HARDWARE MODELLING OF A 32-BIT, SINGLE CYCLE RISC PROCESSOR USING VHDL

Safaa S. Omran,

College of Electrical and Electronic Techniques

Baghdad/Iraq

omran_safaa@ymail.com

Hadeel S. Mahmood

College of Electrical and Electronic Techniques

Baghdad/Iraq

hadeel_shakir@yahoo.com

Abstract

In this research, the VHDL (Very high speed IC Hardware Description Language) hardware modelling of the complete design of a 32-bit MIPS (Microprocessor without Interlocked Pipeline Stages), single cycle RISC (Reduced Instruction Set Computer) processor is presented. First the work defines the MIPS ISA (Instruction Set Architecture) and then describes how to divide the processor's complete design into two parts: the datapath unit, which includes a multiplication/division unit and the control unit. Next, a top level is implemented by connecting data and instruction memories to the processor. The MIPS and top level is designed using **(Xilinx ISE Design Suite 13.4)** program. Finally a procedure that performs a multiplication of two numbers is executed and the results are discussed.

Keywords - VHDL, RISC, MIPS, ISA, datapath.

1 INTRODUCTION

VHDL (Very high speed IC Hardware Description Language) is a hardware description language. It describes the behaviour of an electronic circuit or system, from which the physical circuit or system can then be implemented. A fundamental motivation to use VHDL is that VHDL is a standard, technology/vendor independent language, and is therefore portable and reusable [1].

RISC (Reduced Instruction Set Computer) architecture is a processor architecture in which all operations on data apply to data in registers and typically change the entire register. The only operations that affect memory are load and store operations that move data from memory to a register or to memory from a register, respectively. Load and store operations that load or store less than a full 32_bit register (e.g., a byte, or 16 bits) are often available, so it is also called load-store architecture. The instruction formats are few in number with all instructions typically being one size. This makes its implementation easier [2].

Single cycle MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC processor that can execute an entire instruction in one cycle. The cycle time is limited by the slowest instruction.

Many previous researches have implemented the simple design of a single cycle RISC processor in VHDL. Some of them performed the simple design of MIPS processor which can execute basic instructions (add, sub, lw, sw, branch)[3][4] and (jump)[5].while others made a VHDL model of the DLX processor [6][7].

Since one of the major utilities of VHDL is that it allows the synthesis of a circuit or system in a programmable device or in an ASIC (Application Specific Integrated Circuit), this paper studies the designing and prototyping of a complete single cycle MPS RISC processor in VHDL.

2 INSTRUCTION SET ARCHITECTURE

To command a computer's hardware, we must speak its language. The words of a machine's language are called instructions, and its vocabulary is called an instruction set [8].

For design simplicity of RISC (Reduced Instruction Set Computer) processor, all instructions must be kept in the same length and had a single instruction format. MIPS (Microprocessor without Interlocked Pipeline Stages) uses 32-bit instructions and defines three instruction formats as shown in table 1. :

1. R-type instructions: is short for register-type. They use three registers as operands: two as sources, and one as destination.
2. I-type instructions: is short for immediate-type. They use two registers operand and one 16-bit immediate operand.
3. J-type instructions: is short for jump-type. It is used only with jump instructions and uses a single 26-bit address operand.

Table 1. Formats of MIPS instructions

Field size	6-Bits	5-Bits	5-Bits	5-Bits	5-Bits	6-Bits
R-Type	op	rs	rt	rd	shamt	funct
I-Type	op	rs	rt	imm		
J-Type	op	addr				

Where op: basic operation of the instruction, traditionally called the opcode.

rs: the first register source operand.

rt: the second register source operand

rd: the register destination operand, it gets the result of the operation.

shamt: shift amount, it is used in shift instruction to hold shift amount.

funct: function, it selects the specific variant of the operation in the op field.

imm: the 16-bit address which is used in data transfer instructions.

addr: the 26-bit address which is used in jump instructions.

