JSS MAHAVIDYAPEETHA

JSS SCIENCE AND TECHNOLOGY UNIVERSITY

SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

- Constituent College of JSS Science and Technology University
- Approved by A.I.C.T.E
- Governed by the Grant-in-Aid Rules of Government of Karnataka
- Identified as lead institution for World Bank Assistance under TEQIP Scheme

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Microcontrollers and Embedded Systems Lab (20EC48L)

IV SEM (A and B Section)

2021-2022

Lab location: AB208

Faculty in Charge

Dr. Shankaraiah

Prof. Renuka B S

Vision statement of the JSS Science and Technology University

- **Advancing JSS S&T University as a leader in education, research and technology on the International arena.**
- **To provide the students a universal platform to launch their careers, vesting the industry and research community with skilled and professional workforce.**
- **Accomplishing JSS S&T University as an epicenter for innovation, centre of excellence for research with state of the art lab facilities.**
- **Fostering an erudite, professional forum for researchers and industrialist to coexist and to work cohesively for the growth and development of science and technology for betterment of society.**

Mission statement of the JSS Science and Technology University

1. Education, research and social outreach are the core doctrines of JSS S&T University that are responsible for accomplishment of in-depth knowledge base, professional skill and innovative technologies required to improve the socio economic conditions of the country.

2. Our mission is to develop JSS S&T University as a global destination for cohesive learning of engineering, science and management which are strongly supported with interdisciplinary research and academia.

3. JSS S&T University is committed to provide world class amenities, infrastructural and technical support to the students, staff, researchers and industrial partners to promote and protect innovations and technologies through patents and to enrich entrepreneurial endeavors.

4. JSS S&T University core mission is to create knowledge led economy through appropriate technologies, and to resolve societal problems by educational empowerment and ethics for better living.

JSS MAHAVIDYAPEETHA
JSS SCIENCE AND TECHNOLOGY UNIVERSITY
SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

- **Constituent College of JSS Science and Technology University**
- **Approved by A.I.C.T.E**
- **Governed by the Grant-in-Aid Rules of Government of Karnataka**
- **Identified as lead institution for World Bank Assistance under TEQIP Scheme**

Vision statement of the department of E&CE

Be a leader in providing globally acceptable education in electronics and communication engineering with emphasis on fundamentals-to-applications, creative-thinking, research and career-building.

Mission statement of the department of E&CE

1. To provide best infrastructure and up-to-date curriculum with a conducive learning environment.

2. To enable students to keep pace with emerging trends in Electronics and Communication Engineering

3. To establish strong industry participation and encourage student entrepreneurship.

4. To promote socially relevant eco-friendly technologies and inculcate inclusive innovation activities.

Program Outcomes (POs)

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

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

3. **Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate**

consideration for public health and safety, cultural, societal and environmental considerations.

4. Conduct investigations of complex problems: Using research based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.

5. 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

6. The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.

7. Environment and Sustainability: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

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

9. Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings

10. 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.

11. Lifelong Learning: Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

12. Project Management and Finance: Demonstrate knowledge and understanding of 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.

Program Specific Outcomes (PSOs)

1. Analyze, design and provide engineering solutions in the areas of electronic circuits and systems.

2. Demonstrate the mathematical modeling techniques, nurture analytical and computational skills to provide engineering solutions in the areas of electronics and communication.

3. Ability to address multidisciplinary research challenges and nurture entrepreneurship

Program Educational Objectives (PEOs)

1. To enable the graduates to have strong Engineering fundamentals in Electronics & Communication, with adequate orientation to mathematics and basic sciences.

2. To empower graduates to formulate, analyze, design and provide innovative solutions in Electronics & Communication, for real life problems.

3. To ensure that graduates have adequate exposure to research and emerging technologies through industry interaction and to inculcate professional and ethical values.

4. To nurture required skill sets to enable graduates to pursue successful professional career in industry, higher education, competitive exams and entrepreneurship.

JSS MAHAVIDYAPEETHA
JSS SCIENCE AND TECHNOLOGY UNIVERSITY
SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

- Constituent College of JSS Science and Technology University
- Approved by A.I.C.T.E
- Governed by the Grant-in-Aid Rules of Government of Karnataka
- Identified as lead institution for World Bank Assistance under TEQIP Scheme

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

RECORD OF CIE FOR PERFORMANCE IN THE LAB CLASSES

Evaluation Sheet

Section		**Batch**		**Group No**	
Staff in Charge		**Day**		**Timings**	

Sl. No.	**USN**	**Name of the Students**		
			AC1: Preparedness (8M) **AC2: Conduction (8M)** **AC3: Viva (8M)** **AC4: Report Writing (8M)** **AC5: Result Interpretation (8M)** **T: Total (40M)**	**SUBJECT: Microcontrollers and Embedded systems Lab** **CODE : 20EC48L**
1.				
2.				
3.				
4.				

Sl. No	**Date**	**Experiments**	**Student-1**						**Student-2**						**Student-3**						**Student-4**					
			AC1	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**
1.																										
2.																										

3.																										
4.																										
5.																										

Sl. No.	**Date**	**Experiments**	**Student-1**						**Student-2**						**Student-3**						**Student-4**					
			AC1	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**T**
6.																										
7.																										
8.																										
9.																										
10.																										
11.																										
12.																										
Average marks from experiments 1 to 10 (40 Marks)																										
Lab Test =(10Marks)																										
Total CIE (50 MARKS)																										
Percentage of Attendance																										

Signature of the Staff in Charge: **Signature of the Lab in Charge:**

1. **Dr.Shankaraiah**
2. **Prof. Renuka B S**

Course Outcomes:

After completing this course, students should be able to:

CO1:	**Understand Assembly Language/embedded C programming of Microcontroller.**
CO2:	**Understand interfacing simple peripheral devices to Microcontroller.**
CO3:	**Engage student groups to design and implement simple embedded systems.**

Course Articulation matrix

Course Outcomes	**Program Outcomes**												**PSO's**		
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**P012**	**PSO1**	**PSO2**	**PSO3**
CO1	**3**	**3**	**3**		**3**								**3**		
CO2	**3**	**3**	**3**		**3**								**3**		
CO3	**3**	**3**	**3**		**3**								**3**	**3**	

LAB EXPERIMENTS:

Software programs: To be implemented on 8051 microcontroller
1. Problems related with data transfer and exchange.
2. Problems related with arithmetic and logical operations.
3. Problems related with programming timers in all modes with and without interrupts.
4. Problems related with programming serial communication with and without interrupts.
5. Program related with handling external interrupts.
Hardware programs: To be implemented on 8051 and ARM CORTEX-M3 (using Embedded C)
1. Interface LCD.
2. Interfacing of matrix keypad.
3. Interfacing of ADC and DAC.
4. Interfacing of multi digit 7 segment displays.
5. Interfacing of stepper motor and D C motor.

MICROCONTROLLER LAB
Set of experiments
SET I

1. **Write an 8051 assembly level program to add 10 bytes of data.**
2. **Write an 8051 assembly level program to transfer 10 bytes of data from external RAM location starting with 2000h to internal RAM starting from 30h.**
3. **Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 30h to location 35h.**
4. **Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 35h to location 30h.**
5. **Write an 8051 assembly level program to exchange 10 bytes of data from location starting at 30h with data from location starting from 1000h.**
6. **Write an 8051 assembly level program to transfer 10 bytes of data starting from location 8000h to location 9000h within the external memory.**

SET II

1. **Write an 8051 assembly level program to add 'N' bytes of data taking into account the possible carry output.**
2. **Write an 8051 assembly level program to add 'N' bytes of BCD numbers taking into account the possible carry output.**
3. **Write an 8051 assembly level program to find the average of 'N' bytes of data.**
4. **Write an 8051 assembly level program to subtract two BCD numbers.**
5. **Write an 8051 assembly level program to add two multi-byte numbers.**

SET III

1. **Write an 8051 assembly level program to count the number of even numbers and number of odd numbers in an array of 'N' bytes of data.**
2. **Write an 8051 assembly level program to count the number of +ve numbers and number of -ve numbers in an array of 'N' bytes of data.**
3. **Check whether the given byte of data is present in an array of 'N' bytes of data. If present send 00 in Port 0 else send FF in Port 0.**
4. **Read the data from Port 1. If P1.1 is at logic 0, find the largest number in an array of 'N' bytes of data and store in location 40h. If P1.0 is at logic 1, find the smallest number in the array and store in the location 40h.**

SET IV

1. **Write an 8051 assembly level program to arrange an array of 'N' bytes of data in ascending order.**
2. **Write an 8051 assembly level program to arrange an array of 'N' bytes of data in descending order.**
3. **Write an 8051 assembly level program to find whether the given number is prime or not. If prime send FF to Port 0 else send 00 to Port 0.**
4. **Write an 8051 assembly level program to find the factorial of a given number (using recursive procedure).**
5. **Write an 8051 assembly level program for BCD up counter. Show each count in Port 0 with appropriate delay.**
6. **Write an 8051 assembly level program for BCD down counter. Show each count in Port 0 with appropriate delay.**

SET V

1. **Write an 8051 assembly level program to check whether the given byte of data is palindrome. If 'yes' send 00 to Port 0 else send FF to Port 0.**
2. **Write an 8051 assembly level program to check whether the lower nibble is greater than higher nibble of A. If 'yes' send 00 to Port 0 else send FF to Port 0.**
3. **Write an 8051 assembly level program to convert 2 digit BCD to ASCII numbers and store them in location 30h(LSB) and 31h(MSB).**
4. **Write an 8051 assembly level program to find the square of a number using look up table technique.**
5. **Write an 8051 assembly level program to find the square root of a number.**

SET VI

1. **Write an 8051 assembly level program to find LCM and HCF of two numbers.**
2. **Write an 8051 assembly level program to check whether the given number is 2 out of 5 code. If 'yes' send 00 to Port 0 else send FF to Port 0.**
3. **Write an 8051 assembly level program to generate Fibonacci series.**

SET VII

1. **Write an 8051 assembly level program to generate square wave on P1.5 with 50% duty cycle. Program timers in mode 0, mode1 and mode2 to generate the delay.**
2. **Write an 8051 assembly level program to generate square wave with ON period of 20ms and OFF period of 40ms. Use timers in mode 0, mode 1 and mode 2 to generate the delay.**
3. **Repeat the problems with interrupt for timers.**
4. **Repeat the above problems by writing programs in embedded C.**

SET VIII

1. **A switch is connected to P2.5. write an 8051 assembly level program to read the status of switch and if switch is closed send serially 'HELLO', else send 'WELCOME' at baud rate 9600.**
2. **Write an 8051 assembly level program to transfer a message SJCE serially by programming serial communication in interrupt mode with baud rate 9600.**
3. **Repeat the problem 1 & 2 using embedded C program.**

SET IX

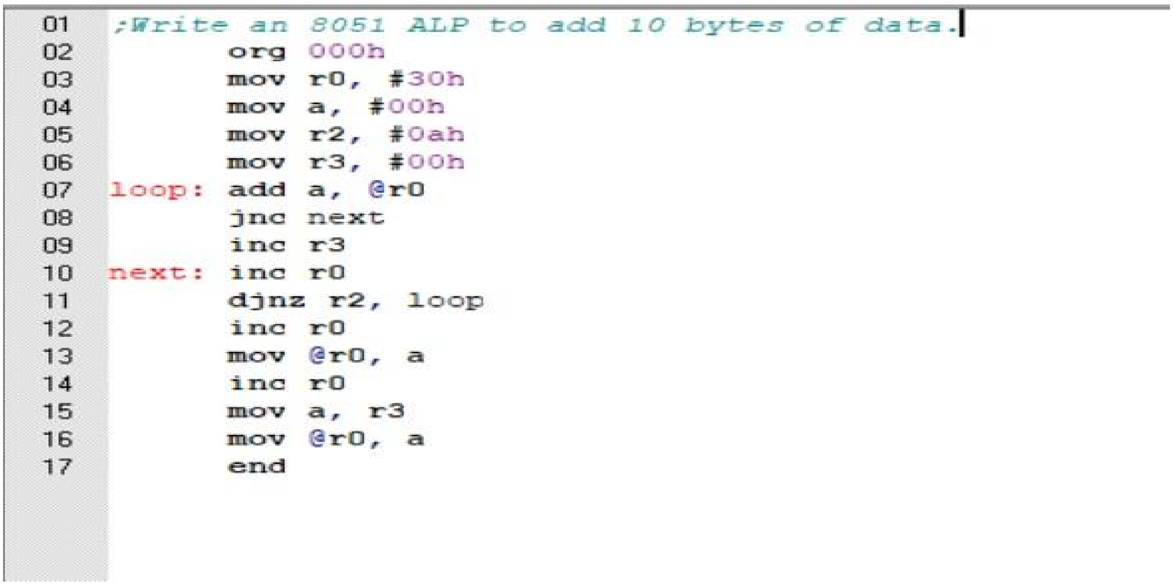
1. **Interface 4 digit multiplexed 7 segments LED and writes a program to display the message SJCE.**
2. **Interface 4x4 hex keyboards and write a program to read the key closure and display hex code for key pressed on 7 segment display.**

SET X

1. **Interface LCD module and write program to display a message.**
2. **Interface ADC a write a program to sample the signal and convert it into digital**
3. **Interface DAC and write program to generate various waveform.**

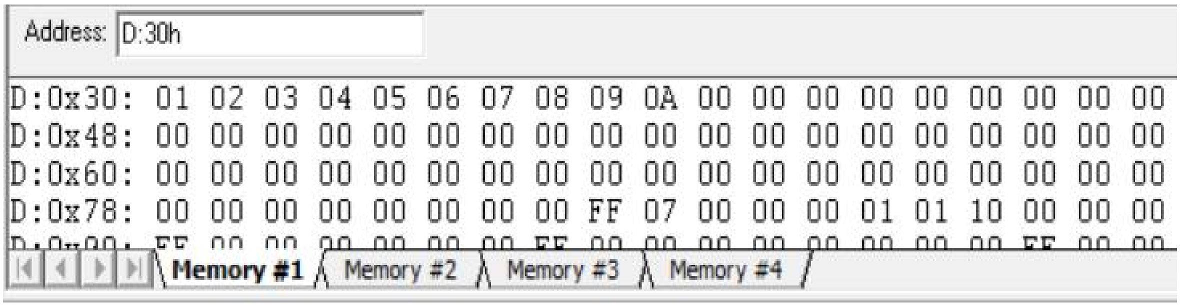
SET 1 PROGRAMS

1.1 write an 8051 alp to add 10 bytes of data



```
;Write an 8051 ALP to add 10 bytes of data.
        org 000h
        mov r0, #30h
        mov a, #00h
        mov r2, #0ah
        mov r3, #00h
loop:   add a, @r0
        jnc next
        inc r3
next:   inc r0
        djnz r2, loop
        inc r0
        mov @r0, a
        inc r0
        mov a, r3
        mov @r0, a
        end
```

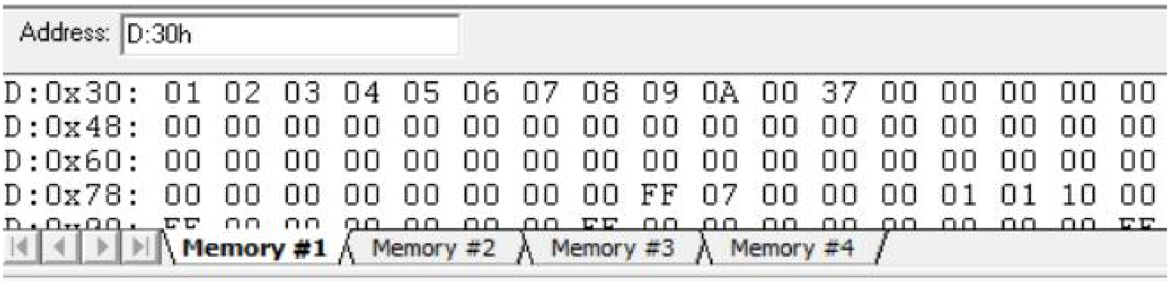
Before Execution:



Address: D:30h

```
D:0x30: 01 02 03 04 05 06 07 08 09 0A 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00
```

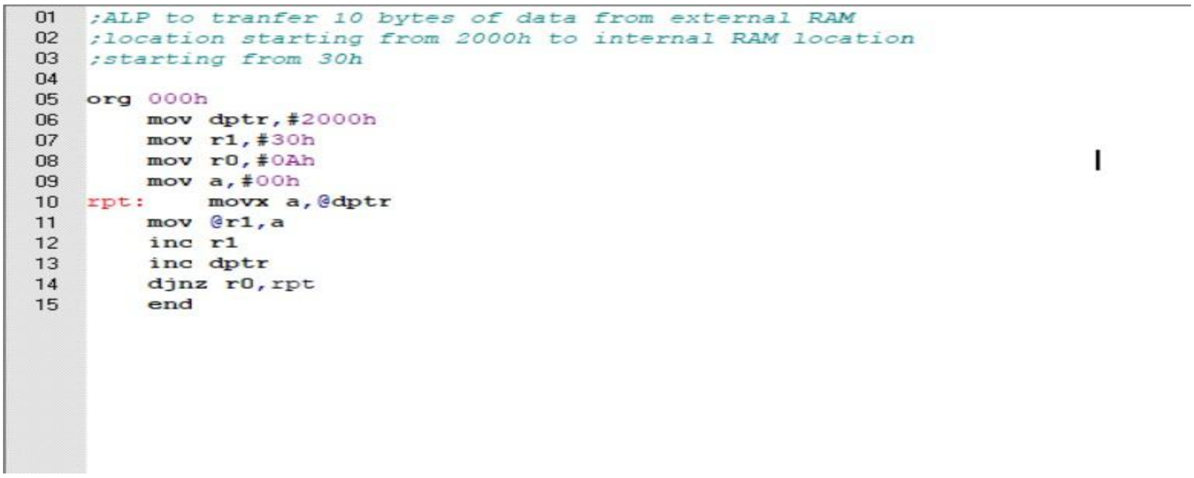
After Execution:



Address: D:30h

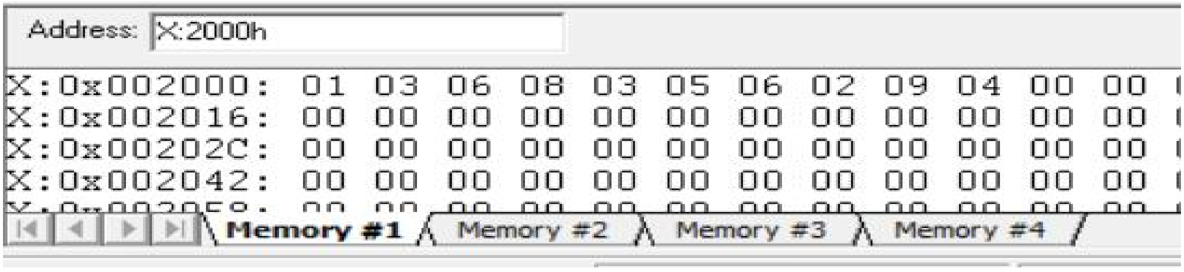
```
D:0x30: 01 02 03 04 05 06 07 08 09 0A 00 37 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00
```

1.2 Write an 8051 assembly level program to transfer 10 bytes of data from external RAM location starting with 2000h to internal RAM starting from 30h



```
;ALP to tranfer 10 bytes of data from external RAM
;location starting from 2000h to internal RAM location
;starting from 30h

org 000h
      mov dptr,#2000h
      mov r1,#30h
      mov r0,#0Ah
      mov a,#00h
rpt:        movx a,@dptr
      mov @r1,a
      inc r1
      inc dptr
      djnz r0,rpt
      end
```

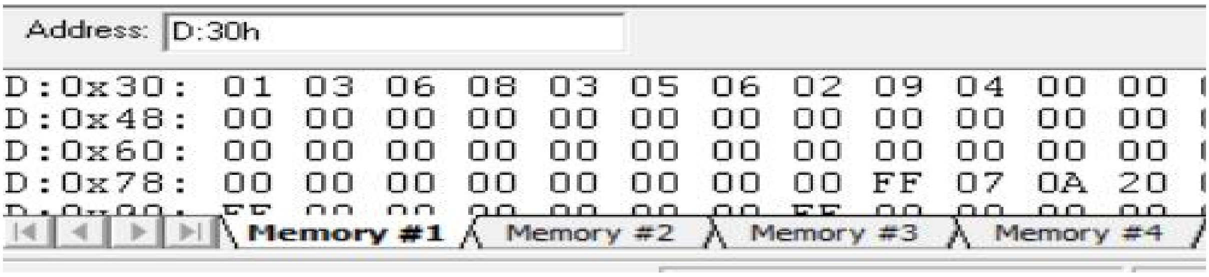
Before Execution:



Address: X:2000h

```
X:0x002000: 01 03 06 08 03 05 06 02 09 04 00 00
X:0x002016: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00202C: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x002042: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x002058: 00 00 00 00 00 00 00 00 00 00 00 00
```

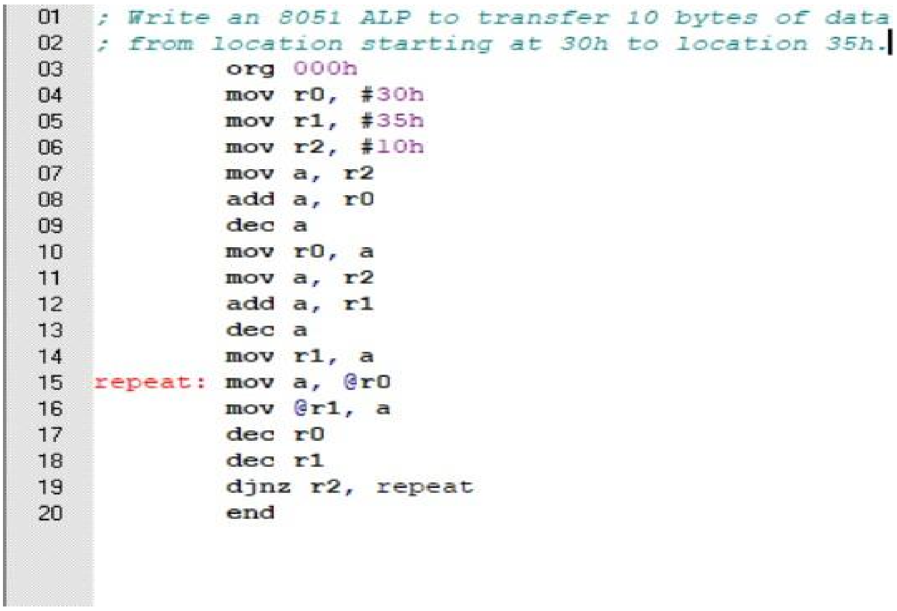
After Execution:



Address: D:30h

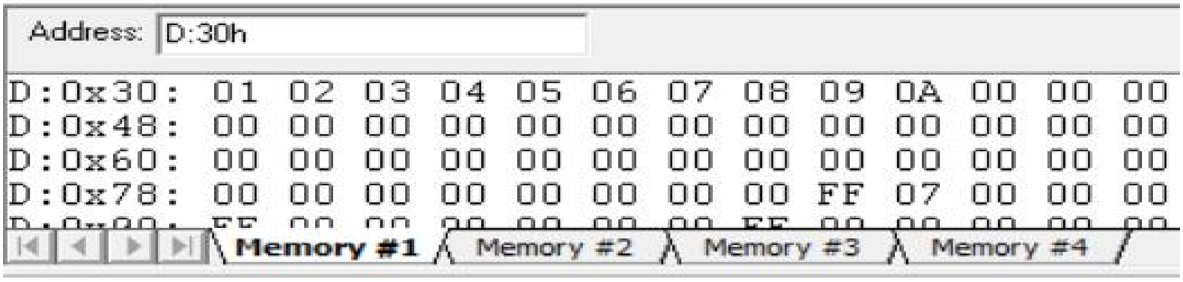
```
D:0x30: 01 03 06 08 03 05 06 02 09 04 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 0A 20
D:0x90: FF 00 00 00 00 00 00 FF 00 00 00 00
```

1.3 Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 30h to location 35h.



```
; Write an 8051 ALP to transfer 10 bytes of data
; from location starting at 30h to location 35h.
        org 000h
        mov r0, #30h
        mov r1, #35h
        mov r2, #10h
        mov a, r2
        add a, r0
        dec a
        mov r0, a
        mov a, r2
        add a, r1
        dec a
        mov r1, a
repeat: mov a, @r0
        mov @r1, a
        dec r0
        dec r1
        djnz r2, repeat
        end
```

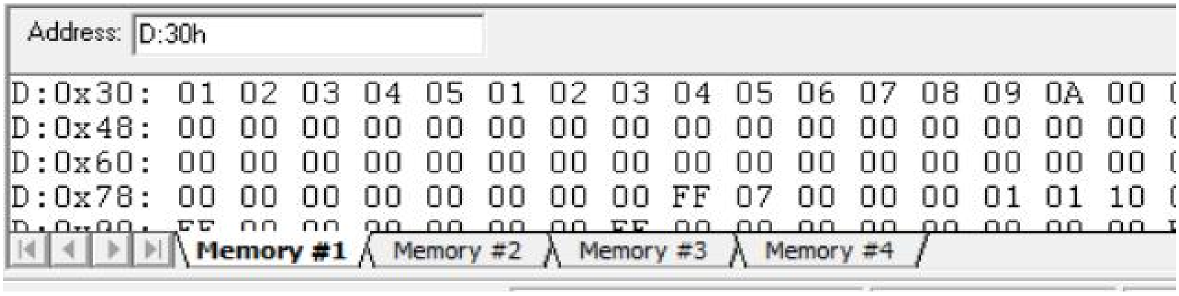
Before Execution :



Address: D:30h

```
D:0x30: 01 02 03 04 05 06 07 08 09 0A 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00
D:0x90: FF 00 00 00 00 00 00 FF 00 00 00 00 00
```

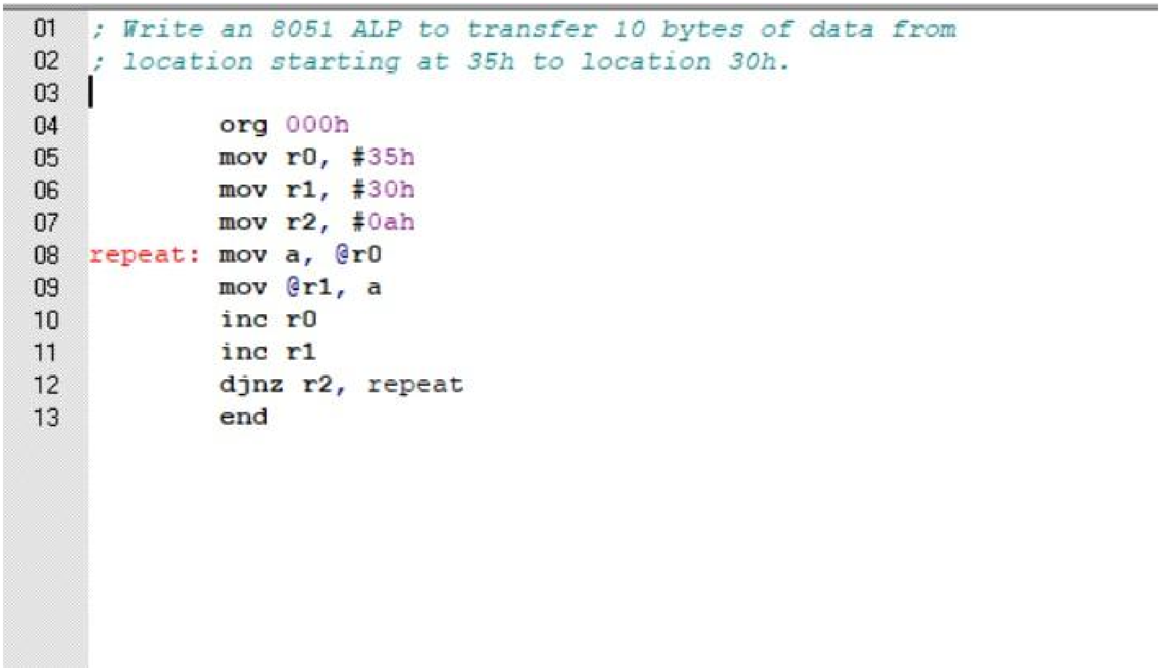
After Execution:



Address: D:30h

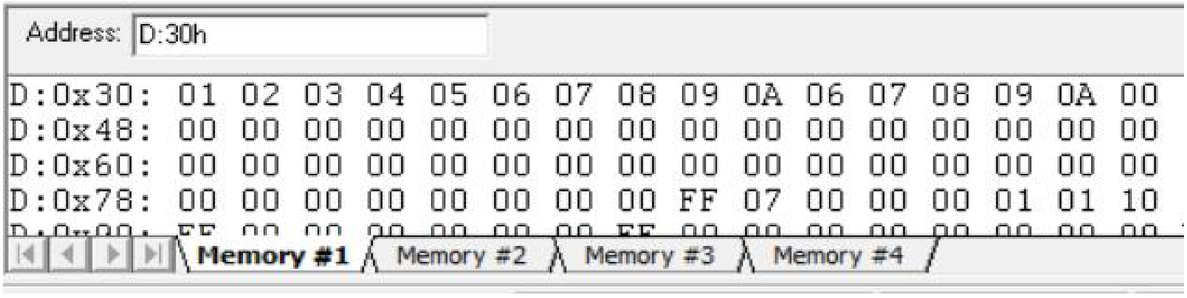
```
D:0x30: 01 02 03 04 05 01 02 03 04 05 06 07 08 09 0A 00 (
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 (
D:0x90: FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 I
```

1.4 Write an 8051 assembly level program to transfer 10 bytes of data from location starting at 35h to location 30h.



```
; Write an 8051 ALP to transfer 10 bytes of data from
; location starting at 35h to location 30h.

        org 000h
        mov r0, #35h
        mov r1, #30h
        mov r2, #0ah
repeat: mov a, @r0
        mov @r1, a
        inc r0
        inc r1
        djnz r2, repeat
        end
```

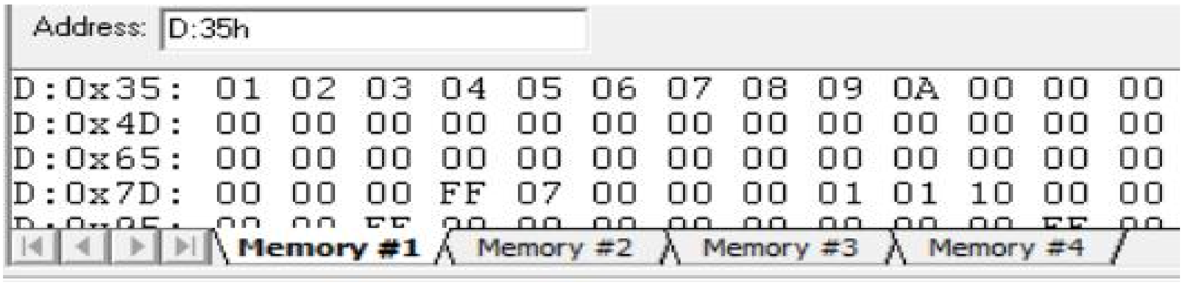
Before Execution:



Address: D:30h

```
D:0x30: 01 02 03 04 05 06 07 08 09 0A 06 07 08 09 0A 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10
```

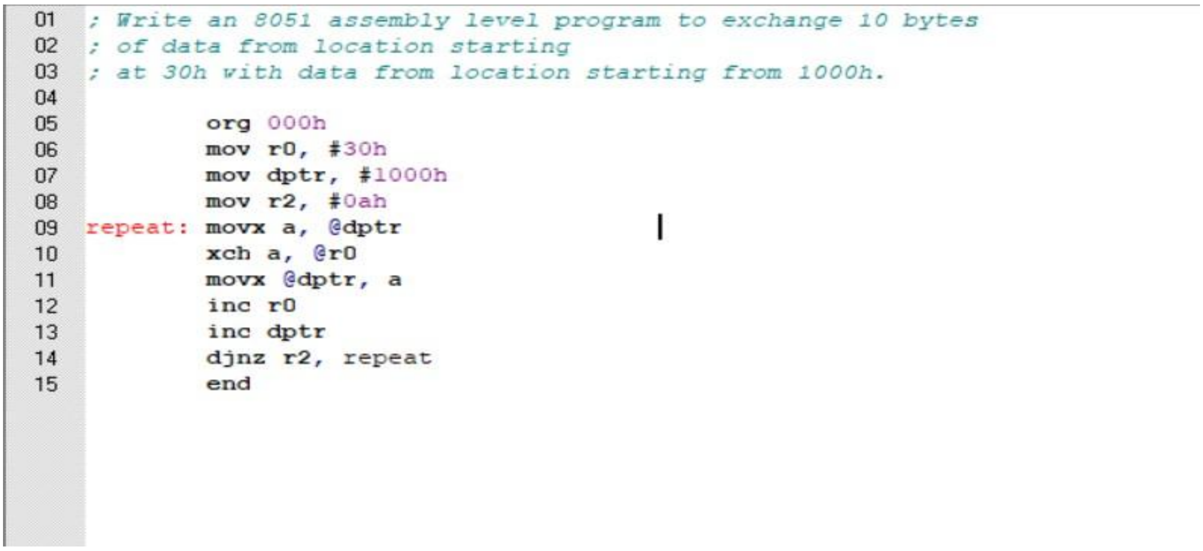
After execution:



Address: D:35h

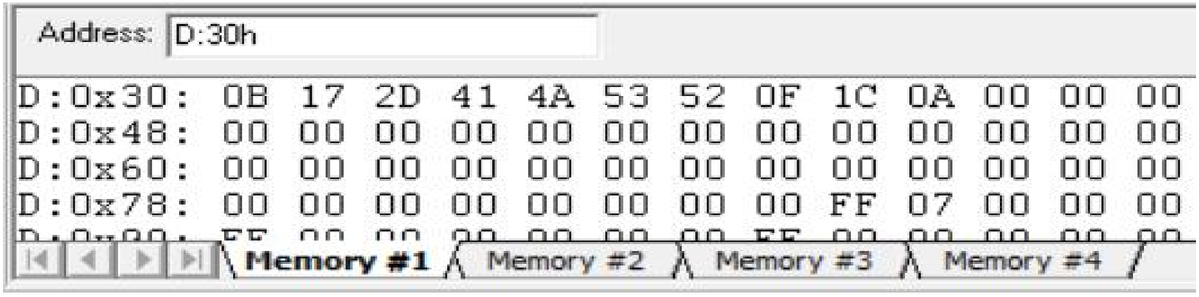
```
D:0x35: 01 02 03 04 05 06 07 08 09 0A 00 00 00
D:0x4D: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x65: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x7D: 00 00 00 FF 07 00 00 00 01 01 10 00 00
```

1.5 Write an 8051 assembly level program to exchange 10 bytes of data from location starting at 30h with data from location starting from 1000h.



```
; Write an 8051 assembly level program to exchange 10 bytes
; of data from location starting
; at 30h with data from location starting from 1000h.

        org 000h
        mov r0, #30h
        mov dptr, #1000h
        mov r2, #0ah
repeat: movx a, @dptr
        xch a, @r0
        movx @dptr, a
        inc r0
        inc dptr
        djnz r2, repeat
        end
```

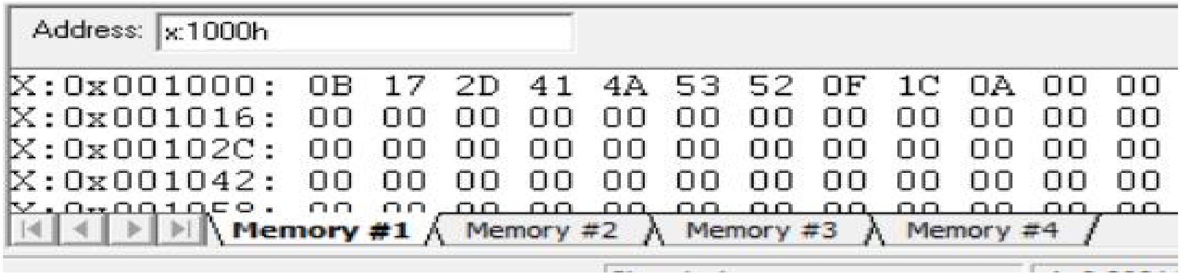
Before Execution
1000h:0BH,17H,2DH,41,4AH,53,52H,0FH,1CH,0AH
30H: 01H,02H,3H,4H,5H,6H,7H,8H,9H,0AH



Address: D:30h

```
D:0x30:  0B  17  2D  41  4A  53  52  0F  1C  0A  00  00  00
D:0x48:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x60:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x78:  00  00  00  00  00  00  00  00  FF  07  00  00  00
D:0x90:  FF  00  00  00  00  00  00  FF  00  00  00  00  00
```

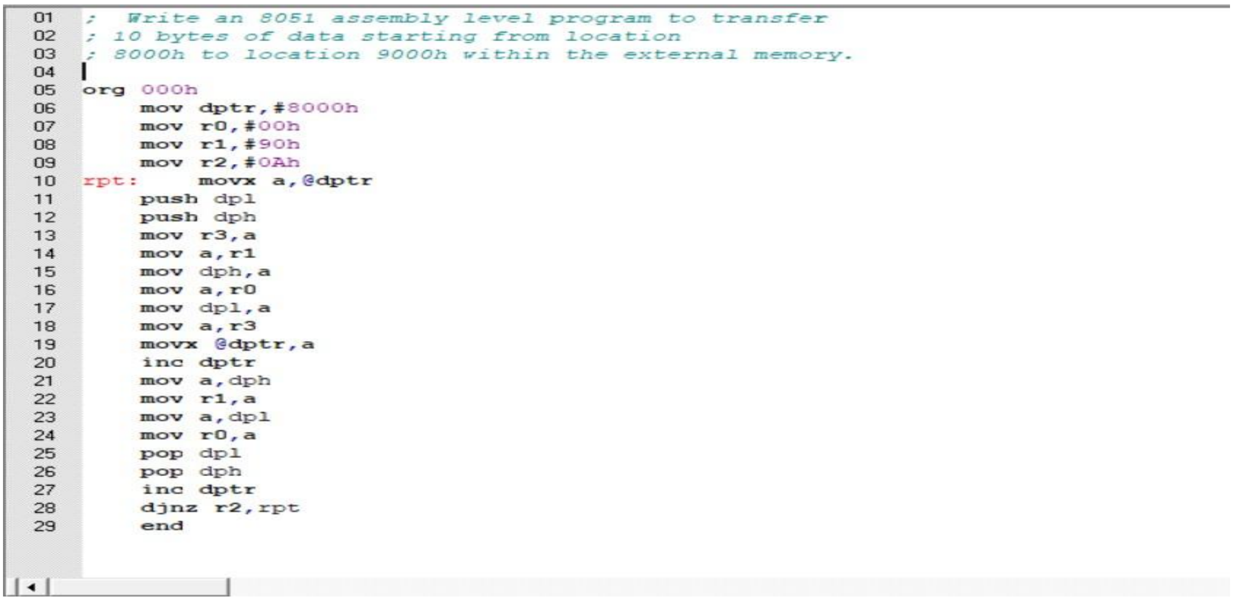
After Execution:
30H: 0BH, 17H, 2DH, 41, 4AH, 53, 52H, 0FH, 1CH, 0AH
1000H:01H,02H,3H,4H,5H,6H,7H,8H,9H,0AH



Address: x:1000h

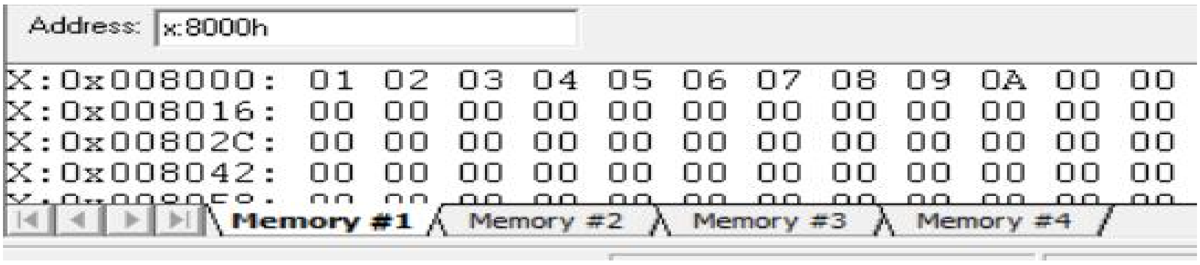
```
X:0x001000:  0B  17  2D  41  4A  53  52  0F  1C  0A  00  00
X:0x001016:  00  00  00  00  00  00  00  00  00  00  00  00
X:0x00102C:  00  00  00  00  00  00  00  00  00  00  00  00
X:0x001042:  00  00  00  00  00  00  00  00  00  00  00  00
X:0x001058:  00  00  00  00  00  00  00  00  00  00  00  00
```

1.6 Write an 8051 assembly level program to transfer 10 bytes of data starting from location 8000h to location 9000h within the external memory.



```
;  Write an 8051 assembly level program to transfer
; 10 bytes of data starting from location
; 8000h to location 9000h within the external memory.

org 000h
    mov dptr,#8000h
    mov r0,#00h
    mov r1,#90h
    mov r2,#0Ah
rpt:    movx a,@dptr
    push dpl
    push dph
    mov r3,a
    mov a,r1
    mov dph,a
    mov a,r0
    mov dpl,a
    mov a,r3
    movx @dptr,a
    inc dptr
    mov a,dph
    mov r1,a
    mov a,dpl
    mov r0,a
    pop dpl
    pop dph
    inc dptr
    djnz r2,rpt
    end
```

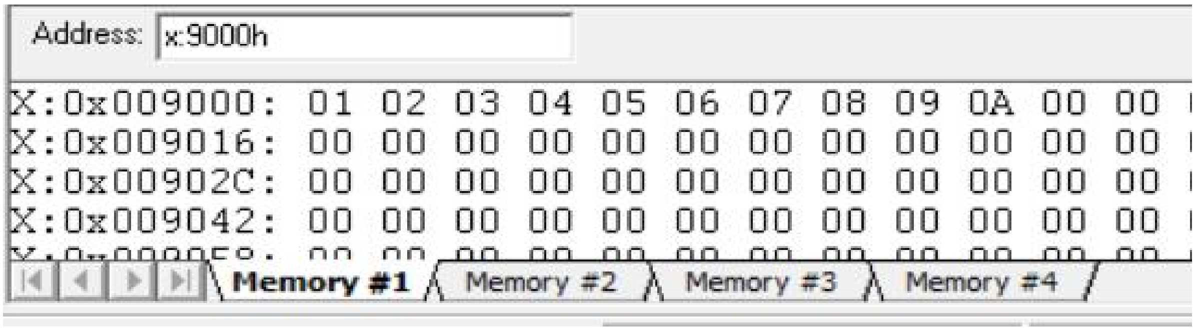
Before Execution:



Address: x:8000h

```
X:0x008000: 01 02 03 04 05 06 07 08 09 0A 00 00
X:0x008016: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00802C: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x008042: 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution:

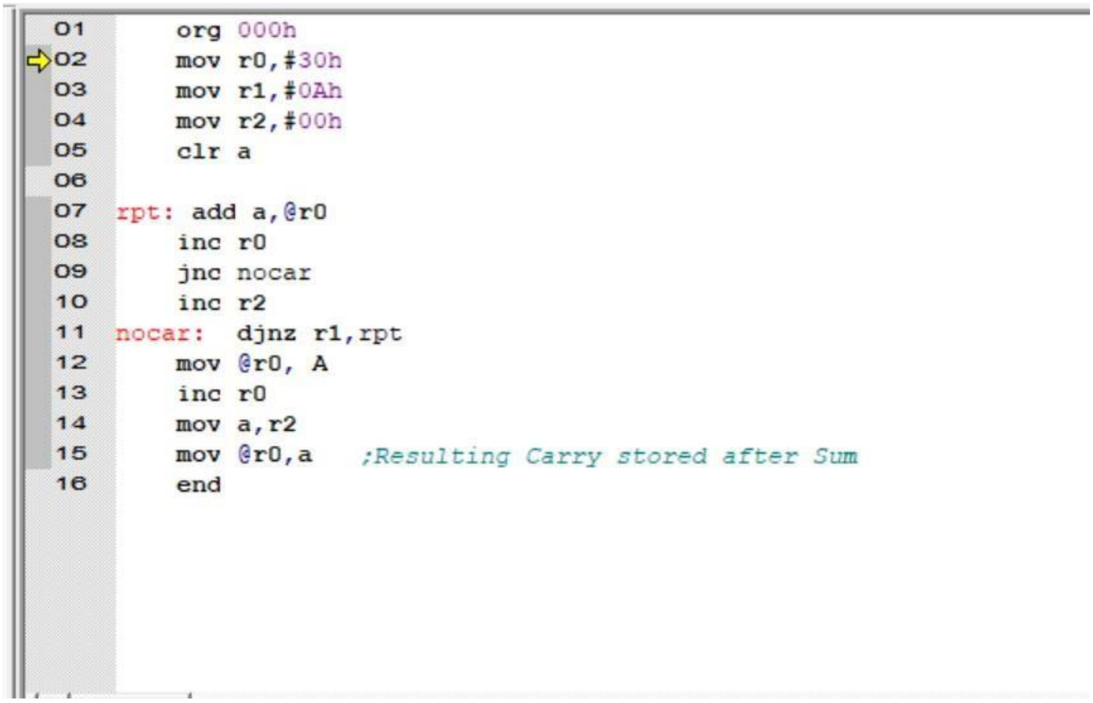


Address: x:9000h

```
X:0x009000: 01 02 03 04 05 06 07 08 09 0A 00 00
X:0x009016: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00902C: 00 00 00 00 00 00 00 00 00 00 00 00
X:0x009042: 00 00 00 00 00 00 00 00 00 00 00 00
```

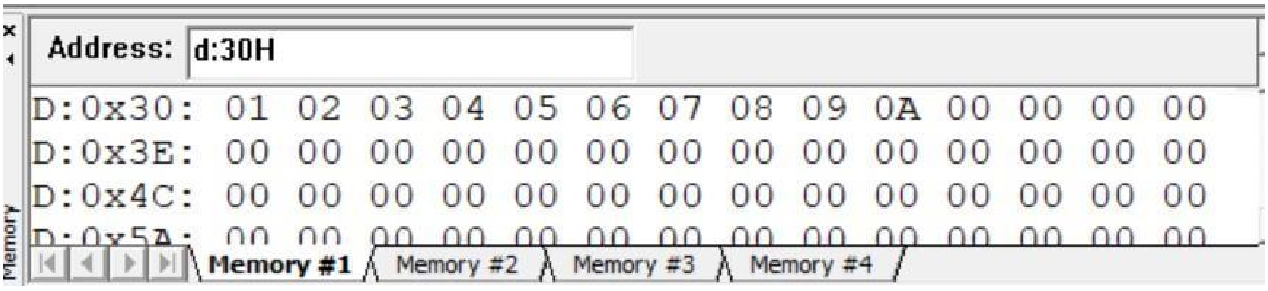
SET 2 PROGRAMS

2.1 Write an alp to add N bytes of data taking into account the possible carry output.



