UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERIA
ARQUITECTURA Y DISEÑO DE SISTEMAS DIGITALES
IELE 3222 2120-2

DESIGN PROBLEM Módulo-02: Implementación de algoritmo multiplicador en RISC-V. (Grupos de 2 o 3 integrantes máximo) – (10 %)

1. MOTIVACIÓN PARA LA REALIZACIÓN DEL PROYECTO

En el curso hemos identificado qué es un conjunto de instrucciones dentro del contexto de una microarquitectura o un procesador, cómo este se utiliza para traducir instrucciones que vienen desde el nivel de software y al mismo tiempo cómo controla los componentes hardware del procesador para la ejecución de tareas. En este problema se propone la utilización de este conocimiento para generar un programa que pueda cumplir ciertos requerimientos y que los pueda completar de forma eficiente. Para realizar este trabajo se utilizará el simulador RIPES v2.2.2 (https://github.com/mortbopet/Ripes/releases) que permite compilar y editar código de ensamble en la especificación base de RISC-V así como poder observar la ejecución de cada instrucción en los componentes de la arquitectura, trabajar con memoria caché y con puertos de entrada-salida.

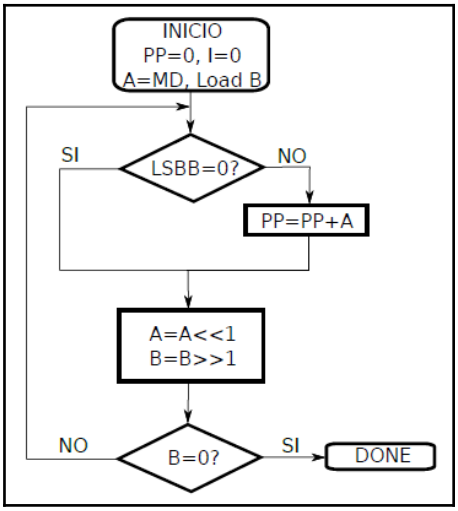
2. CONTEXTO DEL PROYECTO

En este problema se propone implementar el algoritmo de multiplicación basado en desplazamientos para dar solución a una operación matricial sencilla (nxn) * (nx1) = (nx1) pero que es de gran utilidad para una amplia cantidad de aplicaciones.

$$\begin{bmatrix} w & x \\ y & z \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} w * a + x * b \\ y * a + z * b \end{bmatrix}$$

(Ejemplo con n=2. Para obtener el resultado de esta operación se deben realizar 4 operaciones de multiplicación y 2 operaciones de suma entre 6 operandos)

El algoritmo de multiplicación a implementar es el mismo que se desarrolló en las primeras clases del curso de **Arquitectura y Diseño de Sistema Digitales** el cual presenta un sencillo macro-algoritmo descrito en los siguientes pasos.



Donde A es el multiplicando, B es el multiplicador y PP es el acumulador donde se calcula la respuesta.

El objetivo principal de este problema, aparte de familiarizarse con el lenguaje ensamblador de RISC-V para ejecutar operaciones y algoritmos sencillos, es poder entender con mayor detalle la ejecución de dichas operaciones en la arquitectura de un procesador "real". De esta manera, se espera que el estudiante comprenda cómo se ejecutan las distintas instrucciones en el procesador, cómo es el flujo de información y control de los elementos hardware que lo componen y cómo afecta el *performance* de la arquitectura la ejecución de diferentes procesos. Para esto, se van a utilizar cálculos elementales de desempeño que relacionan 3 principales variables: clock rate, instruction count y CPI (IPC). Estos cálculos permiten obtener el desempeño general del algoritmo y el procesador con información básica que proporciona el mismo simulador RIPES. Así, los indicadores que deben calcularse son los siguientes:

GENERAL PERFORMANCE

$$Performance = \frac{1}{execution\ time(s/\ program)}$$

CPU PERFORMANCE

$$execution\ time(s/\ program) = \frac{clock\ cycles(count/\ program)}{clock\ rate(1/s)}$$

