暨南大學
JINAN UNIVERSITY

Undergraduate Lab Report

Course Title: Experiment of Computer Organization

Course No: 60080014

Student Name: ______________________

Student No: ______________________

School: International School

Department: ______________________

Major: Computer Science & Technology

Instructor： SUN Heng

Academic Year:202~202, Semester: 1st [√] 2nd[ ]

Academic Affairs Office of Jinan University

Date (dd/mm/yyyy)__________

Computer Organization Lab List

Student Name:________________ Student No:_______________

ID	Lab Name	Type
1	Number Storage Lab	Individual
2	Manipulating Bits	Individual
3	Simulating Y86-64 Program	Individual
4	Performance Lab	Team
5	A Simple Real-life Control System	Team
6	System I/O	Individual

Undergraduate Lab Report of Jinan University

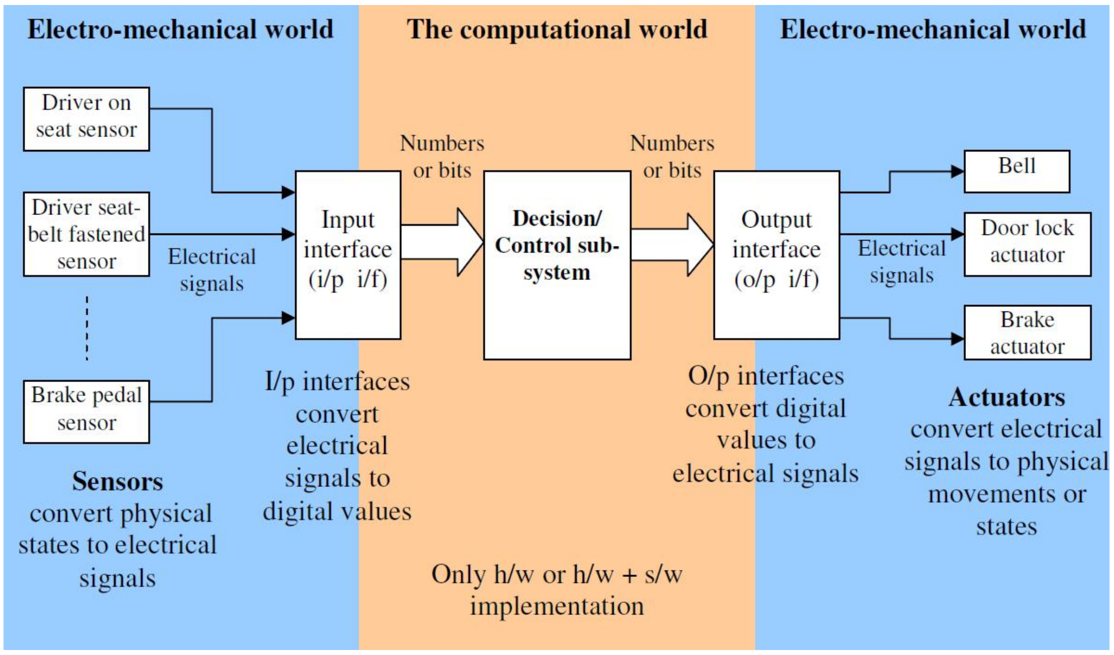
Course Title__Experiment of Computer Organization__Evaluation______
Lab Name A Simple Real-life Control System Instructor___SUN Heng__
Lab Address____
Student Name__________________________Student No.______________
College________International School_____________________________
Department__________________________Major_______CST_________
Date________/________/________ Afternoon

1. Introduction

This lab involves in the I/O of chapter 10. Please be patient to read full instructions.

