Sistemas Digitales

Preguntas teóricas para el parcial 1C2024

1. **Observando las instrucciones que realizan operaciones aritméticas, describir a qué componentes conectaría la ALU en el datapath. Explicar detalladamente el funcionamiento del siguiente programa en assembler:**

```
main:
    addi a1, x0, 1
    addi a2, x0, 2
    bltu a2, a1, test
    sub a3, a1, a2
    jal end
test:
    sub a3, a2, a1
end:
    nop
```

Voy a listar las líneas del programa para, en cada caso, notar qué conexiones son requeridas en el datapath:

```
addi a1, x0, 1
addi a2, x0, 2
```

- Register File para leer y modificar registros.
- Extensor con signo para los inmediatos.
- Multiplexor para seleccionar los inmediatos como entradas para la ALU.
- ALU para la suma de inmediatos y registros.

```
bltu a2, a1, test
```

- Register File para acceder a los registros.
- Extensor con signo para el offset de test.
- Multiplexor para seleccionar el offset extendido como entrada para la ALU.
- ALU para la suma de PC+offset.
- ALU para realizar la comparación.
- Multilplexor para que el valor del PC sea PC+offset en caso de que la comparación sea verdadera.

```
sub a3, a2, a1
```

- Register File para acceder y modificar los registros.
- ALU para realizar la resta.

Por lo tanto, y para correr este programa, en el datapath la ALU estará conectada principalmente al Register File, multiplexores y unidades de control. Estas conexiones permiten que la ALU reciba los operandos adecuados y realice las operaciones aritmético-lógicas necesarias, y luego envíe los resultados a los registros correspondientes o tomar decisiones en instrucciones de bifuración.

Por otra parte, la explicación detallada del programa es la siguiente:

- En el registro a1 se carga el valor 1.
- En el registro a2 se carga el valor 2.
- Se comparan los valores de los registros sin tener en cuenta el signo. Si a2<a1, el PC incrementa lo necesario para saltar a la instrucción que está en la dirección de la etiqueta test. Como los valores ya los tenemos definidos, sabemos que no va a haber salto porque a2>a1. Por lo tanto, se continúa con la siguiente instrucción (PC+4).
- Se hace la resta de a2-a1 y se guarda el resultado en el registro a3.
- Se realiza un salto hacia la etiqueta end, guardando el valor PC+4 en el registro ra.
- Se termina la ejecución, ya que la etiqueta nop no realiza ninguna operación, y es la última instrucción.

2. ¿Qué significa que la arquitectura de RISC-V sea modular? ¿Qué ventajas puede tener esto?

La arquitectura RISC-V se describe como modular porque está diseñada con un conjunto básico de instrucciones obligatorias, conocidas como el *ISA base*, y varios conjuntos de instrucciones opcionales, llamados *extensiones*. Este enfoque modular permite a los diseñadores de hardware elegir solo las partes del conjunto de instrucciones que necesitan para su aplicación específica, lo que facilita la creación de procesadores personalizados y optimizados al dar mayor flexibilidad, eficiencia, simplicidad y un ecosistema abierto.

3. ¿Cuáles son los objetivos no funcionales (o métricas de diseño) qué guían el diseño del set de instrucciones de RISC-V? ¿Cómo impactan en su diseño?

El diseño del conjunto de instrucciones (ISA) de RISC-V se guía por varios objetivos no funcionales. A continuación se describen estas métricas, y cómo influyen en el diseño de RISC-V:

- *Costo*: reducción de costos manteniendo simplicidad.
- *Simplicidad*: reduce tiempo y costos de desarrollo, facilita la verificación y disminuye la complejidad para los programadores y compiladores.
- *Rendimiento*: permite ejecutar instrucciones más rápidamente.
- *Aislamiento de arquitectura e implementación*: permite flexibilidad y adaptabilidad a diferentes tecnologías y mejoras futuras sin cambios significativos en el software.
- *Espacio para crecer*: permitir extensiones opcionales según sea necesario para dar una base estable al desarrollo de software.
- *Tamaño del programa*: minimizar el tamaño del código generado por el compilador para reducir la cantidad de memoria necesaria y mejorar el rendimiento del caché.
- *Facilidad de programar, compilar y linkear*: RISC-V cuenta con 32 registros, por lo que facilita la tarea de programar y compilar. Además, permite hacer saltos del PC para tener cargado un programa en memoria e ir moviéndose en instrucciones que se encuentran en partes distintas de la memoria.

