UNIT-5
8051 INSTRUCTION SET AND PROGRAMMING

- Introduction,
- Addressing modes of 8051,
- Instruction set of 8051,
- Data Transfer Instructions,
- Data and Bit-Manipulation Instructions,
- Arithmetic Instructions,
- simple programs,
- **Interfacing Examples:** External memory interfacing in 8051,
- interfacing of push button switches and LEDS,
- Interfacing of Relay, Interfacing of seven segment displays,
- Interfacing of Key board.
- **Case Study:**
- 1. Interfacing of Seven segment display with 8051 microcontroller
- 2. Switch interfacing with 8051 microcontroller
- 3. Relay interfacing with 8051 microcontroller

INTRODUCTION

With the basic idea on the architecture and the memory organization of 8051, it is easy to study the instruction set and its flexibility for control applications. Unlike the 8085 instruction set, 8051 instruction set has the instructions for bit manipulations. the 8051 instruction set supports the addressing modes such as indexed addressing and relative addressing.

ADDRESSING MODES OF 8051

The way by which a data in specified in an instruction is called as addressing mode. The data fetched for execution depends upon the addressing mode.

The instruction set of **8051 supports 5 addressing modes**.

1. Immediate Addressing
2. Register Addressing
3. Direct Addressing or Memory Direct addressing
4. Register Indirect Addressing or Memory indirect Addressing
5. Indexed Addressing

Immediate Addressing Mode:

- The data to be manipulated is directly given in the instruction itself.
- The data is preceded by a # symbol.
- **E.g. ADD A, #80h.**
- This instruction adds the data 80h to the contents of the accumulator and the result is stored in the accumulator itself.

For example –

ADD A, #77; Adds 77 (decimal) to A and stores in A

ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A

MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

Register Addressing Mode:

The register, that contains the data to be manipulated, is specified in the instruction.

E.g. ADD A, R0.

This instruction will add the contents stored in register R0 with the accumulator contents and store the result in accumulator.
The registers A, DPTR and R0 to R7 are used in Register direct addressing.
This addressing mode uses temporary registers which hold the data for the operation.

Direct Addressing or Memory Direct Addressing:

- The memory address that contains the data to be operated is specified here in the instruction.
- **E.g. ADD A, 74h**.
- This instruction **adds the data in accumulator** with that stored in **memory address 74h**.
- All **internal RAM addresses** including that of special function registers can be used in memory direct addressing instructions.
- This addressing mode is used when the **data stored in memory** is to be used in arithmetic and logical instructions.
- The data in memory used in the direct addressing can be changed at any other point in the program

Register indirect Addressing Mode:

- It is also called Memory Indirect Addressing mode
- The register, which contains the actual memory address of the data, is specified in the instruction.
- The register specified is preceded by @ symbol in assembly language format.
- **E.g. ADD A, @R0.**
- The value stored in the register R0 is now the address of the memory location of the data to be fetched.
- From this memory location, the data is fetched and the instruction is executed.
- The **data pointer register (DPTR)** is used to access the data in the **external memory with 16-bit addresses**.

Indexed Addressing:

- In this type of addressing, the instruction consists of two parts - a base address and an offset.
- This type of addressing is useful in **relative memory a**ccessing and relative jumping.
- The **base address** is stored in **data pointer (DPTR)** or any other register.
- The **offset** value is stored in **Accumulator**
- ***e.g.* MOVC A, @A+DPTR**.
- **MOVX A, @R0;** Moves content of 8-bit address pointed by R0 to A
- **MOVX A, @DPTR;** Moves content of 16-bit address pointed by DPTR to A

INSTRUCTION SET OF 8051:

Instruction supported by 8051 can be classified into different types depending upon their operational functions.

- The instruction set classification is as followed.
    1. Arithmetic Instructions
    2. Logical Instructions or Bit Manipulation Instructions
    3. Data Transfer instructions
    4. Boolean Variable Instructions
    5. Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while write instructions.
A: Accumulator
B: "B" register
C: Carry bit
Rn: Register R0 - R7 of the currently selected register bank

Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).
@Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.
#data8: Immediate 8-bit data available in the instruction.
#data16: Immediate 16-bit data available in the instruction.
Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).
Addr16: 16-bit destination address for long call or long jump.
Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.
bit: Directly addressed bit in internal RAM or SFR

Arithmetic Instructions:

- These instructions are used to do **arithmetic operations**.
- The common arithmetic operations like addition, subtraction, multiplication and division are possible with 8051.
- All the data used in arithmetic instructions must be available inside the **controller i.e. in the internal RAM area only**.
- **ADD** instruction is used to add any 8 bit data with Accumulator and the result is stored in Accumulator (A) register. The carry generated if any is stored in Carry flag of the processor status word.
- The **ADDC** instruction is also used to add any 8 bit data with Accumulator along with Carry bit.
- The **SUBB** instruction -subtract contents of a register from the Accumulator content and during this subtraction, the Carry bit is also subtracted from the accumulator.
- For **ADD and SUBB** instructions, one of the data must be in Accumulator and the other data - in any direct addressed or indirect addressed internal memory location or can be an immediate data.
- The register B is exclusively used for these two instructions. The operands should be stored in the registers A and B for the **MUL and DIV** instructions.
- The MUL instruction multiplies the contents of **A and B** registers and stores the 16 bit result in the combined **A and B** registers.
- The lower order byte -result is stored in **A register** and the higher order byte - stored in **B register**.
- The DIV instruction upon execution will divide the contents of **A register by the contents of B register**.
- The **quotient of the result - stored in A register** and the **remainder is stored in B register**.
- A division by 0 i.e. 0 in the B register before executing **DIV AB** will result in the overflow flag (OV) set to 1.
- DAA instruction -to convert binary sum obtained after adding two BCD numbers into BCD number.

