1 8051 Basics

This chapter describes the basic 8051 micro-controller and explains its internal organization and uses of the internal special function registers. Many web pages, books (see bibliography list [1], [2], [3], [8], [9], [13], [14], [15], [18]), and tools are available for the 8051 developer, and many of them are free!. This chapter will assist the reader in mastering basic 8051 programming (using both assembly language and C language) and should eliminate the need to have an additional book specifically on the 8051.

1.1 Introduction

Despite its relatively old age, the 8051 (developed by Intel Corporation in the early 1980s) is one of the most popular micro-controllers in use today. Many derivative micro controllers have since been developed that are based on and compatible with the 8051. Thus, the ability to program an 8051 is an important skill for anyone who plans to develop products that will take advantage of most micro controllers.

The various sections of the first two chapters will explain the 8051micro-controller step by step. The sections in these chapters are targeted at students who are attempting to learn the 8051 assembly language programming and are also useful to those who prefer using C. The appendices are a useful reference tool that will assist both the novice programmer as well as the experienced professional developer, since they provide a wide range of programs complete with source code.

No knowledge of the 8051 is assumed; however, it is assumed some amount of programming has been done before with a basic understanding of the hardware and a firm grasp on the three numbering systems mentioned above. The concept of converting a number from decimal to hexadecimal and/or to binary is not within the scope of this book, and familiarity with these types of conversions would help in understanding some concepts.

This chapter attempts to address the need of the typical programmer. For example, there are certain features that are nifty and in some cases very useful, but 95% of the programmers will never use these features. Those already familiar with the 8051 may skim over some details described in this chapter.

The basic 8051 is a 40-pin IC as shown in Figure 1-1.

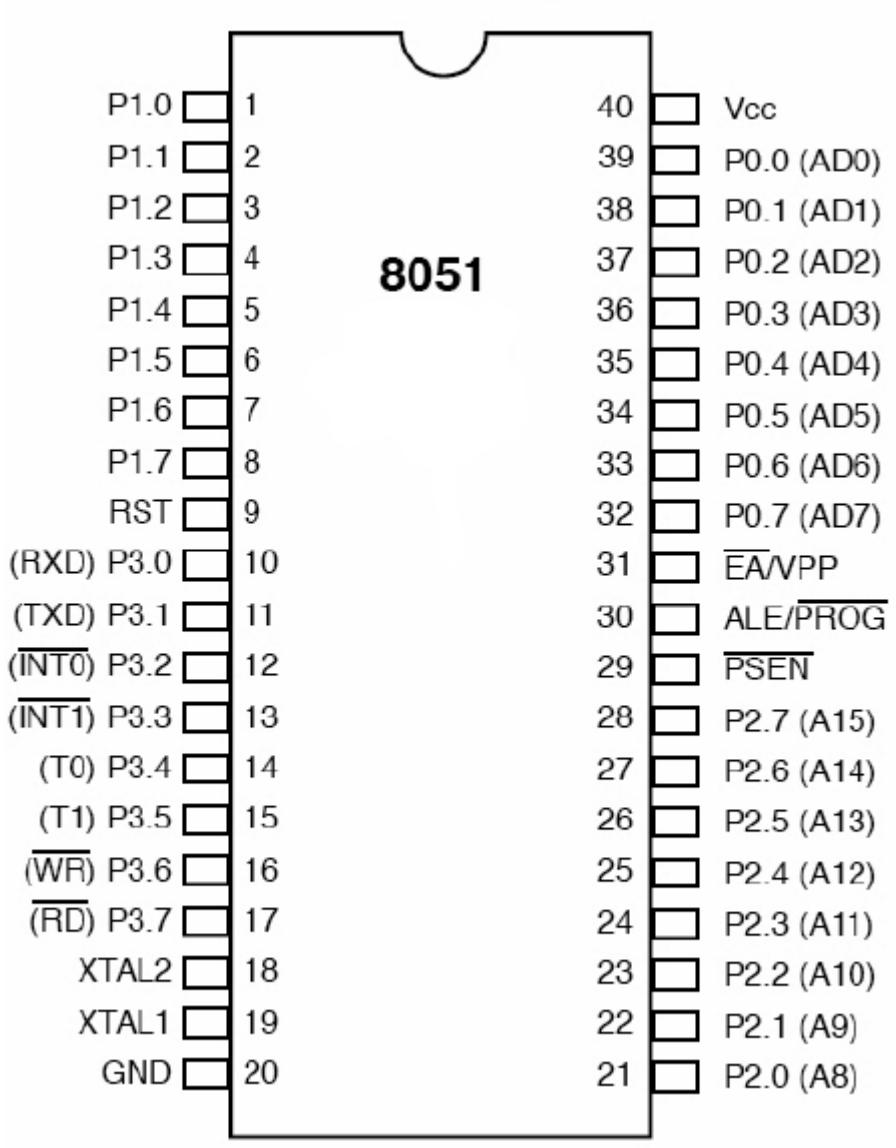

Figure 1-1 Basic 8051

We shall now deal with the internal organisation of the 8051 micro-controller.

1.2 Memory Types

The 8051 has three very general types of memory and each type has to be addressed in a different way. To effectively program the 8051 it is necessary to have a basic understanding of these memory types and how to address them, especially when programming directly in assembly language. The memory types found on the 8051 are illustrated in Table 11 namely the On-Chip Memory, the External Code Memory and External Data RAM. Addresses throughout this book are shown suffixed either with a lower case h (i.e. 0Fh) or with a upper case H (i.e. 0FH) to signify that they are hexadecimal numbers.

Higher memory non existent	FFFFH	FFFFH
0FFH Not Available on basic 8051 80H	External Code Memory	External Data Memory
7FH Internal On-Chip Memory 00H	0000H	0000H

Table 1-1 8051 memory space

It is also very common, especially in many development boards, that the external ram is organised as a contiguous memory map, made up as shown in Table 1-2. Generally, the EEPROM (or ROM) would occupy the lower address area, since the 8051 starts executing instructions from location number 0000H.

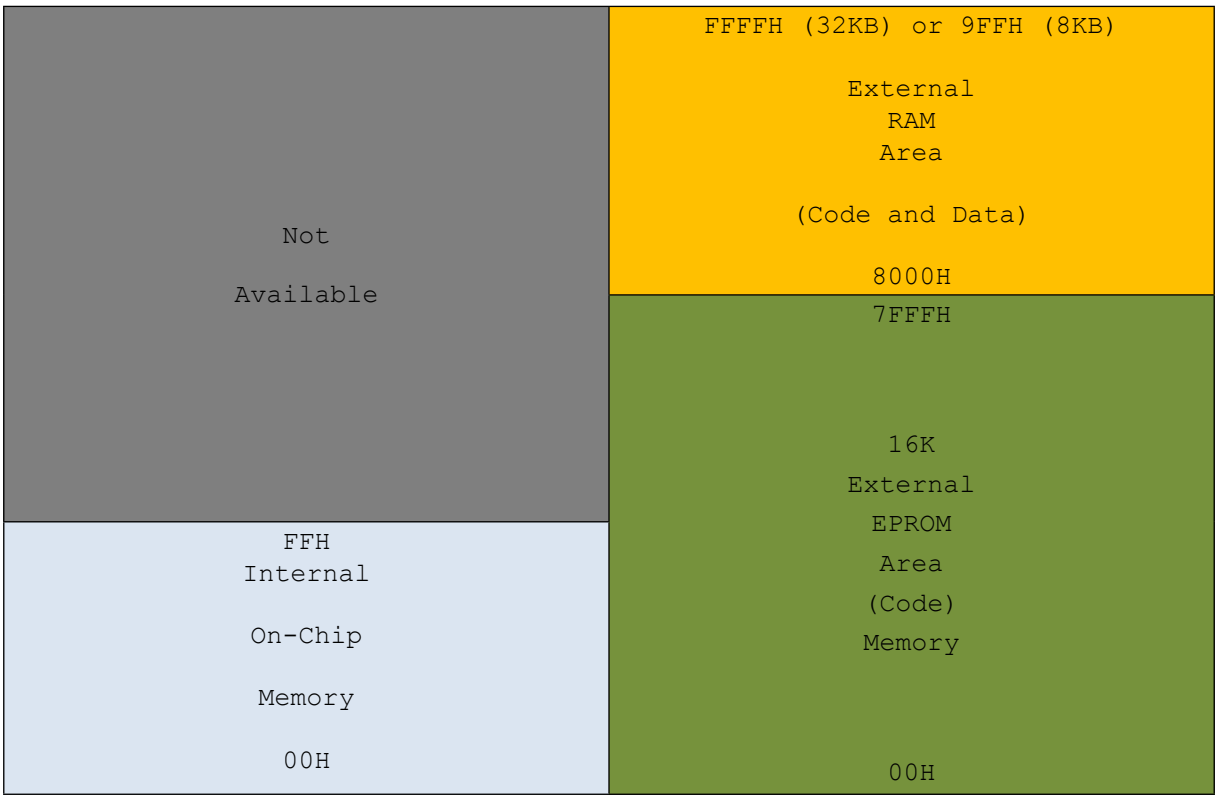

Table 1-2 8032 memory map (Development System)

