DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGG

MICRO CONTROLLER LAB MANUAL
μC LAB

IV Semester (17EC4DLMCR)
Autonomous Course
2018-19

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

Dayananda Sagar College of Engineering

Shavige Malleshwara Hills, Kumaraswamy Layout,
Banashankari, Bangalore-560078, Karnataka
Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789
Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu
(An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)
(Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade)

DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGINEERING

MICROCONTROLLER LAB MANUAL

IV Semester (17EC4DLMCR)

Autonomous Course

2018-2019

Dr. T.C. Manjunath & Dr. M. Roopa
Vibha T.G.,Mahima U,Chaitra A
H.S. Veena,Nutesh S

Name of the Student	:	
Semester /Section	:	
USN	:	
Batch	:	

Dayananda Sagar College of Engineering
Shavige Malleshwara Hills, Kumaraswamy Layout,
Banashankari, Bangalore-560078, Karnataka
Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789
Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu
(An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)
(Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade)

Dayananda Sagar College of Engineering

Dept. of E & C Engg

Name of the Laboratory	:	Microcontroller Lab / 17EC4DLMCR
Semester/Year	:	IV/2018-2019 (Autonomous)
No. of Students/Batch	:	20
No. of Computers	:	30
Major Equipment's	:	Dell Computers Microcontroller Board DSP TMS320 Kit Dc Motor Interface Elevator Interface Dual DAC interface Stepper Motor Interface Logic Controller Interface Digital Oscilloscope Power Supply ±5v, ±12v, ± 30v
Operating System & Application	:	Windows 8.1, UPS Keil Micro Vision, Hardware µC interfacing kits Matlab 2014, CC Studio-V
Area in square meters	:	104 Sq mts
Location	:	Level – 3
Total Cost of Lab	:	Rs. 15,00,000/-

Lab Incharge/s: Dr. Prof. M. Roopa
Prof. Vibha T.G.
Prof. Mahima U
Prof. Chaitra A

Instructor	:	Mrs. H.S. Veena, Mr Nuthesh
HOD	:	Dr. T.C. Manjunath, Ph.D. (IIT Bombay)

About the college & the department

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programmes under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of "Dayananda Sagar Institutions" (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centres in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.'s & 30^{+} staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 10 sophisticated labs & dozen class room, R & D centre, etc…

Vision and Mission of the Institute:

Vision:

- To impart quality technical education with a focus on Research and Innovation emphasizing on Development of Sustainable and Inclusive Technology for the benefit of society.

Mission:

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence.
- To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.
- To train the students to the changing technical scenario and make them to understand the importance of sustainable and inclusive technologies.

Vision and Mission of the Department

Vision :

- To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional success.

Mission:

- Offering quality education in Electronics and Communication Engineering with effective teaching learning process in multidisciplinary environment.
- Training the students to take-up projects in emerging technologies and work with team spirit.
- To imbibe professional ethics, development of skills and research culture for better placement opportunities.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

After four years, the students will be

PEO1 : ready to apply the state-of-art technology in industry and meeting the societal needs with knowledge of Electronics and Communication Engineering due to strong academic culture.

PEO2 : competent in technical and soft skills to be employed with capability of working in multidisciplinary domains.

PEO3 : professionals, capable of pursuing higher studies in technical, research or management programs.

PROGRAM SPECIFIC OBJECTIVES (PSOs):

Students will be able to

PSO1 : Design, develop and integrate electronic circuits and systems using current practices and standards.

PSO2 : Apply knowledge of hardware and software in designing Embedded and Communication systems.

Course Objectives

1. To provide knowledge on fundamental concepts of 8051.
2. To provide understanding of assembly language programming concepts and improve the programming skill.
3. To familiarize students with Kiel software.
4. To familiarize students with different sets of instructions available for programming.
5. To give exposure on interfacing concepts using C language with different peripherals.
6. To provide foundation for developing 8051 based applications.

Course Outcomes

After the completion of this laboratory the students will have the ability to ….

CO1	Employ the knowledge of 8051 architecture & memory organization, for writing assembly language programs using Kiel software.
CO2	Apply the assembly language programming skills to build ALPs for arithmetic & logical operations.
CO3	Analyze & code for timers, serial communication & interrupts.
CO4	Use hardware kit and various peripherals to analyze hardware interfacing.
CO5	Apply embedded C programming skills to develop programs for hardware interfacing.
CO6	Demonstrate simulated hardware programs on 8051 kit interfaced with various peripherals.

Mapping of Course outcomes to Program outcomes

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	2	1	-	-	-	-	-	1	1	-	-	-	-
CO2	3	2	1	-	-	-	-	-	1	1	-	-	-	-
CO3	3	2	1	-	-	-	-	-	1	1	-	-	-	-
CO4	3	2	-	-	-	-	-	-	1	1	-	-	-	-
CO5	3	3	2	1	1	-	-	-	1	1	-	-	-	1
CO6	3	3	2	1	1	-	-	-	1	1	-	-	-	1
CO AVG	3	2	1	1	1	-	-	-	1	1	-	-	-	1

MICRO-CONTROLLER LAB

Course code : 17EC4DLMCR

Credits: 2

L : P : T : S : 1 : 2 : 0 : 0 — CIE Marks: 50

Exam Hours : 3 — SEE Marks: 50

Expt	Course Content	Hours	COs
Software Programs : ALP simulation programs on 8051			
1.	Data Transfer Programs – 8051 : a. Block data transfer without overlap b. Block exchange.	03	C01 C02
2.	Arithmetic operation : a. Addition, subtraction, multiplication and division of two 8 bit numbers. b. Bubble Sorting algorithm.	03	C01 C02
3.	Bit manipulation, Boolean & Logical Instructions programs : a. To perform logical operation on two 8 bit numbers. b. Conditional bitwise logical operations.	03	C01 C02
4.	Counters : a. BCD counter using software delay. b. Hex counter using timer delay of 1 sec.	03	C01 C02 CO3
5.	Conversion: 8051 a. HEX– Decimal b. Decimal – HEX	03	C01 C02
6.	Serial data transmission with variable baud rate – 8051.	03	C01 C02 CO3

Hardware programs to interface 8051 chip to Interfacing modules			
7.	Implementation of DAC 0808 interface to 8051 to generate square, triangular, ramp waveforms.	03	C04 C05 C06
8.	Stepper motor interface to 8051.	03	C04 C05 C06
9.	DC motor interface to 8051.	03	C04 C05 C06
10.	Alphanumeric LCD panel interface to 8051.	03	C04 C05 C06
11.	Elevator interface to 8051.	03	C04 C05 C06

Cycle of experiments

DO's

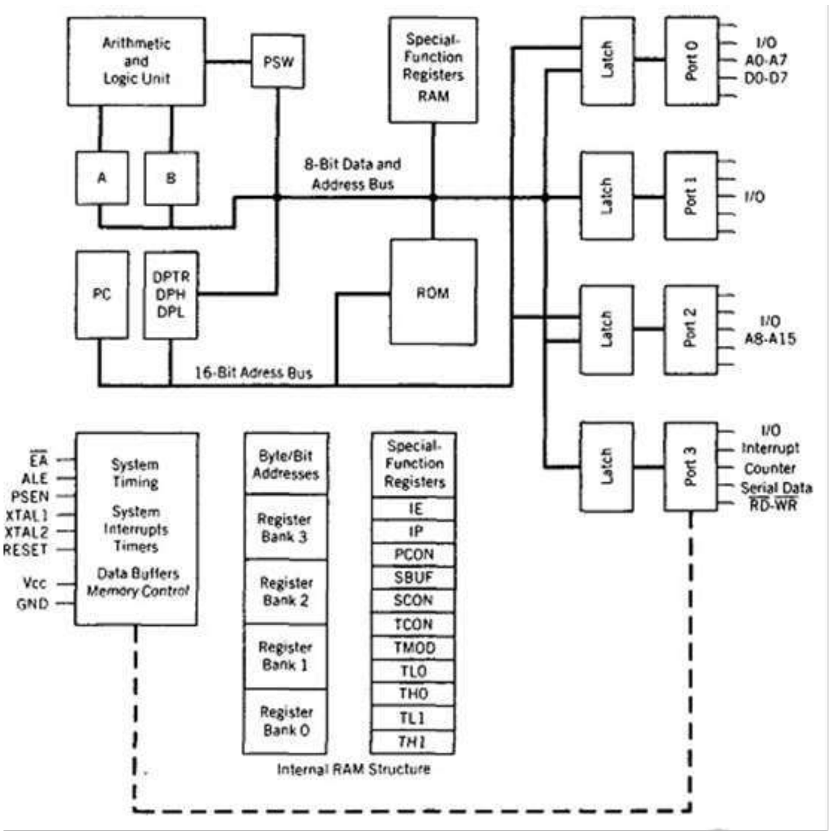
- All the students should come to LAB on time with proper dress code and identity cards.
- Keep your belongings in the corner of laboratory.
- Students have to enter their name, USN, time-in/out and signature in the log register maintained in the laboratory.
- All the students should submit their records before the commencement of Laboratory experiments.
- Students should come to the lab well prepared for the experiments which are to be performed in that particular session.
- Students are asked to do the experiments on their own and should not waste their precious time by talking, roaming and sitting idle in the labs.
- Observation book and record book should be complete in all respects and it should be corrected by the staff member.
- Before leaving the laboratory students should arrange their chairs and leave in orderly manner after completion of their scheduled time.
- Prior permission to be taken, if for some reasons, they cannot attend lab.
- Immediately report any sparks/ accidents/ injuries/ any other untoward incident to the faculty /instructor.
- Once the experiment is completed in all respects, students should take the sign of the staff / lab incharge & before coming to the next lab, the practical record should be written & complete in all respects, else, marks will be reduced as the record will be incomplete.
- In case of an emergency or accident, follow the safety procedure.
- Switch OFF the power supply after completion of experiment.

DONT's

- Do not make noise in the Laboratory & do not sit on experiment table.
- Do not make loose connections and avoid overlapping of wires
- Don't switch on power supply without prior permission from the concerned staff.
- Never leave the experiments while in progress.
- Do not leave the Laboratory without the signature of the concerned staff in observation book.
- Do not switch on the power supply before verification of the connected circuits by concerned staff.
- Do not feed higher voltages than rated to the device.
- Do not upload, delete or alter any software on the laboratory PC's.
- Do not write or mark on the equipment's.
- Usage of mobile phone is strictly prohibited.
- Ragging is punishable.
- If student damages the equipment or any of the component in the lab, then he / she is solely responsible for replacing that entire amount of the equipment or else, replace the equipment.

INTRODUCTION TO MICRO-CONTROLLER & ITS ARCHITECTURE INSTRUCTION SETS :

BREIF INTRODUCTION ABOUT THE 8051 MICRO-CONTROLLER



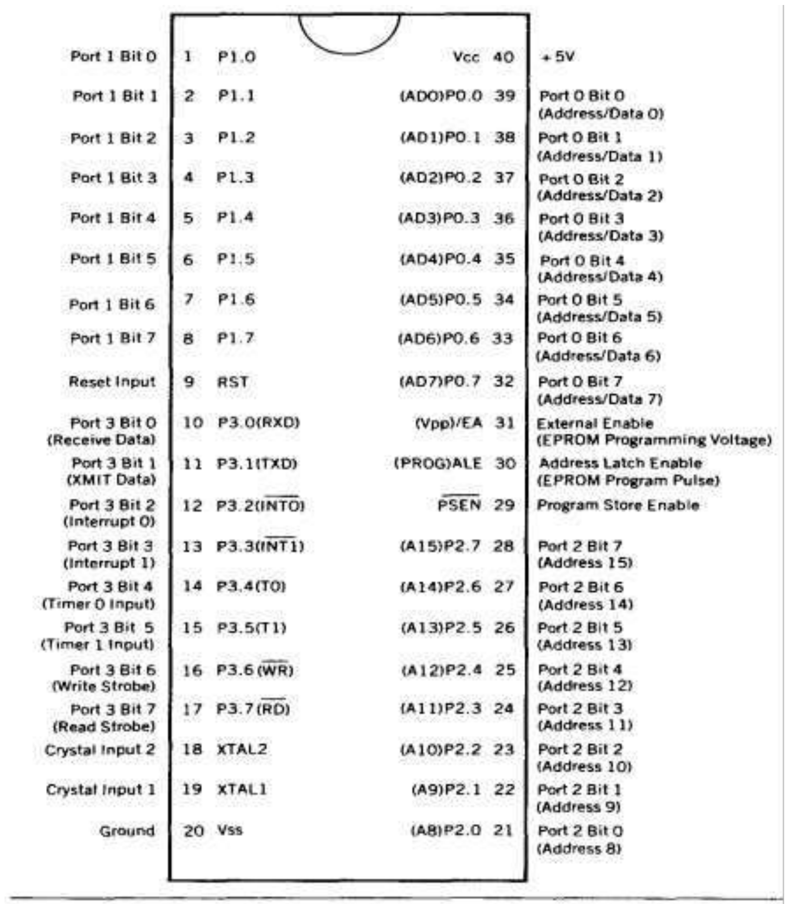
8051 Architecture:

Architecture shows usual CPU components such as Program counter (PC), ALU, working registers and clock circuits.

Features:

- 8 bit CPU with registers A (Accumulator) & B
- 16 bit program counter (PC) and Data pointer (DPTR).
- 8 bit Program status word (PSW).
- 8 bit stack pointer.
- Internal ROM of 4kbytes.
- Internal RAM of 128 bytes.
- 4 register banks each containing 8 registers.
- 16 bytes of bit addressable registers.
- 8 bytes of general purpose data memory.
- 32 I/O pins arranged in four 8 pin ports (P0 to P3).
- Two 16 bit Timer/Counter (T0 & T1).
- Full duplex serial data receiver / transmitter (SBUF).

- Control registers, TCON, TMOD, SCON, PCON, IP & IE.
- 2 external & 3 internal interrupt source.
- Oscillator and clock circuits.


Port 1 Bit 0 | 1 P1.0 | Vcc 40 | +5V
Port 1 Bit 1 | 2 P1.1 | (AD0)P0.0 39 | Port 0 Bit 0 (Address/Data 0)
Port 1 Bit 2 | 3 P1.2 | (AD1)P0.1 38 | Port 0 Bit 1 (Address/Data 1)
Port 1 Bit 3 | 4 P1.3 | (AD2)P0.2 37 | Port 0 Bit 2 (Address/Data 2)
Port 1 Bit 4 | 5 P1.4 | (AD3)P0.3 36 | Port 0 Bit 3 (Address/Data 3)
Port 1 Bit 5 | 6 P1.5 | (AD4)P0.4 35 | Port 0 Bit 4 (Address/Data 4)
Port 1 Bit 6 | 7 P1.6 | (AD5)P0.5 34 | Port 0 Bit 5 (Address/Data 5)
Port 1 Bit 7 | 8 P1.7 | (AD6)P0.6 33 | Port 0 Bit 6 (Address/Data 6)
Reset Input | 9 RST | (AD7)P0.7 32 | Port 0 Bit 7 (Address/Data 7)
Port 3 Bit 0 (Receive Data) | 10 P3.0(RXD) | (Vpp)/EA 31 | External Enable (EPROM Programming Voltage)
Port 3 Bit 1 (XMIT Data) | 11 P3.1(TXD) | (PROG)ALE 30 | Address Latch Enable (EPROM Program Pulse)
Port 3 Bit 2 (Interrupt 0) | 12 P3.2(INT0) | PSEN 29 | Program Store Enable
Port 3 Bit 3 (Interrupt 1) | 13 P3.3(INT1) | (A15)P2.7 28 | Port 2 Bit 7 (Address 15)
Port 3 Bit 4 (Timer 0 Input) | 14 P3.4(T0) | (A14)P2.6 27 | Port 2 Bit 6 (Address 14)
Port 3 Bit 5 (Timer 1 Input) | 15 P3.5(T1) | (A13)P2.5 26 | Port 2 Bit 5 (Address 13)
Port 3 Bit 6 (Write Strobe) | 16 P3.6(WR) | (A12)P2.4 25 | Port 2 Bit 4 (Address 12)
Port 3 Bit 7 (Read Strobe) | 17 P3.7(RD) | (A11)P2.3 24 | Port 2 Bit 3 (Address 11)
Crystal Input 2 | 18 XTAL2 | (A10)P2.2 23 | Port 2 Bit 2 (Address 10)
Crystal Input 1 | 19 XTAL1 | (A9)P2.1 22 | Port 2 Bit 1 (Address 9)
Ground | 20 Vss | (A8)P2.0 21 | Port 2 Bit 0 (Address 8)

Note: Alternate functions are shown below the port name (in parentheses). Pin numbers and pin names are shown inside the DIP package.

