ATME College of Engineering

Department of Electrical and Electronics Engineering

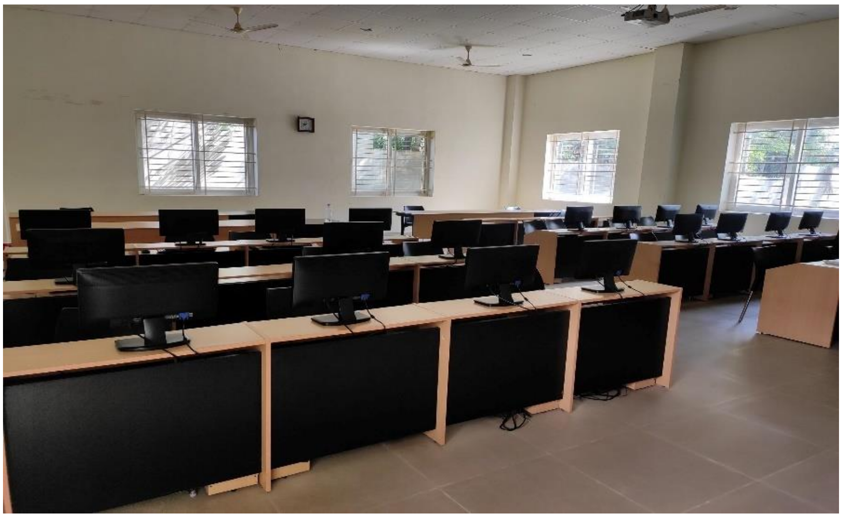
Laboratory Manual

Microcontrollers

BEE403

Academic Year: 2024-25

Semester: IV



Compiled by | Verified by | Approved by

ATME College of Engineering
13th km Stone, Mysuru-Kanakapura-Bengaluru Road, Mysuru-570028

INSTITUTIONAL VISION AND MISSION

VISION:

Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

MISSION:

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torchbearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence

DEPARTMENT VISION AND MISSION

VISION:

To create Electrical and Electronics Engineers who excel to be technically competent and fulfill the cultural and social aspirations of the society.

MISSION:

- To provide knowledge to students that builds a strong foundation in the basic principles of electrical engineering, problem solving abilities, analytical skills, soft skills and communication skills for their overall development.
- To offer outcome based technical education.
- To encourage faculty in training & development and to offer consultancy through research & industry interaction.

PROGRAMME OUTCOMES:

Engineering Graduates will be able to:

PO1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of EXPERIMENTs, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

At the end of graduation, the student will be able,

PSO1: Apply the concepts of Electrical & Electronics Engineering to evaluate the performance of power systems and also to control Industrial drives using power electronics.

PSO2: Demonstrate the concepts of process control for Industrial Automation, design models for environmental and social concerns and also exhibit continuous self- learning.

Program Educational Objectives (PEOs)

PEO1: To produce competent and Ethical Electrical and Electronics Engineers who will exhibit the necessary technical and managerial skills to perform their duties in society.

PEO2: To make Graduates continuously acquire and enhance their technical and socio-economic skills.

PEO3: To aspire Graduates on R&D activities leading to offering solutions and excel in various career paths.

PEO4: To produce quality engineers who have the capability to work in teams and contribute to real time projects.

LIST OF EXPERIMENTS

CYCLE-I

Sl. No	Experiment Name	COs	BTL
1	Data Transfer – Block move, Exchange, Sorting, Finding largest element in an array.	CO1/CO2	L_5
2	Arithmetic Instructions – Addition/subtraction, multiplication and division, square, Cube – (16 bits Arithmetic operations – bit addressable).	CO1/CO2	L_5
3	Up/Down BCD/ Binary Counters	CO3	L_5
4	Boolean & Logical Instructions (Bit manipulations).	CO2/CO3	L_5
5	Code conversion: BCD – ASCII; ASCII – Decimal; Decimal - ASCII; HEX - Decimal and Decimal –HEX.	CO3	L_5
6	Programs to generate delay, Programs using serial port and on-Chip timer / counter.	CO3/CO4	L_5

CYCLE-II

Note: Single chip solution for interfacing 8051 is to be with C Programs for the following experiments.

Sl. No	Experiment Name	COs	BTL
7	Stepper motor interface for direction and speed control	CO5	L_5
8	Simulate and Test a PWM controlled DC Motor	CO5	L_5
9	Alphanumerical LCD panel interface.	CO5	L_5
10	Generate different waveforms: Sine, Square, Triangular, Ramp using DAC interface	CO5	L_5

REFERENCE BOOK:
1. "The 8051 Microcontroller and Embedded Systems – using assembly and C"- Muhammad Ali Mazidi and Janice Gillespie -,PHI,2006/pearson,2006
2. "The 8051 Microcontroller", V.Udayashankar and Mallikarjuna Swamy, TMH,2009
3. "MSP430 Microcontroller Basics", John Davies, Elsevier, 2008
4. http://www.magzter.com/IN/EFY-Enterprises-Pvt-Ltd/Micro-Controller-Based-Projects-2nd-Edition/Technology/22026

COURSE OUTCOMES

At the end of the course the student will be able to:

CO-1: Outline the 8051 architecture, registers, internal memory organization, addressing modes

CO-2: Discuss 8051 addressing modes, instruction set of 8051, accessing data and I/O port programming.

CO-3: Develop 8051C programs for time delay, I/O operations, I/O bit manipulation, logic and arithmetic operations, data conversion and timer/counter programming.

CO-4: Summarize the basics of serial communication and interrupts, also develop 8051 programs for serial data communication and interrupt programming

CO-5: Program 8051to work with external devices for ADC, DAC, Stepper motor control, DC motor control.

The Correlation of Course Outcomes (CO's) and PO's and PSOs

Course Code:	**BEE403**	**Title: Microcontrollers**											
Course Outcomes	**Program Outcomes**											**PSOs**	
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO-1	2	2	-	-	3	-	-	-	2	2	-	2	-	3
CO-2	2	3	2	2	3	-	-	-	3	2	-	2	-	3
CO-3	2	3	2	2	3	-	-	-	3	2	-	2	-	3
CO-4	2	3	2	2	3	-	-	-	3	2	-	2	-	3
CO-5	2	3	2	2	3	-	-	-	3	2	-	2	-	3

Note: 3 = Strong Contribution 2 = Average Contribution 1 = Weak Contribution "-" = No Contribution

TABLE OF CONTENTS

VTU Syllabus

I. PROGRAMMING

1. Data Transfer – Block move, Exchange, Sorting, Finding largest element in an array.
2. Arithmetic Instructions – Addition/subtraction, multiplication and division, square, Cube – (16 bits Arithmetic operations – bit addressable).
3. Up/Down BCD/ Binary Counters.
4. Boolean & Logical Instructions (Bit manipulations).
5. Code conversion: BCD – ASCII; ASCII – BCD; ASCII-Decimal, Decimal - ASCII; HEX - Decimal and Decimal –HEX.
6. Programs to generate delay, Programs using serial port and on-Chip timer / counter.

Note: Single chip solution for interfacing 8051 is to be with C Programs for the following experiments.

II. INTERFACING:

Write C programs to interface 8051 chip to Interfacing modules to develop single chip solutions.

7. Stepper motor interface for direction and speed control
8. Simulate and Test a PWM controlled DC Motor
9. Alphanumerical LCD panel interface.
10. Generate different waveforms: Sine, Square, Triangular, Ramp using DAC interface.

INTRODUCTION

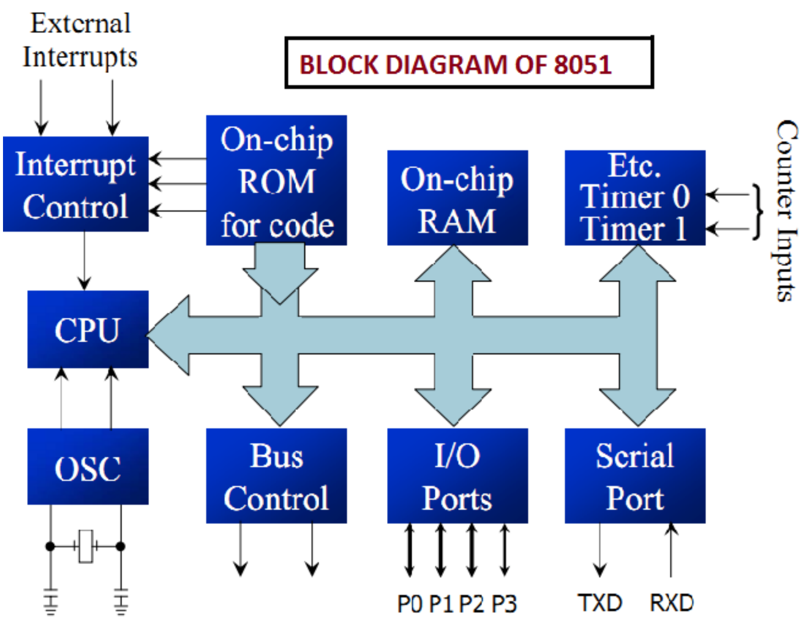


Fig 1: Block diagram of 8051

The 8051 PIN DIAGRAM

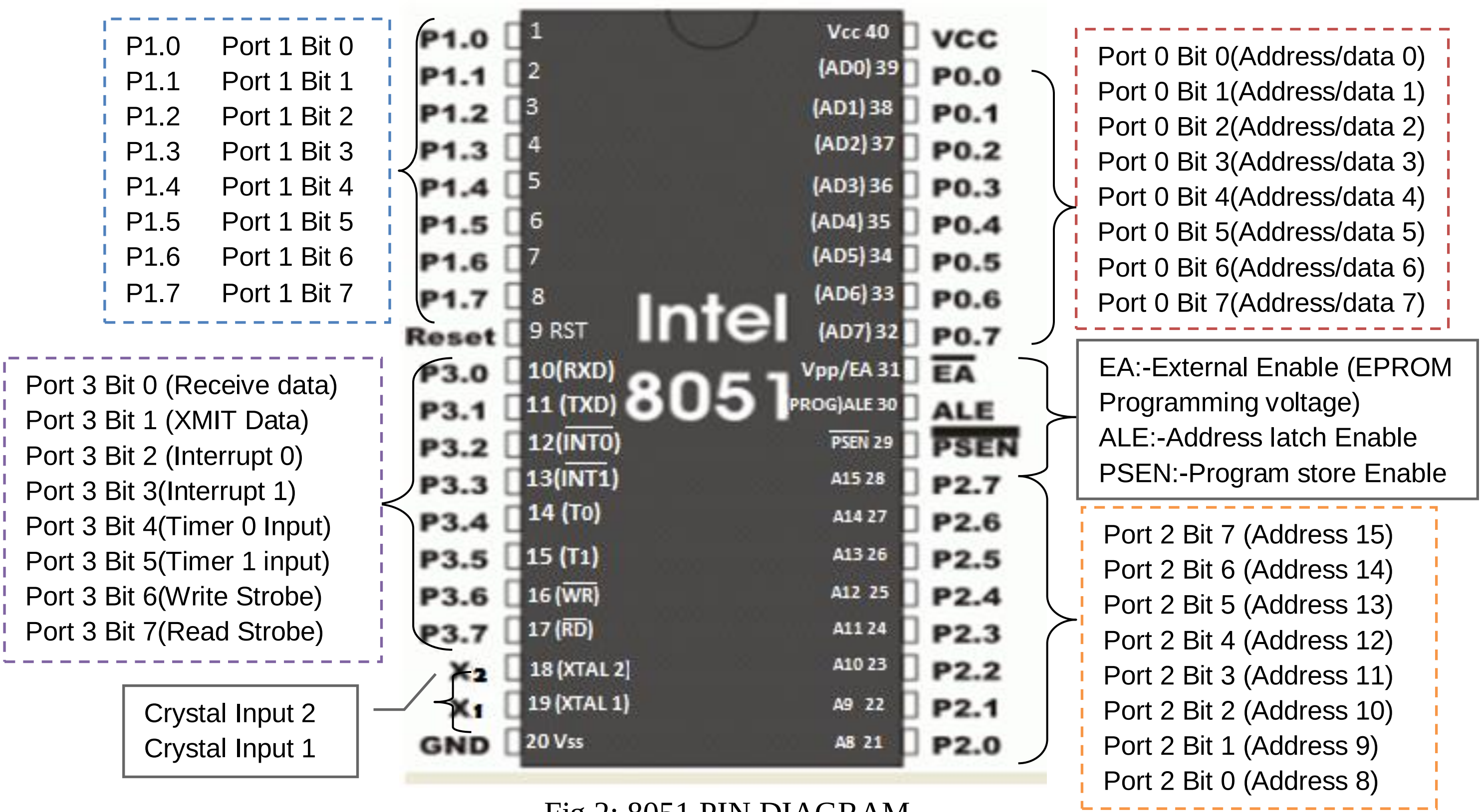


Fig 2: 8051 PIN DIAGRAM

PINOUT DESCRIPTION

Pins 1-8: Port 1 Each of these pins can be configured as an input or an output.

Pin 9: RS A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.

Pins10-17: Port 3 Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions:

Pin 10: RXD Serial asynchronous communication input or Serial synchronous communication output.

Pin 11: TXD Serial asynchronous communication output or Serial synchronous communication clock output.

Pin 12: INT0 Interrupt 0 inputs.

Pin 13: INT1 Interrupt 1 input.

Pin 14: T0 Counter 0 clock input.

Pin 15: T1 Counter 1 clock input.

Pin 16: WR Write to external (additional) RAM.

Pin 17: RD Read from external RAM.

Pin 18, 19:X2 X1 Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramics resonators can also be used for frequency stability. Later versions of microcontrollers operate at a frequency of 0 Hz up to over 50 Hz.

Pin 20: GND Ground.

Pin 21-28: Port 2 If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.

Pin 29: PSEN If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30: ALE Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external register (usually 74HCT373 or 74HCT375 add-on chip) memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.

Pin 31: EA By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

Pin 32-39: Port 0 Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

Pin 40: VCC +5V power supply.

The 8051 Architecture

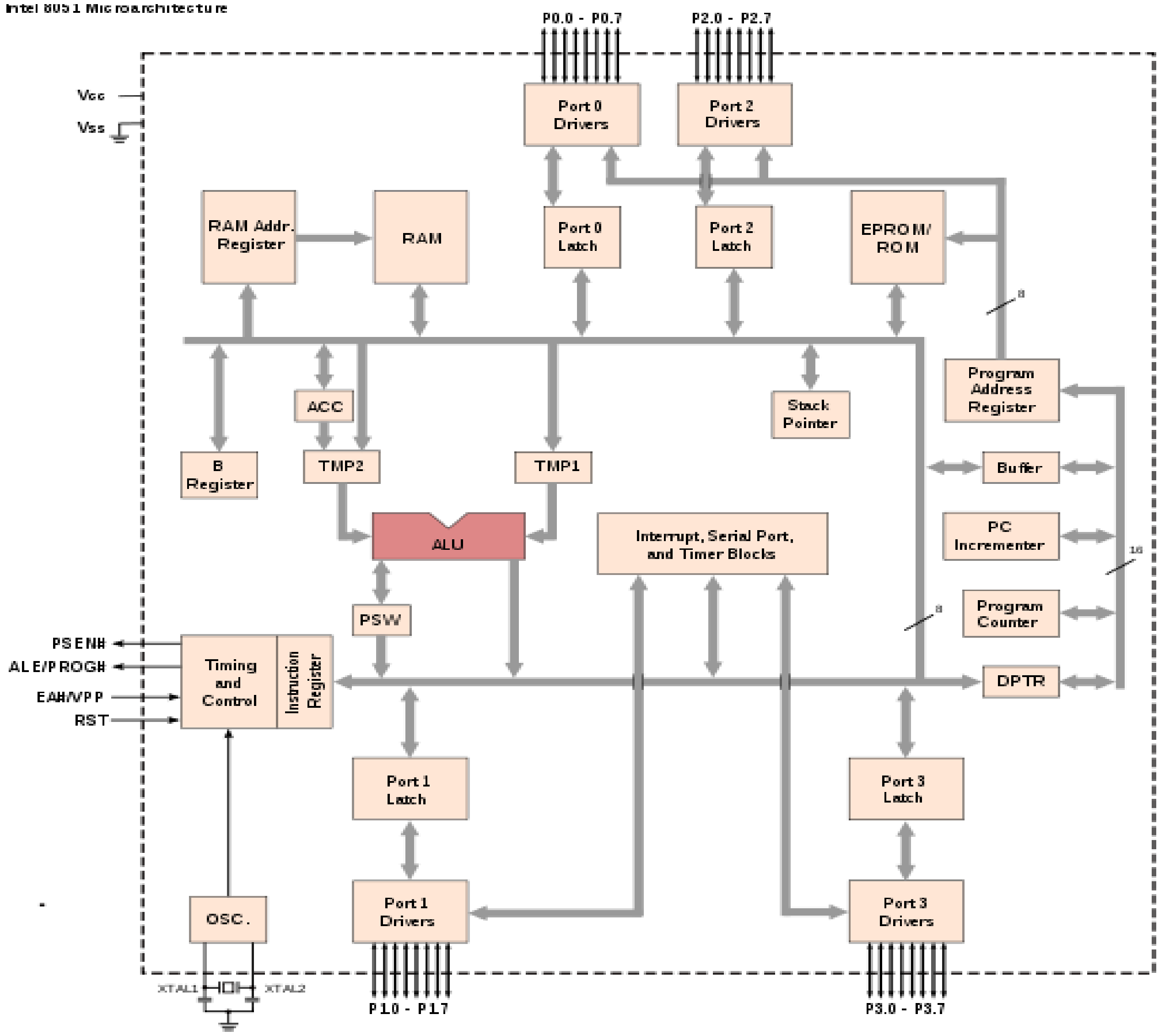


Fig 3 :8051 Architecture

8051 SPECIFIC FEATURES

- The 8051 architecture provides many functions (CPU, RAM, ROM, I/O, interrupt logic, timer, etc.) in a single package
- 8-bit ALU, Accumulator and 8-bit Registers; hence it is an 8-bitmicrocontroller
- 8-bit data bus – It can access 8 bits of data in one operation
- 16-bit address bus – It can access 2^{16} memory locations – 64 KB (65536 locations) each of RAM and ROM
- On-chip RAM – 128 bytes (data memory)
- On-chip ROM – 4 Kbyte (program memory)
- Four byte bi-directional input/output port
- UART (serial port)
- Two 16-bit Counter/timers
- Two-level interrupt priority and Power saving mode (on some derivatives)

The 8051 Programming Model

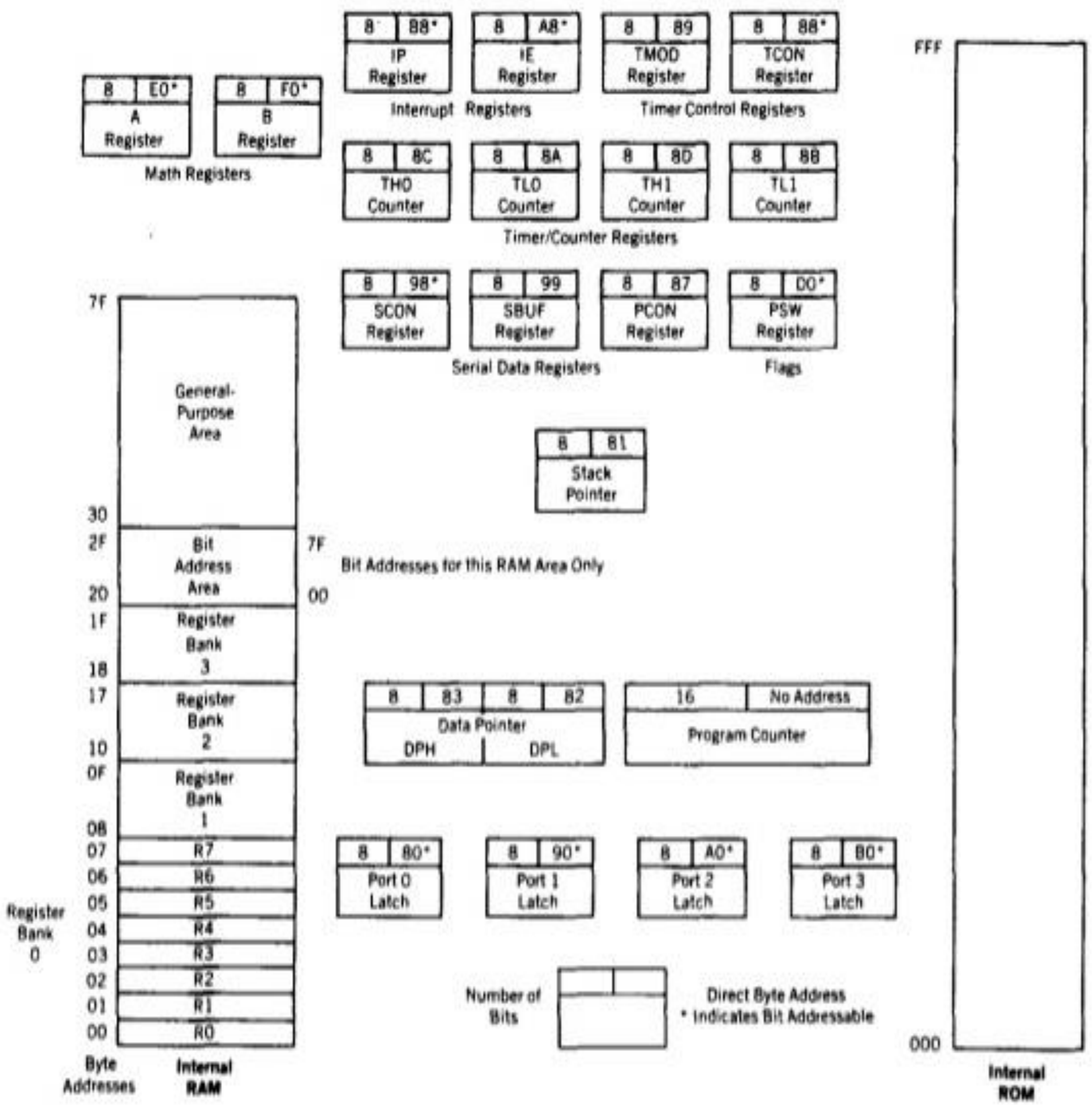


Fig 4:8051 Programming Model

8051 Microcontroller Instruction Set

Table 1.1: Instructions that Affect Flag Settings

Instruction	Flag			Instruction	Flag		
	C	**OV**	**AC**		**C**	**OV**	**AC**
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Note: 1. Operations on SFR byte address 208 or bit addresses 209-215 (that is, the PSW or bits in the PSW) also affect flag settings.

Table 1.2The Instruction Set and Addressing Modes

R_n	Register R7-R0 of the currently selected Register Bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O
@R_i	8-bit internal data RAM location (0-255) addressed indirectly through register R1or R0.
#data	8-bit constant included in instruction.
#data 16	16-bit constant included in instruction.
addr 16	16-bit destination address. Used by LCALL and LJMP. A branch can be anywhere within the 64K byte Program
addr 11	11-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2K byte page ofprogram memory as the first byte of the following instruction.
Rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127
Bit	Direct Addressed bit in Internal Data RAM or Special Function Register.

