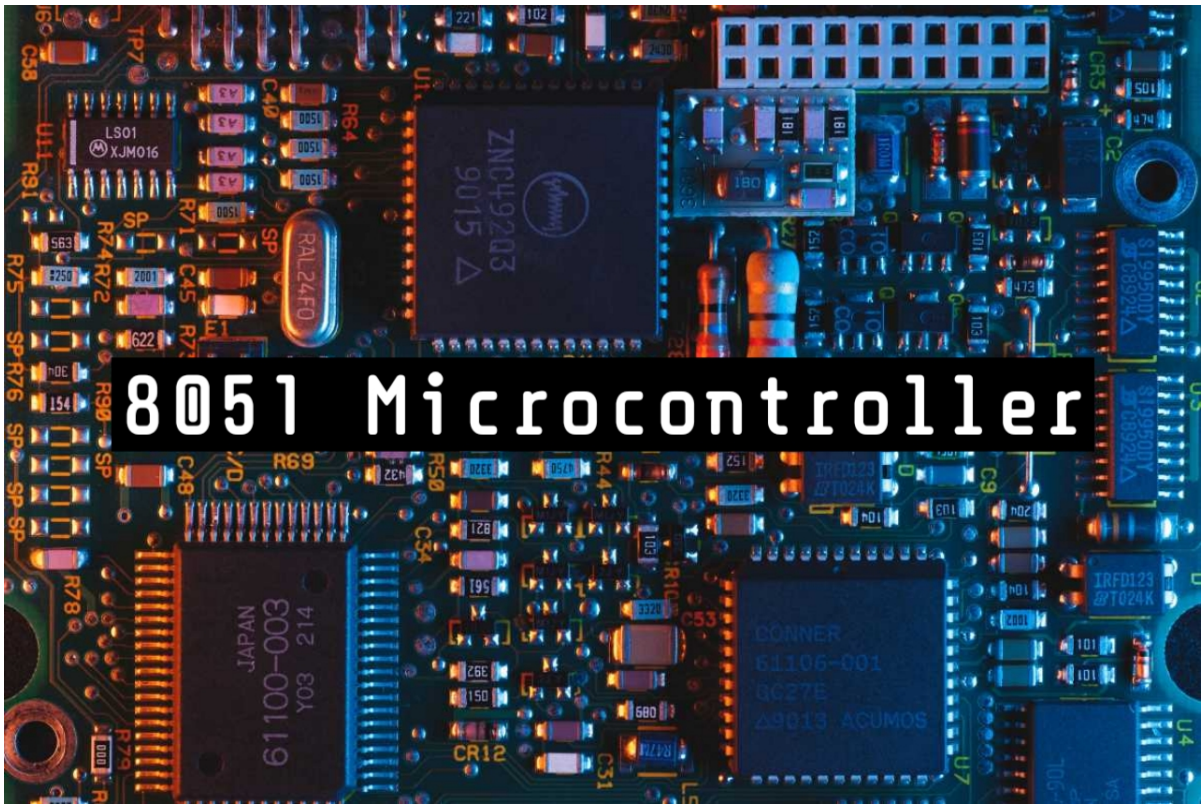
Microcontroller Laboratory Manual



Ballari Institute of Technology and Management

∗ Autonomous Institute under VTU, Belagavi ∗ Approved by AICTE, New Delhi ∗
∗ Recognized by Govt. of Karnataka ∗ NBA ∗ NAAC A+ ∗
∗ ISO 9001:2015 Certified Institution ∗
⋆"Janan Gangothri" campus,# 873/2, Ballari-Hosapete Road, Near Allipura, Ballari-583104⋆

This laboratory manual is designed to have students experience the use and applications of 8051 Microcontroller, they can apply programming concepts to design the embedded projects that will confront them in their personal, professional and civic lives.
The primary audience for this lab manual is 5^{th} ***Semester undergraduate students relevance to 2018 scheme. The Lab Manual is the revised from earlier edition of 2017 scheme.***
The content of manual is referred from text book titled "The 8051 Microcontroller and Embedded Systems Using Assembly and C", by Muhammad Ali Mazadi, Pearson-2nd Edition, 2008. And also from various online reference materials and websites.

First REVISED Reprint August 2020
Second REVISED Reprint September 2021

Prepared by:

Mr. Shridhar. S. M
Assistant Professor, Department of EEE,
Ballari Institute of Technology and Management,
Ballari.

Associates:

Mr. Manjunath. D
Instructor, Department of EEE,
BITM, Ballari.

Contents

1 Introduction to RIDE IDE

Getting Started

The Raisonance Integrated Development Environment (RIDE)tool provides a brief overview of the RKit51+XA package(s). Its main goal is to show how easy it is to implement projects based on either the 8-bit 80C51 or the 16-bit XA microcontrollers.

2 Ride

The Raisonance RC-51 compiler is a part of RKit51, which is a Windows integrated development environment (IDE) containing a set of development tools for 8051 microcontrollers applications. This development environment version 06.10.16 includes a syntax-highlighting editor, a project manager and online help.

The following software products can be supplied as software KIT, or bought separately.
- The ANSI-C RC-51 compiler, with its libraries.
- The MA-51 macro assembler.
- The LX-51 linker.
- The KR-51 kernel.
- The SIMICE-51 integrated debugger/simulator.

2.1 Creating New Project and Execution

2.1.1 Creating a New Project

1. Click on **Project** → select **New project** a new window appears, as shown in figure-1.

2. Specify **Application Name** which is "Project Name" (Give the suitable project name relevance to the Program which can be reopened/identified later).

3. Select the location to save the project by changing the **Directory**. This directory/folder will be considered as the base directory/folder for all the relative paths.

4. Select the target family as **80C51**.

5. Click on **Next** which opens a project window, as shown in figure-2.

6. Search and select the device **P80C51** and click on **terminate**.

7. The project creation with the path will appear at the top corner, as shown in figure-3.

2.2 Creating file and Linking File with Project

8. Click on **File** and select **New file**, as shown in figure-4.

9. Select the type of file **C file/Assembler file**.

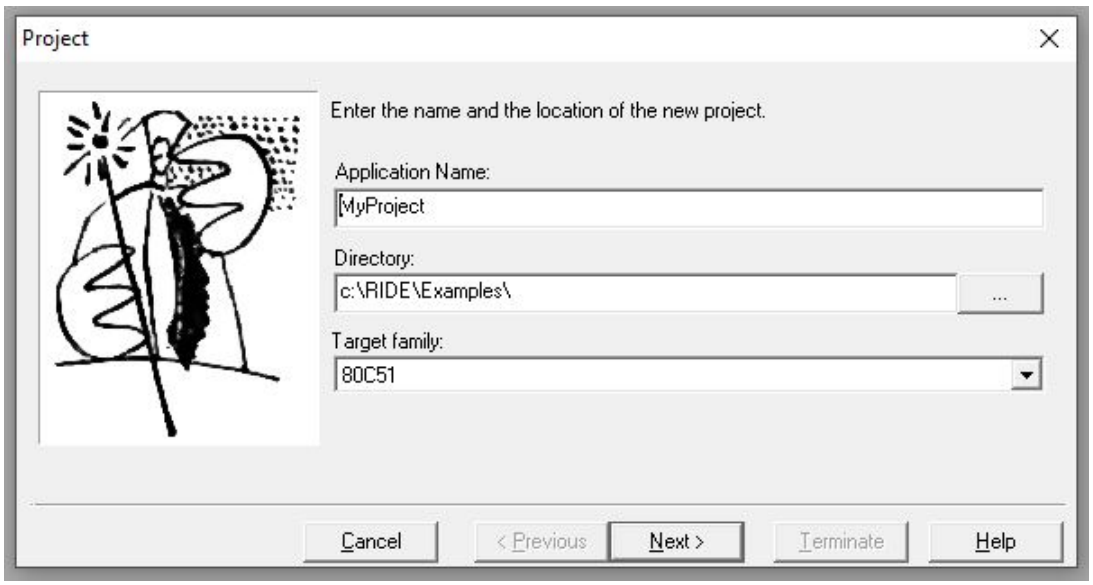


Figure 1: Project Name, Directory and Target family

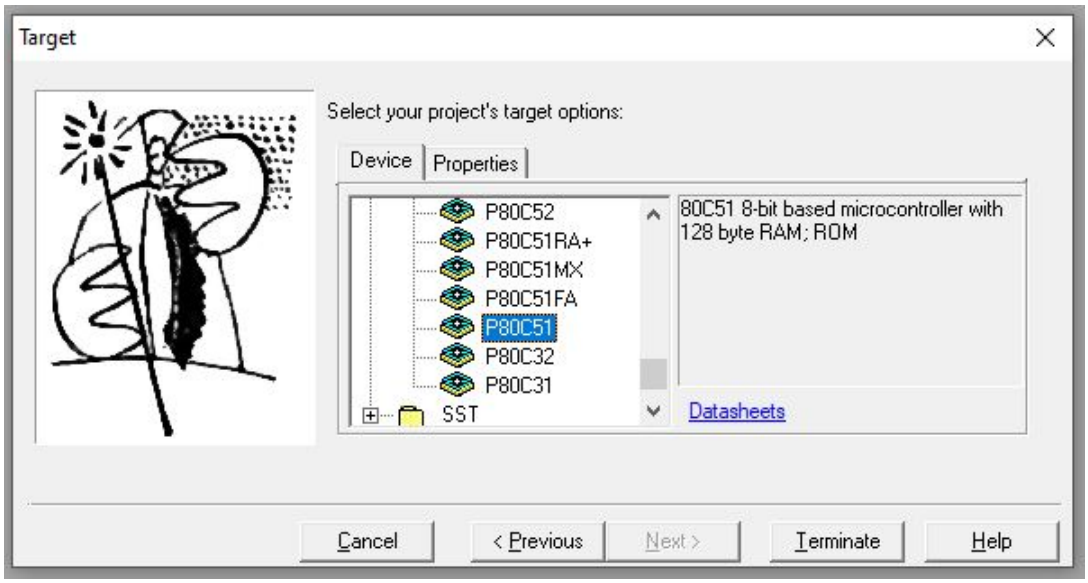


Figure 2: Device family

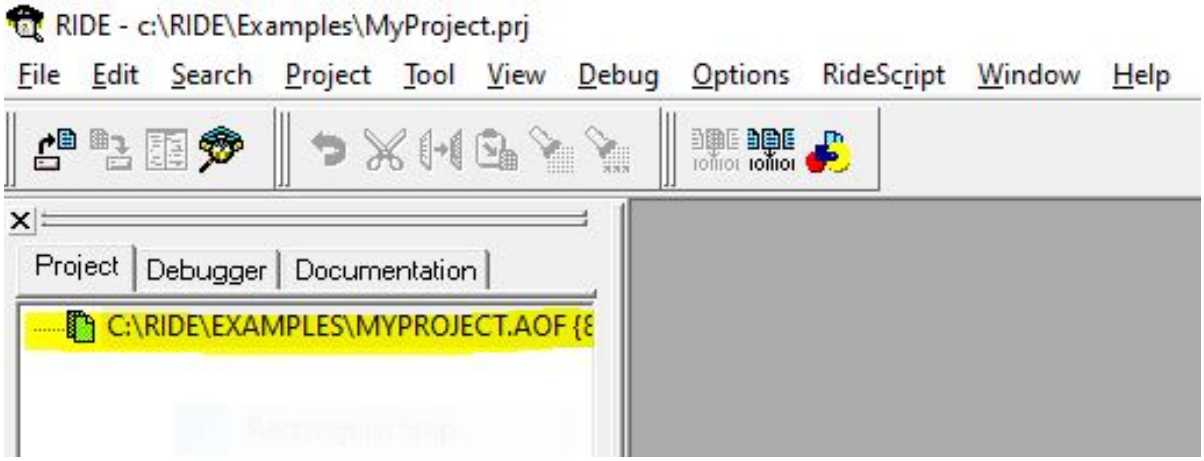


Figure 3: My Project Created

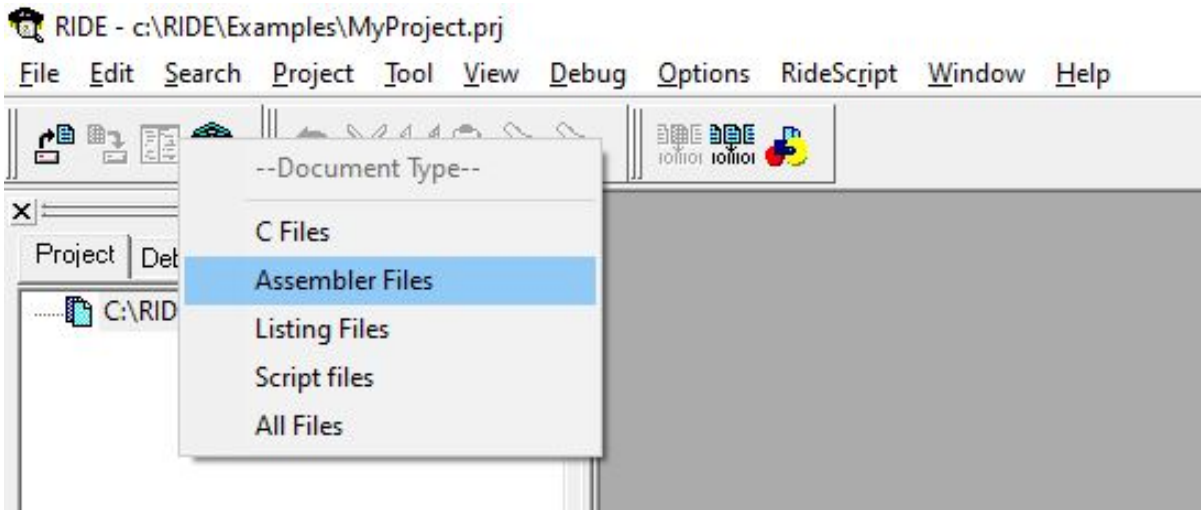


Figure 4: Selecting file

10. By selecting the **Assembler file** a **"untitled.a51"** editor window will open, as shown in figure-5.

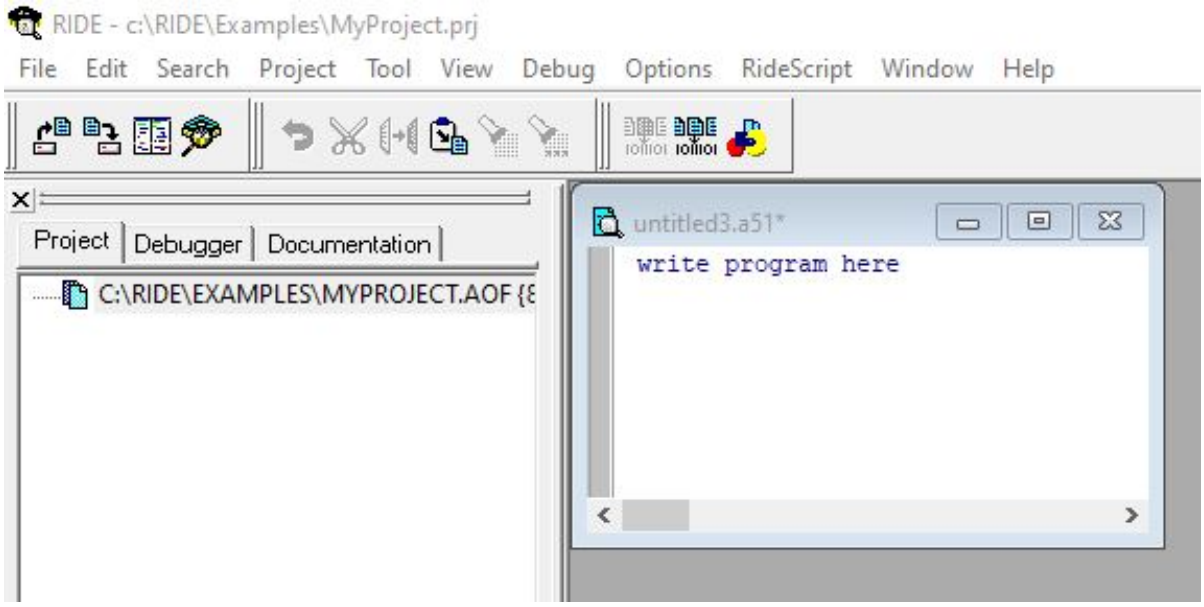


Figure 5: New file

11. Write the program and **Save** the assembler file, with the same file name as the Project Name (so that it can be easily linked with the project), with file extension of **.a51**. And for C program **Save** the C file, with file extension of **.c**. In RIDE the file extensions are assigned automatically depending on the file type.

2.2.1 Linking File with Project

12. Select on **Project** and select **Add node source/Application**, as shown in fig-6.

13. A new window appears, shown in figure-7. Select the type of file **C file/assembler file** to add with the project.

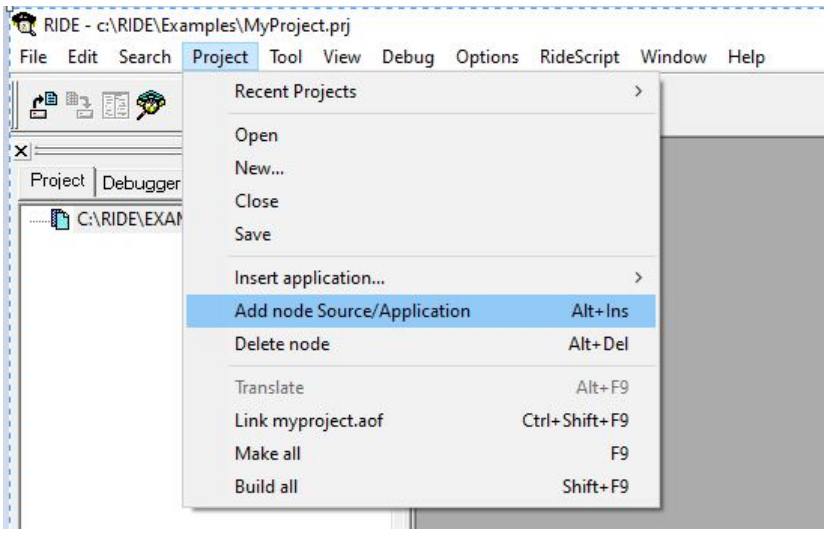


Figure 6: Add Node Source/ Application

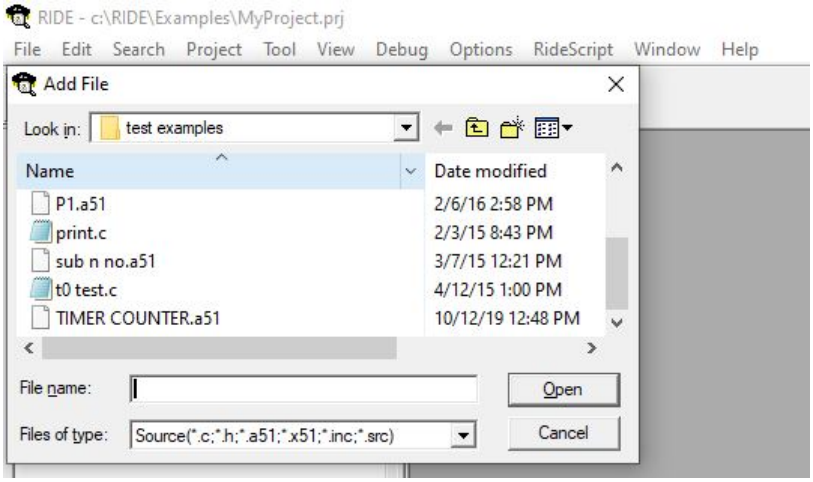


Figure 7: Selecting Assembly/C File

14. The selected **Assembler file/C file** will get attached to the Project, as shown in figure-8.

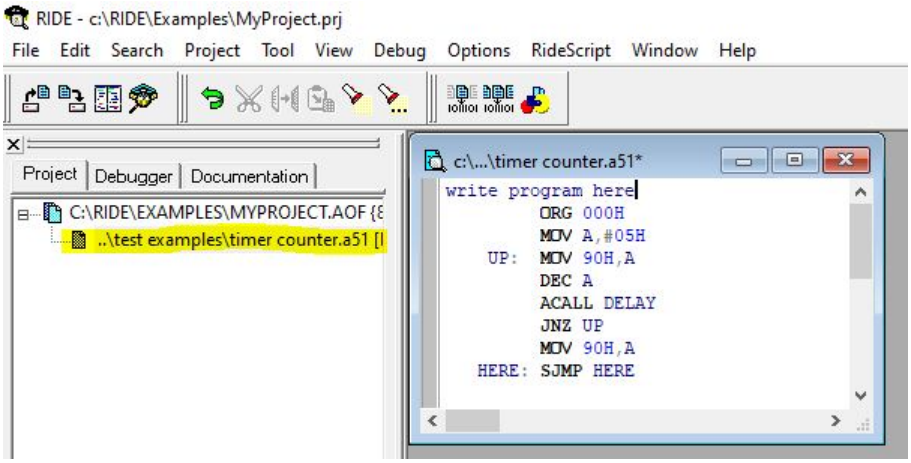


Figure 8: Attached File with My Project

3 List of Experiments:

Sl.No.	Experiments
	Software Programs
1	Data Transfer Programs – Programs for block data movement from internal RAM to internal RAM, external RAM to internal RAM, sorting, exchanging, finding largest element in an array.
2	Arithmetic Programs: Addition, subtraction, multiplication and division, Square and cube operations.
3	Code Conversion Programs – BCD to ASCII, ASCII to BCD, ASCII to Hexadecimal, Hexadecimal to ASCII, Decimal to Hexadecimal, Hexadecimal to and Decimal, use of Boolean and logical instructions.
4	Counters Programs using delay, Serial port Programming and on-chip timer/counters, sorting positive and negative numbers, sorting odd and even numbers, use of conditional call and return instructions.
	Interfacing Programs
5	Stepper motor interface- rotating in clock and anticlockwise direction and rotating in angle.
6	DC motor interface for direction and speed control using PWM.
7	Alphanumerical LCD panel interface.
8	Generate different waveforms: Sine, Square, Triangular, Ramp using DAC interface.
9	Elevator interface.

Note: *For the experiments 1 to 4, 8051 assembly programming is used. and for the experiments 5 to 9 Single chip solution for interfacing 8051 with C Programs.*

Software Programs

4 Experiment 1: Data Transfer Programs

Programs:

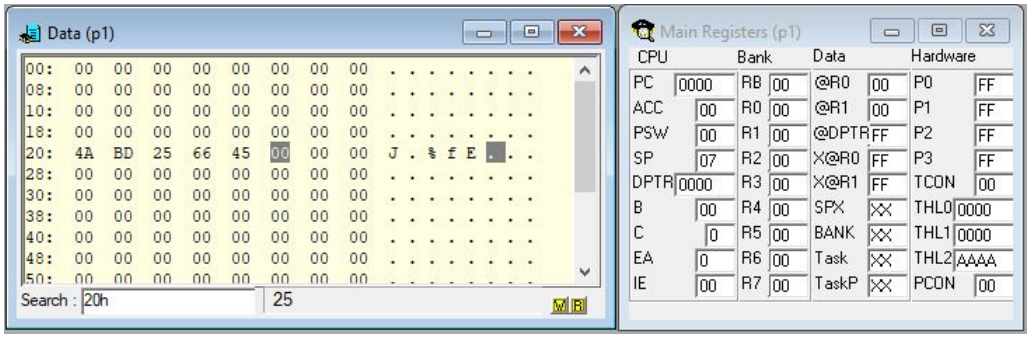
1. To move a block of 5 bytes of data from source 20h to destination 40h in internal RAM.
2. To move a block of 5 bytes of data from source 2000h to destination 2050h in external RAM.
3. To transfer a block of 8 bytes of data from source 9000h external RAM to destination 40h internal RAM.
4. To exchange the source block starting with address 20h in internal RAM containing 5 bytes of data with destination block starting with address 40h in internal RAM.
5. To exchange the source block starting with address 10h in internal RAM containing 5 bytes of data with destination block starting with address 2000h in external RAM.
6. To arrange 5 bytes of data in ascending order stored starting at 30h in internal RAM.
7. To arrange 5 bytes of data in descending order stored starting at 30h in internal RAM.
8. To find largest number from an array of 6 bytes stored starting from 20h in internal RAM and store the result at 40h in internal RAM.
9. To find smallest number from an array of 6 bytes stored starting from 20h in internal RAM and store the result at 45h in internal RAM.

Program 1.a: TO MOVE A BLOCK OF 5 BYTES OF DATA FROM SOURCE 20H TO DESTINATION 40H IN INTERNAL RAM.

