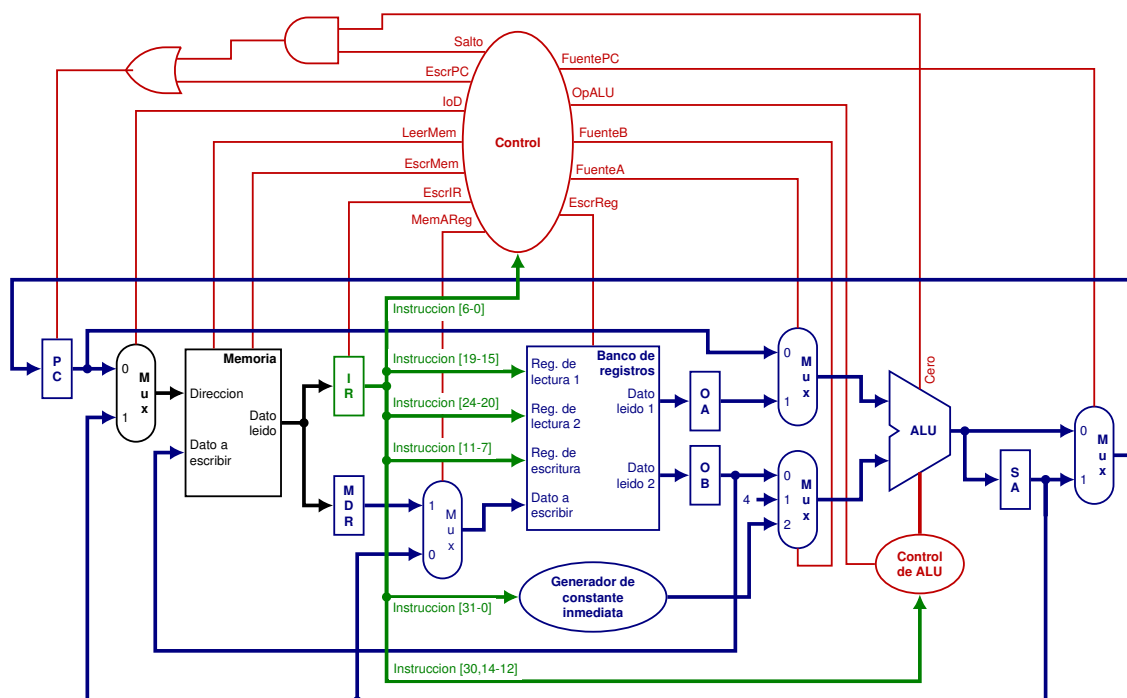




Soluciones Completas

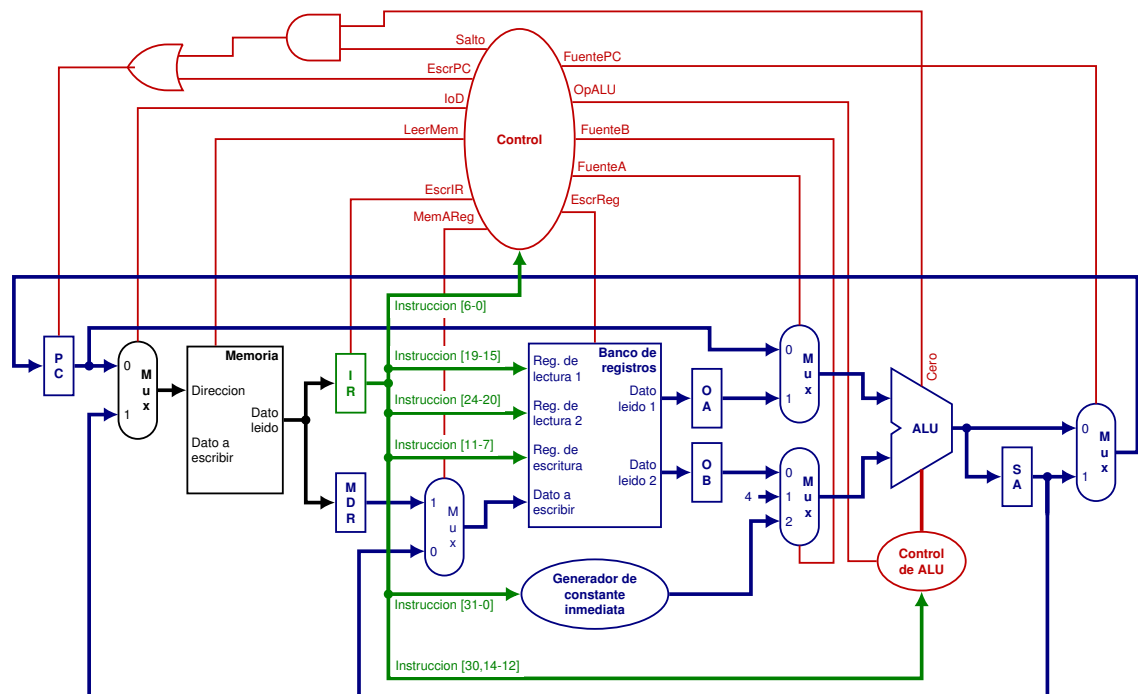
5) a) El camino de datos (datapath) es el conjunto de componentes de hardware que realizan las operaciones del procesador, como la memoria, la ALU, el banco de registros y los buses de conexión. Las instrucciones Store (S-type) son responsables de escribir datos desde un registro (rs2) a una dirección de memoria calculada.

