Ejercicios Tema 2

Ejercicio 1. Dar tres ejemplos de la arquitectura RISC-V de cada uno de Los principios de diseño de la arquitectura: (1) la regularidad apoya la simplicidad; (2) hacer el caso común rápido; (3) más pequeño es más rápido; y (4) buenas demandas de diseño buenos compromisos. Explica cómo cada uno de tus ejemplos exhibe el diseño principio.

1. Regularidad apoya la simplicidad:

- La estructura de la arquitectura RISC-V es altamente regular y predecible, lo que facilita el diseño y la implementación de hardware y software. Por ejemplo, todas las instrucciones siguen un formato fijo de 32 bits, lo que simplifica la decodificación y la ejecución. También existen un conjunto reducido de tipos de instrucciones, lo que simplifica el diseño de la unidad de decodificación.
- RISC-V utiliza una estructura de registro de ventana, lo que permite una implementación más simple del procesador y reduce la cantidad de registros necesarios en el hardware.
- El conjunto de instrucciones RISC-V está organizado en clases, lo que facilita el diseño y la implementación de las unidades de ejecución y reduce la complejidad del hardware.

2. Hacer el caso común rápido:

- RISC-V utiliza el registro cero como un registro especial que siempre contiene el valor cero. Esto permite que las operaciones comunes, como la carga de una dirección de memoria, se realicen de manera más eficiente.
- Las instrucciones RISC-V pueden ser ejecutadas en una sola etapa, lo que permite una ejecución más rápida de las instrucciones y una mejora en el rendimiento general del procesador.
- El conjunto de instrucciones RISC-V tiene un enfoque en las operaciones básicas, lo que significa que las operaciones más comunes, como las operaciones aritméticas y lógicas, se pueden ejecutar de manera más rápida y eficiente.

3. Más pequeño es más rápido:

- RISC-V utiliza registros de 32 bits en lugar de registros de 64 bits, lo que reduce el tamaño del procesador y mejora su eficiencia.
- La estructura de registro de ventana utilizada en RISC-V permite una implementación más compacta del procesador y reduce la cantidad de registros necesarios.
- RISC-V utiliza una arquitectura de carga y almacenamiento, en la que todas las operaciones que implican el acceso a la memoria se realizan a través de un conjunto reducido de instrucciones, lo que reduce el tamaño del procesador.

4. Buenas demandas de diseño buenos compromisos:

- RISC-V incluye un conjunto básico de instrucciones que se consideran esenciales para cualquier procesador, pero también permite la inclusión de extensiones opcionales para añadir funcionalidades específicas a los procesadores.
- La arquitectura RISC-V está diseñada para ser escalable, lo que significa que puede ser utilizada en sistemas desde dispositivos IoT hasta servidores de alto rendimiento.
- RISC-V se diseñó con un enfoque en la modularidad, lo que significa que los diferentes componentes del procesador pueden ser intercambiados para adaptarse a las necesidades específicas del sistema. Esto permite una mayor flexibilidad en el diseño y la implementación de procesadores basados en RISC-V.

Ejercicio 2. La arquitectura RISC-V tiene un conjunto de registros que consta de 32 registros de 32 bits. ¿Es posible diseñar una arquitectura de computador sin un ¿Conjunto de registro? Si es así, describa brevemente la arquitectura, incluido el conjunto de instrucciones.
¿Cuáles son las ventajas y desventajas de esta arquitectura sobre el RISC-V?
¿arquitectura?

Sí, es posible diseñar una arquitectura de computador sin un conjunto de registros. Una posible alternativa es la arquitectura de acumulador, en la que todas las operaciones se realizan entre el acumulador y los operandos de la memoria. En

esta arquitectura, los operandos se cargan desde la memoria en el acumulador, se realizan las operaciones necesarias y el resultado se almacena de nuevo en la memoria.

El conjunto de instrucciones para esta arquitectura sería diferente al de RISC-V y estaría diseñado para trabajar con el acumulador y los operandos de la memoria. Las instrucciones podrían incluir operaciones aritméticas y lógicas, como sumas y restas, y operaciones de transferencia de datos, como cargas y almacenamientos.