Mnemonics	Description	Bytes	Instruction Cycles
ADD A, Rn	A ⟵A + Rn	1	1
ADD A, direct	A ⟵A + (direct)	2	1
ADD A, @Ri	A ⟵A + @Ri	1	1
ADD A, #data	A ⟵A + data	2	1
ADDC A, Rn	A ⟵A + Rn + C	1	1
ADDC A, direct	A ⟵ A + (direct) + C	2	1
ADDC A, @Ri	A ⟵A + @Ri + C	1	1
ADDC A, #data	A ⟵A + data + C	2	1
DA A	Decimal adjust accumulator	1	1
DIV AB	Divide A by B A ⟵quotient B ⟵remainder	1	4
DEC A	A ⟵A -1	1	1
DEC Rn	Rn ⟵Rn - 1	1	1
DEC direct	(direct) ⟵(direct) - 1	2	1
DEC @Ri	@Ri ⟵@Ri - 1	1	1
INC A	A ⟵A+1	1	1
INC Rn	Rn ⟵Rn + 1	1	1
INC direct	(direct) ⟵(direct) + 1	2	1
INC @Ri	@Ri ⟵@Ri +1	1	1
INC DPTR	DPTR ⟵DPTR +1	1	2
MUL AB	Multiply A by B A ⟵low byte (A*B) B ⟵high byte (A* B)	1	4
SUBB A, Rn	A ⟵A - Rn - C	1	1
SUBB A, direct	A ⟵A - (direct) - C	2	1
SUBB A, @Ri	A ⟵A - @Ri - C	1	1
SUBB A, #data	A ⟵A - data - C	2	1

Logical Instructions:

In addition to logical **AND, OR and XRL operation**, 8051 has additional instructions - **CLR, CPL**.

All the data for the logical instructions -**available in the internal RAM only**.

The instruction **CLR A** -to clear the contents of **A register**,

CPL is used to complement or logically invert the contents of the A register and

SWAP - to swap the **nibbles of A register**.

8051 supports **four rotate operations** with the options –**rotating left or right and rotating through carry or not**.

Mnemonics	Description	Bytes	Instruction Cycles
ANL A, Rn	A ⟵A AND Rn	1	1
ANL A, direct	A ⟵A AND (direct)	2	1
ANL A, @Ri	A ⟵A AND @Ri	1	1
ANL A, #data	A ⟵A AND data	2	1
ANL direct, A	(direct) ⟵(direct) AND A	2	1
ANL direct, #data	(direct) ⟵(direct) AND data	3	2
CLR A	A⟵ 00H	1	1
CPL A	A⟵A	1	1
ORL A, Rn	A ⟵A OR Rn	1	1
ORL A, direct	A ⟵A OR (direct)	1	1
ORL A, @Ri	A ⟵A OR @Ri	2	1
ORL A, #data	A ⟵A OR data	1	1
ORL direct, A	(direct) ⟵(direct) OR A	2	1
ORL direct, #data	(direct) ⟵(direct) OR data	3	2
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within Acumulator	1	1
XRL A, Rn	A ⟵A EXOR Rn	1	1
XRL A, direct	A ⟵A EXOR (direct)	1	1
XRL A, @Ri	A ⟵A EXOR @Ri	2	1
XRL A, #data	A ⟵A EXOR data	1	1
XRL direct, A	(direct) ⟵(direct) EXOR A	2	1
XRL direct, #data	(direct) ⟵(direct) EXOR data	3	2

Data Transfer Instructions:

- As the name indicates, instructions in this set are used to transfer data.
- The data can be transferred from or to external RAM or within the internal memory itself.
- The instruction MOV is used to transfer the data between internal registers/memory.
- The general format is
  - **MOV Reg destination, Reg source.**
- The source and destination registers within the 8051 chip can be addressed by any one of the addressing modes except indexed addressing mode discussed earlier.

Mnemonics	Description	Bytes	Instruction Cycles
MOV A, Rn	A ⟵Rn	1	1
MOV A, direct	A ⟵(direct)	2	1
MOV A, @Ri	A ⟵@Ri	1	1
MOV A, #data	A ⟵data	2	1
MOV Rn, A	Rn ⟵A	1	1
MOV Rn, direct	Rn ⟵(direct)	2	2
MOV Rn, #data	Rn ⟵data	2	1
MOV direct, A	(direct) ⟵A	2	1
MOV direct, Rn	(direct) ⟵Rn	2	2
MOV direct1, direct2	(direct1) ⟵(direct2)	3	2
MOV direct, @Ri	(direct)⟵ @Ri	2	2
MOV direct, #data	(direct) ⟵#data	3	2
MOV @Ri, A	@Ri ⟵A	1	1
MOV @Ri, direct	@Ri ⟵(direct)	2	2
MOV @Ri, #data	@Ri ⟵data	2	1
MOV DPTR, #data16	DPTR ⟵data16	3	2
MOVC A, @A+DPTR	A ⟵Code byte pointed by A + DPTR	1	2
MOVC A, @A+PC	A ⟵Code byte pointed by A + PC	1	2
MOVC A, @Ri	A ⟵Code byte pointed by Ri 8-bit address)	1	2
MOVX A, @DPTR	A ⟵External data pointed by DPTR	1	2
MOVX @Ri, A	@Ri ⟵A (External data - 8bit address)	1	2
MOVX @DPTR, A	@DPTR ⟵A(External data - 16bit address)	1	2
PUSH direct	(SP) ⟵(direct)	2	2
POP direct	(direct) ⟵(SP)	2	2
XCH Rn	Exchange A with Rn	1	1
XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Program Branching Instructions:

- 8051 supports unconditional jumping and subroutine calling in three different ways.
- They are Absolute jump AJMP, ACALL, long jump LJMP, LCALL, and short jump SJMP.

Un Conditional Branching Instructions

Mnemonics	Operation
SJMP rel_addr	Jump to (PC) + 8-bit rel_addr.
AJMP 11-bit addr	Jump to PC:addr.
LJMP addr	Jump to addr.
JMP @A + DPTR	Jump to A + DPTR.
ACALL 11-bit addr	Call subroutine at PC:addr.
LCALL addr	Call subroutine at addr.
RET	Return from subroutine.
RETI	Return from interrupt.
NOP	No operation

- The syntax for short jump instruction- **SJMP 8-bit address.**
- This 8 bit address is a relative address- to the program counter.
- **The branching address** -by adding the address given in the instruction with the program counter content.
- The **8-bit address** is a 2's complement number i.e., the most significant bit -sign + or -. The remaining **7 bits - specify the address**.
- using **SJMP** -branch to anywhere between 127 bytes after the program counter content and 128 bytes before it.(From (PC-128 bytes) to (PC+127 bytes))

For example,

8800: SJMP 06h

This instruction shift the execution to the location 8808h. The program counter content after fetching the 2 byte - SJMP instruction is 8802h. So, 06h added to 8802H results in 8808h.

