Single Cycle Datapath Processor using MIPS

Piero Morales
Computer Science University Student
University of Engineering and Technology
Lima, Peru
piero.morales@utec.edu.pe

Angel Motta
Computer Science University Student
University of Engineering and Technology
Lima, Peru
angel.motta@utec.edu.pe

***Abstract*—The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. The CPU has become the nerve center of any computer, from mobile devices to supercomputers. From the beginning of computer era scientists have tried to improve processor performance not only increasing the number of transistors, but also by improving the instructions that the processor executes. A major change that happened for CPUs is the change from single core to multi core that increasing they performance. In this way, Moore's law, that until this moment had traced teh future of processors, is discarded.**

***Index Terms*—Computer architecture, risc, verilog, processor, big endgian, microprocessor without interlocked pipeline stages**

I. Introduction

There are two major architectures for CPUs, the Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC). RICS design architecture points to solve problems about the CPU time processing although compare with CISC it increase the lines of code to the software developer, nevertheless the main advantage of using RISC architecture is the reduced amount of clock cycles needed to executed and instruction due to the specialiazed instructions.

In this context we have the 32 bits MIPS Instruction Set Architecture (ISA) [1] which support all the necesary functions needed by the software, MIPS is composed by 32 general-purpose registers and instructions format to clasify all the instructions. There are 3 types of instruction formats: the R-Type which uses 3 registers, the I-Type who uses 2 registers and a 16-bit immediate value and finally we have the J-Type who supports the "jumps" between the lines of instructions.

There is no definite architecture in the present CPU technology, the industry use both RICS and CISC architecture separaredly or in combination, which of those are going to use will depend on the requirement.

II. Methodology

Since manufacturing a physical processor require a state-of-the-art technology and also a huge amount of money, we use an Hardware Description Language (HDL) to design and simulate our processor and all the components related. We choose Verilog [2] as HDL because is widely used in the industry and the access to the student license for ModelSim [3].

We choose the single cycle as a design metodology with focussing in the basic operations with integers, covering the following R-type, I-type and J-type instructions from the 32 bits MIPS ISA:

TABLE I
R Type

Instructions		
ADD	Subtraction(SUB)	AND
NOR	OR	Set Less Than (SLT)
Jump Register (JR)		

TABLE II
I Type

Instructions		
Add Immediate (ADDI)	Subtraction Inmediate (SUBI)	AND Inmediate (ANDI)
OR Immediate (ORI)	Set Less Than Immediate (SLTI)	Store Byte (SB)
Store Halfword (SH)	Store Word (SW)	Load Byte (LB)
Load Halfword (LH)	Load Word (LW)	Load Upper Immediate (LUI)
Branch On Equal (BEQ)	Branch On Not Equal (BNEQ)	Branch On Greater than equal zero (BGEZ)

TABLE III
J Type

Instructions		
Jump (J)	Jump and Link (JAL)	

A. Datapath

In order to cover all the instructions mentioned we need to model the following components:

- Instruction Memory, stores the instructions to be executed.
- PC Counter, points to the line of the instruction of the program which not necessary always will be the following next since the we are using branchs and jumps.
- Register File, stores the 32 registers for the MIPS ISA.
- Data Memory, stores the data for the program and also the stack.

- Aritmetic Logic Unit (ALU), the ”brain” of the processor who make the operations of adition, subtraction, the comparation between two numbers, logic AND, logic OR, logic NOR.
- Multiplexor 2 to 1, this component indicates which of the 2 inputs input take based on a selector signal i.e. in the selection between the PC Counter, the branch or the jump.
- Adder, we use this to add the number 4 to the actual PC counter to point to the next instruction, also is used for the offset to cover the branch instruction.
- Shift Left 2 and 16, to be used to calculate the offset for the branch and load a number up to 32 bits respectively.
- Sign extend, this component is used to extend the most significant bit of the number.
- and the control component for support all the instructions deciding which signal activate depending on the type of instruction and the operation.

Putting all the components together we get our datapath:

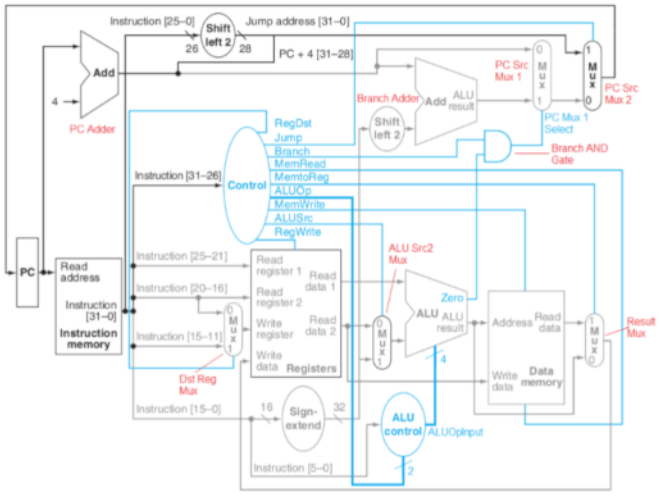

Fig. 1. Datapath.