INSTRUCTION PERFORMANCE

$$clock\ cycles(count/\ program) = instructions\ count\ *\ clock\ cycles\ per\ instruction$$

Adicionalmente, el simulador RIPES presenta dos métricas importantes para los cálculos anteriores, que son:

CPI (CLOCK CYCLES PER INSTRUCTION)

$$execution\ time(s/program) = \frac{instruction\ count\ *\ CPI}{clock\ rate(1/s)}$$

IPC (INSTRUCTIONS PER CLOCK CYCLE)

$$IPC = \frac{1}{CPI}$$

Tomado de Computer Organization and Design, The Hardware/Software Interface RISC-V Edition – David Patterson, John Hennessy

3. DESCRIPCIÓN DEL PROBLEMA DE DISEÑO: ESPECIFICACIONES Y RESTRICCIONES

Se debe implementar un algoritmo de multiplicación matricial [(nxn) * (nx1) = (nx1)] en lenguaje ensamblador y además calcular métricas de desempeño básicas para un procesador RISC-V. El código desarrollado debe poder realizar dicha operación para casi cualquier combinación de números en esas matrices con valores $2 \leq n \leq 4$. Lo anterior implica que el programa diseñado debe ser óptimo y parametrizado. Por simplicidad y manejo de los operandos se van a utilizar únicamente números enteros sin signo de 32 bits (unsigned int 32, representación desde 0 hasta 4.294.967.295).

Para desarrollar este algoritmo, se debe tener en cuenta que la información (datos) de los operandos para ejecutar las operaciones se encuentran en la memoria (no en los registros del procesador). Así, el valor de **n** se encuentra en la dirección de memoria (0x10000000); los datos (operandos de 32 bits) se deben cargar desde la posición 0x10000004 hasta la posición determinada por el valor de **n** (nxn operandos de la primera matriz y nx1 operandos de la segunda matriz, ordenados en memoria de manera ascendente). También, al final de la operación se debe retornar la respuesta (elementos ordenados del vector de solución) en una ubicación concreta de la misma memoria (direcciones desde 0x10000100).

Previo al desarrollo de este problema se debe tener instalado en su equipo el simulador RISC-V RIPES en su versión 2.2.2 (https://github.com/mortbopet/Ripes), este le permitirá ejecutar un programa de Assembler en la especificación base de RISC-V (RV32I sin extensión de multiplicación 'M') con una microarquitectura de 32 bits. Además de esto,

también se puede observar el comportamiento de cada etapa del procesador a lo largo de la ejecución y el contenido de registros y memoria en cada ciclo. Esto sumado al manejo de memoria caché de datos e instrucciones. También, ofrece la oportunidad de realizar comparaciones de la ejecución del mismo código en diferentes microarquitecturas, observando y calculando el desempeño de los programas en distinto hardware. Por esto, se propone realizar pruebas comparativas del algoritmo de solución desarrollado ejecutándolo en tres microarquitecturas distintas: ***single cycle***, ***pipeline*** y ***dual-issue***, demostrando con cálculos sencillos los resultados obtenidos.

Para el desarrollo de este problema de diseño se proponen las siguientes etapas:

- **Etapa 1, Diseño del Macroalgoritmo:**

Describir las etapas funcionales del programa por medio de un macroalgoritmo en el que se hagan evidentes las etapas de ejecución, momentos de decisión y operaciones necesarias.

- **Etapa 2, Implementación de la solución en lenguaje de alto nivel (opcional):**

Antes de aventurarse a escribir el código de Assembler de RISC-V se recomienda a los grupos realizar su propia implementación de este algoritmo usando el lenguaje de alto nivel de su preferencia (C/Python/Matlab/Java…), esto les permitirá identificar que ciclos o saltos han de necesitar en un futuro, qué variables se deben implementar para este fin, qué funciones son necesarias y que operaciones iterativas.

