Diseño e Implementación de un Procesador RISC-V Uniciclo

***Abstract*—This document details the design and implementation of a unicycle RISC-V processor. It covers the architectural overview, the functionality of individual modules such as the Program Counter, Instruction Memory, Register File, ALU, Data Memory, and Sign Extension unit. The control logic for decoding and executing various RISC-V instruction types within the main unicycle module is also described.**

***Index Terms*—RISC-V, Unicycle Processor, Computer Architecture, SystemVerilog, HDL.**

I. Introducción

Este informe presenta el diseño e implementación de un procesador RISC-V de 32 bits con arquitectura uniciclo. El procesador está descrito en SystemVerilog e incluye los módulos esenciales necesarios para la obtención, decodificación y ejecución de un subconjunto de la arquitectura de conjunto de instrucciones (ISA) RISC-V. El objetivo principal de este proyecto es demostrar el funcionamiento de un camino de datos y unidad de control uniciclo. Este documento detalla el marco teórico, la arquitectura general del sistema, el diseño detallado de cada módulo hardware y el conjunto de instrucciones soportado.

II. Marco Teórico

A. Arquitectura RISC-V

La ISA RISC-V es una arquitectura de conjunto de instrucciones de código abierto basada en los principios establecidos de computadoras con conjunto reducido de instrucciones (RISC). Su diseño modular permite una ISA base entera con extensiones opcionales para diversas necesidades computacionales. Este proyecto implementa un subconjunto del conjunto base entero RV32I.

B. Procesador Uniciclo

Un procesador uniciclo ejecuta cada instrucción en un solo ciclo de reloj. Esta filosofía de diseño simplifica la unidad de control, pero puede resultar en un ciclo de reloj más largo, ya que debe acomodar el camino crítico de la instrucción más compleja. Los componentes del camino de datos operan en paralelo dentro de ese único ciclo.

C. Referencias clave

Los principios de diseño y conceptos específicos de RISC-V discutidos en este documento están fuertemente influenciados por la literatura clásica de arquitectura de computadoras, en particular la obra de Patterson y Hennessy [**?**].

III. Arquitectura del Sistema

A. Camino de Datos General

El camino de datos del procesador consiste en varios módulos interconectados: Program Counter (PC), Memoria de Instrucciones, Banco de Registros, Unidad de Extensión de Signo, ALU y Memoria de Datos. Estos módulos trabajan en conjunto para obtener instrucciones, leer operandos, realizar cálculos y acceder a memoria de datos. El módulo principal 'RISCVunicycle' instancia y conecta estos componentes.

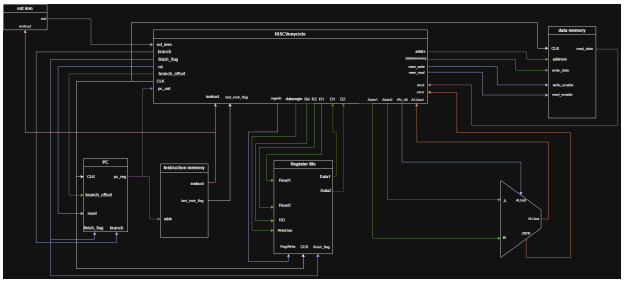

Fig. 1. Diagrama de bloques de alto nivel del procesador RISC-V uniciclo. Se observa la interconexión entre los módulos principales: PC, memoria de instrucciones, extensión de signo, banco de registros, ALU, memoria de datos y el módulo de control principal ('RISCVunicycle'). Cada flecha representa el flujo de datos y señales de control entre los diferentes componentes para la correcta ejecución de las instrucciones.

B. Unidad de Control

La unidad de control está embebida dentro del módulo 'RISCVunicycle'. Decodifica los campos opcode, funct3 y funct7 de la instrucción obtenida para generar las señales de control necesarias para los componentes del camino de datos. Estas señales determinan la operación de la ALU, el acceso a memoria, la habilitación de escritura en registros y la actualización del PC (incluyendo saltos condicionales).

IV. Descripción de Módulos

A. Contador de Programa (PC.sv)

El módulo Program Counter (PC) es responsable de mantener la dirección de la instrucción que está siendo obtenida.

1) Entradas:

- `clk`: Señal de reloj.
- `reset`: Señal de reset asíncrono.
- `branch`: Señal de control para instrucciones de salto.
- `branch_offset`: Desplazamiento con signo de 32 bits para instrucciones de salto.
- `finish_flag`: Bandera que indica finalización del programa.

2) Salidas:

- `pc_reg`: Valor actual del contador de programa (32 bits).

