8051 ADDRESSING MODES AND INSTRUCTION SET

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the instruction syntax and data types of the 8051
- Define a subroutine and Explain its uses
- Explain the addressing modes of the 8051
- Explain the instruction timings of the 8051
- Explain the instruction set of the 8051

3.1 INSTRUCTION SYNTAX

In assembly language, all operations and addresses can be identified by symbols. This frees the programmer from memorising or looking up the machine code for instructions and keeping track of the addresses of the entire data and instructions. 8051 instructions are divided into four fields as follows.

Scanned by TapScanner

LABEL: OPCODE OPERAND; COMMENT

Label The label is the symbolic address for the instruction. As the program is assembled, the label will be given the value of the address in which the instruction is stored. This facilitates referencing of the instruction at any point in the given program. Of course, not all instructions will have labels. It is not necessary to define a symbol for the address of an instruction, unless that address is needed by a branch statement elsewhere in the program. For instance in Example 3.1, only one instruction is referred by a branch statement. A label can be any combination of upto 8 letters (A–Z), numbers (0–9) and period (.).

EXAMPLE 3.1

8051 program

```
MOV R5,# 05H    ; Load counter R5 = 05H
MOV A, 40H      ; Copy data from RAM location 40H to register A
MOV R0, #30H    ; Copy immediate data 30H to register R0
LOOP: ADD A, R0 ; Add contents of register R0 with contents of
                  register A
DJNZ R5, LOOP   ; Repeat addition until R5 is zero
END
```

Opcode The opcode field contains a symbolic representation of the operation. The operation tells the assembler what action the statement has to perform. The 8051 assembler converts the opcode into a unique machine language (binary code) that can be acted on by the 8051 internal circuitry. For instance, in Example 3.1, the mnemonics MOV and ADD are the opcodes. MOV will copy data from one location to register, or immediate data to register. ADD will add contents of one register to another register.

Operand The opcode specifies what action to perform, whereas the operand indicates where to perform the action. The operand field contains the address of the operand or the operand. For instance in Example 3.1, MOV A, 40H; the operand field contains two addresses. MOV will copy data from one location (source) to another location (destination). In MOV R0, #30H; the operand field contains operand 30H and it is moved to register R0.

Comment To improve program clarity, a programmer uses comments throughout the program. A comment always begins with a semicolon (;) and wherever we code it, the assembler assumes that all characters to its right are comments. A comment may contain any printable character, including a blank. We can insert a comment on a line all by itself or following an instruction on the same line. A comment appears only on a listing of an assembled program and generates no machine code.

SECTION REVIEW

1. Write the format of 8051 instructions.
2. _____special character is used to begin comments.

Scanned by TapScanner

3.2 DATA TYPES

The 8051 microcontroller supports only 8 bit data. It supports both signed and unsigned 8 bit numbers. Unsigned numbers are defined as data in which all the bits are used to represent the data. For 8 bit, the data can be 00 to FFH or 00 to 255 in decimal. In signed number representation, the most significant bit identifies the number as positive or negative, and the remaining bits are used to represent magnitude. If the most significant bit is zero, the number is positive, or if it is one, then the number is negative. In negative numbers, the magnitude is represented in two's complement form as shown in Table 3.1. For 8 bit, the data can be from +127 to –128.

TABLE 3.1 *8051 data types*

Decimal	Hex	Binary
– 128	80	10000000
– 127	81	10000001
– 1	FF	11111111
0	00	00000000
+ 1	01	00000001
+ 127	7F	01111111

3.3 SUBROUTINES

Good program design is based on the concept of modularity—the partitioning of a large program into subroutines. A subroutine is a sequence of instructions stored in the memory at a specified address for performing repeatedly needed tasks. Subroutines are usually handled by special instructions, CALL and RET. The CALL instruction is of the form CALL address. The subroutine processing is as shown in Fig. 3.1.

