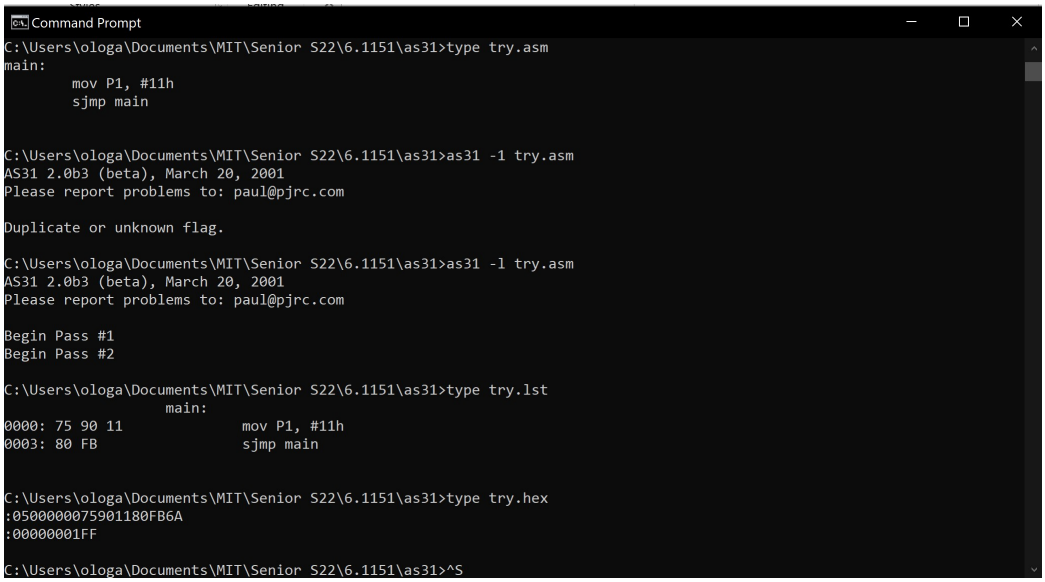
6.115 Laboratory 0 (Pre-lab)

David Ologan

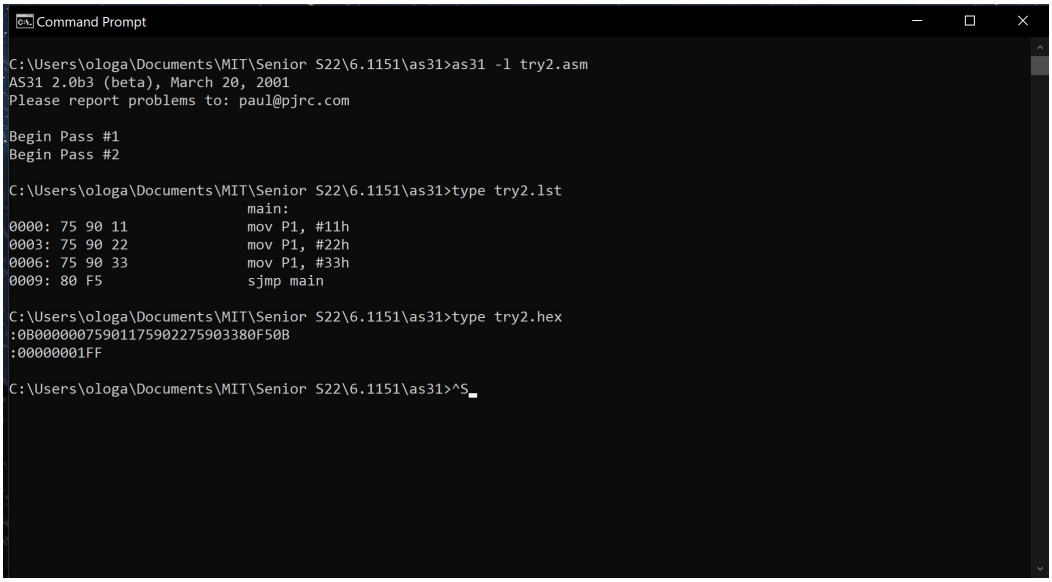
June 9, 2022

1 Exercise 2: Assembly Language Party with AS31

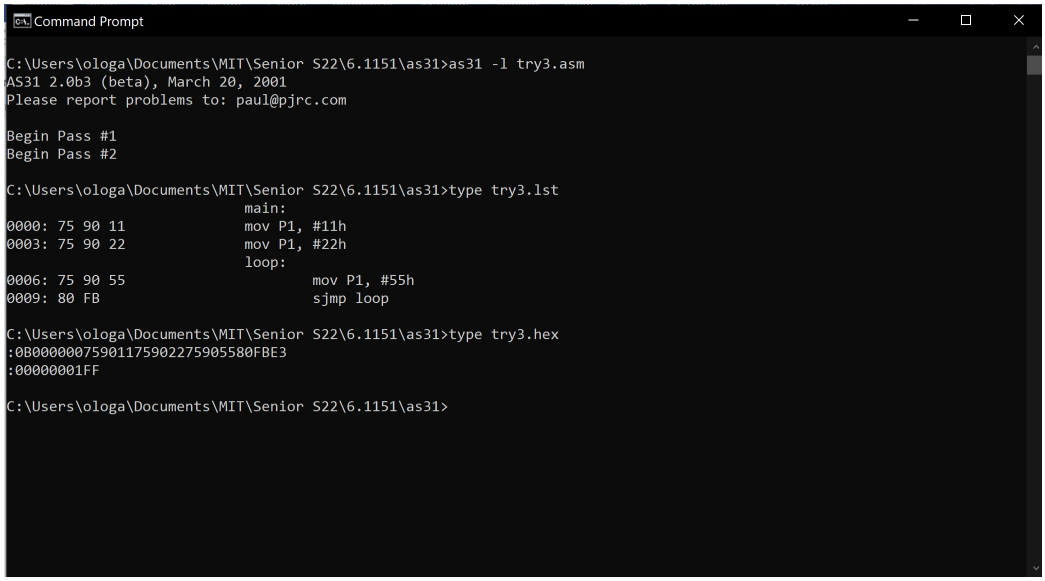
- Use AS31 to assemble the try.asm file and produce a LST file and a HEX file. Examine the LST file and the HEX file, and confirm the observations made in the discussion above.



- Make your own try2.asm file with a total of five lines. This new ASM file should move the byte 11h to P1 as before. Then move 22h to P1. then move 33h to P1. Then loop back and do it endlessly. Assemble your try2.asm file to produce LST and HEX files. Be prepared to explain your LST and HEX files.



- Make your own try3.asm file. This file should move the byte to 11h to P1. Then move 22h to P1. Then the program should enter a loop that endlessly writes 55h to P1. Assemble your try3.asm file to produce LST and HEX files. Be prepared to explain your LST and HEX files.



- Why is it important that all our files end in a controlled way, with a loop to somewhere? Suppose your entire program was just one line: mov P1, 11h. If you try to assembling this one line with the AS31, it should assemble with no trouble. Why is this program a bad idea? Why would the following three line program, which achieves the same goal for P1 be a better idea?

It is important that all our files end in a controlled way, with a loop to somewhere because this prevents the 8051 from reading whatever else was stored in the chip's ROM. If you try assembling this line, and the file doesn't end in a loop somewhere, it will move 11h to P1, but will subsequently increment and read whatever else you have stored in the ROM. This program is a bad idea since it probably won't accomplish what you intended for your program. The following jumps back to the loop subroutine infinitely, completing our move command once without any unintended action, effectively trapping it.

