4

Addressing Modes and Data Movement Instructions

Objectives

- Introduce the data movement instructions and their significance
- List and classify the addressing modes of the 8051
- Compare the addressing modes along with their uses
- Classify the 8051 addressing space with respect to instruction set
- Discuss notations used for the data of different number systems
- Discuss the operand modifiers # and @
- List the instructions for external data and code memory access
- Describe PUSH, POP and data exchange instructions
- Develop the programs of data movement within any of the address space of the 8051

Key Terms

- Address Space
- Addressing Modes
- Data Memory
- Data Movement
- Direct Addressing Mode
- External Memory Access
- Immediate Addressing Mode
- Indexed Addressing Mode
- Indirect Addressing Mode
- Operand Modifiers :# and @
- Program Addressing Modes
- Program Memory
- Register Addressing
- Register Addressing Mode
- Stack

Introduction to the Instruction Set

Every microcontroller has its own instruction set, the designer of the microcontroller decides these instructions based on the number and type of operations it is required to perform. The instructions are classified according to type of operation performed by them. All instructions can be divided into four categories.

1. Data movement instructions
2. Arithmetic instructions
3. Logical instructions
4. Program flow control instructions

All types of instructions are discussed in detail in subsequent chapters. To start with concepts of assembly language programming, the data movement instructions are discussed in this chapter.

4.1 WHY DATA MOVEMENT INSTRUCTIONS FIRST?

The microcontroller (or any computer system) typically spends maximum time in data movement operations. Therefore, maximum instructions are provided for the data movement operations than for any other type of operation, thus variety of data movement instructions of a microcontroller determines flexibility and ease with which efficient programs can be developed. Moreover, it is easy to understand programming model of any microcontroller with data movement instructions. The 8051 have 30 (27 for 8 bits +1 for 16 bits + 2 for 1 bit) instructions only for data movement out of total 111 instructions. It comes out to be 27%!!!

4.2 ADDRESSING MODES

The way by which source or destination (usually source) operands are specified in an instruction is called *addressing mode*. The instruction may contain one, two or no operand. The data is accessed from source address, processed in the ALU and stored at destination address. The ways by which these data (or address of data) are specified are called addressing modes.

There are essentially four addressing modes used by all microcontrollers (or computer systems), all other modes commonly found in microcontroller/processor literature are types of these four basic addressing modes. They are,

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Indirect addressing

Yet, there is another way of classifying addressing modes based on whether data is accessed or new instruction is accessed (i.e. instruction execution flow is changed) as a result of instruction execution. They are data addressing and program addressing modes. Data movement, arithmetic and logical instructions use the data addressing modes and program flow control instruction uses the program addressing modes.

The 8051 supports all addressing modes as discussed above, they are summarized in Table 4.1.

Table 4.1 *Addressing modes of the 8051*

Data addressing modes	Program addressing modes
1. Immediate addressing	1. Short (relative) addressing
2. Register addressing	2. Absolute addressing
3. Direct addressing	3. Long (direct) addressing
4. Indirect addressing • Register indirect addressing • Indexed addressing	

Before we start discussion of the addressing modes and type of the instructions, we should know types of memories from where data are accessed by different instructions. The memory of the 8051 may be divided in five types of address spaces. The operands are located in any of the address spaces. The five address spaces are,

1. Bank registers (R0-R7)

2. Accumulator
3. Internal data RAM (00-7F & SFRs)
4. External data RAM (data memory)
5. Program ROM (internal + external ROM)

Note that the Accumulator is considered as a special case even though it is a SFR. In addition with these address spaces, one more place from where data can be obtained is from instruction itself, i.e. in case of immediate addressing mode, data is an integral part of the instruction.

With this basic understanding of the address spaces, let us discuss each addressing mode in detail.

Acronyms used in the Instructions

Acronyms used by Intel in the 8051 instructions are used throughout the book.

data : 8-bit data
data16 : 16-bit data
direct : address in internal RAM or SFRs
Rn : R0 to R7
Ri : R0 or R1

4.2.1 Immediate Addressing Mode

The data (constant) is specified as a part of instruction in a program memory. The data is available immediately as a part of instruction itself, therefore immediate addressing is very fast. However, since the data is fixed, at runtime it is not flexible. The instructions using an immediate operand have an 8-bit or 16-bit number following the op-code. For example,

```
MOV A, #data        // load 8 bit immediate data into Accumulator
MOV A, #55H         // A= 55H

MOV Rn, #data       // load 8 bit immediate data into Rn
MOV R3, #0FFH       // R3= FFH

MOV DPTR, #data16   // load 16-bit number into DPTR
MOV DPTR, #2000H    // DPTR=2000H
```

Note, in instruction 'MOV R3, #0FFH', 0 is used between # and FF to indicate that F is a number and not a letter. This is a requirement for assembler, not a microcontroller. The operation of instruction MOV R0, #10H is shown in Figure 4.1.

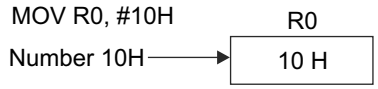

Fig. 4.1 *Immediate addressing mode*

The symbol for the immediate data is pound sign (#). The omission of # sign will result in a valid instruction with other addressing mode but with change in meaning of the instruction. During the execution of the immediate data movement instruction, the program counter is automatically incremented to point to data byte or (bytes for 16-bit number) immediately following the op-code byte; the data byte is copied to the destination. Note that the immediate data byte can not be a destination in an instruction.

- MOV (move) actually means copy data from source to destination.
- The data movement instruction does not affect the flags.

Example 4.1

Write instructions to initialize registers R0, R1 and R2 with values 10H, 20H and 30H respectively.