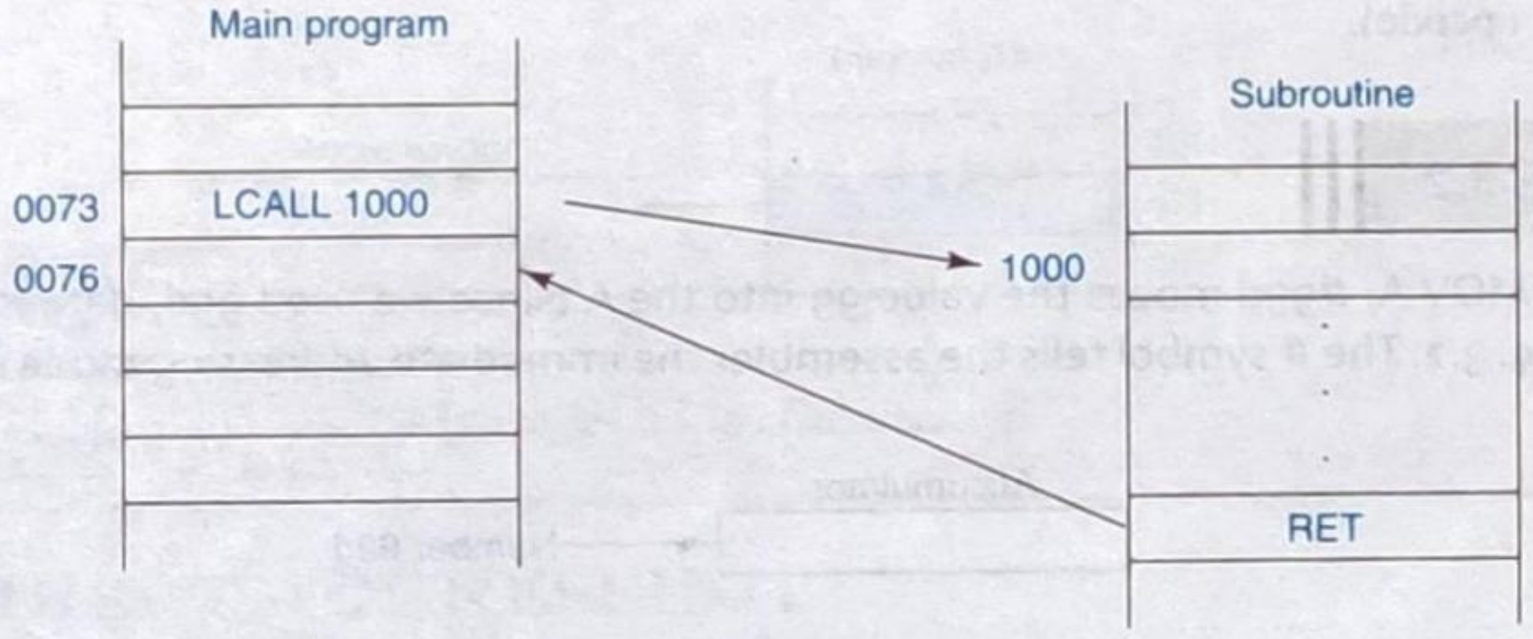

Figure 3.1 *Subroutine processing*

Scanned by TapScanner

The address refers to the address of the subroutine. When CALL instruction is executed, the contents of the program counter are saved in the stack and the program counter is loaded with the address, which is a part of CALL instruction. The RET instruction is usually the last instruction in the subroutine. When this instruction is executed, the return address previously saved in the stack is retrieved and is loaded into the program counter. Thus, the control is transferred to the calling program.

SECTION REVIEW

1. List types of data supported by the 8051.
2. In signed numbers, _____ bit represents the sign.
3. In signed numbers, the magnitude of the negative numbers is represented in _____.
4. Name the instructions used in a subroutine.
5. When CALL instruction is executed, then the contents of the program counter are saved in _____.

3.4 ADDRESSING MODES

A microcontroller provides, for the convenience of the programmer, various methods for accessing data needed in the execution of an instruction. The various methods of accessing data are called *addressing modes*. The 8051 addressing modes can be classified into the following categories

- Immediate addressing
- Register addressing
- Direct addressing
- Indirect addressing
- Relative addressing
- Absolute addressing
- Long addressing
- Indexed addressing
- Bit inherent addressing
- Bit direct addressing

3.4.1 IMMEDIATE ADDRESSING

Immediate addressing means that the data is provided as part of the instruction (which immediately follows the instruction opcode).

EXAMPLE 3.2

Instruction MOV A, #99d moves the value 99 into the A (since we used 99d, data is in decimal) as shown in Fig. 3.2. The # symbol tells the assembler the immediate addressing mode is to be used.

