Embedded Engineer's Development Tool

(EEDT 5.0)

User Manual and Tutorial Handbook

DeccanRobots

Developed and Distributed by DeccanRobots
As a part of
"Embedded Engineer's Development Tool 5.0"

www.deccanrobots.com

Forth Edition 24th July 2007

DeccanRobots
205, 2nd Floor, Decision Tower,
Next to City Pride, Satara Road,
Pune 411037 India
Ph: +91 020 24228818
www.deccanrobots.com
support@deccanrobots.com

Congratulations & Welcome!!!

Dear Customer, thank you for purchasing "Embedded Engineer's Development Tool".

About Embedded Engineer's Development Tool:

"Embedded Engineer's Development Tool" (EEDT), is a hardware + Software platform created for engineers willing to develop their embedded applications using ATMEL's AVR and 89S series microcontrollers.

EEDT is combination of various interfacing circuits, in-built microcontroller programmer, user manual, sample source codes, tutorial software, function libraries, and lifelong support via email.

EEDT has been developed for novice as well as an expert embedded engineer to develop their project & prototypes.

EEDT is simple to use and easy to understand, which enables a new user to start working with it in less than 5 minutes.

EEDT is evolved and has proved its usefulness since its first version in the year 2003.

Using "Embedded Engineer's Development Tool", one can develop their applications like Robots, Automation systems, IR apps, multi channel analog data capturing and logging system, RS232 based communication projects, PWM based motor control and lots more. With the help of external devices like GSM modems, one can make exciting projects based on mobile and telephony.

We at DeccanRobots, use the same EEDT, to develop real world projects like Cash registers, POS machines, Treadmills and Industrial equipments for our clients.

About the "User Manual and Tutorial Handbook":

This Handbook will explain how to use your "Embedded Engineer's Development Tool", how to install required software, how to use CD, internals of 89S series microcontrollers, internals AVR series microcontrollers, how to write assembly code for 89S series controller, how to write C language code for AVR microcontrollers.

About Lifelong support via Email:

As a buyer of EEDT, you have lifelong support from us via email.
We reply all the queries we receive within 24hrs during weekdays.
We anticipate that your queries will be related to the EEDT hardware, softwares and source code provided along with EEDT.

Hand Book Index

1. Embedded Engineer’s Development Tool (EEDT)

List of items and direction to use them
You must find following items as a part of your EEDT.

1. One Assembled PCB with all components mounted
2. 8S52 & mega8 Included, Other controllers can be purchased separately
3. One 9 Pin Female-Female (F-F) cable
4. One 12V Wall mounting adaptor (Not included for buyers outside India)
5. Four 8 Pin F-F multicolor connector cables
6. Four 4 Pin F-F multicolor connector cables
7. Four 2 Pin F-F multicolor connector cables
8. One 6 Pin F-F multicolor connector cable (Find it connected on the PCB)
9. One CD
10. One Handbook (which you are reading now)
11. Enclosure bag / Antistatic bubble bag
12. Additional components if you have ordered.

How to use EEDT?
Follow the procedure to get started with EEDT.

1. Unpack all above listed items.
2. Connect 12 V-1 Amp Adaptor’s output to EEDT PCB
3. Connect 12V adaptor to the mains power supply.
4. Connect 9 Pin F-F cable between your PC and EEDT PCB. You will find two numbers of 9 pin male connectors on the EEDT PCB. Use the one, which is near to the IC with sticker of “HandyProg”. This way you have connected your PC and EEDT’s in-built programmer (HandyProg).
5. Make sure that 6 Pin F-F cable is connected between the in-built programmer called HandyProg and the target device section of your choice.
6. Switch ON the power of 12V adaptor.
7. You will find a GREEN LED glowing indicating EEDT PCB has received the required power supply.
8. Now its time to work with your PC.
9. You may start your PC, if you have not yet started it.
10. Insert DeccanRobots CD in to your PC’s CD Drive.
11. Open the CD using Windows Explorer or My Computer
12. You will find these folders inside the CD
    a. “Install These Softwares”
    b. “Project Source Code”
    c. “Read These Datasheets”
13. You have to install all the softwares provided under the first directory.
14. You may copy the “Project Source Code” folder to your drive. All files from this folder will have a read-only mark. You may remove the read-only property if needed, after copying to your drive.
15. “Read These Datasheets” folder contains various datasheets you may need while developing projects.
16. Once you have installed all the softwares from the CD, you can continue using the EEDT for your project development.
17. If you are new to the field of microcontrollers, then you must completely read E-Learning tutorials. You will find E-Learning tutorials installed under Start=>Programs (All Programs) => DeccanRobots on your PC.
18. Remember that you have to keep your EEDT connected to your PC in ON condition till you are reading the tutorials.
19. These tutorials require MSOffice 2000 (to be precise PowerPoint 2000) or higher installed on your PC.
20. E-Learning tutorial software will use the same COM port as it will be used by HandyProg software. Thus you may keep either one of this open on your PC. Also E-Learning tutorial software will disable you from Copy-Paste actions.

21. If you are an embedded developer then, you can write your code for selected microcontroller using an IDE (Editor) of your choice or one which you have installed from the CD.
22. Once you are finish with the code writing, you have to start the HandyProg ISP software located under Start => Programs (All Programs) => DeccanRobots => HandyProg 4.0
23. While using HandyProg software, you have to Select COM Port number, Hex File, Device Name. You can click on Program button if you are sure about your selection. Notice the Log Window of HandyProg software for the status of your Programming activity.
24. EEDT PCB has various sections. We follow open architecture for EEDT's interfacing section. This enables you to connect any device to any pin of the selected microcontrollers or vise versa you may connect microcontrollers ports to any section provided on the EEDT PCB or to any external customized board or device.
25. To connect the selected microcontroller to interfacing section, you may use the multicolor Female - Female connectors.
26. Once you make appropriate connections and Program the selected microcontroller using HandyProg software, you can see the results of your work by observing the interfacing sections you are using. If you don't see the expected results, then you may have to look in-to your code and/or your connections between the selected microcontroller and the interfacing board to make necessary changes.
27. Read "How to Use Interfacing Sections?" for detailed circuit diagram and description.

How to use Interfacing Sections?

EEDT PCB has various interfacing sections.
One can use these sections as per the project requirement.
You can connect a particular interfacing device to microcontroller port the way you wish to. This way the EEDT has maintained the flexibility for you to experiment so that you can design your project's connections without any constraints In simple words, interfacing circuits have their input/output pins pulled out and are kept open for the usage. It will be your responsibility to connect these pins to microcontroller using multicolor F-F connectors provided with the EEDT. Required power (VCC & GND) has been provided to all interfacing sections internally, hence you do not have to connect the power to these sections.

Refer **"EEDT Board Layout.pdf"**

Name of Section	Purpose of Section
Section 1	Target Section for Tiny13 & pin compatible Tiny AVRs
Section 2	Target Section for Tiny26 & pin compatible Tiny AVRs
Section 3	Target Section for Tiny2313 & pin compatible Tiny AVRs
Section 4	Target Section for 89S52 & pin compatible controllers
Section 5	Target Section for mega8 & pin compatible mega-AVR
Section 6	Target Section for mega32 & pin compatible mega-AVR
Section 7	Target Section for mega128 & pin compatible mega-AVR
Section 8	Four numbers of 7-Segment displays
Section 9	4 pulled-up push-to-on switches
Section 10	ULN 2803 motor driver
Section 11	Real Time Clock DS1307
Section 12	TSOP 1738 IR receiver
Section 13	16x2 LCD
Section 14	DC Motor driver L293D
Section 15	24C256 EEPROM
Section 16	RS232 Interfacing
Section 17	4x4 Keyboard
Section 18	HandyProg ISP
Section 19	8 LEDs (Section name is missing)

Interfacing Sections in detail:

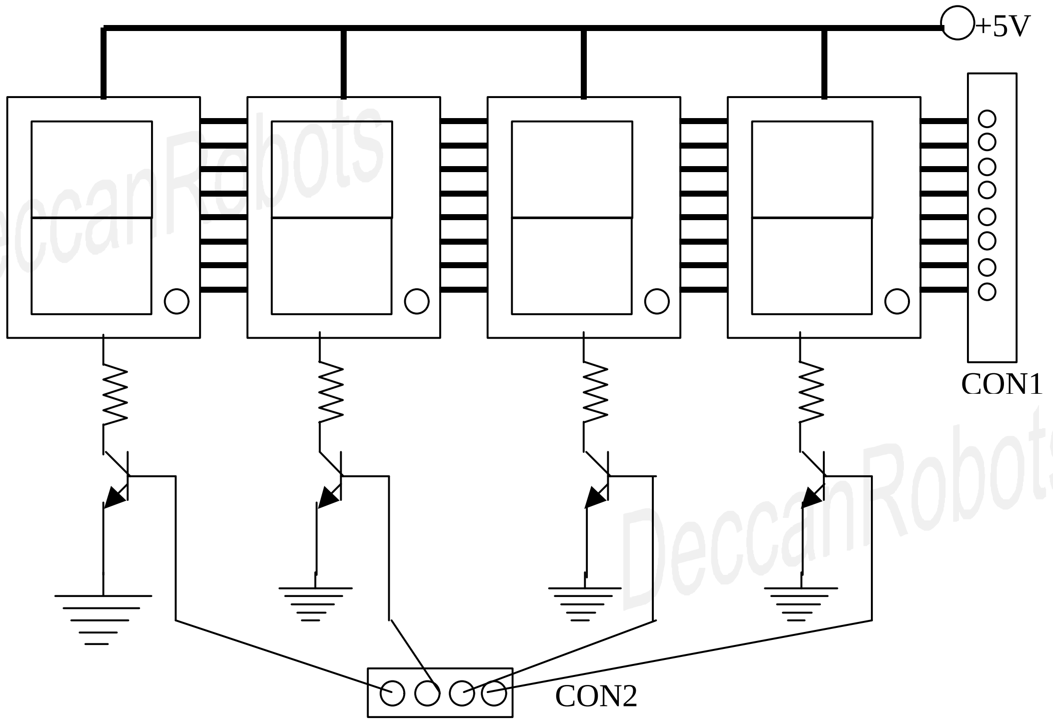
Read this part of the book to get insight of all the sections of the EEDT PCB. Sectional details are not arranged serially.

If you want to use these sections, then refer the section related to the microcontroller of your choice, e.g. “Interfacing Techniques for 89S” or “Interfacing Techniques for AVR”.

Section 8: 7-Segment LED Display

This section has four 7-segment LED displays.
CON1 and CON2 are its input connectors.
CON2’s each pin controls individual display’s power connection.
CON1 is data port and common to all 4 displays.
Refer CKT 1 for more details:

CKT 1



Section 10: ULN 2803 based high current driver

ULN 2803 is 8 channel high current driver IC. It has 8 Darlington transistors inside to drive the external load, which any microcontroller cannot drive directly.

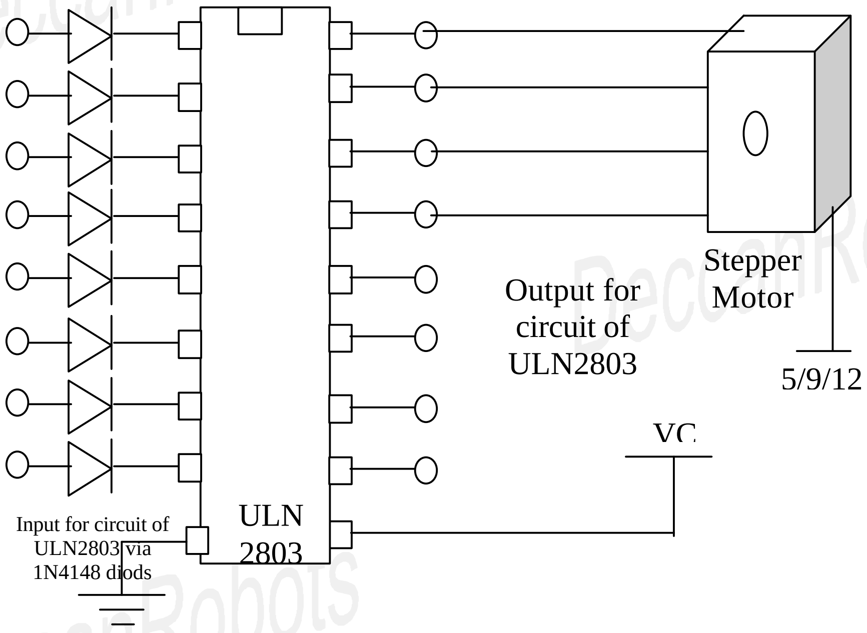
Think of controlling a stepper motor from microcontroller. Microcontroller can generate signals required to run the stepper motor, but it can’t actually handle the required amount of high current.