Solution:

The required operation can be performed by loading immediate values into registers. (Use of immediate addressing mode)

```
MOV R0, #10H        // initialize R0 with immediate value 10H
MOV R1, #20H        // initialize R1 with immediate value 20H
MOV R2, #30H        // initialize R2 with immediate value 30H
```

If bank 0 is selected, same operations can be done by following instructions:

```
MOV 00H, #10H      // initialize memory location 00 (R0) with value 10H
MOV 01H, #20H      // initialize memory location 01 (R1) with value 20H
MOV 02H, #30H      // initialize memory location 02 (R2) with value 30H
```

1. *Notations for Numbers of Different Number Systems*

Microcontroller understands only binary language. Hexadecimal numbers are compact representation of binary numbers so that it becomes easy to handle them. In above examples, the hexadecimal numbers are represented with suffix 'H'. If we want to work with decimal (or octal, ASCII) numbers directly, it is not allowed because they are not understood by the microcontrollers. However, assemblers provide the facility to work with all types of numbers (decimal, octal, ASCII, negative) numbers. The assembler will finally convert all numbers in to hexadecimal before generating machine codes. The format for numbers of different number systems is summarized in Table 4.2 with examples.

Table 4.2 *Notations of numbers of different number systems*

Number type	Suffix	Example	Instruction	Assembled instruction
Binary	B, b	11111010b	MOV A, #10001010b	MOV A, #8AH
		11101011B	MOV A,#11101011B	MOV A, #0EBH
Decimal	D, d	15D	MOV A, #15D	MOV A, #0FH
		25d	MOV A, #25d	MOV A, #19H
		115	MOV A, #115	MOV A, #73H
Octal	O, o,	20O,20o	MOV A, #20O	MOV A, #10H
	Q, q	75Q,75q	MOV A, #75Q	MOV A, #3DH
Hexadecimal	H, h,	25H	MOV A, #25H	MOV A, #25H
	0x (prefix)	FFh	MOV A, #0FFh	MOV A, #0FFH
		0x80	MOV A, #0x80	MOV A, #80H
ASCII	No suffix	A-Z, a-z, 0-9	MOV A, # 'C'	MOV A, #43H
Negative	-	-12, -50	MOV A, #-12	MOV A, #0F4H*

* F4 is 2's complement of decimal 12

> ***Note:*** Numbers without any suffix are decimal numbers

2. *Use of Expressions*

Mathematical or logical expressions may be used in an instruction in place of any numerical value. For example,

MOV A, #10H+20H is assembled as MOV A, #30H and
MOV 10+20, R0 is same as MOV 30, R0 or MOV 1EH, R0

Refer assembler user's guide for more details of use and precedence of expressions.

Example 4.2

How are the following instructions assembled?

(i) MOV A, #20 **(ii) MOV R0, #-1**
(iii) MOV R1, #25H **(iv) MOV R3, #10o**
(v) MOV R1, #15+20 **(vi) MOV R3, #01010101B**
(vii) MOV DPTR, #0x1000 **(viii) MOV DPTR, #1000**

Solution:

(i) MOV A, #0x14 (ii) MOV R0, #0xFF
(iii) MOV R1, #0x25 (iv) MOV R3,#0x08
(v) MOV R1, #0x23 (vi) MOV R3, #0x55
(vii) MOV DPTR, #0x1000 (viii) MOV DPTR, #0x3E8

4.2.2 Register Addressing Mode

In register addressing mode, the operands are specified by register names. Register A and R0 to R7 may be named as a part of the instruction mnemonic. The advantage of register addressing mode is that it occupies only one byte memory,

and is fast because only on-chip registers are accessed, i.e. instruction takes only one machine cycle for execution. For example,

```
MOV A, Rn    // copy contents of register Rn to Accumulator
MOV A, R2    // If R2=10H → A=10H

MOV Rn, A    // copy contents of Accumulator to register Rn
MOV R1, A    // If A=20H → R1=20H
```

The operation of instruction MOV A, R0 is shown in Figure 4.2.

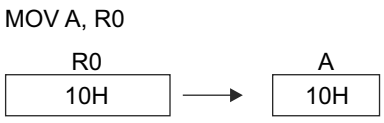

Fig. 4.2 *Register addressing mode*

4.2.3 Direct Addressing Mode

The data is accessed directly from the memory address specified as one of the operand, i.e. one of the operand is an 8-bit address for internal RAM location. Internal RAM includes 128 bytes of RAM from (00H–7FH) and any special function register. It is more flexible compared to immediate and register addressing because the value to be accessed from address may be variable. These are 2-byte instructions (3 bytes when source and destination are both direct addresses). The address refers to either byte location or a specific bit in a bit addressable byte. For example,

```
MOV A, direct        // copy data from (contents of) address direct in to A
MOV A, 10H           // If address 10H contains data 50H i.e (10H) = 50H → A=50H

MOV direct, A        // copy data from A to address direct
MOV 10H, A           // If A=44H → (10H) = 44H

MOV Rn, direct       // copy data from address direct into register Rn
MOV R5, 80H          // If (80H)= FFH → R5 = FFH

MOV direct, Rn       // copy data from Rn to address direct
MOV 50H, R3          // R3=10H → (50H) = 10H

MOV direct, #data    // load 8-bit immediate data in to address direct
MOV 0A0H, #20H       // (A0H) = 20H

MOV direct1, direct2 // copy data from address direct2 to address direct1
MOV 50H, 83H         // If (83H)=10H → (50H) =10H
```

Note that the brackets '()' specify the contents of the address specified in them and may be read as 'the contents of'.

The operation of instruction MOV R0, 15H is shown in Figure 4.3.

The programmer may use numeric address or name for any SFR. For example, following two instructions does the same operation.

```
MOV P0, #55H    // load constant value 55H into port0 latch
MOV 80H, #55H   // load constant value 55H into port0 latch
```

It should be noted that the access to memory locations between 80H to FFH that do not exist physically will result into errors. Refer Figure 2.2 (Programming model of the 8051) for SFR addresses. Bit addressable data movement instructions are discussed in detail in Chapter 6.

