Familiarization with 8051/8052 Microcontroller

Lab Exercises on September 25, 2020

Department of Electronics and Computer Engineering
Pulchowk Campus, Lalitpur

Ashlesh Pandey

PUL074BEX007

Contents

List of Figures

Source Codes

1 Introduction

1.1 Microcontroller

A microcontroller is an electronic device that consists of all the necessary parts viz. central processing unit, I/O ports, timers, counters, clocks, memory units and registers embedded on a single chip. It is simply a computer capable of performing a specific task and hence are termed as special purpose computers. Generally having small size and low cost, a microcontroller is specifically designed for implementations in embedded systems.

1.2 MCS-51 Family Microcontroller Chips

The MCS-51 microcontroller chips, originally developed by Intel for use in small-scale embedded systems is based on the Harvard architecture of designing. Initial batches of the chips were designed using the N-type metal-oxide-semiconductors which were later replaced by a more generic MOS, the CMOS (Complementary metal-oxide-semiconductor) hence giving the later versions of the chip the identity of 80C51. Current versions of the microcontroller have clock frequencies of up to 100 MHz, which is a drastic improvement from their original 12 MHz clock frequency.

The MCS-51 microcontroller was first introduced by Intel in 1980. Since then, MCS-51, more commonly termed as the 8051 microcontroller has been adapted by various vendors like Silicon Laboratories, ASTX, Dallas Semiconductors, Texas Instruments, Atmel and many more for their wide usages in embedded systems.

The different features of the 8051 microcontroller include:

- 8-bit ALU, Accumulator and Registers, making it an 8-bit microcontroller.
- 8-bit data bus meaning that it can access 8 bits of data in one operation.
- 4KB of ROM for the programs, also called program memory.
- 128 Bytes of RAM for the variables, also called data memory.
- 32 I/O lines, i.e. 4 ports with 8 lines each.
- 16-bit address bus meaning that it can access 65536 locations of RAM and ROM.
- 2 16-bit timers/counters.
- 1 full-duplex serial port for serial communication (UART).
- 6 interrupt sources(2 external interrupts, 2 timer interrupts & 2 serial interrupts).

1.2.1 Memory Architecture

The 8051 microcontroller has four different types of memory specifications, viz. Internal RAM, Program Memory, External Data Memory, and Special Function Registers.

Internal RAM

The Internal RAM, or generally referred to as the IRAM has an 8-bit address space taking up the addresses from 0x00 to 0xFF. IRAM from 0x00 to 0x7F can be accessed directly whereas the remaining must be accessed indirectly using the indirect addressing from registers R0 or R1 as @R0 or @R1. The original 8051 microcontroller has 128 bytes of internal RAM whereas some later versions may include 256 bytes of IRAM.

Program Memory

Program memory, referred as PMEM is up to 64 KB of read-only memory, starting at address 0 in a separate address space. Both on and off chip versioned microcontroller are available in the market. PMEM isn't used as much as IRAM and External RAM.
In addition to program code, lookup tables can be stored in the PMEM and retrieved with the MOVC A, @A+DPTR or MOVC A,@A+PC instructions such that the address is calculated with the summation of 8-bit accumulator and 16-bit PC or DPTR.

External Data Memory

XRAM is a third address space memory space starting at address 0 with 16-bit address space. It is accessed using the MOVX instruction even though it can be on or off chip.The full 64 KB XRAM can be accessed using MOVX A,@DPTR and MOVX @DPTR,A.

Special Function Registers

SFR are located at the same address as IRAM i.e. at 0x80 to 0xFF and accessed just as lower half of IRAM. SFR can only be accessed directly since indirect addressing will refer to the remaining half of IRAM. SFRs with addresses multiple of 8 are bit-addressable.

1.2.2 Registers

Program Counter is the only register in 8051 that isn't memory mapped. Eight 8-bit general purpose registers(R0-R7) are mapped to IRAM between 0x00 and 0x1F. Only eight bytes of that range are used at any given time, determined by the two bank select bits in the PSW(RS0 & RS1). A 8-bit Stack Pointer(SP), 16-bit Data Pointer(DP) and 8-bit Program Status Word(PSW) are also present in the 8051 microcontroller. The PSW doesn't consist of Negative or Zero flags. Parity flag (PSW.0), Overflow(PSW.2), Auxiliary Carry(PSW.6) and Carry(PSW.7) are the other flags of the PSW than the RS0(PSW.3), RS1(PSW.4), User defined(PSW.1) and Flag 0(PSW.5). Accumulator(0xE0) is used by most instructions for storing immediate results and the B register extends the accumulator for multiplication and division routines.

1.2.3 Programming

Assembly Level Programming

Pseudo-English representations of the machine languages, or more generally known as Mnemonics are used along side hexadecimal codes to write assembly language codes for the 8051 microcontroller. It is generally written in human understandable mnemonics rather than the actual machine level codes of 0s and 1s. However, it is still a low level programming language and extensive understanding of the architecture is essential. Programs in assembly language are executed faster and occupy less memory. Maximum features of the microcontroller can be used with the help of assembly code and it provides direct and accurate control of all the resources such as I/O ports, RAM, SFRs, etc.

High Level Programming

Various high-level programming languages have different compilers for the MCS-51 family. The most generic of these languages is the embedded C which allows the programmer to specify where variables are stored in the different memory architecture. Since data can be present in one of three memory spaces, IRAM, XRAM and PMEM(read-only), and all these have an address 0, the C compilers have mechanism to determine the memory pointed by either constraining the pointer type to include the memory space, or by storing metadata along with the pointer.

Other languages such as C++, Object Pascal, Pascal, BASIC, PL/M, Modula-2 are available for the MCS-51 family but aren't generally in use as compared to the C compilers.

1.2.4 Instruction Sets

The instructions in 8051's assembly level programming are all 1 to 3 bytes long. For 3 byte instructions, the first byte represents the operation code, and the last two represent the operands. For most of the instructions, accumulator is necessary but 8051 is not an accumulator based microcontroller. The various mnemonics along with their functions can be looked up in [1].

2 Objectives

The primary objective of this lab experiment is to understand the various features and architecture of 8051 microcontroller. Familiarization with the 8051/8052 microcontroller will enable us to write assembly language code for the 8051/8052 microcontroller capable of:

- Data manipulation
- Looping and branching techniques
- Arithmetic and logical operations
- Subroutine calls

3 Lab Experiment Environment

The lab experiments will be performed virtually via various simulation softwares. The basic usages of these tools allows us to visualize and determine the different functional units of the 8051 microcontroller to perform simple arithmetic and logical tasks.

3.1 Circuit Simulation

Proteus Design Suite, which is a professional PCB layout, circuit design and simulation tool, will be used to simulate the circuit for 8051 microcontroller alongside various basic electronic components such as resistors, switches, LEDs, etc. Circuit diagram made in Proteus will consist of a AT89C52 microcontroller with 8-bit LED in a pull-up configuration connected to its Port 0. This circuit will be used to confirm the code written in assembly and C languages.

3.2 Code Editor and Compiler

A microcontroller actually only understands hex codes(machine code). Writing codes in the machine level language is something that microcontroller based companies have tried to avoid. ARM Limited is one such company that produces development tools and MDKs along with the hardwares. KEIL products from ARM include C/C++ compilers, debuggers, integrated development and simulation environments, RTOS and middleware libraries, and evaluation boards for ARM, Cortex-M, Cortex-R4, 8051, C166, and 251 processor families. The KEIL μVision IDE will be used for writing and assembling the assembly code as well as compiling Embedded C codes into hex code that will be used by the microcontroller.
We will be utilizing the compiler from KEIL to generate hex codes and hook that up with the circuit simulation in Proteus for code checking and debugging purpose. KEIL itself provides debugging and simulation features for the 8051 microcontroller, and we will also be using that feature to verify our codes.

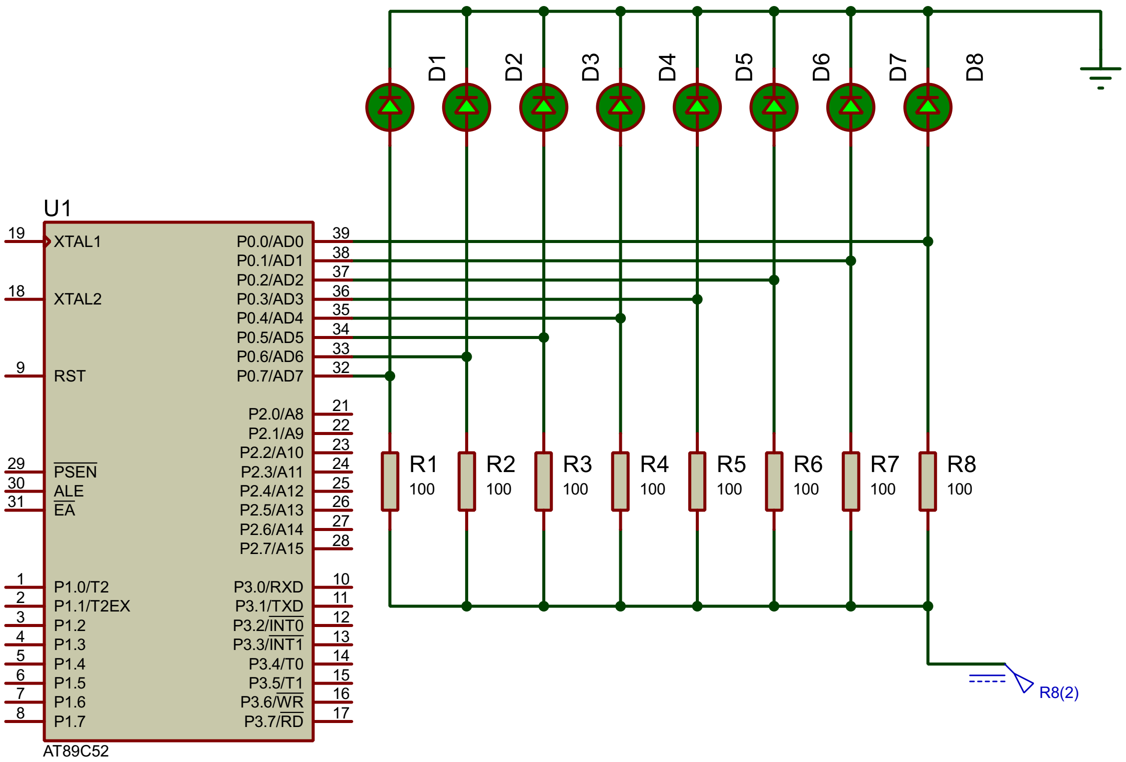


Figure 1: Circuit Diagram for Proteus Simulation

4 Lab Problems

Problem 1

Write code to add the numbers 897F9AH and 34BC48H and save the result in internal RAM starting at 40H. The result should be displayed continuously on the LEDs of the development board starting from least significant byte with an appropriate timing interval between each byte. Use port zero (P0) of the micro-controller to interface with LEDs.

