Assignment-3 Submission

Indian Institute of Technology Delhi

COL216: Computer Architecture

Name: Deepanshu **Entry Number: 2019CS50427**
Name: Prashant Mishra **Entry Number: 2019CS50506**

1 Introduction

The assignment is about writing a C++ program that reads a MIPS assembly language program as input and interprets ("executes") it.

2 Approach taken to solve the problem

Following is the step wise description of the approach taken to solve the problem.

2.1 Choosing proper data structures

As mentioned in the problem statement, we need to look for appropriate data structures for maintaining processor components such as Register File and Memory. We carried out the hunt for data structures as follows:

1. **Array:** Array is used for representing memory as expected. In addition to memory, it is also used to store the values of register and also the frequency of each operation performed (to show statistics in the end).
2. **Unordered Map:** It is used to give an integer token to each of the 10 commands so that instructions can be stored in memory.
3. **Vector:** It is used to process single line of input and store the command entered and various parameters required for that command. This vector is then used along with unordered list to load the instruction in the memory array.

2.2 Taking major design decisions

We need to take many design decision along the way. Some of them are as follows:

1. We took the name of registers from $r0 to $r31 which is unlike what is used in QTSpim.

2. Defining the syntax. Out syntax takes one argument per line. Note that multiple consecutive commas and spaces between arguments will lead to an error.
   For example:
   Valid input: $add $t0,$t0,$t1
   Invalid input: $add $t0,,$t0, $t1

3. Using switch statement instead of nested if statements. As mentioned by Prof. Panda in the lecture, we should prefer using switch instead of nested if-else in cases when we have more number of comparisons.
   Hence, we use unordered map of C++ to give integer id to each operation and then use switch statement over these integer values.

4. We are giving 4 spaces in the array corresponding to each instruction. In case we have operations like j that take only one argument, we leave the other two values as zero so as to maintain the design principle of **simplicity favours regularity.**

2.3 Choosing various functions for the program

We also need to think about various functions which are required to run this program. Some of them include:

1. void printRegisters()
   This will be used after every clock cycle to print the content of the register in hexadecimal format as specified in the problem statement.

2. printStats()
   This will be used to give the total statistics in the end of the program to give various statistics like total number of times each command is executed and total clock cycles taken to execute the code. This is again as specified in the problem statement.

3. decToHexa()
   This will be to convert decimal number to hexadecimal number of the register values.

3 Testing Strategy

3.1 Testing the C++ code

In this , we took the C++ code and run against it for different input files(i.e test cases) which has MIPS commands written on it and finally matched MIPS output for same commands with C++ output.

3.2 Error Handling

Following are the possible errors that can occur while running the code:

1. **Syntax Error :** Following are the various possibilities of syntax error:
   (a) Instructions which are not defined.
   For example : addu $t1,$t2,$t3
   (b) Statement involving unknown characters:
   For example : add $t1 ; $t2 , $t3
   (c) Lesser number of registers used for operation than required:
   For example : add $t1 , $t2
   (d) More number of operators used for operation than required :
   For example : add $t1,$t2,$t3,$t4,$t5
2. **Memory Limited Exceed:** There is a possibility of memory limit exceed if total memory used for storing instruction and data exceed 2^{20} bytes .
3. **Segmentation Fault :** It may occur when we to excess the memory which is not available in the memory.
4. **Large offset used:** If the user tend to give the offset out of bound i.e larger than 2^{20} or less than 0. Even if it is not divisible by 4, we are flagging it as an error.

3.3 Choosing the test cases

All the test cases are provided in the zip file. We covered all the cases extensively ranging from error handling to looping. Some of the cases are as follows:

1. **Empty File :** Gives total commands as 0 and it takes 1 clock cycle to process the entire file.
2. **Basic Operation :** Following are some basic operations to separately test the functioning of the individual command correctly.
   (a) Binary Operation :
   For example : add $t1,$t2,$t3
   (b) Conditional jump :
   For example : beq $t1,$t2, label
   (c) Loading/Storing :
   For example : lw $t1, label

(d) Unconditional jump :
For example : j label

(e) Boolean Operation :
For example : slt $t1,$t2,$t3

3. Testing against various standard programs. Some of them are follows:

   (a) Finding factorial of a positive integer.

   (b) Given a positive integer n, find sum of first n positive integers.

   (c) Going in infinite loop.