



**EECE 340 – Introduction to Microprocessors w/Lab**  
**Section A**

**Term Project**

*Parking Visitor Counter*

**Group Members:**

*Joseph Bejjani*

*Mohammad Omar Al Abrach*

*Sharbel Dahlan*

**Instructor:** *Dr. Jinane Biri*

**Due date:** *Sunday, Dec. 16, 2012*

Table of Contents

1. Objective ..... 2

2. Introduction and Problem Description ..... 2

    2.1. Hardware..... 2

    2.2. Software ..... 3

3. Procedure ..... 3

    3.1. Building the circuit ..... 3

    3.2. Writing the code..... 3

    3.3. Pre-testing the system ..... 3

    3.4. Testing the system..... 4

4. Program Logical Flow ..... 4

    4.1. General Program Flow ..... 4

    4.2. Detailed Flow ..... 5

5. Code ..... 9

6. Code Analysis ..... 11

7. Hardware Design and Schematics..... 12

8. Results ..... 14

    8.1. Testing the system..... 14

    8.2. Register Values and Output LED States ..... 15

9. Conclusion and Future Improvements ..... 17

## 1. Objective

The purpose of this project is to learn and be able to use the instructions of an 8085 microprocessor in a code, as well as the come up with the hardware, in order to implement a real life application.

## 2. Introduction and Problem Description

We have been learning code creation and hardware connection to enable the microprocessor to perform different tasks, thereby exploring the wide range of applications the simple educational 8085 microprocessor can do. Now, many of the real-life applications are recognized to be implemented using microprocessors, whose simplest forms (like the 8085) can actually do much of those applications that seem to be complex. In this project, we have used the microprocessor to fully design a simple parking visitor counter system. The system is composed of three things:

1. The **sensors** located at each of the parking entrance and the exit gates to detect the cars passing.
2. The **Primer Trainer Kit**, which contains the 8085 microprocessor and relates the hardware part of the system to its code.
3. The **counter display**, which will show the number of cars that are currently inside the parking. The counter will increment or decrement depending on whether a car enters or exits the parking, respectively.

When the counter reaches its maximum, it will keep displaying the maximum number and therefore any interrupt to the sensors will not change the display. The same applies to when the counter reaches its minimum value, where the display will remain zero.

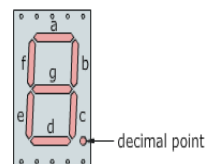
### 2.1. Hardware

The hardware is composed of the following:

- Two pairs of **infrared sensors** like the ones shown in *Figure 1*. Consisting of a phototransistor and a LED, each pair is put on separate locations to represent “entrance” and “exit” gates.
- The **7-segment display**, which has the role of the counter display. Because we are using a one-digit seven segment display, we are limited to having the maximum number of 9.
- The 7-segment BCD **decoder** to convert the logic states at the outputs of the CN<sub>3</sub> port.
- Two 150  $\Omega$  and two 150 k $\Omega$  **resistors** to be connected to the sensor circuits.
- The **Primer Trainer Kit** which contains the microprocessor and the peripheral connections to the other hardware components.



**Figure 1: The IR LED Sensors** (image from: <http://www.digikey.com>)



**Figure 2: The 7-segment display** (image from Project Assignment)

## 2.2. Software

The instructions that compose the code are based on a simple logical method that lets the hardware give the desired result. Although the problem suggests that interrupts are better used to build the code, limitations of the apparatus has made us used another method called polling. This method is based on infinite looping, where the program continuously takes input from the sensors and displays the counter value. The software accounts for the interruptions of the sensor even after the extremes are reached and keeps each extreme number displayed (according to the case). The software also includes a delay between the stages of checking the input and changing accordingly, to ensure that no error happens while counting an entering or exiting car that might take time. The delay duration can be set and modified by the coder for any duration needed.

## 3. Procedure

**3.1. Building the circuit** First, we had to build the hardware of the project by doing the following:

- *The output circuit:* Connecting the BCD to 7-segment decoder with the 7-segment display, and checking that the connections work by testing the output of the 7-segment display for different BCD input combinations.
- *The input circuit:* Connecting the IR sensors with their resistors and testing the circuit by trying to interrupt the signal from the phototransistor to the receiver and checking how there will be a voltage value (of 4 V); otherwise there will be no voltage.
- *The processor circuit:* Connecting both the *input* and *output* circuits through the Primer Trainer Kit.

**3.2. Writing the code** The program for this experiment was written and loaded into memory. The code is composed of the following parts which will be explained in later sections such as in the *Program Logical Flow* and *Code Analysis* sections:

- Obtaining the voltage state (high or low) from the *input circuit*.
- Processing the input to see which case should give what output.
- Creating the output and letting the *output circuit* to display it.

**3.3. Pre-testing the system** After loading the program into memory, we have tested the Primer Trainer Kit by checking how its output LEDs turn on or off in different combinations according to different inputs.

