1

Introduction to Microcontrollers

1.1 Introduction

A microcontroller is a computer with most of the necessary support chips onboard. All computers have several things in common, namely:

- A central processing unit (CPU) that 'executes' programs.
- Some random-access memory (RAM) where it can store data that is variable.
- Some read only memory (ROM) where programs to be executed can be stored.
- Input and output (I/O) devices that enable communication to be established with the outside world i.e. connection to devices such as keyboard, mouse, monitors and other peripherals.

There are a number of other common characteristics that define microcontrollers. If a computer matches a majority of these characteristics, then it can be classified as a 'microcontroller'. Microcontrollers may be:

- 'Embedded' inside some other device (often a consumer product) so that they can control the features or actions of the product. Another name for a microcontroller is therefore an 'embedded controller'.
- Dedicated to one task and run one specific program. The program is stored in ROM and generally does not change.
- A low-power device. A battery-operated microcontroller might consume as little as 50 milliwatts.

A microcontroller may take an input from the device it is controlling and controls the device by sending signals to different components in the device. A microcontroller is often small and low cost. The components may be chosen to minimise size and to be as inexpensive as possible.

The actual processor used to implement a microcontroller can vary widely. In many products, such as microwave ovens, the demand on the CPU is fairly low

and price is an important consideration. In these cases, manufacturers turn to dedicated microcontroller chips – devices that were originally designed to be low-cost, small, low-power, embedded CPUs. The Motorola 6811 and Intel 8051 are both good examples of such chips.

A typical low-end microcontroller chip might have 1000 bytes of ROM and 20 bytes of RAM on the chip, along with eight I/O pins. In large quantities, the cost of these chips can sometimes be just a few pence.

In this book the authors will introduce the reader to some of the Philips' 8051 family of microcontrollers, and show their working, with applications, throughout the book. The programming of these devices is the same and, depending on type of device chosen, functionality of each device is determined by the hardware devices onboard the chosen device.

1.2 Microcontroller types

The predominant family of microcontrollers are 8-bit types since this word size has proved popular for the vast majority of tasks the devices have been required to perform. The single byte word is regarded as sufficient for most purposes and has the advantage of easily interfacing with the variety of IC memories and logic circuitry currently available. The serial ASCII data is also byte sized making data communications easily compatible with the microcontroller devices. Because the type of application for the microcontroller may vary enormously most manufacturers provide a family of devices, each member of the family capable of fitting neatly into the manufacturer's requirements. This avoids the use of a common device for all applications where some elements of the device would not be used; such a device would be complex and hence expensive. The microcontroller family would have a common instruction subset but family members differ in the amount, and type, of memory, timer facility, port options, etc. possessed, thus producing cost-effective devices suitable for particular manufacturing requirements. Memory expansion is possible with off-chip RAM and/or ROM; for some family members there is no on-chip ROM, or the ROM is either electrically programmable ROM (EPROM) or electrically erasable PROM (EEPROM) known as flash EEPROM which allows for the program to be erased and rewritten many times. Additional on-chip facilities could include analogue-to-digital conversion (ADC), digital-to-analogue conversion (DAC) and analogue comparators. Some family members include versions with lower pin count for more basic applications to minimise costs. Many microcontroller manufacturers are competing in the market place and rather than attempting to list all types the authors have restricted the text to devices manufactured by one maker. This does not preclude the book from being useful for applications involing other manufacturer's devices; there is a commonality among devices from various sources, and descriptions within the text can, in most cases, be applied generally. The chapters that follow will deal with microcontroller family members available from Philips Semiconductors, and acknowledgement is due to the considerable assistance given by that

company in the production of this text. The Philips products are identified by the numbering system:

8XCXXX

where in general, since there are exceptions, the digit following the 8 is:

0 for a ROMless device
3 for a device with ROM
7 for a device with EPROM/OTP (one time programmable)
9 for a device with FEEPROM (flash).

Following the C there may be 2 or 3 digits. Additional digits, not shown above, would include such factors as clock speed, pin count, package code and temperature range. Philips also produces a family of 16-bit microcontrollers in the eXtended Architecture (XA) range. For these devices Philips claims compatibility with the 80C51 at source code level with full support for the internal registers, operating modes and 8051 instructions. Also claimed is a much higher speed of operation than the 8051 devices. The XA products are identified by the numbering system:

PXAG3XXXX

where:

PXA is Philips 80C51 XA
G3 is the derivative name
next digit is memory option:
  0 = ROM less
  3 = ROM
  5 = Bond-out (emulation)
  7 = EPROM/OTP
  9 = FEEPROM (flash)
next digit is speed:
  J = 25 MHz
  K = 30 MHZ
next digit is temperature:
  B = 0°C to + 70°C
  F = −40°C to + 85°C
final digit is package code:
  A = Plastic Leaded Chip Carrier (PLCC)
  B = Quad Flat Pack (QFP)
 etc.

The XA architecture supports:

- 16-bit fully static CPU with a 24-bit program and data address range;
- eight 16-bit CPU registers, each capable of performing all arithmetic and logic operations as well as acting as memory pointers;

- both 8-bit and 16-bit CPU registers, each capable of performing all arithmetic and logic operations;
- an enhanced instruction set that includes bit-intensive logic operations and fast signed or unsigned 16×16 multiplies and 32/16 divide;
- instruction set tailored for high-level language support;
- multitasking and real-time executives that include up to 32 vectored interrupts, 16 software traps, segmented data memory and banked registers to support context switching.

The next section of this chapter will look at a member of the Philips 80C51 family in more detail.

1.3 P89C66x microcontroller

Figure 1.1 shows a P89C664 microcontroller in a PLCC package.

