UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un procesador MIPS R2000 con el simulador de circuitos lógicos CircuitVerse

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Dimitar Todorov Delchev

Tutor: Xavier Molero Prieto

Curso 2020-2021

Resum

El present treball pretén provar i demostrar la vàlua per a l'educació del nou simulador de circuits lògics CircuitVerse. Es tracta d'un simulador enfocat totalment a l'ensenyament, que destaca per la seua senzillesa i facilitat d'ús, encara que sense deixar de ser una eina molt potent que ofereix moltes possibilitats. Per a evidenciar les capacitats de CircuitVerse, en l'actual treball es pretén dissenyar, implementar i simular el processador MIPS R2000.

En aquest treball s'elabora una introducció al processador MIPS R2000, tot situant-lo en el seu context històric i introduint la seua història, així com una descripció detallada de la ruta de dades monocicle utilitzada com a base per a aquest projecte. Es descriu el simulador CircuitVerse, les seues principals característiques, els seus avantatges i la motivació per utilitzar-ho en l'actual treball. Una vegada introduït el MIPS R2000 i el simulador CircuitVerse, s'explica pas a pas com s'ha desenvolupat el processador utilitzant l'eina CircuitVerse.

Paraules clau: CircuitVerse, simulador lògic, processador, MIPS R2000, circuits digitals

Resumen

El presente trabajo pretende probar y demostrar la valía para la educación del nuevo simulador de circuitos lógicos CircuitVerse. Un simulador enfocado totalmente en la enseñanza, que destaca por su sencillez y facilidad de uso, aunque sin dejar de ser una herramienta muy potente que ofrece muchas posibilidades. Para evidenciar las capacidades de CircuitVerse, en el actual trabajo se pretende diseñar, implementar y simular el procesador MIPS R2000.

En el presente trabajo se elabora una introducción al procesador MIPS R2000, ubicándolo en su contexto histórico, introduciendo su historia, así como una descripción detallada de la ruta de datos monociclo utilizada como base para este proyecto. Se describe el simulador CircuitVerse, sus principales características, sus ventajas y la motivación para utilizarlo en el actual trabajo. Una vez introducido el MIPS R2000 y el simulador CircuitVerse, se explica paso a paso cómo se ha desarrollado el procesador utilizando la herramienta CircuitVerse.

Palabras clave: CircuitVerse, simulador lógico, procesador, MIPS R2000, circuitos digitales

Abstract

The current work pretends to test and prove the value for the education of the new logic circuits simulator CircuitVerse. A simulator totally focused on teaching, which main highlights are it's simplicity and ease of use, although while still being a very powerful tool that offers many possibilities. To demonstrate the capabilities of CircuitVerse, the current work intends to design, implement and simulate the MIPS R2000 processor.

In this paper an introduction to the MIPS R2000 processor is elaborated, placing it in its historical context, introducing its history, as well as a detailed description of the unicycle data path used as the basis for this project. The CircuitVerse simulator, its main characteristics, its advantages and the motivation to use it in the current job are described. Once the MIPS R2000 and the CircuitVerse simulator have been introduced, it is explained step by step how the processor has been developed using the CircuitVerse tool.

Key words: CircuitVerse, logic simulator, processor, MIPS R2000, digital circuits

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En el primer capítulo de esta memoria se presentará brevemente el trabajo, la motivación para llevarlo a cabo, los objetivos de este y la estructura de la misma. Tras ello se explicará brevemente la bibliografía consultada durante el desarrollo de este trabajo. Adicionalmente se hablará sucintamente sobre los Objetivos de Desarrollo Sostenible de la ONU en los que incide este trabajo y por último se ha introducido una breve sección de agradecimientos.

1.1 Motivación

Cada año, cada seis meses o incluso cada mes, se lanzan nuevos procesadores al mercado, modificando algún aspecto de los diseños anteriores para obtener una mejora sobre la versión anterior. Aunque esta observación sea cierta, esta mejora en la productividad de los procesadores suele ser pequeña, alrededor de unos pocos por cien respecto a la generación anterior. Pero cada cierto tiempo se presenta al mundo un diseño de procesador revolucionario que modifica las bases antes establecidas y trae consigo cambios importantes para toda la industria.

Este es el caso del procesador MIPS R2000, que trajo consigo un novedoso diseño y un revolucionario punto de vista que sigue siendo actual a día de hoy casi treinta años después y cada vez está tomando más fuerza.

En cambio en la enseñanza, desde los comienzos de la arquitectura de computadores, se han buscado diseños de procesador adecuados para introducir a los nuevos alumnos a este campo de la ingeniería. Diseños que al mismo tiempo deben cumplir características contrarias en su naturaleza, ser más simple para ser fácilmente comprensible su funcionamiento por alumnos noveles y al mismo tiempo ser suficientemente complejos para exponer las características de los procesadores actuales. Son claramente unos objetivos difíciles de cumplir por un mismo diseño.

A lo largo del tiempo se han utilizado diseños ficticios, otros reales pero ninguno satisfacía completamente los requisitos perseguidos por los docentes. En este momento se puso el foco en el procesador antes mencionado, el MIPS R2000, que en sí mismo es la obra de un catedrático de la Universidad de Stanford, John

L. Hennessy[1]. El MIPS R2000 con sus distintas versiones, empezando por la versión monociclo, la más simple y la utilizada en este proyecto, junto con la versión segmentada y sus derivadas, fueron vistas por los catedráticos como diseños muy adecuados para su uso en la enseñanza.

Además de un diseño de procesador adecuado para utilizarlo como base en las explicaciones, en la enseñanza existe una búsqueda constante de nuevas herramientas, simuladores, que se puedan utilizar para enriquecer el plan de formación de los alumnos. Mediante el uso de estas herramientas que pretenden simular el funcionamiento interno del MIPS R2000 y exponerlo a los alumnos, se persigue hacer la asimilación de los nuevos contenidos más asequible para los alumnos y por consiguiente obtener en el futuro unos profesionales mejor formados.

Con este punto de vista se crea CircuitVerse, un simulador de circuitos digitales dirigido totalmente a la enseñanza, que tiene como objetivo principal hacer más asequible el diseño, implementación y posterior simulación de circuitos digitales.

En el actual trabajo de fin de grado se pretende diseñar e implementar el MIPS R2000 en su versión monociclo en el novedoso simulador CircuitVerse. Con el fin de comprobar y demostrar su idoneidad para su uso en la enseñanza.

1.2 Objetivos

El actual trabajo persigue cumplir los siguientes objetivos:

1. Exponer las principales características del simulador CircuitVerse, junto con sus ventajas y desventajas respecto a otros simuladores disponibles.

2. Familiarizar al lector con el procesador MIPS R2000. Exhibir sus características y demostrar su importancia. Adicionalmente mostrar las diferencias entre las arquitecturas RISC y CISC.

3. Diseñar, implementar y simular el MIPS R2000 en su versión monociclo en CircuitVerse.

4. Demostrar que CircuitVerse es un simulador apropiado y útil para su uso en la enseñanza.

1.3 Estructura de la memoria

El actual trabajo está formado por X capítulos que permiten cubrir los objetivos del trabajo. A continuación se señala concisamente el contenido de cada capítulo que forma parte del trabajo:

[1] Entrada de la Wikipedia que trata sobre John L. Hennesy: https://es.wikipedia.org/wiki/John_L._Hennessy.

1. **Capítulo 1:** Se expone la motivación para llevar este trabajo a cabo, cuáles son los objetivos que se espera que este cumpla, qué estructura seguirá la memoria y por último se manifestarán algunos comentarios sobre la bibliografía consultada para su realización.

2. **Capítulo 2:** Este capítulo versa sobre el simulador CircuitVerse, el empleado en el trabajo; trata de hacer una introducción a CircuitVerse, mostrando sus principales características y capacidades. Por último, se exponen otros posibles simuladores disponibles, sus ventajas y desventajas respecto a CircuitVerse.

3. **Capítulo 3:** Trata sobre el procesador MIPS R2000, se expone brevemente su historia, sus características y sus diferentes versiones y se destaca su importancia en la educación.

4. **Capítulo 4:** Este capítulo expone el diseño, la implementación y simulación de la ruta de datos monociclo del procesador MIPS R2000 en el simulador CircuitVerse. En primer lugar se explica la implementación interna de los componentes que conforman la ruta de datos y posteriormente se describe el diseño e implementación de la propia ruta de datos.

1.4 Notas sobre la bibliografía

Para la realización de este trabajo se han consultado varios libros tanto en lengua castellana como en inglés. Estas obras están citadas en la bibliografía final junto con otras, algunos artículos y páginas web relevantes para el trabajo.

Todo la información relacionada con el simulador CircuitVerse se ha obtenido gracias a experiencia propia y al manual de este [4]. Además para exponer otros simuladores disponibles se han consultado sus paginas web y en algún caso artículos que tratan sobre el simulador en cuestión [5].

En lo respectivo al MIPS R2000 y su diseño interno se han consultado dos libros de los cuales son co-autores el creador de este procesador, John L. Hennessy, y el diseñador principal en la Universidad de Berkeley de la arquitectura RISC I, David A. Patterson[2]. Juntos, estos autores escribieron los dos libros más usados en la enseñanza de arquitectura de computadores: [2, 8]. Además para realizar las comparaciones entre las arquitecturas CISC y RISC se ha consultado el libro *Computer Organization and Architecture: Designing for Performance* escrita por William Stallings [9]. Toda la información proporcionada por los profesores de la asignatura de Estructura de Computadores, mientras se cursaba esta y también durante la realización del trabajo fue vital para la realización de este.

