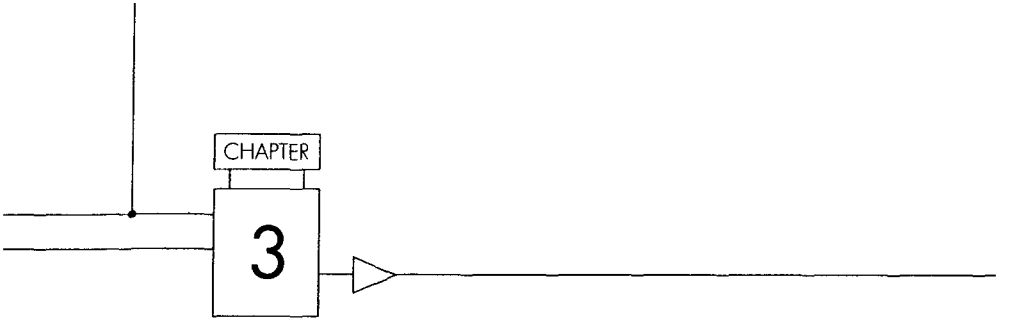


Moving Data

Chapter Outline

Introduction

A computer typically spends more time moving data from one location to another than it spends on any other operation. It is not surprising, therefore, to find that more instructions are provided for moving data than for any other type of operation.

Data is stored at a *source* address and moved (actually, the data is *copied*) to a *destination* address. The ways by which these addresses are specified are called the *addressing modes*. The 8051 mnemonics are written with the *destination* address named *first*, followed by the source address.

A detailed study of the operational codes (opcodes) of the 8051 begins in this chapter. Although there are 28 distinct mnemonics that copy data from a source to a destination, they may be divided into the following three main types:

1. MOV destination, source
2. PUSH source or POP destination
3. XCH destination, source

The following four addressing modes are used to access data:

1. Immediate addressing mode
2. Register addressing mode

3. Direct addressing mode
4. Indirect addressing mode

The MOV opcodes involve data transfers within the 8051 memory. This memory is divided into the following four distinct physical parts:

1. Internal RAM
2. Internal special-function registers
3. External RAM
4. Internal and external ROM

Finally, the following five types of opcodes are used to move data:

1. MOV
2. MOVX
3. MOVC
4. PUSH and POP
5. XCH

Addressing Modes

The way in which the data sources or destination addresses are specified in the mnemonic that moves that data determines the addressing mode. Figure 3.1 diagrams the four addressing modes: immediate, register, direct, and indirect.

Immediate Addressing Mode

The simplest way to get data to a destination is to make the source of the data part of the opcode. The data source is then immediately available as part of the instruction itself.

When the 8051 executes an immediate data move, the program counter is automatically incremented to point to the byte(s) following the opcode byte in the program memory. Whatever data is found there is copied to the destination address.

The mnemonic for immediate data is the pound sign (#). Occasionally, in the rush to meet a deadline, one forgets to use the # for immediate data. The resulting opcode is often a legal command that is assembled with no objections by the assembler. This omission guarantees that the rush will continue.

