Investigación: Arquitectura ARM

Fabián Montero Villalobos
Ingeniería en Computadores
Tecnológico de Costa Rica
Cartago, Costa Rica
fmonterov@estudiantec.cr

José Julián Camacho Hernández
Ingeniería en Computadores
Tecnológico de Costa Rica
Cartago, Costa Rica
jcamacho341@estudiantec.cr

Alejandro Soto Chacón
Ingeniería en Computadores
Tecnológico de Costa Rica
Cartago, Costa Rica
soto@estudiantec.cr

I. Introducción

A grandes razgos, cualquier programa que se ejecute en una computadora se debe traducir a un lenguaje que un procesador sea capaz de comprender. Esto tiene una gran cantidad de implicaciones. Algunas de ellas son:

- Para que el código fuente de un programa pueda convertirse en un binario ejecutable, tiene que pasar por un proceso de compilación, para así convertirse en instrucciones que un procesador pueda ejecutar.
- Como existen muchos tipos de procesadores diferentes, existen también muchas formas de escribir una instrucción para un procesador, aunque se produzca a partir del mismo código fuente.

A los diferentes «tipos» de procesadores, se les llama comúnmente **arquitecturas**. Un poco más formalmente, la arquitectura de un computador es un grupo de reglas y métodos que describen la funcionalidad, organización, e implementación de un procesador.

Al conjunto de instrucciones, tipos de dato, registros, y otras características generales de esos procesadores se les conoce como ISA (*instruction set architecture*, por sus siglas en inglés).

II. Aspectos básicos de la arquitectura ARM

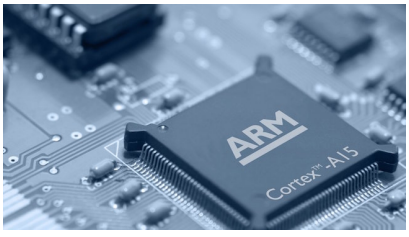

Figura 1. Ilustración de un proceador ARM genérico.

En esta investigación se expondrá acerca de la arquitectura ARM, la cual tiene las siguientes variantes válidas: [1]

- ARMv4
- ARMv4T
- ARMv5T
- ARMv5TExP
- ARMv5TE
- ARMv5TEJ
- ARMv6 clásico
- ARMv6-M
- ARMv6-R
- ARMv7-A
- ARMv7-M
- ARMv7-R
- ARMv8-A (introduce A64, un modo de 64 bits)

Especificamente, se detallará acerca de **ARMv4**.

ARMv4 es una arquitectura RISC (*Reduced Instruction Set Computer*, por sus siglas en inglés) lo cual implica que, en teoría, tiene un grupo de instrucciones reducida en comparación a arquitecturas CISC, que tienden a ser más complejas.

Las características principales del diseño de ARM son: [1]

- Un *register file* uniforme y relativamente grande
- Arquitectura *load/store*. Esto implica que las instrucciones están divididas en dos grandes categorías: acceso de memoria y operaciones aritméticas.
- Instrucciones de tamaño uniforme y de longitud fija
- Control sobre la *ALU* y el *shifter* durante en la mayoría de instrucciones de procesamiento de datos
- Modos de direccionamiento auto-incrementales y auto-decrementales
- Instrucciones de *load* y *store multiple*
- Ejecución condicional de instrucciones

A continuación, se resumirán aspectos básicos acerca de la arquitectura ARMv4.

II-1. Registros: ARM tiene 31 registros de 32 bits de uso general, de los cuales solo 16 son visibles a la vez. Los otros 16 se utilizan para procesamiento de excepciones. De los 16 registros visibles, tres de ellos tienen un rol especial: [1]

- ***Stack pointer*:** El registro **R13** es utilizado normalmente como el stack pointer.
- ***Link register*:** El registro **R14** es comúnmente utilizado como *link register*, es decir, se utiliza para

guardar la dirección de la siguiente instrucción luego de haber hecho una llamada.
- ***Program counter*:** Comúnmente se utiliza el registro 15 como el *program counter*. Representa la dirección de la instrucción actual más 8 bytes. Se incrementa automáticamente luego de cada instrucción.