As we can see all the components are connected using wires.

B. ModelSim

We start developing our components in verilog, in order to maintenance and following the standard principles of software developing we create each module separatedly in its corresponding .v file, e.g. regfile.v for the Register File component. In total we have the following 17 files for all the modules:

regfile_lab5.v	shift16_module.v
alu_lab5.v	instrumem.v
extendbit_lab5.v	ControlDataPath.v
alucontrol.v	mux31.v
Select_word_half.v	mux_5bits.v
shift_left_2.v	pccounter_4.v
PC_module.v	concatenar.v
adder32.v	complete_32bits.v
proyecto_final.v	

To clarify the terms every CPU component will be created in ModelSim as a module [4].

C:/Programas/Modeltech_pe_edu_10.4a/examples/Sabado/regfile_lab5.v - Default

```
Ln#
1  //------------------------------------------------------
2  // Design Name : regfile
3  // File Name   : regfile_lab5.v
4  // Function    : Store the MIPS 32 registers
5  // Coder       : Raúl Mosquera Pumaricra
6  //------------------------------------------------------
7  module regfile(in1, in2, in3, wreg, clk, sel, out1, out2);
8  input in1;
9  input in2;
10 input in3;
11 input clk;
12 input wreg;
13 input sel;
14 output out1;
15 output out2;
16
17 wire sel;
18 wire [4:0] in1, in2, in3;
19 wire [31:0] wreg;
20 wire [31:0] out1, out2;
21 reg[31:0] memory[31:0];
22
```

Fig. 2. Register File module in ModelSim.

III. Experimental Setup

To test our datapath we first test each component separetly to isolate the problems, once all the components pass its respective tests we continue with a complete test of our datapath, in order to do that we agroup the previous instructions in 3 files to be loaded in the instruction memory.

Each test bench file contains the instructions and its respective operands (registers). Previously for purpose testing we load the register file with random values.

TABLE IV
Registers

Register number	Number in decimal notation	Number in hexadecimal notation
16	49527	0000C177
17	63767	0000F917
18	31778	00007C22
19	23198	00005A9E
20	917	00000395
21	24182	00005E76
22	52687	0000CDCF
23	20726	000050F6
29	150	00000096

The others registers, 1 to 15, 24 to 28 and 30,31 remain with zero.

In each test bench are using 10 nanoseconds as a positive clock signal and negative clock signal which give us 20 nanoseconds in total per clock cycle.

To get a real aproach of the use of the processor we will run the following C code.

The translation into MIPS instructions it would be as follow:

As we can use the factorial function will use most of the intructions implemented and the recursivity technique, this will be our test bench 4.

TABLE V
TESTBENCH 1

Instructions		
ADD	Subtraction (SUB)	AND
NOR	OR	Set Less Than (SLT)
Add Immediate (ADDI)	Subtraction Inmediate (SUBI)	AND Inmediate (ANDI)
OR Immediate (ORI)	Set Less Than Immediate (SLTI)	

TABLE VI
TESTBENCH 2

Instructions		
Store Byte (SB)	Store Halfword (SH)	Store Word (SW)
Load Byte (LB)	Load Halfword (LH)	Load Word (LW)
Load Upper Inmediate (LUI)		

To calculate the CPU time [5] (time processing) for each test bench we use the following formula:

$$Time = PI * CPI * TimeperClockCycle$$

where PI is Program Instructions and CPI is Clock Cycles per instruction.

IV. EVALUATION

We executed the test bench 1, 2, 3 and 4, these were the results:

We ran the test bench in intervals of 100 nanoseconds as we can see in Fig. 5. . For the test bench 1 we get 100 nanoseconds + 100 nanoseconds + 20 nanoseconds with in total give us 220 nanoseconds.

We apply the same procedure to the test bench 2 , Fig. 8. and the result was 100 nanoseconds + 40 nanoseconds = 140 nanoseconds.

In test bench 3, Fig. 9., we got 100 nanoseconds + 100 nanoseconds + 100 nanoseconds + 40 nanoseconds with th total of 340 nanoseconds.

