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:

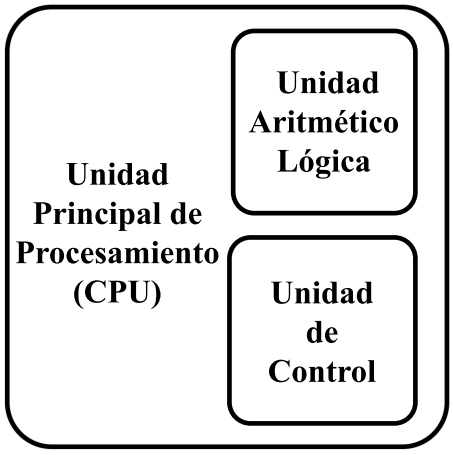

Figura 2.1: Procesador DRAE

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.

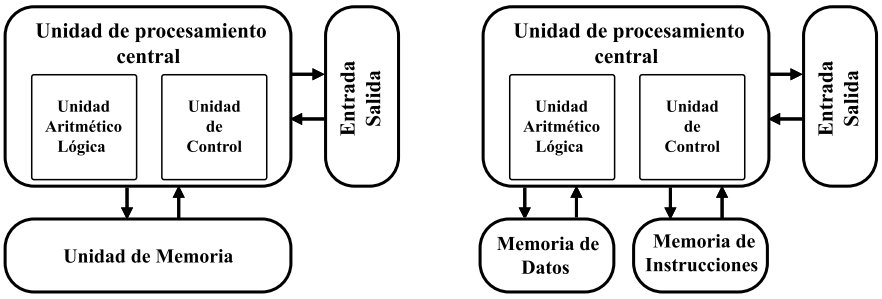

Figura 2.2: Arquitectura Von Neumann y Arquitectura Harvard

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.

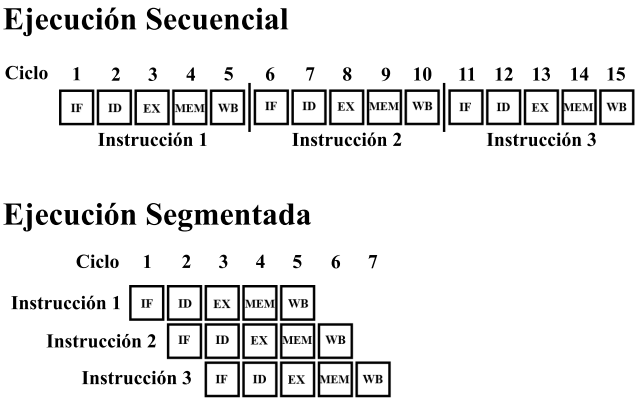

Figura 2.3: Ejecución secuencial comparada con ejecución segmentada

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.

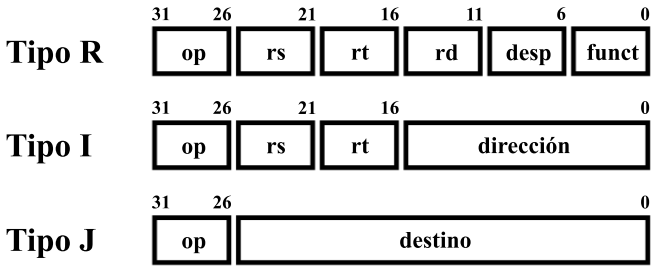

Figura 2.4: Formato de instrucciones DLX

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

...

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. ????

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.