|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics form for Microprocessor Systems and Interfacing** | | | | | | | | | | | | | | |
| **Lab #:** | | | **#06** | | | | | | | | | | | |
| **Lab Title:** | | | **Stack Operations and Rotate Instructions** | | | | | | | | | | | |
| **Submitte** | **d by:** |  |  |  |  |  |  |  |  |  |  | |  |  |
| **Name** | | | | | | | | | | | | **Registration #** | | |
| ***Saad Rafique***  ***Wajeh Ul Hassan*** | | | | | | | | | | | | ***FA19-BCE-020***  ***FA19-BCE-024*** | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics name & number** | | | | | | | **Marks** | | | |
| **In-Lab** | | **Post-Lab** | |
| **Engineering Knowledge** | ***R2: Use of Engineering Knowledge and follow Experiment Procedures:*** *Ability to follow experimental procedures, control variables, and record procedural steps on lab report.* | | | | | |  | |  | |
| ***R3: Interpretation of Subject Knowledge:***  *Ability to interpret and explain mathematical and/or visual forms, including equations, diagrams, graphics, figures and tables.* | | | | | |  | |  | |
| **Problem Analysis** | ***R4: Limitations and Implications:***  *Ability to point out*  *limitations and implications of Hardware and software Components.* | | | | | |  | |  | |
| ***R5: Data/Evidence Measurements:***  *Ability to record raw data / evidence.* | | | | | |  | |  | |
| ***R6: Experimental Data Analysis:***  *Ability to interpret findings, compare them to values in the literature, identify weaknesses and limitations.* | | | | | |  | |  | |
| **Design** | **R7: Implementing Design Strategy:** Ability to execute a solution taking into consideration design requirements and pertinent contextual elements.  [Block Diagram/Flow chart/Circuit Diagram] | | | | | |  | |  | |
| ***R8: Best Coding Standards:***  *Ability to follow the coding standards and programming practices.* | | | | | |  | |  | |
| **Modern Tools Usage** | ***R11: Tools Evaluation:***  *Ability to simulate the experiment and then using hardware tools to verify the results.* | | | | | |  | |  | |
| **Rubrics #** | R2 | R3 | R4 | R5 | R6 | R7 | | R8 | | R11 |
| **In –Lab** |  |  |  |  |  |  | |  | |  |
| **Post- Lab** |  |  |  |  |  |  | |  | |  |

**LAB#6**

**Stack Operations and Rotate Instructions**

**Objectives:**

* To be familiar with rotate instruction.
* To understand concept of pushing and popping in stack along with stack pointer.

**Introduction:**

**Stack Memory Allocation in 8051 Microcontroller**

The stack is an area of random-access memory (RAM) allocated to hold temporarily all the parameters of the variables. The stack is also responsible for reminding the order in which a function is called so that it can be returned correctly. Whenever the function is called, the parameters and local variables associated with it are added to the stack (PUSH). When the function returns, the parameters and the variables are removed (“POP”) from the stack. This is why a program’s stack size changes continuously while the program is running.

**ROTATE INSTRUCTION**

In many applications there is a need to perform a bitwise rotation of an operand. In the 8051 the rotation instructions RL, RR, RLC, and RRC are designed specifically for that purpose. They allow a program to rotate the accumulator right or left. We explore the rotate instructions next since they are widely used in many different applications. In the 8051, to rotate a byte the operand must be in register A. There are two types of rotations. One is a simple rotation of the bits of A, and the other is a rotation through the carry.

**In-Lab Tasks**

**Task#1**

Show the stack and the stack pointer for each line of the following:

**Solution:**

|  |  |
| --- | --- |
| **Instruction** | **Stack Pointer** |
| Org 0 | 07H |
| MOV SP,#70H | 70H |
| MOV R5,#66H | 70H |
| MOV R2,#7FH | 70H |
| MOV R7,#5DH | 70H |
| PUSH 5 | 71H |
| PUSH 2 | 72H |
| PUSH 7 | 73H |
| CLR A | 73H |
| MOV R2,A | 73H |
| MOV R7,A | 73H |
| POP 7 | 72H |
| POP 2 | 71H |
| POP 5 | 70H |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Initial Empty Stack** | | **1ST PUSH** | **2ND PUSH** | **3RD PUSH** |
| **74H** | - | - | - | - |
| **73H** | - | - | - | 5DH |
| **72H** | - | - | 7FH | 7FH |
| **71H** | - | 66H | 66H | 66H |
| Sp=70H | | Sp=71H | Sp=72H | Sp=73H |

**Task#2**

Write a program to transfer value 41H serially (one bit at a time) via pin P2.1. Connect LED at pin 2.1 and insert the delay to observe it. Put two highs at the start and end of the data. Send the byte LSB first. Also simulate the circuit in Proteus.

