Diseño y análisis de un procesador tolerante a fallos transitorios compatible con ARM a nivel de instrucciones

TRABAJO FIN DE GRADO

Andrés Gamboa Meléndez

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Documento maquetado con TEXIS v.1.0.

Este documento está preparado para ser imprimido a doble cara.

Diseño y análisis de un procesador tolerante a fallos transitorios compatible con ARM a nivel de instrucciones

Trabajo fin de grado
Grado en Ingeniería de Computadores

Versión 1.0

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Al duque de Béjar
y
a tí, lector carísimo

I can't go to a restaurant and order food because I keep looking at the fonts on the menu.
Donald Knuth

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Groucho Marx decía que encontraba a la televisión muy educativa porque cada vez que alguien la encendía, él se iba a otra habitación a leer un libro. Utilizando un esquema similar, nosotros queremos agradecer al Word de Microsoft el habernos forzado a utilizar LaTeX. Cualquiera que haya intentado escribir un documento de más de 150 páginas con esta aplicación entenderá a qué nos referimos. Y lo decimos porque nuestra andadura con LaTeX comenzó, precisamente, después de escribir un documento de algo más de 200 páginas. Una vez terminado decidimos que nunca más pasaríamos por ahí. Y entonces caímos en LaTeX.

Es muy posible que hubíeramos llegado al mismo sitio de todas formas, ya que en el mundo académico a la hora de escribir artículos y contribuciones a congresos lo más extendido es LaTeX. Sin embargo, también es cierto que cuando intentas escribir un documento grande en LaTeX por tu cuenta y riesgo sin un enlace del tipo "*Author instructions*", se hace cuesta arriba, pues uno no sabe por donde empezar.

Y ahí es donde debemos agradecer tanto a Pablo Gervás como a Miguel Palomino su ayuda. El primero nos ofreció el código fuente de una programación docente que había hecho unos años atrás y que nos sirvió de inspiración (por ejemplo, el fichero `guionado.tex` de TeXiS tiene una estructura casi exacta a la suya e incluso puede que el nombre sea el mismo). El segundo nos dejó husmear en el código fuente de su propia tesis donde, además de otras cosas más interesantes pero menos curiosas, descubrimos que aún hay gente que escribe los acentos españoles con el `\'{\i}`.

No podemos tampoco olvidar a los numerosos autores de los libros y tutoriales de LaTeX que no sólo permiten descargar esos manuales sin coste adicional, sino que también dejan disponible el código fuente. Estamos pensando en Tobias Oetiker, Hubert Partl, Irene Hyna y Elisabeth Schlegl, autores del famoso "The Not So Short Introduction to LaTeX 2_ε" y en Tomás

Bautista, autor de la traducción al español. De ellos es, entre otras muchas cosas, el entorno `example` utilizado en algunos momentos en este manual.

También estamos en deuda con Joaquín Ataz López, autor del libro "Creación de ficheros LaTeX con GNU Emacs". Gracias a él dejamos de lado a WinEdt y a Kile, los editores que por entonces utilizábamos en entornos Windows y Linux respectivamente, y nos pasamos a emacs. El tiempo de escritura que nos ahorramos por no mover las manos del teclado para desplazar el cursor o por no tener que escribir `\emph` una y otra vez se lo debemos a él; nuestro ocio y vida social se lo agradecen.

Por último, gracias a toda esa gente creadora de manuales, tutoriales, documentación de paquetes o respuestas en foros que hemos utilizado y seguiremos utilizando en nuestro quehacer como usuarios de LaTeX. Sabéis un montón.

Y para terminar, a Donal Knuth, Leslie Lamport y todos los que hacen y han hecho posible que hoy puedas estar leyendo estas líneas.

Resumen

...

...

...

Capítulo 1

Introducción

...

...

Resumen: En este capítulo se realiza una introducción al trabajo realizado durante el proyecto. Se plantea el problema, se enumeran los objetivos del trabajo y se define la estructura de este documento.

1.1. Introducción

Esta monografía es el resultado del estudio e investigación realizados para la asignatura «Trabajo de fin de grado» del Grado en Ingeniería de Computadoresque se ha llevado a cabo en el departamento de «Arquitectura de Computadores y Automática» de la Universidad Complutense de Madrid, bajo la dirección del (Doctor), D. Jose Miguel Montañana Aliaga .

El trabajo se centra en la implementación de un microprocesador tolerante a fallos y con un diseño que le permita ser compatible con las instrucciones ARM del repertorio del microprocesador «ARM Cortex M3» Sadasivan (2006). La tolerancia a fallos aplicada ha sido el «modelo de replicado triple de módulos(TMR)» Habinc (2002).

