Familiarization with 8051/8052 Microcontroller

Brihat Ratna Bajracharya

*Department of Electronics and Computer Engineering, IOE Central Campus, Pulchowk  
Lalitpur, Nepal*

070bct513@ioe.edu.np

***Abstract*—a computer in a single chip is called microcontroller. All necessary blocks of computer like central processing unit, memory, input and output ports, clock, timers/counters and registers are all embedded into a single chip that is used for various educational and other purposes. Intel first introduced MCS-51 microcontroller in 1980. Today various other vendors like Atmel, Infineon Technologies, NXP, Silicon Laboratories, Texas Instruments, Dallas Semiconductors, ASIX, etc. are manufacturing microcontroller compatible with Intel’s MCS-51 that can be used in various embedded systems.**

1. Introduction

The Intel MCS-51 (commonly termed **8051**) is an internally Harvard with CISC (Complex Instruction Set Computing) architecture single chip microcontroller series developed by Intel in 1980 for use in embedded systems. The original MCS-51 family was made using N-type metal-oxide-semiconductor (NMOS) but later versions identified by letter ‘C’ in their name (e.g. 80C51) used complementary metal-oxide-semiconductor (CMOS) technology.

The 8051 architecture provides many functions (like CPU (Central Processing Unit), RAM (Random Access Memory), ROM (Read Only Memory), I/O (Input/Output), Interrupt logic, Timer, etc.) in a single chip/package.

MCS-51 based microcontrollers typically include one or two UARTs, two or three Timers, 128 or 256 bytes of internal data RAM, 128 bytes of I/O, 512 bytes to 64 kilo-bytes of internal program memory and external data space. The original 8051 runs at 12 MHz clock frequency. Today’s 8051 microcontroller has clock frequencies of up to 100 MHz.

1. *Registers*

There are eight 8-bit general purpose registers (R0 – R7). 8051 also have 8-bit Stack Pointer (SP, 0x81), 16-bit Data Pointer (DP, 0x82-83) and 8-bit Program Status Word (PSW). PSW consists of status flags like Parity (PSW.0), Overflow (PSW.2), Register Select (PSW.3 and PSW.4), Auxiliary Carry (PSW.6) and Carry (PSW.7). PSW does not contain Negative and Zero flags.

Accumulator (A, 0xE0) stores all intermediate result and B register (0xF0) along with accumulator is used for multiplication and division instructions.

1. *Memory Architecture*

The MCS-51 has four distinct types of memory.

1. *Internal RAM* – It has an 8 bit address space that allows access through 0x00 to 0xFF. RAM from 0x00 to 0x7F can be accessed directly and rest is accessed indirectly.
2. *Special Function Registers* – They are located in the same address space as RAM from address location 0x80 to 0xFF and are accessed directly.
3. *Program Memory* – It uses up to 64 kilo-bytes of ROM starting at address 0x00 in separate address space. It is accessed by the MOVC A,@DPTR instruction.
4. *External Data Memory* – It is a third address space starting at address 0x00 and allowing 16 bits of address space and is accessed using MOVX (MOVe eXternal) instruction. The first 256 bytes can be accessed using MOVX A,@R0 instruction whereas full 64 kilo-bytes can be accessed using MOVX A,@DPTR and MOVX @DPTR,A instructions.
5. *Instruction Set*

Instructions in 8051 are all one to three bytes long, consisting of op-code byte followed by two bytes of operands. The most significant nibble of op-code specifies the operation and least significant nibble specifies one of twelve addressing modes as follows.