En un datapath RISC-V estándar, esto implica la activación de varios componentes:

- ALU: Calcula la dirección de memoria sumando el valor de un registro base (rs1) y un desplazamiento inmediato de 12 bits (Imm). La salida de la ALU se convierte en la dirección de la memoria de datos.
- Banco de Registros: La instrucción Store utiliza el valor del registro rs2 como el dato que se va a escribir en la memoria.
- Memoria de Datos: Recibe la dirección desde la ALU y el dato desde rs2, y realiza la operación de escritura.
- Señal de Control MemWrite: Esta es la señal de control crítica que habilita la operación de escritura en la memoria de datos.

La eliminación de las instrucciones Store en el SM2025 haría que todos estos elementos, en su función de escritura, se volvieran innecesarios.

- Eliminación del Bus de Escritura a Memoria: El bus dedicado que transporta el dato del registro rs2 hacia el puerto Data In de la memoria de datos ya no sería necesario. Este bus físico, que va desde la salida del banco de registros hasta el puerto de entrada de la memoria de datos, puede ser eliminado por completo. Esta simplificación reduce el número de interconexiones y el enrutamiento del chip.
- Eliminación de la Señal de Control MemWrite: La señal que activa el pin de escritura de la memoria de datos (Write Enable) se vuelve obsoleta. La lógica combinacional en la unidad de control responsable de generar esta señal puede ser eliminada, lo que simplifica directamente el diseño del circuito de control.



b) La máquina de estado finita (FSM) es la encargada de generar las señales de control que dirigen las operaciones del datapath en cada ciclo. Su lógica depende directamente del conjunto de instrucciones del procesador.

Para cada opcode y campo de función de una instrucción, la FSM produce una combinación única de señales de control. La eliminación de las instrucciones de tipo Store del ISA del SM2025 simplifica la

lógica de la FSM de la siguiente manera:

- Reducción de la Lógica de Decodificación: La FSM ya no necesita detectar los opcodes de las instrucciones SB, SH y SW. En un diseño de control cableado, esto significa que la lógica combinatorial que detecta estos opcodes y activa la señal MemWrite se puede eliminar por completo.
- Simplificación de la Tabla de Control: Eliminamos los estados relacionados con los comandos Store. La tabla de control se volvería más pequeña, con menos filas y columnas, lo que a su vez se traduce en un menor número de puertas lógicas y un circuito más simple y fiable. Esta simplificación de la unidad de control reduce la complejidad del circuito en general, lo que potencialmente puede mejorar la fiabilidad y reducir el costo de diseño y fabricación. La simplificación del ISA en un procesador RISC es un beneficio tangible a nivel de hardware.

- c) El principio fundamental de una arquitectura load-store, como RISC-V, es que las operaciones de procesamiento se realizan exclusivamente sobre los registros del procesador. El acceso a la memoria, ya sea para leer o escribir datos, se maneja de manera separada y explícita mediante instrucciones dedicadas (Load y Store).

La pregunta de si las instrucciones Store pueden ser emuladas por una combinación de otras instrucciones tiene una respuesta definitiva: no, no es posible. La justificación para esta afirmación radica en la arquitectura subyacente del camino de datos y la unidad de control, por la inexistencia de un camino físico que lo permita.

Ninguna de las instrucciones de tipo R, I, J —como las aritméticas (add, sub), lógicas (and, or), o de control de flujo (beq, jal)— activa el bus que envía datos del banco de registros a la memoria de datos, ni activa la señal de control MemWrite. La función de una instrucción está estrictamente definida por su opcode y su lógica de decodificación. Por lo tanto, no existe una combinación de instrucciones de cálculo o control de flujo que, de forma incidental, pueda tener el efecto secundario de escribir en la memoria.

La imposibilidad de emular la funcionalidad de escritura en la memoria transforma el microprocesador SM2025 de una máquina de propósito general en una arquitectura de "solo lectura" de datos. Esto significa que cualquier cálculo realizado por el procesador sobre los datos leídos de la memoria o presentes en los registros no puede ser persistido para su uso posterior.

Esta limitación inherente hace que el SM2025 sea inviable para la mayoría de las aplicaciones prácticas, como la ejecución de un sistema operativo, la manipulación de archivos o la comunicación con periféricos que requieren memoria de datos para la transferencia de información. El diseño solo sería útil en un nicho muy especializado, como el procesamiento de señales en tiempo real donde los datos se consumen, procesan y transmiten inmediatamente, sin necesidad de almacenamiento intermedio.