Assembly Code

```
    ORG 00H

    MOV R0,#9AH
    MOV R1,#48H
    MOV R2,#7FH
    MOV R3,#0BCH
    MOV R4,#89H
    MOV R5,#34H

    MOV A,R0
    ADD A,R1
    MOV 40H,A

    MOV A,R2
    ADDC A,R3
    MOV 41H,A

    MOV A,R4
    ADDC A,R5
    MOV 42H,A

    MOV A,#0H
    ADDC A,#0H
    MOV 43H,A

AGAIN:  MOV R1,#04H
    MOV R0,#40H
NEXT: MOV P0,@R0
```

```
    ACALL DELAY
    INC R0
    DJNZ R1,NEXT
    AJMP AGAIN

DELAY:  MOV R4,#7
LOOP1:  MOV R5,#255
LOOP2:  MOV R7,#255
LOOP3:  DJNZ R7,LOOP3
    DJNZ R5,LOOP2
    DJNZ R4,LOOP1
    RET

    END
```

Code 1: Problem 1 - Assembly

Embedded C Code

```c
#include <reg51.h>
unsigned char data d[4] _at_ 0x40;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main(void)
{
  unsigned long a = 0x897f9a;
  unsigned long b = 0x34bc48;
  unsigned long c = a + b;
  unsigned int i;

  for(i = 0; i < 4; i++)
  {
    d[i] = c%0x100;
    c >>= 8;
  }

  while(1)
    for(i = 0; i < 4; i++)
    {
      P0 = d[i];
      delay(1000);
    }
}
```

Code 2: Problem 1 - Embedded C

Problem 2

Implement a subroutine that replaces the SWAP instruction using rotate right instructions. Test your program on the contents of the accumulator when it contains the number 6BH.

Assembly Code

```
ORG 00H

AGAIN:  MOV A,#6BH
    MOV P0,A
    ACALL DELAY
    ACALL SWAP_RR
    MOV P0,A
    ACALL DELAY
    AJMP AGAIN

SWAP_RR:RR A
    RR A
    RR A
    RR A
    RET

DELAY:  MOV R4,#7
LOOP1:  MOV R5,#255
LOOP2:  MOV R7,#255
LOOP3:  DJNZ R7,LOOP3
    DJNZ R5,LOOP2
    DJNZ R4,LOOP1
    RET

    END
```

Code 3: Problem 2 - Assembly

Embedded C Code

```c
#include<reg51.h>

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main()
{
  unsigned char value = 0xb6;
  unsigned char reversevalue;
  unsigned char a,b;
  a=value/0x10;
  b=value%0x10;
  reversevalue = b*(0x10) + a;

  while(1)
  {
    P0 = value;
    delay(1000);
    P0 = reversevalue;
    delay(1000);
  }
}
```

Code 4: Problem 2 - Embedded C

Problem 3
Multiply, by using looping and successive addition technique, the data in RAM location 22H by the data in RAM location 15H and put the result in RAM locations 19H (low byte) and 1AH (high byte). Data in 22H should be FFH and data in 15H should be DEH.

Assembly Code

```asm
    ORG 00H

    MOV 22H,#0FFH
    MOV 15H,#0DEH

    MOV A,#0H
    MOV R1,#0H

    MOV R0,22H
AGAIN:  ADD A,15H
    JNC SKIP
    INC R1
SKIP: DJNZ R0,AGAIN

    MOV 19H,A
    MOV 1AH,R1

LOOP: MOV P0,A
    ACALL DELAY
    MOV P0,R1
    ACALL DELAY
    AJMP LOOP

DELAY:  MOV R4,#7
LOOP1:  MOV R5,#255
LOOP2:  MOV R7,#255
LOOP3:  DJNZ R7,LOOP3
    DJNZ R5,LOOP2
    DJNZ R4,LOOP1
    RET

    END

```

Code 5: Problem 3 - Assembly

Embedded C Code

```c
#include <reg51.h>
unsigned char data multiplicand _at_ 0x22;
unsigned char data multiplier _at_ 0x15;
unsigned char data answer[2] _at_ 0x19;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main(void)
{
  unsigned int result = 0x0;
  unsigned char i;

  multiplicand = 0xff;
  multiplier = 0xde;

  for(i=0x0;i<multiplier;i++)
    result += multiplicand;

  answer[0] = result%0x100;
  result >>= 8;
  answer[1] = result%0x100;

  while(1)
  {
    P0 = answer[0];
    delay(1000);
    P0 = answer[1];
    delay(1000);
  }
}
```

Code 6: Problem 3 - Embedded C

Problem 4
Divide, by using looping and successive subtraction technique, the data in RAM location 3EH by the number 12H; put the quotient in R4 and remainder in R5. Data in 3EH should be AFH.

Assembly Code

```asm
    ORG 00H

    MOV 3EH,#0AFH

    MOV A,3EH
    MOV R4,#0H

AGAIN:  SUBB A,#12H
    JC DONE
    INC R4
    AJMP AGAIN
DONE: ADD A,#12H
    MOV R5,A

LOOP: MOV P0,R4
    ACALL DELAY
    MOV P0,R5
    ACALL DELAY
    AJMP LOOP

DELAY:  MOV R1,#7
LOOP1:  MOV R2,#255
LOOP2:  MOV R3,#255
LOOP3:  DJNZ R3,LOOP3
    DJNZ R2,LOOP2
    DJNZ R1,LOOP1
    RET

    END

```

Code 7: Problem 4 - Assembly

Embedded C Code

```c
#include <reg51.h>
int data dividend _at_ 0x3e;
unsigned char data reg4 _at_ 0x04;
unsigned char data reg5 _at_ 0x05;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main(void)
{
  unsigned char divisor = 0x12;
  unsigned char quotient = 0x00, remainder;

  dividend = 0x00af;

  while(1)
  {
    dividend -= divisor;
    if(dividend < 0x0)
      break;
    quotient += 0x1;
  }
  remainder = dividend + divisor;

  reg4 = quotient;
  reg5 = remainder;

  while(1)
  {
    P0 = quotient;
    delay(1000);
    P0 = remainder;
    delay(1000);
  }
}
```

Code 8: Problem 4 - Embedded C

Problem 5
Store ten hexadecimal numbers in internal RAM starting from memory location 50H. The list of numbers to be used is: D6H, F2H, E4H, A8H, CEH, B9H, FAH, AEH, BAH, CCH. Implement a subroutine that extracts both the smallest and largest numbers from the stored numbers.

Assembly Code

```
    ORG 00H

    MOV 50H,#0D6H
    MOV 51H,#0F2H
    MOV 52H,#0E4H
    MOV 53H,#0A8H
    MOV 54H,#0CEH
    MOV 55H,#0B9H
    MOV 56H,#0FAH
    MOV 57H,#0AEH
    MOV 58H,#0BAH
    MOV 59H,#0CCH

    MOV R0,#50H

    MOV A,@R0
    MOV R7,A
    MOV R1,A

    MOV R2,#09H

NEXT: INC R0
    MOV A,R7
    SUBB A,@R0
    JNC NSMALL
    MOV A,@R0
    MOV R7,A
NSMALL: MOV A,R1
    SUBB A,@R0
    JC NLARGE
    MOV A,@R0
    MOV R1,A
NLARGE: DJNZ R2,NEXT

LOOP: MOV P0,R7
    ACALL DELAY
    MOV P0,R1
    ACALL DELAY
    AJMP LOOP

DELAY:  MOV R3,#7
LOOP1:  MOV R4,#255
LOOP2:  MOV R5,#255
LOOP3:  DJNZ R5,LOOP3
    DJNZ R4,LOOP2
    DJNZ R3,LOOP1
    RET

    END

```

Code 9: Problem 5 - Assembly

Embedded C Code

```c
#include <reg51.h>
unsigned char data d[10] _at_ 0x50;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main(void)
{
  unsigned char smallest, largest;
  unsigned char i;

  d[0] = 0xd6; d[1] = 0xf2; d[2] = 0xe4;
  d[3] = 0xa8; d[4] = 0xce; d[5] = 0xb9;
  d[6] = 0xfa; d[7] = 0xae; d[8] = 0xba;
  d[9] = 0xcc;

  smallest = largest = d[0];
  for(i=1;i<10;i++)
  {
    if(d[i] < smallest)
      smallest = d[i];
    if(d[i] > largest)
      largest = d[i];
  }

  while(1)
  {
    P0 = smallest;
    delay(1000);
    P0 = largest;
    delay(1000);
  }
}
```

Code 10: Problem 5 - Embedded C

Problem 6

Store ten hexadecimal numbers in internal RAM starting from memory location 60H. The list of numbers to be used is: A5H, FDH, 67H, 42H, DFH, 9AH, 84H, 1BH, C7H, 31H. Implement a subroutine that orders the numbers in ascending order using bubble or any other sort algorithm and implement a subroutine that order the numbers in descending order using selection sort algorithm.