* x8-xF – Register direct (R0-R7)
* x6-x7 – Register indirect (@R0 and @R1)
* x5 – Memory direct, next byte specifies RAM or SFR

location

* x4 – Immediate, next byte specifies 8-bit constant

The operation in 8051 uses mnemonics as follows:

* 0y – INC operand, e.g. 04 specifies INC A
* 1y – DEC operand, e.g. 14 specifies DEC A
* 2y – ADD A, operand, adds operand to A
* 3y – ADDC A, operand, adds with carry
* 4y – ORL A, operand, (A 🡨 A **or** operand)
* 5y – ANL A, operand, (A 🡨 A **and** operand)
* 6y – XRL A, operand, (A 🡨 A x**or** operand)
* 7y – MOV operand, #data,

e.g. 74 specifies MOV A, #data

* 8y – MOV address, operand,

Moves data to RAM or SFR

* 9y – SUBB A, operand,

Subtracts operand from A with borrow

* Ay – MOV operand, address

Moves data from RAM or SFR

* By – CJNE operand, #data, offset,

Compares operand to data and branch to PC + offset if not equal. B4 and B5 performs CJNE A, operand, offset. There is no compare and jump if equal instruction

* Cy – XCH A, operand, swaps A and operand
* Dy – DJNZ operand, offset

Decrement the operand and branch to PC + offset if result is not zero

* Ey – MOV A, operand, moves operand to A
* Fy – MOV operand, A, moves A to operand

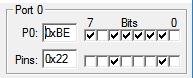
Only ADD, ADDC, and SUBB affects PSW flags and CJNE instruction modifies the carry bit only

1. *AT89S52 Micro-controller*

The AT89S52 is a low power, high performance CMOS eight bit microcontroller with 8 kilo-bytes of in-system programmable flash memory. The device is manufactured using Atmel’s high-density non-volatile memory technology and is compatible with the industry-standard 80C51 instruction set. The Atmel AT89S52 is a powerful microcontroller which provides a highly flexible and cost effective solution to many embedded control applications.

The AT89S52 provides the following standard features:

* 8 kilo-bytes of flash memory,
* 256 bytes of RAM,
* 32 I/O lines,
* Watchdog timer,
* 2 data pointers (DP),
* 3 16-bit timer/counters,
* A six-vector two-level interrupt architecture,
* A full duplex serial port,
* On-chip oscillator, and
* Clock circuitry.

In addition, the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The **idle mode** stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The **power down mode** saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

1. Activity I

Result of 89 7F 9A H plus 34 BC 48 H is 00 BE 38 E2 H

Write code to add the numbers 897F9AH to 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 cycle. Use port zero (P0) of the microcontroller to interface the 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

HERE1: MOV R5,#255

HERE2: MOV R7,#255

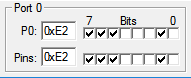
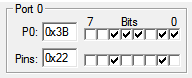
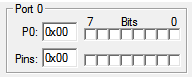
HERE3: DJNZ R7,HERE3

DJNZ R5,HERE2

DJNZ R4,HERE1

RET

END

Output:

1. Activity II

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. The original number and the result should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

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

HERE1: MOV R5,#255

HERE2: MOV R7,#255

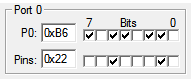
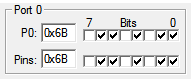
HERE3: DJNZ R7,HERE3

DJNZ R5,HERE2

DJNZ R4,HERE1

RET

END

Output:

Product of FF H and DE H is DD 22 H

Swapping upper and lower nibble of accumulator

1. Activity III

Multiply 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. Use looping and successive addition technique. The product (high byte and low byte) should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

Assembly Code:

ORG 00H

MOV 22H,#0FFH

MOV 15H,#0DEH

MOV A,#0H

MOV R1,#0H

MOV R0,51H

AGAIN: ADD A,50H

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

HERE1: MOV R5,#255

HERE2: MOV R7,#255

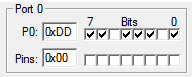
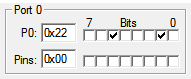
HERE3: DJNZ R7,HERE3

DJNZ R5,HERE2

DJNZ R4,HERE1

RET

END

Output:

1. Activity IV

Divide the data in RAM location 3EH by the number 12H; put the quotient in R4 and the remainder in R5. Data in 3EH should be AFH. Use looping and successive subtraction technique. The quotient and remainder should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

Assembly Code:

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

HERE1: MOV R2,#255

HERE2: MOV R3,#255

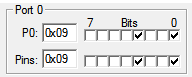
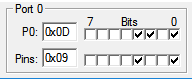
HERE3: DJNZ R3,HERE3