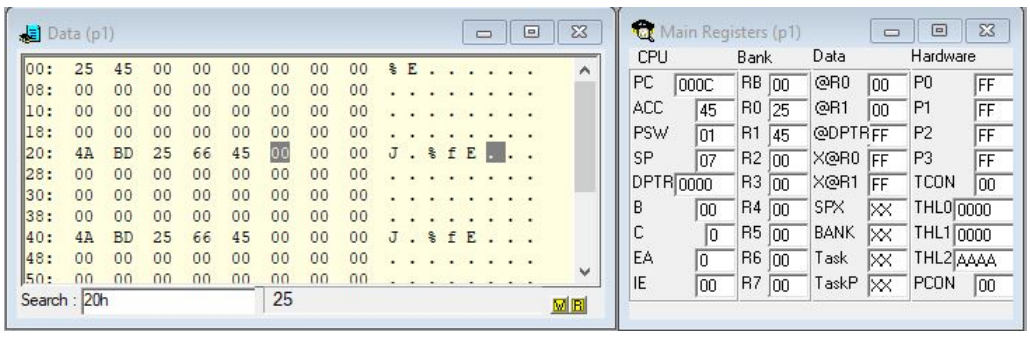
```
        ORG 0000H          ; start program at 0000h location
        MOV R0,# 20H       ; initialize R0 with beginning address of source in
                           internal memory
        MOV R1,# 40H       ; initialize R1 with beginning address of destina-
                           tion in internal memory
        MOV R5,# 05H       ; r5 is initialized with 5 counts
check:  MOV A, @R0         ; move the source data to accumulator
        MOV @R1,A          ; move the content of accumulator to destination
        INC R0             ; incremet source address
        INC R1             ; incremet destination address
        DJNZ R5, check     ; decrement R5 i.e., till count becomes zero
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

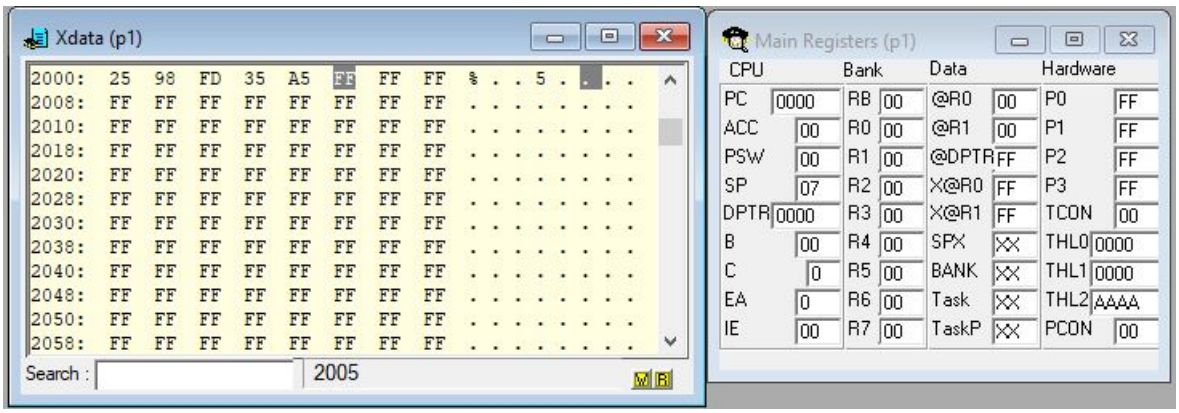


Program 1.b: TO MOVE A BLOCK OF 5 BYTES OF DATA FROM SOURCE 2000H TO DESTINATION 2050H IN EXTERNAL RAM.

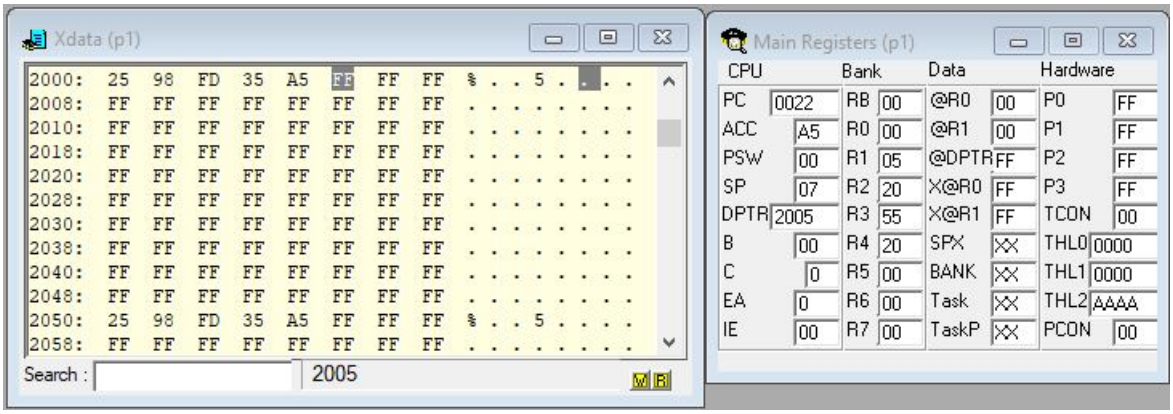
```
        ORG 0000H                ; start program at 0000h location
        MOV DPTR, # 2050H        ; intitlize destination address into DPTR
        MOV R7, # 05H            ; r7 is initialized with 5 counts
        MOV R3, 82H              ; save DPL destination address into R3 register
        MOV R4, 83H              ; save DPH destination address into R4 register
        MOV DPTR,# 2000H         ; intitlize source address into DPTR
loop:   MOVX A,@DPTR             ; move source data to accumulator
        INC DPTR                 ; increment source address
        MOV R1, 82H              ; save DPL source address into R1 register
        MOV R2, 83H              ; save DPH source address into R2 register
        MOV 82H, R3              ; take back DPL destination address from R3
        MOV 83H, R4              ; take back DPH destination address from R4
        MOVX @DPTR, A            ; move data from accumulator to destination
        INC DPTR                 ; increment destination address
        MOV R3, 82H              ; save DPL destination address into R3 register
        MOV R4, 83H              ; save DPH destination address into R4 register
        MOV 82H, R1              ; take back DPL destination address from R1
        MOV 83H, R2              ; take back DPH destination address from R2
        DJNZ R7, loop            ; decrement R7 i.e., till count becomes zero
stop:   SJMP stop                ; stop incrementing Program Counter
        END                      ; end of the program
```

Result:

Before Execution-



After Execution-

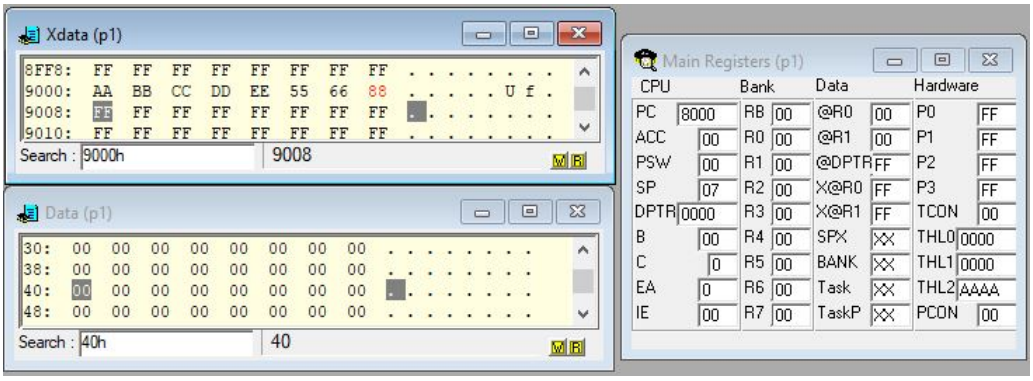


Program 1.c: TO TRANSFER A BLOCK OF 8 BYTES OF DATA FROM SOURCE 9000H EXTERNAL RAM TO DESTINATION 40H INTERNAL RAM.

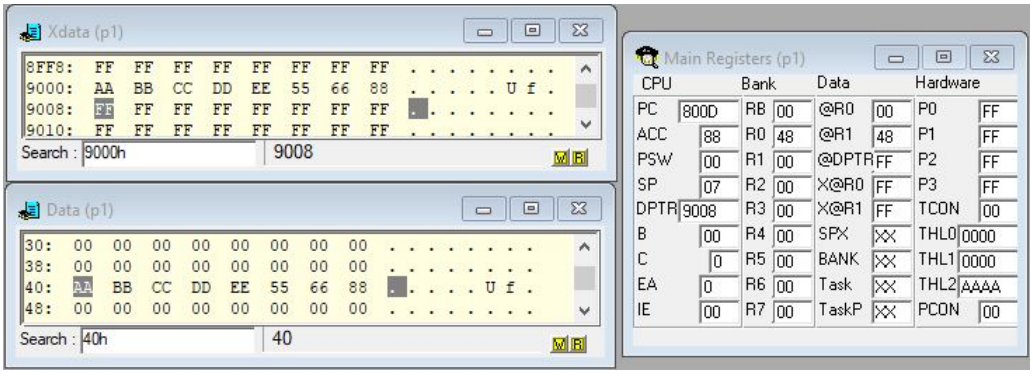
```
        ORG 8000H               ; start the program at 8000h
        MOV DPTR,# 9000H        ; initialize DPTR with address of source memory
        MOV R0,# 40H            ; initialize R0 with address of destination memory
        MOV R7,# 08H            ; initialize count value into R7
loop:   MOVX A,@DPTR            ; move the data into acc. from external memory
        MOV @R0,A               ; move the acc. contents into internal memory
        INC DPTR                ; increment DPTR to point next memory locn.
        INC R0                  ; increment R0 to point next memory location
        DJNZ R7 , loop          ; decrement R7 i.e., till count becomes zero
stop:   SJMP stop               ; stop incrementing Program Counter
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-

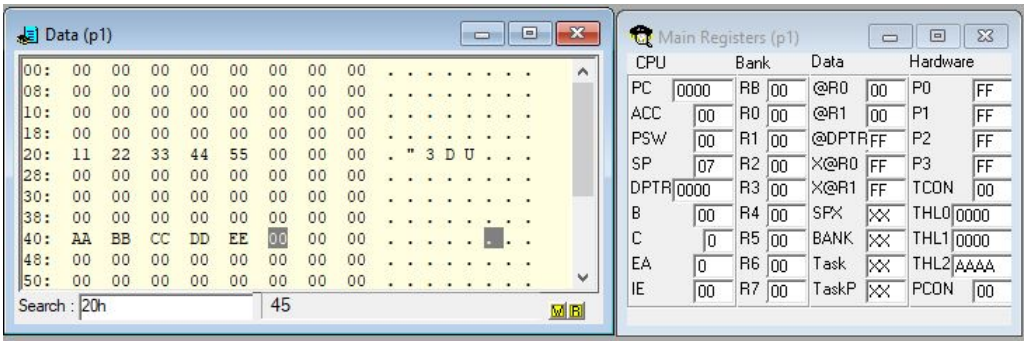


Program 1.d: TO EXCHANGE THE SOURCE BLOCK STARTING WITH ADDRESS 20H IN INTERNAL RAM CONTAINING 5 BYTES OF DATA WITH DESTINATION BLOCK STARTING WITH ADDRESS 40H IN INTERNAL RAM.

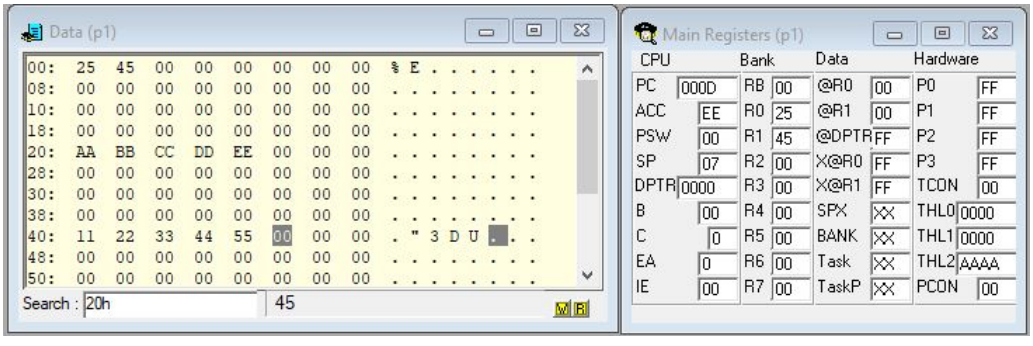
```
        ORG 0000H          ; start the program at 0000h
        MOV R0,# 20H       ; initialize R0 with address of source locn.
        MOV R1,# 40H       ; initialize R1 with address of destination locn.
        MOV R4,# 05H       ; initialize count value into R4
loop:   MOV A,@R0          ; move data from source locn. to Acc.
        XCH A,@R1          ; exchange content of Acc. and destination
        MOV@R0,A           ; move destination data present in Acc. to source
        INC R0             ; increment R0 to point next source locn.
        INC R1             ; increment R1 to point next destination locn.
        DJNZ R4, loop      ; decrement R4 i.e., till count becomes zero
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

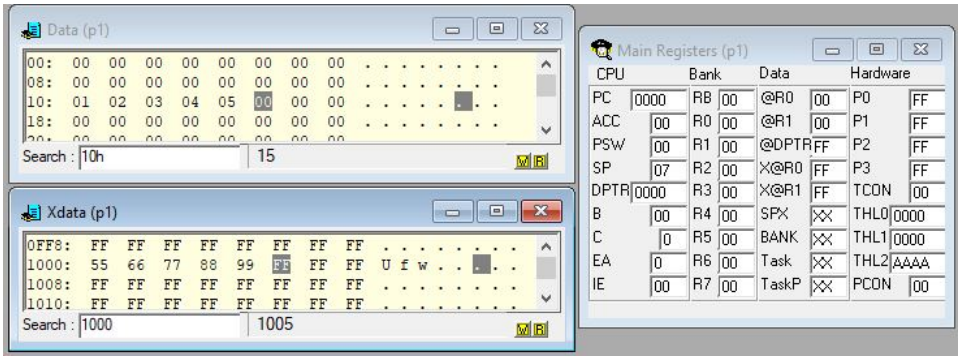


Program 1.e: TO EXCHANGE THE SOURCE BLOCK STARTING WITH ADDRESS 10H IN INTERNAL RAM CONTAINING 5 BYTES OF DATA WITH DESTINATION BLOCK STARTING WITH ADDRESS 2000H IN EXTERNAL RAM.

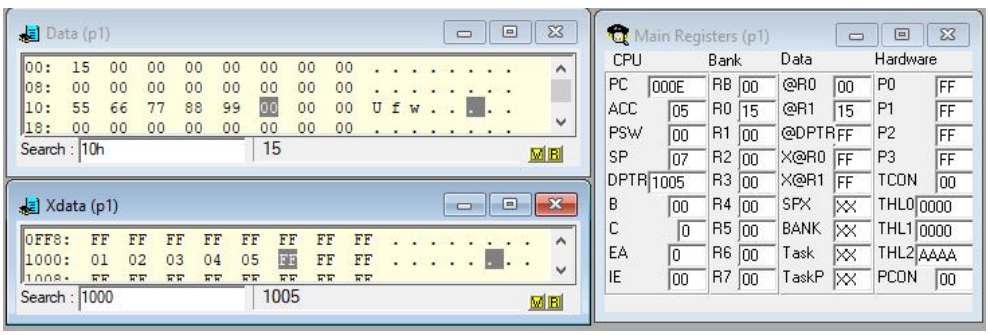
```
        ORG 0000H              ; start the program at 0000h
        MOV R0,# 10H           ; initialize R0 with address of source locn.
        MOV DPTR,# 1000H       ; initialize DPTR with address of destination locn.
        MOV R7,# 05H           ; initialize count value into R7
loop:   MOVX A,@DPTR           ; move data from destination to Acc.
        XCH A,@R0              ; exchange content of Acc. and source
        MOVX @DPTR,A           ; move source data present in Acc. to destination
        INC R0                 ; increment R0 to point to next source locn.
        INC DPTR               ; increment DPTR to point next destination locn.
        DJNZ R7, loop          ; decrement R7 i.e., till count becomes zero
stop:   SJMP stop              ; stop incrementing Program Counter
        END                    ; end of the program
```

Result:

Before Execution-



After Execution-

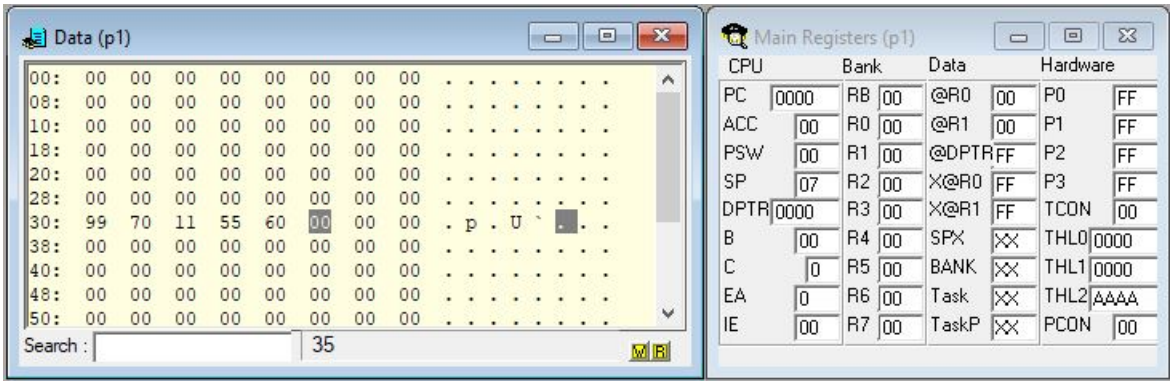


Program 1.f: TO ARRANGE 5 BYTES OF DATA IN ASCENDING ORDER STORED STARTING AT 30H IN INTERNAL RAM.

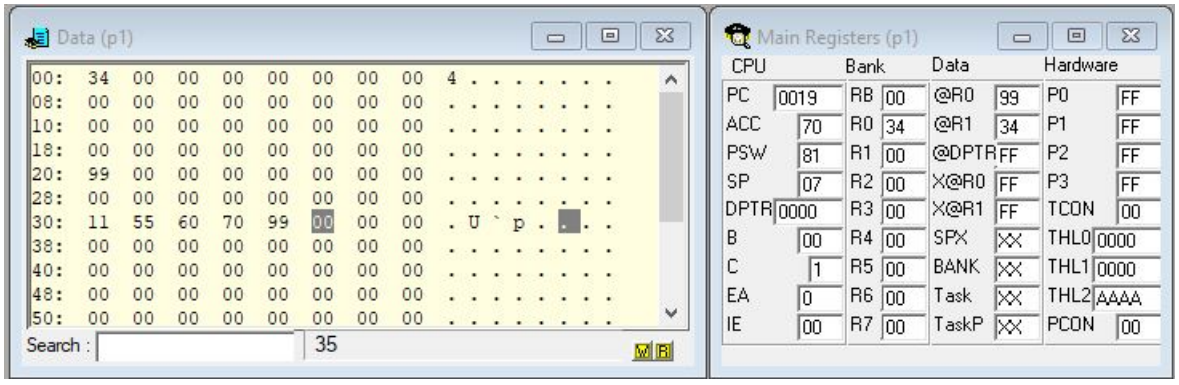
```
        ORG 000H                ; start the program at 000h
        MOV R2,# 04H            ; initialize external count value in R2
back:   MOV R0,# 30H            ; initialize R0 with starting address of array
        MOV R3,# 04H            ; initialize R3 with internal count value
loop:   MOV A,@R0               ; move data into Acc.
        INC R0                  ; increment R0 to point next location
        MOV 20H,@R0             ; move next data into temporary locn. 20H
        CJNE A, 20H, check      ; compare two data if not equal jump to label check
        SJMP next               ; if two data are equal jump to label next
check:  JC next                 ; if destination data < than source data jump to
                                  label next
        XCH A,@R0               ; else exchange first and second data
        DEC R0                  ; decrement R0 to point previous locn.
        MOV @R0,A               ; place smaller number
        INC R0                  ; increment R0 to point next locn.
next:   DJNZ R3, loop           ; repeat comparison with all data
        DJNZ R2, back           ; repeat for all data
stop:   SJMP stop               ; stop incrementing Program Counter
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-

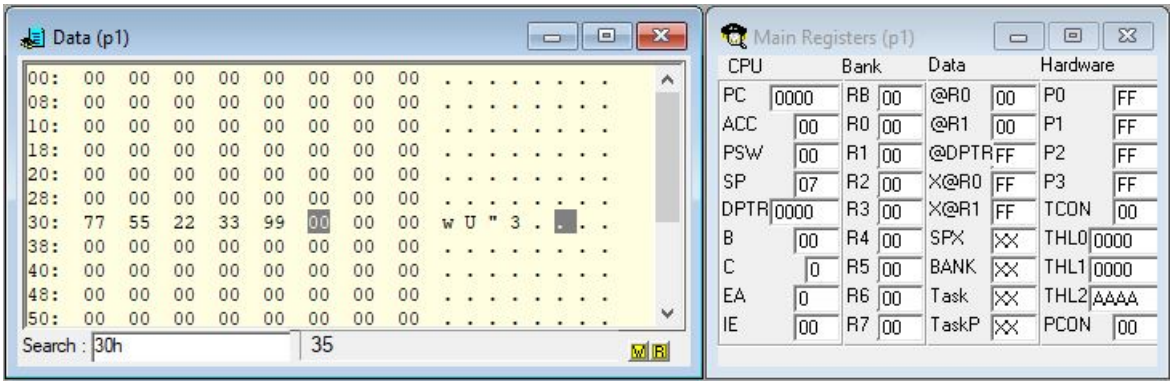


Program 1.g: TO ARRANGE 5 BYTES OF DATA IN DESCENDING ORDER STORED STARTING AT 30H IN INTERNAL RAM.