Junto con las obras citadas anteriormente se han consultado puntualmente otras relacionadas con el sector y la temática, principalmente *Computer Organization* de V. Carl Hamacher, Safwat G. Zaky, Zvonko G. Vranesic[1], Organización de computadoras: un enfoque estructurado de Andrew S. Tanenbaum [10], Ar-

[2]Entrada de la Wikipedia que trata sobre David A. Patterson: https://es.wikipedia.org/wiki/David_A._Patterson.

quitectura de Computadores de J. Ortega, M. Anguita y A. Prieto [6] y Arquitectura de Computadores de B. Parhami [7].

Para consultas relacionadas con circuitería electrónica y digital se ha consultado mayormente Circuitos electrónicos: Análisis, simulación y diseño escrita por Norbert R. Malik [3], libro enfocado totalmente a este campo.

1.5 Objetivos de Desarrollo Sostenible

Los Objetivos de Desarrollo Sostenible son un conjunto de objetivos, 17 concretamente, planteados por la ONU y plasmados en su Agenda 2030 para el Desarrollo Sostenible. Estos objetivos apuntan a los mayores desafíos a los cuales se enfrenta el mundo en la actualidad. Hacen referencia a problemas de los ámbitos social, económico y ambiental. En la Figura 1.1 se pueden contemplar los objetivos expuestos por la ONU.

Este trabajo intenta contribuir a todos ellos, pero donde mayor influencia puede tener es en los expuestos a continuación:

1. **Objetivo 4: Educación de calidad**. Este trabajo desde su concepción fue planteado con un objetivo claro: buscar y probar nuevas herramientas para la enseñanza con el único propósito de mejorar y facilitar la educación al mayor número de alumnos posibles.

2. **Objetivo 5: Igualdad de género**. Como tal el trabajo actual intenta facilitar la enseñanza de todos los alumnos, sin importar su género, sexo o su ideología. Además la herramienta principal del trabajo, CircuitVerse, está desarrollada por un equipo formado por personas de distinto género, diferente nacionalidad y de diferentes ideologías.

3. **Objetivo 9: Industria, Innovación e Infraestructura**. El nivel de educación tiene un impacto claro en la industria y en todos los ámbitos en general. Una mejora en la educación significa directamente una mejora en la industria, en la innovación y en la infraestructura.

4. **Objetivo 10: Reducción de las desigualdades**. En el trabajo actual se plantea el uso de una herramienta gratuita, de código libre y accesible mediante un navegador web. Todo ello facilita su uso sin tener un equipo informático muy exigente, pudiendo usarse incluso a través del teléfono móvil. Estas características son vitales, permitiendo que gente con menos recursos tenga acceso a la herramienta sin dificultades.

1.6 Agradecimientos

Antes de terminar el capítulo tengo que dar las gracias a varias personas que han hecho posible este trabajo y este viaje emocionante que fue estudiar el Grado de Ingeniería Informática en esta maravillosa universidad.

Figura 1.1: Los 17 Objetivos de Desarrollo Sostenible planteados por la Organización de las Naciones Unidas. Observando estos objetivos se puede percibir que cada uno es independiente de los otros, pero al mismo tiempo todos los objetivos están fuertemente conectados unos con otros.

En primer lugar tengo que agradecerles a mis padres por brindarme la oportunidad de seguir estudiando, ofrecerme los medios para ello, su incesante apoyo cuando creía que todo estaba perdido y por último su inagotable paciencia conmigo durante todos estos años.

En segundo lugar tengo que mencionar cómo empezó mi interés por la tecnología. Gracias a mis abuelos, ambos técnicos electricistas, desde edades muy tempranas me fueron introduciendo en cuestiones de electricidad, electrónica y de la tecnología en general. Con cada concepto nuevo que explicaban aumentaba mi curiosidad y mis ganas de aprender. Aun siendo ya mayores, tengo que agradecer a ambos el apoyo que me siguen ofreciendo y su inagotable fe en mí.

Por último, pero no menos importante, debo agradecer a mi tutor, Xavier Molero Prieto, sus consejos, su paciencia, sus ganas de enseñar y ayudar a sus alumnos y sus ideas cuando estaba estancado. Darle las gracias por su total disponibilidad, su rapidez en las correcciones y sus consejos para mejorar. Sin su ayuda este proyecto no habría sido posible.

CAPÍTULO 2
El simulador CircuitVerse

Al ser CircuitVerse el simulador elegido para implementar el procesador MIPS R2000, el capítulo actual está dedicado a describir este simulador, sus principales características, las herramientas que ofrece, las ventajas que aporta así como algunas desventajas que presenta frente a otros simuladores disponibles. Además se va a justificar por qué se ha elegido CircuitVerse para este proyecto.

2.1 ¿Qué es CircuitVerse?

CircuitVerse es un simulador de circuitos lógicos desarrollado por el International Institute of Information Technology Bangalore. Es un proyecto *Open-Source*, es decir, de código abierto, totalmente gratuito y enfocado en la enseñanza. Permite el diseño y simulación de circuitos lógicos mediante una interfaz gráfica, siendo esta muy amigable para usuarios nuevos e inexpertos en la materia. Aunque esto es cierto, CircuitVerse esconde mucha funcionalidad, permitiendo el diseño no solo de circuitos simples construidos únicamente con un puñado de puertas lógicas sino que, además, permite el desarrollo de proyectos mucho más complejos como, por ejemplo, un procesador completo.

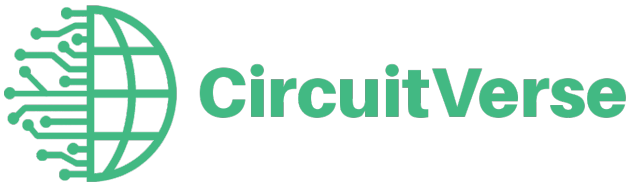

Figura 2.1: Logo del simulador CircuitVerse

La idea principal de este simulador está en la interfaz gráfica. Un circuito lógico se construye añadiendo elementos a una cuadrícula y estableciendo conexiones entre ellos. Todos estos componentes son elementos visuales disponibles en un menú flotante llamado catálogo de elementos, desde el cual se arrastran y se sueltan en el lugar deseado sobre la cuadrícula. Todos los componentes tienen propiedades y parámetros que son personalizables, y varían desde los puramente estéticos, la etiqueta que los identifica, a otros mucho más técnicos, los bits de

control y hasta el retardo que poseen, además de muchos otros dependiendo del componente en cuestión.

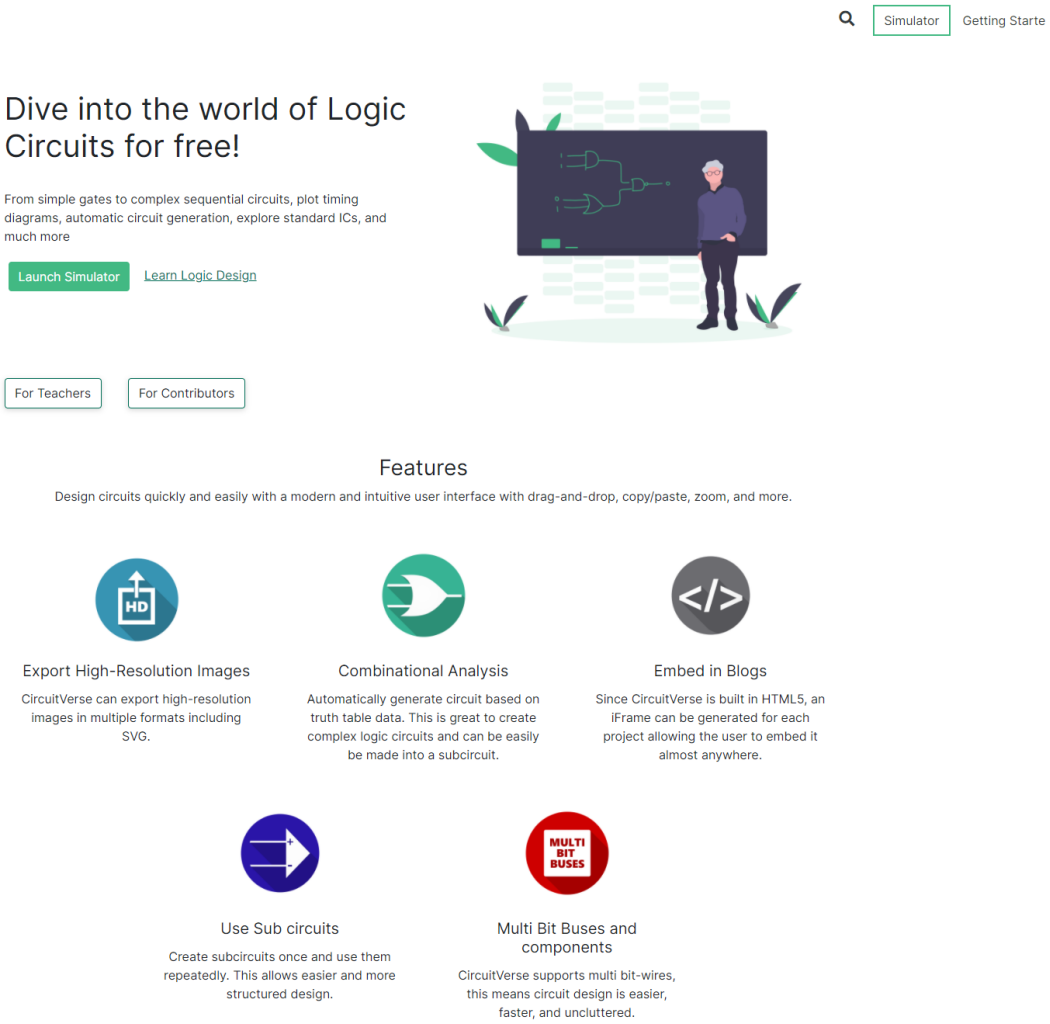

Figura 2.2: Página principal de CircuitVerse que recibe al usuario exponiendo las principales características del simulador.

