Microprocessors & Microcontrollers : Unit-4

Prof. Prajay P. Raikar
Visiting Faculty
Department of Computer Engineering
Goa College of Engineering, Farmagudi-Ponda, Goa
prajayraikar@gmail.com

Table of Contents

Introduction to Intel's 8051 Microcontroller

The 8051 microcontroller, originally developed by Intel, is now produced by various manufacturers. Some notable brands include:

- **Microchip Technology**: They offer a range of 8051-based microcontrollers with advanced features and compatibility for modern applications.
- **NXP Semiconductors**: Known for their robust and versatile 8051 microcontrollers.
- **Silicon Labs**: They provide high-performance 8051 microcontrollers tailored for embedded systems.
- **Atmel (now part of Microchip)**: Offers 8051 microcontrollers with enhanced performance and low power consumption.

The 8051 microcontroller is a versatile and widely used component in electronics and embedded systems. Here are some notable applications:

1. **Consumer Electronics**:
   - Remote controls
   - Washing machines and microwave ovens
   - Home automation systems
2. **Automotive Applications**:
   - Engine control units (ECUs)
   - Anti-lock braking systems (ABS)
   - Vehicle security systems
3. **Industrial Automation**:
   - Motor control
   - Robotics
   - Process control systems
4. **Medical Devices**:
   - Blood pressure monitors
   - Heart rate monitors
   - Portable diagnostic equipment
5. **Communication Systems**:
   - Telephone systems
   - GPS devices
   - Modems and networking equipment
6. **Educational Tools**:
   - Training kits for students learning embedded systems
   - Development boards for prototyping and experimentation
7. **Security and Surveillance**:
   - Biometric devices
   - CCTV systems
   - Access control systems

Its simplicity, affordability, and reliability make it a staple in numerous applications.

Microcontroller vs Microprocessor

What is the difference between a microprocessor and microcontroller? By microprocessor is meant the general-purpose microprocessors such as Intel's x86 family (8086, 80286, 80386, 80486, and the Pentium) or Motorola's 680x0 family (68000, 68010, 68020, 68030, 68040, etc.). These

microprocessors contain no RAM, no ROM, and no I/O ports on the chip itself. For this reason, they are commonly referred to as general-purpose microprocessors.

A system designer using a general-purpose microprocessor such as the Pentium or the 68040 must add RAM, ROM, I/O ports, and timers externally to make them functional. Although the addition of external RAM, ROM, and I/O ports makes these systems bulkier and much more expensive, they have the advantage of versatility such that the designer can decide on the amount of RAM, ROM and I/O ports needed to fit the task at hand. This is not the case with microcontrollers. A microcontroller has a CPU (a microprocessor) in addition to a fixed.

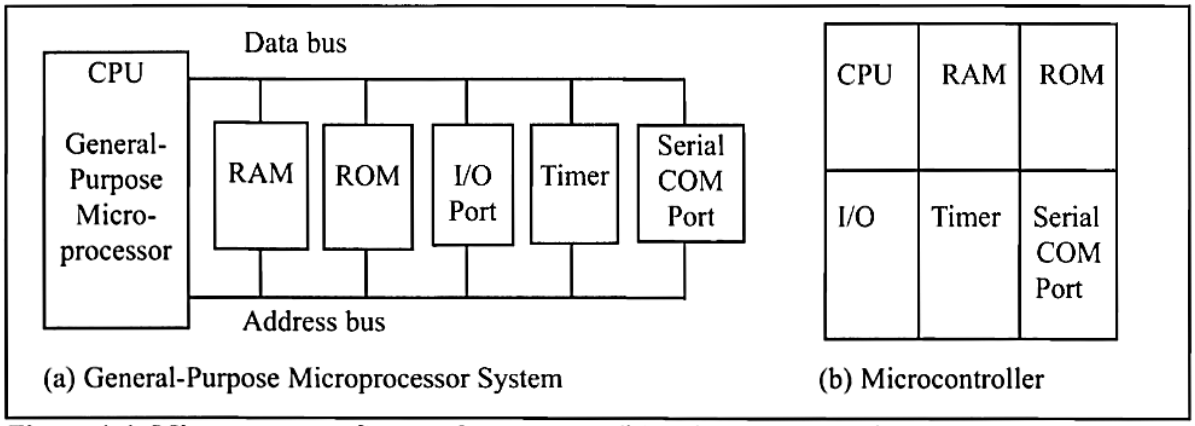


Figure 1-1. Microprocessor System Contrasted With Microcontroller System

Aspect	Microcontroller	Microprocessor
Definition	A compact computer on a single chip, including CPU, memory, and I/O peripherals.	A CPU that requires external components like memory and I/O devices to function.
Purpose	Designed for specific tasks in embedded systems (e.g., appliances, IoT devices).	Used for general-purpose computing (e.g., PCs, servers).
Components	Includes CPU, RAM, ROM, EEPROM, and I/O ports on a single chip.	Contains only the CPU; external components are needed for complete functionality.
Cost	Low-cost and optimized for specific applications.	Higher cost due to external components and greater processing power.
Power Consumption	Low power consumption, suitable for battery-operated devices.	Higher power consumption, requiring robust power sources.
Applications	Embedded systems like washing machines, smart refrigerators, and IoT devices.	Computing systems like desktops, laptops, and gaming consoles.

Microcontrollers are ideal for dedicated tasks, while microprocessors excel in complex, multitasking environments.

Architecture of 8051

FIGURE 2.1a 8051 Block Diagram

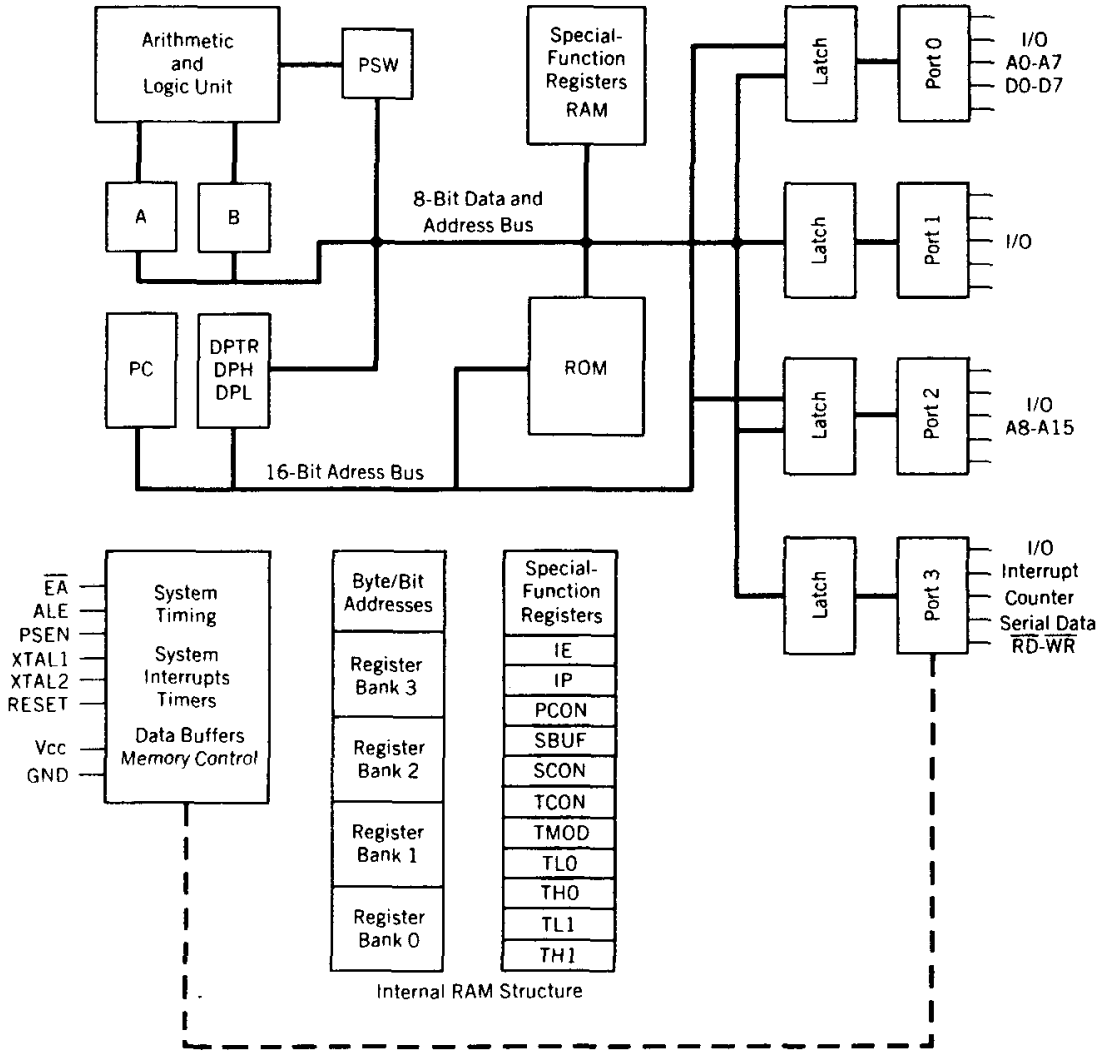
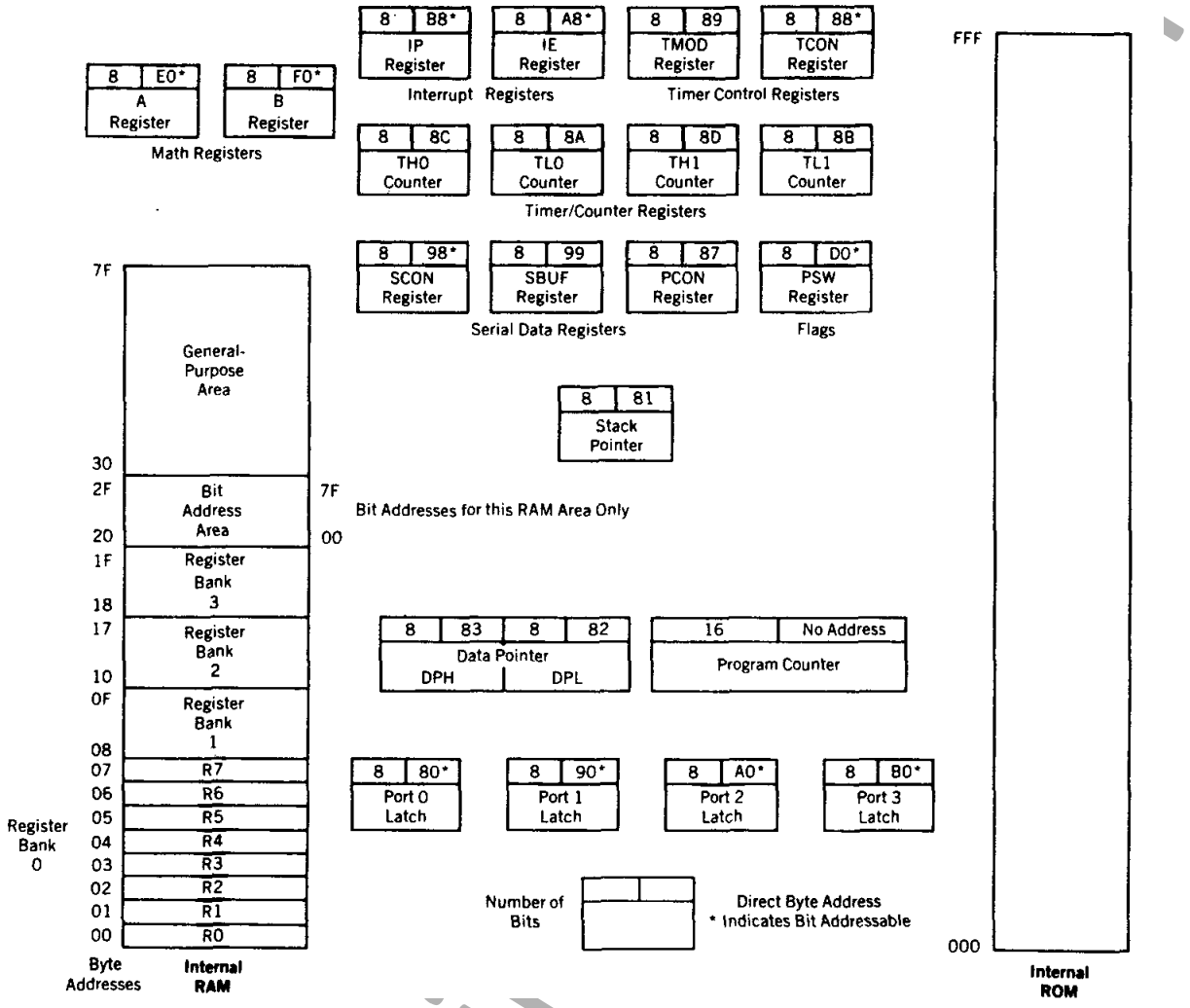


FIGURE 2.1b 8051 Programming Model



Note: For exams, draw only the block diagram and explain it. Programming model is shown only for developing deep-level understanding.

