do nhận
                        ; ký tự thì truyền
     CLR RI
     MOV A,SBUF         ; Nhận ký tự
     MOV P0,A           ; Xuất ra Port 0
     SJMP exitSerial
Transmit:               ; Truyền ký tự
     CLR TI
     MOV A,R7
     MOV SBUF,A         ; Truyền ký tự
     INC R7             ; Qua ký tự kế
     CJNE R7,#'Z'+1,exitSerial   ; Nếu chưa truyền'Z' thì
                        ; tiếp tục truyền, ngược lại thì
     MOV R7,#'A'        ; bắt đầu truyền từ ký tự 'A'
exitSerial:
     RETI
     END
```

3.5. Ngắt ngoài

MCS-51 có 2 nguồn ngắt ngoài khác nhau: ngắt ngoài 0 và ngắt ngoài 1. Ngắt ngoài xảy ra khi bit IEx chuyển lên mức 1, quá trình chuyển mức của bit IEx xảy ra khi:

- Bit ITx = 0 và xuất hiện mức logic 0 tại chân INTx tương ứng (P3.2 cho ngắt ngoài 0 hay P3.3 cho ngắt ngoài 1).
- Bit ITx = 1 và xuất hiện cạnh âm tại chân INTx.

Khi có ngắt xảy ra và cho phép ngắt (dùng thanh ghi IE), chương trình sẽ được chuyển đến địa chỉ của ISR tương ứng (0003h cho ngắt ngoài 0 và 0013h cho ngắt ngoài 1) và xoá cờ ngắt TFx.

Lưu ý rằng các cờ ngắt được lấy mẫu trong mỗi chu kỳ nên để phát hiện ngắt, yêu cầu phải:

- Ở mức thấp tối thiểu 1 chu kỳ nếu tác động bằng mức logic (ITx = 0).
- Ở mức cao tối thiểu 1 chu kỳ trước khi chuyển xuống mức thấp và mức thấp cũng phải tồn tại tối thiểu 1 chu kỳ (ITx = 1).

Quá trình điều khiển ngắt ngoài mô tả như sau:

- Xác định yêu cầu ngắt bằng cạnh âm hay bằng mức logic.

Sưu tầm bởi: www.daihoc.com.vn

- Cho phép ngắt tại ngắt ngoài tương ứng (dùng thanh ghi IE).
- Xác định mức ưu tiên (thanh ghi IP).
- Viết ISR cho các ngắt.

Ví dụ: Viết chương trình sao cho mỗi khi có mức logic 0 xuất hiện tại P3.2 (ngắt ngoài 0) thì tạo xung 1 KHz tại P1.0. Quá trình tạo xung chỉ dừng khi có mức logic 0 xuất hiện tại P3.3 (ngắt ngoài 1).

Giải

Chương trình thực hiện có 3 ngắt xảy ra: ngắt ngoài 0 cho phép timer chạy để tạo xung tại P1.0, ngắt ngoài 1 cấm timer để ngừng quá trình tạo xung và ngắt timrer để tạo xung.

f = 1 KHz → T = 1ms (1000 chu kỳ): giá trị đếm là 500 (chế độ 16 bit)

- Nội dung thanh ghi TMOD:

GATE1	C/T1	M11	M10	GATE0	C/T0	M01	M00
0	0	0	1	0	0	0	0
Không dùng INT1	Đếm bằng dao động nội	Chế độ 16 bit		Timer 0 không dùng			

TMOD = 0001 0000b (10h)

- Nội dung thanh ghi IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0
1	0	0	0	1	1	0	1

IE = 1000 1101b (8Dh)

Chương trình thực hiện như sau:

```
ORG 0000h
LJMP main

ORG 0003h          ; Địa chỉ ISR ngắt ngoài 0
SETB TR1           ; Timer 1 chạy
RETI

ORG 0013h          ; Địa chỉ ISR của ngắt ngoài 1
CLR TR1            ; Cấm timer 1
RETI

ORG 001Bh          ; Địa chỉ ISR timer 1
MOV TH1,#HIGH(-500); Chế độ 16 bit nên mỗi lần tràn
MOV TL1,#LOW(-500); phải nạp lại giá trị
CPL P1.0           ; Đảo bit P1.0 để tạo xung
RETI
```


Sưu tầm bởi: www.daihoc.com.vn

```
Main:
     MOV TMOD,#10h
     MOV TH1,#HIGH(-500)
     MOV TL1,#LOW(-500)
     MOV IE,#8Dh   ; Cho phép ngắt tại ngắt ngoài 0, 1 và
     SJMP $        ; timer 1
     END
```


Sưu tầm bởi: www.daihoc.com.vn

BÀI TẬP CHƯƠNG 3

1. Viết đoạn chương trình theo yêu cầu:

- Khởi động cổng nối tiếp ở chế độ UART 8 bit với tốc độ truyền 4800 bps.
- Định thời 1s thì đọc dữ liệu từ P1, lưu vào ô nhớ 30h và xuất dữ liệu vừa đọc ra cổng nối tiếp.

2. Viết đoạn chương trình theo yêu cầu:

- Khởi động cổng nối tiếp ở chế độ UART 9 bit với tốc độ truyền 9600 bps.
- Khi có ngắt xảy ra tại ngắt ngoài 0 thì xuất dữ liệu tại ô nhớ 30h ra cổng nối tiếp trong đó bit truyền thứ 9 là bit parity.
- Khi có ngắt tại ngắt ngoài 1 thì đọc dữ liệu từ P0 và lưu kết quả vào ô nhớ 30h.

3. Viết đoạn chương trình theo yêu cầu:

- Khi có ngắt tại ngắt ngoài 0 (tác động bằng cạnh) thì đọc dữ liệu tại P2 và lưu vào ô nhớ 30h đồng thời tăng giá trị trong ô nhớ lên 1.
- Dùng ngắt timer 0 định thời 30s thì đọc giá trị trong ô nhớ 30h, xoá nội dung trong ô nhớ 31h và kiểm tra giá trị theo yêu cầu:

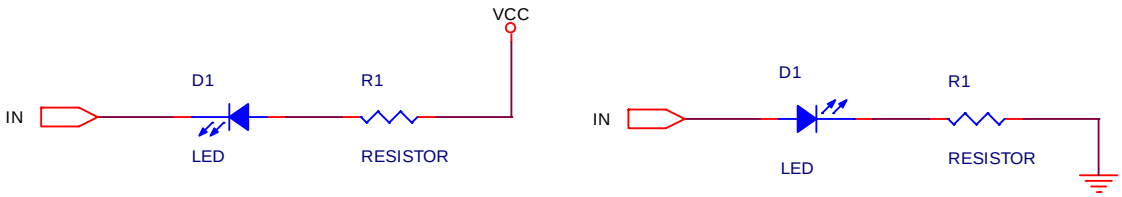
Giá trị	**Thực hiện**
> 200	Đặt bit P1.0 = 1, xoá bit P1.1 = 0 và P1.2 = 0 Tạo xung f = 1KHz tại P1.3 dùng ngắt timer 1
< 100	Đặt bit P1.1 = 1, xoá bit P1.0 = 0 và P1.2 = 0 Ngừng tạo xung tại P1.3
Khác	Đặt bit P1.2 = 1, xoá bit P1.0 = 0 và P1.1 = 0

Sưu tầm bởi: www.daihoc.com.vn

Chương 4: CÁC ỨNG DỤNG DỰA TRÊN VI ĐIỀU KHIỂN MCS-51

Chương này giới thiệu về một số ứng dụng của MCS-51 trong thực tế: điều khiển Led đơn, Led 7 đoạn, ma trận Led, LCD, động cơ bước, giao tiếp 8255.

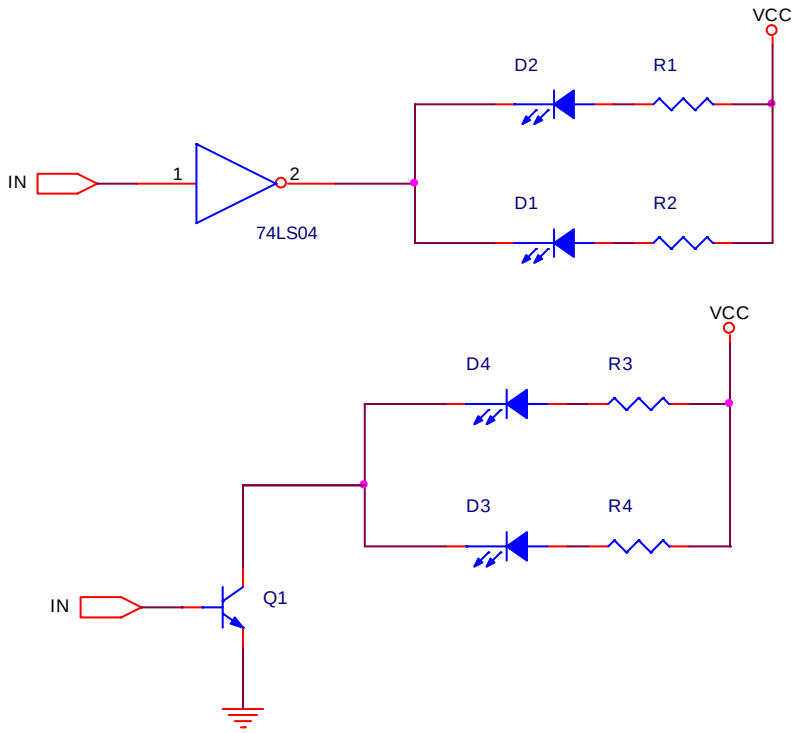
1. Điều khiển Led đơn



Hình a Hình b

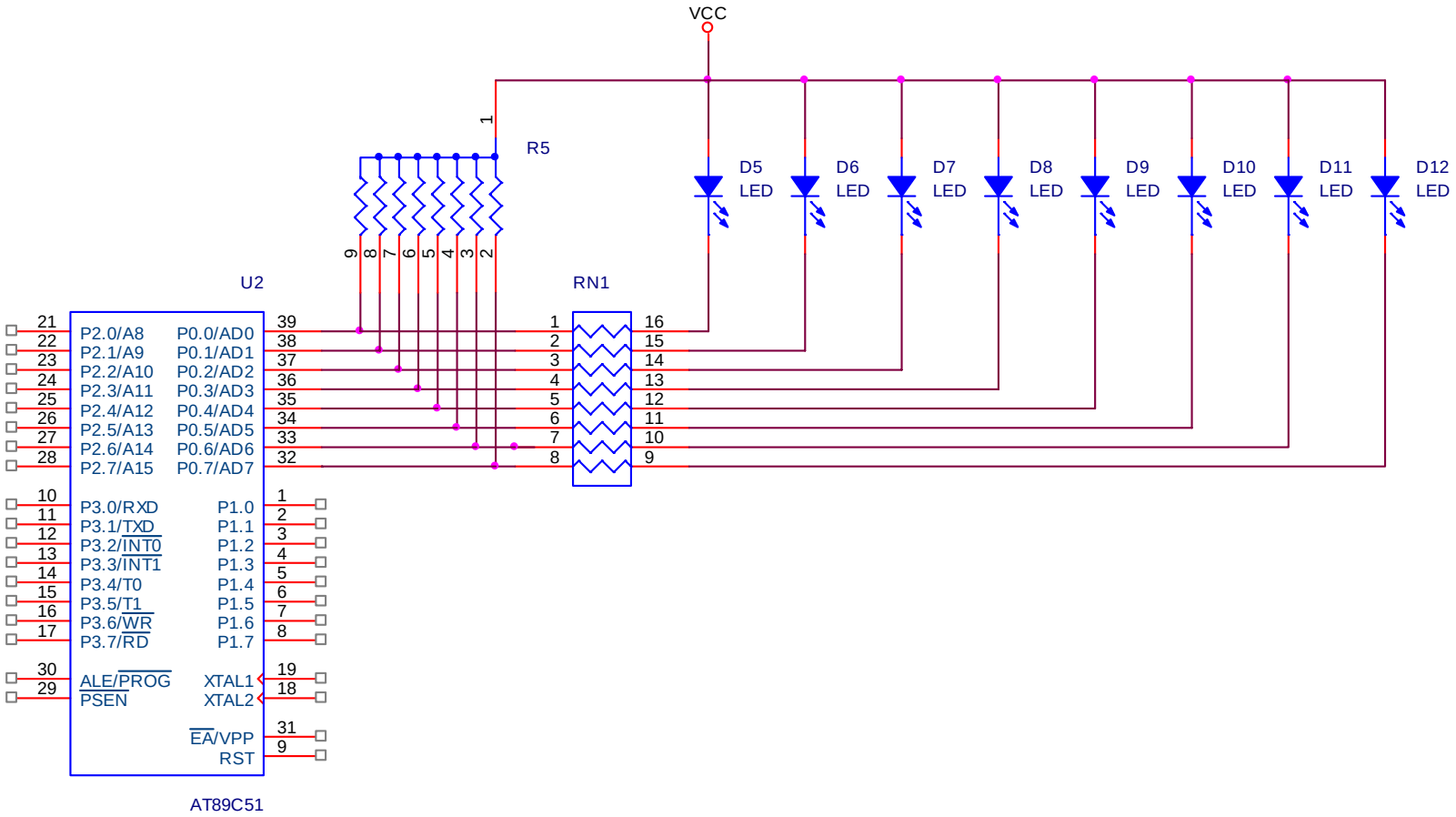
Hình 4.1 – Sơ đồ kết nối Led đơn

Mạch điều khiển led đơn mô tả như hình 4.1. Lưu ý rằng các port của AT89C51 có dòng tối đa là 10 mA (xem thêm chương 1, phần đặc tính DC) nên khi cần điều khiển nhiều Led cần mắc thêm mạch khuếch đại.



Hình 4.2 – Sơ đồ kết nối dùng mạch khuếch đại

Sưu tầm bởi: www.daihoc.com.vn



Hình 4.3 – Kết nối Led đơn với AT89C51

Sưu tầm bởi: www.daihoc.com.vn

Ví dụ: Xét sơ đồ kết nối Led như hình 4.3. Viết chương trình điều khiển Led sáng tuần tự từ trái sang phải, mỗi lần 1 Led.

Giải

Các Led nối với Port 0 của AT89C51 (P0 khi dùng như các cổng nhập / xuất thì cần phải có điện trở kéo lên nguồn) nên muốn Led sang thì phải gởi dữ liệu ra P0. Theo sơ đồ mạch, Led sang khi các bit tương ứng tại P0 là 0.

Yêu cầu điều khiển Led sang từ trái sang phải (theo thứ tự lần lượt từ P0.0 đến P0.7) nên dữ liệu gởi ra là:

- Lần 1: 1111 1110b (0FEh) – sáng 1 Led trái
- Lần 2: 1111 1101b (0FDh)
- Lần 3: 1111 1011b (0FBh)
- Lần 4: 1111 0111b (0F7h)
- Lần 5: 1110 1111b (0EFh)
- Lần 6: 1101 1111b (0DFh)
- Lần 7: 1011 1111b (0BFh)
- Lần 8: 0111 1111b (7Fh)
- Lần 9: quay lại giống như lần 1

Chương trình thực hiện như sau:

```
     MOV DPTR,#MaLed          ; DPTR chứa vị trí bảng mã Led
Main:
     MOV R7,#0                ; Phần tử đầu tiên của bảng mã
Loop:
     MOV A,R7
     MOVC A,@A+DPTR           ; Đọc bảng mã
     MOV P0,A                 ; Chuyển vào P0 để sáng Led
     CALL Delay               ; Chờ để mắt người có thể thấy
     INC R7                   ; Chuyển qua trạng thái kế
     CJNE R7,#8,Loop          ; Đã hết bảng mã thì lặp lại
     SJMP main
MaLed: DB 0FEh,0FDh,0FBh,0F7h,0EFh,0DFh,0BFh,7Fh
Delay:
     MOV TMOD,#01h
     MOV TH0,#HIGH(-50000)    ; Chờ 50 ms
     MOV TL0,#LOW(-50000)
     SETB TR0
     JNB TF0,$
     CLR TF0
```

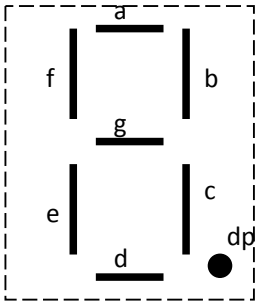

Sưu tầm bởi: www.daihoc.com.vn

```
CLR TR0
RET
END
```

2. Điều khiển Led 7 đoạn

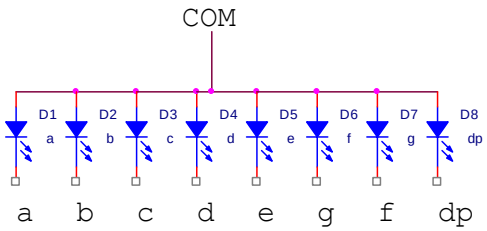
2.1. Cấu trúc và bảng mã hiển thị dữ liệu trên Led 7 đoạn

- Dạng Led:



Hình 4.4 – Hình dạng của Led 7 đoạn

- Led Anode chung:



Hình 4.5 – Led 7 đoạn dạng anode chung

Đối với dạng Led anode chung, chân COM phải có mức logic 1 và muốn sáng Led thì tương ứng các chân a – f, dp sẽ ở mức logic 0.

Bảng 4.1 - Bảng mã cho Led Anode chung (a là MSB, dp là LSB):

Số	a	b	c	d	e	f	g	dp	Mã hex
0	0	0	0	0	0	0	1	1	03h
1	1	0	0	1	1	1	1	1	9Fh
2	0	0	1	0	0	1	0	1	25h
3	0	0	0	0	1	1	0	1	0Dh
4	1	0	0	1	1	0	0	1	99h
5	0	1	0	0	1	0	0	1	49h
6	0	1	0	0	0	0	0	1	41h

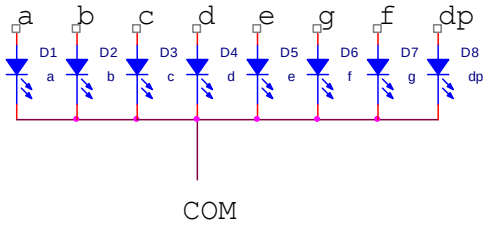
Sưu tầm bởi: www.daihoc.com.vn

7	0	0	0	1	1	1	1	1	1Fh
8	0	0	0	0	0	0	0	1	01h
9	0	0	0	0	1	0	0	1	09h

Bảng 4.2 - Bảng mã cho Led Anode chung (a là LSB, dp là MSB):

Số	**dp**	**g**	**f**	**e**	**d**	**c**	**b**	**a**	**Mã hex**
0	1	1	0	0	0	0	0	0	0C0h
1	1	1	1	1	1	0	0	1	0F9h
2	1	0	1	0	0	1	0	0	0A4h
3	1	0	1	1	0	0	0	0	0B0h
4	1	0	0	1	1	0	0	1	99h
5	1	0	0	1	0	0	1	0	92h
6	1	0	0	0	0	0	1	0	82h
7	1	1	1	1	1	0	0	0	0F8h
8	1	0	0	0	0	0	0	0	80h
9	1	0	0	1	0	0	0	0	90h

- Led Cathode chung



Hình 4.6 – Led 7 đoạn dạng cathode chung

Đối với dạng Led Cathode chung, chân COM phải có mức logic 0 và muốn sáng Led thì tương ứng các chân a – f, dp sẽ ở mức logic 1.

Bảng 4.3 - Bảng mã cho Led Cathode chung (a là MSB, dp là LSB):

Số	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**dp**	**Mã hex**
0	1	1	1	1	1	1	0	0	0FCh
1	0	1	1	0	0	0	0	0	60h
2	1	1	0	1	1	0	1	0	0DAh
3	1	1	1	1	0	0	1	0	0F2h
4	0	1	1	0	0	1	1	0	66h
5	1	0	1	1	0	1	1	0	0B6h
6	1	0	1	1	1	1	1	0	0BEh
7	1	1	1	0	0	0	0	0	0E0h

Sưu tầm bởi: www.daihoc.com.vn

8	1	1	1	1	1	1	1	0	0FEh
9	1	1	1	1	0	1	1	0	0F6h

Bảng 4.4 - Bảng mã cho Led Anode chung (a là LSB, dp là MSB):

Số	**dp**	**g**	**f**	**e**	**d**	**c**	**b**	**a**	**Mã hex**
0	0	0	1	1	1	1	1	1	3Fh
1	0	0	0	0	0	1	1	0	06h
2	0	1	0	1	1	0	1	1	5Bh
3	0	1	0	0	1	1	1	1	4Fh
4	0	1	1	0	0	1	1	0	66h
5	0	1	1	0	1	1	0	1	6Dh
6	0	1	1	1	1	1	0	1	7Dh
7	0	0	0	0	0	1	1	1	07h
8	0	1	1	1	1	1	1	1	7Fh
9	0	1	1	0	1	1	1	1	6Fh

2.2. Các phương pháp hiển thị dữ liệu

2.2.1. Phương pháp quét

Khi kết nối chung các đường dữ liệu của Led 7 đoạn, các Led không thể sáng đồng thời (do ảnh hưởng lẫn nhau giữa các Led) mà phải thực hiện quét Led, nghĩa là tại mỗi thời điểm chỉ sáng một Led và tắt các Led còn lại. Do hiện tượng lưu ảnh của mắt, ta sẽ thấy các Led sáng đồng thời.

Ví dụ 1: Xét sơ đồ kết nối như hình 4.7. Viết chương trình hiển thị số 0 ra Led1 và số 1 ra Led2.

Giải

Led có chân COM nối với Vcc (thông qua Q2, Q3) nên Led là loại anode chung và Q2, Q3 là transistor PNP nên để Led sáng thì dữ liệu tương ứng tại các chân điều khiển (P1.0, P1.1) phải là 1.

Theo sơ đồ kết nối, chân g của Led nối với P0.6, chân a nối với P0.0 nên bảng mã Led là bảng 4.2, dữ liệu cho số 0 và 1 lần lượt là 0C0h và 0F9h.

Phương pháp sử dụng là phương pháp quét nên cần phải có thời gian trì hoãn giữa 2 lần quét, thời gian này được thực hiện thông qua timer (thời gian trì hoãn khoảng 200 µs).

Sưu tầm bởi: www.daihoc.com.vn

Chương trình thực hiện như sau:

```
     MOV P1,#0      ; Xoá P1 để tắt Led
Main:
     MOV P0,#0C0h   ; Mã số 0
     SETB P1.0      ; Sáng Led1
     CALL Delay     ; Thời gian trì hoãn để thấy Led sáng
     CLR P1.0       ; Tắt Led1
     MOV P0,#0F9h   ; Mã số 1
     SETB P1.1      ; Sáng Led2
     CALL Delay
     CLR P1.1       ; Tắt Led2
     SJMP main
     ;--------------------
Delay:
     MOV TMOD,#01h
     MOV TH0,#(-200)
     MOV TL0,#(-200)
     SETB TR0
     JNB TF0,$
     CLR TF0
     CLR TR0
     RET

     END
```

Ví dụ 2: Viết lại chương trình trên nhưng sử dụng ngắt của timer.

Giải

Đối với chương trình trong ví dụ 1, khi đang thực hiện quét led thì chương trình không làm gì cả trong khi đó, các ứng dụng thực tế thường xử lý các công việc khác đồng thời với quá trình quét. Vấn đề này có thể giải quyết bằng cách sử dụng ngắt của timer: mỗi khi timer tràn thì thực hiện hiển thị trên 1 Led.

Chương trình thực hiện như sau:

```
     Led1 EQU 30h       ; Địa chỉ chứa dữ liệu của Led1
     Led2 EQU 31h       ; Địa chỉ chứa dữ liệu của Led2
     Led_Pos EQU 32h    ; Vị trí Led hiện hành
     ORG 0000h
     LJMP main
     ORG 000Bh          ; Địa chỉ ISR của timer 0
     LJMP Timer0_ISR
```


Sưu tầm bởi: www.daihoc.com.vn

```
Main:
     SETB EA            ; Cho phép ngắt tại timer 0
     SETB ET0
     MOV Led1,#0C0h     ; Số 0
     MOV Led2,#0F9h     ; Số 1
     MOV Led_Pos,#01h   ; Vị trí sáng đầu tiên là Led1
     MOV R0,#Led1  ; Dữ liệu gởi ra đầu tiên là ở Led1
     MOV TMOD,#01h
     MOV TH0,#(-200)
     MOV TL0,#(-200)
     SETB TR0
     SJMP $         ; Không làm gì cả, các ứng dụng thực tế
                    ; có thể thêm chương trình vào
     ;----------------------
Timer0_ISR:
     MOV A,Led_Pos      ; Xác định vị trí Led hiện hành
     MOV P1,A           ; Sáng Led hiện hành
     RL A               ; Dịch trái để chuyển qua Led kế
     MOV Led_Pos,A      ; trong qua trình tràn tiếp theo
     MOV A,@R0          ; Đọc dữ liệu hiện hành
     MOV P0,A
     INC R0             ; Chuyển qua dữ liệu kế
     CJNE R0,#Led_Pos,exitTimer0 ; Nếu đã quét hết toàn bộ
     MOV Led_Pos,#01h   ; Led thì bắt đầu lại từ Led1
     MOV R0,#Led1
exitTimer0:
     RETI
     END
```

Ví dụ 2 có thể mở rộng thêm cho 8 Led trong đó các bit điều khiển từ P1.0 đến P1.7 bằng cách khai báo thêm các ô nhớ cho các Led như sau:

```
     Led1 EQU 30h       ; Địa chỉ chứa dữ liệu của Led1
     Led2 EQU 31h       ; Địa chỉ chứa dữ liệu của Led2
     Led3 EQU 32h
     Led4 EQU 33h
     Led5 EQU 34h
     Led6 EQU 35h
     Led7 EQU 36h
     Led8 EQU 37h
     Led_Pos EQU 38h    ; Vị trí Led hiện hành
```


Sưu tầm bởi: www.daihoc.com.vn

Ví dụ 3: Viết chương trình hiển thị nội dung trong ô nhớ 30h ra 2 Led trong đó Led1 chứa số hàng chục và Led2 chứa số hàng đơn vị (giả sử giá trị trong ô nhớ 30h tối đa là 99).

Giải

Để xuất nội dung trong ô nhớ 30h ra Led 7 đoạn cần thực hiện:

- Chuyển nội dung trong ô nhớ 30h thành số hàng chục và hàng đơn vị (thực hiện chia cho 10).
- Chuyển giá trị số thành mã Led 7 đoạn (bằng cách tra bảng).

Chương trình thực hiện như sau:

```
     Led1 EQU 30h       ; Địa chỉ chứa dữ liệu của Led1
     Led2 EQU 31h       ; Địa chỉ chứa dữ liệu của Led2
     Led_Pos EQU 32h    ; Vị trí Led hiện hành
     ORG 0000h
     LJMP main
     ORG 000Bh          ; Địa chỉ ISR của timer 0
     LJMP Timer0_ISR
Main:
     SETB EA            ; Cho phép ngắt tại timer 0
     SETB ET0
     MOV Led_Pos,#01h   ; Vị trí sáng đầu tiên là Led1
     MOV R0,#Led1  ; Dữ liệu gởi ra đầu tiên là ở Led1
     MOV TMOD,#01h
     MOV TH0,#(-200)
     MOV TL0,#(-200)
     SETB TR0
Begin:
     MOV A,30h
     CALL Chuyenma
     SJMP Begin
     ;----------------------
Chuyenma:
     MOV B,#10      ; Chia cho 10: A chứa số hàng chục,
     DIV  AB        ; B chứa số hàng đơn vị
     CALL BCDtoLed7     ; Chuyển sang mã Led 7 đoạn
     MOV Led1,A     ; Đưa vào ô nhớ 31h (Led1)
     MOV A,B        ; Chuyển sang mã Led 7 đoạn của
     CALL BCDtoLed7; số hàng đơn vị
     MOV Led2,A
```


Sưu tầm bởi: www.daihoc.com.vn

```
     RET
     ;----------------------
BCDtoLed7:
     MOV DPTR,#MaLed7
     MOVC A,@A+DPTR
     RET
MaLed7: DB 0C0h,0F9h,0A4h,0B0h,99h,92h,82h,0F8h,80h,90h
     ;----------------------
Timer0_ISR:
     PUSH ACC
     MOV A,Led_Pos      ; Xác định vị trí Led hiện hành
     MOV P1,A           ; Sáng Led hiện hành
     RL A               ; Dịch trái để chuyển qua Led kế
     MOV Led_Pos,A      ; trong qua trình tràn tiếp theo
     MOV A,@R0          ; Đọc dữ liệu hiện hành
     MOV P0,A
     INC R0             ; Chuyển qua dữ liệu kế
     CJNE R0,#Led_Pos,exitTimer0 ; Nếu đã quét hết toàn bộ
     MOV Led_Pos,#01h   ; Led thì bắt đầu lại từ Led1
     MOV R0,#Led1
exitTimer0:
     POP ACC
     RETI
     END
```

2.2.2. Phương pháp chốt

Khi thực hiện tách riêng các đường dữ liệu của Led, ta có thể cho phép các Led sáng đồng thời mà sẽ không có hiện tượng ảnh hưởng giữa các Led. IC chốt cho phép lưu trữ dữ liệu cho các Led có thể sử dụng là 74LS373, 74LS374. Khi thực hiện bằng phương pháp chốt, khi nào cần xuất dữ liệu ra Led thì gởi dữ liệu và tạo xung để chốt.

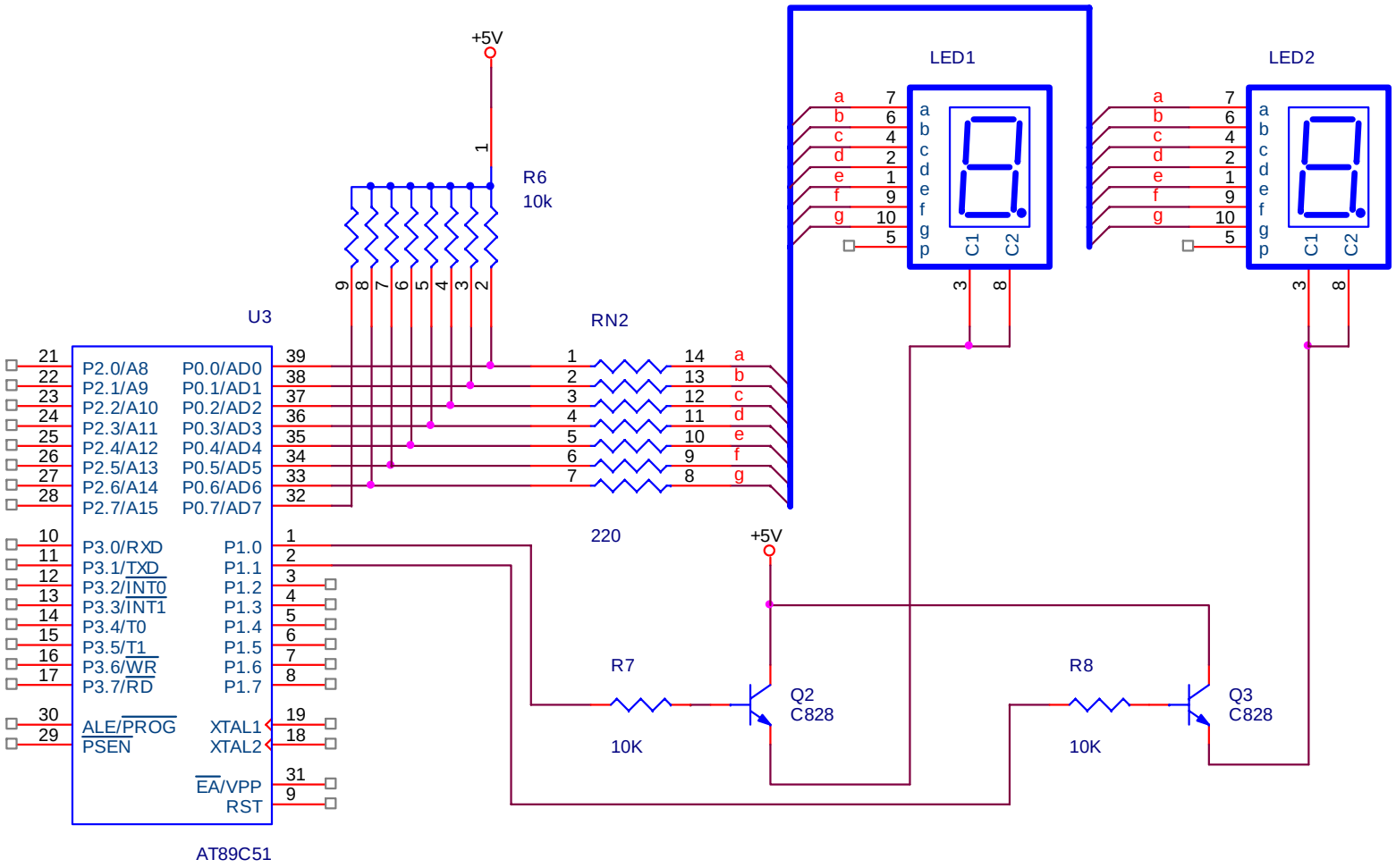
Ví dụ: Xét sơ đồ mạch kết nối như hình 4.8. Viết chương trình xuất số 2 ra Led3 và số 3 ra Led4.

Giải

Do Led3 nối với 74LS374 (U5) điều khiển bằng chân P1.0 nên để hiển thị trên Led3, cần phải:

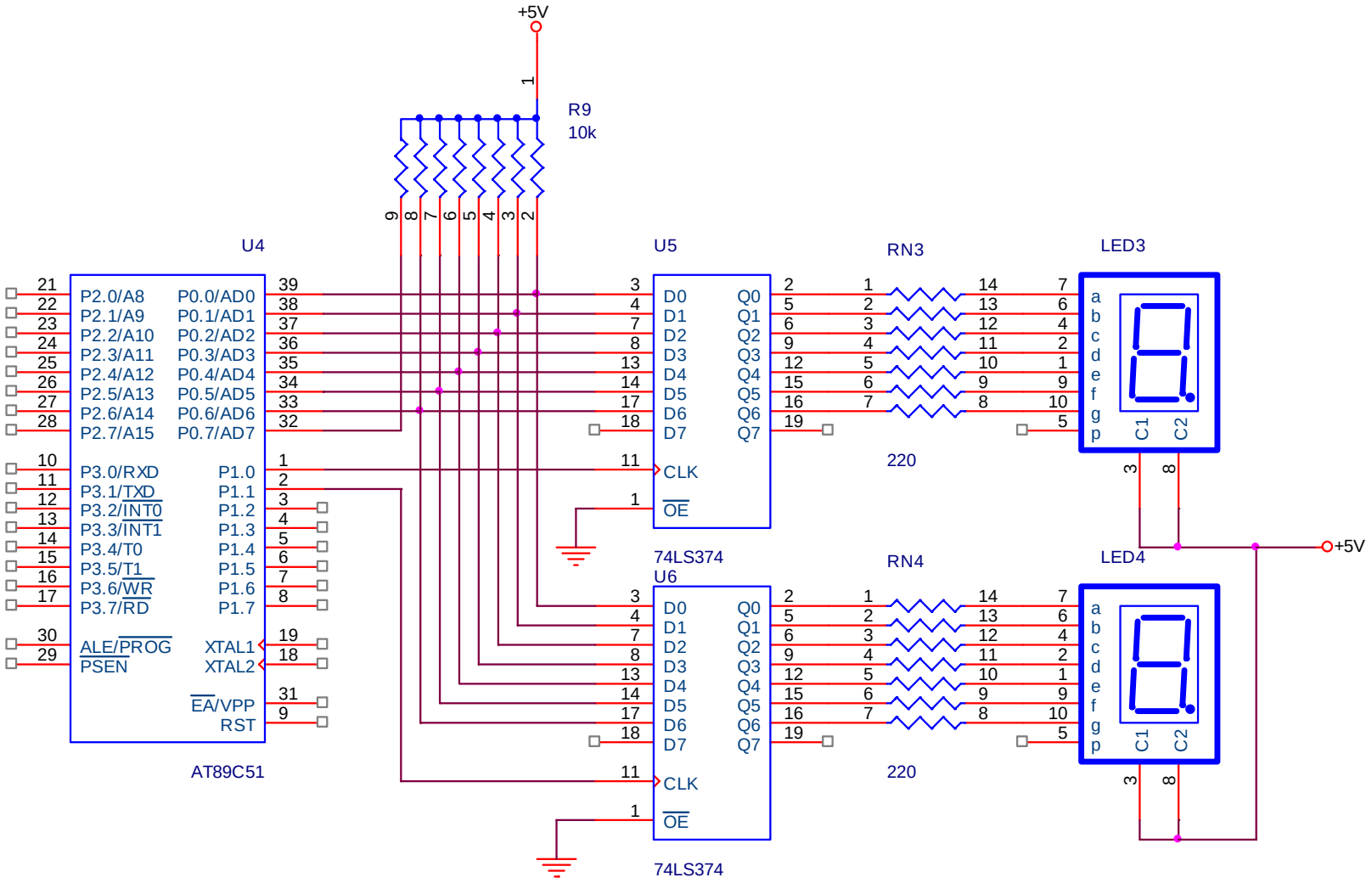
- Xuất dữ liệu ra P0.
- Kích xung tại chân P1.0 để chốt dữ liệu

Sưu tầm bởi: www.daihoc.com.vn



Hình 4.7 – Kết nối Led 7 đoạn dùng phương pháp quét

Sưu tầm bởi: www.daihoc.com.vn



Hình 4.8 – Kết nối Led 7 đoạn dùng phương pháp chốt

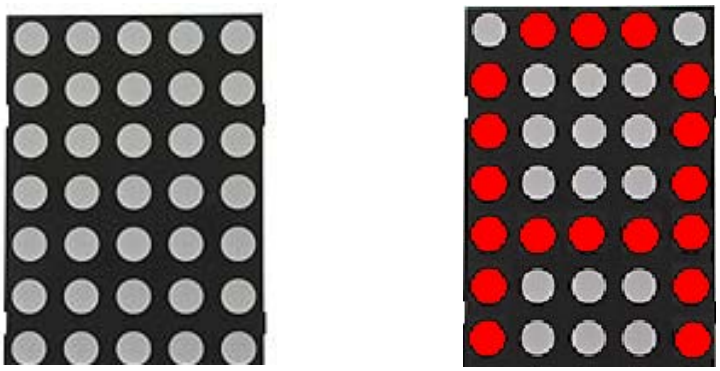
Sưu tầm bởi: www.daihoc.com.vn

Chương trình thực hiện như sau:

```
MOV P0,#0B0h
CLR P1.0
SETB P1.0
MOV P0,#99h
CLR P1.1
SETB P1.1
END
```

3. Điều khiển ma trận Led

Ma trận LED bao gồm nhiều LED cùng nằm trong một vỏ chia thành nhiều cột và hàng, mỗi giao điểm giữa hàng và cột có thể có 1 LED (ma trận LED một màu) hay nhiều LED (2 LED tại một vị trí tạo thành ma trận LED 3 màu). Để LED tại một vị trí nào đó sáng thì phải cấp hiệu điện thế dương giữa Anode và Cathode. Trên cơ sở cấu trúc như vậy, ta có thể mở rộng hàng và cột của ma trận LED để tạo thành các bảng quang báo.



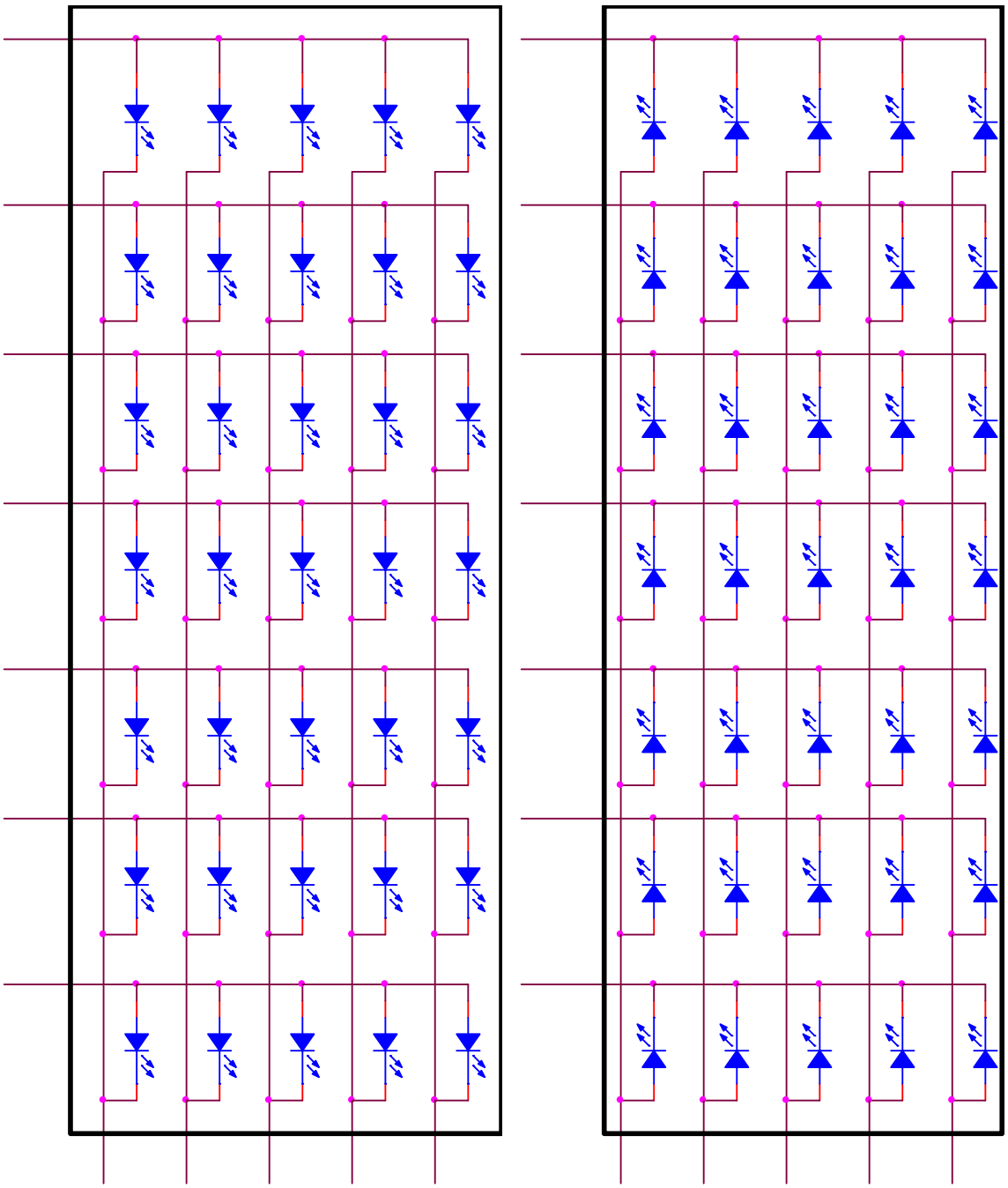
Hình 4.9 – Hình dạng ma trận Led

Kết nối của ma trận Led có 2 cách: anode nối với hàng, cathode nối với cột hay ngược lại. Sơ đồ kết nối mô tả như hình 4.10. Theo cấu trúc kết nối như hình vẽ, 2 Led trên 2 cột không thể sáng đồng thời. Xét sơ đồ kết nối như mạch hình b, một Led sáng khi tương ứng hàng của Led = 0 và cột = 1.

Giả sử ta cần sáng Led đồng thời tại hàng 1, cột 1 và hàng 2, cột 2. Như vậy, ta phải có hàng 1 = 0, cột 1 = 1 (sáng Led tại hàng 1, cột 1) và hàng 2 = 0, cột 2 = 1 (sáng Led tại hàng 2, cột 2). Từ đó, do hàng 1 = 0, cột 2 = 1 và hàng 2 = 0, cột 2 = 1 nên ta cũng có các Led tại hàng 1, cột 2 và hàng 2, cột 1 cũng sáng. Nghĩa là, khi ta cho 2 Led tại hàng 1, cột 1 và hàng 2, cột 2 sáng đồng thời thì sẽ dẫn đến các Led tại hàng 1, cột 2 và hàng 2, cột 1 cũng sáng.

Do đó, để thực hiện sáng một ký tự trên ma trận Led, ta phải dùng cơ chế quét, tại mỗi thời điểm chỉ sáng 1 cột, các cột còn lại tắt đi nhưng nếu cho thời gian quét đủ nhanh thì ta vẫn thấy giống như các cột sáng đồng thời.

Sưu tầm bởi: www.daihoc.com.vn



Hình a Hình b

Hình 4.10 – Sơ đồ kết nối ma trận Led

Sưu tầm bởi: www.daihoc.com.vn

Dữ liệu cho số 0:

	X	X	X	
X				X
X				X
X				X
X				X
X				X
	X	X	X	

Để sáng số 0 trên ma trận Led, ta thực hiện quá trình quét như sau:

Lần 1: Hàng = 0100 0001b, cột = 0001 0000b
Lần 2: Hàng = 0011 1110b, cột = 0000 1000b
Lần 3: Hàng = 0011 1110b, cột = 0000 0100b
Lần 4: Hàng = 0011 1110b, cột = 0000 0010b
Lần 5: Hàng = 0100 0001b, cột = 0000 0001b

Ví dụ: Xét sơ đồ kết nối ma trận Led như hình 4.11. Viết chương trình sáng số 0 trên ma trận Led.

Giải

```
main:
     MOV R0,#0
lap:
     MOV A,R0
     MOV DPTR,#cot
     MOVC A,@A+DPTR     ; Xuất cột
     MOV P1,A

     MOV A,R0
     MOV DPTR,#hang
     MOVC A,@A+DPTR
     MOV P0,A           ; Xuất hàng
     CALL delay         ; Tạo thời gian trì hoãn để thấy
     INC R0             ; Chuyển sang cột kế
     CJNE R0,#5,lap     ; Nếu quét đủ 5 cột thì lặp lại
```


Sưu tầm bởi: www.daihoc.com.vn