CircuitVerse, ver Figura 2.1, es un simulador que se puede utilizar en línea, es accesible a través de cualquier navegador moderno, sin la necesidad de instalar nada especial. Se accede a través de la página web de la organización,[1] donde únicamente se necesita crear una cuenta para poder utilizar el simulador y empezar a diseñar circuitos lógicos.

En la Figura 2.2 se puede observar la página que nos recibe al entrar en la web de CircuitVerse. Entre otros recursos se pueden examinar circuitos lógicos diseñados por otras personas, que se pueden abrir y simular para comprender mejor su funcionamiento y los componentes que los integran.Además de los proyectos que sus creadores han hecho públicos en la página web, se pueden observar otros dos apartados en los que se muestran diseños de circuitos. En estos apartados se exponen diseños que muestran de qué es capaz el simulador, muy representativos de sus funcionalidades; algunos ejemplos son más simples, otros son circuitos complejos, llegando a estar expuestos varios diseños de procesadores de

[1] Página web de CircuitVerse: `https://circuitverse.org/`.

8 bits que utilizan las herramientas que proporciona CircuitVerse para facilitar el diseño de estos dispositivos complejos. Los apartados mencionados tienen como finalidad inspirar a los nuevos usuarios a utilizar el simulador, mostrando qué se puede conseguir con este y las facilidades que ofrece. En la Figura 2.3 se pueden observar algunos de estos diseños.

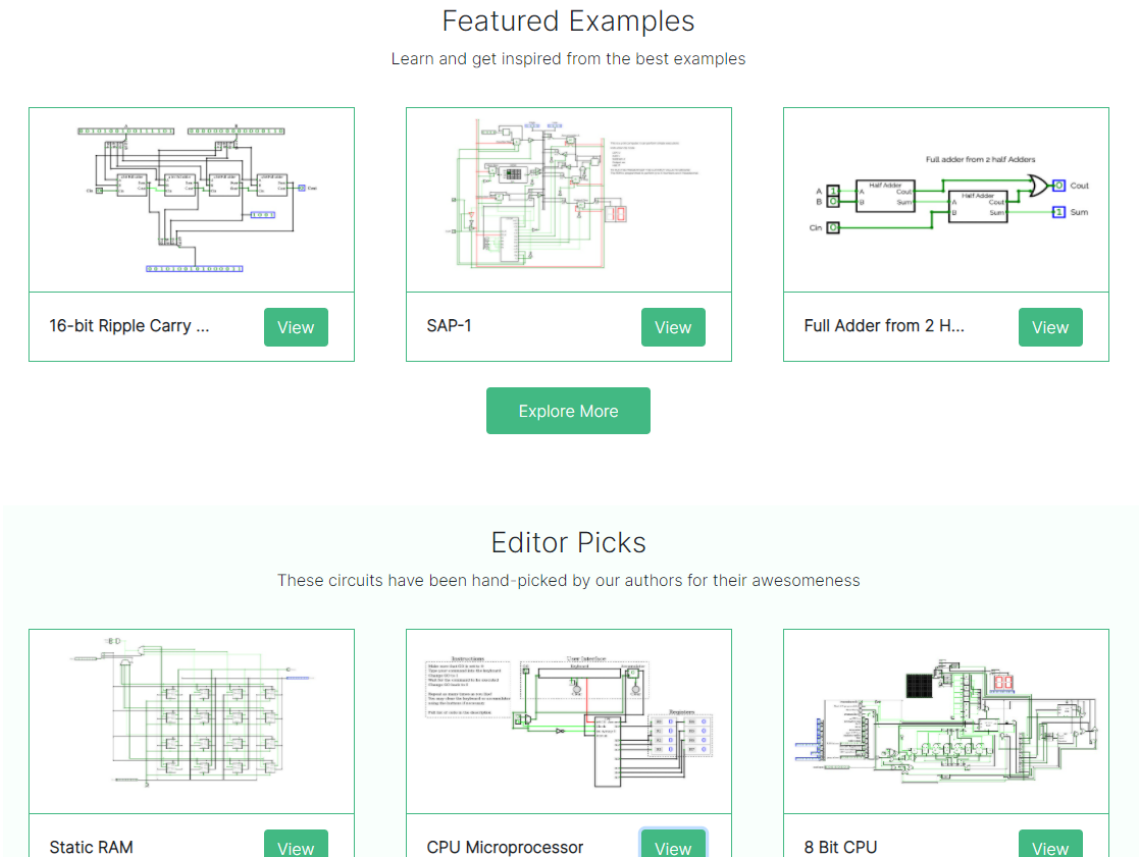

Figura 2.3: Algunos diseños creados por miembros de la comunidad mostrados en la página principal de CircuitVerse. Entre ellos podemos observar una memoria estática y dos microprocesadores.

CircuitVerse tiene en su página web un manual para usuarios noveles en su apartado *Getting Started*. Este manual recoge gran parte de los conceptos relacionados con los circuitos digitales. Empieza con aspectos muy básicos relacionados con la informática, como por ejemplo, los números binarios y las operaciones con estos números, y acaba con conceptos más complejos como, por ejemplo, máquinas de estados finitos, pasando por multiplexores, decodificadores, varios tipos de contadores, solo por poner unos ejemplos. En los capítulos donde trata componentes de circuitos lógicos están integradas instancias del simulador con el componente en cuestión o pequeños circuitos diseñados con el componente que se trata en el capítulo para que los lectores puedan interactuar con este elemento. De esta forma un usuario nuevo puede comprender por sí mismo qué uso tiene un componente dado en un circuito y los elementos que se explican pasan de ser unos conceptos abstractos a algo tangible que el usuario puede de alguna forma tocar y ver su funcionamiento y utilidad a medida que interacciona con estos.

La forma que tiene el libro interactivo de mezclar la explicación teórica con las simulaciones se puede ver en la Figura 2.4. Esta forma de aprendizaje no es po-

sible con un libro convencional y debido a ello herramientas como CircuitVerse, y más concretamente la guía interactiva que está accesible en su página web, son inmensamente útiles, porque mezclan las explicaciones teóricas con experiencias prácticas, acelerando enormemente el aprendizaje de conceptos complejos. El libro interactivo es accesible a través de esta URL.[2]

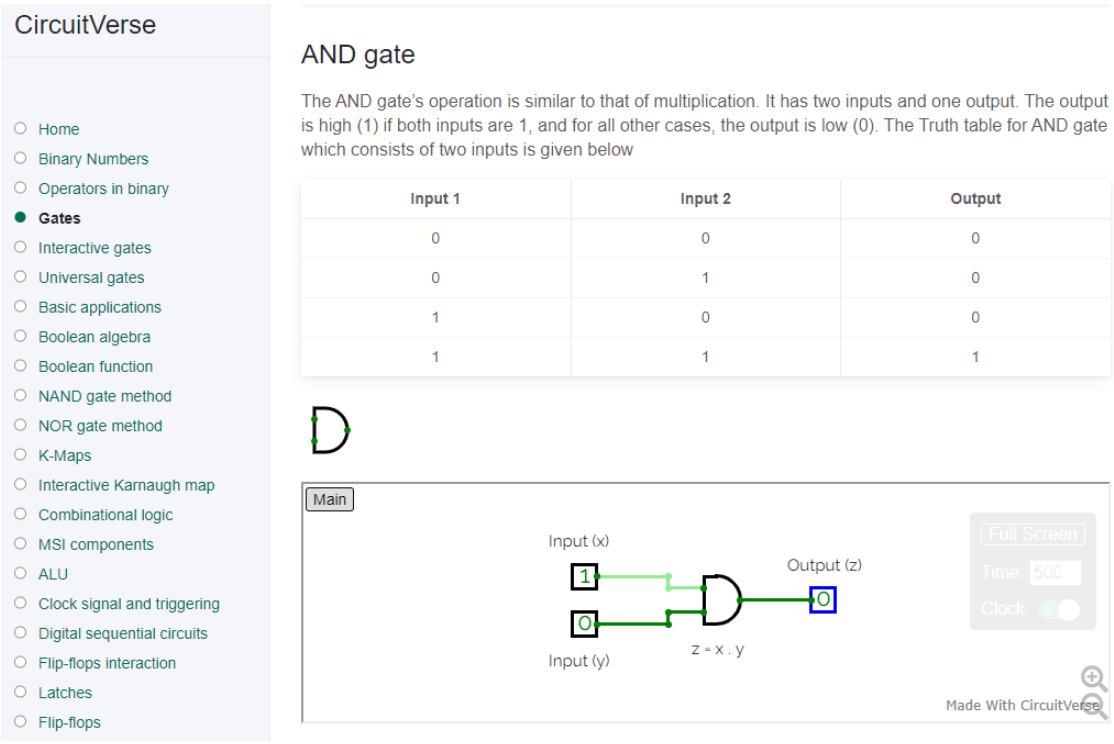

Figura 2.4: Captura del libro guía de CircuitVerse donde se explica la puerta AND, aparece la explicación sobre la puerta AND, su tabla de verdad y una simulación de esta puerta con dos entradas y una salida.

