ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 10(3), DOI: 10.17485/ijst/2017/v10i3/110622 January 2017* ISSN (Online) : 0974-5645

Design and Implementation of a Five Stage Pipelining Architecture Simulator for RiSC-16 Instruction Set

Rashidah F. Olanrewaju[1*], Fawwaz E. Fajingbesi[1], S. B. Junaid[2], Ridzwan Alahudin[1], Farhat Anwar[1] and Bisma Rasool Pampori[3]

[1]Department of Electrical and Computer Engineering, IIUM, Malaysia; frashidah@iium.edu.my, fawwazfajingbesi@yahoo.com, wanxneo89@gmail.com, farhat@iium.edu.my
[2]Department of Electrical and Computer Engineering, Ahmadu Bello University, Zaria; abuyusra@gmail.com
[3]Department of Information Technology, Central University of Kashmir, Srinagar–190015, Jammu and Kashmir, India

Abstract

In modern computing, multitasking is the most favorable aspect. An un-pipelined instruction cycle (fetch-execute cycle) CPU processes instructions one after another increasing duration at lesser speed in completing tasks. With pipelined computer architecture, unprecedented improvement in size and speed are achievable. This work investigates the possibility of a better improvement to computer architecture through understanding the inner workings of instruction pipelining in operating system. A design of a 5 stage pipelined architecture simulator for RiSC-16 processors using Visual Basic programming has been achieved contrary to the common available four stage simulators. The simulator also future two most common pipeline instruction hazards generally missing in most available simulators. Thus, the designed simulator becomes an appropriate tool for understanding the concept of pipelining on a step-by-step visualization based instruction-cycle processors hence facilitating a more efficient design in computer architecture. The simulator has been evaluated based on its closeness to real time pipelined computer architecture and through execution of all 8 basic RiSC-16 instruction set with data dependency and control hazard.

Keywords: Computer Processor & Architecture, Instruction Pipelining, RiSC-16 Simulator

1. Introduction

Computer technology has evolved with various architectures since the birth of the first generation of computers around the 1940s until now and people are always looking for ways to improve the performance of computer[1]. An instruction pipeline is a technique often used in the design of modern microprocessors, microcontrollers and CPUs to increase their instruction throughput per unit time[1,2].

Pipelining is a standard feature in Reduced Instruction Set Computing (RISC) processors analogous to a manufacturing plant assembly line. This is because the processor works on different steps of the instruction at the same time and more instructions can then be executed in a shorter period of time[3]. Pipelining is implemented through RISC processor rather than in Complex Instruction Set Computing (CISC) processor. Pipelining has proved to be more efficient as traditional instruction cycle leads to waste of CPU resources as instructions may include other services such as read/write from/to memory, storage or input/output devices, and CPU becomes idle at this time. This will prolong the latency of an instruction as well as the throughput of a program. As computer systems evolve, greater performances are achieved by taking advantage of improvements in technology, such as faster circuitry and organizational enhancements such as adding instruction pipelining to the processor[1,4–6]. By implementing pipelining, the processing of instructions is overlapped as illustrated in Figure 1, meaning while

* *Author for correspondence*

the first instruction is in the decode stage; the second instruction is fetched. When the first instruction is in the execute stage, the second instruction moves to the decode stage and another instruction, instruction 3 is fetched. The first instruction is complete only after three cycles. Subsequently, an instruction is completed on every cycle.

	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6
Instruction 1	Fetch	Decode	Execute			
Instruction 2		Fetch	Decode	Execute		
Instruction 3			Fetch	Decode	Execute	
Instruction 4				Fetch	Decode	Execute

Figure 1. Fetch-execute cycle with pipelining.

This differs from a non-pipelined system shown in Figure 2 where three cycles are required per instruction.

	cycle 1	cycle 2	cycle 3	cycle 4	cycle 5	cycle 6
Instruction 1	Fetch	Decode	Execute			
Instruction 2				Fetch	Decode	Execute

Figure 2. Un-pipelined fetch-execute cycle.

Note that however beautiful pipelining may sound, there are situations where the next set of instruction cannot execute in the next clock cycle. The situation isusually termed hazards. There are three basic types of hazards which include structural, data and control hazards[7]. Understanding the hazard and how they affect processor operations would increase overall efficiency of such systems hence the need for a simulator design.

As seen in 1997[8] a pipeline simulator for the DLX processor known as WinDLX software was created. It was a MS-Windows (16 bit) based pipeline simulator written in C++. The simulator model and design was on Hennessy-Patterson's DLX at the architectural level. It was intended for educational purpose to help visualize the concept of instruction pipelining.

Other implementation of pipeline simulation was include ModelSim simulator and Xilinx ISE tool software[2]. They divided pipelining into four sub stages such as fetch, decode, execute, store using Verilog HDL to design each stage of pipelining. The technology schematics for each stage were presented and by using this technique, a user could gain better understanding of the internal workings of a processor. However, the user is required to possess special skills in order to simulate pipelining in ModelSim and Xilinx tool.

Similar work on pipeline simulator was using Java programming with a specific focus on student interactivity[9]. They chose RiSC-16 processor because it is simple, complete and has been designed for educational purposes. Their system offered user the ability to define its own programs in assembly language and the ability to see graphically the corresponding internal dynamic behavior of the processor.

Subsequently, other design have future like web-simulation model capable of exploring the state-space defined by a Unified Parallel Model and simulator[7,10]. They stated that the simulator could be used as a calculator, deterministically calculating speedups given input parameters. The Unified Parallel Model and simulator can be used to explore the continuity of parallel speedup possibilities allowing users to explore different computer architectures with hardware support at any or all of five levels of parallelism, from intra-instruction (pipeline) through a distributed n-tier client/server system. The tool developed supports the simulation of various user-configurable architectures and interconnection networks, running a user-configurable and variable workload.

The rest of this work is organized as follows: Section II presents the methodology, Section III results and discussion while Section IV and V are conclusion and references respectively.

2. Methodology

The focus of this work is on a 5-stage pipeline simulator design and implementation for a RiSC-16 instruction set processor. Figures 3 and 4 are examples of five-stage pipeline architecture.

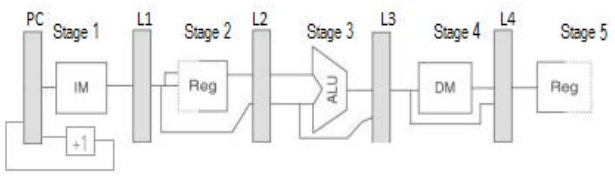

Figure 3. Five stage pipeline model[11].

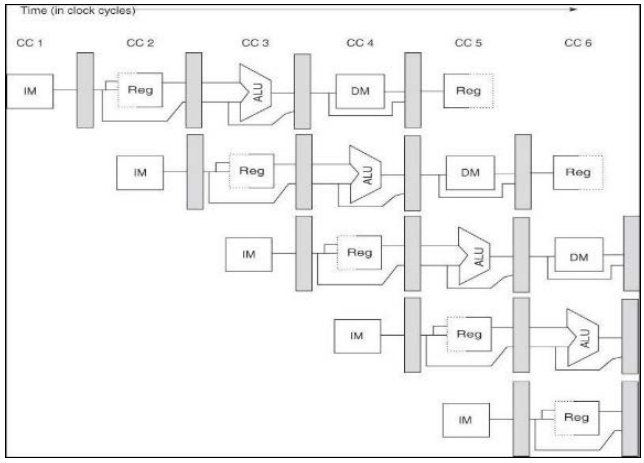

Figure 4. Five stages pipelined[11].

The designed algorithm is simple yet novel as it offers not only user-friendliness but also detail features required for understanding and visualizing the concept of pipelining instruction cycle. Features offered include Data and control Hazards with their solutions, multiple methods for instruction supply and translations making it unique amongst its pairs. The system can be broken down into five stages performing the following functions, which include Instruction fetch (IM), Decode with Register read (REG), Execute (ALU), Data Memory (DM) and Register write (REG).

2.1 Design and Algorithm of Components

Algorithm

For **CLOCK**

For the timer in VB.net which alternates between true and false: positive and negative clock pulses. Boolean variable:

IF clocksig is false then clocksig = true

ELSE IF clocksig is true then clocksig = false

Controlling the five stages and the 5 latches

IFclocksig is true; call all the functions of the five stages and the five latches

For **LATCH**

Five latches represented by set of rich textbox. All of these latches will update their values when there is positive clock pulse. Each latch may contain different numbers of values:

PC: contains program counter value (the address of an instruction that will be fetched). The value will be updated by an increment of a value provided by stage 2.

L1 contains an instruction that has been fetched by IM stage

L2 contains the operand that is to be executed at ALU and the memory access signal

L3 contains the result of the ALU operation together with register address and the memory access

L4 contains a value that will be stored into a register together with the register address

An adder in which the increment is set to one by default controls the value of PC.

While, for L1 to L4, functions latch*() will copy the output from their respective stages at every positive clock pulse where * represents 1-4.

For **STAGES**

FOR Instruction Fetch

Set default output: "0000000000000000"

Only fetch if the PC is less than number of program memory or the program will stop.

Skip stage if the program memory slot at address given by PC contains nothing

Increment PC if an instruction is fetched

FOR Decode &regRead

Set default outputs to zeros

Skip if previous stage has not fetched any instruction

Set default PC increment to 1

Set default multiplexers control signal values (for bypassing data dependencies)

Read the first 3bits from an instruction string as opcode

Get the address of register A (3bits) from an instruction (except for BEQ operation)

Identify the operation of the instruction based on the opcode:

"000" or "010" (add or nand)

"001" or "100" or "101" (addi or sw or lw)

"011" (lui)

"110" (beq)

"111" (jalr)

Check for dependency (*only if user ticked bypassing /forwarding in hazard group box)

Write the meaning of the instruction in "Operation:" rich textbox

IF "000" or "010" (add or nand)

Get the address of register B (3bits) and address of register C (3bits) from instruction code

Get the values of register B and register C

Set output of stage 2:

Output0: "00" (addition) /"01" (Nand)
Output1:regB value in decimal
Output2: regC value in decimal
Output3: regA address
Output4: 0 (no Data Memory Access)

Dependency: Check if there are registers B and C in latch 2 or 3, if yes then set the multiplexer input to the next latch that will be the output of the respective registers. Exception for register with address "0"

IF "001" or "100" or "101" (addi or sw or lw)
Get the address of register B from instruction code
Get the value of signed immediate from the instruction code
Get the value of register B
Set the output of stage 2:
Output0: 00(add)
Output1: regB value in decimal
Output2: signed Imm value in decimal
Output3: regA address/ regA value (for sw operation)
Output4:
= 0 for "001" (addi) operation -no Data Memory stage)
= 1 for "100" (sw) operation-from memory to register)
= -1 for "101" (lw) operation-from register to memory)

*Dependency: Check for register B content in latch 2 or 3, if yes then set the multiplexer input to the next latch that will be the output of the respective register. Exception for register r0

IF "011" (lui)
Get the value of unsigned immediate (10 bits) from instruction code
Set the output of stage 2:
Output0: "10" (and)
Output1: unsigned immediate value in decimal
Output2: &HFFC0
Output3: regA address
Output4: 0 (no Data Memory stage)
IF "110" (beq)
Get the address of register B
Get the value of register B and register A
Get the value of signed Imm

*Dependency: Check if there are registers A and B in latch 2, 3, or 4,

IF True, then replace register(s) value based on statement below (except for register r0):
IF register is in latch 2 then the latest value will be in stage 3 output
IF register is in latch 3 then the latest value will be in latch 3 output
IF register is in latch 4 then the latest value will be in latch 4 output
Perform Branch-if-equal operation:
If Ra = Rb Then 'beq operation (branch if equal), change the PC increment to signed Imm value
Set stage 2 outputs to default.
IF "111" (jalr)
Get the address of register B
Get the value of register B, register A and latch (PC-1)

*Dependency: Check for register B content in latch 2 or 3, if yes then set the multiplexer input to the next latch which will be the output of the respective register. Exception for register

Store the address of the jalr operation (PC-1) to register A and PC and move PC to register B value
Set the output of stage 2 to default.
FOR ALU-perform logic and mathematical operation
Get the operands from the multiplexers
Get the address of register A and operation code from latch 2
Perform operation for the given operation code:
IF 00: operand1 + operand2
IF 01: Not (operand1 and operand2)
IF 10: operand1 and operand2
Set the outputs of stage 3:
Output0: result of the operation
Output1: register A
Output2: Output4 of stage 2.
Data Memory-perform memory access
Get third output of latch 3
Perform operation based on the output as follow:
IF "0" (dummy), Pass the other two outputs to the next latch
Output0 of stage 4 = Output0 of latch 3
Output1 of stage 4 = Output1 of latch 3
IF "1" (get data from memory)
Output0 of stage 4 = dataMemory [ouput0 of latch 3]
Output1 of stage 4 = Output1 of latch 3
IF "-1" (store data to memory)
dataMemory [ouput0 of latch 3] = Output1 of latch 3
Set the output of stage 4 to zeros
FOR RegWrite- write the result into specified register
Get the outputs of latch 4
Store outputs of latch 4 into register r [output1 of latch 4]

For PROGRAM MEMORY

Represent instructions code of 16-bits in binary format

Array of textbox which form 40 textboxes in total
Addresses are in decimal format
16-bit addressing

For DATA MEMORY

Present 16-bits data in binary format
Array of textbox that form 10 textboxes in total
Addresses are in decimal format
16-bit addressing

For REGISTER

Use to store data of 16-bits in binary format
Array of textbox that form 8 textboxes in total
Addresses are in decimal format
16-bit addressing
The register 0 is read-only and contains the null value

For MISCELLANEOUS

"Hazard" group box: allow user to choose whether to use dependency checker or not

"Buttons"

"RUN": activate/deactivate Clock that controls the pipeline system.

"Reset Program": reset latches and stages input/output to default

"Write New": open a new form for user to write set of instructions.

"Load": open a txt file to be loaded into program memory.

"Clear": reset the simulation to default.

"Operation" richtextbox: provide translation of each instruction to the user view

toDecimal() function: convert binary number to decimal format

toBinary() function: convert decimal number to binary

mux1() and mux2() functions: act as multiplexer between latches to provide input to stage 3(ALU) which for dependency hazard solution.

3. Result and Discussion

The designed simulator user interface and data input screenshot are Figures 5and 6 respectively. The simulator evaluation was majorly to test the response to data hazard, control hazard and combination of all 8 basic RiSC-16 Instructions set and its results are shown in Figures 7–12. The operations and code are been given below.

For **Data Hazard evaluation:**

Perform addition of operands:

addi 1, 0, 4 addi 2, 0, 5 add 1, 1, 2 add 3, 2, 1

Translate as:
R1 should contain 4+5 = 9: [1001B]
R3 should contain 9+5 = 13: [1110B]
Data dependency between:
addi 2, 0, 5 and add 1, 1, 2
add 1, 1, 2 and add 3, 2, 1

First solution: stall/nop operation –*increase IPC*
Second solution: bypassing/forwarding

- Implement multiplexers which are control by Decode stage
- Checking dependency by comparing operands' registers with previous store-to register at each latch
- Set the multiplexer to correspond latest value of the operand(s)

For **Control Hazard evaluation**

Branch operation:

addi 1, 0, 2 addi 2, 0, -1 addi 3, 0, 3 beq 1, 0, 4
add 0, 0, 0 add 1, 1, 2 jalr 7, 3 add 0, 0, 0
add 7, 0, 2

Translate as:
r1 = 2, r2 = -1, r3 = 3
While r1 not equal r0,
do r1= r1+r2
End while
r0+r2→ r7

Decrement r1 by one until it equals to zero and finally adds r2 to r7

- Instruction after branch operation will be fetched and executed even if the condition is true or false

To avoid any false execution, branch-delay slot method is applied at compiler and the slots are filled with "nop" operations.

For **Combination of all 8 basic instructions:**

addi 1, 0, 5 addi 2, 0, -3 add 3, 1, 1 add 5, 2, 2
add 3, 3, 1 add 5, 5, 2 add 3, 3, 1 nand 5, 5, 5
add 3, 3, 1 addi 5, 5, 1 add 4, 3, 3 add 6, 5, 5
add 6, 6, 5 add 7, 6, 4 sw 7, 0, 3 lui 1, 77
addi 2, 0, -1 addi 3, 0, 19 lw 4, 0, 3 beq 4, 1, 5
add 0, 0, 0 add 4, 4, 2 jalr 6, 3 add 0, 0, 0
add 0, 0, 0 sw 2, 0, 0.

Translates as:
Perform $2x^2 + 3y^2$ with $x = 5$ and $y = -3$
Store the result in data memory at location 3.
Decrement the result until it equals 64
Then store to R2 value to data memory at location 0

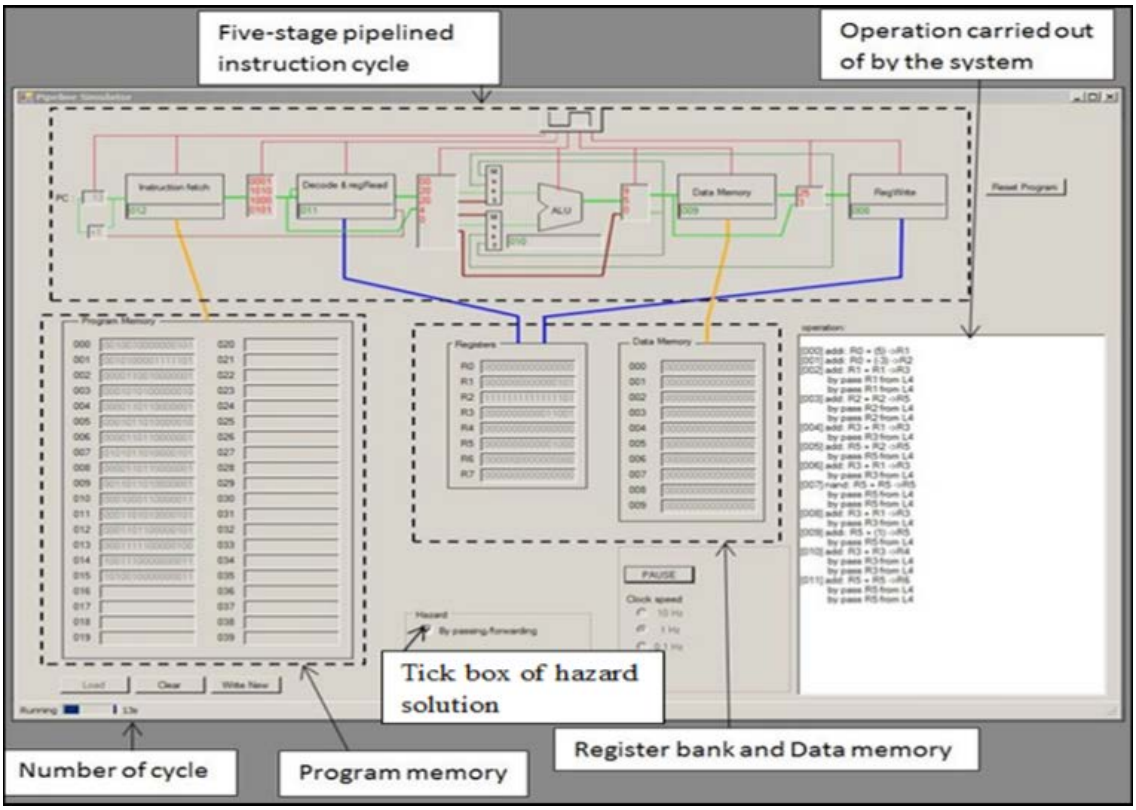

Figure 5. Main user interface of the simulator.

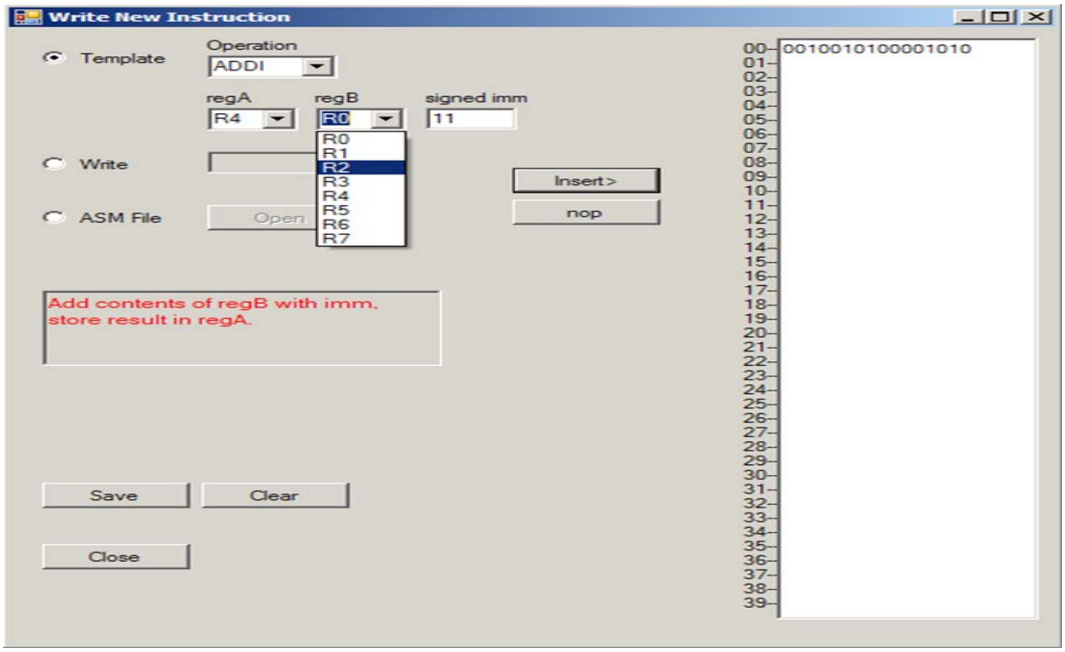

Figure 6. "Write New Instruction" window.

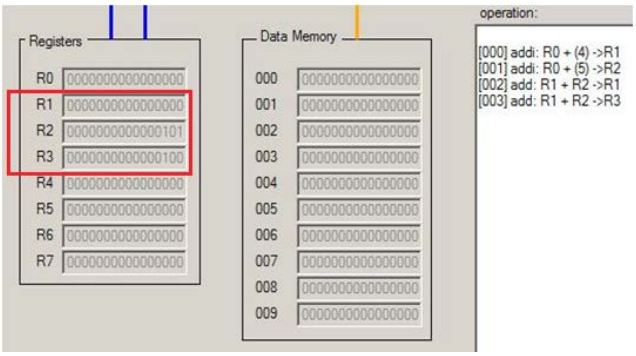

Figure 7. Operation with occurrence of data dependency.

