

**BMS COLLEGE OF ENGINEERING, BANGALORE-19**

**(Autonomous College under VTU)**



**Department of Electronics and Communication  
Engineering**

**MICROCONTROLLER LAB MANUAL  
2022**

**Faculty in charge**

**Dr. Rajanikanth Kashi  
Mrs. Bhavana H T  
Mrs. Pooja A P  
Mrs. Monika Sharama D**

**Head of the Department**

**Dr. Siddappaji**

## **List of Experiments**

### **PART – A : ASSEMBLY LEVEL PROGRAMMING (ALP)**

#### **Data Transfer Instructions**

1. Block Transfer of data from internal memory to internal memory (without overlapping)
2. Block Transfer of data from internal memory to internal memory (with overlapping)
3. Block Transfer of data from internal memory to external memory
4. Block exchange of data bytes (Internal memory)
5. Block exchange of data bytes (External memory)
6. Block exchange of data bytes in internal memory using stack.
7. Finding largest/smallest number of block of data bytes
8. Sorting block of data bytes in ascending/descending order
9. Searching for an element and its position in an array.

#### **Arithmetic Instructions**

1. Addition of two 16-bit numbers
2. Subtraction of two 16-bit numbers
3. Finding square of a number
4. Finding cube of a number
5. Division of two 8-bit numbers
6. Addition of 10 bytes of data in RAM

#### **Logic and Boolean instructions**

1. Compare two numbers and interpret result in bit addressable region.
2. Program to count the number of 1's and 0's in a byte (Parity detector)
3. Program to find a number is odd or even
4. Program to perform logical operations.
5. Program to check a number is palindrome
6. Program to count number of positive and negative values in 10 bytes of data

#### **Code Conversions**

1. Program to Hex value to decimal.
2. Program to convert BCD to ASCII
3. Program to convert ASCII to BCD
4. Program to convert decimal number to hex

#### **Counter Programming**

1. Binary up/down counter
2. Decimal up/down counter

#### **Generation of delay using on chip Timer**

#### **Serial communication**

1. Program to serially transmit a message to PC
2. Program to serially receive data elements form PC

## **PART – B : INTERFACING EXPERIMENTS (C PROGRAMMING)**

- 1. 7 segment display unit interfacing to 8051**
- 2. Stepper motor interfacing to 8051**
- 3. Generate sine wave, square wave, triangular wave by interfacing DAC unit to 8051**
- 4. Elevator interfacing to 8051**
- 5. Keyboard interfacing to 8051**

### **Note:**

- Before using kits refer to user manual.
- Handle connectors and interface boards gently with proper care.
- Use 5v Battery supply only.
- Switch off and make connections of cables.

### **General Instructions to Students**

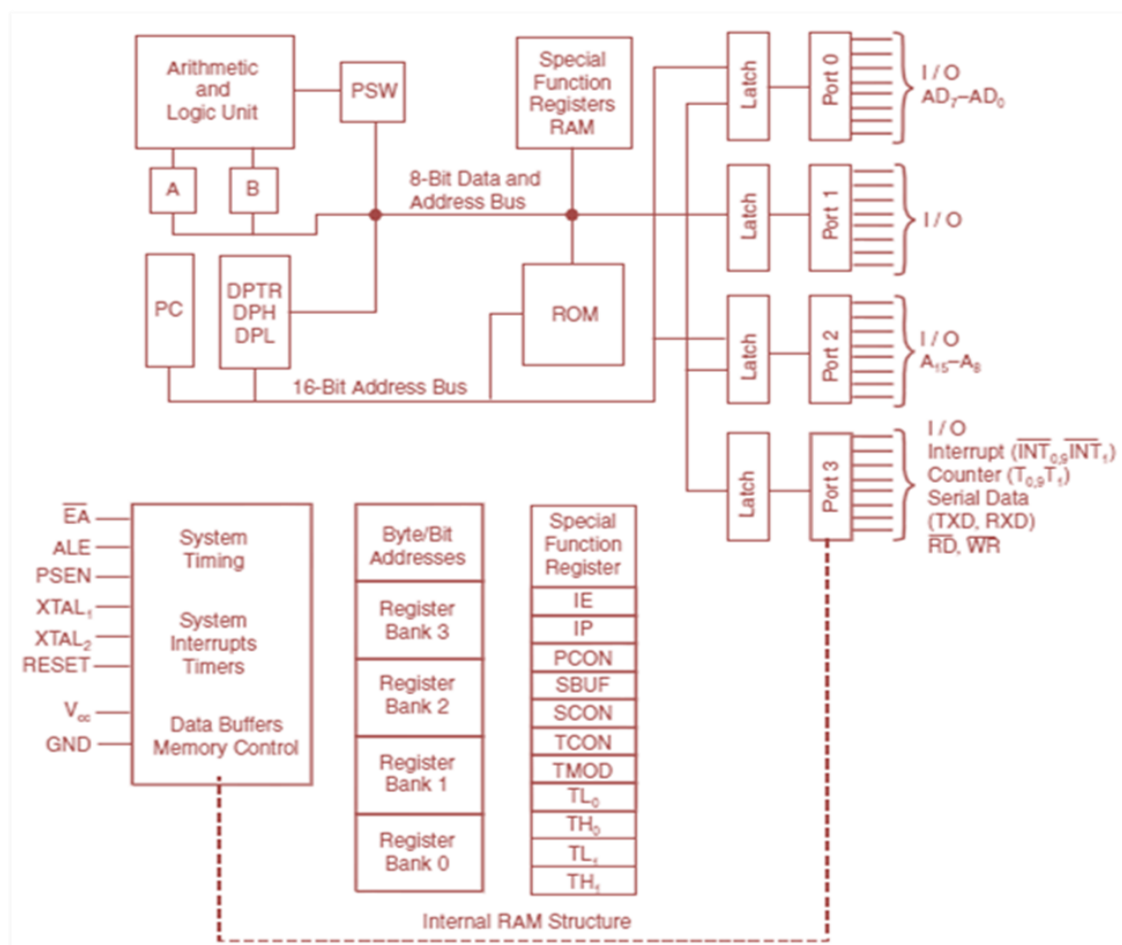
1. Students Should come on time to the lab with thorough preparation for the experiment to be conducted
2. Students should take prior permission from faculty before availing leave
3. Students should maintain practical record book and observation book neatly.
4. Students must submit record book in every lab for evaluation.
5. Students must take signature of faculty for observation book after completion of experiment in each lab.

## Introduction

Earlier to microcontrollers, microprocessors were used extensively for different purposes. Microprocessors contain ALU, general purpose register, stack pointer, program counter clock circuit and so many other circuits which today's microcontrollers also possess. Difference between microcontrollers exist with respect to the number of instructions, access time, size, reliability, PCB size and so on. Microprocessor contains large instruction set called CISC processor whereas microcontrollers contain small instruction set called RISC processor. The input output ports and internal memory (RAM & ROM) makes the microcontroller a microcomputer. In this course we will study about 8051 architecture, features, programming and interfacing.

The 8051 group of microcontroller includes a family of microcontrollers having numbers ranging from 8031 to 8751 and are available in NMOS or CMOS construction. MCS 8051 is 8-bit, single chip (40 pin) microcontroller with many built in functions and is core for all MCS-51 devices.

## Architecture of Intel 8051



The architecture of 8051 consists of following features:

- Operates with single power supply +5V
- **8 – Bit ALU:** ALU or Arithmetic Logic Unit is the heart of a microcontroller. It performs arithmetic and bitwise operation on binary numbers. The ALU in 8051 is an 8 – Bit ALU i.e. it can perform operations on 8 – bit data.
- **8 – Bit Accumulator:** The Accumulator is an important register associated with the ALU. The accumulator in 8051 is an 8 – bit register.
- 16-bit Program Counter (PC) and 16-bit Data Pointer (DPTR).
- 8-bit Stack Pointer (SP)
- **RAM:** 8051 Microcontroller has 128 Bytes of RAM which includes SFRs and Input / Output Port Registers.
- **ROM:** 8051 has 4 KB of on-chip ROM (Program Memory).
- **I/O Ports:** 8051 has four 8 – bit Input / Output Ports which are bit addressable and bidirectional.
- **Timers / Counters:** 8051 has two 16 – bit Timers / Counters.
- **Serial Port:** 8051 supports full duplex UART Communication.
- **External Memory:** 8051 Microcontroller can access two 16 – bit address line at once: one each for RAM and ROM. The total external memory that an 8051 Microcontroller can access for RAM and ROM is 64KB ( $2^{16}$  for each type).
- **Additional Features:** Interrupts, on-chip oscillator, Boolean Processor, Power Down Mode, etc.