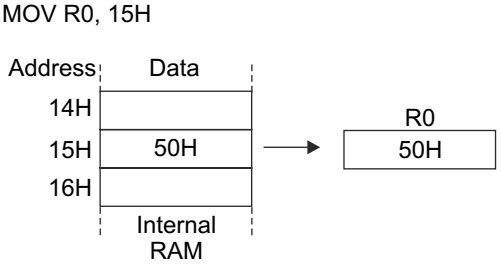

Fig. 4.3 *Direct addressing mode*

THINK BOX 4.1

Can we modify P (parity) bit by writing data directly in to PSW? What will be status of P flag after execution of each of the following instructions?
MOV A, # 07H
MOV PSW, #00H

No. When we write data to the PSW, parity bit remains unchanged because it is affected only by hardware to reflect the parity of A.
After execution of first instruction (MOV A, # 07H), P=1, because A contains odd number of 1's.
After execution of second instruction (MOV PSW, #00H), P remains unaffected (P=1) because its value is affected only by contents of A.

Example 4.3

Illustrate how to load 8-bit data into registers, internal RAM, and moving data between registers and internal RAM.

Solution:

```
MOV   A, #10H      // Load number10H into A register, A=10H (immediate addressing mode)
MOV   R1, A        // Copy contents of A to register R1, R1=10H (Moving data between registers, Register addressing mode)
MOV   R0, A        // R0=10H, copy contents of A to R0 of selected bank (Register addressing mode)
MOV   10H, #20H    // (10H) =20H, load immediate value 20H into internal RAM address 10H
MOV   R1, 10H      // R1= 20H (data present at internal RAM address 10H)
MOV   20H, R1      // (20H) =20H load contents of R1 into internal RAM address 20H
MOV   30H, 10H     // (30H) = (10H) copy contents of internal RAM 10H into 30H
```

Example 4.4

Copy the contents of the DPTR to internal RAM address 10H (DPL) and 11H (DPH)

Solution:

The DPTR is a 16-bit register and internal addresses 10H and 11H are only 8-bit registers, therefore we will move lower byte of DPTR (DPL) into address 10H and higher byte of DPTR (DPH) in to address 11H.

```
MOV 10H, DPL
MOV 11H, DPH
```

4.2.4 Indirect Addressing Mode

The data is specified indirectly in an instruction, i.e. address of the data (rather than data itself) is specified as one of the operand. Here, the register holds the address that contains the data, i.e. the number in the register is treated as a pointer to address. Indirect addressing mode is the most flexible and useful in array operations. There are two types of indirect addressing in the 8051.

1. Register Indirect Addressing Mode

The register indirect addressing uses only register R0 or R1 to hold address of the data in internal RAM, these two registers are also referred to as pointer registers or simply pointers. The symbol @ is used along with R0 or R1 to indicate indirect addressing. For example,

MOV @*Ri*, *#data*	// load constant value in to address contained in *Ri*
MOV @R0, #30H	// If R0=40H, → (40H)=30H
MOV @*Ri, direct*	// copy data form address *direct* to address *Ri*
MOV @R1, 10H	// If (10H)=50H, R1=15H →(15H)=50H
MOV *direct,* @*Ri*	// copy data from address in *Ri* to address *direct*
MOV 10H, @R1	// If R1=50H, (50H)=15H → (10H)=15H
MOV @*Ri,* A	// copy data from A to address in *Ri*
MOV @R0, A	// If A=50H, R0=15H → (15H)=50H
MOV A, @*Ri*	// copy data from address in *Ri* to A
MOV A, @R1	// If R1=50H, (50H)=15H → A=15H

The operation of instruction MOV A, @R1 is shown in Figure 4.4.

Indirect addressing can access internal as well as external memory area. The indirect addresses can be 8 bits or 16 bits. The 8-bit addresses, as mentioned above, are held by R0, R1 or SP only. The 16-bit addresses are held by DPTR, for external memory. External memory data-movement instructions are discussed in Section 4.4.1.

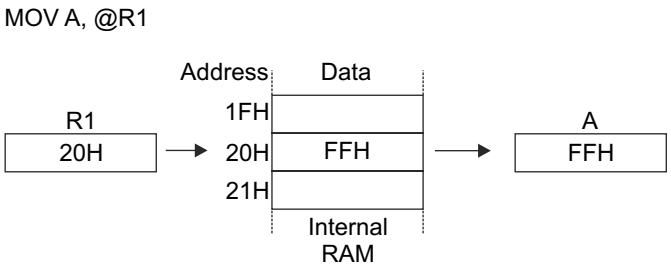

Fig. 4.4 *Indirect addressing mode*

Note: Indirect addressing is used only for accessing RAM locations 00H to 7FH (128 memory locations) and never for accessing SFRs. However, it can access additional 128 RAM locations (from 80H to FFH) available in later versions of the 8051.

Example 4.5

Discuss different methods for moving contents of R2 to R1 (Assume that register bank 0 is selected). Compare all methods with respect to program size and execution speed. Which method is the best?

Solution:

```
(i) Using register addressing
        MOV A, R2                   // 1 byte, 1 machine cycle
        MOV R1, A                   // 1 byte, 1 machine cycle
                                    // Total= 2 bytes, 2 machine cycles (Best method)
(ii) Using register and direct addressing
        MOV A, 02H                  // 2 byte, 1 machine cycle, direct addressing
        MOV 01H, A                  // 2 byte, 1 machine cycle, register addressing
                                    // Total= 4 bytes, 2 machine cycles
(iii) Using direct addressing
        MOV 01H, 02H                // 3 byte, 2 machine cycle, direct addressing
                                    // Total= 3 bytes, 2 machine cycles
(iv) Using register and direct addressing
        MOV 10H, R2                 // 2 byte, 2 machine cycle, register addressing
        MOV R1, 10H                 // 2 byte, 2 machine cycle, direct addressing
                                    // Total= 4 bytes, 4 machine cycles
(v) Using indirect addressing
        MOV R0, #02H                // 2 byte, 1 machine cycle, immediate addressing
        MOV A, @R0                  // 1 byte, 1 machine cycle, indirect addressing
        MOV R1, A                   // 1 byte, 1 machine cycle, register addressing
                                    // Total= 4 bytes, 3 machine cycles
```

2. Indexed Addressing Mode

Two registers are used to form the address of the data. The contents of either DPTR or PC are used as a base address and the A is used as index (or offset) address. The final address is formed by adding these two registers. It results in a forward reference of 0 to 255 bytes from the base address. They are used to access only program memory (internal as well as external.)

