UNIVERSIDAD DE COSTA RICA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS DE LA COMPUTACIÓN E INFORMÁTICA

CI0122 – SISTEMAS OPERATIVOS
Prof. Francisco Arroyo

TRABAJO DE INVESTIGACIÓN #2
AVANCE: ARQUITECTURA DE MICROPROCESADORES ARM

Elaborado por:

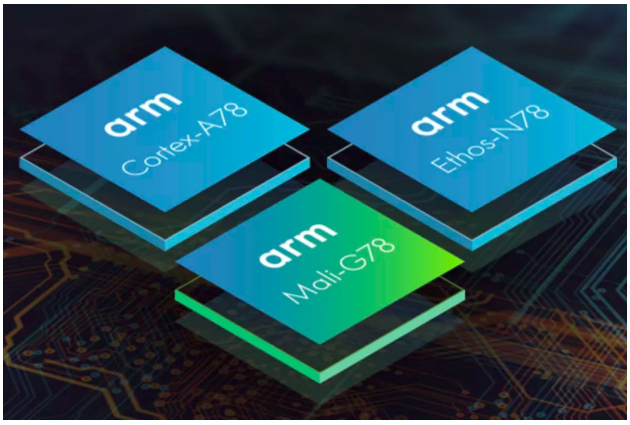
Rodrigo Vílchez Ulloa B78292
rodrigo.vilchez@ucr.ac.cr

20 de noviembre del 2020

Introducción

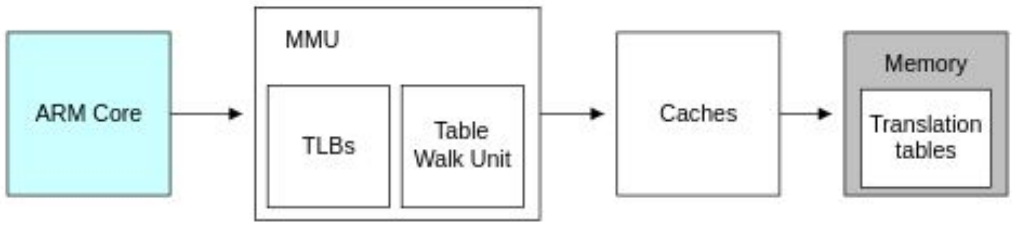
La familia de procesadores ARM se puede dividir en tres grupos: ARM Cortex-A, ARM Cortex-R y ARM Cortex-M. Los primeros se utilizan para procesadores orientados a soportar sistemas operativos, los segundos se orientan para utilizarse en procesamiento de señales en tiempo real y el último se utiliza para aplicaciones MCU y SoC. Los procesadores ARM utilizan operandos de 32 bits, lo que significa que las direcciones de memoria también deben ser de 32 bits.

A diferencia de otras arquitecturas, los procesadores no se crearon con la intención de darle funcionamiento a computadoras de escritorio o de uso diario como las de portátiles de hoy día, sin embargo, su enfoque va más allá a dispositivos con "menor capacidad" que estas computadoras tradicionales, esto quiere decir que los dispositivos son un poco más limitados en cuanto a recursos, pero precisamente esto hace que los procesadores ARM encuentren un buen rendimiento en dispositivos móviles o sistemas empotrados.



Jerarquía de memoria

Los procesadores ARM en general constan de un conjunto de 37 registros cada uno, donde se destaca que el registro r13 es el stack pointer, el registro r15 es el contador de programa y donde todos los registros son de 32 bits. El siguiente nivel jerárquico puede variar un poco dependiendo de la versión del procesador. Por ejemplo, los procesadores Cortex-A8 constan de una caché L1 para instrucciones y para datos, donde el tamaño puede variar de 16 KB a 32 KB cada uno; para la caché L2, se opta por unificar ambas para instrucciones y datos y tener una sola, con una capacidad para 2 MB y con la estrategia asociativa por conjunto de ocho vías. En el modelo Cortex-A9, los procesadores constan de una caché L1 para datos y otra para instrucciones donde el tamaño puede variar de 16 KB a 64 KB y la estrategia utilizada es la asociativa por conjuntos de cuatro vías. Con los procesadores Cortex A-15 MPCore, la caché L1 de datos e instrucciones tienen un tamaño fijo de 32 KB, también constan de una caché L2 donde la capacidad varía entre 512 KB y 4 MB. Versiones más recientes de procesadores como Cortex-A53 y Cortex-A57, que utilizan la arquitectura ARMv8-A, pueden llegar a tener una caché L3 tanto para datos como para instrucciones. Los procesadores también constan de una caché que ejerce la función de TLB, para traducir los números de páginas lógicas a números de páginas físicas.