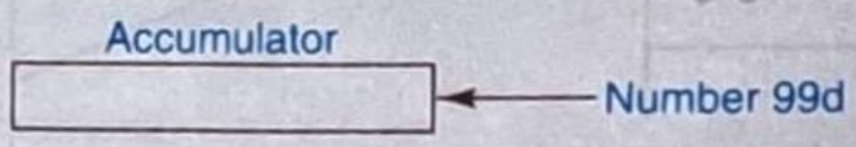

Figure 3.2 *Immediate addressing*

Scanned by TapScanner

3.4.2 REGISTER ADDRESSING

Register addressing mode involves the use of registers to hold the data to be manipulated. The lowest 32 bytes of the 8051 internal RAM are organised as four banks of eight registers. Only one bank is active at a time. Using names, R0 to R7 can access any active register. One of the eight general registers (R0 to R7) can be specified as the instruction operand. The assembly language documentation refers to a register generically as Rn.

EXAMPLE 3.3

Instruction MOV A, R5 copies contents of register R5 to A.

Here, the content of R5 is copied to the A as shown in Fig. 3.3. The advantage of register addressing is that the instruction tends to be short and is a single–byte instruction.

Figure 3.3 *Register addressing*

3.4.3 DIRECT ADDRESSING

Direct addressing mode is provided to allow us access to internal data memory, including Special Function Register (SFR). In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

EXAMPLE 3.4

Consider the instruction: MOV A, 47H.

The instruction reads the data from internal RAM address 47H and copies this into the A as shown in Fig. 3.4.

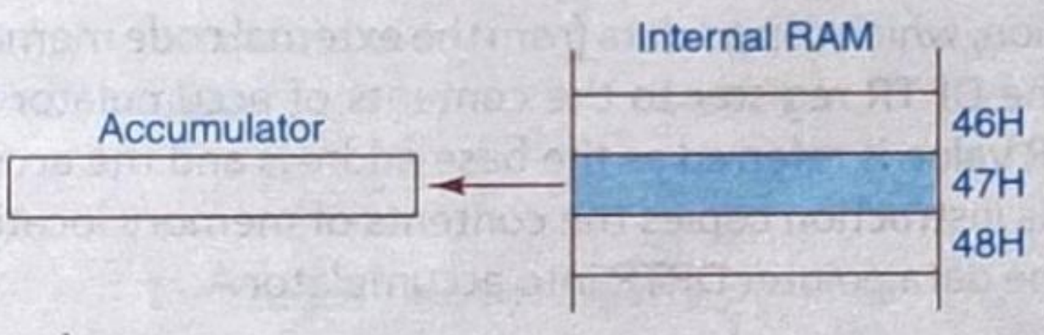

Figure 3.4 *Direct addressing*

3.4.4 INDIRECT ADDRESSING

Indirect addressing provides a powerful addressing capability, which needs to be appreciated. The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0,

Scanned by TapScanner

R1, and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

EXAMPLE 3.5

The instruction which uses indirect addressing, is MOV A, @Ro.

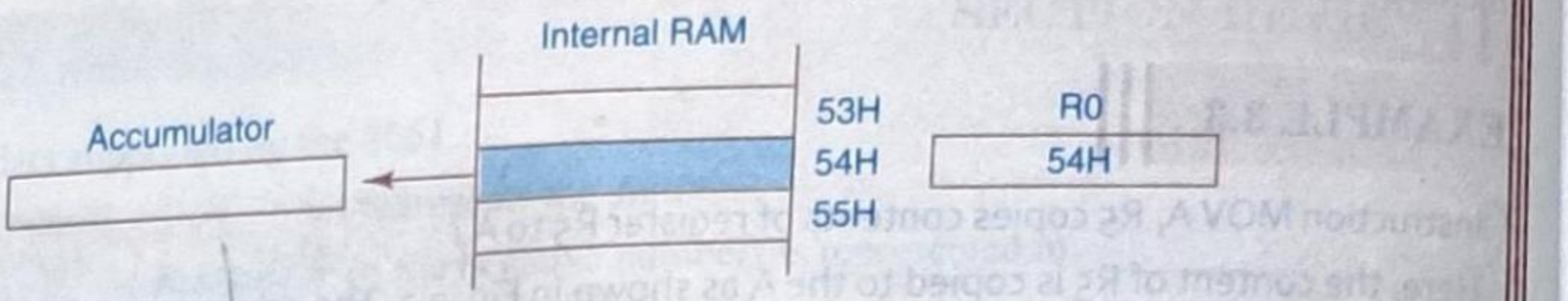

Figure 3.5 *Indirect addressing*

The @ symbol in the instruction indicates indirect addressing mode. Ro contains the address of the internal RAM location (54H). It copies the contents of memory location pointed by Ro into accumulator A.