Part I

Assembly Code

```
    ORG 00H

    MOV 60H,#0A5H
    MOV 61H,#0FDH
    MOV 62H,#67H
    MOV 63H,#42H
    MOV 64H,#0DFH
    MOV 65H,#9AH
    MOV 66H,#84H
    MOV 67H,#1BH
    MOV 68H,#0C7H
    MOV 69H,#31H

    MOV R1,#09H
AGN2: MOV A,R1
    MOV R2,A

    MOV R0,#60H
    MOV A,@R0

AGN1: INC R0
    MOV R3,A
    MOV A,@R0
    MOV R4,A

    MOV A,R3
    SUBB A,R4
    JC SKIP

    MOV A,R3
    MOV @R0,A
    MOV A,R4
    DEC R0
    MOV @R0,A
    INC R0

SKIP: MOV A,@R0
    DJNZ R2,AGN1
    DJNZ R1,AGN2

REP:  MOV R1,#0AH
    MOV R0,#60H
LOOP: MOV A,@R0
    MOV P0,A
    ACALL DELAY
    INC R0
    DJNZ R1,LOOP
    AJMP REP

DELAY:  MOV R3,#7
LOOP1:  MOV R4,#255
LOOP2:  MOV R5,#255
LOOP3:  DJNZ R5,LOOP3
    DJNZ R4,LOOP2
    DJNZ R3,LOOP1
    RET

    END
```

Code 11: Problem 6 - Assembly

Embedded C Code

```
#include <reg51.h>
unsigned char data a[10] _at_ 0x60;
void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}
```

```c

void main(void)
{
  unsigned char i, j, temp;
  a[0] = 0xa5; a[1] = 0xfd; a[2] = 0x67;
  a[3] = 0x42; a[4] = 0xdf; a[5] = 0x9a;
  a[6] = 0x84; a[7] = 0x1b; a[8] = 0xc7;
  a[9] = 0x31;

  for(i=0;i<10;i++)
    for(j=0;j<i;j++)
      if(a[j] > a[i])
      {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }

  while(1)
  {
    for( i = 0;i<10;i++)
    {
      P0 = a[i];
      delay(1000);
    }
  }
}
```

Code 12: Problem 6 - Embedded C

Part II

Assembly Code

```asm
    ORG 00H

; DATA FROM QUESTION
    MOV 60H,#0A5H
    MOV 61H,#0FDH
    MOV 62H,#67H
    MOV 63H,#42H
    MOV 64H,#0DFH
    MOV 65H,#9AH
    MOV 66H,#84H
    MOV 67H,#1BH
    MOV 68H,#0C7H
    MOV 69H,#31H

; MAIN ROUTINE FOR SORTING
    MOV R0,#60H
    MOV R6,#09H
AGN:  ACALL F_LARGE
    MOV @R0,A
    INC R0
    DJNZ R6,AGN

; DISPLAYING SORTED LIST
AGAIN:  MOV R1,#0AH
    MOV R0,#60H
LOOP: MOV A,@R0
    MOV P0,A
    ACALL DELAY
    INC R0
    DJNZ R1,LOOP
    AJMP AGAIN

; SELECTION SORT ALGORITHM
F_LARGE:MOV B,R0
    MOV A,R6
    MOV R2,A

    MOV A,@R0
    MOV R1,A

NEXT: INC R0
    MOV R4,A
    SUBB A,@R0
    JNC SKIP

    MOV A,@R0
    MOV R1,A
    MOV A,R4
    MOV @R0,A

SKIP: MOV A,R1
    DJNZ R2,NEXT
    MOV R0,B
    RET

DELAY:  MOV R3,#7
LOOP1:  MOV R4,#255
LOOP2:  MOV R5,#255
LOOP3:  DJNZ R5,LOOP3
    DJNZ R4,LOOP2
    DJNZ R3,LOOP1
    RET

    END
```

Code 13: Problem 6 - Assembly

Embedded C Code

```c
#include <reg51.h>
unsigned char data a[10] _at_ 0x60;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main(void)
{
  unsigned char i, j, temp;
  unsigned char largest = a[0];

  a[0] = 0xa5; a[1] = 0xfd; a[2] = 0x67;
  a[3] = 0x42; a[4] = 0xdf; a[5] = 0x9a;
  a[6] = 0x84; a[7] = 0x1b; a[8] = 0xc7;
  a[9] = 0x31;

  for(i=0;i<10;i++)
  {
    for(j=i;j<10;j++)
      if(a[j] > a[i])
      {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
      }

  }

  while(1)
  {
    for( i = 0;i<10;i++)
    {
      P0 = a[i];
      delay(1000);
    }
  }
}
```

Code 14: Problem 6 - Embedded C

Problem 7

Store numbers from 00H to 20H in internal RAM starting from memory location 40H. Implement a subroutine that extracts only the prime numbers.

Assembly Code

```
    ORG 00H

    MOV R0,#40H
    MOV A,#00H
AGAIN:  MOV @R0,A
    INC A
    INC R0
    MOV R1,A
    SUBB A,#20H
    JZ DONE
    MOV A,R1
    AJMP AGAIN

DONE: MOV A,42H
    MOV P0,A
    ACALL DELAY
    MOV A,43H
    MOV P0,A
    ACALL DELAY

    MOV R0,#44H
    MOV R1,#1DH
NEXT: ACALL PRIME
    INC R0
    DJNZ R1,NEXT
    AJMP DONE

PRIME:  MOV A,@R0
    MOV R4,A

    MOV R2,#02H
INC_B:  MOV A,R4
    MOV B,R2
    DIV AB

    MOV A,B

    JNZ N_RET
    RET
N_RET:  INC R2
    MOV A,R2
    SUBB A,@R0
    JNZ INC_B
    MOV A,R4
    MOV P0,A
    ACALL DELAY
    RET

DELAY:  MOV R7,#7
LOOP1:  MOV R6,#255
LOOP2:  MOV R5,#255
LOOP3:  DJNZ R5,LOOP3
    DJNZ R6,LOOP2
    DJNZ R7,LOOP1
    RET

    END
```

Code 15: Problem 7 - Assembly

Embedded C Code

```c
#include <reg51.h>
unsigned char data d[21] _at_ 0x40;

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

int isprime(unsigned char val)
{
  unsigned char j;
  for(j=0x2;j<val;j++)
    if(val % j == 0x0)
        break;
  if(j==val)
      return 1;
  return 0;
}


void main(void)
{
  unsigned char a[20];
  unsigned char i, count=0;
  for(i = 0x0; i<0x21; i++)
    d[i] = i;

  a[count++] = 0x2;

  for(i=0x3;i<0x21;i++)
  {
    if(isprime(d[i]))
      a[count++] = d[i];
  }

  while(1)
  {
    for(i = 0;i<count;i++)
    {
      P0 = a[i];
      delay(1000);
    }
  }
}
```

Code 16: Problem 7 - Embedded C

Problem 8

Find the factorial of a number stored in R3. The value in R3 could be any number in the range from 00H to 05H. Implement a subroutine that calculates the factorial. The factorial needs to be represented in both hexadecimal and decimal formats.

Assembly Code

```asm
    ORG 00H

    MOV R3,#04H

    MOV B,R3
    MOV R1,B

    ACALL FACTORIAL

    MOV R1,A
AGAIN:  MOV A,R1
    MOV P0,A
    ACALL DELAY

    ACALL HTOD
    MOV P0,A
    ACALL DELAY

    MOV A,B
    MOV P0,A
    ACALL DELAY
    SJMP AGAIN

HTOD: MOV R4,#00H
    MOV B,#0AH
    DIV AB
    MOV R2,A
    SUBB A,#0AH
    JC SKIP
    MOV A,R2
    MOV R3,B
    MOV B,#0AH
    DIV AB
    MOV R4,A
    MOV P0,A
    MOV A,B
    MOV B,R3
    MOV R2,A
SKIP: MOV A,R2
    SWAP A
    ADD A,B
    MOV B,R4
    RET

DELAY:  MOV R7,#7
HERE1:  MOV R6,#255
HERE2:  MOV R5,#255
HERE3:  DJNZ R5,HERE3
    DJNZ R6,HERE2
    DJNZ R7,HERE1
    RET

FACTORIAL:  MOV A,#01H
LOOP: MOV B,R1
    MUL AB
    DJNZ R1,LOOP
    RET
    END

```

Code 17: Problem 8 - Assembly

Embedded C Code

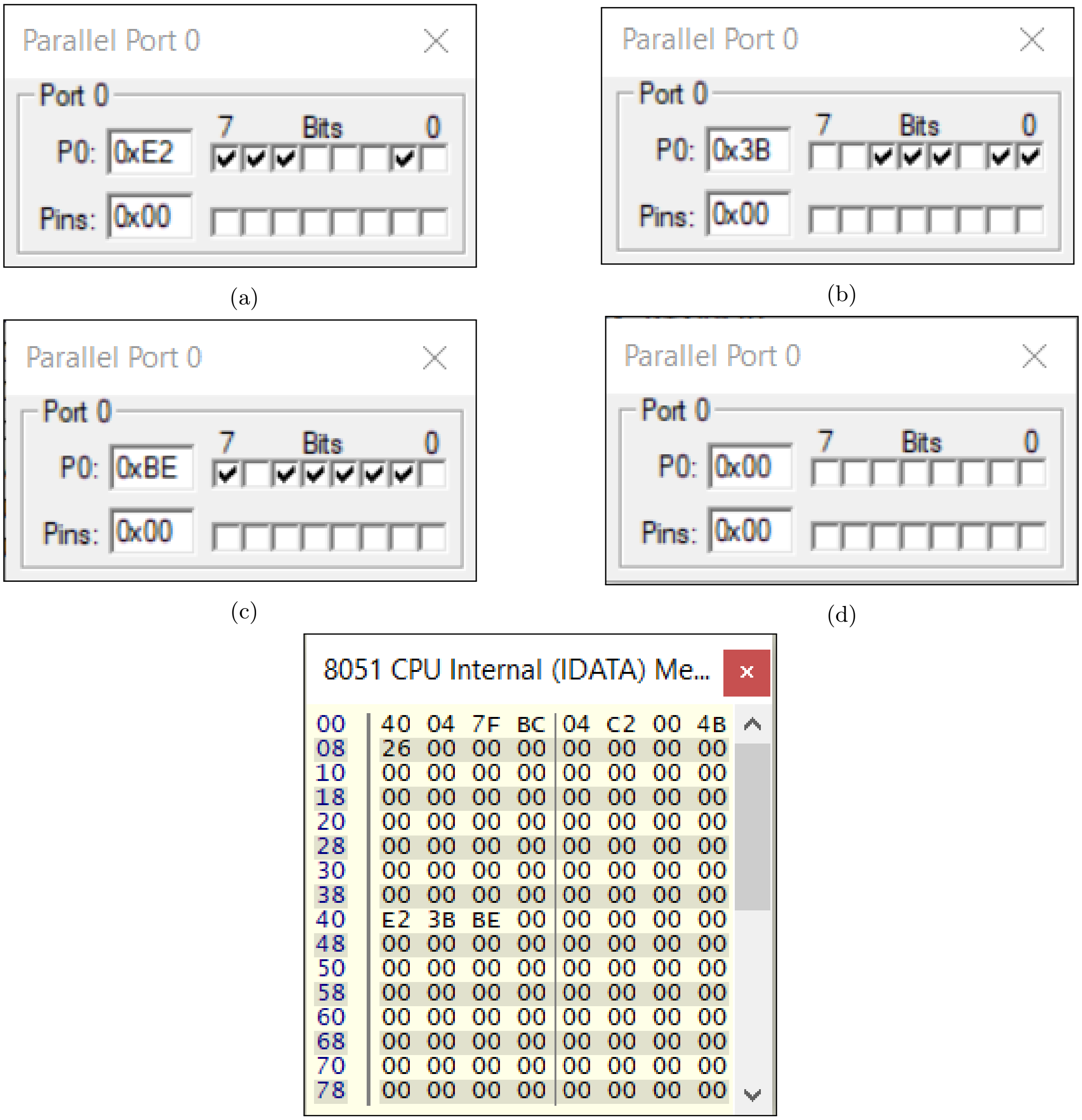
```
#include<reg51.h>

void delay(int time)
{
  unsigned int i,j;
  for (i=0;i<time;i++)
    for (j=0;j<125;j++);
}

void main()
{
  unsigned int a = 0x4;
  unsigned int fact = 0x1;
  unsigned char i;
  unsigned char x, d1, d2, d3;

  for(i = 0x1;i<=a;i++)
    fact *=i;

  x = fact / 0xa;
  d1 = fact % 0xa;
  d2 = x % 0xa;
  d3 = x / 0xa;
  while(1)
  {
    P0 = fact;
    delay(1000);
    P0 = d1;
    delay(1000);
    P0 = d2;
    delay(1000);
    P0 = d3;
    delay(1000);
  }
}
```

