**Proyecto 10 - Nucleo Risc-V, primera versión**

Autor: Cruz Martin Matías

**Resumen**

El informe siguiente desarrolla los temas referentes a Nucleo Risc-V, entre ellos la Arquitectura y micro arquitectura de Computadoras, el conjunto de registros de la arquitectura RISC-V y el conjunto de instrucciones RV32I. Más tarde habla de cómo se armó un datapath multiciclo en el cual se incluyó los bloques desarrollados en proyectos anteriores y se le incluyo  operaciones necesarias para la implementación en el datapath diseñado de las operaciones RV32I lw, sw y las de codigos de operación 19, 51, 99, 103 y 111

**Introducción**

**Arquitectura de computadora**

La arquitectura de computadora es la forma lógica de la computadora desde el punto de vista del programador, que puede hacer el procesador de la computadora, que instrucciones acepta de manera cruda, que operaciones soporta , como realiza esas operaciones de forma lógica, como esta organizada la memoria de la computadora,

**Microarquitectura**

La **microarquitectura** (a veces abreviada como **µarch** o **uarch**), también llamada como **organización de la computadora**, es la manera en que una arquitectura del conjunto de instrucciones (ISA, *Instruction Set Architecture*) se implementa en un procesador. La [arquitectura de computadora](https://es.wikipedia.org/wiki/Arquitectura_de_computadora) es la combinación del conjunto de instrucciones y el diseño concreto de la microarquitectura.

<https://es.wikipedia.org/wiki/Microarquitectura>

Durante este proyecto luego de realizar la investigación correspondiente se trabajó siguiendo el diseño del procesador nos pusimos a diseñar en vhd sus componentes, algunos fueron sacados de proyectos anteriores y adaptados, con el propósito de plasmar los resultados de la simulación en el informe

**Desarrollo**

**Arquitectura RISC-V**

**RISC-V** es una arquitectura de computadoras moderna en particular, que usa la filosofía de simplificar todo, de analizar como se están usando actualmente las computadoras, ver las operaciones mas comunes para el publico a quien voy a dirigir mi diseño, hacer el diseño basado en hacer solo lo necesario, soportar las operaciones mas comunes de forma conveniente y de modo que sea mas fácil hacer el hardware de la computadora.

**Conjunto de instrucciones RV32I**

El conjunto de instrucciones **RV32I** es la base de RISC-V para procesadores de 32 bits. Este conjunto es fundamentalmente minimalista, diseñado para aplicaciones generales, y sirve como base para extensiones opcionales que agregan capacidades adicionales, como operaciones de multiplicación, manejo de enteros o compatibilidad con punto flotante.

Five Embeddev

[**https://five-embeddev.com/**](https://five-embeddev.com/)

**Conjunto de registros de la arquitectura RISC-V**

Cuando vemos las instrucciones de la arquitectura Risc-V veremos que operan sobre algo que denominamos registro, los registros son posiciones de memoria que están asociados directamente al procesador o están incluidas las memorias esas dentro del procesador, los registros son unas memorias rápidas que están integradas al procesador.

Los registros son los mismos para cualquier versión, lo que varia es que la de 32 bits , cada registro es de 32 bits, el numero del registro en el conjunto de instrucciones se refiere a que tiene un lugar para poner un registro, RD para el registro del destino, RS1 RS2 para el registro de operador de la ALU, lo que hay que hacer ahí es sustituirlo por alguno de los registros, todos los registros los podemos referenciar con X seguido del numero, en la tabla aparecen los nombres oficiales.

* **Estudiar, del conjunto de instrucciones RV32I**
  + **Describir la estructura de las instrucciones tipo R, I, S, B, U y J.**

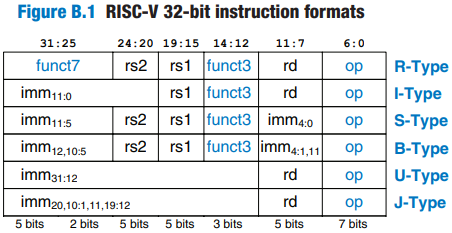
RISC-V se compromete a definir cuatro formatos de instrucción principales: tipo R, tipo I, tipo S/B y tipo U/J. Este pequeño número de formatos permite cierta regularidad entre las instrucciones y, por lo tanto, un hardware de decodificador más simple, al tiempo que se adapta a diferentes necesidades de instrucción.

Las instrucciones de tipo R (registro), como agregar s0, s1, s2, operan en tres registros.