CircuitVerse además de las facilidades que proporciona para usuarios no familiarizados con los circuitos lógicos, al estar totalmente enfocado en la enseñanza, proporciona grandes comodidades también para profesores. El simulador se puede integrar fácilmente en el flujo de las clases y en las tareas que el profesor decida asignar a los estudiantes. La herramienta permite al profesor o profesora crear distintos grupos, uno para cada clase o, si los alumnos trabajan en equipos uno por equipo, a criterio del profesor. Una vez creado un grupo el maestro puede administrar sus integrantes, tiene la posibilidad de enviar invitaciones a usuarios que todavía no tengan una cuenta de CircuitVerse para que se la creen y el sistema automáticamente los incorporará al grupo cuando tengan su cuenta establecida. El profesor puede crear tareas que todos los integrantes del grupo deben hacer. Estas tareas tendrán una descripción y una fecha límite tras la cual los estudiantes no podrán seguir trabajando en la tarea, aunque el profesor tendrá la posibilidad de reabrir la tarea si así lo desea. Una vez expirada la fecha límite para la tarea, CircuitVerse permite al profesor calificar el circuito realizado por cada integrante del grupo.

Al ser un simulador que funciona totalmente en línea, CircuitVerse permite de manera sencilla exportar el código HTML5 que posibilitará añadir una simu-

[2]Libro interactivo de CircuitVerse: https://learn.circuitverse.org/.

lación del circuito a exportar en varios tipos de documentos. Por ejemplo, en una presentación o en una página web, simplemente añadiendo este código HTML al documento si este lo soporta. Automáticamente se creará la ventana para albergar el simulador con el circuito exportado y se accederá a CircuitVerse para poblar la ventana con el propio simulador. Esta función puede ser útil para profesores que deseen hacer pruebas o mostrar el funcionamiento de circuitos durante sus clases para enriquecer sus explicaciones, eliminando la necesidad de acceder a la página web de CircuitVerse o tener la aplicación ejecutándose si se trata de algún otro simulador. El hecho de estar incrustada la simulación en la propia presentación permite al profesor no cortar el flujo de su explicación accediendo a aplicaciones externas para seguir su enseñanza. Cabe destacar que la función de exportar un simulador mediante HTML5 no es exclusiva para miembros de la comunidad CircuitVerse que son profesores, es accesible para todos los usuarios.

2.2 El entorno del simulador

Al acceder a la página web del simulador CircuitVerse, inicialmente aparecerá la página principal en la cual primero se observa una breve descripción de las principales características del simulador que tiene como intención captar el interés de los usuarios nuevos, y seguidamente se presentan las características principales del simulador, cada una con una breve explicación (véase Figura 2.2). Además, a través de la página principal el usuario tiene distintos recursos disponibles para acceder, entre los cuales están las anteriormente mencionadas secciones donde están publicados circuitos diseñados por la comunidad, el libro interactivo para nuevos usuarios. A parte de los recursos citados, se observa un botón *For Contributors* que redirige al usuario a otra página que indica al usuario cómo puede contribuir al proyecto. En esta página se hace alusión a que CircuitVerse es un proyecto de código libre y sin ánimo de lucro, seguidamente se muestran varias opciones para contribuir al proyecto dependiendo del rol del usuario y qué intereses tiene en CircuitVerse; finalmente se exponen los medios por los que un usuario puede donar fondos para soportar al proyecto si así lo desea.

Volviendo a la página principal de CircuitVerse, en la parte superior derecha hay varias opciones disponibles. Una de ellas es *About*: si el usuario accede a esta página encontrará información sobre el proyecto CircuitVerse. Aquí se recalca de nuevo que su función es puramente educativa, seguidamente se puede acceder a las páginas que proporcionan al usuario información detallada sobre los términos de uso y sobre la política de protección de datos. Finalmente se exponen los perfiles GitHub de las personas que más han contribuido a CircuitVerse.

De nuevo en la página principal un usuario nuevo, sin perfil en CircuitVerse, tiene varias posibilidades. El sistema le da la capacidad para utilizar el simulador sin crear un perfil, accediendo a través del botón *Simulator* aunque de esta forma se pierden varias utilidades, entre las que está guardar su trabajo directamente en los servidores de CircuitVerse. Esta manera de trabajar sin un perfil creado podría ser útil para alguien que simplemente quiere diseñar un circuito lógico pequeño, bien para testear este circuito o bien para probar las funcionalidades de la herramienta CircuitVerse antes de decidir si el simulador le será útil para llevar

a cabo su proyecto y en caso afirmativo entonces realizar los pasos para crear un perfil.

Aunque siendo el anterior un punto de vista totalmente válido, cabe destacar que el proceso de crearse un perfil en CircuitVerse es muy sencillo. Primero un usuario nuevo entraría a la página *Login* que es accesible a través del botón del mismo nombre que se encuentra en la parte superior derecha de la página principal junto con las demás opciones antes mencionadas. La página *Login* es el portal mediante el cual los usuarios registrados acceden a su perfil. En caso de ser usuario nuevo existen varias opciones, este puede utilizar su cuenta de Google, Facebook o GitHub para que el sistema automáticamente le cree un perfil y las próximas veces acceder mediante este. Si el usuario que ha decidido registrarse no desea vincular alguna de sus cuentas externas a CircuitVerse, tiene la opción de crearse un perfil directamente en CircuitVerse accediendo al formulario mediante *Sign In*, para este únicamente necesita ingresar su nombre, correo electrónico y una contraseña.

Tras completar estos pasos el sistema llevará al usuario a su *Dashboard* que en castellano se podría entender como página principal, que es el lugar desde el cual los usuarios registrados acceden a todos sus proyectos. Además este es el lugar a través del cual los usuarios pueden editar su perfil, por ejemplo agregando una foto o la organización a la que pertenecen. En el *Dashboard* se observan tres pestañas una para los proyectos propios, una donde se encuentran proyectos de la comunidad guardados para ser accesibles cuando se desee y la última donde tendremos los proyectos de otros usuarios en los que colaboramos. Estando en el *Dashboard* el usuario puede abrir un proyecto en el simulador para seguir con su implementación o abrir directamente el simulador con un entorno vacío para comenzar con el diseño de un nuevo circuito.

Una vez abierto el simulador propiamente dicho se encuentra con un entorno como el que se puede observar en la Figura 2.5. En este entorno se observa una cuadrícula de fondo, la cual hace de mesa de trabajo sobre la que se irán añadiendo los elementos y conexiones que se crean oportunos. A parte de la cuadrícula existen varios menús flotantes y una barra de menús adicional en la parte superior de la página.

El menú de las propiedades, llamado *Properties* en la interfaz del simulador dado que el simulador actualmente está únicamente en inglés, es el menú que permitirá al usuario modificar todos los atributos del elemento seleccionado. En el caso de que no esté seleccionado ningún componente se podrán modificar las propiedades del proyecto y del circuito sobre el que se trabaje actualmente. Es posible modificar el nombre del proyecto y del circuito, ajustar la frecuencia de reloj y desactivar el reloj completamente en caso de que no se precisara para el circuito en cuestión (Véase la Figura 2.6). Otra propiedad que se permite modificar es alterar el aspecto que tendrá el circuito en caso de que se integre en otro circuito como un subcircuito o componente: apretando *Edit Layout* se llega a otra pantalla donde se muestra el aspecto que tiene el componente visto desde fuera actualmente; en ese momento el usuario es capaz de modificar la estatura, la anchura del componente y el lugar donde estarán sus entradas y salidas, entre otros aspectos menores. En caso que se haya seleccionado un componente del circuito

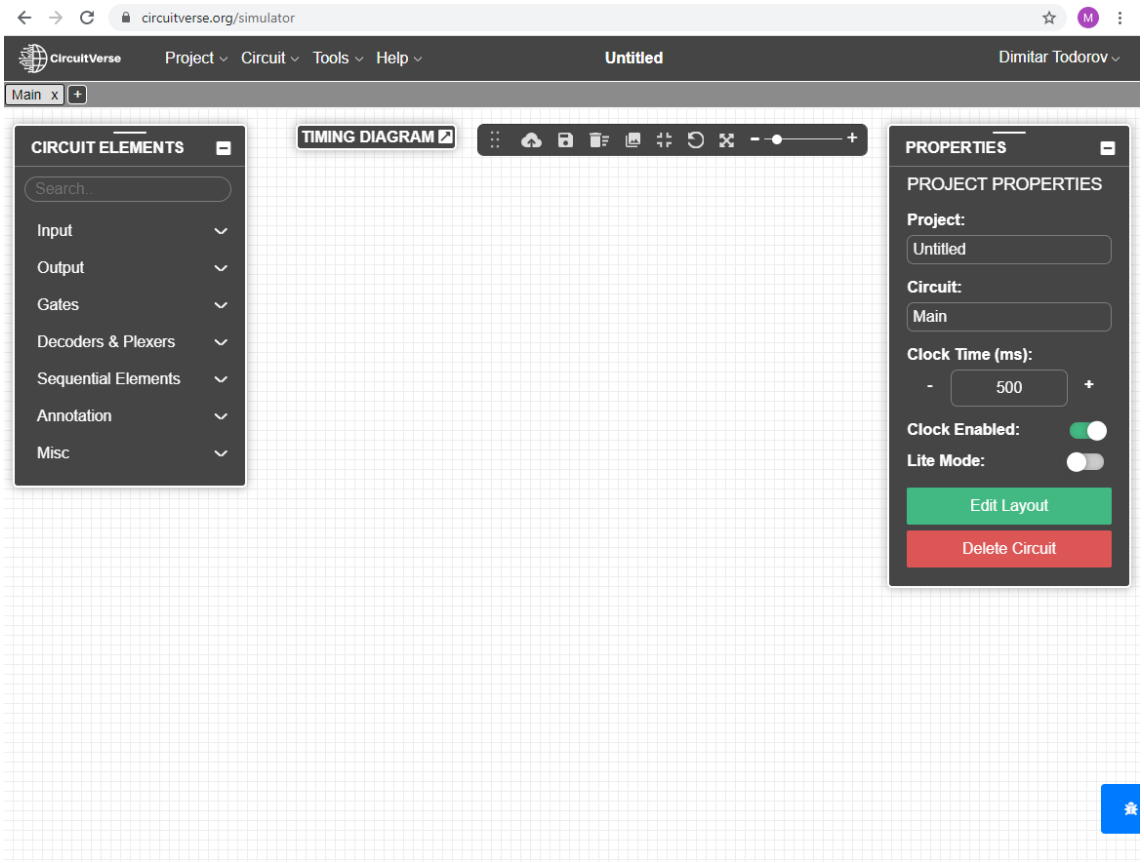

Figura 2.5: Entorno de trabajo vacío del simulador CircuitVerse donde el usuario añade los componentes de sus circuitos y además puede crear y navegar entre los diferentes subcircuitos, si los tuviera, utilizando las pestañas de la parte superior.

desde el menú de las propiedades se podrán modificar todos sus atributos para adecuarlo a los requisitos del circuito.