**3.4. Testing the system** We have tested the system by checking all the possible cases it might encounter, including extreme cases, as explained below:

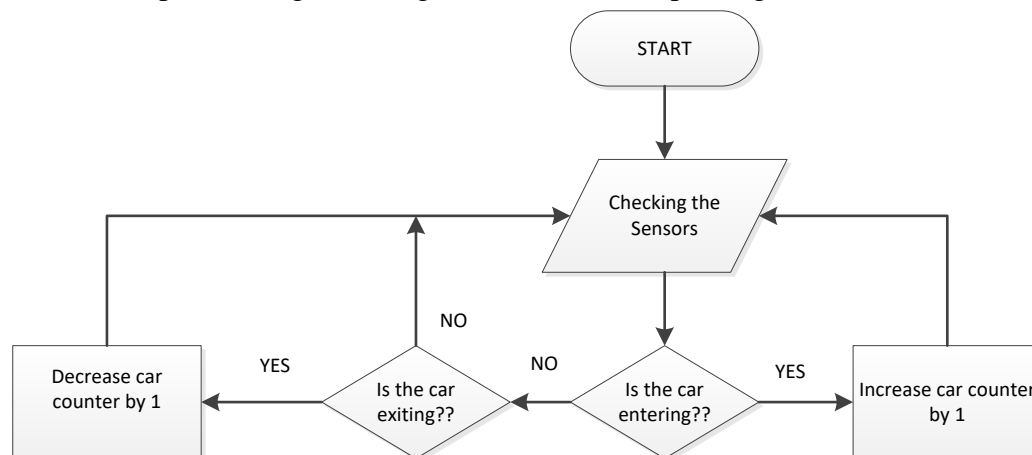
- *Normal cases:*
  - If the sensors of the entrance gate are interrupted, the counter display should be incremented.
  - If the sensors of the exit gate are interrupted, the counter display should be decremented.
- *Extreme cases:*
  - If the sensors of the entrance gate are interrupted after the counter display has reached its maximum, the counter should keep displaying its maximum (which is 0).
  - If the sensors of the exit gate are interrupted after the counter display has reached its minimum, the counter should keep displaying its minimum (which is 0).
  - If the both sensors at the two gates are interrupted simultaneously (when a car enters and another exits at the same time), the counter should keep displaying its current value (i.e. if it was at 5, it should remain 5).

## 4. Program Logical Flow

The following subsections contain flowcharts to represent the logical flow of our program. First, the general algorithm is shown. Next, the detailed algorithm that contains all the main routines and subroutines of the code is explained by several flowcharts.

### 4.1. General Program Flow

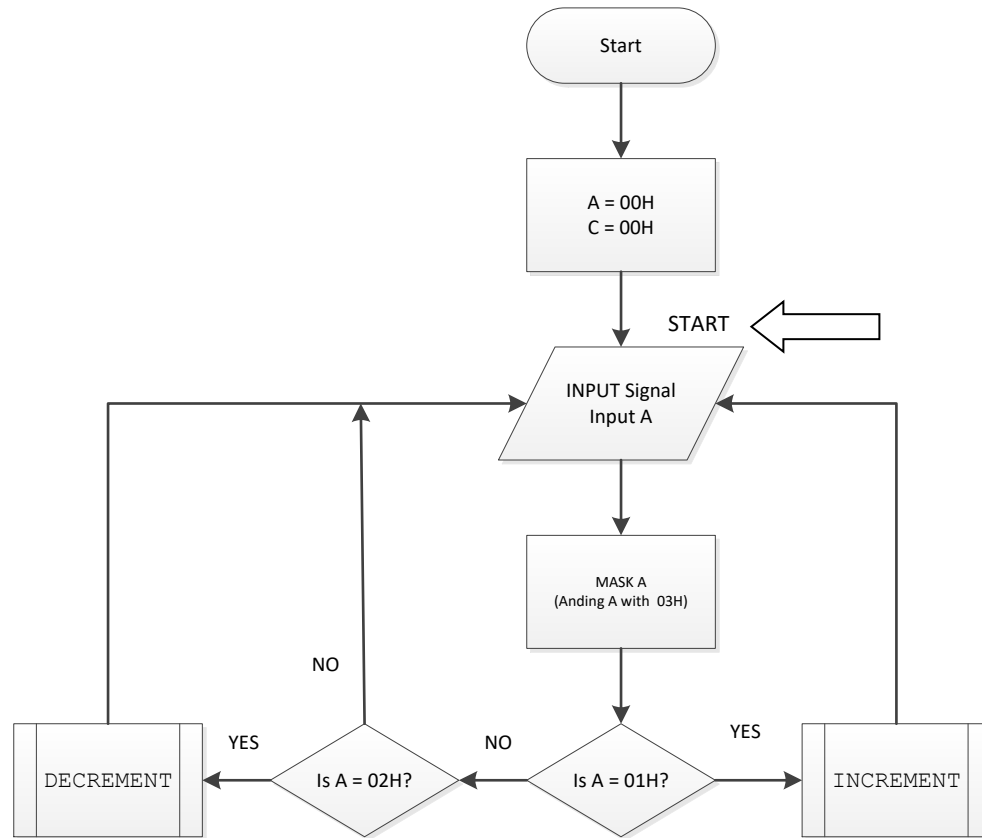
This flowchart explains the general algorithm of how the parking counter works.



