Verilog Project

Simple Single Cycle RISC_V

Submitted by: AbdelRahman Atef Eid Yassin Yousef

Track: Digital IC Design

Supervisor: TA\ Fayza

Date of submission : 16/8/2023

Objective:

Design and implement RISC V Single Cycle processor which can execute some simple reduced Instruction set .

Problem Statement:

Implement a single cycle 32-bit RISC V processor using Verilog language that can execute a stored program, which consists of some simple reduced Instruction set architecture. The Design should be able to execute the instruction in just one clock cycle .

Design & Implementation

Main Design

The main design is implemented in file attached RISC_v.v which implement the schematic diagram below.

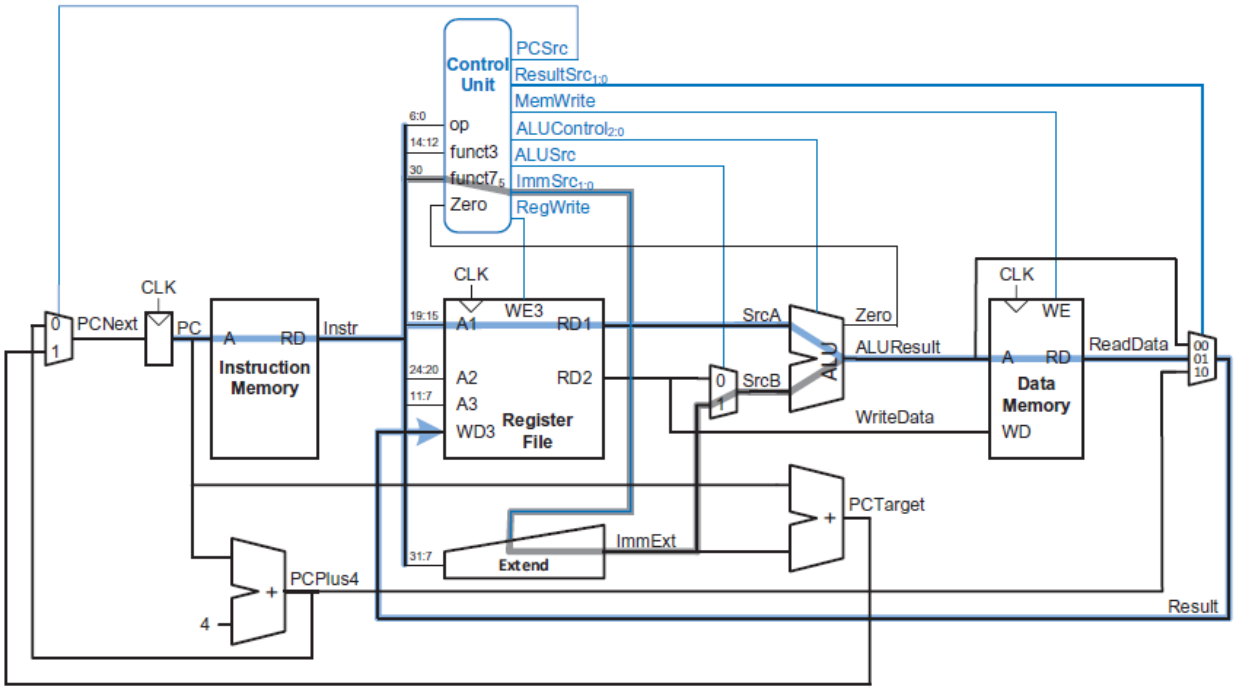


Figure 7.17 Critical path for `lw`

The Structural Design consists of many smaller components:

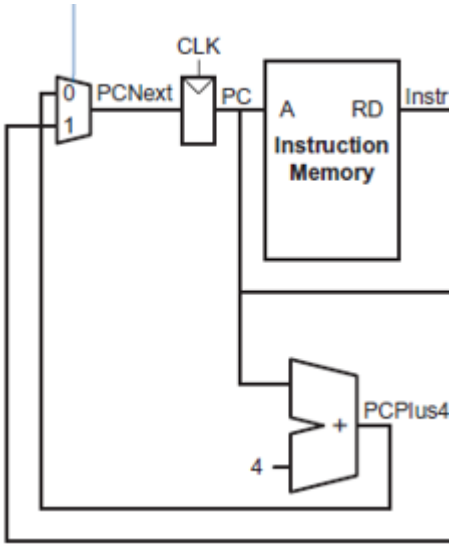
1- PC (Program Counter to store address of current instruction executed)
2- Instruction Memory (Store Program to be executed)
3- Register File (store variables we want to use)
4- ALU (performs arithmetic and logic operations)
5- Data Memory (large storage than reg file)
6- Immediate extender (extend sign bit to allow negative numbers evaluation)
7- Muxs (choose the inputs to alu and pc)
8- Control Unit (Controls interconnection between All Component

Instruction Set Architecture

Our design supports the following reduced instruction set :

Instruction	Type	illustration
Add $s1,$s2,$s3	R Type	Addition of two numbers
Sub $s1,$s2,$s3		Subtraction of two numbers
Or $s1,$s2,$s3		Oring logic operation of two numbers
And $s1,$s2,$s3		Anding logic operation of two numbers
SLT $s1,$s2,$s3		If s2 < s3 set s1 by 1
Addi $s1,$s2,5	I type	Immediate Addition
Andi $s1,$s2,5		Immediate Anding
Ori $s1,$s2,5		Immediate Oring
Slti $s1,$s2,5		Immediate Set less than
Lw $s1,4($s2)		Load value from memory to register
Sw $s1,4($s2)	S type	Store value from register to memory
Jal rd,immediate	J Type	Jump to a block of program and save next instruction to a register to return to it

Components of the design



1- PC

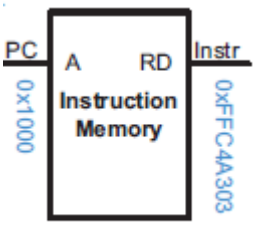
A program Counter register which holds current instruction address which is 32 bit wide is implemented in **PC.v** which gets a new address each clock positive edge .

A mux is used to choose address of PC+4 or branch address

2- Instruction Memory

An Instruction memory of 256 location for testing which each location has 4 byte word of 32 bit . Each location is denoted by 32 bit Address in multiple of 4 .

Memory is implemented in file Instruction_Memory.v which reads the instructions in file mymem2.dat which contains the stored program we want to execute in hexadecimal notation .



mymem2.dat - Not

File Edit Format Vi

```
002081B3
403202B3
00808383
0013F333
0080c623
001122B3
00210463
002081B3
002081B3
00110293
00B564B3
00312293
00517293
0056E793
```

memory_instructions_illustrations.txt - Notepad

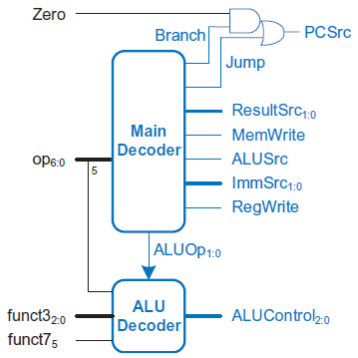
File Edit Format View Help

```
002081B3; // add $3, $1, $2
403202B3; // sub $5, $4, $3
00808383; // lw $7, 8($1)
0013F333; // and $6, $1, $7
0080c623; // sw $8,12($1)
001122B3; // slt $5, $2, $1
00210463; // beq $2, $2, 8 // 8 is offset
002081B3; // add $3, $1, $2
002081B3; // add $3, $1, $2
00110293; // addi $5, $2, 1
00B564B3; // OR $9,$10,$11
00312293; // slti $5, $2, 3
00517293; // andi $5, $2, 5
0056E793; // ori $15,$13,5
```

3- Control Unit

A control unit implemented in file Control_Unit.v which has two main components

1- Main Decoder : Responsible for generating all control signals of other components implemented in file Main_Decoder.v
2- ALU Decoder : Responsible for generating control signals for ALU
implemented in file ALU_Decoder.v



Control Unit is responsible for decoding instruction to tell other components what to do.

