ỦY BAN NHÂN DÂN QUẬN 5

TRƯỜNG TRUNG CẤP NGHỀ

KỸ THUẬT CÔNG NGHỆ HÙNG VƯƠNG

GIÁO TRÌNH

TÊN MÔ ĐUN: **LẬP TRÌNH VI ĐIỀU KHIỂN**

NGÀNH, NGHỀ: **ĐIỆN TỬ CÔNG NGHIỆP**

TRÌNH ĐỘ: **TRUNG CẤP**

(Ban hành kèm theo Quyết định số: ……/QĐ-KTCNHV ngày … tháng … năm 2024 của Hiệu trưởng Trường Trung cấp nghề Kỹ thuật Công nghệ Hùng Vương)

Năm 2024

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Biên soạn giáo trình không chỉ là một hoạt động nghiên cứu khoa học, mà còn là một hành trình sáng tạo. Các giáo viên và giảng viên, dựa trên các phương pháp và nguyên tắc chung, đã khéo léo vận dụng vào điều kiện cụ thể để xây dựng nội dung giảng dạy hấp dẫn và thu hút người học.

Nhằm cung cấp tài liệu cơ sở cho đội ngũ giáo viên để tiến hành các buổi dạy học hiệu quả, cũng như hỗ trợ học sinh, sinh viên trong quá trình học tập, Khoa Điện tử đã tổ chức biên soạn cuốn giáo trình "**Lập trình Vi điều khiển**". Sản phẩm này là kết quả của sự đóng góp công sức từ nhóm giáo viên chuyên ngành tại trường Trung cấp nghề Kỹ thuật Công nghệ Hùng Vương.

Cuốn giáo trình được viết bởi đội ngũ giáo viên giàu kinh nghiệm, với sự hỗ trợ và phản biện từ các doanh nghiệp trong lĩnh vực liên quan. Dù đã cố gắng biên soạn một cách kỹ lưỡng, chúng tôi hiểu rằng cuốn sách khó tránh khỏi những khiếm khuyết. Vì vậy, chúng tôi mong nhận được ý kiến đóng góp từ bạn đọc để cuốn giáo trình này ngày càng hoàn thiện hơn trong các lần tái bản sau.

Xin trân trọng giới thiệu cuốn giáo trình tới bạn đọc!

Quận 5, ngày ... tháng ... năm ……

Tham gia biên soạn

1. Lê Bảo Khanh

2. Bùi Quốc Trưởng

3. Lê Huỳnh Quân

Contents

Contents 1
BÀI 1: Vi điều khiển AT89S52 3
1.1 Cấu trúc phần cứng 3
1.2 Chức năng chính của các thành phần bên trong Vi điều khiển 3
1.3 Sơ đồ chân vi điều khiển 89S52 4
BÀI 2: HỆ THỐNG SỐ 6
1. Hệ thống số thập phân (Decimal Number System) 6
2. Hệ thống số nhị phân (Binary Number System) 6
3. Chuyển đổi giữa hệ nhị phân (BIN) và hệ thập phân (DEC) 7
3.1. Chuyển đổi số thập phân sang số nhị phân 7
3.2. Chuyển đổi số nhị phân sang số thập phân 8
4. Mối tương quan các phép tính giữa hệ thập phân và hệ nhị phân 8
5. Hệ thống số thập lục phân (Hexa-Decimal Number System) 8
6. Chuyển đổi giữa hệ nhị phân (BIN) - thập phân (DEC) - thập lục phân (HEX) 9
6.1. Chuyển đổi từ DEC sang HEX 9
6.2. Chuyển đổi từ HEX sang DEC 9
6.3. Chuyển đổi từ HEX sang BIN 9
6.4. Chuyển đổi từ BIN sang DEC 9
7. Mã hóa số hệ thập phân: 9
8. Hệ thống số bát phân *(Octal Number System)* 9
BÀI 3: CÁC KIỂU DỮ LIỆU 10
BÀI 4: CÁC TOÁN TỬ 11
4.1 Toán tử số học (+, -, *, /, %) 13
4.2 Toán tử gán phức hợp (+=, -=, *=, /=, %=, >>, <<=, ^=, |=) 13
4.3 Toán tử tăng và giảm (++, --) 13
4.4 Toán từ quan hệ (==, !=, >, <, >=, <=) 13
4.5 Toán từ logic các điều kiện (1, &&, ||) 13
4.6 Toán tử xử lý bít (&,, ^,, <<, >>) 14
4.7 Toán tử lấy kích thước chuỗi theo byte sizeof() 14
BÀI 5: CÁC LỆNH CƠ BẢN 15
5.1 Lệnh if và else 15
5.2 Lệnh lặp while 15
5.3 Lệnh lặp do-while 16
5.4 Lệnh lặp for 16
5.5 Lệnh rẽ nhánh break 16
5.6 Lệnh continue 16
5.7 Lệnh rẽ nhánh goto: 17
5.8 Lệnh switch case 17

BÀI 6: TRÌNH BIÊN DỊCH KEIL C 18
6.1 Phần mở rộng của trình biên dịch KeilC 18
6.2 Các từ khoá 18
6.3 Bộ nhớ chương trình ROM 18
6.4 Khai báo biến và hằng số 18
6.5 Các BIT chứa chức năng đặc biệt 18
6.6 Định nghĩa các biến 18
6.7 Con trỏ dữ liệu 18
6.8 Khai báo mảng 19
6.9 Khai báo chương trình con phục vụ ngắt 19
6.10 Cấu trúc của chương trình C 19
6.11 Các thành phần của chương trình C 20
6.12 File thư viện cho họ AT89X52 20
BÀI 7: TIMER và COUNTER 21
7.1 Thanh ghi TMOD (Timer Mode Register) 21
7.2 Thanh ghi TCON (Timer Control Register) 22
7.3 Thanh ghi IE (Interrupt Enable Register) 23

BÀI 1: Vi điều khiển AT89S52

Vi điều khiển AT89S52 là một vi điều khiển thuộc họ 8051 do Atmel sản xuất (nay thuộc Microchip), dựa trên kiến trúc 8-bit. Nó được gọi là "máy tính trên một chip" vì chứa các thành phần cần thiết để thực hiện các chức năng tính toán và điều khiển.

1.1 Cấu trúc phần cứng

Vi điều khiển 89S52 chứa các thành phần cơ bản như sau:

- **CPU 8-bit**: Xử lý dữ liệu 8-bit và các lệnh dựa trên kiến trúc của 8051.
- **Bộ nhớ chương trình (ROM/Flash):** 8 KB bộ nhớ Flash, có thể lập trình và xóa tới 1000 lần, dùng để lưu chương trình điều khiển.
- **Bộ nhớ dữ liệu (RAM):** 256 byte bộ nhớ RAM để lưu trữ dữ liệu tạm thời.
- **Bộ nhớ ngoài:** Hỗ trợ mở rộng với bộ nhớ ngoài lên đến 64 KB cho cả chương trình và dữ liệu.
- **Các cổng I/O:** 32 chân vào/ra (I/O) chia thành 4 cổng (Port 0, Port 1, Port 2, Port 3), mỗi cổng 8 bit.
- **Ba bộ định thời (Timers):** Tích hợp 3 bộ Timer/Counter 16-bit (Timer 0, Timer 1, và Timer 2), giúp đếm sự kiện hoặc tạo ra độ trễ chính xác.
- **Giao tiếp nối tiếp (UART):** Cho phép truyền và nhận dữ liệu nối tiếp thông qua chuẩn UART.
- **Hai ngắt ngoài:** INT0 và INT1 là hai ngắt ngoài kích hoạt thông qua các chân ngoại vi.
- **Bộ dao động (Oscillator):** Có thể hoạt động với tần số lên đến 33 MHz, với mạch dao động ngoài hoặc dao động thạch anh.
- **Bộ điều khiển ngắt:** Hỗ trợ 6 nguồn ngắt với 2 mức ưu tiên, cho phép quản lý hiệu quả các sự kiện ngắt.

1.2 Chức năng chính của các thành phần bên trong Vi điều khiển

1.2.1 Bộ xử lý trung tâm (CPU):

- Là phần "não" của vi điều khiển, chịu trách nhiệm thực hiện các lệnh của chương trình, xử lý dữ liệu và điều khiển các thành phần khác.

1.2.2 Bộ nhớ chương trình (ROM/Flash):

- Lưu trữ chương trình điều khiển (phần mềm) được nạp vào vi điều khiển. Bộ nhớ này thường là loại không biến đổi, nghĩa là dữ liệu sẽ không bị mất khi ngắt nguồn điện.
- Loại phổ biến là bộ nhớ **Flash**.

1.2.3 Bộ nhớ dữ liệu (RAM):

- Dùng để lưu trữ dữ liệu tạm thời trong quá trình vi điều khiển thực thi chương trình. Khi tắt nguồn, dữ liệu trong RAM sẽ bị mất.

1.2.4 Các cổng vào/ra (I/O Ports):

- Cho phép vi điều khiển giao tiếp với thế giới bên ngoài, điều khiển các thiết bị ngoại vi như cảm biến, động cơ, đèn LED, và nhận dữ liệu từ các thiết bị khác như nút nhấn, cảm biến, bàn phím,...
- Các cổng này có thể là cổng số (Digital) hoặc cổng tương tự (Analog).