To handle high current, one has to use driver circuit. The driver circuit based on ULN 2803 IC requires 8 inputs and has 8 outputs.

ULN 2803 can drive external load like stepper motors having voltage rating of 24V. Refer ULN 2803 datasheet from the CD for precise technical information.

CKT 2 will explain you the details of the driver circuit.

CKT 2



Stepper Motor
Output for circuit of ULN2803
5/9/12
VC
ULN 2803
Input for circuit of ULN2803 via 1N4148 diods

Section 19: 8 LEDs

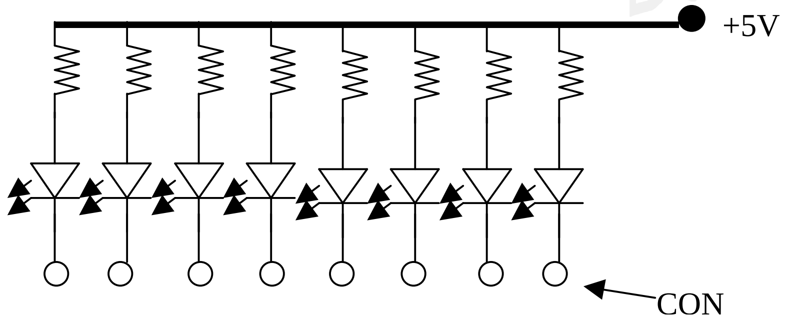
This section is madeup of 8 red color LEDs.

These LEDs can be used to test your binary results or to show status of your operation in your projects.

These LEDs will be ON if a Low signal is applied at respective pin of CON of the EEDT PCB.

CKT 3 will explain it in detail:

CKT3

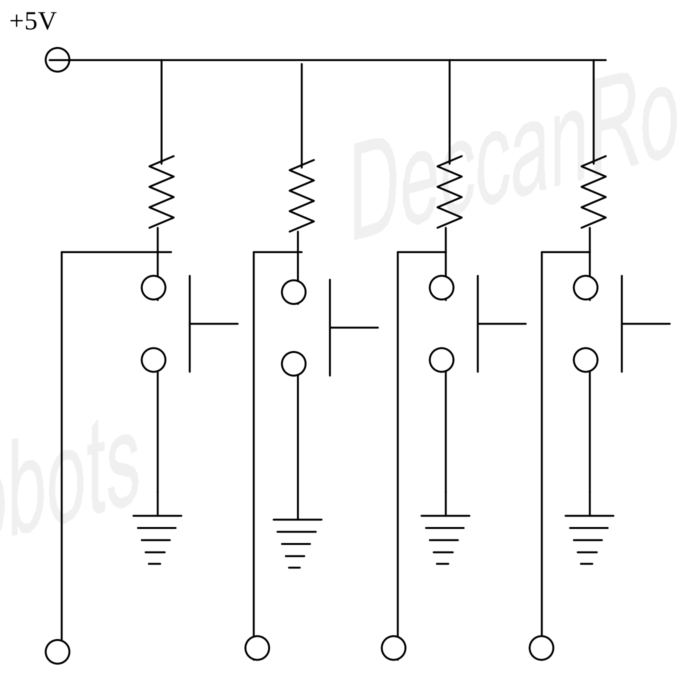


Section 9: Pull up Switches

These are push to ON switches. Four switches are pulled up, so that, when you press them, you will get a Low pulse

CKT 4 will explain it in detail.

CKT 4

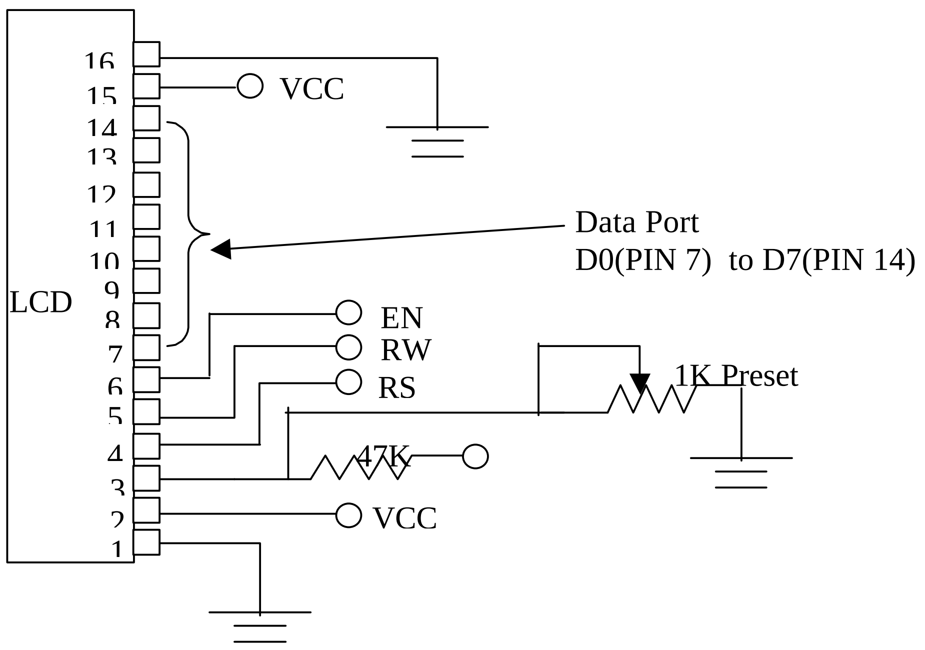


CON

Section 13: 16 X 2 LCD

16x2 LCD, i.e. 16 characters per line, and total 2 lines in the LCD. This place holder has 16x2 LCD placed by default. You can replace it by 16X1 or 16X4 or any other pin compatible LCD module. Have a look at CKT 5 .

CKT 5



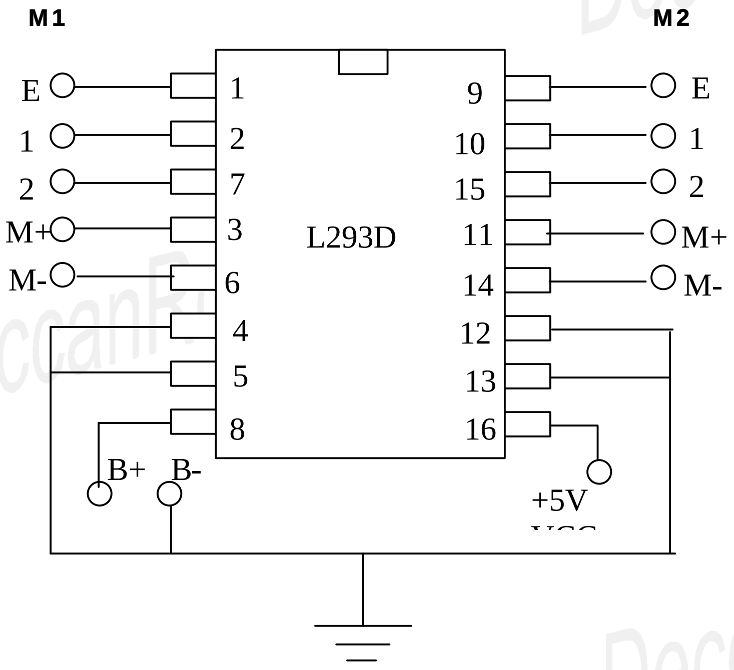
Section 14: L293D Based DC Motor Driver

One can connect 2 DC motors to this circuit. E=Enable, 1 & 2=Digital Input
M+ & M-=Connect your DC motor here, B+ & B- =Connect the required voltage to run DC motor e.g. 3V or 4.5V or 6V

If E is set to High and 1 & 2 are High and Low respectively, then motor connected between M+ and M- will rotate in one direction. If you change 1 & 2 to Low and High respectively keeping E at High, then motor will rotate in the reverse direction. If E is set to low then motor will not rotate. More details can be found under application section of the kit.

This is how the circuit of driver section is:

CKT 7

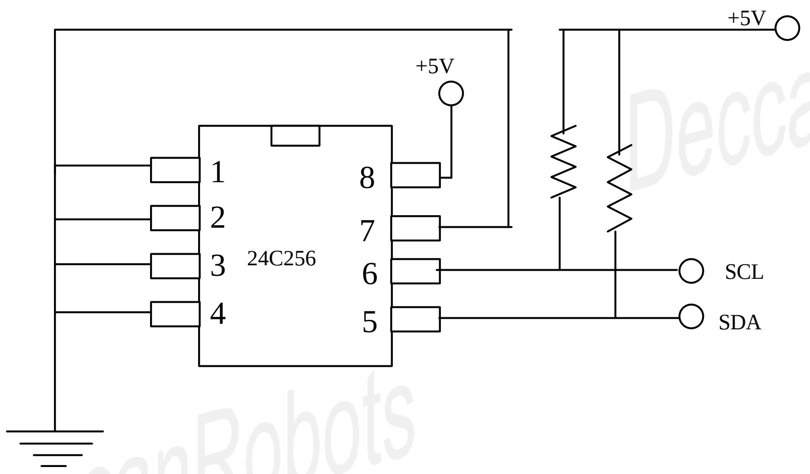


Section 15: 24C246 EEPROM

EEPROM is the memory IC that can retain the value once stored even if power is switched off. 24C256 is one of the EEPROM available in the market. Refer its datasheet from CD for more details about its internals.

This is how the circuit is arranged on EEDT PCB.

CKT 8



Section 4: 89S51 / S52 / S8253 / mega8515 target section

There are 3 target sections on the EEDT PCB. Section 4 can be used for placing 89S51 / S52 / S8253 or mega8515 microcontrollers. All port pins are taken out for connection purpose. There is no internal connection made between target sections and the interfacing sections. User has to make the connections as per the project requirement. There is not RESET circuit on the target board. The in-built programmer called HandyProg controls RESET. Looking at EEDT PCB, you can make out that all port pins are pulled out and connected to the 8 pin connectors. White color male 6 pin ISP connector has to be connected to HandyProg for programming purpose using a 6 Pin F-F multicolor cable.

Section 1: 8 Pin tiny AVRs (tiny13 and pin compatible tiny AVR controllers)
Section 2: 20 Pin tiny AVRs (tiny26 and pin compatible tiny AVR controllers)
Section 3: 20 Pin tiny AVRs (tiny2313 and pin compatible tiny AVR controllers)
Section 5: 28 Pin mega AVRs (mega8 and pin compatible AVR controllers)
Section 6: 40 Pin mega AVRs (mega16/32/8535 and pin compatible AVR controllers)
Section 7: 40 Pin mega AVRs (mega128 and pin compatible AVR controllers)

AVR Mega controllers do not require external crystal oscillators for their normal operation. AVR microcontroller has in-built R-C oscillator. But remember that this oscillator has very wide tolerance level. i.e. AVR's internal oscillator can not be used to generate stable frequency. If your application is time based then you must use external crystal oscillator. You can connect your crystal oscillator to XTAL pins provided on the PCB. White color male 6 pin ISP connector has to be connected to HandyProg for programming purpose using a 6 Pin F-F multicolor cable.

