# Microprocessors & Microcontrollers (ECE3004)

**Module – 03 (Part – 02)**

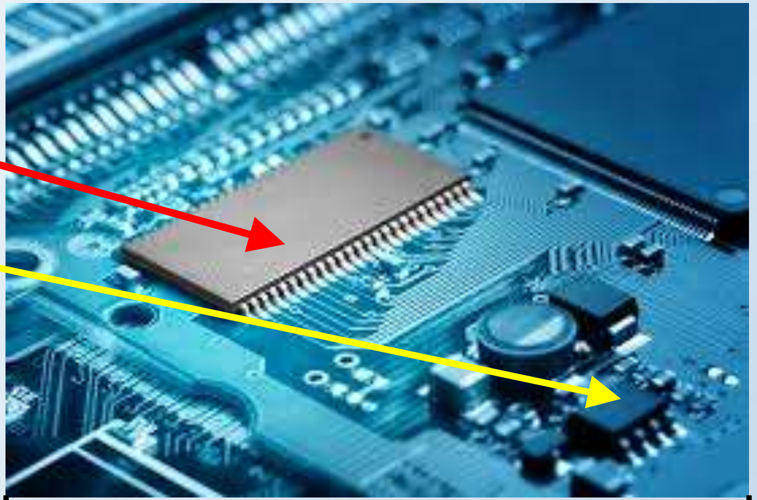
# 8051 Microcontroller

**Dr. Susant Kumar Panigrahi**
**Assistant Professor**
**School of Electrical & Electronics Engineering**

# Learning Methodology

❖ Hardware Information

- Internal architecture of the processor (8085, 8086, 8051 etc.)
- Internal architecture of peripheral interface controller (viz. 8255, 8251, 8253, 8237, 8259 etc.)
- Circuit connections of the processor & peripheral devices (like connection among components in a PCB)



❖ Software Instructions

- Complex Instruction Set Computer (CISC)
- Reduced Instruction Set Computer  (RISC)

# Module-3 Syllabus

**8051 Microcontroller:**

- Intel MCS-51family features – 8051 -organization and architecture, addressing modes, Instruction set, conditional instructions, I/O Programming, Arithmetic logic instructions, single bit instructions, interrupt handling, programming counters, timers and Stack.

# Unit-III (Part-02): Microcontroller Programming

# 8051 Instruction Set



The structure of the 8051 Microcontroller Instruction: An 8051 Instruction consists of an

1. Opcode (short of Operation – Code)
2. Operand(s) of size Zero Byte, One Byte or Two Bytes.

The **Op-Code** part of the instruction contains the Mnemonic, which specifies the type of operation to be performed. All Mnemonics or the Opcode part of the instruction are of One Byte size.

Operand defines the data being processed by the instructions.

- ✓ No Operand
- ✓ Data value
- ✓ I/O Port
- ✓ Memory Location
- ✓ CPU register

**Instruction Format**

MNEMONIC DESTINATION OPERAND, SOURCE OPERAND

**NOTE**: Instruction can be:

1. One-byte instruction, which contains only opcode
2. Two-byte instructions, where the second byte is the operand
3. Three byte instructions, where the operand makes up the second and third byte.

# 8051 Instruction Set



- Based on the operation they perform, all the instructions in the 8051 Microcontroller Instruction Set are divided into five groups:

  i. Data Transfer Instructions

  ii. Arithmetic Instructions

  iii. Logical Instructions

  iv. Boolean or Bit Manipulation Instructions

  v. Program Branching Instructions

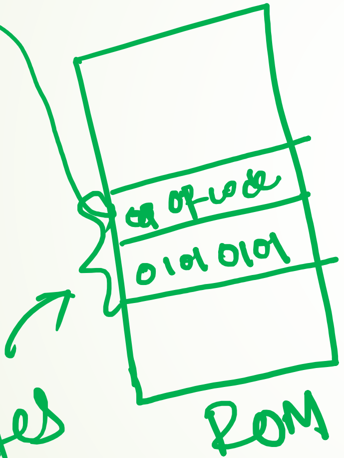
# i. Data Transfer Instructions

The Data Transfer Instructions are associated with transfer with data between registers or external program memory or external data memory.

i. MOV
ii. MOVC
iii. MOVX
iv. PUSH
v. POP
vi. XCH
vii. XCHD

Mov A, #55H

op-code
8-Bits




2-Bytes

| Mnemonic | Instruction | Description | Addressing Mode | # of Bytes | # of Cycles |
|---|---|---|---|---|---|
| MOV | A, #Data | A ← Data | Immediate | 2 | 1 |
| | A, Rn | A ← Rn | Register | 1 | 1 |
| | A, Direct | A ← (Direct) | Direct | 2 | 1 |
| | A, @Ri | A ← @Ri | Indirect | 1 | 1 |
| | Rn, #Data | Rn ← data | Immediate | 2 | 1 |
| | Rn, A | Rn ← A | Register | 1 | 1 |
| | Rn, Direct | Rn ← (Direct) | Direct | 2 | 2 |
| | Direct, A | (Direct) ← A | Direct | 2 | 1 |
| | Direct, Rn | (Direct) ← Rn | Direct | 2 | 2 |
| | Direct1, Direct2 | (Direct1) ← (Direct2) | Direct | 3 | 2 |
| | Direct, @Ri | (Direct) ← @Ri | Indirect | 2 | 2 |
| | Direct, #Data | (Direct) ← #Data | Direct | 3 | 2 |
| | @Ri, A | @Ri ← A | Indirect | 1 | 1 |
| | @Ri, Direct | @Ri ← Direct | Indirect | 2 | 2 |
| | @Ri, #Data | @Ri ← #Data | Indirect | 2 | 1 |
| | DPTR, #Data16 | DPTR ← #Data16 | Immediate | 3 | 2 |
| MOVC | A, @A+DPTR | A ← Code Pointed by A+DPTR | Indexed | 1 | 2 |
| | A, @A+PC | A ← Code Pointed by A+PC | Indexed | 1 | 2 |
| | A, @Ri | A ← Code Pointed by Ri (8-bit Address) | Indirect | 1 | 2 |
| MOVX | A, @DPTR | A ← External Data Pointed by DPTR | Indirect | 1 | 2 |
| | @Ri, A | @Ri ← A (External Data 8-bit Addr) | Indirect | 1 | 2 |
| | @DPTR, A | @DPTR ← A (External Data 16-bit Addr) | Indirect | 1 | 2 |
| PUSH | Direct | Stack Pointer SP ← (Direct) | Direct | 2 | 2 |
| POP | Direct | (Direct) ← Stack Pointer SP | Direct | 2 | 2 |
| XCH | Rn | Exchange ACC with Rn | Register | 1 | 1 |
| | Direct | Exchange ACC with Direct Byte | Direct | 2 | 1 |
| | @Ri | Exchange ACC with Indirect RAM | Indirect | 1 | 1 |
| XCHD | A, @Ri | Exchange ACC with Lower Order Indirect RAM | Indirect | 1 | 1 |



**Example**

**Operation:** XCHD
**Function:** Exchange Digit
**Syntax:** XCHD A, @R0/@R1

| Instructions | OpCode | Bytes | Cycles | Flags |
|---|---|---|---|---|
| XCHD A,@R0 | 0xD6 | 1 | 1 | None |
| XCHD A,@R1 | 0xD7 | 1 | 1 | None |

# ii. Arithmetic Instructions

These instructions are used to perform various mathematical operations like addition, subtraction, multiplication, and division etc.

i. ADD
ii. ADDC
iii. SUBB
iv. INC
v. DEC
vi. MUL
vii. DIV
viii. DA A

a. The arithmetic instructions has no knowledge about the data format i.e. signed, unsigned, ASCII, BCD, etc.
b. The operations performed by the arithmetic instructions affect flags like carry, overflow, zero, etc. in the PSW Register.

