|  |
| --- |
| 32-bit FLOATING POINT MULTIPLIER IMPLEMENTATION |

*Version 1.0 (Wednesday, April 17, 2019)*

Preface:

We[[1]](#footnote-0) have created this multiplier to handle the multiplication process of two 32-bit IEEE 754 based numbers as a part of FPU project[[2]](#footnote-1).

Content:

1. IEEE 754 representation ....................................................................1
2. The implementation JOURNY ...............................................................2
   1. calculating fraction .................................................................2
   2. Calculating exponent ................................................................
   3. THE REASEONS WHY WE WERE SO LUCKY IMPLEMETING THIS .......
3. Special cases .......................................................................................
4. VERILOG CODE ........................................................................................
5. Sharing the happiness of a working code (results) ...................

APPENDEX a. OUR TOOLKIT

APPENDEX B. REFERENCES

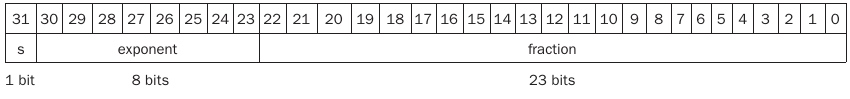
chapter 1:

“IEEE 754” representation

“This kind of refreshing part for the sake of comprehension.”

Floating-Point Representation:

Floating-point numbers are usually a multiple of the size of a word. The representation of a MIPS floating-point number is shown below, where s is the sign of the floating-point number (1 meaning negative), exponent is the value of the 8-bit exponent field (including the sign of the exponent), and fraction is the 23-bit number.



In general, floating-point numbers are of the form

DeepinScreenshot_select-area_20190429043519

These formats go beyond MIPS. They are part of the IEEE 754 floating-point standard, found in virtually every computer invented since 1980. This standard has greatly improved both the ease of porting floating-point programs and the quality of computer arithmetic.

Before we go on board we have to take these notes with us:

1. Number must be normalized in an understandable language ( 1.01000 \* 2^E not 101.000\* 2^E-3 )
2. Exponents are biased by 127 which means 0 = 127 to get the the real exponent form IEEE754 one subtract 127 from it.

chapter 2:

Our implementation journey[[3]](#footnote-2)

“Our implementation wasn’t the most complicated ever but we tend to call it a “journey” because we had explored a new world on-board”

2.1 Calculating fraction:

To calculate fractions we first have to notice that the input has the form of “1.(mantissa)\*2 exponent\*(-1) sign “ in this section we only concerned with the “1.(mantissa)” part we can express it in other form as “1.0 + 0.mantissa->(also knows as fraction we may use these terms interchangeably so don’t get confused)” multiplying each input we can easily notice that the result has the form

|  |
| --- |
| **Formula 1:**  Multiplying each input gives  Result = (1+ fraction1 + fraction2 + fraction1 \* fraction2)  Resulting fraction = Result - 1 or  **( fraction1 + fraction2 + fraction1 \* fraction2 )** |

With the use of formula 1 we can calculate our resulting fraction in a single Verilog line more on this later at (Chapter 4 & Appendix .A)

2.2 Calculating Exponent:

To calculate exponents we just add them together as we learned at the primary school nothing amazing here.

|  |
| --- |
| **Formula 2:**  Resulting exponent = (exponent1 + exponent2) |

1. **Team members:**

   - Hussien Mostafa

   - Abdulrahman Ragab

   - Abdullah Khaled

   - Abdallah Mohamed

   - Mahmoud Hassan

   **Supervisor:**

   - Dr. Muhammad Mahmoud Muhammad Ibrahim [↑](#footnote-ref-0)
2. **FPU project:** the floating point unit to be used in our implementation of MIPS based processor [↑](#footnote-ref-1)
3. side-note: we roughly assume that the reader have the basic knowledge of Algebra and programming basics [↑](#footnote-ref-2)