Section 18: HandyProg (inbuilt ISP Programmer)

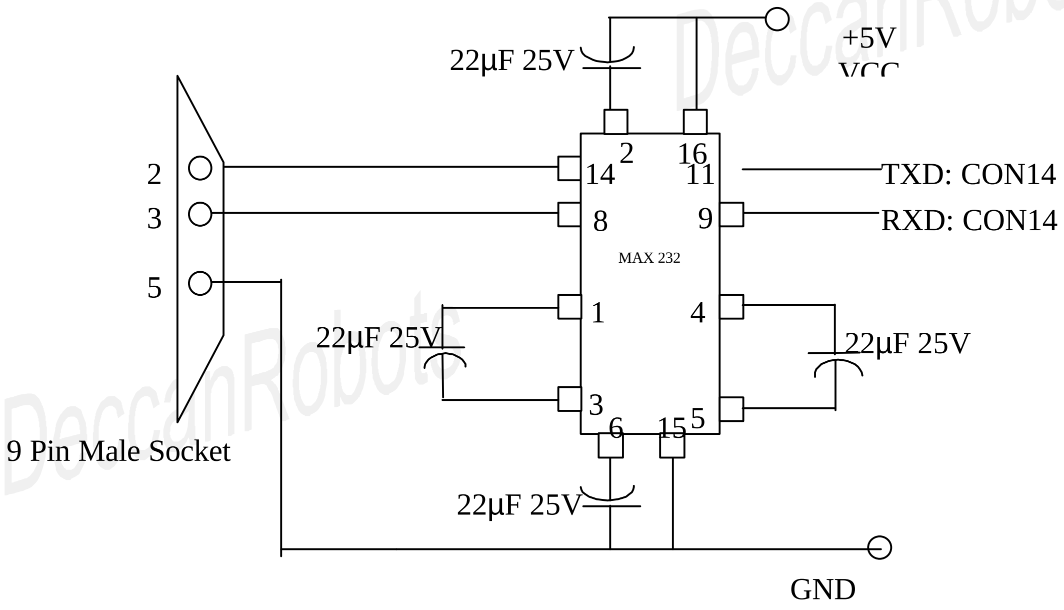
This is a ISP programmer hardware, which can program 89S, mega AVRs, tiny AVRs. You can use this section to program microcontrollers that are placed on the PCB or any microcontroller placed on your project PCB. You can take help of DeccanRobots' support team for external microcontroller programming connections.

Power Supply Section:

Use 12VDC-1000mA adaptor with center-positive male connector to supply power to the EEDT PCB. You will find a green LED glowing near to this connector, indicating that EEDT PCB has received required power. All sections on the EEDT PCB are connected to the Power Supply section. Hence you do not have to make power connections for individual sections. If you need a regulated 5V DC supply for some reason, then lookout for CON30 connected from bottom side of the PCB.

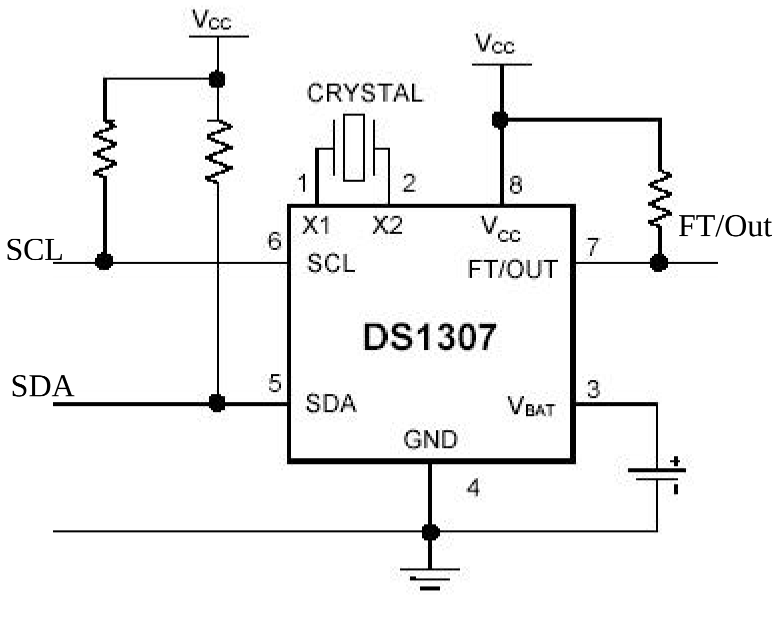
Section 14: RS232 Interface for your project application

Use the TX and RX pins from this section to connect to your microcontroller, and use the male 9 pin connector to connect to you PC, to establish your own RS232 connectivity for your project. You have to write appropriate source code for controller and PC.



Section 11: Real Time Clock using DS1307

Battery backed DS1307 is a I2C serial device which can remember date and time once set. You need to connect SCL and SDA lines to your controller for seting and retriving date/time. This is the interfacing circuit for RTC.

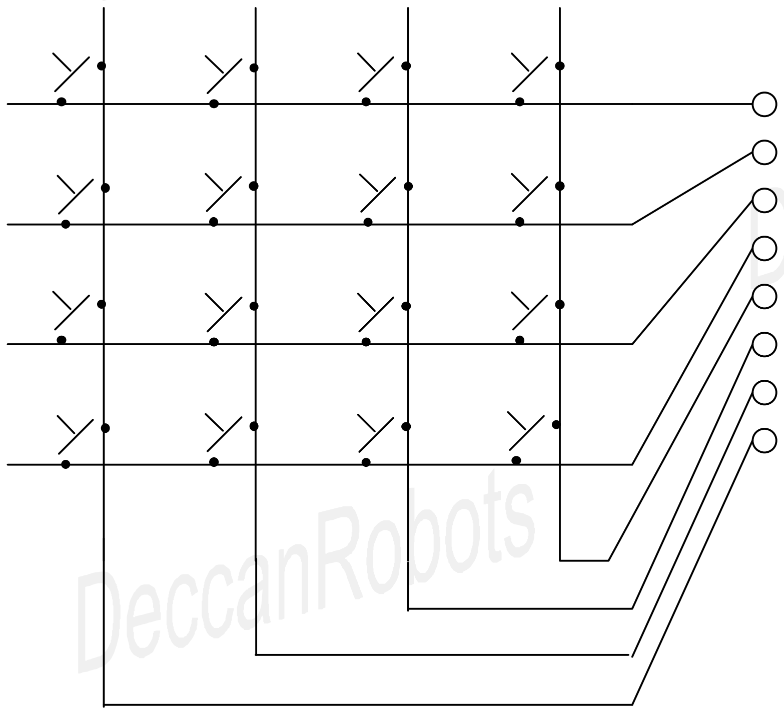


Section 17: 4x4 Keyboard Matrix

A matrix keyboard has 8 output lines, 4 from rows and 4 from columns.

Each key is connected in between one row line and one column line. each line is also pulled up by 1K resistance, which is not shown in the circuit. Notice that instead of 8 numbers of 1K resistances, we have used a compact resistance having set of 8 resistances in-built.

Refer the circuit diagram:



2. **8051 Architecture**

This chapter will teach you 8051 basics and will make you ready to write assembly code on your own. Read it carefully because each and every step of this chapter will be counted towards the success of your programming skills.

The 8051 have three types of memory. To effectively program the 8051 it is necessary to have a basic understanding of these memory types.

Code Memory

Code memory is the memory that holds the actual 8051 program that is to be run. Code memory for 89S52 is 8KB.

Internal RAM

Internal RAM is the memory, used to store values in registers, bit-addressable area and general-purpose area.

Special Function Register (SFR) Memory

Special Function Registers (SFRs) are areas of memory that control specific functionality of the 8051 processor.

! *As 8051 programmer, you need to understand details of Internal RAM and SFR (Memory).*

FAQ:

What do I need to know so that I can start developing projects / products based on 89S52?

Answer:

Knowledge of 89S52 Architecture i.e. Internal RAM, SFR Memory and Pin details

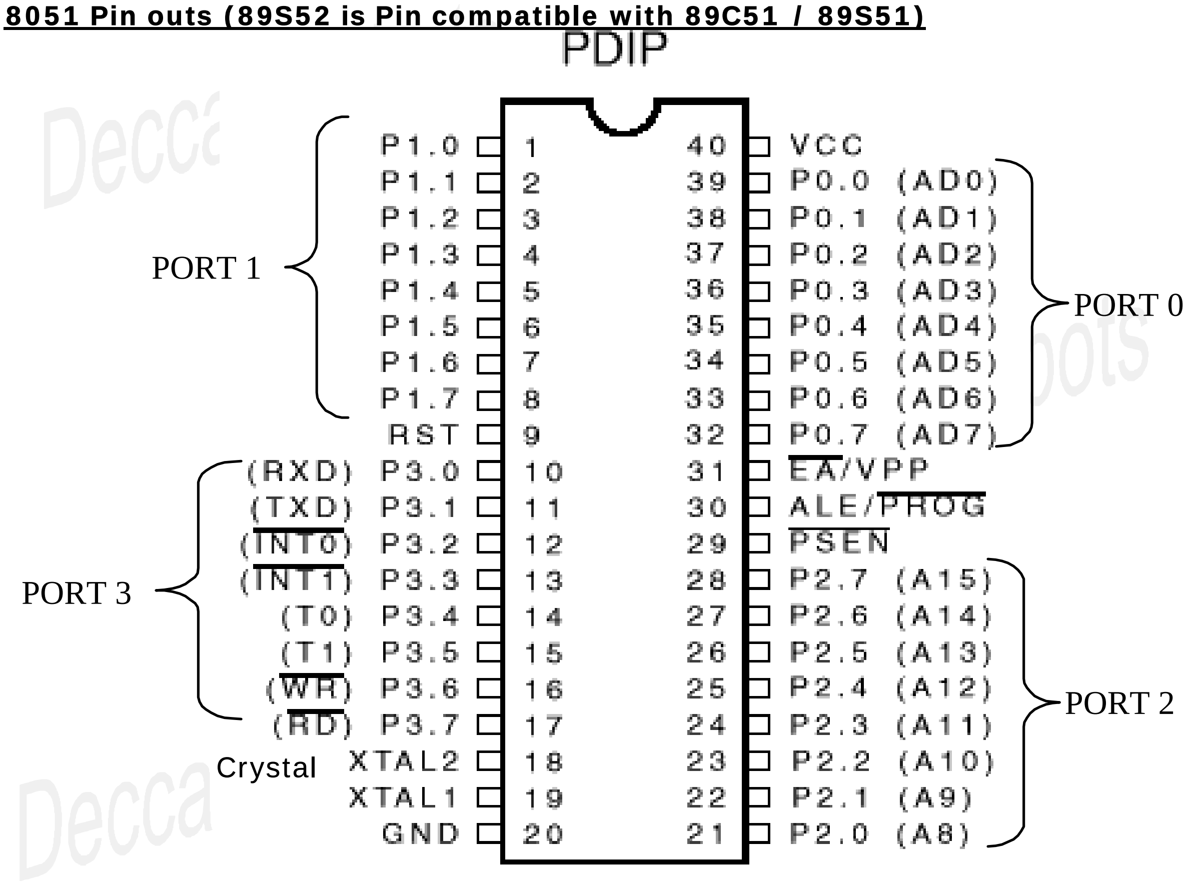
+

Knowledge of Instruction Set i.e. Assembly Code

+

Basics of Digital Electronics

8051 Pin outs (89S52 is Pin compatible with 89C51 / 89S51)

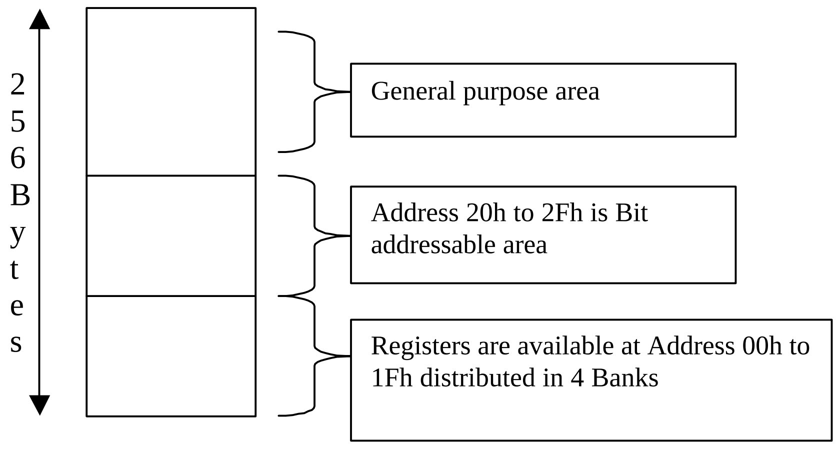


***!** Important to note:*

4 Input / Output Ports (Port 0, Port 1, Port 2, Port 3)
2 External Interrupts (Int0, Int1)
2 Timer/Counter (T0, T1)
1 Serial Port Connection (RXD, TXD)

89S52 CPU can process 8 bits at a time. Its internal RAM is of 256 bytes.