The 8051 architecture consists of these specific features:

- 8-bit CPU with registers A (the accumulator) and B
- 16-bit program counter (PC) and data pointer (DPTR)
- 8-bit program status word (PSW)
- 8-bit stack pointer (SP)
- Internal ROM or EPROM of 4KB
- Internal RAM of 128 bytes:
    - 4 register banks, each containing 8 registers
    - 16 bytes, which may be addressed at the bit level
    - 80 bytes of general-purpose data memory
- 32 input/output pins arranged as four 8-bit ports: P0, P1, P2 & P3
- Two 16-bit timer/counters: T0 and T1
- Full duplex serial data receiver/transmitter: SBUF
- Control registers: TCON, TMOD, SCON, PCON, IP, and IE
- Two external and three internal interrupt sources

- Oscillator and clock circuits

Progrm Counter (PC) and Data Pointer (DPTR)

The 8051 contains two 16-bit registers: the program counter (PC) and the data pointer (DPTR). Each is used to hold the address of a byte in memory.

Program instruction bytes are fetched from locations in memory that are addressed by the **PC**. Program ROM may be on the chip at addresses 0000h to OFFFh (4KB), external to the chip for addresses that exceed OFFFh, or totally external for all addresses from 0000h to FFFFh (64KB). The PC is automatically incremented after every instruction byte is fetched and may also be altered by certain instructions. The PC is the only register that does not have an internal address.

The **DPTR** register is made up of two 8-bit registers, named DPH and DPL, that are used to furnish memory addresses for internal and external code access and external data access**. The DPTR is under the control of program instructions** and can be specified by its 16-bit name, DPTR, or by each individual byte name, DPH and DPL. DPTR does not have a single internal address; **DPH and DPL are each assigned an address.**

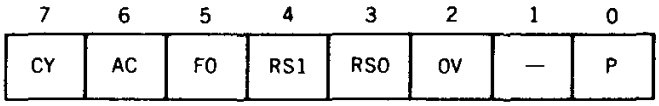
A & B CPU Registers

- The 8051 contains 34 general-purpose, or working, registers. Two of these, registers A and B, comprise the mathematical core of the 8051 central processing unit (CPU).
- The other **32 are arranged as part of internal RAM** in four banks, B0-B3, of eight registers each, named R0 to R7.
- The **A (accumulator)** register is the most versatile of the two CPU registers and is used for many operations, including addition, subtraction, integer multiplication and division, and Boolean bit manipulations.
- The A register is also used for all data transfers between the 8051 and any external memory.
- The B register is used with the A register for multiplication and division operations and has no other function other than as a location where data may be stored.

Flags & Program Status Word (PSW)

- Flags are 1-bit registers provided to store the results of certain program instructions. Other instructions can test the condition of the flags and make decisions based upon the flag states.
- In order that the flags may be conveniently addressed, they are grouped inside the program status word (PSW) and the power control (PCON) registers.
- The 8051 has four math flags that respond automatically to the outcomes of math operations and three general-purpose user flags that can be set to 1 or cleared to 0 by the programmer as desired.
- The math flags include carry (C), auxiliary carry (AC), overflow (OV), and parity (P). User flags are named F0, GF0, and GF1; they are general-purpose flags that may be used by the programmer to record some event in the program.
- Note that all of the flags can be set and cleared by the programmer at will.
- The math flags, however, are also affected by math operations.

FIGURE 2.4 PSW Program Status Word Register



THE PROGRAM STATUS WORD (PSW) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	CY	Carry flag; used in arithmetic, JUMP, ROTATE, and BOOLEAN instructions
6	AC	Auxilliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0

RS1	RS0	
0	0	Select register bank 0
0	1	Select register bank 1
1	0	Select register bank 2
1	1	Select register bank 3

2	OV	Overflow flag; used in arithmetic instructions
1	—	Reserved for future use
0	P	Parity flag; shows parity of register A: 1 = Odd Parity

Bit addressable as PSW.0 to PSW.7

Internal Memory

In the 8051 microcontroller, **RAM** and **ROM** serve distinct purposes:

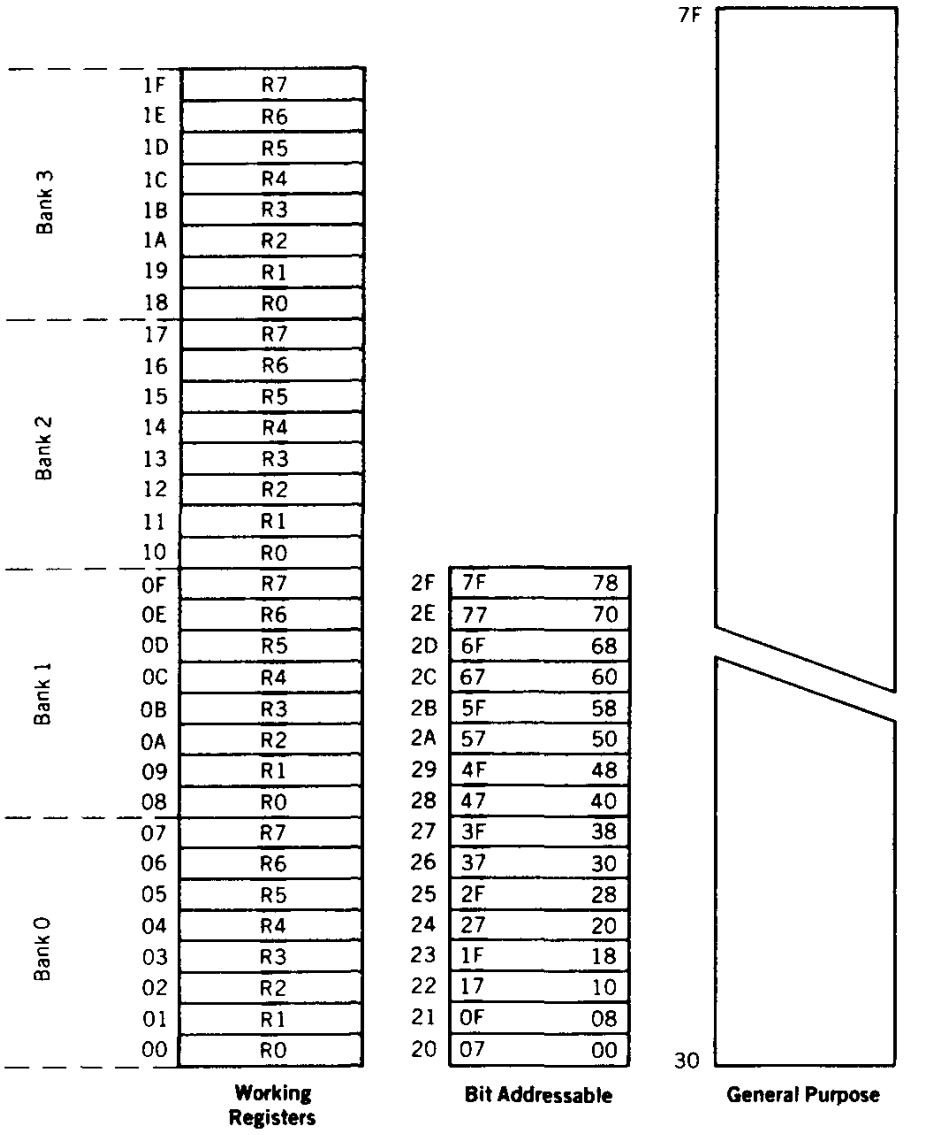
1. **RAM (Random Access Memory)**:
    - RAM is used for **temporary data storage** during program execution. It stores variables, intermediate results, and the stack.
    - External RAM can be interfaced for additional storage if needed.
2. **ROM (Read-Only Memory)**:
    - ROM is used for **permanent storage** of the program code or firmware.
    - The 8051 has an internal ROM of **4KB**, which can be extended up to **64KB** using external program memory.

The RAM structure of the 8051 microcontroller is organized into three main sections:

1. **Register Banks**: The first 32 bytes (addresses 00H to 1FH) are divided into four register banks, each containing eight registers. These banks are used for general-purpose operations.
2. **Bit-Addressable Memory**: The next 16 bytes (addresses 20H to 2FH) are bit-addressable, meaning individual bits can be accessed directly. This is useful for applications requiring precise control.
3. **General-Purpose RAM**: The remaining 80 bytes (addresses 30H to 7FH) are general-purpose RAM, which can be used for **temporary data storage or as a stack**. This RAM area is also known as scratchpad RAM.

Additionally, the 8051 microcontroller has **Special Function Registers (SFRs)** located in the upper memory area (addresses 80H to FFH), which are used for controlling various hardware features.

FIGURE 2.5 Internal RAM Organization



Note: Byte addresses are shown to the left; bit addresses registers are shown inside a location.

Note: RAM Organization is shown only for developing deep-level understanding. Not to be studied for exam.

Special Function Registers (SFRs)

The 8051 operations that do not use the internal 128-byte RAM addresses from 00h to 7Fh are done by a group of specific internal registers, each called a special-function register (SFR), which may be addressed much like internal RAM, using addresses from 80h to FFh.

Some SFRs (marked with an asterisk (*) in Figure 2. 1b are also bit addressable, as is the case for the bit area of RAM. This feature allows the programmer to change only what needs to be altered, leaving the remaining bits in that SFR unchanged.

NAME	FUNCTION	INTERNAL RAM ADDRESS (HEX)
A	Accumulator	0E0
B	Arithmetic	0F0
DPH	Addressing external memory	83
DPL	Addressing external memory	82
IE	Interrupt enable control	0A8
IP	Interrupt priority	0B8
P0	Input/output port latch	80
P1	Input/output port latch	90
P2	Input/output port latch	A0
P3	Input/output port latch	0B0
PCON	Power control	87
PSW	Program status word	0D0
SCON	Serial port control	98
SBUF	Serial port data buffer	99
SP	Stack pointer	81
TMOD	Timer/counter mode control	89
TCON	Timer/counter control	88
TL0	Timer 0 low byte	8A
TH0	Timer 0 high byte	8C
TL1	Timer 1 low byte	8B
TH1	Timer 1 high byte	8D

- Note that the PC is not part of the SFR and has no internal RAM address.
- SFRs are named in certain opcodes by their functional names, such as A or TH0, and are referenced by other opcodes by their addresses, such as 0E0h or 8Ch.
- Note that any address used in the program must start with a number; thus, address E0h for the A SFR begins with 0.
- Failure to use this number convention will result in an assembler error when the program is assembled.

