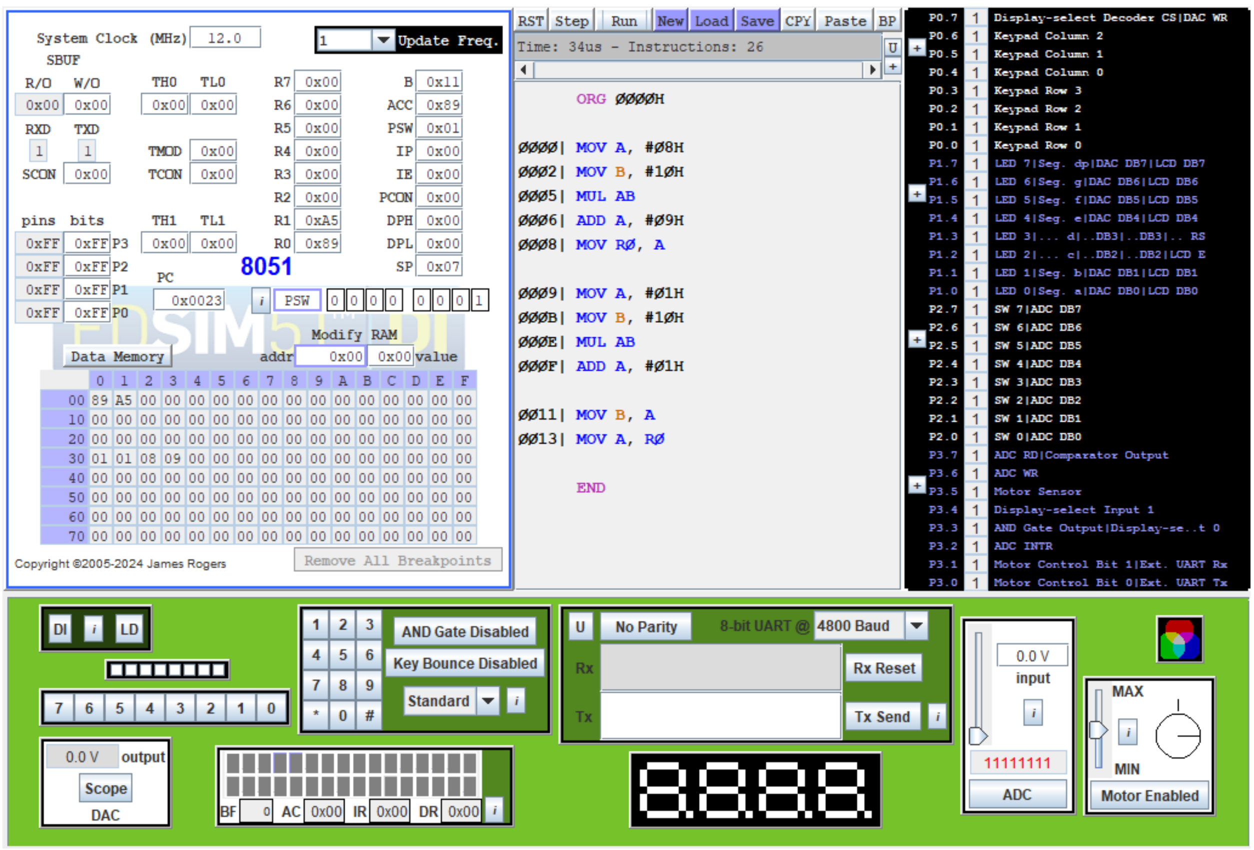
NAME: Rutva Sawarkar
PRN: 24070521189
SEM: 4th , C