MOVX A, @DPTR

It copies the contents of external data memory location pointed by DPTR into accumulator A.

3.4.5 INDEXED ADDRESSING

In indexed addressing, a separate register—either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

EXAMPLE 3.6

Consider the instruction: MOVC A, @A+DPTR

MOVC is a move instruction, which copies data from the external code memory space. In this example, adding the content of the DPTR register to the contents of accumulator forms the address of the operand. Here, the DPTR value is referred as the base address and the accumulator value is referred as the index address. This instruction copies the contents of memory location pointed by the sum of the accumulator A and the data pointer DPTR into accumulator A.

MOVC A, @A+PC

This instruction copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.

The above–discussed addressing modes are illustrated as shown in Fig. 3.6

Scanned by TapScanner

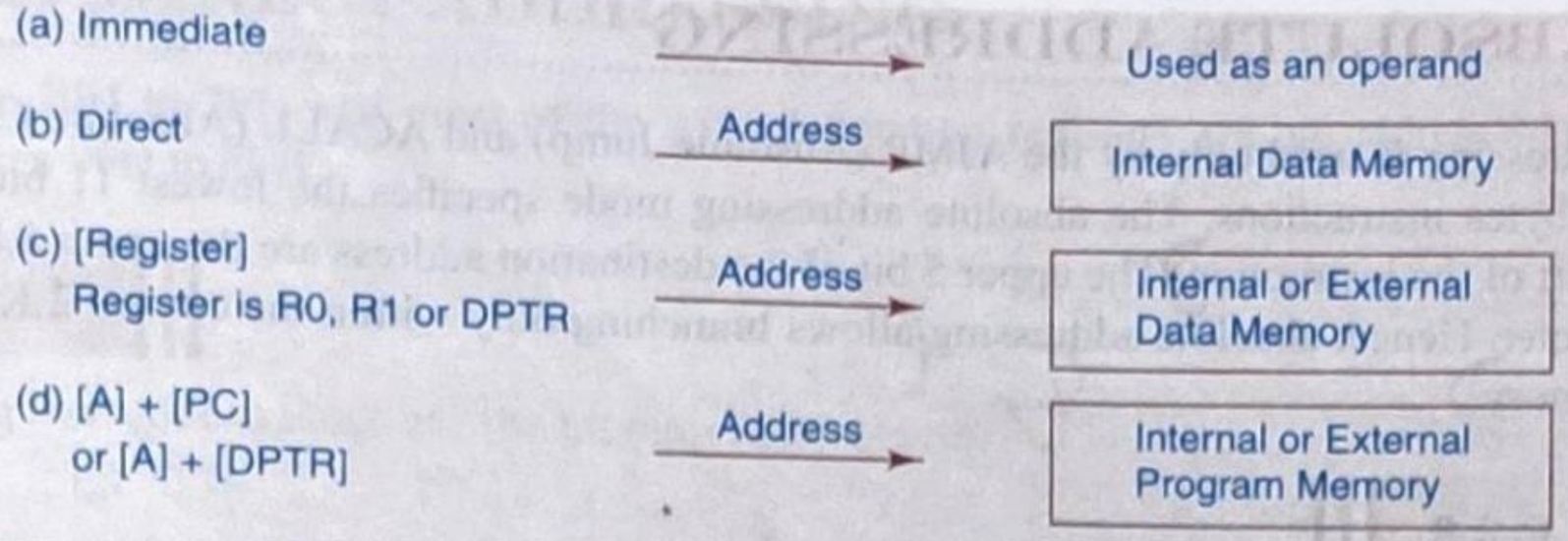

Figure 3.6 *Addressing modes (a) Immediate (b) Direct (c) Indirect (d) Indexed*

3.4.6 RELATIVE ADDRESSING

Relative addressing is used only with conditional jump instructions. The relative address, often referred to as an offset, is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to –128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly.