I/O Pins, Ports & Circuits

- Each port has a D-type output latch for each pin.
- The SFR for each port is made up of these eight latches.
- The port latches should not be confused with the port pins; the data on the latches does not have to be the same as that on the pins.
- The two data paths are shown in Figure 2.7 by the circuits that read the latch or pin data using two entirely separate buffers.
- The top buffer is enabled when latch data is read, and the lower buffer, when the pin state is read.
- The status of each latch may be read from a latch buffer, while an input buffer is connected directly to each pin so that the pin status may be read independently of the latch state.

Port 0

- Port 0 pins may serve as inputs, outputs, or, when used together, as a bi-directional low- order address and data bus for external memory.
- For example, when a pin is to be used as an **input**, a 1 must be written to the corresponding port 0 latch by the program, thus turning both of the output transistors off, which in turn

causes the pin to "float" in a high-impedance state, and the pin is essentially connected to the input buffer.

- When used as an **output**, the pin latches that are programmed to a 0 will turn on the lower FET, grounding the pin. All latches that are programmed to a 1 still float.
- "Float" means the pin will not have clear voltage assigned to it and can pick up random voltage values. Also, P0 cannot supply logic high to external device on its own. Thus, **external pullup resistors** will be needed to supply a logic high when using port 0 as an output. Upper FET is used only when port is configured as address bus.
- When **port 0 is used as an address bus** to external memory, internal control signals switch the address lines to the gates of the Field Effect Transistors (FETs). A logic 1 on an address bit will turn the upper FET on and the lower FET off to provide a logic high at the pin. When the address bit is a zero, the lower FET is on and the upper FET off to provide a logic low at the pin. After the address has been formed and latched into external circuits by the Address Latch Enable (ALE) pulse, the bus is turned around to become a data bus.
- Port 0 now reads data from the external memory and must be configured as an input, so a logic 1 is automatically written by internal control logic to all port 0 latches.

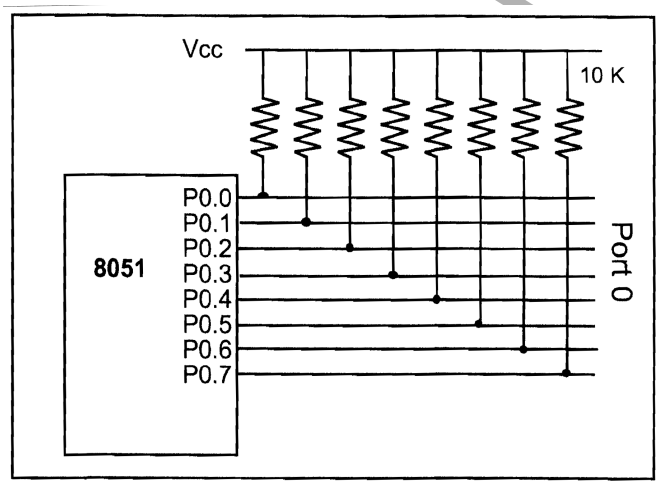


Figure 4-2. Port 0 with Pull-Up Resistors

Port 1

- Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labelled "Internal FET Pullup" as an active pullup load.
- Used as an **input**, a 1 is written to the latch, turning the lower FET off; the pin and the input to the pin buffer are pulled high by the FET load (pull-up resistor). An external circuit drive the pin low to input a 0 or leave the input high for a 1.
- If used as an **output**, the latches containing a 1 can drive the input of an external circuit high through the pullup.
- If a 0 is written to the latch, the lower FET is on, the pullup is off, and the pin can drive the input of the external circuit low.

```
MOV    A,#0FFH  ;A=FF hex
MOV    P1,A     ;make P1 an input port
                ;by writing all 1s to it
```

Port 2

- Port 2 may be used as an input/output port similar in operation to port 1.

- The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.
- Port 2 pins are momentarily changed by the address control signals when supplying the high byte of a 16-bit address.
- Port 2 latches remain stable when external memory is addressed, as they do not have to be turned around (set to 1) for data input as is the case for port 0.

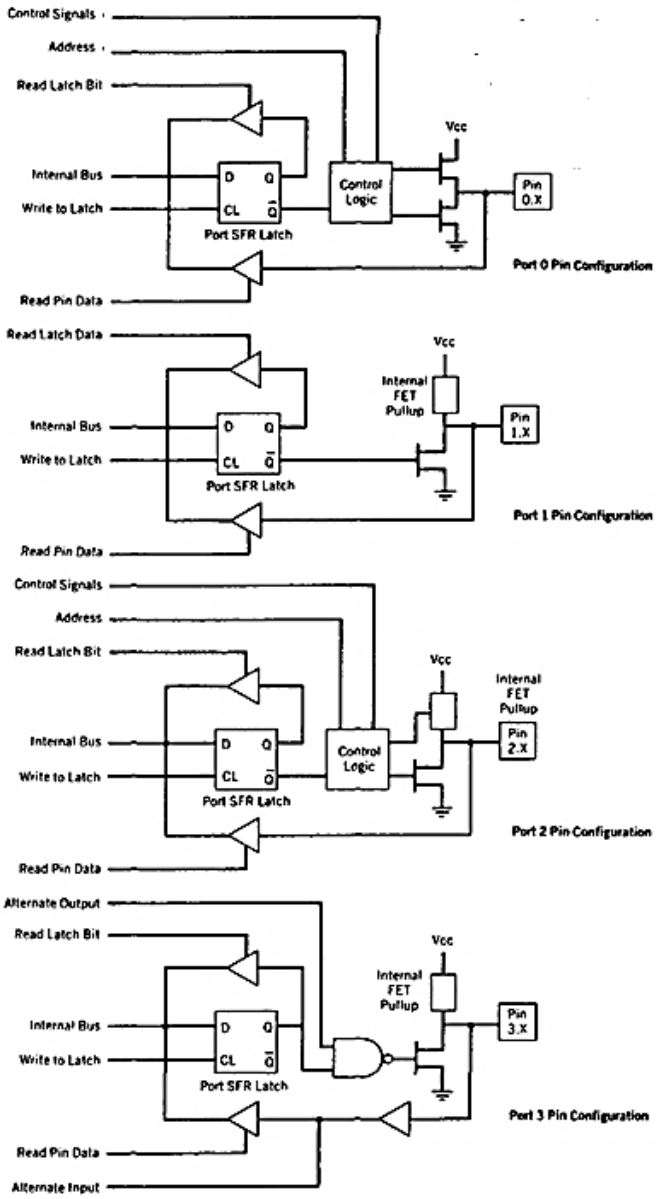
Port 3

- Port 3 is an input/output port like port 1.
- The input and output functions can be programmed under the control of the P3 latches or under the control of various other special function registers. The port 3 alternate uses are shown in the following table:

PIN	ALTERNATE USE	SFR
P3.0–RXD	Serial data input	SBUF
P3.1–TXD	Serial data output	SBUF
P3.2–$\overline{\text{INT0}}$	External interrupt 0	TCON.1
P3.3–$\overline{\text{INT1}}$	External interrupt 1	TCON.3
P3.4–T0	External timer 0 input	TMOD
P3.5–T1	External timer 1 input	TMOD
P3.6–$\overline{\text{WR}}$	External memory write pulse	—
P3.7–$\overline{\text{RD}}$	External memory read pulse	—

- Unlike ports 0 and 2, which can have external addressing functions and change all eight port bits when in alternate use, each pin of port 3 may be individually programmed to be used either as I/O or as one of the alternate functions.

FIGURE 2.7 Port Pin Circuits



Addressing Modes

Immediate Addressing Mode

In this addressing mode, the source operand is a constant. In immediate addressing mode, as the name implies, when the instruction is assembled, the operand comes immediately after the opcode. Notice that the immediate data must be preceded by the pound sign, "**#**". This addressing mode can be used to load information into any of the registers, including the DPTR register. Examples follow.

```
 MOV A,#25H          ;load 25H into A
 MOV R4,#62          ;load the decimal value 62 into R4
 MOV B,#40H          ;load 40H into B
 MOV DPTR,#4521H     ;DPTR=4512H
"MOV P1,#55H"
```

Register Addressing Mode

Register addressing mode involves the use of registers to hold the data to be manipulated. Examples of register addressing mode follow.

```
MOV A,R0  ;copy the contents of R0 into A
MOV R2,A  ;copy the contents of A into R2
ADD A,R5  ;add the contents of R5 to contents of A
ADD A,R7  ;add the contents of R7 to contents of A
MOV R6,A  ;save accumulator in R6
```

Source and destination register sizes should match. **Movement of data between Rn (n=0 to 7) are not allowed. *MOV R4, R7* is Invalid!!!**

Direct Addressing Mode

Used to access 128 bytes of RAM in the 8051. That is addresses from 00 to 7FH.

```
MOV R0,40H    ;save content of RAM location 40H in R0
MOV 56H,A     ;save content of A in RAM location 56H
MOV R4,7FH    ;move contents of RAM location 7FH to R4
```

As discussed earlier, RAM locations 0 to 7 are allocated to bank 0 registers R0 - R7. These registers can be accessed in two ways, as shown below.

```
MOV A,4       ;is same as
MOV A,R4      ;which means copy R4 into A
```

The above examples should reinforce the importance of the **"#" sign** in 8051 instructions. See the following code.

```
MOV R2,#5  ;R2 with value 5
MOV A,2    ;copy R2 to A (A=R2=05)
MOV B,2    ;copy R2 to B (B=R2=05)
MOV 7,2    ;copy R2 to R7
           ;since "MOV R7,R2" is invalid
```

Although it is easier to use the names R0-R7 than their memory addresses, RAM locations 30H to 7FH cannot be accessed in any way other than by their addresses since they have no names.

Register Indirect Addressing Mode

In the register indirect addressing mode, a register is used as a pointer to the data. If the data is inside the CPU, only registers **RO and R1 are used** for this purpose. In other words, R2 - R7 cannot be used to hold the address of an operand located in RAM when using this addressing mode. When R0 and R1 are used as pointers, that is, when they hold the addresses of RAM locations, they must be preceded by the **"@" sign**, as shown below.

```
MOV A,@R0  ;move contents of RAM location whose
           ;address is held by R0 into A
MOV @R1,B  ;move contents of B into RAM location
           ;whose address is held by R1
```

However, there are times when we need to access data stored in external RAM or in the code space of on-chip ROM. Whether accessing externally connected RAM or on-chip ROM, we need a 16-bit pointer. In such cases, the DPTR register is used.

Programming 8051 Timers

- The 8051 has two timers: Timer 0 and Timer 1. They can be used either as timers or as event counters.
- Both **Timer 0** and **Timer 1** are **16-bits** wide.
- Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte. Each timer is discussed separately.

Timer 0 Registers

The 16-bit register of Timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (Timer 0 low byte) and the high byte register is referred to as THO (Timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2, etc. For example, the instruction "MOV TL0, #4FH" moves the value 4FH into TL0, the low byte of Timer 0. These registers can also be read like any other register. For example, "MOV R5, TH0" saves TH0 (high byte of Timer 0) in R5.