```
     SJMP main
;--------------------
delay:
     MOV TMOD,#01h
     MOV TL0,#LOW(-500)
     MOV TH0,#HIGH(-500)
     SETB TR0
     JNB TF0,$
     CLR TF0
     CLR TR0
     RET
;--------------------
cot: DB 01h,02h,04h,08h,10h
hang: DB 41h,3Eh,3Eh,3Eh,41h
END
```

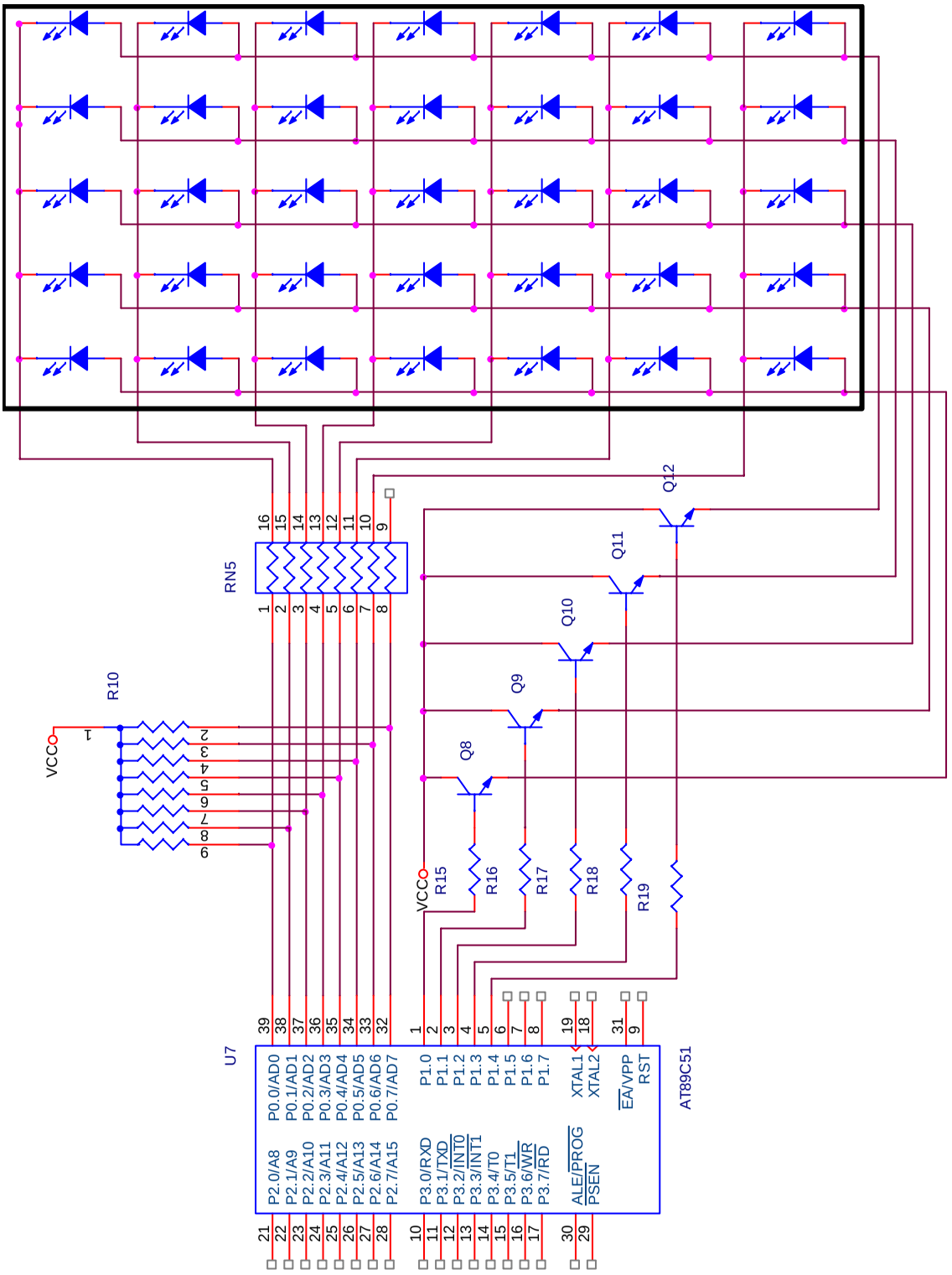
Ví dụ 2: Viết chương trình cho chuỗi 'KTCN' di chuyển từ trái sang phải trên ma trận Led.

Giải

Giải thuật để Led di chuyển từ trái sang phải tham khảo thêm tại ***Tài liệu Thí nghiệm Vi xử lý – Bài 3 (ma trận Led và bàn phím)*** (download tại Website http://eed.hutech.edu.vn).

```
main2:
     MOV R2,#0
main1:
     MOV R1,#20     ; Một ký tự quét 20 lần
main:
     MOV R0,#0
lap:
     MOV A,R0
     MOV DPTR,#cot
     MOVC A,@A+DPTR
     MOV P1,A

     MOV A,R0
     ADD A,R2
     MOV DPTR,#hang
     MOVC A,@A+DPTR
     MOV P0,A
```


Sưu tầm bởi: www.daihoc.com.vn



Hình 4.11 – Sơ đồ kết nối ma trận Led với AT89C51

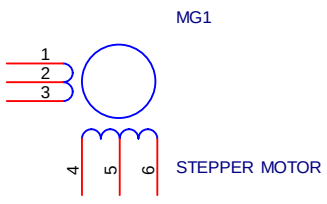
Sưu tầm bởi: www.daihoc.com.vn

```
      CALL delay
      INC R0
      CJNE R0,#5,lap

      DJNZ R1,main
      INC R2
      CJNE R2,#31,main1  ; Nếu quét hết chuỗi thì lặp lại
      SJMP main2
      ;-----------
delay:
      MOV TMOD,#01h
      MOV TL0,#LOW(-500)
      MOV TH0,#HIGH(-500)
      SETB TR0
      JNB TF0,$
      CLR TF0
      CLR TR0
      RET
cot: DB 01h,02h,04h,08h,10h
hang: DB 00h,77h,6Bh,5Dh,3Eh,7Fh ;Mã chữ K
      DB 7Eh,7Eh,00h,7Eh,7Eh,7Fh ;Mã chữ T
      DB 41h,3Eh,3Eh,3Eh,5Dh,7Fh ;Mã chữ C
      DB 00h,7Dh,7Bh,77h,00h,7Fh ;Mã chữ N
      DB 7Fh,7Fh,7Fh,7Fh,7Fh      ; Các cột trống
END
```

4. Điều khiển động cơ bước

Động cơ bước là động cơ cho phép dịch chuyển mỗi lần một bước hay nửa bước tuỳ theo xung điều khiển. Góc quay của mỗi bước tuỳ theo loại động cơ, thường là 1.8^0/bước hay 7.2^0/bước.

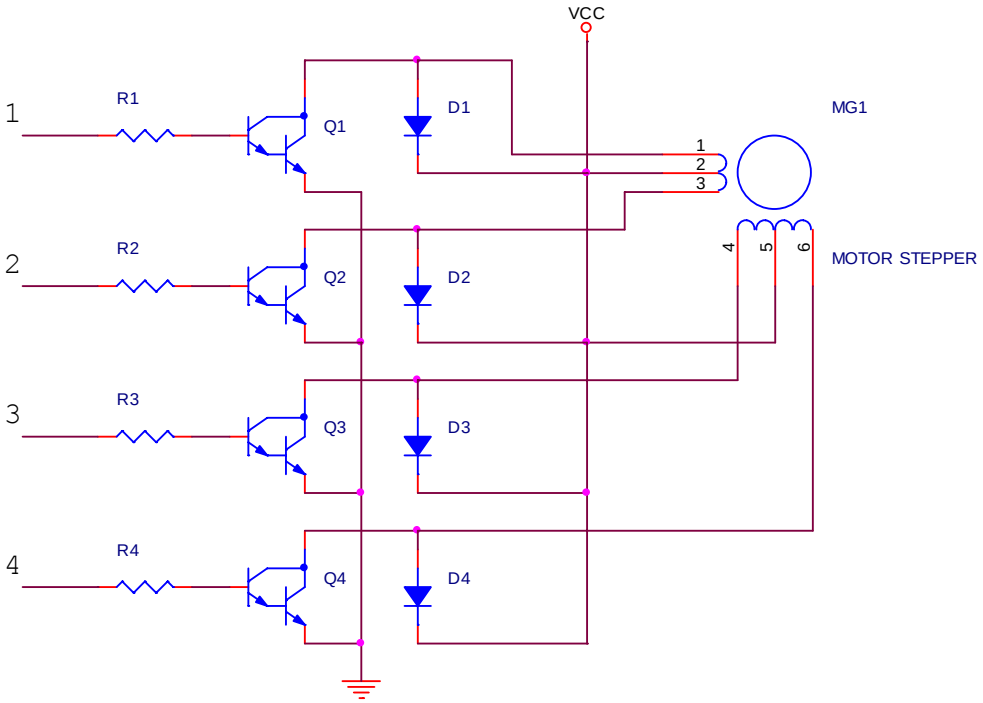
Động cơ bước gồm 4 cuộn dây: 1-2, 2-3, 4-5 và 5-6 như sơ đồ sau:



Hình 4.12 – Động cơ bước

Sưu tầm bởi: www.daihoc.com.vn

Mạch điều khiển động cơ như sau:



Hình 4.13 – Sơ đồ điều khiển động cơ bước

Xung điều khiển động cơ như sau:

Bảng 4.5 - Điều khiển một bước

Ngược				Thuận			
1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1	0	0	0	1	0	0	0
0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	0	0	1	0	1	0	0
1	0	0	0	1	0	0	0

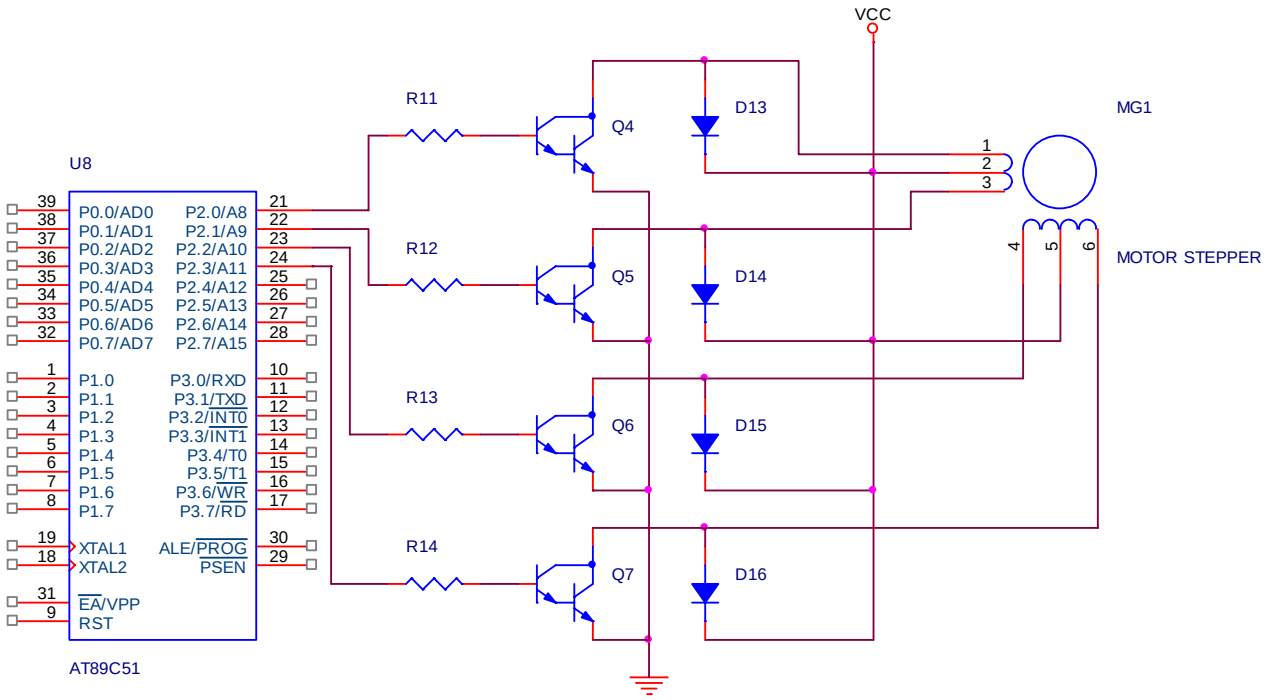
Bảng 4.6 - Điều khiển nửa bước

Ngược				Thuận			
1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1	0	0	1	1	0	0	1
1	0	0	0	0	0	0	1
1	1	0	0	0	0	1	1
0	1	0	0	0	0	1	0
0	1	1	0	0	1	1	0

Sưu tầm bởi: www.daihoc.com.vn

0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1

Ví dụ: Xét sơ đồ kết nối động cơ như hình 4.14. Viết chương trình điều khiển động cơ quay thuận mỗi lần một bước với tốc độ 50 vòng/phút (giả sử động cơ có góc quay là 7.2^0/bước).



Hình 4.14 – Sơ đồ kết nối AT89C51 với động cơ bước

Giải

Góc quay 7.2^0/bước → 1 vòng quay cần $360^0/7.2^0 = 50$ bước → 50 vòng quay cần thực hiện 2500 bước.

Tốc độ 50 vòng / phút → 1 phút (60s) thực hiện 2500 bước → mỗi bước cần 60/2500 = 0.024s = 24,,000 µs.

Thứ tự kích xung như bảng 4.5. Chương trình thực hiện như sau:

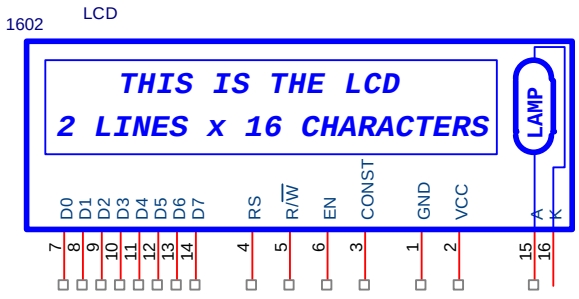
```
main:
     MOV R0,#0
     MOV DPTR,#thuan1buoc
```


Sưu tầm bởi: www.daihoc.com.vn

```
begin:
     MOV A,R0
     MOVC A,@A+DPTR
     MOV  P2,A          ; Xuất ra P2 để điều khiển động cơ
     CALL Delay
     INC R0
     CJNE R0,#4,begin
     SJMP main
     ;-----------------
Delay:
     MOV TMOD,#01h
     MOV TH0,#HIGH(-24000)
     MOV TL0,#LOW(-24000)
     SETB TR0
     JNB TF0,$
     CLR TF0
     CLR TR0
     RET

thuan1buoc: DB 08h,04h,02h,01h
END
```

5. Điều khiển LCD (Liquid Crystal Display)

❖ **Sơ đồ của LCD1602A:**



Hình 4.15 – LCD 1602A

- CONST (contrast): chỉnh độ tương phản (độ sáng của hình ảnh trên LCD).
- EN (Enable): cho phép đọc/ghi dữ liệu. Trong chế độ đọc, EN tác động bằng xung dương (cạnh lên) và trong chế độ ghi, EN tác động bằng xung âm (cạnh xuống).

Sưu tầm bởi: www.daihoc.com.vn

- RS (register selection): chọn thanh ghi lệnh (RS = 0) hoặc thanh ghi dữ liệu (RS = 1)
- R/W: đọc (R/W = 1) hay ghi (R/W = 0)
- D7 – D4: bus dữ liệu (chế độ 8 bit: 4 bit cao, chế độ 4 bit: dùng cho truyền 4 bit cao và 4 bit thấp). Ngoài ra, bit D7 còn dùng làm ngõ ra cho cờ Busy.
- D3 – D0: 4 bit thấp trong chế độ 8 bit hay bỏ trống trong chế độ 4 bit.
- A, K: anode và cathode đèn nền của LCD.

❖ **Các thành phần chức năng của LCD1602A:**

- **Cờ Busy (BF – Busy flag):** Nếu BF = 1, LCD đang trong quá trình thực thi một lệnh. Khi đó, các lệnh gởi tiếp theo sẽ bị bỏ qua. BF được đọc tại chân D7 khi RS = 0 và R/W = 1. Do đó, trước khi thực hiện một lệnh, cần kiểm tra BF trước, nếu BF = 0 thì mới gởi lệnh.
- **DDRAM (Display Data RAM):** chứa các ký tự sẽ hiển thị trên LCD, tối đa là 80x8 bit (80 ký tự). Khi hiển thị ở chế độ 1 dòng, địa chỉ của DDRAM có phạm vi từ 00h ÷ 4Fh còn khi ở chế độ 2 dòng, địa chỉ DDRAM từ 00h ÷ 27h cho dòng 1 và 40h ÷ 67h cho dòng 2.
- **Bộ đếm địa chỉ (AC - Address Counter):** dùng để lưu địa chỉ hiện hành của DDRAM và CGRAM, có thể thực hiện đọc AC khi RS = 0 và R/W = 1.
- **CGROM (Character Genaration ROM):** chứa các mô hình ký tự sẽ hiển thị trên LCD, bao gồm 192 ký tự 5x7 theo bảng mã ASCII (nghĩa là khi DDRAM chứa giá trị 41h tương ứng với mã ASCII của ký tự 'A' thì trên LCD sẽ hiện 'A'), trong đó chỉ có các mã từ 00h – 0Fh sẽ không lấy theo mã ASCII mà lấy theo các ký tự đã định nghĩa trong CGRAM.
- **CGRAM (Character Genaration RAM):** chứa các mô hình ký tự do người sử dụng định nghĩa để hiển thị các ký tự không có sẵn trong CGROM. CGRAM cho phép tạo tối đa 8 ký tự 5x8 (xem bảng 4.7).

Bảng 4.7 – Các ký tự định nghĩa trong CGRAM

DDRAM	Địa chỉ CGRAM	Dữ liệu CGRAM	Ký tự
00h hay 08h	000 000	xxx ?????	1
	000 001	xxx ?????	
	000 010	xxx ?????	
	000 011	xxx ?????	
	000 100	xxx ?????	
	000 101	xxx ?????	
	000 110	xxx ?????	
	000 111	xxx ?????	

Sưu tầm bởi: www.daihoc.com.vn

01h hay 09h	001 000 001 001 001 010 001 011 001 100 001 101 001 110 001 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	2
02h hay 0Ah	010 000 010 001 010 010 010 011 010 100 010 101 010 110 010 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	3
03h hay 0Bh	011 000 011 001 011 010 011 011 011 100 011 101 011 110 011 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	4
04h hay 0Ch	100 000 100 001 100 010 100 011 100 100 100 101 100 110 100 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	5
05h hay 0Dh	101 000 101 001 101 010 101 011 101 100 101 101 101 110 101 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	6

Sưu tầm bởi: www.daihoc.com.vn

06h hay 0Eh	110 000 110 001 110 010 110 011 110 100 110 101 110 110 110 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	7
07h hay 0Fh	111 000 111 001 111 010 111 011 111 100 111 101 111 110 111 111	xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ????? xxx ?????	8

Để định nghĩa một ký tự, thực hiện thay thế dấu ? bằng các giá trị 0 hay 1 tương ứng và gởi vào CGRAM.

Ví dụ: Để định nghĩa chữ Đ tại vị trí 1 trong CGRAM, địa chỉ và dữ liệu tương ứng là:

Địa chỉ	**Dữ liệu**					
00h	1	1	1	1	0	1Eh
01h	0	1	0	0	1	09h
02h	0	1	0	0	1	09h
03h	1	1	1	0	1	1Dh
04h	0	1	0	0	1	09h
05h	0	1	0	0	1	09h
06h	1	1	1	1	0	1Eh
07h	0	0	0	0	0	00h

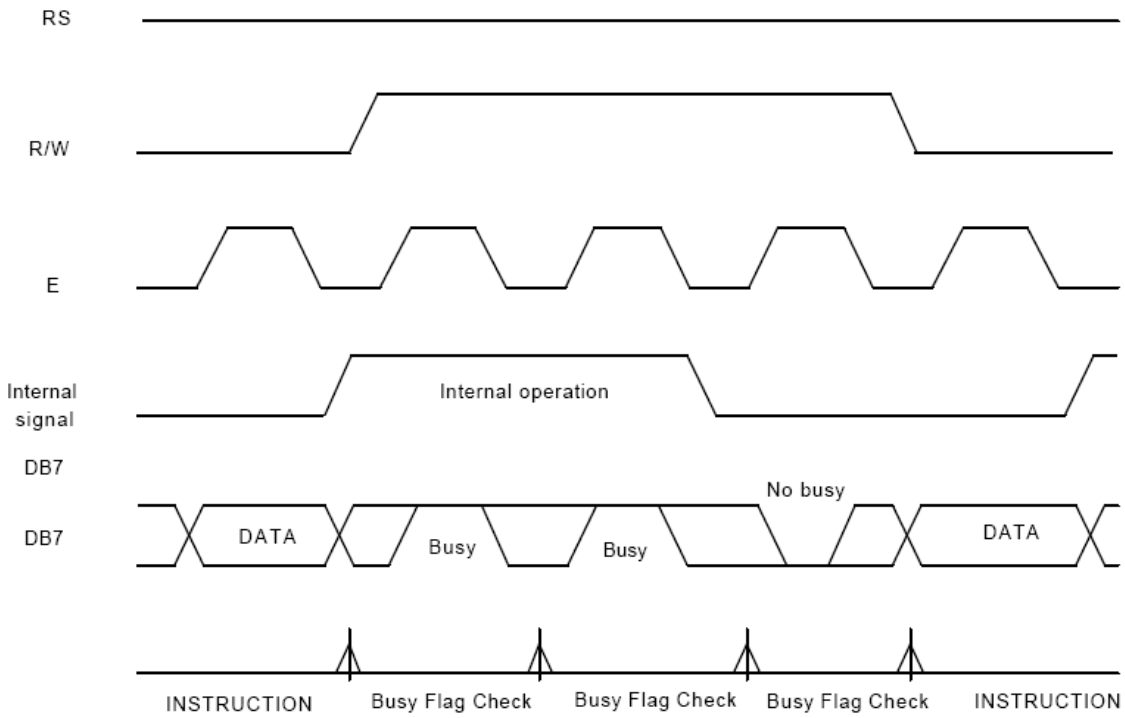
Nghĩa là tại địa chỉ 00h của CGRAM chứa giá trị là 1Eh và tương tự cho đến địa chỉ 07h.

❖ **Các chế độ truyền dữ liệu:**

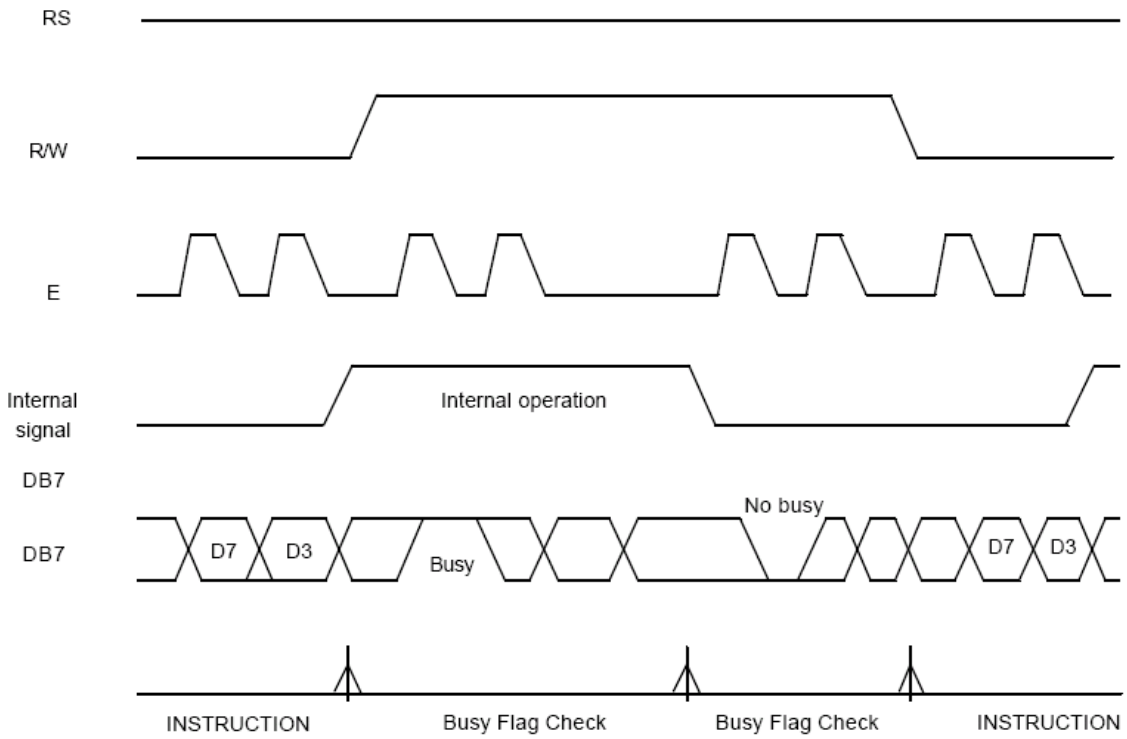
LCD1602A có 2 chế độ truyền dữ liệu: chế độ 8 bit (dùng cả D0 – D7) và chế độ 4 bit (không dùng D3 – D0, chỉ dùng D7 – D4). Trong trường hợp dùng chế độ 4 bit, dữ liệu 8 bit sẽ được truyền 2 lần: truyền 4 bit cao rồi tiếp tục truyền 4 bit thấp.

Sưu tầm bởi: www.daihoc.com.vn

Sau khi thực hiện truyền xong 8 bit, BF mới chuyển lên 1. Hai chế độ truyền này mô tả như hình 4.16 và 4.17.



Hình 4.16 – Định thời giao tiếp ở chế độ 8 bit



Hình 4.17 – Định thời giao tiếp ở chế độ 4 bit

Sưu tầm bởi: www.daihoc.com.vn

❖ Tập lệnh:

Bảng 4.8 - Tập lệnh của LCD1602A

Lệnh	Mã lệnh										Mô tả	Thời gian thực thi (f_{OSC} = 270 KHz)
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Xoá màn hình	0	0	0	0	0	0	0	0	0	1	Ghi 20h (khoảng trắng) vào DDRAM và đặt địa chỉ DDRAM là 00h	1.53ms
Về đầu chuỗi	0	0	0	0	0	0	0	0	1	X	Đặt địa chỉ DDRAM là 00h và trả con trỏ về vị trí đầu (nội dung DDRAM không đổi)	1.53ms
Định chế độ	0	0	0	0	0	0	0	1	I/D	S	- Chiều di chuyển con trỏ I/D = 1: tăng I/D = 0: giảm - Dịch toàn màn hình khi ghi DDRAM: S = 1: cho phép dịch S = 0: cấm	39µs
Điều khiển hiển thị	0	0	0	0	0	0	1	D	C	B	D = 1: hiện màn hình D = 0: cấm C = 1: hiện con trỏ C = 0: cấm B = 1: nhấp nháy B = 0: cấm	39µs
Dịch con trỏ hay màn hình	0	0	0	0	0	1	S/C	R/L	X	X	S/C = 1: dịch màn hình S/C = 0: dịch con trỏ R/L = 1: dịch phải R/L = 0: dịch trái	39µs
Chức năng	0	0	0	0	1	DL	N	F	X	X	DL = 1: 8 bit DL = 0: 4 bit N = 1: 2 dòng N = 0: 1 dòng	39µs

Sưu tầm bởi: www.daihoc.com.vn

											F = 0: ký tự 5x7 F = 1: ký tự 5x10	
Định địa chỉ CGRAM	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Xác định địa chỉ của CGRAM	39µs
Định địa chỉ DDRAM	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Xác định địa chỉ của DDRAM	39µs
Đọc BF và địa chỉ hiện hành	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Xác định địa chỉ hiện hành và kiểm tra xem có thể gởi lệnh tiếp hay không BF = 1: không BF = 0: có thể	0
Ghi dữ liệu	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Ghi dữ liệu vào DDRAM hay CGRAM	43µs
Đọc dữ liệu	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Đọc dữ liệu từ DDRAM hay CGRAM	43µs

I/D: Increment/Decrement S: Screen S/C: Screen/Cursor
R/L: Right/Left DL: Data Length N: Line number
F: Font type AC: Address Counter

Các giá trị thường dùng mô tả như sau:

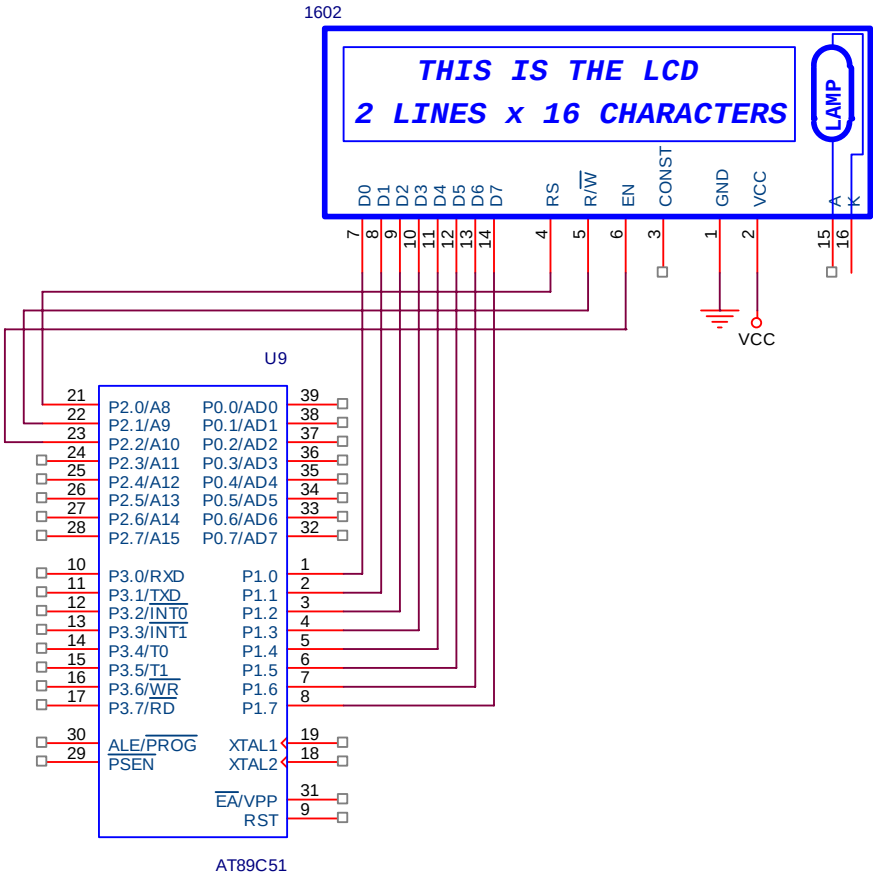
Bảng 4.9 – Các lệnh thường dùng

Lệnh	Mô tả
01H	Xóa màn hình màn hình
02H	Trở về đầu chuỗi
04H	Dịch con trỏ sang trái
06H	Dịch con trỏ sang phải
05H	Dịch màn hình sang phải
07H	Dịch màn hình sang trái
08H	Tắt con trỏ, tắt hiển thị
0AH	Tắt hiển thị, bật con trỏ
0CH	Bật hiển thị, tắt con trỏ
0EH	Bật hiển thị, nhấp nháy con trỏ
0FH	Tắt hiển thị, nhấp nháy con trỏ
10H	Dịch vị trí con trỏ sang trái
14H	Dịch vị trí con trỏ sang phải
18H	Dịch toàn bộ màn hình sang trái

Sưu tầm bởi: www.daihoc.com.vn

1CH	Dịch toàn bộ màn hình sang phải
80H	Đưa con trỏ về đầu dòng 1
C0H	Đưa con trỏ về đầu dòng 2
38H	Xác lập chế độ 2 dòng và độ phân giải chữ 5x7

Ví dụ 1: Cho sơ đồ kết nối LCD 1602A với AT89C51 như hình vẽ. Viết chương trình hiển thị chuỗi "KHOA DIEN – DIEN TU" trên dòng 1 và "BO MON DIEN TU – VIEN THONG" trên dòng 2.



Hình 4.18 – Kết nối LCD và 89C51

Giải

8 bit dữ liệu của LCD nối với P1 → chế độ 8 bit. Yêu cầu hiện trên 2 dòng → chế độ 2 dòng.

Chương trình thực hiện như sau:

```
EN          BIT         P2.2
RS          BIT         P2.0
```


Sưu tầm bởi: www.daihoc.com.vn

```
     RW        BIT       P2.1
     LCD_DATA      EQU       P1
     ;-------------------------------------------------
main:
     MOV LCD_DATA,#38h      ; đặt chế độ 2 dòng
     CALL write_command

     MOV LCD_DATA,#0Ch  ; bật hiển thị
     CALL write_command

     MOV LCD_DATA,#01h      ;xoá màn hình
     CALL write_command

     MOV LCD_DATA,#80h      ; Chuyển về địa chỉ 00h (dòng 1)
     CALL write_command
     MOV DPTR,#Line1
     CALL write         ; Ghi vao DDRAM

     MOV LCD_DATA,#0C0h ; Chuyển về địa chỉ 40h (dòng 2)
     CALL write_command
     MOV DPTR,#Line2
     CALL write         ; Ghi vao DDRAM

     SJMP $
;-------------------------------------------------------
write:
     CLR A
     MOVC A,@A+DPTR
     CJNE A,#0FFh,write1;Nếu giá trị là 0FFh thì hết chuỗi
     RET
write1:
     MOV LCD_DATA,A
     call write_data
     INC DPTR
     SJMP write
;-------------------------------------------------
write_command:
     CLR RS
     CLR RW
     CLR EN
     NOP
```


Sưu tầm bởi: www.daihoc.com.vn

```
      SETB EN
      NOP
      CLR EN
      CALL Delay
      RET
;-------------------------------------------------
write_data:
      SETB RS
      CLR RW
      CLR EN
      NOP
      SETB EN
      NOP
      CLR EN
      CALL Delay
      RET
;-------------------------------------------------
Delay:
      PUSH 07h
      PUSH 06h
      MOV R6,#50
      MOV R7,#255
      DJNZ R7,$
      DJNZ R6,$-4
      POP 06h
      POP 07h
      RET
      ;-------------------------------------------------
      Line1: DB 'KHOA DIEN – DIEN TU', 0FFh
      Line2: DB 'BO MON DIEN TU – VIEN THONG', 0FFH
      END
```

Ví dụ 2: Yêu cầu giống như ví dụ 1 nhưng cứ mỗi 1s thì dịch chuỗi sang trái một ký tự.

Giải

Chương trình thực hiện như trên nhưng thêm phần xử lý ngắt cho timer 0: cứ định thời 1s thì dịch chuỗi sang trái (nghĩa là dịch toàn bộ màn hình sang phải). Theo bảng 4.9, lệnh cần gởi ra LCD có mã lệnh là 1Ch.

Chương trình thực hiện như sau:

Sưu tầm bởi: www.daihoc.com.vn

```
      …
      ORG 0000h
      LJMP main
      ORG 000Bh
      LJMP Timer0_ISR
Main:
      MOV IE,#82h   ; Cho phép ngắt tại Timer 0
      MOV TMOD,#01h
      MOV TH0,#HIGH(-50000)
      MOV TL0,#LOW(-50000)
      MOV R7,#20
      SETB TR0
      …
Timer0_ISR:
      MOV TH0,#HIGH(-50000)
      MOV TL0,#LOW(-50000)
      DJNZ R7,exitTimer0
      MOV R7,#20
      MOV LCD_DATA,#1Ch  ;Dịch toàn màn hình sang phải
      CALL write_command
exitTimer0:
      RETI
      …
      END
```

Ví dụ 3: Cho mạch kết nối LCD như hình 4.18, viết chương trình xuất chuỗi "Khoa Điện – Điện tử" trên dòng 1 và "Bộ môn Điện tử - Viễn thông" trên dòng 2.

Giải

Ví dụ này yêu cầu các ký tự không có trong bảng mã nên phải định nghĩa thêm trong CGRAM. Các ký tự cần định nghĩa là: Đ, ệ, ử, ộ, ô, ễ, tổng cộng là 6 ký tự (có thể thực hiện được do LCD 1602A cho phép định nghĩa tối đa 8 ký tự).

Sưu tầm bởi: www.daihoc.com.vn

Địa chỉ và dữ liệu tương ứng là:

Địa chỉ	Dữ liệu						Ký tự	Mã DDRAM
00h	1	1	1	1	0	1Eh	Đ	00h
01h	0	1	0	0	1	09h		
02h	0	1	0	0	1	09h		
03h	1	1	1	0	1	1Dh		
04h	0	1	0	0	1	09h		
05h	0	1	0	0	1	09h		
06h	1	1	1	1	0	1Eh		
07h	0	0	0	0	0	00h		
08h	0	0	1	0	0	04h	ệ	01h
09h	0	1	1	1	0	0Eh		
0Ah	1	0	0	0	1	11h		
0Bh	1	1	1	1	0	1Eh		
0Ch	1	0	0	0	0	10h		
0Dh	0	1	1	1	1	0Fh		
0Eh	0	0	1	0	0	04h		
0Fh	0	0	1	0	0	00h		
10h	0	1	0	0	0	08h	ử	02h
11h	0	0	1	0	1	05h		
12h	0	1	0	0	1	09h		
13h	1	0	0	1	0	12h		
14h	1	0	0	1	0	12h		
15h	1	0	0	1	0	12h		
16h	0	1	1	0	0	0Ch		
17h	0	0	0	0	0	00h		
18h	0	0	1	0	0	04h	ộ	03h
19h	0	1	0	1	0	0Ah		
1Ah	0	1	1	1	0	0Eh		
1Bh	1	0	0	0	1	11h		
1Ch	1	0	0	0	1	11h		
1Dh	0	1	1	1	0	0Eh		
1Eh	0	0	1	0	0	04h		
1Fh	0	0	0	0	0	00h		

Sưu tầm bởi: www.daihoc.com.vn

Địa chỉ	Dữ liệu						Ký tự	Mã DDRAM
20h	0	0	1	0	0	04h	ô	04h
21h	0	1	0	1	0	0Ah		
22h	0	1	1	1	0	0Eh		
23h	1	0	0	0	1	11h		
24h	1	0	0	0	1	11h		
25h	0	1	1	1	0	0Eh		
26h	0	0	0	0	0	00h		
27h	0	0	0	0	0	00h		
28h			1		1	05h	ễ	05h
29h		1	1	1		0Eh		
2Ah		1		1		0Ah		
2Bh	1	1	1	1	1	1Fh		
2Ch	1	1	1	1	1	1Fh		
2Dh	1					10h		
2Eh		1	1	1	1	0Fh		
2Fh	0	0	0	0	0	00h		

Chương trình thực hiện như sau:

```
EN          BIT         P2.2
RS          BIT         P2.0
RW          BIT         P2.1
LCD_DATA        EQU         P1
;-------------------------------------------------
org 0
ljmp main
main:
MOV LCD_DATA,#38h
CALL write_command

MOV LCD_DATA,#0Ch
CALL write_command

MOV LCD_DATA,#01h       ;xoá màn hình
CALL write_command

MOV LCD_DATA,#40h       ; Địa chỉ đầu của CGRAM
call write_command      ; là 00h
MOV DPTR,#cgram_data
CALL write
```


Sưu tầm bởi: www.daihoc.com.vn

```
     MOV LCD_DATA,#80h
     CALL write_command
     MOV DPTR,#Line1
     CALL write

     MOV LCD_DATA,#0C0h
     CALL write_command
     MOV DPTR,#Line2
     CALL write

     here:SJMP here
;-------------------------------------------------------
write:
     CLR A
     MOVC A,@A+DPTR
     CJNE A,#0FFh,write1
     RET
write1:
     MOV LCD_DATA,A
     call write_data
     INC DPTR
     SJMP write
;-------------------------------------------------------
Delay:
     PUSH 07h
     PUSH 06h
     MOV R6,#50
     MOV R7,#255
     DJNZ R7,$
     DJNZ R6,$-4
     POP 06h
     POP 07h
     RET
     ;-------------------------------------------------
write_command:
     CLR RS
     CLR RW
     CLR EN
     NOP
     SETB EN
     NOP
```


Sưu tầm bởi: www.daihoc.com.vn

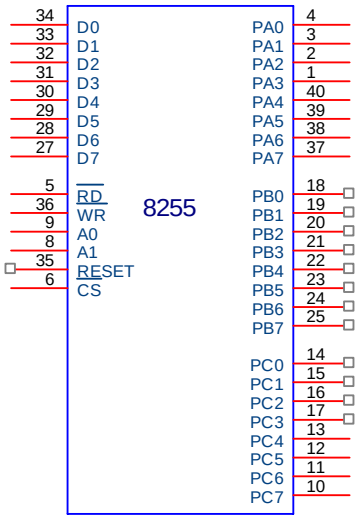
```
     CLR EN
     SJMP wait
;-------------------------------------------------
write_data:
     SETB RS
     CLR RW
     CLR EN
     NOP
     SETB EN
     NOP
     CLR EN
wait:
     call delay
     ret