The syntax for **LJMP -"LJMP 16-bit address**". After the execution of this instruction the Program counter -loaded with the 16 bit address and the execution shifts to that location.

The syntax for **AJMP instruction is "AJMP 11 bit jump address".**

Conditional Branching Instructions:

Mnemonics	Description	Bytes	Instruction Cycles
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A, #data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn, #data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2
CJNE @Ri, #data, rel	Compare with @Ri A, jump (PC + rel) if not equal	3	2
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2

JZ rel	If A=0, jump to PC + rel	2	2
JNZ rel	If A ≠ 0 , jump to PC + rel	2	2

Boolean Variable Instructions or Bit Manipulation Instructions:

- The special feature of the 8051 micro controller is that it can handle **bit data** also like that of **byte data.**
- The **internal data memory map of 8051** has a **bit- addressable area** also.
- The special function registers that have the **address with 0 or 8** as last digit in their hex address are also **bit addressable**.
- The **bit manipulation instructions** include logical instructions and conditional branching.

Mnemonic	Operation
ANL C,bit	**C = C AND bit**
ANL C,/bit	**C = C AND (NOT bit)**
ORL C,bit	**C = C OR bit**
ORL C,/bit	**C = C OR (NOT bit)**
MOV C,bit	**C = bit**
MOV bit,C	**bit = C**
CLR C	**C = 0**
CLR bit	**bit = 0**
SETB C	**C = 1**
SETB bit	**bit = 1**
CPL C	**C = NOT C**
CPL bit	**bit = NOT bit**
JC rel	**Jump if C = 1**
JNC rel	**Jump if C = 0**
JB bit,rel	**Jump if bit = 1**
JNB bit,rel	**Jump if bit = 0**
JBC bit,rel	**Jump if bit = 1 ; CLR bit**

-
- **The logical instructions - ANL and ORL. Conditional branching - JC, JNC, JB, JNB, JBC.**
- **The other instructions available -CLR, SETB, CPL, and MOV.**
- There are no instructions for halting the machine execution.

Instructions that affect Flag bits

Instruction	Flags Affected		
	C	OV	AC
ADD	√	√	√
ADDC	√	√	√
SUBB	√	√	√
MUL	0	√	
DIV	0	√	
DAA	√		
RRC	√		
RLC	√		
SETB C	1		
CLR C	0		
CPL C	√		
ANL C,bit	√		
ANL C,/bit	√		
ORL C,bit	√		
ORL C,/bit	√		
MOV C,bit	√		
CJNE	√		

Programming Examples Using 8051 Instruction Set

Example 1: **Program to add three 8-bit numbers**

- **The following program is developed assuming that the numbers are in memory locations 30h, 31h and 32h of the internal data RAM and the result is stored in memory locations in 50h and 51h of the internal RAM.**

- ***Algorithm:***

  1.The first byte is moved to the accumulator and the second byte is added with it.
  2.If carry flag is set, register R1 is incremented.
  3.The third byte is added with the intermediate result.
  4.If carry flag is set, register R1 is again incremented.
  5.The accumulator forms the least significant byte of the result and register R1 forms the most significant byte of the result.

- *Program:*

```
START:MOV R1, #00h        ;Set a register for MSB for result
      MOV R0, #30h        ;Set starting address for memory location
      MOV A, @R0          ;Get a data
      INC R0              ;Point to next memory location
      ADD A, @R0          ;Add the data
      JNC L1              ;check for carry
      INC R1        ; If carry is present, increment MSB of result
      L1:INC R0           ;Point to next memory location
      ADD A, @R0          ;Add the third data
      JNC L2              :Check for carry
      INC R1        ; If carry is present, increment MSB of result
      L2: MOV 50h,A       ; Save the result
      MOV 51h,R1
```

Example 2

Program to fill a block of memory in internal RAM with a specific data

- ***Program:***

```
START: MOV R1, #COUNT      ;load number count
       MOV R0, #30         ;load starting address of memory
LOOP:MOV @R0, #DATA        ;load data to memory location pointed by R0
       INC R0              ;Point to next memory location
       DJNZ R1, LOOP       ;Check for count and loop
```

Following program uses direct addressing for memory location for achieving the same for the memory locations from 30h to 34h.

```
       MOV 30, #DATA
       MOV 31, #DATA
       MOV 32, #DATA
       MOV 33, #DATA
       MOV 34, #DATA
```

Example 3

- **Program to add two BCD numbers**

  The following program is developed assuming that the BCD numbers are in memory locations 30h, and 31h of the internal data RAM and the result is stored in memory locations in 50h and 51h of the internal RAM with the lower order sum in 50h and carry if any in 51h.

Algorithm:

- **The first byte is moved to the accumulator and is added with the second byte.**
- **The accumulator is now decimal adjusted.**
- **The value 00h is moved to the accumulator and is added with carry.**
- The result is stored in the memory locations 50h and 51h.

- ***Program:***

```
       MOV A, 30h          ;Get a data
       ADD A, 31h          ;Add the second data
       DAA                 ;Decimal adjust accumulator
       MOV 50h,A           ;Save the sum
       MOV A, #00h   ;
       ADDC A, #00h        ;Get the MSB of sum to A register
       MOV 51h,A           ;Save that
```

Example 4

Program to add two 16 bit data

In this example, the data are assumed to be initially stored in the external memory locations. First data is stored in locations 4000H and 4001H while the second data is stored in locations 4002H and 4003H.

- **Program:**

```
        MOV  DPTR, #4000H          ;Point to first data
        MOVX A,@DPTR               ;
        MOV R0,A            ;Get the LSB of first data to R0
        INC DPTR            ;Point to MSB of first data
        MOVX A, @DPTR
        MOV R1,A            ;Get the MSB of first data to R1
        INC DPTR
        MOVX A,@DPTR        ;Get the LSB of second data
        ADD A, R0           ; Add the LSB of two data
        MOV R0,A            ; Store the sum to  the R0 register
        INC DPTR
        MOVX A,@DPTR        ;Get the MSB of second data
        ADDC A,R1          ;Add the MSBs of data along with carry out of previous addition
        MOV R1,A            ;Store the MSB of sum to R1 register
```

- **The sum is stored in the R0 and R1 registers at the end of the execution of above program.**

Example 5

Program to shift a 4-digit BCD number to left by 1 digit. Assume that the data is stored in 30h and 31h.