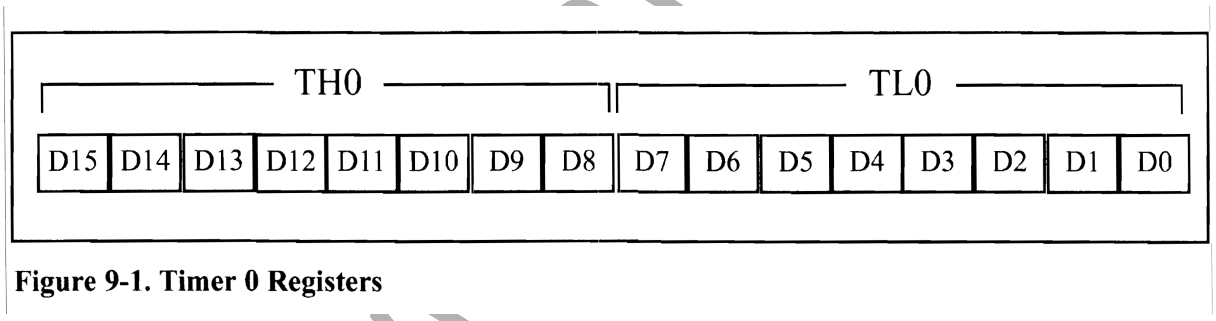


Figure 9-1. Timer 0 Registers

Timer 1 Registers

Timer 1 is also 16 bits, and its 16-bit register is split into two bytes, referred to as TL1 (Timer 1 low byte) and TH1 (Timer 1 high byte). These registers are accessible in the same way as the registers of Timer 0.

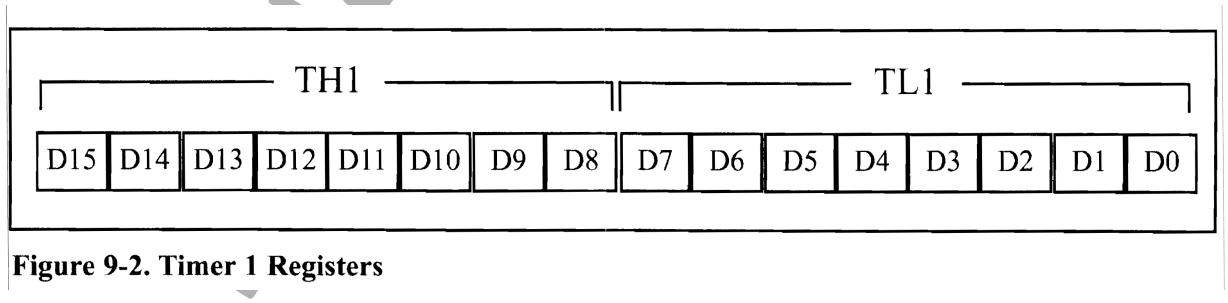


Figure 9-2. Timer 1 Registers

TCON (Timer Control Register)

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.
IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

Figure 11-6. TCON (Timer/Counter) Register (Bit-addressable)

TMOD (Timer Mode) Register

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE Gating control when set. The timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0

M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	16-bit timer mode 16-bit timer/counters THx and TLx are cascaded; there is no prescaler
1	0	2	8-bit auto reload 8-bit auto reload timer/counter; THx holds a value that is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

Figure 9-3. TMOD Register

Note: Only Mode 1 & Mode 2 are widely used.

Example 9-1

Indicate which mode and which timer are selected for each of the following.
(a) `MOV TMOD,#01H` (b) `MOV TMOD,#20H` (c) `MOV TMOD,#12H`

Solution:

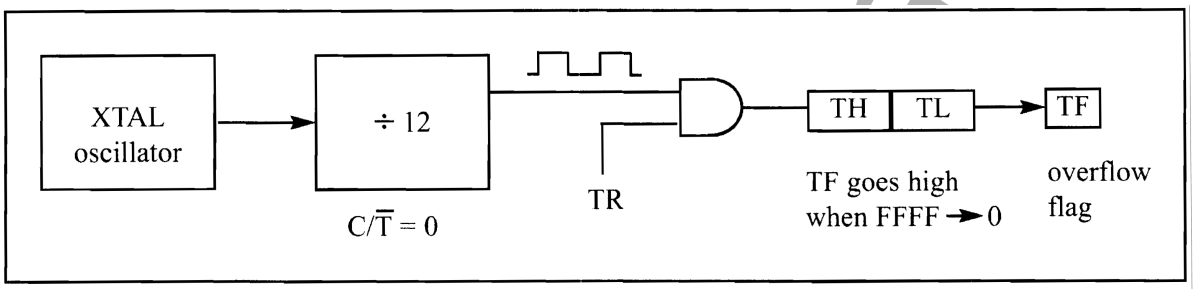
We convert the values from hex to binary. From Figure 9-3 we have:

(a) `TMOD = 00000001, mode 1 of Timer 0 is selected.`
(b) `TMOD = 00100000, mode 2 of Timer 1 is selected.`
(c) `TMOD = 00010010, mode 2 of Timer 0, and mode 1 of Timer 1 are selected.`

Timer Mode 1 Programming

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "SETB TRO" for Timer 0 and "SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH.
4. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored.
5. When this timer flag is raised, one option would be to stop the timer with the instructions
6. "CLR TRO" or "CLR TR1", for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TF0 for Timer 0, and TF1 for Timer 1.
7. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to 0.

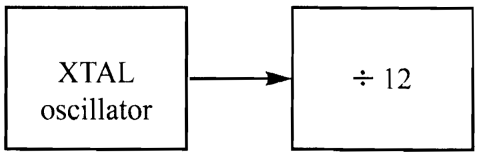


Example 9-2

Find the timer's clock frequency and its period for various 8051-based systems, with the following crystal frequencies.

(a) 12 MHz
(b) 16 MHz
(c) 11.0592 MHz

Solution:



(a) 1/12 × 12 MHz = 1 MHz and T = 1/1 MHz = 1 µs

(b) 1/12 × 16 MHz = 1.333 MHz and T = 1/1.333 MHz = .75 µs

(c) 1/12 × 11.0592 MHz = 921.6 kHz;
T = 1/921.6 kHz = 1.085 µs

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

Example 9-4

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program.

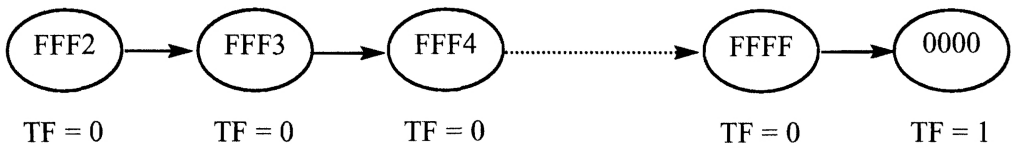
```
            MOV    TMOD,#01          ;Timer 0, mode 1(16-bit mode)
HERE:       MOV    TL0,#0F2H         ;TL0 = F2H, the Low byte
            MOV    TH0,#0FFH         ;TH0 = FFH, the High byte
            CPL    P1.5              ;toggle P1.5
            ACALL  DELAY
            SJMP   HERE              ;load TH, TL again
;----------delay using Timer 0
DELAY:
            SETB   TR0               ;start Timer 0
AGAIN:      JNB    TF0,AGAIN         ;monitor Timer 0 flag until
                                     ;it rolls over
            CLR    TR0               ;stop Timer 0
            CLR    TF0               ;clear Timer 0 flag
            RET
```

Solution:

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0 - TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, Timer 0 is started by the "`SETB TR0`" instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0 = 1). At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction "`CLR TR0`". The DELAY subroutine ends, and the process is repeated.

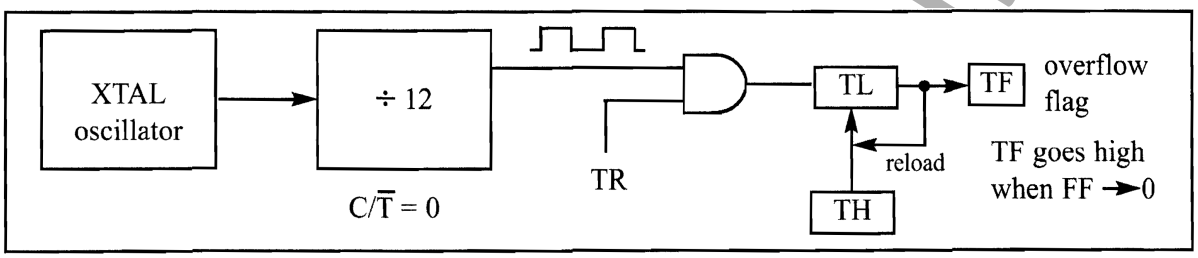
Notice that to repeat the process, we must reload the TL and TH registers and start the timer again.



Timer Mode 2 Programming

The following are the characteristics and operations of mode 2.

1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by the instruction "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1. This is just like mode 1.
3. After the timer is started, it starts to count up by incrementing the TL register.
4. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag). If we are using Timer 0, TFO goes high; if we are using Timer 1, TF1 is raised.



To generate a time delay using the timer's mode 2, take the following steps.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used, and select the timer mode (mode 2).
2. Load the TH registers with the initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "JNB TFx, target" instruction to see whether it is raised. Get out of the loop when TF goes high.
5. Clear the TF flag.
6. Go back to Step 4, since mode 2 is auto-reload.
7. Example 9-14 illustrates these points. To achieve a larger delay, we can use multiple registers as shown in Example 9-15.

Timer Mode 0

Setting timer X mode bits to 00b in the TMOD register results in using the THX register as an 8-bit counter and TLX as a 5-bit counter; the pulse input is divided by 32d in TL so that TH counts the original oscillator frequency reduced by a total 384d. As an example, the 6-megahertz oscillator frequency would result in a final frequency to TH of 15625 hertz. The timer flag is set whenever THX goes from FFh to 00h, or in .0164 seconds for a 6-megahertz crystal if THX starts at 00h.

Timer Mode 3

Timers 0 and I may be programmed to be in mode 0, 1, or 2 independently of a similar mode for the other timer. This is not true for mode 3; the timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TFI are then used by timer 0. Timer 0 in mode 3 becomes two completely separate 8-bit counters. TLO is controlled by the gate arrangement of Figure 2.11 and sets timer flag TFO whenever it overflows from FFh to 00h. THO receives the timer clock (the oscillator divided by 12) under the control of TR1 only and sets the TFI flag when it overflows. Timer 1 may still be used in modes 0, 1, and 2, while timer 0 is in mode 3 with one important exception: No interrupts will be generated by timer 1 while timer 0 is using the TF1 overflow flag. Switching timer 1 to mode 3 will stop it (and hold whatever count is in

timer 1). Timer 1 can be used for baud rate generation for the serial port, or any other mode 0, 1, or 2 function that does not depend upon an interrupt (or any other use of the TFI flag) for proper operation.