1.2. Motivación

Hoy en día, el uso de la tecnología y la informática se extienden a nivel mundial y es de aplicación a cada vez, un mayor número de campos. La tecnología cada vez está más presente en nuestras vidas, ya no se concibe un hogar o puesto de trabajo sin un ordenador sobre la mesa. El uso de los computadores con carácter personal cada año es más común como podemos

ver en la figura 1.1. Las estadísticas, realizadas por el Instituto Nacional de Estadística (INE), muestran que en España más del 95 % de los hogares posee al menos un teléfono móvil, normalmente teléfonos inteligentes, y más del 70 % posee un ordenador personal, lo que es un indicativo de la necesidad tecnológica que existe en estos tiempos.

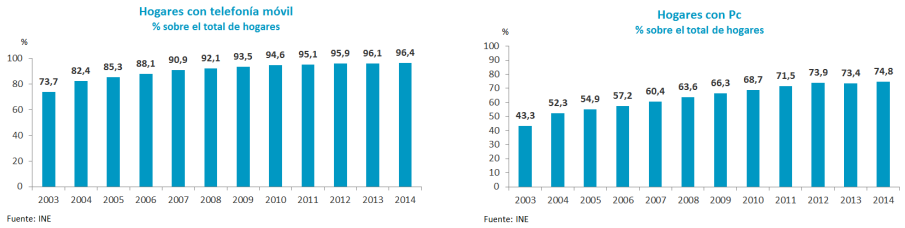

Figura 1.1: Estadísticas de uso en hogares españoles.

Todos los sistemas son susceptibles de sufrir fallos y errores a varios niveles (se explican en la sección 2.2), errores que provocan comportamientos erráticos y no deseados.

En sistemas personales como pueden ser los teléfonos inteligentes y los ordenadores personales, los fallos que sufren, normalmente no provocan consecuencias serias. Pero estos sistemas son solo una pequeña parte de los que existen y que se utilizan día a día en todo el mundo.

En los campos de medicina y transporte también hay multitud de dispositivos diferentes basados en microelectrónica. Sistemas biomédicos que asisten a la vida de una persona, que controlan un marca pasos o asisten a la respiración, si sufren un error pueden provocar consecuencias irremediables. Los sistemas de transporte controlados por sistemas electrónicos pueden sufrir los mismos tipos de fallos y causar catástrofes

La importancia de asegurar estos dispositivos frente a posibles fallos se deriva de su control sobre sistemas vitales y por ser responsables de la vida de seres humanos. Si uno de estos sistemas falla se pone en riesgo la vida de una o muchas personas.

En el campo aeroespacial, los sistemas pueden no tener incidencia directa sobre pérdidas humanas como consecuencia de sus fallos, pero pueden causar la pérdida del sistema completo con un coste económico muy elevado.

Las radiaciones cósmicas puede provocar fallos en cualquier sistema electrónico, dañando el mismo permanente o temporalmente, así, los satélites que orbitan alrededor de la tierra y los aviones que se mueven a una gran altura deben ser mucho más robustos que los sistemas que trabajan a nivel del suelo. Como podemos observar en las gráficas 1.2, los fallos se producen más frecuentemente a mayor altitud. Esto es causado por tener una menor protección frente a la mayor radiación y los rayos cósmicos presentes en el espacio.

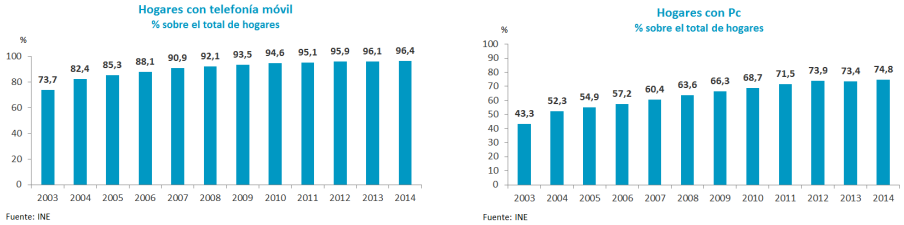

Figura 1.2: Estadísticas de fallos según la altitud.

Cada vez los sistemas electrónicos contienen transistores de menor tamaño, lo que disminuye su tolerancia a los «ruidos de transmisión» haciéndolos más sensibles a la radiación y a los rayos cósmicos que inducen fallos a los sistemas.

Lo que tienen en común los dispositivos mencionados, y muchos otros presentes en nuestra vida cotidiana, es que son sistemas con componentes micro-electrónicos. Poseen un microprocesador que es el cerebro y responsable de dirigir el sistema ejecutando los programas y como tal sistema electrónico, susceptible de sufrir fallos. Fallo que en este componente puede causar graves consecuencias en el resto del sistema.