Main Decoder Truth Table :

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	00	1	0	01	0	00	0
sw	0100011	0	01	1	1	xx	0	00	0
R-type	0110011	1	xx	0	0	00	0	10	0
beq	1100011	0	10	0	0	xx	1	01	0
I-type ALU	0010011	1	00	1	0	00	0	10	0
jal	1101111	1	11	x	0	10	0	xx	1

ALU Decoder Truth Table :

Table 7.3 ALU Decoder truth table

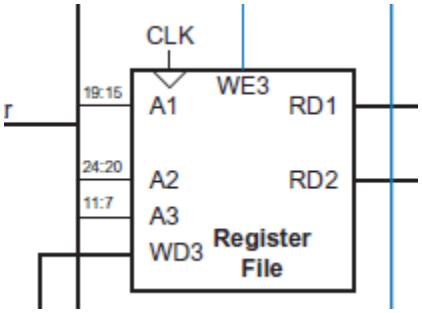
ALUOp	funct3	{op5, funct75}	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

4- Register File

A 32 locations of 32 bits wide register file is implemented in **REG_FILE.v** . Read operation doesn't depend on clock but the address of A1 and A2 are put on the output . Write Operation is synchronous with clock .

Table 6.1 RISC-V register set

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0–2	x5–7	Temporary registers
s0/fp	x8	Saved register/Frame pointer
s1	x9	Saved register
a0–1	x10–11	Function arguments/Return values
a2–7	x12–17	Function arguments
s2–11	x18–27	Saved registers
t3–6	x28–31	Temporary registers



5- Immediate Extender

It is implemented in **Immediate_extender.v** which is responsible for extending the sign of branch address to match 32 bit address we deal with . It is implemented according to truth table :

ImmSrc	ImmExt	Type	Description
00	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
01	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
10	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate
11	{{12{*Instr*[31]}}, *Instr*[19:12], *Instr*[20], *Instr*[30:21], 1'b0}	J	21-bit signed immediate

6- ALU

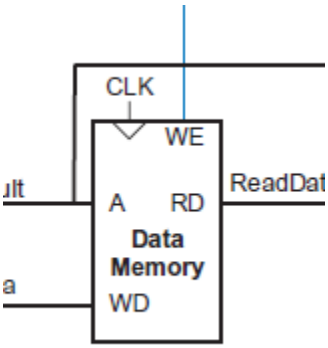
It is implemented in file **ALU.v** which does the Arthimetic and logic operations according to truth table below :

ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

7- Data Memory

A Data Memory also implemented in **Data_Memory.v** which has asynchronous read operation according to 32 bit address location which is also should be multiple of 4 as each word in memory is 32 bit wide (4 bytes) .

It has a synchronous write signal with the positive edge of the clk.



Simulation Results

Tables below explains how the program executes

A register file initialized with hex values inside file myreg.dat

Data Memory initialized with hex values in file mydmem.dat

```
1  00000000
2  00000001
3  00000002
4  00000003
5  00000004
6  00000005
7  00000006
8  00000007
```

Figure 2 Data Memory Data

```
 1  0
 2  4
 3  10
 4  12
 5  18
 6  20
 7  24
 8  28
 9  30
10  32
11  34
12  32
13  30
14  28
15  26
16  24
17  28
18  30
19  32
20  34
21  30
22  80
23  90
24  60
25  70
26  50
27  40
28  31
29  20
30  22
31  24
32  37
```

Figure 1 Reg initialization

A table below summarizes the expected output to each instruction

Instruction	Effect on next clock cycle
add $3, $1, $2	$3 = 14h
sub $5, $4, $3	$5 =04h
lw $7, 8($1)	$7 = 03h
and $6, $1, $7	$6 = 4h and 12h = 0
sw $8,12($1)	Location number 4 in memory = 30h
slt $5, $2, $1	$5 =0
beq $2, $2, 8 // 8 is offset to label l1	Pc =
add $3, $1, $2	skipped
L1:add $3, $1, $2	$3 = 14h
addi $5, $2, 1	$5 =11h
OR $9,$10,$11	$9 = 34 h or 32h = 36h
slti $5, $2, 3	$5 =0
andi $5, $2, 5	$5 = 10h and 5 = 0
ori $15,$13,5	$15 = 28 h or 5 = 2dh

A simulation wave form is taken below and we check the values on register and memory if executes the program