VHDL PROTOTYPING OF A 5-STAGES PIPELINED RISC PROCESSOR FOR EDUCATIONAL PURPOSES

Conference Paper · February 2014

CITATION
1

READS
2,143

2 authors:

Safaa Omran
middle technical university
70 PUBLICATIONS **181** CITATIONS
SEE PROFILE

Hadeel Shakir Mahmood
Middle Technical University
7 PUBLICATIONS **17** CITATIONS
SEE PROFILE

VHDL PROTOTYPING OF A 5-STAGES PIPELINED RISC PROCESSOR FOR EDUCATIONAL PURPOSES

Safaa S. Omran and Hadeel Sh. Mahmood
Computer Engineering Techniques
College of Electrical and Electronic Techniques
Baghdad, Iraq
E-mail: omran_safaa@ymail.com , hadeel_shakir@yahoo.com

KEYWORDS
VHDL, Pipeline, MIPS, Data Hazard, Control Hazard, FPGA.

ABSTRACT

This paper describes the **VHDL** (**V**ery High Speed IC **H**ardware **D**escription **L**anguage) implementation of a complete 5-stages, 32-bit, pipelined **MIPS** (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages) processor with integer multiplication and division support. The processor design supports all the 50 integer instructions including 25 R-type, 9 J-type and 16 I-type instructions. At the beginning, the processor's complete design is divided into three units: the pipelined datapath unit, the pipelined control unit and the hazard unit which solves all the problems of data hazards and control hazards. Then a program that finds the factorial of number 6 is executed and results are discussed. The **VHDL** design of the complete pipelined **MIPS** processor is implemented by using (*Xilinx ISE Design Suite 13.4*) program and configured on *Xilinx Spartan-3AN* **FPGA** (**F**ield **P**rogrammable **G**ate **A**rray) starter kit.

INTRODUCTION

In a single-cycle MIPS processor, an entire instruction can be executed in one cycle and the cycle time is limited by the slowest instruction (Omran and Mahmood, 2013).
Pipelining is an implementation technique in which multiple instructions are overlapped in execution (Hennessy and Patterson, 2012). Therefore, pipelining is achieved by subdividing the single-cycle MIPS processor into several parts. Each part is called a stage. Such that, in the 5-stages pipelined MIPS, five instructions can be executed simultaneously, one in each stage (Linder and Schmid, 2007). Each stage takes a single clock cycle, so under ideal conditions, the speed up from pipelining is approximately equal to the number of pipeline stages (Hennessy and Patterson, 2012).
The VHDL design of a basic 5-stages, 32-bit, pipelined MIPS processor has been made by several previous researches which implement the simple design that can execute the basic instructions (*add*, *sub*, *lw*, *sw*, *and*, *or*, *beq*, *bne* and *j*) (Robio, 2004. Singh and Parmar, 2012).
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 (**A**pplication **S**pecific **I**ntegrated **C**ircuit) (Pedroni, 2004. Perry, 2002), the MIPS-32 compatible CPU (**C**entral **P**rocessing **U**nit) was designed, tested, and synthesized using VHDL.

INSTRUCTION SET FOR PIPELINING

Designing instruction set for pipelining required:
First, fetching instructions in the first pipeline stage and decoding them in the second stage required that all MIPS instructions to be of the same length.
Second, reading the register file in the second stage at the same time that the hardware is determining what type of instruction was fetched required that MIPS to have only a few instruction formats, with the source register fields being located in the same place in each instruction.
Third, calculating the memory address in the execute stage and then accessed in the following stage required that memory operands only appear in loads or stores in MIPS.
Fourth, operands must be aligned in memory (Hennessy and Patterson, 2012).

PIPELINE HAZARDS

Pipelined processor may suffer from hazards. Hazards occur when the next instruction cannot execute in the following clock cycle. There are three different types of hazards:

1. **Structural hazard:** occurs when the hardware cannot support the combination of instructions that required to execute in the same clock cycle [Alekya. 2011].
2. **Data hazard:** occurs when an instruction tries to read a register that has not yet been written back by a previous instruction. Data hazards are solved with forwarding (bypassing) and stalls.
3. **Control hazard:** occurs when the decision of what instruction to fetch next has not been made by the time the fetch takes place. Control hazards are solved with either stalling until the branch is complete which is too slow or branch prediction (static or dynamic) and flushes, or delaying slot.

