CONTENTS

Date : 18-09022 Expt. No. : Page No. : 01

Follow the steps below to download and install Keil

Step 1: Go to site "https://www.keil.com/download/product/" → download → product downloads → Hit on C51 setup.

Enter your contact information as required in fields press Enter

Step 2: A new site pops up click on "C5149-54A.exe" and download it on your pc at desired location.

Step 3: Run the download setup file. A new window pops-up. hit next and proceed

Step 4: Accept the argument terms and condition and Click next.

Step 5: Choose the installation directory fill up the contack into detail & click next.

Step 6: The installation is in progress. once the installation will be done, click the finish bottom on the bottom right of the window

Follow the steps below to download and install proteus 8 sp0

Step1: Download and store the proteus 8 sp0. setup proteus in desired location from site "Libcenter.com"

Step 2: Run the setup file, a new window pop-up click on Next

Step 3: A checkbox the terms of agreement and click Next.

Step 4: In this new window choose the option "Use a locally installed License Key" and Click on Next

Step 5: A new window pops up where we have to browse for the key file.

Step 6: To browse, click on Browsex for key file on the bottom left Corner.

Step 7: To the downloaded folder search for file "Licence.txt" and press open. once opened click on install button on button right.

Step 8: A window pops up asking to continue. Click Yes and then the file will be installed click close then.

step 9: Again a window pops up, click next to continue and then click on the choice of installation, which will be Typical.

step 10: The installation is in progress once done click on finish (or) close

step 11: Now from the folder you had downloaded before select and copy two folders 'BIN' and MODELS' paste these two folders where your proteus is installed. [In program file > labcenter, electronics > proteus 8 professional]

[Experiment no: 1]

Date : 23-09-022 Expt. No. : 01(a) Page No. : 04

Data Transfer programming:

AIM:

Write an assembly language program to transfer N: — types of data from location A: --h to location B: ---h.

Components Required:

→ Keiluvision Simulator

program:

⇒ Let N= 05h A: 30h B: 40h

```
MOV r0, # 30h    || Source address
Mov r1, # 40h    || Destination address
Mov r7, # 05h    || Number of bytes to be removed.
back: Mov a, @r0
      Mov @r1, a
      inc r0
      inc r1
      djnz r7, back   || repeat till all data transferred
      end
```

Result:

Before execution

Address : d : 30h

D : 0x30 : 01 02 03 04 05 00 00

D : 0x3A : 00 00 00 00 00 00 00

After execution.

Address : d : 40h

D : 0x40 : 01 02 03 04 05 00 00

D : 0x4A : 00 00 00 00 00 00 00

[C - language program]

To transfer N : — bytes of data from location A : — h to location B : —

⇒ Let N = 05h A : 30h B : 40h

```
#include <reg 51h>
 Void main (void) {
  Unsigned Char source[5] = { 0x11, 0x12, 0x13, 0x14, 0x15};
  unsigned char destination[5];
  unsigned Char i;
  for( i=0; i<5 ; i++)
  {
   destination[i]= source[i];
   }
   while (i);
    }
```

Date : 28-09-022 Expt. No. : 1 (b) Page No. : 05

AIM :

Write an assembly language program to exchange N: __ h bytes of data at location A: __h and at location B: __h

Component required

→ keiluvision simulator

program :

∋ Let N = 05h A = 30h B = 40h

```
Mov r0, # 30h    || Source address
Mov r1, # 40h    || Destination address
Mov r7, # 05h    || Number of bytes to be moved

Back: Mov a ,@r0
      Mov r2, a
      Mov a,@r1
      Mov @r0, a
      Mov a, r2
      Mov @r1, a
      inc r0
      inc r1
      djnz r7, back
      end
```

Result:

Before execution

Address: d:30h

D: 0X30: 01 02 03 04 05 00

D: 0X3A: 00 00 00 00 00 00

Address: d:40h

D: 0X40: 06 07 08 09 10 00

D: 0X4A: 00 00 00 00 00 00

After execution

Address: d:30h

D: 0X30: 06 07 08 09 10 00

D: 0X3A: 00 00 00 00 00 00

Address: d:40h

D: 0X40: 01 02 03 04 05 00

D: 0X4A: 00 00 00 00 00 00

[Experiment no: 2]

Date : 06-10-022 Expt. No. : 2 (a) Page No. : 06

AIM :

Write an assembly language program to perform the addition of two 16-bit numbers.