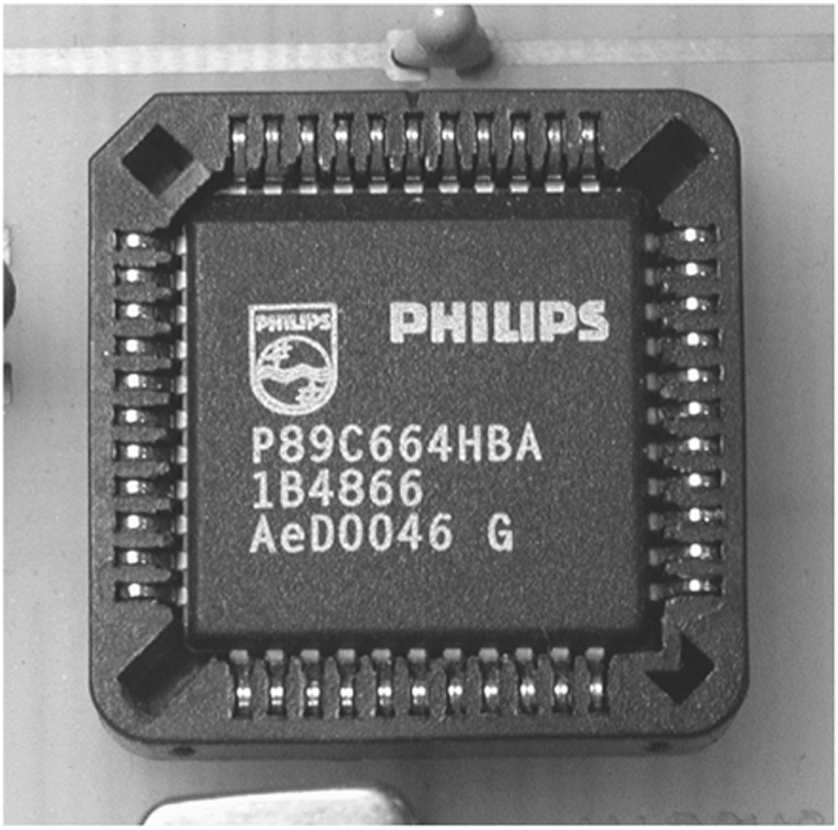

Figure 1.1 Philips P89C664 PLCC package microcontroller

P means the device is manufactured by Philips Semiconductors
8 means the micro belongs to the 8-bit 8051 family
9 means Flash code (program) memory
C means CMOS technology and
664 belongs to the 66x family

where:

x = 0 16 KB Flash code memory, 512 bytes onboard RAM
x = 2 32 KB Flash code memory, 1 KB onboard RAM
x = 4 64 KB Flash code memory, 2 KB onboard RAM
x = 8 64 KB Flash code memory, 8 KB onboard RAM

All devices belonging to this family of devices have a universal asynchronous receive transmit (UART), which is a serial interface similar to the COM interface on a PC. Figure 1.2 shows the logic symbol for the device and illustrates the pin functions.

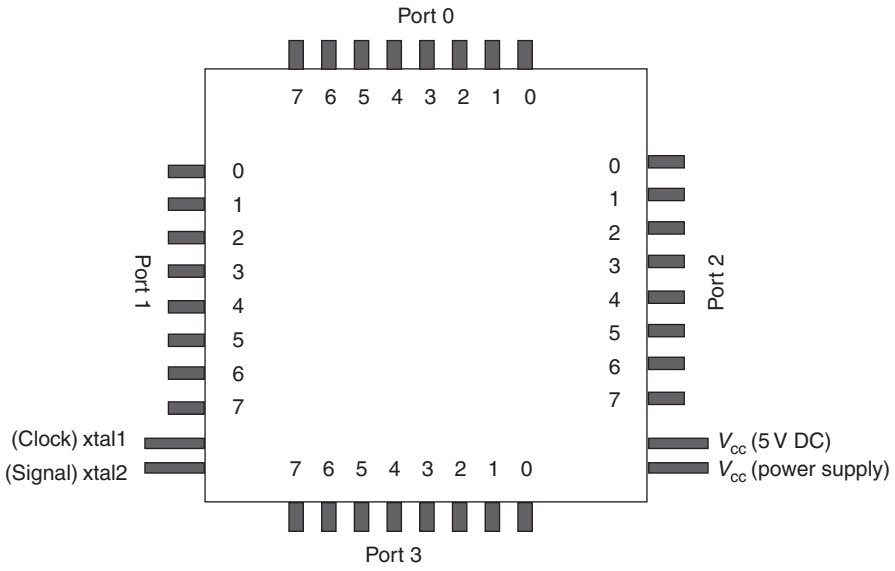

Figure 1.2 Logic symbol for the P89C66x family

The P89C66x family of microcontrollers have four 8-bit ports: port 0, port 1, port 2 and port 3.

Traditionally in the 80C51 family of microcontrollers the function of port 0 and port 2 is primarily to allow for connection to an external PROM (code memory chip). Port 0 provides both the 8-bit data and the lower 8 bits of the address bus, A0 to A7. Port 2 provides the upper 8 address bits, A8 to A15. All of the flash microcontrollers referred to in this text have onboard code memory, which can be as much as 64 KB.

Port 0 pins are all from open-drain transistors and the port pins should have pull-up resistors (e.g. 2.7 kΩ from pin to 5 V DC supply) if the port is to be used as a general-purpose interface.

Port 3 has some special function pins, e.g. pins 0 and 1 of port 3 may be used as receive and transmit for the UART. Functions of other pins will be covered in later chapters.

In the 80C51 family of microcontrollers the RAM is organised into 8-bit locations.

MSB							LSB
7	6	5	4	3	2	1	0

The bits are numbered 7, 6, 5, 4, 3, 2, 1, 0 where bit 7 is the most significant bit (MSB) and bit 0 the least significant bit (LSB).

A bit (binary digit) has two values, logic 0 or logic 1. Electrically logic 0 is 0 V whereas logic 1 is the value of the microcontroller IC positive supply voltage. Logic 1 is usually 5 V but nowadays with increasing use of batteries for power supplies logic 1 could be 3 V or 1.8 V.

Power depends on the square of the voltage and there is a significant saving in power (i.e. battery lasts longer) if the microcontroller is powered by 3 V or 1.8 V power supplies.