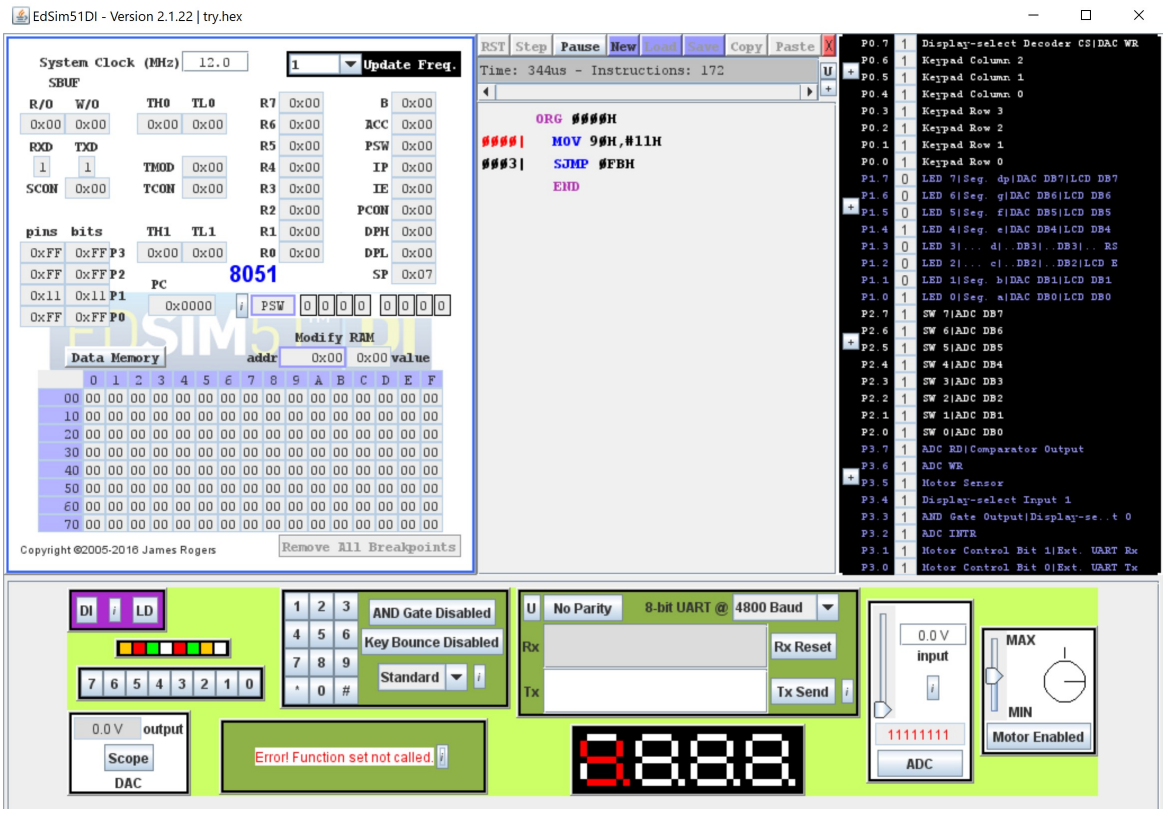
- Based on the information you have in the pictures above and the Intel manual, is the T800 Terminator using an 8051 family micro-controller?

Based on the information given in the pictures above and the Intel manual, the T800 Terminator is not using an 8051 family micro-controller. Given the figure on page 2, many of the opcodes used by the T800 do not exist for the 8051, leading me to believe it operates using a different micro-controller.

2 Exercise 3: Run your code!

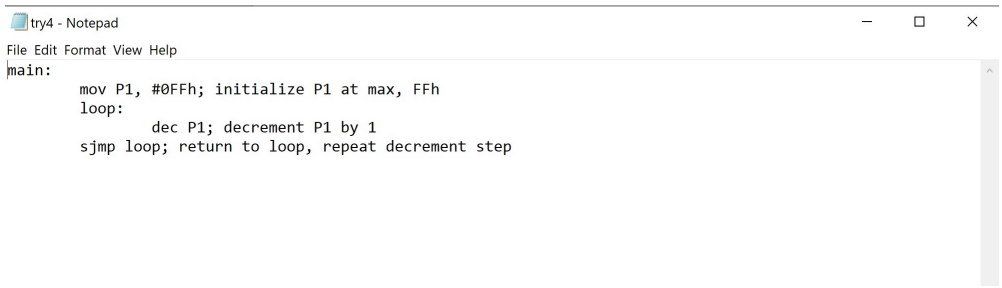
- Use AS31 to assemble the try.asm file and produce a LST file and a HEX file. Load the try.hex file into Edsim. Run it using "step," and also using "run". Explain what you see in Edsim areas I, III, and IV, including comments on the state of P1, the "individual LED's" and the character LED at the bottom of area IV.

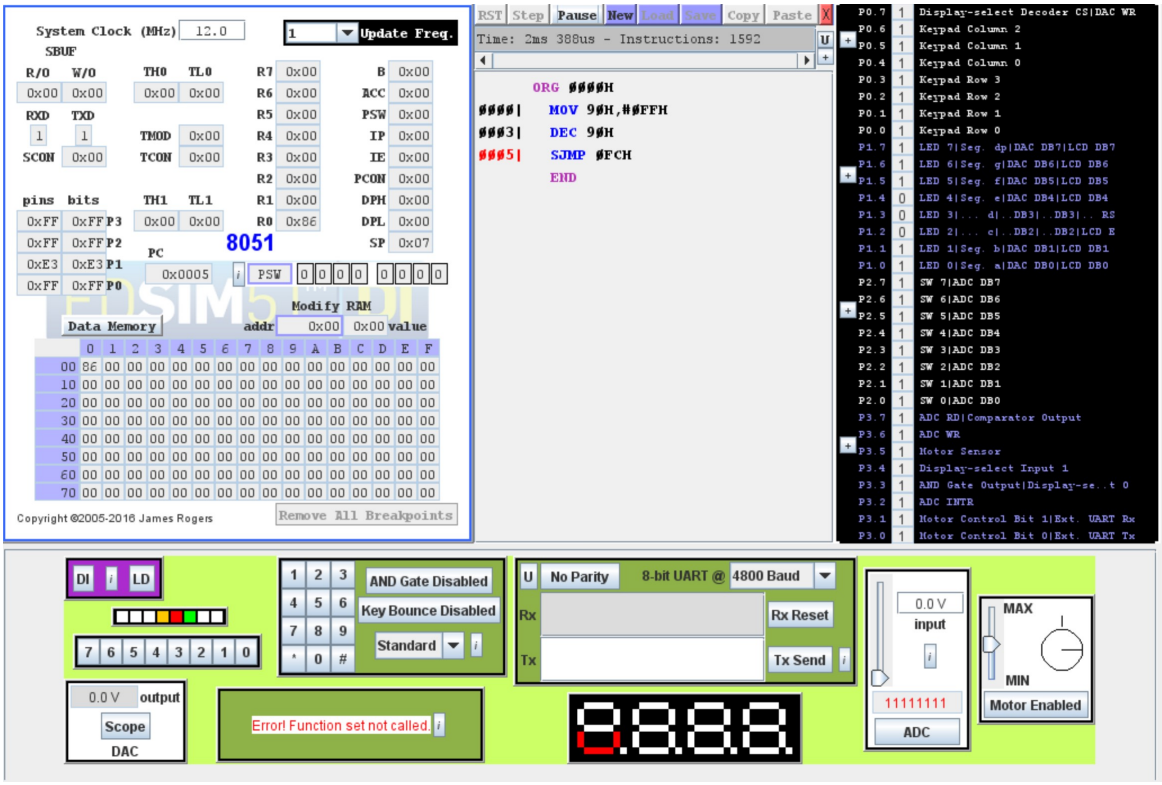
After running the try.asm file in Edsim, in area I, the program counter PC begins to increase, and register P1 reads the bits we input, namely 11h. In area II we see our inputted code, with the highlighted line at which the code is currently reading. In area III we see the state of the metal pins, and at P1, we can see the bit we read earlier. Finally in area IV, we see this surface in our LED's, and our 7-Seg display. 11h in binary is 00001011, so the appropriate LED's and portions of the display are lit. Running the program in step, allows the program to progress line by line through the code. Visually, running the program in step and run are indistinguishable since, it changes the state of the register once, and is stuck in an infinite loop.



- Notice that P1 could have any of 256 different values, ranging between 00h and FFh (same as 0d through 255d). Write a program in a file called try.asm that cycles P1 through all possible values sequentially In your program be sure to use 8051 opcodes: MOV, DEC, SJMP. Assemble the program using AS31 and then run your program HEX file in Edsim. What update frequency makes sense for watching Edsim re-execute this program? explain what you see in each Edsim area I, II, III, IV. Using your try4.lst file, explain what you see in Edsim area I "code memory" section after you have uploaded "try4.hex". Does your program count P1 up or down? What do the individual LED lights do in area IV? Why?

