Report for Lab3

Wan Haochuan

I. Screenshots

The screenshots of passing 3 test cases.

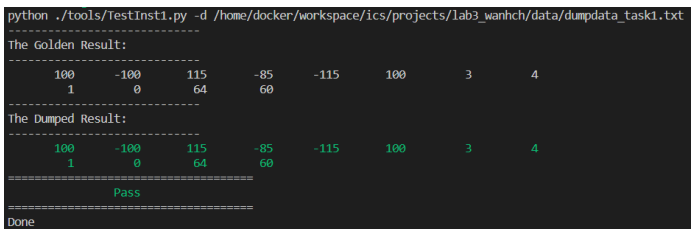

Fig. 1. Result of task1.

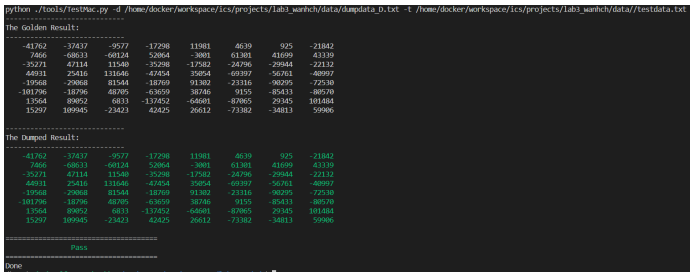

Fig. 2. Result of task2.

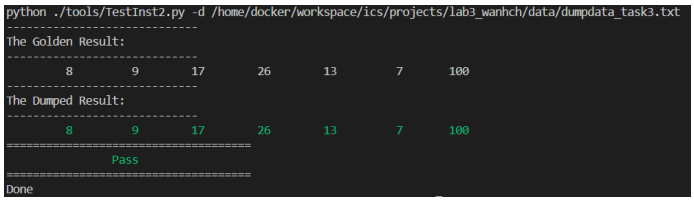

Fig. 3. Result of task3.

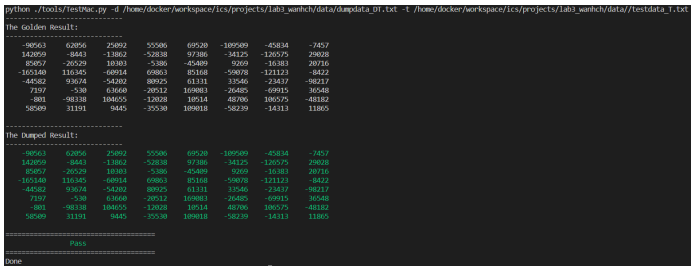

Fig. 4. Result of task4.

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.
slti: 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: inst_fetch, v_inst_decode, v_regfile, v_mem, v_write_back.
vse32.v: inst_fetch, v_inst_decode, v_regfile, v_mem, v_write_back.
vadd.vv: inst_fetch, v_inst_decode, v_regfile, v_execute, v_write_back.
vadd.vi: inst_fetch, v_inst_decode, v_regfile, v_execute, v_write_back.
vadd.vx: inst_fetch, v_inst_decode, v_regfile, v_execute, v_write_back.
vmul.vv: 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. Than 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.