**Figure 3: Simplified Flowchart**

## 4.2. Detailed Flow

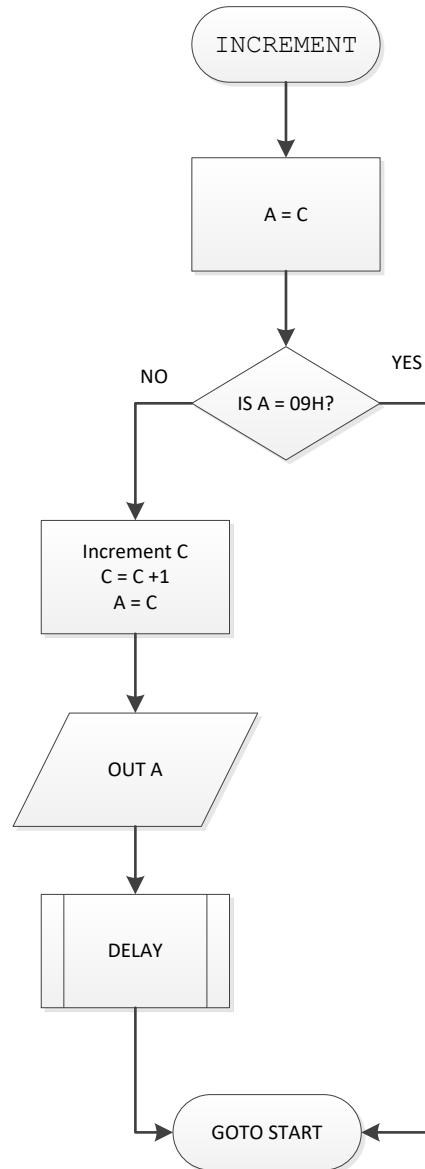
The following flowchart explains the *main* code:



**Figure 4: Main Function**

Before starting our program, we set the A and C registers' values to 00H. Then, the program gets a signal from the sensors. Based on this signal, the program decides whether the car is entering or exiting. If the car is entering the parking, the program calls INCREMENT subroutine, and if it is exiting, the program calls DECREMENT subroutine. However, if there is no car entering or exiting, the program does nothing. START, after variable initialization, is the point where both of INCREMENT and DECREMENT subroutine go after executing.

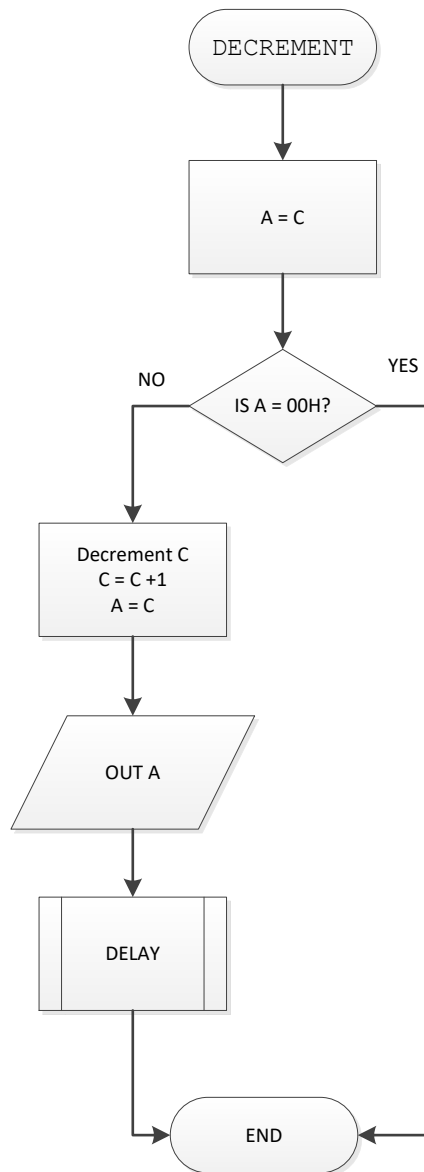
This flowchart explains *INCREMENT* subroutine:



**Figure 5:** Increment subroutine

This subroutine increments the car counter by one, and then it returns to *START* in main function.

This flowchart explains *DECREMENT* subroutine:

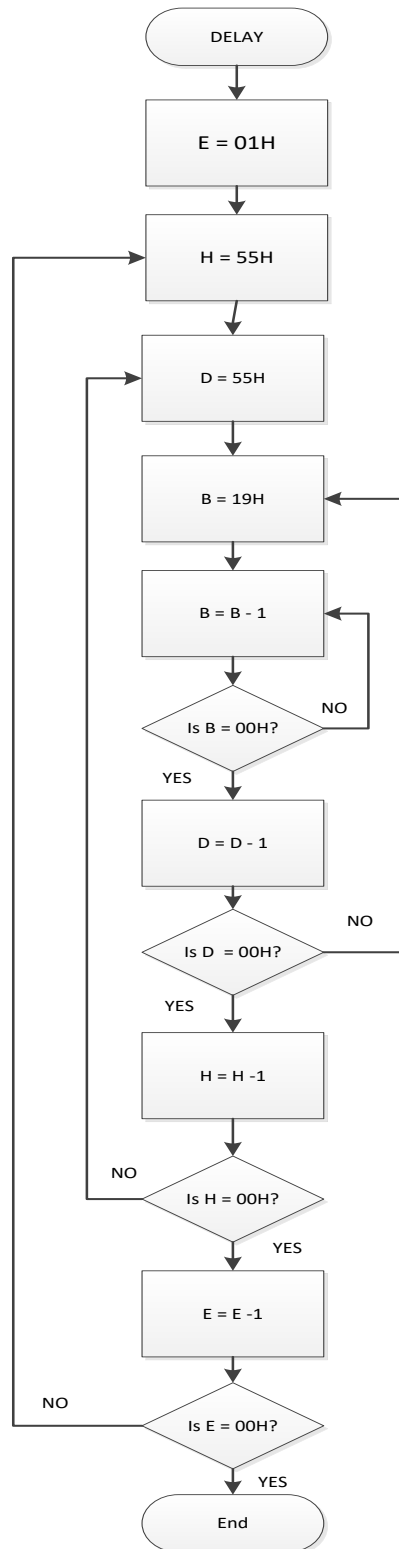


**Figure 6:** Decrement subroutine

This subroutine decrements the car counter by one, and then it returns to START in main function.



This flowchart explains *DELAY* subroutine:



**Figure 7:** Delay subroutine

This subroutine causes a delay on our program. It is essential because it prevents double counting while cars entering or exiting. It is consisted of nested if-else statements.

## 5. Code

This section contains the assembly code used to build the software part of this project, and the corresponding Hex code that was entered to the Primer Trainer Kit.

ADDRESS	DATA	LABEL	INSTRUCTION	COMMENTS
FF00	3E		MVI A,00	; Clear Accumulator
FF01	00			
FF02	0E		MVI C,00	; Initialize the counter
FF03	00			
FF04	D3		OUT 11	; Output to the output LEDs (all
FF05	11			; will be ON in order for the
				; display to be initially 0)
FF06	DB	START	IN 12	; Take the input from the IR
FF07	12			; sensors
FF08	E6		ANI 03	; Mask all the bits and
FF09	03			; keep the two least significant
FF0A	FE		CPI 01	; Compare to 1H, i.e. check if the
FF0B	01			; entrance sensor is interrupted
FF0C	CA		JZ INCREMENT	; If the entrance sensor is
FF0D	17			; interrupted, go to INCREMENT
FF0E	FF			; subroutine
FF0F	FE		CPI 02	; Compare to 2H, i.e. check if the
FF10	02			; exit sensor is interrupted
FF11	CA		JZ DECREMENT	; If the exit sensor is
FF12	27			; interrupted, go to DECREMENT
FF13	FF			; subroutine
FF14	C3		JMP START	; Keep looping infinitely to
FF15	06			; continuously check the system
FF16	FF			; input
FF17	79	INCREMENT	MOV A,C	; Get the current counter value
FF18	FE		CPI 09	; Compare it to 09H (the maximum
FF19	09			; case)
FF1A	CA		JZ START	; If the counter is at its maximum
FF1B	06			; then do nothing and loop to the
FF1C	FF			; beginning (keep counter at 9)
FF1D	0C		INR C	; Increment the counter
FF1E	79		MOV A,C	; Store the counter value in A
FF1F	D3		OUT 11	; Output to the output LEDs (whose
FF20	11			; BCD combination will translate
				; to the 7-segment display output
FF21	CD		CALL TIME_DELAY	; Delay to avoid over-counting a
FF22	37			; car
FF23	FF			