MICROCONTROLLERS AND EMBEDDED SYSTEMS

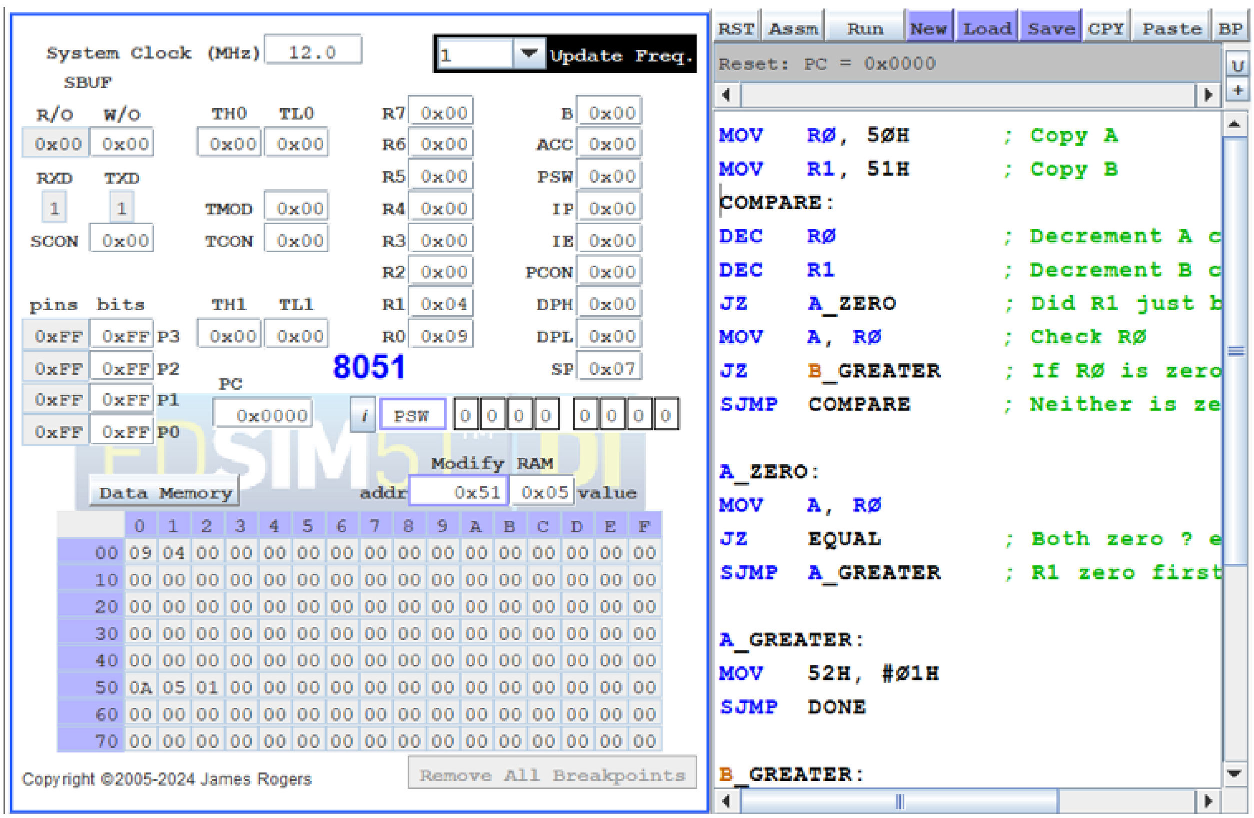
Q.1 Write an 8051 Assembly Language Program (ALP) to generate the last four digits of your PRN using any arithmetic instructions. The program should not directly load the complete PRN number as an immediate value. Instead, it must use appropriate arithmetic operations such as ADD, MUL, or INC to form the number logically. The final result must be stored in the Accumulator register (AX). For example, if a student's PRN is 24070521211, the last four digits are 1211, and the value 1211 should be available in AX at the end of program execution.