2. Lab Instructions or Steps

Assume you are hired by Ford Motor Company to develop the embedded software for their new model of cars. This software executes on the main computer that sits inside the car. Your manager wrote down the "Software Requirement Specification (SRS)" document and gave it to you. Based on the "requirements" listed in this document you have to develop a C program that materializes these requirements/specifications. Given below is a portion of the SRS document which includes the schematic of the car's electronics and computer systems:



Electro-mechanical world
The computational world
Electro-mechanical world
Driver on seat sensor
Driver seat-belt fastened sensor
Brake pedal sensor
Electrical signals
Input interface (i/p i/f)
Numbers or bits
Decision/ Control sub-system
Numbers or bits
Output interface (o/p i/f)
Electrical signals
Bell
Door lock actuator
Brake actuator
Sensors convert physical states to electrical signals
I/p interfaces convert electrical signals to digital values
O/p interfaces convert digital values to electrical signals
Actuators convert electrical signals to physical movements or states
Only h/w or h/w + s/w implementation

The sensors convert the physical states (pressure induced by the driver on the seat when he is seating on it, etc.) and mechanical motions (driver pressing the brake pedal) to electrical signals (voltage, current). The input interface sub-system (commonly denoted as “i/p i/f”, “i/p” for input, “i/f” for interface) converts electrical signal into digital form (number, bits etc.). The decision/control sub-system is the main functional block which provides the intelligence to the car so that it can take the right action based on the given situation. The decision logic provides digital output (numbers, bits), which is converted back to electrical signals by the Output interface (o/p i/f). These electrical signals are used to activate the electro-mechanical actuators (motors, electro-magnets etc.). For this particular lab class, your job is to *design the decision control logic*. You will design the i/p i/f and o/p i/f sub-systems.

Available sensors -

Each hardware sensor provides a “high” (1) or “low” (0) output. The input interface sub-system sets the value of the corresponding global integer variable to the respective value. The decision/control logic sub-system will read this integer to take the right decision. It is assumed that this integer is available for C programming. The i/p i/f sub-system keeps monitoring the sensor hardware outputs and takes action (changes value of the integer variable) as soon as the sensor output state changes.

1. DOS – driver on seat. This sensor indicates whether a driver is present. This sensor provides logical “high” (1) as output when a person is sitting on the driver’s seat and “low” (0) if he is not on the seat.

2. DSBF – driver seat belt fastened. This sensor indicates whether the driver seat belt is fastened or not. The sensor hardware provides “high” when driver’s seat belt is fastened, “low” otherwise. The corresponding integer variable that reflects the physical states inside the computer and programming world is “driver_seat_belt_fastened”. The i/p i/f sub-system sets this integer to 1 when DSBF output is high and to 0 when DSBF is low, and the decision/control sub-system code reads/uses it.

3. ER – engine running. This sensor indicates whether engine is running or not. It provides “true” when engine is running, false otherwise. The

corresponding integer variable to read and use is “engine_running”.

4. DC- doors closed. Indicates whether all doors are closed or not. The corresponding integer is “doors_closed”.

5. KIC – key in car. Indicates that the keys are still inside the key hole, the corresponding integer variable is “key_in_car”.

6. DLC – door lock lever. This indicates whether the door lock lever is closed or not. To close the electronic door locks the driver has to close this door lock lever. When the car’s computer finds that door lock lever is closed it checks all other variables to assess the situation and finally decides whether to activate the electronic door locks to lock the doors or not. For example if the car keys are still inside but the driver is not on seat (has gone out of car) then the doors should never be locked even though the driver has closed the door lock lever.

7. BP – brake pedal. This indicates that the brake pedal is pressed by the driver.

8. CM – car moving. This sensor indicates the car is moving and at least one of its wheels is turning. The corresponding integer variable “car_moving” has value 1 when the car is moving, and has value 0 if the car is not moving.

Available actuators -

1. BELL - A beeper/chime that sounds/plays to alert the driver of any abnormal/hazardous situation. A global integer variable named “bell” is provided in the computer, if your code that implements the decision/control sub-system sets this variable to 1, then the output interface sub-system will read this value and turn on the voltage on the electrical wire that feeds the beeper/chime. As a result, the beeper/chime will start beeping. The beeper will stop when the decision/control sub-system code sets the value of “bell” to 0, because then the o/p i/f will turn down the electrical voltage feeding the beeper hence it will stop. The o/p i/f keeps on monitoring the integer variable and takes action (change the voltage) when the integer changes its value.