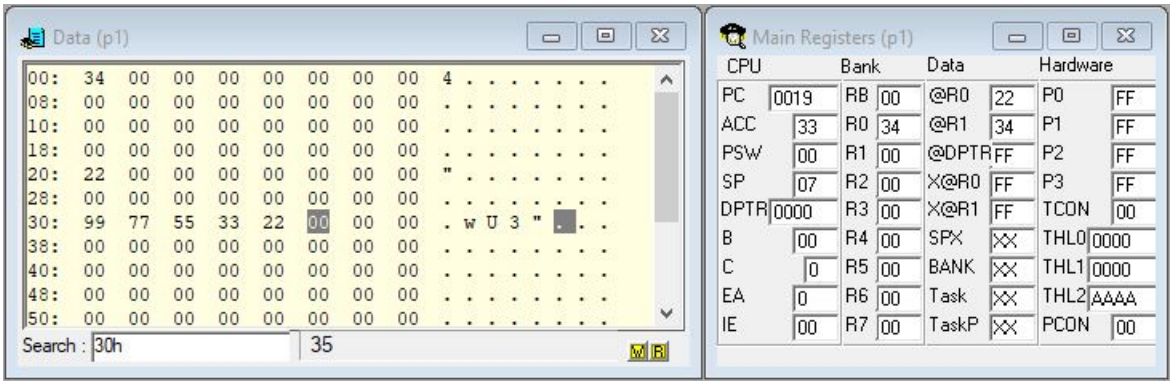
```
        ORG 000H                ; start the program at 000h
        MOV R2,# 04H            ; initialize external count value in R2
back:   MOV R0,# 30H            ; initialize R0 with starting address of array
        MOV R3,# 04H            ; initialize R3 with internal count value
loop:   MOV A,@R0               ; move data into Acc.
        INC R0                  ; increment R0 to point next location
        MOV 20H,@R0             ; move next data into temporary locn. 20H
        CJNE A, 20H, check      ; compare two data if not equal jump to label check
        SJMP next               ; if two data are equal jump to label next
check:  JNC next                ; if destination data > than source data jump to
                                  label next
        XCH A,@R0               ; else exchange first and second data
        DEC R0                  ; decrement R0 to point previous locn.
        MOV @R0,A               ; place largest number
        INC R0                  ; increment R0 to point next locn.
next:   DJNZ R3, loop           ; repeat comparison with all data
        DJNZ R2, back           ; repeat for all data
stop:   SJMP stop               ; stop incrementing Program Counter
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-

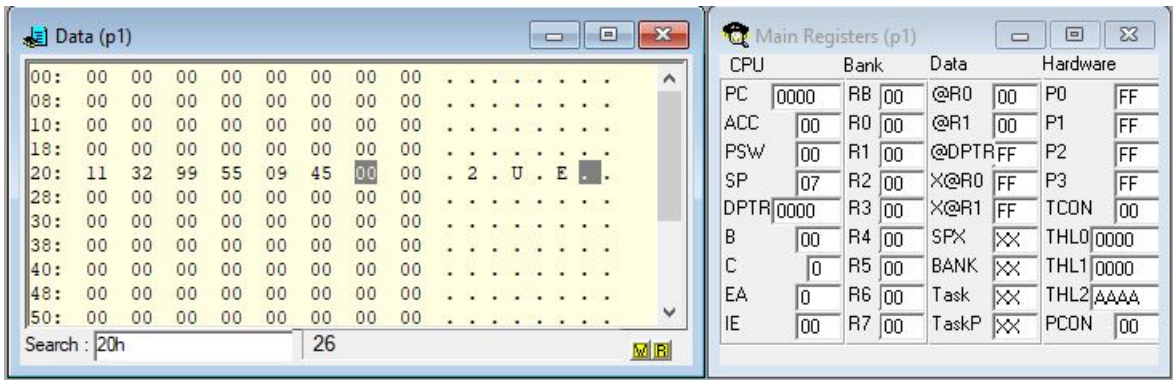


Program 1.h: TO FIND LARGEST NUMBER FROM AN ARRAY OF 6 BYTES STORED STARTING FROM 20H IN INTERNAL RAM AND STORE THE RESULT AT 40H IN INTERNAL RAM.

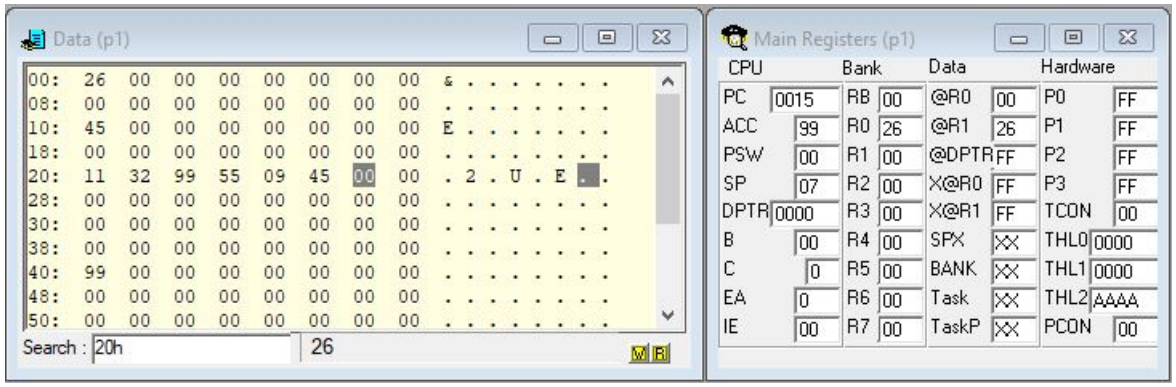
```
        ORG 0000H           ; start the program at 0000h
        MOV R0,# 20H        ; initialize R0 with address of source locn.
        MOV R2,# 05h        ; initialize count value into R2
        MOV A,@R0           ; move data from source array to Acc.
        INC R0              ; increment R0 to point next locn.
back:   MOV 10H,@R0         ; move next locn. data to temporary locn. 10h
        CJNE A, 10H, next   ; compare two data, if not equal jump to label next
        SJMP skip           ; else if equal jump to label skip
next:   JNC skip            ; if destination data > source data,jump to label
                            skip
        MOV A,@R0           ; hold the largest number in Acc.
skip:   INC R0              ; increment R0 to point next locn.
        DJNZ R2, back       ; decrement R2 i.e., till count becomes zero
        MOV 40H, A          ; store the largest number at 40h
stop:   SJMP stop           ; stop incrementing Program Counter
        bEND                ; end of the program
```

Result:

Before Execution-



After Execution-

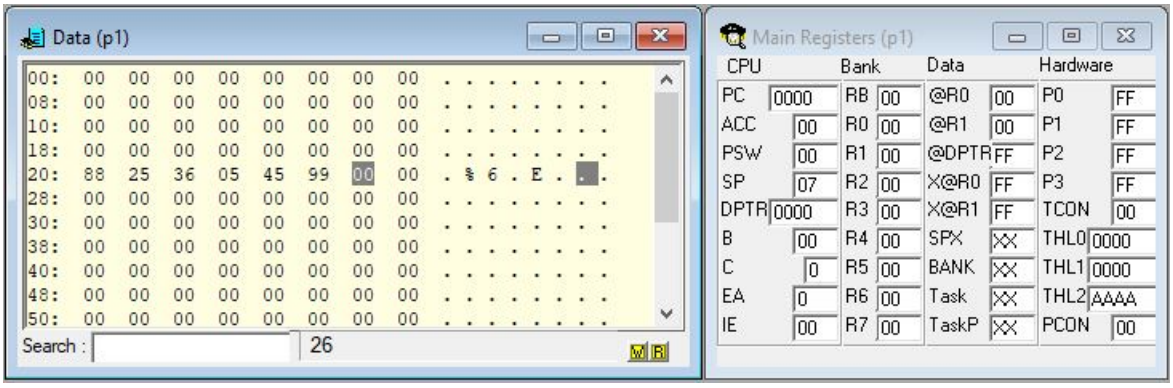


Program 1.i: TO FIND SMALLEST NUMBER FROM AN ARRAY OF 6 BYTES STORED STARTING FROM 20H IN INTERNAL RAM AND STORE THE RESULT AT 45H IN INTERNAL RAM.

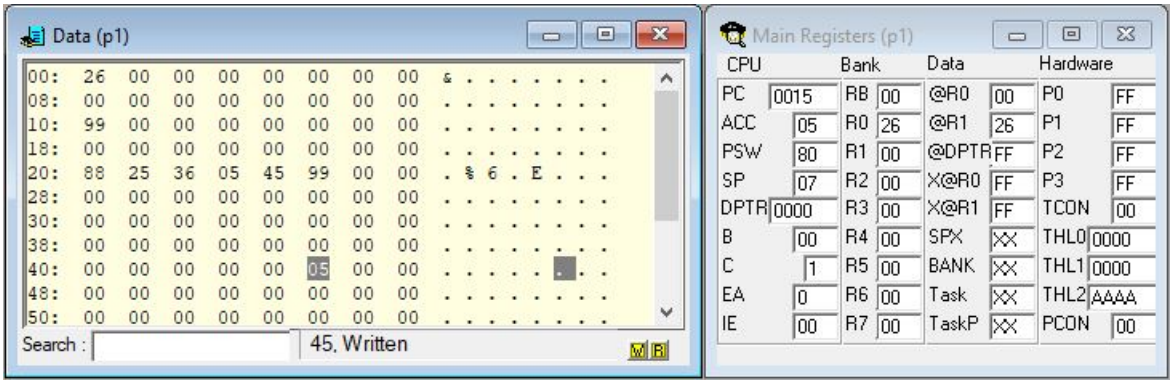
```
        ORG 0000H           ; start the program at 0000h
        MOV R0,# 20H        ; initialize R0 with address of source locn.
        MOV R2,# 05h        ; initialize count value into R2
        MOV A,@R0           ; move data from source array to Acc.
        INC R0              ; increment R0 to point next locn.
back:   MOV 10H,@R0         ; move next locn. data to temporary locn. 10h
        CJNE A, 10H, next   ; compare two data, if not equal jump to label next
        SJMP skip           ; else if equal jump to label skip
next:   JC skip             ; if destination data < source data,jump to label
                            skip
        MOV A,@R0           ; hold the smallest number in Acc.
skip:   INC R0              ; increment R0 to point next locn.
        DJNZ R2, back       ; decrement R2 i.e., till count becomes zero
        MOV 45H, A          ; store the smallest number at 45h
stop:   SJMP stop           ; stop incrementing Program Counter
        bEND                ; end of the program
```

Result:

Before Execution-



After Execution-



5 Experiment 2: Arithmetic Programs

Programs:

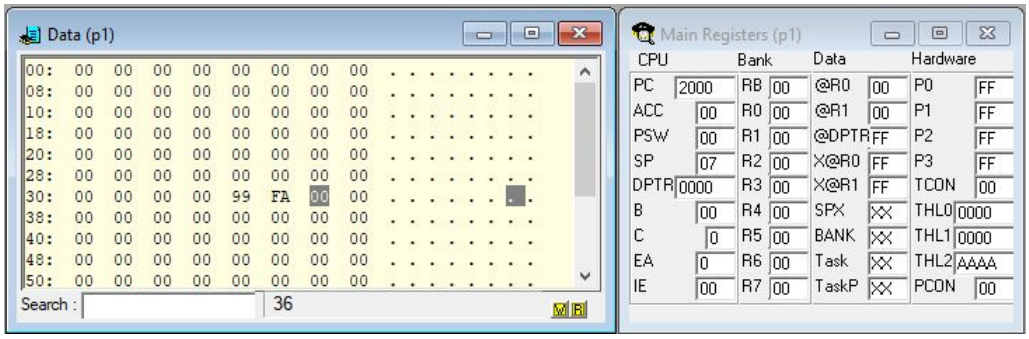
1. To add two 8-bits numbers using register addressing mode stored in 34h and 35h internal RAM, store the 16-bit result in register R5-LSB and R6-MSB.
2. To Add two 8-bits numbers using indirect addressing mode, stored in 34h and 35h internal RAM, store the 16-bit result in register R5-LSB and R6-MSB.
3. To Add two 8-bits number stored at 9050h and 9051h external RAM. Store the 16-bit result in 9052h-LSB and 9053h-MSB in external RAM.
4. To find the sum of 6 bytes stored in the internal RAM starting with address 40h. Store the 16-bits result at the end of the block.
5. To add two 16 bits numbers stored at 20h (LSB-N1), 21H (MSB-N1) and 22H (LSB-N2), 23H (MSB-N2) internal RAM. Store the result at 24H-LSB and 25H-MIDDLE BIT and 26H-MSB internal RAM, use direct addressing mode.
6. To subtract two 8-bits numbers using indirect addressing mode, stored in 34h and 35h internal RAM, store the 16-bit result in register R5-LSB and R6-MSB.
7. To subtract two 8-bits number stored at 9050h and 9051h external RAM. Store the 16-bit result in 9052h-LSB and 9053h-MSB in external RAM.
8. To find the subtraction of 6 bytes stored in the internal RAM starting with address 40h. Store the 16-bits result at the end of the block.
9. To subtract two 16 bits numbers stored at 20h (LSB-N1), 21H (MSB-N1) and 22H (LSB-N2), 23H (MSB-N2) internal RAM. Store the result at 24H-LSB and 25H-MIDDLE BIT and 26H-MSB internal RAM, use direct addressing mode.
10. To multiply two 8-bit numbers stored at 30h and 31h in internal RAM. store 16-bit result in 32h and 33h in internal RAM.
11. To multiply two 8-bit numbers stored at 8100h and 8101h external RAM. Store 16-bit result in 8200h and 8201h of external RAM.
12. To multiply 16-bit number with 8-bit number , first 16-bit number stored at 8100h-LSB, 8001h-MSB and then 8-bit number at 8002h external RAM. Store 16-bit result at 8200h onwards external RAM.
13. To perform division operation on 8-bit number by 8-bit number.
14. To find square of an 8-bit number.
15. To find cube of a number stored in 20h internal ram. Store at 50h-LSB, 51h MIDDLE bit and 52h-MSB in internal RAM.

Program 2.a: TO ADD TWO 8-BITS NUMBERS USING REGISTER ADDRESSING MODE STORED IN 34H AND 35H INTERNAL RAM, STORE THE 16-BIT RESULT IN REGISTER R5-LSB AND R6-MSB.

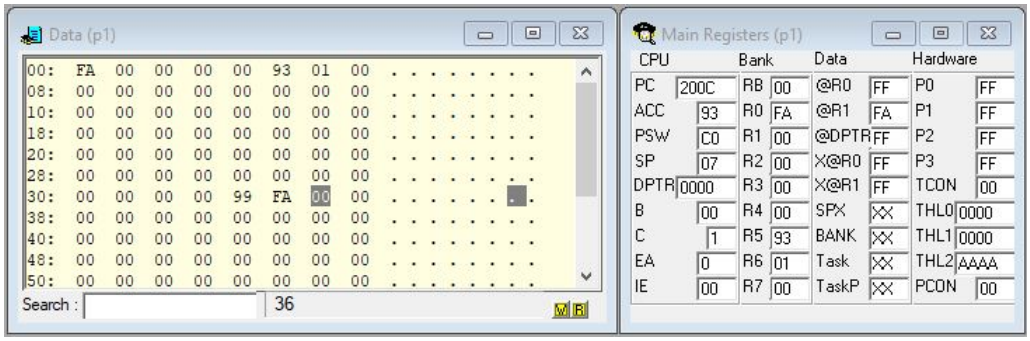
```
        ORG 2000H          ; start the program at 2000h
        MOV R6,# 00H       ; clear R6 register to hold MSB of result
        MOV A, 34H         ; move first byte in accumulator
        MOV R0, 35H        ; move second byte in R0 register
        ADD A, R0          ; add first byte and second byte
        JNC skip           ; check for MSB i.e. Carry else jump to label skip
        MOV R6,# 01H       ; store MSB of result i.e. Carry in R6 register
skip:   MOV R5,A           ; store LSB of result in R5 register
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

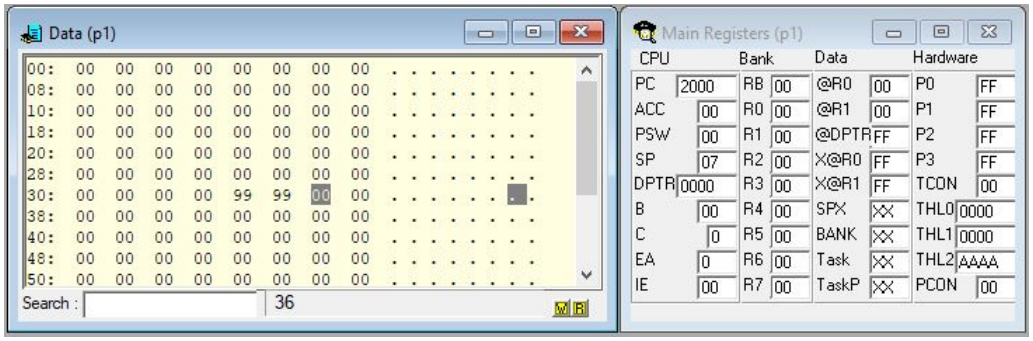


Program 2.b: TO ADD TWO 8-BITS NUMBERS USING INDIRECT ADDRESSING MODE, STORED IN 34H AND 35H INTERNAL RAM, STORE THE 16-BIT RESULT IN REGISTER R5-LSB AND R6-MSB.

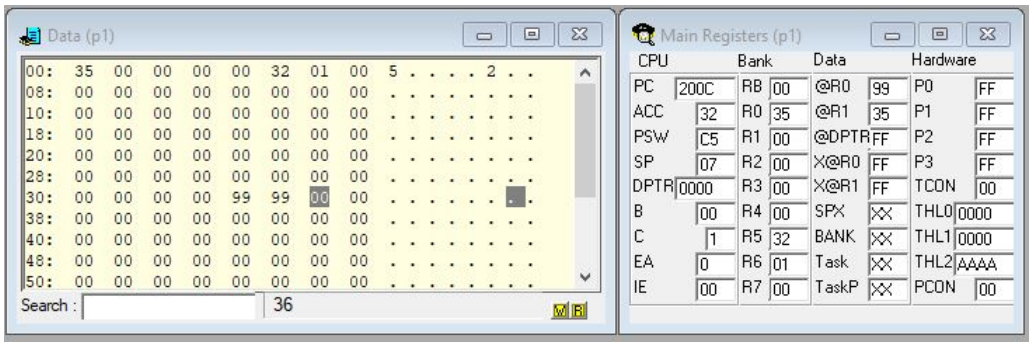
```
        ORG 2000H          ; start the program at 2000h
        MOV R6,# 00H       ; clear R6 register to hold MSB of result
        MOV R0,# 34H       ; load the address of first byte in R0
        MOV A,@R0          ; move first byte in accumulator
        INC R0             ; increment R0 to point next location
        ADD A,@R0          ; add first and second byte indirectly
        JNC skip           ; check for MSB i.e. Carry else jump to label skip
        MOV R6,# 01H       ; store MSB of result i.e. Carry in R6 register
skip:   MOV R5,A           ; store LSB of result in R5 register
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

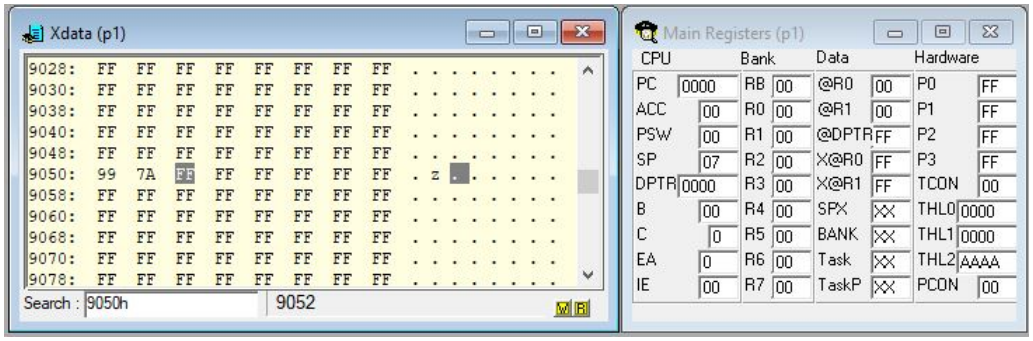


Program 2.c: TO ADD TWO 8-BITS NUMBER STORED AT 9050H and 9051H EXTERNAL RAM . STORE THE 16-BIT RESULT IN 9052H-LSB AND 9053H-MSB IN EXTERNAL RAM.

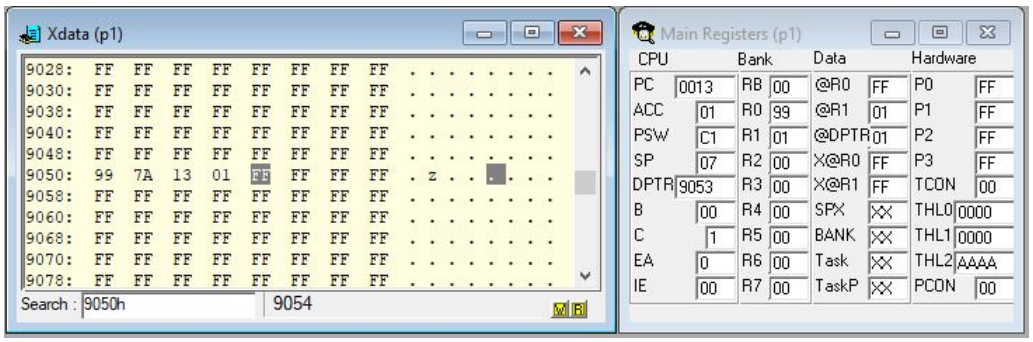
```
        ORG 0000H              ; start the program at 0000h
        MOV R1,# 00H           ; clear R1 register to hold MSB of result
        MOV DPTR,# 9050H       ; initialize DPTR with beginning address
        MOVX A,@DPTR           ; move first data into Acc. from external locn.
        MOV R0,A               ; store first data into R0
        INC DPTR               ; increment DPTR to point next data
        MOVX A,@DPTR           ; move second data into Acc.
        ADD A,R0               ; add second and first data
        JNC skip               ; check for MSB i.e.CY else jump to skip
        MOV R1,# 01H           ; store MSB of result i.e. Carry in R1 register
skip:   INC DPTR               ; increment DPTR to point next locn.
        MOVX @DPTR,A           ; save LSB of result at external locn.
        INC DPTR               ; increment DPTR to point next locn.
        MOV A, R1              ; move MSB to acc.
        MOVX @DPTR,A           ; save MSB of result at external locn.
stop:   SJMP stop              ; stop incrementing Program Counter
        END                    ; end of the program
```

Result:

Before Execution-



After Execution-

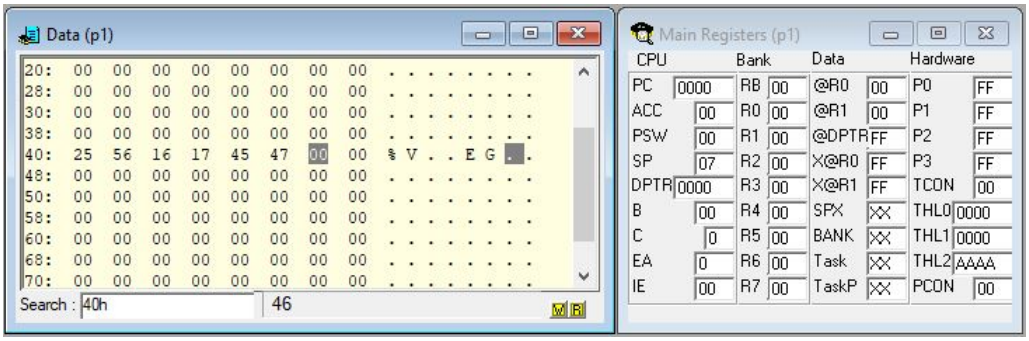


Program 2.d: TO FIND THE SUM OF 6 BYTES STORED IN THE INTERNAL RAM STARTING WITH ADDRESS 40H. STORE THE 16-BITS RESULT AT THE END OF THE BLOCK.

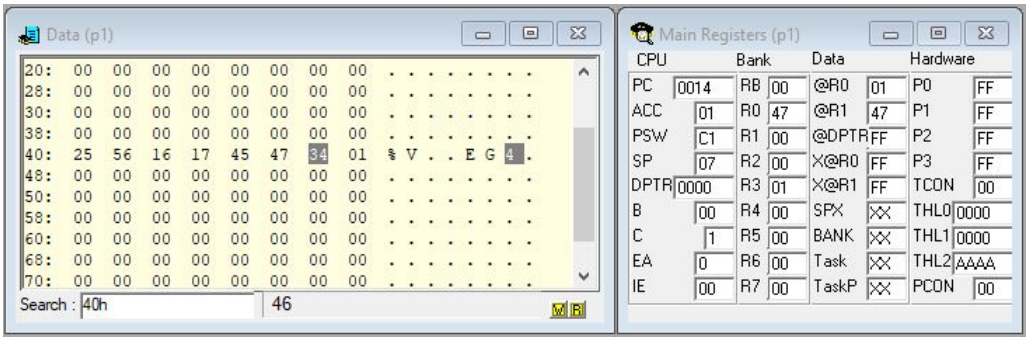
```
        ORG 000H          ; start the program at 0000h
        CLR C             ; clear carry flag
        MOV R0,# 40H      ; initialize source location in R0
        MOV R2,# 06H      ; initialize count value into R2
        MOV R3,# 00H      ; clear R3 to hold the MSB
        MOV A,# 00H       ; clear accumulator
loop:   ADD A,@R0         ; add Acc. and data pointed by R0
        JNC skip          ; check for MSB i.e. Carry else jump to label skip
        INC R3            ; store MSB of result i.e. Carry in R3 register
skip:   INC R0            ; increment R0 to point next data
        DJNZ R2, loop     ; decrement R2 i.e., till count becomes zero
        MOV @R0,A         ; store LSB of result at end of the block
        MOV A,R3          ; move MSB of result into Acc.
        INC R0            ; increment R0 to point next locn.
        MOV @R0,A         ; store MSB of result at end of the block
stop:   SJMP stop         ; stop incrementing Program Counter
        END               ; end of the program
```

Result:

Before Execution-



After Execution-

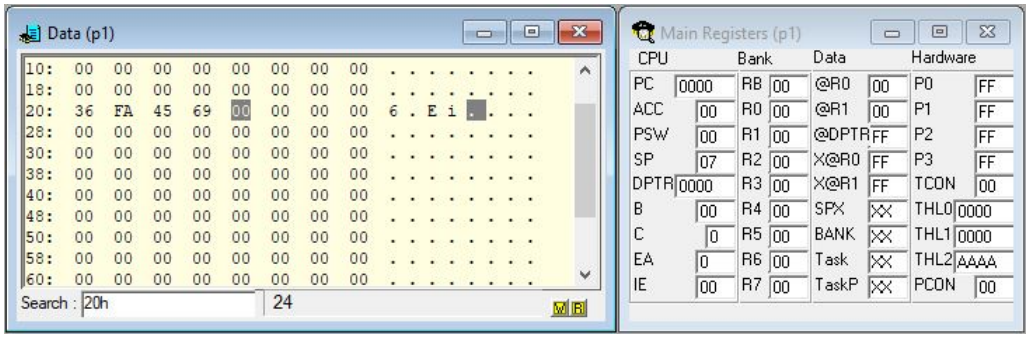


Program 2.e: TO ADD TWO 16 BITS NUMBERS STORED AT 20H (LSB-N1), 21H (MSB-N1) AND 22H (LSB-N2), 23H (MSB-N2) INTERNAL RAM. STORE THE RESULT AT 24H-LSB AND 25H- MIDDLE BIT AND 26H-MSB INTERNAL RAM, USE DIRECT ADDRESSING MODE.

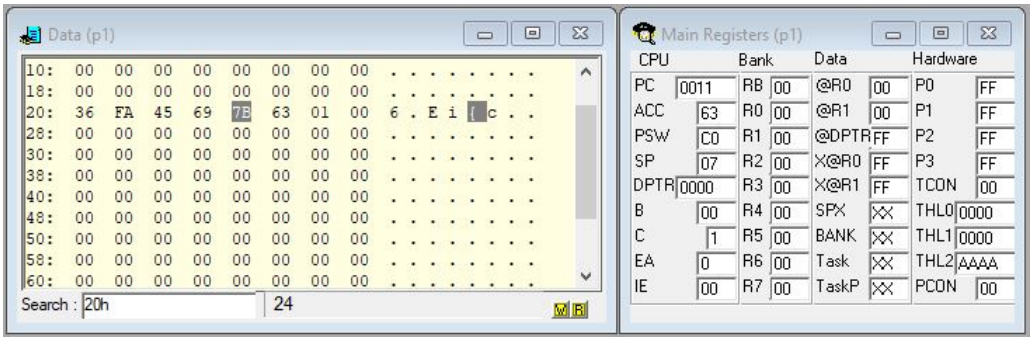
```
        ORG 0000H          ; start the program at 0000h
        MOV A, 20H         ; move LSB-N1 into Acc.
        ADD A, 22H         ; Add LSB-N1 with LSB-N2
        MOV 24H, A         ; store LSB of result in 24h
        MOV A, 21H         ; move MSB-N1 into Acc.
        ADDC A, 23H        ; Add to MSB-N2
        JNC skip           ; check for MSB i.e. Carry else jump to label skip
        MOV 26H,# 01H      ; store MSB of result at end of the block
skip:   MOV 25H,A          ; store MIDDLE byte of result in 25h
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

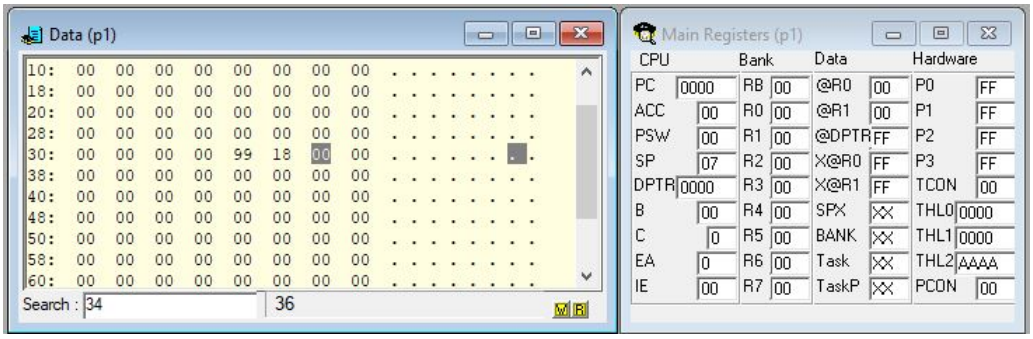


Program 2.f: TO SUBTRACT TWO 8-BITS NUMBERS USING INDIRECT ADDRESSING MODE, STORED IN 34H AND 35H INTERNAL RAM, STORE THE 16-BIT RESULT IN REGISTER R5-LSB AND R6-MSB.