Table 1.3:Instruction Set Summary

	0	1	2	3	4	5	6	7
0	NOP	JBC bit,rel [3B, 2C]	JB bit, rel [3B, 2C]	JNB bit, rel [3B, 2C]	JC rel [2B, 2C]	JNC rel [2B, 2C]	JZ rel [2B, 2C]	JNZ rel [2B, 2C]
1	AJMP (P0) [2B, 2C]	ACALL (P0) [2B, 2C]	AJMP (P1) [2B, 2C]	ACALL (P1) [2B, 2C]	AJMP (P2) [2B, 2C]	ACALL (P2) [2B, 2C]	AJMP (P3) [2B, 2C]	ACALL (P3) [2B, 2C]
2	LJMP addr16 [3B, 2C]	LCALL addr16 [3B, 2C]	RET [2C]	RETI [2C]	ORL dir, A [2B]	ANL dir, A [2B]	XRL dir, a [2B]	ORL C, bit [2B, 2C]
3	RR A	RRC A	RL A	RLC A	ORL dir, #data [3B, 2C]	ANL dir, #data [3B, 2C]	XRL dir, #data [3B, 2C]	JMP @A + DPTR [2C]
4	INC A	DEC A	ADD A, #data [2B]	ADDC A, #data [2B]	ORL A, #data [2B]	ANL A, #data [2B]	XRL A, #data [2B]	MOV A, #data [2B]
5	INC dir [2B]	DEC dir [2B]	ADD A, dir [2B]	ADDC A, dir [2B]	ORL A, dir [2B]	ANL A, dir [2B]	XRL A, dir [2B]	MOV dir, #data [3B, 2C]
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, @data [2B]
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, #data [2B]
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, #data [2B]
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, #data [2B]
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, #data [2B]
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, #data [2B]
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, #data [2B]
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, #data [2B]
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, #data [2B]
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, #data [2B]

Note: Key: [2B] = 2 Byte, [3B] = 3 Byte, [2C] = 2 Cycle, [4C] = 4 Cycle, Blank = 1 byte/1 cycle

Table 1.3. Instruction Set Summary (Continued)

	8	9	A	B	C	D	E	F
0	SJMP REL [2B, 2C]	MOV DPTR,# data 16 [3B, 2C]	ORL C, /bit [2B, 2C]	ANL C, /bit [2B, 2C]	PUSH dir [2B, 2C]	POP dir [2B, 2C]	MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
1	AJMP (P4) [2B, 2C]	ACALL (P4) [2B, 2C]	AJMP (P5) [2B, 2C]	ACALL (P5) [2B, 2C]	AJMP (P6) [2B, 2C]	ACALL (P6) [2B, 2C]	AJMP (P7) [2B, 2C]	ACALL (P7) [2B, 2C]
2	ANL C, bit [2B, 2C]	MOV bit, C [2B, 2C]	MOV C, bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]	MOVX A, @R0 [2C]	MOVX wR0, A [2C]
3	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @RI [2C]	MOVX @RI, A [2C]
4	DIV AB [2B, 4C]	SUBB A, #data [2B]	MUL AB [4C]	CJNE A, #data, rel [3B, 2C]	SWAP A	DA A	CLR A	CPL A
5	MOV dir, dir [3B, 2C]	SUBB A, dir [2B]		CJNE A, dir, rel [3B, 2C]	XCH A, dir [2B]	DJNZ dir, rel [3B, 2C]	MOV A, dir [2B]	MOV dir, A [2B]
6	MOV dir, @R0 [2B, 2C]	SUBB A, @R0	MOV @R0, dir [2B, 2C]	CJNE @R0, #data, rel [3B, 2C]	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
7	MOV dir, @R1 [2B, 2C]	SUBB A, @R1	MOV @R1, dir [2B, 2C]	CJNE @R1, #data, rel [3B, 2C]	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
8	MOV dir, R0 [2B, 2C]	SUBB A, R0	MOV R0, dir [2B, 2C]	CJNE R0, #data, rel [3B, 2C]	XCH A, R0	DJNZ R0, rel [2B, 2C]	MOV A, R0	MOV R0, A
9	MOV dir, R1 [2B, 2C]	SUBB A, R1	MOV R1, dir [2B, 2C]	CJNE R1, #data, rel [3B, 2C]	XCH A, R1	DJNZ R1, rel [2B, 2C]	MOV A, R1	MOV R1, A
A	MOV dir, R2 [2B, 2C]	SUBB A, R2	MOV R2, dir [2B, 2C]	CJNE R2, #data, rel [3B, 2C]	XCH A, R2	DJNZ R2, rel [2B, 2C]	MOV A, R2	MOV R2, A
B	MOV dir, R3 [2B, 2C]	SUBB A, R3	MOV R3, dir [2B, 2C]	CJNE R3, #data, rel [3B, 2C]	XCH A, R3	DJNZ R3, rel [2B, 2C]	MOV A, R3	MOV R3, A
C	MOV dir, R4 [2B, 2C]	SUBB A, R4	MOV R4, dir [2B, 2C]	CJNE R4, #data, rel [3B, 2C]	XCH A, R4	DJNZ R4, rel [2B, 2C]	MOV A, R4	MOV R4, A
D	MOV dir, R5 [2B, 2C]	SUBB A, R5	MOV R5, dir [2B, 2C]	CJNE R5, #data, rel [3B, 2C]	XCH A, R5	DJNZ R5, rel [2B, 2C]	MOV A, R5	MOV R5, A
E	MOV dir, R6 [2B, 2C]	SUBB A, R6	MOV R6, dir [2B, 2C]	CJNE R6, #data, rel [3B, 2C]	XCH A, R6	DJNZ R6, rel [2B, 2C]	MOV A, R6	MOV R6. A
F	MOV dir, R7 [2B, 2C]	SUBB A, R7	MOV R7, dir [2B, 2C]	CJNE R7, #data, rel [3B, 2C]	XCH A, R7	DJNZ R7, rel [2B, 2C]	MOV A, R7	MOV R7, A

Note: Key: [2B] = 2 Byte, [3B] = 3 Byte, [2C] = 2 Cycle, [4C] = 4 Cycle, Blank = 1 byte/1 cycle

Table 1.4:AT89 Instruction Set Summary

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Right Rotate Accumulator	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct stack	Push direct byte onto	2	24
POP	direct stack	Pop direct byte from	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Acc	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACAL L	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24

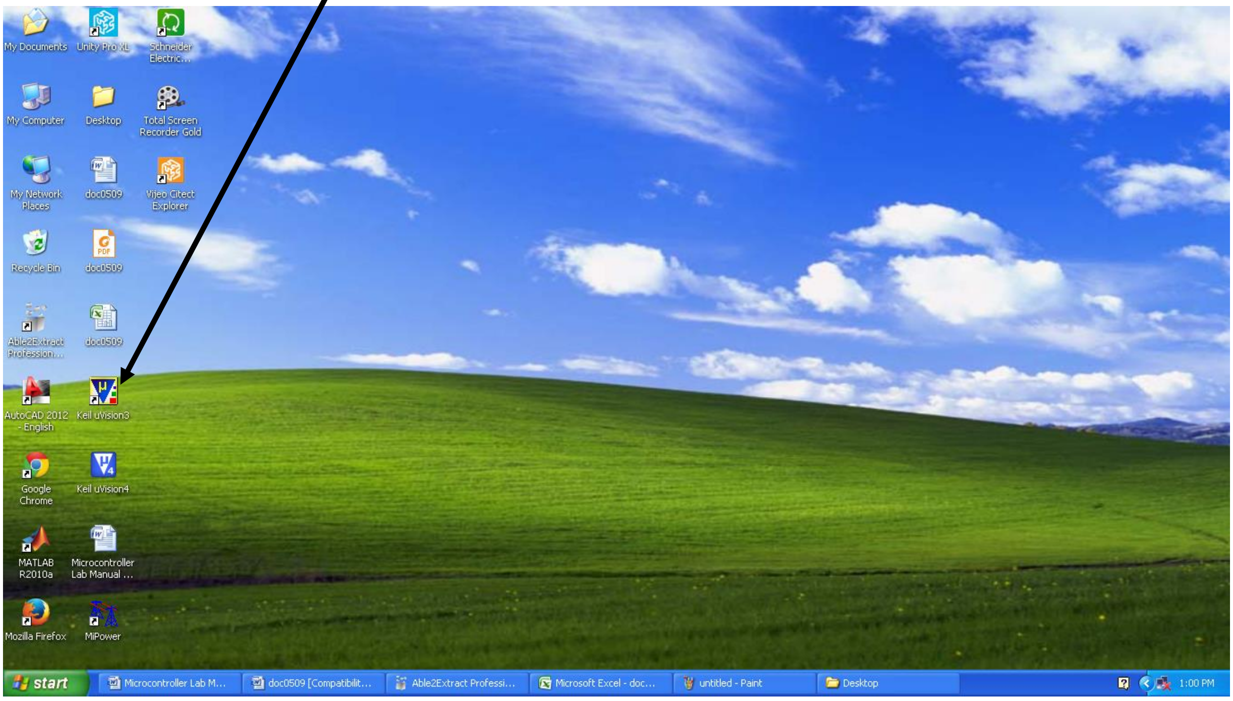
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

SOFTWARE PROGRAMS

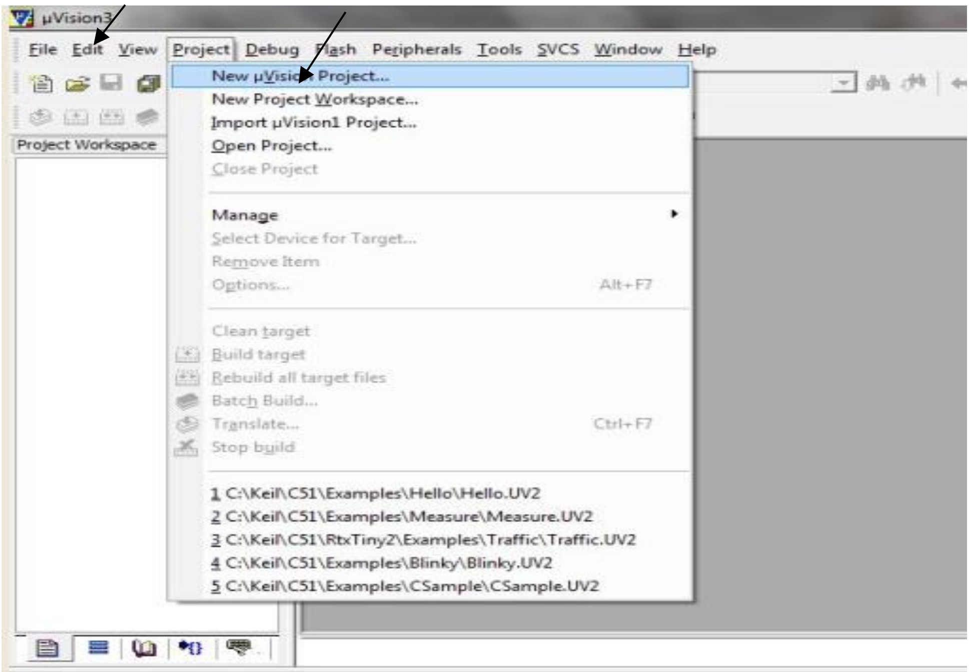
MICRO VISION COMPILER AND SIMULATOR

STEPS FOR EXECUTING THE SOFTWARE PROGRAM:

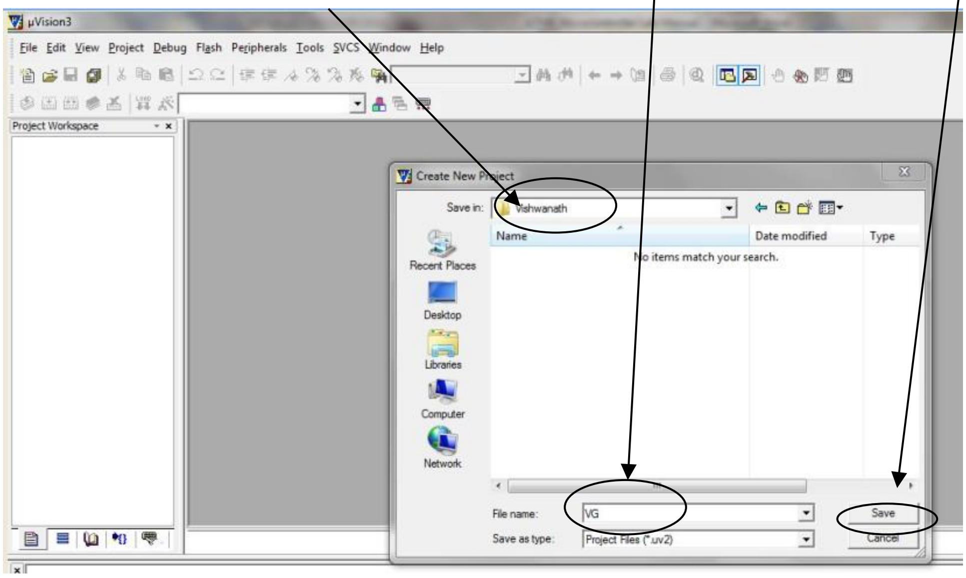
STEP 1: Select the **"Kiel µVision 3"** software.



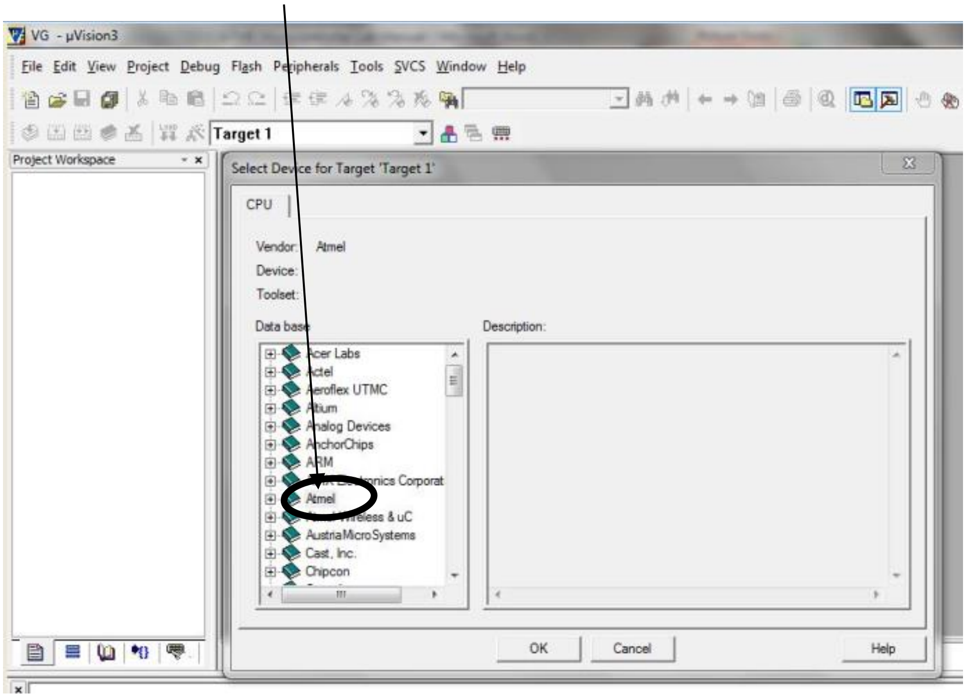
STEP 2: Select **"Project"** then **"New µVision Project".**



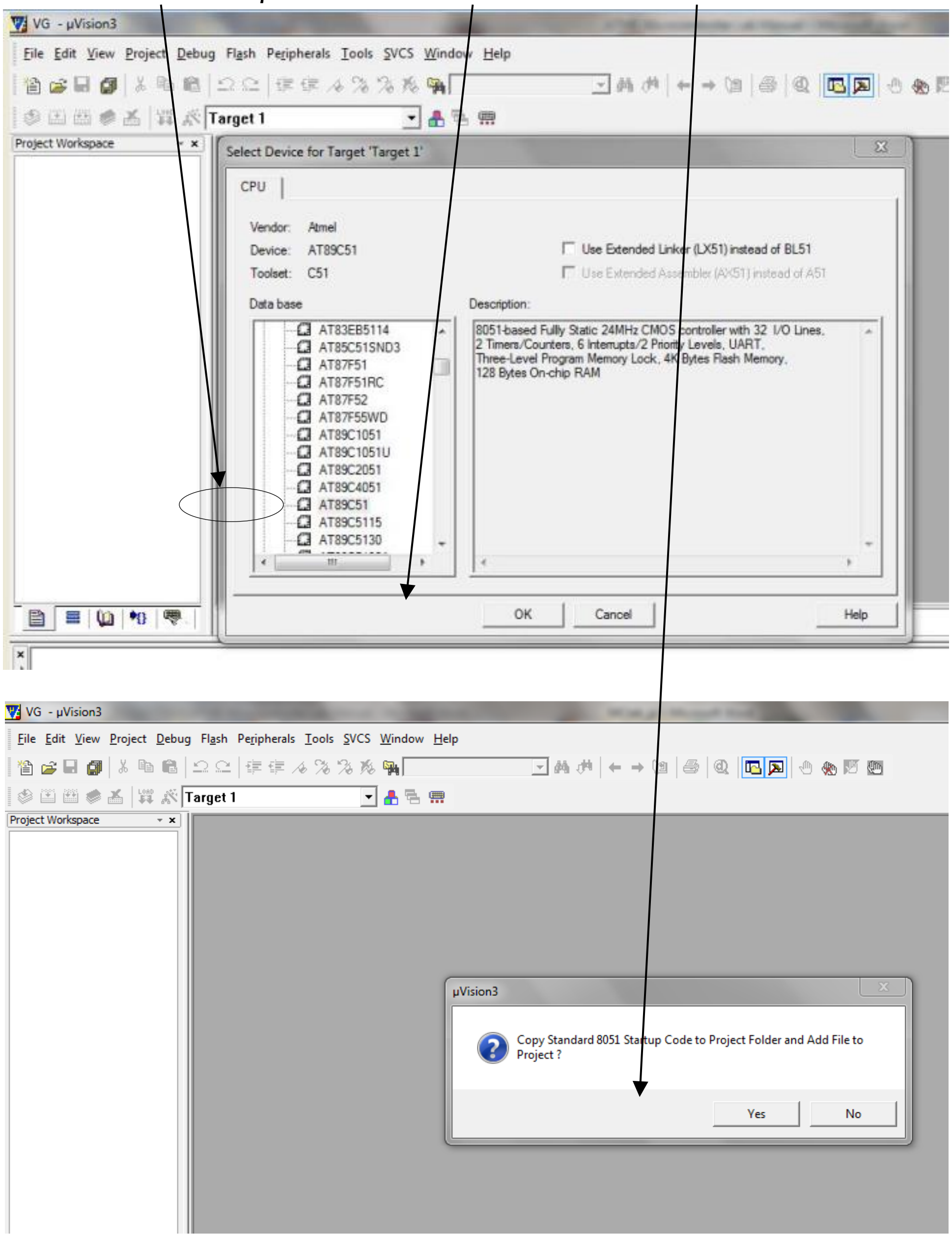
STEP 3: Create new project by entering your **"File name"** and then **"Save"** your file



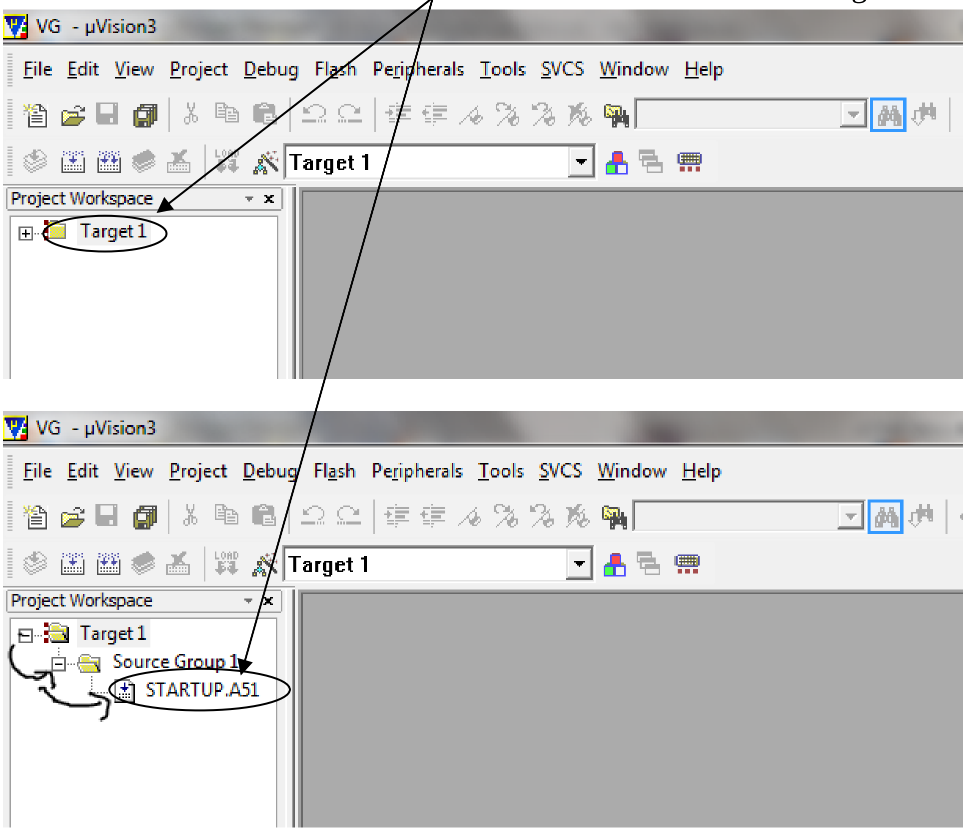
STEP 4: Choose **"Atmel"** microcontroller from the database



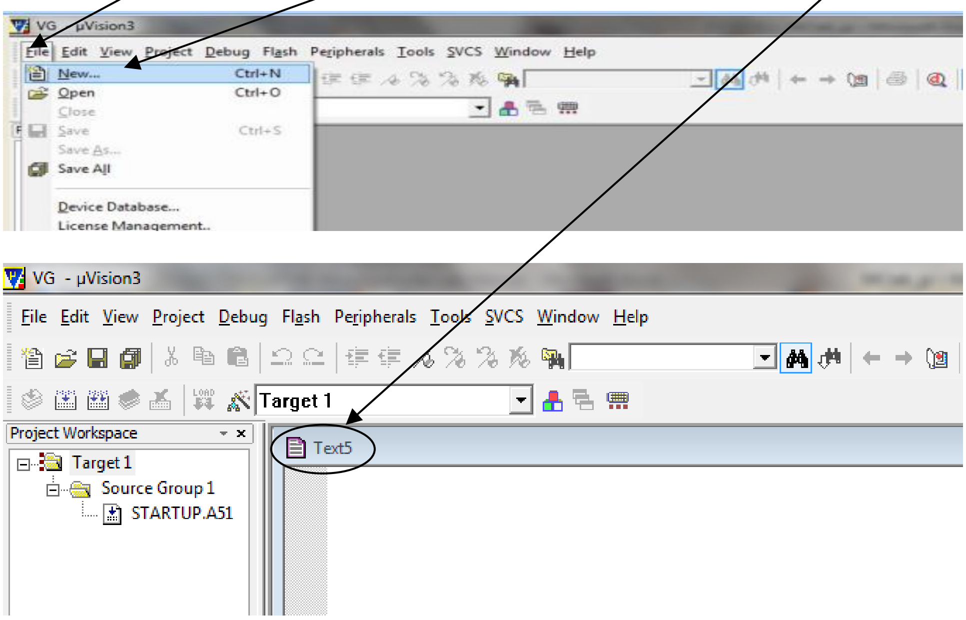
STEP 5: Select **"AT89C51"** µC and click **"OK"** and then **"YES"**



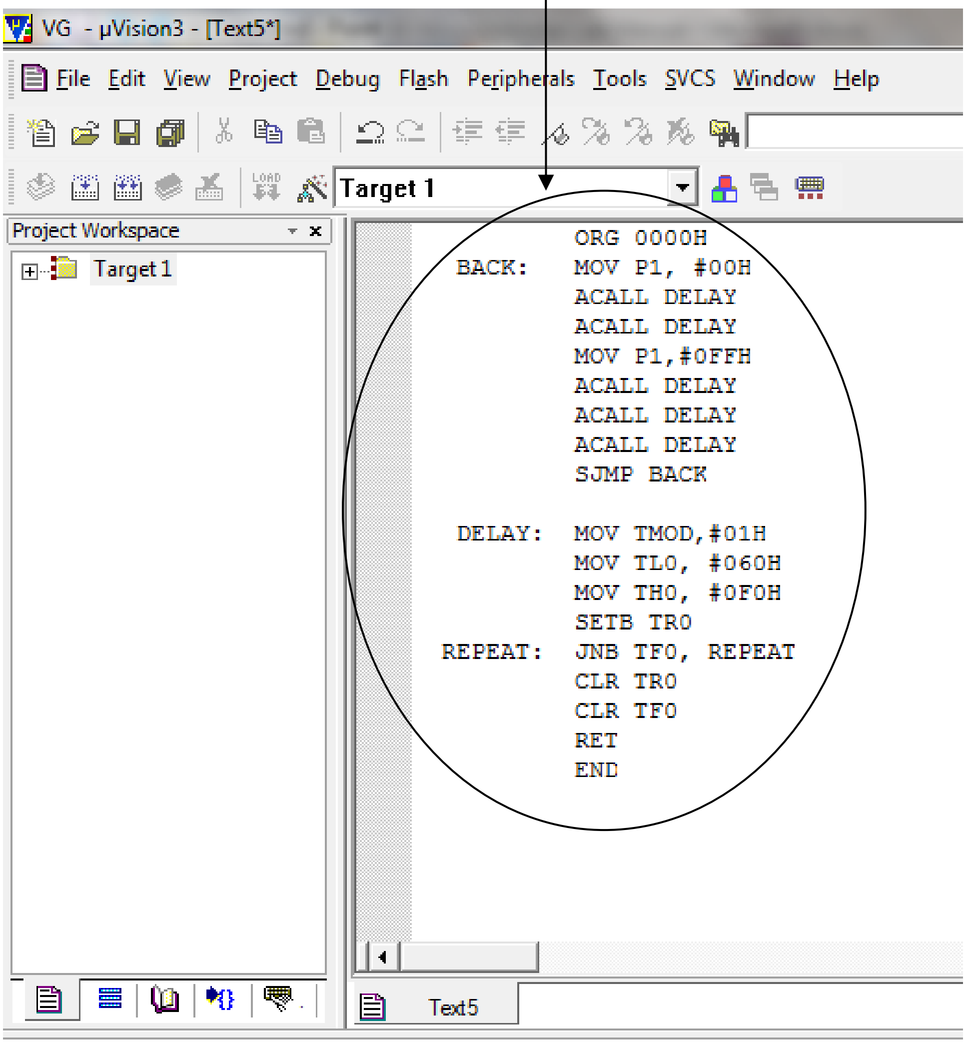
STEP 6: Make sure that **"STARTUP.A51"** file is added to the target.



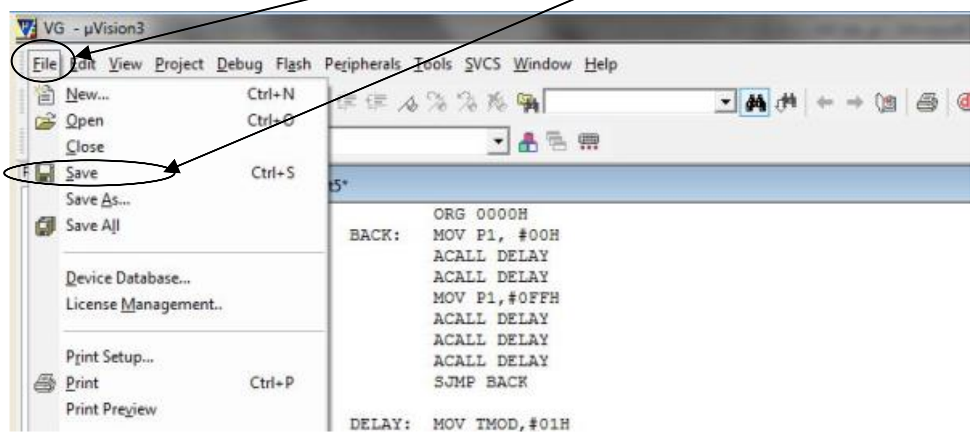
STEP 7: Go to **"File"** and select **"New"** for text (program) **Editing Window**.



STEP 8: Type your program in the **editing window**.