Lo que tienen en común los dispositivos mencionados, y muchos otros presentes en nuestra vida cotidiana, es que son sistemas con componentes micro-electrónicos. Poseen un microprocesador que es el cerebro y responsable de dirigir el sistema ejecutando los programas y como tal sistema electrónico, susceptible de sufrir fallos. Fallo que en este componente puede causar graves consecuencias en el resto del sistema.

Para garantizar el correcto funcionamiento de estos sistemas existen múltiples técnicas de tolerancia a fallos, técnicas que ayudan a detectar los fallos y a recuperar el sistema antes de que causen errores. Aplicando una o varias de estas técnicas se ofrece un sistema robusto y capaz de funcionar ante los fallos inducidos por la radiación u otros agentes externos.

1.3. Planteamiento del problema

Este trabajo tiene en cuenta que el motor principal de muchos sistemas es el microprocesador. El procesador de un sistema es su *cerebro*, más concretamente es el encargado de ejecutar las instrucciones que componen los programas.

Si no se toman medidas de prevención y tolerancia, este *cerebro* puede ver alterado su comportamiento por efectos externos, provocando errores de ejecución del programa. Estos errores a su vez pueden ser causa de un comportamiento no deseado alterando los datos, modificando el funcionamiento del propio procesador o de otros componentes del sistema como las memorias

o los controladores entrada/salida.

Con este trabajo se pretende ofrecer un medio para evitar estas situaciones concediendo un grado extra de fiabilidad a los sistemas basados en microprocesadores. Para ello se quiere diseñar e implementar un microprocesador sencillo capaz de ejecutar un conjunto reducido de instrucciones (RISC), al que posteriormente se le aplicarán las técnicas de tolerancia a fallos, aumentando así su supervivencia mediante un incremento en su capacidad de detectar e incluso recuperarse de los fallos.

1.4. Objetivos

Por lo comentado en el apartado anterior se ha decidido que este trabajo esté dedicado a diseñar un procesador con un grado de fiabilidad mayor que un procesador convencional.

El proyecto se ha divido en cuatro tareas dedicadas a la implementación del microprocesador y a la aplicación de la tolerancia a fallos.

1. Primera. Implementación del procesador.

   Se ha implementado el procesador segmentado en 5 etapas. Para ello se ha partido de la arquitectura DLX vista en las asignaturas de computadores de nuestro grado.

2. Segunda. Ruta de control

   Se ha rediseñado la ruta de control y parte de la ruta de datos. El nuevo juego de instrucciones usado, completamente distinto al que utiliza un DLX convencional, obliga a cambiar la ruta de control. Simulando unos pequeños programas se comprueba que el procesador es capaz de decodificar y ejecutar las nuevas instrucciones.

3. Tercera. Diseño de tolerancia a fallos.

   Una vez implementado el procesador completo y comprobado su funcionamiento se diseña y se incorpora la tolerancia a fallos. Para ello se triplican los módulos que pueden causar mayor número de fallos y se insertan votadores de mayoría.

4. Cuarta. Diseño del sistema de inserción de fallos.

   Para finalizar se ha diseñado un sistema externo de inserción de fallos. Este sistema es capaz de alterar los valores de las salidas de los módulos triplicados, para comprobar después como afecta esto al funcionamiento del procesador.

1.5. Estructura del documento

Capitulo 1 *Introducción*:

En el capitulo 1 se realiza la introducción al trabajo propuesto y realizado para el trabajo de fin de grado en el que se basa el presente documento.

Capitulo ?? *Trabajo relacionado*:

En el capitulo ?? se realiza una exposición de la investigación llevada a cabo para realizar el trabajo. Se exponen definiciones y otros datos de interés para el lector.

Capitulo ?? *Procesador*:

En el capitulo ?? se describe el procesador implementado con su estructura y arquitectura.

Capitulo ?? *Proporcionando tolerancia a fallos*:

En el capitulo ?? se describe cómo se ha proporcionado la tolerancia a fallos y qué técnicas se han utilizado.

Capitulo ?? *Resultados*:

En el capitulo ?? se muestran los resultados obtenidos de las simulaciones realizadas.

Capitulo ?? *Análisis de los resultados*:

En el capitulo ?? se analizan los datos.

Capitulo ?? *Conclusiones*:

En el capitulo ?? se describen las conclusiones tras analizar los resultados.

Capítulo 2

Titulo por definir

«La verdadera ciencia enseña, sobre todo, a dudar y a ser ignorante.»

Ernest Rutherford

Resumen: En este capítulo se define con detalle lo que es un procesador y su importancia en el mundo hoy en día. También se tratan dos arquitectura más concretas, la arquitectura DLX y la arquitectura ARM.