2. DLA – door lock actuator. This actuator locks the doors. A corresponding global integer variable named “door_lock” is provided in the computer, if you set this integer to 1 all the doors are locked, it unlocks all doors when you set the value of “door_lock” to 0.

3. BA – brake actuator. This actuator will actually activate the disk brakes in each of the four wheels if the global integer variable “brake” is set to 1. The brake will be released when this variable is set to 0 by the code that implements the decision/control logic sub-system.

Requirements -

1. The BELL should chime/sound when the driver starts the engine without fastening his seatbelt.

2. The BELL should sound when the driver starts the car without closing all the doors.

3. The BELL should be off as soon as the conditions change to normal.

4. The doors should not lock when the driver has got out of the car but the keys are still inside the engine, even though the driver has closed the door lock lever. Note: If the driver is on the seat and requests the doors to be locked, the doors must lock.

5. The brake should be engaged when the driver presses the brake pedal. Brakes should disengage when the brake pedal is released. The brake should engage only when the car is moving, when the car is stationary the brake should not unnecessarily engage to reduce mechanical wear and tear of the brake's hydraulic system.

Activities to do -

a) For each requirement, separately provide the Boolean expressions that you decided to use.

b) Create a single combined truth table with all the available sensor inputs and actuator outputs for all the five requirements together. Some of the truth table entries will be don't-care states (represented by an X) instead of true or false. For this sub-problem assume that these five requirements together constitute a complete system.

c) Write a C program using the Boolean logic concepts and tools learnt in class to materialize these five requirements. Use if-then-else structure to do this. Submit the C code and your executable. A basic code framework (file name "lab5.c") for a general control system has been provided. This is available in the lab files that you downloaded. Use this code to learn how to develop C program for a real-life control system.

3. Lab Device or Environment

Ubuntu 16.04 (64-bit) with AMD Ryzen 9 5900HS CPU @ 3.30GHz and 4GB memory on virtual machine (Oracle VM VirtualBox)

4. Results and Analysis

Analysis:

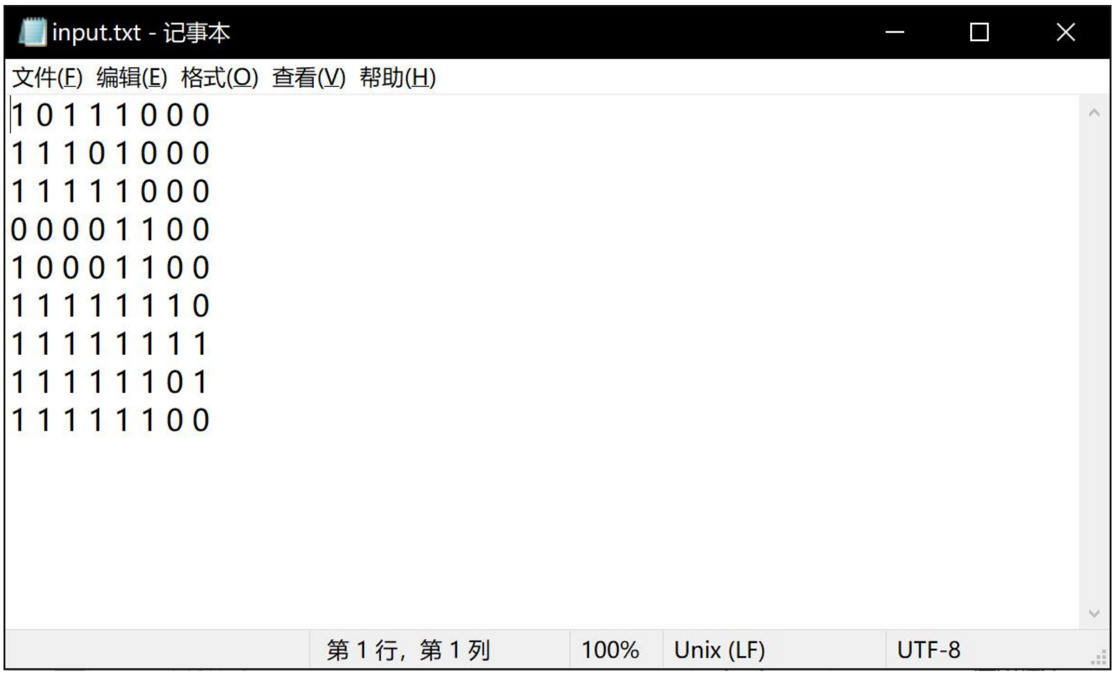
According to requirements, we create a single combined truth table with all the available sensor inputs and actuator outputs for all the five requirements together. Some of the truth table entries will be **don't-care** states (represented by an **X**) instead of **true** or **false**.

