**Instituto Tecnológico de Costa Rica**

**Escuela de Ingeniería Electrónica**

# Implementación de un modelo RTL de microprocesador RISC con herramientas de lenguaje de modelado de hardware

**Informe de Proyecto de Graduación**

**para optar por el grado de Licenciado**

**en Ingeniería Electrónica**

**Alfonso Chacón Rodríguez**

**Cartago, febrero de 2003**

**Resumen**

Este proyecto pretendía generar la base sobre la cual desarrollar futuros proyectos de investigación en del área de integración en alta escala de circuitos electrónicos (VLSI) y particularmente comp. arte de las labores desarrolladas del Laboratorio de Investigación y Desarrollo en VLSI de la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica. Para ello, se decidió implementar el modelo a nivel de transferencia de registros (RTL) de un microprocesador de juego reducido de instrucciones con arquitectura de Harvard y estructura segmentada y encauzada. Dicho modelo fue llevado a cabo sobre la base computacional Linux que posee el laboratorio, mediante el uso de las herramientas de codificación, compilación, simulación y síntesis de circuitos integrados digitales de la compañía Synopsys. Ello significó, aparte de la realización del modelo RTL mismo, la puesta a punto de los sistemas computacionales que se necesitaron como apoyo.

**Abstract**

The objective of this project was to generate a foundation for future research in the area of Very Large Scale Integration of electronic circuits and particularly as part of the activities undertaken by the VLSI Research and Development Laboratory of the School of Electronics Engineering at the Costa Rica Institute of Technology. For that purpose, a VHDL model of a 32 bit RISC microprocessor with Harvard Architecture and pipelined segmentation has been implemented for simulation at the RTL level. The model was programmed on the Linux-based system of the laboratory, using the tools of coding, compilation, simulation and synthesis of digital systems as provided by Synopsys Inc. That meant, besides the achieving of the RTL model itself, the fine-tuning of the supporting software and hardware tools needed.

**Keywords:** VHDL; Microprocessor Pipelined Architecture; VLSI Design; RTL model; Hardware description language, EDA tools.

# ÍNDICE GENERAL

## ÍNDICE DE FIGURAS

## ÍNDICE DE TABLAS



# CAPÍTULO 1: INTRODUCCIÓN

## 1.1 Descripción de la empresa

La Escuela de Electrónica tiene 27 años de fundada, y cuenta actualmente con 23 profesores y aproximadamente 700 estudiantes. Imparte un diplomado en Electrónica, un bachillerato en Ingeniería Electrónica, y acaba de aprobar el grado académico de licenciatura para esta especialidad. Además, la Escuela se encuentra en la actualidad en un proceso de acreditación académica internacional que significa nuevos retos para sus docentes y estudiantes, sobre todo con el rápido avance de las tecnologías electrónicas y que han puesto al país en la necesidad de generar nuevos graduados con un mayor grado de conocimientos especializados en alta tecnología. Una de dichas áreas, en las que busca incursionar precisamente la Escuela, es la del desarrollo de circuitos integrados de alta integración (VLSI), para cuya investigación se decidió equipar un laboratorio con equipo donado en gran parte por Intel Inc.. Este laboratorio es dirigido por el MSc. Roberto Pereira, graduado en la Universidad Técnica de Dinamarca en Ingeniería de Computadores, y con experiencia profesional en el modelado en alto nivel y producción de circuitos electrónicos integrados. Además, laboran en el laboratorio el autor y José Bogarín, estudiante de Electrónica y experto en sistemas Linux, quien es el encargado de la administración del laboratorio. Las labores típicas del laboratorio incluyen el mantenimiento de las herramientas, soporte a los estudiantes y profesores usuarios de las mismas, e investigación en el área de diseño de VLSI por medio de herramientas de software.

## 1.2 Definición del problema y su importancia

Como se mencionó en el apartado anterior, la Escuela de Ingeniería en Electrónica pretende impulsar la investigación en el área de desarrollo de VLSI para así proveer a sus estudiantes de las capacidades que los vuelvan más atractivos para el mercado tecnológico actual mundial. Igualmente, dicha investigación puede

impulsar el crecimiento académico de la Escuela e incluso motivar la creación de un posgrado en un área clave para el desarrollo del país: la producción de alta tecnología desarrollada en Costa Rica.

Clave para los mencionados propósitos era la puesta en marcha del laboratorio de Investigación y Desarrollo en VLSI, y la generación de una ruta de trabajo efectiva que tuviese al laboratorio funcionando a principios del segundo semestre del 2003. Para alcanzar dicha meta, el encargado del laboratorio, Ing. Pereira, sugirió la realización de un modelo de circuito electrónico avanzado que, por un lado, sirviera de prueba para los equipos a instalar y, por otro, proveyera de una base académica sobre la cuál construir nuevos proyectos.

## 1.3 Objetivos

### 1.3.1 Objetivo general

El objetivo de este proyecto era desarrollar un modelo de transferencia de registros (RTL) de un microprocesador RISC por medio de un lenguaje de descripción de hardware, para que sirviera como caso de estudio para ser usado en los cursos de laboratorio de sistemas digitales de la Escuela de Ingeniería Electrónica.

### 1.3.2 Objetivos específicos

1. Diseñar un modelo de microprocesador RISC basado en arquitectura MIPS, con su conjunto definido de instrucciones, y documentar su estructura por medio de bloques funcionales.

2. Participar activamente en la configuración de la base computacional sobre la que correrían las herramientas de modelado y síntesis adquiridas por la escuela.

3. Programar el modelo RISC por medio del lenguaje de descripción de hardware VHDL.

4. Simular el modelo RISC en una herramienta de software que permitiese verificar, mediante la ejecución de un código de programa que probara las instrucciones definidas para el microprocesador, su correcta implementación.

5. Generar un documento final que resuma los resultados del proyecto y sirva de base para la elaboración de un microprocesador RISC funcional, usando las herramientas de síntesis disponibles en la Escuela.

# CAPÍTULO 2: ANTECEDENTES

## 2.1 Estudio del problema a resolver

La escuela de Ingeniería Electrónica ofrece varios cursos en el área de diseño digital y de computadoras, los cuales requieren cada vez más de una base tecnológica adecuada que, por un lado, ponga a sus alumnos al corriente de los procesos industriales usados en la fabricación y diseño de sistemas electrónicos; y por otro, que sirva de apoyo no solo en la enseñanza de sistemas típicos digitales, sino que también impulsen la investigación y desarrollo de nuevas tecnologías y soluciones electrónicas. Para ello, y con el apoyo de Intel Inc., la Escuela adquirió un conjunto de herramientas de software para el diseño e implementación de circuitos electrónicos de muy alta escala de integración (en adelante VLSI, de sus siglas en inglés: *Very large scale integration*), con la intención de crear un laboratorio de investigación y desarrollo en esa área. Dicho laboratorio se montaría utilizando las estaciones de trabajo Dell donadas por Intel Inc, sobre una plataforma de sistema operativo Linux RedHat. La idea de la Escuela era tener la primera etapa del laboratorio de VLSI en funcionamiento para el primer semestre del 2003, en apoyo al plan de certificación de la carrera y de acuerdo con los objetivos de Intel Inc. para con el equipo donado por ellos.

El problema podía entonces dividirse en tres áreas:

a. Montaje de la base computacional del laboratorio. Este apartado incluía la configuración básica de las máquinas —i.e. instalación de sistema operativo, administración de disco duro y otros recursos, necesidades de protección eléctrica, etc.— y la puesta a punto de su conectividad para hacer funcional al laboratorio e integrarlo a la red de computadoras de la Escuela. Ello significaba la estructuración y configuración de distintos servicios de red y seguridad para el funcionamiento de las herramientas y su acceso por parte de los futuros usuarios, tales como: transferencia y

almacenamiento centralizado de archivos, autenticación centralizada de usuarios, acceso limitado a Internet y acceso remoto gráfico desde otras plataformas. Dicha implementación debía realizarse sobre una plataforma Linux, lo que significaba introducir en este ambiente a la Escuela, que no contaban aún con suficiente experiencia en el uso de este sistema operativo.

b. Instalación de las herramientas adquiridas y ejecución de pruebas básicas de funcionamiento. Esto implicaba la configuración de los servidores de licencias necesarios y de las diferentes variables y scripts de ambiente Linux para facilitar el uso del software.

c. Elaboración de un caso de estudio que sirviese como prueba exhaustiva de algunas de las herramientas adquiridas, y que a la vez funcione como base didáctica para futuros desarrollos dentro de la Escuela. Con este fin, se escogió una estructura básica de microprocesador con conjunto reducido de instrucciones (en adelante RISC) basado en arquitectura MIPS. El microprocesador sería modelado en lenguaje de descripción de hardware (HDL). La idea era crear un modelo a nivel de transferencia de registros (RTL) de dicho microprocesador, con la intención de que sirviera de base para una posterior síntesis del modelo, como paso previo a su implantación en un prototipo FPGA o incluso ASIC. Ello implicaba entonces no solo la implementación de un modelo de simulación HDL para pruebas del diseño, sino además la instalación de herramientas de programación de arquitectura MIPS —tales como un ensamblador y simulador de MIPS— que permitieran la generación de códigos de prueba.

## 2.2 Requerimientos de la empresa

La escuela requería que la primera etapa de su laboratorio de Investigación y Desarrollo en VLSI funcionase para el primer semestre del año 2003. Ello significaba:

a. Tener una base computacional multiusuario estable para ejecución por parte de estudiantes y profesores de las herramientas de modelado y síntesis de chips, y del software básico de apoyo: programas de graficación, editores de texto, calculadoras, etc.

b. Poseer una interfaz gráfica de acceso remoto a estos sistemas, para que los usuarios pudiesen acceder a sus proyectos o ejecutar las herramientas desde otras zonas de la Escuela.

c. Tener instalado un compilador de lenguaje de HDL para crear modelos de hardware de diferentes circuitos electrónicos.

d. Poseer un simulador de HDL eficaz que permita efectuar pruebas exhaustivas de funcionamiento sobre los circuitos modelados.

Por otra parte, el modelo de procesador RISC debía estar documentado de manera que fuera fácil continuar con su proceso de síntesis, y pudiera integrarse dentro de los programas educativos de la Escuela.

### 2.3 Solución propuesta

Montar la base computacional requerida para desarrollar el modelo de RTL de un microprocesador con conjunto reducido de instrucciones, de arquitectura de Harvard con ruta de datos segmentada. El microprocesador sería modelado con VHDL, para ser probado con las herramientas de simulación Synopsys adquiridas por la Escuela.

# CAPÍTULO 3: PROCEDIMIENTO METODOLÓGICO

Para cumplir los objetivos planteados, fue necesario planear dos estrategias de desarrollo paralelas:

1. Montaje de la base computacional para las herramientas de Sinopsis, e instalación de las mismas.

2. Implementación del modelo de RTL del microprocesador.

Debido a las diferentes aproximaciones a cada estrategia, ambas se explican por separado, entendiendo que la realización satisfactoria de las últimas etapas de la segunda estrategia dependieron de la finalización satisfactoria de la primera (el gráfico de programación de tareas incluido en el apéndice No. 1 expone de manera más clara esta afirmación).

## 3.1 Montaje de la base computacional para las herramientas de Synopsys e instalación de las mismas.

### 3.1.1 Diseño jerárquico de la red de computadoras a usar como base computacional.

### 3.1.2 Obtener el software y la documentación de apoyo sobre el software a instalar, para evaluar requerimientos del sistema.

