32 Bit Risc Cpu

Karthika Madhavanpillai[#1], Vineela Velicheti[*2]

[#]*Computer Engineering Department, San Jose State University*
San Jose California

[1]karthika.madhavanpillaisasidharannair@sjsu.edu

[3]vineela.velicheti@sjsu.edu

***Abstract*— This document gives an overview of the construction of a 32-bit RISC CPU that executes fixed point instructions and floating pointing instructions by incorporating the forwarding paths to avoid data hazards.**

***Keywords*— RISC CPU, Fixed point instructions, Floating point instructions, data hazards.**

I. INTRODUCTION

This document manifests the various steps involved in the design and implementation of the 32- bit RISC CPU using Model sim. First, the individual modules for the implementation of the CPU have been developed and tested individually. After that we integrated all to form a top module. The design involves taking data inputs from the user and storing it in the register file and then executing the instruction. Each module involved in the development of the CPU has been explained in detail below.

II. MODULES

The individual modules involved in the construction of the CPU is listed below. These modules were integrated later to form the CPU.

a.) Program Counter
b.) Instruction Memory
c.) Instruction Register
d.) Opcode decoder
e.) Register File
f.) Fixed Point ALU
g.) Floating Point ALU for Add and Multiplication
h.) Data Memory
i.) Multiplexers
j.) Data Hazards Modules
k.) Top Module

a.) Program Counter :
Program counter outputs 32-bit address of the instruction memory. PC is implemented using D flip flop logic which gives output only on the positive edge of the clock. During reset, PC is set to FFFFFFFFFFFFF

b.) Instruction Memory :
A 32x32 bit instruction memory is designed. It contains the addresses of the instructions that is to be executed. The instructions are stored in RISC instruction format. The output of instruction memory is the 32-bit instruction

c.) Instruction Register :
The instruction register takes 32-bit instruction from the instruction memory and it splits the instruction in the opcode, source register addresses and destination register address and immediate values as per the instruction type and outputs them.

d.) Opcode Decoder :
Opcode decoder receives the 6 bit opcode input from instruction memory and depending on the opcodes, it gives the control signals to perform the write operations in register file and data memory.

e.) Register File :
Register file is designed as 32 x 32. It takes the 5-bit source register addresses and destination address as input from instruction register. It does the read operation for every source register address and performs the write operation whenever destination address and data to be written is inputted. Write operation happens at positive cycle and read operation takes place at negative edge of the clock cycle.

f.) Fixed Point ALU(Arithmetic Logic Unit) :
Fixed point ALU performs the arithmetic operations for all the 23 instructions which are provided in the requirement specification document. Arithmetic operations are carried out depending on the opcodes for each instruction.

g) Floating Point ALU:
Floating point ALU designed implements ADDF and MULF instructions which adds and multiplies two floating point numbers directly.

h) Data Memory:
Data memory designed has 32 rows and 32 columns. LOAD operation takes values stored in data memory and writes it in register file at the corresponding destination address while STORE operation reads the input from register file and stores it in data memory at the corresponding address.

i) Multiplexers:
We used a total of 5 multiplexers which provide the output depending on the opcode they receive in each clock cycle.

j) Data Hazard Modules:

Modules were created to handle data hazards by providing data forwarding paths. Source and destination addresses of instructions are compared of the corresponding cycles and control signals for alu-alu, data memory bypass- alu, data memory- alu hazards are given as outputs and depending on these control signals inputs to the ALU is selected from either the register file or ALU output/data memory output of previous instructions.

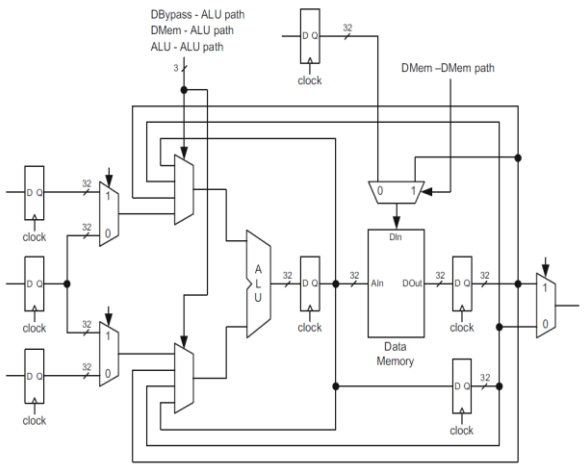

Fig : CPU schematic containing all Data Hazard conditions

III. Top Module

Design of top module involves, program counter incrementing for each instruction and outputting the 32- bit address of the instruction memory. Instruction memory stores the instructions to be executed and outputs the instruction depending on the address from the PC. This instruction is split to the corresponding opcodes and source and destination register address and instruction register and these values are taken as input in opcode decoder and register file respectively. The data stored in the register file is given as output whenever the 5-bit source addresses are inputted in to register file. These values and the immediate values if present is inputted in to the multiplexers and depending on the opcode values the inputs to ALU are selected. The arithmetic operations are performed in ALU depending on the opcodes and its either written in to register memory or provided as input address to the data memory. The CPU is designed such that an instruction competes its execution in 5 cycles. The corresponding stages are explained below:

1. First stage – Instruction Fetch
2. Second Stage – Register File stage (Output from register file is delayed for one cycle and fed in to ALU)
3. ALU stage – Output of ALU is delayed for one cycle and fed in to either data memory or bypassed.
4. Data Memory /Bypass stage – The output of Data Memory / the output from ALU bypassed is again delayed for one cycle and outputted via multiplexer depending on opcode
5. Write back Stage – Data is written to register file

IV. Part-2 Implementation

In Part-2 of this project, we have implemented an example that shows the use of forwarding paths to avoid data and control hazards on an instruction chart. The use of NOP instruction is also shown in various situations.

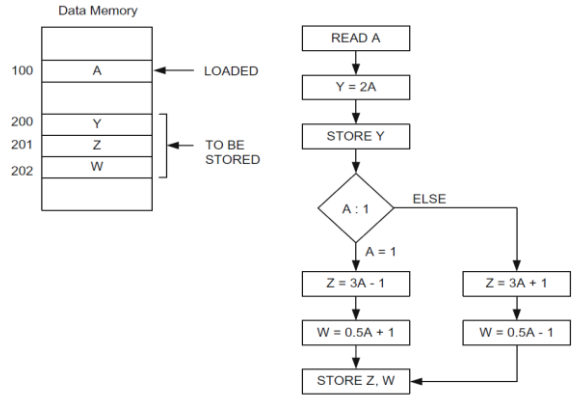

Fig: Flow-Chart of an example program

V. Floating Point Instructions

The RISC instruction set contains 2 Floating Point instructions: ADDF, MULF. Both instructions use the IEEE single precision floating-point format which defines the most significant bit to be the sign, the next eight most significant bits to be the exponent and the least 23 significant bits to be the fraction.

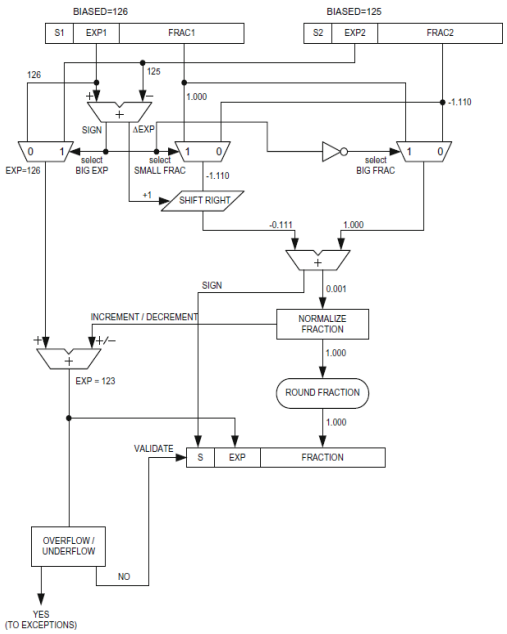

Fig: Floating-Point adder implementation

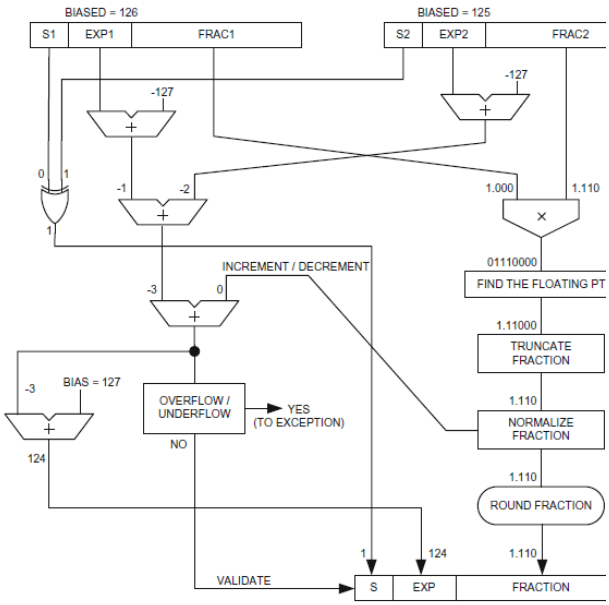

Fig: Floating-Point multiplier implementation

Conclusions:

In this way by combining all the individual modules, we created a top module which shows the successful implementation of a 32 Bit RISC CPU which can take both Fixed and Floating Point instructions along with handling various types of hazards.

REFERENCES

[1] Ahmet Bindal, *Fundamentals Of Computer Architecture and Design, 5th Ed.*