The EEPROM would generally contain the monitor program so that the user can communicate with the board via the RS232, and also he would be able to transfer his own program into the upper RAM area, where it would be executed for testing and prototyping. The monitor program usually sets the serial port, perhaps under interrupt control (see section 2.9). It would also map the interrupt vector table into the RAM area so that the user application can make use of interrupts by having access to the interrupt vector table. If the interrupt vector table is left in the ROM, the user would not be able to write the address of his Interrupt Service Routines (ISRs) in the EEPROM directly and easily, (he would have to burn a new EEPROM each time! Recent versions with Flash memory have eliminated this problem.). The monitor system must at least have enough commands to be able to transfer and run the program. More commands are usually available depending on the sophistication required. Some have built-in assemblers, dis-assemblers, step-by-step execution and trace facilities for de-bugging purposes.

Most common memory set-ups involve an 8KB or a 16KB EPROM and at least 8KB RAM, both of which can be expanded. The address range would normally be selected by means of shorting some links. The internal memory on chip is only 128 bytes for the normal 8051 (see Table 1-1) but is doubled on the 8032 to 256 bytes as shown in Table 1-2.

On-Chip Memory refers to any memory (Code, RAM, or other) that physically exists on the micro controller itself. On-chip memory can be of several types, but we'll get into that shortly.

External Code Memory is code (or program) memory that resides off-chip. This is often in the form of an external EEPROM.

External RAM is RAM memory that resides off-chip. This is often in the form of standard static RAM or flash RAM.

1.3 Code Memory

Code memory is the memory that holds the actual 8051 program that is to be executed. This memory is normally limited to 64KB although it comes in many shapes and sizes. Since there are many variants of the basic 8051 the Code memory may be found in various forms depending on the device. It can either be burned into the micro-controller as ROM or as an EEPROM and it may also be stored completely off-chip in an external ROM or, more commonly in basic versions, as an external EEPROM. Flash memory is also another popular method of storing a program or code. Various combinations of these memory types may also be used, that is to say, it is possible to have 4KB of code memory on-chip and 64KB of code memory off-chip in an EEPROM.

When the program is stored on-chip, the 64KB maximum value is often reduced to 4KB, 8KB, or 16KB. This varies depending on the version of the micro-controller that is being used. Each version offers specific capabilities and one of the distinguishing factors from one chip to another is how much ROM/EEPROM space the chip has. 64KB and even 128KB flash eprom devices are now available, such as the Silicon Labs C8051F020.

However, code memory is most commonly implemented as off-chip EEPROM, in low-cost development systems and in systems developed by students.

Speeds (and hence performance) are also rapidly increasing with improved architecture and now we have high-speed devices running at 40MHz and using only one clock cycle per instruction instead of the original twelve clock cycles found on the early devices.

1.4 External RAM

As an obvious opposite of Internal RAM, the 8051 also supports what is called External RAM.

As the name suggests, External RAM is any random access memory which is found off-chip. Since the memory is off-chip it is not as flexible in terms of accessing, and is also slower. For example, to increment an Internal RAM location by 1 (such as INC R1) requires only one instruction which is executed in one instruction cycle. To increment a 1-byte value stored in External RAM requires four instructions which are executed in seven instruction cycles. In this case, external memory is seven times slower!

```
MOV DPTR, #address      (2 instruction cycles)
MOVX A, @DPTR           (2 instruction cycles)
INC A                   (1 instruction cycle)
MOVX @DPTR, A           (2 instruction cycles)
```

What External RAM loses in speed and flexibility it gains in quantity. While the Internal RAM is limited to 128 bytes (256 bytes with an 8032/8052), the 8051 supports an External RAM of up to 64KB.

Modern devices now also have this so-called external RAM, physically residing on the same chip, but it is still referred to as external (or XDATA) and all the information listed in this book still holds.

1.4.1 On-Chip Memory

As mentioned at the beginning of this chapter, the 8051 includes a certain amount of on-chip memory. On-chip memory is really one of two types: Internal RAM usually used to store variable and Special Function Register (SFR) memory, used to store the registers which control the built-in peripherals. The layout of the 8051's internal memory is presented in the memory map shown in Table 1-3.

The 8051 has a bank of 128 bytes of Internal RAM. This Internal RAM is found on-chip on the 8051 so it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying its contents. Internal RAM is volatile, so that when the 8051 is reset this memory is cleared.

Hex Byte Address	Notes		Hex Byte Address
Not Available On the Basic 8051	(8032 ONLY) Accessible By Indirect Addressing only	SFR area Accessible By Direct Addressing only	FFH 80H
7FH Lower 128 bytes 00H	Accessible By Direct And Indirect Addressing.		

Table 1-3 8051 Total Internal RAM organisation

The 128 bytes of internal ram is subdivided as shown on the memory map in Table 1-4. The first eight bytes (00h – 07h) are referred to as register bank 0. By manipulating certain SFR bits (in the PSW special function register), a program may choose to use register banks 1, 2, or 3. These alternative register banks are located in internal RAM, occupying addresses 08h through 1Fh. We will discuss register banks in more detail in section 1.5. For now it is sufficient to know that they are part of the internal RAM.

Bit Memory is also another part of internal RAM, which as the name implies is able to store and manipulate bit variables. We will say more about the bit memory area later (see section 1.6), but for now we just have to keep in mind that the bit memory actually resides in internal RAM, ranging from address 20h through address 2Fh.

The 80 bytes that remain in Internal RAM, from address 30h through address 7Fh, may be used to store any user variables that need to be accessed frequently or at high-speed during the execution of the program. This area is also utilised by the micro-controller as a storage area for the operating stack.

Hex Byte Address	Hex Bit Address								Notes
7FH 30H	Directly and Indirectly Addressable General Purpose RAM								Used as a STACK Area and to store user variables
2FH	7F	7E	7D	7C	7B	7A	79	78	Bit Addressable Section (Bit Addresses shown are in hex)
2EH	77	76	75	74	73	72	71	70	
2DH	6F	6E	6D	6C	6B	6A	69	68	
2CH	67	66	65	64	63	62	61	60	
2BH	5F	5E	5D	5C	5B	5A	59	58	
2AH	57	56	55	54	53	52	51	50	
29H	4F	4E	4D	4C	4B	4A	49	48	
28H	47	46	45	44	43	42	41	40	
27H	3F	3E	3D	3C	3B	3A	39	38	
26H	37	36	35	34	33	32	31	30	
25H	2F	2E	2D	2C	2B	2A	29	28	
24H	27	26	25	24	23	22	21	20	
23H	1F	1E	1D	1C	1B	1A	19	18	
22H	17	16	15	14	13	12	11	10	
21H	0F	0E	0D	0C	0B	0A	09	08	
20H	07	06	05	04	03	02	01	00	
1FH 18H	Register Bank 3 (R0 – R7)								Bank is Selected Using RS0 and RS1 In the PSW Register. See SFRs.
17H 10H	Register Bank 2 (R0 – R7)								
0FH 08H	Register Bank 1 (R0 – R7)								
07H 00H	Register Bank 0 (R0 – R7)								

Table 1-4 8051 Internal RAM organisation

The stack is used to save return addresses when calling functions or subroutines. It is also used to store some values temporarily until they are retrieved again when needed. The fact that the stack size is rather small severely limits the 8051's stack use since, as illustrated in the memory map of Table 1-4, the area reserved for the stack is only 80 bytes, and usually it is effectively a little bit less since these 80 bytes have to be shared between the stack and user variables.

1.5 Register Banks

The 8051 uses eight so-called R registers which are used in many of its instructions. These R registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6, and R7) and are generally used to assist in manipulating values and moving data from one memory location to another. For example, to add the value of R4 to the Accumulator, we would execute the following instruction:

```
ADD A,R4
```

Thus if the Accumulator (A) contained the value 6 and R4 contained the value 3, the Accumulator would contain the value 9 after this instruction was executed.

