Trabajo Propio ARM vs RISC

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

Objetivo: El proyecto trata de la implementación de un procesador VHDL segmentado, compatible a nivel de instrucciones con un ARM 32 bits. La tolerancia a fallos transitorios, se basará en la detección de fallos mediante la replicación triple de módulos, y posterior recuperación del fallo mediante:

1. El modulo triplicado permite conocer el valor correcto de la señal
2. Volvemos a ejecutar las instrucciones (por ejemplo todos los módulos discrepan)
3. Reiniciamos el sistema (por ejemplo error en el registro PC)

El principal objetivo del proyecto es proporcionar el procesador tolerante a fallos, frente a las prestaciones del procesador. La planificación del proyecto tiene estrictamente que dedicarle como mínimo un tercio del tiempo a la recolección de resultados, por lo que el nivel de segmentación, y el conjunto de instrucciones implementadas vendrá limitado por el tiempo disponible.

Procesador

INTRODUCCION PROCESADORES

Se eligió implementar un procesador ARM por ser uno de los procesadores más ampliamente utilizados.

Los procesadores ARM se pueden ver en en todo tipo de mercados, son el corazón de la mayoría de telefonos móviles y tablets actuales, en videoconsolas y routers de comunicación. Además se pueden encontrar en una gran variedad de sistemas empotrados, por ejemplo en los hogares podemos encontrarlos dentro de electrodomésticos como las neveras, las lavadoras y los lavaplatos, en los televisores y en los robots aspiradoras.

Se pueden encontrar en coches, que cada vez son más autónomos.

Y dentro de juguetes como LEGO mindstorm.

Los procesadores ARM son conocidos por su gestión de energía

INTRODUCCION INSTRUCCIONES

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344c/DDI0344C\_cortex\_a8\_r2p0\_trm.pdf

# 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. 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 un subconjunto del juego de instrucciones THUMB-2. Este juego de instrucciones es utilizado principalmente por los procesadores de la gama ARM CORTEX M. Otras gamas de procesadores ARM también soportan el juego de instrucciones THUMB-2 además de las instrucciones ARM.

## Estructura

* El banco de registros dispone de 16 registros (R0 .. R15) de propósito general con un tamaño de 32 bits. Estos registros se pueden utilizar para guardar datos leídos de memoria y enviar los valores a memoria. Podemos trabajar con los valores que tengan almacenados ejecutando operaciones sobre ellos. El registro R15 no es accesible puesto que el identificador de este registro se utiliza para diferenciar unas instrucciones de otras.
* Además existe 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 puede alterar este mecanismo por medio de instrucciones de salto.
* 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 desplazamiento (~~Direccionamiento relativo a registro base~~). ~~Éstas instrucciones utilizan un registro para indicar la dirección de acceso y un entero de 12 bits para el desplazamiento.~~
* Las instrucciones se componen de 32 bits con formato variable.

## Tipos de datos

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

Los datos con los que se trabajan siempre son palabras de 32 bits.

## Instrucciones

El juego de instrucciones elegido está compuesto por instrucciones de 32 bits con formato variable.

A diferencia de la arquitectura DLX y sus instrucciones sencillas, se ha utilizado un juego de instrucciones más complejo, lo que requiere una unidad de control compleja para decodificar las instrucciones.

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

* Accesos a memoria, lectura y escritura
* Operaciones sobre registros

1. Operaciones con dos registros
2. Operaciones con un registro y un inmediato

* Operaciones de salto

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

### Accesos a memoria, lectura y escritura

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

Es posible acceder a las direcciones de memoria direccionadas por byte. En esta implementación se está obligado a cargar valores de 4 bytes de tamaño (tamaño de palabra).

Para el cálculo de la dirección de carga o almacenamiento se ha implementado un único modo de direccionamiento. Registro base "Rn + imm12".

La dirección base se obtiene del registro Rn, y se suma un inmediato de 12 bits obtenido de la instrucción.

### Procesamiento de datos

Las operaciones de procesamiento se realizan aplicando ciertas operaciones sobre 2 operandos.

Las cálculos aplicables son operaciones aritmético-lógicas. Se aplican sobre dos operandos y el resultado (si existe) se almacena en un registro.