EXAMPLE 3.7

Consider the example SJMP X

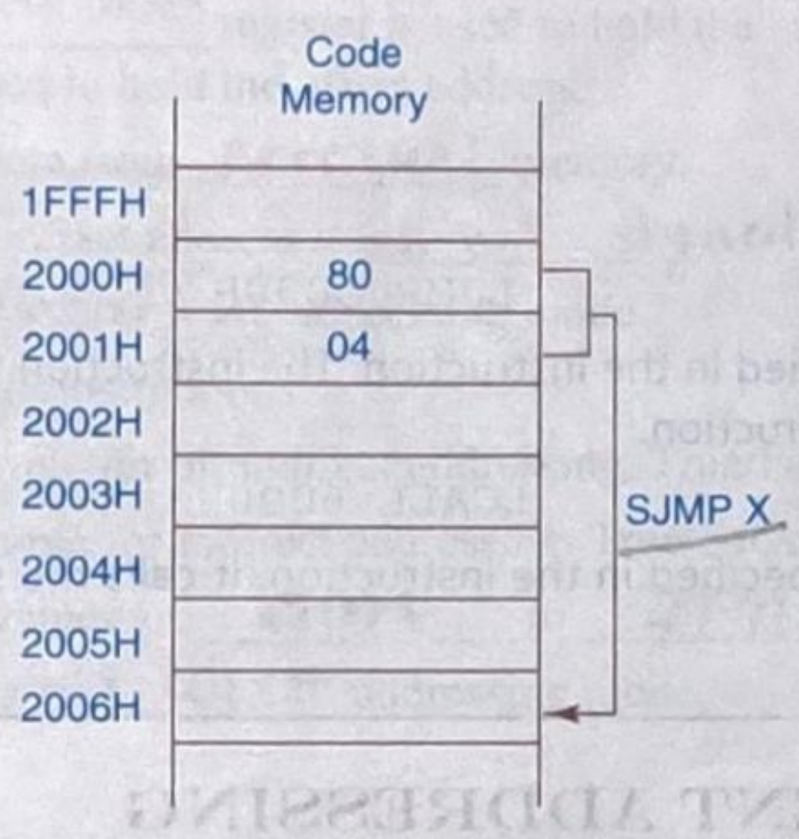

Figure 3.7 *Relative addressing mode*

PC is set to next instruction address: 2002H, when SJMP begins execution. The target address is then the sum of the PC + relative offset needed to reach X. If X is 4, then PC is set to 2002H + 4H = 2006H as shown in Fig. 3.7. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

Scanned by TapScanner

3.4.7 ABSOLUTE ADDRESSING

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction. The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

EXAMPLE 3.8

Consider the instruction

```
AJMP loop1
```

The instruction with the label loop1 will be executed after the current instruction.

```
ACALL loop1
```

Calls the subroutine that is started at the label loop1

3.4.8 LONG ADDRESSING

The long addressing mode within the 8051 is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space ($2^{16} = 64K$).

EXAMPLE 3.9

Consider the instruction

```
LJMP 5000H
```

16 bit branch address is specified in the instruction. The instruction with the address 5000H will be executed after the current instruction.

```
LCALL 6000H
```

16 bit subroutine address is specified in the instruction. It calls the subroutine, which starts at the address 6000H.

3.4.9 BIT INHERENT ADDRESSING

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

EXAMPLE 3.10

The following instruction illustrates bit inherent addressing.

```
CLR C      ; Clears the carry flag to 0
```

Scanned by TapScanner

3.4.10 BIT DIRECT ADDRESSING

The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

EXAMPLE 3.11

The following instructions illustrate the bit direct addressing.

```
CLR 07H    ; Clears the bit 7 of 20H RAM space
SETB 07H   ; Sets the bit 7 of 20H RAM space.
```