### 3.1.3 Configuración de una computadora prototipo: instalación de software de licencias y herramientas de Synopsys

### 3.1.4 Instalar software de acceso remoto gráfico para utilizar las herramientas GUI de Synopsys desde otras plataformas o terminales.

### 3.1.5 Instalar un servidor de autenticación para centralizar la administración de usuarios

3.1.6 Instalar un servidor de archivos para centralizar la información de los usuarios, y permitirles acceso remoto a ellos desde otras computadoras fuera del laboratorio.

3.1.7 Crear scripts y variables de ambiente extras para facilitar la administración de las herramientas.

## 3.2 Implementación del modelo RTL del microprocesador

3.2.1 Escoger una estructura de base segmentada canalizada.

3.2.2 Diseñar el juego de instrucciones a implementar.

3.2.3 Diseñar la estructura básica canalizada.

3.2.4 Diseñar los bloques funcionales de las unidades de búsqueda de instrucciones, decodificación, ejecución, manejo de memoria y *write back* del microprocesador.

3.2.5 Introducir bloques de reenvío de datos y de manejo de riesgos de datos y brincos.

3.2.6 Codificar en VHDL las etapas en diferentes bloques.

3.2.7 Unir los bloques y verificar el funcionamiento del modelo de comportamiento del microprocesador con un simulador avanzado.

3.2.8 Realizar pruebas básicas de síntesis para acercar lo más posible al modelo a una codificación sintetizable.

## CAPÍTULO 4: DESCRIPCIÓN DEL HARDWARE UTILIZADO

El trabajo se realizó en su totalidad sobre plataformas Dell. Las estaciones de trabajo tienen las siguientes características:

- Dos microprocesadores Intel Xeon de 1,2 2 GHz
- 1,2 GB de memoria RAM
- Un disco duro SCSI II de 30 GB.
- Monitor Dell de 21 pulgadas
- Tarjeta de red 3COM 100BaseT

El servidor Dell, por su parte, consta de la siguientes características:

- Cuatro microprocesadores Pentium IV de 2,4 Ghz
- Seis discos duros SCSI II conectados en arreglo RAID 5 por una controladora de dos canales0
- Monitor Dell de 14 pulgadas
- 5,7 GB de memoria RAM
- Tarjeta de red 3COM 100BaseT
- Tarjeta de red 3COM 1000BaseT

## CAPÍTULO 5: DESCRIPCIÓN DEL SOFTWARE UTILIZADO

El software utilizado se divide en dos: lenguajes y herramientas propias para la implementación de dispositivos electrónicos con ayuda de computadora, y soporte operativo.

El lenguaje usado y las herramientas provistas por Synopsys se desglosan a continuación:

a. VHDL. Lenguaje para la descripición de hardware. Se usó la codificación especificada por la IEEE VHDL-93.

b. VHDLAN (analizador de código VHDL). Software de chequeo de sintaxis de VHDL.

c. SCS. Compilador C para crear ejecutables simulables basados en lenguaje VHDL y simulador de línea de comandos.

d. SCIROCCO. Simulador GUI de módulos VHDL compilados por SCS.

e. FPGA compiler. Software de síntesis de modelos en HDL para implementar en dispositivos programables FPGA.

Las aplicaciones de soporte operativo utilizadas fueron las siguientes:

a. Sistema operativo RedHat Linux ver 7.3. Distribución comercial del sistema operativo Linux GNU. Dentro de este paquete los siguientes servicios incluidos en la distribución estándar fueron necesarios para alcanzar los objetivos propuestos

    i. Servidor de archivos NFS (Network File System) con demonio AutoFS de montaje automatizado para centralización de archivos de usuarios.

ii. Servidor WU-FTPD para acceso y transferencia de archivos desde terminales remotas bajo protocolo de FTP (File Trasnfer Protocol).

iii. OpenLDAP. Paquete completo GNU de administración de bases de datos basadas en LDAP.

iv. Servidor y clientes VNC (virtual network computing) de AT&T Laboratories, para acceso gráfico remoto a las herramientas.

v. Biblioteca Ncurses4.0 para soporte de interfaces gráficas.

vi. Editores NEDIT y EMACS para edición de archivos con soporte de revisión de sintaxis para varios lenguajes, incluidos BASH script y VHDL.

vii. MIPS cross-compiler para Linux. Utilidades de ensamblador y enlace de MIPS para generar código ejecutable para MIPS desde una plataforma i486.

# CAPÍTULO 6: ANÁLISIS DE RESULTADOS

## 6.1 Explicación del diseño

Como ya se expuso en la sección de Procedimiento metodológico, el diseño de este proyecto puede dividirse en dos etapas: la implantación de la base computacional y el desarrollo del modelo del microprocesador RISC. A continuación se detallan los procesos llevados a cabo, junto con los problemas encontrados las soluciones aplicadas.

## 6.2 Montaje de la base computacional para las herramientas de Synopsys e instalación de las mismas

Era necesario poner a funcionar la base de doce estaciones de trabajo Dell disponibles para el Laboratorio de desarrollo e investigación en VLSI, que serviría de base para las herramientas a utilizar. El diseño de la correspondiente LAN implicaba una red integrada dentro de la red Ethernet local de la Escuela, que compartiera sus recursos y a la vez ofreciera acceso a los usuarios desde puntos afuera del laboratorio. Ahora, tomando en cuenta que la red a instalar iba a funcionar bajo un sistema operativo Linux, se facilitaban varios de los aspectos administrativos, al ser este un sistema operativo del tipo UNIX planeado precisamente para trabajos multiusuario.

En la figura 6.1 se detalla la estructura lógica de red utilizada. Como se aprecia, se instaló un servidor de archivos y autenticación sobre una plataforma Dell, tres servidores de licencias distribuidas para las herramientas de Synopsys, con sus respectivas herramientas de simulación y síntesis, y nueve estaciones solo con las herramientas. Todo el software de soporte es GNU de código abierto, es decir, software libre, lo que libera a la Escuela de problemas de licencias, y a la vez provee

la oportunidad de introducir mejoras en el software mismo para proyectos futuros. El software de Synopsys tiene, por su parte, una licencia renovable de un año.

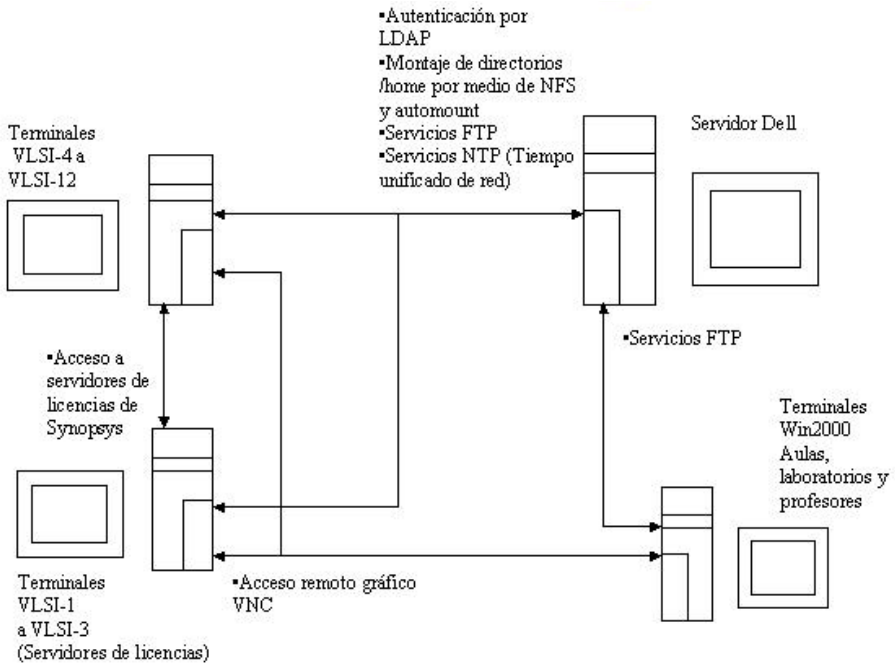



**Figura 6.1** Estructura lógica de la red del Laboratorio de Investigación y Desarrollo en VLSI

En el servidor Dell, se instalaron los siguientes servicios:

a. Servicio de almacenamiento de archivos. Este servicio se instaló sobre el protocolo NFS. De esta manera, los usuarios tienen acceso a sus archivos personales desde cualquier terminal dentro del laboratorio, por medio del demonio de automontaje AutoFS. La estructura de directorios creada sigue la siguiente filosofía:

   */home/<grupo>/<usuario>*

   El demonio autofs posee archivos de configuración que permiten el montaje de directorios remotos conforme son solicitados. A la vez, el mapeo indirecto de directorios casa permite que, sin importar de qué máquina se conecte el

usuario, obtenga en /home/usuario su directorio específico desde el servidor. Ello evita montajes excesivos que puedan traerse abajo el servidor o saturar la red, y simplifican la labor de administración.

b. Servidor FTP. Se instaló el demonio WU-FTPD que provee servicio de transferencia de los archivos de los usuarios desde otros sitios en la red de Electrónica. No se habilitaron los accesos anónimos al sistema para otorgar seguridad. Es necesario indicar que una desventaja del servicio FTP es que transmite contraseñas sin encriptar por la red. Este es un problema sobre el que todavía nos encontrábamos trabajando al finalizar el proyecto. Sin embargo, se prefirió este método a instalar el Samba para acceso NetBios, sobre todo por la ineficiencia del protocolo CIFS de Microsoft y los choque que podrían generarse entre los dominios Windows de la escuela y los que genera Samba.

c. Servidor VNC AT&T Laboratories. Este servicio de conexión gráfica remota provee de acceso a las aplicaciones GUI que se ejecutan en las estaciones de trabajo. Un demonio llamado *vncserver* corre sobre la estación de trabajo. Este servicio permite el montaje de sesiones X sobre diferentes variables de pantalla (conocidas como número de pantalla o *display*), según sean invocadas por el usuario desde una terminal de texto remota o local. Así, la máquina cliente solo debe ejecutar un pequeño utilitario (vncviewer) que se conecta con la variable *display* que se indique. Este cliente puede correr sobre Linux o Windows.

   El VNC no transmite información de sesión sobre la red, como un servidor X de acceso remoto, sino mapas de bits completos. Ello hace que sea levemente ineficiente. Su ventaja, sin embargo, radica en el hecho de que las sesiones X se arrancan en el mismo servidor y pueden mantenerse corriendo incluso si el cliente se desconecta, de manera que, días después, al volverse a conectar el cliente, encuentre su sesión X donde la dejó. Esto lo vuelve

especialmente práctico para los usuarios, que pueden cambiar de máquina sin perder el flujo de su trabajo.

d. Servidor LDAP. Este servicio permite centralizar la administración de usuarios en una sola base de datos, de una manera bastante similar a los servicios NIS (Network Information Services) de Sun. Se escogió esta implementación — instalada en su versión GNU OpenLDAP— sobre NIS por la compatibilidad que aquel ofrece con dominios de Windows, y sobre todo por que permite crear dominios de usuarios totalmente integrados con la jerarquía de usuarios de Linux y sus prestaciones de seguridad. Debido a las capacidades del LDAP, existen varias prestaciones que ayudan en gran manera:

   - Soporte del método de encriptación de contraseñas *hash* MD5, usado por UNIX estándar.
   - Servicios PAM (Puggable Authentication Modules) de autenticación segura, que evita envíos de contraseñas desprotegidas sobre la red.
   - Exportación de esqueletos de directorios casa (*home*) de usuarios.
   - Creación de listas de acceso (Access Control Lists ACLs) para limitar el acceso de ciertos usuarios a ciertas computadoras
   - Interfaz GUI de administración, que facilita el uso de las capacidades del LDAP.