Dependiendo de la instrucción los operandos pueden ser:

#### 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 utilizar para diferenciar ciertas operaciones de otras. Por ejemplo, si el código de operación es "0010" y el registro origen Rn es R15 ("1111") 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 operaciones. Se permite mover un inmediato de 16 bits a un registro, pudiendo elegir si los dos bytes se almacenaran en los 16 bits más significativos o en los 16 bits menos significativos. Además se permite sumar o restar un inmediato de 12 bits a un registro.

### 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. 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. El tamaño del inmediato se ve afectado porque se requiere un campo de 4 bits para la condición de salto.

Previamente a un salto condicional se debe ejecutar una operación de comparación. Esta operación activa unos flags dependiendo del resultado de la comparación, y estos se mantienen hasta que se vuelva a ejecutar otra comparación. Si no se ejecuta la comparación, el estado de los flags es desconocido y el procesador se comportará de manera no controlada.

## Formato de instrucciones

http://www.uv.es/varnau/ManualDLX.pdf

Procesador con arquitectura RISC compatible con ARM a nivel de instrucción.

Se ha implementado un procesador con arquitectura RISC segmentada en 5 etapas. ~~Búsqueda, Decodificación, Ejecución, Memoria, Escritura (IF, ID, EXE, MEM, WB).~~

El procesador ha sido adaptado para ser compatible con un subconjunto de instrucciones de las arquitecturas ARM ~~de los procesadores ARM CORTEX M3 y M4~~. Este subconjunto es el conocido como THUMB-2, del que se hablará con detalle más adelante.

Las etapas en las que se divide el procesador son las siguientes:

1. Búsqueda (IF, Instruction Fetch)

Trae la siguiente instrucción de memoria para que el procesador pueda trabajar con ella.

1. Decodificación (ID, Instruction Decode)

Analiza la instrucción para obtener los datos y las señales de control para realizar la operación.

1. Ejecución (EXE, Execution)

Ejecuta una operación aritmética o lógica sobre los datos con los que se trabaja. También calcula la dirección de salto si es necesario.

1. Memoria (MEM, Memory)

Fase de acceso a memoria, se escriben o se leen datos de memoria.

1. Escritura (WB, Write Back)

Se almacenan los resultados de la operaciones en los registros de trabajo del procesador.

D:\TFG\TFG\Memoria\Imagenes\usar\Procesador\Procesador segmentado.tif

Ilustración . Ruta de datos de procesador segmentado.

# Arquitectura - Diseño

La arquitectura implementada ha sido basada en el procesador DLX que se ha estudiado en las asignaturas de la rama de computadores. Se ha adaptado la arquitectura del DLX para ser compatible con las instrucciones ARM.

<DLX: http://eprints.ucm.es/13047/1/Simulador\_DLX\_con\_repertorio\_multimedia.pdf>

La arquitectura del procesador permite ejecutar programas sencillos. Permite ejecutar instrucciones de acceso a memoria, saltos de dirección y realizar operaciones aritmético-lógicas.

El procesador implementado tiene una arquitectura RISC (Reduced Instruction Set Computer) con instrucciones de 32 bits. Está segmentado en 5 etapas al igual que los procesadores DLX.

Todas las instrucciones implementadas duran un máximo de cinco ciclos.

El acceso a memoria no se ha incorporado y se ha sustituido por una memoria integrada en la FPGA de pequeño tamaño con un tiempo de acceso de un ciclo.

El procesador está segmentado en 5 etapas.

1. Búsqueda

Obtención de la instrucción

1. Decodificación

Análisis de la instrucción, obtención de los valores de los operandos y extracción de las señales de control para las siguientes etapas.

1. Ejecución

Ejecución de la operación aritmético-lógica y cálculo de la dirección de salto.

1. Memoria

Acceso a memoria de datos, escritura y lectura.

1. Escritura

Escritura de resultados en el banco de registros.

Gracias a la segmentación se consigue un aumento en el rendimiento. Permitiendo al procesador lanzar varias instrucciones sin esperar a que terminen las anteriores.

## -------------- Cómo se tratan los riesgos??

En esta implementación no se tratan los posibles riesgos que pueden aparecer por la segmentación. Si se va a ejecutar un programa, los riesgos se deben evitar en la compilación insertando instrucciones NOP después de las instrucciones con posibles riesgos. Por lo general no serán necesarias más de 3 instrucciones NOP para evitar cualquier tipo de riesgo.

