

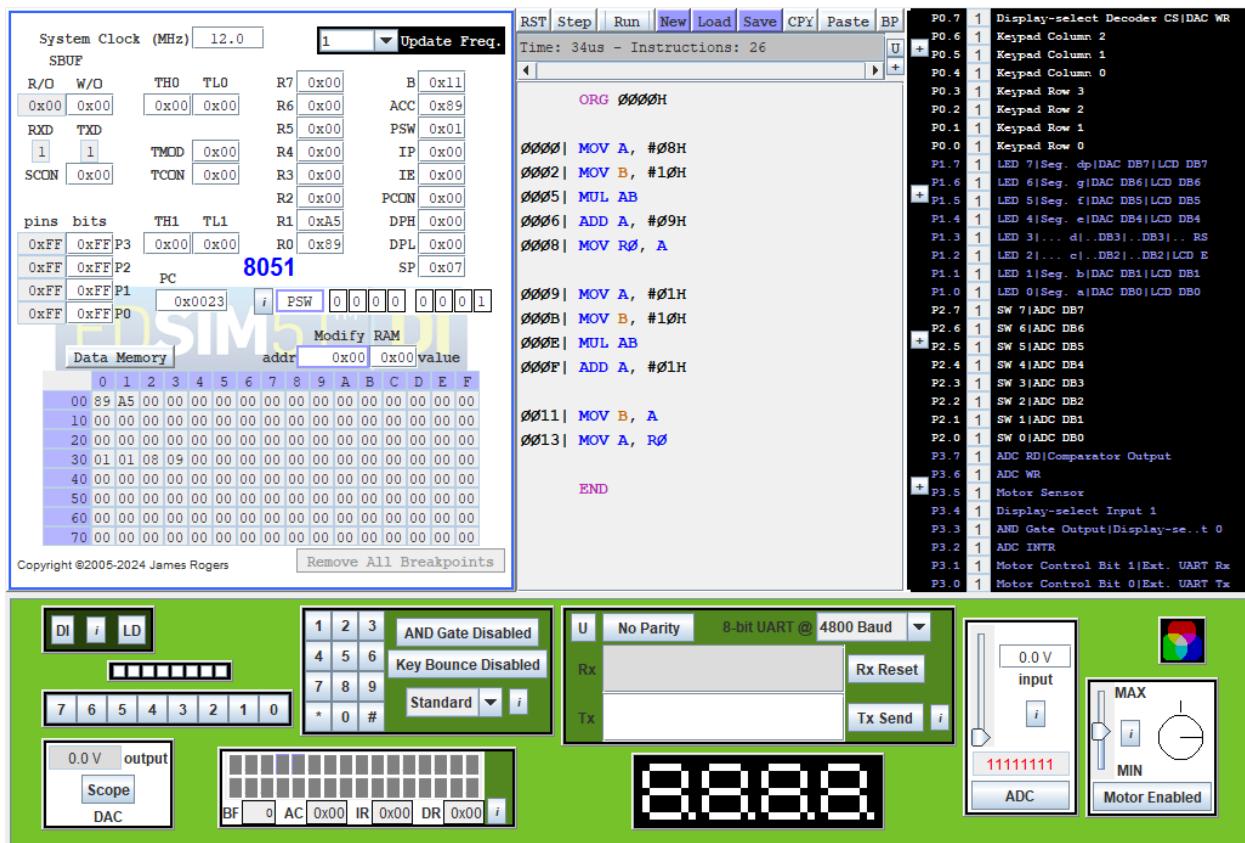
**NAME:** Rutva Sawarkar

**PRN:** 24070521189

**SEM:** 4<sup>th</sup>, 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:**

The screenshot shows the F1 SIMH 8051 simulator interface. On the left, the 8051 chip is shown with various registers and pins. The CPU section displays the assembly code. The memory dump section shows the state of RAM, with address 51 having a value of 05. The assembly code is as follows:

```
RST Assm Run New Load Save CPY Paste BP
Reset: PC = 0x0000
MOV R0, 50H ; Copy A
MOV R1, 51H ; Copy B
COMPARE:
DEC R0 ; Decrement A
DEC R1 ; Decrement B
JZ A_ZERO ; Did R1 just become zero?
MOV A, R0 ; Check R0
JZ B_GREATER ; If R0 is zero
SJMP COMPARE ; Neither is zero

A_ZERO:
MOV A, R0
JZ EQUAL ; Both zero? equal?
SJMP A_GREATER ; R1 zero first

A_GREATER:
MOV 52H, #01H
SJMP DONE

B_GREATER:

```

Copyright ©2005-2024 James Rogers

Remove All Breakpoints

**CASE 2:**

NAME: Rutva Sawarkar PRN: 24070521189

System Clock (MHz) 12.0      1 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x00	0x00	0x00	R6	0x00	ACC	0x00
RXD	TXD			R5	0x00	PSW	0x00
1	1	TMOD	0x00	R4	0x00	IP	0x00
SCON	0x00			R3	0x00	IE	0x00
		TCON	0x00	R2	0x00	PCON	0x00
pins	bits	TH1	TL1	R1	0x00	DPH	0x00
0xFF	0xFF	P3	0x00	R0	0x00	DPL	0x00
0xFF	0xFF	P2	PC			SP	0x07
0xFF	0xFF	P1	0x0043	i	PSW	0 0 0 0 0 0 0 0 0	
0xFF	0xFF	P0					

8051

Modify RAM

Data Memory      addr 0x51 0x04 value

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	04	04	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers      Remove All Breakpoints

Time: 75us - Instructions: 63

```

; Load values
0000| MOV A, 50H
0002| MOV R0, A          ; A
0003| MOV A, 51H
0005| MOV R1, A          ; B

COMPARE:
0006| DEC R0
0007| DEC R1

0008| MOV A, R0
0009| JZ R0_ZERO          ; A exhausted

000B| MOV A, R1
000C| JZ R1_ZERO          ; B exhausted

000E| SJMP COMPARE

R0_ZERO:
0010| MOV A, R1

```

Case 3:

System Clock (MHz) 12.0      1 Update Freq.

SBUF

R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x00	0x00	0x00	R6	0x00	ACC	0x00
RXD	TXD			R5	0x00	PSW	0x00
1	1	TMOD	0x00	R4	0x00	IP	0x00
SCON	0x00			R3	0x00	IE	0x00
		TCON	0x00	R2	0x00	PCON	0x00
pins	bits	TH1	TL1	R1	0x05	DPH	0x00
0xFF	0xFF	P3	0x00	R0	0x00	DPL	0x00
0xFF	0xFF	P2	PC			SP	0x07
0xFF	0xFF	P1	0x0000	i	PSW	0 0 0 0 0 0 0 0 0	
0xFF	0xFF	P0					

8051

Modify RAM

Data Memory      addr 0x50 0x04 value

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	05	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	04	09	FF	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers      Remove All Breakpoints

Reset: PC = 0x0000

```

; Load values
MOV A, 50H
MOV R0, A          ; A
MOV A, 51H
MOV R1, A          ; B

COMPARE:
DEC R0
DEC R1

MOV A, R0
JZ R0_ZERO          ; A exhausted first

MOV A, R1
JZ R1_ZERO          ; B exhausted first

SJMP COMPARE

R0_ZERO:
MOV A, R1

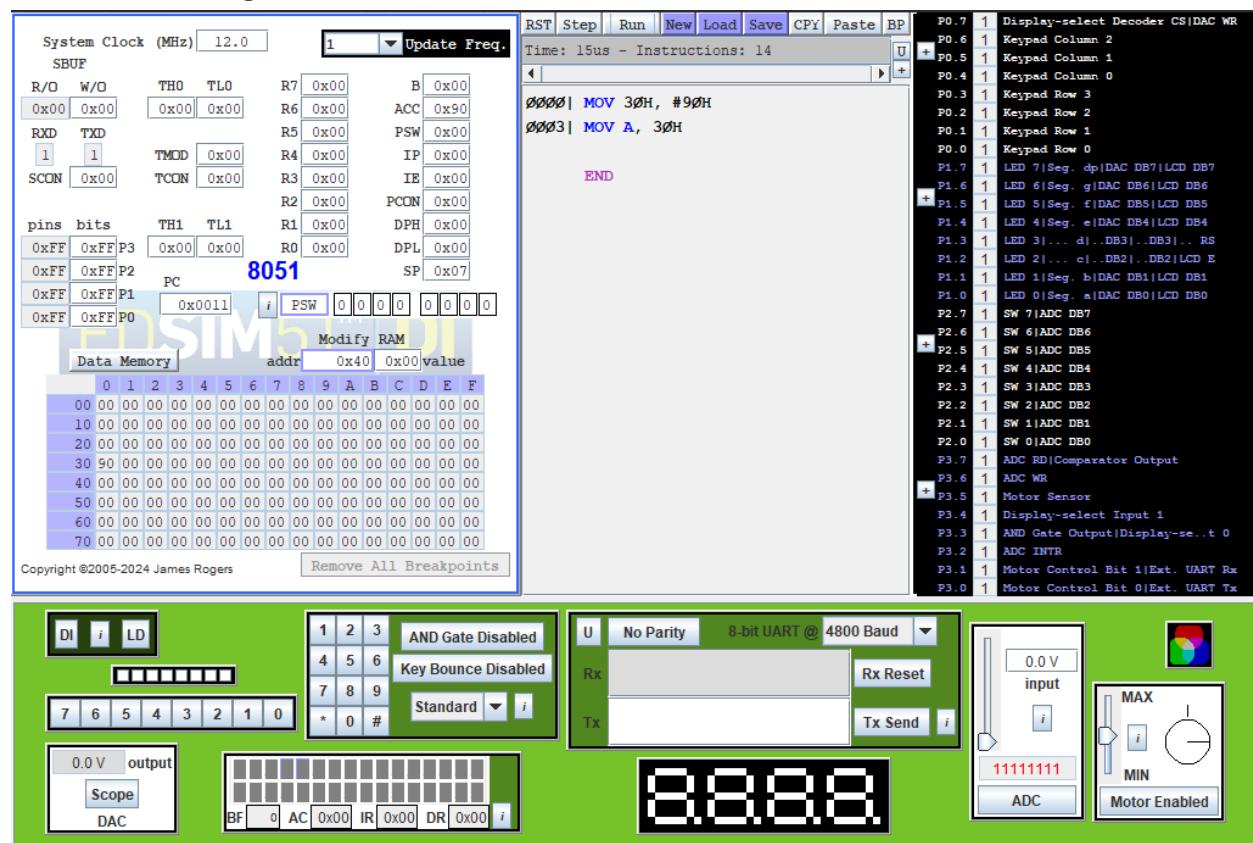
```