   El LDAP funciona de la siguiente manera: sobre el servidor corre un demonio llamado ldap, que administra una base de datos donde se guardan los datos correspondientes a cada usuario, tal y como son definidos en un archivo de configuración: i.e. nombre completo de usuario y nombre de acceso, grupos a los que pertenece, números UID (User ID) y GID (Group ID) asignados a nombre de acceso y grupo primario (una ventaja extras sobre otros servicios de acceso concentrado, pues asegura de que los UID y GID de todas las

máquinas cliente estén sincronizados, lo que evita posteriores problemas de permisos), contraseña, directorio casa, permisos de acceso y ejecución a la estructura de disco, alias de correo electrónico, permisos de acceso a computadoras, etc. Puesto que LDAP es una base de datos totalmente configurable, se pueden definir tantas opciones como se deseen (algo que NIS no puede hacer), las cuales quedan guardadas en registros totalmente sincronizados con el ambiente Linux.

Desde la computadora cliente, se ejecuta un demonio de conexión LDAP (sladp) conectado a un puerto seguro TCP que puede ser especificado por el usuario (que por razones de seguridad no se puede indica aquí). Este demonio atrapa las invocaciones a los distintos servicios de autenticación, búsqueda de nombre de hosts, acceso NFS, etc., según son requeridos por el archivo /etc/nsswitch.conf (archivo de configuración del servicio Name Service Switch, que indica al sistema donde buscar sus recursos). Al ser así invocado por el NSS, el demonio hace una petición de conexión a través del puerto especificado usando un usuario llamado proxy (con acceso general de lectura a la base de datos) para conectarse al servidor LDAP, a quien solicita la función requerida (permitir log en el host que solicita el acceso, chequear contraseña, etc.). Si el usuario es autenticado por la base de datos, el demonio sldap devuelve al NSS los datos solicitados, y a su vez este los devuelve al sistema o pasa a la siguiente opción de búsqueda, según definida en el archivo /etc/nsswitch.

e. Servidor de tiempo de red (NTP). Finalmente, debido a la centralización de las licencias de Synopys, fue necesario instalar un servidor NTP que mantuviera a las terminales y servidores sincronizadas en hora y fecha, para evitar desfases tanto en la asignación de las licencias (que generalmente significaban la interrupción de los procesos de compilación, simulación o síntesis) hasta errores de desfase o *clock skewing* en los archivos generados. Este servidor ajusta la hora de cada cliente cada cierto tiempo, a la vez que se sincroniza

con un servidor NTP internacional a través de Internet (generalmente a su vez sincronizado con un reloj atómico).

Por otra parte, en las estaciones de trabajo Dell se instalaron las herramientas de Synopsys necesarias para la ejecución de la segunda parte del proyecto. Estas herramientas fueron:

a. Vhdlan. Analizador de sintaxis de código VHDL

b. SCS. Compilador en línea y simulador de línea de comandos de VHDL

c. Scirocco. Simulador GUI de código VHDL.

d. Design Compiler. Compilador y sintetizador de código VHDL a estructura de transferencia de registros.

e. Design Analyzer. Interfaz GUI del DesignCompiler.

f. SCL. Software de licenciamiento para las herramientas de Synopsys. Este software se instaló en tres de las estaciones de trabajo, para que funcionaran de manera distribuida.

Para proveer de acceso remoto estas herramientas fue necesario igualmente instalar los servicios de VNC, ya descritos, en cada estación, e integrar al sistema la biblioteca ncurses4.0, que da soporte gráfico a aplicaciones del tipo GUI como Scirocco y Design Analyzer. Además, se instalaron los editores Emacs y Nedit para manipulación de archivos. Estos editores, como se dijo, tienen la ventaja de ser sensibles a la sintaxis de muchos lenguajes de programación, entre ellos VHDL y Bash scripting. Ello facilitó mucho la programación, al disminuir el tiempo empleado en mero chequeo de sintaxis.

## 6.3 Implementación del modelo RTL del microprocesador

A la vez que se montaba la infraestructura computacional, se fue desarrollando el modelo de microprocesador de conjunto reducido de instrucciones (RISC) de 32 bits a implementar. Se decidió escoger una arquitectura Harvard por la facilidad de su interfaz con las memorias de datos y de programa, al estar ambos buses completamente separados. Del mismo modo, se optó por un juego de instrucciones compatible con las del microprocesador comercial MIPS, por ser este un caso de estudio bastante documentado y para el que existen varias herramientas de programación GNU. Se decidió llamar a este modelo de microprocesador Tesla, en honor al ingeniero eléctrico serbio-estadounidense, inventor de la radio, la transmisión de energía eléctrica en CA, y el alternador, entre otros muchos logros que hasta hace muy poco le han sido reconocidos por las instituciones oficiales estadounidenses.

La estructura de bloques del sistema diseñado se muestra en la figura 6.2. Como se ve, era necesario utilizar un módulo RTL para el microprocesador, y dos módulos de comportamiento para las respectivas memorias. Estos módulos fueron provistos por el asesor del proyecto, a los que hubo que hacer una breve modificación al estar diseñados para un modelo de microprocesador de 16 bits, y mapear sus direcciones de palabra de una en una, en vez de cuatro en cuatro como lo hacen los microprocesadores de 32 bits.

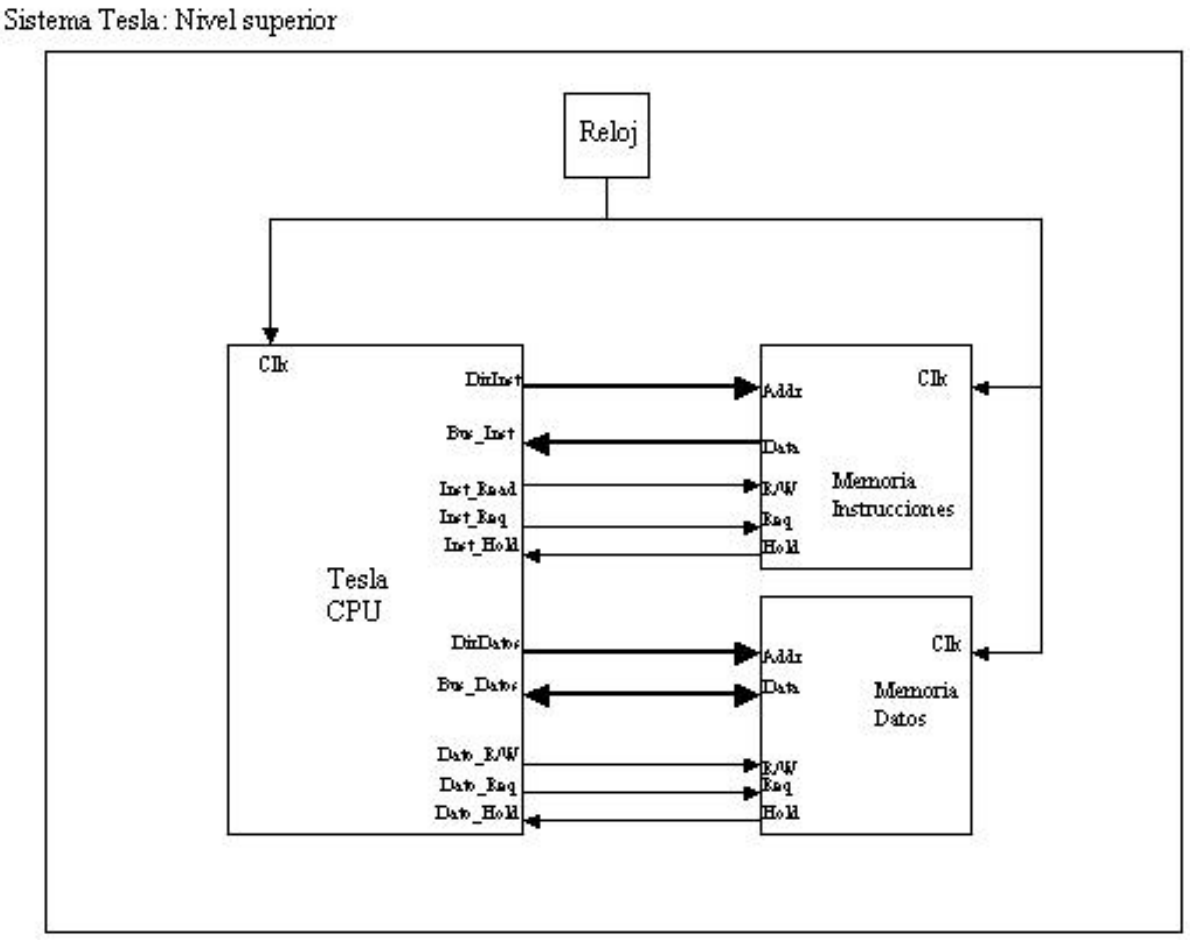



**Figura 6.2** Estructura superior de bloques del sistema de prueba del modelo de microprocesador Tesla

El juego de instrucciones final es el detallado en la tabla 6.2, donde se indica además la estructura básica de dicha instrucción según sea su tipo de registro, de operando inmediato o de control. Se indican además los códigos de función y desplazamiento (funct y shamt) y los operandos que implica, según el formato de instrucción definido en la tabla 6.1. Este formato se obtuvo del libro *Computer Design. The Hardware and Software Interfaces,* de Patterson y Hennesey, que hace un estudio muy detallado de la estructura segmentada canalizada básica de un microprocesador DLX, y que fue la base bibliográfica más importante para el posterior desarrollo del proyecto. Dichas instrucciones son típicas de un microprocesador RISC pues son de un

tamaño fijo (32 bits), las operaciones lógico-aritméticas son solo posibles entre registros o entre un registro y un operando inmediato, y solo se permiten operaciones simples de transferencia de registro a memoria y de memoria a registro, o de un operando inmediato a un registro. Igualmente, las instrucciones de salto y control están limitadas a saltos sobre registro, salto indirecto indizado sobre el contador de programa, salto directo o bifurcaciones condicionales relativos al contador de programa.

**Tabla 6.1** Formato de los tres tipos básicos de instrucción en un MIPS de 32 bits: operaciones de formato aritmético, operaciones con operando inmediato, transferencia o bifurcación, y operaciones con formato de salto (En: Hennessy, D.A., y Patterson, J.L. p.140)

| Name | Fields | | | | | | Comments |
|---|---|---|---|---|---|---|---|
| Field size | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits | All MIPS instructions 32 bits |
| R-format | op | rs | rt | rd | shamt | funct | Arithmetic instruction format |
| I-format | op | rs | rt | address/immediate | | | Transfer, branch, imm. format |
| J-format | op | target address | | | | | Jump instruction format |

Como puede apreciarse en la tabla 6.1, el juego escogido de instrucciones a la vez define el modelo de registros a usar, asunto necesario para proveer compatibilidad de lenguaje de máquina con un MIPS de 32 bits. Ello significó por tanto un banco de 32 registros de 32 bits, con arquitectura totalmente ortogonal, es decir, sin registros forzosamente dedicados a determinadas instrucciones. El direccionamiento de dichos registros es totalmente compatible con MIPS, lo que permite que el ensamblador mapee los pseudoregistros definidos por convención para el MIPS según la tabla 6.3 con los registros adecuados en la arquitectura diseñada. La idea era sentar las bases para que el modelo estuviera en un futuro en capacidad de ejecutar código generado desde lenguajes de alto nivel, tal como C, por ejemplo.