Algorithm:

- **The value 00h is stored in the memory location 35h.**
- **The least significant two digits (byte) are moved to the accumulator.**
- **The nibbles of the accumulator are reversed and then the least significant nibble is exchanged with the value stored in the memory location 35h.**
- **The result is stored in the memory location 50h.**
- The same process is repeated for the next byte and the result is stored in the memory locations 51h and 52h.

- ***Program:***

```
        MOV 35h, #00h   ;Initialize intermediate storage register
        MOV R0, #35h    ;initialize memory pointer
        MOV A, 30h                ;Get the Least Significant 2 digits of the data
        SWAP A          ;Exchange the two digits(nibbles)  within the data
        XCHD A, @R0     ;Move the tens digit to memory location
        MOV 50h,A    ;A reg has unit digit of BCD data shifted to left nibble with LSB
        as 0
        MOV A, 31h      ;Higher order data is brought to A reg
        SWAP A          ;Exchange lower and higher order nibbles
        XCHD A, @R0    ;Move the thousandth digit to memory and tens digit to A
                                                                    register
        MOV 51h,A       ;Save the shifted data to memory
        MOV 52h,@R0     ;Save the thousandth digit
```

Example 6

Program to read a byte from port 0 and depending upon which bit is set, jump to one of the 8 different locations.

Algorithm:

- **First the byte is moved from the port0 to one of the bit addressable bytes (i.e., within 20h-27h of the SFR)**
- **Then depending upon which bit is set, control should be transferred to one of the eight different locations.**
- Control returns to the first instruction LOOP after executing the control block for each bit for proper operation.

- ***Program:***

```
LOOP    : MOV 20h, PORT0          ;Get the data from port 0
          JB 00,L1 ; Check the LSB using the bit address and if set jump to relative
                                                                    address L1
          JB 01,L2
          JB 02,L3
          JB 03,L4
          JB 04,L5
          JB 05,L6
          JB 06,L7
          JB 07,L8
          LJMP LOOP
```

In the above program L1 toL8 are the 8 bit relative addresses to which the branching has to take place. The 8 bit relative address is assumed to be a 2's complement number and branching takes place above or below the main program.

Example 7

Program to reverse the bits within a byte.

Algorithm:

- **Assume that the byte to be reversed is stored in register R0.**
- **Initialize register R1 with 00h and register R2 with 08h.**
- **The byte from register R0 should be loaded into the accumulator.**
- **The accumulator should be shifted left through carry and has to be exchanged with register R1. Now, the LSB of the data is moved to Carry and the shifted data is moved to R1**
- **The accumulator should be shifted right through carry and has to be exchanged again with register R1. Now the LSB in the carry is shifted into R1 register. After subsequent shifts, it is moved into MSB of R1.**
- Decrement the value stored in register R2 and jump to step 5 if not zero.

- ***Program:***

```
        MOV R0, #data
        MOV R1, #00h
        MOV R2, #08h        ;Initialize counter
        MOV A, R0           ;Get the data
LOOP:   RLC A               ;Bring one bit in to Carry
        XCH A, R1
        RRC A               ;Move this bit in reversed order into R1
        XCH A, R1
        DJNZ R2, LOOP       ; Check for 8 bits and if not, repeat.
```

Example 8

Program to find the biggest number in a block of data from the memory location 70h to 7Fh.

Algorithm:

- **Initialize a memory pointer to point to starting point of the memory location.**
- **Initialize a counter for number of data**
- **The first byte stored in the block is assumed to be the biggest number and stored in R1 register**
- **The next data is compared with the biggest in R1**
- **I f the data in R1 is smaller, and then the data in the block is stored as big in the R1 register.**
- Above process is repeated for all the data in the block.

- ***Program:***

```
        MOV R1,#00h     ;Initialise R1 to store biggest
        MOV R0,#70h     :Initialise memory pointer
        MOV 30h,@R0     ; Store first data as biggest in R1 register
        MOV R1,30h
LOOP:   INC R0          ; Point to next location
        MOV A,R1
        SUBB A,@R0      ;compare by subtracting biggest in R1 register
        JNC NEXT
        MOV 30h,@R0     ; If bigger, bring the biggest to R1 register
        MOV R1, 30h
        NEXT:CJNE R0, #80h, LOOP    ;Repeat the above step for all the
data.
```

The above are only illustrative programs of basic instructions of 8051. In general, the instruction set of 8051 is very flexible and the readers are recommended to go through all the instructions to understand it.

Example 9

Write a program to find the sum of elements in an array.

ALGORITHM:

1. Set the count value.Read 1st no. As zero and set a reg. to store the carry.
2. Load the array in the consecutive memory location and initialize the memory pointer with the starting address.
3. Add the register(R1) with the accumulator.
4. Check for carry, if exist, increment the carry register by 1(R2). otherwise, continue
5. Decrement the counter (R0) and if it reaches 0, stop. Otherwise increment the memory pointer by 1 and add the previous result with next element in the array.

PROGRAM:

```
        MOV R0, #05H MOV R1, #00H
        MOV R2,#00H
        MOV DPTR, #4200
NEXT:   MOVX A, @DPTR

        ADD A, R1
        MOV R1, A
        JNC LOOP
        INC R2
LOOP:   INC DPTR
```

```
        MOV DPTR, #4500H
        MOV A, R1
        MOVX @DPTR, A
        INC DPTR
        MOV A, R2
        MOVX @DPTR, A
STOP:   SJMP STOP
```

INPUT		**OUTPUT:**	
4200	02	4500	09
4201	01	4501	00
4202	03		
4203	02		
4204	01		

Example 10

Write a program to Subtract two 8 – bit numbers using 8051.

ALGORITHM:

1. Clear C – register for Carry
2. Get the data immediately.
3. Subtract the two data
4. Store the result in memory pointed by DPTR

PROGRAM:

```
        ORG 4100
        CLR C
        MOV R0, #00H
        MOV A,#data1
        SUBB A,#data2
        JNC STORE
        INC R0
STORE:  MOV DPTR, #4500
        MOVX @DPTR, A
        INC DPTR
        MOV A,  R0
        MOVX @DPTR, A
HERE:   SJMP HERE
```

Example 11:

Write a program to multiply two 8 – bit numbers using 8051.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be multiplied in B – reg.
3. Multiply the two data
4. The higher order of the result is in B – reg.
5. The lower order of the result is in A – reg.
6. Store the results.