op code

op code

| Mnemonic | Instruction | Description | Addressing Mode | # of Bytes | # of Cycles |
|---|---|---|---|---|---|
| ADD | A, #Data | A ← A + Data | Immediate | 2 | 1 |
| | A, Rn | A ← A + Rn | Register | 1 | 1 |
| | A, Direct | A ← A + (Direct) | Direct | 2 | 1 |
| | A, @Ri | A ← A + @Ri | Indirect | 1 | 1 |
| | | | | | |
| ADDC | A, #Data | A ← A + Data + C | Immediate | 2 | 1 |
| | A, Rn | A ← A + Rn + C | Register | 1 | 1 |
| | A, Direct | A ← A + (Direct) + C | Direct | 2 | 1 |
| | A, @Ri | A ← A + @Ri + C | Indirect | 1 | 1 |
| | | | | | |
| SUBB | A, #Data | A ← A – Data – C | Immediate | 2 | 1 |
| | A, Rn | A ← A – Rn – C | Register | 1 | 1 |
| | A, Direct | A ← A – (Direct) – C | Direct | 2 | 1 |
| | A, @Ri | A ← A – @Ri – C | Indirect | 1 | 1 |
| | | | | | |
| MUL | AB | Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B) | -- | 1 | 4 |
| | | | | | |
| DIV | AB | Divide A by B (A ← Quotient and B ← Remainder) | -- | 1 | 4 |
| | | | | | |
| DEC | A | A ← A – 1 | Register | 1 | 1 |
| | Rn | Rn ← Rn – 1 | Register | 1 | 1 |
| | Direct | (Direct) ← (Direct) – 1 | Direct | 2 | 1 |
| | @Ri | @Ri ← @Ri – 1 | Indirect | 1 | 1 |
| | | | | | |
| INC | A | A ← A + 1 | Register | 1 | 1 |
| | Rn | Rn ← Rn + 1 | Register | 1 | 1 |
| | Direct | (Direct) ← (Direct) + 1 | Direct | 2 | 1 |
| | @Ri | @Ri ← @Ri + 1 | Indirect | 1 | 1 |
| | DPTR | DPTR ← DPTR + 1 | Register | 1 | 2 |
| | | | | | |
| DA | A | Decimal Adjust Accumulator | -- | 1 | 1 |

# iii. Logical Instructions

The logical instructions are the instructions which are used for performing some operations like AND, OR, NOT, X-OR and etc., on the operands.

i. ANL
ii. ORL
iii. XRL
iv. CLR
v. CPL
vi. RL
vii. RLC
viii. RR
ix. RRC
x. SWAP

| Mnemonic | Instruction | Description | Addressing Mode | # of Bytes | # of Cycles |
|---|---|---|---|---|---|
| ANL | A, #Data | A ← A AND Data | Immediate | 2 | 1 |
| | A, Rn | A ← A AND Rn | Register | 1 | 1 |
| | A, Direct | A ← A AND (Direct) | Direct | 2 | 1 |
| | A, @Ri | A ← A AND @Ri | Indirect | 1 | 1 |
| | Direct, A | (Direct) ← (Direct) AND A | Direct | 2 | 1 |
| | Direct, #Data | (Direct) ← (Direct) AND #Data | Direct | 3 | 2 |
| | | | | | |
| ORL | A, #Data | A ← A OR Data | Immediate | 2 | 1 |
| | A, Rn | A ← A OR Rn | Register | 1 | 1 |
| | A, Direct | A ← A OR (Direct) | Direct | 2 | 1 |
| | A, @Ri | A ← A OR @Ri | Indirect | 1 | 1 |
| | Direct, A | (Direct) ← (Direct) OR A | Direct | 2 | 1 |
| | Direct, #Data | (Direct) ← (Direct) OR #Data | Direct | 3 | 2 |
| | | | | | |
| XRL | A, #Data | A ← A XRL Data | Immediate | 2 | 1 |
| | A, Rn | A ← A XRL Rn | Register | 1 | 1 |
| | A, Direct | A ← A XRL (Direct) | Direct | 2 | 1 |
| | A, @Ri | A ← A XRL @Ri | Indirect | 1 | 1 |
| | Direct, A | (Direct) ← (Direct) XRL A | Direct | 2 | 1 |
| | Direct, #Data | (Direct) ← (Direct) XRL #Data | Direct | 3 | 2 |
| | | | | | |
| CLR | A | A← 00H | -- | 1 | 1 |
| | | | | | |
| CPL | A | A ← A | -- | 1 | 1 |
| | | | | | |
| RL | A | Rotate ACC Left | -- | 1 | 1 |
| | | | | | |
| RLC | A | Rotate ACC Left through Carry | -- | 1 | 1 |
| | | | | | |
| RR | A | Rotate ACC Right | -- | 1 | 1 |
| | | | | | |
| RRC | A | Rotate ACC Right through Carry | -- | 1 | 1 |
| | | | | | |
| SWAP | A | Swap Nibbles within ACC | -- | 1 | 1 |

# iv. Bit Manipulation Instructions

As the name suggests, Boolean or Bit Manipulation Instructions will deal with bit variables. We know that there is a special bit-addressable area in the RAM and some of the Special Function Registers (SFRs) are also bit addressable.

i. CLR
ii. SETB
iii. MOV
iv. JC
v. JNC
vi. JB
vii. JNB
viii. JBC
ix. ANL
x. ORL
xi. CPL

| Mnemonic | Instruction | Description | # of Bytes | # of Cycles |
|---|---|---|---|---|
| CLR | C | C ← 0 (C = Carry Bit) | 1 | 1 |
| | Bit | Bit ← 0 (Bit = Direct Bit) | 2 | 1 |
| | | | | |
| SET | C | C ← 1 | 1 | 1 |
| | Bit | Bit ← 1 | 2 | 1 |
| | | | | |
| CPL | C | C ← $\overline{\text{C}}$ | 1 | 1 |
| | Bit | Bit ← $\overline{\text{Bit}}$ | 2 | 1 |
| | | | | |
| ANL | C, /Bit | C ← C. $\overline{\text{Bit}}$ (AND) | 2 | 1 |
| | C, Bit | C ← C . Bit (AND) | 2 | 1 |
| | | | | |
| ORL | C, /Bit | C ← C + $\overline{\text{Bit}}$ (OR) | 2 | 1 |
| | C, Bit | C ← C + Bit (OR) | 2 | 1 |
| | | | | |
| MOV | C, Bit | C ← Bit | 2 | 1 |
| | Bit, C | Bit ← C | 2 | 2 |
| | | | ELECTRONICS HUB | |
| JC | rel | Jump is Carry (C) is Set | 2 | 2 |
| JNC | rel | Jump is Carry (C) is Not Set | 2 | 2 |
| JB | Bit, rel | Jump is Direct Bit is Set | 3 | 2 |
| JNB | Bit, rel | Jump is Direct Bit is Not Set | 3 | 2 |
| JBC | Bit, rel | Jump is Direct Bit is Set and Clear Bit | 3 | 2 |

# v. Branch and Looping Instructions



These instructions control the flow of program logic.

