ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

IMPLANTACIÓN DE UN PROCESADOR MIPS CON FINES DIDÁCTICOS

(Implementation of a MIPS processor for educational purposes)

Para acceder al Título de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN

Autor: Pablo López Martínez

Septiembre – 2016

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Director del PFC: Esteban Stafford Fernández
Título: "Implantación de un procesador MIPS con fines didácticos"
Title: "Implementation of a MIPS processor for educational purposes"

Presentado a examen el día:

Para acceder al Título de

INGENIERO TÉCNICO DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre):	Martínez Fernández, Carmen
Secretario (Apellidos, Nombre):	Stafford Fernández, Esteban
Vocal (Apellidos, Nombre):	San Martín, Carlos

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del PFC (Sólo si es distinto del Secretario)
Vº Bº del Subdirector	Proyecto Fin de Carrera Nº (a asignar por Secretaría)

Agradecimientos

Me gustaría expresar en estas líneas mi agradecimiento a todas aquellas personas que han estado a mi lado todos estos años, por su paciencia y comprensión.

En primer lugar, quiero expresar mi agradecimiento al director de este trabajo de fin de carrera, Esteban Stafford Fernández, por el apoyo y dedicación que me ha brindado en estos últimos meses en la realización de este trabajo.

Llegar a realizar un trabajo como este también es fruto del apoyo que nos ofrecen las personas que nos estiman, que nos dan fuerza y energía día a día para que podamos llegar a conseguir nuestros objetivos. Por ello quiero transmitir un agradecimiento especial a mi familia, que me ha brindado el apoyo necesario para logar esta meta.

Y por supuesto, a todos mis compañeros de clase de todos estos años, gracias por vuestro apoyo y por todos esos buenos momentos que hemos pasado juntos.

Muchas Gracias.

Tabla de contenido

“IMPLANTACIÓN DE UN PROCESADOR MIPS CON FINES DIDÁCTICOS”

Palabras Clave:

MIPS, Procesador Monociclo, Procesador Segmentado, Arquitectura de Von Neumann, Docencia, Arquitectura de computadores, TkGate, Simulación

Capítulo I. Introducción

1.1 Motivación

El procesador MIPS es ampliamente utilizado en el mundo de la docencia de arquitectura de computadores. Sin embargo, el estudio detallado de su implementación interna requiere herramientas muy complejas cuyo aprendizaje se escapa a los objetivos de asignaturas básicas. Actualmente el diseño de los procesadores hace muy compleja su comprensión y funcionamiento a nivel interno, por ello se considera de gran utilidad el uso de un simulador que permita ver el proceso de ejecución en un microprocesador MIPS sencillo. Otro aspecto deseable es el de poder realizar cambios en el diseño enfocado a el estudio de la arquitectura.

1.2 Objetivos

En este proyecto se pretende hacer una implementación de un procesador MIPS sencillo sobre una herramienta de simulación de circuitos lógicos visualmente atractiva. Para ello se considerará que subconjunto del estándar MIPS es conveniente implementar para poder desarrollar ejercicios y experimentos típicos de la docencia de la arquitectura de computadores. Así mismo será necesario evaluar diferentes herramientas de simulación teniendo en cuenta su facilidad de uso y su capacidad de simular sistemas de una complejidad significativa.

También se propondrán implementaciones de procesadores MIPS que sean capaces de ejecutar las instrucciones anteriores. Concretamente se pretende acometer la implementación de un procesador monociclo y otro segmentado, ya que son los más típicos en la docencia de arquitectura de computadores.

Finalmente, el proyecto demostrará la capacidad didáctica de las soluciones adoptadas a través de la resolución de varios problemas típicos de arquitectura.

Capítulo II. Conceptos fundamentales y herramientas

2.1 Las primeras computadoras

Las primeras máquinas de computación estaban diseñadas para programas fijos y cambiar su programa era un proceso laborioso de notas, diagramas, diseño y reconstrucción o recableado físico de la máquina (Figura 2.1.1).

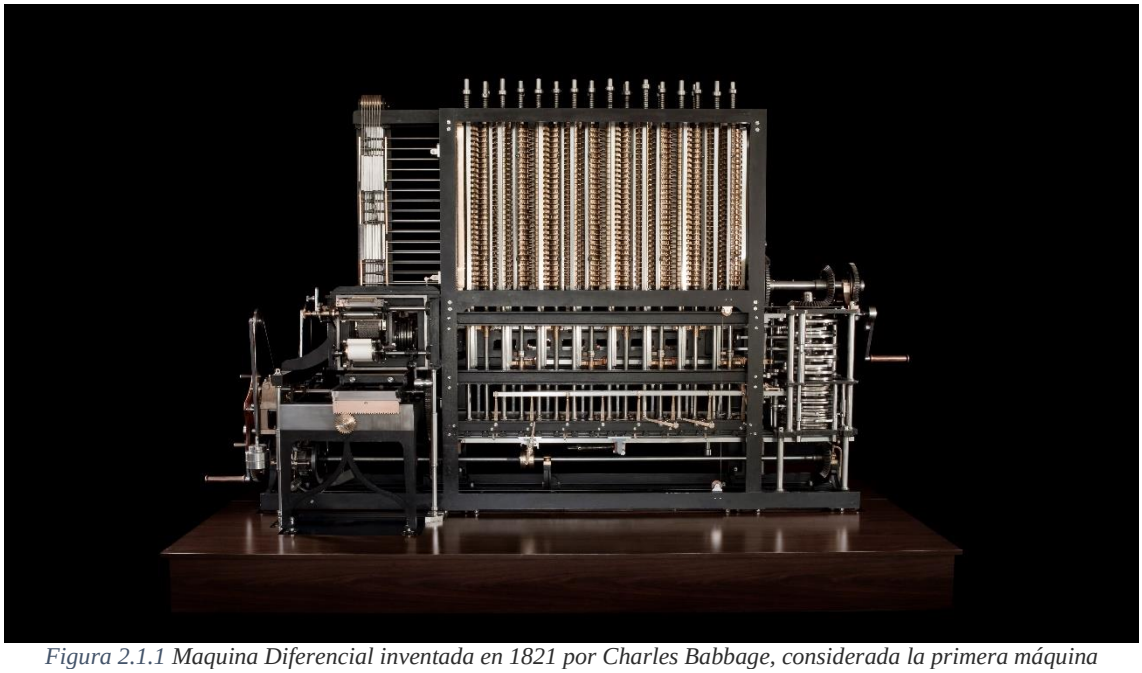

Figura 2.1.1 Maquina Diferencial inventada en 1821 por Charles Babbage, considerada la primera máquina computacional automática capaz de calcular funciones polinómicas usadas en aplicaciones generales de matemáticas e ingeniería.

Ante este problema se planteó el diseño de una computadora con programas almacenados en memoria que contaba con un conjunto de instrucciones que lo interpretaban, lo que permitía la modificación del programa original sin rediseñar la máquina.

Aunque siguen extendiendo algunos equipos específicos de programa fijo, con el paso del tiempo el concepto de una computadora de programa almacenado ha evolucionado y se ha perfeccionado, siendo el adoptado por la inmensa mayoría de computadoras.

2.2 Arquitectura de Von Neumann

En el año 1944 el matemático y físico John Von Neumann estaba involucrado en el Proyecto Manhattan en el Laboratorio Nacional Los Álamos (Nuevo Méjico, EE.UU.), el cual requería grandes cantidades de cálculos. Esta circunstancia le condujo al proyecto desarrollado por J. Presper Eckert y John Mauchly conocido como **ENIAC** (del inglés ***E**lectronic **N**umerica **I**ntegrator **A**n* **C**omputer, Figura 2.2.1). La primera computadora de propósitos generales, la cual funcionaba según el sistema decimal y podía ser reprogramada para resolver diferentes clases de problemas numéricos. Allí se incorporó a los debates sobre el diseño de una computadora con programas almacenados, el

EDVAC (del inglés ***E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer*) que a diferencia de la ENIAC era binaria y tuvo el primer programa diseñado para ser almacenado. Este diseño se convirtió en estándar de arquitectura para la mayoría de las computadoras modernas.

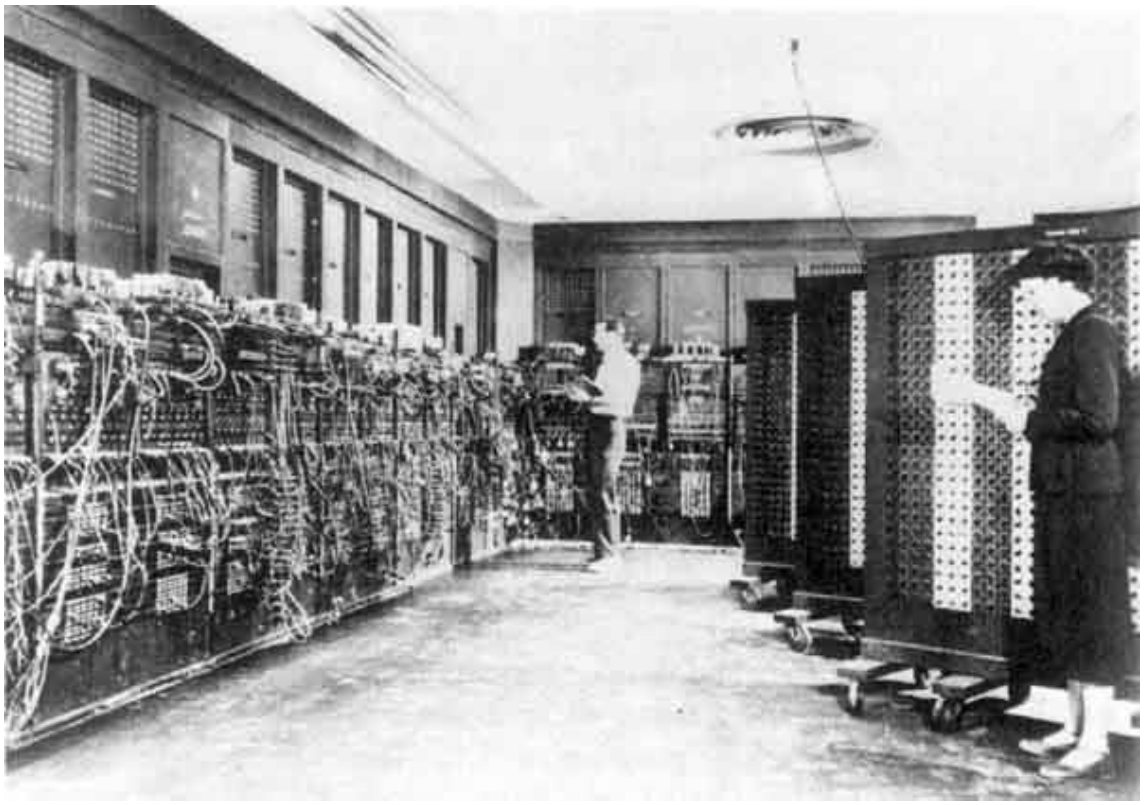

Figura 2.2.1 ENIAC (Electronic Numerical Integrator And Computer) en Philadelphia, Pennsylvania. Glen Beck (fondo) y Betty Snyder (derecha) programan la ENIAC en el edificio 328 en el Ballistic Research Laboratory (BRL).

En el año 1945 se publicó el artículo "First Draft of a Report on the EDVAC" que, aunque incluía ideas de Eckert y Mauchly, solo contenía el nombre de Von Neumann y fue el que dio origen al término "arquitectura de Von Neumann".

La arquitectura Von Neumann, también conocida como modelo de Von Neumann, describe el fundamento de todo ordenador electrónico con programas almacenados, mediante una arquitectura de diseño para un computador digital electrónico con sus unidades conectadas permanentemente y que consta de varias partes:

- Unidad Central de Procesamiento (**CPU**, del inglés **C**entral **P**rocessing **U**nit), contiene la unidad aritmético lógica (**ALU**, del inglés ***A**rithmetic **L**ogic **U**nit*), los registros del procesador y la unidad de control (**CU**, del inglés **C**ontrol **U**nit) que contiene el registro de instrucciones y el contador de programa (Figura 2.2.2).
- Memoria para almacenar tanto datos como instrucciones. Una computadora digital de programa almacenado tiene sus instrucciones de programa y sus datos en una memoria de acceso aleatorio (**RAM**, del inglés ***R**andom **A**ccess **M**emory*) de lectura y escritura.
- Mecanismos de entrada y salida (**I/O** del inglés ***I**nput/**O**utput*).

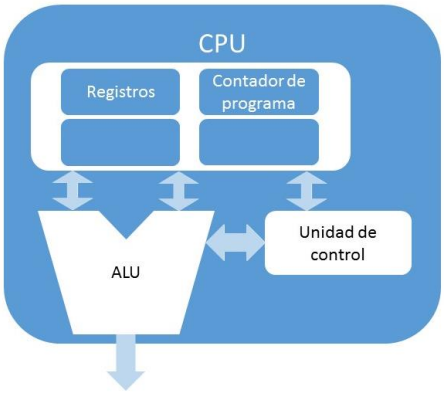

Figura 2.2.2 Diagrama de una CPU en la arquitectura de Von Neumann.

Antes de continuar con los detalles de las diferentes partes de un procesador deben conocerse los siguientes conceptos:

- **Registros**, es el lugar donde se almacenan temporalmente los datos para procesarlos.
- **Buses**, son los encargados de unir las diferentes unidades y distribuir los datos entre la memoria y los periféricos. Hay tres tipos de buses:
    - **Bus de datos,** se encarga del intercambio de datos o instrucciones entre los diferentes elementos de la computadora. Por ejemplo, mediante el bus de datos la ALU y los elementos de I/O reciben y envían los datos desde la memoria.
    - **Bus de direcciones,** envía las direcciones de memoria que van a ser desde la CPU para poder seleccionar los datos necesarios.
    - **Bus de control,** envía las ordenes de la Unidad de Control para controlar cada uno de los componentes del sistema.

El término ha cambiado hasta referirse a cualquier computador de programa almacenado en el que no puede leerse una instrucción y realizarse una operación de datos al mismo tiempo, ya que comparten un bus común (Figura 2.2.3). Esto se crea el efecto conocido como el cuello de botella Von Neumann, que es el responsable de limitar el rendimiento del sistema en muchas ocasiones.

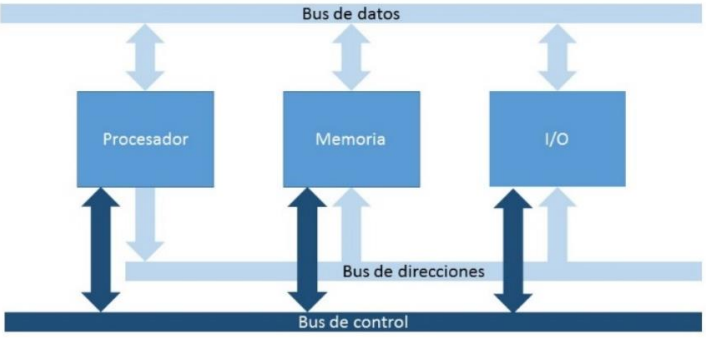

Figura 2.2.3 Diagrama de buses en la arquitectura de Von Neumann.

Este problema de rendimiento puede ser minimizado por medio de diversos métodos, por ejemplo, ofreciendo una memoria caché entre la CPU y la memoria principal, proporcionando cachés separadas o vías de acceso independientes para datos e instrucciones (arquitectura Harvard).

La arquitectura Harvard es también un sistema de programa almacenado, pero posee un diseño más moderno con una unidad de memoria para instrucciones y otra para los datos. Además de esto también incorpora un conjunto dedicado de direcciones y buses de datos para leer datos desde memoria y escribir datos en la misma, y otro conjunto de direcciones y buses de datos para ir a buscar instrucciones.

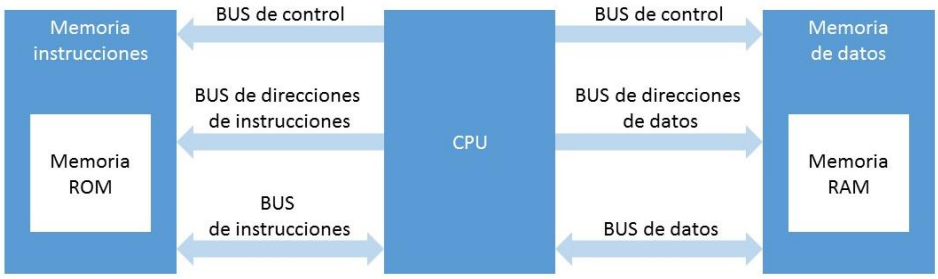

Figura 2.2.4 Diagrama de buses y unidades de la arquitectura Harvard

En la gran mayoría de las computadoras modernas se utiliza la misma memoria tanto para datos como para instrucciones. La distinción entre Von Neumann y Harvard se aplica a la arquitectura de memoria caché, pero no a la memoria principal. De esta forma es la memoria caché la que se divide en una de datos y otra de instrucciones (Figura 2.2.5).

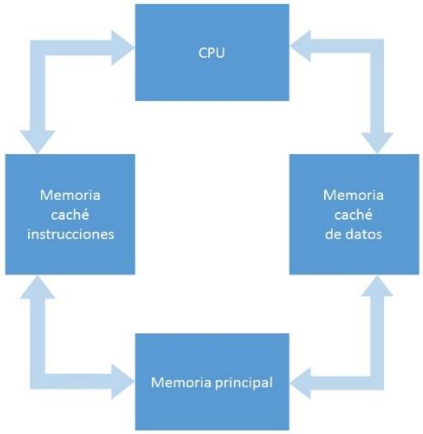

Figura 2.2.5 Separación de la memoria principal y las memorias caché en la arquitectura Harvard.

En este proyecto se simulará un procesador de arquitectura Harvard sencillo, sin una memoria principal conjunta para datos e instrucciones.

2.2.1 Partes fundamentales

Cada uno de los componentes principales tiene una función definida que se detalla a continuación:

Unidad de proceso central (CPU)

Se encarga de controlar todo el sistema de la computadora y contiene la unidad aritmético lógica, los registros del procesador y la unidad de control que contiene el registro de instrucciones y el contador de programa.

Unidad de control (CU)

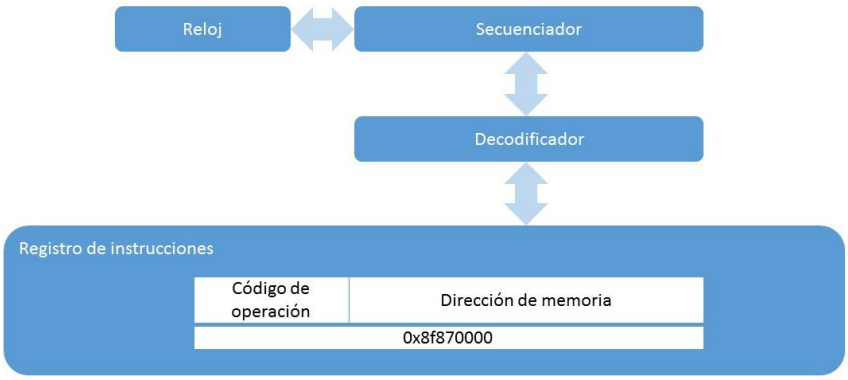

Figura 2.2.6 Diagrama de la Unidad de Control.