1.2.5 Bộ định thời và bộ đếm (Timers/Counters):

- Bộ định thời có thể dùng để tạo ra các độ trễ chính xác, trong khi bộ đếm đếm số sự kiện hoặc xung tín hiệu.

1.2.6 Giao tiếp nối tiếp (Serial Interfaces):

- Cho phép vi điều khiển giao tiếp với các thiết bị khác thông qua các chuẩn giao tiếp như UART, SPI, I2C,... để truyền/nhận dữ liệu một cách tuần tự.

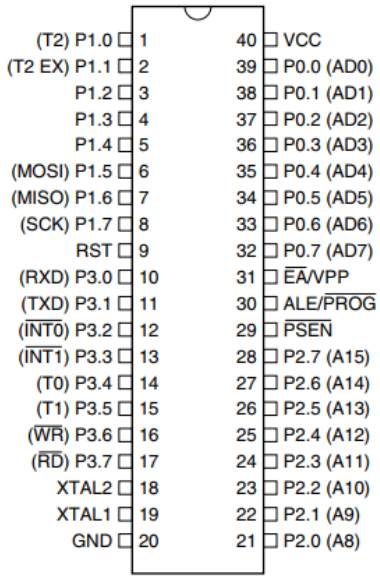
1.2.7 Bộ chuyển đổi tín hiệu số-tương tự (ADC):

- Chuyển đổi tín hiệu tương tự (Analog) từ các cảm biến thành tín hiệu số (Digital) để vi điều khiển có thể xử lý được.

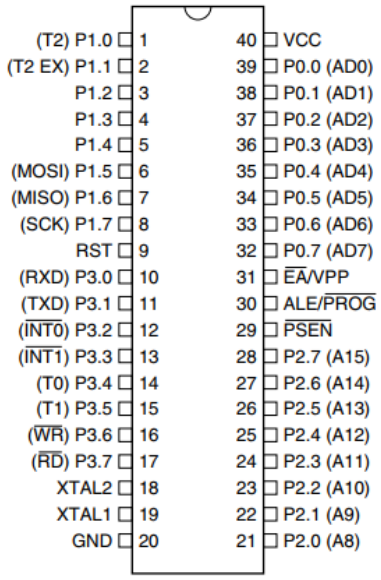
1.2.8 Bộ ngắt (Interrupts):

- Cho phép vi điều khiển ngừng thực hiện công việc hiện tại và chuyển sang xử lý một sự kiện khẩn cấp (ngắt) mà không phải kiểm tra liên tục.

1.3 Sơ đồ chân vi điều khiển 89S52



Chân	Chức năng
❖ **Chân nguồn**	
40 – VCC	Cấp nguồn dương cho vi điều khiển (+5V).
20 – GND	Chân nối đất, nguồn âm cho vi điều khiển.
❖ **Chân dao động (Crystal/Clock)**	
19 - XTAL1 18 - XTAL2	Chân kết nối với thạch anh hoặc bộ dao động ngoài (*)
❖ **Port 0**	
32 – P0.0 (A0/D0) 33 – P0.1 (A1/D1) 34 – P0.2 (A2/D2) 35 – P0.3 (A3/D3) 36 – P0.4 (A4/D4) 37 – P0.5 (A5/D5) 38 – P0.6 (A6/D6) 39 – P0.7 (A7/D7)	• Khi sử dụng Port 0 lưu ý nên mắc thêm điện trở kéo lên (Pull-up Resistor). • Các chân Port 0 có chức năng hai chiều vào/ra (I/O) thông thường ký hiệu P0.0 đến P0.7 • Khi sử dụng truy cập bộ nhớ ngoài các chân Port 0 biến thành bus dữ liệu (D0 đến D7) và bus địa chỉ byte thấp (A0 đến A7) kết hợp địa chỉ byte cao (A8 đến A15) là các chân Port 2
❖ **Port 1**	
1 – P1.0 (T2) 2 – P1.1 (T2EX) 3 – P1.2 4 – P1.3 5 – P1.4 6 – P1.5 7 – P1.6 8 – P1.7	• Bên trong VĐK các chân của Port 1 đều có **điện trở kéo lên nội bộ**, vì vậy chúng có thể duy trì trạng thái logic ổn định mà không cần thêm điện trở kéo lên bên ngoài. • Các chân Port 1 có chức năng hai chiều vào/ra (I/O) thông thường ký hiệu P1.0 đến P1.7 Một số chân có thể có chức năng khác như: • **T2**: chân **đầu vào** cho **Timer/Counter 2** khi vi điều khiển được cấu hình để sử dụng bộ đếm ngoài (chế độ đếm xung từ bên ngoài). • **T2EX**: chân này được sử dụng để điều khiển hoạt động của Timer 2
❖ **Port 2**	
21 – P2.0 (A8) 22 – P2.1 (A9) 23 – P2.2 (A10) 24 – P2.3 (A11) 25 – P2.4 (A12) 26 – P2.5 (A13) 27 – P2.6 (A14) 28 – P2.7 (A15)	• Có **điện trở kéo lên nội bộ**. • Các chân Port 2 có chức năng hai chiều vào/ra (I/O) thông thường ký hiệu P2.0 đến P2.7 • Khi sử dụng truy cập bộ nhớ ngoài các chân Port 2 biến thành bus địa chỉ byte cao (A8 đến A15) kết hợp với địa chỉ byte thấp là (A0 đến A8) là các chân port 0.
❖ **Port 3**	
10 – P3.0 (RXD) 11 – P3.1 (TXD) 12 – P3.2 (INT0) 13 – P3.3 (INT1) 14 – P3.4 (T0) 15 – P3.5 (T1) 16 – P3.6 (WR)	• Có **điện trở kéo lên nội bộ**. • Các chân Port 3 có chức năng hai chiều vào/ra (I/O) thông thường ký hiệu P3.0 đến P3.7 • Nhưng mỗi chân của Port 3 đều có các chức năng đặc biệt:



17 – P3.7 (RD)	• **RXD (**Receiver Data Input) nhận dữ liệu trong giao tiếp nối tiếp UART. • **TXD** (Transmitter Data Output) truyền dữ liệu trong giao tiếp nối tiếp UART. • **INT0** (External Interrupt 0) ngắt ngoài 0 • **INT1 (**External Interrupt 1): Ngắt ngoài 1 • **T0**: Đầu vào xung của Timer0/Counter0. • **T1**: Đầu vào xung của Timer1/Counter1. • **WR:** Tín hiệu ghi dữ liệu vào bộ nhớ ngoài. • **RD**: Tín hiệu đọc dữ liệu từ bộ nhớ ngoài.
❖ **Các chân điều khiển bộ nhớ**	
29 - PSEN (Program Store Enable)	Tín hiệu cho phép đọc bộ nhớ chương trình ngoài (ROM ngoài). Khi vi điều khiển thực hiện lệnh từ bộ nhớ ngoài, chân **PSEN** sẽ được kích hoạt (mức thấp).
30 - ALE (Address Latch Enable)	Chân này được sử dụng để **chốt địa chỉ** khi kết nối với bộ nhớ ngoài. **ALE** phát xung địa chỉ thấp từ Port 0 và kích hoạt bộ chốt địa chỉ bên ngoài.
31- EA (External Access)	Chân này dùng để quyết định vi điều khiển sẽ làm việc với bộ nhớ trong hay ngoài. • **1**: bộ nhớ trong (Flash ROM). • **0**: bộ nhớ ngoài (External ROM).

❖ **(*) Chân 18 (XTAL2) và 19 (XTAL1)**

– Kết nối với thạch anh để tạo xung nhịp cho vi điều khiển hoạt động. Thông thường, tần số của thạch anh phổ biến là 11.0592 MHz (phù hợp cho việc tạo ra tốc độ baud trong giao tiếp nối tiếp UART) hoặc các giá trị khác như 12 MHz, 16 MHz,...

– Hai tụ điện mắc song song với các chân XTAL1 và XTAL2 để ổn định dao động. Giá trị tiêu chuẩn của các tụ này thường là 22pF.

BÀI 2: HỆ THỐNG SỐ

Hệ thống số là hệ thống biểu diễn các số bằng cách sử dụng tập hợp *các **chữ số*** (hay còn gọi là các *ký hiệu, ký số - Digits*). Số lượng *các chữ số* dùng trong hệ thống số gọi là các ***cơ số*** của hệ thống số đó.