;-------------------------------------------------
Line1:  DB  'Khoa  ',00h,'i',01h,'n  -  ',00h,'i',01h,'n
t',02h,0FFh ; Chuỗi 'Khoa Điện – Điện tử'
Line2: DB 'B',03h,' m',04h,'n ',00h,'i',01h,'n t',02h,' -
vi',05h,'n th',04h,'ng', 0FFH ; Chuỗi 'Bộ môn Điện tử -
Viễn thông'
;-------------------------------------------------
cgram_data:   DB 1Eh,09h,09h,1Dh,09h,09h,1Eh,00h ; Chữ Đ
              DB 04h,0Eh,11h,1Eh,10h,0Fh,04h,00h ; Chữ ệ
              DB 08h,05h,09h,12h,12h,12h,0Ch,00h ; Chữ ử
              DB 04h,0Ah,0Eh,11h,11h,0Eh,04h,00h ; Chữ ộ
              DB 04h,0Ah,0Eh,11h,11h,0Eh,00h,00h ; Chữ ô
              DB 05h,0Eh,0Ah,1Fh,1Fh,10h,0Fh,00h ; Chữ ễ
              DB 0FFh
     END
```

6. Giao tiếp với PPI8255

PPI8255 là IC giao tiếp lập trình được, cho phép mở rộng port trong trường hợp các port của 89C51 không đủ dùng. Các chế độ hoạt động của 8255 có thể tham khảo thêm tại **Giáo trình Vi xử lý** (cùng tác giả). 8255 có tổng cộng 2 chế độ: BSR (Bit Set/Reset) và I/O (Input/Output) trong đó I/O chia thành 3 chế độ khác nhau, tron tài liệu này chỉ xét ở chế độ 0 (xuất/nhập cơ bản).

8255 có tổng cộng 3 port, mỗi port 8 bit trong đó port C có thể chia thành 4 bit cao và 4 bit thấp tạo thành 2 nhóm: nhóm A (PA + PCH) và nhóm B (PB và PCL).

Sưu tầm bởi: www.daihoc.com.vn



D7 – D0: bus dữ liệu

PA7 – PA0: Port A

PB7 – PB0: Port B

PC7 – PC0: Port C

A1, A0: giải mã

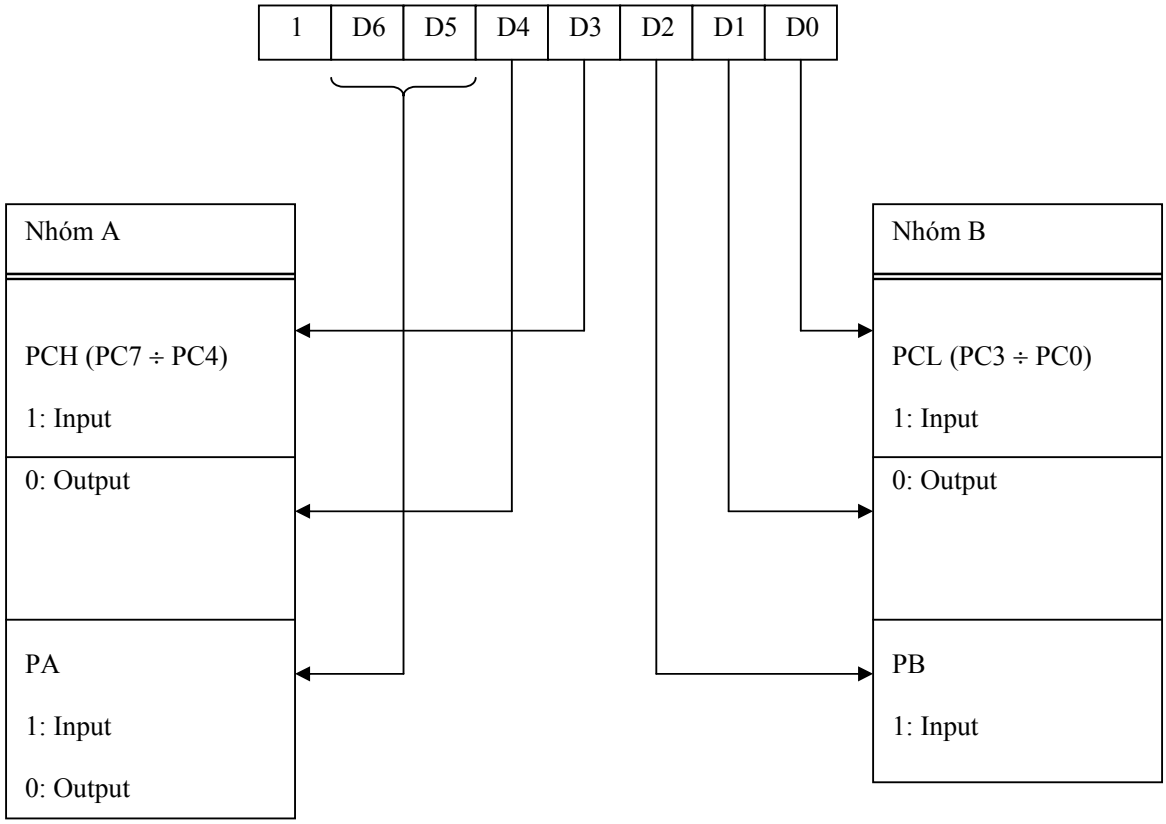
RESET: ngõ vào Reset

$\overline{CS}$: Chip Select

$\overline{RD}$: Read

Hình 4.19 – Sơ đồ chân của 8255

Để điều khiển 8255, bên trong có một thanh ghi điều khiển (CR – Control Register) cho phép chọn chế độ hoạt động. Nội dung của CR như sau:



Hình 4.20 – Dạng từ điều khiển cho 8255A ở chế độ I/O

Sưu tầm bởi: www.daihoc.com.vn

D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	X				S/R
Mode BSR	Không sử dụng			Chọn bit 000: PC0 001: PC1 010: PC2 011: PC3 100: PC4 101: PC5 110: PC6 111: PC7			0: Xoá (Reset) 1: Đặt (Set)

Hình 4.21 - Dạng từ điều khiển cho 8255A ở chế độ BSR

Lưu ý rằng khi cần Set/Reset bit thì phải gởi dữ liệu ra CR chứ không gởi ra PC.

Như vậy, để xác lập điều khiện làm việc cho 8255, cần thực hiện định cấu hình cho 8255 (chọn các chế độ hoạt động cho PA, PB và PC). Để thực hiện quá trình này, cần tác động đến CR của 8255. Logic chọn các port cho 8255 mô tả như sau:

Bảng 4.10 – Logic chọn các port của 8255

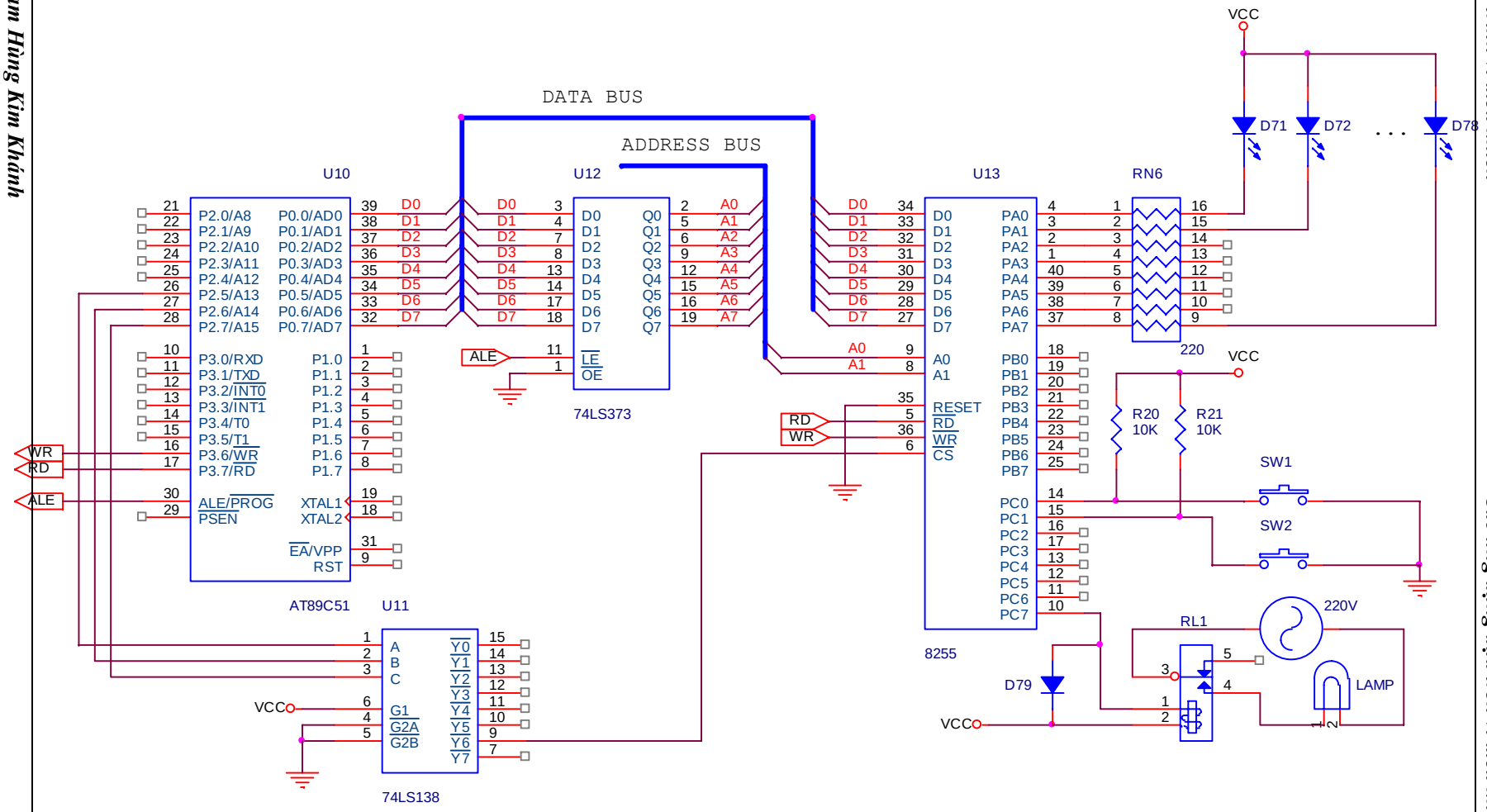
$\overline{CS}$	**A1**	**A0**	**Chọn**
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Thanh ghi điều khiển
1	x	x	8255A không hoạt động

Ví dụ: Cho mạch kết nối giữa AT89C51 và 8255 như hình 4.22. Viết chương trình điều khiển theo yêu cầu:

- Nhấn SW1: sáng 4 Led trái và sáng Lamp.
- Nhấn SW2: sáng 4 Led phải và tắt Lamp.

Giải

Sưu tầm bởi: www.daihoc.com.vn



Hình 4.22 – Sơ đồ kết nối 89C51 với 8255

Sưu tầm bởi: www.daihoc.com.vn

Do PA điều khiển Led, PCL điều khiển công tắc nhấn, PCH điều khiển RL1 nên PA xuất, PCL nhập và PCH xuất (còn PB tuỳ ý). Nội dung thanh ghi điều khiển như sau:

1	0	0	0	0	0	0	1	81h
I/O	Chế độ 0		PA xuất	PCH xuất	Chế độ 0	PB xuất	PCL nhập	

Led đơn nối với các bit của PA tại cathode và anode nối với Vcc nên để Led sáng thì dữ liệu tại PA là 0 và Led tắt khi dữ liệu là 1.

Đèn LAMP được điều khiển bằng RL1: khi RL1 đóng (ứng với PC7 = 0) thì LAMP sáng và ngược lại, khi RL1 ngắt (ứng với PC7 = 1) thì LAMP tắt. Nội dung thanh ghi điều khiển khi điều khiển PC7 như sau:

PC7 = 0

0	0	0	0	1	1	1	0	0Eh
BSR	Không dùng			PC7			= 0	

PC7 = 1

0	0	0	0	1	1	1	1	0Fh
BSR	Không dùng			PC7			= 1	

Công tắc SW1, SW2 nối với PC0 và PC1: khi nhấn công tắc thì chân tương ứng tại PC = 0 và khi không nhấn thì = 1. Do đó, để kiểm tra công tắc có nhấn hay không thì đọc dữ liệu từ PCL và kiểm tra tương ứng các bit PC0, PC1.

Địa chỉ các port của 8255:

$\overline{CS}$ = 0 (Y6 = 0)			Tuỳ ý											A1	A0	Port	Địa chỉ hex
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
1	1	0	X	X	X	X	X	X	X	X	X	X	X	0	0	A	C000h
														0	1	B	C001h
														1	0	C	C002h
														1	1	CR	C003h

Chương trình thực hiện như sau:

```
     MOV DPTR,#0C003h   ; Địa chỉ CR
     MOV A,#81h; PA: xuất, PB: xuất, PCH: xuất, PCL: nhập
     MOVX @DPTR,A       ; Xuất ra CR
Begin:
     MOV DPTR,#0C002h   ; Địa chỉ PC
```


Sưu tầm bởi: www.daihoc.com.vn

```
     MOVX A,@DPTR       ; Đọc vào
     JNB ACC.0,SW1      ; Nếu PC0 = 0 thì đến SW1
     JNB ACC.1,SW2      ; Nếu PC1 = 0 thì đến SW2
     SJMP begin
SW1:
     CALL Delay         ; Tránh rung phím
     MOV A,11110000b    ; Sáng 4 Led trái
     MOV DPTR,#0C000h   ; Địa chỉ PA (do PA nối với Led)
     MOVX @DPTR,A

     MOV A,0Eh          ; PC7 = 0 → đóng RL1 → sáng LAMP
     MOV DPTR,#0C003h   ; Địa chỉ CR (do dùng chế độ BSR)
     MOVX @DPTR,A
     SJMP begin
SW2:
     CALL Delay
     MOV A,00001111b    ; Sáng 4 Led phải
     MOV DPTR,#0C000h   ; Địa chỉ PA (do PA nối với Led)
     MOVX @DPTR,A