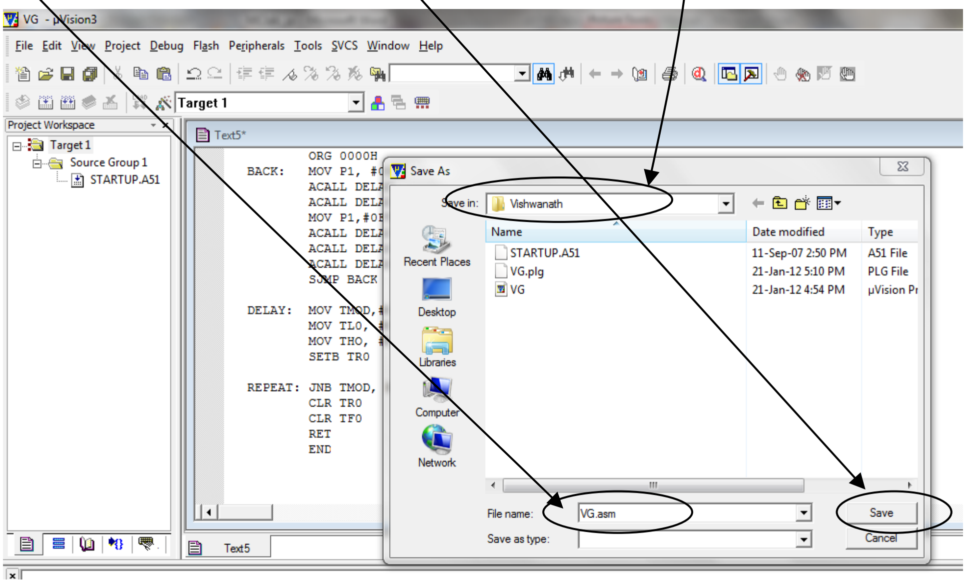


STEP 9: Save your program by going to **"File"** then **"Save"** option



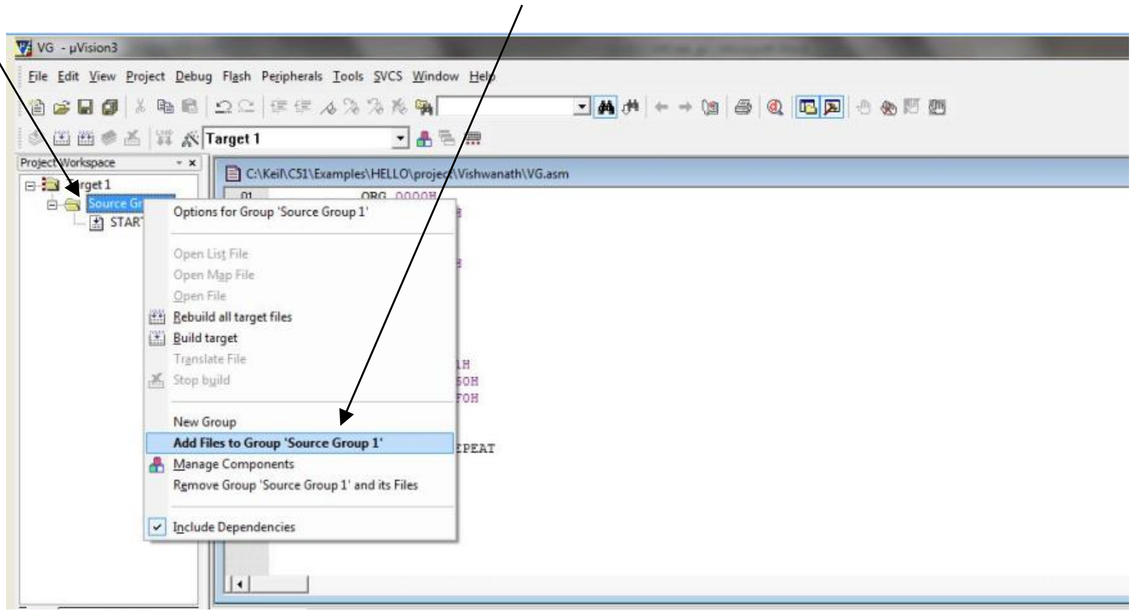
STEP 10:

- **"Save in"** your project folder.
- Give file name with **"*.asm" "extension".**
- And then click on **"Save"** option



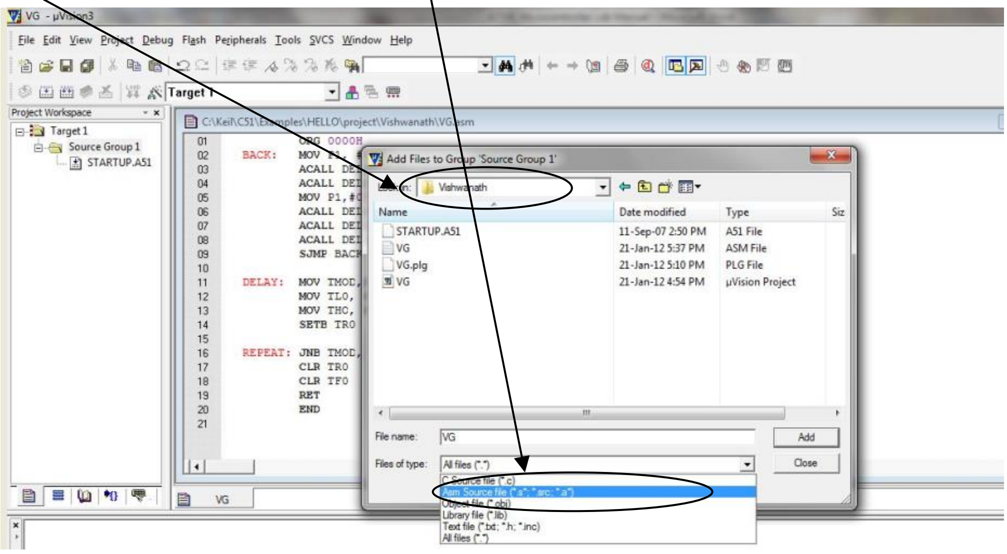
STEP 11:

- Right click on **"Source Group1"**
- Select "Add Files to **"Group Source Group 1".**



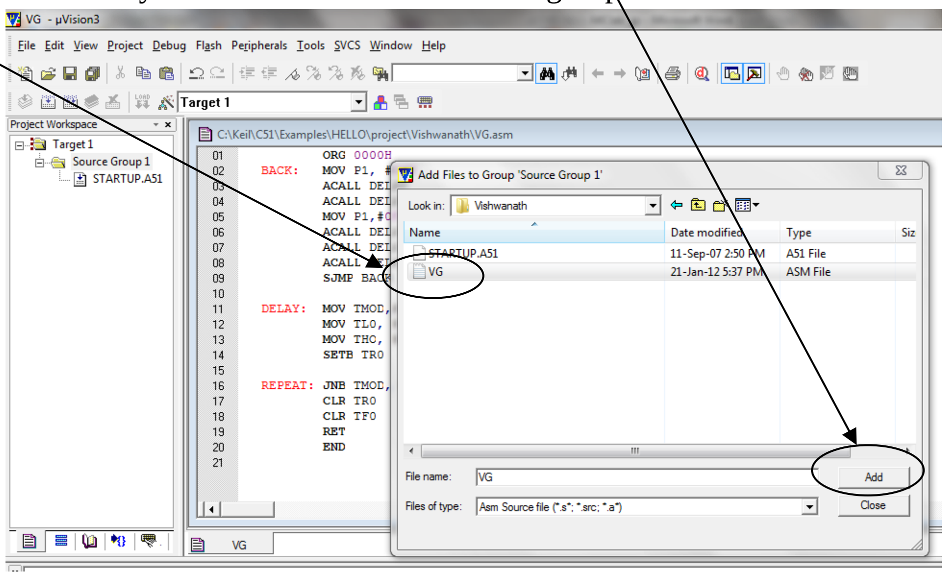
STEP 12:

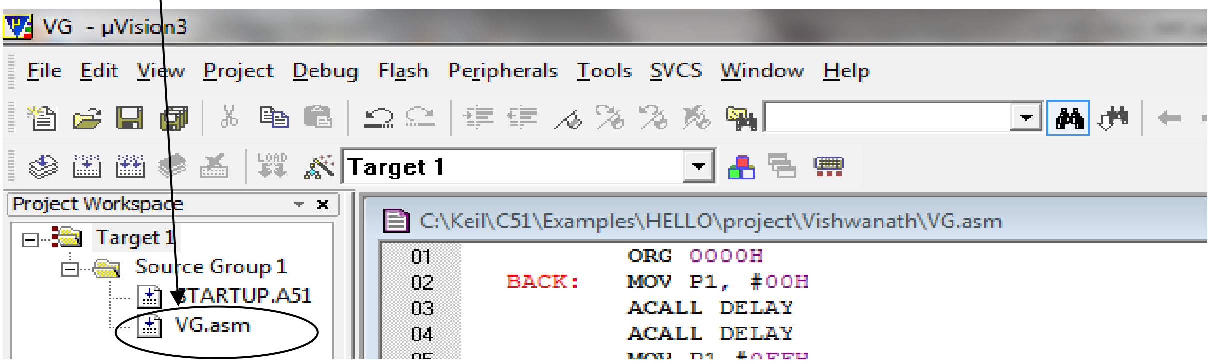
- Select to your Project folder
- Select **"Files of type"** as **"ASM source file"** if your program is written in assembly level language or else select **"C file"** if your program is in C language



STEP 13:

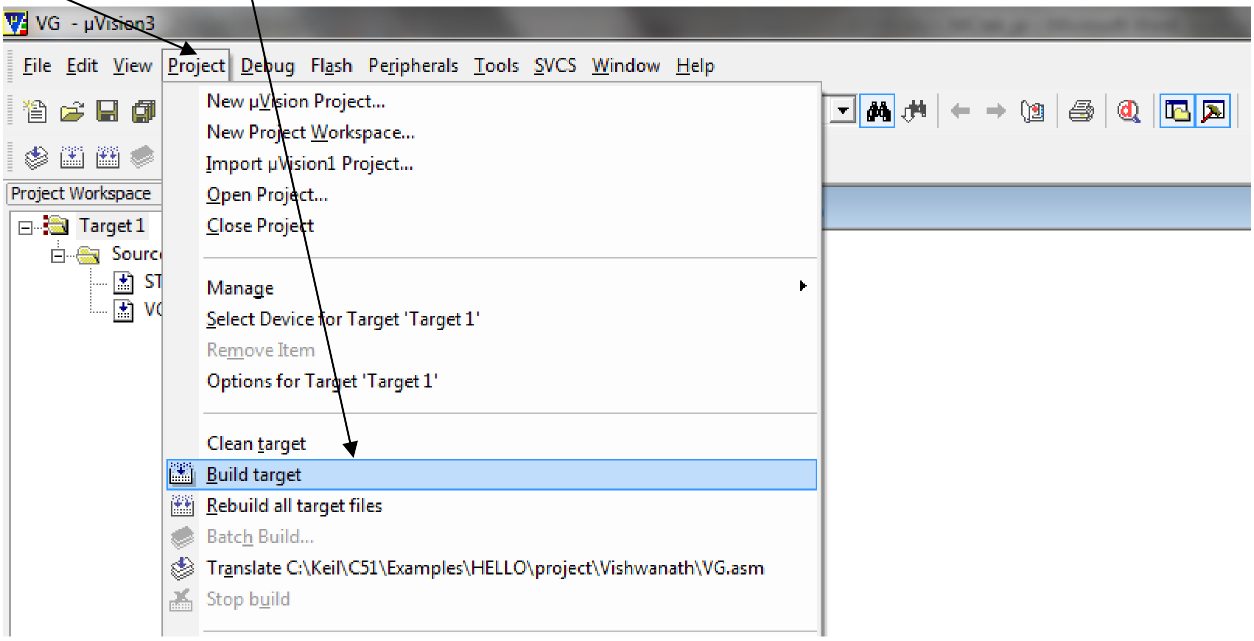
- Select your program file and then click on **"Add"** to add the file to your source group.
- Notice that your file is added to the Source group



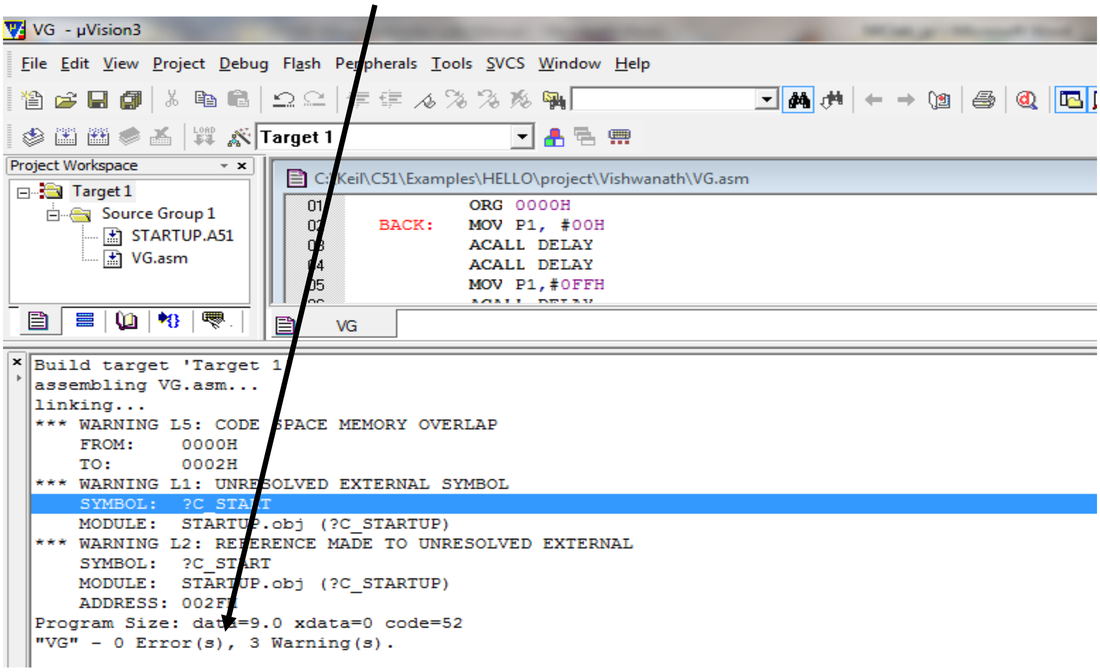


STEP 14: Build the target.

- Go to **“Project”.**
- Select **“Build Target”** or press **“F7”** key.

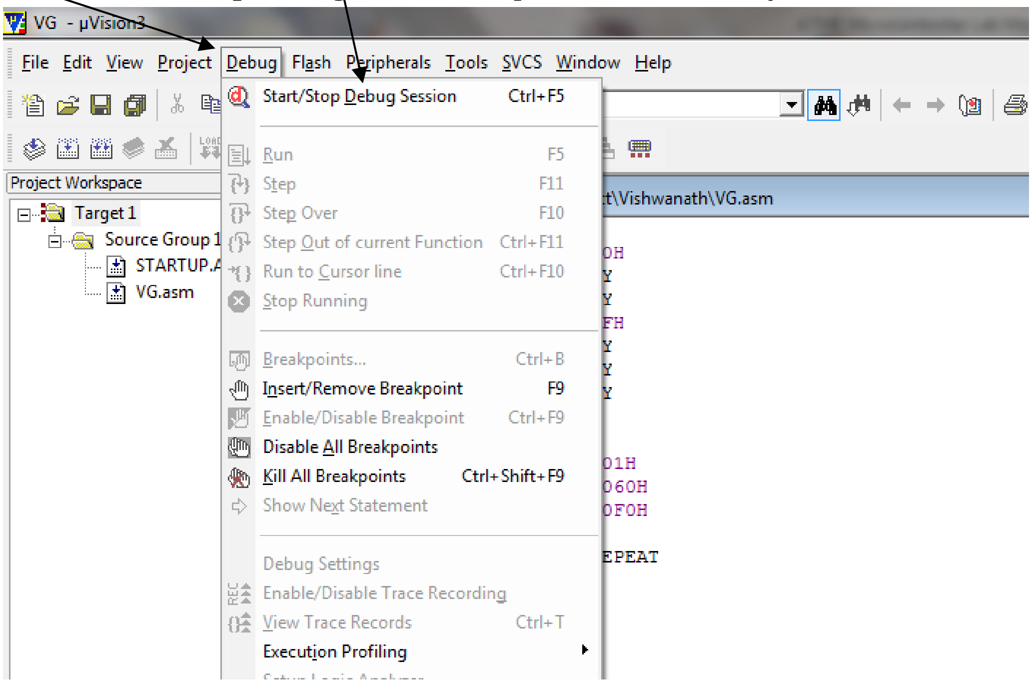


Important: After building the target check for the error(s). If there is any error(s) go back to your program, correct the error(s). The output window shows the line where error is found. After correcting the error go back to Step 14 and repeat the processes until there is zero error

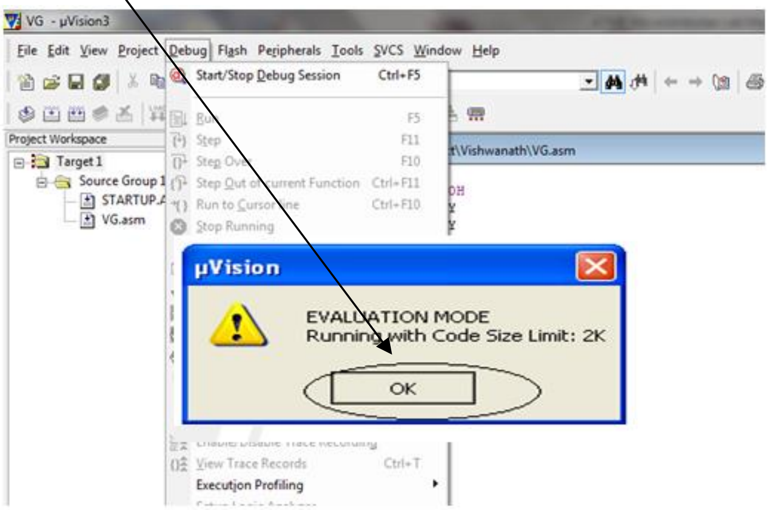


STEP 15: Debugging.

- Go to **"Debug".**
- Select **"Start/ Stop Debug Session"** or press **"Ctrl+F5"** key.



Select **"OK".**

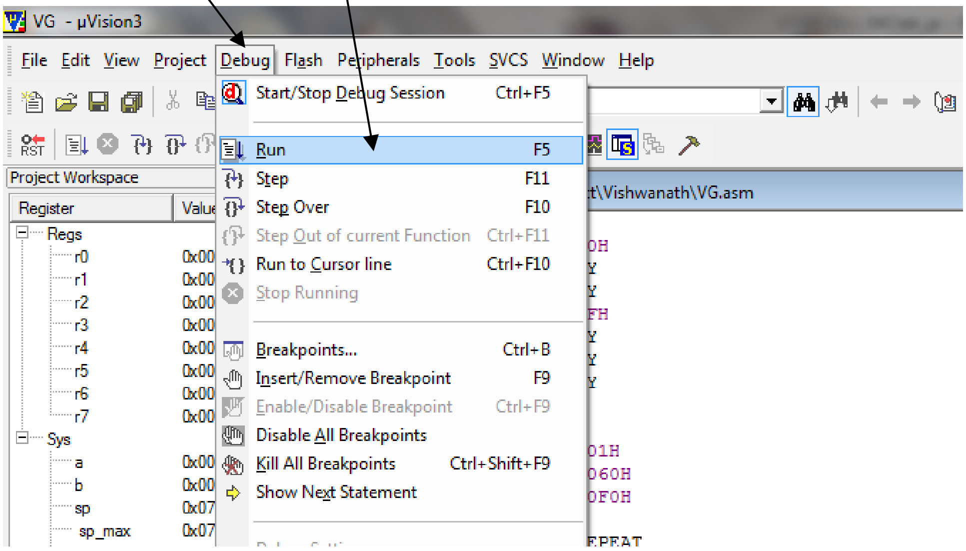


STEP 15: Selecting Output Window.

- Choose appropriate Output window (Memory/serial/logic analyzer) according to your program output.
- Type in the input parameters (memory address/ port address/ timer) according to your program.

STEP 16: Execution.

- Go to **"Debug"**, Select **"Run"** or press **"F5"** key for one time execution.
- For single step execution Press **"F11"**.



Outcome:

Before Execution

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0xFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x00
0x8201	0x00
0x8202	0x00
0x8203	0x00
0x8204	0x00
0x8205	0x00
0x8206	0x00
0x8207	0x00

After Execution

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0XFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x12
0x8201	0x24
0x8202	0x56
0x8203	0xFF
0x8204	0xEE
0x8205	0xAB
0x8206	0x10
0x8207	0x03

1.Data Transfer – Block move, Exchange, Sorting, Finding largest element in an Array

Program no 1: Data Transfer - Block move, Exchange

Objective: To transfer 8 bytes of data from external memory location starting from 8100h to external memory location starting from 8200h

Software: Keil µVision 3

```
        MOV R0, #08H      ; initialize the count
        MOV R1, #81H      ; initialize the source memory location higher byte
        MOV R2, #82H      ; initialize the destination memory location higher byte
        MOV R3, #00H      ; initialize the destn& source location lower byte
BACK:   MOV DPH, R1       ; get the source memory location address to DPTR
        MOV DPL, R3
        MOVX A, @DPTR     ; get the data from source memory to Accumulator
        MOV DPH, R2       ; get the destination memory location address to DPTR
        MOVX @DPTR, A     ; copy the accumulator content to destination memory
        INC R3            ; increment to next source and destination memory
        DJNZ R0, BACK     ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Algorithm

1. Initialize registers to hold count data & also the source & destination addresses.

2. Get data from source location into accumulator and transfer to the destination location.

3. Decrement the count register and repeat step 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.

Outcome:

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0XFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03
Before exec	

Address	Data
0x8200	0x12
0x8201	0x24
0x8202	0x56
0x8203	0xFF
0x8204	0xEE
0x8205	0xAB
0x8206	0x10
0x8207	0x03
After Exe	

At the end of the program

1. Students will be able to program for data movement

Result: At the end of the Program execution, block of data is transferred from source memory to destination memory

Program no:2

Objective: To exchange 8 bytes of data between external memories location starting from 8100h and external memory location starting from 8200h

```
        ORG 0000H
        MOV R0, #08H        ; initialize the count
        MOV R1, #81H        ; initialize the memory1 location higher byte
        MOV R2, #82H        ; initialize the memory2 location higher byte
        MOV R3, #00H        ; initialize the memory1&memory2 location lower byte
BACK:   MOV DPH, R1         ; get the memory1 location address to DPTR
        MOV DPL, R3
        MOVX A, @DPTR       ; get the data from memory1 to Accumulator
        MOV B,A             ; copy the accumulator content to B register
        MOV DPH, R2         ; get the memory2 location address to DPTR
        MOVX A,@DPTR        ; get the data from memory2 to Accumulator
        XCH A,B             ; exchange the accumulator and B register content
        MOVX @DPTR,A        ; copy the accumulator content to memory2
        MOV A,B             ; get the B register content to accumulator
        MOV DPH,R1          ; get the memory1 location address to DPTR
        MOVX @DPTR, A       ; copy the accumulator content to memory1
        INC R3              ; increment to next source and destination memory
        DJNZ R0, BACK       ; decrement count. If count! =0 go to label "BACK"
        SJMP $
        END
```

Algorithm

1. Initialize registers to hold count data (array size) & also the source & destination addresses.
2. Get data from source location into accumulator and save in a register.
3. Get data from the destination location into accumulator.
4. Exchange the data at the two memory locations.
5. Decrement the count register and repeat steps till count is zero.

Outcome: Program No: 2

Before Execution

OUTCOME

Address	Data
0x8100	0x12
0x8101	0x24
0x8102	0x56
0x8103	0xFF
0x8104	0xEE
0x8105	0xAB
0x8106	0x10
0x8107	0x03

Address	Data
0x8200	0x32
0x8201	0xFF
0x8202	0xAD
0x8203	0xDA
0x8204	0x88
0x8205	0x99
0x8206	0x56
0x8207	0x55

After Execution

OUTCOME

Address	Data
0x8100	0x32
0x8101	0xFF
0x8102	0xAD
0x8103	0xDA
0x8104	0x88
0x8105	0x99
0x8106	0x56
0x8107	0x55

Address	Data
0x8200	0x12
0x8201	0x24
0x8202	0x56
0x8203	0xFF
0x8204	0xEE
0x8205	0xAB
0x8206	0x10
0x8207	0x03

At the end of the program

Students will be able to program for data exchange between two external memory locations

Result

After execution data stored in 8 memory location of both 8100h-8107h and 8200h-8207h gets interchanged.

Data Transfer – Largest/Smallest element in an Array

Program no: 3

Objective: To find the largest number in a given array of size 5 starting from 5100h external memory location. The largest number has to be stored in 8100h external memory location

```
        ORG 0000H
        MOV R1,#04H          ; initialize the count
        MOV DPTR, #5100H     ; initialize the external memory location
        MOVX A,@DPTR         ; get the data from memory to accumulator
 BACK:  MOV B,A              ; move the content from accumulator to B register
        INC DPTR             ; increment the external memory location
        MOVX A,@DPTR         ; get the data from memory to accumulator
        CJNE A,B,NEXT        ; compare accumulator content and B register content, if not
                               equal Jump to label 'NEXT'
        DJNZ R1,BACK         ; if A & B are equal, then decrement count, if count! =0
                               Jump to label 'BACK'
        SJMP LAST            ; If count=0, then short jump to label' LAST'
NEXT:   JNC L2               ; If A & B are not equal, then check CY=1(A<B)
                             ;                  If CY! =1(A>B) jump to label 'L2'
        XCH A,B              ; If CY=1, Exchange A & B
L2:     DJNZ R1, BACK        ; Decrement count, if count! =0, jump to label,' BACK'
LAST:   MOV DPTR, #8100H     ; Initialize new memory location for storing largest data
        MOVX @DPTR,A         ; move the largest data from accumulator to new memory
                               Location.
        SJMP $
        END
```

Outcome: Program No: 3

Before execution	***After execution***	
Address / **Data**	**Address** / **Data**	
0x5100 / 0x12	0x8100 / 0xFF	**For largest**
0x5101 / 0x24		
0x5102 / 0x56		
0x5103 / 0xFF		
0x5104 / 0xEE		

At the end of the program

Students will be able to program for determining the largest number in an given array

Result:

At the end of the program, the largest number in a given array of size 5 starting from 5100h external memory location is entered & the largest number has to be stored in 8100h external memory location

Program no: 4

Objective: To find the smallest number in a given array of size 5 starting from 5100h external memory location. The largest number has to be stored in 8100h external memory location

```
        ORG 0000H
        MOV R1,#04H          ; initialize the count
        MOV DPTR, #5100H     ; initialize the external memory location
        MOVX A,@DPTR         ; get the data from memory to accumulator
BACK:   MOV B,A              ;move the content from accumulator to B register
        INC DPTR             ;increment the external memory location
        MOVX A,@DPTR         ; get the data from memory to accumulator
        CJNE A,B, NEXT       ; compare accumulator content and B register content, if not
                             equal Jump to label 'NEXT'
        DJNZ R1,BACK         ;if A & B are equal, then decrement count, if count! =0
                             Jump to label 'BACK'
        SJMP LAST            ;if count=0, then short jump to label' LAST'
NEXT:   JC L2                ; If A& B are not equal, then check CY=1(A<B)
                             If CY=1jump to label 'L2'
        XCH A,B              ;If CY! =1, Exchange A & B
L2:     DJNZ R1, BACK        ; Decrement count, if count! =0, jump to label,' BACK'
LAST:   MOV DPTR, #8100H     ; Initialize new memory location for storing smallest data
        MOVX @DPTR, A        ; move the smallest data from accumulator to new memory
                             Location
        SJMP $
        END
```

Outcome: Program no: 4

Before Execution

Note: Replace JNC by JC to find smallest number in a given array.

Address	**Data**
0x5100	0x12
0x5101	0x24
0x5102	0x56
0x5103	0xFF
0x5104	0xEE

After Execution

For smallest

Address	**Data**
0x8100	0x12

At the end of the program

Students will be able to program for determining the smallest number in an given array

Result:

At the end of the program, the smallest number in a given array of size 5 starting from 5100h external memory location is entered & the smallest number has to be stored in 8100h external memory location

Data Transfer –Sorting

Program no: 5

Objective: The array of data which has to be arranged in the ascending order starts from 5100h external memory location. The array contains 5 data's. Rearrange the data in the ascending order

```
        ORG 0000H
        MOV R1, #04H          ; initialize the step count
L1:     MOV A, R1             ; move the count to accumulator
        MOV R2, A             ; move accumulator content to R2 (comparison)
        MOV DPTR, #5100H      ; Initialize the external memory location
L2:     MOVX A,@DPTR          ; get the data from memory to accumulator
         MOV B,A              ; move the accumulator content to B register
        INC DPTR              ; increment the external memory location.
        MOVX A, @DPTR         ; get the data from memory to accumulator
        CJNE A, B, L3         ; compare accumulator content and B register content, if not
                                equal Jump to label 'L3'
        SJMP L5               ; short jump to label L5
L3:     JC L4                 ; If A& B are not equal, then check CY! =1(A<B)
                               If CY =1(A>B) jump to label 'L4'
        SJMP L5                ; short jump to label L5
L4:     XCH A,B               ;Exchange A & B
       MOVX @DPTR, A          ; move accumulator content to external memory
        INC DPTR              ; increment the external memory location
L5:     DJNZ R2, L2           ; decrement comparison count, if count! =0 then jump to
                              ; label L2'.
        DJNZ R1, L1           ; decrement step count, if count! =0 then jump to   label 'L1'
        SJMP $
        END
```

Outcome: Program no: 5

Before Execution

Address	**Data**
0x5100	0x1F
0x5101	0xD4
0x5102	0x56
0x5103	0xFF
0x5104	0x01
before	

After Execution-Ascending

Address	**Data**
0x5100	0x01
0x5101	0x1F
0x5102	0x56
0x5103	0XD4
0x5104	0xFF

At the end of the program

Students will be able to program to sort number in an given array in ascending order.

Result

After execution, The array of data which has to be arranged in the ascending order starts from 5100h external memory location and the array contains 5 data rearranged in the ascending order

Program no: 6

Objective: The array of data which has to be arranged in the descending order starts from 5100h external memory location. The array contains 5 data’s. Rearrange the data in the ascending order

```
        ORG 0000H
        MOV R1, #04H        ; initialize the step count
L1:     MOV A,R1            ; move the count to accumulator
        MOV R2, A           ; move accumulator content to R2 (comparison)
        MOV DPTR, #5100H    ; Initialize external memory location
L2:     MOVX A,@DPTR        ; get the data from memory to accumulator
        MOV B,A             ; move the accumulator content to B register.
        INC DPTR            ; increment the external memory location.
        MOVX A, @DPTR       ; get the data from memory to accumulator
        CJNE A,B,L3         ;compare accumulator content and B register content, if not
                            ;   equal Jump to label ‘L3’
        SJMP L5             ; short jump to label L5
L3:     JNC L4              ;If  A& B are not equal, then check CY=1(A<B)
                            ;  If CY! =1(A>B) jump to label ‘L4’
        SJMP L5             ; short jump to label L5
L4:     XCH A,B             ;If CY! =1, Exchange A & B
        MOVX @DPTR,A        ; move the data from accumulator to external memory
        DEC DPL             ; decrement the lower byte of external memory
        XCH A,B             ;Exchange A & B
        MOVX @DPTR, A       ; move accumulator content to external memory
        INC DPTR            ; increment the external memory location
L5:     DJNZ R2, L2         ; decrement comparison count, if count! =0 then jump to
                            ; label’ L2’.
        DJNZ R1, L1         ; decrement step count, if count! =0 then jump to   label ‘L1’
        SJMP $
        END
```

Outcome: Program no: 6

Before Execution

Note: Replace JNC by JC for arranging the given data in ascending order.

Address	**Data**
0x5100	0x1F
0x5101	0xD4
0x5102	0x56
0x5103	0xFF
0x5104	0x01
before	

After Execution

Descending

Address	**Data**
0x5100	0xFF
0x5101	0xD4
0x5102	0x56
0x5103	0X1f
0x5104	0x01

At the end of the program

Students will be able to program to sort number in an given array in descending order.

Result

After execution, the array of data which has to be arranged in the descending order starts from 5100h external memory location and the array contains 5 data rearranged in the descending order

2. Arithmetic Instructions – Addition, Subtraction, Multiplication and Division, Square, Cube – (16 Bits Arithmetic Operations – Bit Addressable)

Program no: 7

Objective: To add two 8 bit numbers placed in 8100h and 8101h external memory location. The Outcome has to be stored in 8200h and 8201h external memory location.