After running the try4.asm file in Edsim, in area I, again the program counter increases, and the register P1 reads the current value of the bit, starting from FFh and decreasing. In area II, we see the code we uploaded to Edsim, as well as whatever line we're currently if were in step mode. In area 3, the physical pin output states are shown, and those in P1 mirror the desired value from the register. Finally, in area 4, we see the 7-Seg and LED's progress as the value of P1 sequentially decreases. My program counts P1 down, since we used DEC, it decrements by 1 each cycle through. An update frequency of 1 sec makes sense, since we want to set it to a slow enough speed such that we can see the visible progression of the register and the LED's





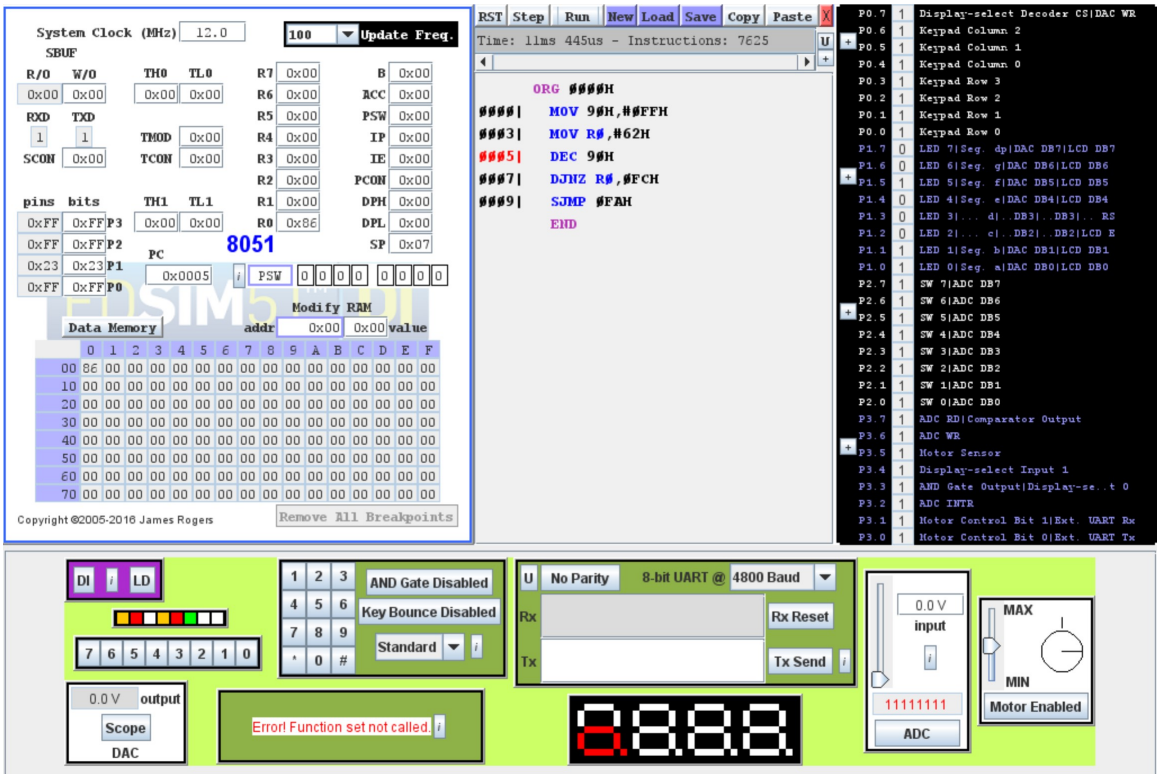
- Set Edsim for an update frequency of 100. Run your try4.asm program file from the previous exercise again. What do you see on the LED's? Why? Fix it. Leave the update frequency at 100. Modify your code to use DJNZ opcode and register R0 to create a reasonable delay that lets you see all of the counting action on the LED's. Note that this is a "real" problem. The 8051 executes opcodes very quickly with a clock frequency of 11.0592 MHz.

After setting Edsim to an update frequency of 100, the program only updates every 100 steps, so we fail to see any recognizable patterns, i.e the decrements of P1. In order to see the counting action of the LED's we would need to add a delay, so we can visibly see the 8051 working.

try5 - Notepad

File Edit Format View Help

```
main:
        mov P1, #0FFh; initialize P1 at max, FFh
        mov R0, #62h; use register to store a reasonable delay, 62 since the delay step takes another turn
        loop:
                dec P1; decrement P1 by 1
                DJNZ R0, loop; implement the delay
        sjmp loop; return to loop, repeat decrement step
```

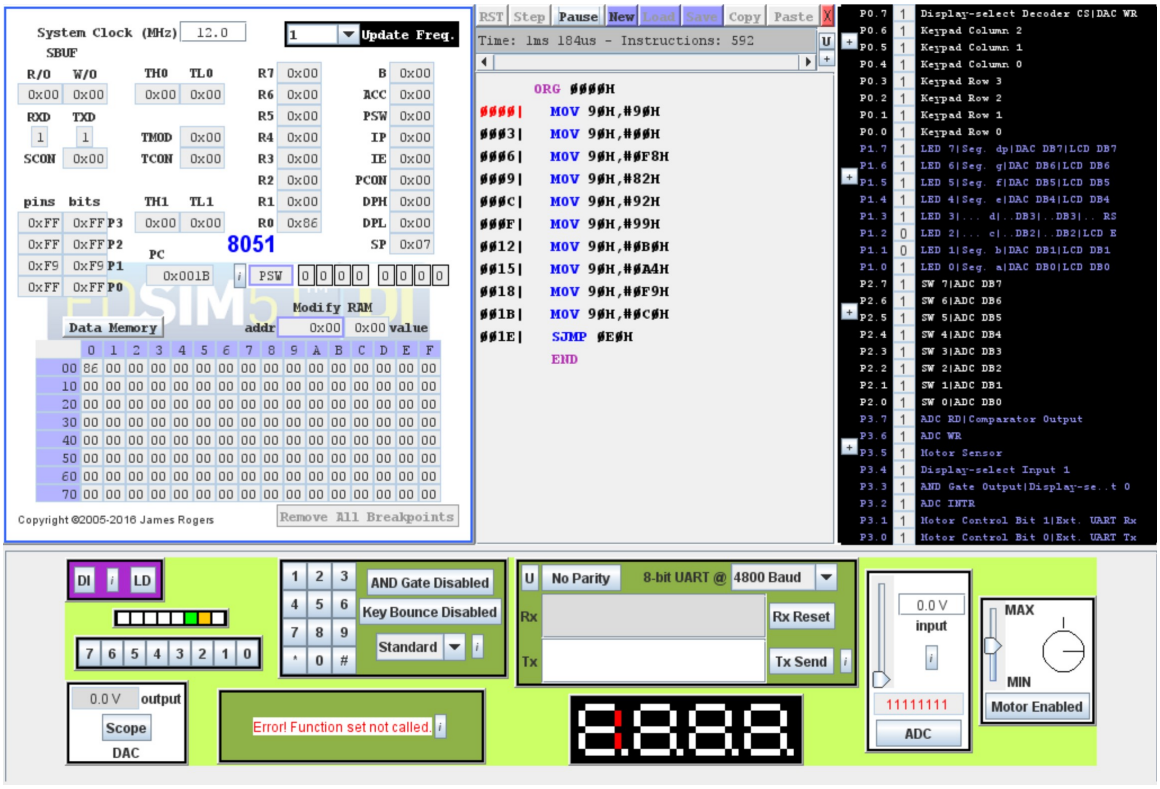


- Now lets write a program that displays something interesting on the character LED's, the "digit displays" that look like a number 8 at the bottom of area Iv in Edsim. There are four digit displays. By default Edsim will drive the leftmost digit display, which is labeled DISP 3 in the Edsim peripheral hardware schematics. Figure out the ten hex numbers or code you would have to write to display each of the decimal numbers 0,1,2,3,4,5,6,7,8,9 on the digit display. Then write a program that counts down from 9 to 8 to 7 sequentially down to 0 and then repeats this endlessly. Use AS31 and Edsim to write and test your program.

countdown - Notepad

File Edit Format View Help

```
main:
        mov P1, #90h; Set the Hex code for 9
        mov P1, #00h; Set the Hex code for 8
        mov P1, #0F8h; Set the Hex code for 7
        mov P1, #82h; Set the Hex code for 6
        mov P1, #92h; Set the Hex code for 5
        mov P1, #99h; Set the Hex code for 4
        mov P1, #0B0h; Set the Hex code for 3
        mov P1, #0A4h; Set the Hex code for 2
        mov P1, #0F9h; Set the Hex code for 1
        mov P1, #0C0h; Set the Hex code for 0
        sjmp main; Go back to beginning
```