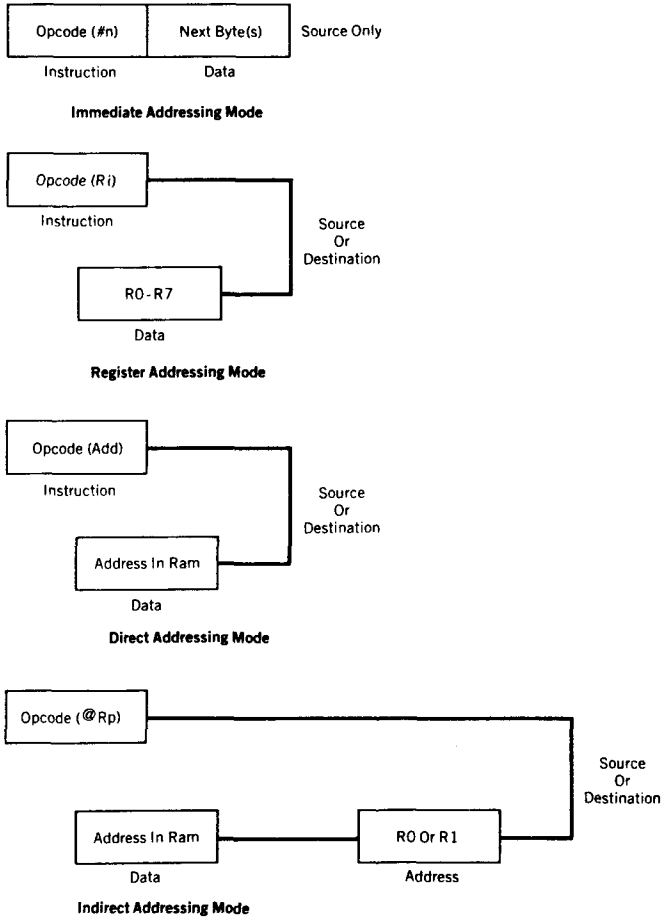
Register Addressing Mode

Certain register names may be used as part of the opcode mnemonic as sources or destinations of data. Registers A, DPTR, and R0 to R7 may be named as part of the opcode mnemonic. Other registers in the 8051 may be addressed using the direct addressing mode. Some assemblers can equate many of the direct addresses to the register name (as is the case with the assembler discussed in this text) so that register names may be used in lieu of register addresses. Remember that the registers used in the opcode as R0 to R7 are the ones that are *currently* chosen by the bank-select bits, RS0 and RS1 in the PSW.

The following table shows all possible MOV opcodes using immediate and register addressing modes:

Mnemonic	Operation
MOV A,#n	Copy the immediate data byte n to the A register
MOV A,Rr	Copy data from register Rr to register A
MOV Rr,A	Copy data from register A to register Rr
MOV Rr,#n	Copy the immediate data byte n to register Rr
MOV DPTR,#nn	Copy the immediate 16-bit number nn to the DPTR register

FIGURE 3.1 Addressing Modes



A data MOV does not alter the contents of the data source address. A *copy* of the data is made from the source and moved to the destination address. The contents of the destination address are replaced by the source address contents. The following table shows examples of MOV opcodes with immediate and register addressing modes:

Mnemonic	Operation
MOV A,#0F1h	Move the immediate data byte F1h to the A register
MOV A,R0	Copy the data in register R0 to register A
MOV DPTR,#0ABCDh	Move the immediate data bytes ABCDh to the DPTR
MOV R5,A	Copy the data in register A to register R5
MOV R3,#1Ch	Move the immediate data byte 1Ch to register R3

CAUTION

It is impossible to have immediate data as a destination.

All numbers *must* start with a decimal number (0–9), or the assembler assumes the number is a *label*.

Register-to-register moves using the register addressing mode occur between registers A and R0 to R7.

Direct Addressing Mode

All 128 bytes of internal RAM and the SFRs may be addressed directly using the single-byte address assigned to each RAM location and each special-function register.

Internal RAM uses addresses from 00 to 7Fh to address each byte. The SFR addresses exist from 80h to FFh at the following locations:

SFR	ADDRESS (HEX)
A	0E0
B	0F0
DPL	82
DPH	83
IE	0A8
IP	0B8
P0	80
P1	90
P2	0A0
P3	0B0
PCON	87
PSW	0D0
SBUF	99
SCON	98
SP	81
TCON	88
TMOD	89
TH0	8C
TL0	8A
TH1	8D
TL1	8B

Note the use of a leading 0 for all numbers that begin with an alphabetic (alpha) character.

RAM addresses 00 to 1Fh are *also* the locations assigned to the four banks of eight working registers, R0 to R7. This assignment means that R2 of register bank 0 can be

addressed in the register mode as R2 or in the direct mode as 02h. The direct addresses of the working registers are as follows:

BANK	REGISTER	ADDRESS (HEX)	BANK	REGISTER	ADDRESS (HEX)
0	R0	00	2	R0	10
0	R1	01	2	R1	11
0	R2	02	2	R2	12
0	R3	03	2	R3	13
0	R4	04	2	R4	14
0	R5	05	2	R5	15
0	R6	06	2	R6	16
0	R7	07	2	R7	17
1	R0	08	3	R0	18
1	R1	09	3	R1	19
1	R2	0A	3	R2	1A
1	R3	0B	3	R3	1B
1	R4	0C	3	R4	1C
1	R5	0D	3	R5	1D
1	R6	0E	3	R6	1E
1	R7	0F	3	R7	1F

Only one bank of working registers is *active* at any given time. The PSW special-function register holds the bank-select bits, RS0 and RS1, which determine which register bank is in use.

