COL216

ASSIGNMENT 3

Anjali Sharma Hardeep Kaur

2019CS50422 2019CS10354

Approach:

AIM : Take a C++ program as input and interpret the instructions.

Internal data structures used:

1. ALU

2. Memory

3. Register File

For each instruction, the register file is used to read the values of operands from registers. ALU is used to perform the respective operation and the result is again written in the register file using register file.

Program counter increases by 1 after each instruction .
In case of jump statement, program counter points to the instruction referred by corresponding label.
The register values are initialised by reading from the “RF_init.txt”, where the 32 values are written in decimal form.

Memory consists of two seperate units :
1. Data Memory, which is initialised by reading from file “dataMem.txt”. The declarations in this file are of format:
<variable_name> <space> <value (in decimals)

Since this implementation has only lw and sw commands, we omitted the use of .word directive.

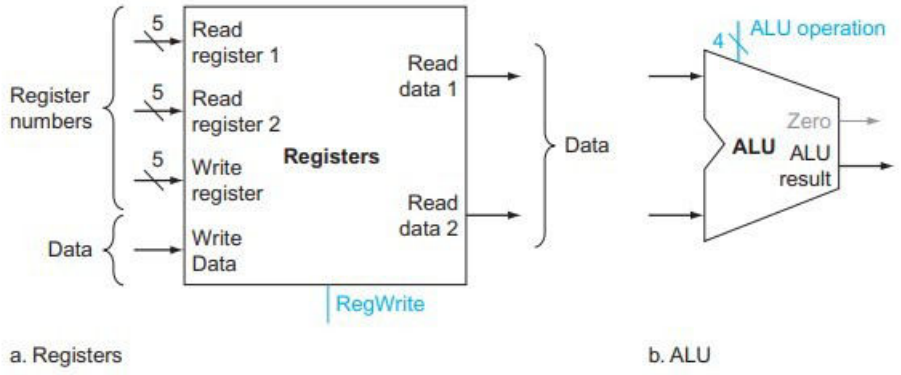
2. Instruction Memory
This is initialised by reading the instructions from the “input .txt” file. The instructions should follow MIPS format convention. The size of memory is 2^{20} bytes in total. Hence, a check is performed to ensure that the memory is in the defined limit (2^{20} bytes). Both the word declarations in data memory and instructions in the instruction memory take 4 bytes each. Hence, memory is word addressable.

Design:

Register File
This implementation has a RF class (register file), a Memory class and an ALU class.

RF class acts as a container for the 32, 32-bit registers.
The constructor of this class would initialise the register values according to the file “RF_init.txt” .
ReadWrite function facilitates the reading and writing of registers into the register file.
The OutputRF function writes the values of all the 32 registers after every instruction is executed .

Memory
ReadWrite function is to read or write in the memory. As in the figure below the ReadWrite function takes three 5 bit registers (2 read registers and one write register),a 32 bit write data and 1 bit write enable as input. The output is the operands type, which contains two 32 bit data (op1 and op2 here)



ALU
The function ALUOperation() takes the 2, 32 bit values (operand1 and operand2) and a 4 bit ALUOP(operation to be performed) as input. The output is of the type ALUresult (i.e. the values result and the Boolean is zero) which is consistent with the above diagram. In ALU class all the calculations are performed depending upon the type of instruction. Each operation is coded as an integer value and then passed as bitset<4> (stored in ALUoper here) . It returns an object of type ALUResult, which stores the calculation result, CalcResult and one bit data field, isZero which tells if the result of computation is zero or not.

Testing Strategy:

- This implementation executes all the instructions mentioned in statement.
- A check is also performed to ensure that input follows MIPS convention.
- In case of wrong input format, program is terminated with warnings printed on console.
- Following is one test case which is interpreted successfully by this implementation.

Data declarations in “dataMem.txt” :
value 8
num 2

Instructions in “input.txt” :

```
lw $t2, value
lw $t1, num
loop:
sub $t2, $t2, $t1
bne $t2, $zero, loop
slt $s1, $t2, $t1

sw $s1, num
```

OUTPUT :
The updated values in data memory are written to “dataMem_final.txt”.
The states of registers after each instruction are written to “RF_final.txt”.
Statements related to processing are printed on console.
Statistics involving the number of clock cycles and the number of times different instructions are executed, are also printed to console.