DJNZ R2,HERE2

DJNZ R1,HERE1

RET

END

Output:

Dividing AF H by 12 H gives quotient = 9 H and remainder = D H

1. Activity V

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. The smallest and largest numbers should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

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 ;SMALLEST

MOV R1,A ;LARGEST

MOV R2,#09H

NEXT: INC R0

MOV A,R7

SUBB A,@R0

JNC NO\_SMALL

MOV A,@R0

MOV R7,A

NO\_SMALL: MOV A,R1

SUBB A,@R0

JC NO\_BIG

MOV A,@R0

MOV R1,A

NO\_BIG: DJNZ R2,NEXT

LOOP: MOV P0,R7

ACALL DELAY

MOV P0,R1

ACALL DELAY

AJMP LOOP

DELAY: MOV R3,#7

HERE1: MOV R4,#255

HERE2: MOV R5,#255

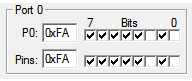
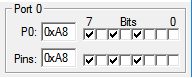
HERE3: DJNZ R5,HERE3

DJNZ R4,HERE2

DJNZ R3,HERE1

RET

END

Output:

Largest number = FA H and smallest number = A8 H

1. Activity VI

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.

1. Implement a subroutine that orders the numbers in ascending order using bubble sort algorithm. The sorted list of numbers should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

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

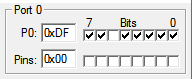
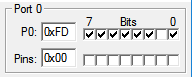
MOV 68H,#0C7H

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

Sorted order (Ascending) using Bubble sort algorithm

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

HERE1: MOV R4,#255

HERE2: MOV R5,#255

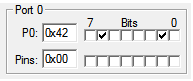
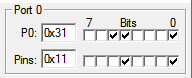
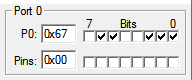
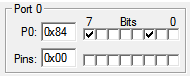
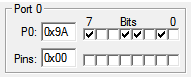
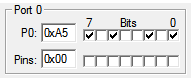
HERE3: DJNZ R5,HERE3

DJNZ R4,HERE2

DJNZ R3,HERE1

RET

END

Output:

2. Implement a subroutine that orders the numbers in descending order using selection sort algorithm. The sorted list of numbers should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

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 R0,#60H

MOV R6,#09H

AGN: ACALL F\_LARGE

MOV @R0,A

INC R0

DJNZ R6,AGN

AGAIN: MOV R1,#0AH

MOV R0,#60H

LOOP: MOV A,@R0

MOV P0,A

ACALL DELAY

INC R0

DJNZ R1,LOOP

AJMP AGAIN

F\_LARGE: MOV B,R0

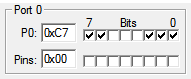
MOV A,R6

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

HERE1: MOV R4,#255

HERE2: MOV R5,#255

HERE3: DJNZ R5,HERE3

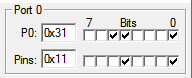
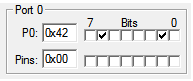
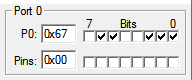
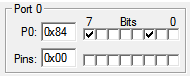
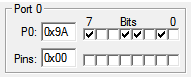
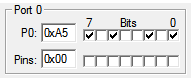
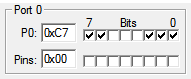
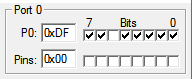
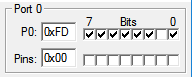
DJNZ R4,HERE2

DJNZ R3,HERE1

RET

END

Output:



1. Activity VII

Store numbers from 00H to 20H in internal RAM starting from memory location 40H. Implement a subroutine that extracts only the prime numbers. The prime numbers should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

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 DONE2

MOV A,R1

AJMP AGAIN

DONE2: 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 DONE2

PRIME: MOV A,@R0

MOV R4,A ; SAVE 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

Sorted order (Descending) using Selection sort algorithm

MOV A,R4

MOV P0,A

ACALL DELAY

RET

DELAY: MOV R7,#7

HERE1: MOV R6,#255

HERE2: MOV R5,#255

HERE3: DJNZ R5,HERE3

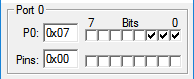
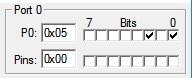
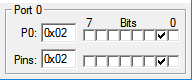
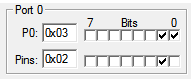
DJNZ R6,HERE2