IMPLEMENTATION

Figure 1 shows the complete design of the pipelined MIPS processor which consists of:

A. **Pipelined datapath unit.**
B. **Pipelined control unit.**
C. **Hazard unit.**

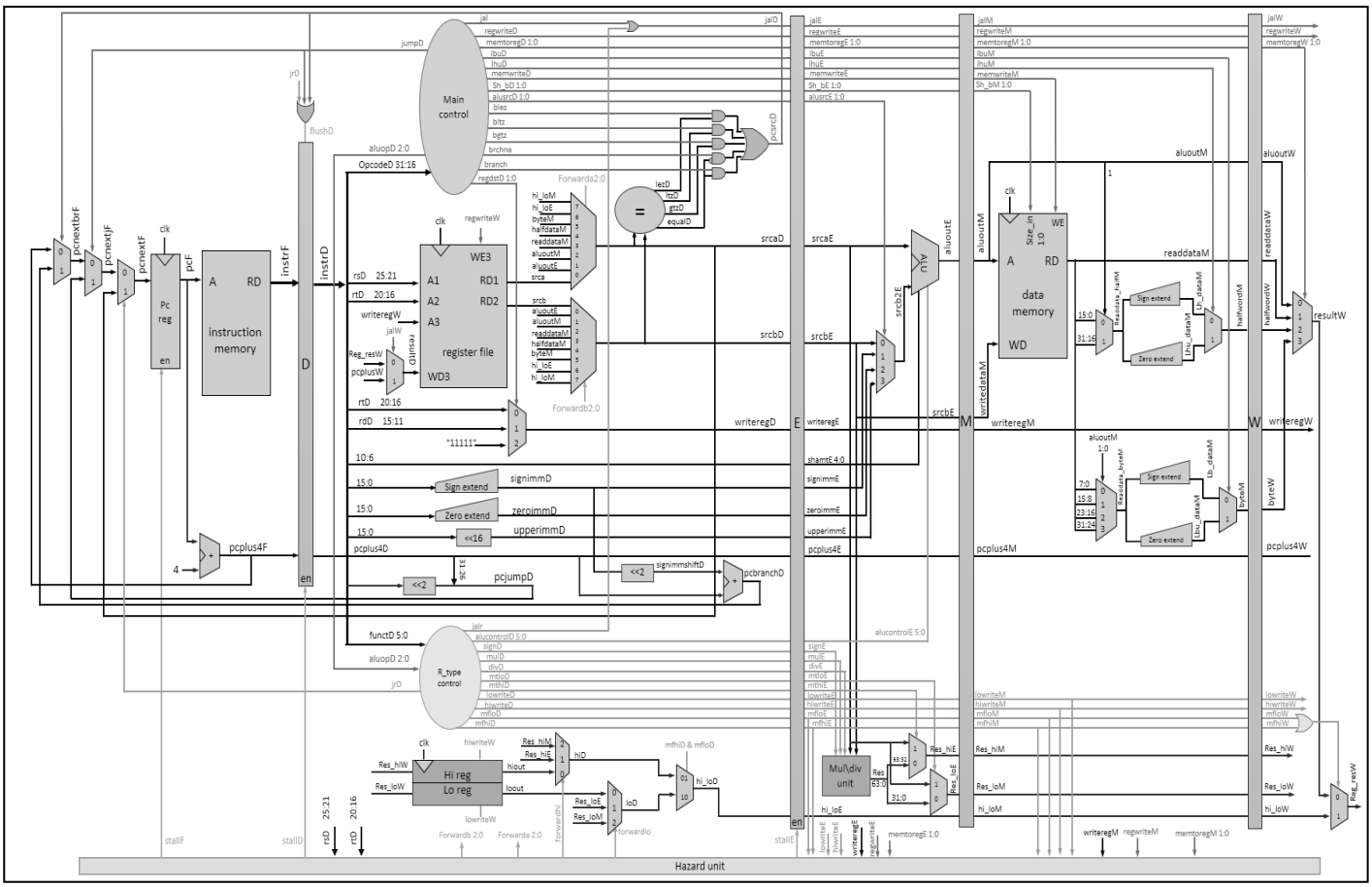

Figure 1: Complete Design of a Pipelined MIPS

After adding necessary electronic circuitry for each instruction execution, the final MIPS processor design supports all the 50 integer instructions.