     MOV A,0Fh          ; PC7 = 1 → đóng RL1 → sáng LAMP
     MOV DPTR,#0C003h   ; Địa chỉ CR (do dùng chế độ BSR)
     MOVX @DPTR,A
     SJMP begin
;----------------------------
Delay:
     MOV TMOD,#02h
     MOV TH0,#HIGH(-50000)
     MOV TL0,#LOW(-50000)
     SETB TR0
     JNB TF0,$
     CLR TF0
     CLR TR0
     RET
     END
```


Sưu tầm bởi: www.daihoc.com.vn

BÀI TẬP CHƯƠNG 4

1. Cho sơ đồ kết nối như hình 4.3. Viết chương trình sáng Led theo yêu cầu: sáng lần lượt 1 Led từ phải sang trái và thực hiện 4 lần; nhấp nháy 8 Led 5 lần; sáng Led từ ngoài vào trong, mỗi lần 2 Led và thực hiện 3 lần (thời gian trì hoãn giữa 2 lần sáng là 300ms, dùng timer 1).

2. Cho sơ đồ kết nối như hình 4.7. Viết chương trình tăng nội dung của ô nhớ 30h từ 00 – 99 và hiển thị giá trị trên 2 Led 7 đoạn (hiển thị Led bằng ngắt timer 1 và thời gian trì hoãn khi tăng nội dung của ô nhớ 30h là 1s dùng ngắt timer 0).

3. Cho sơ đồ kết nối như hình 4.7 trong đó kết nối thêm 4 Led (Led2 – 6) được điều khiển bằng các bit của P1: P1.2 – P1.5. Viết chương trình hiển thị giờ, phút giây trên 6 Led (Led1,2: giờ; Led3,4: phút; Led5,6: giây) trong đó giờ chứa trong ô nhớ 30h, phút trong ô nhớ 31h, giây trong ô nhớ 32h (thời gian trì hoãn 1s dùng ngắt timer 0, quét Led dùng ngắt timer 1).

4. Cho sơ đồ kết nối như hình 4.11. Viết chương trình cho chuỗi "DAI HOC KY THUAT CONG NGHE TPHCM" di chuyển từ trái sang phải trên ma trận Led.

5. Cho sơ đồ kết nối như hình 4.14. Viết chương trình điều khiển động cơ quay thuận 100 vòng với tốc độ 10 vòng/phút (giả sử mỗi bước có góc quay là 7.2^0).

6. Cho sơ đồ kết nối như hình 4.18. Viết chương trình cho chuỗi "Đại học Kỹ thuật Công nghệ" trên dòng 1 và "Khoa Điện – Điện tử" trên dòng 2 di chuyển từ trái sang phải (thời gian dịch chuyển là 300ms dùng ngắt timer 1).

Sưu tầm bởi: www.daihoc.com.vn

Phụ lục 2: MÔ PHỎNG BẰNG PROTEUS

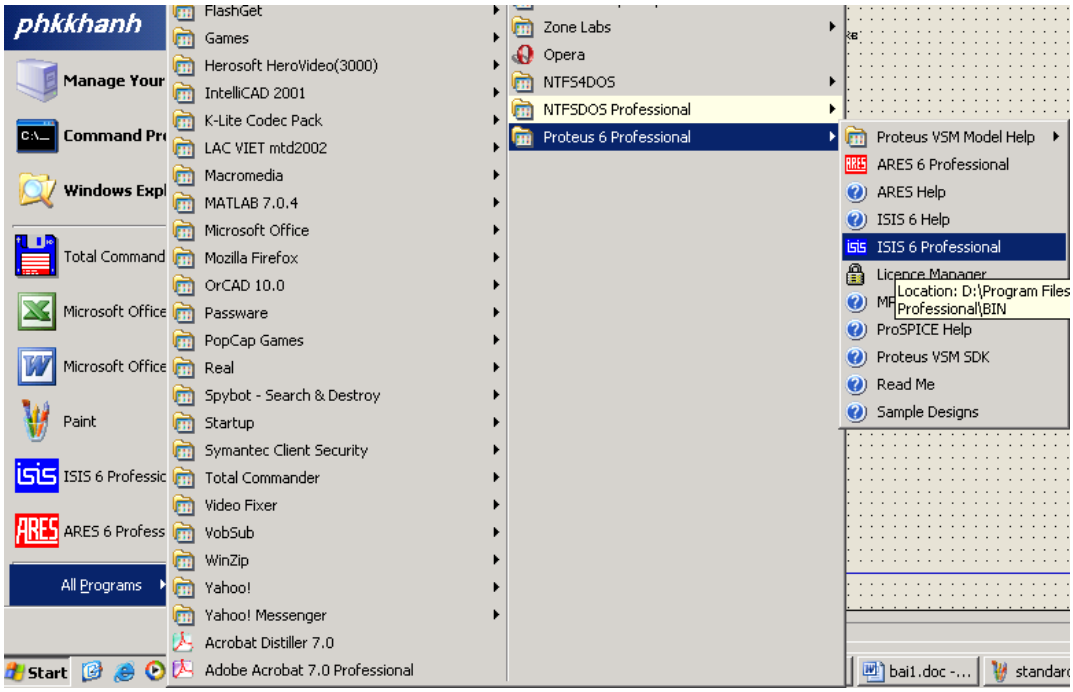
1. Giới thiệu

Phần mềm Proteus là phần mềm cho phép mô phỏng hoạt động của mạch điện tử bao gồm phần thiết kế mạch và viết chương trình điều khiển cho các họ vi điều khiển như MCS-51, PIC, AVR, …

Phần mềm bao gồm 2 chương trình: ISIS cho phép mô phỏng mạch và ARES dùng để vẽ mạch in.

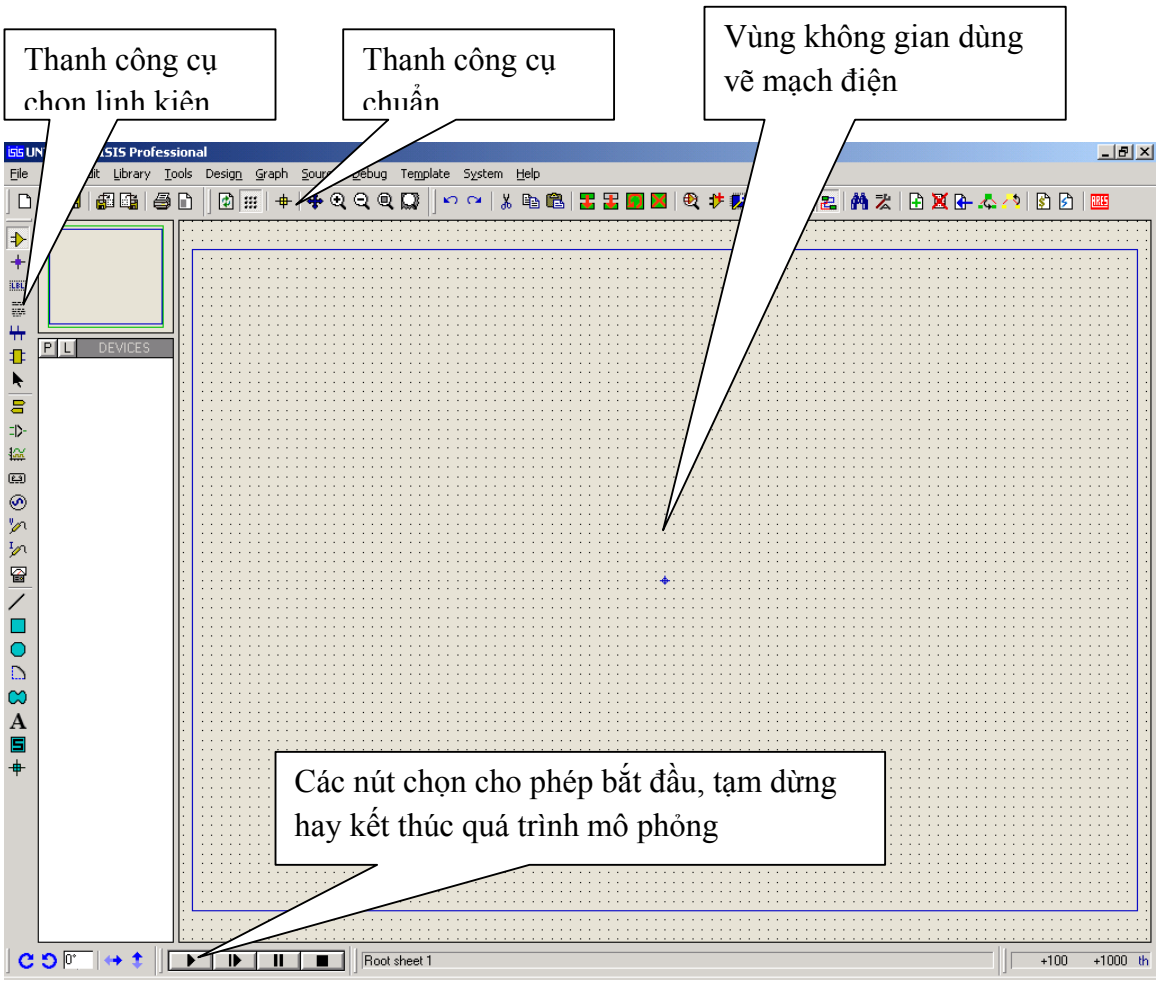
Khởi động chương trình

- Start > All Program > Proteus 6 Professional > ISIS 6 Professional



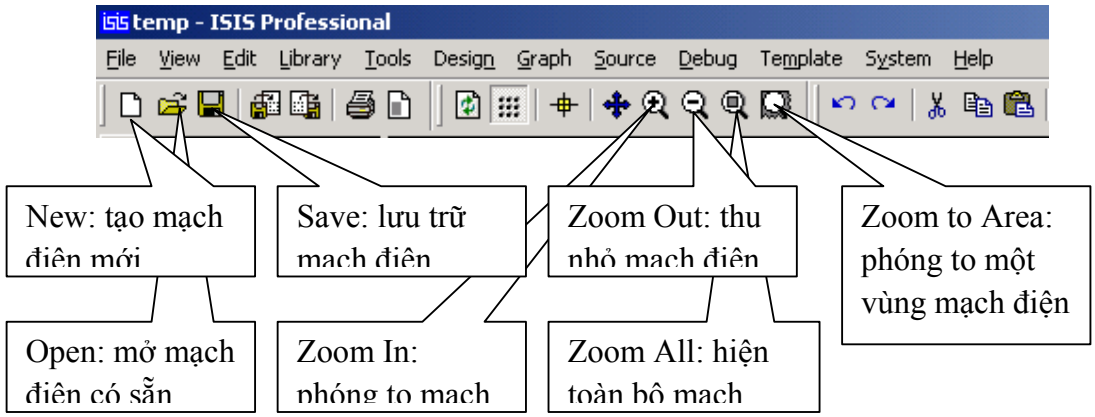
Sưu tầm bởi: www.daihoc.com.vn

- Cửa sổ chương trình sau khi khởi động:



Các thao tác cơ bản

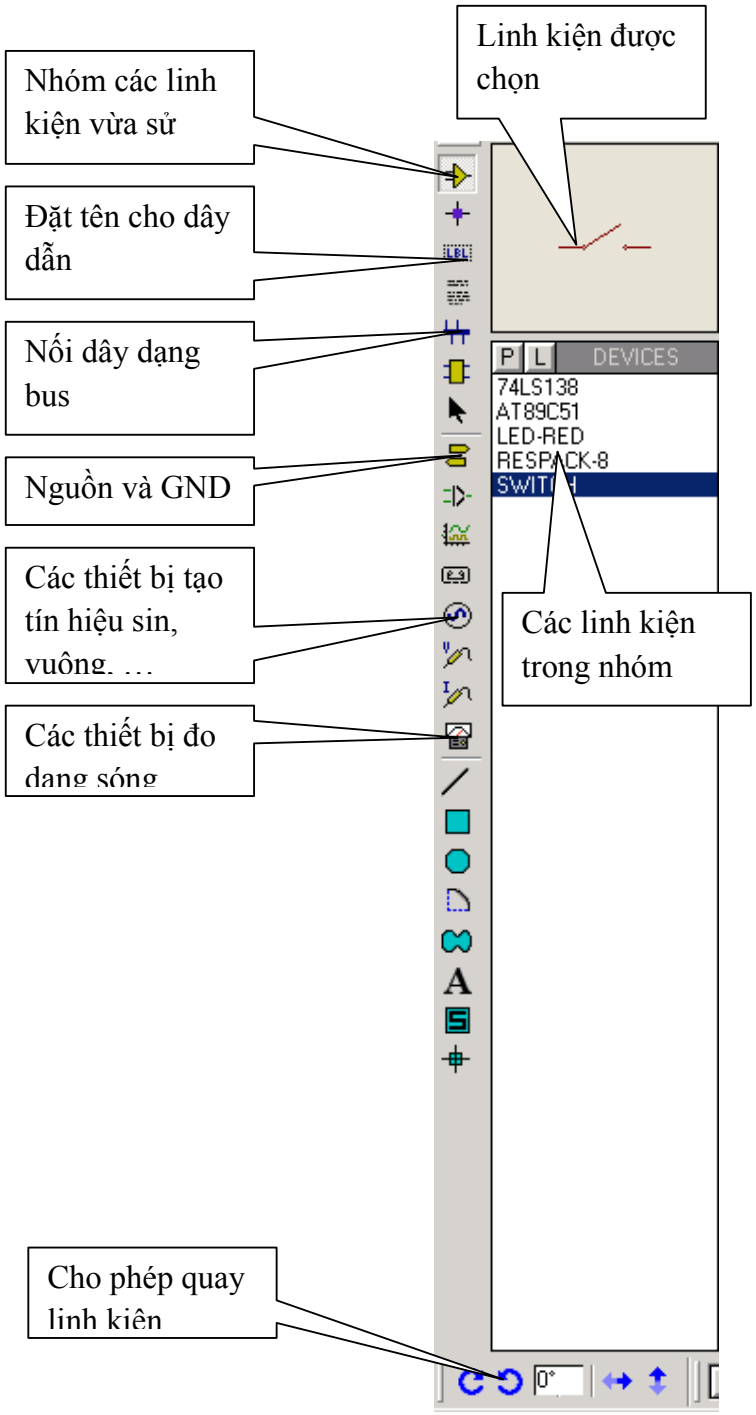
❖ **Sử dụng thanh công cụ chuẩn:**



Các thao tác trên thanh công cụ chuẩn cũng có thể thực hiện thông qua menu File và menu Edit.

Sưu tầm bởi: www.daihoc.com.vn

❖ **Sử dụng thanh linh kiện:**



Để đưa linh kiện vào vùng thiết kế, ta thực hiên chọn linh kiện rồi nhấn chuột trái trên vùng làm việc.

Để thực hiện chọn linh kiện, ta thực hiện nhấn chuột phải trên linh kiện, nó sẽ chuyển sang màu đỏ cho biết trạng thái đang chọn.

Sưu tầm bởi: www.daihoc.com.vn

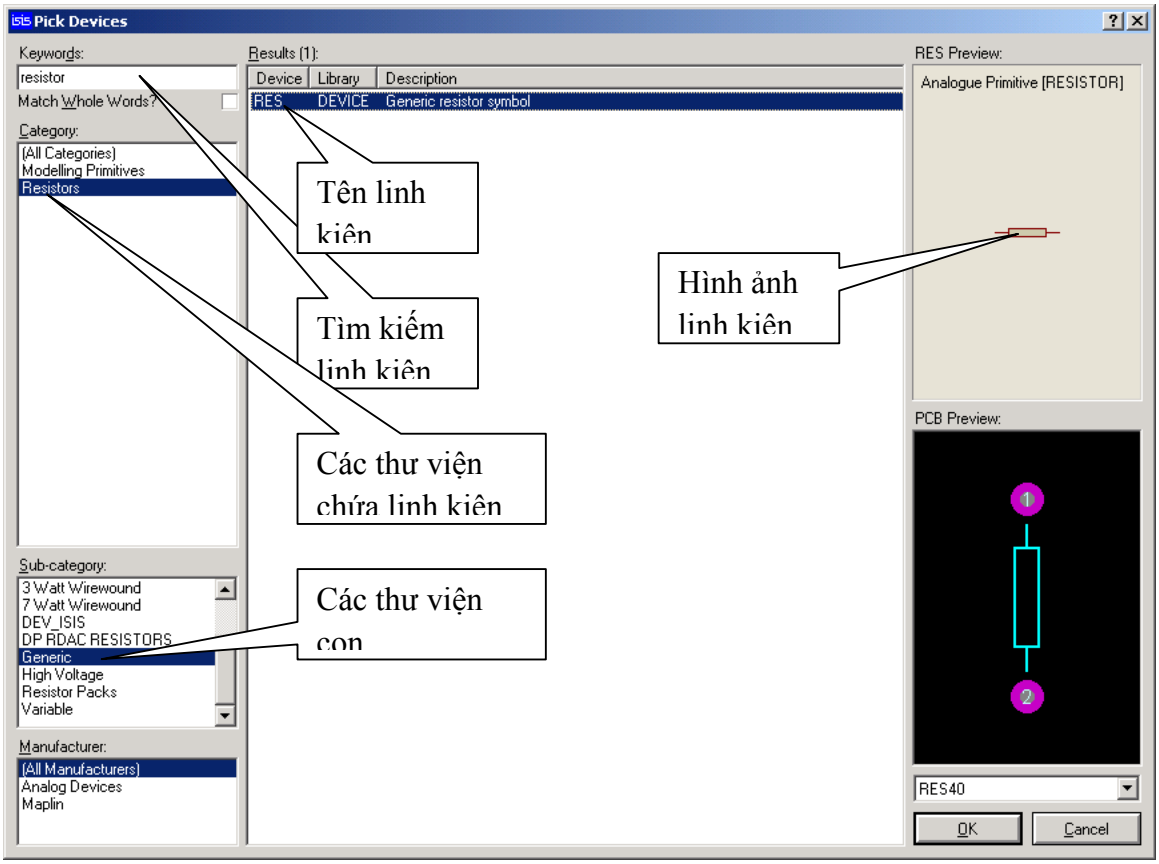
Sau khi đã chọn linh kiện, ta có thể di chuyển linh kiện bằng cách thực hiện thao tác drag-and-drop (nhấn chuột trái và giữ rồi di chuyển chuột đến vị trí kế).

Để xoá linh kiện, ta chọn linh kiện rồi nhấn chuột phải làn nữa để xoá.

❖ **Thêm linh kiện mới:**

Nếu linh kiện không tồn tại trong thanh linh kiện, ta phải thực hiện thêm mới từ các thư viện có sẵn bằng cách chọn menu **Library > Pick** hay nhấn **P**.

Cửa sổ lấy linh kiện:



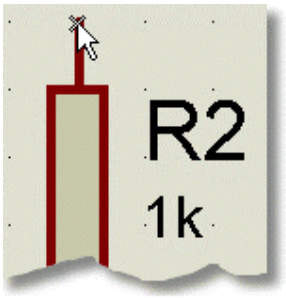
Sưu tầm bởi: www.daihoc.com.vn

Ví dụ như để tìm linh kiện điện trở:

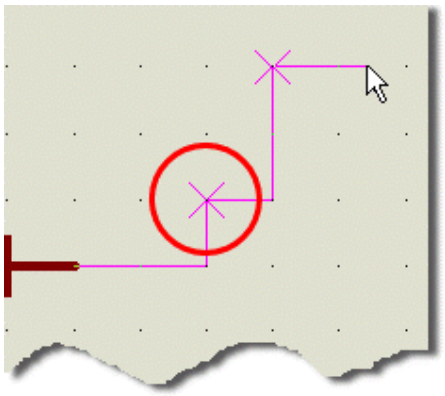
- Gõ Resistor trong vùng Keywords.
- Chọn Category là Resistors.
- Chọn Sub-category là Generic.

❖ **Nối dây:**

- Chuyển con trỏ chuột đến vị trí cần nối dây, trên con trỏ chuột sẽ xuất hiện dấu X



- Di chuyển chuột và nhấn chuột trái khi cần thiết xác định vị trí dây dẫn



- Khi kéo dây đến vị trí cần thiết thì nhấn chuột trái để nối dây.

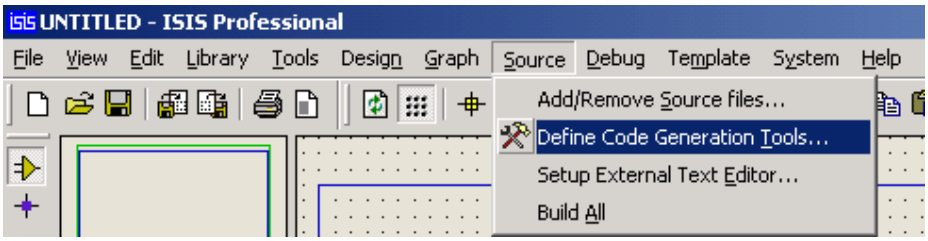
2. Mô phỏng 89C51

Để thực hiện quá trình mô phỏng 89C51 trong Proteus, ta cần thực hiện các bước sau:

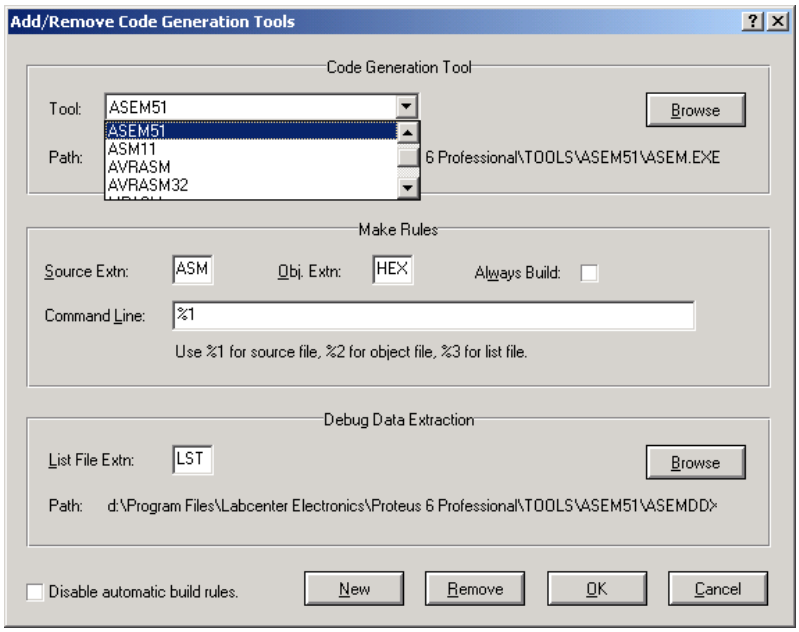
- **Bước 1:** Vẽ mạch nguyên lý.
- **Bước 2:** Định nghĩa chương trình dịch

Chọn menu **Source > Define Code Generation Tools**

Sưu tầm bởi: www.daihoc.com.vn



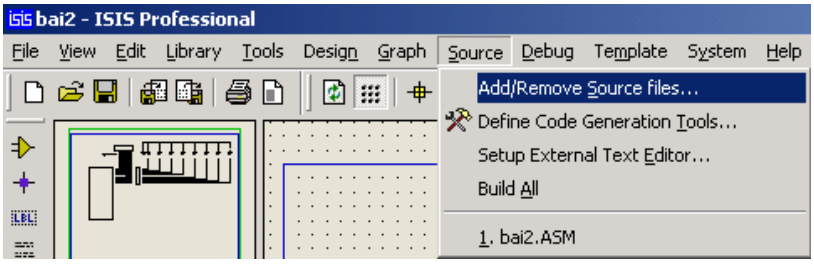
Sau đó thực hiện chọn chương trình dịch mong muốn. Ở đây ta thực hiện mô phỏng cho 89C51 nên chọn chương trình ASEM51.



Phần **Tools**: chọn **ASEM51**, phần **Command Line**: gõ vào **%1**.

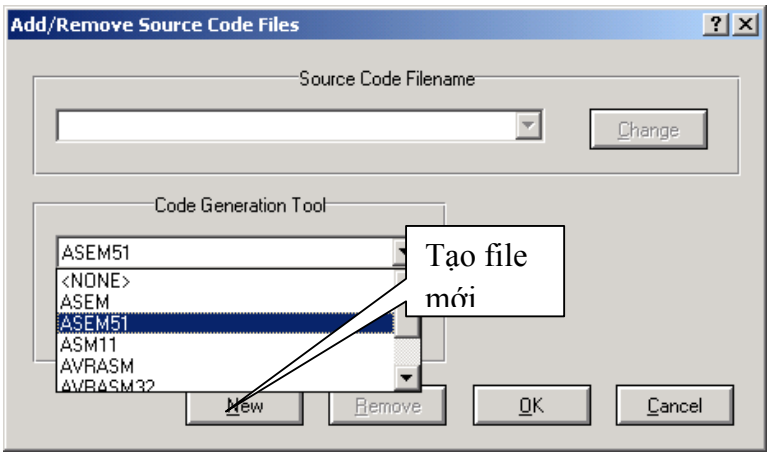
- **Bước 3:** Định nghĩa file chương trình cho 89C51.

Chọn menu **Source > Add/Remove Source File**

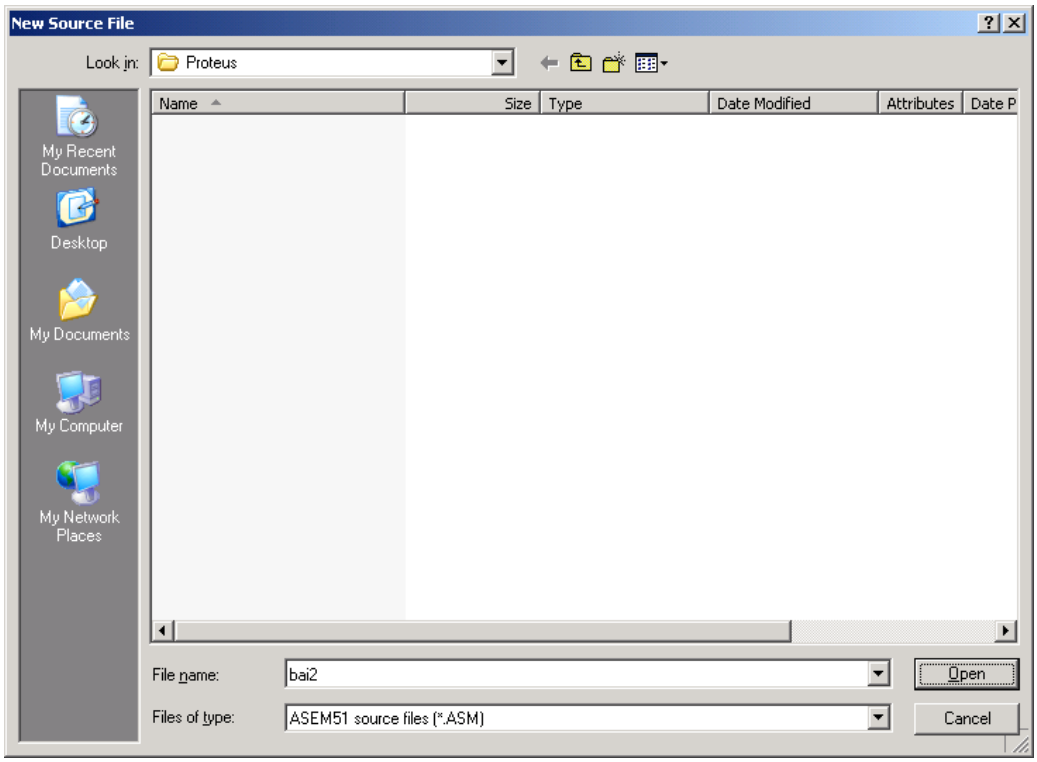


Chọn phần **Code Generation Tool** là **ASEM51**.

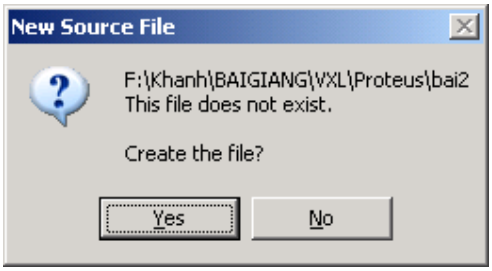
Sưu tầm bởi: www.daihoc.com.vn



Do chưa có chương trình cho 89C51, ta nhấn vào nút New để tạo file. Trong phần **File name**, ta gõ vào tên chương trình (giả sử gõ vào bai2).

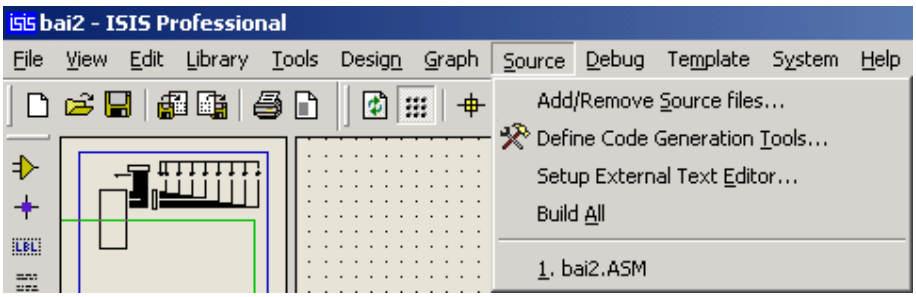


Nếu chưa có file bai2.ASM, Proteus sẽ xuất hiện thông báo yêu cầu tạo file, nhấn **Yes** để tạo:



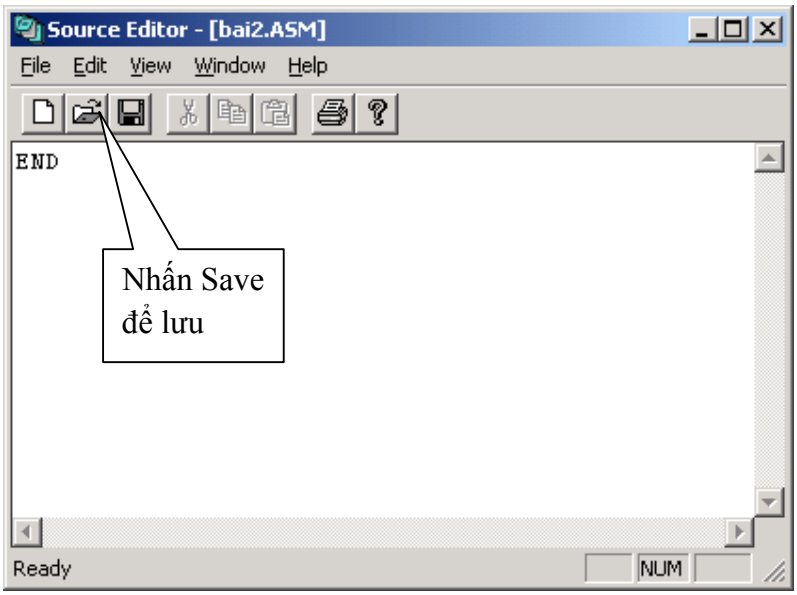
Sưu tầm bởi: www.daihoc.com.vn

Sao khi tạo file thành công, trên menu Source sẽ xuất hiện thêm file bai2.ASM.

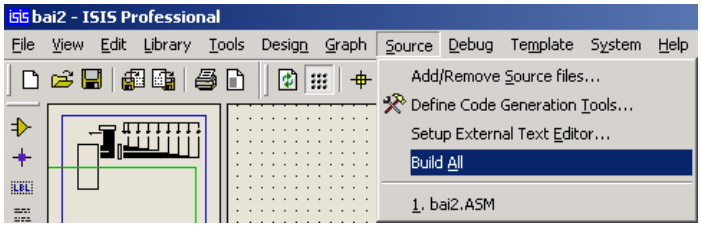


- **Bước 4:** Định nghĩa file thực thi cho 89C51

Chọn file bai2.ASM để soạn thảo chương trình nguồn, nhập vào **END** và nhấn nút **Save**.

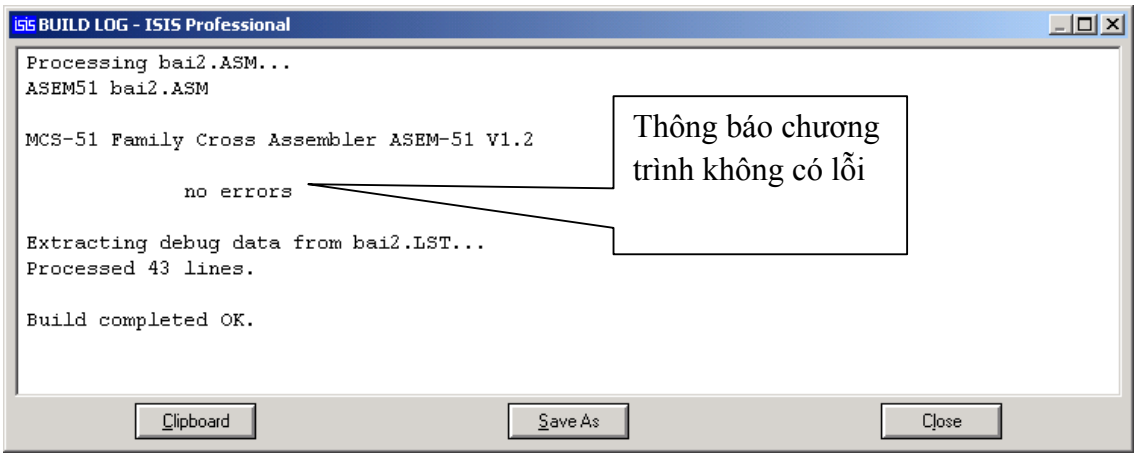


Sau khi lưu file nguồn, ta thực hiện dịch chương trình nguồn.

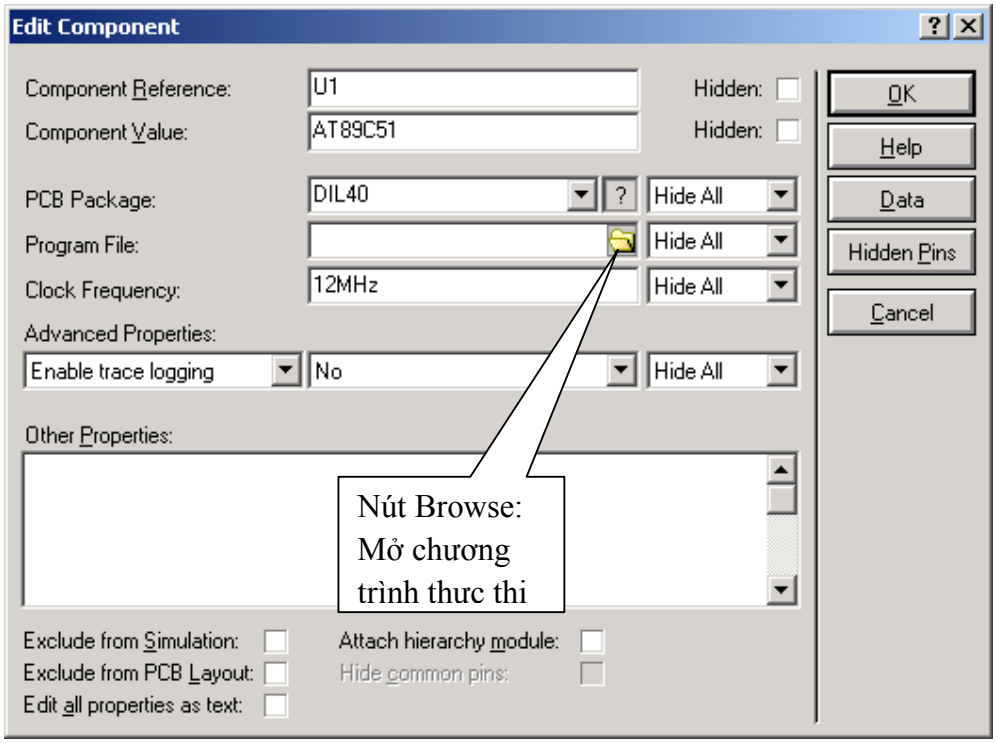


Khi biên dịch, nếu có lỗi, chương trình dịch sẽ thông báo lỗi, nếu không thì sẽ tạo ra file bai2.HEX.

Sưu tầm bởi: www.daihoc.com.vn

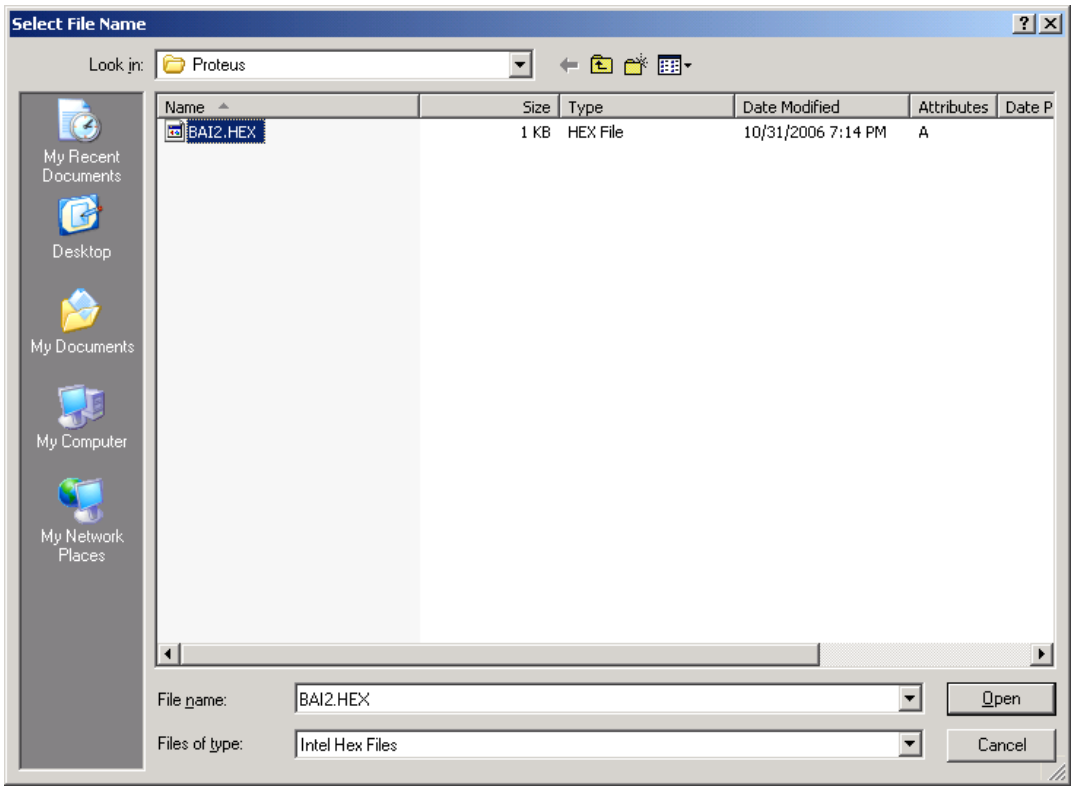


Thực hiện gán file thực thi cho 89C51 bằng cách nhấn chuột phải lên 89C51 để chọn (89C51 sẽ chuyển sang màu đỏ) rồi nhấn chuột trái để mở cửa sổ thuộc tính của 89C51.



Nhấn vào nút **Browse** (hình vẽ trên) để mở chương trình thực thi, chọn chương trình là bai2.HEX

Sưu tầm bởi: www.daihoc.com.vn



Nhấn nút Open để mở file, khi đó trong thuộc tính **Program File** của 89C51 sẽ có tên chương trình là bai2.HEX.

Sau khi gán file thực thi cho 89C51, ta chỉ cần thực hiện sửa chương trình nguồn và biên dịch lại mà không cần gán lại file thực thi.

Có thể tham khảo thêm phần hướng dẫn sử dụng của Proteus ứng dụng tong mô phỏng 89C51 tại Website: http://eed.hutech.edu.vn, phần **Hỗ trợ học tập**

Sưu tầm bởi: www.daihoc.com.vn

Phụ lục 3: TÓM TẮT TẬP LỆNH

Mnemonic	Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS			
ADD A,Rn	Add register to Accumulator	1	12
ADD A,direct	Add direct byte to Accumulator	2	12
ADD A,@Ri	Add indirect RAM to Accumulator	1	12
ADD A,#data	Add immediate data to Accumulator	2	12
ADDC A,Rn	Add register to Accumulator with Carry	1	12
ADDC A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC A,#data	Add immediate data to Acc with Carry	2	12
SUBB A,Rn	Subtract Register from Acc with Borrow	1	12
SUBB A,direct	Subtract direct byte from Acc with Borrow	2	12
SUBB A,@Ri	Subtract indirect RAM from ACC with Borrow	1	12
SUBB A,#data	Subtract immediate data from Acc with borrow	2	12
INC A	Increment Accumulator	1	12
INC Rn	Increment register	1	12
INC direct	Increment direct byte	2	12
INC @Ri	Increment direct RAM	1	12
DEC A	Decrement Accumulator	1	12
DEC Rn	Decrement Register	2	12
DEC direct	Decrement direct byte	2	12
DEC @Ri	Decrement indirect RAM	1	12
INC DPTR	Increment Data Pointer	1	24
MUL AB	Multiply A & B	1	48
DIV AB	Divide A by B	1	48
DA A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS			
ANL A,Rn	AND Register to Accumulator	1	12
ANL A,direct	AND direct byte to Accumulator	2	12
ANL A,@Ri	AND indirect RAM to Accumulator	1	12
ANL A,#data	AND immediate data to Accumulator	2	12
ANL direct,A	AND Accumulator to direct byte	2	12
ANL direct,#data	AND immediate data to direct byte	3	24

Sưu tầm bởi: www.daihoc.com.vn

ORL A,Rn	OR register to Accumulator	1	12
ORL A,direct	OR direct byte to Accumulator	2	12
ORL A,@Ri	OR indirect RAM to Accumulator	1	12
ORL A,#data	OR immediate data to Accumulator	2	12
ORL direct,A	OR Accumulator to direct byte	2	12
ORL direct,#data	OR immediate data to direct byte	3	24
XRL A,Rn	Exclusive-OR register to Accumulator	1	12
XRL A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL direct,A	Exclusive-OR Accumulator to direct Byte	2	12
XRL direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR A	Clear Accumulator	1	12
CPL A	Complement Accumulator	1	12
RL A	Rotate Accumulator Left	1	12
RLC A	Rotate Accumulator Left through the Carry	1	12
RR A	Rotate Accumulator Right	1	12
RRC A	Rotate Accumulator Right through the Carry	1	12
SWAP A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER			
MOV A,Rn	Move register to Accumulator	1	12
MOV A,direct	Move direct byte to Accumulator	2	12
MOV A,@Ri	Move indirect RAM to Accumulator	1	12
MOV A,#data	Move immediate data to Accumulator	2	12
MOV Rn,A	Move Accumulator to register	1	12
MOV Rn,direct	Move direct byte to register	2	24
MOV Rn,#data	Move immediate data to register	2	12
MOV direct,A	Move Accumulator to direct byte	2	12
MOV direct,Rn	Move register to direct byte	2	24
MOV direct,direct	Move direct byte to direct	3	24
MOV direct,@Ri	Move indirect RAM to direct byte	2	24
MOV direct,#data	Move immediate data to direct byte	3	24
MOV @Ri,A	Move Accumulator to indirect RAM	1	12
MOV @Ri,direct	Move direct byte to indirect RAM	2	24
MOV @Ri,#data	Move immediate data to indirect RAM	2	12
MOV DPTR,#data16	Load Data Pointer with a 16-bit Constant	3	24

Sưu tầm bởi: www.daihoc.com.vn

MOVC A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX A,@DPTR	Move Exernal RAM (16-bit addr) to Acc	1	24
MOVX @Ri,A	Move Acc to External RAM (8-bit address)	1	24
MOVX @DPTR,A	Move Acc to External RAM (16-bit address)	1	24
PUSH direct	Push direct byte onto stack	2	24
POP direct	Pop direct byte from stack	2	24
XCH A,Rn	Exchange register with Accumulator	1	12
XCH A,direct	Exchange direct byte with Accumulator	2	12
XCH A,@Ri	Exchange indirect RAM with Accumulator	2	12
XCHD A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12
BOOLEAN VARIABLE MANIPULATION			
CLR C	Clear Carry	1	12
CLR bit	Clear direct bit	2	12
SETB C	Set Carry	1	12
SETB bit	Set direct bit	2	12
CPL C	Complement Carry	1	12
CPL bit	Complement direct bit	2	12
ANL C,bit	AND direct bit to Carry	2	24
ANL C,/bit	AND complement of direct bit to Carry	2	24
ORL C,bit	OR direct bit to Carry	2	24
ORL C,/bit	OR complement of direct bit to Carry	2	24
MOV C,bit	Move direct bit to Carry	2	12
MOV bit,C	Move Carry to direct bit	2	24
JC rel	Jump if Carry is set	2	24
JNC rel	Jump if Carry not set	2	24
JB bit,rel	Jump if direct Bit is set	3	24
JNB bit,rel	Jump if direct Bit is Not set	3	24
JBC bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING			
ACALL addr11	Absolute Subroutine Call	2	24
LCALL addr16	Long Subroutine Call	3	24
RET	Return from Subroutine	1	24
RETI	Return from interrupt	1	24
AJMP addr11	Absolute Jump	2	24
LJMP addr16	Long Jump	3	24
SJMP rel	Short Jump (relative address)	2	24
JMP @A+DPTR	Jump indirect relative to the DPTR	1	24
JZ rel	Jump if Accumulator is Zero	2	24
JNZ rel	Jump if Accumulator is Not Zero	2	24

Sưu tầm bởi: www.daihoc.com.vn

CJNE A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE @Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP	No Operation	1	12

Sưu tầm bởi: www.daihoc.com.vn

Phụ lục 4: MÔ TẢ TẬP LỆNH

1. ACALL addr11

Function: Absolute Call
Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7 through 5, and the second byte of the instruction. The subroutine called must therefore start within the same 2 K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.
Example: Initially SP equals 07H. The label SUBRTN is at program memory location 0345 H. After executing the following instruction,

ACALL SUBRTN

at location 0123H, SP contains 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC contains 0345H.
Bytes: 2
Cycles: 2
Encoding:

A10	A9	A8	1	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

Operation: ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC10-0) ← page address

2. ADD A,<src-byte>

Function: Add
Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.
OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise, OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Sưu tầm bởi: www.daihoc.com.vn

Example: The Accumulator holds 0C3H (1100001lB), and register 0 holds 0AAH (10101010B). The following instruction,

ADD A,R0

leaves 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

2.1. ADD A,Rn

Bytes: 1
Cycles: 1
Encoding:

0	0	1	0	1	r	r	r

Operation: ADD
(A) ← (A) + (Rn)

2.2. ADD A,direct

Bytes: 2
Cycles: 1
Encoding:

0	0	1	0	0	1	0	1	direct address

Operation: ADD
(A) ← (A) + (direct)

2.3. ADD A,@Ri

Bytes: 1
Cycles: 1
Encoding:

0	0	1	0	0	1	1	i

Operation: ADD
(A) ← (A) + ((Ri))

2.4. ADD A,#data

Bytes: 2
Cycles: 1
Encoding:

0	0	1	0	0	1	0	0	immediate data

Operation: ADD
(A) ← (A) + #data

3. ADDC A, <src-byte>

Function: Add with Carry
Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set respectively, if there is a carry-out from bit 7 or bit 3, and

Sưu tầm bởi: www.daihoc.com.vn

cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.
OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.
Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The following instruction,

ADDC A,R0

leaves 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

3.1. ADDC A,Rn

Bytes: 1
Cycles: 1
Encoding:

0	0	1	1	1	r	r	r

Operation: ADDC
(A) ← (A) + (C) + (Rn)

3.2. ADDC A,direct

Bytes: 2
Cycles: 1
Encoding:

0	0	1	1	0	1	0	1	direct address

Operation: ADDC
(A) ← (A) + (C) + (direct)

3.3. ADDC A,@Ri

Bytes: 1
Cycles: 1
Encoding:

0	0	1	1	0	1	1	i

Operation: ADDC
(A) ← (A) + (C) + ((Ri))

3.4. ADDC A,#data

Bytes: 2
Cycles: 1
Encoding:

0	0	1	1	0	1	0	0	immediate data

Sưu tầm bởi: www.daihoc.com.vn

Operation: ADDC
(A) ← (A) + (C) + #data

4. AJMP addr11

Function: Absolute Jump
Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7 through 5, and the second byte of the instruction. The destination must therfore be within the same 2 K block of program memory as the first byte of the instruction following AJMP.
Example: The label JMPADR is at program memory location 0123H. The following instruction,

AJMP JMPADR

is at location 0345H and loads the PC with 0123H.
Bytes: 2
Cycles: 2
Encoding:

A10	A9	A8	0	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

Operation: AJMP
(PC) ← (PC) + 2
(PC10-0) ← page address

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

Function: Logical-AND for byte variables
Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.
The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.
Example: If the Accumulator holds 0C3H (1100001lB), and register 0 holds 55H (01010101B), then the following instruction,

ANL A,R0

leaves 41H (01000001B) in the Accumulator.
When the destination is a directly addressed byte, this instruction clears combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The following instruction,

ANL P1,#01110011B

clears bits 7, 3, and 2 of output port 1.

5.1. ANL A,Rn

Sưu tầm bởi: www.daihoc.com.vn

Bytes: 1
Cycles: 1
Encoding:

0	1	0	1	1	r	r	r

Operation: ANL
(A) ← (A) ∧ (Rn)

5.2. ANL A,direct

Bytes: 2
Cycles: 1
Encoding:

0	1	0	1	0	1	0	1	direct address

Operation: ANL
(A) ← (A) ∧ (direct)

5.3. ANL A,@Ri

Bytes: 1
Cycles: 1
Encoding:

0	1	0	1	0	1	1	i

Operation: ANL
(A) ← (A) ∧ ((Ri))

5.4. ANL A,#data

Bytes: 2
Cycles: 1
Encoding:

0	1	0	1	0	1	0	0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

5.5. ANL direct,A

Bytes: 2
Cycles: 1
Encoding:

0	1	0	1	0	0	1	0	direct address

Operation: ANL
(direct) ← (direct) ∧ (A)

5.6. ANL direct,#data

Bytes: 3
Cycles: 2
Encoding:

Sưu tầm bởi: www.daihoc.com.vn

0	1	0	1	0	0	1	1	direct address	immediate data

Operation: ANL
(direct) ← (direct) ∧ #data

6. ANL C,<src-bit>

Function: Logical-AND for bit variables
Description: If the Boolean value of the source bit is a logical 0, then ANL C clears the carry flag; otherwise, this instruction leaves the carry flag in its current state. A slash (/) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.
Only direct addressing is allowed for the source operand.
Example: Set the carry flag if, and only if, P1.0 = 1, ACC.7 = 1, and OV = 0:
MOV C,P1.0 ;LOAD CARRY WITH INPUT PIN STATE
ANL C,ACC.7 ;AND CARRY WITH ACCUM. BIT 7
ANL C,/OV ;AND WITH INVERSE OF OVERFLOW FLAG

6.1. ANL C,bit

Bytes: 2
Cycles: 2
Encoding:

1	0	0	0	0	0	1	0	bit address

Operation: ANL
(C) ← (C) ∧ (bit)

6.2. ANL C,/bit

Bytes: 2
Cycles: 2
Encoding:

1	0	1	1	0	0	0	0	bit address

Operation: ANL
(C) ← (C) ∧ NOT (bit)

7. CJNE <destbyte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.
Description: CJNE compares the magnitudes of the first two operands and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

Sưu tầm bởi: www.daihoc.com.vn

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
CJNE R7, # 60H, NOT_EQ
; . . . . . ;R7 = 60H.
NOT_EQ: JC REQ_LOW ;IF R7 < 60H.
; . . . . . ;R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the following instruction,