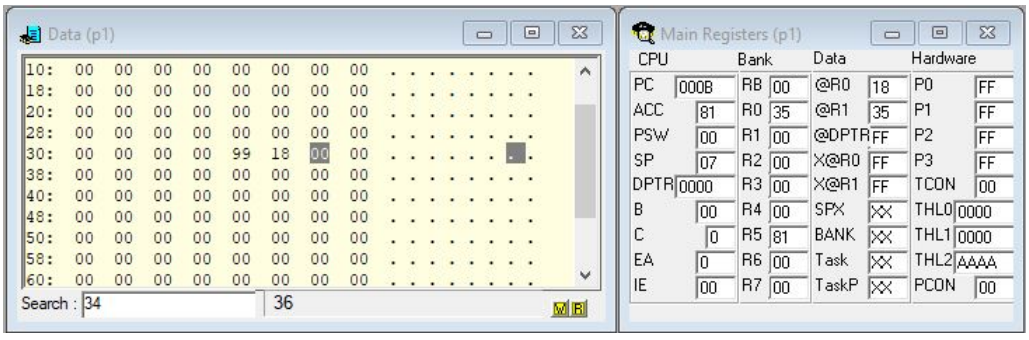
```
        ORG 0000H          ; start the program at 0000h
        MOV R6,# 00H       ; clear R6 register to hold MSB of result
        MOV R0,# 34H       ; load the address of first byte in R0
        MOV A,@R0          ; move first byte in accumulator
        INC R0             ; increment R0 to point next location
        SUBB A,@R0         ;subtract first and second byte indirectly
        JNC skip           ; check for MSB i.e. Carry else jump to label skip
        DEC R6             ; if borrow generated store in R6
skip:   MOV R5,A           ; store LSB of result in R5 register
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

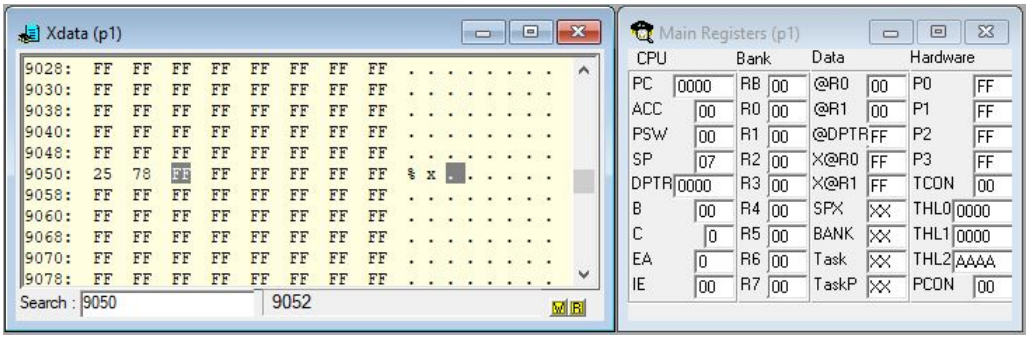


Program 2.g: TO SUBTRACT TWO 8-BITS NUMBER STORED AT IN 9050H AND 9051H EXTERNAL RAM . STORE THE 16-BIT RESULT IN 9052H-LSB AND 9053H-MSB IN EXTERNAL RAM.

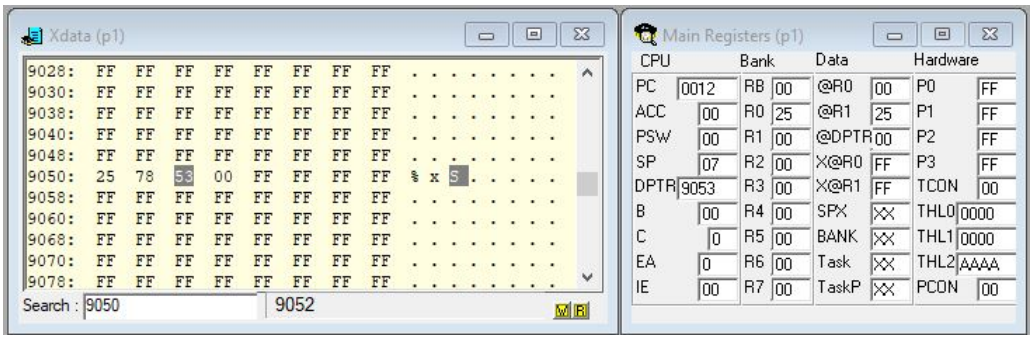
```
        ORG 0000H           ; start the program at 0000h
        MOV R1,# 00H        ; clear R1 register to hold MSB of result
        MOV DPTR,# 9050H    ; initialize DPTR with beginning address
        MOVX A,@DPTR        ; move first data into acc. from external locn.
        MOV R0,A            ; store first data into R0
        INC DPTR            ; increment DPTR to point next data
        MOVX A,@DPTR        ; move second data into Acc.
        SUBB A, R0          ; subtract second and first data
        JNC skip            ; check for MSB i.e. borrow else jump to skip
        DEC R1              ; store MSB of result i.e. borrow in R1 register
skip:   INC DPTR            ; increment DPTR to point next locn.
        MOVX @DPTR,A        ; save LSB of result at external locn.
        INC DPTR            ; increment DPTR to point next locn.
        MOV A, R1           ; move MSB to Acc.
        MOVX @DPTR,A        ; save MSB of result at external locn.
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:

Before Execution-



After Execution-

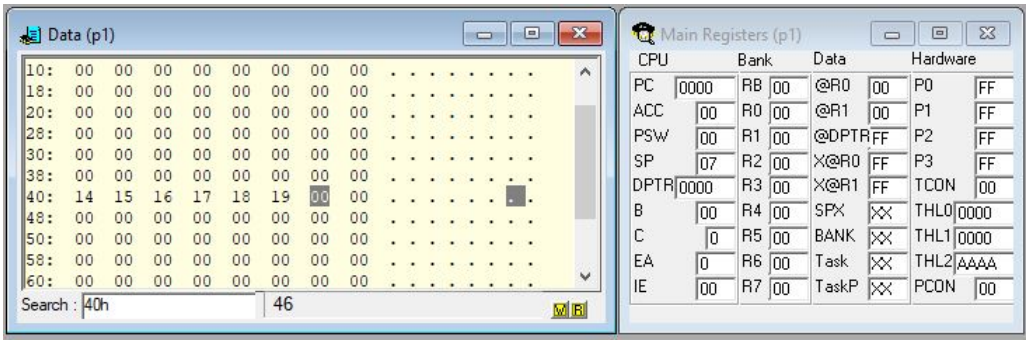


Program 2.h: TO FIND THE SUBTRACTION OF 6 BYTES STORED IN THE INTERNAL RAM STARTING WITH ADDRESS 40H. STORE THE 16-BITS RESULT AT THE END OF THE BLOCK

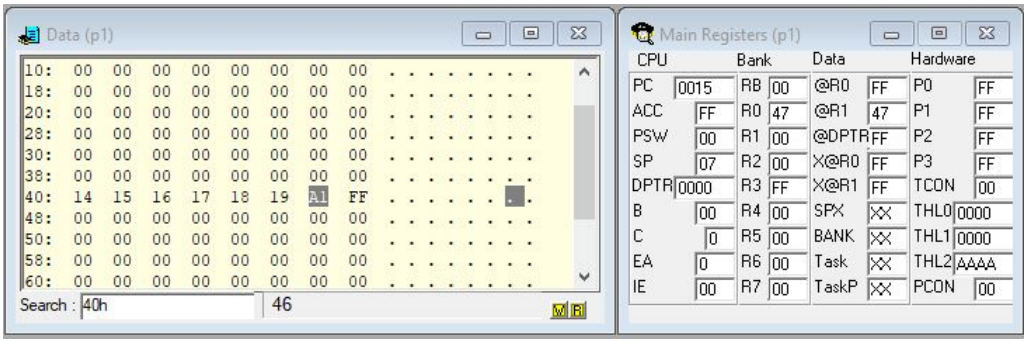
```
        ORG 0000H          ; start the program at 0000h
        CLR C              ; clear carry flag
        MOV R0,# 40H       ; initialize source location in R0
        MOV R2,# 05H       ; initialize count value into R2
        MOV R3,# 00H       ; clear R3 to hold the MSB
        MOV A,@R0          ; move first byte in accumulator
        INC R0             ; increment R0 to point next data
loop:   CLR C              ; make sub out of subb
        SUBB A,@R0         ; subtract Acc. and data pointed by R0
        JNC skip           ; check for MSB i.e. borrow else jump to skip
        DEC R3             ; store MSB of result i.e. borrow in R3 register
skip:   INC R0             ; increment R0 to point next data
        DJNZ R2, loop      ; decrement R2 i.e., till count becomes zero
        MOV @R0,A          ; store LSB of result at end of the block
        MOV A,R3           ; move MSB of result into Acc.
        INC R0             ; increment R0 to point next locn.
        MOV @R0,A          ; store MSB of result at end of the block
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

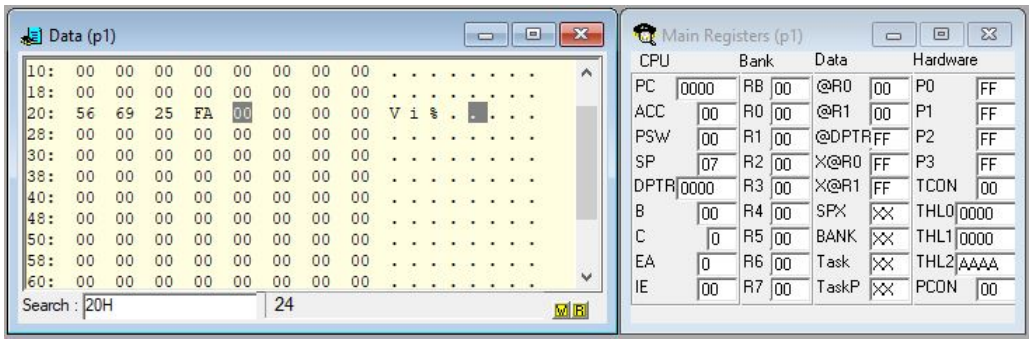


Program 2.i: TO SUBTRACT TWO 16 BITS NUMBERS STORED AT 20H (LSB-N1), 21H (MSB-N1) AND 22H (LSB-N2), 23H (MSB-N2) INTERNAL RAM. STORE THE RESULT AT 24H-LSB AND 25H- MIDDLE BIT AND 26H-MSB INTERNAL RAM, USE DIRECT ADDRESSING MODE.

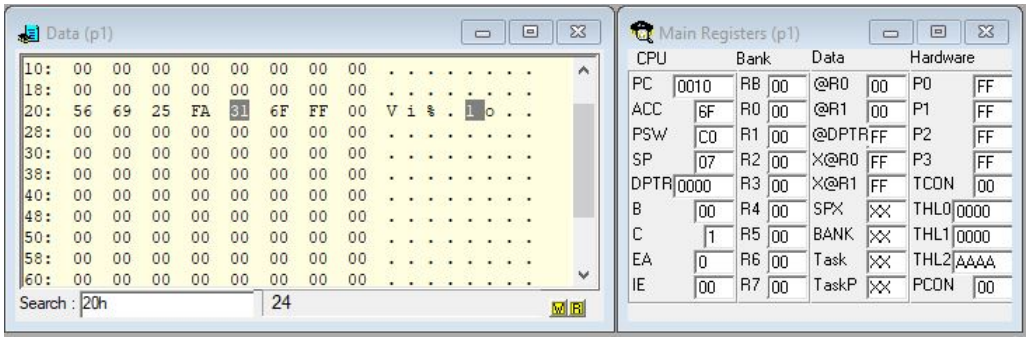
```
        ORG 0000H       ; start the program at 0000h
        MOV A, 20H      ; move LSB-N1 into acc.
        SUBB A ,22H     ; subtract LSB-N1 with LSB-N2
        MOV 24H, A      ; store LSB of result in 24h
        MOV A, 21H      ; move MSB-N1 into acc.
        SUBB A,23H      ; subtract with MSB-N2
        JNC skip        ; check for MSB i.e.borrow else jump to label skip
        DEC 26H         ; store MSB of result at end of the block
skip:   MOV 25H,A       ; store MIDDLE byte of result in 25h
stop:   SJMP stop       ; stop incrementing Program Counter
        END             ; end of the program
```

Result:

Before Execution-



After Execution-

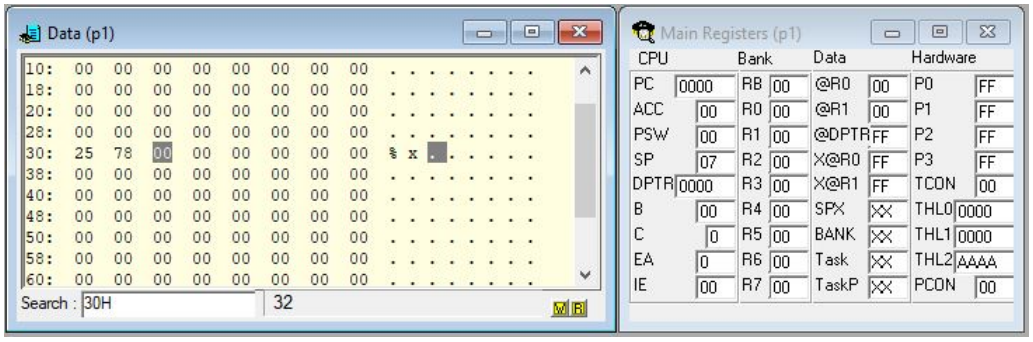


Program 2.j: TO MULTIPLY TWO 8-BIT NUMBERS STORED AT 30H AND 31H IN INTERNAL RAM. STORE 16-BIT RESULT IN 32H AND 33H IN INTERNAL RAM.

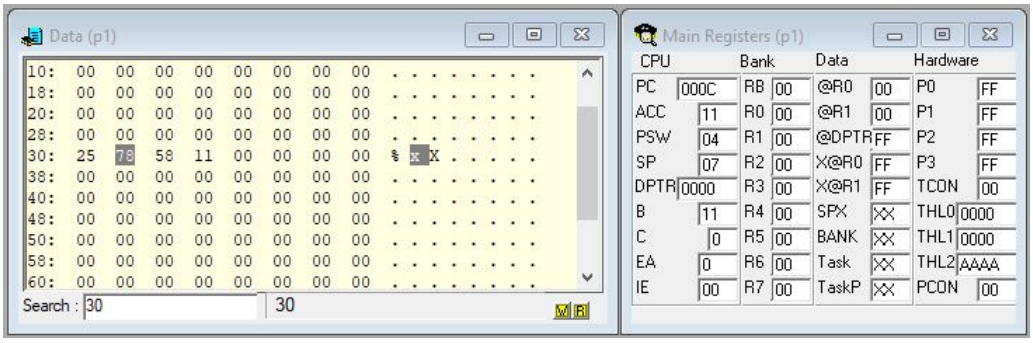
```
        ORG 0000H           ; start the program at 0000h
        MOV A, 30H          ; move the multiplier to Acc.
        MOV 0F0H, 31H       ; move the multiplicand to Reg. B
        MUL AB              ; multiply two numbers
        MOV 32H, A          ; store LSB of product in 32h
        MOV A, 0F0H         ; move MSB of product in Acc.
        MOV 33H,A           ; move MSB of product from Acc. to 33h
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:

Before Execution-



After Execution-

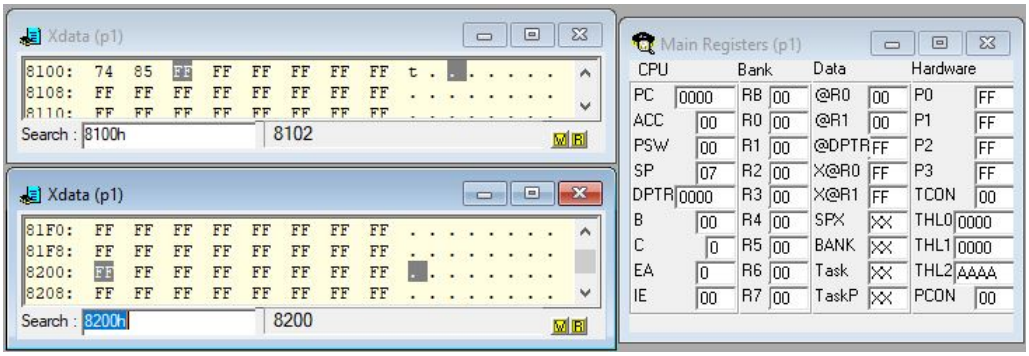


Program 2.k: TO MULTIPLY TWO 8-BIT NUMBERS STORED AT 8100H AND 8101H EXTERNAL RAM. STORE 16-BIT RESULT IN 8200H AND 8201H OF EXTERNAL RAM.

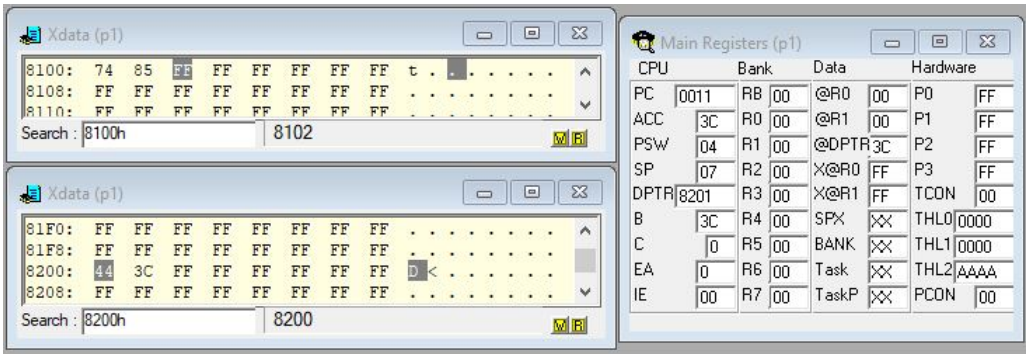
```
        ORG 000H              ; start the program at 000h
        MOV DPTR,# 8100H      ; initialize DPTR with multiplicand address
        MOVX A,@DPTR          ; move multiplicand in acc.
        MOV 0F0H,A            ; move multiplicand in register B
        INC DPTR              ; increment DPTR to point multiplier address
        MOVX A,@DPTR          ; move multiplier in Acc.
        MUL AB                ; multiply two numbers
        MOV DPTR,# 8200H      ; initialize DPTR with destination address
        MOVX @DPTR, A         ; store LSB of product in 8200h
        MOV A,0F0H            ; move MSB of product in Acc.
        INC DPTR              ; increment DPTR to point next locn.
        MOVX @DPTR,A          ; move MSB of product in 8201h
stop:   SJMP stop             ; stop incrementing Program Counter
        END                   ; end of the program
```

Result:

Before Execution-



After Execution-

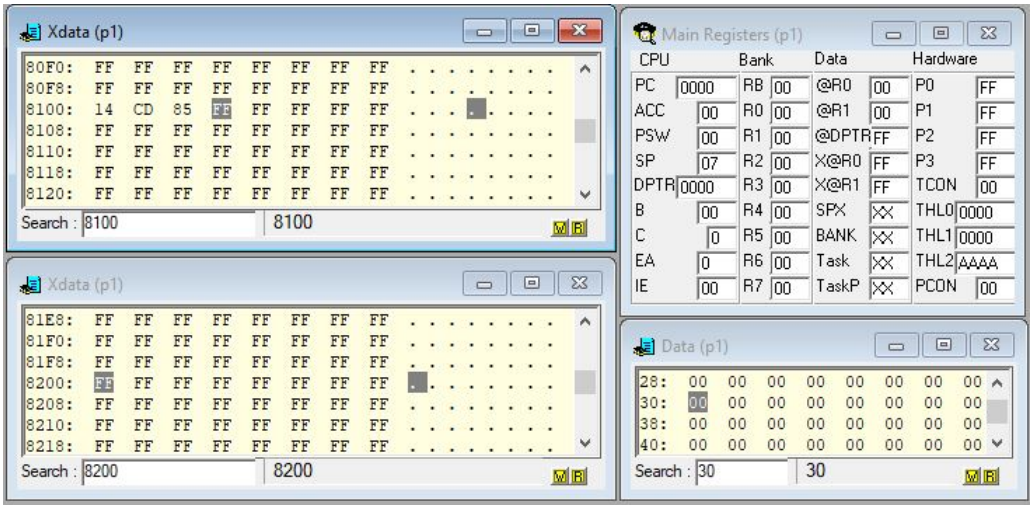


Program 2.l: TO MULTIPLY 16-BIT NUMBER WITH 8-BIT NUMBER , FIRST 16-BIT NUMBER STORED AT 8100H-LSB, 8001H-MSB AND THEN 8-BIT NUMBER AT 8002H EXTERNAL RAM. STORE 16-BIT RESULT AT 8200H ONWARDS EXTERNAL RAM.

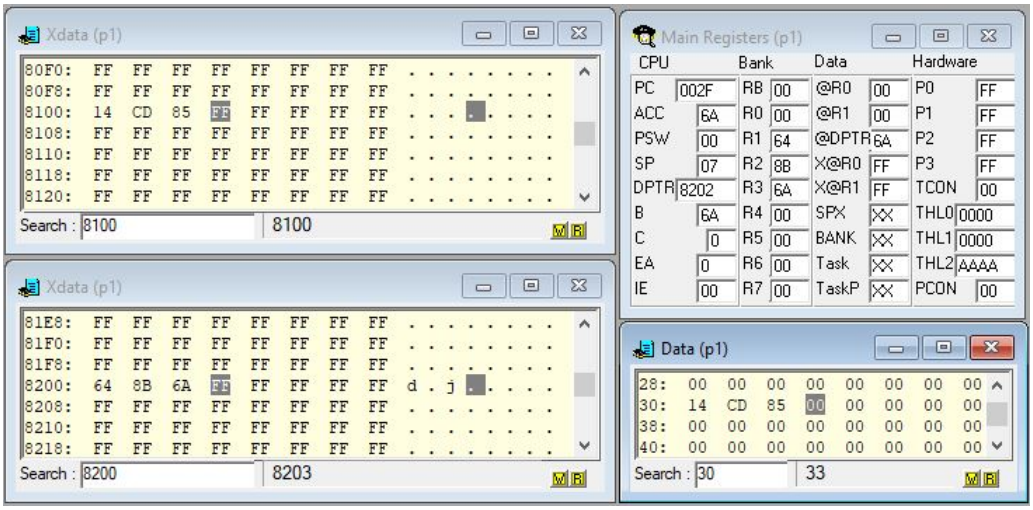
```
ORG 000H              ; start the program at 000h
MOV DPTR,# 8100H      ; initialize the source locn. in DPTR
MOVX A,@DPTR          ; move LSB of 16-bit Number into Acc.
MOV 30H,A             ; save in internal locn. 30h
INC DPTR              ; point to next locn.
MOVX A,@DPTR          ; move MSB of 16-bit Number into Acc.
MOV 31H,A             ; save in internal locn. 31h
INC DPTR              ; point to next locn.
MOVX A,@DPTR          ; move 8-bit Number into Acc.
MOV 32H,A             ; save in internal locn.
MOV A,30H             ; move LSB of 16-bit Number into Acc.
MOV 0F0H,32H          ; move 8-bit Number into Acc.
MUL AB                ; multiply, 16-bit result in A(LSB) and B(MSB)
MOV R1,A              ; save LSB of partial result in R1
MOV R2,0F0H           ; save MSB of result in R2
MOV A,31H             ; move MSB of 16-bit Number into Acc.
MOV 0F0H,32H          ; move 8-bit Number into Acc.
MUL AB                ; multiply, 16-bit result in A(LSB) and B(MSB)
ADD A,R2              ; add MSB of partial result
MOV R2,A              ; save back the result in R2
MOV A,0F0H            ; move MSB of Partial result into Acc.
ADDC A,# 00H          ; add with carry MSB
MOV R3,A              ; save the MSB of result in R3
MOV DPTR,# 8200H      ; initialize the destination locn.
MOV A,R1              ; move LSB of result into Acc.
MOVX @DPTR,A          ; save LSB of result into 8200h locn.
INC DPTR              ; increment DPTR to point next location
MOV A,R2              ; move MIDDLE bit of the result to Acc.
MOVX @DPTR,A          ; save MIDDLE bit of the result 8201h locn.
```

```
        INC DPTR            ; increment DPTR to point next location
        MOV A,R3            ; move MSB bit of the result to Acc.
        MOVX @ DPTR,A       ; move MSB bit of the result to 8202h locn.
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:
Before Execution-



After Execution-

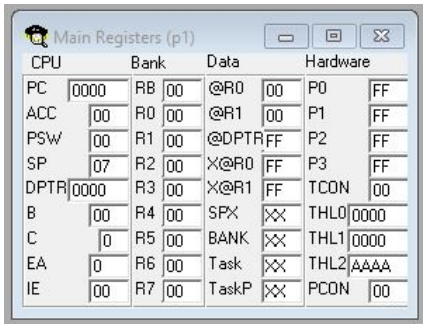


Program 2.m: TO PERFORM DIVISION OPERATION ON 8-BIT NUMBER BY 8-BIT NUMBER.

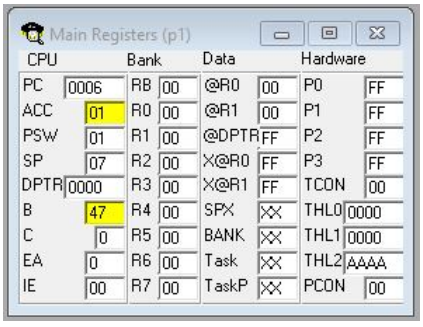
```
        ORG 000H            ; start the program at 000h
        MOV A,# 99H         ; move dividend in Acc.
        MOV 0F0H,# 52H      ; move divisor in B register
        DIV AB              ; divide A by B, Quotient stored in A, Reminder
                              stored in B
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:

Before Execution-



After Execution-

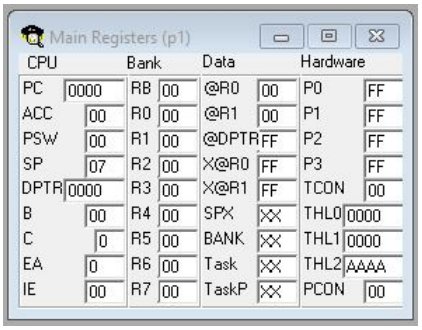


Program 2.n: TO FIND SQUARE OF AN 8-BIT NUMBER.