- **Etapa 3, Implementación del algoritmo en lenguaje Assembler:**

En el archivo ***base.s*** se encuentran los datos cargados en memoria que son el valor de **n**, los **nxn** operandos correspondientes de la matriz principal y los siguientes **n** elementos de la matriz **nx1** que deben ser operados por el algoritmo para obtener la matriz **nx1** de respuesta. Estos datos se indican debajo de la línea **.data**. Las instrucciones en Assembler se deben escribir dentro debajo de la línea **.text**, desde la función **main**. El final de la ejecución del programa está marcado por el **label end: nop**.

Para escribir el programa se recomiendan los siguientes pasos:
1. Describa los registros a utilizar, sus nombres y su función dentro de cada función (registros de uso temporal, registros de variables fijas, registros para almacenar direcciones de memoria, etc.).
2. Describa cada etapa de operación (líneas de código que modifican registros por medio de operaciones lógicas) y de carga de memoria que han de incluirse en cada etapa o función del programa. Todavía no involucre saltos entre cada bloque y enfóquese en escribir las instrucciones con los operandos correctos.
3. Describir los saltos que van a utilizarse dentro de cada función (saltos condicionales) que nos permiten formar bucles cortos o tomar decisiones con respecto al estado de los registros. Todavía no involucre la interacción entre cada bloque y enfóquese en definir los

puntos de salto de cada instrucción, así como el orden en el que se van a leer las instrucciones.
4. Describir los saltos entre cada bloque funcional (saltos no condicionales) que requieren conocer las instrucciones a las cuales deben regresar, describa como guardará las direcciones (uso del *stack pointer* y de registros de propósito general diferenciados de los de propósito especial) y en que partes se ha de llamar a un callee o se retornará a un caller.

- **Etapa 4, Validación:**

Probar con 5 vectores de prueba adicionales al proyecto base que se entrega asegurándose de entender la ejecución de las instrucciones del programa en el procesador. Se debe validar el correcto funcionamiento del algoritmo para los 5 vectores de prueba entregados y para las tres microarquitecturas mencionadas.

- **Etapa 5, Cálculo de métricas de desempeño:**

Calcular las métricas de desempeño presentadas previamente y explicar claramente su significado e interpretación en el algoritmo (código) específicamente desarrollado y las microarquitectura probadas. Presentar dichas métricas para cada vector de prueba y para las tres microarquitecturas en forma de tabla comparativa (tomar como base para todas las simulaciones 100ms (clock rate ≈ 10Hz) de ejecución por pasos).

- **Etapa 6, Análisis de microarquitecturas:**

Finalmente, con la intención de explorar más a profundidad las microarquitecturas RISC-V, compare y analice (con base en las simulaciones de RIPES y las métricas de desempeño calculadas) las 3 microarquitecturas explicando por qué cambia el IPC (o CPI) y qué componentes hardware afectan la ejecución del algoritmo en cada microarquitectura.

4. CRONOGRAMA DE ENTREGAS

Conformación de grupos: El proyecto debe realizarse en grupos de 2 o 3 personas máximo. No se aceptarán modificaciones en la conformación de los grupos durante el desarrollo del proyecto. Los grupos DEBEN estar conformados por estudiantes de la misma sección de laboratorio, para que puedan reunirse y aprovechar tanto las horas de clase como las horas de laboratorio para trabajar en equipo. La asistencia es obligatoria a las sesiones de laboratorio. Este trabajo está pensado para ser desarrollado por 2 o 3 personas, sea muy responsable en la elección de su grupo, establezca responsabilidades, cronograma de trabajo y sobre todo propóngase una meta y trabajen hasta conseguirla.

Domingo 24 de octubre: Entrega de Informe, descripción de la solución (100%)

La entrega consiste en un sencillo informe de laboratorio en el que se deben incluir la descripción del macroalgoritmo de la solución Se deben describir brevemente los registros y espacios de memorias usados en la solución, el funcionamiento de cada bloque funcional del programa y los diferentes métodos que se llevaron a cabo para mejorar el rendimiento del código. Anexe capturas de las simulaciones realizadas. Adicionalmente anexe su código en Assembler de la solución.