i. LJMP
ii. AJMP
iii. SJMP
iv. JZ
v. JNZ
vi. CJNE
vii. DJNZ
viii. NOP
ix. LCALL
x. ACALL
xi. RET
xii. RETI
xiii. JMP

| Mnemonic | Instruction | Description | # of Bytes | # of Cycles |
|---|---|---|---|---|
| ACALL | ADDR11 | Absolute Subroutine Call<br>PC + 2 → (SP); ADDR11 → PC | 2 | 2 |
| LCALL | ADDR16 | Long Subroutine Call<br>PC + 3 → (SP); ADDR16 → PC | 3 | 2 |
| RET | -- | Return from Subroutine<br>(SP) → PC | 1 | 2 |
| RETI | -- | Return from Interrupt | 1 | 2 |
| AJMP | ADDR11 | Absolute Jump<br>ADDR11 → PC | 2 | 2 |
| LJMP | ADDR16 | Long Jump<br>ADDR16 → PC | 3 | 2 |
| SJMP | rel | Short Jump<br>PC + 2 + rel → PC | 2 | 2 |
| JMP | @A + DPTR | A + DPTR → PC | 1 | 2 |
| JZ | rel | If A=0, Jump to PC + rel | 2 | 2 |
| JNZ | rel | If A ≠ 0, Jump to PC + rel | | |
| | | | | |
| CJNE | A, Direct, rel | Compare (Direct) with A. Jump to PC + rel if not equal | 3 | 2 |
| | A, #Data, rel | Compare #Data with A. Jump to PC + rel if not equal | 3 | 2 |
| | Rn, #Data, rel | Compare #Data with Rn. Jump to PC + rel if not equal | 3 | 2 |
| | @Ri, #Data, rel | Compare #Data with @Ri. Jump to PC + rel if not equal | 3 | 2 |
| | | | | |
| DJNZ | Rn, rel | Decrement Rn. Jump to PC + rel if not zero | 2 | 2 |
| | Direct, rel | Decrement (Direct). Jump to PC + rel if not zero | 3 | 2 |
| | | | | |
| NOP | | No Operation | 1 | 1 |



a. All these instructions, except the NOP (No Operation) affect the Program Counter (PC) in one way or other.

[Label]: Nemonics Destination Operand Source operand [; Comment]

Instructions

opcode (Machine Code)

Compulsory

operand

(User Defined)

programmer defined values

# Microcontroller Programming

4-Tstates = 1 M/C

4 clock period/cycle = 1 M/C

8051 µp

HERE : MOV R1, A ; comment . . .

Label

Instruction

Destination operand

Source register

Opcode → machine code (Hex value)

ROM

HERE: MOV A, #055H

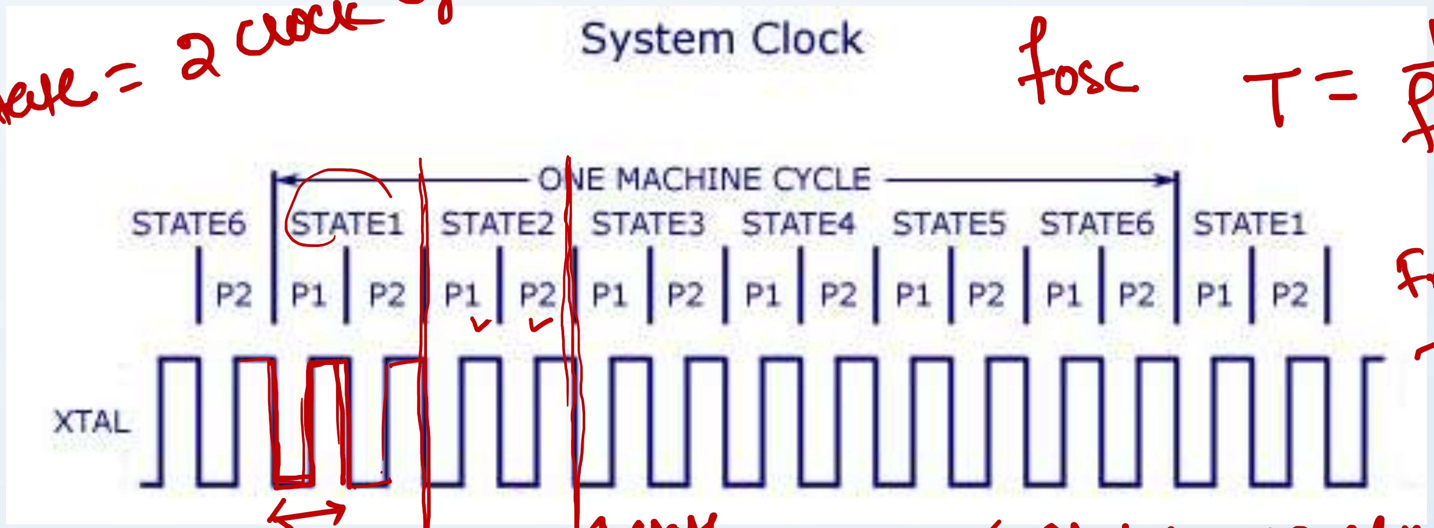
opcode

operand

Hex

# Clock Cycle, Machine Cycle and Instruction Cycle

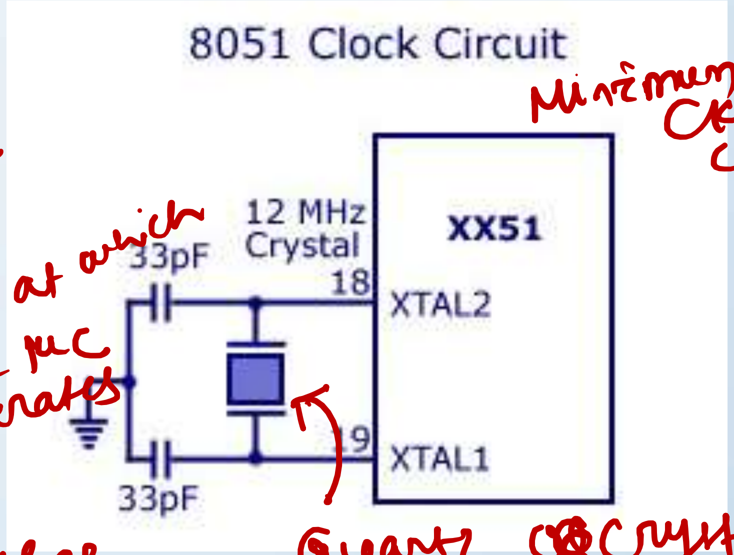
1 State = 2 clock cycles




fosc

$T = \frac{1}{f_{osc}}$

Freq at which the μC operates




Minimum Clk connection

6 States = 12 clock cycles = 1 Machine cycle

Quartz Crystal

$f_{osc} = 11.059\ MHz$

- The CPU takes a certain number of clock cycles to execute an instruction. These clock cycles are referred to as machine cycles.

fosc = clock freq.

$T = \frac{1}{f_{osc}}$

12-clock cycles to make one MC

freq. for 1 MIC $= \frac{1}{T_{MIC}} = f_{osc}/12$

✓ $12\ Clock\ Cycles = 1\ Machine\ Cycle.$

✓ $1\ Clock\ Cycle(period) = \frac{1}{f_{osc.}}\ Secs.$

✓ $12\ Clock\ Cycles = 1\ Machine\ Cycle = \frac{12}{f_{osc,}}\ Secs.$ ‘OR’ $Machine\ Cycle\ Freq. = \frac{f_{osc}}{12}\ Hz.$

- 1 Machine cycle = 1 Byte of Instruction.