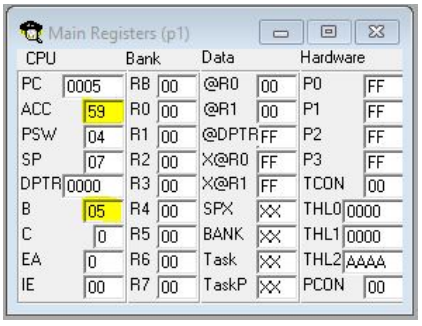
```
        ORG 000H        ; start the program at 000h
        MOV A,# 25H     ; take a number into A reg.
        MOV 0F0H,A      ; its copy into B register
        MUL AB          ; multiply both and square is in Acc. and in B
stop:   SJMP stop       ; stop incrementing Program Counter
        END             ; end of the program
```

Result:

Before Execution-



After Execution-

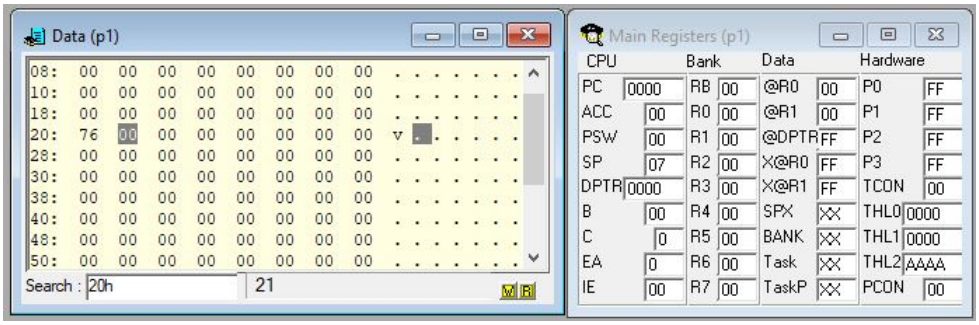


Program 2.o: TO FIND CUBE OF A NUMBER STORED IN 20H INTERNAL RAM. STORE AT 50H-LSB, 51H MIDDLE BIT AND 52H-MSB IN INTERNAL RAM.

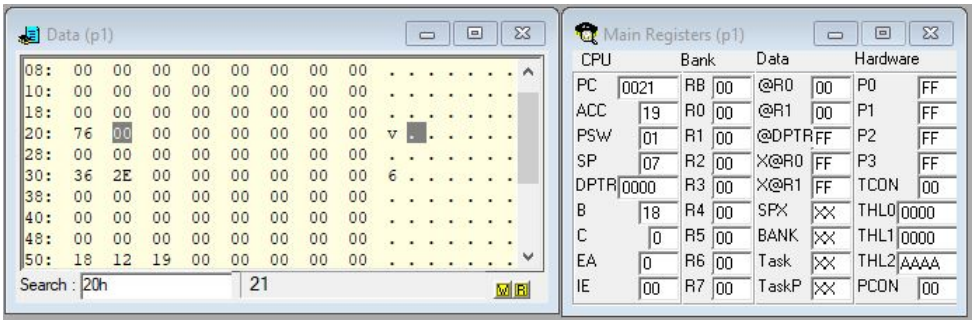
```
        ORG 000H          ; start the program at 000h
        MOV A,20H         ; copy a number into Acc.
        MOV 0F0H,A        ; take its copy into B
        MUL AB            ; multiply A and B
        MOV 30H,0F0H      ; save MSB of partial result in 30h
        MOV 0F0H,20h      ; take a same number in B
        MUL AB            ; multiply A and B
        MOV 50H,A         ; save LSB of result in 50h
        MOV 31H,0F0H      ; save MSB of partial result in 31h
        MOV A,30H         ; take back MSB of partial result in Acc.
        MOV 0F0H,20h      ; take a same number in B
        MUL AB            ; multiply A and B
        ADD A,31H         ; form the MIDDLE bit
        MOV 51H,A         ; save MIDDLE bit of result in 51h
        MOV A,0F0h        ; move MSB bit of result in Acc.
        ADDC A,# 00H      ; add carry bit with MSB
        MOV 52H,A         ; save MSB of result in 52h
stop:   SJMP stop         ; stop incrementing Program Counter
        END               ; end of the program
```

Result:

Before Execution-



After Execution-



6 Experiment 3: Code Conversion Programs

Programs:

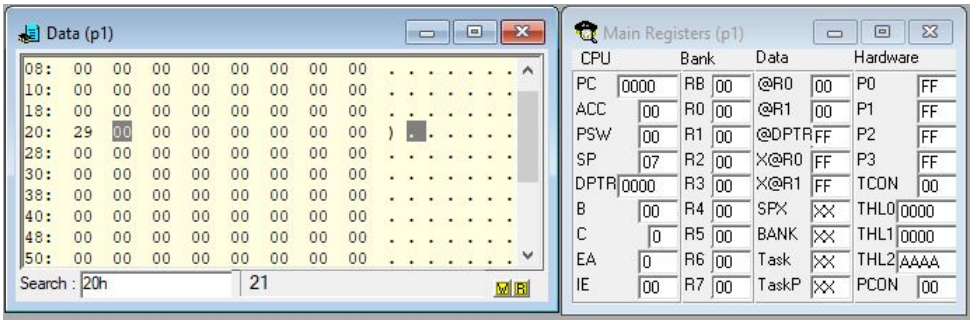
1. To convert packed BCD to ASCII convertion, stored at 20h in internal RAM. Store result at 50h and 51h in internal RAM.
2. To convert two ASCII numbers to packed BCD, stored at 20 and 21h in internal RAM. Store result at 50h in internal RAM.
3. To convert HEXADECIMAL to ASCII, stored at 20h in internal RAM. Store result at 50h in internal RAM.
4. To convert ASCII number to HEXADECIMAL, stored at 20h in internal RAM. Store result at 50h in internal RAM.
5. To convert DECIMAL to HEXADECIMAL, stored at 20h in internal RAM. Store result at 21h in internal RAM.
6. To convert HEXADECIMAL to DECIMAL, stored at 20h in internal RAM. Store result at 21h onwards in internal RAM.

Program 3.a: TO CONVERT PACKED BCD TO ASCII CONVERTION, STORED AT 20H IN INTERNAL RAM. STORE RESULT AT 50H AND 51H IN INTERNAL RAM.

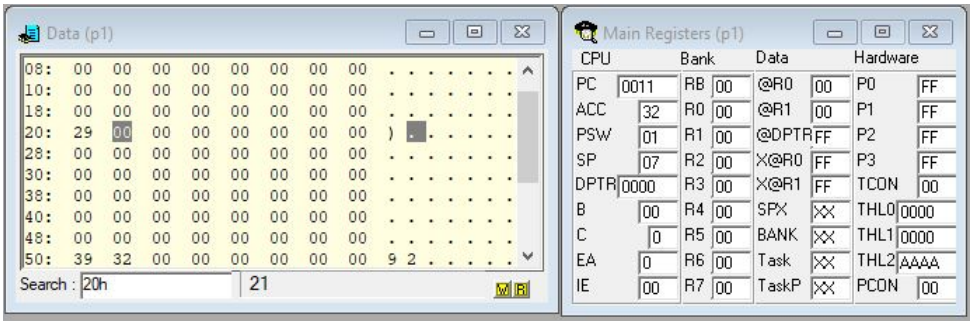
```
        ORG 0000H          ; start the program at 0000h
        MOV A,20H          ; load the BCD into Acc.
        ANL A,# 0FH        ; mask the lower byte
        ADD A,# 30H        ; convert lower byte to ASCII
        MOV 50H,A          ; save the result in 50h
        MOV A,20H          ; load the BCD into Acc.
        ANL A,# 0F0H       ; mask the higher byte
        SWAP A             ; swap the content of Acc. to make it lower byte
        ADD A,# 30H        ; convert higher byte to ASCII
        MOV 51H,A          ; save the result in 51h
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

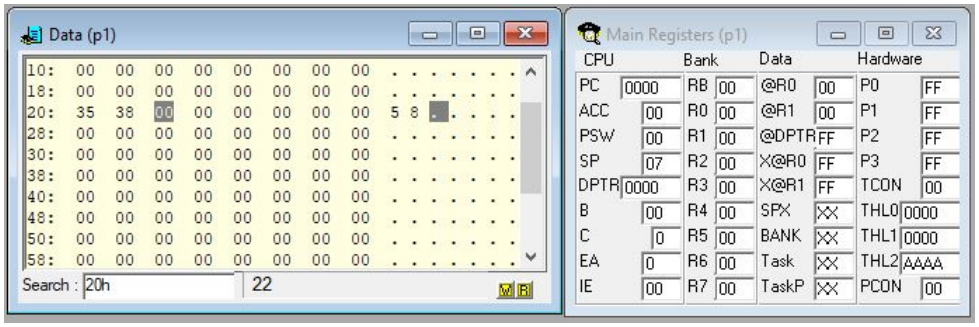


Program 3.b: TO CONVERT TWO ASCII NUMBERS TO PACKED BCD, STORED AT 20 AND 21H IN INTERNAL RAM. STORE RESULT AT 50H IN INTERNAL RAM.

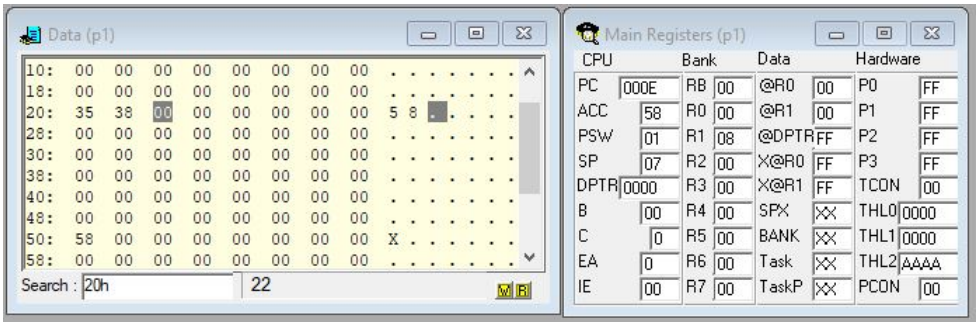
```
        ORG 0000H          ; start the program at 0000h
        MOV A,21H          ; take second ASCII number into Acc.
        SUBB A ,# 30H      ; convert to BCD
        MOV R1,A           ; save into R1
        MOV A,20H          ; take first ASCII number into Acc.
        CLR C              ; clear CY flag
        SUBB A,# 30H       ; convert to BCD
        SWAP A             ; convert to higher byte by swapping
        ADD A,R1           ; pack the BCD result
        MOV 50H,A          ; save the result into 50h
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

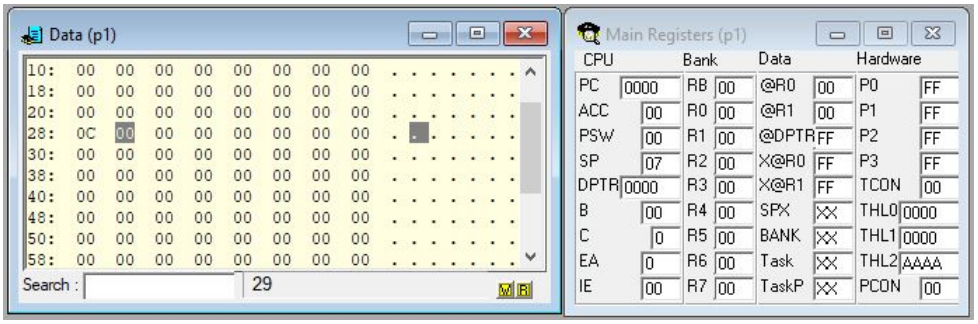


Program 3.c: TO CONVERT HEXADECIMAL TO ASCII, STORED AT 20H IN INTERNAL RAM. STORE RESULT AT 50H IN INTERNAL RAM.

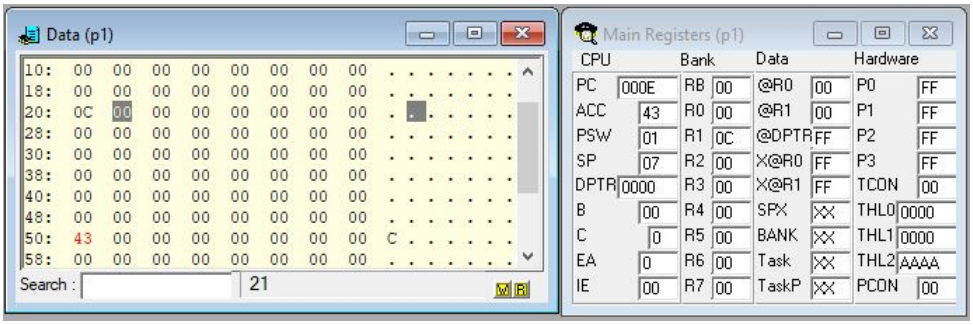
```
        ORG 0000H          ; start the program at 0000h
        MOV A,20H          ; take hex number into Acc.
        MOV R1,A           ; save in R1 register
        SUBB A,# 0AH       ; compare hex number with 10
        MOV A,R1           ; get back the hex number into Acc.
        JC skip            ; if the hex number is > 10, jump to label skip
        ADD A,# 07H        ; else add 7 and also add 30
skip:   ADD A,# 30H        ; convert to ASCII
        MOV 50H,A          ; save the result into 50h
stop:   SJMP stop          ; stop incrementing Program Counter
        END                ; end of the program
```

Result:

Before Execution-



After Execution-

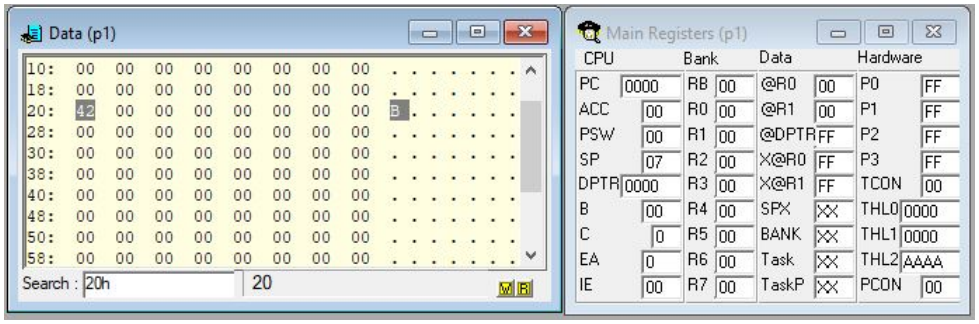


Program 3.d: TO CONVERT ASCII NUMBER TO HEXADECIMAL, STORED AT 20H IN INTERNAL RAM. STORE RESULT AT 50H IN INTERNAL RAM.

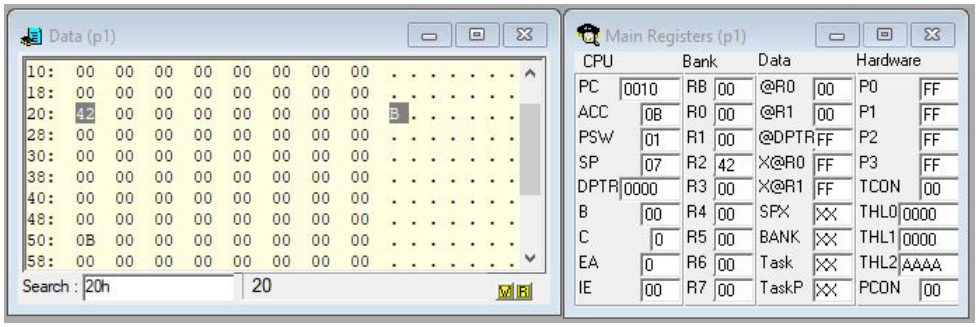
```
        ORG 0000H       ; start the program at 0000h
        MOV A,20H       ; take the ASCII number into Acc.
        MOV R2,A        ; save a copy into R2 register
        SUBB A,# 41H    ; compare ASCII number with 41h
        MOV A,R2        ; get back the ASCII number into Acc.
        JC skip         ; if ASCII number is > 41, jump to label to skip
        CLR C           ; else, clear CY flag
        SUBB A,# 07H    ; subtract 7 and also 30
skip:   CLR C           ; clear CY flag
        SUBB A,# 30H    ; convert to hexa
        MOV 50H,A       ; save the result into 50h
stop:   SJMP stop       ; stop incrementing Program Counter
        END             ; end of the program
```

Result:

Before Execution-



After Execution-

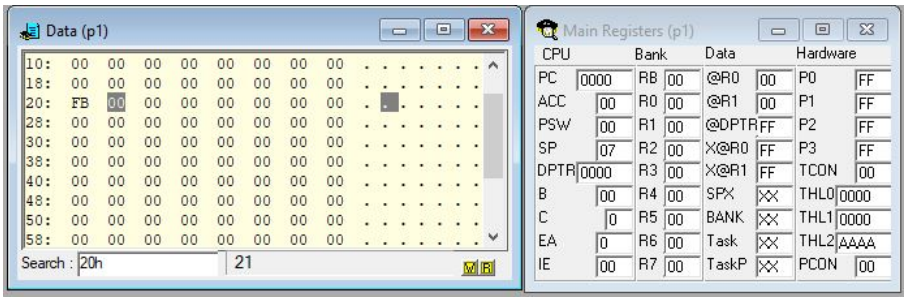


Program 3.e: TO CONVERT DECIMAL TO HEXADECIMAL, STORED AT 20H IN INTERNAL RAM. STORE RESULT AT 21H IN INTERNAL RAM.

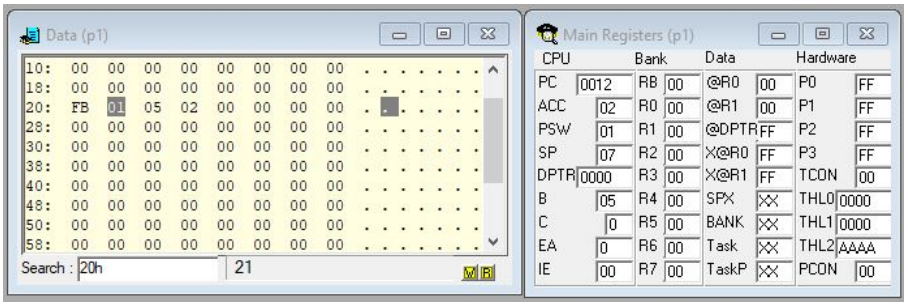
```
        ORG 0000H           ; start the program at 0000h
        MOV R1,20H          ; take the decimal number into R2
        MOV A,R1            ; get the decimal no. into Acc.
        ANL A,# 0FH         ; mask lower nibble, to get lower byte
        MOV R2,A            ; save into R2 reg.
        MOV A,R1            ; take the decimal number into Acc.
        ANL A,# 0F0H        ; mask the higher nibble, to get higher byte
        SWAP A              ; convert to lower nibble
        MOV 0F0H,# 0AH      ; load 10 into register B
        MUL AB              ; multiply higher nibble with 10
        ADD A,R2            ; convert by adding the result with higher byte
        MOV 21H,A           ; save the result in 21h
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:

Before Execution-



After Execution-

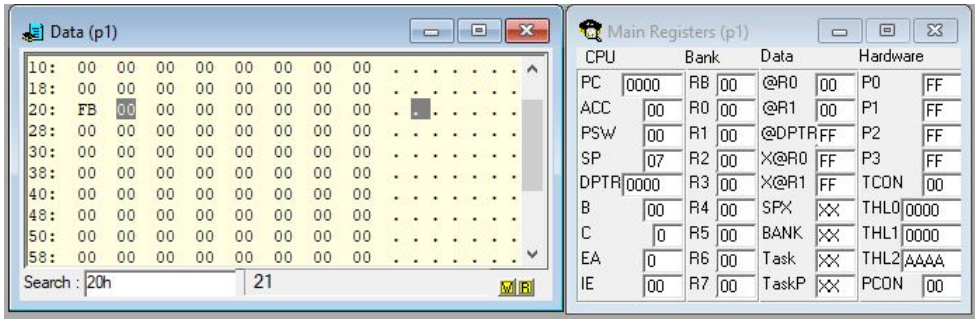


Program 3.f: TO CONVERT HEXADECIMAL TO DECIMAL, STORED AT 20H IN INTERNAL RAM. STORE RESULT AT 21H ONWARDS IN INTERNAL RAM.

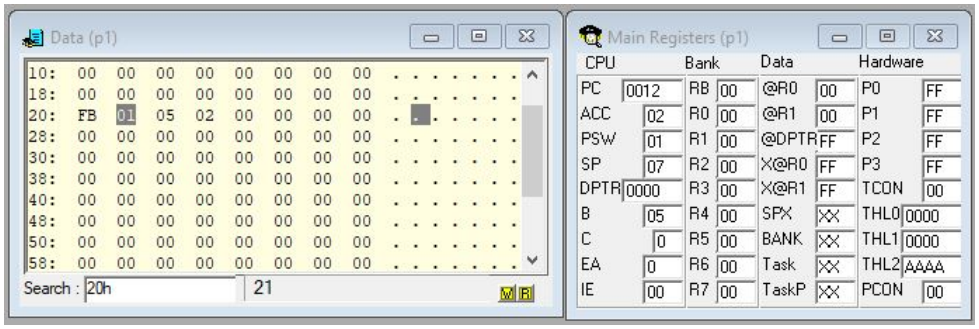
```
        ORG 0000H           ; start the program at 0000h
        MOV A,20H           ; take hexa number into Acc.
        MOV 0F0H,# 0AH      ; take 10 into B reg.
        DIV AB              ; divide hexa number by 10
        MOV 21H,0F0H        ; save unit place at 21h
        MOV 0F0H ,# 0AH     ; take 10 into B reg.
        DIV AB              ; divide quotient by 10
        MOV 22H,0F0H        ; save 10th place at 22h
        MOV 23H,A           ; save 100th place at 23h
stop:   SJMP stop           ; stop incrementing Program Counter
        END                 ; end of the program
```

Result:

Before Execution-



After Execution-



7 Experiment 4: Counters, Timers, Serial PORT Programming and Bit Manipulation

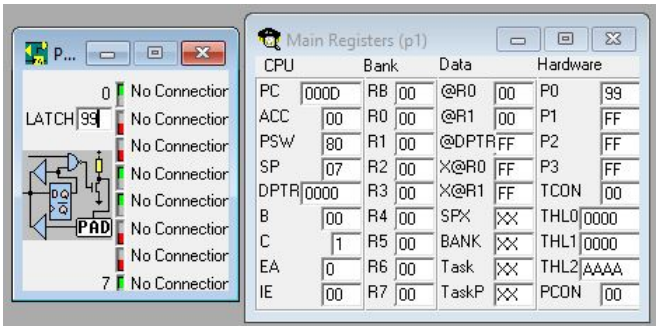
Programs:

1. To demonstrate Decimal UP counter from 00 to 99.
2. To demonstrate Decimal DOWN counter from 99 to 00.
3. To demonstrate Binary UP counter from 00h-FFh.
4. To demonstrate Binary DOWN counter from FFh to 00h.
5. To count from 00h to E9h (i.e., upper limit is specified).
6. To separate EVEN or ODD numbers in a given array stored form 3000h external RAM. Store EVEN numbers from 40h and ODD numbers from 48h internal RAM.
7. To separate Positive or Negative numbers in a given array stored form 3000h external RAM. Store positive numbers from 40h and negative numbers from 48h internal RAM.
8. To transfer a string of data through SERIAL PORT.
9. To receive a string of data serially and transmit the same serially using SERIAL PORT.

Program 4.a: TO DEMONSTRATE DECIMAL UP COUNTER FROM 00 TO 99.

```
        ORG 0000H          ; start program from 0000h
        MOV A,# 00H        ; initialize acc. with 00
        MOV R1,# 64H       ; load R1 with number of counts
loop:   MOV 80H,A          ; send COUNT to Port-0
        ADD A,# 01H        ; increment COUNT by 1
        DA A               ; convert to decimal
        ACALL delay        ; call the delay
        DJNZ R1, loop      ; decrement R1 i.e., till count becomes zero
stop:   SJMP stop          ; stop incrementing PC
          ; subroutine program to create some delay using instructions
delay:  MOV R2,# 03H       ; load count to R2
go:     MOV R3,# 02H       ; load count to R3
wait:   DJNZ R3, wait      ; decrement count until it becomes zero
        DJNZ R2, go        ; decrement count until it becomes zero
        RET                ; return to main program
        END                ; end the program
```

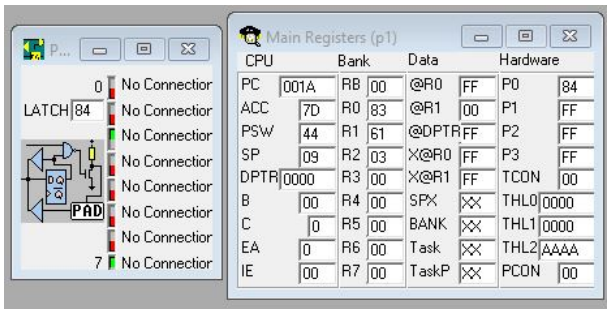
Result: The decimal count form 00 to 99 is observed on PORT 0.
After Execution-



Program 4.b: TO DEMONSTRATE DECIMAL DOWN COUNTER FROM 99 TO 00.

