

DISEÑO DE UN PROTOTIPO DE CONTROLADOR PID EN DRONES
CON PROCESADOR A LA MEDIDA EN ARQUITECTURA RISC-V
PARA IMPLEMENTACIÓN EN FPGA

MANUAL DE PROCESADOR INVENIO RV

Autores:

Iván Ricardo Diaz Gamarra
Omar Steck Espinel Santamaría
Magda Daniela Latorre Ortiz

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
DEPARTAMENTO DE ELECTRÓNICA
BOGOTÁ, D.C.

Contenido

Introducción:.....	4
1. Diagrama de bloques del procesador:.....	5
2. Descripción de bloques y señales del procesador:.....	6
ProcessorRV.....	6
IR:	7
PC:.....	8
Registers:	8
MAR:.....	11
Control:.....	11
ALU:.....	13
Decoder:.....	14
Sumador:.....	14
LogicalShiftRight:.....	15
LeftShift.....	15
ArithmeticShiftRight.....	15
And, Or y Xor.....	15
Multiplier32Bits.....	15
CSR:.....	15
Counter:.....	16
SP:.....	16
3. Modos de direccionamiento:.....	17
4. Instrucciones:.....	18
LUI.....	18
AUIPC.....	18
JAL	19
JALR.....	19
BEQ	19
BNE.....	20
BLT.....	20
BGE.....	20
BLTU.....	21
BGEU.....	21
LB.....	21

LH.....	22
LW.....	22
LBU.....	22
LHU.....	22
ADDI.....	24
SLTI.....	24
SLTIU.....	24
XORI.....	25
ORI.....	25
ANDI.....	25
SLLI.....	25
SRLI.....	26
SRAI.....	26
ADD.....	26
SUB	27
SLL.....	27
SLT.....	27
SLTU.....	27
XOR.....	28
SRL.....	28
SRA	28
OR.....	28
AND.....	29
CSRRW.....	29
CSRRS.....	29
CSRRC.....	29
CSRRWI.....	30
CSRRSI.....	30
CSRRCI.....	30
MUL	31
SWSP (PUSH).....	31
LWSP (POP).....	31
5. Referencias:.....	31

Introducción:

El procesador Invenio RV es un procesador de tipo RISC con una longitud de palabra 32 bits que implementa el ISA RISC-V. En este documento se detallan los bloques que lo componen, las señales que maneja el procesador, las instrucciones que implementa y el modo de direccionamiento.

1. Diagrama de bloques del procesador:

En la se muestra el diagrama en bloques propuesto para el procesador, la notación para leer las señales entre bloques es: BloqueDeOrigen_BloqueDeDestino. Las señales que son de más de un bit, incluyen su tamaño, que se muestra después de una línea diagonal así: /bits de la señal. El diagrama está separado por colores, donde cada bloque tiene su propio color para bloques y señales. El diagrama incluye una línea punteada que muestra los límites del procesador, afuera

de esta línea se encuentran los elementos externos del procesador, como la memoria y señales de interrupciones o error.

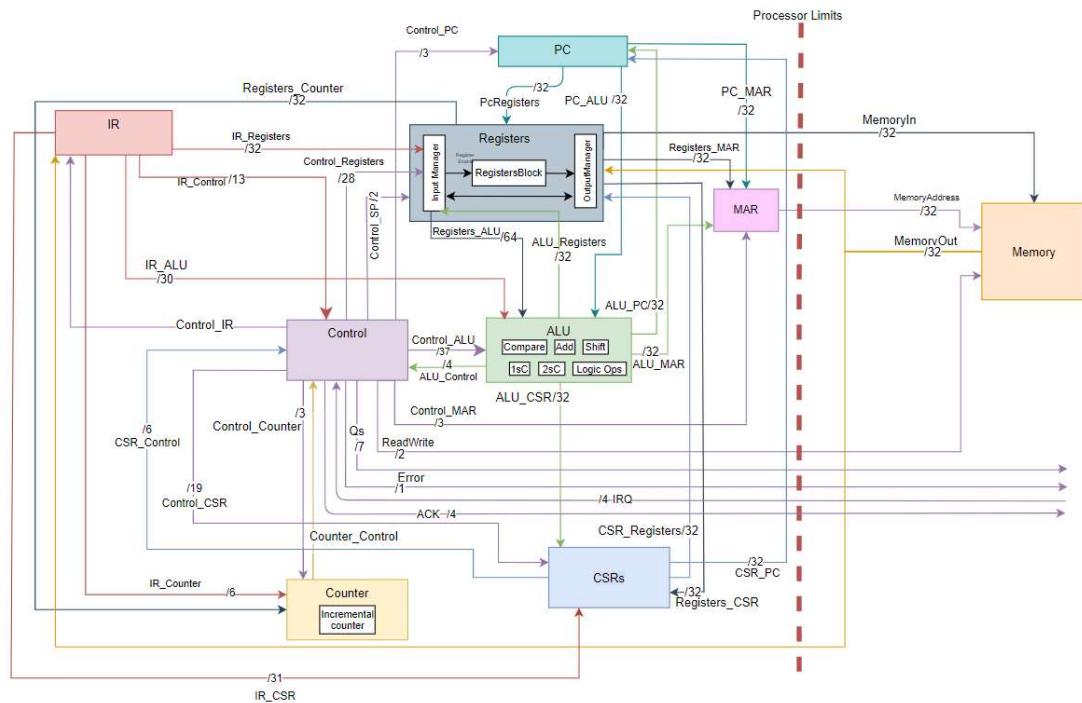


Figura. 1 diagrama de bloques del procesador

2. Descripción de bloques y señales del procesador:

A continuación, se muestra la descripción de los bloques mostrados en la Figura. 1, así como la descripción de las señales de interconexión, en caso de que los bloques contengan bloques internos, se mencionan estos bloques junto a una descripción.

ProcessorRV:

Bloque del procesador, contiene las entradas y salidas del sistema. En el diagrama para facilitar la notación, se muestra como la línea punteada que define los límites del procesador.

- Entradas
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - MemoryOut: 32 bits. Datos provenientes de la memoria
 - IRQ: 4 bits. Bits de interrupciones para periféricos.