Serial Port Programming

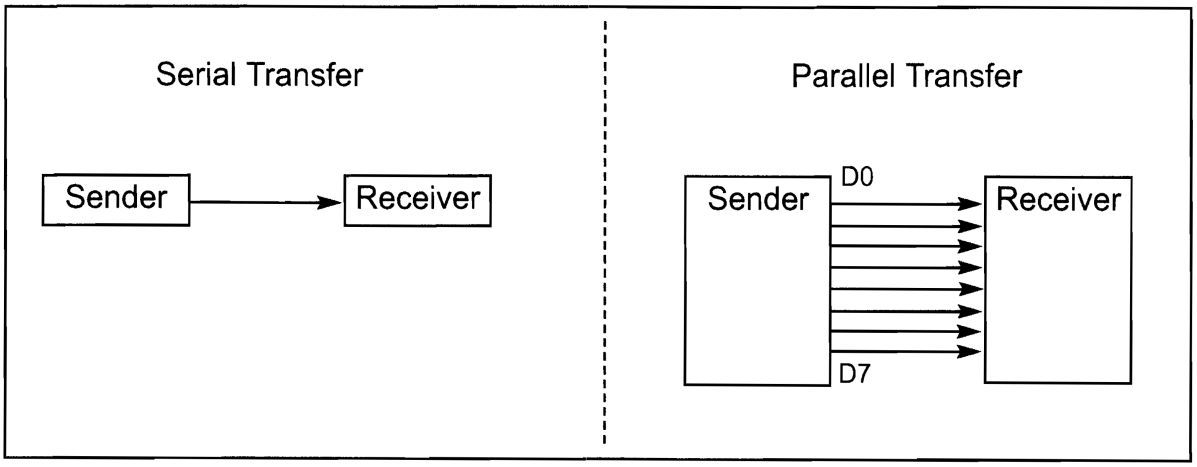


Figure 10-1. Serial versus Parallel Data Transfer

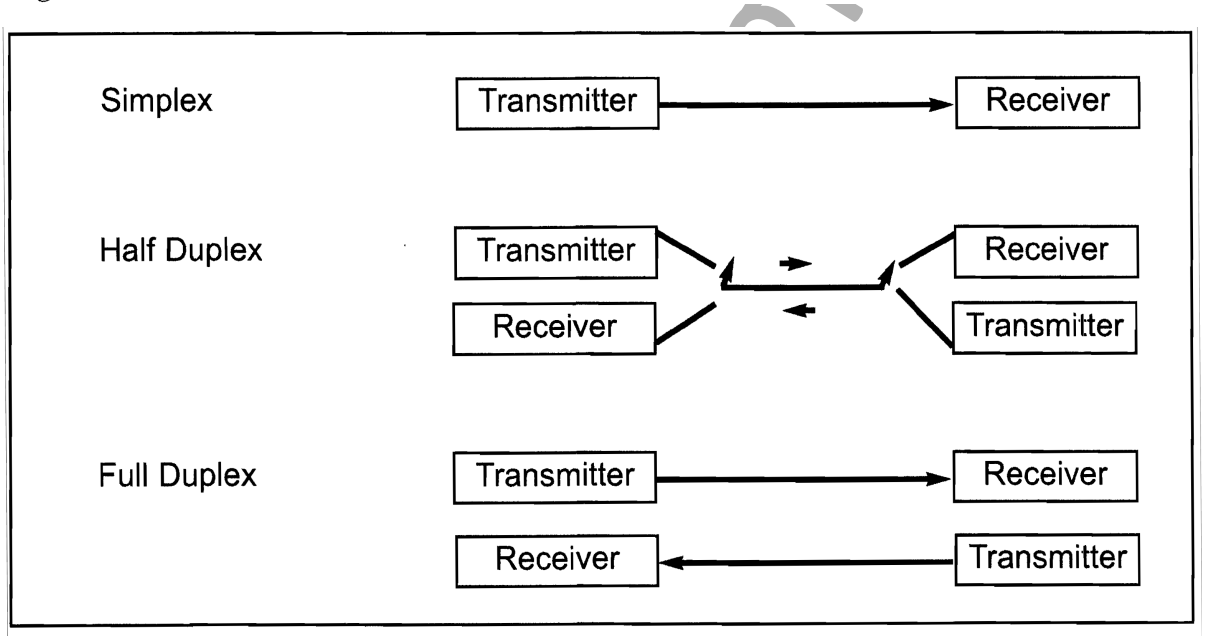


Figure 10-2. Simplex, Half-, and Full-Duplex Transfers

The **baud rate** refers to the number of signal changes or symbols transmitted per second in a communication channel. It's a measure of the speed of data transmission. For example, a baud rate of 9600 means 9600 symbols are transmitted per second.

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. This is done with the help of Timer 1. Before we discuss how to do that, we will look at the relationship between the crystal frequency and the baud rate in the 8051.

As discussed in previous chapters, the 8051 divides the crystal frequency by 12 to get the machine cycle frequency. In the case of XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 kHz (11.0592 MHz / 12 = 921.6 kHz). The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6 kHz by 32 once more before it is used by Timer 1 to set the baud rate.

Therefore, 921.6 kHz divided by 32 gives **28,800 Hz**. This is the number we will use throughout this section to find the Timer 1 value to set the baud rate. **When Timer 1 is used to set the baud rate it must be programmed in mode 2, that is 8-bit, auto-reload.** To get baud rates compatible with the PC, we must load TH1 with the values shown in Table 10-4. Example 10-1 shows how to verify the data in Table 10-4.

Table 10-4: Timer 1 TH1 Register Values for Various Baud Rates

Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	–3	FD
4800	–6	FA
2400	–12	F4
1200	–24	E8

Note: XTAL = 11.0592 MHz.

Example 10-1

With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

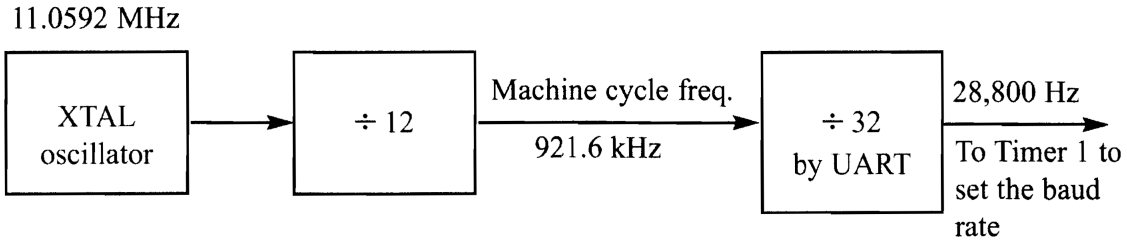
Solution:

With XTAL = 11.0592 MHz, we have:

The machine cycle frequency of the 8051 = 11.0592 MHz / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is the frequency provided by UART to Timer 1 to set baud rate.

(a) 28,800 / 3 = 9600 where –3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where –12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where –24 = E8 (hex) is loaded into TH1

Notice that 1/12th of the crystal frequency divided by 32 is the default value upon activation of the 8051 RESET pin. We can change this default setting. This is explained at the end of this chapter.



SBUF (Serial Buffer) Register

SBUF is an 8-bit register used solely for serial communication in the 8051. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Similarly, SBUF holds the byte of

data when it is received by the 8051's RxD line. SBUF can be accessed like any other register in the 8051. Look at the following examples of how this register is accessed:

```
MOV SBUF,#'D'        ;load SBUF=44H, ASCII for 'D'
MOV SBUF,A           ;copy accumulator into SBUF
MOV A,SBUF           ;copy SBUF into accumulator
```

The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD pin. Similarly, when the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in the SBUF.

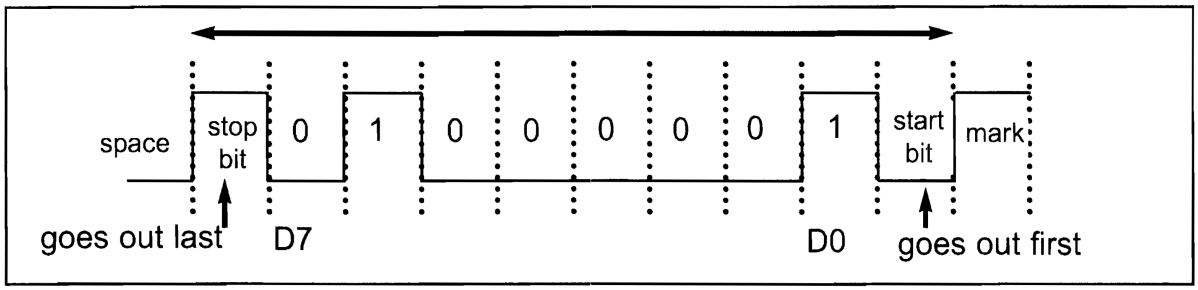


Figure 10-3. Framing ASCII "A" (41H)

SCON (Serial Control) Register

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication. (Make it 0.)
REN	SCON.4	Set/cleared by software to enable/disable reception.
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

Figure 10-9. SCON Serial Port Control Register (Bit-Addressable)

SM0	*SM1*	
0	0	Serial Mode 0
0	1	Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

Note: Only Mode 1 is of interest

Example 10-2

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV   TMOD,#20H   ;Timer 1, mode 2(auto-reload)
        MOV   TH1,#-6     ;4800 baud rate
        MOV   SCON,#50H   ;8-bit, 1 stop, REN enabled
        SETB  TR1         ;start Timer 1
AGAIN:  MOV   SBUF,#"A"   ;letter "A" to be transferred
HERE:   JNB   TI,HERE     ;wait for the last bit
        CLR   TI          ;clear TI for next char
        SJMP  AGAIN       ;keep sending A
```

PCON (Power Control) Register

7	6	5	4	3	2	1	0
SMOD	—	—	—	GF1	GF0	PD	IDL

THE POWER MODE CONTROL (PCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	SMOD	Serial baud rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. Cleared to 0 by program to use timer 1 baud rate.
6-4	—	Not implemented.
3	GF1	General purpose user flag bit 1. Set/cleared by program.
2	GF0	General purpose user flag bit 0. Set/cleared by program.
1	PD	Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
0	IDL	Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors. PCON is not bit addressable.

Interrupt Programming

Interrupts in 8051

There are 6 interrupts in 8051 listed below.

Table 11-1: Interrupt Vector Table for the 8051

Interrupt	ROM Location (Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External hardware interrupt 0 (INT0)	0003	P3.2 (12)	Auto
Timer 0 interrupt (TF0)	000B		Auto
External hardware interrupt 1 (INT1)	0013	P3.3 (13)	Auto
Timer 1 interrupt (TF1)	001B		Auto
Serial COM interrupt (RI and TI)	0023		Programmer clears it.

Enabling & Disabling Interrupts

- Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated.
- 8051 Interrupts are enabled (unmasked) or disabled (masked) by Interrupt Enable (IE) Register.

From Figure 11-2 notice that bit D7 in the IE register is called EA (enable all). This must be set to 1 in order for the rest of the register to take effect.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA IE.7 Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

-- IE.6 Not implemented, reserved for future use.*

ET2 IE.5 Enables or disables Timer 2 overflow or capture interrupt (8052 only).

ES IE.4 Enables or disables the serial port interrupt.

ET1 IE.3 Enables or disables Timer 1 overflow interrupt.

EX1 IE.2 Enables or disables external interrupt 1.

ET0 IE.1 Enables or disables Timer 0 overflow interrupt.

EX0 IE.0 Enables or disables external interrupt 0.

*User software should not write 1s to reserved bits. These bits may be used in future flash microcontrollers to invoke new features.

Figure 11-2. IE (Interrupt Enable) Register

Example 11-1

Show the instructions to (a) enable the serial interrupt, Timer 0 interrupt, and external hardware interrupt 1 (EX1), and (b) disable (mask) the Timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

Solution:

(a)
```
MOV IE,#10010110B   ;enable serial, Timer 0, EX1
```
Since IE is a bit-addressable register, we can use the following instructions to access individual bits of the register.

(b)
```
CLR IE.1        ;mask(disable) Timer 0 interrupt only
```
(c)
```
CLR IE.7        ;disable all interrupts
```
Another way to perform the "`MOV IE,#10010110B`" instruction is by using single-bit instructions as shown below.
```
SETB IE.7       ;EA=1, Global enable
SETB IE.4       ;enable serial interrupt
SETB IE.1       ;enable Timer 0 interrupt
SETB IE.2       ;enable EX1
```

Interrupt Priority Setting

Table 11-3: 8051/52 Interrupt Priority Upon Reset

Highest to Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

Interrupt priority can be altered by using **Interrupt Priority (IP) Register**.

D7 D0

--	--	PT2	PS	PT1	PX1	PT0	PX0

Priority bit = 1 assigns high priority. Priority bit = 0 assigns low priority.

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

User software should never write 1s to unimplemented bits, since they may be used in future products.

Figure 11-8. Interrupt Priority Register (Bit-addressable)

Example 11-12

(a) Program the IP register to assign the highest priority to INT1 (external interrupt 1), then (b) discuss what happens if INT0, INT1, and TF0 are activated at the same time. Assume that the interrupts are both edge-triggered.

Solution:

(a) `MOV IP,#00000100B ;IP.2=1 to assign INT1 higher priority`
The instruction "`SETB IP.2`" also will do the same thing as the above line since IP is bit-addressable.

(b) The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0. This is due to the fact that INT1 has a higher priority than the other two because of the instruction in Step (a). The instruction in Step (a) makes both the INT0 and TF0 bits in the IP register 0. As a result, the sequence in Table 11-3 is followed, which gives a higher priority to INT0 over TF0.

LCD & Keyboard Interfacing

LCD Interfacing

LCD Pin Description:

RS (Register Select): There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS = 0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W (read/write): R/W input allows the user to write information to the LCD or read information from it.

R/W = 1 when reading; R/W = 0 when writing.

E (enable): The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide.

Table 12-1: Pin Descriptions for LCD

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS = 0 to select command register, RS = 1 to select data register
5	R/W	I	R/W = 0 for write, R/W = 1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

D0-D7 (Data Pins): The 8-bit data pins, D0 - D7, are used to send information to the LCD or read the contents of the LCD's internal registers. To display letters and numbers, we send ASCII codes for the letters A - Z, a - z, and numbers 0 - 9 to these pins while making RS = 1.

There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor. Table 12-2 lists the instruction command codes.

We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7 and can be read when R/W = 1 and RS = 0, as follows: if R/W =1,RS =0. When D7= 1 (busy flag = 1), the LCD is busy taking care of internal operations and will not accept any new information. When D7 = 0, the LCD is ready to receive new information. Note: It is recommended to check the busy flag before writing any data to the LCD.

Table 12-2: LCD Command Codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

Note: This table is extracted from Table 12-4.

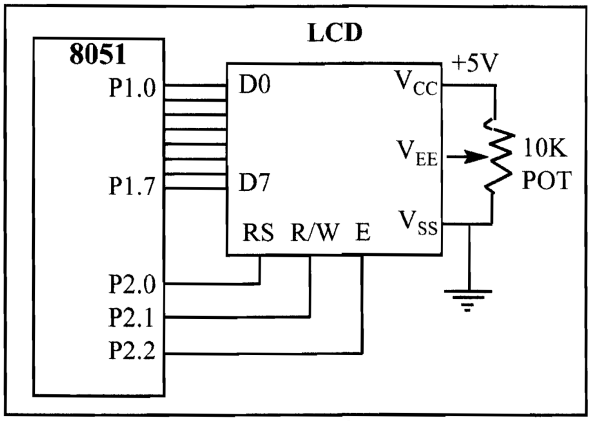
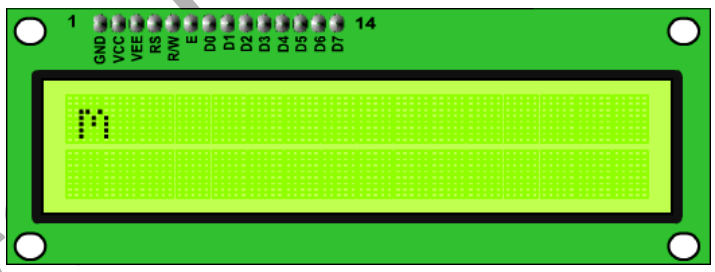


Figure 12-2. LCD Connections