DIP pin assignments:

It is a 40 pin IC, where 32 pins are used for 4 ports, P0,P1,P2,P3 (each of 8 pins). The rest of the pins are Vcc, Gnd,XTAL1, XTAL2, RST, EA (Low enable), PSEN (Lew enable) & ALE.

Port 0: It is 8 pins (32 to 39) port which can be used as input or output. To use it as both I/O pins, each pin must be connected externally to 10k-ohm pull up resistor as these pins are open drain unlike other ports. 8051 multiplexes address and data through port0 to save pins hence this port can be used as both address and data port (Ad0-AD7).

Port 1: 8 pins (1 to 8) port with internal pull up resistors. On reset it is configured as input port (all pins 1).

Port 2: 8 pins (21 to 28) port with internal pull up resistors. On reset it is configured as input port (all pins 1). This port is used as address pins while interfacing external memory of 64kB. P0 provides lower 8 bit address and P2 provides higher 8 bits of address but P2 is not multiplexed.

Port 3: 8 pins (10 to 17) port with internal pull up resistors. On reset it is configured as input port (all pins 1). P3 has the additional function of providing some important signals such as interrupts as shown in pin diagram.

Pin 18 & 19: External oscillator pins XTAL1 & XTAL2. 8051 has an on chip oscillator but requires external clock to run it. A quartz crystal oscillator is connected to XTAL1 & XTAL2. If TTL oscillator is used then, it is connected to XTAL1 and XTAL2 is left open.

Pin 9: Reset pin RST. It is an input and active high pin. On reset microcontroller terminates all activities.

Pin 29: Program store enable is low enable pin PSEN. This pin should be connected to OE pin of ROM chip when external memory is interfaced.

Pin 30: Address latch enable pin ALE. It is an output pin and active high. Used for de-multiplexing the address and data by connecting to the G pin of 74LS373 chip.

Pin 31: External access pin which is active low EA. Connected to V_{cc} for on chip ROM access and should be connected to Gnd while accessing external memory.

Internal RAM Organization

Lower 128 Bytes of RAM

	Byte Addresses								
	7F 30	General purpose RAM (Scratch pad)							
Bit Addressable Locations	2F	7F	7E	7D	7C	7B	7A	79	78
	2E	77	76	75	74	73	72	71	70
	2D	6F	6E	6D	6C	6B	6A	69	68
	2C	67	66	65	64	63	62	61	60
	2B	5F	5E	5D	5C	5B	5A	59	58
	2A	57	56	55	54	53	52	51	50
	29	4F	4E	4D	4C	4B	4A	49	48
	28	47	46	45	44	43	42	41	40
	27	3F	3E	3D	3C	3B	3A	39	38
	26	37	36	35	34	33	32	31	30
	25	2F	2E	2D	2C	2B	2A	29	28
	24	27	26	25	24	23	22	21	20
	23	1F	1E	1D	1C	1B	1A	19	18
	22	17	16	15	14	13	12	11	10
	21	0F	0E	0D	0C	0B	0A	09	08
	20	07	06	05	04	03	02	01	00
	18 – 1F	Bank 3							
	10 – 17	Bank 2							
	08 – 0F	Bank 1							
	00 – 07	Bank 0 (Default Register bank R0 – R7)							

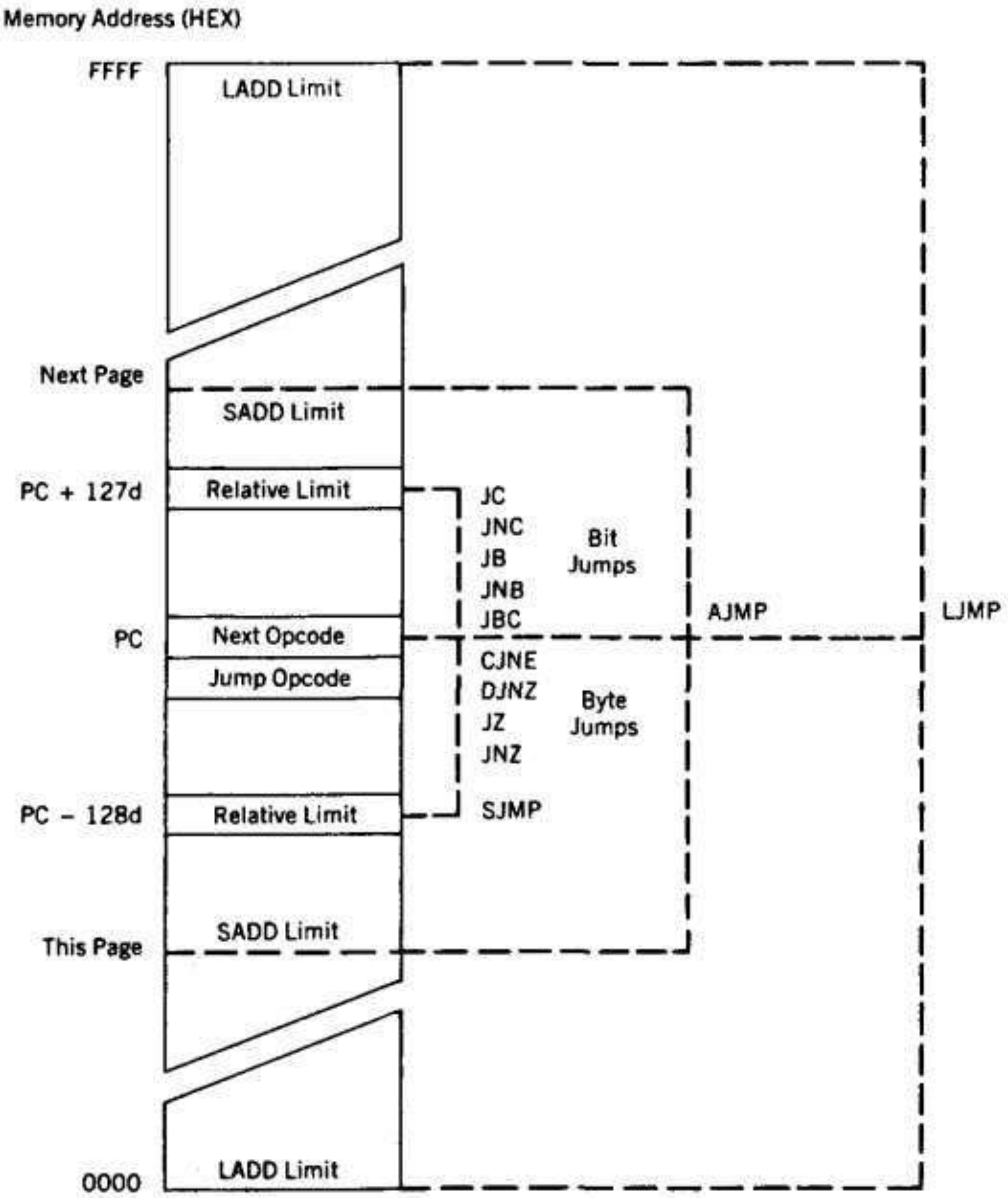
128 Bytes of Special Function Registers (SFR)

Byte Addresses	Bit Addresses								SFR Name
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Not bit addressable								SBUF
98	9f	9e	9d	9c	9b	9a	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Not bit addressable								TH1
8C	Not bit addressable								TH0
8B	Not bit addressable								TL1
8A	Not bit addressable								TL0
89	Not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit addressable								PCON
83	Not bit addressable								DPH
82	Not bit addressable								DPL
81	Not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Special Function Register (SFR) Addresses

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B Register	0F0H
PSW*	Program Status World	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt Priority Control	0B8H
IE*	Interrupt Enable Control	0A8H
TMOD	Timer / counter mode control	89H
TCON*	Timer / counter control	88H
T2CON*	Timer / counter 2 control	0C8H
T2MOD	Timer / counter mode control	0C9H
TH0	Timer / counter 0 high byte	8CH
TL0	Timer / counter 0 low byte	8AH
TH1	Timer / counter 1 high byte	8DH
TL1	Timer / counter 1 low byte	8BH
TH2	Timer / counter 2 high byte	0CDH
TL2	Timer / counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power Control	87H
*** Bit-addressable**		

Jump Instruction Ranges



Alphabetical List of Instructions

- ACALL - Absolute Call
- ADD, ADDC - Add Accumulator (With Carry)
- AJMP - Absolute Jump
- ANL - Bitwise AND
- CJNE - Compare and Jump if Not Equal
- CLR - Clear Register
- CPL - Complement Register
- DA - Decimal Adjust
- DEC - Decrement Register
- DIV - Divide Accumulator by B
- DJNZ - Decrement Register and Jump if Not Zero
- INC - Increment Register
- JB - Jump if Bit Set
- JBC - Jump if Bit Set and Clear Bit
- JC - Jump if Carry Set
- JMP - Jump to Address
- JNB - Jump if Bit Not Set
- JNC - Jump if Carry Not Set
- JNZ - Jump if Accumulator Not Zero
- JZ - Jump if Accumulator Zero
- LCALL - Long Call
- LJMP - Long Jump
- MOV - Move Memory
- MOVC - Move Code Memory
- MOVX - Move Extended Memory
- MUL - Multiply Accumulator by B
- NOP - No Operation
- ORL - Bitwise OR
- POP - Pop Value From Stack
- PUSH - Push Value Onto Stack
- RET - Return From Subroutine
- RETI - Return From Interrupt

- RL - Rotate Accumulator Left
- RLC - Rotate Accumulator Left Through Carry
- RR - Rotate Accumulator Right
- RRC - Rotate Accumulator Right Through Carry
- SETB - Set Bit
- SJMP - Short Jump
- SUBB - Subtract From Accumulator With Borrow
- SWAP - Swap Accumulator Nibbles
- XCH - Exchange Bytes
- XCHD - Exchange Digits
- XRL - Bitwise Exclusive OR
- Undefined - Undefined Instruction

INTRODUCTION

PROCESSOR used is Atmel AT89C51ED2 - Micro controller that has 64Kbytes of on-chip program memory. It is a version of 8051 with enhanced features. AT 89C51ED2 operates at 11.0592 MHz

PROCESSOR FEATURES :

ON-CHIP MEMORY :

CODE MEMORY : 64 KBytes of flash.

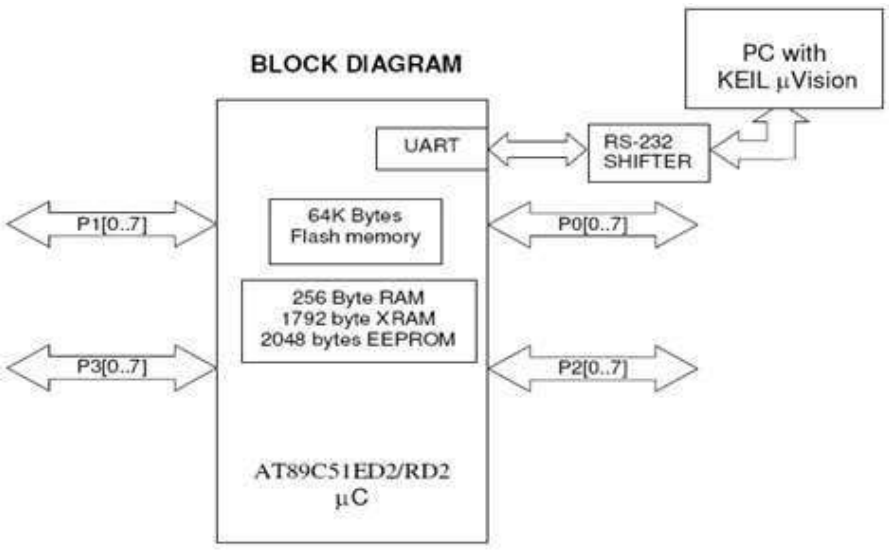
DATA MEMORY : 256 Bytes of RAM, 1792 Bytes of XRAM, 2K Bytes of EEPROM.

ON-CHIP PERIPHERALS : 2 16-bit Timers/Counters, Watch Dog Timer, Programmable Counter Array (PCA) on Port1 i.e. PWM and Capture & Compare, SPI (Serial Peripheral Interface) on Port1, Full duplex enhanced UART.

INTERRUPTS : Nine sources of interrupt (both external and internal). Two External interrupts INT0 and INT1 are provided with push button switches; these can also be used as general-purpose switches.

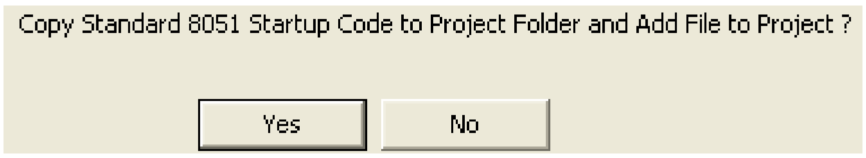
I/O (Port) Lines : Four 10-pin connectors for all the 32 I/O lines. P0, P1 and P2 Port lines are available on a 26-pin connector.

16X2 LCD & SERIAL I/O : are also available.

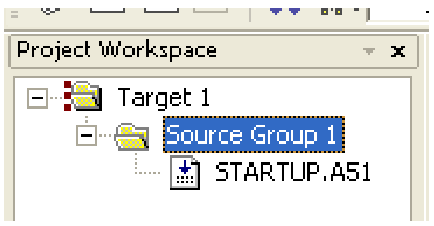


Creating and compiling a µVision5 project (8051 ALP Programs)

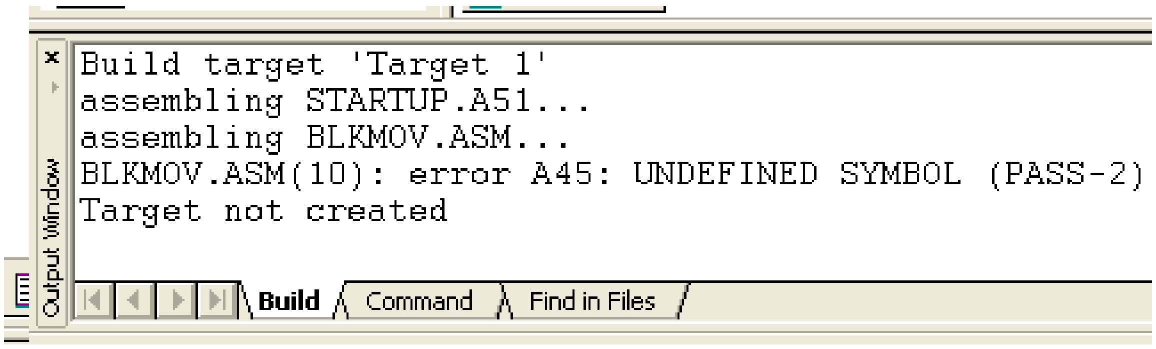
1. Double Click on the µVision5 icon on the desktop.
2. Close any previous projects that were opened using – Project->Close.
3. Start **Project – New Project,** and select the CPU from the device database (Database-Atmel- AT89C51ED2). (Select AT89C51ED2 or AT89C51RD2 as per the board).On clicking 'OK', the following option is displayed. Choose No.


Copy Standard 8051 Startup Code to Project Folder and Add File to Project ?

4. Create a source file (using File->New), type in the assembly or C program and save this (filename.asm/ filename.c) and add this source file to the project by right clicking on the Source Group in the Project Window and the **Add Files to group** option.