A continuación se define qué es un fallo y qué tipos de fallos pueden ocurrir en los sistemas. Además se explican algunas técnicas de tolerancia a fallos.

Para terminar se justifica la importancia de la tolerancia en los sistemas y concretamente porque es necesaria la tolerancia en los microprocesadores.

Introducción

...

2.1. Procesador

El Diccionario de la Real Academia Española (DRAE) define el procesador como la «unidad central de proceso (CPU), formada por uno o dos chips». Figura 2.1.

La CPU es el circuito integrado encargado de acceder a las instrucciones de los programas informáticos y ejecutarlas. Para poder ejecutar un programa, el procesador debe realizar las siguientes tareas:

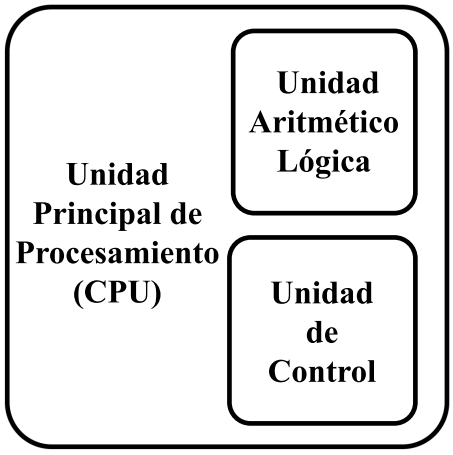

Figura 2.1: Procesador DRAE

1. Acceder a las instrucciones almacenadas en memoria.
2. Analizar las instrucciones y establecer las señales de control internas.
3. Ejecutar operaciones sobre datos.
4. Almacenar los resultados en memoria.

A continuación se definen los elementos fundamentales para definir un procesador.

2.1.1. Arquitectura

Un procesador está formado por una serie de módulos conectados entre sí, siendo la arquitectura del mismo la que define el diseño de los módulos que lo componen y de qué manera se conectan entre ellos.

La arquitectura del procesador diseñada por Von Neumann separa los componentes del procesador en módulos básicos. La CPU es el verdadero núcleo de los computadores, donde se realizan las funciones de computación y control, y contiene todos los componentes la memoria y los elementos de entrada y salida [Hennessy y Patterson (1993)].

Según el juego de instrucciones que sea capaz de ejecutar un procesador, su arquitectura puede clasificarse como:

1. *Reduced instruction set computer (RISC).* Utiliza un repertorio de instrucciones reducido, con instrucciones de tamaño fijo y poca variedad en su formato.
2. *Complex instruction set computer (CISC).* Utiliza un repertorio de instrucciones muy amplio, permite realizar operaciones complejas tales

como realizar cálculos entre los datos en memoria y los datos en registro.

3. *Simple instruction set computer (SISC)*. Utiliza un repertorio de instrucciones enfocado al procesamiento paralelo.

2.1.2. Repertorio de instrucciones

El repertorio de instrucciones define todas las operaciones que el procesador es capaz de entender y ejecutar. Este juego de instrucciones incluye las operaciones aritmético-lógicas que puede aplicar a los datos, las operaciones de control sobre el flujo del programa, las instrucciones de lectura y escritura en memoria, así como todas las instrucciones propias que se hayan diseñado para el procesador.

2.1.3. Memoria

Los procesadores tienen una serie de registros donde se almacenan temporalmente los valores con los que está trabajando. El conjunto de estos registros se conoce como «banco de registros». Estos registros de propósito general son muy limitados. Por ello el procesador necesita de apoyo externo para alojar la información, para lo que tiene acceso a una memoria externa.

El acceso a la memoria externa divide las arquitecturas en dos tipos. La arquitecturas Von Neumann utiliza una única memoria para almacenar tanto los datos como las instrucciones. Las arquitecturas Harvard, sin embargo, separan la memoria de datos de la memoria de instrucciones. Figura 2.2.

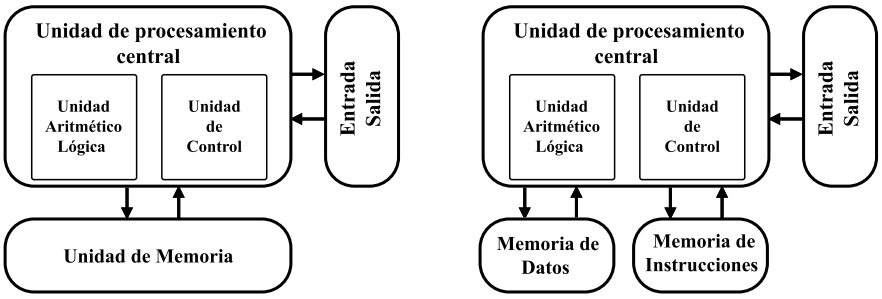

Figura 2.2: Arquitectura Von Neumann y Arquitectura Harvard

2.1.4. Segmentación

La segmentación es una técnica de implementación, que no siendo imprescindible aumenta el rendimiento del procesador. Permite que haya varias

instrucciones en ejecución al mismo tiempo en el mismo procesador. El procesador es dividido en etapas y en cada una de ellas se realiza una parte del trabajo completo de la instrucción de forma secuencial.