Indexed addressing is used to access data tables (lookup tables) from the program memory and implementing jump tables. They are also suitable for multidimensional array operations.

The instructions are,

```
MOVC A, @A+PC       // copy data (or code) byte from program memory address formed by
                    //addition of contents of A and PC into A

MOVC A, @A+DPTR     // copy data (or code) byte from program memory address formed by
                    // addition of contents of A and DPTR into A
```

The mnemonic MOVC means ‘move constant’. The term ‘constant’ indicates that the contents of program memory can only be read but cannot be modified by instructions, they remain constant. As a memory aid the mnemonic MOVC may also be considered as ‘move from code memory’!

The operation of instruction MOVC A, @A+DPTR is shown in Figure 4.5.

Note: For MOVC A, @A+PC, the PC is incremented by 1 (to point to the next instruction) before added to A to form effective address of the code byte. The original data in A is lost and addressed data is placed in the A. The concept of data

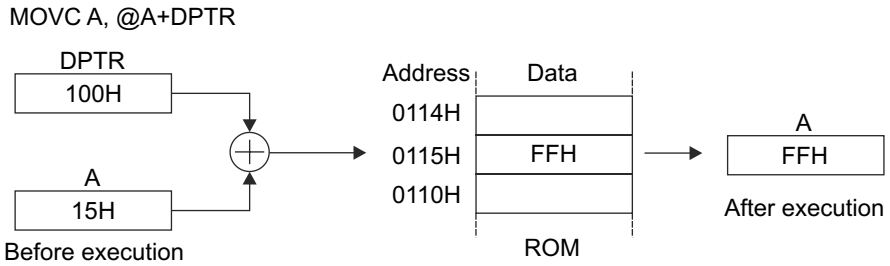

Fig. 4.5 *Indexed addressing mode*

table is discussed in detail in Chapter 8.The use of instruction MOVC A, @A+PC to access look-up tables is illustrated in Example 8.1.

> ***Note:*** Data movement instructions do not change source operands

Example 4.6

Read the contents of the program memory address 50H and copy it into external data memory (RAM) address 50H as well as 100H.

Solution:

Since we have to read data from program memory (ROM), we have to use MOVC' instruction, and to write data in to external RAM, we have to use 'MOVX' instruction.

```
MOV DPTR, #0050H      // initialize DPTR with desired address
MOV A, #00H
MOVC A, @ A+DPTR      // read program memory from effective address
MOVX @DPTR, A         // copy A into external RAM pointed by DPTR (50H)
MOV DPTR, #100H       // point DPTR to 100H
MOVX @DPTR, A         // copy A into external RAM pointed by DPTR (1000H)
```

4.3 OPERAND MODIFIERS: # AND @

As mentioned previously, the 8051 instructions uses symbols # and @ symbols to distinguish addressing mode. The symbol # written before operand indicates that it is an immediate operand while the symbol @ placed before an operand indicates that indirect addressing mode is used.

```
MOV A, #50H      // 50H is immediate operand
MOV A, 50H       // 50H is direct operand
MOV A, R1        // R1 is register operand
MOV A, @R1       // R1 is indirect register operand
```

It is important to note that these two symbols are the potential reasons for the logical errors in the programs, i.e. forgetting them while typing an instruction may result into program with no syntax error but the program will not function as desired. It is difficult and time consuming to find such errors.

Example 4.7

What would be the contents of A and address 10(decimal) after executing the following instructions?

```
MOV R0, #10H
MOV 10H, #20
MOV 35H, #40H
MOV 35, #44H
MOV A, 35
MOV 10, @R0
```

Solution:

A = 44H, (10) = (0AH) = 20d =14H

Example 4.8

The machine code for the instruction MOV A, #10H is 74 10.
Identify the following for the above instruction.
(i) addressing mode (ii) mnemonic (iii) op-code
(iv) source operand (v) destination operand (vi) operation performed

Solution:

(i) addressing mode: Immediate
(ii) mnemonic: MOV
(iii) op-code: 74
(iv) source operand: immediate number 10H
(v) destination operand: A
(vi) operation performed: Immediate number 10H is loaded into A

Summary of the addressing modes and their related memory spaces is given in Table 4.3.

Table 4.3 *Addressing modes and related memory spaces*

Addressing Mode	Memory Space	Operand
Immediate	Program (ROM) memory	Source only
Register	Bank registers (R0 to R7), A, B, CY (Bit)	Source/destination
Direct	Internal RAM, SFRs	Source/destination
Register indirect	Internal/external RAM (@Ri), external RAM @DPTR only)	Source/destination
Index	Program memory (@A +DPTR, @A + PC)	Source only

4.4 EXTERNAL MEMORY DATA MOVEMENTS

The external memory in the 8051 based system may be either data memory (RAM) or program memory (ROM). Instructions that access external memory always use indirect addressing mode. Two different set of instructions are used to transfer data to/from data memory and from program memory.

4.4.1 Data Memory Access

The data memory can be as large as 64 Kbytes. Register R0, R1 and DPTR can be used as a pointer (indirect addressing) to access data bytes from the external RAM. R0 and R1 have limitation to access only address range from 00H to FFH because they are 8-bit register (using 8-bit, we can access 2^8 locations, i.e. 256 bytes – 00H to FFH). DPTR can address any location from 0000H to FFFFH (64 Kbytes, 2^{16} locations). Again to remind, only R0 and R1 can be used with indirect addressing mode.