When the 8051 is reset, RS0 and RS1 are set to 00b to select the working registers in bank 0, located from 00h to 07h in internal RAM. Reset also sets SP to 07h, and the stack will grow *up* as it is used. This growing stack will overwrite the register banks above bank 0. Be *sure* to set the SP to a number above those of any working registers the program may use.

The programmer may choose any other bank by setting RS0 and RS1 as desired; this bank change is often done to "save" one bank and choose another when servicing an interrupt or using a subroutine.

The moves made possible using direct, immediate, and register addressing modes are as follows:

Mnemonic	Operation
MOV A,add	Copy data from direct address add to register A
MOV add,A	Copy data from register A to direct address add
MOV Rr,add	Copy data from direct address add to register Rr
MOV add,Rr	Copy data from register Rr to direct address add
MOV add,#n	Copy immediate data byte n to direct address add
MOV add1,add2	Copy data from direct address add2 to direct address add1

The following table shows examples of MOV opcodes using direct, immediate, and register addressing modes:

Mnemonic	Operation
MOV A,80h	Copy data from the port 0 pins to register A
MOV 80h,A	Copy data from register A to the port 0 latch
MOV 3Ah,#3Ah	Copy immediate data byte 3Ah to RAM location 3Ah
MOV R0,12h	Copy data from RAM location 12h to register R0

MOV 8Ch,R7	Copy data from register R7 to timer 0 high byte
MOV 5Ch,A	Copy data from register A to RAM location 5Ch
MOV 0A8h,77h	Copy data from RAM location 77h to IE register

MOV instructions that refer to direct addresses above 7Fh that are not SFRs will result in errors. The SFRs are physically on the chip; all other addresses above 7Fh do not physically exist.

Moving data to a port changes the port *latch;* moving data from a port gets data from the port *pins.*

Moving data from a direct address to itself is not predictable and could lead to errors.

Indirect Addressing Mode

For all the addressing modes covered to this point, the source or destination of the data is an absolute number or a name. Inspection of the opcode reveals exactly what are the addresses of the destination and source. For example, the opcode MOV A,R7 says that the A register will get a copy of whatever data is in register R7; MOV 33h,#32h moves the hex number 32 to hex RAM address 33.

The indirect addressing mode uses a register to *hold* the actual address that will finally be used in the data move; the register itself is *not* the address, but rather the number *in* the register. Indirect addressing for MOV opcodes uses register R0 or R1, often called "data pointers," to hold the address of one of the data locations, which could be a RAM or an SFR address. The number that is in the pointing register (Rp) cannot be known unless the history of the register is known. The mnemonic symbol used for indirect addressing is the "at" sign, which is printed as @.

The moves made possible using immediate, direct, register and indirect addressing modes are as follows:

Mnemonic	**Operation**
MOV @Rp,#n	Copy the immediate byte n to the address in Rp
MOV @Rp,add	Copy the contents of add to the address in Rp
MOV @Rp,A	Copy the data in A to the address in Rp
MOV add,@Rp	Copy the contents of the address in Rp to add
MOV A,@Rp	Copy the contents of the address in Rp to A

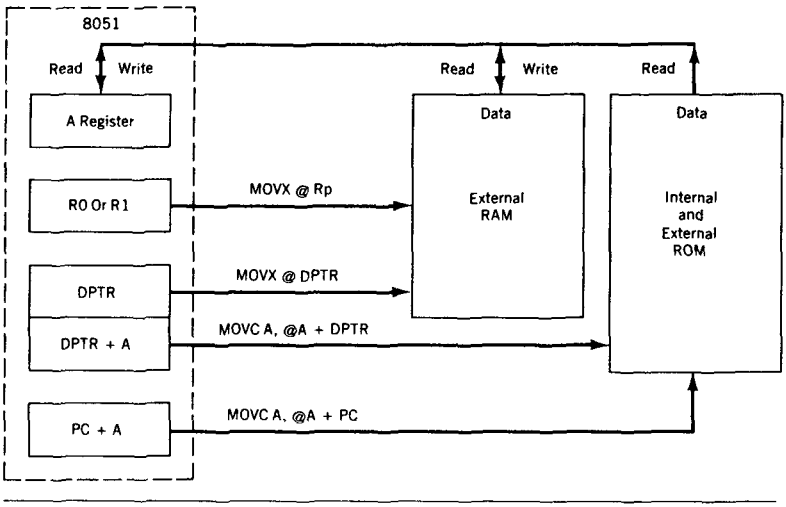
The following table shows examples of MOV opcodes, using immediate, register, direct, and indirect modes