## Pin Description of 8051

Sl. No	Mnemonic	Pin No.	Type	Description
1.	V <sub>ss</sub>	20	I	<b>Ground:</b> 0 V reference
2.	V <sub>cc</sub>	40	I	<b>Power supply:</b> Power supply voltage for normal, idle and shut down operations
3.	P0.0-0.7	39-32	I/O	<b>Port 0:</b> Port 0 is an open-drain, bidirectional I/O port with Schmitt trigger inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external program and data memory. Port 0 also outputs the code bytes during program verification and received code bytes during EPROM programming.
4.	P1.0-P1.7	1-8	I/O	<b>Port 1:</b> Port 1 is an 8-bit bidirectional I/O port with internal pull-ups and Schmitt trigger inputs. Port 1 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 1 pins that are externally pulled low will source current because of the internal pull-ups. Port 1 also receives the low-order address byte during program memory verification. Alternate functions for Port 1 include:
5.	P2.0-P2.7	21-28	I/O	<b>Port 2:</b> Port 2 is an 8-bit bidirectional I/O port with internal pull-ups and Schmitt trigger inputs. Port 2 pins that have 1s written to them are pulled high by the internal pull-ups and can be used as inputs. As inputs, port 2 pins that are externally being pulled low will source current because of the internal pull-ups. Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses. Some Port 2 pins receive the high order address bits during EPROM programming and verification.

6.	P3.0-P3.7	10-17	I/O	<p><b>Port 3:</b> Port 3 is an 8-bit bidirectional I/O port with internal pull-ups and Schmitt trigger inputs. Port 3 also serves the special features of the 80C51 family, as listed below:</p> <p><b>RxD (P3.0):</b> Serial input port</p> <p><b>TxD (P3.1):</b> Serial output port</p> <p><b>INT0 (P3.2):</b> External interrupt</p> <p><b>INT1 (P3.3):</b> External interrupt</p> <p><b>T0 (P3.4):</b> Timer 0 external input</p> <p><b>T1 (P3.5):</b> Timer 1 external input</p> <p><b>WR (P3.6):</b> External data memory write strobe</p> <p><b>RD (P3.7):</b> External data memory read strobe</p>
		10	I	
		11	O	
		12	I	
		13	I	
		14	I	
		15	I	
		16	O	
		17	O	
7.	RST	9	I	<p><b>Reset:</b> A high on this pin for two machine cycles while the oscillator is running, resets the device. An internal diffused resistor to VSS permits A power-on reset using only an external capacitor to VCC</p>
8.	ALE/PROG	30	O	<p><b>Address Latch Enable/Program Pulse:</b> Output pulse for latching the low byte of the address during an access to external memory. In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency, and can be used for external timing or clocking. Note that one ALE pulse is skipped during each access to external data memory. This pin is also the program pulse input (PROG) during EPROM programming. ALE can be disabled by setting SFR auxiliary.0. With this bit set, ALE will be active only during a MOVX instruction.</p>
9.	PSEN	29	O	<p><b>Program Store Enable:</b> The read strobe to external program memory. When the device is executing code from the external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory. PSEN is not activated during fetches from internal program memory.</p>
10.	EA/Vpp	31	I	<p><b>External Access Enable/Programming Supply Voltage:</b> EA must be externally held low to enable the device to fetch code from external program memory locations 0000H to 0FFFFH. If EA is held high, the device executes from internal program memory unless the program counter contains an address greater than the on-chip ROM/OTP. This pin also receives the 12.75 V programming supply voltage (VPP) during EPROM programming. If security bit 1 is programmed, EA will be internally latched on Reset.</p>
11.	XTAL1	19	I	<p><b>Crystal 1:</b> Input to the inverting oscillator amplifier and input to the internal clock generator circuits</p>
12.	XTAL2	18	O	<p><b>Crystal 2:</b> Output from the inverting oscillator amplifier</p>

## Instruction set for AT89C51ED2

Sl No.	Mnemonics	Opcode	Bytes	No. of machine cycles	Description
<b>Arithmetic operations</b>					
1	ADD A, Rn	28-2F	1	1	Add register to the accumulator
2	ADD A, direct	25	2	1	Add direct byte to the accumulator
3	ADD A, @Ri	26-27	1	1	Add indirect RAM to accumulator
4	ADD A, #data	24	2	1	Add immediate data to the accumulator
5	ADDC A, Rn	38-3F	1	1	Add register to Accumulator with carry
6	ADDC A, direct	35	2	1	Add direct byte to Accumulator with carry
7	ADDC A,@Ri	36-37	1	1	Add indirect RAM to Accumulator with carry
8	ADDC A, #data	34	2	1	Add immediate data to ACC with carry
9	SUBB A, Rn	98-9F	1	1	Subtract Register from ACC with borrow
10	SUBB A, direct	95	2	1	Subtract direct byte from ACC with borrow
11	SUBB A, @Ri	96-97	1	1	Subtract indirect RAM from ACC with borrow
12	SUBB A, #data	94	2	1	Subtract immediate data from ACC with borrow
13	INC A	04	1	1	Increment Accumulator
14	INC Rn	08-0F	1	1	Increment register
15	INC direct	05	2	1	Increment direct byte
16	INC @Ri	06-07	1	1	Increment indirect RAM
17	DEC A	14	1	1	Decrement Accumulator
18	DEC Rn	18-1F	1	1	Decrement Register
19	DEC direct	15	2	1	Decrement direct byte
20	DEC @Ri	16-17	1	1	Decrement indirect RAM
21	INC DPTR	A3	1	2	Increment Data Pointer
22	MUL AB	A4	1	3	Multiply A and B
23	DIV AB	84	1	3	Divide A by B
24	DA A	D4	1	1	Decimal Adjust Accumulator
<b>Logical operations</b>					
25	ANL A, Rn	58-5F	1	1	AND Register to Accumulator
26	ANL A, direct	55	2	1	AND direct byte to Accumulator
27	ANL A, @Ri	56-57	1	1	AND indirect RAM to Accumulator
28	ANL A, #data	54	2	1	AND immediate data to Accumulator
29	ANL direct, A	52	2	1	AND Accumulator to direct byte
30	ANL direct, #data	53	3	2	AND immediate data to direct byte
31	ORL A, Rn	48-4F	1	1	OR register to Accumulator
32	ORL A, direct	45	2	1	OR direct byte to Accumulator
33	ORL A, @Ri	46-47	1	1	OR indirect RAM to Accumulator
34	ORL A, #data	44	2	1	OR immediate data to Accumulator
35	ORL direct, A	42	2	1	OR Accumulator to direct byte
36	ORL direct, #data	43	3	2	OR immediate data to direct byte
37	XRL A, Rn	68-6F	1	1	Exclusive-OR register to Accumulator
38	XRL A, direct	65	2	1	Exclusive-OR direct byte to Accumulator
39	XRL A, @Ri	66-67	1	1	Exclusive-OR indirect RAM to Accumulator
40	XRL A, #data	64	2	1	Exclusive-OR immediate data to Accumulator
41	XRL direct, A	62	2	1	Exclusive-OR Accumulator to direct byte
42	XRL direct, #data	63	3	2	Exclusive-OR immediate data to direct byte
43	CLR A	E4	1	1	Clear Accumulator