3 PROCESSOR'S DESIGN

The complete design of a 32-bit, single cycle MIPS (Microprocessor without Interlocked Pipeline Stages) processor consists of two interacting parts:

1. 32-bit datapath
2. Control unit.

The design has the ability to execute each instruction in a single clock cycle; therefore the clock cycle period is limited by the slowest instruction's execution time. Fig. 1 shows the complete design of the single cycle MIPS with its datapath includes mul/div unit which used to perform signed/unsigned multiplication and signed/unsigned division operations and the control unit. The design can perform the integer arithmetic-logical instructions, the memory reference instructions and the branch instructions.

3.1 32-BIT DATAPATH

A 32-bit MIPS (Microprocessor without Interlocked Pipeline Stages) requires a 32-bit datapath. In Fig.1 the datapath elements are black in colour. The datapath operates on 32_bit words of data, 16_bit half_words of data or 8_bit bytes of data. It contains elements such as memories, registers, ALUs (arithmetic logic unit), multiplexers, sign and zero extenders. A description of each datapath element is given below:

1. Program counter (PC): is 32-bit register. Its output (**pc**) represents the address of the current instruction (**instr**) to be executed while its input (**pcnext**) represents the address of the next instruction.

2. Instruction memory takes a 32-bit address from PC register and read a 32-bit data at the output port.

3. Register file consists of 32 registers each of 32-bit in size. It has two read ports (**RD1**, **RD2**) and one write port (**WD3**). Read ports (**RD1**, **RD2**) take 5-bit address inputs(**A1**, **A2**) which in turn select one of the 32 registers to be read on the read output ports (**RD1**, **RD2**). Write port (**WD3**) takes 5-bit

address input (**A3**) which in turn select one of the 32 registers to which the 32-bit at (**WD3**) input port will be written if the **WE** signal is **1** on the raising edge of the clock signal.