However, as the memory map of Table 1-4 shows, register R4 is really part of Internal RAM. Specifically, R4 (of bank 0) is located at address 04h. Thus the above instruction accomplishes the same thing as the following operation:

```
ADD A,04h
```

This instruction adds the value found in Internal RAM address 04h (the contents of location 04h) to the value of the Accumulator, leaving the result in the Accumulator. Since R4 is really residing in Internal RAM address 04h, the above instruction has therefore effectively accomplished the same thing as the ADD A,R4 instruction.

But we must be careful since as the memory map shows, the 8051 has four distinct register banks. When the 8051 is first booted up, register bank 0 (addresses 00h through 07h) is used by default. However, our program may instruct the 8051 to use one of the alternate register banks; i.e., register banks 1, 2, or 3. In this case, R4 will no longer be in Internal RAM address 04h but somewhere else. For example, if our program instructs the 8051 to use register bank 3, register R4 will now be located at Internal RAM address 1Ch (see Table 1-4).

The concept of register banks adds a great level of flexibility to the 8051, especially when dealing with interrupts, where we can allocate a specific register bank to a particular interrupt, so as not to corrupt other main program information stored in another bank of registers. (we shall cover interrupts in more detail later, see section 2.9). However we must always remember that the register banks really reside in the first 32 bytes of Internal RAM.

1.6 Bit Memory

The 8051, being a communications-oriented micro-controller, gives the user the ability to access a number of bit variables. These variables may only take the value of either 1 or 0.

There are 128 bit variables available to the user (see Table 1-4), individually numbered 00h through 7Fh. We may make use of these variables with assembly language commands such as *SETB bit address* and *CLR bit address*. For example, to set bit number 24 (hex) to 1 we would execute the instruction:

```
SETB 24h
```

It is important to note that the Bit Memory area is really a part of the Internal RAM. In fact, the 128 bit variables occupy the 16 bytes of Internal RAM from address 20h through address 2Fh. Thus, if we write the value FFh to Internal RAM address 20h we have effectively set bits 00h through 07h to 1 with just one instruction. That is to say that:

```
MOV 20h, #0FFh
```

is equivalent to the following 8 instructions, where we are setting the bits one at a time:

```
SETB 00h
SETB 01h
SETB 02h
SETB 03h
SETB 04h
SETB 05h
SETB 06h
SETB 07h
```

As illustrated in Table 1-4, the bit memory is not a new type of memory but it is just a subset of Internal RAM. Since the 8051 provides special instructions to access these 16 bytes (or 128 bits) of memory on a bit by bit basis it is useful to think of it as a separate type of memory. However, since it is just a subset of Internal RAM then we must remember that any operations performed on the Internal RAM can change the values of these bit variables.

Bit variables 00h through 7Fh are for user-defined variables used in the program. These are not the only bit variables available on the 8051. Other bits in certain Special Function Registers (SFRs) can also be addressed individually as explained in the next section. These bits variables have an address of 80h or higher and are actually used to access certain Special Function Registers (SFRs) on a bit-by-bit basis so as to program and control certain peripherals of the 8051. For example, if output lines P0.0 through P0.7 are all cleared (0) and we want to turn on the P0.0 output line (set bit 0 of port 0 to logic 1) we may either execute:

```
MOV P0,#01h
```

or

```
ORL P0,#01h              ; logically OR P0 with 00000001 binary
```

or

```
SETB 80h
```

or even

```
SETB P0.0                ; the assembler knows that P0.0 = 80h
```

All these instructions listed above accomplish the same thing, although there are some slight differences. Using the SETB or the ORL command will turn on (set to 1) the P0.0 line without affecting the status of any of the other P0 output lines. The MOV command effectively would indeed turn on (1) the P0.0 line but it would also turn off (0) all the other seven output lines (P0.1 to P0.7) which in some cases, may not be what is actually required. Hence caution has to be taken to ensure that we use the correct and most efficient method when setting or clearing bits.

1.7 Special Function Register (SFR) Memory

Special Function Registers (SFRs) reside in areas of internal memory that control specific functionality of the 8051 processor. For example, four SFRs permit access to the 8051's 32 input/output lines. Another SFR allows a program to read or write to the 8051's serial port. Other SFRs allow the user to set the serial baud rate, control and access timers and configure the 8051's interrupt system.

When programming, we may get the illusion that the SFRs are Internal Memory. This is because they are directly addressable. For example, if we want to write the value 1 to Internal RAM location 50 hex we would execute the instruction:

```
MOV 50h, #01h
```

Similarly, if we want to write the value 1 to the 8051's serial port we would write this value to the SBUF SFR, which has an SFR address of 99 Hex. Thus, to write the value 1 to the serial port we would execute the instruction:

```
MOV 99h,#01h or MOV SBUF,#01h
```

When using this method of memory access (called direct addressing mode), any instruction that has an address of 00h through 7Fh refers to an Internal RAM memory address while any instruction with an address of 80h through FFh refers to an SFR control register.

1.7.1 SFR Addresses

The 8051 is a flexible micro-controller with a relatively large number of modes of operations. In order to be able to make full use of these different modes or ways of using the built in peripherals of this versatile micro-control, our program may inspect and/or change the operating mode of the 8051 by manipulating the values of some specific 8051's SFRs.

They are accessed as if they were normal Internal RAM. The only difference is that Internal RAM for the 8051 resides from address 00h through 7Fh whereas the SFR registers exist in the address range of 80h through FFh. Each SFR has an address (80h through FFh) and a name.

Table 1-5a and 1-5b provide a graphical representation of the 8051's SFRs, their name, and their address in hexadecimal. Although the address range is from 80h through FFh, thus offering 128 possible addresses, there are only 21 SFRs in a standard 8051. The free locations are reserved for future enhanced and upgraded versions of the 8051 family, such as the 8032 discussed in Chapter 4. Moreover, reading data from these empty addresses will in general return some meaningless random data while writing data to these addresses will have no effect. In fact the actual memory cell of these free locations might not be physically present.

Hex Byte Address	Hex Bit Address								Symbol
FF – F9	Not implemented on chip								-
* F8 *	Not implemented on chip								-
F7 – F1	Not implemented on chip								-
* F0 *	F7	F6	F5	F4	F3	F2	F1	F0	B
EF – E9	Not implemented on chip								-
* E8 *	Not implemented on chip								-
E7 – E1	Not implemented on chip								-
* E0 *	E7	E6	E5	E4	E3	E2	E1	E0	ACC
DF – D9	Not implemented on chip								-
* D8 *	Not implemented on chip								-

Table 1-5a 8051 Special Function Registers (SFRs)-DIRECT addressing ONLY

Hex Byte Address	Hex Bit Address								Symbol
D7 – D1	Not implemented on chip								-
* D0 *	D7	D6	D5	D4	D3	D2	D1	D0	PSW
CF – C9	Not implemented on chip								-
* C8 *	Not implemented on chip								-
C7 – C1	Not implemented on chip								-
* C0 *	Not implemented on chip								-
BF – B9	Not implemented on chip								-
* B8 *	-	-	-	BC	BB	BA	B9	B8	IP
B7 – B1	Not implemented on chip								-
* B0 *	B7	B6	B5	B4	B3	B2	B1	B0	P3
AF – A9	Not implemented on chip								-
* A8 *	AF	-	-	AC	AB	AA	A9	A8	IE
A7 – A1	Not implemented on chip								-
* A0 *	A7	A6	A5	A4	A3	A2	A1	A0	P2
9F – 9A	Not implemented on chip								-
99									SBUF
* 98 *	9F	9E	9D	9C	9B	9A	99	98	SCON
97 – 91	Not implemented on chip								-
* 90 *	97	96	95	94	93	92	91	90	P1
8F –8E	Not implemented on chip								-
8D									TH1
8C									TH0
8B									TL1
8A									TL0
89									TMOD
* 88 *	8F	8E	8D	8C	8B	8A	89	88	TCON
87									PCON
86 – 84	Not implemented on chip								-
83									DPH
82									DPL
81									SP
* 80 *	87	86	85	84	83	82	81	80	P0

Hex addresses shown within asterisks are bit-addressable locations.

Table 1-5b 8051 Special Function Registers (SFRs)-DIRECT addressing ONLY

We should therefore stick to the rule that any user developed software should not write anything to these unimplemented locations, since they may be used in future products to invoke new features. All unimplemented addresses in the SFR range (80h through0 FFh) are considered invalid and writing to or reading from these non-existent register locations may produce undefined values or behaviour.