The maximum number that can be stored in an 8-bit memory location is $2^8 - 1$, which equals 255. This would occur when all the bits are equal to 1 i.e.:

MSB							LSB
1	1	1	1	1	1	1	1

Binary is a base 2 number system and the electronic devices in the microcontroller's logic circuits can be set to logic 0 and logic 1.

The value of each bit is:

MSB							LSB
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Example 1.1
Show that if an 8-bit register contains all logic 1s then the value stored is 255.

Solution
With all bits of the register set to logic 1 the total value stored is given by:

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

Remember the sequence by recalling that the LSB is 1 and the other numbers are successively doubled.

Exercise 1.1
What is the maximum number that can be stored in a 10-bit wide register?

1.4 Bits, nibbles, bytes and number conversions

BITS, BYTES AND NIBBLES

A bit is a single binary digit of value logic 1 or logic 0. A nibble is a group of 4 bits, e.g. 1010 is a nibble. A byte is a group of 8 bits e.g. 10100111 is a byte and the byte is made up of two nibbles 1010 and 0111.

DECIMAL TO BINARY CONVERSION

A decimal number may be converted to binary by dividing the number by 2, giving a quotient with a remainder of 0 or 1. The process repeats until the final quotient is 0. The remainders with the first remainder being the least significant digit determine the binary value. The process is best explained with an example.

Example 1.2
Express the decimal number 54 as a binary number.

Solution

$$\begin{aligned} 54/2 &= 27, \text{ remainder } 0 \\ 27/2 &= 13, \text{ remainder } 1 \\ 13/2 &= 6, \text{ remainder } 1 \\ 6/2 &= 3, \text{ remainder } 0 \\ 3/2 &= 1, \text{ remainder } 1 \\ 1/2 &= 0, \text{ remainder } 1 \end{aligned}$$

Thus 54 decimal = 110110 and in an 8-bit register the value would be 00110110. A binary value is often expressed with a letter B following the value i.e. 00110110B.

It may be easier to use the weighted values of an 8-bit register to determine the binary equivalent of a decimal number i.e. to break the decimal number down to those weighted elements, which have a logic 1 level.

Example 1.3
Express the decimal number 54 as a binary number using weighted values.

Solution

$$54 = 32 + 16 + 6$$

$0 \times 128 \quad + 0 \times 64 \quad + 1 \times 32 \quad + 1 \times 16 \quad + 0 \times 8 \quad + 1 \times 4 \quad + 1 \times 2 \quad + 0 \times 1$

0	0	1	1	0	1	1	0

Hence 54 decimal = 00110110B.

Example 1.4
Express decimal 167 as a binary number.

Solution
Using the technique of Example 1.3:

$$167 = 128 + 32 + 4 + 2 + 1 = 10100111\text{B}$$

Exercise 1.2
Represent decimal numbers 15 and 250 in binary format.

It follows that to convert binary to decimal the reverse procedure applies i.e. to convert the binary number 00110110 to decimal is achieved by simply adding the weighted values of the logic 1 states. This is shown in the answer to Example 1.3 where 00110110B = 54 decimal.

BINARY, HEXADECIMAL (HEX) AND DECIMAL

When working out values at the port pins, the tendency is to think in binary, e.g. which LED to turn on, the logic level on a switch, etc.

The assembly language software tends to use hexadecimal, a base 16 number system useful for grouping nibbles. Since childhood we have been taught to become familiar with the base 10 decimal system. It is useful to be able to work between the three number systems:

Binary	Hex	Decimal
0 0 0 0	00	00
0 0 0 1	01	01
0 0 1 0	02	02
0 0 1 1	03	03
0 1 0 0	04	04
0 1 0 1	05	05
0 1 1 0	06	06
0 1 1 1	07	07
1 0 0 0	08	08
1 0 0 1	09	09
1 0 1 0	0A	10
1 0 1 1	0B	11
1 1 0 0	0C	12
1 1 0 1	0D	13
1 1 1 0	0E	14
1 1 1 1	0F	15

Consider the following examples:

Example 1.5
Express ABCD as binary.

Solution

$$\text{ABCD} = 1010\,1011\,1100\,1101$$

Example 1.6
Express 101111000001 as a hexadecimal value.

Solution

$$1011\ 1100\ 0001 = \text{BC1}$$

Example 1.7
Express 01110011110 as a hexadecimal value.

Solution

$$0011\ 1001\ 1110 = \text{39E}$$

Because in this last example the number of bits does not subdivide into groups of four bits, the method used is to group into nibbles from the right, filling the spaces at the front with zeros.

Example 1.8
Express decimal 71 as a hex number.

Solution

$$71/16 = 4 \text{ remainder } 7 = 47 \text{ Hex, usually written as } 47\text{H}$$

Example 1.9
Express decimal 143 as a hex number.

Solution

$$143/16 = 8 \text{ remainder } 15 = 8\text{FH}$$

Conversion from binary to decimal can be achieved quickly by first converting the binary number to hex and then converting the hex number to decimal. An example illustrates the process.

Example 1.10
Express 11000101B in decimal form.

Solution
Converting to hex:

$$11000101\text{B} = \text{C5H}$$

The hex number represents a nibble of binary data and each value is to a power of 16 with the least significant nibble equal to $16^0(=1)$ and the next significant nibble equal to $16^1(=16)$. Hence the decimal number is:

$$(C \times 16) + (5 \times 1) = (12 \times 16) + (5 \times 1) = 197 \text{ decimal}$$

Check:

$$11000101 = (1 \times 128) + (1 \times 64) + (1 \times 4) + (1 \times 1) = 197 \text{ decimal}$$

Exercise 1.3
Express decimal 200 as a hex number and then as a binary number.

Exercise 1.4
Express the following binary numbers as hex and then decimal numbers.

1. 10000110
2. 10011000011

1.5 Inside microcontrollers

Microcontrollers normally contain RAM, ROM (EEPROM, EPROM, PROM), logic circuits designed to do specific tasks (UART, I^2C, SPI) and square-wave oscillator (clock).

Built from the logic circuitry the microcontroller has two parts, the processor core and the onboard peripherals. See Figure 1.3.

