DESIGN OF A FLOATING POINT UNIT FOR 32-BIT 5 STAGE PIPELINE PROCESSOR

BY

LOW WAI HAU

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER ENGINEERING (HONS)

Faculty of Information and Communication Technology
(Kampar Campus)

JAN 2020

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **Design of A Floating Point Unit for 32-Bit 5 Stage Pipeline Processor**

Academic Session: JAN 2020

I **LOW WAI HAU**

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

LOW WAI HAU

(Author's signature)

(Supervisor's signature)

Address:
1059, Jalan Seksyen 1/1,
Bandar Barat,
31900 Kampar.

MOK KAI MING
Supervisor's name

Date: 24 April 2020

Date: 24 April 2020

DESIGN OF A FLOATING POINT UNIT FOR 32-BIT 5 STAGE PIPELINE PROCESSOR

BY

LOW WAI HAU

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER ENGINEERING (HONS)

Faculty of Information and Communication Technology
(Kampar Campus)

JAN 2020

DECLARATION OF ORIGINALITY

I declare that this report entitled "**DESIGN OF A FLOATING POINT UNIT FOR 32-BIT 5 STAGE PIPELINE PROCESSOR**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other reward.

Signature : **LOW WAI HAU**

Name : LOW WAI HAU

Date : 24 APRIL 2020

ACKNOWLEDGEMENTS

First of all, I would like express deepest gratitude to my supervisor, Mr. Mok Kai Ming who has been providing me guidance with patience throughout the planning and development of this project.

I would also like to thank my family members for the support and encouragement throughout my undergraduate years. Nevertheless, I would like to thank all my fellow course mates and friends who supported me throughout the entire course of this project. All the supports and helps contribute to the accomplishment of this project.

ABSTRACT

This project is about the design of a Floating Point Unit (FPU), integrate the FPU into RISC32 processor and synthesize the FPU design on Field Programmable Gate Array (FPGA). The stand-alone FPU has been modeled by a senior student in Universiti Tunku Abdul Rahman, Liu Hing Yun. However, there was no integration test made on the FPU to the processor and the aforesaid FPU can only perform operation on single precision numbers. Hence, this project is required to develop a FPU which can perform operation on both single and double precision numbers.

The development project will start by studying the algorithm of addition on floating point numbers. The addition algorithm is then implemented in the FPU so that the FPU can perform addition on floating point numbers. Also, a dedicated register file is developed for FPU to store 32-bits or 64-bits of data.

This project will use top down design methodology: system specification, architecture level and microarchitecture level development. Microarchitecture level will perform unit partitioning of the system and block partitioning of the units. RTL modelling using Verilog will be performed on each block following the units and eventually the complete system. Verification will be made to determine functionality correctness of FPU.

The project will integrate the FPU into the RISC32 pipeline processor and the verification will be carried out to prove the functionality of FPU. In the end of this project, the FPU will be synthesized on FPGA.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALU	Arithmetic Logic Unit
FPU	Floating Point Unit
CPU	Central Processing Unit
MIPS	Microprocessor without Interlocked Pipelined Stages
VHDL	VHSIC Hardware Description Language
RISC	Reduced Instruction Set Computer
VHDL	VHSIC Hardware Description Language
FPGA	Field Programmable Gate Array

CHAPTER 1: INTRODUCTION

1-1 Background Information

1-1-1 Floating Point Unit

Floating point unit (FPU) was a part of a computer system dedicated to carry out operations on floating point numbers. It could be defined as a specialized coprocessor that could manipulate numbers quicker than the basic microprocessor (CPU) itself. The typical operations on floating point numbers were addition, subtraction, multiplication, division, square root and bit shifting. An ALU was designed to handle the operations on the fixed point numbers such as integers. The operations on fixed point numbers were similar to the operations on floating point numbers. ALU could also carry out the operations on floating point numbers. However, the difference between the ALU and the FPU was their speed on carrying out the operations on floating point numbers. ALU performed the operation on floating point numbers in such a slow way. Therefore, this was the reason for the existence of the FPU coprocessor in the market or integrated with the CPU.

Early years back, personal computing was common in IBM PC or compatible microcomputers for the FPU to be entirely separate from the CPU, and sold as an optional add-on. The FPU could be purchased if the user wished to enhance the processor's speed to achieve math-intensive computation especially on floating point numbers. Starting with the Intel Pentium and Motorola 68000 series in the late 1990s, the FPU became a physical part of the microprocessor chip.

When a CPU was executing a program that called for a floating-point operation, there were three ways to carry it out: Floating-point unit emulator, Add-on FPU and Integrated FPU. FPU could support the following arithmetic operations that is addition, subtraction, multiplication, division and square root. The supported rounding modes for each operation are round to nearest even, round to zero, round up and round down.

1-1-2 MIPS

MIPS also known as Microprocessor without Interlocked Pipelined Stage, which based on the Reduced Instruction Set Computer (RISC) architecture was developed by a team led by John L. Hennessy and David A. Patterson. MIPS Technologies, formerly known as MIPS Computer System Inc. was co-founded at 1984 by John L. Hennessy. The MIPS architecture could be found in the book called Computer Organization and Design: The Hardware/ Software Interface (Patterson and Hennessy, 2005). This book showed the architecture of MIPS, the instruction sets, pinelined stages, just to name a few and how to build a microprocessor. MIPS processors operated by breaking instruction execution into multiple small independent "stages" and since the stages were independent, multiple instructions could be in varying stages of completion at any one time (Integrated Device Technology. Inc, 1994, p.1-2).

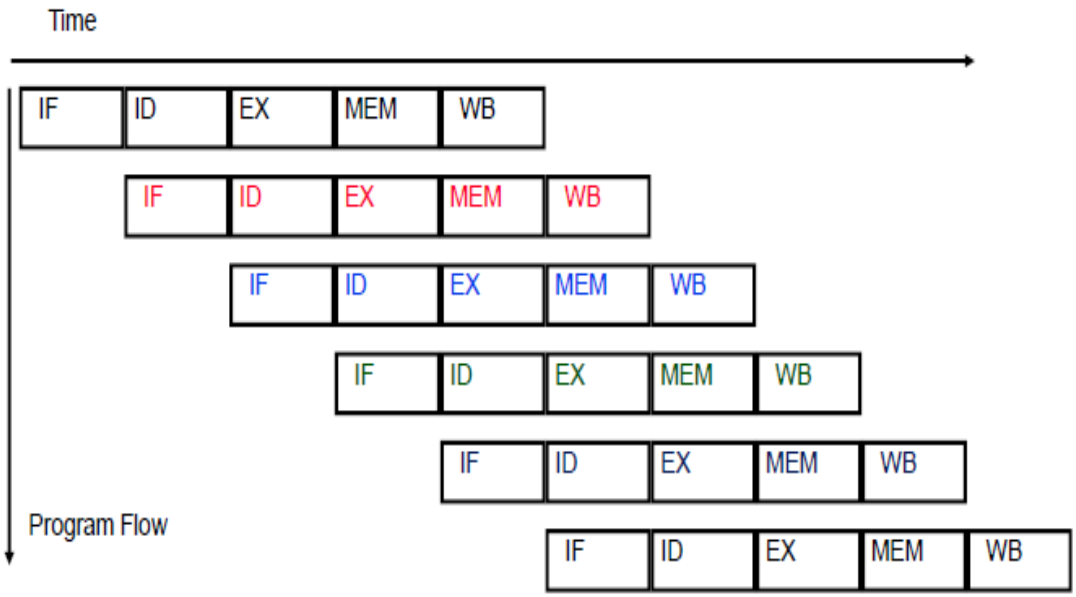


Figure 1-1-2-F1: MIPS 5-stage pipeline (Mok, 2008, p.9).

The instruction execution cycle was divided to 5 stages, IF ("Instruction Fetch"), ID ("Instruction Decode and Registers Fetch"), EX ("Execute"), MEM ("Memory") and WB ("Write Back").

- IF: Instruction Fetch and update PC.
- ID: Decodes the instruction and fetches the contents of any CPU registers it uses.

- EX: Execute R-type, calculate memory address.
- MEM: Read/write the data from/to the Data Memory.
- WB: Write the result data into the register file.

1-2 Motivation and Problem Statement

1-2-1 Motivation

A 32-bit pipelined RISC microprocessor has been developed in Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman (UTAR) using Verilog which is a hardware description language (HDL). The project is based on the Reduced Instruction Set Computing (RISC) architecture. The motivations to initiate the project are due to following reasons:

- Microchip design companies designed microprocessor as Intellectual Property or IP for commercial purpose. The microprocessor IP contains information on the entire design process for the front-end (modeling and verification) to back-end (physical design) integrated circuit (IC) design. These are trade secrets of a company and certainly not made available in the market at an affordable price for research purpose.

- Several freely available microprocessor cores can be found in internet, most of them can be found at OpenCores (http://www.opencores.org/). Unfortunately, these processors do not implement the entire MIPS Instruction Set Architecture (ISA) and lack of comprehensive documentation which is hard to be understand.

- The verification specification for a freely available RISC microprocessor core that is available on the Internet is not well developed and it is incomplete. Thus design process will be slowed down without a complete verification specification.

- The lack of well-developed verification specifications for these microprocessor cores will affect the physical design phase. A design needs to be functionally proven before the physical design phase can proceed smoothly. Otherwise, if the front-end design has to be changed, the physical design process has to be redone.

1-2-2 Problem Statement

So far, there is MIPS-compatible ISA which includes the Central Processing Unit (CPU), PS/2 mouse system, PS/2 keyboard system, basic memory, coprocessor 0 (CP0), and Universal Asynchronous Receiver/Transmitter (UART). However, Floating Point Unit (FPU) has not been designed and integrated in RISC32 yet. In general, although ALU could perform operations on floating point numbers, it was considered slow to meet the expectation. Hence, this project is initiated to design a floating point unit and then integrate it into RISC32 processor.

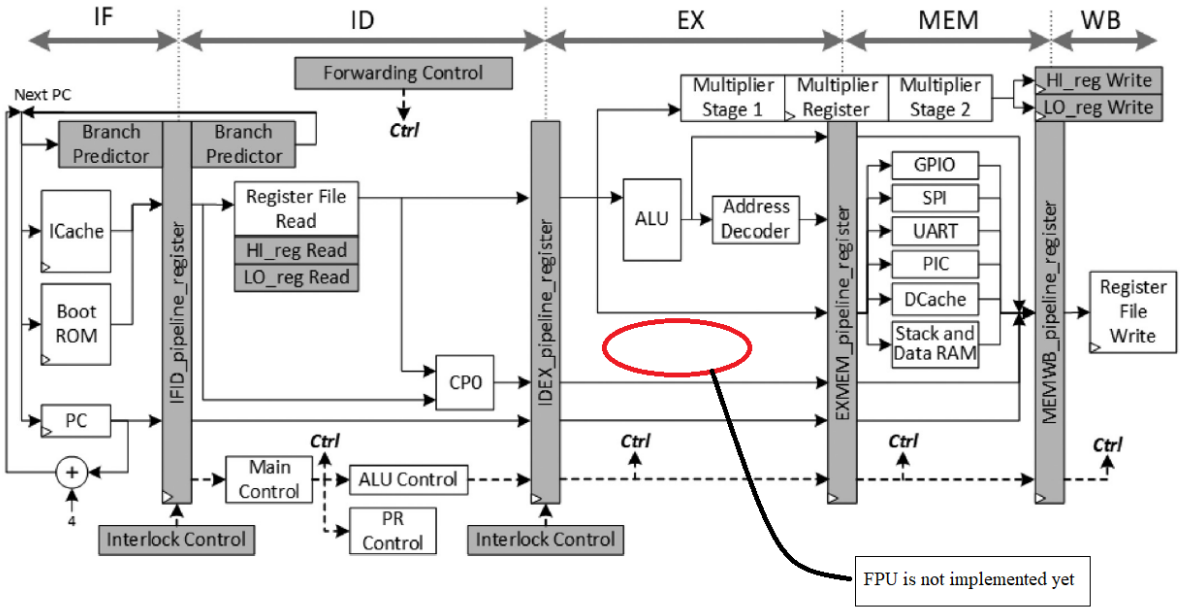


Figure 1-2-2-F1: RISC32 Microarchitecture (FPU not implemented on datapath unit).

1-3 Project Scope

This project is to design a FPU model with Verilog for RISC32 processor. The specifications of FPU and its internal blocks will be developed. The functionality of the FPU will be verified by using test bench. The FPU will be integrated into existing available RISC32 processor and verification will be done to ensure it is working. Lastly, the FPU will be synthesis on FPGA.

1-4 Project Objectives

Here are the objectives of the project:

- To design and develop the RTL model of FPU which include microarchitecture specification and testbench.
- To integrate the FPU into RISC32 processor.
- To synthesis the FPU module on FPGA with completes documented timing and resource usage information.

1-5 Impact, Significance and Contribution

In short, there is lacking of well-developed FPU based development environment out there. The development environment referred to the availability of the following:

- A well-developed design documentation of chip specification, architecture specification and micro-architecture specification from top level to bottom level.
- A fully functional well-developed FPU integrated into RISC32 processor in the form of synthesis-ready RTL written in Verilog.
- A well-developed verification specification of the FPU. The verification specification should contain complete verification methodology and its techniques as well as test plan, test bench architecture etc.

- A complete physical design in FPGA with documented timing and resources usage information.

This project is to develop an environment that mentioned above: to integrate the RISC32 processor core-based platform with the FPU which could support hardware modeling research work.

1-6 Report Organization

This report contains 7 chapters. The chapters are Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 Methodology, Chapter 4 System Specification, Chapter 5 Microarchitecture Specification, Chapter 6 Verification Specification, Chapter 7 Conclusion.

Chapter 1 Introduction states the motivation for the project, following by problem statement, project scope and objective and the background information of FPU and MIPS.

Chapter 2 Literature Review explains about the information related to FPU such as floating point number and format, single and double precision as well as arithmetic of floating point number. In this chapter, a previously developed FPU is also reviewed regarding its design.

Chapter 3 Methodology discuss about the flow of how the project is conducted. Proposed solution is also documented at this chapter.

Chapter 4 System design gives the overview of the system on the top level and the naming convention used within the system

Chapter 5 Microarchitecture Specification contains the units or components involved in the system design. This chapter identifies each unit involved in the system and gives an overview about each unit. Also contains the detailed discussion and design for each unit.

Chapter 6 Verification Specification shows the test written to verify the integration of the system. Result of the verification test is also documented here.

Chapter 7 Conclusion concludes the overall project development.

CHAPTER 2 LITERATURE REVIEW

2-1 Previous Works Done by Other Engineers/Researchers

Since ALU performed floating point operations slower in term of speed, FPU came to existence to speed up the mathematical operations of floating point numbers. FPU this project had been done by a couple of engineers before. For instance, according to Al-Eryani (2006), he used VHDL language to model the 32-bit floating point unit which complies fully with the IEEE 754 Standard.

From his project, the proposed FPU was able to support some arithmetic operations such as addition, subtraction, multiplication, division and square root. All arithmetic operations had these three stages:

1. Pre-normalize: The operands were transformed into formats that makes them easy and efficient to handle internally.

2. Arithmetic core: The basic arithmetic operations were done here.

3. Post-normalize: The result would be normalized if possible (leading bit before decimal point is 1, if possible) and then transformed into the format specified by the IEEE standard (Al-Eryani 2006).

Besides, the FPU also able to perform four rounding modes which were rounding to nearest even, to zero, rounding up and down. The FPU was tested with test cases created using SoftFloat which was a software implementation of floating-point that conforms to the IEC/IEEE Standard for Binary Floating-Point Arithmetic. Moreover, the FPU was tested in ModelSim with 100,000 test cases for each arithmetic operation and for each rounding mode. As a result, an FPU with features of 100 MHz operating frequency, few clock cycles and logic elements was implemented.

However, the FPU that done by him could only support single precision format floating point numbers. In addition, in order for the FPU to achieve high frequency, the FPU had to trade off its clock cycles in which it required more pipelining. The number of clock cycles that the FPU needs for each arithmetic operation was listed below:

Operation	Number of clock cycles
Addition	7
Subtraction	7
Multiplication	12
Division	35
Square-root	35

Table 2-1-T1: Number of clock cycles for each arithmetic operation (Al-Eryani 2006).

On the other hand, Lundgren (2014) also used VHDL to model double precision floating point core. By using double precision format, it could represented a wider range of numeric values. This core was designed to meet the IEEE 754 standard for double precision floating point arithmetic. This unit had been extensively simulated, covering all four operations (add, subtract, multiply, divide), rounding modes, exceptions like underflow and overflow, and even the obscure corner cases, like when overflowing from denormalized to normalized, and vice-versa. The floating point unit supports denormalized numbers, 4 operations, and 4 rounding modes (nearest, zero, + infinity, - infinity). The unit was synthesized with an estimated frequency of 185 MHz, for a Virtex5 target device.

Operation	Number of clock cycles
Addition	20
Subtraction	21
Multiplication	24
Division	71

Table 2-1-T2: Number of clock cycles for each arithmetic operation.

The floating point unit he developed supported denormalized numbers which required more signals and logic levels to accommodate gradual underflow. The supported clock speed of 185 MHz makes up for the large number of clock cycles required for each operation to complete which led to longer latency as it required more logic levels.

Apart from that, floating point numbers required costly processing hardware or lengthy software implementations as it had larger range of values. Therefore, powerful computations and techniques which reduced hardware and improved the performance like power, area and timing was required. This concern led Kukati et al. (2013) designed a 32-bit floating point arithmetic unit with faster carry save adders and clock gating techniques to reduce power dissipation. The low power optimizing technique 'Multi Threshold Voltage' is used for reducing the power consumption of the arithmetic unit. Below was their proposed hardware:

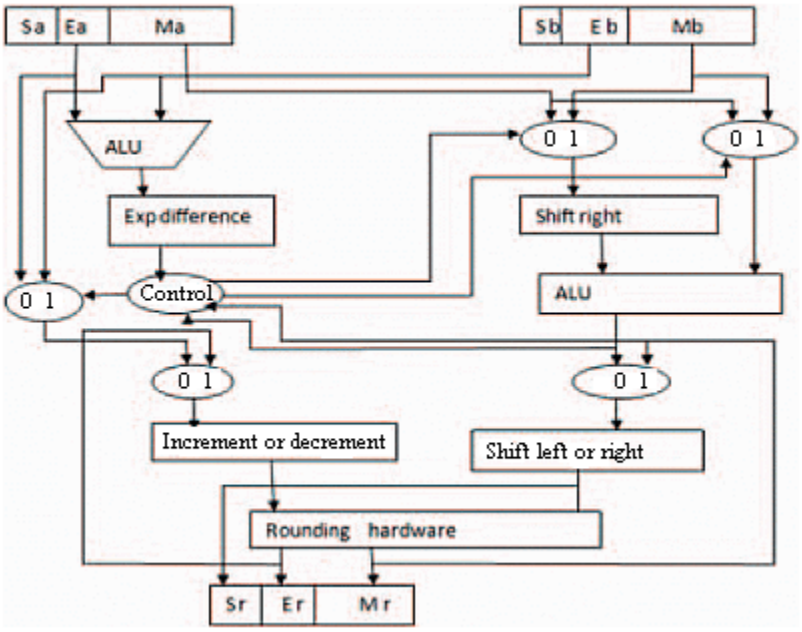


Figure 2-1-F1: Carry save adder (Kukati et al. 2013).

Figure 2-1-F1 shows the data flow of the computation of two floating point numbers. The arithmetic operation flow will be discussed later in this chapter.

2-2 Floating Point Number

There were several ways to represent real numbers on computer system. Fixed point places a radix point somewhere in the middle of the digits, and was equivalent to using integers that represent portions of some unit. For instance, a fixed-point number with 3 digits after the decimal point could be used to represent numbers such as: 1.005, 3.209, 28.000, etc. Another approach was to use rational, and represent every number as the ratio of two integers.

A number in scientific notation that has no leading 0s was called a normalized number, which was the usual way to write it. For example, 1.0×10^{-9} was in normalized scientific notation, but $0.1_{ten} \times 10^{-8}$ was not (Patterson & Hennesy 2014).

Binary numbers in scientific notation:

$$1.0_{two} \times 2^{-1}$$

Computer arithmetic that supported such numbers was called floating point because it represents numbers in which the binary point was not fixed, as it was for integers.

Floating point solved a number of representation problems. Fixed point had a fixed window of representation, which limited it from representing very large or very small numbers. Also, fixed-point was prone to a loss of precision when two large numbers were divided. Floating point, on the other hand, employed a sort of "sliding window" of precision appropriate to the scale of the number. This allowed it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease.

A standard scientific notation for reals in normalized form offers three advantages. It not only simplifies exchange of data that includes floating-point numbers, it also simplified the floating-point arithmetic algorithms to know that numbers would always be in this form as well as increasing the accuracy of the numbers that could be stored in a word, since the unnecessary leading 0s were replaced by real digits to the right of the binary point (Patterson & Hennesy 2014).

2-2-1 Single Precision Floating Point Number Representation

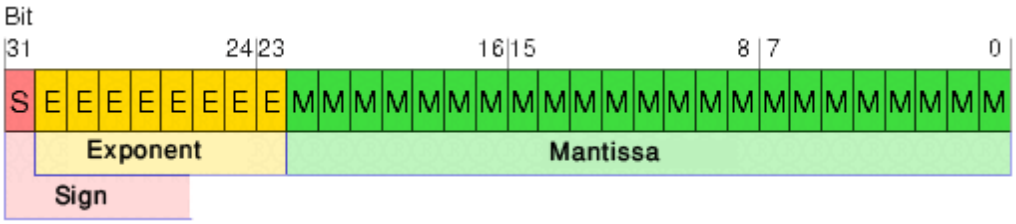


Figure 2-2-1-F1: Single precision floating point number representation.

S represents 1 sign bit.

E represents 8 exponent bits.

M represents 23 Mantissa or fraction (f) bits.

Floating point notation: $(-1)^s 2^e \times 1.f$ (normalized)

$f = (b_{23}^{-1} + b_{22}^{-2} + b_i^n + \ldots + b_0^{-23})$ where $b_i^n = 1$ or 0

s = sign (0 was positive; 1 was negative)

e = unbiased exponent; e = E – 127 (bias)

E_{max}= 255, E_{min}=0. E=255 and E=0 were used to represent special values.

2-2-2 Double Precision Floating Point Number Representation

One way to reduce chances of underflow or overflow was to offer another format that had a larger exponent. In C, this number was called double and operations on doubles were called double precision floating-point arithmetic.

The double precision format was a method of storing approximations to real numbers in a binary format. The term double came from the full name, double precision floating-point numbers. Originally, a 4-byte floating-point number was used, (float), however, it was found that this was not precise enough for most scientific and engineering calculations, so it was decided to double the amount of memory allocated, hence the abbreviation double. The word ‘double’ here meant 64 bits.

The representation of a double precision floating-point number took two MIPS words, where *s* is still the sign of the number, exponent is the value of the 11-bit exponent field, and fraction is the 52-bit number in the fraction field.

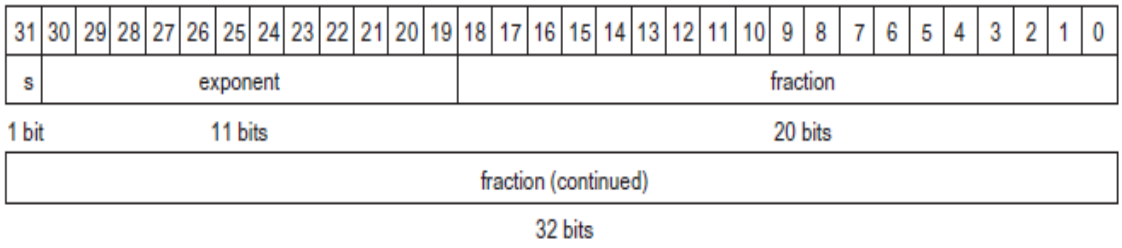


Figure 2-2-2-F1: Double precision floating point number representation.

