ARM Instruction Decoder and Simulator

This project implements a 32-bit ARM instruction decoder and a basic execution engine in Python. It can decode ARM instructions from a binary file, identify opcodes, operands, and flags, and simulate their execution, updating registers and memory as needed.

Features

- **Instruction Decoding:** Decodes 32-bit ARM instructions from binary, identifying opcodes, operands, and flags.
- **Instruction Types Supported:**
  - Data movement: `MOV`, `LDR`, `STR`
  - Arithmetic: `ADD`, `SUB`, `MUL`
  - Bitwise logic: `AND`, `ORR`
  - Comparison & conditional execution: `CMP`, `SUBNE`, `ADDEQ`
  - Shifts: `LSL`, `LSR`, `ASR`, `ROR`
- **Instruction Execution:** Simulates the execution of the supported instructions, updating registers and memory.
- **Condition Codes and Flags:** Evaluates condition codes and sets flags in the CPSR (Current Program Status Register).
- **Testing and Output:** Provides clear test case output showing the decoded instruction, modified register/memory values, and updated flags.

Files

- `arm_decoder.py`: Contains the `ARMInstruction` class for decoding ARM instructions.

- `arm_executor.py`: Contains the `ARMCpu` class for simulating ARM CPU execution.
- `main_simulator.py`: The main script to run the simulation, loading binary instructions and executing them.
- `generate_test_binary.py`: A utility script to generate a binary file with predefined ARM instruction encodings for testing.
- `simple_assembler.py`: A utility script to convert assembly instructions to a simplified binary format (for internal testing/development).

How to Use

1. **Generate Test Binary (Optional):** You can use `generate_test_binary.py` to create a `test_instructions.bin` file with a set of pre-defined instructions. `bash python3 generate_test_binary.py`

2. **Run the Simulator:** Execute the `main_simulator.py` script. It will read instructions from `test_instructions.bin` (or `instructions.bin` if `test_instructions.bin` is not present) and simulate their execution. `bash python3 main_simulator.py`

Output

For each instruction executed, the simulator will print:

- The decoded instruction (type, condition, set flags, and operands).
- The CPU state (register values and CPSR flags) after execution.

Limitations

- **Thumb Instructions:** Basic placeholder support for 16-bit Thumb instructions is included in `arm_decoder.py`, but the execution engine primarily focuses on 32-bit ARM instructions as per the initial requirements.
- **Memory:** A simple 1KB memory model is used.

- **Full ARM Instruction Set:** Only the 15 specified instruction types are fully implemented for decoding and execution.
- **Branching:** Branch instructions (`B`, `BL`) are decoded but their execution is currently limited to printing a message.

Development Notes

- The `arm_decoder.py` handles the parsing of instruction fields, including condition codes, opcodes, and various operand formats (immediate and shifted registers).
- The `arm_executor.py` manages the CPU state (registers, CPSR) and implements the logic for each supported instruction type.
- Conditional execution is handled based on the CPSR Z flag for `EQ` and `NE` conditions.