1. Hệ thống số thập phân (Decimal Number System)

- Hệ thống số thập phân sử dụng **10 chữ số** gồm: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Hệ thống số thập phân có ***cơ số*** là 10, còn được gọi là hệ thập phân, hệ 10 hay hệ **DEC**
- Hệ thập phân là một ***hệ thống số theo vị trí*** vì trong đó ***giá trị của một chữ số*** phụ thuộc vào vị trị đứng của nó và trọng số của hệ DEC là $\mathbf{10^n}$.
- Ví dụ: số 752.25 ở hệ thập phân ký hiệu 752.25_{10} phân tích gồm:

  $752.25 = 700 + 50 + 2 + 0.2 + 0.05$

Vị trí hàng	Trăm	Chục	Đơn vị	Phần chục	Phần trăm
752.25	7	5	2	2	5
Dạng luỹ thừa	7×10^{2}	5×10^{1}	2×10^{0}	2×10^{-1}	5×10^{-2}
Vị trí các chữ số	thứ 2	thứ 1	thứ 0	thứ -1	thứ -2

- Trong đó:
  + ***Trọng số*** $\mathbf{10^n}$ thể hiện ***vị trí*** của chữ số.
  + ***Chữ số 7*** có giá trị lớn nhất gọi là ***chữ số có nghĩa lớn nhất*** (MSD-***M**ost **S**ignificant **D**igit)*
  + ***Chữ số 5*** có giá trị nhỏ nhất gọi là ***chữ số có nghĩa nhỏ nhất*** (LSD-***L**east **S**ignificant **D**igit)*
- **Các phép toán trong hệ số thập phân**

• Phép cộng	• Phép trừ	• Phép nhân	• Phép chia
24 + 12 36	324 - 168 156	36 x 10 00 + 36 360	432 | 12 - 36 | 36 72 - 72 0

2. Hệ thống số nhị phân (Binary Number System)

- Hệ thống số nhị phân sử dụng **2 chữ số** gồm: 0, 1.
- Hệ thống số nhị phân có ***cơ số là 2***, còn được gọi là hệ nhị phân, hệ 2 hay hệ **BIN**
- Hệ nhị phân cũng là một hệ thống số theo vị trí và trọng số của hệ nhị phân là $\boldsymbol{2^n}$.
- Ví dụ: số $\mathbf{11010011_2}$ trong hệ nhị phân được phân tích gồm:

Vị trí các chữ số	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
11010011	1	1	0	1	0	0	1	1
Dạng luỹ thừa	1×2^{7}	1×2^{6}	0×2^{5}	1×2^{4}	0×2^{3}	0×2^{2}	1×2^{1}	1×2^{0}

- Trong đó:
  + Mỗi ***chữ số*** trong số nhị phân gọi là ***một bit*** và ***tên mỗi bit*** được gắn theo thứ tự từ phải sang trái gồm bit 0; bit 1; bit 2 … bit 7.
  + ***Bit 7*** có giá trị lớn nhất được gọi là bit có nghĩa nhất (MSB -***M**ost **S**ignigicant **B**it*)
  + ***Bit 0*** có giá trị nhỏ nhất được gọi là bit có nghĩa ít nhất (LSB - ***L**east **S**ignigicant **B**it*).
- Quy ước:
  + Số nhị phân có 4 bit gọi là 1 nipple.
  + Số nhị phân có 8 bit gọi là 1 byte.
  + Số nhị phân có 16 bit được gọi là word.

+ Số nhị phân có 32 bit được gọi là double word.
+ Số nhị phân có 64 bit được gọi là quadword.

$$TB(tetra\ B) \xrightarrow{\times 2^{10}} GB\ (giga\ B) \xrightarrow{\times 2^{10}} MB\ (mega\ B) \xrightarrow{\times 2^{10}} KB\ (kilo\ B) \xrightarrow{\times 2^{10}} B(Byte) \xrightarrow{\times 8} bit$$

Ví dụ: 01 : 2 bit.
1001 : 4 bit = 1 nipple.
10011010 : 8 bit = 1 byte.
10011010 10011010: 16 bit = 2 byte = 1 word

– **Các phép tính trên số nhị phân**

a) Phép cộng ▪ 0 + 0 = 1 ▪ 0 + 1 = 1 ▪ 1 + 0 = 1 ▪ 1 + 1 = 0 (nhớ 1)	Ví dụ: 11000 + 1100 = ? 11000 + 1100 100100
b) Phép trừ: ▪ 0 – 0 = 0 ▪ 0 – 1 = 1 (mượn 1) ▪ 1 – 0 = 1 ▪ 1 – 1 = 0	Ví dụ: 100100 – 1101 = ? 100100 - 1101 10111
c) Phép nhân ▪ 0 x 0 = 0 ▪ 0 x 1 = 0 ▪ 1 x 0 = 0 ▪ 1 x 1 = 1	Ví dụ: 100100 x 1100 = ? 100100 x 11**00** 100100 + 100100 110110**00**
d) Phép chia: ▪ 0 : 0 = 0 ▪ 1 : 1 = 1	Ví dụ: 110110000 : 1100 =? 110110**00** | 1100 – 1100 | 1001**00** **1**100 0

3. Chuyển đổi giữa hệ nhị phân (BIN) và hệ thập phân (DEC)

3.1. Chuyển đổi số thập phân sang số nhị phân

– Nguyên tắc chuyển đổi một số thập phân bất kỳ thành số nhị phân bằng cách lấy số thập phân ***thực hiện phép chia 2 lấy dư*** liên tiếp cho đến khi thương bằng 0. Số nhị phân thu được là các số dư.

Thí dụ: đổi số thập phân là 12 thành số nhị phân ta làm như sau:

Chia liên tiếp cho 2: 0 ←:2– 1 ←:2– 3 ←:2– 6 ←:2– 12_{10}
(dư) (dư) (dư) (dư)
Số dư tương ứng: 1 1 0 0

Kết quả: $(12)_{10} = (1100)_2$

Tương tự $(24)_{10} = (11000)_2$

3.2. Chuyển đổi số nhị phân sang số thập phân

– Nguyên tắc chuyển đổi một số nhị phân bất kỳ thành số thập phân bằng cách tính **tổng các tích** của mỗi ***chữ số nhị phân*** với ***trọng số tương ứng***.

– Ví dụ:

$(10\ 0100)_2 = (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$

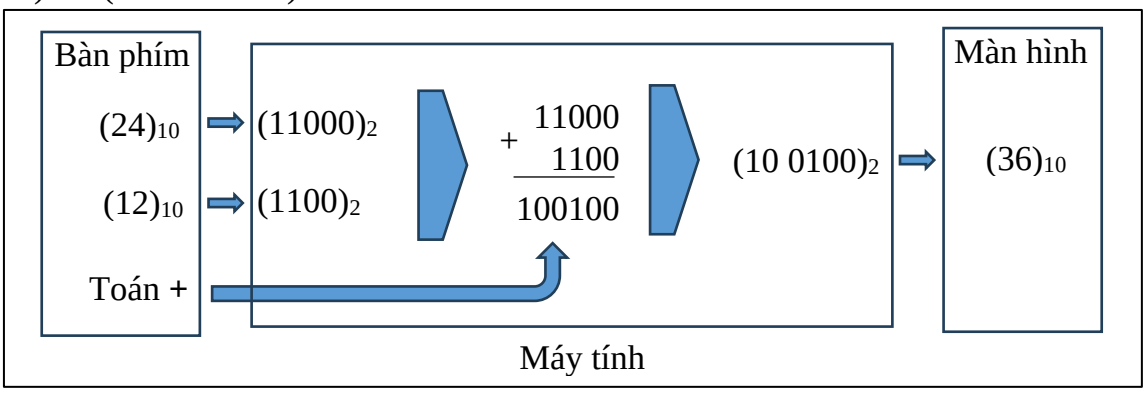
$= 32 + 0 + 0 + 4 + 0 + 0$

$= (36)_{10}$

Tương tự : $(100\ 1000)_2 = (72)_{10}$ và $(1\ 1011\ 0000)_2 = (432)_{10}$

4. Mối tương quan các phép tính giữa hệ thập phân và hệ nhị phân

- $(12)_{10} = (1100)_2$
- $(24)_{10} = (11000)_2$
- $(36)_{10} = (10\ 0100)_2$
- $(72)_{10} = (100\ 1000)_2$
- $(432)_{10} = (1\ 1011\ 0000)_2$



5. Hệ thống số thập lục phân (Hexa-Decimal Number System)

– Hệ thống số thập lục phân sử dụng **16 chữ số** gồm: 0, 1, 2, 3, 4, 5, 6, 7, 9, A, B, C, D, E, F.

– Hệ thống số thập lục phân có ***cơ số là 16*** còn được gọi là hệ 16, hệ thập lục phân hay hệ **HEX**.

– Hệ thập phân lục phân cũng là một hệ thống số theo vị trí và trọng số của hệ HEX là 16^n.

– Ví dụ: số $2F0.4_{16}$ trong hệ thập lục phân được phân tích gồm:

Vị trí các chữ số	thứ 2	thứ 1	thứ 0	thứ -1
2F0.4	2	F	0	4
Dạng luỹ thừa	$2\times16^{\mathbf{2}}$	$F\times16^{\mathbf{1}}$	$0\times16^{\mathbf{0}}$	$2\times16^{\mathbf{-1}}$

– **Các phép toán trong hệ số thập lục phân**

- Phép cộng

$$\begin{array}{r} 18 \\ +\ C \\ \hline 24 \end{array}$$

- Phép trừ

$$\begin{array}{r} 144 \\ -\ A8 \\ \hline 9C \end{array}$$

- Phép nhân

$$\begin{array}{r} 24 \\ \times\ A \\ \hline 168 \end{array}$$

- Phép chia

$$\begin{array}{r|l} 1B0 & C \\ \hline 0 & 24 \end{array}$$

– Khác với hệ thập phân, phép chia trong các hệ số khác như nhị phân, thập lục phân có thể phức tạp hơn và thường được thực hiện bằng cách chuyển đổi các số về hệ thập phân, thực hiện phép chia ở hệ thập phân, rồi chuyển kết quả về lại hệ số ban đầu.