Although double precision did increase the exponent range, its primary advantage was its greater precision because of the much larger fraction. Since 0 had no leading 1, it was given the reserved exponent value 0 so that the hardware would not attach a leading 1 to it. The exponent was stored by adding a bias of 011111111112 to the actual exponent. Thus, this was all the information we need to interpret a double precision floating point number in binary form.

2-3 Rounding

Although there were infinitely many integers, in most programs the result of integer computations could be stored in 32 bits. In contrast, given any fixed number of bits, most calculations with real numbers would produce quantities that could not be exactly represented using that many bits. Hence, the result of a floating-point calculation must often be rounded in order to fit back into its finite representation. The resulting rounding error is the characteristic feature of floating point computation (Goldberg 1991).

2-4 Arithmetic on Floating Point Numbers

2-4-1 Addition operation

Based on the report from Singh and Bhole (2014), they had implemented arithmetic unit that are specially designed to carry out operations on floating point numbers. Floating point addition and subtraction algorithms consisted of five stages

- Firstly, difference between exponent was to be calculated, difference $d = e1 - e2$. If $e1 < e2$ then $d = e2 - e1$.
- In second stage pre-alignment of mantissas was achieved by shifting smaller mantissa right by d bits.
- In third stage addition of mantissa was done to get tentative result for mantissa.
- Then normalization is done. If there were leading-zeros in tentative result, result was shifted to left and exponent is decreased by number of leading zeros. If overflows, then result was shifted right and exponent increased by 1 bit.
- Last stage was rounding and produce final output.

Figure below explains the data flow on how the floating point arithmetic operation work.

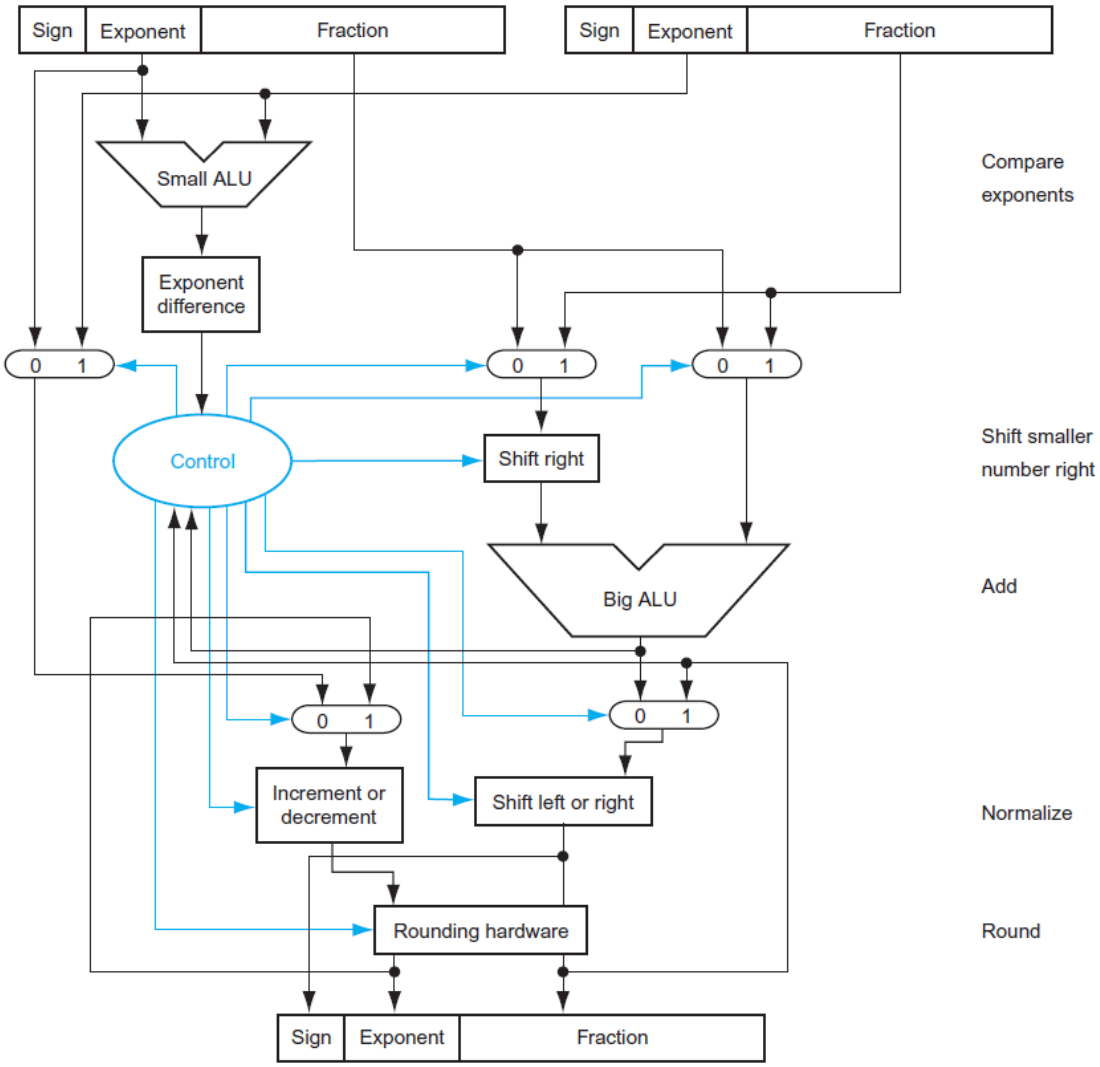


Figure 2-4-1-F1: Algorithm of addition operation.

Since multiplication and division were far more complicated, so they would not be discussed here. In this project, only addition operation is being focused and the above algorithm is being implemented.

2-5 Floating Point Pipeline

Due to FP operations required larger amount of logics to handle which arise some performance issue for the original 5-stage MIPS pipeline. This is because it is impractical to complete FP operations in 1 clock cycle as it will increase the latency for operations. The solution comes to extending the MIPS Pipeline for FP operations.

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

Figure 2-5-F1: Latencies and initiation intervals for functional units

Pipeline latency is equal to 1 cycle less than the depth of the execution pipeline, which is the number of stages from the EX stage to the stage that produces the result. Therefore, based on the figure above, the number of stages in an FP add unit is four.

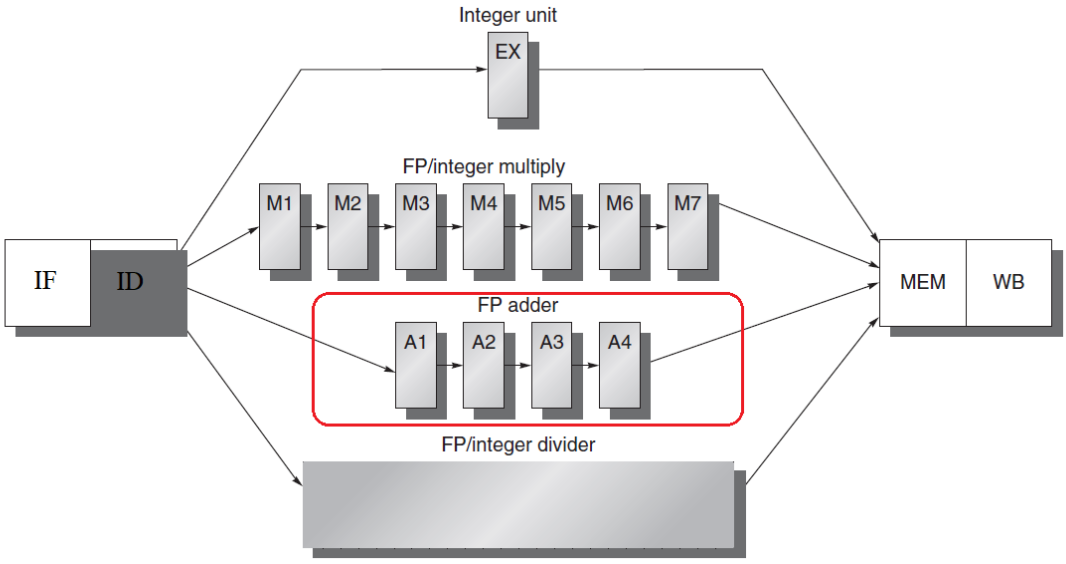


Figure 2-5-F1: Extended Pipeline for FP

In this project, only FP adder is being focused. Based on the figure above, the EX stage of the original pipeline is extended to 4 stages, A1 to A4 for FP adder. However, this only applies to floating point instruction such as add.s and add.d. The other instructions follow the original 5-stage pipeline for one complete execution.

CHAPTER 3: PROPOSED METHOD AND APPROACH

3-1 Design Methodology

There were two types of design methodology were available, Top-down design methodology and Bottom-up design methodology. In top-down design methodology, the top level representation of a chip was first defined then partitioned into lower level representations. For bottom-up design methodology, the leaf nodes were first defined. The leaf nodes were then integrated to form a higher level model of the chip. This process was repeated until the top level of the chip was reached. Since digital system often uses the abstraction concepts to simplify the design process, thus top-down design methodology was used in this project.

Top-down design methodology process flow was shown in Figure 4.1. This methodology would keep on repeating until the system design met the requirement on functionality. If the design did not meet the requirement, the design flow had to be repeated. This project focused on micro-architecture level design.

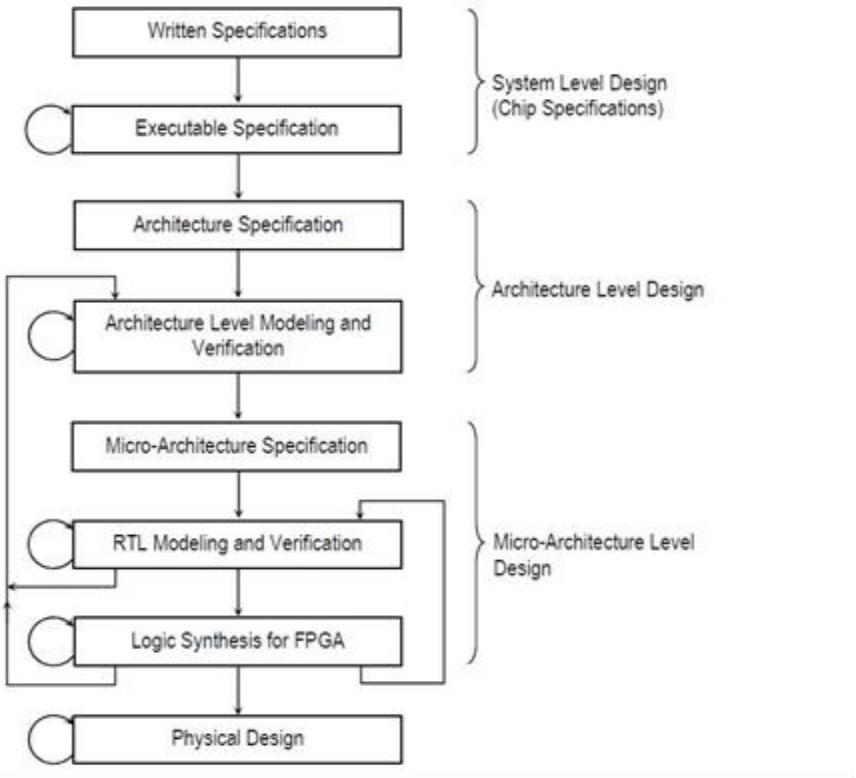


Figure 3-1-F1: Top-down design methodology.

3-1-1 Micro-Architecture Specification

Micro-architecture specification described the internal design of a unit. The internal design was described with design-specific technical information for RTL coding to begin. For this project, the information included for each internal block of FPU were:

- FPU functionality description
- FPU operating procedures
- FPU interfaces and I/O pin description
- FPU internal operation
- FPU functional partitioning into blocks
- For each blocks,
    - Block interfaces and I/O pin description
    - Block functionality
    - Block internal operation
    - Finite-state machine (FSM)
    - Block test plan

3-1-2 RTL Modeling and Verification

With the micro-architecture specification developed, the RTL coding on FPU internal block could begin. The functional correctness of the model was verified at two levels:

- Micro-architecture level: Internal blocks of FPU were individually verified before they were integrated into the architecture level.
- Architecture level: The individual blocks of FPU were integrated into a unit. Verification was performed on the FPU.

3-2 Design Tools

3-2-1 ModelSim

Since this design would be using Verilog, it was crucial to discuss commonly used design software that could support Verilog. There would be 3 design software discussed here:

Simulator	VCS	ModelSim	Quartus II
Company	SYNOPSYS® Predictable Success	Mentor Graphics®	ALTERA®
Language Supported	VHDL-2002 V2001 SV2005	VHDL-2002 V2002 SV2005	VHDL Verilog HDL
Platform Supported	Linux	-Windows XP/Vista/7/8 -Linux	-Windows XP/7/8 -Linux
Availability for free	No	YES (SE Edition only)	No

Table 3-2-1-T1: Comparison between simulation tools.

Since all of the design tools mentioned above were licensed product, ModelSim would be chose since free license was provided for student edition.

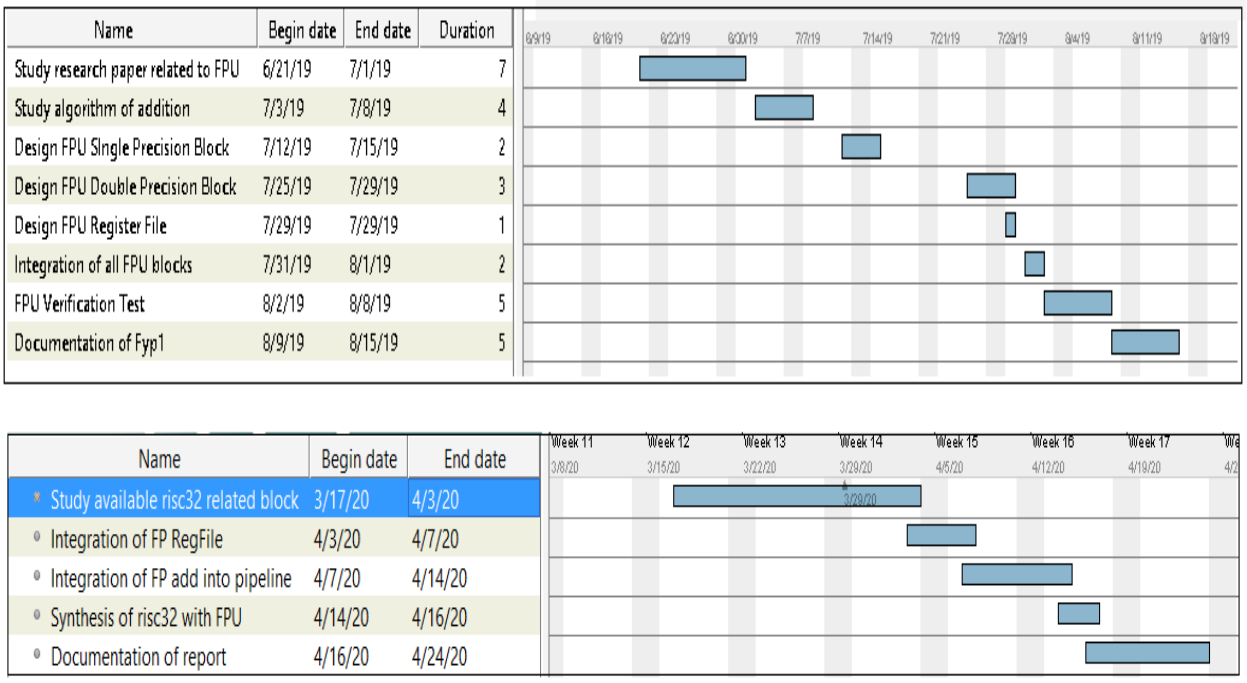
3-2-2 PC Spim

PC Spim was a simulator that provides a MIPS environment to simulate MIPS assembly language. It would be used to develop test program to verify the functionality of the design.

3-2-3 Xilinx Vivado

The Vivado development software was designed by Xilinx. This software was designed for synthesis and analysis of Verilog designs, enabling the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

3-3 Grantt Chart



Name	Begin date	End date	Duration
Study research paper related to FPU	6/21/19	7/1/19	7
Study algorithm of addition	7/3/19	7/8/19	4
Design FPU SIngle Precision Block	7/12/19	7/15/19	2
Design FPU Double Precision Block	7/25/19	7/29/19	3
Design FPU Register File	7/29/19	7/29/19	1
Integration of all FPU blocks	7/31/19	8/1/19	2
FPU Verification Test	8/2/19	8/8/19	5
Documentation of Fyp1	8/9/19	8/15/19	5

Name	Begin date	End date
Study available risc32 related block	3/17/20	4/3/20
Integration of FP RegFile	4/3/20	4/7/20
Integration of FP add into pipeline	4/7/20	4/14/20
Synthesis of risc32 with FPU	4/14/20	4/16/20
Documentation of report	4/16/20	4/24/20

Figure 3-3-F1: Grantt Chart of Project.

CHAPTER 4 SYSTEM SPECIFICATION

4-1 System Feature

	RISC32 with FPU
Dummy Instruction Cache (KB)	16
Dummy Data Cache (KB)	16
Data width (bits)	32
Instruction width (bits)	32
General Purpose Register	32
Special Purpose Register	HILO, PC
Co-Processor Register	32
Floating Point Register	**32**
Pipelined Stage	5
Data Hazard Handling	Yes
Control Hazard Handling	Yes
Interlock Handling	Yes
Exception Handling	Yes (4)
Data Dependency Forwarding	Yes
Branch Prediction	Dynamic – 2bits scheme
Multiplication (size of multiplier and multiplicand)	yes – 32 bits
Branch Delay Slot	Not supported
Instruction supported	**44**

Table 4-1-F1 RISC32 Features

4-1-1 System Functionality

1. Divide execution of instruction into 5 stages:

-IF(Instruction Fetch)	Instruction fetch and update PC
-ID(Instruction Decode)	Decode instruction and fetch operand
-EX(Execute)	Execute instruction
-MEM(Memory)	Read/write data from/ memory
-WB(Write Back)	Write back the result to the register file

2. Resolve data hazard by data forwarding.

3. Resolve load-use instructions problem using stalling.

4. Resolve structural hazards using separating data and instruction cache

5. Resolve control hazards by branch prediction.

6. Resolve exception interrupt with exception handler.

4-2 Operating Procedure

1. Start the system.
2. Porting sequence of instruction into instruction cache.
3. Reset the system for at least 2 clocks.
4. After the reset, the system will automatically fetch and run the program inside instruction cache.
5. Observe the waveform from development tools (Modelsim).

4-3 Naming Convention

Module - [lvl][mod. name]

Instantiation - [lvl][abbr. mod. name]

Pin - [lvl][type][abbr. mod. name]_[pin name]

- [lvl][type][abbr. mod. name]_[stage]_[pin name]

- [lvl][type][abbr. mod. name]_[abbr. mod. name]_[pin name]

	Description	Case	Available	Remark
lvl	Level	Lower	c : Chip u : Unit b : Block sb : sub-block	
mod. name	Module name	Lower all	Any	
abbr. mod. name	Abbreviated module name	Lower all	Any	Maximum 3 characters
type	Pin type	Lower	o : output i : input	
stage	Stage name	Lower all	if, id ,ex, mem, wb	
pin name	Pin name	Lower all	Any	Several word separated by "_"

Table 4-3-T1: Naming convention.

4-4 System Interface

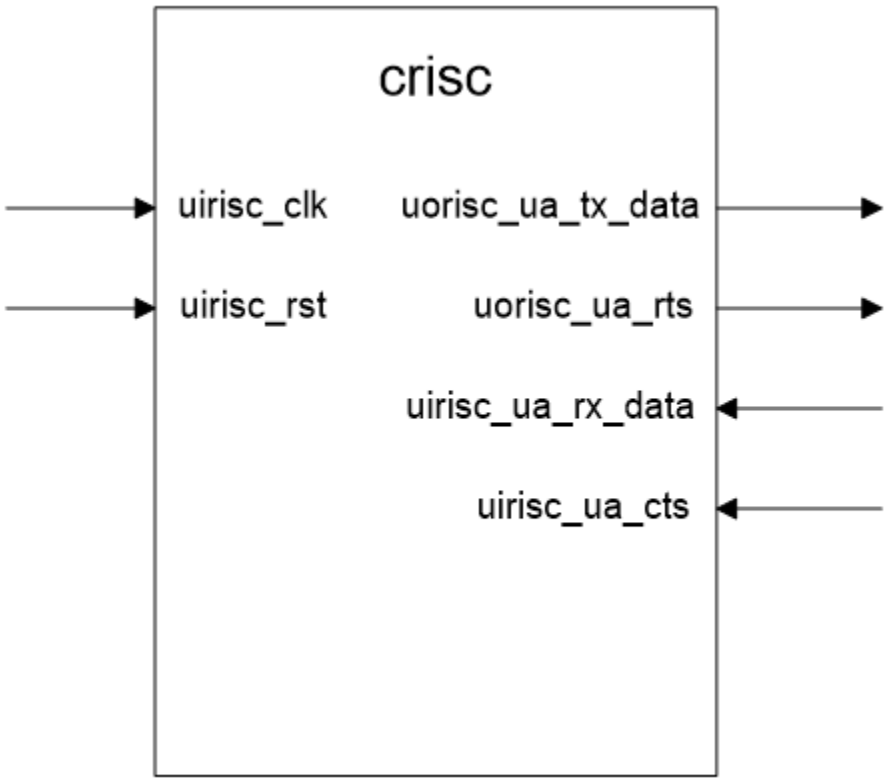


Figure 4-4-F1: Block diagram of RISC32 processor.

4-4-1 Input Pin Description

Pin Name: uirisc_ua_rx_data, Receive data	Source -> Destination: DCE -> crisc	Active: High
Pin Function: Serial data to be received from DCE to DTE. When no data is transfer, this port is held at logic "1". **DCE - Data Communication Equipment (External Modem) **DTE - Data Terminal Equipment (UART)		
Pin Name: uirisc_ua_cts, Clear-To-Send	Source -> Destination: DCE -> crisc	Active: High
Pin Function: To inform DTE that it can start transmit at uorisc_ua_tx_data port.		

Pin Name: uirisc_clk, System Clock	Source -> Destination: External source -> crisc	Active: High
Pin Function: System clock for the integration of UART and RISC32 processor.		
Pin Name: uirisc_rst, Reset	Source -> Destination: External source -> crisc	Active: High
Pin Function: System reset for the full chip. It is synchronous to the system clock.		

Table 4-4-1-T1: Input pin description of RISC32 chip.

4-4-2 Output Pin Description

Pin Name: uorisc_ua_tx_data, Transmit Data	Source -> Destination: crisc -> DCE	Active: High
Pin Function: Serial data to be sent from DTE to DCE. DTE shall hold this line at logic '1' when no data is transfer.		
Pin Name: uorisc_ua_rts, Request-To-Send	Source -> Destination: crisc -> DCE	Active: High
Pin Function: Transmission circuit will be enabled by this signal. Together with Clear-To-Send signal, data transmission between DTE and DCE will be coordinated. Request-To-Send shall be asserted by UART when UART has data in transmission buffer. Can be de asserted any time after START bit is sent.		

Table 4-4-2-T1: Output pin description of RISC32 chip.