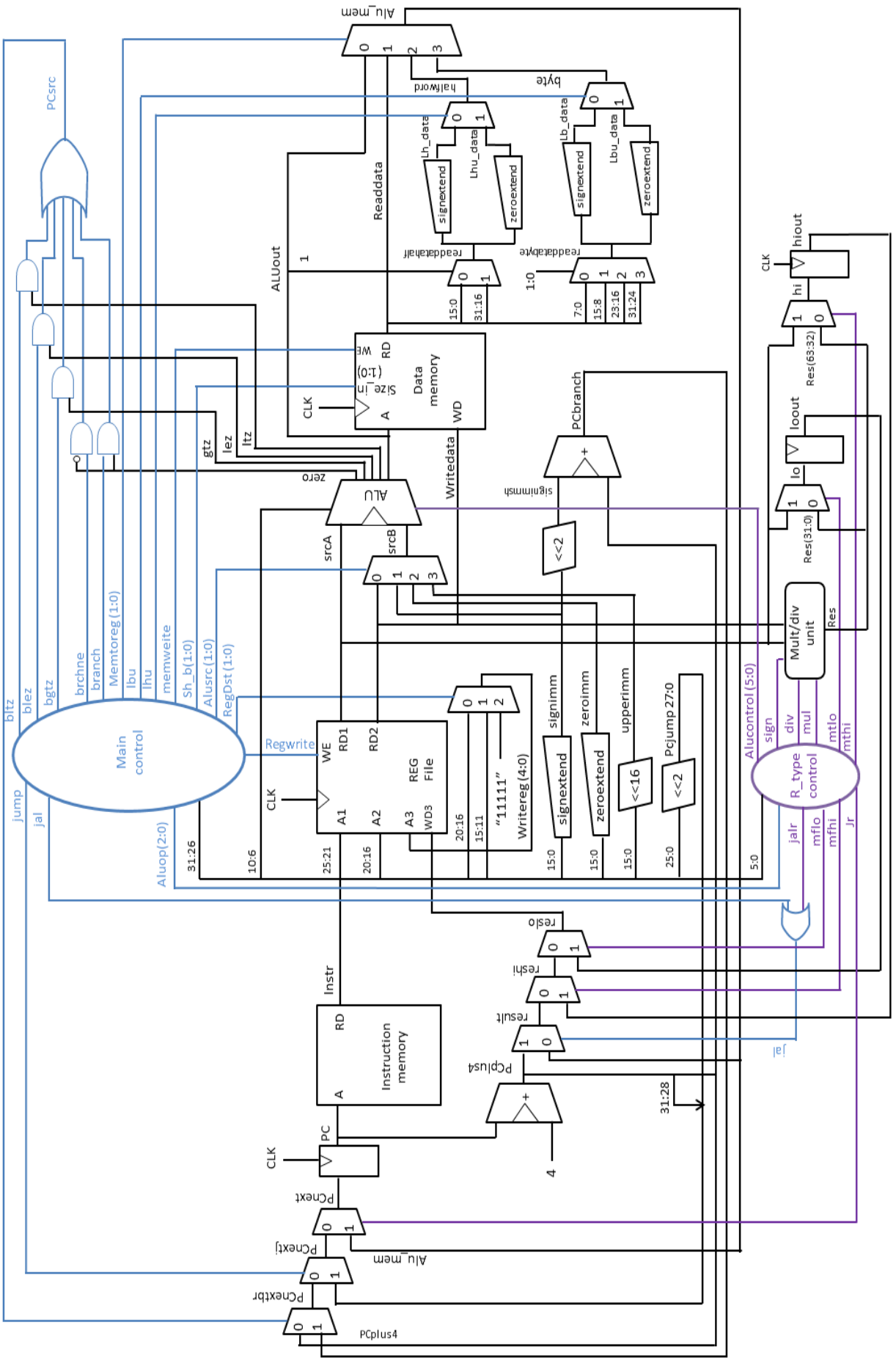


Fig.1. The complete datapath control scheme of MIPS

4. Data memory has one output read port (**RD**) and one input write port (**WR**). 32-bit data at input (**WR**) port is written to memory location specified by the address (**A**) if (**WE**) signal is **1** at the raising edge of the clock signal. Signal (**size_in**) limits the size of data to be writing either to byte (8-bit) or half word (16-bit). The content of memory location selected by (**A**) input is always available at (**RD**) output port.

5. Multiplexers are used to select one input from several inputs and pass it to the output. Mux (multiplexer) select line is controlled by a signal provided by the control unit.

6. Sign extender simply copies the sign bit (most significant bit) of a short input to all of the upper bits of the longer output.

7. Zero extender takes a short input and simply puts zeros in all of the upper bits of the longer output.

8. ALU (arithmetic logic unit) has been optimised in order to execute all the arithmetic-logical instructions. Fig. 2 shows the new architecture of the extended ALU. ALU takes **alucontrol (5:0)** as inputs and generates the ALU functions according to it. Table 2 illustrates functions that can be executed by the ALU.