<b>FF24</b>	C3		JMP	START	; Return to the beginning to check
<b>FF25</b>	06				; for new inputs
<b>FF26</b>	FF				
<b>FF27</b>	79	DECREMENT	MOV	A,C	; Get the current counter value
<b>FF28</b>	FE		CPI	00	; Compare it to 00H (the minimum
<b>FF29</b>	00				; case)
<b>FF2A</b>	CA		JZ	START	; If the counter is at its minimum
<b>FF2B</b>	06				; then do nothing and loop to the
<b>FF2C</b>	FF				; beginning (keep counter at 0)
<b>FF2D</b>	0D		DCR	C	; Decrement the counter
<b>FF2E</b>	79		MOV	A,C	; Store the counter value in A
<b>FF2F</b>	D3		OUT	11	; Output to the output LEDs (whose
<b>FF30</b>	11				; BCD combination will translate
					; to the 7-segment display output
<b>FF31</b>	CD		CALL	TIME_DELAY	; Delay to avoid over-counting a
<b>FF32</b>	37				; car
<b>FF33</b>	FF				
<b>FF34</b>	C3		JMP	START	; Return to the beginning to check
<b>FF35</b>	06				; for new inputs
<b>FF36</b>	FF				
<b>FF37</b>	1E	TIME_DELAY	MVI	E,01	; number of times the 1s time delay
<b>FF38</b>	01				; to repeat, this case one time
<b>FF39</b>	26	DELAY_S	MVI	H,55	; load 55H into the reg. H
<b>FF3A</b>	55				
<b>FF3B</b>	16	DELAY	MVI	D,55	;load 55H into the reg. D
<b>FF3C</b>	55				; the loop will be repeated 85 time
<b>FF3D</b>	06	REPL	MVI	B,19	; load 19H into the reg. B
<b>FF3E</b>	19				
<b>FF3F</b>	05	LOOP	DCR	B	; DCR the value of in Reg. B by 1
<b>FF40</b>	C2		JNZ	LOOP	; Jump to LOOP(FF3F)if B is not
<b>FF41</b>	3F				; Zero
<b>FF42</b>	FF				; the loop will be repeated 25 time
<b>FF43</b>	15		DCR	D	; DCR the value of in Reg. D by 1
<b>FF44</b>	C2		JNZ	REPL	; Jump to REPL(FF3D)if D is not
<b>FF45</b>	3D				; Zero
<b>FF46</b>	FF				; the loop will be repeated 85 time
<b>FF47</b>	25		DCR	H	; DCR the value of in Reg. H by 1
<b>FF48</b>	C2		JNZ	DELAY	; Jump to Delay(FF3B)if H is not
<b>FF49</b>	3B				; Zero
<b>FF4A</b>	FF				; the loop will be repeated 85 time
<b>FF4B</b>	1D		DCR	E	; DCR the value of in Reg. E by 1
<b>FF4C</b>	C2		JNZ	DELAY_S	; Jump to DELAY_S(FF39)if E is not
<b>FF4D</b>	39				; Zero
<b>FF4E</b>	FF				; the loop will be repeated once
<b>FF4F</b>	C9		RET		; return to program

## 6. Code Analysis

The code is divided into main routines and subroutines, each of which is explained individually:

- **Initialization:** First, both the accumulator and the C register (which will act as the counter) are initialized.
- The `OUT 11` instruction will output the binary value of the accumulator by lighting the output LEDs of the Primer Trainer Kit. The output LEDs are active low, so if the accumulator has a value of 00H, all the 8 output LEDs will be on. *Figure 8* in the *Hardware Design and Schematics* section shows a picture of the eight output LEDs being lit at the initial state of the system (when the counter is zero).
- *The main code* starts at the “START” label and is composed of performing incrementing, decrementing, or none, according to the input of the system, which is the sensors. In other words, it uses conditionals to jump to different subroutines according to the input of the system. First, the input is taken from the IR sensors. Then, this input is ANDed with 03H, which is equivalent to 0000 0011 in binary. This ANDing will cause the input to be “masked”, and only the least two significant bits remain showing (the others are Zeroed). The reason for the masking is that only the least two significant bits are needed to determine whether the input means that a car is entering, exiting or none. The masking is illustrated below:

Input:								LSB
	x	x	x	x	x	x	x	x
ANDed with 03H							1	1
Gives	0	0	0	0	0	0	x	x

Those two least significant bits of the inputs can have 4 possible combinations (00, 01, 10, 11), which basically represent the 4 possibilities of an input to this system (car entering, car exiting, no car entering or exiting, a car entering and another exiting simultaneously). In our code, the first two cases are represented as follows:

**Table 1: Input combinations meaning**

Least two significant bits	Meaning
0 1	Car entering (entrance IR sensors interrupted)
1 0	Car exiting (exit IR sensors interrupted)

The other two cases were not shown here because they are both dealt with in the same way in the code. That is, if none or both of the sensors are interrupted, the counter value must not

change. What the code does next is that it compares the masked input immediately to 01H or 02H and jumps to different subroutines accordingly. So the conditionals actually jump to the two increment and decrement subroutines depending on the two combinations of input LSBs as shown above in *Table 1*. If after the comparison, the input was equivalent to neither of 01H or 02H, the program loops to the start to check for new inputs. The reason for using 01 to indicate the interruption of the entrance gate and 10 for the exit gate is explained in the *Hardware Design and Schematics* section.

- *The Increment and Decrement* subroutines deal with changing the counter display according to the current state and the new input. Both subroutines work similarly by taking the input and first comparing it to the extreme case (0 when decrementing or 9 when incrementing) in order to keep the extreme values displayed if they were initially reached. Then, if the extreme case was not the initial state of the counter, the subroutine does its job to either increment or decrement the counter.
- *Time Delay*: At the end of each of the increment or decrement subroutines, the time delay subroutine is called. This subroutine is significant to avoid over-counting a car that enters or exits. Since the program loops continuously to check for inputs, it is not desirable to have it loop fast to the extent that it takes the input of the car interrupting the sensors more than once. Therefore, the time delay, which can be adjusted by the programmer, is used.

## 7. Hardware Design and Schematics

Part (a) of the following figure shows the schematics for the *output circuit*.  
Part (b) of the following figure shows the schematics for the *input circuit*.