```
ORG 0000H
MOV DPTR, #8100H      ; initialize external memory location
MOVX A,@DPTR          ; get the data from memory to accumulator
MOV B, A              ; move the content from accumulator to B register
INC DPTR              ; increment the memory location
MOVX A, @DPTR         ; get the data from memory to accumulator
ADD A, B              ; add the content of A and B
MOV DPTR, #8201H      ; initialize new memory location
MOVX @DPTR, A         ; move the content from accumulator to memory
MOV A, #00H           ; move the value '00' to accumulator
ADDC A, #00H          ; add accumulator data with carry
DEC DPL               ; decrement lower byte of memory
MOVX @DPTR, A         ; move the accumulator content to memory
SJMP $
END
```

Outcome:

Before Execution		***After Execution***	
Address	**Data**	**Address**	**Data**
0x8100	0xFF	0x8200	0x01
0x8101	0xFF	0x8201	0xFE

At the end of the program

Students will be able to understand practical utilization of 8 bit Addition

Result: *Addition of two 8 bit numbers placed in 8100h and 8101h external memory location is performed and the Outcome is stored in 8200h and 8201h external memory location.*

Program no: 8

Objective: To add two 16 bit numbers, first 16 bit number placed in 8100h and 8101h external memory location and second 16 bit number placed in 8200h and 8201h external memory location. The Outcome has to be stored in 8300h, 8301h and 8302h external memory location.

```
ORG 0000H
MOV DPTR,#8101H       ; initialize the external memory location
MOVX A,@DPTR          ; get the 1st LSB data from memory to accumulator
MOV B,A               ; move the content from accumulator to B register
MOV DPTR,#8201H       ; initialize new memory location
MOVX A,@DPTR          ; get the 2nd LSB data from memory to accumulator
ADD A,B               ; add the content of A and B
MOV DPTR,#8302H       ; initialize new memory location
MOVX @DPTR,A          ; move the accumulator content to memory
MOV DPTR,#8100H       ; initialize new memory location
MOVX A,@DPTR          ; get the 1st MSB data from memory to accumulator
MOV B,A               ; move the content from accumulator to B register
MOV DPTR,#8200H       ; initialize new memory location
MOVX A,@DPTR          ; get the 2nd MSB data from memory to accumulator
ADDC A,B              ; add the content of A and B with carry
MOV DPTR,#8301H       ; initialize new memory location
MOVX @DPTR,A          ; move the accumulator content to memory
MOV A,#00H            ; move the value '00' to accumulator
ADDC A,#00H           ; add accumulator data with carry
DEC DPL               ; decrement lower byte of memory
MOVX @DPTR,A          ; move the accumulator content to memory
SJMP $
END
```

Outcome Program No: 8

Before Execution **After Execution**

Before execution	
Address	**Data**
0x8100	0xFF
0x8101	0xFF
Before execution	
0x8200	0xFF
0x8201	0xFF
Before execution	
0x8300	0x00
0x8301	0x00
0x8301	0x00

Address	**Data**
0x8300	0x01
0x8301	0xFF
0x8301	0xFE

At the end of the program

Students will be able to understand practical utilization of 16 bit Addition

Result

Addition of two 16 bit numbers is performed, first 16 bit number placed in 8100h and 8101h external memory location and second 16 bit number placed in 8200h and 8201h external memory location. The Outcome is stored in 8300h, 8301h and 8302h external memory location.

Program no: 9

Objective: To subtract two 8 bit numbers placed in 8100h and 8101h external memory location. The Outcome has to be stored in 8200h and 8201h external memory location. The 8200h memory location indicates the sign of the Outcome.

```
ORG 0000H
MOV DPTR, #8100H        ; initialize external memory location
MOVX A,@DPTR            ; get the data from memory to accumulator
MOV B,A                 ; move the content from accumulator to B register
INC DPTR                ; increment the memory location
MOVX A,@DPTR
SUBB A, B               ; Subtract the content of B from Accumulator with borrow
MOV DPTR, #8201H        ; initialize new memory location
MOVX @DPTR, A           ; move the content from accumulator to memory
MOV A, #00H             ; move the value '00' to accumulator
SUBB A, #00H            ; subtract '00' from A with borrow
DEC DPL                 ; decrement lower byte of memory location
MOVX @DPTR, A           ; move the accumulator content to memory location
SJMP $
END
```

Outcome:

CASE 1: Negative Outcome		***CASE 2: Positive Outcome***	
Before Execution		***Before Execution***	
Address	**Data**	**Address**	**Data**
0x8100	0x02	0x8100	0x01
0x8101	0x01	0x8101	0x02
0x8200	0x00	0x8200	0x00
0x8201	0x00	0x8201	0x00
After Execution		***After Execution***	
Address	**Data**	**Address**	**Data**
0x8100	0x02	0x8100	0x02
0x8101	0x01	0x8101	0x01
0x8200	0xFF	0x8200	0x00
0x8201	0xFF	0x8201	0x01

At the end of the program

Students will be able to understand subtraction of two 8 bit numbers

Result

Subtraction of two 8 bit numbers placed in 8100h and 8101h external memory location is performed and the Outcome is stored in 8200h and 8201h external memory location.

Program no: 10

Objective: To subtract two 16 bit numbers, first 16 bit number placed in 8100h and 8101h external memory location and second 16 bit number placed in 8200h and 8201h external memory location. The Outcome has to be stored in 8300h, 8301h and 8302h external memory location. The 8300h memory location indicates the sign of the Outcome.

```
ORG 0000H
MOV DPTR,#8101H      ; initialize the external memory location
MOVX A,@DPTR         ; get the 1st LSB data from memory to accumulator
MOV B,A              ; move the content from accumulator to B register
MOV DPTR,#8201H      ; initialize new memory location
MOVX A,@DPTR         ; get the 2nd LSB data from memory to accumulator
SUBB A,B             ; Subtract the content of B from Accumulator with borrow
MOV DPTR,#8302H      ; initialize new memory location
MOVX @DPTR,A         ; move the accumulator content to memory
MOV DPTR,#8100H      ; initialize new memory location
MOVX A,@DPTR         ; get the 1st MSB data from memory to accumulator
MOV B,A              ; move the content from accumulator to B register
MOV DPTR,#8200H      ; initialize new memory location
MOVX A,@DPTR         ; get the 2nd MSB data from memory to accumulator
SUBB A,B             ; Subtract the content of B from Accumulator with borrow
MOV DPTR,#8301H      ; initialize new memory location
MOVX @DPTR,A         ; move the accumulator content to memory
MOV A,#00H           ; move the value '00' to accumulator
SUBB A,#00H          ; subtract '00' from A with borrow
DEC DPL              ; decrement lower byte of memory location
MOVX @DPTR,A         ; move the accumulator content to memory
SJMP $
END
```

Outcome Program no: 10

CASE 1:
Negative Outcome

Before execution	
Address	**Data**
0x8100	0x23
0x8101	0x12
0x8200	0x12
0x8201	0x45
0x8300	0x00
0x8301	0x00
0x8302	0x00

Before execution	
Address	**Data**
0x8300	0XFF
0x8301	0XEF
0x8302	0x33

CASE 2:
Positive Outcome

Before execution	
Address	**Data**
0x8100	0x12
0x8101	0x45
0x8200	0x23
0x8201	0x12
0x8300	0x00
0x8301	0x00
0x8302	0x00

Before execution	
Address	**Data**
0x8300	0X00
0x8301	0X10
0x8302	0xCD

At the end of the program

Students will be able to understand 16 bit subtraction of positive and negative outcome.

Result

Subtraction of two 16 bit numbers is performed, first 16 bit number placed in 8100h and 8101h external memory location and second 16 bit number placed in 8200h and 8201h external memory location. The Outcome is stored in 8300h, 8301h and 8302h external memory location.

Program no: 11

Objective: To multiply two 8 bit numbers placed in external memory location 8100h and 8101h. The Outcome will be stored in external memory location 8200h and 8201h.

```
ORG 0000H
MOV DPTR, #8100H        ; initialize the external memory location
MOVX A,@DPTR            ; get the data from memory to accumulator
MOV B,A                 ; move the content from accumulator to B register
INC DPTR                ; increment the memory location
MOVX A,@DPTR            ; get the data from memory to accumulator
MUL AB                  ; Multiply the content of A and B
MOV DPTR,#8201H         ;initialize the new memory location
MOVX @DPTR,A            ; move the accumulator content (LSB of multiplied
                        ; ans.) To memory location 8201h
MOV A,B                 ; Move B content (MSB of multiplied ans.) To A
DEC DPL                 ; decrement lower byte of memory location
MOVX @DPTR,A            ; move the accumulator content to memory location
SJMP $
END
```

Outcome Program no: 11

Before Execution

Address	Data
0x8100	0xFF
0x8101	0xFF

After Execution

Address	Data
0x8200	0xFE
0x8201	0x01

At the end of the program

Students will be able to understand 8 bit multiplication.

Result At the end of the execution *two 8 bit numbers are placed in external memory location 8100h and 8101h and the multiplication Outcome is stored in external memory location 8200h and 8201h*

Program no: 12

Objective: To multiply 8 bit number placed in external memory location 8100h with the 16 bit number placed in external memory location 8200h and 8201h .The Outcome will be stored in external memory location 8300h, 8301h and 8302h.

```
ORG 0000H
MOV DPTR,#8100H      ; initialize the external memory location
MOVX A,@DPTR         ; get the data from memory to accumulator
MOV B,A              ; move the content from accumulator to B register
MOV R0,A             ; get the multiplier to R0 register
MOV DPTR,#8201H      ; get the lower byte of multiplicand to accumulator
MOVX A,@DPTR
MUL AB               ; multiply multiply*lower byte multiplicand
MOV DPTR, #8302H     ;store the lower byte Outcome in Outcome+2 memory
MOVX @DPTR,A
MOV R1,B             ; move the upper byte Outcome in R1
MOV DPTR,#8200H      ; get the upper byte of multiplicand to accumulator
MOVX A,@DPTR
MOV B,R0             ; get the multiplier to B register
MUL AB               ; multiply multiply*upper byte multiplicand
ADDC A,R1            ;Add lower byte Outcome with R1 (upper byte Outcome of  lower multiplicand multiplication)
MOV DPTR,#8301H      ; store the Outcome in Outcome memory+1 location
MOVX @DPTR,A
MOV A,B              ; get the upper byte Outcome of upper multiplicand
ADDC A,#00H          ; add the carry to upper multiplicand Outcome
DEC DPL
MOVX @DPTR,A         ; store the Outcome in Outcome memory location
SJMP $
END
```

Outcome: Program no:12

Before Execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0xFF
0x8201	0xFF

Address	Data
0x8300	0x00
0x8301	0x00
0x8302	0x00

After Execution

Address	Data
0x8100	0xFF

Address	Data
0x8300	0xFE
0x8301	0xFF
0x8302	0x01

Address	Data
0x8200	0xFF
0x8201	0xFF

At the end of the program

Students will be able to understand Program to multiply 8bit number with 16 bit number.

Result

At the end of the execution, 8 bit number placed in external memory location 8100h is multiplied with the 16 bit number placed in external memory location 8200h and 8201h .The Outcome is stored in external memory location 8300h, 8301h and 8302h.

Program no: 13

Objective: To multiply 16 bit numbers placed in internal memory location 30h and 31h with the 16 bit number placed internal memory location 40h and 41h .The Outcome will be stored in internal memory location 50h, 51h, 52h and 53h.

```
      ORG 0000H
      MOV R2,#00H      ; clear R2 register
      MOV B,31H        ; get lower byte of input1 to register B
      MOV A,41H        ; get lower byte of input2 to register A
      MUL AB           ; multiply two inputs
      MOV 53H,A        ; store the lower byte Outcome+3 memory location
      MOV R0,B         ; save the partial Outcome1 in R0
      MOV B,31H        ; get lower byte of input1 to register B
      MOV A,40H        ; get upper byte of input2 to register A
      MUL AB           ; multiply two inputs
      MOV R1,B         ; store the partial Outcome2 in register R1
      ADD A,R0         ; add the partial Outcome1 with lower byte Outcome
      JNC L1           ; after addition if carry=0, jump to label "L1"
      INC R1           ; if carry! = 0, increment partial Outcome2
L1:   MOV R0,A         ; store the partial Outcome3 in R0
      MOV B,30H        ; get upper byte of input1 to register B
      MOV A,41H        ; get lower byte of input2 to register A
      MUL AB           ; multiply two inputs
      ADD A,R0         ; add partial Outcome3 with lower byte of the multiplied Outcome
      JNC L2           ; after addition if carry=0, jump to label "L1"
      INC R1           ; if carry! = 0, increment partial Outcome2
L2:   MOV 52H,A        ; store the partial Outcome3 in Outcome+2 memory location
      MOV A,B          ; get the upper byte of the Outcome to accumulator
      ADD A,R1         ; add partial Outcome2 with the accumulator content
      JNC L3           ; after addition if carry=0, jump to label "L1"
```

```
      INC R2            ; if carry! = 0, increment register R2
   L3:MOV R1,A          ; store the partial Outcome2 to register R1
      MOV B,30H         ; get upper byte of input1 to register B
      MOV A,40H         ; get upper byte of input2 to register A
      MUL AB            ; multiply two inputs
      ADD A,R1          ; add partial Outcome2 with the accumulator content
      JNC L4            ; after addition if carry=0, jump to label "L1"
      INC R2            ; if carry! = 0, increment register R2
L4:MOV 51H,A            ; store the lower byte Outcome+1 memory location
      MOV A,B           ; get the upper byte Outcome of the multiplication
      ADD A,R2          ; add the accumulator content with R2 content
      MOV 50H,A         ; store the upper byte Outcome in Outcome+0 memory location
      SJMP $
      END
```

Outcome:

Before Execution

Address	Data
0x0030	0xFF
0x0031	0xFF

Address	Data
0x0040	0xFF
0x0041	0xFF

Address	Data
0x0050	0x00
0x0051	0x00
0x0052	0x00
0x0053	0x00

After Execution

Address	Data
0x0030	0xFF
0x0031	0xFF

Address	Data
0x0040	0xFF
0x0041	0xFF

Address	Data
0x0050	0XFF
0x0051	0XFE
0x0052	0x00
0x0053	0x01

At the end of the program

Students will be able to understand Program to multiply two 16 bit numbers.

Result

At the end of the execution, 16 bit numbers placed in internal memory location 30h and 31h is multiplied with the 16 bit number placed internal memory location 40h and 41h .The Outcome is stored in internal memory location 50h, 51h, 52h and 53h.

Program no: 14

Objective: To perform 8 bit / 8bit division. Dividend is placed in external memory location 8200h, and divisor is placed in the external memory 8100h, the Outcome will be placed in the memory location 8300h (quotient) and 8301h (remainder)

```
ORG 0000H
MOV DPTR, #8100H        ; get the divisor data address
MOVX A, @DPTR           ; get the divisor to accumulator
MOV B, A                ; save the divisor in the register B
MOV DPTR, #8200H        ; get the dividend data address
MOVX A, @DPTR           ; get the dividend to accumulator
DIV AB                  ; divide A/B
MOV DPTR, #8300H        ;get the quotient memory address to DPTR
MOVX @DPTR, A           ; store the quotient in 8300h memory location
MOV A,B                 ; get the remainder to accumulator
INC DPTR                ; get the next address to store the remainder
MOVX @DPTR,A            ; store the remainder in 8301h memory location
SJMP $
END
```

Outcome:

Before Execution

Address	Data
0x8100	0x13
0x8200	0x45

After Execution

Address	Data
0x8300	0x03
0x8301	0x0C

At the end of the program

Students will be able to understand Program 8 bit / 8bit division.

Result : *At the end of the execution, Dividend is placed in external memory location 8200h, and divisor is placed in the external memory 8100h, the Outcome is placed in the memory location 8300h (quotient) and 8301h (remainder).*

Program no: 15

Objective: To find square of given number, input is placed in external memory location 8100h, and Outcome is placed in the external memory 8101h and 8102h.

```
ORG 0000H
MOV DPTR,#8100H        ; get the source address
MOVX A,@DPTR           ; get the input data to accumulator
MOV B, A               ; move the input data to B register
MUL AB                 ; get the square of the number
INC DPTR               ; get the Outcome+1 address to store the square Outcome
INC DPTR
MOVX @DPTR, A          ; save the lower byte of the Outcome
DEC DPL                ; get the Outcome memory location
MOV A, B               ; get the upper byte of the Outcome to the Accumulator
MOVX @DPTR, A          ; store the upper byte of the Outcome to memory location
SJMP $
END
```

Outcome:

Before Execution

Address	Data
0x8100	0xFF

After Execution

Address	Data
0x8101	0xFE
0x8102	0x01

At the end of the program

Students will be able to understand Program find square of a given numbers.

Result

At the end of the execution, input is placed in external memory location 8100h, and Outcome is placed in the external memory 8101h and 8102h

Program no: 16

Objective: To find cube of given number, input is placed in external memory location 8100h, and Outcome is placed in the external memory 8200h, 8201h and 8202h

```
ORG 0000H
MOV DPTR,#8100H       ; get the source address
MOVX A,@DPTR          ; get the input data to accumulator
MOV B, A              ; move the input data to B register
MOV R0,A              ; copy the input data to the register R0
MUL AB                ; get the square of the input number
MOV R1,B              ; copy the upper byte of the square Outcome in the R1 register
MOV B,R0              ; get the input data to register B
MUL AB                ; get the lower byte of the cube Outcome
MOV DPTR,#8202H       ; get the Outcome+2 memory location
MOVX @DPTR,A          ; store the lower byte of cube output in Outcome+2 memory
MOV R2,B              ; store the upper byte partial Outcome in R2
MOV B,R1              ; get the previous partial Outcome to register B
MOV A,R0              ; get the input to accumulator
MUL AB                ; get the second upper byte partial Outcome
ADDC A,R2             ; add the input data to the partial Outcome with the previous carry
DEC DPL               ; get the Outcome+1 memory location
MOVX @DPTR,A          ; store the 2nd  byte of cube output in Outcome+1 memory
MOV A,B               ; get the upper byte of the multiplied output to accumulator
ADDC A,#00H           ; add with the previous carry
DEC DPL               ; get the Outcome memory location
MOVX @DPTR, A         store the 3rd byte of cube output in Outcome memory
SJMP $
END
```

Outcome Program no: 16

Before Execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0X00
0x8201	0X00
0x8202	0x00

After Execution

Address	Data
0x8100	0xFF

Address	Data
0x8200	0XFD
0x8201	0X02
0x8202	0xFF

At the end of the program

Students will be able to understand Program to find cube of a given numbers.

Result

At the end of the execution, input is placed in external memory location 8100h, and Outcome is placed in the external memory 8200h, 8201h and 8202h

Program no: 17

Objective: To check the given number placed in external memory location 8100h is odd or even, If the given number is odd store FFh in R1 register else if even store 11h in R1 register.

```
        ORG 0000H
        MOV DPTR,#8100H     ; get the input data from source memory location
        MOVX A,@DPTR
        RRC A               ; get the 0th bit of input data to carry flag
        JC ODD              ; if 0th bit=1, input number is odd
        MOV R1, #11H        ; store "11" in R1 to indicate even number
        SJMP LAST
ODD:    MOV R1,#0FFH        ; store "FF" in R1 to indicate odd number
LAST:   SJMP $
        END
```

Outcome:

Case 1: Odd Number

Before	
Address	**Data**
0x8100	0xFF
R1	0x00

After	
Address	**Data**
0x8100	0xFF
R1	0xFF

Indicate Odd Number

Case 2: Even Number

Before	
Address	**Data**
0x8100	0xFE
R1	0x00

After	
Address	**Data**
0x8100	0xFF
R1	0x11

Indicate Even Number

At the end of the program

Students will be able to understand Program to find the given number is odd or even.

Result

At the end of the execution, the given number placed in external memory location 8100h is verified and if the given number is odd FFh is stored in R1 register else if even 11h is stored in R1 register

Program no: 18

Objective*:* To check the given number placed in external memory location 8100h is Positive or Negative., If the given number is Negative store FFh in R1 register else if Positive store 11h in R1 register.

```
            ORG 0000H
            MOV DPTR,#8100H     ; get the input data from source memory location
            MOVX A,@DPTR
            RLC A               ; get the 0th bit of input data to carry flag
            JC negative         ; if 0th bit=1, input number is negative
            MOV R1, #11H        ; store "11" in R1 to indicate positive number
            SJMP LAST
Negative:   MOV R1, #0FFH       ; store "FF" in R1 to indicate negative number
LAST:       SJMP $
            END
```

Outcome:

Case 1: Odd Number

Before	
Address	**Data**
0x8100	0xFF
R1	0x00

After	
Address	**Data**
0x8100	0xFF
R1	0xFF

Indicates Negative Number

Case 2: Even Number

Before	
Address	**Data**
0x8100	0xFE
R1	0x00

After	
Address	**Data**
0x8100	0xFF
R1	0x11

Indicates Positive Number

Note: "RRC A" instruction is used to find odd or even. If we replace it by "RLC A" and change the loop name from ODD to +ve, we can find the given number is positive or negative.

At the end of the program

Students will be able to understand Program to find the given number is Positive or Negative.

Result

At the end of the execution, the given number placed in external memory location 8100h is verified and if the given number is Positive FFh is stored in R1 register else if Negative. *11h is stored in R1 register.*

Program no: 19

Objective: To check the number of logical zeroes and ones in the given number placed in the external memory location 8100h. The number of logical ones is indicated in the R2 register and the number of logical zeroes is indicated in the register R3.

```
          ORG 0000H
          MOV DPTR,#8100H       ;get the input data from source memory location
          MOVX A,@DPTR
          MOV R1,#08H           ; keep the count in R1 to check 8 bits of input data
          MOV R2,#00H           ; counter for logical ones
          MOV R3,#00H           ; counter for logical zeroes
NEXTBIT:  RRC A                 ; get the LSB bit to carry flag
          JC ONES               ; If bit is one jump to label ONES
          INC R3                ; if no carry increment zero counter
          SJMP LAST
ONES:     INC R2                ; if no carry increment ones counter
LAST:     DJNZ R1, NEXTBIT      ; if all the 8 bits are not checked, go back to label NEXTBIT
          SJMP $
          END
```

Outcome:

Before execution

Address	**Data**
0x8100	0x72
R2	0x00
R3	0x00

After execution

Address	**Data**	
R2	0x04	→ Logical ones
R3	0x04	→ Logical zeros

At the end of the program

Students will be able to understand Program to find the logical ones and zeroes in the given number.

Result

At the end of the execution, the given number placed in the external memory location 8100h. The number of logical ones is indicated in the R2 register and the number of logical zeroes is indicated in the register R3.

Program no: 20

Objective: To generate the ten Fibonacci numbers. It should be stored in external memory location starting from 9400h

```
        ORG 0000H
        MOV R0,#09H          ; Set Counter to generate 10 Fibonacci numbers

        MOV DPTR,#9400H      ; initialize the memory location to store the Fibonacci series

        MOV R1,#00H          ; get the first number to R1

        MOV A,R1             ; get the first number to accumulator

        MOVX @DPTR,A         ; store the first Fibonacci number in memory.