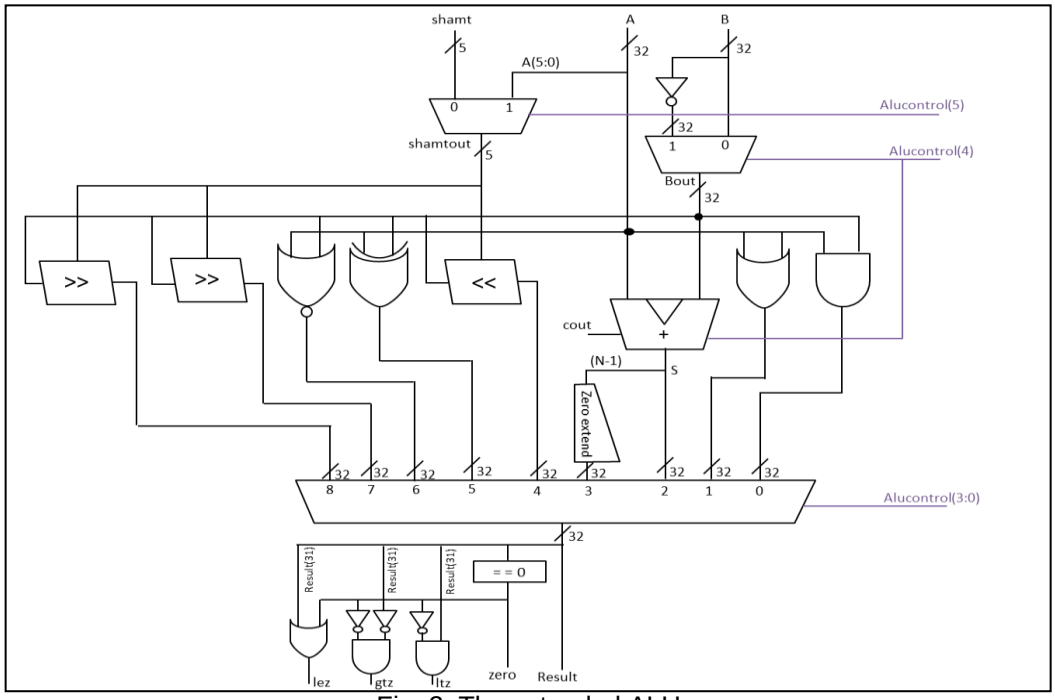


Fig. 2. The extended ALU

Alucontrol (5:0)	function
000000	A and B
000001	A or B
000010	A + B
000011	Not used
000100	Sll
000101	A xor B
000110	A nor B
000111	Srl
001000	Sra
010000	A & B’
010001	A or B’
010010	A – B
010011	Slt
010100	Not used
010101	A xor B’
010110	A nor B’
100100	Sllv
100111	Srlv
101000	Srav

9. mul/div unit performs signed/unsigned multiplication and signed/unsigned division. It takes two inputs of 32 bits (**A** and **B**) and produce **y** output of 64 bits, if **sign** is **1** then signed operation will be performed, otherwise it will perform unsigned operation. When **mult** input is **1** and **div** is **0**, **Y(63:0)=A*B** and when **mult** is **0** and **div** is **1**, **Y(31:0)=a/b** and **Y(63:32)= remainder**. Then **Y(63:32)** is stored at hi register while **Y (31:0)** is stored at lo register. Fig. 3 represents the mul/div unit with its inputs, outputs and control signals.

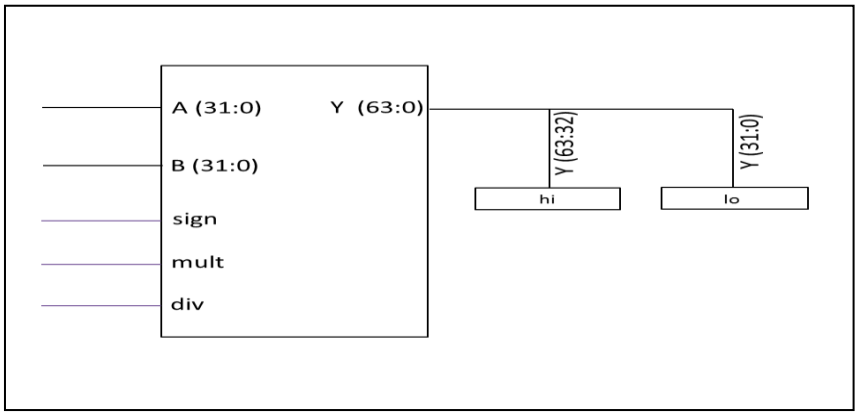


Fig.3. mul/div unit

3.2 CONTROL UNIT

The Control unit receives **opcode (instr 31:26)** and **funct (instr5:0)** fields of the current instruction from the datapath then tell it what to do in order to execute that instruction. The control unit provides multiplexer select, memory write, and control signals of ALU and mul/div unit. It consists of two parts: main control and R-type control.