Estos objetivos impactan el diseño de RISC-V al hacer que el ISA sea eficiente, flexible y fácil de implementar en una amplia gama de aplicaciones y tecnologías, desde pequeños microcontroladores hasta grandes supercomputadoras. La modularidad y simplicidad permiten que el ISA sea adaptable y escalable, mientras que el enfoque en el costo y rendimiento asegura que los procesadores basados en RISC-V sean competitivos en términos de eficiencia.

4. **¿Cuáles son las reglas de nomenclatura de las instrucciones del set RISC-V? ¿Qué tipo de instrucciones tiene?**

   La nomenclatura de las instrucciones en RISC-V sigue un formato claro y conciso, lo cual facilita la programación y comprensión del conjunto de instrucciones:
   - Reglas de nomenclatura:
     - *Formato consistente*: todas las instrucciones son de 32 bits, lo que simplifica la decodificación y mejora la eficiencia.
     - *Uso de sufijos*: los sufijos permiten combinar instrucciones base con términos como immediate, unsigned, byte, word, etc… lo que permite la composición de instrucciones y brinda un set fácil de leer, deducir y recordar.
   - Tipos de instrucciones:
     - *R* (registros): para operaciones entre registros;
     - *I* (inmediatos): para inmediatos cortos y cargas;
     - *S* (stores) para almacenamientos;
     - *B* (branches) para bifurcaciones;
     - *U* (upper immediate) para inmediatos largos;
     - *J* (jump) para saltos incondicionales.

5. **¿Cuál es la ventaja de tener un registro de valor constante 0? ¿Cómo maneja las escrituras a este registro?**

   Permite tener instrucciones declarativas, simplifica muchas operaciones y reduce la necesidad de instrucciones adicionales, ya que no hay necesidad de reservar registros adicionales para almacenar el valor cero. Cualquier intento de escribir un valor en el registro x0 es ignorado, ya que este siempre mantiene el valor cero sin importar las operaciones de escritura que se le apliquen.
   La mayoría de las pseudoinstrucciones de RISC-V dependen de x0, por lo que apartar uno de los 32 registros para que esté alambrado a cero simplifica significativamente el set de instrucciones de RISC-V permitiendo muchas operaciones populares como pseudoinstrucciones.

6. **¿Cómo se resuelve la lógica de control (branching)?**

   RISC-V incluye un conjunto básico de instrucciones de branch condicional para comparar dos registros y tomar decisiones basadas en el resultado de la comparación, las cuales son beq, bne, blt, bge, bltu y bgeu. Dado que las instrucciones de RISC-V deben ser múltiplos de dos bytes, el modo de direccionamiento de branches multiplica el valor inmediato de 12 bits por 2, le extiende el signo y lo suma al PC.

7. **¿Cómo resuelve el overflow?**
   La forma de resolver el overflow no depende de la ISA, sino de quien diseñe el software, ya que en el conjunto de instrucciones de RISC-V no se generan excepciones de overflow para operaciones aritméticas. Esto simplifica la arquitectura y mejora el rendimiento del procesador, a la vez que permite a los programadores implementar la detección de overflow en software de ser necesario.

8. **¿Cómo resuelven los saltos incondicionales?**

   En RISC-V, los saltos incondicionales se manejan mediante la instrucción jal, jalr, j y jr. La instrucción jal cumple dos propósitos: llamadas a funciones (almacena la dirección de la siguiente instrucción en el registro destino ra), y saltos incondicionales (se utiliza el registro x0 como destino, ya que este no cambia. La dirección de destino se calcula multiplicando el inmediato de 20 bits por 2, extendiéndole el signo y sumándolo al PC). La instrucción jalr también se utiliza para dos propósitos: llamadas a funciones de direcciones dinámicas, y retornos de funciones usando ra como registro de origen y x0 como destino. Las otras dos son pseudoinstrucciones derivadas de estas dos primeras.

9. **¿Cómo se resuelve la multiplicación de números enteros?**

   La multiplicación como instrucción en sí no existe en el ISA base, sino que debe ser implementada en el software por el programador, por ejemplo, con una secuencia de sumas y corrimientos. De todos modos, existe una extensión de funciones que permite multiplicar con la instrucción mul.