Code 18: Problem 8 - Embedded C

5 Observations

The observations for all the lab problems are presented in this section. Port 0 display values are snipped from KEIL μVision IDE in debug mode with appropriate breakpoints. The internal RAM and register values are snipped from the Proteus simulation during the VSM debugging. These values simulate the hardware for 8051 MCU so slight variation from the actual states may be visible. Since the lab experiments are performed in simulated environment, the following observations are chosen such that higher accuracy in data visualization for 8051 registers, ports and IRAM can be made. Circuit behaviors from Proteus simulation aren't included in this report, however the observations are clear enough to make conclusions for the lab.

Problem 1



Parallel Port 0

Port 0

P0: 0xE2 7 Bits 0 ☑☑☑☐☐☐☑☐

Pins: 0x00 ☐☐☐☐☐☐☐☐

(a)

Parallel Port 0

Port 0

P0: 0x3B 7 Bits 0 ☐☐☑☑☑☐☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(b)

Parallel Port 0

Port 0

P0: 0xBE 7 Bits 0 ☑☐☑☑☑☑☑☐

Pins: 0x00 ☐☐☐☐☐☐☐☐

(c)

Parallel Port 0

Port 0

P0: 0x00 7 Bits 0 ☐☐☐☐☐☐☐☐

Pins: 0x00 ☐☐☐☐☐☐☐☐

(d)

8051 CPU Internal (IDATA) Me...

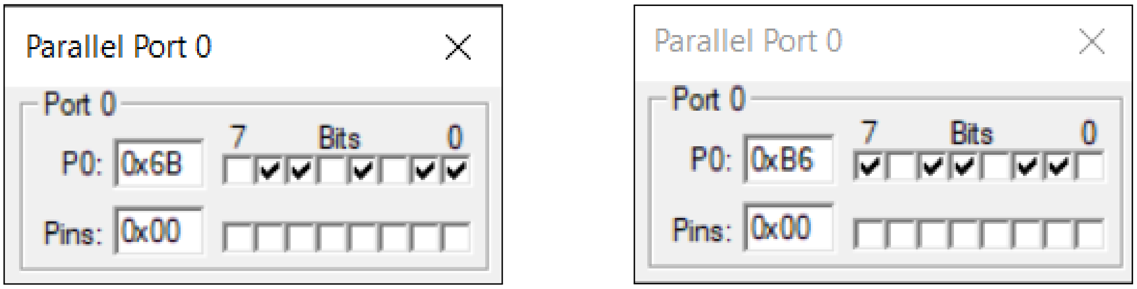
```
00 | 40 04 7F BC |04 C2 00 4B
08 | 26 00 00 00 |00 00 00 00
10 | 00 00 00 00 |00 00 00 00
18 | 00 00 00 00 |00 00 00 00
20 | 00 00 00 00 |00 00 00 00
28 | 00 00 00 00 |00 00 00 00
30 | 00 00 00 00 |00 00 00 00
38 | 00 00 00 00 |00 00 00 00
40 | E2 3B BE 00 |00 00 00 00
48 | 00 00 00 00 |00 00 00 00
50 | 00 00 00 00 |00 00 00 00
58 | 00 00 00 00 |00 00 00 00
60 | 00 00 00 00 |00 00 00 00
68 | 00 00 00 00 |00 00 00 00
70 | 00 00 00 00 |00 00 00 00
78 | 00 00 00 00 |00 00 00 00
```

(e)

Figure 2: Observations for Problem 1

Figure(2) shows the various outputs on Port 0 during the execution of Problem 1. The addition of 897F9AH and 34BC48H gives 00BE3BE2H which is continuously displayed on Port 0 starting from the LSB. Moreover, Figure(2e) shows the IRAM values once the program is run on Proteus simulation. The result for the addition is stored starting from the LSB at 40H location.

Problem 2



Parallel Port 0

Port 0

P0: 0x6B — Bits 7 … 0: ☐ ☑ ☑ ☐ ☑ ☐ ☑ ☑

Pins: 0x00 — ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(a)

Parallel Port 0

Port 0

P0: 0xB6 — Bits 7 … 0: ☑ ☐ ☑ ☑ ☐ ☑ ☑ ☐

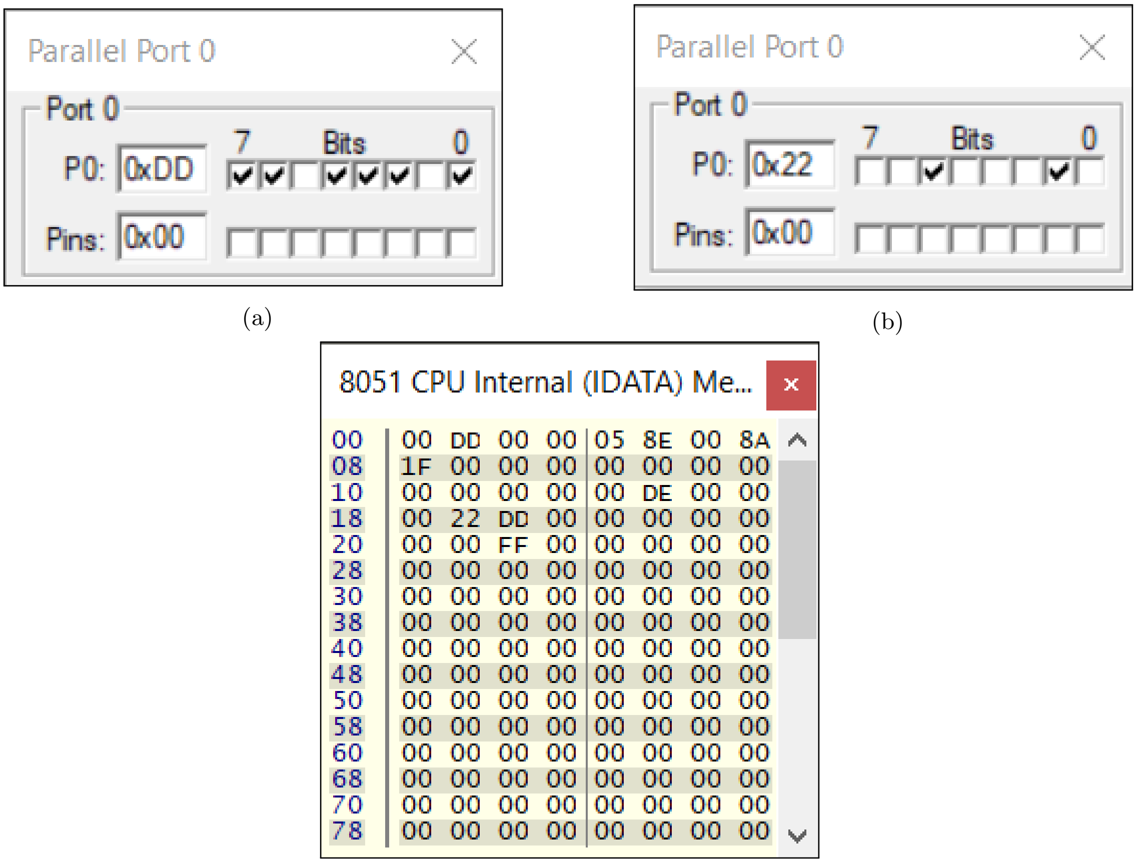
Pins: 0x00 — ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(b)

Figure 3: Observations for Problem 2

Figure(3) shows the output on Port 0 on execution of Problem 2. The upper and lower nibbles of accumulator are swapped without using the SWAP instruction. Hence, 6BH becomes B6H once the swap is performed.