Main control takes **opcode (instr 31:26)** field as inputs and produce **multiplexer select**, **memory**

Table 3. main control truth table

Instruction	Opcode	Sh_B	Lbu	Lhu	Regwrite	Regdst	Alusrc	Branch	Brchne	Blez	Bltz	Bgtz	Memwrite	Memtoreg	Jump	Jal	Aluop
R_type	000000	xx	x	x	1	01	00	0	0	0	0	0	0	00	0	0	110
lw	100011	xx	x	x	1	00	01	0	0	0	0	0	0	01	0	0	000
sw	101011	11	0	0	0	xx	01	0	0	0	0	0	1	xx	0	0	000
beq	000100	xx	x	x	0	xx	00	1	0	0	0	0	0	xx	0	0	001
bne	000101	xx	x	x	0	xx	00	0	1	0	0	0	0	xx	0	0	001
blez	000111	xx	x	x	0	xx	00	0	0	1	0	0	0	xx	0	0	001
bltz	000001	xx	x	x	0	xx	00	0	0	0	1	0	0	xx	0	0	001
bgtz	000110	xx	x	x	0	xx	00	0	0	0	0	1	0	xx	0	0	001
addi	001000	xx	x	x	1	00	01	0	0	0	0	0	0	00	0	0	000
addiu	001001	xx	x	x	1	00	01	0	0	0	0	0	0	00	0	0	000
j	000010	xx	x	x	0	xx	xx	x	x	x	x	x	0	xx	1	0	xxx
jal	000011	xx	x	x	1	10	xx	x	x	x	x	x	0	xx	1	1	xxx
andi	001100	xx	x	x	1	00	10	0	0	0	0	0	0	00	0	0	010
ori	001101	xx	x	x	1	00	10	0	0	0	0	0	0	00	0	0	011
xori	001110	xx	x	x	1	00	10	0	0	0	0	0	0	00	0	0	100
slti	001010	xx	x	x	1	00	01	0	0	0	0	0	0	00	0	0	101
sltiu	001011	xx	x	x	1	00	01	0	0	0	0	0	0	00	0	0	101
lui	001111	xx	x	x	1	00	11	0	0	0	0	0	0	00	0	0	000
lb	100000	xx	0	0	1	00	01	0	0	0	0	0	0	11	0	0	000
lbu	100100	xx	1	0	1	00	01	0	0	0	0	0	0	11	0	0	000
lh	100001	xx	0	0	1	00	01	0	0	0	0	0	0	10	0	0	000
lhu	100101	xx	0	1	1	00	01	0	0	0	0	0	0	10	0	0	000
sb	101000	00	x	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000
sh	101001	01	x	x	0	xx	01	0	0	0	0	0	1	xx	0	0	000

write and **3-bit ALUop** signals as shown in table 3. The meanings of ALUop signals are giving in table 4.

Table 4. ALUop meaning

Aluop	Meaning
000	Add
001	Sub
010	And
011	Or
100	Xor
101	Slt
110	Look at funct field
111	N/a

ALUop signals with **funct (instr 5:0)** field of instruction are used by the R-type control to produce **ALUcontrol (5:0)** signals and several signals that are necessary in the execution of R-type instructions. Table 5 shows the truth table of R-type control.

Table 5. R-type control truth table.