Una ventaja de esta arquitectura es su simplicidad, ya que elimina la complejidad de tener que gestionar un conjunto de registros. Sin embargo, esto también puede ser una desventaja, ya que puede limitar la capacidad de procesamiento y la flexibilidad del procesador. Además, las operaciones que implican múltiples operandos pueden ser menos eficientes en esta arquitectura, ya que requieren múltiples accesos a la memoria.

Ejercicio 3. La instrucción nor no forma parte del conjunto de instrucciones RISC-V porque la misma funcionalidad se puede implementar utilizando las instrucciones existentes. Escriba un fragmento de código de ensamblado corto que tenga la siguiente funcionalidad: s3 = s4 NOR s5. Utilice la menor cantidad de instrucciones posible.

La operación NOR se puede implementar utilizando las instrucciones NOT y AND en RISC-V. El fragmento de código para realizar s3 = s4 NOR s5 utilizando el número mínimo de instrucciones es:

```
not s4, s4
not s5, s5
and s3, s4, s5
```

Ejercicio 4. Convertir el siguiente código ensamblador RISC-V en lenguaje máquina. Escriba las instrucciones en hexadecimal.

```
addi s3, s4, 28
sll t1, t2, t3
```

```
srli s3, s1, 14
sw s9, 16(t4)
```

El código ensamblador RISC-V muestra cuatro instrucciones diferentes:

- La primera instrucción `addi s3, s4, 28` suma el valor 28 al registro s4 y almacena el resultado en el registro s3 utilizando una operación de suma inmediata.
- La segunda instrucción `sll t1, t2, t3` realiza un desplazamiento a la izquierda de los bits en el registro t2 en la cantidad especificada en el registro t3, y almacena el resultado en el registro t1 utilizando una operación de desplazamiento a la izquierda lógico.
- La tercera instrucción `srli s3, s1, 14` realiza un desplazamiento a la derecha lógico de los bits en el registro s1 en la cantidad especificada en 14, y almacena el resultado en el registro s3 utilizando una operación de desplazamiento a la derecha lógico inmediato.
- La cuarta instrucción `sw s9, 16(t4)` almacena el contenido del registro s9 en la dirección de memoria especificada por el registro t4 más un desplazamiento de 16 bytes, utilizando una operación de almacenamiento en palabra.

En lenguaje máquina, el código se vería así:

```
0x00178793
0x00b505b3
0x00e12ea6
0x02318723
```

Para convertir el código de ensamblado RISC-V en lenguaje máquina, se sigue el siguiente proceso:

1. Se identifica la instrucción en el código de ensamblado.
2. Se busca el código de operación de la instrucción en la tabla de instrucciones RISC-V.
3. Se identifican los códigos de registro y valores inmediatos involucrados en la instrucción.
4. Se convierten los códigos de operación, códigos de registro y valores inmediatos en valores hexadecimales de 32 bits.

5. Se combinan los valores hexadecimales para formar la instrucción en lenguaje máquina.

Por ejemplo, para la instrucción `addi s3, s4, 28`:

1. La instrucción es `addi s3, s4, 28`.
2. El código de operación para la suma inmediata es `0x001`.
3. Los códigos de registro son `s3`, `s4` y el valor inmediato `28`.
4. Los códigos de registro `s3` y `s4` se convierten en sus valores hexadecimales correspondientes (`0x13` y `0x9` respectivamente), y el valor inmediato `28` se convierte en su valor hexadecimal (`0x1c`).
5. Los valores hexadecimales se combinan en orden para formar la instrucción en lenguaje máquina: `0x00178793`.

Este proceso se repite para cada instrucción en el código de ensamblado, hasta que se hayan convertido todas las instrucciones en lenguaje máquina.

Ejercicio 5. 5.1 ¿Cuál es el número máximo de instrucciones que una rama como la instrucción (como beq) puede ramificarse hacia adelante (es decir, a direcciones de instrucción superiores)?
5.2 ¿Cuál es el número máximo de instrucciones que una rama que la instrucción (como beq) puede ramificarse hacia atrás (es decir, para direcciones de instrucción más bajas)?