44	CPL A	F4	1	1	Complement Accumulator
45	RL A	23	1	1	Rotate Accumulator left
46	RLC A	33	1	1	Rotate Accumulator left through the carry
47	RR A	03	1	1	Rotate Accumulator right
48	RRC A	13	1	1	Rotate Accumulator right through the carry
49	SWAP A	C4	1	1	Swap nibbles within the Accumulator
<b>Data transfer</b>					
50	MOV A, Rn	E8-EF	1	1	Move register to Accumulator
51	MOV A, direct	E5	2	1	Move direct byte to Accumulator
52	MOV A, @Ri	E6-E7	1	1	Move indirect RAM to Accumulator
53	MOV A, #data	74	2	1	Move immediate data to Accumulator
54	MOV Rn, A	F8-FF	1	1	Move Accumulator to register
55	MOV Rn, direct	A8-AF	2	2	Move direct byte to register
56	MOV Rn, #data	78-7F	2	1	Move immediate data to register
57	MOV direct, A	F5	2	1	Move Accumulator to direct byte
58	MOV direct, Rn	88-8F	2	2	Move register to direct byte
59	MOV direct, direct	85	3	2	Move direct byte to direct
60	MOV direct, @Ri	86-87	2	2	Move indirect RAM to direct byte
61	MOV direct, #data	75	3	2	Move immediate data to direct byte
62	MOV @Ri, A	F6-F7	1	1	Move Accumulator to indirect RAM
63	MOV @Ri, direct	A6-A7	2	2	Move direct byte to indirect RAM
64	MOV @Ri, #data	76-77	2	1	Move immediate data to indirect RAM
65	MOV DPTR, #data16	90	3	2	Load Data Pointer with a 16-bit constant
66	MOVC A, @A+DPTR	93	1	2	Move Code byte relative to DPTR to ACC
67	MOVC A, @A+PC	83	1	2	Move Code byte relative to PC to ACC
68	MOVX A, @Ri	E5-E6	1	2	Move external RAM (8-bit adder) to ACC
69	MOVX A, @DPTR	E0	1	2	Move external RAM (16-bit adder) to ACC
70	MOVX A, @Ri, A	F2-F3	1	2	Move ACC to external RAM (8-bit address)
71	MOVX @DPTR, A	F0	1	2	Move ACC to external RAM (16-bit address)
72	PUSH direct	C0	2	2	Push direct byte onto stack
73	POP direct	D0	2	2	Push direct byte onto stack
74	XCH A, Rn	C8-CF	1	1	Exchange register with Accumulator
75	XCH A, direct	C5	2	1	Exchange direct byte with Accumulator
76	XCH A, @Ri	C6-C7	1	1	Exchange indirect RAM with Accumulator
77	XCHD A, @Ri	D6-D7	1	1	Exchange low-order digit indirect RAM with ACC
<b>Boolean variable manipulation</b>					
78	CLR C	C3	1	1	Clear carry
79	CLR bit	C2	2	1	Clear direct bit
80	SETB C	D3	1	1	Set carry
81	SETB bit	D2	2	1	Set direct bit
82	CPL C	B3	1	1	Complement carry
83	CPL bit	B2	2	1	Complement direct bit
84	ANL C, bit	82	2	2	AND direct bit to carry
85	ANL C, /bit	B0	2	2	AND complement of direct bit to carry
86	ORL C, bit	72	2	2	OR direct bit to carry
87	ORL C, /bit	A0	2	2	OR complement of direct bit to carry
88	MOV C, bit	A2	2	1	Move direct bit to carry
89	MOV bit, C	92	2	2	Move carry to direct bit
90	JC rel <sup>1</sup>	40	2	2	Jump if carry is set
91	JNC rel	50	2	2	Jump if carry not set
92	JB rel	20	3	2	Jump if direct bit is set
93	JNB bit, rel	30	3	2	Jump if direct bit is not set



94	JBC bit, rel	10	3	2	Jump if direct bit is set and clear bit
<b>Program branching</b>					
95	ACALL addr11	X1 <sup>2</sup>	2	2	Absolute subroutine call
96	LCALL addr16	12	3	2	Long subroutine call
97	RET	22	1	2	Return from subroutine
98	RETI	32	1	2	Return from interrupt
99	AJMP addr11	Y1 <sup>3</sup>	2	2	Absolute jump
100	LJMP addr16	02	3	2	Long jump
101	SJMP rel	80	2	2	Short jump (relative address)
102	JMP @A+DPTR	73	1	2	Jump indirect relative to the DPTR
103	JZ rel	60	2	2	Jump if Accumulator is zero
104	JNZ rel	70	2	2	Jump if Accumulator is not zero
105	CJNE A, direct, rel	B5	3	2	Compare direct byte to ACC and jump if not equal
106	CJNE A, #data, rel	B4	3	2	Compare immediate to ACC and jump if not equal
107	CJNE @Rn, data, rel	B6-B7	3	2	Compare immediate to register and jump if not equal
108	CJNE @Ri, #data, rel	B8-BF	3	2	Compare immediate to indirect and jump if not equal
109	DJNZ Rn, rel	D8-DF	2	2	Decrement register and jump if not zero
110	DJNZ direct, rel	D5	3	2	Decrement direct byte and jump if not zero
111	NOP	00	1	1	No operation

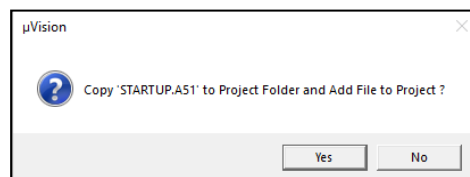
NOTE:

1. rel is any label.
2. X= 1,3,5,7,9,B,D,F
3. Y=0,2,4,6,8,A,C,E

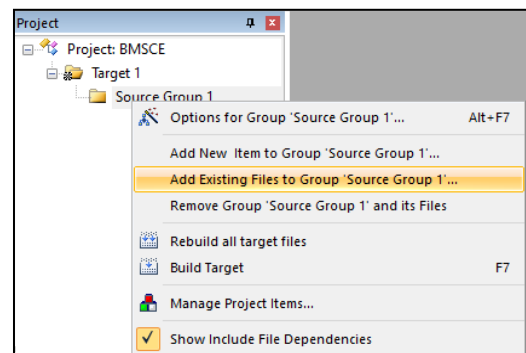
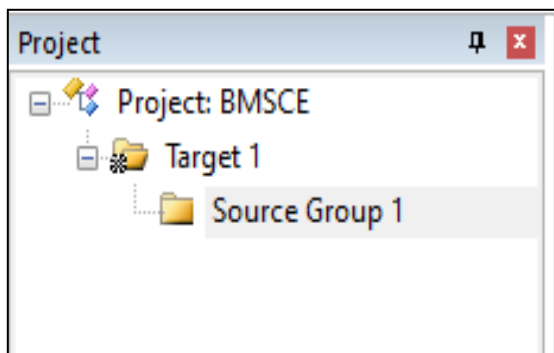
## Steps to Create and Compile Keil $\mu$ vision - 5 Projects.



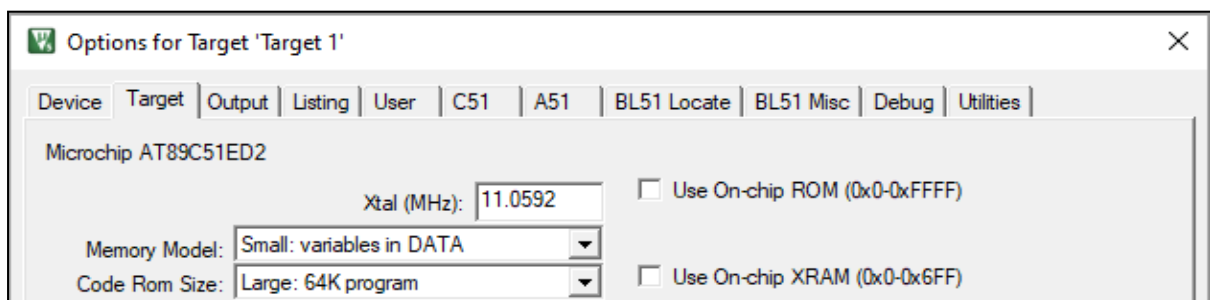
1. Double Click on Keil  $\mu$ vision - 5 icon on the desktop.
2. Close any previous projects that were opened using **Project**  $\rightarrow$  **Close**
3. Create a folder with your USN as folder in E/F drive.
4. Create a project by clicking on **Project**  $\rightarrow$  **New Project** and give one name of the project (Automatically saved as .uvproj)
5. Select the target device from the device database (Database- AT89C51ED2 as per the board). On clicking OK the following window will be displayed, choose **No**.

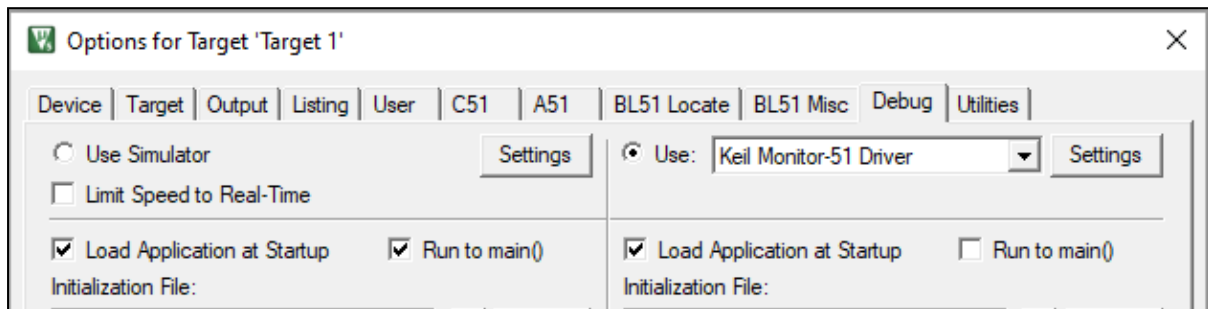


6. Create a source file using **File**  $\rightarrow$  **New**, type the assembly or C program and save the file (filename.asm/filename.c). Add this file to the source group by right clicking on source group in the Project Window and add existing file to the source group.