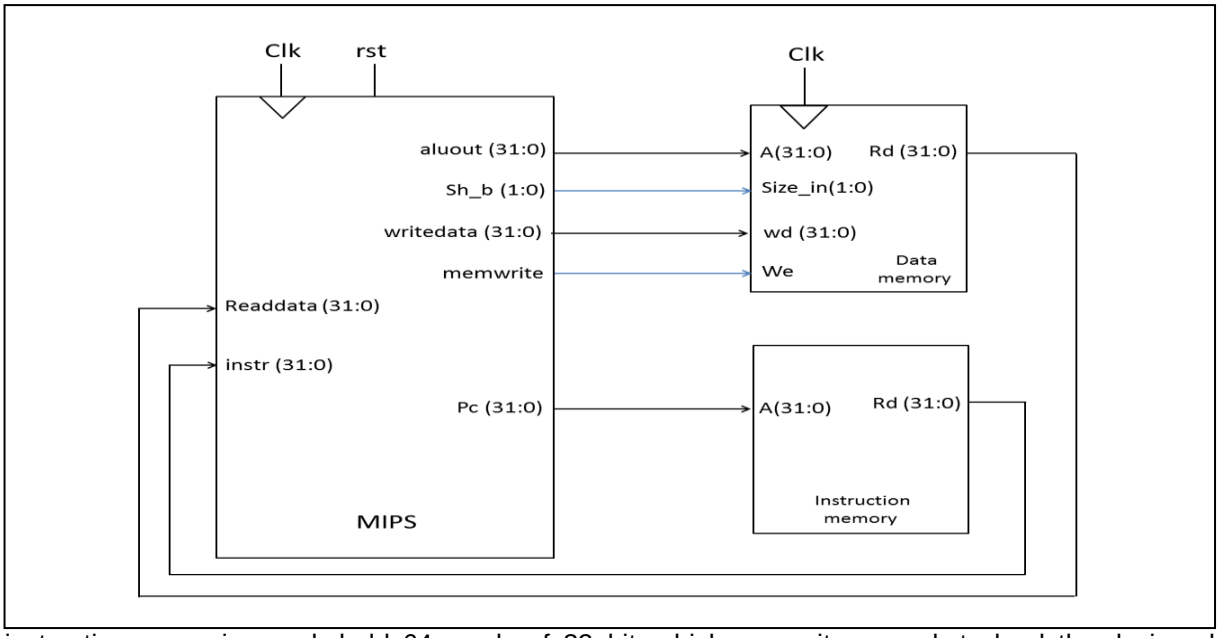
Aluop	Funct	Alucontrol	Jr	Jalr	div	Mult	Sign	Mthi	Mtlo	Mfhi	Mflo
000	xxxxxx	000010 (add)	0	0	0	0	0	0	0	0	0
001	xxxxxx	010010 (sub)	0	0	0	0	0	0	0	0	0
010	xxxxxx	000000 (and)	0	0	0	0	0	0	0	0	0
011	xxxxxx	000001 (or)	0	0	0	0	0	0	0	0	0
100	xxxxxx	000101 (xor)	0	0	0	0	0	0	0	0	0
101	xxxxxx	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	100000	000010 (add)	0	0	0	0	0	0	0	0	0
11x	100001	000010 (add)	0	0	0	0	0	0	0	0	0
11x	100010	010010 (sub)	0	0	0	0	0	0	0	0	0
11x	100011	010010 (sub)	0	0	0	0	0	0	0	0	0
11x	100100	000000 (and)	0	0	0	0	0	0	0	0	0
11x	100101	000001 (or)	0	0	0	0	0	0	0	0	0
11x	100110	000101 (xor)	0	0	0	0	0	0	0	0	0
11x	100111	000110 (nor)	0	0	0	0	0	0	0	0	0
11x	101010	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	101011	010011 (slt)	0	0	0	0	0	0	0	0	0
11x	000000	000100 (sll)	0	0	0	0	0	0	0	0	0
11x	000010	000111 (srl)	0	0	0	0	0	0	0	0	0
11x	000011	001000 (sra)	0	0	0	0	0	0	0	0	0
11x	000100	100100 (sllv)	0	0	0	0	0	0	0	0	0
11x	000110	100111 (srlv)	0	0	0	0	0	0	0	0	0
11x	000111	101000 (srav)	0	0	0	0	0	0	0	0	0
11x	001000	000010 (jr)	1	0	0	0	0	0	0	0	0
11x	001001	000010 (jalr)	1	1	0	0	0	0	0	0	0
11x	011000	000000 (mult)	0	0	0	1	1	0	0	0	0
11x	011001	000000 (multu)	0	0	0	1	0	0	0	0	0
11x	011010	000000 (div)	0	0	1	0	1	0	0	0	0
11x	011011	000000 (divu)	0	0	1	0	0	0	0	0	0
11x	010001	000000 (mthi)	0	0	0	0	0	1	0	0	0
11x	010011	000000 (mtlo)	0	0	0	0	0	0	1	0	0
11x	010000	000000 (mfhi)	0	0	0	0	0	0	0	1	0
11x	010010	000000 (mflo)	0	0	0	0	0	0	0	0	1