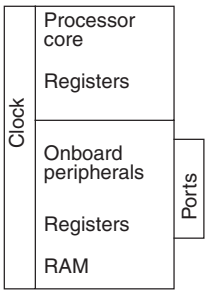

Figure 1.3 Constituent parts of a microcontroller

RAM locations that have special functions and support the processor core and onboard peripheral circuitry are called special function registers (SFRs) and are reserved areas.

The program instructions provide the primary inputs to the processor core circuitry. See Figure 1.4.

The microcontroller program resides in the PROM (programmable ROM), which, in the microcontrollers we are considering, uses Flash technology and is located in the microcontroller IC.

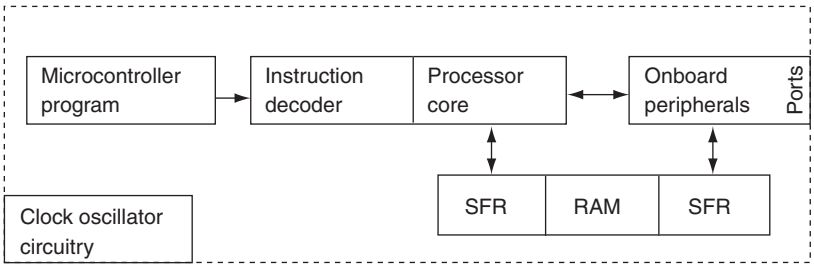

Figure 1.4 Block diagram of a microcontroller

1.6 Microcontroller programming

The microcontroller program comprises a set of instructions written by the program designer. There are four classes of instructions:

1. Arithmetic operations
2. Logic operations
3. Data transfer operations
4. Branch operations.

ARITHMETIC OPERATIONS

Arithmetic instructions operate on whole numbers only and support addition, subtraction, multiplication and division.

Addition

```
ADD   A,#66H   ; add the hex number 66 to the accumulator A
```

This is an example of immediate addressing.

The # sign is important, if it were omitted the operation would have a different meaning.

```
ADD   A,66H    ; add to accumulator A the contents of RAM address
               ; 0066H
```

This is an example of direct addressing.

Accumulator A is an SFR; it is an 8-bit register and its RAM address is 00E0H. A large number of instructions use accumulator A, but not all.

```
INC   66H   ; increment (add 1) the contents of address 0066H
```

Exercise 1.5
Is there any difference between the following two instructions?
A) INC A B) ADD A,#1

Subtraction

```
SUBB   A, #66H   ; subtract hex66 from the contents of A
```

The extra B in the instruction implies Borrow. If the contents of A are less than the number being subtracted then bit 7 of the program status word (PSW) SFR will be set. (For details of the PSW and other SFRs, see Appendix C.)

```
DEC   A   ; decrement A by 1, put result into A
```

Exercise 1.6
Is there any difference between the following two instructions?

(1) DEC A (2) SUBB A,#1

Multiplication

```
MUL   AB   ; multiply the contents of A and B, put the answer in AB
```

A is the accumulator and B is another 8-bit SFR provided for use with the instructions multiply and divide. A and B are both 8-bit registers. The product of the multiplication process could be a 16-bit answer.

Example 1.11
A = 135 decimal, B = 36 decimal. What would be the value in each register after executing the instruction MUL AB?

Solution

A × B = 4860 = 0001 0010 1111 1100B = 12FCH
0001 0010 or 12H would be placed in A, 1111 1100 or FCH in B

Exercise 1.7
If A = 2FH and B = 02H, what would each register contain after execution of the instruction MUL AB?

Division

```
DIV   AB   ; divide A by B, put quotient in A and remainder in B
```

Example 1.12
A = 135, B = 36. What would be the value in each register after execution of the instruction DIV AB?

Solution
Decimal values are assumed if the value quoted is not followed by an H

A/B = 3, remainder 27 (27 = 1BH). Hence 03H in A, 1BH in B

If multiplication or division is not being used then register B, which is bit addressable, can be used as a general-purpose register.

Exercise 1.8
If A = 2FH and B = 02H, what would each register contain after the execution of the instruction DIV AB?

LOGIC OPERATIONS

The set of logic functions include:

ANL	AND Logic
ORL	OR Logic
XRL	exclusive OR Logic
CPL	Complement (i.e. switch to the opposite logic level)
RL	Rotate Left (i.e. shift byte left)
RR	Rotate Right (i.e. shift byte right)
SETB	Set bit to logic 1
CLR	Clear bit to logic 0

AND operation

The ANL instruction is useful in forcing a particular bit in a register to logic 0 whilst not altering other bits. The technique is called masking.

Suppose register 1 (R1) contains EDH (1110 1101B),

1	1	1	0	1	1	0	1

bit 1 and bit 4 are at logic 0, the rest at logic 1.

```
ANL   R1, #7FH   ; 7FH = 0111 1111B, forces bit 7 to zero
```

1	1	1	0	1	1	0	1

AND

0	1	1	1	1	1	1	1

=

0	1	1	0	1	1	0	1

Exercise 1.9
If A = 2D, what would be the accumulator contents after execution of the instruction ANL A, #3BH?

ORL operation

Another aspect of masking is to use the ORL instruction to force a particular bit to logic 1, whilst not altering other bits.

The power control (PCON) SFR in the 8051 family, is not bit addressable and yet has a couple of bits that can send the microcontroller into idle mode or power down mode, useful when the power source is a battery.

The contents of the PCON SFR are:

PCON

SMOD1	SMOD2		POF	GPF1	GPF2	PD	IDL

SMOD1 and 2 are used when setting the baud rate of the serial onboard peripheral. POF, GPF1 and GPF2 are indictor flag bits. IDL is the idle bit; when set to 1 the microcontroller core goes to sleep and becomes inactive. The on-chip RAM and SFRs retain their values. PD is the Power Down bit, which also retains the on-chip RAM and SFR values but saves the most power by stopping the oscillator clock.

```
ORL   PCON,#02H   ; enables Power Down
ORL   PCON,#01H   ; enables Idle mode
```

Either mode can be terminated by an external interrupt signal. Details of all device SFRs are to be found in Appendix C.