Los registros desde R0 hasta R7 son registros *unbanked*. Esto significa que cada uno de ellos se refiere al mismo registro físico de 32 bits en cualquier modo del procesador. Son de uso general y no tienen ninguna función implicada por la arquitectura.

Los registros desde R8 hasta R14 son registros *banked*. Esto significa que la direccón a la que se refieren depende del modo del procesador. Normalmente el registro R13 se utiliza como *stack pointer* y el R14 se utiliza comúnmente como *link register*.

El registro R15 se utiliza normalmente como el *program counter*. También, en lugar de otros registros de uso general para producir algunos efectos especiales. Estos efectos dependen de cada instrucción que utilice este registro.

Como se dijo arriba, el resto de los registros no son visibles y se utilizan para acelerar el procesamiento de excepciones.

II-2. Excepciones: La arquitectura ARM soporta siete tipos de excepciones: [1]

- reset
- intento de ejecución de una instrucción indefinida
- interrupción de software
- aborto de *pre-fetch*
- aborto de *data*
- IRQ interrupción normal
- FIQ interrupción rápida

Cuando sucede una interrupción, los registros estándar son reemplazados por registros relacionados a la interrupción que está sucediendo.

II-A. Registros de estado

Los registros de estado almacenan información acerca del estado actual del procesador. El estado actual del programa se almacena en el el *Current Program Status Register* (CPSR, por sus siglas en inglés), el cual contiene lo siguiente:

- banderas de condición (***N**egative*, ***Z**ero*, ***C**arry, y o**V**erflow*)
- dos bits de desactivación de interrupciones
- cinco bits que codifican el modo actual del procesador
- dos bits que codifican si se están utilizando instrucciones ARM, instrucciones Thumb, o códigos Jazelle

Field	Description	Architecture
N Z C V	Condition code flags	All
J	Jazelle state flag	5TEJ and above
GE[3:0]	SIMD condition flags	6
E	Endian Load/Store	6
A	Imprecise Abort Mask	6
I	IRQ Interrupt Mask	All
F	FIQ Interrupt Mask	All
T	Thumb state flag	4T and above
Mode[4:0]	Processor mode	All

Figura 2. Registros de estado en la arquitectura ARM. Tomado del Manual de Referencia ARM. [1]

II-B. Set de instrucciones ARM

Para efectos de esta investigación, el set de instrucciones ARM se va a subdividir en nueve clases de instrucciones:

- Procesamiento de datos
- Instrucciones de salto (branching)
- Load/store simple
- Load/store múltiple
- Instrcciones privilegiadas o de coprocesador
- Instrucciones privilegiadas o de coprocesador
- Funciones de PSR.
- instrucciones de llamadas al sistema

Nótese que en ARM, todas las instrucciones son condicionales y también todas las relacionadas con manipulación de datos, pueden hacer bitwise shifts.

A continuación, un breve resumen de cada grupo y sus instrucciones.

II-B1. Procesamiento de datos: Instrucciones para manipulación general de datos.

- adc: Suma aritmética con *carry* de dos datos.
- add: Suma aritmética simple de dos datos.
- and: Conjunción lógica entre dos datos.
- bic: Conjunción lógica entre un dato y su complemento.
- cmn: Comparación entre un dato y con el complemento a dos de otro dato.
- cmp: Comparación simple de dos datos.
- eor: Disyunción lógica exclusiva entre dos datos.
- mov: Escribe un valor en un registro de destino.
- mvn: Genera el complemento a uno de algún valor.
- orr: Disyunción lógica inclusiva entre dos datos.
- rsb: Resta el primer valor del segundo valor.
- rsc: Resta el primer valor del segundo valor, con *carry*.
- sbc: Resta el segundo valor del primero, con *carry*.
- sub: Resta el segundo valor del primero.

- teq: Realiza disyunción lógica exclusiva entre cada elemento de dos datos para comprobar su equivalencia.
- tst: Realiza conjunción lógica exclusiva entre cada elemento de dos datos para comprobar su equivalencia.

En la figura 3 se muestra la codificación de la instrucción add.

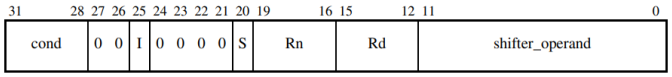

Figura 3. Codificación de instrucción add. Tomado del Manual de Referencia ARM. [1]

II-B2. branching: Instrucciones para control de flujo.

- b: Causa un salto a una dirección.
- bl: Causa un salto a una dirección y guarda la dirección de retorno en el registro R14.

En la figura 4 se muestra la codificación de ambas instrucciones.

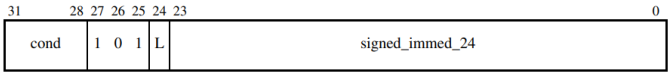

Figura 4. Codificación de instrucciones b y bl. Tomado del Manual de Referencia ARM. [1]

II-B3. Load/Store simple: Instrucciones para manipular 1 datum a la vez. Nótese que las instrucciones que tienen en su nombre la letra «b», manipulan 1 solo byte (en lugar de una palabra entera).

Las que tienen en su nombre la letra «t», realizan una traducción de modo. Esto sirve para que un proceso en *userspace* tenga que solicitar al kernel realizar manipulación de memoria en regiones en las que solo el kernel puede leer o escribir, en lugar de hacerlo él mismo. Esto, en contraste a lo que se hace en otras arquitecturas, que es costosamente verificar las tablas de paginación en software para demostrar que un *buffer* indicado por *userspace* realmente le es accesible.

Finalmente, las que tienen en su nombre las letras «s» y «h», realizan extensión de signos y utilizan 16 bits, respectivamente.

- ldr: Carga una palabra de una dirección de memoria.
- ldrb: Carga un byte de una dirección de memoria.
- ldrbt: Carga un byte de una dirección de memoria y realiza extensión de ceros para convetirla en una palabra de 32bits.
- ldrt: Carga una palabra de una direccíon de memoria.
- str: Guarda una palabra de algún registro en memoria.
- strb: Guarda un byte de algún registro en memoria.
- strbt: Guarda una palabra de algún registro en memoria, con traducción de modo.
- strt: Guarda una palabra de algún registro en memoria, con traducción de modo.
- ldrh: Carga media palabra de una dirección de memoria y realiza extensión de ceros para convertirla en una palabra de 32 bits.
- ldrsb: Carga un byte de una dirección de memoria y realiza extensión de ceros para convertirla en una palabra de 32 bits.
- ldrsh: Carga media palabra de una dirección de memoria y realiza extensión de signos para convertirla en una palabra de 32 bits.
- strh: Guarda media palabra de algún registro.

En la figura 5 se muestra la codificación de la instrucción ldr.

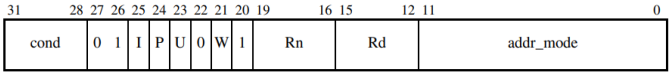

Figura 5. Codificación de instrucción ldr. Tomado del Manual de Referencia ARM. [1]

II-B4. Load/Store múltiple: Instrucciones para manejo de stack. Pueden manipular secuencialmente cualquier cantidad de registros al mismo tiempo.

- ldm: Carga algunos o todos los valores de los registros de uso general de ubicaciones secuenciales de memoria.
- stm: Guarda algunos o todos los valores de los registros de uso general de ubicaciones secuenciales de memoria.

En la figura 6 se muestra la codificación de la instrucción ldm.

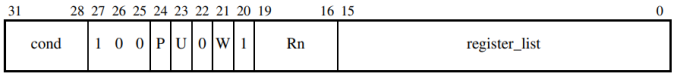

Figura 6. Codificación de instrucción ldm. Tomado del Manual de Referencia ARM. [1]

II-B5. Multiplicación: Operaciones de multiplicación aritmética, con o sin signo, y opcionalmente, *fused multiply add*. Únicas instrucciones que pueden ser de 64bits.

- mla: Multiplica dos valores de 32bits y añade un tercer valor.
- mul: Multiplica dos valores de 32bits.
- smlal: Multiplica dos valores con signo de 32bits para producir un valor de 64 bits, el cual es acumulado con un valor de 64bits.
- smull: Multiplica dos valores de 32 bits con signo y extrae los 32 bits más significativos del resultado.
- umlal: Se multiplican los valores de los registros Rm y Rs. A este producto se le suman los valores de los registros RdHi y RdLo.

- umull: Se multiplican los valores de los registros Rm y Rs. Los 32bits de más arriba se guardan en RdHi y los de más abajo en RdLo.

En la figura 7 se muestra la codificación de la instrucción mla.

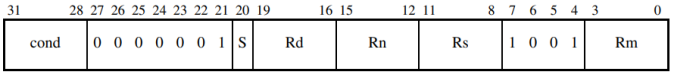

Figura 7. Codificación de instrucción mla. Tomado del Manual de Referencia ARM. [1]

II-B6. Swap: Instrucciones para intercambiar valores en registros con memoria.

- swp: Intercambia una palabra entre registros y memoria.
- swpb: Intercambia un byte entre registros y memoria.

En la figura 8 se muestra la codificación de la instrucción swp.

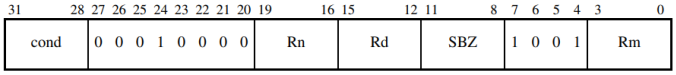

Figura 8. Codificación de instrucción swp. Tomado del Manual de Referencia ARM. [1]

II-B7. Instrucciones privilegiadas o de coprocesador: Estas instrucciones las ejecuta un coprocesador. Sirven para extender el rango de instrucciones que tiene el procesador. Es decir, funciones adicionales que no son expresadas en los registros de propósito general. Un ejemplo de estas funciones son:

- MMU para memoria virtual
- Control de acceso no alineados
- *Caches*
- *Endianess*

Nótese que estas funciones están relacionadas al funcionamiento del sistema.

- cdp: indica una operación que no involucra registros ni memoria del procesador principal.
- ldc: realiza una carga de datos a registros de un coprocesador.
- mcr: copia un registro de coprocesador a un registro del procesador principal.
- mrc: copia un registro del procesador princial a un registro del coprocesador.
- stc: realiza un almacenamiento de datos a registros de un coprocesador.

En la figura 9 se muestra la codificación de la instrucción cdp.

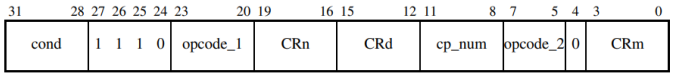

Figura 9. Codificación de instrucción cdp. Tomado del Manual de Referencia ARM. [1]

II-B8. PSR: Instrucciones que mueven datos que están almacenados en los registros de estado hacia registros de uso general, y viceversa.

- mrs: Mueve el valor del CPSR o el SPSR del modo de ejecución actual a un registro de propósito general.
- msr: Mueve el valor de un registro de propósito general al CPSR o el SPSR.

En la figura 10 se muestra la codificación de la instrucción mrs.

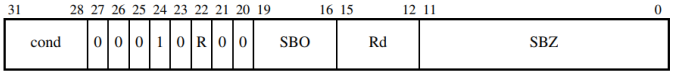

Figura 10. Codificación de instrucción mrs. Tomado del Manual de Referencia ARM. [1]

II-B9. System calls/interrupciones por software: Instrucción para invocar funciones del kernel.

- swi: Causa una interrupción por software.

En la figura 11 se muestra la codificación de la instrucción swi.

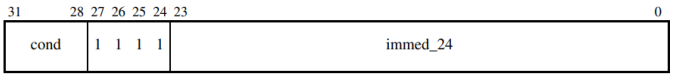

Figura 11. Codificación de instrucción swi. Tomado del Manual de Referencia ARM. [1]

III. Herramientas

Los microprocesadores ARM se encuentran en gran cantidad de los dispositivos que se utilizan en la actualidad. Es por esto que, con el fin de optimizar su desarrollo, resulta fundamental contar con herramientas de simulación, compilación, traducción, entre otras útiles para llevar el flujo de su implementación.

Uno de los tipos de herramientas más útiles para el desarrollo son los emuladores ARM. Estos permiten ejecutar un dispositivo ARM emulado, ya sea Windows, Linux o algún otro sistema operativo. Esto le permite desarrollar y probar los dispositivos en software, para después solo mover el software a un dispositivo real de hardware cuando tenga mayor completitud. [2]

Por medio de la emulación, se intenta de imitar el diseño interno de un dispositivo, en este caso de placas ARM. Muy similar a la simulación, donde solo se imitan las funciones del un dispositivo. [3]

III-1. QEMU: Entre estas herramientas de emulación se encuentra QEMU. Este es un emulador y virtualizador de máquinas genérico y de código abierto. Cuando es utilizado como emulador, es capaz de ejecutar sistemas operativos y programas creados para una máquina (por ejemplo, una placa ARM) en una máquina diferente. Esto lo realiza mediante el uso de la traducción dinámica, logrando un rendimiento óptimo.

Por otro lado, cuando se usa como virtualizador, QEMU consigue un rendimiento casi nativo al ejecutar el código externo directamente en la CPU que se utilice. Adicionalmente, QEMU admite la virtualización cuando se ejecuta bajo el hipervisor Xen o se usa el módulo kernel KVM en Linux. Al usar KVM, es posible virtualizar x86, PowerPC embebido y como servidor, S390, ARM de 32 y 64 bits, entre otros. [4]

III-2. Unicorn: Otro emulador que de hecho está basado en QEMU es Unicorn. Este es un framework ligero, multiplataforma y multiarquitectura para la emulación de CPU. Entre las arquitecturas que es capaz de emular se encuentran ARM, ARM64 (ARMv8), m68k, MIPS, PowerPC, RISC-V, S390x (SystemZ), SPARC, TriCore y x86.

Además, cuenta con compatibilidad con Windows, nix, macOS, Linux, Android, entre otros. Puede llegar a tener un alto rendimiento por medio del uso de la técnica de compilación Just-In-Time. [5]

III-3. ARMware: ARMware es otra alternativa para la emulación de hardware. ARMware proporciona un entorno de emulación para la plataforma ARM (es capaz de emular Intel StrongARM SA-1110). Para la arquitectura ARM versión 4 se pueden emular lo siguiente:

- Conjuntos de instrucciones ARM (sin incluir Thumb)
- Todos los modos de procesador (usuario, interrupción, interrupción rápida, supervisor, cancelación, instrucción indefinida, sistema).

Además, ARMware tiene un compilador dinámico incorporado que traducirá un EBB de códigos ARM en un bloque de códigos de máquina x86, de modo que pueda aumentar el rendimiento en el tiempo de ejecución. [6]

III-4. SkyEye: Otra herramienta es SkyEye, que es una plataforma de código abierto de simulación de hardware de múltiples arquitecturas y núcleos. Es compatible con la estructura del sistema de ARM, Blackfin, Coldfire, PowerPC, MIPS, SPARC y x86. Específicamente, es capaz de emular los núcleos de CPU ARM7TDMI, ARM720T, StrongARM, XScale y Blackfin. El emulador funciona en Linux, Windows (con el ayuda de Cygwin) y FreeBSD. [7]

Algunas de las herramientas anteriormente descritas utilizan archivos ejecutables que son el resultado de la traducción a lenguaje máquina del código fuente, con el fin de ser ejecutados en la emulación de las placas ARM, en este caso. Para realizar dicha tarea, se necesitan ciertas herramientas que concatenadas son capaces de llevarla a cabo, que se denominan como un todo una *toolchain*. En la figura 12 se presenta el flujo del *toolchain* y sus principales componente se explican a continuación: [8]

- Si el programa está escrito en un lenguaje de mayor nivel, es necesario un compilador C/C++ que acepta código fuente y produce archivos en lenguaje ensamblador.
- El ensamblador traduce los archivos fuente en lenguaje ensamblador a módulos de código objeto en lenguaje máquina. Los archivos fuente pueden contener instrucciones, directivas de ensamblador y directivas de macro. Es posible la utilización de directivas de ensamblador para controlar el proceso de ensamblaje, incluido el formato de lista de origen, los datos alineación y contenido de la sección.
- El enlazador combina archivos que contienen código objeto, en un solo módulo objeto ejecutable. Se encarga de realizar una reubicación simbólica, así como de resolver las referencias externas.
- El archivador le permite recopilar un grupo de archivos en un solo archivo, denominado biblioteca estática.
- Finalmente, se pueden utilizar herramientas de depuración para refinar y/o corregir el código. Entre ellas se pueden incluir los simuladores o emuladores.

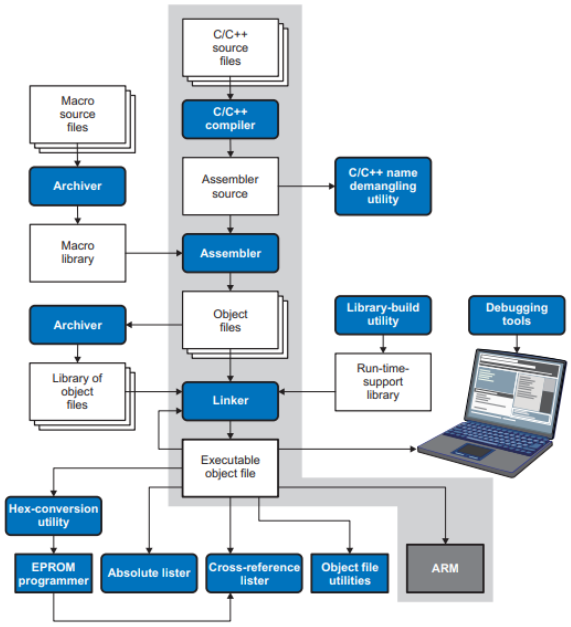

Figura 12. Flujo de desarrollo de software de dispositivos ARM. [8]

El producto principal de este proceso de desarrollo, para este caso, es un archivo de código objeto ejecutable para dispositivos ARM. Este proceso se denomina compilación cruzada. Esta se refiere al acto de compilar código para un sistema (a menudo conocido como destino) en un sistema diferente, denominado *host*. Es una técnica muy útil, por ejemplo, cuando el sistema de destino es demasiado

pequeño para albergar el compilador y todos los archivos relevantes, como lo es el caso de una placa ARM. [9]

Para llevar a cabo estas tareas existen herramientas como el ARM GNU Toolchain, que puede trabajar en conjunto con el compilador GCC. Este es una *toolchain* por sí misma con la cual se puede seleccionar el nivel de traducción que se desea para así obtener un archivo intermedio en el proceso. Cuando se configura para detenerse en determinado punto de la cadena de compilación, se realizan automáticamente todos los pasos anteriores hasta llegar al deseado.

Otro ejemplo es la *toolchain* LLVM, que es una herramienta de código abierto para procesadores ARM que está compuesta por las siguientes herramientas [10]:

- Núcleo LLVM
- Compilador Clang
- Enlazador LLD
- Depurador LLDB

Referencias

[1] Terasic Technologies, *DE1-SoC User Manual*, English, ver. V1.0, 2013, 110 págs.

[2] «ARM Emulators.» (26 de jun. de 2018), dirección: https://www.thefreecountry.com/emulators/arm.shtml (visitado 16-10-2022).

[3] M. Fayzullin. «How To Write a Computer Emulator.» (2000), dirección: http://fms.komkon.org/EMUL8/HOWTO.html (visitado 16-10-2022).

[4] «QEMU: Main Page.» (9 de jul. de 2020), dirección: https://wiki.qemu.org/Main_Page (visitado 16-10-2022).

[5] «Unicorn: The Ultimate CPU emulator.» (), dirección: https://www.unicorn-engine.org/ (visitado 16-10-2022).

[6] «ARMware: An ARM / Compaq iPAQ emulator.» (), dirección: https://halajohn.github.io/ARMware/ (visitado 16-10-2022).

[7] M. Kang. «SkyEye User Manual.» (26 de ene. de 2011), dirección: https://skyeye.sourceforge.net/wiki/index.php/SkyEye_User_Manual (visitado 16-10-2022).

[8] Texas Instruments, *ARM Assembly Language Tools: User's Manual*, English, ver. v18.1.0.LTS, 2018, 356 págs. dirección: https://www.ti.com/lit/ug/spnu118u/spnu118u.pdf.

[9] L. BELAARIBI. «What is cross compilation?» (23 de mar. de 2021), dirección: https://www.linkedin.com/pulse/what-cross-compilation-loutfi-belaaribi (visitado 16-10-2022).

[10] «LLVM Toolchain.» (2022), dirección: https://developer.arm.com/Tools%20and%20Software/LLVM%20Toolchain (visitado 16-10-2022).