Diseño e Implementación de un ASIP de desencriptación mediante RSA

Jasson Rodríguez Méndez, Marco Herrera Valverde, Kenneth Hernández Salazar, Edgar Chaves González
jassonrm@icloud.com m.herrera0799@gmail.com khken21@gmail.com edjchg@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—A Modular Image Decryption Engine (MIDE) approach is presented in this paper. The main goal of implementing this *Application Specific Instruction Set Processor*(ASIP) is to apply a certain decryption algorithm (in this particular case a Rivest-Shamir-Adleman(RSA)) over an image. A microarchitecture and achitecture have been developed to process and decrypt images using a known public and private keys for RSA algorithm. The architecture implemented is RISC and the microarchitecture is pipelined in combination with vector units, required for enhancing the specific process of image decryption. The original image was found after the algorithm implemented using MIDE in 13.64 ms, showing that even though RSA uses the module operation (very complex computationally) the performance is the expected. Due to the micro and architecture chosen and implemented, the FPGA resources were enough to store the image, the decrypted image, instructions, and implement the entire microarchitecture including the VGA controller.**

***Keywords*—ASIP, RSA, FPGA, Computer Microarchitecture, Computer Architecture, RISC, MIMD.**

I. Introducción

La segmentación en los procesadores es una técnica importante ya que de acuerdo con [3] esta nos permite mejorar la eficiencia del procesador, ya que a la hora de segmentarlo la cantidad de operaciones que se hacen sobre una instrucción se reduce, lo cual nos permite el usar frecuencias de reloj mas altas en comparativa con su contraparte sin segmentación, pero también nos señalan que la implementación de este tipo de micro-arquitectura tiene sus lados negativos, como lo son el consumo de energía y nuevos riesgos de datos y control. Siguiendo en el reino de los procesadores podemos hablar de los vectoriales, [2] menciona que estos tiene unidades de ejecución acondicionadas a procesar con una sola instrucción diferentes datos, lo cual lo hace eficiente a la hora de procesar una gran cantidad de información.

En actualidad los métodos de encriptación se han convertido en factores muy importantes en la comunicación a través internet, estos nos ayudan a mantener nuestros datos seguros, a partir de esta necesidad nace el método de encriptación de llave pública, según R. Rivest, A. Shamir y L.Adleman [1] este consiste en que cada usuario tiene un procedimiento de encriptación, el cual es público, pero el usuario mantiene en secreto su proceso especifico de desencripción, de esta forma manteniendo el dato seguro ya que la forma de desencriptarlo es secreta.

En este documento se explica el desarrollo de un procesador MIMD, este procesador implementa segmentación y unidades vectoriales de 256 bits, esto para poder decodificar una imagen que implementa el tipo de encriptación de llave publica, en la sección de algoritmo desarrollado II se explicara a detalle la micro-arquitectura y la arquitectura implementada para la desencriptación de las imágenes.

La solución consta de diferentes partes, por un lado está el diseño de un ISA con la cantidad de bits necesarios para poder direccionar 16 registros escalares (32 bits) y 8 vectoriales (256 bits), el uso de inmediatos, direccionamiento de memoria y control de flujo de algoritmo. Por otro lado está la organización en hardware(procesador) que permite que el *set* de instrucciones antes mencionado pueda ser ejecutado. Otra de las partes importantes es un compilador que facilite la conversión de instrucciones legibles para las personas, a código máquina, el cual pueda ser leído por el procesador y al final ejecutado. Otra parte importante es el conjunto de memorias que serán utilizadas para guardar y leer la imagen encriptada, escribir la imagen desencriptada, lectura de instrucciones y almacenamiento de valores temporales. Y finalmente, otra parte involucrada es el controlador VGA, encargado de mostrar la imagen que ha sido escrita por el procesador en la memoria. Todas estas secciones serán profundizadas a lo largo del *paper*.

II. Algoritmo Desarrollado

II-A. Arquitectura

En esta sección se explicará en grandes rasgos el tipo de instrucciones creadas y su propósito dentro de la solución del problema, las instrucciones se dividen en 4 categorías, las operaciones R son normalmente operaciones entre registros, las tipo I involucran inmediatos explícitos o implícitos mediante el uso de *labels*, tipo J son instrucciones de control que involucran una dirección de memoria a una instrucción y las tipo V son las instrucciones que implementan uso de registros vectoriales y operaciones entre ellos, estas se pueden observar en la green card en el siguiente **enlace** .

II-A1. Instrucciones aritméticas entre registros: Las instrucciones vistas en la sección *R Type* de la green card son operaciones básicas entre registros, son ejecutadas en una unidad lógica aritmética estándar.