$\boxed{\text{Time for 1 MIC} = 1\ MC = 12 \times \frac{1}{f_{osc}}}$

# Clock Cycle, Machine Cycle and Instruction Cycle

**Example # 1:**

- Lets find the time period of the machine cycle in each case for the following crystal frequency of different 8051 based systems: $\underbrace{11.0592\text{ MHz}}_{f_{osc1}}$, $\underbrace{16\text{ MHz}}_{f_{osc2}}$, $\underbrace{20\text{ MHz}}_{f_{osc3}}$

**Answer**:

✓ 11.0592 MHz:

11.0592/12 = 921.6 KHz

Machine cycle = 1/921.6 KHz = 1.085us [us=microsecond]

✓ 16 MHz:

16MHz/12 = 1.333 MHz

Machine cycle = 1/1.333 MHz = 0.75us [us=microsecond]

✓ 20MHz:

20MHz/12 = 1.66 MHz

Machine Cycle = 1/1.66 MHz = 0.60us [us=microsecond]

$f_{osc1} = 11.059\text{MHz}$

$T = \frac{1}{f_{osc1}} = \frac{1}{11.059\times10^{6}}$ secs

clock freq. $= 0.0904\ \mu$secs.

1 Machine cycle $= 12 \times T$

$= 12 \times 0.0904$

$= 1.085\ \mu$sec

$f_{osc} = 16\text{MHz}$

$T = \frac{1}{f_{osc}} = \frac{1}{16\times10^{6}} = 0.0625\ \mu$secs.

1 M/C $= 12 \times 0.0625$

$= 0.75\ \mu$sec

1 M/c T.P. $= 0.60\ \mu$sec

# Clock Cycle, Machine Cycle and Instruction Cycle

$f_{osc} = 11.059$ MHZ

$T = \frac{1}{11.059 \times 10^6}$

**Example # 2**: "8051"

- Lets find how long it takes to execute each of the following instructions, for a crystal frequency of 11.0592 MHz. The machine cycle of a system of 11.0592 MHz is 1.085 us.

$= 0.0904$ us

$1 M/C = 12 \times T$

$= 1.085$ us

| INSTRUCTION | MACHINE CYCLE | TIME TO EXECUTE |
|---|---|---|
| MOV R2,#55H Immediate A.M | 1 | 1x1.085 us = 1.085 us |
| DEC R2 | 1 | 1x1.085 us = 1.085 us |
| DJNZ R2,target AGAIN | 2 | 2x1.085 us = 2.17 us |
| LJMP | 2 | 2x1.085 us = 2.17 us |
| SJMP 2 M/C. | 2 | 2x1.085 us = 2.17 us |
| NOP | 1 | 1x1.085 us = 1.085 us |
| MUL AB 4 machine cycle | 4 | 4x1.085 us = 4.34 us |

$\Sigma$ ?

# Generate Delay

# Square wave Generation (S/W)


```
DELAY: MOV R3, #0FFH     ← 1 M/C
AGAIN: DJNZ R3, AGAIN    ← 2 M/C
       RET
END
```

Time Delay

Loop

DJNZ: Decrement Jump not equal

$T_{ON} = T_{off} = 5\,msecs$

R3 = 255

R3 ← 255-1

R3 ← 254-1

ACALL DELAY

for

$1\,M/C = \frac{12}{f_{osc}} = \frac{12}{11.059\,MHz} = 1.085\,\mu s$

R3 = FFH = 255

R3 ← R3-1

R3 ← 254

org 0000H

ACALL DELAY ← function

END

PC ←

PC+1

first line — 1 M/C

2nd line — 255 X2 M/C

$\overline{[(255 \times 2) + 1]}$ M/C

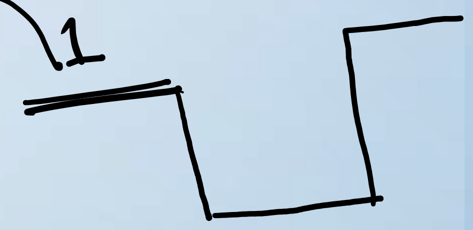
for i = 1 : 100

{

Loop (100 times)

}

Time Delay = $[(255 \times 2) + 1] \times 1.085$ μsecs




Square wave

```
HERE:   SETB  P1.0
        ACALL DELAY
        CLR   P1.0   ← ACALL DELAY
        SJMP  HERE
```

# Square Wave Generation using 8051

$(1 + (255 \times 2) + 2)$ M/C

$f_{osc} = 11.059$ MHz

1 M/C = 1.085 μsec

$\frac{T_{ON}}{T_{ON} + T_{OFF}}$

## Subroutine

```
DELAY:  MOV R1, #0FFH        ; 1 M/C
AGAIN:  DJNZ R1, HERE        ; 2 M/C
        RET                  ; 2 M/C
```