5. Build the project; using Project -> Build Project. µVision translates all the user application and links. Any errors in the code are indicated by – "Target not created" in the Build window, along with the error line. Debug the errors. After an error free build, go to Debug mode.



```
Build target 'Target 1'
assembling STARTUP.A51...
assembling BLKMOV.ASM...
BLKMOV.ASM(10): error A45: UNDEFINED SYMBOL (PASS-2)
Target not created
```

6. Now user can enter into **Debug** mode with **Debug- Start / Stop Debug session** dialog. Or by clicking in the icon.

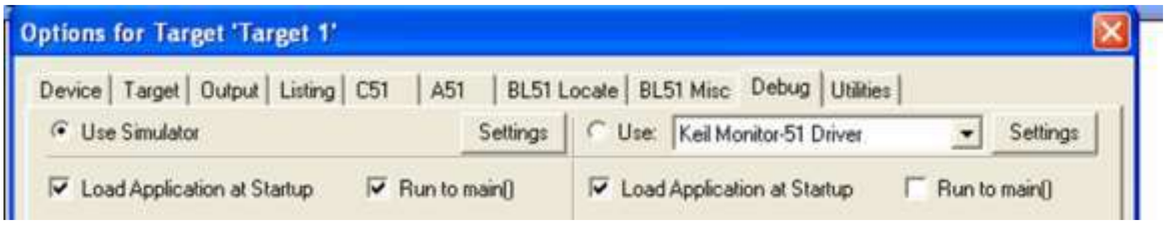
7. The program is run using the **Debug-Run** command & halted using **Debug-Stop Running.** Also the (reset, run, halt) icons can be used. Additional icons are (step, step over, step into, run till cursor).

NOTE:

1. If it is an ALP program, the appropriate memory window is opened using View -> memory window (for data RAM & XRAM locations), Watch window (for timer program), serial window for serial data transmission.
   To access data RAM area type address as D:0020h.
   Similarly to access the DPTR region (XRAM-present on chip in AT89C51ED2) say
   9000h location type in X:09000H.
   To access the code memory type address as C:0020h.

2. If it is an interface program an extra step has to be followed before step 5 as illustrated for ALP programs to see the outputs on the LCD, CRO, motor, led etc.
   Set the Target options using -> **Project** – **Options for Target** opens the µVision3 **Options for Target – Target** configuration dialog. Set the Xtal

frequency as 11.0592 Mhz, and also the **Options for Target – Debug – use either Simulator / Keil Monitor- 51 driver.**



(●) Use Simulator Settings
( ) Use: Keil Monitor-51 Driver Settings
☑ Load Application at Startup ☑ Run to main()
☑ Load Application at Startup ☐ Run to main()

If **Keil Monitor- 51 driver is used,**

Click on run to main() option. Then click on Settings -> COM Port settings

Select com port to which the board is connected and select the baud rate as 9600

Enable **Serial Interrupt** option.

If Simulator is used,

Go to view click on analysis window select logic analyzer to see the waveforms.

Software (Programming) Experiments

Experiment No. : 1	Date : / / .

DATA TRANSFER PROGRAMS

Aim 1 : a) Write an assembly language program to transfer n = 10 bytes of data from location 8035h to location 8050h (without overlap).

Algorithm :

1. Initialize origin of program at 0000H.
2. Jump to 30H and initialize origin at 30H.
3. Initialize registers to hold count & also the source & destination addresses of code memory.
4. Load lower byte of address into DPL register.
5. Get data from source location into accumulator.
6. Move destination address into DPL register.
7. Transfer data to the destination location.
8. Increment source and destination addresses.
9. Decrement the counting register and check if it has reached Zero.
10. Repeat step 5 to 9 till count is zero.

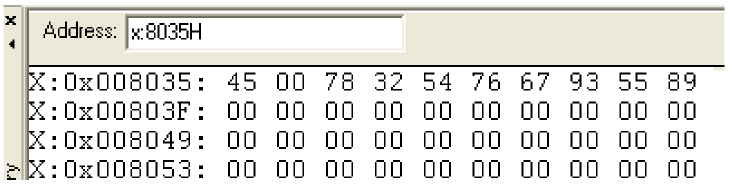
Note : For data transfer with overlap start transferring data from the last location of source array to the last location of the destination array.

Program :

Label	Mnemonic/ Operands	Comments
	ORG 0000H	//Origins program from 0000H location
	SJMP 30H	//Unconditional jump to 30H
	ORG 30H	//Program starts from 30H
	MOV DPH,#80H	//Higher byte of address is stored in DPH
	MOV R0,#35H	//Lower byte of source address
	MOV R1,#50H	//Lower byte of destination address
	MOV R3,#0AH	//count- Number of bytes to be transferred
BACK:	MOV DPL, R0	//DPTR stores complete source address
	MOVX A,@DPTR	//Read content at source address
	MOV DPL, R1	//Update DPTR with destination address
	MOVX @DPTR,A	//Write data at destination
	INC R0	//Increment source address lower byte
	INC R1	//Increment destination address lower byte
	DJNZ R3, BACK	//Decrement R3 & jump to BACK if its not Zero
HERE:	SJMP HERE	//Infinite looping
	END	//End directive

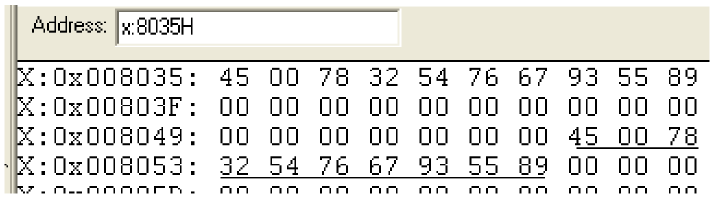
RESULT :

***Before Execution* :** 10 locations X:8035h are filled up with data.



```
Address: x:8035H
X:0x008035: 45 00 78 32 54 76 67 93 55 89
X:0x00803F: 00 00 00 00 00 00 00 00 00 00
X:0x008049: 00 00 00 00 00 00 00 00 00 00
X:0x008053: 00 00 00 00 00 00 00 00 00 00
```

After Execution : 10 locations X: 8050h are filled up with data from 8035h.



```
Address: x:8035H
X:0x008035: 45 00 78 32 54 76 67 93 55 89
X:0x00803F: 00 00 00 00 00 00 00 00 00 00
X:0x008049: 00 00 00 00 00 00 00 45 00 78
X:0x008053: 32 54 76 67 93 55 89 00 00 00
```

Aim 2 : b) Write an assembly language program to exchange n = 5 bytes of data at Location 0027h and at location 0041h.

<u>Algorithm:</u>

1. Initialize origin of program at 0000H.
2. Jump to 30H and initialize origin at 30H.
3. Initialize registers to hold count data & also the source & destination addresses.
4. Initialize registers to hold count (array size) & also the source & destination addresses.
5. Get data from source location into accumulator and save in a register.
6. Get data from the destination location into accumulator.
7. Exchange the data at the two memory locations.
8. Increment source and destination addresses.
9. Decrement the counting register and check if it has reached Zero.
10. Repeat from step 5 to 9 till count is zero.

Alter using XCH command

Algorithm:

1. Initialize origin of program at 0000H.
2. Jump to 30H and initialize origin at 30H.
3. Initialize registers to hold count data & also the source & destination addresses.
4. Initialize registers to hold count (array size) & also the source & destination addresses.
5. Get data from source location into accumulator and save in a register.
6. Exchange data using XCH command.
7. Increment source and destination addresses.
8. Decrement the counting register and check if it has reached Zero.
9. Repeat from step 5 to 8 till count is zero.

Program

Without XCH command

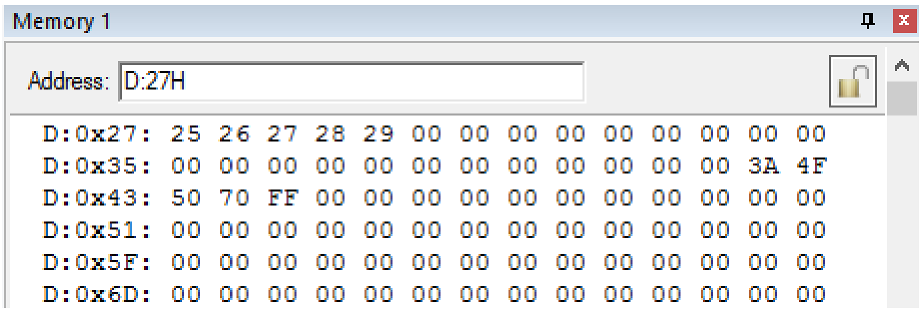
Label	Mnemonic/ Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,#27H	
	MOV R1,#41H	
	MOV R3,#05H	
BACK:	MOV A,@R0	
	MOV R2,A	
	MOV A,@R1	
	MOV @R0,A	
	MOV A, R2	
	MOV @R1,A	
	INC R0	
	INC R1	
	DJNZ R3, BACK	
HERE:	SJMP HERE	
	END	

With XCH command

Label	Mnemonic/ Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,#27H	
	MOV R1,#41H	
	MOV R3,#05H	
BACK:	MOV A,@R0	
	XCH A,@R1	
	MOV @R0,A	
	INC R0	
	INC R1	
	DJNZ R3, BACK	
HERE:	SJMP HERE	
	END	

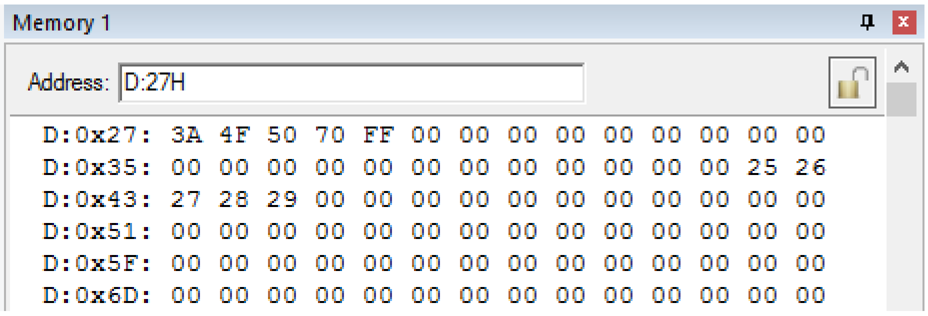
RESULT:

Before Execution : 5 locations at X:0027h & X:0041h are filled up with data.



```
D:0x27: 25 26 27 28 29 00 00 00 00 00 00 00 00 00
D:0x35: 00 00 00 00 00 00 00 00 00 00 00 00 3A 4F
D:0x43: 50 70 FF 00 00 00 00 00 00 00 00 00 00 00
D:0x51: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x6D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution : The data at X:8027h & X:8041h are exchanged.



```
D:0x27: 3A 4F 50 70 FF 00 00 00 00 00 00 00 00 00
D:0x35: 00 00 00 00 00 00 00 00 00 00 00 00 25 26
D:0x43: 27 28 29 00 00 00 00 00 00 00 00 00 00 00
D:0x51: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5F: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x6D: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Results / Conclusion :

The programming experiment was conducted successfully, the programming o/p is observed & the results are neatly tabulated.

Applications:

1.

2.

3.

4.

Remarks:

1.

2.

3.

4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 2	Date : / / .

ARITHMETIC OPERATIONS

ASSEMBLY LANGUAGE PROGRAM ILLUSTRATING ADDITION, SUBTRACTION, MULTIPLICATION AND DIVISION

Aim 1 : a) Write an ALP to perform the following:

If x = 0-perform w + v;

Else if x = 1-perform w - v;

Else if x = 2-perform w * v;

Else if x = 3-perform w / v, where w & v are eight bit numbers.

Algorithm:

1. Store the condition x in R1.
2. Load the first and second numbers to A and B registers respectively
3. Compare the contents of R1 and perform the operations add, sub, etc accordingly.
4. Store the result present in A and B registers to the appropriate memory locations.

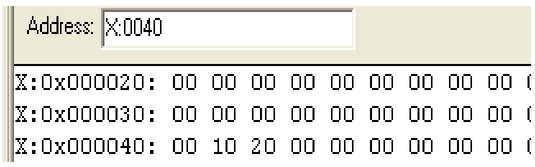
Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0, #40H	
	MOVX A,@R0	
	MOV R1, A	
	INC R0	
	MOVX A,@R0	
	MOV B, A	
	INC R0	
	MOVX A,@R0	
	CJNE R1,#00,CKSUB	
	ADD A,B	
	MOV B,#00	
	JNC SKIP	
	MOV B,#01H	
SKIP:	SJMP LAST	
CKSUB:	CJNE R1,#01,CKMUL	
	CLR C	
	SUBB A,B	
	MOV B,#00	
	JNC SKIP1	

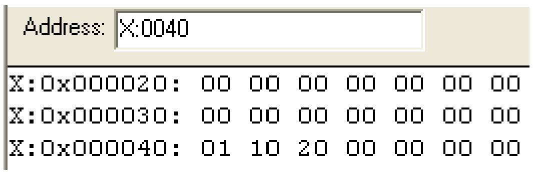
```
        MOV B,#0FFH
SKIP1:  SJMP LAST
CKMUL:  CJNE R1,#02,CKDIV
        MUL AB
        SJMP LAST
CKDIV:  CJNE R1,#03,OTHER
        DIV AB
        SJMP LAST
OTHER:  MOV A,#00
        MOV B,#00
LAST:   INC R0
        MOVX @R0,A
        INC R0
        MOV A,B
        MOVX @R0,A
HERE:   SJMP HERE
        END
```

RESULT:

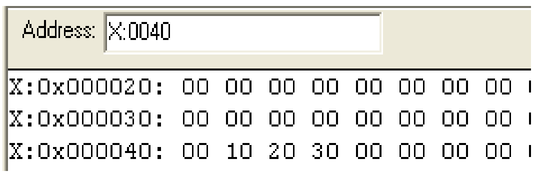
Before Execution: ADD


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00 00 00 00
X:0x000040: 00 10 20 00 00 00 00 00 00
```

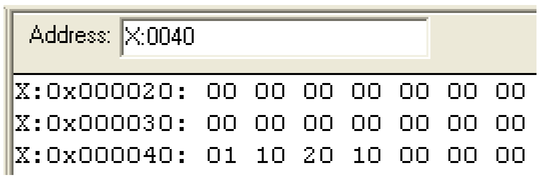
Before Execution: SUB


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00 00
X:0x000040: 01 10 20 00 00 00 00
```

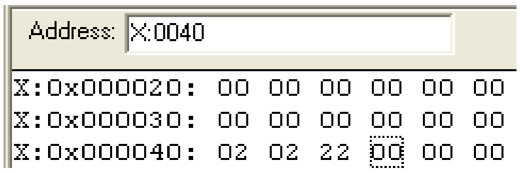
After Execution: ADD


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00 00 00
X:0x000040: 00 10 20 30 00 00 00 00
```

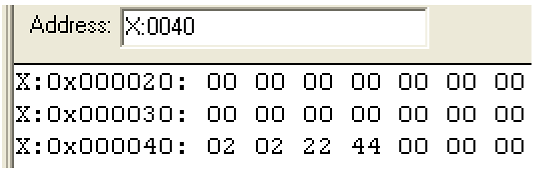
After Execution: SUB


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00 00
X:0x000040: 01 10 20 10 00 00 00
```

Before Execution: MUL


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00
X:0x000040: 02 02 22 00 00 00
```

After Execution: MUL


```
Address: X:0040
X:0x000020: 00 00 00 00 00 00 00
X:0x000030: 00 00 00 00 00 00 00
X:0x000040: 02 02 22 44 00 00 00
```

Aim 2 : b) Write an assembly language program to sort an array of n= 6 bytes of data in Descending order stored from location 9000h. (Use bubble sort algorithm).

Algorithm

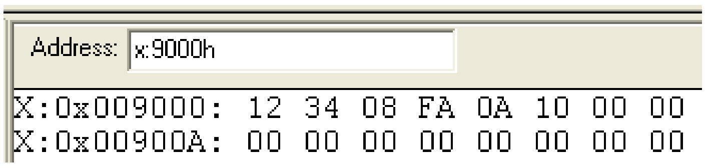
1. Store the elements of the array from the address 9000h
2. Initialize a pass counter with array size-1 count (for number of passes).
3. Load compare counter with pass counter contents & initialize DPTR to point to the start address of the array (here 9000h).
4. Store the current and the next array elements pointed by DPTR in registers B and r2 respectively.
5. Subtract the next element from the current element.
6. If the carry flag is set (for ascending order) then exchange the 2 numbers in the array.
7. Decrement the compare counter and repeat through step 4 until the counter becomes 0.
8. Decrement the pass counter and repeat through step 3 until the counter becomes 0.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,#05	
L1:	MOV DPTR,#9000h	
	MOV A,R0	
	MOV R1,A	
L2:	MOVX A,@DPTR	
	MOV B, A	
	INC DPTR	
	MOVX A, @DPTR	
	CLR C	
	MOV R2, A	
	SUBB A, B	
	JC NOEXCHG	
	MOV A,B	
	MOVX @DPTR,A	
	DEC DPL	
	MOV A,R2	
	MOVX @DPTR,A	
NOEXCHG:	INC DPTR	
	DJNZ R1,L2	
	DJNZ R0,L1	
HERE:	SJMP HERE	
	END	

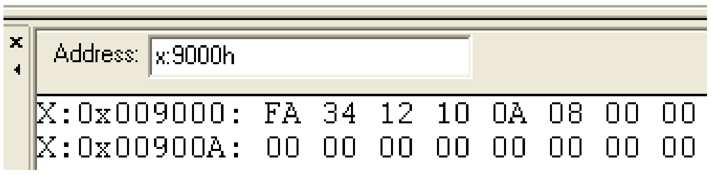
RESULT:

Before Execution: Unsorted Array at 9000h