SECTION REVIEW

1. List addressing modes of the 8051 microcontroller.
2. Immediate addressing is identified by ___#___ symbol in the assembler.
3. MOV A, R4 is an example for ___Register___ addressing mode.
4. List the types of addressing modes in memory operations.
5. Direct addressing mode is used to access ___Internal RAM___ memory.
6. List the registers used as data pointers in indirect addressing.
7. Name the instructions used to access internal data memory using indirect addressing mode.
8. In indexed addressing, ___PC or DPTR___ register is used to hold the base address and ___A___ register is used to hold the offset address.
9. MOV C instruction moves data from ___external___ memory.
10. In relative addressing mode, offset address is ___8 bit signed no.___
11. SJMP 25 is an example of ___Relative___ addressing mode.
12. Differentiate SJMP and AJMP instructions.
13. CLRC instruction is an example for bit indirect addressing. True/False? F
14. MOVXA@DPTR is an example for indirect addressing. True/False? T
15. In bit direct addressing, bit values are ___00H___ to ___7FH___.
16. SETB 07H is an example for ___bit direct___ addressing mode.

3.5 INSTRUCTION TIMINGS

The 8051 internal operations and external read/write operations are controlled by the oscillator clock input signal. T-state, Machine cycle and Instruction cycle are terms used in instruction timings. Let us define these terms.

T-state T-state is defined as one subdivision of the operation performed in one clock period. The terms 'T-state' and 'clock period' are often used synonymously.

Scanned by TapScanner

Machine cycle Machine cycle in 8051 is defined as 12 oscillator periods. According to the Intel literature, a machine cycle consists of six states and each state lasts for two oscillator periods. The 8051, take one to four machine cycles to execute an instruction.

Instruction cycle Instruction cycle is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

EXAMPLE 3.12

If 8051 microcontroller is operated with 12 MHz oscillator, find the execution time for the following four instructions.

(a) ADD A, 45H
(b) SUBB A, #55H
(c) MOV DPTR, #2000H
(d) MUL AB

Since the oscillator frequency is 12 MHz, the clock period is

Clock period = 1/12 MHz = 0.08333 µs.

Time for 1 machine cycle = 0.08333 µs × 12 = 1 µs.

Execution timings are tabulated in Table 3.2

TABLE 3.2 *Instruction timing*

Instruction	Execution time in machine cycle	Execution time in µsec
ADD A, 45H	1 machine cycle	1 µs
SUBB A, #55H	2 machine cycles	2 µs
MOV DPTR, #2000H	2 machine cycles	2 µs
MUL AB	4 machine cycles	4 µs

SECTION REVIEW

1. If the 8051 operates with 6 MHz crystal, then the execution time for ADD A, # 45H instruction is _____.
2. If the 8051 operates with 3MHz oscillator, then the clock period is _____.

3.6 8051 INSTRUCTIONS

8051 instructions consist of 1 byte of opcode and 0 to 2 bytes of operands. The instructions use 8 bit register A, B, R0, R1, R2, R3, R4, R5, R6, R7 and also 16 bit registers, DPTR (data pointer) and PC (program counter). The instructions of 8051 can be broadly classified under the following headings:

Scanned by TapScanner

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

The following notations are used in the description of the instruction.

Rn means, any of the eight registers (R0 to R7) in the selected banks.
Ri means either of the pointing registers (R0 or R1) in the selected banks.
M means internal data memory.
E.D.M. means external data memory.
P.M. means internal or external program memory.

3.6.1 DATA TRANSFER INSTRUCTIONS

In this group, the instructions perform data transfer operations of the following types.

1. Move the contents of a register A to Rn
2. Move the contents of a register Rn to A
3. Move an immediate 8 bit data to register A or to Rn or to a memory location
4. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing
5. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
6. Move the contents of memory location to another memory location using direct and indirect addressing
7. Move the contents of an external memory to A or A to an external memory
8. Move the contents of program memory to A
9. Push and Pop instructions
10. Exchange instructions

Note

In this group of instructions, none of the flags of status register are affected.

Move the contents of a register Rn to A: MOV A, Rn This instruction copies the contents of register Rn to A. It is a 1 byte instruction.

EXAMPLE 3.13

```
MOV A, R2          [A] ← [R2]
              Before execution  After execution
[A]                    25              45
[R2]                   45              45
```

Move the contents of a register A to Rn: MOV Rn, A This instruction copies the contents of A to Rn. It is a 1 byte instruction.