255 M/C

R1 ← 255

R1 ← R1 − 1

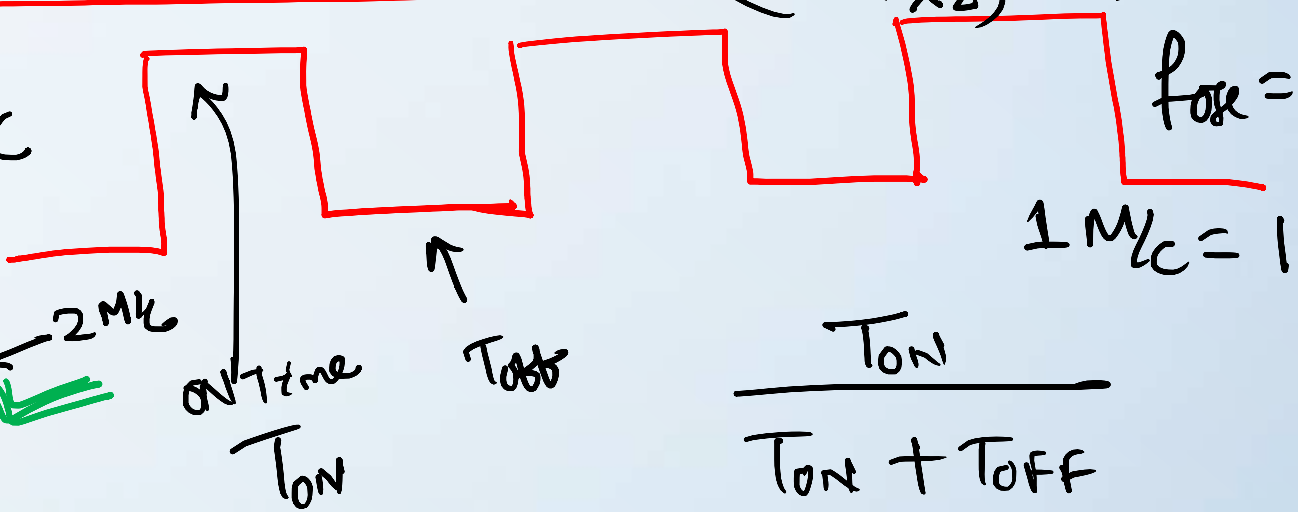
R1 ← 254

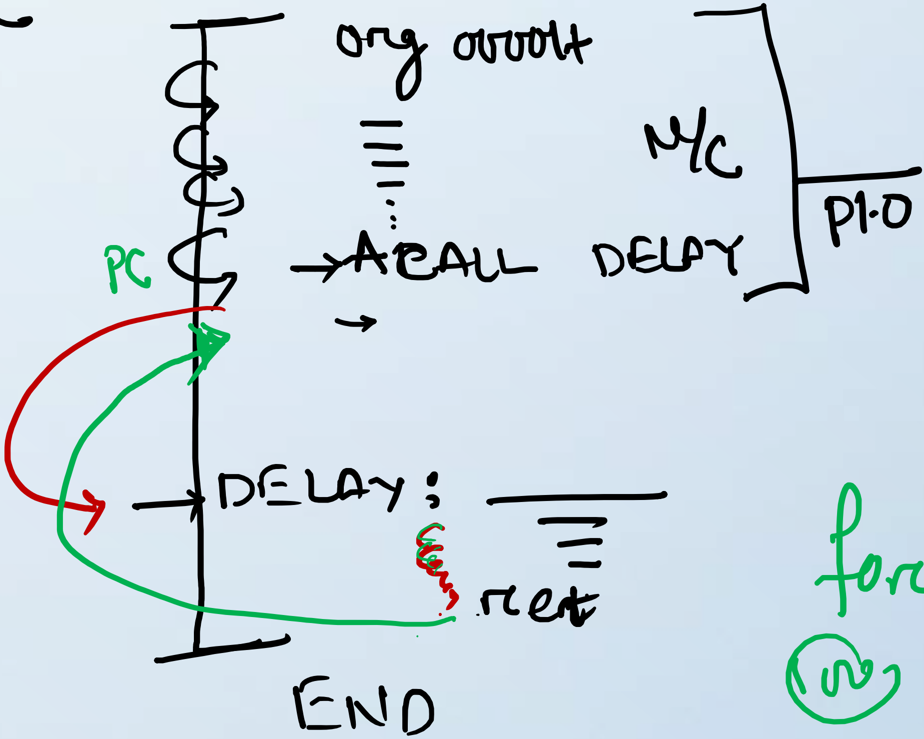
R1 ← R1 − 1

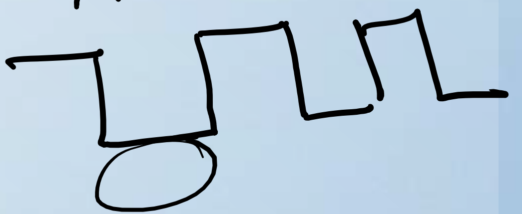
R1 ← 253

#Times: 255

T = 556 μs



for i = 1 : 100
{
end
i = i + 1

Ex:-

Calculate Total time delay of $f_{osc} = 11.059\,MHz$

```
DELAY:  MOV   R4, #05H     + 1 M/C
L3:     MOV   R5, #0FFH
L2:     MOV   R6, #0FFH
L1:     DJNZ  R6, L1   ; 255 X 2 M/C
        DJNZ  R5, L2   ; 255 x 255 x 2
        DJNZ  R4, L3   ; (255 X 255 X 05 x 2) M/C
        RET            + 2 M/C
```

$\left[1\,M/C + 2\,M/C + (255 \times 255 \times 05 \times 2)\right] M/C$

$\times 1.085$

$\approx$ 785 msecs

$\approx$ 785, 286 μsecs $\approx$ 0.78 sec.

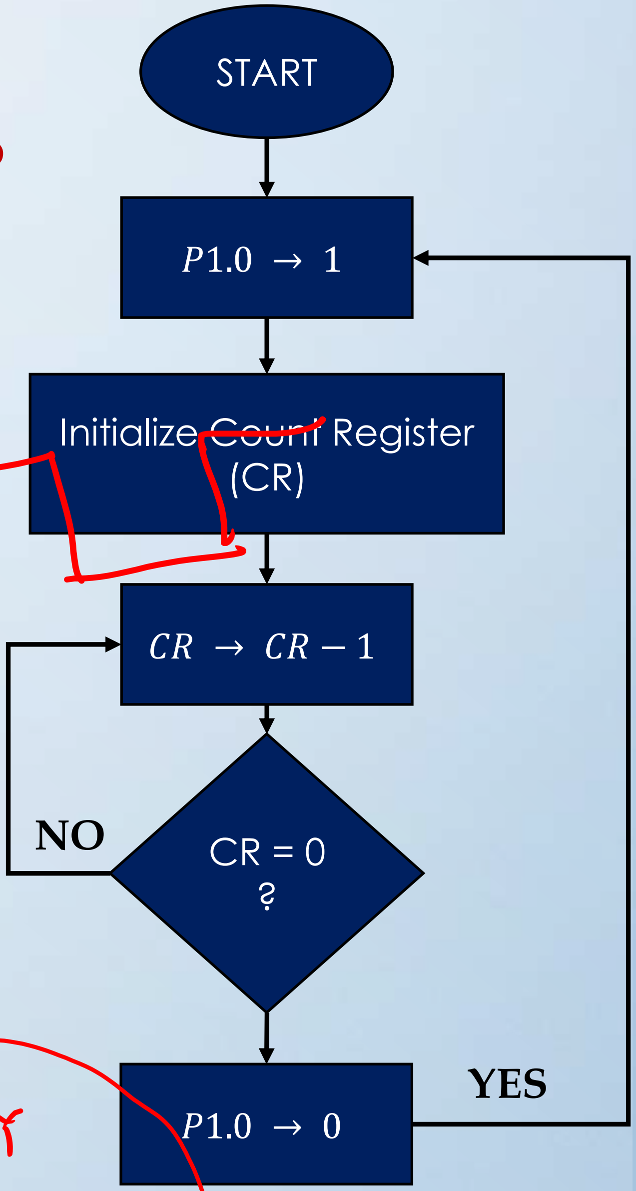
10 x 100 x 50

```
for i = 1:10
  for j = 1:100
    for k = 1:50
    end
  end
end
```

# Square Wave Generation & LED Interfacing

- Write a program to generate square wave at port pin P1.0. Also calculate the time period of the generated square wave.

```
; -----------------------------------------------------------------
; Programe to generate Square Wave.
; Output can be seen in Port P1.0.
; Author: Dr. Susant Kumar Panigrahi
; Dt: 30.12.2020 (Version 1.0)
; -----------------------------------------------------------------
        ORG 0000H           ; Initialize the program counter (PC)
        LJMP START          ; Jump to the level 'START'

        ORG 0050H           ; Start the program from this location
START:  SETB    P1.0        ; Set the Port P1.0
        ACALL   DELAY       ; Call Delay Subroutine (On Time)
        CLR     P1.0        ; Clear the port pin P1.0
        ACALL   DELAY       ; Call Delay Subroutine (Off Time)
        LJMP    START       ; Repeate the process.

DELAY:  MOV R4, #05H        ; Delay Subroutine
    L3: MOV R5, #0FFH       ; Three registers are used to generate delay
    L2: MOV R6, #0FFH

    L1: DJNZ R6, L1
        DJNZ R5, L2
        DJNZ R4, L3
        RET
    END
```




P1.0
LED "glow"