3) Funcionamiento: Al hacer reset, el PC se inicializa a `32'hFFFFFFFC`. En cada ciclo de reloj, si no hay salto y el programa no ha terminado, el PC se incrementa en 4. Si se toma un salto (`branch` en alto), `pc_reg` se actualiza a la suma del PC actual y `branch_offset`.

B. Memoria de Instrucciones (Instmemory.sv)

Este módulo almacena las instrucciones del programa.

1) Entradas:

- `addr`: Dirección de 32 bits proveniente del PC.

2) Salidas:

- `instruct`: Instrucción de 32 bits obtenida de la dirección indicada.
- `last_instr_flag`: Bandera que se activa si la instrucción obtenida es `32'hFFFFFFFF`.

3) Funcionamiento: Inicializa su contenido desde el archivo `FinalT.hex`. Proporciona la instrucción ubicada en la dirección de byte especificada por `addr`. La memoria se implementa como un arreglo de bytes y cuatro bytes se concatenan para formar la instrucción de 32 bits. La bandera `last_instr_flag` es utilizada por el módulo de control principal para detectar el fin del programa.

C. Banco de Registros (registerfile.sv)

El banco de registros contiene los 32 registros de propósito general del procesador.

1) Entradas:

- `Read1`, `Read2`: Direcciones de 5 bits para los registros a leer.
- `RD`: Dirección de 5 bits para el registro a escribir.
- `WriteData`: Dato de 32 bits a escribir en el banco de registros.
- `RegWrite`: Señal de control que habilita la escritura.
- `clock`: Señal de reloj (aunque la escritura parece ser combinacional con `WriteData` en el código, típicamente es sincronizada).
- `finish_flag`: Bandera de finalización del programa.

2) Salidas:

- `Data1`, `Data2`: Datos de 32 bits leídos de los registros especificados por `Read1` y `Read2`.

3) Funcionamiento: Permite dos lecturas simultáneas y una escritura. El registro `x0` está cableado a cero (inicializado en 0 y típicamente no se escribe por convención RISC-V). La escritura ocurre si `RegWrite` está activa y el programa no ha terminado (verificado con `finish_flag`). Algunos registros se inicializan con valores específicos para pruebas.

D. Unidad de Extensión de Signo (signext.sv)

Este módulo extiende los valores inmediatos de las instrucciones a 32 bits.

1) Entradas:

- `instruct`: Instrucción de 32 bits que contiene el campo inmediato.

2) Salidas:

- `out`: Valor inmediato extendido a 32 bits con signo.

3) Funcionamiento: Determina el tipo de inmediato según el opcode (bits 6:0) de la instrucción y realiza la extensión de signo correspondiente para inmediatos tipo I, S y B. Para instrucciones de carga (opcode `7'b0000011`), también realiza extensión tipo I.

E. Unidad Aritmético-Lógica (RISCVALU.sv)

La ALU realiza operaciones aritméticas y lógicas.

1) Entradas:

- `ALUctl`: Señal de control de 4 bits que especifica la operación.
- `A`, `B`: Operandos de 32 bits.
- `reset_zero_flag`: Señal para reiniciar la bandera interna zero.

2) Salidas:

- `ALUout`: Resultado de la operación (32 bits).
- `zero`: Bandera que se activa si el resultado de una resta (A-B) es cero.

3) Funcionamiento: Las operaciones soportadas incluyen AND, OR, ADD, SUB y SLT (Set Less Than). La bandera `zero` se actualiza específicamente tras una resta si el resultado es cero y puede ser reiniciada mediante `reset_zero_flag`.

F. Memoria de Datos (datamem.sv)

La memoria de datos se utiliza para operaciones de carga y almacenamiento.

1) Entradas:

- `clk`: Señal de reloj.
- `address`: Dirección de 32 bits para acceso a memoria.
- `write_data`: Dato de 32 bits para operaciones de almacenamiento.
- `write_enable`: Señal de control para habilitar escritura.
- `read_enable`: Señal de control para habilitar lectura.

2) Salidas:

- `read_data`: Dato de 32 bits leído de memoria para operaciones de carga.

3) Funcionamiento: Modela una memoria de 256 palabras de 32 bits. Inicializa su contenido desde `ProyectoCorto_data.hex`. Las operaciones de escritura se realizan en el flanco positivo del reloj si `write_enable` está activa. Las operaciones de lectura también son síncronas, proporcionando `read_data` según la dirección si `read_enable` está activa.

G. Módulo Principal y Control (RISCVunicycle.sv)

Este módulo de nivel superior integra todos los componentes y contiene la lógica de control del procesador uniciclo.