Internal RAM


256 Bytes
General purpose area
Address 20h to 2Fh is Bit addressable area
Registers are available at Address 00h to 1Fh distributed in 4 Banks

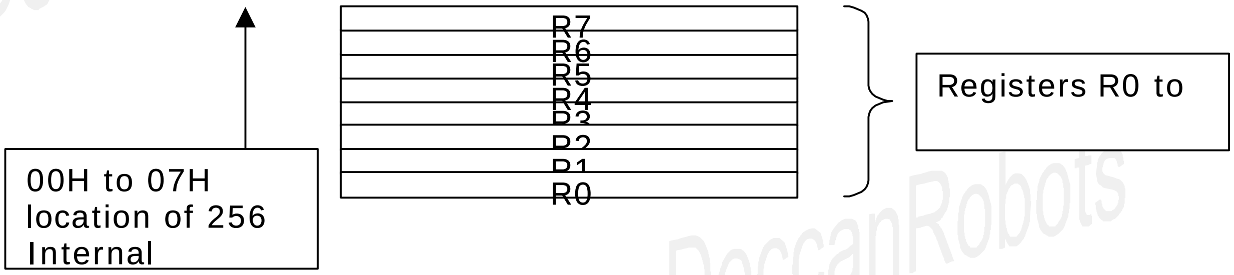
! *All addresses are generally specified in HEX format. But one can also express it in decimal format.*

FAQ:

I don't understand HEX values, Can you help me?
Answer: Yes, its simple to understand.
We use Decimal Numbering System, which is made up of numbers from 0 to 9 and its combinations. Think beyond 9. In Hexadecimal numbering system 9 is not the last number, next numbers of 9 are A, B, C, D, E, F.
e.g. 10 in decimal (denoted as 10d in this book) will be 0A in hex (denoted as 0Ah in this book).
Few more examples will clear your concept, Next number of 4Ah is 4Bh.
Previous number of 30h is 2Fh
4Ah + 30h = 7Ah
Remember that "h" stands for HEX, "d" stands for Decimal.

What is Bit Addressable Area in above diagram?
Answer: Bit addressable area is explained later in this chapter.

256 Byte Memory area starts with Address 00h.
Address 00h covers 8 bits i.e. 1 Byte. Address 00h is also called as R0, address 01h is also called as R1 and so on till address 07h is calld as R7.
These locations which are 1 Byte (8Bit) side are called as REGISTERS and called with their Nick names as seen above.



Special Function Register (SFR) Memory

What is SFR Memory?
These are the registers available to us for programming various aspects of 89S52. e.g. Serial data Transmission at 2400 Baud rate is possible only if we Set some specific values of related registers.
e.g. To generate one second time delay, we will need to set values of few registers like TMOD and TCON etc

In simple words, Blank (89S52 without any program in it) cannot perform any task on its own. We have to instruct it to behave in certain fashion by specifying values at Special Function registers (SFRs).

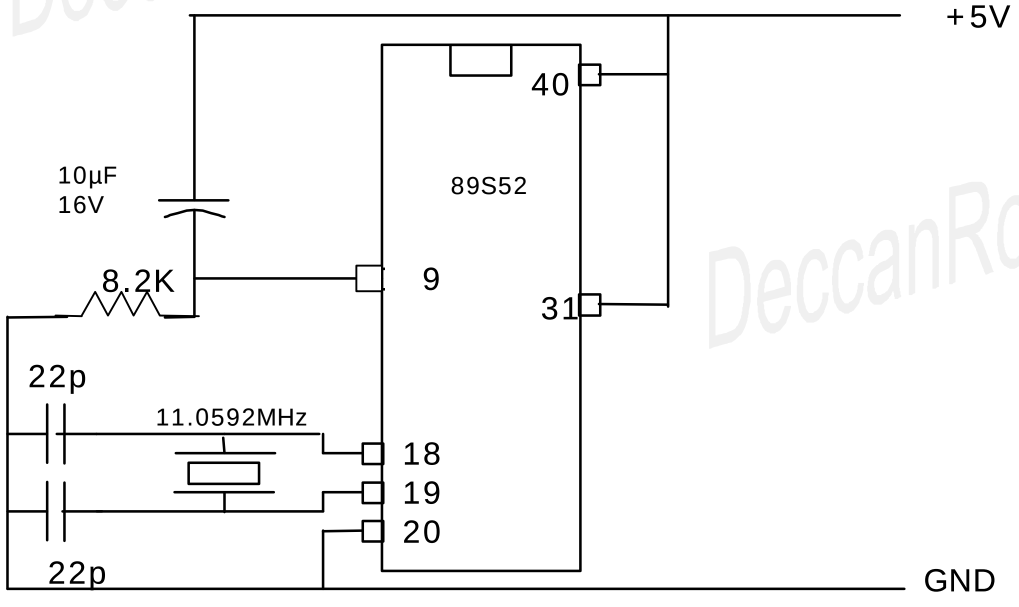
Is SFR Memory part of 256 Byte Internal RAM?
No. SFRs do not share Internal RAM memory.

How Many SFRs are there in 89S52?
Following List will describe important SFRs and their Addresses.

Name of Register	Address	**Name of Register**	Address
A	E0h	**TL0**	8Ah
B	F0h	**TH0**	8Ch
P0	80h	**TL1**	8Bh
P1	90h	**TH1**	8Dh
P2	A0h	**TMOD**	89h
P3	B0h	**TCON**	88h
SCON	98h	**IP**	B8h
SBUF	99h	**IE**	A8h
PSW	D0h	**SP**	81h
DPH	83h	**DPL**	82h

***!** By this time you are aware about Internal RAM and SFR memory area. May not be in detail but you have an overview by this time. Simply try to remember details given above, as we will need addresses and names of SFRs every time once we start writing codes for 89S52.*

<u>Minimum required Circuit Diagram for 89S52's normal working:</u>



You will not find the R-C circuit for RESET on the EEDT PCB.
HandyProg controls the RESET of the target circuit.

Make sure to add some RESET circuit in your own PCB, as during the real application of your project, HandyProg will not be present at client's site.

Your First Program for 89S52: Glowing LEDs using 89S52

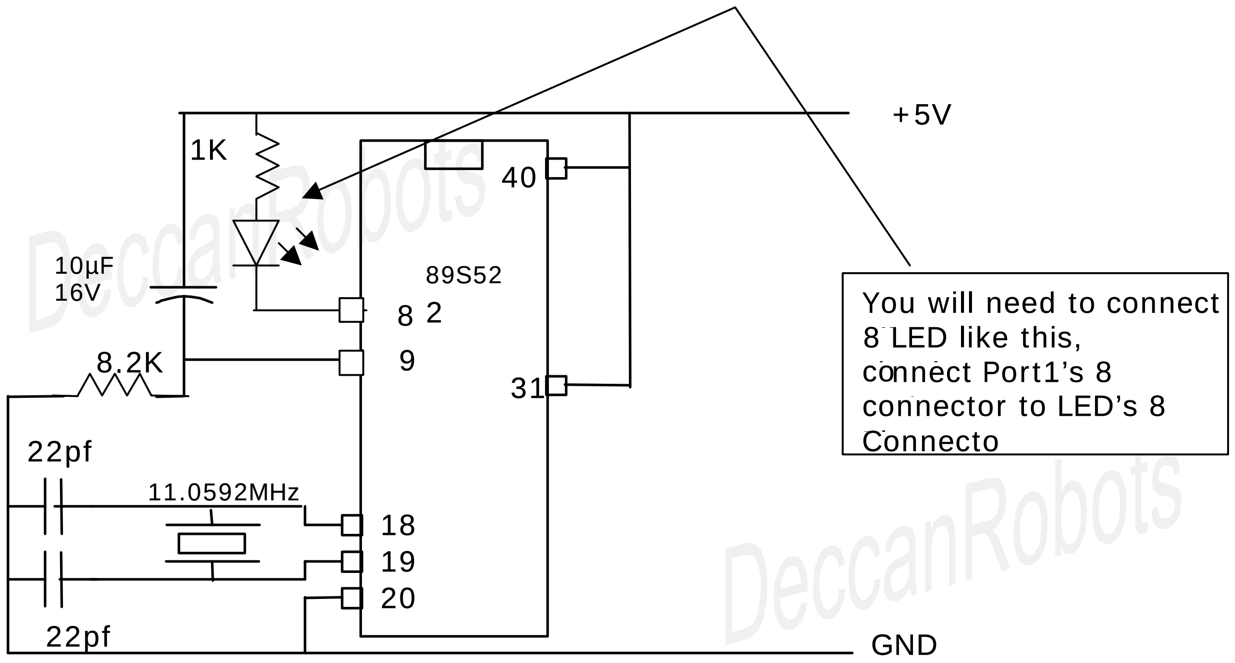
```
mov P1,#01010101b
```

Use " 8051 IDE" to write the above line of code. "8051 IDE" is provided in CD. If you have not yet installed it, get it installed before you proceed.

Save this program in a separate directory and assemble it using Menus available in Editor Software. Start "HandyProg Software" to download the HEX file of our program to 89S52.

Our one line program is coping some data to Port 1 of 89S52.

Binary value 0 will make LED ON, and binary value 1 will make LED OFF.


+5V
1K
40
89S52
10µF
16V
8 2
9
8.2K
31
22pf
11.0592MHz
18
19
20
22pf
GND
You will need to connect 8 LED like this, connect Port1's 8 connector to LED's 8 Connecto

I am sure that by this time you have started using the Editor Software, HandyProg Programmer Software and Embedded Engineer's Development Tool.

3. 8051 Instruction Set

Instruction set/ op-codes / code/ program are all similar words used by professionals working on 89C51 / 89S52 projects. In this chapter we will list out important Instruction set. I expect you should practice it immediately as you finish with one instruction set.

Bytes and Cycles specified at the end of each instruction set are not meant for those readers who are at beginners level. Bytes indicate the space occupied by this instruction set in Code Memory and Cycle indicates the time taken to execute the instruction set by 89C51 / 89S52.
1 Cycle = 1 / (Crystal frequency in Mhz /12) Microseconds
We have used Crystal of 11.0592 Mhz Frequency.
Hence 1 Cycle= 1.085 microseconds

Commands to copy data

MOV A,Rn
Function: Move Register's value to Accumulator
Bytes: 1 ***Cycles:*** 1
e.g. MOV A,R0

MOV A,direct
Function: Move Direct address's value to Accumulator
Bytes: 2 ***Cycles:*** 1
e.g. MOV A,00h
here 00h is the address, and the instruction set will copy value from 00h location to register A. Remember that 00h address is also called as R0

MOV A,#data
Function: Move value (#data) to Accumulator
Bytes: 2 ***Cycles:*** 1
e.g. MOV A,#23h
here value 23h i.e. 00100011b will be copied to register A

MOV Rn,A
Function: Move Accumulator's value r to Register
affected.
Bytes: 1 ***Cycles:*** 1
e.g. MOV R1,A

MOV Rn,direct
Function: Move Direct to Register
Bytes: 2 ***Cycles:*** 2
e.g. MOV R0,05h
here value from location 05h will be copied to R0

MOV Rn,#data
Function: Move Immediate value to Register
Bytes: 2 ***Cycles:*** 1
e.g. MOV R0,#34h
here value 34h will be copied to R0

MOV direct,A
Function: Move Accumulator to Direct Memory
Bytes: 2 **Cycles: 1**
e.g. MOV 20h,A
here value of register A will be copied to the location 20h of Internal RAM

MOV direct,Rn
Function: Move Register to Direct Memory
Bytes: 2 **Cycles: 2**
e.g. MOV 30h,R0
here value of R0 will be copied to location 30h of internal RAM

MOV direct,direct
Function: Move Direct Memory to Direct Memory
Bytes: 3 **Cycles: 2**
e.g. MOV 20h,30h
here value from 30h location will be copied to location 20h

MOV direct,#data
Function: Move Immediate to Direct Memory
Bytes: 3 ***Cycles:*** 2
e.g. MOV 30h, #75d
here value 75d (75 decimal) will be copied at location 30h

! Use # to specify its VALUE and NOT address. MOV statement is used to copy contents of right hand operand to left hand operand. If we use name of register i.e. R0 in instruction set then it is called as we are using Registering addressing Mode, if we use 00h address in instruction set then it is said that we are using direct addressing mode. If we use value in form of #34h, then it is called as we are using Immediate-addressing mode. One more addressing mode is left out, which is called as Indirect addressing mode, lets understand it now.