```
     org 000h
     mov r0,#30h
     mov r1,#0Ah
     mov r2,#00h
     clr a

rpt: add a,@r0
     inc r0
     jnc nocar
     inc r2
nocar:  djnz r1,rpt
     mov @r0, A
     inc r0
     mov a,r2
     mov @r0,a    ;Resulting Carry stored after Sum
     end
```

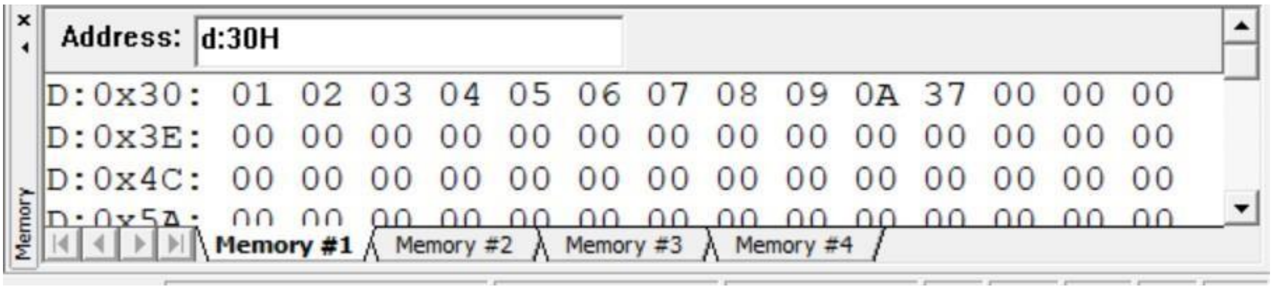
Before Execution:



Address: d:30H

```
D:0x30:  01  02  03  04  05  06  07  08  09  0A  00  00  00  00
D:0x3E:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4C:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x5A:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
```

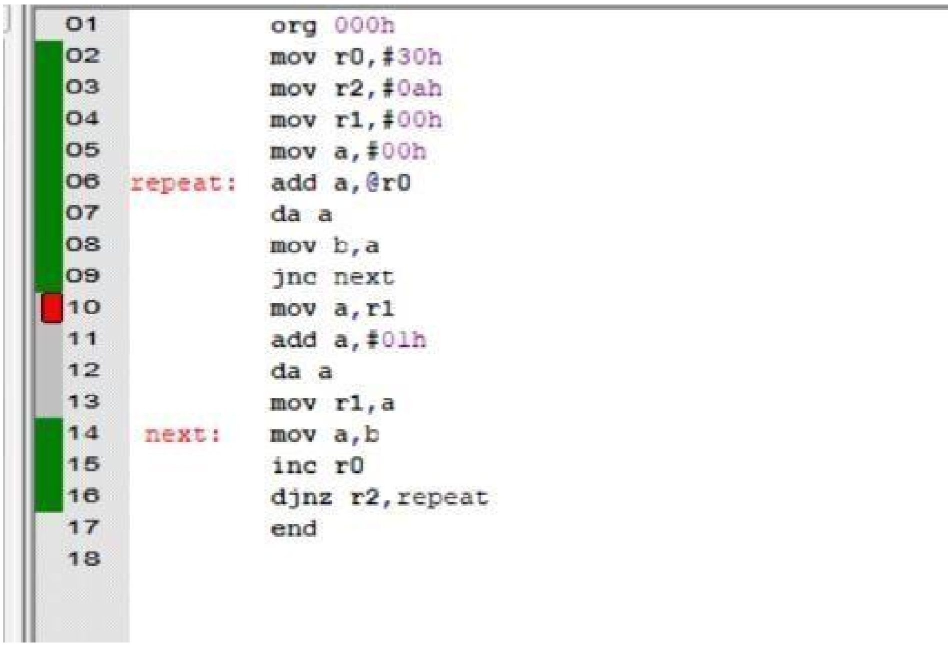
After Execution:



Address: d:30H

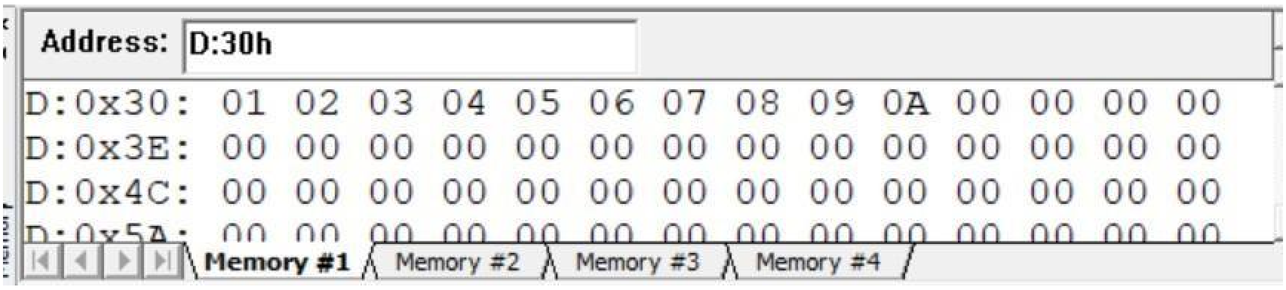
```
D:0x30:  01  02  03  04  05  06  07  08  09  0A  37  00  00  00
D:0x3E:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4C:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x5A:  00  00  00  00  00  00  00  00  00  00  00  00  00  00
```

2.2 Write an alp to add N bytes of BCD numbers talking into account the possible carry output.



```
          org 000h
          mov r0,#30h
          mov r2,#0ah
          mov r1,#00h
          mov a,#00h
repeat:   add a,@r0
          da a
          mov b,a
          jnc next
          mov a,r1
          add a,#01h
          da a
          mov r1,a
 next:    mov a,b
          inc r0
          djnz r2,repeat
          end
```

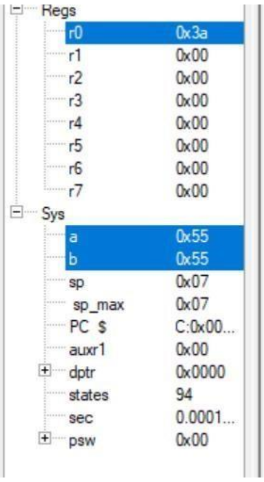
Before Execution:



Address: D:30h

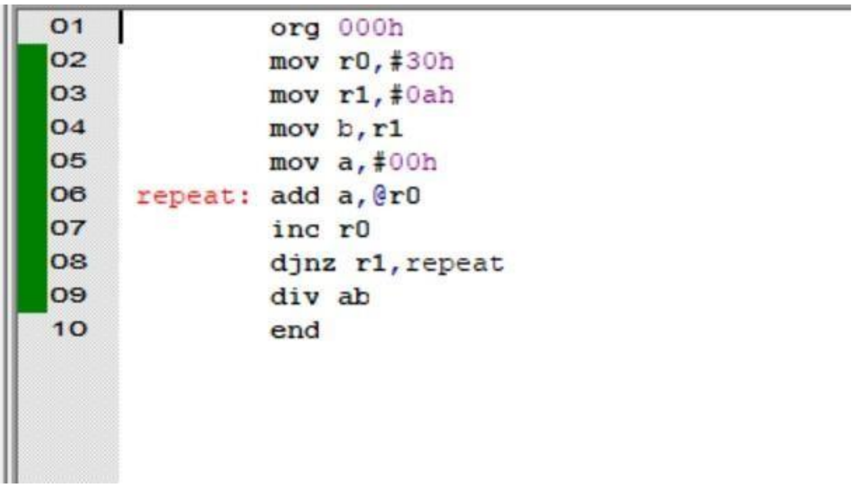
```
D:0x30:  01 02 03 04 05 06 07 08 09 0A 00 00 00 00
D:0x3E:  00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4C:  00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5A:  00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution:



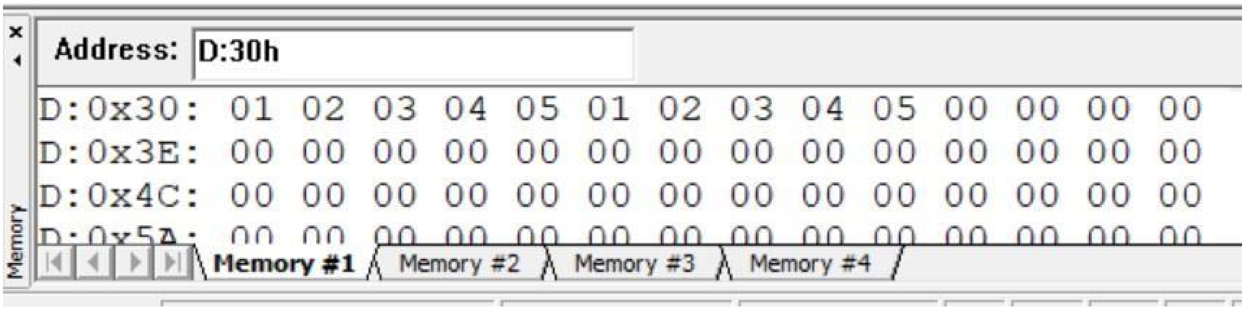
```
Regs
  r0        0x3a
  r1        0x00
  r2        0x00
  r3        0x00
  r4        0x00
  r5        0x00
  r6        0x00
  r7        0x00
Sys
  a         0x55
  b         0x55
  sp        0x07
  sp_max    0x07
  PC $      C:0x00...
  auxr1     0x00
  dptr      0x0000
  states    94
  sec       0.0001...
  psw       0x00
```

2.3 Write an alp to find the average of N bytes of data.



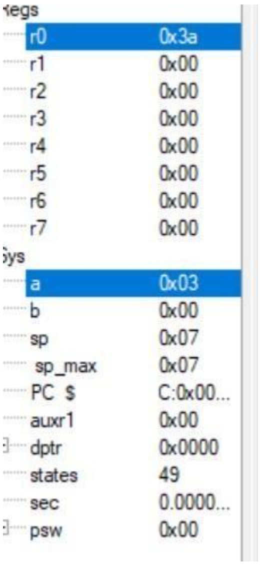
```
         org 000h
         mov r0,#30h
         mov r1,#0ah
         mov b,r1
         mov a,#00h
repeat:  add a,@r0
         inc r0
         djnz r1,repeat
         div ab
         end
```

Before Execution:



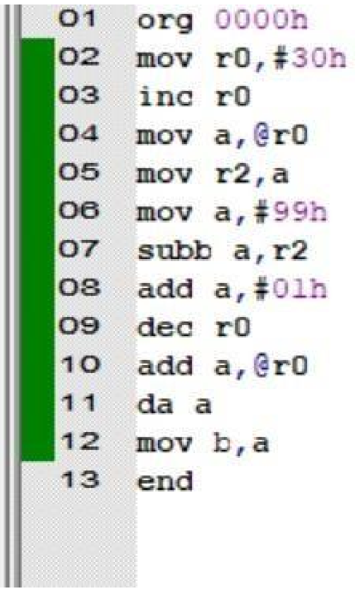
```
Address: D:30h
D:0x30: 01 02 03 04 05 01 02 03 04 05 00 00 00 00
D:0x3E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution



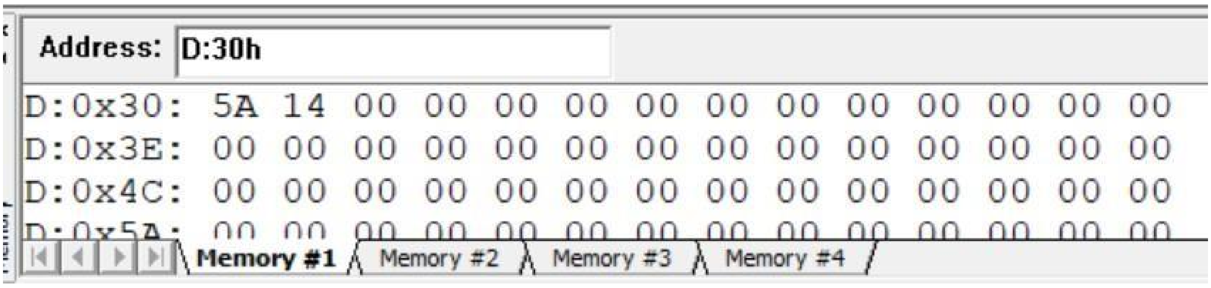
```
Regs
  r0        0x3a
  r1        0x00
  r2        0x00
  r3        0x00
  r4        0x00
  r5        0x00
  r6        0x00
  r7        0x00
Sys
  a         0x03
  b         0x00
  sp        0x07
  sp_max    0x07
  PC $      C:0x00...
  auxr1     0x00
  dptr      0x0000
  states    49
  sec       0.0000...
  psw       0x00
```

2.4 Write an alp to subtract two BCD numbers.



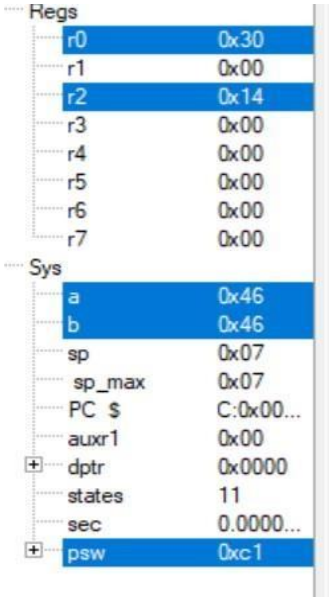
```
org 0000h
mov r0,#30h
inc r0
mov a,@r0
mov r2,a
mov a,#99h
subb a,r2
add a,#01h
dec r0
add a,@r0
da a
mov b,a
end
```

Before Execution:



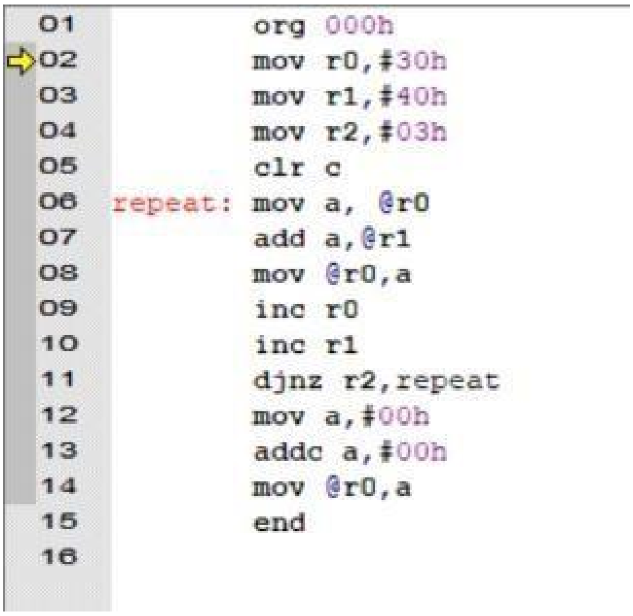
```
Address: D:30h
D:0x30: 5A 14 00 00 00 00 00 00 00 00 00 00 00 00
D:0x3E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution:



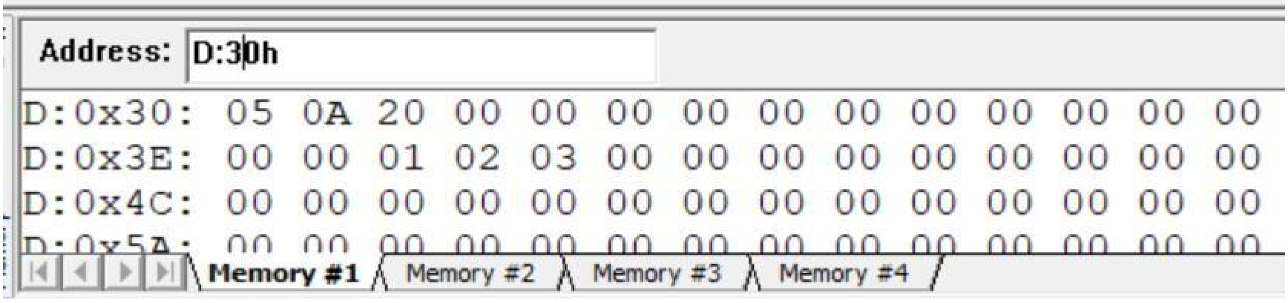
```
Regs
  r0        0x30
  r1        0x00
  r2        0x14
  r3        0x00
  r4        0x00
  r5        0x00
  r6        0x00
  r7        0x00
Sys
  a         0x46
  b         0x46
  sp        0x07
  sp_max    0x07
  PC $      C:0x00...
  auxr1     0x00
  dptr      0x0000
  states    11
  sec       0.0000...
  psw       0xc1
```

2.5 Write an alp to add 2 multibyte numbers. Numbers starts from location with address 30h and 40h. Store the results starting from location 30h.



```
        org 000h
        mov r0,#30h
        mov r1,#40h
        mov r2,#03h
        clr c
repeat: mov a, @r0
        add a,@r1
        mov @r0,a
        inc r0
        inc r1
        djnz r2,repeat
        mov a,#00h
        addc a,#00h
        mov @r0,a
        end
```

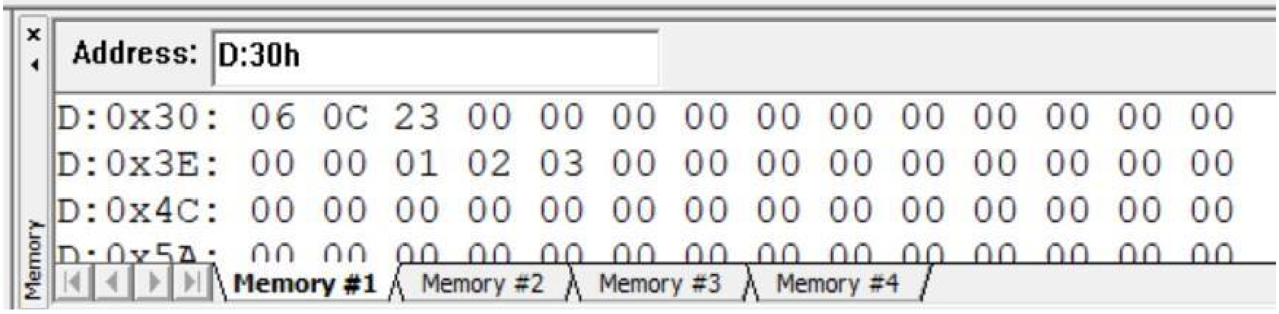
Before Execution:



Address: D:30h

```
D:0x30: 05 0A 20 00 00 00 00 00 00 00 00 00 00 00
D:0x3E: 00 00 01 02 03 00 00 00 00 00 00 00 00 00
D:0x4C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution:

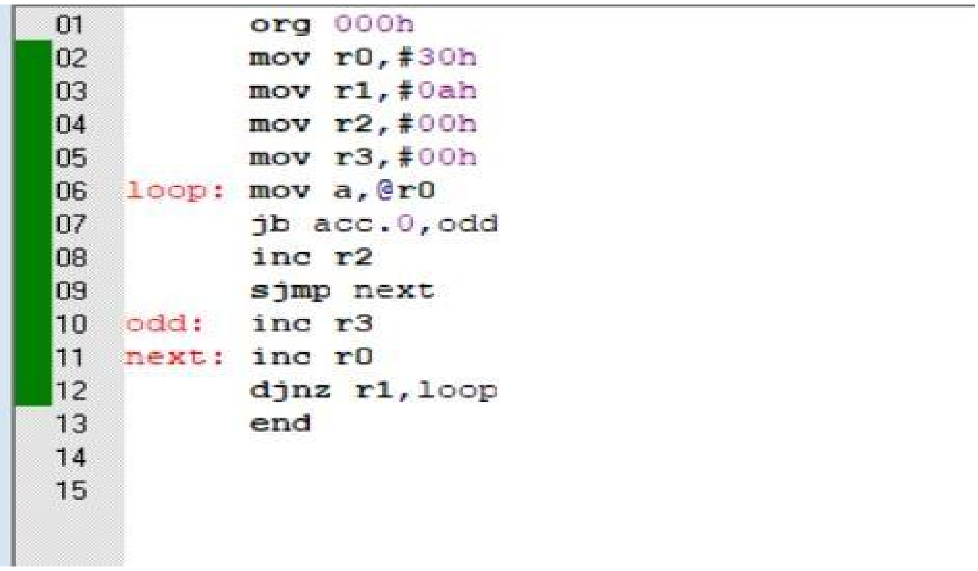


Address: D:30h

```
D:0x30: 06 0C 23 00 00 00 00 00 00 00 00 00 00 00
D:0x3E: 00 00 01 02 03 00 00 00 00 00 00 00 00 00
D:0x4C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x5A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

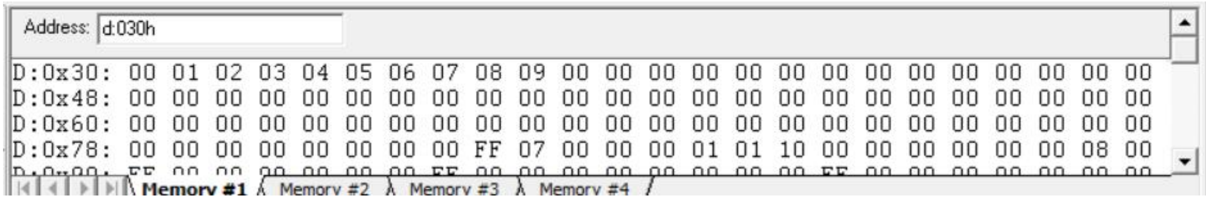
SET 3 PROGRAMS

3.1 Write an 8051 assembly level program to count the number of even Numbers and number of odd numbers in an array of ‘N’ bytes of data



```
        org 000h
        mov r0,#30h
        mov r1,#0ah
        mov r2,#00h
        mov r3,#00h
loop:   mov a,@r0
        jb acc.0,odd
        inc r2
        sjmp next
odd:    inc r3
next:   inc r0
        djnz r1,loop
        end
```

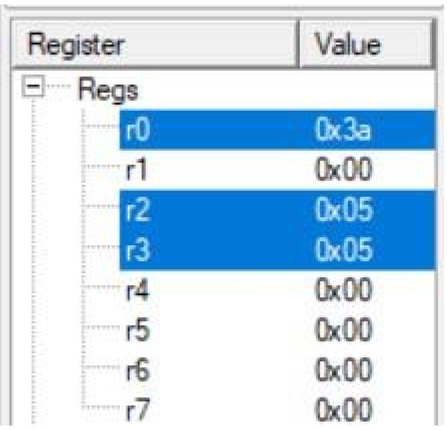
Before execution



Address: d:030h

```
D:0x30:  00 01 02 03 04 05 06 07 08 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78:  00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
D:0x90:  FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00
```

After Execution

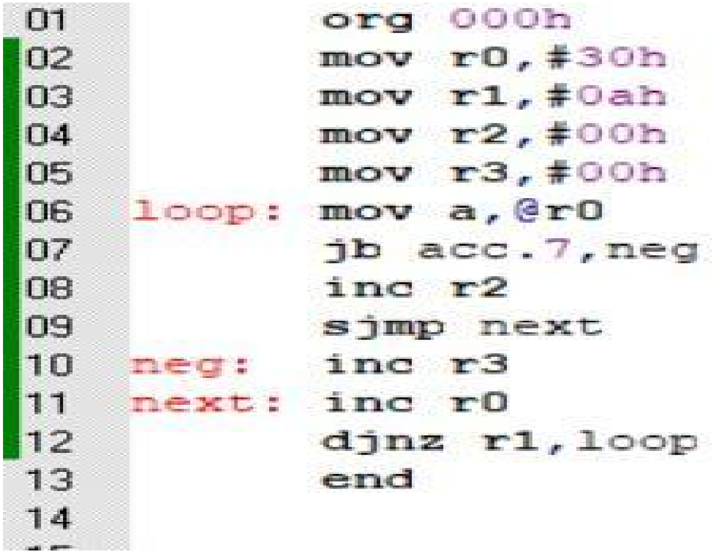


Register	Value
Regs	
r0	0x3a
r1	0x00
r2	0x05
r3	0x05
r4	0x00
r5	0x00
r6	0x00
r7	0x00

3.2 Write an 8051 assembly level program to count the number of +ve numbers and number of –ve numbers in an array of 'N' bytes of data.

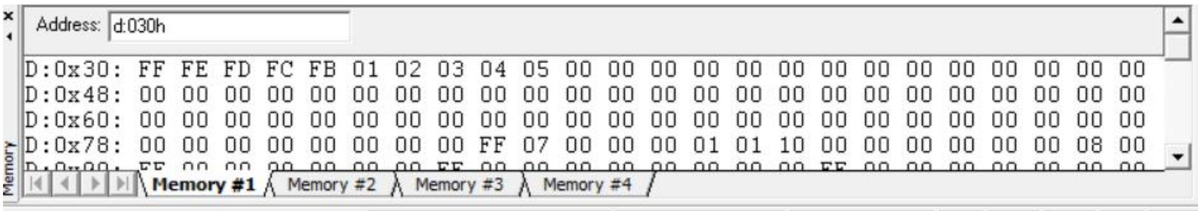
MSB bit of data represents sign of the number. If MSB is 1, then the number is negative.

If MSB is 0, then the number is positive



```
        org  000h
        mov  r0,#30h
        mov  r1,#0ah
        mov  r2,#00h
        mov  r3,#00h
loop:   mov  a,@r0
        jb   acc.7,neg
        inc  r2
        sjmp next
neg:    inc  r3
next:   inc  r0
        djnz r1,loop
        end
```

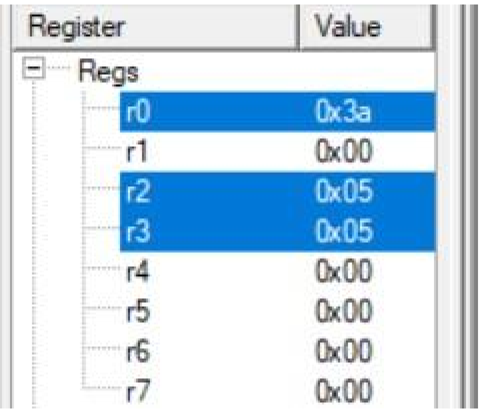
Before execution:



Address: d:030h

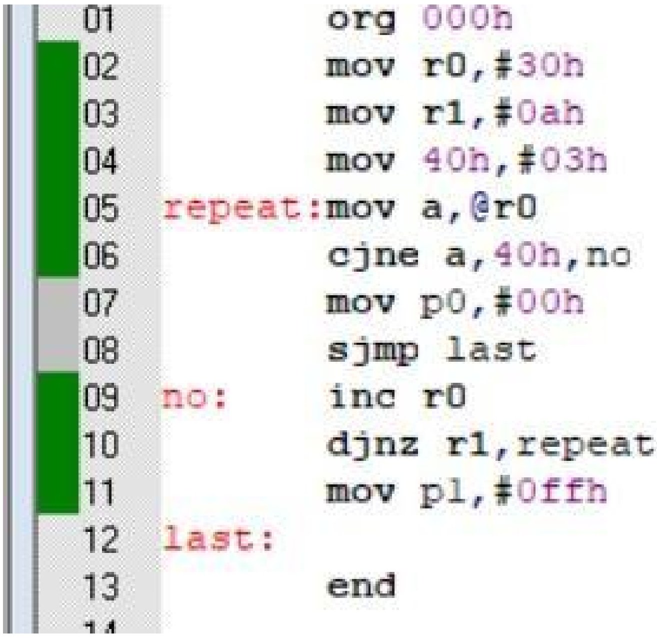
```
D:0x30: FF FE FD FC FB 01 02 03 04 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
```

After execution



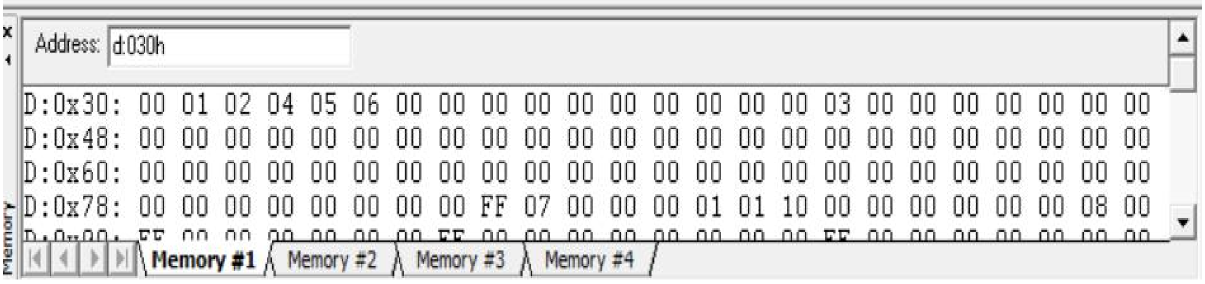
Register	Value
Regs	
r0	0x3a
r1	0x00
r2	0x05
r3	0x05
r4	0x00
r5	0x00
r6	0x00
r7	0x00

3.3 Check whether the given byte of data is present in an array of 'N' bytes of data. If present send 00 in Port 0 else send FF in Port 0



```
        org 000h
        mov r0,#30h
        mov r1,#0ah
        mov 40h,#03h
repeat:mov a,@r0
        cjne a,40h,no
        mov p0,#00h
        sjmp last
no:     inc r0
        djnz r1,repeat
        mov p1,#0ffh
last:
        end
```

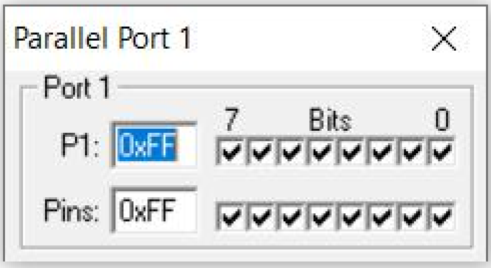
Before execution:



Address: d:030h

```
D:0x30: 00 01 02 04 05 06 00 00 00 00 00 00 00 00 00 00 03 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
```

After Execution:



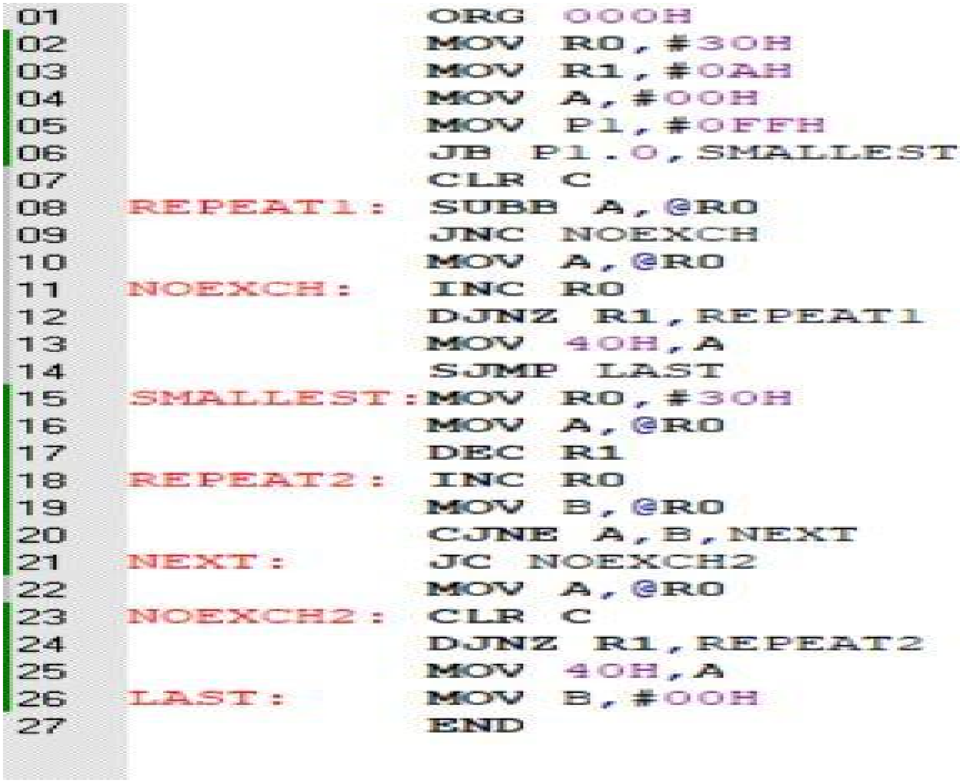
Parallel Port 1

Port 1

P1: 0xFF ☑☑☑☑☑☑☑☑

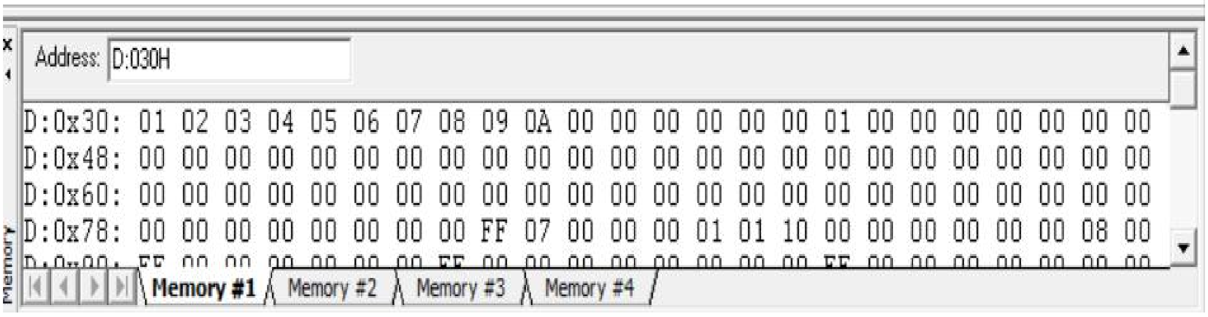
Pins: 0xFF ☑☑☑☑☑☑☑☑

3.4 Read the data from Port1. IfP1.1 is at logic0, find the largest number in an array of 'N' bytes of data and store in location 40h. If P1.0 is at logic1, find the smallest number in the array and store in the location 40h



```
                  ORG  000H
                  MOV  R0,#30H
                  MOV  R1,#0AH
                  MOV  A,#00H
                  MOV  P1,#0FFH
                  JB  P1.0,SMALLEST
                  CLR  C
REPEAT1:          SUBB  A,@R0
                  JNC  NOEXCH
                  MOV  A,@R0
NOEXCH:           INC  R0
                  DJNZ  R1,REPEAT1
                  MOV  40H,A
                  SJMP  LAST
SMALLEST:MOV  R0,#30H
                  MOV  A,@R0
                  DEC  R1
REPEAT2:          INC  R0
                  MOV  B,@R0
                  CJNE  A,B,NEXT
NEXT:             JC  NOEXCH2
                  MOV  A,@R0
NOEXCH2:          CLR  C
                  DJNZ  R1,REPEAT2
                  MOV  40H,A
LAST:             MOV  B,#00H
                  END
```

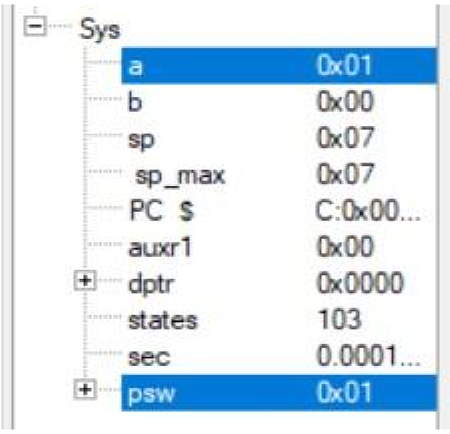
Before execution:



Address: D:030H

```
D:0x30: 01 02 03 04 05 06 07 08 09 0A 00 00 00 00 00 00 01 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
D:0x90: FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00
```

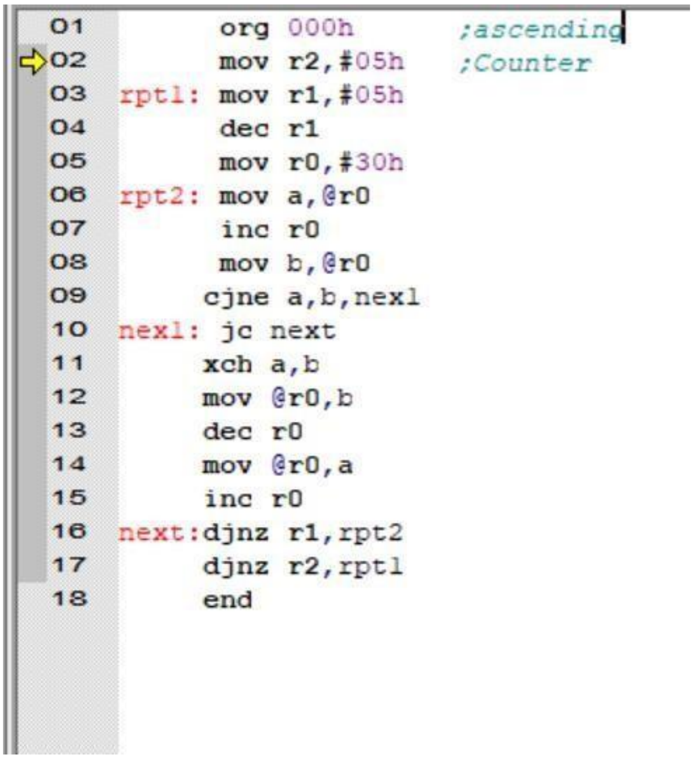
After execution:



```
Sys
  a         0x01
  b         0x00
  sp        0x07
  sp_max    0x07
  PC  $     C:0x00...
  auxr1     0x00
  dptr      0x0000
  states    103
  sec       0.0001...
  psw       0x01
```

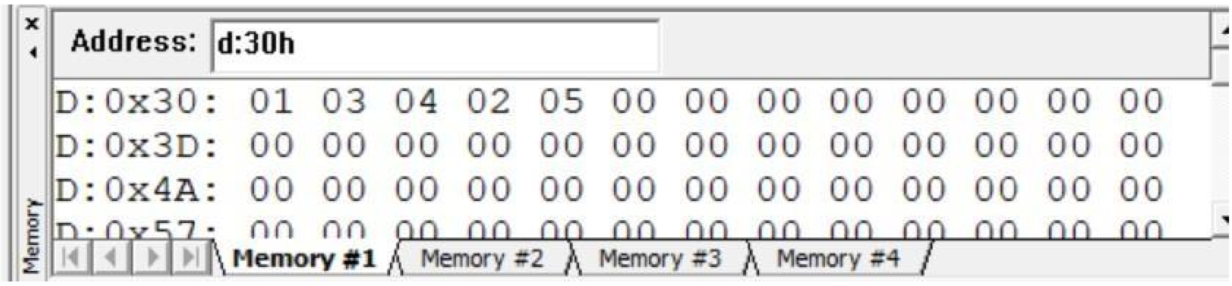
SET 4 PROGRAMS

4.1 Write an 8051 assembly level program to arrange an array of 'N' bytes of data in ascendingorder.



```
       org 000h        ;ascending
       mov r2,#05h     ;Counter
rpt1:  mov r1,#05h
       dec r1
       mov r0,#30h
rpt2:  mov a,@r0
       inc r0
       mov b,@r0
      cjne a,b,nexl
nexl:  jc next
      xch a,b
      mov @r0,b
      dec r0
      mov @r0,a
      inc r0
next:djnz r1,rpt2
      djnz r2,rpt1
      end
```

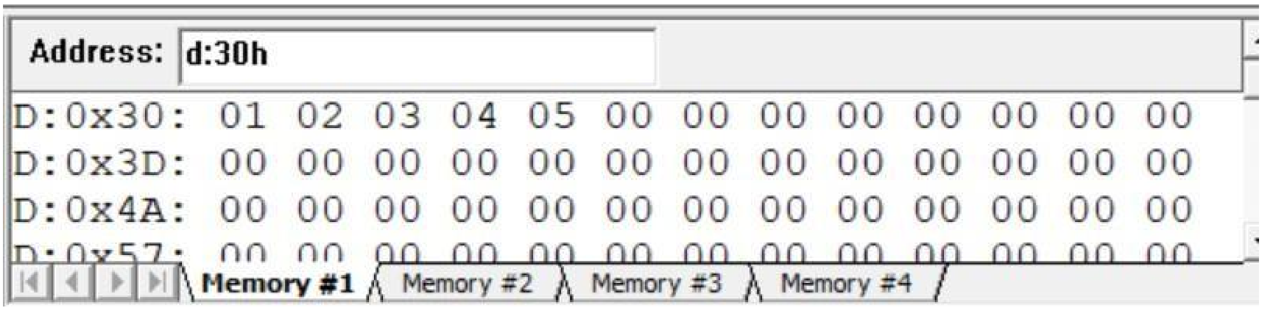
Before Execution:



Address: d:30h

```
D:0x30: 01 03 04 02 05 00 00 00 00 00 00 00 00
D:0x3D: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4A: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x57: 00 00 00 00 00 00 00 00 00 00 00 00 00
```

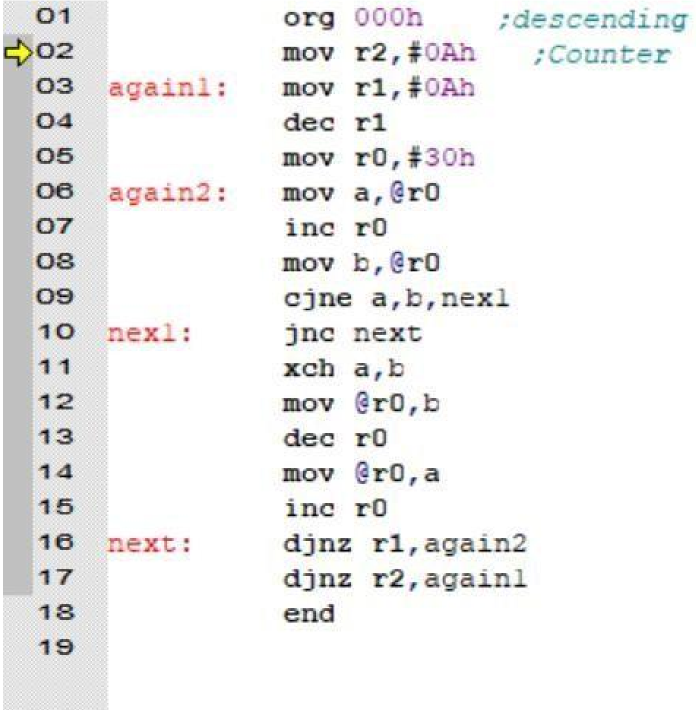
After Execution:



Address: d:30h

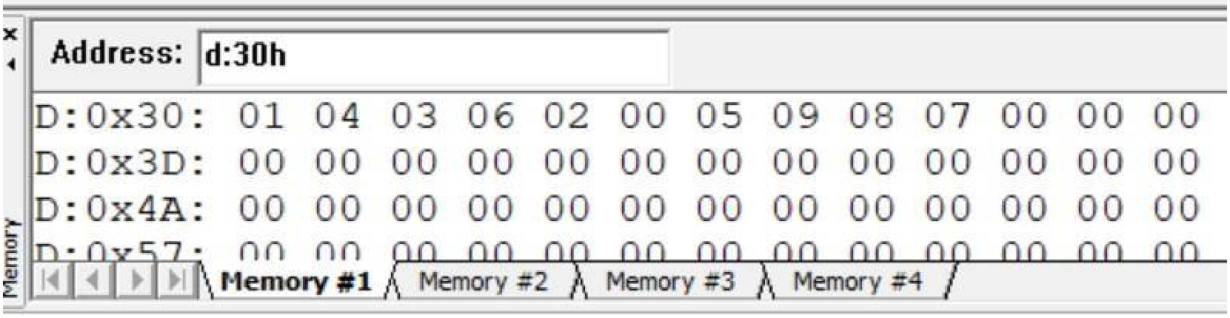
```
D:0x30: 01 02 03 04 05 00 00 00 00 00 00 00 00
D:0x3D: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4A: 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x57: 00 00 00 00 00 00 00 00 00 00 00 00 00
```

4.2 Write an 8051 assembly level program to arrange an array of ‘N’ bytes of data in descendingorder.



```
            org 000h        ;descending
            mov r2,#0Ah      ;Counter
again1:     mov r1,#0Ah
            dec r1
            mov r0,#30h
again2:     mov a,@r0
            inc r0
            mov b,@r0
            cjne a,b,nex1
nex1:       jnc next
            xch a,b
            mov @r0,b
            dec r0
            mov @r0,a
            inc r0
next:       djnz r1,again2
            djnz r2,again1
            end
```

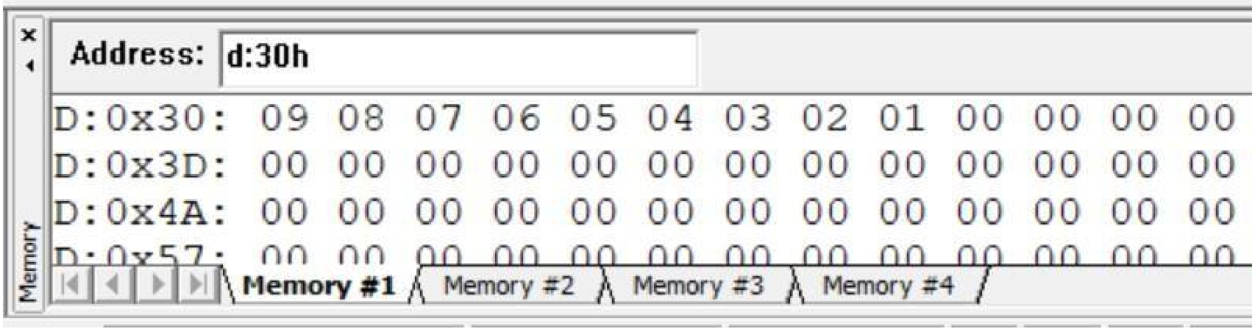
Before Execution:



Address: d:30h

```
D:0x30:  01 04 03 06 02 00 05 09 08 07 00 00 00
D:0x3D:  00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4A:  00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x57:  00 00 00 00 00 00 00 00 00 00 00 00 00
```