Tipos de riesgos:

1. Riesgos estructurales

Estos riesgos no aparecen en esta implementación debido a que cada etapa tiene una duración máxima de 1 ciclo. Si apareciesen deberían introducirse instrucciones NOP para evitarlos.

1. Riesgos por dependencia de datos

Los riesgos de dependencia se deben evitar insertando instrucciones NOP antes de la instrucción que provoca el riesgo. Se deben insertar tantas como se necesiten para que el dato se almacene en el banco de registros antes de que se haga uso del mismo.

1. Riesgos de control

Se deben insertar 3 instrucciones NOP después de una instrucción de salto. Con estas 3 instrucciones el contador de programa será modificado antes de que se cargue la siguiente instrucción. Así se asegura que la instrucción cargada sea correcta.

## Formato de instrucciones

<Informacion sobre thumb2: An Introduction to the ARM Cortex-M3 Processor>

<Informacion sobre thumb2: ARM® Compiler toolchain>

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 más información sobre el juego de instrucciones THUMB-2 se puede acudir al libro <Thumb-2SupplementReferenceManual.pdf>

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

Las instrucciones implementadas, divididas por grupo, son:

1. Transferencia

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Formato general | 1 | 1 | 1 | 1 | 1 | 0 | 0 |  | | | | | | | | |
| Rn + imm12 |  | | | | | | | S | 1 | Size | | L | Rn | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Formato general |  | | | | | | | | | | | | | | | |
| Rn + imm12 | Rt | | | | imm12 | | | | | | | | | | | |

|  |  |
| --- | --- |
| S | Extensión de signo (S==1) o extensión con cero (S==0) |
| Size | ??? Tamaño de dato cargado |
| L | Indica si es una operación LOAD (L==1) o STORE (L==0) |
| Rn | Registro con la dirección de carga |
| Rt | Registro destino del dato |
| imm12 | Desplazamiento de dirección |

Restricciones:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instrucción | S | Size | L | Rt |
| LDR | X | 0b0X | 1 | Not R15 |
| STR | 0 | Not 0b11 | 0 | Not R15 |

Notas:

Las instrucciones LOAD y STORE se identifican por los bits [31..25] y deben ser "111\_1100".

El origen de la dirección debe ser un registro (Rn), previamente cargado con el valor dirección del dato deseado.

1. Operaciones

Las operaciones se separan según el tipo de operandos que se apliquen. El operando A siempre es obtenido de un registro. Mientras que el operando B puede ser el valor de un segundo registro o puede formar parte de la instrucción.

* 1. Con inmediato

Se han implementado dos tipos de operaciones con inmediato, operaciones de carga de inmediato y operaciones aritméticas.

Las operaciones aritméticas se componen principalmente de la operación, un registro origen, un inmediato de 12 bits y un registro destino. Las operaciones implementadas son la suma y la resta.

Las operaciones de carga de inmediato, se componen principalmente de la operación, un inmediato de 16 bits y el registro destino. Las operaciones implementadas son "move top" (carga en los 16 bits más significativos) y "move wide" (carga en los 16 bits menos significativos).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Formato general | 1 | 1 | 1 | 1 | 0 |  | | | | | | | | | | |
| Add, Subtract, plain 12-bit immediate |  | | | | | i | 1 | 0 | OP | 0 | OP2 | | Rn | | | |
| Move, plain 16-bit immediate |  | | | | | i | 1 | 0 | OP | 1 | OP2 | | imm4 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Formato general | 0 |  | | | | | | | | | | | | | | |
| Add, Subtract, plain 12-bit immediate |  | imm3 | | | Rd | | | | imm8 | | | | | | | |
| Move, plain 16-bit immediate |  | imm3 | | | Rd | | | | imm8 | | | | | | | |

|  |  |
| --- | --- |
| OP:OP2 | Código de operación |
| Rn | Registro origen para operando 1 |
| Operaciones con inmediato de 12 bits | |
| Rd | Registro destino |
| i:imm3:imm8 | Inmediato para operaciones de 12 bits |
| Operaciones con inmediato de 16 bits | |
| imm4:i:imm3:imm8 | Inmediato para operaciones de 16 bits |

Restricciones:

No hay restricciones especiales.