```
WAIT: CJNE A, P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program loops at this point until the P1 data changes to 34H.)

7.1. CJNE A,direct,rel

Bytes: 3
Cycles: 2
Encoding:

1	0	1	1	0	1	0	1	direct address	relative address

Operation: (PC) ← (PC) + 3
IF (A) < > (*direct*) THEN
(PC) ← (PC) + *relative offset*
IF (A) < (*direct*) THEN
(C) ← 1
ELSE
(C) ← 0

7.2. CJNE A,#data,rel

Bytes: 3
Cycles: 2
Encoding:

1	0	1	1	0	1	0	0	immediate data	relative address

Operation: (PC) ← (PC) + 3
IF (A) < > *data* THEN
(PC) ← (PC) + *relative offset*
IF (A) < *data* THEN
(C) ← 1
ELSE
(C) ← 0

7.3. CJNE Rn,#data,rel

Sưu tầm bởi: www.daihoc.com.vn

Bytes: 3
Cycles: 2
Encoding:

1	0	1	1	1	r	r	r	immediate data	relative address

Operation: (PC) ← (PC) + 3
IF (Rn) < > *data* THEN
(PC) ← (PC) + *relative offset*
IF (Rn) < *data* THEN
(C) ← 1
ELSE
(C) ← 0

7.4. CJNE @Ri,data,rel

Bytes: 3
Cycles: 2
Encoding:

1	0	1	1	0	1	1	i	immediate data	relative address

Operation: (PC) ← (PC) + 3
IF ((Ri)) < > *data* THEN
(PC) ← (PC) + *relative offset*
IF ((Ri)) < *data* THEN
(C) ← 1
ELSE
(C) ← 0

8. CLR A

Function: Clear Accumulator
Description: CLR A clears the Accumulator (all bits set to 0). No flags are affected
Example: The Accumulator contains 5CH (01011100B). The following instruction,
CLR A
leaves the Accumulator set to 00H (00000000B).
Bytes: 1
Cycles: 1
Encoding:

1	1	1	0	0	1	0	0

Operation: CLR
(A) ← 0

9. CLR bit

Function: Clear bit
Description: CLR bit clears the indicated bit (reset to 0). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Sưu tầm bởi: www.daihoc.com.vn

Example: Port 1 has previously been written with 5DH (01011101B). The following instruction,

CLR P1.2

leaves the port set to 59H (01011001B).

9.1. CLR C

Bytes: 1
Cycles: 1
Encoding:

1	1	0	0	0	0	1	1

Operation: CLR
(C) ← 0

9.2. CLR bit

Bytes: 2
Cycles: 1
Encoding:

1	1	0	0	0	0	1	0	bit address

Operation: CLR
(bit) ← 0

10. CPL A

Function: Complement Accumulator
Description: CPLA logically complements each bit of the Accumulator (one's complement). Bits which previously contained a 1 are changed to a 0 and vice-versa. No flags are affected.
Example: The Accumulator contains 5CH (01011100B). The following instruction,

CPL A

leaves the Accumulator set to 0A3H (10100011B).
Bytes: 1
Cycles: 1
Encoding:

1	1	1	1	0	1	0	0

Operation: CPL
(A) ← NOT (A)

11. CPL bit

Function: Complement bit
Description: CPL bit complements the bit variable specified. A bit that had been a 1 is changed to 0 and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.
Note: When this instruction is used to modify an output pin, the value used as the original data is read from the output data latch, *not* the input pin.

Sưu tầm bởi: www.daihoc.com.vn

Example: Port 1 has previously been written with 5BH (01011101B). The following instruction sequence,

CPL P1.1
CPL P1.2

leaves the port set to 5BH (01011011B).

11.1. CPL C

Bytes: 1
Cycles: 1
Encoding:

1	0	1	1	0	0	1	1

1 0 1 1 0 0 1 1
Operation: CPL
(C) ← NOT (C)

11.2. CPL bit

Bytes: 2
Cycles: 1
Encoding:

1	0	1	1	0	0	1	0	bit address

Operation: CPL
(bit) ← NOT (bit)

12. DA A

Function: Decimal-adjust Accumulator for Addition
Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.
If Accumulator bits 3 through 0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition sets the carry flag if a carry-out of the low-order four-bit field propagates through all high-order bits, but it does not clear the carry flag otherwise.
If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this sets the carry flag if there is a carry-out of the high-order bits, but does not clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.
All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.
Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Sưu tầm bởi: www.daihoc.com.vn

Example: The Accumulator holds the value 56H (01010110B), representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B), representing the packed BCD digits of the decimal number 67. The carry flag is set. The following instruction sequence

```
ADDC A,R3
DA A
```

first performs a standard two's-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags are cleared. The Decimal Adjust instruction then alters the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag is set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum of 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the following instruction sequence,

```
ADD A, # 99H
DA A
```

leaves the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 - 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	0	0

Operation: DA

-contents of Accumulator are BCD

IF [[(A3-0) > 9] $\vee$ [(AC) = 1]] THEN (A3-0) $\leftarrow$ (A3-0) + 6

AND

IF [[(A7-4) > 9] $\vee$ [(C) = 1]] THEN (A7-4) $\leftarrow$ (A7-4) + 6

13. DEC byte

Function: Decrement

Description: DEC byte decrements the variable indicated by 1. An original value of 00H underflows to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively.

The following instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

leaves register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

Sưu tầm bởi: www.daihoc.com.vn

13.1. DEC A

Bytes: 1
Cycles: 1
Encoding:

0	0	0	1	0	1	0	0

Operation: DEC
(A) ← (A) - 1

13.2. DEC Rn

Bytes: 1
Cycles: 1
Encoding:

0	0	0	1	1	r	r	r

Operation: DEC
(Rn) ← (Rn) - 1

13.3. DEC direct

Bytes: 2
Cycles: 1
Encoding:

0	0	0	1	0	1	0	1	direct address

Operation: DEC
(direct) ← (direct) - 1

13.4. DEC @Ri

Bytes: 1
Cycles: 1
Encoding:

0	0	0	1	0	1	1	i

Operation: DEC
((Ri)) ← ((Ri)) - 1

14. DIV AB

Function: Divide
Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B.
The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags are cleared.
Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register are undefined and the overflow flag are set. The carry flag is cleared in any case.
Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The following instruction,

Sưu tầm bởi: www.daihoc.com.vn

DIV AB

leaves 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 x 18) + 17. Carry and OV are both cleared.

Bytes: 1

Cycles: 4

Encoding:

1	0	0	0	0	1	0	0

Operation: DIV

$(A)_{15-8} \leftarrow (A)/(B)$

$(B)_{7-0}$

15. DJNZ <byte>,<rel addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H underflows to 0FFH. No flags are affected. The branch destination is computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction. The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The following instruction sequence,

```
DJNZ 40H,LABEL_1
DJNZ 50H,LABEL_2
DJNZ 60H,LABEL_3
```

causes a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way to execute a program loop a given number of times or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The following instruction sequence,

```
MOV R2, # 8
TOGGLE: CPL P1.7
DJNZ R2,TOGGLE
```

toggles P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse lasts three machine cycles; two for DJNZ and one to alter the pin.

15.1. DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1	1	0	1	1	r	r	r	relative address

Operation: DJNZ

Sưu tầm bởi: www.daihoc.com.vn

(PC) ← (PC) + 2
(Rn) ← (Rn) - 1
IF (Rn) > 0 or (Rn) < 0 THEN
    (PC) ← (PC) + rel

15.2. DJNZ direct,rel

Bytes: 3
Cycles: 2
Encoding:

1	1	0	1	0	1	0	1	direct address	relative address

Operation: DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) - 1
IF (direct) > 0 or (direct) < 0 THEN
    (PC) ← (PC) + rel

16. INC <byte>

Function: Increment
Description: INC increments the indicated variable by 1. An original value of 0FFH overflows to 00H. No flags are affected.
Three addressing modes are allowed: register, direct, or register-indirect.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.
Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The following instruction sequence,

```
INC @R0
INC R0
INC @R0
```

leaves register 0 set to 7FH and internal RAM locations 7EH and 7FH holding 00H and 41H, respectively.

