Fall 2021

ECE 414 FINAL PROJECT

RISC Processor Design in Verilog HDL

ABSTRACT

This document summarizes the details of a 16 bit RISC processor design to be completed in Verilog-HDL. The overall design steps starting with block level design to DFT has to be completed. The 16 bit RISC processor architecture features 16 bit instruction words, 16 internal general-purpose registers, and 6 external address lines to a ROM and 6 external address lines to an external memory (RAM).

INTRODUCTION

Features of the processor can be summarized as follows;

- 16-bit RISC microprocessor based on a simplified version of the MIPS architecture
- Every instruction is completed in two cycles
- 16 instructions in the instruction set architecture.
- 16 general purpose registers.
- 6 external address lines to a ROM and 6 external address lines to a RAM
- 16 data lines to a ROM and 16 data lines to a RAM
- One bit execution complete line

PIN DESCRIPTION

data_from_rom [15:0]	input	data bus from ROM (instruction memory)
reset	input	reset signal (resets any internal registers in the CPU)
clk	input	clock
address_to_rom [5:0]	output	address lines to the ROM
enable_to_rom	output	enable signal to ROM
data_ram[15:0]	input / output	data bus from RAM
write_enable_to_ram	output	write enable signal to RAM
read_enable_to_ram	output	read enable signal to RAM
address_to_ram [5:0]	output	address lines to the RAM
enable_ram_read	output	This is a signal for testing and evaluation part. It should be high once your CPU finishes executing the code.

NOTES

1. The number of assembly level lines in the program has to be hardcoded to the CPU to detect end of execution.
2. Although 16 bits are available (register width) to address RAM, physical RAM address is only 6 bits wide. (64 memory locations)

Instruction Set Architecture

The instruction set of the processor is given in following instruction table. The instruction words are 16-bits long. The following chart describes the instruction formats.

Operation	Opcode				Destination Reg				Source Reg				Target Reg			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	0	Rd				Rs				Rt			
SUB	0	0	0	1	Rd				Rs				Rt			
AND	0	0	1	0	Rd				Rs				Rt			
OR	0	0	1	1	Rd				Rs				Rt			
XOR	0	1	0	0	Rd				Rs				Rt			
NOT	0	1	0	1	Rd				Rs							
SLA	0	1	1	0	Rd				Rs							
SRA	0	1	1	1	Rd				Rs							
LI	1	0	0	0	Rd				Immediate							
LW	1	0	0	1	Rd				Rs							
SW	1	0	1	0					Rs				Rt			
BIZ	1	0	1	1	Rs				Offset							
BNZ	1	1	0	0	Rs				Offset							
JAL	1	1	0	1	Rd				Offset							
JMP	1	1	1	0					Offset							
JR	1	1	1	1	0	0	0	0	Rs							
EOE	1	1	1	1	1	1	1	1								

1. Arithmetic (Two's Complement) ALU operation

- **ADD**: Rd = Rs + Rt

Operands A and B stored in register locations Rs and Rt are added and written to the destination register specified by Rd.

- **SUB**: Rd = Rs - Rt

  Operand B (Rt) is subtracted from Operand A (Rs) and written to Rd.

2. Logical ALU operation (6)

- **AND**: Rd = Rs & Rt

  Operand A (Rs) is bitwise anded with Operand B (Rt) and written into Rd.

- **OR**: Rd = Rs | Rt

Operand A (Rs) is bitwise ored with Operand B (Rt) and written into Rd.

- **XOR**: Rd = Rs ^ Rt

Operand A (Rs) is bitwise Xored with Operand B (Rt) and written into Rd.

- **NOT**: Rd = ~Rs

Operand A (Rs) is bitwise inverted and written into Rd.

- **SLA**: Rd = Rs << 1

Operand A (Rs) is arithmetically shifted to the left by one bit and written into Rd.

- **SRA**: Rd = Rs >> 1

Operand A (Rs) is arithmetically shifted to the right by one bit and written into Rd. The MSB (sign bit) will be preserved for this operation.

3. Memory operations (3)

- **LI:** Rd = 8-bit Sign extended Immediate

The 8-bit immediate in the Instruction word is sign-extended to 16-bits and written into the register specified by Rd.

- **LW:** Rd = Mem[Rs[5:0]]

The memory word at address specified by the first six bits of register Rs is loaded into register Rd.

- **SW:** Mem[Rs[5:0]] = Rt

The data in register Rt is stored into the memory location specified by Rs[5:0].

4. Conditional Branch operations (2)

- **BIZ:** PC = PC + 1 + Offset **if** Rs = 0

If all the bits in register Rs are zero than the current Program Count (PC + 1) is offset to PC + 1 + Offset. The count is offset from PC + 1 because it is incremented and stored during the Fetch cycle.

- **BNZ:** PC = PC + 1 + Offset **if** Rs != 0

If all the bits in register Rs are not zero than the current Program Count (PC + 1) is offset to PC + 1 + Offset.

5. Program Count Jump operations (3)

- **JAL**: Rd = PC + 1 and PC = PC + 1 + Offset

Jump and Link instruction would write current Program Count in register Rd and offset the program count to PC + 1 + Offset

- **JMP**: PC = PC + 1 + Offset

Unconditional jump instruction will Offset the program count to PC + 1 + Offset.

- **JR**: PC = Rs

Jump Return instruction will set the Program Count to the one previously stored in JAL.

- **EOE**: End of execution. This is a pseudo-instruction that you can use to indicate the end of your code.