UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería
CC3084 – Construcción de Compiladores
Sección 10
Ing. Bydkar Pojoy

Generación de CI

Samuel Argueta - 211024
Alejandro Martinez - 21430

GUATEMALA, 14 de octubre de 2024

https://msyksphinz-self.github.io/riscv-isadoc/html/rv64i.html

RV64I

RV64I es la variante de 64 bits del conjunto de instrucciones base de RISC-V, una arquitectura abierta, modular y escalable que permite a desarrolladores adaptar y personalizar el hardware sin restricciones de licencia. **RV64I** amplía la versión de 32 bits (RV32I) para trabajar con registros, direcciones y operaciones de 64 bits, manteniendo retrocompatibilidad con las instrucciones de 32 bits.

Componentes principales

1. **Instrucciones soportadas en RV64I**

   La ISA define instrucciones básicas para realizar tareas fundamentales:

   - **Aritméticas:** Suma (ADD), resta (SUB), multiplicación (MUL).
     i. Estas instrucciones realizan operaciones matemáticas básicas, similares a las de muchos lenguajes de programación:
        1. **ADD** (+): Suma dos valores.
        2. **SUB** (-): Resta un valor del otro.
        3. **MUL** (*): Multiplica dos valores.
        4. **DIV** (/): Realiza división entre dos valores.
        5. **MOD** (%): Calcula el resto de una división.
   - **Lógicas:** AND, OR, XOR, desplazamiento (SLL, SRL, SRA).
     i. Estas instrucciones realizan operaciones lógicas, con un comportamiento parecido al de expresiones booleanas en lenguajes como C o Java:
        1. **AND** (&&): Conjunción lógica, devuelve verdadero si ambos operandos son verdaderos.
        2. **NOT** (!): Negación lógica, invierte el valor booleano.
        3. **OR** (||): Disyunción lógica, devuelve verdadero si al menos uno de los operandos es verdadero.
   - **Control de flujo:** Branches (BEQ, BNE), jumps (JAL, JALR).
     i. Estos operadores comparan dos valores y devuelven un resultado booleano:
        1. **NEQ** (!=): Verifica si dos valores son distintos.
        2. **LEQ** (<=): Verifica si un valor es menor o igual al otro.
        3. **GEQ** (>=): Verifica si un valor es mayor o igual al otro.
        4. **EQ** (==): Verifica si dos valores son iguales.
        5. **LT** (<): Verifica si un valor es menor que otro.
        6. **GT** (>): Verifica si un valor es mayor que otro.

- **Instrucciones específicas de control y gestión**
    i. Estas instrucciones se usan para gestionar **llamadas a funciones**, **control del flujo del programa** y **operaciones con memoria**:
        1. **ALLOCATE:** Reserva espacio para un array en la **ARRAY_STACK**.
        2. **SYSCALL:** Realiza una llamada al sistema para imprimir valores.
        3. **RETURN:** Extrae un valor de la **CALL_STACK** y retorna a la posición donde se hizo la llamada.
        4. **GO_TO:** Salta a una etiqueta específica y saca parámetros de la **PARAM_STACK**.
        5. **PRINT:** Imprime el valor almacenado en una dirección específica.
        6. **CALL:** Llama a una función en una etiqueta específica, saca parámetros de la **PARAM_STACK** y guarda la posición actual en la **CALL_STACK** para poder regresar luego.
        7. **PUSH:** Inserta un valor en la **PARAM_STACK**.
        8. **MOV:** Mueve un valor dentro del **ARRAY_STACK** utilizando desplazamientos (offsets).
        9. **IF:** Compara dos valores, útil para evaluaciones condicionales.
- **Uso de las pilas (Stacks)**
    i. RISC-V y este conjunto extendido gestionan múltiples pilas internas para mantener el control sobre **funciones, parámetros y memoria**.
        1. ARRAY_STACK: Almacena todos los elementos de los arrays. Es útil para gestionar estructuras de datos que requieren acceso secuencial o por índices.
        2. PARAM_STACK: Contiene los parámetros que se pasan entre funciones, especialmente útil para funciones recursivas.
        3. CALL_STACK: Almacena las direcciones de retorno cuando una función realiza una llamada a otra. Facilita la ejecución de funciones anidadas o recursivas.
- **Variables internas importantes**
    i. IT_ARRAY_PTR: Registra la posición y el desplazamiento (offset) para los elementos dentro de los arrays. Esto es útil para gestionar acceso a datos en estructuras como matrices multidimensionales.

2. Supuestos Considerados Durante la Traducción

- **Uso de Pilas para Parámetros:**
  Los parámetros se colocan en la **PARAM_STACK** antes de hacer la llamada a la función.
- **Gestión del Flujo con CALL y RETURN:**
  La función se invoca con **CALL**, y el flujo se restablece con **RETURN** usando la **CALL_STACK**.
- **Sin optimización temprana de parámetros:**
  Los parámetros se suman sin realizar verificaciones adicionales, asumiendo que el tipo de dato es correcto.
- **Control del Flujo con PRINT:**
  El valor devuelto por sum se imprime usando **PRINT**, que realiza una llamada al sistema.

3. Decisiones de Diseño del Lenguaje Intermedio

- **Simplicidad en la implementación:**
  Se evitó el uso de estructuras complejas para mantener la claridad en el flujo del programa.
- **Uso explícito de pilas:**
  Las pilas gestionan el flujo de parámetros y funciones para facilitar la recursividad y las llamadas anidadas.
- **Optimización a nivel de diseño:**
  La instrucción **MOV** con offsets permite trabajar eficientemente con estructuras de datos en memoria.
- **Modularidad y reutilización:**
  El uso de **CALL** y **RETURN** garantiza que el código pueda ser fácilmente modularizado.