Exercise 1.10
If the contents of register 0 (R0) = 38H, what would the contents of that register be after execution of the following instruction?

```
ORL   R0,#9AH
```

CPL complement operation

The instructions described so far have operated on bytes (8 bits) but some instructions operate on bits and CPL is an example.

```
CPL   P1.7   ; complement bit 7 on Port 1
```

Port 1 is one of the microcontroller's ports with 8 pins.

Port 1

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

MSB LSB

Complement has the action of the inverter logic gate as shown in Figure 1.5.

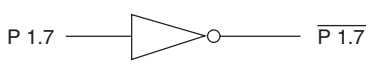

Figure 1.5 Production of the complement of pin function

Exercise 1.11
If the contents of port 0 (P0) = 125, what would be the port contents after execution of the following instruction?

```
CPL   P0
```

RL, rotate left one bit, RR, rotate right one bit operations

Suppose the accumulator A contents are 0000 0001B; this is 01H.

```
RL A   ; contents of A become 0000 0010B or 02H
RL A   ; 0000 0100B or 04H
RL A   ; 0000 1000B or 08H
```

RL three times has the effect of multiplying A by 2^3 i.e. by 8.

Suppose the accumulator A contents are 1000 0000B, or 128 decimal, then:

```
RR A   ; contents of A become 0100 0000B which is 64 decimal
RR A   ; A becomes 0010 0000B = 32 decimal
RR A   ; A becomes 0001 0000B = 16 decimal
RR A   ; A becomes 0000 1000B = 8 decimal
```

RR four times has the same effect as dividing A by 2^4 i.e. 16.

$$\frac{128}{16} = 8$$

Exercise 1.12
If the content of A is 128 and B is 2, what would the register contents be after execution of the following instructions?

```
RR A
RL B
RR A
RR A
RL B?
```

SETB set bit, CLR clear bit operations

This instruction operates on a bit, setting it to logic 1.

```
SETB   P1.7   ; set bit 7 on Port 1 to logic 1
```

Consider Figure 1.6 where pin 7 of port 1 is connected as shown.

SETB P1.7 puts logic 1 (e.g. 5 V) onto the inverter input and therefore its output, the LED cathode, is at 0 V causing current to flow through the LED. The LED has a particular forward voltage V_f (refer to component specification e.g. www.farnell.com).

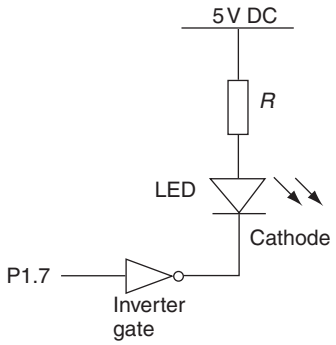

Figure 1.6 Use of an LED to indicate the state of port 1, pin 7

Typically $V_f = 2.2\,V$ and forward current $I_f = 8\,mA$ so that:

$$R = \frac{5\,V - V_f}{I_f} = \frac{5 - 2.2}{8 \times 10^{-3}} = \frac{2.8 \times 1000}{8} = 350\,\Omega = 330\,\Omega \text{ (preferred value)}$$

```
CLR   P1.7   ; clears bit 7 on port 1 to zero
```

CLR P1.7 puts logic 0 on the inverter gate input and therefore its output, the LED cathode, becomes logic 1 which is 5 V. This gives a voltage difference (5 V DC – cathode voltage) of 0 V and the LED turns off.

The inverter gate in the above circuit provides a good current buffer protecting the microcontroller port pin from unnecessary current loading. In the above circuit the current flow is between the inverter gate and the 5 V DC supply.

If an inverter gate is not used to drive a LED then the control may be directly from the port pin but this will demand a current in milliamps from the port pin.

Generally a microcontroller port pin can sink current better than it can source current. See Figure 1.7.

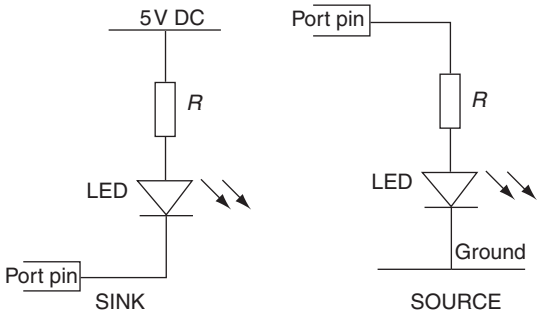

Figure 1.7 Arrangements to allow a port pin to SINK or SOURCE current

CLR port_pin; will ground the LED cathode in the SINK circuit and turn it on. This will turn the LED off in the SOURCE circuit.

SETB port_pin; will put logic 1 on the LED cathode in the SINK circuit and turn it off. This will turn the LED on in the SOURCE circuit.

Exercise 1.13
If $V_{cc} = 5\,V$ and for an LED, $V_f = 0.7\,V$ and the pin P0.0 of the microcontroller port can sink 10 mA and source 50 μA.

1. How you connect the LED to the microcontroller and
2. Calculate the value of series resistor R.

Data transfer operations

This is mainly concerned with transfer of data bytes (8 bits). SETB and CLR have just been covered; they operate on bits.

MOV operation

MOV moves bytes of data. Consider driving a seven-segment display (decimal point dp included) where each LED is driven by the sink method. See Figure 1.8.

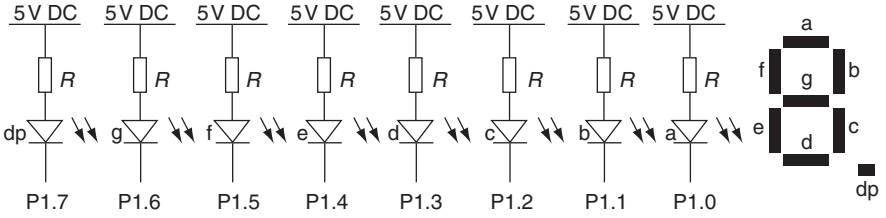

Figure 1.8 Arrangement for a seven-segment LED display

Each LED illuminates a segment. The seven-segment display is shown to the right with its standard segment identification letters.