PROGRAM:

```
ORG 4100
CLR C
MOV A,#data1
MOV B,#data2
MUL AB
MOV DPTR,#4500
MOVX @DPTR,A
INC DPTR
MOV A,B
MOVX @DPTR,A
HERE: SJMP HERE
```

Example 12

Write a program to divide two 8 – bit numbers using 8051.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be divided in B – reg.
3. Divide the two data
4. The quotient is in A – reg.
5. The remainder is in B – reg.
6. Store the results.

PROGRAM:

```
ORG 4100
CLR C
MOV A,#data1
MOV B,#data2
DIV AB
MOV DPTR,#4500
MOVX @DPTR,A
INC DPTR
MOV A,B
MOVX @DPTR,A
HERE: SJMP HERE
```

Interfacing Examples:
External memory interfacing in 8051

If external program/data memory are to be interfaced, they are interfaced in the following way.

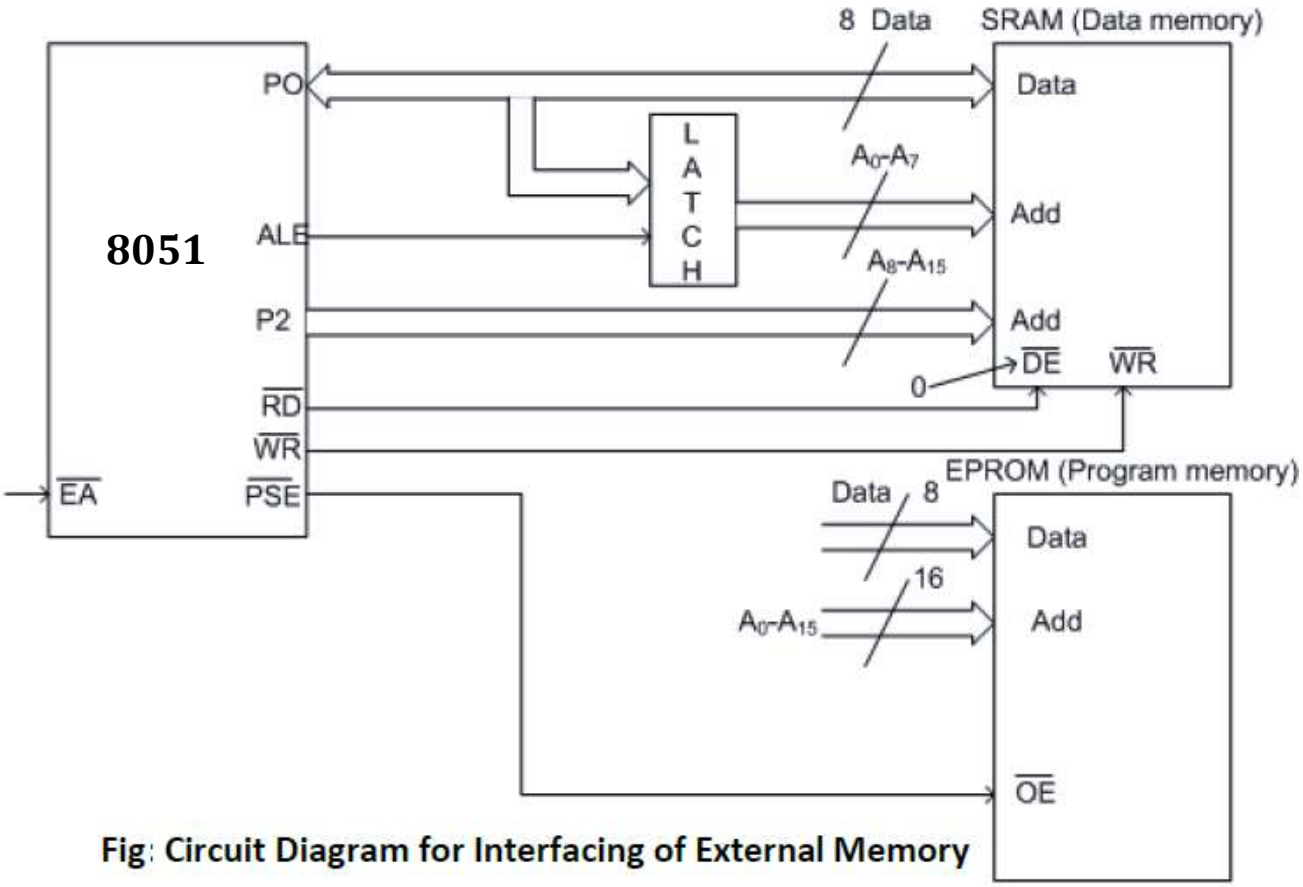


Fig: Circuit Diagram for Interfacing of External Memory

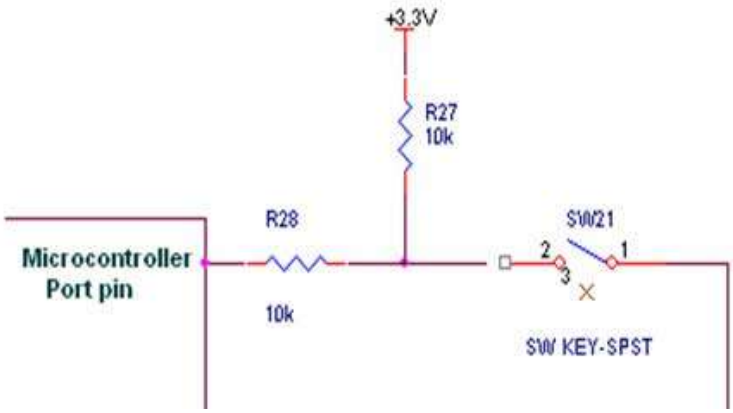
External program memory is fetched if either of the following two conditions are satisfied.
1. EA' (External Access) is low. The microcontroller by default starts searching for program from external program memory.
2. PC is higher than FFFH for 8051

$PSEN$ 'tells the outside world whether the external memory fetched is program memory or data memory.
EA' is user configurable. $PSEN$' is processor controlled.