La segmentación permite que en cada ciclo de reloj se busque una instrucción y se comience su ejecución, con ello se consigue reducir el número de ciclos total que necesita el programa.

En la tabla 2.1 podemos ver cómo se lanzan una serie de instrucciones. Se observa cómo las instrucciones ocupan unicamente una etapa del procesador, y cómo avanzan por el procesador dejando libre la etapa anterior para la siguiente instrucción.

	Ciclo de reloj								
Número de instrucción	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i + 1		IF	ID	EX	MEM	WB			
i + 2			IF	ID	EX	MEM	WB		
i + 3				IF	ID	EX	MEM	WB	
i + 4					IF	ID	EX	MEM	WB

Tabla 2.1: Segmentacion simple de 5 etapas

Ventajas de la segmentación

La segmentación proporciona la ventaja de poder lanzar una instrucción por cada ciclo de reloj. Esta característica aumenta el rendimiento del procesador al obtener un menor número total de ciclos por instrucción para un mismo programa. Para conocer los ciclos por instrucción que necesita un programa se utiliza la formula.

$$\text{Ciclos por instrucción (CPI)} = \frac{\text{Número de ciclos total}}{\text{Número de instrucciones}} \tag{2.1}$$

A modo de ejemplo, veamos que sucede al ejecutar un programa de 3 instrucciones sobre un procesador que tarde 5 ciclos de reloj en ejecutar cualquier instrucción, pero en un caso no segmentado, y en otro caso segmentado en 5 etapas de 1 ciclo cada una.

Como podemos ver en la figura 2.3, el procesador no segmentado tarda 15 ciclos en ejecutar las 3 instrucciones y utilizando la formula anterior se obtiene un valor de CPI es 5. Al ejecutar el mismo programa en el procesador segmentado, este tarda 5 ciclos en llenar las 5 etapas del procesador. A partir de ahora cada ciclo de reloj termina una instrucción, completandose la ejecución del programa en 7 ciclos de reloj. El nuevo valor de CPI es de 2,33. Así pues, la segmentación ha reducido el número de ciclos por instrucción de este programa a menos de la mitad.

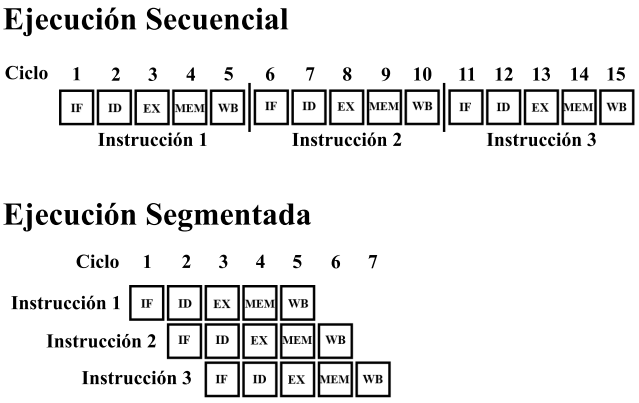

Figura 2.3: Ejecución secuencial comparada con ejecución segmentada

Riesgos de la segmentación

Además de las ventajas vistas en el apartado anterior, la segmentación también implica unos riesgos a la hora de ejecutar las instrucciones. Estos riesgos implican que las instrucciones deban esperar un número de ciclos para poder continuar su ejecución, retrasando la entrada de instrucciones en el microprocesador. Estos riesgos pueden ser de los siguientes tipos[Hennessy y Patterson (1993)]:

1. *Riesgos estructurales.* Surgen cuando 2 o más instrucciones necesitan acceder a los mismos recursos.

2. *Riesgos de datos.* Surgen cuando una instrucción depende del resultado de una instrucción anterior, y este todavía no se ha escrito en el registro correspondiente. A su vez pueden ser:

   - *Lectura después de escritura (RAW).* Una instrucción intenta leer un dato antes de que se escriba en el registro.
   - *Escritura después de lectura (WAR).* La *instrucción i+1* escribe el resultado en el registro antes de que la *instrucción i* haya leído el dato del mismo registro. Esto solo ocurre con instrucciones que realicen una escritura anticipada como por ejemplo instrucciones de auto-incremento de direccionamiento.
   - *Escritura después de escritura (WAW).* Ocurre cuando las escrituras se realizan en orden incorrecto. Por ejemplo la *instrucción i+1* escribe su resultado antes de que lo haga la *instrucción i*, ambas escriben en el mismo registro.

