Métricas sobre un procesador RISCV

Carlos Steve Alvarado Mendez
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Correo: stevealv@estudiantec.cr

Axel Bruno Flores Lara
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Correo: axeel@estudiantec.cr

Kendall Marín Muñoz
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Correo: kendallmarin@estudiantec.cr

Carlos Rodriguez Segura
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
Correo: c.rodriguez@estudiantec.cr

***Resumen*—Este taller examina cómo funciona un procesador RISC-V usando un simulador llamado Ripes 1 y programación en lenguaje ensamblador. Queremos entender cómo accede a la memoria, salta de una parte del programa a otra y realiza operaciones con registros, todo relacionado con teorías ya conocidas. Primero, miramos cómo accede a la memoria usando pruebas que hemos creado, para ver con qué frecuencia y de qué manera lo hace. Luego, evaluamos cómo los saltos en el programa afectan al flujo del proceso y su eficiencia. Después, nos enfocamos en las operaciones que realiza con registros y qué tan importantes son para la eficiencia general. Cada paso lo hacemos usando ejemplos de código y explicaciones detalladas. También analizamos la información obtenida de diferentes tipos de procesadores para entender cómo varía el rendimiento. Si no podemos solucionar problemas con el hardware, buscamos soluciones en el software y hablamos sobre cómo afectan al rendimiento. Finalmente, comparamos cómo funciona nuestro procesador en una tabla con lo que esperábamos que hiciera. Con esta investigación, esperamos entender mejor cómo se diseñan y se usan los procesadores, lo que nos ayuda a entender mejor cómo funcionan las computadoras en general.**

1. Introducción

En el estudio de la arquitectura de computadores, es fundamental entender cómo diferentes configuraciones de procesadores afectan el rendimiento de programas. Esta tarea tiene como objetivo principal comparar dos tipos de procesadores, un procesador uniciclo y un procesador con pipeline de 5 etapas, utilizando el simulador de procesador RISC-V: Ripes. Para llevar a cabo esta comparación, se programaron tres ejercicios en ensamblador RISC-V que simulan distintas cargas de trabajo: accesos a memoria, saltos, y operaciones con registros.

Para cumplir con el objetivo de la investigación, se desarrollan tres programas en el lenguaje ensamblador para cuantificar diferentes aspectos de cada procesador, por ejemplo: el número de ciclos, cantidad de instrucciones, clock rate, CPI (Ciclos por Instrucción) y PCI (instrucciones ejecutadas por ciclo). Además, se creara una tabla 3x4 comparando el funcionamiento del sistema según los benchmarks obtenidos.

En el desarrollo de este trabajo, se analizarán y discutirán los datos obtenidos de la ejecución de los programas en ambos tipos de procesadores. La discusión se centrará en las principales diferencias de estos procesadores, considerando ventajas y desventajas de cada procesador. Se explorará cómo la estructura del pipelining influye en la eficiencia de ejecución comparada con un ciclo único y se evaluarán las implicaciones de estos resultados en términos de accesos a memoria, saltos y operaciones con registros. Además, se abordará cómo las transformaciones de código en software pueden mitigar las limitaciones del hardware en cada configuración, proporcionando una visión comprensiva sobre la optimización del rendimiento de los microprocesadores.

2. Desarrollo

En la teoría, los procesadores con pipeline pueden procesar múltiples instrucciones en paralelo, lo que mejora el IPC. Sin embargo, esta ventaja puede verse limitada por los riesgos de datos y control, que son manejados mediante técnicas de forwarding, stalls o transformaciones de código, esto genera una desventaja de mayor complejidad en la gestión de riesgos de datos y control, lo que puede llevar a ciclos adicionales (NOPs) y mayor CPI. El procesador de ciclo único, aunque más simple, tiene un rendimiento limitado debido a su incapacidad para ejecutar instrucciones en paralelo, debiendo completar todos los pasos de cada instrucción antes de que la siguiente instrucción comience [1].

Para demostrar cómo la arquitectura del procesador y las técnicas de optimización pueden influir significativamente en el rendimiento de un sistema

se desarrollaron tres programas específicos para investigar accesos a memoria, saltos y operaciones con registros.