Mnemonic used for external data memory transfer is MOVX. The letter 'X' in the mnemonic indicates the external data memory. For example, to write data into external RAM, the following instructions are used.

```
MOVX @DPTR, A     // copy contents A into external RAM address contained in DPTR,
                  // If DPTR=1000H and A=10H, → External RAM (1000H)=10H
MOVX @Ri, A       // copy contents A into external RAM address contained in Ri
MOVX @R0, A       // If R0=50H and A=10H, → External RAM (0050H)=10H
```

Similarly, to read data from external RAM, the following instructions are used.

```
MOVX A, @DPTR     // read data from external RAM address pointed by DPTR into A
                  // If DPTR=1000H, (1000H)=20H, → A= 20H
MOVX A, @Ri       // read data from external RAM address pointed by Ri in to A
MOVX A, @R0       // If R0=50H and (50) =10H, → A=10H
```

Here, DPTR or Ri register should be initialized with address of the desired data. Note that all data transfer in this group occurs through Accumulator only, i.e. all external data movements require the A register either as source or destination operand.

Example 4.9

Write instructions to load the value FFH into
(i) Internal RAM addresses 10H to15H
(ii) Port P0 and P1 latch
(iii) External RAM address 1000H

Solution:

(i) One simple way is to load the immediate value directly into the desired addresses as shown below:

```
MOV 10H, #0FFH      // load immediate value FFH into specified addresses directly
MOV 11H, #0FFH
MOV 12H, #0FFH
MOV 13H, #0FFH
MOV 14H, #0FFH
MOV 15H, #0FFH
```

Second way is to load the value into Accumulator and then from A to each address as shown below,

```
MOV A, #0FFH        // First, load immediate value FFH into A and then copy content of A to each address
MOV 10H, A          // Copy contents of A to all addresses from 10H to 15H
MOV 11H, A
MOV 12H, A
MOV 13H, A
MOV 14H, A
MOV 15H, A
```

The advantage of the second method is that it is faster and requires less memory locations to store its machine code. First program requires 12 machine cycles (2 x 6, *two machine cycles for each instruction*) and 18 bytes (3 x 6), while second requires only 7 machine cycles (1 x 7, *only one machine cycle for each instruction*) and 14 bytes (2 x 7).

```
(ii)  MOV P0, #0FFH       // Load value FFH into P0 and P1
      MOV P1, #0FFH
(iii) MOV DPTR, #1000H    // initialize DPTR to point to address 1000H
      MOV A, #0FFH        // load data FFH into A
      MOVX @DPTR, A       // copy contents of A to external RAM address pointed by DPTR (1000H in this example)
```

4.4.2 Program Memory Access

This is the "Read only" type of the data transfer. The data stored in the RAM (either internal or external) are temporary and lost when system is powered down. Sometimes preprogrammed (pre calculated) group of data bytes, i.e. lookup tables, password, strings etc. are needed in some applications. This data must be permanently stored in ROM so that it is always available as and when required. These instructions are also used to read code bytes in some situations.

The instructions to access program memory with examples is already discussed in previous section (see indexed addressing in Section 4.2.4).

Example 4.10

The word 'HI' is burned (written) in the flash ROM address 100H ('H') and 101H ('I'). Copy this word in to internal RAM address 10H and 11H.

Solution:

```
ORG 0000H
MOV DPTR, #100H     // Initialize DPTR to point to ROM address 100H
MOV R0, #10H        // Initialize R0 to point to internal RAM address 10H
CLR A
MOVC A, @A+DPTR     // Read byte from ROM address 100H and place into A
MOV @R0, A          // Copy byte read from ROM to internal RAM address 10H
INC R0              // Increment R0 and DPTR to point to next byte
```

```
INC DPTR
CLR A
MOVC A, @A+DPTR     // Read next byte from ROM address 101H and place into A
MOV @R0, A           // Copy byte to the internal RAM address 11H
ORG 0100H            // Store data in ROM from address 100H onwards
DB 'HI'              // The word to be stored in ROM
```

Note that there are more efficient ways of performing above task, i.e. using loops. The concept of looping is discussed in Chapter 7.

4.5 DATA EXCHANGE

All instructions discussed thus far moves data in only one direction at a time, i.e. from source to destination and an original content of the source operand is not changed.

Exchange instructions as the name suggests, moves data in two directions simultaneously, i.e. from the source to the destination as well as from the destination to the source. All exchanges are only internal to the 8051 and use A register. The examples of exchange instructions are given below.

XCH A, *Rn*	// exchange contents of A and *Rn*
XCH A, R7	// If A= 10H, R7= 30H, → A=30H, R7=10H
XCH A, *direct*	// exchange contents of A and address *direct*
XCH A, 30H	// If A= 10H, (30H)= 20H, → A=20H, (30H)=10H
XCH A, *@Ri*	// exchange contents of A and address in *Ri*
XCH A, @R1	// If A= 10H, R1=30, (30H)= 20H, → A=20H, (30H)=10H
XCHD A, *@Ri*	// exchange lower nibble of A and lower nibble of address in *Ri*
XCHD A, @R1	// If A= 15H, R1=30, (30H)= 20H, → A=10H, (30H)=25H // higher nibbles of both operands are not affected

Example 4.11

Exchange the contents of the PSW and internal RAM address 50H.

Solution:

```
MOV A, PSW    // copy the contents of PSW into A because exchange
              // operation can be done only through A
XCH A, 50H    // exchange A with contents of address 50H
              // now A has contents of address 50H
MOV PSW, A    // move A in to PSW
```

THINK BOX 4.2

Realize the operation performed by XCH A, R2 using MOV instructions.

```
MOV 30H, R2
MOV R2, A
MOV A, 30H
```

4.6 PUSH AND POP INSTRUCTIONS

PUSH and POP are special instructions that are associated with stack operation. Using these instructions, the data is transferred between stack and specified direct address.