Component required

→ Keiluvision simulator

program

```
Mov r0, # 34h        || Lower nibble of No. 1
Mov r1, # 12h        || higher nibble of No. 1
Mov r2, # 0dch       || Lower nibble of No. 2
Mov r3, # 0feh       || higher nibble of No. 2

clr c
Mov a, r0
add a, r2
Mov 22h, a
Mov a, r1
addc a, r3
Mov 21h, a
Mov 00h, c
end
```

Result:

Before execution

Address : d : 20h

D : 0x20 : 00 00 00 00 00 00

D : 0x3B : 00 00 00 00 00

After execution

Address : d : 20h

D : 0x20 : 01 10 00 00 00

D : 0x3B : 00 00 00 00 00

[Embedded C - program]

```
#include <reg 51.h>
void main (void) {
unsigned char a,b,c;
unsigned char add 0 = 0x32;
unsigned char add 1 = 0x34;
unsigned char add 2 = 0x36;
unsigned char add 3 = 0x33;

add 0 = add 0 & 0x0F;
add 1 = add 1 & 0x0F;
add 2 = add 1 << 4;
a = add 1 | add r0;
b = add 3 | add r2;
c = a+b;
while (1) }
```

[Experiment no: 2]

Date : 06-10-022 Expt. No. : 2 (b) Page No. : 07

AIM:

Write an assembly language to perform the subtraction of two 16 bit numbers.

Component required

→ Keiluvision simulator

program

```
MOV r0 , # 0dch     // Lower nibble of NO.1
MOV r1, # 0fch      // higher nibble of NO.1
MOV r2, # 34h       // Lower nibble of NO. 2
MOV r3, # 12 h      // higher nibble of NO. 2

clr c
MOV a, r0
subb a, r2
MOV 22h, a
Mov a, r1
MOV a, r3
MOV 21h, a
MOV 00h, c
end
```

Result:

Before execution:

Address : d : 20h

D: 0x20 : 00 00 00 00

D: 0x3B : 00 00 00 00

After execution:

D: 0x20 : 00 FC AB 00 00

D: 0x3B : 00 00 00 00 00

[Embedded C - program]

```
#include <reg 51h>
void main (void)
{
   unsigned char a,b,c
   unsigned char subb 0 = 0x32;
   unsigned char subb 1 = 0x34;
   unsigned char subb 2 = 0x36;
   unsigned char subb 3 = 0x33;

    subb 0 = subb 0 & 0x0F;
    subb1 = subb 1 & 0x0F;
    subb 2 = subb 1 << 4;
     a = subb1 | subb r0;
     b = subb 3 | subb r2;
     c = a-b;
    while (1)
    }
```

Date : 06-10-022 Expt. No. : 2(c) Page No. : 08

AIM:

To find whether given eight bit number is odd or even. If odd store 00h in accumulator. If even store FFh in accumulator.

Component required

→ keiluvision simulator

program

```
Mov a, 20h      // 20h = given number, to find is it
                   even or odd
Jb acc.0, odd   // Jump if direct bit is set i.e. if
                   lower bit is 1 then number is odd
Mov a, # 0FFh
Sjmp ext
add Mov a, # 00h
ext
   end
```

Result:

Address : d: 20h;

D: 0x20 : 22 00 00 00

Output

a: 0xFF

Address: d: 20h

D: 0x20 : 23 00 00 00

Output

a: 0x00

Date : 06-10-022 Expt. No. : 2(d) Page No. : 09

AIM:

To perform logical operations AND, OR, XOR, on two eight bit numbers stored in internal RAM location 21h and 22h