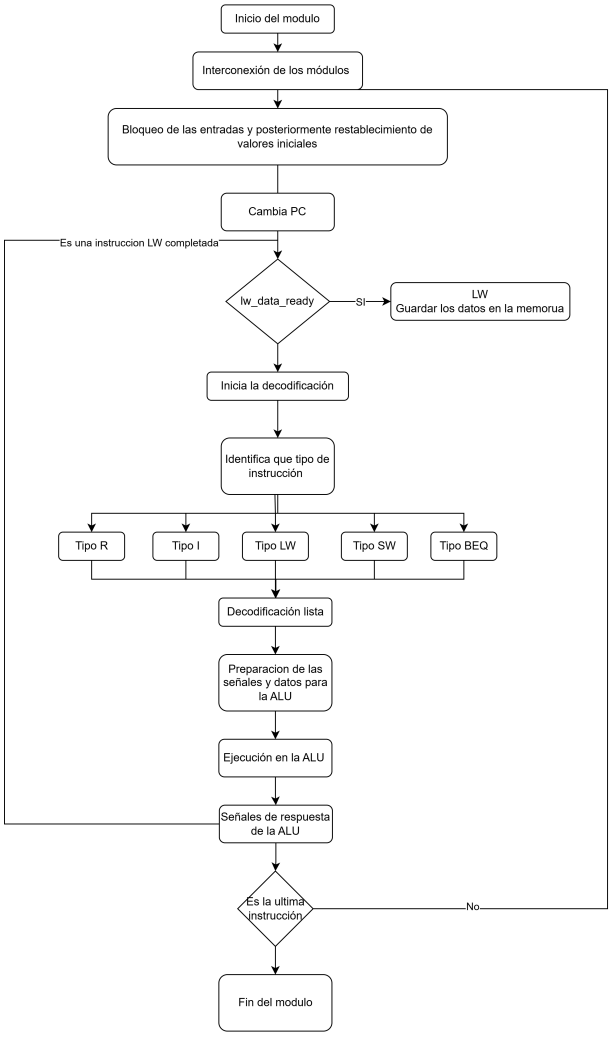

Fig. 2. Diagrama de flujo del módulo principal RISCVunicycle

1) Funcionamiento: Obtiene instrucciones usando el PC y la Memoria de Instrucciones. Decodifica las instrucciones según sus campos opcode, funct3 y funct7. Según la instrucción decodificada, genera señales de control para el Banco de Registros (lectura/escritura, habilitación), la Unidad de Extensión de Signo, la ALU (selección de operandos y operación) y la Memoria de Datos (lectura/escritura, dirección). Soporta instrucciones tipo R, I (aritméticas y de carga), S (almacenamiento) y B (salto condicional). También soporta JAL y JALR. Una instrucción especial `32'hFFFFFFFF` actúa como señal de parada, activando `finish_flag`.

V. Conjunto de Instrucciones Implementado

Según la lógica de decodificación en `RISCVunicycle.sv`, el procesador implementa los siguientes tipos de instrucciones:

- **Tipo R**
  - ADD
  - SUB
  - AND
  - OR
- **Tipo I**
  - ADDI
  - ANDI
  - ORI
- **Tipo L**
  - LW
  - LUI
- **Tipo S**
  - SW
- **Tipo B**
  - BEQ

VI. Resultados

Para la simulación del procesador, se utilizó el archivo `FinalT.hex` como memoria de instrucciones y el archivo `ProyectoCorto_data.hex` como memoria de datos.

Contenido de FinalT.hex (memoria de instrucciones):

```
addi x1, x0, 7      # Inicializa x1 con 7
addi x2, x0, -10    # Inicializa x2 con -10
lw x4, 1(x0)        # Carga el primer dato
    (0x00000005) desde memoria a x4
lw x5, 3(x0)        # Carga el segundo dato
    (0x00000006) desde memoria a x5
addi x0, x0, 0      # NOP
add x6, x4, x5      # Suma x4 y x5, guarda el
    resultado en x6
sw x6, 1(x0)        # Almacena el valor de x6 en la
    direccion 4 de memoria
beq x6, x1, begin   # Si x6 == x1, salta al inicio
addi x8, x0, 10     # Si no salta, inicializa x8
    con 10 (fin del programa)
```

Contenido de ProyectoCorto_data.hex (memoria de datos):

```
00000005
00000006
00000000
00000001
```

Estos archivos contienen, respectivamente, el conjunto de instrucciones a ejecutar y los datos necesarios para las operaciones de carga y almacenamiento durante la simulación.

Los resultados completos de la salida obtenida en la simulación del procesador pueden consultarse en el apéndice.

