Preface

This book is an outgrowth of the notes and experiments developed for the graduate classes at the University of Florida. It is intended for students, hobbyists, engineers, and scientists who would like to learn about the 8051 microcontroller. It takes a hands-on pragmatic approach with applications focused on mechatronics and robotics. The subject material and example software are interspersed to aid the fluidity of the learning experience. The examples build towards mechatronics applications, merging the mechanics and electronics. In this sense, the topic area is oriented towards systems integration, not only in mechanics and electronics, but also in hardware and software. DC motor control, sensing environmental variables such as light and temperature, and the various electronic circuitry given in the following chapters have immediate applications to robotics. The book concludes by building an autonomous mobile robot. The robot is viewed as an experimentation platform upon which the reader may add other hardware and software components.

This book should be read while the accompanying experiments are implemented concurrently. The reader needs access to an 8051 board, available from various manufacturers (see Appendix D for a list). Most of the experiments may be carried out with the standard 8051 microcontroller. A few examples use enhanced members of the family to illustrate some newer peripherals. The software is developed with the READS51 integrated development environment from Rigel Corporation. READS51 and the source code for all the experiments are available from the web site www.rigelcorp.com (see Appendix A). The experiments are written in READS51 and downloaded to the boards to run. READS51 contains an editor, an assembler, a C compiler, and serial communications module for the host PC to communicate with the board. The software runs on an IBM PC and requires the Microsoft Windows 95/98/NT operating system. The reader needs access to a personal computer to conduct the experiments.

Since the experiments focus on interfacing the microcontroller, a good set of laboratory test and measurement instruments is recommended. A basic voltmeter and a logic probe will go a long way in debugging hardware. A logic probe may be built simply be connecting an LED to a 470-ohm resistor. Laboratory studies, especially for demanding applications, would benefit from a logic analyzer and an oscilloscope. The Hewlett Packard HP54645D mixed signal storage oscilloscope was used in the development of the experiments in the book. Such oscilloscopes are particularly suited for microcontroller circuits, since they display both digital and analog information. The digital and analog signals may be stored and analyzed, similar to features found in logic analyzers. They also measure times and voltages.

The chapters progress from specific low-level concepts toward the more modular high-level software structures. Readers with some previous programming skills may simply follow the chapters in succession, implementing the experiments along the way. Readers should download an 8051 microcontroller data book from the web site of one of the manufacturers (see Appendix D). The data book is the definitive source of complete information about the microcontroller. In fact, it is useful to download the data books of several manufacturers, since each data book has unique application notes and sample software. The reader should also download data sheets of the integrated circuits and transistors used in the experiments. Almost all data sheets include sample circuits and operating principles.

Chapters 1 and 2 provide the foundation. They discuss the 8051 architecture, memory spaces, input/output (I/O) ports, instruction set, and programming. There are sample assembly and C programs to illustrate the use of the 8051 instructions and access the I/O ports. The READS51 compiler uses inline assembly to access the low-level microcontroller functions. We find this particularly useful for the learning experience as it provides an awareness and appreciation for machine-level concerns.

Chapter 3 discusses serial I/O. It is an example of using one of the 8051 peripherals, namely the serial port. Again, sample software is given in C and assembly. Modular programming practices are illustrated as high-level functions (such as “printf()”) are written to call the lower-level functions (such as “putc()”).

Chapter 4 deals with the I/O ports and external circuitry to drive high-current loads. This chapter also introduces the concept of “interrupts.” Experiments include running stepper motors and interrupt-driven inputs.

Chapter 5 discusses the 8051 timers. Both timing and counting functions are presented and illustrated with sample code. The power of timers, when combined with interrupts, as system timekeepers is illustrated with sample code. The use of pulse-width modulation as a means to vary the output energy is discussed and illustrated with a motor speed control experiment.

Chapter 6 investigates analog-to-digital (ADC) and digital-to-analog conversion (DAC). Besides the on-chip ADC units, the experiments are given for an external timer and an external serial ADC unit. The DAC is performed with an external serial DAC unit. These experiments also illustrate interfacing to intelligent integrated circuits and subsystems.