```
        ORG 0000H          ; start program from 0000h
        MOV A,# 99h        ; initialize acc. with 99
        MOV R1, # 64H      ; load R1 with number of counts
loop:   MOV 80H, A         ; send COUNT to Port-0
        DEC A              ; decrement COUNT by 1
        MOV R0, A          ; save acc. value into R0
        SUBB A,# 0AH       ; compare count with 10
        JC skip            ; if count is < 10, jump to label skip
        MOV A, R0          ; else, load back the count into Acc.
        SUBB A,# 06H       ; convert it to decimal by subtracting 6
        SJMP down          ; stop incrementing PC
skip:   MOV A, R0          ; continue by taking back the count
down:   ACALL DELAY        ; call a delay
        DJNZ R1, loop      ; decrement R1 i.e., till count becomes zero
stop:   SJMP stop          ; stop incrementing PC
          ; subroutine program to create some delay using instructions
delay:  MOV R2,# 03H       ; load count to R2
go:     MOV R3,# 02H       ; load count to R3
wait:   DJNZ R3, wait      ; decrement count until it becomes zero
        DJNZ R2, go        ; decrement count until it becomes zero
        RET                ; return to main program
        END                ; end the program
```

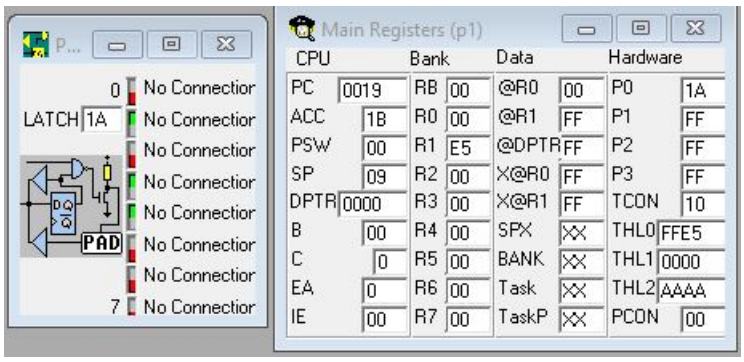
Result: The decimal count form 99 to 00 is observed on PORT 0.
After Execution-



Program 4.c: TO DEMONSTRATE BINARY UP COUNTER FROM 00H-FFH.

```
        ORG 000H            ; start program from 000h
        MOV A,# 00H         ; load acc with initial count value of 00h
        MOV R1,# 255        ; load R1 with count of 256
loop:   MOV 80H, A          ; send COUNT to port-0
        ADD A,# 01H         ; increment COUNT by 1
        ACALL delay         ; call the delay
        DJNZ R1, loop       ; decrement R1 i.e., till count becomes zero
stop:   SJMP stop           ; stop incrementing PC
            ; subroutine program to create 100 μs delay using TIMER
delay:  MOV 89H,# 01H       ; select timer-0, mode-1
up:     MOV 8AH,# 0A4H      ; load the TL0 with delay value
        MOV 8CH,# 0FFH      ; load TH0 with delay value
        SETB 8CH            ; start timer TR0
wait:   JNB 8DH, wait       ; monitor the overflow flag TF0
        CLR 8CH             ; stop timer
        CLR 8DH             ; clear the overflow flag TF0
        RET                 ; return to main program
        END                 ; end the program
```

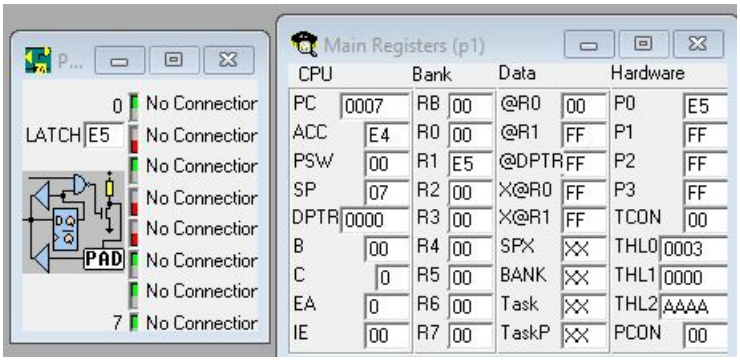
Result: The binary counts from 0000000b to 11111111b are observed on PORT0 After Execution-



Program 4.d: TO DEMONSTRATE BINARY DOWN COUNTER FROM FFH TO 00H.

```
        ORG 000H            ; start program from 000h
        MOV A,# 0FFH        ; load acc. with initial count value FFh
        MOV R1,# 255        ; load R1 with count of 255
loop:   MOV 80H, A          ; send COUNT to Port-0
        DEC A               ; decrement COUNT by 1
        ACALL DELAY         ; call the delay
        DJNZ R1, loop       ; decrement R1 i.e., till count becomes zero
stop:   SJMP stop           ; stop incrementing PC
            ; subroutine program to create 100 μs delay using TIMER
delay:  MOV 89H,# 01H       ; select timer-0, mode-1
up:     MOV 8AH,# 0A4H      ; load the TL0 with delay value
        MOV 8CH,# 0FFH      ; load TH0 with delay value
        SETB 8CH            ; start timer TR0
wait:   JNB 8DH, wait       ; monitor the overflow flag TF0
        CLR 8CH             ; stop timer
        CLR 8DH             ; clear the overflow flag TF0
        RET                 ; return to main program
        END                 ; end the program
```

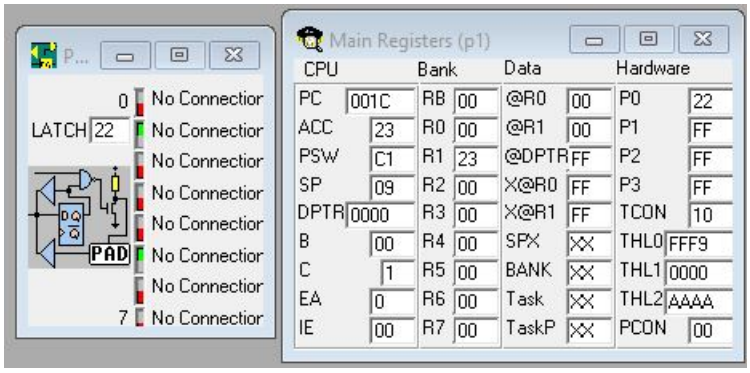
Result: The binary counts from 11111111b to 0000000b are observed on PORT0 After Execution-



Program 4.e: TO COUNT FROM 00H TO E9H (i.e., UPPER LIMIT IS SPECIFIED).

```
        ORG 000H              ; start program from 000h
        MOV A,# 00H           ; load a with first count 00h
loop:   MOV 80H, A            ; send COUNT to Port-0
        INC A                 ; increment COUNT by 1
        ACALL delay           ; call the delay
        CJNE A,# 0E9H,loop    ; check COUNT has reached the specified value
                              or not, if carry generated, display next count and
                              repeat
stop:   SJMP stop             ; stop incrementing PC
            ; subroutine program to create 100 μs delay using TIMER
delay:  MOV 89H,# 01H         ; select timer-0, mode-1
up:     MOV 8AH,# 0A4H        ; load the TL0 with delay value
        MOV 8CH,# 0FFH        ; load TH0 with delay value
        SETB 8CH              ; start timer TR0
wait:   JNB 8DH, wait         ; monitor the overflow flag TF0
        CLR 8CH               ; stop timer
        CLR 8DH               ; clear the overflow flag TF0
        RET                   ; return to main program
        END                   ; end the program
```

Result: The count form 00h to E9h is observed on PORT 0.
After Execution-

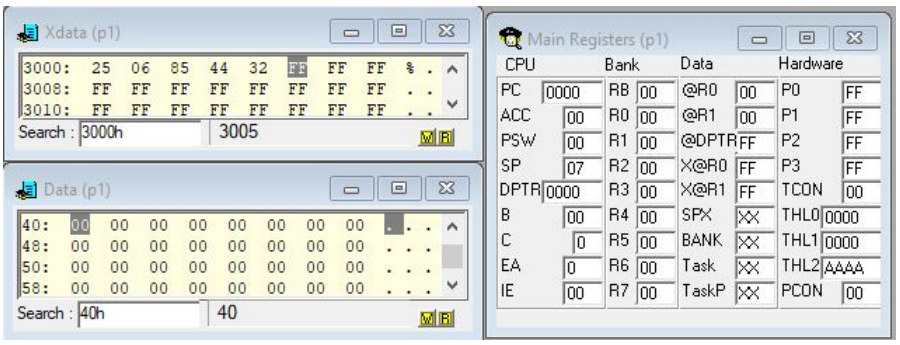


Program 4.f: TO SEPARATE EVEN OR ODD NUMBERS IN A GIVEN ARRAY STORED FORM 3000H EXTERNAL RAM. STORE EVEN NUMBERS FROM 40H AND ODD NUMBERS FROM 48H INTERNAL RAM.

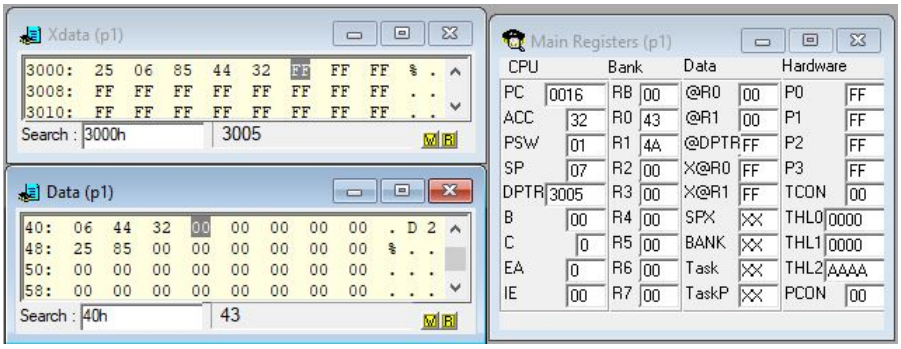
```
        ORG 0000H               ; start the program at 0000h
        MOV R2,# 05H            ; load the count value in reg R2
        MOV DPTR,# 3000H        ; load DPTR with start address of array 3000h
        MOV R0,# 40H            ; load R0 with 40h where even numbers are stored
        MOV R1,# 48H            ; load R1 with 48h where even number are stored
loop:   MOVX A, @DPTR           ; copy the number into register Acc.
        JB 0E0H, next           ; check D0 bit of Acc. to find odd or even number,
                                  If Odd jump to label next
        MOV @R0, A              ; else, store the even number starting at 40h
        INC R0                  ; increment R0 to point next locn.
        SJMP skip               ; an unconditional jump to skip
next:   MOV @R1, A              ; store the odd number starting at 48h
        INC R1                  ; increment R1 to point next locn.
skip:   INC DPTR                ; increment DPTR to point next data
        DJNZ R2, loop           ; repeat until count (R2) becomes zero
stop:   SJMP stop               ; stop incrementing Program Counter
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-

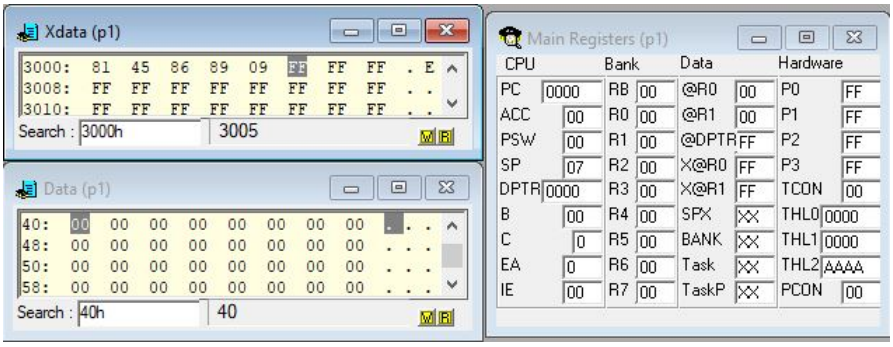


Program 4.g: PROGRAM TO SEPARATE POSITIVE OR NEGATIVE NUMBERS IN A GIVEN ARRAY STORED FORM 3000H EXTERNAL RAM. STORE POSITIVE NUMBERS FROM 40H AND NEGATIVE NUMBERS FROM 48H INTERNAL RAM.

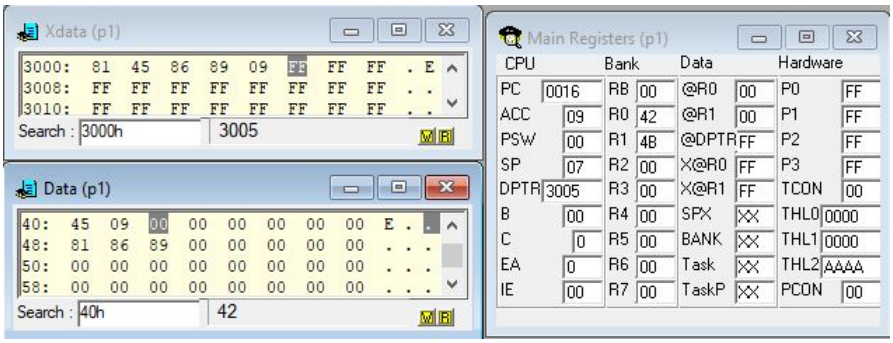
```
        ORG 0000H              ; start the program at 0000h
        MOV R2,# 05H           ; load the count value in reg R2
        MOV DPTR,# 3000H       ; load DPTR with start address of array 3000h
        MOV R0,# 40H           ; load R0 with 40h where positive numbers are stored
        MOV R1,# 48H           ; load R1 with 48h where negative number are stored
loop:   MOVX A, @DPTR          ; copy the number into register Acc.
        JB 0E7H, next          ; check D7 bit of Acc. to find positive or negative number, If negative jump to label next
        MOV @R0, A             ; else, store the positive number starting at 40h
        INC R0                 ; increment R0 to point next locn.
        SJMP skip              ; an unconditional jump to skip
next:   MOV @R1, A             ; store the negative number starting at 48h
        INC R1                 ; increment R1 to point next locn.
skip:   INC DPTR               ; increment DPTR to point next data
        DJNZ R2, loop          ; repeat until count (R2) becomes zero
stop:   SJMP stop              ; stop incrementing Program Counter
        END                    ; end of the program
```

Result:

Before Execution-



After Execution-

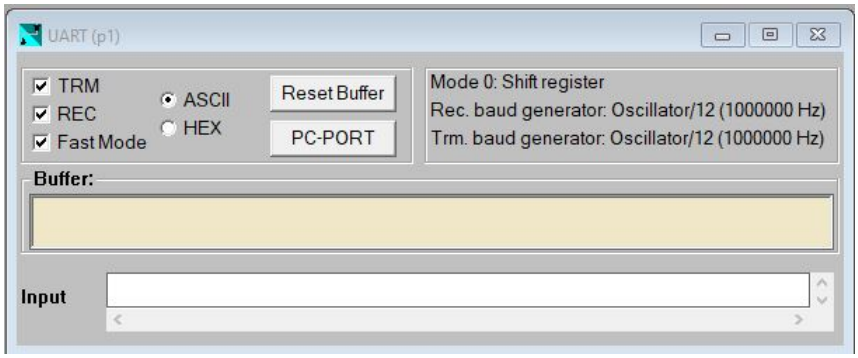


Program 4.h: TO TRANSFER A STRING OF DATA THROUGH SERIAL PORT.

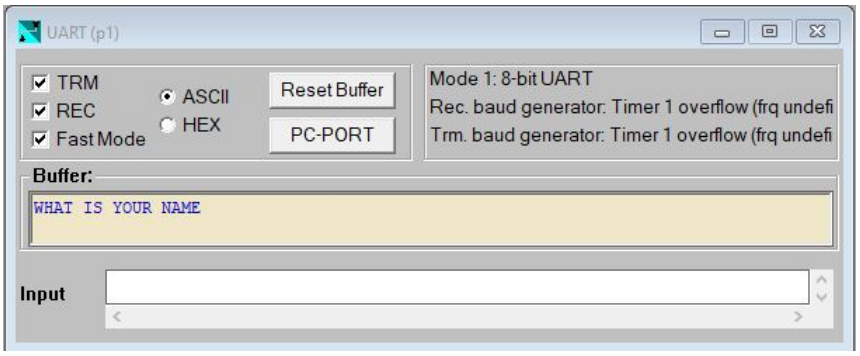
```
        ORG 0000H               ; start the program at 0000h
        MOV 89H,# 20H           ; select timer-1, mode- 2 in TMOD register
        MOV 98H,# 50H           ; 8-bit, 1 stop bit, REN enabled in SCON register
        MOV 8DH,# -3            ; select 9600 band rate in TH1 flag
        SETB 8EH                ; start the timer
        MOV DPTR,# table        ; point to the lookup table
nxtchar:CLR A                   ; clear acc before accessing code space
        MOVC A, @A+DPTR         ; access the code area and load the character into
                                  acc. pointed by A+DPTR
        ACALL transfer          ; call subroutine t transfer character serially
        INC DPTR                ; increment DPTR to point next location
        JNZ nxtchar             ; repeat until all the characters has been transfer
stop:   SJMP stop               ; stop incrementing the PC
                ; subroutine program to transfer character serially
transfer:MOV 99H, A             ; load the character from acc. to SBUF register
wait:   JNB 99H, wait           ; monitor TI flag and wait for lost bit to transfer
        CLR 99H                 ; clear TI flag to transfer next byte
        RET                     ; return to subroutine
table:  DB "WHAT IS             ; use DB directive to store the string
        YOUR NAME ",0
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-

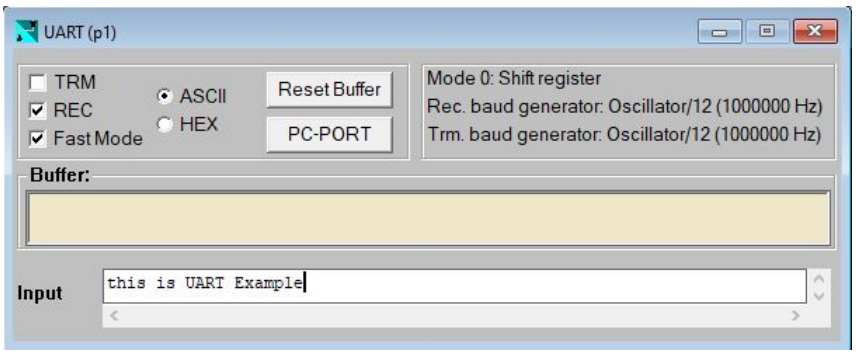


Program 4.i: TO RECEIVE A STRING OF DATA SERIALLY AND TRANSMIT THE SAME SERIALLY USING SERIAL PORT.

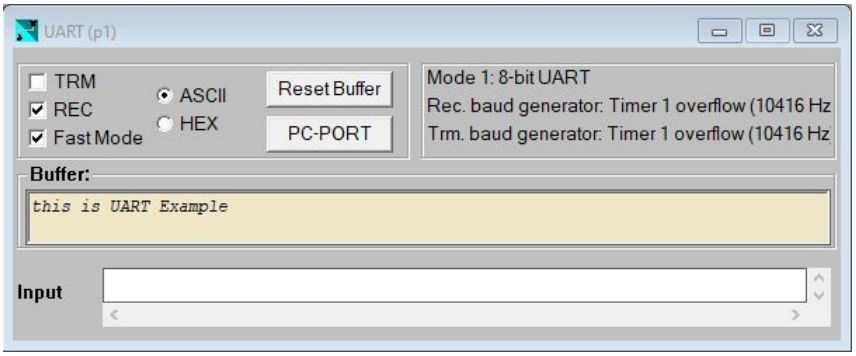
```
        ORG 0000H               ; start the program at 0000h
        MOV 89H,# 20H           ; select timer-1, mode- 2 in TMOD register
        MOV 98H,# 50H           ; 8-bit, 1 stop bit, REN enabled in SCON register
        MOV 8DH,# 0FDH          ; select 9600 band rate in TH1 flag
        SETB 8EH                ; start the timer
        MOV R0,# 20             ; load the count value of number of characters
nxtchar:CLR A                   ; clear acc. before accessing code space
        MOVC A, @A+DPTR         ; access the code area and load the data into acc.
                                  pointed by A+DPTR
        ACALL receive           ; call subroutine to receive data serially
        ACALL transfer          ; call subroutine to transfer data serially
        INC DPTR                ; increment DPTR to point next location
        DJNZ R0, nxtchar        ; repeat until all characters has been transferred
stop:   SJMP stop               ; stop incrementing the PC
                ; subroutine program to receive character serially
receive: MOV A, 99H             ; load the character from SBUF register to acc.
wait1:  JNB 98H, wait1          ; monitor RI flag and wait for last bit to receive
        CLR 98H                 ; clear RI flag to transfer next byte
        RET                     ; return from subroutine
                ; subroutine program to transfer character serially
transfer:MOV 99H, A             ; load the character from acc. to SBUF register
wait:   JNB 99H, wait           ; monitor T1 flag and wait for lost bit to transfer
        CLR 99H                 ; clear TI flag to transfer next byte
        RET                     ; return to subroutine
        END                     ; end of the program
```

Result:

Before Execution-



After Execution-



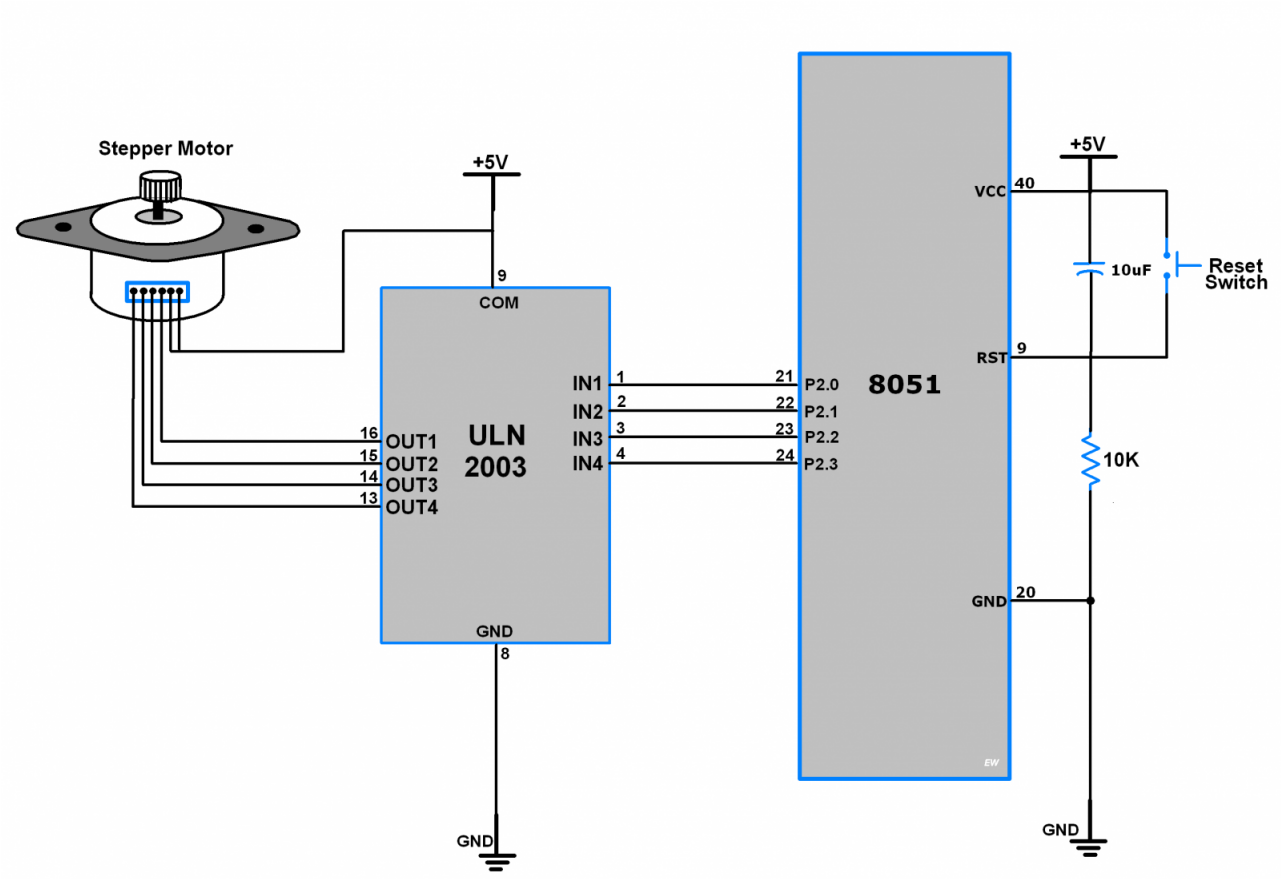
Interfacing Programs

8 Experiment 5: Stepper Motor Interface

Programs:

1. To interface stepper motor and rotate in clockwise direction continuously.
2. To interface stepper motor and rotate in anticlockwise direction continuously.
3. To interface stepper motor and rotate in 360° clockwise direction.

Stepper Motor Interfacing Diagram with 8051



Stepper motor takes 1.8° per step, to rotate 360° , it takes $\frac{360°}{1.8°} = 200$ steps. The following table lists the number of values to send to the stepper motor to rotate angle wise.

Angle	No. of Values to Send
45°	25
90°	50
180°	100
360°	200
720°	400

Program 5.a: TO INTERFACE STEPPER MOTOR AND ROTATE IN CLOCKWISE DIRECTION CONTINUOUSLY.