Es un circuito combinacional que se encarga de interpretar las instrucciones de los programas almacenados en la memoria y enviar las ordenes a los diferentes componentes de la computadora para que las ejecuten (Figura 2.2.6).

El proceso de ejecución empieza cuando llega una instrucción al registro de instrucciones (una cadena de bits que se divide en distintas partes, que contienen la propia instrucción y los datos o argumentos que se usarán). Posteriormente el decodificador interpreta la instrucción a realizar y como deben de actuar los componentes del procesador para llevarla a cabo.

La unidad de control contiene el registro contador de programa, que guarda la dirección de memoria de la siguiente instrucción y se incrementa tras realizar una instrucción. Con la ayuda de este registro la unidad de control va recorriendo el programa en la memoria.

Unidad Aritmético Lógica (ALU)

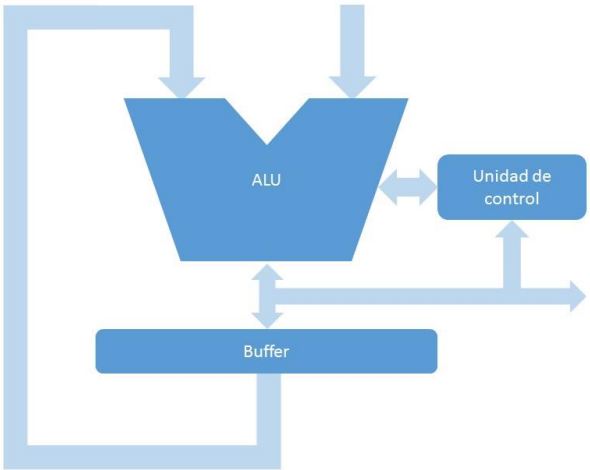

Figura 2.2.7 Diagrama de una ALU sencilla.

Es la encargada de realizar todas las operaciones aritméticas (sumas, multiplicaciones...) y lógicas (comparaciones, AND, OR, …).

Esta unidad puede tener distintos diseños, en la imagen (Figura 2.2.7) se muestra un diseño básico. En él se puede ver una Unidad de Control, que envía a la ALU las

operaciones que debe realizar, junto con un buffer o acumulador que guarda el resultado saliente de la ALU y vuelve a introducirlo en esta como un nuevo operando.

Memoria principal

En la arquitectura inicial de Von Neumann es la RAM, pero en las nuevas computadoras se han añadido memorias caché con algoritmos capaces de predecir qué datos van a ser usados más frecuentemente. Como se puede ver en la Figura 2.2.5, se implementan dos memorias caché, una para datos y otra para instrucciones, entre la memoria principal (contiene datos e instrucciones) y la CPU. En este proyecto se obvian estos detalles y de la jerarquía de memoria usando solo dos memorias, una para datos y otra para instrucciones.

La memoria es una tabla (Figura 2.2.8) los datos e instrucciones. La memoria dispone de un registro de direcciones de memoria (RDM) y un registro de intercambio de memoria (RIM). En el registro de direcciones se almacena la dirección en la que se guardará o leerá un dato, y en los registros de datos se encontraran los datos leídos o que se quieren guardar.

Figura 2.2.8 Representación de una tabla de memoria.

Dispositivo de Entrada/Salida (I/O)

Son los dispositivos auxiliares conectados a la CPU de una computadora. Se definen como elementos que permiten realizar operaciones de entrada/salida complementarias al procesado de datos que realiza la CPU y que no pertenecen a esta.

2.3 El repertorio de instrucciones (ISA)

El funcionamiento de la CPU viene determinado por las instrucciones que es capaz de ejecutar, a las cuales se denomina instrucciones máquina.

Cada una de estas instrucciones máquina contiene la información que necesita la CPU para llevarla a cabo está formada por los siguientes campos:

- Código de operación, indica la instrucción a realizar (resta, suma, lógica, E/S, acceso a memoria, …).
- Operandos fuente, indica el operando u operandos de la instrucción. Son elementos de entrada.
- Operando resultado, se necesita indicar que hacer con el resultado
- Referencia a la siguiente instrucción, indica a la CPU donde se encuentra la siguiente instrucción después de la ejecución de la instrucción actual. Por normal general la siguiente instrucción se encuentra después de la instrucción en ejecución, por lo que no es necesario hacer una referencia concreta. Pero existen casos en los que la siguiente instrucción no es la siguiente y es necesaria una referencia que permita a la CPU encontrar su ubicación.

Los operandos fuente y resultado pueden estar en la memoria principal, en los registros de la CPU o en un dispositivo de entrada/salida.

Al conjunto de estas instrucciones se le denomina repertorio de instrucciones (**ISA,** del inglés ***I**nstruction **S**et **A**rchitecture*). El ISA se encarga de detallar todas las instrucciones máquina que una CPU es capaz de entender y ejecutar, y describe otros aspectos como los datos nativos, los registros, la arquitectura de la memoria y las interrupciones, entre otros aspectos.

El repertorio de instrucciones no está ligado al diseño de un microprocesador concreto, de esta forma dos procesadores con diferente diseño pueden compartir un repertorio casi idéntico.

Existen dos tipos de ISA: **CISC** (del inglés ***C**omplex **I**nstruction **S**et **C**omputer*) y **RISC** (del inglés ***R**educed **I**nstruction **S**et **C**omputer*).

2.3.1 Computador con Conjunto de Instrucciones Reducidas (RISC)

Define un modelo de procesador con instrucciones simples y pequeñas que se ejecutan rápidamente y en el que solo las instrucciones de lectura y escritura acceden a la memoria. Debido a esto se necesitan más líneas de código para escribir un mismo programa que en un modelo CISC.

Con este tipo de diseño se pretende reducir los tiempos de ejecución de las instrucciones y su optimización, por ejemplo, con el paralelismo y la segmentación. Un ejemplo de procesador RISC es la familia ARM usada en la mayoría de dispositivos móviles en la actualidad.

RISC tiene la ventaja de que permite incrementar la frecuencia de reloj respecto al CISC.

2.3.2 Computador con Conjunto de Instrucciones Complejas (CISC)

Describe un modelo de procesador con una amplia variedad instrucciones complejas que permiten operaciones con operandos que pueden estar en la memoria o en los registros. Debido a esto un mismo programa necesitará un menor número de líneas de código que en un modelo RISC.

Debido a su arquitectura dificulta las operaciones en paralelo por lo que la mayoría de procesadores CISC de actualidad contienen un sistema que convierte sus instrucciones en varias más simples de tipo RISC.

Los CISC pertenecen a la primera corriente de construcción de procesadores, antes del desarrollo de los RISC. Un ejemplo es toda la familia Intel x86 usada en la mayoría de las computadoras personales actuales.

CISC tiene la ventaja de que permite una mayor compactación de código.

2.4.1 Implementación de procesadores

Existen varias implementaciones de procesadores, pero en este proyecto se centrará en el procesador monociclo y segmentado, y más concretamente en el procesador MIPS DLX dado su valor didáctico.

El procesador monociclo

Es el modelo de procesador más básico. En el procesador monociclo el período del reloj se ajusta con la ruta establecida por la instrucción que más tarda en ejecutarse por completo (Figura 2.4.1). El periodo de reloj se mantiene constante, de manera que todas las instrucciones tardan lo mismo, independientemente de su complejidad.

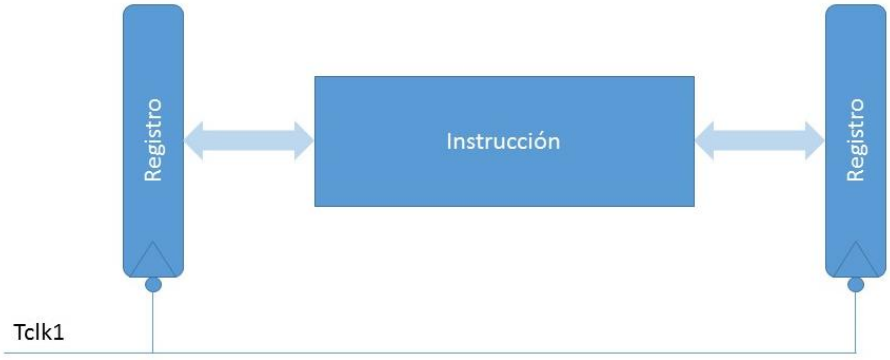

Figura 2.4.1 Diagrama del tiempo de instrucción de un diseño monociclo, donde el ciclo de reloj se ajusta con la instrucción.

En este diseño los recursos (Memoria, ALU, ...) solo pueden utilizarse una vez por cada ciclo del reloj.

El procesador segmentado

La segmentación (*pipelining*) es una técnica de implementación de procesadores que basándose en que una instrucción utiliza varias partes de la CPU, pero no al mismo tiempo, permite introducir nuevas instrucciones en la CPU, cada una en una fase diferente de ejecución de manera solapada. Esto posibilita comenzar una nueva instrucción sin que la anterior haya terminado aún.

La segmentación divide la ejecución de cada instrucción en varias etapas conectadas de forma que la salida de una es la entrada de la siguiente. Para ellos se introducen unos

registros de segmentación entre ellas (Figura 2.4.2). Un detalle importante es la capacidad de ejecutar cada una de las tareas, aunque estas tengan diferentes datos, lo que se consigue mediante la separación, con registros, de las etapas.

Esta división en etapas permite reducir considerablemente el periodo de reloj haciendo que, aunque una sola instrucción se ejecute más despacio, el procesador es capaz de ejecutar un mayor número de instrucciones por segundo que un monociclo. Con ello se mejora la velocidad de ejecución cuando hay más de unas pocas instrucciones.

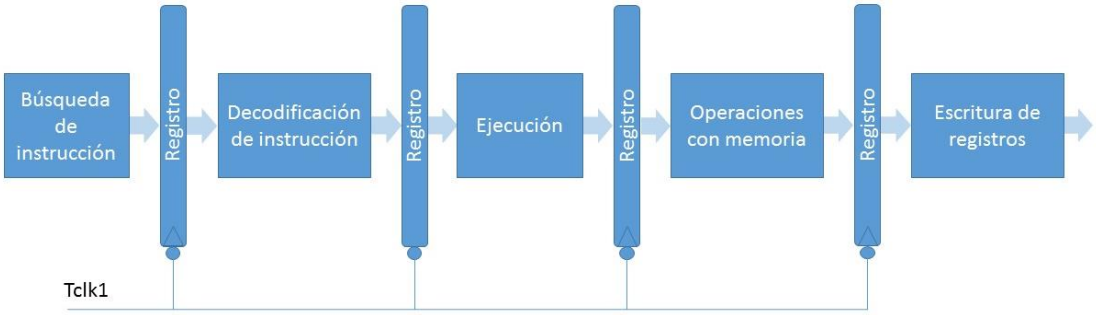

Figura 2.4.2 Esquema de etapas de la segmentación.

Otro aspecto a tener en cuenta en este tipo de procesadores para que su tiempo de latencia sea el mínimo posible es el equilibrio de del tiempo de ejecución de cada etapa. De esta forma se evita que haya partes del proceso que vayan más rápido que otras y tengan que estar a la espera sin realizar ningún trabajo, disminuyendo de esta manera el rendimiento.

Procesador superescalar

Se denomina arquitectura superescalar a aquella capaz de ejecutar, de manera independiente, más de una instrucción por ciclo de reloj. Para ello se usan múltiples cauces, con lo que varias instrucciones pueden iniciar su ejecución de manera independiente siempre que no presenten algún tipo de dependencia. El diseño superescalar es compatible con la segmentación como se puede ver en la Figura 2.4.3.

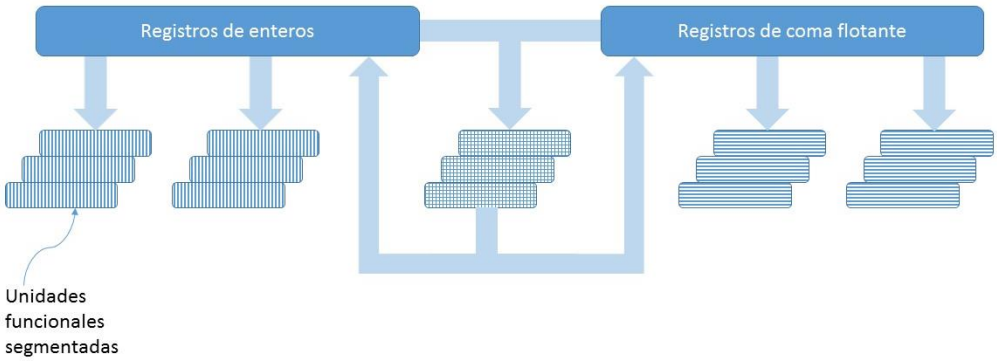

Figura 2.4.3 Diagrama de un procesador superescalar.

Aunque se puede llevar a cabo una implementación superescalar mediante una arquitectura CISC este concepto no se planteó hasta poco después de la aparición de la arquitectura RISC, la cual facilita este tipo de diseños.

Desde 1998 casi todas las CPUs desarrolladas son superescalares, pero en este proyecto se centra en los procesadores monociclo y segmentado dado su orientación didáctica.

2.4.2 El procesador MIPS

El acrónimo MIPS (del inglés ***M**icroprocessor without **I**nterlocked **P**ipeline **S**tages*) representa a la familia de procesadores de arquitectura RISC desarrollados por la empresa MIPS Technologies.

Aunque existen numerosos tipos de procesadores MIPS con diferentes capacidades y usos, este proyecto se centrará en las características del procesador MIPS DLX, un modelo simplificado de carga y almacenamiento usado, principalmente, con fines educativos en asignaturas de arquitectura de computadores.

Este procesador se compone de 32 registros (R0 a R31) de 32 bits para los cuales, a pesar de ser de propósito general, existe una convención del uso que se da a cada uno (Tabla 2.4.4). Esta convención es una sugerencia no obligatoria, pero si no se sigue es posible que provoque fallos en el uso de un programa en sistemas diferentes al original o cuando se haga uso de librerías estándar o de terceros.

El repertorio de instrucciones del MIPS DLX se puede dividir en tres tipos de instrucciones:

opcode	rs	rt	rd	sa	function
31 26	25 21	20 16	15 11	10 6	5 0
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Tipo R					

Tabla 2.4.1 Tabla de codificación de una instrucción de tipo R.

Las instrucciones de tipo R (Tabla 2.4.1) están formadas por el código de operación más los registros de los operandos y de salida, también posee el campo “function” que es el encargado de identificar el tipo de operación a realizar. Se caracteriza por tener dos argumentos y un destino.

opcode	rs	rt	offset
31 26	25 21	20 16	15 0
6 bits	5 bits	5 bits	16 bits
Tipo I			

Tabla 2.4.2 Tabla de codificación de una instrucción de tipo I.

Las instrucciones de tipo I (Tabla 2.4.2) están formadas por el código de operación más el registro operando y el valor inmediato. En este caso también se caracteriza por tener un destino y dos argumentos, solo que en este caso uno es un valor inmediato.

Las instrucciones de tipo J (Tabla 2.4.3) están formadas por el código de operación más el registro operando y el campo de dirección. Se caracteriza por usarse para instrucciones de salto que necesitan argumentos inmediatos muy largos.

opcode	instr_index
31 26	25 0
6 bits	26 bits
Tipo J	

Tabla 2.4.3 Tabla de codificación de una instrucción de tipo J.

En cuanto a su implementación se puede considerar la estrategia monociclo, aunque el DLX se suele utilizar también para estudiar la segmentación. En este caso la ejecución de una instrucción se divide en las siguientes etapas (Figura 2.4.4):

- IF (del inglés ***I**nstruction **F**etch*), búsqueda de instrucción.
- ID (del inglés ***I**nstruction **D**ecode*), decodificación de instrucción y lectura de registros. En esta etapa se extrae el código de operación y los operandos de la etapa anterior (IF). También se leen los valores de los registros si es necesario.
- EX (del inglés ***Ex**ecute*), ejecución. En esta etapa se ejecutan las operaciones aritmético lógicas.
- MEM (del inglés ***Mem**ory*), operaciones con memoria de datos. Se obtienen los datos de la memoria controlada por las etapas ID y EX.
- WB (del inglés ***W**rite **B**ack*), escritura en el banco de registros.