Para el programa que demuestra los accesos a memoria de un sistema se desarrolló el código presente en la figura 1, el cual define un arreglo de 5 enteros en la sección de datos y en la sección de texto carga la dirección base del arreglo en un registro, realiza cinco accesos a memoria para cargar los valores individuales del arreglo en registros separados, realiza una operación de suma entre los dos primeros elementos del arreglo y almacena el resultado de la suma de vuelta en la primera posición del arreglo.

```
# Programa que demuestra accesos a memoria

.data
array: .word 1, 2, 3, 4, 5  # Define un arreglo de 5 enteros

.text
.global _start

_start:
    la t0, array          # Carga la dirección base del array en el registro t0
    lw t1, 0(t0)          # Carga el primer elemento del array en t1 (acceso a memoria)
    lw t2, 4(t0)          # Carga el segundo elemento del array en t2 (acceso a memoria)
    lw t3, 8(t0)          # Carga el tercer elemento del array en t3 (acceso a memoria)
    lw t4, 12(t0)         # Carga el cuarto elemento del array en t4 (acceso a memoria)
    lw t5, 16(t0)         # Carga el quinto elemento del array en t5 (acceso a memoria)
    add t6, t1, t2        # Suma de registros (operación entre registros)
    sw t6, 0(t0)          # Almacena el resultado en la primera posición del array (acceso a memoria)

# Total de accesos a memoria: 6 (5 lecturas + 1 escritura)
```

Figura 1. Programa de accesos a memoria del sistema.

Para el programa que demuestra saltos dentro del programa se desarrollo el código presente en la figura 2, el cual implementa un contador que cuenta de 1 a 10 utilizando saltos en ensamblador. En la sección de texto se declara el punto de entrada del programa con la etiqueta start, se carga el valor 10 en el registro t0 para establecer el límite del contador, se carga el valor 0 en el registro t1, que actuará como el contador inicial.

Dentro de la etiqueta loop se verifica si el valor en t1 es igual al valor en t0 utilizando una instrucción de salto condicional beq (Branch Equal). Si son iguales, el programa salta a la etiqueta endloop, se incrementa el valor en t1 en 1 utilizando la instrucción addi (Add Immediate), lo que avanza el contador en 1 en cada iteración, se realiza un salto incondicional hacia la etiqueta loop utilizando la instrucción j (Jump), lo que asegura que el programa continúe ejecutándose en un bucle hasta que se cumpla la condición de salida, cuando se alcanza el límite del contador, el programa salta a la etiqueta endloop, donde se encuentra una instrucción nop (No Operation), lo que indica el final del programa.

```
# Programa que demuestra saltos
# Contador de 1 a 10 por medio de saltos

.text
.global _start

_start:
    li t0, 10             # Carga el valor 10 en t0
    li t1, 0              # Carga el valor 0 en t1
loop:
    beq t1, t0, end_loop  # Si t1 == t0, salta a end_loop (salto condicional)
    addi t1, t1, 1        # Incrementa t1 en 1 (operación entre registros)
    j loop                # Salta incondicionalmente a loop (salto incondicional)
end_loop:
    nop                   # No operación

# Total de saltos: 2 (1 condicional + 1 incondicional)
```

Figura 2. Programa saltos en el sistema.

Finalmente, para el programa que demuestra el uso de operaciones entre registros, se diseñó el código de la figura 3, el cuál realiza diversas operaciones entre registros en un entorno de ensamblador. En la sección de texto, se declara el punto de entrada del programa con la etiqueta start. Posteriormente, se carga el valor 5 en el registro t0 y el valor 10 en el registro t1.

Dentro del programa, se llevan a cabo las siguientes operaciones entre registros:

Se suma el contenido de los registros t0 y t1, y el resultado se almacena en el registro t2. A continuación, se resta el contenido de t0 de t1, y el resultado se guarda en el registro t3. Luego, se realiza la operación AND bit a bit entre los contenidos de t0 y t1, y el resultado se almacena en t4. Después, se ejecuta la operación OR bit a bit entre los valores de t0 y t1, y el resultado se guarda en t5. Por último, se realiza la operación XOR bit a bit entre t0 y t1, y el resultado se almacena en t6.