Component required

→ Keiluvision Simulator

program

```
Mov a, 21h   // don't use #, as data ram 21h is to be accessed
anl a, 22h   // Logical and operation
Mov 30h, a   // and operation result stored in 30h
Mov a, 21h
orl a, 22h   // Logical or operation
Mov 31h, a   // or operation result stored in 31h
Mov a, 21h
xrl a, 22h   // Logical xor operation
Mov 32h, a   // xor operation result stored in 32h
end
```

Result:

Before execution:

Address: d: 21h

D: 0x21: 05 03 00 00 00 (Input)

After execution:

Address: d: 30h

D: 0x30: 01 07 06 00 00 (output)

[C-program]

```
#include <reg51.h>
Void main (void) {
P0 = 0x15 & 0x85 ; // Logical And operation
P1 = 0x4A || 0x15 ; // Logical OR operation
P2 = 0x53 ^ 0x15 ; // Logical XOR operation
while (1);
}
```

[Experiment no: 3]

Date: 07-10-021 Expt. No.: 3 (a) Page No.: 10

Code Conversion programming

AIM:

Write an assembly language program to convert a BCD number to ASCII

Component required:

→ Keiluvision simulator

program

```
MOV A, # 09H // The BCD number to be Converted to ASCII
Mov R0, A
SWAP A
MOV DPTR , # 9000H // output will be in 9000H and
                      9001 H
ACALL ASCII
MOVA , R0 ACALL ASCII
Sjmp $
ASCII : ANL A, # 0F H
ADD A, # 30H
Mov X@DPTR, A (There should be space between Mov
                and X)
INC DPTR
RET
END
```

Result:

Before execution

Input : x : 9000h

X : 0x009000 : 00 00 00

X : 9001h → Address

X : 0x009001 : 00 00 00

After execution

X : 9000h : 30 39 00 00

X : 9001h : 39 00 00 00

Date : 07-10-022 Expt. No. : 3(b) Page No. : 11

AIM:

To write an assembly language program to convert ASCII number into packed BCD

Component required:

→ Keiluvision Simulator

program

```
org 0000h
Sjmp 30h
org 30h
MOV r0, # 50h
MOV r1, # 70h
MOV r3, #03
again: mov a ,@ r0
anl a, # 0Fh
Swap a
mov r4, a
~~dec~~ inc r0
mov a, @r0
anl a, # 0Fh
add a, r4          djnz r3, again
MOV @r1, a         Sjmp $
inc r1             end
inc r0
```

Result:

Before execution

Address : d : 50h

D : 0x50 : 30 31 32 33 34 00 00

Address : d : 70h

D : 0x70 : 00 00 00 00 00 00

After execution

Address : d : 50h

D : 0x50 : 30 31 32 33 34 00 00

Address : d : 70h

D : 0x70 : 01 23 40 00 00

Date : 07-10-022 Expt. No. : 3.c[i] Page No. : 12

AIM:

Write an assembly language program to convert a ASCII number into Decimal.

Component required

→ Keiluvision simulator

program

```
MOV DPTR, # 9000H   // ASCII NO. To be converted to
                       decimal is stored in 9000h
MOV X A, @ dptr
Subb a, #30h        // The value 30 is decimal
inc DPTR
MOV x @ dptr, a
End
```

Result:

Before execution:

Address : x : 9000h

x: 0x009000 : 35 00 00 00 00

x: 0x00900A : 00 00 00 00 00

After execution:

x: 0x009000 : 35 05 00 00 00

x: 0x00901A : 00 00 00 00 00

Date:- 07-10-022 | Expt No:- 3.c [ii]

AIM:

To Convert a decimal number into ASCII

Component required:-

→ keiluvision Simulator

program

```
Mov dptr, #9000h
Mov x a, @dptr
add a, #30h
inc dptr
Mov x @dptr, a
end
```

Result:

Before execution

Address: x: 9000h

X: 0x009000 : 05 00 00 00 00

X: 0x00901A : 00 00 00 00 00

After execution

Address: x: 9000h

X: 0x00 9000 : 05 35 00 00 00

X: 0x00901A : 00 00 00 00 00

Date : 07-10-022 Expt. No. : 3.d[i] Page No. : 13

AIM:

Write an assembly language program to convert a binary (hex) number into decimal.