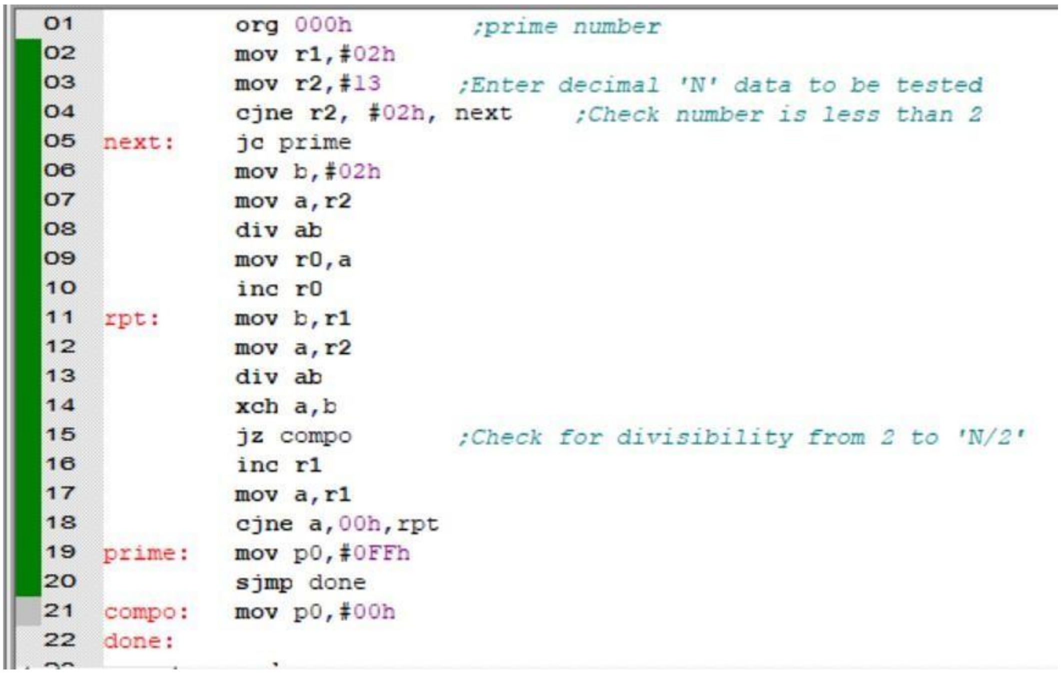
After Execution:



Address: d:30h

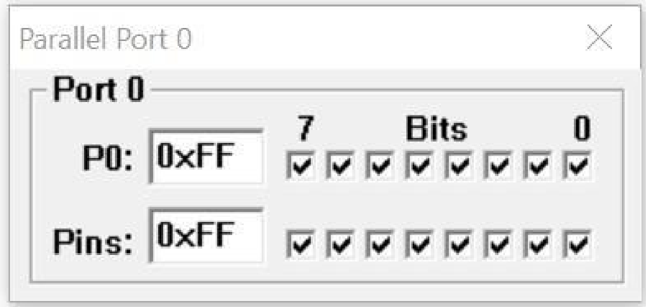
```
D:0x30:  09 08 07 06 05 04 03 02 01 00 00 00 00
D:0x3D:  00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x4A:  00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x57:  00 00 00 00 00 00 00 00 00 00 00 00 00
```

4.3 Write an 8051 assembly level program to find whether the given number is prime or not. If prime send FF to Port 0 else send 00 to Port 0.



```
         org 000h         ;prime number
         mov r1,#02h
         mov r2,#13      ;Enter decimal 'N' data to be tested
         cjne r2, #02h, next     ;Check number is less than 2
next:    jc prime
         mov b,#02h
         mov a,r2
         div ab
         mov r0,a
         inc r0
rpt:     mov b,r1
         mov a,r2
         div ab
         xch a,b
         jz compo        ;Check for divisibility from 2 to 'N/2'
         inc r1
         mov a,r1
         cjne a,00h,rpt
prime:   mov p0,#0FFh
         sjmp done
compo:   mov p0,#00h
done:
```

After Execution:



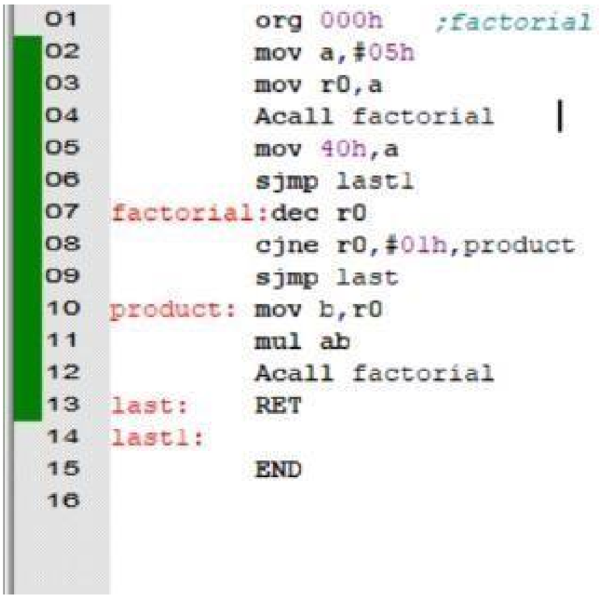
Parallel Port 0

Port 0

P0: 0xFF 7 Bits 0 ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑

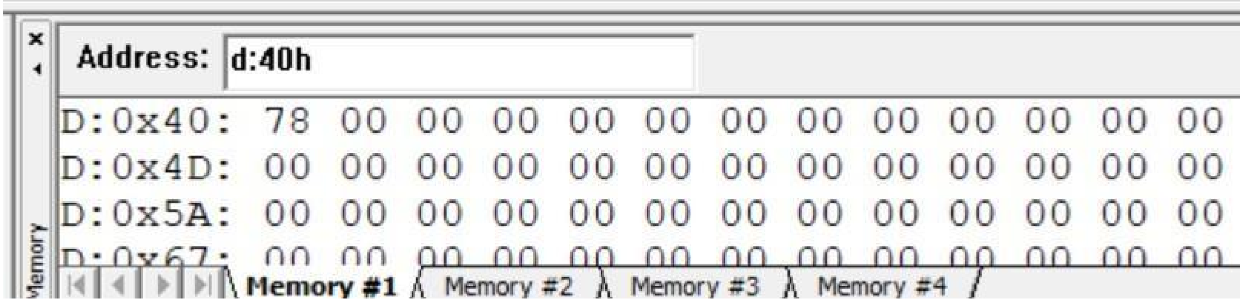
Pins: 0xFF ☑ ☑ ☑ ☑ ☑ ☑ ☑ ☑

4.4 Write an 8051 assembly level program to find the factorial of a given number (using recursiveprocedure).



```
          org 000h    ;factorial
          mov a,#05h
          mov r0,a
          Acall factorial
          mov 40h,a
          sjmp last1
factorial:dec r0
          cjne r0,#01h,product
          sjmp last
product:  mov b,r0
          mul ab
          Acall factorial
last:     RET
last1:
          END
```

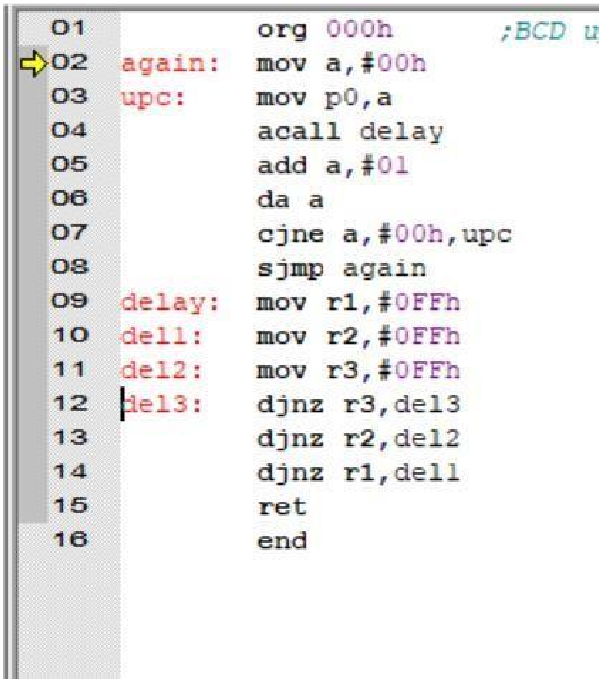
After Execution:



Address: d:40h

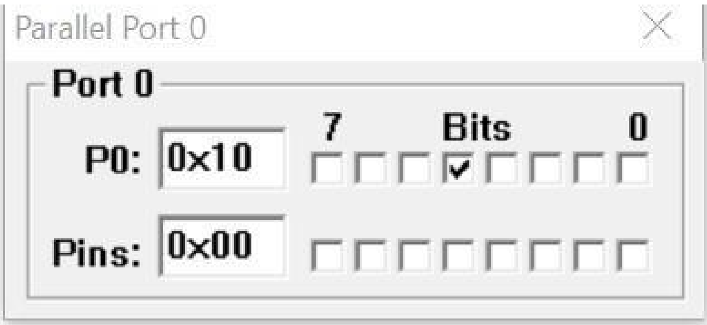
```
D:0x40:  78  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4D:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x5A:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x67:  00  00  00  00  00  00  00  00  00  00  00  00  00
```

4.5 Write an 8051 assembly level program for BCD up counter. Show each count in Port 0 with appropriate delay.



```
        org 000h       ;BCD u
again:  mov a,#00h
upc:    mov p0,a
        acall delay
        add a,#01
        da a
        cjne a,#00h,upc
        sjmp again
delay:  mov r1,#0FFh
del1:   mov r2,#0FFh
del2:   mov r3,#0FFh
del3:   djnz r3,del3
        djnz r2,del2
        djnz r1,del1
        ret
        end
```

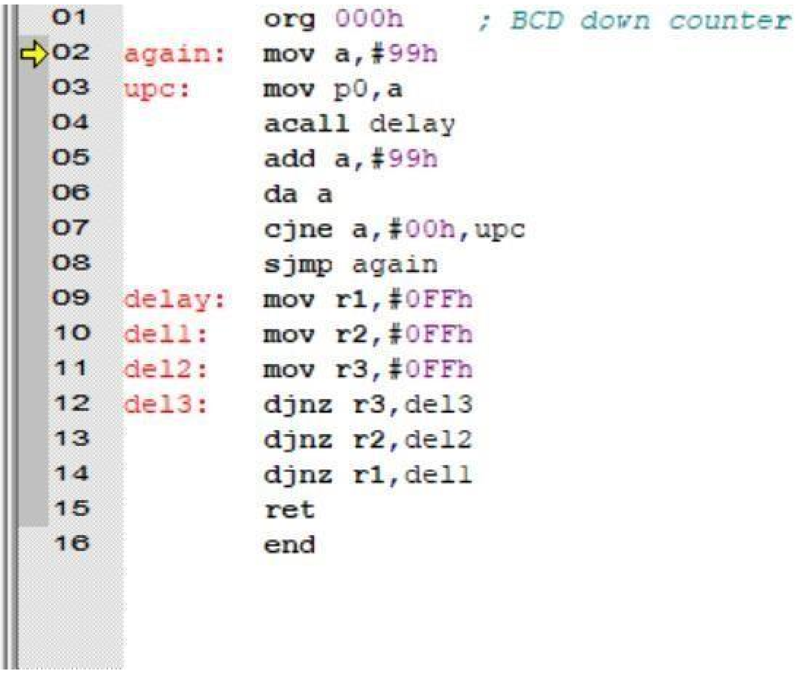
After Execution:



P0: 0x10 Bits 7 … 0: □ □ □ ☑ □ □ □ □

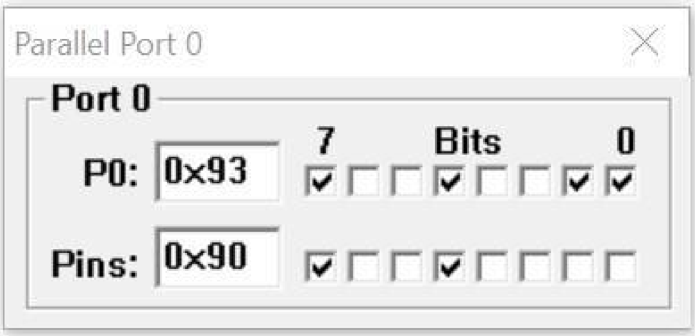
Pins: 0x00 □ □ □ □ □ □ □ □

4.6 Write an 8051 assembly level program for BCD down counter. Show each count in Port 0 with appropriate delay.



```
        org 000h     ; BCD down counter
again:  mov a,#99h
upc:    mov p0,a
        acall delay
        add a,#99h
        da a
        cjne a,#00h,upc
        sjmp again
delay:  mov r1,#0FFh
del1:   mov r2,#0FFh
del2:   mov r3,#0FFh
del3:   djnz r3,del3
        djnz r2,del2
        djnz r1,del1
        ret
        end
```

After Execution:



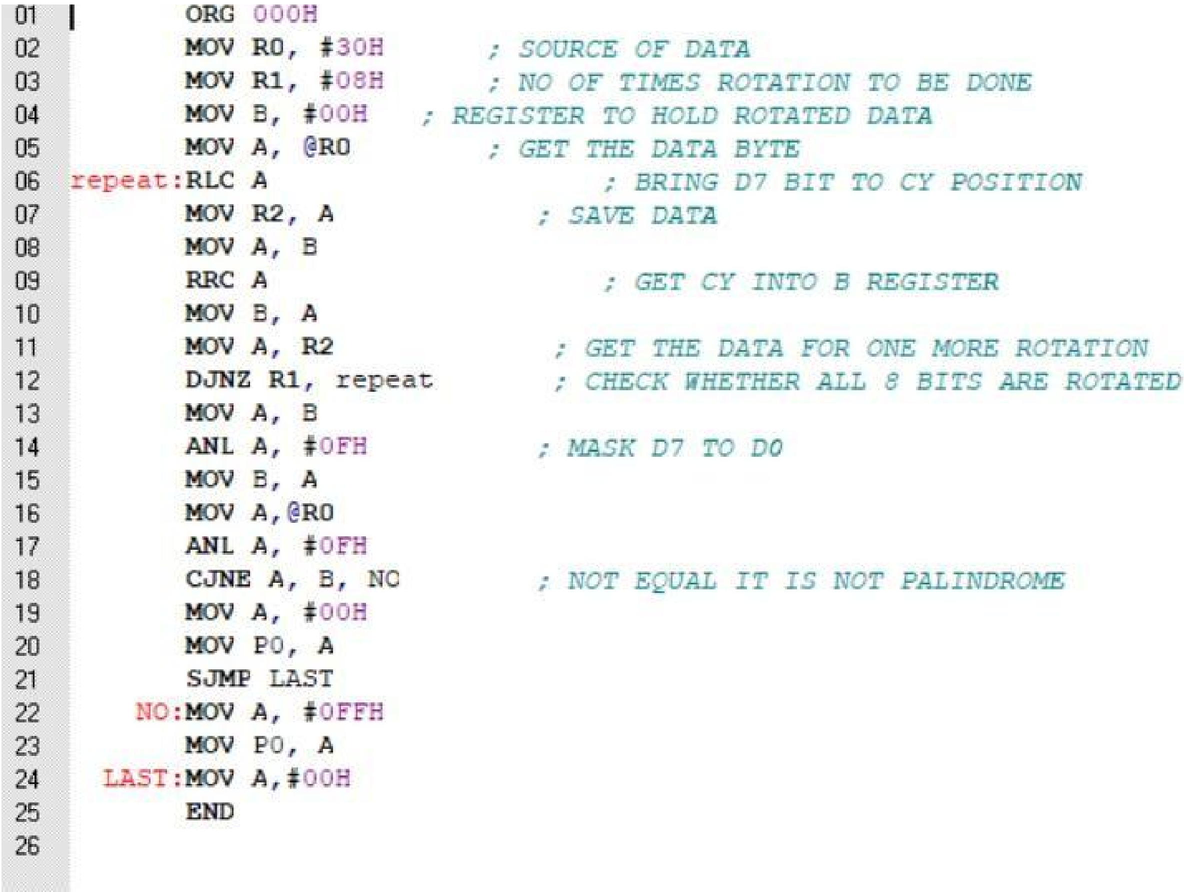
Parallel Port 0

Port 0

	7			Bits				0
P0: 0x93	☑	□	□	☑	□	□	☑	☑
Pins: 0x90	☑	□	□	☑	□	□	□	□

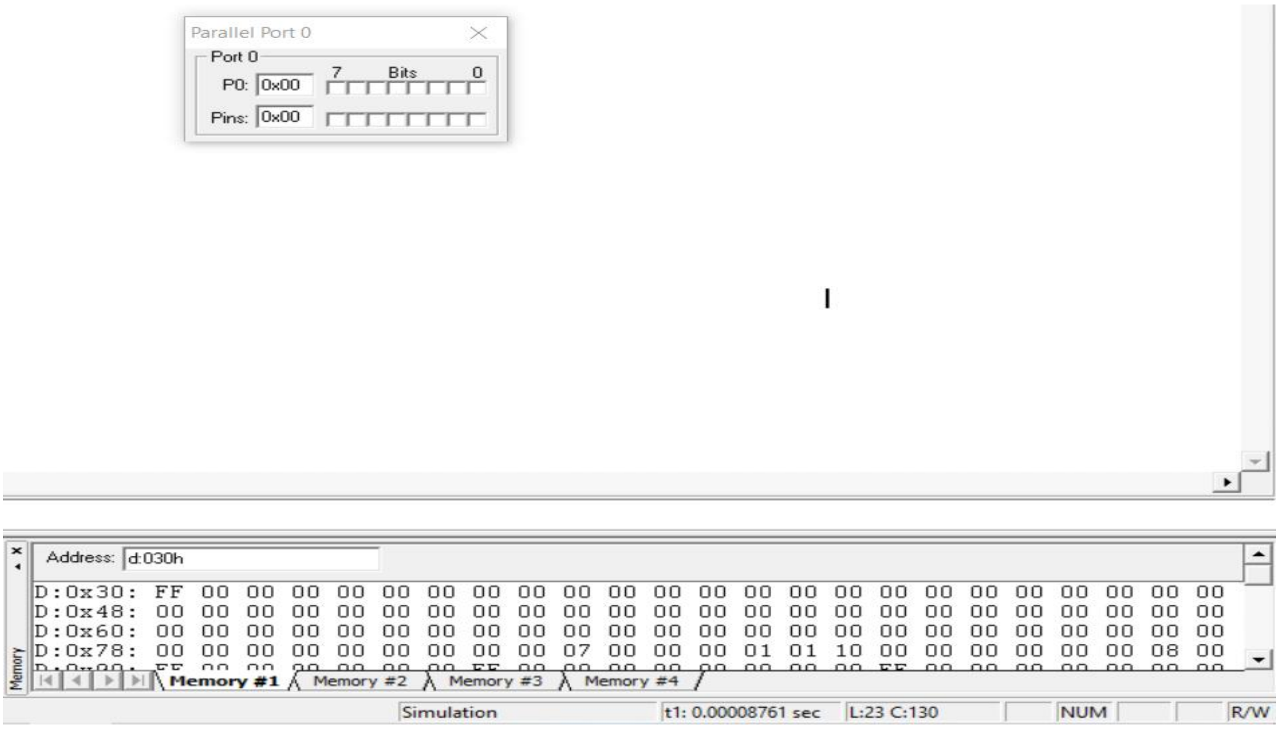
SET 5 PROGRAMS

5.1 Write an 8051 assembly level program to check whether the given byte of data is palindrome. If 'yes' send 00 to Port 0 else send FF to Port 0



```
        ORG 000H
        MOV R0, #30H        ; SOURCE OF DATA
        MOV R1, #08H        ; NO OF TIMES ROTATION TO BE DONE
        MOV B, #00H     ; REGISTER TO HOLD ROTATED DATA
        MOV A, @R0          ; GET THE DATA BYTE
repeat:RLC A                        ; BRING D7 BIT TO CY POSITION
        MOV R2, A               ; SAVE DATA
        MOV A, B
        RRC A                       ; GET CY INTO B REGISTER
        MOV B, A
        MOV A, R2                 ; GET THE DATA FOR ONE MORE ROTATION
        DJNZ R1, repeat           ; CHECK WHETHER ALL 8 BITS ARE ROTATED
        MOV A, B
        ANL A, #0FH             ; MASK D7 TO D0
        MOV B, A
        MOV A,@R0
        ANL A, #0FH
        CJNE A, B, NO           ; NOT EQUAL IT IS NOT PALINDROME
        MOV A, #00H
        MOV P0, A
        SJMP LAST
    NO:MOV A, #0FFH
        MOV P0, A
  LAST:MOV A,#00H
        END
```

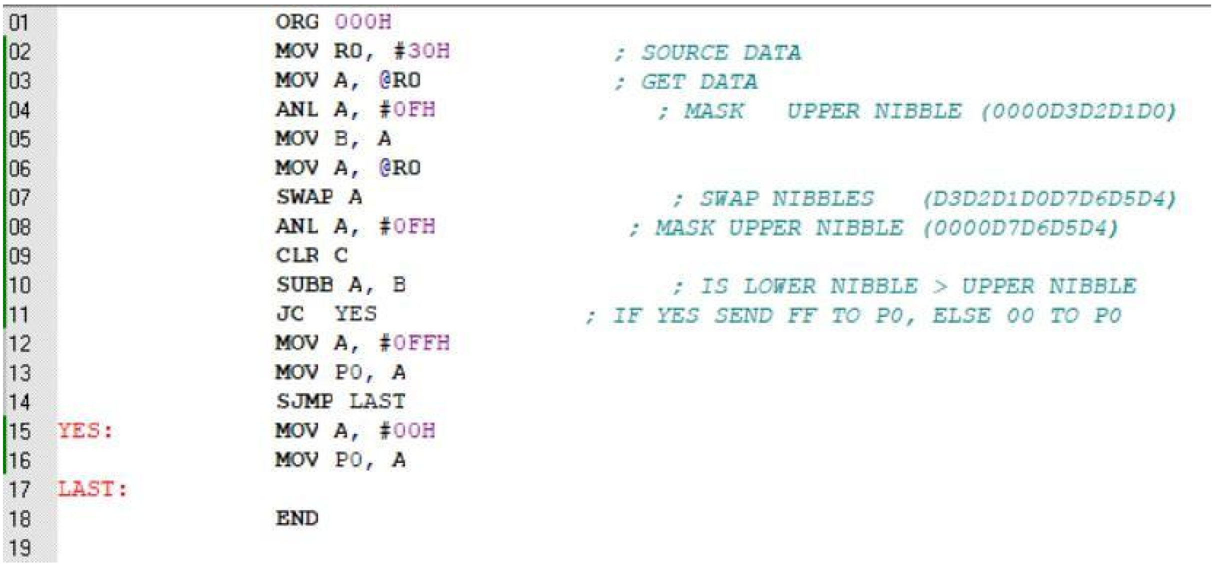
Before and after execution



Address: d:030h

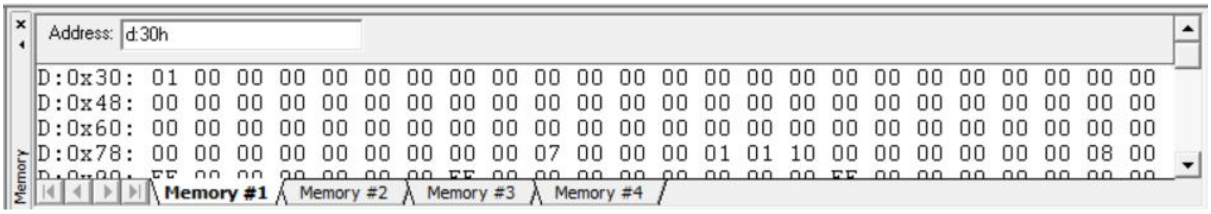
```
D:0x30: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 00 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
```

5.2 Write an 8051 assembly level program to check whether the lower nibble is greaterthanuppernibbleofA.If‘yes’send00toPort0elsesend FF to Port0.



```
                ORG 000H
                MOV R0, #30H          ; SOURCE DATA
                MOV A, @R0            ; GET DATA
                ANL A, #0FH              ; MASK   UPPER NIBBLE (0000D3D2D1D0)
                MOV B, A
                MOV A, @R0
                SWAP A                     ; SWAP NIBBLES    (D3D2D1D0D7D6D5D4)
                ANL A, #0FH            ; MASK UPPER NIBBLE (0000D7D6D5D4)
                CLR C
                SUBB A, B                  ; IS LOWER NIBBLE > UPPER NIBBLE
                JC   YES            ; IF YES SEND FF TO P0, ELSE 00 TO P0
                MOV A, #0FFH
                MOV P0, A
                SJMP LAST
YES:            MOV A, #00H
                MOV P0, A
LAST:
                END
```

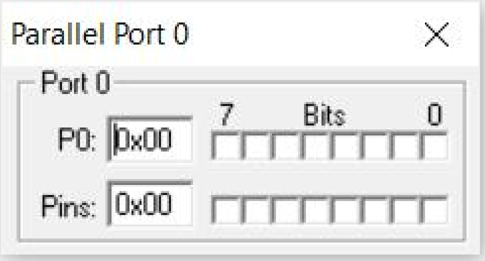
Before execution



Address: d:30h

```
D:0x30: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 00 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
D:0x90: FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00
```

After execution



Parallel Port 0

Port 0

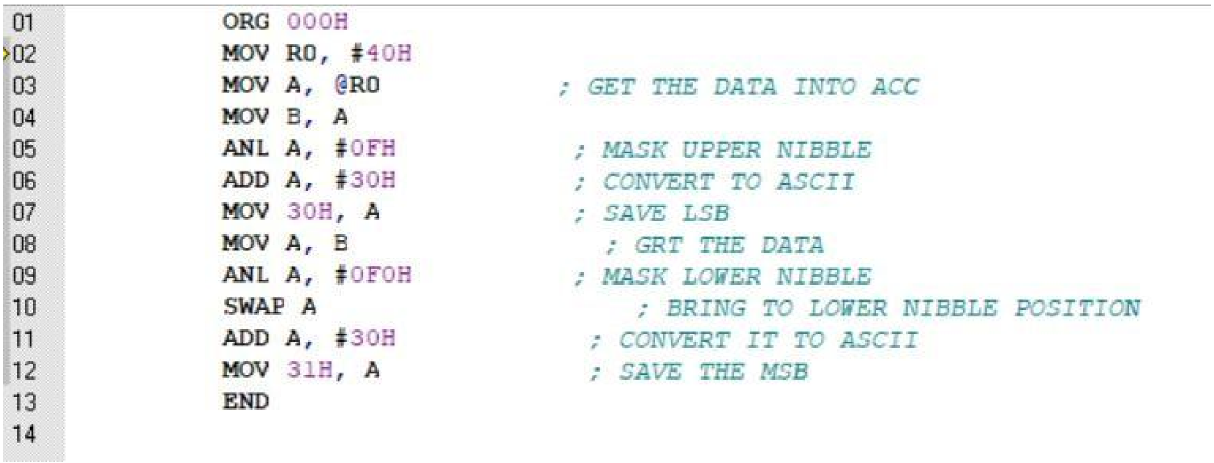
P0: 0x00

Pins: 0x00

□□□□□□□□

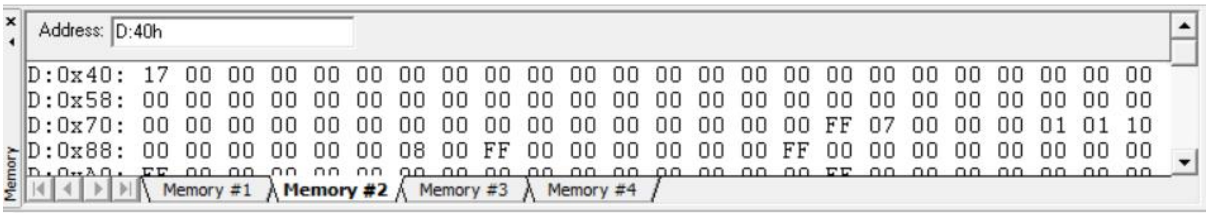
□□□□□□□□

5.3 Write an 8051 assembly level program to convert 2 digit BCD to ASCII numbersandstorethemin location30h(LSB)and31h(MSB)



```
            ORG 000H
            MOV R0, #40H
            MOV A, @R0          ; GET THE DATA INTO ACC
            MOV B, A
            ANL A, #0FH          ; MASK UPPER NIBBLE
            ADD A, #30H          ; CONVERT TO ASCII
            MOV 30H, A           ; SAVE LSB
            MOV A, B               ; GRT THE DATA
            ANL A, #0F0H         ; MASK LOWER NIBBLE
            SWAP A                   ; BRING TO LOWER NIBBLE POSITION
            ADD A, #30H           ; CONVERT IT TO ASCII
            MOV 31H, A            ; SAVE THE MSB
            END
```

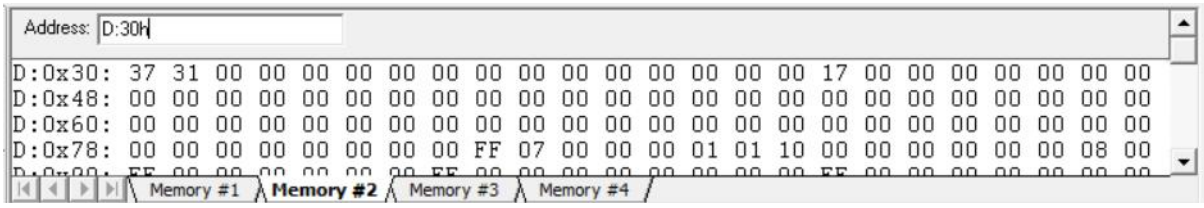
Before execution



Address: D:40h

```
D:0x40: 17 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x58: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10
D:0x88: 00 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
```

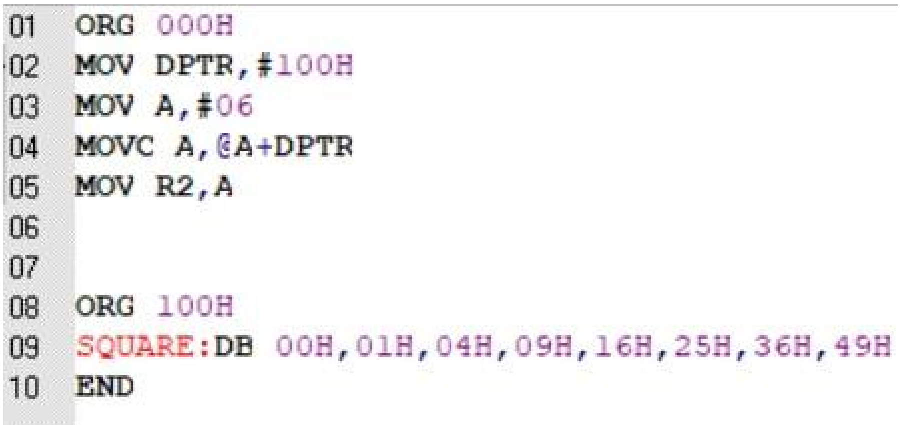
After execution



Address: D:30h

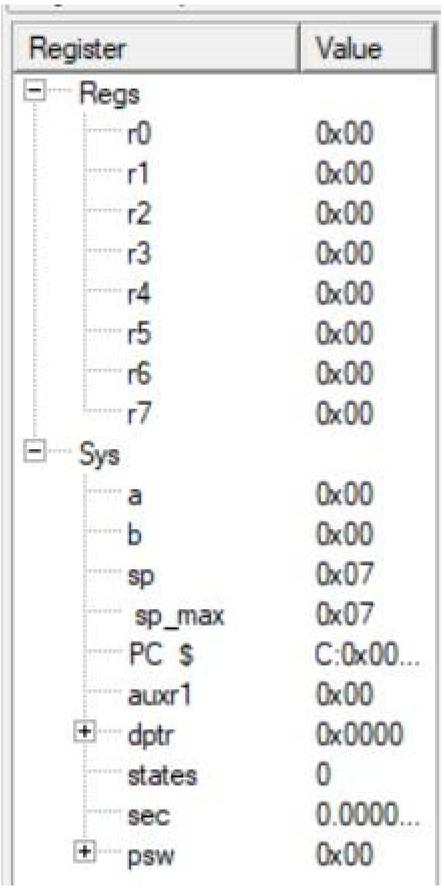
```
D:0x30: 37 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00 17 00 00 00 00 00 00 00
D:0x48: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00 00 00 08 00
```

5.4 Write an 8051 assembly level program to find the square of a number using look up table technique



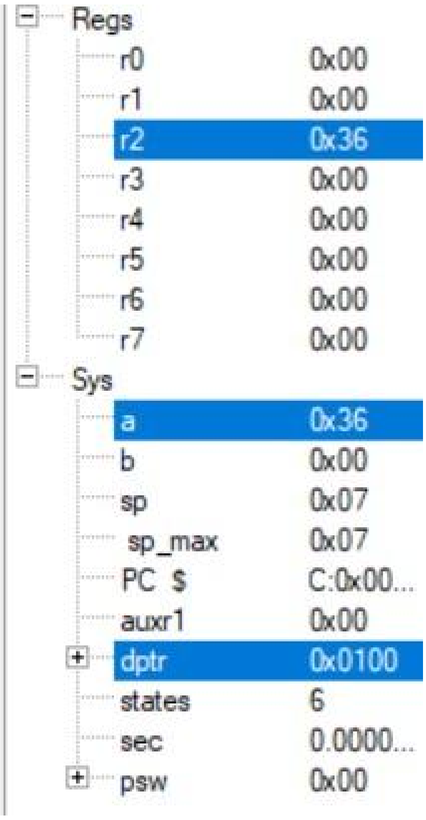
```
ORG 000H
MOV DPTR,#100H
MOV A,#06
MOVC A,@A+DPTR
MOV R2,A


ORG 100H
SQUARE:DB 00H,01H,04H,09H,16H,25H,36H,49H
END
```

Before execution



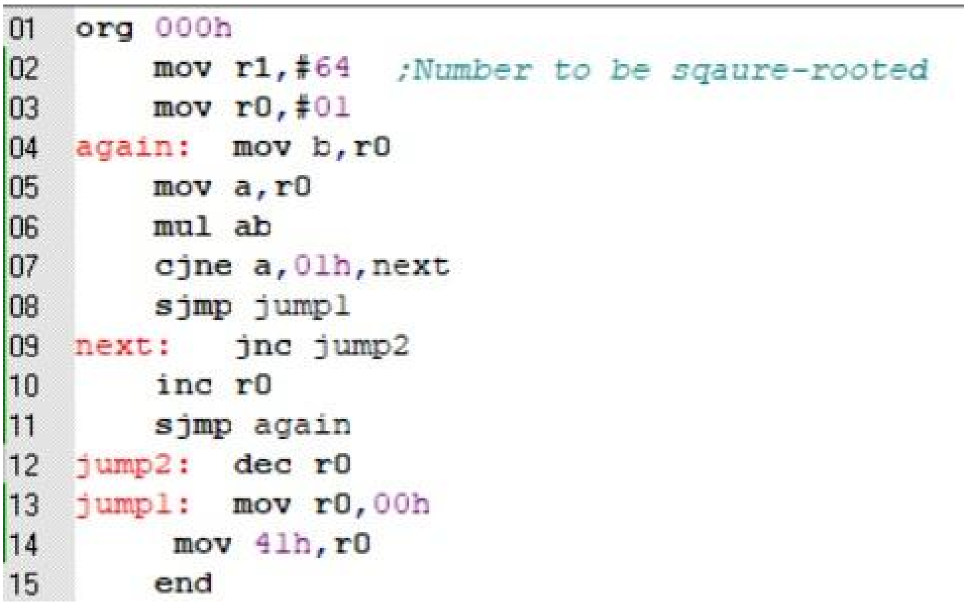
Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x07
sp_max	0x07
PC $	C:0x00...
auxr1	0x00
dptr	0x0000
states	0
sec	0.0000...
psw	0x00

After execution



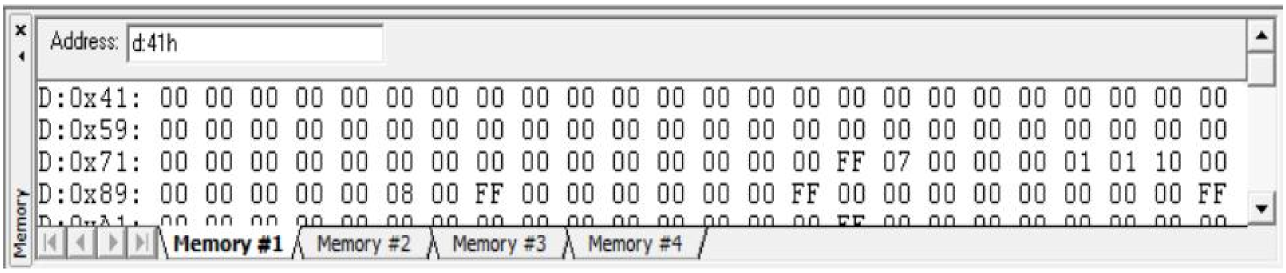
Regs	
r0	0x00
r1	0x00
r2	0x36
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x36
b	0x00
sp	0x07
sp_max	0x07
PC $	C:0x00...
auxr1	0x00
dptr	0x0100
states	6
sec	0.0000...
psw	0x00

5.5 Write an 8051 assembly level program to find the square root of a number



```
org 000h
    mov r1,#64   ;Number to be sqaure-rooted
    mov r0,#01
again:  mov b,r0
    mov a,r0
    mul ab
    cjne a,01h,next
    sjmp jump1
next:   jnc jump2
    inc r0
    sjmp again
jump2:  dec r0
jump1:  mov r0,00h
     mov 41h,r0
    end
```

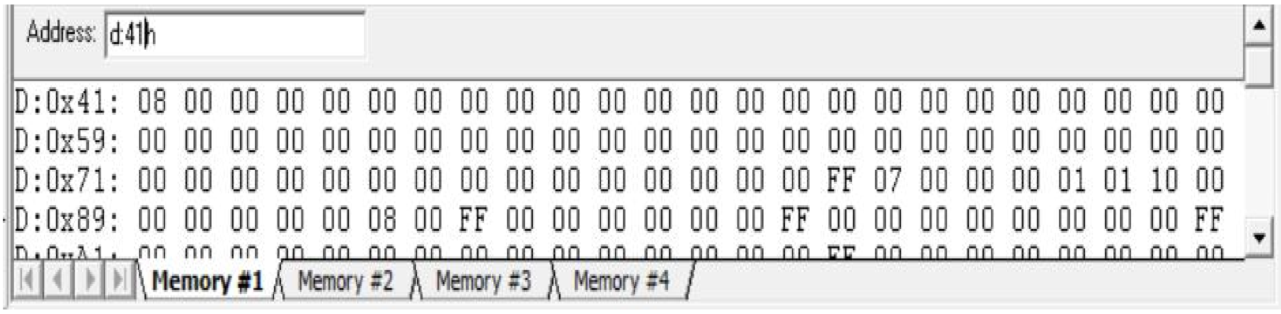
Before execution



Address: d:41h

```
D:0x41: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x59: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x71: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00
D:0x89: 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 FF
D:0xA1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
```

After execution

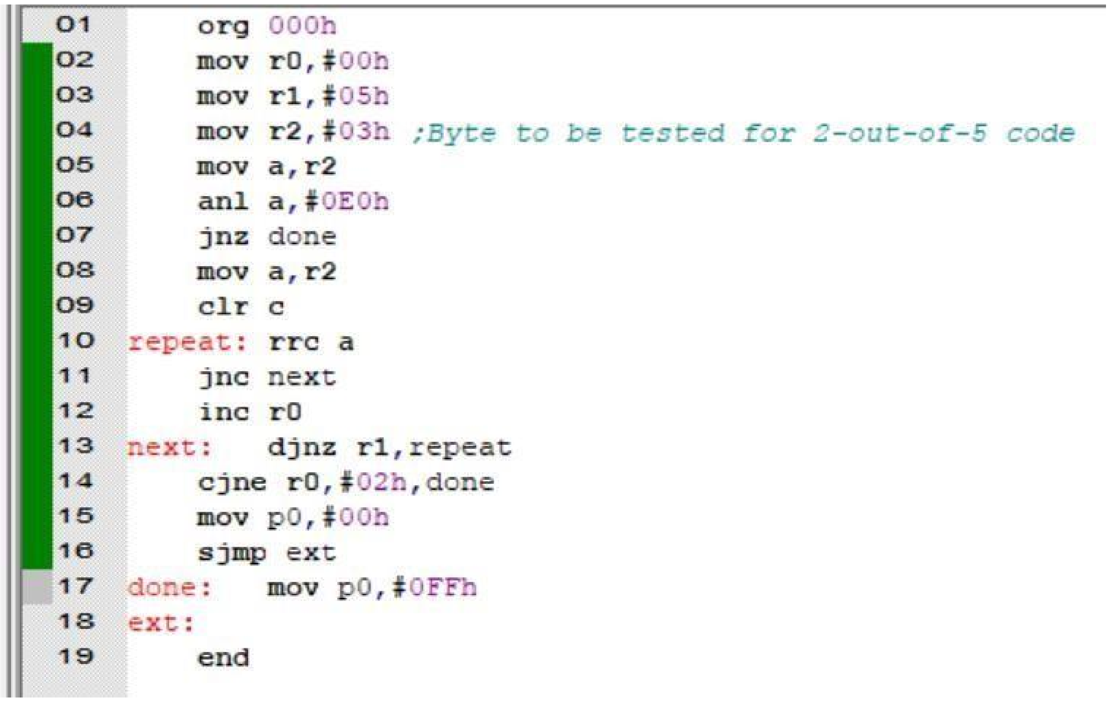


Address: d:41h

```
D:0x41: 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x59: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x71: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00
D:0x89: 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 FF
D:0xA1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
```

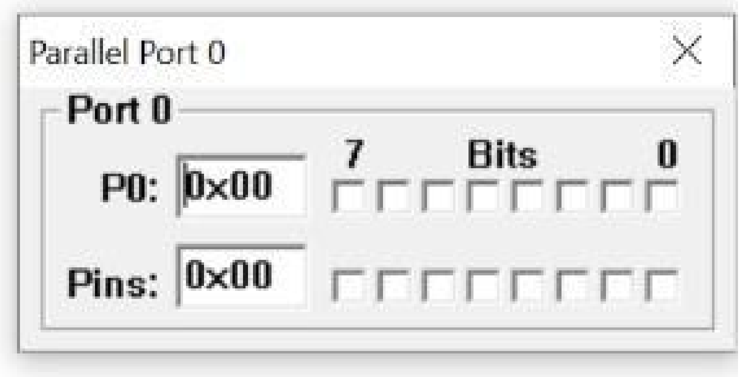
SET 6 PROGRAMS

6.1Write a assembly level program (8051) to check if the number is a 2 out of 5 code or not.



```
    org 000h
    mov r0,#00h
    mov r1,#05h
    mov r2,#03h ;Byte to be tested for 2-out-of-5 code
    mov a,r2
    anl a,#0E0h
    jnz done
    mov a,r2
    clr c
repeat: rrc a
    jnc next
    inc r0
next:   djnz r1,repeat
    cjne r0,#02h,done
    mov p0,#00h
    sjmp ext
done:   mov p0,#0FFh
ext:
    end
```

After Execution:



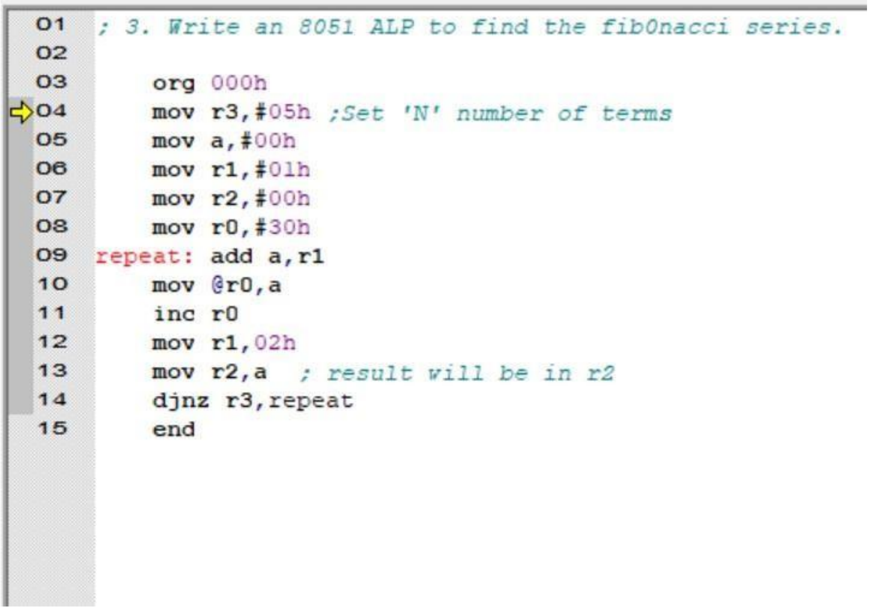
Parallel Port 0

Port 0

P0: 0x00 □□□□□□□□

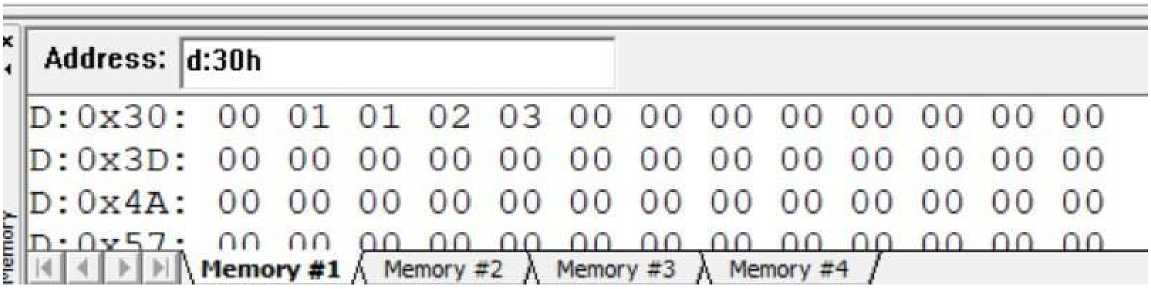
Pins: 0x00 □□□□□□□□

6.2 Write a assembly level program to find the Fibonacci series.



```
; 3. Write an 8051 ALP to find the fib0nacci series.

    org 000h
    mov r3,#05h ;Set 'N' number of terms
    mov a,#00h
    mov r1,#01h
    mov r2,#00h
    mov r0,#30h
repeat: add a,r1
    mov @r0,a
    inc r0
    mov r1,02h
    mov r2,a   ; result will be in r2
    djnz r3,repeat
    end
```