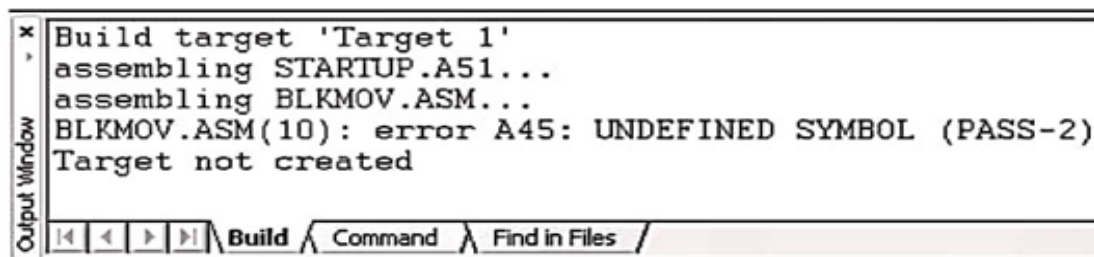



7. Set the target option using **Project**  $\rightarrow$  **Options for Targets**, it opens the  $\mu$ Vision target options dialogue box. Set the **Xtal frequency as 11.0592MHz**. Options for Targets Debug use either **Simulator** (for simulation ALP programs)/ **Keil Monitor 51 driver** (for interfacing experiments).







8. Build the project by using **Project** → **Build project**.  $\mu$ vision translates all the user applications and link. Any errors in the code are indicated by "Target not Created" message in the build window along with the error line. Debug the error and build the project again. Once the script is error free go to **Debug** option.



9. The programmer can enter into debug mode by clicking on Start/Stop Debug session dialog or by clicking on  icon.

10. The program is run using **Debug** → **Run** command and halted using **Debug** → **Stop Running** command. Also  icons (reset, run, halt) can be used, additional icons such as  (step, step over, step out, run to cursor) can be used to execute each instruction step by step or to come out of the loop.

11. For assembly level programs, the appropriate result can be seen in memory window or register window. Click on **View** → **Memory** windows to open memory windows. In memory window type **internal RAM I: XX H** (locations ranging from 00H to 7FH), **data RAM D:XX H** (locations ranging from 80H to FFH) and for **external RAM X: xxxx H** (locations ranging from 0000H to FFFFH). Watch window (Timer program), Serial window are also used to check the results.

12. In interfacing experiments, the output can be seen using LCD display, LED display, motor, seven segment display, CRO etc.

## PART – A : ASSEMBLY LEVEL PROGRAMMING (ALP)

### Data Transfer Instructions

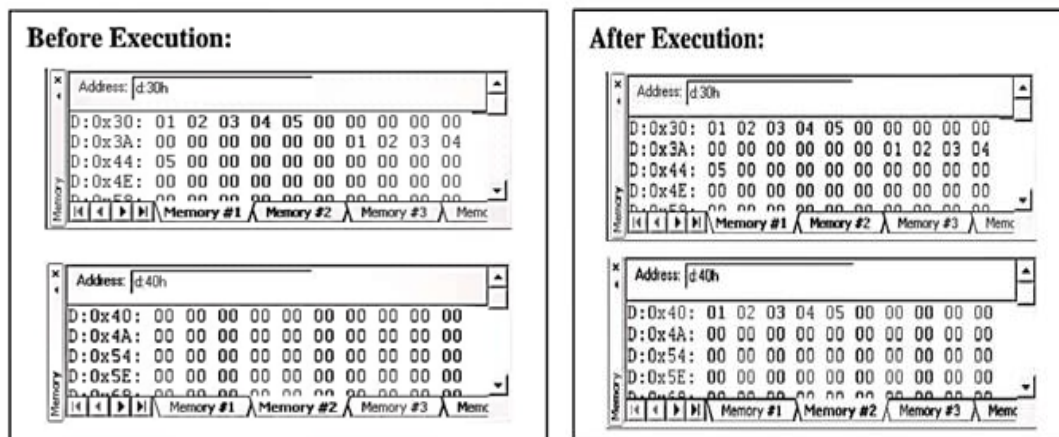
#### 1. Write and ALP to transfer 5 byte of data from 30H to 40H (without overlapping).

```
ORG 00H
    MOV R7, #05H      //5 DATA ELEMENTS
    MOV R0, #30H      //Source Address
    MOV R1, #40H      //Destination address

NEXT: MOV A, @R0
      MOV @R1, A
      INC R0
      INC R1
      DJNZ R7, NEXT    // Repeat till all 5 data bytes are copied
HERE: SJMP HERE

END
```

#### Result



#### 2. Write and ALP to transfer 5 byte of data from 30H to 33H (with overlapping)

```
ORG 00H
    MOV R7, #05H      //5 Data elements
    MOV R0, #34H      //Source last address
    MOV R1, #37H      //Destination last address

NEXT: MOV A, @R0
      MOV @R1, A
      DEC R0
      DEC R1
      DJNZ R7, NEXT    // Repeat till all 5 data bytes are copied
HERE: SJMP HERE

END
```

## Result

### Before Execution

Memory 1	
Address:	I:30H
I:0x30:	12 5C 9F 6D 3E 00 00 00 00
I:0x48:	00 00 00 00 00 00 00 00 00
I:0x60:	00 00 00 00 00 00 00 00 00
I:0x78:	00 00 00 00 00 00 00 00 00
I:0x90:	00 00 00 00 00 00 00 00 00

### After Execution

Memory 2	
Address:	I:33H
I:0x33:	12 5C 9F 6D 3E 00 00 00 00
I:0x4B:	00 00 00 00 00 00 00 00 00
I:0x63:	00 00 00 00 00 00 00 00 00
I:0x7B:	00 00 00 00 00 00 00 00 00
I:0x93:	00 00 00 00 00 00 00 00 00

### 3. Write an ALP to move five data from internal memory I: 0x90 to external data memory x: 0x1000

ORG 00H

```
MOV R7,#05H      //5 Data elements
MOV R0,#90H      //Source address
MOV DPTR,#1000H  //Destination address
```

```
NEXT: MOV A,@R0
      MOVX @DPTR,A
      INC R0
      INC DPTR
      DJNZ R7,NEXT      //Repeat till all 5 data bytes are copied
      HERE: SJMP HERE
```

END

## Result

### Before Execution

Memory 2	
Address:	I:90H
I:0x90:	45 33 89 FC DA 00 00 00
I:0xA8:	00 00 00 00 00 00 00 00
I:0xC0:	00 00 00 00 00 00 00 00
I:0xD8:	00 00 00 00 00 00 00 00
I:0xF0:	00 00 00 00 00 00 00 00

### After Execution

Memory 1	
Address:	X:1000H
X:0x001000:	45 33 89 FC DA 00 00 00
X:0x001017:	00 00 00 00 00 00 00 00
X:0x00102E:	00 00 00 00 00 00 00 00
X:0x001045:	00 00 00 00 00 00 00 00
X:0x00105C:	00 00 00 00 00 00 00 00

### 4. Write an ALP to exchange 5 bytes of data at location I: 030H and at I: 40H.

ORG 00H

```
MOV R0,#30H      //5 Data elements
MOV R1,#40H      //Source address
MOV R7,#05H      //Destination address
```

```
NEXT: MOV A,@R0
      XCH A,@R1
      XCH A,@R0
      INC R0
```

```

        INC R1
        DJNZ R7,NEXT          //Repeat till all 5 data bytes are copied
HERE:   SJMP HERE

        END

```

## Result

### Before Execution

Memory 1	
Address:	I:30H
I:0x30:	01 02 03 04 05 00
I:0x48:	00 00 00 00 00 00

Memory 2	
Address:	I:40H
I:0x40:	0A 0B 0C 0D 0E 00
I:0x58:	00 00 00 00 00 00

### After Execution

Memory 1	
Address:	I:30H
I:0x30:	0A 0B 0C 0D 0E 00
I:0x48:	00 00 00 00 00 00

Memory 2	
Address:	I:40H
I:0x40:	01 02 03 04 05 00
I:0x58:	00 00 00 00 00 00

## 5. Write an ALP to exchange 5 bytes of data at location X: 0100H and at X: 0200H.

```

ORG 00H

        MOV DPTR,#0100H      //External address
        MOV R0,#05H          //5 Data elements
        MOV R1,#01H          // DPH data
        MOV R2,#02H          // DPL data
BACK:   MOVX A,@DPTR
        MOV R3,A
        MOV 83H,R2            //Switch DPTR to 0200H
        MOVX A,@DPTR
        MOV 83H,R1            //Switch DPTR to 0100H
        MOVX @DPTR,A
        MOV A,R3
        MOV 83H,R2
        MOVX @DPTR,A
        MOV 83H,R1
        INC DPTR
        DJNZ R0,BACK          //Repeat till all 5 data bytes are exchanged
HERE : SJMP HERE

        END