**Tabla 6.2** Juego de instrucciones a implementar para Tesla

| op (31:26) | funct (5:0) | Instrucción | Significado | Ejemplo |
|---|---|---|---|---|
| 0 | 000000 | sll (shift left logical) | rd = rt << shamt | sll $2, $2, 10 |
| 0 | 000010 | srl (shift right logical) | rd = rt >> shamt | srl $1, $4, 5 |
| 0 | 000011 | sra (shift right arithmetic) | | sra $1, $4, 5 |
| 0 | 001000 | jr | PC= rs | jr $25 |
| 0 | 100000 | add | rd <= rs + rt | add $4, $4, $6 |
| 0 | 100010 | sub | rd <= rs - rt | sub $9, $4, $17 |
| 0 | 100100 | and | rd <= rs and rt | and $13, $8, $6 |
| 0 | 100101 | or | rd <= rs or rt | or $2, $12, $16 |
| | 100110 | xor | rd <= rs xor rt | xor $24, $4, $5 |
| | 101000 | slt | si rs > rt => rd <= 1<br>si no, rd<=0 | slt $4, $14, $10 |
| | 101001 | sltu | Igual que slt, pero sin signo | sltu $5, $4, $6 |
| 8 | NA | addi | rd <= rs + inmediato | addi $13, $8, 98 |
| 2 | NA | j | PC <= destino | j salir |
| 3 | NA | jal | PC <= destino<br>ra <= PC+4 | jal subrutina |
| 4 | NA | beq | si rs=rt<br>=> PC <= PC+4+desplaz. | beq $6, $8, salto |
| 5 | NA | bne | si rs==rt<br>=> PC <= PC+4+desplaz. | bne $6, $8, lazo |
| 35 | NA | lw | rd <= mem [rs] | lw $5, $3 |
| 43 | NA | sw | mem[rs] <= rd | sw $15, $30 |
| 0 | 0 | nop | No se ejecuta nada | nop |
| 15 | NA | lui | $rt << imm&0000h | lui $5, 1024 |

**Tabla 6.3** Convención de asignaciones de registros en un MIPS. El registro 1, llamado $at, se reserva para el ensamblador. Los registros 26-27, llamados $k0-$k1, se reservan para el sistema operativo (En: Hennessy, D.A., y Patterson, J.L. p.140)

| Name | Register number | Usage | Preserved on call? |
|---|---|---|---|
| $zero | 0 | the constant value 0 | n.a. |
| $v0–$v1 | 2–3 | values for results and expression evaluation | no |
| $a0–$a3 | 4–7 | arguments | yes |
| $t0–$t7 | 8–15 | temporaries | no |
| $s0–$s7 | 16–23 | saved | yes |
| $t8–$t9 | 24–25 | more temporaries | no |
| $gp | 28 | global pointer | yes |
| $sp | 29 | stack pointer | yes |
| $fp | 30 | frame pointer | yes |
| $ra | 31 | return address | yes |

En cuanto a la estructura interna del microprocesador, se usó como base la propuesta por Hennessy y Patterson, a su vez fundamentada en la estructura segmentada de cinco etapas del DLX. Estas etapas son: Instruction Fetch (búsqueda de intrucción, en adelante etapa IF), Instruction Decode (decodificación de instrucción, en adelante etapa ID), Instruction Execution (ejecución de instrucción, en adelante EX), Data Memory Access (acceso a memoria de datos, en adelante etapa MEM), y Register Write Back (escritura en registros, en adelante etapa WB), tal y como son definidas por Hennessy y Patterson (pp. 449-498), y que se muestran en la figura 6.3. Puede apreciarse la división entre cada etapa por los respectivos registros de canalización o *pipelining*.

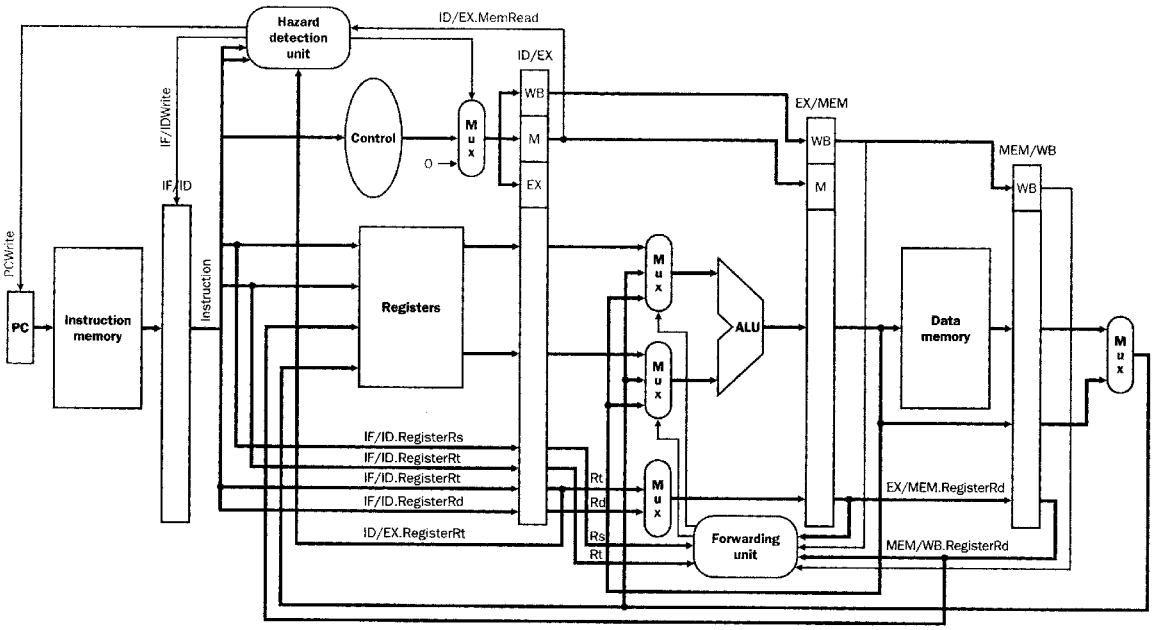



**Figura 6.3** Estructura básica de un microprocesador segmentado con encauzamiento de cinco etapas. (En: Hennessy, D.A., y Patterson, J.L. p.p. 454-70)

Fue necesario, sin embargo, hacer algunas modificaciones a la estructura propuesta que devinieron de las siguientes convenciones: el microprocesador debía tener una ranura de salto de una única instrucción (en vez de las dos que posee la estructura DLX tal como la proponen en su texto Hennessy y Patterson), y el microprocesador debía estar en capacidad de aceptar instrucciones en las que el registro operando acabase de ser modificado en la instrucción anterior. Ello significó trasladar las unidades de control y detección de salto, y de reenvío por riesgo de datos, desde la etapa de EX a la etapa ID, donde habían sido colocadas inicialmente. Por ello, la estructura utilizada es la que se muestra en la figura 6.4 (en el apéndice 9.2 se ofrecen las declaraciones de entidad de cada etapa).

Es necesario hacer notar, a la vez, que por razones de comodidad en la codificación VHDL, se muestran las etapas de MEM y WB en un solo bloque. No obstante, al

analizar cada etapa por separado, se notará como en realidad se cumple el principio de segmentación dentro de este bloque.

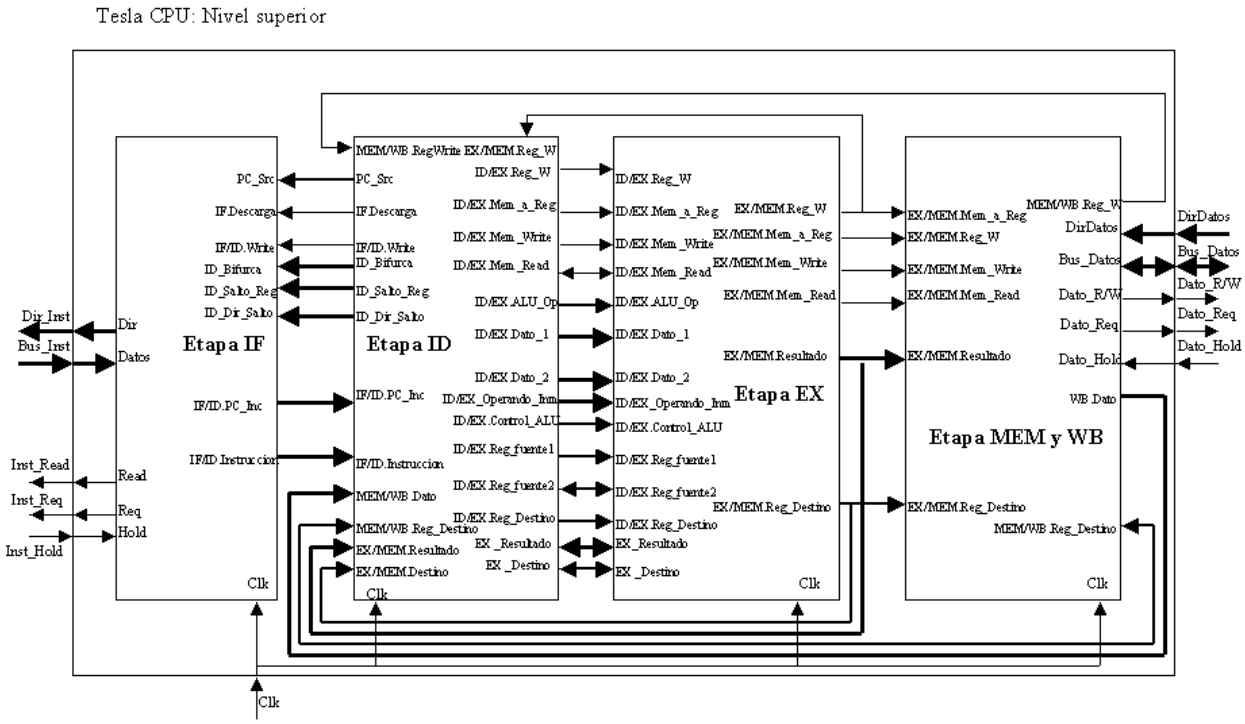



**Figura 6.4** Estructura de bloques de microprocesador Tesla

Ahora, puesto que la idea era sentar las bases de una estructura susceptible de ser mejorada o modificada desde VHDL, se modularizó la programación de cada etapa según sus funciones, de manera que el modificar una de ellas no implicara necesariamente una recodificación del resto.

Precisamente, por ser VHDL un lenguaje modular de programación que permite, fue muy sencillo ir introduciendo cambios en cada módulo conforme se fueron hallando problemas, como el mencionado traslado de la etapa de reenvío de datos y detección de riesgos de salto y de datos.

La etapa de IF (figura 6.5) es la encargada de sacar las instrucciones de la memoria de programa e ir avanzando el contador respectivo. Aquí se generan las direcciones

de acceso a la memoria de programa y se alimentan los datos recibidos (la instrucción misma) al registro de encauzado que comunica la etapa IF con la de ID. Para acceder a dichos datos, esta etapa debe respetar el protocolo de acceso al modelo de memoria utilizado en este proyecto, según se define en su diagrama de tiempos (figura 6.6).

