```
#### RISCY - Assembler simulator

### Introduction
 RISCY is an assembly simulator. It attempts to mimic a load-store ISA (instruction set architecture).

### Instruction Set

 The instruction set is composed of 32 bits, broken into
* OpCode - First Octet of 4 bits
* Destination Register - Second Octet of 4 bits
* First Register - Third Octet of 4 bits
* Second Register - Fourth Octet of 4 bits
* UNUSED - Fifth Octet of 4 bits
* Offset - Sixth Octet of 8 bits

#### Opcode
  This is essentially the assembly command. This is represented as a map -

            {"LOADI", "0001"}, // LOADI R0, #10
            {"ADDRR", "0010"}, // ADDRR R1, R2, R3
            {"ADDRI", "0011"}, // ADDRI R1, #10
            {"BRNCH", "0100"}, // BRNCH <INSTRUCTION>
            {"EQUAL", "0101"}, // EQUAL R4, R5, R6
            {"NQUAL", "0110"}, // NQUAL R4, R5, R6
            {"CLOSE", "1111"}, // CLOSE

#### Destination Register
  This is location of the Destination Register. This is represented in a binary format for the location and in code is
represented by Registers interface.

#### First Register
This is location of the First Register. This is represented in a binary format for the location and in code is
represented by Registers interface.

```

```
35 #### Second Register
36 This is location of the Second Register. This is
   represented in a binary format for the location and
   in code is
37 represented by Registers interface.
38
39 #### Offset
40 This is the memory value of what we are trying to
   load.
41
42 ### Commands
43
44 #### LOADI
45 This instruction is used to load a register with a
   value
46
47     LOADI R0, #10
48     # This will load value 10 into register R0
49
50 #### ADDRI
51 This instruction is used to increment existing
   register with a value
52
53     ADDRI R0, #25
54     # This will increment value in R0 register by 25
55
56 #### ADDRR
57 This instruction is used to add values from two
   registers and save it in a 3rd register
58
59     ADDRR R1, R2, R3
60     # This will sum the values of R2 and R3, and load
    it into R1
61
62 #### EQUAL
63 This instruction is used to compare values between
   two registers, if equal save 1 in a 3rd register,
   else 0
64
65     EQUAL R4, R5, R6
66     # This will check the values of R5 and R6, if
```

```
equal will load 1 to R4 else will load 0 to R4

#### NQUAL
This instruction is used to compare values between two registers, if not equal save 1 in a 3rd register, else 0

    NQUAL R7, R5, R6
    # This will check the values of R5 and R6, if not equal will load 1 to R7, else will load 0 to R7

### Building instructions
 Requires only g++ to build. Command:

    cd <PROJECT_DIRECTORY>
    g++ --std=c++17 main.cpp \
        .\architecture\Architecture.cpp \
        .\architecture\Architecture.h \
        .\architecture\Executor.cpp \
        .\architecture\Executor.h \
        .\architecture\Parser.cpp
        .\architecture\Parser.h
        .\architecture\Registers.cpp
        .\architecture\Registers.h
        .\architecture\Stack.cpp
        .\architecture\Stack.h
        -o runRiscy

### Codebase

This section will document the underlying C++ classes.

#### ArchitectureWrapper

Description: Represents the architecture of a system, encapsulating registers and a stack.

##### Attributes:
- `ArchName` (`std::string`): The name of the architecture.
```

```
- `ArchRegisters` (`std::vector<RegisterWrapper>`): A vector that stores the architecture's registers.
- `ArchStack` (`StackWrapper`): A stack that holds the architecture's data.

##### Methods:
- Constructors:
    - Default Constructor: Initializes an empty architecture.
    - Parameterized Constructor: Accepts a string representing the architecture name.

- Other Methods:
    - `createStack()`: Initializes the stack for the architecture.
    - `createRegisters()`: Initializes the architecture's registers (inferred from `Architecture.cpp` but not fully visible).

#### Executor

Description: Represents an executor that manipulates the architecture.

##### Attributes:
- `simpleRISC` (`ArchitectureWrapper`): The architecture this executor works on.

##### Methods:
- Constructors:
    - Architecture Name-based Constructor: Accepts a string representing the architecture name and initializes an `ArchitectureWrapper`.

- Other Methods:
    - `getExecutorArchitectureName()`: Returns the name of the architecture.
    - `getExecutorRegisters()`: Returns a vector of registers.
    - `setExecutorRegister(std::string registerName, int value)`: Sets the value of a specific register
```

```
.

#### RegisterWrapper

Description: Represents a single register.

##### Attributes:
- `registerName` (`std::string`): Name of the register.
- `registerValue` (`int`): Value stored in the register.

##### Methods:
- Constructors:
    - Default Constructor: Initializes an empty register.
    - Name-based Constructor: Accepts a string representing the register's name.
    - Name and Value-based Constructor: Accepts a string for the register's name and an integer for the value.

- Other Methods:
    - `getRegisterName()`: Returns the name of the register.
    - `getRegisterValue()`: Returns the value stored in the register.

#### StackWrapper

Description: Represents a stack, handling the storage of data.

##### Attributes
- `stackPointer` (`int`): Points to the current position in the stack.
- `stackSize` (`int`): Size of the stack.
- `StackImpl` (`std::stack<int>`): Implementation of the stack using STL.

##### Methods:
```

```
157 - Constructors:
158     - Default Constructor: Initializes an empty
    stack.
159     - Size-based Constructor: Accepts an integer
    representing the stack's size.
160
161 - Other Methods:
162     - `getStack()`: Returns the actual stack
    implementation.
163     - `getStackSize()`: Returns the stack size.
164
165 #### Parser
166
167 Description: The `Parser` class parses instructions
    into different formats, such as binary and assembly.
168
169 ##### Attributes:
170 - `origInstruction` (`std::string`): The original
    instruction in its raw format.
171 - `tokenInstruction` (`std::vector<std::string>`): A
     vector of tokens representing the instruction.
172 - `rawBinaryInstruction`** (`std::string`): The
    binary representation of the instruction.
173 - `binaryInstruction` (`binaryPrep`): A struct
    containing the parsed instruction in binary format.
174 - `assemblyInstruction` (`assemblyPrep`): A struct
    representing the parsed instruction in assembly
    format.
175
176 ##### Methods:
177 - Constructors:
178     - Default Constructor: Initializes an empty `
    Parser` object.
179     - Parameterized Constructor**: Accepts a string
    representing the entire instruction.
180
181 - Getters and Setters:
182     - `getOrigInstruction()`: Returns the original
    instruction in its raw format.
183     - `tokenize()`**: Tokenizes the original
    instruction.
```

```
184     - `getTokenInstructions()`: Returns the
    tokenized instructions.
185     - `tokenToBinary()`: Converts the tokens into
    binary.
186     - `getBinaryInstructions()`: Returns the parsed
    instruction in binary format as a `binaryPrep`
    struct.
187     - `getRawBinaryInstructions()`: Returns the raw
    binary instruction as a string.
188     - `getAssemblyInstructions()`: Returns the
    parsed instruction in assembly format as an `
    assemblyPrep` struct.
189
190
191
```