10. **¿En qué consiste la convención de llamadas? ¿Qué registros se preservan? ¿Qué debe hacer la persona que escribe código que sigue esta convención con los registros que no se preservan?**

    Consiste en un contrato entre programadores sobre cómo se pasan los argumentos a las funciones, cómo se retornan los valores y cómo se manejan los registros durante las llamadas a funciones. A continuación se detalla esta convención:
    1. *Poner los argumentos*: Los primeros ocho argumentos se pasan a través de los registros a0-a7, y si hay más de ocho, los adicionales se pasan en la pila.
    2. *Saltar a la función*: Se utiliza la instrucción jal para saltar a la función, la cual guarda la dirección de retorno en el registro ra.
    3. *Reservar espacio en la pila*: La función llamada debe reservar espacio en la pila para su stack frame y almacenar los registros necesarios.
    4. *Realizar la tarea requerida*: La función ejecuta su tarea utilizando los registros y la pila según sea necesario.
    5. *Restaurar registros y liberar memoria*: Antes de retornar, la función debe restaurar los registros guardados y liberar el espacio en la pila.
    6. *Retornar al punto de origen*: la instrucción ret se utiliza para volver al punto desde donde se llamó la función, utilizando el valor almacenado en ra.

    Se preservan los saved registers (s0-s11), el frame pointer (fp), y el stack pointer (sp). Uno debe operar con los registros que no se preservan o, en caso de utilizar

registros que se preservan, garantizar que vuelvan al estado de antes de ser llamados al terminar la llamada a la función.

11. **¿Qué sucede si no hay suficientes registros como para pasar los parámetros de una función?**

    Si hay demasiados argumentos y variables en la función para caber en los registros, el prólogo de la función debe reservar espacio en el stack para la función, a esto se le llama frame. Luego que la función ha concluido, el epílogo debe restaurar el stack frame y regresar al punto de origen. Por convención, se suele mover el stack pointer en 16 posiciones.

12. **¿Qué son las pseudoinstrucciones? ¿Esto es microprogramación?**

    Las pseudoinstrucciones en RISC-V son instrucciones que no se implementan directamente en el hardware, sino que el ensamblador las traduce a una o más instrucciones reales de RISC-V. Estas pseudoinstrucciones hacen que el código ensamblador sea más fácil de escribir y entender, proporcionando una sintaxis más intuitiva para operaciones comunes que podrían requerir múltiples instrucciones reales. Por otra parte, la microprogramación es una técnica de diseño de hardware en la que las instrucciones de alto nivel del ISA se implementan mediante una secuencia de instrucciones más simples en un nivel microarquitectural. En contraste, las pseudoinstrucciones en RISC-V son una característica del ensamblador y no del hardware, y por lo tanto no son lo mismo que microprogramación.

13. **¿Qué son las directivas de ensamblador?**

    Son comandos que no se traducen a código máquina y sirven al ensamblador para indicarle en qué parte de la memoria poner el código y datos en el mapa de memoria, o sea, cómo segmentar la información de nuestra arquitectura.

14. **¿Qué significa Position Independent Code? ¿Qué ventaja tiene sobre el código dependiente de posición?**

    Significa código independiente de su posición, y nos permite realizar saltos relativos al PC, es decir, modificarlo a diversas direcciones para ejecutar instrucciones que no son necesariamente contiguas en la memoria donde están cargadas. Una ventaja visible es la facilidad para realizar ciclos (iteraciones) y funciones recursivas.

15. **¿A qué se llama el heap? ¿A qué se conoce como un heap overflow?**

    No entra.

16. **Describa las similitudes y diferencias entre las instrucciones de formatos B y S. Ídem entre las instrucciones J y U.**

    Similitudes entre B y S:
    - Ambos formatos tienen campos para dos registros fuente.

- Ambos usan un campo inmediato de 12 bits.
- Ambos usan el campo funct3 para especificar la operación exacta.

Diferencias entre B y S:

- Propósito (el formato B es para bifurcaciones condicionales, mientras que S para almacenar datos en memoria).
- Uso del inmediato (en el formato B se utiliza para calcular la dirección de salto, mientras que en S se utiliza para calcular la dirección de memoria donde se almacenarán los datos).
- Distribución del inmediato (en el formato B se distribuye en varias partes dentro de la instrucción, mientras que en S se divide en 2 partes de 5 y 7 bits.