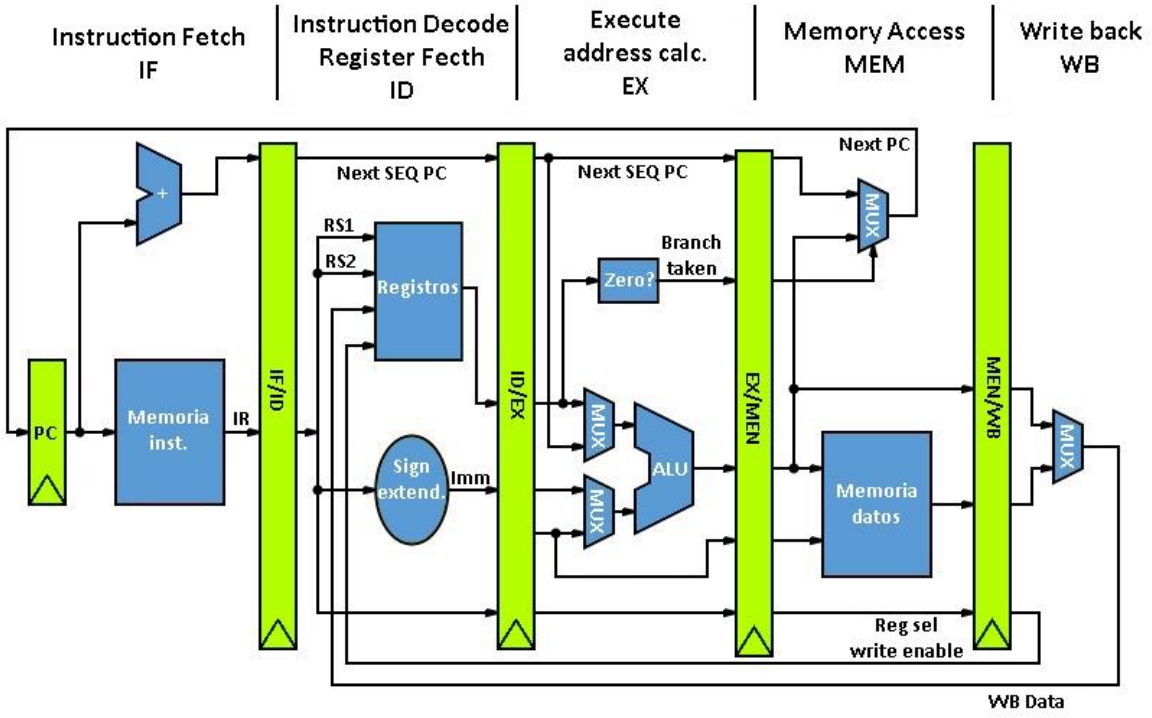

Figura 2.4.4 Esquema de segmentación de MIPS

Número de registro	Nombre	Descripción
$0	$zero	Constante de valor 0
$1	$at	Reservado para el ensamblador
$2, $3	$v0, $v1	Valores de retorno de subprogramas
$4 - $7	$a0 - $a3	Argumentos para funciones
$8 - $15	$t0 - $t7	Temporales, no se conserva en llamadas a funciones
$16 - $23	$s0 - $s7	Temporales, se conserva en llamadas a funciones
$24, $25	$t8, $t9	Temporales, no se conserva en llamadas a funciones
$26, $27	$k0, $k1	Reservado para el núcleo (*Kernel*) del SO
$28	$gp	Puntero a la zona global de datos
$29	$sp	Puntero de pila
$30	$fp	Frame Pointer
$31	$ra	Dirección de retorno (usado en la instrucción jal)

Tabla 2.4.4 Tabla de convención de registros del MIPS

2.5 El simulador TkGate

Para la elección del simulador se ha tenido en cuenta que en este proyecto se pretende simular la arquitectura interna de un procesador MIPS. Para ello conviene estudiar diferentes tipos de simuladores. Según su nivel de detalle se pueden destacar los siguientes:

- Simuladores circuitales, simulan las ecuaciones diferenciales que definen los circuitos analógicos. Aunque se pueden usar para circuitos digitales sencillos su uso no es recomendable para simular un procesador. Un ejemplo de este tipo de simulador es PSPICE.
- Simuladores de transferencia de registros, son comunes en las industrias de fabricación digitales, sin embargo, su coste suele ser elevado y por ello se han descartado. Un ejemplo de este tipo de simulador es ModelSim.
- Simuladores lógicos, similares a los anteriores, existen varias alternativas de bajo coste o código abierto, por ejemplo, LogicWorks y TkGate. Estos parecen ideales para el desarrollo de del proyecto, gracias a su interfaz de usuario gráfico. Sin embargo, TkGate tiene una ventaja considerable sobre LogicWorks al permitir especificar los circuitos mediante el lenguaje de descripción Verilog y su distribución gratuita.
- Simuladores funcionales, estos implementan la ISA de un procesador, pero ignoran los detalles de su arquitectura o no permiten su modificación. Esto los hace menos atractivos para la docencia de la arquitectura de computadores. Un ejemplo de este tipo de simulador es PCSpim o MIPSIT.

Así pues, se ha escogido TkGate ya que permite una edición sencilla a nivel de hardware y la simulación de un circuito mediante señales digitales.

TkGate es un simulador de circuitos digitales escrito en C y FCL, usando la librería TK para la interfaz de usuario, que se distribuye como software libre bajo la licencia pública GPL. Soporta una amplia gama de elementos de circuitos como puertas lógicas, sumadores, multiplicadores, registros, memorias, etc., así como módulos complejos definidos por el usuario como ALUs o Unidades de Control. También permite asignar

memorias a archivos binarios para las simulaciones. TkGate incluye una serie de tutoriales de circuitos y ejemplos que se pueden cargar desde el menú "Ayuda".

2.5.1 Características TkGate

Entre sus características se pueden destacar las siguientes:

- Diseño gráfico de circuitos:
  - Diseño jerárquico a través de módulos definidos por el usuario.
  - Interfaz fácil de usar.
  - Creación de hiperenlaces para moverse en el circuito o cargar ficheros.
  - Interfaz multidioma.
  - Formato de fichero como verilog (*.v).

- Simulador lógico:
  - Control a través del interfaz o a través de ficheros scripts.
  - Apropiado para la simulación a nivel de transistor, puerta o registro.
  - Seis modelos de valores lógicos incluyendo 0, 1, flotante, desconocido, "bajo" y "alto".
  - Soporte de modelos de retraso personalizados.
  - Ventana gráfica de resultados de simulación.
  - Puntos de control, control de simulación por reloj.
  - Análisis estático de pasos críticos.

- Elementos de circuitos incluidos:
  - Puertas básicas (AND, OR, etc.)
  - Transistores NMOS y PMOS.
  - Buffers triestado.
  - Componentes para ALU (sumadores, desplazadores, multiplicadores).
  - Elementos de memoria (registros, RAMs, ROMs).
  - Elementos interactivos que permiten al diseño de circuito interactuar con el usuario.

- Herramientas de soporte para lenguaje ensamblador que permiten generar código usable por las memorias y en los circuitos a simular.

2.5.2 Opciones de ejecución desde la línea de comandos

La sintaxis para la ejecución es la siguiente:

tkgate [-xqs] [-X script] [-l file] [-L lang] [-P printer] [-p file] [files ...]

Los usos de las diferentes opciones son:

- -X script, comienza el simulador de forma automática y ejecuta el script de simulación específico.
- -l file, lee el fichero especificado como si fuera una librería.
- -x, arranca de forma automática el simulador.
- -q, suprime los mensajes de arranque.

- -P printer, imprime el circuito especificado sin necesidad de arrancar el interfaz de usuario.
- -p file, imprime en un fichero sin necesidad de arrancar el interfaz de usuario.
- -L lang, especifica el lenguaje de la interfaz.

Es importante que los nombres de las carpetas donde se guardan los datos de TkGate no estén espaciados, ya que tiene problemas para reconocerlos y puede provocar errores.

2.5.3 Interfaz de TkGate

La interfaz principal de TkGate (Figura 2.5.1) consiste en una barra de menú superior, una sección con los módulos (Tree, List), las redes y los puertos en la parte izquierda, una barra de estado en la parte inferior y la ventana principal de edición gráfica. Las barras de desplazamiento pueden usarse para desplazarse por los listados y por la ventana de edición.

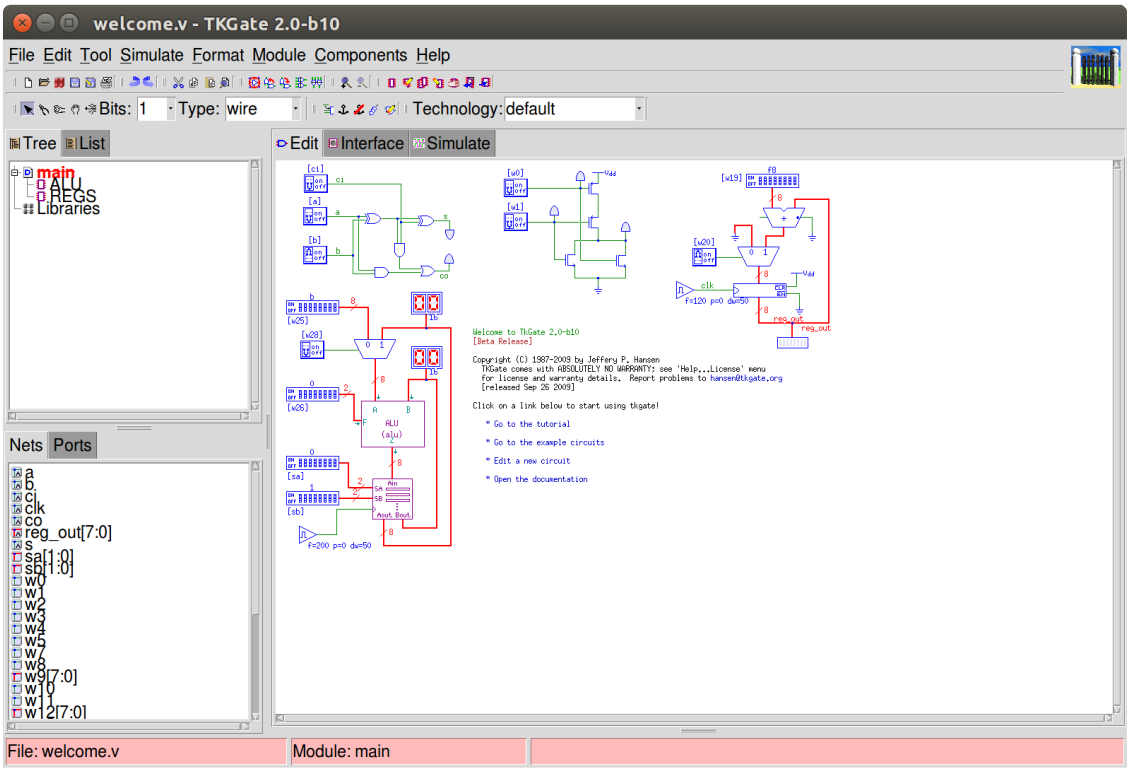

Figura 2.5.1 Interfaz principal de TkGate.

La ventana "Tree" muestra, de manera jerárquica, los módulos que son parte del circuito que está siendo editado. Existen dos nodos principales, en la sección "main" se muestran los módulos en uso, y en caso de haber módulos sin usar los muestra en la sección "unused". El módulo superior es el que tiene un signo + ó – a su izquierda, botón con el cual se puede extender u ocultar el listado de módulos. La ventana "List" muestra un listado con los diferentes módulos.

Debajo de la sección de módulos se muestra un listado con las redes, "Nets", del circuito. En las redes con varios bits se indica el rango de bits a su derecha. Los nombres de cada red pueden estar visibles u ocultos en la ventana de edición, para cambiar esto se debe

hacer click con el botón derecho sobre el nombre de red que se desea mostrar u ocultar e indicar la opción elegida en el menú desplegable.

En la esquina superior derecha de la ventana de edición gráfica están los diferentes modos:

- Edit, es el modo de edición.
- Interface, para editar la interfaz de los módulos.
- Simulate, inicia el simulador.

La mayoría de opciones frecuentes (guardar, abrir, imprimir, control del simulador, …) pueden encontrarse en la barra de herramientas superior de TkGate.

La barra de estado de la parte inferior indica el fichero que está siendo editado y el módulo actual del fichero que está siendo mostrado en la ventana de circuito. Un "*" tras el nombre del fichero indica que el circuito ha sido modificado desde la última vez que fue guardado. La sección más a la derecha muestra información referente al objeto seleccionado en ese momento.
Muchos de los elementos del interfaz muestran un mensaje de ayuda que aparece colocando el puntero del ratón sobre ellos.

2.5.4 Controles básicos (Abrir, guardar, imprimir, …)

Las funciones de abrir y guardar circuitos además de las opciones de impresión se encuentran en la pestaña "File" de la barra de herramientas. También se puede acceder a estas opciones desde los botones correspondientes del menú ().

TkGate también dispone de otras opciones básicas como cortar, pegar, deshacer, rehacer, alinear, rotar… a las que se puede acceder desde la pestaña "File" de la barra de herramientas. También se puede acceder a estas opciones desde los botones correspondientes del menú ().

2.5.5 Opciones de TkGate

TkGate tiene varias opciones (Figura 2.5.2) que se pueden configurar accediendo desde la ruta de la barra de herramientas Tool > Options. Los cambios en las opciones de configuración son permanentes siendo guardados para los próximos inicios de TkGate.

Las opciones básicas son:

- "General", permite configurar las opciones generales tales como nombre de usuario y organización u opciones de guardado de los diseños.
- "Interface", permite escoger varias opciones de la interfaz como por ejemplo los mensajes de ayuda al poner el puntero del ratón sobre un botón.
- "Toolbars", permite escoger las barras de herramientas mostradas en los diferentes modos de TkGate.
- "HDL", permite escoger algunas opciones del editor.
- "Print", permite escoger las opciones de impresión por defecto: escala, tipo de hoja, doble cara, …

- "Simulate", permite escoger varias de las opciones de simulación, más adelante se explican algunas de las más relevantes.
- "Libraries", permite escoger una lista de directorios para buscar librerías de TkGate.
- "Security", permite seleccionar opciones sobre los cambios que puede realizar el usuario en archivos a los que tiene acceso TkGate.
- "Color", permite escoger los colores con los que se representaran varios de los elementos en la ventana de edición gráfica.
- "HTML", permite escoger con que programa abrir links html o de envío de emails.

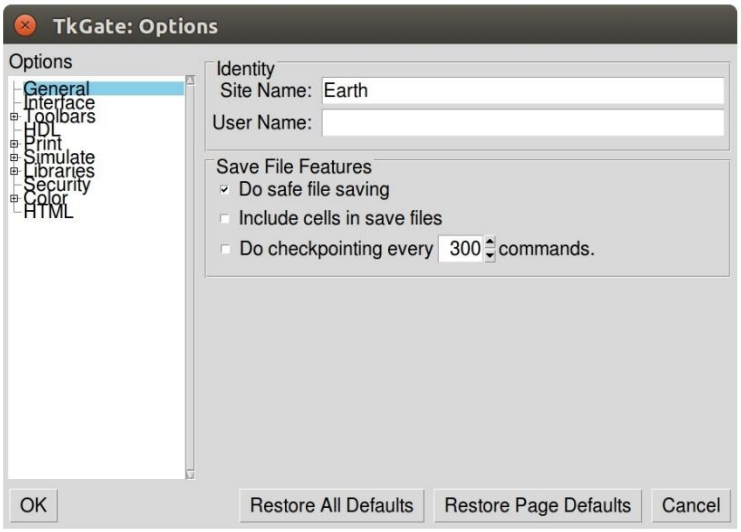

Figura 2.5.2 Ventana de opciones de TkGate.

2.5.6 Herramientas de edición básicas

Las herramientas básicas disponibles en TkGate para la edición son las siguientes:

- "Move/connect" (), esta herramienta se usa para el mayor número de funciones: creación de puertas, conexión de cables, movimiento de puertas y cables, …. Pulsando con el botón derecho del ratón sobre un elemento del circuito se puede eliminar o modificar sus propiedades. Su tecla de acceso rápido es F1.
- "Connect" (), aparece al mantener pulsado el botón izquierdo del ratón sobre un cable y permite extender al cable y conectarlo a otras partes del circuito. Su tecla de acceso rápido es F1.
- "Cut wire" ( ), se usa para cortar cables y su tecla de acceso rápido es F2.
- "Invert" (), esta herramienta sirve para añadir o quitar inversores de los puertos de los componentes. Su tecla de acceso rápido es F3.
- "Scroll" (), pincha y arrastra para desplazar la ventana de edición. Su tecla de acceso rápido es F4.
- "Net attributes" (), esta herramienta permite cambiar los atributos de un cable, el número de bits y el tipo de cable que se configuran en las casillas junto al botón de la herramienta en la barra de herramientas. Su tecla de acceso rápido es F5.
- "Add port" (), esta herramienta añade un puerto a el componente seleccionado cada vez que se pulsa.
- "Anchor/Unanchor" (), esta herramienta ancla o desancla un elemento en la ventana de edición gráfica para evitar que se mueva por error durante una edición.

- "Replicate" (), con esa herramienta es posible seleccionar un elemento del circuito y replicarlo varias veces en la dirección indicada por el desplazamiento del ratón en la ventana de edición gráfica.
- "Delete the selection" (), borra los elementos seleccionados.

También se pueden editar algunos elementos del circuito haciendo doble click con el botón izquierdo del ratón sobre ellos, tras lo cual aparecerá una ventana en la que se pueden cambiar algunos parámetros dependiendo del elemento seleccionado. Al hacer click con el botón derecho del ratón aparecerá un menú con una serie de opciones de edición y propiedades del elemento seleccionado.

2.5.7 Colocación de puertas y conexión de cables

Al hacer click sobre la ventana de edición gráfica aparecerá el símbolo "x" en ese punto como marca de la posición actual, que es utilizada para la colocación de nuevos componentes.

Para colocar una nueva puerta se puede hacer click con el botón izquierdo y a continuación ir a "Components" en la barra de herramientas y seleccionar el deseado o usar las teclas de acceso rápido, por ejemplo, pulsando la tecla "A" se insertará una puerta AND. También se puede hacer click con el botón derecho y en el menú desplegable ir a "Components" y seleccionar el deseado.

Para eliminar un componente bastará con seleccionarla y presionar la tecla "Suprimir" o hacer click en ella con el botón derecho y seleccionar "Delete" en el menú desplegable.

Para conectar los componentes o cables se debe seleccionar la herramienta "Move/Connect", después se debe hacer click con el botón izquierdo, manteniéndolo pulsado, sobre el cable que se desea conectar. El icono del ratón cambiará () y se debe arrastrar el cable hasta el punto donde se desea conectar y dejar de pulsar el ratón.

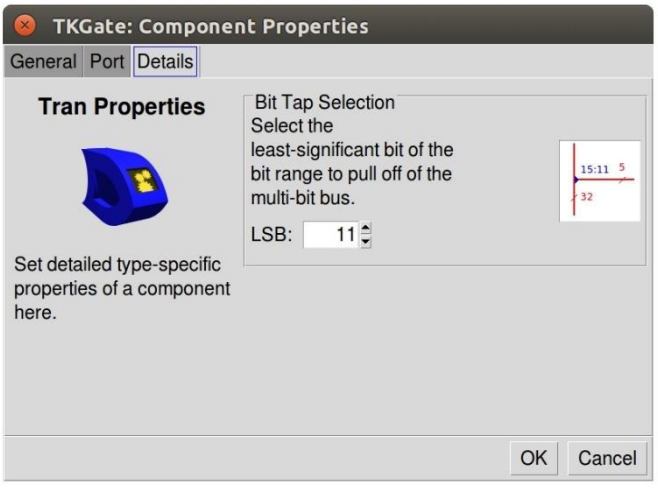

Figura 2.5.3 Ventana de propiedades de una conexión. En el recuadro de la parte derecha se puede ver el punto de conexión y los bits seleccionados.

Al conectar cables de diferente ancho de banda aparecerá un punto como el que se ve en el recuadro de la Figura 2.5.3. Al hacer doble click sobre este o hacer click con el botón

derecho y seleccionar “Instance properties” aparecerá una ventana en la que entre otras opciones se podrá seleccionar el bit menos significativo (LSB) del rango del bus. Esto sirve por ejemplo en el caso de que se conecte un cable de 32 bits con uno de 5 bits para indicar que 5 bits compartir. Se compartirán 5 bits a partir del menos significativo indicado.

2.5.8 Herramientas de módulos

Es posible agrupar esquemas de componentes en módulos que se suelen ver como un componente rectangular (Figura 2.5.4). Los módulos creados aparecen en las listas de componentes del circuito.

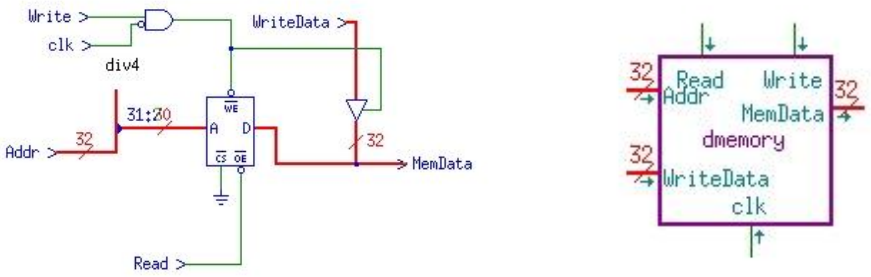

Figura 2.5.4 Esquema de componentes de una memoria (izq.) y su representación como módulo (dcha.).