Pipelined Datapath Unit

The biggest delays in the single-cycle processor are caused by reading and writing the memories and register file and using the ALU (**A**rithmetic / **L**ogic **U**nit) [1]. Therefore, to form the 5-stages pipelined processor, the single-cycle datapath must be chopped into five stages so that each stage performs exactly one of these slow steps.

Whenever a stage has finished its task on an instruction, it will pass it for the next stage and receive the following instruction. So pipeline registers are required to save the results of each stage. These registers are represented in Fig. 1 by long, dark green bars labeled as: **D, E, M,** and **W**.
These five stages are:

1. Fetch stage (**F**): in this stage, the processor uses the address of the instruction to execute contained in the PC (**P**rogram **C**ounter) register to read this instruction from instruction memory, and computes the address of the next instruction by incrementing the PC value by 4. This stage must contain the following components:
   - A 32-bit PC register.
   - A 32-bit word-addressable instruction memory.
   - An adder to increase PC by 4.
   - Multiplexers for selecting either branch address or jump address.
2. Decode stage (**D**): in this stage, the processor decodes the instruction to produce the control signals and reads the source operands from the register file. This stage must contain the following components:
   - A register file which consists of 32-bit * 32 registers.
   - A sign extender.
   - A zero extender.
   - Equal unit that makes the decision of branch instructions.
   - An adder and a 2-bit shifter to compute the target address in the case of jump and branch instructions.
   - Two 32- bit registers (*Lo* and *Hi*) to hold the results of **mul** and **div** instructions.
   - Multiplexers are used to select one input from several inputs and pass it to the output. Select line is controlled by a signal provided by the control unit.
3. Execute stage (**EXE**): in this stage, the processor uses either the ALU or the Mul/Div unit to perform the required computations which may be the calculation of values of two registers or address calculation for **lw** or **sw** instructions. EXE stage contains:
   - An ALU, which performs arithmetic and logic operations as described in table 1.
   - Mul/Div unit, which performs integer multiplication and division operations.
   - Multiplexers.
4. Memory stage (**MEM**): in this stage, the processor reads or writes data memory so only **lw** and **sw** instructions use this stage. MEM stage contains:

- A byte-addressable data memory.
- Sign and zero extenders.
- Multiplexers.

5. Write back stage (**WB**): in this stage, the processor writes the results back to the register file when needed. It only contains:
   - Multiplexors to choose which value to be written back to register file.

Table 1: Functions of 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 and 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

As noticed, the register file must be read in decode stage and written in write back stage within a single cycle, and this is possible only if the register file is written in the first part of a cycle and read in the second part as bellow:

```
if (clk'event and clk='0') then
    if (we3='1') then
        reg(CONV_INTEGER(a3)) <= wd3;
    end if;
end if;
```

Pipelined Control Unit

The processor recognizes the operation required by each instruction by examining the *opcode* and *funct* fields of the instruction in the decode stage to produce the control signals. These control signals must be pipelined along with the data so that they remain synchronized with the instruction. The pipelined control unit consists of:

1. Main control: this takes opcode (instrD 31:26) field as inputs and produces **multiplexer select**, **memory write** and **3-bit ALUop** signals as shown in table 2. The meanings of **ALUop** signals were given in table 3.

2. R-Type control: this uses **funct** (instrD 5:0) field of instruction with **ALUop** signals generated by the main decoder to produce **ALUcontrol (5:0)** signals and several signals that are necessary in the execution of R-type instructions. Table 4 shows the truth table of R-type control.