Scanned by TapScanner

EXAMPLE 3.14

```
                    MOV R3, A          [R3] ← [A]
                Before execution After execution
                      25                25
[A]                   45                25
[R3]
```

Move an immediate 8 bit data to register A or to Rn or to a memory location. This type of instruction loads the immediate data to A or to Rn or to the memory by using direct and indirect addressing.

```
MOV A, #data
```

EXAMPLE 3.15

```
                    MOV A, #23         [A] ← 23
                Before execution After execution
                      F5                23
[A]
```

```
MOV Rn, #data
```

EXAMPLE 3.16

```
                    MOV R2, #23        [R2] ← 23
                Before execution After execution
                      F3                23
[R2]
```

```
MOV direct, #data
```

EXAMPLE 3.17

```
                    MOV 30, #23        M [30] ← 23
                Before execution After execution
                      F3                23
M [30]
```

```
MOV @Ri, #data
```

EXAMPLE 3.18

```
                    MOV @R1, #23       M [[R1]] ← 23
                Before execution After execution
                      34                34
[R1]
 M [34]               F3                23
```

```
MOV DPTR, #16bitdata
```

Scanned by TapScanner

EXAMPLE 3.19

```
                MOV DPTR, #2300   [DPTR] ← 2300
              Before execution After execution
[DPTR]            3422              2300
[DPL]               22                00
[DPH]               34                23
```

Move the contents of a memory location to A or A to a memory location using direct and indirect addressing

Direct addressing:

```
MOV A, direct
```

EXAMPLE 3.20

```
                MOV A, 40        A ← M [40]
              Before execution After execution
[A]                 23                55
M [40]              55                55
```

```
MOV direct, A
```

EXAMPLE 3.21

```
                MOV 45,A         M [45] ← [A]
              Before execution After execution
[A]                 23                23
M [45]              55                23
```

Indirect addressing:

```
MOV A, @Ri
```

EXAMPLE 3.22

```
                MOV A,@R1      [A] ← M [[R1]]
              Before execution After execution
[A]                 23                55
[R1]                42                42
M[42]               55                55
```

```
MOV @Ri, A
```

Scanned by TapScanner

EXAMPLE 3.23

```
                    MOV @R1, A      M [[R1]] ← [A]
                Before execution After execution
                      23                23
[A]                   40                40
[R1]                  55                23
M[40]
```

Move the contents of a memory location to Rn or Rn to a memory location using direct addressing

```
MOV Rn, direct
MOV direct, Rn
```

EXAMPLE 3.24

```
                    MOV R2, 50      R2 ← M [50]
                Before execution After execution
                      23                55
[R2]                  55                55
M [50]
```

EXAMPLE 3.25

```
                    MOV 60,R3       M [60] ← R3
                Before execution After execution
                      23                23
[R3]                  55                23
M [60]
```

Move the contents of a memory location to another memory location using direct and indirect addressing

```
MOV direct1, direct2
```

EXAMPLE 3.26

```
                    MOV 55,67     M [55] ← M [67]
                Before execution After execution
M [55]                23                55
M [67]                55                55
```

```
MOV direct, @Ri
```

Scanned by TapScanner

EXAMPLE 3.27

```
                    MOV 57, @R0   M [57] ← M [[R0]]
                Before execution After execution
[R0]                    30              30
M [30]                  33              33
M [57]                  55              33
```

Move the contents of an external data memory location pointed by Ri or DPTR to A or A to an external data memory pointed by Ri or DPTR.

```
MOVX A, @Ri
```

EXAMPLE 3.28

```
                    MOVX A, @R1       A ←E.D.M[[R1]]
                Before execution      After execution
[A]                     23                  55
[R1]                    10                  10
E.D.M. [10]             55                  55
```

```
MOVX @Ri,A
```

EXAMPLE 3.29

```
                    MOVX @R1, A     E.D. M [[R1]] ← [A]
                Before execution     After execution
[A]                     23                  23
[R1]                    20                  20
E.D.M [20]              55                  23
```

```
MOVX A, @DPTR
```

EXAMPLE 3.30

```
                    MOVX A, @DPTR     A ← E.D.M[[DPTR]]
                Before execution      After execution
[A]                     23                  55
[DPTR]                1000                1000
E.D.M. [1000]           55                  55
```

```
MOVX @DPTR, A
```

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner

Scanned by TapScanner