4-5 Memory Map

Purpose	start address	Direction	Segment
Kernel module	0xC000 0000	Up	Kseg2
Boot Rom		Up	Kseg1
I/O register(if below 512MB)	0xA000 0000	Up	
Direct view of memory to 512MB linux kernel code and data		Up	Kseg0
Exception Entry point	0x8000 0180	Up	
Stack	0x7FFF FFFC	Down	Kuseg
Program heap	0x1000 8000	Up	
Dynamic library code and data	0x1000 0000	Up	
Main program	0x0040 0000	Up	
Reserved	0x0000 0000	Up	

Table 4-5-T1: Memory Map

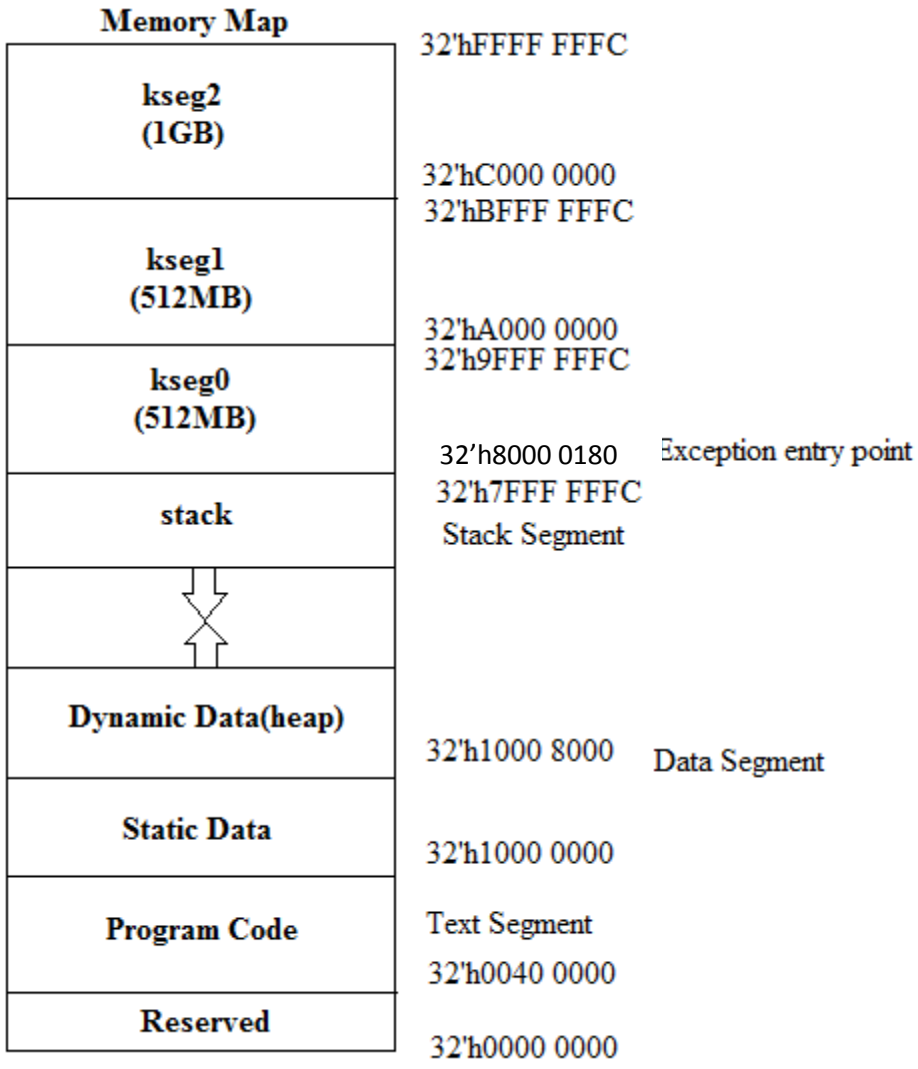


Figure 4-5-F1: Memory Map

However, due to the limitation of modelsim student edition version which only support up to 8k memory, the cache size will set text segment from 32'h0040_0000 to 32'h0040_1FFC, data segment from 32'h1000_0000 to 32'h1000_1FFC, stack segment from 32'h7fff_e000 to 32'h7fff_fffc, kernel text segment from 32'h8000_0000 to 32'h8000_1FFC and kernel data segment from 32'h9000_0000 to 32'h9000_0FFC.

4-5-1 Memory Map Description

Purpose	Description
Kernel Module	Accessible by kernel*
Boot Rom	Start-up ROM which keep the system configuration*
I/O registers(if below(512MB)	External IO device registers*
Direct view of memory to 512MB linux kernel code and data	Memory Allocation to view linux kernel code and data*
Exception Entry point	Software exception handling*
Stack	Use for argument passing
Program heap	Dynamic memory allocation such as malloc()
Dynamic library code and data	Data segment which is access by variable
Main program	Text segment which contain the main program

Table 4-5.1-T1: Memory Map Description

Note *: required CP0

4-6 System Register

4-6-1 General Purpose Register

Width : 32-bits

Size : 32 units

Retrieving method : 5-bits address as index

Name	Address	Use	Preserved Across A Call?
$zero	0	Constant Value 0 (hardwired)	N.A.
$at	1	Assembler Temporary	No
$v0 - $v1	2 - 3	Value for Function Results and Expression Evaluation	No
$a0 - $a3	4 - 7	Arguments	No
$t0 - $t7	8 – 15	Temporaries	No
$s0 - $s7	16 - 23	Saved temporaries	Yes
$t8 - $t9	24 – 25	Temporaries	No
$k0 - $k1	26 -27	Reserved for OS kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Table 4-6-1-T1: General Purpose Registers

4-6-2 Special Purpose Register

Width : 32-bits

Size : 2 units

Retrieving method : Via instructions: MFHI, MTHI, MFLO, MTLO, MULT or MULTU

Name	definition	location in double [64:0]
HI	Most Significant Word	Double [63:32]
LO	Least Significant Word	Double [31:0]

Table 4-6-2-T1: Special Purpose Register

4-6-3 Program Counter Register

Width : 32-bits

Size : 1 unit

Retrieving method : Control by instruction address generator control.

4-6-4 CP0 Register

Name	Address	Use
$bcp0_stat	12	Interrupt mask, enable bits and status when exception occurred
$bcp0_cause	13	Exception type and pending interrupt
$bcp0_epc	14	Address of instruction that caused exception

Table 4-6-4-T1: CP0 Register

4-6-5 FP Register

Width : 32 bits

Size : 32 units

Retrieving method : 5-bits address as index

Name	Usage
$f0 - $f3	Floating point return values
$f4 - $f10	Temporary registers. Not preserved across procedure calls
$f12 - $f14	First two arguments to subprograms, not preserved by subprograms
$f16 - $f18	More temporary registers, not preserved by subprograms
$f20 - $f31	Saved registers, preserved by subprograms

Table 4-6-5-T1: FP Register

4-7 Instruction Formats and Addressing Modes

4-7-1 Basic Instruction Formats

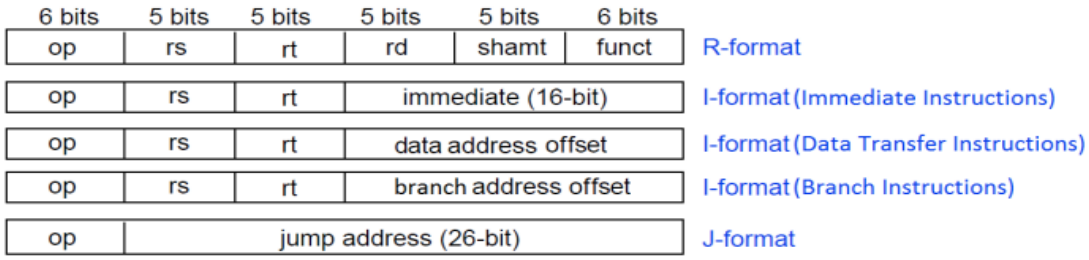


Figure 4-7-1-F1: Instruction Format

Abbreviation	Definitiion	Width
op	Operation code	6
rs	Source register	5
rt	Target register	5
rd	Destination register	5
shamt	Shift amount	5
funct	Function field	6
immediate	Immediate	16
data address offset	Data address offset	16
branch address offset	Branch address offset	16
jump address	Jump address	26

Table 4-7-1-T1: Instruction Format Definition

4-7-2 FP Instruction Formats

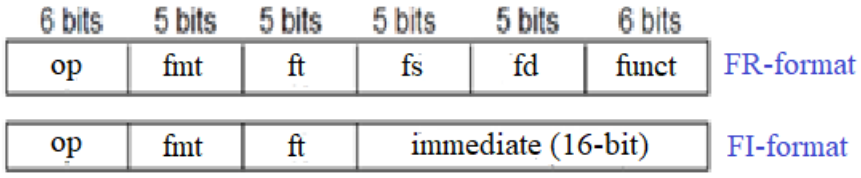


Figure 4-7-2-F1: FP Instruction Format

4-7-3 Addressing Modes

a) R-format

Register addressing: Perform operation on source and target register and store the result into destination register

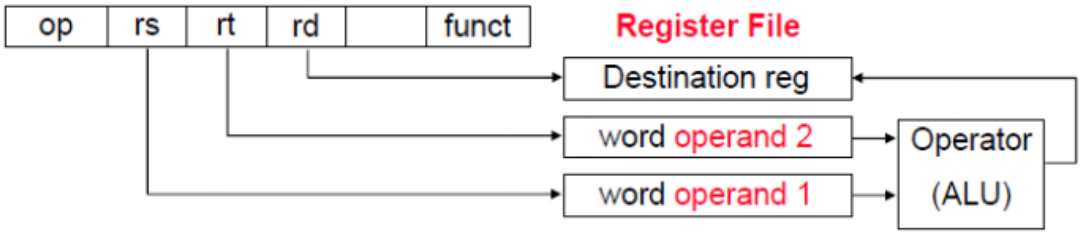


Figure 4-7-3-F1: R-format Addressing

b) I-format

i. Immediate addressing: Perform operation on source register and immediate and store the result into target register

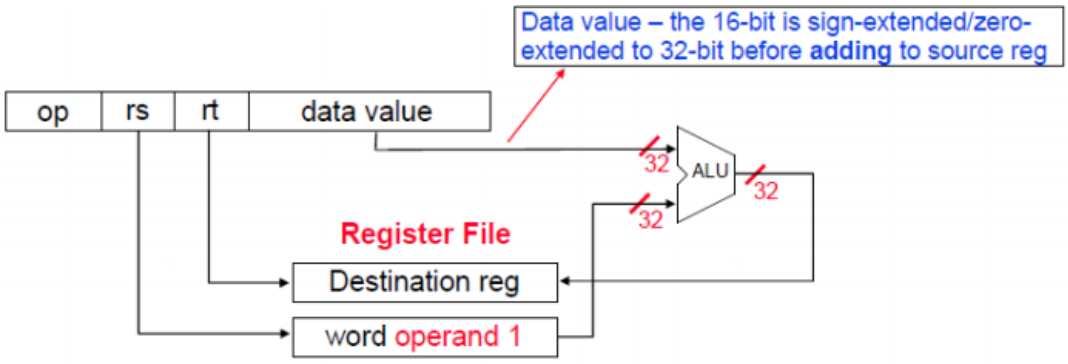


Figure 4-7-3-F2: Immediate Addressing

ii. Based displacement addressing: Perform operation on source register and immediate, the result is then uses as address to access the data memory to load/store data to/from target register

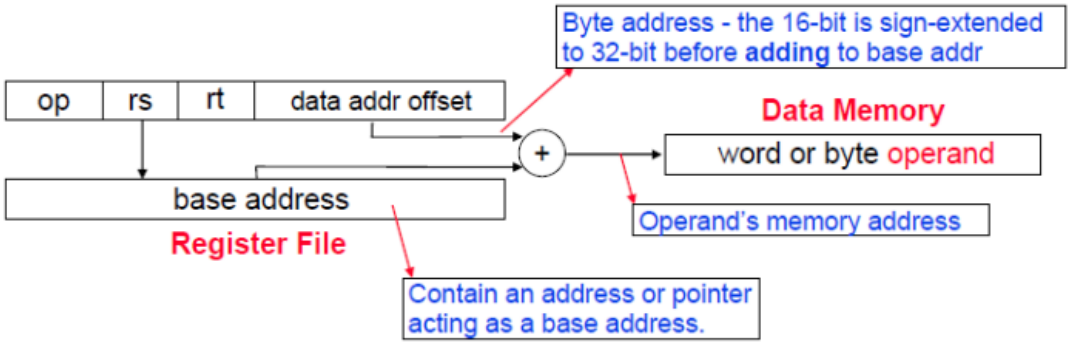


Figure 4-7-3-F3: Based Displacement Addressing

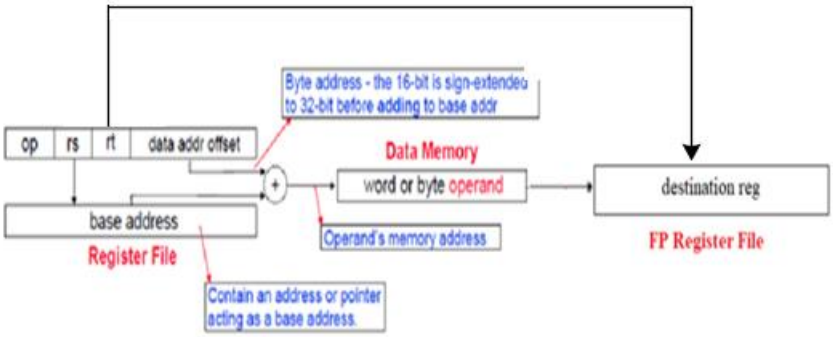


Figure 4-7-3-F4: Based Displacement Addressing with FP Register File (Used by lwc1, swc1)

iii. PC-relative addressing: Perform operation on source and target register to determine next PC condition, the immediate is uses as address offset for next PC

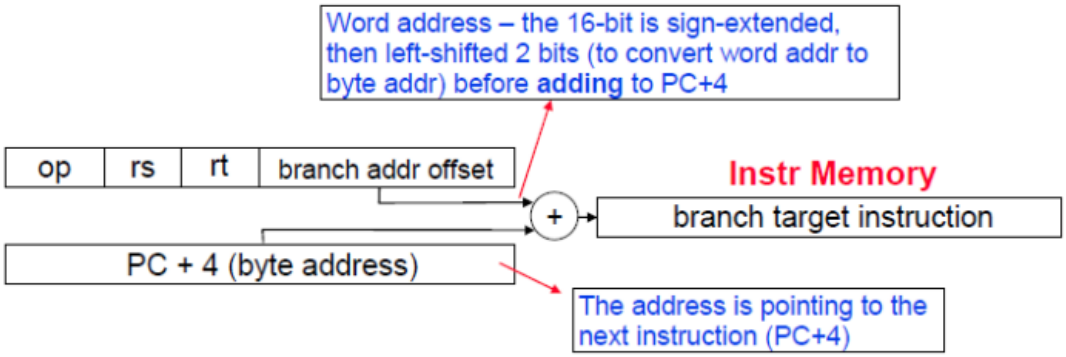


Figure 4-7-3-F5: PC-Relative Addressing

c) J-format

Pseudo-direct addressing: Perform operation by concatenating the upper bits of PC with the jump address

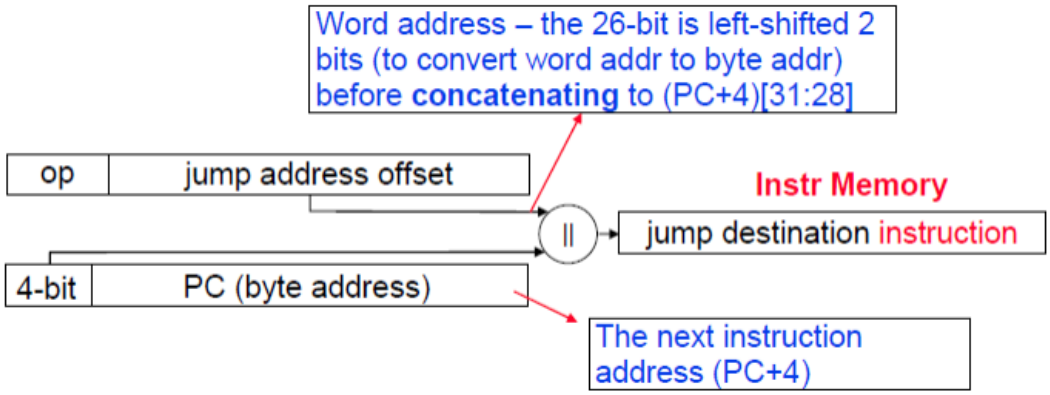


Figure 4-7-3-F6: Pseudo-Direct Addressing

d) FR-format

Register addressing: Perform operation on source and target register and store the result into destination register

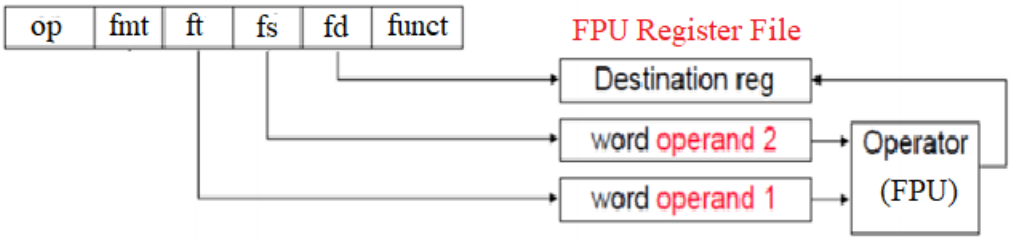


Figure 4-7-3-F7: Register Addressing for FR format (Used by add.s, add.d)

e) FI-format

PC-relative addressing: Perform operation on source and target register to determine next PC condition, the immediate is uses as address offset for next PC

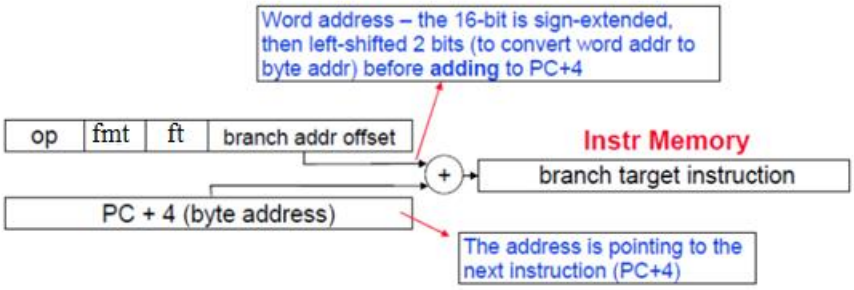


Figure 4-7-3-F8: Register Addressing for FR format (Used by FP branching instructions)

4-8 Supported Instructions Set

Instruction	Format	Addr. Mode	Machine Language						Register Transfer Notation	Assembly Format	Overflow
			OpCode	Rs	Rt	Rd	Shamt	Funct			
sll	R	Register	0x00	0	$rt	$rd	n	0x01	R[rd] =R[rs] << n	sll $rd, $rt, n	no
srl	R	Register	0x00	0	$rt	$rd	n	0x03	R[rd] =R[rs] >> n	srl $rd, $rt, n	no
sra	R	Register	0x00	0	$rt	$rd	n	0x04	R[rd] =R[rs] >>> n	sra $rd, $rt, n	no
jr	R	Register	0x00	$rs	0	0	0	0x0A	PC = R[rs]	jr $rs	no
jalr	R	Register	0x00	$rs	0	0	0	0x0B	PC = R[rs], R[31] = PC + 4	jalr $rs	no
mfhi	R	Register	0x00	0	0	$rd	0	0x10	R[rd] = HI	mfhi $rd	no
mthi	R	Register	0x00	$rs	0	0	0	0x11	HI = R[rs]	mthi $rs	no
mflo	R	Register	0x00	0	0	$rd	0	0x12	R[rd] = LO	mflo $rd	no
mtlo	R	Register	0x00	$rs	0	0	0	0x13	LO = R[rs]	mtlo $rs	no
mult	R	Register	0x00	$rs	$rt	0	0	0x24	HILO = R[rs] * R[rt]	mult $rs, $rt	no
multu	R	Register	0x00	$rs	$rt	0	0	0x24	HILO = U(R[rs]) * U(R[rt])	multu $rs, $rt	no
add	R	Register	0x00	$rs	$rt	$rd	0	0x20	R[rd] = R[rs] + R[rt]	add $rd, $rs, $rt	yes
addu	R	Register	0x00	$rs	$rt	$rd	0	0x21	R[rd] = U(R[rs]) + U(R[rt])	addu $rd, $rs, $rt	no
sub	R	Register	0x00	$rs	$rt	$rd	0	0x22	R[rd] = R[rs] - R[rt]	sub $rd, $rs, $rt	yes
subu	R	Register	0x00	$rs	$rt	$rd	0	0x23	R[rd] = U(R[rs]) - U(R[rt])	subu $rd, $rs, $rt	no
and	R	Register	0x00	$rs	$rt	$rd	0	0x24	R[rd] = R[rs] & R[rt]	and $rd, $rs, $rt	no
or	R	Register	0x00	$rs	$rt	$rd	0	0x25	R[rd] = R[rs] | R[rt]	or $rd, $rs, $rt	no
xor	R	Register	0x00	$rs	$rt	$rd	0	0x26	R[rd] = R[rs] ^ R[rt]	xor $rd, $rs, $rt	no
nor	R	Register	0x00	$rs	$rt	$rd	0	0x27	R[rd] = ~(R[rs] | R[rt])	nor $rd, $rs, $rt	no
slt	R	Register	0x00	$rs	$rt	$rd	0	0x2A	R[rd] = (R[rs] < R[rt]) ? 1 : 0	slt $rd, $rs, $rt	no
sltu	R	Register	0x00	$rs	$rt	$rd	0	0x2B	R[rd] = (U(R[rs]) < U(R[rt])) ? 1 : 0	sltu $rd, $rs, $rt	no
j	J	Pseudo-Direct	0x02	JumpAddr (Label)					PC = {(PC+4) [31:28], JumpAddr, 2'b00}	j label	no

beq	I	PC-Relative	0x04	$rs	$rt	BranchAddr (Label)	PC = (R[rs] == R[rt]) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	beq $rs, $rt, label	no
bne	I	PC-Relative	0x05	$rs	$rt	BranchAddr (Label)	PC = (R[rs] != R[rt]) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	bne $rs, $rt, label	no
blez	I	PC-Relative	0x06	$rs	0	BranchAddr (Label)	PC = (R[rs] <=0) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	blez $rs, $rt, label	no
bgtz	I	PC-Relative	0x07	$rs	0	BranchAddr (Label)	PC = (R[rs] > 0) ? (PC + 4 + (SE(BranchAddr)<<2)) : (PC + 4)	bgtz $rs, $rt, label	no
addi	I	Immediate	0x08	$rs	$rt	Imm	R[rt] = R[rs] + SE(Imm)	addi $rt, $rs, imm	yes
addiu	I	Immediate	0x09	$rs	$rt	Imm	R[rt] = U(R[rs]) + U(ZE(Imm))	addiu $rt, $rs, imm	no
slti	I	Immediate	0x0A	$rs	$rt	Imm	R[rt] = (R[rs] < SE(Imm)) ? 1 : 0	slti $rt, $rs, imm	no
sltiu	I	Immediate	0x0B	$rs	$rt	Imm	R[rt] = (U(R[rs]) < U(SE(Imm))) ? 1 : 0	sltiu $rt, $rs, imm	no
andi	I	Immediate	0x0C	$rs	$rt	Imm	R[rt] = R[rs] & ZE(Imm)	andi $rt, $rs, imm	no
ori	I	Immediate	0x0D	$rs	$rt	Imm	R[rt] = R[rs] | ZE(Imm)	ori $rt, $rs, imm	no
xori	I	Immediate	0x0E	$rs	$rt	Imm	R[rt] = R[rs] ^ ZE(Imm)	xori $rt, $rs, imm	no
lui	I	Immediate	0x0F	$rs	$rt	Imm	R[rt] = Imm << 16	lui $rt, imm	no
lw	I	Based-Displacement	0x23	$rs	$rt	Imm	R[rt] = MEM[R[rs] + SE(Imm)]	lw $rt, imm($rs)	no
sw	I	Based-Displacement	0x2B	$rs	$rt	Imm	MEM[R[rs] + SE(Imm)] = R[rt]	sw $rt, imm($rs)	no

mfc0		Register	0x10	0x00	$rt	$rd	0x00	0x00	R[rt] = R[rd] (from CP0)	mfc0 $rt, $rd	no
mtc0		Register	0x10	0x04	$rt	$rd	0x00	0x00	R[rd] (from CP0) = R[rt]	mtc0 $rt, $rd	no
eret		Register	0x10	0x10	0x00	0x00	0x00	0x18	PC = R[epc] (from CP0)	eret	no
add.s	**FR**	**Register**	**0x11**	**0x10**	**$ft**	**$fs**	**$fd**	**0x00**	**F[fd] = F[fs] + F[ft]**	**add.s $fd, $fs, $ft**	**yes**
add.d	**FR**	**Register**	**0x11**	**0x11**	**$ft**	**$fs**	**$fd**	**0x00**	**{F[fd], F[fd+1]} = {F[fs], F[fs+1]} + {F[ft], F[ft+1]}**	**add.d $fd, $fs, $ft**	**yes**
lwc1	**I**	**Based-Displacement**	**0x31**	**$rs**	**$rt**	**Imm**			**F[rt] = MEM[R[rs] + SE(Imm)]**	**lwc1 $rt, imm($rs)**	**no**
swc1	**I**	**Based-Displacement**	**0x39**	**$rs**	**$rt**	**Imm**			**MEM[R[rs] + SE(Imm)] = F[rt]**	**swc1 $rt, imm($rs)**	**no**
mfc1	**R**	**Register**	**0x11**	**0x00**	**$rt**	**$rd**	**0x00**	**0x00**	**R[rt] = F[rd]**	**mfc1 $rt, $rd**	**no**
mtc1	**R**	**Register**	**0x11**	**0x04**	**$rt**	**$rd**	**0x00**	**0x00**	**F[rd] = R[rt]**	**mtc1 $rt, $rd**	**no**