6. Chuyển đổi giữa hệ nhị phân (BIN) - thập phân (DEC) - thập lục phân (HEX)

Nhị phân **(BIN)**	Thập phân **(DEC)**	Thập lục phân (HEX)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

6.1. Chuyển đổi từ DEC sang HEX

Chia liên tiếp cho 16: $0 \xleftarrow{:16} 9 \xleftarrow{:16} 156_{10}$

dư dư

Số dư tương ứng: 9 12 = C

Kết quả: $432_{10} = 9C_{16}$

6.2. Chuyển đổi từ HEX sang DEC

$9C_{16} = 9\times16^1 + C\times16^0$

$= 9\times16 + 12\times1 = 156_{10}$

6.3. Chuyển đổi từ HEX sang BIN

$9C_{16} = 1001\ 1100_2$

6.4. Chuyển đổi từ BIN sang DEC

$1001\ 1100_2 = 1\times2^7 + 1\times2^4 + 1\times2^3 + 1\times2^2$

$= 128 + 16 + 8 + 4 = 156$

7. Mã hóa số hệ thập phân:

BCD (Binary-Coded Decimal) là một phương pháp **mã hóa** số thập phân trong đó mỗi chữ số thập phân được biểu diễn bằng một chuỗi bit cố định, thường là **4 bit**. Nói cách khác, BCD là một cách để chuyển đổi các số từ hệ thập phân (cơ số 10) sang hệ nhị phân (cơ số 2) một cách trực tiếp, mà không cần chuyển đổi qua một hệ số trung gian.

Bảng mã BCD	
Nhị phân **(BIN)**	Thập phân **(DEC)**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10

Ví dụ: Số thập phân mã BCD

32 ⇒ **0011** 0010

2**5**7 ⇒ 0010 **0101** 0111

- Mỗi một chữ số thập lục phân được biểu diễn bởi một nhóm nhị phân 4 bit.

  Ví dụ: $1001\ 1110\ 1010\ 0111_B = 9E\ A\ 7_H$
- Khi chuyển đổi số **hex** ra số nhị phân thì ta làm ngược lại, chuyển đổi từng chữ số của số hex thành các **nhóm nhị phân 4 bit** có giá trị tương ứng.
- *Ví dụ:* $6C3D_H = 0110\ 1100\ 0011\ 1101_B$

8. Hệ thống số bát phân *(Octal Number System)*

- Hệ bát phân có 8 *chữ số* (ký hiệu) gồm: 0, 1, 2, 3, 4, 5, 6, 7.
- Hệ thống số bát phân có *cơ số* là 8, còn được gọi là *hệ 8* hay *hệ bát phân*.
- Hệ bát phân có *trọng số* là 8^n.
- Mỗi một chữ số bát phân được biểu diễn bởi một nhóm nhị phân 3 bit.

BÀI 3: CÁC KIỂU DỮ LIỆU

Kiểu dữ liệu là một khái niệm cơ bản trong lập trình, nó xác định loại dữ liệu mà một biến có thể chứa và các phép toán có thể thực hiện trên nó. Kiểu dữ liệu giúp máy tính hiểu được cách lưu trữ và xử lý thông tin một cách hiệu quả.

TT	Kiểu dữ liệu	Số bit	Giới hạn giá trị
1.	bit	1	0 hoặc 1
2.	unsigned char	8	Từ 0 đến 255
3.	signed char	8	Từ -128 đến 127
4.	unsigned int	16	Từ 0 đến 65535
5.	signed int	16	Từ -32768 đến 32767
6.	unsigned long	32	Từ 0 đến 4294967295
7.	signed long	32	Từ -2147483648 đến 2147483647
8.	float	32	$\pm 1.175494E^{-38}$ to $\pm 3.402823E^{+38}$
9.	sbit	1	0 hoặc 1
10.	sfr	8	0-255

Trong lập trình vi điều khiển sử dụng Keil C, **sbit** và **bit** là hai kiểu dữ liệu đặc biệt dùng để làm việc với các bit trong vi điều khiển. Mặc dù cả hai đều đại diện cho một bit dữ liệu, nhưng chúng có những điểm khác biệt quan trọng về cách sử dụng và mục đích.

❖ **Sbit**

- Định nghĩa: sbit là viết tắt của "special bit". Nó được sử dụng để trực tiếp truy cập vào một bit cụ thể trong một thanh ghi (register) của vi điều khiển.
- Thường được sử dụng khi cần điều khiển trực tiếp các chân I/O hoặc các bit flag trong các thanh ghi của vi điều khiển.
- Ưu điểm: cho phép trực tiếp truy cập và thao tác với các bit một cách nhanh chóng và hiệu quả.
- Nhược điểm: phụ thuộc vào kiến trúc vi điều khiển: Cách khai báo và sử dụng sbit có thể khác nhau giữa các loại vi điều khiển và chỉ giới hạn trong việc truy cập các bit trong các thanh ghi đã được định nghĩa sẵn.
- Ví dụ:

```
sbit LED = P1^0; // Điều khiển LED ở chân P1.0
```

❖ **bit**

- Định nghĩa: bit là một kiểu dữ liệu đơn giản chỉ đại diện cho một bit (0 hoặc 1).
- Thường được sử dụng khi cần lưu trữ một giá trị logic đơn giản (lưu trữ các giá trị true/false hoặc các trạng thái bật/tắt) hoặc thực hiện các phép toán bit (AND, OR, XOR…)
- Ưu điểm: có thể được sử dụng trong nhiều ngữ cảnh khác nhau và cú pháp đơn giản.
- Nhược điểm: không trực tiếp truy cập phần cứng, cần phải gán giá trị của bit vào một thanh ghi hoặc biến khác để điều khiển phần cứng.
- Ví dụ:

```
bit flag;
flag = 1; // Thiết lập flag thành 1
```

```
// Điều khiển LED nhấp nháy bằng sbit
sbit LED = P1^0;
void main() {
  while(1) {
    LED = 1; // Bật LED
    delay_ms(500);
    LED = 0; // Tắt LED
    delay_ms(500);
  }
}
```

```
// Sử dụng bit để kiểm tra một bit trong một byte
unsigned char data = 0x5A;
bit bit3 = (data & 0x08) >> 3; // Kiểm tra bit thứ 3
```

BÀI 4: CÁC TOÁN TỬ

Các toán tử là thành phần quan trọng trong lập trình, để lập trình thì chúng ta cần phải hiểu rõ ràng chức năng của các loại toán tử

TT	Toán tử	Chức năng của toán tử	Ví dụ
	Gán		
1.	=	**Gán** Sử dụng để gán một giá trị nào đó cho một biến	int z; *// khai báo z là biến kiểu int* int x = 3; *// biến x kiểu int và gán x = 3* int y = 5; *// biến x kiểu int và gán y = 5*
	Số học		
2.	+	Cộng	int x = 3; int y = 5; int z; z = x + y; *// biến z gán giá trị của x + y* *// kết quả z = 8*
3.	+=	Cộng và gán	int x = 3; int y = 5; int z; x = x + y; *// biến x gán giá trị của x + y* x += y; *// viết gọn và kết quả x = 8*
4.	-	Trừ	int x = 3; int y = 5; int z; x = y - x; *// biến x gán giá trị của y – x* *// kết quả x = 2*
5.	-=	Trừ và gán	x = y - x; viết gọn x -= y;
6.	*	Nhân	
7.	*=	Nhân và gán	x = x * y; viết gọn x *= y
8.	/	Chia	int x = 3; int y = 5; • int z = y/x; *// y/x = 5/3 = 1,666…* *vì khai báo z là int nên z = 2 làm tròn đến số nguyên gần nhất.* • float z = y/x; *// y/x = 5/3 = 1,666…* *vì khai báo z là float nên z = 1,6667*
9.	/=	Chia và gán.	x = x / y; viết gọn x /= y;
10.	%	Module hoặc toán tử lấy dư. thực hiện phép chia hai số nguyên và trả về **số dư** của phép chia đó.	int x = 3; int y = 5; int z; z = y % x; *// kết quả 5/3 bằng 1 dư 2* *// kết quả z = 2*
11.	%=	Chia lấy số dư và gán	x = x % y; viết gọn x % = y;
12.	++	Tăng giá trị lên 1 đơn vị • z++; dạng hậu tố • ++z; dạng tiền tố	int x = 3; int y, z; • y = x++; *// biến sẽ tăng giá trị trước khi sử dụng trong biểu thức y = 3 và x = 4* • z = ++x; *// biến sử dụng trong biểu thức trước khi tăng giá trị x = 4 và z = 4*
13.	--	Giảm	
	Toán tử so sánh		
14.	==	**Bằng** được sử dụng để so sánh hai giá trị xem chúng có bằng nhau hay không.	• Nếu x = y; // kết quả true = 1 • Nếu x ≠ y; // kết quả false = 0

