DESIGN DOCUMENT: RISCV-32I-IMPLEMENTATION

This document describes the design aspect of **RISCV** simulator created as a project, as part of Computer Architecture.

Contents:

Input:

Input to the simulator is a .mc file. A preliminary file is obtained from the venus simulator dump. It is then edited manually to also contain the address at which the instruction is to be stored.

The format of the .mc file is:
<location of memory in hex><space><instruction in hex>
For example:

```
0x0 0x100014b7
0x4 0x000482b3
0x8 0x00000333
```

Output:

The simulator gives 3 kinds of output:

1. **Message** after execution of each function (for every instruction).
2. **Clock cycle** number at the end of every clock cycle.
3. **Information** on terminating the simulator, at the end of the program.

Message after execution of each function:

The simulator's main functionality is divided into five functions, which are the same as discussed in class- ie, Fetch, Decode, Execute, Memory Access, and Register Writeback. For each instruction, the simulator prints a message at the end of each function.

For example:

Fetch prints-
FETCH: Fetch instruction 0x100014b7 from address 0x0

Decode prints-
DECODE: Decoded the instruction.

Execute prints-
EXECUTE: Operation is ADD, First Operand is R0, Second Operand is R0, Destination is R0.

Memory Access prints-
MEMORY: Load 1 Byte of Memory Value 12345 from address 20
MEMORY: Store 4 Bytes of Memory Value 12345 to address 24
MEMORY: No memory operation.

Register Writeback prints-
WRITEBACK: Write 11 to R5
WRITEBACK: No register writeback operation.

Clock cycle number at the end of each cycle:

Outputs a message of the format "Clock cycle <number> finished."

Information of ending the simulator, at end of program:

Tells the user that memory is feeded to the .mc file, and how many clock cycles the program took.

Design of Simulator:

Data structures:

1. C++ defined **map<uint32_t, uint8_t>** -> Used for Memory
2. **uint_32 array** -> Used for Register File
3. **int and long** -> used for Control Lines and ALU Variables
4. **Int array** -> Used for Instruction

Simulator flow:

There are four steps:

1. Memory is loaded
2. An instance of class RISCV is created to initialize the processor and its register.
3. Instructions are executed one at a time
4. Final memory is loaded into the .mc file. (Only included memory which has been used in the program to save compilation time).

Step 1: Step 1 opens the .mc file, checks for errors in the call of the program, and stores the value of N to R3. N is an integer required for all test programs. N can be user defined. (Default value is 5.)

Step 3: Step 3 is an infinite loop. It simulates all instructions till it reads the instruction “0xfffffff”. This is the termination code and the simulator ends.

Error Handling:

1. When a pc address value which does not have any valid instruction is fetched, the program fetches “0xffffffff” and terminates the program.
2. When an invalid op_code has been decoded from the instruction, the program runs the function: instruction_exit() which terminates the program.
3. If a memory location in which no value has been stored is accessed, the value at the memory location is assumed to be 0.

Test plan:

We test the simulator with the following assembly programs:
1. Fibonacci Program
2. Sum of the array of N elements. Initialize an array in the first loop with each element equal to its index. In the second loop find the sum of this array, and store the result at Arr[N].
3. Bubble Sort Program