

Name : Priya Prasad

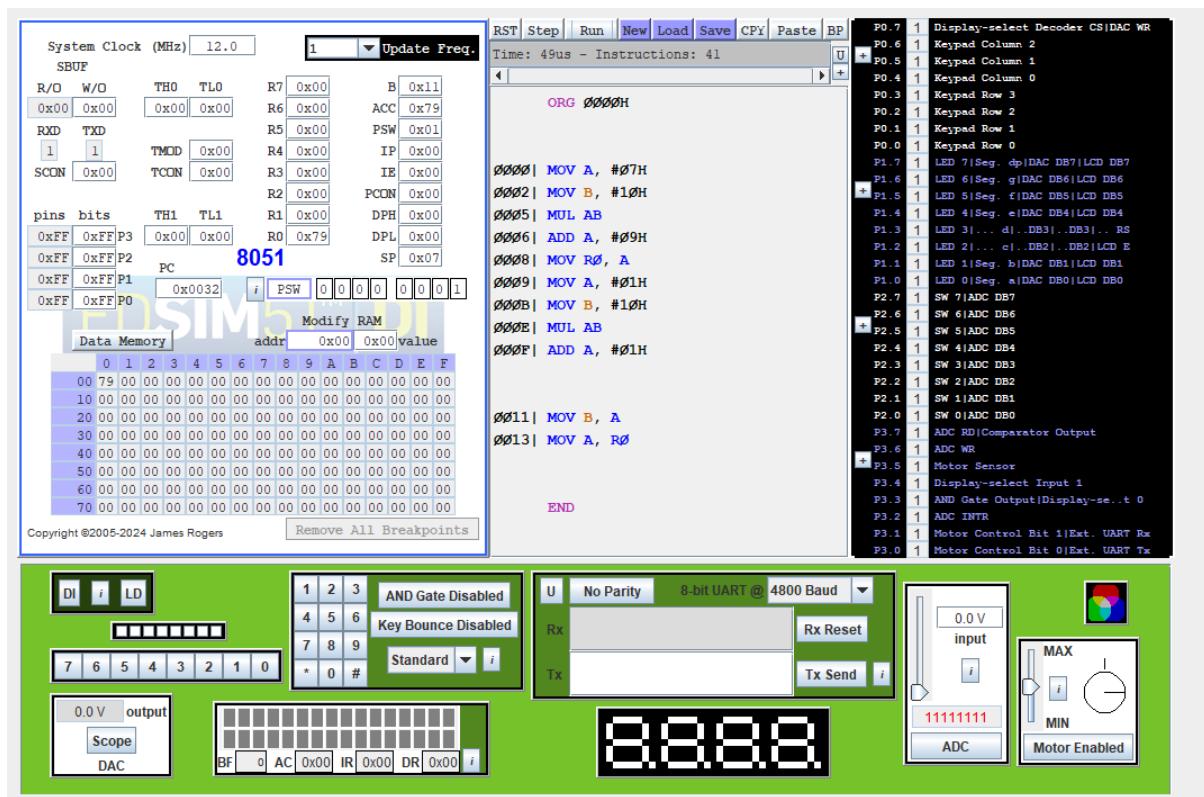
PRN : 24070521179

Sec : C

Sem : 4th

Sub : MES CA1

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 : Priya Prasad

PRN : 24070521179

Sec : C

Q2. 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.

CASE I : When A>B. Result is 01H

The screenshot shows the Proteus SIM software interface for an 8051 microcontroller. The top section displays the CPU status with various registers and pins. The assembly code window shows the following sequence:

```
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 c
DEC R1 ; Decrement B c
JZ A_ZERO ; Did R1 just b
MOV A, R0 ; Check R0
JZ B_GREATER ; If R0 is zero
SJMP COMPARE ; Neither is ze

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

A_GREATER:
MOV 52H, #01H
SJMP DONE

B_GREATER:

```

The data memory window shows the initial values at addresses 50H and 51H. The assembly code uses the Zero flag (ZF) to determine if R0 (A) is greater than R1 (B). If R0 is zero, it jumps to the A_GREATER label. If R1 is zero, it jumps to the B_GREATER label. If neither is zero, it compares them. The A_GREATER label moves the value 01H to address 52H and then jumps to the DONE label. The B_GREATER label also jumps to the DONE label.

Name : Priya Prasad

PRN : 24070521179

Sec : C

CASE II : When A = B. Result is 00H

The screenshot shows the EDISIM5 software interface for an 8051 microcontroller. The top bar includes buttons for RST, Step, Run, New, Load, Save, CPY, Paste, and BP. A status bar at the top right indicates "Time: 75us - Instructions: 63".