The stack is a special memory area in the internal RAM that is used for temporary storage and retrieval of the data, while the execution of a program. It is Last In First Out (LIFO) type memory. The data stored in the stack can be numbers or

address. This section of memory is accessed using PUSH, POP, calls and interrupts. The stack is accessed with the help of stack pointer (SP) register.

SP holds address (of internal RAM) where data from source operand will be saved (pushed) or data to destination address will be retrieved (popped). Stack pointer always points to top of the stack, i.e. last memory address accessed. It should be initialized to a defined value (default value of SP is 07H; refer Topic 7.4 for more details on SP initialization). PUSH instruction saves data on to location pointed by SP, while saving a new data byte, the SP is automatically incremented by 1 and data byte is stored at SP+1 address. POP instruction retrieves data from location pointed by SP, while retrieving, data will be read from address in SP and then SP is decremented by 1. PUSH and POP instructions use SP register indirectly and not specified in an instruction, i.e. these instructions assumes that SP is pointing to top of the stack. These instructions supports only direct addressing mode. The formats of PUSH and POP instructions are given below. Note that a programmer should define the stack at proper place before it can be used; it is done by loading suitable internal RAM address in the SP.

```
PUSH direct     // Increment SP, copy data from address direct to address in SP
POP direct      // copy data from address in SP to address direct, and then decrement SP
```

For example,

```
MOV SP, #50H    // initialize SP to point to address 50H
MOV 35H, #10H   // load data 10H into address 35H
PUSH 35H        // Increment SP to 51H, copy contents of address 35H to address 51H i.e. (51H) = 10H
POP 00H         // copy data from address 51H to address 00H, i.e. (00H) =10H and decrement SP to 50H.
```

Example 4.12

Explain how contents of Accumulator and B registers can be stored and retrieved from the stack.

Solution:

The given task will be accomplished by following set of instructions.

```
MOV SP, #50H    // SP =50 H initialize the stack pointer
MOV A, #10H     // A = 10H
MOV B, #20H     // B = 20H
PUSH ACC        // SP=51H, and (51H) = 10H, SP is incremented by 1 and contents of
                // Accumulator (address E0H) is stored at memory location 51H
PUSH B          // SP = 52, (52H) = 20H, SP is again incremented by 1 and contents of B is stored at memory location 52H
POP B           // B = (52H) = 20H, SP = 51H, data is retrieved into B from address 52H and SP is decremented by 1.
POP ACC         // A= (51H) = 10H, SP = 50H; data is retrieved into A from address 51H and SP is again decremented by 1.
```

Since PUSH and POP instructions support only direct addressing mode, the instruction PUSH ACC will be assembled as PUSH 0E0H and PUSH B instruction as a PUSH 0F0H, i.e. the assembler will replace the names of SFRs with their addresses. The operation of above program is illustrated in Figure 4.6.

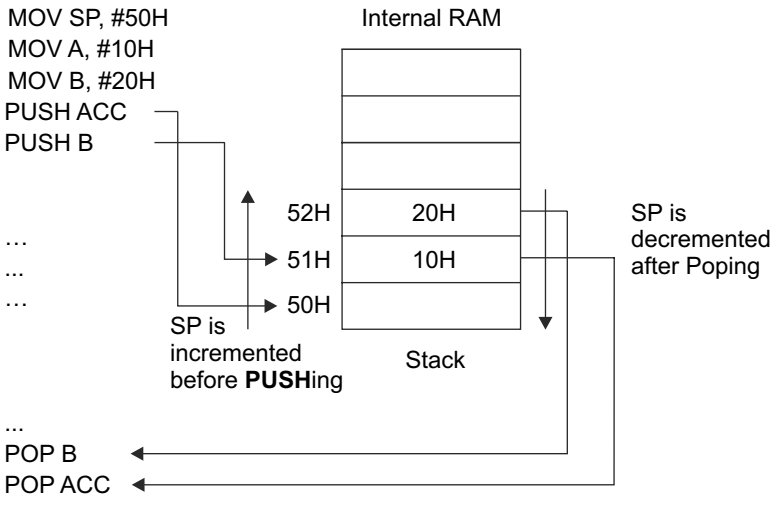

Fig. 4.6 *Stack operation*

Example 4.13

Copy the contents of registers R0 to R7 into internal RAM addresses 40H to 47H respectively using PUSH instructions. Assume bank 0 is selected.

Solution:

Since PUSH instruction automatically increment SP by 1 before saving data on to the stack, we have to initialize SP with 3FH to store data at 40H.

```
MOV SP, #3FH        // Initialize SP with address 3FH
PUSH 00H            // Store R0 (address 00H) at 40H
                    // PUSH will first increment SP to point to 40H and contents of R0 is saved at address40H
PUSH 01H            // Similarly, save R1 to R7 from 41H to 47H
PUSH 02H
PUSH 03H
PUSH 04H
PUSH 05H
PUSH 06H
PUSH 07H
```

Bitwise data transfer instructions are discussed in Chapter 6, and program memory addressing is discussed in detail in Chapter 7.

Example 4.14

State validity of following instructions. Give reason if an instruction is invalid.

(a) MOV R0,#10H	**(b) MOV R0, 10H**	**(c) MOV R2, #256**	**(d) MOV A, @R0**	**(e) MOV A, @R3**
(f) PUSH A	**(g) PUSH 0E0**	**(h) MOV R1, R2**	**(i) MOV 10H, 20H**	**(j) MOV DPTR, A**

Solution:

(a) Valid
(b) Valid
(c) Invalid, R2 is 8-bit register, therefore it can hold maximum value 255 (FFH)
(d) Valid
(e) Invalid, only R0 and R1 can be used for indirect addressing
(f) Invalid, PUSH instructions can be used with only direct addressing.
(g) Valid
(h) Invalid, Register to register data transfer is not allowed
(i) Valid
(j) Invalid, size of both operands should be same

Summary of data movement instructions with examples is given in Table 4.4.