1.7.2 SFR Types

As mentioned in Table 1-5 itself, some SFRs (P0, P1, P2 and P3) are SFRs related to the I/O ports. The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by these SFRs. It should be noted that all of these ports are Bit-addressable. This means that we can read from or write to a single bit of any port.

Some other SFRs are used to control the operation or the configuration of some aspect of the 8051. For example, TCON and TMOD control the timers while SCON controls serial port operations.

The other remaining SFRs can be thought of as auxillary SFRs in the sense that they do not directly configure the 8051 but obviously the 8051 cannot operate without them. For example, once the serial port has been configured using SCON, the program may read or write data characters or bytes to the serial port using the SBUF register.

The SFRs whose address has an asterisk (*) in the Table 1-5 above, are special SFRs that may also be accessed via bit operations (i.e., using the SETB and CLR instructions). The other SFRs cannot be accessed using bit operations but have to be handled using byte operations. As we can see, all SFRs whose addresses are divisible by 8 (having an address ending with a 0h or 8h) can be accessed with bit operations, meaning that they are bit-addressable.

1.8 SFR Descriptions

This section will endeavour to quickly overview each of the standard SFRs found in the above SFR chart map (Table 1-5). It is not the intention of this section to fully explain the functionality of each SFR, as this information will be covered in separate dedicated sections of this chapter.

1.8.1 P0 (Port 0, Address 80h, Bit-Addressable)

All four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output ports. To use any bit of these ports as an input port bit, it must be programmed to do so, by writing a 1 to that particular bit. The operation of the ports is well explained in ([9] Mazidi & Mazidi 2000, pp. 384–390), and is being reproduced with some added comments in the following paragraphs dealing with the ports.

The structure of input/output port 0 or P0, is shown in Figure 1-2. Each bit of this SFR corresponds to one of the port pins on the micro-controller. For example, bit 0 of port 0 is pin P0.0, bit 7 is pin P0.7. Writing a value of 1 to a bit (SETB P0.7) of this SFR will send a high level on the corresponding I/O pin whereas writing a value of 0 (CLR P0.7) will bring it to a low level. If used as an input, the status of a bit can be checked by the program by using for example:

```
JB P0.7,Label ; (Jump to Label if bit P0.7 is 1).
```

or in C, assuming that Port0_bit7 was declared by ujsing the **sbit** bit variable declaration

```
sbit Port0_bit7 = P0^7;
```

then we can write

……

```
if (Port0_bit7 == 1) { …… }
```

or

```
if (Port0_bit7) { …… }
```

The **sbit** variable declaration enables us to gain access to the port pin by giving the pin a name. The caret symbol (^) is used in C instead of the dot (.) when referring to bits, since the dot is used in C when referring to union members. Although the caret symbol is also used in C for the bitwise XOR (exclusive OR) operator, no XOR operation is involved here. The sbit keyword helps the compiler to sort this out.

To use the pins of port 0 as both input and output ports, each pin must be connected externally to a +5V rail via a 10k ohm pull-up resistor. This is due to the fact that P0 uses an open drain configuration, unlike P1, P2 and P3. *Open drain* is a term used for MOS chips in the same way that an *open collector* is used for TTL chips. With external pull-up resistors connected, upon reset port 0 is configured as an output port (default mode). This setup is also shown in Figure 1-2 with the pull-up resistor shown in a shaded box to highlight the fact that this is an additional external connection.

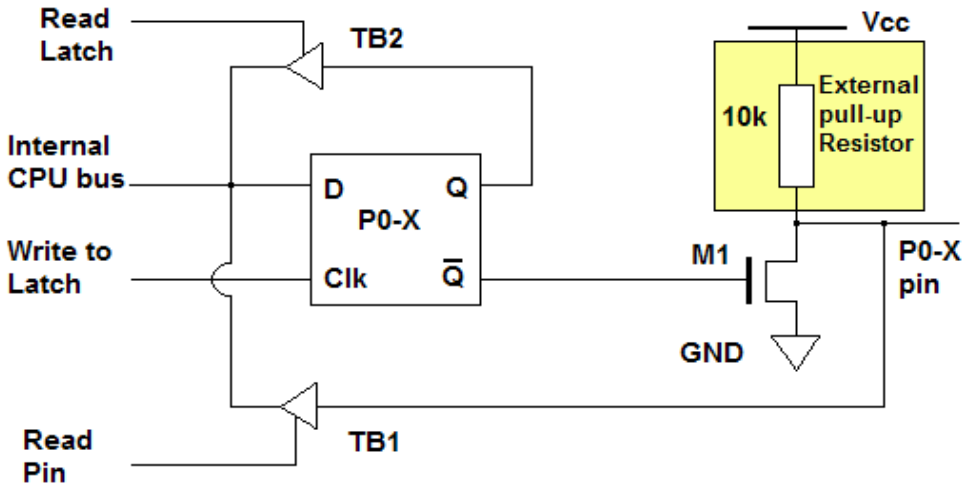

Figure 1-2 Pull-Up resistors

With resistors connected, in order to make it an input port, the port must first be programmed to the input mopde. This is achieved by writing a 1 to all the bits required to act as an input. For example to make all port 0 act as an input port, we must first use:

```
MOV P0, #0FFH
```

or in C

```
P0 = 0xFF;
```

And then we can read data from the port into the accumulator by using

```
MOV A, P0
```

or in C, assuming Inputdata was previously declared as an 8-bit variable

```
Inputdata = P0;
```

Hex Byte Address	Bit-addressable								Symbol
80	87	86	85	84	83	82	81	80	P0
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	Bit – ASM
	P0^7	P0^6	P0^5	P0^4	P0^3	P0^2	P0^1	P0^0	Bit – KEIL C

Table 1-6 P0

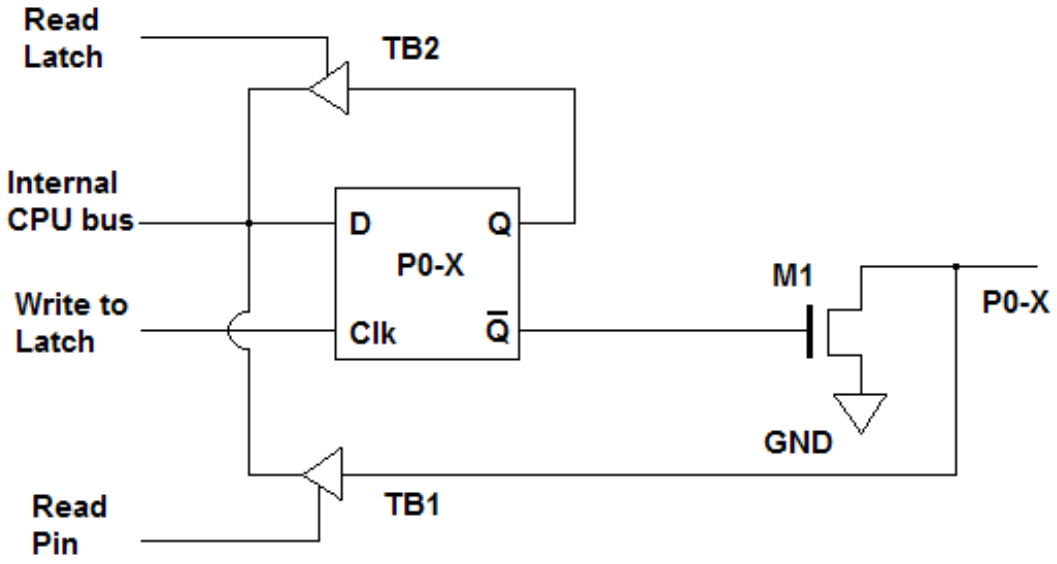

Figure 1-3 8051 Port 0 Structure

Port 0 has a dual role, allowing it to be used for both address and data transfers. When connecting the 8051 to an external memory, port 0 provides both the address and the data signals. The 8051 multiplexes address and data functions through port 0 in order to save on the number of pins on the IC, the Address Latch Enable (ALE) pin providing the necessary control function. If ALE = 0, port 0 provides data and when ALE = 1 it carries address bits A0 to A7.

