# Proyecto Final

Fiorella Delgado León, Jonathan Guzmán Araya, Mariano Muñoz Masís
fiorelladelgado53@gmail.com, jonathana1196@gmail.com, marianomm1301@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica, Cartago, Costa Rica, 2021

***Resumen*—Este documento presenta los resultados del proyecto final del curso.**

***Palabras clave*—FPGA**

## I. Introducción

El procesador ARM es un componente clave de muchos sistemas integrados. El primer ARM se introdujo en 1985, bajo el nombre de Acorn RISC Machine (luego renombrado Advanced RISC Machine). La relativa simplicidad de los procesadores ARM los hizo adecuados para aplicaciones de bajo consumo permitiendo así su actual amplia adopción. ARM es inusual en el sentido de que no vende proesadores directamente, si no que autoriza a otras compañías a construir sus diseños de procesadores, a menudo como parte de un sistema en chip más grande, por ejemplo: Samsung, Altera, Apple, Qualcomm construyen procesadores ARM, ya sea utilizando microarquitecturas compradas en ARM o microarquitecturas desarrolladas bajo licencia ARM [1].

La arquitectura de un computador es la visión del programador de una computadora, esta arquitectura está definida por un conjunto de instrucciones (lenguaje) y ubicaciones de los operandos (registros y memoria) [2].

El primer paso para entender la arquitectura de un computadora es entender su lenguaje, las palabras en el lenguaje de una computadora son llamadas instrucciones, el vocabulario de una computadora es llamado conjunto de instrucciones, de manera que todos los programas en esta computadora utilizan el mismo conjunto de instrucciones, incluso las más complejas.

### *I-A. Tipos de instrucciones y su codificación*

Las instrucciones de una computadora indican el tipo de operación a realizar y los operandos a utilizar, los operandos vienen desde la memoria a los registros y de estos a las instrucciones. Las computadoras solo comprender 1's y 0's, es por esto que las instrucciones son codificadas en como números binarios en un formato llamado lenguaje máquina [2].

El lenguaje Ensamblador es la representación humana de este lenguaje máquina, cada instrucción en Ensamblador especifica tanto el tipo de instrucción como sus operandos. En ARMv4 existen cuatro formatos principales de instrucciones: data procesing, memory, branch y miscellaneous. Este pequeño número de formatos permite cierta regularidad entre las instrucciones, y por lo tanto, un hardware de decodificador simple [2].

*I-A1. Data Procesing Instructions:* Las instrucciones de procesamiento de datos tienen dos registros fuentes, donde el segundo puede ser un inmediato y el tercer registro que es el destino.

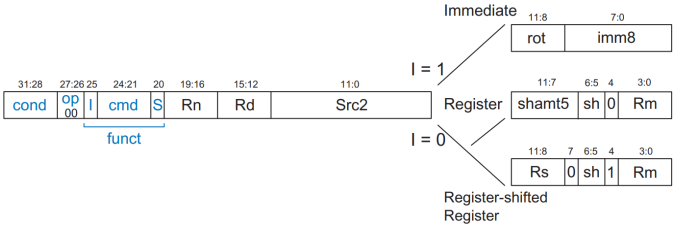


Figura 1. Codificación de instrucciones de procesamiento de datos [2]

Los 32 bits de estos tipos de instrucciones tienen bits específicos asignados para una serie de datos que tienen que ir en esos bits para especificar el tipo de instrucción y que hacer con los mismos, esto se detalla a continuación.

- cond: ejecución condicional basada en las banderas que se especificaron en la Tabla VI.
- op: código del tipo operación a realizar.
- funct: código de instrucción a realizar, a su vez en las instrucciones de procesamiento de datos este se divide en tres partes.
  - I: especifica si se está o no trabajando con un inmediato.
  - cmd: código de la instrucción a realizar.
  - S: si las banderas se tienen que guardar o no.
- Rn: registro para el operando 1.
- Rd: registro destino donde se guardará el resultado de los operandos.
- Src2: en las instrucciones de procesamiento puede presentar tres casos.
  - Inmediato
    - rot: rotación circular de 8 bits para armar números de 32 bits.
    - imm8: inmediato, este no puede ser más de 255.
  - Registro
    - shamt5: constante de desplazamiento.
    - sh: indica el tipo de desplazamiento a realizar como se observa en la Tabla I.
    - 0: bit por defecto según la instrucción.
    - Rm: registro para el operando 2.
  - Registro con desplazamiento
    - Rs: registro de desplazamiento.

- 0: bit por defecto según la instrucción.
- sh: indica el tipo de desplazamiento a realizar como se observa en la Tabla I.
- 1: bit por defecto según el tipo de instrucción.
- Rm: registro para el operando 2.

| Instrucción | sh | Operación |
|---|---|---|
| LSL | $00_2$ | Logical left shift |
| LSR | $01_2$ | Logical right shift |
| ASR | $10_2$ | Arithmetic shift right |
| ROR | $11_2$ | Rotate right |

Cuadro I
CÓDIGO DE OPERACIÓN DE SH

*I-A2. Memory Instructions:* Las instrucciones de memoria utilizan un formato similar al que utilizan las instrucciones de procesamiento de datos, con las mimas seis divisiones, sin embargo, utilizan un código de operación diferente al utilizado en las instrucciones de procesamiento de datos tal y como se muestra en la Figura 2.

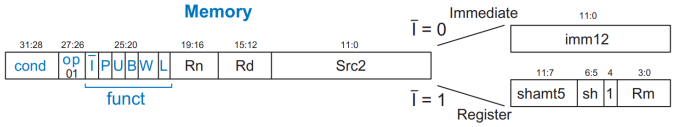


Figura 2. Codificación de instrucciones de memoria [2]

- cond: ejecución condicional basada en las banderas que se especificaron en la Tabla VI.
- op: código del tipo operación a realizar.
- funct: código de instrucción a realizar, a su vez está compuesto de seis bits de control.
  - $\overline{I}$ - U: Los bits $\overline{I}$ (inmediato) y U (sumar) determinan si el desplazamiento es inmediato o de registro y si se debe sumar o restar de acuerdo a la Tabla II.
  - P - W: Los bits P (pre-índice) y W (escritura) especifican el modo de índice de acuerdo a la Tabla III.
  - B - L: Los bits L (carga) y B (byte) especifican el tipo de operación de memoria de acuerdo a la Tabla IV.
- Rn: registro para el operando 1.
- Rd: registro destino donde se guardará el resultado de los operandos.
- Src2: en las instrucciones de memoria puede presentar dos casos.
  - Inmediato
    - imm12: desplazamiento inmediato sin signo.
  - Registro
    - shamt5: constante de desplazamiento.
    - sh: indica el tipo de desplazamiento a realizar como se observa en la Tabla I.
    - 1: bit por defecto según la instrucción.
    - Rm: registro para el operando 2.

*I-A3. Branch Instructions:* Las instrucciones de bifurcación utilizan un desplazamiento de 24 bits, por lo que los 32 bits quedan divididos como se muestra en la Figura 3.

| Bit | $\overline{I}$ | U |
|---|---|---|
| 0 | Desplazamiento inmediato | Restar |
| 1 | Registro | Sumar |

Cuadro II
INSTRUCCIONES DE CONTROL PARA $\overline{I}$ Y U

| P | W | Modo |
|---|---|---|
| 0 | 0 | Pos-índice |
| 0 | 1 | No soportado |
| 1 | 0 | Offset |
| 1 | 1 | Pre-índice |

Cuadro III
INSTRUCCIONES DE CONTROL PARA P Y W

- cond: ejecución condicional basada en las banderas que se especificaron en la Tabla VI.
- op: código del tipo operación a realizar.
- funct: código de instrucción a realizar.
  - 1L: su tamaño siempre es de 2 bits y el MSB siempre es 1, mientras que el LSB indica el tipo de bifurcación a realizar.
- imm24: se utiliza para especificar la dirección de una dirección de instrucción relativa a $PC + 8$.

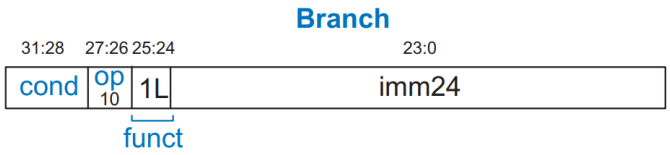


Figura 3. Codificación instrucciones de bifurcación [2]

*I-A4. Miscellaneous Instructions:* El conjunto de instrucciones de ARMv4 tiene otro grupo que se detalla en la Tabla V.

Las instrucciones ARM opcionalmente establecen indicadores de condición en función del resultado Las instrucciones posteriores se ejecutan condicionalmente, dependiendo del estado de esos indicadores, también llamados indicadores de estado que se observan en la Tabla VI.

Estas banderas se activan dependiendo del tipo de instrucción que se esté ejecutando, en la Tabla VII se detallan las instrucciones y los tipos de banderas que estos activarían.

La ALU establece estos indicadores y se mantienen en los cuatro bits superiores del CPSR de32 bits tal y como se muestra en la Figura, mientras que los cinco bits menos significativos son los bits de modo, esto es porque un procesador ARM puede operar en una serie de modos de ejecución con diferentes niveles de privilegios. Los diferentes modos permiten que excepciones tomen lugar sin dañar el estado en que se encuentra. Estos modos están especificados en la parte

| L | B | Instrucción |
|---|---|---|
| 0 | 0 | STR |
| 0 | 1 | STRB |
| 1 | 0 | LDR |
| 1 | 1 | LDRB |

Cuadro IV
INSTRUCCIONES DE CONTROL PARA L Y B

| Instrucciones | Descripción | Propósito |
|---|---|---|
| LDM, STM | cargar/guardar múltiple | Guardar y recuperar registros en llamadas de subrutina |
| SWP, SWPB | Intercambiar byte | Carga atómica y almacenamiento para sincronización de procesos |
| LDRT, LDRBT, STRT, STRBT | cargar/guardar palabra/byte con traslado | Permitir que el sistema operativo acceda a la memoria en el espacio de memoria virtual del usuario |
| SWI | Interrupción de software | Crear una excepción, que a menudo se usa para llamar al sistema operativo |
| CDP, LDC, MCR, MRC, STC | Acceso de coprocesador | Comunicarse con coprocesador opcional |
| MRS, MSR | Mover desde/a registro de estado | Guardar el registro de estado durante las excepciones |

Cuadro V
INSTRUCCIONES MISCELÁNEAS

| Bandera | Nombre | Descripción |
|---|---|---|
| N | Negativo | La instrucción resulta negativa si el bit 31 del resultado es 1 |
| Z | Cero | El resultado es 0 |
| C | Acarreo | La instrucción provoca acarreo |
| V | Desbordamiento | La instrucción causa desbordamiento |

Cuadro VI
BANDERAS

baja del CPSR como se muestra en la Figura 4y se detallan en Tabla VIII.

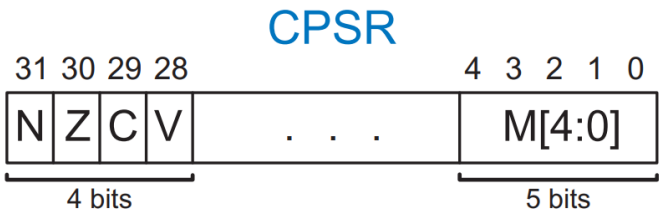


Figura 4. Bits de las banderas [2]

### *I-B. Registros*

Las instrucciones necesitan acceso a operandos rápidamente así estas pueden a su vez ejecutarse rápidamente también, pero los operandos almacenados en memoria toman demasiado tiempo en ser traídos, por lo tanto la mayoría de arquitecturas

| Tipo | Instrucciones | Banderas |
|---|---|---|
| Add | ADDS, ADCS | N, Z, C, V |
| Subtract | SUBS, SBCS, RSBS, RSCS | N, Z, C, V |
| Compare | CMP, CMN | N, Z, C, V |
| Shifts | ASRS, LSLS, LSRS, RORS, RRXS | N, Z, C |
| Logical | ANDS, ORRS, EORS, BICS | N, Z, C |
| Test | TEQ, TST | N, Z, C |
| Move | MOVS, MVNS | N, Z, C |
| Multiply | MULS, MLAS, SMLALS, SMULLS, UMLALS, UMULLS | N, Z |

Cuadro VII
INSTRUCCIONES QUE ACTIVAN LAS BANDERAS

| Modo | $CPSR_{4:0}$ |
|---|---|
| User | 10000 |
| Supervisor | 10011 |
| Abort | 10111 |
| Undefined | 11011 |
| Interrupt | 10010 |
| Fast interrupt | 10001 |

Cuadro VIII
MODOS DEL PROCESADOR

pueden ser visibles por el programador, ARMv4 tiene 16 registros, además de un Registro de Estado del Programa Actual (CPSR), este contiene el estado del procesador y la información de control. La distribución del uso de estos registros se detalla en la Tabla IX.

| Registro | Número | Uso | Reservado |
|---|---|---|---|
| R0-R3 | 0-3 | Registro de parámetros | No |
| R4-R11 | 4-11 | Registro de guardado | Sí |
| R12 | 12 | Registro temporal | No |
| R13 | 13 | Puntero de pila | Sí |
| R14 | 14 | Registro de enlace | Sí |
| R15 | 15 | Contador del programa | |

Cuadro IX
REGISTROS ARMV4

### *I-C. Memoria*

Si los registros fueran la única manera de almacenar operandos estaríamos condenados a realizar solamente programas simples con no más 15 variables, sin embargo, los datos pueden ser almacenados en memoria, a pesar de que los registros son muy rápidos, estos son pequeños, por otro lado, las memorias son grandes pero lentas en comparación con los registros. Por esta razón, usamos variables para ser almacenadas en los registros y existen instrucciones como LDR para cargar un dato de memoria en un registro.

## II. SISTEMA DESARROLLADO

Para el desarrollo de este proyecto, se realizó la implementación de una micro arquitectura, utilizando un procesador ARMv4, para descifrar el texto introducido (el cual estará cifrado) mediante uno de los tres algoritmos de descifrado, por medio del lenguaje System Verilog.

## III. RESULTADOS

## IV. ANÁLISIS DE RESULTADOS

## V. CONCLUSIONES

Al desarrollar este proyecto, se obtuvieron las siguientes conclusiones:

## VI. BIBLIOGRAFÍA

## REFERENCIAS

[1] J. Gomar. (2018, Diciembre) Qué es un procesador ARM y como funciona. Profesional review. [Online]. Available: https://www.profesionalreview.com/2018/12/05/que-es-procesador-arm-como-funciona/

[2] D. M. H. Sarah L. Harris, *Digital Design and Computer Architecture*. Morgan Kauffmann, 2010.