- Salidas
 - MemoryAddress: 32 Bits. señal que envía la dirección almacenada en MAR a la memoria.
 - Relojs: 1 Bit. Salida de la señal de reloj para análisis, se añade una letra s de salida al final del nombre de la señal para diferenciarla del reloj de entrada.
 - error: 1 Bit. Señal de error en caso de no estar en ningún estado.
 - Qs: 6 Bits. Q de los estados para comprobación codificado en binario.
 - ACK: 4 bits, bits de respuesta de interrupciones.
 - MemoryIn: 32 bits. Datos provenientes de la memoria

IR:

Instruction Register, registro especial encargado de almacenar las instrucciones provenientes de la memoria y de enviar diferentes partes de la instrucción a los demás bloques para llevar a cabo las acciones necesarias.

- Entradas:
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - MemoryOut: 32 bits. Salida de datos de la memoria, en este caso, contiene la instrucción
 - Control_IR: 1 bit. Señal del control que indica al IR reemplazar su valor con el valor de Memoryout
- Salidas:
 - IR_Registers: 32 bits envían la dirección de los registros rd, rs1 o rs2 de las instrucciones para acceder a los registros necesarios. La cantidad de registros usados en la instrucción depende de cada instrucción, adicionalmente enviar datos inmediatos para almacenar.
 - IR_Control: 13 bits. Envían al control los opcodes de la instrucción para poder determinar los saltos de las instrucciones.
 - IR [25] | IR [30] | IR[15:12] | IR[6:0]
 - IR_ALU: 30 bits. Envía los datos que deben ser operados en la ALU, tales como datos inmediatos en las instrucciones.
 - IR_ALU (24:0): IR (31: 7)
 - IR_ALU (30:25) IR (6:2)
 - IR_CSRs: 32 bits. Envía las direcciones de los CSR donde se realiza la instrucción. O el IR completo para almacenarlo en las excepciones.
 - IR_Counter: 6 bits. Se usa para cargar directamente un valor al Counter

- IR_Counter(5:0): (25:20)

PC:

Program Counter, registro especial encargado de almacenar y aumentar la dirección de la memoria donde se recuperan las instrucciones, permite la carga de datos para hacer saltos, al iniciar empieza en la dirección 0x0000.

En esencia el bloque PC, se comporta como un contador unitario, ya que las instrucciones almacenadas en la memoria están sucesivas una de la otra, este contador solo se incrementa.

- Entradas
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - Control_PC: 3 bits.
 - Control_PC[0]: Señal de control que indica al PC que se auto incremente, estos auto incrementos aumentan 4 al PC actual.
 - Control_PC[1]: Señal de control que indica al PC que reemplace su valor actual, con el valor de ALU_PC.
 - Control_PC[2]: Señal de control que indica al PC que reemplace su valor actual, con el valor de CSR.
 - ALU_PC: 32 bits, contiene el nuevo PC para ser almacenado.
 - CSR_PC: 32 bits, contiene el nuevo valor para PC.
- Salidas
 - PC_MAR: 32 bits. Señal que envía la dirección desde el PC al MAR para obtener la instrucción de memoria.
 - PC_ALU: 32 bits. Señal que envía el contenido del PC a la ALU.

Registers:

Bloque de registros, contiene el selector y 32 registros, como se ve en la Figura. 2

31	x0 / zero	Alambrado a cero
	x1 / ra	Dirección de retorno
	x2 / sp	Stack pointer
	x3 / gp	Global pointer
	x4 / tp	Thread pointer
	x5 / t0	Temporal
	x6 / t1	Temporal
	x7 / t2	Temporal
	x8 / s0 / fp	Saved register, frame pointer
	x9 / s1	Saved register
	x10 / a0	Argumento de función, valor de retorno
	x11 / a1	Argumento de función, valor de retorno
	x12 / a2	Argumento de función
	x13 / a3	Argumento de función
	x14 / a4	Argumento de función
	x15 / a5	Argumento de función
	x16 / a6	Argumento de función
	x17 / a7	Argumento de función
	x18 / s2	Saved register
	x19 / s3	Saved register
	x20 / s4	Saved register
	x21 / s5	Saved register
	x22 / s6	Saved register
	x23 / s7	Saved register
	x24 / s8	Saved register
	x25 / s9	Saved register
	x26 / s10	Saved register
	x27 / s11	Saved register
	x28 / t3	Temporal
	x29 / t4	Temporal
	x30 / t5	Temporal
	x31 / t6	Temporal
32		

Figura. 2 Distribución de registros

El bloque registers posee un selector que realiza el manejo organiza cuales de los registros son requeridos para esta instrucción, tanto para ser leídos o para almacenar datos dentro de ellos.

- Entradas
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - IR_Registers: 26 bits. Dirección de los registros a usar en la instrucción y en algunos casos datos *immediate* a guardar.
 - Bits 5:0 = IR (11:7)
 - Bits 10:6 = IR (19:15)
 - Bits 15:11 = IR (24:20)
 - Bits 20:16 = IR (6 : 2)
 - Control_Registers: 25 Bits señal de control a los registros para indicar que datos se asignan a la salida y que datos se guardan, así como la ubicación de estos, se detalla en los decoders.
 - ALU_Registers: 32 bits. Señal de salida de la ALU, envía a los registros el resultado de una operación.
 - MemoryOut: 32 bits. Señal que lleva los datos de memoria a ser almacenados en los registros

- CSR_Registers: 32 bits Señal de los CSR a los registros que contiene el valor a guardar
- Salidas
 - Registers_ALU: 64 bits. Señal que transmite los datos de los registros a la ALU para ser operados.
 - Registers_ALU(63:32) segundo dato 32 bits
 - Registers_ALU(31:0) primer dato 32 bits
 - Registers_Counter: 32 bits. Señal que contiene la cantidad a contar.
 - Memory_In: 32 bits, Señal que contiene el valor a guardar en memoria.
 - Registers_CSR: 32 bits. Señal que contiene un dato a guardar en los CSR.
 - Registers_MAR: 32 bits, señal del banco de registros al MAR para una nueva dirección.

Bloques internos registers

El bloque de registros, mostrado en la Figura. 3 posee los siguientes bloques internos:

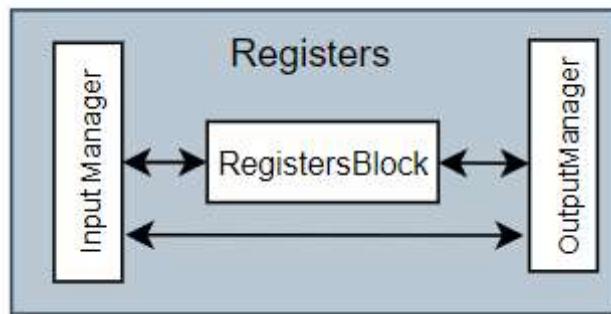


Figura. 3 Bloque de registros

- InputManager, que decodifica la dirección desde el IR para guardar los datos o desde que registros sacar los datos.
- Un banco de registros donde se almacenan los datos, llamado RegisterBlock.
- SP un registro especial con dirección 2, más detalle en bloque SP, incluido dentro del RegistersBlock.
- OutputManager, asigna los datos de salida según los estados del control.