DJNZ R7,HERE1

RET

END

Output:

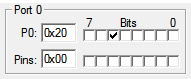


1. Activity VIII

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. The factorials in hexadecimal and decimal formats should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

Assembly Code:

ORG 00H

 MOV R3,#05H

MOV B,R3

MOV R1,B

ACALL FACTO

MOV R1,A

Factorial of 5 is 78 H or 120 D

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

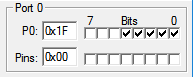
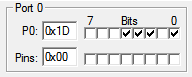
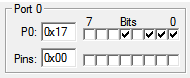
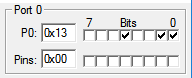
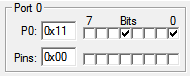
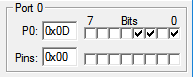
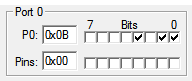
JC SKIP

MOV A,R2

MOV R3,B

MOV B,#0AH

DIV AB



Prime numbers between 00 H and 20 H

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

FACTO: MOV A,#01H

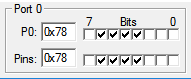
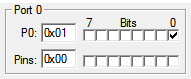
LOOP: MOV B,R1

MUL AB

DJNZ R1,LOOP

RET

END

Output:

1. Activity IX

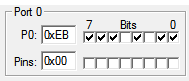
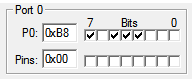
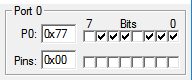
Store ten hexadecimal numbers in internal RAM starting from memory location 55H. The list of numbers to be used is: 25H, 3DH, F7H, 13H, C9H, 4EH, 62H, 77H, B8H, EBH. Implement a subroutine that searches for the occurrence of the binary sequence (11)2 within a number amongst the given list of numbers. Only those numbers that contain the specified binary sequence should be displayed continuously on the LEDs of the development board one-by-one with an appropriate timing interval between them. Use port zero (P0) of the microcontroller to interface the LEDs.

Assembly Code:

ORG 00H

MOV 55H,#25H

MOV 56H,#3DH



Numbers with binary sequence (11)2

MOV 57H,#0F7H

MOV 58H,#13H

MOV 59H,#0C9H

MOV 5AH,#4EH

MOV 5BH,#62H

MOV 5CH,#77H

MOV 5DH,#0B8H

MOV 5EH,#0EBH

LOOP: MOV R0,#55H

MOV R6,#0AH

AGN: ACALL B\_SEQ

INC R0

DJNZ R6,AGN

AJMP LOOP

B\_SEQ: MOV A,@R0

MOV R1,A

MOV R2,#08H

BITT: RLC A

DEC R2

JC CHK\_NXT

DJNZ R2,BITT

RET

CHK\_NXT: RLC A

DEC R2

JNC BITT

MOV A,R1

MOV P0,A

ACALL DELAY

RET

DELAY: MOV R3,#7

HERE1: MOV R4,#255

HERE2: MOV R5,#255

HERE3: DJNZ R5,HERE3

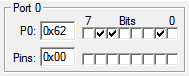
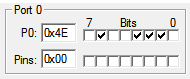
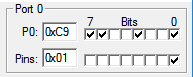
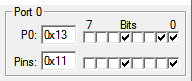
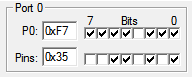
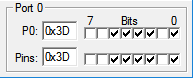
DJNZ R4,HERE2

DJNZ R3,HERE1

RET

END

Output:

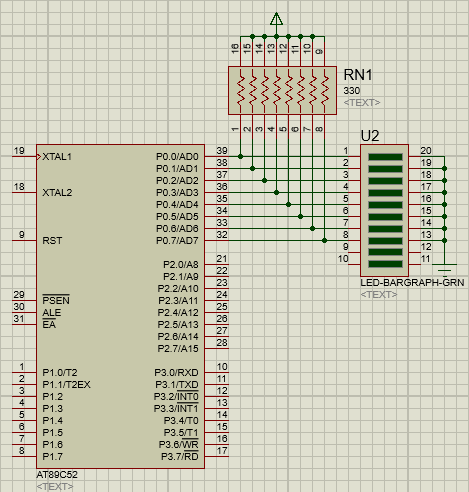


Conclusion

To be familiar with 8051 microcontroller and assembly language, various lab activities were done in assembly as well as in C programming Language. Keil IDE and Proteus Simulation Software were used to verify the result. Schematic diagram made in Proteus is included in Appendix section. Codes to all activities in assembly language are included in this report. In addition, all activities are also done in C programming language and their source code is given in Appendix section.

Appendix

**Appendix A**

Proteus Schematic Capture

**Appendix B**

Programs in C programming language

1. **C code for Activity I**

#include <reg51.h>

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);

}

}

1. **C code for Activity II**

#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 ivalue;

unsigned char a,b;

a = value / 0x10;

b = value % 0x10;

ivalue = b \* (0x10) + a;

while(1)

{

P0 = value;

delay(1000);

P0 = ivalue;

delay(1000);

}

}

1. **C code for Activity III**

#include <reg51.h>

char data multiplicand \_at\_ 0x22;

char data multiplier \_at\_ 0x15;

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);

}

}

1. **C code for Activity IV**

#include <reg51.h>

unsigned 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;

unsigned char 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);

}

}

1. **C code for Activity V**

#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);

}

}

1. **C code for Activity VI - 1**

#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;

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);

}

}

}

1. **C code for Activity VI 2**

#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);

}

}

}

1. **C code for Activity VII**

#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);

}

}

}

1. **C code for Activity VIII**

#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 val = 0x5;

unsigned int fact = 0x1;

unsigned char i;

unsigned char x, d1, d2, d3;

for(i=0x1; i<=val; 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);

}

}

1. **C code for Activity IX**

#include <reg51.h>

char data a[10] \_at\_ 0x55;

void delay(int time)

{

unsigned int i,j;

for (i=0; i<time; i++)

for (j=0; j<125; j++);

}

int check\_seq(unsigned char val)

{

unsigned char j;

for(j=0; j<8; j++)

if((val & 0x3) == 0x3)

return 1;

else

val >>= 1;

return 0;

}

void main(void)

{

unsigned char i, count = 0, b[10];

a[0] = 0x25; a[1] = 0x3d;

a[2] = 0xf7; a[3] = 0x13;

a[4] = 0xc9; a[5] = 0x4e;

a[6] = 0x62; a[7] = 0x77;

a[8] = 0xb8; a[9] = 0xeb;

for(i=0;i<10;i++)

if(check\_seq(a[i]))

b[count++] = a[i];

while(1)

for(i = 0;i<count;i++)

{

P0 = b[i];

delay(1000);

}

}

Acknowledgment

This lab and the report is prepared to be familiar with the assembly language programming as well as C programming for 8051 microcontroller using Keil IDE. This report is made accurate and professional as far as possible. I would like to express our deepest gratitude to our teacher, Mr. Dinesh Baniya Kshatri, for guiding us in the practical. I am very grateful to the Department of Electronics and Computer Engineering (DoECE) of IOE Central Campus, Pulchowk for arranging such a schedule on our academic side. I would also like to thank my friends who helped me on understanding certain things in assembly and C programming in this lab.

References

[1] (2016) The Wikipedia website. [Online]. Available: <https://en.m.wikipedia.org/wiki/Intel_MCS-51>

[2] ATMEL, AT89S52 [portable document]. Available: Offline

[3] (2016) Keil Embedded C Tutorial. [Online], Available: <http://www.8051projects.net/wiki/Keil_Embedded_C_Tutorial>

[4] C. P. Young, The 8051 Microcontroller and Embedded Systems Using Assembly and C [Presentation]. Available: Offline.

[5] D. Kshatri, Familiarization with 8051/8052 Microcontroller [handout]. Available: Printed form.