Chapter 7 is the capstone experiment, combining all the tools, techniques, and concepts into building a roving robot. The motor drivers of Chapter 4, the interrupt-driven timebase of Chapter 5, and the ADC of Chapter 6 constitute the modules of the final application. A general high-level application software structure is developed using the concept of state spaces.

The reader is assumed to have some programming knowledge. A brief review of the C programming language is given in Appendix B. Readers who are familiar with other programming languages may refer to this appendix. Appendix C explains the decimal, hexadecimal, binary, and binary-coded decimal numbers.

The subject of embedded control and microcontrollers is a very rapidly changing field. Even the mature architectures like the 8051 display a flurry of activity as new members of the family are introduced and new and more powerful development tools are made available. Much information and resources are available on the web. We strongly suggest that the reader closely follow the field by periodically checking the relevant web sites. Appendix D lists manufacturers and information sources as of January 2000.

Sencer Yeralan and Helen Emery
Gainesville, Florida
January 2000

CHAPTER 1
The 8051 ARCHITECTURE

1.1 The 8051 Microcontroller

A microcontroller is a single-chip computer. It is an integrated circuit that combines a processor and the necessary peripherals, such as code and data memory, parallel and serial ports, timers, counters, and interrupt logic. We will discuss and use these peripherals in the experiments that follow. Microcontrollers are almost always used as embedded controllers. Embedded control refers to a computer system that is physically put inside the device it monitors and controls. By their nature, embedded controllers contain dedicated application-specific software. In an industrial or commercial application, code is placed into Read Only Memory (ROM). ROM is nonvolatile, i.e. is retained even if power is removed. On the other hand, Random Access Memory (RAM) refers to volatile memory, which loses its contents if power is interrupted. RAM is useful as a scratchpad to store the run-time variables and intermediate computation values. The 8051 has internal ROM and RAM. Commercial applications may be developed to run with the 8051 in a single-chip mode. However, the 8051 also has the capability to interface with external memory. We will use external memory, more specifically external RAM, to download and run our programs. This approach allows us to quickly modify our code and try it on the hardware.

The 8051 is the oldest microcontroller architecture, developed by Intel in the late 70s. Over the years, the 8051 has attracted a very wide user base. There is much public know-how about the 8051. Moreover, many variants of the 8051 have been developed over the years by many manufacturers. This makes the 8051 architecture perhaps the most prolific microcontroller family in history. There are members of the 8051 family with various sizes and types of memory and peripherals, such as more ports, analog-to-digital converters, high-speed synchronous serial channels, network interfaces, high-speed mathematical processing units. The entire family uses the same architecture and the same instruction set. This reduces the demands on software development. Specialized applications may be pursued by picking the member of the 8051 family with just the right set of peripherals.

1.2 The Central Processing Unit

Almost all computer systems contain a Central Processing Unit (CPU), memory, and Input / Output (I/O) devices. The CPU performs the operations, memory holds code and data, and I/O devices allow the system to interact with its environment. The CPU

endlessly loops through a set of steps, known as the instruction cycle. The cycle starts with the processor reading instructions from code memory. The address from which the instruction is read is stored in a special register, called the Program Counter (PC). The PC is automatically updated by the processor as the instruction bytes are fetched. The fetched instructions are interpreted and executed. The cycle then repeats. Instructions may take one or more code bytes, and take one or more cycles to execute. The simplest instructions take a single byte of code and execute in one machine cycle. The standard 8051 machine cycle is equal to 12 oscillator cycles. So with a 12 MHz oscillator, the 8051 may perform 1 Million Instructions Per Second (MIPS).

The status of the 8051 CPU is reflected by the Program Status Word (PSW). The PSW is a register, i.e., a memory cell within the 8051. The PSW contains bit-wide status information about the CPU. For example, an external carry generated by an addition is placed in one of the PSW flags.

Bit	7	6	5	4	3	2	1	0
Flag	CY	AC	F0	RS1	RS0	OV	F1	P
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflow Flag	User Flag 1	Parity Flag

Figure 1.1. The Fields of the Program Status Word (PSW).

The PSW contains two user flags F0 and F1. Lastly, the PSW contains a two-bit control field that selects the active register bank.