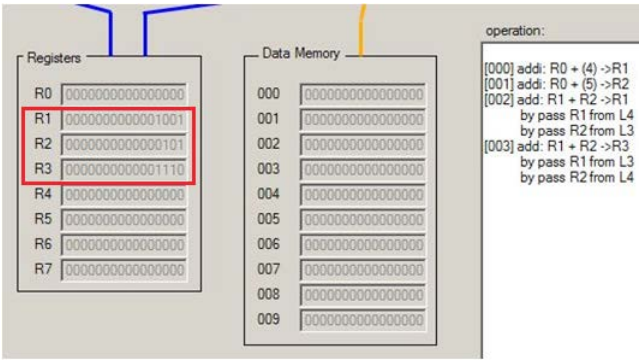

Figure 8. Operation with bypassing method.

Figure 9. Register values before the "beq" operations.

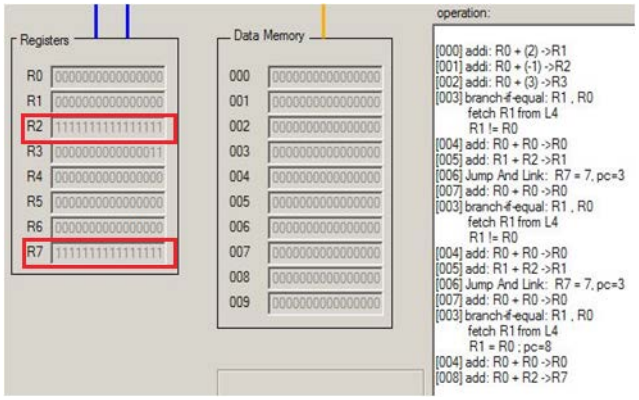

Figure 10. Register values after all operations.

Figure 11. Registers and data memory after computation.

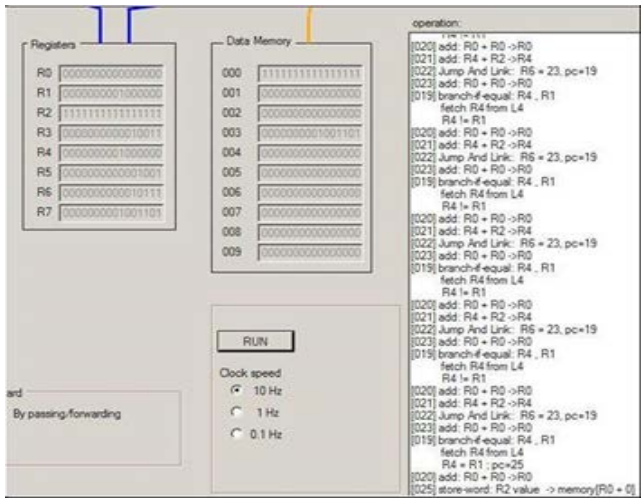

Figure 12. Registers and data memory after completion of all instructions.

4. Conclusion

This research has presented a novel design and implementation for a pipeline instruction set simulator using Visual Basic. The simulator offers visualization hence proper understanding of the concept and processes involved in pipelined instruction cycle for a RiSC-16 processor. It is capable of simulating and solving data dependency and control hazard experienced in real processors which is lacking in most available simulators as ascertained by review; and is however a key component in pipeline concept. The solutions to this hazard are also within the simulator. When users implement bypassing and branch delay techniques to counter hazards, the process can be visualized within the simulator. The designed simulator is also equipped to operate at three

different clock speeds, accept up to forty instructions at once in three different user friendly ways (ASM, Text and Predefined Template), thereby providing its user with the best learning environment for computer architecture and processor pipelining.