Mnemonic	**Operation**
MOV A,@R0	Copy the contents of the address in R0 to the A register
MOV @R1,#35h	Copy the number 35h to the address in R1
MOV add,@R0	Copy the contents of the address in R0 to add
MOV @R1,A	Copy the contents of A to the address in R1
MOV @R0,80h	Copy the contents of the port 0 pins to the address in R0

The number in register Rp must be a RAM or an SFR address.

Only registers R0 or R1 may be used for indirect addressing.

FIGURE 3.2 External Addressing using MOVX and MOVC



External Data Moves

As discussed in Chapter 2, it is possible to expand RAM and ROM memory space by adding external memory chips to the 8051 microcontroller. The external memory can be as large as 64K bytes for each of the RAM and ROM memory areas. Opcodes that access this external memory *always* use indirect addressing to specify the external memory.

Figure 3.2 shows that registers R0, R1, and the aptly named DPTR can be used to hold the address of the data byte in external RAM. R0 and R1 are limited to external RAM address ranges of 00h to 0FFh, while the DPTR register can address the maximum RAM space of 0000h to 0FFFFh.

An X is added to the MOV mnemonics to serve as a reminder that the data move is external to the 8051, as shown in the following table.

Mnemonic	Operation
MOVX A,@Rp	Copy the contents of the external address in Rp to A
MOVX A,@DPTR	Copy the contents of the external address in DPTR to A
MOVX @Rp,A	Copy data from A to the external address in Rp
MOVX @DPTR,A	Copy data from A to the external address in DPTR

The following table shows examples of external moves using register and indirect addressing modes:

Mnemonic	Operation
MOVX @DPTR,A	Copy data from A to the 16-bit address in DPTR
MOVX @R0,A	Copy data from A to the 8-bit address in R0

MOVX A,@R1	Copy data from the 8-bit address in R1 to A
MOVX A,@DPTR	Copy data from the 16-bit address in DPTR to A

CAUTION

All external data moves must involve the A register.

Rp can address 256 bytes; DPTR can address 64K bytes.

MOVX is normally used with external RAM or I/O addresses.

Note that there are two sets of RAM addresses between 00 and 0FFh: one internal and one external to the 8051.

Code Memory Read-Only Data Moves

Data moves between RAM locations and 8051 registers are made by using MOV and MOVX opcodes. The data is usually of a temporary or "scratch pad" nature and disappears when the system is powered down.

There are times when access to a preprogrammed mass of data is needed, such as when using tables of predefined bytes. This data must be permanent to be of repeated use and is stored in the program ROM using assembler directives that store programmed data anywhere in ROM that the programmer wishes.

Access to this data is made possible by using indirect addressing and the A register in conjunction with either the PC or the DPTR, as shown in Figure 3.2. In both cases, the number in register A is *added* to the pointing register to form the address in ROM where the desired data is to be found. The data is then fetched from the ROM address so formed and placed in the A register. The original data in A is lost, and the addressed data takes its place.

As shown in the following table, the letter C is added to the MOV mnemonic to highlight the use of the opcodes for moving data from the source address in the Code ROM to the A register in the 8051:

Mnemonic	Operation
MOVC A,@A+DPTR	Copy the code byte, found at the ROM address formed by adding A and the DPTR, to A
MOVC A,@A+PC	Copy the code byte, found at the ROM address formed by adding A and the PC, to A

Note that the DPTR and the PC are not changed; the A register contains the ROM byte found at the address formed.

The following table shows examples of code ROM moves using register and indirect addressing modes:

Mnemonic	Operation
MOV DPTR,#1234h	Copy the immediate number 1234h to the DPTR
MOV A,#56h	Copy the immediate number 56h to A
MOVC A,@A+DPTR	Copy the contents of address 128Ah to A
MOVC A,@A+PC	Copies the contents of address 4059h to A if the PC contained 4000h and A contained 58h when the opcode is executed.

CAUTION

The PC is incremented by one (to point to the next instruction) *before* it is added to A to form the final address of the code byte.

All data is moved *from* the code memory *to* the A register.

MOVC is normally used with internal or external ROM and can address 4K of internal or 64K bytes of external code.

