Project1

Overview

In Project 1, our objective is to familiarize you with some fundamental concepts within RISC-V and enable you to write basic programs using RISC-V assembly language. In this assignment, your first task is to implement a simple ALU (Arithmetic Logic Unit) that produces correct outputs based on the provided test inputs. Following that, you are required to utilize RISC-V to implement dot product, such as vector ⊙ vector and matrix ⊙ vector.

Before you start working on this project, you need to:

- Review the content of RISC-V on blackboard

Requirements

1. ALU

In general, the ALU here is a rather simple Arithmetic Logic Unit. There is no need for instruction parsing. Instead, you are only required to implement various ALU functions corresponding to different instructions, such as add, sub and so on.

1.1 ALU Functions

You are required to support the following RISC-V instruction in your ALU.

- add, sub
- beq, bne, slt
- sll, srl, sra
- and, nor, or, xor

The inputs for functions (add, sub, and, nor, or, xor, slt) come in single forms: rs1, rs2, rd which specify the locations of registers of inputs and output. Then, the inputs of functions(beq, bne) are rs1, rs2. The inputs of

functions(sll, srl, sra) are rs1, shamt, rd, which shamt means the shift number.
For example,

```
int ADD(int rs1, int rs2, int rd)
        ......
        if overflow:
                return 1
        return 0

int BEQ(int rs1, int rs2)
        result = ...
        ......
        return result

int SLL(int rs1, int shamt, int rd)
        ......
        return 0
```

1.2 Function Example

```
AND:
        lui     a4,%hi(registers)
        addi    a3,a4,%lo(registers)
        slli    a0,a0,2
        slli    a1,a1,2
        slli    a2,a2,2
        add a0, a3, a0
        add a1, a3, a1
        add a2, a3, a2
        lw      a0,0(a0)
        lw      a1,0(a1)
        and     a1,a0,a1
        sw      a1,0(a2)
        li a0, 0
        jr      ra
```

- **Load the Base Address of the Registers:**
  - `lui a4, %hi(registers)`: Load the high 20 bits of the address of `registers` into `a4`.
  - `addi a3, a4, %lo(registers)`: Add the low 12 bits of the address of `registers` to `a4`, storing the complete address in `a3`. Now, `a3` contains the base address of `registers`.
- **Shift and Scale Indices (Assuming 4-Byte Elements):**
  - `slli a0, a0, 2`: Scale the value in `a0` (index for the first element) by 4 to account for 4-byte (32-bit) elements.
  - `slli a1, a1, 2`: Scale the value in `a1` (index for the second element) by 4.
  - `slli a2, a2, 2`: Scale the value in `a2` (index for storing the result) by 4.
- **Calculate Addresses of Elements:**
  - `add a0, a3, a0`: Add the scaled offset to the base address to get the actual address of the first element.
  - `add a1, a3, a1`: Calculate the address of the second element.
  - `add a2, a3, a2`: Calculate the address where the result will be stored.
- **Load and Perform the Bitwise AND Operation:**
  - `lw a0, 0(a0)`: Load the value of the first element into `a0`.
  - `lw a1, 0(a1)`: Load the value of the second element into `a1`.
  - `and a1, a0, a1`: Perform the bitwise AND operation between the two loaded values, storing the result in `a1`.
- **Store the Result:**
  - `sw a1, 0(a2)`: Store the result of the AND operation into the third element of the registers (location calculated earlier).
- **Return and End of Function:**
  - `li a0, 0`: Set the return value to 0 (optional, depends on calling convention).
  - `jr ra`: Return from the subroutine.

We define a0 to represent the value of the first input, a1 for the second input, and a2 to represent the value of the third input. You should do this in your program and we will check it randomly. If you don't follow the pattern, you will miss 50% points related to the alu task.

1.3 compile and run file

```
riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 -O3 alu.s -o
alu
./alu #if it does not work, use qemu-riscv32 alu
```

2.Dot Product

This assignment focuses on implementing two primary functions using RISC-V assembly language:

1. Vector Dot Product: Compute the dot product of two vectors.
2. Matrix-Vector Dot Product: Compute the dot product of a matrix and a vector.

2.1 Detailed Requirements

Vector Dot Product

The vector dot product function will take two input vectors, each of equal length, and compute their dot product. The dot product of two vectors $\mathbf{a}$ and $\mathbf{b}$ of length n is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Where a_i and b_i are the elements of vectors $\mathbf{a}$ and $\mathbf{b}$ respectively.

Matrix-Vector Dot Product

The matrix-vector dot product function computes the product of a matrix and a vector. For a matrix M of size $m \times n$ and a vector $\mathbf{v}$ of length n, the result is a new vector $\mathbf{r}$ of length m, where each element of $\mathbf{r}$ is the dot product of a row of the matrix M with the vector $\mathbf{v}$.

$r_i = \sum_{j=1}^{n} M_{ij} v_j$

Where M_{ij} is the element in the i-th row and j-th column of the matrix, and v_j is the j-th element of the vector.

2.2 Function Format

```
int vector_dot_product_vector(int* va, int* vb, int n) {
    result = ...
    return result
}

void matrix_dot_vector(int rows, int cols, int matrix[rows]
[cols], int vector[cols], int result[rows])
```

In this program, you should use a0 to represent the initial address of va, a1 for vb, a2 for n at first function. Besides, you should use a0 to represent rows, a1 for cols, a2 for the base address of matrix, a3 for vector, a4 for result at second function. Similar to the ALU program, failing to adhere to these specifications may result in a 50% point deduction.

2.3 compile and run file

```
riscv64-unknown-elf-gcc -march=rv32im -mabi=ilp32 -O3 dot_prod.s
-o dot
./dot #if it does not work, use qemu-riscv32 dot
```

Report

The report of this project should be no longer than 5 pages. Keep your words concise and clear.
In your report, you should include:

1. Your big picture thoughts and ideas, showing us you really understand RISC-V assembly.

2. The high level implementation ideas. i.e. how you break down the problem into small problems, etc.
3. The implementation details. i.e. explain some special tricks used.
4. the screenshots of result

Submission and Grading

1. Submission

You should put all of your source files (alu.s, dot_prod.s, report.pdf) in a folder and compress it in a zip. Name it with your student ID. Submit it through BB. The deadline for this report is 2024/09/30. If you fail to submit your assignment by the deadline, 10 points will be deducted from your original mark for each day you are late, up to a maximum of 30 points. Assignments submitted more than 3 days after the deadline will not be considered valid and will be marked as 0 points.

2. Grading Details

Implement the task about ALU - 50 %
Implement the task about Dot Product - 40%
Report - 10%

In grading, we have more tests about tasks. Please make sure the completeness of codes.

3. Honesty

We take your honesty seriously. If you are caught copying others' code, you will get an automatic 0 in this project. Please write your own code.