```
X:0x009000: 12 34 08 FA 0A 10 00 00
X:0x00900A: 00 00 00 00 00 00 00 00
```

After Execution: Sorted Array (Descending order) at 9000h



```
X:0x009000: FA 34 12 10 0A 08 00 00
X:0x00900A: 00 00 00 00 00 00 00 00
```

Results / Conclusion :

The programming experiment was conducted successfully, the programming o/p is observed & the results are neatly tabulated.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 3	Date : / / .

PROGRAM ILLUSTRATING BIT MANIPULATIONS
(Bit manipulation, Boolean & Logical Instructions programs)

Aim 1 : a) Assembly Program Illustrating Logical Instructions (Byte Level)
3 eight bit numbers X, NUM1 & NUM2 are stored in internal data RAM locations 20h, 21h & 22H respectively.
Write an ALP to compute the following:
IF X=0; THEN NUM1 (AND) NUM2,
IF X=1; THEN NUM1 (OR) NUM2,
IF X=2; THEN NUM1 (XOR) NUM2,
ELSE RES =00,
STORE RES AT 23H LOCATION

Algorithm:

1. Point to the data RAM register 20h and store the condition x.
2. Point to 21h and 22h and move the first number to A register.
3. Compare the contents of r1 and perform the operations accordingly.
4. The result will be stored in 23H register.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV A, 20h	
	MOV R1, A	
	MOV A, 21H	
	CJNE R1,#0,CKOR	
	ANL A, 22H	
	SJMP END1	
CKOR:	CJNE R1,#01,CKXOR	
	ORL A, 22H	
	SJMP END1	
CKXOR:	CJNE R1,#02,OTHER	
	XRL A, 22H	
	SJMP END1	
OTHER:	CLR A	
END1:	MOV 23H, A	
HERE:	SJMP HERE	
	END	

RESULT:

1) Before Execution: D: 020H =00, 21=0f, 22 = 12
   After Execution D: 023H = 02
2) Before Execution: D: 020H =01, 21=0f, 22 = 12
   After Execution D: 023H = 1F
3) Before Execution: D: 020H =02, 21=0f, 22 = 12
   After Execution D: 023H = 1D
4) Before Execution: D: 020H =34, 21=0f, 22 = 12
   After Execution D: 023H = 00

Aim 2 : b) 3 eight bit numbers X, NUM1 & NUM2 are stored in internal data RAM Locations 20h, 21h & 22H respectively.
Write an ALP to compute the following:
IF X=0; THEN LSB OF NUM1 (AND) LSB OF NUM2,
IF X=1; THEN MSB OF NUM1 (OR) MSB OF NUM2,
IF X=2; THEN COMPLEMENT MSB OF NUM1
STORE THE BIT RESULT IN RES,
WHERE RES IS MSB OF 23H LOCATIONS

Algorithm:

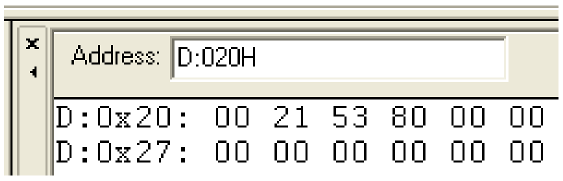
1. Move the condition X (from 20h location) into R0 register.
2. If X=0; then move LSB bit of 21h to carry flag and 'AND' Carry flag with LSB bit of 22h. Go to step5
3. If X=1; then move MSB bit of 21h to carry flag and 'OR' Carry flag with MSB bit of 22h. Go to step5
4. If X=0; then complement MSB bit of 21h and move it to carry flag. Go to step5
5. Store Carry flag at MSB bit of 23h location.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,20H	
	CJNE R0,#0,CK1	
	MOV C, 08H	
	ANL C, 10H	
	SJMP LAST	
CK1:	CJNE R0, #1, CK2	
	MOV C, 0FH	
	ANL C, 17H	
	SJMP LAST	
CK2:	CJNE R0,#2,CK3	
	CPL 0FH	
	MOV C, 0FH	
	SJMP LAST	
CK3:	CLR C	
LAST:	MOV 1FH, C	
HERE:	SJMP HERE	
	END	

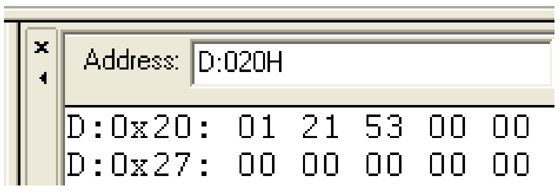
RESULT:

20h = 00 => AND OF LSBs =1 (hence 80 in 23h location)


Address: D:020H

```
D:0x20: 00 21 53 80 00 00
D:0x27: 00 00 00 00 00 00
```

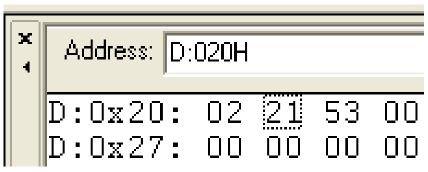
20h = 01 => OR of MSBs = 0 (hence 00 in 23h location)


Address: D:020H

```
D:0x20: 01 21 53 00 00
D:0x27: 00 00 00 00 00
```

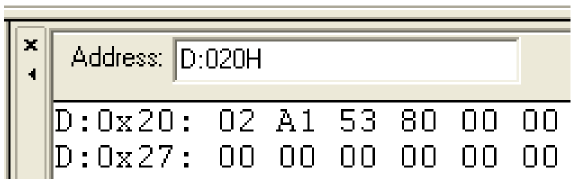
20h = 01 =>complement of MSB of 21h location. Hence 21h is changed to A1 and 23h location has 80h

Before Execution


Address: D:020H

```
D:0x20: 02 21 53 00
D:0x27: 00 00 00 00
```

After Execution


Address: D:020H

```
D:0x20: 02 A1 53 80 00 00
D:0x27: 00 00 00 00 00 00
```

Results / Conclusion :

The programming experiment was conducted successfully, the programming o/p is observed & the results are neatly tabulated.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 4	Date : / / .

COUNTERS

Aim 1 : a) Write an ALP to implement (display) an eight bit up/down BCD counters on watch window.

Note: To run this program, after selecting DEBUG session in the main menu use

View-> Watch& call Stack window, in the Watches select watch 1(or 2) and press F2 and enter a (for accumulator A)

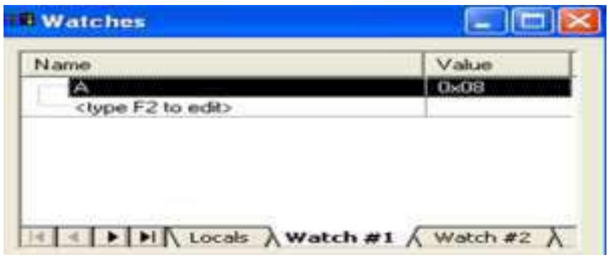
Algorithm:

1. Move 00 to A register
2. Call the delay subroutine for 1 second (in delay program move FFH to registers R1, R2 and R3, loop and decrement until 0).
3. Increment A register(add 99h for down counter)
4. Decimal adjust accumulator for the BCD up/down counter.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV a,#00	
BACK:	ACALL DELAY	
	ADD A,#99H	
	DA A	
	JNZ BACK	
HERE:	SJMP HERE	
DELAY:	MOV R1,#35H	
DECR1:	MOV R2,#0FFH	
DECR:	MOV R3, #0FFH	
	DJNZ R3,$	
	DJNZ R2, DECR	
	DJNZ R1, DECR1	
	RET	
	END	

RESULT: Accumulator A is incremented in BCD from 00, 01, 02…09, 10, 11,…99.



Name	Value
A	0x08
<type F2 to edit>	

Aim 2 : b) Write an ALP to implement (display) an eight bit up/down BCD counters by using timer delay.

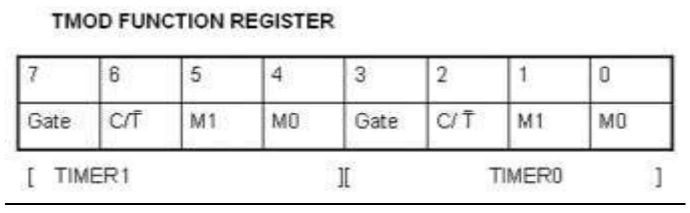
Algorithm:

1. Set up timer0 in mode 2 operation
2. Load TH1 with 118 to generate an interrupt every 0.05msec.
3. Reset registers a, r1 & r0.
4. Repeat step 4 continuously
5. On interrupt; ISR at 000B location goes to step 6
6. Disable timer0
7. Update r1 & r0
8. Check if 20000 interrupts (=1 sec) over. Yes –increment accumulator a.
9. Enable timer & return from ISR.

Program:

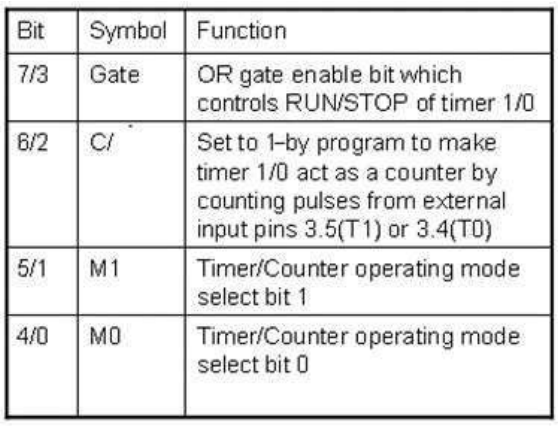
Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 0BH	
	SJMP ISR	
	ORG 30H	
	MOV A, #00	
	MOV R0,#00	
	MOV R1,#00	
	MOV TMOD, #02H	
	MOV TH0, #118	
	MOV IE, #82H	
	SETB TCON.4	
HERE:	SJMP HERE	
ISR:	CLR TCON.4	
	INC R1	
	CJNE R1,#100,SKIP	
	MOV R1,#00	
	INC R0	
	CJNE R0,#200,SKIP	
	MOV R0,#00H	
	INC A	
SKIP:	SETB TCON.4	
	RETI	
	END	

RESULT **:** Accumulator A is incremented in hex from 00, 01,02…09,0A, 0B, …, 0F, 10, 11, …FF every 1 second (for 33MHz clock setting & every 3 seconds for 11.0592MHz)

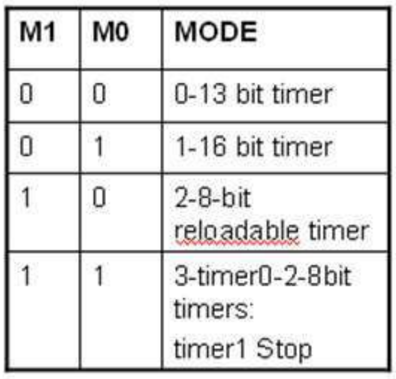

TMOD FUNCTION REGISTER

7	6	5	4	3	2	1	0
Gate	$C/\overline{T}$	M1	M0	Gate	$C/\overline{T}$	M1	M0

[TIMER1][TIMER0]



Bit	Symbol	Function
7/3	Gate	OR gate enable bit which controls RUN/STOP of timer 1/0
6/2	C/	Set to 1-by program to make timer 1/0 act as a counter by counting pulses from external input pins 3.5(T1) or 3.4(T0)
5/1	M1	Timer/Counter operating mode select bit 1
4/0	M0	Timer/Counter operating mode select bit 0

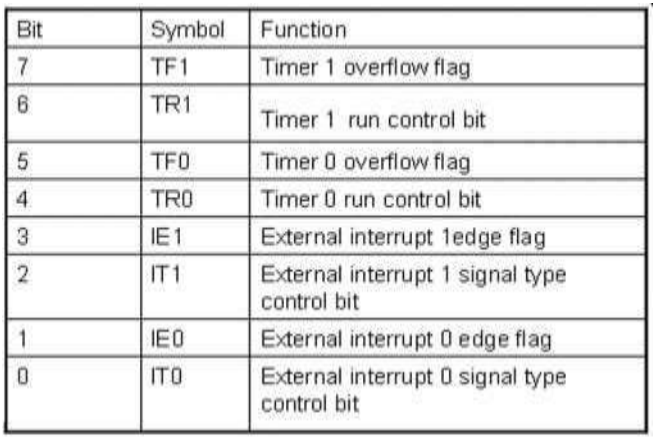


M1	M0	MODE
0	0	0-13 bit timer
0	1	1-16 bit timer
1	0	2-8-bit reloadable timer
1	1	3-timer0-2-8bit timers: timer1 Stop

TCON Function Register

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Interrupt	Address(Hex)
IE0	0003
TF0	000B
IE1	0013
TF1	001B
Serial	0023



Bit	Symbol	Function
7	TF1	Timer 1 overflow flag
6	TR1	Timer 1 run control bit
5	TF0	Timer 0 overflow flag
4	TR0	Timer 0 run control bit
3	IE1	External interrupt 1edge flag
2	IT1	External interrupt 1 signal type control bit
1	IE0	External interrupt 0 edge flag
0	IT0	External interrupt 0 signal type control bit

To get 1sec delay

1/0.05msec = 200*100 in the ISR

(Assuming 33 MHz crystal frequency.

For 11 MHz, the calculations change).

Timer delay = 12 * (257-delay)/frequency

Timer delay=0.05 msec

Delay = 256-((timer delay * frequency)/12)

= 256-(0.05*10 $^{-3}$ * 33*10^{6})/12

= 256-137.5

= 118.5 //loaded in TH0

Results / Conclusion :

The programming experiment was conducted successfully, the programming o/p is observed & the results are neatly tabulated.

Applications:

1.

2.

3.

4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 5	Date : / / .

CONVERSION PROGRAMS

Aim 1 : a) Write an ALP to implement decimal to hex conversion.

Algorithm:

1. Move the decimal data to be converted from external memory 40h to accumulator.
2. AND A reg with 0f0h and obtain the upper MSB of the decimal digit and swap the LSB and MSB of accumulator to bring the same to units place.
3. Move 0ah to B register and multiply with A reg to convert to hex value, store the converted tens value in r1
4. Get the LSB of the decimal number and add to the converted tens value
5. Point to the next memory location and store the result (hexadecimal).

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV DPTR,#40H	
	MOVX A, @DPTR	
	ANL A, #0F0H	
	SWAP A	
	MOV B,#0AH	
	MUL AB	
	MOV R1,a	
	MOVX A,@DPTR	
	ANL A,#0FH	
	ADD A,R1	
	INC DPTR	
	MOVX @DPTR,A	
HERE:	SJMP HERE	
	END	

***RESULT**:*	Before execution- X: 0040H = 45 (Decimal/BCD)
	After Execution: X: 0041h = 2D (hex value)

Aim 2 : b) Write an ALP to implement hex to decimal conversion

Algorithm:

1. Move the hex data to be converted to accumulator.
2. Move 10 to B register and divide with A reg to convert to ASCII value
3. Store the converted LSB value in r7
4. Repeat the step 2 to obtain the converted MSB value
5. Store the same in r6

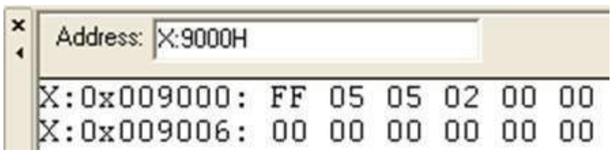
Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30h	
	ORG 30h	
	MOV DPTR,#9000H	
	MOVX A,@DPTR	
	MOV B,#10	
	DIV AB	
	INC DPTR	
	XCH A,B	
	MOVX @DPTR, A	
	XCH A,B	
	MOV B,#10	
	DIV AB	
	INC DPTR	
	XCH A,B	
	MOVX @DPTR, A	
	XCH A,B	
	INC DPTR	
	MOVX @DPTR, A	
HERE:	SJMP HERE	
	END	

RESULT : 9000H – FF (HEX NUMBER)

9001 to 9003 – unpacked BCD number (decimal) - 5, 5, 2

(i.e., 255 stored Lower digit first)


Address: X:9000H