Component required → Keiluvision Simulator

program

```
Mov a, # 0feh
Mov b, # 0ah
div ab
Mov r0, b
Mov b, #0ah
div ab
Mov 30h, a
MOV a, b
Swap a.
orl a, r0
MOV 31h, a
end
```

Result:

Before execution:

Address : d: 30h

D: 0x30 : 00 00 00 00

D: 0x4B : 00 00 00 00

After execution:

Address : d: 30h

D: 0x30 : 02 54 00 00 00

D: 0x4B : 00 00 00 00 00

Date: 07-10-022 | Expt.NO: 3.d (ii)

AIM:

Write an assembly language program to convert a decimal number into ASCII

Component required:

→ keiluvision simulator

program

```
Mov a, # 95h
Mov b, # 10h
div ab
Mov R1, b
Mov b, # 0Ah
Mul ab
add a, R1
Mov 30h, a
end
```

Result:

Before execution:

Address : d: 30h

D: 0x30: 00 00 00 00

D: 0x4B: 00 00 00 00

After execution:

Address : d: 30h

D: 0x30: 5F 00 00 00 00

D: 0x4B: 00 00 00 00 00

[Experiment no: 4]

Date: 19-10-2022 Expt. No.: 04 Page No.: 14

AIM:

(a) Write an Assembly Level Code to flash an LED Connected at a specified output port.

Source Code:

```
        Org 000h
        MOV A, # 0FFH
BACK:   MOV P1, A
        ACALL DELAY
        CPL A
        SJMP BACK
DELAY:  MOV R0, # 8H
HERE 3: MOV R1, # 0FFH
HERE 2: MOV R2, # 0FFH
HERE:   DJNZ R2, HERE
        DJNZ R2, HERE 2
        DJNZ R0, HERE 3
        RET
```

Result:

Before Execution:

P7	P6	P5	P4	P3	P2	P1	P0
1	1	1	1	1	1	1	1

After Execution

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	0	0

Both the keys are toggling and the LED blink leaving some delay or gap between toggling.

[Expt No: 4(b)] [Date: 19-10-2022]

AIM:

Write an Assembly level code to flash an LED connected at a specified pin of an output port

Component required:

→ Keiluvision Simulator.

Program:

```
        ORG 0000H
AGAIN : SETB P2.0
        ACALL DELAY
        CLR P2.0
        ACALL DELAY
        SJMP AGAIN
DELAY : MOV R0, # 8H
HERE 3: MOV R1, # 0FFH
HERE 2: MOV R2, # 0FFH
HERE :  DJNZ R2, HERE
        DJNZ R1, HERE 2
        DJNZ R0, HERE 3
        RET
```

Result:

Before execution:

P7	P6	P5	P4	P3	P2	P1	P0
1	1	1	1	1	1	1	1

After Execution

P7	P6	P5	P4	P3	P2	P1	P0
1	1	1	1	1	1	1	0

only specified pin is toggling and that LED blinking leaving some delay or gap between toggling

Date : Expt. No. : Page No. : 15

[4(a) C-program]

Write C program to flash an LED connected to a specified port.

Source code:

```
# include <reg 51.h>
  void MsDelay (unsigned int);
   void main (void)
   {
     while (1)
      {
       P1 = 0X 55;
       MsDelay (250);
        P1 = 0XAA;
       MsDelay (250);
       }
   }
   void MSDelay (unsigned int itime)
    {
      unsigned int i, j;
      for (i=0; i=itime ; i++)
      for (j=0; j<1275 ; j++)
      }
```

[4(b) C-program]