3.3 VHDL TOP-LEVEL IMPLEMENTATION

In VHDL (Very high speed IC Hardware Description Language), each element in the datapath and control unit is composed as component then these components are combined to form the single cycle MIPS (Microprocessor without Interlocked Pipeline Stages) processor. This processor is connected to

external separate instruction and data memories through the data and address busses and this shows how the MIPS can communicate with the outside world.

Fig. 4 demonstrates MIPS interfacing with data and instruction memories. In this work, data and



instruction memories each hold 64 words of 32_bit, which are quite enough to load the designed programs, later a testbench is written and used to execute a program.

4 RESULTS AND DISCUSSION

The program shown in Fig. 5 is stored in the instruction memory, this program uses a procedure to perform multiplication between x"fffffffe" and x"00000004" and it should produce x"00000003 fffffff8" if all of the instructions executed properly.

```
# If successful, it should write the values x"00000003" to address 84  and x"fffffff8" to address 80
#              Assembly              Discribtion              Address    Machine
 main:         lui $a0, 0xffff       # $a0 = 0xffff            0         3c04ffff
               ori $a0, $a0, 0xfffe  # $a0 = 0xfffffffe        4         3484fffe
               addi $a1, $0, 4       # $a1 = 4                 8         20050004
               jal multiplicate      # should be taken         c         0c000007
               sw $v0, 0x80($0)      #write adr 80 = 0xfffffff8 10       ac020080
               sw $v1, 0x84($0)      #write adr 84 = 3         14        ac030084
multiplicate:  multu $a0,$a1         # hi = 3                  18        00850019
                                     # lo = 0xfffffff8
               mfhi $v1              # $v1 = 3                 1c        00001810
               mflo $v0              # $v0 = 0xfffffff8        20        00001012
               jr $ra                #should be taken         24        03e00008
```

Fig. 5. Assembly and machine code for MIPS test program

The program will write the values x"00000003" to address 84 and x"fffffff8" to address 80 if it runs correctly. But not getting the desired result meaning incorrect hardware design of MIPS (Microprocessor without Interlocked Pipeline Stages).

By using the earlier constructed top-level, memories and testbench, the program shown in Fig. 5 is executed and the results are shown in Fig. 6. As long as **memwrite** signal is **1**, the results are stored at data memory. Where x"00000003" is stored at address 84 and x"fffffff8" is stored at address 80.