Since all the ports of the 8051 are bi-directional, they all have the following three basic components:

- D latch
- Output driver
- Input buffer

Now the question arises when reading the port, are we reading the status of the input pin or are we reading the status of the latch? That is an extremely important question and its answer depends on the type of instruction we are using to address the port. The instruction itself would dictate which tri-state input buffer is to be activated, whether TB1(pin) or TB2 (latch). The explanation is given in section 1.8.5.

1.8.2 Reading the input pin

As stated earlier, in order to make any bit of any port of the 8051 an input port, we first must write a 1 (logic high) to that port bit. With reference to Figure 1-4, since we have chosen port 0, the load R1 would be an externally connected pull-up resistor of say 10kΩ (shown in a shaded box to denote an additional external component).

Writing a 1 to the port bit, causes a 1 to be written to the latch and the D latch therefore has a logic high on its pin. Therefore Q = 1 and $\overline{Q} = 0$.

Consequently the transistor M1 gate is 0 or at a low level and the transistor is therefore turned off.

M1 therefore blocks the path to ground for any signal connected to the input pin P0.X and the input signal is therefore directed to the tri-state buffer TB1.

When reading the input port with an instruction such as using MOV A, P1 we are therefore actually reading directly the data present at the pin since this instruction activates the read pin of TB1 and lets the data flow into the CPU's internal bus.

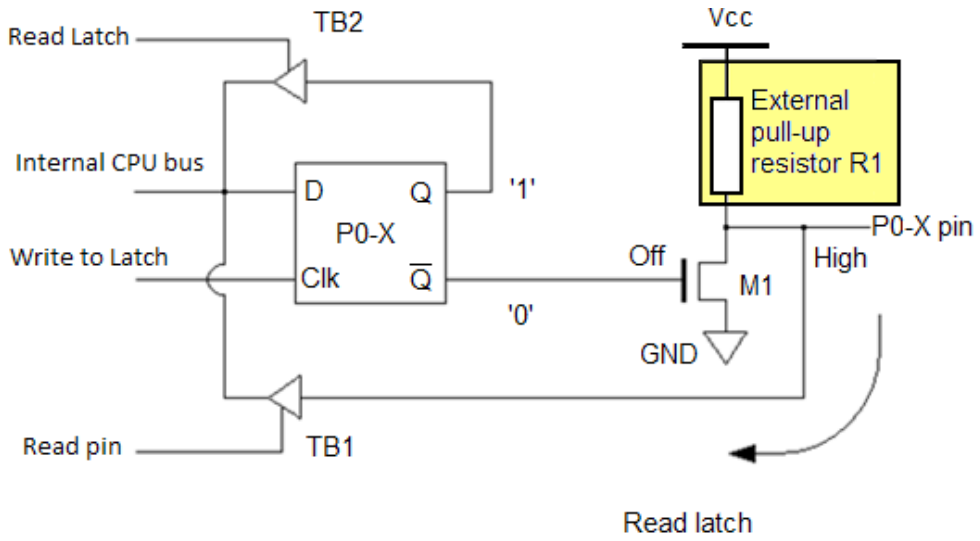

Figure 1-4 Setting 8051 Port 0.X as an input pin

Writing to a pin which was previously set to the input mode can have serious repercussions on the port and this is dealt with in section 1.8.4 where we explain how we can damage the port. This particular situation can easily occur if we are not careful when programming the device.

1.8.3 P1 (Port 1, Address 90h, Bit-Addressable)

This is input/output port 1 as shown in Figure 1-5. Each bit of this SFR corresponds to one of the pins on the micro-controller. For example, bit 0 of port 1 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit (SETB P1.7) of this SFR will send a high logic level (say 3.5–5 V) on the corresponding I/O pin whereas a value of 0 (CLR P1.7) will bring it to a low logic level (0V). If used as an input, the status of a bit can be checked by the program by using for example

```
JB P1.7,Label ;(Jump to Label if bit P1.7 is 1).
```

or in C, again assuming the sbit variable declaration

```
sbit Port1_bit7 = P1^7;
```

….

```
If (Port1_bit7 == 1) { … … }
```

As seen in Figure 1-5, in contrast to port 0, this port does not need any pull-up resistors since they are already built-in internally. However, once again as in port 0, in order to make it an input port, the port must first be programmed by writing a 1 to all the bits required to act as an input.

Hex Byte Address	Bit-addressable								Symbol
90	97	96	95	94	93	92	91	90	P1
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	Bit – ASM
	P1^7	P1^6	P1^5	P1^4	P1^3	P1^2	P1^1	P1^0	Bit – KEIL C

Table 1-7 P1

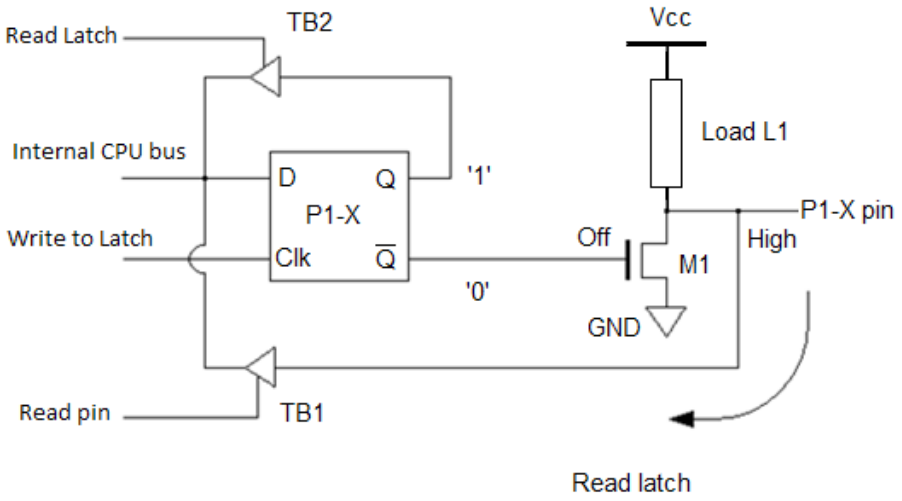

Figure 1-5 8051 Port 1 Structure, with internal load

1.8.4 Damaging the port

Looking at Figure 1-6 we can see that if we write a 0 (low) to a port bit, then $Q = 0$ and $\overline{Q} = 1$. As a result transistor M1 is now ON and therefore provides a path to ground for load L1 and the pin P1.X is effectively grounded.

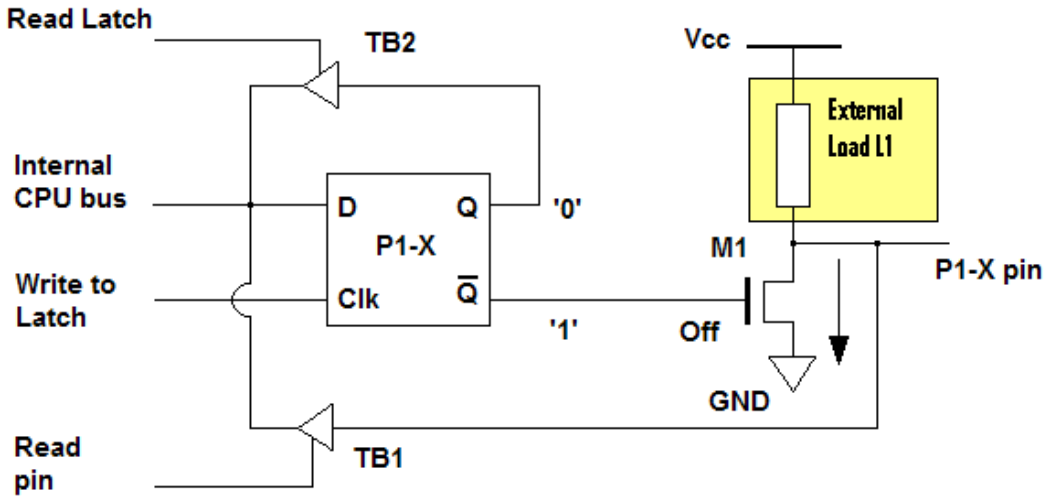

Figure 1-6 Writing '0' to a port

This is normal and correct, since when we write a zero to an output port, we expect to have 0V at the output pin. If however the port was originally intended to be used as an input port and we had an external connection as shown in Figure 1-7 the effect of inadvertently writing a '0' to an input configured port could have a very damaging effect. With the transistor switched on and if a two-way switch between supply Vcc and ground is connected directly to the pin as shown, then the transistor will sink current from both the internal load L1 and the external Vcc via the switch. This will cause too much current to flow in M1 and thus damaging permanently the port bit. In order to avoid damaging the port even if we use the wrong instruction by mistake, the correct kind of connection should be used when using switches or when supplying signals to an input port.