Table 2: Main Decoder 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

Table 3: ALUop Meaning

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

Hazard Unit

Like any pipelined processor, this pipelined MIPS design suffers from three types of hazards:

1. **Structural hazards.**
2. **Data hazards.**
3. **Control hazards.**

Structural hazard happen when instructions compete for the same hardware resource. Memory is accessed by fetch (F) stage and memory (MEM) stage at the same clock cycle which led to a structural hazard. In this design, this hazard is solved by using two memories, one for data and the other for instructions.

During a program execution, some instructions will depend on the results if an instruction that did not have finished yet so this will lead to data hazards. Fig. 2 and Fig. 3 give brief ideas on how all data hazards are solved in this design:

- At clock cycle $\mathbf{t}_4$, if instruction $\mathbf{I}_4$ depends on results of instruction $\mathbf{I}_1$, no data hazard will happen. Since $\mathbf{I}_1$ has already reached the write back (**WB**) stage and written its result in the register file at the negative clock edge as shown in figure 2
- At clock cycle $\mathbf{t}_3$, if instruction $\mathbf{I}_3$ depends on results of instruction $\mathbf{I}_1$, forwarding will be possible. Since $\mathbf{I}_1$ has already reached the memory **(MEM)** stage and result has been calculated but has not been written back to register file yet as shown in figure 2.

Table 4: R-Type Decoder Truth Table

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

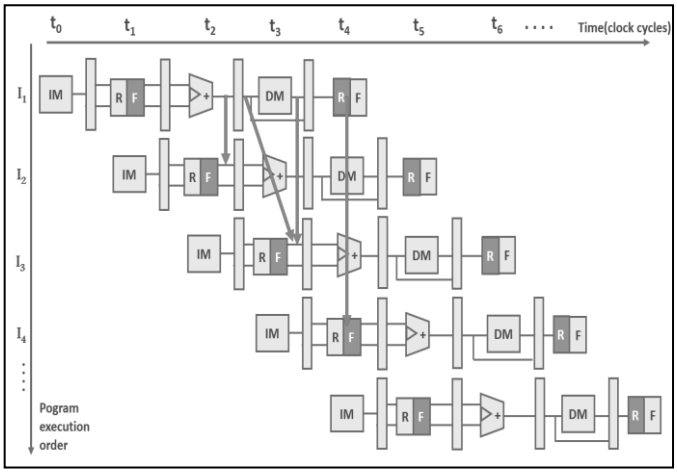

Figure 2: Forwarding to Solve Data Hazard

- At clock cycle t_2, if instruction I_2 depends on results of instruction I_1, two cases should be taken into consideration:

  1. If instruction I_1 result comes from the execute (**EXE**) stage, the result has already calculated and forwarding is possible but the result has not been written back to register file yet as shown in figure 2.

  2. instruction I_1 result comes from the memory (**MEM**) stage, the result has not been ready yet. The only solution is to stall the pipeline, holding up operation until the data is available as shown in figure 3.

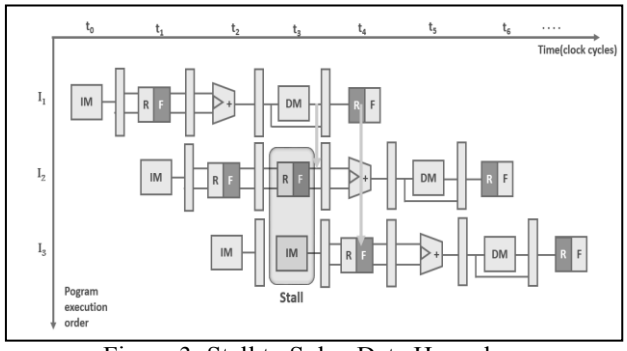

Figure 3: Stall to Solve Data Hazard.

The VHDL stall equations are:

```
stallF <= '0' when ((memtoregE = "01" or memtoregE = "10"
          or memtoregE = "11") and ((regwriteE ='1') and
          ((writeregE = rsD) or (writeregE = rtD))))    else '1';
stallD <= stallF;
```

The hazard unit shown in figure 4 along with four forwarding multiplexers are added to the pipelined processor so that it can solve data hazards with forwarding and stalls. The hazard unit generates control signals for the forwarding multiplexers as explained in tables 5 and 6.