        MOV A,#01H           ; get the second data to accumulator

BACK:   INC DPTR

        MOVX @DPTR,A         ; store the next data in memory+1 location

        MOV R2,A             ; store the present number inR2 register

        ADD A,R1             ; get the previous data to present data in accumulator

        DA A                 ; decimal adjust the Outcome

        MOV R1,02H           ; get the R2 content to R1 register

        DJNZ R0, BACK        ; loop back until count is zero

STOP:   SJMP STOP

        END
```

Outcome:

Address	Data
0x9400	0x00
0x9401	0x00
0x9402	0x00
0x9403	0x00
0x9404	0x00
0x9405	0x00
0x9406	0x00
0x9407	0x00
0x9408	0x00
0x9409	0x00

Before Execution

Address	Data
0x9400	0x00
0x9401	0x01
0x9402	0x01
0x9403	0x02
0x9404	0x03
0x9405	0x05
0x9406	0x08
0x9407	0x13
0x9408	0x21
0x9409	0x34

After Execution

At the end of the program

Students will be able to understand Program to find working of Fibonacci series.

Result

At the end of the execution, ten Fibonacci is stored in external memory location starting from 9400h.

3. Up/Down BCD/ Binary Counters

Program no: 21

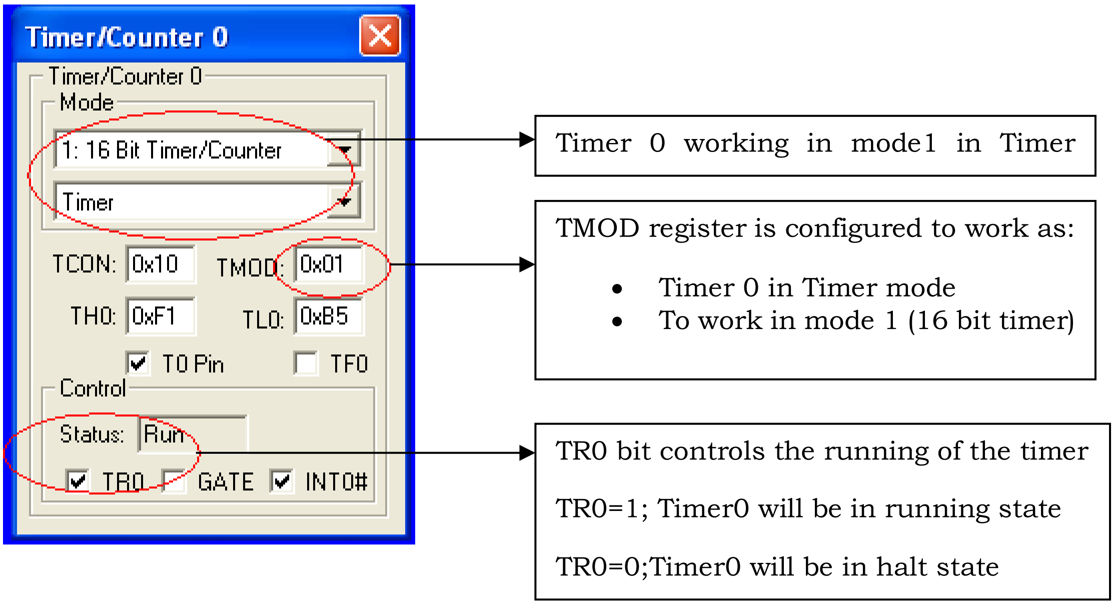
Objective: To display **BCD** up count **(00 to 99**) continuously in Port1. The delay between two counts should be 1 second. Configure TMOD register in Timer0 Mode1 configuration.

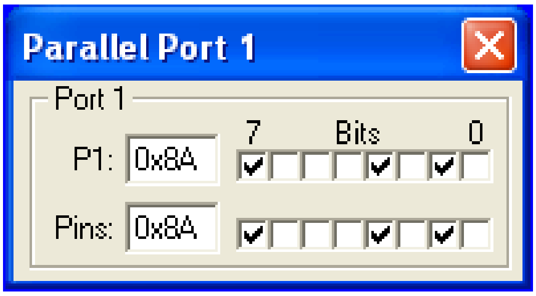
```
        ORG 0000H
        MOV A,#00H          ; A=00H
L1:     MOV P1,A            ; A=00H---→P1=00H
        ADD A,#01H          ; A=00H + 01H =01H→A
        DA A                ; 00 01 02 03 04 05 06 07 08 09 10
        LCALL DELAY         ;
        SJMP L1             ;
DELAY:  MOV TMOD,#01H       ; configure timer0 in mode1
        MOV R0, #1FH        ; get the count for repetition of timer register count
BACK:   MOV TL0, #00H       ; set the initial count for 1sec
        MOV TH0, #00H
        SETB TR0            ; start the timer
REPEAT: JNB TF0, REPEAT     ; wait until timer overflows
        CLR TR0             ; halt the timer
        CLR TF0             ; clear the timer0 overflow interrupt
        DJNZ R0, BACK       ; if repetition count!= 0, go to label back
        RET                 ; return to the main program
        END
```

Outcome: Program no: 21

Observe the BCD up count operation in Port1.

Sample view:





At the end of the program

1. Students will be able to understand the way in which subroutines are called and returns made in counters.
2. Analyze the calls and subroutines made in the program

Result

At the end of the execution, BCD up count is displayed continuously in Port1.

Program no: 22

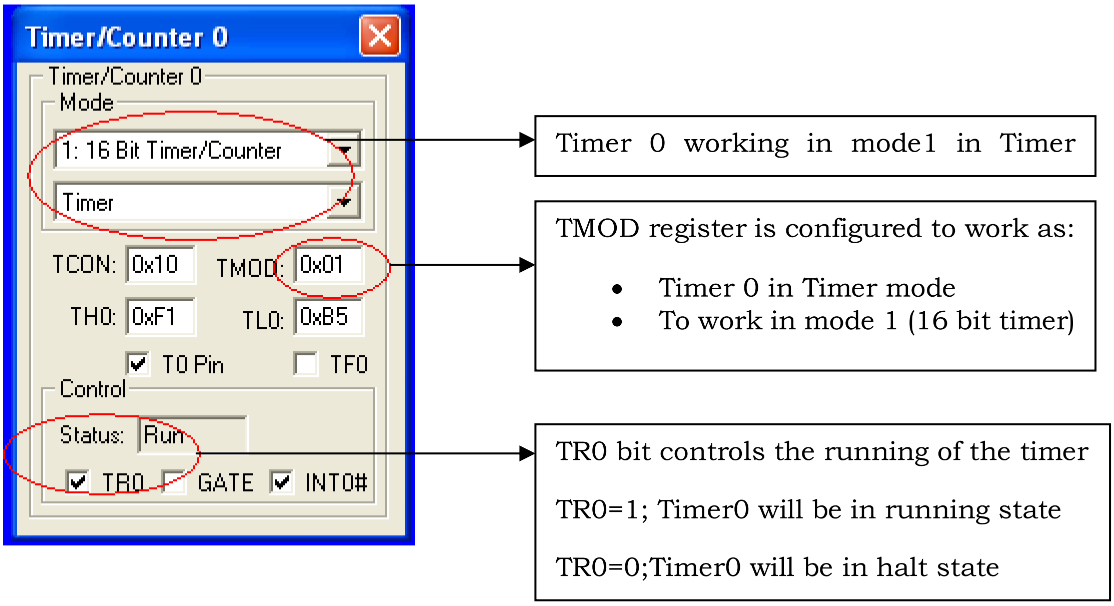
Objective: To display BCD down count (99 to 00) continuously in Port1. The delay between two counts should be 1 second. Configure TMOD register in Timer0 Mode1 configuration.

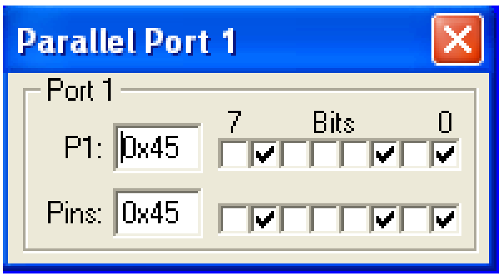
```
        ORG 0000H
        MOV A, #99H         ; get the first BCD value to accumulator
L1:     MOV P1, A           ; display the count in P1
        ADD A, #99H         ; get the next BCD down count value
        DA A                ; decimal adjust the count
        LCALL DELAY         ; call the delay of 1sec
        SJMP L1             ; repeat forever
DELAY:  MOV TMOD, #01H      ; configure timer0 in mode1
        MOV R0, #1FH        ; get the count for repetition of timer register count
BACK:   MOV TL0, #00H       ; set the initial count for 1sec
        MOV TH0, #00H
        SETB TR0            ; start the timer
REPEAT: JNB TF0, REPEAT     ; wait until timer overflows
        CLR TR0             ; halt the timer
        CLR TF0             ; clear the timer0 overflow interrupt
        DJNZ R0, BACK       ; if repetition count!= 0, go to label back
        RET                 ; return to the main program
        END
```

Outcome: Program no: 22

Observe the BCD down count operation in Port1.

Sample view:





At the end of the program

1. Students will be able to understand the way in which subroutines are called and returns made in counters.
2. Analyze the calls and subroutines made in the program

Result

At the end of the execution, BCD down count is displayed continuously in Port1.

4. Boolean & Logical Instructions (Bit manipulations)

Program no: 23

Objective: To relies the Boolean expression $\bar{A}$BD+A$\bar{B}\bar{D}$+AB$\bar{D}$. And A=1, B=1, D=0. Store the input in the 00h, 01h and 02h bit memory location. Store the Outcome of $\bar{A}$BD in 03h bit memory location and store the Outcome of A$\bar{B}\bar{D}$ in 04h bit memory location. Store the final Outcome in 08h bit memory location.

```
ORG 0000H
SETB 00H        ; initialize input A=1
SETB 01H        ; initialize input B=1
CLR 02H         ;initialize input D=00
MOV C,01H       ; get B input to carry flag
ANL C,02H       ; AND D with B
ANL C,/00H      ; get the expression ĀBD
MOV 03H,C       ; store it in 03h bit memory location
MOV C,00H       ; get A input to carry flag
ANL C,02H       ; AND D with A
ANL C,/01H      ; get the expression AB̄D̄
MOV 04H,C       ; store it in 04h bit memory location
MOV C,00H       ; get A input to carry flag
ANL C,01H       ; AND B with A
ANL C,/02H      ; get the expression ABD̄
ORL C,03H       ;ABD̄ + ĀBD
ORL C, 04H      ;ABD̄ + ĀBD + AB̄D̄
MOV 08H,C       ; store the Outcome in the internal bit memory 08h
SJMP $
END
```

Outcome Program no: 23

Before Execution:

Bit addresses

20 byte address

07	06	05	04	03	02	01	00
0	0	0	0	0	0	0	0

$A\bar{B}D$ (04), $\bar{A}BD$ (03), D (02), B (01), A (00)

Bit addresses

21 byte address

0F	0E	0D	0C	0B	0A	09	08
0	0	0	0	0	0	0	0

$\bar{A}BD+A\bar{B}D+AB\bar{D}$. (08)

After Execution:

Bit addresses

20 byte address

07h	06h	05h	04h	03h	02h	01h	00h
0	0	0	0	0	0	1	1

$A\bar{B}D$ (04h), $\bar{A}BD$ (03h), D (02h), B (01h), A (00h)

Bit addresses

21 byte address

0Fh	0Eh	0Dh	0Ch	0Bh	0Ah	09h	08h
0	0	0	0	0	0	0	1

$\bar{A}BD+A\bar{B}D+AB\bar{D}$. (08h)

At the end of the program

Students will be able to write program to realize boolean expression.

.

Result

At the end of the execution, Boolean expression $\bar{A}BD+A\bar{B}D+AB\bar{D}$ is realized and the Outcome of $\bar{A}BD$ in 03h bit memory location and the Outcome of $A\bar{B}D$ in 04h bit memory location. The final Outcome in 08h bit memory location.

5. Code Conversion: BCD – ASCII; ASCII – Decimal; Decimal – ASCII; HEX – Decimal and Decimal - HEX

Program no: 24

Objective: To convert ASCII (30-39) number placed in internal memory location 20h to its equivalent unpacked BCD number (00-09). The Outcome as to be stored in internal memory location 40h.

```
ORG 0000H
MOV R0, #20H      ; get the source memory address in R0
MOV R1, #40H      ; get the destination memory address in R1
MOV A,@R0         ; @20H=33----→A=33H
XRL A, #30H       ;  A=33H   X-0R   30H   =03H  ----→A=03H
MOV @R1, A        ; A=03H------@40H=03H
SJMP $
END
```

Outcome:

Before Execution

Address	Data
0x0020	0x36

Address	Data
0x0040	0x00

After Execution

Address	Data
0x0020	0x36

Address	Data
0x0040	0x06

At the end of the program

Students will be able to understand program to convert ASCII number to its equivalent unpacked BCD number

Result

At the end of the execution, ASCII (30-39) numbers placed in internal memory location 20h is converted to its equivalent unpacked BCD number (00-09). The Outcome is stored in internal memory location 40h.

Program no: 25

Objective: To convert unpacked BCD number (00-09) placed in internal memory location 20h to its equivalent ASCII number (30-39). The Outcome as to be stored in internal memory location 40h.

```
ORG 0000H

MOV R0, #20H      ; get the source memory address in R0

MOV R1, #40H      ; get the destination memory address in R1

MOV A,@R0         ; get the BCD data from source memory to accumulator

ORL A, #30H       ; convert to ASCII by adding 30h to input BCD data

MOV @R1, A        ; store the ASCII Outcome in destination memory

SJMP $

END
```

Outcome:

Before Execution

Address	Data
0x0020	0x06
0x0040	0x00

After Execution

Address	Data
0x0040	0x36

At the end of the program

Students will be able to understand program to convert ASCII number to its equivalent unpacked BCD number

Result

At the end of the execution, ASCII (30-39) numbers placed in internal memory location 20h is converted to its equivalent unpacked BCD number (00-09). The Outcome is stored in internal memory location 40h.

Program no: 26

Objective: To convert unpacked BCD number (00-99) placed in internal memory location 20h to its equivalent ASCII number (30-39). The Outcome as to be stored in internal memory location 40h and 41h. 54 35 34

```
ORG 0000H
MOV R0,#20H      ; get the source memory address in R0  76
MOV R1,#40H      ; get the destination memory address in R1
MOV A,@R0        ; get the input data to accumulator
ANL A,#0F0H      ; mask off the lower nibble   01110110  AND 11110000  = O/P= 70
SWAP A           ; exchange upper and lower nibble   A=07
ORL A,#30H       ; convert upper nibble to ASCII   07 OR30   0111  OR 00110000
MOV @R1,A        ; send the ASCII data to destination memory  a=37
MOV A,@R0        ; get the input data to accumulator
ANL A,#0FH       ; mask off the upper nibble
ORL A,#30H       ; convert lower nibble to ASCII
INC R1           ; increment the destination memory location
MOV @R1,A        ; send the ASCII data to destination memory
SJMP $
END
```

Outcome:

Before Execution			*After Execution*	
Address	**Data**		**Address**	**Data**
0x0020	0x76		0x0020	0x76
0x0040	0x00		0x0040	0x37
0x0041	0x00		0x0041	0x36

At the end of the program

Students will be able to understand code conversion program from packed BCD number to its equivalent ASCII number.

Result

At the end of the execution, unpacked BCD number (00-99) placed in internal memory location 20h is converted to its equivalent ASCII number (30-39). The Outcome is stored in internal memory location 40h and 41h.

Program no: 27

Objective: To convert ASCII (30-39) number placed in internal memory location 20h and 21h to its equivalent packed BCD number (00-99). The Outcome as to be stored in internal memory location 40h

```
ORG 0000H
MOV R0,#20H     ; get the source memory address in R0
MOV R1,#40H     ; get the destination memory address in R1
MOV A,@R0       ; get the ASCII input data to accumulator
ANL A,#0FH      ; mask off the upper nibble (convert to unpacked BCD)
SWAP A          ; exchange upper and lower nibble
MOV R2,A        ; save the accumulator content in R2 register
INC R0          ; get the second input memory location
MOV A,@R0       ; get the second data to accumulator
ANL A,#0FH      ; mask off the upper nibble (convert to unpacked BCD)
ORL A, R2       ; convert the two unpacked BCD data to packed data
MOV @R1,A       ; store in Outcome memory location
SJMP $
END
```

Outcome:

Before execution	
Address	**Data**
0x0020	0x34
0x0021	0x33
0x0040	0x00

After execution	
Address	**Data**
0x0020	0x34
0x0040	0x33
0x0041	0x43

At the end of the program

Students will be able to understand code conversion program from ASCII number to its equivalent packed BCD number.

Result

At the end of the execution, ASCII (30-39) number placed in internal memory location 20h and 21h is converted to its equivalent packed BCD number (00-99). The Outcome is stored in internal memory location 40h.

Program no: 28

Objective: To convert the hexadecimal number placed in the external memory location 8100h to decimal number and store the Outcome in the external memory location 8200h and 8201h.

```
ORG 0000H
MOV DPTR,#8100H     ; get the input data (hex number ) memory location
MOVX A,@DPTR        ; get the input data to accumulator
MOV B,#0AH          ; get the divisor to B register
DIV AB              ; divide input data by 10d
MOV R1,B            ; store the remainder in register in R1
MOV B,#0AH          ; get the divisor to B register
DIV AB              ; divide the quotient of previous division by 10d
MOV R0,A            ; move the quotient to R0 register
MOV A,B             ; get the remainder to accumulator
SWAP A              ; interchange upper and lower nibble
ORL A,R1            ; concatenate units and tens place
MOV DPTR,#8201H     ; get the Outcome+1 memory location
MOVX @DPTR,A        ; store the tens and units(accumulator) place Outcome
DEC DPL             ; get the Outcome+0 memory address
MOV A,R0            ; get the hundreds place value of the output to accumulator
MOVX @DPTR,A        ; store the Outcome.
SJMP $
END
```

Outcome: Program no: 28

Before Execution

Address	Data
0x8200	0x02
0x8201	0x55

After Execution

Address	Data
0x8100	0xFF
0x8200	0x00
0x8201	0x00

At the end of the program

Students will be able to understand code conversion program from hexadecimal number to decimal number.

Result

At the end of the execution, hexadecimal number placed in the external memory location 8100h is converted to decimal number and the Outcome is stored in the external memory location 8200h and 8201h.

Program no: 29

Objective: To convert the decimal number placed in the external memory location 8100h to hexadecimal number and store the Outcome in the external memory location 8101h

```
ORG 0000H
MOV DPTR,#8100H      ; get the input data (decimal number) memory location
MOVX A,@DPTR         ; get the input data (decimal number) to accumulator
MOV B,A              ; get the data to register B
ANL A,#0FH           ; mask off the upper nibble of the input data
MOV R1,A             ; save the accumulator data in register R1
MOV A,B              ; get the input data to accumulator
ANL A,#0F0H          ; mask off the lower nibble
SWAP A               ; interchange the upper and lower nibble
MOV B,#0AH           ; get the multiplier to register B
MUL AB               ; multiply upper nibble of input data with 0Ah
ADD A,R1             ; add multiplied data with input data's lower nibble value
INC DPTR             ; get the Outcome memory location address to DPTR
MOVX @DPTR,A         ; store the hex decimal value in the Outcome memory location
SJMP $
END
```

Outcome:

Before Execution

Address	Data
0x8100	0x63
0x8101	0x3F

After execution

Address	Data
0x8100	0x63
0x8101	0x00
0x8201	0x00

At the end of the program

Students will be able to understand code conversion decimal number to hexadecimal number.

Result

At the end of the execution, decimal number is placed in the external memory location 8100h and the converted result is stored in the external memory location 8101h

6.Programs to generate delay, Programs using serial port and on-Chip timer / counter

Program no: 30

Objective: To generate the square wave in P1 with the 50% duty cycle and the time delay of 10ms using timer. Assume the crystal frequency of 24 MHz Configure the timer in Timer0 mode1.

```
                ORG 0000H
                MOV P1, #0FFH       ; initialize P1
BACK:           XRL 90H, #0FFH      ; generate square wave signal
                ACALL DELAY         ; call 10ms delay
                SJMP BACK           ; repeat forever
DELAY:          MOV TMOD, #01H      ; configure the timer0 in mode1
                MOV TL0, #0E0H      ; set the initial value in timer register for 10ms
                MOV TH0, #0B1H
                SETB TR0            ; start the timer
REPEAT:         JNB TF0, REPEAT     ; wait until timer overflows
                CLR TR0             ; halt the timer
                CLR TF0             ; clear the timer0 overflow interrupt
                RET                 ; ret to the main program
                END
```

Outcome: Program no: 30

Observed the 50% duty cycle square wave in P1 and measured the time delay of 10ms.

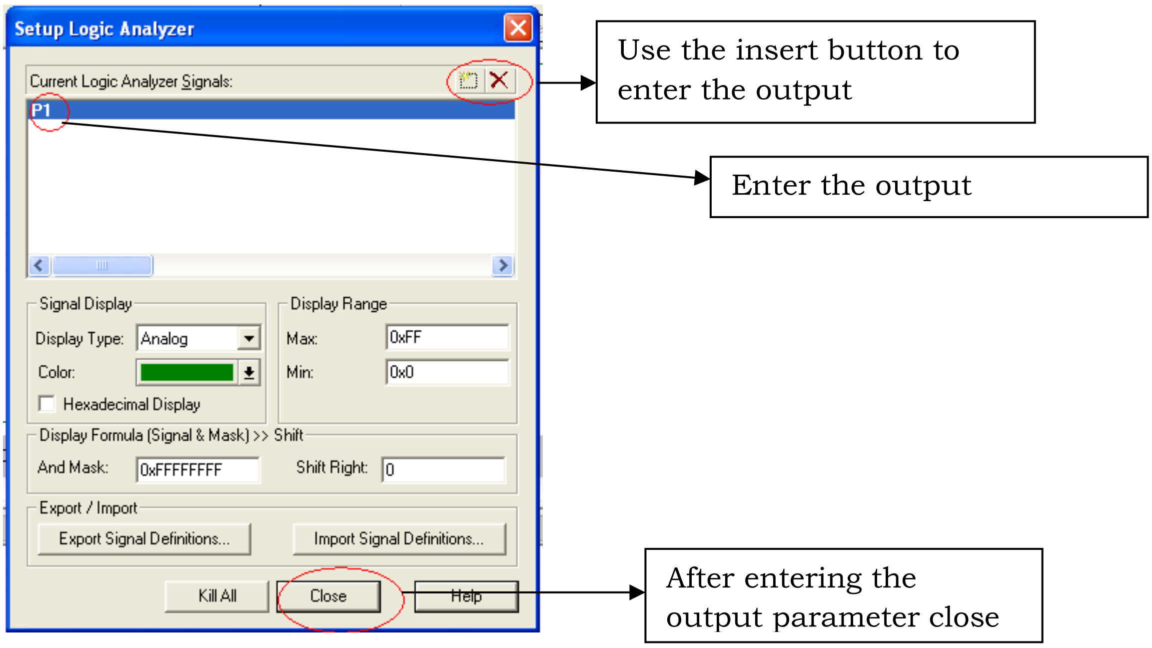
At the end of the program

Students will be able to program generating delays using timers and serial programming

Result

At the end of the execution, square wave is generated in P1 with the 50% duty cycle and the time delay of 10ms using timer.

Sample view:



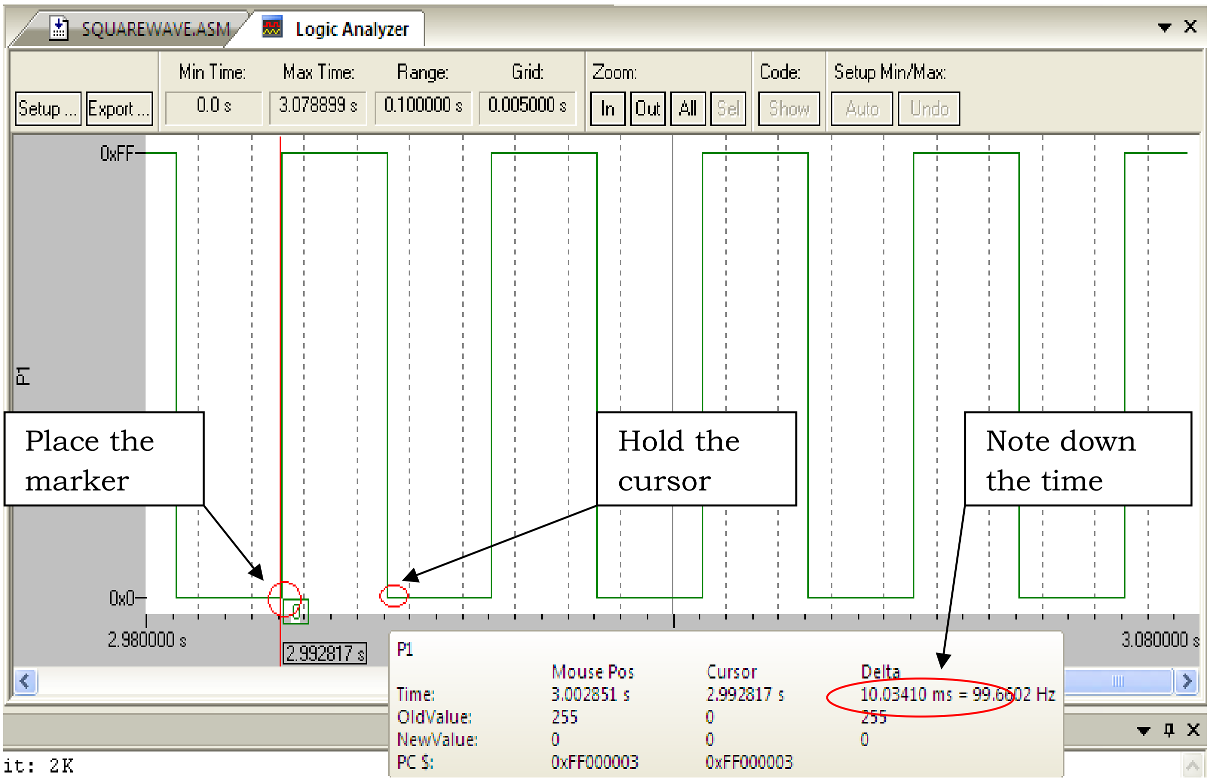


Fig 5: Screenshot of waveform in Logic analyzer window

Program no: 31

Objective: To generate the square wave with the on time delay of 6ms and off time delay of 4msec.Configure the timer in Timer0 mode1. Assume the crystal frequency of 24 MHz