Some examples of the correct type of connection are shown in Figure 1-8 to Figure 1-11.

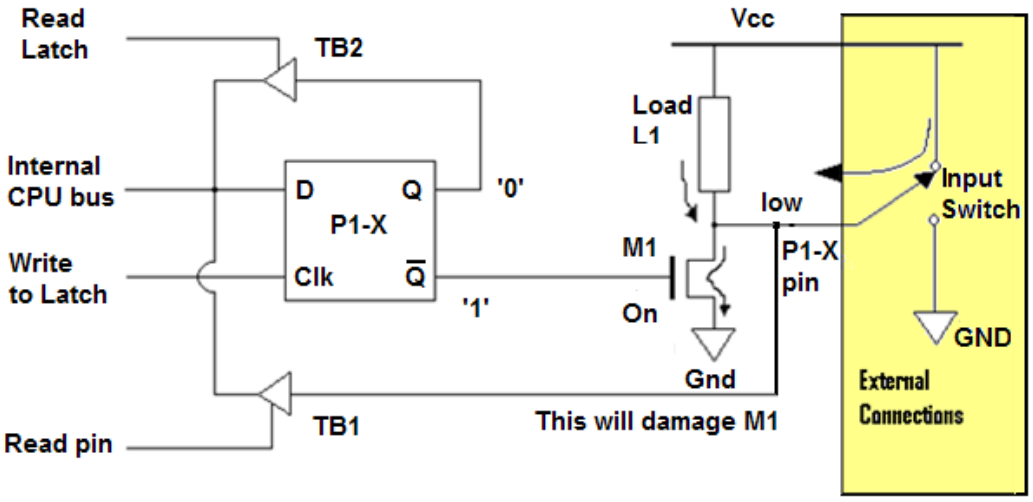

Figure 1-7 Never connect an input port pin directly to Vcc

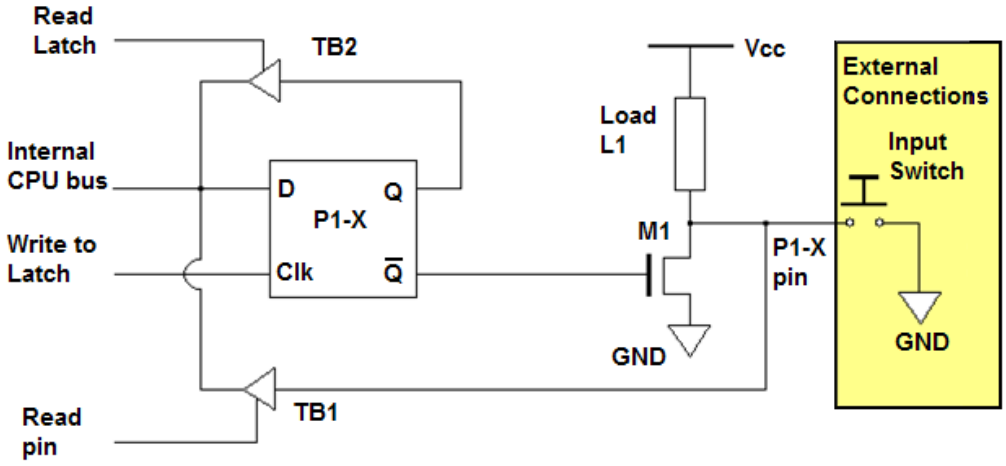

Figure 1-8 Input switch with no Vcc

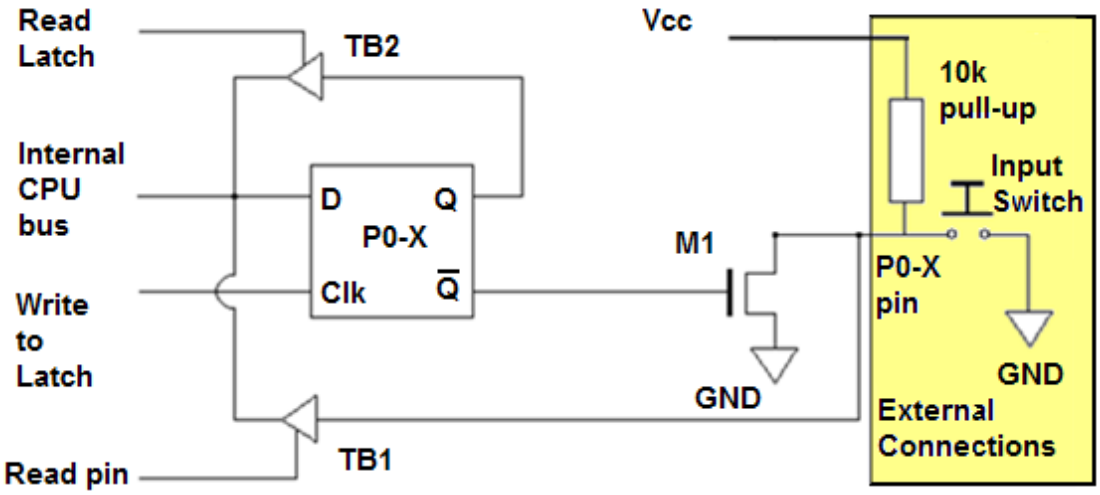

Figure 1-9 Input switch with pull-up resistor on Port 0

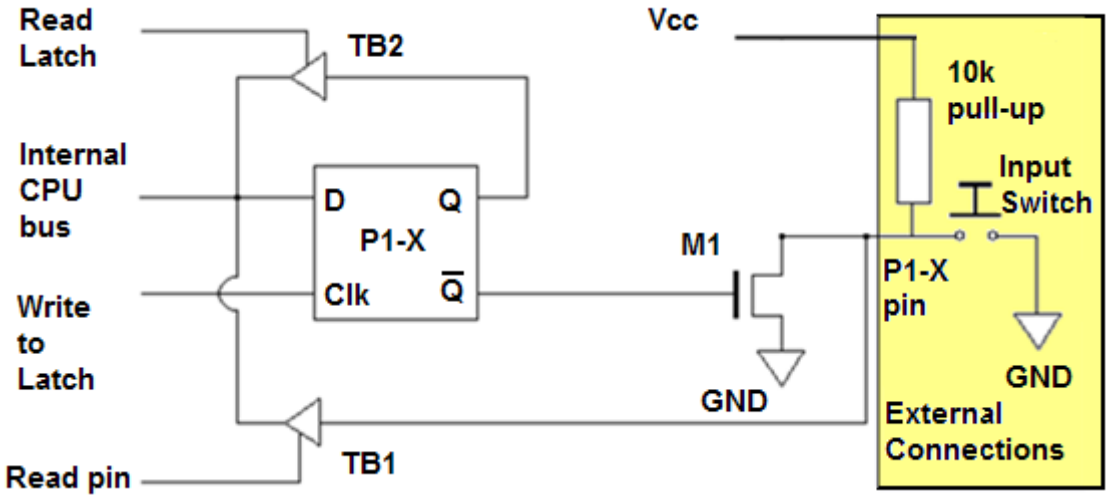

Figure 1-10 Input switch with pull-resistor on Port 1

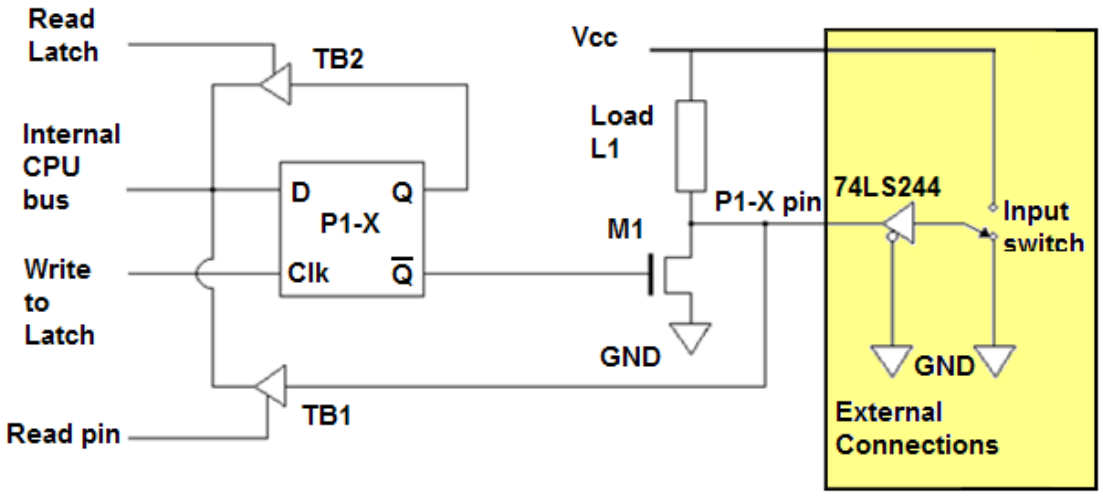

Figure 1-11 Buffering input switch connected directly to Vcc

1.8.5 Reading the latch

In reading the port, (see Figure 1-12) we may be reading the latch instead of the actual port pin. Consider the case of the logical AND instruction

```
ANL P1,A
```

which is actually a READ-MODIFY-WRITE instruction. This is typical for **bit-wise** operations such as ANL, ORL and XRL. There are usually no side-effects of READ-MODIFY-WRITE instructions when accessing registers that have the same values when read or when written (because they act like RAM). However, READ-MODIFY-WRITE instructions can cause problems when the register being accessed is write-only or reads a different value than what was written.