```

## Result

### Before Execution

Memory 2	
Address:	x:0100h
X:0x000100:	11 22 33 44 55 00
X:0x000117:	00 00 00 00 00 00

### After Execution

Memory 2	
Address:	x:0100h
X:0x000100:	AA BB CC DD EE 00
X:0x000117:	00 00 00 00 00 00

Memory 1	
Address:	x:0200H
X:0x000200:	AA BB CC DD EE
X:0x000217:	00 00 00 00 00

Memory 1	
Address:	x:0200H
X:0x000200:	11 22 33 44 55 00
X:0x000217:	00 00 00 00 00 00

## 6. Write an ALP to exchange 3 bytes of data at location I: 30H and at I: 40H using stack

ORG 00H

```

PUSH 30H    //Copy data of 30H to stack
PUSH 31H
PUSH 32H
PUSH 40H    //Copy data of 40H to stack
PUSH 41H
PUSH 42H
POP 32H     //Copy data from stack to 32H
POP 31H
POP 30H
POP 42H     //Copy data from stack to 42H
POP 41H
POP 40H

```

HERE: SJMP HERE  
END

## Result

### Before Execution

Memory 1	
Address:	I:30H
I:0x30:	56 22 89 00 00 00
I:0x48:	00 00 00 00 00 00

Memory 1	
Address:	I:30H
I:0x30:	CD FF A6 00 00
I:0x48:	00 00 00 00 00 00

### After Execution

Memory 2	
Address:	I:40H
I:0x40:	CD FF A6 00 00 00
I:0x58:	00 00 00 00 00 00

Memory 2	
Address:	I:40H
I:0x40:	56 22 89 00 00 00
I:0x58:	00 00 00 00 00 00

## 7. Write an ALP to find largest element in an array of 6 bytes from x: 4000H location and place it in X:4062H.

ORG 00H

```

MOV R3, #05H           // Length of an array
MOV DPTR, #4000H       // Starting address of array
MOVX A, @DPTR
MOV R1, A
NEXTBYTE: INC DPTR
MOVX A, @DPTR
CLR C                  // Reset borrow flag
MOV R2, A              // Next number in the array

```

```

SUBB A, R1                //Other number/previous larger number
JC SKIP                  //JNC for smallest element
MOV A, R2
MOV R1, A                //Update larger number in R1
SKIP: DJNZ R3, NEXTBYTE
MOV DPTR, #4062H         //Location of the result
MOV A, R1
MOVX @DPTR, A            //Store largest number in 4062H

HERE: SJMP HERE
END

```

## Result

### Before Execution

Memory 1									
Address: <input type="text" value="x:4000h"/>									
X:0x004000:	01	02	03	09	FF	66	00		
X:0x004025:	00	00	00	00	00	00	00		

### After Execution

Memory 2									
Address: <input type="text" value="x:4062h"/>									
X:0x004062:	FF	00	00	00	00	00	00		
X:0x004087:	00	00	00	00	00	00	00		

## 8. Write an ALP to sort 5 data bytes stored in X :9000H in ascending order (Bubble sort algorithm).

```

ORG 00H
MOV R0, #05H              // Length of an array-1
LOOP1: MOV DPTR, #9000H   //Starting address of array FROM 9000H
MOV R1, #05H              //Initialize exchange counter
LOOP2: MOVX A, @DPTR
MOV B, A                  //Get number from array and store in B reg
INC DPTR
MOVX A, @DPTR             //Next number in the array
CLR C                     //Reset borrow flag
MOV R2, A
SUBB A, B                 //2nd -1st number
JNC NOEXCHANGE            //JC for descending order
MOV A, B                  //Exchange two numbers in the array
MOVX @DPTR, A
DEC DPL                   //No "DEC DPTR" in instruction set of 8051
MOV A, R2
MOVX @DPTR, A
INC DPTR
NOEXCHANGE: DJNZ R1, LOOP2 //Decrement compare counter
DJNZ R0, LOOP1           //decrement pass counter
HERE: SJMP HERE
END

```



## Result

### Before Execution

Memory 1	
Address:	X:9000h
X:0x009000:	66 55 44 33 22 11 00 00
X:0x009025:	00 00 00 00 00 00 00 00
X:0x00904A:	00 00 00 00 00 00 00 00

### After Execution

Memory 1	
Address:	X:9000h
X:0x009000:	11 22 33 44 55 66 00 00
X:0x009025:	00 00 00 00 00 00 00 00
X:0x00904A:	00 00 00 00 00 00 00 00

9. Write an ALP to search given element which is in X:0100H from an array of numbers at location X:0101H-0105H. If found indicate its position in 0200h.

ORG 00H

```
MOV DPTR, #0100H
MOVX A, @DPTR           //Data byte which needs to be searched
MOV R0, A
MOV R1, #05H            //Number of data bytes in array
MOV R2, #00H
NEXT: INC DPTR
INC R2                  //Position of data byte
MOVX A, @DPTR
CJNE A, 00H, LOOP       //Compare data byte with each elements in an array
MOV DPTR, #0200H        //If data byte is found indicate position in 0200H
MOV A, R2
MOVX @DPTR, A
SJMP HERE
LOOP: DJNZ R1, NEXT
MOV DPTR, #0200H
MOV A, #00H
MOVX @DPTR, A           //If data byte not found put #00H in 0200H address.
HERE: SJMP HERE
END
```

## Result

### Before Execution

Memory 3	
Address:	X:100H
X:0x000100:	1F 89 36 1F 6D F0 00 00
X:0x000117:	00 00 00 00 00 00 00 00

### After Execution

Memory 2	
Address:	X:0200H
X:0x000200:	03 00 00 00
X:0x000217:	00 00 00 00

## Arithmetic Instructions

### 1. Write an ALP to add two 16-bit numbers

ORG 00h

```
MOV R0, #34H           //Lower byte of 1st number
MOV R1, #12H           //Higher byte of 1st number
MOV R2, #0DCH          //Lower byte of 2nd number
MOV R3, #0FEH          //Higher byte of 2nd number
CLR C
```

```

MOV A, R0
ADD A, R2           //Add first byte
MOV 23H, A
MOV A, R1
ADDC A, R3          //Add with carry
MOV 22H, A
JNC HERE
INC 21H             //Carry from bit position 15 to 16
HERE: SJMP HERE

END

```

### Result

#### Before Execution

12	34	H
+	FE	DC H
<hr/>		

#### After Execution

Memory 2	
Address:	I:21H
I:0x21:	01 11 10 00
I:0x39:	00 00 00 00

## 2. Write an ALP to subtract two 16-bit numbers.

```
ORG 00h
```

```

MOV R0, #0DCH      //Lower byte of 1st number
MOV R1, #0FEH      //Higher byte of 1st number
MOV R2, #34H       //Lower byte of 2nd number
MOV R3, #12H       //Higher byte of 2nd number
CLR C
MOV A, R0
SUBB A, R2          //Subtract first byte
MOV 22H, A
MOV A, R1
SUBB A, R3          //Subtract with borrow
MOV 21H, A
MOV 00H, C          //Borrow from bit position 16 to 15 indicates result is negative

```

```
HERE: SJMP HERE
```

```
END
```

### Result

#### Before Execution

FE	DC	H
-	12	34 H
<hr/>		

#### After Execution

Memory 2	
Address:	I:20H
I:0x20:	00 EC A8 00
I:0x38:	00 00 00 00

### 3. Write an ALP to find a square of an 8-bit number.

ORG 00h

```
MOV A, #7FH
MOV B, A
MUL AB           //Multiply two numbers result in A B registers
MOV 30H, A
MOV 31H, B
```

HERE: SJMP HERE

END

#### Result

**Before Execution**      A = 7FH

**After Execution**

Memory 2			
Address: 1:30H			
I:0x30:	01	3F	00
I:0x48:	00	00	00

### 4. Write an ALP to find a cube of an 8-bit number.

ORG 00H

```
MOV R0, #30h
MOV A, @R0           // Move value at address pointed by R0 to A
MOV R1, A             // Copy the value to R1
MOV B, A              // Copy value to B
MUL AB
MOV R2, B             // R2 holds higher byte of square
MOV B, R1              // Copy number again to B
MUL AB                // Multiply lower byte of square and number
MOV 33H, A             // 33H holds lower byte of result
MOV R3, B              // R3 holds higher byte of intermediate result
MOV A, R1              // Copy number to A
MOV B, R2              // Copy higher byte of square to R2
MUL AB                // Intermediate result, lower byte to be added to R3
CLR C
ADD A, R3
MOV 32H, A             // 32h holds middle byte
JNC SKIP
INC B
SKIP: MOV A, B
MOV 31H, A             // 31h holds higher byte
HERE: SJMP HERE
```