El número máximo de instrucciones que una rama como la instrucción (como beq) puede ramificarse hacia adelante es 1023, ya que el campo de desplazamiento de la instrucción (que especifica el número de instrucciones que se deben saltar hacia adelante) tiene una longitud de 12 bits. Esto significa que puede representar valores de desplazamiento entre 0 y 2^11-1 (es decir, 0 y 2047). Sin embargo, el número de instrucciones que se pueden saltar hacia adelante se reduce a la mitad, ya que el desplazamiento se interpreta como un número con signo. Por lo tanto, el número máximo de instrucciones que una rama como la instrucción (como beq) puede ramificarse hacia adelante es (2^11-1)/2 = 1023.

El número máximo de instrucciones que una rama como la instrucción (como beq) puede ramificarse hacia atrás es 4095, ya que el campo de desplazamiento de la instrucción (que especifica el número de instrucciones que se deben saltar hacia atrás) tiene una longitud de 12 bits. Esto significa que puede representar valores de

desplazamiento entre 0 y 2^11-1 (es decir, 0 y 2047). Sin embargo, el número de instrucciones que se pueden saltar hacia atrás se reduce a la mitad, ya que el desplazamiento se interpreta como un número con signo. Por lo tanto, el número máximo de instrucciones que una rama como la instrucción (como beq) puede ramificarse hacia atrás es (2^11-1)*2 = 4095.

Ejercicio 6. Escribir código ensamblador que se bifurque condicionalmente a una instrucción 32 Minstrucciones enviadas de una instrucción dada. Recordemos que 1 Minstruction = 220 instrucciones = 1.048.576 instrucciones. Suponga que el código comienza en la dirección 0x8000. Utilice un número mínimo de instrucciones.

Para bifurcarse condicionalmente a una instrucción 32 Minstrucciones enviadas de una instrucción dada, se puede utilizar la instrucción `beq` en RISC-V. La instrucción `beq` compara dos registros y salta a una dirección de memoria específica si son iguales.

Para saltar a 32 Minstrucciones (es decir, 32 * 2^20 instrucciones) de una instrucción dada, se puede utilizar un registro como contador y sumarle un valor inmediato equivalente a 32 Minstrucciones. Luego, se puede comparar este registro con otro registro que contenga un valor específico y bifurcarse a una dirección de memoria específica si son iguales.

El siguiente fragmento de código ensamblador muestra cómo se puede implementar esto en RISC-V:

```
# cargar el valor de la dirección de destino en un registro
lui t0, %hi(destino)  # cargar los bits altos de la dirección de destino en t0
addi t0, t0, %lo(destino)  # cargar los bits bajos de la dirección de destino en t0

# cargar el valor del contador en un registro
li t1, 32*2^20  # cargar el valor de 32 Minstrucciones en t1

# sumar el valor de 32 Minstrucciones al contador
add t2, ra, t1

# comparar el valor del contador con el valor específico en otro registro
beq t2, t3, destino  # saltar a la dirección de destino si son iguales

# continuar con el resto del código aquí

destino:
# código a ejecutar después de la bifurcación
```

En este código, `destino` es la etiqueta de la dirección de memoria de destino a la que se desea bifurcar. El código primero carga el valor de la dirección de destino en un registro (`t0`) y el valor de 32 Minstrucciones en otro registro (`t1`). Luego, suma el valor de 32 Minstrucciones al contador (`ra`) y almacena el resultado en otro registro (`t2`). Finalmente, compara el valor de `t2` con el valor de otro registro (`t3`) y si son iguales, salta a la etiqueta `destino`.

Este código utiliza un número mínimo de instrucciones y se puede ajustar para diferentes valores de Minstrucciones y direcciones de destino.