NAME: Rutva Sawarkar
PRN: 24070521189

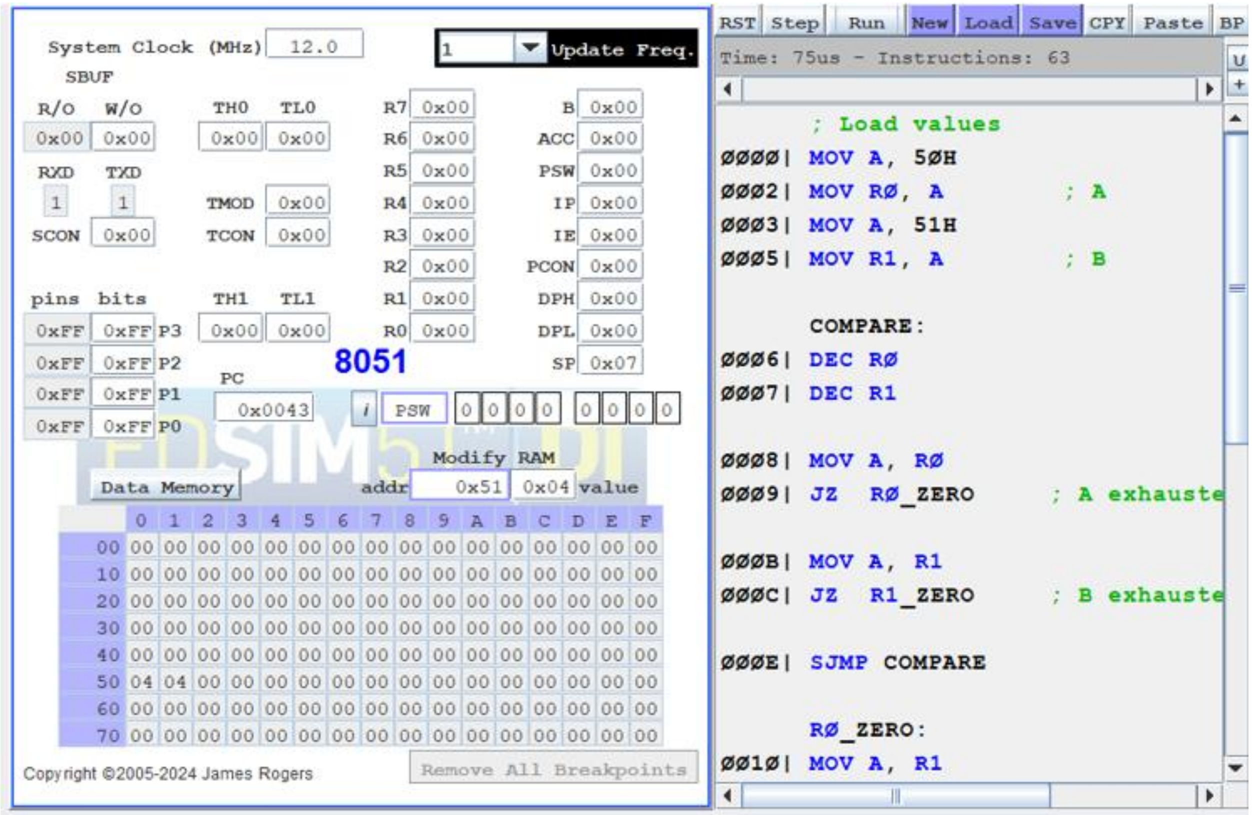
Q.2 Execute an 8051-assembly language program for a safety-certified system in which the instructions CJNE, DJNZ, and SUBB are not permitted. Two unsigned numbers are stored in internal RAM locations 50H and 51H. The program must compare these two numbers using only the allowed instruction set (MOV, INC, DEC, JZ, JNZ, CLR, SETB, ANL, ORL) and store the comparison result in a register or memory location such that 01H indicates the value at 50H is greater than the value at 51H, 00H indicates both values are equal, and FFH indicates the value at 50H is less than the value at 51H. The program should be simulated for all three possible cases (A > B, A = B, A < B), and the solution must clearly explain how flag behavior (especially the Zero flag) is utilized to achieve comparison under the given instruction constraints.