Las instrucciones de tipo I (inmediatas), como addi s3,s4,42,

Las instrucciones de tipo S/B (almacén/rama), como sw a0,4(sp) o beq a0,a1,L1, operan en dos registros y un inmediato firmado de 12 o 13 bits.

Las instrucciones de tipo U/J (inmediato superior/salto), como jal ra, factorial, operan en un registro y en un registro inmediato de 20 o 21 bits.



(Harris, Harris, 2022, p 332 )

* + **Determinar cómo se calcula el valor inmediato para los tipos I, S, B, U y J.**

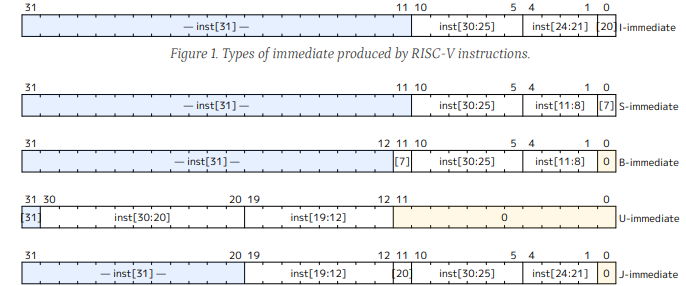
RISC-V utiliza archivos inmediatos formados de 32 bits. Solo de 12 a 21 bits del inmediato están codificados en la instrucción. La figura 6.26 muestra cómo se forman los inmediatos para cada tipo de instrucción.

.Las instrucciones de tipo I y S codifican inmediatos formados de 12 bits.

.Las instrucciones de tipo J y B utilizan inmediatos con signo de 21 y 13 bits, donde el bit menos significativo es siempre 0 .

.Las instrucciones de tipo U codifican los 20 bits superiores de un inmediato de 32 bits.

Los tipos inmediatos muestran los inmediatses producidos por cada uno de los formatos de instrucción base y están etiquetados para mostrar qué bit de instrucción (inst[y]) produce cada bit del valor inmediato.



(Harris, Harris, 2022, p 340 )

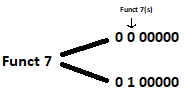
The RISC-V Instruction Set Manual Volume I, 2010-2017, p 25.

* + **Analizar las instrucciones de código de operación decimal 19 y 51, encontrar la manera en que se codifica la operación matemática de la ALU**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrucciones con OP:19** | | | | |
| **Funct 3** | **Imm10** | **Instrucción (nemonico)** | **Descripción** | **OPE ALU** |
| **000** | **-** | **addi** | **Suma valor inmediato** | **Suma** |
| **001** | **0** | **slli** | **Desplaza a izquierda por valor inmediato** | **Desplaza a izquierda** |
| **010** | **-** | **slti** | **Menor que valor inmediato** | **Menor (con signo)** |
| **011** | **-** | **sltiu** | **Menor que valor inmediato (sin signo)** | **Menor (sin signo)** |
| **100** | **-** | **xori** | **‘o’ exclusiva con valor inmediato** | **Xor** |
| **101** | **0** | **srli** | **Desplaza a derecha por valor inmediato (sin signo)** | **Desplaza a derecha (sin signo)** |
| **101** | **1** | **srai** | **Desplaza a derecha por valor inmediato (con signo)** | **Desplaza a derecha (con signo)** |
| **110** | **-** | **ori** | **‘o’ con valor inmediato** | **or** |
| **111** | **-** | **andi** | **‘y’ con valor inmediato** | **and** |

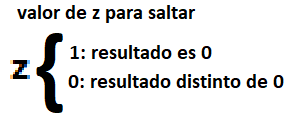
Debo poner la operación de la ALU necesaria en cada caso para llevar a cabo esas instrucciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrucciones con OP:51** | | | | |
| **Funct 3** | **Funct 7(s)** | **Instrucción (mnemonico)** | **Descripción** | **OPE ALU** |
| **000** | **0** | **add** | **Suma a rs1 + rs2** | **Suma** |
| **000** | **1** | **sub** | **Resta de rs1 — rs2** | **resta** |
| **001** | **0** | **sll** | **desplazar rs1 a la izq por rs2** | **Desplaza izquierda** |
| **010** | **0** | **slt** | **rs1 menor que rs2 (con signo)** | **Menor (con signo)** |
| **011** | **0** | **sltu** | **rs1 menor que rs2 (sin signo)** | **Menor (sin signo)** |
| **100** | **0** | **xor** | **rs1 xor rs2** | **Xor** |
| **101** | **0** | **srl** | **Desplaza rs1 a la derecha por rs2 (sin signo)** | **Desplaza a derecha (sin signo)** |
| **101** | **1** | **sra** | **Desplaza rs1 a la derecha por rs2 (con signo)** | **Desplaza a derecha (con signo)** |
| **110** | **0** | **or** | **Rs1 or rs2** | **or** |
| **111** | **0** | **and** | **Rs1 and rs2** | **and** |