Para editar un módulo se puede hacer doble click en el elemento de la lista “Tree”, usar los botones de abrir y cerrar módulo ( ) o las opciones disponibles en la pestaña “Module” de la barra de herramientas.

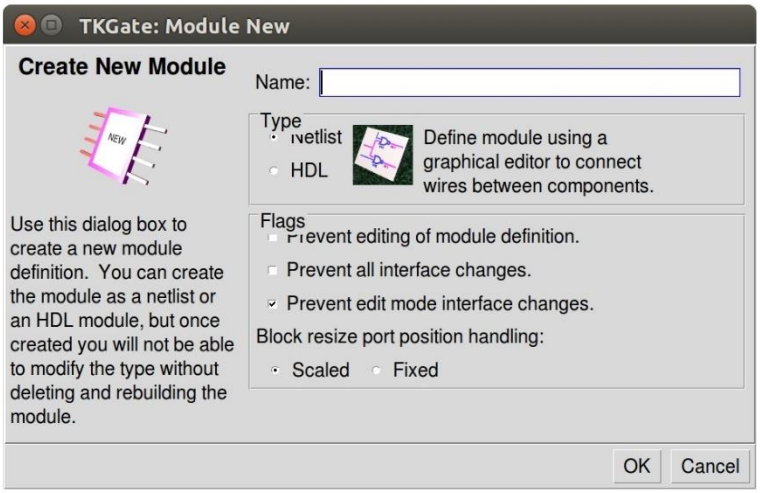

Figura 2.5.5 Ventana de creación de un módulo.

Se puede crear un nuevo módulo desde la pestaña “Module” o desde el botón ( ). Para crearlo se puede utilizar un esquema de componentes conectados entre sí o se puede definir mediante el lenguaje Verilog (Figura 2.5.5), lo que permite detallar el comportamiento de un módulo mediante este código de programación (Figura 2.5.6).

```
module ALU(A,B,CTL,Z,ZERO);
  input [31:0] A;
  input [31:0] B;
  input [2:0] CTL;
  output ZERO;
  output [31:0] Z;

  assign Z = alu_out(A,B,CTL);
  //assign ZERO = ~| Z;
  assign ZERO = ~ z_flag(Z);

  function [31:0] alu_out;
     input [31:0] a,b;
     input [2:0] ctl ;
     case ( ctl )
        3'b000: alu_out=a&b;
        3'b001: alu_out=a|b;
        3'b010: alu_out=a+b;
        3'b011: alu_out=0;
        3'b100: alu_out=0;
        3'b101: alu_out=0;
        3'b110: alu_out=a-b;
        3'b111: alu_out=a-b >> 30; // Sign of difference
        default : begin
           alu_out=32'hxxxx;
           //$display("Illegal CTL detected!!");
        end
     endcase
  endfunction
  function z_flag ;
     input [31:0] a ;
     begin
        z_flag =
        ^{a[0]|a[1]|a[2]|a[3]|a[4]|a[5]|a[6]|a[7]|a[8]|a[9]|a[10]|a[11]|a[12]|a[13]|a[14]|a[15]|a[16]|a[17]|a[18]|a[19]|a[20]|a[21]|a[22]|a[23]|a[24]|a[25]|a[26]|a[27]|a[28]|a[29]|a[30]|a[31]};
     end
  endfunction
endmodule
```

Figura 2.5.6 Código Verilog de ejemplo perteneciente a una ALU.

2.5.9 Simulación TkGate

Se puede acceder a los controles del simulador desde la pestaña “Simulate” en la barra de herramientas o desde los botones situados bajo ella. Para comenzar una simulación se debe pulsar sobre la opción “Begin Simulation” desde la pestaña “Simulate” o presionando el botón “play” en la barra de botones. En este modo no se puede modificar el conexionado del circuito, aunque si se puede interaccionar con ciertos componentes de entrada, como pulsadores o interruptores.

En TkGate el tiempo es medido en unidades denominadas “epochs”. Cada puerta tiene un retraso de una cierta cantidad de epochs. Algunas puertas complejas tienen varias constantes de retraso. Además, algunas puertas como registros y memorias tienen parámetros adicionales de retraso que afectan a cambios internos de estado.

Los comandos básicos del simulador son:

- “Run” (), inicia el modo de simulación en ejecución continua mientras que existan eventos en la cola de eventos. Si existen puertas de reloj en el circuito, esto significa que la simulación continuará de forma indefinida. Si el circuito es combinacional, la simulación continuará hasta que se alcance el estado estable.
- “Pause” (), produce una pausa en la simulación que está en proceso.
- “Step Epoch” (), hace avanzar la simulación un número fijo de epochs. El número de epochs que avanza puede ser establecido en las opciones del menú de simulación. Puede usarse la barra espaciadora para ejecutar este comando.
- “Step Cycle” (), hace avanzar la simulación al flanco de subida del reloj. Se puede establecer el número de ciclos de reloj a simular y el número de epochs que deben pasar para un ciclo. Al seleccionar esta opción el simulador no se para justo en el flanco de reloj a menos que se configure en las opciones de simulación. Puede usarse la tecla tabulación para ejecutar este comando.
- “End Simulation” (), hace que la simulación finalice y borra todas las sondas.

2.5.10 Opciones del simulador

Se puede acceder a las opciones del simulador desde la ruta Tool > Options > Simulate. Las opciones principales del simulador son:

- "Epoch Step Size", especifica el número de epochs para avanzar el simulador cada vez que se avance un paso empleando el botón "Step Epoch".
- "Clock Overstep", especifica el número de epochs para simular pasado el flanco de reloj cuando estamos haciendo paso del reloj.
- "Clock Cycle Step Size", especifica el número de ciclos de reloj que se avanza cada vez que se avance empleando el botón "Step Cycle".

También se puede acceder a más opciones del simulador en la ruta File > Circuit Properties. En esta sección, en las pestañas Scripts y Simulation, se encuentran las siguientes opciones destacadas:

- "Initialization Script", indica los scripts de simulación para ejecutar de forma automática cuando se comienza una simulación. Los ficheros scripts especificados aquí son una propiedad global y se aplican a cualquier circuito que se cargue en TkGate.
- "Clock step stops on all clock posedges", indica que el comando de paso de reloj debería ser disparado en los flancos positivos en todos los relojes del circuito.
- "Clock step on clock", indica que el comando de paso de reloj debe ser disparado en los flancos positivos sólo en un reloj especificado. Esta opción sólo es útil para circuitos con varios relojes.

2.5.11 Observar la salida

TkGate emplea la sintaxis de Verilog para mostrar y especificar valores. Esta sintaxis está formada por un prefijo para indicar el número de bits, un indicador de la base y los dígitos del valor observado. Los caracteres usados para indicar la base en TkGate son "b" para binario y "h" para hexadecimal. Por ejemplo, "4'hb" es el número hexadecimal de 4 bits b, en binario seria el número "4'b1011". El simulador soporta multitud de valores lógicos (0, 1, x (desconocido), z (flotante), L (bajo), H (alto), ...).

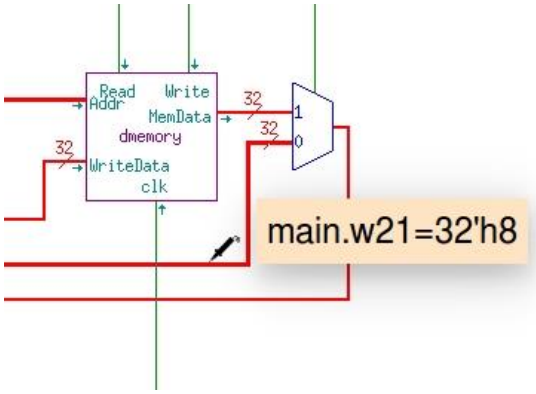

Figura 2.5.7 Manteniendo el botón izquierdo del ratón pulsado sobre un cable se muestra su valor durante la simulación.

Para mostrar el valor de una señal en un circuito, se debe hacer clic y mantener pulsado el botón del ratón sobre el cable. Esto mostrará el valor que circula por el cable mediante la sintaxis Verilog como se indicó previamente (Figura 2.5.7). Al dejar de pulsar el botón del ratón sobre el cable el valor desaparecerá. Esta característica puede ser empleada tanto cuando la simulación está pausada como cuando está en modo de simulación continua.

Si el simulador está en el modo de simulación continua, se muestra el valor del momento en el que el botón del ratón se pulsó por primera vez.

TkGate también permite establecer puntos de observación permanentes de una señal, para ello se debe hacer doble clic sobre el cable en el que se desea observar la señal (Figura 2.5.8). Al hacer doble clic en un cable que ya tenga establecido un punto de observación permanente se eliminará ese punto. Una vez establecido un punto de observación los valores de este serán mostrados de manera continua en una nueva ventana llamada "Scope" (Figura 2.5.9).

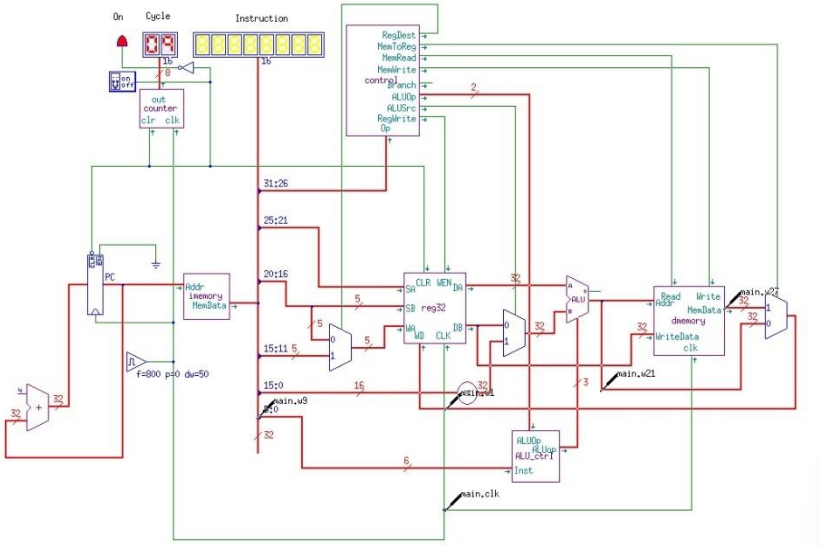

Figura 2.5.8 Circuito con puntos de observación permanentes establecidos.

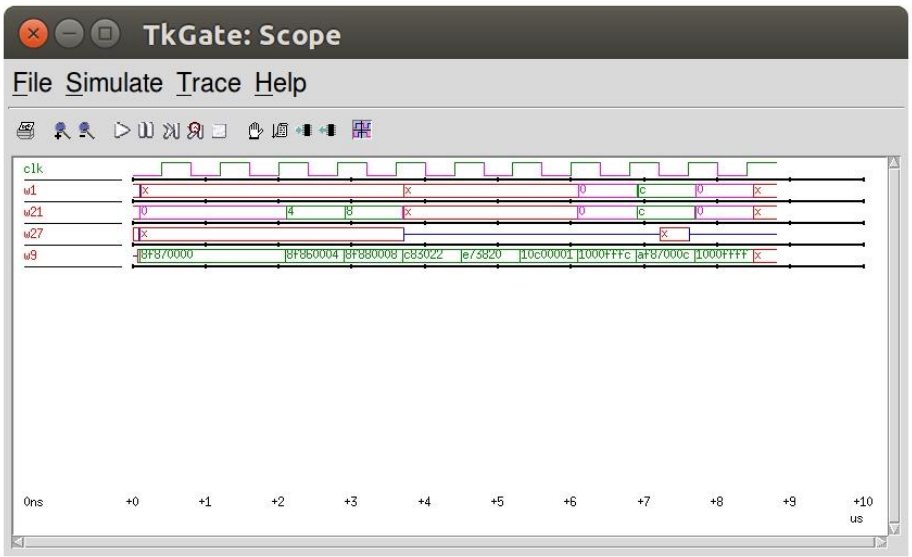

Figura 2.5.9 Ventana "Scope" donde se muestran los valores registrados una la simulación.

Las señales de varios bits mostrarán su valor en formato hexadecimal.

Para cambiar el rango de los valores mostrados y la escala de la ventana se puede emplear la barra de desplazamiento, el ratón o los comandos de teclado. Moviendo las barras de desplazamiento o haciendo clic en la ventana con el botón izquierdo del ratón en la ventana de trazas es posible desplazarse por a través del rango de la simulación. Para hacer zoom in se puede presionar el botón izquierdo del ratón mientras se mantiene la

tecla “ctrl” pulsada y para hacer zoom out se puede emplear el botón derecho del ratón o los botones de barra de herramientas. También es posible controlar la simulación desde la barra de herramientas para no tener que salir de la ventana “Scope”.

TkGate permite imprimir las trazas, para ello hay que ir a File > Print o al botón de la barra de herramientas ( ). Una vez en la ventana de impresión (Figura 2.5.10) en la pestaña “Output” se puede escoger una impresora externa o guardar la impresión en un archivo *.ps, también se permiten seleccionar las opciones de página (tamaño, orientación, doble cara). La pestaña “Content” permite seleccionar el rango temporal que se desea imprimir y la escala, junto a estas opciones muestra una estimación del número páginas que se emplearán para la impresión.

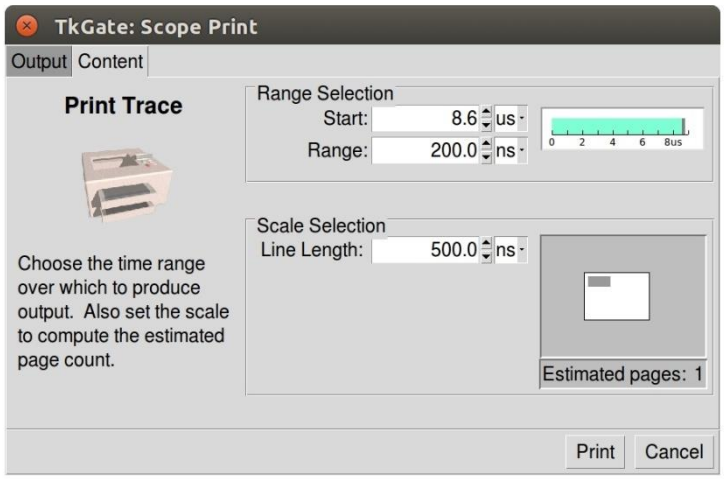

Figura 2.5.10 Ventana de impresión de trazas.

2.5.12 Estableciendo puntos de ruptura

Se permite establecer puntos de ruptura que detienen una simulación que se ejecuta en modo continuo al darse una condición establecida por el usuario. Los puntos de ruptura se pueden establecer desde la ruta Simulate > Breakpoint de la barra de herramientas o accediendo directamente desde la pestaña “Breakpoint” de la ventana inferior de la simulación (Figura 2.5.11). Una vez en esta sección se pueden añadir, editar, eliminar, activar o desactivar los puntos de ruptura con los botones correspondientes.

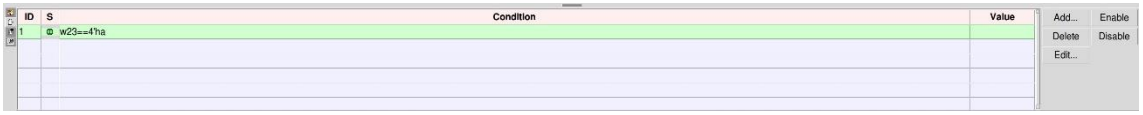

Figura 2.5.11 Captura de la interfaz para introducir breakpoints donde se puede ver el breakpoint w23=4’ha que detendrá la simulación cuando el valor de w23 sea igual a 10 (decimal).

Los operadores de condición son los mismo que se usan en Verilog, los valores deben de seguir la sintaxis de Verilog. También se puede indicar un nombre de señal para introducir un punto de ruptura cuando la señal no es cero o mediante el operador “!” delante del nombre de la señal para indicar que la ruptura debe producirse cuando la señal se hace cero, por ejemplo “w20!”.

2.5.13 Inicializando memorias

Un circuito puede contener una o más memorias. Para que tengan unos valores iniciales TkGate permite inicializar memorias desde un fichero y también permite guardar el contenido de una memoria a un fichero mediante las siguientes opciones:

- "Load memory", carga los datos de una memoria desde un fichero seleccionado. Si se asigna un fichero a una memoria este se cargará por defecto. Si el fichero de memoria contiene uno o más palabras "memory" las memorias especificadas serán cargadas con los contenidos del fichero. Los ficheros que se cargan se buscaran en la carpeta del circuito.
- "Dump memory", guarda el contenido de la memoria seleccionada en un fichero.

La extensión de los ficheros de memoria es *.mem. En los ficheros de memoria las líneas en blanco o empezando con "#" son ignoradas. Los valores de la memoria se escriben en hexadecimal y separados por un espacio. Los datos de la memoria se guardan seguidos desde la dirección inicial indicada de la memoria. Para indicarla se usa la sintaxis "@dir_hex" ó "dir_hex/" y en la siguiente línea se comienzan a escribir los valores. Un ejemplo de un fichero de memoria puede ser:

```
@memory imemory.m1

@0
8f870000 8f860004 8f880008 c83022 e73820 10c00001 0bffffc af87000c
200a0020 0140000a
```

Un ejemplo de cómo indicar que guarde los valores en direcciones indicadas puede ser:

```
@100
e124 f000 1ab0
@128
725a ab53 6789
@160
ed89 0000
```

La dirección al principio de la línea específica la dirección de memoria en la que comenzar a almacenar los valores.

Para seleccionar la memoria en la que se cargaran los valores se usa la palabra "memory" seguida de un argumento con el nombre de una memoria para cargar los valores en esta. Si no hay ningún argumento para "memory" los valores se cargaran en la puerta de memoria seleccionada. Por ejemplo, el fichero del ejemplo anterior cargará la memoria m1 en la memoria denominada "imemory".

En las memorias RAM, durante la simulación, los valores no se alteran mientras no se active la opción de escritura para no eliminar valores no deseados.

2.5.14 Informe de errores

Al iniciar una simulación aparece en la parte inferior una ventana (Figura 2.5.12) en la cual se muestran, en caso de producirse alguno, los errores y los módulos a los que estos afectan. Para comprobar el error se pueden observar los módulos correspondientes.

Figura 2.5.12 Ventana de informe de errores.

