**Documentation**

1. **Overview**

This is a Java-based application that emulates a simple computer system. It consists of two main components:

1. A machine simulator that provides a graphical user interface (GUI) for users to interact with simulated computer components such as registers, memory, and control operations.
2. An assembler printer that processes assembly code and generates load and listing files for the machine simulator.

The simulator is designed to help users understand basic computer architecture, instruction execution, and memory management, while the assembler printer facilitates the translation of assembly code into machine-readable format.

**Usage Guidelines**

**1 Program Loading**

1. Run Main.java
2. Enter program file path or use file selector and select assemblyLoad.txt
3. Click IPL to load program
4. Verify successful loading via register displays

**2 Program Execution**

1. Use Step for single instruction execution
2. Use Run for continuous execution
3. Monitor register and memory states
4. Use Halt to stop execution

**3 Memory Operations**

1. Set MAR to target address
2. Use Load/Store for memory access
3. Use Load+/Store+ for sequential operations
4. Monitor MBR for data transfer

**2. Core Components**

**2.1 Main Classes**

* **Main.java**: Entry point of the application
  + Sets up the main JFrame window
  + Initializes the MainUI component
* **MainUI.java**: Primary user interface
  + Defines the entire user interface layout
  + Handles user interactions and button events
  + Communicates with ControlUI to execute operations

 **Assembler.java**: Assembly code processor

* Processes assembly source files into machine code
* Generates load and listing files for program execution
* Maps labels to memory locations
* Separates code from comments
* Key methods:
  + readInstructionFile(): Reads the source file
  + firstPass(): Maps out labels
  + secondPass(): Translates instructions to machine code
  + writeFiles(): Generates load and listing files
* **InstructionTranslator.java**: Instruction translation helper
* Handles the translation of individual instructions
* Manages label address mapping
* Key methods:
  + translateLDR(), translateSTR(), translateLDA(), etc.: Translate specific instructions
  + parseRegister(), parseIndexRegister(): Handle register parsing
  + encodeAddress(), parseAddress(): Manage address encoding and parsing

**2.2 Control and Processing**

* **ControlUI.java**: Backend controller
  + Manages communication between UI and CPU
  + Handles register operations
  + Implements button click events
  + Manages program loading and execution
  + Key methods:
    - loadGPRx(): Load values into general purpose registers
    - whenIPLClick(): Initial Program Load functionality
    - whenRunClicked(): Execute entire program
    - whenStepClicked(): Single instruction execution
    - whenHaltClicked(): Program halt handling
* **CPU.java**: Core processing unit
  + Implements instruction execution logic
  + Manages register operations
  + Handles memory access
  + Key features:
    - Instruction decoding and execution
    - Memory address calculation
    - Register value management
    - Error handling and fault detection

**2.3 Data Management**

* **CPUExecute.java**: Instruction execution package
  + Stores instruction execution details
  + Fields:
    - effectiveAddress: Calculated memory address
    - I: Indirect addressing flag
    - R: Register identifier
    - IX: Index register identifier
    - address: Base address
    - instructionString: Current instruction type
* **DataType.java**: Register definitions
  + Defines all register types and sizes
  + Includes:
    - General Purpose Registers (GPR0-GPR3): 16 bits
    - Index Registers (IXR1-IXR3): 16 bits
    - Special Registers (MAR, MBR, PC, etc.)
    - Control Registers (CC, MFR, HLT)

**2.4 Memory Management**

* **Memory.java**: Memory simulation
  + Implements main memory functionality
  + Manages memory read/write operations
  + Handles memory size constraints
  + Provides memory reset capabilities
* **Register.java**: Register implementation
  + Implements individual register functionality
  + Manages register value storage and retrieval
  + Handles register size constraints

**3. Instruction Set**

**3.1 Supported Instructions**

 Load/Store Operations:

* LDR: Load Register
* STR: Store Register
* LDA: Load Address
* LDX: Load Index Register
* STX: Store Index Register

 Control Flow:

* JZ: Jump if Zero

 System:

* HLT: Halt

**3.2 Instruction Format**

* 16-bit instruction word:
  + Bits 15-10: Operation Code
  + Bits 9-8: General Purpose Register (R)
  + Bits 7-6: Index Register (IX)
  + Bit 5: Indirect Addressing (I)
  + Bits 4-0: Address

**4. Memory Organization**

**4.1 Memory Structure**

* Size: 2048 words
* Word size: 16 bits
* Addressing: 12-bit addresses (0-2047)
* Protected locations: 0-5 (reserved)

**4.2 Memory Access**

* Direct addressing
* Indirect addressing
* Indexed addressing
* Combined indirect and indexed addressing

**5. Register Organization**

**5.1 General Purpose Registers**

* Four 16-bit registers (GPR0-GPR3)
* Used for arithmetic, logic, and data operations

**5.2 Index Registers**

* Three 16-bit registers (IXR1-IXR3)
* Used for address modification

**5.3 Special Purpose Registers**

* PC (Program Counter): 12 bits
* MAR (Memory Address Register): 12 bits
* MBR (Memory Buffer Register): 16 bits
* IR (Instruction Register): 16 bits
* CC (Condition Code): 4 bits
* MFR (Machine Fault Register): 4 bits
* HLT (Halt): 1 bit

**6. Control Flow**

**6.1 Program Execution**

1. Program loading via IPL
2. Instruction fetch from memory
3. Instruction decode
4. Effective address calculation
5. Instruction execution
6. Program counter update
7. Repeat until halt or error

**6.2 Error Handling**

* Memory fault detection
* Invalid instruction handling
* Protected memory access control
* Halt condition management

**7. User Interface Features**

**7.1 Display Elements**

* Register value displays
* Memory content viewer
* Program status indicators
* Binary input field

**7.2 Control Elements**

* Program loading controls
* Execution control buttons
* Register manipulation buttons
* Memory operation buttons

**8. File Organization**

**8.1 Source Files Structure**

**src/**

**├── Main.java**

**├── MainUI.java**

**├── ControlUI.java**

**├── CPU.java**

**├── CPUExecute.java**

**├── DataType.java**

**├── Memory.java**

**├── Register.java**

**├── Assembler.java**

**├── Input**

**└── InstructionTranslator.java**

**8.2 Dependencies**

* Java Swing for GUI
* Java AWT for event handling

Java BitSet for binary operations

Group No 10

Anket Vilasrao Patil

Pradnya Vinayak Tendolkar

Priya Ganta