Esquema de paralelismo y Pipeline

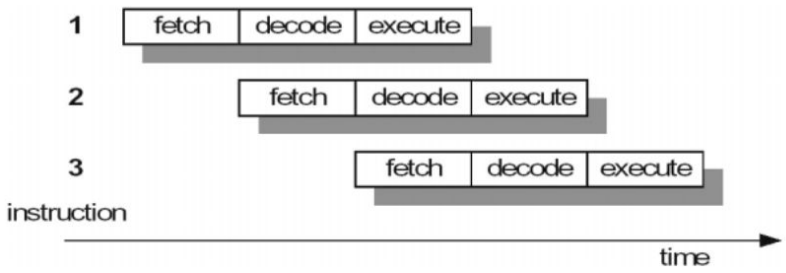
El pipeline en un procesador se refiere a los pasos lógicos que siguen las instrucciones al momento de ejecutarlas, también, implica qué dispositivos lógicos del procesador se utilizan en el proceso. El pipeline es necesario para dividir en etapas la ejecución en su totalidad de una instrucción, donde en medio de cada etapa existen registros que funcionan como buffers o almacenamiento temporal, los que permiten que por cada etapa del pipeline haya una instrucción distinta ejecutándose, pues una vez que una etapa ha concluido, otra instrucción puede entrar a ejecutar en esa etapa e ir completándolos paso a paso conforme avanza entre todas las etapas. Este es el nivel de paralelismo por hardware ofrecido por un procesador, por lo que dependiendo del procesador, de la arquitectura, del diseño, etc., un procesador puede tener una cantidad mayor o menor de etapas en el pipeline.

En los procesadores ARM, la etapa del pipeline se divide en tres: fetch, decode, execute. Al ser un procesador con operandos de 32 bits y con la necesidad de buscar agilidad al momento de ejecutar instrucciones, se decidió dividir el pipeline en esas tres etapas para sacar el máximo provecho posible.

En la etapa fetch, la instrucción se extrae de memoria y se coloca en la instrucción del pipeline. En la etapa decode, la instrucción es decodificada y las señales de control de ruta de datos es preparada para el siguiente ciclo, en esta etapa hay tres puertos para lectura de operandos, de manera que la mayoría de instrucciones puedan obtener sus operandos en un ciclo. En la última etapa, se leen los registros temporales, se realizan operaciones y la ALU genera el resultado necesario para que sea escrito en el registro destino, es aquí donde se calcula la dirección de memoria en caso de que la instrucción fuese un store o un load y en caso de ser necesario, se accede a la memoria.

En esta arquitectura se hace especial atención a ciertos peligros que previenen que la siguiente instrucción sea ejecutada durante el ciclo de reloj designado para ella, y que pueden afectar el rendimiento del pipeline. Existen tres tipos de peligros presentes en esta arquitectura: **peligros estructurales,** que suelen surgir de conflictos de recursos cuando

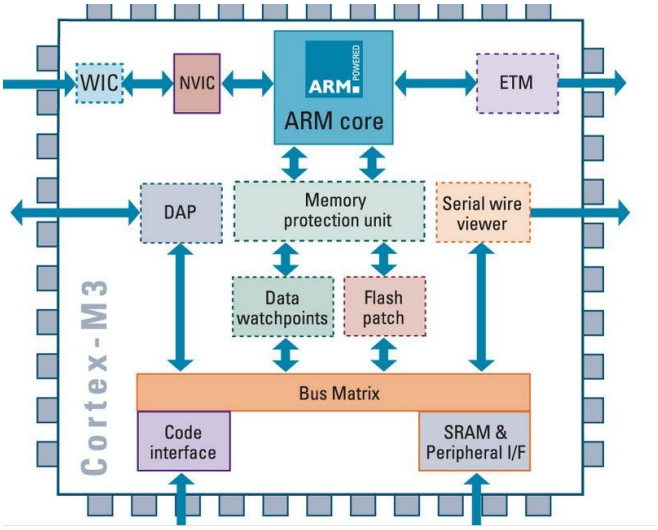
el hardware del procesador no puede admitir todas las combinaciones posibles de instrucciones en ejecución superpuesta simultánea. **Peligros de datos,** que surgen cuando una instrucción depende del resultado de una instrucción anterior en cierta manera que se expone al traslapo de instrucciones en el pipeline. Y por último, **peligros de control** que ocurren de las etapas del pipeline al momento de ejecutar branches y otras instrucciones que cambian el contador del programa. Existen ciertas maneras de resolver estos problemas, por ejemplo, para los peligros de datos, se realiza una técnica denominada forwarding, que permite que ciertas etapas del pipeline le envíen datos a etapas anteriores, de manera que no tienen que esperar a que se haga un write back al registro destino, en la última etapa del pipeline.



