Exercise 1: Assembly Language Experiments using Simulator

1.1
Aim: To write an assembly language program for LED Blink Using Keil simulator.

Steps:

1. **Create a New Project:** Open Keil IDE and create a new project. Go to "Project" in the menu bar, then select "New µVision Project...". Choose a location and give your project a name. Click "Save".
2. **Select Target Device:** In the dialog box that appears, select the appropriate target device for your simulation. This depends on the microcontroller model you're simulating. For example, if you're simulating an 8051 microcontroller, select the corresponding device (Honeywell -HT83C51) from the list.
3. **Create an Assembly Source File:** Right-click on the "Source Group" folder in the Project window and select "Add New Item to Group...". Choose "Assembly Source File" and give it a name, like "blink.asm".
4. **Write Assembly Code:** Open the assembly source file you created and write your LED blink assembly code.
5. Algorithm for the LED Blink is given below:

```
1. Set up initial values:
        ● Set register R4 to 05H.
2. Start of the loop:
        ● Set Port 1 (P1) to all high (11111111b).
        ● Call the subroutine DELAY.
        ● Set Port 1 (P1) to all low (00000000b).
        ● Jump back to the label HERE.
3. Subroutine DELAY:
        ● Set register R0 to 100.
        ● Start nested loop:
                ● Set register R1 to 200.
                ● Start inner loop:
                        ● Set register R2 to the address of the label AGAIN.
                        ● Decrement R2 and jump to AGAIN until R2 becomes zero.
                ● Decrement R4 and jump back to BACK1 until R4 becomes zero.
        ● Decrement R0 and jump back to BACK2 until R0 becomes zero.
        ● Return from the subroutine.
4. End of the code.
```

6. **Build the Project:** Click on the "Build" button in the toolbar or go to "Project" > "Build Target" to build the project. This assembles the source code into machine code.

7. **Simulate the Program:** After building the project successfully, you can simulate the program by clicking on the "Debug" button in the toolbar or going to "Debug" > "Start/Stop Debug Session". This will start the debugger and open the simulator window.

8. **Run the Simulation:** In the simulator window, you can start the simulation by clicking on the "Run" button or pressing F5. This will execute your assembly code, and you should see the LED blink simulation.

Program

```
ORG 00H
MOV R4,#05H
HERE: MOV P1,#11111111b
ACALL  DELAY
MOV P1,#00000000B
SJMP HERE
DELAY:      MOV R0,#100
BACK2:      MOV R1,#200
BACK1:      MOV R2,AGAIN
AGAIN:      DJNZ R2,AGAIN
            DJNZ R4,BACK1
            DJNZ R0,BACK2
RET
END
```

Result: Thus the assembly language program for LED blink is executed and the output is verified.

Exercise 2 Test data transfer between registers and memory

2.1

Aim:

To write an assembly language program for Data Transfer from register to memory

Algorithm:

1. Load immediate value:
    - Load the immediate value 12h into register A.
2. Store value:
    - Move the contents of register A into memory location 30h.
3. End of the code.

Program

```
Mov A,#12h
Mov 30h,A
End
```

2.2

Aim:

To write an assembly language program for Data Transfer from memory to register

Algorithm:

1. Move immediate value to memory:
    - Move the immediate value 12h to memory location 30h.
2. Move memory to register:
    - Move the contents of memory location 30h to register R1.
3. End of the code.

Program

```
Mov 30h,#12h
Mov R1,30h
End
```

2.3.

Aim:

To write an assembly language program for Data Transfer from memory to memory

Algorithm

1. Move immediate value to memory:
    - Move the immediate value 17D to memory location 40h.
2. Move memory to memory:
    - Move the contents of memory location 40h to memory location 50h.
3. End of the code.

Program:

```
MOV 40H,#17D
MOV 50H,40H
END
```

Result: Thus the assembly language programs for data transfer are executed and the output is verified.

Exercise 3: Perform ALU operations.

3.1

Aim:
To write an assembly language program to perform arithmetic operations addition and subtraction (30+10)-(20+2).

Algorithm:

1. Load immediate value to accumulator:
    - Load the immediate value 2D (decimal 45 in base 10) into the accumulator.
2. Add immediate value to accumulator:
    - Add the immediate value 20D (decimal 32 in base 10) to the accumulator.
3. Move accumulator to register:
    - Move the contents of the accumulator to register R1.
4. Load immediate value to accumulator:
    - Load the immediate value 30D (decimal 48 in base 10) into the accumulator.
5. Add immediate value to accumulator:
    - Add the immediate value 10D (decimal 16 in base 10) to the accumulator.
6. Subtract with borrow from register:
    - Subtract the contents of register R1 from the accumulator using the "subtract with borrow" (SUBB) instruction. The borrow is automatically set based on the result of the previous operation.
7. Move accumulator to register:
    - Move the contents of the accumulator to register R2.
8. End of the code