Similitudes entre J y U:

- Ambos formatos tienen un campo inmediato de 20 bits.
- Ambos usan un registro destino.

Diferencias entre J y U:

- Propósito (el formato J es para saltos incondicionales, mientras que U es para cargar inmediatos altos).
- Uso del inmediato (en el formato J se utiliza para calcular una dirección de salto, mientras que en U se utiliza para cargar un valor en los bits altos de un registro).
- Distribución del inmediato (el orden en que se cargan los bits de cada inmediato es distinto).

17. Dados dos registros, mostrar cómo intercambiarlos sin intervención de un tercero.

```
xor x1 x1 x2
xor x2 x1 x2
xor x1 x1 x2
```

18. Sabiendo que a1 = 0xFFFFFFFF, ¿cuánto queda almacenado en a2 luego de realizar: andi a2, a1, 0xF00?

0xFFFFFF00

19. ¿En qué posición dentro de la instrucción se encuentran los bits de los registros destino y origen? ¿Depende del tipo de instrucción o de la instrucción en sí? ¿Por qué fue diseñado así el formato de instrucción?

El registro destino se encuentra siempre en las posiciones 11 a 7 de la tira de bits. El registro origen de 19 a 15, y en caso de ser necesario otro registro origen, va ser 24 a 19. Esto es siempre así, no depende del tipo de instrucción (más allá que hay instrucciones que no toman registro destino o fuente), y fue pensado así para ahorrar tiempo en la decodificación, así se puede acceder a estos antes de la decodificación.

20. ¿Qué problemas puede ocasionar utilizar un registro de propósito general para el PC?

Utilizar un registro de propósito general para el PC en una arquitectura como RISC-V presenta varios problemas graves que afectan tanto el diseño del hardware como la estabilidad y seguridad del software ya que, por ejemplo, la mayoría de programas y sistemas operativos están diseñados con la suposición de que el PC se comporta de una manera específica y predecible.

21. ¿Cómo se hace una lectura del PC?

En el datapath se pasa el valor a la memoria de instrucciones, la cual devuelve la instrucción codificada en binario y se procede a ejecutar la instrucción con diferentes elementos del diseño. En simultáneo, se calculan los posibles valores del PCNext, ya sea la siguiente instrucción (PC+4) o algún salto (PC+offset), que será elegido mediante una señal de control con un multiplexor.

22. RISC-V lee los datos little-endian, ¿qué significa? Dé un ejemplo. ¿Cuándo es importante?

Significa que los bytes de menor peso se almacenan en la dirección más baja de memoria, y los de mayor peso en la parte más alta. Por ejemplo:

```
.data
ejemplo1: .byte 0x00 0x01 0x2 0x03
ejemplo2: .word 0x03020100
```

En este caso, ejemplo1 y ejemplo2 son formas equivalentes de definir una palabra en memoria. Esto es importante cuando queremos acceder a un mismo dato en modo word y byte por ejemplo.

23. RISC-V maneja el principio de "simplicidad", en relación a esto responda:

a) **¿Acceder a un operando en registro es más rápido que buscar el operando en memoria?**
Sí, y de hecho una gran ventaja de RISC-V es la cantidad de registros que dispone.

b) **A partir del inciso anterior, ¿cómo cree que impacta al rendimiento del programa y a la arquitectura la cantidad de registros disponibles?**
A mayor cantidad de registros, se requieren menos accesos a memoria y, por lo tanto, mejora el rendimiento en relación al tiempo.

24. ¿Cómo resuelve BGT (branch greater than) con RISC-V32?

En RISC-V, bgt es una pseudoinstrucción, ya que se resuelve invirtiendo el orden de los registros a comparar.

```
bgt x1, x2 label = blt x2, x1 label
```

pues x1>x2 si y sólo si x2<x1

25. Ensamblar el siguiente código:

```
add a0, a1, a6
bltz x1, 0xABC
```

Los pasos son:

1. Resolver las pseudoinstrucciones.
2. Resolver las etiquetas.
3. Codificar las instrucciones de ensamblador a binario.

00000001000001100000010110110011
00101010000100000100111011100011