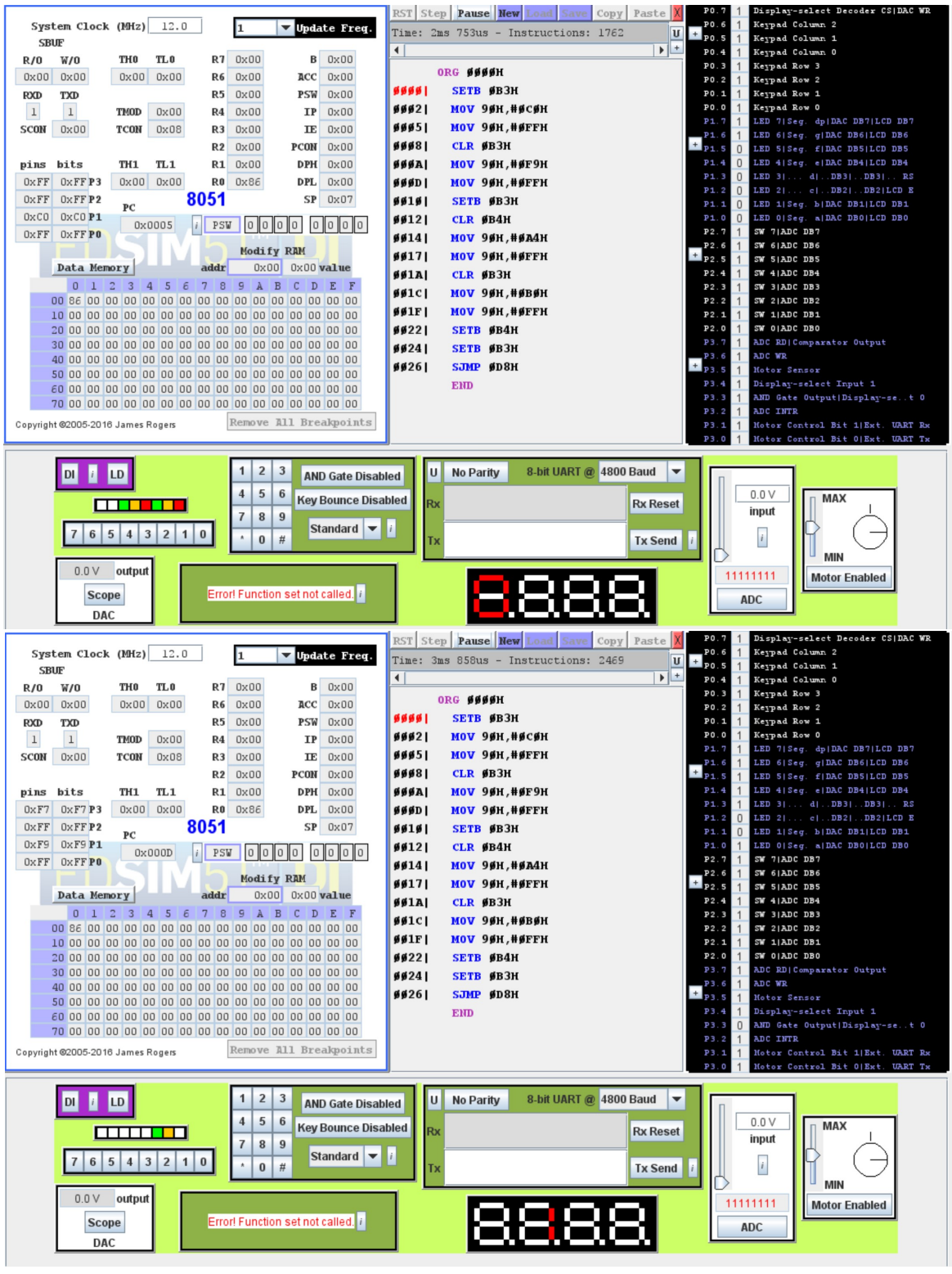


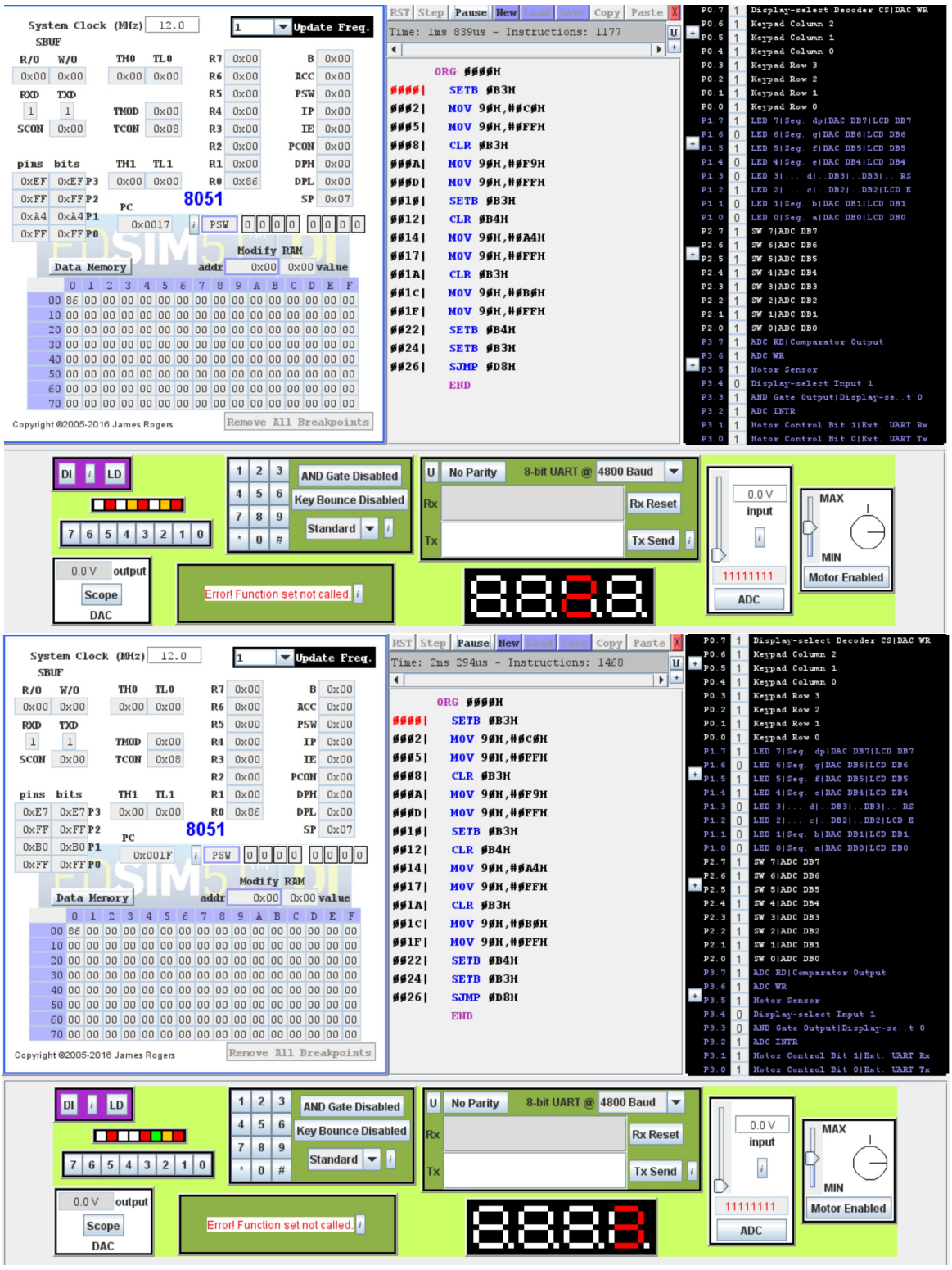
- Let's try another program on the digit displays. This time, let's use all four digit displays, DISP 3 though DISP 0, on e at a time. First, display the number "0" on DISP 3. then "1" on DISP 2 then "2" on DISP 1, and finally "3" on DISP 0. Then repeat endlessly. Some points to consider: notice that you select which DISP display you want to use by properly setting port pins 3.3 and 3.4 (thus controlling the decoder). Sue opcodes SETB and CLR to set the pin states for pin 3.3 and 3.4. Also note that you should display a number on a particular DISP, but then clear the number from the display before moving to the next DISP. If you don't do this clearing step, your number will show up on two DISP's sequentially. Here's are snapshots of the digit display as the staff solution runs in Edsim; your program should create the same digit count progressively, looping endlessly.

Sequential - Notepad

File Edit Format View Help

```
main:
        SETB P3.3; Bring Pin 3.3 High, Select First Display
        mov P1, #0C0h; Display 0
        mov P1, #0FFh; Clear Display
        CLR P3.3; Bring Pin 3.3 Low, Select Second Display
        mov P1, #0F9h; Display 1
        mov P1, #0FFh; Clear Display
        SETB P3.3; Reset State of 3.3

        CLR P3.4; Bring Pin 3.4 Low,3.3 High, Select Third Display
        mov P1, #0A4h; Display 2
        mov P1, #0FFh; Clear Display
        CLR P3.3; bring Pin 3.3 Low, Choose Display 4
        mov P1, #0B0h; Display 3
        mov P1, #0FFh; Clear Display
        SETB P3.4;
        SETB P3.3; Reset initial states of Pins
        sjmp main
```