Interfacing of push button switches and LEDS:

Switch:

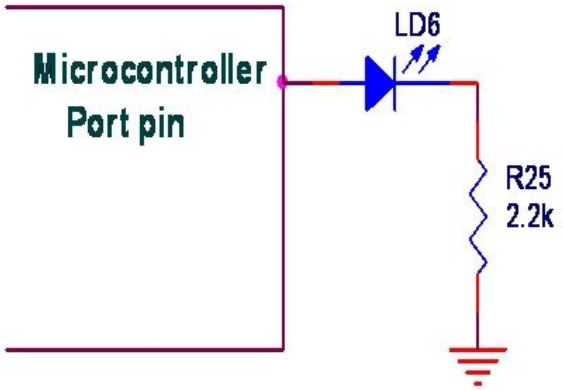
A switch is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another. A switch may be directly manipulated by a human as a control signal to a system, or to control power flow in a circuit.



Interfacing Switch to 8051:

The Fig. shows how to interface the switch to microcontroller. A simple switch has an open state and closed state. However, a microcontroller needs to see a definite high or low voltage level at a digital input. A switch requires a pull-up or pull-down

voltage when it is open or closed. A resistor placed between a digital input and the supply voltage is called a "pull-up" resistor because it normally pulls the pin's voltage up to the supply.



LED (LIGHT EMITTING DIODE)

Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

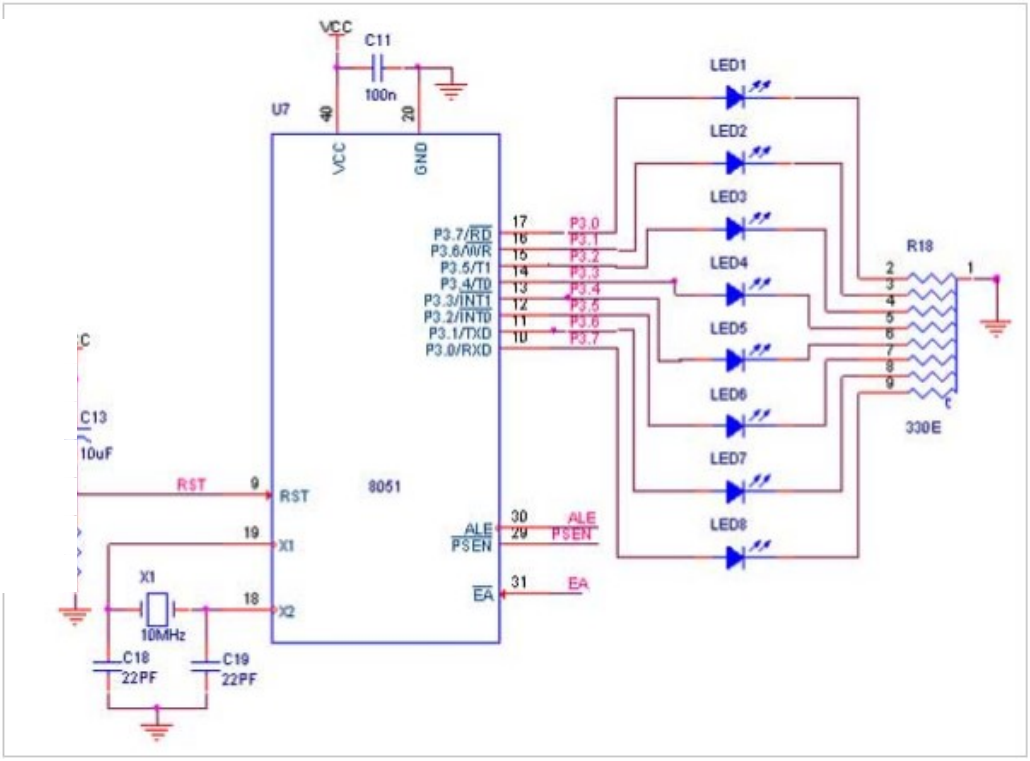
INTERFACING LED to 8051

The Fig. shows how to interface the LED to microcontroller. As you can see the Anode is connected through a resistor to GND & the Cathode is connected to the Microcontroller pin. So when the Port Pin is HIGH the LED is OFF & when the Port Pin is LOW the LED is turned ON.

Program

```
    L1:MOV A,#FF
       MOV P3,A
       LCALL DELAY
       MOV A,#00
       MOV P3,A
       LCALL DELAY
       SJMP L1
DELAY: MOV R5,#05
H3     :MOV R4,#FF
H2     :MOV R3,#FF
H1     :DJNZ R3,H1
       DJNZ R4,H2
       DJNZ R5,H3
       RET
```