2.5.14 El lenguaje Verilog

Verilog es un lenguaje de descripción de hardware (**HDL**, del inglés ***H**ardware **D**escription **L**anguage*) creado por Phil Moorby en 1985 con el cual se puede diseñar, probar e implementar circuitos digitales, analógicos y mixtos con varios niveles de abstracción. En su comienzo Verilog fue un lenguaje propietario, pero en el año 1990 fue liberado para su estandarización que finalmente se llevó a cabo en el año 1995 y dio origen al *IEEE Verilog standard 1364*.

Su sintaxis es similar al lenguaje de programación C, lo que hace que sea fácil familiarizarse con él compartiendo la mayoría de palabras reservadas (if, while, …) aunque carece de algunos elementos como estructuras o funciones recursivas. A diferencia de lo que ocurre en C los bloques de sentencias no empiezan y terminan con una llave "{/}" si no que se usan las palabras "begin/end".

En este proyecto no se profundiza en este lenguaje, pero sí que es necesario tener algunos conocimientos mínimos sobre algunas sentencias que se explican a continuación:

- Case (n), compara el valor de n con las diferentes opciones dadas hasta que encuentra una coincidencia. Tiene la siguiente sintaxis:

```
case (n)
    2'b00: begin u = 3; end
    2'b0x, 2'b0z: begin u = 4; end
    2'b10, 2'b11, 2'bx1: begin u = 5; end
    2'bxx: begin u = 6; end
      ……
    default: begin u = 7; end
  endcase
```

- If (n), llevando a cabo una acción si se cumple una condición. Tiene la siguiente sintaxis:

```
if (n==2'b01) begin
   x = 2; end
 else begin
   x = 1; end
 endif
```

Capítulo III. Procesador monociclo

3.1 Construcción de un procesador monociclo básico

Se pretende que el alumno comprenda el funcionamiento de cada uno de los componentes de un procesador monociclo de modo que sea capaz de añadir nuevas funciones a este. Para ello un buen ejercicio es solicitar que detalle el proceso de creación de un procesador monociclo mediante TkGate: describir su ISA, detallar los componentes e identificar su funcionamiento mediante la correcta interpretación del camino de datos. El primer paso del alumno debería ser la definición del repertorio de instrucciones y las instrucciones que podrá ejecutar el procesador. En este caso se usará un repertorio de instrucciones basado en MIPS explicado en capítulo *2.4.2 El procesador MIPS* aplicado solo con unas instrucciones sencillas (Tabla 3.1.1).

Una vez definido el repertorio de instrucciones el alumno deberá detallar los componentes físicos necesarios para que se puedan llevar a cabo unas instrucciones básicas que se le propondrán, ADD, ADDI, SUB, BEQ, LW, SW.

Instrucción	Uso	Tipo	Opcode	Funct	Sintaxis	Operación
add	Add	R	0x00	0x20	$d, $s, $t	$d = $s + $t
addi	Add Immediate	I	0x08	NA	$d, $s, i	$d = $s + SE(i)
beq	Branch if Equal	I	0x04	NA	$s, $t, label	if ($s == $t) pc += i << 2
j	Jump to Address	J	0x02	NA	label	pc += i << 2
jr	Jump to Address in Register	R	0x00	0x08	labelR	pc = $s
lw	Load Word	I	0x23	NA	$t, i ($s)	$t = MEM [$s + i]:4
sw	Store Word	I	0x2B	NA	$t, i ($s)	MEM [$s + i]:4 = $t

Tabla 3.1.1 Tabla de instrucciones que se utilizarán.

A continuación, se detallan los módulos que se han seleccionado para la implementación del procesador, algunos de ellos ha sido necesario crearlos mediante su descripción en Verilog.

ALU

Se requiere una ALU (Figura 3.1.1) capaz de realizar las operaciones aritméticas (suma, resta, …) del procesador. La ALU también activa la señal Zero en caso de que el resultado sea cero. Esto es útil para realizar las instrucciones de tipo *branch*.

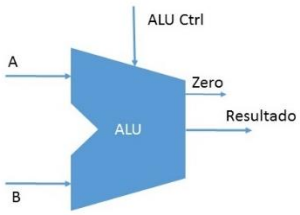

Figura 3.1.1 Representación de una ALU.

Contador de programa (**PC** del inglés ***P**rogram **C**ounter*)

Se requiere un registro para guardar la dirección de la instrucción que se está ejecutando (Figura 3.1.2). Cuando WE está activo se produce el cambio Instr = Nuevo PC.

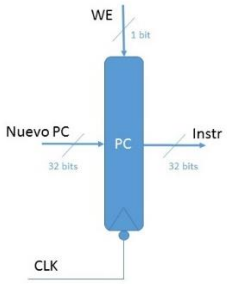

Figura 3.1.2 Representación del contador de programa.

Control de programa

Es un sumador que permite calcular PC+4, se podría usar la ALU, pero usando este recurso adicional se pueden realizar las operaciones de la ALU y el cálculo de la nueva dirección en paralelo. Se tiene en cuenta que, salvo en operaciones de salto, la dirección de la próxima instrucción en la memoria de instrucciones es siempre la siguiente de la actual.

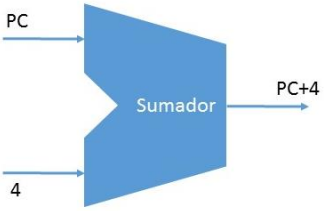

Figura 3.1.3 Representación de un control de programa.

Unidad extensora

Se requiere una unidad capaz de extender un operando o campo de instrucción de 16 bits a 32 bits. Se utilizará para realizar la extensión de bits necesaria para las instrucciones con valores inmediatos. La unidad añadirá ceros si es un valor positivo y unos si es uno negativo. Los números enteros se representan en complemento a dos.

Figura 3.1.4 Representación de una unidad extensora.

Multiplexor

Se requieren varios multiplexores para escoger señales en varios puntos del circuito: las entradas de la ALU, la entrada a registros, la salida del procesador y para el cálculo de una nueva dirección que serán controlados por la Unidad de Control.

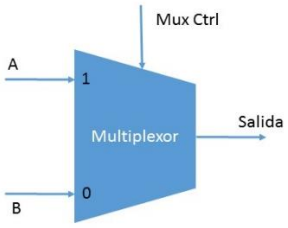

Figura 3.1.5 Representación de un multiplexor de 2 entradas.

Memorias

Se requiere una memoria de instrucciones donde estarán almacenadas previamente las instrucciones para ser ejecutadas y otra para leer y escribir los datos (Figura 3.1.6). Aunque esto se puede realizar con una sola memoria se escoge tener dos diferentes. La memoria de datos podrá escribirse cuando la Unidad de Control active la salida de escritura de memoria, en cambio la de instrucciones no se podrá modificar.

Como se ha explicado en el capítulo *2.2 Arquitectura de Von Neumann* el diseño del procesador de este proyecto se basará en una arquitectura Harvard, pero obviando la memoria principal que contiene instrucciones y datos, de forma que habrá dos memorias separadas, como si fuesen las memorias caché, una para datos y otra para instrucciones.

Figura 3.1.6 Esquema de conexiones de la memoria de instrucciones (izqu.) y de datos (drch.).

Banco de registros

Se requiere de un banco de registros para almacenar datos y usarlos como argumentos en próximas instrucciones.

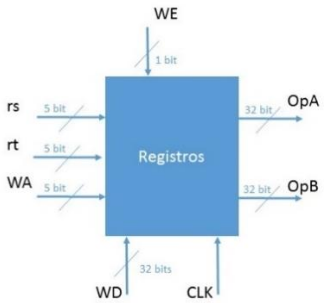

Figura 3.1.7 Esquema del banco de registros.

Unidad de Control

Se requiere una Unidad de Control que sea capaz de identificar las instrucciones e indicar a cada uno de los componentes el proceso que debe realizar.

Figura 3.1.8 Esquema de la Unidad de Control.

La Unidad de Control es un circuito combinacional descrito mediante lenguaje Verilog. Se le proporcionará parcialmente construido al alumno para que esté solo tenga que añadir ciertas funciones, ya que crearlo desde cero requiere unos conocimientos más amplios sobre Verilog, lo cual no es un objetivo contemplado en este proyecto.

Puertas AND

Se requiere una puerta AND para la implementación de las instrucciones de tipo *Branch* de forma que cuando se deba llevar a cabo un salto active el multiplexor correspondiente a la entrada del PC.

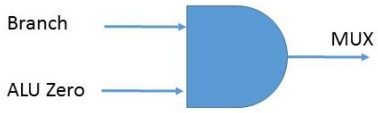

Figura 3.1.9 Puerta AND

Puerta de desplazamiento

Se requiere una puerta capaz de realizar un desplazamiento de 2 bits hacia la izquierda.

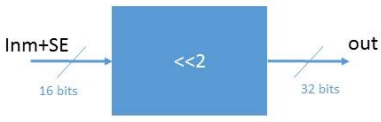

Figura 3.1.10 Unidad shift de desplazamiento.

Los elementos que requieren una sincronización poseen una conexión de reloj. Esta estructura, así como con unos *displays* para que se pueda observar el ciclo de reloj e instrucción en curso, se les proporcionará construida a los alumnos.

Una vez el alumno ha detallado los elementos de hardware necesarios debe realizar las conexiones entre ellos para que el procesador funcione correctamente:

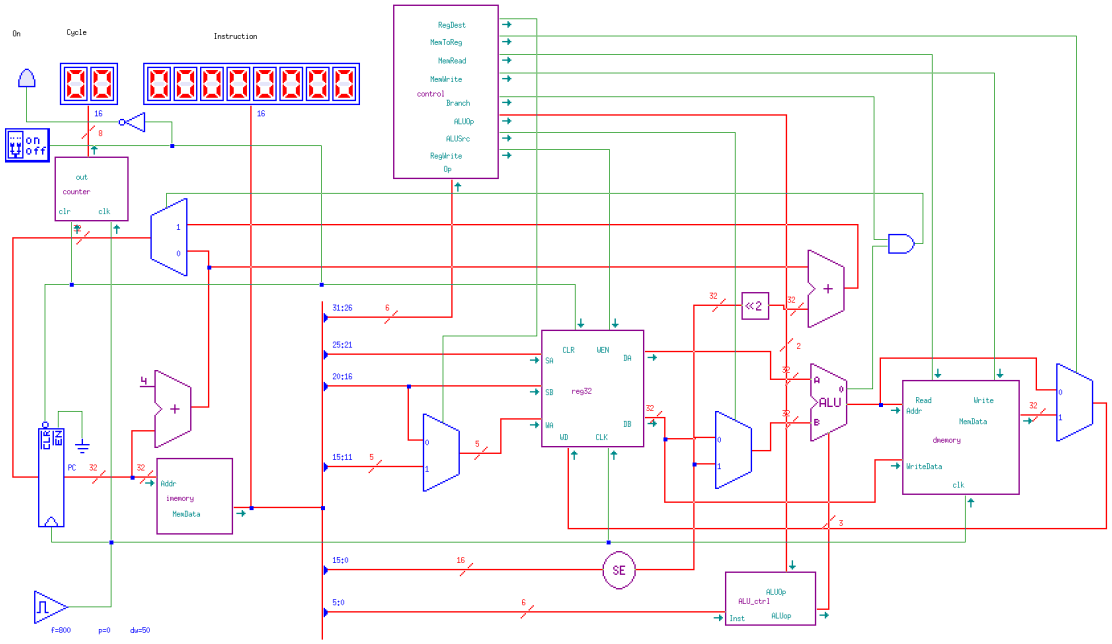

Figura 3.1.11 Esquema del circuito del procesador con sus elementos conectados.

Una vez finalizado el montaje una buena forma de comprobar el funcionamiento del procesador es estudiar la ejecución de las diferentes instrucciones implementadas. Por ejemplo, la instrucción lw rt, offset(rs), ya que con ella trabajan casi todos los elementos del circuito, como se detalla a continuación.

El PC envía la dirección de la instrucción a la memoria de instrucciones para que la busque. Seguidamente la instrucción se decodifica de forma que sus 6 bits más significativos son enviados a la Unidad de Control. Esta identifica la instrucción y envía, a través de sus salidas, ordenes de lo que hacer a cada elemento del circuito. En este caso para la instrucción lw debería activar a valor 1 las salidas MemToReg, RegWrite, ALUSrc y MemRead. Estas acciones provocan que:

- El banco de registros carga el valor apuntado por rs y lo envía a la ALU.
- ALUSrc activa el multiplexor anterior a la ALU, que deja pasar el valor del offset que previamente ha sido extendido a 32 bits con la unidad extensora.
- La ALU suma al valor de rs el offset.
- Se activa la lectura de datos en la memoria, que lee el dato de la dirección rs+offset.
- MemToReg activa el multiplexor a la salida de la memoria de datos para que deje paso al valor leído de ella.
- RegWrite activa la escritura de registros para poder guardar el valor recibido de la memoria en el registro rt seleccionado con el multiplexor previo al banco de registros.

A la vez que este proceso se realiza el sumador que se encuentra después del PC ha realizado la operación PC+4 y enviado esta dirección, que es la de la siguiente instrucción, al PC.

Cuando el alumno ha finalizado el montaje y comprendido el funcionamiento básico del procesador se le puede solicitar la implementación de nuevas instrucciones para lo cual deberá realizar varias modificaciones. Dos ejemplos de instrucciones que se puede solicitar implementar son las siguientes:

LWR rd, (rs+rt)

Esta instrucción carga en el registro rd el valor apuntado por el puntero resultante de la suma de dos registros (rs+rt). El primer paso para su implementación será codificar la instrucción (Tabla 3.1.2).

Opcode	rs	rt	rd	sa	function
000000	reg1	reg2	rdestino	00000	101000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Tipo R					

Tabla 3.1.2 Tabla de codificación de la instrucción lwr rd, (rs+rt).

A la vista de la instrucción se observa que será de tipo R. Por tanto, el principio de la instrucción no se tendrá que cambiar y habrá que buscar un código que no esté en uso para los bits de "function". La Unidad de Control necesitará recibir la información correspondiente a estos 6 últimos bits (*function*) para poder identificarla, por lo que se deberá añadir una nueva entrada capaz de recibir los nuevos datos. Para ello deberán usarse las opciones de edición de módulos que posee TkGate y definir nuevas entradas mediante lenguaje Verilog, pero dado que requiere más conocimientos sobre este lenguaje que no son explicados al alumno se le proporcionará la Unidad de Control con la entrada ya creada.

El siguiente paso es definir el comportamiento de la Unidad de Control para llevar a cabo la instrucción (Tabla 3.1.3).

Salida	Valor
RegDest	1
Branch	0
MemRead	1
MemToReg	1
ALUOp	00
MemWrite	0
ALUSrc	0
RegWrite	1

Tabla 3.1.3 Tabla de configuración de salidas de la Unidad de Control para la instrucción lwr rd, (rs+rt).

Para que la Unidad de Control pueda interpretar la nueva instrucción se debe realizar una modificación en su comportamiento mediante la edición de su código inicial en lenguaje Verilog, para ello habrá que colocarse sobre el componente y pulsar el botón de abrir módulo. La edición consistirá en añadir una sentencia condicional "if" al código original de manera que pueda interpretar los 6 bits correspondientes a la parte "function" de la

instrucción. Una vez editado el código quedará de la siguiente forma (en negrita el código añadido):

```
module control(RegWrite, MemToReg, ALUSrc, RegDest, ALUOp, Branch, Op, Func, MemRead, MemWrite,O1,O2);
  input [5:0] Op;
  input [5:0] Func;
  output reg RegWrite, MemToReg, ALUSrc, RegDest, Branch, MemRead, MemWrite, O1, O2;
  output reg [1:0] ALUOp;

  always @(Op) begin
    MemToReg=0;
    RegWrite=0;
    MemRead=0;
    MemWrite=0;
    Branch=0;
    RegDest=0;
    ALUSrc=0;
    ALUOp=2'b00;
    O1=0;
    O2=0;
    case(Op)
      6'b000000: begin // Rform
            if(Func==6’b101000) // lw rd, (rs+rt)
                begin
                RegDest=1;
                MemRead=1;
                MemToReg=1;
                RegWrite=1;
                end
          else
                begin
                RegDest=1;
                RegWrite=1;
                ALUOp=2'b10;
                end
      end
      6'b100011: begin // LW
        MemToReg=1;
        RegWrite=1;
        ALUSrc=1;
        MemRead=1;
      end
      6'b101011: begin // SW
        ALUSrc=1;
        MemWrite=1;
      end
      6'b000100: begin // BEQ
        Branch=1;
```

```
        ALUOp=2'b01;
      end
      6'b001000: begin // ADDI
        ALUSrc=1;
        RegWrite=1;
      end
    endcase
  end
endmodule
```

Para comprobar que la edición realizada funciona correctamente se le puede pedir al alumno que escriba y simule un programa que compare dos vectores y devuelva el valor 0 si son iguales y el valor 1 si son diferentes. Para ello deberá usar la nueva instrucción y algunas de las ya implementadas.

```
addi $8, $0, 1 // Carga el valor 1 y para restar cada iteración.
addi $9, $0, 4 // Carga el número de iteraciones de la serie.
addi $12, $0, 4 // Carga el puntero a los elementos del vector 1
addi $13, $0, 24 // Carga el puntero a los elementos del vector 2
lwr $10, ($12+$28) // Carga el valor n del vector 1
lwr $15, ($13+$28) // Carga el valor n del vector 2

beq $10, $15, 1 // Comprueba si los elementos del vector son iguales
beq $0, $0, 5 // Si no son iguales finaliza

addi $12, $12, 4 // Calcula la dirección de V1(n1+1)
addi $13, $13, 4 // Calcula la dirección de V2(n2+1)
sub $9, $9, $8 // Resta una iteración a la serie.

beq $9, $0, 5   // Comprueba si el valor de iteraciones de la serie ha llegado a su
beq $0, $0, -9  // fin, en caso de que no realiza una nueva iteración.

addi $7, $0, 1 // Carga el valor 1 porque son diferentes
sw $7, 0($28) // Devuelve y guarda el valor -1 porque no son iguales
beq $0, $0, -1  // Dado que no se dispone de instrucciones de fin de programa se usa este
                bucle infinito para que el procesador no avance evitando posibles
                conflictos.
addi $7, $0, 0 // Carga el valor 0 porque son iguales
sw $7, 0($28) // Devuelve y guarda el valor -1 porque son iguales
beq $0, $0, -1  // Dado que no se dispone de instrucciones de fin de programa se usa este
                bucle infinito para que el procesador no avance evitando posibles
                conflictos.
```

LWACC rd, offset(rt)

Esta instrucción coge el valor apuntado por la dirección de memoria resultante del puntero $rt+offset y se lo suma al valor actual del registro rd y lo guarda en ese mismo registro. rd=rd+*(offset+rt)

Al igual que en la anterior instrucción lo primero que se debe hacer es especificar su código de instrucción (Tabla 3.1.4)

opcode	rs	rt	offset
001011	rdestino	r1	Inm
6 bits	5 bits	5 bits	16 bits
Tipo I			

Tabla 3.1.4 Tabla de codificación de la instrucción lwacc rd, offset(rt).