CASE1:

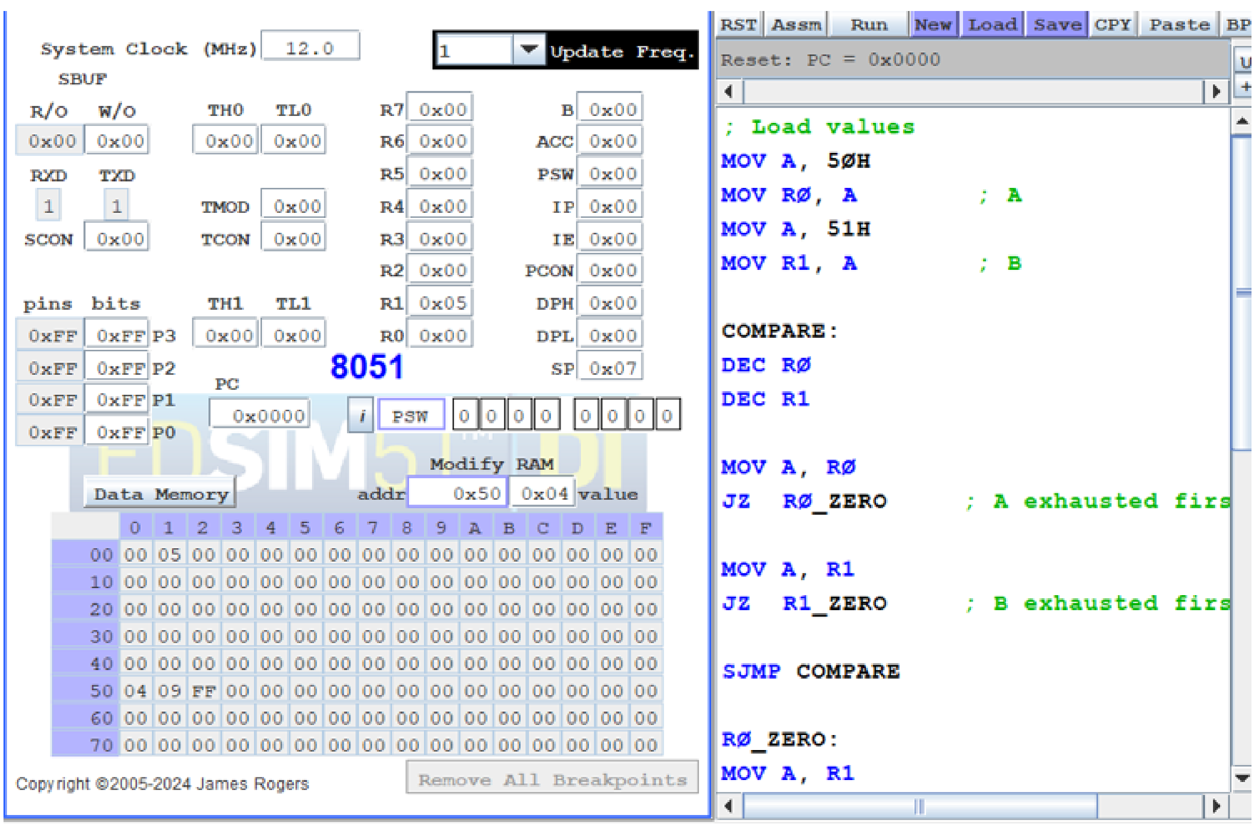


CASE 2:

NAME: Rutva Sawarkar **PRN:** 24070521189



Case 3:

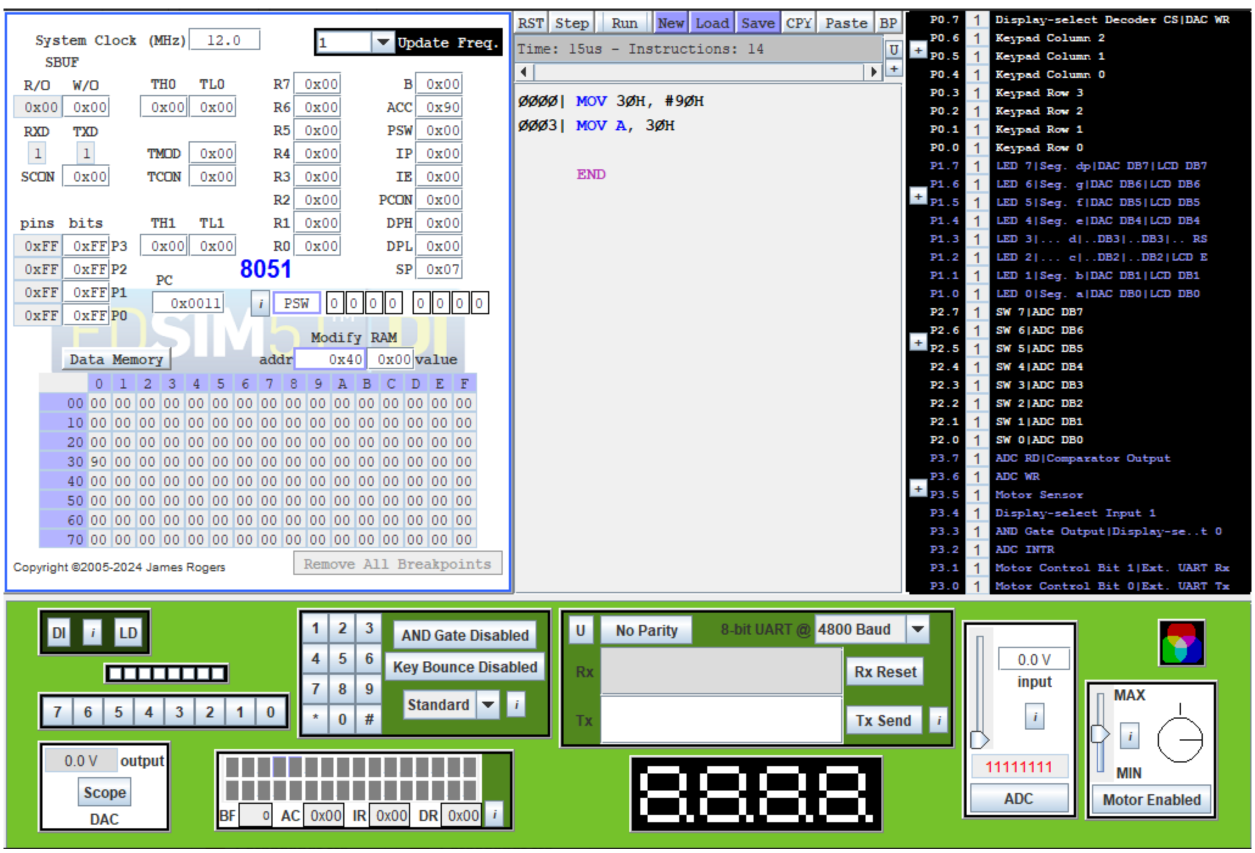


NAME: Rutva Sawarkar

PRN: 24070521189

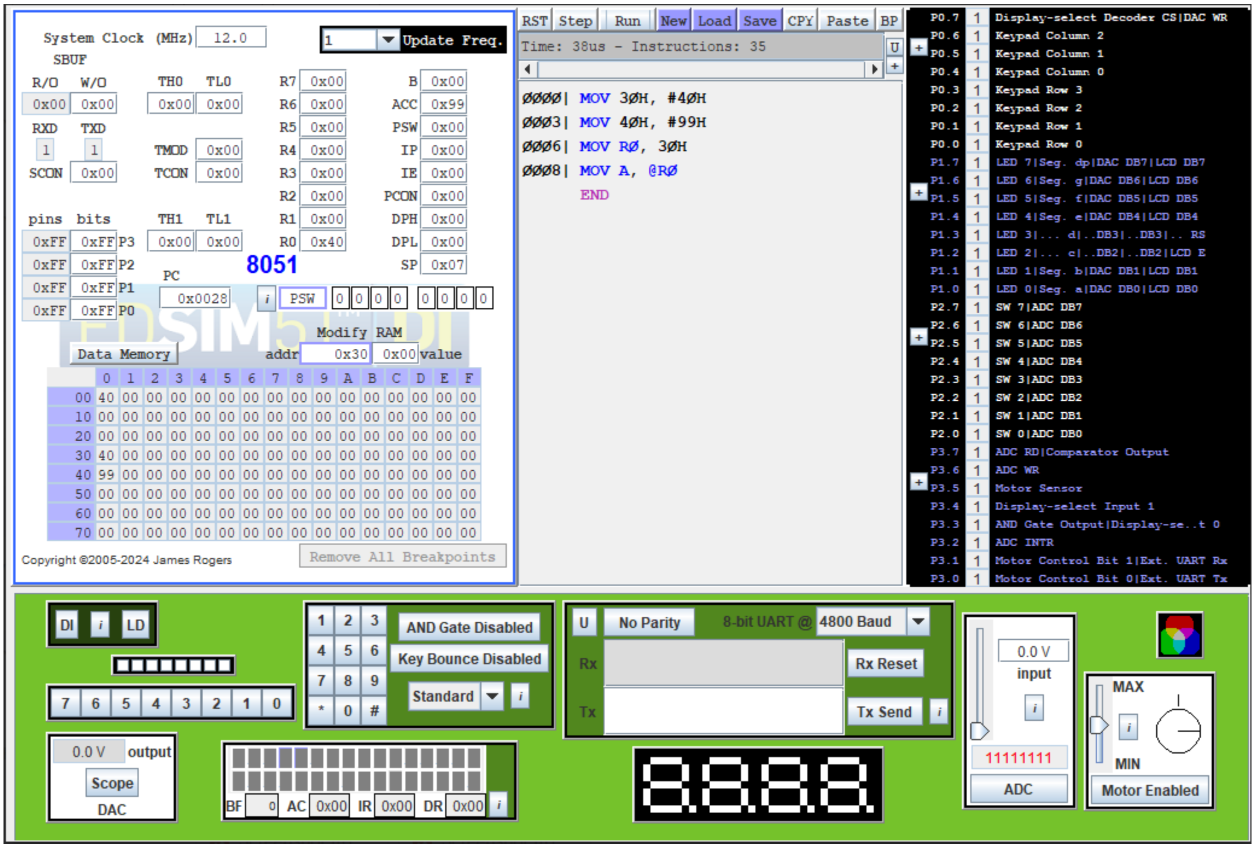
Q.3 A student claims that two assembly programs are equivalent because both access the same RAM address; however, this claim is incorrect due to the difference in addressing modes. In this case study, write two short assembly programs—one using direct addressing and the other using indirect addressing—such that both reference the same RAM location. Using an appropriate initial RAM configuration, demonstrate a situation where the outputs of the two programs differ even though the base address is the same. Support the observation with register and RAM snapshots from simulation, and explain that the difference arises because direct addressing accesses the data stored at the given address, whereas indirect addressing treats the contents of that address as a pointer to another memory location, leading to different data being fetched and hence different outputs.

