Interfacing 8051 with 7 segment display

1. Design in Proteus

Search for: at89c51rd2

7 segment common cathode

Connect pin p2.0 (21) to p2.7 (28) serially to 7 segment display

2. Write assembly in Keil micro vision
    a. Create a New Project
    b. Add a new file
    c. Copy paste the following assembly code and save .asm

```
ORG 4000H
DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H, 7FH, 6FH, 0
; Lookup table for digits 0 to 9

ORG 0000H
main:   MOV DPTR, #4000H
repeat: CLR A
    MOVC A, @A+DPTR     ; Copy data from external location to accumulator
    MOV P2, A           ; Move the pattern of the digit into port P2
    ACALL delay         ; Call a delay to so that the transition is visible
    INC DPTR            ; Point to the next pattern
    CJNE A, 0, repeat   ; Repeat till 0 (Stop bit) is received
    SJMP main           ; Run this forever till externally stopped

; generate a decent enough delay between transitions
delay:
    MOV R0, #08H
LP2:   MOV R1, #0FFH
LP1:   MOV R2, #0FFH
LP3:   DJNZ R2, LP3
    DJNZ R1, LP1
    DJNZ R0, LP2
    RET
END
```

    d. Add the file to the target group
    e. Set output as hex
    f. Build target to generate hex file
3. Upload the program file (hex file) to the microcontroller

Reference Link: https://technobyte.org/seven-segment-8051-interfacing-tutorial-quad-single-module/

Meaning of each code

This assembly code is written for the 8051 microcontroller. It demonstrates how to continuously read and display patterns of data stored at an external memory location, with a delay to make transitions visible. Here's a breakdown of each line:

ORG 0000H

- **ORG 0000H**: This directive sets the starting address for the program code to memory location 0000H. The code execution will begin from this location when the microcontroller resets.

main: MOV DPTR, #4000H

- **MOV DPTR, #4000H**: This instruction loads the DPTR (Data Pointer) register with the address 4000H. This is the starting address in external memory where the data (patterns for display) is stored.

repeat: CLR A

- **CLR A**: This instruction clears the accumulator (A register), setting its value to 00H. It's preparing to load new data into the accumulator in the next step.

MOVC A, @A+DPTR

- **MOVC A, @A+DPTR**: This instruction moves the content of the external memory location, pointed to by DPTR plus the value in the accumulator (A), into the accumulator. It allows the program to fetch data from external memory where the address is calculated by adding the value in A (which is 0 after CLR A) and the DPTR.

MOV P2, A

- **MOV P2, A**: This moves the value in the accumulator (which is now the data fetched from memory) into Port 2 (P2). Port 2 controls external devices, likely an LED display or similar peripheral, which will show the fetched pattern.

ACALL delay

- **ACALL delay**: This calls the subroutine labeled delay. It introduces a delay between displaying each pattern so that the transition from one pattern to the next is visible.

INC DPTR

- **INC DPTR**: This instruction increments the DPTR by one. It moves the pointer to the next memory location, so that the program can fetch the next data byte in the next iteration of the loop.

CJNE A, 0, repeat

- **CJNE A, 0, repeat**: This is a conditional jump. It compares the value in the accumulator (A) with 0. If A is not equal to 0, the program jumps to the repeat label and continues the loop. This means the loop will continue fetching data and displaying it until it encounters a 0 byte, which signals the end of data.

SJMP main

- **SJMP main**: This is an unconditional jump to the main label. It causes the program to go back to the start of the program, where DPTR is reloaded with 4000H and the whole process repeats forever.

---

delay: MOV R0, #08H

- **MOV R0, #08H**: This instruction loads the register R0 with the value 08H, which will be used as a counter in the delay loop.

LP2: MOV R1, #0FFH

- **MOV R1, #0FFH**: This loads register R1 with 0FFH, which will serve as another counter in the inner loop.

LP1: MOV R2, #0FFH

- **MOV R2, #0FFH**: This loads register R2 with 0FFH as a third counter for the innermost loop.

LP3: DJNZ R2, LP3

- **DJNZ R2, LP3**: This instruction decrements R2 and jumps to LP3 if R2 is not zero. It effectively introduces a delay by looping until R2 reaches zero.

DJNZ R1, LP1

- **DJNZ R1, LP1**: Similar to the previous instruction, this decrements R1 and loops to LP1 if R1 is not zero, adding another level of delay.

DJNZ R0, LP2

- **DJNZ R0, LP2**: This decrements R0 and loops back to LP2 if R0 is not zero, introducing the final level of delay.

RET

- **RET**: This instruction returns from the subroutine (in this case, it ends the delay subroutine and goes back to the main loop).

END

- **END**: This indicates the end of the program.