- *Lectura después de lectura (RAR).* Realmente no es un riesgo como tal, ya que no se modifica ningún dato.

3. *Riesgos de control.* Surgen a consecuencia de las instrucciones que afectan al registro del contador de programa (PC).

2.1.5. DLX

El microprocesador DLX fue diseñado por John Hennessy y David A. Patterson, diseñadores de las arquitecturas MIPS y Berkeley RISC respectivamente. Es un procesador sencillo con arquitectura RISC y proporciona una base fácil de comprender. Se utiliza ampliamente en educación universitaria para explicar las arquitecturas de computadores [Pascual (2011)].

Basado en las máquinas de carga/almacenamiento, el DLX se centra en proporcionar[Hennessy y Patterson (1993)]:

- Un sencillo repertorio de instrucciones de carga/almacenamiento.
- Un diseño de segmentación eficiente.
- Un repertorio de instrucciones fácil de descodificar.

Arquitectura RISC

El microprocesador DLX utiliza una arquitectura RISC con instrucciones de 32 bits. Posee un banco de registros compuesto por 32 registros de propósito general, además de un segundo conjunto de registros que se pueden usar como 32 registros de simple precisión o como 16 registros en punto flotante.

Instrucciones DLX

Todas las instrucciones del repertorio del procesador DLX tienen un tamaño de 32 bits y están alineadas en memoria. En cualquier instrucción los bits [31:26] forman el campo de código de operación que se debe ejecutar.

Se dividen en tres tipos según su formato [Arnau Llombart]:

- *Tipo R.* Instrucciones aritmético-lógicas.
- *Tipo I.* Instrucciones de transferencia.
- *Tipo J.* Instrucciones de bifurcación.

Como podemos observar en la figura 2.4, el formato es muy similar en los tres tipos, lo que reduce la ruta de datos, simplificando su implementación.

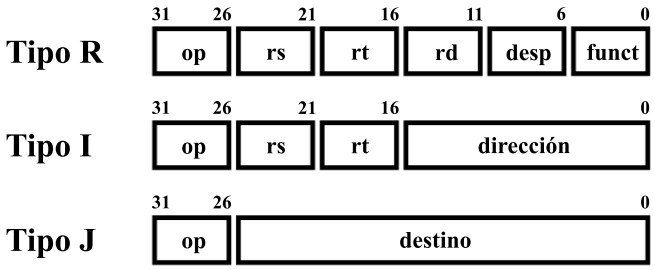

Figura 2.4: Formato de instrucciones DLX

Segmentación DLX

El DLX basa su rendimiento en la segmentación y se divide en 5 etapas

- *Búsqueda de la instrucción (IF)*

  Esta primera etapa es la encargada de acceder a memoria y traer la siguiente instrucción.

- *Descodificación de la instrucción (ID)*

  En la segunda etapa se descodifica la instrucción cargada en la primera etapa, obteniendo las señales de control. Extrae los operandos del banco de registro o de la propia instrucción.

- *Ejecución y cálculo de direcciones efectivas (EX)*

  La tercera etapa se encarga de ejecutar la instrucción utilizando las unidades funcionales. Las unidades funcionales pueden estar segmentadas y/o duplicadas.

- *Acceso a memoria (MEM)*

  En la etapa de memoria es cuando se ejecutan las operaciones de carga y almacenamiento. Las instrucciones de carga traen datos de la memoria y los almacenan en los registros, mientras que las instrucciones de almacenamiento guardan los datos en memoria.

- *Postescritura (WB)*

  En la última etapa se almacenan los resultados de las instrucciones en los registros.

Como pudimos ver en la figura 2.1, las instrucciones se buscan en cada ciclo de reloj, a menos que surjan riesgos debido a la segmentación. Como

vimos en el apartado 2.1.4, la segmentación implica ciertos riesgos. Para solucionar o reducir estos, el DLX implementa las siguientes técnicas[Hennessy y Patterson (2006)]:

1. Duplicar y/o segmentar las unidades funcionales. Con ello se reducen los ciclos de espera debidos a los *riesgos estructurale*s. Se consigue un mayor número de etapas para poder cargar nuevas instrucciones.

2. «Adelantamiento»(forwarding) o «Cortocircuito». Técnica encaminada a resolver los *riesgos de datos*. Se consigue proporcionar un acceso a los resultados de instrucciones previas que todavía no han almacenado los datos en el «banco de registros».