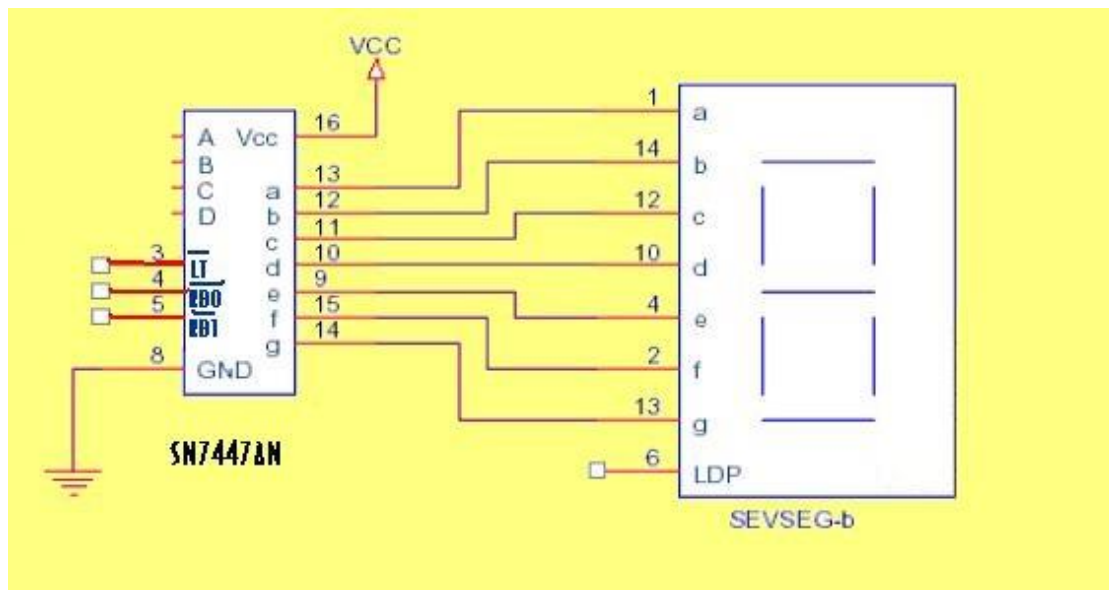


Figure 8a: Seven Segment Display connections

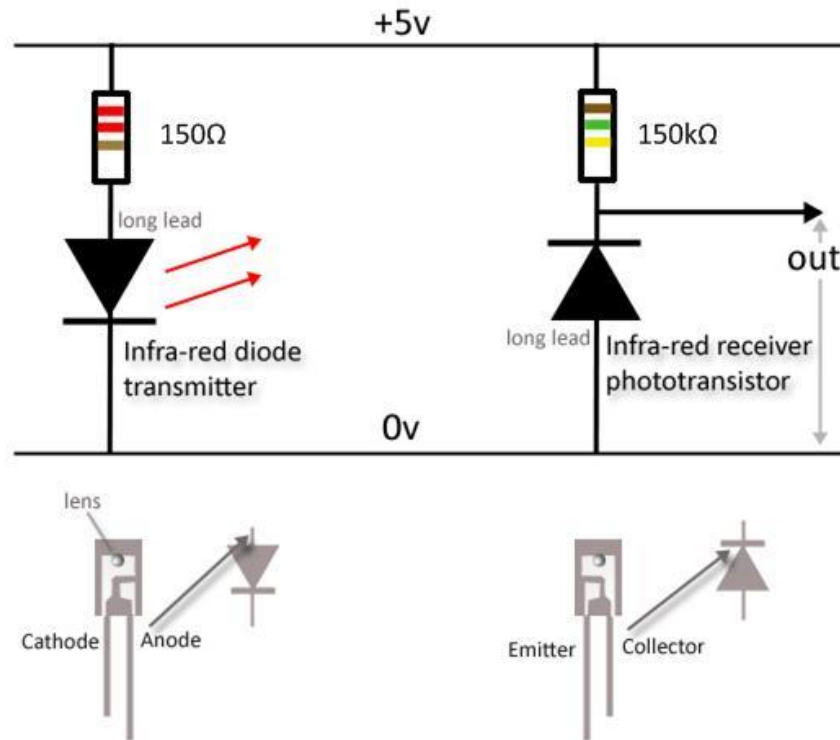


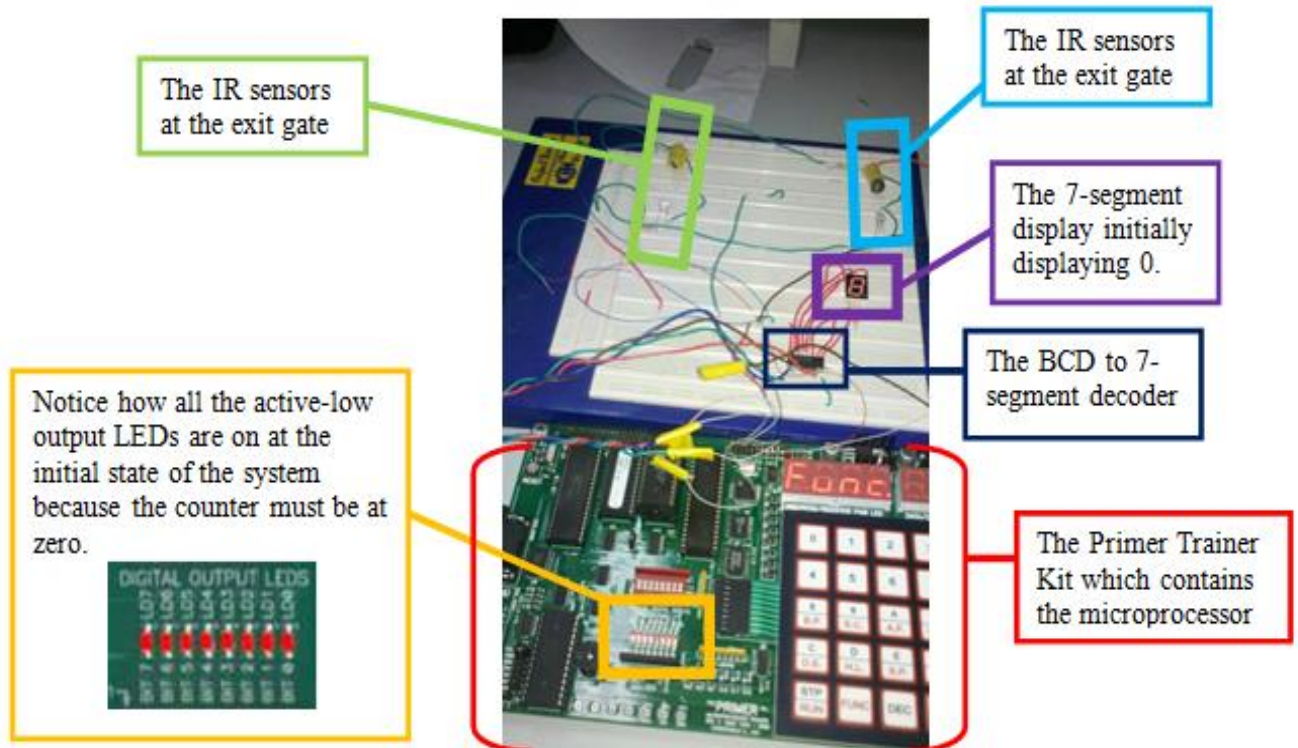
Figure 8b: Infrared Connections

Note that when the IR sensors are interrupted in the *input circuit*, a high voltage signal (a signal of 1) is sent to the *processing