A characteristic of the I/O ports of the 8051 is that the value you write may not be the value you read (since reading the port returns the state of the port pins). However, these registers are specially treated for READ-MODIFY-WRITE instructions.

Looking at the sequence of actions taking place when this instruction (ANL P1,A) is executed, we shall see exactly why and what we are reading:

- The read latch in this case activates the tri-state buffer TB2 and brings the data from the Q latch (not the input pins via TB1 as in previous examples) into the CPU.
- This data is ANDed with the contents of register A.
- The result is re-written to the latch.

After re-writing the result to the latch there are two possibilities:

If $Q = 0$, then $\overline{Q} = 1$ and M1 is ON. The output pin has 0, the same as the status of the Q latch.

If $Q = 1$, then $\overline{Q} = 0$ and M1 is OFF. The output pin has a 1, the same as the status of the Q latch.

From the above discussion, we conclude that the instruction that reads the latch normally reads a value, performs an operation (possibly changing the value), and re-writes it to the latch. Hence this is often called a READ-MODIFY-WRITE instruction. Table 1-8 provides a list of examples for such instructions, which ALL use the port as the destination operand.

- ANL P1,A
- ORL P1,A
- XRL P1,A
- JBC P1.1, LABEL
- CPL P1.2
- INC P1
- DEC P1
- DJNZ P1, LABEL
- MOV P1.2,C
- CLR P1.3
- SETB P1.4

Table 1-8 Read-Modify-Write Instructions

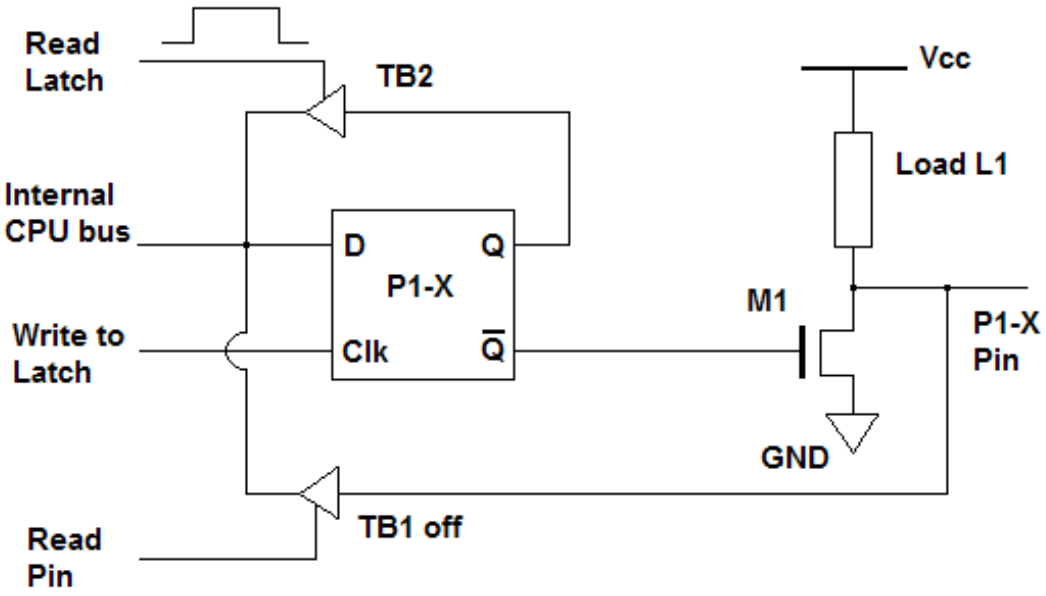

Figure 1-12 Reading the latch

1.8.6 P2 (Port 2, Address A0h, Bit-addressable)

This is input/output port 2. Each bit of this SFR corresponds to one of the pins on the micro-controller. For example, bit 0 of port 2 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit (SETB P2.7) of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 (CLR P2.7) will bring it to a low level. If used as an input, the status of a bit can be checked by the program by using for example:

```
JB P2.7, Label ; (Jump to Label if bit P2.7 is 1).
```

or in C, with a previous sbit declaration of the variable Port2_bit

```
if (Port2_bit7) { ……}
```

Same as port 1, this port does not need any pull-up resistors since they are already built-in internally. Also as in port 1, in order to make it an input port, the port must first be programmed by writing a 1 to all the bits required to act as an input.

Hex Byte Address	Bit-addressable								Symbol
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	Bit – ASM
	P2^7	P2^6	P2^5	P2^4	P2^3	P2^2	P2^1	P2^0	Bit – KEIL C

Table 1-9 P2

Port 2 as port 0, also has a dual role, allowing it to be used to provide the higher 8-bit address when connecting the 8051 to external memory. Used in conjunction with port 0 provides the address bits A8 to A15 thus making the 8051 capable of addressing up to 64KB (16-bit) of external memory.

1.8.7 P3 (Port 3, Address B0h, Bit-addressable)

This is input/output port 3 and each bit of this SFR corresponds to one of the pins on the micro-controller. For example, bit 0 of port 3 is pin P3.0, bit 7 is pin P3.7.

Hex Byte Address	Bit-addressable								Symbol
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
	RD	WR	T1	T0	INT1	INT0	TXD	RXD	Other use
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0	Bit – ASM
	P3^7	P3^6	P3^5	P3^4	P3^3	P3^2	P3^1	P3^0	Bit – KEIL C

Table 1-10 P3

Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas writing a value of 0 will bring it down to a low level.

P3 is also used for interrupts as well as other signals as shown in Table 1-10. Port 3 too, does not need any pull-up resistors, same as P1 and P2. Although port 3 is configured as an output port upon reset, this is not the way it is commonly used.

- Bits 0 and 1 are used for the RxD (input data) and TxD (output data) serial communications signals.
- Bits 2 and 3 are set aside for external interrupt input signals.
- Bits 4 and 5 can be used as input signals for the timers and
- Bits 6 and 7 can provide the write and read signals for any external memories connected to the 8051.

Thus P3 has some pins dedicated for specific jobs which restrict its use for other purposes.

1.8.8 SP (Stack Pointer, Address 81h)

This is the stack pointer of the micro-controller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If we push a value onto the stack, the value will be written to the address of SP + 1. That is to say, if SP holds the value 07h (this is the default reset value), a PUSH instruction will push the value onto the stack at address 08h. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the micro-controller.

1.8.9 DPL/DPH (Data Pointer Low/High, Addresses 82h/83h)

The SFRs DPL and DPH work together to represent a 16-bit value called the Data Pointer (DPTR) and is used in operations regarding external RAM and some instructions involving code memory. Having 16-bits it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

1.8.10 PCON (Power Control, Address 87h)

The Power Control SFR is used to control the 8051's power control modes. Certain operating modes of the 8051 allow the 8051 to go into a sort of sleep mode which requires much less power. These modes of operation are controlled through specific bits in PCON.Additionally, one of the bits in PCON (PCON.7 also known as SMOD) is used to double the effective baud rate of the 8051's serial port. Other bits are not implemented (-). Note that this SFR is not Bit-addressable, and hence in order to set SMOD to 1, without altering the other bits in the SFR, we should use:

```
ORL PCON, #80h
```

or in C

```
PCON |= 0x80;
```

Hex Byte Address	Not Bit-addressable								Symbol
	7	6	5	4	3	2	1	0	
87	-	-	-	-	-	-	-	-	PCON
	SMOD	-	-	-	GF1	GF2	PD	IDL	Bit

Table 1-11 PCON

1.8.11 TCON (Timer Control, Addresses 88h, Bit-addressable)

The Timer Control (TCON) SFR is used to configure and modify the way in which the 8051's two timers operate.

