COL216-Assignment-3 Report

Anil Kumar Uchadiya 2017CS10327

Akhilesh 2017CS10323

Approach Taken:

The Approach we have taken is a simple code structure based on which we first did the file reading of a text file which will have our Assembly instructions and then by using string operations we split every instruction (line) and simultaneously checked it for syntax error which can be caused by the wrong format which lack of space commas and number of argument needed for a particular command these all syntax check is done in the function split_inst with the use of an unordered map to check for the invalid instruction than after reading the whole file we stored in a vector of a struct which consists the instruction name and their argument in the vector of the string after storing instruction struct in the vector we start with PC=0 and keep execute each instruction by increasing the program counter(PC) and simultaneously handling each instruction in if else condition while updating the resister value which is used in the instruction and we also handled the branching instruction which is line no based by changing the PC according to the line number on which we want to branch and also handled jump statements after this we printed every register and there containing value in Hex after every instruction which is being executed we used “stol” which only parse the value with the limit of 2^32 -1 so value more than this will get negative or overflow. we finished the code by putting END CODE in last of every program.

TestCase:

We considered different types of test cases as following

test_0.asm

```
lw $4, $zero
addi $4, $4, 7
lw $6 ,$4
END CODE
```

value of register 4: hex=7, int=7

value of register 6: hex=7, int=7

Test_1.asm //sum of integer 1 to 10

```
lw $0, $zero
addi $0 ,$0 ,10
lw $1, $zero
lw $2 ,$zero
addi $0,  $0, 1
beq $0, $2 ,10
add $1, $1 ,$2
addi $2, $2 ,1
j 6
END CODE
```

value of register 0: hex=b, int=11

value of register 1: hex=37, int=55

value of register 2: hex=b, int=11

Test_2.asm //infinite loop

```
lw $1, $0
lw $2, $0
lw $3, $0
addi $1, $1 ,10
addi $2, $2 ,5
j 4
END CODE
```

test_3.asm

```
lw $1, $0
lw $2, $0
lw $3, $0
addi $1, $1 ,10
addi $2 ,$2 ,4
mul $3 ,$1 ,$2
END CODE
```

value of register 1: hex=a, int=10
value of register 2: hex=4, int=4
value of register 3: hex=28, int=40

test_4.asm

```
lw $0, $zero
lw $1, $zero
lw $2, $zero
addi $2, $2 ,10
addi $1 ,$1, 5
slt $3, $1, $2
bne $3, $0, 5
END CODE
```

value of register 0: hex=0, int=0
value of register 1: hex=a, int=10
value of register 2: hex=a, int=10

test_5.asm

```
lw $1, $zero
lw $2, $zero
lw $3, $zero
addi $1, $1, 10000000
addi $2, $2, 5
sub $3 ,$1, $2
END CODE
```

value of register 1: hex=989680, int=10000000

value of register 2: hex=5, int=5

value of register 3: hex=98967b, int=9999995

Test_6.asm \\\empty file and END CODE

```

```

```
END CODE
```

Error in first and no changes in second

test_7.asm

```
lw $1, $zero
lw $2, $zero
lw $3 ,$zero
addi $1, $1 ,10
addi $2 ,$2, 5
slt $3 ,$1, $2
bne $3, $0, 5
END CODE
```

value of register 0: hex=0, int=0

value of register 1: hex=a, int=10

value of register 2: hex=5, int=5

Test_8.asm \\ ignoring empty lines

```
lw $1, $0


lw $2, $0

```

```
lw $3, $0

addi $1, $1 ,10



addi $2, $2 ,5









j 38



j 25









END CODE
```

We are attaching some output of test cases we consider in the zip.

Errors Handling:

We handled syntax error and argument error. We ignored empty lines but consider them as empty instruction to facilitates jumping and branching instruction.