```
;calls a time delay before sending next data/command
; P1.0-P1.7 are connected to LCD data pins D0-D7
; P2.0 is connected to RS pin of LCD
; P2.1 is connected to R/W pin of LCD
; P2.2 is connected to E pin of LCD
        ORG   0H
        MOV   A,#38H          ;init. LCD 2 lines,5x7 matrix
        ACALL COMNWRT         ;call command subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#0EH          ;display on, cursor on
        ACALL COMNWRT         ;call command subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#01           ;clear LCD
        ACALL COMNWRT         ;call command subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#06H          ;shift cursor right
        ACALL COMNWRT         ;call command subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#84H          ;cursor at line 1,pos. 4
        ACALL COMNWRT         ;call command subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#'N'          ;display letter N
        ACALL DATAWRT         ;call display subroutine
        ACALL DELAY           ;give LCD some time
        MOV   A,#'O'          ;display letter O
        ACALL DATAWRT         ;call display subroutine
AGAIN:  SJMP  AGAIN           ;stay here
COMNWRT:                      ;send command to LCD
        MOV   P1,A            ;copy reg A to port1
        CLR   P2.0            ;RS=0 for command
        CLR   P2.1            ;R/W=0 for write
        SETB  P2.2            ;E=1 for high pulse
        ACALL DELAY           ;give LCD some time
        CLR   P2.2            ;E=0 for H-to-L pulse
        RET
DATAWRT:                      ;write data to LCD
        MOV   P1,A            ;copy reg A to port1
        SETB  P2.0            ;RS=1 for data
        CLR   P2.1            ;R/W=0 for write
        SETB  P2.2            ;E=1 for high pulse
        ACALL DELAY           ;give LCD some time
        CLR   P2.2            ;E=0 for H-to-L pulse
        RET
```

```
DELAY:    MOV    R3,#50          ;50 or higher for fast CPUs
HERE2:    MOV    R4,#255         ;R4=255
HERE:     DJNZ   R4,HERE         ;stay until R4 becomes 0
          DJNZ   R3,HERE2
          RET
          END
```

Program 12-1. *(continued from previous page)*

Keyboard Interfacing

Refer to the textbook – 8051 microcontroller and Embedded Systems by Mazidi.

ADC, DAC & Sensor Interfacing

ADC (Analog to Digital Converter) Interfacing

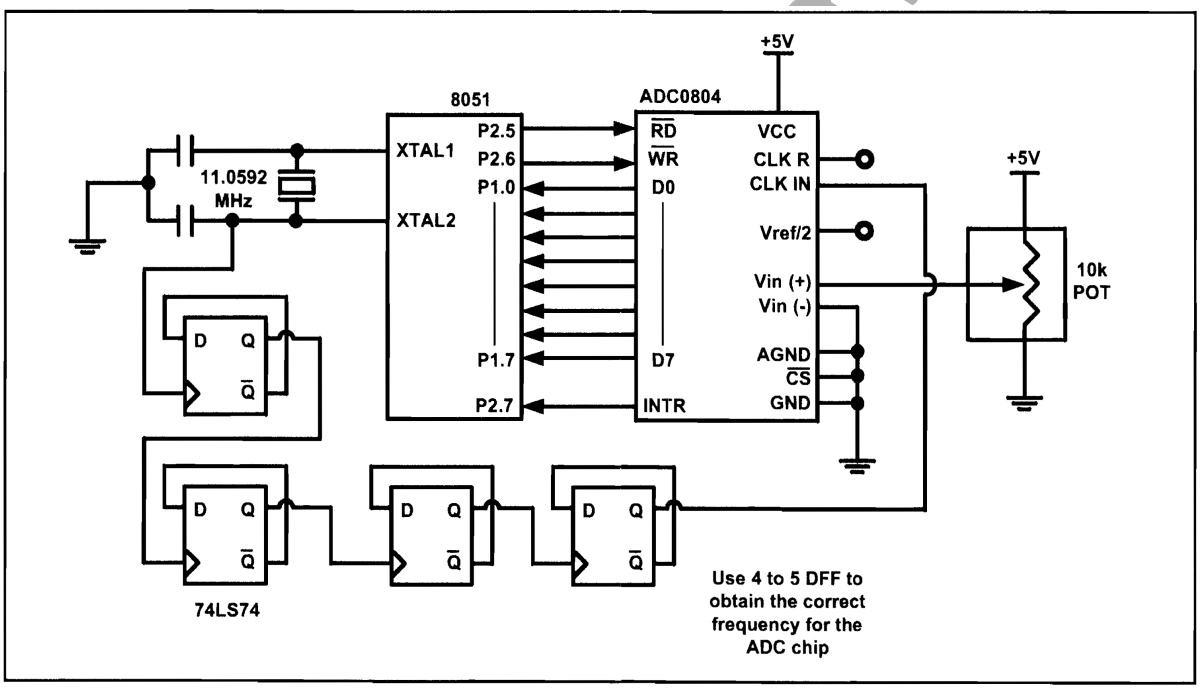


Figure 13-4. 8051 Connection to ADC0804 with Clock from XTAL2 of the 8051

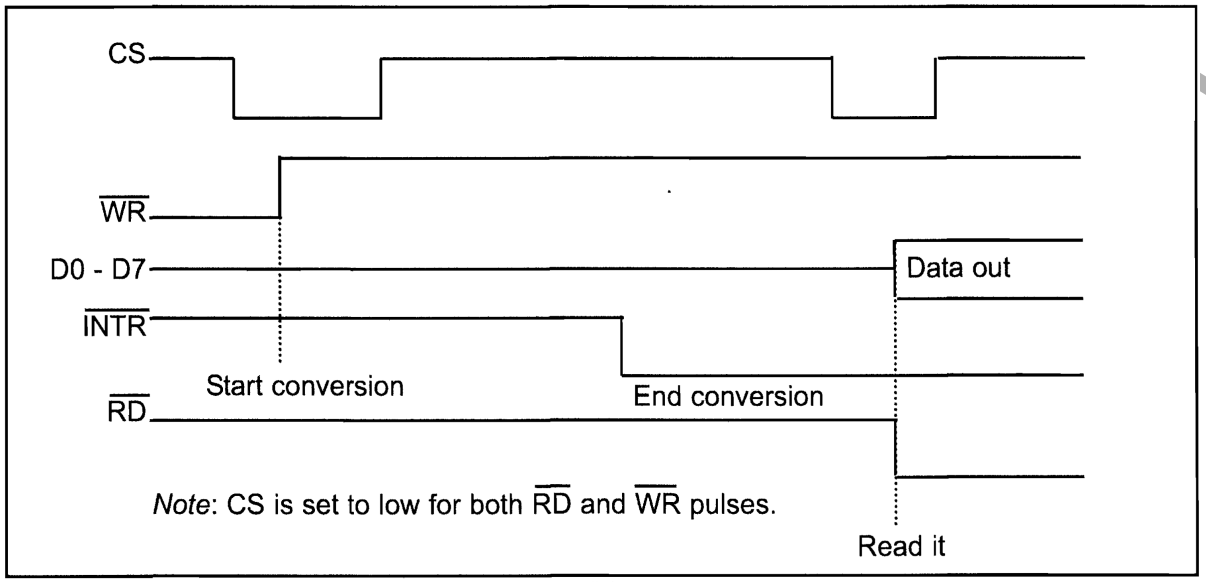


Figure 13-2. Read and Write Timing for ADC0804