Direct Addressing mode



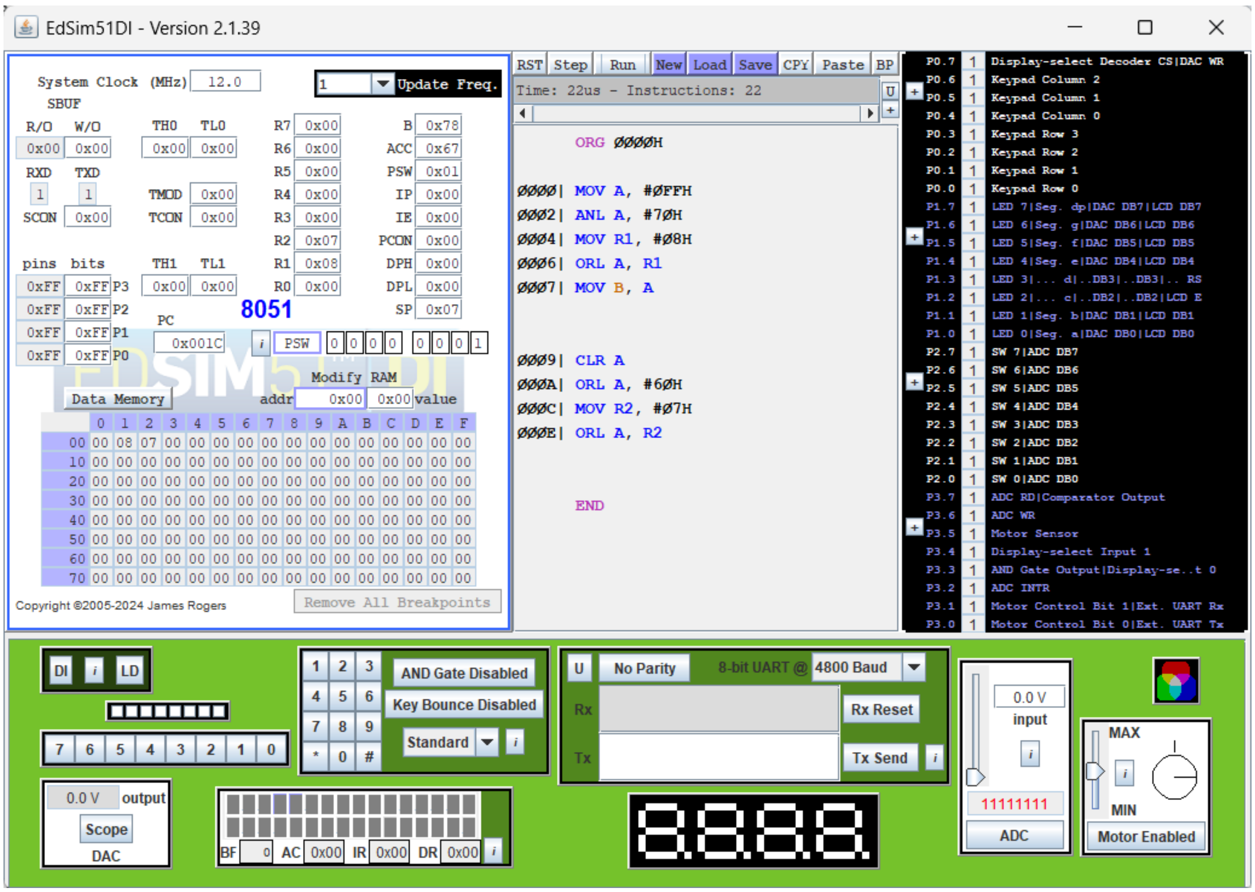
NAME: Rutva Sawarkar
PRN: 24070521189

Indirect addressing mode



NAME: Rutva Sawarkar
PRN: 24070521189

Q.4 Write an 8051 Assembly Language Program in which you must use logical instructions to construct a numeric result. Using multiple logical instructions such as ANL, ORL, and CLR, generate the last four digits of your own mobile number through a suitable sequence of operations (you may split the digits and combine them logically as required). Do not directly load the complete 4-digit number as an immediate value. The program should use more than one logical instruction, and at the end of execution the Accumulator (A) must contain the last four digits of your mobile number. Simulate the program and verify that the final value in the Accumulator matches your mobile number's last four digits.



NAME: Rutva Sawarkar
PRN: 24070521189

Q.5 An embedded logger stores event codes in internal RAM from 40H to 5FH, but due to strict memory limitations the data must be compacted in-place without using any additional RAM or the stack. Write an assembly language program that scans the memory range 40H–5FH using only indirect addressing, removes all occurrences of the value FFH, shifts the remaining valid data bytes to the left to eliminate gaps, and fills the unused memory locations at the end of the range with 00H. Execute the program to show the RAM contents before and after execution, and clearly explain the pointer movement logic used to identify valid data, shift it correctly, and overwrite invalid entries under the given constraints.