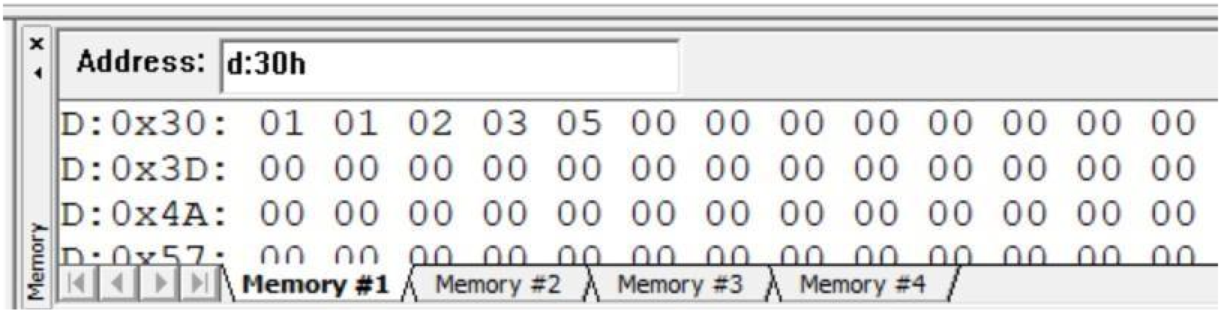
Before Execution:



Address: d:30h

```
D:0x30:  00  01  01  02  03  00  00  00  00  00  00  00  00
D:0x3D:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4A:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x57:  00  00  00  00  00  00  00  00  00  00  00  00  00
```

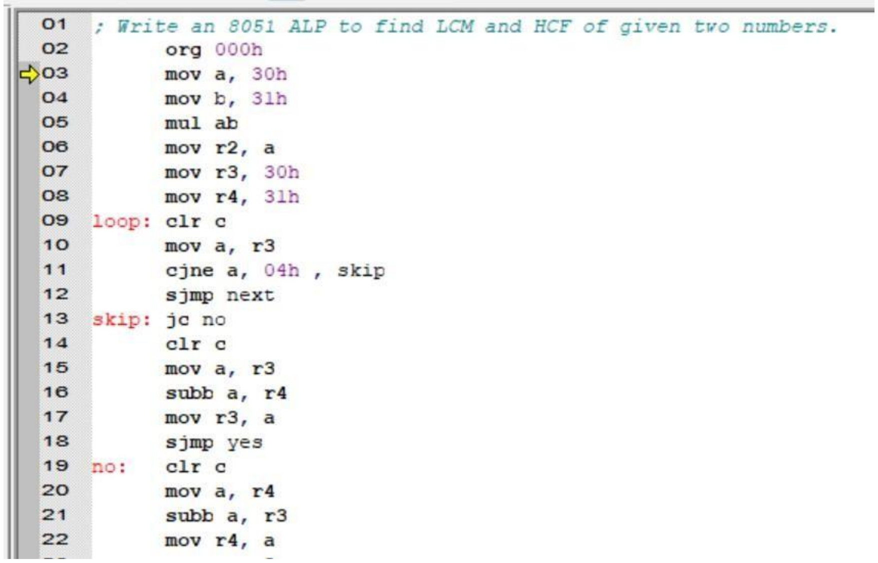
After Execution:



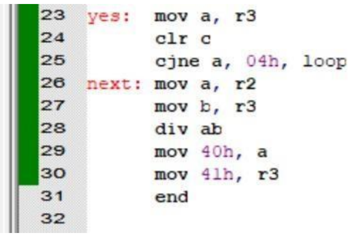
Address: d:30h

```
D:0x30:  01  01  02  03  05  00  00  00  00  00  00  00  00
D:0x3D:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4A:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x57:  00  00  00  00  00  00  00  00  00  00  00  00  00
```

6.3 Write an assembly level program to find LCM and HCF of 2 numbers.

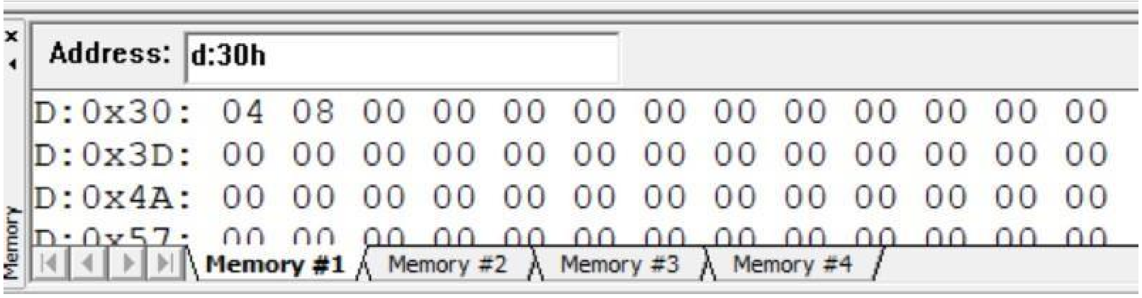


```
; Write an 8051 ALP to find LCM and HCF of given two numbers.
      org 000h
      mov a, 30h
      mov b, 31h
      mul ab
      mov r2, a
      mov r3, 30h
      mov r4, 31h
loop: clr c
      mov a, r3
      cjne a, 04h , skip
      sjmp next
skip: jc no
      clr c
      mov a, r3
      subb a, r4
      mov r3, a
      sjmp yes
no:   clr c
      mov a, r4
      subb a, r3
      mov r4, a
```



```
yes:  mov a, r3
      clr c
      cjne a, 04h, loop
next: mov a, r2
      mov b, r3
      div ab
      mov 40h, a
      mov 41h, r3
      end
```

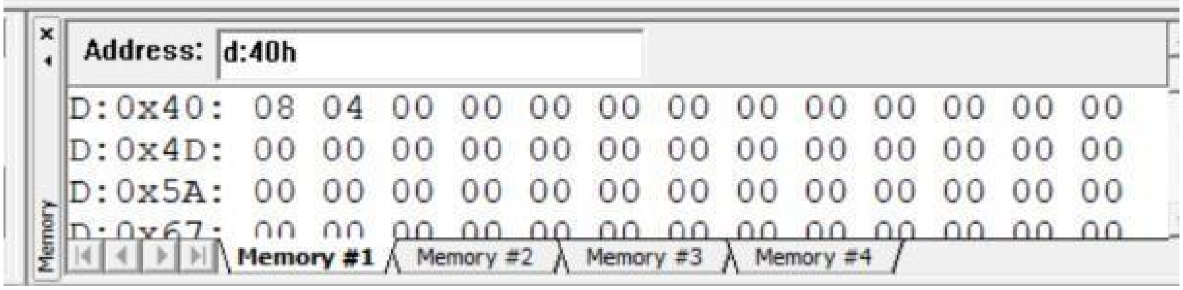
Before Execution:



Address: d:30h

```
D:0x30:  04  08  00  00  00  00  00  00  00  00  00  00  00
D:0x3D:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x4A:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x57:  00  00  00  00  00  00  00  00  00  00  00  00  00
```

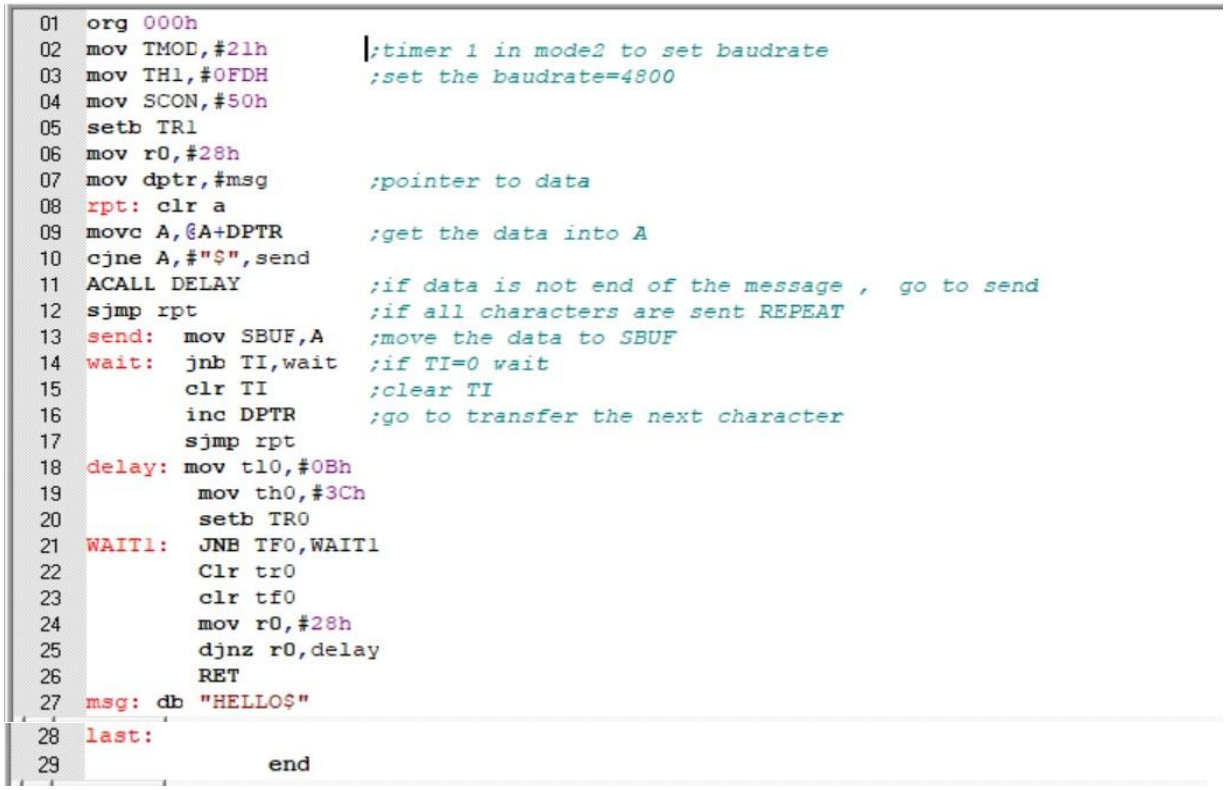
After Execution:



Address: d:40h

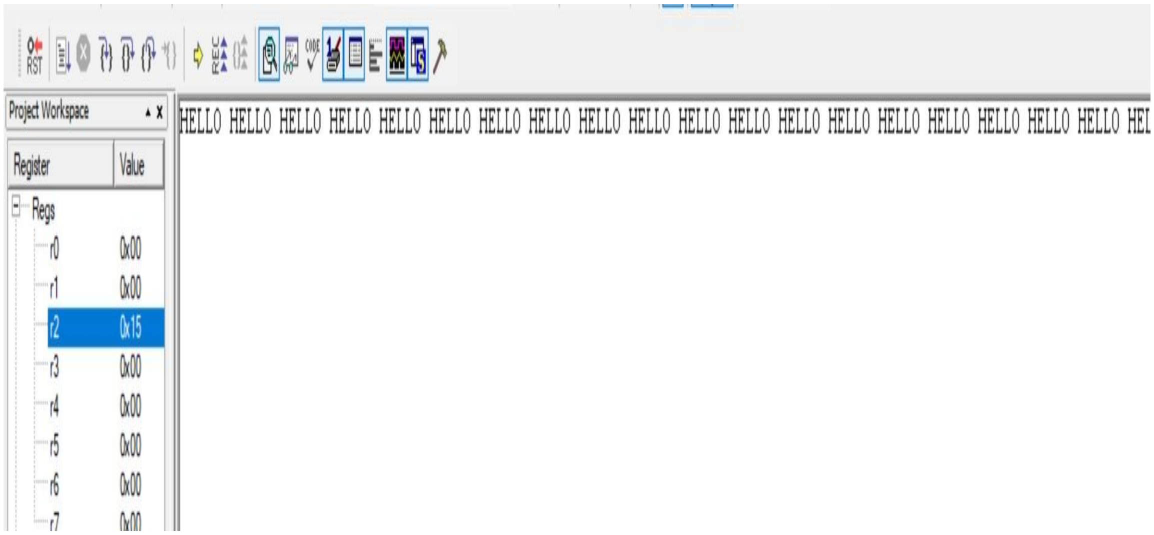
```
D:0x40:  08  04  00  00  00  00  00  00  00  00  00  00  00
D:0x4D:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x5A:  00  00  00  00  00  00  00  00  00  00  00  00  00
D:0x67:  00  00  00  00  00  00  00  00  00  00  00  00  00
```

6.4 Write an 8051 ALP to send message HELLO serially once in every 2 sec.Program timer and serial communication in interrupt mode



```
org 000h
mov TMOD,#21h       ;timer 1 in mode2 to set baudrate
mov TH1,#0FDH       ;set the baudrate=4800
mov SCON,#50h
setb TR1
mov r0,#28h
mov dptr,#msg       ;pointer to data
rpt: clr a
movc A,@A+DPTR      ;get the data into A
cjne A,#"$",send
ACALL DELAY         ;if data is not end of the message ,  go to send
sjmp rpt            ;if all characters are sent REPEAT
send:  mov SBUF,A   ;move the data to SBUF
wait:  jnb TI,wait  ;if TI=0 wait
       clr TI       ;clear TI
       inc DPTR     ;go to transfer the next character
       sjmp rpt
delay: mov tl0,#0Bh
        mov th0,#3Ch
        setb TR0
WAIT1:  JNB TF0,WAIT1
        Clr tr0
        clr tf0
        mov r0,#28h
        djnz r0,delay
        RET
msg: db "HELLO$"
last:
             end
```

RESULT:



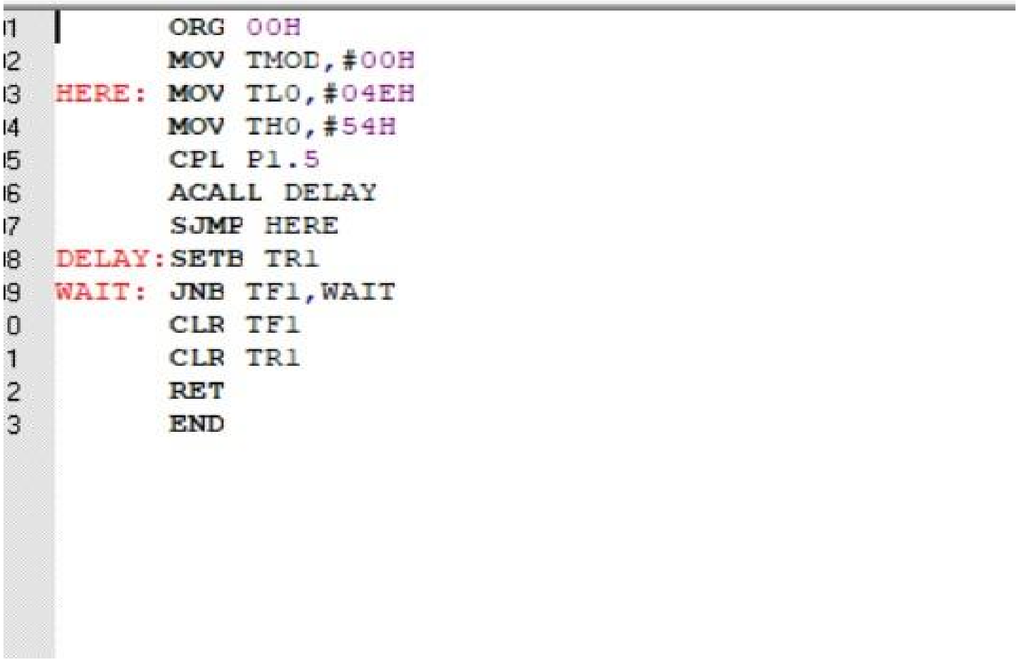
```
HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HELLO HEL
```

```
r0   0x00
r1   0x00
r2   0x15
r3   0x00
r4   0x00
r5   0x00
r6   0x00
r7   0x00
```

SET 7 PROGRAMS

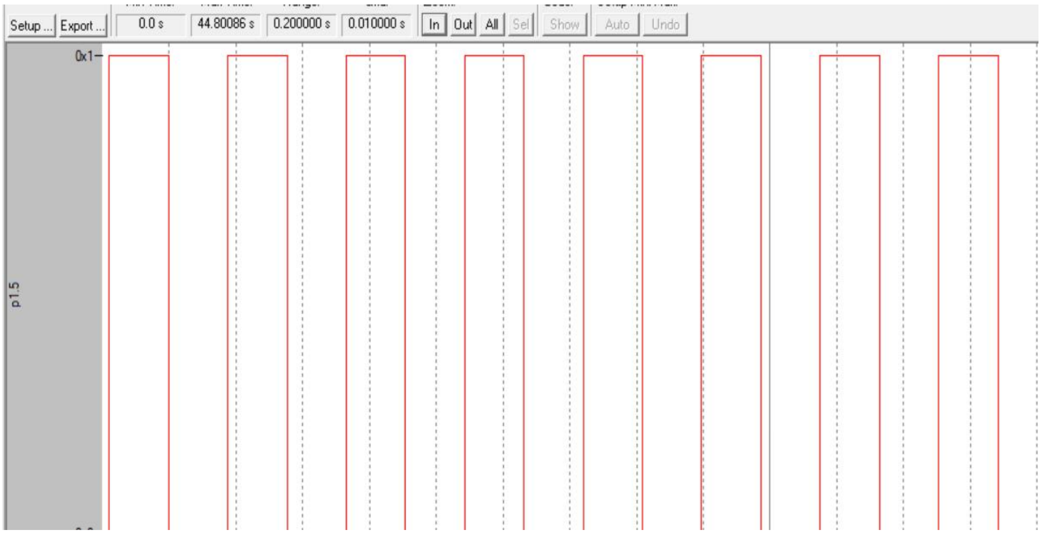
7.1 WAP to generate a square wave on P1.5 with 50% Duty Cycle. Program timers in mode0, mode1 and mode2 to generate delay

Mode 0

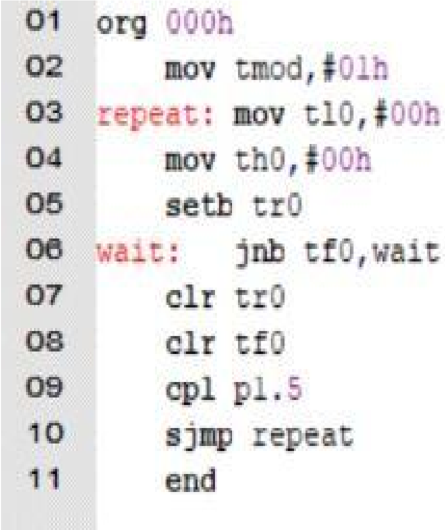


```
        ORG 00H
        MOV TMOD,#00H
HERE:   MOV TL0,#04EH
        MOV TH0,#54H
        CPL P1.5
        ACALL DELAY
        SJMP HERE
DELAY:SETB TR1
WAIT:  JNB TF1,WAIT
        CLR TF1
        CLR TR1
        RET
        END
```

After Execution

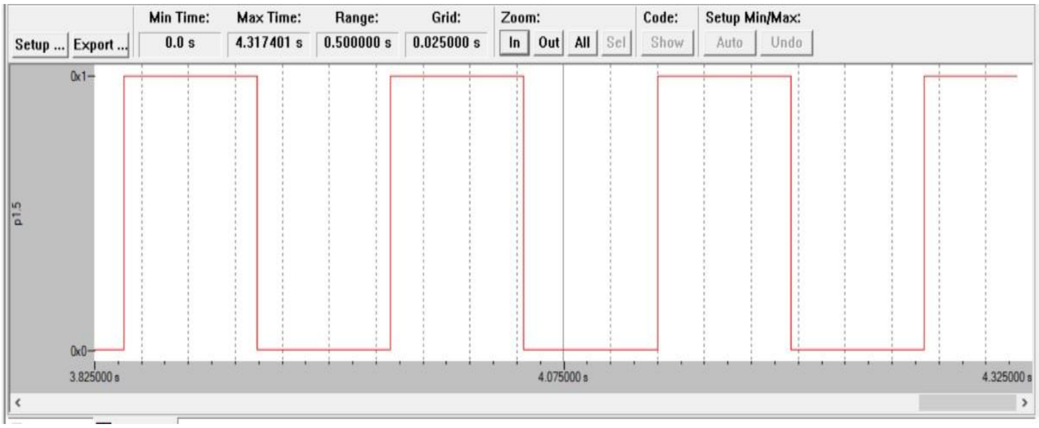


Mode 1



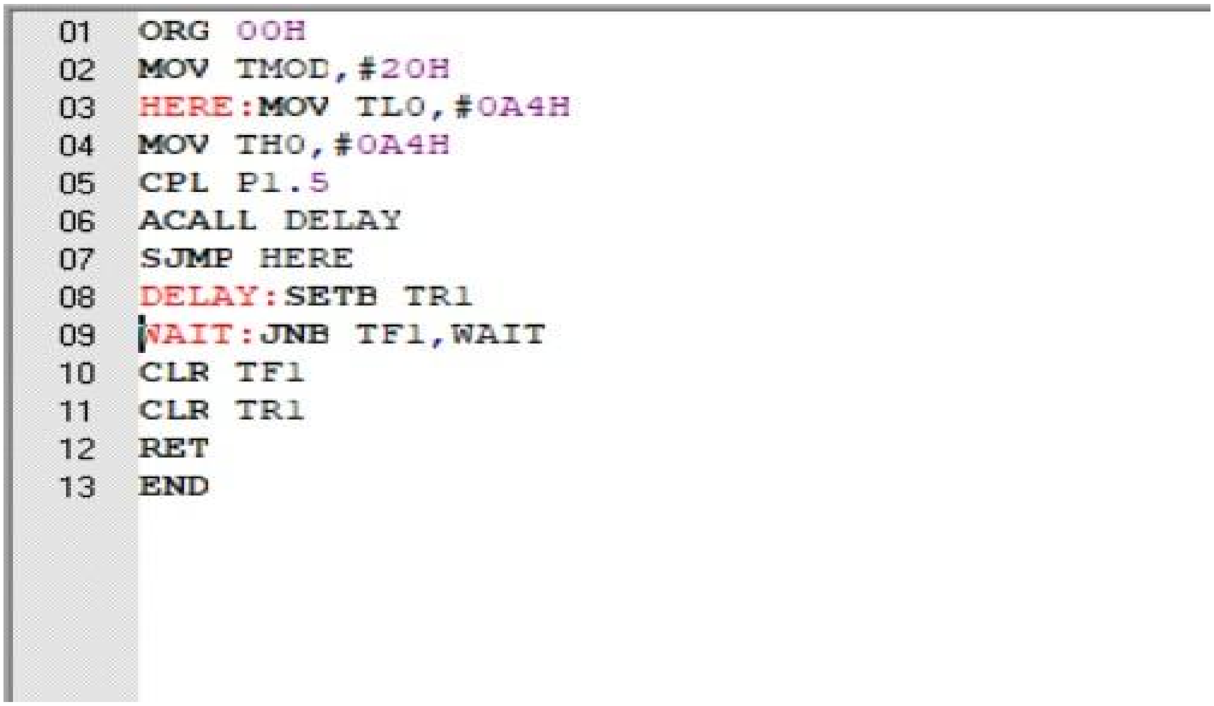
```
org 000h
    mov tmod,#01h
repeat: mov tl0,#00h
    mov th0,#00h
    setb tr0
wait:   jnb tf0,wait
    clr tr0
    clr tf0
    cpl p1.5
    sjmp repeat
    end
```

After Execution



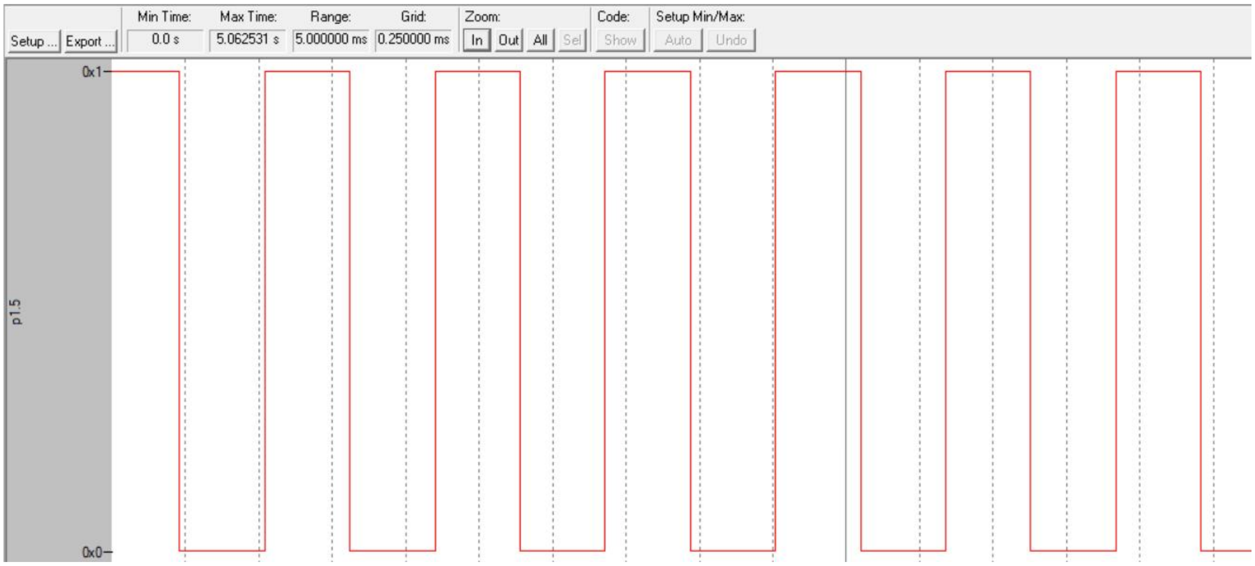
Min Time: 0.0 s
Max Time: 4.317401 s
Range: 0.500000 s
Grid: 0.025000 s

Mode 2



```
ORG 00H
MOV TMOD,#20H
HERE:MOV TL0,#0A4H
MOV TH0,#0A4H
CPL P1.5
ACALL DELAY
SJMP HERE
DELAY:SETB TR1
WAIT:JNB TF1,WAIT
CLR TF1
CLR TR1
RET
END
```

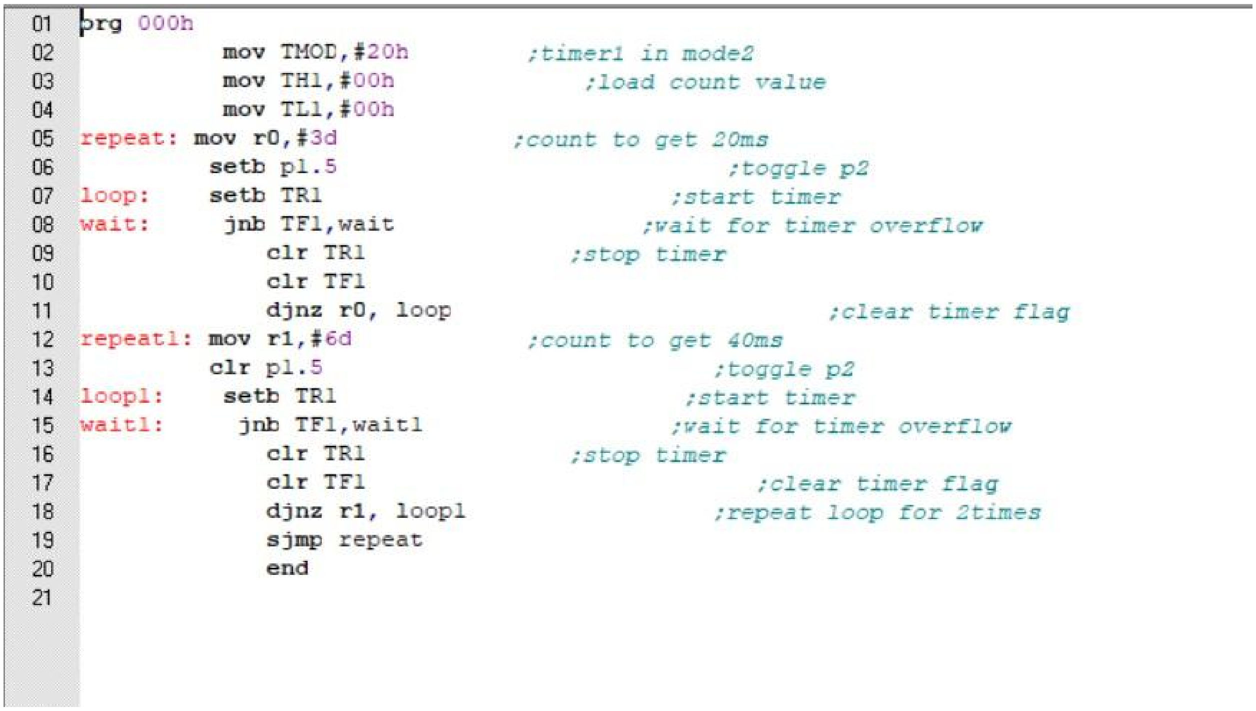
After Execution



Min Time: 0.0 s
Max Time: 5.062531 s
Range: 5.000000 ms
Grid: 0.250000 ms

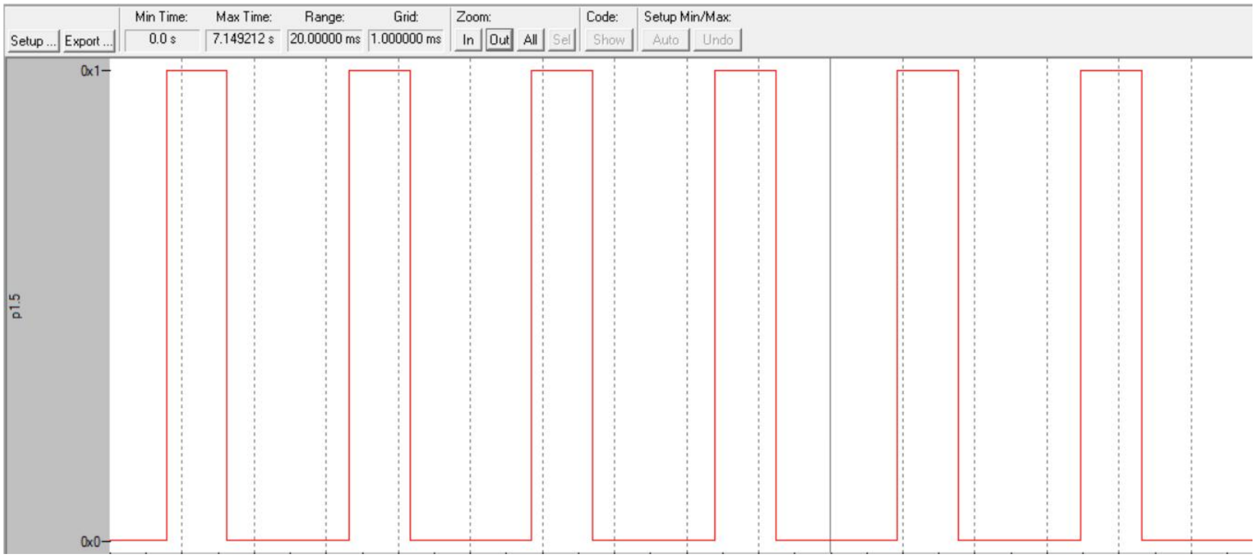
7.3 WAP to generate a rectangular wave with T(on)=20ms and T(off)=40ms. Use Timer0 in mode0, mode1 and mode2 to generate delay.

Mode 0



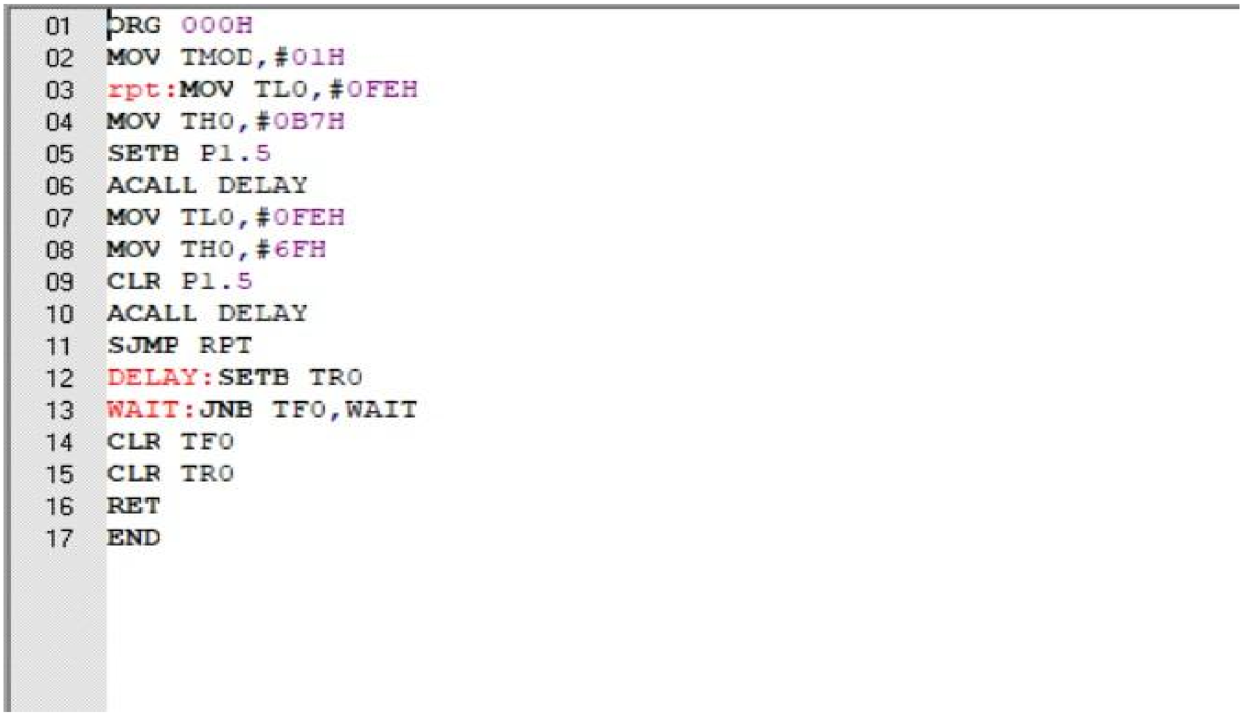
```
org 000h
          mov TMOD,#20h        ;timer1 in mode2
          mov TH1,#00h             ;load count value
          mov TL1,#00h
repeat: mov r0,#3d            ;count to get 20ms
         setb p1.5                            ;toggle p2
loop:    setb TR1                          ;start timer
wait:     jnb TF1,wait                   ;wait for timer overflow
            clr TR1               ;stop timer
            clr TF1
            djnz r0, loop                                ;clear timer flag
repeat1: mov r1,#6d           ;count to get 40ms
         clr p1.5                            ;toggle p2
loop1:    setb TR1                          ;start timer
wait1:     jnb TF1,wait1                    ;wait for timer overflow
            clr TR1               ;stop timer
            clr TF1                               ;clear timer flag
            djnz r1, loop1                      ;repeat loop for 2times
            sjmp repeat
            end
```

After Execution



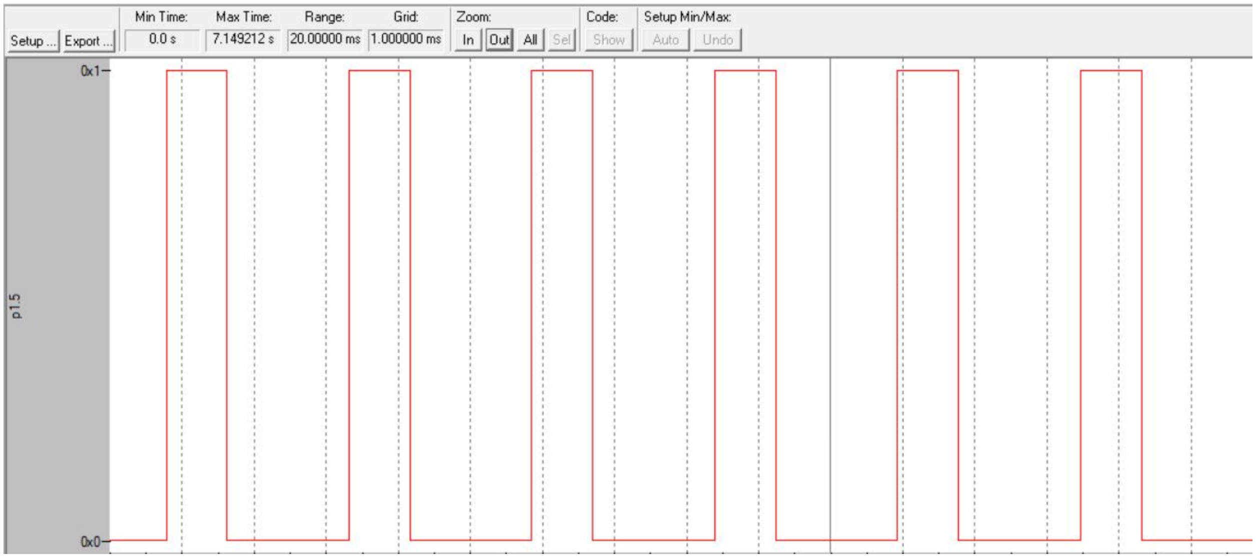
Min Time: 0.0 s
Max Time: 7.149212 s
Range: 20.00000 ms
Grid: 1.000000 ms

Mode 1



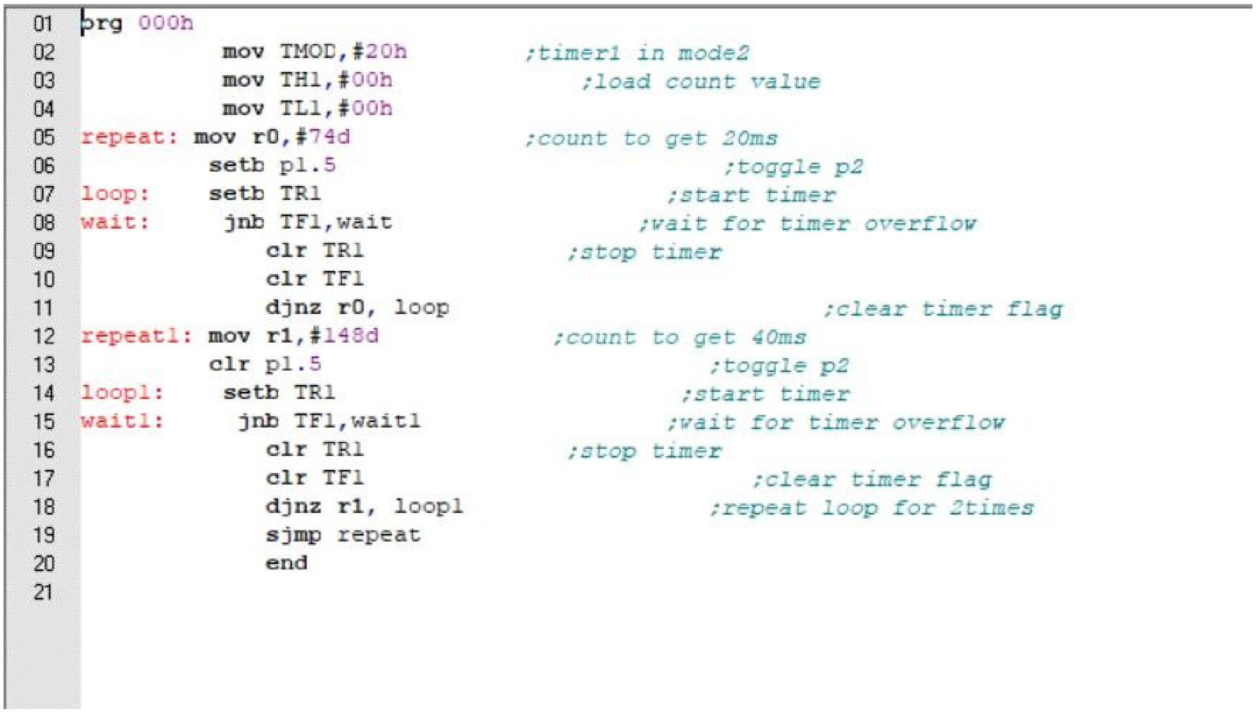
```
ORG 000H
MOV TMOD,#01H
rpt:MOV TL0,#0FEH
MOV TH0,#0B7H
SETB P1.5
ACALL DELAY
MOV TL0,#0FEH
MOV TH0,#6FH
CLR P1.5
ACALL DELAY
SJMP RPT
DELAY:SETB TR0
WAIT:JNB TF0,WAIT
CLR TF0
CLR TR0
RET
END
```

After Execution



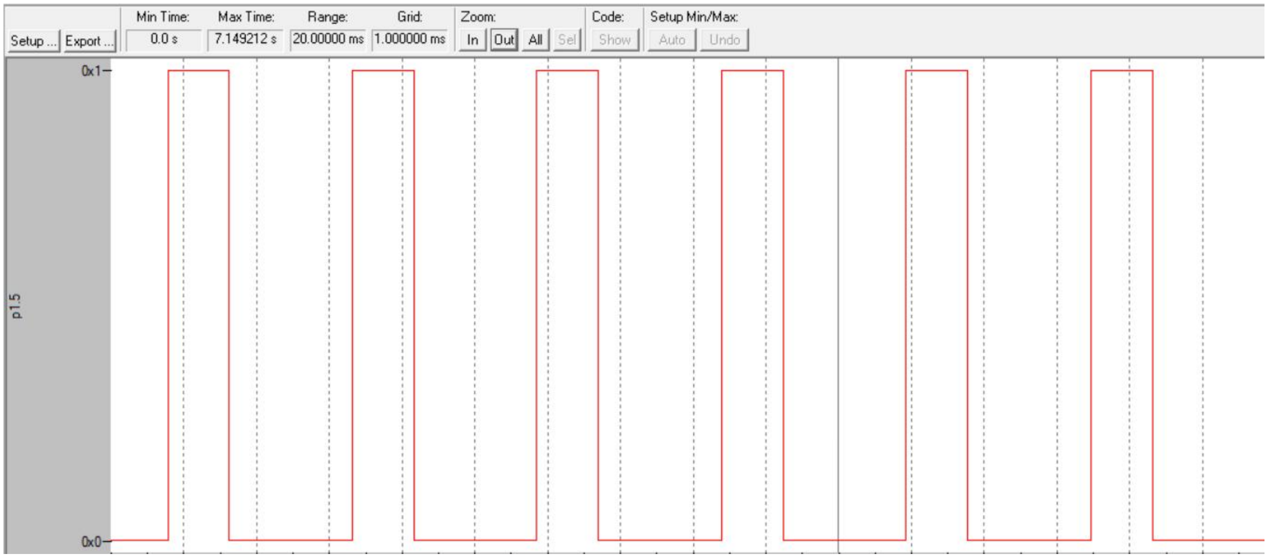
Min Time: 0.0 s
Max Time: 7.149212 s
Range: 20.00000 ms
Grid: 1.000000 ms

Mode 2



```
org 000h
          mov TMOD,#20h       ;timer1 in mode2
          mov TH1,#00h            ;load count value
          mov TL1,#00h
repeat: mov r0,#74d           ;count to get 20ms
         setb p1.5                          ;toggle p2
loop:    setb TR1                      ;start timer
wait:     jnb TF1,wait                ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1
             djnz r0, loop                          ;clear timer flag
repeat1: mov r1,#148d          ;count to get 40ms
         clr p1.5                          ;toggle p2
loop1:    setb TR1                      ;start timer
wait1:     jnb TF1,wait1                ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1                          ;clear timer flag
             djnz r1, loop1                ;repeat loop for 2times
             sjmp repeat
             end
```

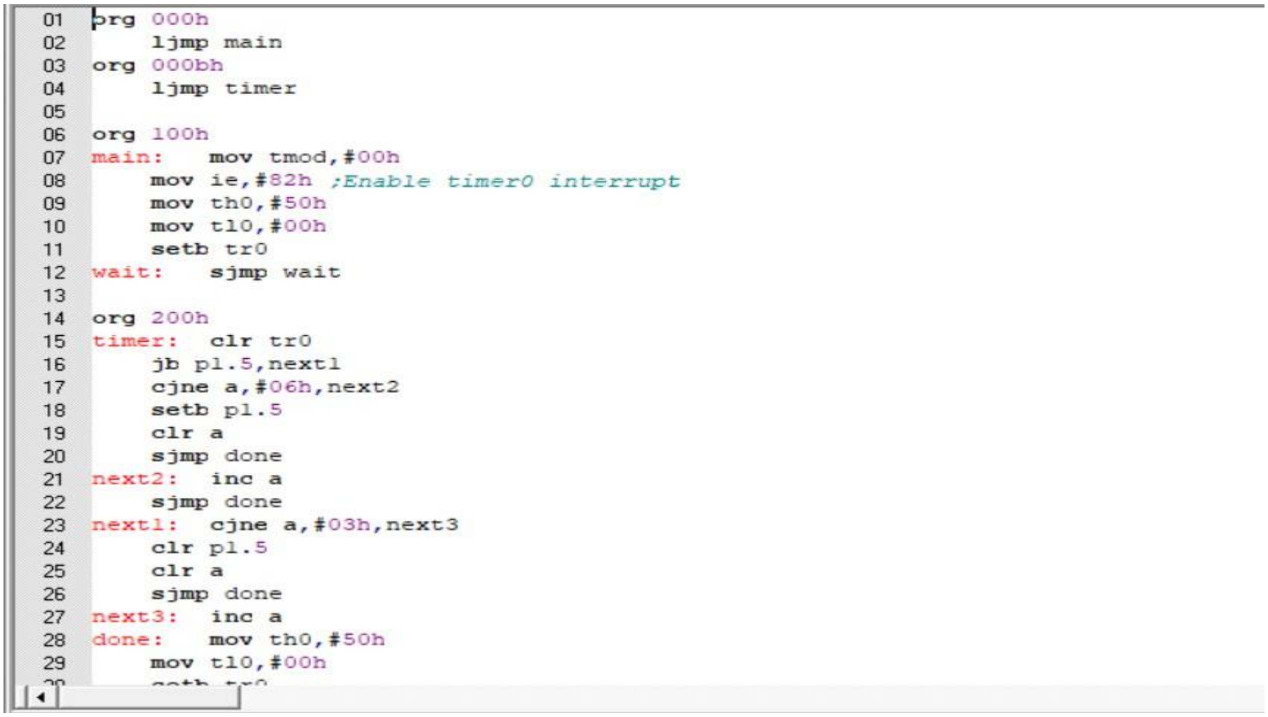
After Execution



Min Time: 0.0 s
Max Time: 7.149212 s
Range: 20.00000 ms
Grid: 1.000000 ms

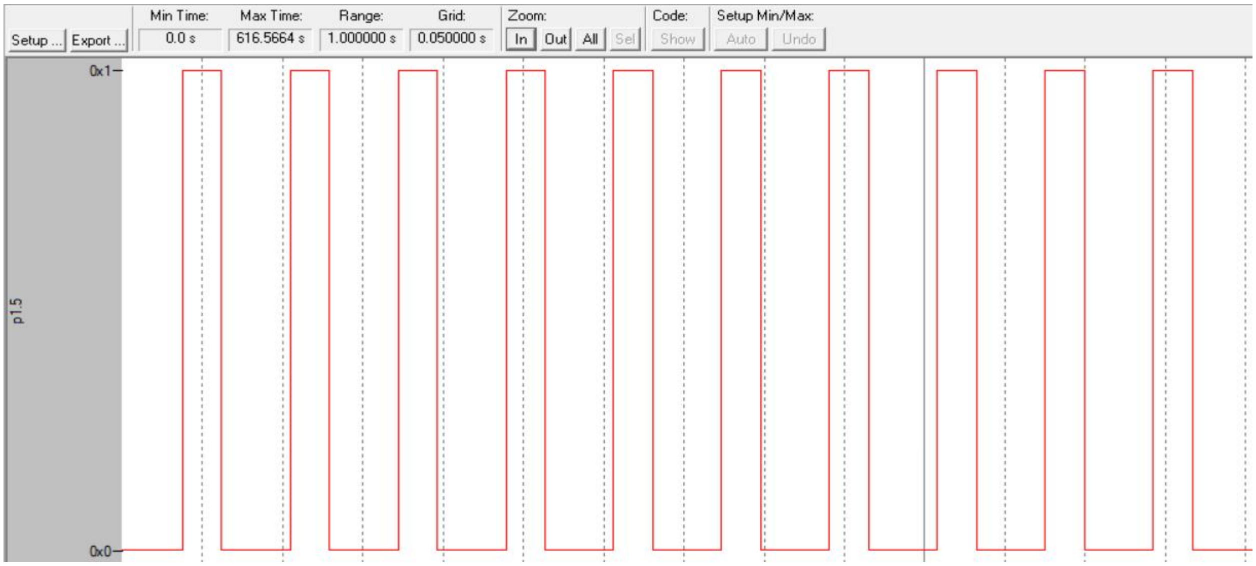
7.4 Repeat above problems to generate square wave with interrupts for timers