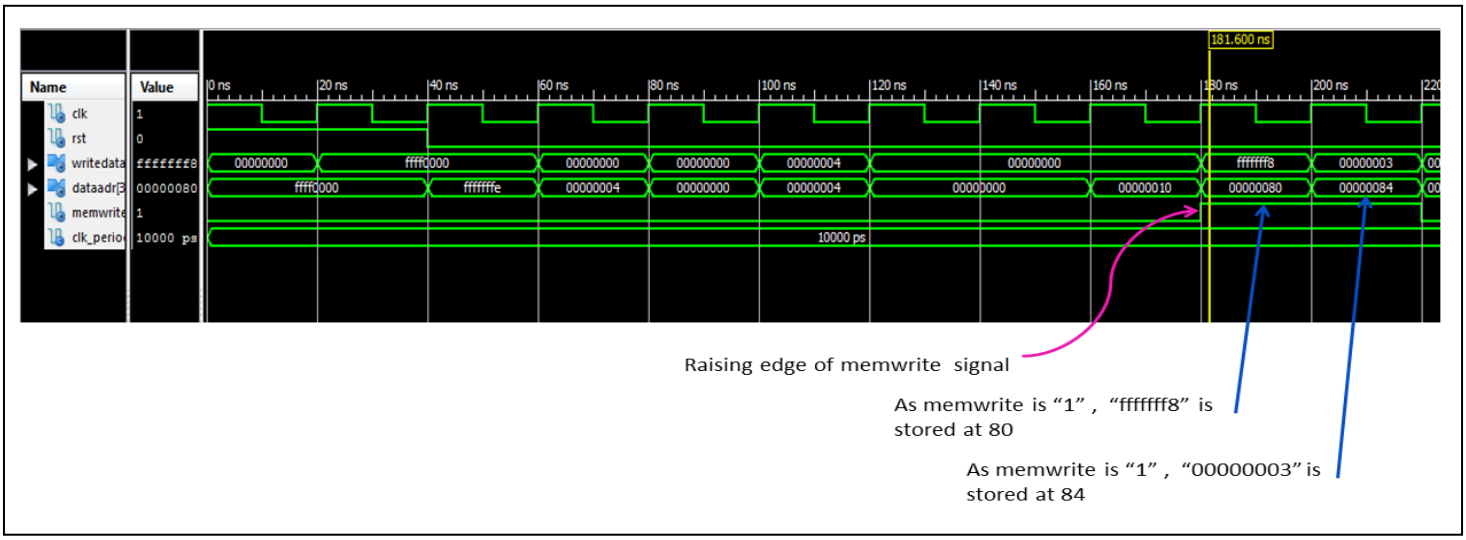


Fig. 6. Simulation of program's execution

5 CONCLUSION

The aim of this research is to implement the complete design of a 32-bit, single cycle MIPS (Microprocessor without Interlocked Pipeline Stages) processor using VHDL (Very high speed IC Hardware Description Language). After completing the design, various programs were simulated using **(Xilinx ISE Design Suite 13.4)** program and the desired results were obtained which indicate the correctness of the design.

Final design which has the ability to execute all I-type, J-type and R-type instructions including mult, multu, div, divu, mfhi, mflo, mthi, and mtlo instructions has been implemented on a **SPARTAN 3AN (XC3S700AN)** starter kit and the desired results have been gotten.

6 REFERENCES

[1] Pedroni V., "*circuit design with VHDL"*, MIT Press, London, England, 2004.

[2] Hennessy J., Patterson D., "*Computer Architecture: A Quantitative Approach",* Morgan Kaufmann, San Francisco, USA, 2007.

[3] Reaz M., et al., "Single Core Hardware Modelling Of 32-Bit MIPS RISC Processor With A Single Clock ",Research Journal Of Applied Sciences, Engineering And Technology, Vol.4, No.7 , pp.825-832, 2012.

[4] Anjana R., Krunal G., " VHDL Implementation of a MIPS RISC Processor", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, No.8, pp.83-88, 2012.

[5] Robio V., "*A FPGA Implementation of A MIPS RISC Processor for Computer Architecture Education",* MSc. Thesis, New Mexico State University, Las Cruses, New Mexico, America, 2004.

[6] Bühler M., Baitinger U., "Vhdl-Based Development of a 32-Bit Risc Processor for Educational Purposes", the 9th IEEE Mediterranean Electrotechnical Conference (Melecon), Tel-Aviv, Israel, PP.138 – 142, 1998.

[7] Anthony I., "*VHDL Implementation of Pipelined DLX Microprocessor"*, MSc. Thesis, University Teknologi Malaysia (UTM), Malaysia, 2008.

[8] Hennessy J., Patterson D., "*Computer Organization and Design: The Hardware/Software Interface",* Morgan Kaufmann, Waltham, USA, 2012.