```
X:0x009000: FF 05 05 02 00 00
X:0x009006: 00 00 00 00 00 00
```

Results / Conclusion :

The programming experiment was conducted successfully, the programming o/p is observed & the results are neatly tabulated.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 6 Date : / / .

SERIAL DATA TRANSMISSION with variable baud rate–8051

Aim : Write a program illustrating serial ASCII data transmission (data-BHARAT). Conduct an experiment to configure 8051 microcontroller to transmit characters (BHARAT) to a PC using the serial port and display on the serial window.

Algorithm:

1. Initialize timer 1 to operate in mode 2 by loading TMOD register.
2. load TH1 with -3 to obtain 9600 baud.
3. Initialize the asynchronous serial communication transmission (SCON) register.
4. Start timer1 to generate the baud rate clock.
5. Transmit the characters "BHARAT" by writing into the SBUF register and waiting for the TI flag.

Program:

Label	Mnemonic/Operands	Comments
	ORG 0000H	
	SJMP 30H	
	ORG 30H	
	MOV R0,#05H	
	MOV DPTR, #300H	
	MOV TMOD,#20H	
	MOV TH1, #-3	//-3=FD loaded into TH1 for 9600 baud, 11.0592MHz.
	MOV SCON, #50H	
	SETB TR1	
AGAIN:	CLR A	
	MOVC A, @A+DPTR	
	JZ BACK	
	ACALL TRANS	
	INC DPTR	
	SJMP AGAIN	
BACK:	SJMP BACK	
TRANS:	MOV SBUF, A	
HERE:	JNB TI, HERE	
	CLR TI	
	RET	
MYDATA:	ORG 300H	
	DB "BHARAT",0	
	END	

Note: To use result of this program, after selecting DEBUG session in the main menu use **View-> serial window #1**. On running & halting the program, the data is seen in the serial window.

RESULT: “BHARAT” is printed on the serial window each time the program is executed.

Theory: In serial transmission as opposed to parallel transmission, one bit at a time is transmitted. In serial asynchronous transmission, the data consists of a Start bit (high), followed by 8 bits of data to be transmitted and finally the stop bit. The byte character to be transmitted is written into the SBUF register. It transmits the start bit. The 8-bit character is transferred one bit at a time. The stop bit is transferred. After the transmission, the TI flag = 1 indicating the completion of transmission. Hence in the subroutine wait until TI is set. Later clear the TI flag and continue with transmission of the next byte by writing into the SBUF register. (The program can also be written in interrupt mode). The speed of the serial transmission is set by the baud rate which is done with the help of timer 1. (Refer Ayala). Timer1 must be programmed in mode 2 (that is, 8-bit, autos reload).

Baud rate Calculation:

Baud Rate = Crystal freq/ (12*32)
= (11.0592MHz)/(12*32)
= 28800

To get 9600, 28800/3 is obtained by loading timer1 with -3 (i.e., FF – 3 = FD) for further clock division. For 2400 baud rate, 28800/12 => -12 = F4 in TH1.

Results / Conclusion :

The experiment was conducted successfully, the o/p is observed & the results are neatly tabulated and the conclusions are drawn.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Hardware Interfacing Experiments

<u>Features of Embedded C :</u>

- C is a simple programming language and so very easy to code.
- Embedded C has most features of C-language with more stress on certain bit manipulative instructions.
- This feature makes it easy to write program for µC and µP.
- Keil is a versatile software with a cross compiler that will convert the C program to assembly language and thus the program can be executed on the desired target (say 8051).
- Some of the bit manipulative instructions used are

Symbol	**Operation**
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
>>	Shift right
<<	Shift left
^	Dot operator

Experiment No. : 7 Date : / / .

Dual DAC Interface to generate different types of waveforms

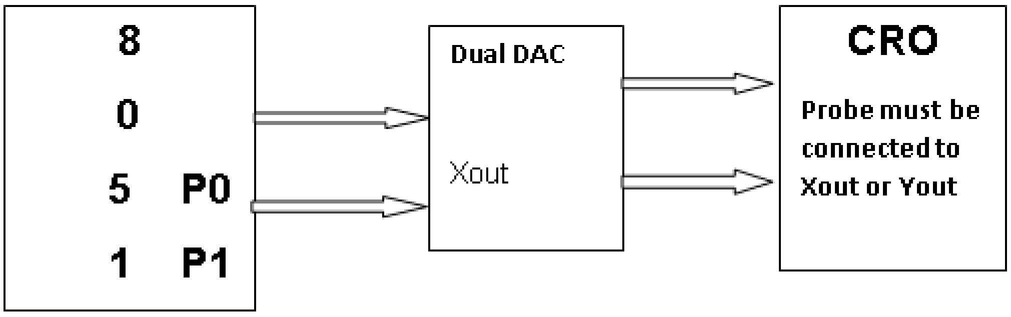
Aim : a) Implementation of DAC 0808 interface to 8051 to generate square, triangular, ramp waveforms. Dual DAC Interface to generate .....

a. Square waveform

b. Triangular Waveform

c. Ramp waveform

d. Sine waveform



a). *Algorithm for Square wave generation:*

- Let initial, amplitude of the square wave be 2.5v (7F) and frequency count 100.
- Output the values 00h (0ff) and 7fh (on) Values through P0.
- If amplitude key is pressed then increase the voltage in steps of 0.15v (8).
- If the frequency key is pressed increment the count in steps of 50. If the count exceeds 1000 reset it back to 100.
- Every time amplitude and frequency changes output the value thro P0 and note the waveform on CRO.

Theory: Majority of the integrated circuits of DAC use the R/2R method since it can achieve higher degree of precision. The basic criterion for judging a DAC is its resolution, which is a function of the binary inputs. The common ones are 8, 10 and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog levels is equal to 2^n, where n is number of data nit inputs. Therefore the 8-input DAC such as DAC0800 provides 256 discrete voltage (or current) levels of output.

The digital inputs are converted to current (I_{out}) and by connecting resistor or op-amp to the I_{out} pin, we convert the result into voltage. The total current provided by the Iout pin is a function of the binary numbers at the D0-D7 inputs of the DAC and the reference current (I_{ref}), and is as follows:

I_{out}= I_{ref} (D7/2 + D6/3 + D5/8 + D4/16 + D3/32 + D2/64 + D1/128 + D0/256) Where D0 is the LSB and D7 is the MSB of the inputs, and Iref is the input current that must be applied to pin 14. The Iref current is generally 2mA. Some DAC also use zener diode (LM336) which overcomes any flu8ctuations associated with the power supply. If I_{ref} is 2mA then when all inputs are high the maximum current is 1.99mA.

Driver Circuit Description:

The Dual DAC interface can be used to generate different waveforms using microcontroller. There are two 8-bit analog to digital converters provided based on DAC0800. The digital inputs to these DACs are provided through the Port 0 and Port 1. The analog output from the DAC is given to operational amplifier which acts as current to voltage converter and isolator between CRO circuit and DAC chip. The output of the op-amp is connected to Xout and Yout points on board from which the waveforms can be observed on CRO. Two 10k Ohm pots are provided for the offset balancing of op-amps. The reference voltage required for the DAC is obtained from onboard voltage regulator uA723. The voltage generated by this regulator is about 8V. The output of the DAC vary from 0 V to 5V corresponding to values between 00 to FF respectively.

Installation:

- The interface module has a 26-pin connector at one edge of the card which is connected to Microcontroller board through FRC (Flat Ribbon Cable) connector.
- External power supply of +12, -12 and GND are connected to points marked through 4-pin connector provided.

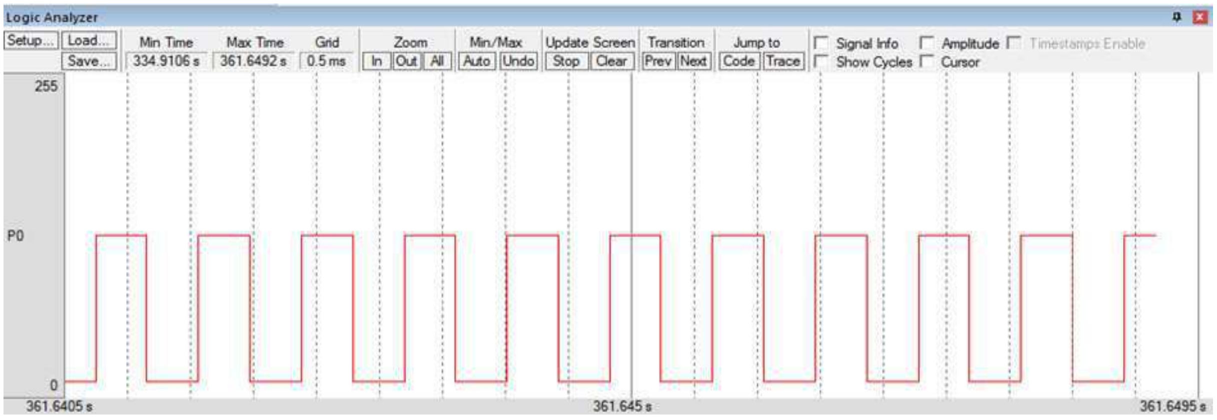
Applications: Sound card, CD players, Digital music players etc…

Program for square wave:

```
#include <REG51xD2.H>
sbit Amp = P3^3;                /* Port line to change amplitude */
sbit Fre = P3^2;                /* Port line to change frequency */
void delay (unsigned int x)     /* delay routine */
{
    for (;x>0;x--);
}
main()
```

```
{
    unsigned char on = 0x7f,off=0x00;
    unsigned int fre = 100;
    while(1)
        {
            if(!Amp)                  /* if user choice is to change amplitude */
            {
                while(!Amp);          /* wait for key release */
                on+=0x08;             /* Increase the amplitude */
            }
        if(!Fre)     /* if user choice is to change frequency */
            {
            if(fre > 1000)            /* if frequency exceeds 1000 reset to default */
            fre = 100;
            while(!Fre);              /* wait for key release */
            fre += 50;
            }                         /* Increase the frequency */
        P0=on;                        /* write amplitude to port */
        P1=on;
        delay(fre);
        P0 = off;                     /* clear port */
        P1 = off;
        delay(fre);
        }
            }
```

Simulation output:



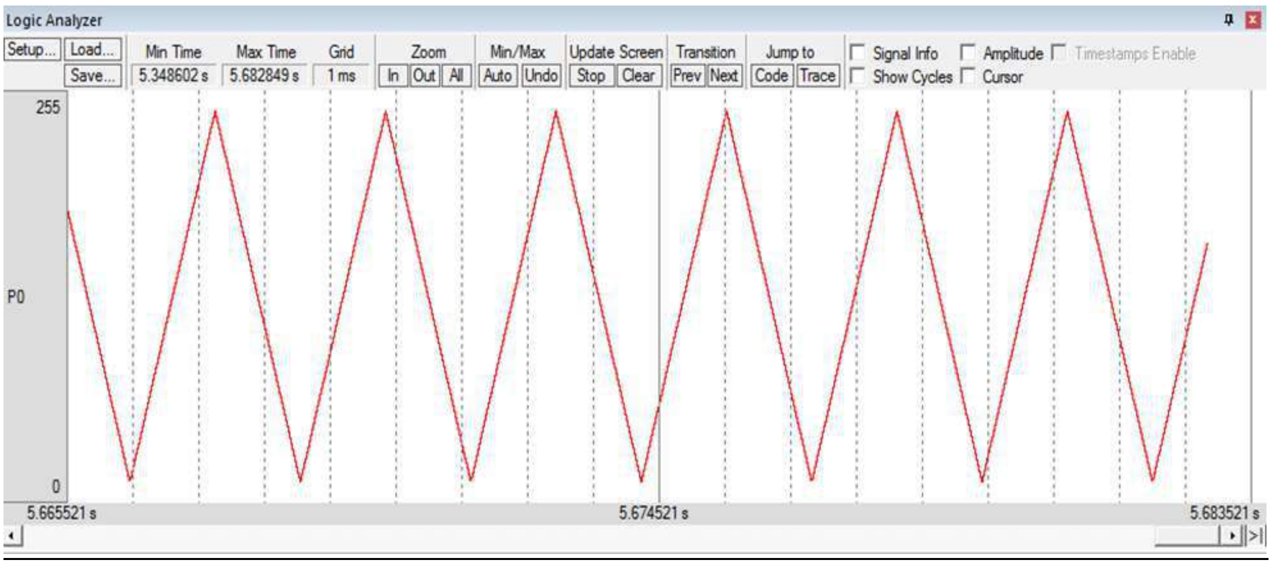
b) <u>Algorithm for Triangular wave generation:</u>

- Output the initial value 00 through P0.
- Increment it in steps of 1 until a count value of FFh (5V) is reached. Every time repeat step 1.
- Decrement it in steps of 1 until a zero value is reached and repeat step 1.

Program for triangular wave:

```
#include <REG51xD2.H>
main()
{
    unsigned char i=0,slope=1;
    P0 = 0x00;                  /* P0 as Output port */
    while(1)
    {
        for(i=0;i<0xfe;)        /* Generate ON pulse */
        {
                P1 = i;
                P0 = i;
                i=i+slope;
        }
        for(i=0xfe;i>0x00;)     /* Generate OFF pulse */
        {
                P0 = i;
                P1 = i;
                i=i-slope;
        }
    }
}
```

Simulation output:



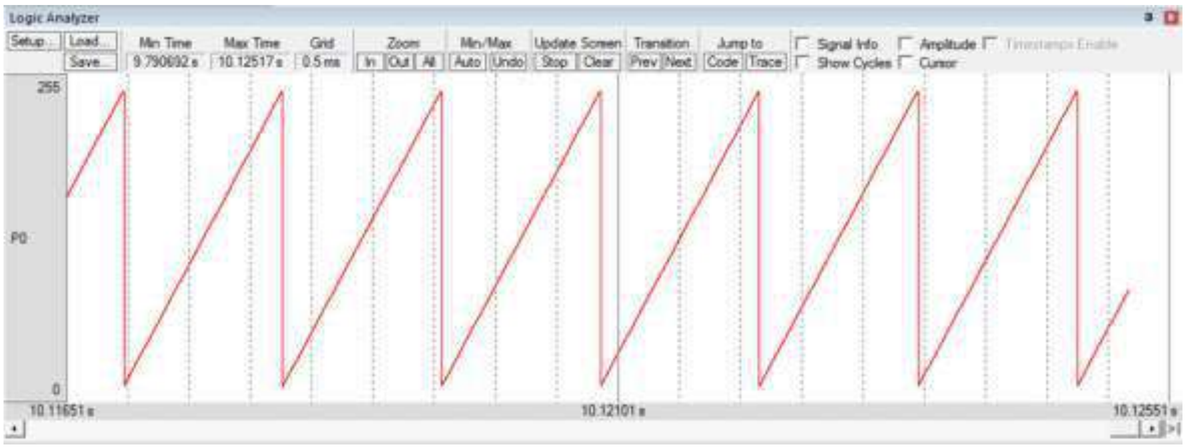
c) **Algorithm for Ramp wave generation**

- Output the initial value 00 through P0.
- Increment it in steps of 1 until a count value of FFh (5V) is reached. Every time repeat step 1.
- Repeat step 1 & 2 continuously.

Program for Ramp waveform