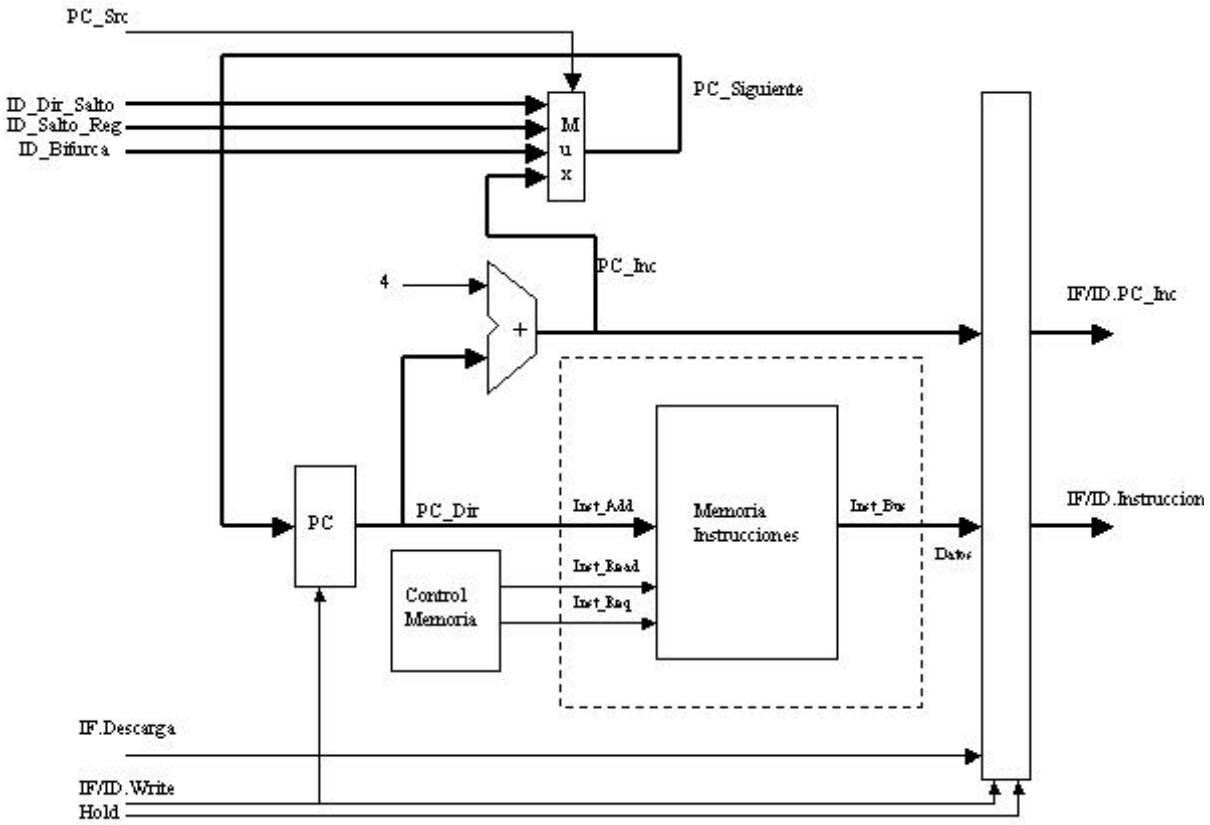
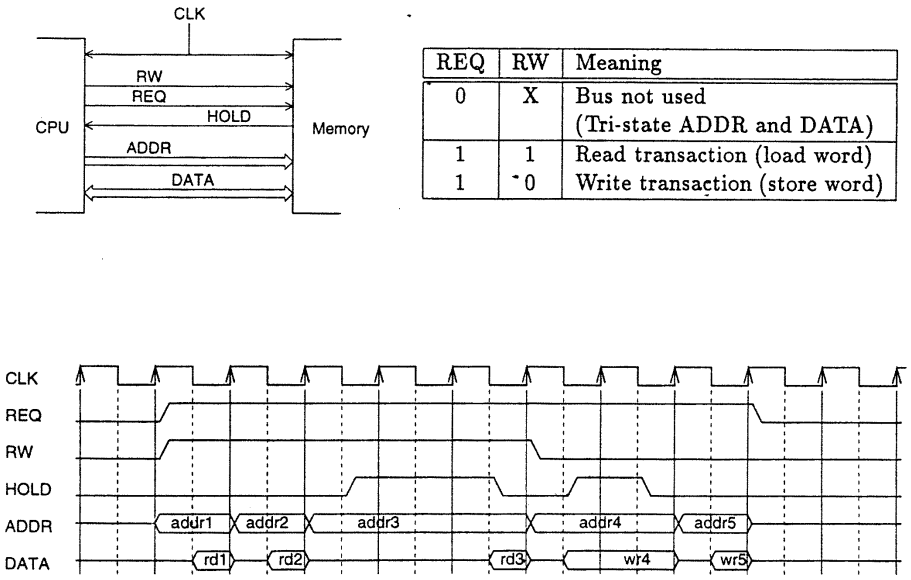



**Figura 6.5** Etapa de búsqueda de instrucción (IF)




| REQ | RW | Meaning |
|---|---|---|
| 0 | X | Bus not used (Tri-state ADDR and DATA) |
| 1 | 1 | Read transaction (load word) |
| 1 | 0 | Write transaction (store word) |

**Figura 6.6** Interfaz Memoria-CPU y diagrama de tiempos de acceso a memoria requeridos

A la vez, las señales de control provenientes de la etapa ID permiten decidir a la unidad sobre la siguiente dirección (si es un salto, bifurcación o el PC próximo, escogidas por señal PC_Source), si debe escribirse el registro de encauzado (útil en caso de que se deba detener la tubería debido a una señal de Hold de alguna de las memorias) y sobre si debe conservar o no la instrucción recibida. (Esta última señal, llamada IF_descarga, literalmente bota la instrucción del registro de encauzado, y se generó en caso de que el compilador no escogido no estuviera optimizado para aceptar un micro con una ranura de ejecución. Este no fue el caso, por lo que en este modelado esta señal está siempre habilitada. Permanece, sin embargo, por razones didácticas y de futura compatibilidad con otros tipos de compiladores).

La etapa de ID (figura 6.7) es la encargada de generar las señales de control necesarias para el resto de la estructura, a partir de la operación que recibe. Estas señales se detallan en la tabla 4. Igualmente, en esta sección se genera la escritura y lectura del banco de registros que provee los operandos implicados en la operación solicitada. En el flanco de bajada del reloj, el banco coloca en sus salidas los datos

direccionados por las variables rs y rt de la operación decodificada. Pero antes, en el flanco de subida de reloj, ya se han escrito en el banco de registro los datos provenientes de la etapa de WB, en la dirección indicada por la variable MEM.WB.RegDestino. Ello ahorra un riesgo de datos: si se necesita usar un registro en esta etapa cuyo dato proviene de una instrucción en la etapa de WB, este se actualiza medio ciclo antes de ser necesitado.

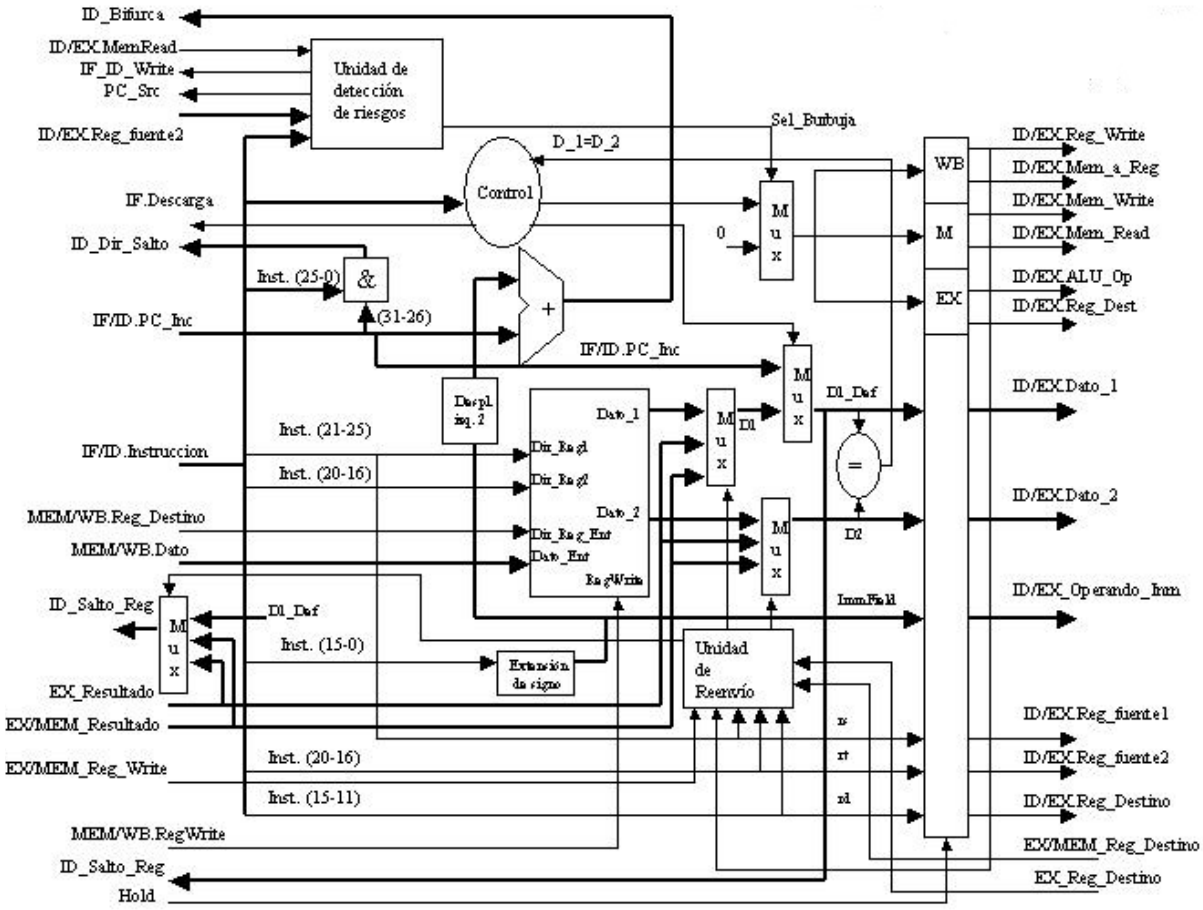



**Figura 6.7** Etapa de decodificación de instrucción

En el caso desarrollado, sin embargo, hubo que añadir otras funciones que resolvieran los riesgos de datos inherentes a una estructura segmentada. Estas funciones generan una unidad de detección de riesgos y reenvío de datos que aseguran la ejecución de las instrucciones de manera correcta, sin necesidad de

introducir burbujas que disminuyan la eficiencia del microprocesador. Estos riesgos se deben a la necesidad de usar en una instrucción un registro que acaba de ser alterado por la instrucción precedente, como por ejemplo la secuencia:

```
sltu $4, $5

bne $4, $0, sitio
```

Como en este caso, además, Tesla solo debe tener una ranura de ejecución para las operaciones de salto (es decir, que solo se tome una instrucción de programa de más antes de ejecutar una instrucción de brinco), dichos reenvíos deben hacerse ya desde la etapa de ID, pues es en ella donde se calculan o resuelven las bifurcaciones y saltos de programa. La unidad de reenvío, por tanto, debe evaluar las necesidades de registros para una operación recién codificada y traerse si es posible los datos de las distintas partes de la canalización. Puesto que se asume que un dato escrito por la etapa de Reescritura o WB es leído en el mismo periodo de reloj en el flanco negativo, solo queda traer el dato requerido desde las etapas de EX o MEM, según lo determinan las siguientes condiciones

EX.Registro destino = ID Registro Fuente 1
EX.Registro destino = ID Registro Fuente 2
EX.MEM.Registro destino = ID Registro Fuente 1
EX.MEM.Registro destino = ID Registro Fuente 2

Debe añadirse además, para los reenvíos de la etapa MEM, que el registro destino en la etapa de ejecución sea diferente de los registros fuentes, para evitar un reenvío desde esta etapa cuando el dato más reciente se encuentra en la etapa EX (Hennessy p.483). Una complicación resultante de esta estrategia es que una carga de registro desde memoria seguida de un acceso a ese mismo registro no puede resolverse por medio de reenvío, al hallarse la lectura del dato de memoria a una etapa de distancia en el futuro. Este caso se ilustra con detalle en la figura 6.8, donde se explican las consecuencias de dicho reenvío “falso” y la manera de solucionarlo: parar la tubería insertando una burbuja de ejecución que evite la carga de operandos

erróneos sin necesidad de que el compilador se vea obligado a insertar operaciones NOP.