And finally for the test bench 4, Fig. 6, we use intervals of 500 nanoseconds since the execution is elevated, we get 2610 nanoseconds in total also the output of the factorial of 10 was 0x003750f00 which in decimal notation correspond to 3628800.

Comparing the results of Fig. 5., Fig. 8. and Fig. 9. with the Table VIII we get the same amount of clock cycles and the time for each file, also the results of the instructions are as we expected.

TABLE VII
TESTBENCH 3

Instructions		
Branch On Equal (BEQ)	Branch On Not Equal (BNEQ)	Branch On Greater than equal zero (BGEZ)
Jump (J)	Jump and Link (JAL)	Jump Register (JR)

Since we need to simulate the branch and the jumps between the instructions additional instructions were added i.e. ADDI and SUB

```
int fact(int n){
        if (n<1)
        return 1;
        else
        return n*factorial(n-1)
        }
variable = factorial(10);
```

Fig. 3. Factorial function - C code.

ADDi	$a0	$0	10
JAL		factorial	
ADD	$s0	$v0	$0
SUBI	$sp	$sp	8
SW	$a0	$sp	0
SW	$ra	$sp	4
SLTI	$t0	$a0	1
BEQ	$t0	$0	label1
ADDI	$v0	$0	1
ADDI	$sp	$sp	8
JR		$ra	
SUBI	$a0	$a0	1
JAL		factorial	
LW	$a0	$sp	0
LW	$ra	$sp	4
ADDI	$sp	$sp	8
MULTI	$v0	$v0	$a0
JR		$ra	

Fig. 4. Factorial function - MIPS.

V. CONCLUSION

- Since the single cycle datapath was the original solution in the early days of RISC architecture nowadays is not efficient because it takes 1 cycle per instruction when another aproachs using the pipeline technique could reduce the cycle needed per instruction.
- It became mandatory calculate the expected time processing for test our datapath, otherwise we could get unexpected result since the execution continue with the next instruction, like for example in our test bench for the factorial.
- Before start coding in ModelSim we need to decide if we are going to apply the clock edge triggered and which modules will apply for that, because if not the major changes needed in the modules would be very risky and will take more time for testing.
- Verilog was not specially designed to upload files that why for our tests purposes we need to modify the code to point a specific file.

VI. COMMENTS

- When we are simulating our component in ModelSim no warnings must appear when the simulation starts,

TABLE VIII
CLOCK CYCLES

	Total instructions	Total executed instructions (Expected)	Clock Cycles	CPU Time (Nanoseconds)
Test bench 1	11	11	11	220
Test bench 2	7	7	7	140
Test bench 3	22	17	17	340
Test bench 4	18	131	131	2620

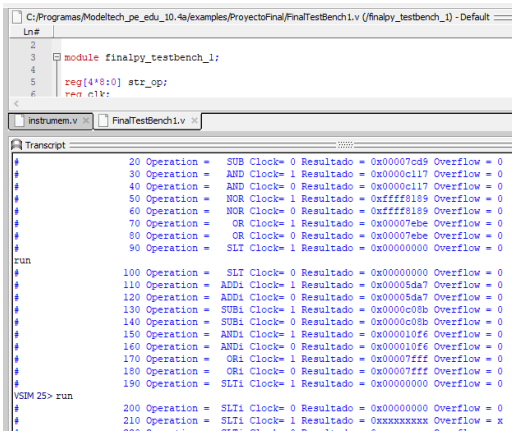

Fig. 5. Execution results for test bench 1.

otherwise there was some error or unexpected behaviour. One common issue is referring a wire or register as an input or output of a module with diferent length.

- In ModelSim is the identifier is not declare verilog assume is a wire.
- The verilog compiler doesn't warn you when a module instantiation does not exists until you simulate it
- One common problem is asume the execution of the code in the components of the datapth will be sequential, that is not correct since we have the always @ block and that could be executed in the upper sign of the clock or the lower sign.
- For those who are used to the conditional statements of the programming languages it is a little difficult at the beginning use verilog, because at the digital circuit level there we only have and, or, xor and all the gates.
- To find the errors in the testing fase we can navigate in the windows objects in ModelSim throught the modules to find the issue.

REFERENCES

[1] MIPS.com. (2016). MIPS® Architecture for Programmers Volume II-A: The MIPS32® Instruction Set Manual. [online] Available at: https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf [Accessed 27 Nov. 2018].

[2] IEEE Standard for Verilog Hardware Description Language. IEEE Standard 1364-2005 (Revision of IEEE Standard 1364-2001). http://dx.doi.org/10.1109/IEEESTD.2006.99495, 2006. Last access 26 November 2018.