    - *Lectura después de escritura (RAW).* Debido al cortocircuito implementado, el dato es recibido de las etapas siguientes y no es necesario que se haya escrito en los registros.
    - *Escritura después de lectura (WAR).* No puede ocurrir debido a que todas las lecturas se realizan al comienzo de la ejecución, en la etapa de descodificación, y las escrituras al final, en la etapa de postescritura.
    - *Escritura después de escritura (WAW).* Solo se presenta en segmentaciones que escriben en más de una etapa, esta arquitectura no se ve afectada ya que solo escribe en la etapa de postescritura. puede ocurrir de

3. *Riesgos de control.* Si se ejecuta una instrucción de salto, el cambio no se ve reflejado hasta la fase de memoria, esto implica una detención de 3 ciclos. En DLX se utiliza lógica especializada para averiguar si el salto es efectivo y para la calcular la dirección destino de salto en la etapa de descodificación.

Cuando existe un riesgo que no es posible evitar con estas técnicas se aplica un interbloqueo de la segmentación. Esta técnica detecta un riesgo y detiene la ejecución de la instrucción hasta que el riesgo desaparece. El bloqueo se realiza en la fase de decodificación, donde es posible determinar si existe algún riesgo.

Memoria DLX

Todos las referencias a memoria se realizan a través de instrucciones de carga y almacenamiento, cargando los datos en los registros, donde se pueden acceder y trabajar con ellos, y almacenándolos en memoria.

El acceso a memoria es direccionable por bytes en el modo «Big endian» con una dirección de 32 bits almacenada previamente en un registro. Los accesos pueden realizarse a un byte, media palabra o una palabra completa.

Además se puede acceder a palabras en doble precisión para almacenarlas en los registros de punto flotante.

2.1.6. ARM

...

Arquitectura ARM

...

Repertorio de instrucciones ARM

...

Segmentación ARM

...

Memoria ARM

...

2.2. Fallos

En un sistema electrónico pueden ocurrir una gran variedad de fallos, se clasifican en fallos de software y fallos de hardware, y se pueden encontrar, desde fallos en la definición de requisitos que se propagan hasta la fase de producción, hasta fallos producidos en el sistema por agentes externos como la radiación.

En esta sección se tratará esta última clase de fallos, los fallos producidos por agentes externos que no se pueden evitar en las fases de diseño. Y que afectan al hardware, dañando sus componentes o alterando los valores de las señales con las que se trabaja.

Los fallos analizados en esta sección se van a dividir en dos categorías: fallos permanentes y fallos transitorios.

2.2.1. Fallos Permanentes

Los fallos permanentes son aquellos que afectan al sistema de forma irreversible. Producen cambios en el diseño que estropean el correcto funcionamiento del módulo o circuito que lo sufre. Estos fallos no se solucionan reiniciando el sistema. [Jedec (2006)]

critical charge:

hard error:
single-event functional interrupt (SEFI):
single-event latch-up (SEL):
single event transient (SET):
single-event upset (SEU):
single-event upset (SEU) rate:
soft error, device:
soft error rate (SER):

2.2.2. Fallos Transitorios

...

2.3. Tolerancia a Fallos

La tolerancia a fallos se define como la capacidad de un sistema de funcionar correctamente incluso si se produce un fallo o anomalía en el sistema

Existen dos tipos de tolerancia; tolerancia estática y tolerancia dinámica.

2.3.1. Tolerancia estática

...

2.3.2. Tolerancia en dinámica

...

2.3.3. Tolerancia en microprocesadores

...

Capítulo 3

Procesador

...

...

Resumen: ...

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 Hennessy y Patterson (2006). 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.

3.1. 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.

- La memoria es accesible por palabras de 32 bits. Es decir, todo acceso a memoria carga o almacena 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.

- Las instrucciones se componen de 32 bits con formato variable.

3.2. 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 de 32 bits donde todos los bits cargados tienen valor.

3.3. 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 3.1 se puede observar el formato equivalente para una misma instrucción de los repertorios DLX y ARM.

Figura 3.1: 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.

3.3.1. 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.

3.3.2. 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 3.2: 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.

3.3.3. 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 con el desplazamiento 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 3.3: Formato para instrucciones de control.