circuit*, while when not interrupted, a low voltage signal is sent (a signal of 0). This explains why we are using 01 to tell that the entrance sensors are interrupted and 10 to tell that the exit sensors are interrupted.



## 8. Results

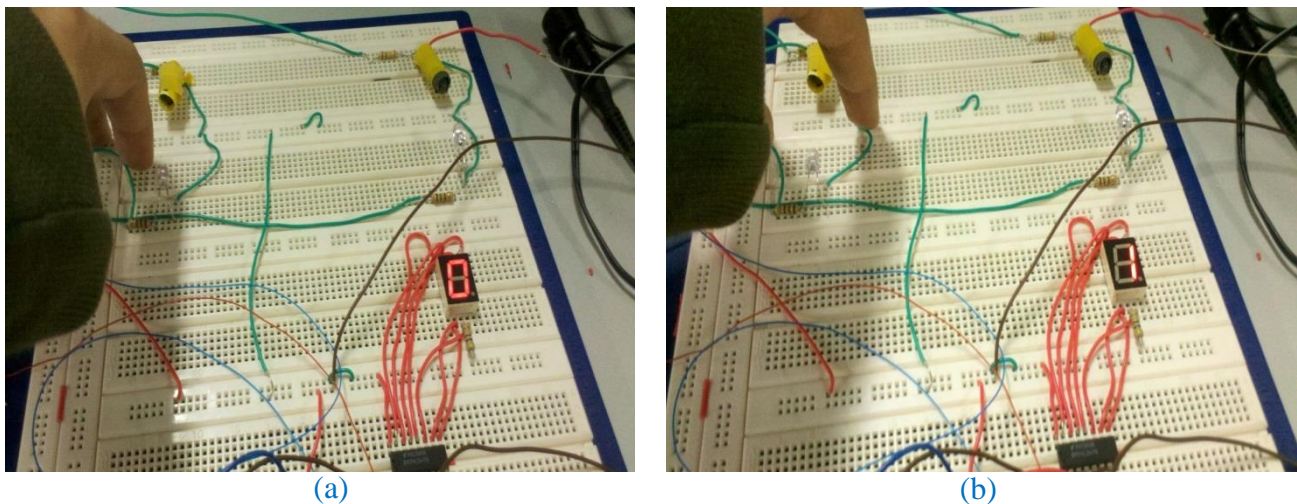
The system is shown and identified in its initial state in *Figure 9*.