15.	!=	**Không bằng** hay khác	
16.	>	Lớn hơn	
17.	>=	Lớn hơn hay bằng	
18.	<	Nhỏ hơn	
19.	<=	Nhỏ hơn hay bằng	
	Toán tử Logic		
20.	&&	kết hợp hai biểu thức logic và chỉ trả về **true** khi cả hai biểu thức đều **true**, ngược lại là **false**	(Biểu thức 1) && (Biểu thức 2); (True) && (True); *// kết quả trả về true*
21.	||	kết hợp hai biểu thức logic và chỉ trả về **true** khi cả hai biểu thức đều **false**, ngược lại là **true**	(Biểu thức 1) || (Biểu thức 2); (False) && (False); *// kết quả trả về false*
22.	!	NOT (phủ định logic) dùng để đảo ngược giá trị logic của một biểu thức.	!(Biểu thức); • Nếu (Biểu thức = true) trả về false • Nếu (Biểu thức = false) trả về true
	Toán tử bitwise		
23.	&	Thực hiện phép AND trên từng bit. Nếu hai bit tương ứng vị trí đều là 1, thì bit tương ứng trong kết quả sẽ là 1, ngược lại sẽ là 0	int x = 3; *// x = 0011* int y = 5; *// y = 0101* int z = x & y; *// z = 0001 → z = 1*
24.	&=	**&** và **gán**	x = x & y; viết gọn x **&=** y;
25.	|	Thực hiện phép OR trên từng bit. Nếu hai bit tương ứng vị trí đều là 0, thì bit kết quả sẽ là 0 ngược lại bằng 1.	int x = 5; *// x = 0101* int y = 3; *// y = 0011* int z = x | y; *// z = 0111 → z = 7*
26.	|=	| và **gán**	x = x | y; viết gọn x **|=** y;
27.	^	Ex-Or (XOR - Exclusive OR). Toán tử so sánh từng cặp bit. • Nếu hai bit tương ứng ***khác nhau***, bit kết quả sẽ là 1. • Nếu hai bit tương ứng ***giống nha***u, bit kết quả sẽ là 0.	int x = 3; *// x = 0011* int y = 5; *// y = 0101* int z = x ^ y; *// z = 0110 → z = 6*
28.	^=	^ và gán.	x = x ^ y; viết gọn x ^= y;
29.	~	bù 1 hay đảo bit	int x = 3; *// x = 0011* int y = ~x; *// y = 1100*
30.	>> x >> y	Toán tử dịch phải x: số cần dịch chuyển y: số lượng bit cần dịch chuyển	*dịch x sang phải 2 bit* int x = 3; *// x = 0000 0011* int y = x >> 2; *// y = 0000 1100*
31.	>>=	Toán tử dịch phải và gán	x = x >> y viết gọn x >>= y
32.	<<	Toán tử dịch trái	
33.	<<=	Toán tử dịch trái và gán	x = x << y viết gọn x <<= y
34.	->	Toán tử con trỏ cấu trúc	

	Khác		
35.	*	Toán tử truy xuất gián tiếp, đi trước con trỏ	
36.	sizeof	Xác định kích thước theo byte của toán tử	

4.1 Toán tử số học (+, -, *, /, %)

- Có 5 toán tử để thực hiện các phép toán cộng, trừ, nhân, chia và chia lấy phần dư.
- Ví dụ 1: A = 24; B = A % 5; → B = 4
  → Gán A bằng 24, B gán số dư của A chia cho 5, kết quả B bằng 4
- Ví dụ 2: tách số A = 123 thành từng con số đơn vị, chục, trăm gán cho ba biến X, Y, Z
  - A = 123;
  - X = A%10 → **X = 3**
  - A = A/10; → A = 12
  - Y = A%10; → **Y = 2**
  - Z = A/10; → **Z = 1**
- Ví dụ 3: tách số A = 1234 thành từng con số đơn vị, chục, trăm, ngàn gán cho 4 biển X, Y, Z, V
  - A = 1234; X = A%10; → **X = 4**
  - A = A/10; → A = 123
  - Y = A%10; → **Y = 3**
  - A = A/10; → A = 12
  - Z = A%10; → **Z = 2**
  - V = A/10; → **V = 1**

4.2 Toán tử gán phức hợp (+=, -=, *=, /=, %=, >>, <<=, ^=, |=)

- Tổng quát: [biến += giá trị] tương đương với [biến = biến + giá trị].
- Ví dụ:
  - A += 5; tương đương với A = A +5;
  - A/= 5; tương đương A = A/5;
  - B *= X + 1; tương đương B = B * (X+1);

4.3 Toán tử tăng và giảm (++, --)

- Tổng quát: [biến += giá trị] tương đương với [biến = biến + giá trị]
- Ví dụ:
  - A = 5; Gán A bằng 5
    A++; tương đương với A = A +1 hay A+=1; kết quả A bằng 6
  - B = 3; gán B bằng 3
    A=++B; kết quả A bằng 4, B bằng 4
  - B = 3; A = B++; gán B bằng 3 kết quả A bằng B và bằng 3, B tăng lên 1 bằng 4
- Sự khác nhau là "++" đặt trước thì tính trước rồi mới gán, đặt sau thì gán trước rồi mới tính.

4.4 Toán từ quan hệ (==, !=, >, <, >=, <=)

- Các toán tử quan hệ dùng để so sánh các biểu thức, kết quả so sánh là **đúng** hoặc **sai.**
- Các toán tử trên tương ứng là bằng, khác, lớn hơn, nhỏ hơn, lớn hơn hay bằng, nhỏ hơn hay bằng.
- Ví dụ: if (X<100) X+=1;
  → Lệnh trên kiểm tra nếu X còn nhỏ hơn 100 thì tăng X lên 1.

4.5 Toán từ logic các điều kiện (1, &&, ||)

- Các toán tử trên tương đương là NOT, AND và OR.
- Ví dụ:
  - !true sẽ cho kết quả là false
  - ((5==5) && (6>4)) and hai điều kiện lại với nhau và kết quả là true.

4.6 Toán tử xử lý bít (&,, ^,, <<, >>)

- Các toán tử xử lý bít với bit, các toán từ trên tương đương là AND. OR, XOR, NOT, SHL (dịch trái), SHR (dịch phải).
- Ví dụ 1:

  A = 0x77; //gán A = 0111 0111B
  B = 0xC9; //gán B = 1100 1001B
  X = A & B; // X bằng A and với B, kết quả X = 01000001B = 0x41
  Y = A | B; // Y bằng A or với B, kết quả Y = 1111 1111B = 0xFF
  Z = A^ B; // Z bằng A xor với B, kết quả Z = 1011 1110B = 0xBE
  W = -A: // W bằng not A, kết quả W = 1000 1000B = 0x88
- Ví dụ 2:

  A=0x01; //gán A = 0000 0001B
  A= (A <<1); //dịch A sang trái 1 bit, kết quả A = 0000 0010B.
  A= (A <<1); //dịch A sang trái 1 bit, kết quả A = 0000 0100B.
  Khi dịch trái thì bit bên trái mất, bit 0 thêm vào bên phải.
- Ví dụ 3:

  A = 0x81; //gán A = 1000 0001B
  A = (A >>1); //dịch A sang phải 1 bit, kết quả A = 0100 0000B.
  Khi dịch phải thì bit bên phải mất, bit 0 thêm vào bên trái.
- Ví dụ 4:

  A = 0x00; //gán A = 0000 0000B
  A = (A << 1) + 0x01; //dịch A sang trái 1 bit rồi cộng với 1 → A = 0000 0001B.
  A = (A << 1) + 0x01; //dịch A sang trái 1 bit rồi cộng với 1 → A=0000 0011B.
  Khi dịch trái và cộng với 1 thì có chức năng đẩy số 1 thêm vào bên phải, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy 8 bit 1.
- Ví dụ 5:

  A = 0x00; // gán A = 0000 0000B
  A = (A >>1) + 0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000 → A = 1000 0000B.
  A = (A >>1) + 0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000 → A = 1100 0000B
  Khi dịch phải và cộng với 0x80 thì có chức năng đẩy số 1 thêm vào bên trái, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy 8 bit 1.
- Ví dụ 6:

  unsigned char X, Y; // X, Y là biển 8 bit
  unsigned int A; // A là biến 16 bit
  A = 0x1234;
  X = A; // gán X = 0x34 là gán byte thấp (8 bit thấp)
  Y = A >> 8; //dịch A sang phải 8 bit rồi gán Y → Y=0x12 có nghĩa là gán byte cao (8 bit cao)

4.7 Toán tử lấy kích thước chuỗi theo byte sizeof()

- Ví dụ 1: A = sizeof (charac); kết quả là A sẽ chứa số byte của chuỗi charac

BÀI 5: CÁC LỆNH CƠ BẢN

STT	Lệnh	Chức năng chính
1.	if-else	Điều kiện
2.	while	Lặp khi điều kiện đúng
3.	do-while	Lặp ít nhất một lần, sau đó kiểm tra điều kiện
4.	for	Lặp một số lần xác định trước
5.	break	Thoát khỏi vòng lặp
6.	continue	Bỏ qua phần còn lại của lần lặp hiện tại
7.	goto	Nhảy đến một nhãn

5.1 Lệnh if và else

- Chức năng: được sử dụng để thực hiện các lệnh khác nhau dựa trên một điều kiện cho trước.
- Cách hoạt động:
  - **if:** kiểm tra một điều kiện. Nếu điều kiện đúng thì khối lệnh bên trong if được thực hiện.
  - **else:** nếu điều kiện trong if sai thì khối lệnh bên trong else được thực hiện (nếu có).