MOV @Ri,direct
Function: Move Direct Memory to Indirect Memory
Bytes: 2 ***Cycles:*** 2
e.g.
```
MOV 20h,#0Fh
MOV R0,#04h
MOV @R0,20h
```

Understand this line by line.
In first line of code we are coping value #0Fh to 20h location.
In second line we have copied value #04h to R0 location
In third line we are copying values of 20h location to the indirect location specified by R0. Here R0 is indirectly pointing to 04h memory location. Hence the last line will copy value from 20h location to 04h location.
Similar Indirect addressing mode instruction sets are listed below:

MOV @Ri,A
Function: Move Accumulator to Indirect Memory
Bytes: 1 ***Cycles:*** 1

MOV @Ri,#data
Function: Move Immediate to Indirect Memory
Bytes: 2 ***Cycles:*** 1

Next 2 commands are PUSH and POP. We will be using these commands while learning Interrupts.

PUSH direct
Function: Push onto stack
Bytes: 2 ***Cycles:*** 2

POP direct
Function: Pop from stack
Bytes: 2 ***Cycles:*** 2

e.g.

```
MOV SP,#50h
MOV R0,#12h
PUSH 00h
PUSH 00h
POP 03h
```

Understand it line by line.

First line is copying value #50h to SP.

SP is stack pointer which stores destination address of PUSHed or POPed data.

Second line is copying value #12h to location R0

Third line will increment the value of SP register by one and PUSH the data, i.e. copy the data from 00h location to the location pointed by SP.

In this case SP was having value 50h, which will be incremented by one and will be 51h, hence the data from 00h location will be copied to 51h location.

Fourth line will again increment SP by one, making it point to 52h and then end up copying the value from 00h to 52h.

Remember that always data will be copied from the location specified in PUSH statement like 00h in this case.

The last line is POP line, and it will copy data from 52h location to 03h location, as currently SP is point to 52h.

SP will be decremented after copying the data by one. Hence SP will become 51h. POP statement always dumps data to the location specified in POP statement.

Data Exchange Commands

XCH A,Rn
Function: Exchange Accumulator's value with Register's value
Bytes: 1 ***Cycles:*** 1

XCH A,direct
Function: Exchange Accumulator's value with Direct Memory's value
Bytes: 2 ***Cycles:*** 1

XCH A,@Ri
Function: Exchange Accumulator' value with value at the address specified by Ri
Bytes: 1 ***Cycles:*** 1

XCHD A,@Ri
Function: Exchange lower nibbles. A nibble is half of byte. i.e. 4 bits.
Bytes: 1 ***Cycles:*** 1

Logical Operation Commands

ANL A,Rn
Function: Logical-AND accumulator with register
Bytes: 1 ***Cycles:*** 1

ANL A,direct
Function: Logical-AND accumulator with direct memory
Bytes: 2 ***Cycles:*** 1

ANL A,@Ri
Function: Logical-AND accumulator with indirect memory
Bytes: 1 ***Cycles:*** 1

ANL A,#data
Function: Logical-AND accumulator with immediate data
Bytes: 2 ***Cycles:*** 1

ANL direct,A
Function: Logical-AND direct memory with accumulator
Bytes: 2 ***Cycles:*** 1

ANL direct,#data
Function: Logical-AND direct memory with immediate data
Bytes: 3 ***Cycles:*** 2

ORL A,Rn
Function: Logical-OR accumulator with register
Bytes: 1 ***Cycles:*** 1

ORL A,direct
Function: Logical-OR accumulator with direct memory
Bytes: 2 ***Cycles:*** 1

ORL A,@Ri
Function: Logical-OR accumulator with indirect memory
Bytes: 1 ***Cycles:*** 1

ORL A,#data
Function: Logical-OR accumulator with immediate data
Bytes: 2 ***Cycles:*** 1

ORL direct,A
Function: Logical-OR direct memory with accumulator
Bytes: 2 ***Cycles:*** 1

ORL direct,#data
Function: Logical-OR direct memory with immediate data
Bytes: 3 ***Cycles:*** 2

XRL A,Rn
Function: Logical Exclusive-OR Accumulator with register
Bytes: 1 ***Cycles:*** 1

XRL A,direct
Function: Logical Exclusive-OR Accumulator with direct memory
Bytes: 2 ***Cycles:*** 1

XRL A,@Ri
Function: Logical Exclusive-OR Accumulator with indirect memory
Bytes: 1 ***Cycles:*** 1

XRL A,#data
Function: Logical Exclusive-OR Accumulator with immediate data
Bytes: 2 ***Cycles:*** 1

XRL direct,A
Function: Logical Exclusive-OR direct memory with accumulator
Bytes: 2 ***Cycles:*** 1

XRL direct,#data
Function: Logical Exclusive-OR direct memory with immediate data
Bytes: 3 ***Cycles:*** 2

CPL A
Function: Complement Accumulator
Bytes: 1 ***Cycles:*** 1

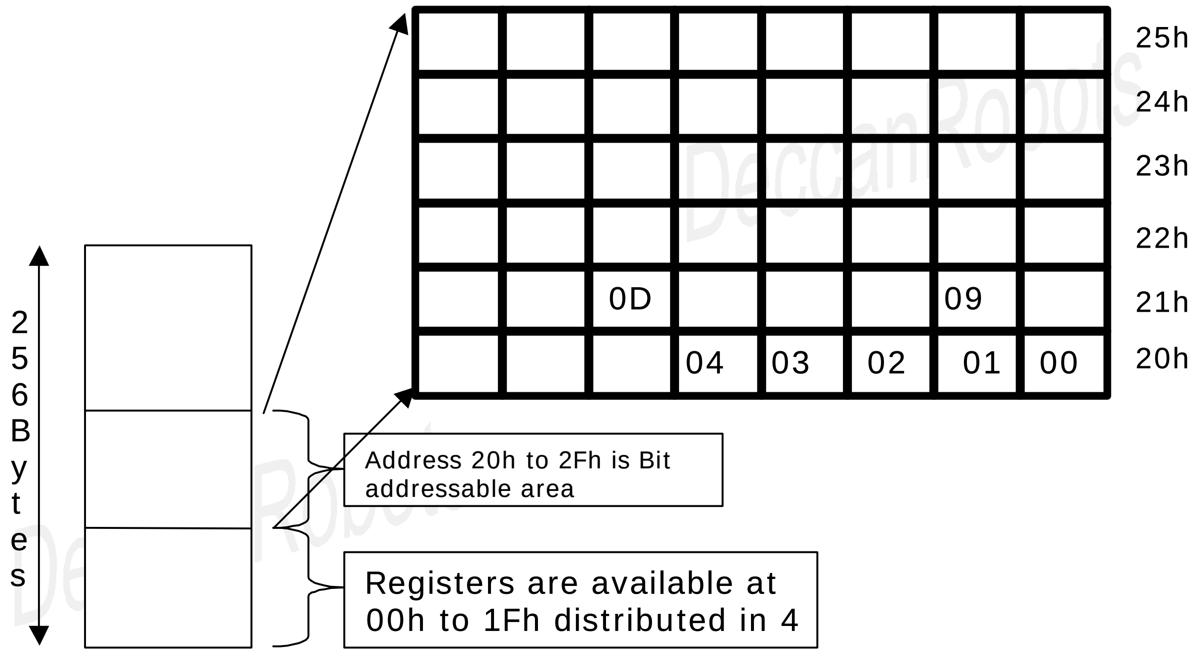
CLR A
Function: Clear Accumulator
Bytes: 1 ***Cycles:*** 1

If you are familiar with C programming or any other programming, then you must be waiting for if.. else, statements, For Loop, While Loop etc. Assembly language does not have any such kind of statements. Using above mentioned logical operators we can write decision-making statements and Loops with the help of jumps and calls, which will be explained in sometime from now.

All the above-mentioned commands are used to manipulate BYTE size data.
Internal RAM has a section called as Bit-Addressable Area. Lets understand this memory area and commands related to manipulation of bits.

Bit Addressable Area

Memory location from 20h to 2Fh of Internal RAM is called as Bit-Addressable.
There are 15 Bytes in this area. Each bit of every byte from this area is NAMED uniquely and can be accessed using some commands.
Same time Register R0 to R7 are not bit-addressable, i.e. individual bit of R0 cannot be accessed using any command.


25h
24h
23h
22h
21h
20h
0D
09
04
03
02
01
00
256Bytes
Address 20h to 2Fh is Bit addressable area
Registers are available at 00h to 1Fh distributed in 4

Referring to this figure, you can understand that 0th bit of 20h location is called as 00h, 4th bit of 20h is called as 04h. Similarly 1st Bit of 21h location is named as 09h.

Using this information can you locate 39d bit in bit addressable area?

While developing any project using 89S52, you may not need to use and engage full byte. e,g. remembering status of Motors and relays requires one BIT to store whether connected device is in ON or OFF state.

MOV statement can be used to copy data from one location to other bit location

```
MOV C, P1.0
MOV 00h, C
```

! 7th Bit of PSW register is called as Carry or simply "C"

Above MOV statement will copy ONE bit from P1.0 location to C
Second line in above code will copy ONE bit from C to location 00h, i.e. 0th bit of 20h
One may mistake in understanding the second line. In second line 00h is NOT R0, but it is 0th bit of 20h location which is bit-addressable area in internal RAM.
Remember always that data copies between Locations with SAME SIZE, a bit s on both sides or bytes on both sides.

Commands related to Bit operations

MOV C,bit
Function: Move Bit to Carry
Bytes: 2 ***Cycles:*** 1

MOV bit,C
Function: Move Carry to Bit
Bytes: 2 ***Cycles:*** 2

ANL C,bit
Function: Logical-AND Carry flag with bit value
Bytes: 2 ***Cycles:*** 2

ANL C,/bit
Function: Logical-AND Carry flag with complement of bit value
Bytes: 2 ***Cycles:*** 2

ORL C,bit
Function: Logical-OR Carry with bit variable
Bytes: 2 ***Cycles:*** 2

ORL C,/bit
Function: Logical-OR Carry with complement of bit variable
Bytes: 2 ***Cycles:*** 2

SETB C
Function: Set Carry
Bytes: 1 ***Cycles:*** 1

SETB bit
Function: Set Bit
Bytes: 2 ***Cycles:*** 1

CPL C
Function: Complement Carry
Bytes: 1 ***Cycles:*** 1

CPL bit
Function: Complement bit location
Bytes: 2 ***Cycles:*** 1

CLR C
Function: Clear Carry flag
Bytes: 1
Cycles: 1

CLR bit
Function: Clear bit location
Bytes: 2 ***Cycles:*** 1

Some of SFRs are also Bit addressable. i.e. we can access a bit of SFR like we already have accessed P1.0 List of these SFR is:
A, B, IE, IP, P0, P1, P2, P3, PSW, TCON, SCON

To access bit of register A use ACC.3 format, to access a bit from TCON then use TCON.3s

Rotate and Swap Commands

RL A
Function: Rotate Accumulator's value to Left
Bytes: 1 ***Cycles:*** 1

RLC A
Function: Rotate Accumulator's value Left through the Carry flag
Bytes: 1 ***Cycles:*** 1

RR A
Function: Rotate Accumulator's value Right
Bytes: 1 ***Cycles:*** 1

RRC A
Function: Rotate Accumulator's value Right through the Carry flag
Bytes: 1 ***Cycles:*** 1

Increment, decrement and miscellaneous commands

INC A
Function: Increment Accumulator's value
Bytes: 1 ***Cycles:*** 1

INC Rn
Function: Increment Register's value
Bytes: 1 ***Cycles:*** 1

INC direct
Function: Increment Direct Memory's value
Bytes: 2 ***Cycles:*** 1

INC @Ri
Function: Increment Indirect Memory's value
Bytes: 1 ***Cycles:*** 1

INC DPTR
Function: Increment Data Pointer
Bytes: 1 ***Cycles:*** 2

DEC A
Function: Decrement Accumulator
Bytes: 1 ***Cycles:*** 1

DEC Rn
Function: Decrement Register
Bytes: 1 ***Cycles:*** 1

DEC direct
Function: Decrement Direct Memory
Bytes: 2 ***Cycles:*** 1

DEC @Ri
Function: Decrement Indirect Memory
Bytes: 1 ***Cycles:*** 1

NOP
Function: No Operation
Bytes: 1 ***Cycles:*** 1

Jumps and Calls Commands

When an 89S52 is first initialized, it resets the PC(Program Counter) to 0000h. The 89S52 then begin to execute instructions sequentially in memory unless a program instruction causes the PC to be otherwise altered. There are various instructions that can modify the value of the PC; specifically, conditional branching instructions, direct jumps and calls, and "returns" from subroutines. Additionally, interrupts, when enabled, can cause the program flow to deviate from it's otherwise sequential scheme.