Decoder Registers

El *decoder* de los registros funciona mediante una señal de control que ha sido diseñada para agrupar los estados con las mismas operaciones sobre los registros y datos de entrada y/o salida, al encontrarse en alguno de esos estados agrupados se escribe un 1 en una de las señales correspondientes, esta señal

llega al *decoder* de los registros que realiza la operación con los datos de entrada y salida ya configurados para los estados de esa señal correspondiente.

MAR:

Memory address register. Registro de dirección para almacenar o leer datos de la memoria. Se compone de un registro para una palabra de 32 bits, donde se almacena la dirección de memoria, esta dirección se recibe de ALU, PC o Registers y mediante una señal de control, se selecciona por cuál de las señales de los otros bloques se reemplaza.

- Entradas:

- Reset: 1 Bit. Señal para reiniciar el sistema
- Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
- Control_MAR: 3 bits. Señal que maneja el comportamiento del MAR de acuerdo con los estados:
 - Bit 0: bit para que reciba Pc
 - Bit 1: bit para que reciba ALU
 - Bit 2: bit para que reciba registers
- ALU_MAR: 32 bits. Señal que envía el resultado de una operación de la ALU al MAR
- PC_MAR: 32 bits. Señal que envía la dirección desde el PC al MAR para obtener la instrucción de memoria.
- Registers_MAR: 32 bits, señal del banco de registros al MAR para una nueva dirección.

- Salidas:

- MemoryAddress: 32 bits. Señal que envía la dirección almacenada en MAR a la memoria.

Control:

El bloque control, contiene la máquina de estados del procesador, esta decide el estado actual del procesador basado en las instrucciones, que se transmiten mediante la señal del IR y las entradas de control, y envía señales a los demás bloques ejecutar las diferentes operaciones que componen cada instrucción.

- Entradas

- Reset 1 Bit. Señal para reiniciar el sistema
- Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques

- IR_Control: 13 bits. Envía al control los opcodes de la instrucción para poder determinar los saltos de las instrucciones.
 - IR[25]|IR[30]|IR[15:12]|IR[6:0]
- ALU_Control: 4 bits
 - ALU_Control[0] Bandera de fin de operación de la ALU
 - ALU_Control[1] bandera de cero, se pone en uno si el resultado de una operación de ALU es cero
 - ALU_Control[2] bit de signo resultado de ALU
 - ALU_Control[3] bit de carry resultado de ALU
- Counter_Control: 1 bit Señal que notifica el fin del conteo.
- IRQ: 4 bits. Bits de interrupciones para periféricos.
- CSR_Control: 6 bits Señal que envía los datos de banderas importantes para cambios de estados y manejo de interrupciones al control.
- Salidas
 - Control_MAR: 3 bits. Señal que maneja el comportamiento del MAR de acuerdo con los estados:
 - Bit 0: bit para que reciba Pc
 - Bit 1: bit para que reciba ALU
 - Bit 2: bit para que reciba registers
 - Control_PC: 3 bits.
 - Control_PC[0]: Señal de control que indica al PC que se auto incremente, estos auto incrementos aumentan 4 al PC actual.
 - Control_PC[1]: Señal de control que indica al PC que reemplace su valor actual, con el valor de ALU_PC.
 - Control_PC[2]: Señal de control que indica al PC que reemplace su valor actual, con el valor CSR_PC.
 - Control_Registers: 28 Bits señal de control a los registros para indicar que datos se asignan a la salida y que datos se guardan, así como la ubicación de estos, se detalla en los decoders.
 - Control_IR: una señal de control para que reemplace su valor por el valor del cable Memoryout
 - Control_ALU: 37 bits. Señal del control a la ALU, que dependiendo del estado indica cual operación, con que datos se lleva acabo y en donde se almacena. Para más detalle se ve en el decoder.
 - Control_Counter: 3 bits. Señal de control para el contador.

- Control_Counter[0]: Señal de control hacia el contador para iniciar la cuenta.
 - Control_Counter[1]: Señal de control hacia el contador para reemplazar el valor de la cuenta por IR.
 - Control_Counter[2]: Señal de control hacia el contador para reemplazar el valor de la cuenta por Registers.
- Control_SP: 2 bits señal para el comportamiento del SP.
 - Bit 0: Aumenta el conteo por 4.
 - Bit 1: Disminuye el conteo por cuatro.
- Qs: 85 bits. Señal de los Q de la máquina de estados del control para verificar
- Read: Señal para la memoria de escritura o lectura
 - Read = 1 se activa la lectura
 - Read = 0 se activa la escritura
- Error: 1 bit Señal de error de la máquina de estados
- Control_CSR: 18 bits. Señal de activación de los CSR,
- ACK: 4 bits, bits de respuesta de interrupciones

ALU:

Aritmetic logic unit. Bloque que realiza las operaciones aritméticas y lógicas del procesador, generando a partir de una señal de control el resultado de una operación específica a los datos ingresados.

- Entradas:
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - Registers_ALU: 64 bits. Señal que transmite los datos de los registros a la ALU para ser operados.
 - IR_ALU: 30 bits. Envía los datos que deben ser operados en la ALU, tales como datos inmediatos en las instrucciones.
 - IR_ALU (24:0): IR (31 : 7)
 - IR_ALU (30:25) IR(6:2)
 - Control_ALU: 37 bits. Señal del control a la ALU, que dependiendo del estado indica cual operación, con que datos se lleva acabo y en donde se almacena.
 - PC_ALU: 32 bits. Señal del PC que es operada para los saltos del PC.
- Salidas:
 - ALU_Registers: 32 bits. Señal de salida de la ALU, envía a los registros el resultado de una operación.