Registers:

System Clock (MHz)	12.0						
R/O	W/O	TH0	TL0	R7	0x00	B	0x00
0x00	0x00	0x00	0x00	R6	0x00	ACC	0x00
RXD	TxD	TMOD	0x00	R5	0x00	PSW	0x00
1	1	TCON	0x00	R4	0x00	IP	0x00
SCON	0x00	Pins	bits	R3	0x00	IE	0x00
		TH1	TL1	R2	0x00	PCON	0x00
0xFF	0xFF	P3	0x00	0x00	R1	DPH	0x00
0xFF	0xFF	P2	PC	0x00	R0	DPL	0x00
0xFF	0xFF	P1		0x0043	i	PSW	0 0 0 0 0 0 0 0 0
0xFF	0xFF	P0				SP	0x07

Memory Dump:

Modify RAM															
Data Memory		addr		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

Assembly Code:

```

; 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 III : When A < B. Result is FFH

The screenshot shows the Z80 SIM51 software interface. The top menu bar includes RST, Assm, Run, New, Load, Save, CPY, Paste, and BP. The System Clock (MHz) is set to 12.0. A dropdown menu is open for 'Update Freq.' with the value 1 selected. The Registers window shows various寄存器 (R0-R7, ACC, PSW, IP, IE, PCON) with their current values. The Pins window shows port pins P0-P3 with their current states. The assembly code window contains instructions for loading values into registers A and B, performing comparisons, and jumping if register R0 or R1 is zero. The Modify RAM window allows changing memory values at address 0x50. The Data Memory dump window shows a 16x16 grid of memory contents from 00 to FF.

Name : Priya Prasad

PRN : 24070521179

Sec : C

Q3. 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.

CASE 1 Direct

The screenshot shows a simulation environment for a PIC16F877A microcontroller. The assembly code window displays the following instructions:

```
0000| MOV 30H, #90H
0003| MOV A, 30H
END
```

The registers window shows the following initial values:

Register	Value
R7	0x00
R6	0x00
R5	0x00
R4	0x00
R3	0x00
R2	0x00
R1	0x00
R0	0x00
PSW	0x00
DPL	0x00
DPH	0x00
SP	0x07
PC	0x0011
TH1	0x00
TL1	0x00
SCON	0x00
TCON	0x00
RXDATA	0x00
TXDATA	0x00
WDTCON	0x00
INTCON	0x00
PIR1	0x00
PIR0	0x00
PIR2	0x00
PIR3	0x00
PIR4	0x00
PIR5	0x00
PIR6	0x00
PIR7	0x00
PIR8	0x00
PIR9	0x00
PIR10	0x00
PIR11	0x00
PIR12	0x00
PIR13	0x00
PIR14	0x00
PIR15	0x00
PIR16	0x00
PIR17	0x00
PIR18	0x00
PIR19	0x00
PIR20	0x00
PIR21	0x00
PIR22	0x00
PIR23	0x00
PIR24	0x00
PIR25	0x00
PIR26	0x00
PIR27	0x00
PIR28	0x00
PIR29	0x00
PIR30	0x00
PIR31	0x00
PIR32	0x00
PIR33	0x00
PIR34	0x00
PIR35	0x00
PIR36	0x00
PIR37	0x00
PIR38	0x00
PIR39	0x00
PIR40	0x00
PIR41	0x00
PIR42	0x00
PIR43	0x00
PIR44	0x00
PIR45	0x00
PIR46	0x00
PIR47	0x00
PIR48	0x00
PIR49	0x00
PIR50	0x00
PIR51	0x00
PIR52	0x00
PIR53	0x00
PIR54	0x00
PIR55	0x00
PIR56	0x00
PIR57	0x00
PIR58	0x00
PIR59	0x00
PIR60	0x00
PIR61	0x00
PIR62	0x00
PIR63	0x00
PIR64	0x00
PIR65	0x00
PIR66	0x00
PIR67	0x00
PIR68	0x00
PIR69	0x00
PIR70	0x00
PIR71	0x00
PIR72	0x00
PIR73	0x00
PIR74	0x00
PIR75	0x00
PIR76	0x00
PIR77	0x00
PIR78	0x00
PIR79	0x00
PIR80	0x00
PIR81	0x00
PIR82	0x00
PIR83	0x00
PIR84	0x00
PIR85	0x00
PIR86	0x00
PIR87	0x00
PIR88	0x00
PIR89	0x00
PIR90	0x00
PIR91	0x00
PIR92	0x00
PIR93	0x00
PIR94	0x00
PIR95	0x00
PIR96	0x00
PIR97	0x00
PIR98	0x00
PIR99	0x00
PIR100	0x00
PIR101	0x00
PIR102	0x00
PIR103	0x00
PIR104	0x00
PIR105	0x00
PIR106	0x00
PIR107	0x00
PIR108	0x00
PIR109	0x00
PIR110	0x00
PIR111	0x00
PIR112	0x00
PIR113	0x00
PIR114	0x00
PIR115	0x00
PIR116	0x00
PIR117	0x00
PIR118	0x00
PIR119	0x00
PIR120	0x00
PIR121	0x00
PIR122	0x00
PIR123	0x00
PIR124	0x00
PIR125	0x00
PIR126	0x00
PIR127	0x00
PIR128	0x00
PIR129	0x00
PIR130	0x00
PIR131	0x00
PIR132	0x00
PIR133	0x00
PIR134	0x00
PIR135	0x00
PIR136	0x00
PIR137	0x00
PIR138	0x00
PIR139	0x00
PIR140	0x00
PIR141	0x00
PIR142	0x00
PIR143	0x00
PIR144	0x00
PIR145	0x00
PIR146	0x00
PIR147	0x00
PIR148	0x00
PIR149	0x00
PIR150	0x00
PIR151	0x00
PIR152	0x00
PIR153	0x00
PIR154	0x00
PIR155	0x00
PIR156	0x00
PIR157	0x00
PIR158	0x00
PIR159	0x00
PIR160	0x00
PIR161	0x00
PIR162	0x00
PIR163	0x00
PIR164	0x00
PIR165	0x00
PIR166	0x00
PIR167	0x00
PIR168	0x00
PIR169	0x00
PIR170	0x00
PIR171	0x00
PIR172	0x00
PIR173	0x00
PIR174	0x00
PIR175	0x00
PIR176	0x00
PIR177	0x00
PIR178	0x00
PIR179	0x00
PIR180	0x00
PIR181	0x00
PIR182	0x00
PIR183	0x00
PIR184	0x00
PIR185	0x00
PIR186	0x00
PIR187	0x00
PIR188	0x00
PIR189	0x00
PIR190	0x00
PIR191	0x00
PIR192	0x00
PIR193	0x00
PIR194	0x00
PIR195	0x00
PIR196	0x00
PIR197	0x00
PIR198	0x00
PIR199	0x00
PIR200	0x00
PIR201	0x00
PIR202	0x00
PIR203	0x00
PIR204	0x00
PIR205	0x00
PIR206	0x00
PIR207	0x00
PIR208	0x00
PIR209	0x00
PIR210	0x00
PIR211	0x00
PIR212	0x00
PIR213	0x00
PIR214	0x00
PIR215	0x00
PIR216	0x00
PIR217	0x00
PIR218	0x00
PIR219	0x00
PIR220	0x00
PIR221	0x00
PIR222	0x00
PIR223	0x00
PIR224	0x00
PIR225	0x00
PIR226	0x00
PIR227	0x00
PIR228	0x00
PIR229	0x00
PIR230	0x00
PIR231	0x00
PIR232	0x00
PIR233	0x00
PIR234	0x00
PIR235	0x00