- 6) a) Consecuencias: La ALU recibe la fuente A incorrecta. Normalmente, ALUSrcA=0 en Fetch/Branch (usa PC para PC+4 o PC+imm), y =1 en Execute/MemAdr (usa rs1). Invertido: en estados que necesitan PC (e.g., incrementar PC), usa rs1; en cálculos con rs1, usa PC. Esto causa que el procesador falle en todas las instrucciones llevando a ejecución caótica:

- Cálculos erróneos en ALU (e.g., add usa PC + op2 en vez de rs1 + op2). PC no se incrementa correctamente, causando loops o saltos erróneos. Registros se corrompen si PC se usa como operando. Instrucciones afectadas:
- Aritméticas/Logic (R-type e I-type como ADD, ADDI, AND): En Execute, ALUSrcA debería ser 1 (rs1), pero invertido a 0 (PC). Resultado: ALU computa $PC + rs2/imm \rightarrow rd$, no $rs1 + rs2/imm$. Afecta todos los cálculos registro-registro/inmediato.
- Loads/Stores (I-type load, S-type): En MemAdr, ALUSrcA=1 (rs1), invertido a 0 (PC). Dirección = PC + imm, no rs1 + imm. Loads leen de dirección errónea; stores escriben en errónea (corrupción de memoria).
- Branches y Jumps (SB-type como BEQ): ALUSrcA=0 (PC), invertido a 1 (rs1). Target = $rs1 + imm \ll 1$, no $PC + imm \ll 1$. Branches van a direcciones absolutas basadas en rs1, no relativas, causando saltos impredecibles.

- b) Si la línea de control IoD, que selecciona la fuente de la dirección para la memoria (0 para el PC en fetch de instrucciones, y 1 para el resultado de la ALU en accesos de datos), queda fijada en 0 de manera permanente debido a un fallo en la unidad de control de un procesador RISC-V, el impacto en la operación del procesador sería el siguiente:

- Fetch de instrucciones: Funciona correctamente. Durante la etapa de fetch, la dirección de memoria se toma del Program Counter (PC), lo cual es el comportamiento esperado cuando IoD = 0. Por lo tanto, el procesador puede continuar extrayendo instrucciones secuenciales sin problemas iniciales.
- Accesos a memoria para datos (instrucciones de load y store): Fallan por completo en su propósito original. El multiplexor de direcciones a memoria siempre seleccionará el PC en lugar del resultado

computado por la ALU (que sería la dirección efectiva para loads/stores, como en lw o sw). Esto implica que cualquier acceso a datos usará la dirección actual del PC, que apunta a la zona de código (memoria de instrucciones), en lugar de la zona de datos.

Consecuencias específicas por tipo de instrucción:

- Instrucciones de carga (loads, e.g., lw, lh, lb): En lugar de leer el valor de memoria en la dirección calculada (base + offset), se lee el contenido de la memoria en la posición apuntada por el PC. El registro destino recibe un valor incorrecto, posiblemente un opcode o parte de una instrucción de código, en vez del dato esperado. Esto puede causar corrupción en los cálculos posteriores que dependan de ese registro.
- Instrucciones de almacenamiento (stores, e.g., sw, sh, sb): En lugar de escribir el valor en la dirección calculada, se escribe directamente en la posición del PC. Sobrescribe parte del código del programa en ejecución. Esto puede corromper instrucciones futuras, llevando a un comportamiento impredecible, como ejecuciones erróneas, bucles infinitos o crashes del procesador (e.g., intentos de ejecutar datos no válidos como opcodes).
- Instrucciones de cómputo (e.g., add, sub, and, operaciones aritméticas/lógicas): Funcionan normalmente, ya que no dependen del acceso a memoria de datos.
- Instrucciones de salto y branches (e.g., beq, jal, jr): También funcionan, siempre que no involucren accesos indirectos a memoria. El PC se actualiza correctamente, pero si el programa intenta saltar a código corrupto (por un store previo), fallará.

En resumen, el procesador puede ejecutar programas que no usen loads/stores (e.g., solo cálculos en registros), pero cualquier programa con accesos a memoria de datos fallará, potencialmente corrompiendo el código y deteniendo la ejecución útil.

c) La línea de control PCSource toma el valor 0 para instrucciones secuenciales ($PC=PC+4$), mientras que PCSource toma el valor 1 para branches o jumps. Si esa línea de control quedara fija en 0, no funcionarían branches ni jumps ya que PC siempre se actualizaría a $PC+4$, mientras las demás instrucciones funcionarían sin problemas.

- Instrucciones afectadas: Branches y Jumps, ya que la condición de salto se computa, pero PC no salta (queda $PC+4$). Branches actúan como NOPs; loops condicionales fallan (ejecución infinita o skip). No saltan; $PC=PC+4$ en vez de target. Funciones no se llaman, returns fallan.
- Instrucciones no afectadas: Todas las demás (instrucciones aritméticas, loads y stores), funcionan sin problemas.

Por lo tanto, los programas lineales, sin control de flujo (saltos) funcionan, pero cualquier programa con saltos (condicionales o incondicionales) y llamadas a subrutinas o funciones, falla. El procesador ejecuta en línea recta, lo que sólo resulta útil en casos muy particulares.