Cada componente tiene distintas características que podremos modificar, estas propiedades varían desde el número de bits que recibe el componente, el ancho en bits de la salida del componente, el número de bits de control, entre otros dependiendo del elemento seleccionado. Una propiedad posible de modificar que cabe recalcar es el retardo que presenta el componente seleccionado. Esto permite simular circuitos con temporización mucho más precisa y acorde a la realidad, dado que los circuitos reales no presentan retardo nulo y, dependiendo de su complejidad, son más o menos lentos.

Otra característica interesante que es posible modificar es la relacionada con el aspecto, como la dirección en la que apunta el componente, el lugar que ocupa la etiqueta o el texto de la etiqueta. La imagen de la derecha en la Figura 2.6 muestra un ejemplo de este menú habiendo seleccionado un multiplexor. Desde este menú existe la opción de acceder a la página del manual del simulador referente al componente seleccionado, en caso de que se necesite información adicional sobre este.

Otro menú en pantalla es el mostrado en la Figura 2.7. Este menú flotante contiene varios mandos para el circuito actual o todo el proyecto con una funcionalidad variada. Desde aquí se puede guardar el proyecto tanto *on-line* como

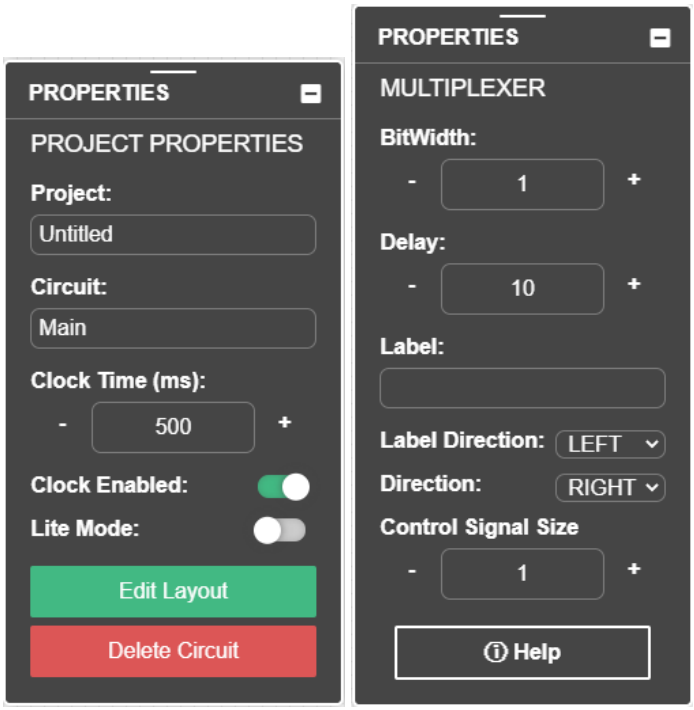

Figura 2.6: Menú *Properties* del simulador, a la derecha sin ningún elemento seleccionado y a la izquierda estando seleccionado un multiplexor.

Figura 2.7: Menú con controles para la vista y el guardado.

off-line bajándolo al equipo del propio usuario. Una funcionalidad muy útil accesible a través de este menú es la posibilidad de guardar una foto del circuito o de una parte suya; CircuitVerse te permite además elegir el formato de la imagen y la resolución de esta. La funcionalidad anterior es muy práctica para todo aquel que quiere incluir su circuito o una parte de este como contenido de otro documento, por ejemplo en un documento PDF o una presentación. Además, desde el menú actual se pueden deshacer acciones y controlar la vista sobre el circuito.

El menú observado en la Figura 2.8 es el encargado de hacer disponibles todos los componentes accesibles en CircuitVerse para su inclusión en el circuito del usuario. Estos elementos están recogidos en categorías. Algunos de los componentes disponibles están diseñados para simplificar el diseño de circuitos lógicos. La existencia de estos componentes más complejos directamente disponibles para su uso en CircuitVerse se debe a que el simulador tiene fines didácticos. Un ejemplo interesante del tipo de componentes antes mencionado es una ALU disponible para su uso directo en los circuitos. En la entrada del manual de CircuitVerse sobre este elemento están disponibles los códigos de control de todas las operaciones que soporta e información auxiliar sobre su uso.[3] Otro tipo de elementos accesibles son campos de texto, anotaciones, entre otros, disponibles

[3]Entrada del manual de CircuitVerse para el componente ALU: https://docs.circuitverse.org/#/miscellaneous?id=alu.

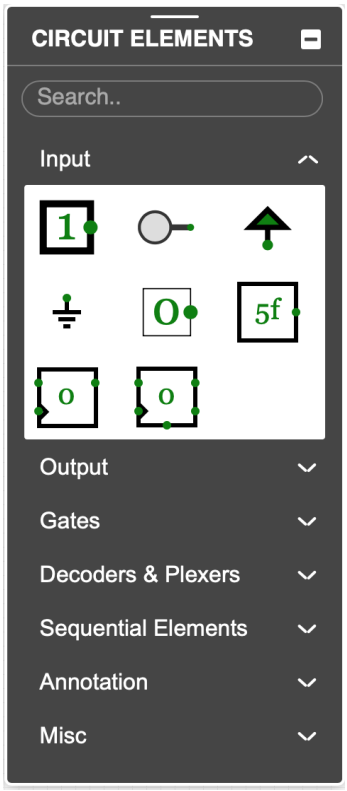

Figura 2.8: Menú con todos los componentes disponibles, donde se puede observar desplegada la categoría que contiene los componentes de entrada.

todos en la categoría *Annotation*, no tienen otro efecto sobre el circuito que su utilidad visual.

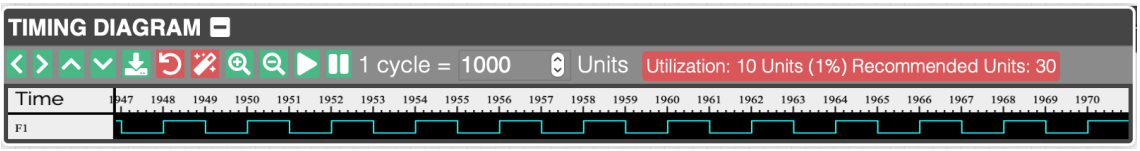

Figura 2.9: Menú que muestra los diagramas temporales y contiene los controles de estos diagramas.

Como último menú flotante en el simulador se observa el menú *Timing Diagram*, (Figura 2.9). Este menú, utilizado junto a un elemento llamado *Flag*, permite observar valores en un diagrama temporal. El componente *Flag* se puede conectar a cualquier conexión, tanto entrada como salida, como una conexión intermedia entre componentes. Por cada elemento de este tipo que hay en el circuito el sistema crea un diagrama temporal en el menú antes mencionado. Además cada componente *Flag* muestra en decimal el valor que recibe en su entrada. En sus propiedades se puede establecer el número de bits que se reciben, permitiendo de esta forma conectarse tanto a conexiones de un bit de ancho como a buses de anchura variada. El menú, además de mostrar el diagrama temporal de cada *Flag*, permite pausar los diagramas, moverse adelante y atrás por ellas, establecer la duración de un ciclo del diagrama, entre otras opciones. Otra interesante capacidad que presenta el sistema es la posibilidad de guardar una parte de los diagramas temporales en el sistema del usuario como una imagen, para un próximo examen o consulta.

En la Figura 2.10 se puede observar una puerta NAND de dos entradas con un *Flag* conectado a su salida. Para este *Flag* con identificador F1 en el menú *Timing Diagram* aparece un diagrama temporal en el que se puede observar como varia su salida a medida que se modifican sus entradas.

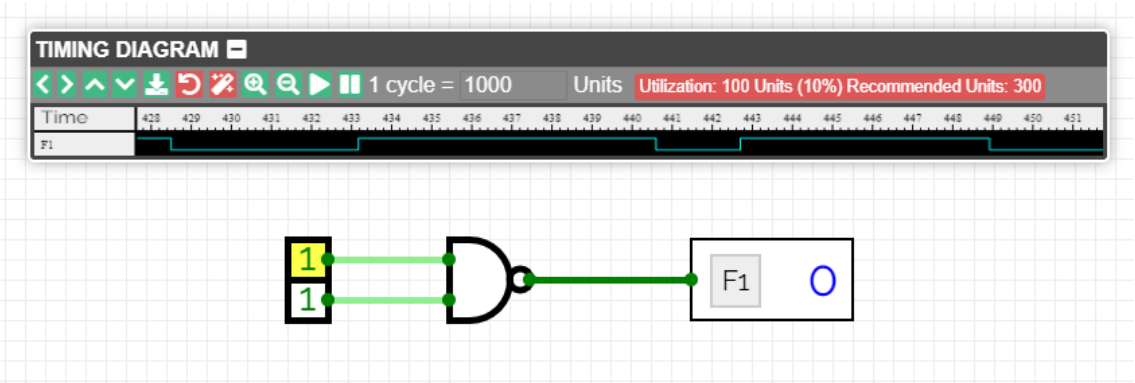

Figura 2.10: Ejemplo en el que se observa el uso del menú *Timing Diagram* junto con un componente *Flag* conectado a la salida de una puerta NAND.