Example 1.13
Write two program lines, one to display 3, the second to display 4. In both cases turn the decimal point off.

Solution

	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	
	dp	g	f	e	d	c	b	a	
3	1	0	1	1	0	0	0	0	B0H
4	1	0	0	1	1	0	0	1	99H

```
MOV   P1,#B0H   ; display 3
MOV   P1,#99H   ; display 4
```

Note: MOV P1,#B0H would give a syntax error. In common with a number of cross assemblers the software would see B0H as a label because it starts with a hex symbol; 99H would be acceptable since it starts with a number. The correct program line should be MOV P1,#0B0H i.e. a zero must be placed in front of the hex symbol.

The instruction MOV is used to move RAM data that is onboard the microcontroller.

Examples

```
MOV   0400H,#33H   ; move the number 33 hex to RAM address 0400 hex
MOV   A,P1         ; move the contents of port 1 to accumulator A
MOV   R0,P3        ; move the contents of port 3 into register R0
```

Note: As well as the accumulator A the microcontroller has 32 registers in four banks of eight in the processor core. These 32 bytes are fast RAM and should be used in preference to standard onboard RAM.

Each of the banks contain 8 registers R7, R6, R5, R4, R3, R2, R1, R0. There are four banks: 0, 1, 2 and 3.

Bank 0 is the default bank; the other banks can be selected by two bits (RS1,RS0) in the program status word (PSW) SFR

PSW

CY	AC	F0	RS1	RS0	OV	F1	P

0	0	Register bank 0 (default)
0	1	Register bank 1
1	0	Register bank 2
1	1	Register bank 3

Other PSW bits are indicator flags:

CY (carry flag)
AC (auxiliary carry flag)
OV (overflow flag)
P (parity flag)
F0, F1 (general-purpose user-defined flags)

More information on the register banks and the SFRs can be found in Appendix C. MOVX is used to move data between the microcontroller and the external RAM. MOVC is used to move data (e.g. table data) from PROM (also called code memory) to RAM.

Exercise 1.14
Write an instruction to select the register bank 2 of the microcontroller.

Branch operations

There are two types, unconditional and conditional branching. Unconditional branch operations are

ACALL absolute call
LCALL long call

ACALL calls up a subroutine, the subroutine must always have RET as its last operation. ACALL range is limited to +127 places forward or −128 places backward. If your program jumps further than ACALL the compiler will report that the program is jumping out of bounds and replacement by LCALL will solve the problem.

ACALL is two bytes long, LCALL is three bytes long.

AJMP absolute jump
LJMP long jump
SJMP short jump

Similar to ACALL and LCALL, AJMP and LJMP jump to addresses whereas SJMP, which has a similar range to ACALL and AJMP, jumps a number of places.

The difference could be seen in the machine code. Consider the program:

```
$INCLUDE (REG66X.INC)     ; lists all sfr addresses
         ORG    0         ; sets start address to 0
         SJMP   START     ; short jump to START label
         ORG    0040H     ; puts next program line at address 0040H
START:   SETB   P1.7      ; set pin 7 on port 1 to logic 1
         CLR    P1.7      ; clear pin 7 on port 1 to logic 0
         AJMP   START     ; jump back to START label
         END               ; no more assembly language
```

The machine code can be viewed in the list file, progname.lst:

SJMP START
Shows as 803E 80 is the hex for instruction SJMP
3E is the relative jump to reach 0040H where START is; it jumps from address 0002, the address after SJMP START, 0002 + 3E = 0040H

AJMP START
Shows as 0140 01 is the hex for instruction AJMP
40 is short for address 0040

If LJMP had been used instead of AJMP then,
LJMP START
Shows as 020040 02 is the hex for instruction LJMP
0040 is the full address

Exercise 1.15
In your own words describe the difference between ACALL and AJMP instructions.
Conditional branch operations:

JZ Jump if zero
JNZ Jump if not zero
DJNZ Decrement and jump if not zero

Consider an example (a subroutine called by ACALL):

```
DELAY:   MOV    R0,#34     ; move decimal 34 into register R0
TAKE:    DJNZ   R0,TAKE    ; keep subtracting 1 from R0 until zero
RET                        ; return from subroutine
```

CJNE Compare and jump if not equal
Consider:

```
DELAY:   MOV    R0,#34          ; move decimal 34 into register R0
TAKE:    DEC    R0              ; decrement R0
         CJNE   R0,#12,TAKE     ; compare R0 with 12 jump to TAKE if not
         RET                    ; return when R0 equals 12
```

Other instructions are:

JC jump if carry is 1
JNC jump if carry is 0

JB jump if bit = 1
JNB jump if bit = 0

Consider a practical example of testing switched logic levels. Refer to Figure 1.9. If the switch is not pressed the voltage on the port pin is 0 V. When the switch is pressed and held, then the port pin is connected directly to 5 V. To test for switch being pressed, the following program could be used:

```
$INCLUDE (REG66X.INC)       ; lists all sfr addresses
         ORG    0           ; sets start address to 0
         SJMP   START       ; short jump to START label
         ORG    0040H       ; puts next program line at address 0040H
START:   JB     P1.0,PULSE  ; jump to PULSE if pin 0 port 1 is logic 1
         CLR    P1.7        ; otherwise clear pin 7 port 1 to zero
         SJMP   START       ; go to START check switch
```

```
PULSE:   SETB   P1.7          ; set pin 7 on port 1 to logic 1
         CLR    P1.7          ; clear pin 7 on port 1 to logic 0
         AJMP   START         ; go to START check switch
         END                  ; no more assembly language
```

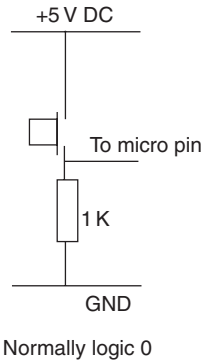

Figure 1.9 Circuit to produce logic levels 0 or 1 at a port pin. Circuit normally producing logic 0

Also, consider the case when pressing the switch generates a logic '0', as shown in Figure 1.10.