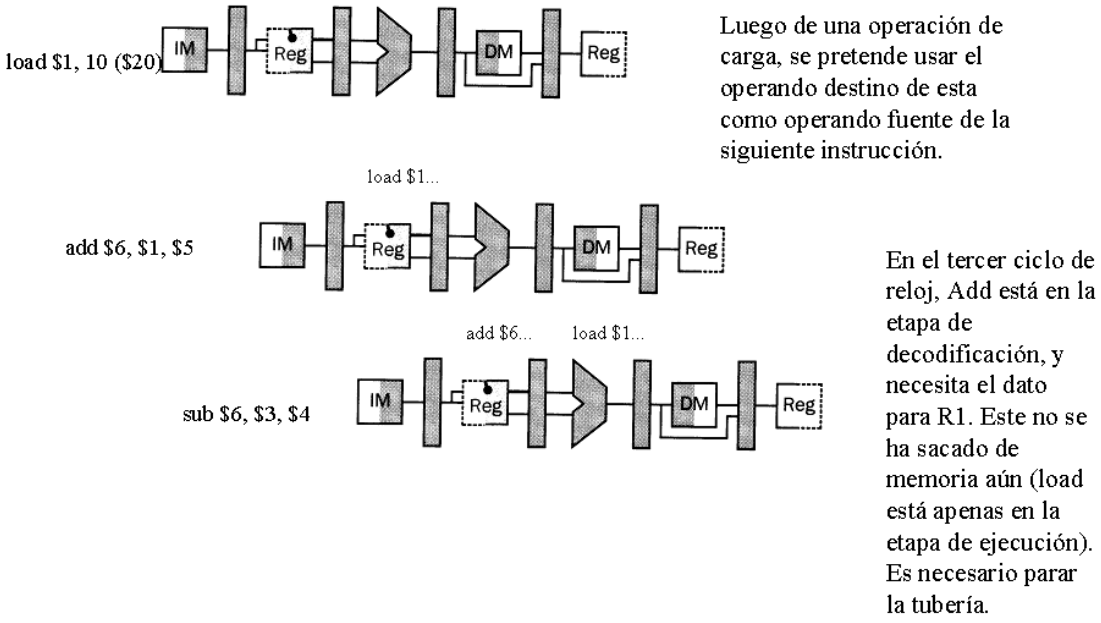



**Figura 6.8** Caso en que luego de una operación de carga de registro no se puede ejecutar de inmediato una operación que use ese registro. Debe insertarse una burbuja.

Igualmente sucede en el caso de la carga de un registro con un operando inmediato. Cuando este se requiere en la instrucción siguiente, el dato preciso ya se encuentra en la etapa EX esperando pasar a la etapa de MEM. Podría pensarse en hacer simplemente el reenvío a la unidad de ID para que el dato se escriba de una vez y esté disponible en el banco de registros para el flanco de bajada de reloj en la salida, pero como la escritura en el banco de registros se hace en el flanco positivo, que ya ha pasado para el momento que hemos decodificado la necesidad de usar ese operando, se debe esperar un nuevo ciclo de reloj para hacer el reenvío. Cuando acontece esta situación, al igual que en el caso anterior, no queda más que insertar

una burbuja para dar chance a que la unidad de reenvío pueda hacer su trabajo de manera correcta.

En el diagrama de bloques de la etapa ID pueden apreciarse todas las señales que necesitan chequearse para resolver los riesgos indicados. Igualmente, se muestran las etapas de cálculo de direcciones de salto, lo que significa que estas operaciones se ejecutan en esta etapa. En este último caso, la figura 6.9 muestra la conformación de las direcciones de salto a partir de la instrucción a ejecutar.

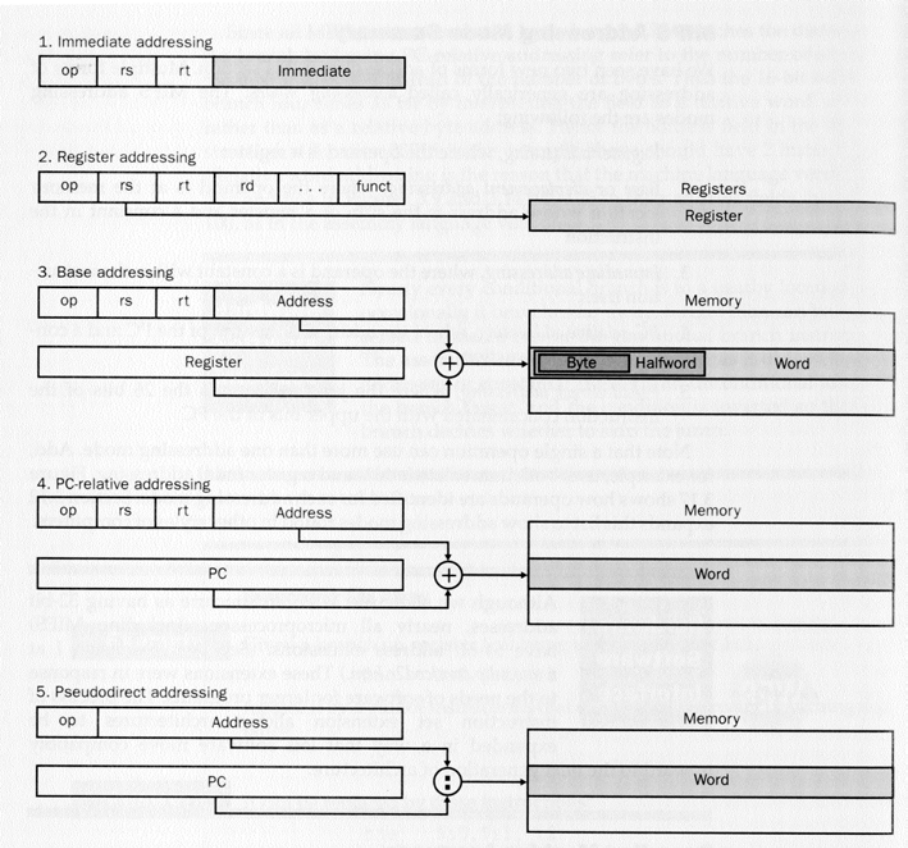



**Figura 6.9** Configuración de direccionamiento en un MIPS (En: Hennessy, D.A., y Patterson, J.L. p.p. 152)

En la figura 6.10 tenemos a la etapa EX. Puesto que ya los riesgos de datos han sido resueltos en la etapa ID, no hay necesidad de chequear en esta etapa el que los operandos correctos estén disponibles para la unidad lógico aritmética. El único

multiplexor en la ruta de datos es para seleccionar el segundo operando ya sea a partir del dato 2 del banco de registros o el operando inmediato que acompaña a la instrucción. En este mismo operando, además, se incluyen las variables de cantidad de desplazamiento (shamt) para operaciones de desplazamiento, y la función (funct) para operaciones de registro. Entre estas variables y el código de operación que se alimenta desde la etapa ID, se decide la operación lógico-aritmética que debe ejecutar la ALU. Existe además otro multiplexor que debe seleccionar el registro destino de la operación, cuya dirección vienen en diferentes partes de la instrucción según se trate de una operación con dos o un operando (ver tabla 2 para más claridad). Puede observarse en esta etapa la realimentación de señales desde dentro del bloque y desde el registro de encauzado hacia la etapa ID. Estas realimentaciones de señales permiten detectar los riesgos de salto y de datos que pueden generarse, y proveen de los datos recién calculados en caso de que la unidad de reenvío los necesite.

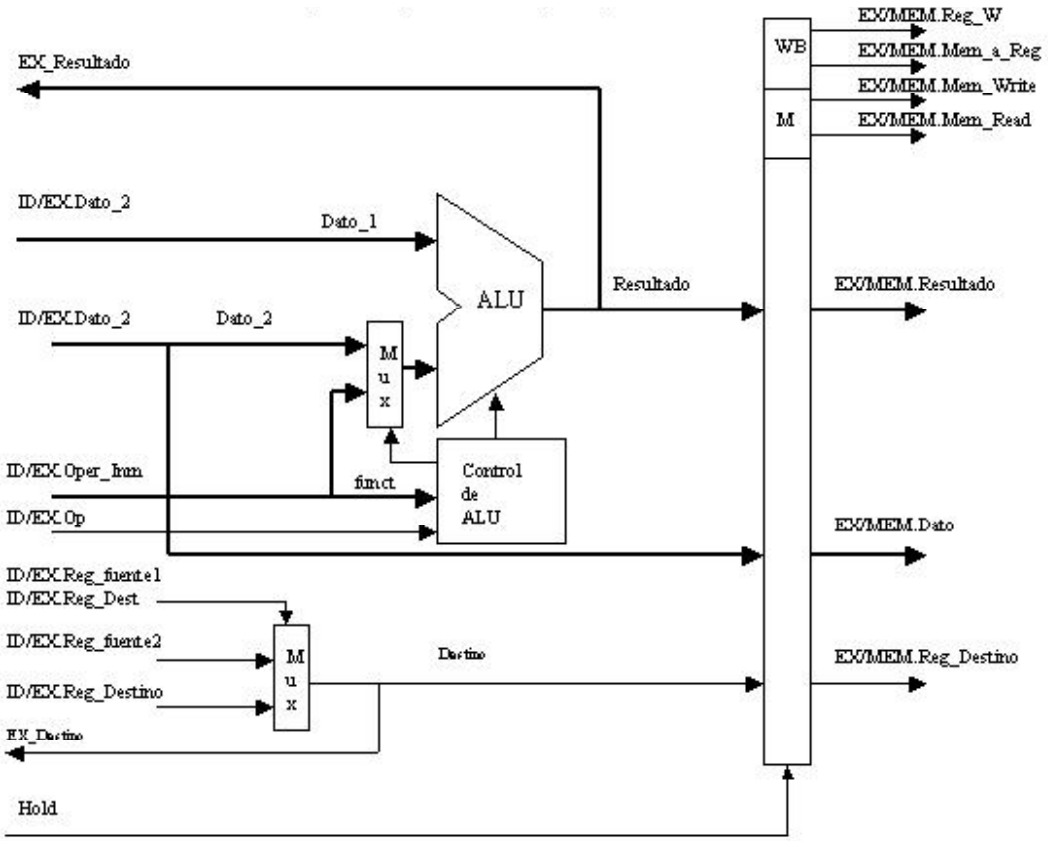



**Figura 6.10** Etapa de ejecución (EX)

El bloque MEM/WB (figura 6.11) une estas etapas en una sola entidad VHDL, pero estructuralmente continúan separadas por el registro de encauzado entre ellas. Se consideró poco práctico el colocar el código para la etapa WB en una entidad aparte, pues esta sección consiste únicamente de un multiplexor que escoge si el dato alimentado al banco de registros viene de la memoria de datos o de la ruta de datos del microprocesador. Debe notarse en la etapa de MEM la colocación de búferes de tercer estado. Estos, junto con las señales de control de la memoria, permiten la utilización de un bus bidireccional de datos.