Cú pháp:	Viết gọn	Ví dụ	Viết gọn
if (điều_kiện){ *//điều_kiện_Đúng* Lệnh al; Lệnh a2; … } **else**{ *//điều_kiện_Sai* Lệnh b1; Lệnh b2; … }	**if** (điều_kiện) Lệnh al; **else** Lệnh b1;	**if** (x == 50){ x = 1; } **else**{ x = x+1; }	**if** (x == 50) x = 1; **else** x = x+1; *// vì chỉ có 1 một lênh nên bỏ { }*
	// trường hợp không sử dụng else: **if** (điều_kiện) Lệnh al;	**if** (x == 50){ x = 1; }	**if** (x == 50) x = 1;

5.2 Lệnh lặp while

- Chức năng: thực hiện một khối lệnh nhiều lần cho đến khi một điều kiện nhất định trở thành sai
- Cách hoạt động:
  - Kiểm tra điều kiện trước khi thực hiện khối lệnh.
  - Nếu điều kiện đúng, thực hiện khối lệnh và quay lại kiểm tra điều kiện.

Cú pháp:	Viết gọn	Ví dụ	Viết gọn
while (điều_kiện){ *//điều_kiện_Đúng* Lệnh l; Lệnh 2; … }	**while** (điều_kiện) Lệnh 1;	**while** (x > 0){ x = x-1; }	**while** (x > 0) x = x-1;
while (1){ *// luôn_Đúng* Lệnh l; }	**while (1)** Lệnh l;	**while(1)**{ x = x+1; }	**while(1)** x = x+1;
		// điều kiện 1 luôn đúng nên vòng lặp này chạy vô hạn và x tăng mãi mãi	
		while(1){ ; }	while(1);
		// vòng lặp vô hạn hay dừng tại đây	

5.3 Lệnh lặp do-while

- Chức năng: Tương tự như while nhưng luôn thực hiện khối lệnh ít nhất một lần trước khi kiểm tra điều kiện.
- Cách hoạt động:
  - Thực hiện khối lệnh một lần.
  - Kiểm tra điều kiện. Nếu đúng, thực hiện lại khối lệnh và quay lại kiểm tra điều kiện.

Cú pháp:	Viết gọn	Ví dụ	Viết gọn
do { Lệnh l; Lệnh 2; … } **while** (điều_kiện)	**do** Lệnh l; **while** (điều_kiện)	**do** { x = x+10; } **while** (x < 100)	**do** x = x+10; **while** (x < 100)

5.4 Lệnh lặp for

- Chức năng: thực hiện một khối lệnh một số lần xác định trước.
- Cách hoạt động:
  - Khởi tạo biến đếm → Kiểm tra điều kiện
  - Thực hiện khối lệnh.
  - Cập nhật biến đếm.→ Quay lại Kiểm tra điều kiện.

Cú pháp:	Viết gọn	Ví dụ	Viết gọn
for (giá trị_bắt_đầu; điều_kiện_kết thúc; tăng giá_trị){ Lệnh 1; Lệnh 2; … }	**for** (giá trị_bắt_đầu; điều_kiện_kết thúc; tăng giá_trị) Lệnh 1;	do { x = x+10; } while (x < 100)	do x = x+10; while (x < 100)
		// Vòng lặp thực hiện với biến n bằng 0 cho đến khi n bằng 100 thì ngừng.	

5.5 Lệnh rẽ nhánh break

- Chức năng: được sử dụng để thoát khỏi một vòng lặp (while, do-while, for hoặc switch-case) ngay lập tức.
- Cú pháp: **break;**
- Ví dụ:

```
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            break; // Kết thúc vòng lặp khi i là 3
        }
        printf("%d ", i); // kết quả là: 1 2 đến 3 thì thoát khỏi vòng lặp
    }
```

5.6 Lệnh continue

- Chức năng: được sử dụng để kết thúc phần còn lại của vòng lặp hiện tại và chuyển sang thực hiện vòng lặp tiếp theo.
- Cú pháp: **continue;**
- Ví dụ:

```
    for (i = 1; i <= 10; i++) {
        if (i == 5) {
            continue; // Bỏ qua phần còn lại của vòng lặp khi i bằng 5 thực hiện tiếp tục
        }
        printf("%d ", i); // kết quả là: 1 2 3 4 6 7 8 9 10 bỏ 5
    }
```

5.7 Lệnh rẽ nhánh goto:

- Chức năng: sử dụng để nhảy đến một nhãn (label) được đặt trước đó trong chương trình.
- Lưu ý: Việc sử dụng goto thường không khuyến khích vì có thể làm cho code khó đọc và khó bảo trì. Nên sử dụng các cấu trúc điều khiển khác như if, while, for để thay thế.
- Ví dụ:

```
myLabel: // nhãn
printf("Giá trị của i: %d\n", i);
i++;
if (i < 5) {
   goto myLabel; // Chuyển đến nhãn myLabel
}
```

5.8 Lệnh switch case

- Chức năng: được sử dụng thực hiện các hành động khác nhau dựa trên giá trị của một biểu thức.
- Cách hoạt động:
  - So sánh giá trị của biểu thức với các giá trị trong các trường hợp case.
  - Khi tìm thấy trường hợp khớp, các lệnh bên trong trường hợp đó được thực hiện.

Cú pháp:	Viết gọn	Ví dụ	Viết gọn
switch (cmd) { Case constant1: Lệnh a1; Lệnh a2; … Break; Case constant2: Lệnh b1; Lệnh b2; … Break; Default: Lệnh c1; Lệnh c2; … Break; }	**switch** (cmd) { Case constant1: Lệnh a1; Break; Case constant1: Lệnh b1; Break; Default: Lệnh c1; Break; }	**switch** (cmd) { Case 0: printf("cmd 0"); Break; Case 1: printf("cmd 1"); Break; Default: printf("bad cmd"); Break; }	

BÀI 6: TRÌNH BIÊN DỊCH KEIL C

Keil C là một môi trường phát triển tích hợp (IDE) phổ biến được sử dụng rộng rãi trong lĩnh vực lập trình vi điều khiển. Nó cung cấp một nền tảng mạnh mẽ và dễ sử dụng để viết, biên dịch và gỡ lỗi các chương trình nhúng.Với những kiến thức cơ bản đã trình bày thì bạn có thể lập trình được cho vi điều khiển, tuy nhiên do mỗi vi điều khiển khác nhau nên ta cần phải tìm hiểu thêm các tính năng khác như phần trình bày tiếp theo.

6.1 Phần mở rộng của trình biên dịch KeilC

Trình biên dịch C51 cung cấp một số phần mở rộng cho ngôn ngữ C chuẩn ANSI. Các phần mở rộng này cung cấp hỗ trợ trực tiếp cho các thành phần của kiến trúc vi điều khiển 8051 bao gồm:

- Các vùng nhớ của vi điều khiển 8051
- Các kiểu mô hình bộ nhớ
- Các chỉ dẫn loại bộ nhớ
- Các chỉ dẫn loại biến dữ liệu bộ nhớ
- Các biến thuộc dữ liệu bịt và dữ liệu cho phép truy xuất bịt
- Các thanh ghi đặc biệt
- Con trỏ
- Các thuộc tính hàm

6.2 Các từ khoá

Từ khóa: Để tránh lỗi trong lập trình, phần mềm C51 cung cấp các từ khóa sau:
_at_ ; alien; bdata; bit; code; compact data; idata; interrupt; large; pdata; _priority_; reentrant; sbit; sfr; sfr16; small; _task_; using; xdata

6.3 Bộ nhớ chương trình ROM

Bộ nhớ này chỉ dùng để lưu chương trình, không có chức năng lưu dữ liệu, ngoại trừ các hằng số thì cho phép lưu bằng các khai báo sau trong chương trình:

```
unsigned char code constant_1 = 0x03;
unsigned char code array_1[3] = {'1', '2', '3'};
```

6.4 Khai báo biến và hằng số

Ví dụ 1: *Khai báo các hằng trong bộ nhớ chương trình.*

```
unsigned char code dem = 0x03;
unsigned char code ma_ascii[10] = {'0','1', '2', '3', '4', '5', '6', '7', '8', '9'};
```

Ví dụ 2: *Khai báo biến và khởi gán giá trị cho biến trong bộ nhớ dữ liệu RAM*

```
unsigned int speed = 3;
```

6.5 Các BIT chứa chức năng đặc biệt

Trong vi điều khiển có vùng nhớ cho phép truy xuất từng bịt và cho phép truy xuất trực tiếp và vùng nhớ này có tên là BDATA.

Ví dụ 1: *Khai báo các biến byte và bit*

```
char bdata test; //khai báo biến test là kí tự byte thuộc vùng dữ liệu cho phép truy xuất bit.
sbit sign = test^7; //khai báo biến sign là bit thứ 7 của biến test.
sfr P1 = 0x90; //gán P1 là ô nhớ có địa chỉ 0x90.
```

6.6 Định nghĩa các biến

Ví dụ 1: định nghĩa các biến

```
#define HANG P0; // định nghĩa biển HANG là port 0 của vi điều khiển AT89xx
sbit HANG_0 HANG^0; // định nghĩa bit HANG_0 là bit thứ 0 của vi điều khiển AT89xx
```

6.7 Con trỏ dữ liệu

Con trỏ dữ liệu trong ngôn ngữ C chính là kiểu truy xuất gián tiếp dùng thanh ghi, địa chỉ của ô nhớ cần truy xuất lưu trong thanh ghi.

