Capítulo 4

Procesador

...

...

Resumen: En este capitulo se presenta el diseño del procesador implementado: su arquitectura,

Procesador

Se ha diseñado e implementado manualmente un procesador con arquitectura RISC basado en la arquitectura de los procesadores DLX estudiados durante el grado en ingeniería de computadores [10]. Se trata de un procesador con un ancho de palabra de 32 bits y una segmentación en 5 etapas.

La implementación ha sido adaptada para poder ejecutar instrucciones del repertorio ARM. En concreto, se permite ejecutar un subconjunto del juego de instrucciones THUMB-2. Este juego de instrucciones es utilizado principalmente por los procesadores de la gama ARM CORTEX M.

4.1. Esctructura

- El *banco de registros* dispone de 16 registros (R0, R1, ..., R15) de propósito general con un tamaño de 32 bits. Estos registros se pueden utilizar tanto para guardar datos leídos de memoria como enviar los valores a memoria. Se puede trabajar con los valores que tengan almacenados ejecutando operaciones sobre ellos. El registro R15 es accesible de forma limitada puesto que el identificador de este registro se utiliza para diferenciar unas instrucciones de otras.

- Además se cuenta con el registro del contador de programa (PC). Este registro especial almacena la dirección de memoria de la instrucción que debe ejecutarse a continuación. Se incrementa automáticamente en 4 cada ciclo y solo se puede alterar este mecanismo por medio de instrucciones de control.
- Las instrucciones tienen un formato variable pero un tamaño constante de 32 bits.
- La memoria es accesible por palabras. Es decir, todo acceso a memoria carga o almacena un dato de 32 bits. Para acceder a memoria se dispone de instrucciones de lectura y escritura de una palabra con direccionamiento relativo a registro base. La dirección de acceso se calcula con un registro base al que se le suma un inmediato de 12 bits.

4.2. Tipos de datos

Para simplificar la arquitectura del procesador, se ha limitado el tamaño de datos a palabras completas de 32 bits. Se trabaja con un bus de ancho de palabra del mismo tamaño donde todos los bits cargados tienen valor.

4.3. Instrucciones

El juego de instrucciones elegido está compuesto por instrucciones de 32 bits con formato variable. El formato de las diferentes instrucciones se explicarán más adelante.

A diferencia de la arquitectura DLX, caracterizada por emplear instrucciones sencillas, se ha optado por utilizar un juego de instrucciones de mayor complejidad, por lo que se requiere una unidad de control compleja para decodificarlas. En la figura 4.1 se puede observar el formato equivalente para una misma instrucción de los repertorios DLX y ARM.

Figura 4.1: Comparación de instrucciones DLX y ARM

El procesador implementado es capaz de ejecutar 3 tipos de instrucciones:

- Accesos a memoria
- Operaciones sobre registros
  a). Operaciones con dos registros
  b). Operaciones con un registro y un inmediato
- Operaciones de salto

A continuación se explican brevemente los diferentes tipos de instrucciones. Más adelante se expondrán las instrucciones con más detalle, explicando los campos de cada una.

4.3.1. Accesos a memoria

Las instrucciones de acceso a memoria son necesarias cuando se requiere cargar (load) un dato desde la memoria al banco de registros, o almacenar (store) el valor de un registro en la memoria.

Aunque es posible acceder a las direcciones de memoria direccionadas por media palabra. En esta implementación se está obligado a cargar valores de tamaño 4 bytes (tamaño de palabra), siendo por tanto recomendable utilizar direcciones de memoria que sean múltiplos de 4.

Para el cálculo de la dirección efectiva de carga o almacenamiento se ha implementado un único modo de direccionamiento. Registro base Rn + imm12", es decir, la dirección base se obtiene del registro Rn, y se suma un inmediato de 12 bits extraído de la instrucción.

4.3.2. Procesamiento de datos

Las instrucciones de procesamiento realizan cálculos aritméticos y lógicos. Se aplican sobre dos operandos y el resultado (si existe) se almacena en un registro.

Dependiendo de la instrucción los operandos pueden ser:

Figura 4.2: Formato para instrucciones aritmético-lógicas.

Operaciones con dos registros

Los datos con los que se trabaja se extraen de dos registros codificados en 4 bits.

Al utilizar el registro R15 se deben tener en cuenta ciertas restricciones. Este registro se utiliza para diferenciar ciertas operaciones de otras. Por ejemplo, si el código de operación es "0010", el registro origen Rn es R15 ("1111") entonces la operación ejecutada será la operación "MOVE". Si el registro Rn es cualquier otro, se ejecutará una "Ó lógica"(operación or).