PUSH and POP Opcodes

The PUSH and POP opcodes specify the direct address of the data. The data moves between an area of internal RAM, known as the stack, and the specified direct address. The stack pointer special-function register (SP) contains the address in RAM where data *from* the source address will be PUSHed, or where data to be POPed *to* the destination address is found. The SP register actually is used in the indirect addressing mode but is *not* named in the mnemonic. It is *implied* that the SP holds the indirect address whenever PUSHing or POPing. Figure 3.3 shows the operation of the stack pointer as data is PUSHed or POPed to the stack area in internal RAM.

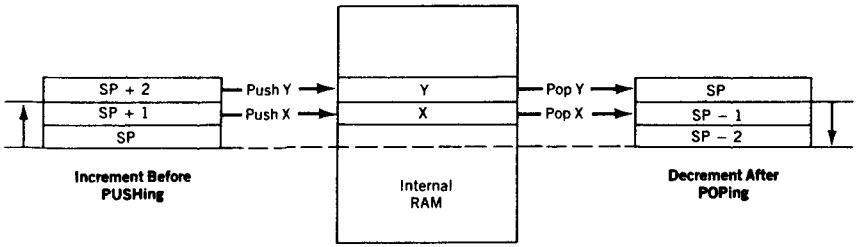
A PUSH opcode copies data from the source address to the stack. SP is *incremented* by one *before* the data is copied to the internal RAM location contained in SP so that the data is stored from low addresses to high addresses in the internal RAM. The stack grows *up* in memory as it is PUSHed. Excessive PUSHing can make the stack exceed 7Fh (the top of internal RAM), after which point data is lost.

A POP opcode copies data from the stack to the destination address. SP is *decremented* by one *after* data is copied from the stack RAM address to the direct destination to ensure that data placed on the stack is retrieved in the same order as it was stored.

The PUSH and POP opcodes behave as explained in the following table:

Mnemonic	Operation
PUSH add	Increment SP; copy the data in add to the internal RAM address contained in SP
POP add	Copy the data from the internal RAM address contained in SP to add; decrement the SP

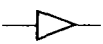
FIGURE 3.3 PUSH and POP the Stack



The SP register is set to 07h when the 8051 is reset, which is the same direct address in internal RAM as register R7 in bank 0. The first PUSH opcode would write data to R0 of bank 1. The SP should be initialized by the programmer to point to an internal RAM address above the highest address likely to be used by the program.

The following table shows examples of PUSH and POP opcodes:

Mnemonic	Operation
MOV 81h,#30h	Copy the immediate data 30h to the SP
MOV R0, #0ACh	Copy the immediate data ACh to R0
PUSH 00h	SP = 31h; address 31h contains the number ACh
PUSH 00h	SP = 32h; address 32h contains the number ACh
POP 01h	SP = 31h; register R1 now contains the number ACh
POP 80h	SP = 30h; port 0 latch now contains the number ACh



CAUTION

When the SP reaches FFh it "rolls over" to 00h (R0).

RAM ends at address 7Fh; PUSHes above 7Fh result in errors.

The SP is usually set at addresses above the register banks.

The SP may be PUSHed and POPed to the stack.

Note that direct addresses, *not* register names, must be used for most registers. The stack mnemonics have no way of knowing which bank is in use.

Data Exchanges

MOV, PUSH, and POP opcodes all involve copying the data found in the source address to the destination address; the original data in the source is not changed. Exchange instructions actually move data in two directions: from source to destination and from destination to source. All addressing modes except immediate may be used in the XCH (exchange) opcodes:

Mnemonic	Operation
XCH A,Rr	Exchange data bytes between register Rr and A
XCH A,add	Exchange data bytes between add and A
XCH A,@Rp	Exchange data bytes between A and address in Rp
XCHD A,@Rp	Exchange lower nibble between A and address in Rp

Exchanges between A and any port location copy the data on the port *pins* to A, while the data in A is copied to the port *latch*. Register A is used for so many instructions that the XCH opcode provides a very convenient way to "save" the contents of A without the necessity of using a PUSH opcode and then a POP opcode.

The following table shows examples of data moves using exchange opcodes:

Mnemonic	Operation
XCH A,R7	Exchange bytes between register A and register R7
XCH A,0F0h	Exchange bytes between register A and register B
XCH A,@R1	Exchange bytes between register A and address in R1
XCHD A,@R1	Exchange lower nibble in A and the address in R1

CAUTION

All exchanges are *internal* to the 8051.