1.3 The Programmer's View of the 8051 Memory Spaces

The on-chip peripherals distinguish the microcontroller from general-purpose microprocessors. These peripherals are accessed by their associated registers, called the Special Function Registers (SFRs). In addition, the 8051 has internal RAM to store variables and data, and optionally, internal code memory to store the application program. If the internal memory is sufficient for the application, the 8051 may be operated in the single-chip mode. Applications that have large memory requirements are accommodated by the 8051's external memory interface. Starting with the 8052, the enhanced versions of the 8051 contain more internal data RAM. The additional RAM, referred to as indirect data RAM, is also treated differently.

1.3.1 The Harvard Architecture

Some processors, especially most of the general-purpose processors use a so-called "linear" memory model. For example, the "Von Neumann" architecture uses a

linear memory space. Code memory and data memory occupies different ranges in this space. On the other hand, the "Harvard" architecture distinguishes between code and data memory. In this arrangement, knowing its address is not sufficient to uniquely specify the memory cell. In addition, we need to specify which memory space (code or memory) we are dealing with. Harvard architectures are more reliable in embedded control. Isolating code memory, which is almost always placed in ROM, from data memory introduces an additional level of reliability.

The 8051 does not use a linear memory model. Rather, the different memory types are considered to exist in different domains, or memory spaces. This is a generalization of the Harvard architecture. The 8051 has the following memory spaces:

1. code memory (internal and external)
2. external data memory
3. internal direct memory (internal data RAM and SFRs)
4. internal indirect memory (additional internal RAM of the 8052 and higher microcontrollers)
5. internal bit memory (overlaps with some of the internal direct memory)

Consider, for example, the address 29h. This could be in code memory, external data memory, internal direct memory, or bit memory. As mentioned, the address does not uniquely specify the memory cell. The memory space is determined by the instruction that accesses the memory. The various addressing modes, discussed in Chapter 2, allow the programmer to access the different memory spaces. Consider, for example, the following two instructions, which use the address 29h.

```
mov  A, 29h    ; A <- direct internal memory
mov  C, 29h    ; C <- internal bit
```

The first instruction moves the contents of the internal data RAM 29h into the accumulator. The second instruction moves a single bit, whose address is 29h, into the carry flag. In this case, the memory space is determined by the first operand (C). The move instruction is said to be context dependent, since the specific action (memory space) is not evident from the “mov” instruction alone. The first operand must be inspected to determine which memory space is involved. In contrast, the move instruction for external data memory uses a different instruction mnemonic.

```
movx A, (source operand) ; A <- external data source operand
```

In this case, when the assembler parses the “movx” mnemonic, it is already known that the source resides in external data memory (and hence the ‘x’ suffix). There is a

similar instruction, namely “movc” that specifies that the source is in the code memory space.

Note that code memory is not partitioned into internal and external spaces. Internal code memory, if present, starts from address 0. Internal code memory may also be disabled by connecting the EA# input pin to ground. If the code memory address does not fall in the range of internal code memory, external code memory is assumed.

The internal direct memory contains 128 bytes of data RAM, occupying addresses 0 to 127. Internal direct memory also contains 128 bytes of SFRs, in the address range 128 to 255. The additional 128 bytes of internal data RAM also occupy addresses 128 to 255. The two memory spaces, SFRs and additional internal data RAM, are distinguished by the way they are accessed. If the address is directly given in the instruction, as in,

```
mov  A, 90h   ; direct internal memory in the SFR range
```

then the memory space is internal direct SFR memory. If the instruction uses the register indirect addressing mode, as in,

```
mov  R0, #90h ; constant 90h placed in register R0
mov  A, @R0   ; indirect internal data RAM
```

then the source byte is in the internal data RAM space.

1.3.2 Register Space

The 8051 architecture and instruction set supports general-purpose registers to enhance the data processing capabilities. Registers are addressed in instructions by their name, rather than by an address. Most architectures use at least two registers to carry out operations. The two operands are placed in the registers and the operation, say addition, is performed. The result usually remains in one of the registers. In some cases, such as in multiplication, the result occupies both registers. These two registers are often called the primary and secondary registers. Typically, they are designated by ‘A’ (the accumulator) and ‘B’ (the B register). The 8051 supports this convention. There are instructions that refer to the A and B registers by name. For example,

```
clr  A
```

clears the accumulator. The operand A is really part of the instruction rather than any direct memory address. For example,

```
clr  45h
```

does not clear internal data RAM location 45h. Rather it clears bit 45h. The MCS-51 instruction set does not include an instruction to clear the contents of an internal data RAM location.