```
        RD      BIT  P2.5        ;RD
        WR      BIT  P2.6        ;WR (start conversion)
        INTR    BIT  P2.7        ;end-of-conversion
        MYDATA  EQU  P1          ;P1.0-P1.7=D0-D7 of the ADC804
        MOV     P1,#0FFH         ;make P1 = input
        SETB    INTR
BACK:   CLR     WR               ;WR=0
        SETB    WR               ;WR=1 L-to-H to start conversion
HERE:   JB      INTR,HERE        ;wait for end of conversion
        CLR     RD               ;conversion finished,enable RD
        MOV     A,MYDATA         ;read the data
        ACALL   CONVERSION       ;hex-to-ASCII conversion(Chap 6)
        ACALL   DATA_DISPLAY     ;display the data(Chap 12)
        SETB    RD               ;make RD=1 for next round
        SJMP    BACK
```

DAC (Digital to Analog Converter) Interfacing

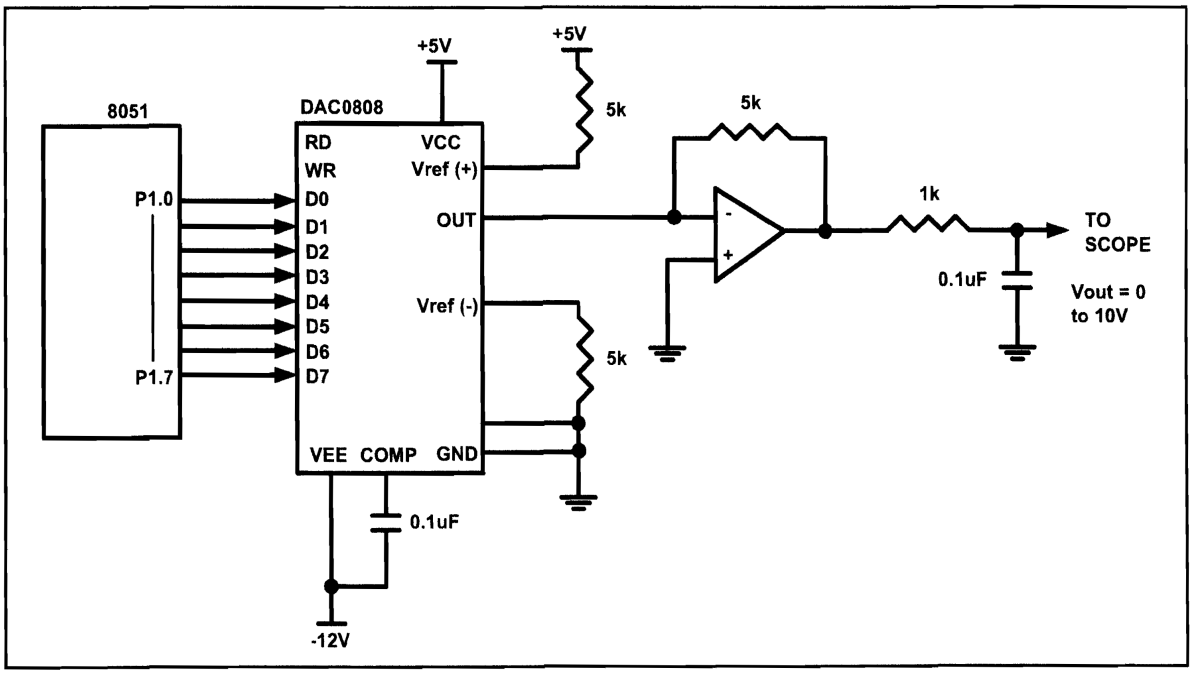


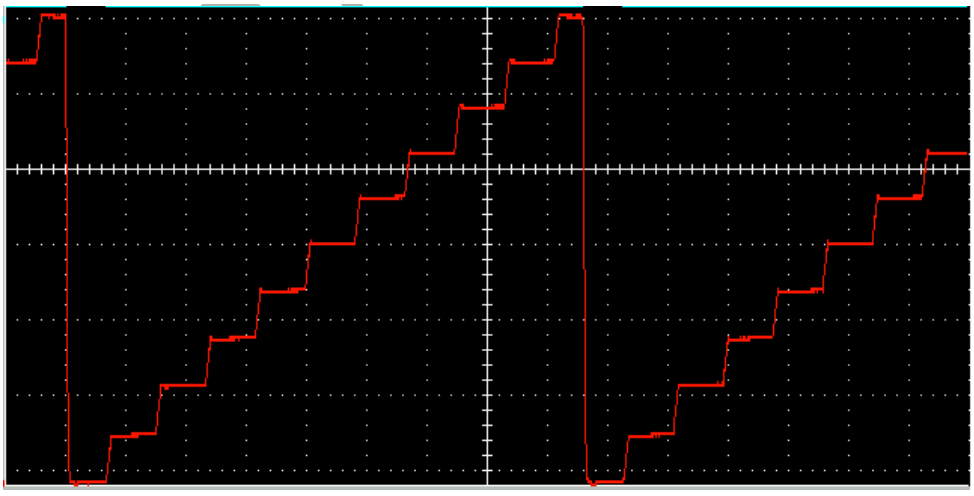
Figure 13-18. 8051 Connection to DAC808

Example 13-4

In order to generate a stair-step ramp, set up the circuit in Figure 13-18 and connect the output to an oscilloscope. Then write a program to send data to the DAC to generate a stair-step ramp.

Solution:

```
        CLR   A
AGAIN:  MOV   P1,A          ;send data to DAC
        INC   A             ;count from 0 to FFH
        ACALL DELAY         ;let DAC recover
        SJMP  AGAIN
```



Temperature Sensor Interfacing

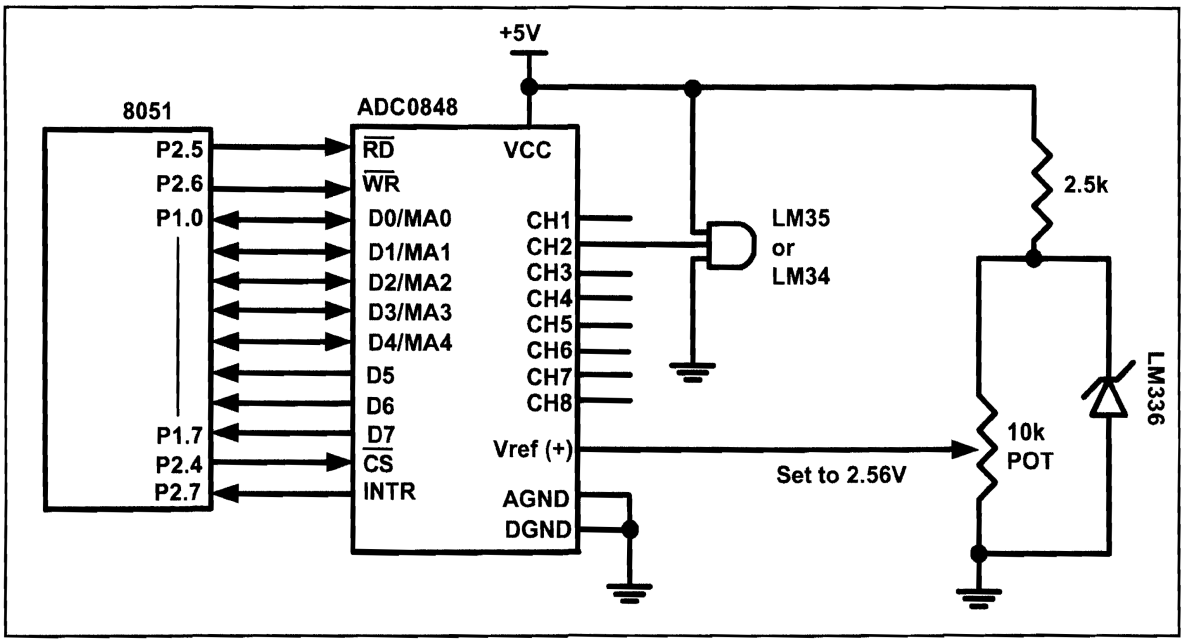


Figure 13-21. 8051 Connection to ADC0848 and Temperature Sensor

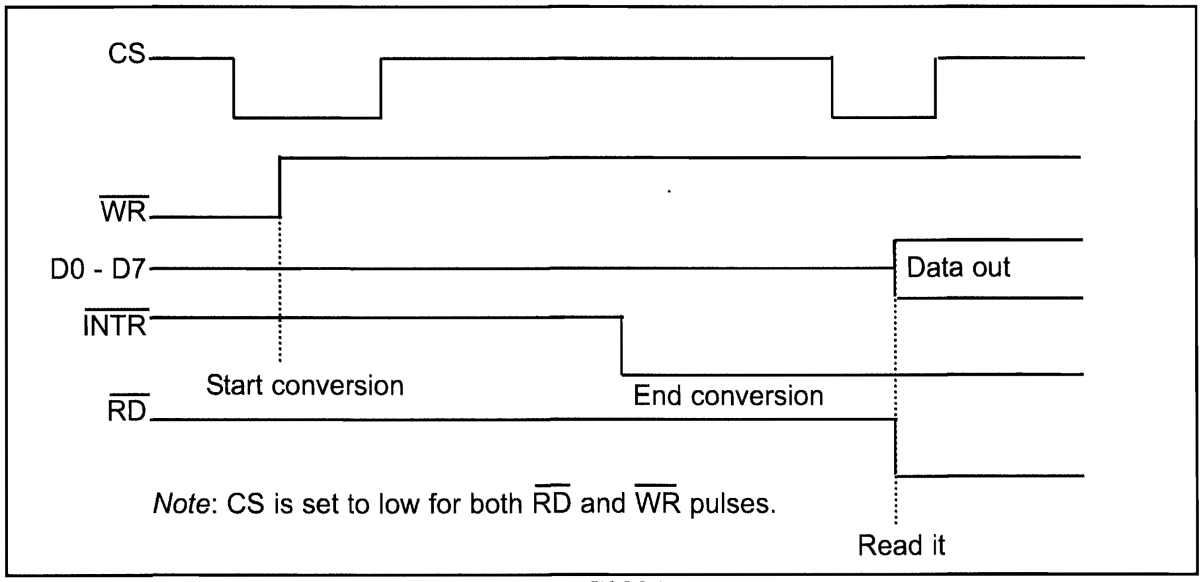


Figure 13-2. Read and Write Timing for ADC0804

```
;Program 13-1
;Assembly code to read temperature, convert it,
;and put it on P0 with some delay
      RD    BIT P2.5          ;RD
      WR    BIT P2.6          ;WR (start conversion)
      INTR  BIT P2.7          ;end-of-conversion
      MYDATA     EQU P1       ;P1.0-P1.7=D0-D7 of the ADC0848
      MOV   P1,#0FFH          ;make P1 = input
      SETB  INTR
BACK: CLR   WR                ;WR=0
      SETB  WR                ;WR=1 L-to-H to start conversion
HERE: JB    INTR,HERE         ;wait for end of conversion
      CLR   RD                ;conversion finished,enable RD
      MOV   A,MYDATA          ;read the data from ADC0848
      ACALL CONVERSION             ;hex-to-ASCII conversion
      ACALL DATA_DISPLAY      ;display the data
      SETB  RD                ;make RD=1 for next round
      SJMP  BACK

CONVERSION:
      MOV   B,#10
      DIV   AB
      MOV   R7,B        ;least significant byte
      MOV   B,#10
      DIV   AB
      MOV   R6,B
      MOV   R5,A        ;most significant byte
      RET