Para el diseño de esta nueva instrucción se deberán introducir cambios en la Unidad de control y añadir nuevo hardware. A nivel de hardware se implementará un sumador que sea capaz de sumar el valor del dato apuntado por el puntero resultante de la suma del registro y el offset y el valor de rs. Una vez hecho esto se debe añadir un multiplexor que permita controlar la salida del circuito en caso de que no se use esta instrucción. Con los componentes de hardware ya añadidos el siguiente paso será realizar las modificaciones oportunas en la unidad de control (Figura 3.1.12).

En la Unidad de Control se deberá modificar su código añadiendo un nuevo "case" capaz de interpretar el "opcode" de la nueva instrucción. También se debe añadir una nueva salida de un bit que se llamará "Acc" que será la encargada de controlar el nuevo multiplexor colocado en el circuito. Una vez hecho esto se debe definir su comportamiento ante la nueva instrucción (Tabla 3.1.5).

Salida	Valor
RegDst	0
Branch	0
MemToRead	1
MemReg	1
ALUOp	00
MemWrite	0
ALUSrc	1
RegWrite	1
Acc	1

Tabla 3.1.5 Tabla de configuración de salidas de la Unidad de Control para la instrucción lwacc rd, offset(rt).

El nuevo código de la Unidad de Control será el siguiente (en negrita el código añadido):

```
module control(RegWrite, MemToReg, ALUSrc, RegDest, ALUOp, Branch, Op, Func,
MemRead, MemWrite,O1,O2,Acc);
  input [5:0] Op;
  input [5:0] Func;
  output reg RegWrite, MemToReg, ALUSrc, RegDest, Branch, MemRead, MemWrite,
O1, O2, Acc;
  output reg [1:0] ALUOp;

  always @(Op) begin
    MemToReg=0;
    RegWrite=0;
    MemRead=0;
    MemWrite=0;
    Branch=0;
```

```
    RegDest=0;
    ALUSrc=0;
    ALUOp=2'b00;
    O1=0;
    O2=0;
    Acc=0;
    case(Op)
      6'b000000: begin // Rform
            if(Func==6’b101000) //
                begin
                RegDest=1;
                MemRead=1;
                RegWrite=1;
                end
          else
                begin
                RegDest=1;
                RegWrite=1;
                ALUOp=2'b10;
                end
      end
      6'b100011: begin // LW
        MemToReg=1;
        RegWrite=1;
        ALUSrc=1;
        MemRead=1;
      end
      6'b101011: begin // SW
        ALUSrc=1;
        MemWrite=1;
      end
      6'b000100: begin // BEQ
        Branch=1;
        ALUOp=2'b01;
      end
      6'b001011: begin // LWACC
        MemRead=1;
        MemToReg=1;
        ALUSrc=1;
        RegWrite=1;
        Acc=1;
      end
    endcase
  end
endmodule
```

Para comprobar que la edición realizada funciona correctamente se le puede pedir al alumno que escriba y simule un programa que sume los elementos de un vector. Para ello deberá usar la nueva instrucción y algunas de las ya implementadas.

```
addi $8, $0, 1 // Carga el valor 1 y para restar cada iteración.
addi $9, $0, 4 // Carga el número de iteraciones de la serie.
addi $12, $28, 4 // Carga el puntero a los elementos del vector t

lwacc $13, 0($12) // r13=r13+*(r12+offset)

addi $12, $12, 4 // Calcula la dirección de V(t)
sub $9, $9, $8 // Resta una iteración a la serie.

beq $9, $0, 1   // Comprueba si el valor de iteraciones de la serie ha llegado a su
beq $0, $0, -5  // fin, en caso de que no realiza una nueva iteración.

sw $13, 0($28) // Guarda el valor resultante
beq $0, $0, -1 // Dado que no se dispone de instrucciones de fin de programa se usa este
                  bucle infinito para que el procesador no avance evitando posibles
                  conflictos.
```

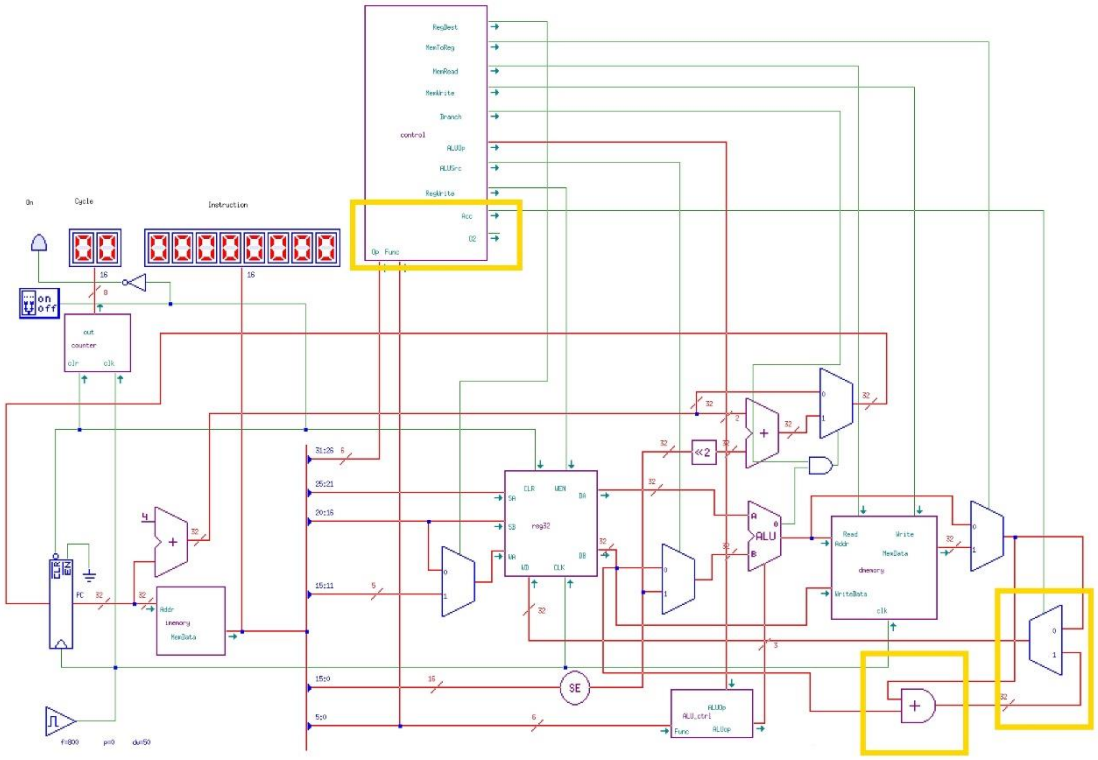

Figura 3.1.12 Esquema del circuito con la nueva configuración (en amarillo los cambios introducidos).

Capitulo IV. Procesador segmentado

Uno de los retos a los que se enfrenta un alumno en esta materia es la de comprender la segmentación y sus implicaciones en la ejecución de instrucciones.

4.1 Construcción de un procesador segmentado básico

Se pretende que el alumno comprenda las dificultades que presenta la implementación de un procesador segmentado y los problemas que conlleva. Como se ha indicado en el apartado *2.4.1 Implementación de procesadores* el procesador segmentado se aprovecha de las ventajas de dividir una instrucción en varias etapas para conseguir que diferentes instrucciones se ejecuten de forma solapada de manera simultánea. El procesador MIPS se suele dividir en 5 etapas, como se ha detallado en capítulo *2.4.2 El procesador MIPS*: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) y Write Back (WB).

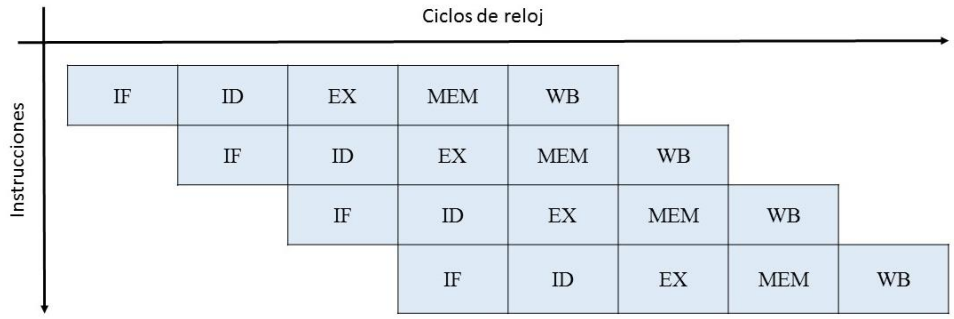

Figura 4.1.1 Esquema básico de la tubería de un procesador segmentada donde en cada ciclo de reloj comienza una nueva instrucción.

Aunque a primera vista la segmentación es un concepto sencillo el proceso de implementación puede resultar confuso. Es necesario controlar lo que ocurre en cada ciclo de reloj para que el mismo recurso no sea accedido por dos instrucciones al mismo tiempo. También se tiene que asegurar que unas instrucciones no interfieren con otras en las diferentes etapas de la segmentación, para ello se añaden los registros de segmentación entre etapas (Figura 2.4.4). De esta forma en cada ciclo de reloj todos los resultados intermedios son guardados en registros, que funciona como entrada de la siguiente etapa en el próximo ciclo de reloj.

Además de estas complicaciones también surge un problema cuando los datos de una instrucción en ejecución dependen de otra que también lo está. Esto se conoce como riesgo y puede reducir la rapidez del procesador segmentado. Existen tres clases:

- Riesgos estructurales, se produce cuando el hardware no puede soportar todas las combinaciones posibles de las instrucciones superpuestas. Es decir, varias instrucciones intentan acceder al mismo recurso a la vez.
- Riesgo de datos, se produce cuando una instrucción depende del resultado de una anterior pudiendo las dos instrucciones superponerse y no estar el dato preparado.
- Riesgo de control, se produce por las operaciones de tipo *branch* o que cambian el PC al intentar tomar una decisión sobre una condición que aún no se ha evaluado.

Los riesgos en la segmentación pueden hacer necesario detener temporalmente parte del procesador. Cuando se produce una detención las instrucciones previas pueden continuar mientras que todas las posteriores se detienen. A continuación, se pasa a describir cada uno de estos en más detalle.

Riesgos estructurales

En una máquina segmentada, la ejecución solapada de las instrucciones requiere la segmentación de unidades y duplicación de recursos para permitir todas las posibles combinaciones de instrucciones. El procesador no puede ejecutar una secuencia de instrucciones en las que todas utilicen esa unidad. También aparecen riesgos estructurales cuando algunos recursos no se han duplicado para permitir la ejecución de todas las combinaciones de instrucciones. Cuando se produzca este riesgo el procesador tendrá que parar una de las instrucciones hasta que el recurso necesario esté disponible.

Riesgos de datos

Los riesgos por dependencias de datos se producen cuando, debido a la segmentación, el orden de acceso a los operandos (lectura/escritura) cambia respecto al de un procesador normal. Si se considera la ejecución de las siguientes instrucciones:

ADD r3, r4, r5
SUB r6, r3, r7

Como se observa en la Figura 4.1.2, la instrucción ADD escribe el valor de r3 en la etapa WB, pero la instrucción SUB lee el valor de su operando de entrada r3 durante su etapa ID. Este problema se denomina riesgo por dependencias de datos. La instrucción SUB leerá y utilizará el valor erróneo.

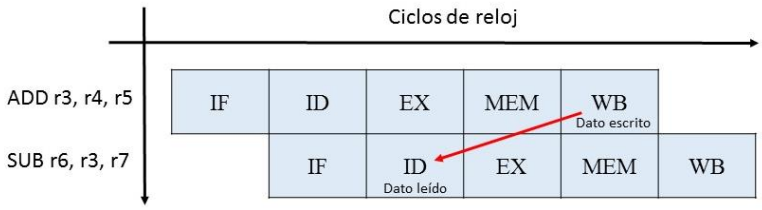

Figura 4.1.2 Riesgo de datos, la instrucción SUB intenta utilizar un dato que aún no ha sido guardado.

Una manera de resolver esto es parar la ejecución de la instrucción SUB hasta que el valor de r3 se haya escrito. Sin embargo, esto supone una pérdida de rendimiento. También se puede resolver con una técnica de hardware llamada adelantamiento (*forwarding*), con la que la perdida de rendimiento es menor. Esta técnica funciona de la siguiente forma:

- El resultado de la ALU de los registros EX/MEM y MEM/WB se realimenta a las entradas de la ALU.
- Cuando el hardware de adelantamiento detecta que la operación previa de la ALU ha escrito en un registro perteneciente a un operando de la operación actual de la ALU, la lógica de control selecciona el resultado adelantado como entrada de a ALU en lugar del valor leído en el fichero de registros.

En los casos en los que SUB sea detenido y se complete ADD este desvío no se activará y se usara el valor del registro r3. Hay que tener en cuenta que este riesgo también puede producirse en las demás instrucciones siguientes si utilizan el mismo operando (Figura 4.1.3).

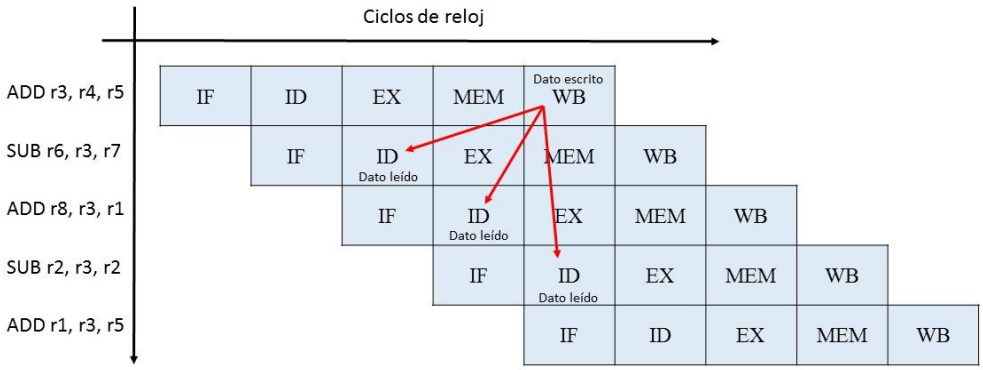

Figura 4.1.3 Riesgo de datos, las sucesivas instrucciones intentan utilizar un dato que aún no ha sido guardado.

Por cada instrucción que deba ser adelantada se requiere un hardware añadido, por lo que cuanto menor sea el número será mejor. La Figura 4.1.4 muestra el momento en el que el dato está disponible y cuando es requerido por instrucciones sucesivas.

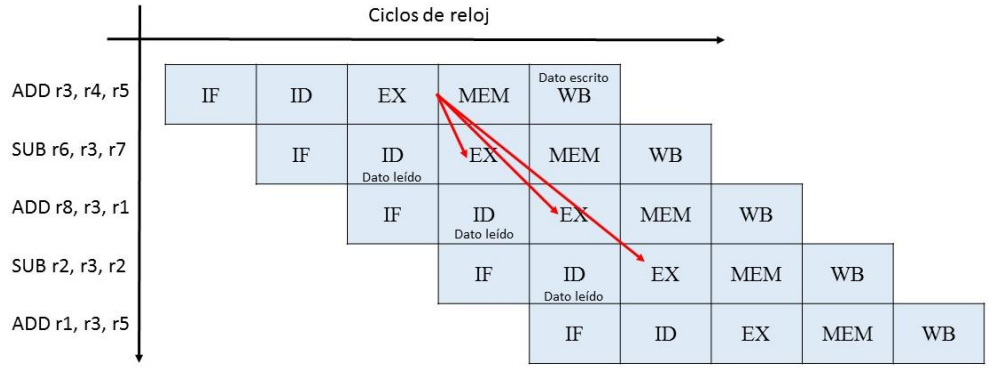

Figura 4.1.4 Riesgo de datos

En cada ciclo de reloj se accede a la escritura en el registro en la primera mitad de la etapa WB y a la lectura en la segunda mitad de la etapa ID. Con esto se consigue que para la tercera instrucción el dato ya se encuentre disponible (Figura 4.1.5) y no se requiere un adelantamiento.

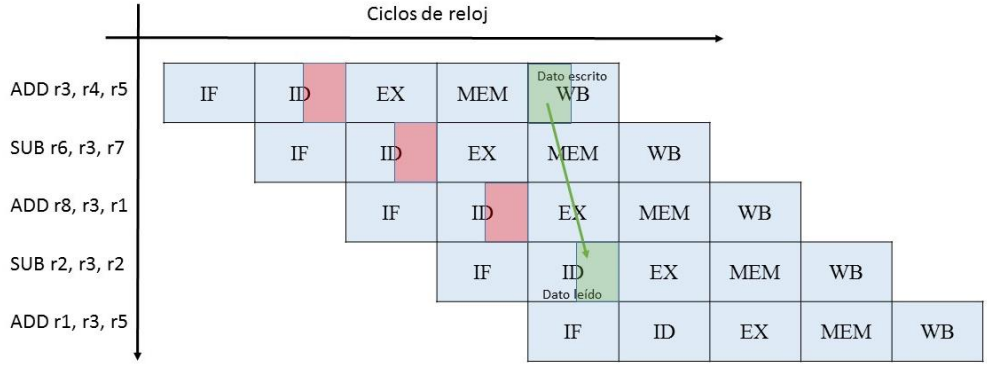

Figura 4.1.5 En la zona verde se puede leer el dato sin reisgo.

Para que el adelantamiento sea posible cada nivel requiere de un buffer que almacena el resultado de la ALU y un comparador para examinar si las instrucciones tienen un destino y fuente común. De esta forma cuando una operación necesita un dato adelantado se le suministra al entrar en su etapa EX. Para que se pueda escoger entre el dato adelantado o el valor del bus de datos se colocan unos multiplexores antes de las entradas de la ALU. Estas funciones se agrupan en la Unidad de Adelantamiento, que debe controlar las direcciones de los registros de origen y destino (Figura 4.1.6). Debido a que la ALU opera en una sola etapa de la segmentación, no hay necesidad de ninguna detención por ninguna combinación de instrucciones de la ALU, una vez que se ha implementado el desvío.

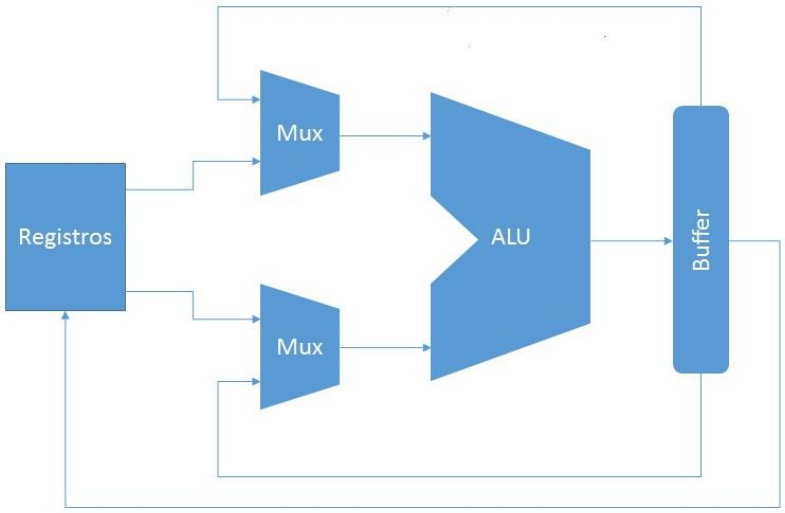

Figura 4.1.6 Ciricuito con adelantamiento.