Oscillscope

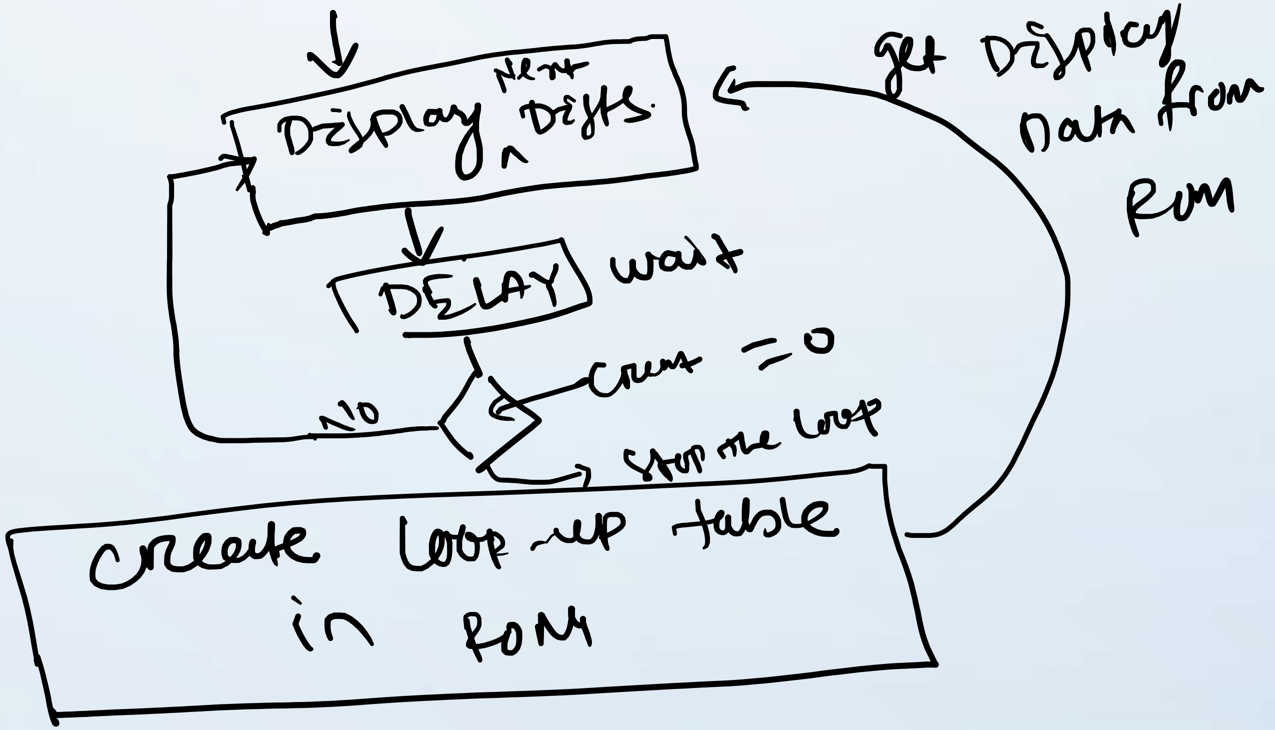
DELAY

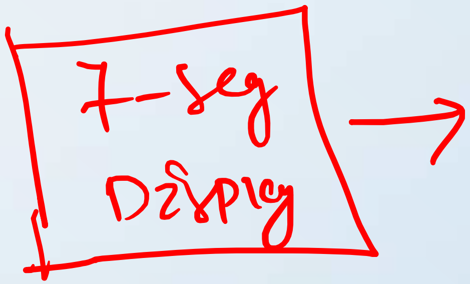
SETB P1.0
ACALL DELAY
CLR P1.0
ACALL DELAY

# 7-Seg Display interfacing using Delay Look-up Table

**Prog. Algorithm**

Counter — R1 ← 10



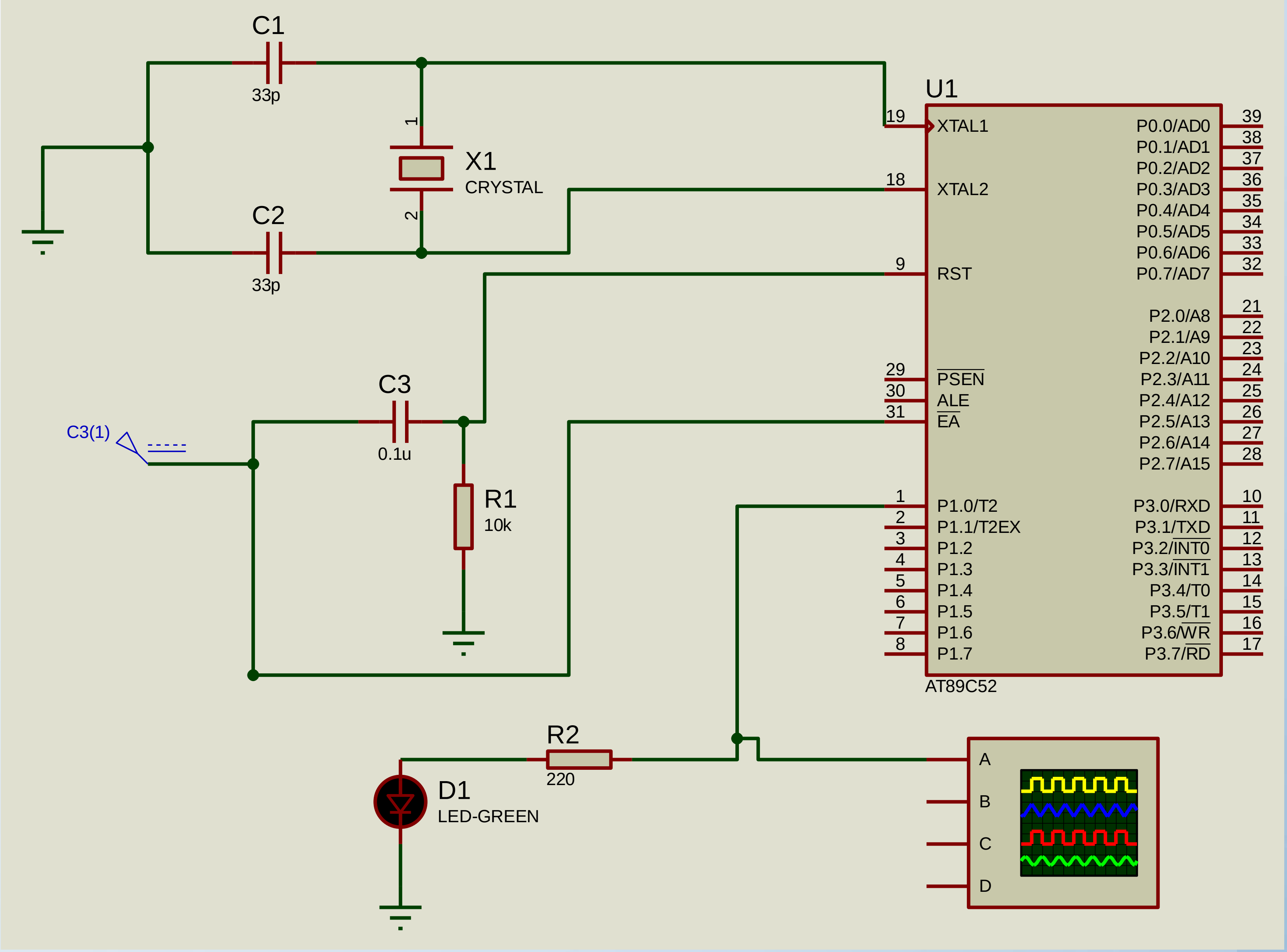




0 → 3FH
1 — 06H
2 — 5BH
⋮
9 — 6FH

Burn the value in Loop-up Table into ROM.

# Square Wave Generation & LED Interfacing

- Write a program to generate square wave at port pin P1.0. Also calculate the time period of the generated square wave.




Calculate the Delay ?