Mode 0



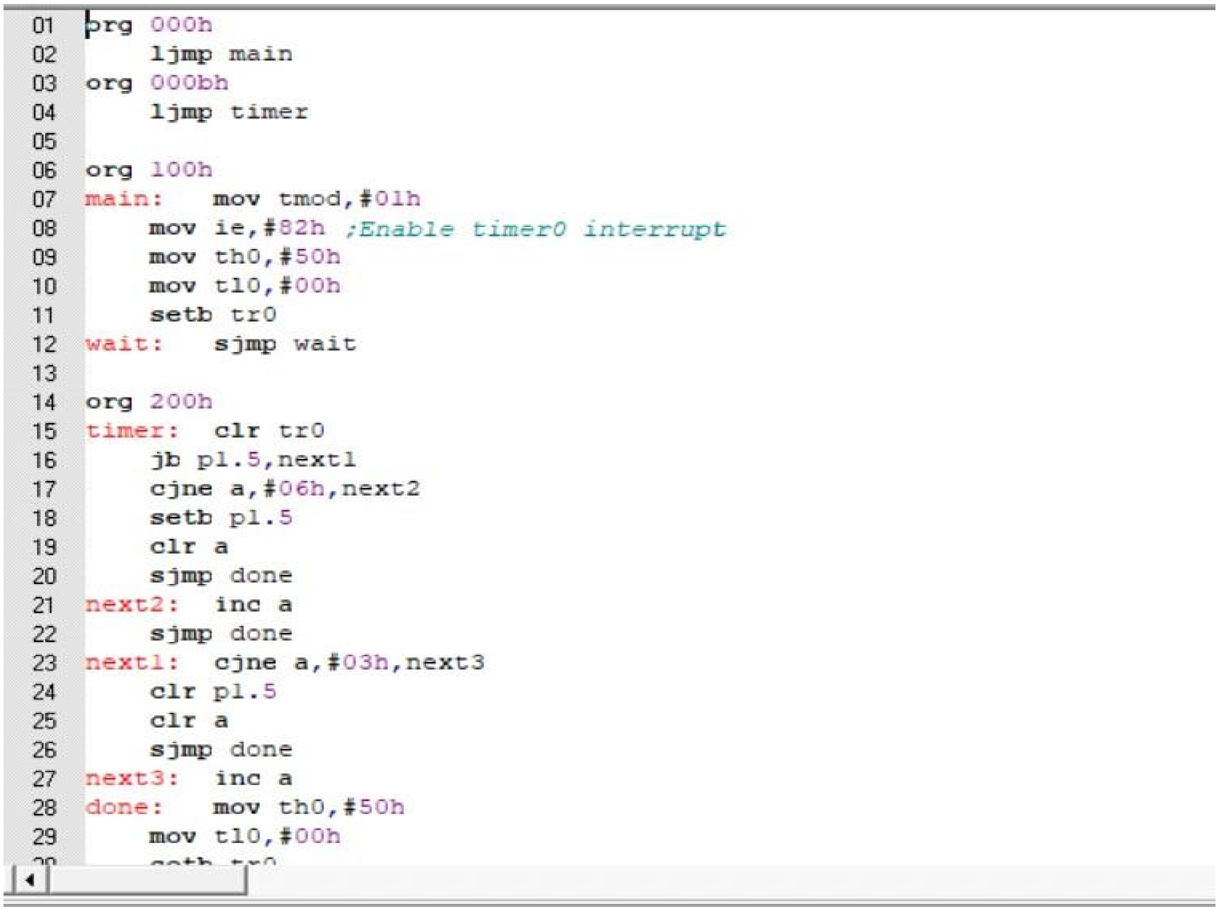
```
org 000h
    ljmp main
org 000bh
    ljmp timer

org 100h
main:   mov tmod,#00h
    mov ie,#82h ;Enable timer0 interrupt
    mov th0,#50h
    mov tl0,#00h
    setb tr0
wait:   sjmp wait

org 200h
timer:  clr tr0
    jb p1.5,next1
    cjne a,#06h,next2
    setb p1.5
    clr a
    sjmp done
next2:  inc a
    sjmp done
next1:  cjne a,#03h,next3
    clr p1.5
    clr a
    sjmp done
next3:  inc a
done:   mov th0,#50h
    mov tl0,#00h
```

After Execution

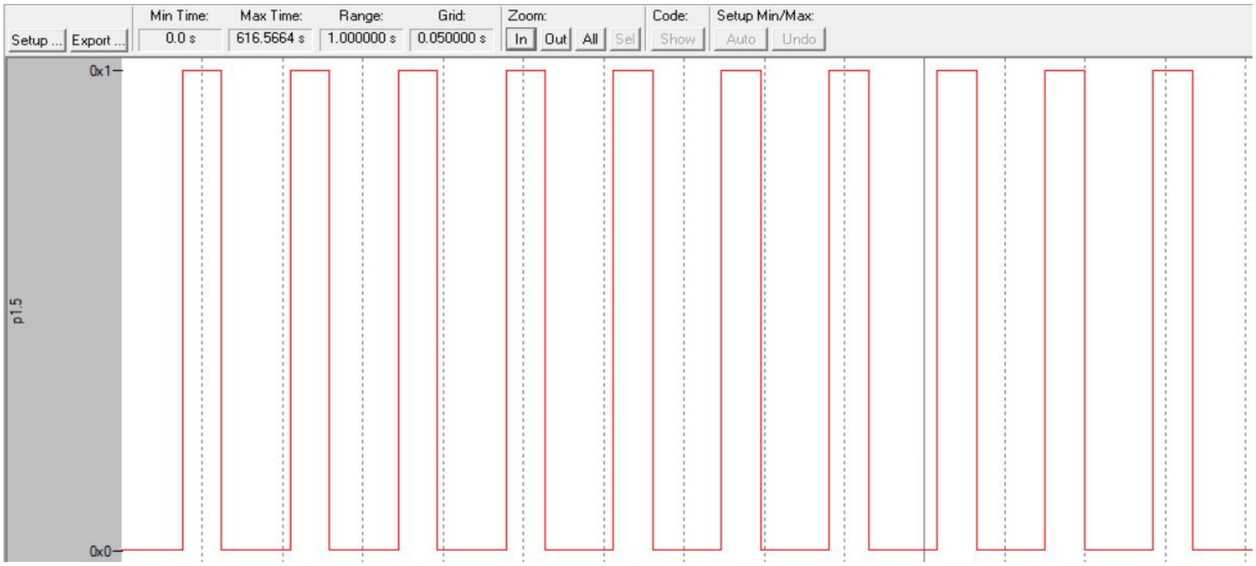


Min Time: 0.0 s Max Time: 616.5664 s Range: 1.000000 s Grid: 0.050000 s

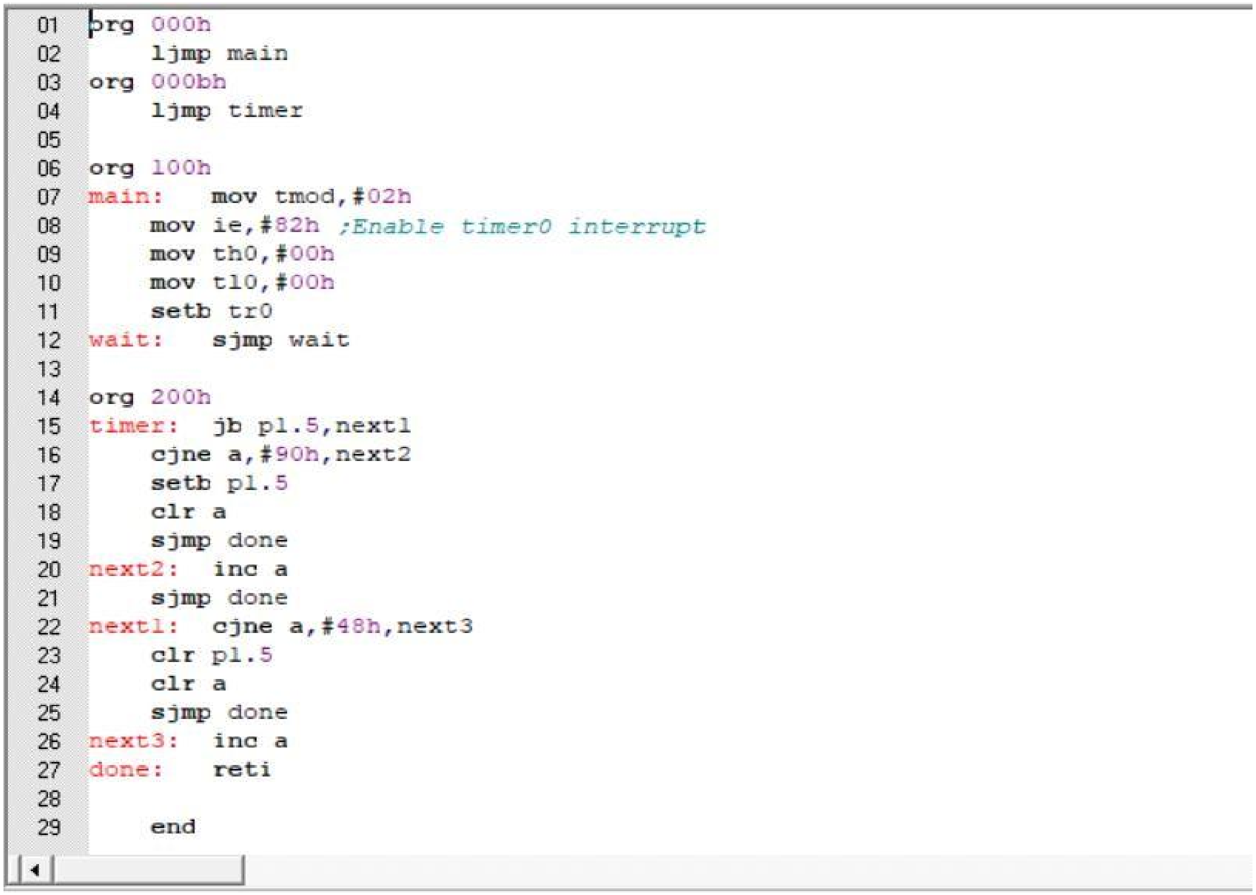
Mode 1



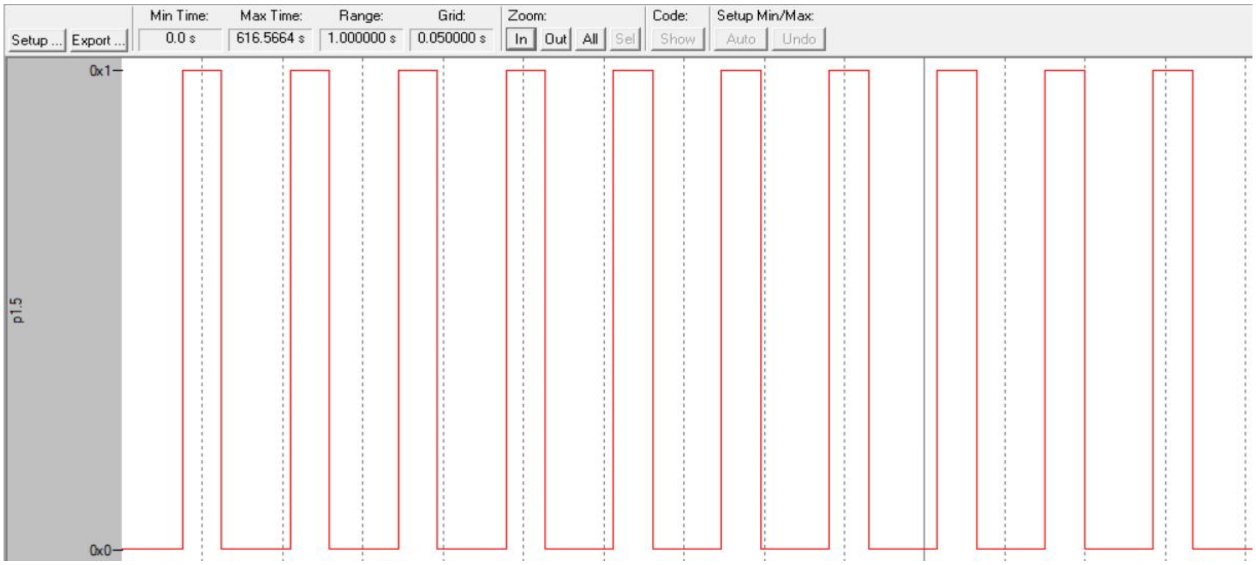
```
org 000h
    ljmp main
org 000bh
    ljmp timer

org 100h
main:   mov tmod,#01h
    mov ie,#82h ;Enable timer0 interrupt
    mov th0,#50h
    mov tl0,#00h
    setb tr0
wait:   sjmp wait

org 200h
timer:  clr tr0
    jb p1.5,next1
    cjne a,#06h,next2
    setb p1.5
    clr a
    sjmp done
next2:  inc a
    sjmp done
next1:  cjne a,#03h,next3
    clr p1.5
    clr a
    sjmp done
next3:  inc a
done:   mov th0,#50h
    mov tl0,#00h
    setb tr0
```



Min Time: 0.0 s
Max Time: 616.5664 s
Range: 1.000000 s
Grid: 0.050000 s



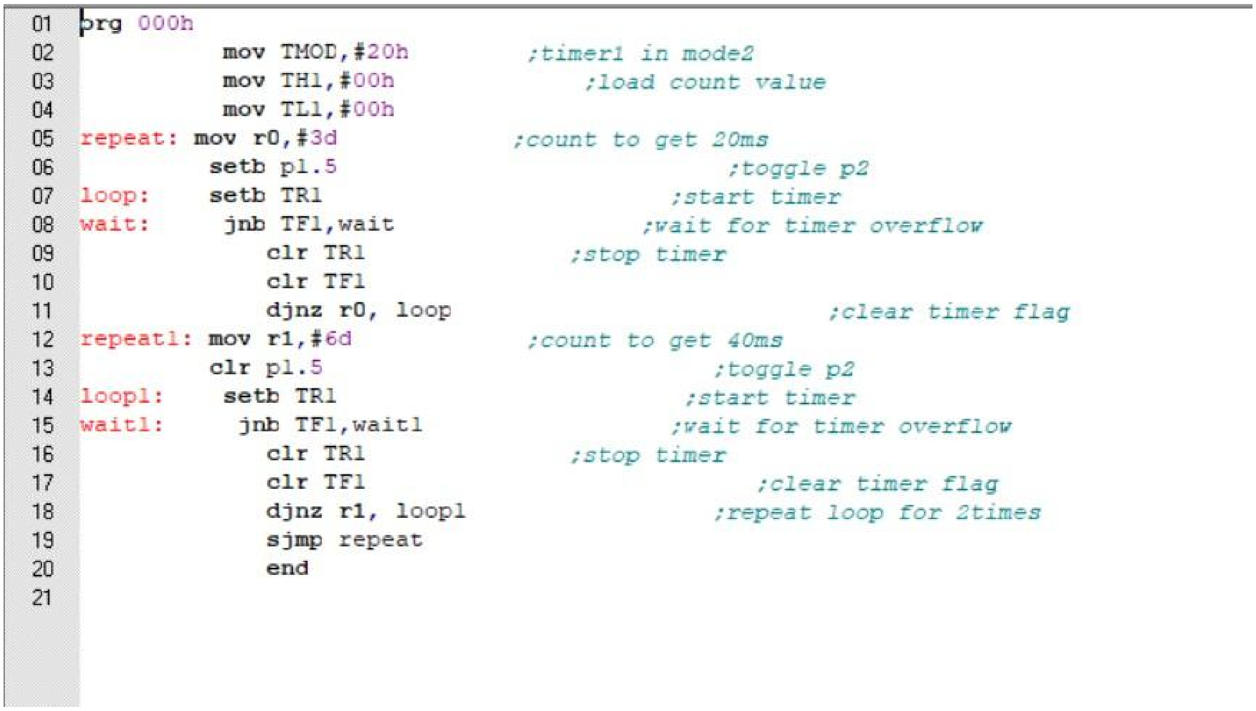
```
org 000h
    ljmp main
org 000bh
    ljmp timer

org 100h
main:   mov tmod,#02h
    mov ie,#82h ;Enable timer0 interrupt
    mov th0,#00h
    mov tl0,#00h
    setb tr0
wait:   sjmp wait

org 200h
timer:  jb p1.5,next1
    cjne a,#90h,next2
    setb p1.5
    clr a
    sjmp done
next2:  inc a
    sjmp done
next1:  cjne a,#48h,next3
    clr p1.5
    clr a
    sjmp done
next3:  inc a
done:   reti

    end
```



Min Time: 0.0 s
Max Time: 616.5664 s
Range: 1.000000 s
Grid: 0.050000 s

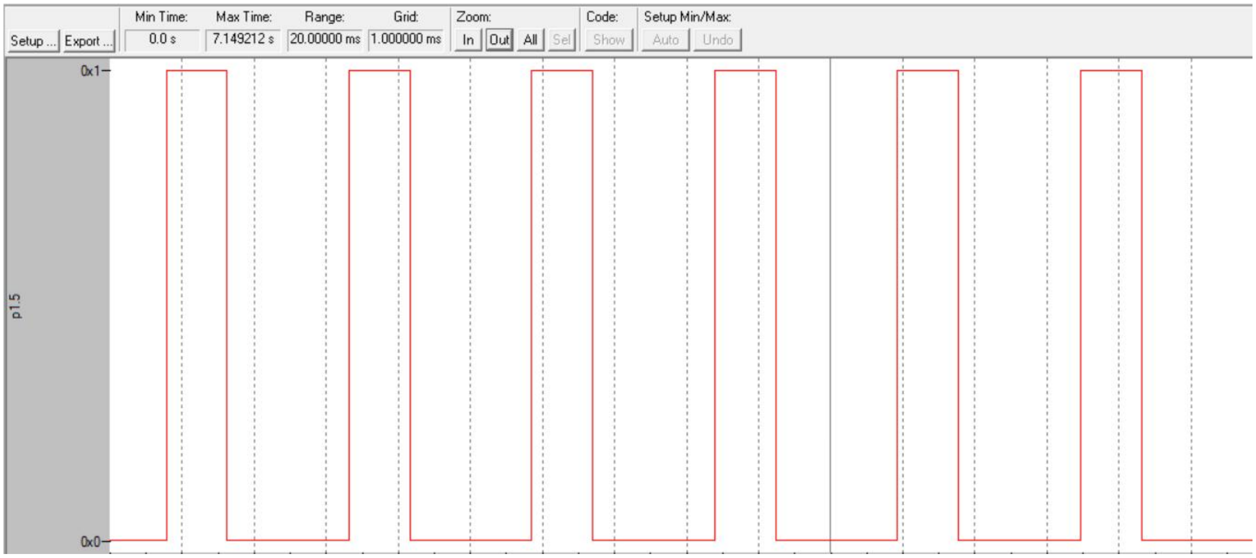
1. WAP to generate a rectangular wave with T(on)=20ms and T(off)=40ms. Use Timer0 in mode0, mode1 and mode2 to generate delay.

Mode 0



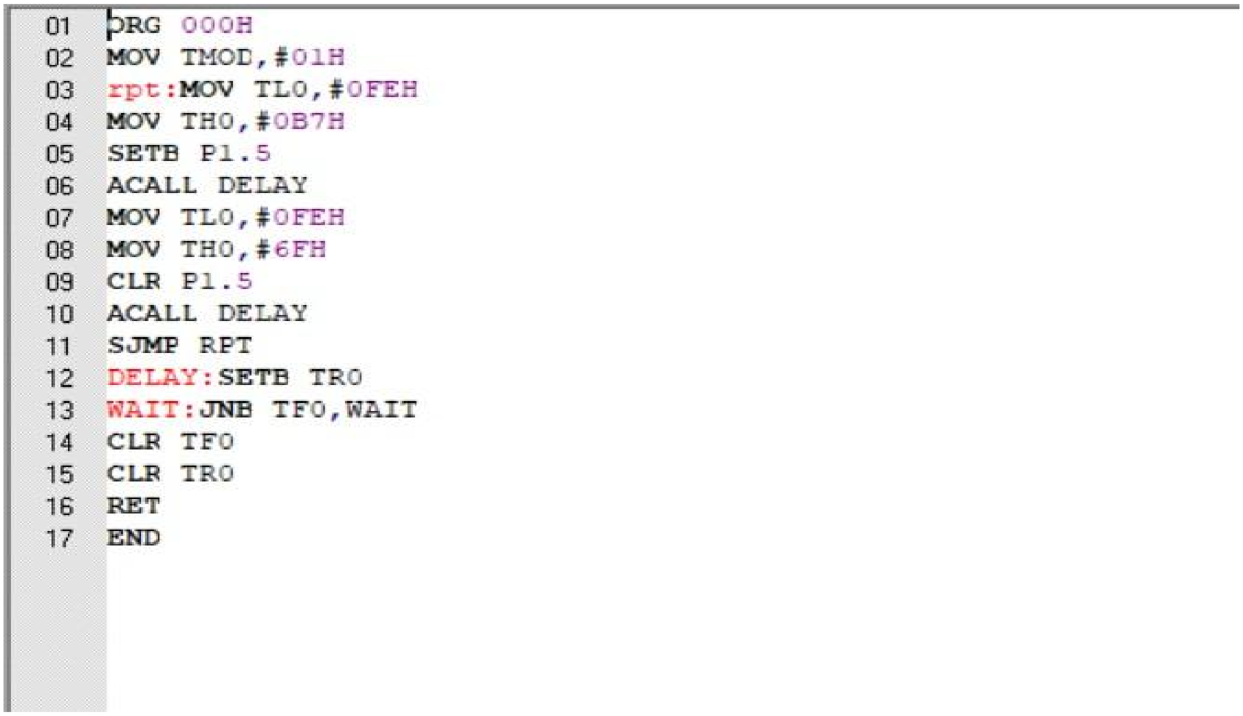
```
org 000h
          mov TMOD,#20h         ;timer1 in mode2
          mov TH1,#00h              ;load count value
          mov TL1,#00h
repeat: mov r0,#3d             ;count to get 20ms
         setb p1.5                            ;toggle p2
loop:    setb TR1                         ;start timer
wait:     jnb TF1,wait                  ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1
             djnz r0, loop                                 ;clear timer flag
repeat1: mov r1,#6d            ;count to get 40ms
         clr p1.5                            ;toggle p2
loop1:    setb TR1                        ;start timer
wait1:     jnb TF1,wait1                 ;wait for timer overflow
             clr TR1              ;stop timer
             clr TF1                             ;clear timer flag
             djnz r1, loop1                  ;repeat loop for 2times
             sjmp repeat
             end
```

After Execution:



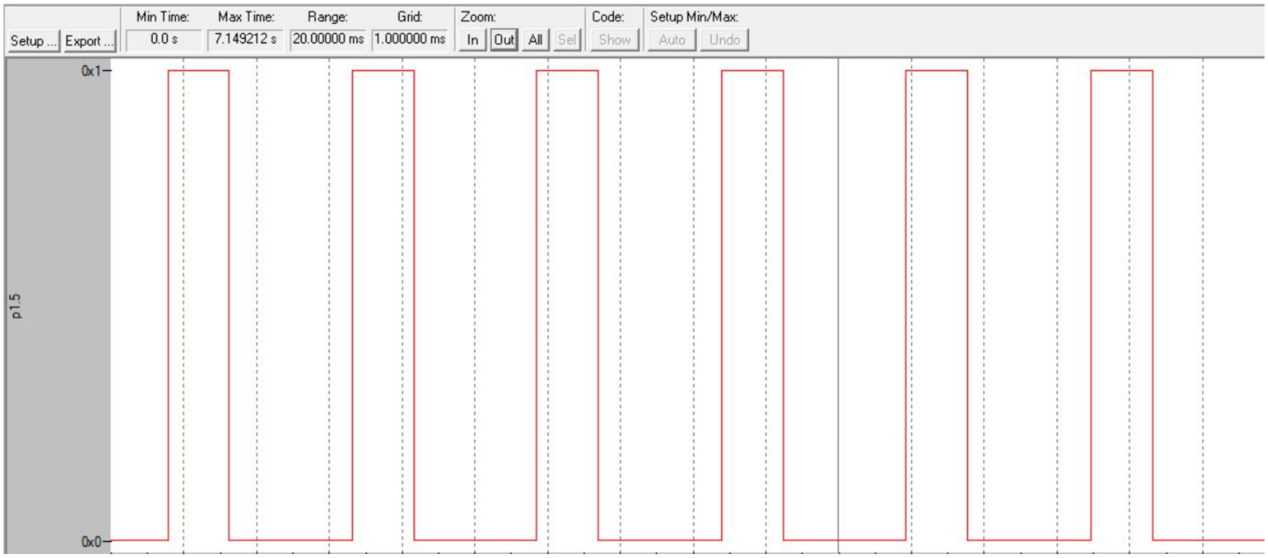
Min Time: 0.0 s
Max Time: 7.149212 s
Range: 20.00000 ms
Grid: 1.000000 ms

Mode 1



```
ORG 000H
MOV TMOD,#01H
rpt:MOV TL0,#0FEH
MOV TH0,#0B7H
SETB P1.5
ACALL DELAY
MOV TL0,#0FEH
MOV TH0,#6FH
CLR P1.5
ACALL DELAY
SJMP RPT
DELAY:SETB TR0
WAIT:JNB TF0,WAIT
CLR TF0
CLR TR0
RET
END
```

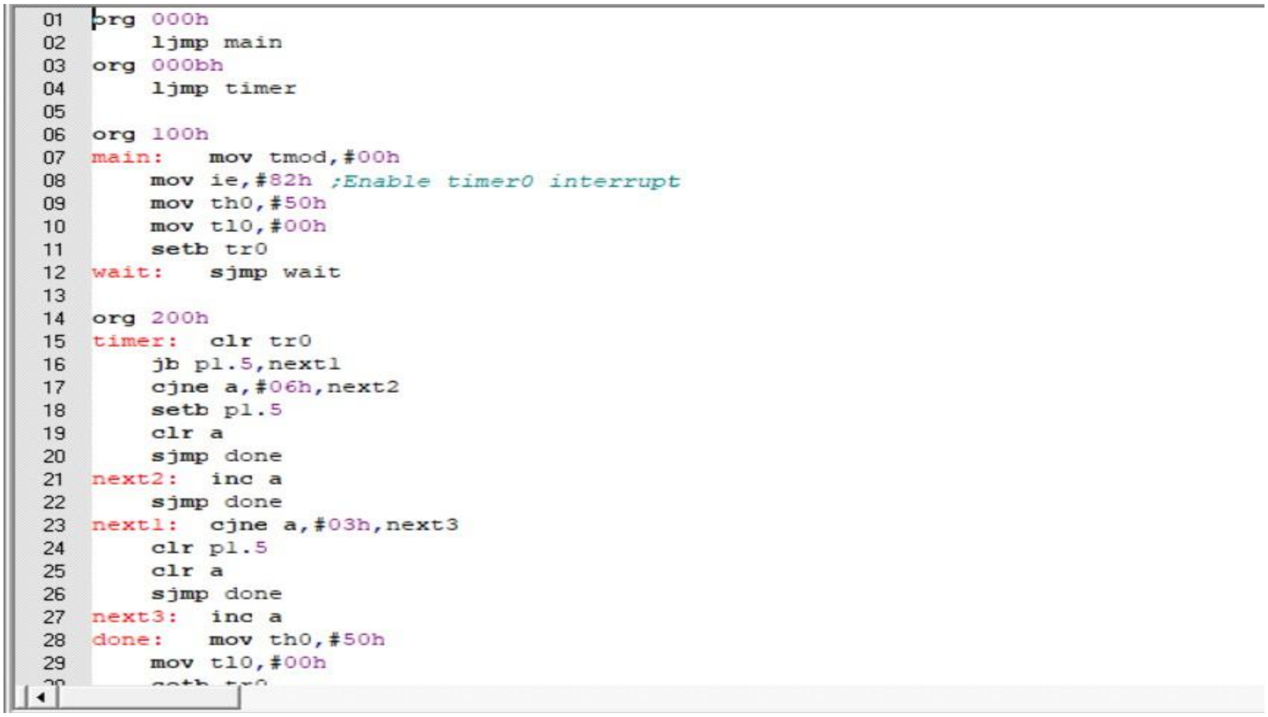
After execution



Min Time: 0.0 s
Max Time: 7.149212 s
Range: 20.00000 ms
Grid: 1.000000 ms

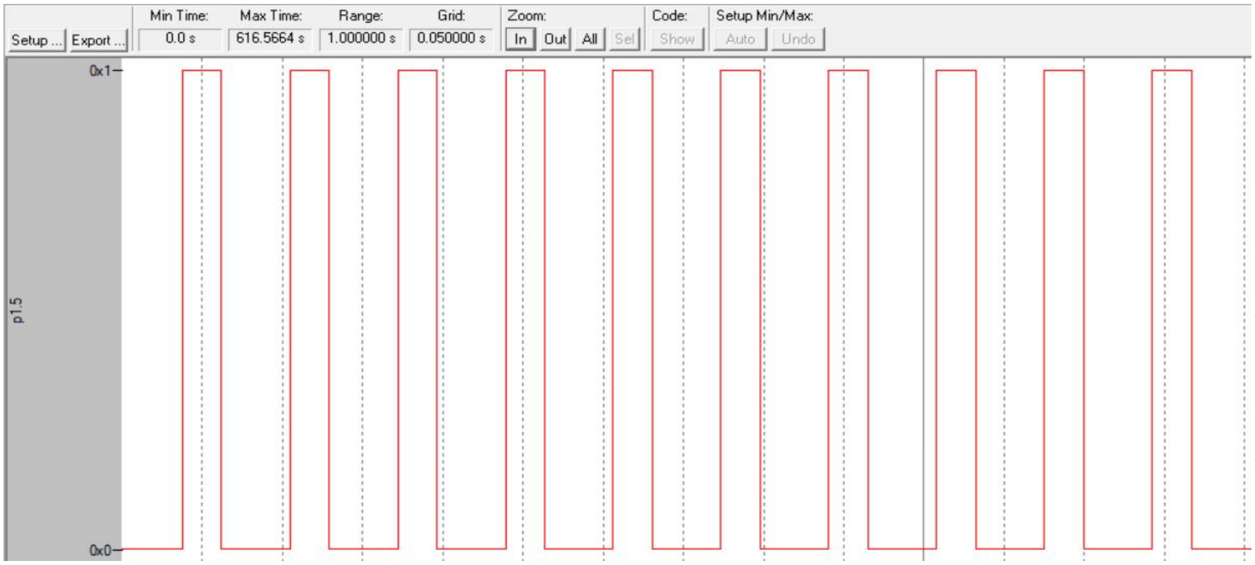
Repeat above problems to generate square wave with interrupts for timers

Mode 0



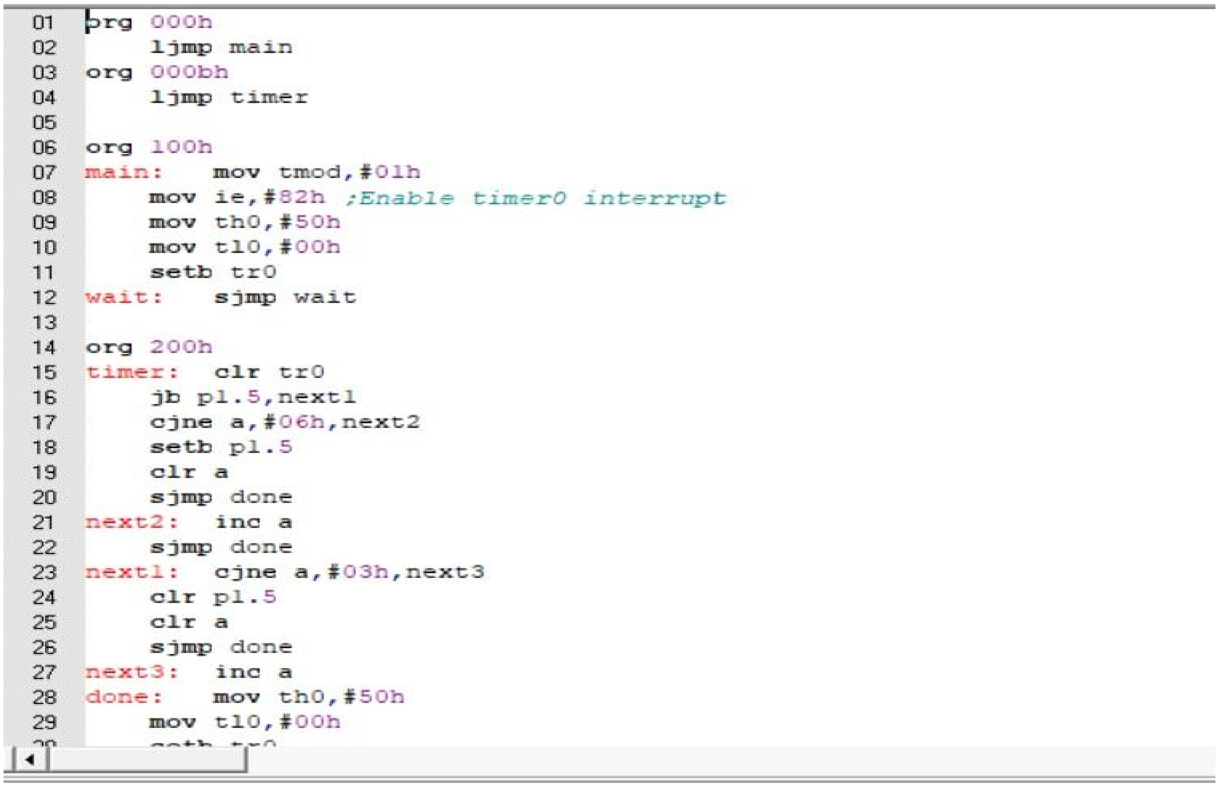
```
org 000h
     ljmp main
org 000bh
     ljmp timer

org 100h
main:    mov tmod,#00h
     mov ie,#82h ;Enable timer0 interrupt
     mov th0,#50h
     mov tl0,#00h
     setb tr0
wait:    sjmp wait

org 200h
timer:   clr tr0
     jb p1.5,next1
     cjne a,#06h,next2
     setb p1.5
     clr a
     sjmp done
next2:   inc a
     sjmp done
next1:   cjne a,#03h,next3
     clr p1.5
     clr a
     sjmp done
next3:   inc a
done:    mov th0,#50h
     mov tl0,#00h
     setb tr0
```

After execution



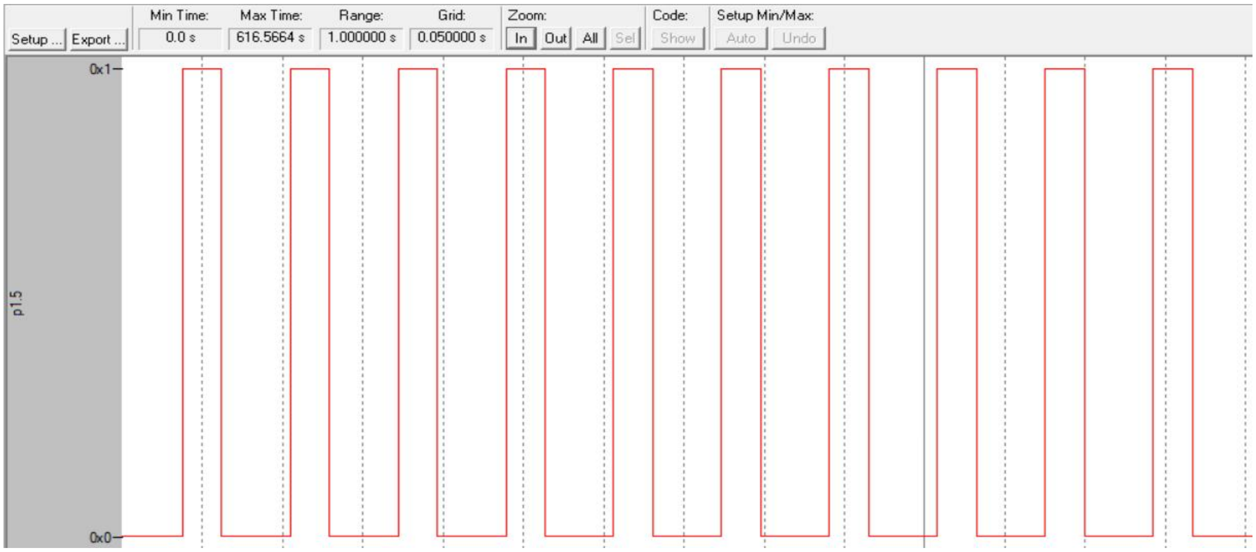
Min Time: 0.0 s
Max Time: 616.5664 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 1



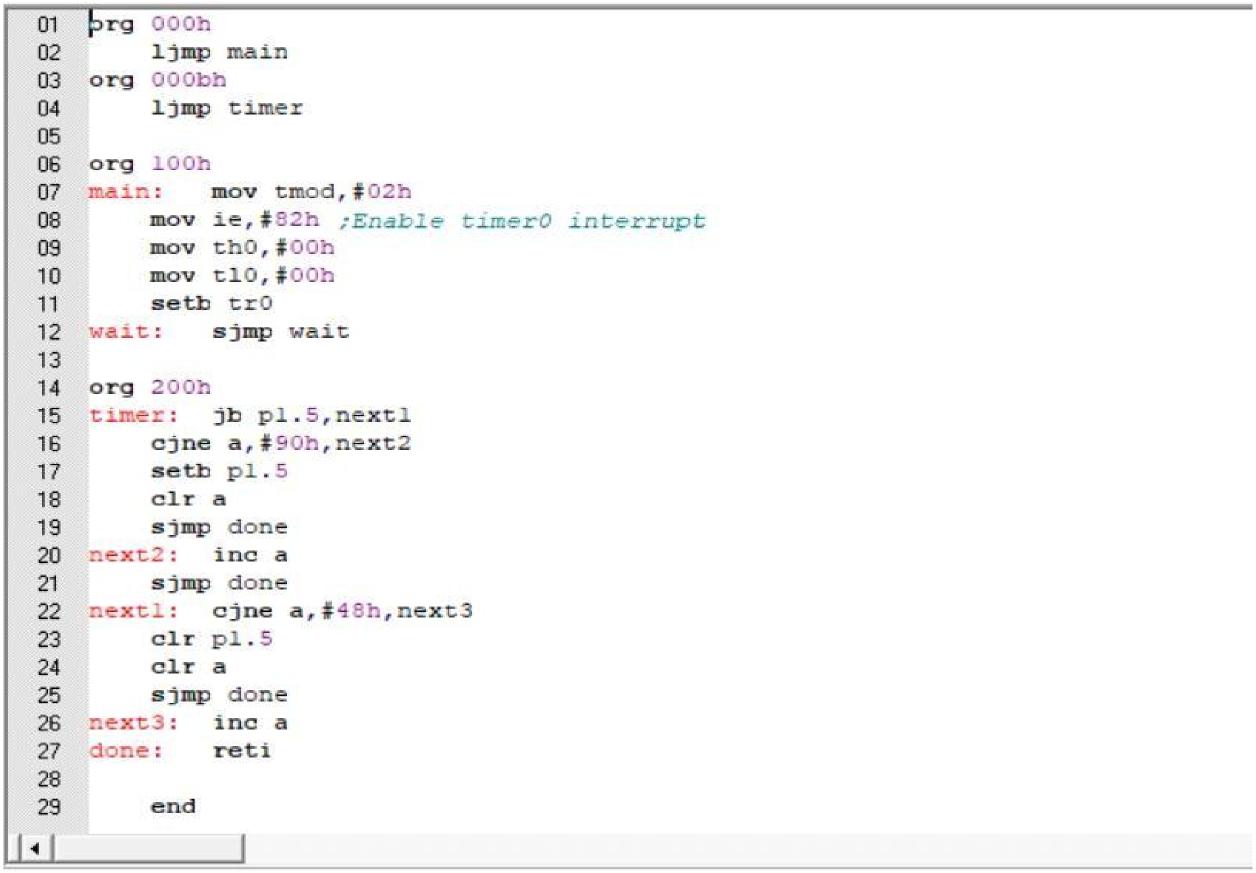
```
org 000h
     ljmp main
org 000bh
     ljmp timer

org 100h
main:    mov tmod,#01h
     mov ie,#82h ;Enable timer0 interrupt
     mov th0,#50h
     mov tl0,#00h
     setb tr0
wait:    sjmp wait

org 200h
timer:   clr tr0
     jb p1.5,next1
     cjne a,#06h,next2
     setb p1.5
     clr a
     sjmp done
next2:   inc a
     sjmp done
next1:   cjne a,#03h,next3
     clr p1.5
     clr a
     sjmp done
next3:   inc a
done:    mov th0,#50h
     mov tl0,#00h
     setb tr0
```

After execution



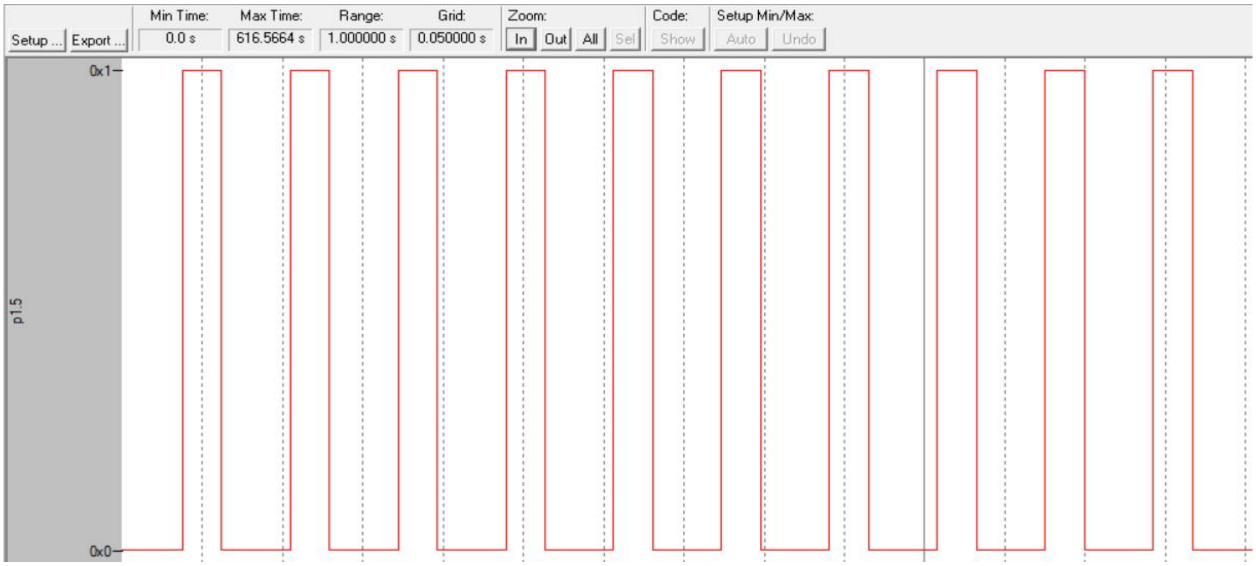
Min Time: 0.0 s
Max Time: 616.5664 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 2



```
org 000h
    ljmp main
org 000bh
    ljmp timer

org 100h
main:   mov tmod,#02h
    mov ie,#82h ;Enable timer0 interrupt
    mov th0,#00h
    mov tl0,#00h
    setb tr0
wait:   sjmp wait

org 200h
timer:  jb p1.5,next1
    cjne a,#90h,next2
    setb p1.5
    clr a
    sjmp done
next2:  inc a
    sjmp done
next1:  cjne a,#48h,next3
    clr p1.5
    clr a
    sjmp done
next3:  inc a
done:   reti

    end
```

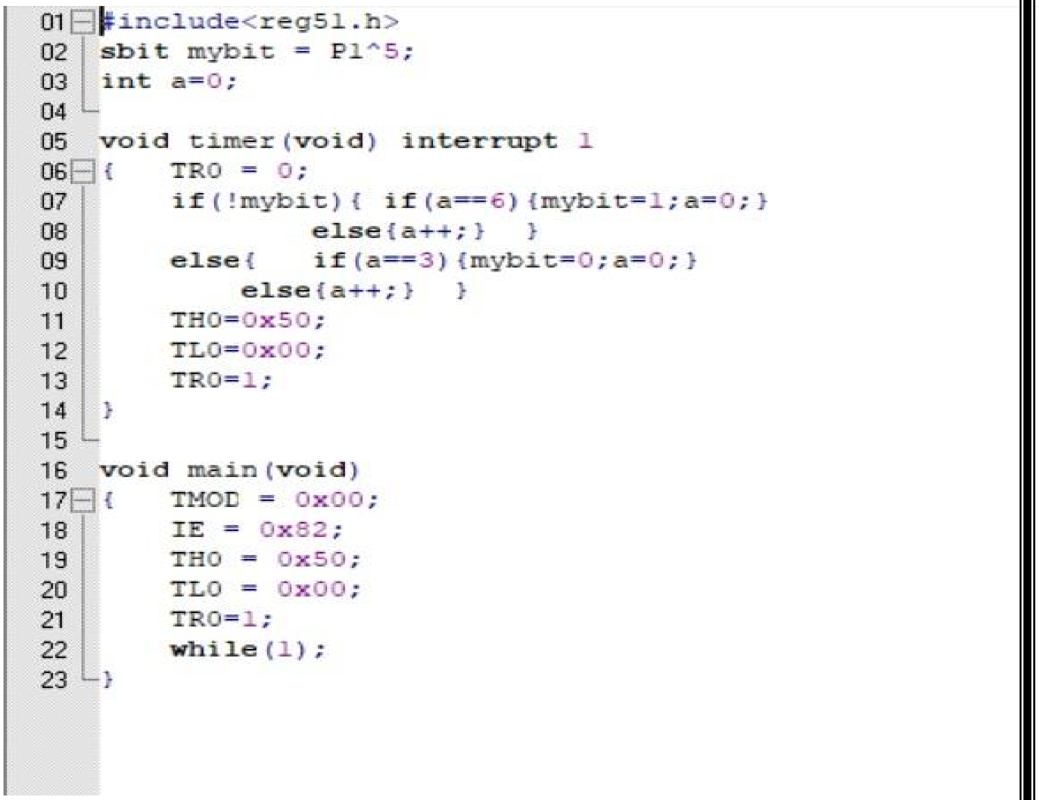
After execution



Min Time: 0.0 s
Max Time: 616.5664 s
Range: 1.000000 s
Grid: 0.050000 s

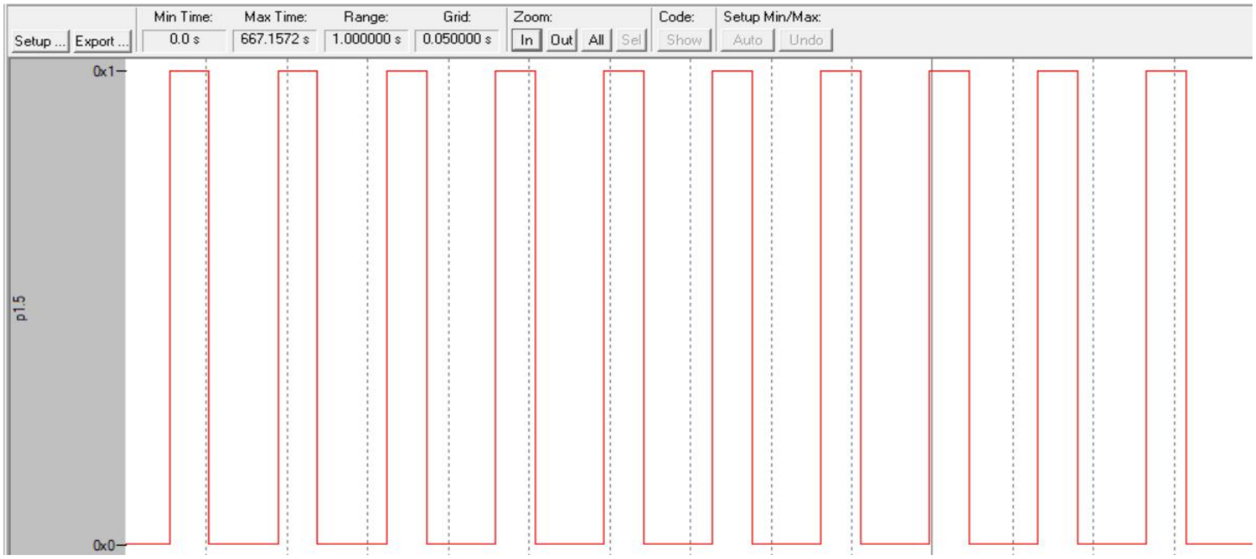
Repeat above problems of status check and interrupt methods to generate square wave using embedded c programs.