System Clock (MHz) 12.0
Update Freq.
SBUF
R/O W/O TH0 TL0 R7 B
RXD TXD R6 ACC
R5 PSW
TMOD R4 IP
SCON TCON R3 IE
R2 PCON
pins bits TH1 TL1 R1 DPH
R0 DPL
8051
PC
SP
PSW
Modify RAM
Data Memory
addr value
Copyright ©2005-2016 James Rogers
Remove All Breakpoints
RST Step Pause New Load Save Copy Paste
Time: 2ms 753us - Instructions: 1762
ORG $$$$H
$$$$| SETB $B3H
$$$2| MOV 9$H,#$C$H
$$$5| MOV 9$H,#$FFH
$$$8| CLR $B3H
$$$A| MOV 9$H,#$F9H
$$$D| MOV 9$H,#$FFH
$$1$| SETB $B3H
$$12| CLR $B4H
$$14| MOV 9$H,#$A4H
$$17| MOV 9$H,#$FFH
$$1A| CLR $B3H
$$1C| MOV 9$H,#$B$H
$$1F| MOV 9$H,#$FFH
$$22| SETB $B4H
$$24| SETB $B3H
$$26| SJMP $D8H
END
P0.7 Display-select Decoder CS|DAC WR
P0.6 Keypad Column 2
P0.5 Keypad Column 1
P0.4 Keypad Column 0
P0.3 Keypad Row 3
P0.2 Keypad Row 2
P0.1 Keypad Row 1
P0.0 Keypad Row 0
P1.7 LED 7|Seg. dp|DAC DB7|LCD DB7
P1.6 LED 6|Seg. g|DAC DB6|LCD DB6
P1.5 LED 5|Seg. f|DAC DB5|LCD DB5
P1.4 LED 4|Seg. e|DAC DB4|LCD DB4
P1.3 LED 3|... d|..DB3|..DB3|.. RS
P1.2 LED 2|... c|..DB2|..DB2|LCD E
P1.1 LED 1|Seg. b|DAC DB1|LCD DB1
P1.0 LED 0|Seg. a|DAC DB0|LCD DB0
P2.7 SW 7|ADC DB7
P2.6 SW 6|ADC DB6
P2.5 SW 5|ADC DB5
P2.4 SW 4|ADC DB4
P2.3 SW 3|ADC DB3
P2.2 SW 2|ADC DB2
P2.1 SW 1|ADC DB1
P2.0 SW 0|ADC DB0
P3.7 ADC RD|Comparator Output
P3.6 ADC WR
P3.5 Motor Sensor
P3.4 Display-select Input 1
P3.3 AND Gate Output|Display-se..t 0
P3.2 ADC INTR
P3.1 Motor Control Bit 1|Ext. UART Rx
P3.0 Motor Control Bit 0|Ext. UART Tx
DI LD
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity 8-bit UART @ 4800 Baud
Rx Reset
Tx Send
0.0 V input
MAX
MIN
Motor Enabled
11111111
ADC
0.0 V output
Scope
DAC
Error! Function set not called.
Time: 3ms 858us - Instructions: 2469

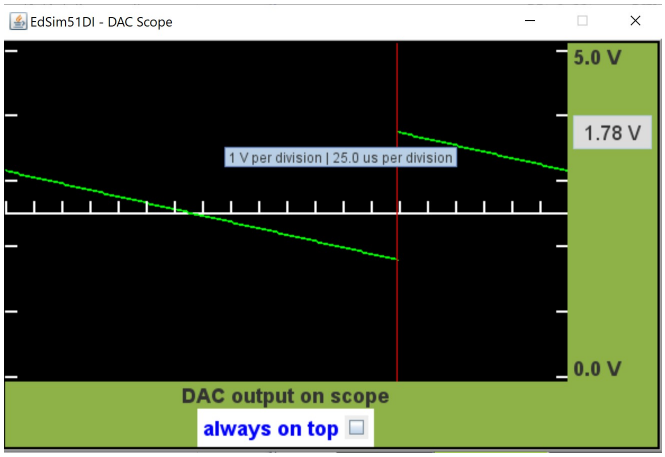


- Write a program that makes a voltage ramp like the one shown below in the Edsim scope screen shown below. Make use of the opcode DEC.

DAC - Notepad

File Edit Format View Help

```
main:
        mov P1, #0FFh; Intialize Max Value 5V
        loop:
        CLR P0.7; Turn on DAC
        DEC P1; Decrement DAC
        SETB P0.7; Reset DAC

        sjmp loop; Repeat Endlessly
```

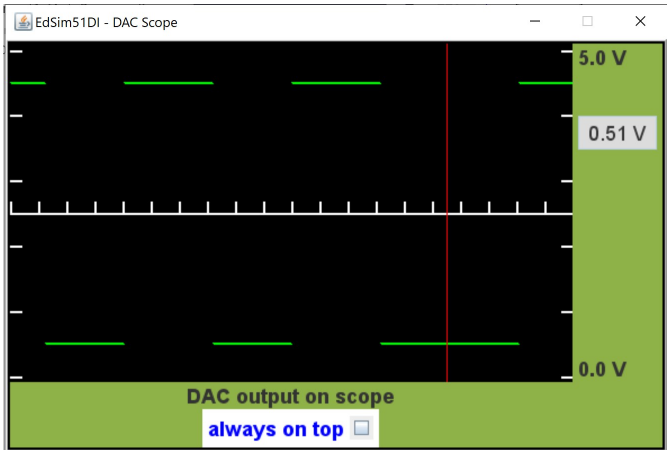


- Continuing with the DAC, write another program to make a square wave that oscillates between 0.5 volts and 4.5 volts. Note that the scope will not show "edges" of the square wave, since the slew rate or change between levels is infinite in the simulation.

Square - Notepad

File Edit Format View Help

```
main:
        mov R0, #10h; Set Time Delay for High
        mov R1, #10h; Set Time Delay for Low
        loop:
        CLR P0.7; DAC Turn On
        mov P1, #0E6h;  Set to 4.5 V
        DJNZ R0, loop; Iterate for Delay Time
        loop2:
        mov P1, #1Ah; Set to 0.5
        DJNZ R1, loop2; Iterate for same Delay Time
        SETB P0.7; Reset WR pin
        sjmp main
```

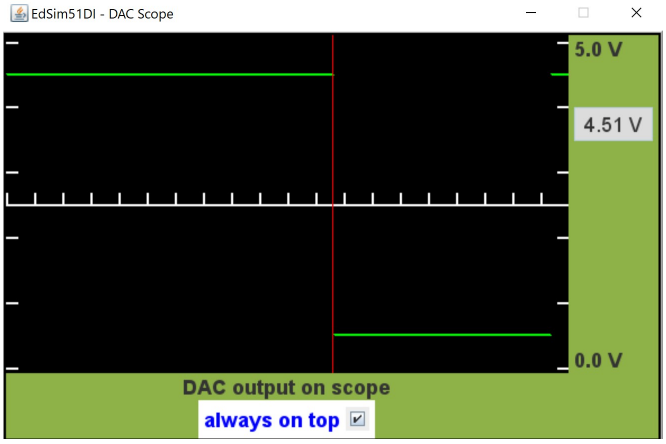


- Let's make another version of the DAC Square Wave generator with a little more flexibility. At the very start of the program, use opcode PUSH to load the 8051 stack with a "high value' (like 4.5) and a low value (like 0.5). Then enter a main routine that uses the opcode POP to load the high and low values into R0 and R1. The main routine should use the opcode LCALL to call a subroutine named "dacme" that loads the value in the accumulator acc to the DAC. Create the square wave by having the main routine put a high value ACC, call dacme, put a low value in ACC, call dacme, and then loop forever to continue creating the square wave. then create a second version of this program that lets you adjust the duty cycle and period of the waveform based on values in R2 and R3.

PUSHPOP2 - Notepad

File Edit Format View Help

```
mov R0, #0E6h; Load High 4.5 in Register 0
mov R1, #1Ah; Load Low 4.5 in Register 1
mov R2, #2h; Duty Cycle can be controlled by varying relation between R2 and R3
mov R3, #2h; Period
PUSH 0; Add to Stack
PUSH 1; Add to Stack
main:
        POP 0;Take from Stack
        POP 1;Take from Stack
        PUSH 0; Readd to Stack or else this will fail after 1 cycle
        PUSH 1;Readd to Stack or else this will fail after 1 cycle

        mov A, 0; Give accumulator value in Register 0
        LCALL dacme; Call Dacme
        loop:
                DJNZ R2, loop; Show Value for Sepcified Duty Cycle
        mov A, 1; Give accumulator value in Register 1
        LCALL dacme; Call Dacme
        loop2:
                DJNZ R3, loop2; Show Value for Sepcified Duty Cycle
        sjmp main; Reset

dacme:
        CLR P0.7; Activate DAC
        mov P1, A; Move value of accumulator into Pin 1
        ret; Return to main
```