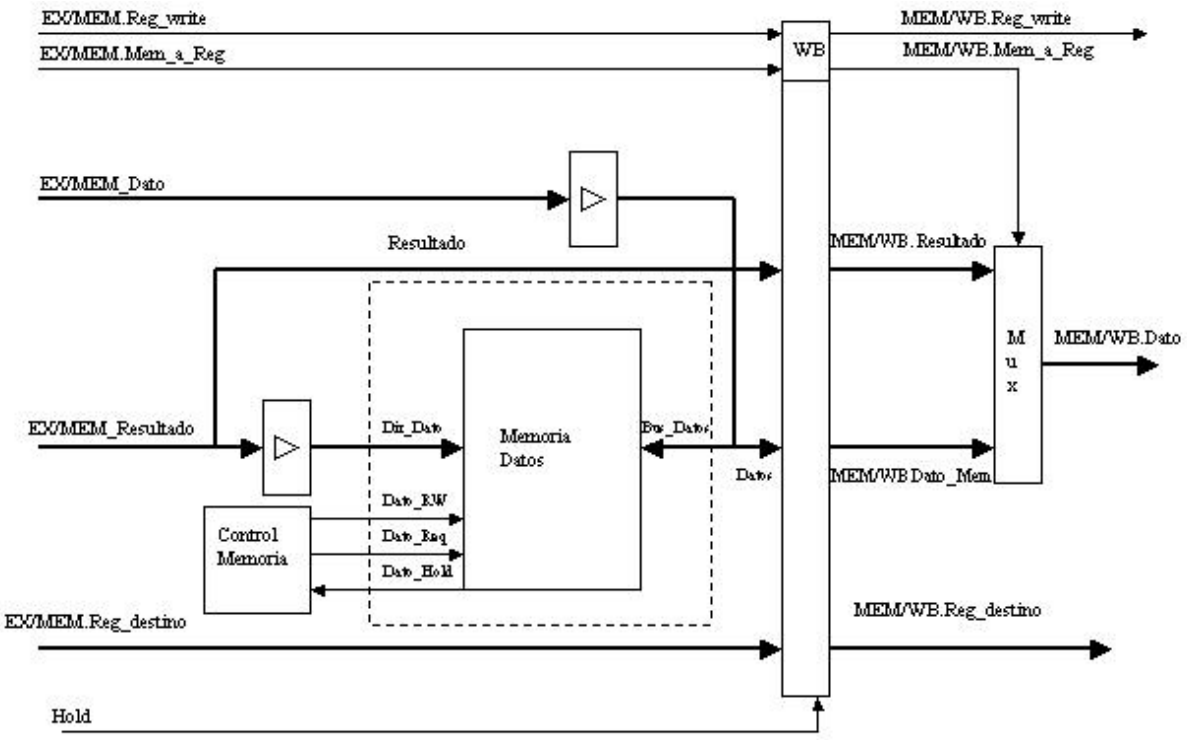



**Figura 6.11** Etapas MEM y WB

Todas las etapas se reúnen bajo una identidad llamada igual que el nombre escogido para este proyecto: Tesla. En esta entidad se define una arquitectura de donde se interconectan las instanciaciones VHDL de los diferentes módulos que representan cada etapa. En el siguiente bloque, el llamado banco de pruebas se interconectó el micro Tesla con los modelos de memoria de datos y de programa y de reloj. Este banco de pruebas genera además la señal de reset que inicializa al microprocesador (ver diagrama de la estructura de archivos VHDL en figura 6.13).

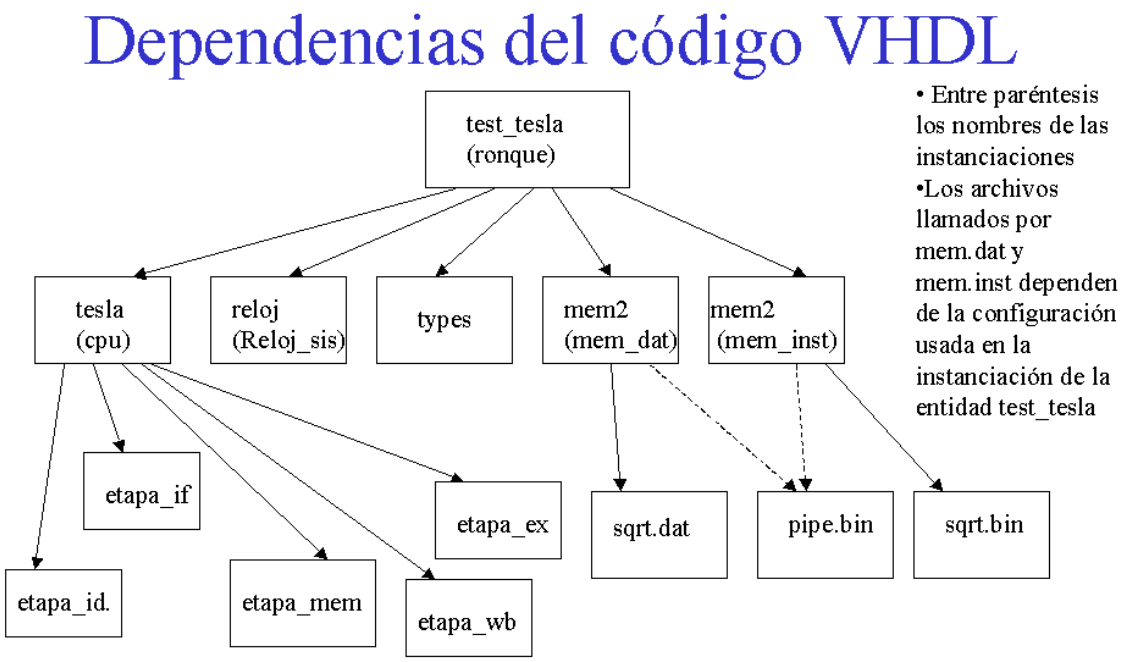



**Figura 6.12** Diagrama de relaciones VHDL para pruebas de Tesla

Se programaron dos configuraciones del banco de pruebas tal como se muestra en el listado de la figura 6.14. Una de las configuraciones invoca al microprocesador junto con un programa simple de prueba de todas las instrucciones. La segunda configuración invoca con un pequeño código (sqrt.bin) que obtiene una aproximación de raíz cuadrada para un entero definido en un archivo llamado sqrt.dat. Este código fue generado a partir del listado ofrecido en el apéndice 9.3, usando las herramientas de compilación y enlace de MIPS obtenidas. El programa lleva un control de iteraciones y ejecuta varias operaciones de brinco, lo que lo volvía útil para depurar la etapa de reenvío y detección de riesgos de salto. Una vez ensamblado el programa, el enlazador (*linker*) provee de una salida hexadecimal para grabado de memorias PROM, lo que permitiría en un futuro introducir dicho código dentro de un prototipo alambrado. En nuestro caso, solo fue necesario transformar el texto hexadecimal en texto binario y colocarlo en las filas respectivas del archivo sqrt.bin.

```
Configuration test_basico_cfg of test_tesla is

       for ronque

         For Reloj_Sis : reloj
              USE ENTITY WORK.reloj(funcionamiento);
         end for;
        For CPU :  tesla
             USE ENTITY WORK.tesla(comportamiento);
         End for;
        for Memoria_Inst: mem4
              use entity WORK.mem4(behaviour)
              GENERIC map (filename => "pipe.bin",
                           n =>   0);
         end for;
         for Memoria_Datos: mem4
              use entity WORK.mem4(behaviour)
              GENERIC map (filename => "pipe.bin",
                          n =>   0);
         end for;
       end for;
end test_basico_cfg;

 Configuration test_sqrt_cfg of test_tesla is

       for ronque

          For Reloj_Sis : reloj
               USE ENTITY WORK.reloj(funcionamiento);
          end for;
          For CPU :  tesla
              USE ENTITY WORK.tesla(comportamiento);
          End for;
          for Memoria_Inst: mem4
               use entity WORK.mem4(behaviour)
               GENERIC map (filename => "sqrt.bin",
                            n =>   0);
          end for;
         for Memoria_Datos: mem4
              use entity WORK.mem4(behaviour)
              GENERIC map (filename => "sqrt.dat",
                          n =>   0);
           end for;
       end for;
end test_sqrt_cfg;
```

**Figura 6.13** Estructura final del banco de pruebas

Para esta labor, fue muy útil el simulador para MIPS *xspim* que permitió ir comparando la simulación del modelo VHDL en Scirocco con lo que debía hacer el programa en un MIPS verdadero. Incidentalmente, el simulador hizo descubrir una instrucción faltante en la implementación inicial: *lui $reg, imm* (load upper immediate), necesaria para generar el código de máquina de las pseudoinstrucciones *lw $reg, mem* y *sw $reg, mem* (hay que recordar que el juego de instrucciones del MIPS no

permite cargar o escribir directamente a memoria sino a través de un registro operando que guarde la dirección de memoria a acceder).

El microprocesador fue simulado exitosamente con varias combinaciones de datos para el algoritmo de raíz cuadrada. Asimismo, se llevaron a cabo algunas pruebas de síntesis que sugirieron que la mayor parte del modelo es sintetizable, lo que abre un camino para la implementación a corto plazo de un prototipo basado en el mismo. Hubo sin embargo algunos problemas detectados por las herramientas de síntesis que debían ser solucionados antes de tener una implementación en compuertas funcional. Dichos problemas derivaron del irrespeto de algunas de las convenciones de codificación que establece Synopsys para el uso de sus programas.

## 6.4 Alcances y limitaciones

Al iniciar la codificación de este proyecto, se tuvo siempre en mente la utilización de las capacidades del lenguaje VHDL que fuesen sintetizables, dentro de los lineamientos de estilo de codificación que sugiere Synopsys. Hubiera sido ideal alcanzar un modelo de prototipo para este proyecto, y de hecho, el que las pruebas de síntesis sugirieran que la mayor parte del modelo puede ser llevado al nivel de compuertas en un elemento programable, pone de manifiesto que tal objetivo era alcanzable en el corto plazo. Sin embargo, debido a la necesidad de afinar los detalles del laboratorio de VLSI para su entrada en funcionamiento en el primer semestre del 2003, no hubo tiempo de llevar más adelante el diseño. Igualmente, aunque el código estuviese completamente sintetizado, todavía haría falta dedicar algunas semanas al alambrado de las memorias físicas y las etapas de I/O necesarias para tener un prototipo funcional.

En todo caso, se considera que este modelo provee de una buena base sobre la cual construir más investigaciones sobre el área de desarrollo en VLSI, y de hecho es la intención del autor continuar con la elaboración del prototipo hasta llevarlo a una etapa más avanzada y funcional. El poseer ya un modelo completamente operativo de Tesla sería fundamental para avanzar en otras áreas tales como el procesamiento

de señales o las comunicaciones digitales, al poseer ya la escuela el fundamento o *core* (para usar el término inglés) de un microprocesador RISC, totalmente desarrollado dentro de la institución.

## CAPÍTULO 7: CONCLUSIONES Y RECOMENDACIONES

De este proyecto pueden sacarse varias conclusiones importantes:

- El diseño a partir de lenguajes de descripción de hardware es versátil y eficiente, y permite ganar tiempo en la producción de circuitos electrónicos digitales de alta complejidad.
- Una buena y ordenada codificación facilita la simulación y espulgado de un modelo VHDL.
- El uso de pruebas de banco (test benches) eficientes en la etapa de simulación permite capturar muchos errores de implementación antes de pasar a la síntesis.
- El seguir las reglas establecidas por los fabricantes de herramienta de síntesis de hardware, permite alcanzar la etapa de implementación en hardware más rápidamente.