Otra funcionalidad interesante que permite CircuitVerse es crear una tabla de verdad y a partir de esta tabla de verdad el simulador automáticamente creará un circuito que implemente la funcionalidad establecida por la tabla de verdad. Además admite introducir directamente la función booleana que se desee implementar mediante circuito, sin necesidad de crear la tabla de verdad asociada a esta, el propio sistema la crea según la función booleana introducida y te permite antes de crear el circuito modificar la tabla de verdad en caso que se precisará alguna modificación. Este es un aspecto del simulador muy útil que permite crear circuitos de cierta complejidad de manera muy rápida si se conoce la función booleana que se implementará, o de otro modo la tabla de verdad del circuito.

Como último rasgo interesante del simulador hay que destacar la posibilidad de utilizar subcircuitos. Esto es, utilizar un circuito como componente de otro circuito más complejo o de mayor nivel. Existen varias formas de incluir un subcircuito a otro: se puede importar desde otro proyecto disponible en CircuitVerse o crearlo en el mismo proyecto que el circuito principal para su posterior inclusión como componente de este. Una vez importado el subcircuito se crea una pestaña para él, accediendo a la pestaña puedes modificar el subcircuito como si fuera un circuito más del proyecto, los cambios se verán reflejados inmediatamente en el funcionamiento del circuito principal. Además, como fue mencionado antes, se puede modificar el aspecto del subcircuito, esto es, donde están sus entradas, salidas, su nombre, entre otras. La posibilidad de utilizar subcircuitos permite simplificar circuitos complejos, reduciendo los componentes en pantalla y por consiguiente permite visualizar mejor los diferentes elementos que componen un circuito complejo facilitando su comprensión.

Antes de terminar cabe hacer hincapié en que, a finales del año 2020, fue lanzada una nueva versión de CircuitVerse que introdujo grandes mejoras en la interfaz, modernizándola, introduciendo un diseño más simplista y agradable a la vista, permitiendo cierta personalización al permitir mover ciertos menús por pantalla al lugar que mejor le parezca al usuario. Otra notable inclusión es la introducción de soporte para código Verilog. Este es un lenguaje de descripción de

hardware utilizado para modelar circuitos, el más empleado en todo el mundo junto con VHDL. CircuitVerse permite tanto crear módulos en Verilog para su posterior integración en un circuito, como exportar circuitos y proyectos creados en CircuitVerse a Verilog. La inclusión de Verilog a CircuitVerse abre nuevos horizontes al simulador, permitiendo la compatibilidad de circuitos hechos en este con otras herramientas disponibles que soporten Verilog y posibilitando incluso el diseñar un circuito en CircuitVerse y llevarlo a la realidad.

2.3 Estado del arte

Existen muchos simuladores que permiten diseñar, implementar y simular circuitos digitales, aparte de CircuitVerse. Es interesante mencionar algunos, exponer sus principales características y compararlos con el simulador escogido para este proyecto.

Figura 2.11: Logo del simulador EasyEDA.

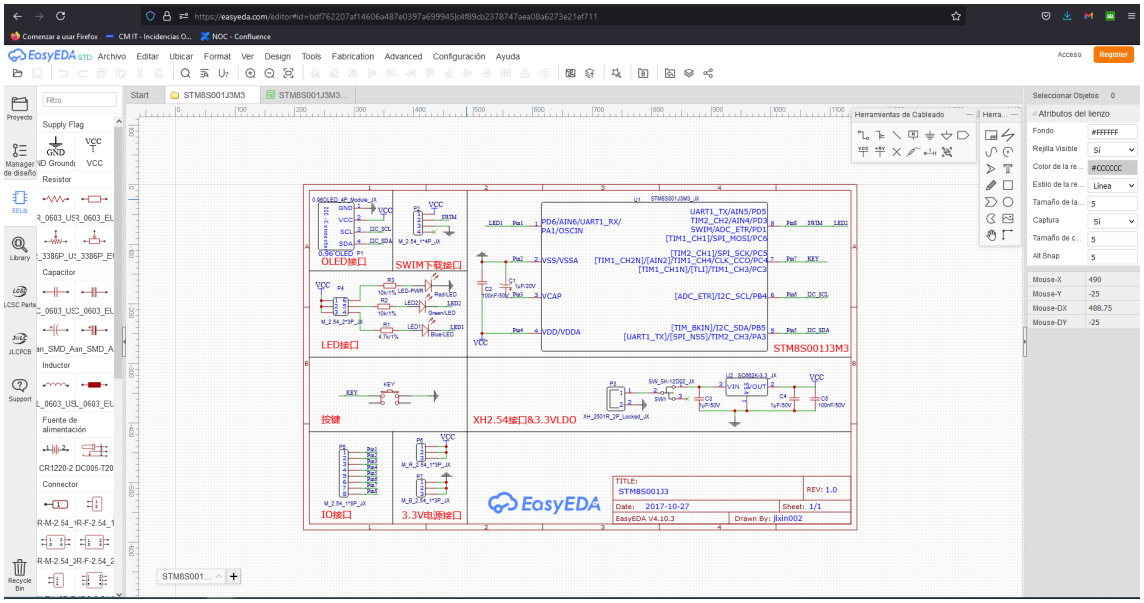

Figura 2.12: Entorno de trabajo del simulador EasyEDA donde podemos observar abierto uno de los ejemplos creados por los desarrolladores de la herramienta.

EasyEDA[4] es un simulador de circuitos digitales accesible desde internet y gratuito. Es una herramienta muy conocida y utilizada tanto por estudiantes como por profesionales. EasyEDA y CircuitVerse, aunque los dos son simuladores de circuitos digitales, son muy diferentes. EasyEDA, al ser una herramienta dirigida a profesionales, ofrece más posibilidades para poder abarcar más ámbitos y satisfacer las expectativas del mayor número de usuarios posibles. En cambio, esto mismo que beneficia a los profesionales, aumenta la complejidad de la herramienta, haciendo crecer la curva de aprendizaje para estudiantes. Además el diseño en el simulador EasyEDA comienza en la capa física, obligando a los usuarios a lidiar con transistores, resistencias, diodos y tener en cuenta intensidades y voltajes. Todos ellos conceptos de electrónica que puede ser compliquen el uso de este simulador innecesariamente a un estudiante cuyo aprendizaje no va enfocado en estos ámbitos.

Con todo esto establecido, las diferencias entre CircuitVerse y EasyEDA están claras, este último es una herramienta dirigida más a profesionales, en cambio CircuitVerse al estar centrada en la enseñanza reduce la complejidad innecesaria en beneficio de la claridad y la sencillez de uso.

Figura 2.13: Logo del simulador PartSim.

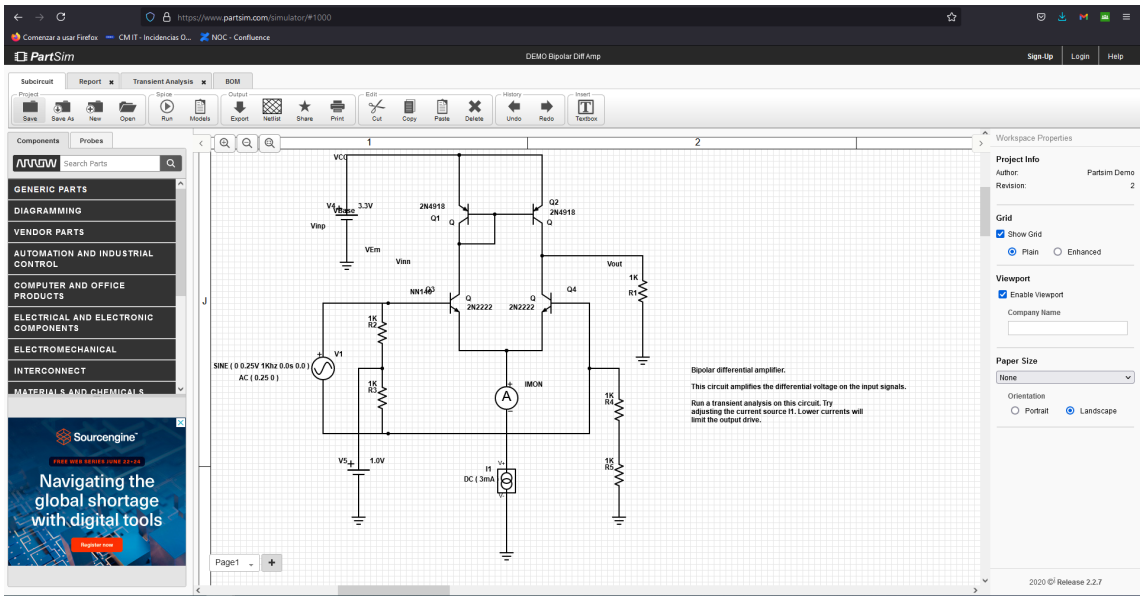

Figura 2.14: Entorno de trabajo del simulador PartSim en el que está abierto el diseño de un amplificador de voltajes.

PartSim[5] es una alternativa al simulador EasyEDA mencionado anteriormente y, al igual que este, es muy utilizado, ofreciendo para el diseño de los circuitos digitales el uso de componentes reales. Es una herramienta también centrada

[4]Página web de EasyEDA. https://easyeda.com/es.
[5]Página web de PartSim. https://www.partsim.com.

mayormente en electrónica ofreciendo grandes posibilidades para crear circuitos desde cero a nivel de transistor, resistencia y diodos pero recae en el mismo problema que EasyEDA, todas estas posibilidades que ofrece incrementan la complejidad del simulador. Aunque las diferencias entre PartSim y CircuitVerse son claras cabe destacar que sus interfaces son similares, siguiendo un diseño parecido y una organización de los menús semejante.