Mode 0 using interrupt

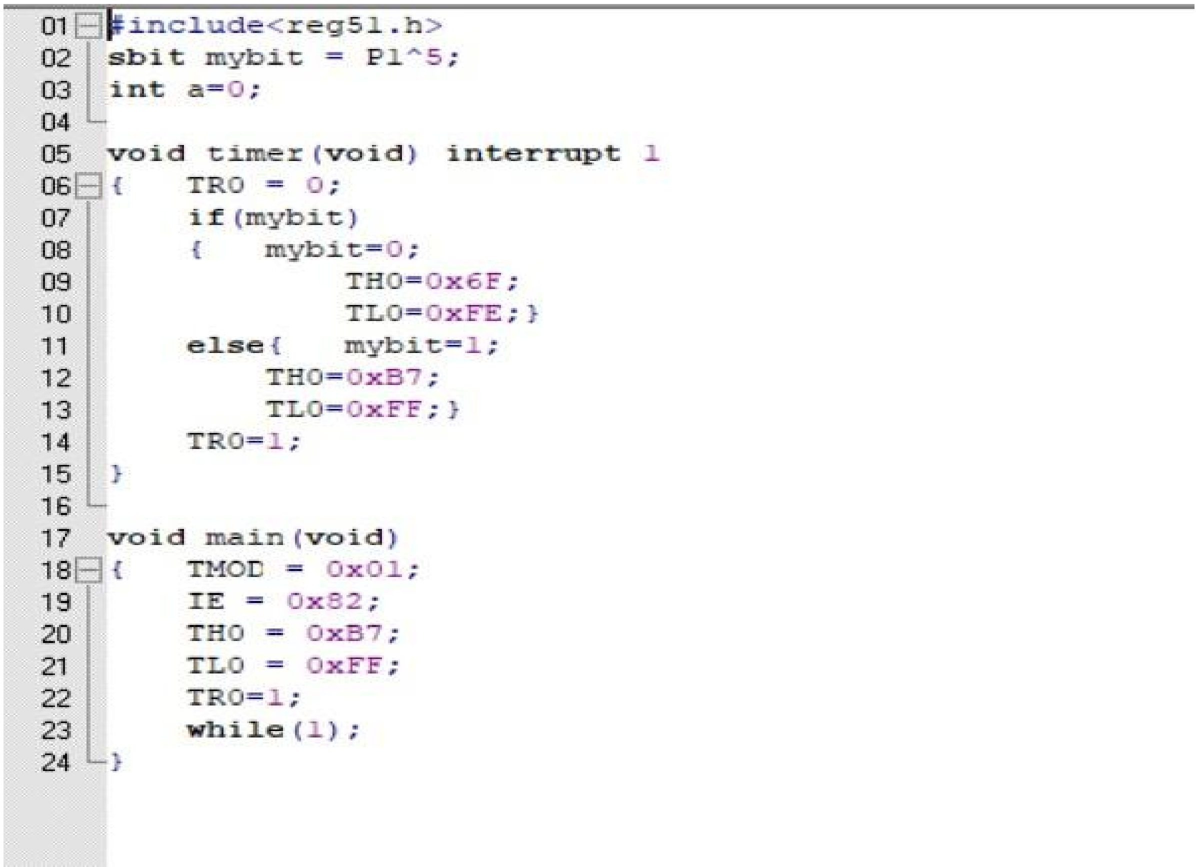


```
#include<reg51.h>
sbit mybit = P1^5;
int a=0;

void timer(void) interrupt 1
{    TR0 = 0;
     if(!mybit){ if(a==6){mybit=1;a=0;}
                 else{a++;}   }
     else{    if(a==3){mybit=0;a=0;}
             else{a++;}   }
     TH0=0x50;
     TL0=0x00;
     TR0=1;
}

void main(void)
{    TMOD = 0x00;
     IE = 0x82;
     TH0 = 0x50;
     TL0 = 0x00;
     TR0=1;
     while(1);
}
```

After execution

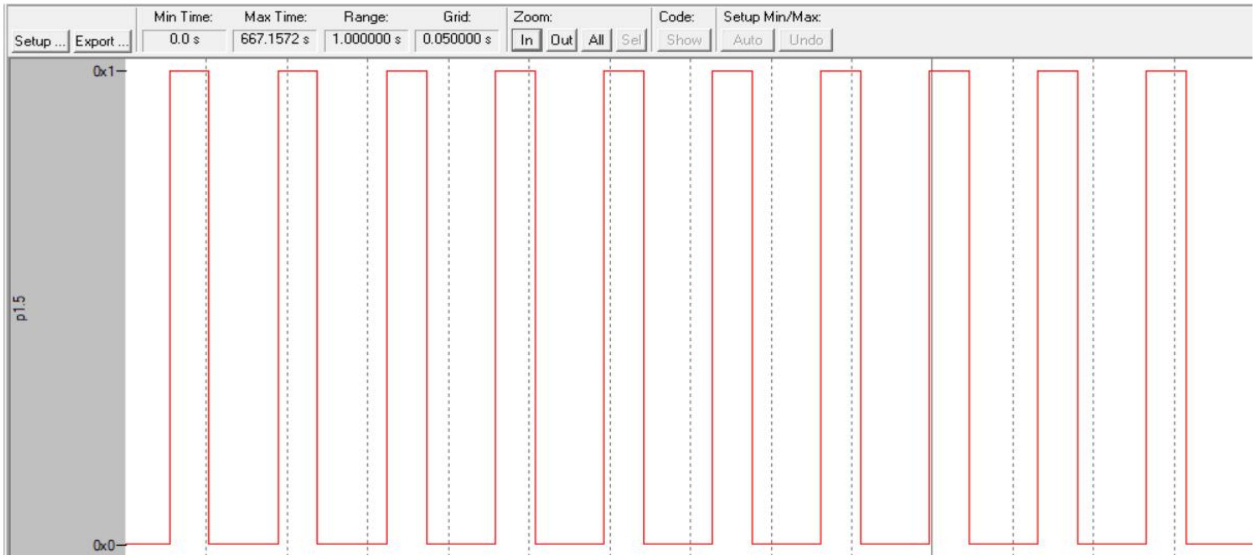


Min Time: 0.0 s
Max Time: 667.1572 s
Range: 1.000000 s
Grid: 0.050000 s



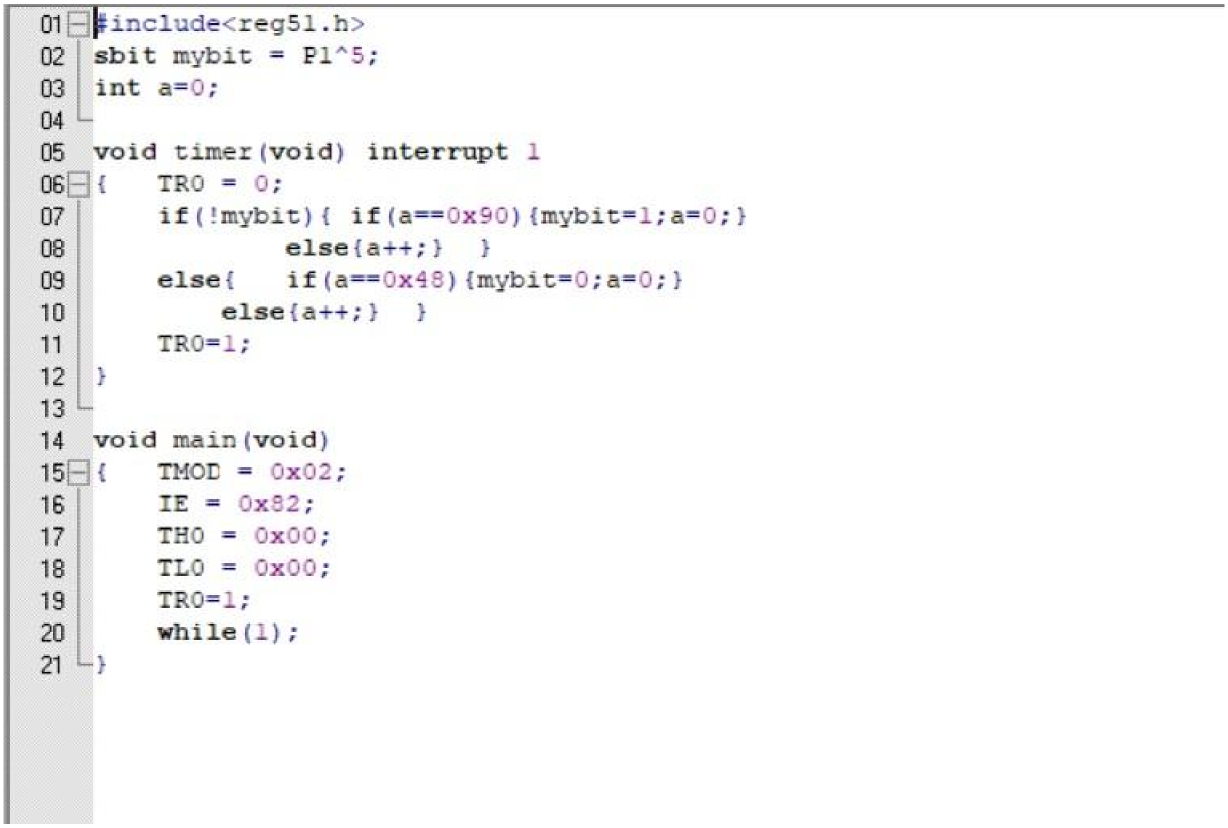
```
#include<reg51.h>
sbit mybit = P1^5;
int a=0;

void timer(void) interrupt 1
{    TR0 = 0;
     if(mybit)
     {    mybit=0;
               TH0=0x6F;
               TL0=0xFE;}
     else{    mybit=1;
          TH0=0xB7;
          TL0=0xFF;}
     TR0=1;
}

void main(void)
{    TMOD = 0x01;
     IE = 0x82;
     TH0 = 0xB7;
     TL0 = 0xFF;
     TR0=1;
     while(1);
}
```

After execution

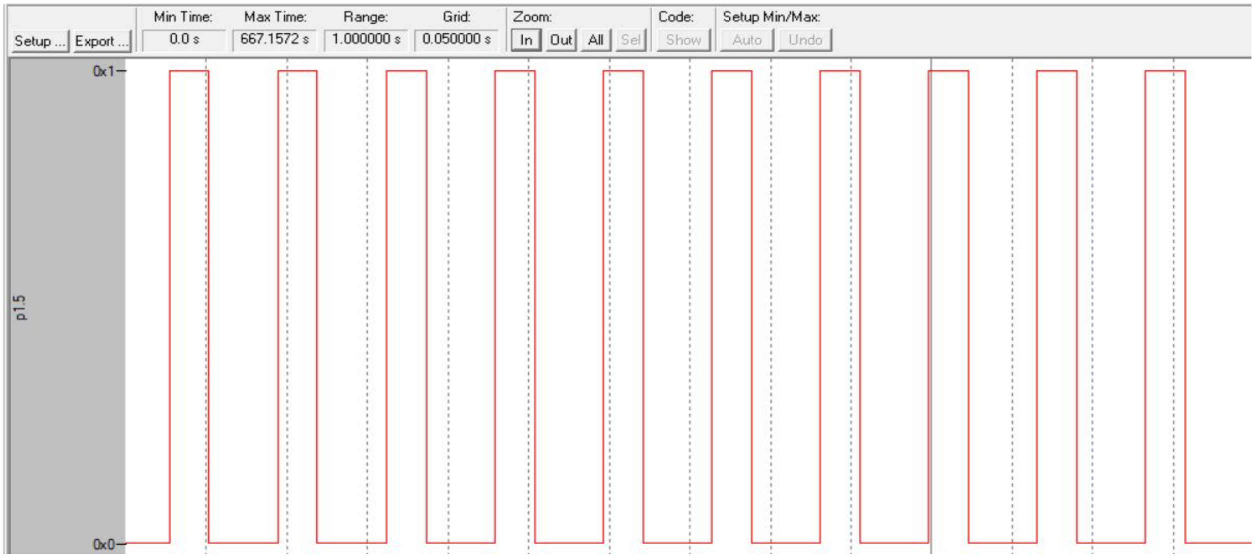


Min Time: 0.0 s
Max Time: 667.1572 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 2 using interrupt

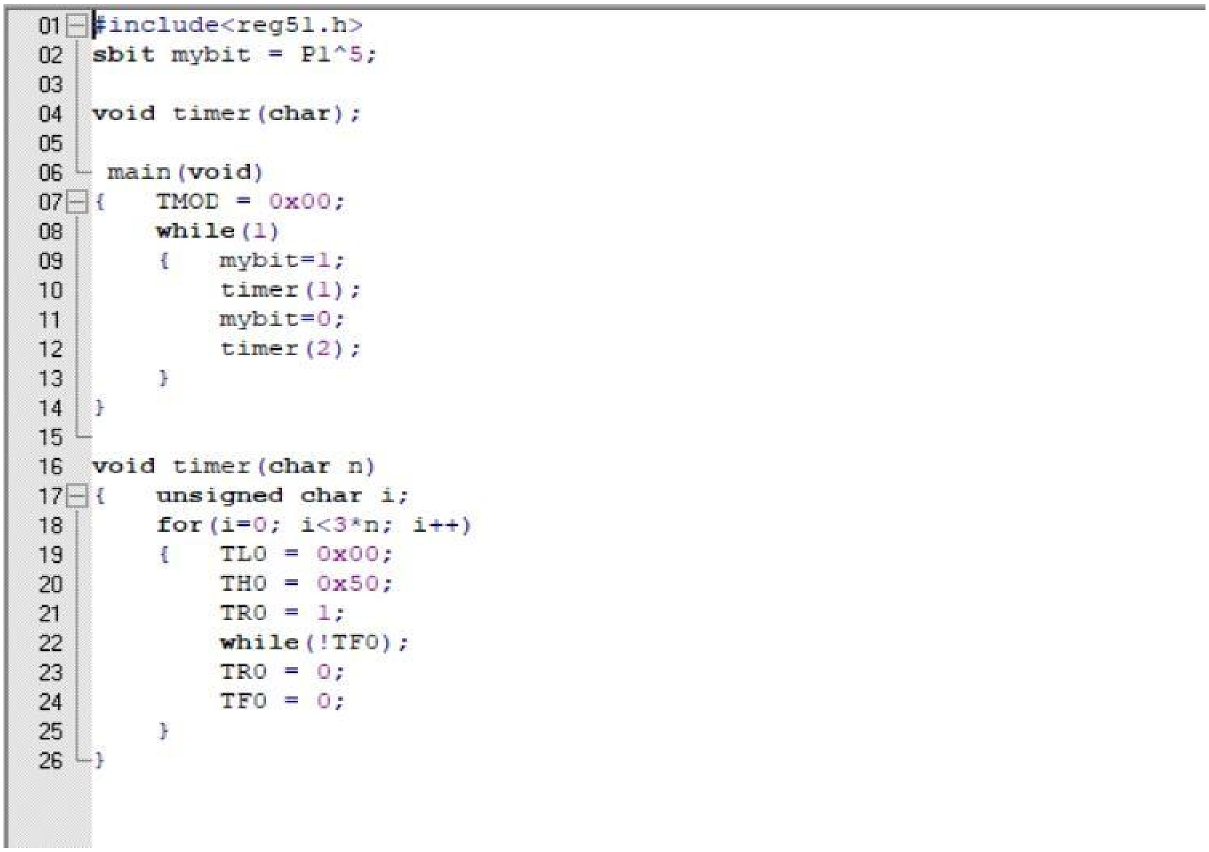


```
#include<reg51.h>
sbit mybit = P1^5;
int a=0;

void timer(void) interrupt 1
{    TR0 = 0;
     if(!mybit){ if(a==0x90){mybit=1;a=0;}
             else{a++;}  }
     else{    if(a==0x48){mybit=0;a=0;}
         else{a++;}  }
     TR0=1;
}

void main(void)
{    TMOD = 0x02;
     IE = 0x82;
     TH0 = 0x00;
     TL0 = 0x00;
     TR0=1;
     while(1);
}
```

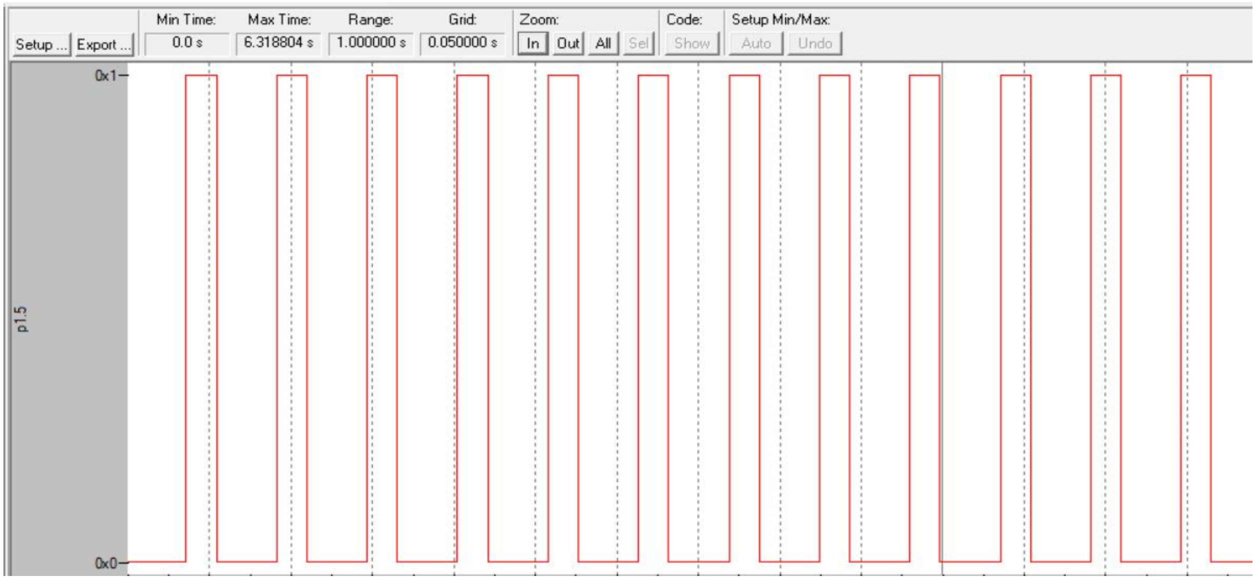


Min Time: 0.0 s
Max Time: 667.1572 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 0 using status check

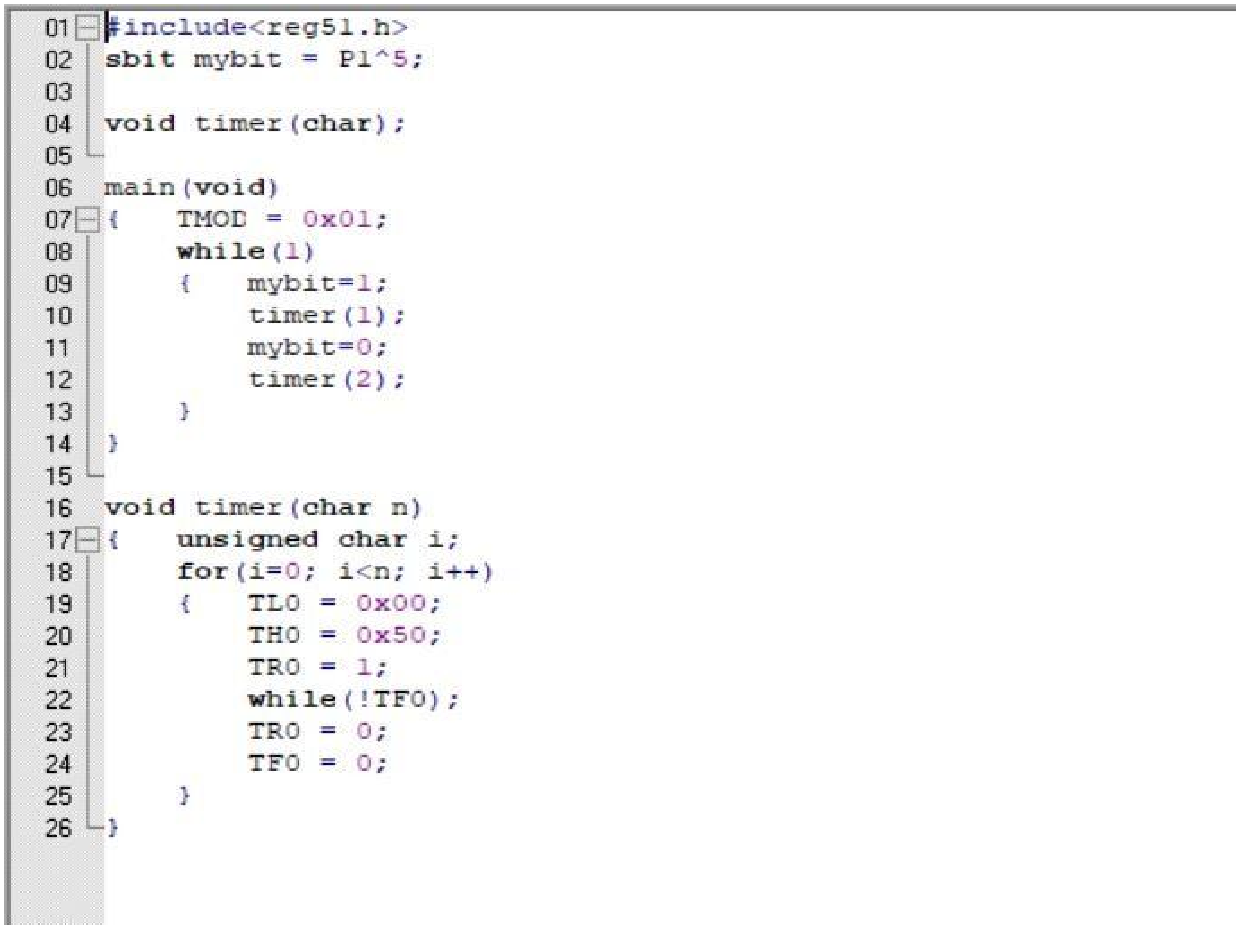


```
#include<reg51.h>
sbit mybit = P1^5;

void timer(char);

 main(void)
{    TMOD = 0x00;
     while(1)
     {    mybit=1;
          timer(1);
          mybit=0;
          timer(2);
     }
}

void timer(char n)
{    unsigned char i;
     for(i=0; i<3*n; i++)
     {    TL0 = 0x00;
          TH0 = 0x50;
          TR0 = 1;
          while(!TF0);
          TR0 = 0;
          TF0 = 0;
     }
}
```



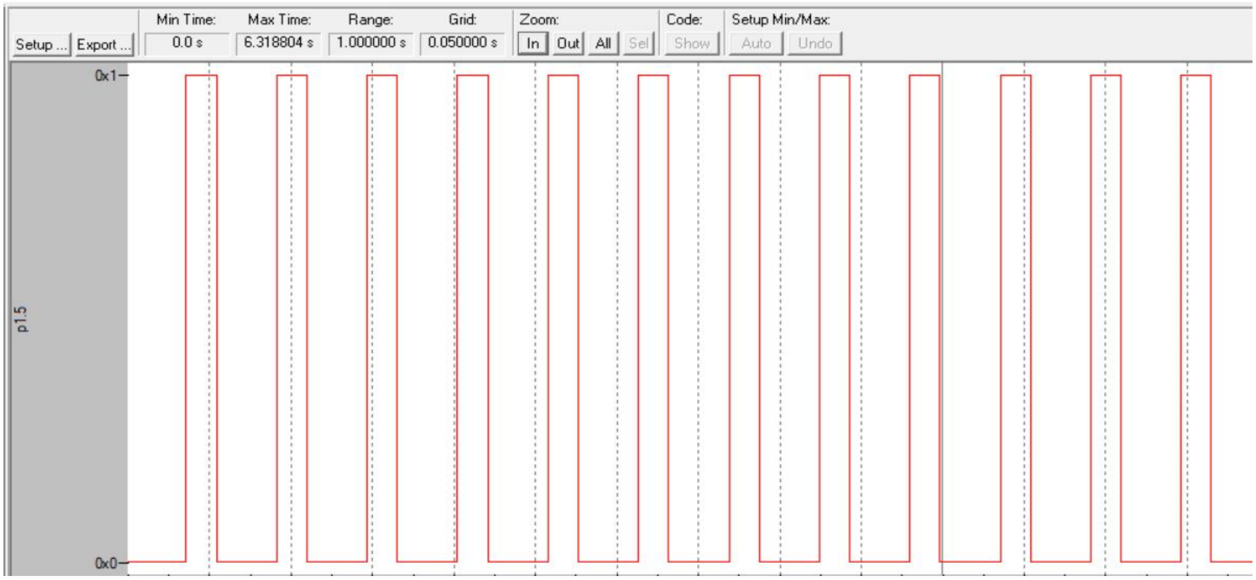
Min Time: 0.0 s
Max Time: 6.318804 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 1 using status check



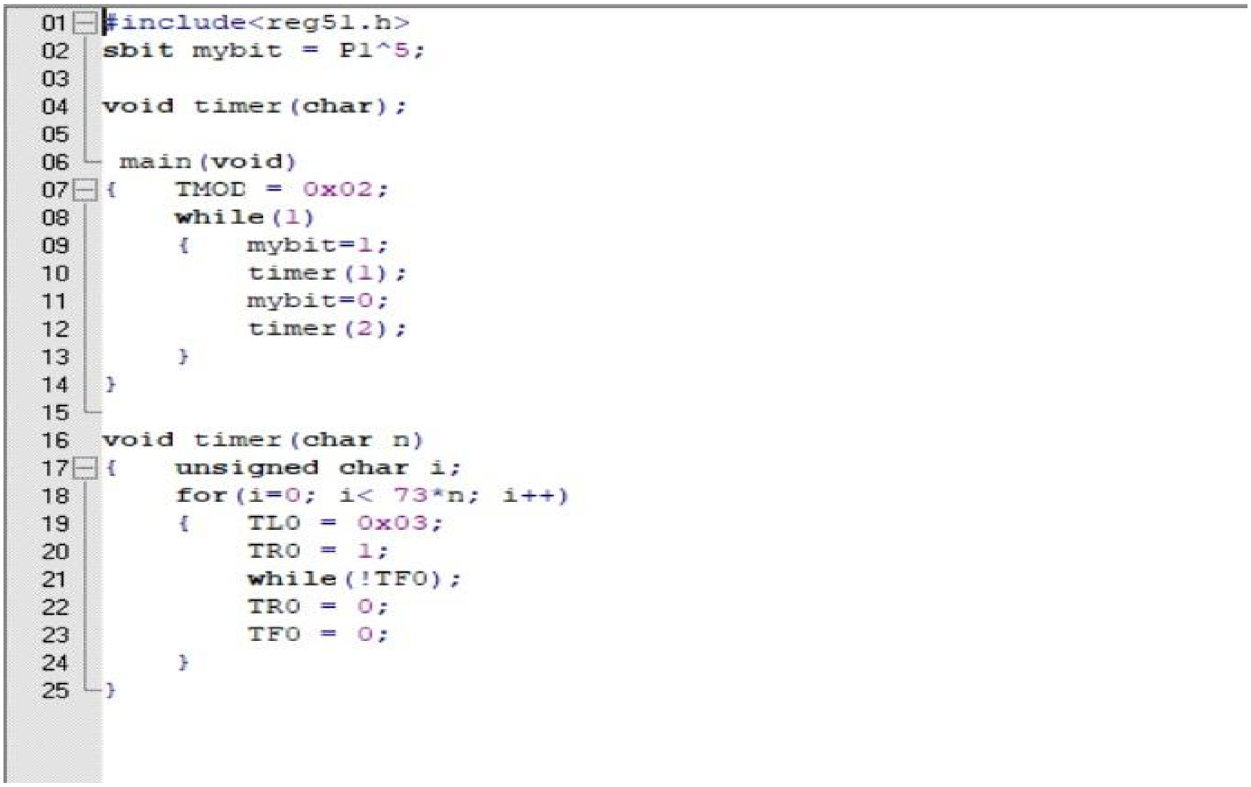
```c
#include<reg51.h>
sbit mybit = P1^5;

void timer(char);

main(void)
{    TMOD = 0x01;
     while(1)
     {    mybit=1;
          timer(1);
          mybit=0;
          timer(2);
     }
}

void timer(char n)
{    unsigned char i;
     for(i=0; i<n; i++)
     {    TL0 = 0x00;
          TH0 = 0x50;
          TR0 = 1;
          while(!TF0);
          TR0 = 0;
          TF0 = 0;
     }
}
```

After execution

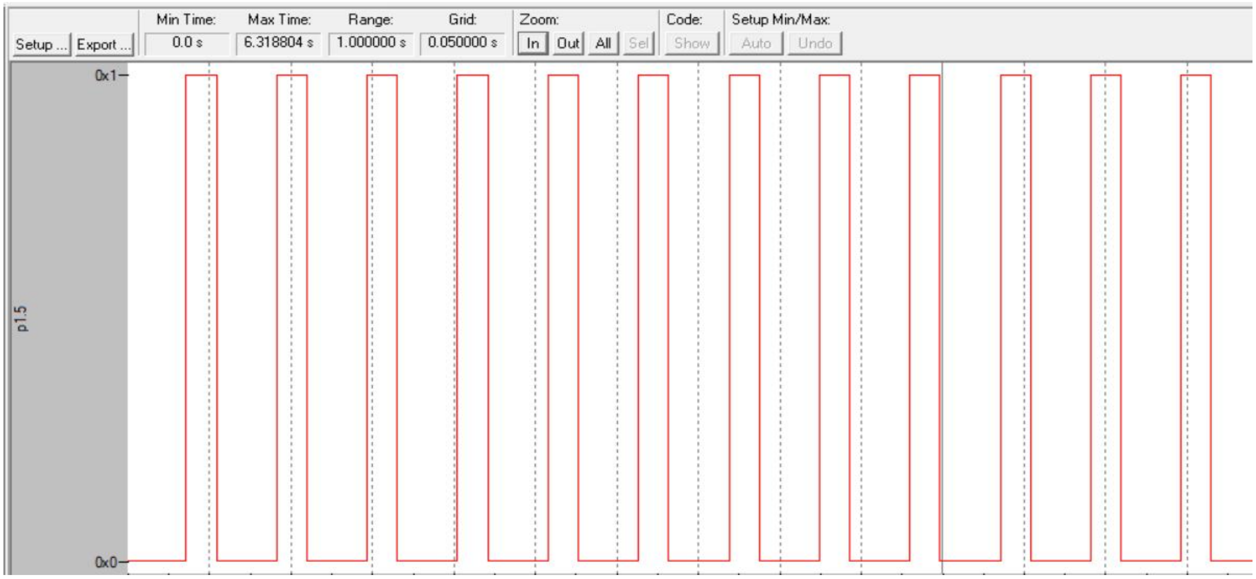


Min Time: 0.0 s
Max Time: 6.318804 s
Range: 1.000000 s
Grid: 0.050000 s

Mode 2 using status check



```
#include<reg51.h>
sbit mybit = P1^5;

void timer(char);

 main(void)
{    TMOD = 0x02;
     while(1)
     {    mybit=1;
          timer(1);
          mybit=0;
          timer(2);
     }
}

void timer(char n)
{    unsigned char i;
     for(i=0; i< 73*n; i++)
     {    TL0 = 0x03;
          TR0 = 1;
          while(!TF0);
          TR0 = 0;
          TF0 = 0;
     }
}
```



Min Time: 0.0 s

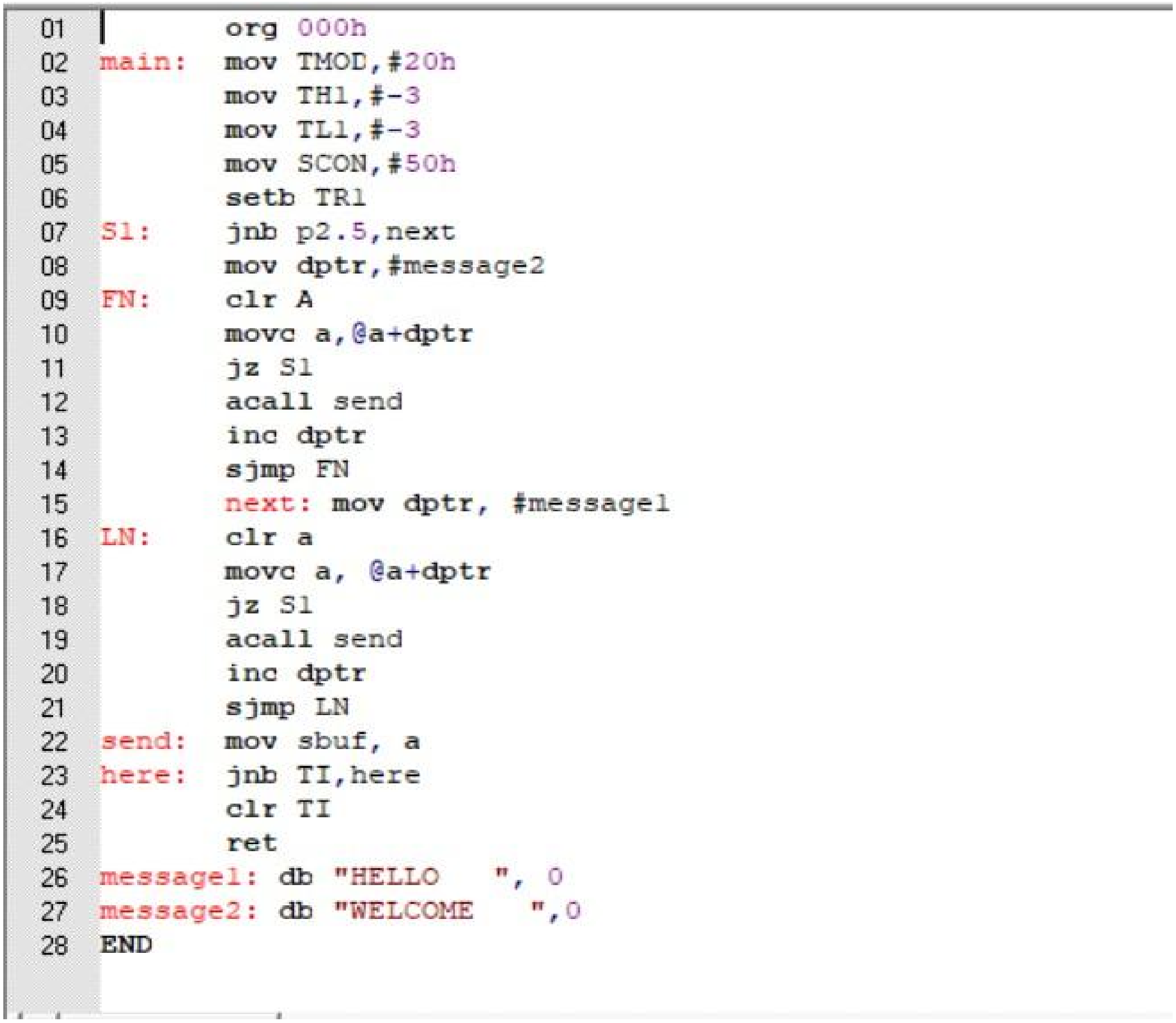
Max Time: 6.318804 s

Range: 1.000000 s

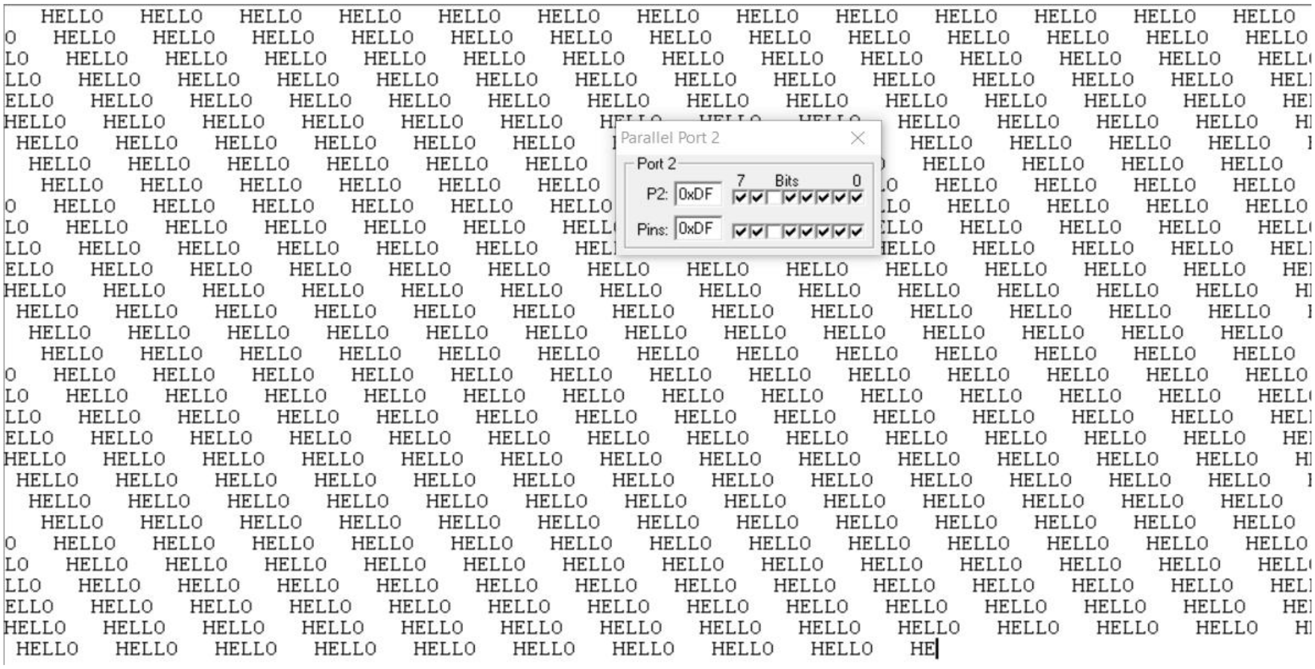
Grid: 0.050000 s

SET 8 PROGRAMS

8.1 A switch is connected to P2.5. write an 8051 assembly level program to read the status of switch and if switch is closed send serially "HELL0', else send WELCOME' at baud rate 9600.



```
          org 000h
main:     mov TMOD,#20h
          mov TH1,#-3
          mov TL1,#-3
          mov SCON,#50h
          setb TR1
S1:       jnb p2.5,next
          mov dptr,#message2
FN:       clr A
          movc a,@a+dptr
          jz S1
          acall send
          inc dptr
          sjmp FN
          next: mov dptr, #message1
LN:       clr a
          movc a, @a+dptr
          jz S1
          acall send
          inc dptr
          sjmp LN
send:     mov sbuf, a
here:     jnb TI,here
          clr TI
          ret
message1: db "HELLO    ", 0
message2: db "WELCOME    ",0
END
```



```
   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
O   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
LO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELL
LLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HEL
ELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HE
HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
 HELLO   HELLO   HELLO   HELLO   HELLO   HELLO                            HELLO   HELLO   HELLO   HELLO
  HELLO   HELLO   HELLO   HELLO   HELLO   HELLO                            HELLO   HELLO   HELLO   HELLO
   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO                          O   HELLO   HELLO   HELLO   HELLO
O   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO                         LO   HELLO   HELLO   HELLO   HELLO
LO   HELLO   HELLO   HELLO   HELLO   HELLO   HELL                         LLO   HELLO   HELLO   HELLO   HELL
LLO   HELLO   HELLO   HELLO   HELLO   HELLO   HEL                         ELLO   HELLO   HELLO   HELLO   HEL
ELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HE
HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
 HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
  HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
O   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
LO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELL
LLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HEL
ELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HE
HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
 HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
  HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
O   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO
LO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELL
LLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HEL
ELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HE
HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   H
 HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HELLO   HE
```