Código de operación	Formato de instrucción	Nombre de instrucción	Descripción
0110111	U-type	LUI	Carga la parte superior de una dirección en un registro
0010111	U-type	AUIPC	Carga la parte superior de una dirección en un registro y agrega un desplazamiento inmediato
1101111	UJ-type	JAL	Salta a una dirección de memoria y almacena la dirección de retorno en un registro
1100111	I-type	JALR	Salta a una dirección de memoria y almacena la dirección de retorno en un registro, basado en un índice y un desplazamiento inmediatos
1100011	SB-type	BEQ	Salta a una dirección de memoria si dos registros son iguales
1100011	SB-type	BNE	Salta a una dirección de memoria si dos registros son diferentes
1100011	SB-type	BLT	Salta a una dirección de memoria si un registro es menor que otro
1100011	SB-type	BGE	Salta a una dirección de memoria si un registro es mayor o igual que otro
1100011	SB-type	BLTU	Salta a una dirección de memoria si un registro sin signo es menor que otro
1100011	SB-type	BGEU	Salta a una dirección de memoria si un registro sin signo es mayor o

			igual que otro
0000011	I-type	LB	Carga un byte con signo de la memoria en un registro
0000011	I-type	LH	Carga media palabra con signo de la memoria en un registro
0000011	I-type	LW	Carga una palabra con signo de la memoria en un registro
0000011	I-type	LBU	Carga un byte sin signo de la memoria en un registro
0000011	I-type	LHU	Carga media palabra sin signo de la memoria en un registro
0100011	S-type	SB	Almacena un byte en la memoria
0100011	S-type	SH	Almacena media palabra en la memoria
0100011	S-type	SW	Almacena una palabra en la memoria
0010011	I-type	ADDI	Suma un valor inmediato a un registro
0010011	I-type	SLTI	Establece un registro en 1 si es menor que un valor inmediato, de lo contrario establece en 0
0010011	I-type	SLTIU	Establece un registro en 1 si es menor que un valor inmediato sin signo, de lo contrario establece en 0
0010011	I-type	XORI	Realiza una operación XOR entre un registro y un valor inmediato
0010011	I-type	ORI	Realiza una operación OR entre un registro y un valor inmediato
0010011	I-type	ANDI	Realiza una operación AND entre un registro y un valor inmediato
0010011	I-type	SLLI	Realiza un desplazamiento a la izquierda lógico en un registro con un valor inmediato
0010011	I-type	SRLI	Realiza un desplazamiento a la derecha lógico en un registro con un valor inmediato
0010011	I-type	SRAI	Realiza un desplazamiento a la derecha aritmético en un registro

			con un valor inmediato
0110011	R-type	ADD	Suma dos registros
0110011	R-type	SUB	Resta dos registros
0110011	R-type	SLL	Realiza un desplazamiento a la izquierda lógico en un registro con otro registro
0110011	R-type	SLT	Establece un registro en 1 si es menor que otro registro, de lo contrario establece en 0
0110011	R-type	SLTU	Establece un registro en 1 si es menor que otro registro sin signo, de lo contrario establece en 0
0110011	R-type	XOR	Realiza una operación XOR entre dos registros
0110011	R-type	SRL	Realiza un desplazamiento a la derecha lógico en un registro con otro registro
0110011	R-type	SRA	Realiza un desplazamiento a la derecha aritmético en un registro con otro registro
0110011	R-type	OR	Realiza una operación OR entre dos registros
0110011	R-type	AND	Realiza una operación AND entre dos registros
0110111	U-type	FENCE	Sincroniza la ejecución de instrucciones
1110011	I-type	ECALL	Llama a un servicio del sistema operativo
1110011	I-type	EBREAK	Detiene la ejecución y entra en modo de depuración

La tabla muestra todas las instrucciones de RISC-V, su formato de instrucción, su nombre y una breve descripción de su función. Las instrucciones se dividen en varias categorías, como instrucciones de carga y almacenamiento, instrucciones aritméticas y lógicas, instrucciones de bifurcación condicional y otras instrucciones especiales.

Las instrucciones se representan mediante un código de operación de 7 bits, que se utiliza para identificar la instrucción. Cada instrucción tiene un formato de instrucción

específico, que indica los registros y los valores inmediatos involucrados en la instrucción. La mayoría de las instrucciones tienen un formato de 32 bits, aunque algunas, como las instrucciones de salto y las instrucciones de carga con signo, tienen formatos especiales.

