**Jumpabug**



**MIPS64 Processor Design**

**External Architecture Specification**

John Dawson

256 37th street

Pittsburgh PA, 15201

This page is intentionally blank.

Table of Contents

[1. Introduction 4](#_Toc397422904)

[1.1 References 4](#_Toc397422905)

[1.2 Glossary of Terms 4](#_Toc397422906)

[1.3 Revision History 4](#_Toc397422907)

[2. Overview 5](#_Toc397422908)

[3. Pipeline Detailed Design 6](#_Toc397422909)

[3.1 Instruction Fetch (IF) 6](#_Toc397422910)

[3.2 Instruction Decode (ID) 8](#_Toc397422911)

[3.2.1 Overview 8](#_Toc397422912)

[3.2.2 Register Block 9](#_Toc397422913)

[3.2.3 Control Table 10](#_Toc397422914)

[3.3 Instruction Execute (IE) 12](#_Toc397422915)

[3.3.1 Overview 12](#_Toc397422916)

[3.3.2 ALU 13](#_Toc397422917)

[3.4 Memory Access (MEM) 13](#_Toc397422918)

[3.4.1 Overview 13](#_Toc397422919)

[3.5 Writeback (WB) 13](#_Toc397422920)

[3.5.1 Overview 13](#_Toc397422921)

[3.6 Data Forwarding Unit 13](#_Toc397422922)

[3.7 Hazard Unit 13](#_Toc397422923)

[4. Coprocessor 0 Detailed Design 14](#_Toc397422924)

[4.1 Overview 14](#_Toc397422925)

[4.2 Control Register Set 14](#_Toc397422926)

[5. Coprocessor 1 Detailed Design 15](#_Toc397422927)

[5.1 Overview 15](#_Toc397422928)

[5.2 Control Register Set 15](#_Toc397422929)

# Introduction

This document details the design of the Jumpabug custom 64 bit MIPS soft core processor for use within Altera FPGA devices.

## References

This section outlines the references used for this document.

|  |  |
| --- | --- |
| MIPS64Instruction Set | *http://equipe.nce.ufrj.br/gabriel/arqcomp2/MIPS64\_Instruction\_Set\_v0.95.pdf* |
| Computer Organization and Design |  |
| See Mips Run Linux |  |
|  |  |
|  |  |

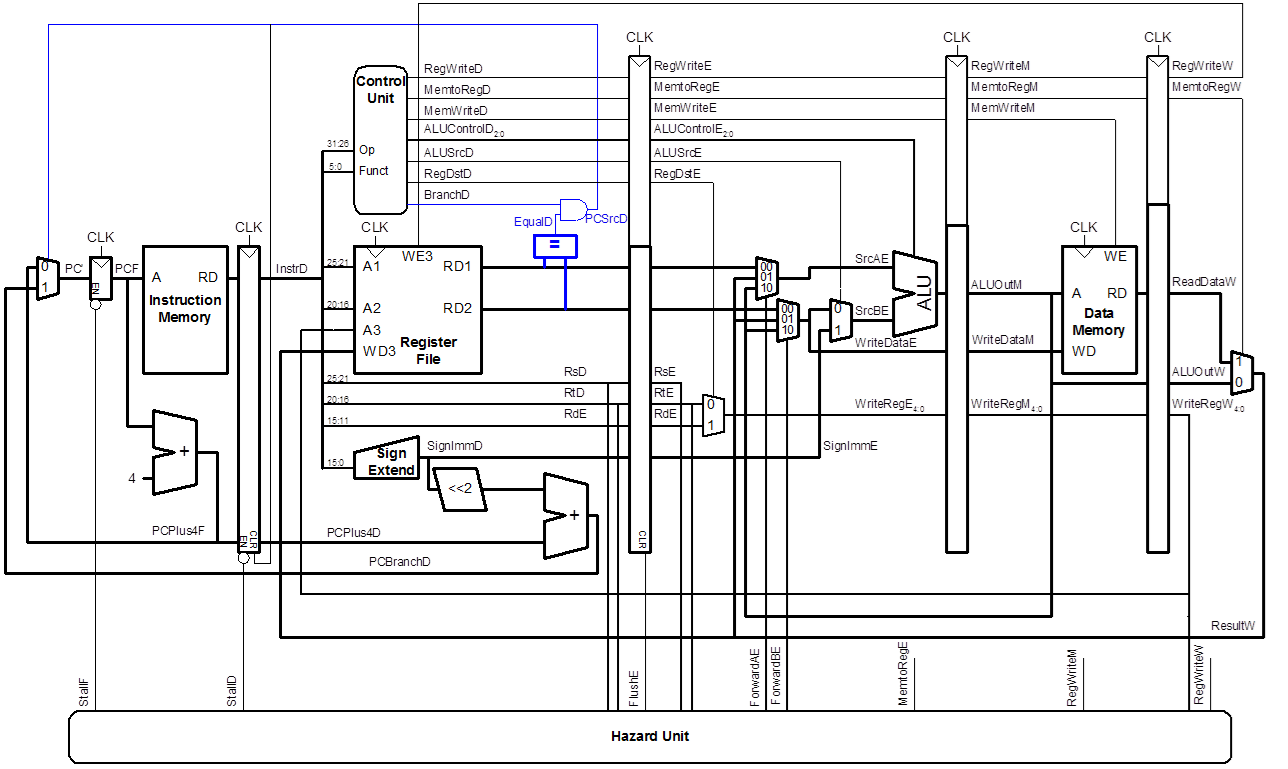
## Glossary of Terms

## Revision History

| **Version** | **Date** | **Changed By** | **Modifications** |
| --- | --- | --- | --- |
| 0.1 | 9Sept02 | J. Dawson | Initial draft. |

# Overview

The Jumpabug MIPS64 IP consists of a single core 64 bit MIPS processor designed for use within Altera brand FPGA devices.



## Main Features

* Five stage pipeline
* Full compliance to MIPS64 ISA V0.95
* Core operating frequency of TBD MHz
* Integrated Coprocessor 0 and Coprocessor 1 support

# Pipeline Detailed Design

## Instruction Fetch (IF)

### Overview

The instruction fetch stage of the Jumpabug core is responsible for requesting the next instruction from memory and passing it to the decode stage. The block continually requests instructions from the instruction memory interface in increments of 4 bytes. On each clock cycle the address programmed into the program counter (PC) is updated from one of the following sources:

* Previous PC Value (Normal Operation)
* Branch/Jump Address
* Exception/Interrupt Address

In the case of a memory access wait condition, the IF block will initiate an IF/ID Pipeline stage stall which will allow the remaining stages to complete while waiting on the next instruction.



Figure : IF Block Diagram (Black - Data, Red - Control)

### Port Descriptions

The IF block design is contained in a single Verilog file: **IF\_Block.v**. The block contains the following I/O ports.

|  |  |  |  |
| --- | --- | --- | --- |
| **I/O Name** | **Direction** | **Size** | **Description** |
| P\_clk | IN | 1 | Core clock |
| P\_rst\_l | IN | 1 | Core reset |
| P\_IF\_INST\_MemDataIn | IN | 32 | Instruction data from memory |
| P\_IF\_INST\_MemWait | IN | 1 | Memory access wait from memory |
| P\_IF\_INST\_MemAddress | OUT | 64 | Instruction address to memory |
| P\_IF\_INST\_MemRead | OUT | 1 | Instruction memory read request |
| P\_IF\_BranchAddress | IN | 64 | Branch address (from MEM stage) |
| P\_IF\_ExceptionAddress | IN | 64 | Exception address (From COPROC 0 ) |
| P\_IF\_PCSrc | IN | 1 | Branch address select |
| P\_IF\_ExceptionSrc | IN | 1 | Exception address select |
| P\_IF\_Instruction | OUT | 32 | Instruction to ID stage |
| P\_IF\_PC\_Counter | OUT | 64 | Current instruction PC value to ID stage |
| P\_IF\_Stall | OUT | 1 | IF stage stall signal |

Table : IF Block I/O

## Instruction Decode (ID)

### Overview



Table : Instruction Decode Block Diagram

The Instruction Decode block is responsible for:

* Decoding the current address
* Generating the correct IE, MEM, and WB stage control signals
* Read out the required register data from the 32x64 register block
* Sign extend any immediate values
* Pass instruction RT and RD fields to the IE stage (Possible writeback Destination)

To accomplish this, the 32 bit instruction from the IF stage is broken down to multiple bit fields as outlined below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction[31:26]** | **Instruction[25:21]** | **Instruction[20:16]** | **Instruction[15:25]** | **Instruction[4:0]** |
| OpCode | Register Address A | Register Address B | Immediate Value | |
|  |  |  |  | Funct |

Table : Instruction Fields

The following table describes each of these fields in more detail:

|  |  |
| --- | --- |
| **Instruction Field** | **Description** |
| OpCode | Instruction operation code, used by the control decoder to determine the instructions ID and assign the correct control signals. |
| Register Address A | Register block port A address |
| Register Address B | Register block port B address |
| Immediate Value | For Immediate instructions this 15 bit value is sign extended and used for the immediate operation. |
| Funct | Instruction function code, used by the control decoder to determine the instructions ID and assign the correct control signals. |

Table : Instruction Field Descriptions

### Port Descriptions

The ID block design is contained in a single Verilog file: **ID\_Block.v**. The block contains the following I/O ports.

|  |  |  |  |
| --- | --- | --- | --- |
| **I/O Name** | **Direction** | **Size** | **Description** |
| **P\_clk** | **IN** | **1** | **Core clock** |
| **P\_rst\_l** | **IN** | **1** | **Core reset** |
| **P\_ID\_IN\_Instruction** | **IN** | **32** | **Instruction from IF stage** |
| **P\_ID\_DataA** | **OUT** | **64** | **Data from port A of the register block** |
| **P\_ID\_DataB** | **OUT** | **64** | **Data from port B of the register block** |
| **P\_ID\_SignExtended** | **OUT** | **64** | **Sign extended immediate to IE stage** |
| **P\_ID\_RT\_RegDest** | **OUT** | **5** | **Instruction RT field for WB destination** |
| **P\_ID\_RD\_RegDest** | **OUT** | **5** | **Instruction RD field for WB destination** |
| **P\_ID\_WB\_Data** | **IN** | **64** | **Writeback data** |
| **P\_ID\_WB\_Addr** | **IN** | **5** | **Writeback address** |
| **P\_ID\_WB\_WE** | **IN** | **1** | **Writeback address enable** |
| **P\_ID\_IE\_Ctrl\_Bus** | **OUT** | **32** | **IE stage control signals** |
| **P\_ID\_MEM\_Ctrl\_Bus** | **OUT** | **32** | **MEM stage control signals** |
| **P\_ID\_WB\_Ctrl\_Bus** | **OUT** | **32** | **WB stage control signals** |

Table : ID Block I/O

The ID block consists of the following two sub components: Register Block and the control table. The next two sections will outline these in more detail.

### Register Block

The ID block contains 32 general purpose 64 bit registers arranged in a single write port, dual read port register block:



Figure : 32x64 Dual Port Register Block

All registers are read/writable with the exception of register 0 which always returns the value zero on a read.

### Control Table

The control table block contains all the logic to decode the current instruction and produce three control bus groups: IE\_control, MEM\_Control, and WB\_Control.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| OpCode | Funct | Branch | Instruction Name | IE Control | Mem Control | WB Control |
| 000000 | 100000 | XXXXX | ADD |  |  |  |
| 001000 | XXXXXX | XXXXX | ADDI |  |  |  |
| 001001 | XXXXXX | XXXXX | ADDIU |  |  |  |
| 000000 | 100001 | XXXXX | ADDU |  |  |  |
| 000000 | 100100 | XXXXX | AND |  |  |  |
| 001100 | XXXXXX | XXXXX | ANDI |  |  |  |
| 000100 | XXXXXX | XXXXX | BEQ |  |  |  |
| 000001 | XXXXXX | XXXXX | BGEZAL |  |  |  |
| 010100 | XXXXXX | XXXXX | BEQL |  |  |  |
| 000001 | XXXXXX | 00001 | BGEZ |  |  |  |
| 000001 | XXXXXX | 10001 | BGEZAL |  |  |  |
| 000001 | XXXXXX | 10011 | BGEZALL |  |  |  |
| 000001 | XXXXXX | 00011 | BGEZL |  |  |  |
| 000111 | XXXXXX | 00000 | BGTZ |  |  |  |
| 010111 | XXXXXX | 00000 | BGTZL |  |  |  |
| 000110 | XXXXXX | 00000 | BLEZ |  |  |  |
| 010110 | XXXXXX | 00000 | BLEZL |  |  |  |
| 000001 | XXXXXX | 00000 | BLTZ |  |  |  |
| 000001 | XXXXXX | 10000 | BLTZAL |  |  |  |
| 000001 | XXXXXX | 10010 | BLTZALL |  |  |  |
| 000001 | XXXXXX | 00010 | BLTZL |  |  |  |
| 000101 | XXXXXX | XXXXX | BNE |  |  |  |
| 010101 | XXXXXX | XXXXX | BNEL |  |  |  |
| 000000 | 001101 | XXXXX | BREAK |  |  |  |
| 011100 | 100001 | XXXXX | CLO |  |  |  |
| 011100 | 100000 | XXXXX | CLZ |  |  |  |
| 000000 | 101100 | XXXXX | DADD |  |  |  |
| 011000 | XXXXXX | XXXXX | DADDI |  |  |  |
| 011001 | XXXXXX | XXXXX | DADDIU |  |  |  |
| 000000 | 101101 | XXXXX | DADDU |  |  |  |
| 011100 | 100101 | XXXXX | DCLO |  |  |  |
| 011100 | 100100 | XXXXX | DCLZ |  |  |  |
| 000000 | 011110 | XXXXX | DDIV |  |  |  |
| 000000 | 011111 | XXXXX | DDIVU |  |  |  |
| 000000 | 011010 | XXXXX | DIV |  |  |  |
| 000000 | 011011 | XXXXX | DIVU |  |  |  |
| 000000 | 011100 | XXXXX | DMULT |  |  |  |
| 000000 | 011101 | XXXXX | DMULTU |  |  |  |
| 000000 | 111000 | XXXXX | DSLL |  |  |  |
| 000000 | 111100 | XXXXX | DSLL32 |  |  |  |
| 000000 | 010100 | XXXXX | DSSLV |  |  |  |
| 000000 | 111011 | XXXXX | DSRA |  |  |  |
| 000000 | 111111 | XXXXX | DSRA32 |  |  |  |
| 000000 | 010111 | XXXXX | DSRAV |  |  |  |
| 000000 | 111010 | XXXXX | DSRL |  |  |  |
| 000000 | 111110 | XXXXX | DSRL32 |  |  |  |
| 000000 | 010110 | XXXXX | DSRLV |  |  |  |
| 000000 | 101110 | XXXXX | DSUB |  |  |  |
| 000000 | 101111 | XXXXX | DSUBU |  |  |  |
| 000010 | XXXXXX | XXXXX | J |  |  |  |
| 000011 | XXXXXX | XXXXX | JAL |  |  |  |
| 000000 | 001001 | XXXXX | JALR |  |  |  |
| 000000 | 001000 | XXXXX | JR |  |  |  |
| 100000 | XXXXXX | XXXXX | LB |  |  |  |
| 100100 | XXXXXX | XXXXX | LBU |  |  |  |
| 110111 | XXXXXX | XXXXX | LD |  |  |  |
| 011010 | XXXXXX | XXXXX | LDL |  |  |  |
| 011011 | XXXXXX | XXXXX | LDR |  |  |  |
| 100001 | XXXXXX | XXXXX | LH |  |  |  |
| 100101 | XXXXXX | XXXXX | LHU |  |  |  |
| 110000 | XXXXXX | XXXXX | LL |  |  |  |
| 110100 | XXXXXX | XXXXX | LLD |  |  |  |
| 001111 | XXXXXX | XXXXX | LUI |  |  |  |
| 100011 | XXXXXX | XXXXX | LW |  |  |  |
| 100010 | XXXXXX | XXXXX | LWL |  |  |  |
| 100110 | XXXXXX | XXXXX | LWR |  |  |  |
| 100111 | XXXXXX | XXXXX | LWU |  |  |  |
| 011100 | XXXXXX | XXXXX | MADD |  |  |  |
| 011100 | XXXXXX | XXXXX | MADDU |  |  |  |
| 000000 | 010000 | XXXXX | MFHI |  |  |  |
| 000000 | 010010 | XXXXX | MFLO |  |  |  |
| 000000 | 001011 | XXXXX | MOVN |  |  |  |
| 000000 | 001010 | XXXXX | MOVZ |  |  |  |
| 011100 | 000100 | XXXXX | MSUB |  |  |  |
| 011100 | 000101 | XXXXX | MSUBU |  |  |  |
| 000000 | 010001 | XXXXX | MTHI |  |  |  |
| 000000 | 010011 | XXXXX | MTLO |  |  |  |
| 011100 | 000010 | XXXXX | MUL |  |  |  |
| 000000 | 011000 | XXXXX | MULT |  |  |  |
| 000000 | 011001 | XXXXX | MULTU |  |  |  |
| 000000 | 000000 | XXXXX | NOP |  |  |  |
| 000000 | 100111 | XXXXX | NOR |  |  |  |
| 000000 | 100101 | XXXXX | OR |  |  |  |
| 001101 | XXXXXX | XXXXX | ORI |  |  |  |
| 101000 | XXXXXX | XXXXX | SB |  |  |  |
| 111000 | XXXXXX | XXXXX | SC |  |  |  |
| 111100 | XXXXXX | XXXXX | SCD |  |  |  |
| 111111 | XXXXXX | XXXXX | SD |  |  |  |
| 011100 | 111111 | XXXXX | SDBBP |  |  |  |
| 101100 | XXXXXX | XXXXX | SDL |  |  |  |
| 101101 | XXXXXX | XXXXX | SDR |  |  |  |
| 101001 | XXXXXX | XXXXX | SH |  |  |  |
| 000000 | 000000 | XXXXX | SLL |  |  |  |
| 000000 | 000100 | XXXXX | SLLV |  |  |  |
| 000000 | 101010 | XXXXX | SLT |  |  |  |
| 001010 | XXXXXX | XXXXX | SLTI |  |  |  |
| 001011 | XXXXXX | XXXXX | SLTIU |  |  |  |
| 000000 | 101011 | XXXXX | SLTU |  |  |  |
| 000000 | 000011 | XXXXX | SRA |  |  |  |
| 000000 | 000111 | XXXXX | SRAV |  |  |  |
| 000000 | 000010 | XXXXX | SRL |  |  |  |
| 000000 | 000110 | XXXXX | SRLV |  |  |  |
| 000000 | 100010 | XXXXX | SUB |  |  |  |
| 000000 | 100011 | XXXXX | SUBU |  |  |  |
| 101011 | XXXXXX | XXXXX | SW |  |  |  |
| 101010 | XXXXXX | XXXXX | SWL |  |  |  |
| 101110 | XXXXXX | XXXXX | SWR |  |  |  |
| 000000 | 001100 | XXXXX | SYSCALL |  |  |  |
| 000000 | 110100 | XXXXX | TEQ |  |  |  |
| 000001 | XXXXXX | 01100 | TEQI |  |  |  |
| 000000 | 110000 | XXXXX | TGE |  |  |  |
| 000001 | XXXXXX | 01000 | TGEI |  |  |  |
| 000001 | XXXXXX | 01001 | TGEIU |  |  |  |
| 000000 | 110010 | XXXXX | TLT |  |  |  |
| 000001 | XXXXXX | 01010 | TLTI |  |  |  |
| 000001 | XXXXXX | 01011 | TLTIU |  |  |  |
| 000000 | 110011 | XXXXX | TLTU |  |  |  |
| 000000 | 110110 | XXXXX | TNE |  |  |  |
| 000001 | XXXXXX | 01110 | TNEI |  |  |  |
| 000000 | 100110 | XXXXX | XOR |  |  |  |
| 001110 | XXXXXX | XXXXX | XORI |  |  |  |

Figure : CPU Instruction Decode table

## Instruction Execute (IE)

### Overview

The Instruction Execute stage of pipeline is responsible for executing the following functions in the MIPS pipeline:

* All Arithmetic Operations
* Branch Condition Calculations
* Jump Condition Calculations
* Memory Address Calculations



### IE Stage Control Signals

The 32 bit control bus generated by the control decoder in the ID stage is broken out into the following control signals in the IE block.

|  |  |  |
| --- | --- | --- |
| IE Stage Control Bus | IE Stage Signal | Description |
| IE\_Control[3:0] | W\_IE\_ALUop | ALU OpCode. Selects which ALU operation to perform |
| IE\_Control[4] | W\_IE\_RegDest | Selects between the RT and RD instruction fields for the writeback result address. |
| IE\_Control[5] | W\_IE\_ALUSRC | Selects between register data B or the sign extended immediate as the second ALU input |
| IE\_Control[6] | W\_IE\_ShiftInputSwitch | Selects between register data A or register data B as the first ALU input. |
| IE\_Control[7] | W\_IE\_SHAMTSel | Selects between register data A or the sign extended immediate (bits [10:6] ) as the ALU shift input |
| IE\_Control[8] | W\_IE\_BranchSel |  |
| IE\_Control[9] | W\_IE\_ReverseB |  |
| IE\_Control[10] | W\_IE\_DoubleOp | If asserted, IE stage operation is a Doubleword (64 bit ) operation |
| IE\_Control[11] | W\_IE\_32OffsetShift | Shift amount equals (SHAMT + 32) to allow for up to 63 bit shift amounts. |

Table : IE Stage Control Signals

### Arithmetic Operations

The IE stage of the MIPS pipeline is capable of performing the following arithmetic operations:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Description | Latency | Class | Size | Notes |
| ADD | Add two registers | 1 |  | 32 |  |
| ADDI | Add immediate to register | 1 |  | 32 |  |
| ADDIU | Add immediate unsigned to register | 1 |  | 32 |  |
| ADDU | Add two unsigned integer registers | 1 |  | 32 |  |
| AND | And two registers | 1 |  | 32 |  |
| ANDI | And register with immediate value | 1 |  | 32 |  |
| CLO | Count leading 1’s | ? |  | 32 |  |
| CLZ | Count leading 0’s | ? |  | 32 |  |
| DADD | Doubleword add | 1 |  | 64 |  |
| DADDI | Doubleword add immediate | 1 |  | 64 |  |
| DADDIU | Doubleword add immediate unsigned | 1 |  | 64 |  |
| DADDU | Doubleword add unsigned | 1 |  | 64 |  |
| DCLO | Doubleword count leading ones | ? |  | 64 |  |
| DCLZ | Doubleword count leading zeros | ? |  | 64 |  |
| DDIV | Doubleword divide | ? |  | 64 |  |
| DDIVU | Doubleword divide unsigned | ? |  | 64 |  |
| DIV | Divide | ? |  | 64 |  |
| DMULT | Doubleword multiplication | ? |  | 64 |  |
| DMULTU | Doubleword unsigned multiplication | ? |  | 64 |  |
| DSSL | Doubleword shift left logical | 1 |  | 64 |  |
| DSSL32 | Doubleword shift left logical > 32 | 1 |  | 64 |  |
| DSSLV | Doubleword shift left logical variable | 1 |  | 64 |  |
| DSRA | Doubleword shift right arithmetic | 1 |  | 64 |  |
| DSRA32 | Doubleword shift right arithmetic > 32 | 1 |  | 64 |  |
| DSRL | Doubleword shift right logical | 1 |  | 64 |  |
| DSRL32 | Doubleword shift right logical > 32 | 1 |  | 64 |  |
| DSRLV | Doubleword shift right logical variable | 1 |  | 64 |  |
| DSUB | Doubelword subtract | 1 |  | 64 |  |
| DSUBU | Doubleword unsigned subtract | 1 |  | 64 |  |
| MADD | Multiply and add | ? |  | 32 |  |
| MADDU | Multiply and add unsigned | ? |  | 32 |  |
| MSUB | Multiply and subtract | ? |  | 32 |  |
| MSUBU | Multiply and subtract unsigned | ? |  | 32 |  |
| MUL | Multiply | ? |  | 32 |  |
| MULT | Multiply | ? |  | 32 |  |
| MULTU | Multiply unsigned | ? |  | 32 |  |
| NOR | Not or | 1 |  | 32 |  |
| OR | Or | 1 |  | 32 |  |
| ORI | Or immediate | 1 |  | 32 |  |
| SLL | Shift left logical | 1 |  | 32 |  |
| SLLV | Shift left logical variable | 1 |  | 32 |  |
| SLT | Set less than | 1 |  | 32 |  |
| SLTI | Set less than immediate | 1 |  | 32 |  |
| SLTIU | Set less than immediate unsigned | 1 |  | 32 |  |
| SLTU | Set less than unsigned | 1 |  | 32 |  |
| SRA | Shift right arithmetic | 1 |  | 32 |  |
| SRAV | Shift right arithmetic variable | 1 |  | 32 |  |
| SRL | Shift right logical | 1 |  | 32 |  |
| SRLV | Shift right variable | 1 |  | 32 |  |
| SUB | Subtract | 1 |  | 32 |  |
| SUBU | Unsigned subtract | 1 |  | 32 |  |
| XOR | XOR | 1 |  | 32 |  |
| XORI | XORI | 1 |  | 32 |  |

Table : IE Stage Operations

### ALU

The central component in the IE stage used to perform all arithmetic, logical, shift, and comparison operations is the Arithmetic Logic Unit (ALU) which can be found in the ALU.v file.



Figure : IE Stage ALU

The table below summarizes the ALU I/O ports.

|  |  |  |  |
| --- | --- | --- | --- |
| **I/O Name** | **Direction** | **Size** | **Description** |
| P\_ALUop | Input | 4 | ALU operation control port. Each bit pattern represents a unique ALU operation (See table xxxx) |
| P\_DoubleOp | Input | 1 | ALU operation size port. Selects between 32 bit and 64 bit operations. |
| P\_SHAMT | Input | 5 | Shift input select port |
| P\_A | Input | 64 | Operand A port |
| P\_B | Input | 64 | Operand B Port |
| P\_Zero | Output | 1 | Result zero port (if zero, operation produced a zero result) |
| P\_ALUResult | Output | 64 | ALU operation output port |

The ALU is subdivided into four functional processes, one for each type of operation: arithmetic, logical, comparison and shift. All operation types are performed in parallel and a combination of multiplexers select the correct output given the ALUop control bus.

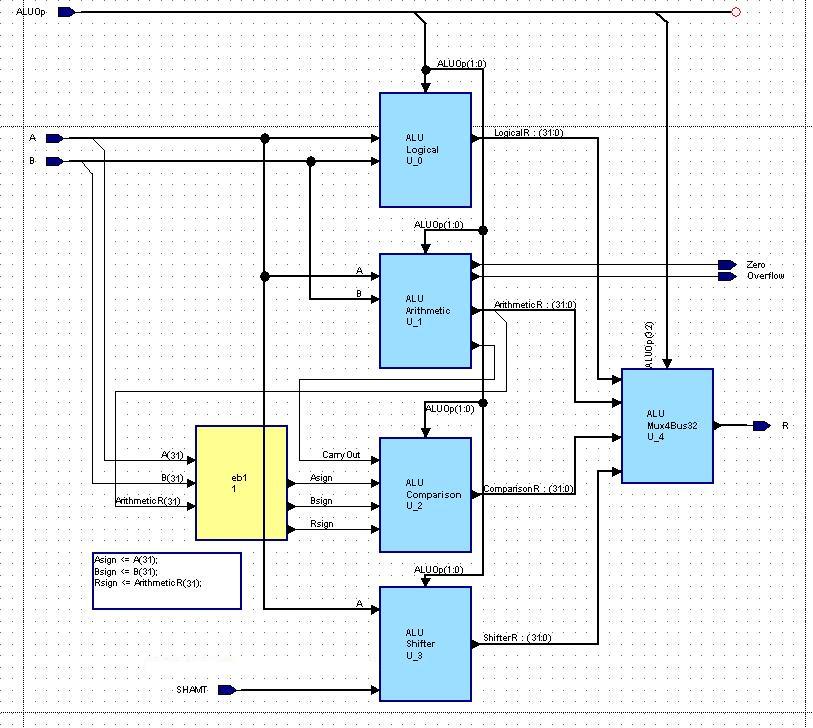


Figure : ALU Subcomponent Organization

### Pipelined Integer Multiplier Unit

#### Generic Unit

#### Altera IP Unit

### Pipelined Integer Divider Unit

#### Generic Unit

#### Altera IP Unit

## Memory Access (MEM)

### Overview

## Writeback (WB)

### Overview

## Data Forwarding Unit

## Hazard Unit

# Coprocessor 0 Detailed Design

## Overview

## Control Register Set

# Coprocessor 1 Detailed Design

## Overview

## Control Register Set