END

#### Result

### Before Execution

Memory 2	
Address:	I:30H
I:0x30:	9D 00 00
I:0x48:	00 00 00

### After Execution

Memory 2	
Address:	I:30H
I:0x30:	9D 3B 0C C5 00
I:0x48:	00 00 00 00 00

## 5. Program to divide an 8bit number stored in memory by another 8 bit number.

ORG 00H

```
MOV DPTR,#0100H
MOVX A,@DPTR
MOV 0F0H,A           // Divider in B register
INC DPTR
MOVX A,@DPTR         // Dividend in A register
DIV AB              // Quotient in A register
INC DPTR
MOVX @DPTR,A
MOV A,0F0H          // Remainder in B register
INC DPTR
MOVX @DPTR,A
```

HERE: SJMP HERE

END

### Result

#### Before Execution

Memory 2	
Address:	X:0100H
X:0x000100:	12 DE 00 00
X:0x000117:	00 00 00 00

#### After Execution

Memory 2	
Address:	X:0100H
X:0x000100:	12 DE 0C 06 00
X:0x000117:	00 00 00 00 00

## 6. Write an ALP to add 10 data bytes in RAM

ORG 00H

```
CLR C
MOV R3, #09H
MOV R0, #50H          //Initial address of array
MOV A, @R0
LOOP: MOV B, A        //First byte
INC R0
MOV A, @R0            //Second byte
ADD A, B
JNC SKIP              //Check carry and increment one register if carry exists
INC R1
SKIP: DJNZ R3, LOOP   //Addition of all 10 bytes
MOV R2, A
HERE: SJMP HERE
```

END

## Result

**Before Execution :**

Memory 2															
Address:		I:50H													
I:0x50:		12	89	76	DC	12	FA	B9	41	02	AC	00			
I:0x68:		00	00	00	00	00	00	00	00	00	00	00			

**After Execution:** Result is in R1 and R2 registers

Regs	
r0	0x59
r1	0x04
r2	0xa1

## Logic and Boolean instructions

1. Write an ALP to compare two 8-bit numbers, NUM1 and NUM2 stored in external memory locations 8000H and 8001H respectively. Reflect your result as: If NUM1<NUM2, SET LSB of data RAM locations 2FH. If NUM1>NUM2, SET MSB of 2FH. If NUM1=NUM2, then clear both MSB and LSB of bit addressable memory location 2FH

ORG 00H

```
MOV DPTR, #8000H
MOVX A, @DPTR          //NUM1
MOV R0, A
INC DPTR
MOVX A, @DPTR          //NUM2
CLR C
SUBB A, R0              //Subtract to compare two numbers
JZ EQUAL                //If result is zero both numbers are equal
JNC SMALL               //If no carry NUM < NUM2
SETB 7FH                // MSB bit of byte address 2FH
SJMP HERE
SMALL: SETB 78H          //LSB bit of byte address 2FH
      SJMP HERE
EQUAL: CLR 78H
      CLR 7FH
HERE: SJMP HERE
```

END

## Result

1. Before Execution: X:8000H                      &                      X:8001H:  
After Execution I:2FH:

2. Before Execution: X:8000H                      &                      X:8001H:  
After Execution I:2FH:

3. Before Execution: X:8000H                      &                      X:8001H:  
After Execution I:2FH:

## 2. Write an ALP to count number of Ones and Zeros in an 8-bit number.

```
ORG 00H

    MOV R1, #00H           //To count number of Zeros
    MOV R2, #00H           //To count number of Ones
    MOV R7, #08H           //To check 8-bits of a byte
    MOV A, #97H            //Data to be verified
AGAIN: RLC A               //Rotate left with carry
    JC NEXT
    INC R1                 //Zero count
    SJMP HERE
NEXT: INC R2               //One count
HERE: DJNZ R7, AGAIN
HERE1: SJMP HERE1

END
```

### Result

**Before Execution :** A = 97H

### After Execution

R1 - No of Zeros

R2 - No of Ones

Register	Value
Regs	
r0	0x00
r1	0x03
r2	0x05

## 3. Write an ALP to find a number is odd or even. If odd store 00H in the accumulator, if even store FFH in the accumulator.

```
ORG 00H

    MOV A, #21H           //The numb to be evaluated
    JB ACC.0, ODD         //Checking LSB of accumulator
    MOV A, #0FFH
    SJMP HERE
ODD:  MOV A, #00H
    HERE: SJMP HERE

END
```

### Result

**Before Execution :** A = 20H

**After Execution: A=0FFH**

**Before Execution :** A = 21H

**After Execution: A=00H**

## 4. Write an ALP to perform logical operations AND, OR AND XOR on two 8-bit numbers stored in internal RAM locations 21H and 22H

```
ORG 00H

    MOV A, 21H            //Operand 1 in address 22H
    ANL A, 22H            //Operand 2 in address 22H
```

```

MOV 30H, A           // AND operation Result in 30H
MOV A, 21H
ORL A, 22H           // OR operation Result in 30H
MOV 31H, A
MOV A, 21H
XRL A, 22H           // XOR operation Result in 30H
MOV 32H, A
HERE: SJMP HERE

END

```

## Result

### Before Execution :

Memory 3	
Address:	I:21h
I:0x21:	75 12 00 00

### After Execution:

Address:	I:30H
I:0x30:	10 77 67 00
I:0x48:	00 00 00 00

**5. Write an ALP to check whether given number is palindrome or not. If palindrome store FFH in accumulator else store 00H in accumulator.**

```

ORG 00H

MOV 30H, #81H        //Number to be verified
MOV R0, 30H
MOV R1, #08H         //8-bits of a byte are to be verified.
MOV 31H, #00H
CLR C

BACK: MOV A, 30H
RLC A                //Rotate left with carry
MOV 30H, A
MOV A, 31H
RRC A                //Rotate right with carry
MOV 31H, A
DJNZ R1, BACK
CJNE A, 00H, NPAL    //Compare data of accumulator and 00h address and
MOV A, #0FFH         //jump to NPAL if they are not equal
SJMP HERE

NPAL: MOV A, #00H
HERE: SJMP HERE

END

```

## Result

### Before Execution :

1. 30H = 81H
2. 30H = 80H

### After Execution:

A = 0FFH  
A = 00H

**6. Write an ALP to count number of positive and negative numbers in 10 bytes of data from 30H to 3AH.**

```
ORG 00H

    MOV R3, #00H           //Count of positive numbers
    MOV R4, #00H           //Count of negative numbers
    MOV R2, #0AH           //10 bytes of data
    MOV R0, #30H           //Initial address of data bytes
    MOV A, @R0
    RLC A                  //Rotate left with carry to check MSB
    INC R0
    JC NEGTV               //If MSB is one then negative number
    INC R3
    SJMP LOOP
NEGTV: INC R4
LOOP: DJNZ R2, L1          //Check all 10 bytes

HERE: SJMP HERE

END
```

**Result**

**Before Execution :**

Memory 2	
Address:	I:30H
I:0x30:	01 54 56 79 9A 11 14 17 90 88 00
I:0x48:	00 00 00 00 00 00 00 00 00 00 00

**After Execution:**

r3	0x07
r4	0x03

**Code Conversions**

**1. Write an ALP to convert a hexadecimal number to decimal number.**

```
ORG 00H

    MOV A, #0FFh           // Any hex number
    MOV B, #100
    DIV AB                  // Divide number by 100d
    MOV R1, A               //Store quotient as it is first digit of decimal number
    MOV A, 0F0H
    MOV B, #10
    DIV AB                  // Divide the remainder by 10d
    SWAP A
    ORL A, B
    MOV R2, A               //R1,R2 has decimal number
HERE: SJMP HERE

End
```

**Result**



**Before Execution : A=0FFH**

**After Execution:**

Register	Value
Regs	
r0	0x00
r1	0x02
r2	0x55

## 2. Write an ALP to convert BCD number to ASCII

ORG 00H

```
MOV A, #49H           //BCD number to be converted to ASCII
MOV R0, A
SWAP A
MOV DPTR, #9000H      //Result in external address
ACALL ASCII
MOV A, R0
ACALL ASCII
HERE: SJMP HERE        //Call ASCII subroutine

ASCII: ANL A, #0FH     //Logical AND operation to mask higher nibble
ADD A, #30H           //Append 3 to higher nibble of BCD number
MOVX @DPTR, A
INC DPTR
RET

END
```

**Result**

**Before Execution : A=0FFH**

**After Execution:**

Memory 2			
Address: X:9000H			
X:0x009000:	34	39	00
X:0x009017:	00	00	00

## 3. Write an ALP to convert ASCII to BCD

ORG 00H

```
MOV DPTR, #9000H      //Number to be converted in external memory
MOVX A, @DPTR
SUBB A, #30H          //Subtract 30 value from content of A
INC DPTR
MOVX @DPTR, A          //Result in 9001H

HERE: SJMP HERE

END
```