DOS	**DSBF**	**ER**	**DC**	**KIC**	**DLC**	**BP**	**CM**	**BELL**	**DLA**	**BA**
	false	true	X					true		

	X	true	false					true		
	true	true	true					false		
false				true	true				false	
true				true	true				true	
						true	true			true
						true	false			false
						false	X			false

Therefore, for requirements 1, 2, and 3, the Boolean expression is BELL = ER && !(DSBF && DC); for requirement 4, the Boolean expression is DLA = DOS && KIC && DLC; for requirement 5, the Boolean expression is BA = BP && CM.

Test Input (Use file input):



```
1 0 1 1 1 0 0 0
1 1 1 0 1 0 0 0
1 1 1 1 1 0 0 0
0 0 0 0 1 1 0 0
1 0 0 0 1 1 0 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 1
1 1 1 1 1 1 0 0
```

1 0 1 1 1 0 0 0

1 1 1 0 1 0 0 0

1 1 1 1 1 0 0 0

0 0 0 0 1 1 0 0

1 0 0 0 1 1 0 0

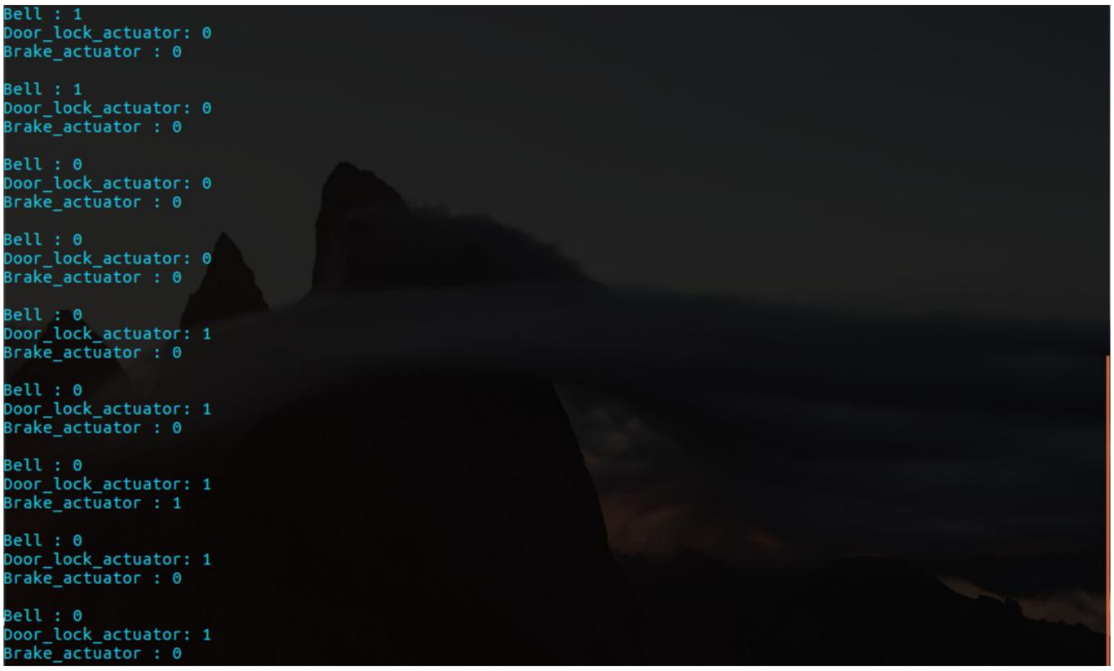
1 1 1 1 1 1 1 0

1 1 1 1 1 1 1 1

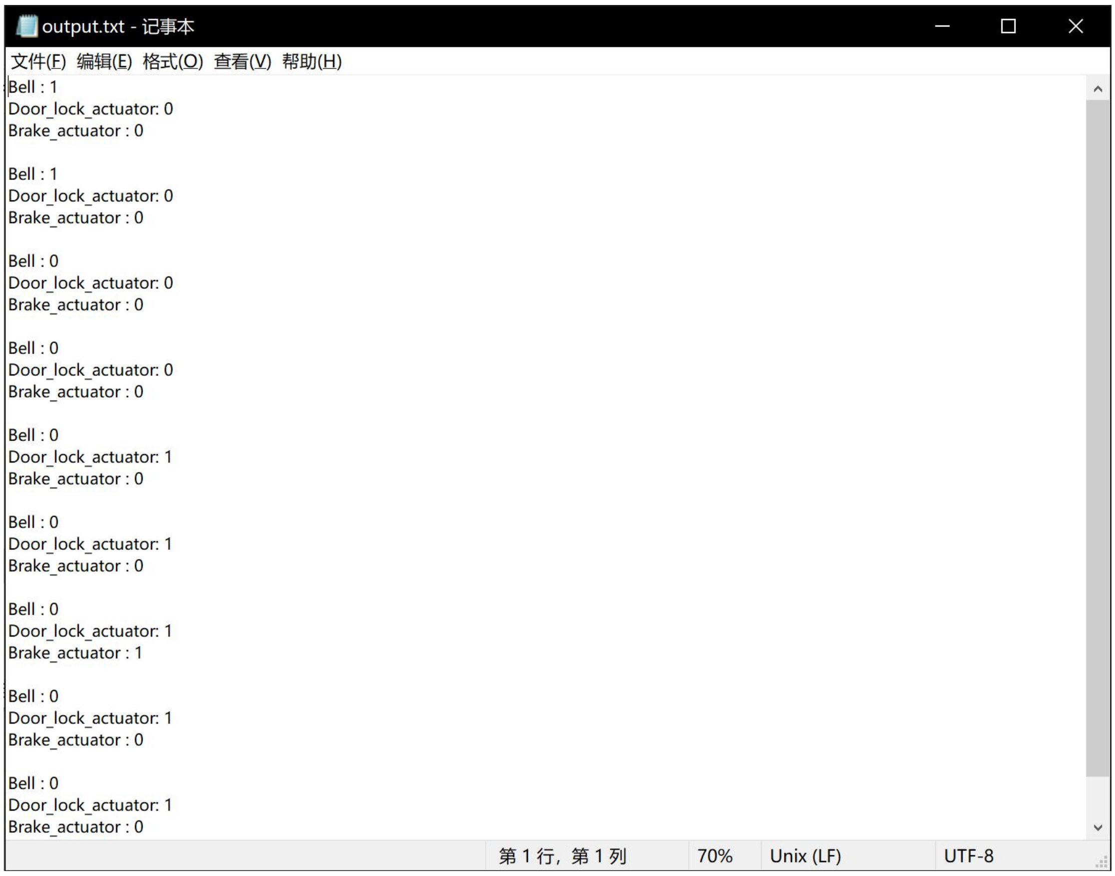
1 1 1 1 1 1 0 1

1 1 1 1 1 1 0 0

Results(Use both file output and standard output):



```
Bell : 1
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 1
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 1

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0
```



```
Bell : 1
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 1
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 0
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 1

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0

Bell : 0
Door_lock_actuator: 1
Brake_actuator : 0
```