8051 C-program to flash an LED connected to a specified pin.

Source Code:

```
#include <reg51.h>
  void MYBIT = P1.5;
  void MSDelay (unsigned int);
    void main (void)
    {
       unsigned int z;
    for (z=0; z<50000; z++)
    {
       MYBIT = 1;
       MSDelay (250);
        MYBIT = 0;
        MSDelay (250);
         }
    }
   void MSDelay (unsigned int itime)
   {
     unsigned int i, j;
     for (i=0; i<itime, i++)
     for (j=0; j<1275, j++)
     }
```

[Experiment no: 07]

Date : 31-10-2022 Expt. No. : 07 Page No. : 16

AIM:

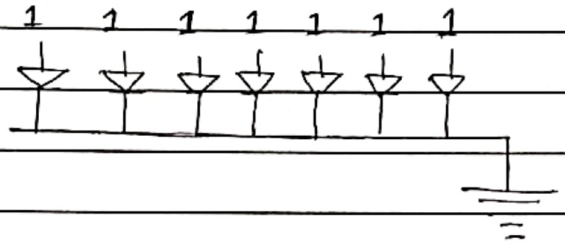
Interface BCD to 7 segment Display to display the BCD Numbers.

(a) Interface 7 segment to a 8051 to display the BCD numbers continuously with a suitable delay.

Common cathode configuration:

1 → segment - Turn ON

0 → segment - Turn OFF



→ All the negative cathode is connected with ground

→ All the positive ~~cathod~~ anode is connected with logic 1

Truth table

BCD	A	B	C	D	h	g	f	e	d	c	b	a
0	0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	0	1	1	0
2	0	0	1	0	0	1	0	1	1	0	1	1
3	0	0	1	1	0	1	0	0	1	1	1	1
4	0	1	0	0	0	1	1	0	0	1	1	0
5	0	1	0	1	0	1	1	0	1	1	0	1
6	0	1	1	0	0	1	1	1	1	1	0	1
7	0	1	1	1	0	0	0	0	0	1	1	1
8	1	0	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	0	1	1	0	0	1	1	1

C- program

```
# include (reg 51.h>
  void delay (unsigned int ms);
  # define seg-out P1
   void main (void)
   {
     while (1)
     {
      P1 = 0x00;
      seg-out = 0x3F;
      delay(500);
      seg-out = 0x06;
       delay (500);
       seg-out = 0x5B;
       delay (500);
       seg-out = 0x4F;
       delay (500);
```

```
Seg-out = 0x66;
delay (500);
Seg-out = 0x6D;
delay (500);
Seg-out = 0x70;
delay (500);
Seg-out = 0x07;
delay (500);
Seg-out = 0x7F;
delay (500);
Seg-out = 0x67;
delay (500);
}
}
void delay (unsigned int ms)
{
  unsigned int t1, t2;
for (t1=0; t1<ms; t1++)
{
  for (t2=0; t2<334; t2++)
  }
}
```

Result:

Port 1	Bits
P1: 0x3F	0 0 1 1 1 1 1 1
pins: 0x3F	0 0 1 1 1 1 1 1
P1: 0x06	0 0 0 0 0 1 1 0
pins: 0x06	0 0 0 0 0 1 1 0
P1: 0x5B	0 1 0 1 1 0 1 1
pins: 0x5B	0 1 0 1 1 0 1 1
P1: 0x4F	0 1 0 0 1 1 1 1
pins: 0x4F	0 1 0 0 1 1 1 1
P1: 0x66	0 1 1 0 0 1 1 0
pins: 0x66	0 1 1 0 0 1 1 0
P1: 0x6D	0 1 1 0 1 1 0 1
pins: 0x6D	0 1 1 0 1 1 0 1
P1: 0x7D	0 1 1 1 1 1 0 1
pins: 0x7D	0 1 1 1 1 1 0 1
P1: 0x07	0 0 0 0 0 1 1 1
pins: 0x07	0 0 0 0 0 1 1 1
P1: 0x7F	0 1 1 1 1 1 1 1
pins: 0x7F	0 1 1 1 1 1 1 1
P1: 0x67	0 1 1 0 0 1 1 1
pins: 0x67	0 1 1 0 0 1 1 1