Asistencia de hardware para sincronización

Para sacar el máximo provecho a los recursos y al potencial del procesador, en la arquitectura ARM se permite la programación simultánea de multihilos para hacer más eficiente la ejecución de procesos en el procesador. Esta arquitectura permite varios hilos ejecutándose al mismo tiempo a través de todo el sistema. La idea detrás de todo esto es que, cuando un hilo está corriendo, se le provee de todos los recursos necesarios para su ejecución, lo que también permite que se compartan la mayor cantidad de recursos entre cada hilo así lo requieran. Las estructuras físicas que comparten los hilos y que permiten hacer una sincronización apropiada, ya sea para compartir datos entre hilos o para sincronizarlos pues es necesario para realizar una tarea en particular, son: unidades de ejecución, programadores, archivos de registro, decodificadores, cachés, colas, TLB de

datos y de instrucciones, solo que estos últimos pueden ser accesado únicamente por el hilo que los posee, de manera que lo que se comparte son las direcciones para que los hilos tengan esa referencia. Se podría decir que el recurso más importante para realizar una correcta sincronización es el reloj, el cual permite que los hilos no corran a "tiempos diferentes" y que esperen a otros hilos de ser necesario.



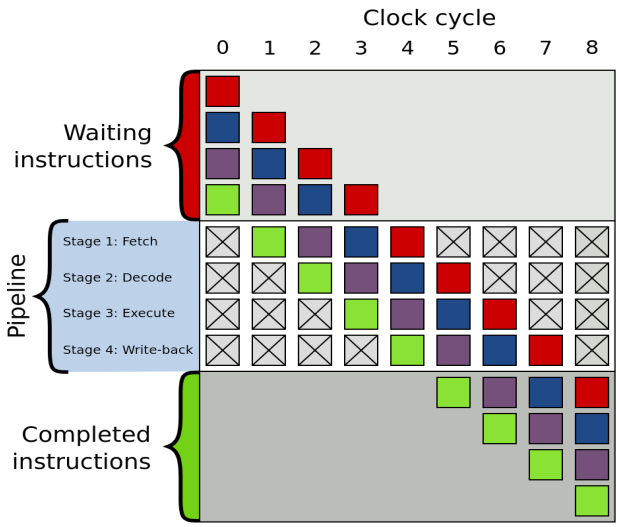
Predicción

En un procesador, la predicción es útil cuando en un branch se necesita saber el resultado para saber la dirección de memoria de la instrucción siguiente al branch, de manera que el uso de los recursos del procesador se hace más eficiente.

En los procesadores ARM, al igual que en otras arquitecturas, existen dos tipos de predicción, las cuales son predicción dinámica y predicción estática, que intentan predecir si el branch va a ser tomado o no tomado, para predecir igualmente la dirección de la siguiente instrucción a partir de las condiciones del branch.

Con la predicción dinámica, el procesador mantiene un registro, que se va actualizando con cada branch, de los resultados de branches anteriores, esto es, si fueron tomados o no tomados. Esta información se almacena como una relación entre

branch-resultado y se almacena en una caché la cual se denomina Branch Target Address Cache (BTAC), aunque en ciertos modelos de procesadores se utiliza la BTB: Branch Target Buffer. Si la información del branch no está presente en esta caché, el procesador opta por utilizar la predicción estática, la cual presume que siempre un branch va a ser tomado solamente en el caso de que ese branch sea un condicional que su dirección de resultado va hacia atrás del PC, y se presume que el branch es no tomado si la dirección resultado va hacia adelante en comparación con el contador de programa. Esto da paso a que, en los procesadores ARM, se pueda predecir con una probabilidad de acierto del 85% si el branch va a ser tomado. En caso de que la predicción falle, el pipeline del procesador debe ser llenado con instrucciones diferentes, y la instrucción actual es detenida en ese momento y se saca del pipeline.



Referencias a utilizar

Alonso, R. (2020, 4 julio). *Todo lo que necesitas saber sobre los procesadores ARM*. HardZone. https://hardzone.es/tutoriales/componentes/procesador-arm/

ARM, D. (2020). *Documentation – Arm Developer*. ARM Developer. https://developer.arm.com/documentation/den0024/a/The-Memory-Management-Unit

Arm Ltd. (2019). *Education Kits – Arm*. Arm | The Architecture for the Digital World. https://www.arm.com/resources/education/education-kits

Presentaciones: The ARM Architecture With a focus on v7A and Cortex-A8
y ARM Processor Architecture (Jin-Fu Li) obtenidas de: http://os.ecci.ucr.ac.cr/ci0122/

Hsiung, P. (2007). *ARM Processor Architecture*. National Chung Cheng University. https://www.cs.ccu.edu.tw/~pahsiung/courses/ese/notes/ESD_03_ARM_Architecture.pdf

Kernel, L. (2014). *Branch Prediction logic in ARM*. Linux Kernel Hacker. http://linuxkernelhacker.blogspot.com/2014/07/branch-prediction-logic-in-arm.html