CSCI 6461 Computer Architecture - Project Part 3
Design Notes and User Guide: I/O Operations

Submitted By: *Sribalaji Annamalai Senthilkumar, Abenezer Golda, Razia Noor, Christabell Rego*

1. Overview

Part 3 of the project focuses on implementing robust Input/Output (I/O) operations and error handling mechanisms. The simulator has been extended to support:

1. **Advanced I/O Instructions:** Implementation of the CHK (Check Device Status) instruction to enable polling-based I/O.
2. **Software Interrupts:** Implementation of the TRAP instruction to handle system calls and exceptions.
3. **Machine Faults:** Automatic triggering of traps for illegal operations (e.g., illegal opcode, invalid memory access).
4. **"Program 2" Execution:** A new test program that reads a paragraph from a file (simulated card reader), prints it to the console, and searches for a user-specified character within that text.

2. Design Justification

Core Component: Assembler.java Updates

The assembler was updated to support the new instructions required for advanced control flow and I/O.

- **New Opcode Support:** Added support for CHK (Opcode 63) and TRAP (Opcode 30).
- **Alias Support:** Implemented BEQ (Branch if Equal) as an alias for JZ (Jump if Zero), simplifying the assembly code readability for loops and comparisons.
- **Robust Parsing:** Improved the parsing logic to handle simplified 2-operand syntax for jump instructions (e.g., BEQ R, Label), making the assembler more flexible and user-friendly.

Core Component: CPU.java Updates

The CPU logic was expanded to handle device status checks and exception processing.

- **CHK Instruction:** Implemented logic to check the status of I/O devices.
  - Device 0 (Keyboard): Always returns 1 (Ready) in this simulation model.
  - Device 1 (Printer): Always returns 1 (Ready).
  - Device 2 (File Reader): Returns 1 (Ready) if the file buffer has data, or 0 if empty.

- **TRAP Instruction:** Implemented a mechanism to save the current PC to memory location 2 and jump to the trap vector at memory location 0 + trap code. This simulates a basic software interrupt.
- **Machine Faults:** Added automatic fault detection. If the CPU attempts to execute an undefined opcode or access invalid memory, it triggers a fault. The fault code is stored in the MFR (Machine Fault Register), and the machine halts (or traps, depending on configuration).

Core Component: SimulatorGUI.java Updates

The GUI was enhanced to support the specific requirements of "Program 2".

- **File Input (Device 2):** Added a "Load Paragraph" button that reads a text file (paragraph.txt) into a dedicated buffer (fileInputBuffer). This simulates a card reader or tape drive.
- **Keyboard Input (Device 0):** Refined the keyboard input logic to accept single characters for the search function.
- **Printer Output (Device 1):** Enhanced the printer logic to correctly display the character stream from the executing program.

3. User Guide: How to Operate the Simulator

Step 1: Build the Application

To compile the source code and build the executable JARs, use the following commands in your project root:

```
# 1. Compile Source Code
javac -d out src/Assembler.java src/Cache.java src/CPU.java src/SimulatorGUI.java

# 2. Build Assembler JAR
jar cfe CS6461_Assembler.jar Assembler -C out .

# 3. Build Simulator JAR (Optional, can run class directly)
jar cfe CS6461_Part3.jar SimulatorGUI -C out .
```

Step 2: Assemble the Program

Use the assembler to convert the source code into machine code.

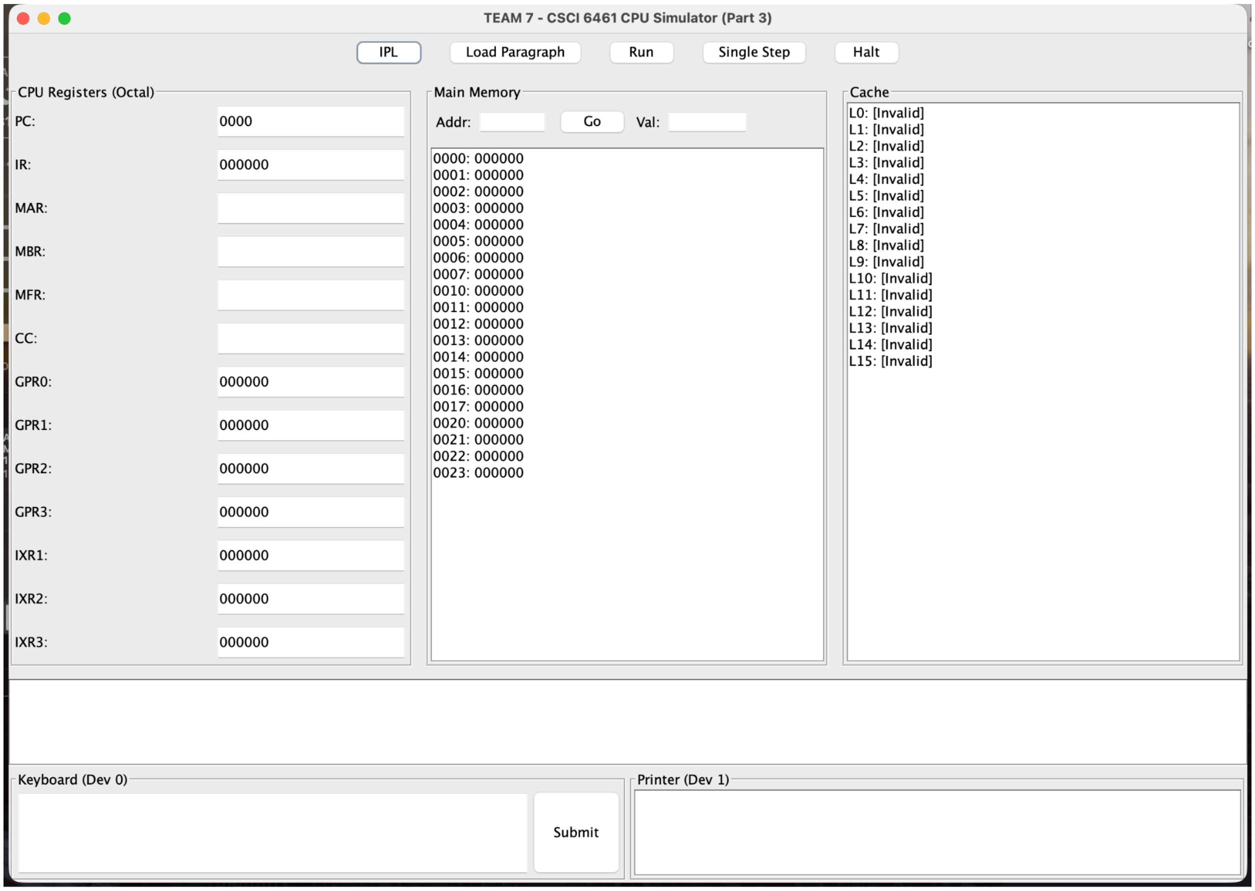
```
java -jar CS6461_Assembler.jar program2.txt
```