5. Appendix (Program Code)

```
/*
This program sounds the bell when driver is on seat AND haven't closed the
doors. Use the general framework and the function shells, replace the code
inside the control_action() function with your own code.

Note: This code is designed to run in an infinite loop to mimic a real control
system. If you prefer to read the inputs from a file, modify the code
appropriately to terminate the loop when all the inputs have been processed.

run this file as : gcc filename.c -o executableName

*/

#include <stdio.h>   //This is useful to do i/o to test the code
#include <stdlib.h>

```

```c
unsigned int driver_on_seat;
unsigned int driver_seat_belt_fastened;
unsigned int engine_running;
unsigned int doors_closed;
unsigned int key_in_car;
unsigned int door_lock_lever;
unsigned int brake_pedal;
unsigned int car_moving;

unsigned int bell = 0;
unsigned int door_lock_actu = 0;
unsigned int brake_actu = 0;

FILE *infile = NULL;
FILE *outfile = NULL;
int end_of_file = 0;

void read_inputs_from_ip_if() {
    // place your input code here
    // to read the current state of the available sensors
    if (infile == NULL) {
        infile = fopen("./input.txt", "r");
    }

    if (fscanf(infile, "%u", &driver_on_seat) == EOF) {
        fclose(infile);
        end_of_file = 1;
        return;
    }

    fscanf(infile, "%u", &driver_seat_belt_fastened);

    fscanf(infile, "%u", &engine_running);

    fscanf(infile, "%u", &doors_closed);

    fscanf(infile, "%u", &key_in_car);

    fscanf(infile, "%u", &door_lock_lever);

    fscanf(infile, "%u", &brake_pedal);

    fscanf(infile, "%u", &car_moving);

    return;
}

void write_output_to_op_if() {
    // place your output code here
    // to display/print the state of the 3 actuators (DLA/BELL/BA)

    if (end_of_file) {
        fclose(outfile);
        exit(0);
    }

    if (outfile == NULL) {
        outfile = fopen("./output.txt", "w");
    }

    fprintf(outfile, "Bell : %d\n", bell);
    printf("Bell : %d\n", bell);

    fprintf(outfile, "Door_lock_actuator: %d\n", door_lock_actu);
    printf("Door_lock_actuator: %d\n", door_lock_actu);

    fprintf(outfile, "Brake_actuator : %d\n\n", brake_actu);
    printf("Brake_actuator : %d\n\n", brake_actu);

    return;
}

// The code segment which implements the decision logic
void control_action() {
    /*
    The code given here sounds the bell when driver is on seat
    AND hasn't closed the doors. (Requirement-2)
    Replace this code segment with your own code.
    */

    // Bell Actuator
    if (engine_running && !(doors_closed && driver_seat_belt_fastene
d)) {
        bell = 1;
        } else {
            bell = 0;
        }

        // Door lock Actuator
        if (driver_on_seat && key_in_car && door_lock_lever) {
            door_lock_actu = 1;
        } else {
            door_lock_actu = 0;
        }

        // Braking Actuator
        if (brake_pedal && car_moving) {
            brake_actu = 1;
        } else {
            brake_actu = 0;
        }

        return;
    }

    /* ---      You should not have to modify anything below this line ---------*/

    int main(int argc, char *argv[]) {
        /*The main control loop which keeps the system alive and responsive for
        ever, until the system is powered off */
        for (;;) {
            read_inputs_from_ip_if();
            control_action();
            write_output_to_op_if();
        }

        return 0;
    }
```