ICS Lab6

3200102555

Yunfan Li

The LC-3 Simulator is designed to execute LC-3 assembly instructions using an emulator-like approach. The solution involves several key components, including memory management, register handling, instruction execution, and condition code updates. The primary steps of the algorithm are as follows:

1. **Initialization (Init):** The `Init` function initializes the LC-3 simulator by setting all the registers and memory locations to a default value of `0x7777`. This function ensures a clean slate for the simulation.
2. **Load Instructions (Load):** The `Load` function reads input instructions from standard input, converts them to binary form, and stores them in memory starting from a specified memory address (head). The function reads assembly instructions line by line, converts them to binary, and stores them in memory sequentially. The head pointer keeps track of the starting address of the loaded instructions.
3. **Execute Instructions (execute):** The `execute` function is responsible for fetching, decoding, and executing LC-3 instructions. It begins execution at a specified memory address (`begin`) and iterates through the memory, one instruction at a time. The instruction is fetched, its opcode is decoded, and the corresponding LC-3 operation is executed based on the opcode.
4. **Instruction Execution Functions:** The solution defines separate functions for each LC-3 instruction, such as `br`, `add`, `ld`, `st`, `jsr`, `and`, `ldr`, `str`, `not`, `ldi`, `sti`, `jmp`, `lea`, and `halt`. Each of these functions implements the specific behavior of its corresponding instruction, including condition code updates and register/memory manipulations.
5. **Condition Code (cc) Updates:** The `setcc` function is responsible for updating the condition code (`cc`) based on the result of an operation. The `cc` reflects whether the result is positive, zero, or negative.
6. **Utility Functions:** Several utility functions are provided to assist in the execution process, such as `str2num` for converting binary strings to integers and `regnum` for extracting register numbers from instruction fields.

Main Code (with `SEXT` Explanation):

The main code consists of the `main` function, which orchestrates the entire LC-3 simulation. It first calls `Init` to initialize the simulator, followed by `Load` to load instructions into memory. Finally, it invokes the `execute` function, passing the starting address (`begin`) as an argument to begin instruction execution.

Explanation of `SEXT` Function:

The `SEXT` function plays a critical role in simulating the sign-extension (`SEXT`) element in LC-3. It is used to sign-extend immediate values in LC-3 instructions, ensuring that they are correctly represented as signed values in the simulator.

Here are a couple of examples illustrating how `SEXT` is used for different instruction types:

1. **Load (`LD`) Instruction:**
    - In the case of the `LD` instruction, `SEXT` is used to sign-extend the offset value, which is a 9-bit two's complement number. For example, if the offset is `0b111111111` (which represents -1 in decimal), `SEXT` ensures that it is correctly sign-extended to a 16-bit signed value of `0xFFFF` (which represents -1 in the simulator).

```
inline void ld()
{
    tmp1 = SEXT(ir, 9);
    tmp2 = regnum(ir, 11);
    tmp3 = pc + tmp1;
    regs[tmp2] = mem[tmp3];
    setcc();
    return;
}
```

2. **Add (`ADD`) Instruction:**
    - In the case of the `ADD` instruction with an immediate value, `SEXT` is used to sign-extend the immediate value before performing the addition. For example, if the immediate value is `0b11111` (which represents -1 in decimal), `SEXT` ensures that it is correctly sign-extended to a 16-bit signed value of `0xFFFF` (which represents -1 in the simulator), and then the addition is performed with the registers.

```
inline void add()
{
    if (ir & 0b100000)
    {
        tmp1 = SEXT(ir, 5);
        tmp2 = regnum(ir, 11);
        tmp3 = regnum(ir, 8);
        regs[tmp2] = regs[tmp3] + tmp1;
    }
    else
    {
        tmp1 = regnum(ir, 2);
        tmp2 = regnum(ir, 11);
        tmp3 = regnum(ir, 8);
        regs[tmp2] = regs[tmp3] + regs[tmp1];
    }
```

```
    setcc();
    return;
}
```

The `SEXT` function ensures that immediate values are correctly treated as signed values during instruction execution, maintaining the accuracy and integrity of the LC-3 simulation.

Difficult Part:

The most challenging part of implementing the LC-3 simulator was managing the program counter (`pc`) correctly. The program counter is a crucial element in instruction fetching and execution, and it must be incremented or modified in a precise order to ensure correct execution.

Initially, there was a mistake in managing the program counter's incrementation and autonomous changes. Debugging this issue was time-consuming because incorrect handling of the program counter could lead to fetching incorrect instructions, causing various execution errors.

Another challenging aspect was ensuring that the machine code for LC-3 instructions had the correct number of bits. Any discrepancy in the number of bits would lead to decoding errors and incorrect execution. Debugging and verifying the machine code generation required careful attention to detail.

In summary, the implementation of the LC-3 simulator involved careful attention to the program counter, accurate machine code generation, and detailed debugging to ensure correct execution of LC-3 assembly instructions. The `SEXT` function played a critical role in correctly sign-extending immediate values, ensuring the simulator's accuracy.