Las instrucciones de RISC-V son diseñadas para ser simples y regulares, lo que facilita su implementación en hardware y software. Las instrucciones se pueden combinar en una variedad de formas para crear programas complejos y eficientes.

La siguiente tabla muestra algunas de las instrucciones de RISC-V junto con ejemplos de su uso:

Código de operación	Formato de instrucción	Nombre de instrucción	Ejemplo
0110111	U-type	LUI	`lui t0, 0x10000` carga la dirección `0x10000000` en el registro `t0`
0010111	U-type	AUIPC	`auipc t0, 0x10000` carga la dirección actual más `0x10000000` en el registro `t0`
1101111	UJ-type	JAL	`jal destino` salta a la dirección de etiqueta `destino` y almacena la dirección de retorno en el registro `ra`
1100111	I-type	JALR	`jalr t0, t1, 16` salta a la dirección en el registro `t1` más un desplazamiento de 16 bytes y almacena la dirección de retorno en el registro `t0`
1100011	SB-type	BEQ	`beq t0, t1, destino` salta a la dirección de etiqueta `destino` si los registros `t0` y `t1` son iguales
1100011	SB-type	BNE	`bne t0, t1, destino` salta a la dirección de etiqueta `destino` si los registros `t0` y `t1` son diferentes
1100011	SB-type	BLT	`blt t0, t1, destino` salta a la dirección de etiqueta `destino` si el registro `t0` es menor que el registro `t1`
1100011	SB-type	BGE	`bge t0, t1, destino` salta a la dirección de etiqueta `destino` si el registro `t0` es mayor o igual que el registro `t1`

1100011	SB-type	BLTU	`bltu t0, t1, destino` salta a la dirección de etiqueta `destino` si el registro `t0` es menor que el registro `t1` sin signo
1100011	SB-type	BGEU	`bgeu t0, t1, destino` salta a la dirección de etiqueta `destino` si el registro `t0` es mayor o igual que el registro `t1` sin signo
0000011	I-type	LB	`lb t0, 16(t1)` carga un byte con signo de la dirección en el registro `t1` más un desplazamiento de 16 bytes en el registro `t0`
0000011	I-type	LH	`lh t0, 16(t1)` carga media palabra con signo de la dirección en el registro `t1` más un desplazamiento de 16 bytes en el registro `t0`
0000011	I-type	LW	`lw t0, 16(t1)` carga una palabra con signo de la dirección en el registro `t1` más un desplazamiento de 16 bytes en el registro `t0`
0000011	I-type	LBU	`lbu t0, 16(t1)` carga un byte sin signo de la dirección en el registro `t1` más un desplazamiento de 16 bytes en el registro `t0`
0000011	I-type	LHU	`lhu t0, 16(t1)` carga media palabra sin signo de la dirección en el registro `t1` más un desplazamiento de 16 bytes en el registro `t0`
0100011	S-type	SB	`sb t0, 16(t1)` almacena un byte en la dirección en el registro `t1` más un desplazamiento de 16 bytes desde el registro `t0`
0100011	S-type	SH	`sh t0, 16(t1)` almacena media palabra en la dirección en el registro `t1` más un desplazamiento de 16 bytes desde el registro `t0`
0100011	S-type	SW	`sw t0, 16(t1)` almacena una palabra en la dirección en el registro `t1` más un desplazamiento de 16 bytes desde el registro `t0`

0010011	I-type	ADDI	`addi t0, t1, 16` suma el valor inmediato 16 al registro `t1` y almacena el resultado en el registro `t0`
0010011	I-type	SLTI	`slti t0, t1, 16` establece el registro `t0` en 1 si el registro `t1` es menor que el valor inmediato 16, de lo contrario establece en 0
0010011	I-type	SLTIU	`sltiu t0, t1, 16` establece el registro `t0` en 1 si el registro `t1` es menor que el valor inmediato 16 sin signo, de lo contrario establece en 0
0010011	I-type	XORI	`xori t0, t1, 16` realiza una