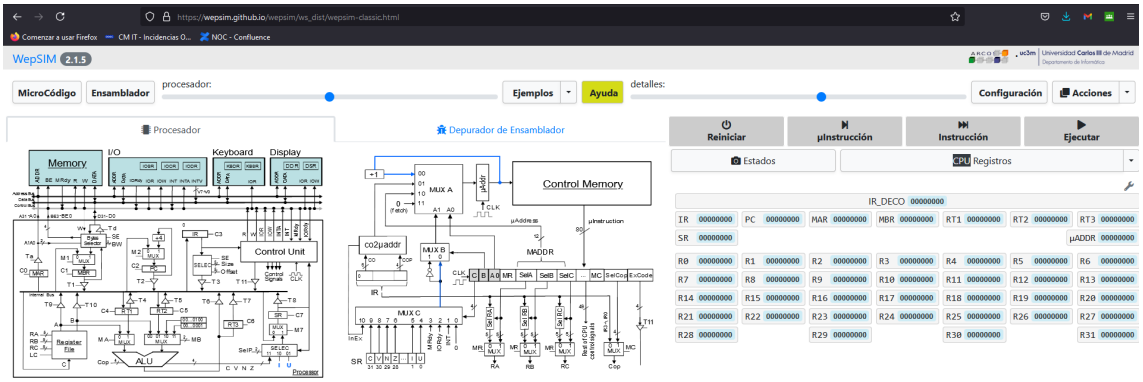

Figura 2.15: Entorno de trabajo del simulador WepSIM en el que podemos observar los componentes del procesador y su interconexión en la parte izquierda y al lado derecho de la ventana se puede contemplar el contenido de los registros en cada ciclo de reloj.

WepSIM[6] es un simulador de procesadores. Al igual que CircuitVerse, está totalmente enfocado en la enseñanza, aunque su enfoque y objetivos son distintos. WepSIM está enfocado únicamente en la simulación de procesadores y sus componentes, además ofrece la posibilidad de escribir código en lenguaje ensamblador y posteriormente ejecutarlo sobre el procesador simulado. WepSIM permite crear nuevos procesadores, junto con sus correspondientes juegos de instrucciones. Este simulador está preparado para crear automáticamente la correspondencia entre las distintas instrucciones pertenecientes al juego de instrucciones y el código binario subyacente con el que trabaja el procesador en cuestión. WepSIM está dirigido principalmente a alumnos de la asignatura Estructura de Computadores del grado de Ingeniería Informática y por ello es más específico que CircuitVerse, en WepSIM podemos simular todos los componentes de un procesador, incluyendo el juego de instrucciones. Aquí reside la principal diferencia con el simulador CircuitVerse, este no está únicamente centrado en procesadores, CircuitVerse permite diseñar y simular circuitos digitales de diversa índole y por tanto, ser utilizado en más ámbitos que WepSIM, que por su parte está centrado completamente en el diseño y simulación de procesadores [5].

La principal característica que diferencia CircuitVerse de otros simuladores de circuitos lógicos es su sencillez: su forma de construir circuitos de forma totalmente gráfica facilita su uso por usuarios noveles. Además, al ser un simulador reciente está a la par de las tendencias actuales de simplificar las interfaces gráfi-

[6]Página web del simulador WepSIM. https://wepsim.github.io.

cas dejando solo lo esencial, permitiendo de esta forma mantener la atención del usuario en lo que realmente interesa, el circuito diseñado, a la vez que es agradable a la vista al no haber sobrealimentación de información visual. CircuitVerse asimismo al ser online es claramente multiplataforma y no está restringido solo a uno o a unos pocos sistemas operativos específicos. Su principal virtud, en este caso su sencillez, puede ser al mismo tiempo su mayor desventaja frente a otros simuladores disponibles. Algunos usuarios, principalmente usuarios experimentados en los circuitos lógicos, pueden verse a veces faltos de algunas funciones complejas y esto puede impedir utilizar CircuitVerse para sus proyectos, funciones que otros simuladores del mercado sí ofrecen. Aunque siendo esto cierto los desarrolladores de CircuitVerse siempre han recalcado que el objetivo del simulador es didáctico y por tanto, no intentan competir con los simuladores existentes, más dedicados al ámbito profesional.

Para finalizar este capítulo es necesario destacar que CircuitVerse es una nueva herramienta totalmente enfocada en la enseñanza y en continua evolución. Desde que se lanzó el proyecto al público se fueron introduciendo numerosas mejoras, impulsadas por el equipo detrás del proyecto y por el hecho de que el simulador es de código abierto, es decir, su código fuente está disponible para todo el mundo para modificarlo y hacer mejoras. Además CircuitVerse cuenta con una comunidad apasionada formada por estudiantes, profesores, programadores y gente con intereses en los circuitos lógicos en general. Todos estos aspectos juntos propician un clima de desarrollo para CircuitVerse pudiendo este convertirse en un referente en cuanto a la enseñanza de circuitos lógicos en los años venideros.

Cualquier usuario que se familiarice con el simulador y tenga interés en el diseño de circuitos quedará muy satisfecho y tendrá una herramienta que le será útil en años próximos.

CAPÍTULO 3
El procesador MIPS R2000

Este proyecto se basa en el procesador MIPS R2000, por tanto, el actual capítulo está destinado a familiarizar al lector con este procesador, con sus características, ventajas y desventajas. Además se hará una descripción del mercado en la época en la que se concibió, sus competidores y por qué el MIPS R2000 es muy utilizado para la educación.

3.1 Conceptos básicos

Un ordenador es una máquina formada por varios componentes que funcionan en perfecta sinergia. Sin duda, el elemento más importante de todo este sistema es el procesador, o CPU, por sus siglas en inglés, *Central Processing Unit*. El resto de componentes giran entorno a la CPU, apoyando su trabajo.

El procesador tiene muchas funciones, pero la más básica es ejecutar una instrucción en lenguaje máquina y producir un cambio en el sistema, según qué le indicaba esta.

Un procesador se puede ver como una conglomeración de transistores, incluso es interesante mencionar que el poder computacional y capacidades de una CPU muchas veces se mide en la cantidad de transistores que la componen. Aunque este método de medición no es totalmente correcto, da una idea general de las capacidades de un procesador. Por ejemplo, el procesador Intel 4004 considerado el primer microprocesador, contenía únicamente 2300 transistores. Un procesador de gama alta actual puede albergar alrededor de 23.000.000.000 transistores, Figura 3.1.

Aunque un procesador se pueda ver como una agrupación de transistores, lo importante es cómo están organizados esos transistores.

La CPU está formada por una serie de componentes, cada uno de los cuales tiene una función específica dentro del sistema complejo que es un procesador. Algunos componentes básicos que tienen todos los procesadores son los registros, la unidad aritmético-lógica, la unidad de control y el contador de programa, entre otros. Hablaremos de los anteriores elementos con más profundidad a continuación en el capítulo. Estos componentes junto con algunos más forman la llamada ruta de datos. Además de los elementos antes mencionados una parte

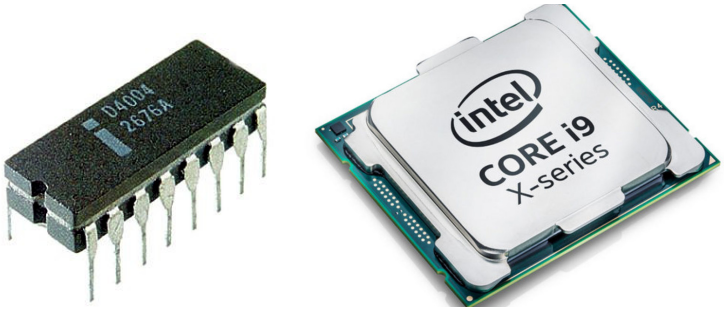

Figura 3.1: En la imagen de la izquierda vemos el procesador Intel 4004, esta CPU tiene 4 bits de ancho de palabra y 16 pines. En la imagen de la derecha aparece el procesador Intel Core i9-10980XE, este tiene 64 bits de ancho de palabra y 2066 pines de conexión.
Figura del Intel 4004 extraída de la página web
https://es.wikipedia.org/wiki/Intel_4004
Figura del Intel Core i9 extraída de la página web
https://www.theitdepot.com/details-Intel+Core+i9-10980XE+Extreme+Edition+3.00+GHz+Processor_C30P33432.html

íntegra de la ruta de datos es la memoria, de datos y de instrucciones, aunque esta no se encuentre dentro del propio dado del procesador.

La ruta de datos se puede entender de manera simplificada como el camino que sigue una instrucción hasta ejecutarse completamente y por consiguiente producir un resultado o un cambio en el sistema, que es el computador.

Hay varios requisitos o capacidades generales que un procesador debe cumplir para llevar a cabo correctamente sus funciones. Estas funciones necesarias en toda CPU son [9]:

- Buscar y leer una instrucción de memoria
- Decodificación de la instrucción
- Leer los datos necesarios para la ejecución
- Ejecución, normalmente alguna operación aritmética o lógica
- Escritura de los datos obtenidos

Cada procesador, dependiendo de la arquitectura que implementa, llevará a cabo estas funciones en distinto orden, utilizando distinta tecnología y diferente estructura.