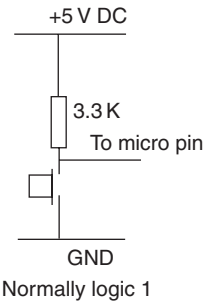

Figure 1.10 Circuit to produce logic levels 0 or 1 at a port pin. Circuit normally producing logic 1

If the switch is not pressed the voltage on the port pin is 5 V. When the switch is pressed and held, the port pin is directly connected to ground or 0 V. The test instruction could be:

```
CHECK:   JNB    P1.0,PULSE   ; jump to PULSE if pin 0 port 1 is logic 0
         SJMP   CHECK
PULSE:
```

Exercise 1.16

In your own words describe the difference between JNB and JNC instructions.

1.7 Commonly used instructions of the 8051 microcontroller

The P89C664 is a member of the 8051 family; it is a CISC device having well over 100 instructions. The instructions used in this text could be the first set to become familiar with.

MOV	move a byte
SETB CLR	set or clear bits
ACALL RET	call up a subroutine
SJMP AJMP	unconditional jump
JB JNB	bit test, conditional jump
DJNZ CJNE	byte test, conditional jump
ORL	OR logic, useful for forcing bits to logic 1
ANL	AND logic, useful for forcing bits to logic 0

The full 8051 instruction set is shown in Appendix A.

COMMONLY USED ASSEMBLER DIRECTIVES

ORG	define address
DB	define bytes, useful for table data
END	all assembly language programs must end with this.

1.8 Microcontroller clock

The microcontroller may be likened to a logic circuit whose logic states change in synchronism with the microcontroller clock signal. This is a square-wave signal as shown in Figure 1.11.

Knowledge of the microcontroller clock cycle time is useful in defining timing events used in applications.

Example 1.14
A P89C664 microcontroller has a clock frequency of 11.0592 MHz. What is the time for each cycle?

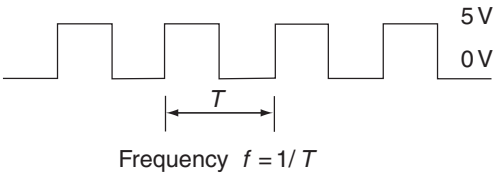

Figure 1.11 Square-wave signal at a frequency f Hz

Solution

$$\text{Cyclic time } (T) = \frac{1}{11.0592 \times 10^6} = 90.423\,\text{ns} \qquad (n = 10^{-9})$$

Now let us look at the previous program:

```
$INCLUDE (REG66X.INC)        ; lists all sfr addresses
         ORG    0            ; sets start address to 0
         SJMP   START        ; short jump to START label
         ORG    0040H        ; puts next program line at address 0040H
START:   JB     P1.0,PULSE   ; jump to PULSE if pin 0, port 1 is logic 1
         CLR    P1.7         ; otherwise clear pin 7 port 1 to zero
         SJMP   START        ; go to START check switch
PULSE:   SETB   P1.7         ; set pin 7 on port 1 to logic 1
         CLR    P1.7         ; clear pin 7 on port 1 to logic 0
         AJMP   START        ; go to START check switch
         END                 ; no more assembly language
```

Initial inspection might lead to the conclusion that the output signal on port 1, pin 7 is a square wave being turned on by SETB and off by CLR. Closer inspection reveals that CLR is held for the extra duration of AJMP and JB. Reference to Appendix A shows that:

SETB	takes 6 microcontroller clock cycles
CLR	takes 6 microcontroller clock cycles
AJMP	takes 12 microcontroller clock cycles
JB	takes 12 microcontroller clock cycles

SETB is held for 6 clock cycles and CLR is held for 30 clock cycles, not an equal on/off waveform as Figure 1.12 shows.

If an equal on/equal off waveform is required then the NOP (No OPeration) can be used. The NOP operation takes 6 clock cycles. The program could be modified:

```
$INCLUDE (REG66X.INC)        ; lists all sfr addresses
         ORG    0            ; sets start address to 0
         SJMP   START        ; short jump to START label
         ORG    0040H        ; puts next program line at address 0040H
START:   JB     P1.0,PULSE   ; jump to PULSE if pin 0, port 1 is logic 1
         CLR    P1.7         ; otherwise clear pin 7, port 1 to zero
         SJMP   START        ; go to START check switch
```

```
PULSE:   SETB   P1.7          ; set pin 7 on port 1 to logic 1
         NOP                  ; hold logic 1 on pin 7, port 1
         NOP
         NOP
         NOP
         CLR    P1.7          ; clear pin 7 on port 1 to logic 0
         AJMP   START         ; go to START check switch
         END                  ; no more assembly language
```

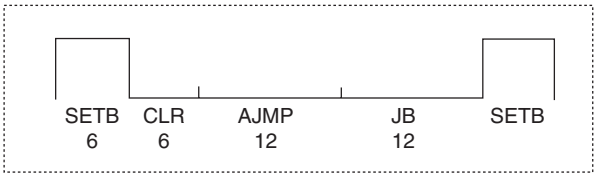

Figure 1.12 Waveform produced using specified instructions. Note that the waveform is not a square wave (i.e. there are unequal ON and OFF periods)

The modified waveform is shown in Figure 1.13.

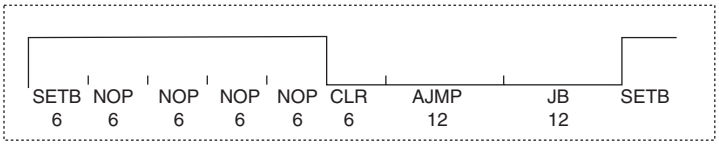

Figure 1.13 Modification to the waveform of Figure 1.12 using NOP instructions to produce a square-wave output

The cycle time of the equal on/off waveform = 60 microcontroller clock cycles. If the microcontroller had a clock frequency of 11.0592 MHz then a clock cycle period T is the reciprocal of this frequency, $T = 90.423$ ns. Therefore the cycle time of the equal on/off signal is $60 \times 90.423\,\text{ns} = 5.43\,\mu\text{s}$. The frequency of this signal is $1/5.43\,\mu\text{s} = 184\,\text{kHz}$. The maximum signal frequency would depend on the maximum microcontroller clock frequency; for the P89C664 microcontroller the maximum clock frequency is 20 MHz. Quite often there is a requirement to generate accurate lower frequency signals and for these the basic signal must be slowed down using a time delay.