THINK BOX 4.3

Which register of the 8051 can not be accessed using MOV instruction?
PC

POINTS TO REMEMBER

- The microcontrollers typically spend maximum time in data movement operations and usually have maximum instructions for data movement.
- Immediate, register, direct and indirect addressing modes are essentially four addressing modes used by all microcontrollers.

Table 4.4 *Data movement instructions with examples*

Mnemonics	Operation	Addressing Modes			
		Direct	Indirect	Register	Immediate
MOV A, **<src>**	A= **<src>**	MOV A, **direct**	MOV A,@**Ri**	MOV A,**Rn**	MOV A,**#data**
		MOV A, **20H**	MOV A,@**R0**	MOV A,**R3**	MOV A,**#10H**
MOV <dest>, **A**	<dest>= **A**	MOV direct, **A**	MOV @Ri,**A**	MOV Rn,**A**	
		MOV 10H, **A**	MOV @R1,**A**	MOV R2,**A**	
MOV <dest>, **<src>**	<dest>= **<src>**	MOV direct, **direct**	MOV direct,@**Ri**	MOV direct,**Rn**	MOV direct,**#data**
		MOV 05H, **12H**	MOV 10H,@**R0**	MOV 50H,**R5**	MOV 50H,**#10H**
MOV DPTR, **#data 16**	DPTR= 16 bit immediate constant				MOV DPTR,**#data16**
					MOV DPTR,**#1000H**
MOVC A, @**A+DPTR**	Read Program memory at (A+DPTR)		MOVC A,@**A+DPTR**		
			MOVC A,@**A+DPTR**		
MOVC A, @**A+PC**	Read Program memory at (A+PC)		MOVC A,@**A+PC**		
			MOVC A,@**A+PC**		
MOVX A, **<dest>**	Read external RAM from <dest>		MOVX A**,@Ri**		
			MOVX A,@**R0**		
			MOVX A,@**DPTR**		
			MOVX A,@**DPTR**		
MOVX <dest>, **A**	Write external RAM at <dest>		MOVX @Ri, **A**		
			MOVX @R0**, A**		
			MOVX @DPTR**, A**		
			MOVX @DPTR**, A**		
PUSH **<src>**	INC SP: MOV "@SP", <src>	PUSH **direct**			
		PUSH **10H**			
POP **<dest>**	MOV <dest>, "@SP": DEC SP	POP **direct**			
		POP **12H**			
XCH A, **<BYTE>**	ACC & <BYTE> exchange data	XCH A, **direct**	XCH A,@**Ri**	XCH A,**Rn**	
		XCH A, **12H**	XCH A,@**R1**	XCH A,**R0**	
XCHD A, @**Rn**	ACC & @Rn exchange low nibbles		XCHD A,@**Ri**		
			XCHD A,@**R1**		

- The data-movement instruction does not affect the flags and actually copy data from source to destination.
- Immediate addressing is fast but at run-time it is least flexible. Register addressing occupies only one-byte memory, and is fast. Direct addressing is more flexible than immediate and register addressing but requires more bytes. Indirect addressing is the most flexible.
- Indirect addressing is used only for accessing RAM locations 00H to 7FH (128 memory locations) and never for accessing SFRs.
- Indexed addressing is used to access data tables (lookup tables) from program memory and implementing jump tables. They are also suitable for multidimensional array operations.
- Instructions that access external memory always use indirect addressing mode and involves use of A register.
- All exchanges are only internal to the 8051 and use A register. Exchange operation moves data in two directions simultaneously.
- PUSH and POP instructions supports only direct addressing mode.

OBJECTIVE QUESTIONS

1. The instructions that copy data from register R0 to register R5 when bank 0 is active are,
   (a) MOV R5, R0 (b) MOV 05, 00 (c) MOV R5, 00 (d) all of the above
2. The addressing mode for an instruction MOV R0, #30H is,
   (a) register addressing mode (b) direct addressing mode
   (c) immediate addressing mode (d) none of the above
3. Determine the incorrect instruction.
   (a) MOVX A,@R1 (b) MOVC A, @A+DPTR (c) MOV @R0, A (d) MOV @DPTR, A
4. MOV A, @ R1 will,
   (a) copy R1 to the accumulator
   (b) copy the accumulator to R1
   (c) copy the contents of internal RAM pointed by R1 to the accumulator
   (d) copy the contents of external RAM pointed by R1 to the accumulator
5. Which of the following instructions will move the number 27H in the accumulator?
   (a) MOV A, P27 (b) MOV A, #27H (c) MOV A, 27H (d) MOV A, @27
6. Which of the following instructions will move the value of port 3 to the register R2?
   (a) MOV P2, R3 (b) MOV R3, P2 (c) MOV 3P, R2 (d) MOV R2, P3
7. The following instruction will copy the contents of A to the address 50H, MOV 50H, A.
   (a) True (b) False
8. Which of the following instructions will copy the contents of RAM whose address is in register R0 to port 3?
   (a) MOV @ P3, R0 (b) MOV @ R0, P3 (c) MOV P3, @ R0 (d) MOV P3, R0
9. Which of the following is invalid instruction/s?
   (a) MOVX R0, @DPTR (b) MOVC A, @A+PC (c) XCH A, 10H (d) XCH A, #10H
10. MOV 10H, 20H is ___byte instruction.
    (a) 1 (b) 2 (c) 3 (d) 4
11. PUSH/POP instructions support only _____ addressing mode.
    (a) Register (b) Immediate (c) Direct (d) Register indirect
12. MOV *destination byte, source byte* instruction has total ___ formats.
    (a) 10 (b) 12 (c) 15 (d) 18
13. The Immediate operand can be,
    (a) source operand only (b) destination operand only
    (c) either source or destination depending upon instruction (d) source and destination operand for some instructions
14. Which of the following instructions is an example for the direct addressing mode?
    (a) MOVA, @R0 (b) MOV R0, #10H (c) MOV 10H, A (d) MOV R5, A
15. Direct addressing mode is used to access,
    (a) internal data memory (b) external data memory
    (c) internal program memory (d) external program memory

