

# Report for Lab3

Wan Haochuan

## I. SCREENSHOTS

The screenshots of passing 3 test cases.

```
python ./tools/TestInst1.py -d /home/docker/workspace/ics/projects/lab3_wanhch/data/dumpdata_task1.txt
The Golden Result:
-----
100    -100    115    -85    -115    100    3    4
1      0      64      60
-----
The Dumped Result:
-----
100    -100    115    -85    -115    100    3    4
1      0      64      60
-----
Pass
-----
Done
```

Fig. 1. Result of task1.

```
python ./tools/TestInst2.py -d /home/docker/workspace/ics/projects/lab3_wanhch/data/dumpdata_01.txt -t /home/docker/workspace/ics/projects/lab3_wanhch/data/testdata_01.txt
The Golden Result:
-----
-41762  -37037  -9577  -17298  11901  4039  925  -21042
7466  -68633  -60124  52864  -3801  61301  41699  43319
-35271  47114  11540  -35298  -17562  -24796  29944  -22112
48011  75416  11646  -42454  50254  -69392  56761  -60997
13568  29868  81544  -18769  91362  -23116  80295  -72510
-181796  18796  48705  -63659  38246  8155  45413  -88770
13564  88052  6813  -137452  -64601  -87865  29345  101484
15297  109945  -23423  42425  26812  -73382  -34813  59986
-----
The Dumped Result:
-----
-41762  -37037  -9577  -17298  11901  4039  925  -21042
7466  -68633  -60124  52864  -3801  61301  41699  43319
-35271  47114  11540  -35298  -17562  -24796  29944  -22112
48011  75416  11646  -42454  50254  -69392  56761  -60997
13568  29868  81544  -18769  91362  -23116  80295  -72510
-181796  18796  48705  -63659  38246  8155  45413  -88770
13564  88052  6813  -137452  -64601  -87865  29345  101484
15297  109945  -23423  42425  26812  -73382  -34813  59986
-----
Pass
-----
Done
```

Fig. 2. Result of task2.

```
python ./tools/TestInst2.py -d /home/docker/workspace/ics/projects/lab3_wanhch/data/dumpdata_task3.txt
The Golden Result:
-----
8      9      17      26      13      7      100
-----
The Dumped Result:
-----
8      9      17      26      13      7      100
-----
Pass
-----
Done
```

Fig. 3. Result of task3.

```
python ./tools/TestInst2.py -d /home/docker/workspace/ics/projects/lab3_wanhch/data/dumpdata_01.txt -t /home/docker/workspace/ics/projects/lab3_wanhch/data/testdata_01.txt
The Golden Result:
-----
10563  62056  25892  55586  69520  109509  -45834  -7457
142059  -8443  -13862  -52838  97386  -34125  -126575  29808
80657  -20529  18181  -5386  -45489  9269  -16381  20716
-105140  116345  -68914  69863  85168  -59078  -121123  -8422
-40582  93674  -54302  88925  61131  13546  -23417  -98217
7197  -538  63668  20512  109881  -26485  -69315  36548
-801  -98138  104655  -12028  10514  48786  386575  -48182
56509  11191  9445  35538  109818  -56239  -14111  11805
-----
The Dumped Result:
-----
10563  62056  25892  55586  69520  109509  -45834  -7457
142059  -8443  -13862  -52838  97386  -34125  -126575  29808
80657  -20529  18181  -5386  -45489  9269  -16381  20716
-105140  116345  -68914  69863  85168  -59078  -121123  -8422
-40582  93674  -54302  88925  61131  13546  -23417  -98217
7197  -538  63668  20512  109881  -26485  -69315  36548
-801  -98138  104655  -12028  10514  48786  386575  -48182
56509  11191  9445  35538  109818  -56239  -14111  11805
-----
Pass
-----
Done
```

Fig. 4. Result of task4.

## II. IMPLEMENTATION DETAILS

### A. task1

Functions of each modules.

**inst\_decode**: decode instructions, read operands and give control signal to regfile and mem.

**inst\_fetch**: fetch instructions.

**execute**: execute different calculation by instructions.

**mem**: read and write mem and ram.

**regfile**: read and write registers.

**write\_back**: control whether write data to memory or registers.

Operation flow of each instructions.

**mul**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**add**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**and**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**sll**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**addi**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**slli**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**lw**: inst\_fetch, inst\_decode, regfile, execute, mem, write\_back.

**sw**: inst\_fetch, inst\_decode, regfile, execute, mem, write\_back.

**blt**: inst\_fetch, inst\_decode, regfile, execute.

**lui**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

**jal**: inst\_fetch, inst\_decode, regfile, execute, write\_back.

### B. task3

Functions of each modules.

**v\_inst\_decode**: decode instructions, read operands and give control signal to v\_regfile and v\_mem.

**v\_execute**: execute different calculation by instructions.

**v\_mem**: read and write mem and ram.

**v\_regfile**: read and write registers.

**v\_write\_back**: control whether write data to memory or registers.

Operation flow of each instructions.

**vle32.v**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_mem, v\_write\_back.

**vse32.v**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_mem, v\_write\_back.

**vadd.vv**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

**vadd.vi**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

**vadd.vx**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

**vmul.vv**: v\_inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

**vmul.vi:** inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

**vmul.vx:** inst\_fetch, v\_inst\_decode, v\_regfile, v\_execute, v\_write\_back.

### III. ASSEMBLY CODE DETAILS

#### A. task2

The calculation flow in task2:

Step1. Read the D and save the data to C's address one by one.

Step2. Read each element in the first row of A and first column of B and the same. Then multiply two elements together and add with the corresponding C's element. Update the corresponding C's result.

Step3. Repeat Step2 similarly but use the next column of B every time.

Step4. Repeat Step2-3 similarly but use the next row of A every time.

#### B. task4

The calculation flow in task4:

Step1. Read the D and save the data to CT's address column by column.

Step2. Read and multiply the first column of A and first element of B together. Then read and add the first column of C. Update the corresponding C's result.

Step3. Repeat Step2 similarly but use the next column of A and next element of B in the same column every time.

Step4. Repeat Step2-3 similarly but use the next column of B and C every time.

### IV. INTRODUCTION & COMPARISONS

Vector processor can only operate 1 or 2 operands at same time, while scalar processor can operate a batch of operands with the other batch of operands or one operand.

In software, scalar processor can use less instructions to execute more computation. But scalar processor need to consider the calculation flow to get the maximum efficiency, which makes scalar processor less flexible than vector processor.

In hardware, scalar need much more ALUs, registers and memory access than vector processor.

### V. CONCLUSIONS

In this lab, vector and scalar RISC-V processors with speechified instructions are implemented successfully. MAC operation is also completed successfully by these two kinds of processors.