**Result**

**Before Execution :**

Memory 2			
Address: X:9000h			
X:0x009000:	39	00	
X:0x009017:	00	00	

**After Execution:**

Memory 2			
Address: X:9000h			
X:0x009000:	39	09	00
X:0x009017:	00	00	00

#### 4. Write an ALP to convert decimal number into binary

```
ORG 00H

    MOV A, #95H           //Number to be converted in external memory
    MOV B, #10H
    DIV AB                //Divide by 10H
    MOV R1, B
    MOV B, #0AH
    MUL AB                //Multiply a and B values and store result in A and B
    ADD A, R1
    MOV 30H, A            //Result in 30H RAM address

HERE: SJMP HERE

END
```

##### Result

**Before Execution : A=95H**

**After Execution:**

Memory 2	
Address:	0:30H
I:0x30:	5F 00 00
I:0x48:	00 00 00

#### Counter Programming

##### 1. Write an ALP to implement (display) binary(hex) up/down counter on watch window

```
ORG 00H

    MOV A, #00H           //MOV A, #0FFH for down counter
BACK: ACALL DELAY         //Call delay subroutine
    INC A                 //Increment accumulator content
    JNZ BACK
HERE: SJMP HERE

DELAY: MOV R1, #0FFH      //Subroutine to create delay
DECR1: MOV R2, #0FFH
DECR:  MOV R3, #0FFH
        DJNZ R3, $
        DJNZ R2, DECR
        DJNZ R1, DECR1

RET

END
```

## Result

### After Execution:

The content of accumulator will be incremented from 00H 0FFH and can be seen in the watch window.

Watch 1		
Name	Value	Type
A	0x13	uchar
<Enter expression>		

## 2. Write an ALP to implement (display) decimal up/down counter on watch window

ORG 00H

```
        MOV A, #00H                                //MOV A, #99H for decimal down counter
BACK:   ACALL DELAY                                //Call delay subroutine
        ADD A, #01H                                //ADD A, #99H for decimal down counter
        DAA                                         //Decimal adjustment after addition
        JNZ BACK
HERE:   SJMP HERE
```

```
DELAY:  MOV R1, #0FFH                                //Subroutine to create delay
DECR1:  MOV R2, #0FFH
DECR:   MOV R3, #0FFH
        DJNZ R3, $
        DJNZ R2, DECR
        DJNZ R1, DECR1
RET
END
```

## Result

### After Execution:

The content of accumulator will be incremented from 00H 0FFH and can be seen in the watch window.

Watch 1	
Name	Value
A	0x12
<Enter expression>	

## Timer Program to generate delay

### 1. Conduct an experiment to generate 1s delay using on chip timer.

ORG 00H

```
        MOV TMOD, #02H                                //Timer 0 mode 2
        MOV TH0, #00H
        CLR 1.0
        CLR A
        SETB TR0                                        //Start Timer 0
AGAIN:  MOV R7, #0FFH                                //1.085X10-6 X14X256X255 = 1sec delay for 11.0592MHz
LOOP:   MOV R6, #14D
```

```

WAIT:  JNB TF0, WAIT          //Jump if bit condition is zero
        CLR TF0
        DJNZ R6, WAIT
        DJNZ R7, LOOP
        CPL P1.0

```

```

SJMP AGAIN

```

```

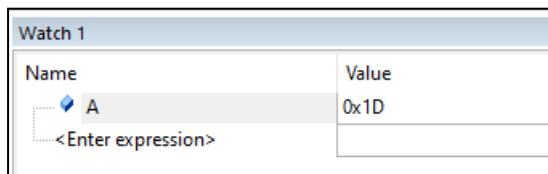
END

```

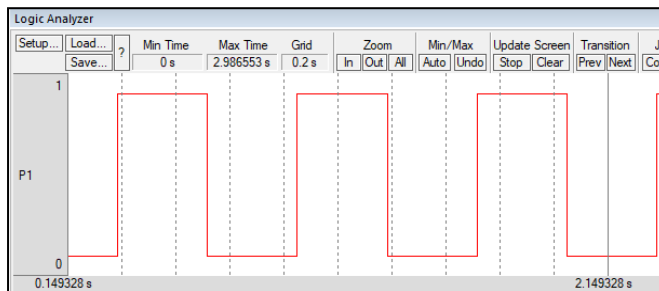
## Result

### After Execution:

Accumulator is incremented in binary FROM 00,01,02,03,..... 09,0A,0B, .....0F,10,.....FF every one second (For 33MHz clock setting and every 3 seconds for 11.0592MHz)



Measure the time delay and frequency of signal generated through logic analyzer window.



## Serial Communication Programming

**1. Write an ALP to transfer your Name and USN burnt in program memory serially out at 9600 baud rate.**

```

ORG 00H

```

```

        MOV TMOD, #20H          //Timer 0 mode 2
        MOV SCON, #70H
        MOV TH0, #-3            //9600 baud rate
        SETB TR1                //Start timer 1
AGAIN:  MOV R0, #11H
        MOV DPTR, #0050H        //Message starts from program memory address 50H
NXTCHAR: CLR A
        MOVC A, @A+DPTR         //Copy from code memory
        INC DPTR
        ACALL TRANSFER
        DJNZ R0, NXTCHAR
        SJMP AGAIN
TRANSFER: MOV SBUF, A           //Transfer of data byte starts.
WAIT:   JNB TI, WAIT
        CLR TI

```

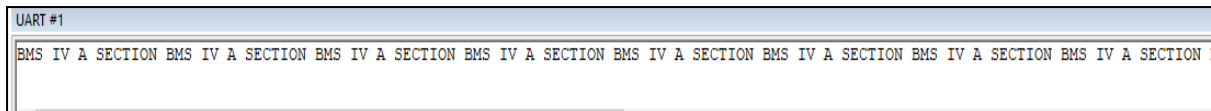
RET

```
ORG 50H //Message burnt into program memory 50H
DB "BMS IV A SECTION " //Define byte directive
```

END

## Result

**After Execution:** Each time the program is executed, "BMS IV A SECTION " will be displayed on the serial window.



## Baud rate calculation

$$\text{Crystal Freq}/(12 \times 32) = 11.0592\text{MHz}/(12 \times 32) = 28800$$

Serial communication circuitry divides the machine cycle frequency by 32 before it is used by the timer to set the baud rate by timer 1.

To get 9600 baud rate  $9600 = 28000/3$ , hence -3 (FFH-3 = FDH) has to be loaded into TH1 register

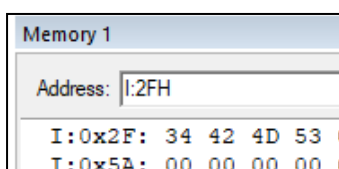
## 2. Write an ALP to Receive character serially from keyboard.

ORG 00H

```
MOV TMOD, #20H //Timer 1 in mode 2
MOV TH1, #-3 //9600 baud rate
MOV SCON, #50H //9 bit data transfer with one start and stop bit
SETB TR1
MOV R0, #2FH //Received character in 2F H address
HERE: JNB RI, HERE
MOV A, SBUF
MOV @R0, A
INC R0
CLR RI
SJMP HERE

END
```

**Result:** Run the program continuously, keep the cursor on UART window and type any character. The respective ASCII value will be displayed from 2FH RAM location onwards. When 4BMS was typed by placing cursor on UART window for each character following result was shown from 2FH address.



## **PART B: Interfacing Programs**

### **Experiment: 1 – Program to Interface 7 SEGMENT DISPLAY KIT**

Connect the interface over 26PIN connector of ESA MCB 51 board \*/

```
#include<stdio.h>
#include <REG51.H>
void delay(int g);
int port[20] = {0xff,0xff,0xff,0xff,
                0xc6,0x86,0xc7,0x86,
                0xbf,0xc0,0xde,0x87,
                0xbf,0x92,0x91,0x92,
                0x92,0xc8,0x86,0x87},i;
void main(){
    int d,b,j,k,s;
    while(1){
        i=0;
        for(d=0;d<5;d++){
            for(b=0;b<4;b++){
                k=port[i++];
                for(j=0;j<8;j++){
                    s=k;
                    s=s&0x80;
                    if(s==0x00)
                        P1= 0x00;
                    else
                        P1=0x1;
                    P2 = 0x1;
                    P2=0x00;
                    s=k;
```

```

        s=s<<1;
        k=s;
    }

}

delay(10000);
delay(10000);
    }
}

}

void delay(int g)
{
    int h;
    for(h=0;h<=g;g++)
    {
        ;}}

```

## Result

Try Display :DEPT OF ECE

## **Experiment: 2– Program to Interface STEPPER MOTOR**

Connection: Connect the stepper motor interface to 26 pin FRC connector J7 of ESAMCB51.