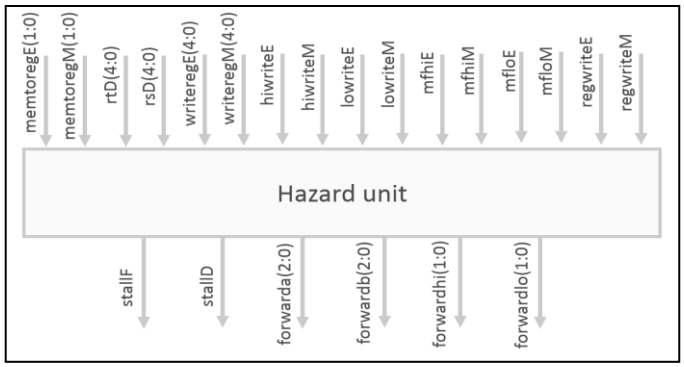

Figure 4: Hazard Unit

Control hazard happens when branches are needed. Depending on the branch decision that is made in the decode stage the processor does not know whether to fetch the following instruction or to take the branch. To reduce this hazard effect an always not taken static branch prediction is used in this design.
When a branch is executed, it is first fetched from instruction memory and then passed to the decode stage where the branch decision is made, at the same time, the following instruction is fetched. If decision is made and branch must be taken, the decode pipeline register is flushed and the instruction at the target address is executed otherwise continue executing the program in order.

Table 5: Hazard Unit Truth Table

Data source	Data destination	Condition	inputs							outputs	
			regwriteE	regwriteM	MemtoregM	mfhiE	mfloE	mfhiM	mfloM	forwarda	forwardb
Execute stage (EXE)	Decode stage (D)	(rsD ≠ "00000") and (rsD = writeregE)	1	x	xx	0	0	x	x	001	xxx
			1	x	xx	1	0	x	x	110	xxx
			1	x	xx	0	1	x	x	110	xxx
			1	x	xx	1	1	x	x	110	xxx
memory stage (MEM)	Decode stage (D)	(rsD ≠ "00000") and (rsD=writeregM)	x	1	00	x	x	0	0	010	xxx
			x	1	01	x	x	0	0	011	xxx
			x	1	10	x	x	0	0	100	xxx
			x	1	11	x	x	0	0	101	xxx
			x	1	00	x	x	1	0	111	xxx
			x	1	00	x	x	0	1	111	xxx
			x	1	00	x	x	1	1	111	xxx
Execute stage (EXE)	Decode stage (D)	(rsD ≠ "00000") and (rsD = writeregE)	1	x	xx	0	0	x	x	xxx	001
			1	x	xx	1	0	x	x	xxx	110
			1	x	xx	0	1	x	x	xxx	110
			1	x	xx	1	1	x	x	xxx	110
memory stage (MEM)	Decode stage (D)	(rsD ≠ "00000") and (rsD=writeregM)	x	1	00	x	x	0	0	xxx	010
			x	1	01	x	x	0	0	xxx	011
			x	1	10	x	x	0	0	xxx	100
			x	1	11	x	x	0	0	xxx	101
			x	1	00	x	x	1	0	xxx	111
			x	1	00	x	x	0	1	xxx	111
			x	1	00	x	x	1	1	xxx	111

The VHDL code for the flush signal is:

flushD <= pcsrcD or jumpD or jrD;

Table 6: Forward Signals for Mul/Div Unit

Data source	Data destination	hiwriteE	lowriteE	hiwriteM	lowriteM	forwardhi	forwardlo
Execute stage (EXE)	Decde stage (D)	1	x	x	x	01	xx
		x	1	x	x	xx	01
memory stage (MEM)	Decde stage (D)	0	x	1	x	10	xx
		x	0	x	1	xx	10

RESULTS

Figure 5 shows a program that finds the factorial of number 6. This program should write (0)h to memory location (84)h and (2d0)h to memory location (80)h if all instructions run correctly, if not, it means the VHDL design of the 5-stages, pipelined MIPS processor is incorrect.