Hex Byte Address	Bit-addressable								Symbol
88	8F	8E	8D	8C	8B	8A	89	88	TCON
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	Bit Symbol
	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0	Bit – ASM
	TCON^7	TCON^6	TCON^5	TCON^4	TCON^3	TCON^2	TCON^1	TCON^0	Bit – KEIL C

Table 1-12 TCON

This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that the timer has overflowed. Additionally, some non-timer related bits are also located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occured.

1.8.12 TMOD (Timer Mode, Address 89h)

The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR, (see Table 113) our program may configure each timer to be a 16-bit timer, an 8-bit auto-reload timer, a 13-bit timer, or two separate timers. The timer can be used to count pulses from the internal clock ($C/\overline{T} = 0$) or to count events ($C/\overline{T} = 1$) connected to an external pin (P3.5 T1 for timer 1 or P3.4 T0 for timer 0). Additionally, in order to facilitate pulse-width measurements, we may configure the timers (setting the GATE bit to 1) to only start counting when an external pin (P3.3 INT1 for timer 1 or P3.2 INT0 for timer 0) is high. This is further explained in section 2.11.15.

Hex Byte Address	Not Bit-addressable								Symbol
89	–	–	–	–	–	–	–	–	TMOD
	GATE	C/	M1	M0	GATE	C/	M1	M0	Bit
	Timer 1				Timer0				

Table 1-13 TMOD

1.8.13 TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Ch)

These two SFRs, taken together, represent the timer 0 counting registers. Their exact behaviour depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value. Note that these two registers do not occupy consecutive address locations, and hence cannot be loaded together say by using an SFR16 data type variable in KEIL C. (see note on Little Endian / Big Endian in section 6.1).

1.8.14 TL1/TH1 (Timer 1 Low/High, Addresses 8Bh/8Dh)

These two SFRs, taken together, represent the timer 1 counting registers. As for timer 0, their exact behaviour depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value. Note that these two registers, same as TL0 and TH0, do not occupy consecutive address locations, and hence once again they cannot be loaded together say by using an SFR16 data type variable in KEIL C.

1.8.15 SCON (Serial Control, Address 98h, Bit-Addressable)

The Serial Control SFR is used to configure the behaviour of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags (TI and RI) that are set when a byte is successfully sent or received. These in turn can also be programmed to generate interrupts, thus providing the capability to have an interrupt controlled serial reception and/or transmission.

Hex Byte Address	Bit-addressable								Symbol
98	9F	9E	9D	9C	9B	9A	99	98	SCON
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	Bit Symbol
	SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0	Bit – ASM
	SCON^7	SCON^6	SCON^5	SCON^4	SCON^3	SCON^2	SCON^1	SCON^0	Bit – KEIL C

Table 1-14 SCON

1.8.16 SBUF (Serial Buffer Address 99h)

The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin (which is actually pin P3.1). Likewise, any value which the 8051 receives via the serial port's RXD pin (which is actually pin P3.0) will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from. Although SBUF has just one address, it is actually two separate registers, one activated by a READ instruction (to read a character which has been received) and the other activated by a WRITE instruction used to send the data which has to be transmitted. Simultaneous transmit and receive operations (full-duplex) can thus be handled.

Moreover, when data is received into SBUF, RI flag (bit SCON.0) is set. This may in turn be programmed to generate an interrupt, signaling that a character has been received which can then be read by the program. Similarly, when a character has been sent by the device, TI flag (bit SCON.1) is set which can also be programmed to trigger an interrupt. This would indicate that a character has been transmitted and thus SBUF can be loaded again which a fresh character for subsequent transmission. Both RI and TI trigger the same serial interrupt and therefore the Interrupt Service Routine would have to check which flag caused the interrupt (it may even be both of them at the same time!) and branch accordingly.

1.8.17 IE (Interrupt Enable, Addresses A8h)

The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, whereas the most significant bit (msb) is used to enable or disable ALL the interrupts. Thus, if the msb of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

Hex Byte Address	Bit-addressable								Symbol
A8	AF	AE	AD	AC	AB	AA	A9	A8	IE
	EA	-	ET2	ES	ET1	EX1	ET0	EX0	Bit Symbol
	IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0	Bit – ASM
	IE^7	IE^6	IE^5	IE^4	IE^3	IE^2	IE^1	IE^0	Bit – KEIL C

Table 1-15 IE

1.8.18 IP (Interrupt Priority, Address B8h, Bit-Addressable)

The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt can be of any one of two types. It may either be of a low (0) priority or a high (1) priority.

Hex Byte Address	Bit-addressable								Symbol
B8	BF	BE	BD	BC	BB	BA	B9	B8	IP
	-	-	PT2	PS	PT1	PX1	PT0	PX0	Bit Symbol
	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0	Bit – ASM
	IP^7	IP^6	IP^5	IP^4	IP^3	IP^2	IP^1	IP^0	Bit – KEIL C

Table 1-16 IP

An interrupt may only interrupt other interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing its service routine. However, if a serial interrupt service routine is executing then no other interrupt will be able to interfere with the serial interrupt service routine since the serial interrupt has the highest priority.

1.8.19 PSW (Program Status Word, Address D0h, Bit-Addressable)

The Program Status Word is used to store a number of important bits that are set and cleared by some of the 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags (RS1 and RS0) which are used to select which of the register banks is currently selected. Bits 3 and 4 of the PSW SFR determine which register bank is currently being used as shown in Table 1-18 Register Bank Selection bits. The default (at switch-on) reset value is bank 0 (RS0 = RS1 = 0).

Hex Byte Address	Bit-addressable								Symbol
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	CY	AC	F0	RS1	RS0	OV	-	P	Bit Symbol
	PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0	Bit – ASM
	PSW^7	PSW^6	PSW^5	PSW^4	PSW^3	PSW^2	PSW^1	PSW^0	Bit – KEIL C

Table 1-17 PSW

RS1	RS0	Register Bank	Address Range
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

Table 1-18 Register Bank Selection bits

1.8.20 ACC (Accumulator A, Address E0h, Bit-Addressable)

The Accumulator is one of the most used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction MOV A, #20h is really the same as MOV 0E0h, #20h. However, it is a good idea to use the first method since it only requires two bytes whereas the latter instruction requires three bytes.

Hex Byte Address	**Bit-addressable**								**Symbol**
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0	Bit – ASM
	ACC^7	ACC^6	ACC^5	ACC^4	ACC^3	ACC^2	ACC^1	ACC^0	Bit – KEIL C

Table 1-19 ACC

1.8.21 B (B Register, Address F0h, Bit-Addressable)

The B register is used specifically in two instructions: the multiply (MUL AB) and divide (DIV AB) operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values.

Hex Byte Address	**Bit-addressable**								**Symbol**
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0	Bit – ASM
	B^7	B^6	B^5	B^4	B^3	B^2	B^1	B^0	Bit – KEIL C

Table 1-20 B

1.8.22 Other SFRs

As we have already seen, Table 1-5 gives a summary of all the SFRs that exist in a standard 8051. All derivative micro-controllers of the 8051 must support these basic SFRs in order to maintain compatibility with the underlying MSCS51 standard.

A common practice when semiconductor firms wish to develop a new 8051 derivative is to add additional SFRs to support new functions that exist in the new chip. For example, the Dallas Semiconductor DS80C320 is upwards compatible with the 8051. This means that any program that runs on a standard 8051 should run without modification on the DS80C320. It also means that all the SFRs defined above apply to the Dallas device.

However, since the DS80C320 provides many new features or devices which the standard 8051 does not support, there must be some way to control and configure these new features. This is accomplished by implementing additional SFRs to those listed here. For example, since the DS80C320 supports two serial ports (as opposed to just one on the 8051), the SFRs SBUF2 and SCON2 have been added. In addition to all the SFRs listed above, the DS80C320 also recognizes these two new SFRs as valid and uses their values to determine the mode of operation of the secondary serial port. Obviously, these new SFRs have been assigned to SFR addresses that were unused in the original 8051.

In this manner, new 8051 derivative chips may be developed which will still run existing 8051 programs. This is also one of the reasons stated earlier, why SFR addresses which are not utilised on one 8051 type should not be used, so that the program would still be compatible with other 8051 versions.