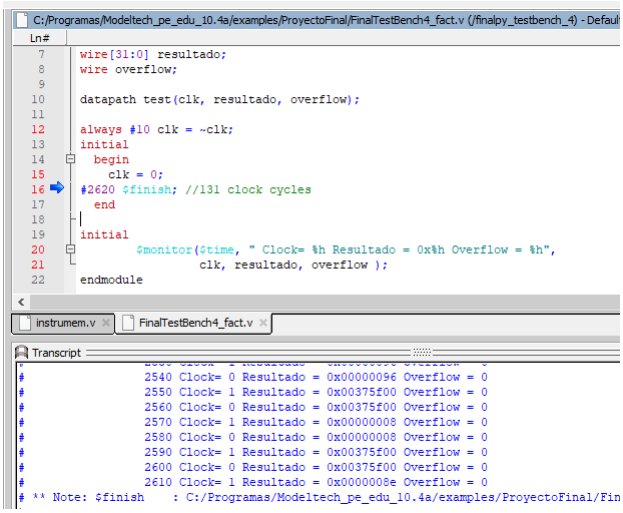

Fig. 6. Execution results of the test bench for the factorial.

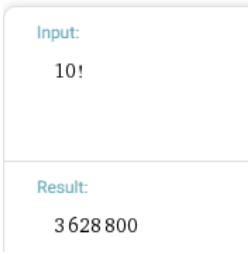

Fig. 7. Wolfram Alpha - Factorial of 10. [6]

[3] Mentor.com. (2018). ModelSim PE Student Edition. [online] Available at: https://www.mentor.com/company/higher_ed/modelsim-student-edition [Accessed 27 Nov. 2018].

[4] Ashenden, P. (2008). Digital Design: An Embedded Systems Approach Using Verilog. Burlington, MA: Elsevier Science, pp.22,23.

[5] Patterson, D., Hennessy, J. and Alexander, P. (2012). Computer organization and design. 4th ed. Waltham, Mass: Morgan Kaufmann, pp.35.

[6] Wolframalpha.com (2018). Wolfram—Alpha: Making the world's knowledge computable. [online] Wolframalpha.com. Availableat:https://www.wolframalpha.com/input/?i=factorial+10 [Accessed 28 Nov. 2018].

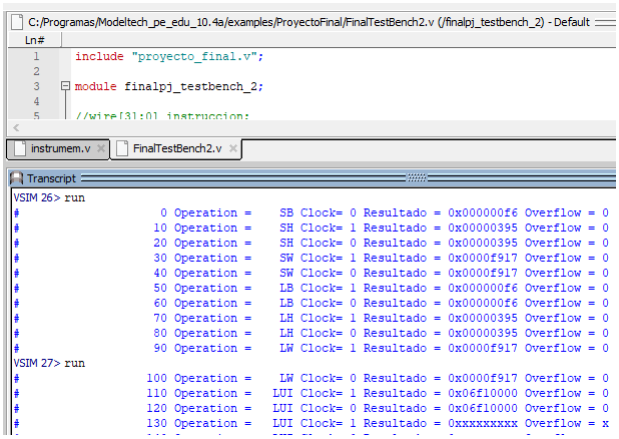

Fig. 8. Execution results for test bench 2.

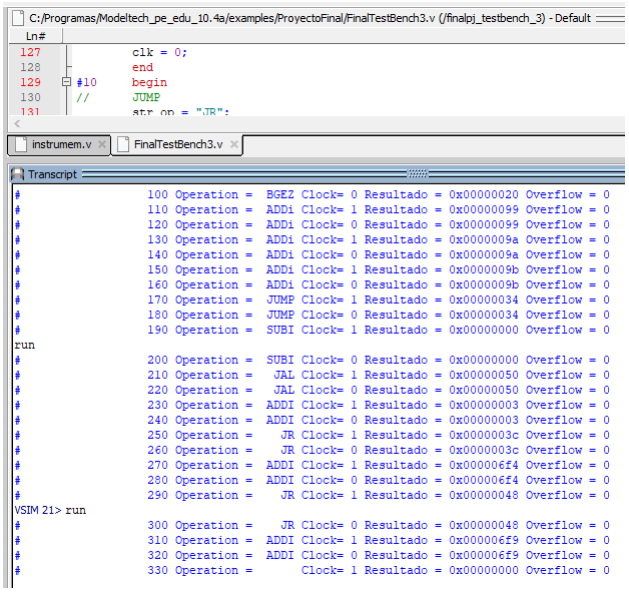

Fig. 9. Execution results for test bench 3.