DATA_DISPLAY
      MOV   P0,R7
      ACALL DELAY
      MOV   P0,R6
      ACALL DELAY
      MOV   P0,R5
      ACALL DELAY
      RET
```

External Memory Interface

External Program Memory (ROM) Interfacing

8051 can support **64KB** of external ROM chip. Which allows the program code to be as large as 64KB in size. The **EA (External Access) pin** of the 8051, if connected to Vcc=5V supply, the 8051 accesses program code from internal on-chip ROM. And if EA pin is grounded (0V), 8051 ignores internal on-chip ROM and accesses program code from external ROM. Ports P0 and P2 are used to address the external ROM. P0 provides lower 8-bits (A0-A7) of the address and P2 provides upper 8-bits (A8-A15) of the address. P0 is also used to provide 8-bit (D0-D7) data. Using the P0 to provide address as well as data is termed as **address/data multiplexing**. This saves the number of pins used in microcontroller. Now, to control when P0 will be used for data and address transfer, ALE pin of 8051 is used. **ALE (Address Latch Enable) pin** latches the address on the address bus latch (74LS373) and holds the address bits (A0-A7) stable while P0 is being used for data transfer. When ALE=1, P0 is used for data transfer and when ALE=0, P0 is used for address transfer. The **PSEN (program Store Enable)** pin of 8051 is used to enable

ROM for data output. Hence, connected to the OE (Output Enable) pin of the ROM chip. It an active low signal. CE (Chip Enable) and used to activate entire ROM chip. OE is used to enable output drivers only.

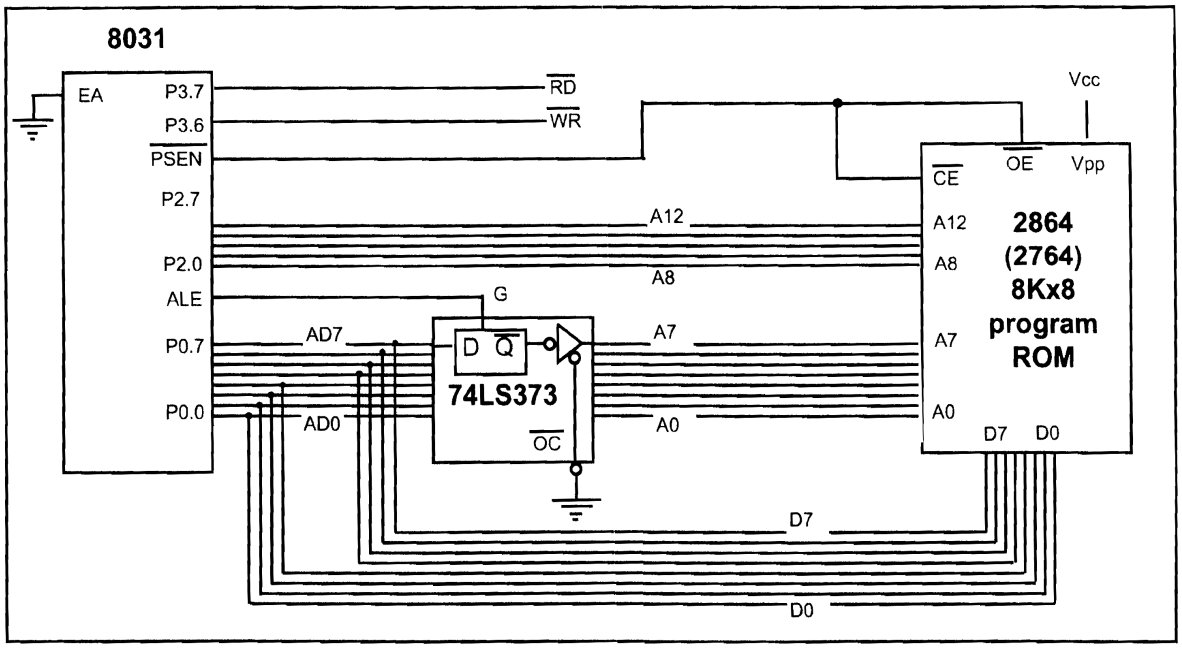


Figure 14-11. 8031 Connection to External Program ROM

External Data Memory (RAM) Interfacing

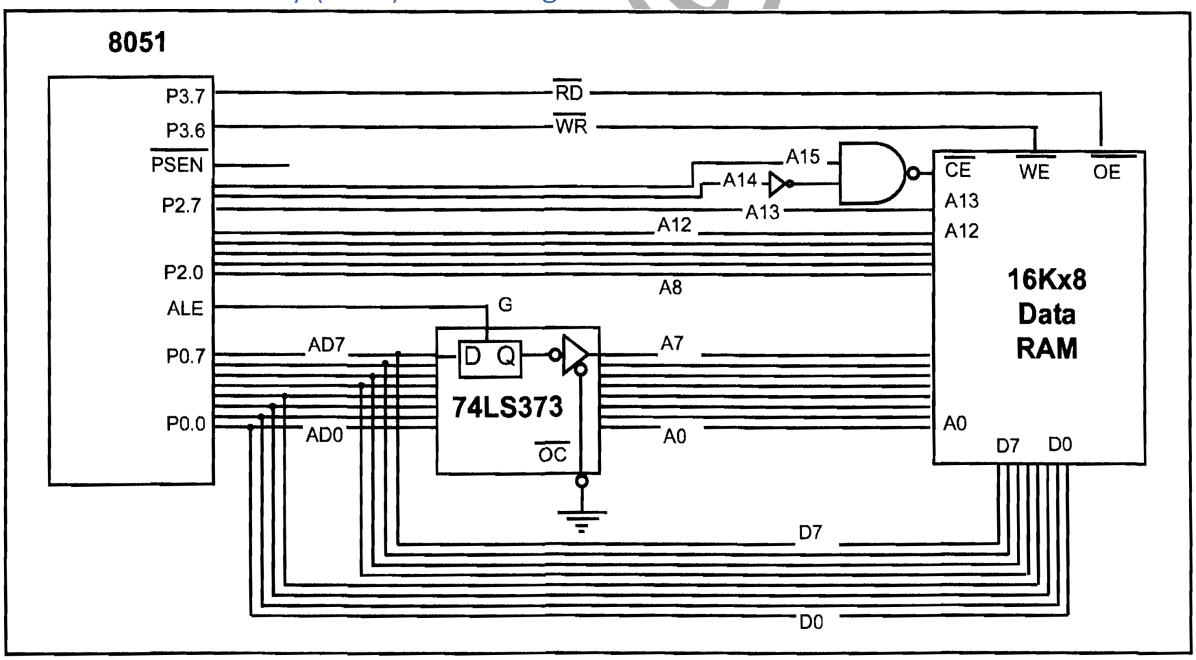


Figure 14-15. 8051 Connection to External Data RAM

Note: Explanation for RAM interfacing remains the same. But PSEN is left unused. And additional RD WR signals are used which control read and write access to the RAM chip.

Stepper Motor and Waveform Generation

Stepper Motor Interfacing

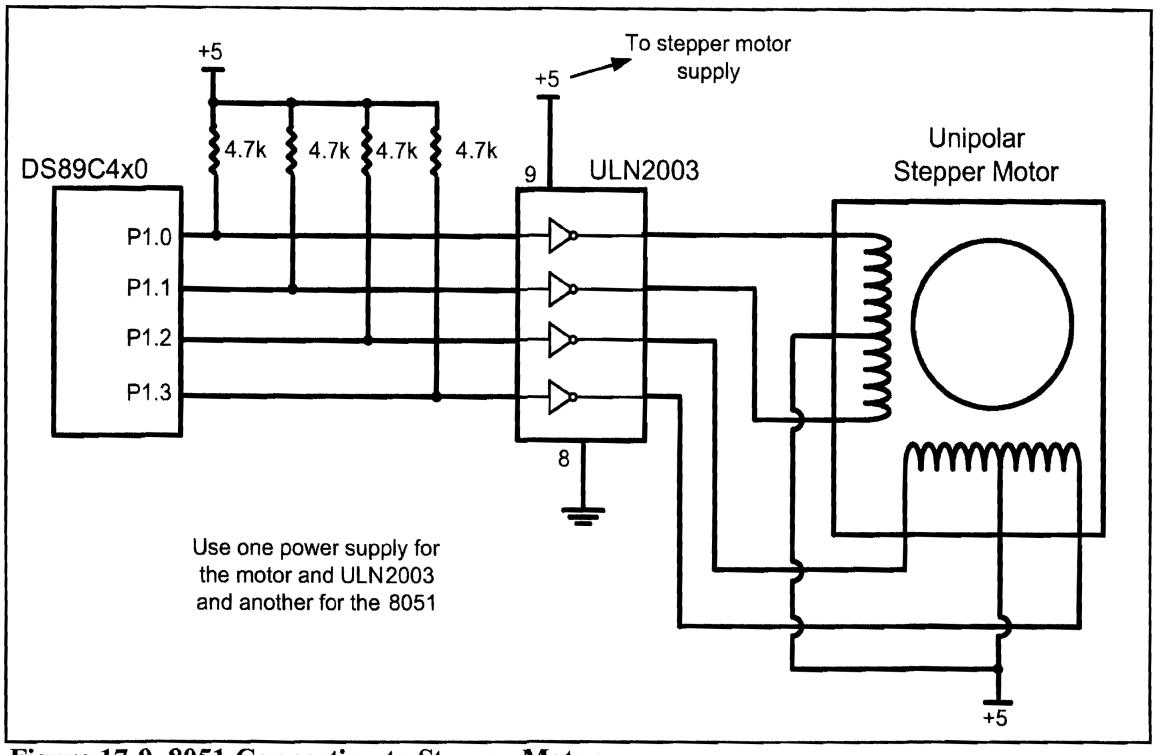


Figure 17-9. 8051 Connection to Stepper Motor

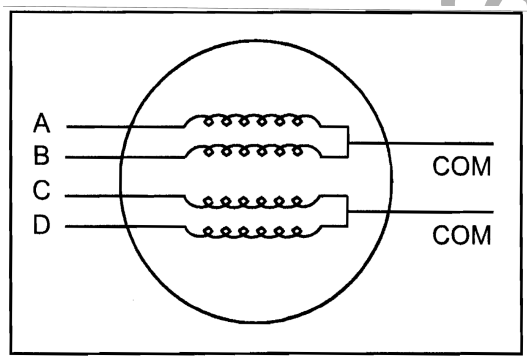


Figure 17-8. Stator Windings Configuration

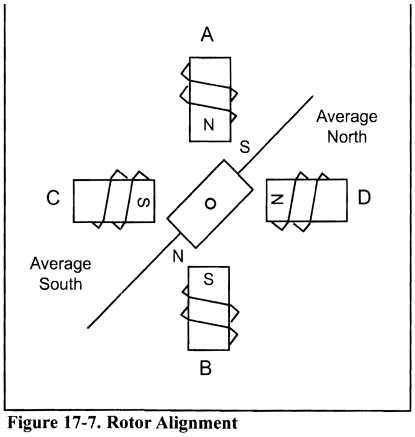


Figure 17-7. Rotor Alignment

Table 17-3: Normal 4-Step Sequence

Clockwise ↓

Step #	Winding A	Winding B	Winding C	Winding D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Counter-clockwise ↑

It must be noted that although we can start with any of the sequences in Table 17-3, once we start, we must continue in the proper order. For example, if we start with step 3 (0110), we must continue in the

sequence of steps 4, 1, 2, etc.

Example 17-1

Describe the 8051 connection to the stepper motor of Figure 17-9 and code a program to rotate it continuously.

Solution:

The following steps show the 8051 connection to the stepper motor and its programming.

1. Use an ohmmeter to measure the resistance of the leads. This should identify which COM leads are connected to which winding leads.
2. The common wire(s) are connected to the positive side of the motor's power supply. In many motors, +5 V is sufficient.
3. The four leads of the stator winding are controlled by four bits of the 8051 port (P1.0 - P1.3). However, since the 8051 lacks sufficient current to drive the stepper motor windings, we must use a driver such as the ULN2003 to energize the stator. Instead of the ULN2003, we could have used transistors as drivers, as shown in Figure 17-9. However, notice that if transistors are used as drivers, we must also use diodes to take care of inductive current generated when the coil is turned off. One reason that using the ULN2003 is preferable to the use of transistors as drivers is that the ULN2003 has an internal diode to take care of back EMF.

```
          MOV  A,#66H     ;load step sequence
BACK:     MOV  P1,A       ;issue sequence to motor
          RR   A          ;rotate right clockwise
          ACALL DELAY     ;wait
          SJMP BACK       ;keep going

          ...
DELAY
          MOV  R2,#100
H1:       MOV  R3,#255
H2:       DJNZ R3,H2
          DJNZ R2,H1
          RET
```

Change the value of DELAY to set the speed of rotation.
We can use the single-bit instructions SETB and CLR instead of RR A to create the sequences.

RR- Rotate Right, ACALL- Absolute Call (Can jump within 2KB range), SJMP- Short Jump