Ví dụ 1*: khai báo con trỏ, gán địa chỉ con trỏ và lưu dữ liệu:*

```
unsigned char *pointer(); //khai báo con trỏ
```

pointer() = mem_address; *//gán địa chỉ bắt đầu của vùng nhớ cho con trỏ*
*pointer() = 0xFF; *//lưu giá trị 0xFF vào ô nhớ có địa chỉ lưu trong con trỏ.*
Đối với vùng nhớ ngoại thì xem ví dụ sau
Ví dụ 2: khai báo con trỏ, gán địa chỉ con trỏ và lưu dữ liệu:
unsigned char *asb_pointer; *//khai báo con trỏ*
asb_pointer = (char*) 0x8000; *// gán địa chỉ bắt đầu của vùng nhớ là 0x8000 cho con trỏ*
* asb_pointer = 0xFF; *//lưu giá trị 0xFF vào ô nhớ có địa chỉ lưu trong con trỏ.*
* asb_pointer++; *//tăng địa chỉ con trỏ đến ô nhớ kế*

6.8 Khai báo mảng

Ví dụ 1: khai báo mảng:
unsigned char ma7doan[] {0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
char chuoi_hien_thi = { "HELLO!"};

6.9 Khai báo chương trình con phục vụ ngắt

Ví dụ: khai báo chương trình con phục vụ ngắt
Timer0_int() interrupt 1 using 2
{
}
Khai báo tên chương trình con phục vụ ngắt và sử dụng ngắt thứ nhất đúng với ngắt của timer0 và dùng bank thanh ghi thứ hai cho chương trình con phục vụ ngắt.

6.10 Cấu trúc của chương trình C

Tất cả các chương trình viết bằng ngôn ngữ C bao gồm các chỉ dẫn, các khai báo biến, các định nghĩa biến, các biểu thức, các lệnh và hàm.
Chương trình C đơn giản như sau

```
/* my first C program*/
#include <AT89X52.h>
unsigned char MA7D[10] = {0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
int GIAY, CHUC, DONVI;
void DELAY_TIMER0(){
    int BDN;
    TR0 = 1;
    for (BDN = 0; BDN<20; BDN++){
        do {}
        while(TF0==0);
        TFO = 0;TH0 = 0X3C;TLO = 0XB0;
    }
    TR0 = 0;
}
void MAIN (){
    TMOD = TO_M0; THO=0X3C: TLO = 0XB0;
    while(1){
    for (GOLD = 0; GOLD <60; GOLD ++){
        TIME = TIME/10;          DONVI = GIAY %10;
        P0 = MA7D[DONVI];        P1 = MA7D[CHUC];
        DELAY_TIMER0();
    }
}
```

- **"/* my first C program*/":** là chú thích cho chương trình nằm trong cặp dấu "/* .... */" hoặc nằm sau "//".
- **"#include < AT89X52.h >":** có chức năng báo cho trình biên dịch biết chương trình dùng các thành phần của vi điều khiển có trong file "AT89X52.h".

- Tiếp theo là **khai** báo biến toàn cục, chương trình con và các lệnh của chương trình con nằm trong dấu {}.
- **"void main ()":** cho biết bắt đầu chương trình chính và các lệnh tiếp theo của chương trình chính nằm trong cặp dấu {}.

6.11 Các thành phần của chương trình C

6.11.1 Chỉ dẫn tiền xử lý

Có 2 chỉ dẫn là #define và #include:

Chỉ dẫn #define có chức năng thay thế một đoạn chuỗi bằng một chỉ định đặc biệt nào đó.

Chỉ dẫn # include có chức năng chèn vào một đoạn lệnh từ một file khác vào trong chương trình.

Ví dụ:

#define ported Pl;

#include <AT89X52.H>;

Chỉ dẫn thứ nhất có chức năng định nghĩa biến portled có giá trị là port1

Chỉ dẫn thứ hai có chức năng khai báo chương trình thư viện của vi điều khiển AT89X52.H

6.11.2 Khai báo

Khai báo biến bao gồm tên và thuộc tính, khai báo hàm, khai báo hằng. Khai báo biến toàn cục nằm bên ngoài hàm, khai báo biển cục bộ thì năm bên trong hàm.

6.11.3 Định nghĩa giá trị cho biến

Dùng để thiết lập giá trị cho một biến hoặc một hàm.

6.11.4 Biểu thức

Là sự kết hợp của toán tử và tác tổ để cho ra một kết quả duy nhất.

6.11.5 Hàm

Một hàm bao gồm các khai báo biển, định nghĩa giá trị, các biểu thức và các lệnh để thực hiện một chức năng đặc biệt nào đó.

6.12 File thư viện cho họ AT89X52

Khi dùng phần mềm Keil lập trình cho vị điều khiển AT89S52 thì phải khai báo thư viện <AT89X52.h>. Trong file này đã định nghĩa tên các thành phần của vi điều khiển họ AT89X52.

BÀI 7: TIMER và COUNTER

- **Timer:** Là một bộ đếm nội bộ, được tăng lên một đơn vị tại mỗi xung clock của vi điều khiển. Khi giá trị đếm đạt đến một giá trị nhất định (giá trị nạp vào thanh ghi), một sự kiện ngắt sẽ xảy ra, cho phép thực hiện các tác vụ khác.
- **Counter:** Cũng là một bộ đếm, nhưng nó đếm các xung bên ngoài được đưa vào chân của vi điều khiển. Điều này cho phép đếm các sự kiện xảy ra ngoài vi điều khiển, chẳng hạn như số lần nhấn nút, số xung từ một cảm biến,...

7.1 Thanh ghi TMOD (Timer Mode Register)

- Xác định chế độ hoạt động của timer 0 và timer 1.
- Gồm 8 bit, chia làm 2 phần, mỗi phần 4 bit điều khiển một timer.

Timer1				Timer0			
bit 7	bit 6	**bit 5**	**bit 4**	bit 3	bit 2	bit 1	bit 0
GATE1	C/T1	**M1_T1**	**M0_T1**	GATE0	C/T0	M1_T0	M0_T0

- **Giải thích chi tiết từng bit:**

Bit	**Chức năng**
Bit 7 – GATE1	**Thiết lập chế độ bật/tắt của Timer1** ▪ 1: Timer1 hoạt động khi chân INT1 = P3.3 = 1 và phụ thuộc vào bit TR1 = 1 trong thanh ghi TCON. ▪ 0: Timer1 hoạt động không phụ thuộc chân INT1 mà chỉ phụ thuộc bit TR1 = 1
Bit 6 – C1/T1	**Thiết lập chế độ hoặc động Timer1 (định thời) hoặc Counter1 (đếm)** ▪ 1: chế độ Counter. Đếm xung ngoài nhận xung từ chân T1 = P3.5 ▪ 0: chế độ Timer. Đếm thời gian nhận xung từ dao động nội bên trong.
Bit 5 – M1_T1 Bit 4 – M0_T1	**Chọn chế độ hoạt động của Timer1** ▪ 00: Chế độ 0 (13-bit Timer/Counter) + Giới hạn đếm từ 0000H đến 1FFFH (2^{13} = 8192 lần). + Giá trị đếm được lưu trong 2 thanh ghi TH1 (chỉ 5 bit thấp) và TL1 (8 bit) ▪ 01: Chế độ 1 (16-bit Timer/Counter) (thường sử dụng) + Giới hạn đếm từ 0000H đến FFFFH (2^{16} = 65536 lần). + Giá trị đếm được lưu trong 2 thanh ghi TH1 (8 bit) và TL1 (8 bit) ▪ 10: Chế độ 2 (8-bit tự động nạp lại) + Giới hạn đếm từ 00H đến FFH (2^{8} = 256 lần). + Giá trị đếm được lưu trong TL1 (8 bit) và được nạp từ TH1 (8bit) ▪ 11: Chế độ 3 (Timer 1 ngưng, Timer 0 hoạt động ở chế độ 2 kênh riêng biệt) + Giới hạn đếm từ 00H đến FFH (2^{8} = 256 lần). + TL0 và TH0 hoạt động như 2 Timer 8-bit riêng biệt. + Chế độ này hiếm khi được sử dụng, chủ yếu được sử dụng khi cần hai bộ đếm 8-bit hoạt động song song từ Timer 0.
Bit 3 – GATE0	Thiết lập chế độ bật/tắt của Timer0 ▪ 1: Timer0 hoạt động khi chân INT0 = P3.2 = 1 và phụ thuộc vào bit TR0 = 1 ▪ 0: Timer0 hoạt động không phụ thuộc chân INT0 mà chỉ phụ thuộc bit TR0 = 1
Bit 2 – C0/T0	Thiết lập chế độ hoặc động **T**imer0 (định thời) hoặc **C**ounter0 (đếm)
Bit 1 – M1_T0 Bit 0 – M0_T0	Chọn chế độ hoạt động của Timer0