Table 4-8-T1: Supported Instruction Set

CHAPTER 5: MICROARCHITECTURE SPECIFICATION

5-1 Design Hierarchy and Partitioning

Chip Partitioning (Top Level) at Architecture Level	Unit Partitioning at Micro-Architecture Level	Block and Functional Block Partitioning at RTL (Micro-Architecture Level)	Sub-Block
RISC32 Pipeline Processor (crisc)	Datapath (udata_path)	Branch Predictor (bbp_4way)	
		Register File (brf)	
		Interlock Control (bitl_ctrl)	
		Forward Control (bfw_ctrl)	
		32-bit Multiplier (bmult32)	add_lvl1_lastrow
			adder_lvl1
			adder_lvl1_firstrow
			adder_lvl2
			adder_lvl2_lastrow
			adder_lvl3
			adder_lvl4
			adder_lvl5
			sub_lvl1_lastrow
		ALB (balb)	
		Coprocessor0(bcp0)	
		FP Register File(bfp_rf)	
		FP Pre-Normalize (bfp_pre_norm)	
		FP Adder (bfp_adder)	
		FP Post-Normalize (bfp_post_norm)	
		FP Rounding (bfp_rounding)	
	Controlpath (uctrl_path)	Main Control (bmain_ctrl)	
		ALB Control (balb_ctrl)	
	Cache (ucache)		
	IO Bus (uiobusarbiter)		
	PS/2 Controller (ups2)	PS/2 Receiver (bps2rx)	
		PS/2 Transmitter (bps2tx)	
		PS/2 Address Decoder (bps2addr_decoder)	
	UART Controller	UART Address Decoder (bua_decoder)	

	(uuart)	UART CPU Interface (bcpuif)	
		UART Receiver (brx)	UART Receiver Controller (sbrx_ctr)
		UART Transmitter (btx)	UART Transmitter Controller (sbtx_ctr)
		UART Baud Rate Generator (bbaud)	

Table 5-1-T1: Design Hierarchy of RISC32 Processor with FP Register File, FP Pre-Normalize, FP Adder, FP Post-Normalize and FP Rounding

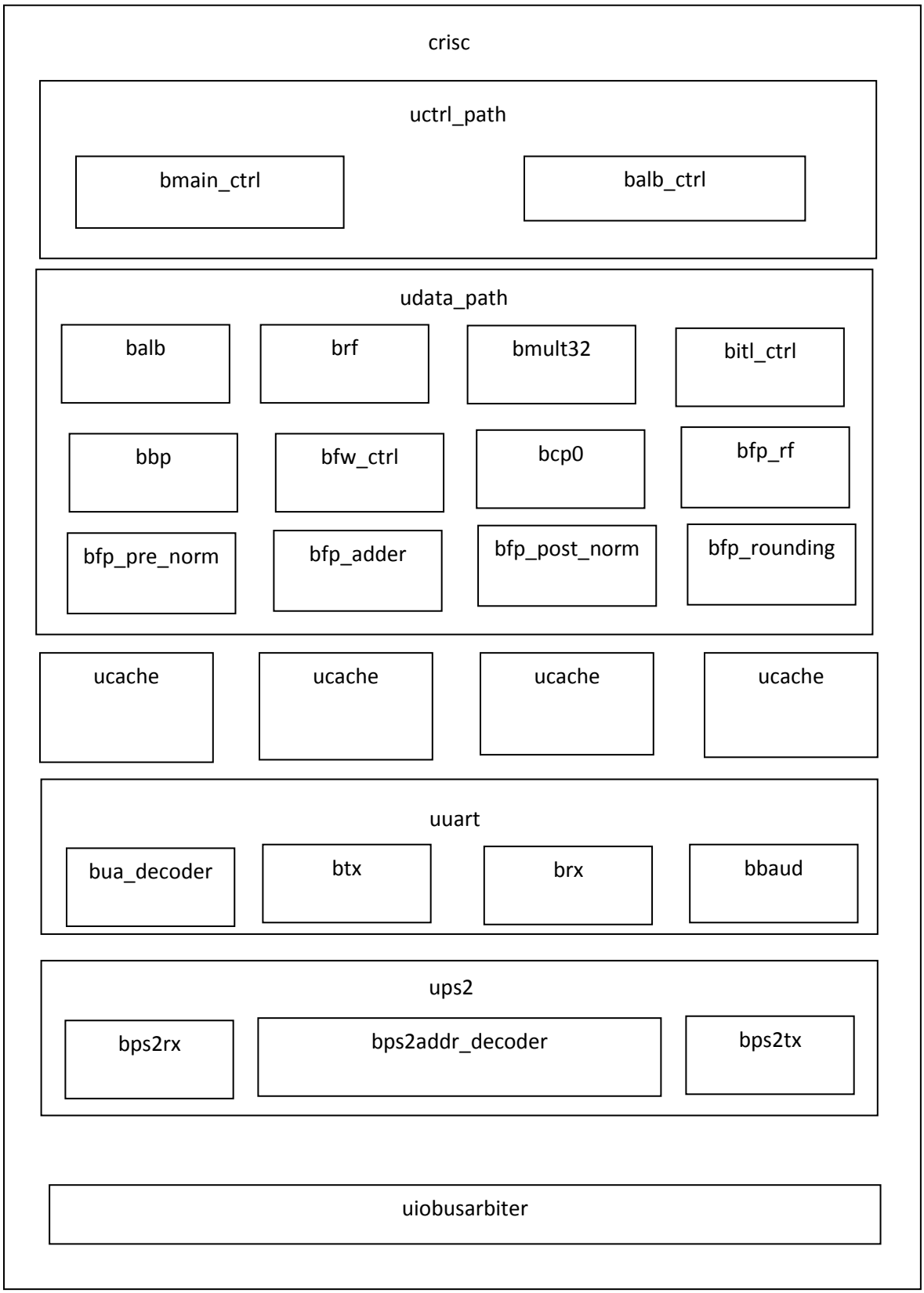


Figure 5.1-F1: Block Partitioning

5-2 Microarchitecture of RISC32 processor

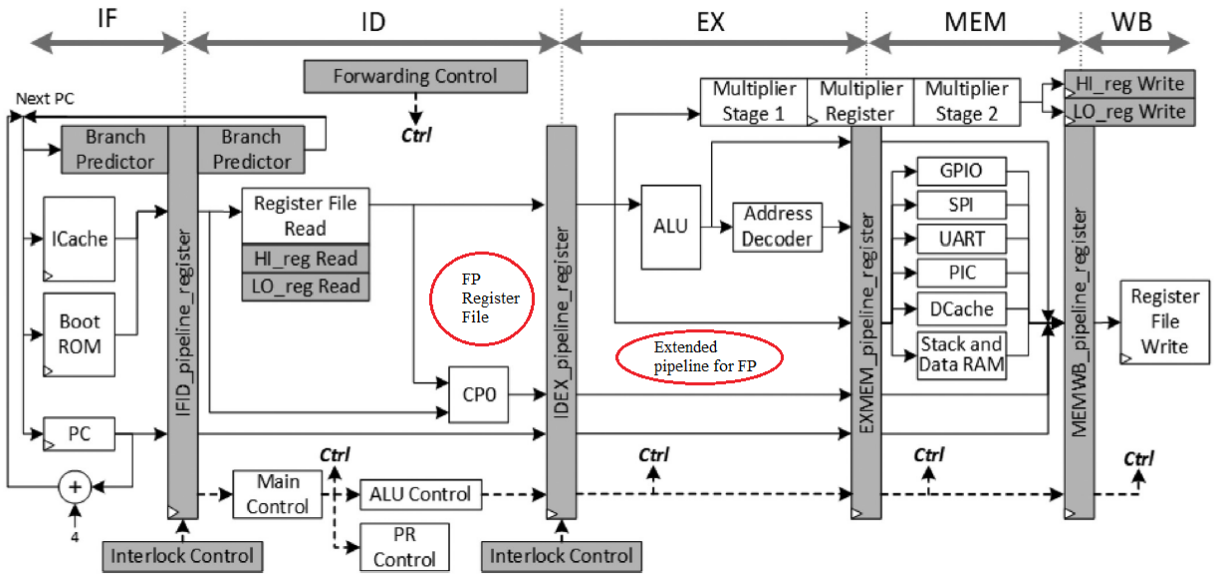


Figure 5-2-F1: Microarchitecture of RISC32 processor

Figure above is the microarchitecture of RISC32 processor with 5 stages pipeline. The register file and the FP blocks will be implemented as shown in microarchitecture view above. The more detailed microarchitecture view will be shown in next page.

5-2-1 Interface of FP Register File and Extended Pipeline with Datapath Unit

The figure below shows the interface between FP register file and extended pipeline with datapath Unit. Only related signal of Datapath Unit is shown.

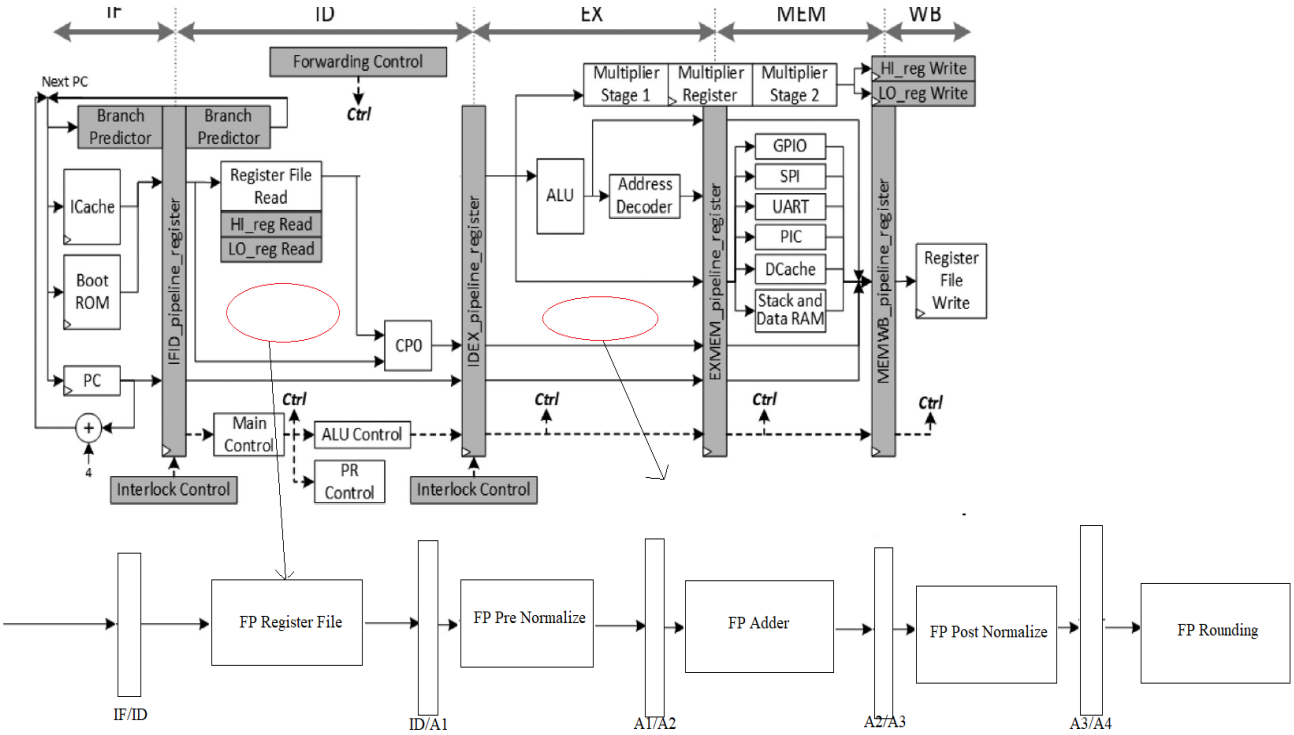


Figure 5-2-1-F1: Interface of FP Register File and Extended Pipeline with Datapath Unit

Based on microarchitecture above, FP register file is integrated into ID stage with is same as the general register file. The second circle shows the extended FP pipeline which has to 4 stages of A1, A2, A3 and A4. The extended pipeline only meant for the FP instruction such as addition. The four stages consist of FP Pre Normalize block, FP adder block, FP Post Normalize block and FP Rounding block. Therefore, for FP arithmetic operation likes addition, it will take total of 8 stages to complete the instruction. The ALU and address decoder that in the EX stage will not be affected as these functional blocks still follow the original 5-stage pipeline.

5-3 Datapath Unit

5-3-1 Datapath Unit Interface

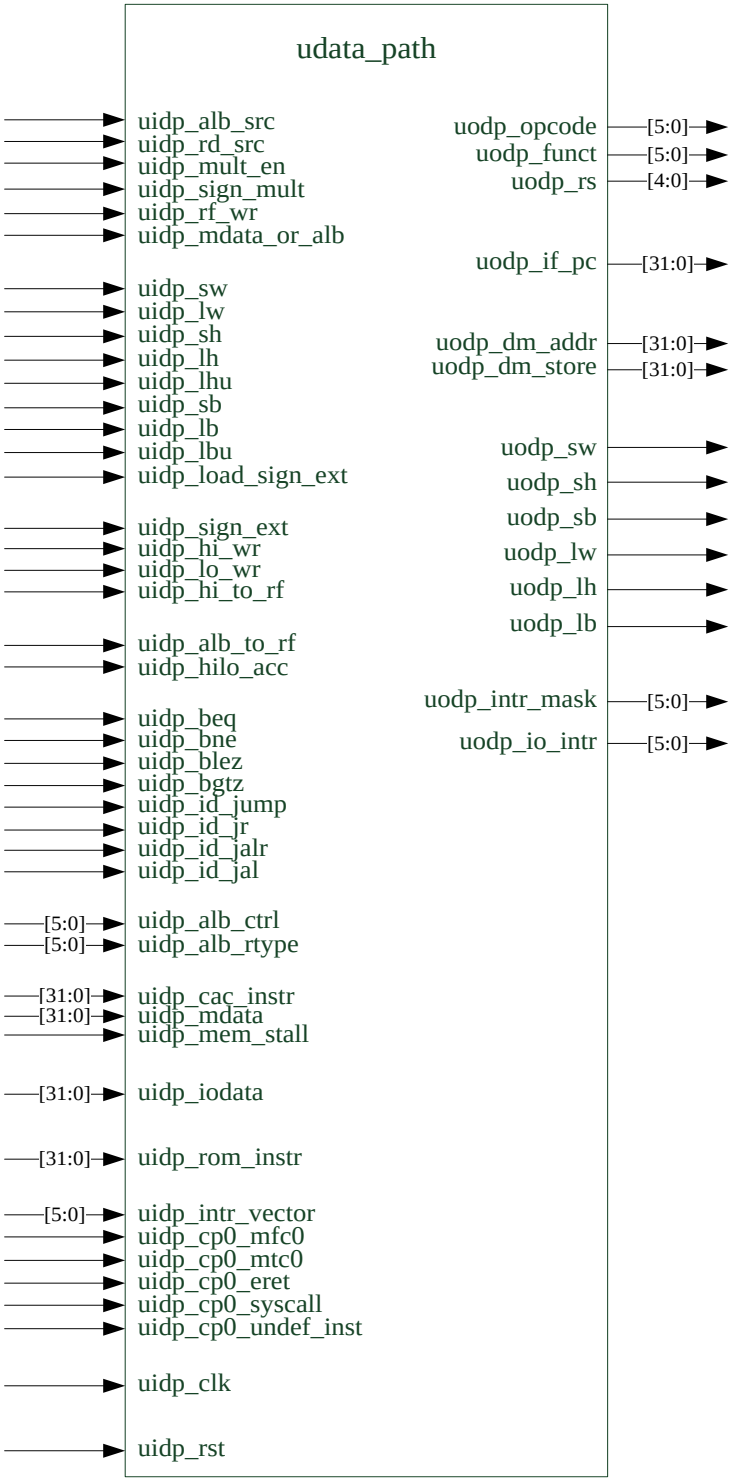


Figure 5-3-1-F1: Datapath Unit Interface

5-3-2 FPU Register File Block

5-3-2-1 Functionality

- Act as temporary storage of FPU to hold data and address.
- Able to read and write data.

5-3-2-2 FPU Register File Block Interface

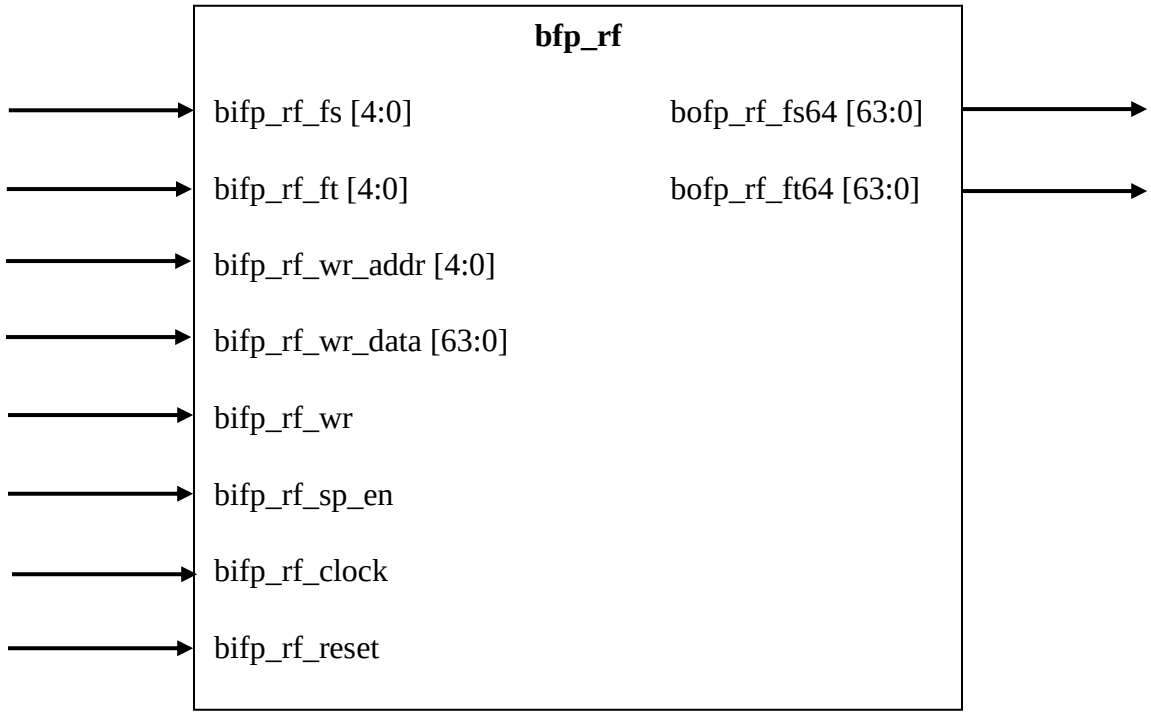


Figure 5-3-2-2-F1: Block Interface of FPU Register File

5-3-2-3 Input Pin Description

Pin Name: bifp_rf_fs[4:0]	Source -> Destination: udata_path -> bfp_rf	Pin Class: Data
Pin Function: 5 bits fs address to indicate FPU register file location.		
Pin Name: bifp_rf_ft[4:0]	Source -> Destination: udata_path -> bfp_rf	Pin Class: Data
Pin Function: 5 bits ft address to indicate FPU register file location.		
Pin Name: bifp_rf_wr_addr[4:0]	Source -> Destination: udata_path -> bfp_rf	Pin Class: Data
Pin Function: 5 bits destination address to indicate FPU register file location.		
Pin Name: bifp_rf_wr_data[63:0]	Source -> Destination: udata_path -> bfp_rf	Pin Class: Data
Pin Function: 64 bits data to be written in FPU register file.		
Pin Name: bifp_rf_wr	Source -> Destination: udata_path -> bfp_rf	Pin Class: Control
Pin Function: Use as enable signal to write data to FPU register file.		
Pin Name: bifp_rf_sp_en	Source -> Destination: udata_path -> bfp_rf	Pin Class: Control
Pin Function: Use as control signal to indicate single precision when asserted and double precision when de-asserted.		
Pin Name: bifp_rf_clock	Source -> Destination: udata_path -> bfp_rf	Pin Class: Global
Pin Function: Clock signal for FPU register file.		
Pin Name: bifp_rf_reset	Source -> Destination: udata_path -> bfp_rf	Pin Class: Global
Pin Function: Reset signal for FPU register file.		

Table 5-3-2-3-T1: Input Pin Description of FPU Register File

5-3-2-4 Output Pin Description

Pin Name: bofp_rf_fs64 [63:0]	Source -> Destination: bfp_rf -> udata_path	Pin Class: Data
Pin Function: 64 bits data output is read out to perform operation.		
Pin Name: bofp_rf_ft64 [63:0]	Source -> Destination: bfp_rf -> udata_path	Pin Class: Data
Pin Function: 64 bits data output is read out to perform operation.		