Output: Rotate motor in clockwise/anticlockwise with INT0

DO : speed control with INT1.

If you want to change the direction press the interrupt button ( P3.2/INT0\*). when you press the button, INT0 interrupts the main program and changes the direction of the motor .

```

#include <REG51xD2.H>
static bit Dir=0;

```

```

void delay(unsigned int x)                /* Delay Routine */
{
    for(;x>0;x--);
}

```

```

void ChangeDir(void) interrupt 0          /* Int Vector at 000BH, Reg Bank 1 */
{
    Dir = ~Dir;                          /* Complement the Direction flag */
    if(Dir)

```

```

    {    delay(32000);
    }
else
    {    delay(32000);
    }
}
main()
{
    unsigned char Val,i;
    EA=0x1;                                /* Enable Interrupt flag ,Interrupt 0 & Serial Interrupt
*/
    EX0=0x1;
    ES=0x1;
    P0=0x00;                                /* monitor is using the serial interrupt –enable it*/
    while(1)
    {
        if(Dir)                            /* If Dir Clockwise */
        {
            Val = 0x88;
            for(i=0;i<4;i++)
            {
                P0 = Val;                    /* Write data for clock wise direction*/
                Val = Val>>1;
                delay(575);
            }
        }
        else                                /* AntiClockwise Direction */
        {
            Val = 0x11;
            for(i=0;i<4;i++)
            {
                P0 = Val;                    /* Write data for anticlock wise direction*/
                Val = Val<<1;
                delay(575);
            }
        }
    }
}

```

**Try: Change the speed of Motor using INT1**

## **Experiment: 3– Program to Interface DAC KIT**

### **P3.1. Generate the Square wave**

Connection: Connect the interface module to 26pin FRC connector J7 of ESAMCB51.

Output : **Generate the square wave with 2.5v amplitude.**



You can change Amplitude by pressing P3.3/INT1 on ESAMCB51. Frequency by P3.2/INT0

```
#include <REG51xD2.H>
```

```
#include "lcd.h"
```

```
sbit Amp = P3^3;          /* Port line to change amplitude */
```

```
sbit Fre = P3^2;          /* Port line to change frequency */
```

```
void delay(unsigned int x) /* delay routine */
```

```
{
```

```
    for(;x>0;x--);
```

```
}
```

```
main()
```

```
{
```

```
    unsigned char on = 0x7f, off=0x00;
```

```
    unsigned int fre = 100;
```

```
    InitLcd();                /* Initialize LCD */
```

```
    WriteString("Squarewave"); /* Write to LCD */
```

```
    while(1)
```

```
{
```

```
    if(!Amp)                  /* if user choice is to change amplitude */
```

```
{
```

```
        while(!Amp);          /* wait for key release */
```

```
        on+=0x08;              /* Increase the amplitude */
```

```
}
```

```
    if(!Fre)                  /* if user choice is to change frequency */
```

```
{
```

```
        if(fre > 1000)          /* if frequency exceeds 1000 reset to default */
```

```
            fre = 100;
```

```
            while(!Fre);        /* wait for key release */
```

```
            fre += 50;           /* Increase the frequency */
```

```
}
```

```
    P0=on;                    /* write amplitude to port */
```

```
    P1=on;
```

```
    delay(fre);
```

```
    P0 = off;                 /* clear port */
```

```
    P1 = off;
```

```

delay(fre);

}

}

```

### **P3.2. Generate the Triangular wave**

```

#include <REG51xD2.H>
#include "lcd.h"

main()
{
    unsigned char i=0;
    InitLcd();                                /* Initialise LCD */
    WriteString("Triangular Wave");           /* Display on LCD */
    P0 = 0x00;                                /* P0 as Output port */
    while(1)
    {
        for(i=0;i<0xff;i++){                  /* Generate ON pulse */
            P1 = i;

        P0 = i; }
        for(i=0xfe;i>0x00;i--)                 /* Generate OFF pulse */
        {
            P0 = i;
            P1 = i;}
    }
}

```

### **P3.3 Generate the Sine wave**

```

#include<reg51.h>

#include<math.h>    /* Port line to change frequency */

void delay(unsigned int x)    /* delay routine */
{
    for(;x>0;x--);
}

```

```

void main()
{
    unsigned char value[72];
    unsigned int i=0,q=0;
    unsigned char x;

    for(x=0;x<36;x++)
    {
        value[x]=(128*sin(i*3.14/180))+128;
        i+=5;
    }
    for(x=36;x<72;x++)
    {
        value[x]=128*sin(i*3.14/180)-128;
        i+=5;
    }

```

```

while(1)
{

    for(x=0;x<72;x++)
    {
        P0=value[x];
        P1=value[x];
        delay(5);
    }
}

```

### **Experiment: 4 – Program to Interface Elevator KIT**

Connection : Connect the interface to 26 pin FRC connector J7 of ESAMCB51

Output : **move Elevator for requested floor and halt**

```

#include <REG51.H>
void delay(unsigned int);
main()
{
    unsigned char Flr[9] = {0xff,0x00,0x03,0xff,0x06,0xff,0xff,0xff,0x09};
    unsigned char FCflr[9] = {0xff,0x0E0,0x0D3,0xff,0x0B6,0xff,0xff,0xff,0x79};
    unsigned char ReqFlr, CurFlr = 0x01,i,j;
    P0 = 0x00;

```

```

P0 = 0x0f0;
while(1)
{
    P1 = 0x0f;
    ReqFlr = P1 | 0x0f0;
    while(ReqFlr == 0x0ff)
        ReqFlr = P1 | 0x0f0;          /* Read Request Floor from P1 */
    ReqFlr = ~ReqFlr;
    if(CurFlr == ReqFlr)              /* If Request floor is equal to Current Floor */
    {
        P0 = FClr[CurFlr];          /* Clear Floor Indicator */
        continue;                    /* Go up to read again */
    }
    else if(CurFlr > ReqFlr)           /* If Current floor is > request floor */
    {
        i = Flr[CurFlr] - Flr[ReqFlr]; /* Get the no of floors to travel */
        j = Flr[CurFlr];
        for(;i>0;i--)                 /* Move the indicator down */
        {
            P0 = 0x0f0|j;
            j--;
            delay(25000);
        }
    }
    else                             /* If Current floor is < request floor */
    {
        i = Flr[ReqFlr] - Flr[CurFlr]; /* Get the no of floors to travel */
        j = Flr[CurFlr];
        for(;i>0;i--)                 /* Move the indicator Up */
        {
            P0 = 0x0f0 | j;
            j++;
            delay(25000);
        }
    }
    CurFlr = ReqFlr;                  /* Update Current floor */
    P0 = FClr[CurFlr];              /* Clear the indicator */
}
}

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

```

## **Experiment: 5– Program to Interface Keyboard KIT**

Connection: Connect the interface module to 26 pin FRC connector J7 of ESAMCB51.

**Output : Display the pressed key's key code on the On-board LCD**

USE : LCD library to write on the LCD. Include lcd.h in your project file which has all LCD routines.

```
#include <REG51xD2.H>
#include "lcd.h"
unsigned char getkey();
void delay(unsigned int);
main()
{
    unsigned char key;
    InitLcd();                                /* Initialise LCD */
    WriteString("Key Pressed=");              /* Display msg on LCD */
    while(1)
    {
        GotoXY(12,0);                          /* Set Cursor Position */
        key = getkey();                          /* Call Getkey method */
    }
}

unsigned char getkey()
{
    unsigned char i,j,k,indx,t;
    P0=0x0ff;
    P1=0x0ff;
    P2 = 0x00;                                /* P2 as Output port */
    indx = 0x00;                              /* Index for storing the first value scanline */
    for(i=0x00E;i>=0x00B;i<=1)              /* for 4 scanlines */
    {
        P2 = i;                                /* write data to scanline */
        t = P0;                                /* Read readlines connected to P0 */
        t = ~t;
        if(t>0)                                /* If key press is true */
        {
            delay(6000);                        /* Delay for bouncing */
            for(j=0;j<=7;j++)                  /* Check for 8 lines */
            {
                t >>=1;
                if(t==0)                        /* if get pressed key */
                {
                    k = indx+j;                /* Display that by converting to Ascii */
                    t = k>>4;
                    t +=0x30;
                }
            }
        }
    }
}
```

```

WriteChar(t);                                /* Write upper nibble */
    t = k & 0x0f;
    if(t > 9)
        t+=0x37;
    else
        t+=0x30;
    WriteChar(t);                            /* write lower nibble */
    return(indx+j);                          /* Return index of the key pressed */
}
}
}
indx += 8;                                  /* If no key pressed increment index */
}
}

void delay(unsigned int x)                   /* Delay routine */
{
    for(;x>0;x--);
}

```