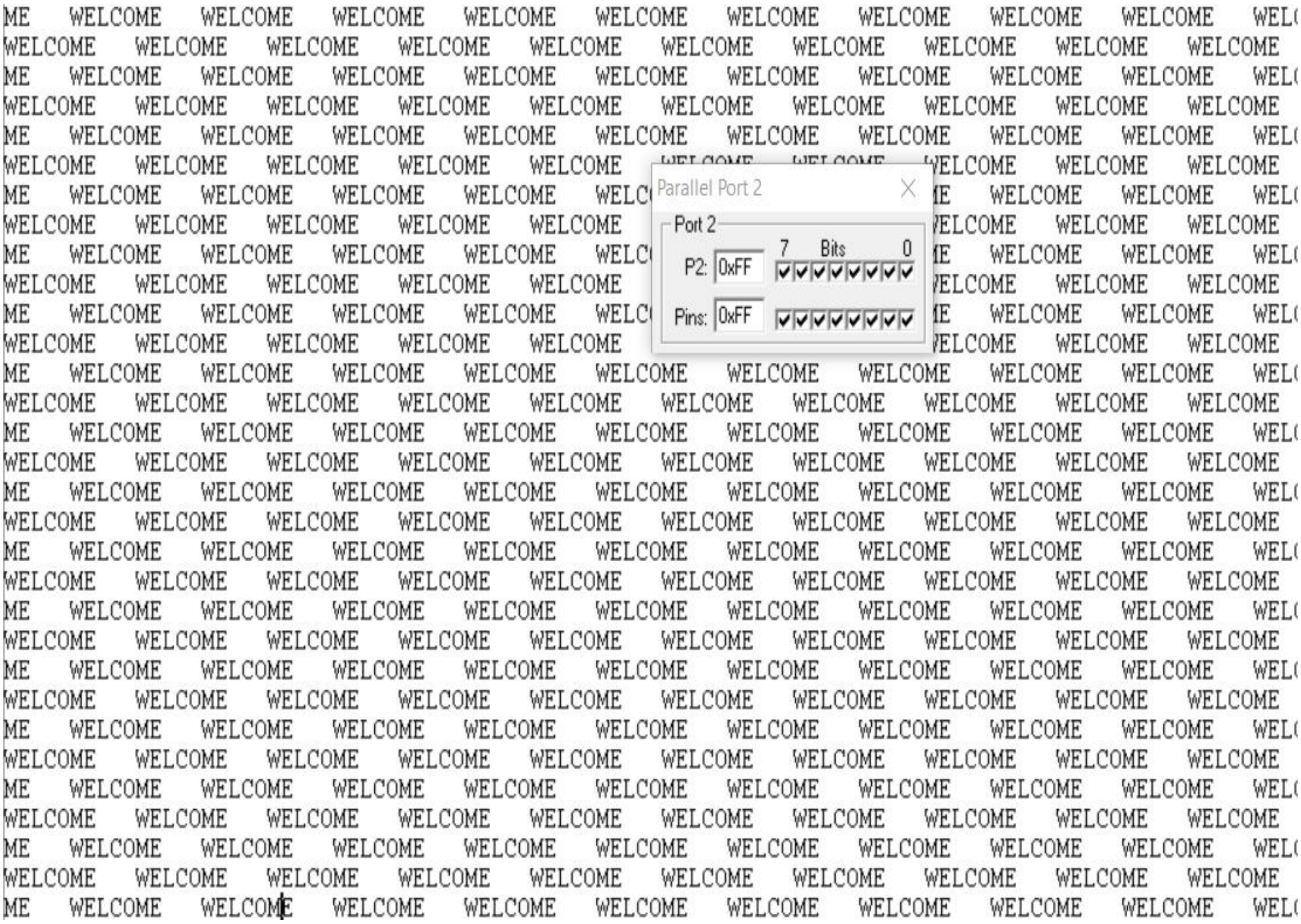
Parallel Port 2

Port 2

7 Bits 0

P2: 0xDF ☑☑☐☑☑☑☑☑

Pins: 0xDF ☑☑☐☑☑☑☑☑



```
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELC                        ME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME                         ELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELC                        ME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME                         ELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELC                        ME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME                         ELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME
ME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELCOME   WELC
```

Parallel Port 2

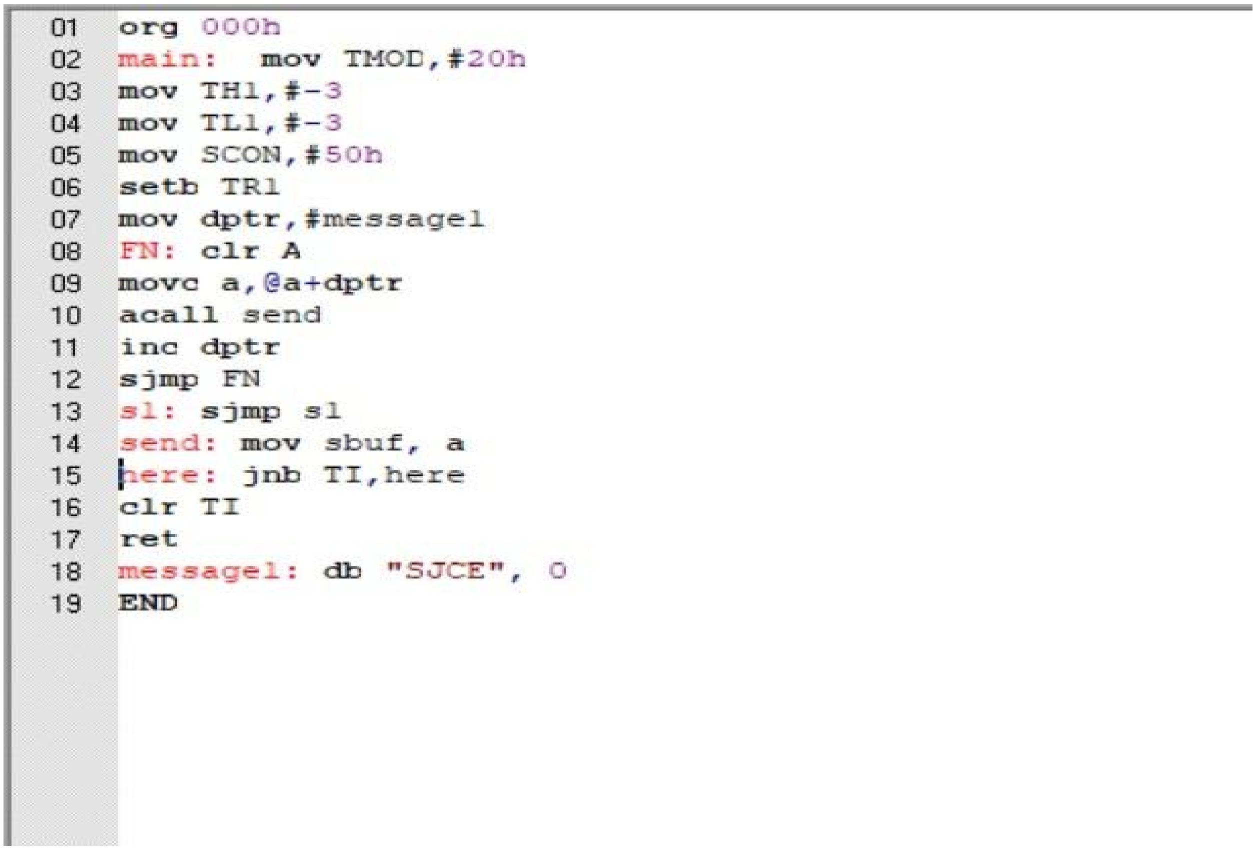
Port 2

7 Bits 0

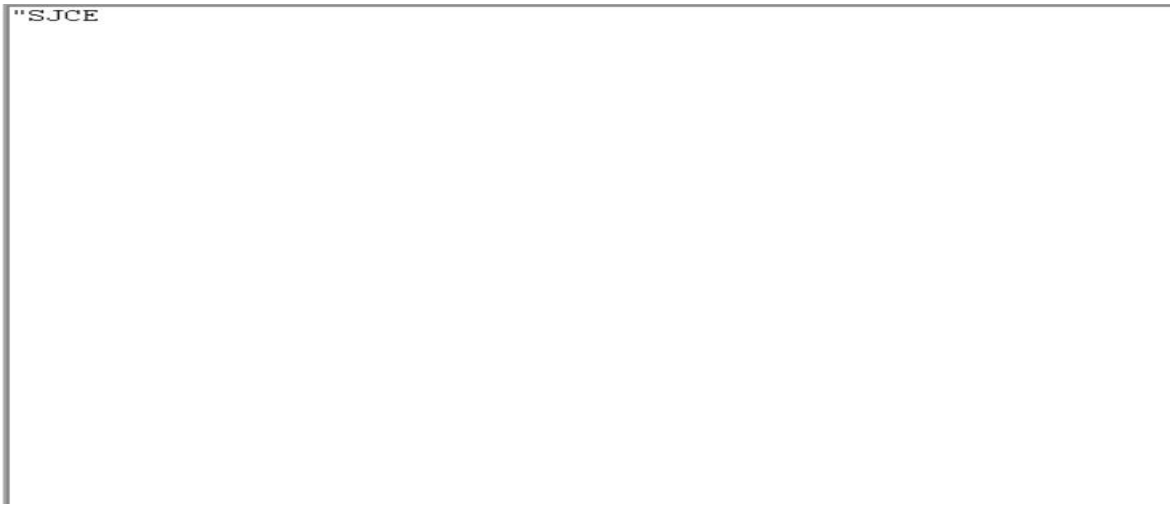
P2: 0xFF ☑☑☑☑☑☑☑☑

Pins: 0xFF ☑☑☑☑☑☑☑☑

2.Write an 8051 assembly level program to transfer a message SJCE serially by programming serial1 communication in interrupt mode with baud rate 9600.

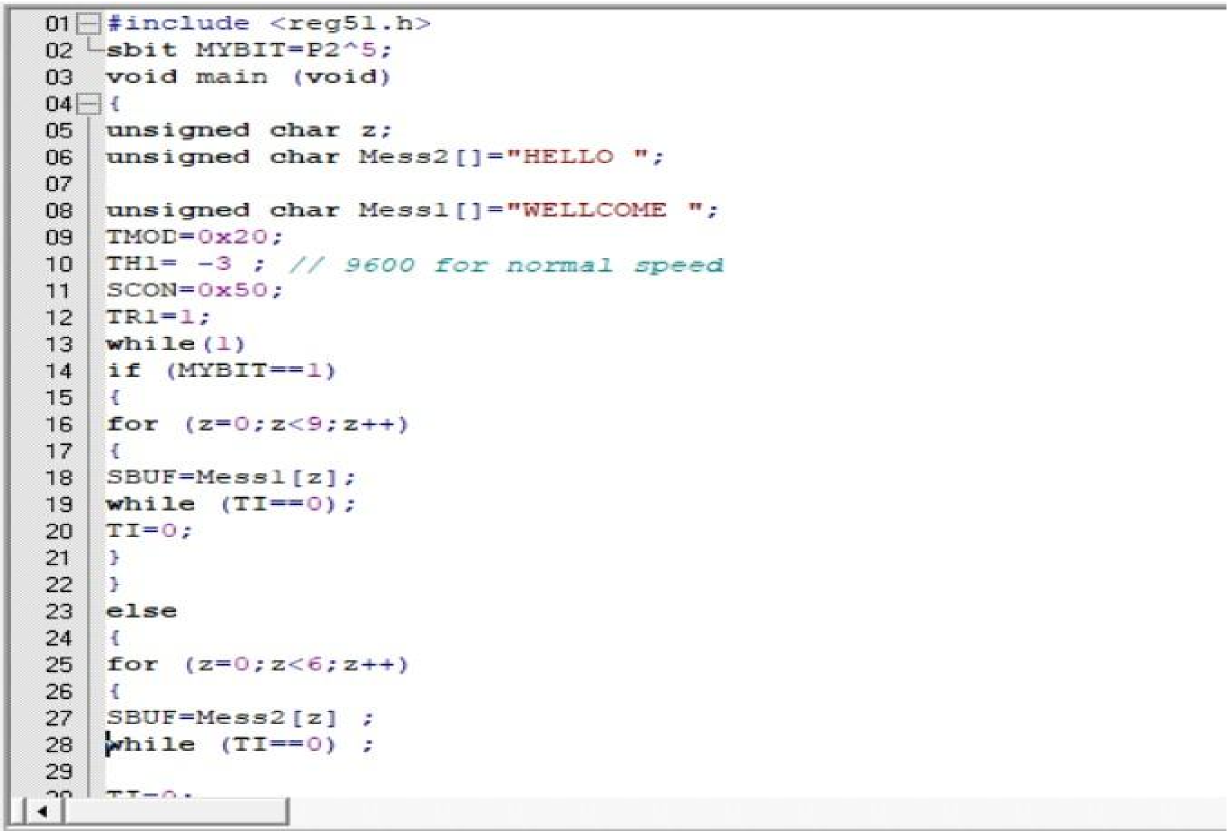


```
org 000h
main:  mov TMOD,#20h
mov TH1,#-3
mov TL1,#-3
mov SCON,#50h
setb TR1
mov dptr,#message1
FN: clr A
movc a,@a+dptr
acall send
inc dptr
sjmp FN
s1: sjmp s1
send: mov sbuf, a
here: jnb TI,here
clr TI
ret
message1: db "SJCE", 0
END
```

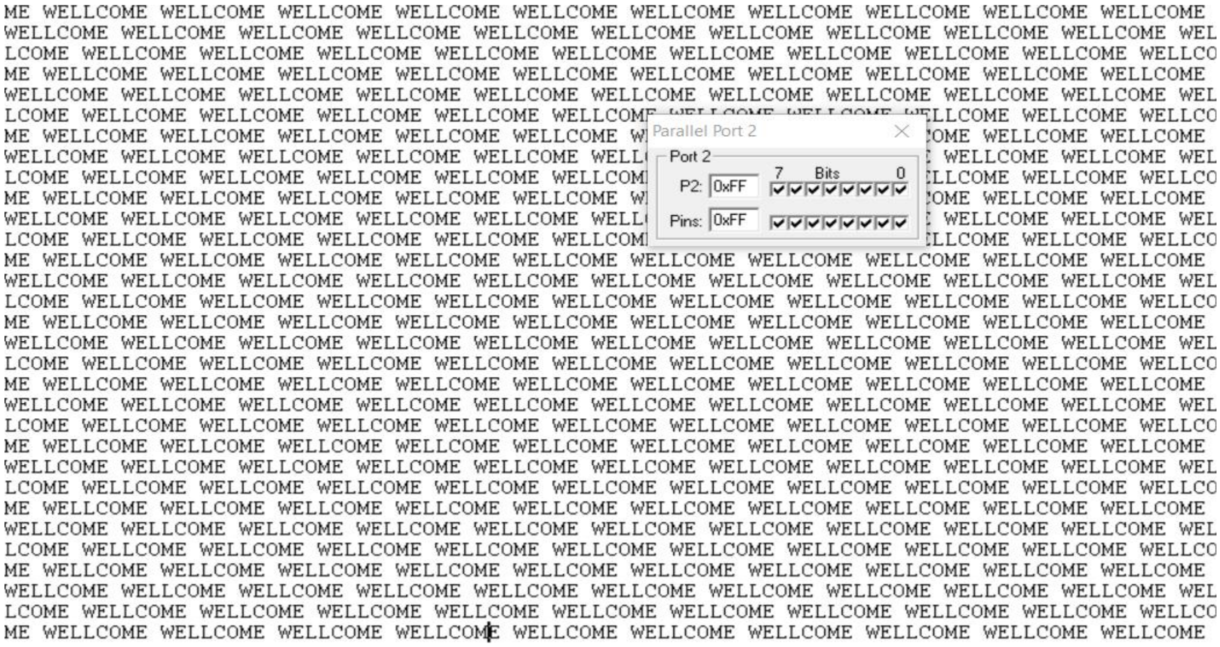


```
"SJCE
```

Repeat the problem 1 & 2 using embedded C program.
Problem 1



```
#include <reg51.h>
sbit MYBIT=P2^5;
void main (void)
{
unsigned char z;
unsigned char Mess2[]="HELLO ";

unsigned char Mess1[]="WELLCOME ";
TMOD=0x20;
TH1= -3 ; // 9600 for normal speed
SCON=0x50;
TR1=1;
while(1)
if (MYBIT==1)
{
for (z=0;z<9;z++)
{
SBUF=Mess1[z];
while (TI==0);
TI=0;
}
}
else
{
for (z=0;z<6;z++)
{
SBUF=Mess2[z] ;
while (TI==0) ;

TI=0;
```



```
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME W                        OME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELL                        WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOM                        LLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME W                        OME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELL                        WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOM                        LLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WEL
LCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCO
ME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME WELLCOME
```

Parallel Port 2

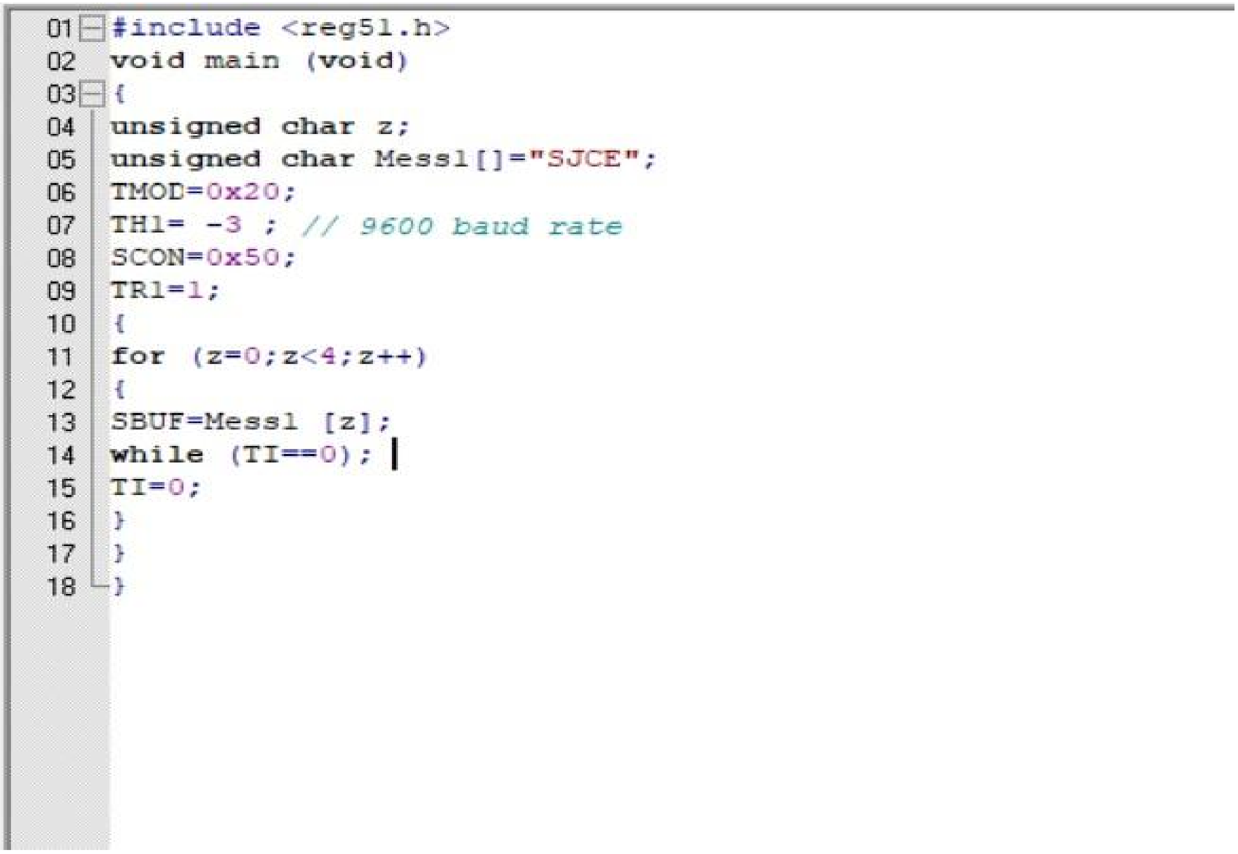
Port 2

7 Bits 0

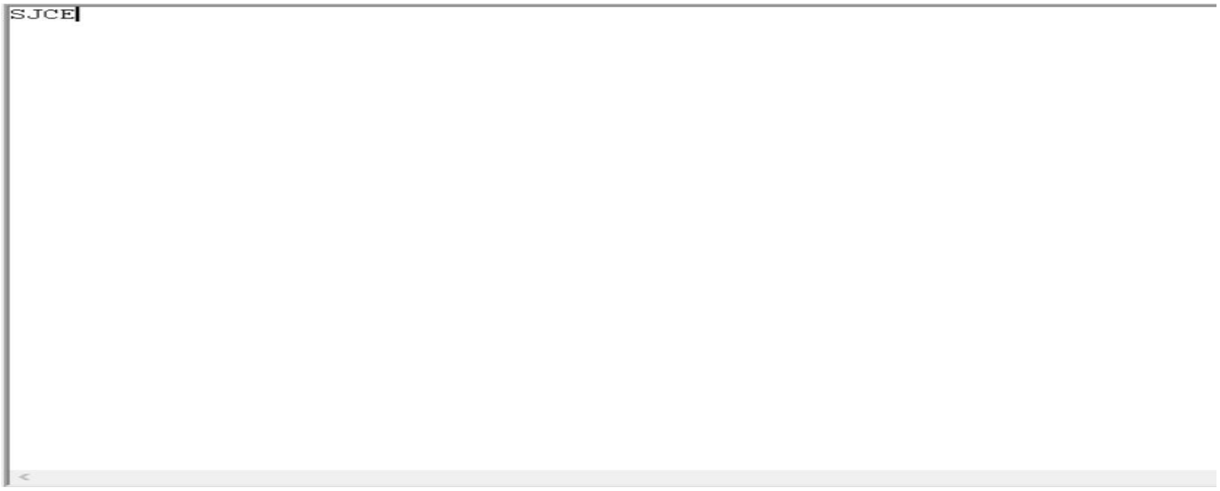
P2: 0xFF ☑☑☑☑☑☑☑☑

Pins: 0xFF ☑☑☑☑☑☑☑☑

Problem 2



```
#include <reg51.h>
void main (void)
{
unsigned char z;
unsigned char Mess1[]="SJCE";
TMOD=0x20;
TH1= -3 ; // 9600 baud rate
SCON=0x50;
TR1=1;
{
for (z=0;z<4;z++)
{
SBUF=Mess1 [z];
while (TI==0);
TI=0;
}
}
}
```



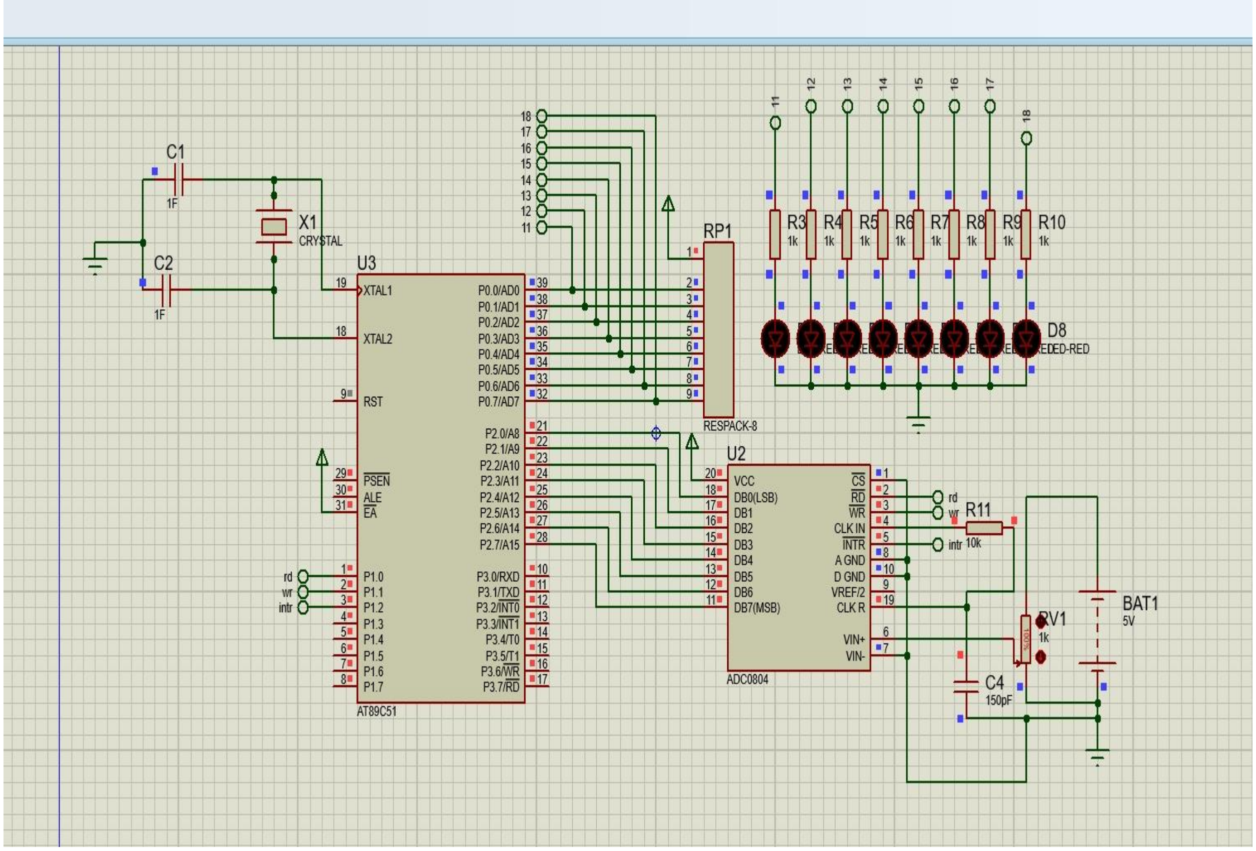
```
SJCE
```

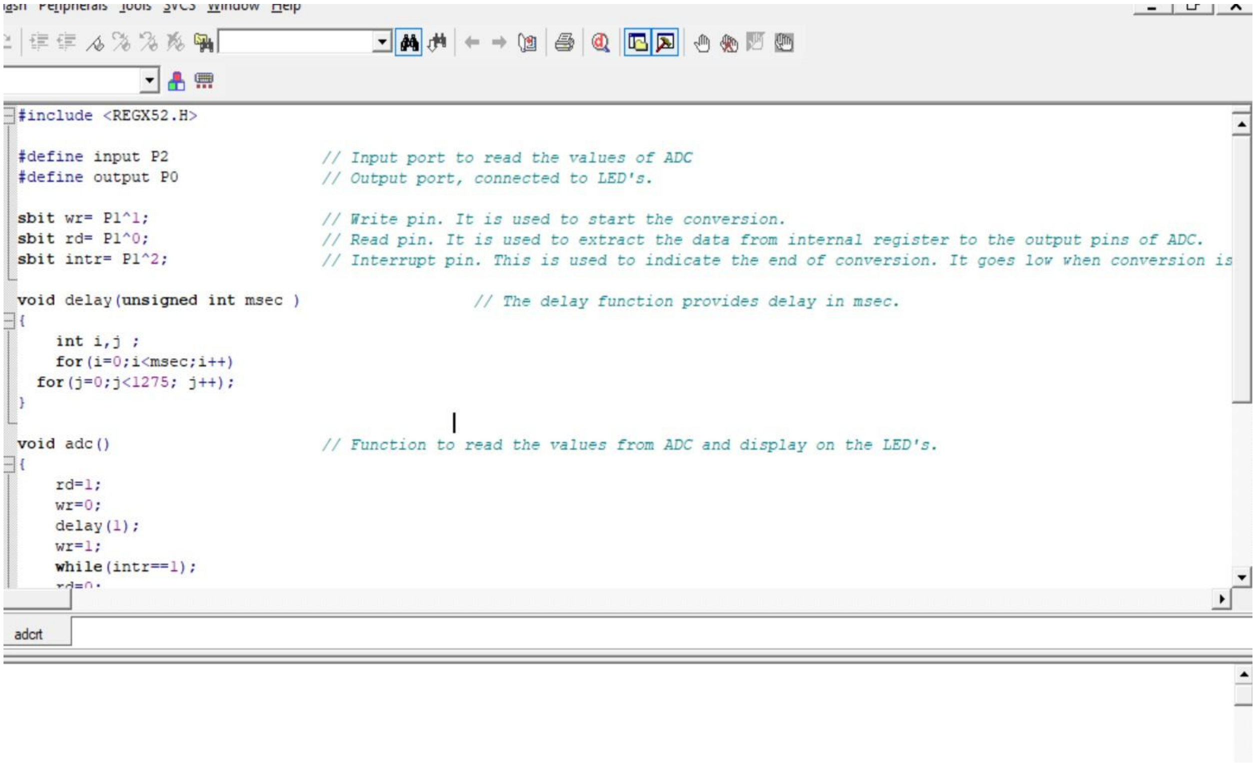
HARDWARE EXPERIMENT

Question Number - 1

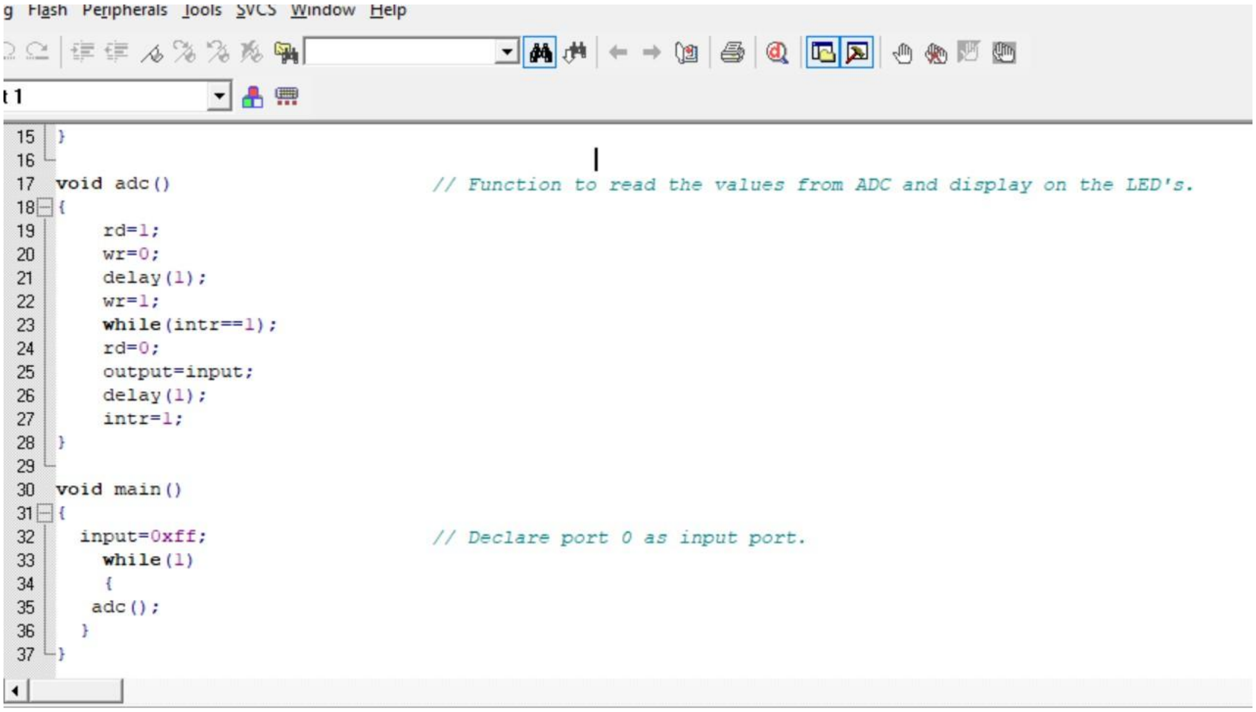
Interface a Multichannel ADC to 8051 and develop a program to read the analog data, Convert into digital value and display the digitized value on port 0 connected to leds

Circuit Diagram :

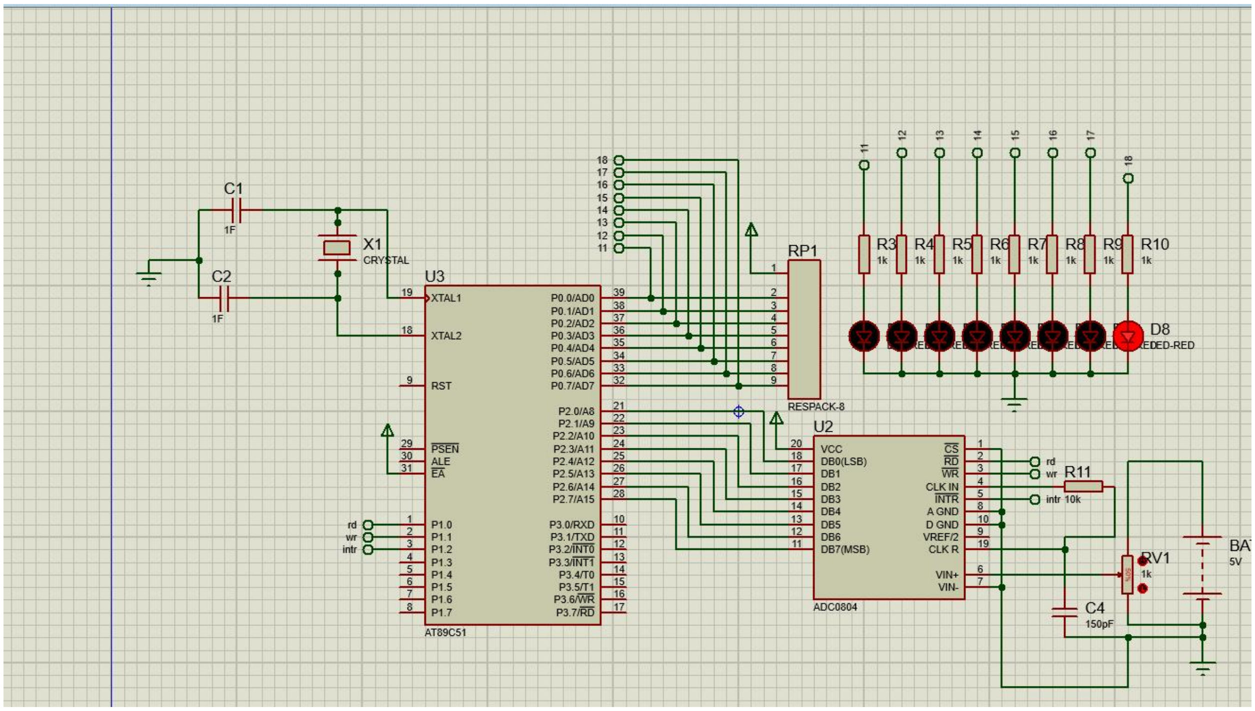




```
#include <REGX52.H>

#define input P2                // Input port to read the values of ADC
#define output P0               // Output port, connected to LED's.

sbit wr= P1^1;                  // Write pin. It is used to start the conversion.
sbit rd= P1^0;                  // Read pin. It is used to extract the data from internal register to the output pins of ADC.
sbit intr= P1^2;                // Interrupt pin. This is used to indicate the end of conversion. It goes low when conversion is

void delay(unsigned int msec )              // The delay function provides delay in msec.
{
    int i,j ;
    for(i=0;i<msec;i++)
  for(j=0;j<1275; j++);
}

void adc()                      // Function to read the values from ADC and display on the LED's.
{
    rd=1;
    wr=0;
    delay(1);
    wr=1;
    while(intr==1);
    rd=0;
```

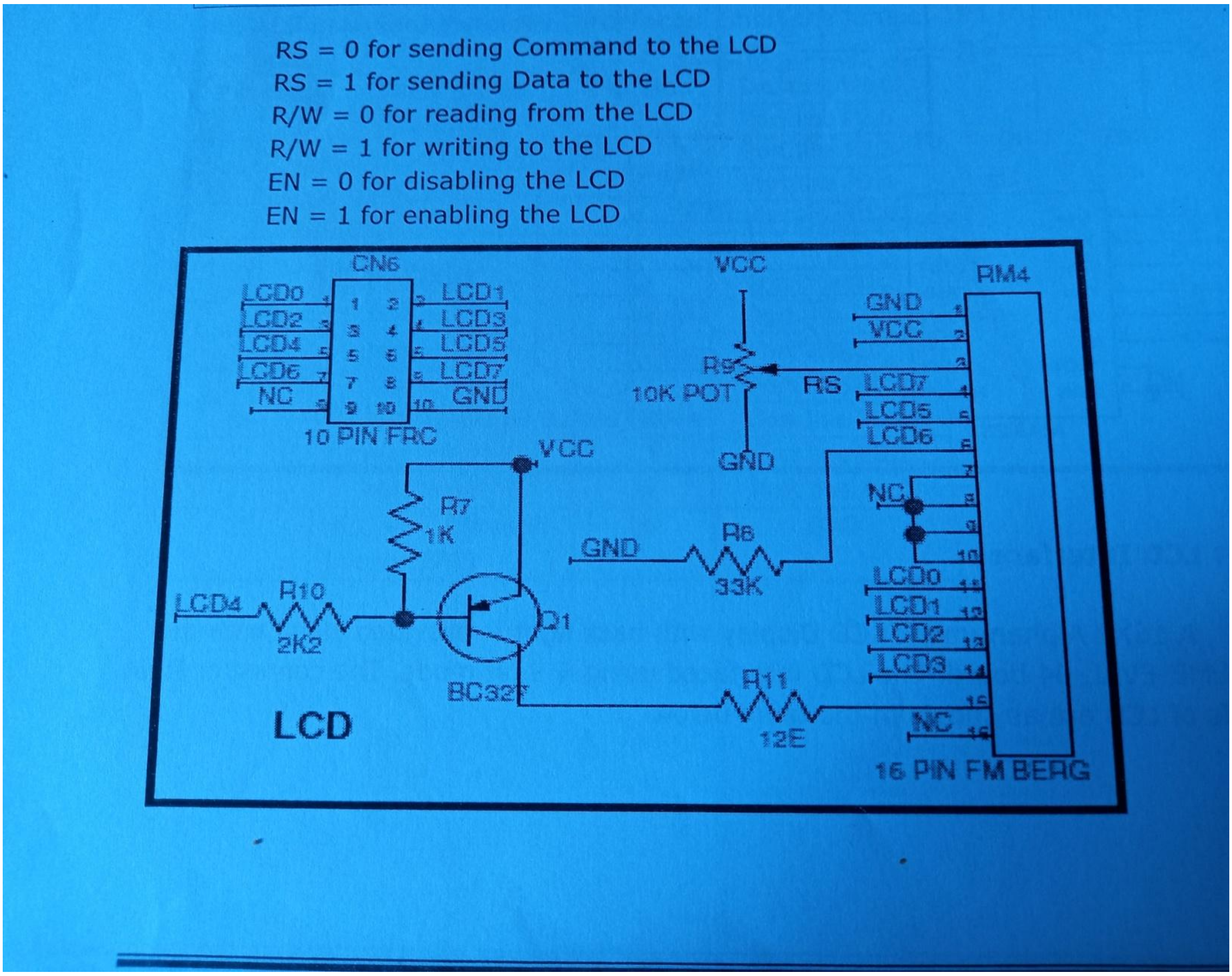


```
}

void adc()                      // Function to read the values from ADC and display on the LED's.
{
    rd=1;
    wr=0;
    delay(1);
    wr=1;
    while(intr==1);
    rd=0;
    output=input;
    delay(1);
    intr=1;
}

void main()
{
  input=0xff;                   // Declare port 0 as input port.
    while(1)
    {
   adc();
  }
}
```



For the analog voltage of 2.5 volts output is 10000000

LCD interface



RS = 0 for sending Command to the LCD
RS = 1 for sending Data to the LCD
R/W = 0 for reading from the LCD
R/W = 1 for writing to the LCD
EN = 0 for disabling the LCD
EN = 1 for enabling the LCD

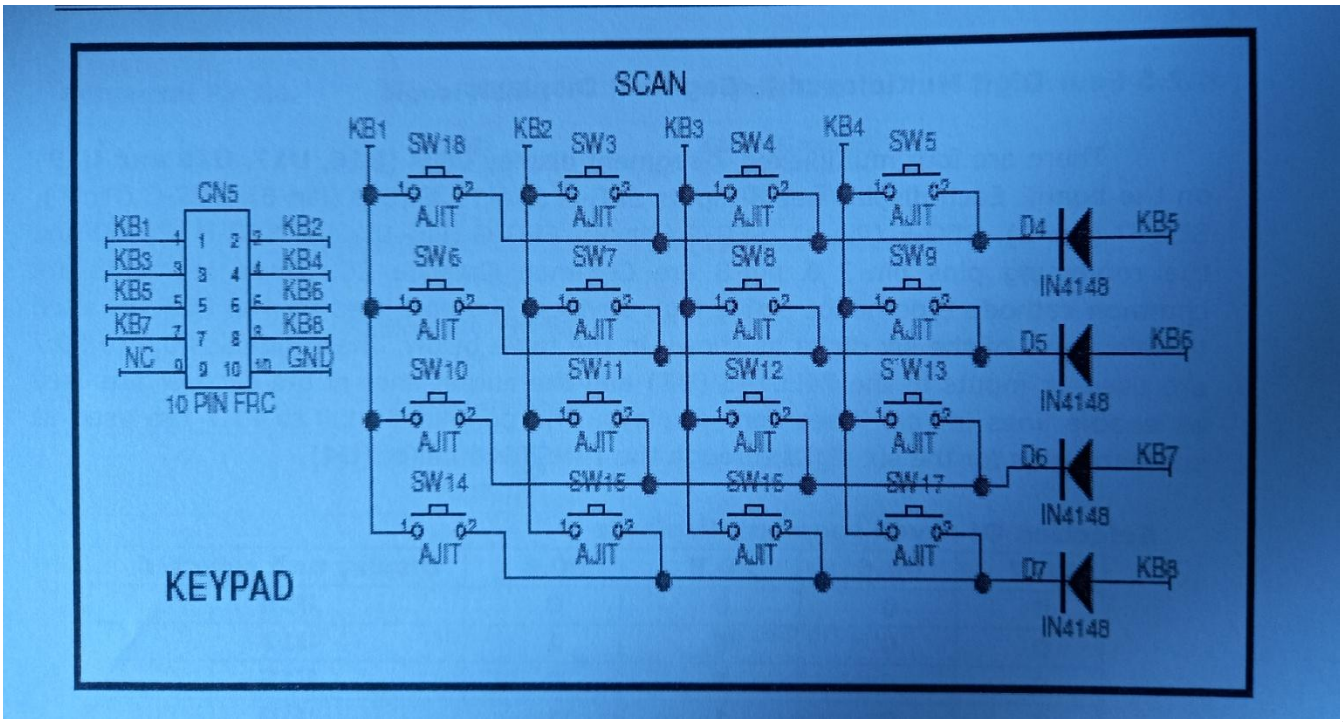
LCD routine:

```
/* LCD related functions. */
#include "at89c51ed2.h"
#include <intrins.h>       //For _nop_();

// LCD FUNCTION PROTOTYPE

extern void lcd_comm(void);
extern void wr_cn(void);
extern void wr_dn(void);
extern void lcd_data_cmnd(char, char);
void delay(int);

extern unsigned char temp1;
extern unsigned char temp2;
```

```
unsigned char var;

void lcd_init(void)
{
   temp1 = 0x30;   // D5(P3.3)=1,D4(P3.2)=1
        wr_cn();
        delay(500);

   temp1 = 0x30;   // D5(P3.3)=1,D4(P3.2)=1
        wr_cn();
        delay(500);

        temp1 = 0x30;       // D5(P3.3)=1,D4(P3.2)=1
        wr_cn();
        delay(500);

        temp1 = 0x20;       // D5(P3.3)=1
        wr_cn();
        delay(500);

   temp1 = 0x28;   // Set
        lcd_comm();
        delay(500);

        temp1 = 0x0f;  //display on, cursor on, cursor blinking
        lcd_comm();
        delay(500);

        temp1 = 0x06;  //shift cursor right with auto increment
        lcd_comm ();
        delay (500);

        temp1 = 0x80;  //clear display with cursor on first position
        lcd_comm ();
        delay (500);

        temp1 = 0x01;
        lcd_comm ();
        delay(500);
}

// Function to pass commands to LCD
void lcd_comm(void)
{
        var = temp1;
        temp1 = temp1 & 0xf0;
        temp1 = temp1 >> 4;

        wr_cn ();

        temp1 = var & 0x0f;
        wr_cn ();

        delay (60);
}

// Function to pass data to LCD
Void lcd_data (void)
{
        var = temp2;
        temp2 = temp2 & 0xf0;  // convert the byte into nibble
        temp2 = temp2 >> 4;
        wr_dn();

        temp2 = var & 0x0f;
//      temp2 = temp2 << 2;
        wr_dn();

        delay(60);
}

// Function to write to command reg of LCD
void wr_cn(void)
{
        temp1 = temp1 & 0x7f; //        RS(P3^7)=0
        temp1 = temp1 & 0xDF;
        temp1 = temp1 | 0x40;   //      EN(P3^6)=1, TXD(P3^1)=1, RXD(P3^0)=1

        P2 = temp1;

        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();


        temp1 = temp1 & 0xbf; //        EN(P3^6)=0,
        P2 = temp1;

}
```

```
// Function to write to data reg of LCD
void wr_dn(void)
{
        temp2 = temp2 | 0xc0;   //      RS(P3^7)=1,EN=1,TXD=1,RXD=1
        temp2 = temp2 & 0xDF;
        P2 = temp2;

        _nop_();
        _nop_();
        _nop_();
        _nop_();
        _nop_();

        temp2 = temp2 & 0xbf; //        EN = 0
        P2 = temp2;
}

// Function to clear the LCD display
/*void clear_lcd()
{
         temp1 = 0x01;
         lcd_comm();
         delay(500);
}
          */
void delay(int count)
{
        int i;

        for(i=0;i<count;i++);
}
 /*
void lcd_data_cmnd(char cmnd, char l_data)
{
        char check;

        check =cmnd;
        var = l_data;

        l_data = l_data & 0xf0;
        temp1 = l_data >> 2;
        if(check == 1)
                wr_cn();
        else if(check == 2)
                {
                temp2 = temp1;
                wr_dn();
                }
        l_data = var & 0x0f;
        temp1 = l_data << 2;
        if(check == 1)
                wr_cn();
        else if(check == 2)
                        temp2 = temp1;
                        wr_dn();

        delay(60);
}
```

