Assessment 1

This is the first assessment for SYS102: Basic Computer Architecture (Fall 2024). The assessment covers Numeral Systems and Number Representation, Digital Circuit Problem-Solving, and Interrupts and Polling. The total score before weighting is 200, distributed as follows: 25 for Numeral Systems and Number Representation, 50 for Digital Circuit Problem-Solving, and 25 for Interrupts and Polling.

Numeral System and Number Representation

Instructions:

1. Use the digits from a random number generator under the range **minimum of 1234 and maximum of 9876** to generate unique answers. (Please capture the output from the number generator you choose)
2. Ensure you explain your steps clearly where necessary.
3. Submit your answers in a typed document.

Part 1: Converting Between Numeral Systems *(total 18 points)*

Use the randomly generated digits, please clearly shows the conversion step that covered in the first lecture:

a) Convert from **decimal** to **binary**.
b) Convert the same **decimal** number to **hexadecimal.**
c) Convert the same **decimal** number to **octal**.
d) Convert the resulting **binary** number (from *a)* **to decimal.**
e) Convert the resulting **binary** number (from *a)* **to octal.**
f) Convert the resulting **binary** number (from *a)* **to hexadecimal**

Part 2: Signed Number Representation (Two's Complement) *(total 7 points)*

Using the **randomly generated digit from Part 1.**

a) Treat the first two digits as a signed decimal number. Convert this number into its 8-bit two's complement representation. **(3 points)**
b) Now, convert the last two digits into their 8-bit two's complement form. Add this number to the two's complement result from (a). Check if an overflow occurs and explain why or why not. **(4 points)**

Digital Circuit Problem-Solving Assessment

Introduction

Please assume you are an enthusiastic maker who recently studied the topic of Basic Digital Circuits. Up to this point, you have learned about basic logic gates, truth tables, Boolean algebra, and derived gates. Now it's time to use your new knowledge to solve a real-world problem that has been bothering you.

Task

Instructions

- Think about what you've learned and identify a real-world problem that can be solved using basic digital circuits.
- Look around your environment or talk to your friends and family for inspiration.
- Requirements:
    - The total number of inputs and outputs combined must be at least five (e.g., 3 inputs and 2 outputs).
    - Your design must use at least 5 basic logic gates (AND, OR, NOT).
    - Provide a circuit diagram for your proposed solution.

Submission and Evaluation Criteria

1. Problem Description:
    - Describe the problem clearly (maximum 300 words). You may include a simple figure as a visual aid to help explain the problem.
2. Circuit Design and Optimization:
    - Clearly describe how you optimized your circuit.
        - You can use Boolean algebra or Karnaugh maps (K-maps) for optimization.
    - Provide a step-by-step explanation of your design process.

Evaluation Rubric

1. Problem Clarity (10 points)
    - The problem is described clearly and concisely.
    - The problem is relevant and can be solved with basic digital circuits.
    - The input and output mapping are clearly explained.

2. Circuit Optimization (20 points)
    - The circuit is optimized effectively using either Boolean algebra or K-maps.
    - The steps to optimize the circuit are easy to follow and well-explained.
3. Digital Circuit Design (15 points)
    - The final circuit diagram is correct and uses the required number of inputs, outputs, and gates.
    - The circuit works to solve the proposed problem.
4. Creativity (5 points)
    - The solution is creative and shows a good understanding of digital circuits.

Additional Tips

- **Visual Aid**: Use any diagramming tool or software (like CircuitVerse, Logisim) to draw your circuit and make it clear.
- **Optimization**: Try simplifying the circuit as much as possible without losing functionality. This will show your mastery of Boolean algebra and K-maps.
- **Mocking Sensors/Actuators**: If your task requires sophisticated sensors or actuators (e.g., temperature sensors, motors), you may **mock** them in your design. This means you can **simulate** their behavior using simple inputs and outputs to represent the functionality. Please make sure to clearly describe them.
- **Creativity Encouragement:** Please avoid doing the bare minimum—this limits your creativity! Explore different possibilities, use additional inputs and outputs if needed, and challenge yourself to come up with an interesting, innovative solution.

Introduction to Interrupt and Polling in C Programming

Objective

Implement a C program demonstrating both polling and interrupt-driven input handling. You will compare the two approaches in terms of performance and provide a report based on your results.

Instructions

1. **Polling Mode**
    a. Write a C program that continuously checks (polls) the state of an input (e.g., button press or keypress).
    b. When the button is pressed, toggle a flag (e.g., 0 to 1 or "ON" to "OFF"). Print the flag state after every toggle.
2. **Interrupt Mode:**
    a. Modify the same program to use interrupts. Use an Interrupt Service Routine (ISR) to toggle the flag when the button is pressed.
    b. The ISR should toggle the flag without the program continuously checking the input.
3. **Performance Comparison:**
    a. Discuss the differences between polling and interrupts in terms of CPU usage and efficiency. Which method consumes more CPU resources and why?
    b. Include observations from your program's execution, such as how fast the flag toggles in each mode, and how this impacts system performance.

Submission Criteria

Report Format

Your submission must be in the form of a PDF report.

The report must include the following sections:

1. **Code Implementation**: Include your C code for both the polling and interrupt modes. Explain how the code works, especially focusing on the ISR in the interrupt mode.
2. **Performance Comparison**: Discuss your observations regarding the performance of polling vs interrupts. Which method is better for handling input in terms of CPU usage and responsiveness? Include an analysis of CPU time.

3. **Conclusion**: Summarize your findings. Which approach would you recommend for real-world applications (e.g. when to use polling/interrupts and why)?

Code Execution

- The program must toggle the flag at least 5 times in both polling and interrupt modes.
- Simulated input using a keyboard is acceptable.
- Include comments in your code explaining important sections.

Submission

Submit your report as a PDF. You can include screenshots of your code execution or print outputs to demonstrate functionality.

Evaluation Rubric (Total: 25 Points)

Criteria	Points	Description
Code Implementation (Polling)	5	- Correct polling logic is implemented. - The flag toggles correctly based on the button press or keypress. - The program includes appropriate comments.
Code Implementation (Interrupts)	5	- Correct use of interrupts and ISR is implemented. - The flag toggles correctly via interrupts without continuous polling. - The program includes appropriate comments.
Performance Comparison	10	- Clear discussion of performance differences between polling and interrupts. - Detailed comparison of CPU usage and responsiveness for each method. - CPU time is measured and compared for both polling and interrupt methods. - Accurate discussion of the CPU time measurements and analysis of efficiency.
Report Quality	5	- Well-organized report with clear sections. - Clear explanations and observations.