Conditional Branching

The 89S52 contains a suite of instructions which, as a group, are referred to as "conditional branching" instructions. These instructions cause program execution to follow a non-sequential path if a certain condition is true.

Take, for example, the JB instruction. This instruction means "Jump if Bit Set." An example of the JB instruction might be:

```
        JB 45h,HELLO
        NOP
HELLO:  ....
```

In this case, the 8051 will analyze the contents of bit 45h. If the bit is set program execution will jump immediately to the label HELLO, skipping the NOP instruction. If the bit is not set the conditional branch fails and program execution continues, as usual, with the NOP instruction, which follows.

Conditional branching is really the fundamental building block of program logic since all "decisions" are accomplished by using conditional branching. Conditional branching can be thought of as the "IF...THEN" structure in 8051 assembly language.

An important note worth mentioning about conditional branching is that the program may only branch to instructions located within 128 bytes prior to or 127 bytes following the address, which follows the conditional branch instruction. This means that in the above example the label HELLO must be within +/- 128 bytes of the memory address, which contains the conditional branching instruction.

Direct Jumps

While conditional branching is extremely important, it is often necessary to make a direct branch to a given memory location without basing it on a given logical decision. This is equivalent to saying "Goto" in BASIC. In this case you want the program flow to continue at a given memory address without considering any conditions.

This is accomplished in the 8051 using "Direct Jump and Call" instructions. As illustrated in the last paragraph, this suite of instructions causes program flow to change unconditionally.

Consider the example:

```
                    LJMP NEW_ADDRESS
                    .
                    .
                    .
      NEW_ADDRESS:  ....
```

The LJMP instruction in this example means "Long Jump." When the 8051 executes this instruction the PC is loaded with the address of NEW_ADDRESS and program execution continues sequentially from there.

The obvious difference between the Direct Jump and Call instructions and the conditional branching is that with Direct Jumps and Calls program flow always changes. With conditional branching program flow only changes if a certain condition is true.

It is worth mentioning that, aside from LJMP, there are two other instructions which cause a direct jump to occur: the SJMP and AJMP commands. Functionally, these two commands perform the exact same function as the LJMP command--that is to say, they always cause program flow to continue at the address indicated by the command. However, SJMP and AJMP differ in the following ways:

- The SJMP command, like the conditional branching instructions, can only jump to an address within +/- 128 bytes of the SJMP command.
- The AJMP command can only jump to an address that is in the same 2k block of memory as the AJMP command. That is to say, if the AJMP command is at code memory location 650h, it can only do a jump to addresses 0000h through 07FFh (0 through 2047, decimal).

You may be asking yourself, "Why would I want to use the SJMP or AJMP command which have restrictions as to how far they can jump if they do the same thing as the LJMP command which can jump anywhere in memory?" The answer is simple: The LJMP command requires three bytes of code memory whereas both the SJMP and AJMP commands require only two. Thus, if you are developing an application that has memory restrictions you can often save quite a bit of memory using the 2-byte AJMP/SJMP instructions instead of the 3-byte instruction.

Recently, I wrote a program that required 2100 bytes of memory but I had a memory restriction of 2k (2048 bytes). I did a search/replace changing all LJMPs to AJMPs and the program shrunk downto 1950 bytes. Thus, without changing any logic whatsoever in my program I saved 150 bytes and was able to meet my 2048 byte memory restriction.

Direct Calls

Calls are similar to Jumps except that the control comes back to the next line.

Consider the example:

```
             LCALL NEW_ADDRESS
             MOV A,R0
             .
             .
NEW_ADDRESS:
             ..
             ..
             ..
             ..
             RET
```

Program Will jump to New_Address location, will execute the subroutine and return to MOV statement as it encounters RET in subroutine.
In simple words, CALLS are temporary branching of program flow.

Syntax for Jumps:
Unconditional Jumps

SJMP rel
Function: Short Jump
Bytes: 2 ***Cycles:*** 2

AJMP addr11
Function: Absolute Jump
Bytes: 2 ***Cycles:*** 2

LJMP addr16
Function: Long Jump
Bytes: 3 ***Cycles:*** 2

Conditional jumps

You will find word “rel” in following commands. It means Label or name of Subroutine e.g.

```
CJNE A,34h,Loop
..
..
..

Loop:
..
..
..
```

Here Loop is label or referred as “rel” in following syntax.

CJNE A,direct,rel
Function: Compare accumulator to direct memory and Jump if Not Equal
Bytes: 3 ***Cycles:*** 2

CJNE A,#data,rel
Function: Compare accumulator to immediate data and Jump if Not Equal
Bytes: 3 ***Cycles:*** 2

CJNE Rn,#data,rel
Function: Compare register value to immediate data and Jump if Not Equal
Bytes: 3 ***Cycles:*** 2

CJNE @Ri,#data,rel
Function: Compare indirect memory with immediate data and Jump if Not Equal
Bytes: 3 ***Cycles:*** 2

DJNZ Rn,rel
Function: Decrement Register and Jump if Not Zero
Bytes: 2 ***Cycles:*** 2

DJNZ direct,rel
Function: Decrement Direct Memory and Jump if Not Zero
Bytes: 3 ***Cycles:*** 2

JNZ rel
Function: Jump if Accumulator Not Zero
Bytes: 2 ***Cycles:*** 2

JZ rel
Function: Jump if Accumulator Zero
Bytes: 2 ***Cycles:*** 2

Conditional Jumps (Bit wise)

JB bit,rel
Function: Jump if Bit set
Bytes: 3 ***Cycles:*** 2

JBC bit,rel
Function: Jump if Bit is set and Clear bit
Bytes: 3 ***Cycles:*** 2

JC rel
Function: Jump if Carry is set
Bytes: 2 ***Cycles:*** 2

JNB bit,rel
Function: Jump if Bit Not set
Bytes: 3 ***Cycles:*** 2

JNC rel
Function: Jump if Carry not set
Bytes: 2 ***Cycles:*** 2

<u>Call Commands</u>

ACALL addr
Function: Absolute Call (Execute code section denoted by addr and come back)
Bytes: 2 ***Cycles:*** 2

LCALL addr16
Function: Long call (Execute code section denoted by addr and come back)
Bytes: 3 ***Cycles:*** 2

4. Applications (For 89S)

Note:

Refer **How to use Interfacing Sections?** part from this book while working with the applications.

To avoid redundancy, we have not drawn the complete circuit diagram as each interfacing section, as it is well described in the "How to use Interfacing Sections?" part of this book.

Use multicolor wires to connected between 89S target controller and interfacing section.

App 1: Glowing LED

Connections: (Refer sectional circuit diagram in "Interfacing Section Details")
Connect 8 LEDs to Port2 of the 89S52 using 8-pin F-F multi color cable.

Description:
Our one line program is coping some data to Port 2 of 89S52.
LEDs will glow on all lines that are having data 0
Rest of the lines having 1 as data which will keep LEDs in OFF state.

Source Code:

```
Loop:
mov P2,#01010101b
Sjmp Loop
```

Explanation:
Connect 8 LEDs to Port 2. LEDs will be continuously ON or OFF depending on value (0 or 1) is applied on a particular port pin. LEDs will glow if corresponding port pin is LOW or 0 because LEDs are connected to VCC via resistance and will get ground connectivity via microcontroller's port pin. Similarly LEDs will remain OFF if corresponding pin is set to HIGH or 1.

App 2: Toggle LED using switch

Connections:
Refer Section 3 and Section 4 description from "Interfacing Section Details"
Section 3 has 8 LEDs.
Section 4 has 6 Switches.

Connect a 4 Pin F-F multicolor connector between CON6 of Section 4 AND Port 2 of 89S52. Make sure that you are connecting the 4 Pins to the lower nibble i.e. P2.0 to P2.3. While writing code, we will be using only P2.0 . Rest 3 connection will remain un-used.

Connect a 2 pin F-F multicolor connector between P1.0-P1.1 AND any 2 LEDs (CON 5's any 2 pins)

This way we have connected one pull up switch to P2.0 and one LED to P1.0.Even though there are connection for 3 more switches and one more LED, we will not be using them now.

Source Code:

```
SETB P1.0
Loop:
        JB P2.0, Loop
        CPL P1.0
        Debounce:
                JNB P2.0, Debounce
        SJMP Loop
```

Explanation:
Initially P1.0 is set to high so that LED will be in OFF state. Inside label "Loop", if P2.0 is in high state then program will jump to Loop label again for reading switch status again. This looping or in other words polling will continue till the switch is in open state. The moment switch is pressed, P2.0 will receive a negative pulse, and program will continue to next line to complement the P1.0 where we have connected LED. Debounce is label provided to check the switch status again. LED should not flicker as user may take part of a second to release the switch. Hence we have to check that the switch must be released before changing status of LED. If P2.0 is still LOW then program will jump to label Debounce. This will loop till switch is in pressed condition. Once the switch is released then, P2.0 will receive a high state and program will continue to the last line and finally will jump to Loop label to again monitor the switch.

App 3: Rotate Stepper Motor

Connections:

Refer Section 2 description from "Interfacing Section Details"
Connect a 4 Pin F-F connector between P2's lower nibble (P2.0 to P2.3) and Input of ULN2803 based high current driver section. Connect another 4 Pin F-F multicolor connector between output of ULN2803 and stepper motor. Connect Stepper motor's 5th wire or in some motors 5th and 6th wire to the rated voltage e.g. 9V or 12V. 12V Dc is available for you on EEDT PCB from the green MKDSN connector placed near the power supply section.
You may write your stepper motor related queries to support@deccanrobots.com

As we will be rotating the stepper motor using a switch, you have to connect one pulled-up switch to P1.0. Read App 2 if you don't remember the process to connect a switch to P1.0.

Source Code:

```
    mov P2,#00h
    mov A,#10001000b
    Start:
        MOV C, P1.0
        JC Start
        mov P2,A
        RR A
  ChkForSwitchRelease:
    MOV C, P1.0
    JNC ChkForSwitchRelease
    SJMP Start
```

Explanation:

Initially P2 is set to low so that Stepper Motor will be in OFF state.

Remember that ULN 2803 is connected to Port 2, and ULN 2803 requires High signal to switch ON the stepper motor.

Register A is initialized to value 10001000 binary, which will be used to energies one pole of stepper motor at a time.

Inside label "Start", data from P1.0 is copied to PSW's 7th bit called as "C" to read the status of switch. If C is in high state then program will jump to Start label again for reading switch status again. This looping or in other words polling will continue till the switch is in open state.

The moment switch is pressed, P1.0 will receive a negative pulse, which will be copied to C, and program will continue to next line to copy Values from A to Port 2 where we have connected ULN2803.

At this state, P2's 3rd and 7th bit will be high. Referring to connection details, we can find that 7th bit of Port 2 is unused. In this condition P2.0, P2.1, P2.2 will be at low state and P2.3 will be at high state. Hence only one pole of stepper motor will be ON and motor will move one step ahead either clockwise or anti clockwise based on your connection. Next statement will Rotate value of A register and A's value will be 01000100 binary. This value will be useful when switch is pressed next time. ChkForSwitchRelease is label provided to check the switch status again. Stepper Motor should not get jerk/flicker as user may take part of second to release the switch. Hence we have to check that the switch must be released before changing status of LED. For this work, We have copied P1.0 to C and if C is still LOW then program will jump to label ChkForSwitchRelease. This will loop till switch is in pressed condition. Once the switch is released then, C will receive a high state and program will continue to the last line and finally will jump to Start label to again monitor the switch.

Try pressing switch again and again, you will find that motor is moving in one direction. If you find that motor is oscillating between 2 places instead of rotating, then you need to interchange the stepper motor's wires connected to ULN2803. This happens if poles are not connected in required sequence. Try various combinations to make motor running.

App 4: Interfacing 16X2 LCD

Connections:

Make connections as described in the following table:

LCD Pin Number	89S52 Pin / Port Number
4(RS)	P3.5
5(RW)	P3.6
6(EN)	P3.7
7 to 14(D0 to D7)	P2.0 to P2.7 Respectively

Refer section 5 of "Interfacing Section Details" from this book.