- **Input:** program2.txt
- **Output:** program2_load.txt

Step 3: Run the Simulator

Launch the simulator GUI.

```
java -cp out SimulatorGUI
```



PC: 0000

IR: 000000

MAR:

MBR:

MFR:

CC:

GPR0: 000000

GPR1: 000000

GPR2: 000000

GPR3: 000000

IXR1: 000000

IXR2: 000000

IXR3: 000000

```
0000: 000000
0001: 000000
0002: 000000
0003: 000000
0004: 000000
0005: 000000
0006: 000000
0007: 000000
0010: 000000
0011: 000000
0012: 000000
0013: 000000
0014: 000000
0015: 000000
0016: 000000
0017: 000000
0020: 000000
0021: 000000
0022: 000000
0023: 000000
```

```
L0: [Invalid]
L1: [Invalid]
L2: [Invalid]
L3: [Invalid]
L4: [Invalid]
L5: [Invalid]
L6: [Invalid]
L7: [Invalid]
L8: [Invalid]
L9: [Invalid]
L10: [Invalid]
L11: [Invalid]
L12: [Invalid]
L13: [Invalid]
L14: [Invalid]
L15: [Invalid]
```

Step 4: Execute Program 2 (Text Search)

1. **Load Data:** Click the **"Load Paragraph"** button and select the paragraph.txt file. This populates the simulated File Reader buffer.
2. **Load Program:** Click the **"IPL"** button and select the program2_load.txt file.
3. **Run:** Click the **"Run"** button.
   - The simulator will read the paragraph from Device 2 and print it to the **Printer** window.

   - Wait for the printing to finish.
4. **Search:** The console will indicate it is waiting for input. Type a single character (e.g., x) into the **Keyboard** field and click **Submit**.
5. **View Result:**
   - If the character is found in the paragraph, the printer will display F.
   - If the character is not found, the printer will display N.
   - The program will then Halt.

4. Test Case: program2.txt

This program demonstrates the capability to read from a file device, perform a linear search on the data, and interact with the user via keyboard and printer.

Source Code (program2.txt):

```
; CSCI 6461 - Program 2 (Text Search)
    LOC 100
START:
    ; 1. Read Paragraph from Dev 2 (File) loop
    LDX 1,BUFFER_START  ; X1 points to buffer
READ_LOOP:
    CHK 0,2             ; Check File Reader status
    BEQ 0,READ_LOOP     ; Loop if busy
    IN 0,2              ; Read char into R0
    JZ 0,READ_DONE      ; If 0 (null term), done
    STR 0,1,0           ; Store char
    OUT 0,1             ; Echo to printer
    AIR 1,1             ; Increment pointer
    JMA 0,READ_LOOP
READ_DONE:

    ; 2. Ask for Search Char
    IN 0,0              ; Read from Keyboard
    STR 0,0,SEARCH_CHAR

    ; 3. Search Loop
    LDX 1,BUFFER_START
SEARCH_LOOP:
    LDR 1,1,0           ; Load char from buffer
    JZ 1,NOT_FOUND      ; End of buffer
```

```
        SMR 1,SEARCH_CHAR   ; Compare with target
        JZ 1,FOUND
        AIR 1,1             ; Next char
        JMA 0,SEARCH_LOOP

FOUND:
        LDA 0,70            ; 'F'
        OUT 0,1
        HLT

NOT_FOUND:
        LDA 0,78            ; 'N'
        OUT 0,1
        HLT

        LOC 500
BUFFER_START: DATA 0
SEARCH_CHAR: DATA 0
```

Expected Outcome:

1. The contents of paragraph.txt are printed to the "Printer" window.
2. Upon entering a character that exists in the paragraph (e.g., 'e'), an 'F' is printed.
3. Upon entering a character that does not exist (e.g., 'z'), an 'N' is printed.