**Task description:**

In this task we are passing 41H serially (one bit at a time) via pin P2.1. An LED is connected at pin 2.1 and insert the delay to observe it. Putting two highs at the start and end of the data. Send the byte LSB first by using rotate instruction. Also simulate the circuit in Proteus.

|  |
| --- |
| **Code:** |
| ORG 0H  MOV A, #41H  SETB P2.1 ;high  SETB P2.1 ;high  MOV R5, #8  HERE: RRC A  ACALL DELAY  MOV P2.1,C ;send the carry bit to P2.1  ACALL DELAY  DJNZ R5, HERE  SETB P2.1 ;high  SETB P2.1 ;high  MOV R6, 255  DELAY:  MOV R7, 255  DELAY1:  DJNZ R7, DELAY1  DJNZ R6, DELAY  RET  END |

**Dry Run:**

41H=01000001

P2.1=1 (LED ON)

P2.1=0 (LED OFF)

P2.1=0 (LED OFF)

P2.1=0 (LED OFF)

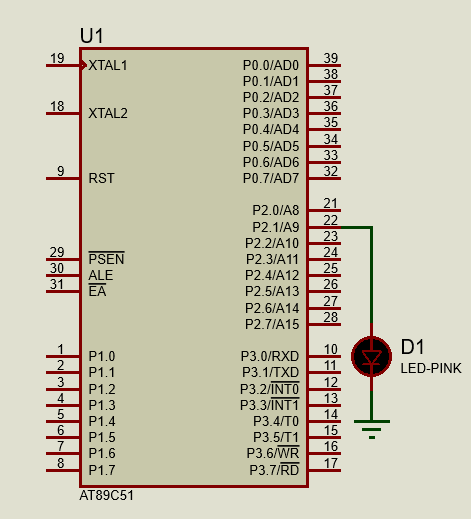
P2.1=0 (LED OFF)

P2.1=0 (LED OFF)

P2.1=1 (LED ON)

P2.1=0 (LED OFF)

**Simulation:**



**Results and Discussion:**

Using rotate instruction LSB of 41H are passed at p2.1 pin that turns LED at pin p2.1 on and off. The LED turns on for only two times when bits of 41H are passed through p2.1.

**Post-Lab Task**

**Task#1**

This program stores two values in A and B. It finds remainder of division operation between A and B, and stores it to B. A’s value need not to be altered. Clearly observe all the steps and write what happens.

**Task Description:**

This program stores two values in A and B. It finds remainder of division operation between A and B, and stores it to B. A’s value need not to be altered.

|  |
| --- |
| **Code:** |
| ORG 0H  COUNT EQU 30  MOV A, #COUNT  MOV B, #4  DIV AB  END |

|  |  |
| --- | --- |
| **Code and Dry Run** | **Register Status** |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | COUNT EQU 30  MOV A, #COUNT  MOV B, #4 | A=00011110  B=00000100 | |  |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | DIV AB | A/B | |  |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | MOV A, #67H | A=01100111  CY=1  P=1  AC=1  OV=1  PSW=11010101 | |  |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | ADD A, #0BEH | A=A+79H  A=01100111+01111001  A=11100001  CY=0  P=0  AC=1  OV=1  PSW=01000100 | | MOV R1, A | R1=11100001 | |  |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | MOV A, #35H | A=00110101  CY=0  P=0  AC=1  OV=1  PSW=01000100 | |  |
| |  |  | | --- | --- | | **Instruction** | **Dry Run** | | ADD A, #0BEH | A=A+A6H  A=00110101+10100110  A=11011011  CY=0  P=0  AC=0  OV=0  PSW=00000000 | | MOV R2, A | R2=11011011 | |  |

**Results and Discussion:**

The most significant nibble of the result is stored in R2 and the least significant nibble is stored in R0 and the whole result is stored in R0, R1 and R2, this because we cannot store 24 bit result in any of the register of AT89C51 MC.

**Task#2**

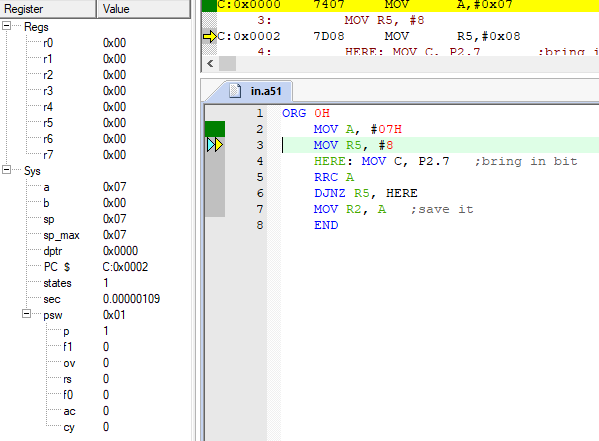
Write a program to bring in a byte of data serially one bit at a time via pin P2.7 and save it in register R2. The byte comes in with the LSB first.

**Task description:**

In this task we are required to bring in a byte of data serially one bit at a time via pin P2.7 and save it in register R2. The byte comes in with the LSB first. This can be done by using RRC command.

|  |
| --- |
| **Code:** |
| ORG 0H  MOV R5, #8  HERE: MOV C, P2.7 ;bring in bit  RRC A  DJNZ R5, HERE  MOV R2, A ;save it  END |

**Output:**



**Conclusion:**

At the end of this lab we are now familiar with rotate instructions & understood concept of pushing and popping in stack along with stack pointer.