Program:

```
MOV A,#2D
ADD A,#20D
MOV R1,A
MOV A,#30D
ADD A,#10D
SUBB A,R1
MOV R2,A
END
```

3.2
Aim:
To write an assembly language program to perform arithmetic operations multiplication and division (20*4)/2.

Algorithm:

1. Load immediate value to accumulator:
    - Load the immediate value 20D (decimal 32 in base 10) into the accumulator.
2. Load immediate value to register:
    - Load the immediate value 4D (decimal 77 in base 10) into register B.
3. Multiply accumulator and register:

  - Multiply the contents of the accumulator and register B, storing the result in register A.
4. Move accumulator to register:
   - Move the contents of the accumulator to register R1.
5. Move register to register:
   - Move the contents of register B to register R2.
6. Load immediate value to register:
   - Load the immediate value 2D (decimal 45 in base 10) into register B.
7. Divide accumulator by register:
   - Divide the contents of the accumulator by register B, storing the quotient in register A and the remainder in register B.
8. Move accumulator to register:
   - Move the contents of the accumulator to register R3.
9. Move register to register:
   - Move the contents of register B to register R4.
10. End of the code.

Program:

```
MOV A,#20D
MOV B,#4D
MUL AB
MOV R1,A
MOV R2,B
MOV B,#2D
DIV AB
MOV R3,A
MOV R4,B
END
```

3.3

Aim
To write an assembly language program to perform Logical operations.

Algorithm:

1. Load immediate value to accumulator:
   - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
2. Load immediate value to register:
   - Load the immediate value 67H (hexadecimal 103 in base 10) into register R0.
3. Logical AND between accumulator and register:
   - Perform a bitwise logical AND operation between the accumulator and register R0. Store the result in the accumulator.
4. Move accumulator to memory:
   - Move the contents of the accumulator to memory location 20H.
5. Load immediate value to accumulator:
   - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
6. Logical OR between accumulator and register:
   - Perform a bitwise logical OR operation between the accumulator and register R0. Store the result in the accumulator.
7. Move accumulator to memory:

- Move the contents of the accumulator to memory location 21H.

8. Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
9. Exclusive OR between accumulator and register:
    - Perform a bitwise exclusive OR operation between the accumulator and register R0. Store the result in the accumulator.
10. Move accumulator to memory:
    - Move the contents of the accumulator to memory location 22H.
11. Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
12. Complement accumulator:
    - Perform a bitwise complement operation on the accumulator (NOT logic).
13. End of the code.

Program:

```
MOV A,#45H
MOV R0,#67H
ANL A,R0
MOV 20H,A
MOV A,#45H
ORL A,R0
MOV 21H,A
MOV A,#45H
XRL A,R0
MOV 22H,A
MOV A,#45H
CPL A   ; NOT LOGIC
END
```

3.4

Aim:
To write an assembly language program to find the largest number in an array.

Algorithm:

.

1. Initialize register R0 with value 10.

2. Initialize the data pointer (DPTR) with the initial address of ROM memory (600h).
3. Load the first byte from ROM memory using move code (MOVC) instruction and store it in accumulator A.
4. Copy the value from accumulator A to register B.
5. Start a loop labeled as "back":
    - Increment the data pointer.
    - Clear the accumulator.
    - Load the next byte from ROM memory using move code (MOVC) instruction and store it in accumulator A.
    - Compare the value in accumulator A with the value in register B.
    - If they are not equal, jump to the label "check".
    - If they are equal, continue with the loop.

6. Label "check":
    - Check the carry flag. If the carry flag is set (indicating A > B), jump to the label "next".
    - If the carry flag is not set (indicating A <= B), copy the value from accumulator A to register B.
7. Label "next":
    - Decrement register R0.
    - If R0 is not zero, jump back to the label "back".
8. Store the value from register B into memory location 60h.
9. Initialize the ROM memory starting at address 600h with the following values: 34h, 54h, 73h, 12h, 31h, 99h, 13h, 11h, 09h, 76h.
10. End of the code.

Program:

```
MOV R0,#10
MOV dptr,#600h    ; initial address of rom memory
movc a,@a+dptr    ; 1st number from ROM memory
mov b,a
back:  inc dptr
clr a
movc a,@a+dptr
cjne a,b,check
check:   jc next    ; if c=0 a<b or c=1, b>a
    mov b,a
    next: djnz r0,back
    mov 60h,b   ; biggest number saved in 60h
    org 600h
       db 34h,54h,73h,12h,31h,99h,13h,11h,09h,76h
          end
```

Exercise 4: Write Basic and arithmetic Programs Using Embedded C.

4.1
Aim:
To write embedded C code for the 8051 microcontroller to convert 11111111 to decimal and display the digits on Po,P1,P2 using the Keil C51 compiler.