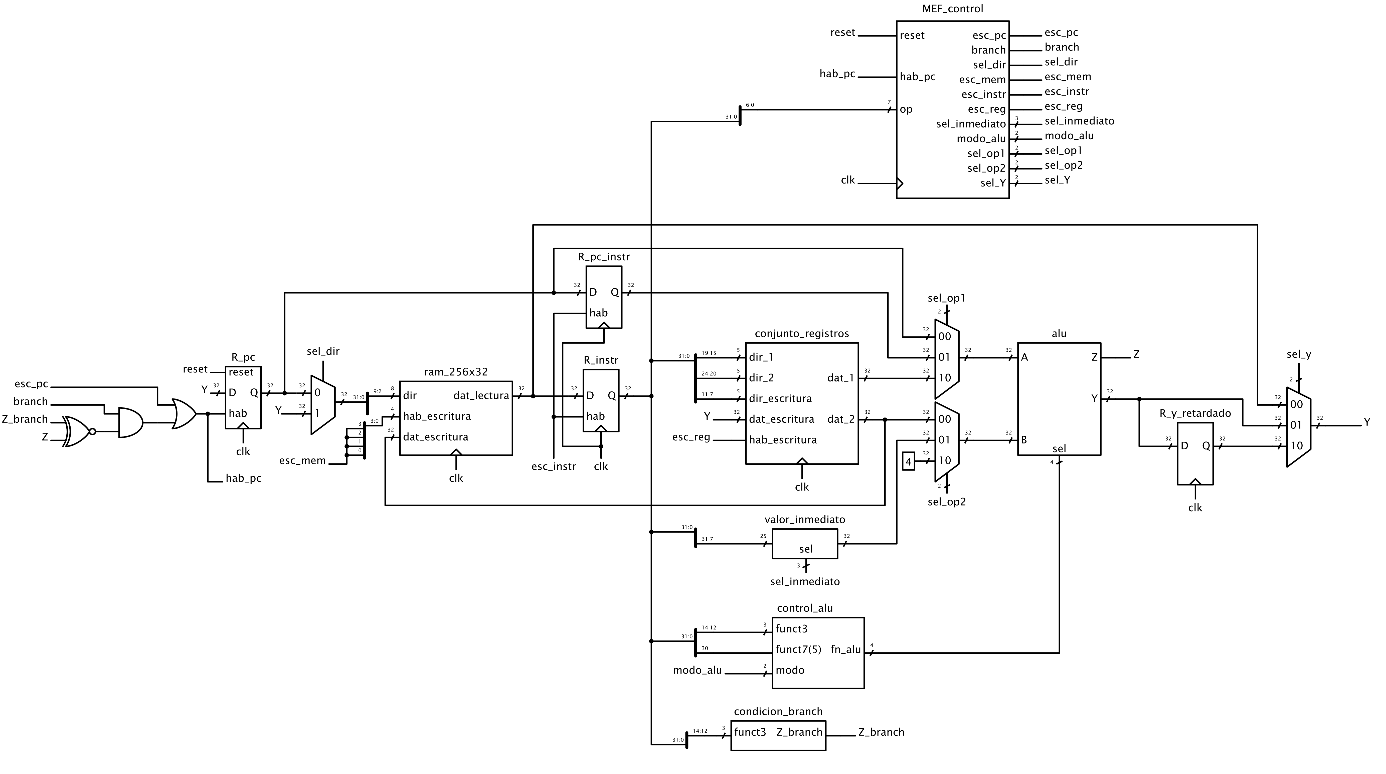
* + **Analizar las instrucciones de código de operación decimal 99 (saltos condicionales), determinar la forma en que se codifica la operación de la ALU y la condición de cero necesaria para tomar el salto.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instrucciones con OP:99** | | | | |
| **Funct 3** | **Instrucción** | **Descripción** | **OPE ALU** | **Valor de z para saltar** |
| **000** | **beq** | **Salta si : rs1 = rs2** | **Resta** | **1** |
| **001** | **bne** | **Salta si : rs1 ≠ rs2** | **Resta** | **0** |
| **100** | **blt** | **Salta si : rs1 < rs2 (con signo)** | **Menor (con signo)** | **0** |
| **101** | **bge** | **Salta si : rs1 ≥ rs2 (con signo)** | **Menor (con signo)** | **1** |
| **110** | **bltu** | **Salta si : rs1 < rs2 (sin signo)** | **Menor (sin signo)** | **0** |
| **111** | **bgeu** | **Salta si : rs1 ≥ rs2 (sin signo)** | **Menor (sin signo)** | **1** |