Notas:

Las operaciones con inmediato se identifican por los bits [31..27] y el bit [15], estos deben tener los valores "1\_1110" y "0" respectivamente.

Tabla de operaciones con inmediatos de 12 bits

|  |  |  |  |
| --- | --- | --- | --- |
| Operación | OP | OP2 | |
| ADD | 0 | 0 | 0 |
| SUB | 1 | 1 | 0 |

Tabla de operaciones con inmediatos de 16 bits

|  |  |  |  |
| --- | --- | --- | --- |
| Operación | OP | OP2 | |
| MOVT | 1 | 0 | 0 |
| MOV | 0 | 0 | 0 |

* 1. Con registro

Las operaciones que hacen uso de dos registros son aritméticas (sumar, restar y mover), lógicas (and, or y or exclusiva) y de comparación que activan diferentes flags (Negativo, Cero). En el caso de una comparación no se modifican los registros.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Formato general | 1 | 1 | 1 |  | 1 | 0 | 1 |  | | | | | | | | |
| Data processing: Constant shift |  | | | 0 |  | | | OP | | | | S | Rn | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Formato general |  | | | | | | | | | | | | | | | |
| Data processing: Constant shift | SBZ | imm3 | | | Rd | | | | imm2 | | type | | Rm | | | |

|  |  |
| --- | --- |
| OP | Código de operación |
| S | Actualiza los flags si se cumplen las condiciones |
| Rn | Registro origen para operando 1 |
| SBZ | Debería ser cero ("Should be Zero") |
| imm3:imm2 | Desplazamiento |
| Rd | Registro destino |
| type | Tipo de desplazamiento usado |
| Rm | Registro origen para operando 2 |

Restricciones:

Existen restricciones para cada operación, se pueden consultar junto a los códigos de operación más adelante.

Notas:

Las operaciones con inmediato se identifican por los bits [31..29] y [28..25], estos deben tener los valores "111" y "101" respectivamente.

En esta implementación los campos "S", "imm3:imm2" y "type" no se utilizan.

Las instrucciones implementadas junto con sus respectivos códigos de operación son los siguientes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operación | Código | | | | Notas |
| ADD | 1 | 0 | 0 | 0 |  |
| AND | 0 | 0 | 0 | 0 |  |
| CMP | 1 | 1 | 0 | 1 | (Rd=="1111", S==1) |
| EOR | 0 | 1 | 0 | 0 |  |
| MOV | 0 | 0 | 1 | 0 | (Rn=="1111") |
| ORR | 0 | 0 | 1 | 0 |  |
| SUB | 1 | 1 | 0 | 1 |  |

1. Control de flujo

Las instrucciones de operación de flujo son aquellas que pueden alterar el contador del programa. En esta implementación se aceptan los saltos condicionales, haciendo uso de los flags activados por las operaciones de comparación, y los saltos incondicionales.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Formato general | 1 | 1 | 1 | 1 | 0 |  | | | | | | | | | | |
| Salto |  | | | | | S | offset[21:12] | | | | | | | | | |
| Salto condicional |  | | | | | S | cond | | | | offset[17:12] | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Formato general | 1 |  | | | | | | | | | | | | | | |
| Salto |  | 0 | I1 | 1 | I2 | offset[11:1] | | | | | | | | | | |
| Salto condicional |  | 0 | J1 | 0 | J2 | offset[11:1] | | | | | | | | | | |

|  |  |
| --- | --- |
| S | Extension de signo (S==1) o Extensión con cero (S==0) |
| offset | Desplazamiento del salto |
| I1, I2 | bits 23 y 22 del desplazamiento, respectivamente |
| J1, J2 | bits 19 y 18 del desplazamiento, respectivamente |
| cond | Condiciones necesarias para realizar el salto |

Restricciones:

No hay restricciones especiales.

Notas:

Las instrucciones de control de flujo se reconocen por los bits [31..27] cuando tienen el valor "1\_1110" y por el bit [15] que debe ser "1".

Los bits [1..0] del campo "cond" (bits [23..22] de la instrucción) indican los flags N y Z respectivamente.

-- Implementación

- Votadores

- Procesador Tolerante a fallos

- Inserción de fallos

-