```
            ORG 0000H
BACK:       MOV P1, #00H        ; generate OFF time through P1
            ACALL DELAY         ; Call 2ms delay subroutine twice to get 4ms
            ACALL DELAY
            MOV P1, #0FFH       ; generate ON time through P1
            ACALL DELAY         ; Call 2ms delay subroutine thrice to get 6ms
            ACALL DELAY
            ACALL DELAY
            SJMP BACK           ; repeat the processes forever
DELAY:      MOV TMOD, #01H      ; configure the timer0 in mode1
            MOV TL0, #060H      ; set the initial value in timer register for 2ms
            MOV TH0, #0F0H
            SETB TR0            ; start the timer
REPEAT:     JNB TF0, REPEAT     ; wait until timer overflows
            CLR TR0             ; halt the timer
            CLR TF0             ; clear the timer0 overflow interrupt
            RET                 ; ret to the main program
            END
```

Outcome Program no: 31

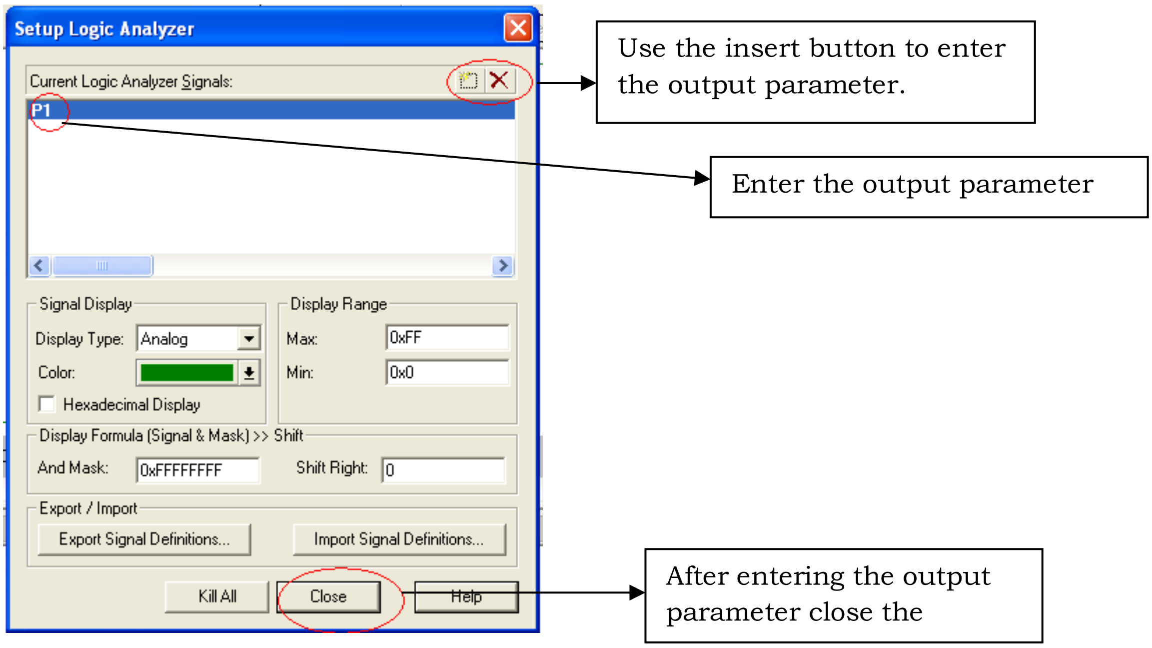
At the end of the program

Students will be able to program generating delays using timers.

Result

At the end of the execution, square wave is generated in P1 with 6msec on time and 4msec off time delay.

Sample view:



OFF Time measure:

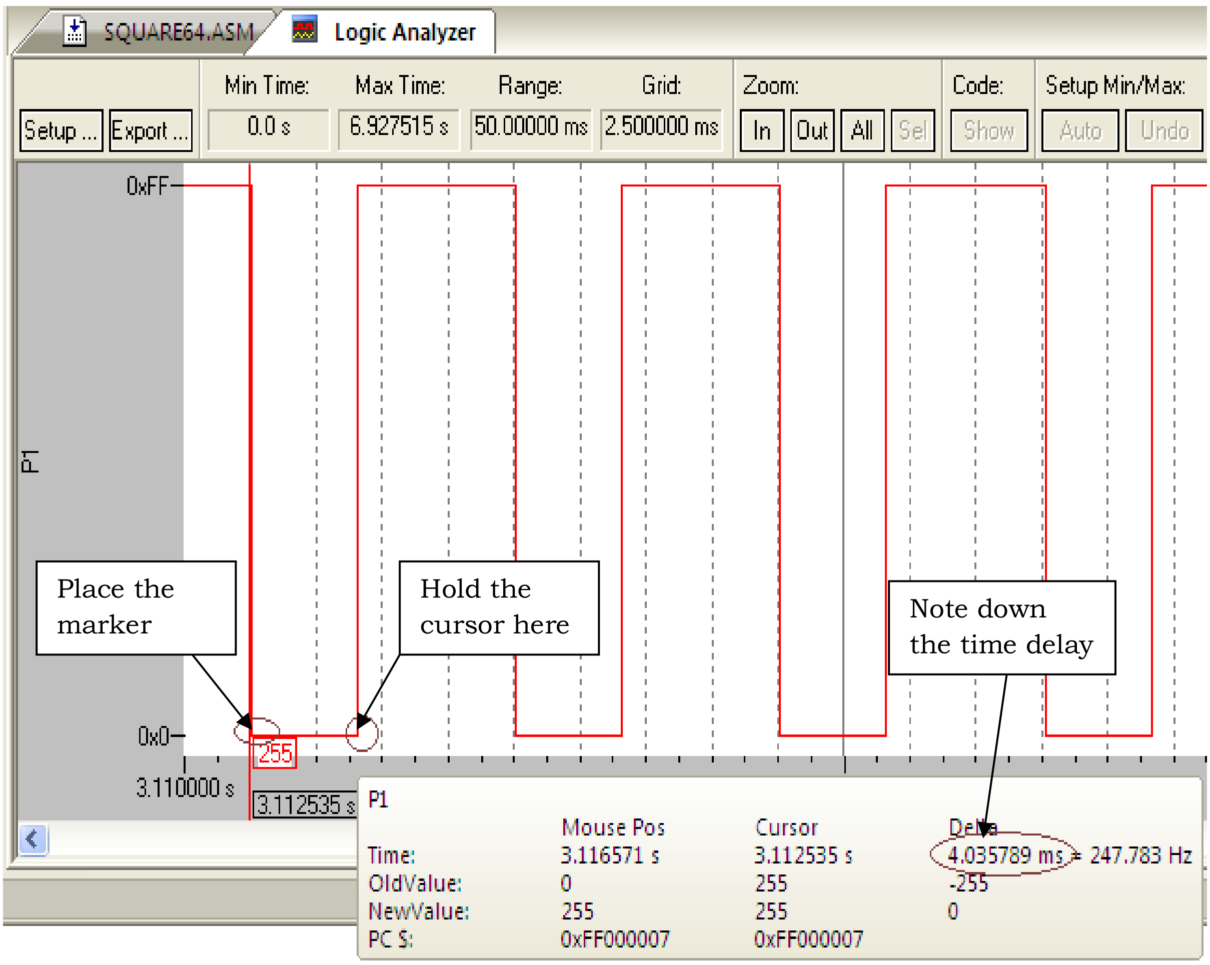


Fig 6: Screenshot of waveform in Logic analyzer window for OFF Time measure

ON Time measure:

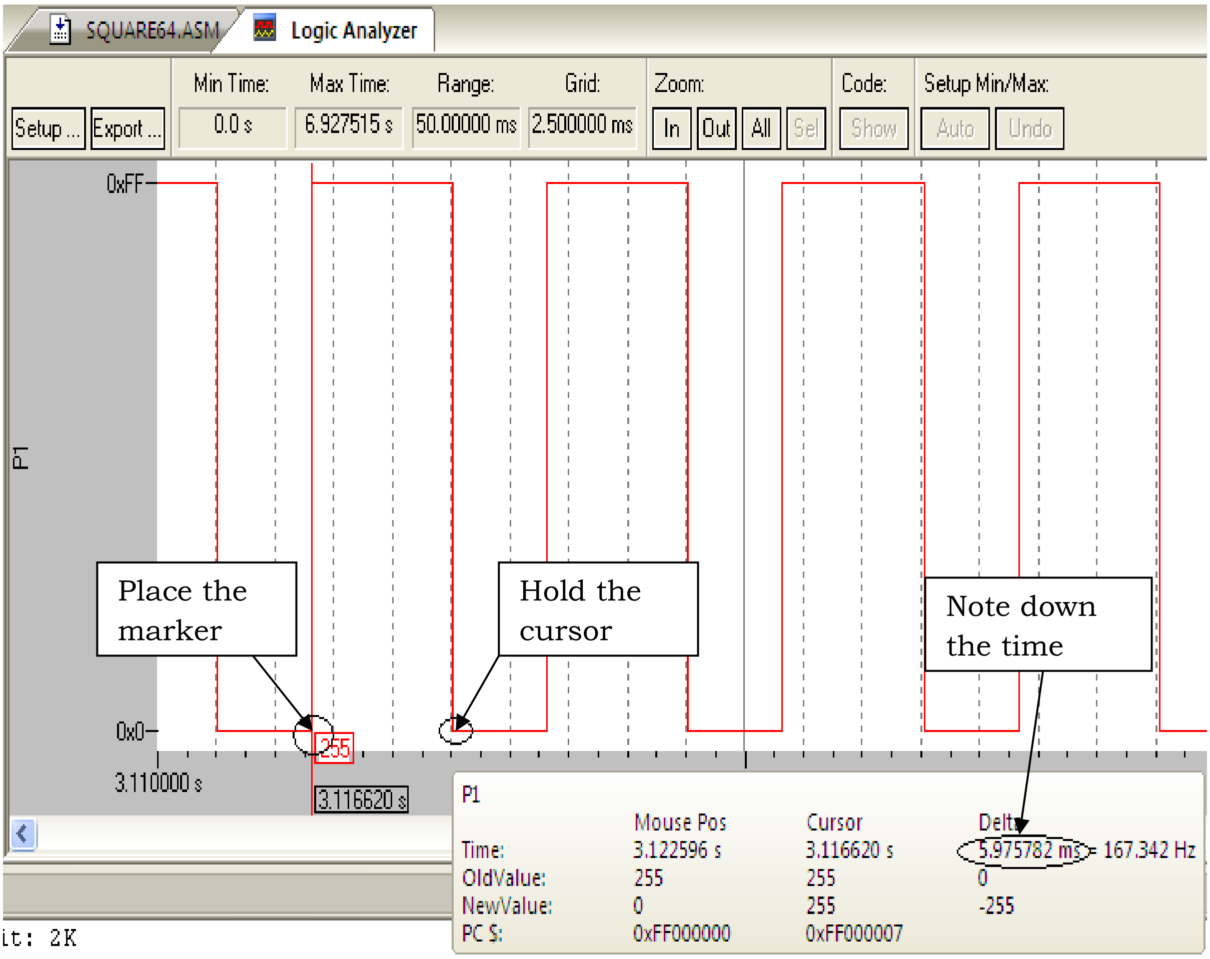


Fig 7: Screenshot of waveform in Logic analyzer window for ON Time measure

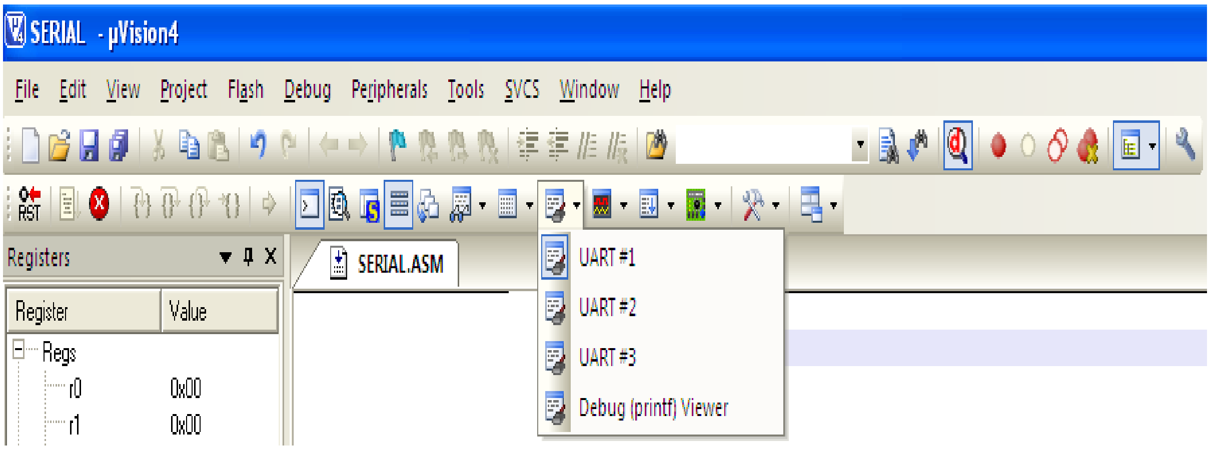
Program no: 32

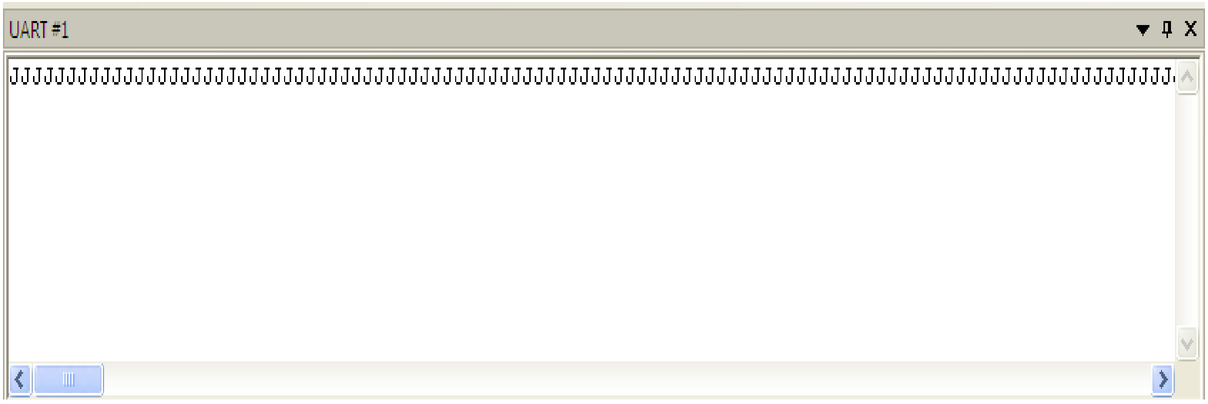
Objective: To send the letter ‘J’ serially using the UART at the baud rate of 9600. Configure SCON register in mode 1. Assume the crystal frequency of 11.0592 MHz

```
        ORG 0000H
BACK:   MOV TMOD, #20H      ; configure the timer1 in mode2
        MOV TH1, #-3        ; count for the baud rate of 9600
        MOV SCON, #50H      ; configure SCON to mode1
        SETB TR1            ; start the timer
        MOV SBUF, #'J'      ; send the letter ‘J’ through SBUF register
HERE:   JNB TI, HERE        ; wait until ‘J’ character is sent (8bits are transferred)
        CLR TI              ; clear serial interrupt for next character to be sent
        SJMP BACK           ; repeat the processes
        SJMP $
        END
```

Outcome:

Transmitted the letter ‘J’ serially using UART at the baud rate of 9600.





At the end of the program

Students will be able to write program for serial programming.

Result

At the end of the execution, letter 'J' is transmitted serially using the UART at the baud rate of 9600.

Program no: 33

Objective: To find the GCD and LCM of the given two numbers, which are placed in the external memory location 9400h and 9401h. The GCD of the two given numbers as to be stored in external memory location 9402h and the LCM as to be stored in the external memory location 9403h and 9404h.

```
        ORG 0000H
        MOV DPTR,#9400H        ; get the source memory address
        MOVX A,@DPTR           ; get the first value to Accumulator
        MOV R1,A               ; store the first input value in R1 register
        MOV R3,A               ; store the first input value in R3 register
        INC DPTR               ; get the source memory+1 address
        MOVX A,@DPTR           ; get the second value to accumulator
        MOV R2, A              ; store the second input value to register R2
        MOV R4, A              ; store the second input value to register R4
AGAIN:  MOV A, R1              ; get the first input value back to accumulator
        CJNE A, 02H, CHECK     ; if input1! = input2 jump to label "CHECK"
        SJMP OVER              ; if two inputs are equal jump to label "OVER"
CHECK:  JNC GCD                ; if input1 > input2, jump to label "GCD"
        XCH A,R2               ; if input2 > input1, exchange both the inputs
        MOV R1,A
GCD:    CLR C                  ; clear carry flag
        SUBB A,R2        ; subtract first number from second number
        MOV R1,A               ; get the Outcome of subtraction to register R1
        SJMP AGAIN             ; go back to label "AGAIN"
OVER:   INC DPTR               ; get the Outcome memory address
        MOVX @DPTR, A          ; store the GCD of two input numbers in Outcome memory.
        MOV B, A               ; get the GCD output to B register
        MOV A, R3              ; get the first input to Accumulator
        DIV AB                 ; divide second input number by GCD value
        MOV B, R4              ; get the second number to register B
        MUL AB                 ; multiply second number with previous division's quotient
```

```
        INC DPTR
        INC DPTR            ;get the Outcome+2 memory address
        MOVX @DPTR,A        ; store the lower byte of LCM output to Outcome+2 memory
        MOV A,B             ; get the upper byte of LCM value to register A
        DEC DPL             ; get the Outcome+1 memory address
        MOVX @DPTR,A        ; store the upper byte of LCM output to Outcome+1 memory
        SJMP $
END
```

Outcome:

Before Execution

Address	**Data**
0x9400	0x05
0x9401	0x06

Address	**Data**
0x9402	0x00
0x9403	0x00
0x9404	0x00

After Execution

Address	**Data**
0x9400	0x05
0x9401	0x06

Address	**Data**
0x9402	0x01
0x9403	0x00
0x9404	0x1E

At the end of the program

Students will be able to program GCD and LCM of the given two numbers.

Result

At the end of the execution, GCD and LCM of the given two numbers is placed in the external memory location 9400h and 9401h. The result is stored in external memory location 9402h and the LCM as to be stored in the external memory location 9403h and 9404h.

Program no: 34

Objective: To find the factorial of the given number placed in the external memory location 8300h. The Outcome as to be stored in the memory location 8400h and 84001h

```
        ORG 0000H
        MOV DPTR, #8300H        ; get the input memory address
        MOVX A, @DPTR           ; get the input number to accumulator
        MOV B, #00H             ; clear register B
        CJNE A, #00H, NEXT      ; if input number is! = 00 jump to label "NEXT"
        MOV A, #01H             ; if input number is = 00 store factorial as 01 in accumulator
        SJMP L2                 ; jump to label "L2"
NEXT:   CJNE A, #01H, FACTO     ; if input number is! = 01 jump to label "FACTO"
        SJMP L2                 ; jump to label "L2"
FACTO:  MOV R1, #01H            ; Initialize register R1 with 01
        MOV R2, #01H            ; Initialize register R2 with 01
        MOV R0, A               ; copy the input data to register R0
REPEAT: MOV A, R2               ; get the R2 register content to accumulator
        INC R1                  ; increment the register R1 content
        MOV B, R1               ; get the R1 register content to register B
        MUL AB                  ; multiply the accumulator and B register content
        MOV R2, A               ; store the lower byte of Outcome to register R2
        MOV A, R0               ; get the input number to accumulator
        CJNE A, 01H, REPEAT     ; if input number! = register R1 content, jump to "REPEAT"
        MOV A, R2               ; if equal, get lower byte of factorial output to accumulator
L2:     MOV DPTR, #8401H        ; get the Outcome+1 memory address
        MOVX @DPTR, A           ; store the lower byte of Outcome in Outcome+1 memory
        DEC DPL                 ; get the Outcome memory address
        MOV A, B                ; get the upper byte of factorial Outcome to accumulator
        MOVX @DPTR, A           ; store the upper byte of Outcome in Outcome memory
        SJMP $
        END
```

Outcome Program no: 34

Before execution

Address	Data
0x8300	0x06

Address	Data
0x8400	0x00
0x8401	0x00

After execution

Address	Data
0x8400	0x02
0x8401	0xD0

At the end of the program

Students will be able to program factorial of the given number

Result

At the end of the execution, factorial of the given number is placed in the external memory location 8300h. The Outcome is stored in the memory location 8400h and 84001h

HARDWARE PROGRAMS

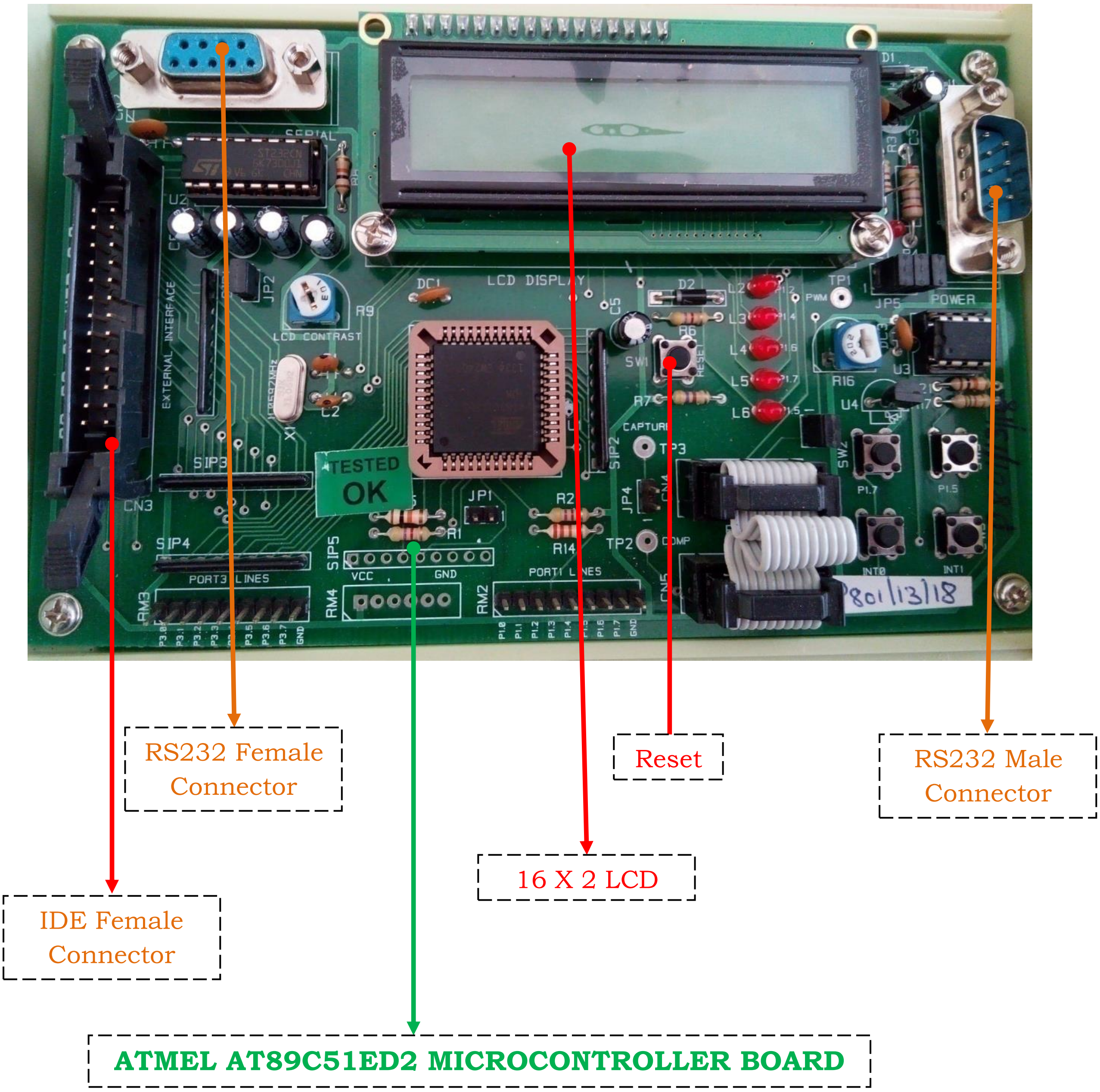


Fig 1..: AT89C51 Development Board

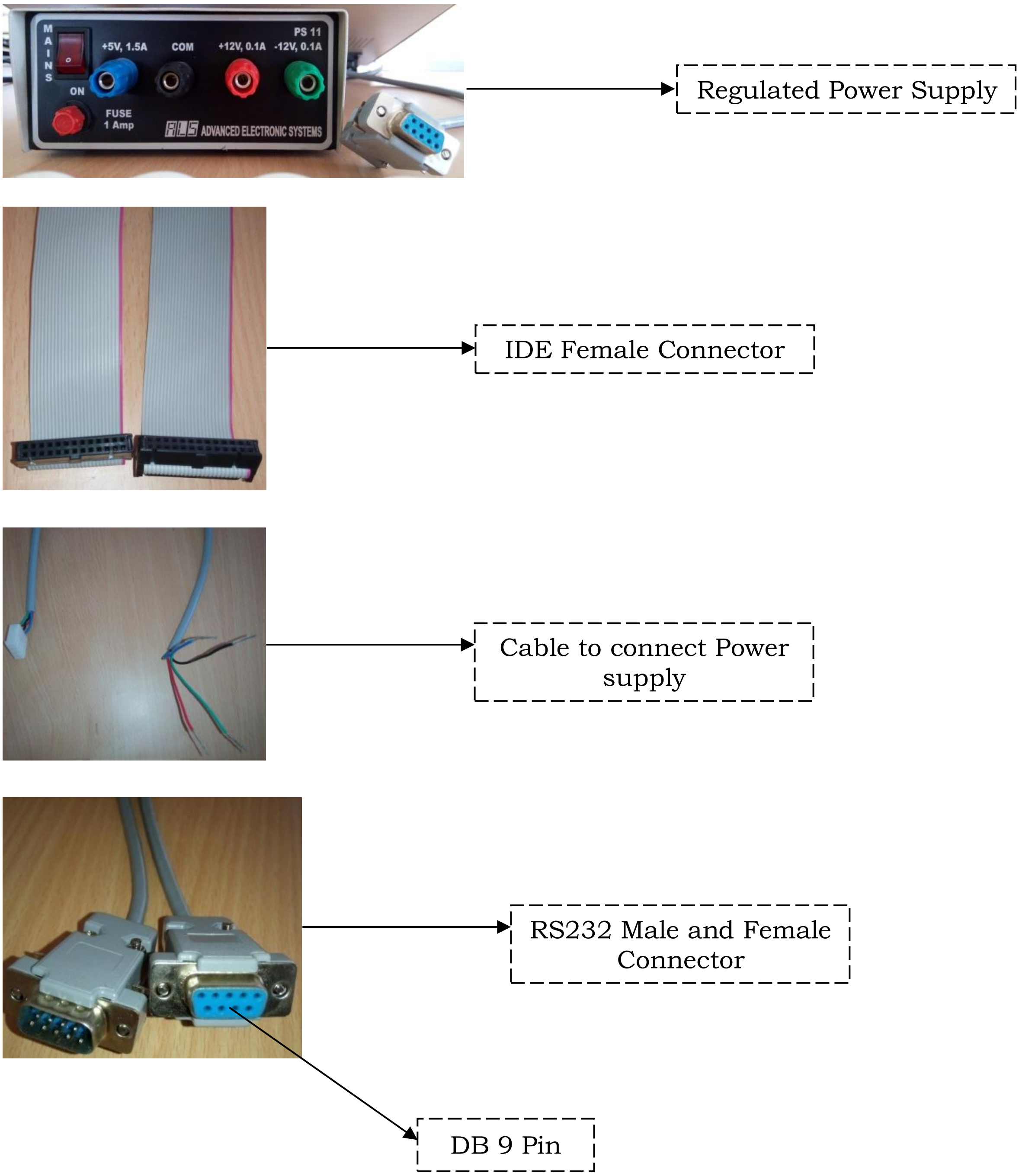


Fig 2..: Components for Interfacing

II. INTERFACING:

Hardware Programs

7. Generate different waveforms: Sine, Square, Triangular, Ramp using DAC interface.

Objective: Write a C program to generate square wave on Port1 and display the ramp wave in CRO using DAC interface.

Components: AT89C51ED2 Development board, DAC interface, RS 232 Cable, DC Power supply Vcc: +5V, 1.5A, Vdd:+/-12V,0.1A, CRO, probes

Block Diagram of DAC Interface:

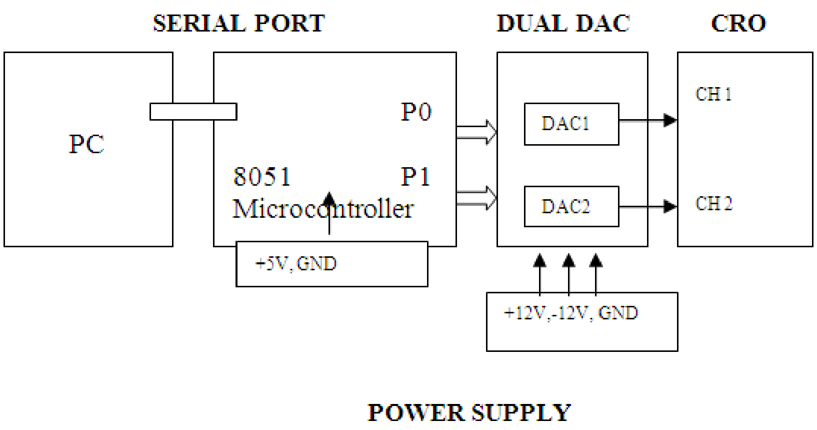


Fig 1.1.: Interfacing diagram of DAC

Program