**Resultados**

Diseño del procesador: diagrama esquemático del diseño final esperadojqe



**Listado de bloques (partes que lo componen y banco de prueba)**

**Componentes de proyectos anteriores (Modificados para cumplan su función en el procesador)**

.Conjunto de registros: aquí se modificó al registro 0 , lo convertimos a solo lectura

.RAM 256x32

.Registró 32 bits: aquí le dimos la opción de tener reset y entrada de habilitación sincronico

.Unidad aritmética lógica (ALU)

***Bloques que van dentro de la alu:***

.función cero

.función AND

.función OR

.función XOR

.función SUMA/RESTA

.función MENOR QUE

.función desplazamiento a la izquierda

.función desplazamiento a la derecha

**Componentes hechos ahora**

.Valor inmediato

.Control ALU:

A partir de los parámetros de la instrucción y de un modo, determinar la configuración de la ALU, como el modo de solo suma, operación con valor inmediato, operación entre registros y de cálculo de operacion

.Condición Branch:

De la instrucción extrae el valor que tiene que tener la señal Z para que tome el salto, tengo distintas instrucciones branch con distintas condiciones y una parte de las condiciones es que si z tiene que ser 0 o si tiene que ser 1 eso es lo que determina, si me pongo a ver el bloque tiene una sola salida que se llama Z\_branch.

.Máquina de estado finito de control (MEF control)

Los archivos donde están definidos los componentes tienen el nombre del componente con la extensión vhd y los archivos donde están definidas las simulaciones tienen sim\_nombre\_del\_componente.vhd

|  |  |
| --- | --- |
| ***Componentes*** | ***Descripción*** |
| ConjuntodeRegistros\_32x32\_b | Aquí se modificó al registro 0 , lo convertimos a solo lectura |
| Memoria\_RAM\_dp\_256x32\_B | Almacenar instrucciones y dato |
| Registro | Aquí le dimos la opción de tener reset y entrada de habilitación sincrónico |
| ALU | Bloque funcional dentro del procesador encargado de realizar las operaciones aritméticas (como suma, resta, etc.) y lógicas (como AND, OR, XOR) |
| Valor\_inmediato | Genera valores constantes o desplazamientos (offsets) que se requieren en ciertas instrucciones |
| Control\_alu | A partir de los parámetros de la instrucción y de un modo, determinar la configuración de la ALU , como el modo de solo suma, operación con valor inmediato, operación entre registros y de cálculo de operación |
| Condicion\_branch | De la instrucción extrae el valor que tiene que tener la señal Z para que tome el salto, tengo distintas instrucciones branch con distintas condiciones y una parte de las condiciones es que si z tiene que ser 0 o si tiene que ser 1 eso es lo que determina, si me pongo a ver el bloque tiene una sola salida que se llama Z\_branch |
| MEF\_control | Es un modelo matemático usado para representar y controlar el comportamiento secuencial de un sistema |

Bloques que conforman la Unidad aritmética lógica (ALU)

|  |  |
| --- | --- |
| ***Componente*** | ***Descripción*** |
| FuncionCero | Produce una salida de 0 independientemente de las entradas |
| FuncionAND | Realiza una **operación lógica AND** bit a bit entre dos entradas |
| FuncionOR | Realiza una **operación lógica OR** bit a bit entre dos entradas. |
| FuncionXOR | Realiza una **operación lógica XOR** (o exclusiva) bit a bit entre dos entradas |
| Funcion\_SUMA\_RESTA | Realiza una suma/resta binaria de las dos entradas, considerando el acarreo |
| Funcion\_Menor | Compara las entradas y genera un bit de salida que indica si una entrada es menor que la otra. |
| Funcion\_desplaza\_izq | Desplaza todos los bits de una entrada hacia la izquierda, introduciendo ceros por el lado derecho |
| Funcion\_desplaza\_der | Desplaza todos los bits de una entrada hacia la derecha, introduciendo ceros por el lado izq |

**Conclusiones**

**Referencias**

Sarah L. Harris y David Money Harris (2022). Digital Design and Computer Architecture. RISC-V Edition

John F. Wakerly (2006). Digital Design [ Principles and Practices Fourth Edition