Operaciones con un registro y un inmediato

El conjunto de operaciones con inmediato se limita a cuatro. Se permite mover un inmediato de 16 bits a un registro, pudiendo elegir si los dos bytes se almacenarán en los 16 bits más significativos o en los 16 bits menos sig-

nificativos. Además se permite sumar o restar un inmediato de 12 bits a un registro.

4.3.3. Operaciones de control

Las operaciones de control intervienen en la ejecución normal del programa. Se utilizan para modificar el valor del registro del contador de programa.

Los procesadores ARM combinan instrucciones de 32 bits con instrucciones de 16 bits. Por ello, el inmediato es desplazado un bit hacia la izquierda. En nuestro caso nos debemos asegurar de codificar las instrucciones con un 0 en el bit menos significativo del inmediato. Con esto se evita acabar en una dirección equivocada, y leer dos mitades de dos instrucciones distintas.

Existen dos tipos de instrucciones de salto. El primero es el salto incondicional y permite sumar un entero de 24 bits al valor del contador de programa y almacenar el resultado en el mismo.

La segunda operación de control es el salto condicional. Para este tipo de salto se reduce el tamaño del inmediato a 20 bits. Esto es debido a que campo extra para la condición de salto, el tamaño de este es de 4 bits.

Previamente a un salto condicional se debe ejecutar una operación de comparación. Esta operación activa los flags de la unidad aritmético-lógica dependiendo del resultado de la comparación, y estos se mantienen hasta que se vuelva a ejecutar otra comparación. Los flags se comparan a la condición de salto y en caso de coincidir, se efectúa el salto. Si no se ejecuta la comparación, el estado de los flags es desconocido y el procesador se comportará de manera no controlada.

Figura 4.3: Formato para instrucciones de control.

4.4. Arquitectura

4.4.1. Ruta de datos

En la figura 4.4 se muestra el diseño de la ruta de datos del procesador. La ruta de datos consta de 5 etapas que se explican a continuación.

Búsqueda de instrucción

La primera etapa es la encargada de cargar las instrucciones de memoria y transmitirlas a la siguiente, simultáneamente se calcula la dirección de la siguiente instrucción. Para realizar estas tareas los elementos utilizados son:

- El acceso a la memoria de instrucciones se realiza a través de un módulo que recibe la dirección de memoria y devuelve la instrucción a ejecutar.

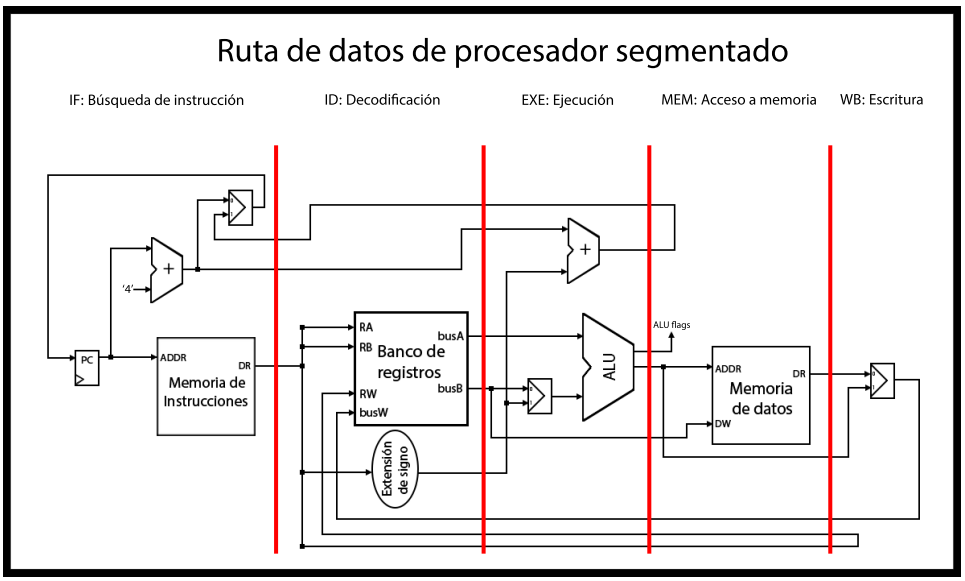

Figura 4.4: Ruta de datos DLX-ARM

Este módulo es una memoria ROM que contiene las instrucciones del programa en código binario.

- El contador de programa es el registro que almacena la dirección de memoria donde se localiza la instrucción.
- Un sumador encargado de incrementar en 4 el contador de programa.
- Un multiplexor encargado de seleccionar la siguiente dirección. En caso de haberse ejecutado un salto, se seleccionará la dirección calculada en la etapa de ejecución. En caso contrario se continúa la ejecución normal en la que la siguiente instrucción será la salida del sumador de la etapa.