`



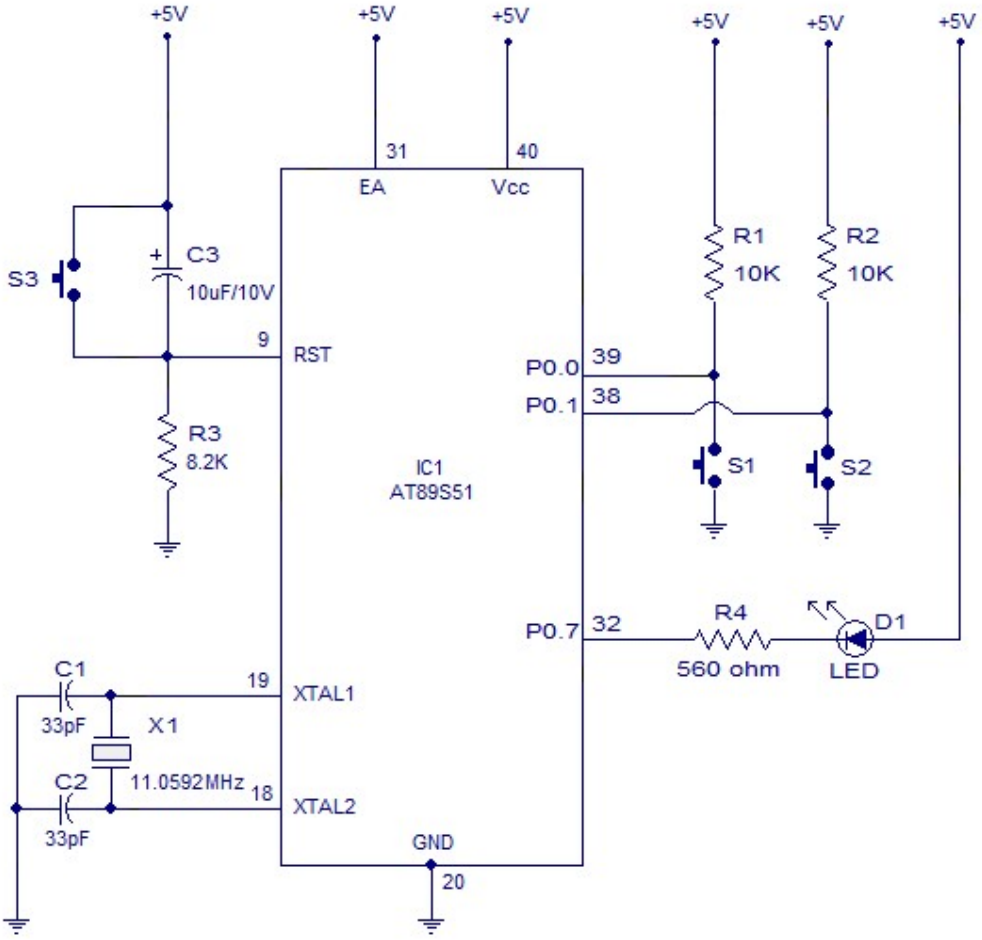
Circuit Applications

- LEDs are widely used in many applications like in seven segments.
- They are used in dot matrix displays.
- They can be used for street lights.
- They are used as indicators.
- They can be used in traffic lights.
- They are used in emergency lights

Interfacing of push button switches and LEDS with 8051:

The circuit diagram for interfacing push button switch to 8051 is shown below. AT89S51 is the microcontroller used here. The circuit is so designed that when push button **S1** is **depressed** the LED **D1 goes ON** and remains ON until push button **switch S2** is **depressed** and this cycle can be **repeated.** Resistor R3, capacitor C3 and push button S3 forms the reset circuitry for the

microcontroller. Capacitor C1, C2 and crystal X1 belongs to the clock circuitry. R1 and R2 are pull up resistors for the push buttons. R4 is the current limiting resistor for LED.

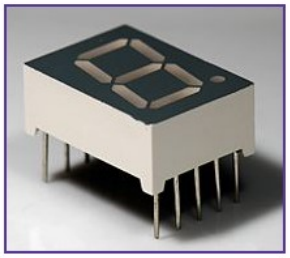


Program:

```
        MOV P0, #83H   ; Initializing push button switches and initializing LED in OFF state.
READSW: MOV A, P0      ; Moving the port value to Accumulator.
        RRC A          ; Checking the vale of Port 0 to know if switch 1 is ON or not
        JC NXT         ; If switch 1 is OFF then jump to NXT to check if switch 2 is ON
        CLR P0.7       ; Turn ON LED because Switch 1 is ON
        SJMP READSW    ; Read switch status again.
   NXT: RRC A          ; Checking the value of Port 0 to know if switch 2 is ON or not
        JC READSW      ; Jumping to READSW to check status of switch 1 again (provided sw2 is OFF)
        SETB P0.7      ; Turning OFF LED because Switch 2 is ON
        SJMP READSW    ; Jumping to READSW to read status of switch 1 again.
```

Interfacing of seven segment displays:

The 7 segment LED display is very popular and it can display digits from **0 to 9** and quite a few characters like **A, b, C, ., H, E, e, F, n, o, t, u, y**, etc. Knowledge about how to interface a seven segment display to a micro controller is very essential in designing embedded systems. A seven segment display consists of seven LEDs arranged in the form of a squarish '8' slightly inclined to the right and a single LED as the dot character. Different characters can be displayed by selectively glowing the required LED segments. Seven segment displays are of two types, **common cathode and common anode.** In common cathode type , the cathode of all LEDs are tied together to a single terminal which is usually



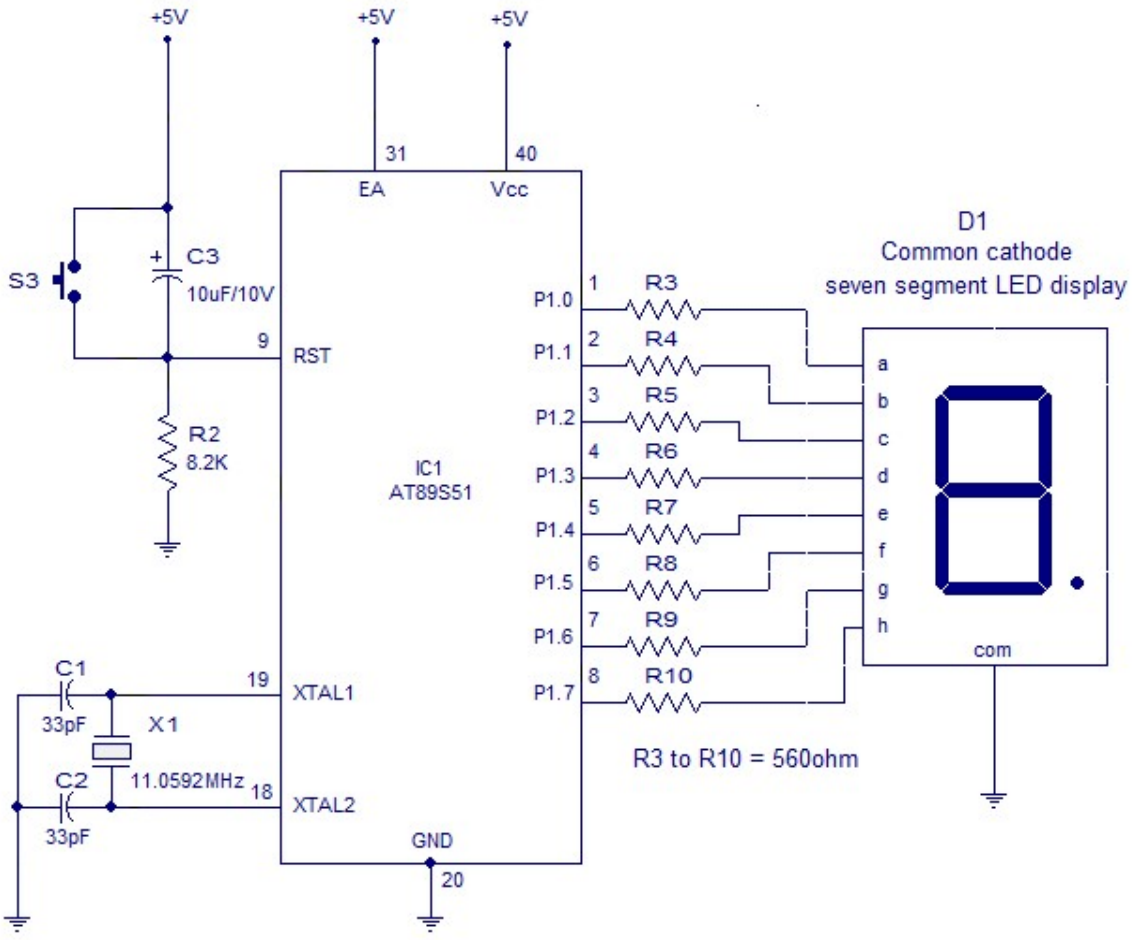
labeled as 'com' and the anode of all LEDs are left alone as individual pins labeled as **a, b, c, d, e, f, g & h (or dot)** . In common anode type, the anode of all LEDs are tied together as a single terminal and cathodes are left alone as individual pins. The pin out scheme and picture of a typical 7 segment LED display is shown in the image below.