Algorithm:

1. **Include Header File**:
   - #include <reg51.h>: This includes the header file necessary for programming the 8051 microcontroller family.
2. **Main Function**:
   - void main(): This is the main function where the program execution starts.
3. **Variable Declaration**:
   - unsigned char x, binbyte, d1, d2, d3;: These are unsigned char variables used for storing data.
4. **Assign Value to binbyte**:
   - binbyte = 0xFF;: This assigns the value 0xFF (255 in decimal) to the variable binbyte.
5. **Perform Division and Modulus Operations**:
   - x = binbyte / 10;: This performs integer division of binbyte by 10 and stores the result in x.
   - d1 = binbyte % 10;: This calculates the remainder of binbyte divided by 10 and stores it in d1.
   - d2 = x % 10;: This calculates the remainder of x divided by 10 and stores it in d2.
   - d3 = x / 10;: This performs integer division of x by 10 and stores the result in d3.
6. **Output to Ports**:
   - P0 = d1;, P1 = d2;, P2 = d3;: These statements output the values of d1, d2, and d3 to the respective ports P0, P1, and P2.
7. **End of Program**:
   - The main function does not have a return statement, and thus it implicitly returns control to the system.

Program:

```
#include <reg51.h>
void main()
{
   unsigned char x,binbyte,d1,d2,d3;
   binbyte=0xFF;
   x=binbyte/10;
   d1=binbyte%10;
   d2=x%10;
   d3=x/10;
   P0=d1;
   P1=d2;
   P2=d3;
}
```

4.2

Aim:
To write a 8051 Embedded C program to perform addition and subtraction.

Algorithm:

1. **Include Header File**:
   - Include the header file <reg51.h>, which contains definitions specific to the 8051 microcontroller family.
2. **Main Function**:
   - Define the main function (void main()) where the program execution starts.
3. **Variable Declaration**:
   - Declare three unsigned char variables: num1, num2, and result, representing the two numbers to be added and subtracted, and the result of the operations, respectively.
4. **Initialization**:
   - Initialize num1 with the value 50 and num2 with the value 20.
5. **Addition Operation**:
   - Perform addition of num1 and num2 and store the result in the variable result.
6. **Output of Addition Result**:
   - Output the result of the addition operation (result) to port P0. This assumes that P0 is connected to an output device for display.
7. **Subtraction Operation**:
   - Perform subtraction of num2 from num1 and store the result in the variable result.
8. **Output of Subtraction Result**:
   - Output the result of the subtraction operation (result) to port P1. This assumes that P1 is connected to an output device for display.
9. **Infinite Loop**:
   - Enter an infinite loop (while(1)) to prevent the program from exiting.
10. **End of the Program**:
    - End of the main() function.

Program:

```
#include <reg51.h>

void main() {
   unsigned char num1 = 50;   // First number
   unsigned char num2 = 20;   // Second number
   unsigned char result;      // Variable to store the result

   // Addition
   result = num1 + num2;

   // Output result of addition
   P0 = result;  // Assuming P0 is connected to an output device for display

   // Subtraction
   result = num1 - num2;
```

```
    // Output result of subtraction
    P1 = result;  // Assuming P1 is connected to an output device for display

    while(1);  // Infinite loop to prevent the program from exiting
}
```

Ex. 9: Setup a cloud platform to log the data
Aim

To set up a ThingSpeak cloud to log data from IoT devices and sensors, analyze it, and visualize it in real-time.
Steps:
1.Sign Up /Login to ThinkSpeak

Go to the ThingSpeak website (https://thingspeak.com/) and sign up for a new account or log in if you already have one.
2.Create a New Channel

- After logging in, click on the "Channels" tab at the top of the page.
- Click on the “New Channel” button.
- Enter the necessary information such as the name, description, and field labels for your channel.
- Configure additional settings like the number of fields, update rate, and privacy settings according to your requirements.
- Click on the "Save Channel" button to create the channel

3. Get Write API Key

- Once the channel is created, click on the "API Keys" tab.
- You will find the "Write API Key" listed there. Copy this key as you will need it to send data to your channel.

4. Configure Data Source (Device)

- Configure your IoT device or sensor to send data to ThingSpeak.
- Use the Write API Key obtained in the previous step to authenticate data transmission
    - Depending on your device and communication protocol (HTTP, MQTT, etc.), configure it to send data to ThingSpeak using the appropriate method.

5.Send Data to ThingSpeak

- Start sending data from your device to ThingSpeak using the Write API Key.
- You can send data using HTTP requests, MQTT, or other supported protocols.
- Make sure to send the data in the format expected by ThingSpeak, typically as HTTP GET or POST requests with parameters for each field.

6.View and Analyze Data:

- Once data is being sent to your ThingSpeak channel, you can view and analyze it using the ThingSpeak web interface.
- Go to your channel's page and click on the "Charts" or "Visualizations" tab to see your data plotted on graphs.

Result:
Thus the setting up of ThingSpeak cloud to log data is done.