All exchanges use register A.

When using XCHD, the upper nibble of A and the upper nibble of the address location in Rp do not change.

This section concludes the listing of the various data moving instructions; the remaining sections will concentrate on using these opcodes to write short programs.

Example Programs

Programming is at once a skill and an art. Just as anyone may learn to play a musical instrument after sufficient instruction and practice, so may anyone learn to program a computer. Some individuals, however, have a gift for programming that sets them apart from their peers with the same level of experience, just as some musicians are more talented than their contemporaries.

Gifted or not, you will not become adept at programming until you have written and rewritten many programs. The emphasis here is on practice; you can read many books on how to ride a bicycle, but you do not know how to ride until you do it.

If some of the examples and problems seem trivial or without any "real-world" application, remember the playing of scales on a piano by a budding musician. Each example will be done using several methods; the best method depends upon what resource is in short supply. If programming time is valuable, then the best program is the one that uses the fewest lines of code; if either ROM or execution time is limited, then the program that uses the fewest code bytes is best.

EXAMPLE PROBLEM 3.1

Copy the byte in TCON to register R2 using at least four different methods.

- **Method 1:** Use the direct address for TCON (88h) and register R2.

Mnemonic	Operation
MOV R2,88h	Copy TCON to R2

- **Method 2:** Use the direct addresses for TCON and R2.

Mnemonic	Operation
MOV 02h,88h	Copy TCON to direct address 02h (R2)

- **Method 3:** Use R1 as a pointer to R2 and use the address of TCON.

Mnemonic	Operation
MOV R1,#02h	Use R1 as a pointer to R2
MOV @R1,88h	Copy TCON byte to address in R1 (02h = R2)

- **Method 4:** Push the contents of TCON into direct address 02h (R2).

Mnemonic	Operation
MOV 81h,#01h	Set the SP to address 01h in RAM
PUSH 88h	Push TCON (88h) to address 02h (R2)

EXAMPLE PROBLEM 3.2

Set timer T0 to an initial setting of 1234h.

- **Method 1:** Use the direct address with an immediate number to set TH0 and TL0.

Mnemonic	Operation
MOV 8Ch,#12h	Set TH0 to 12h
MOV 8Ah,#34h	Set TL0 to 34h
	Totals: 6 bytes, 2 lines

- **Method 2:** Use indirect addressing with R0 for TL0 and R1 for TH0.

Mnemonic	Operation
MOV R0,#8Ah	Copy 8Ah, the direct address of TL0, to R0
MOV R1,#8Ch	Copy 8Ch, the direct address of TH0, to R1
MOV @R0,#34h	Copy 34h to TL0
MOV @R1,#12h	Copy 12h to TH0
	Totals: 8 bytes, 4 lines

The first method is also the better method in this example.

EXAMPLE PROBLEM 3.3

Put the number 34h in registers R5, R6, and R7.

- **Method 1:** Use an immediate number and register addressing.

Mnemonic	Operation
MOV R5,#34h	Copy 34h to R5
MOV R6,#34h	Copy 34h to R6
MOV R7,#34h	Copy 34h to R7
	Totals: 6 bytes, 3 lines

- **Method 2:** Since the number is the same for each register, put the number in A and MOV A to each register.

Mnemonic	Operation
MOV A,#34h	Copy a 34h to A
MOV R5,A	Copy A to R5
MOV R6,A	Copy A to R6
MOV R7,A	Copy A to R7
	Totals: 5 bytes, 4 lines

- **Method 3:** Copy one direct address to another.

Mnemonic	Operation
MOV R5,#34h	Copy 34h to register R5
MOV 06h,05h	Copy R5 (add 05) to R6 (add 06)
MOV 07h,06h	Copy R6 to R7
	Totals: 8 bytes, 3 lines

EXAMPLE PROBLEM 3.4

Put the number 8Dh in RAM locations 30h to 34h.