KEYBOARD INTERFACE:



```
/
*



Program to demonstrate keyboard operation
Takes a key from key board and displays it on LCD screen*/

#include "at89c51xd2.h"

// This project includes the following files:
// 1. kbdisp.c   the source program   to keypad
// 2. LCD_routine.c  to display the key       read

void scan(void);
void get_key(void);
void display(void);
void delay_ms(int i);
void uart_init(void);

/******* LCD FUNCTION PROTOTYPE*******/
void lcd_init(void);
void lcd_comm(void);
void lcd_data(void);
void clear_lcd(void);
void delay(int);

unsigned char temp1=0x00;
unsigned char temp2;

idata unsigned char  row,col,key;

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

unsigned char LED_CODE[16]= {0x3f,0x66,0x7f,0x39,
                              0x06,0x6d,0x6f,0x5e,
                              0x5b,0x7d,0x77,0x79,
                              0x4f,0x07,0x7c,0x71};

unsigned char LCD_CODE[16] = {'0','4','8','C',
                               '1','5','9','D',
                               '2','6','A','E',
                               '3','7','B','F'          };

idata unsigned char temp,temp4,temp3,res1,flag,result=0x3F;
 /* in this key pad program keypad lines are not kept high .we will make them high
           low throgh writing data to port.*/
void main ()
{
        lcd_init();

        while(1)
        {
                get_key();
                display();
                P3 = 0xFF;
        }
}  //end of main()



void get_key(void)                              // get_key() function calls scan() function
   {                                                  // on sensing a key from scan()
function it
    int i;                      // will compare the received scan code with
    display();                  // scan code lookup table and returns ASCII code
    flag = 0x00;                // rows are read from Port P0 is scan() function
    while(flag == 0x00)             // this function is in an eternal loop
                             // wiil return to main() only after getting a key
       {
             for(row=0;row<4;row++)              // This for loop makes the one of the
ROW low at
             {                                          // one time .Then scan function
is called
                if( row == 0)
                    temp3=0xFE;
                else if(row == 1)
                    temp3=0xFD;
                else if(row == 2)
                    temp3=0xFB;
                else if(row == 3)
                    temp3=0xF7;
             P0 = temp3;                         // each time temp3 value is put into
  this
                    scan();
                    delay_ms(10);                // on sensing a key scan() function
will make flag = 0xff
                    if(flag == 0xff)
                    break;
             }                                          // end of for

             if(flag == 0xff)
             break;
       }  // end of while

             P3 = 0x00;                          //       Enable U21
             for(i=0;i<16;i++)             // in this for for loop scan code received
which is in res1
             {                                          // variable is compared with
             if(scan_code[i] == res1) // the lookup table for array scan code[] and
when a match is
                                                        // found will return the
correspoding led code for the key pressed
                          {
                          temp1 = 0x87;
                          lcd_comm();
                          temp2 = LCD_CODE[i];
                          lcd_data();
                          result =  LED_CODE[i];
                          break;
                          }

                    }
```

```
}                                                                   // end of get_key();

void scan(void)                                              // will return the scan code for the
key pressed
   {                                             // Both row lines and column lines  are connected
to
              unsigned char t;                       //Port 0 only.Columns are connected to
P0.0-P0.3
              temp4 = P0;                                       // P0.4-P0.7 are connected to
rows
              temp4 = temp4 & 0xF0;    //read port0 ,mask with 0xF0h

       if(temp4 != 0xF0)              // Means a key is sensed
       {
              delay_ms(30);
              delay_ms(30);              // give some delay for debouncing

     temp4 = P0;                // read the port again
              temp4 = temp4 & 0xF0;

       if(temp4 != 0xF0)           // debounce
       {
              flag = 0xff;           // set the flag denoting a key is received
              res1 = temp4;
              t = temp3 & 0x0F;
              res1 = res1 | t;              // and OR it with column value
       }                                                     // to get the scan code of the key
pressed
       else
              {
              flag = 0x00;
              }

       }
}                         // end of scan()

    void display(void)
     {
   P1 = result;
     }

    void delay_ms(int i)
   {
  int j;
  for(j=0;j<i;j++);
    }
```

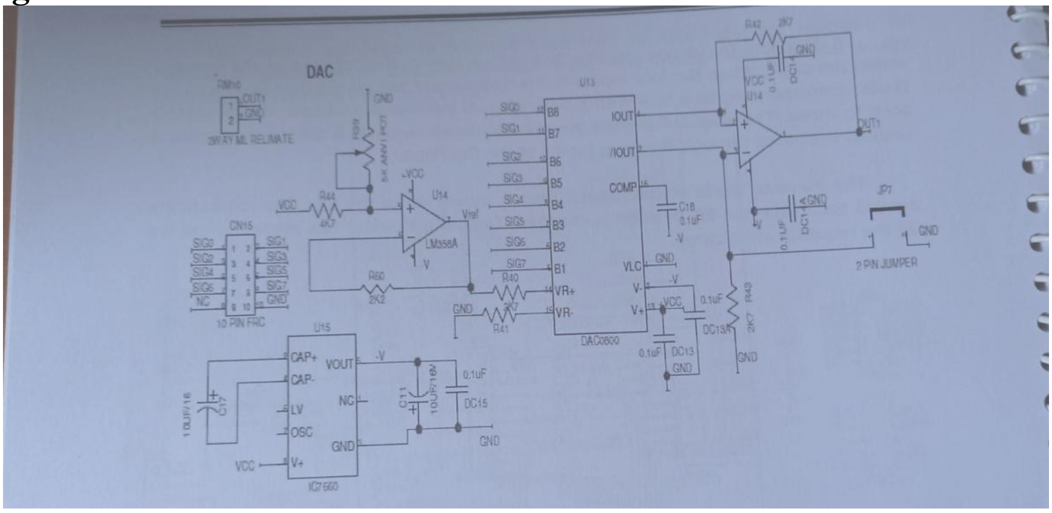
DAC INTERFACE:

1. Interfacing DAC with 8051 and developing an algorithm to generate the followingoutputs

- **I) Square wave with 50% duty cycle**
- **II) Square wave with 75% duty cycle**
- **III) Triangle wave.**
- **IV) Ramp(+ve &-ve)**
- **V) Sine wave**
- **VI) Staircase**
- **VII) Staircase triangle**

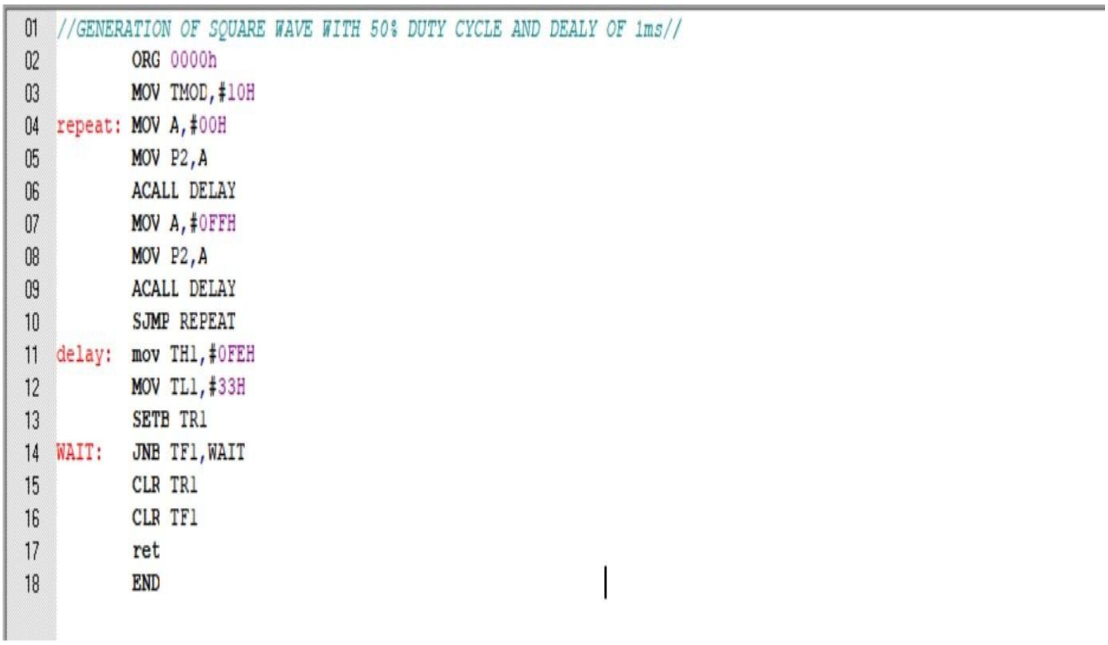
Solution:

Interfacing Circuit:



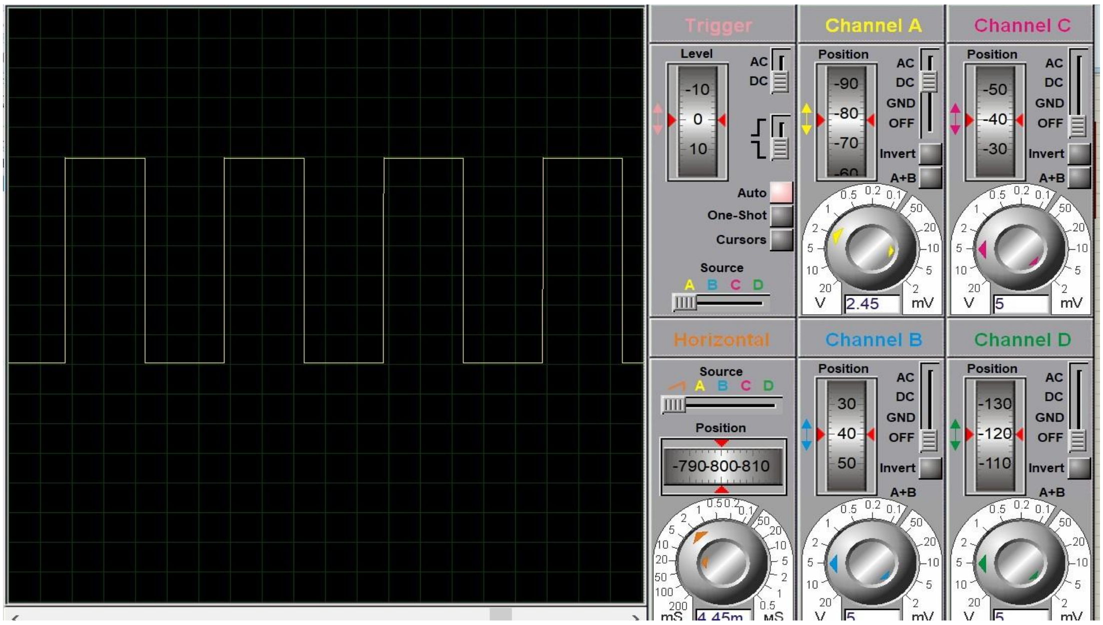
Generation of square wave with 50%duty cycle.

Solution:



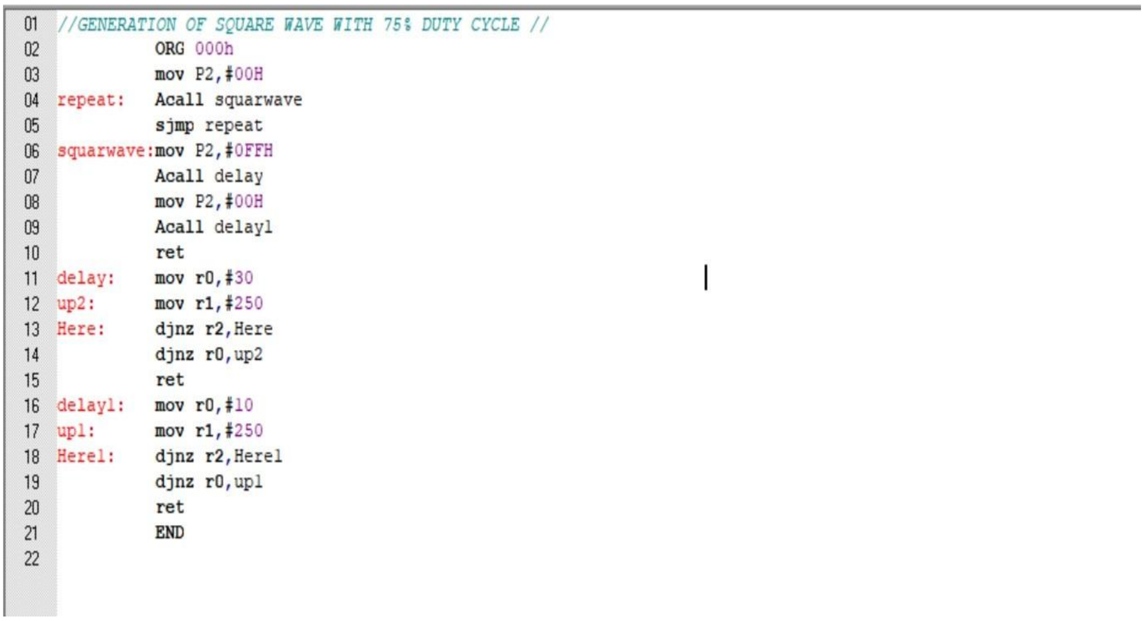
```
//GENERATION OF SQUARE WAVE WITH 50% DUTY CYCLE AND DEALY OF 1ms//
        ORG 0000h
        MOV TMOD,#10H
repeat: MOV A,#00H
        MOV P2,A
        ACALL DELAY
        MOV A,#0FFH
        MOV P2,A
        ACALL DELAY
        SJMP REPEAT
delay:  mov TH1,#0FEH
        MOV TL1,#33H
        SETB TR1
WAIT:   JNB TF1,WAIT
        CLR TR1
        CLR TF1
        ret
        END
```

RESULT:



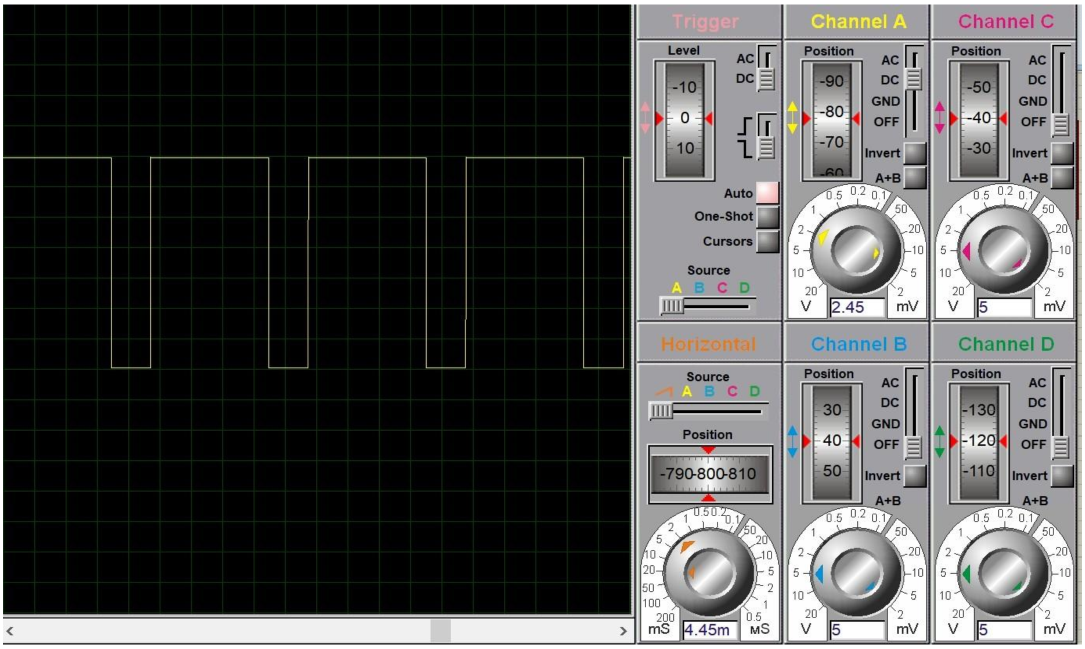
2.Generation of square wave with 75%duty cycle.

Solution:



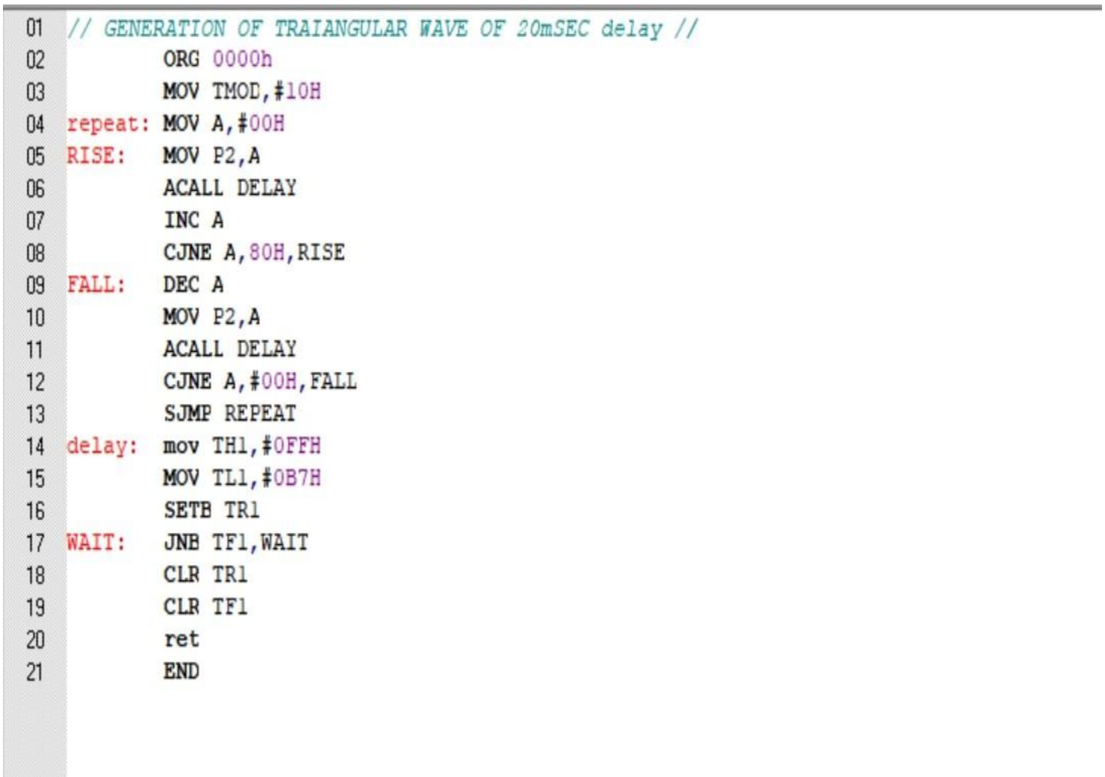
```
//GENERATION OF SQUARE WAVE WITH 75% DUTY CYCLE //
           ORG 000h
           mov P2,#00H
repeat:    Acall squarwave
           sjmp repeat
squarwave:mov P2,#0FFH
           Acall delay
           mov P2,#00H
           Acall delay1
           ret
delay:     mov r0,#30
up2:       mov r1,#250
Here:      djnz r2,Here
           djnz r0,up2
           ret
delay1:    mov r0,#10
up1:       mov r1,#250
Here1:     djnz r2,Here1
           djnz r0,up1
           ret
           END
```

RESULT:



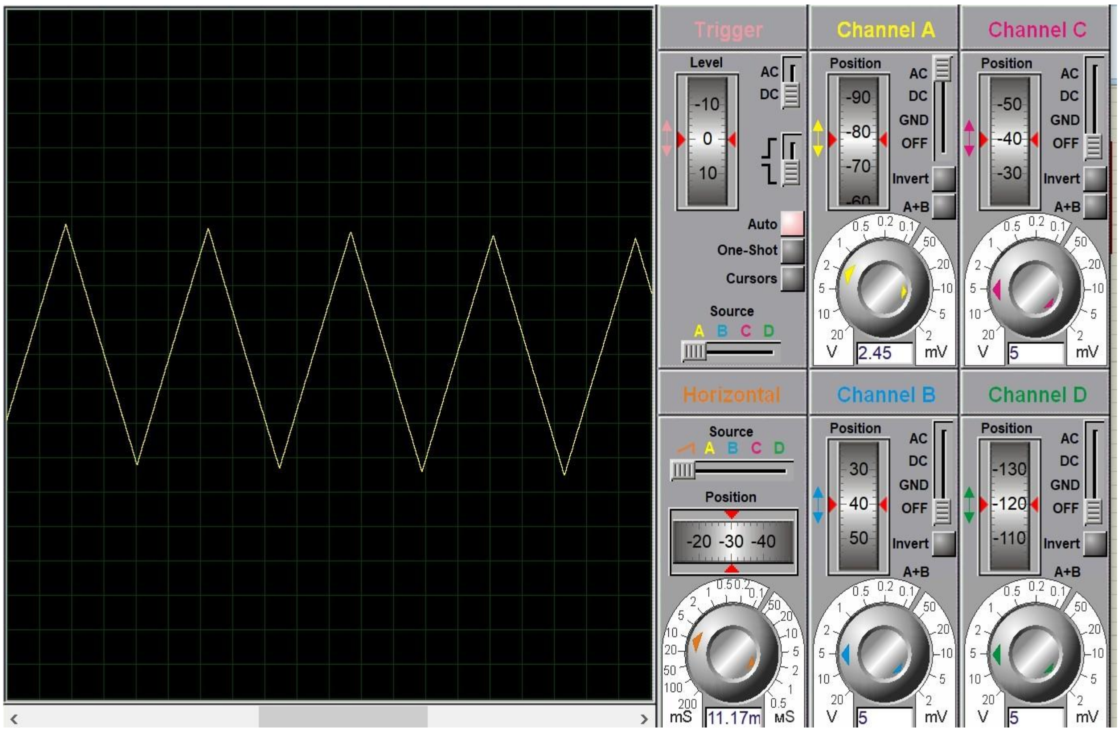
Generation of Triangular wave.

➢ ALGORITHM



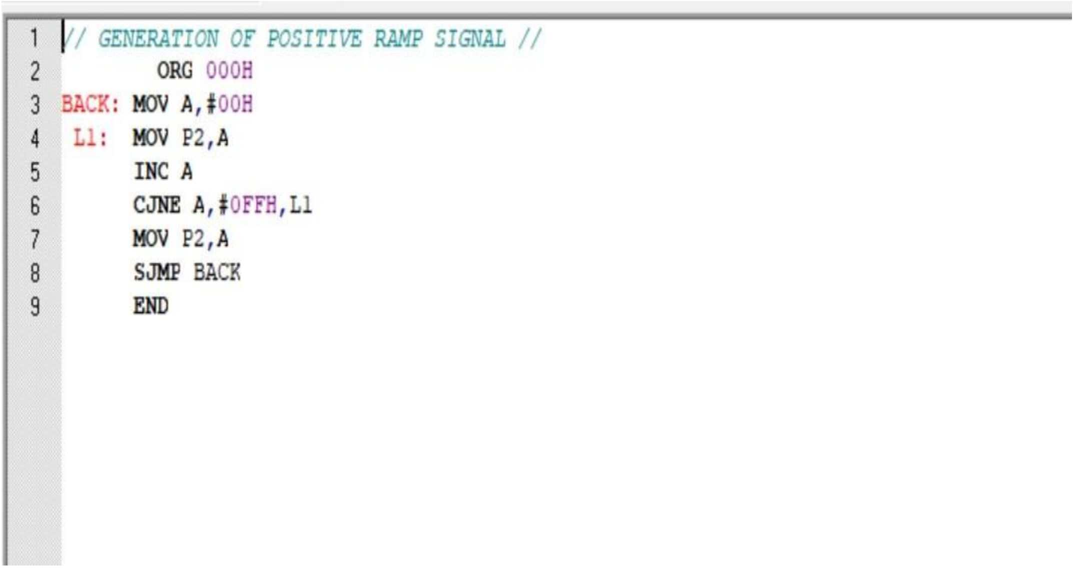
```
// GENERATION OF TRAIANGULAR WAVE OF 20mSEC delay //
        ORG 0000h
        MOV TMOD,#10H
repeat: MOV A,#00H
RISE:   MOV P2,A
        ACALL DELAY
        INC A
        CJNE A,80H,RISE
FALL:   DEC A
        MOV P2,A
        ACALL DELAY
        CJNE A,#00H,FALL
        SJMP REPEAT
delay:  mov TH1,#0FFH
        MOV TL1,#0B7H
        SETB TR1
WAIT:   JNB TF1,WAIT
        CLR TR1
        CLR TF1
        ret
        END
```

RESULT:



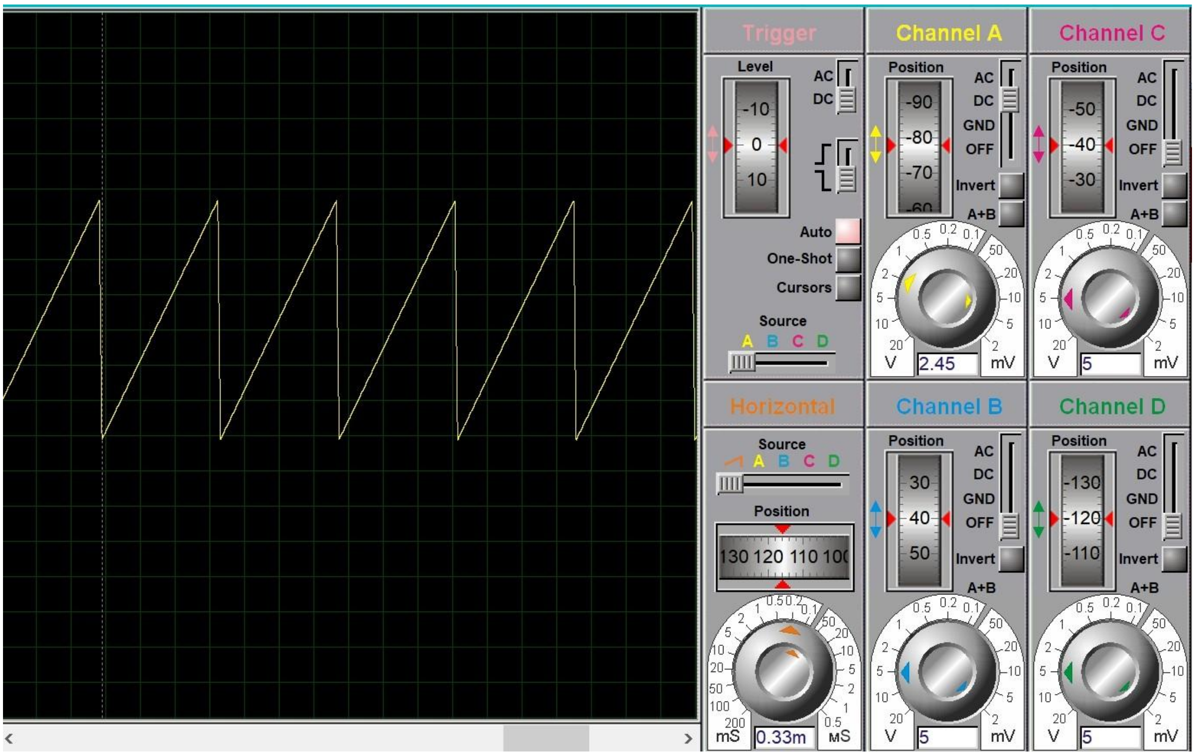
I) A. Generation of positive Ramp with signal.

ALGORITHM



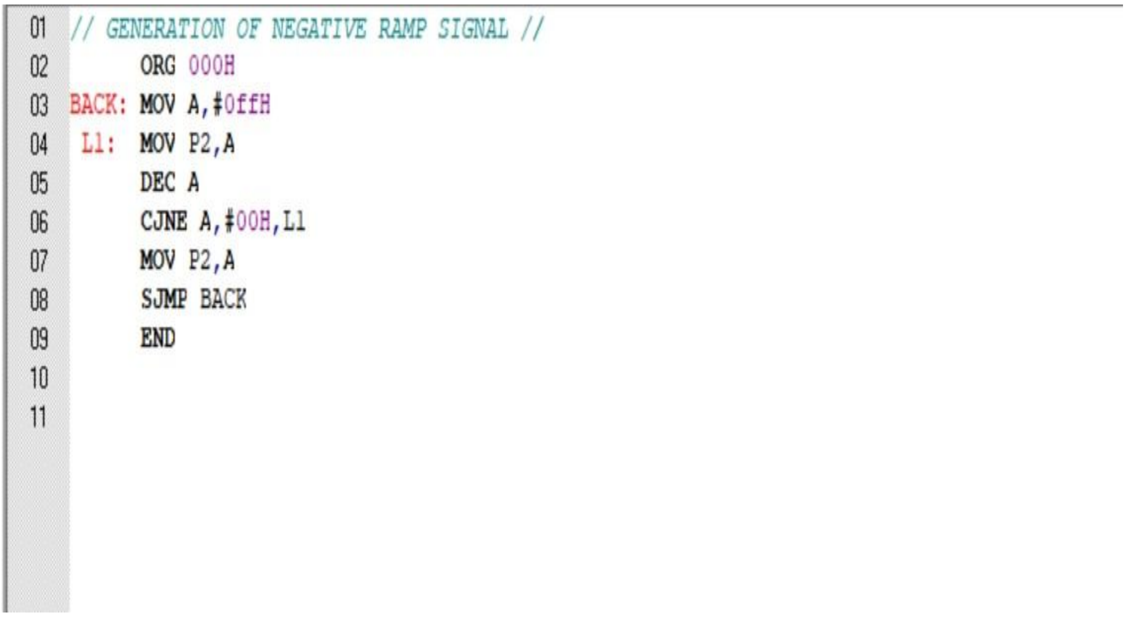
```
// GENERATION OF POSITIVE RAMP SIGNAL //
        ORG 000H
BACK: MOV A,#00H
 L1:  MOV P2,A
      INC A
      CJNE A,#0FFH,L1
      MOV P2,A
      SJMP BACK
      END
```

RESULT:



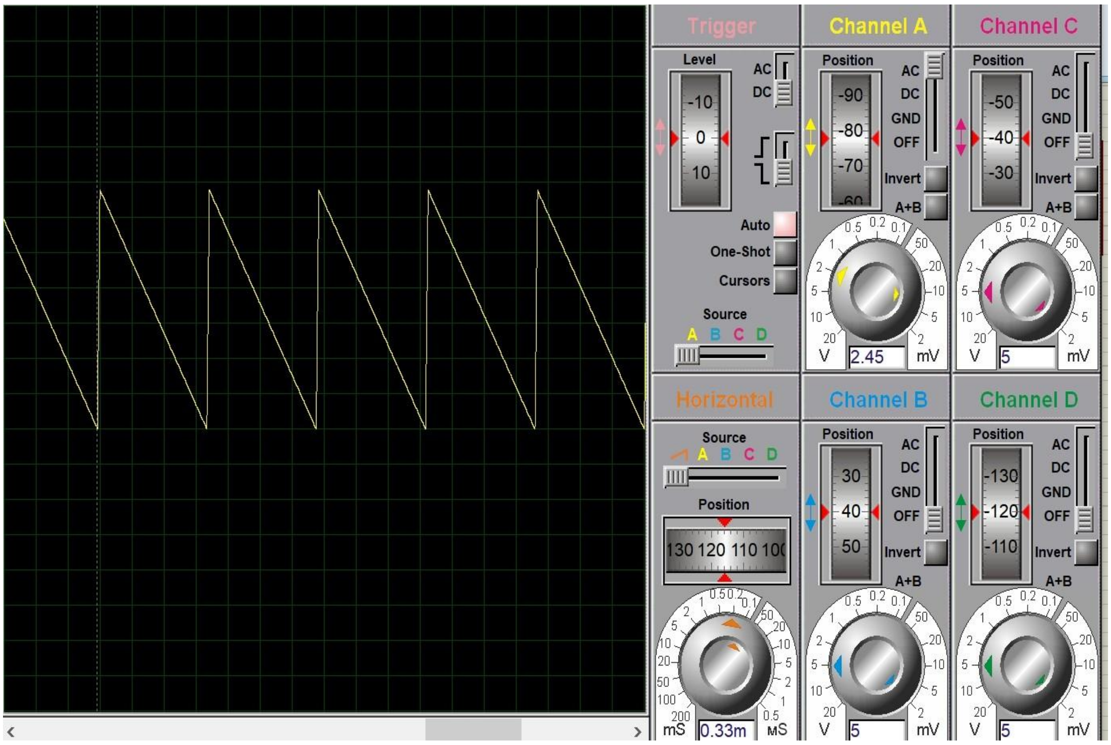
B. Generation of negative Ramp with signal.

ALGORITHM



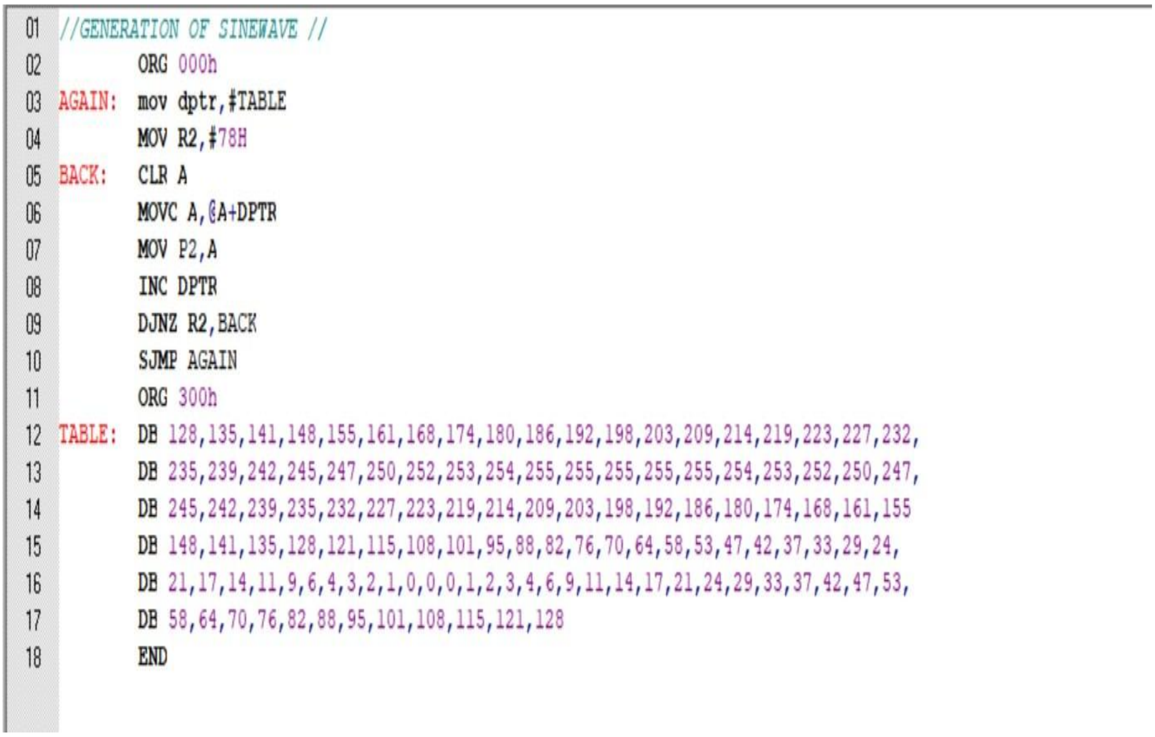
```
// GENERATION OF NEGATIVE RAMP SIGNAL //
       ORG 000H
BACK:  MOV A,#0ffH
 L1:   MOV P2,A
       DEC A
       CJNE A,#00H,L1
       MOV P2,A
       SJMP BACK
       END
```

RESULT:



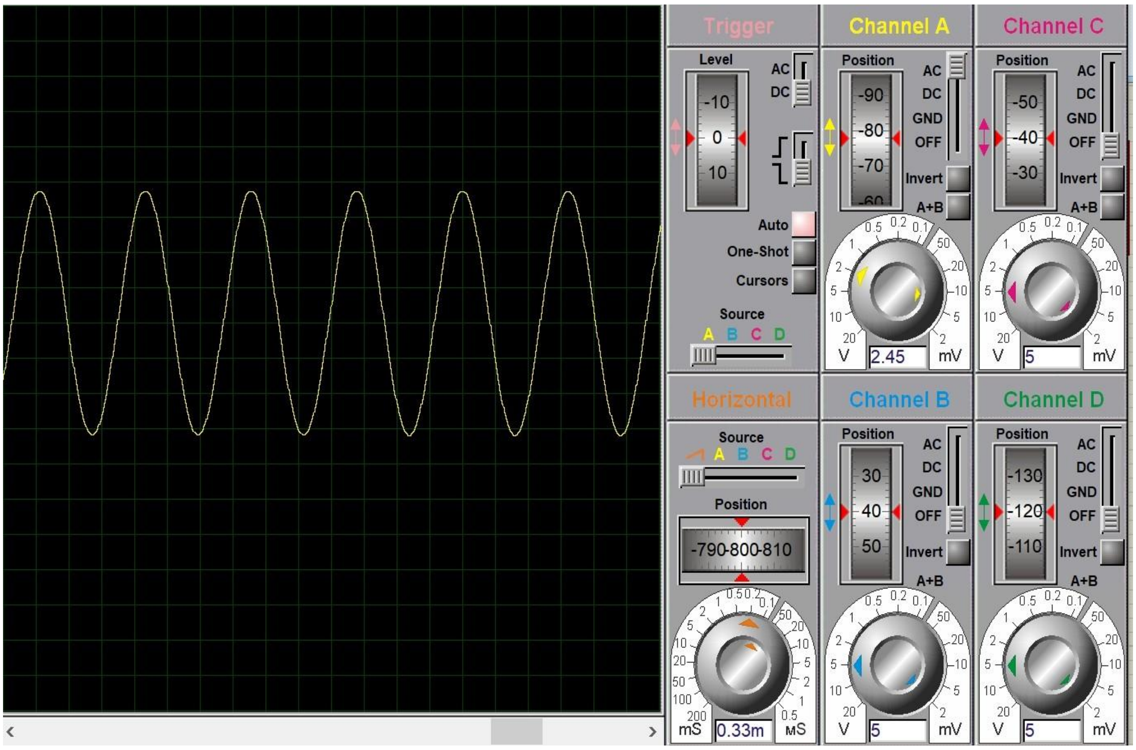
GENERATION OF SINE WAVE

ALGORITHM

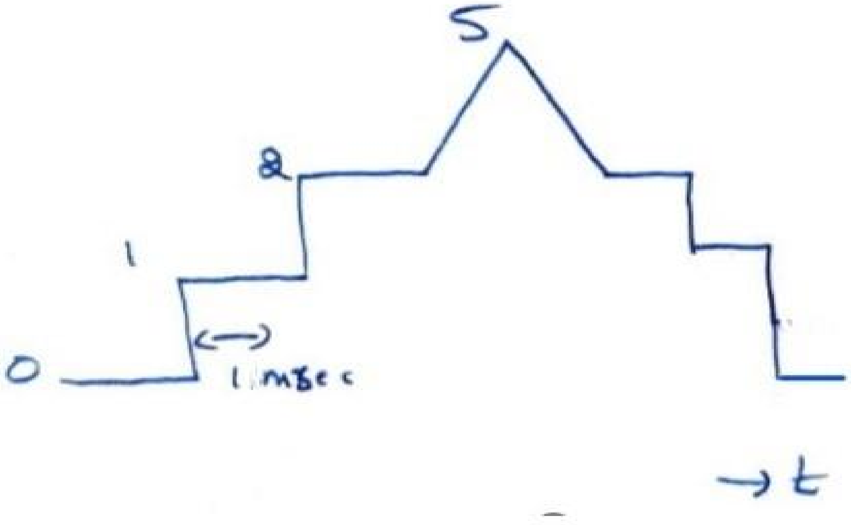


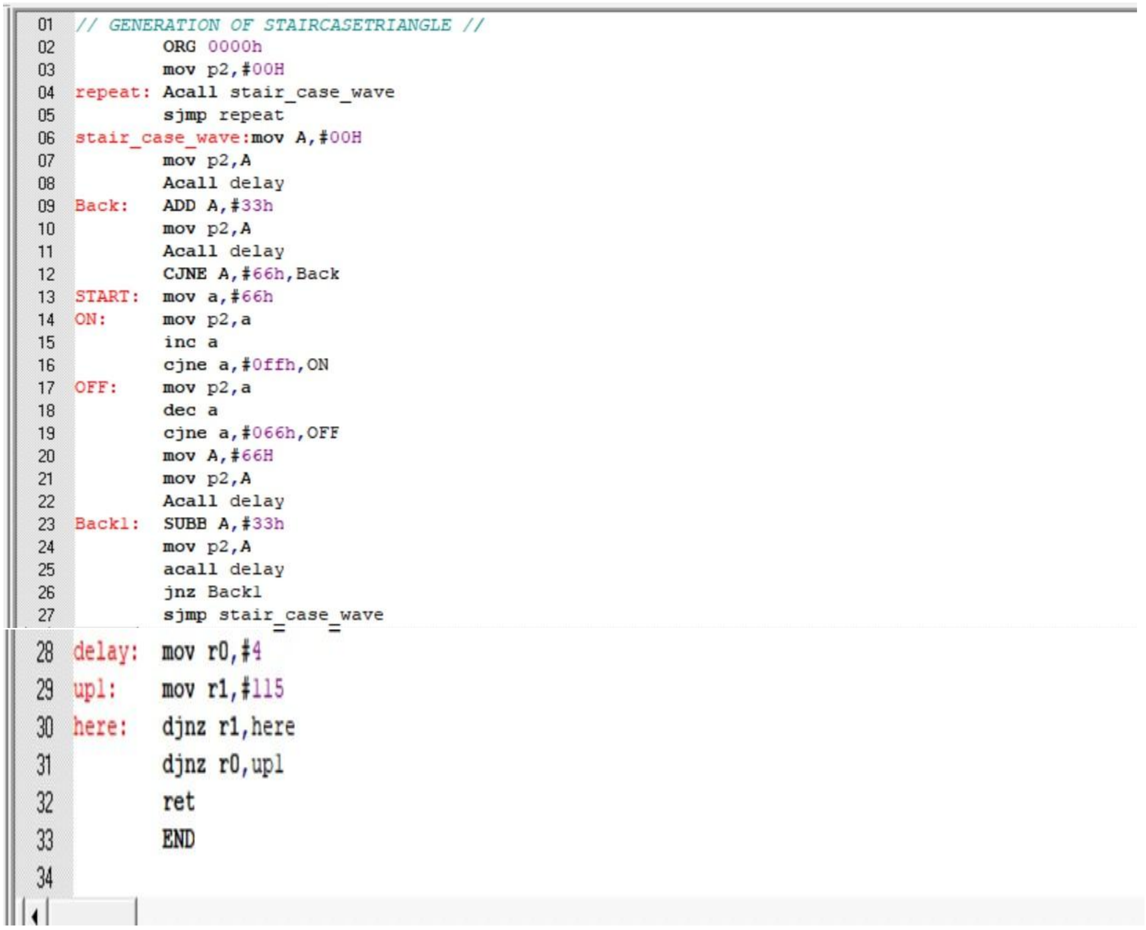
```
//GENERATION OF SINEWAVE //
        ORG 000h
AGAIN:  mov dptr,#TABLE
        MOV R2,#78H
BACK:   CLR A
        MOVC A,@A+DPTR
        MOV P2,A
        INC DPTR
        DJNZ R2,BACK
        SJMP AGAIN
        ORG 300h
TABLE:  DB 128,135,141,148,155,161,168,174,180,186,192,198,203,209,214,219,223,227,232,
        DB 235,239,242,245,247,250,252,253,254,255,255,255,255,255,254,253,252,250,247,
        DB 245,242,239,235,232,227,223,219,214,209,203,198,192,186,180,174,168,161,155
        DB 148,141,135,128,121,115,108,101,95,88,82,76,70,64,58,53,47,42,37,33,29,24,
        DB 21,17,14,11,9,6,4,3,2,1,0,0,0,1,2,3,4,6,9,11,14,17,21,24,29,33,37,42,47,53,
        DB 58,64,70,76,82,88,95,101,108,115,121,128
        END
```

RESULT



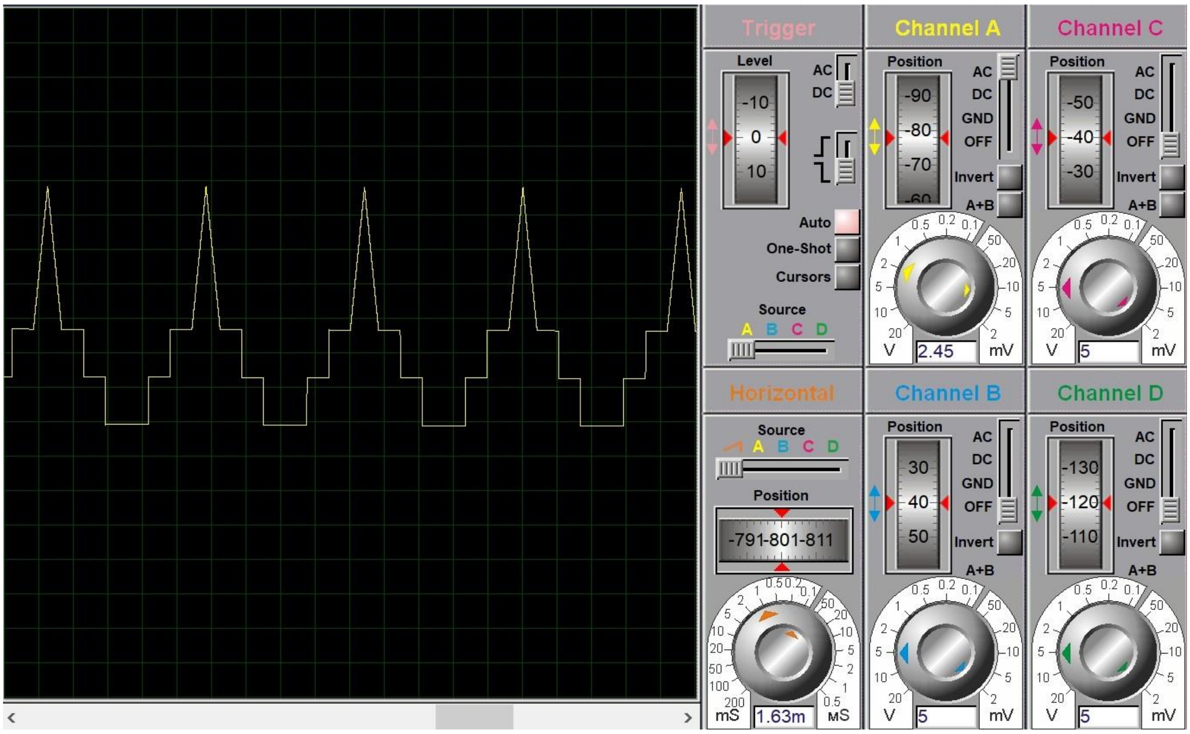
II) Generation of the following waveform.





```
// GENERATION OF STAIRCASETRIANGLE //
        ORG 0000h
        mov p2,#00H
repeat: Acall stair_case_wave
        sjmp repeat
stair_case_wave:mov A,#00H
        mov p2,A
        Acall delay
Back:   ADD A,#33h
        mov p2,A
        Acall delay
        CJNE A,#66h,Back
START:  mov a,#66h
ON:     mov p2,a
        inc a
        cjne a,#0ffh,ON
OFF:    mov p2,a
        dec a
        cjne a,#066h,OFF
        mov A,#66H
        mov p2,A
        Acall delay
Back1:  SUBB A,#33h
        mov p2,A
        acall delay
        jnz Back1
        sjmp stair_case_wave
delay:  mov r0,#4
up1:    mov r1,#115
here:   djnz r1,here
        djnz r0,up1
        ret
        END
```

RESULT


Trigger
Level
-10
0
10
AC
DC
Auto
One-Shot
Cursors
Source
A B C D
Channel A
Position
-90
-80
-70
AC
DC
GND
OFF
Invert
A+B
V
2.45
mV
Channel C
Position
-50
-40
-30
AC
DC
GND
OFF
Invert
A+B
V
5
mV
Horizontal
Source
A B C D
Position
-791-801-811
mS
1.63m
MS
Channel B
Position
30
40
50
AC
DC
GND
OFF
Invert
A+B
V
5
mV
Channel D
Position
-130
-120
-110
AC
DC
GND
OFF
Invert
A+B
V
5
mV