II-A2. Instrucciones Aritméticas con Inmediatos: Estas instrucciones son en el tipo de operación que se ve en la sección *I Type*, consisten en un registro que sera operado con un inmediato y otro registro en el cual se guardara el resultado,

la instrucción *Add Immediate* en combinación con el registro R0 el cual siempre contiene el valor 0 es utilizado como la forma de cargar inmediatos a registros.

II-A3. Instrucciones de control: En esta sección se encuentran todas las instrucciones que realizan diferentes saltos en la memoria de instrucciones, se encuentran en la green card en las secciones *I Type, R Type y J Type*, estas utilizan un *Label* que indica la instrucción a la que saltarán. En cuanto a los saltos condicionales estos utilizan dos registros y un *label* el cual es cargado en el formato I como un inmediato, la instrucción *CALL Label* en el compilador es interpretada como una carga de la posición de memoria de instrucciones a un registro RA y posteriormente un salto al *Label*.

II-A4. Instrucciones de manejo de memoria: Estas son instrucciones básicas de accesos de memoria, se encuentran en la sección *I Type*,estas contienen dos registros, en caso de escritura uno contiene la posición de memoria que sera accedida y el otro es el registro en el que el dato leído se va a escribir en memoria, al contrario en el de lectura este es el registro en el que se va a escribir.

II-A5. Instrucciones Vectoriales: Estas instrucciones hacen uso de la unidad de ejecución vectorial, y de los accesos vectoriales a memoria y unidad de adelantamiento, esta realizan operaciones de carga y aritméticas básicas como las mencionadas previamente pero acondicionadas al manejo de buses de 256 bits. Hay que resaltar la instrucción *Vector Conditional Subtraction* que si se cumple que si los elementos de los vectores son iguales entre ellos se realiza una resta.Estas instrucciones se puede ver en la hoja *V Type*.

II-B. Microarquitectura

Para la microarquitectura se creo un procesador con segmentación, dicho procesador también presenta registros y unidad de ejecución vectoriales, en esta sección se explicaran las secciones importantes del procesador, haciendo énfasis en las adaptaciones vectoriales, el diagrama del procesador se puede ver en el siguiente **enlace**.

II-B1. Búsqueda: Esta sección se encarga de la carga de instrucciones de la memoria, esta compuesta por la memoria de instrucciones, el registro PC que contiene la posición de donde va a ser accedida la memoria, también tiene un seleccionador para seleccionar a que posición de memoria avanzar, este tiene de entrada PC+4 y direcciones de memoria que surgen de las instrucciones de control.

II-B2. Decodificación: Esta sección se dedica a la carga de los diferentes registros y extensión de números inmediatos, sus principales compuestos son dos bancos de registros, el primer banco de registros es estándar, contiene registros de 32 bits con entradas de escritura y lectura, el otro banco de registros es uno vectorial, cada registro esta compuesto por 8 sub registros de 32 bits, esto para dar un total de 256 bits.

II-B3. Unidad de control: Esta unidad es básicamente un decodificador encargado de general todas las señales de control, este tiene como entrada los códigos de operación y códigos de control de la unidad lógica aritmética, si estos últimos no son proveídos los genera a partir del código de operación único de la instrucción.

II-B4. Ejecución: El factor vectorial esta muy presente en esta sección, ya que esta contiene dos unidades lógicas aritméticas, pero la segunda es vectorial, por lo que sus operandos son de 256 bits, además de que esta contiene la capacidad de hacer restas condicionales, esenciales para la decodificación de la imagen. A la entrada de estas unidades se encuentran seleccionadores para seleccionar entre los operandos disponibles y de salida se tienen banderas para el control de saltos condicionales.

II-B5. Memoria: Una memoria acondicionada a escritura y lectura vectorial es esencial en esta parte del procesador, este tiene entradas de 256 bits como de 32 bits para el manejo de escritura para registros normales como vectoriales, una sección de esta es leída por un VGA para poder mostrar la imagen.

II-B6. Escritura: Esta sección esta encargada de escribir en el banco de registros los datos obtenidos de las secciones de memoria o de ejecución.

II-B7. Unidad de adelantamiento: Esta unidad de adelantamiento compara los registros que están siendo operados en la unidad de ejecución con los que están siendo cargados o escritos en las secciones de escritura y memoria, al encontrar que uno de estos son iguales se genera una señal de control que adelanta el dato obtenido a la operación en ejecución, esta unidad esta también acondicionada para el manejo de adelantar registros vectoriales.