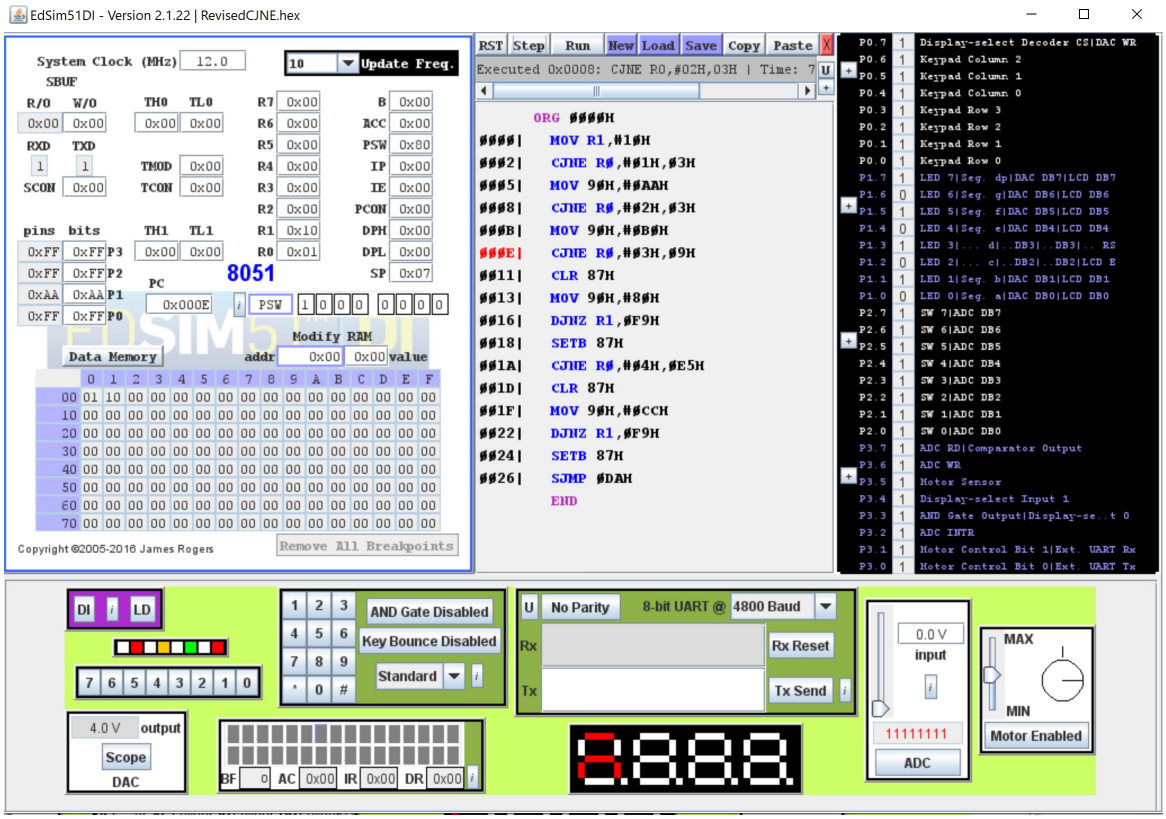


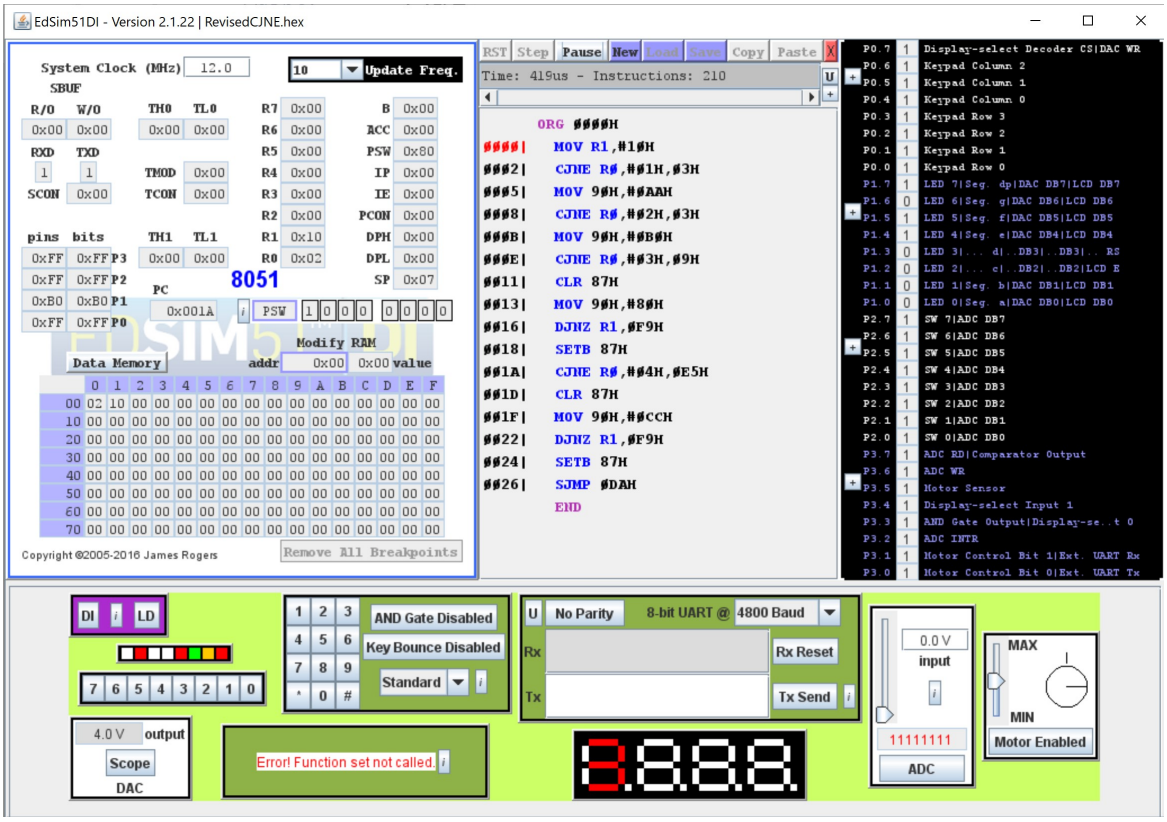
- Write a program that uses the opcode CJNE (more than once) to select and call different functions when given different input numbers in R0. Your program should respond to four possible command values in R0, the numbers 1,2,3,4. A command value of 1 should output AAh to P1. A command value of 2 should display "3" on DISP 3. A command value of 3 should set the DAC output to 2.5 volts. Finally a command value of 4 should set the DAC output to 4 volts. Note that, in Edsim, you can always pause the program, click on r0 in area I, directly modify the value in R0, and run the program. That way you should be able to explore the effect of different command values without having to rerun AS31 or retype new programming in Edsim area II.