Los riesgos de datos pueden clasificarse de tres formas, dependiendo del orden de acceso a escritura y lectura de las instrucciones. Considerando una instrucción *i* que va a ser procesada y una instrucción *j* que ya está siendo procesada. Los posibles riesgos por dependencia de datos son:

- **RAW** *(lectura después de escritura - Read After Write)*, la instrucción *i* trata de leer un registro fuente antes que sea escrito por la instrucción *j*, de forma que utiliza el valor antiguo, que es incorrecto. Este es el tipo más común de riesgos y es el que aparece en la Figura 4.1.3 y Figura 4.1.4.
- **WAR** (*escritura después de lectura - **W**rite **A**fter **R**ead*), la instrucción *i* escribe un registro de destino antes de que este sea leído por la instrucción *j*, de forma que utiliza un valor futuro, que es incorrecto (Figura 4.1.7). Esto no puede ocurrir en ejemplo de segmentación planteado porque todas las lecturas se hacen antes (en ID) y todas las escrituras después (en WB). Este riesgo se presenta cuando hay instrucciones que escriben anticipadamente los resultados en el curso de la instrucción, e instrucciones que leen una fuente después de que una instrucción posterior realice una escritura.
- **WAW** (*escritura después de escritura - **W**rite **A**fter **W**rite*), la instrucción *i* intenta escribir un registro antes de que sea escrito por la instrucción *j*, de forma que las escrituras se realizan en un orden incorrecto, quedando guardado en el registro de destino el valor antiguo de la instrucción *j* en lugar del escrito por *i* (Figura 4.1.7). Este tipo de riesgo solo se produce en procesadores segmentados que escriben en más de una etapa o permiten que una instrucción continúe aun cuando se encuentra

detenida la instrucción anterior. En el caso de este proyecto, la segmentación del DLX solamente escribe un registro en WB y evita esta clase de riesgos.

Conviene mencionar que en la arquitectura segmentada MIPS solo pueden suceder riesgos de datos tipo RAW.

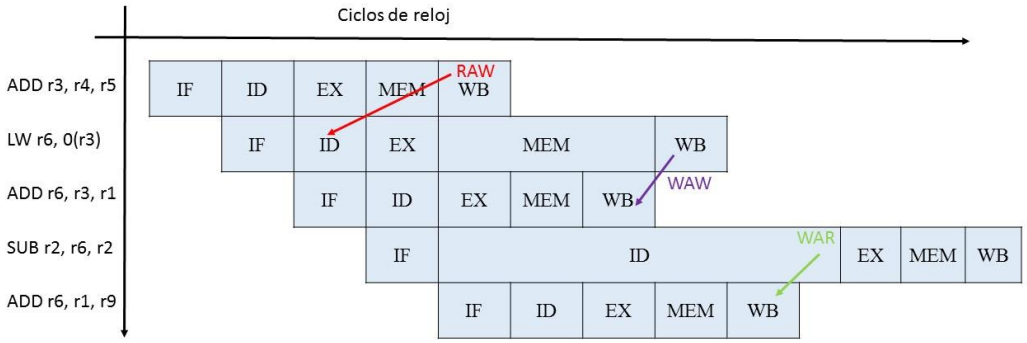

Figura 4.1.7 Representación de los tres tipos de riesgos de datos.

No todos los riesgos por dependencias de datos se pueden cambiar sin que afecten al rendimiento. Por ejemplo, sea la siguiente secuencia de instrucciones:

LW r3, 8(r1)
ADD r5, r3, r6
SUB r10, r3, r7
ADD r1, r3, r6

Este caso no es igual que los que se producen con las operaciones en la ALU. La instrucción LW no tiene el dato hasta el final del ciclo MEM, mientras que la instrucción ADD necesita el dato al comienzo de ese ciclo de reloj. Por tanto, el riesgo de utilizar el dato de una instrucción de carga no se puede eliminar completamente por hardware de adelantamiento. Para la instrucción SUB se puede adelantar el dato desde la memoria con lo que lo recibiría a tiempo. Para evitar que la instrucción ADD se ejecute con argumentos incorrectos es necesario parar el procesador y la pérdida de rendimiento es ineludible.

La solución más común a este problema consiste en añadir un hardware de detección de riesgos. Este hardware detecta estas situaciones y detiene la segmentación hasta que el riesgo desaparece. En este caso el hardware de detección detiene la segmentación cuando una instrucción quiere utilizar un dato que aún no está disponible hasta que la instrucción anterior lo tenga preparado. Esto proceso de detención se conoce como burbuja (*stall*), y permite que el dato llegue desde la memoria pueda ser adelantado por el hardware (Figura 4.1.8).

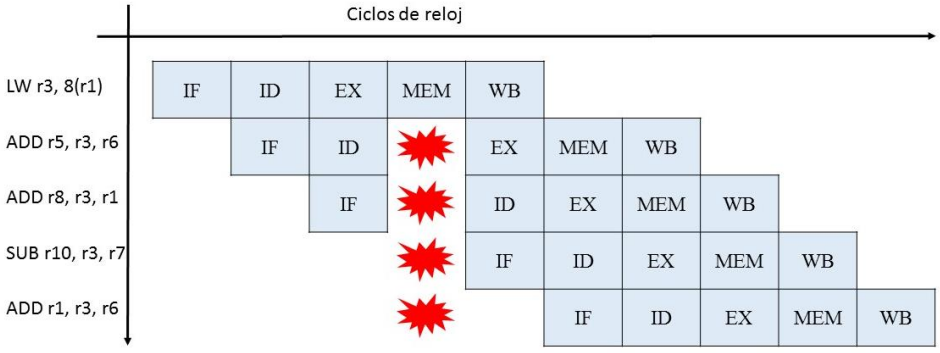

Figura 4.1.8 Efecto de una burbuja en la segmentación.

En el caso de que no se disponga de una Unidad de Detección de Riesgos el programador ha de introducir manualmente las paradas necesarias con la instrucción *nop*.

Riesgos de control

Pueden provocar una mayor pérdida de rendimiento para la segmentación que los riesgos por dependencias de datos. Cuando se ejecuta un salto, puede o no cambiar el PC a algo diferente su valor actual más 4. Recordar que, si un salto cambia el PC a su dirección destino, el salto es efectivo, en caso contrario es no efectivo.

Un salto se resuelve después de la etapa de ejecución, es decir se escribe en el PC la dirección del salto en la etapa de memoria, si es que se cumple la condición para realizar el salto. El salto puede realizarse o no. En caso realizarse el salto ya habrán ingresado en la segmentación las dos instrucciones siguientes al salto y comenzado a ejecutarse, si no se efectúa el salto, ya se tendrá adelantada la ejecución de las instrucciones que le siguen (Figura 4.1.9).

Se puede solucionar este riesgo de control deteniendo las instrucciones siguientes al salta hasta que la decisión se pueda efectuar. Esto producirá dos ciclos de reloj de detención (stall) por cada *branch*.

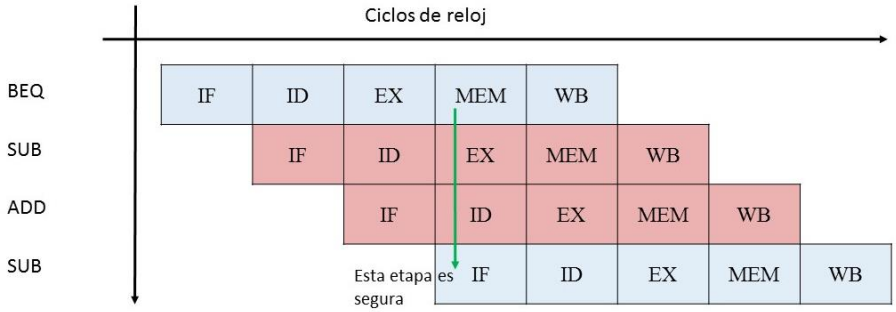

Figura 4.1.9 Riesgo de control con un salto.

Éstas paradas del procesador pueden ser realizadas por la unidad de control de riesgos insertando burbujas (Figura 4.1.10), pero también puede encargarse de esta tarea de detención el programador, que debe intercalar dos instrucciones *nop* después de cada BEQ:

```
beq $t2, $t3, $t4
nop
nop
sub $t8, $t9, $t10
```

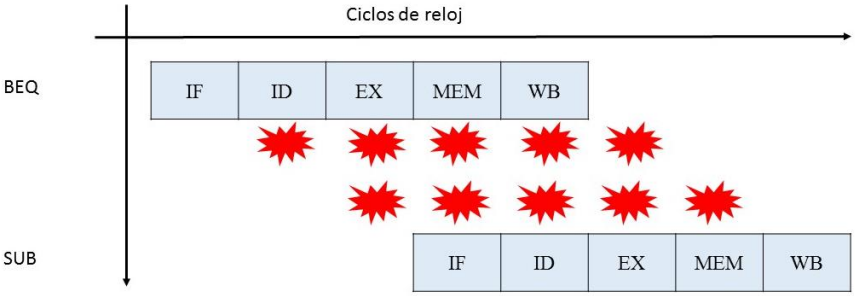

Figura 4.1.10 Secuencia de instrucciones con burbujas para evitar el riesgo en el salto.

Solución de riesgo de control anticipando la comparación

Se puede reducir la perdida de rendimiento anterior anticipando la comparación, para ellos se necesita agregar hardware que permita detectar la condición de igualdad en la etapa de decodificación ID y a la vez escribir el valor de PC. Esto permite reducir a un ciclo la parada del procesador.

Existen varias soluciones para este riesgo de control:

- Detención, en este método la solución por hardware detiene por un ciclo la etapa ID de la instrucción siguiente a un *branch*. La solución por software consiste en intercalar un nop, después de cada *branch*.
- Salto retardado (*delayed branch*), no se detiene la ejecución después de una bifurcación. La instrucción siguiente a una bifurcación comienza a ejecutarse siempre. Este método agrega un ciclo por cada *branch*, si el programador no puede encontrar una instrucción que sea útil, y que pueda realizarse después del *branch*.
- Vaciamiento (*flush*), de la etapa ID después del *branch*. Si el salto no se realiza, se continúa la ejecución. Si el salto se realiza, se debe descartar la instrucción leída y volver a efectuar la etapa ID.

Detalle sobre los saltos retardados retardadas (Delayed Branches)

En un procesador MIPS la instrucción después de un *branch* se ejecuta siempre, aun cuando se efectúe la bifurcación. Esta interpretación de los saltos permite al programador escribir código más eficiente.

Por ejemplo, en el siguiente código, la instrucción SUB después del BEQ se ejecuta aun cuando se efectúe el salto:

```
        ADDi r8, r9, r10
        BEQ r1, r3, salto
        SUB r2, r4, r6 // se ejecuta siempre
        …
salto:  ADD r1, r0, r3
```

Al finalizar la etapa de decodificación del *branch*, se sabe si el salto se realizará o no. Sin embargo, ya se habrá realizado la etapa ID de la instrucción siguiente sin que importe la resolución del salto. Muchas veces un programador puede encontrar una instrucción útil para colocar después de un *branch*. Si no puede encontrar una instrucción debe intercalar una instrucción *nop*.

Para ello podemos contemplar estas opciones:

- Para encontrar una instrucción que se pueda ejecutar después del *branch*, se mueve hacia el salto y se observa si la instrucción puede moverse en forma segura, en caso de que sea así se reemplaza el *nop*.

```
ADDi r8, r9, r10
BEQ r1, r3, salto
nop
```

```
        …
salto:  SUB r2, r4, r6
        ADD r1, r0, r3
```

- Otra opción es rellenar el *nop*, con una instrucción que se encuentre antes de la bifurcación. En el ejemplo, la instrucción *ADDI* que está antes de la bifurcación, no tiene riesgos y puede moverse, quedando:

```
        BEQ r1, r3, salto
        ADDi r8, r9, r10
        …
salto:  SUB r2, r4, r6
        ADD r1, r0, r3
```

Capítulo V. Casos de uso

Una vez el alumno comprende los conceptos detallados en los capítulos anteriores se le puede proponer una serie de ejercicios sobre ello para que los ponga en práctica.

5.1 Práctica procesador monociclo

Para introducir al alumno poco a poco en los conceptos básicos del simulador y el MIPS se puede comenzar con una práctica del procesador monociclo.

Los objetivos de esta práctica serán:

- Familiarizarse con el conjunto de instrucciones de MIPS y el simulador TkGate.
- Entender el funcionamiento de un procesador monociclo sencillo.
- Aprender a implementar una nueva instrucción en el procesador.

Para que puedan realizar la práctica se les debe proporcionar la guía de TkGate desarrollada en capítulo *2.5 El simulador TkGate* para que puedan aprender a controlar el simulador.

Ejercicio 1

El objetivo de este ejercicio es conseguir que el alumno se familiarice con el entorno del simulador. Para ello se le proporcionarán los archivos correspondientes a un procesador monociclo que sea capaz de ejecutar algunas instrucciones básicas: ADD, SUB, LW, SW y BEQ. En el esquema se debe incluir también la circuitería correspondiente al reloj, así como unos *displays* con los que puedan observar la instrucción actual del PC y el ciclo de reloj (Figura 5.1.1).

Junto con esto se incluirá un pequeño programa, que haga uso de las instrucciones del procesador, en un fichero de memoria para que pueda ejecutarlo.

Con estos elementos lo primero que deberá pedírsele al alumno es que abra el archivo del circuito e identifique los componentes de este, el camino de datos y de control. A continuación, se le debe pedir que se familiarice con los archivos de memoria, de instrucciones y de datos, que se le proporcionan. Por ejemplo, se le puede pedir que desensamble el programa de la memoria y comprenda su funcionamiento.

Ahora se deberá familiarizar con el entorno de simulación, para ello primero deberá simular el código y comprobar que se ejecuta correctamente.

Ejercicio 2

En el ejercicio anterior se puede introducir un fallo, ya sea a nivel de cableado o interno del módulo de control (en alguna de sus salidas cuando recibe una instrucción), de forma que la simulación no acabe correctamente. El alumno deberá entonces detectar y resolver la avería para que el programa se ejecute hasta el final.

Ejercicio 3

Cuando el alumno haya solucionado el fallo en el ejercicio anterior se le puede suministrar un código que realice la misma tarea pero que incorpore alguna nueva instrucción que deba implementar, por ejemplo, ADDI.

Se le proporcionará un nuevo fichero de memoria que deberá cargar en el circuito anterior, para ello se le debe indicar como cargar un fichero de memoria distinto en el circuito. Este proceso se realizará abriendo el fichero mips-single.vs y sustituyendo la línea de código $readmemh("mips-single.mem"); por $readmemh("mips-single-addi.mem");.

Cargada la nueva memoria el alumno deberá abrir el circuito e introducir los cambios oportunos en la unidad de control para añadir la nueva instrucción. Una vez realizados los cambios deberá simular el circuito y comprobar su correcto funcionamiento.

Ejercicio 4

El ejercicio anterior se resolvía solo con cambios en la Unidad de Control, en este ejercicio se le puede suministrar un nuevo fichero de memoria de instrucciones con un programa que realice la misma tarea que los anteriores, pero incluyendo las instrucciones de salto J y JR para hacer los saltos, cuya implementación requiere modificar también el camino de datos. Estas dos instrucciones estarán sustituidas por "32´hffffffff" en el código y se le pedirá al alumno que las sustituya por las correctas. De esta forma se pretende que se familiarice y entienda la codificación de estas dos instrucciones para lo que se le pide a continuación.

Proporcionándole un nuevo circuito (Figura 5.1.2) con modificaciones en la unidad de control para recibir los bits necesarios para poder implementar la instrucción y un nuevo módulo que al colocarlo permita realizar la decodificación de la instrucción J para que la puedan ejecutar.

El código en Verilog del nuevo módulo será:

```
module pc_shift_2(in,pc,out);
  input [25:0] in;
  input [31:0] pc;
  output [31:0] out;

  supply0 w4;
  wire [3:0] w0;
  assign w0 = pc[31:28];
  assign out = {w0, in, w4, w4};
endmodule
```

Una vez realizado el cambio por las instrucciones correctas, acabado el montaje y añadido del código necesario en Verilog a la unidad de control, el alumno deberá simular el circuito y comprobar su correcto funcionamiento.

Ejercicios adicionales

Se le puede proponer la creación de nuevos tipos de instrucciones, para lo cual deberá realizar un análisis del circuito y de los caminos de datos que necesitara modificar, y posteriormente llevarlo a cabo y simularlo. Por ejemplo, dos instrucciones que podría implementar son, LWR y LWACC, como se ha visto en el capítulo *3.1 Construcción de un procesador monociclo básico.*

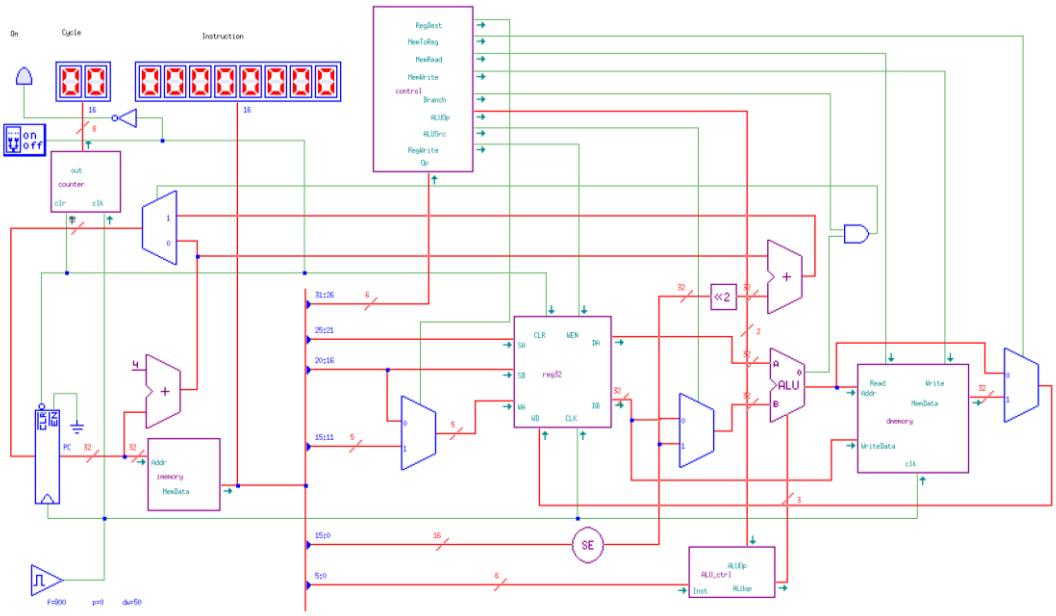

Figura 5.1.1 Circuito del procesador monociclo que se puede proporcionar al alumno.