3.4. Arquitectura

3.4.1. Ruta de datos

En la figura 3.4 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:

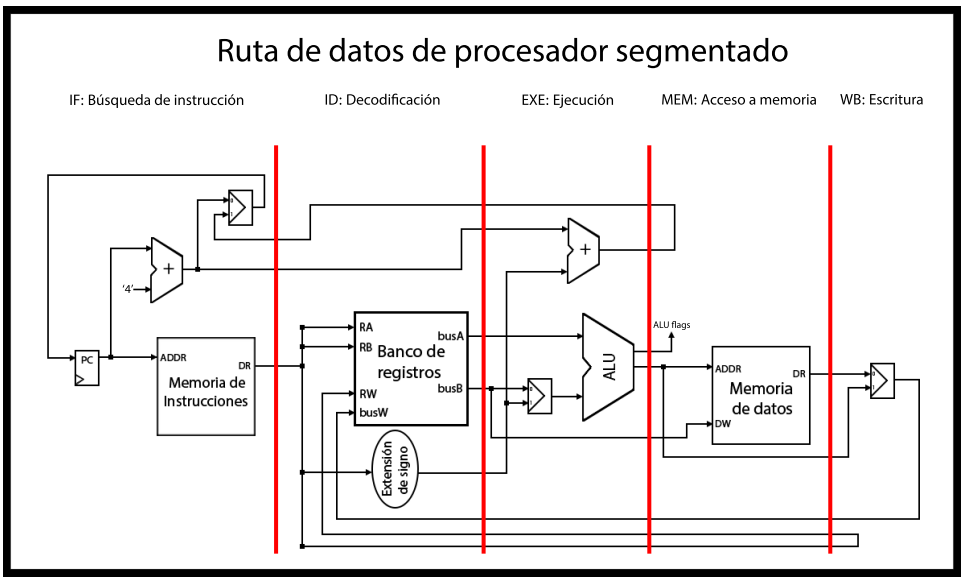

Figura 3.4: Ruta de datos DLX-ARM

- 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. 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.

3.4.2. 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 3.5 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,

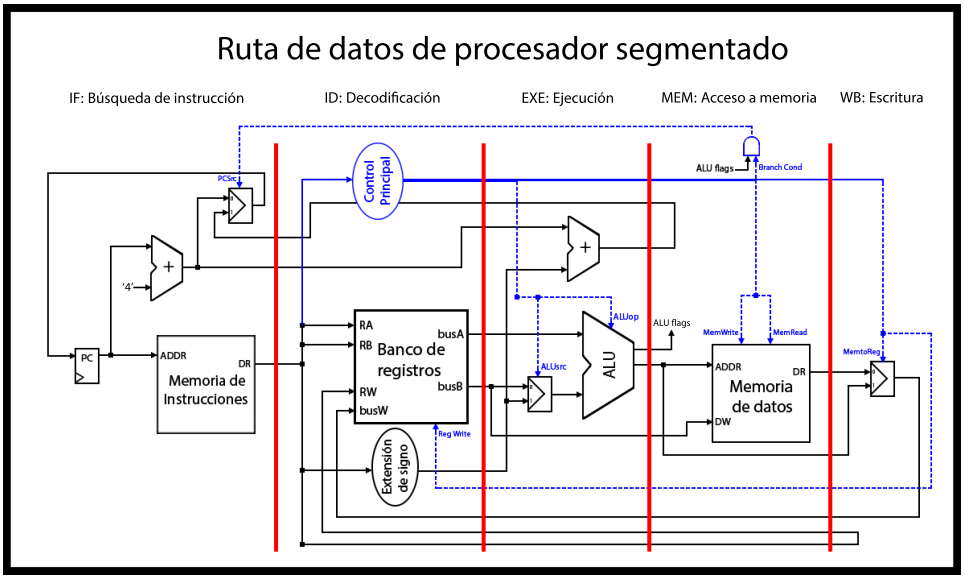

Figura 3.5: Ruta de control DLX-ARM

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.

3.5. 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 Brinkgreve et al. (2011).

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:

3.5.1. Transferencia

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

Apéndice A

Así se hizo...

...

...

Resumen: ...

A.1. Introducción

...

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

Arnau Llombart, V. Manual DLX. ????

Brinkgreve, R., Swolfs, W. y Engin, E. *ARM Architecture Reference Manual Thumb-2 Supplement*. 2011. ISBN 9781597180948.

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

Hennessy, J. L. y Patterson, D. A. *Arquitectura de Computadores: Un enfoque cuantitativo*. Mcgraw Hill Editorial, 1993. ISBN 1558600698.

Hennessy, J. L. y Patterson, D. a. *Computer Architecture, Fourth Edition: A Quantitative Approach*. 0. 2006. ISBN 0123704901.

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

Pascual, J. M. *Simulador DLX con repertorio multimedia*. Tesis Doctoral, Universidad Complutense de Madrid, 2011.

Sadasivan, S. An introduction to the arm cortex-m3 processor. 2006.

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes