
Raisin64 Documentation

Release 0.1

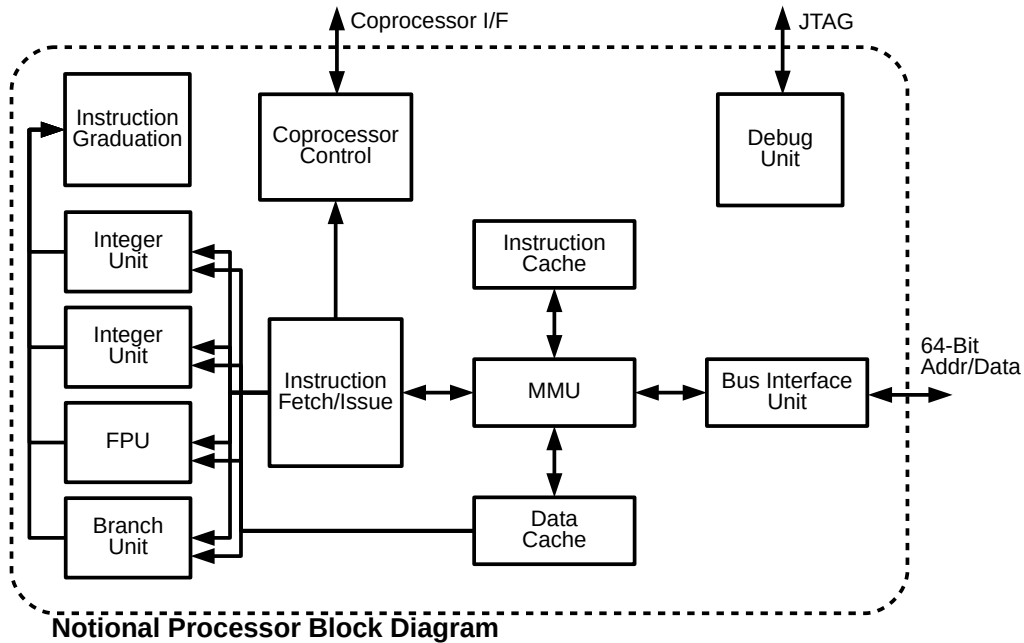
Christopher Parish

Oct 20, 2018

CONTENTS:

1	Raisin64 CPU	3
1.1	Overview	3
1.2	Pipeline Stages	3
1.3	Caches	3
1.4	MMU	3
1.5	Interrupt Unit	3
1.6	Debug Unit	3
2	Code Snippets and Software	5
2.1	Handling Interrupts	5
2.2	Initializing the MMU	5
3	Tools	7
3.1	Assembler	7
3.2	Debugging	7
4	Nexys 4 DDR Reference Implementation	9
4.1	SoC Peripherals	9
4.2	Required Hardware	9
4.3	Synthesizing the Core	9
5	Reference Index	11
5.1	Raisin64 Instruction Set	11
5.2	Verilog Module Index	15

Raisin64 (*RISC Architecture with In-order Superscalar INterlocked-pipeline*) is a pure 64-bit CPU design created as part of an educational project. Architecturally similar to the [MIPS R10000](#) and [POWER3](#), Raisin64 is a superscalar design that employs multiple specialized pipelines for integer operations, floating point, load/store, etc. Unlike most superscalar designs, Raisin64 does not re-order instructions but instead provides a larger architectural register file of 64x64-bit registers.



Major features of the Raisin64 include:

- **Bits:** 64-bit
- **Design:** RISC
- **Type:** Register-Register
- **Branching:** Condition Code
- **Endianness:** Big
- **Page Size:** 16KB Fixed
- **Virtual Address Size:** 47-Bits
- **Page Table:** Three Level
- **Registers:** 61 (R0 = 0)

RAISIN64 CPU

1.1 Overview

1.2 Pipeline Stages

1.3 Caches

1.4 MMU

1.5 Interrupt Unit

1.6 Debug Unit

CODE SNIPPETS AND SOFTWARE

2.1 Handling Interrupts

2.2 Initializing the MMU

- *Assembler*
- *Debugging*
 - *Getting OpenOCD*

3.1 Assembler

3.2 Debugging

3.2.1 Getting OpenOCD

NEXYS 4 DDR REFERENCE IMPLEMENTATION

4.1 SoC Peripherals

4.2 Required Hardware

4.3 Synthesizing the Core

REFERENCE INDEX

5.1 Raisin64 Instruction Set

- *Overview*
- *Instruction Format*
- *Instructions*

5.1.1 Overview

The Raisin64's instruction set draws heavily from MIPS with some concepts graciously borrowed from ARM as well. While the programmer's model and instruction set are decoupled from the underlying microarchitecture of the specific implementation, it was nonetheless decided to design the instructions such that a hardwired control unit (see [TODO:ref:Instruction Decode](#)) could process and set the appropriate signals.

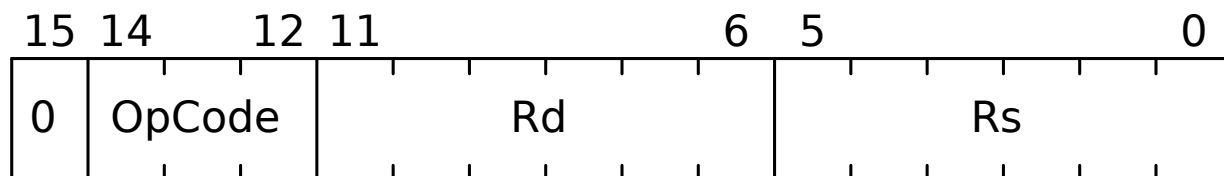
Instructions are variable length (16-64) bit, and some have multiple forms like the *ADD Instruction*. When an instruction has multiple encodings, the opcode is usually the same between the alternate length versions of that instruction, but in all cases the processor expands the 16 and 32-bit versions of the instruction into their canonical 64-bit form, which has a regular encoding. The general instruction formats and opcodes are described below.

But Why?

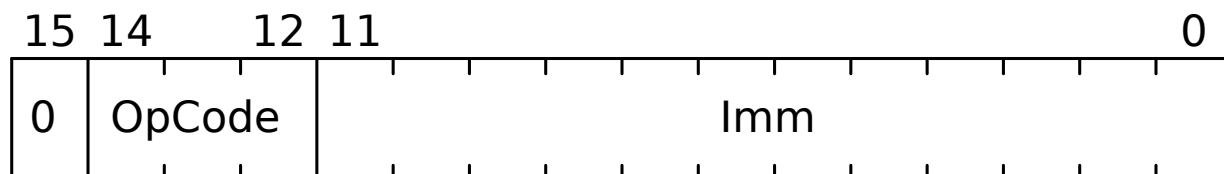
There is a natural appeal to 64 registers on a 64-bit machine. This means 6 bits are needed in the instruction format to address each register. While 64-bit instructions allow this and efficient loading of immediate values, they waste program space more often than not. Variable length instructions are a good compromise to avoid the size penalty when not necessary.

5.1.2 Instruction Format

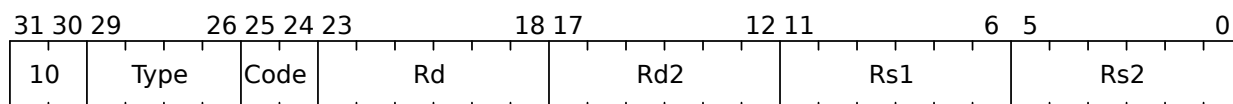
There are 6 instruction formats in Raisin64