VII. Conclusión

Este documento ha descrito el diseño e implementación de un procesador RISC-V uniciclo, explicando la funcionalidad de cada módulo y su rol dentro de la arquitectura general. Tras la simulación, los resultados obtenidos muestran el correcto funcionamiento del procesador, evidenciando la ejecución secuencial de instrucciones tipo I, R, S, B y LW, así como la interacción entre los módulos principales. El procesador soporta un subconjunto fundamental de instrucciones RISC-V, validando los principios básicos de un diseño uniciclo y permitiendo observar el flujo de datos y control en cada ciclo de instrucción.

APÉNDICE

RESULTADOS COMPLETOS DE LA SIMULACIÓN

A continuación se muestra el listado completo de la salida obtenida en la simulación del procesador:

```
Memory initialized:
VCD info: dumpfile RISCVunicycle_tb.vcd opened for output.
reset done
PC reset: fffffffc
Data1:          0
Data2:          0
A:          0
B:          0
ALUctl: 0000
addr:          0, Instruccion: 00700093
Inm ext mod side: 00000007

PC:           0
Instrucion: 00700093
opcode: 0010011
tipo I
Registro destino:  1
funct3: 0000
ADDI
R1:  0
D1:          0
alu_op: 0010
imm:     7
ext_imm:           7
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2:          7
A:          0
B:          7
ALUctl: 0010
Resultado de la ALU:           7
Resultado listo para escritura en registro:           7
continuing to next instruction.
WriteData:           7
Inm ext mod side: fffffff6
addr:          4, Instruccion: ff600113

PC:           4
Instrucion: ff600113
opcode: 0010011
tipo I
Registro destino:  2
funct3: 0000
ADDI
R1:  0
D1:          0
alu_op: 0010
imm:   -10
ext_imm:         -10
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2: 4294967286
A:          0
B: 4294967286
ALUctl: 0010
Resultado de la ALU:         -10
Resultado listo para escritura en registro:         -10
continuing to next instruction.
WriteData:         -10
addr:          8, Instruccion: 00102203

PC:           8
Instrucion: 00102203
opcode: 0000011
Load word
Registro destino:  4
alu_op: 0010
imm:     1
ext_imm:           1
```

```
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2:          1
A:          0
B:          1
ALUctl: 0010
Resultado de la ALU:           1
Leyendo de memoria en direccion:          1
continuing to next instruction.
Cargando datos desde memoria:           6
addr:         12, Instruccion: 00302283
WriteData:           6

PC:          12
Instrucion: 00302283
opcode: 0000011
Load word
Registro destino:  5
alu_op: 0010
imm:     3
ext_imm:           3
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2:          3
A:          0
B:          3
ALUctl: 0010
Resultado de la ALU:           3
Leyendo de memoria en direccion:          3
continuing to next instruction.
Cargando datos desde memoria:           1
addr:         16, Instruccion: 00000013
Inm ext mod side: 00000000

PC:          16
Instrucion: 00000013
opcode: 0010011
tipo I
Registro destino:  0
funct3: 0000
ADDI
R1:  0
D1:          0
alu_op: 0010
imm:     0
ext_imm:           0
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2:          0
A:          0
B:          0
ALUctl: 0010
Resultado de la ALU:           0
Resultado listo para escritura en registro:           0
continuing to next instruction.
WriteData:           0
addr:         20, Instruccion: 00520333

PC:          20
Instrucion: 00520333
opcode: 0110011
tipo R
funct3: 0000
Registro destino:  6
ADD
R1:  4
R2:  5
D1:          6
D2:          1
alu_op: 0010
Data1:          6
Data2:          1
Ejecutando operacion en la ALU
Aluin1:          6
```

```
Aluin2:          1
A:          6
B:          1
ALUctl: 0010
Resultado de la ALU:           7
Resultado listo para escritura en registro:           7
continuing to next instruction.
WriteData:           7
addr:         24, Instruccion: 006020a3

PC:          24
Instrucion: 006020a3
opcode: 0100011
store word
alu_op: 0010
Data1:          0
Data2:          1
imm:     1
ext_imm:           1
Ejecutando operacion en la ALU
Aluin1:          0
Aluin2:          1
A:          0
B:          1
ALUctl: 0010
Resultado de la ALU:           1
Escribiendo en memoria en direccion:          1
Datos a escribir en memoria:           1
continuing to next instruction.
Inm ext mod side: fffffffe4
addr:         28, Instruccion: fe1302e3

PC:          28
Instrucion: fe1302e3
opcode: 1100011
branch?
alu_op: 0110
Data1:          7
Data2:          7
imm:   -14
ext_imm:         -28
Ejecutando operacion en la ALU
Aluin1:          7
Aluin2:          7
A:          7
B:          7
ALUctl: 0110
Resultado de la ALU:           0
Resultado listo para escritura en registro:           0
Branch taken, jumping to address:           0
...
```