Problem 3



Parallel Port 0

Port 0

P0: 0xDD — Bits 7 … 0: ☑ ☑ ☐ ☑ ☑ ☑ ☐ ☑

Pins: 0x00 — ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(a)

Parallel Port 0

Port 0

P0: 0x22 — Bits 7 … 0: ☐ ☐ ☑ ☐ ☐ ☐ ☑ ☐

Pins: 0x00 — ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(b)

8051 CPU Internal (IDATA) Me...

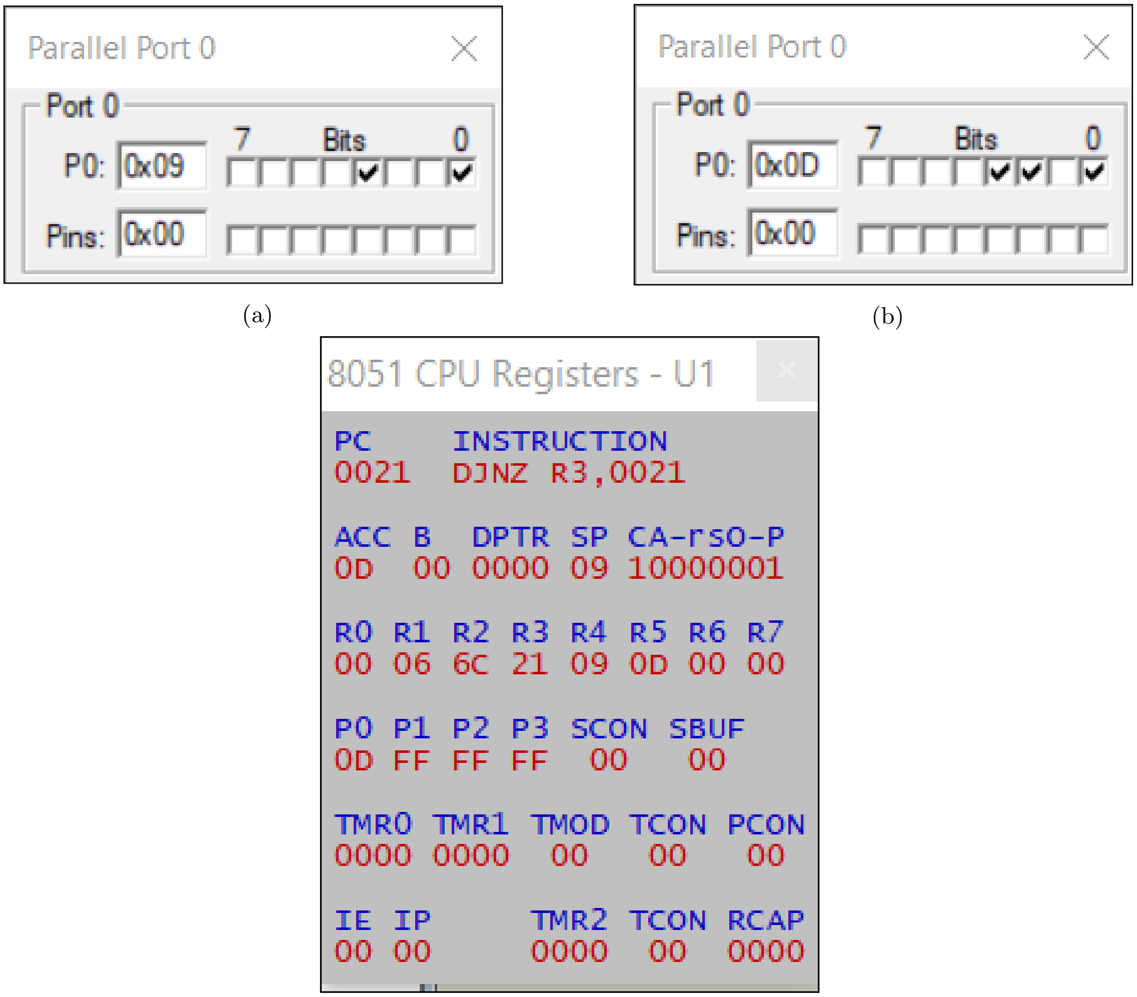
```
00 | 00 DD 00 00 | 05 8E 00 8A
08 | 1F 00 00 00 | 00 00 00 00
10 | 00 00 00 00 | 00 DE 00 00
18 | 00 22 DD 00 | 00 00 00 00
20 | 00 00 FF 00 | 00 00 00 00
28 | 00 00 00 00 | 00 00 00 00
30 | 00 00 00 00 | 00 00 00 00
38 | 00 00 00 00 | 00 00 00 00
40 | 00 00 00 00 | 00 00 00 00
48 | 00 00 00 00 | 00 00 00 00
50 | 00 00 00 00 | 00 00 00 00
58 | 00 00 00 00 | 00 00 00 00
60 | 00 00 00 00 | 00 00 00 00
68 | 00 00 00 00 | 00 00 00 00
70 | 00 00 00 00 | 00 00 00 00
78 | 00 00 00 00 | 00 00 00 00
```

(c)

Figure 4: Observations for Problem 3

Figure(4) shows the result of multiplication of FFH and DEH i.e. DD22H in Port 0. Moreover, the low byte is stored in IRAM location 19H and high byte in 1AH as required by the question which is clear from Figure(4c).

Problem 4



Parallel Port 0

Port 0

P0: 0x09 7 Bits 0 □□□□☑□□☑

Pins: 0x00 □□□□□□□□

(a)

Parallel Port 0

Port 0

P0: 0x0D 7 Bits 0 □□□□☑☑□☑

Pins: 0x00 □□□□□□□□

(b)

8051 CPU Registers - U1

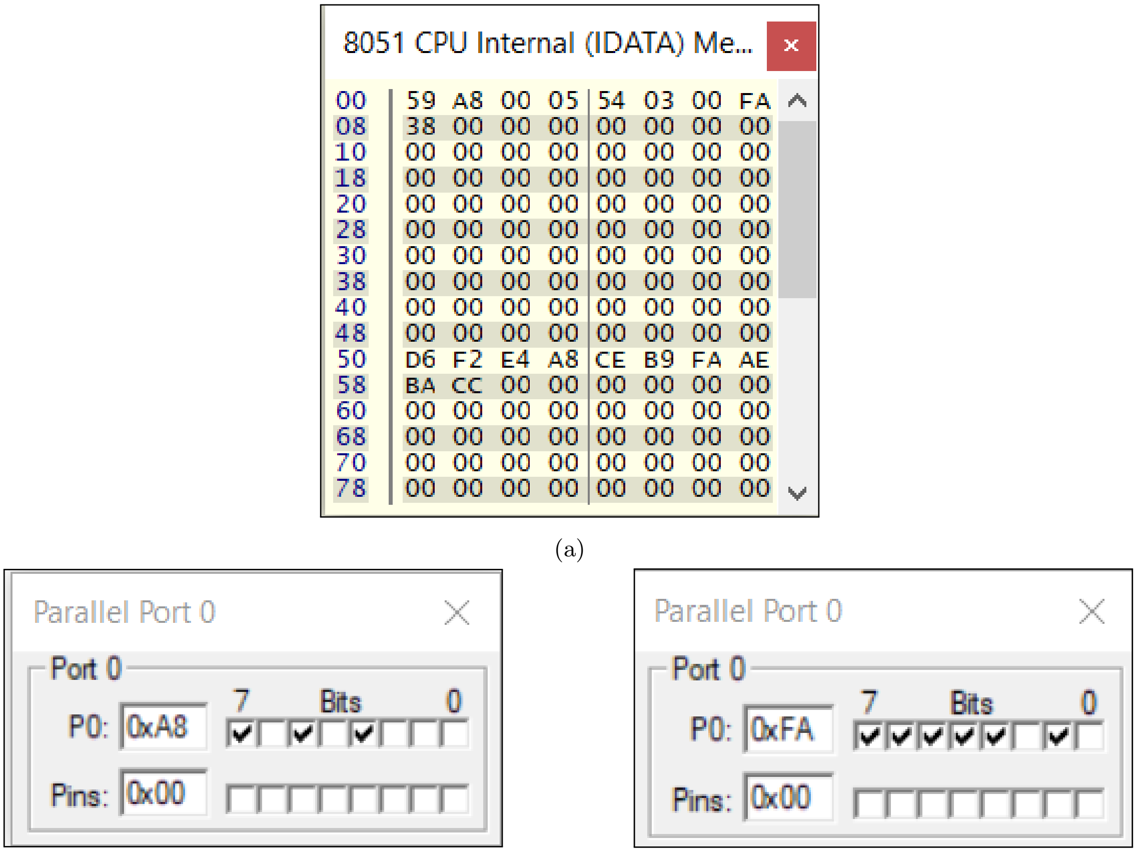
```
PC     INSTRUCTION
0021   DJNZ R3,0021

ACC  B   DPTR SP CA-rsO-P
0D   00 0000 09 10000001

R0 R1 R2 R3 R4 R5 R6 R7
00 06 6C 21 09 0D 00 00

P0 P1 P2 P3 SCON SBUF
0D FF FF FF  00   00

TMR0 TMR1 TMOD TCON PCON
0000 0000  00   00   00

IE IP      TMR2 TCON RCAP
00 00      0000  00  0000
```

(c)

Figure 5: Observations for Problem 4

Figure(5) displays the output for division of AFH by 12H, i.e. quotient=09H and remainder=0DH on the Port 0. The values of quotient and remainder are also stored in R4 and R5 registers as required by the question, which is visible from Figure(5c).

Problem 5



8051 CPU Internal (IDATA) Me...

```
00 | 59 A8 00 05|54 03 00 FA
08 | 38 00 00 00 00 00 00 00
10 | 00 00 00 00 00 00 00 00
18 | 00 00 00 00 00 00 00 00
20 | 00 00 00 00 00 00 00 00
28 | 00 00 00 00 00 00 00 00
30 | 00 00 00 00 00 00 00 00
38 | 00 00 00 00 00 00 00 00
40 | 00 00 00 00 00 00 00 00
48 | 00 00 00 00 00 00 00 00
50 | D6 F2 E4 A8 CE B9 FA AE
58 | BA CC 00 00 00 00 00 00
60 | 00 00 00 00 00 00 00 00
68 | 00 00 00 00 00 00 00 00
70 | 00 00 00 00 00 00 00 00
78 | 00 00 00 00 00 00 00 00
```

(a)

Parallel Port 0

Port 0

P0: 0xA8 7 Bits 0 ☑☐☑☐☑☐☐☐

Pins: 0x00 ☐☐☐☐☐☐☐☐

(b)

Parallel Port 0

Port 0

P0: 0xFA 7 Bits 0 ☑☑☑☑☑☐☑☐

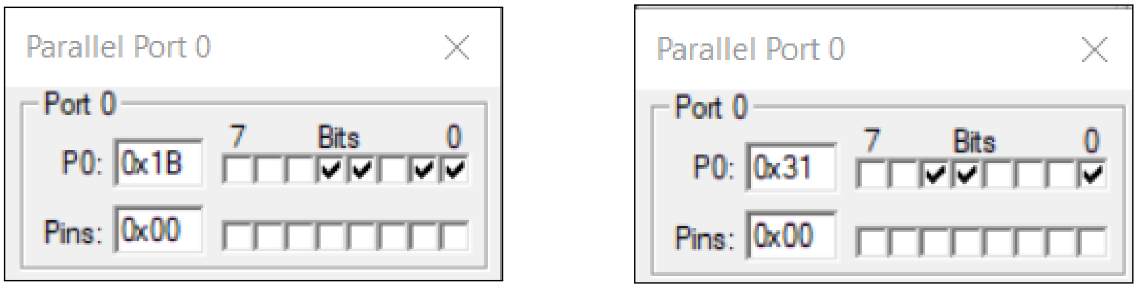
Pins: 0x00 ☐☐☐☐☐☐☐☐

(c)

Figure 6: Observations for Problem 5

Figure(6) shows the display on Port 0 for the smallest and largest hexadecimal numbers i.e. A8H and FAH from a list of 10 numbers stored in the IRAM location starting from 50H, which is observed in Figure(6a).