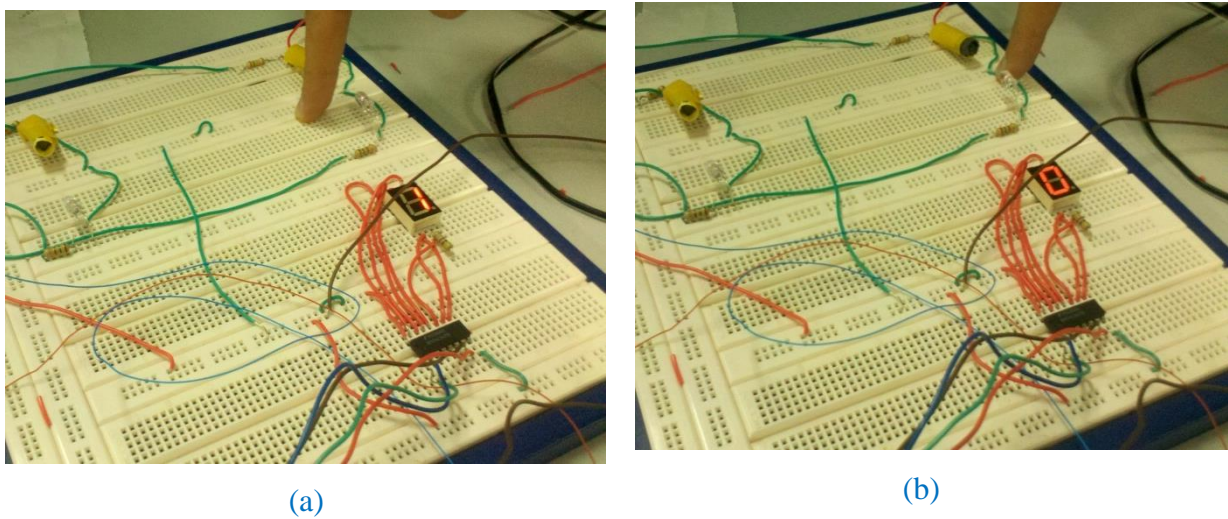
**Figure 9:** The hardware of the system at its initial state

### 8.1. Testing the system

*Figures 10 & 11* show two of the normal test cases.



**Figure 10:** Testing a normal case in which the entrance sensors are interrupted and the counter changes value from 0 as shown in (a) to 1 as shown in (b)







**Figure 8:** Testing a normal case in which the exit sensors are interrupted and the counter changes value from 1 as shown in (a) to 0 as shown in (b)

The next section shows more test cases with more details of the results, such as the register values and the states of the LEDs





### 8.2. Register Values and Output LED States

**Test Case 1:** Interrupting the entrance gate sensor with counter initially at 3 (a *normal test case*)

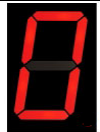
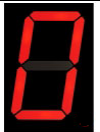


	Before interrupting sensors	After interrupting sensors
7-segment display		
C register	03 H	04 H
A register	03 H	04 H
Primer Trainer Kit output LEDs		
Binary values corresponding to LEDs	0000 0011	0000 0100







**Test Case 2: Interrupting the exit gate sensor with counter initially at 8 (a normal test case)**

	Before interrupting sensors	After interrupting sensors
7-segment display		
C register	08 H	07 H
A register	08 H	07 H
Primer Trainer Kit output LEDs		
Binary values corresponding to LEDs	0000 1000	0000 0111





**Test Case 3: Interrupting the exit gate sensor with counter initially at 0 (an extreme test case)**

	Before interrupting sensors	After interrupting sensors
7-segment display		
C register	00 H	00 H
A register	00 H	00 H
Primer Trainer Kit output LEDs		
Binary values corresponding to LEDs	0000 0000	0000 0000

**Test Case 4: Interrupting the entrance gate sensor with counter initially at 9 (an extreme test case)**

	Before interrupting sensors	After interrupting sensors
7-segment display		
C register	09 H	09 H
A register	09 H	09 H
Primer Trainer Kit output LEDs		
Binary values corresponding to LEDs	0000 1001	0000 1001

**Test Case 5: Interrupting both entrance and exit gate sensors with counter initially at 5 (an extreme test case)**

	Before interrupting sensors	After interrupting sensors
7-segment display		
C register	05 H	05 H
A register	05 H	05 H
Primer Trainer Kit output LEDs		
Binary values corresponding to LEDs	0000 0101	0000 0101

## 9. Conclusion and Future Improvements

At the end of this project, we have seen another application of the microprocessor and how we can use it in order to design a parking visitor counter system. We were able to come up with both the program and the hardware of the system by using what we have in hand from previous labs since it covers concepts including arithmetic and logical operations, delay loops, subroutines, and hardware configurations. Particularly for this project, we have seen how there is more than one way to solve a problem and build a system since we were not able to use interrupts due to limitations but have used continuous looping instead.

Since our parking counter is very limited in number (a maximum of 9 cars), one of the **future improvements** we might think of is the increasing the capacity of the counter by using a two-digit seven-segment display. Another idea is to have LED light indicator near the entrance gate to show if the parking is fully occupied or not. For example, if there is vacancy in the parking, the green LED light is turned on. If the parking is full, the red LED light turns on so that no car enters. Both ideas are not actually hard to implement and just need logic and codes similar to those we have used.