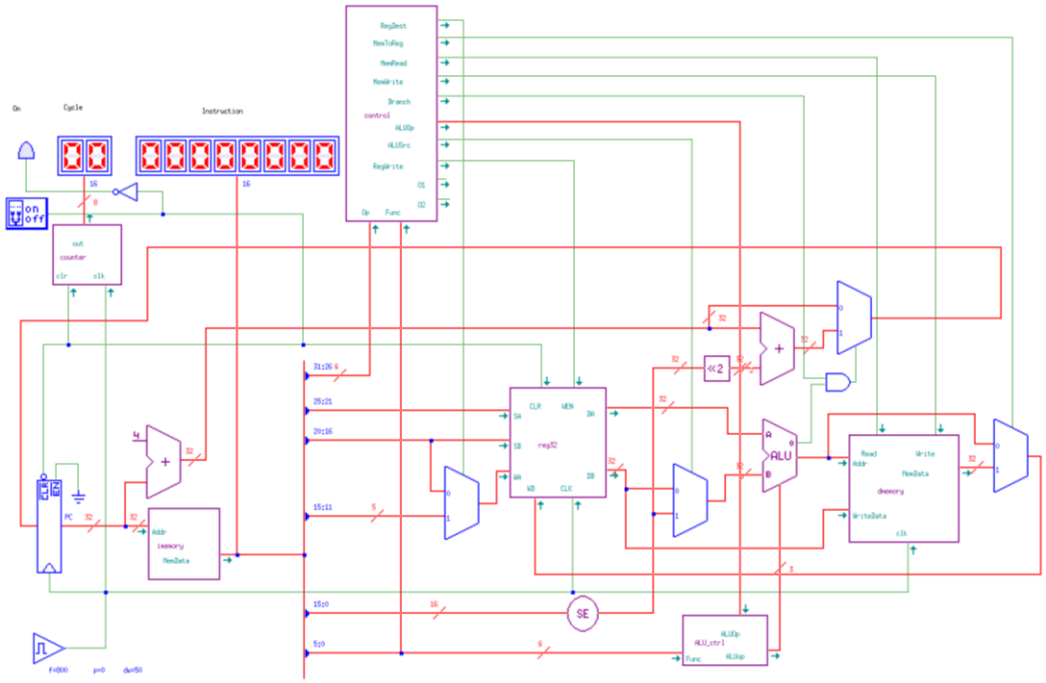

Figura 5.1.2 Circuito del procesador monociclo que se puede proporcionar al alumno para el ejercicio 4.

5.2 Procesador segmentado

Ahora que el alumno ya está familiarizado con el procesador monociclo se encuentra capacitado para acometer ejercicios que le permita entender mejor el funcionamiento de un procesador segmentado.

Los objetivos de esta práctica serán:

- Familiarizarse los riesgos estructurales y de datos.
- Entender el funcionamiento de un procesador segmentado básico.
- Ver diferentes implementaciones de las instrucciones de salto.
- Diseñar una unidad de anticipación sencilla.

Para el desarrollo de esta práctica se les requerirá un uso básico de *gmac*, un compilador de ensamblador sencillo para TkGate que no entiende etiquetas, ni macros y las instrucciones que reconoce han de ser especificadas en el propio fichero ensamblador. En cualquier caso, el uso que se le dará por parte del alumno es la simple introducción de su código de programa para que el compilador lo traduzca y se pueda usar en la memoria de instrucciones de TkGate.

Ejercicio 1

En este ejercicio se le proporcionará al alumno un archivo, mips-pipe.gm, en cuya parte final habrá un programa que deberá examinar y comprender. Tras esto se le pedirá que use *gmac* para compilarlo. Para ello solo debe acceder a través de la consola de Linux a la carpeta del archivo y ejecutar la siguiente sentencia:

gmac -o mips-pipe.mem mips-pipe.gm

Con esto ya tendrá listo un archivo de memoria que podrá simular en TkGate con el procesador monociclo construido anteriormente, comprobando que funciona correctamente. Nota: También se le proporcionará un fichero con la memoria de datos.

Ejercicio 2

Para este ejercicio se le proporcionará una implementación de un procesador MIPS segmentado en un fichero, mips-pipe-jump-exe.v (Figura 5.1.3). El circuito carecerá de unidades de anticipación o de detección de riesgos.

Una vez que haya estudiado el diseño se le pedirá que modifique el programa del ejercicio anterior, mips-pipe.gm, añadiendo instrucciones nop donde crea necesario para que el programa se ejecute correctamente en el procesador segmentado.

El alumno debe asegurase de que actualiza los *offsets* de las instrucciones de salto al realizar las modificaciones.

Cuando consiga que la simulación funcione correctamente deberá indicar cuantos ciclos tarda el programa en ejecutarse. Nota: dado que no hay instrucciones de fin de programa se considerará que el programa finaliza cuando la instrucción beq $0, $0, -1, colocada al final del mismo, termina de ejecutarse por completo por primera vez.

Ejercicio 3

Con unas pequeñas modificaciones es posible conseguir que las instrucciones de salto finalicen en su etapa ID. Con estas indicaciones se le solicitara al alumno que modifique el programa anterior para que se ejecute más rápido en el procesador del archivo, mips-pipe-jump-id.v (Figura 5.1.4). Al acabar de realizar las modificaciones y la simulación el alumno deberá indicar en cuantos ciclos de reloj se ejecuta el programa y ver la mejora de rendimiento.

Ejercicio 4

Se le pedirá al alumno que implemente un camino de datos con anticipaciones para evitar paradas entre instrucciones con dependencias de datos. Para ello se le proporcionara el fichero, mips-pipe-fw.v (Figura 5.1.5), en el que encontrará un procesador con una unidad de adelantamientos parcialmente implementada.

Una vez a estudiado el circuito deberá conectar los cables que crea necesarios e implementar las funciones lógicas del nuevo módulo. Cuando esté completado el montaje debe quitar todas las instrucciones *nop* posibles del código del programa, para reducir el tiempo de ejecución. Tras completar la nueva simulación se le pedirá que indique los ciclos que tarda en ejecutarse el programa ahora y el motivo por el que no se pueden quitar todos los *nop*.

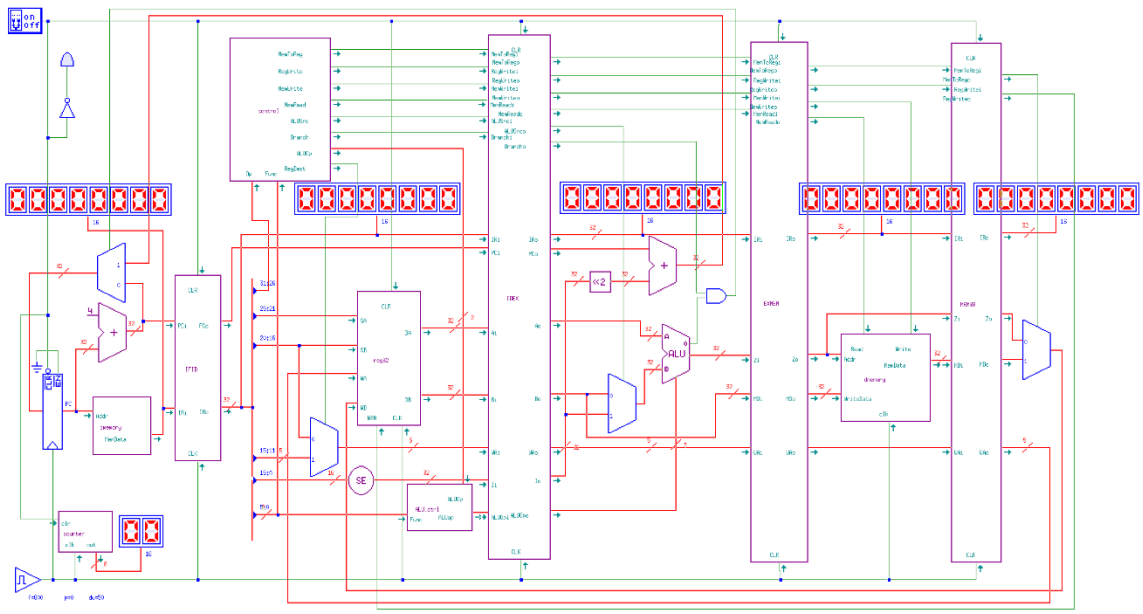

Figura 5.1.3 Circuito mips-pipe-jump-exe.v.

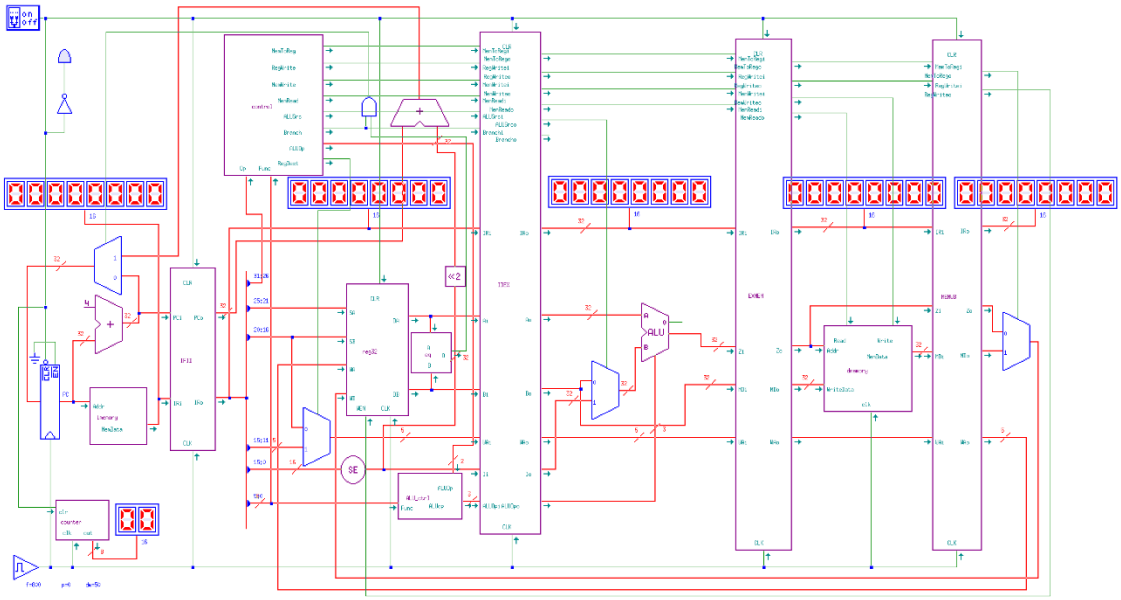

Figura 5.1.4 Circuito pipe-jump-id.v.

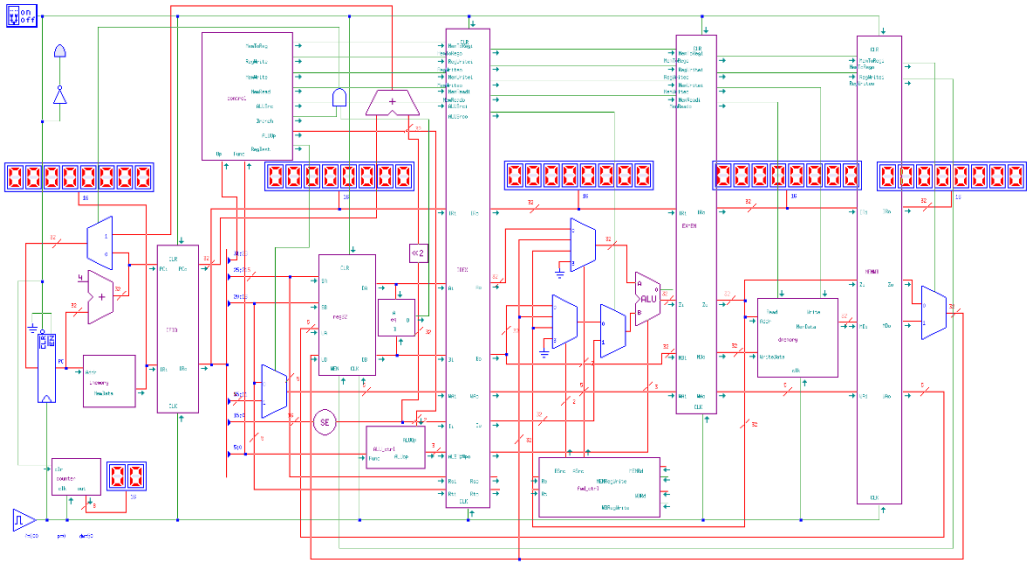

Figura 5.1.5 Circuito pipe-jump-id.v.

Capítulo VI. Conclusiones y líneas futuras

6.1 Conclusiones y líneas futuras

Las computadoras actuales son capaces de reducir su tamaño de forma impresionante, consiguiendo que un pequeño procesador este compuesto por millones de transistores. Esto dificulta analizar todos los componentes y las funciones que realiza cada uno.

Para estudiar su funcionamiento sólo se necesita buscar información en libros o sitios web donde se puede encontrar mucho material, pero para entender y comprender como se ejecuta cada instrucción con los cambios que realiza a lo largo del diseño es necesario ver la respuesta de cada componente durante el proceso de ejecución, algo que hoy en día es muy complicado en los procesadores actuales.

Por ello el uso de un simulador, como TkGate, en las aulas es de vital importancia para que los alumnos puedan llegar a entender el complejo proceso de ejecución de cada instrucción. Para ello, como se ha visto durante el proyecto, el simulador TkGate nos ofrece todas las herramientas necesarias para que los alumnos puedan observar con detalle cada componente de un procesador sencillo, pudiendo además realizar modificaciones para entender aún mejor su diseño.

Durante el proyecto se ha analizado el repertorio de instrucciones del procesador MIPS y se ha seleccionado un subconjunto adecuado para realizar programas sencillos. Dado este subconjunto se ha hecho una implementación de un procesador monociclo que permite ver el aula, de manera gráfica, el funcionamiento del mismo.

Además, se ha implementado un procesador segmentado que con la misma facilidad permite explorar las implicaciones de la segmentación en la ejecución de programas. Ambos procesadores pueden ser extendidos con nuevas instrucciones y unidades, dando lugar a interesantes ejercicios para el alumno.

Como líneas futuras para este proyecto se puede proponer la implementación de otro tipo de procesadores, como por ejemplo el ARM, la construcción de nuevos componentes en TkGate que permitan realizar nuevas funciones, por ejemplo, un módulo que permita observar los datos en el banco de registros o proponer la creación de una práctica donde el alumno trabaje con la detección de riesgos.

Bibliografía

Referencias y bibliografía

[1] «Computer Historry Museum,» [En línea]. Available: http://www.computerhistory.org/babbage/.

[2] J. L. H. &. D. A. Patterson, Computer Architecture A Quantitative Approach, 2011.

[3] J. P. Hansen, *Documentación de ayuda de TkGate.*

[4] «El modelo de Von Newmann,» [En línea]. Available: http://www.sites.upiicsa.ipn.mx/polilibros/portal/polilibros/p_terminados/PolilibroFC/Unidad_II/Unidad%20II_6.htm.

[5] *Apuntes de la asignatura Sistemas electrónicos digitales, Universidad de Cantabria.*

[6] P. L. S. Bijit, «Diseño de un Procesador. (Monociclo),» 2008. [En línea]. Available: http://www2.elo.utfsm.cl/~lsb/elo311/clases/c12.pdf. [Último acceso: 2016].

[7] J. J. Velasco, «John von Neumann, el genio detrás del ordenador moderno,» 2015. [En línea]. Available: http://www.eldiario.es/turing/John-Neumann-revolucionando-computacion-Manhattan_0_380412943.html. [Último acceso: 2016].

[8] «Verilog Doc Com,» [En línea]. Available: http://www.verilog.com/. [Último acceso: 2016].

[9] R. V. B. Cuéllar, «Revista digital investigación y educación, herramientas electrónicas para Linux. Circuitos digitales. TkGate 1.8.,» 2005. [En línea]. Available: http://www.csi-f.es/archivos_migracion_estructura/andalucia/modules/mod_sevilla/archivos/revistaense/n18/tkgate18.pdf. [Último acceso: 2016].

[10] P. L. S. Bijit, «Segmentación,» 2008. [En línea]. Available: http://www2.elo.utfsm.cl/~lsb/elo311/clases/c14.pdf. [Último acceso: 2016].

[11] J. I. V. Luna, R. S. G. G. S. G. y L. A. S. G. , «Arquitectura RISC vs CISC,» [En línea]. Available: http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-2.htm. [Último acceso: 2016].

[12] R. Camacho, «Arquitectura RISC y CISC,» 2012. [En línea]. Available: http://rcmcomputointegrado.blogspot.com.es/2012/03/arquitectura-risc-y-cisc.html. [Último acceso: 2016].

[13] E. Parra, «Instalar TkGate 2.0 en español en Ubuntu,» 2011. [En línea]. Available: http://www.eduardoparra.es/2011/11/instalar-tkgate-20-en-espanol-en-ubuntu.html. [Último acceso: 2016].

[14] L. A. G. Arizabaleta y H. L. Correa, «Energía y computación, Volumen III, No. 1,» 1994. [En línea]. Available:

http://bibliotecadigital.univalle.edu.co/bitstream/10893/1258/1/Microprocesadores%20con%20conjunto%20de%20instrucciones.pdf. [Último acceso: 2016].

[15] J. Weatherford, «MIPS and MIPS-like processors in TKGate,» [En línea]. Available: http://xidus.xidus.net/mips.html. [Último acceso: 2016].

[16] L. R. Córcoles y L. J. R.-A. , «Introducción a los microprocesadores MIPS, Universidad Rey Juan Carlos,» [En línea]. Available: https://www.uclm.es/profesorado/licesio/Docencia/ETC/21_MIPS-Introduccion-itis.pdf. [Último acceso: 2016].

[17] M. A. López, «El repertorio de instrucciones,» [En línea]. Available: http://www.des.udc.es/~margamor/descargas/EC1_tema2.pdf. [Último acceso: 2016].

[18] J. V. Neumann, «First Draft of a Report on the EDVAC,» 1945. [En línea]. Available: http://library.si.edu/digital-library/book/firstdraftofrepo00vonn. [Último acceso: 2016].

[19] J. F. G. MARTÍNEZ y J. V. F. VILLORA, «Procesadores Superescalares: Paralelismo Implícito a Nivel de Instrucción,» 2010. [En línea]. Available: http://ocw.uv.es/ingenieria-y-arquitectura/sistemas-electronicos-para-el-tratamiento-de-la-informacion/seti_materiales/seti7_ocw.pdf. [Último acceso: 2016].