After executing the test program by the 5-stage, pipelined MIPS processor, results that shown in figure 6 have been gotten. These results indicate the correctness of program execution which in turn reflects the correctness of the processor hardware design.

After that, a comparison between the single-cycle MIPS and the 5-stages, pipelined MIPS is made in terms of performance, speedup and FPGA utilization area as shown in table 7. The CPI (Clock Per Instruction) metric is calculated by using equation:

Program Execution time = Instruction count × CPI × Clock period

Where instruction count in the test procedure is 90 instructions.

If successful, the value 0x2d0 is written to address 0x80 and 0x0 to address 0x84

	Assembly	Address	Description	Machine
Main:	addi $a0,$0,6	0	[$a0] = 0x6	20040006
	addi $sp,$0,0xfc	4	[$sp] = 0xfc	201d00fc
	jal factorial	8	should be taken, [$ra] = pc + 4	0c000006
	sw $v0,0x80($0)	c	[0x80] = [$v0] = 0x18	ac020080
	sw $v1,0x84($0)	10	[0x84] = [$v1] = 0x0	ac030084
Factorial:	addi $sp,$sp,-8	14	[$sp] = [$sp] - 0x8	23bdfff8
	sw $a0,4($sp)	18	[$sp + 4] = [$a]	afa40004
	sw $ra,0($sp)	1c	[$sp + 0] = [$ra]	afbf0000
	addi $t0,$0,2	20	[$t0] = 0x2	20080002
	slt $t0,$a0,$t0	24	[$a0] < [$t0] ? [$t0] = 1 : [$t0] = 0	0088402a
	beq $t0,$0,else	28	If [$t0] = 0 go to else	11000003
	addi $v0,$0,1	2c	[$v0] = 0x1	20020001
	addi $sp,$sp,8	30	[$sp] = [$sp] + 8	23bd0008
	jr $ra	34	should be taken, pc = [$ra]	03e00008
else:	addi $a0,$a0,-1	38	[$a0] = [$a0] - 1	2084ffff
	jal factorial	3c	should be taken, [$ra] = pc + 4	0c000006
	lw $ra,0($sp)	40	[$ra] = [$sp+0]	8fbf0000
	lw $a0,4($sp)	44	[$a0] = [$sp+4]	8fa40004
	addi $sp,$sp,8	48	[$sp] = [$sp] + 8	23bd0008
	mult $v0,$a0	4c	{ [hi] , [lo] } = [$v0] × [$a0]	00440018
	mfhi $v1	50	[$v1] = [hi]	00001810
	mflo $v0	54	[$v0] = [lo]	00001012
	jr $ra	58	should be taken, pc = [$ra]	03e00008

Figure 5: Test Procedure

Table 7: Comparison between Single-Cycle MIPS and Pipelined MIPS

processor	Program execution time	No. of clock cycles	Clock period	CPI	Speedup	FPGA area (Number of Slice Registers)
Single-cycle MIPS	1800 ns	90	20 ns	1	1	2,208
Pipelined MIPS	1125 ns	112.5	10 ns	1.25	1.6	2,946

This design is configured on *Xilinx Spartan-3AN* FPGA (**F**ield **P**rogrammable **G**ate **A**rray) starter kit and the results are shown in figure 7.

CONCLUSION

In this research, a 5-stages, 32-bit, pipelined MIPS processor has been designed in VHDL and synthesized using (Xilinx ISE design suite 13.4) program. This design consists of five pipeline stages which are: Fetch stage, Decode stage, Execute stage, Memory stage and Write back stage. It also solves all the problems of data hazard with forwarding and stalls and solves all the control hazard problems using always not taken static branch prediction with flushes.

After completing the design, several programs were executed and the desired results were obtained which indicate the processor ability to execute the designed 49 instructions.

Finally, it is recommended to adopt this research by universities to be used in a computer architecture course where students will be able to define, design, implement, and debug a complete processor, or a processor with specific instructions to be used for a dedicated purposes.

Figure 6: Test Program Execution

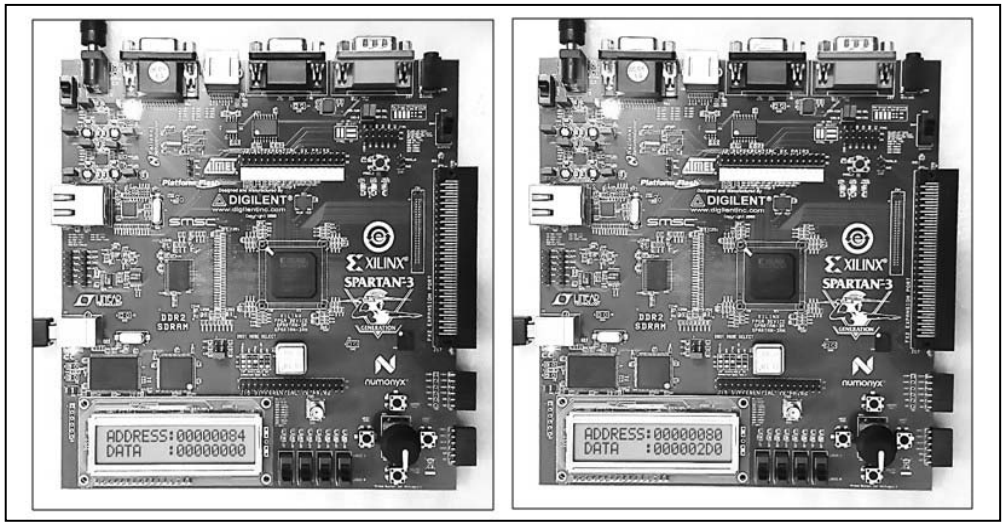

Figure 7: Results on SPARTAN-3AN

BIOGRAPHIES

SAFAA S OMRAN was born in Baghdad, Iraq. He received the B.Sc. Higher Diploma and M.Sc. in Electrical Engineering from Baghdad University, Iraq, in 1978,1981 and 1983 respectively. Since 1979 he joind the Foundation of Technical Education and worked at different Institutes of Technology. Now, he is an assistant prof. in the College of Elect. & Electronic Techniques, Baghdad, Iraq.

HADEEL MAHMOOD was born in Maysan, Iraq and earned her M.Sc. Degree in Computer Engineering Techniques from the College of Electrical and Electronic Techniques, Baghdad, Iraq, where she worked with asst. prof. Safaa Omran on VHDL Implementation of a RISC Processor. She is currently a lecturer assistant in the College of Electrical and Electronic Techniques, Baghdad, Iraq.

REFERENCES

Alekya, N. and P. G. Kumar. 2011. "Design Of 32-Bit RISC CPU Based On MIPS", *Journal of Global Research in Computer Science*, vol. 2, no. 9, 20-24, Sept.

Hennessy, J. L. and D. A. Patterson. 2012. *Computer Architecture: A Quantitative Approach*. 5th ed., San Francisco, USA: Morgan Kaufmann.

Hennessy, J. L. and D. A. Patterson. 2012. *Computer Organization and Design: The Hardware/Software Interface*, 4th ed., Waltham, USA: Morgan Kaufmann.

Linder, M. and M. Schmid. 2007. "Processor Implementation in VHDL", M. Eng. thesis, University of Ulster, Augsburg, Germany.

Omran, S. S. and H. Sh. Mahmood. 2013. "Hardware modelling of a 32- bit, single cycle RISC processor using VHDL." *ICIT 2013 The 6th International Conference on Information Technology,* Amman, Jordon, paper 632.

Pedroni,V. A. 2004. *Circuit Design with VHDL*, London, England: MIT Press.

Perry, L. 2002. *VHDL: Programming by Example*, 4th ed., America: McGraw-Hill.

Robio, V. 2004. "*A FPGA Implementation of A MIPS RISC Processor for Computer Architecture Education*." M. Eng. thesis, NewMexico State University, Las Cruses, New Mexico, America.

Singh, K. P. and S. Parmar. 2012. "Vhdl Implementation of a MIPS-32 Pipeline Processor." *International Journal of Applied Engineering Research*, vol. 7, No.11, 1952-195.