```
#include<reg51.h>
void delay(void);
void main(void)
{
        while(1)                    // infinite loop
        {
                P2=0x07;
                delay();

                P2=0x0B;
                delay();

                P2=0x0D;
                delay();

                P2=0x0E;
                delay();
        }
}

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

Result:

Stepper motor is interfaced with 8051 and programmed to rotate in clockwise direction.

Program 5.b: TO INTERFACE STEPPER MOTOR AND ROTATE IN ANTICLOCKWISE DIRECTION CONTINUOUSLY.

```
#include<reg51.h>
void delay(void);
void main(void)
{
        while(1)                    // infinite loop
        {
                P2=0x0E;
                delay();

                P2=0x0D;
                delay();

                P2=0x0B;
                delay();

                P2=0x07;
                delay();
        }
}

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

Result:

Stepper motor is interfaced with 8051 and programmed to rotate in anticlockwise direction.

Program 5.c: TO INTERFACE STEPPER MOTOR AND ROTATE IN 360° CLOCKWISE DIRECTION.

```
#include<reg51.h>
void delay(void);
void main(void)
{
Int j;
        for(j =0; j <=50; j ++) /*execute for loop 50 times to
        {                        rotate stepper motor 360 degrees*/
                P2=0x07;
                delay();

                P2=0x0b;
                delay();

                P2=0x0d;
                delay();

                P2=0x0e;
                delay();
        }
while(1);
}

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

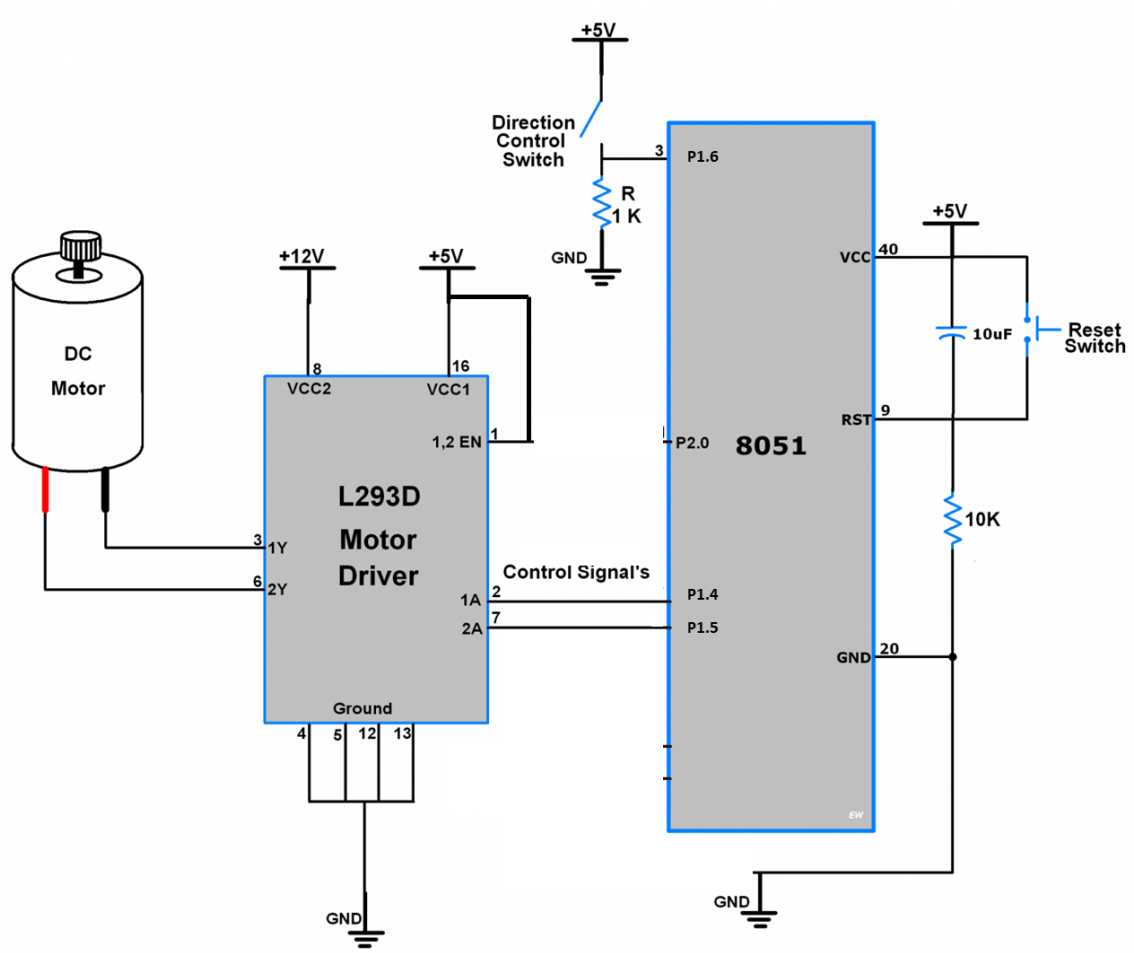
Result:

Stepper motor is interfaced with 8051 and programmed to rotate in 360° clockwise direction.

9 Experiment 6: DC Motor Interface

Program:
DC motor interfacing with 8051 and control the direction.

DC Motor Interfacing Diagram with 8051



Program: DC MOTOR INTERFACING WITH 8051 AND CONTROL THE DIRECTION.

```
#include<reg51.h>
sbit outdir_pin_a1=P1^4;
sbit outdir_pin_a2=P1^5;
sbit dir_pin=P1^6;
void delay(unsigned int);
void main()
{
        dir_pin=1;                          //make P1.6 as output pin
        if(dir_pin==0)                      /*check for input
        {                                   using switch*/
                outdir_pin_a1=0;            //rotate anticlockwise
                outdir_pin_a2=1;
                delay(4000);
        }
        else
        {
                outdir_pin_a1=1;            //rotate clockwise
                outdir_pin_a2=0;
                delay(4000);
        }
}

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

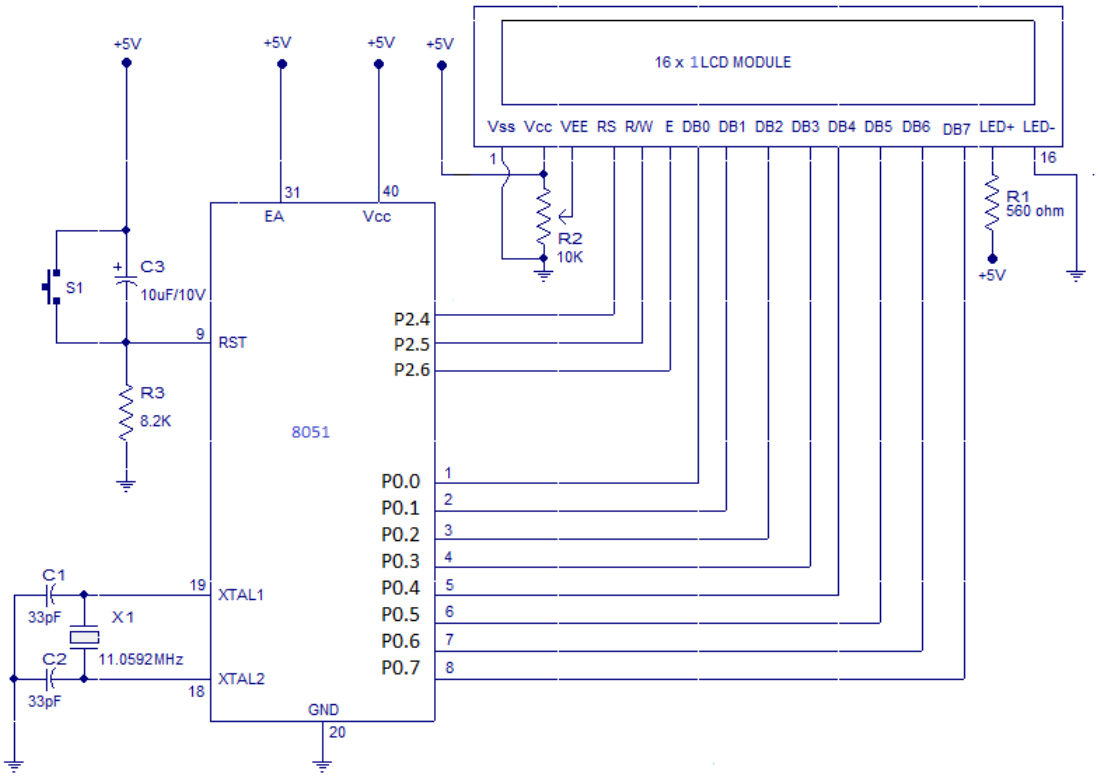
Result:

DC motor is interfaced with 8051 and programmed to rotate in clockwise when switch is high and anticlockwise direction when switch is low.

10 Experiment 7: Alphanumerical LCD Panel Interface

Program:
Liquid Crystal Display interfacing with 8051.

16x1 LCD Interfacing Diagram with 8051



LDC commands:

Codes in Hex	Commands to LCD
1	Clear Display Screen
2	Return Home
4	Decrement Cursor
6	Increment Cursor
5	Shift Display Right
7	Shift Display Left
8	Display OFF, Cursor OFF
A	Display OFF, Cursor ON
C	Display ON, Cursor OFF
E	Display ON, Cursor Blinking
10	Shift Cursor Position to Left
14	Shift Cursor Position to Right
18	Shift entire display to the Left
1C	Shift entire display to the Right
80	Force Cursor to Beginning to 1st line
C0	Force Cursor to Beginning to 2nd line
38	2 lines and 5x7 matrix

Program: Interfacing 16x1 LCD panel with 8051.

```
#include<reg51.h>
void lcddata(unsigned char value);
void lcdcmd(unsigned char value);
void msdelay(unsigned int itime);
sfr ldata=0x80;                    //assign PORT 0 to variable ldata
sbit rs=P2^4;                      //assign P2.4 pin to Register select
sbit rw=P2^5;                      //assign P2.5 pin to read/write select
sbit en=P2^6;                      //assign P2.6 pin to enable

void main()
{
        lcdcmd(0x38);              //2 lines and 5x7 matrix
        msdelay(250);

        lcdcmd(0x0e);              //Display ON, Cursor Blinking
        msdelay(250);

        lcdcmd(0x01);              //Clear Display Screen
        msdelay(250);

        lcdcmd(0x06);              //Increment Cursor
        msdelay(250);

        lcdcmd(0x80);              //Force Cursor to Beginning to 1st line
        msdelay(250);

        lcddata('E');
        msdelay(250);

        lcddata('E');
        msdelay(250);

        lcddata('E');
        msdelay(250);

        lcdcmd(0x14);              //Shift Cursor Position to Right
        msdelay(250);

        lcddata('B');
        msdelay(250);

        lcddata('I');
        msdelay(250);

        lcddata('T');
        msdelay(250);

        lcddata('M');
```

```
        msdelay(250);

        lcdcmd(0xc0);           //Force Cursor to Begin to 2nd line
        msdelay(250);

        lcdcmd(0x14);           //Shift Cursor Position to Right
        msdelay(250);

        lcddata('1');
        msdelay(250);

        lcddata('9');
        msdelay(250);

        lcddata('9');
        msdelay(250);

        lcddata('7');
        msdelay(250);
        here: goto here;
}
void lcdcmd(unsigned char value)
{
        ldata=value;            //send command to port
        rs=0;                   //select command register
        rw=0;                   //select write operation
        en=1;                   /*send high to low pulse
        msdelay(1);             to latch command*/
        en=0;
        return;
}
void lcddata(unsigned char value)
{
        ldata=value;            //send data to port
        rs=1;                   //select data register
        rw=0;                   //select write operation
        en=1;                   //send high to low pulse
        msdelay(1);              to latch data*/
        en=0;
        return;
}
void msdelay(unsigned int itime)
{
        unsigned int i,j;
        for(i=0;i<itime;i++)
                for(j=0;j<1275;j++);
}
```

Result:

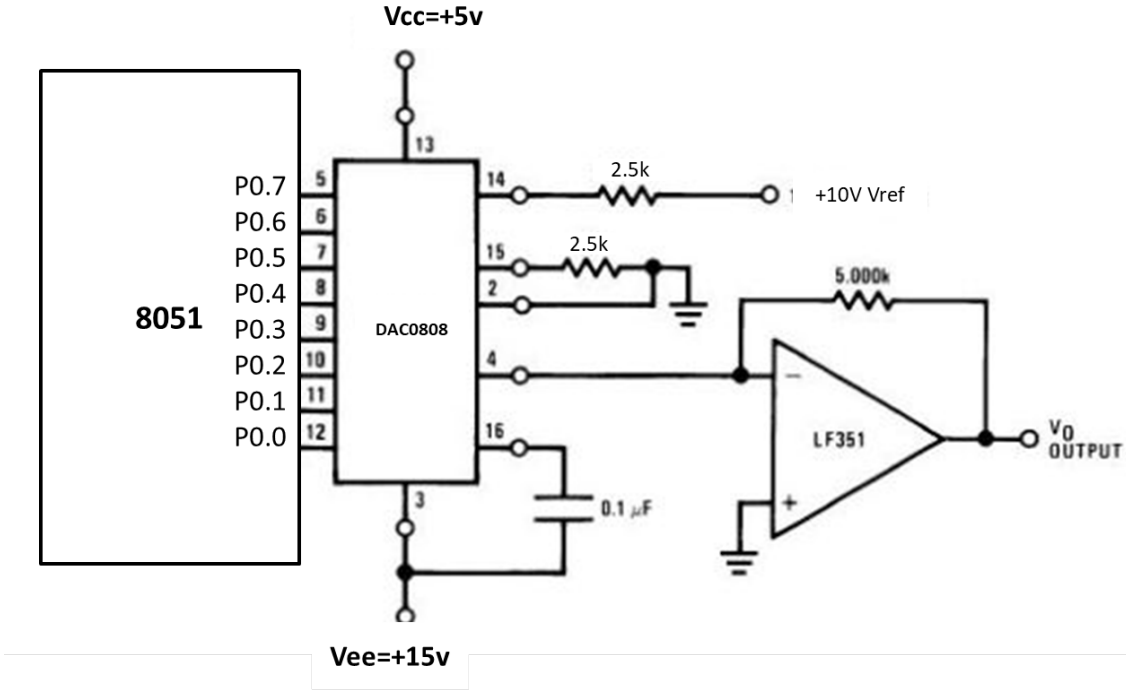
16x1 LCD panel is interfaced with 8051 and EEE BITM 1997 displayed on the LCD.

11 Experiment 8: Digital to Analog Converter (DAC) Interface

Programs:

1. To generate sine waveform.
2. To generate square waveform.
3. To generate triangular waveform.
4. To generate ramp waveform.

DAC Interfacing Diagram with 8051



The 8-bit DAC 0808 IC converts digital data into equivalent analog Current. Hence we require an I to V converter to convert this current into equivalent voltage. Equivalent analog output is given as:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

Where D0 is the LSB and D7 is the MSB. Considering $I_{ref} = 2mA$ the max output current is 1.99 mA.

Program 8.a: TO GENERATE SINE WAVEFORM.

Sine wave calculations:

To generate a sine wave, we need a table representing the magnitude of sine angles between 0 to 360° each angle with 7.5° increments. Considering full-scale voltage for DAC as 10 V output. multiply V_{out} by 256 steps/10 V=25.6 steps per Volt. Formulating below equation:

$$V_{out} = [5V + (5 \times sin\theta)] \times 25.6$$

Sl.No	θ	V_{out}	**Hex**	**Sl.No**	θ	V_{out}	**Hex**
1	0	128	80	27	195	95	5f
2	7.5	144	90	28	202.5	79	4f
3	15	161	a1	29	210	64	40
4	22.5	177	b1	30	217.5	51	33
5	30	192	c0	31	225	38	26
6	37.5	205	cd	32	232.5	26	1b
7	45	218	da	33	240	17	12
8	52.5	229	e5	34	247.5	9	0a
9	60	238	ee	35	255	4	05
10	67.5	246	f6	36	262.5	2	02
11	75	251	fb	37	270	0	00
12	82.5	254	fe	38	277.5	2	02
13	90	256	ff	39	285	4	05
14	97.5	254	fe	40	292.5	9	0a
15	105	251	fb	41	300	17	12
16	112.5	246	f6	42	307.5	26	1b
17	120	238	ee	43	315	38	26
18	127.5	229	e5	44	322.5	51	33
19	135	218	da	45	330	64	40
20	142.5	205	cd	46	337.5	79	4f
21	150	192	c0	47	345	95	5f
22	157.5	177	b1	48	352.5	112	70
23	165	161	a1	49	360	128	80
24	172.5	144	90				
25	180	128	80				
26	187.5	112	70				

```
#include<reg51.h>
unsigned char sine[49]={0x80, 0x90, 0xa1, 0xb1, 0xc0, 0xcd,
0xda, 0xe5, 0xee, 0xf6, 0xfb, 0xfe, 0xff, 0xfe, 0xfb, 0xf6,
0xee, 0xe5, 0xda, 0xcd, 0xc0, 0xb1, 0xa1, 0x90, 0x80, 0x70,
0xf5, 0x4f, 0x40, 0x33, 0x26, 0x1b, 0x12, 0x0a ,0x05, 0x02,
0x00, 0x02, 0x05, 0x0a, 0x12, 0x1b, 0x26, 0x33, 0x40, 0x4f,
0x5f, 0x70, 0x80};
unsigned char count;
void main()
{
        while(1)
        {
                for(count=0;count<49;count++);
                P0=sine[count];
        }
}
```

Result:
DAC is interfaced with 8051 and programmed to generate Sine Waveform.

Program 8.b: TO GENERATE SQUARE WAVEFORM.

```
#include<reg51.h>
void delay (void);
void main()
{
        while(1)
        {
                P0=0x0;
                delay();

                P0=0xff;
                delay();
        }
}

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

Result:
DAC is interfaced with 8051 and programmed to generate Square Waveform.

Program 8.c: TO GENERATE TRIANGULAR WAVEFORM.

```
#include<reg51.h>
idata unsigned char count;
void main()
{
        while(1)
        {
                for(count=0;count!=0xff;count++)
                {
                        P0=count;
                }
                for(count=0xff;count>0;count--)
                {
                        P0=count;
                }
        }
}
```

Result:
DAC is interfaced with 8051 and programmed to generate Triangular Waveform.

Program 8.d: TO GENERATE RAMP WAVEFORM.

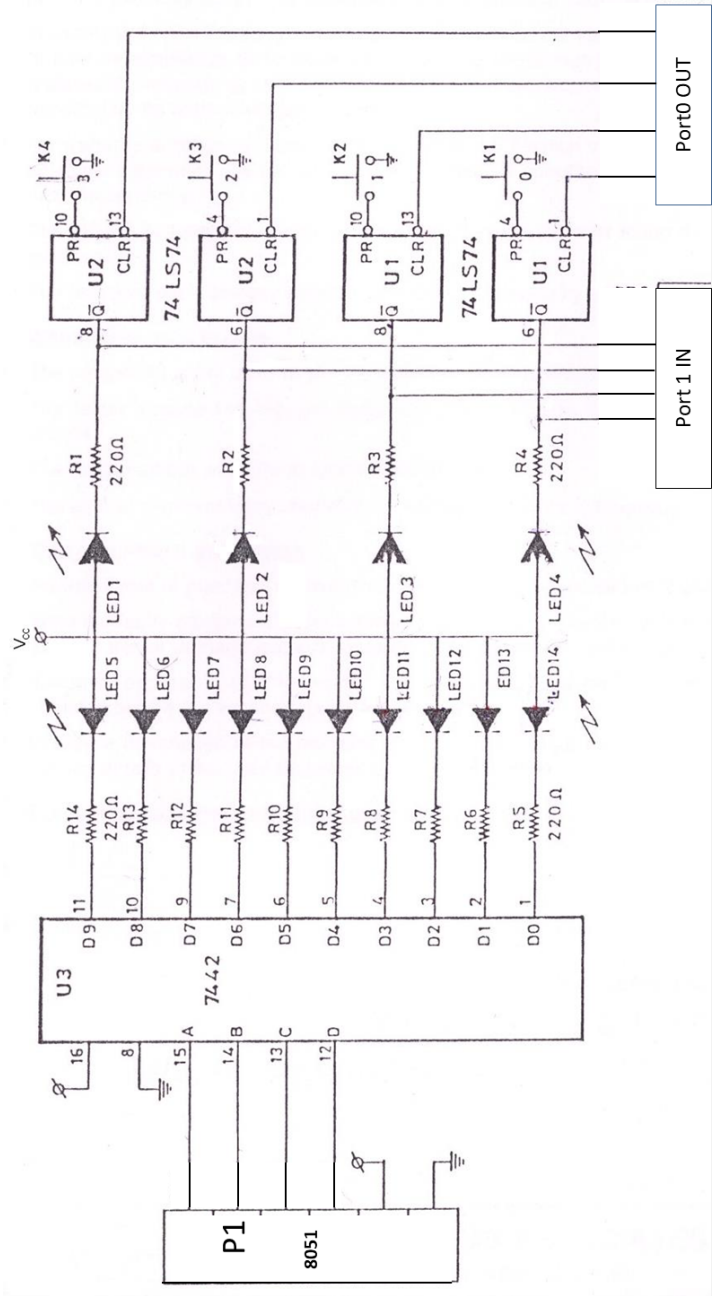
```
#include<reg51.h>
idata unsigned char count;
void main()
{
count=0;
        while(1)
        {
                P0=count;
                count++;
        }
}
```

Result:
DAC is interfaced with 8051 and programmed to generate Ramp Waveform.

12 Experiment 9: Elevator Interface

Program:
Elevator interfacing with 8051 and Simulate the operation.

Elevator Interfacing Diagram with 8051



Program: ELEVATOR INTERFACING WITH 8051 AND SIMULATE THE OPERATION.

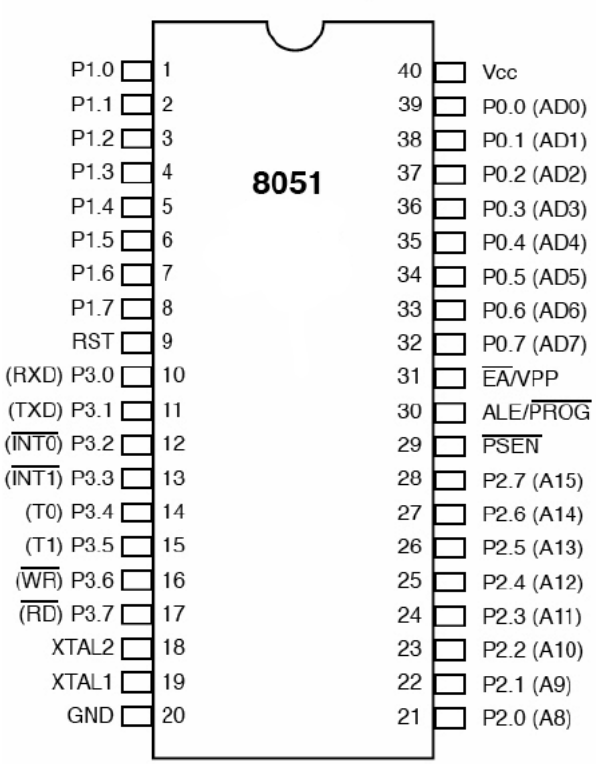
```
#include <reg51.h>
void delay(unsigned int);
main()
{
unsigned char flr[9]={0xff,0x00,0x03,0xff,0x06,0xff,0xff,0xff,0x09};
unsigned char fclr[9]={0xff,0x0E0,0xD3,0xff,0xb6,0xff,0xff,0xff,0x79};
unsigned char reqflr,curflr=0x01,i,j;
P0=0x00;
P0=0X0f0;
while(1)
{
        P1 =0x0f;
        reqflr= P1|0x0f0;
        while(reqflr= =0x0ff)
                reqflr= P1|0x0f0;
        reqflr= ~ reqflr;
        if(curflr= =reqflr)
        {
                P0=fclr[curflr];
                continue;
        }
        else if(curflr>reqflr)
        {
                i=flr[curflr]-flr[reqflr];
                j=flr[curflr];
                for(;i>0;i--)
                {
                        P0 =0x0f0|j;
                        j--;
                        delay(100000);
                }
        }
        else
        {
                i=flr[reqflr]-flr[curflr];
                j=flr[curflr];
                for(;i>0;i--)
                {
                        P0 =0x0f0|j;
                        j++;
                        delay(100000);
                }
        }
        curflr=reqflr;
        P0 =fclr[curflr];
}
}
```

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

Result:
Elevator is interfaced with 8051 and simulated its operation.