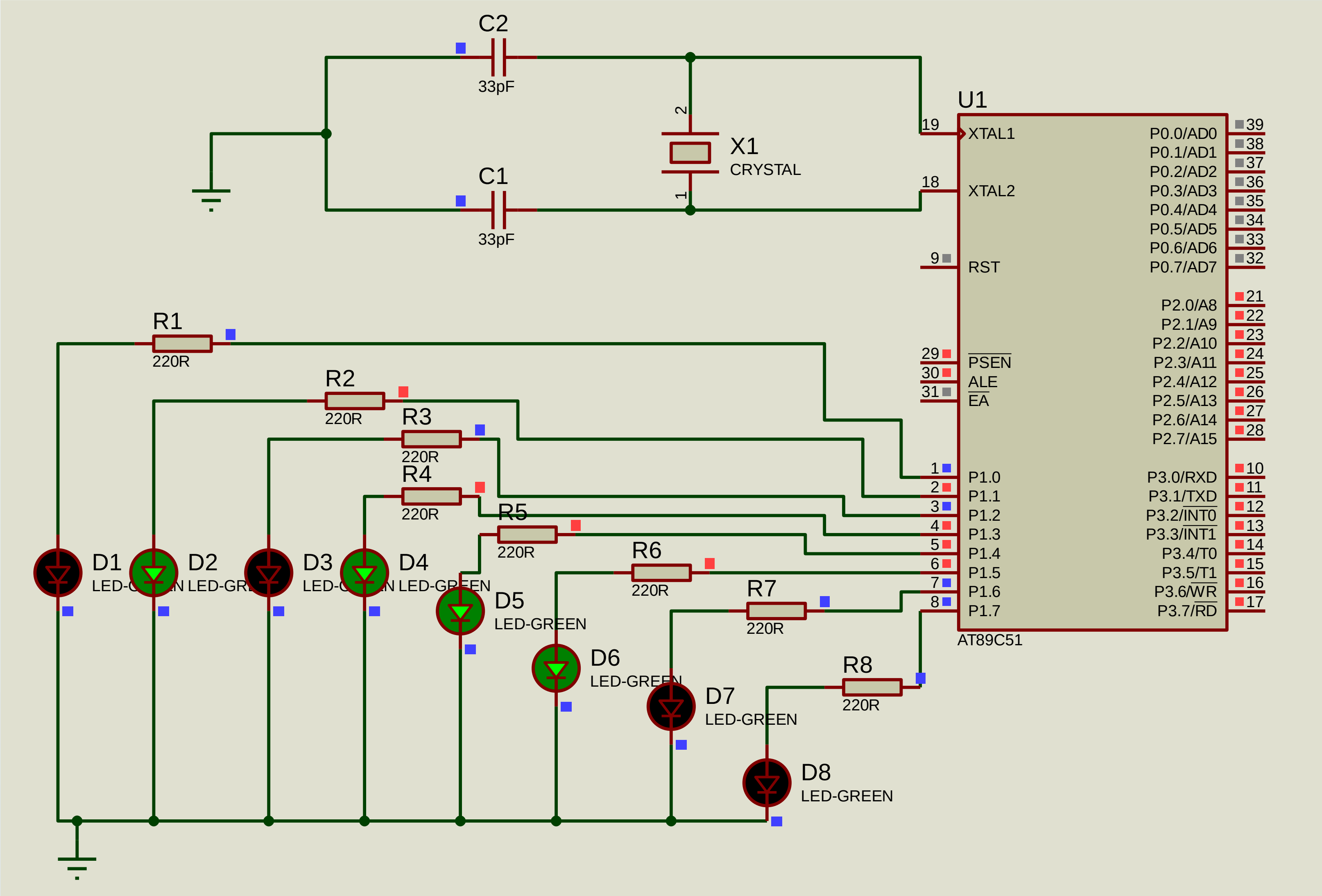
# Add Two Numbers stored in Memory

- Write a program to add two numbers stored in memory location 40H and 41H and store the resultant sum in 50H. Also display the result in Port P1. The carry need to be stored in memory location 51H.

```
  // Program to ADD Two numbers... Display SUM in Port P1
                ORG 0000H
                LJMP START

                ORG 0050H
START:          MOV     P1, #00H        ; Initialize Port P1 as output port
                SETB    PSW.4           ; Select Register BANK 03
                SETB    PSW.3
                MOV     R0,  #1AH       ; Store Values to be added in regsiters
                MOV     R1,  #20H
                MOV     40H, R0         ; Store 1AH in memory location 40H
                MOV     41H, R1         ; Store data in memory location 41H

                ; Addition Operation
                MOV     R2, #00H        ; Indicate Carry bit
                MOV     A,  40H         ; Get the value stored in 40H
                ADD     A,  41H         ; Add two number in 40h and 41H and store the result in A
                MOV     50H, A          ; Resultant Sum store in memory location 50H
                MOV     P1,  A          ; Display Resultant Sum in Port P1
                ; Check if carry bit is set or not ----
                JNC     SKIP            ; Jump if carry bit is not set to SKIP
                INC     R2
SKIP:           MOV     51H, R2         ; Store the carry in 51H
HERE:           SJMP    HERE            ; Stay HERE: Inifinity LOOP
                END         ; HLT The programe HERE
```

# Add Two Numbers stored in Memory (Ckt. Diagram)

- Write a program to add two numbers stored in memory location 40H and 41H and store the resultant sum in 50H. Also display the result in Port P1. The carry need to be stored in memory location 51H.