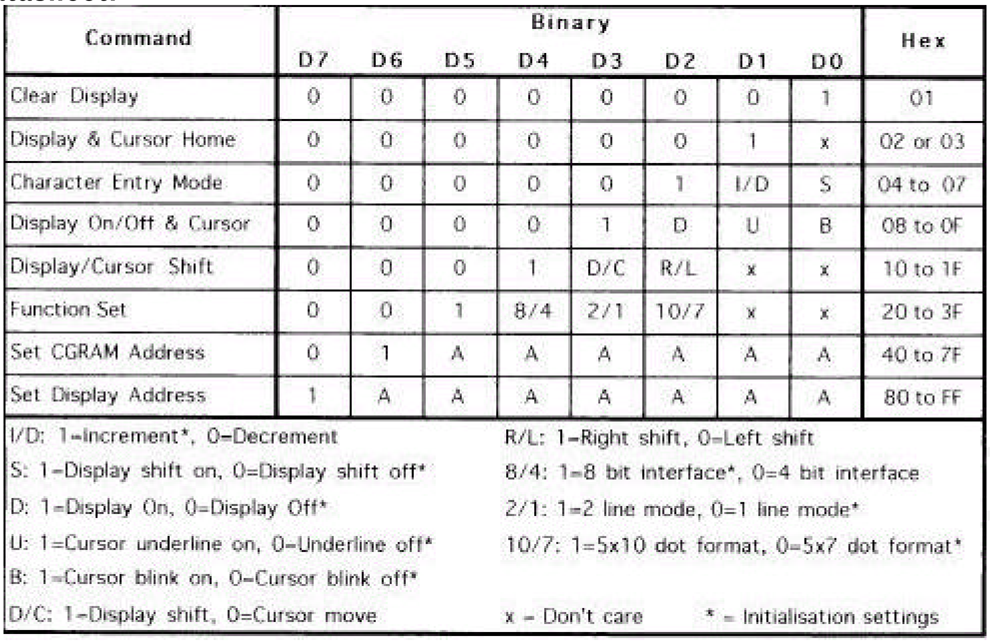
Source Code:

Refer source code from the CD provided along with EEDT.
CD\Project Source Code\89S\Lcd Interfacing

Explanation:

Refer 89S52's self learning tutorial for source code explanation.

LCD Datasheet:



Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D: 1=Increment*, 0=Decrement	R/L: 1=Right shift, 0=Left shift
S: 1=Display shift on, 0=Display shift off*	8/4: 1=8 bit interface*, 0=4 bit interface
D: 1=Display On, 0=Display Off*	2/1: 1=2 line mode, 0=1 line mode*
U: 1=Cursor underline on, 0=Underline off*	10/7: 1=5x10 dot format, 0=5x7 dot format*
B: 1=Cursor blink on, 0=Cursor blink off*	
D/C: 1=Display shift, 0=Cursor move	x = Don't care * = Initialisation settings

App 5: 16 Bit simple timer to blink LED every one-second

(If you are new to microcontroller then you must understand the concept of TIMERS using E-Learning Tutorial for 89S available for installation from CD)

Connections:

Connect one LED to P1.0. Refer App 1 is you are not sure as how to connect it.

Source Code:

```
 Start:
MOV R0,#14d
MOV TMOD,#01h
TSTART:
        SETB TR0
                Loop:
                       JNB TF0, Loop
                       CLR TF0
                       CLR TR0
                       DJNZ R0, TSTART
                       MOV R0, #14d
                       CPL P1.0
         SJMP TSTART
```

Explanation:

As you program the controller, LED Connected to P1.0 will start blinking. It will be ON for 1 sec and OFF for 1 second.

This application is using Timer 0 in Mode 1 configuration, which means 16 Bit timer.

For this configuration TMOD's value will be 00000001 binary.

This timer is independent of external pin or counter.

Setb TR0 will start Timer 0.

T0 values i.e. TL0 and TH0 vales will be incremented every machine cycle.

One machine cycle time =1/(11.0592/12) =1.085 Micro Seconds as we have used 11.0592Mhz crystal oscillator. Have a look at PCB.

After incrementing 65536 times, TF0 flag will be set to high to indicate the Timer 0 overflow.

This overflow will happen in 1.085 X 65536=71106 Micro Seconds = 71 Miliseconds.

For one second, we require approximate 14 overflows like this.

Hence Register R0 is used to count 14 decimal values and clear to 00h when counting gets over. This process will continue forever till power is ON.

This will make LED blink every one-second.

App 6: 16 Bit simple timer controlled by external pulse

Connections:

Connect one LED to P1.0
Connect a pulled down switch to P3.2, which is also named as INT0 in the datasheet.

Source Code:

```
 START:
MOV R0,#14D
MOV TMOD,#00001001B
TSTART:
        SETB TR0
        LOOP:
              JNB TF0,LOOP
              CLR TF0
              CLR TR0
              DJNZ R0,TSTART
              MOV R0,#14D
              CPL P1.0
         SJMP TSTART
```

Explanation:

Remember that we are NOT using any interrupts here. P3.2 can be used to control Timer0's execution externally.
If switch connected to P3.2 is pressed, then Timer0 will run else will stop.

When we say, Timer is running, we mean that Timer's TL0-TH0 values are being incremented every machine cycle. In this application LED will blink every one-second only if the switch is pressed.

We have to set Gate bit of TMOD. Look at the TMOD value in the source code.

We need not to monitor the Switch state. It will happen automatically and will be taken care by microcontroller internally.

App 7: Object counter

Connections:

Connect eight LEDs to all pins of P2
Connect one pulled-down switch to P3.4

Source Code:

```
START:
       MOV TMOD, #00000101B
       TSTART:
           SETB TR0
           LOOP:
                 MOV P2, TL0
           SJMP LOOP
```

Explanation:

As you are using T0 as counter, switch SW1 connected to P3.4 will generate High to Low pulse when pressed once. While writing program, you need not to monitor this switch as this is taken care automatically.

Whenever SW1 is pressed, value of TL0 will get incremented by one.

Loop section will copy value of TL0 to Port 2. You have connected 8 LEDs to P2. These LEDs will represent Binary value present in TL0, i.e. current value of the Counter.

This is 8 bit counter. If you want to make a full 16 bit object counter then you will need another 8 LEDs. Connect these LEDs to P1 and copy TH0's value to P1 inside the loop. Additional 8 LEDs will show their values only after value of TL0 cross 255 once.

App 8: Interrupt – Timer0

Before you read further, lets understand concept behind Interrupt is. To understand this, we have an example for you.

Do you monitor the wall clock for full night so that you can get-up early morning at a particular time?
or
You simply set the alarm clock and enjoy your sleep. Alarm keeps working behind scene without your monitoring and rings up at the time you have set.

Second example demonstrates usage of interrupts. In daily life we use various interrupt services like reminders using cell phone, microwave time settings, etc. This can be called as "concentrating on your core work and let systems take care of your alerts"

We as developer of microcontroller based systems, we have to make sure that the main task assigned to the microcontrollers is efficient and sub tasks are being taken care automatically. To set certain task in automatic mode, in other words we can say, to outsource the sub tasks, we have to use interrupts.

Interrupts in 89S52 can be used for Timer0, Timer1, Timer2, External Event 0 (INT0), External Event 1 (INT1) and RS232. You may refer the 89S52's datasheet for exact details of these interrupts.

Connections:

Connect one LED to P1.0

Source Code:

```
SJMP START

        ORG 000Bh
        LJMP BLINK

START:
        MOV R0,#14d
        MOV TMOD, #01h
        SETB TR0
        MOV IE,#10000010b
        EmptyLoop:
                NOP
                NOP
                ; Imagine this is our most important task in real application
                ; Hence we do not have to write a code to blink LED every one
                ; second as it has been taken car by Interrupt automatically
        SJMP EmptyLoop

BLINK:
       DJNZ R0, SKIP
       MOV R0, #14D
       CPL P1.0
SKIP:
        RETI
```

Explanation:

This application will blink LED every 1 second using Timer Interrupt.
It will keep LED ON and OFF for approximately 1 second.
Read E-Learning Tutorial for 89S52 for more clarity.

App 9: Interrupt – External 0

This application will use one switch to toggle LED from ON to Off and back to ON using external interrupt 0.

Connection:

Connect one LED to P1.0
Connect one Pulled up switch to P3.2 which is INT0 pin.

Source Code:

```
SJMP START

ORG 0003h
        LJMP ChangeState

START:
         MOV IE,#10000001b
         EmptyLoop:
                NOP
                NOP
                SJMP EmptyLoop

        ChangeState:
              CPL P1.0
        RETI
```

Look at the IE value. Read more in E-Learning tutorial.

App 10:
Receiving data from PC using Visual Basic program over RS232

For Source code refer “CD\Project Source Code\89S\Receive Character From PC over RS232”

App 11:
Character transmission from 89S to PC using Visual Basic program over RS232

For source code refer “CD\Project Source Code\89S\Send Character to PC over RS232”

Connections:

Make following connections for APP 11 and APP 12.

Connect LCD as was connected in APP 5.

Connect a 9 Pin cable between 9 Pin connector of Section 14 (RS232 interfacing section) and your PC.

Connect a 2 Pin F-F multicolor wire between Section 14’s TxRx and the 89S52 as described below.
RX to P3.0
TX to P3.1

Explanation:

Refer E-Learning tutorial for 89S52 for details of the source code.

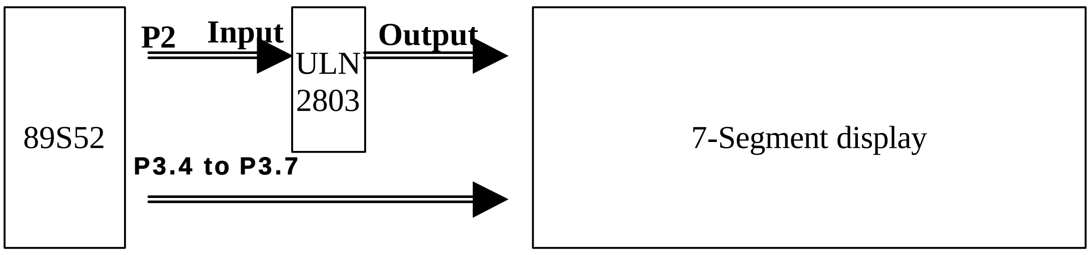
App 12: Interfacing 7-Segment LED display

Connections:

Connect P2 to the input section of ULN 2803

Connect output of ULN 2803 section to 8 Pin F-F multicolor connector cable of 7-segment section.

Connect 4 Pin F-F multicolor connector cable between P3.4 – P3.7 and 4 Pin connector of 7-segment section.



ULN2803 is used as high current driver to drive four number of seven segment displays.

Source Code:

Refer CD\Project Source Code\89S\7-Segment Display

Explanation:

There are 4 7-segment displays. This code will print 12.34.
This is achieved by keeping one display on and rest off at any given time for a delay of approximate 300 microseconds. When you see the result, you will find all displays are ON and showing there respective assigned number. This happens because human eye cannot distinguish the speed of changeover.

AVR

(Concepts and Applications based on mega8 / mega16 / mega32)

1. What do I need to get started with AVR?

This Book is part of EEDT. While writing this book, it is assumed that, you have completed all experiments with 89SXX.

The first part of this book deals with interfacing with 89S52. This part is designed for beginners who are totally new to the concept of microcontrollers.

If you are directly jumping to this section of the book, then you should learn 89S52's all interfacing techniques and features like timer, interrupts, RS232 etc.

All sample codes for AVR are in C language. You must have a *GOOD* sense and experience of C Programming. I am taking about the C Programming, which we type in Borland C or Turbo C and run programs in DOS mode. You must have an excellent understanding of Functions, Arrays, Data Types, Structures, Pointers, Function Pointers, Parameters and return types of Function.

If you don't have all these skills of C Programming then you better acquire it before you start with AVR.

You can use ASSEMBLY language for AVR Controllers, but listen to my words, "NEVER EXPERIMENT AVR WITH ASSEMBLY". This is not the good place to explain "why not to assembly". Drop me an email to know the reason.

In short, this book will guide you assuming, you are an expert 8051 programmer and worked on 89S51/52 controllers.