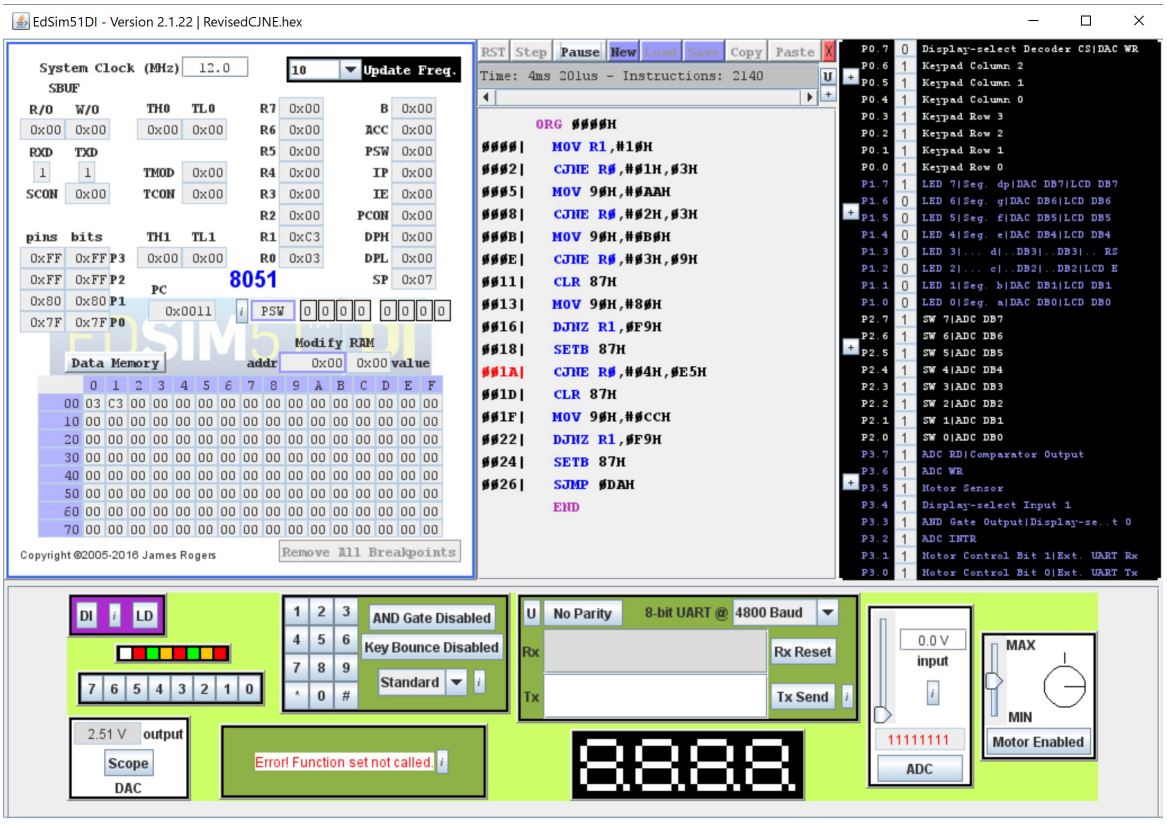
RevisedCJNE - Notepad

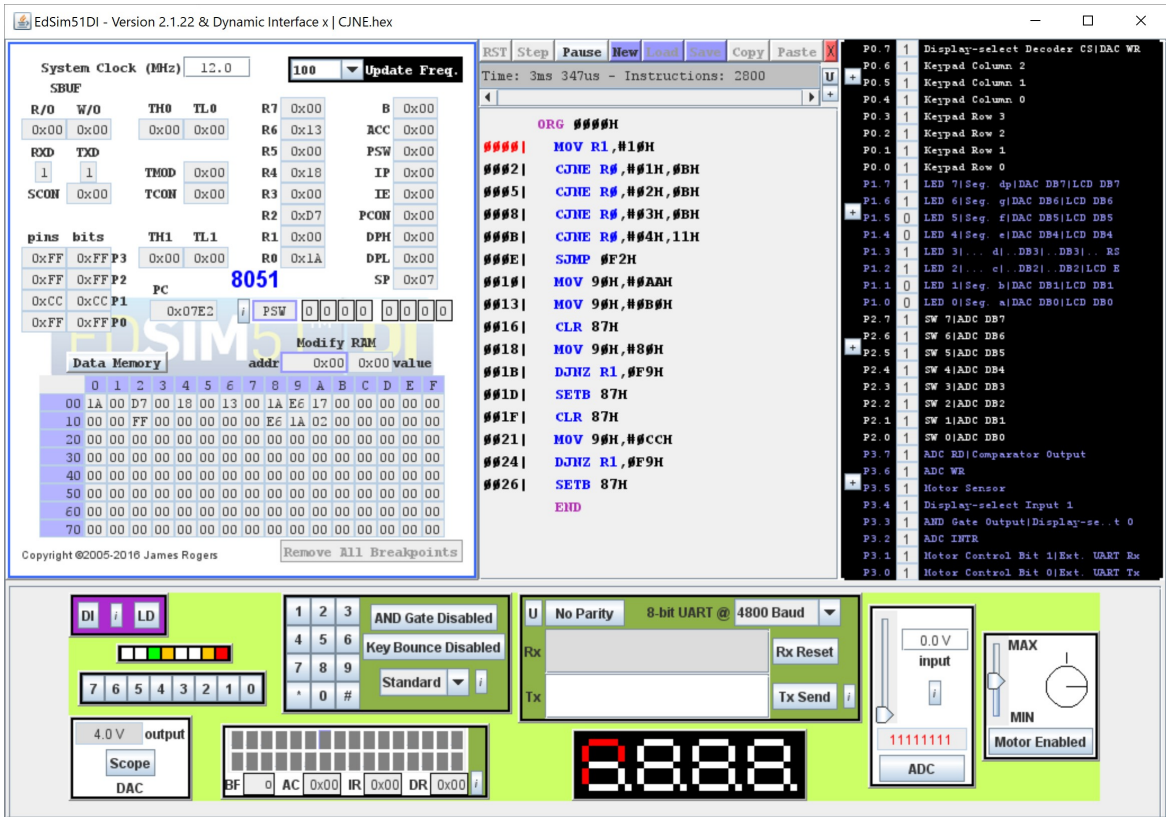
File Edit Format View Help

```
mov R1, #10h; Set Initial Delay for DAC
main:
        s1:
        CJNE R0, #01h, s2; Check if Register R0 has a 1, if not progress to 2
        mov P1, #0AAh; If so, update P1 with AAh
        s2:
        CJNE R0, #02h, s3; Check if Register R0 has a 2, if not, progress to 3
        mov P1, #0B0h; If so update P1 to display a 3
        s3:
        CJNE R0, #03h, s4; Check if Register R0 has a 3, if not progress to 4
        loop:
                CLR P0.7; DAC Turn On
                mov P1, #80h; Set DAC to 2.5 V
                DJNZ R1, loop
                SETB P0.7; Reset DAC
        s4:
        CJNE R0, #04h, s1; Check is Register R0 has a 4, if not cycle back through
        loop1:
                CLR P0.7; DAC Turn On
                mov P1, #0CCh; Set DAC to 4V
                DJNZ R1, loop1
                SETB P0.7; Reset DAC
        sjmp main;
```




EdSim51DI - Version 2.1.22 | RevisedCJNE.hex
System Clock (MHz) 12.0
Update Freq.
8051
Time: 419us - Instructions: 210
Data Memory
Modify RAM
Copyright ©2005-2016 James Rogers
Remove All Breakpoints
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
4.0 V output
Scope
DAC
Error! Function set not called.
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled

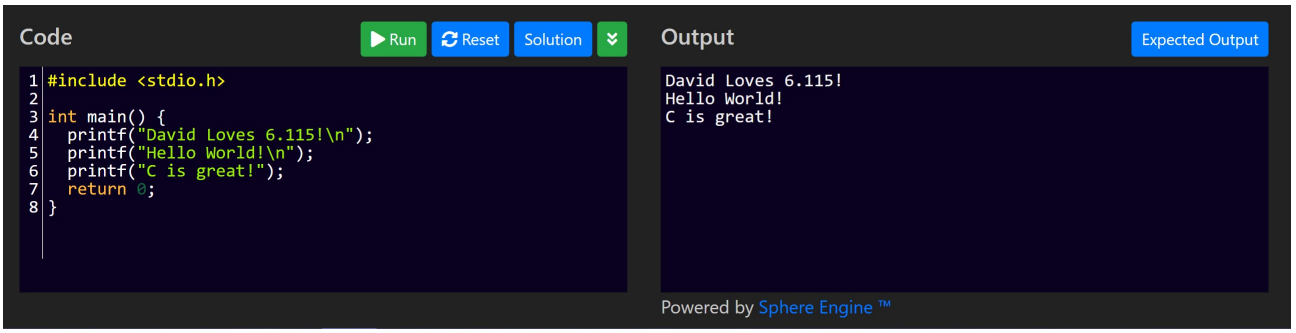

EdSim51DI - Version 2.1.22 | RevisedCJNE.hex
System Clock (MHz) 12.0
Update Freq.
8051
Time: 4ms 201us - Instructions: 2140
Data Memory
Modify RAM
Copyright ©2005-2016 James Rogers
Remove All Breakpoints
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
8-bit UART @ 4800 Baud
Rx Reset
Tx Send
2.51 V output
Scope
DAC
Error! Function set not called.
0.0 V input
11111111
ADC
MAX
MIN
Motor Enabled



3 Exercise 4: See About C

- Complete the 10 "Learn the Basics" exercises, including "Hello World!." through "Static". Also complete the following exercises in the "Advanced" section: Pointers, Arrays and Pointers. Comment your code and add it to your lab report.

1. Hello World!



2. Variables and Types

Code | Run | Reset | Solution

```c
#include <stdio.h>

int main() {
  unsigned int leeb = 3;
  float dologan = 4.5;
  double sixoneoneonefive = 5.25;
  float sum;

  /* Your code goes here */
  sum = leeb + dologan + sixoneoneonefive;
```

Output | Expected Output

```
The sum of leeb, dologan, and 6115 is 12.750000.
```

Powered by Sphere Engine ™

3. Arrays

Code | Run | Reset | Solution

```c
  /* TODO: define the grades variable here */
  int average;
  int voltages[7];
  voltages[0] = 4;
  voltages[1] = 2.5;
  voltages[3] = 1;
  voltages[4] = 2;
  voltages[5] = 3.3;
  voltages[6] = 6;
  voltages[7] = 12;
```

Output | Expected Output

```
The average of the 7 voltages is: 10
```

Powered by Sphere Engine ™

4. Multi-Dimensional Arrays

Code | Run | Reset | Solution

```c
        grades[1][0][1] = 85;
        grades[1][1][2] = 80;
        grades[1][2][3] = 80;
        grades[1][3][4] = 82;
        grades[1][4][3] = 87;

        /* TODO: complete the for loop with appropriate
        for (i = 0; i < 1; i++) {
            average = 0;
            for (j = 0; j < 1; j++) {
```

Output | Expected Output

```
The code written by David in subject 0 is: 677275392.00
```