- ALU_PC: 32 bits, contiene el nuevo PC para ser almacenado.
- ALU_CSR: 32 bits. Señal de salida de la ALU, envía a los CSR el resultado de una operación.
- ALU_MAR: 32 bits señal que envía el resultado de una operación de la ALU al MAR
- ALU_Control: 4 bits
 - ALU_Control[0] Bandera de fin de operación de la ALU
 - ALU_Control[1] bandera de cero, se pone en uno si el resultado de una operación de ALU es cero.
 - ALU_Control[2] bit de signo resultado de ALU
 - ALU_Control[3] bit de carry resultado de ALU

Bloques internos ALU

A continuación se realiza una descripción funcional sobre los bloques internos de la ALU, en algunos casos se indica la cantidad de ciclos de reloj que requiere ese bloque para su operación, para los bloques que no tengan indicada la duración de su operación, esta es de 1 ciclo de reloj, excepto para los bloques de Shift, donde la duración de la operación depende de la cantidad de bits que se desea desplazar, con un tiempo de un ciclo de reloj por cada desplazamiento.

Decoder:

Debido a que los datos que ingresan a la ALU para ser operados están organizados de muchas formas, debido a los varios modos de direccionamiento, es necesario un *decoder* para encargarse de:

- Cuál de las entradas se debe operar.
- De qué forma están organizados los datos a operar.
- Cuál de las operaciones se debe llevar a cabo.
- De cuál de los bloques se genera la salida de la ALU.
- A cuál de las señales de asigna la salida del bloque.

El *decoder* de la ALU funciona mediante una señal de control que ha sido diseñada para agrupar los estados con las mismas operaciones y datos de entrada y salida, al encontrarse en alguno de esos estados agrupados se escribe un 1 en una de las señales correspondientes, esta señal llega al *decoder* de la ALU que realiza la operación con los datos de entrada y salida ya configurados para los estados de esa señal correspondiente.

Sumador:

El sumador realiza la suma o resta de dos datos de 32 bits, en el caso de la resta se realiza haciendo el complemento a 2 del segundo número, el resultado de la suma es un dato de 32 bits y el carry de salida del ultimo sumador, esta operación se realiza en un ciclo de reloj, el sumador esta construido usando una arquitectura Carry Ripple Adder

LogicalShiftRight:

El *Logical Shift Right* es un registro especial, que almacena 32 bits y de acuerdo con una señal de control, desplaza el contenido a la derecha un bit en un ciclo de reloj, reemplazando su bit más significativo, por un cero.

LeftShift

El *Left Logical Shift*, desplaza el contenido del registro hacia la izquierda una vez por ciclo de reloj reemplazando el bit menos significativo por cero.

ArithmeticShiftRight

El *Arithmetic Shift Right*, desplaza el contenido del registro hacia la derecha, replicando el valor del bit más significativo.

And, Or y Xor

El diseño de los bloques lógicos AND, OR y XOR se compone de 32 compuertas del respectivo tipo, AND, OR o XOR operando uno a uno los bits de los dos operandos de entrada y su resultado son los 32 bits de salida del bloque.

Multiplier32Bits

El bloque de multiplicación realiza la operación de multiplicación con signo entre dos datos con signo de 32 bits y a la salida entrega los 32 bits menos significativos de la respuesta, de acuerdo con la operación MUL, descrita en la sección 4. Esta multiplicación se realiza en un ciclo de reloj usando el algoritmo de Booth modificado.

CSR:

El Control and Status Registers, es un bloque de registros propio del estándar RISC-V, este bloque, está compuesto de un conjunto de registros útiles al sistema para desarrollar las instrucciones, como registros para interrupciones y variables de estado útiles al control.

- Entradas
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - ALU_CSR: 32 bits. Señal de salida de la ALU, envía a los CSR el resultado de una operación.
 - IR_CSRs: 12 bits. Envía las direcciones de los CSR donde se realiza la instrucción.
 - IR_CSRs(11:0) : IR(31:20)
 - Registers_CSR : 32 bits. Envía los datos para escribir en los CSR
 - Control_CSR: 19 bits. Señal de activación de los CSR

- Salidas
 - CSR_PC: 32 bits, contiene el nuevo valor para PC.
 - CSR_Registers : 32 bits. Envía a los registros datos
 - CSR_Control: 6 bits Señal que envía los datos de banderas importantes para cambios de estados y manejo de interrupciones al control.

Counter:

Bloque de conteo, se compone de un contador descendente activado por el control, que recibe de forma paralela un valor a contar, mediante una señal de control disminuye de forma unitaria por cada ciclo de reloj, al alcanzar el cero, levanta una bandera hacia control.

- Entradas
 - Reset 1 Bit. Señal para reiniciar el sistema
 - Reloj: 1 Bit. Señal de reloj para el funcionamiento de los bloques
 - Registers_Counter: 32 bits. Señal que contiene la cantidad a contar.
 - IR_Counter : 6 bits. Se usa para cargar directamente un valor al Counter
 - IR_Counter(5:0): (25:20)
 - Control_Counter: 3 bits. Señal de control para el contador.
 - Control_Counter[0]: Señal de control hacia el contador para iniciar la cuenta.
 - Control_Counter[1]: Señal de control hacia el contador para reemplazar el valor de la cuenta por IR.
 - Control_Counter[2]: Señal de control hacia el contador para reemplazar el valor de la cuenta por Registers.
- Salidas
 - Counter_Control: 1 Bit. Señal que notifica el fin del conteo.

SP:

Stack Pointer, es un registro especial que almacena la dirección de memoria correspondiente al Stack, un espacio especial de la memoria, donde se almacenan los datos para hacer llamados a subrutinas.

Entradas:

- Control_SP: 2 bits señal para el comportamiento del SP.
 - Bit 0: Aumenta el conteo por 4.
 - Bit 1: Disminuye el conteo por cuatro.

Salidas:

- SPRegister: salida de 32 bits que contiene el valor actual de SP

Aunque el *stack pointer* es un registro especial, de acuerdo con el estándar, se ubica en el bloque de registros, para escribir la información de este o usarlo para operaciones se usan las operaciones sobre el bloque de registros con la dirección 0 T 010.

3. Modos de direccionamiento:

El procesador Invenio RV implementa los conjuntos de instrucciones del estándar RISC-V pertenecientes al estándar RV32I y la extensión de multiplicación, mostrados en la Figura 4.