Problem 6



Parallel Port 0

Port 0

P0: 0x1B 7 Bits 0 ☐☐☐☑☑☐☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(a)

Parallel Port 0

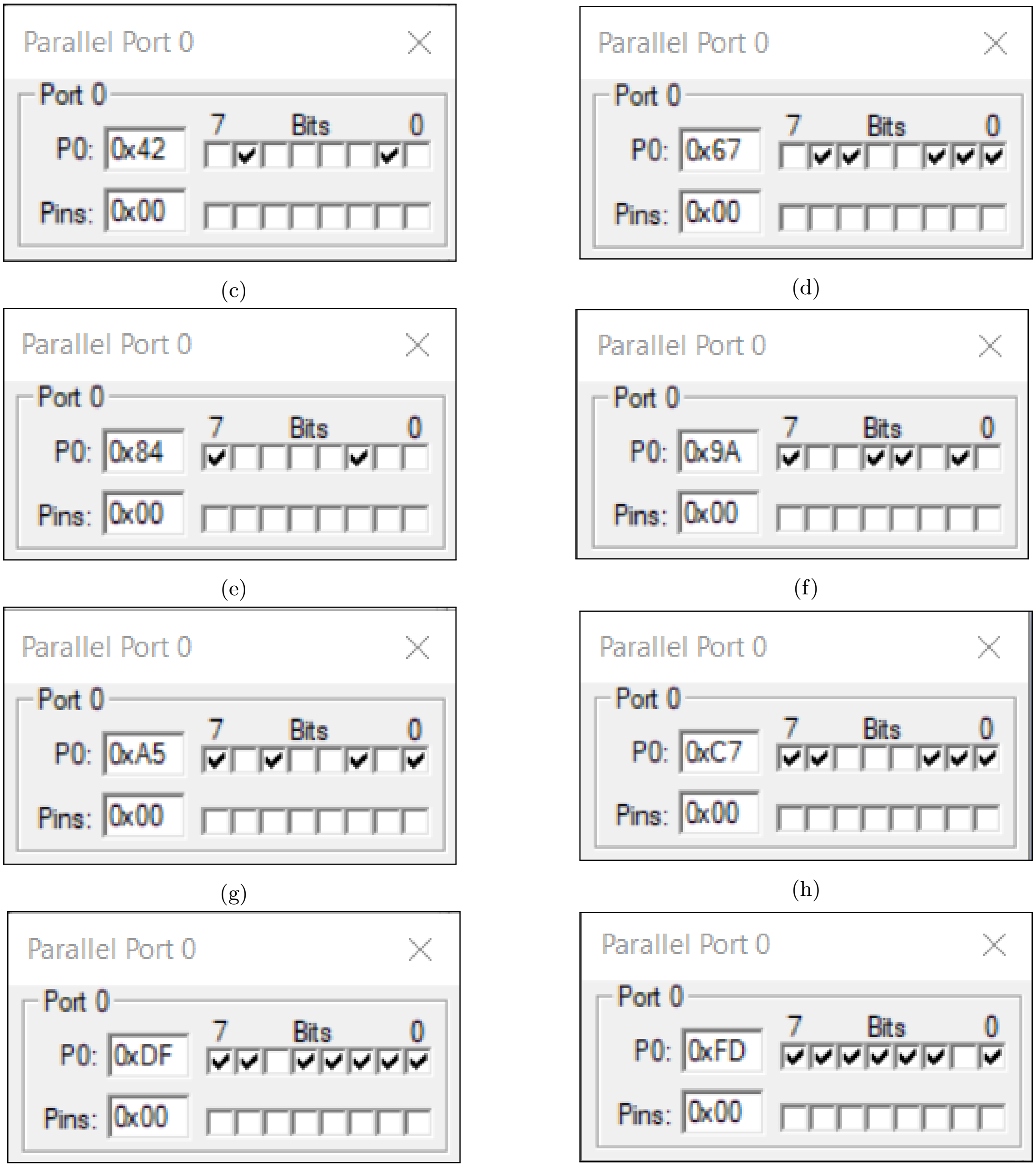
Port 0

P0: 0x31 7 Bits 0 ☐☐☑☑☐☐☐☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(b)

Figure 7: Observations for Problem 6 - Part I



Parallel Port 0
Port 0
7 Bits 0
P0: 0x42 □☑□□□□☑□
Pins: 0x00 □□□□□□□□

(c)

Parallel Port 0
Port 0
7 Bits 0
P0: 0x67 □☑☑□□☑☑☑
Pins: 0x00 □□□□□□□□

(d)

Parallel Port 0
Port 0
7 Bits 0
P0: 0x84 ☑□□□□☑□□
Pins: 0x00 □□□□□□□□

(e)

Parallel Port 0
Port 0
7 Bits 0
P0: 0x9A ☑□□☑☑□☑□
Pins: 0x00 □□□□□□□□

(f)

Parallel Port 0
Port 0
7 Bits 0
P0: 0xA5 ☑□☑□□☑□☑
Pins: 0x00 □□□□□□□□

(g)

Parallel Port 0
Port 0
7 Bits 0
P0: 0xC7 ☑☑□□□☑☑☑
Pins: 0x00 □□□□□□□□

(h)

Parallel Port 0
Port 0
7 Bits 0
P0: 0xDF ☑☑□☑☑☑☑☑
Pins: 0x00 □□□□□□□□

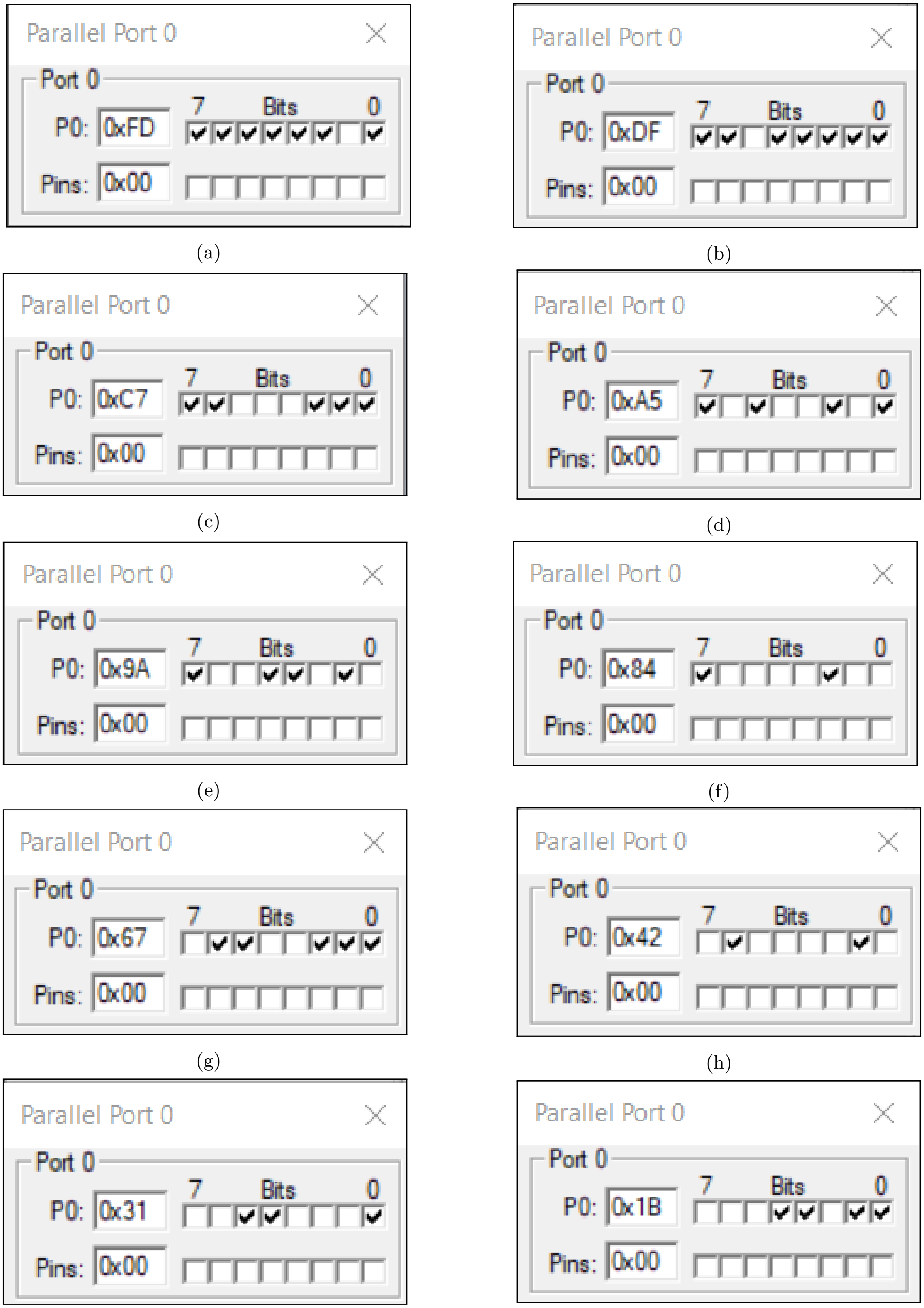
(i)

Parallel Port 0
Port 0
7 Bits 0
P0: 0xFD ☑☑☑☑☑☑□☑
Pins: 0x00 □□□□□□□□

(j)

Figure 7: Observations for Problem 6 - Part I (continued)

Figure(7) shows the different Port 0 outputs which are actually the 10 hexadecimal numbers sorted in ascending order using bubble sort algorithm. Port 0 observations from Figure(7a) to Figure(7j) are arranged in ascending order.