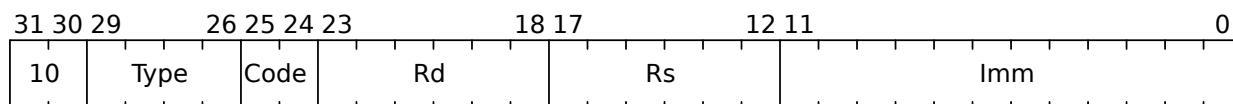
Size



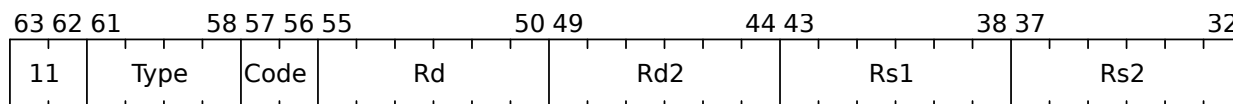
Size



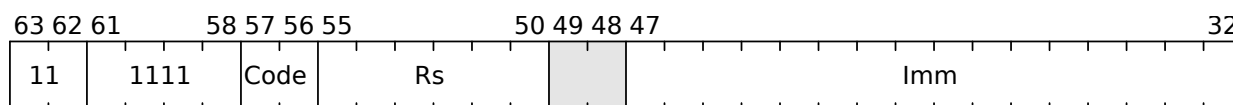
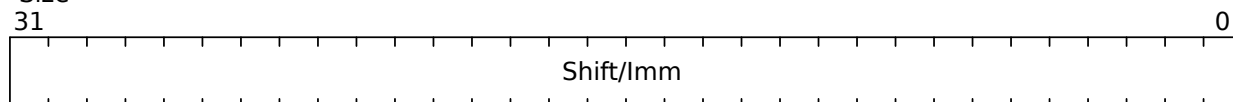
Size



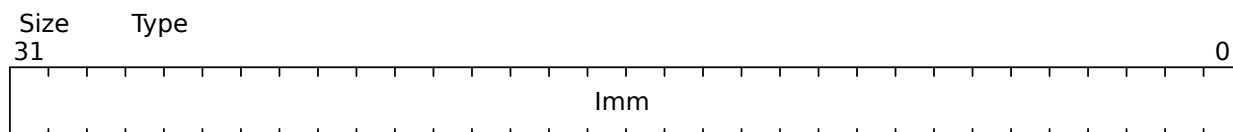
Size



Size



Size



5.1.3 Instructions

ADD - Integer Add

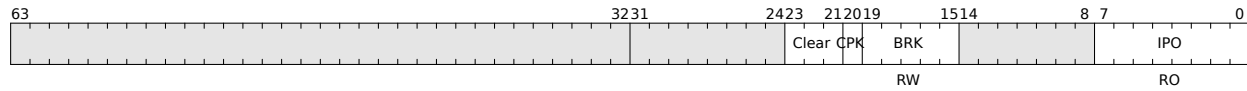


Fig. 1: ADD

ADDI - Integer Add Immediate**AND - Bitwise AND****ANDI - Bitwise AND Immediate****BE - Branch if Even****BO - Branch if Odd****DIV - Integer Divide****DIVI - Integer Divide Immediate****J - Jump****JAL - Jump and Link****JALI - Jump and Link Immediate****JI - Jump Immediate****L16 - Load 16-bit****L16S - Load 16-bit Sign-Extend****L32 - Load 32-bit****L32S - Load 32-bit Sign-Extend****L8 - Load 8-bit****L8S - Load 8-bit Sign-Extend****LW - Load 64-bit Word****MUL - Integer Multiply****MUL - Integer Multiply Immediate****NOR - Bitwise NOR****NORI - Bitwise NOR Immediate****OR - Bitwise OR**