1.9 Time delays

The NOP instruction is a simple time delay but apart from this there are two methods of creating time delays:

- register decrement
- onboard timers.

The use of onboard timers will be described in a later chapter; here the register decrement method will be described.

The basic single loop program lines are:

```
DELAY:   MOV     R0,#number      ; move a number into an 8-bit
                                 ; register R0
TAKE:    DJNZ    R0,TAKE         ; keep decrementing R0 until it is
                                 ; zero
         RET                     ; return from DELAY subroutine
```

MOV takes 6 clock cycles, DJNZ and RET each take 12 clock cycles. The delay is called up from the main program using ACALL, which takes 12 clock cycles. The delay time is $(12 + 6 + (\text{number} \times 12) + 12)$ clock cycles. When the number is small the NOPs (total 24 cycles) should be included,

$$\text{Delay time} = (24 + 12 + 6 + (\text{number} \times 12) + 12) \text{ clock cycles}$$
$$\text{Delay time} = (54 + (12 \times \text{number})) \text{ clock cycles}$$

Example 1.15

A P89C664 microcontroller has an 11.0592 MHz crystal-controlled clock oscillator. Write an assembly language program that will generate a 5 kHz square-wave signal on pin 7 of port 1 when a switch causes pin 0 on the same port to go to logic 1.

Solution

Clock frequency = 11.0592 MHz
Thus period of clock cycle = (1/11.0592 MHz) = 90.423 ns
Signal frequency = 5 kHz
Therefore period of signal cycle = (1/5 kHz) = 200 μs

The delay required is half of this value since the square wave has an equal logic 1/logic 0 time. See Figure 1.14.

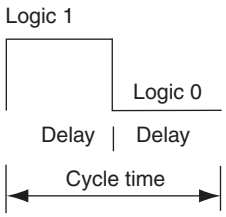

Figure 1.14 Delay period determination for a square-wave signal

$$\text{Delay} = 100\,\mu\text{s} = (54 + (12 \times \text{number})) \times 90.423\,\text{ns}$$

Hence number = ((100 μs/90.423 ns)−54)/12 = 88 decimal (to the nearest whole number).

```
$INCLUDE (REG66X.INC)        ; lists all sfr addresses
         ORG    0            ; sets start address to 0
         SJMP   START        ; short jump to START label
         ORG    0040H        ; puts next program line at address 0040H
START:   JB     P1.0,PULSE   ; jump to PULSE if pin 0, port 1 is logic 1
         CLR    P1.7         ; otherwise clear pin 7, port 1 to zero
         SJMP   START        ; go to START check switch
PULSE:   SETB   P1.7         ; set pin 7 on port 1 to logic 1
         ACALL  DELAY
         NOP                 ; hold logic 1 on pin 7 port 1
         NOP
         NOP
         NOP
         CLR    P1.7         ; clear pin 7 on port 1 to logic 0
         ACALL  DELAY
         AJMP   START        ; go to START check switch
DELAY:   MOV    R0,#88
TAKE:    DJNZ   R0,TAKE
         RET
         END                 ; no more assembly language
```

The delay depended on the chosen microcontroller clock frequency and in the example this was 11.0592 MHz. This apparently unusual number gives standard baud rate values, which will be useful later. For microcontroller clock frequencies in this region the single loop register decrement method gives delays in the region of microseconds. Generally a double loop gives delays in the region of milliseconds and a triple loop delay gives delays in the region of seconds.

Exercise 1.17
Using the techniques above, assuming the clock frequency is 11.0592 MHz, write a program to generate a pulse of 20 kHz on pin 7 of port 1 of the microcontroller.

DOUBLE LOOP DELAY

```
DELAY:  MOV   R1,#number1
INNER:  MOV   R0,#number2
TAKE:   DJNZ  R0,TAKE
        DJNZ  R1,INNER
        RET
```

Approximately the time delay = (number 1) × (number 2) × 12 clock cycle periods. For example, suppose number 1 = 200 and number 2 = 240 and 1 clock cycle = 90.423 ns.

$$\text{Time delay} = 200 \times 240 \times 12 \times 90.423\,\text{ns} = 52.1\,\text{ms}$$

The bigger the values of number 1 and number 2, the better the approximation. The software used has simulation and the values of number 1 and number 2 can be fine tuned to give the accurate delay during simulation.

TRIPLE LOOP DELAY

```
DELAY:  MOV   R2,#number1
OUTER:  MOV   R1,#number2
INNER:  MOV   R0,#number3
TAKE:   DJNZ  R0,TAKE
        DJNZ  R1,INNER
        DJNZ  R2,OUTER
        RET
```

Approximately the delay = (number 1) × (number 2) × (number 3) × 12 clock cycle periods. Suppose number 1 = 40, number 2 = 200, number 3 = 240, 1 clock cycle period = 90.423 ns.

$$\text{Delay} = (40 \times 200 \times 240 \times 12 \times 90.423)\,\text{ns} \approx 2\,\text{s}$$

Long enough to see a LED going on and off.

In later chapters the use of the microcontroller's onboard timers will be used to describe an alternative method of producing time delays. The timer method will require the configuration of the timer SFRs.

The register decrement method described above is a valid alternative, easy to implement and does not require the configuration of SFRs.

Summary

- A microcontroller is a computer with most of the necessary support chips onboard. Microcontrollers can be embedded and are available in a variety of forms to suit practical applications.
- Number systems, such as binary and hexadecimal, are used in microcontroller applications. If decimal numbers are required they can be converted to binary and/or hexadecimal and vice versa.
- There are four classes of instructions namely: arithmetic, logical, data transfer and branch instructions.
- The microcontroller port pins may be required to sink and source currents.
- Time delays may be achieved by using register decrement instructions or by using onboard timer circuits.
- Using register decrement, longer delays can be achieved by the use of double or triple loops.