7.2 Thanh ghi TCON (Timer Control Register)

- Là một thanh ghi 8 bit, chứa các cờ điều khiển và cờ trạng thái của Timer và các ngắt ngoài.

bit 7	**bit 6**	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
TF1	**TR1**	TF0	TR0	IE1	IT1	IE0	IT0

- **Chức năng của từng bit trong thanh ghi TCON:**

Bit	**Chức năng**
Bit 7 – TF1	(Timer1 Overflow Flag) cờ tràn của Timer1 ▪ **1**: tự động lên mức 1 báo hiệu Timer1 đã đếm hết giá trị và tràn về 0. ▪ **0**: thiết lập thủ công để tiếp tục hoạt động đếm của Timer1. (*)
Bit 6 – TR1	(Timer 1 Run Control Bit) điều khiển bật/tắt Timer1 ▪ **1**: Bật **Timer 1** (Timer bắt đầu đếm). ▪ **0**: Dừng **Timer 1**.
Bit 3 – IE1	(External Interrupt 1 Edge Flag) cờ báo hiệu trạng thái của ngắt chân INT1 ▪ **1**: có nghĩa là ngắt ngoài 1 (chân INT1) đã xảy ra và đang chờ được xử lý. Cờ này sẽ tự động được xóa khi chương trình con ngắt được thực thi. ▪ **0**: có nghĩa là ngắt ngoài 1 chưa xảy ra hoặc đã được xử lý.
Bit 2 – IT1	(Interrupt 1 Type Control Bit) xác định kiểu kích hoạt ngắt cho chân INT1 ▪ **1**: tin hiệu ngắt được kích hoạt khi chân INT1 chuyển từ mức cao xuống mức thấp (cạnh xuống) ▪ **0**: khi tín hiệu trên chân INT1 ở mức thấp liên tục (mức thấp)
Bit 5 – TF0	cờ tràn của Timer0
Bit 4 – TR0	điều khiển bật/tắt Timer0
Bit 1 – IE0	cờ báo hiệu trạng thái của ngắt chân INT0
Bit 0 – IT0	xác định kiểu kích hoạt ngắt cho chân INT0

- **(*) Sử dụng Timer không có ngắt và có ngắt**

Sử dụng Timer không ngắt	Sử dụng Timer có ngắt
- Chương trình chính sẽ liên tục kiểm tra cờ tràn của Timer (TF = 1 ?). - Khi kiểm tra chương trình chính phát hiện Timer 1 tràn TF = 1 thì chương trình chính sẽ thực hiện các lệnh liên quan. - Cờ tràn TF sẽ được thiết lập về 0.	- Chương trình chính sẽ không cần kiểm tra cờ tràn của Timer. - Khi Timer tràn (TF = 1) thì tự động một tín hiệu ngắt sẽ xảy ra gửi đến VĐK và VĐK sẽ tự động chuyển tới ***"chương trình con xử lý ngắt"*** để thực hiện các lệnh - Cờ tràn TF sẽ được thiết lập về 0.
- ***Cấu trúc chương trình:*** ▪ Không sử dụng chương trình ngắt. ▪ Kiểm tra liên tục cờ TF trong vòng lặp chính. ▪ Xóa TF thủ công khi Timer tràn.	- ***Cấu trúc chương trình:*** ▪ Sử dụng chương trình ngắt. ▪ Khi Timer tràn, VĐK tự động nhảy đến chương trình con ngắt để xử lý. ▪ Xóa cờ tràn TF trong chương trình con ngắt.
#include <reg52.h> **void main()** { TMOD = 0x01; *// Timer 0 ở chế độ 16-bit (Mode 1)* TH0 = 0x00; *// Nạp giá trị cao của Timer 0* TL0 = 0x00; *// Nạp giá trị thấp của Timer 0* TR0 = 1; *// Bắt đầu chạy Timer 0*	#include <reg52.h> *// Chương trình con phục vụ ngắt cho Timer 0* **void Timer0_ISR(void) interrupt 1** { TF0 = 0; *// Xóa cờ tràn TF0* *// lệnh 1* *// lệnh 2* } **void main() {** TMOD = 0x01; *// Timer 0 ở chế độ 16-bit (Mode 1)* TH0 = 0x00; *// Nạp giá trị cao của Timer 0* TL0 = 0x00; *// Nạp giá trị thấp của Timer 0* ET0 = 1; *// Cho phép ngắt Timer 0* EA = 1; *// Cho phép toàn bộ ngắt* TR0 = 1; *// Bắt đầu chạy Timer 0*

```
while (1) {
  if (TF0 == 1) {  // liên tục kiểm tra cờ tràn TF0
    TF0 = 0;       // Xóa cờ tràn TF0
    // lệnh 1
    // lệnh 2
  }
}
}
```

```
while (1) {
  // Chương trình chính tiếp tục
    thực hiện các tác vụ khác
    mà không cần kiểm tra cờ tràn
}
}
```

- ***Ưu điểm:***
- Dễ hiểu, dễ lập trình.
- Kiểm soát trực tiếp trạng thái Timer từ vòng lặp chính.
- ***Nhược điểm:***
- Hiệu suất thấp: CPU phải kiểm tra liên tục cờ tràn, chiếm thời gian xử lý và không phù hợp cho các hệ thống cần phản hồi nhanh.
- Tốn tài nguyên CPU: Khi CPU dành nhiều thời gian để kiểm tra cờ Timer, nó không thể thực hiện được nhiều tác vụ khác.

- ***Ưu điểm:***
- Hiệu suất cao: CPU không cần kiểm tra liên tục cờ tràn. Khi Timer tràn, ngắt sẽ tự động kích hoạt và CPU xử lý ngay lập tức.
- Tối ưu tài nguyên: Giảm thời gian CPU dành cho việc kiểm tra cờ, cho phép CPU xử lý các tác vụ khác trong khi chờ Timer tràn.
- ***Nhược điểm:***
- Phức tạp hơn: Lập trình ngắt yêu cầu hiểu rõ cách vi điều khiển xử lý ngắt, quản lý ngắt và khôi phục trạng thái CPU sau khi ngắt.
- Phải đảm bảo thời gian xử lý ngắt không quá lâu để tránhảnh hưởng đến các ngắt khác hoặc chương trình chính.

- ***Tóm tắt:***
- Sử dụng Timer không ngắt: Dễ lập trình, nhưng tiêu tốn nhiều tài nguyên CPU do phải kiểm tra cờ liên tục.
- Sử dụng Timer có ngắt: Tối ưu hóa tài nguyên và thời gian xử lý, nhưng yêu cầu kỹ thuật lập trình phức tạp hơn do liên quan đến ngắt.

7.3 Thanh ghi IE (Interrupt Enable Register)

- Là một thanh ghi 8-bit, dùng để **bật hoặc tắt** các ngắt cụ thể.
- Mỗi bit trong thanh ghi này điều khiển một ngắt khác nhau, và chỉ khi bit đó được thiết lập (bằng 1), ngắt tương ứng mới được cho phép hoạt động.

bit 7	bit 6	bit 5	bit 4	**bit 3**	bit 2	bit 1	bit 0
EA	-	ET2	ES	**ET1**	EX1	ET0	EX0

Chức năng của từng bit trong thanh ghi IE:

Bit	**Chức năng**
Bit 7 – EA	(Enable All Interrupts) bật/tắt toàn bộ hệ thống ngắt. ▪ 1: Cho phép tất cả các ngắt (các ngắt sẽ hoạt động nếu bit của chúng cũng được bật). ▪ 0: Vô hiệu hóa toàn bộ ngắt, bất kể các bit ngắt riêng lẻ có bật hay không.
Bit 6	Dự trữ, không sử dụng và luôn bằng 0
Bit 5 – ET2	(Enable Timer 2 Interrupt) cho phép ngắt từ **Timer2** ▪ 1: Cho phép ngắt từ Timer 2 (nếu sử dụng VĐK có hỗ trợ Timer 2) ▪ 0: Vô hiệu hóa ngắt từ Timer 2.
Bit 4 – ES	(Serial Interrupt Enable) cho phép ngắt từ từ **bộ truyền nhận nối tiếp** (UART) ▪ 1: Cho phép ngắt từ **bộ truyền nhận nối tiếp** (UART) ▪ 0: Vô hiệu hóa ngắt từ bộ truyền nhận nối tiếp.
Bit 3 – ET1	(Enable Timer 1 Interrupt) cho phép ngắt từ **Timer1**
Bit 2 – EX1	(External Interrupt 1 Enable) cho phép ngắt ngoài từ chân **INT1** (chân P3.3) ▪ 1: Cho phép ngắt ngoài INT1 ▪ 0: Vô hiệu hóa ngắt ngoài INT1
Bit 1 – ET0	(Enable Timer 0 Interrupt) cho phép ngắt từ **Timer0**
Bit 0 – EX0	(External Interrupt 0 Enable) cho phép ngắt ngoài từ chân **INT0** (chân P3.2)

Tài liệu tham khảo

Giáo trình Vi xử lý – Nguyễn Đình Phú – Trương Ngọc Anh – NXB ĐHQG TPHCM
Giáo trình Kỹ Thuật Lập Trình C – Nguyễn Linh Giang – NXB Giáo Dục