Thus you need to have following things to get started with an AVR:

1. EEDT (Embedded Engineer's Development Tool)
2. AVR Datasheets (Available in CD)
3. WinAVR (Free to use full version IDE to write C codes)
4. Knowledge of C programming for PC or any other microcontroller.

You have to install WinAVR software available under :
CD\Install These Softwares\WinAVR-20060125-install.exe

You have already installed HandyProg (Inbuilt ISP Programmer for 89S and AVR)

If not then you may install it from:
CD\Install These Softwares\DeccanRobots ISP Programmer\setup.exe

EEDT has no AVR controller provided by default. You have to order it as an optional component from DeccanRobots or you may buy it from your local resource.

To experiment using EEDT board, you have to connect 6 Pin F-F multicolor cable between HandyProg's ISP connector and ISP connector of your selected device.

Before you proceed further, read E-Learning Tutorial for AVR controllers.
You must read

- "How to Use WinAVR?".

2. mega and tiny AVRs

Features of various AVR Controllers

mega8

- 130 Powerful Instructions (This is the reason to SAY NO FOR ASSEMBLY LANGUAGE)
- 32 x 8 General Purpose Working Registers
- 8K Bytes of In-System Self-Programmable Flash
- 512 Bytes EEPROM
- 1K Byte Internal SRAM
- Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Three PWM Channels
- 6-channel ADC in PDIP package
- Byte-oriented Two-wire Serial Interface
- Serial USART
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated RC Oscillator
- External and Internal Interrupt Sources
- Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- 23 Programmable I/O Lines
- Operating Voltages 2.7 - 5.5V (ATmega8L) / 4.5 - 5.5V (ATmega8)
- Crystal Frequency 0 - 8 MHz (ATmega8L) / 0 - 16 MHz (ATmega8)

mega16 (All mega8 features are available in mega16 in addition to the following features)

- 131 Powerful Instructions (Reason to SAY NO FOR ASSEMBLY LANG.)
- 16K Bytes of In-System Self-Programmable Flash
- JTAG Interface
- Four PWM Channels
- 8 Channel ADC
- 32 IO Lines

mega32 (All mega8 and mega16 features are available in mega16 in addition to the following features)

- 32K Bytes of In-System Self-Programmable Flash
- 2K Byte Internal SRAM

mega8 is 28 Pin controller where as mega16/32 are 40 pin controllers in DIP package.

tiny13 (8 Pin DIP package)

- 120 Powerful Instructions
- 32 x 8 General Purpose Working Registers
- 1K Byte of In-System Programmable Program Memory Flash
- 64 Bytes In-System Programmable EEPROM
- One 8-bit Timer/Counter with Prescaler and Two PWM Channels
- 4-channel, 10-bit ADC with Internal Voltage Reference
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator

3. How to use AVR's Port Pins?

All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions. The pin driver is strong enough to drive LED displays directly.

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn.

DDxn is used to configure direction of the port or port pin.
PORTxn is used to output data on a port
PINxn is used to read value from a port pin.

Where x stands for numbering letter of the port and n represents the bit number
e.g DDRD0 i.e. DDR of 0th bit for D port.

What is DDR?
The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

Use following DDR values to indicate if you want to input or output the data:
Actual values will be stored with PORTxn

```
To read input data on all 8 pins of a Port D:
        DDRD = 0x00
To output data on all 8 pins of a Port D
        DDRD=0xff
To read data only on 4th bit of Port D
        cbi(DDRD,4);
To write data to 4th bit of Port D
        sbi(DDRD,4);
```

With these values, you have to use PORTD or PIND to output or input values respectively.

Some examples of PORT I/O with C language:

```
DDRD=0xff;              //Set port D in output mode
PORTD=0x55;             //Copy 0x55 value to the port D

cbi(DDRD,3);            //set port D's 3rd bit in input mode
if(bit_is_clear(PIND,3))        // if 3rd bit of port D is clear then do something
{
        //do something
}
```

Alternate Port Functions:
Most port pins have alternate functions in addition to being general digital I/Os.
e.g.
In mega8 Pin number 2 is PortD's 0th bit and also acts as RX for serial communication.
In mega8 Pin number 9 and 10 are used as PB6 and PB7, alternately can be used as XTAL1 and XTAL2.

Now the main question is, how AVR will know your intension of Pin usage.
This is resolved by AVR Fuse Bit settings and certain registers settings.

4. System clock and memory in AVR

AVR microcontrollers are well equipped with internal oscillator to act as a source of system clock. Remember, you have used external crystal oscillator for 89S52 experiments. In AVR microcontrollers, you need not to connect any external crystal as long as your project is not time critical.

In other words, in-built oscillator of AVR is R-C based and has a tolerance of around 10 to 15% in normal operating temperature. Thus if you build a voltmeter using mega8 AVR controller based on its internal oscillator, then your application might show 4.1V reading when placed inside the room and 4.9V when placed under sunlight due to temperature variation for a same input voltage.

To avoid this, you have to use an external crystal oscillator or an internal R-C oscillator with calibration.

External Crystal oscillator requires a crystal of desired clock frequency and a pair of capacitors.

Calibrated internal R-C oscillator method requires an external low frequency crystal with a pair of capacitors. External low frequency crystal is used to calibrate the internal R-C oscillator. The process of calibration is explained in E-Learning tutorial for AVR.

Read "Calibration of Internal Oscillator" from E-Learning tutorial for more clarity.

Types of Clocks

mega8 has 5 types of clocks.

1. System clock (Used by system to process instructions)
2. I/O clock (Used by timers, counters and serial transmission system)
3. Flash Clock (Used by flash interface)
4. Asynchronous Clock (Used by asynchronous timers and counters)
5. ADC clock (Used by internal ADC to reduce noise and increase accuracy)

Types of memories:

mega8 has 4 types of inbuilt memories:

1. In-system re-programmable flash memory (Used to store your program)
2. SRAM Data memory (Used to store your variables and registers values)
3. EEPROM Data memory (Used to store values permanently even if power is switched off)
4. I/O memory (Used to store input/output values)

As a AVR programmer using C language, you do not have to think much regarding addresses of these memory sections. You simply have to write your C code and program the hex in to microcontroller.

5. AVR Fuse Bits

AVR Fuse bits play an important role in configuration of your system.
AVR Fuse bit is a method of expressing your choice of settings like type of oscillator, frequency range of oscillator, eeprom memory erase, brown-out-detection etc.

When we say Fuse Bit is programmed, its value is expressed by 0.
When we say Fuse bit is unprogrammed, it values is expressed by 1.

Find the list of Fuse bits and description:

Fuse Bit Name	Description	Default Value
RSTDISBL	1 = PC6 is reset Pin 0 = PC6 is general I/O Pin	1
WDTON	1 = Watchdog timer disabled 0 = Watchdog timer enabled	1
SPIEN	1 = Disable Serial Programming 0 = Enable serial programming	0
CKOPT	Various uses based on values of CKSEL fuse bits	1
EESAVE	1 = EEPROM memory is not preserved during chip erase 0 = EEPROM memory is preserved during chip erase	1
BOOTSZ1	Selects Boot Size in combination with BOOTZ0	0
BOOTSZ0	Selects Boot Size in combination with BOOTZ1	0
BOOTRST	Reset vector selection	1
BODLEVEL	Brownout detector trigger level	1
BODEN	1 : BOD Disabled 0 : BOD Enabled	1
SUT1	Select startup time	1
SUT0	Select startup time	0
CKSEL3	Select clock source	0
CKSEL2	Select clock source	0
CKSEL1	Select clock source	0
CKSEL0	Select clock source	1

Default combination of CKSEL fuse bits results in to 1Mhz internal oscillator selection.

Use following values of CKSEL 3, CKSEL 2, CKSEL 1, CKSEL 0 respectively for internal oscillator selection

For 1 MHz internal oscillator use 0001
For 2 MHz internal oscillator use 0010
For 4 MHz internal oscillator use 0011
For 8 MHz internal oscillator use 0100

6. Applications for AVR (mega8)

APP 1 :	Stepper Motor control using 3 switches
APP 2 :	6 Channel Voltmeter using 7-Segment display with a resolution of 0.01V
App 3 :	6 Channel Voltmeter using 16x2 LCD with resolution of 0.01V
APP 4 :	PWM generator
APP 5 :	Object counter using 16x2 LCD
App 6 :	RS232 communication using mega8

App 1: Stepper motor control using 3 switches

Connections: (Refer respective interfacing section details in earlier part of this book)
Connect stepper motor to PC0, PC1, PC2, PC3 via ULN2803 circuit
Connect 3 pulled up switches to PD0, PD1, PD2

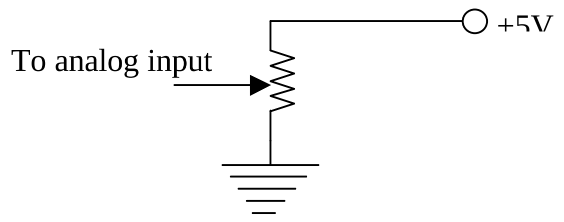
Description:
SW1 connected to PD0 will start / stop the stepper motor
SW2 connected to PD1 will rotate the motor in clockwise direction
SW3 connected to PD2 will rotate the motor in anti-clockwise direction

Source Code:
Refer CD\Project Source Code\Avr\Stepper motor control using 3 switches

Explanation:
Refer E-Learning tutorial for AVR, read APP 1 for AVR

APP 2: 6 Channel Voltmeter using 7-Segment display with a resolution of 0.01V

Connections: (Refer respective interfacing section details in earlier part of this book)
Connect 6 analog inputs to PC0 to PC5 which are analog input channels for mega8
Analog inputs are not provided with the EEDT. You can make it your self. Refer the circuit given below:



Connect 7-Segment display's 8 data lines to PortD via ULN 2803
Connect 7-Segment display's 4 control lines to PB4, PB5, PB6, PB7

Description:
Program will monitor continuously the selected channel.
Sample program need an upgradation for channel selection by key, which is left to you.
This program will display voltage from 0 to 5V

Source code:
Refer CD\Project Source Code\AVR\6 Channel Voltmeter using 7-Segment

Explanation:
Refer E-Learning tutorial for AVR, read APP 2 for AVR

App 3: 6 Channel Voltmeter using 16x2 LCD with resolution of 0.01V

Connections:

ADC connections as per APP 2.

Connect LCD’s control lines as described:

RS	=>	PB5
RW	=>	PB6
EN	=>	PB7

Connect LCD’s 4 data line as described: (Here 4 bit LCD interface is used)

D4	=>	PD0
D5	=>	PD1
D6	=>	PD2
D7	=>	PD3

Description:

Program will monitor continuously the selected channel.
Sample program need an upgradation for channel selection by key, which is left to you.
This program will display voltage from 0 to 5V

Source code:

Refer CD\Project Source Code\AVR\6 Channel Voltmeter using LCD

Explanation:

Refer E-Learning tutorial for AVR, read APP 3 for AVR.

APP 4: PWM Generator

Connections:

Connect one LED to PB1 which is also used to output PWM.
Connect 2 pulled up switches to PD0 and PD1. These keys will be used to control the intensity.
Connect 2 LEDs to PD2 and PD3 to indicate the max and min level of PWM signal.

Description:

This application will reduce and increase glowing intensity of LED

Source code:

Refer CD\Project Source Code\AVR\PWM Generator

Explanation:

Timer 1 is configured as 10 bit PWM generator using interrupt. OC1 i.e. PB1 is used to display the pulse width using LED.

APP 5: Object counter using 16x2 LCD

Connections:

Connect LCD’s control lines as described:

RS	=>	PB5
RW	=>	PB6
EN	=>	PB7

Connect LCD’s 4 data line as described: (Here 4 bit LCD interface is used)

D4	=>	PD0
D5	=>	PD1
D6	=>	PD2
D7	=>	PD3

Connect a pulledup switch to PD4 which is also a input line for external pulses to be counted

Description:

Press switch simulating an object count, LCD will update the total count.

Source code:

Refer CD\Project Source Code\AVR\Object Counter

Explanation: Refer E-Learning tutorial for AVR, read APP 5 for AVR.

App 6: RS232 communication using mega8

Connections:

Make connections between RS232 section and mega8.
Connect LCD.

Description:

This application will send text from PC to controller.

Source code:

Refer CD\Project Source Code\AVR\RS232