31	30	25 24	21 20	19	15 14	12 11	8	7	6	0	
		funct7		rs2	rs1	funct3		rd		opcode	R-type
		imm[11:0]			rs1	funct3		rd		opcode	I-type
		imm[11:5]		rs2	rs1	funct3	imm[4:0]			opcode	S-type
		imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]		opcode	B-type
		imm[31:12]					rd			opcode	U-type
		imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd			opcode	J-type

Figura. 4 Formatos de instrucción RV32I y RV32M

Los formatos de instrucción no son asignados de acuerdo con el tipo de instrucción, como operaciones en la ALU u operaciones sobre registros, sino sobre la operación específica que se requiera para esa instrucción. El estándar usado distribuye el espacio dentro de la instrucción manteniendo direcciones, datos y códigos de operación organizados y en medida de lo posible con los mismos tamaños y en el mismo espacio dentro de la instrucción.

Un elemento del formato de instrucción que es útil para la implementación es la identificación de los segmentos comunes, entre los cuales se tiene:

Código de operación: Primera parte del código de operación almacenada en los bits 0 al 6 de la instrucción, la segunda parte del código de operación, marcada como funct3, esta almacenada dentro de los bits 12 al 14.

Direcciones: Los formatos de instrucción, que contienen direcciones, almacenan las instrucciones en el mismo espacio del formato de instrucción, la dirección rd, desde el bit 7 al 11 de la instrucción, mientras que la dirección de rs1 y rs2 están almacenados desde los bits 15 al 19 para rs1, y 20 al 24 para rs2.

En el caso de las instrucciones de PUSH y POP tomadas de la extensión comprimida, el formato de instrucción cambia, al formato mostrado en la Figura. 5, en este caso ambas instrucciones usan un mismo formato de instrucción, que se presenta a continuación:

funct3	imm	rs2	op
--------	-----	-----	----

Figura. 5Formato instrucción comprimido PUSH y POP

En este formato, el opcode ocupa dos bits, desde el 0 al 1, mientras la segunda parte, el funct3, va desde el bit 13 al bit 15. La dirección del registro, en este caso rs2, se almacena del bit 2 al bit 6.

4. Instrucciones:

A continuación, se muestra una descripción de las instrucciones implementadas por el procesador Invenio RV, la forma de leer estas instrucciones es la siguiente:

NOMBRE DE LA INSTRUCCIÓN

Tipo de la instrucción de acuerdo con los tipos de la Figura 4. Código en binario de la operación

Formato específico de la instrucción.¹

Formato de la instrucción

Descripción en RTL de la instrucción. – Comentarios adicionales sobre la nomenclatura del RTL

LUI

Instrucción tipo U Opcode 0110111

Escribe en el registro con la dirección *rd* el dato almacenado en *immediate*, desplazado 12 bits a la izquierda y rellenado con ceros

31	immediate[31:12]	12 11	7 6	0
				0110111

Registers[IR(7:11)](0:20)<-(IR(12:31))

Registers[IR(7:11)](20:31)<-(0)

AUIPC

Instrucción tipo U Opcode 0010111

Escribe en el registro con la dirección *rd*, la suma del pc más el dato almacenado en *immediate*, desplazado 12 bits a la izquierda y rellenado con ceros

31	immediate[31:12]	12 11	7 6	0
				0010111

¹ En los formatos, debido a que se sigue el estándar RISC-V, a menos que se cambie el formato se usan las imágenes de los formatos de instrucción de la referencia [1] para evitar errores fruto de la reproducción propia de estos formatos debido a la gran cantidad.

Registers[IR(7:11)](0:20)<-(IR(12:31) + PC)

Registers[IR(7:11)](20:31)<-(0)

JAL

Instrucción tipo J Opcode 1101111

Escribe en el registro con la dirección *rd*, la dirección de la próxima instrucción (se le suma 4 al pc actual), luego almacena en el PC la suma de este mas el dato almacenado en *offset* extendido en signo, (es decir, los bits del 21 al 31 se rellenan con el bit más significativo de *offset*) con el siguiente orden desde el LSB al MSB : IR[21:30 ; 20; 12:19; 31]

31 offset[20 10:1 11 19:12]	12 11 rd	7 6 1101111	0
--------------------------------	-------------	----------------	---

Registers[IR (7:11)] <- (pc+4)

Pc<- Pc+ sext(IR (12:31))—sext = sign extended

JALR

Instrucción tipo J Opcode 1100111

Escribe en el registro con la dirección *rd*, la dirección de la próxima instrucción (se le suma 4 al pc actual), luego almacena en el PC la suma del contenido del registro con la dirección *rs1* más el dato almacenado en *offset* extendido en signo, (es decir, los bits del 12 al 31 se rellenan con el bit más significativo de *offset*).

31 offset[11:0]	20 19 rs1	15 14 000	12 11 rd	7 6 1100111	0
--------------------	--------------	--------------	-------------	----------------	---

Registers[IR(7:11)] <- PC+4;

Pc<- (Registers[IR(15:19)]+ sext(IR(20:31)))

BEQ

Instrucción tipo B OpCode 1100011 / 000 (Esta segunda parte del opcode, corresponde a los bits 12:14 del IR)

Compara el contenido del registro con dirección *rs1* con el contenido del registro con dirección *rs2*, si son iguales, suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se rellenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si el contenido de los registros es diferente, no hace nada.

31 offset[12 10:5]	25 24 rs2	20 19 rs1	15 14 000	12 11 offset[4:1 11]	7 6 1100011	0
-----------------------	--------------	--------------	--------------	-------------------------	----------------	---

If (Registers[IR (20:24)] == Registers[IR (15:19)])

Pc= + Pc + sext (offset)-- Desordenado según instrucción tipo B

BNB

Instrucción tipo B OpCode 1100011 / 001

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2, si son diferentes, suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se llenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si el contenido de los registros es igual, no hace nada.

31	25 24	20 19	15 14	12 11	7 6	0
offset[12 10:5]	rs2	rs1	001	offset[4:1 11]	1100011	

If (Registers[IR (20:24)] != Registers[IR (15:19)])

Pc<- Pc + sext (offset) -- Ordenado según instrucción tipo B

BLT

Instrucción tipo B Opcode 1100011 / 100

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2, si el contenido del registro con dirección rs1 es menor al contenido del registro con dirección rs2, (el contenido de ambos registros se trata como números de complemento a dos) suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se llenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si no es menor, no hace nada.

31	25 24	20 19	15 14	12 11	7 6	0
offset[12 10:5]	rs2	rs1	100	offset[4:1 11]	1100011	

If(Registers[IR (15:19)] < Registers[IR (20:24)])

Pc<- Pc + sext (offset) -- Ordenado según instrucción tipo B

BGE

Instrucción tipo B Opcode 1100011 / 101

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2, si el contenido del registro con dirección rs1 es mayor al contenido del registro con dirección rs2, (el contenido de ambos registros se trata como números de complemento a dos) suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se llenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si no es mayor, no hace nada.