16. The addressing mode used by instruction MOVC A, @A+PC is,
    (a) immediate addressing (b) direct addressing
    (c) indirect addressing (d) indexed addressing
17. External data movements can be done using,
    (a) A only (b) A or B (c) any of the port (d) any of the SFR
18. Indirect addressing is used for accessing internal RAM locations,
    (a) 00H to FFH (b) 00H to 7FH (c) 00H to 1FH (d) 20H to 2FH

Answers to Objective Questions

1. (b), (c)	2. (c)	3. (d)	4. (c)	5. (b)	6. (d)	7. (a)	8. (c)	9. (a), (d)
10. (c)	11. (c)	12. (c)	13. (a)	14. (c)	15. (a)	16. (d)	17. (a)	18. (b)

REVIEW QUESTIONS WITH ANSWERS

1. List the address spaces where operands of an instruction may be present.

A. There are five types of address spaces where operands may be present.
   (i) Bank Registers (R0–R7)
   (ii) Accumulator
   (iii) Internal data RAM (00–7F & SFRs)
   (iv) External data RAM (data memory)
   (v) Program ROM (internal + external ROM)

   Apart from these address spaces, one more place from where data can be obtained is from instruction itself, i.e. in case of immediate addressing mode, data is an integral part of the instruction.

2. What is limitation of immediate addressing?

A. In immediate addressing, the data is fixed; therefore at run-time it is not flexible.

3. Size of the immediate data in the 8051instructions is always 8 bits. True/False.

A. False. We can load 16 bit data in to DPTR register.

4. Can we specify data in decimal or ASCII format?

A. No. Microcontrollers understand only binary number system. However, we can use decimal numbers or ASCII characters if they are supported by an assembler. Finally these numbers will be converted into binary by assembler.

5. Which addressing modes can be used with PUSH and POP instructions?

A. Only direct addressing mode can be used with PUSH and POP instructions.

6. Write instruction/s to load value FFH internal RAM address 50H using direct and indirect addressing modes

A. Direct addressing: MOV 50H,#0FFH
   Indirect addressing: MOV R1, #50H
   MOV @R1, #0FFH

7. State the difference between following instructions.

MOV R0, #10
MOV R0, #10H
MOV R0, 10H

A.
```
MOV R0, #10     // Load decimal value 10 (A Hex) in to R0
MOV R0, #10H    // Load hexadecimal value 10H in to R0
MOV R0, 10H     // Move data at internal RAM address 10H to R0
```

8. Write instruction/s to move contents of R0 to R1.

A.
```
MOV A,R0
MOV R1, A
```

9. Can we write instruction MOV R1, R0 for above operation?

A. No, because register to register move instruction is not supported by the 8051.

10. Where is an indexed addressing mode useful?

A. It is used to access look up tables and to implement jump tables.

11. Which address spaces are accessed in direct addressing mode?

A. Internal RAM and SFRs.

12. What is the limitation in using indirect addressing mode?

A. We can use only R0 and R1 to hold indirect data (address of the data)

13. Where is indirect addressing commonly used?

A. It is used commonly used where similar operation is to be performed on different data, i.e. it is more commonly used to implement pointers in loops.

14. Differentiate between mnemonics MOV, MOVX and MOVC.

A. MOV is used to move data within microcontroller RAM, MOVX is used to transfer data between A and external RAM while MOVC is used to read data from code memory (either internal or external) into A .

EXERCISE

1. What is meant by term "addressing mode"? List addressing modes supported by the 8051 with suitable examples.
2. In which address space is the immediate data stored?
3. State addressing mode used in following instructions.
   (a) MOV R0,#10H (b) ADD A, @R0 (c) MOV 50H,@R1 (d) MOVX @DPTR, A
   (e) MOV A, R5 (f) ANL 50H, #10H (g) MOV 50H, 60H (h) ORL 50H, #50H
   (i) MOVC A, @ A+DPTR (j) MOV DPTR, #2500H
4. Differentiate between data and program addressing modes.
5. How are the following instructions assembled?
   (a) MOV A, #20 (b) MOV A, 'C' (c) MOV R0, #-10H
6. What is the use of operand modifiers # and @?
7. Why is the instruction MOV R1, #256 invalid?
8. Find the number of bytes for following instructions.
   (a) MOV R0, A (b) MOV R0, #10H (c) MOV A, #10H
   (d) MOV 50H, #10H (e) MOV 50H, 10H (f) MOV B, #10H
9. Find value of SP and contents of stack after each of the following instructions.
   MOV SP, #50H
   MOV R0, #10H
   MOV R1, #20H
   PUSH 01H
   PUSH 00H
   MOV R0, #05H
   ADD A, R0
   POP 00H
   POP 01H
10. What is meant by stack overflow?
11. Classify the instructions of the 8051 with respect to their functions.
12. Write instruction/s to move the PC into the DPTR and vice versa.
13. Write two instructions for each of the addressing mode of the 8051.
14. Write instruction/s to save R0 of bank 3 on the stack.
15. List the address spaces accessed by each of the addressing mode in the 8051.
16. Indexed addressing mode is type of indirect addressing. Justify
17. Explain operation the MOVC A, @ A+DPTR instruction.
18. What are the advantages of using indirect addressing mode?
19. Can we access SFRs using indirect addressing mode?
20. How external data memory can be accessed?
21. What is the limitation of exchange instruction?
22. What is the maximum size of the stack in the 8051?
23. Write a program to transfer a byte from code memory address 1000H to internal RAM and external RAM address 10H and 1000H respectively.
24. Write a program to transfer a byte from external RAM address 200H to external RAM address 300H.