**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.

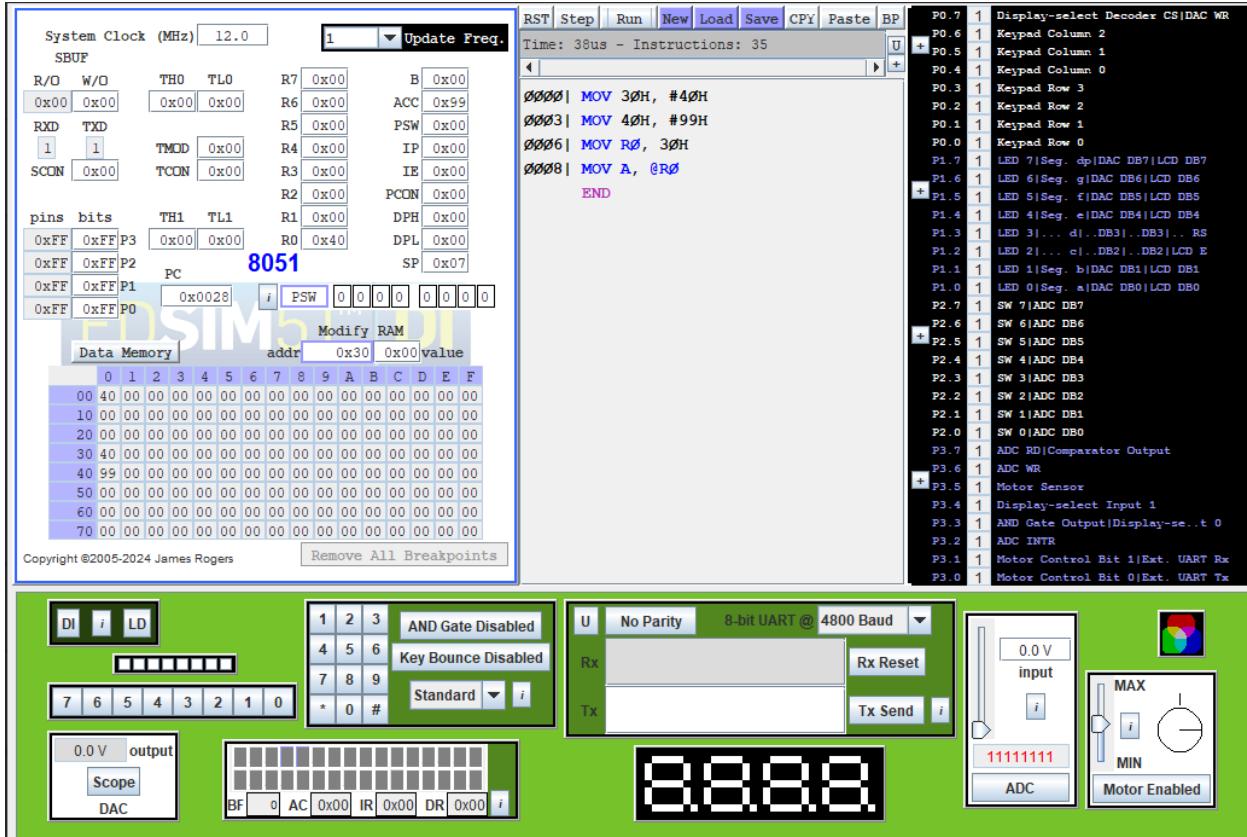
### Direct Addressing mode



**NAME:** Rutva Sawarkar