```
#include <REG51xD2.H>
main ()
{
    Unsigned char i=0,slope=1,rising=1;
    P0 = 0x00;                              /* P0 as Output port */
    while (1)
    {
    If(rising==1)
        {
                for(i=0;i<0xfe;)            /* Generate ON pulse */
                {
                        P1 = i;
                        P0 = i;
                        i=i+slope;
                }
        }
    else
        {
                for(i=0xfe;i>0x00;)         /* Generate OFF pulse */
                {
                        P0 = i;
                        P1 = i;
                        i=i-slope;
                }
            }
        }
     }
```

Simulation output:



Results / Conclusion :

The programming experiment was conducted successfully, the o/p is observed & the results are neatly tabulated, conclusions are drawn.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 8	Date : / / .

STEPPER MOTOR INTERFACE TO 8051

Aim : Perform an experiment to interface a stepper motor to the 8051 microcontroller.

Description

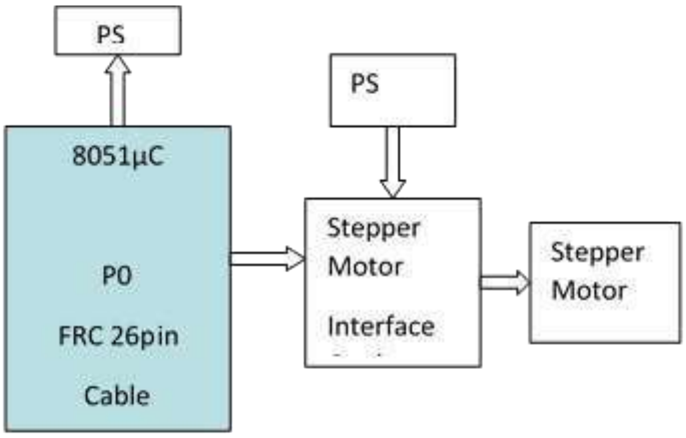
- Unlike DC motor Stepper motor rotates in steps.
- Programmatically following parameters can be controlled
  - Angle of rotation
  - Direction of rotation
  - Speed of rotation (RPM)
- Stepper motor has 4 coils which forms the stator and a central rotor.
- Rotation depends on excitation of stator coils.

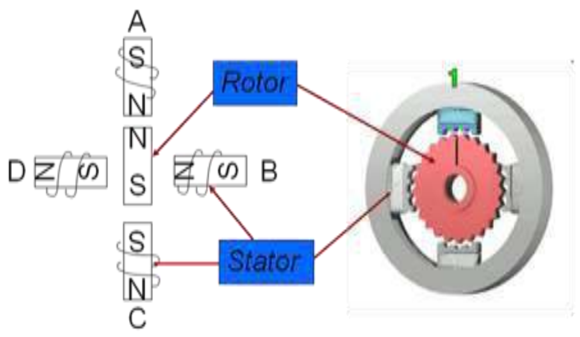
step	coil A	coil B	coil C	coil D
1	0	0	0	1
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1

Anyone of these values forms the initial value. To get 360o revolution 200 steps are required. Step angle= 360^o /200 = 1.8^o (difference between 2 teeth).

Algorithm for Stepper Motor

- Configure P0 as output.
- Apply the initial excitation of 11 to motor coils through P0.
- For clockwise motion -Rotate right once the excitation and repeat step 2.
- For anticlockwise motion -Rotate left once the excitation and repeat step 2.





Theory:

A stepper motor is a device that translates electrical pulses into mechanical movement. The stepper motor shaft moves in a fixed repeatable increment, which allows precise angle control. This repeatable fixed movement is possible as a result of basic magnetic theory where poles of the same polarity repel and opposite polarity attract. The direction of the rotation is dictated by the stator poles. The stator poles are determined by the current sent through the wire coils. As the direction of the current is changed, polarity is also changed causing the reverse motion of the rotor.

Step angle : This depends on the number of teeth on the stator and the rotor. Step angle is the minimum degree of rotation associated with the single step.

We are using a stepper motor with 50 teeth on rotor and 4 on stator, hence the step angle is calculated as

Step Angle = 360/ (No. of teeth on rotor × No of teeth on Stator)
= 360/ (50 × 4)
= 1.8

Therefore steps per revolution are 200.

Steps per second and rpm relation.

Steps/sec = (rpm x Steps per revolution) / 60

Drive sequences:

4 step sequence: 1001,1100,0110,0011
8 Step sequence: 1001, 1000,1100,0100,0110,0010,0011,0001
Wave drive 4 step sequence: 1000,0100,0010,0001
Types of stepper motor:

- Permanent magnet (PM)
- Variable reluctance (VR)

Comparison of different types (based on phase) of stepper motor are :

Parameter	Universal	Unipolar	Bipolar
Number of connections	8	6	4
Modes	All 3	2 (Uni / Bi)	Only Bi
Extra circuitry	-	-	H-bridge
Operational current	Low	Low	High
Holding torque	Low	Low	High

Construction:

Stepper motors commonly have permanent magnet rotor (also referred as shaft) surrounded by a stator. Stepper motor have four stator windings that are paired with the centre tapped common, this type is commonly referred as four phase or Unipolar stepper motor. The centre tap allows change of the current direction in each of two coils when winding is grounded thereby resulting in polarity change of stator. The stepper motor used has total of 6 leads, 4 leads represent 4 stator winding and 2 common for the centre tapped leads.

Driver Circuit Description:

The stepper motor interface uses 4 transistor pairs (SL100 & 2N3055) in a Darlington pair configuration. Each Darlington pair is used to excite the particular winding of the motor connected to 4 pin connector on the interface. The inputs to these transistors are from the Microcontroller board. Lower nibble of Port 0 i.e. P0.0, P0.1, P0.2, p0.3 are the four lines brought out of the 26 pin FRC male connector (J7) on the interface module. The freewheeling diodes across each winding protect transistor from switching transients.

Installation:

- The interface has two 3-pin and one 4pin connectors.
- Plug in 4-pin polarized connector of the motor to interface and the 3-pin connector of the motor to 3-pin connector of the interface marked as “WHT BLK”.
- Connect 3-pin female connector of the stepper motor power supply to the connector of the interface marked as “GND +5/12V”.
- Connect the 26-pin FRC on the interface module to J7 of controller kit.

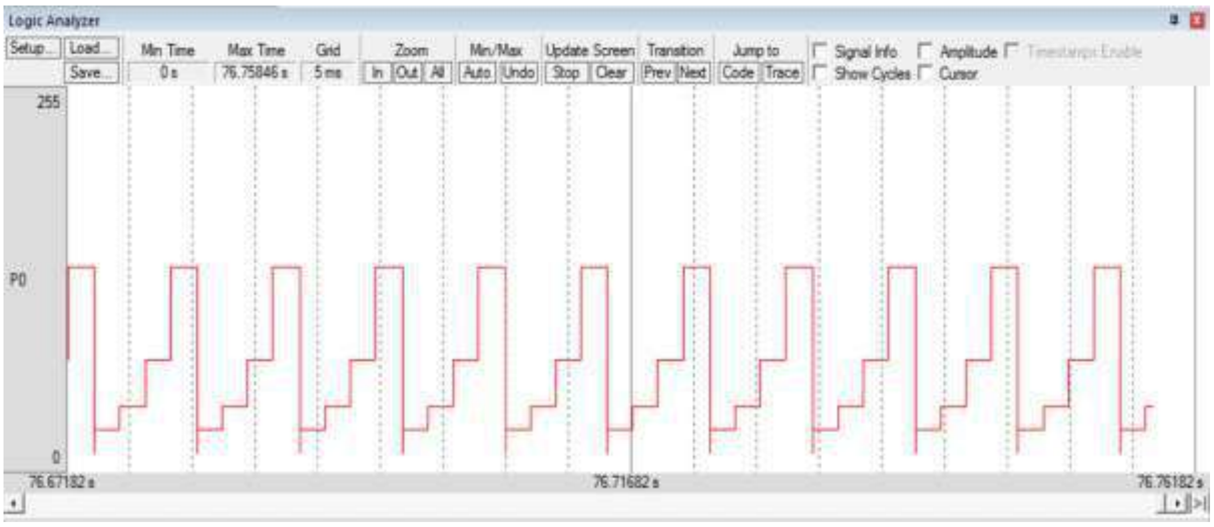
Applications:

Card reader, dot matrix printers, Hard disk drive (HDD), Floppy disk drive (FDD), CD/ DVD drive, Clocks to rotate hands etc...

//Program for stepper motor interface :

```
#include <REG51xD2.H>
void delay (unsigned int x)                /* Delay Routine */
    {
        for(;x>0;x--);
        return;
    }
Main ( )
{
    unsigned char Val, i;
    P0=0x00;
    Val = 0x11;
    for (i=0;i<4;i++)
    {
        P0 = Val;
        Val = Val<<1;   /* Val= Val>>1; for clockwise direction*/
        delay (500);
    }
}
```

Simulation output :



Min Time 0 s
Max Time 76.75846 s
Grid 5 ms

Results / Conclusion :

The programming experiment was conducted successfully, the o/p is observed & the results are neatly tabulated, conclusions are drawn.

Applications:

1.

2.

3.

4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 9	Date : / / .

DC MOTOR INTERFACE TO 8051

Aim : Perform an experiment to interface a DC motor to the 8051 microcontroller.

Algorithm for DC motor interface :

- Configure P0, P1 as output port and P3 as input port.
- Let initially the motor rotate with half speed count 7fh.
- If "INR" button is pressed reduce the count because the speed is inversely proportional to count.
- If "DEC" button is pressed increase the count.

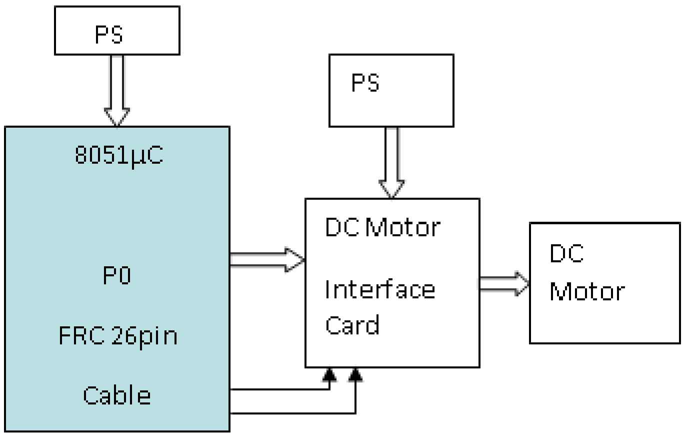


Fig. : Block-diagram of the interfacing of a DC motor to a microcontroller

Theory:

Direct current (DC) motor is another widely used device that translates electrical to mechanical movement. In DC motor we have only + and – leads. Connecting from DC voltage source moves the motor in one direction and by reversing the polarity, the motor will move in opposite direction. DC motors have two rpms : no load and loaded which will be indicated on data sheet specifications. The normal voltage rating varies from 1V to 150V and current rating varies from 25mA to few amperes. The DC motor follows three important relations shown below.

Rpm = k / load; at constant current

Rpm = k × current; at constant load

Current = k × load; at constant Rpm

where, k is constant of proportionality.

DC motor speed can be varied using PWM (Pulse Width Modulation) technique. By changing the width of the pulse applied to the DC motor the speed of the motor is varied. Even though the amplitude of the voltage is same as the width of the ON time of pulse increases speed also increases.

Installation:

- Connect DC motor to Microcontroller board through FRC cable to J7 (26-Pin connector).
- AC main Power supply is independently connected to DC motor.

Application:

1. CPU fan, Processor cooling fan etc…

Program for DC motor :

```
#include <REG51xD2.H>
sbit  inr= P3^2;    //speed increment switch
sbit  dcr= P3^3;   //speed decrement switch
main()
{
    unsigned char i=0x80;
    P0 = 0x7f;         /*Run the motor at half speed.*/
    while (1)
    {
        if (!inr)
        {
            while (!inr);
            if(i>10)
            i=i-10;     //increase the DC motor speed
        }
        if(!dcr)
        {
            while(!dcr);
            if(i<0xf0)
            i=i+10;     //decrease the DC motor speed
        }
        P0=i;
    }
}
```

Results / Conclusion :

The programming experiment was conducted successfully, the o/p is observed & the results are neatly tabulated, conclusions are drawn.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 10	Date : / / .

ALPHANUMERIC LCD PANEL INTERFACE TO 8051

Aim : Perform an experiment to interface an alphanumeric LCD panel to the 8051 microcontroller.

Theory: A liquid crystal display (LCD) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs).LCD panels have in built refreshing controller relieving CPU from the task. LCDs are more energy efficient, and offer safer disposal, than CRTs. Its low electrical power consumption enables it to be used in battery-powered electronic equipment. It is an electronically-modulated optical device made up of any number of pixels filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in color or monochrome.

Pin Description:

Pin	Symbol	I/O	Description
1	V_{ss}	--	Ground
2	V_{cc}	--	+5V Power supply
3	V_{ee}	--	Power supply to control contrast
4	RS	I	‘0’ to select command register ; ‘1’ to select data register
5	R/W	I	‘0’ for write ; ‘1’ for Read
6	E	I/O	Enable
7-14	DB0 – DB 7	I/O	The 8-bit data bus

LCD Command codes:

Hex Code	Instruction description	Hex Code	Instruction description
01	Clear display screen	0E	Display ON, cursor blinking
02	Return home	0F	Display ON, cursor blinking
04	Decrement cursor (Shift cursor to left)	10	Shift cursor position to left
06	Increment cursor (Shift cursor to right)	14	Shift cursor position to right

05	Shift display right	18	Shift the entire display to left
07	Shift display left	1C	Shift the entire display to right
08	Display OFF, cursor OFF	80	Force cursor to beginning of 1st line
0A	Display OFF, cursor ON	C0	Force cursor to beginning of 2nd line
0C	Display ON, cursor OFF	38	2 lines and 5x7 matrix

Driver Circuit Description :

LCD accepts characters in ASCII format. Character display font in LCD module is dot matrix i.e. each character in LCD module can be represented by 7x5 matrix. This module is built over 16x1 LCD in which the display data RAM address for the first line is from 00H to 14H and for second line it is 29H to 3CH

LCD module has got an automatic reset which is critically dependent upon power supply voltage. Voltage has to rise from 0.2V to 5V within 10 to 15 ms for LCD to reset. Since this is not accurate; it can also be reset during initialization. To reset 30H has to be sent 3 times with some delay, busy flag of LCD module is set while LCD is resetting, during this time data can't be written on to the LCD.

- Port 2 of microcontroller is connected to D0 to D7 pins of LCD module.
- Control signals RS, R/W and E are connected to P3.7, P3.6 and P3.5 respectively.

Program :