31	25 24	20 19	15 14	12 11	7 6	0
offset[12 10:5]	rs2	rs1	101	offset[4:1 11]	1100011	

If(Registers[IR (15:19)] >= Registers[IR (20:24)])

Pc<- Pc + sext (offset) -- Ordenado según instrucción tipo B

BLTU

Instrucción tipo B Opcode 1100011 / 110

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2, si el contenido del registro con dirección rs1 es menor al contenido del registro con dirección rs2, (el contenido de ambos registros se trata como números sin signo) suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se llenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si no es menor, no hace nada.

31	25 24	20 19	15 14	12 11	7 6	0
offset[12 10:5]	rs2	rs1	110	offset[4:1 11]	1100011	

If(Registers[IR (15:19)] < Registers[IR (20:24)])

Pc<- Pc + sext (offset) -- Desordenado según instrucción tipo B

BGEU

Instrucción tipo B Opcode 1100011 / 111

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2, si el contenido del registro con dirección rs1 es mayor al contenido del registro con dirección rs2, (el contenido de ambos registros se trata como números sin signo) suma al pc actual el valor del *offset* en signo extendido (los bits del 13 al 31 del offset se llenan con el bit 12 del *offset*), almacenado en el *IR* de la siguiente forma, desde LSB a MSB: IR[8:11 ; 25:30 ; 7 ; 31] Si no es mayor, no hace nada.

31	25 24	20 19	15 14	12 11	7 6	0
offset[12 10:5]	rs2	rs1	111	offset[4:1 11]	1100011	

If(Registers[IR (15:19)] >= Registers[IR (20:24)])

Pc<- Pc + sext (offset) -- Desordenado según instrucción tipo B

LB

Instrucción tipo I Opcode 0000011 / 000

Carga un byte de memoria extendido en signo desde la dirección obtenida por la suma de offset y el contenido almacenado en el registro con dirección rs1. Escribe el dato cargado en memoria en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
offset[11:0]	rs1	000	rd	0000011	

I/O<-Memory[Registers[IR (15:19)]]

Registers[IR (7:11)]<- sext (I/O +sext(IR(20:31)))

LH

Instrucción tipo I Opcode 0000011 / 001

Carga dos bytes de memoria extendidos en signo desde la dirección obtenida por la suma de offset(también extendido en signo) y el contenido almacenado en el registro con dirección rs1. Escribe el dato cargado en memoria en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
offset[11:0]	rs1	001	rd	0000011	

I/O<-Memory[Registers[IR (15:19)]]-- carga 2 byte

Registers[IR (7:11)]<- sext(I/O +sext(IR(20:31)))

LW

Instrucción tipo I Opcode 0000011 / 010

Carga cuatro bytes de memoria extendidos en signo desde la dirección obtenida por la suma de offset(también extendido en signo) y el contenido almacenado en el registro con dirección rs1. Escribe el dato cargado en memoria en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
offset[11:0]	rs1	010	rd	0000011	

I/O<-Memory[Registers[IR (15:19)]]-- carga 4 byte

Registers[IR (7:11)]<- sext(I/O +sext(IR(20:31)))

LBU

Instrucción tipo I Opcode 0000011 / 100

Carga un byte de memoria extendido con ceros, desde la dirección obtenida por la suma de offset y el contenido almacenado en el registro con dirección rs1. Escribe el dato cargado en memoria en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
offset[11:0]	rs1	100	rd	0000011	

I/O<-Memory[Registers[IR (15:19)]] -- Carga 1 byte

Registers[IR (7:11)]<- (I/O +sext(IR(20:31)))

LHU

Instrucción tipo I Opcode 0000011 / 101

Carga dos bytes de memoria extendidos con ceros desde la dirección obtenida por la suma de offset(también extendido en signo) y el contenido almacenado en el registro con dirección rs1. Escribe el dato cargado en memoria en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
offset[11:0]	rs1	101	rd	0000011	

I/O<-Memory[Registers[IR (15:19)]]-- carga 2 byte

Registers[IR (7:11)]<-(I/O +sext(IR(20:31)))—

****NOTA:** Para facilidad en la arquitectura, las instrucciones SB, SH, SW cambian la posición de elementos de la instrucción, moviendo el offset de su posición original en los bits 7:11 | 25:31 para los bits 20:31 por lo que la implementación de estas instrucciones difiere en estos bits de la implementación típica del RV32I

** SB

Instrucción tipo S Opcode 0100011 / 000

Guarda en memoria el byte menos significativo del registro con dirección rs2, en la dirección de memoria obtenida por la suma de offset y el contenido almacenado en el registro con dirección rs1. El offset se encuentra en los bits bits 20:31.

31	30	25 24	21	20 19	15 14	12 11	8	7	6	0
		Offset		rs1	funct3		rs2		opcode	

Memory[IR(15:19+sext(offset))]<- LSB(Registers[IR(20:24)])

** SH

Instrucción tipo S Opcode 0100011 / 001

Guarda en memoria los dos bits menos significativos del registro con dirección rs2, en la dirección de memoria obtenida por la suma de offset y el contenido almacenado en el registro con dirección rs1. El offset se encuentra en los bits bits 20:31.

31	30	25 24	21	20 19	15 14	12 11	8	7	6	0
		Offset		rs1	funct3		rs2		opcode	

Memory[IR(15:19+sext(offset))]<- 2LSB(Registers[IR(20:24)])

** SW

Instrucción tipo S Opcode 0100011 / 010

Guarda en memoria los cuatro bits del registro con dirección rs2, en la dirección de memoria obtenida por la suma de offset y el contenido almacenado en el registro con dirección rs1. El offset se encuentra en los bits bits 20:31.

31	30	25 24	21	20 19	15 14	12 11	8	7	6	0
		Offset		rs1	funct3	rs2			opcode	

Memory[IR(15:19+sext(offset))]<- Registers[IR(20:24)]

ADDI

Instrucción tipo I Opcode 0010011 / 000

Suma el contenido del registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31] y almacena el resultado en el registro con dirección rd.