Decodificación

En la etapa de decodificación se analiza la instrucción y se obtienen los datos necesarios para realizar las operaciones correctamente. Para decodificar las instrucciones se dispone de:

- Un banco de registros que contiene los registros que almacenan los datos con los que se trabaja.
- Un circuito combinacional de extensión de signo. Este circuito obtiene el inmediato codificado correspondiente a la instrucción que se está ejecutando.

Ejecución

En la etapa de ejecución se realizan los cálculos aritméticos o lógicos sobre los datos obtenidos del banco de registro y del circuito de extensión de signo. Para realizar los cálculos se incluye:

- Una Unidad Aritmético-Lógica (ALU) para las operaciones sobre los registros. Junto a la ALU aparece un multiplexor que permite seleccionar el origen de los datos.
- Un sumador para el cálculo de la dirección de salto.

Acceso a memoria

En la etapa de memoria se realizan intercambios de datos con la memoria principal.

- Memoria de datos. Es el módulo encargado de la interacción con los datos almacenados en memoria. Permite cargar los datos de memoria en los registros y volver a almacenarlos después de su utilización. Esta memoria está implementada como una memoria RAM de acceso directo de un ciclo.

Escritura en registros

En la etapa final del procesador se escriben los resultados calculados por la ALU, o los datos cargados de memoria en el banco de registros.

- Contiene un multiplexor para seleccionar el origen de los datos que se almacenarán en el registro.

4.4.2. Ruta de control

Para completar el procesador se ha incluido una unidad de control principal encargada de analizar la instrucción que debe ejecutarse y preparar las señales de control para el resto de módulos. En la figura 4.5 se observar el procesador con la unidad de control y el flujo de las señales de control. Seguidamente se enumeran y se explica la funcionalidad de las señales de control.

Búsqueda de instrucción

- *PCSrc.*

  La señal PCSrc indica si se cargará un salto o se continúa la ejecución normal del programa. Esta señal se deriva de una comparación,

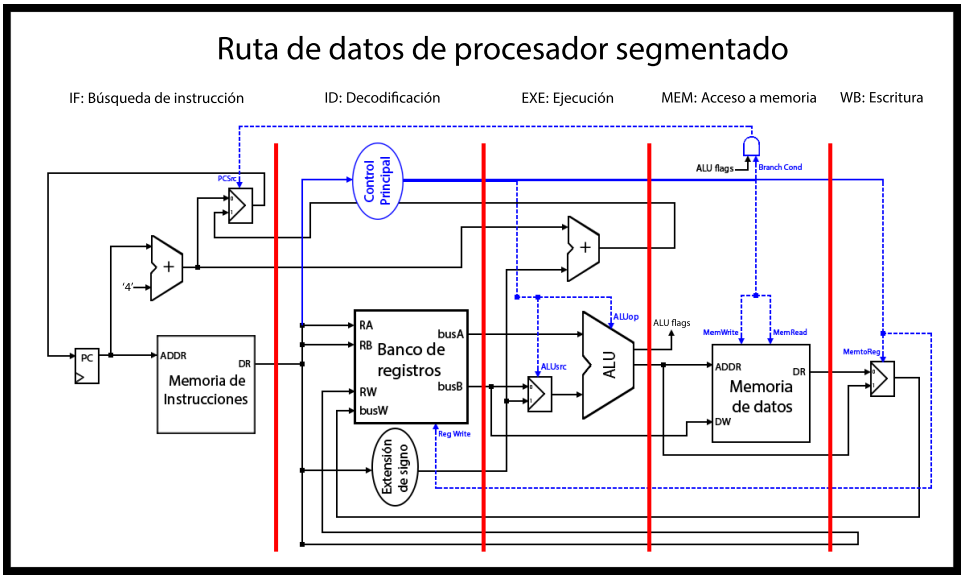

Figura 4.5: Ruta de control DLX-ARM

realizada en la etapa de memoria, entre las condiciones de salto de la instrucción y el valor de las flags calculadas por instrucciones anteriores de comparación.

Decodificación

- *RegWrite*.

  El banco de registros recibe esta señal que proviene de la etapa de escritura en registros.

Ejecución

- *ALUSrc*

  Selecciona el origen del segundo operando de entrada para la unidad aritmético-lógica.

- *ALUop*

  Selecciona la operación que se aplica en la unidad aritmético-lógica.

Acceso a memoria

- *MemWrite*[1].

  Indica que la instrucción debe acceder a memoria en modo escritura.

[1] Las señales MemWrite y MemRead no pueden valer 1 en la misma instrucción

- *MemRead*[1].

  Indica que la instrucción debe acceder a memoria en modo lectura.