```
#include<reg51xd2.h>
//Function prototype declaration
void lcdcmd (unsigned char value);
void msdelay (unsigned int itime);
void lcddata (unsigned char value);
unsigned int i;
sfr ldata=0XA0;    //0xA0 is address of Port 2
sbit rs = P3^7;    //rs -> Register Select, 0 – Command Register: 1-Data
Register
sbit rw = P3^6;    //rw -> Read / Write, 0 – Write : 1- Read
sbit en = P3^5;    //en -> Enable
void main()
{
        lcdcmd(0x38);       //Defines character matrix i.e 7x5
        msdelay(250);       //Delay is introduced as LCD need time to respond
        lcdcmd(0x0E);       //Display on cursor blinking
        msdelay(250);
        lcdcmd(0x01);       //Clear display
        msdelay(250);
        lcdcmd(0x06);       //Increment cursor
        msdelay(250);
        lcdcmd(0x86);
        msdelay(250);
        lcddata('E');
        msdelay(250);
        lcddata('N');
        msdelay(250);
        lcddata('C');
}
void lcdcmd(unsigned char value)
{
        ldata=value;        //Information bits are copied to Port 2
        rs=0;               //Selecting Command register
        rw=0;               //Opted for Write operation
        en=1;
        msdelay(1);
        en=0;
        return;
}
void lcddata(unsigned char value)
{
        ldata=value;
        rs=1;               //Selecting Data Register
        rw=0;
        en=1;
        msdelay(1);
        en=0;
        return;
}

void msdelay(unsigned int itime)
{
        unsigned int i,j;
        for(i=0;i<itime;i++)
                for(j=0;j<1275;j++);
}
```

Results / Conclusion :

The programming experiment was conducted successfully, the o/p is observed & the results are neatly tabulated, conclusions are drawn.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

Experiment No. : 11 Date : / / .

ELEVATOR INTERFACE TO 8051 MICROCONTROLLER

Aim : Perform an experiment to interface an ELEVATOR to the 8051 microcontroller.

Algorithm for elevator interface :

- Read the floor request through input port P1.
- If the current floor and requested floor are the same no change light up the corresponding LED through P0.
- If the requested floor greater than current moving up of the lift is indicated by glowing of LED's from current floor to the requested.
- If the requested floor lesser than current moving down of the lift is indicated by glowing of LED's from current floor to the requested.

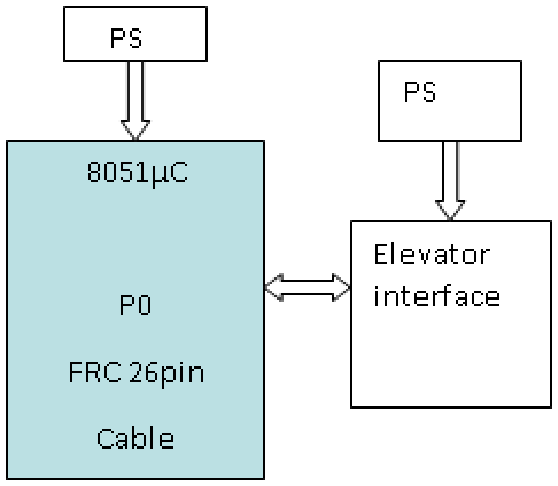
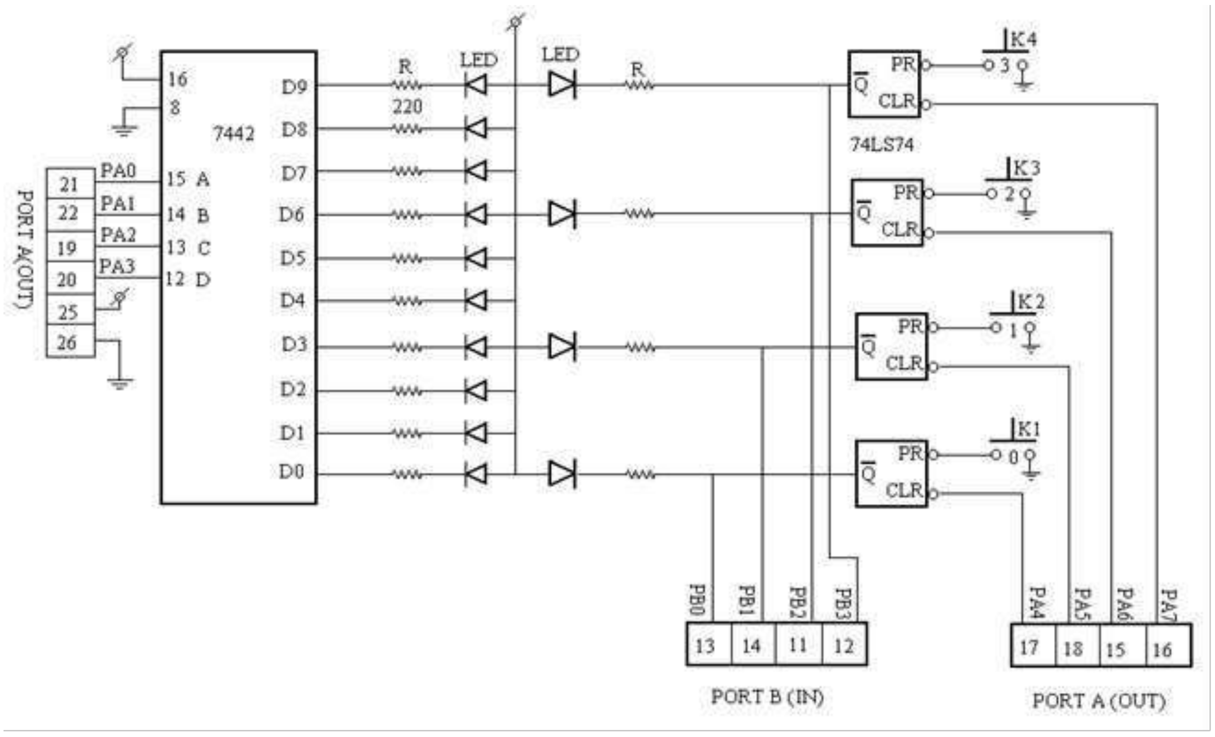


Fig. : Block diagram of an elevator interface to the microcontroller

NOTE:

- All active low Signals.
- Also there are two types of LEDS here.
- One is Floor LED which is controlled by Flip-Flop in turn controlled by Port 0 higher 4 bits.
- Totally there are 4 Floor LEDs (Fl0 Fl1 Fl2 Fl3).



One more type of LED's are Small LED's on left side of diagram. They are just intermediate LEDs which are made to glow as the elevator moves from one floor to another. These LEDS are controlled by ABCD which acts as a decoder.

ABCD= 0000 means Small LED 0 for D0

= 0001 means Small LED 1 for D1
……….
= 1001 means Small LED 9 for D9

Flip flops are cleared to make Floor LEDs stop glowing.

Why the 0xff, 0x00, 0x03, 0xff, 0x06, 0xff, 0xff, 0xff, 0x09 in Flr array ????

0x00-0th floor; 0x03-1st floor; 0x06 2nd floor; 0x09 3rd floor; all other filled with ff just to make it 9 byte array. 0xff do not represent anything.

Similarly for FClr array

Port 0 :

Flipflop 3rd floor clear	Flipflop 2nd floor clear	Flipflop 1st floor clear	Flipflop 0th floor clear	D	C	B	A

Port 1

				FloorLed 3rd floor control	FloorLed 2nd floor control	FloorLed 1st floor control	FloorLed 0th floor control

Theory:

This interface simulates the control and operation of an elevator. Four floors are assumed and for each floor a key and a corresponding LED indicator are provided to serve as request button and request status indicators. The elevator itself is represented by a column of ten LEDs. The motion of elevator is simulated by turning on successive LEds one at a time. The delay between turning OFF one LED and turning ON the next LED indicates speed of elevator. The request status information is read through lower nibble of Port1 and elevator motion control is done through Port0.

Driver Circuit Description:

The interface has 4 keys, marked 0, 1, 2, and 3 representing the request button at the 4 floors. Pressing of a key, cause a corresponding flip flop to be set. The output of the Flip flop can be read through lower nibble of Port1 (P1.0, P1.1, P1.2 and P1.3) also status of these signals is reflected by the set of 4 LEds. The flip flops can be reset (LEDs are cleared) through higher nibble of Port0 (P0.4, P0.5, P0.6 and P0.7). A column of 10 LEds, representing elevator is controlled through lower nibble of Port0 (P0.0, P0.1, P0.2 and P0.3). These port lines are fed to the input of the decoder 7442 whose outputs are used to control the ON/OFF status of the LEDs which simulate the motion of the elevator.

Installation:

- The 26 pin connector is connected to J7 of Microcontroller module through FRC cable.
- No external power supply is required as the needed power is taken through FRC cable itself.

Program for Elevator

```
#include <REG51F.H>
void delay(unsigned int);
main()
{
    unsigned char Flr[9] =  {0xff,0x00,0x03,0xff,0x06,0xff,0xff,0xff,0x09};
    unsigned char FClr[9] = {0xff,0x0E0,0x0D3,0xff,0x0B6,0xff,0xff,0xff,0x79};
    unsigned char ReqFlr,CurFlr = 0x01,i,j;
    P0 = 0x00;
    P0 = 0x0f0;
    while(1)
    {
        P1 = 0x0f;
        ReqFlr = P1 | 0x0f0;
        while(ReqFlr == 0x0ff)
        ReqFlr = P1 | 0x0f0;      /* Read Request Floor from P1 */
        ReqFlr = ~ReqFlr;
        if(CurFlr == ReqFlr)              /* If Request floor is equal to Current
    Floor */
        {
            P0 = FClr[CurFlr];                    /* Clear Floor Indicator */
            continue;
        }                     /* Go up to read again */
        else if(CurFlr > ReqFlr)     /* If Current floor is > request floor */
        {
            i = Flr[CurFlr] - Flr[ReqFlr];                /* Get the no of floors to
        travel */
            j = Flr[CurFlr];
            for(;i>0;i--)                /*Move the indicator down */
            {
                delay(25000);
            }
        }
        else                     /* If Current floor is < request floor */
        {
            i = Flr[ReqFlr] - Flr[CurFlr]; /* Get the no of floors to travel */
            j = Flr[CurFlr];
            for(;i>0;i--)               /*      Move the indicator Up */
            {
                P0 = 0x0f0 | j;
                j++;
                delay(25000);
            }
        }
    CurFlr = ReqFlr;                          /* Update Current floor */
    P0 = FClr[CurFlr];              /* Clear the indicator */
    }
}
```