II-B8. Registros Intermedios: Estos registros se encuentran en cada etapa, se encargan en cada ciclo de reloj de cargar y guardar la instrucción a la etapa siguiente, esta lleva todos los datos necesarios para cada etapa, como lo son los datos como direcciones de memoria,operandos y códigos de operación requeridos para la unidad de adelantamiento.

II-C. Compilador y manejo de riesgos

El compilador como se puede ver en la imagen del siguiente **enlace** tiene 3 botones, el botón *save* abre una ventana como se ve en la imagen de este **enlace**, en esta se escribe el nombre del archivo de salida, se acepta cualquier extensión pero se escribe en formato *memory initialization file (.mif)* utilizado por SystemVerilog, el botón load carga la dirección del .txt o .mide que se quiere leer y compilar, al seleccionar el botón a compilar se genera el archivo de salida, si todo estaba correcto se genera un archivo con los códigos hexadecimales listos para su uso como se ve en esta **imagen**, sino se genera un archivo similar al visto en la imagen de este **enlace**, el cual contiene los errores que se lograron detectar en el código.

II-C1. Proceso de compilación y analizador de sintaxis: El proceso de compilación y analizador de sintaxis es simple, inicialmente se filtran todas las lineas que no contengan una instrucción, seguidamente se lee instrucción por instrucción, en caso de que se encuentre un label este es asignado a la siguiente instrucción.

A la hora de crear el valor binario se hace básicamente con una tabla en la cual se comparan los operandos con valores pre-establecidos, en caso de los labels se busca a cual instrucción están ligados y con el valor del numero de linea de esta instrucción se calcula la dirección de salto y es escrita.

II-C2. Manejo de riesgos de control: Para el manejo de riesgos de control lo que se realizo es añadir stalls después de cada instrucción de control, estos stalls son instrucciones con valores 0 que no realizan ningún tipo de operación.

III. Resultados

III-A. Resultado de desencriptación

Como se ve en la imagen 1 la desencripción se logro realizar, la imagen a la izquierda y derecha son una captura tomada del simulador VGA de **Eric Eastwood**, el procesador desarrollado dura aproximadamente 13,64 ms con un reloj de 500 Mhz en la desencripción de la imagen. Los valores de la imagen fueron comparados a mano con la imagen resultado generado por python, por lo que se cree que la claridad de la imagen es ocasionada por el simulador VGA.

Figura 1. Imagen final desencriptada utilizando la arquitectura desarrollada.

III-B. ALU vectorial

En la imagen 2 se puede ver un ejemplo de multiplicación de dos vectores, es difícil el análisis de esta debido a que son dos operandos de 256 bits con un resultado de 256 bits, pero podemos tomar los dos primeros operandos de 32 bits de cada vector, estos son A23 y 8B5, sus valores decimales respectivos son 2595 y 2229, la multiplicación de estos da como resultado 5784255, cuyo valor hexadecimal es 5842BF, lo cual calza con los primeros 32 bits del resultado, confirmando que la unidad funciona de forma correcta.

IV. Conclusiones

Los registros y unidades lógicas aritméticas vectoriales añaden más hardware a la microarquitectura, esto debido al manejo de los bancos de registros, cantidad de buses que utilizan y el manejo de riesgos debido a ellos, pero su eficiencia a la hora de ejecutar diferentes datos con el solo uso de una instrucción es ideal a la hora de desencriptar la imagen, lo cual le da ventaja sobre los procesadores que no implementan paralelismo a nivel de datos.

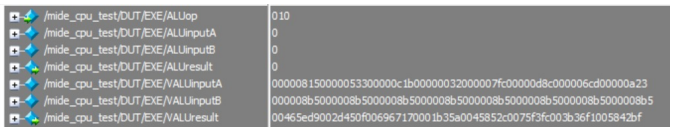

Figura 2. Ejemplo de multiplicación de ALU vectorial.

El uso de vectores no es lo único que llegó a complicar la arquitectura, mientras que el uso de registros en medio de cada etapa ayuda a la eficiencia del procesador este también añadió muchos riesgos de datos y de control. Estos riesgos son mitigados efectivamente mediante el uso de una unidad de adelantamiento, pero los riesgos de control nos remarcan la importancia del análisis del compilador para aplicar el uso de stalls paras su mitigación.

Referencias

[1] R.L. Rivest, A. Shamir & L. Adleman."A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"
[2] Da. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar & K. Asanovic, "Vector Processors for Energy-Efficient Embedded Systems",
[3] I. Finlayson , G. Uh , D. Whalley & G. Tyson,"An Overview of Static Pipelining"(Enero 2012)