- *BranchCond*

  Condición necesaria para activar la señal de control "PCSrc".

Escritura en registros

- *MemtoReg*

  Indica si el resultado de la instrucción tiene origen en la memoria de datos o en la unidad aritmético-lógica.

- *RegWrite*

  Indica si el resultado de la instrucción debe almacenarse en el banco de registros.

4.5. Formato de instrucciones

El juego de instrucciones implementado es un subconjunto de las instrucciones de la arquitectura Thumb-2. En este apartado se explican las instrucciones implementadas con sus campos. Para mayor información sobre el juego de instrucciones THUMB-2 véase el manual de referencias [1].

Organizado en 3 tipos, el juego de instrucciones se divide en instrucciones de transferencia, instrucciones de operaciones e instrucciones de control de flujo.

Las instrucciones implementadas, divididas por grupo, son:

4.5.1. Transferencia

Instrucciones de acceso a memoria, LOAD y STORE de un único dato con desplazamiento.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

[1] R. Brinkgreve, W. Swolfs, and E. Engin. *ARM Architecture Reference Manual Thumb-2 Supplement.* 2011.

[2] S. Brown and J. Rose. Architecture of FPGAs and CPLDs: A tutorial. *IEEE Design and Test of Computers*, 13(2):42–57, 1996.

[3] C. T. Bustillos. Simulador arm en el ámbito docente. 2012.

[4] I. N. de Estadistica. Penetración de ordenador en hogares. 2014.

[5] S. Flash. Nexys4 FPGA Board Reference Manual Ethernet connector. pages 1–29, 2013.

[6] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC V8 architecture. *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 409–415, 2002.

[7] J. C. González Salas. *Filtro adaptativo tolerante a fallos.* PhD thesis, 2014.

[8] S. Habinc. Functional Triple Modular Redundancy (FTMR). *Design and Assessment Report, Gaisler Research*, pages 1–56, 2002.

[9] J. L. Hennessy and D. A. Patterson. *Arquitectura de Computadores: Un enfoque cuantitativo.* Mcgraw Hill Editorial, 1993.

[10] J. L. Hennessy and D. a. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach.* Number 0. 2006.

[11] A. C. Hu and S. Zain. NSEU Mitigation in Avionics Applications. 1073:1–12, 2010.

[12] O. Ieee-std. LEON3 7-Stage Integer Pipeline. (March), 2010.

[13] A. O. Investigation. ATSB TRANSPORT SAFETY REPORT Aviation Occurrence Investigation AO-2008-070 Final. (October), 2008.

[14] Jedec. Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray Induced Soft Error in Semiconductor Devices: JESD89A. *JEDEC Sold State Technology Association*, pages 1–85, 2006.

[15] A. Kadav, M. J. Renzelmann, and M. M. Swift. Fine-grained fault tolerance using device checkpoints. *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems - ASPLOS '13*, page 473, 2013.

[16] H. Kirrmann. Fault Tolerant Computing in Industrial Automation. *Lecture notes ABB Corporate ResearchETH*, 2005.

[17] I. Kuon, R. Tessier, and J. Rose. FPGA Architecture: Survey and Challenges. *Foundations and TrendsÂ® in Electronic Design Automation*, 2(2):135–253, 2007.

[18] A. R. M. Limited. ARM7TDMI-S. (Rev 3), 2000.

[19] a. R. M. Limited. ARM Architecture Reference Manual. pages 1–1138, 2007.

[20] W. K. Melis. *Reconstruction of High-energy Neutrino-induced Particle Showers in KM3NeT .* PhD thesis, 2014.

[21] C. Mobile. Streaming 4K Ultra HD video at home and on the go. pages 0–1.

[22] J. Rose, A. E. Gamal, and A. Sangiovanni-Vincentelli. Architecture of Field-Programmable Gate Arrays.

[23] E. Rotenberg. AR-SMT: a microarchitectural approach to fault tolerance in microprocessors. *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352)*, 1999.

[24] D. J. Sorin and S. Ozev. Fault Tolerant Microprocessors for Space Missions. *Memory*, pages 1–4.

[25] U. States. Reduce Cost and Board Space. 374:1–8, 2011.

[26] I. S. Summary, T. C. Field, M. Long, S. D. Transfer, U. Instruction, and I. S. Examples. ARM Instruction Set. pages 1–60.

[27] J. M. Torrecillas. RAID - Tolerancia a Fallos.

[28] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. *Proceedings of the International Conference on Dependable Systems and Networks*, (July):411–420, 2001.

[29] Xilinx. Xilinx Artix-7 Fpgas: a New Performance Standard for Power-Limited, Cost-Sensitive Markets.