5. Acknowledgement

This work was partially supported by Ministry of Higher Education Malaysia (Kementerian Pendidikan Tinggi) grants under numbers FRGS15-254-0495 and RIGS16-084-0248.

Table 1. Literature review summary

Author	Method	Merits and Comment	
Current work	Built using Visual Basic programming for RiSC16 processor architecture	• Simple design concept • User-friendly with no professional skills required as instruction set can be in plain text, ASM or RiSC key word • Five stage pipeline for higher efficiency • Capable of Simulating Data and control Hazards with their solutions	• Fully capable of handling 5 stage pipelined architecture of RiSC-16 Processor • Capable of handling up to 40 instructions • Visualization for better understanding of pipeline concept • Perfect for educational purpose
Rakesh et al.[2]	Built Using Verilog HDL on a predesigned utility simulator (ModelSim) and Xilinx ISE tools	• Modelled at architecture level • Intended for educational purposes • Requires professional skill to operate • Simulates through pre-published software as no dedicated simulator was built from scratch.	• Designed with focus on handling four stage pipelining • No provision for executing and handling hazards which are crucial in understanding and perfecting pipeline concept
Oséeet al.[9]	Built Using Java for RiSC16 processor architecture	• Designed from scratch • Capable of only 4-stage pipelined architecture of RiSC16 processor • Intended for educational purposes	• No provision for executing and handling hazards which are crucial in understanding and perfecting pipeline concept
Hongansonset al.[7,10]	Built a Web based simulation model	• Complex design • Supports multiple architecture • Capable of handling five stage pipelining • Speed calculation capabilities • Intended for educational purposes	• No provision for executing and handling hazards which are crucial in understanding and perfecting pipeline concept
Grünbacheret al.[8]	Built Using C++ programming for DLX processor (WinDlx) in MS-Windows (16 bit)	• Target at pipeline simulator for the DLX processor (WinDlx) • Modelled at architecture level • Intended for educational purpose	• No provision for executing and handling hazards which are crucial in understanding and perfecting pipeline concept

6. References

1. Rakesh MR, Ajeya B, Mohan AR. Novel architecture of 17 bit address RISC CPU with pipelining technique using Xilinx in VLSI Technology. International Journal of Engineering Research and Applications.2014; 4(5):116–121.
2. Rakesh MR.Design and simulation of four stage pipelining architecture using the Verilog. International Journal of Science and Research. 2014; 3(3):108–12.
3. Rana S, Mehra R. Design &simulation of RISC processor using hyper pipelining technique. IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE). 2013; 9(2):49–57.
4. Trivedi P. Design &analysis of 16 bit RISC processor using low power pipelining. 2015 International Conference on Computing, Communication & Automation (ICCCA); 2015:1294–7.
5. Finlayson I, Uh GR, Whalley DB, Tyson G. An overview of static pipelining. IEEE Computer Architecture Letters. 2012; 11(1):17–20.
6. Cheah HY, Fahmy SA, Kapre N. Analysis and optimization of a deeply pipelined FPGA soft processor. 2014 International Conference on Field-Programmable Technology (FPT); 2015. p. 235–8.
7. Hoganson KE. High-performance computer architecture and algorithm simulator. Journal on Educational Resources in Computing. 2002; 2(1):131–48.
8. Grunbacher H. Teaching computer architecture/organisation using simulators. 28th Annual Frontiers in Education Conference, FIE'98. Treitlstrasse Vienna Austria. 1998; 3:1107–12.
9. Osée M, Richard A, Biest AV, Mathys P. Educational simulation of the RiSC processor. International Conference on Engineering Education(ICEE 2007); 2007.
10. Hoganson K. The unified parallel speedup model and simulator. Southeast Regional ACM Conference; 2001. p. 1–23.
11. Balasubramonia R. CS6810 computer architecture. University of Utah: Youtube; 2012.
12. Jacob PB. The pipelined RiSC-16. ENEE 446: Digital Computer Design, Fall 2000; 2000. p. 1–9.