```
#include <at89c51xd2.h>
void delay(void);
void main ()
{
        while(1)
        {
P1 = 0x0;       ;To get a square wavewith 0V as initial point, minimum 8 bit
                 bit value 0x0 is provided.
delay();                ;delay is provided to control thje frequency of the wave.
        P1 = 0xff;      ;To get a square wave 0f 5V, maximum 8 bit value FF
is provided.
```

```
            delay();
        }
}

void delay(void)
{
        int i;
        for(i=0;i<=300;i++);
}
```

1) To Generate a Square Wave:

- #include <Reg 51h> - This is to use the registers of 8051 microcontroller for programming.
- Reg 51.h is a reader file which contains all the registers of 8051 microcontroller.
- For getting a square wave of 5V (maximum output that can be obtained using the kit) we have to provide the maximum 8 bit number that is 0FFH and to get 0V we have to give 00H.
- So first give 00H as the digital input to DAC and then provide some delay. This delay is used to control the frequency of square wave.
- Then again provide FFH to get 5V output. The loop should be repeated continuously to get a square waveform.

Delay Function:

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

Void means the function does not return any delay is the function name and the parameter passed to the function is of integer data type (that is it can hold 16 bit data). So whatever value is passed to the delay function the variable '*x*' takes that value.

Therefore loop is defined without initialization. Then the x value is decremented until it becomes zero. So the delay can be obtained. For different x value we will get different delay.

In the main program

main ()

Unsigned character ON = 0 X 45, it means 'ON' is a variable of data type unsigned character (i.e., 8 bit) and is initialized with 0 X 45. Similarly for 'OFF'

P0=0 X 00; This is to configure P0 as output port. To configure as output port 00 should be given and to configure as input port FF should be given to the corresponding port special function register.

while (1): - This statement is used to repeat the loop infinite times. So that we will get a continuous waveform.

Then give the value required for ON and OFF condition

For 0V – 00H

5V – FFH

Therefore for getting 1V at the output the digital value should be $\frac{FF}{5}H$.

Then for 2V - $\frac{FF}{5}\ X\ 2_H$ and so n.

Similarly for changing the frequency, change the value that is passed to the delay function.

Delay (1) if we are giving and we will get the wave shown in figure 1.1b

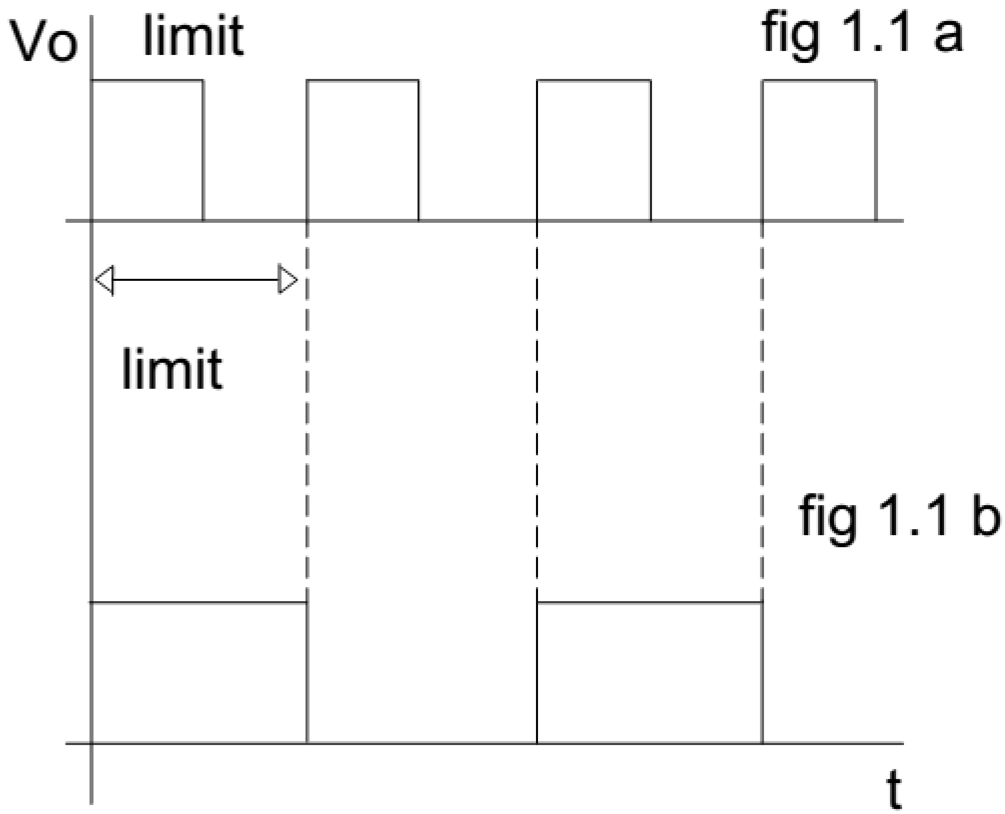


Fig 1.2.: Waveform of Square wave

Outcome

At the end of the program

The exercise shall make the students competent in using DAC interface to 8051 and change the frequency and amplitude

.

Result

At the end of the execution, C program to generate square wave on Port1 is written and the waveform is displayed in CRO using DAC interface.

Objective: Write a C program to generate triangular wave on Port1 and display the triangular wave in CRO using DAC interface.
Components: AT89C51ED2 Development board, DAC interface, RS 232 Cable, DC Power supply Vcc: +5V,1.5A,Vdd:+/-12V,0.1A, CRO, probes

Program

```
#include <at89c51xd2.h>
idata unsigned char count;  //unsigned char-→8 bit data type   count
void main ( )
{
 while(1)
        {
         for(count=0;count!=0xff;count++)
             {
                P1=count;
              }
               for(count=0xff; count>0;count--)
                {
                   P1=count;
                }
            }
}
```

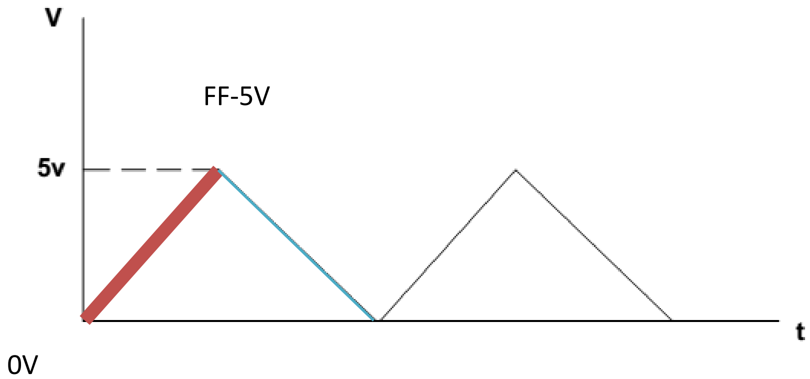


Fig 1.2.: Waveform of triangular wave

2) To generate Triangular waveform:

To get a triangular waveform, the variable value is invreased from 00H to the required amplitude value (max FFH) and after reaching the value the variable is decremented continuously to 00H. So two for loops are used for getting a triangular wave. By changing the delay function value the slope of triangular wave form can be controlled.

Outcome

At the end of the program

The exercise shall make the students competent in using DAC interface to 8051 and change the frequency and amplitude

.

Result

At the end of the execution, C program to generate triangular wave on Port1 is written and the waveform is displayed in CRO using DAC interface.

Objective: Write a C program to generate Sine wave on Port1 and display the Sine wave in CRO using DAC interface.

Components: AT89C51ED2 Development board, DAC interface, RS 232 Cable, DC Power supply Vcc: +5V,1.5A,Vdd:+/-12V,0.1A, CRO, probes

Program

```
#include <at89c51xd2.h>
xdata unsigned char sine_tab[49]={
0x80,0x90,0xA1,0xB1,0xC0,0xCD,0xDA,0xE5,0xEE,0xF6,0xFB,0xFE,0xFF,0xFE,0
xFB,0xF6,0xEE,0xE5,0xDA,0xCD,0xC0,0xB1,0xA1,0x90,0x80,0x70,0x5F,0x4F,0x
40,0x33,0x26,0x1B,0x12,0x0A,0x05,0x02,0x00,0x02,0x05,0x0A,0x12,0x1B,0x26,0x
33,0x40,0x4F,0x5F,0x70,0x80};
//  V=128+128sinθ
idataint count;
void main ()
{

  while (1)
        {
            for(count=0;count<49;count++)
            {
                P1 = sine_tab [count];
            }
        }
 }
```

Calculation:

128+128 sin θ

θ = 0, 128+128 sin0 = 0 x 80

θ = 7.5 = 128+128 sin7.5 = 0 x 90

Take θ value 7.5 and calculate for the 49 hex values in the program

3) To generate Sine wave:

The equation for Sine wave is $V_0 = 128(1 + \sin\theta)$

$$V_0 = 128 + 128\sin\theta$$

By giving different values for θ, different amplitude of the Sine wave can be obtained.

$$\text{When } \theta = 0, V_0 = 128 \text{ because } \sin 0 = 0$$

$$\text{When } \theta = 7.5, V_0 = 128 + 128\sin 7.5$$

$$\text{When } \theta = 350, V_0 = 128 + 128\sin 350,$$

So θ value is increased from 0^0 to 350^0 and the corresponding output voltages are arranged in an array.

The each value is given to port 0 to get the sine wave at the output.

To get 5V Sine wave $V_0 = 128 + 128\sin\theta$

To get 2.5V Sine wave $V_0 = 128 + 128\sin\theta$ and so on.

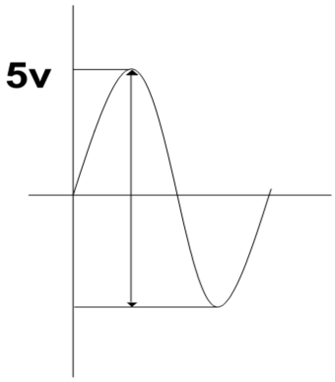


Fig 1.3.: Waveform of Sine wave

Outcome

At the end of the program

The exercise shall make the students competent in using DAC interface to 8051 and change the frequency and amplitude

.

Result

At the end of the execution, C program to generate sine wave on Port1 is written and the waveform is displayed in CRO using DAC interface.

Objective: Write a C program to generate ramp wave on Port1 and display the Ramp wave in CRO using DAC interface.

Components: AT89C51ED2 Development board, DAC interface, RS 232 Cable, DC Power supply Vcc: +5V,1.5A,Vdd:+/-12V,0.1A, CRO, probes

Program

```
#include <at89c51xd2.h>
idata unsigned char count;
void main ()
{
      count = 0x0;
     while(1)
     {
          P1 = count;
          count++;
     }
}
```

00H---------------------------------FFH

0V-------------------------------------5V

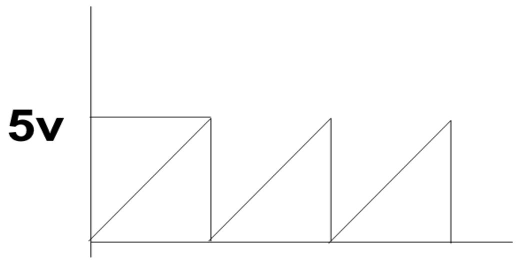


Fig 1.4.: Waveform of Sine wave

4) To Generate Ramp wave:

To get a ramp waveform at the output, a variable is increased from 00H to FFH and then after ready FFH, then again the variable value is increased from 00H to FFH. This loop repeats continuously to generate Ramp waveform.

So inside for loop the value of variable is increased from 00H to FFH.

By changing the delay function value the slope of the Ramp waveform (frequency) can be controlled.

For 00H – 0V

FFH – 5V

7FH – 2.5V etc.

Outcome

At the end of the program

The exercise shall make the students competent in using DAC interface to 8051 and change the frequency and amplitude

Result

At the end of the execution, C program to generate ramp wave on Port1 is written and the waveform is displayed in CRO using DAC interface.

8. DC Motor Interface to 8051

Objective: Write C program to interface DC motor to AT89C51ED2 µC to control the speed of DC motor with different duty cycle.

Components: AT89C51ED2 Development board, DC Motor interface, RS 232 Cable,

DC Power supply: 5V

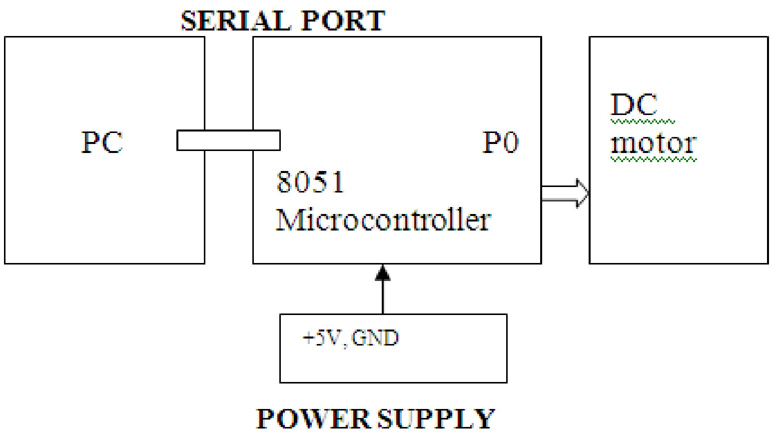
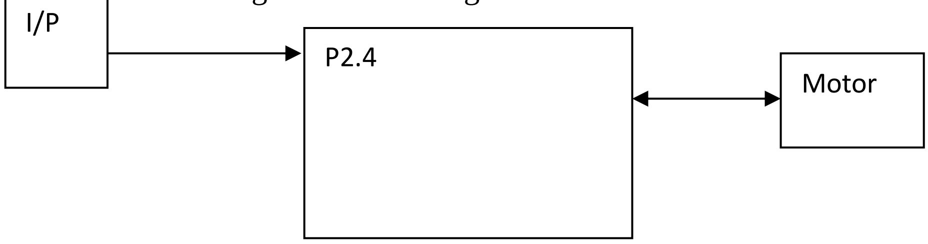


Fig 1.5.: Block diagram DC Motor



Program

```
#include <at89c51xd2.h>
//  off time : variable to hold value for 30 milliseconds
//  on time: variable to hold value for 10 milliseconds
sbit P24= P2^4;   Port 2 bit 4  , Input
idata unsigned char off_time,on_time;
idata unsigned char ii;
void main ()
{
            TCON = 0;
            TMOD = 0x01;                        //select mode 1, timer 0
           off_time = 30;
           on_time = 10;
           while(1)
```

```
        {
            P24 = 1;                    // make P2.4 high
            for(ii=0;ii<on_time;ii++)
            {
                TL0 = 0x66;   //timer count set
                TH0 = 0xFC;  // load timer high and low registers
                TR0 =1;                  // start timer 0
             // each time the timer overflfow occurs at 1 milli second
                while(!TF0)  // till timer does not overflow
    {
                TF0 = 0;          // reset timeroverflow flag
                TR0=0;            // stop timer 0
            }
            P24 = 0;   // reset P2.4
            for(ii=0;ii<off_time;ii++)
            {
                TL0 = 0x66;   //timer count set for
                TH0 = 0xFC;  // load timer high and low registers
                TR0 =1;                  // start timer 0
                while(!TF0)
                {
                }
                TF0 = 0;          // reset timer overflow flag
                TR0=0;                    // stop timer 0
            }
        }
}
```

Outcome

1. The exercise shall make the students competent in utilising DC motor for various applications

Result

At the end of the execution, C program to interface DC motor to AT89C51ED2 µC to control the speed of DC motor with different duty cycle is performed.

9. Stepper Motor Interface to 8051

Objective: Write C program to rotate stepper motor clockwise.

Components: AT89C51ED2 Development board, Stepper Motor interface, RS 232 Cable, DC Power Supply: 5V

Program

Stepper Motor Clockwise.

```
#include <at89c51xd2.h>
Void delay (void);
 Void main (void)
{
while(1)
{
        P2=0x07;  // output 0x07 to port P2
        delay();  // generate delay

        P2=0x0b;  // output 0x0b to port P2
        delay();  // generate delay

        P2=0x0d; // output 0x0d to port P2
        delay(); // generate delay
        P2=0x0e;  // output 0x0e to port P2
        delay();  // generate delay
}
}
void delay(void)
        {
        int i;
        for (i=0;i<=30000;i++);
         }
```

Rotate Stepper Motor Anticlockwise

```
#include <at89c51xd2.h>
void delay(void);
void main(void)
{
while(1)
{
    P2=0x0e;  // output 0x0e to portP2
    delay();  // generate delay

    P2=0x0d; // output 0x0d to port P2
delay();  // generate delay

     P2=0x0b;   // output 0x0b to portP2
     delay();  // generate delay

     P2=0x07; // output 0x07 to portP2
     delay();  // generate delay

}
}
void delay(void)
{
        int i;
        for(i=0;i<=30000;i++);
 }
```

Objective: Write C program to rotate stepper motor N rotation clockwise. (Where N=1, 2, 3…n).

Components: AT89C51ED2 Development board, Stepper Motor interface, RS 232 Cable, DC Power Supply: 5V

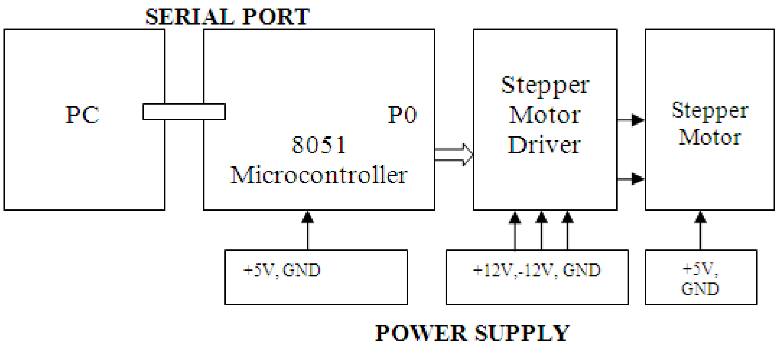


Fig 1.6.: Block diagram StepperMotor

Program

```
#include <at89c51xd2.h>
void delay(void);
void main()
{
unsigned char i, j, k=0;
int value [] = {0x0b, 0x07, 0x0e, 0x0d};          // for anticlockwise {    ,    ,    ,    ,}
unsigned char count1 = 3;                          //N=3
unsigned char count = 200;
for (j=0;j<count1;j++)//count number of rotations
{
for(i=0;i<count; i++)//count for number of steps
{
              P2=value[k];
             k=k+1;
             if(k>3)
             k=0;
             delay();
             }
             }
             while(1);
```

```
        }
void delay(void)
{
unsigned int i;
for(i=0;i<10000;i++);
}
```

Outcome:

At the end of the program, Students will be able to analyze interfacing of stepper motor.

Result

At the end of the execution, C program to interface Stepper motor to AT89C51ED2 µC is performed and step control is observed.

Hobby Project Circuit:

https://www.electronicshub.org/interfacing-dc-motor-8051-microcontroller/

https://www.electronicshub.org/automatic-railway-gate-controller/

10. Alphanumeric LCD Interface

Alphanumeric LCD panel and Hex keypad input interface to 8051

Objective: Write a 8051 C Program to send ‘A’, ‘T’, ‘M’, ‘E’, ‘ ’, ‘M’, ‘Y’, ‘S’, ‘O’, ‘R’, ‘E’, to LCD display.

Components: AT89C51ED2 Development board, LCD panel interface, RS 232 Cable, DC Power Supply: +5V

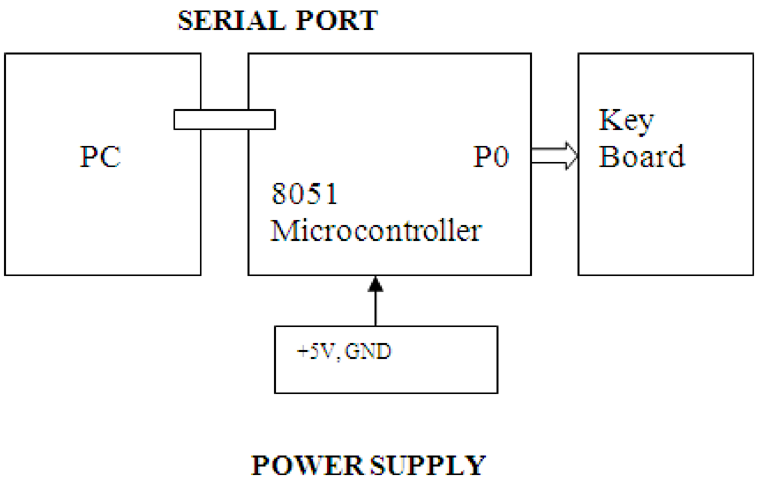


Fig 1.6.: Block diagram LCD and Keypad interface

Program

```
#include <at89c51xd2.h>
sfr ldata = 0x80;
sbit rs=P2^4;
sbit rw=P2^5;
sbit en=P2^6;

void lcddata(unsigned char value);
void lcdcmd(unsigned char value);
void MSDelay( unsigned int itime);
void main()
{
        lcdcmd(0x38);       5X7   matrix
        MSDelay(250);
        lcdcmd(0x0e);       Display on, cursor blinking
        MSDelay(250);
        lcdcmd(0x01);       Clear display screen
        MSDelay(250);
        lcdcmd(0x06);         Increment cursor (shift cursor to right)
        MSDelay(250);
```

```
        lcdcmd(0x80);  //Force cursor to the beginning of first line
        MSDelay(250);
        lcddata('A');
        MSDelay(250);
        lcddata('T');
        MSDelay(250);
        lcddata('M');
        MSDelay(250);
    lcddata('E');
        MSDelay(250);
    lcddata(' ');
        MSDelay(250);
        lcddata('M');
        MSDelay(250);
        lcddata('Y');
        MSDelay(250);
    lcddata('S');
        lcdcmd(0xC0);
        MSDelay(250);
        lcddata('O');
        MSDelay(250);
        lcddata('R');
        MSDelay(250);
     lcddata('E');
  here:  goto here;
}
      void lcdcmd(unsigned char value)
      {
        ldata = value;
            rs=0;
            rw=0;
            en=1;
        MSDelay(1);
            en=0;
            return;
      }

      void lcddata(unsigned char value)
      {
        ldata = value;
            rs=1;
            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++);
}
```

Outcome:

The above exercise shall make the students competent in using LCD for various applications.

Result

At the end of the execution, C program is written for LCD interfacing and the Characters are observed.

Hobby Project circuit:

https://www.electronicshub.org/interfacing16x2-lcd-with-pic-microcontroller/
https://www.electronicshub.org/interfacing-16x2-lcd-avr-microcontroller/

Objective: Write an 8051 C Program to interface HEX Keypad to AT89C51ED2 µC to display the key pressed.

Components: AT89C51ED2 Development board, HEX Keypad interface, RS 232 Cable, DC Power Supply: 5V

Program

```
#include <at89c51xd2.h>
void lcd_init(void);
void clr_disp(void);
void lcd_com(unsigned char );
void lcd_data(unsigned char );
void scan(void);
void get_key(void);
void display(void);
void delay(char);
idata unsigned char  row,col,key;

code unsigned char scan_code[16]={  0xEE,   0xDE,0xBE,0x7E,
0xED,  0xDD, 0xBD,  0x7D,
0xEB,  0xDB,  0xBB,  0x7B,
0xE7,   0xD7,  0xB7,   0x77};

code unsigned char ASCII_CODE[16]= {'0','4','8','C',
                                    '1','5','9','D',
                                    '2','6','A','E',
                                    '3','7','B','F'};
idata unsigned char temp,temp2,temp3,res1,flag, Outcome;
sbit      enable=P2^6;
sbit      rw=P2^5;
sbit      rs=P2^4;
void main ()
{
        lcd_init();
```

```
        delay(5);
        P2=0x0f;

        while(1)
        {
                get_key();
                display();
                delay(100);
        }

}  //end of main()
void get_key(void)
{
        unsigned char  i;

        flag = 0x00;
        while(1)
        {
                for(row=0;row<4;row++)             //check for row depending  on bit
                                                   //assign value to temp3
                {
                   if( row == 0)
                        temp3=0xfe;
                   else if(row == 1)
                        temp3=0xfd;
                   else if(row == 2)
                        temp3=0xfb;
                   else if(row == 3)
                        temp3=0xf7;
                        P1 = temp3;
                        scan();
                        delay(10);
                        if(flag == 0xff)
                        break;
```