PIR236	0x00
PIR237	0x00
PIR238	0x00
PIR239	0x00
PIR240	0x00
PIR241	0x00
PIR242	0x00
PIR243	0x00
PIR244	0x00
PIR245	0x00
PIR246	0x00
PIR247	0x00
PIR248	0x00
PIR249	0x00
PIR250	0x00
PIR251	0x00
PIR252	0x00
PIR253	0x00
PIR254	0x00
PIR255	0x00
PIR256	0x00
PIR257	0x00
PIR258	0x00
PIR259	0x00
PIR260	0x00
PIR261	0x00
PIR262	0x00
PIR263	0x00
PIR264	0x00
PIR265	0x00
PIR266	0x00
PIR267	0x00
PIR268	0x00
PIR269	0x00
PIR270	0x00
PIR271	0x00
PIR272	0x00
PIR273	0x00
PIR274	0x00
PIR275	0x00
PIR276	0x00
PIR277	0x00
PIR278	0x00
PIR279	0x00
PIR280	0x00
PIR281	0x00
PIR282	0x00
PIR283	0x00
PIR284	0x00
PIR285	0x00
PIR286	0x00
PIR287	0x00
PIR288	0x00
PIR289	0x00
PIR290	0x00
PIR291	0x00
PIR292	0x00
PIR293	0x00
PIR294	0x00
PIR295	0x00
PIR296	0x00
PIR297	0x00
PIR298	0x00
PIR299	0x00
PIR300	0x00
PIR301	0x00
PIR302	0x00
PIR303	0x00
PIR304	0x00
PIR305	0x00
PIR306	0x00
PIR307	0x00
PIR308	0x00
PIR309	0x00
PIR310	0x00
PIR311	0x00
PIR312	0x00
PIR313	0x00
PIR314	0x00
PIR315	0x00
PIR316	0x00
PIR317	0x00
PIR318	0x00
PIR319	0x00
PIR320	0x00
PIR321	0x00
PIR322	0x00
PIR323	0x00
PIR324	0x00
PIR325	0x00
PIR326	0x00
PIR327	0x00
PIR328	0x00
PIR329	0x00
PIR330	0x00
PIR331	0x00
PIR332	0x00
PIR333	0x00
PIR334	0x00
PIR335	0x00
PIR336	0x00
PIR337	0x00
PIR338	0x00
PIR339	0x00
PIR340	0x00
PIR341	0x00
PIR342	0x00
PIR343	0x00
PIR344	0x00
PIR345	0x00
PIR346	0x00
PIR347	0x00
PIR348	0x00
PIR349	0x00
PIR350	0x00
PIR351	0x00
PIR352	0x00
PIR353	0x00
PIR354	0x00
PIR355	0x00
PIR356	0x00
PIR357	0x00
PIR358	0x00
PIR359	0x00
PIR360	0x00
PIR361	0x00
PIR362	0x00
PIR363	0x00
PIR364	0x00
PIR365	0x00
PIR366	0x00
PIR367	0x00
PIR368	0x00
PIR369	0x00
PIR370	0x00
PIR371	0x00
PIR372	0x00
PIR373	0x00
PIR374	0x00
PIR375	0x00
PIR376	0x00
PIR377	0x00
PIR378	0x00
PIR379	0x00
PIR380	0x00
PIR381	0x00
PIR382	0x00
PIR383	0x00
PIR384	0x00
PIR385	0x00
PIR386	0x00
PIR387	0x00
PIR388	0x00
PIR389	0x00
PIR390	0x00
PIR391	0x00
PIR392	0x00
PIR393	0x00
PIR394	0x00
PIR395	0x00
PIR396	0x00
PIR397	0x00
PIR398	0x00
PIR399	0x00
PIR400	0x00
PIR401	0x00
PIR402	0x00
PIR403	0x00
PIR404	0x00
PIR405	0x00
PIR406	0x00
PIR407	0x00
PIR408	0x00
PIR409	0x00
PIR410	0x00
PIR411	0x00
PIR412	0x00
PIR413	0x00
PIR414	0x00
PIR415	0x00
PIR416	0x00
PIR417	0x00
PIR418	0x00
PIR419	0x00
PIR420	0x00
PIR421	0x00
PIR422	0x00
PIR423	0x00
PIR424	0x00
PIR425	0x00
PIR426	0x00
PIR427	0x00
PIR428	0x00
PIR429	0x00
PIR430	0x00
PIR431	0x00
PIR432	0x00
PIR433	0x00
PIR434	0x00
PIR435	0x00
PIR436	0x00
PIR437	0x00
PIR438	0x00
PIR439	0x00
PIR440	0x00
PIR441	0x00
PIR442	0x00
PIR443	0x00
PIR444	0x00
PIR445	0x00
PIR446	0x00
PIR447	0x00
PIR448	0x00
PIR449	0x00
PIR450	0x00
PIR451	

Name : Priya Prasad

PRN : 24070521179

Sec : C

CASE 2 Indirect

The screenshot displays a development environment for the PIC16F877A microcontroller. At the top, there is a status bar with "System Clock (MHz) 12.0", "SBUF", and a frequency selection dropdown set to "1". Below the status bar is a menu bar with "RST", "Step", "Run", "New", "Load", "Save", "CPY", "Paste", and "BP". The main window shows assembly code and memory dump.

Assembly Code:

```
0000 | MOV 30H, #40H
0003 | MOV 40H, #99H
0006 | MOV R0, 30H
0008 | MOV A, @R0
END
```

Memory Dump:

R/O	W/O	TH0	TL0	R7	0x00	B	0x00									
0x00	0x00	0x00	0x00	R6	0x00	ACC	0x99									
RXD	TxD	1	1	TMOD	0x00	PSW	0x00									
SCON	0x00	TCON	0x00	R4	0x00	IP	0x00									
pins	bits	TH1	TL1	R3	0x00	IE	0x00									
0xFF	0xFFP3	0x00	0x00	R2	0x00	PCON	0x00									
0xFF	0xFFP2	PC	8051	R1	0x00	DPH	0x00									
0xFF	0xFFP1	0x0028		R0	0x40	DPL	0x00									
0xFF	0xFFP0			PSW	0 0 0 0 0 0 0 0											
Modify RAM																
Data Memory																
addr	0x30	0x00	value													
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	40	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	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	99	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	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	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Copyright ©2005-2024 James Rogers

Remove All Breakpoints

Graphical User Interface (Bottom):

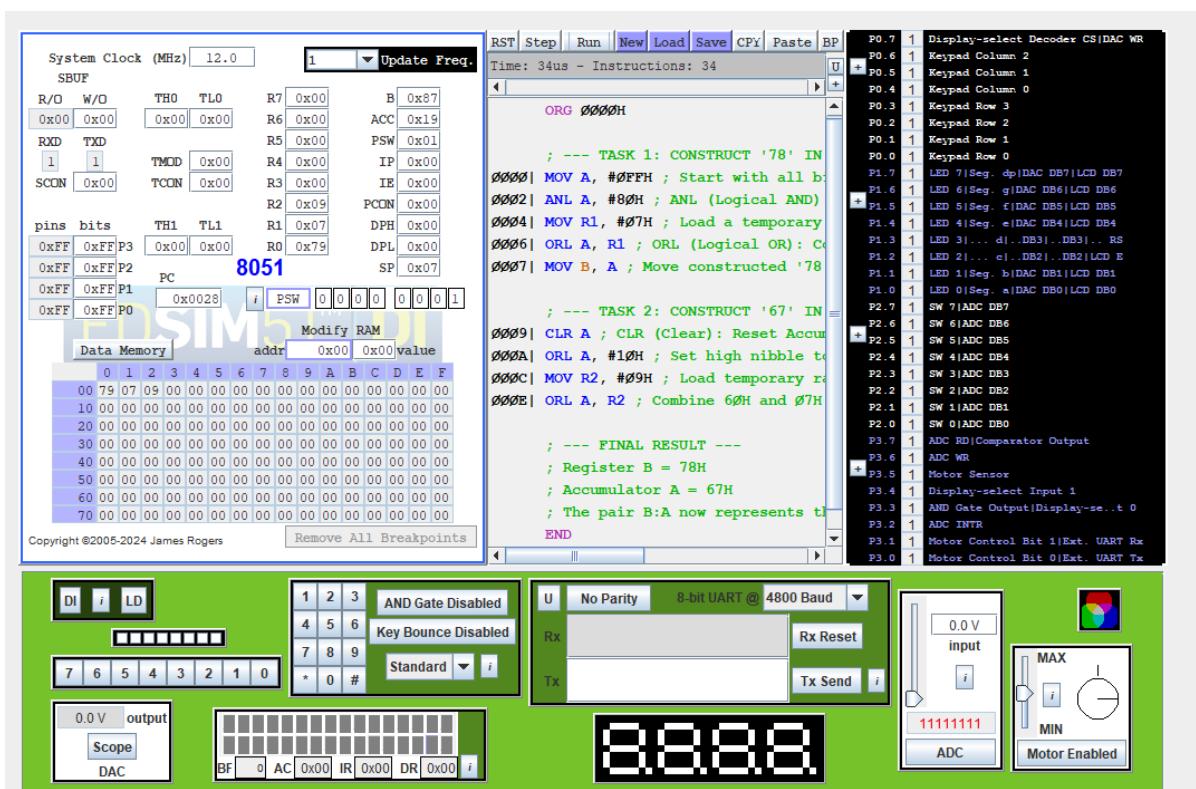
- DI, i, LD buttons and a digital input LED bar.
- Key pad with digits 1-9, *, 0, #, AND Gate Disabled, Key Bounce Disabled, Standard, and a digital output LED bar.
- 0.0 V output, Scope, DAC buttons.
- UART configuration: U, No Parity, 8-bit UART @ 4800 Baud, Rx, Tx, Rx Reset, Tx Send.
- 8-digit digital display showing "8.8.8.8".
- ADC input, MAX, MIN, Motor Enabled buttons.

Name : Priya Prasad

PRN : 24070521179

Sec : C

Q4. 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 : Priya Prasad

PRN : 24070521179

Sec : C

Q5. 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.