```
void delay(unsigned int x)
{
    for(;x>0;x--);
}
```

Results / Conclusion :

The programming experiment was conducted successfully, the o/p is observed & the results are neatly tabulated, conclusions are drawn.

Applications:

1.
2.
3.
4.

Remarks:

1.
2.
3.
4.

Probable viva questions:

1.
2.
3.
4.

References:

1.
2.
3.
4.

Signature of staff incharge with date:

MICROCONTROLLER LABORATORY (ECL48)

PROBABLE/SUGGESTED QUESTION BANK FOR LAB EXAM

[1]. Data Transfer Programs – 8051 : Write an assembly language program to transfer n = 10 bytes of data from location 8035h to location 8050h (without overlap).

[2]. Data Transfer Programs – 8051 : Write an assembly language program to exchange n = 5 bytes of data at Location 0027h and at location 0041h.

[3]. Arithmetic operation : Write an ALP to perform the following: If x = 0-perform w + v;
Else if x = 1-perform w - v;
Else if x = 2-perform w * v;
Else if x = 3-perform w / v, where w & v are eight bit numbers.

[4]. Arithmetic operation : Write an assembly language program to sort an array of n= 6 bytes of data in Descending order stored from location 9000h. (Use bubble sort algorithm).

[5]. Assembly Program Illustrating Logical Instructions (Byte Level) : 3 eight bit numbers X, NUM1 & NUM2 are stored in internal data RAM locations 20h, 21h & 22H respectively. Write an ALP to compute the following:
IF X=0; THEN NUM1 (AND) NUM2,
IF X=1; THEN NUM1 (OR) NUM2,
IF X=2; THEN NUM1 (XOR) NUM2,
ELSE RES =00,
STORE RES AT 23H LOCATION

[6]. Assembly Program Illustrating Logical Instructions (Byte Level) : 3 eight bit numbers X, NUM1 & NUM2 are stored in internal data RAM Locations 20h, 21h & 22H respectively.
Write an ALP to compute the following:
IF X=0; THEN LSB OF NUM1 (AND) LSB OF NUM2,
IF X=1; THEN MSB OF NUM1 (OR) MSB OF NUM2,
IF X=2; THEN COMPLEMENT MSB OF NUM1
STORE THE BIT RESULT IN RES,
WHERE RES IS MSB OF 23H LOCATIONS

[7]. Counters program : Write an ALP to implement (display) an eight bit up/down BCD counters on watch window.

[8]. Counters program : Write an ALP to implement (display) an eight bit up/down BCD counters by using timer delay.

[9]. Conversion program : Write an ALP to implement decimal to hex conversion.

[10]. Conversion program : Write an ALP to implement hex to decimal conversion.

[11]. Serial data transmission with variable baud rate – 8051 : Write a program illustrating serial ASCII data transmission (data-BHARAT). Conduct an experiment to configure 8051 microcontroller to transmit characters (BHARAT) to a PC using the serial port and display on the serial window.

[12]. Perform an hardware experiment by implementing a DAC 0808 interface to 8051 to generate square waveforms.

[13]. Perform an hardware experiment by implementing a DAC 0808 interface to 8051 to generate triangular waveforms.

[14]. Perform an hardware experiment by implementing a DAC 0808 interface to 8051 to generate ramp waveforms.

[15]. Perform an hardware experiment to interface a stepper motor to the 8051 microcontroller.

[16]. Perform an hardware experiment to interface a DC motor to the 8051 microcontroller.

[17]. Perform an hardware experiment to interface an alphanumeric LCD panel to the 8051 microcontroller.

[18]. Perform an hardware experiment to interface an ELEVATOR to the 8051 microcontroller.

Arithmetic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ADD A,Rr	A+Rr → A	1	1	C OV AC
ADD A,add	A+(add) → A	2	1	C OV AC
ADD A,@Rp	A+(Rp) → A	1	1	C OV AC
ADD A,#n	A+n → A	2	1	C OV AC
ADDC A,Rr	A+Rr+C → A	1	1	C OV AC
ADDC A,add	A+(add)+C → A	2	1	C OV AC
ADDC A,@Rp	A+(Rp)+C → A	1	1	C OV AC
ADDC A,#n	A+n+C → A	2	1	C OV AC
DA A	Abin → Adec	1	1	C
DEC A	A−1 → A	1	1	
DEC Rr	Rr−1 → Rr	1	1	
DEC add	(add)−1 → (add)	2	1	
DEC @Rp	(Rp)−1 → (Rp)	1	1	
DIV AB	A/B → AB	1	4	0 0V
INC A	A+1 → A	1	1	
INC Rr	Rr+1 → Rr	1	1	
INC add	(add)+1 → (add)	2	1	
INC @Rp	(Rp)+1 → (Rp)	1	1	
INC DPTR	DPTR+1 → DPTR	1	2	
MUL AB	A×B → AB	1	4	0 0V
SUBB A,Rr	A−Rr−C → A	1	1	C OV AC
SUBB A,add	A−(add)−C → A	2	1	C OV AC
SUBB A,@Rp	A−(Rp)−C → A	1	1	C OV AC
SUBB A,#n	A−n−C → A	2	1	C OV AC

Logic

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ANL A,Rr	A AND Rr → A	1	1	
ANL A,add	A AND (add) → A	2	1	
ANL A,@Rp	A AND (Rp) → A	1	1	
ANL A,#n	A AND n → A	2	1	
ANL add,A	(add) AND A → (add)	2	1	
ANL add,#n	(add) AND n → (add)	3	2	
ORL A,Rr	A OR Rr → A	1	1	
ORL A,add	A OR (add) → A	2	1	
ORL A,@Rp	A OR (Rp) → A	1	1	
ORL A,#n	A OR n → A	2	1	
ORL add,A	(add) OR A → (add)	2	1	
ORL add,#n	(add) OR n → (add)	3	2	
XRL A,Rr	A XOR Rr → A	1	1	
XRL A,add	A XOR (add) → A	2	1	
XRL A,@Rp	A XOR (Rp) → A	1	1	
XRL A,#n	A XOR n → A	2	1	
XRL add,A	(add) XOR A → (add)	2	1	
XRL add,#n	(add) XOR n → (add)	3	2	
CLR A	00 → A	1	1	
CPL A	$\overline{A}$ → A	1	1	
NOP	PC+1 → PC	1	1	
RL A	A0←A7←A6..←A1←A0	1	1	
RLC A	C←A7←A6..←A0←C	1	1	C
RR A	A0→A →A6..→A1→A0	1	1	
RRC A	C→A7→A6..→A0→C	1	1	C
SWAP A	Alsn ↔ Amsn	1	1	

Data Moves

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
MOV A,Rr	Rr → A	1	1	
MOV A,add	(add) → A	2	1	
MOV A,@Rp	(Rp) → A	1	1	
MOV A,#n	n → A	2	1	
MOV Rr,A	A → Rr	1	1	
MOV Rr,add	(add) → Rr	2	2	
MOV Rr,#n	n → Rr	2	1	
MOV add,A	A → (add)	2	1	
MOV add,Rr	Rr → (add)	2	2	
MOV add1,add2	(add2) → (add1)	3	2	
MOV add,@Rp	(Rp) → (add)	2	2	
MOV add,#n	n → (add)	3	2	
MOV @Rp,A	A → (Rp)	1	1	
MOV @Rp,add	(add) → (Rp)	2	2	
MOV @Rp,#n	n → (Rp)	2	1	
MOV DPTR,#nn	nn → DPTR	3	2	
MOVC A,@A+DPTR	(A+DPTR) → A	1	2	
MOVC A,@A+PC	(A+PC) → A	1	2	
MOVX A,@DPTR	(DPTR)^ → A	1	2	
MOVX A,@Rp	(Rp)^ → A	1	2	
MOVX @Rp,A	A → (Rp)^	1	2	
MOVX @DPTR,A	A → (DPTR)^	1	2	
POP add	(SP) → (add)	2	2	
PUSH add	(add) → (SP)	2	2	
XCH A,Rr	A ↔ Rr	1	1	
XCH A,add	A ↔ (add)	2	1	
XCH A,@Rp	A ↔ (Rp)	1	1	
XCHD A,@Rp	Alsn ↔ (Rp)lsn	1	1	

Calls and Jumps

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ACALL sadd	PC+2 → (SP); sadd → PC	2	2	
CJNE A,add,radd	[A<>(add)]: PC+3+radd → PC	3	2	C
CJNE A,#n,radd	[A<>n]: PC+3+radd → PC	3	2	C
CJNE Rr,#n,radd	[Rr<>n]: PC+3+radd → PC	3	2	C
CJNE @Rp,#n,radd	[(Rp)<>n]: PC+3+radd → PC	3	2	C
DJNZ Rr,radd	[Rr−1<>00]: PC+2+radd → PC	2	2	
DJNZ add,radd	[(add)−1<>00]: PC+3+radd → PC	3	2	
LCALL ladd	PC+3 → (SP); ladd → PC	3	2	
AJMP sadd	sadd → PC	2	2	
LJMP ladd	ladd → PC	3	2	
SJMP radd	PC+2+radd → PC	2	2	
JMP @A+DPTR	DPTR+A → PC	1	2	
JC radd	[C=1]: PC+2+radd → PC	2	2	
JNC radd	[C=0]: PC+2+radd → PC	2	2	
JB b,radd	[b=1]: PC+3+radd → PC	3	2	
JNB b,radd	[b=0]: PC+3+radd → PC	3	2	
JBC b,radd	[b=1]: PC+3+radd → PC; 0 → b	3	2	
JZ radd	[A=00]: PC+2+radd → PC	2	2	
JNZ radd	[A>00]: PC+2+radd → PC	2	2	
RET	(SP) → PC	1	2	
RETI	(SP) → PC; EI	1	2	

Boolean

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ANL C,b	C AND b → C	2	2	C
ANL C,$\overline{b}$	C AND $\overline{b}$ → C	2	2	C
CLR C	0 → C	1	1	0
CLR b	0 → b	2	1	
CPL C	$\overline{C}$ → C	1	1	C
CPL b	$\overline{b}$ → b	2	1	
ORL C,b	C OR b → C	2	2	C
ORL C,$\overline{b}$	C OR $\overline{b}$ → C	2	2	C
MOV C,b	b → C	2	1	C
MOV b,C	C → b	2	2	
SETB C	1 → C	1	1	1
SETB b	1 → b	2	1	

MNEMONIC ACRONYMS

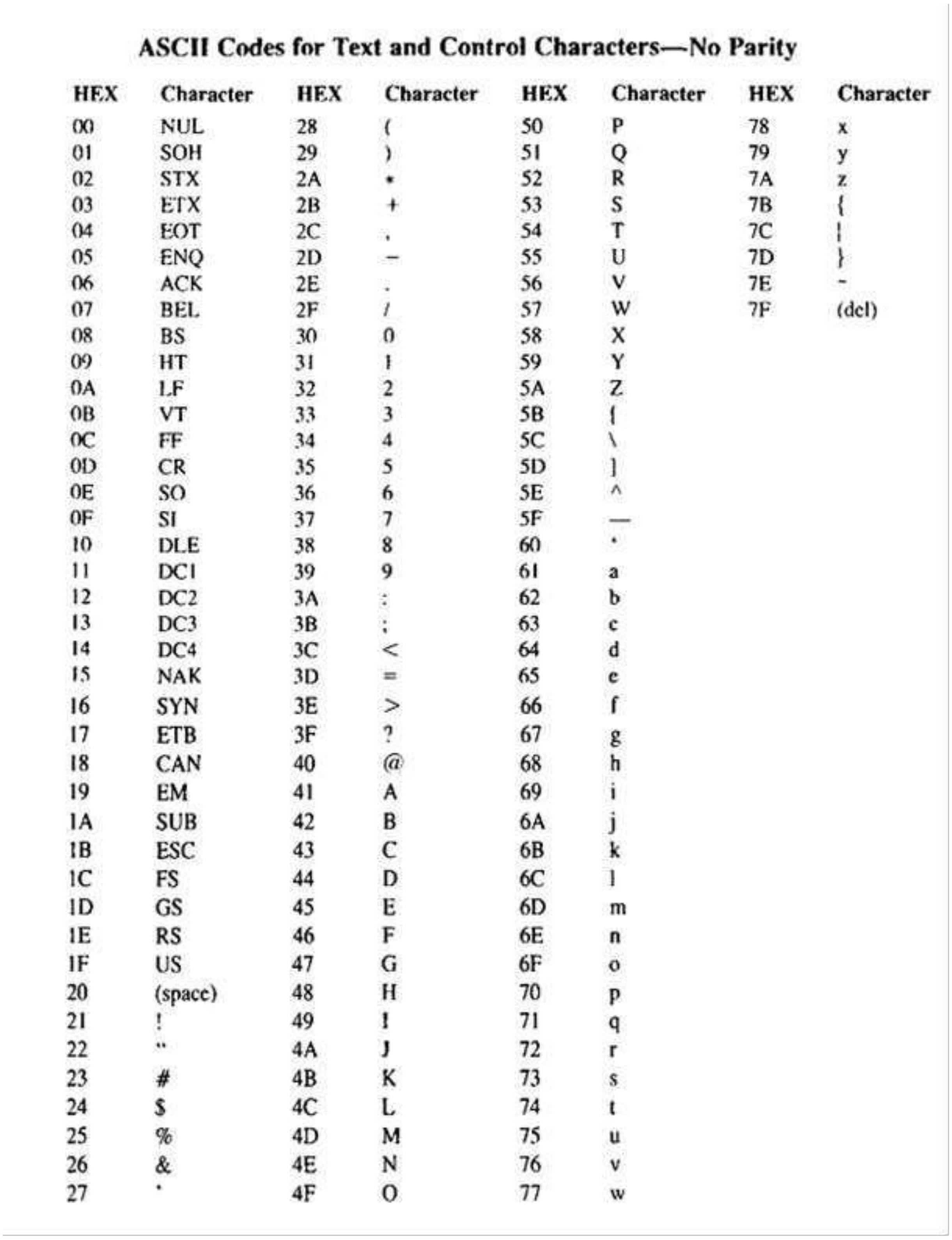
add	Address of the internal RAM from 00h to FFh.
ladd	Long address of 16 bits from 0000h to FFFFh.
radd	Relative address, a signed number from −128d to +127d.
sadd	Short address of 11 bits; complete address = PC11 − PC15 and sadd.
b	Addressable bit in internal RAM or a SFR.
C	The carry flag.
lsn	Least significant nibble.
msn	Most significant nibble.
n	Any immediate 8 bit number from 00h to FFh.
Rr	Any of the eight registers, R0 to R7 in the selected bank.
Rp	Either of the pointing registers R0 or R1 in the selected bank.
[ ]:	IF the condition inside the brackets is *true*, THEN the action listed will occur; ELSE go to the next instruction.
^	External memory location.
( )	Contents of the location inside the parentheses.

Note that flags affected by each instruction are shown where appropriate; any operations that affect the PSW address may also affect the flags.

Intel Corporation Mnemonics, Arranged Alphabetically

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
ACALL addr11	PC+2 → (SP); addr11 → PC	2	2	
ADD A,direct	A+(direct) → A	2	1	C OV AC
ADD A,@Ri	A+(Ri) → A	1	1	C OV AC
ADD A,#data	A+#data → A	2	1	C OV AC
ADD A,Rn	A+Rn → A	1	1	C OV AC
ADDC A,direct	A+(direct)+C → A	2	1	C OV AC
ADDC A,@Ri	A+(Ri)+C → A	1	1	C OV AC
ADDC A,#data	A+#data+C → A	2	1	C OV AC
ADDC A,Rn	A+Rn+C → A	1	1	C OV AC
AJMP addr11	addr11 → PC	2	2	
ANL A,direct	A AND (direct) → A	2	1	
ANL A,@Ri	A AND (Ri) → A	1	1	
ANL A,#data	A AND #data → A	2	1	
ANL A,Rn	A AND Rn → A	1	1	
ANL direct,A	(direct) AND A → (direct)	2	1	
ANL direct,#data	(direct) AND #data → (direct)	3	2	
ANL C,bit	C AND bit → C	2	2	C
ANL C,$\overline{\text{bit}}$	C AND $\overline{\text{bit}}$ → C	2	2	C
CJNE A,direct,rel	[A<>(direct)]: PC+3+rel → PC	3	2	C
CJNE A,#data,rel	[A<>n]: PC+3+rel → PC	3	2	C
CJNE @Ri,#data,rel	[(Ri)<>n]: PC+3+rel → PC	3	2	C
CJNE Rn,#data,rel	[Rn<>n]: PC+3+rel → PC	3	2	C
CLR A	0 → A	1	1	
CLR bit	0 → bit	2	1	
CLR C	0 → C	1	1	0
CPL A	$\overline{\text{A}}$ → A	1	1	
CPL bit	$\overline{\text{bit}}$ → bit	2	1	
CPL C	$\overline{\text{C}}$ → C	1	1	C
DA A	Abin → Adec	1	1	C
DEC A	A−1 → A	1	1	
DEC direct	(direct)−1 → (direct)	2	1	
DEC @Ri	(Ri)−1 → (Ri)	1	1	
DEC Rn	Rn−1 → Rn	1	1	
DIV AB	A/B → AB	1	4	0 OV
DJNZ direct,rel	[(direct)−1<>00]: PC+3+rel → PC	3	2	
DJNZ Rn,rel	[Rn−1<>00]: PC+2+rel → PC	2	2	
INC A	A+1 → A	1	1	
INC direct	(direct)+1 → (direct)	2	1	
INC DPTR	DPTR+1 → DPTR	1	2	
INC @Ri	(Ri)+1 → (Ri)	1	1	
INC Rn	Rn+1 → Rn	1	1	
JB bit,rel	[b=1]: PC+3+rel → PC	3	2	
JBC bit,rel	[b=1]: PC+3+rel → PC; 0 → bit	3	2	
JC rel	[C=1]: PC+2+rel → PC	2	2	
JMP @A+DPTR	DPTR+A → PC	1	2	
JNB bit,rel	[b=0]: PC+3+rel → PC	3	2	
JNC rel	[C=0]: PC+2+rel → PC	2	2	
JNZ rel	[A>00]: PC+2+rel → PC	2	2	
JZ rel	[A=00]: PC+2+rel → PC	2	2	

MNEMONIC	DESCRIPTION	BYTES	CYCLES	FLAGS
LCALL addr16	PC+3 → (SP); addr16 → PC	3	2	
LJMP addr16	addr16 → PC	3	2	
MOV A,direct	(direct) → A	2	1	
MOV A,@Ri	(Ri) → A	1	1	
MOV A,#data	#data → A	2	1	
MOV A,Rn	Rn → A	1	1	
MOV direct,A	A → (direct)	2	1	
MOV direct,direct	(direct) → (direct)	3	2	
MOV direct,@Ri	(Ri) → (direct)	2	2	
MOV direct,#data	#data → (direct)	3	2	
MOV direct,Rn	Rn → (direct)	2	2	
MOV bit,C	C → bit	2	2	
MOV C,bit	bit → C	2	1	C
MOV @Ri,A	A → (Ri)	1	1	
MOV @Ri,direct	(direct) → (Ri)	2	2	
MOV @Ri,#data	#data → (Ri)	2	1	
MOV DPTR,#data16	#data16 → DPTR	3	2	
MOV Rn,A	A → Rn	1	1	
MOV Rn,direct	(direct) → Rn	2	2	
MOV Rn,#data	#data → Rn	2	1	
MOVC A,@A+DPTR	(A+DPTR) → A	1	2	
MOVC A,@A+PC	(A+PC) → A	1	2	
MOVX A,@DPTR	(DPTR)^ → A	1	2	
MOVX A,@Ri	(Ri)^ → A	1	2	
MOVX @DPTR,A	A → (DPTR)^	1	2	
MOVX @Ri,A	A → (Ri)^	1	2	
NOP	PC+1 → PC	1	1	
MUL AB	A×B → AB	1	4	0 OV
ORL A,direct	A OR (direct) → A	2	1	
ORL A,@Ri	A OR (Ri) → A	1	1	
ORL A,#data	A OR #data → A	2	1	
ORL A,Rn	A OR Rn → A	1	1	
ORL direct,A	(direct) OR A → (direct)	2	1	
ORL direct,#data	(direct) OR #data → (direct)	3	2	
ORL C,bit	C OR bit → C	2	2	C
ORL C,$\overline{\text{bit}}$	C OR $\overline{\text{bit}}$ → C	2	2	C
POP direct	(SP) → (direct)	2	2	
PUSH direct	(direct) → (SP)	2	2	
RET	(SP) → PC	1	2	
RETI	(SP) → PC; EI	1	2	
RL A	A0←A7←A6..←A1←A0	1	1	
RLC A	C←A7←A6..←A0←C	1	1	C
RR A	A0→A7→A6..→A1→A0	1	1	
RRC A	C→A7→A6..→A0→C	1	1	C
SETB bit	1 → bit	2	1	
SETB C	1 → C	1	1	1
SJMP rel	PC+2+rel → PC	2	2	
SUBB A,direct	A−(direct)−C → A	2	1	C OV AC
SUBB A,@Ri	A−(Ri)−C → A	1	1	C OV AC
SUBB A,#data	A−#data−C → A	2	1	C OV AC
SUBB A,Rn	A−Rn−C → A	1	1	C OV AC



ASCII Codes for Text and Control Characters—No Parity

HEX	Character	HEX	Character	HEX	Character	HEX	Character
00	NUL	28	(	50	P	78	x
01	SOH	29	)	51	Q	79	y
02	STX	2A	*	52	R	7A	z
03	ETX	2B	+	53	S	7B	{
04	EOT	2C	,	54	T	7C	¦
05	ENQ	2D	–	55	U	7D	}
06	ACK	2E	.	56	V	7E	~
07	BEL	2F	/	57	W	7F	(del)
08	BS	30	0	58	X		
09	HT	31	1	59	Y		
0A	LF	32	2	5A	Z		
0B	VT	33	3	5B	[		
0C	FF	34	4	5C	\		
0D	CR	35	5	5D	]		
0E	SO	36	6	5E	^		
0F	SI	37	7	5F	—		
10	DLE	38	8	60	`		
11	DC1	39	9	61	a		
12	DC2	3A	:	62	b		
13	DC3	3B	;	63	c		
14	DC4	3C	<	64	d		
15	NAK	3D	=	65	e		
16	SYN	3E	>	66	f		
17	ETB	3F	?	67	g		
18	CAN	40	@	68	h		
19	EM	41	A	69	i		
1A	SUB	42	B	6A	j		
1B	ESC	43	C	6B	k		
1C	FS	44	D	6C	l		
1D	GS	45	E	6D	m		
1E	RS	46	F	6E	n		
1F	US	47	G	6F	o		
20	(space)	48	H	70	p		
21	!	49	I	71	q		
22	"	4A	J	72	r		
23	#	4B	K	73	s		
24	$	4C	L	74	t		
25	%	4D	M	75	u		
26	&	4E	N	76	v		
27	'	4F	O	77	w		

NOTES

1. ______
2. ______
3. ______
4. ______
5. ______
6. ______
7. ______
8. ______
9. ______
10. ______
11. ______
12. ______
13. ______
14. ______
15. ______
16. ______
17. ______
18. ______
19. ______
20. ______
21. ______
22. ______
23. ______
24. ______
25. ______
26. ______
27. ______
28. ______

Vision of the Institute

- To impart quality technical education with a focus on Research and Innovation emphasizing on Development of Sustainable and Inclusive Technology for the benefit of society.

Mission of the Institute

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence.
- To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.
- To train the students to the changing technical scenario and make them to understand the importance of sustainable and inclusive technologies.

Vision of the ECE Department

- To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional success.

Mission of the ECE Department

- Offering quality education in Electronics and Communication Engineering with effective teaching learning process in multidisciplinary environment.
- Training the students to take-up projects in emerging technologies and work with team spirit.
- To imbibe professional ethics, development of skills and research culture for better placement opportunities.

DEPARTMENT
OF
ELECTRONICS & COMMUNICATION ENGINEERING