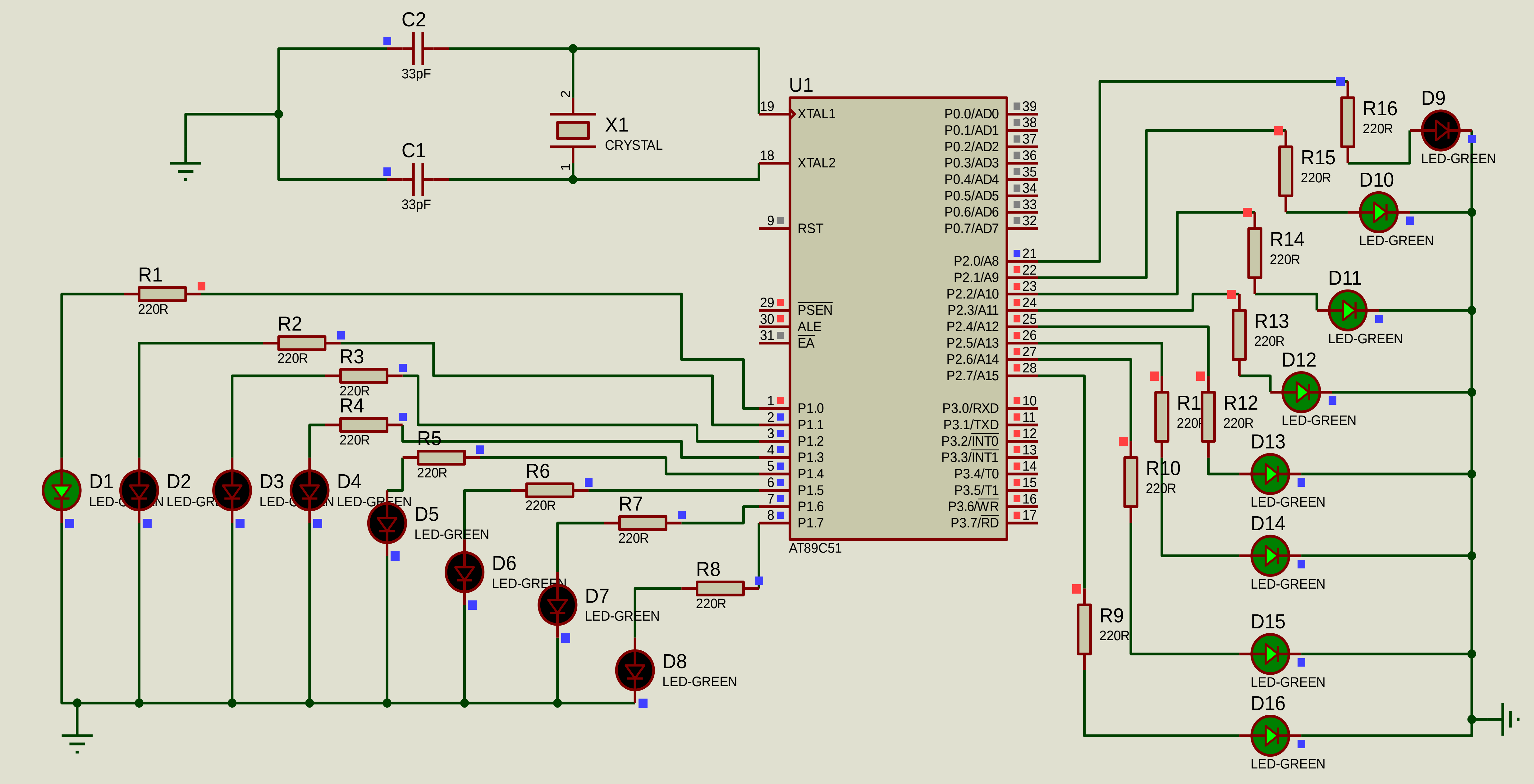


# Multiply Two Numbers stored in Memory

- Write a program to multiply two numbers stored in memory location 40H and 41H and store the resultant product in 50H (Lower Byte) and 51H (Higher Byte), respectively. Also send these values to display the output in Port P1 and P2.

```
     // Program to Mutliply Two numbers... Display Product in Port P1 and P2
            ORG 0000H
            LJMP START

            ORG 0050H
START:      MOV     P1, #00H           ; Initialize Port P1 as output port
            SETB    PSW.4              ; Select Register BANK 03
            SETB    PSW.3
            MOV     R0,  #0FFH         ; Store Values to be multiplied in regsiters
            MOV     R1,  #0FFH
            MOV     40H, R0            ; Store data in memory location 40H
            MOV     41H, R1            ; Store data in memory location 41H

            ; Muliplication Operation
            MOV     A,  40H            ; Get the value stored in 40H
            MOV     B,  41H            ; Get the value stored in 41H
            MUL     AB                 ; Multiply Two Numbers
            MOV     50H, A             ; Store the product in consicutive memory locations
            MOV     51H, B
            MOV     P1,  A             ; Display Resultant Sum in Port P1
            MOV     P2,  B

HERE:       SJMP    HERE               ; Stay HERE: Inifinity LOOP
            END         ; HLT The programe HERE
```

# Multiply Two Numbers stored in Memory (Ckt. Diagram)



# Addition of Two Numbers stored in External Memory

- See the following external RAM memory add the content of location 2000H and 2001H. Store the resultant carry in external RAM location 2003H and sum value in 2002H.

```
// Add the content stored in External RAM

            ORG     0000H
            LJMP    START

            ORG     0050H
START:      SETB    PSW.4           ; Select BANK 03
            SETB    PSW.3
            MOV     R2, #00H    ; Carry BIT indicator
            ; Access the external RAM locations
            MOV     DPTR,   #2000H      ; Ext. RAM location
            MOVX    A,      @DPTR       ; GET the value in Memory Location hold by DPTR
            MOV     R3,     A           ; Store the value in R3
            INC     DPTR                ; Get the next Adsress Location
            MOVX    A,      @DPTR
            ADD     A,      R3          ; ADD two numbers
            JNC     SKIP                ; If not carry is SET jump to SKIP
            INC     R2
            ; NOW Store the addition result value in next memory location
SKIP:       INC     DPTR
            MOVX    @DPTR,  A
            INC     DPTR
            MOV     A,      R2
            MOVX    @DPTR,  A
HERE:       SJMP    HERE
```

| External RAM | |
|---|---|
| 2000H | #Num1 |
| 2001H | #Num2 |
| 2002H | #SUM |
| 2003H | #Carry |

# Bulk data transfer (Internal RAM)

- Move 10 Bytes of data from some internal RAM locations to some other locations of 10 Bytes.

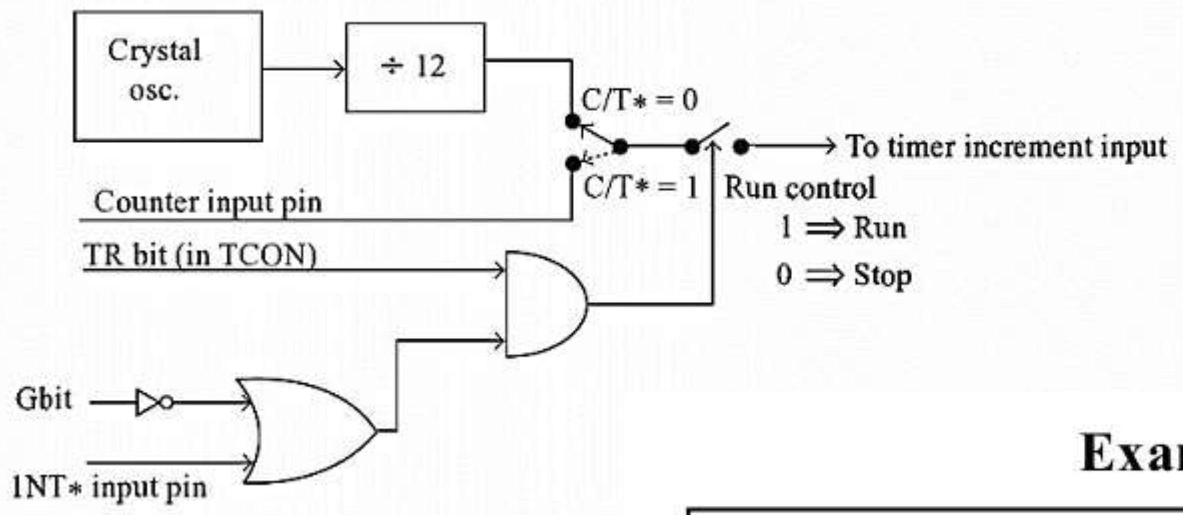
```
// Move 10 bytes of data from internal RAM locations to other locations

            ORG     0000H
            LJMP    START

            ORG     0050H
START:      SETB    PSW.4           ; Select BANK 03
            SETB    PSW.3

            MOV     R0,     #30H    ; Starting location of internal RAM where data is stored
            MOV     R1,     #40H    ; Starting location where data to be transfered
            MOV     R7,     #0AH    ; Counter Register
            ; Start transfer
BACK:       MOV     A,      @R0     ; Indirect addressing mode
            MOV     @R1,    A
            INC     R0              ; Next Address location
            INC     R1
            DJNZ    R7,     BACK    ; Do the transfer for exactly 10 Bytes
HERE:       SJMP    HERE
```

# Practice the following programs

**Text Book**: 8051 Microcontroller and Embedded Systems: Mazidi and Mazidi

1. Move 10 Bytes of data from some internal RAM locations to external RAM locations of 10 Bytes.
2. Transfer data of 10 bytes from external RAM locations 5140H to external RAM location starting from 9384H
3. The word "SAM" is to be burned in the flash ROM locations starting from 0040H. Write a program to do this and also read this data into internal RAM locations starting from 60H.
4. Write an assembly language program (ALP) to divide two numbers '75H' and '25H' and store the result in 'BANK 2' register R0 (Quotient) and R1 (Remainder). Display the values in PORT P1 (Quotient) and P2 (Remainder).
5. Write a ALP for 8051 to find the square root of a perfect square number and display the resultant value in port P1.
6. Write an ALP for 8051 to generate 3-Bit Up Counter and show the changes with exactly 0.5msec delay in PORT pin P1.0 (LSB), P1.1, P1.2 (MSB).
7. Write an ALP for 8051 to generate delay of on time period 0.05mses and off time period 0.1msec.
8. Write a program to continuously get 8-bit of data from Port P1 and send it to Port 0. Simultaneously, generate clock on Port pin P2.0.

# 8051: Timer/Counter Operations




## Example 9-15

Find the frequency of a square wave generated on pin P1.0.

**Solution:**

```
        MOV   TMOD,#2H    ;Timer 0,mode 2
        MOV   TH0,#0
AGAIN:  MOV   R5,#250     ;count 250 times
        ACALL DELAY
        CPL   P1.0
        SJMP  AGAIN
DELAY:  SETB  TR0         ;start
BACK:   JNB   TF0,BACK
        CLR   TR0         ;stop
        CLR   TF0         ;clear TF
        DJNZ  R5,DELAY    ;timer 2: auto-reload
        RET
```

$T = 2\ (250 \times 256 \times 1.085\ \mu s) = 138.88$ ms, and frequency = 72 Hz. 60