Table 5-3-2-4-T1: Output Pin Description of FPU Register File

5-3-3 FP Pre Normalize Block

5-3-3-1 Functionality

- To find the difference of exponent and normalize input for operation

5-3-3-2 FP Pre Normalize Block Interface

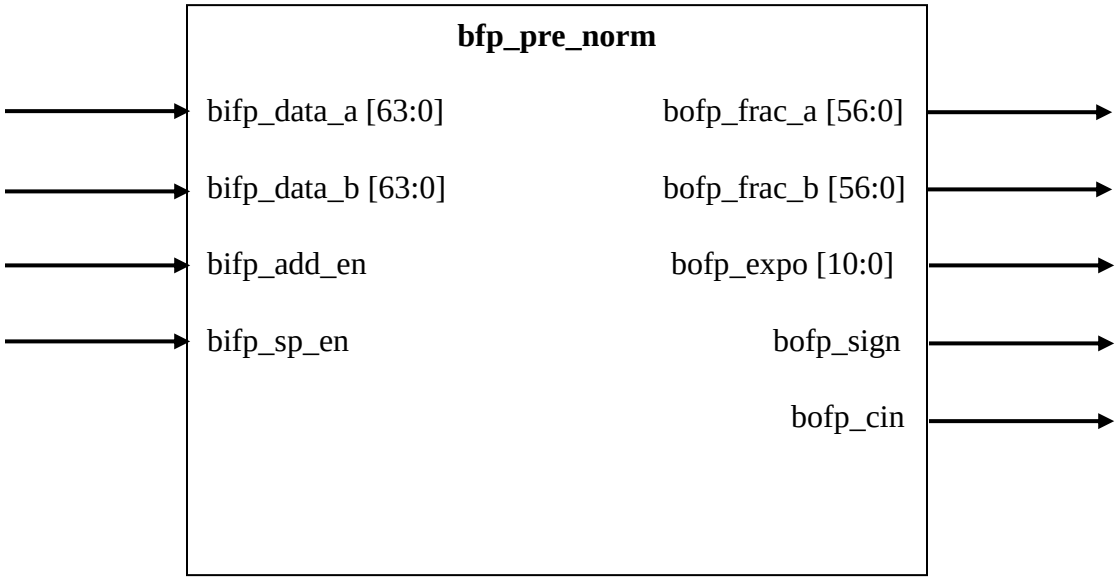


Figure 5-3-3-2-F1: Block Interface of FP Pre Normalize Block

5-3-3-3 Input Pin Description

Pin Name: bifp_data_a [63:0]	Source -> Destination: bfp_rf -> bfp_pre_norm	Pin Class: Data
Pin Function: 64 bits input data from FP register file to perform operation.		
Pin Name: bifp_data_b [63:0]	Source -> Destination: bfp_rf -> bfp_pre_norm	Pin Class: Data
Pin Function: 64 bits input data from FP register file to perform operation.		
Pin Name: bifp_add_en	Source -> Destination: bfp_rf -> bfp_pre_norm	Pin Class: Control
Pin Function: Use as control signal to enable addition operation when asserted.		
Pin Name: bifp_sp_en	Source -> Destination: bfp_rf -> bfp_pre_norm	Pin Class: Control
Pin Function: Use as control signal to enable single precision when asserted and double precision when de-asserted.		

Table 5-3-3-3-T1: Input Pin Description of FP Pre Norm Block

5-3-3-4 Output Pin Description

Pin Name: bofp_frac_a [56:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 57 bits of fraction part data is fetch to next stage for operation.		
Pin Name: bofp_frac_b [56:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 57 bits of fraction part data is fetch to next stage for operation.		
Pin Name: bofp_expo [10:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 11 bits of exponent part data is fetch to next stage for operation.		
Pin Name: bofp_sign	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 1 bit of sign data is fetch to next stage for operation.		
Pin Name: bofp_cin	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 1 bit of cin data is fetch to next stage for operation.		

Table 5-3-3-4-T1: Output Pin Description of FP Pre Norm Block

5-3-3-5 FP Pre Normalize Internal Block Diagram

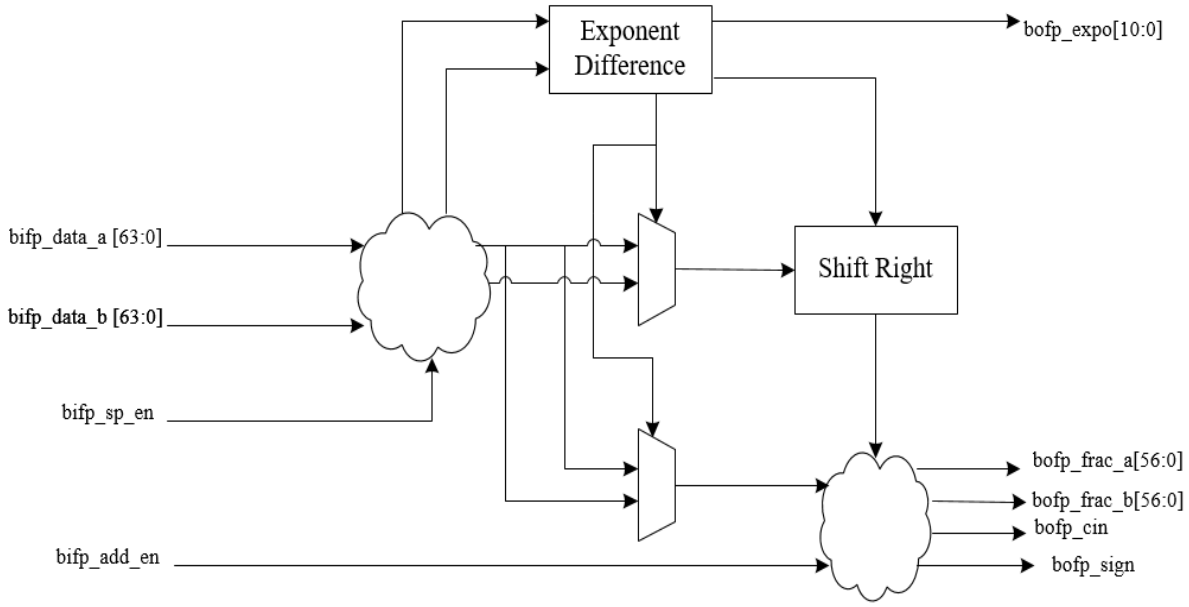


Figure 5-3-3-5-F1 FP Pre Normalize Internal Block Diagram

5-3-4 FP Adder Block

5-3-4-1 Functionality

- To perform addition operations on floating point data.

5-3-4-2 FP Adder Block Interface

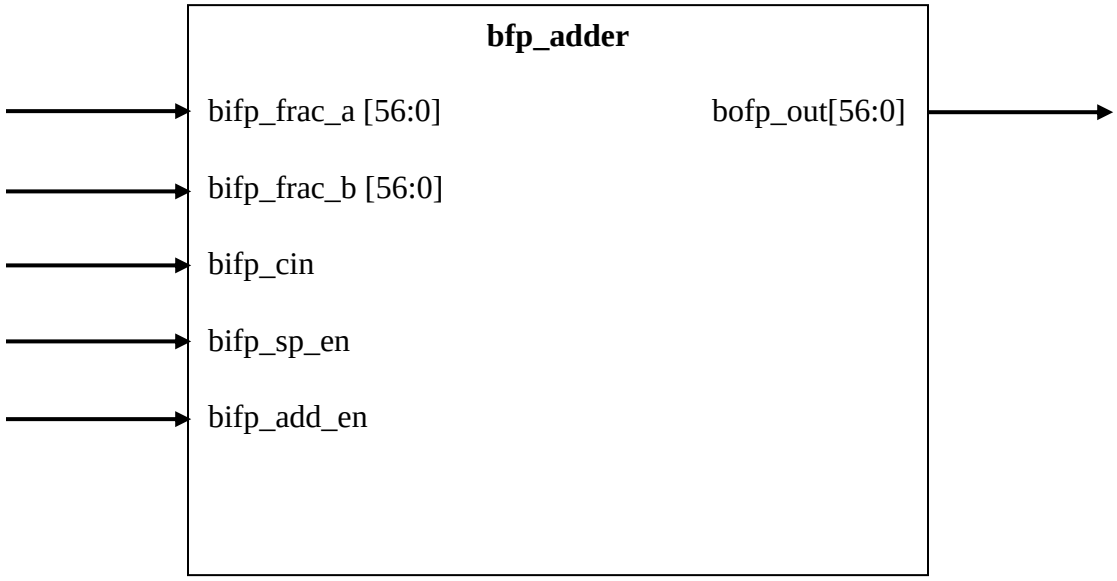


Figure 5-3-4-2-F1 Block Interface of FP Adder Block

5-3-4-3 Input Pin Description

Pin Name: bifp_frac_a [56:0]	Source -> Destination: udata_path -> bfp_adder	Pin Class: Data
Pin Function: 57 bits input fraction part data for addition operation.		
Pin Name: bifp_frac_b [56:0]	Source -> Destination: udata_path -> bfp_adder	Pin Class: Data
Pin Function: 57 bits input fraction part data for addition operation.		
Pin Name: bifp_cin	Source -> Destination: udata_path -> bfp_adder	Pin Class: Data
Pin Function: 1 bit input cin data for addition operation.		
Pin Name: bifp_sp_en	Source -> Destination: udata_path -> bfp_adder	Pin Class: Control
Pin Function: Use as control signal to enable single precision when asserted and double precision when de-asserted.		
Pin Name: bifp_add_en	Source -> Destination: udata_path -> bfp_adder	Pin Class: Control
Pin Function: Use as control signal to enable addition operation when asserted.		

Table 5-3-4-2-T1: Input Pin Description of FP Adder Block

5-3-4-4 Output Pin Description

Pin Name: bofp_out[56:0]	Source -> Destination: bfp_adder -> udata_path	Pin Class: Data
Pin Function: 57 bits result from addition of fraction part of two input data.		

Table 5-3-4-4-T1: Output Pin Description of FP Adder Block

5-3-4-5 FP Adder Internal Block Diagram

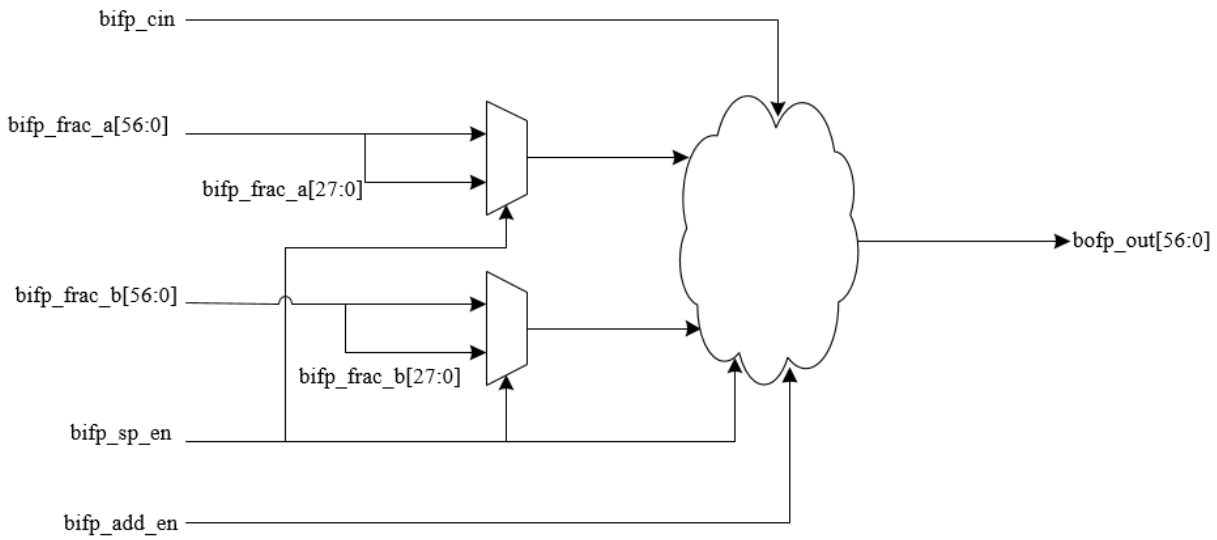


Figure 5-3-4-5-F1 FP Adder Internal Block Diagram

5-3-5 FP Post Normalize Block

5-3-5-1 Functionality

- To normalize the output from FP Adder

5-3-5-2 FP Post Normalize Block Interface

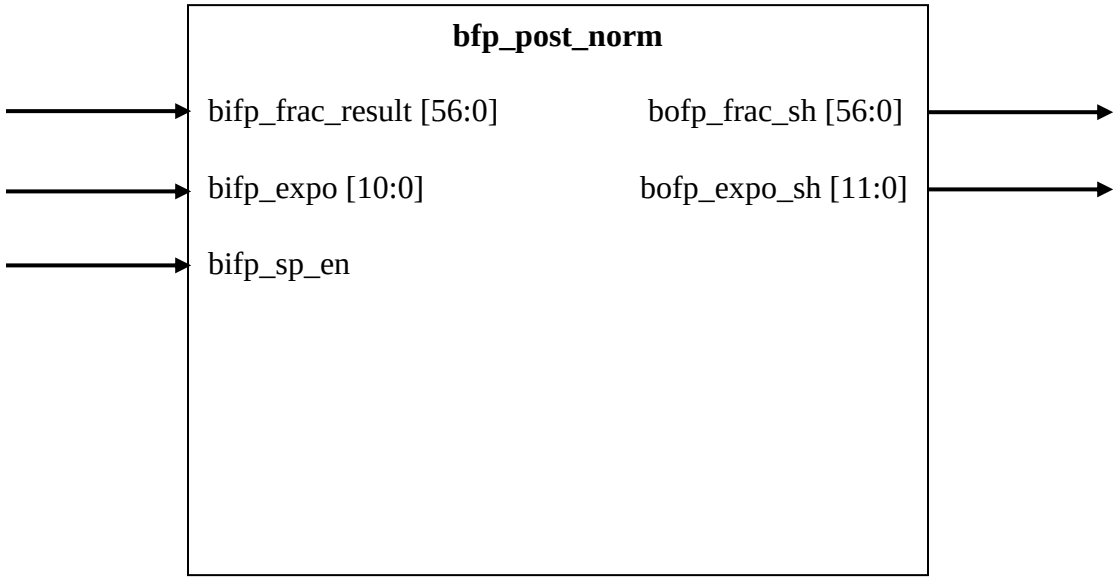


Figure 5-3-5-2-F1: Block Interface of FP Post Normalize Block

5-3-5-3 Input Pin Description

Pin Name: bifp_frac_result [56:0]	Source -> Destination: udata_path -> bfp_post_norm	Pin Class: Data
Pin Function: 57 bits input data of fraction result to be shifted and normalized.		
Pin Name: bifp_expo [10:0]	Source -> Destination: udata_path -> bfp_post_norm	Pin Class: Data
Pin Function: 11 bits input data of exponent data to be shifted and normalized.		
Pin Name: bifp_sp_en	Source -> Destination: udata_path -> bfp_post_norm	Pin Class: Control
Pin Function: Use as control signal to enable single precision when asserted and double precision when de-asserted.		

Table 5-3-5-3-T1: Input Pin Description of FP Post Norm Block

5-3-5-4 Output Pin Description

Pin Name: bofp_frac_sh [56:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 57 bits of shifted fraction part data is fetch to next stage for rounding and produce final output.		
Pin Name: bofp_expo_sh [11:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 12 bits of shifted exponent part data is fetch to next stage for rounding and produce final output.		

Table 5-3-5-4-T1: Output Pin Description of FP Post Norm Block

5-3-5-5 FP Post Normalize Internal Block Diagram

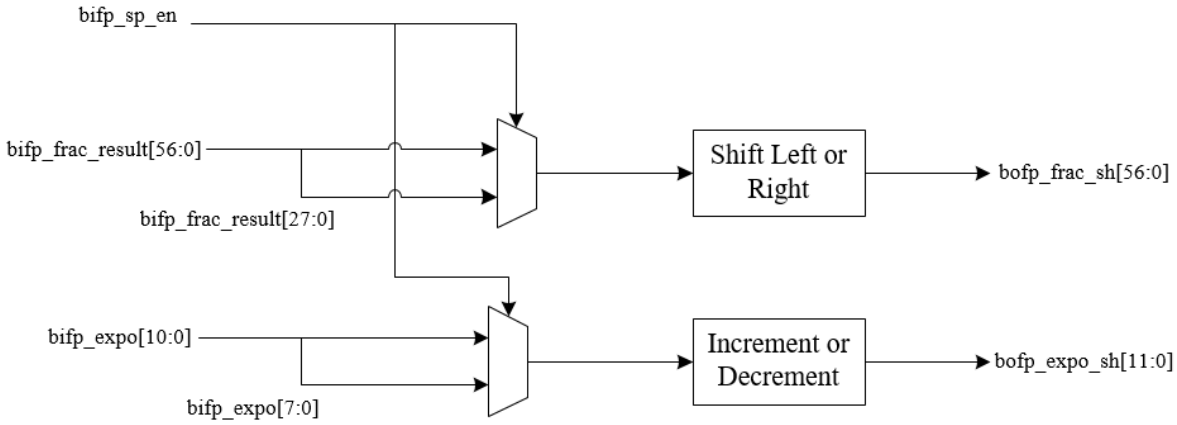


Figure 5-3-5-5-F1 FP Post Normalize Internal Block Diagram

5-3-6 FP Rounding Block

5-3-6-1 Functionality

- To round off and produce final output.

5-3-6-2 FP Rounding Block Interface

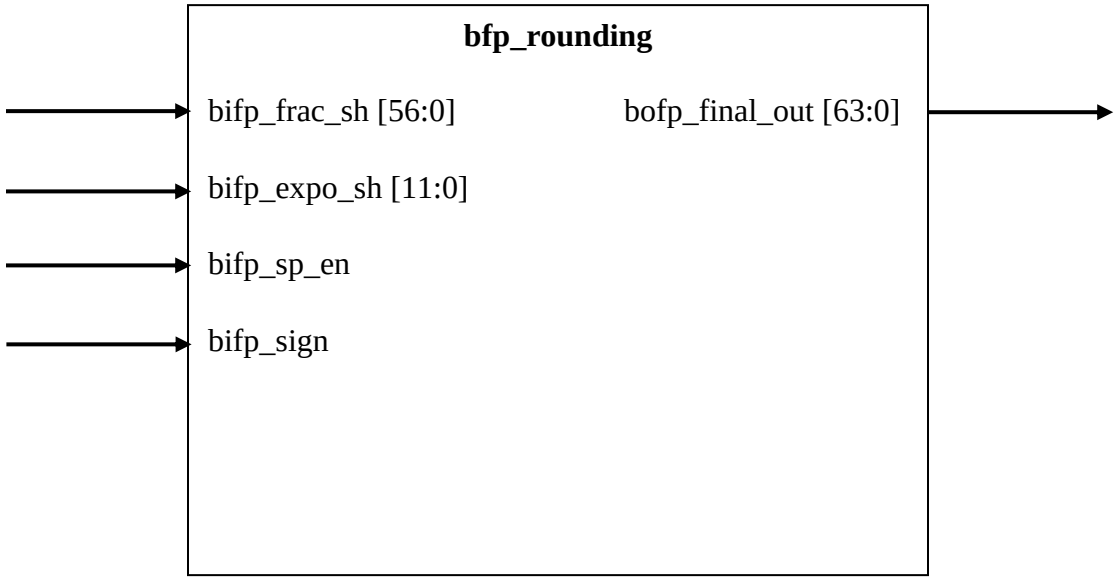


Figure 5-3-6-2-F1: Block Interface of FP Rounding Block

5-3-6-3 Input Pin Description

Pin Name: bifp_frac_sh [56:0]	Source -> Destination: udata_path -> bfp_rounding	Pin Class: Data
Pin Function: 57 bits of shifted fraction result to be rounded off and combined for final output.		
Pin Name: bifp_expo_sh [11:0]	Source -> Destination: udata_path -> bfp_rounding	Pin Class: Data
Pin Function: 12 bits of shifted exponent data to be rounded off and combined for final output.		
Pin Name: bifp_sign	Source -> Destination: udata_path -> bfp_rounding	Pin Class: Data
Pin Function: 1 bit of sign data to be combined for final output.		
Pin Name: bifp_sp_en	Source -> Destination: udata_path -> bfp_rounding	Pin Class: Control
Pin Function: Use as control signal to enable single precision when asserted and double precision when de-asserted.		

Table 5-3-6-3-T1: Input Pin Description of FP Rounding Block

5-3-6-4 Output Pin Description

Pin Name: bofp_final_out [63:0]	Source -> Destination: bfp_pre_norm -> udata_path	Pin Class: Data
Pin Function: 64 bits of final out is produced after completion of addition operation.		

Table 5-3-6-4-T1: Output Pin Description of FP Rounding Block

5-3-6-5 FP Rounding Block Internal Diagram

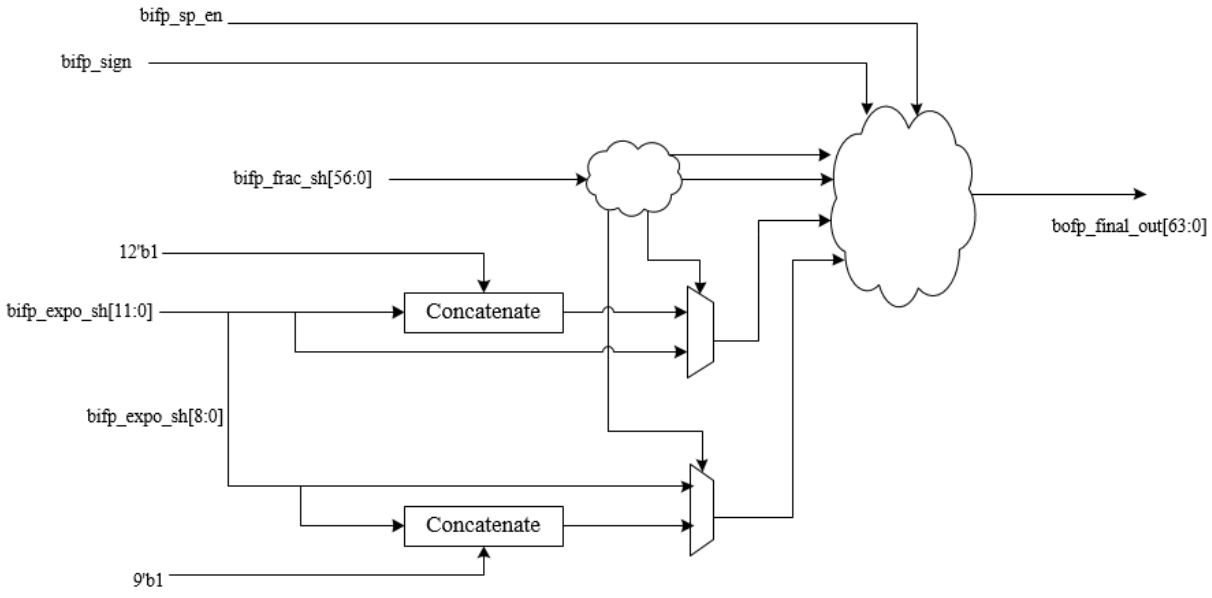


Figure 5-3-6-5-F1 FP Rounding Internal Block Diagram

5-4 Controlpath Unit

5-4-1 Controlpath Unit Interface

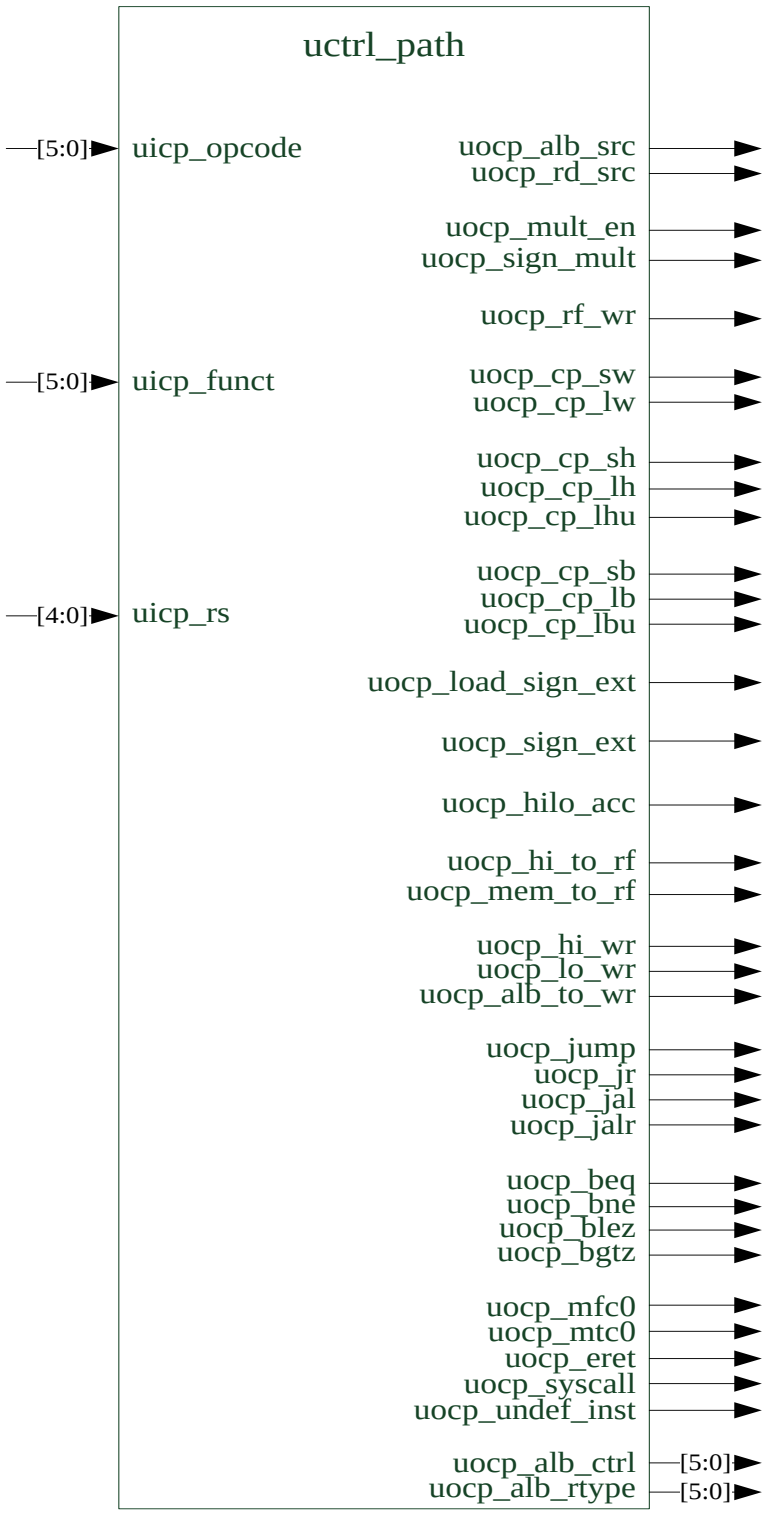


Figure 5-4-1-F1: Controlpath Unit Interface

5-5 Rom Unit

The rom unit stores boot loader code to initialize RISC32 processor. Program Counter directly points to the first line of address in here upon boot up.