En resumen, el programa ejecuta un total de 5 operaciones entre registros para realizar las diversas operaciones aritméticas y lógicas especificadas.

```
# Operaciones entre registros
.text
.global _start
_start:
    li t0, 5              # Carga el valor 5 en t0
    li t1, 10             # Carga el valor 10 en t1
    add t2, t0, t1        # Suma t0 y t1, guarda el resultado en t2
    sub t3, t1, t0        # Resta t0 de t1, guarda el resultado en t3
    and t4, t0, t1        # AND bit a bit entre t0 y t1, guarda el resultado en t4
    or t5, t0, t1         # OR bit a bit entre t0 y t1, guarda el resultado en t5
    xor t6, t0, t1        # XOR bit a bit entre t0 y t1, guarda el resultado en t6

# Total de operaciones entre registros: 5
```

Figura 3. Programa de operaciones entre registros.

Para realizar el análisis del comportamiento de los programas en ensamblador RISC-V utilizando el simulador Ripes, se siguieron los siguientes pasos para cada programa:

Configuración de Procesadores

Se utilizaron dos configuraciones de procesadores en Ripes:

- Procesador de ciclo único (Uniciclo)
- Procesador con pipeline de 5 etapas

Cada programa fue ejecutado en ambas configuraciones y se registraron las estadísticas de ejecución proporcionadas por el simulador. Las métricas consideradas incluyeron el número de ciclos, el número de instrucciones, el CPI (Ciclos por Instrucción), el IPC (Instrucciones por Ciclo) y la frecuencia del reloj (Clock rate).

Se elaboró la tabla 1 para resumir y analizar el comportamiento de cada programa (PG) en ambas configuraciones de procesador.

Tabla 1. COMPARACIÓN DE PROCESADORES

PG	Procesador de ciclo único					Procesador con Pipelining de profundidad 5				
	Ciclos	Instrucciones	CPI	IPC	Clock rate	Ciclos	Instrucciones	CPI	IPC	Clock rate
1	9	9	1	1	10.64	13	9	1.44	0.692	9.09
2	34	34	1	1	10.87	60	34	1.76	0.567	9.01
3	7	7	1	1	10.13	11	7	1.57	0.636	9.01

Cuantificaciones:

- PG 1: 6 accesos a memoria, 1 operación entre Registros
- PG 2: 2 saltos, 1 operación entre registros, 10 iteraciones. 0 Memoria
- PG 3: 5 operaciones entre registros (todas)

3. Conclusión

Este taller comparó el rendimiento de un procesador uniciclo y un procesador con pipeline de 5 etapas utilizando el simulador RISC-V Ripes y programas en lenguaje ensamblador. Los resultados mostraron que el procesador uniciclo es más eficiente para programas simples y de baja complejidad debido a su menor latencia y ejecución directa de instrucciones.

El procesador con pipeline de 5 etapas, aunque más adecuado para tareas complejas y paralelizadas, introduce sobrecarga y latencias adicionales en programas sencillos, lo que disminuye su eficiencia en estos casos. Manejar saltos y accesos a memoria es más directo y rápido en el procesador uniciclo, mientras que el pipeline requiere más gestión y puede sufrir retrasos.

La profundidad del pipeline también puede influir en el rendimiento. Aunque el pipelining puede mejorar el throughput, una profundidad excesiva puede aumentar la posibilidad de conflictos y retrasos, lo que puede afectar negativamente al IPC y, en última instancia, al rendimiento general del procesador

En resumen, el procesador uniciclo demostró ser superior en este estudio ya que se crearon programas con pocas instrucciones y menor complejidad, mientras que el pipeline de 5 etapas es más beneficioso para aplicaciones más complejas y con mayor paralelismo, aunque a costa de una mayor complejidad y latencia.

Referencias

[1] David A. Patterson & John L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann, 2014.