- **Method 1:** Use the immediate number to a direct address:

Mnemonic	Operation
MOV 30h,#8Dh	Copy the number 8Dh to RAM address 30h
MOV 31h,#8Dh	Copy the number 8Dh to RAM address 31h
MOV 32h,#8Dh	Copy the number 8Dh to RAM address 32h
MOV 33h,#8Dh	Copy the number 8Dh to RAM address 33h
MOV 34h,#8Dh	Copy the number 8Dh to RAM address 34h
	Totals: 15 bytes, 5 lines

- **Method 2:** Using the immediate number in each instruction uses bytes; use a register to hold the number:

Mnemonic	Operation
MOV A,#8Dh	Copy the number 8Dh to the A register
MOV 30h,A	Copy the contents of A to RAM location 30h
MOV 31h,A	Copy the contents of A to the remaining addresses
MOV 32h,A	
MOV 33h,A	
MOV 34h,A	Totals: 12 bytes, 6 lines

- **Method 3:** There must be a way to avoid naming each address; the PUSH opcode can increment to each address:

Mnemonic	Operation
MOV 30h,#8Dh	Copy the number 8Dh to RAM address 30h
MOV 81h,#30h	Set the SP to 30h
PUSH 30h	Push the contents of 30h (=8Dh) to address 31h
PUSH 30h	Continue pushing to address 34h
PUSH 30h	
PUSH 30h	Totals: 14 bytes, 6 lines

COMMENT

Indirect addressing with the number in A and the indirect address in R1 could be done; however, R1 would have to be loaded with each address from 30h to 34h. Loading R1 would take a total of 17 bytes and 11 lines of code. Indirect addressing is advantageous when we have opcodes that can change the contents of the pointing registers automatically.

Summary

The opcodes that move data between locations within the 8051 and between the 8051 and external memory have been discussed. The general form and results of these instructions are as follows.

Instruction Type	Result
MOV destination,source	Copy data from the internal RAM source address to the internal RAM destination address

MOVC A,source	Copy internal or external program memory byte from the source to register A
MOVX destination,source	Copy byte to or from external RAM to register A
PUSH source	Copy byte to internal RAM stack from internal RAM source
POP destination	Copy byte from internal RAM stack to internal RAM destination
XCH A,source	Exchange data between register A and the internal RAM source
XCHD A,source	Exchange lower nibble between register A and the internal RAM source

There are four addressing modes: an immediate number, a register name, a direct internal RAM address, and an indirect address contained in a register.

Problems

Write programs that will accomplish the desired tasks listed below, using as few lines of code as possible. Use only opcodes that have been covered up to this chapter. Comment on each line of code.

1. Place the number 3Bh in internal RAM locations 30h to 32h.
2. Copy the data at internal RAM location F1h to R0 and R3.
3. Set the SP at the byte address just above the last working register address.
4. Exchange the contents of the SP and the PSW.
5. Copy the byte at internal RAM address 27h to external RAM address 27h.
6. Set Timer 1 to A23Dh.
7. Copy the contents of DPTR to registers R0 (DPL) and R1 (DPH).
8. Copy the data in external RAM location 0123h to TL0 and the data in external RAM location 0234h to TH0.
9. Copy the data in internal RAM locations 12h to 15h to internal RAM locations 20h to 23h: Copy 12h to 20h, 13h to 21h, etc.
10. Set the SP register to 07h and PUSH the SP register on the stack; predict what number is PUSHed to address 08h.
11. Exchange the contents of the B register and external RAM address 02CFh.
12. Rotate the bytes in registers R0 to R3; copy the data in R0 to R1, R1 to R2, R2 to R3, and R3 to R0.
13. Copy the external code byte at address 007Dh to the SP.
14. Copy the data in register R5 to external RAM address 032Fh.
15. Copy the internal code byte at address 0300h to external RAM address 0300h.
16. Swap the bytes in timer 0; put TL0 in TH0 and TH0 in TL0.
17. Store DPTR in external RAM locations 0123h (DPL) and 02BCh (DPH).
18. Exchange both low nibbles of registers R0 and R1; put the low nibble of R0 in R1, and the low nibble of R1 in R0.

19. Store the contents of register R3 at the internal RAM address contained in R2. (Be sure the address in R2 is legal.)
20. Store the contents of RAM location 20h at the address contained in RAM location 08h.
21. Store register A at the internal RAM location address in register A.
22. Copy program bytes 0100h to 0102h to internal RAM locations 20h to 22h.
23. Copy the data on the pins of port 2 to the port 2 latch.
24. PUSH the contents of the B register to TMOD.
25. Copy the contents of external code memory address 0040h to IE.
26. Show that a set of XCH instructions executes faster than a PUSH and POP when saving the contents of the A register.