Digit drive pattern:

Digit drive pattern of a seven segment LED display is simply the different logic combinations of its terminals 'a' to 'h' in order to display different digits and characters. The common digit drive patterns (0 to 9) of a seven segment display are shown in the table below.

Digit	Dp	g	f	e	d	c	b	a	Hex val
0	0	0	1	1	1	1	1	1	0x3f
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5b
3	0	1	0	0	1	1	1	1	0x4f
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7f
9	0	1	1	0	0	1	1	1	0x67

Diagram of Interfacing seven segment display to 8051.



Program

```
        ORG 000H                ;initial starting address
        MOV A, #00H             ; set a count to point the address of each digit pattern
NEXT:   MOV R0, A
        MOV DPTR, #ARRAY
        MOVC A, @A+DPTR         ;Read the Digit drive pattern
        MOV P1, A               ;Move to the port for display
        ACALL DELAY             ; Calls the delay
        MOV A, R0               ; R0
        INC A
        SJMP NEXT
ARRAY:  DB 3FH          ; digit drive pattern for 0
        DB 06H          ; digit drive pattern for 1
        DB 5BH          ; digit drive pattern for 2
        DB 4FH          ; digit drive pattern for 3
        DB 66H          ; digit drive pattern for 4
        DB 6DH          ; digit drive pattern for 5
        DB 7DH          ; digit drive pattern for 6
        DB 07H          ; digit drive pattern for 7
        DB 7FH          ; digit drive pattern for 8
        DB 6FH          ; digit drive pattern for 9
DELAY:  MOV R2,#0FFH    ; subroutine for delay
WAIT:   DJNZ R2,WAIT
        RET
        END
```

Program Description.

Instruction MOVC A,@A+DPTR is the instruction that produces the required digit drive pattern for the display. Execution of this instruction will add the value in the accumulator A with the content of the data pointer (starting address of the ARRAY) and will move the data present in the resultant address to A. In the program, initial value in A is 00H. Execution of MOVC A,@A+DPTR will add 00H to the content in DPTR . The result will be the address of label DB 3FH and the data present in this address ie 3FH (digit drive pattern for 0) gets moved into the accumulator. Moving this pattern in the accumulator to Port 1 will display 0 which is the first count.

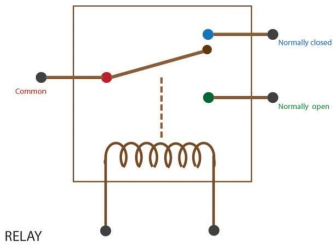
At the next count, value in A will advance to 01H and after the execution of MOVC A,@+DPTR ,the value in A will be 06H which is the digit drive pattern for 1 and this will display 1 which is the next count and this cycle gets repeated for subsequent counts.

Label DB is known as Define Byte – which defines a byte. This table defines the digit drive patterns for 7 segment display as bytes (in hex format). MOVC operator fetches the byte from this table based on the result of adding DPTR and contents in the accumulator.

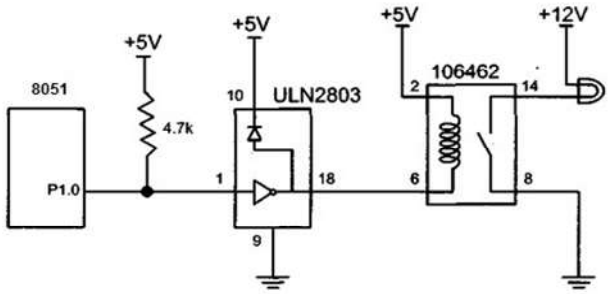
Register R0 is used as a temporary storage of the initial value of the accumulator and the subsequent increments made to accumulator to fetch each digit drive pattern one by one from the ARRAY.

Interfacing of Relay:

In some electronic applications we need to switch or control high voltages or high currents. In these cases we may use electromagnetic or solid state relays. For example, it can be used to control home appliances using low power electronic circuits.



An electromagnetic relay is a switch which is used to switch High Voltage or Current using Low power circuits. It magnetically isolates low power circuits from high power circuits. It is activated by energizing a electromagnet, coil wounded on a soft iron core.




```
        ORG 0H
MAIN:
        SETB P1.0
        MOV R5, #55
        ACALL DELAY
        CLR P1.0
        MOV R5, #55
        ACALL DELAY
        SJMP MAIN
DELAY:
H1:     MOV R4,#100
H2:     MOV R3,#253
H3:     DJNZ R3, H3
        DJNZ R4, H2
        DJNZ R5, H1
        RET
        END
```

```
;Get a byte from P0 and send it to P1
          MOV  A,#0FFH    ;A = FF hex
          MOV  P0,A       ;make P0 an input port
                          ;by writing all 1s to it
BACK:     MOV  A,P0       ;get data from P0
```