Nota: esta operación ignora el overflow aritmético.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	000	rd	0010011	

Registers[IR(7:11)]<- Registers[IR(15:19)] + IR(20:30)

SLTI

Instrucción tipo I Opcode 0010011 / 010

Compara el contenido del registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31], como números de complemento a dos. Si el contenido del registro con dirección rs1 es menor, escribe 1 en el registro con dirección rd, de lo contrario escribe 0 en este registro.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	010	rd	0010011	

Nota: Para facilidad del lector la notación de condicionales se hace usando notación de programación, usando if y else.

If (Registers[IR(15:19)] < IR(20:31)) Registers[IR(7:11)] <- 1—Números 2sC

Else Registers[IR(7:11)] <- 0

SLTIU

Instrucción tipo I Opcode 0010011 / 011

Compara el contenido del registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31], como números sin signo. Si el contenido del registro con dirección rs1 es menor, escribe 1 en el registro con dirección rd, de lo contrario escribe 0 en este registro.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	011	rd	0010011	

If (Registers[IR(15:19)] < IR(20:31)) Registers[IR(7:11)] <- 1 – Numeros unsigned
 Else Registers[IR(7:11)] <- 0

XORI

Instrucción tipo I Opcode 0010011 / 100

Calcula el XOR a nivel de bits entre el registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31], como números sin signo. Escribe el resultado en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	100	rd	0010011	

Registers [IR(7:11)] <- IR (20:31) XOR Registers [IR(15:19)]

ORI

Instrucción tipo I Opcode 0010011 / 110

Calcula el OR a nivel de bits entre el registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31], como números sin signo. Escribe el resultado en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	110	rd	0010011	

Registers [IR(7:11)] <- IR (20:31) OR Registers [IR(15:19)]

ANDI

Instrucción tipo I Opcode 0010011 / 011

Calcula el AND a nivel de bits entre el registro con dirección rs1 con el dato immediate extendido en signo, almacenado en los bits IR[20:31], como números sin signo. Escribe el resultado en el registro con dirección rd.

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	111	rd	0010011	

Registers [IR(7:11)] <- IR (20:31) OR Registers [IR(15:19)]

SLLI

Instrucción tipo I Opcode 0010011 / 001

Mueve el contenido del registro con dirección rs1 a la izquierda el número de bits (igual a la cantidad de desplazamientos a la izquierda) almacenado en shamt, en IR [20:25], se añaden ceros al desplazar. Escribe el resultado en el registro con dirección rd.

Nota: la instrucción solo es válida si shamt[5]=IR[25]=0

31	26 25	20 19	15 14	12 11	7 6	0
000000	shamt	rs1	001	rd	0010011	

Rg[IR(15:19)]<<--IR(20:25) cantidad de veces, reemplaza con 0

Rg[IR(7:11)]<-Rg[IR(15:19)]

SRLI

Instrucción tipo I Opcode 0010011 / 101 / 0000000 , este último opcode son los bits 25:30 del IR

Mueve el contenido del registro con dirección rs1 a la derecha el número de bits (igual a la cantidad de desplazamientos a la derecha) almacenado en shamt, en IR [20:25], se añaden ceros al desplazar. Escribe el resultado en el registro con dirección rd.

Nota: la instrucción solo es válida si shamt[5]=IR[25]=0

31	26 25	20 19	15 14	12 11	7 6	0
000000	shamt	rs1	101	rd	0010011	

Rg[IR(15:19)] >> --IR(20:25) cantidad de veces, reemplaza con 0

Rg[IR(7:11)]<-Rg[IR(15:19)]

SRAI

Instrucción tipo I Opcode 0010011 / 011 / 010000 – este último bit 25:30 del IR

Mueve el contenido del registro con dirección rs1 a la derecha el número de bits (igual a la cantidad de desplazamientos a la derecha) almacenado en shamt, en IR [20:25], se añaden copias del MSB del dato al desplazar. Escribe el resultado en el registro con dirección rd.

Nota: la instrucción solo es valida si shamt[5]=IR[25]=0

31	26 25	20 19	15 14	12 11	7 6	0
010000	shamt	rs1	101	rd	0010011	

Rg[IR(15:19)] >> --IR(20:25) cantidad de veces , reemplaza con el MSB

Rg[IR(7:11)]<-Rg[IR(15:19)]

ADD

Instrucción tipo R Opcode 0110011 / 000 / 0000000 Este último opcode son bits del 25 al 31 del IR

Suma el contenido del registro con dirección rs2 al contenido del registro con dirección rs1 y el resultado lo almacena en el registro con dirección rd.

Nota: esta instrucción ignora el overflow aritmético.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	000	rd	0110011	

Rg[IR (7:11)] 7:11 = Rg[IR (15:19)] - Rg[IR (20:24)]

SUB

Instrucción tipo R Opcode 0110011 / 000 / 0100000 Este opcode se diferencia de ADD solo por el bit 30 del IR

Resta el contenido del registro con dirección rs2 al contenido del registro con dirección rs1 y el resultado lo almacena en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	000	rd	0110011	

Rg[IR (7:11)] 7:11 = Rg[IR (15:19)] - Rg[IR (20:24)]

SLL

Instrucción tipo R Opcode: 0110011 / 001

Mueve el contenido del registro con dirección rs1 a la izquierda, la cantidad de veces almacenada en el registro con dirección rs2 (se toman solo los 5 bits menos significativos), llenando los bits con 0, el resultado es escrito en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	001	rd	0110011	

Rg[IR (15:19)] <<-- { Rg[IR(20:24)] (0:5) } es la cantidad de veces que rellena con 0

Rg[IR (7:11)] <- Rg[IR (15:19)]

SLT

Instrucción tipo R Opcode: 0110011 / 010

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2 (trata ambos números, como números con signo). Si el contenido del registro con dirección rs1 es menor, escribe en el registro con dirección rd 1, de lo contrario escribe 0. Tanto el contenido del registro

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	010	rd	0110011	

If (Registers[IR(15:19)] < Registers[IR(20:24)]) Registers[IR(7:11)] <- 1 – Números sin signo

Else Registers[IR(7:11)] <- 0

SLTU

Instrucción tipo R Opcode: 0110011 / 011

Compara el contenido del registro con dirección rs1 con el contenido del registro con dirección rs2. (trata ambos números, como números sin signo) Si el contenido del registro con dirección rs1 es menor, escribe en el registro con dirección rd 1, de lo contrario escribe 0.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	011	rd	0110011	

If (Registers[IR(15:19)] < Registers[IR(20:24)]) Registers[IR(7:11)] <- 1 – Números sin signo
Else Registers[IR(7:11)] <- 0