Parallel Port 0

Port 0

7 Bits 0

P0: 0xFD ☑ ☑ ☑ ☑ ☑ ☑ ☐ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(a)

Parallel Port 0

Port 0

7 Bits 0

P0: 0xDF ☑ ☑ ☐ ☑ ☑ ☑ ☑ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(b)

Parallel Port 0

Port 0

7 Bits 0

P0: 0xC7 ☑ ☑ ☐ ☐ ☐ ☑ ☑ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(c)

Parallel Port 0

Port 0

7 Bits 0

P0: 0xA5 ☑ ☐ ☑ ☐ ☐ ☑ ☐ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(d)

Parallel Port 0

Port 0

7 Bits 0

P0: 0x9A ☑ ☐ ☐ ☑ ☑ ☐ ☑ ☐

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(e)

Parallel Port 0

Port 0

7 Bits 0

P0: 0x84 ☑ ☐ ☐ ☐ ☐ ☑ ☐ ☐

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(f)

Parallel Port 0

Port 0

7 Bits 0

P0: 0x67 ☐ ☑ ☑ ☐ ☐ ☑ ☑ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(g)

Parallel Port 0

Port 0

7 Bits 0

P0: 0x42 ☐ ☑ ☐ ☐ ☐ ☐ ☑ ☐

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(h)

Parallel Port 0

Port 0

7 Bits 0

P0: 0x31 ☐ ☐ ☑ ☑ ☐ ☐ ☐ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(i)

Parallel Port 0

Port 0

7 Bits 0

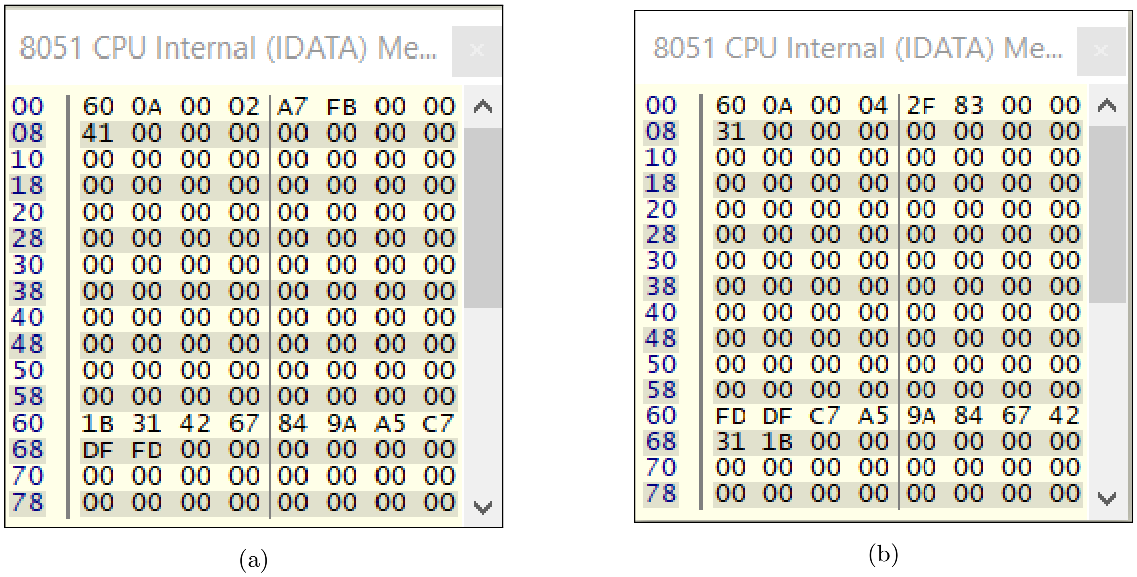
P0: 0x1B ☐ ☐ ☐ ☑ ☑ ☐ ☑ ☑

Pins: 0x00 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

(j)

Figure 8: Observations for Problem 6 - Part II

Figure(8) shows the different Port 0 outputs which are actually the 10 hexadecimal numbers sorted in descending order using selection sort algorithm. Port 0 observations from Figure(8a) to Figure(8j) are arranged in descending order.


8051 CPU Internal (IDATA) Me...

```
00 | 60 0A 00 02 | A7 FB 00 00
08 | 41 00 00 00 | 00 00 00 00
10 | 00 00 00 00 | 00 00 00 00
18 | 00 00 00 00 | 00 00 00 00
20 | 00 00 00 00 | 00 00 00 00
28 | 00 00 00 00 | 00 00 00 00
30 | 00 00 00 00 | 00 00 00 00
38 | 00 00 00 00 | 00 00 00 00
40 | 00 00 00 00 | 00 00 00 00
48 | 00 00 00 00 | 00 00 00 00
50 | 00 00 00 00 | 00 00 00 00
58 | 00 00 00 00 | 00 00 00 00
60 | 1B 31 42 67 | 84 9A A5 C7
68 | DF FD 00 00 | 00 00 00 00
70 | 00 00 00 00 | 00 00 00 00
78 | 00 00 00 00 | 00 00 00 00
```

(a)

8051 CPU Internal (IDATA) Me...

```
00 | 60 0A 00 04 | 2F 83 00 00
08 | 31 00 00 00 | 00 00 00 00
10 | 00 00 00 00 | 00 00 00 00
18 | 00 00 00 00 | 00 00 00 00
20 | 00 00 00 00 | 00 00 00 00
28 | 00 00 00 00 | 00 00 00 00
30 | 00 00 00 00 | 00 00 00 00
38 | 00 00 00 00 | 00 00 00 00
40 | 00 00 00 00 | 00 00 00 00
48 | 00 00 00 00 | 00 00 00 00
50 | 00 00 00 00 | 00 00 00 00
58 | 00 00 00 00 | 00 00 00 00
60 | FD DF C7 A5 | 9A 84 67 42
68 | 31 1B 00 00 | 00 00 00 00
70 | 00 00 00 00 | 00 00 00 00
78 | 00 00 00 00 | 00 00 00 00
```

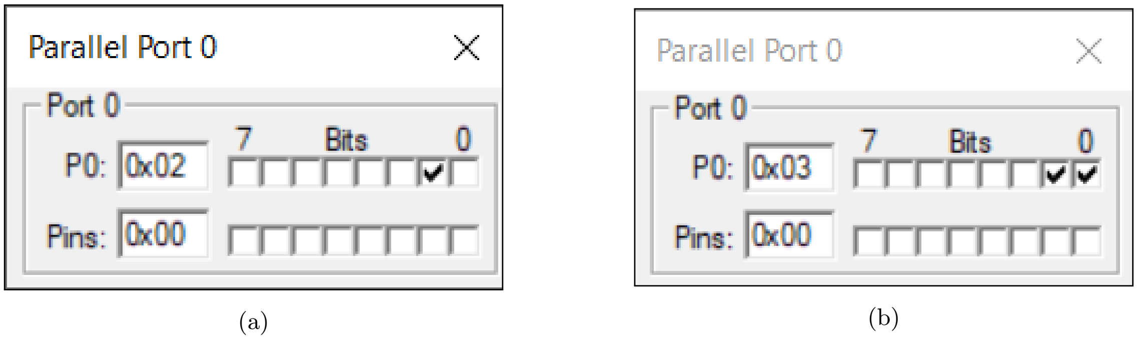
(b)

Figure 9: IRAM observations for Problem 6

Figure(9a) shows the IRAM values for Problem 6 - Part I where the hexadecimal numbers from 60H are sorted in ascending order. Likewise, Figure(9b) shows the Problem 6 - Part II observations where the same hexadecimal numbers are arranged in descending order.

Problem 7

Figure(10) shows the Port 0 outputs for the Problem 7 where only the prime numbers among 00H to 20H stored in memory location starting from 40H were to be shown. Figure(10a) to Figure(10k) display the prime numbers in that range.



P0: 0x02 □□□□□□☑□

Pins: 0x00 □□□□□□□□

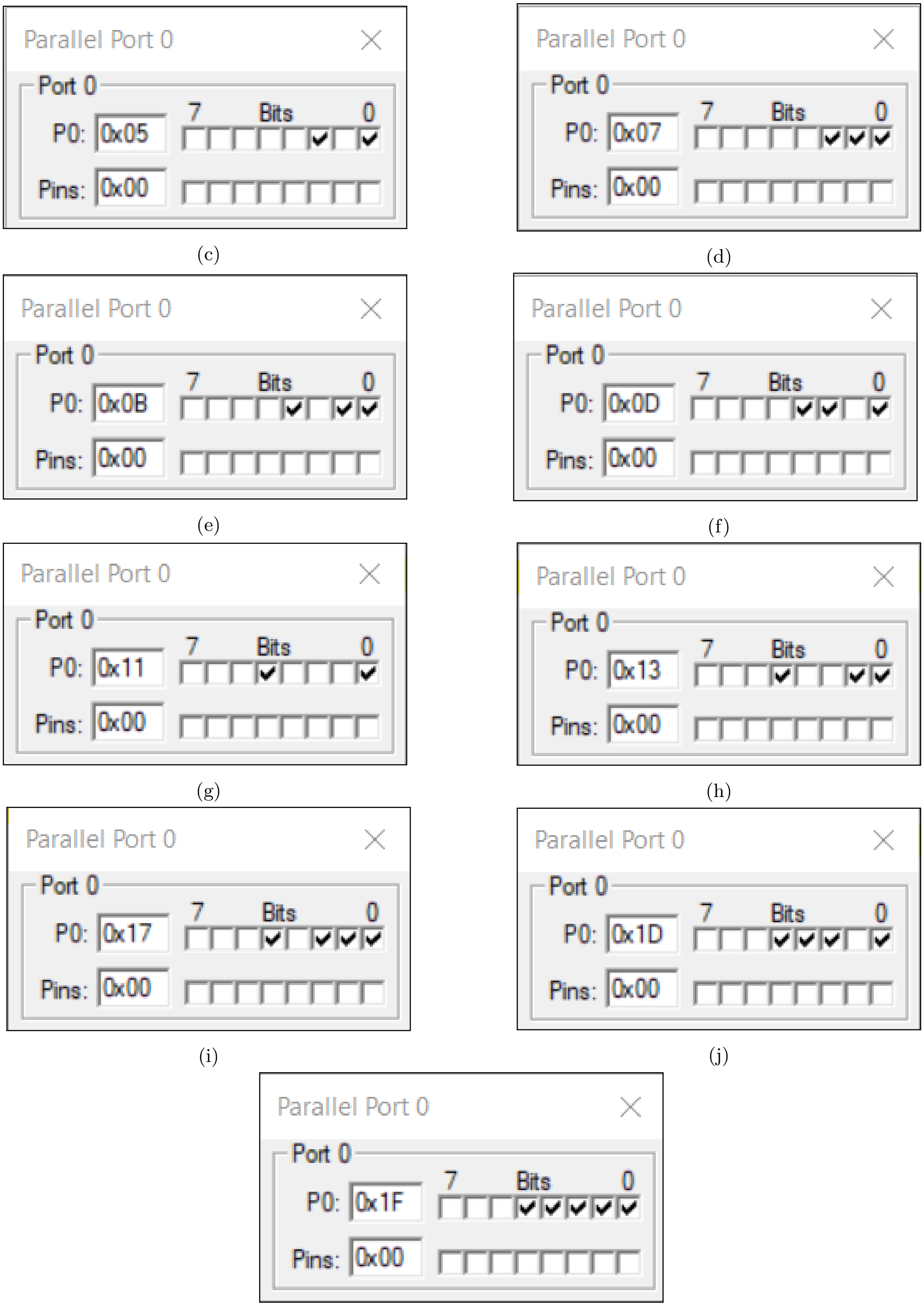
(a)