5-5-1 Rom Unit Interface

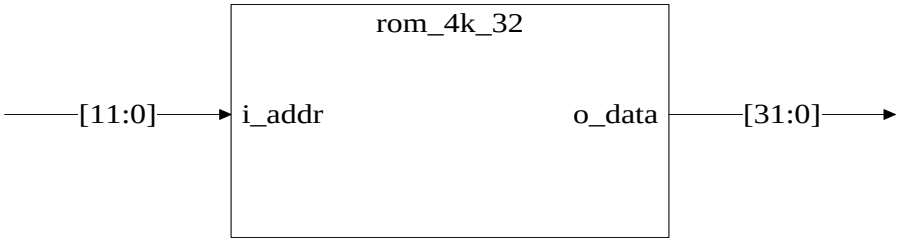


Figure 5-5-1-F1: Rom Unit Interface

5-6 Memory Unit

The memory unit stores machine instruction or program data. Range of address determines what to be stored inside each memory unit.

5-6-1 Memory Unit Interface

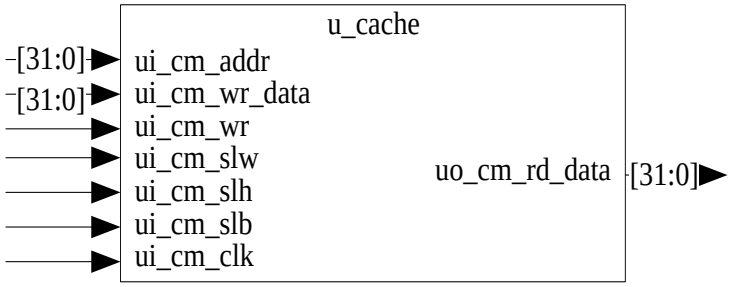


Figure 5-6-1-F1: Memory Unit Interface

5-6-2 Memory Unit Mapping

Module name	Address range [Max Bound: Min Bound]	Memory content description
u_internal_register_kseg0	[32'hbf00_0040:32'hbf00_0030]	Mapped to internal registers such as I/O registers.
u_kdata_kseg0	[32'h9000_0ffc:32'h9000_0000]	Kernel data segment in kernel segment 0.
u_ktext_kseg0	[32'h8000_1ffc:32'h8000_0000]	Kernel text segment in kernel segment 0. Stores kernel codes such as exception handler code.
u_stack_segment	[32'h7fff_fffc:32'h7fff_e000]	Stack segment for user space. Stores program data
u_data_segment	[32'h1000_1ffc:32'h1000_0000]	Data segment for user space. Stores program data
u_text_segment	[32'h0040_1ffc:32'h0040_0000]	Text segment for user space. Stores machine instruction

Table 5-6-2-T1: Memory Unit mapping and its content description

CHAPTER 6: VERIFICATION SPECIFICATION

6-1 Test Plan for FPU

Test	Function to be Tested	Expected Output
Test Case #1: Reset • Set the reset pin to high. • Hold for 3 clock cycle. • Set the reset pin to low.	Reset the whole FPU.	All outputs result in 0.
Test Case #2: Addition function test on single precision numbers • Set ui_fpu_fs = 5'b00010. • Set ui_fpu_ft = 5'b00100. • Set ui_fpu_sp = 1'b1. • Set ui_fpu_add = 1'b1. • Set ui_fpu_wr = 1'b1. • Set ui_fpu_wr_addr = 5'b00011. • Hold for 4 clock cycle.	Test the addition function on single precision floating point numbers.	uo_fpu _result = 32'b0_1000_0000_1100_0000_0000_0000_0000_000 ("3.5"). The result is saved in register file address 4'b00011.
Test Case #3: Addition function test on double precision numbers • Set ui_fpu_fs = 5'b00110. • Set ui_fpu_ft = 5'b01000. • Set ui_fpu_sp = 1'b0. • Set ui_fpu_add = 1'b1. • Set ui_fpu_wr = 1'b1. • Set ui_fpu_wr_addr = 5'b01010. • Hold for 4 clock cycle.	Test the addition function on single precision floating point numbers.	uo_fpu _result = 64'b0_1000_0000_010_0010_0111_0010_00101101_0000_1110_0101_0110_0000_0100_0001_1000 ("9.223"). The result is saved in register file address 4'b01010 and 4'b01011.

Test Case #4: Addition function test on all zero numbers • Set ui_fpu_fs = 5'b00000. • Set ui_fpu_ft = 5'b00001. • Set ui_fpu_sp = 1'b1. • Set ui_fpu_add = 1'b1. • Set ui_fpu_wr = 1'b1. • Set ui_fpu_wr_addr = 5'b01110. • Hold for 4 clock cycle.	Test the addition function on single precision floating point numbers.	uo_fpu _result = 32'b0_0000_0000_0000_0000_0000_0000_0000_000. The result is saved in register file address 4'b01110.
Test Case #5: Infinity inputs test • Set ui_fpu_fs = 5'b00000. • Set ui_fpu_ft = 5'b00001. • Set ui_fpu_sp = 1'b1. • Set ui_fpu_add = 1'b0. • Set ui_fpu_wr = 1'b0. • Hold for 3 clock cycle.	Check the validity of inputs whether inputs are infinity.	uo_fpu _inf_a is asserted.
Test Case #6: NaN inputs test • Set ui_fpu_fs = 5'b10000. • Set ui_fpu_ft = 5'b10010. • Set ui_fpu_sp = 1'b0. • Set ui_fpu_add = 1'b0. • Set ui_fpu_wr = 1'b0. • Hold for 3 clock cycle.	Check the validity of inputs whether inputs are "Not a Number".	uo_fpu _nan_b is asserted.

Table 6-1-T1: Test Plan of FPU

6-2 Simulation Result for FPU

6-2-1 Test Case #1: Reset

Figure 6-2-1-F1: Simulation result for test case #1.

1. After reset signal is asserted, all output signals are set to default state.

6-2-2 Test Case #2: Addition function test on single precision numbers

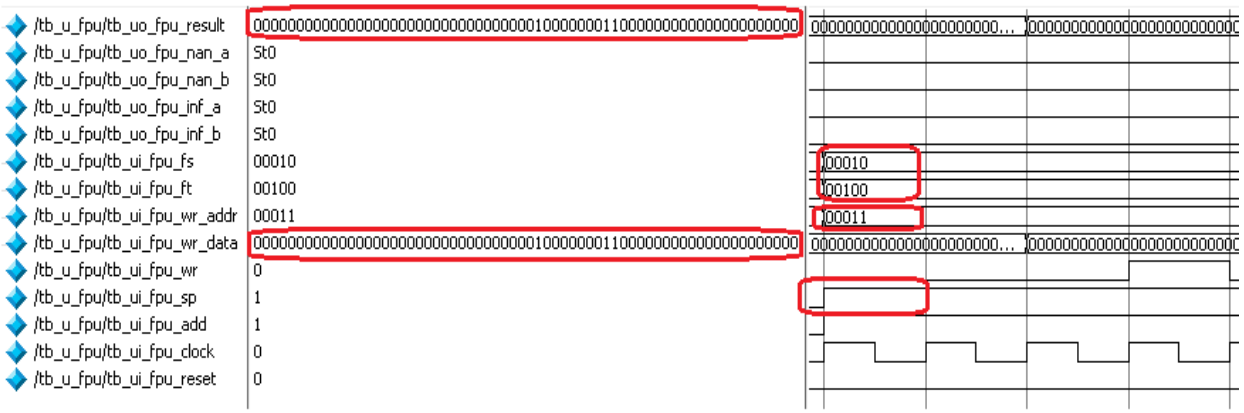


Figure 6-2-2-F1: Simulation result for test case #2.

1. Control signal ui_fpu_sp is asserted indicates that the two data from register file is in single precision.
2. The result of addition of the two single precision numbers is computed.
3. Control signal ui_fpu_wr is asserted to enable the result is written back to location 00011 in register file.

6-2-3 Test Case #3: Addition function test on double precision numbers

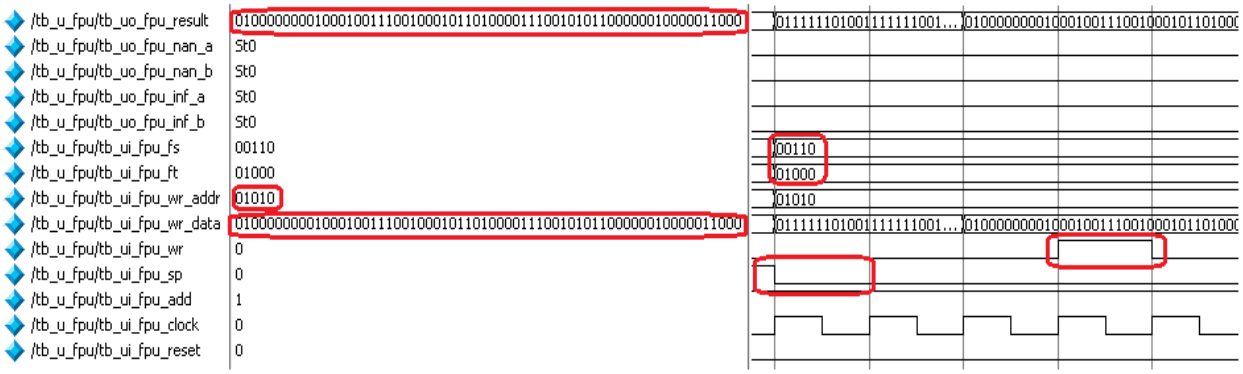


Figure 6-2-3-F1: Simulation result for test case #3.

1. Control signal ui_fpu_sp is de-asserted indicates that the two data from register file is in double precision.
2. The result of addition of the two double precision numbers is computed.
3. Control signal ui_fpu_wr is asserted to enable the result is written back to location 01010 and 01011 in register file.

6-2-4 Test Case #4: Addition function test on all zero numbers

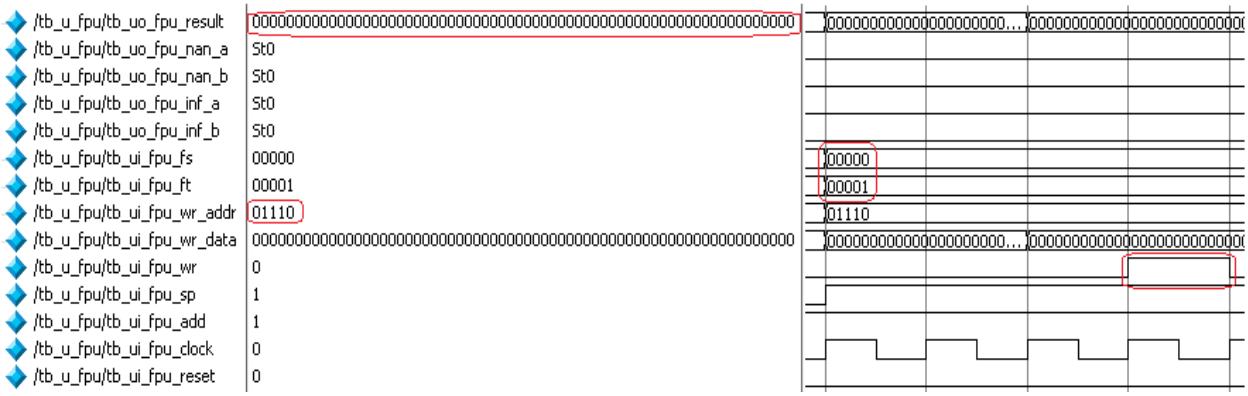


Figure 6-2-4-F1: Simulation result for test case #4.

1. The two all zero inputs are fetched to single precision block to perform addition.
2. Control signal ui_fpu_wr is asserted to enable the result is written back to location 01110 in register file.

6-2-5 Test Case #5: Infinity inputs test

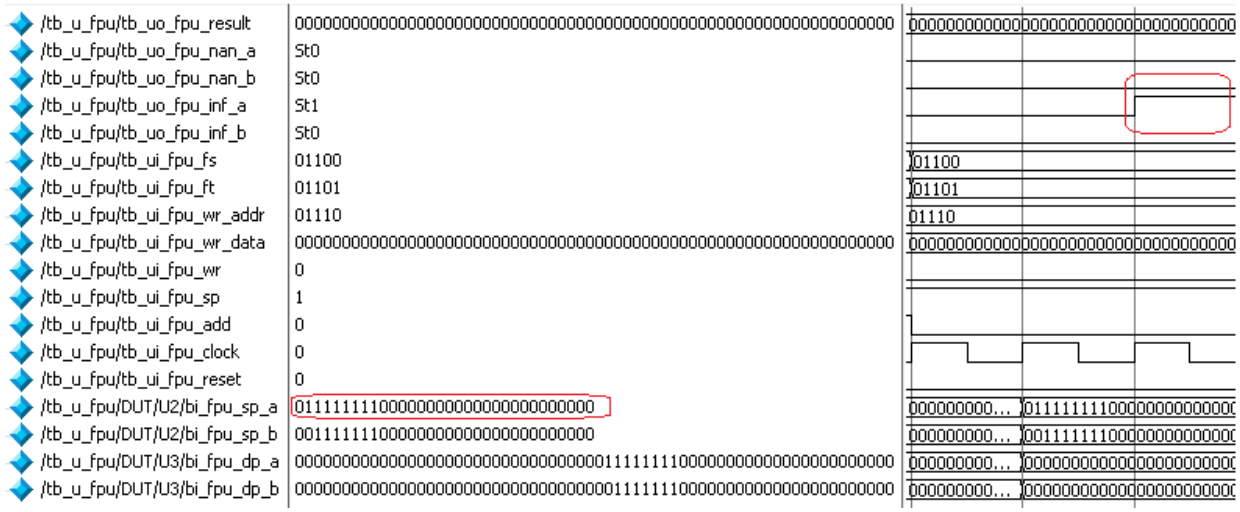


Figure 6-2-5-F1: Simulation result for test case #5.

1. The input operand a is an infinity floating point value.
2. The output signal uo_fpu_inf_a is asserted to indicate it's an infinity value.

6-2-6 Test Case #5: NaN inputs test

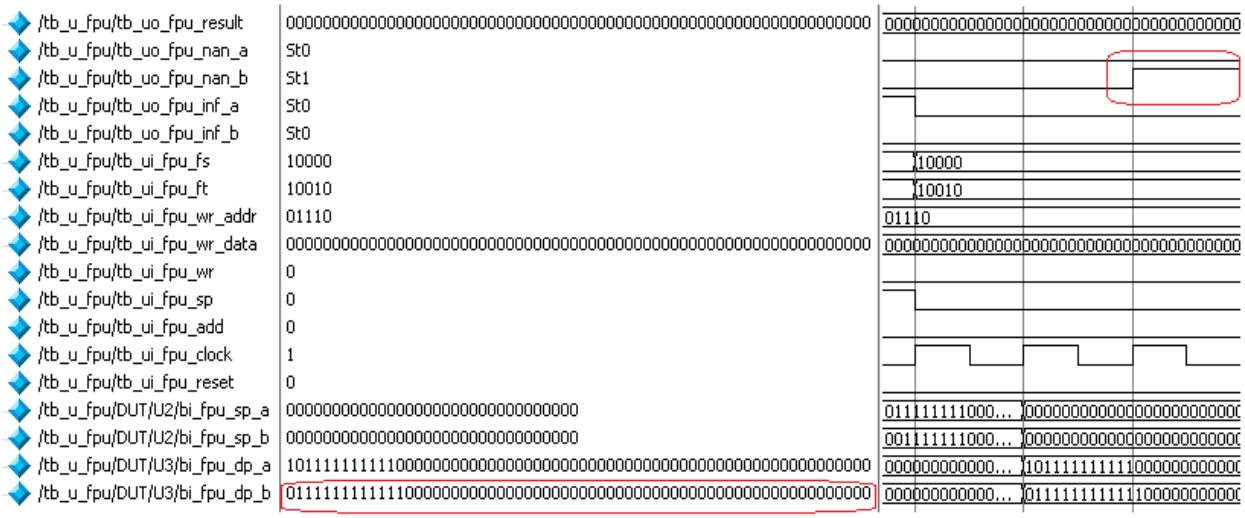


Figure 6-2-6-F1: Simulation result for test case #6.

1. The input operand b is a "Not a Number" in floating point format.
2. The output signal uo_fpu_nan_b is asserted to indicate it's a NaN.

6-3 FP Register File Contents

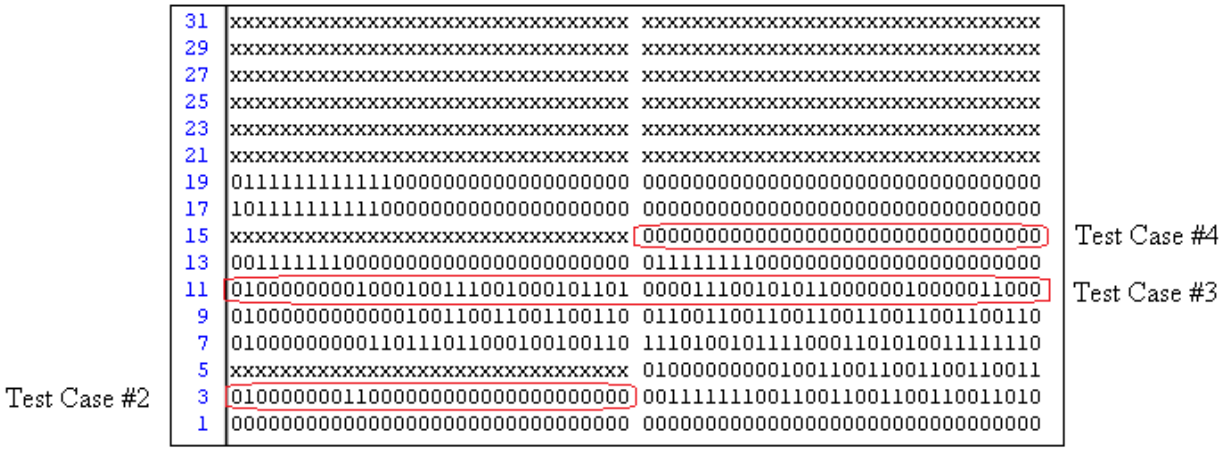


Figure 6-3-F1: FP register file contents

The FPU register file stores the result from test case #2, #3 and #4. The FPU register file can store single and double precision floating numbers. For double precision floating point numbers, two registers are required to store the value.

6-4 Test Bench for FPU

```
//#####################################################
/*
Project/Module : tb_u_fpu
File name : tb_u_fpu.v
Version : Altera 6.5b
Author : Low Wai Hau
Code type :
Description : Testbench for FPU
*/
//#####################################################

module tb_u_fpu();

  wire [63:0] tb_uo_fpu_result;
  wire tb_uo_fpu_nan_a, tb_uo_fpu_nan_b, tb_uo_fpu_inf_a, tb_uo_fpu_inf_b;

  reg [4:0] tb_ui_fpu_fs, tb_ui_fpu_ft, tb_ui_fpu_wr_addr;
  reg [63:0] tb_ui_fpu_wr_data;
  reg tb_ui_fpu_wr, tb_ui_fpu_sp, tb_ui_fpu_add, tb_ui_fpu_clock, tb_ui_fpu_reset;


  //*********************************************************
  // Module instantiation
  u_fpu
  DUT
  (.uo_fpu_result(tb_uo_fpu_result),
   .uo_fpu_nan_a(tb_uo_fpu_nan_a),
   .uo_fpu_nan_b(tb_uo_fpu_nan_b),
   .uo_fpu_inf_a(tb_uo_fpu_inf_a),
   .uo_fpu_inf_b(tb_uo_fpu_inf_b),
   .ui_fpu_fs(tb_ui_fpu_fs),
   .ui_fpu_ft(tb_ui_fpu_ft),
   .ui_fpu_wr_addr(tb_ui_fpu_wr_addr),
   .ui_fpu_wr_data(tb_ui_fpu_wr_data),
   .ui_fpu_wr(tb_ui_fpu_wr),
   .ui_fpu_sp(tb_ui_fpu_sp),
   .ui_fpu_add(tb_ui_fpu_add),
   .ui_fpu_clock(tb_ui_fpu_clock),
   .ui_fpu_reset(tb_ui_fpu_reset));


   initial tb_ui_fpu_clock = 1;
   always #10 tb_ui_fpu_clock = ~tb_ui_fpu_clock;
```

```
  always @(*) begin
    tb_ui_fpu_wr_data = tb_uo_fpu_result;
  end

  //*********************************************************
  //Signals initialization
  initial begin
    tb_ui_fpu_reset = 1'b0;
    tb_ui_fpu_fs = 5'b0;
    tb_ui_fpu_ft = 5'b0;
    tb_ui_fpu_wr_addr = 5'b0;
    tb_ui_fpu_wr_data = 64'b0;
    tb_ui_fpu_wr = 1'b0;
    tb_ui_fpu_sp = 1'b0;
    tb_ui_fpu_add = 1'b0;

    //Test Case #1: Reset
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_reset = 1;
    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_reset = 0;

    //Test Case #2: Addition on single precision numbers
    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_fs = 5'b00010;
    tb_ui_fpu_ft = 5'b00100;
    tb_ui_fpu_wr_addr = 5'b00011;
    tb_ui_fpu_sp = 1'b1;
    tb_ui_fpu_add = 1'b1;

    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_wr = 1'b1;
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_wr = 1'b0;

    //Test Case #3: Addition on double precision numbers
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_fs = 5'b00110;
    tb_ui_fpu_ft = 5'b01000;
    tb_ui_fpu_wr_addr = 5'b01010;
    tb_ui_fpu_sp = 1'b0;
    tb_ui_fpu_add = 1'b1;

    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_wr = 1'b1;
    @(posedge tb_ui_fpu_clock);
```

```
    tb_ui_fpu_wr = 1'b0;

    //Test Case #4: Addition on all zero numbers
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_fs = 5'b00000;
    tb_ui_fpu_ft = 5'b00001;
    tb_ui_fpu_wr_addr = 5'b01110;
    tb_ui_fpu_sp = 1'b1;
    tb_ui_fpu_add = 1'b1;

    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_wr = 1'b1;
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_wr = 1'b0;

    //Test Case #5: Infinity floating point numbers
    @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_fs = 5'b01100;
    tb_ui_fpu_ft = 5'b01101;
    tb_ui_fpu_sp = 1'b1;
    tb_ui_fpu_add = 1'b0;

    //Test Case #6: NaN floating point numbers
    repeat(3) @(posedge tb_ui_fpu_clock);
    tb_ui_fpu_fs = 5'b10000;
    tb_ui_fpu_ft = 5'b10010;
    tb_ui_fpu_sp = 1'b0;
    tb_ui_fpu_add = 1'b0;

    repeat(5) @(posedge tb_ui_fpu_clock);

    $stop;

  end

endmodule
```

6-5 FP Integration with RISC32

6-5-1 Test Program

```
        .text 0x00400000
        .globl main

main:       addi    $t0, $zero, 0x0
            addi    $t1, $zero, 0xff7fffff      #largest floating point number
            addi    $t2, $zero, 0xff7fffff      #smallest floating point number
            addi    $t3, $zero, 0xaaaaaaaa      #$t0 = 101…101
            addi    $t4, $zero, 0x55555555      #$t0 = 010…010
            addi    $t5, $zero, 0x1
            addi    $t6, $zero, 0xf0f0f0f0      #$t0 = 1111_0000…1111_0000
            addi    $t7, $zero, 0x0f0f0f0f      #$t0 = 0000_1111…0000_1111

            sw      $t0, 0($gp)                 #store to data memory to test lwc1
            sw      $t1, 4($gp)
            sw      $t2, 8($gp)
            sw      $t3, 12($gp)
            sw      $t4, 16($gp)
            sw      $t5, 20($gp)
            sw      $t6, 24($gp)
            sw      $t7, 28($gp)

            nop

#test case #1: Test lwc1 and swc1
            lwc1    $f4, 0($gp)                 #load data to FP reg file
            swc1    $f4, 0($gp)                 #store data to data memory
            lwc1    $f5, 4($gp)
            swc1    $f5, 28($gp)
            lwc1    $f6, 8($gp)
            lwc1    $f7,12($gp)
            lwc1    $f8, 16($gp)
            lwc1    $f9, 20($gp)
            nop

#test case #2: Test add.s and add.d
            add.s   $f10, $f4, $f5              #$f10 = 0xff7fffff + 0
            add.s   $f16, $f10, $f9             #$f16 = 0xff7fffff + 1
            add.s   $f17, $f7, $f6              #$f17 = 0x55555555 + 0xaaaaaaaa
            add.s   $f18, $f6, $f5              #$f17 = max fp number + min fp number
```

```
                add.d    $f18, $f4, $f6
                add.d    $f10, $f18, $f6
                nop

#test case #3: Test mfc1 and mtc1
                mfc1 $s6, $f10                    #move data between registers
                mtc1 $t1, $f10                    #move data between registers
                nop
```

6-5-2 Simulation Result

6-5-2-1 Test Case #1: lwc1 instruction

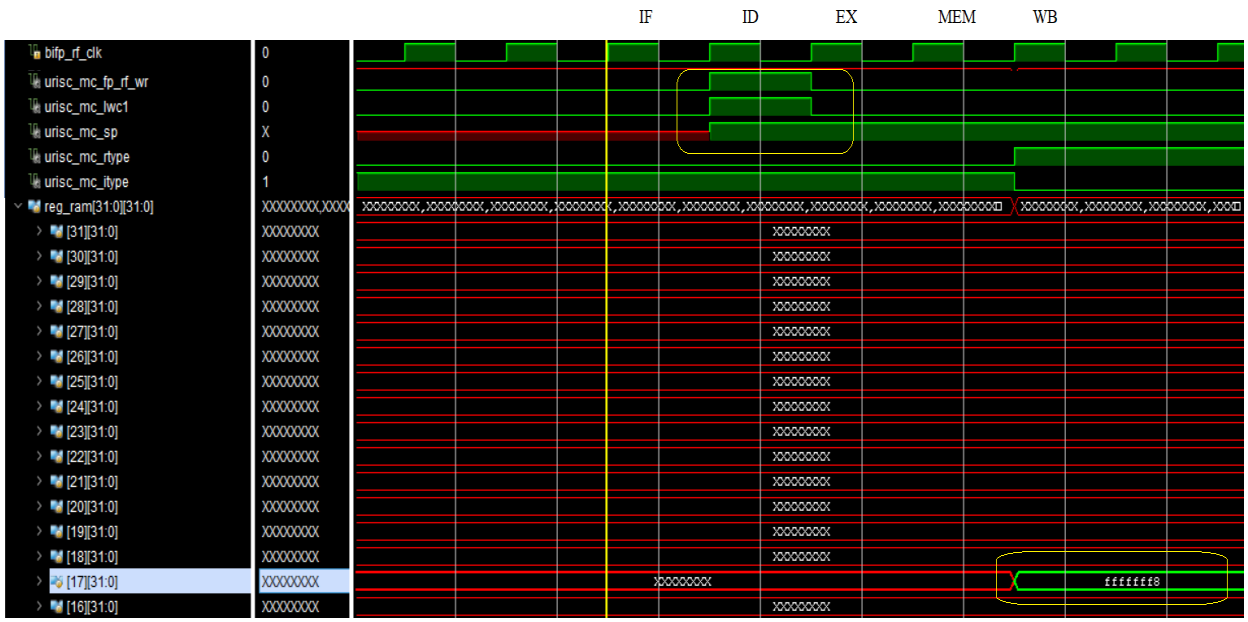


Figure 6-5-2-1-F1: Simulation result of test case #1(lwc1).

1. Control signal lwcl is asserted in ID stage indicate that it is lwc1 instruction and FP register file write is enabled so the data can be written to FP register file.
2. At the WB stage, the data (0xfffffff8) from data memory is written to FP register file.

6-5-2-2 Test Case #1: swc1 instruction

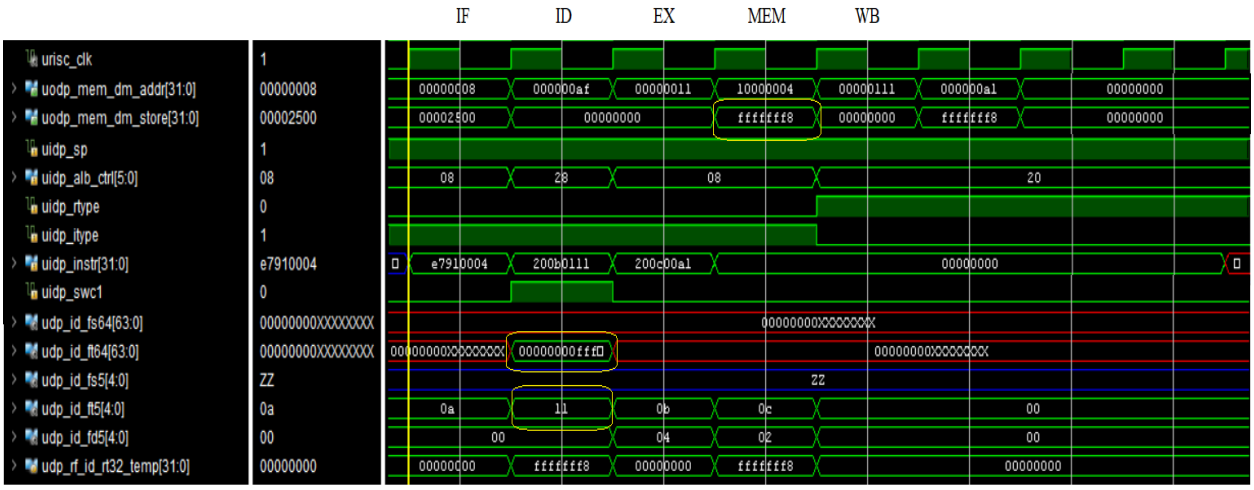


Figure 6-5-2-2-F1: Simulation result of test case #1(swc1).

1. The data is fetched out from general register file at ID stage.
2. The data is then written into data memory at MEM stage.

6-5-2-3 Test Case #3: mfc1 instruction

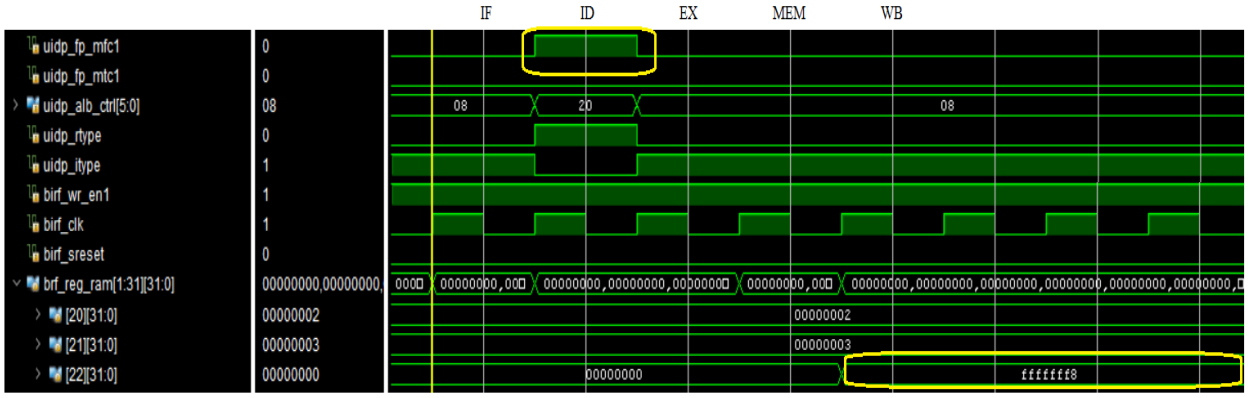


Figure 6-5-2-3-F1: Simulation result of test case #3(mfc1).

1. Control signal udp_fp_mfc1 is enabled at ID stage to indicate that its mfc1 instruction.
2. The data is fetched out from FP register file at ID stage.
3. At WB stage, the data is written to general register file.

6-5-2-4: Test Case #3: mtc1 instruction

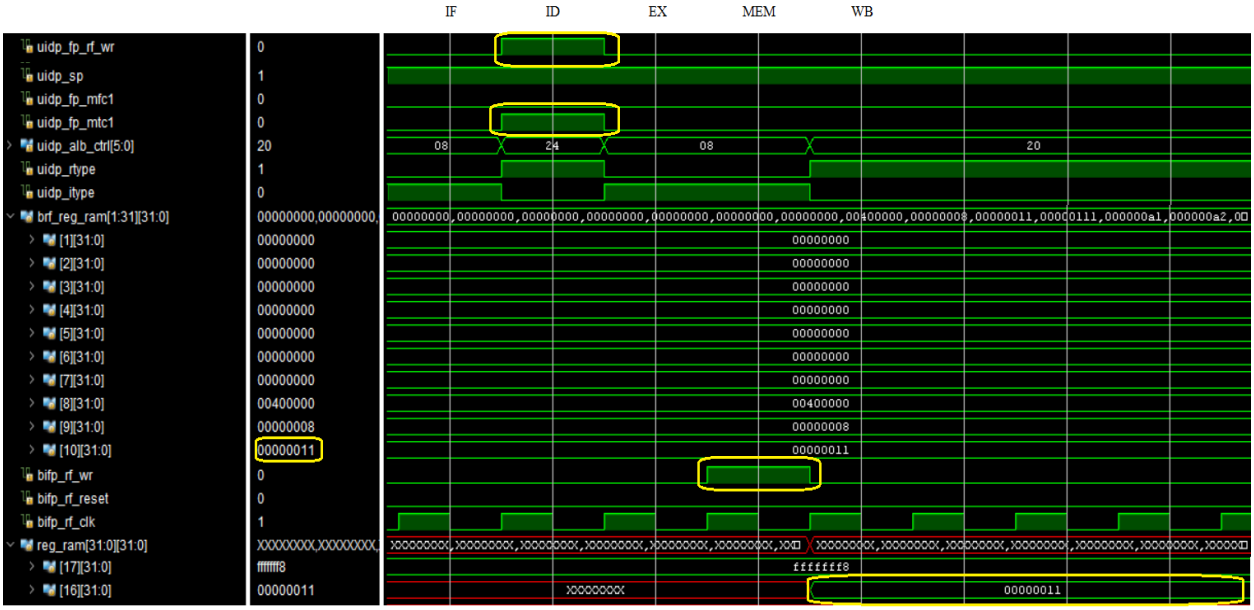


Figure 6-5-2-4-F1: Simulation result of test case #3(mtc1).

1. Control signal udp_fp_mtc1 is enabled at ID stage to indicate that its mtc1 instruction.
2. The data in the general register file is fetched out.
3. The control signal bfp_rf_wr is enabled so that the data is written to the FP register file.

6-5-3 Test Bench

```
`timescale 1ns / 1ps
`default_nettype none
`define FULL_INSTR_TEST 1
//`define demo001_GPIO 1
//`define demo002_GPIO 1
//`define demo003_UART 1
//`define demo004_UART 1
//`define demo005_SPI 1
//`define demo006_DUT_client_model 1
//`define demo115_IoT_MM64bytes

`ifdef MODEL_TECH
        `ifdef FULL_INSTR_TEST
                `define  TEST_CODE_PATH_DUT "tb/fpu_instr_test.txt"
                `define  EXC_HANDLER_DUT "tb/demo/demo000_base_test_03exc_handler.txt"
                `define  TEST_CODE_PATH_CLIENT "tb/demo/demo000_base_test_02program.txt"
     `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
        `endif
        `ifdef demo001_GPIO
                `define  TEST_CODE_PATH_DUT "tb/demo/demo001_GPIO_mem_02program.txt"
                `define  EXC_HANDLER_DUT "tb/demo/demo001_GPIO_mem_03exc_handler.txt"
                `define  TEST_CODE_PATH_CLIENT
"tb/demo/demo006_pending_for_int_mem_02program.txt"
  `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
        `endif
        `ifdef demo002_GPIO
                `define  TEST_CODE_PATH_DUT "tb/demo/demo002_GPIO_mem_02program.txt"
                `define  EXC_HANDLER_DUT "tb/demo/demo002_GPIO_mem_03exc_handler.txt"
                `define  TEST_CODE_PATH_CLIENT
"tb/demo/demo006_pending_for_int_mem_02program.txt"
     `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
        `endif
        `ifdef demo003_UART
                `define  TEST_CODE_PATH_DUT "tb/demo/demo003_UART_mem_02program.txt"
                `define  EXC_HANDLER_DUT "tb/demo/demo003_UART_mem_03exc_handler.txt"
                `define  TEST_CODE_PATH_CLIENT
"tb/demo/demo006_pending_for_int_mem_02program.txt"
     `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
        `endif
        `ifdef demo004_UART
                `define  TEST_CODE_PATH_DUT "tb/demo/demo004_UART_mem_02program.txt"
                `define  EXC_HANDLER_DUT "tb/demo/demo004_UART_mem_03exc_handler.txt"
```

```
                `define  TEST_CODE_PATH_CLIENT
"tb/demo/demo006_pending_for_int_mem_02program.txt"
      `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
         `endif
         `ifdef demo006_DUT_client_model
      `define TEST_CODE_PATH_DUT "tb/demo/demo006_pending_for_int_mem_02program.txt"
      `define EXC_HANDLER_DUT "tb/new_exc_handler_dut.txt"
      `define TEST_CODE_PATH_CLIENT "tb/demo/demo006_all_IOs_send_mem_02program.txt"
      `define EXC_HANDLER_CLIENT "tb/new_exc_handler_client.txt"
         `endif
         `ifdef demo115_IoT_MM64bytes
      `define  TEST_CODE_PATH_DUT "tb/demo/demo115_IoT_MM64bytes_10MHz_02program.txt"
      `define  EXC_HANDLER_DUT "tb/new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"tb/demo/demo006_pending_for_int_mem_02program.txt"
      `define  EXC_HANDLER_CLIENT "tb/new_exc_handler_dut.txt"
         `endif
`else
         `ifdef FULL_INSTR_TEST
                `define  TEST_CODE_PATH_DUT "fpu_instr_test.txt"
        `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
        `define  TEST_CODE_PATH_CLIENT "full_instr_test.txt"
        `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
         `endif
         `ifdef demo001_GPIO
                `define  TEST_CODE_PATH_DUT "demo001_GPIO_mem_02program.txt"
                `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
      `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
         `endif
         `ifdef demo002_GPIO
                `define  TEST_CODE_PATH_DUT "demo002_GPIO_mem_02program.txt"
                `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
      `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
         `endif
         `ifdef demo003_UART
                `define  TEST_CODE_PATH_DUT "demo003_UART_mem_02program.txt"
                `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
      `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
```

```
        `endif
        `ifdef demo004_UART
                `define  TEST_CODE_PATH_DUT "demo004_UART_mem_02program.txt"
                `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
     `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
        `endif
        `ifdef demo005_SPI
     `define  TEST_CODE_PATH_DUT "demo005_SPI_mem_02program.txt"
     `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
     `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
  `endif
        `ifdef demo006_DUT_client_model
     `define TEST_CODE_PATH_DUT "demo006_pending_for_int_mem_02program.txt"
     `define EXC_HANDLER_DUT "new_exc_handler_dut.txt"
     `define TEST_CODE_PATH_CLIENT "demo006_all_IOs_send_mem_02program.txt"
     `define EXC_HANDLER_CLIENT "new_exc_handler_client.txt"
        `endif
        `ifdef demo115_IoT_MM64bytes
     `define  TEST_CODE_PATH_DUT "demo115_IoT_MM64bytes_10MHz_02program.txt"
     `define  EXC_HANDLER_DUT "new_exc_handler_dut.txt"
                `define  TEST_CODE_PATH_CLIENT
"demo006_pending_for_int_mem_02program.txt"
     `define  EXC_HANDLER_CLIENT "new_exc_handler_dut.txt"
  `endif
`endif

module tb_r32_pipeline();
//declaration
//===== INPUT =====
//System signal
reg    tb_u_clk;
reg    tb_u_rst;
//~~~~~~~~~~~~~~~~~~
wire   tb_u_spi_mosi_dut;
wire   tb_u_spi_miso_dut;
wire   tb_u_spi_sclk_dut;
wire   tb_u_spi_ss_n_dut;

//wire   tb_u_fc_sclk_dut;
//wire   tb_u_fc_ss_dut;
```

```
//wire    tb_u_fc_MOSI_dut;
//wire    tb_u_fc_MISO1_dut;
//wire    tb_u_fc_MISO2_dut;
//wire    tb_u_fc_MISO3_dut;

wire    tb_ua_tx_rx_dut;
wire    tb_ua_RTS_dut, tb_ua_CTS_dut;
wire[31:0]  tb_u_GPIO_dut;

//~~~~~~~~~~~~~~~~~~
wire    tb_u_spi_mosi_client;
wire    tb_u_spi_miso_client;
wire    tb_u_spi_sclk_client;
wire    tb_u_spi_ss_n_client;

//wire    tb_u_fc_sclk_client;
//wire    tb_u_fc_ss_client;
//wire    tb_u_fc_MOSI_client;
//wire    tb_u_fc_MISO1_client;
//wire    tb_u_fc_MISO2_client;
//wire    tb_u_fc_MISO3_client;

wire    tb_ua_tx_rx_client;
wire    tb_ua_RTS_client, tb_ua_CTS_client;
wire[31:0]  tb_u_GPIO_client;

crisc c_risc_dut(
//*********** INSTANTIATION *************
//======= INPUT =======
//GPIO
.urisc_GPIO(tb_u_GPIO_dut),

//SPI controller
.uiorisc_spi_mosi(tb_u_spi_mosi_dut),
.uiorisc_spi_miso(tb_u_spi_miso_dut),
.uiorisc_spi_sclk(tb_u_spi_sclk_dut),
.uiorisc_spi_ss_n(tb_u_spi_ss_n_dut),

//UART controller
.uorisc_ua_tx_data(tb_ua_tx_rx_dut),
.uorisc_ua_rts(tb_ua_RTS_dut),
.uirisc_ua_rx_data(tb_ua_tx_rx_client),
.uirisc_ua_cts(tb_ua_CTS_dut),
```

```
//FLASH controller
//.uorisc_fc_sclk(tb_u_fc_sclk_dut),
//.uiorisc_fc_MOSI(tb_u_fc_MOSI_dut),
//.uirisc_fc_MISO1(tb_u_fc_MISO1_dut),
//.uirisc_fc_MISO2(tb_u_fc_MISO2_dut),
//.uirisc_fc_MISO3(tb_u_fc_MISO3_dut),
//.uorisc_fc_ss(tb_u_fc_ss_dut),

// System signal
.uirisc_clk_100mhz(tb_u_clk),
.uirisc_rst(tb_u_rst));


//================================================================================
==============

crisc c_risc_client(
//************ INSTANTIATION **************
//======= INPUT =======
//GPIO
.urisc_GPIO(tb_u_GPIO_client),

//SPI controller
.uiorisc_spi_mosi(tb_u_spi_mosi_client),
.uiorisc_spi_miso(tb_u_spi_miso_client),
.uiorisc_spi_sclk(tb_u_spi_sclk_client),
.uiorisc_spi_ss_n(tb_u_spi_ss_n_client),

//UART controller
.uorisc_ua_tx_data(tb_ua_tx_rx_client),
.uorisc_ua_rts(tb_ua_RTS_client),
.uirisc_ua_rx_data(tb_ua_tx_rx_dut),
.uirisc_ua_cts(tb_ua_CTS_client),

// System signal
.uirisc_clk_100mhz(tb_u_clk),
.uirisc_rst(tb_u_rst));

`ifdef demo006_DUT_client_model
   assign tb_u_spi_mosi_dut = tb_u_spi_mosi_client;
   assign tb_u_spi_miso_dut = tb_u_spi_miso_client;
   assign tb_u_spi_ss_n_dut = tb_u_spi_ss_n_client;
   assign tb_u_spi_sclk_dut = tb_u_spi_sclk_client;
`else
```

```
    assign tb_u_spi_mosi_client = tb_u_spi_mosi_dut;
    assign tb_u_spi_miso_client = tb_u_spi_miso_dut;
    assign tb_u_spi_ss_n_client = tb_u_spi_ss_n_dut;
    assign tb_u_spi_sclk_client = tb_u_spi_sclk_dut;
`endif

assign tb_ua_CTS_dut = tb_ua_RTS_client;
assign tb_ua_CTS_client = tb_ua_RTS_dut;
//**********************Clock************************
initial tb_u_clk = 1'b1;
always #5 tb_u_clk =~ tb_u_clk;

initial begin
    $readmemh(`TEST_CODE_PATH_DUT/*"demo200_Test_SDRAM_Write_02program.txt"*/,
tb_r32_pipeline.c_risc_dut.sdram.Bank1);
    $readmemh(`EXC_HANDLER_DUT, tb_r32_pipeline.c_risc_dut.sdram.Bank1);
    $readmemh(`TEST_CODE_PATH_CLIENT, tb_r32_pipeline.c_risc_client.sdram.Bank1);
    $readmemh(`EXC_HANDLER_CLIENT, tb_r32_pipeline.c_risc_client.sdram.Bank1);
    tb_u_rst = 1'b1;
    repeat(1)@(posedge tb_u_clk);
    tb_u_rst = 1'b0;
    repeat(30000)@(posedge tb_u_clk);
    tb_u_rst = 1'b1;
    repeat(12000000)@(posedge tb_r32_pipeline.c_risc_dut.urisc_clk);
end
endmodule
```

CHAPTER 7: CONCLUSION

A FPU module and FP Register File has been successfully modeled and integrated into RISC32 microprocessor. All the behavior has been verified. The FPU module able to perform addition operation on single and double precision numbers as well as load and store to FP Register File. The flow of addition operation is being mentioned in Chapter 2 of this project.

The integration of FPU module and FP Register File into RISC32 architecture has been accomplished, as shown in previous chapters. The FPU is modeled using Verilog HDL. The design is implemented with top-down design methodology approach. The top level of FPU is first determined and then followed by specifications of block level inside FPU. The data can be load or store to FP register file. FPU is verified with test plan and test bench and the functionality of FPU is proven to be working well.

For future improvement, other FP instruction can be implemented to the FPU module such as subtraction, multiplication and division. Also, can implement exception handling for overflow output.

BIBLIOGRAPHY

Al-Eryani J 2006, 'IEEE Standard 754 for Binary Floating-Point Arithmetic', *Floating Point Unit*, pp. 3-9. Available from: [01 April 2019].

Goldberg D 1991, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. Available from: < https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf>. [01 April 2019].

Integrated Device Technology, Inc. 1994, *IDT R30xx Family Software Reference Manual*.

Kukati S, Sujana DV, Udaykumar S, Jayakrishnan P & Dhanabal R 2013, *Design and Implementation of Low Power Floating Point Arithmetic Unit*. Available from: < https://ieeexplore-ieee-org.libezp2.utar.edu.my/stamp/stamp.jsp?tp=&arnumber=6823429>. [01 April 2019].

Lundgren D 2014, Double Precision Floating Point Core VHDL, pp. 1-8. Available from: <https://opencores.org/projects/fpu_double>. [01 April 2019].

Mok K.M 2018, *Digital System Design*, lecture notes distributed in Faculty of Information and Communication Technology at University of Tunku Abdul Rahman.

Mok K.M 2019, *Computer Organization and Architecture*, lecture notes distributed in Faculty of Information and Communication Technology at University of Tunku Abdul Rahman.

Patterson DA and Hennesy JL 2005, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Francisco, CA: Morgan Kaufmann. [01 April 2019].

Patterson DA and Hennesy JL 2014, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Francisco, CA: Morgan Kaufmann. [01 April 2019].

Singh P and Bhole K 2014, 'Optimized floating point arithmetic unit', *2014 Annual IEEE India Conference (INDICON)*, Pune, pp. 1-4. Available from: < https://ieeexplore-ieee-

org.libezp2.utar.edu.my/stamp/stamp.jsp?tp=&arnumber=7030552&isnumber=7030354>. [01 April 2019].

POSTER

University Tunku Abdul Rahman

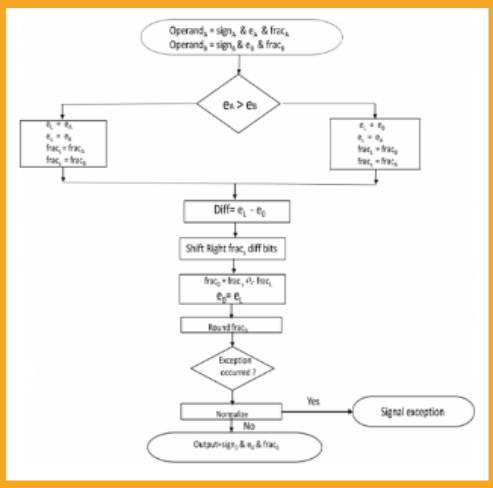
Design of a Floating Point Unit for 5 Stage Pipeline Processor

By Low Wai Hau
Supervised by: Mr. Mok Kai Ming

Introduction

There is MIPS-compatible ISA that consists the Central Processing Unit (CPU), PS/2 keyboard system, PS/2 mouse system, coprocessor 0 (CP0), UART and basic memory. However, Floating Point Unit (FPU) has not been designed and integrated in RISC32 yet. This project is to develop a FPU and integrate FPU into RISC32 processor.

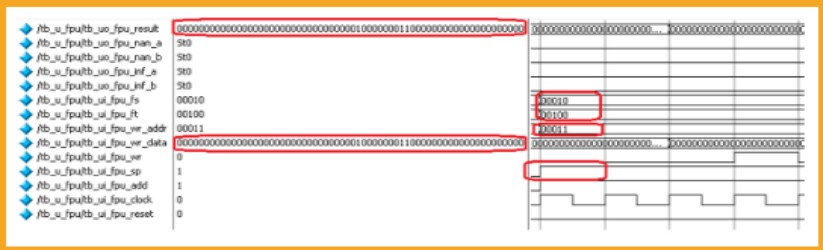
Addition Algorithm



Project Objectives

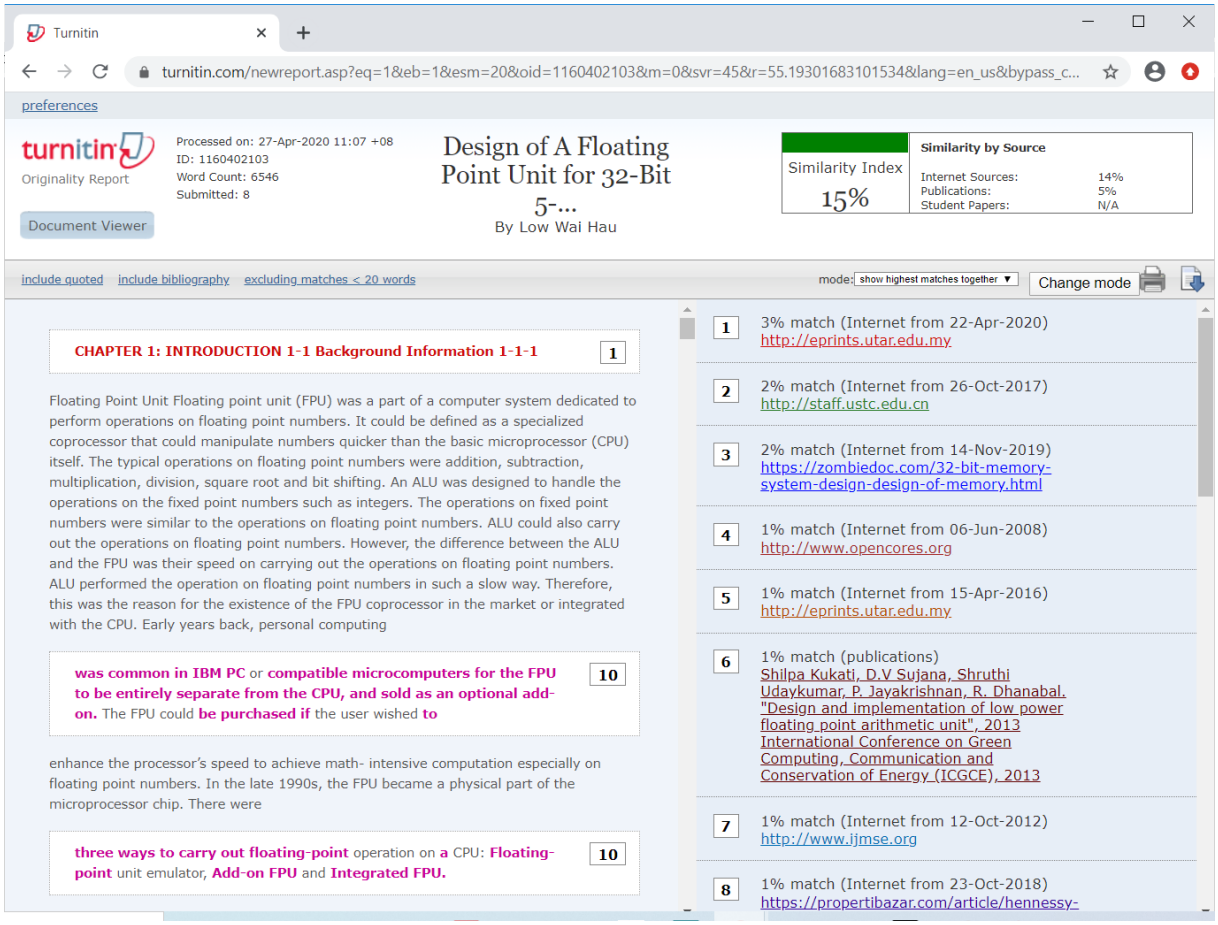
- To design and develop the RTL model of FPU which include microarchitecture specification and testbench.
- To integrate the FPU into RISC32 processor.
- To synthesis the FPU module on FPGA with completes documented timing and resource usage information.

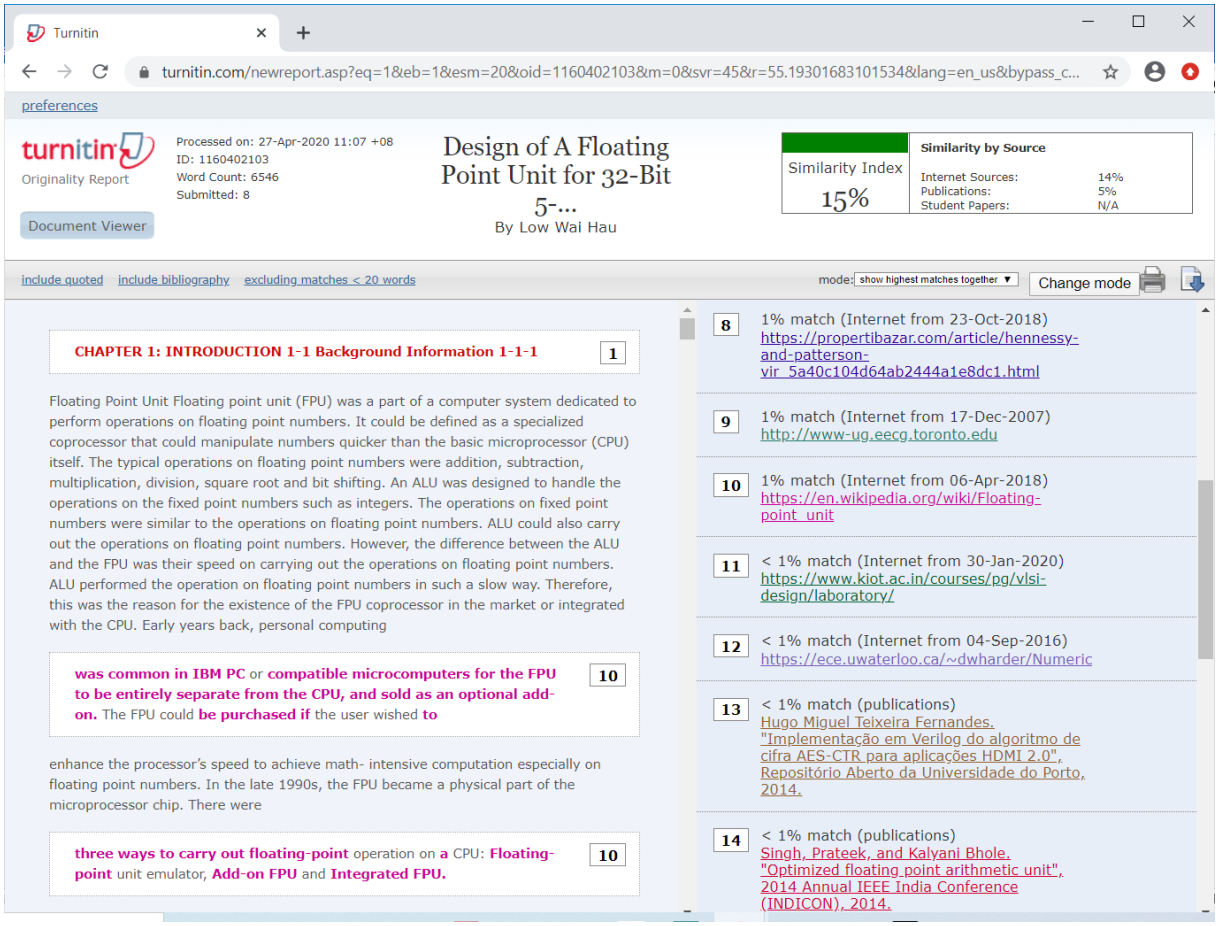
Result

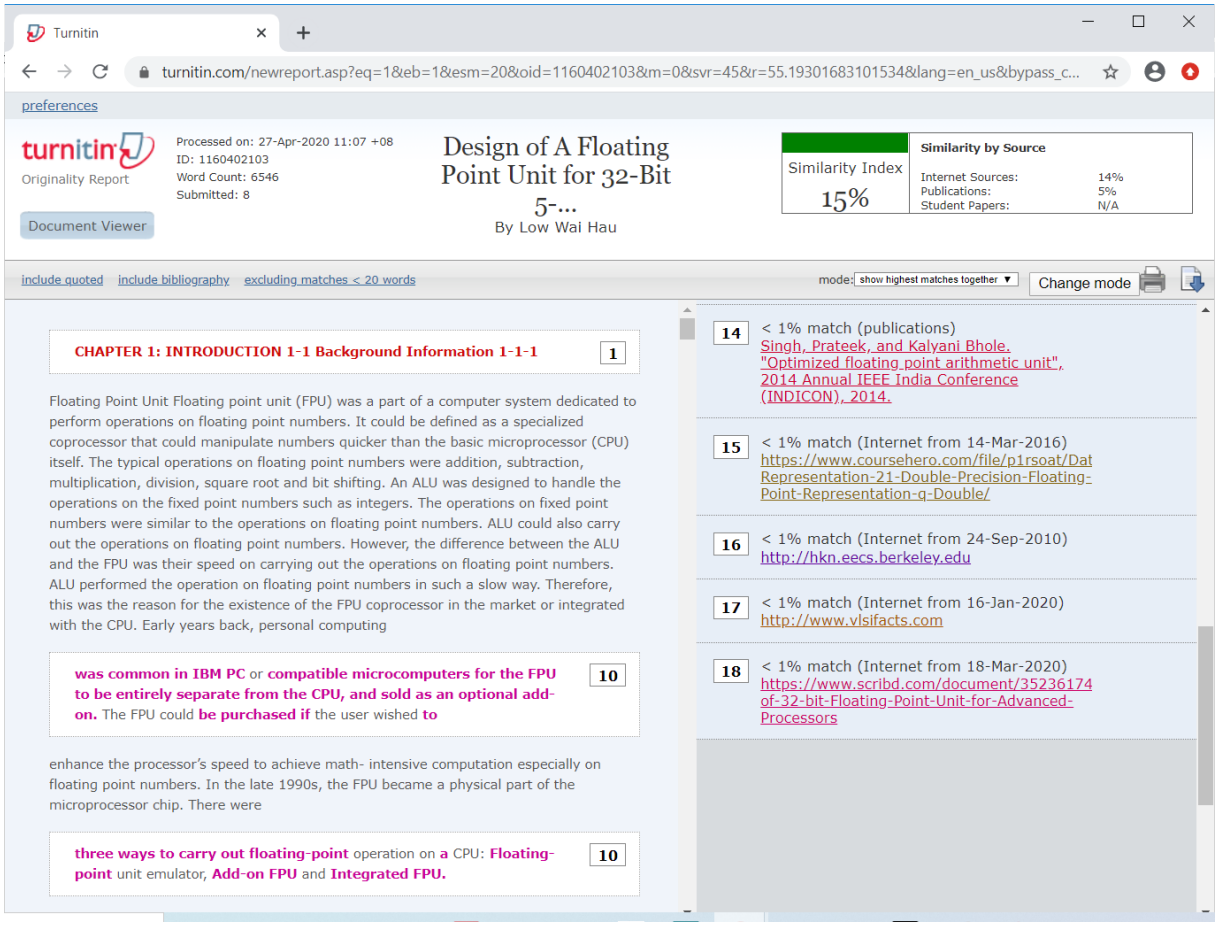


Addition of Floating point numbers.

PLAGIARISM CHECK RESULT




Turnitin
turnitin.com/newreport.asp?eq=1&eb=1&esm=20&oid=1160402103&m=0&svr=45&r=55.19301683101534&lang=en_us&bypass_c...
preferences
turnitin
Originality Report
Processed on: 27-Apr-2020 11:07 +08
ID: 1160402103
Word Count: 6546
Submitted: 8
Document Viewer
Design of A Floating Point Unit for 32-Bit 5-...
By Low Wai Hau
Similarity Index
15%
Similarity by Source
Internet Sources: 14%
Publications: 5%
Student Papers: N/A
include quoted
include bibliography
excluding matches < 20 words
mode: show highest matches together
Change mode
CHAPTER 1: INTRODUCTION 1-1 Background Information 1-1-1 1
Floating Point Unit Floating point unit (FPU) was a part of a computer system dedicated to perform operations on floating point numbers. It could be defined as a specialized coprocessor that could manipulate numbers quicker than the basic microprocessor (CPU) itself. The typical operations on floating point numbers were addition, subtraction, multiplication, division, square root and bit shifting. An ALU was designed to handle the operations on the fixed point numbers such as integers. The operations on fixed point numbers were similar to the operations on floating point numbers. ALU could also carry out the operations on floating point numbers. However, the difference between the ALU and the FPU was their speed on carrying out the operations on floating point numbers. ALU performed the operation on floating point numbers in such a slow way. Therefore, this was the reason for the existence of the FPU coprocessor in the market or integrated with the CPU. Early years back, personal computing
was common in IBM PC or compatible microcomputers for the FPU to be entirely separate from the CPU, and sold as an optional add-on. The FPU could be purchased if the user wished to 10
enhance the processor's speed to achieve math- intensive computation especially on floating point numbers. In the late 1990s, the FPU became a physical part of the microprocessor chip. There were
three ways to carry out floating-point operation on a CPU: Floating-point unit emulator, Add-on FPU and Integrated FPU. 10
8 1% match (Internet from 23-Oct-2018) https://propertibazar.com/article/hennessy-and-patterson-vir_5a40c104d64ab2444a1e8dc1.html
9 1% match (Internet from 17-Dec-2007) http://www-ug.eecg.toronto.edu
10 1% match (Internet from 06-Apr-2018) https://en.wikipedia.org/wiki/Floating-point_unit
11 < 1% match (Internet from 30-Jan-2020) https://www.kiot.ac.in/courses/pg/vlsi-design/laboratory/
12 < 1% match (Internet from 04-Sep-2016) https://ece.uwaterloo.ca/~dwharder/Numeric
13 < 1% match (publications) Hugo Miguel Teixeira Fernandes. "Implementação em Verilog do algoritmo de cifra AES-CTR para aplicações HDMI 2.0", Repositório Aberto da Universidade do Porto, 2014.
14 < 1% match (publications) Singh, Prateek, and Kalyani Bhole. "Optimized floating point arithmetic unit", 2014 Annual IEEE India Conference (INDICON), 2014.


Turnitin
turnitin.com/newreport.asp?eq=1&eb=1&esm=20&oid=1160402103&m=0&svr=45&r=55.19301683101534&lang=en_us&bypass_c...
preferences
turnitin
Originality Report
Processed on: 27-Apr-2020 11:07 +08
ID: 1160402103
Word Count: 6546
Submitted: 8
Document Viewer
Design of A Floating Point Unit for 32-Bit 5-...
By Low Wai Hau
Similarity Index
15%
Similarity by Source
Internet Sources: 14%
Publications: 5%
Student Papers: N/A
include quoted
include bibliography
excluding matches < 20 words
mode: show highest matches together
Change mode
CHAPTER 1: INTRODUCTION 1-1 Background Information 1-1-1
1
Floating Point Unit Floating point unit (FPU) was a part of a computer system dedicated to perform operations on floating point numbers. It could be defined as a specialized coprocessor that could manipulate numbers quicker than the basic microprocessor (CPU) itself. The typical operations on floating point numbers were addition, subtraction, multiplication, division, square root and bit shifting. An ALU was designed to handle the operations on the fixed point numbers such as integers. The operations on fixed point numbers were similar to the operations on floating point numbers. ALU could also carry out the operations on floating point numbers. However, the difference between the ALU and the FPU was their speed on carrying out the operations on floating point numbers. ALU performed the operation on floating point numbers in such a slow way. Therefore, this was the reason for the existence of the FPU coprocessor in the market or integrated with the CPU. Early years back, personal computing
was common in IBM PC or compatible microcomputers for the FPU to be entirely separate from the CPU, and sold as an optional add-on. The FPU could be purchased if the user wished to
10
enhance the processor's speed to achieve math- intensive computation especially on floating point numbers. In the late 1990s, the FPU became a physical part of the microprocessor chip. There were
three ways to carry out floating-point operation on a CPU: Floating-point unit emulator, Add-on FPU and Integrated FPU.
10
14 < 1% match (publications)
Singh, Prateek, and Kalyani Bhole. "Optimized floating point arithmetic unit", 2014 Annual IEEE India Conference (INDICON), 2014.
15 < 1% match (Internet from 14-Mar-2016)
https://www.coursehero.com/file/p1rsoat/Dat Representation-21-Double-Precision-Floating-Point-Representation-q-Double/
16 < 1% match (Internet from 24-Sep-2010)
http://hkn.eecs.berkeley.edu
17 < 1% match (Internet from 16-Jan-2020)
http://www.vlsifacts.com
18 < 1% match (Internet from 18-Mar-2020)
https://www.scribd.com/document/35236174 of-32-bit-Floating-Point-Unit-for-Advanced-Processors

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	LOW WAI HAU
ID Number(s)	16ACB05712
Programme / Course	CT
Title of Final Year Project	Design of A Floating Point Unit For 32-Bits 5 Stage Pipeline Processor

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 15 % **Similarity by source** Internet Sources: 14 % Publications: 5 % Student Papers: N/A %	
Number of individual sources listed of more than 3% similarity: 0	

Parameters of originality required and limits approved by UTAR are as Follows:

(i) Overall similarity index is 20% and below, and

(ii) Matching of individual sources listed must be less than 3% each, and

(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: MOK KAI MING

Date: 24-04-2020

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	16ACB05712
Student Name	LOW WAI HAU
Supervisor Name	MR. MOK KAI MING

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Front Cover
√	Signed Report Status Declaration Form
√	Title Page
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. **LOW WAI HAU** (Signature of Student) Date: 24-04-2020	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. (Signature of Supervisor) Date: 24-04-2020