A partir de estas conclusiones, cabe recomendar la continuación de este proyecto hasta su etapa de prototipo en hardware. Para ello, es conveniente llevar a cabo una revisión del código de cada etapa, para conformarlo a las reglas de estilo definidas para las herramientas de Synopsys.

## CAPÍTULO 8: BIBLIOGRAFÍA


Ashenden, P. A. *The Student's Guide to VHDL.* San Francisco, CA: Morgan Kauffman. 1998.

Hennessy, D. A., y Patterson, J. L. *Computer Organization and Design. The Hardware Software Interface.* 2 ed. San Francisco, CA: Morgan Kauffman. 1998.

Morris, M. M., y Kime C. R. *Fundamentos de diseño lógico y computadoras.* México DF: Prentice Hall. 1998.

Negus, C. *Red Hat 8.0 Bible.* Indianapolis: Wiley Publishing. 2002.

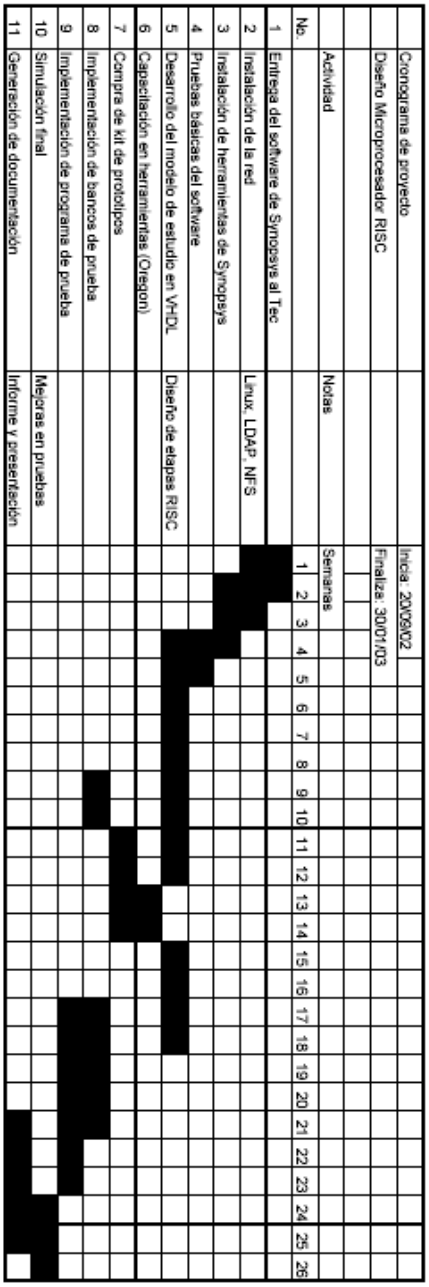
Smith, R. W. *Advanced Linux Networking.* Boston: Pearson Education. 2002.

Sinopsis, Inc. *VHDL Simulation User Guide.* Versión 2002.6. Mountain View, CA: Synopsys, Inc. 2002.

*VHDL Simulation: Coding and Modeling Style Guide.* Version 2002.6. Mountain View: CA: Synopsys, Inc. 2002.

# CAPÍTULO 9: APÉNDICES

## Apéndice 9.1 Cronograma



## Apéndice 9.2 Declaraciones de entidad de cada archivo VHDL

```
entity etapa_IF  IS
     port
       (Clk          : in    BitType   := '1';
        Dir          : out   DataType  := DataAllTriZ;
        Datos        : inout DataType  := DataAllTriZ;
        RW           : out   BitType   := '0';
        Req          : out   BitType   := '0';
        Hold         : in    BitType;    --Memory Hold. Detiene la tuberia
        PC_Src       : in    dosbits;    --Control Mux para proximo PC
        IF_Descarga  : in    BitType;    --Se vacia instruccion de
                                         -- registro canal
        IF_ID_Inst   : out   DataType;   --Instruccion encontrada camino a
                                         -- ID
        ID_Bifurca   : in    DataType;   --Dir de bifurcacion  calculada
                                         --en ID
        ID_Dir_Salto: in     DataType;   -- Dir de salto incondicional
        ID_Salto_Reg: in     DataType;   --Dir de salto por registro
        IF_ID_PC_Inc: out    DataType;   --PC actual + 4
        IF_ID_Write  : in    BitType;    --Control escritura IF/ID y PC
                                       -- para insertar burbuja
        ID_Jump_Add  : in    DataType;   -- Direccion de salto
        Reset        : in    BitType );
end entity;

Entity etapa_id IS  -- Etapa de decodificacion
  PORT (clk          : IN BitType   := '1';
       Hold          : IN BitType;
       IF_ID_PC_Inc  : IN DataType;
  IF_ID_Inst         : IN DataType;
  MEM_WB_Reg_Write   : IN BitType;  --control de escritura en banco
                                    --de registros
  MEM_WB_Reg_Destino : IN cincobits;  --registro a escribir
  MEM_WB_Dato        : IN DataType;  --datos de la etapa de Write Back
 ID_EX_Reg_fuente2   : INOUT cincobits; --dir registro fuente 2 para
                                        --operaciones R, dest para otras
                              --debe consultarse para riesgod e
                                        --datos por load word
  EX_Resultado       : IN DataType; --datos de la ALU
  EX_MEM_Resultado   : IN DataType; --Datos de etapa MEM-WB
  EX_MEM_Reg_Write   : IN BitType;
  EX_MEM_Reg_Destino : in cincobits; --Reg destino en etapa MEM-WB
  Ex_destino        : in cincobits; --registro de destino de
                                    -- operacion  ALU

  ID_EX_Reg_Write    : INOUT BitType;
  ID_EX_Mem_Write    : OUT BitType;
  ID_EX_Mem_Read     : INOUT BitType;  --debe compararse para riesgo de
                                             --datos por load word
  ID_EX_Mem_a_Reg    : OUT BitType;
  ID_EX_Op           : INOUT seisbits;
  ID_EX_Dato_1       : OUT DataType;
  ID_EX_Dato_2       : OUT DataType;
```

```
  ID_EX_Operando_Inm : OUT DataType;  --operando inmediato para ALU
     ID_EX_Reg_fuente1  : INOUT cincobits;
     PC_Src             : OUT dosbits;  --Control de PC proximo
     ID_Bifurca         : OUT DataType;  --Direccion de bifurcaciones
     ID_Dir_Salto     : OUT DataType;  --direccion de salto incondicional
     ID_Salto_Reg     : out DataType; --la direccion de salto en el
                                      --Registro RS.
    ID_EX_Reg_Destino : OUT cincobits;
    ID_EX_Reg_Dest    : out BitType;  --Cual sera el registro destino

    IF_Descarga      : out BitType;  --para vaciar pipe de IF
    IF_ID_Write      : out BitType;  --Control escritura de pipe de IF y
                                           --PC
    Reset            : IN BitType);

END Entity;

Entity etapa_ex IS
  PORT (clk               : IN BitType;
        Hold              : IN BitType;
      Reset          : IN BitType;
      EX_destino           : out cincobits;
      EX_Resultado         : out DataType; --señales para reenvio
                              --en etapa ID para brincos

  ID_EX_Mem_Write    : IN BitType;
  ID_EX_Mem_Read     : IN BitType;
  ID_EX_Mem_a_Reg    : IN BitType;
  ID_EX_Reg_Write    : IN BitType;
  ID_EX_Dato_1       : IN DataType;
  ID_EX_Dato_2       : IN DataType;
  ID_EX_Operando_Inm : IN DataType;
  ID_EX_Op           : IN seisbits;
  ID_EX_Reg_Destino  : IN cincobits;
  ID_EX_Reg_Fuente1  : IN cincobits;
  ID_EX_Reg_Fuente2  : IN cincobits;
  ID_EX_Reg_Dest     : IN BitType;
  MEM_WB_Reg_Destino : IN cincobits; --Vienen de estapa final de WB
  MEM_WB_Reg_Write   : IN BitType;
  EX_MEM_Resultado   : INOUT DataType; --Resultado de salida
  EX_MEM_Dato        : OUT DataType; --El dato que se escribe en un Store
  EX_MEM_Reg_Destino  : INOUT cincobits;
  EX_MEM_Mem_Write    : OUT BitType;
  EX_MEM_Mem_Read     : OUT BitType;
  EX_MEM_Mem_a_Reg    : OUT BitType;
  EX_MEM_Reg_Write    : INOUT BitType;
  WB_Dato_Sal         : IN DataType);

End Entity;

Entity etapa_mem_wb IS
  PORT (clk               : IN BitType;
        Dir               : OUT DataType   := DataAllTriZ;
        Datos             : INOUT DataType := DataAllTriZ;
```

```
        RW                  : OUT BitType;
        REQ                 : OUT BitType;
        Hold                : IN  BitType;
        Reset               : IN  BitType;

        EX_MEM_Resultado    : IN DataType;
        EX_MEM_Dato         : IN DataType;
        EX_MEM_Reg_Destino  : IN cincobits;
        EX_MEM_Mem_Write    : IN BitType;
        EX_MEM_Mem_Read     : IN BitType;
        EX_MEM_Mem_a_Reg    : IN BitType;
        EX_MEM_Reg_Write    : IN BitType;
        MEM_WB_Dato         : OUT DataType;
        MEM_WB_Reg_Destino  : OUT cincobits;
        MEM_WB_Reg_Write    : OUT BitType);

END entity;
```

## Apéndice 9.3 Listado de programa de prueba Sqrt.bin

```
    # Square root test program written for the single cycle non-pipelined
# version of the Mini_MIPS microprocessor.
      .text
      lw   $4,x
      jal  sqrt
      sw   $2,sqrtdata($0)
      sw   $3,rem($0)
loop:   beq  $0,$0,loop

# FUNCTION: int_sqrt(x), int_sqrt_remainder(x)
#
# INPUT PARAMETERS:
#   R4 = x
#
# OUTPUT:
#   R2 = int_sqrt(x)
#   R3 = int_sqrt_remainder(x)
#
# OTHER AFFECTED VARIABLES:
#   R5, R6
#
# ALGORITHM:
#   uneven = - 1
#   repeat
#    uneven = uneven + 2
#    x      = x - uneven
#   until (x < 0)
#   sqrt_remainder = x + uneven
#   sqrt = uneven DIV 2
#
# INTERNAL VARIABLES:
#   R6 = uneven
#
# EXPLANATION:
#   After iteration n, x will have the value X - 1+3+5+... = X - n**2
#   where X is the initial value of x.  When the loop terminates in
#   iteration N+1, we have that X - (N+1)**2 < 0 for the first time, so N
#   is our integer approximation to SQRT(X). To find N, we note that
#   'uneven' has just been subtracted from the value X - (N)**2 from the
#   previous iteration, so that
#
#     X - (N+1)**2 = X - N**2 - uneven =>
#     uneven = 2N + 1 =>
#     N = (uneven - 1) / 2 =>
#     int_sqrt(X) = N = uneven DIV 2
#
#   as 'uneven' is uneven! The square root remainder is the difference
#   between X and N**2. As x has the value X - N**2 - uneven after the last
#   iteration, we have:
```

```
#
#     int_sqrt_remainder = x + uneven

sqrt:   addi $6,$0,-1

top:  addi $6,$6,2
        sub  $4,$4,$6
      slt  $5,$4,$0
      beq  $5,$0,top
      add  $3,$4,$6
      srl  $2,$6,2
      jr   $31

# DATA SEGMENT

      .data
x:      .word 0x0000000000000005
sqrtdata:   .word 0x0000000000000006
rem:    .word 0x0000000000000000
#
```