ORI - Bitwise OR Immediate

S16 - Store 16-bit

S32 - Store 32-bit

S8 - Store 8-bit

SEQ - Set 1 if Equal

SEQI - Set 1 if Equal Immediate

SLE - Set 1 if Less than or Equal

SLEI - Set 1 if Less than or Equal Immediate

SLL - Shift Left Logical

SLLI - Shift Left Logical Immediate

SLT - Set 1 if Less Than

SLTI - Set 1 if Less Than Immediate

SNE - Set 1 if Not Equal

SNEI - Set 1 if Not Equal Immediate

SRA - Shift Right Arithmetic

SRAI - Shift Right Arithmetic Immediate

SRL - Shift Right Logical

SRLI - Shift Right Logical Immediate

SUB - Integer Subtract

SUBI - Integer Subtract Immediate

SW - Store 64-bit Word

SYSCALL - System Call

XOR - Bitwise XOR

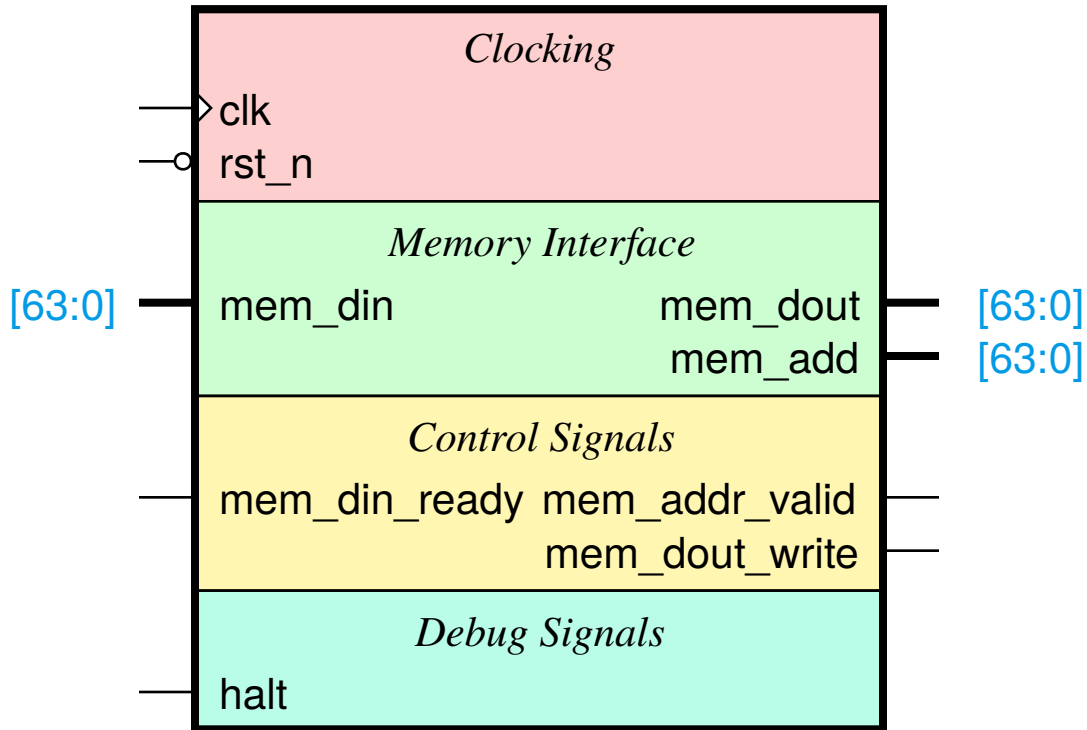


Fig. 2: Raisin64.v

XORI - Bitwise XOR Immediate

5.2 Verilog Module Index

5.2.1 Raisin64.v

```

1  /*
2   * Raisin64 CPU
3   */
4
5  module raisin64 (
6      //# {{clocks|Clocking}}
7      input clk,
8      input rst_n,
9
10     //# {{data|Memory Interface}}
11     input [63:0] mem_din,
12     output [63:0] mem_dout,
13     output [63:0] mem_add,
14
15     //# {{control|Control Signals}}
16     output mem_addr_valid,
17     output mem_dout_write,
18     input mem_din_ready,
19
20     //# {{debug|Debug Signals}}

```

(continues on next page)

(continued from previous page)

```
21     input halt);  
22  
23 endmodule
```