```
        } // end of for

        if(flag == 0xff)
        break;
    }  // end of while
        for(i=0;i<16;i++)
          {
            if(scan_code[i] == res1)            //equate the scan_code with res1
                {
                Outcome =  ASCII_CODE[i];        //same position value of
ascii code
                break;
                }

            }
}
void scan(void)
{
    unsigned char t;
temp2 = P2;
    temp2 = temp2 & 0x0f;          //read port2 ,mask with 0x0fh
    if(temp2 != 0x0f)              //is any change in temp2
    {
        delay(30);            //give debounce delay check again
        delay(30);
        temp2 = P2;
          temp2 = temp2 & 0x0f;
            do
            {
                flag = 0xff;
                res1 = temp2; // store the value in res1
                t = (temp3 << 4)  & 0xf0;
                res1 = res1 | t;
                temp2 = P2;
```

```
                temp2 = temp2 & 0x0f;
            }
    while(temp2 != 0x0f);
        }
        else
        {
            flag = 0x00;
        }

} // end of scan()

void display(void)
{
        lcd_com(0x80);      //display  address for key value
    delay(5);
    lcd_data(Outcome);
    delay(5);
}
void lcd_init(void)
{
        lcd_com(0x38);     //display value for count
    delay(5);

    lcd_com(0x38);
    delay(5);

        lcd_com(0x0f);      // display on ; cursor on
    delay(5);

    lcd_com(0x06);      // shift cursor right
    delay(5);

    clr_disp();
}
```

```
void clr_disp(void)
{
        lcd_com(0x01);
        delay(5);
}
void lcd_com(unsigned char temp )
{
P0 = temp;
rs = 0;
        rw = 0;
  enable=1;
        delay(5);
        enable=0;
 }
void lcd_data(unsigned char temp)
{
        P0 = temp;
rs=1;
        rw=0;
   enable = 1;
   delay(5);
   enable = 0;
}
void delay(char r)
{
        int r1;
        for(r1=0;r1<r;r++);
}
```

Outcome:

The above exercise shall make the students competent in using HEX Keypad interface for various applications.

Result

At the end of the execution, C program is written for Hex Keypad interface and the typed Characters are observed.

Content beyond Syllabus

Conditional CALL, Subroutine, Return instructions

Program no: 1

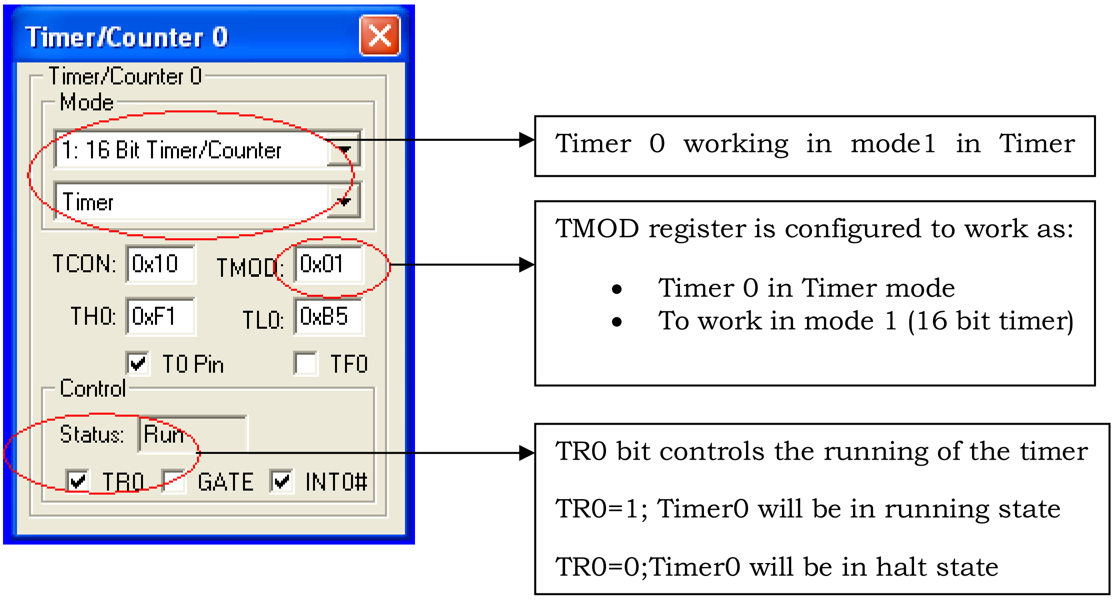
Objective: To display hexadecimal up/down count (00h to FFh and FFh to 00h) continuously in Port1. The delay between two counts should be 1 second. Configure TMOD register in Timer0 Mode1 configuration.

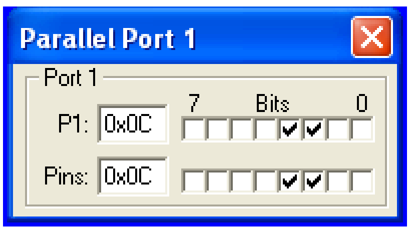
```
        ORG 0000H
        MOV A,#00H          ; get the first BCD value to accumulator
L1:     MOV P1,A            ; display the count in P1
        INC A               ; increment the count
        LCALL DELAY         ; call the delay of 1sec
        CJNE A,#0FFH,L1     ; check count has reached FFh, if not continue up count
L2:     MOV P1,A            ; display the count in P1
        LCALL DELAY         ; call the delay of 1sec
        DEC A               ; decrement the count
        CJNE A,#00H,L2      ; check count has reached 00h, if not continue down count
        SJMP L1             ; repeat forever

DELAY:  MOV TMOD,#01H       ; configure timer0 in mode1
        MOV R0,#1FH         ; get the count for repetition of timer register count
BACK:   MOV TL0,#00H        ; set the initial count for 1sec
        MOV TH0,#00H
        SETB TR0            ; start the timer
REPEAT: JNB TF0, REPEAT     ; wait until timer overflows
        CLR TR0             ; halt the timer
        CLR TF0             ; clear the timer0 overflow interrupt
        DJNZ R0, BACK       ; if repetition count!= 0, go to label back
        RET                 ; return to the main program
        END
```

Outcome Program no: 1

Observe the hexadecimal up/down count operation in Port1.

Sample view:





At the end of the program

1. Students will be able to understand the way in which subroutines are called and returns made in counters.
2. Analyze the calls and subroutines made in the program

Result

At the end of the execution, hexadecimal up/down count (00h to FFh and FFh to 00h) is displayed continuously in Port1.

Elevator Interface to 8051.

Objective: Write a C program to understand the functioning of an elevator.

Components: AT89C51ED2 Development board, elevator interface, RS 232 Cable, DC Power Supply: 5V

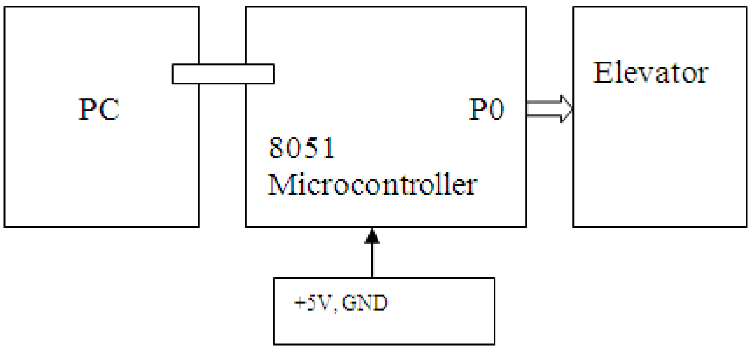


Fig 1.7.: Block diagram elevator interface

Program

```
#include <at89c51xd2.h>
void delay(unsigned int);
main()
{
 unsigned char xdataFlr[9] =  {0xff,0x00,0x03,0xff,0x06,0xff,0xff,0xff,0x09};
 unsigned char xdataFClr[9] = {0xff,0xE0,0xD3,0xff,0xB6,0xff,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;
     ReqFlr = ~ReqFlr;
     if(CurFlr == ReqFlr)
```

```
        {
            P0 = FClr[CurFlr];
         }
    else if(CurFlr>ReqFlr)
            {
        i = Flr[CurFlr] - Flr[ReqFlr];
                    j = Flr[CurFlr];
                  for(;i>0;i--)
                   {
              P0 = 0x0f0|j;
              j--;
             delay(50000);
            }
       }
         else
         {
             i = Flr[ReqFlr] - Flr[CurFlr];
                    j = Flr[CurFlr];
                     for(;i>0;i--)
              {
              P0 = 0x0f0 | j;
             j++;
              delay(50000);
              }
          }
CurFlr = ReqFlr;
            P0 = FClr [CurFlr];
   }
}

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

Outcome:

At the end of the program

Students will be able to understand the functioning of an elevator.

Result

At the end of the execution, C program is written for interface elevator to 8051 microcontroller and the result are observed.

External ADC and Temperature Control Interface.

Objective: Write a C Program to interface temperature sensor.

BLOCK DIAGRAM

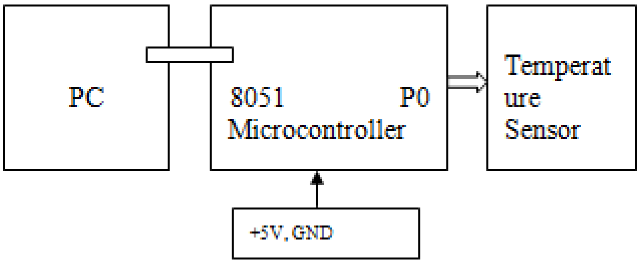


Fig 1.8.: Block diagram of Temperature Interface to 8051

PROGRAM

```
#include < reg51Xd2.h>
  sbit cmpout = P3^4;
 sbit rel_on = P0^0;
#define dac_data P1
   void delay (  )
   {
            Int l;
            for ( l=0; l < 10 ; l ++ ) ;

   }

void main (  )

   {

            unsigned char dacip;

            unpout = '1 ';

            dac_data = 0X00 ;

            P0 = 0X00 ;

            while ( 1 )

               {
```

```
            dacip = 0Xff;

            d0;

              {

                     dacip ++ ;

                     Dac_data = dacip;

                     delay ( );

              }

      while ( cmpout );

            If ( dacip > 0X20 )

                     Rel_on = 1;

            else

                     rel_on = 0;

            }

      }
```

Outcome:

At the end of the program

The students will be able to interface temperature sensor and analyze its output

Result

At the end of the execution, C program is written for temperature sensor and the result are observed.

Hobby Project Circuit:
https://www.electronicshub.org/temperature-controlled-dc-fan-using-microcontroller/
https://www.electronicshub.org/digital-temperature-sensor-circuit/

PROGRAM FOR BLOCK MOVE USING EXCHANGE MNEMONIC

SIZE OF BLOCK = 05
SOURCE DATA: FROM 40H TO 44H INTERNAL RAM (data view)
DESTINATION: FROM 50H TO 54H INTERNAL RAM (data view)

```
        Org 0000h
        mov r3, #05h            ; r3 is counter
        mov r0, #40h            ; r0 is source pointer
        mov r1, #50h            ; r1 is destination pointer
back:   mov a, @r0              ; a ← [[r0]]
        xch a, @r1              ; swap a & [[r1]]
        mov @r0, a              ; [[r0]]←a
        inc r0                  ; increment r0 & r1 to point
        inc r1                    next memory location
        djnz r3, back           ; jump on not zero to back
        end
```

PROGRAM TO SEARCH ELEMENT IN THE ARRAY OF BYTES IN EXTERNAL RAM

VALUES ARE STORED IN EXTERNAL RAM USING XDATA VIEW STARTING FROM 9000H ML.

```
        org 8000h
        mov r0,#03h             ; array size
        mov r1,#10h             ; array element
        mov r2,#00h             ; counter to know search element
        mov dptr ,#9000h
loop:   movx a,@dptr
        clr c
        subb a,r1               ; to check element is present or not
        inc dptr
        jnz skip
        inc r2
skip:   djnz r0,loop
        end
```

PROGRAM TO FIND SQUARE & CUBE OF A NUMBER

SQUARE:

```
Org 0000h
mov r0, #06h                    ; r0 = 06h
mov a, r0                       ; a = 06
mov 0f0h, r0                    ; b = 06
mul ab                          ; a = 24h, b=00h
end
```

CUBE:

```
org 0000h
mov r0, #0ah                    ; r0 = #10h
mov a, r0                       ; a = 10h
mov 0f0h, r0                    ; b = 10h
mul ab                          ; a = 64h, b=00
mov 0f0h, r0                    ; b = 10h
mul ab                          ; a = e8, b=03
end
```

Viva Questions

1. What do you mean by Embedded System? Give examples.

2. Why are embedded Systems useful?

3. What are the segments of Embedded System?

4. What is Embedded Controller? 5. What is Microcontroller?

6. List out the differences between Microcontroller and Microprocessor.

7. How are Microcontrollers more suitable than Microprocessor for Real Time Applications?

8. What are the General Features of Microcontroller? 9. Explain briefly the classification of Microcontroller. 10. Explain briefly the Embedded Tools. 11. Explain the general features of 8051 Microcontroller.

12. How many pin the 8051 has? 13. Differentiate between Program Memory and Data Memory.

14. What is the size of the Program and Data memory?

15. Write a note on internal RAM. What is the necessity of register banks? Explain.

16. How many address lines are required to address 4K of memory? Show the necessary calculations.

17. What is the function of accumulator?

18. What are SFR's? Explain briefly.

19. What is the program counter? What is its use?

20. What is the size of the PC?

21. What is a stack pointer (SP)?

22. What is the size of SP?

23. What is the PSW? And briefly describe the function of its fields.

24. What is the difference between PC and DPTR?

25. What is the difference between PC and SP?

26. What is ALE? Explain the functions of the ALE in 8051.

27. Describe the 8051 oscillator and clock.

28. What are the disadvantages of the ceramic resonator?

29. What is the function of the capacitors in the oscillator circuit?
30. Show with an example, how the time taken to execute an instruction can be calculated.
31. What is the Data Pointer register? What is its use in the 8051?
32. Explain how the 8051 implement the Harvard Architecture?
33. Explain briefly the difference between the Von Neumann and the Harvard Architecture.
34. Describe in detail how the register banks are organized.
35. What are the bit addressable registers and what is the need?
36. What is the need for the general purpose RAM area?
37. Write a note on the Stack and the Stack Pointer.
38. Why should the stack be placed high in internal RAM?
39. Explain briefly how internal and external ROM gets accessed.
40. What are the different addressing modes supported by 8051 Microcontroller ?
41. Explain the Immediate Addressing Mode.
42. Explain the Register Addressing Mode.
43. Explain the Direct Addressing Mode.
44. Explain the Indirect Addressing Mode.
45. Explain the Code Addressing Mode.
46. Explain in detail the Functional Classification of 8051 Instruction set
47. What are the instructions used to operate stack?
48. What are Accumulator specific transfer instructions?
49. What is the difference between INC and ADD instructions?
50. What is the difference between DEC and SUBB instructions?
51. What is the use of OV flag in MUL and DIV instructions?
52. What are single and two operand instructions?
53. Explain Unconditional and Conditional JMP and CALL instructions.
54. Explain the different types of RETURN instructions.
55. What is a software delay?

56. What are the factors to be considered while deciding a software delay?

57. What is a Machine cycle?

58. What is a State?

59. Explain the need for Hardware Timers and Counters?

60. Give a brief introduction on Timers/Counter.

61. What is the difference between Timer and Counter operation?

62. How many Timers are there in 8051?

63. What are the three functions of Timers?

64. What are the different modes of operation of timer/counter?

65. Give a brief introduction on the various Modes.

66. What is the count rate of timer operation?

67. What is the difference between mode 0 and mode 1?

68. What is the difference Modes 0,1,2 and 3?

69. How do you differentiate between Timers and Counters?

70. Explain the function of the TMOD register and its various fields?

71. How do you control the timer/counter operation?

72. What is the function of TF0/TF1 bit

73. Explain the function of the TCON register and its various fields?

74. Explain how the Timer/Counter Interrupts work.

75. Explain how the 8051 counts using Timers and Counters.

76. Explain Counting operation in detail in the 8051.

77. Explain why there is limit to the maximum external frequency that can be counted.

78. What’s the benefit of the auto-reload mode?

79. Write a short note on Serial and Parallel communication and highlight their advantages and disadvantages.

80. Explain Synchronous Serial Data Communication.

81. Explain Asynchronous Serial Data Communication.

82. Explain Simplex data transmission with examples.

83. Explain Half Duplex data transmission with examples.

84. Explain Full Duplex data transmission with examples.

85. What is Baud rate?

86. What is a Modem?

87. What are the various registers and pins in the 8051 required for Serial communication? Explain briefly. 88. Explain SCON register and the various fields.

89. Explain serial communication in general (synchronous and asynchronous). Also explain the use of the parity bit.

90. Explain the function of the PCON register during serial data communication.

91. How the Serial data interrupts are generated?

92. How is data transmitted serially in the 8051? Explain briefly.

93. How is data received serially in the 8051? Explain briefly.

94. What are the various modes of Serial Data Transmission? Explain each mode briefly.

95. Explain with a timing diagram the shift register mode in the 8051.

96. What is the use of the serial communication mode 0 in the 8051?

97. Explain in detail the Serial Data Mode 1 in the 8051.

98. Explain how the Baud rate is calculated for the Serial Data Mode 1.

99. How is the Baud rate for the Multiprocessor communication Mode calculated?

100. Explain in detail the Multiprocessor communication Mode in the 8051.

101. Explain the significance of the 9th bit in the Multiprocessor communication Mode.

102. Explain the Serial data mode 3 in the 8051.

103. What are interrupts and how are they useful in Real Time Programming?

104. Briefly describe the Interrupt structure in the 8051.

105. Explain about vectored and non-vectored interrupts in general.

106. What are the five interrupts provided in the 8051?

107. What are the three registers that control and operate the interrupts in 8051?

108. Describe the Interrupt Enable (IE) special function register and its various bits.

109. Describe the Interrupt Priority (IP) special function register and its need.

110. Explain in detail how the Timer Flag interrupts are generated.

111. Explain in detail how the Serial Flag interrupt is generated.

112. Explain in detail how the External Flag interrupts are generated.

113. What happens when a high logic is applied on the Reset pin?

114. Why the Reset interrupt is called a non-maskable interrupt?

115. Why do we require a reset pin?

116. How can you enable/disable some or all the interrupts?

117. Explain how interrupt priorities are set? And how interrupts that occur simultaneously are handled. 118. What Events can trigger interrupts, and where do they go after getting triggered?

119. What are the actions taken when an Interrupt Occurs?

110. What are Software generated interrupts and how are they generated?

111. What is RS232 and MAX232?

112. What is the function of RS and E pins in an LCD?

113. What is the use of R/W pin in an LCD?

114. What is the significance of DA instruction?

115. What is packed and unpacked BCD?

116. What is the difference between CY and OV flag?

117. When will the OV flag be set?

118. What is an ASCII code?

MICROCONTROLLER - LAB QUESTION BANK

1. a) Write an ALP to move a Block of N-data starting at location X to location Y using 8051/MSP430 b) Write a C program to interface stepper motor to 8051.

2. a) Write an ALP to find cube of given 8-bit data using 8051 /MSP430. b) Write a C program to interface stepper motor to 8051.

3. a) Write an ALP to implement a binary/decimal up/down counter using 8051 /MSP430. b) Write a C program to interface stepper motor to 8051.

4. a) Write an ALP to find the largest / smallest element in an array using 8051. b) Write a C program to interface stepper motor to 8051.

5. a) Write an ALP to exchange two blocks of data present at location X and Y respectively using 8051/MSP430 b) Write a C program to generate Sine waveform using DAC. Display the waveform on CRO.

6. a) Write an ALP to arrange a set of N 8-bit numbers starting at location X in ascending/descending order using 8051 /MSP430. b) Write a C program to generate triangular wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO

7. a) Write an ALP to perform 16-bit multiplication using 8051 /MSP430. b) Write a C program to generate Ramp wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO.

8. a) Write an ALP to convert two digit BCD number to its equivalent ASCII value using 8051 /MSP430. b) Write a C program to generate square wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO.

9. a) Write an ALP to find whether the given number is palindrome or not using 8051. b) Write a C program to generate Sine waveform using DAC. Display the waveform on CRO.

Hobby Project Circuits

Quiz Buzzer Circuit using 8051 Microcontroller

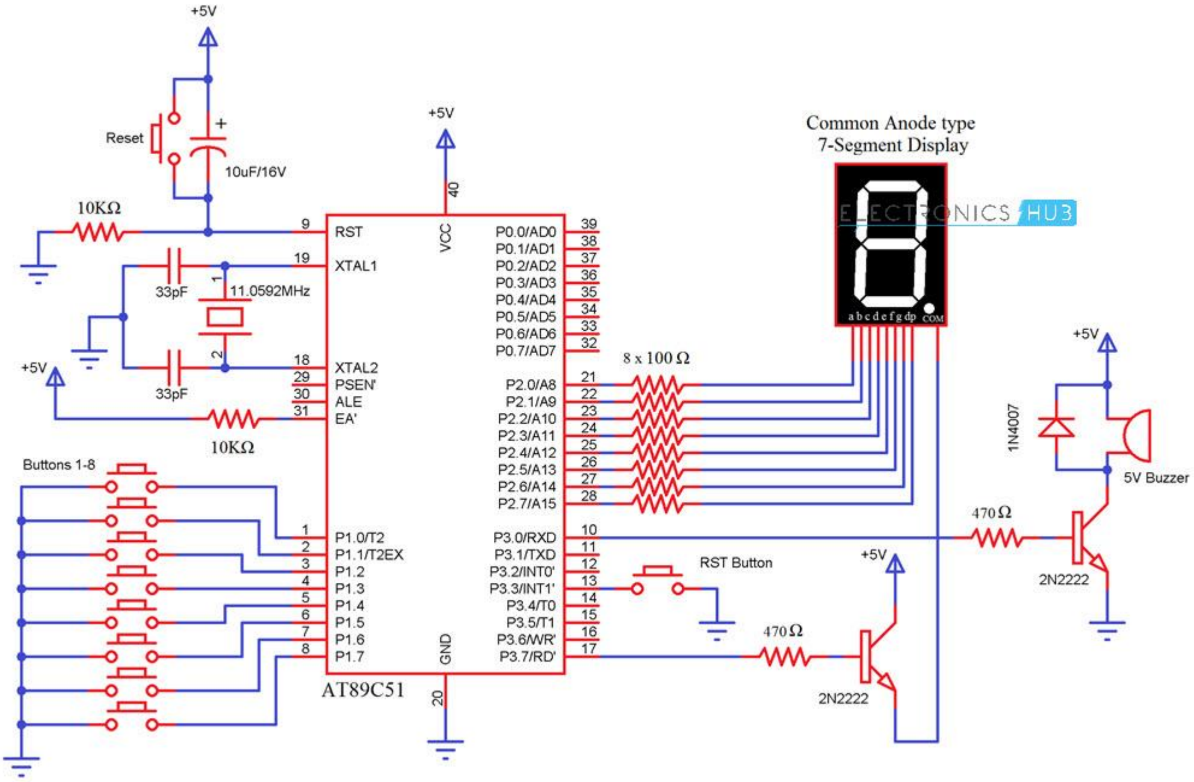
Table of Contents:

1. Principle behind the Quiz Buzzer Circuit

The 8 Channel Quiz Buzzer Circuit using Microcontroller is a simple embedded system with a set of 8 push buttons being the input devices, a microcontroller as the main controller and the output devices being a buzzer and a display.

The whole operation is carried out by a microcontroller through a program written in C language and dumped inside the microcontroller. When one of the buttons is pressed, the buzzer starts ringing and the corresponding number is displayed on the 7 segment display.

2. Circuit Diagram of 8 Player Quiz Buzzer using Microcontroller



3. Components Required

- AT89C51 (8051 Microcontroller)
- 7 Segment Display (Common Anode is used in this project)
- Push Buttons – 10
- 10KΩ Resistors – 2
- 100Ω Resistors – 8
- 470Ω Resistors – 2
- 2N2222 NPN Transistors – 2
- 5V Buzzer
- 1N4007 Diode
- 10µF Capacitor
- 33pF Capacitors – 2
- 11.0592 MHz Crystal
- 8051 Programmer
- 5V Power Supply

4. Design Process

The whole design process involves five steps.

1. First step is designing the circuit.
2. The second step is drawing the schematic using any software.
3. Third step involves writing the code using high level language like C or assembly language and then compiling it on a software platform like Keil µVision.
4. Fourth step is programming the microcontroller with the code.
5. Finally, the fifth step is testing the circuit.

5. Quiz Buzzer Circuit Design

The circuit involves using five major components – 8051 Microcontroller, SPST Push Buttons, a buzzer and a common anode 7 segment display. The microcontroller used in this case is AT89C51, an 8 bit microcontroller manufactured by Atmel (now Microchip).

a. **Reset Circuit Design:** The reset resistor is selected such that the voltage at the reset pin, across this resistor is at minimum of 1.2V and the width of the pulse applied to this pin is greater than 100 ms. Here we select a resistor of 10KΩ and a capacitor of 10µF.

b. **Oscillator Circuit Design:** The oscillator circuit is designed using a crystal oscillator of 11.0592 Mhz and two ceramic capacitors each 33pF. The crystal is connected between pins 18 and 19 of the microcontroller

c. **Microcontroller Interfacing Design:** The set of 8 push buttons are interfaced to port P1 of the microcontroller and a buzzer is interfaced to the port pin P3.3. The 7 segment display is interfaced to the microcontroller such that all the input pins are connected to port P2.

6. **Microcontroller Code:** The code can be written using C language or assembly language. Here, I have written the program in C language using Keil µVision software. This is accomplished by the following steps:

1. Create a new project on Keil window and select the target (microcontroller).
2. Create a new file under the project and write the code.
3. Save the code with .c extension and add the file to the source group folder under the target folder.
4. Compile the code and create the hex file.

Once the code is compiled and a hex file is created, next step is to dump the code into the microcontroller. This can be done with an 8051 Microcontroller Programmer.

CODE

For code: visit the Link
https://www.electronicshub.org/8-channel-quiz-buzzer-circuit-using-microcontroller/

7. How Quiz Buzzer Circuit Works?

Once the circuit is powered, the compiler will initialize the stack pointer and the variables having the non-zero initial values and perform other initialization process and then calls the main function. It then checks if any of the buttons is pressed.

In other words the microcontroller scans for any of its input pins at port P1 to be zero or at logic low level. In case a button is pressed, the display function is called by passing the corresponding number. The microcontroller then sends the relevant signals to the port connected to the 7 segment display.

The microcontroller will turn on the buzzer for a second and turns it off but the number will be continosly displayed on the 7 segment display until the RST button is pressed.

8. Applications of Quiz Buzzer Circuit

1. This circuit can be used at quiz competitions organized at schools, colleges and other institutions.
2. It can be also used for other games shows.
3. It can be used as at public places like banks, restaurants as a digital token display system.

For More Circuits Visit :
https://www.electronicshub.org/microcontroller-based-mini-projects-ideas/