**PRN:** 24070521189

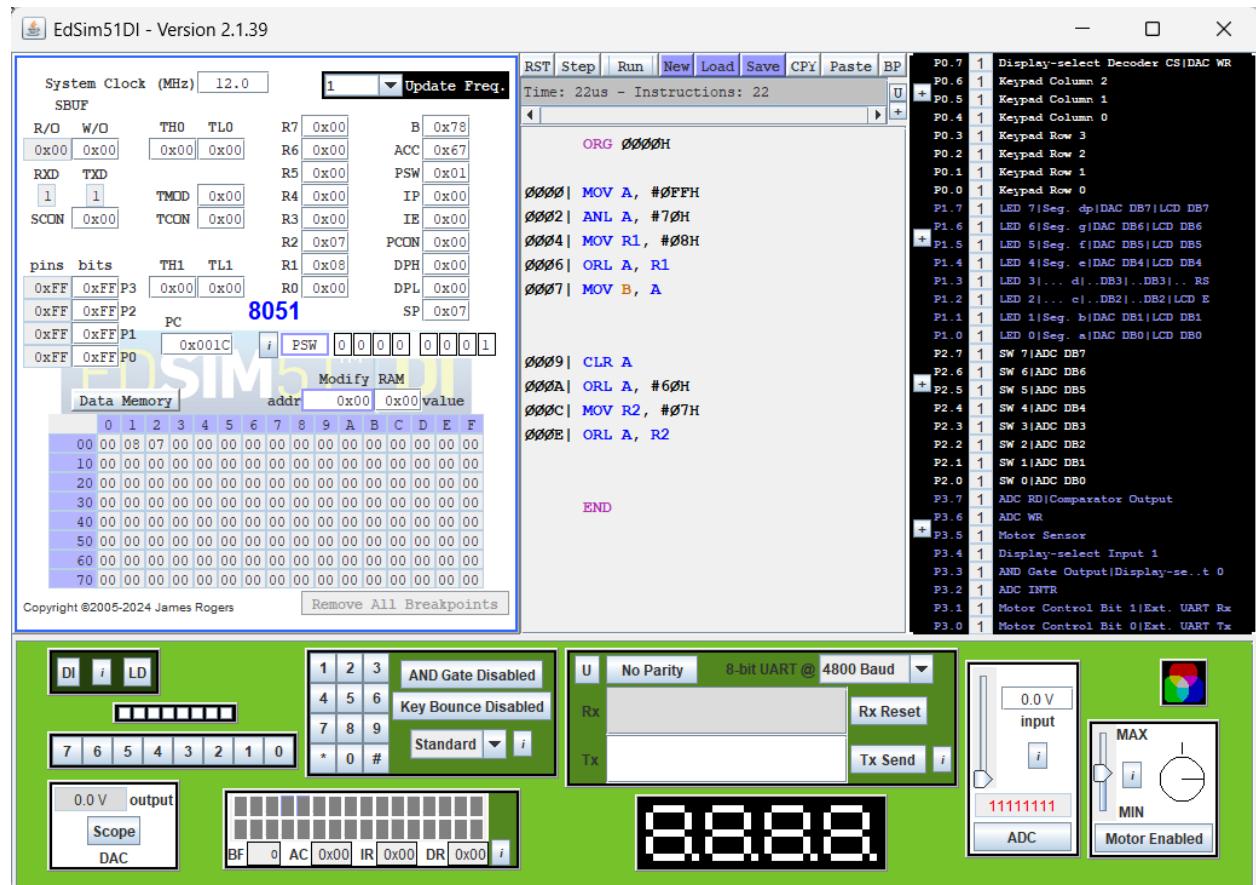
### Indirect addressing mode



**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.