Although, in principle, two registers are sufficient to carry out the operations, they become the bottleneck in common applications. Consider, for example, a loop which adds the values in a list of known size. We may place the size of the list in a counter variable. We need to perform an addition at each iteration, and then decrement the loop counter. If we only had the A and B registers to carry out the operations, the partial sum needs to be saved at each iteration. Then the loop counter is copied into A, decremented, and then compared to zero to see if all values in the list are added. Otherwise, the loop counter must be moved from A back to its memory address, and the partial sum copied back to A. The next value in the list is then copied into the B register to prepare for the next addition. Moving the data into and out of A and B quickly becomes the bottleneck, in the sense that the processor spends more time moving data around than it does carrying out the operations.

General-purpose registers are similar to the A and B registers. They are referred to by their names in the instructions. The 8051 architecture supports 8 registers, named R0, R1, ..., R7. Furthermore, there are four sets of these registers, called the banks. For instance, in the instructions

```
mov  R0, A
```

and

```
dec  R1
```

the registers are specified by their names. Also note that the second instruction, which decrements R1, actually performs an arithmetic operation on a register. That is to say, some operations may be carried out in registers, eliminating the need to first move the data into the accumulator. Clearly, these features relax the bottleneck situation described above.

The banks of 8051 registers are said to overlap internal direct RAM memory. For instance, internal RAM locations 0 to 7 correspond to R0 to R7of register bank 0. That is to say, the same internal direct memory cell, say at location 3, may be specified by its internal direct address or by the register R3. Either way, the same physical cell is involved. Provided that register bank 0 is selected, the two instructions below accomplish the same thing.

```
mov  A, R3
mov  A, 3
```

Nevertheless, as we will see in the next chapter, the two instructions are not identical in their performance. The first takes one byte of code, and the second, two bytes of code, to implement. Referring to registers rather than direct memory usually produces smaller code and faster execution times. Similarly, register banks 1, 2, and 3 overlap with internal direct RAM ranges [8..F], [10..17], and [18..1F], all in hexadecimal.

Table 1.1. Register Banks.

Internal Data RAM								Bank	Register Addressing							
1F	1E	1D	1C	1B	1A	19	18	3	R7	R6	R5	R4	R3	R2	R1	R0
17	16	15	14	13	12	11	10	2	R7	R6	R5	R4	R3	R2	R1	R0
0F	0E	0D	0C	0B	0A	09	08	1	R7	R6	R5	R4	R3	R2	R1	R0
07	06	05	04	03	02	01	00	0	R7	R6	R5	R4	R3	R2	R1	R0

The active register bank is selected by a two-bit field in the Program Status Word (PSW).

1.3.3 Bit-Addressable Memory

The 8051 has several bit-oriented features, which come in handy when interfacing with external inputs and outputs. Furthermore, the control and status registers associated with the peripherals contain many single-bit values or flags. The 8051 has two bit-addressable regions of direct internal memory. Each region contains 128 bits. Sixteen bytes of internal memory, from 20h to 2FH, are bit addressable. This gives the first 128-bits. That is, bit addresses from 0 to 127 are physically the same as the individual bits of the internal data RAM bytes from 20h to 2Fh. The rest of the bits, addresses 128 to 255, are the same as the bits of 16 of the SFRs. Note that a SFR is bit addressable if its (byte) address ends with a 0 or 8. As seen in Table 1.1, the individual bits of bit-addressable internal data and SFR bytes are assigned consecutive bit addresses. For instance, bits 4 and 5 of internal RAM byte 2Bh are given the bit addresses 5Ch and 5Dh. Similarly, the SFR E0h corresponds to the accumulator (ACC). The bits acc.0, acc.1, ..., acc.7 of the accumulator are given bit addresses E0h, E1h, ..., E7h.