P0: 0x03 □□□□□□☑☑

Pins: 0x00 □□□□□□□□

(b)

Figure 10: Observations for Problem 7



Parallel Port 0

Port 0

P0: 0x05 7 Bits 0 ☐☐☐☐☐☑☐☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(c)

Parallel Port 0

Port 0

P0: 0x07 7 Bits 0 ☐☐☐☐☐☑☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(d)

Parallel Port 0

Port 0

P0: 0x0B 7 Bits 0 ☐☐☐☐☑☐☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(e)

Parallel Port 0

Port 0

P0: 0x0D 7 Bits 0 ☐☐☐☐☑☑☐☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(f)

Parallel Port 0

Port 0

P0: 0x11 7 Bits 0 ☐☐☐☑☐☐☐☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(g)

Parallel Port 0

Port 0

P0: 0x13 7 Bits 0 ☐☐☐☑☐☐☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(h)

Parallel Port 0

Port 0

P0: 0x17 7 Bits 0 ☐☐☐☑☐☑☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(i)

Parallel Port 0

Port 0

P0: 0x1D 7 Bits 0 ☐☐☐☑☑☑☐☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(j)

Parallel Port 0

Port 0

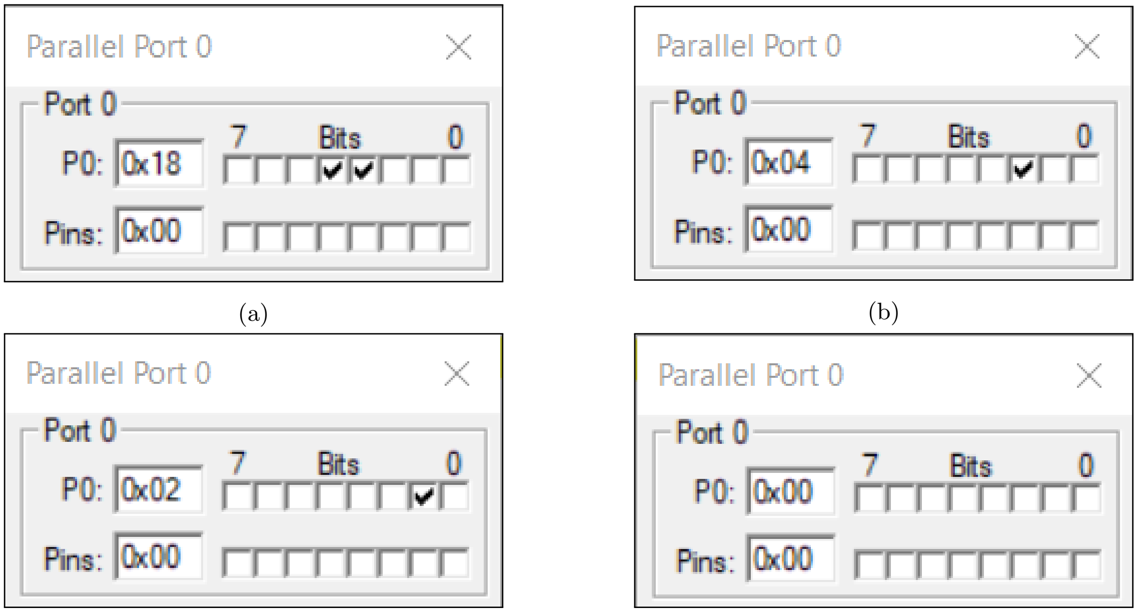
P0: 0x1F 7 Bits 0 ☐☐☐☑☑☑☑☑

Pins: 0x00 ☐☐☐☐☐☐☐☐

(k)

Figure 10: Observations for Problem 7 (continued)

Problem 8



Parallel Port 0

Port 0

P0: 0x18 — 7 Bits 0: □□□☑☑□□□

Pins: 0x00 — □□□□□□□□

(a)

Parallel Port 0

Port 0

P0: 0x04 — 7 Bits 0: □□□□□☑□□

Pins: 0x00 — □□□□□□□□

(b)

Parallel Port 0

Port 0

P0: 0x02 — 7 Bits 0: □□□□□□☑□

Pins: 0x00 — □□□□□□□□

(c)

Parallel Port 0

Port 0

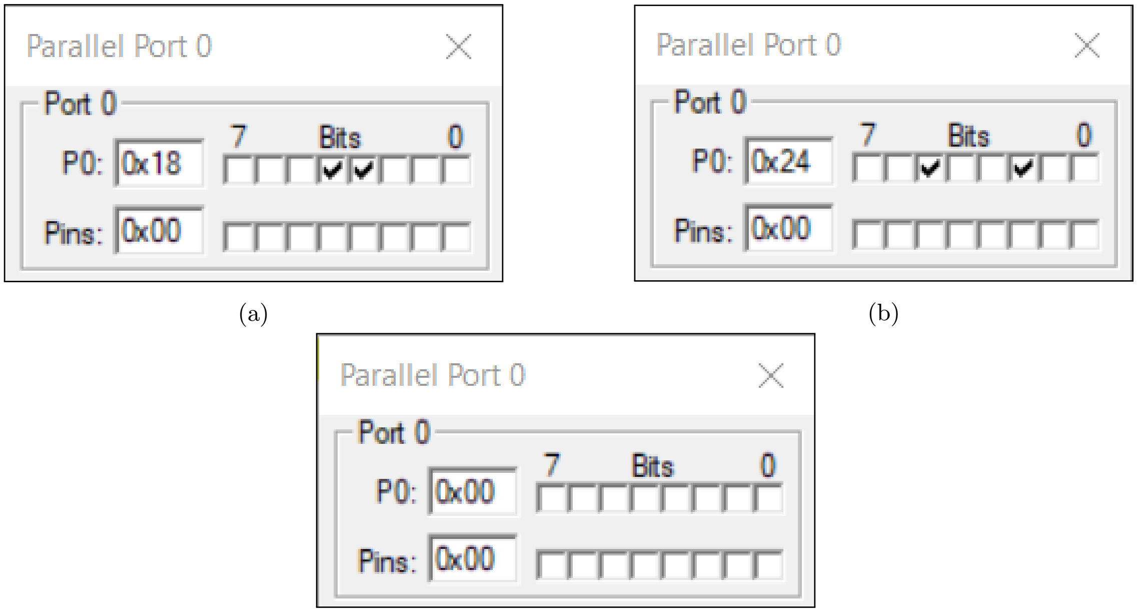
P0: 0x00 — 7 Bits 0: □□□□□□□□

Pins: 0x00 — □□□□□□□□

(d)

Figure 11: Observations for Problem 8 - Embedded C

The hexadecimal number under observation in R3 is 04H. So the factorial of 04H is 18H which is shown in Figure(11a). The decimal equivalent of this is shown in three digits i.e. units, tens and hundreds place in Figure(11b), Figure(11c) and Figure(11d) respectively. The observation for the assembly level code was slightly different for this problem.



Parallel Port 0

Port 0

P0: 0x18 — 7 Bits 0: □□□☑☑□□□

Pins: 0x00 — □□□□□□□□

(a)

Parallel Port 0

Port 0

P0: 0x24 — 7 Bits 0: □□☑□□☑□□

Pins: 0x00 — □□□□□□□□

(b)

Parallel Port 0

Port 0

P0: 0x00 — 7 Bits 0: □□□□□□□□

Pins: 0x00 — □□□□□□□□

(c)

Figure 12: Observations for Problem 8 - Assembly

6 Discussion

In this lab experiment, the 8051/52 microcontroller programming was dealt with various levels of problems. Addition, subtraction, rotation, multiplication, division, additional data manipulation, various logical operations based on flags and subroutine calls were included in the problems that allowed us to be familiar with the basic programming approaches to 8051/52 MCUs. Moreover, the use of KEIL μVision IDE along with Proteus for circuit simulation allowed us to realize the problems in a practical approach. This lab allowed us to learn the basics of programming a 8051/52 MCU in KEIL using both assembly level and Embedded C language. The hex codes generated were then burnt into the simulated circuit in Proteus shown in Figure(1) to visualize the results. The port values, IRAM data and register values were visualized from both KEIL μVision and Proteus in debugging modes such that they represent the output of the problems.

Bibliography

[1] Štěpán Matějka. *8051 Instruction Set Summary*. Accessed: 14-09-2020. URL: https://data.kemt.fei.tuke.sk/AppliedCryptography/_materialy/MCU/it8051.pdf.

Additional References

[2] I. MacKenzie. "The 8051 Microcontroller". In: (June 2009). DOI: 10.1007/978-1-4419-0606-9_3.

[3] Javeed Md. *EMBEDDED C FOR 8051 MICRO CONTROLLER*. Mar. 2019. ISBN: 978-93-5359-284-4.

[4] Jiban S. Parab et al. *Exploring C for Microcontrollers*. [Available Online], Accessed: 17-09-2020. URL: http://ee.sharif.edu/~sakhtar3/books/Exploring%20C%20for%20Microcontrollers.pdf.