Powered by Sphere Engine ™

5. Conditions

Code | Run | Reset | Solution

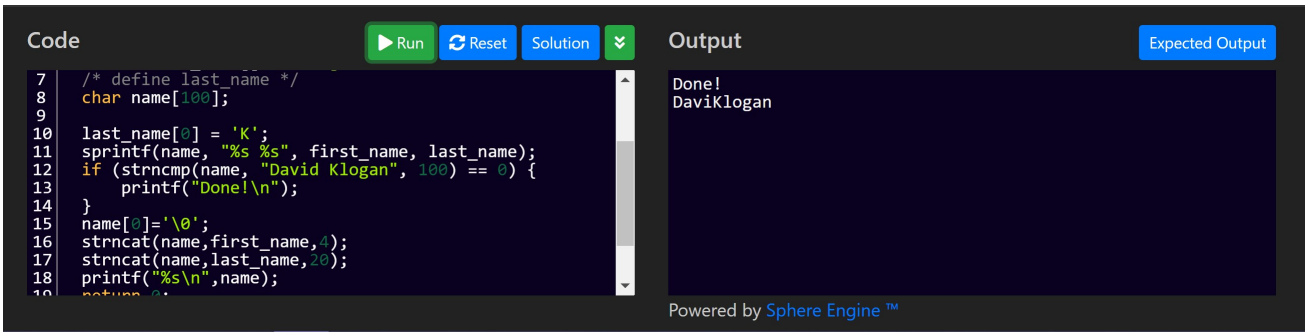
```c
#include <stdio.h>

void guessval(int guess) {
    // TODO: write your code here
    if (guess == 555&& guess < 600){
        printf("Correct. you guessed it!\n");}
    if (guess < 555){
        printf("Your guess is too low.\n");}
    if (guess > 555 || guess >900){
        printf("Your guess is too high.\n");}
}
```

Output | Expected Output

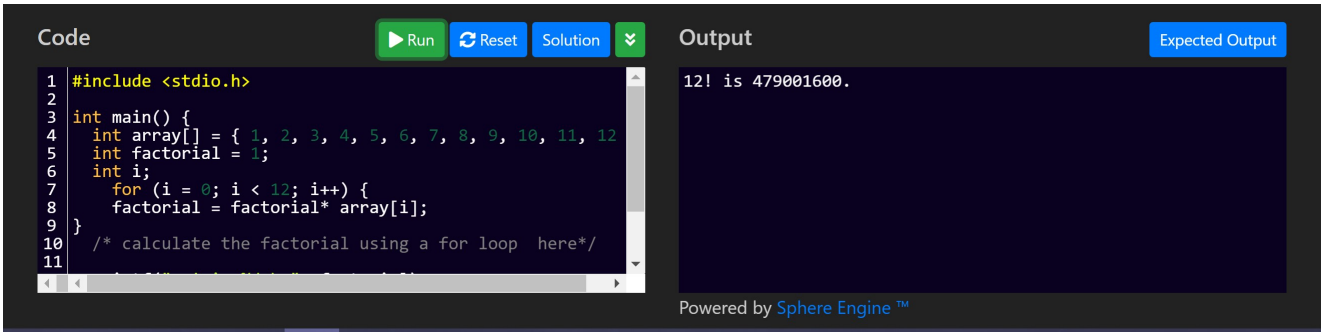
```
Your guess is too low.
Your guess is too high.
Correct. you guessed it!
```

Powered by Sphere Engine ™

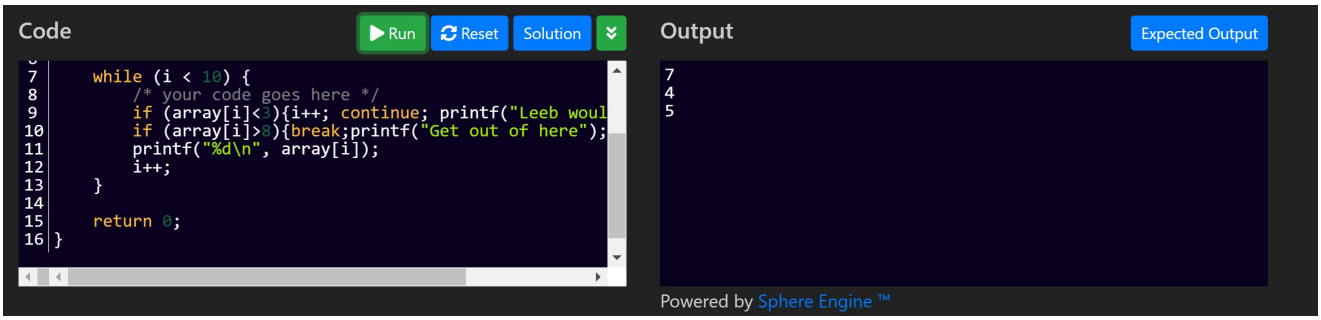
6. Strings



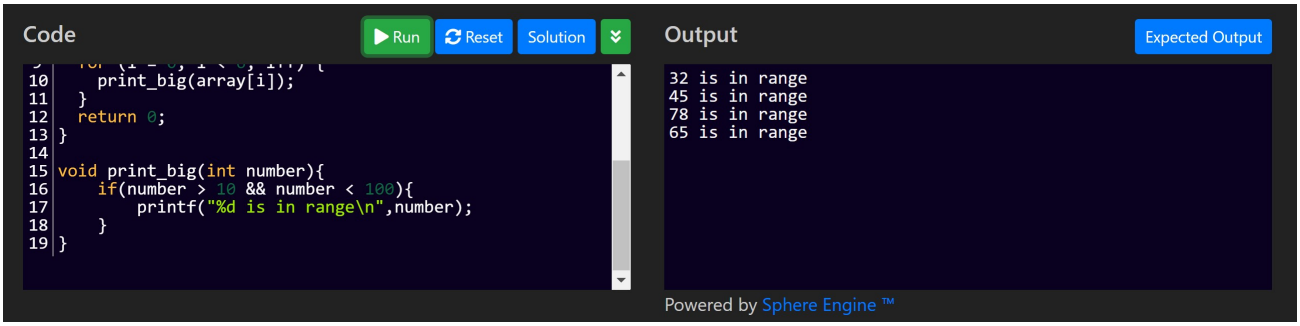
7. For Loops



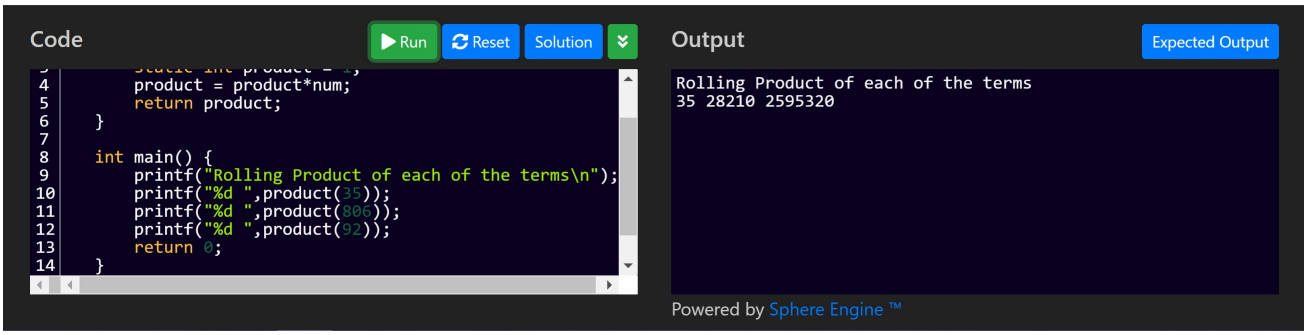
8. While Loops



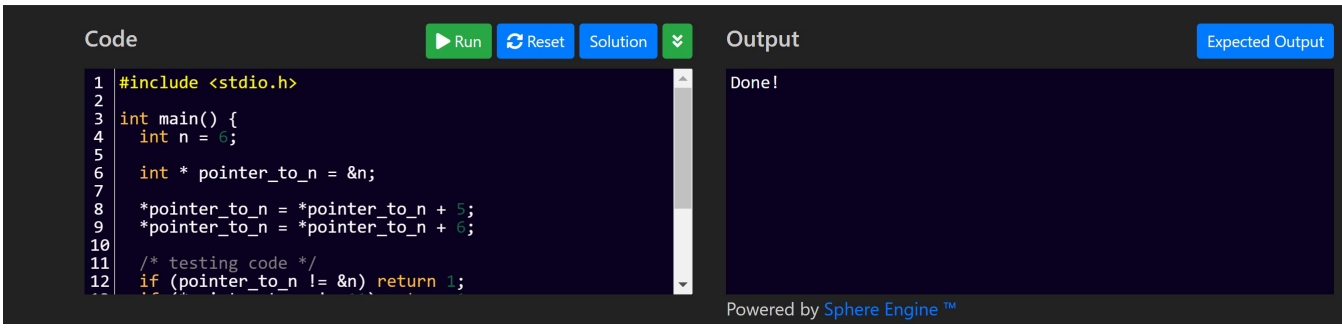
9. Functions



10. Statics



11. Pointers



12. Arrays and Pointers