16.1. INC A

Bytes: 1
Cycles: 1
Encoding:

0	0	0	0	0	1	0	0

Operation: INC
(A) ← (A) + 1

16.2. INC Rn

Bytes: 1
Cycles: 1
Encoding:

Sưu tầm bởi: www.daihoc.com.vn

0	0	0	0	1	r	r	r

Operation: INC
(Rn) ← (Rn) + 1

16.3. INC direct

Bytes: 2
Cycles: 1
Encoding:

0	0	0	0	0	1	0	1	direct address

Operation: INC
(direct) ← (direct) + 1

16.4. INC @Ri

Bytes: 1
Cycles: 1
Encoding:

0	0	0	0	0	1	1	i

Operation: INC
((Ri)) ← ((Ri)) + 1

17. INC DPTR

Function: Increment Data Pointer
Description: INC DPTR increments the 16-bit data pointer by 1. A 16-bit increment (modulo 216) is performed, and an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H increments the high-order byte (DPH). No flags are affected. This is the only 16-bit register which can be incremented.
Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The following instruction sequence,

```
INC DPTR
INC DPTR
INC DPTR
```

changes DPH and DPL to 13H and 01H.
Bytes: 1
Cycles: 2
Encoding:

1	0	1	0	0	0	1	1

Operation: INC
(DPTR) ← (DPTR) + 1

18. JB bit,rel

Function: Jump if Bit set
Description: If the indicated bit is a one, JB jump to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding

Sưu tầm bởi: www.daihoc.com.vn

the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The following instruction sequence,

```
JB P1.2,LABEL1
JB ACC. 2,LABEL2
```

causes program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0	0	1	0	0	0	0	0	bit address	relative address

Operation: JB

(PC) ← (PC) + 3

IF (bit) = 1 THEN

(PC) ← (PC) + rel

19. JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, JBC branches to the address indicated; otherwise, it proceeds with the next instruction. *The bit will not be cleared if it is already a zero*. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The Accumulator holds 56H (01010110B). The following instruction sequence,

```
JBC ACC.3,LABEL1
JBC ACC.2,LABEL2
```

causes program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0	0	0	1	0	0	0	0	bit address	relative address

Operation: JBC

(PC) ← (PC) + 3

IF (bit) = 1 THEN

(bit) ← 0

(PC) ← (PC) +rel

20. JC rel

Function: Jump if Carry is set

Sưu tầm bởi: www.daihoc.com.vn

Description: If the carry flag is set, JC branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.
Example: The carry flag is cleared. The following instruction sequence,

```
JC LABEL1
CPL C
JC LABEL 2
```

sets the carry and causes program execution to continue at the instruction identified by the label LABEL2.
Bytes: 2
Cycles: 2
Encoding:

0	1	0	0	0	0	0	0	relative address

Operation: JC
(PC) ← (PC) + 2
IF (C) = 1 THEN
(PC) ← (PC) + rel

21. JMP @A+DPTR

Function: Jump indirect
Description: JMP @A+DPTR adds the eight-bit unsigned contents of the Accumulator with the 16-bit data pointer and loads the resulting sum to the program counter. This is the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 216): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.
Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions branches to one of four AJMP instructions in a jump table starting at JMP_TBL.

```
MOV DPTR, # JMP_TBL
JMP @A + DPTR
JMP_TBL: AJMP LABEL0
AJMP LABEL1
AJMP LABEL2
AJMP LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution jumps to label LABEL2. Because AJMP is a 2-byte instruction, the jump instructions start at every other address.
Bytes: 1
Cycles: 2
Encoding:

0	1	1	1	0	0	1	1

Operation: JMP

Sưu tầm bởi: www.daihoc.com.vn

(PC) ← (A) + (DPTR)

22. JNB bit,rel

Function: Jump if Bit Not set
Description: If the indicated bit is a 0, JNB branches to the indicated address; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified*. No flags are affected.
Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The following instruction sequence,

```
JNB P1.3,LABEL1
JNB ACC.3,LABEL2
```

causes program execution to continue at the instruction at label LABEL2.
Bytes: 3
Cycles: 2
Encoding:

0	0	1	1	0	0	0	0	bit address	relative address

Operation: JNB
(PC) ← (PC) + 3
IF (bit) = 0 THEN
(PC) ← (PC) + rel

23. JNC rel

Function: Jump if Carry not set
Description: If the carry flag is a 0, JNC branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signal relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.
Example: The carry flag is set. The following instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

clears the carry and causes program execution to continue at the instruction identified by the label LABEL2.
Bytes: 2
Cycles: 2
Encoding:

0	1	0	1	0	0	0	0	relative address

Operation: JNC
(PC) ← (PC) + 2
IF (C) = 0 THEN (PC) ← (PC) + rel

Sưu tầm bởi: www.daihoc.com.vn

24. JNZ rel

Function: Jump if Accumulator Not Zero
Description: If any bit of the Accumulator is a one, JNZ branches to the indicated address; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Example: The Accumulator originally holds 00H. The following instruction sequence,

```
JNZ LABEL1
INC A
JNZ LABEL2
```

sets the Accumulator to 01H and continues at label LABEL2.
Bytes: 2
Cycles: 2
Encoding:

0	1	1	1	0	0	0	0	relative address

Operation: JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0 THEN (PC) ← (PC) + rel

25. JZ rel

Function: Jump if Accumulator Zero
Description: If all bits of the Accumulator are 0, JZ branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Example: The Accumulator originally contains 01H. The following instruction sequence,

```
JZ LABEL1
DEC A
JZ LABEL2
```

changes the Accumulator to 00H and causes program execution to continue at the instruction identified by the label LABEL2.
Bytes: 2
Cycles: 2
Encoding:

0	1	1	0	0	0	0	0	relative address

Operation: JZ
(PC) ← (PC) + 2
IF (A) = 0 THEN (PC) ← (PC) + rel

26. LCALL addr16

Sưu tầm bởi: www.daihoc.com.vn

Function: Long call
Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K byte program memory address space. No flags are affected.
Example: Initially the Stack Pointer equals 07H. The label SUBRTN is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.
Bytes: 3
Cycles: 2
Encoding:

0	0	0	1	0	0	1	0	addr15-addr8	addr7-addr0

Operation: LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC) ← addr15-0

27. LJMP addr16

Function: Long Jump
Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.
Example: The label JMPADR is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.
Bytes: 3
Cycles: 2
Encoding:

0	0	0	0	0	0	1	0	addr15-addr8	addr7-addr0

Operation: LJMP
(PC) ← addr15-0

28. MOV <destbyte>, <src-byte>

Sưu tầm bởi: www.daihoc.com.vn

Function: Move byte variable
Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.
This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.
Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV R0,#30H ;R0 ←30H
MOV A,@R0 ;A ←40H
MOV R1,A ;R1 ←40H
MOV B,@R1 ;B ← 10H
MOV @R1,P1 ;RAM (40H) ← 0CAH
MOV P2,P1 ;P2 ← 0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

28.1. MOV A,Rn

Bytes: 1
Cycles: 1
Encoding:

1	1	1	0	1	r	r	r

Operation: MOV
(A) ← (Rn)

28.2. *MOV A,direct

Bytes: 2
Cycles: 1
Encoding:

1	1	1	0	0	1	0	1	direct address

Operation: MOV
(A) ← (direct)
*** MOV A,ACC is not a valid Instruction.**

28.3. MOV A,@Ri

Bytes: 1
Cycles: 1
Encoding:

1	1	1	0	0	1	1	i

Operation: MOV
(A) ← ((Ri))

28.4. MOV A,#data

Sưu tầm bởi: www.daihoc.com.vn

Bytes: 2
Cycles: 1
Encoding:

0	1	1	1	0	1	0	0	immediate data

Operation: MOV
(A) ← #data

28.5. MOV Rn,A

Bytes: 1
Cycles: 1
Encoding:

1	1	1	1	1	r	r	r

Operation: MOV
(Rn) ← (A)

28.6. MOV Rn,direct

Bytes: 2
Cycles: 2
Encoding:

1	0	1	0	1	r	r	r	direct address

Operation: MOV
(Rn) ← (direct)

28.7. MOV Rn,#data

Bytes: 2
Cycles: 1
Encoding:

0	1	1	1	1	r	r	r	immediate data

Operation: MOV
(Rn) ← #data

28.8. MOV direct,A

Bytes: 2
Cycles: 1
Encoding:

1	1	1	1	0	1	0	1	direct address

Operation: MOV
(direct) ← (A)

28.9. MOV direct,Rn

Bytes: 2
Cycles: 2
Encoding:

Sưu tầm bởi: www.daihoc.com.vn

1	0	0	0	1	r	r	r	direct address

Operation: MOV
(direct) ← (Rn)

28.10. MOV direct,direct

Bytes: 3
Cycles: 2
Encoding:

1	0	0	0	0	1	0	1	direct address (source)	direct address (destination)

Operation: MOV
(direct) ← (direct)

28.11. MOV direct,@Ri

Bytes: 2
Cycles: 2
Encoding:

1	0	0	0	0	1	1	i	direct address

Operation: MOV
(direct) ← ((Ri))

28.12. MOV direct,#data

Bytes: 3
Cycles: 2
Encoding:

0	1	1	1	0	1	0	1	direct address	immediate data

Operation: MOV
(direct) ← #data

28.13. MOV @Ri,A

Bytes: 1
Cycles: 1
Encoding:

1	1	1	1	0	1	1	i

Operation: MOV
((Ri)) ← (A)

28.14. MOV @Ri,direct

Bytes: 2
Cycles: 2
Encoding:

1	0	1	0	0	1	1	i	direct address

Operation: MOV
((Ri)) ← (direct)

Sưu tầm bởi: www.daihoc.com.vn

28.15. MOV @Ri,#data

Bytes: 2
Cycles: 1
Encoding:

0	1	1	1	0	1	1	i	immediate data

Operation: MOV
((Ri)) ← #data

29. MOV <destbit>, <src-bit>

Function: Move bit data
Description: MOV <dest-bit>,<src-bit> copies the Boolean variable indicated by the second operand into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.
Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV P1.3,C
MOV C,P3.3
MOV P1.2,C
```

leaves the carry cleared and changes Port 1 to 39H (00111001B).

29.1. MOV C,bit

Bytes: 2
Cycles: 1
Encoding:

1	0	1	0	0	0	1	0	bit address

Operation: MOV
(C) ← (bit)

29.2. MOV bit,C

Bytes: 2
Cycles: 2
Encoding:

1	0	0	1	0	0	1	0	bit address

Operation: MOV
(bit) ← (C)

30. MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant
Description: MOV DPTR,#data16 loads the Data Pointer with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte

Sưu tầm bởi: www.daihoc.com.vn

(DPL) holds the lower-order byte. No flags are affected. This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR, # 1234H

loads the value 1234H into the Data Pointer: DPH holds 12H, and DPL holds 34H.

Bytes: 3

Cycles: 2

Encoding:

1	0	0	1	0	0	0	0	immed. $data_{15-8}$	immed. $data_{7-0}$

Operation: MOV

(DPTR) ← $\#data_{15-0}$

DPH ←$\#data_{15-8}$

DPL ← $\#data_{7-0}$

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

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte or constant from program memory. The address of the byte fetched is the sum of the original unsigned 8-bit Accumulator contents and the contents of a 16-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC A
MOVC A,@A+PC
RET
DB 66H
DB 77H
DB 88H
DB 99H
```

If the subroutine is called with the Accumulator equal to 01H, it returns with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separate the MOVC from the table, the corresponding number is added to the Accumulator instead.

31.1. MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1	0	0	1	0	0	1	1

Sưu tầm bởi: www.daihoc.com.vn

Operation: MOVC
(A) ← ((A) + (DPTR))

31.2. MOVC A,@A+PC

Bytes: 1
Cycles: 2
Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

32. MOVX <destbyte>,<src-byte>

Function: Move External
Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, which is why "X" is appended to MOV. There are two types of instructions, differing in whether they provide an 8-bit or 16-bit indirect address to the external data RAM.
In the first type, the contents of R0 or R1 in the current register bank provide an 8-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins are controlled by an output instruction preceding the MOVX.
In the second type of MOVX instruction, the Data Pointer generates a 16-bit address. P2 outputs the high-order eight address bits (the contents of DPH), while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents, while the P2 output buffers emit the contents of DPH. This form of MOVX is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.
It is possible to use both MOVX types in some situations. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2, followed by a MOVX instruction using R0 or R1.
Example: An external 256 byte RAM using multiplexed address/data lines is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX A,@R1
MOVX @R0,A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

32.1. MOVX A,@Ri

Bytes: 1

Sưu tầm bởi: www.daihoc.com.vn

Cycles: 2
Encoding:

1	1	1	0	0	0	1	i

Operation: MOVX
(A) ← ((Ri))

32.2. MOVX A,@DPTR

Bytes: 1
Cycles: 2
Encoding:

1	1	1	0	0	0	0	0

Operation: MOVX
(A) ← ((DPTR))

32.3. MOVX @Ri,A

Bytes: 1
Cycles: 2
Encoding:

1	1	1	1	0	0	1	i

Operation: MOVX
((Ri)) ← (A)

32.4. MOVX @DPTR,A

Bytes: 1
Cycles: 2
Encoding:

1	1	1	1	0	0	0	0

Operation: MOVX
(DPTR) ← (A)

33. MUL AB

Function: Multiply
Description: MUL AB multiplies the unsigned 8-bit integers in the Accumulator and register B. The low-order byte of the 16-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH), the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.
Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.
Bytes: 1
Cycles: 4

Sưu tầm bởi: www.daihoc.com.vn

Encoding:

1	0	1	0	0	1	0	0

Operation: MUL
$(A)_{7-0} \leftarrow (A) \text{ X } (B)$
$(B)_{15-8}$

34. NOP

Function: No Operation
Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.
Example: A low-going output pulse on bit 7 of Port 2 must last exactly 5 cycles. A simple SETB/CLR sequence generates a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the following instruction sequence,

```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

Bytes: 1
Cycles: 1
Encoding:

0	0	0	0	0	0	0	0

Operation: NOP
$(PC) \leftarrow (PC) + 1$

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

Function: Logical-OR for byte variables
Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.
The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.
Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, *not* the input pins.
Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the following instruction,

```
ORL A,R0
```

leaves the Accumulator holding the value 0D7H (1101011lB).When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable

Sưu tầm bởi: www.daihoc.com.vn

computed in the Accumulator at run-time. The instruction,

ORL P1,#00110010B

sets bits 5, 4, and 1 of output Port 1.

35.1. ORL A,Rn

Bytes: 1
Cycles: 1
Encoding:

0	1	0	0	1	r	r	r

Operation: ORL
(A) ← (A) ∨ (Rn)

35.2. ORL A,direct

Bytes: 2
Cycles: 1
Encoding:

0	1	0	0	0	1	0	1	direct address

Operation: ORL
(A) ← (A) ∨ (direct)

35.3. ORL A,@Ri

Bytes: 1
Cycles: 1
Encoding:

0	1	0	0	0	1	1	i

Operation: ORL
(A) ← (A) ∨ ((Ri))

35.4. ORL A,#data

Bytes: 2
Cycles: 1
Encoding:

0	1	0	0	0	1	0	0	immediate data

Operation: ORL
(A) ← (A) ∨ #data

35.5. ORL direct,A

Bytes: 2
Cycles: 1
Encoding:

0	1	0	0	0	0	1	0	direct address

Operation: ORL
(direct) ← (direct) ∨ (A)

Sưu tầm bởi: www.daihoc.com.vn

35.6. ORL direct,#data

Bytes: 3
Cycles: 2
Encoding:

0	1	0	0	0	0	1	1	direct address	immediate data

Operation: ORL
(direct) ← (direct) ∨ #data

36. ORL C,<src-bit>

Function: Logical-OR for bit variables
Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (/) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.
Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV C,P1.0 ;LOAD CARRY WITH INPUT PIN P10
ORL C,ACC.7 ;OR CARRY WITH THE ACC. BIT 7
ORL C,/OV ;OR CARRY WITH THE INVERSE OF OV.
```

36.1. ORL C,bit

Bytes: 2
Cycles: 2
Encoding:

0	1	1	1	0	0	1	0	bit address

Operation: ORL
(C) ← (C) ∨ (bit)

36.2. ORL C,/bit

Bytes: 2
Cycles: 2
Encoding:

1	0	1	0	0	0	0	0	bit address

Operation: ORL
(C) ← (C) ∨ (bit)

37. POP direct

Function: Pop from stack.
Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Sưu tầm bởi: www.daihoc.com.vn

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The following instruction sequence,

```
POP DPH
POP DPL
```

leaves the Stack Pointer equal to the value 30H and sets the Data Pointer to 0123H. At this point, the following instruction,

```
POP SP
```

leaves the Stack Pointer set to 20H. In this special case, the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2
Cycles: 2
Encoding:

1	1	0	1	0	0	0	0	direct address

Operation: POP
(direct) ← ((SP))
(SP) ← (SP) - 1

38. PUSH direct

Function: Push onto stack
Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.
Example: On entering an interrupt routine, the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The following instruction sequence,

```
PUSH DPL
PUSH DPH
```

leaves the Stack Pointer set to 0BH and stores 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2
Cycles: 2
Encoding:

1	1	0	0	0	0	0	0	direct address

Operation: PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

39. RET

Function: Return from subroutine
Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Sưu tầm bởi: www.daihoc.com.vn

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The following instruction,

RET

leaves the Stack Pointer equal to the value 09H. Program execution continues at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	0	0	0	1	0

Operation: RET

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

40. RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt was pending when the RETI instruction is executed, that one instruction is executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The following instruction,

RETI

leaves the Stack Pointer equal to 09H and returns program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	1	0	0	1	0

Operation: RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

41. RL A

Sưu tầm bởi: www.daihoc.com.vn

Function: Rotate Accumulator Left
Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.
Example: The Accumulator holds the value 0C5H (11000101B). The following instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.
Bytes: 1
Cycles: 1
Encoding:

0	0	1	0	0	0	1	1

Operation: RL
$(A_{n+1}) \leftarrow (A_n)$ n = 0 - 6
$(A_0) \leftarrow (A_7)$

42. RLC A

Function: Rotate Accumulator Left through the Carry flag
Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example: The Accumulator holds the value 0C5H(11000101B), and the carry is zero. The following instruction,

RLC A

leaves the Accumulator holding the value 8BH (10001010B) with the carry set.
Bytes: 1
Cycles: 1
Encoding:

0	0	1	1	0	0	1	1

Operation: RLC
$(A_{n+1}) \leftarrow (A_n)$ n = 0 - 6
$(A_0) \leftarrow (C)$
$(C) \leftarrow (A_7)$

43. RR A

Function: Rotate Accumulator Right
Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Example: The Accumulator holds the value 0C5H (11000101B). The following instruction,

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.
Bytes: 1

Sưu tầm bởi: www.daihoc.com.vn

Cycles: 1
Encoding:

0	0	0	0	0	0	1	1

Operation: RR
$(A_n) \leftarrow (A_{n+1})$ n = 0 - 6
$(A_7) \leftarrow (A_0)$

44. RRC A

Function: Rotate Accumulator Right through Carry flag
Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.
Example: The Accumulator holds the value 0C5H (11000101B), the carry is zero. The following instruction,

```
RRC A
```

leaves the Accumulator holding the value 62 (01100010B) with the carry set.
Bytes: 1
Cycles: 1
Encoding:

0	0	0	1	0	0	1	1

Operation: RRC
$(A_n) \leftarrow (A_{n+1})$ n = 0 - 6
$(A_7) \leftarrow (C)$
$(C) \leftarrow (A_0)$

45. SETB <bit>

Function: Set Bit
Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.
Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The following instructions,

```
SETB C
SETB P1.0
```

sets the carry flag to 1 and changes the data output on Port 1 to 35H (00110101B).

45.1. SETB C

Bytes: 1
Cycles: 1
Encoding:

1	1	0	1	0	0	1	1

Operation: SETB
$(C) \leftarrow 1$

45.2. SETB bit

Sưu tầm bởi: www.daihoc.com.vn

Bytes: 2
Cycles: 1
Encoding:

1	1	0	1	0	0	1	0	bit address

Operation: SETB
(bit) ← 1

46. SJMP rel

Function: Short Jump
Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction 127 bytes following it.
Example: The label RELADR is assigned to an instruction at program memory location 0123H. The following instruction,

SJMP RELADR

assembles into location 0100H. After the instruction is executed, the PC contains the value 0123H.
Note: Under the above conditions the instruction following SJMP is at 102H. Therefore, the displacement byte of the instruction is the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH is a one-instruction infinite loop.
Bytes: 2
Cycles: 2
Encoding:

1	0	0	0	0	0	0	0	relative address

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

47. SUBB A,<src-byte>

Function: Subtract with borrow
Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7 and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple-precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3 and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6. When subtracting signed integers, OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number. The

Sưu tầm bởi: www.daihoc.com.vn

source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by CLR C instruction.

47.1. SUBB A,Rn

Bytes: 1
Cycles: 1
Encoding:

1	0	0	1	1	r	r	r

Operation: SUBB
(A) ← (A) - (C) - (Rn)

47.2. SUBB A,direct

Bytes: 2
Cycles: 1
Encoding:

1	0	0	1	0	1	0	1	direct address

Operation: SUBB
(A) ← (A) - (C) - (direct)

47.3. SUBB A,@Ri

Bytes: 1
Cycles: 1
Encoding:

1	0	0	1	0	1	1	i

Operation: SUBB
(A) ← (A) - (C) - ((Ri))

47.4. SUBB A,#data

Bytes: 2
Cycles: 1
Encoding:

1	0	0	1	0	1	0	0	immediate data

Operation: SUBB
(A) ← (A) - (C) - #data

Sưu tầm bởi: www.daihoc.com.vn

48. SWAP A

Function: Swap nibbles within the Accumulator
Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3 through 0 and bits 7 through 4). The operation can also be thought of as a 4-bit rotate instruction. No flags are affected.
Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
SWAP A
```

leaves the Accumulator holding the value 5CH (01011100B).
Bytes: 1
Cycles: 1
Encoding:

1	1	0	0	0	1	0	0

Operation: SWAP
$(A_{3-0}) \leftrightarrow (A_{7-4})$

49. XCH A,<byte>

Function: Exchange Accumulator with byte variable
Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.
Example: R0 contains the address 20H. The Accumulator holds the value 3FH (0011111lB). Internal RAM location 20H holds the value 75H (01110101B). The following instruction,

```
XCH A,@R0
```

leaves RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

49.1. XCH A,Rn

Bytes: 1
Cycles: 1
Encoding:

1	1	0	0	1	r	r	r

Operation: XCH
(A) ↔ ((Rn)

49.2. XCH A,direct

Bytes: 2
Cycles: 1
Encoding:

1	1	0	0	0	1	0	1	direct address

Operation: XCH
(A) ↔ (direct)

Sưu tầm bởi: www.daihoc.com.vn

49.3. XCH A,@Ri

Bytes: 1
Cycles: 1
Encoding:

1	1	0	0	0	1	1	i

Operation: XCH
(A) ↔ ((Ri))

50. XCHD A,@Ri

Function: Exchange Digit
Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3 through 0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.
Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The following instruction,

XCHD A,@R0

leaves RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.
Bytes: 1
Cycles: 1
Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD
$(A_{3-0}) \leftrightarrow ((Ri_{3-0}))$

51. XRL <destbyte>,<src-byte>

Function: Logical Exclusive-OR for byte variables
Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.
The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.
Note: When this instruction is used to modify an output port, the value used as the original port data is read from the output data latch, *not* the input pins.
Example: If the Accumulator holds 0C3H (1100001lB) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

leaves the Accumulator holding the value 69H (01101001B).
When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to

Sưu tầm bởi: www.daihoc.com.vn

be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The following instruction,

XRL P1,#00110001B

complements bits 5, 4, and 0 of output Port 1.

51.1. XRL A,Rn

Bytes: 1
Cycles: 1
Encoding:

0	1	1	0	1	r	r	r

Operation: XRL
(A) ¬ (A) XOR (Rn)

51.2. XRL A,direct

Bytes: 2
Cycles: 1
Encoding:

0	1	1	0	0	1	0	1	direct address

Operation: XRL
(A) ¬ (A) XOR (direct)

51.3. XRL A,@Ri

Bytes: 1
Cycles: 1
Encoding:

0	1	1	0	0	1	1	i

Operation: XRL
(A) ¬ (A) XOR (Ri)

51.4. XRL A,@#data

Bytes: 2
Cycles: 1
Encoding:

0	1	1	0	0	1	0	0	immediate data

Operation: XRL
(A) ¬ (A) XOR #data

51.5. XRL direct,A

Bytes: 2
Cycles: 1
Encoding:

0	1	1	0	0	0	1	0	direct address

Sưu tầm bởi: www.daihoc.com.vn

Operation: XRL
(direct) ¬ (direct) XOR (A)

51.6. XRL direct,#data

Bytes: 3
Cycles: 2
Encoding:

0	1	1	0	0	0	1	1	direct address	immediate data

Operation: XRL
(direct) ¬ (direct) XOR #data

Sưu tầm bởi: www.daihoc.com.vn