XOR

Instrucción tipo R Opcode: 0110011 / 100

Calcula el XOR bit a bit entre el contenido de los registros con dirección rs1 y rs2, el resultado lo escribe en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	100	rd	0110011	

Rg[IR (7:11)] 7:11 = Rg[IR (15:19)] XOR Rg[IR (20:24)]

SRL

Instrucción tipo R Opcode: 0110011 / 101 / 0000000 se diferencia de sra por el bit 30 del IR

Mueve el contenido del registro con dirección rs1 a la derecha, la cantidad de veces almacenada en el registro con dirección rs2 (se toman solo los 5 bits menos significativos), llenando los bits con 0, el resultado es escrito en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	101	rd	0110011	

Rg[IR (15:19)] >> -- { Rg[IR(20:24)] (0:5) } veces que rellena con 0

Rg[IR (7:11)] <- Rg[IR (15:19)]

SRA

Instrucción tipo R Opcode: 0110011 / 101 / 0100000

Mueve el contenido del registro con dirección rs1 a la derecha, la cantidad de veces almacenada en el registro con dirección rs2 (se toman solo los 5 bits menos significativos), llenando los bits con el bit más significativo, el resultado es escrito en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0100000	rs2	rs1	101	rd	0110011	

Rg[IR (15:19)] >> -- { Rg[IR(20:24)] (0:5) } veces, rellena con el MSB de Rg[IR (15:19)]

Rg[IR (7:11)] <- Rg[IR (15:19)]

OR

Instrucción tipo R Opcode: 0110011 / 110

Calcula el OR bit a bit entre el contenido de los registros con dirección rs1 y rs2, el resultado lo escribe en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	110	rd	0110011	

Rg[IR (7:11)] 7:11 = Rg[IR (15:19)] OR Rg[IR (20:24)]

AND

Instrucción tipo R Opcode: 0110011 / 111

Calcula el AND bit a bit entre el contenido de los registros con dirección rs1 y rs2, el resultado lo escribe en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	111	rd	0110011	

Rg[IR (7:11)] 7:11 = Rg[IR (15:19)] AND Rg[IR (20:24)]

CSRRW

Instrucción tipo I Opcode 1110011 / 001

Primero escribe el contenido del Control And Status Register de la dirección csr(bits del 20 al 31 del IR) en el registro con dirección rd, luego escribe el contenido del registro con dirección rs1 en el Control And Status Register con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr		rs1	001	rd	1110011

Rg[IR(7:11)] <- CSRs [IR(20:31)]

CSRs [IR(20:31)] <- Rg[IR(15:19)]

CSRRS

Instrucción tipo I Opcode 1110011 / 001

Almacena el valor del CSRs en la posición csr (bits del 20 al 31 del IR) en el registro con dirección rd, escribe el OR entre este valor y el contenido del registro con dirección rs1 en el CSR con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr		rs1	010	rd	1110011

Rg[IR(7:11)] <- CSR [IR(20:31)]

CSR [IR(20:31)] <- Rg[IR(15:19)] OR Rg[IR(7:11)]

CSRRC

Instrucción tipo I Opcode 1110011 / 011

Primero escribe el contenido del Control And Status Register de la dirección csr(bits del 20 al 31 del IR) en el registro con dirección rd, luego escribe el AND entre este valor y el contenido del registro con dirección rs1 en el CSR con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr	rs1	011	rd	1110011	

Rg[IR(7:11)] <- CSR [IR(20:31)]

CSR [IR(20:31)] <- Rg[IR(15:19)] OR Rg[IR(7:11)]

CSRRWI

Instrucción tipo I Opcode 1110011 / 011

Primero escribe el contenido del Control And Status Register de la dirección csr(bits del 20 al 31 del IR) en el registro con dirección rd, luego escribe el dato zimm IR[15:19] extendido en ceros en el CSR con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr	zimm[4:0]	101	rd	1110011	

Rg[IR(7:11)] <- CSR [IR(20:31)]

CSR [IR(20:31)] <- IR(15:19) -- cero extendido

CSRRSI

Instrucción tipo I Opcode 1110011 / 110

Primero escribe el contenido del Control And Status Register de la dirección csr(bits del 20 al 31 del IR) en el registro con dirección rd, luego escribe el OR entre este valor y el dato zimm IR[15:19] extendido en ceros en el CSR con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr	zimm[4:0]	110	rd	1110011	

Rg[IR(7:11)] <- CSR [IR(20:31)]

CSR [IR(20:31)] <- IR(15:19) OR Rg[IR(7:11)]

CSRRCI

Instrucción tipo I Opcode 1110011 / 111

Primero escribe el contenido del Control And Status Register de la dirección csr(bits del 20 al 31 del IR) en el registro con dirección rd, luego escribe el AND entre este valor y el dato zimm IR[15:19] extendido en ceros en el CSR con dirección csr.

31	20 19	15 14	12 11	7 6	0
csr	zimm[4:0]	111	rd	1110011	

Rg[IR(7:11)] <- CSR [IR(20:31)]

CSR [IR(20:31)] <- 1sC(IR(15:19)) AND Rg[IR(7:11)]

MUL

Instrucción tipo R Opcode 0110011 000

Multiplica el contenido del registro con dirección rs1 y rs2. Almacena el resultado de la multiplicación en el registro con dirección rd.

31	25 24	20 19	15 14	12 11	7 6	0
0000001	rs2	rs1	000	rd	0110011	

SWSP (PUSH)

Almacena una palabra en la memoria en la dirección del SP + uimm11

15	13 12	7 6	2 1	0
110	uimm[5:2 7:6]		rs2	10

LWSP (POP)

Carga la palabra en el registro con dirección rd, desde la memoria con dirección sp + uimm

15	13	12	11	7 6	2 1	0
010	uimm[5]		rd	uimm[4:2 7:6]	10	

5. Referencias:

- [1] “guia-practica-de-risc-v-1.0.5.pdf”. Consultado: jun. 16, 2020. [En línea]. Disponible en: <http://riscvbook.com/spanish/guia-practica-de-risc-v-1.0.5.pdf>.

6. Anexos:

- [1] Documento de trabajo de grado, disponible en:
https://livejaverianaedu.sharepoint.com/sites/TrabajodegradoprocesadorRISC-V/_layouts/15/Doc.aspx?OR=teams&action=edit&sourcedoc={D3AA05D2-77AB-423D-9F05-EE2912ED43C5}