APPENDIX

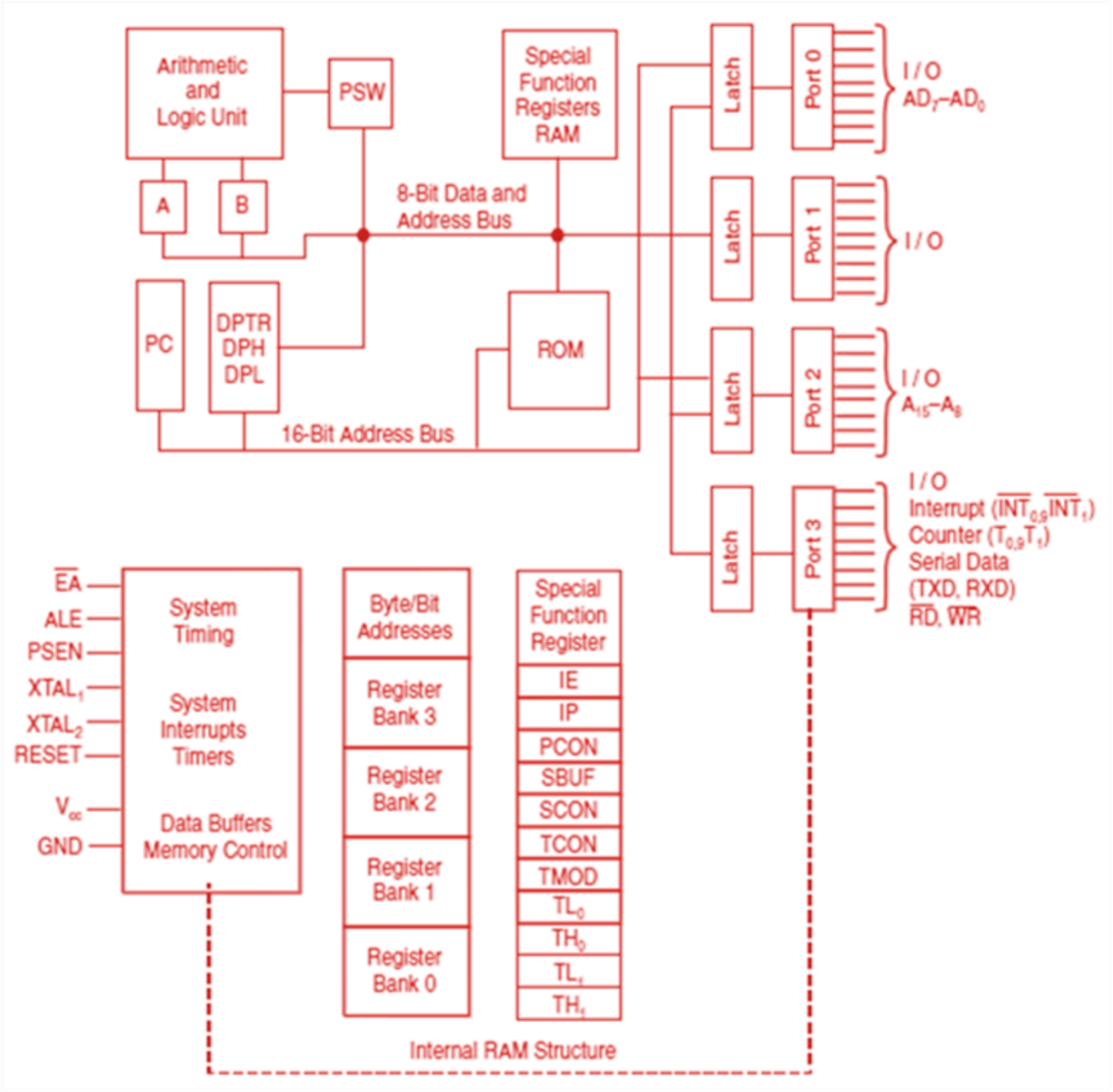
13 Appendix A: 8051 Pin Details



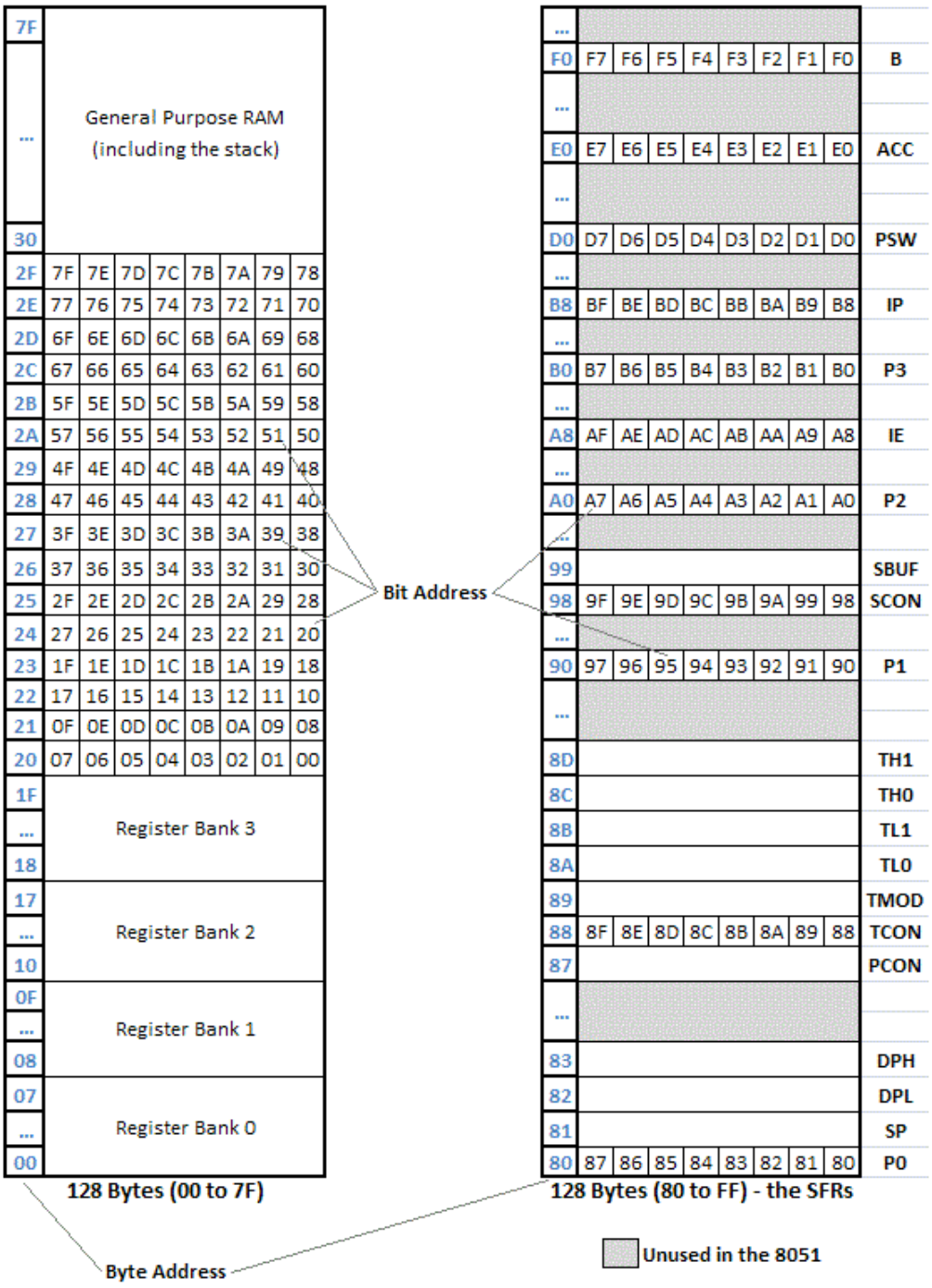
Pins 1-8: Port 1 Each of these pins can be configured as an input or an output.
Pin 9: RS A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins10-17: Port 3 Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions:
Pin 10: RXD Serial asynchronous communication input or Serial synchronous communication output.
Pin 11: TXD Serial asynchronous communication output or Serial synchronous communication clock output.
Pin 12: (INT0) Interrupt 0 input.
Pin 13: (INT1) Interrupt 1 input.
Pin 14: T0 Counter 0 clock input.
Pin 15: T1 Counter 1 clock input.
Pin 16: (WR) Write to external (additional) RAM.
Pin 17: (RD) Read from external RAM.
Pin 18, 19: X2, X1 Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramics resonators can also be used for frequency stability. Later versions of microcontrollers operate at a frequency of 0 Hz up to over 50 Hz. Pin 20: GND Ground.
Pin 21-28: Port 2 If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.
Pin 29: (PSEN) If external ROM is used for storing program then logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30: ALE Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external register (usually 74HCT373 or 74HCT375 add-on chip) memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.
Pin 31: (EA) By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).
Pin 32-39: Port 0 Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).
Pin 40: VCC +5V power supply.

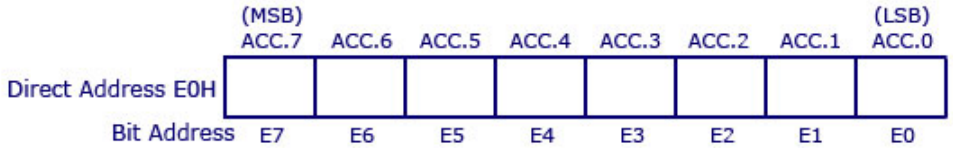
14 Appendix B: 8051 Architecture



15 Appendix C: Memory Organization and SFR

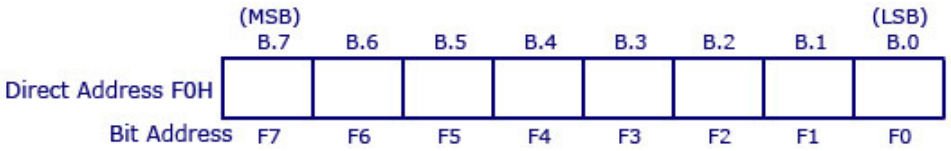


15.1 Register A/Accumulator



The most important of all special function registers is the Accumulator which is also known as ACC or A or Register A. Accumulator holds the result of most of arithmetic and logic operations. ACC is usually accessed by direct addressing and its physical address is E0H. Accumulator is both byte and bit addressable. To access the first bit (i.e bit 0) or to access accumulator as a single byte (all 8 bits at once), you may use the same physical address E0H. Now if you want to access the second bit (i.e bit 1), you may use E1H and for third bit E2H and so on.

15.2 Register B



The major purpose of register B is in only multiplication and division operations. The 8051 micro controller has a single instruction for multiplication (MUL) and division (DIV). Register B is also byte addressable and bit addressable. To access bit 0 or to access all 8 bits (as a single byte), physical address F0 is used. To access bit 1 you may use F1 and so on.

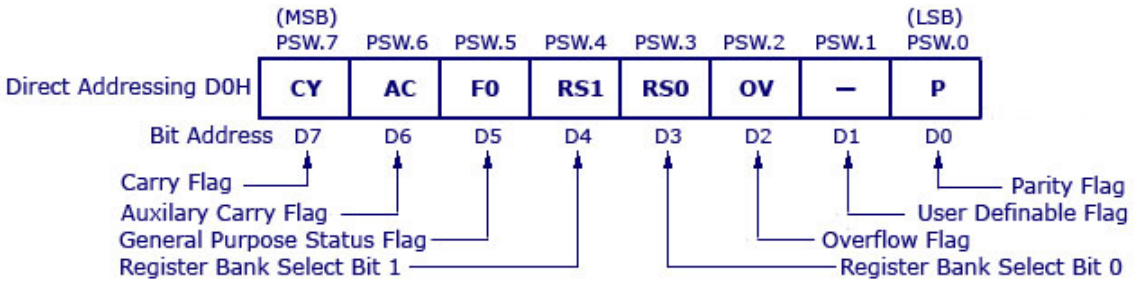
15.3 Stack Pointer - SP

Stack pointer represents a pointer to the the system stack. Stack pointer is an 8 bit register, the direct address of SP is 81H and it is only byte addressable, which means you cant access individual bits of stack pointer. The content of the stack pointer points to the last stored location of system stack or SP always contains top of the stack. To store something new in system stack, the SP automatically incremented by 1 first and then execute the "store" command and to retrieve the content from the stack it reads first and decrement SP by 1. Usually after a system reset SP is initialized as 07H and data can be stored to stack from 08H onwards. This is usually a default case and programmer can alter values of SP to suit his needs.

15.4 Data Pointer - DPTR

The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and can also be accessed as two 8-bit registers as DPL and DPH.

15.5 Program Status Word-PSW



Bit7 CY (Carry Flag) :
This flag is set whenever there is a carry out from the D.7 bit.
This flag bit is affected after 8 bit addition or subtraction.
It can also be set to 1 or 0 directly by an instruction such as " SET BC" and "CLRC".

Bit6 AC(Auxiliary flag):
If there is a carry from D3 to D4 during ADD or SUB operation this bit is set otherwise it is cleared. This flag is used by instruction that perform BCD (binary coded decimal) arithmetic.

Bit5 F0 User Defined Flag:
This flag can be used by the user to set or reset as a status of some operation.

Bit4 and Bit3 (Register bank Select):

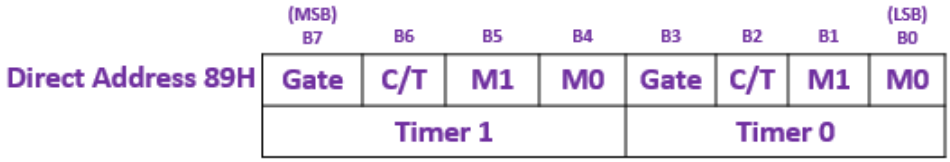
RS1	RS0	Register Bank
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Bit2 OV (Overflow Flag):
This flag is set whenever the result of a signed number operation is too large causing the high order bit to overflow into the sign bit.
The carry flag is used to detect error in unsigned arithmetic operations.
The overflow flag is only used to detect error in signed arithmetic operations.

Bit1 Not Used

Bit0 P (Parity Flag):
The parity flag reflect the number of 1's in the A (accumulator) register only.
If the A register contains an odd number of 1's then p=1 therefore p=0 if A has an even number 1's.

15.6 TMOD Register



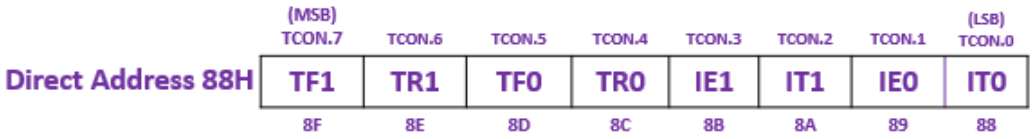
Bit7 and 3 - GATE
OR gate enable bit which controls RUN/STOP of timer 1/0. Set to 1 by program to enable timer to run if bit TR1/TR0 in TCON is set and signal on external interrupt INT1/INT0 is high. Cleared to 0 by program to enable time to run if bit TR1/TR0 is set.

Bit6 and 2 - C/T
Set to 1 by program to make timer1/0 act as a counter by counting pulses from external input pins 3.5/3.4. Cleared to zero by program to make timer act as a timer1/0 by counting internal frequency.

Bit5,4 M1,M0 and Bit1,0 M1,M0
Timer/counter operating mode select, the below table shows the various timer/counter modes.

M1	M0	Operating Mode
0	0	Mode 0: 13-bit timer. The THX register as an 8-bit counter and the TLX as a 5-bit counter.
0	1	Mode 1: 16-bit timer. The THX register as an 8-bit counter and the TLX as an 8-bit counter
1	0	Mode 2: 8-bit Auto reload. Use only the TLX register as an 8-bit counter and THX register is loaded with initial value and THX is given a copy of it.
1	1	Mode 3: Split timer, Timer 0 in mode 3 becomes two separate 8-bit counters. Timer 1 in mode 3 may still be used, but will generate no interrupts.

15.7 TCON Register



Bit7-TF1
Timer 1 Overflow flag. Set when timer rolls from all 1's to 0. Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.

Bit6-TR1
Timer 1 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer.

Bit5-TF0
Timer 0 Overflow flag. Set when timer rolls from all 1's to 0. Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.

Bit4-TR0
Timer 0 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer.

Bit3-IE1
External interrupt 1 Edge flag. Set to 1 when a high-to-low edge signal is received on port 3.3 (INT1). Cleared when processor vectors to interrupt service routine at program address 0013h. Not related to timer operations.

Bit2-IT1
External interrupt 1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low-level signal on external interrupt 1 to generate an interrupt.

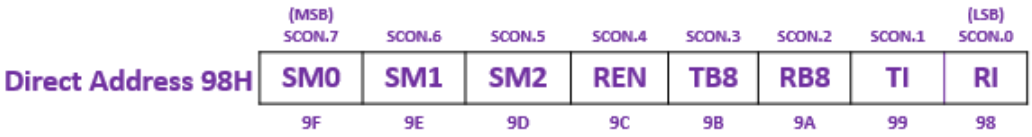
Bit1-IE0
External interrupt 0 Edge flag. Set to 1 when a high-to-low edge signal is received on port 3.2 (INT0). Cleared when processor vectors to interrupt service routine at program address 0003h. Not related to timer operations.

Bit0-IT0
External interrupt 0 signal type control bit. Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal. Set to 0 by program to enable a low-level signal on external interrupt 0 to generate an interrupt.

15.8 Serial Buffer Register - SBUF

Its an 8 bit register used solely for serial communication in 8051. For a byte of data to be transferred via TxD line, it muse be placed in SBUF register. Similarly SBUF register holds the serially inputed data received by TxD line of 8051.

15.9 SCON Register



Bit7,6-Serial Mode Specifier

SM0 SM1	Operating Mode	Description	Baud Rate
0 0	0	Shift Register	Fsoc/12
0 1	1	8-bit UART, 1 Start, 1 Stop bit	Variable Timer 1
1 0	2	9-bit UATR	Fsoc/12 or Fsoc/32
1 1	3	9-bit UATR	Variable

bit5-SM2: Enables multiprocessor communication in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2=1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.

bit4-REN: Receive enable. When high or 1 it allows 8051 to receive data from RxD pin. Used or access as SET SCON.4 and CLR SCON.4. very useful in blocking external serial reception.

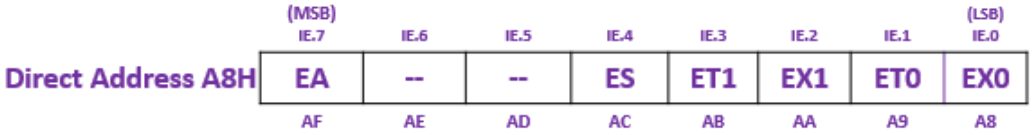
bit3-TB8: Transfer bit8. Used for serial mode 2 and 3 not generally used so set it always to 0.

bit2-RB8: Receive bit8. This is the 9th bit received in modes 2 .and 3. If SM2=0, RB8 is the stop bit.

bit1-TI: Transmit interrupt. When 8051 finishes transfer of 8 bit character, it raises the T1 flag to indicate that it is ready to transfer another byte. Is used at beginning of stop bit.

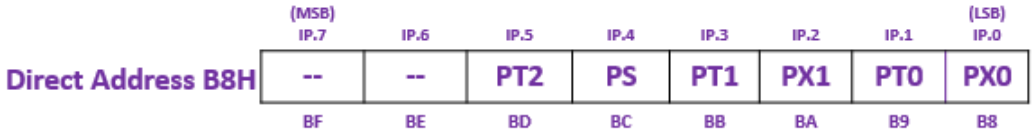
bit0-RI: Receive interrupt. When 8051 finishes receiving data i.e when data is successfully stored in SBUF it raises R1 flag to indicate byte is received and to be picked before it gets lost.

15.10 Interrupt Enable Register - IE



Bit7 - EA: It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
Bit6 : Reserved for future use.
Bit5 : Reserved for future use.
Bit4 - ES: Enables/disables serial port interrupt.
Bit3 - ET1: Enables/disables timer1 overflow interrupt.
Bit2 - EX1: Enables/disables external interrupt1.
Bit1 - ET0: Enables/disables timer0 overflow interrupt.
Bit1 - EX0: Enables/disables external interrupt0.

15.11 Interrupt Priority Register - IP



Bit7: Reserved for future use.
Bit6: Reserved for future use.
Bit5 - PT2: Interrupt for timer 2 (only in 8052 MC).
Bit4 - PS: It defines the serial port interrupt priority level.
Bit3 - PT1: It defines the timer interrupt of 1 priority.
Bit2 - PX1: It defines the external interrupt priority level.
Bit1 - PT0: It defines the timer0 interrupt priority level.
Bit0 - PX0: It defines the external interrupt of 0 priority level.

15.12 PCON Register

Bit7 - SMOD: Serial Comm. Baud Rate Modify Bit. If 1, doubles the baud rate using Timer 1. If 0, normal timer 1 baud rate.
Bit6,5,4: Reserved for future use.
Bit3 - GF1: General Purpose User Flag (Bit 1).
Bit2 - GF0: General Purpose User Flag (Bit 0).
Bit1 - PD: Power Down Bit. To enter Power Down Mode, set to 1.
Bit0 - IDL: Idle Mode Bit. To enter Idle Down Mode, set to 1.

16 Appendix D: Instruction Set

Arithmetic Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ADD A,Rn	A=A +Rn	1	1
ADD A,direct	A = A + (direct)	2	1
ADD A, @Ri	A= A + @Ri	1	1
ADD A,# data	A = A + data	2	1
ADDC A, Rn	A = A + Rn + C	1	1
ADDC A, direct	A= A + (direct) + C	2	1
ADDC A, @Ri	A = A + @Ri + C	1	1
ADDC A,# data	A= A + data + C	2	1
DA A	Decimal adjust Acc.	1	1
DIV AB	Divide A by B, A=quotient, B=rem	1	4
DEC A	A = A -	1	1
DEC Rn	Rn = Rn - 1	1	1
DEC direct	(direct) (direct) - 1	2	1
DEC @Ri	@Ri = @Ri - 1	1	1
INC A	A= A+1	1	1
INC Rn	Rn = Rn +1	1	1
INC direct	(direct) (direct) + 1	2	1
INC @Ri	@Ri= @Ri +1	1	1
INC DPTR	DPTR= DPTR +1	1	2
MUL AB	Multiply A by B, A=L byte, B =H byte	1	4
SUBB A, Rn	A = A - Rn - C	1	1
SUBB A, direct	A=A - (direct) - C	2	1
SUBB A, @Ri	A = A- @Ri - C	1	1
SUBB A,# data	A = A -data - C	2	1

Logical Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ANL A, Rn	A= A AND Rn	1	1
ANL A, direct	A = A AND (direct)	2	1
ANL A, @Ri	A= A AND @Ri	1	1
ANL A,# data	A = A AND data	2	1
ANL direct, A	(direct) = (direct) AND A	2	1
ANL direct,# data	(direct)= (direct) AND data	3	2
CLR A	A= 00H	1	1
CPL A	A= A'	1	1
ORL A, Rn	A= A OR Rn	1	1
ORL A, direct	A=A OR (direct)	1	1
ORL A, @Ri	A=A OR @Ri	2	1
ORL A,# data	A=A OR data	1	1
ORL direct, A	(direct)=(direct) OR A	2	1
ORL direct,# data	(direct)=(direct) OR data	3	2
RL A	Rotate acc. left	1	1
RLC A	Rotate acc. left through carry	1	1
RR A	Rotate acc. right	1	1
RRC A	Rotate acc. right through carry	1	1
SWAP A	Swap nibbles within Acc	1	1
XRL A, Rn	A=A EXOR Rn	1	1
XRL A, direct	A=A EXOR (direct)	1	1
XRL A, @Ri	A=A EXOR @Ri	2	1
XRL A,# data	A=A EXOR data	1	1
XRL direct, A	(direct)=(direct) EXOR A	2	1
XRL direct,# data	(direct)=(direct) EXOR data	3	2

Data Transfer Instructions

Mnemonics	Description	Bytes	Instruction Cycles
MOV A, Rn	A= Rn	1	1
MOV A, direct	A = (direct)	2	1
MOV A, @Ri	A = @Ri	1	1
MOV A,# data	A = data	2	1
MOV Rn, A	Rn= A	1	1
MOV Rn, direct	Rn = (direct)	2	2
MOV Rn,# data	Rn= data	2	1
MOV direct, A	(direct)= A	2	1
MOV direct, Rn	(direct) =Rn	2	2
MOV direct1, direct2	(direct1) = (direct2)	3	2
MOV direct, @Ri	(direct)= @Ri	2	2
MOV direct,# data	(direct)=# data	3	2
MOV @Ri, A	@Ri= A	1	1
MOV @Ri, direct	@Ri=(direct)	2	2
MOV @Ri,# data	@Ri = data	2	1
MOV DPTR,# data16	DPTR data16	3	2
MOVC A, @A+DPTR	A Code byte pointed by A + DPTR	1	2
MOVC A, @A+PC	A Code byte pointed by A + PC	1	2
MOVC A, @Ri	A Code byte pointed by Ri 8-bit address	1	2
MOVX A, @DPTR	A External data pointed by DPTR	1	2
MOVX @Ri, A	@Ri A (External data-8bit address)	1	2
MOVX @DPTR, A	@DPTR A(External data-16bit address)	1	2
PUSH direct	(SP)=(direct)	2	2
POP direct	(direct)=(SP)	2	2
XCH Rn	Exchange A with Rn	1	1

XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Boolean Variable Instructions

Mnemonics	Description	Bytes	Instruction Cycles
CLR C	C= 0	1	1
CLR bit	bit = 0	2	1
SETB C	C = 1	1	1
SETB bit	bit = 1	2	1
CPL C	C=C'	1	1
CPL bit	bit=bit'	2	1
ANL C,/bit	C =C.bit'	2	1
ANL C,bit	C =C.bit	2	1
ORL C,/bit	C = C + bit'	2	1
ORL C,bit	C= C + bit	2	1
MOV C,bit	C = bit	2	1
MOV bit,C	bit = C	2	2

Program Branching Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ACALL addr11	PC + 2=(SP) ; addr 11 PC	2	2
AJMP addr11	Addr11=PC	2	2
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A,# data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn,# data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2

CJNE @Ri,# data, rel	Compare with @Ri A, jump (PC + rel) if not equal	3	2
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2
JMP @A+DPTR	A+DPTR = PC	1	2
JZ rel	If A=0, jump to PC + rel	2	2
JNZ rel	If A not equal to 0 , jump to PC + rel	2	2
LCALL addr16	PC + 3=(SP), addr16= PC	3	2
LJMP addr 16	Addr16 PC	3	2
NOP	No operation	1	1
RET	(SP) PC	1	2
RETI	(SP) PC, Enable Interrupt	1	2
SJMP rel	PC + 2 + rel PC	2	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A = 0. jump PC+ rel	2	2
JNZ rel	If A not equal to 0, jump PC + rel	2	2