**COL216: Assignment – 3**

Abhinav Jain, 2019CS10322

Himanshu Gaurav Singh, 2019CS10358

**An interpreter for a subset of MIPS assembly language instructions**

1. **Program specification**
2. Input Format:

A .txt file supposedly containing a sequence of MIPS commands.

1. Output Format:

The contents of 32-bit integer register after every command execution in signed hexadecimal format. Followed by the number of clock cycles in the whole program and the number of times a particular kind of instruction is executed

1. Assumptions:
2. It is assumed that only a subset of the MIPS instructions(as mentioned in the assignment specifications are used)
3. It is assumed that each instruction takes one clock cycle.
4. The registers are named $zero, $r1 , $r2 , …. $r31 and the $zero register is un-mutable.
5. Each instruction is assumed to be of size 32 bits.
6. We are limited to a memory of 2^20 bytes. Since we are working with signed integers, we have assumed memory to be a sequence of 2^18 spaces. With one space per instruction/integer(32 bits/4 bytes each).
7. The maximum instruction storage is 2^16 and remaining is reserved for data.

**2 . Approach**

An attempt of emulating the interpreting methodology of MIPS as closely as possible is made.

**Memory**

Memory is stored as a sequential array(std::vector in C++) of size 2^18(one for each 4 byte instruction/data). The information in Memory can be of type data or type instruction.

**Register File**

The contents of the registers are stored in a std::vector of size 32.

**Execution**

1. All the instructions in the text are read and stored in the memory sequentially.
2. Instruction memory overflow is checked during the instruction storage into memory.
3. Syntax errors are checked during reading and appropriate exceptions are raised.
4. Labels in the program are assigned the particular position in the instruction sequence they point to. Uniqueness of the labels is checked.
5. The instructions are executed in a sequential manner. After each execution, the register file and Memory is updated. The contents of the register file are printed.
6. Data memory overflow are checked after each “sw” instruction.

**3. Testing Strategy**

Testing of the program has been done under the following paradigms :

1. **Boundary Testing :**
2. **Stress Testing :**
3. **Error Handling :**

The following errors have been handled in the code

1. Syntax error : Unknown command, wrong arguments to commands, incorrect labels
2. Instruction memory overflow : Large number of instructions leading to memory overflow.
3. Data memory overflow : Large number of occupied memory by data causing overflow.
4. Zero register cannot be changed : To ensure un-mutability of the $zero register