**SINGLE CYCLE CORE RV32I REPORT**

Introduction

In this report the implementation of single cycle circuit of RISC-V has been discussed. RISC-V is an open standard instruction set architecture based on established reduced instruction set computer principles. Unlike most other ISA designs, the RISC-V ISA is provided under open source licenses that do not require fees to use.in this report RISC-V ISA has been used to build a single cycle circuit which can perform single operation for example add, sub, mul etc at a time the importance of RISC-V architecture is that in this architecture we have reduced instruction set which means the cycles per instruction will be less which help us to perform tasks quickly. RISC-V architecture used here is of 32 bit which mean we have 32 register each of 32 bit in register file this contains RISC-V Single Cycle 32 Bit Processor simulation on Logic Simulator called [Logisim](http://www.cburch.com/logisim/download.html). This circuit contains 32 bit ALU, 32 bit Data Bus, 16KB ROM/RAM, 12 Bit Address Bus for both RAM MAR(memory address register). Register File contains 32 Registers with data width of 32 bits. Troubleshooting codes used to verify all the circuit components.

Methodology

There are 5 stage in every microprocessor

* Fetch
* Decode
* Execute
* Memory
* Write back

Fetch Stage

Comprises of program counter instruction memory and adder

(PC) counter hold address of instruction which we want to fetch from instruction memory whereas instruction memory hold the different instruction which we want to execute the adder part is used to increment the value of pc so that PC point to the next instruction which you get fetch from our instruction memory this complete our fetch part

Decode stage

Decoding stage comprises of immediate generation control unit and register file control unit has its inputs coming from Instruction memory in the form of BIT 0 to 6 which is opcode of instruction that decided which type of instruction we have in our instruction memory we have six type of instruction which is r-type store-type branch-type unconditional-type and other type immediate generation have two inputs coming into it. first is coming from instruction memory and second is coming from PC input which is our program counter

what immediate generation is done in generation basically add these two inputs and made sign extended and made compatible for ALU so that it may configure and value compute easily

we have register file in which we have 32 register each of resistor 32 bit we have two input form instruction memory to source register and we have write back register which is basically the destination register in our register file

Execute phase

Execute phase comprises of A L U control and ALU. ALU have 3 input function 3 and function 7 which is coming from our instruction memory ALU opcode which basically tells which operation in ALU has to perform

now moving toward ALU. ALU basically perform different operation depending upon the instruction we have if we have add instruction or subtract instruction from instruction memory then it will be compute the result and the result will be marked in ALU result output but if we have any branch instruction and branch gets true then ALU result output branch would get high

Memory stage

If we have any instruction in instruction memory of load and store and if we want any data to load from our memory or any data to store in our memory we use our data memory data memory signal is controlled by a control unit which is which is mem read and mem write signal

Write Back Stage

we will move to write backstage if we want to write any ALU result on load any data from memory we use right backstage how we can use right backstage this is a mux which is controlled by the control unit and we have mem to read signal and if mem to read signal is high then it will write the data of ALU result or data memory to our destination register which is in the register file this data would be written on the destination register

Implementation

At first learn the basic instructions of the RV32I Instruction Set Architecture and learn their functionality. To learn the backend working use Venus Online RV32I Simulator. This Simulator helps grasp the working behind the instruction much faster. On the Logic Simulator software first start with the program counter and memory address register. Then develop the circuit for the immediate generation which uses full instruction and PC to generate respective immediate. after that create register file with 32 registers each 32-bit data width. This register file takes 5-bit address to select one of the 32 registers and write data to it using register enable wire. two 5-bit address RS1 and RS2 are to read one of the 32 registers simultaneously. Now make 32 Bit ALU with 4-bit ALU operation Select which selects which operation to perform according to the instruction. After completing create type decoder which uses 7-bit opcode to decode the type of the instruction. Then in control Decoder depending upon the type of instruction, function3 and function7 different components are controlled. Integrate type decode and control decode, and this will become your control unit. Add RAM and configure its data bits to 32bit and address width to 12 bits. To handle branch instruction Branch circuit is now having to be created using the simple comparators and depending upon the RS1 and RS2 conditional jump is done if branch is true. In the end Add a 32-bit adder for jalr as this instruction requires two additions. One ALU cannot perform 2 operation on a single cycle. Connect the wiring using the splitters, multiplexers, constants, tunnels, and clocks. To troubleshoot the circuit, start with the simpler instructions e.g.; add, addi and watch the circuit behavior using temporary register outputs. To load the machine code on the Instruction Register, simulate the code on Venus then using dump feature copy the machine code and create xyzcode.mem extension file using All file saving option on notepad

Results

Test program

Fibonacci Series

addi x6,x0,0

addi x2,x0,1

addi x4,x0,0

addi x5,x0,10

label:

add x3,x6,x2

add x6,x2,x0

add x2,x3,x0

addi x4,x4,1

beq x4,x5,label2

jal label

label2:

Conclusion

In this project we created a single cycle RISC-V processor which is working perfectly and it can perform many tasks with less complex instructions which shows the importance of RISC-V in the future. RISC-V is surely going to be one of the most valued processors in future. The single cycle processor discussed above performs 1 operation at a time.

References

* Logisim Software [Logisim](http://www.cburch.com/logisim/download.html)
* Reference book for help [David\_A.\_Patterson,\_John\_L.\_Hennessy](https://drive.google.com/file/d/1zJ5JT1A-pmruRShfD6ikY52tFlcvrdYB/view)
* Venus online simulator [Venus](https://venus.cs61c.org/)
* Lecture https://youtube.com/playlist?list=PLz\_BCXrkxocSCY57dwH9qnSd-zkriiedz

Instruction

jal is used for jump any label. Jal is UJ type instruction in which we have immediate of 20 bit the address is calculated by adding immediate and current PC value and then the address is save in x1. Jalr is used for holding return address.

Jalr is I type in which we have x1 and immediate. Base + offset = address in which we want to jump and save address + 4 in destination register

return address = PC + 4

jal and jalr both held the value of RA in their respective rd jal instruction rd is x1 always and jalr rd is specified in instruction jalr x4,x3,8 x4 is rd which hold PC + 4