El concepto de arquitectura se refiere a cómo se interconectan los componentes de un procesador y el diseño interno de estos. A lo largo de los años han existido multitud de diferentes arquitecturas de distintos fabricantes, algunos ejemplos notables son: la arquitectura PowerPC desarrollada por IBM, Apple y Motorola; la arquitectura x86 diseñada por Intel; la arquitectura ARM desarrollada por Advanced RISC Machines y por último la utilizada en este proyecto, la arquitectura MIPS diseñada por MIPS Technologies.

En la actualidad las más utilizadas son la arquitectura x86 de Intel, empleada también por su competidor AMD, y la arquitectura ARM de Advanced RISC

Machines. Los procesadores basados en x86 son mayormente utilizados en ordenadores personales, portátiles y de sobremesa y son muy utilizados en servidores. En cambio los procesadores de la familia ARM son utilizados principalmente en dispositivos móviles, aunque desde hace un tiempo se intenta llevar esta arquitectura a los ordenadores personales, debido a su eficiencia.

Entre las dos arquitecturas antes mencionadas existe una diferencia importante. La primera de ellas, la arquitectura x86 de Intel, es una arquitectura CISC, del ingles *Complex Instruction Set Computer*. Los procesadores CISC cuentan con un conjunto de instrucciones muy amplio y versátil que permite realizar operaciones complejas con una sola instrucción. La arquitectura ARM, y en este caso la MIPS, siguen la filosofía opuesta a los procesadores CISC, los procesadores de estas arquitecturas son procesadores RISC, del inglés *Reduced Instruction Set Computer*. Las CPU RISC tienen conjuntos de instrucciones reducidos, donde las propias instrucciones son más simples y generalmente de tamaño fijo. Estas instrucciones simples son mucho más rápidas de ejecutar que las lentas instrucciones complejas presentes en los procesadores CISC, aunque para una misma operación requieren más instrucciones.

3.2 El camino de una instrucción en el MIPS R2000

Empezaremos describiendo la ruta de datos que sigue una instrucción en el MIPS R2000 y explicando los elementos a medida que estos aparezcan en la ruta de datos, esta se puede observar en la Figura 3.2. No se entrará en detalles del tipo de instrucción ni en la temporización específica de una instrucción, esto se explicará con mayor profundidad más adelante. Se hablará en términos de una instrucción genérica.

El primer elemento que actúa durante la ejecución de una instrucción es el registro del contador de programa o PC, del inglés *program counter*, a este registro se accede para recuperar la dirección de memoria de la instrucción que se ejecutará a continuación. El contador de programa tiene un sumador que añade 4 al valor del registro antes mencionado cada vez que se ejecuta una instrucción, haciendo que el PC apunte a la siguiente instrucción a ejecutar, que estará en la dirección de memoria de la instrucción actual más 4. Excepto si la instrucción ejecutada anteriormente es una instrucción de salto, que permite modificar el contenido del registro del contador de programa para que apunte a la dirección de memoria indicada en la instrucción de salto.

Teniendo al contador de programa apuntando a la instrucción que se ejecutará, se accede a la memoria de instrucciones recuperando la instrucción almacenada en esta dirección.

Los bits de la instrucción recuperada inundan el bus de la ruta de datos permitiendo a cada elemento de esta acceder a los bits que interesan en cada caso.

Son varios los componentes que cambian sus estados en este momento, aunque de especial interés es la unidad de control. Este componente es el encargado de enviar las señales de activación correspondientes al resto de elementos dependiendo del tipo de instrucción. Todos los componentes dentro de la CPU son controlados por las señales que les llegan de la unidad de control. Esta toma una

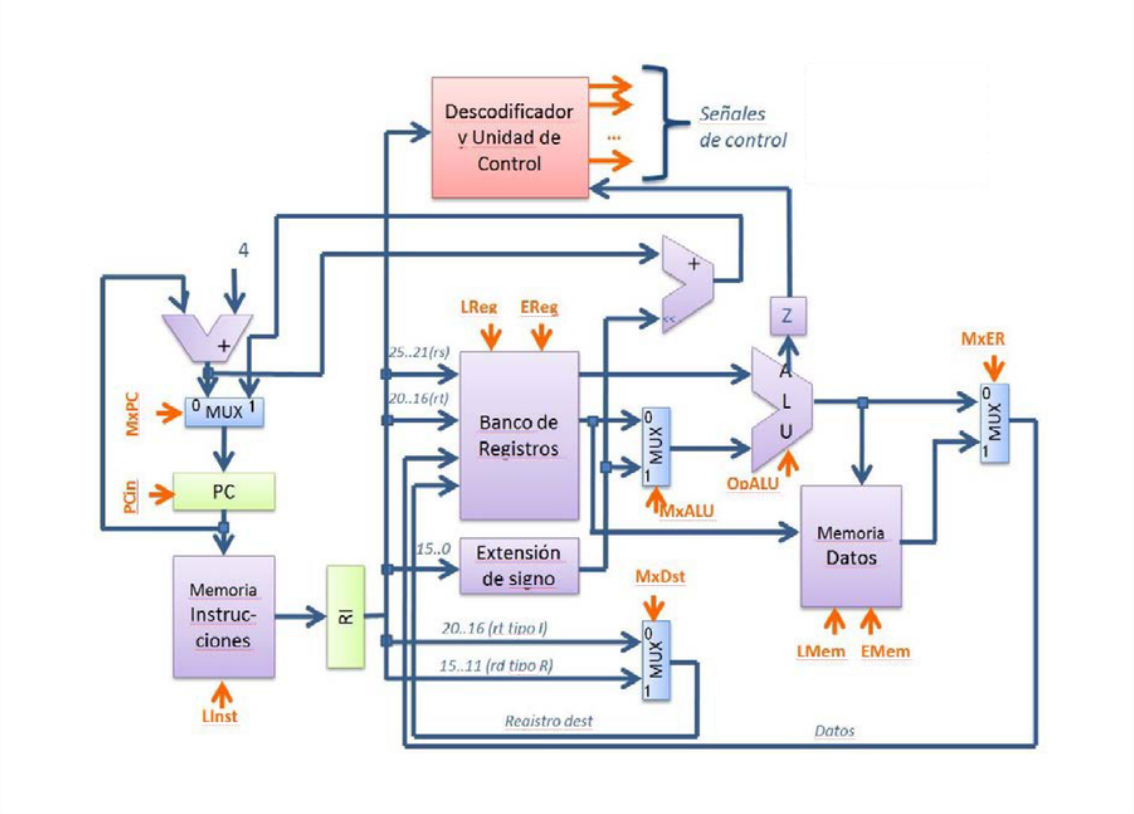

Figura 3.2: Ruta de datos monociclo del procesador MIPS R2000 con todos sus componentes y conexiones que existen entre ellos. Las señales de control están resaltados con color naranja.
Figura obtenida del material de la asignatura Estructura de Computadores

parte de los bits de la instrucción y los utiliza para reconocer de qué instrucción se trata y por tanto activar las señales oportunas.

El banco de registros mantiene la información más inmediata con la que trabaja el procesador, se trata de una memoria muy reducida aunque muy rápida. Por ejemplo, el MIPS R2000 dispone únicamente de 32 registros para números enteros y 32 registros más para números en coma flotante, aunque estos últimos no se encuentran en el mismo banco de registros.

En el mismo instante en que la unidad de control descifra qué instrucción es, el banco de registros lee los bits de la instrucción que corresponderían a las direcciones de los registros que se leerían y escribirían. Las salidas del banco de registros cambian con la información contenida en los registros que se deben leer, en este momento todavía no se conoce si esta información se utilizará, debido a que no se conoce el tipo de instrucción con la que se trata. Consumir o no la información de estos registros dependerá de las señales que reciban los multiplexores que están a las salidas del banco de registros.

Del mismo modo escribir en el registro al que apuntan los bits de la instrucción convenientes no se efectuará mientras no se active la señal de escritura en registro proveniente de la unidad de control. Es interesante mencionar que la operación de escritura en registro suele estar retardada para permitir que se cree el valor que se desea guardar. Por ejemplo, que este se calcule en la unidad aritmético-lógica o se obtenga de memoria, dependiendo de la instrucción.

Tras activarse las señales de control adecuadas y leerse los registros necesarios, los bits o datos llegan a la unidad aritmético-lógica y esta realizará sobre ellos la operación indicada por la unidad de control.

Una vez el resultado está listo los bits obtenidos tras la operación pueden ir a los registros o a la memoria de datos. Dependiendo del tipo de instrucción incluso puede que el resultado obtenido de la unidad aritmético-lógica sea la dirección que se desea leer de la memoria. Esto depende de las señales de control activadas por la unidad de control.

Concurrentemente a esta serie de acciones se ha aumentado en 4 el valor guardado en el registro del contador de programa para que este registro apunte a la próxima instrucción a ejecutar.

Como se ha aludido antes dependiendo del tipo de instrucción el resultado y los elementos de la ruta de datos que se utilizan durante la ejecución de esta pueden variar, teniendo tres tipos de instrucciones principales. Las de tipo R, las de tipo I y las de tipo J. El tipo de instrucción define el formato de esta, es decir, qué indican cada grupo de bits de los 32 presentes en una instrucción del MIPS R2000. Los diferentes tipos de instrucción se comentarán con más detalle en una sección específica de este capítulo, dedicada al conjunto de instrucciones del MIPS R2000.

3.3 Características principales del MIPS R2000

Para comenzar a comentar las características del MIPS R2000 hay que aludir al hecho que en cuanto a su uso en la educación existen dos versiones principales. La primera con una ruta de datos relativamente más simple que la segunda. Este capítulo se centrará en la ruta de datos más simple, ya que en ella se basa este proyecto. No obstante se darán unas pinceladas generales de la ruta de datos segmentada, es decir, la más compleja de las dos.

La primera ruta de datos mencionada anteriormente es la así llamada, ruta de datos monociclo. Su principal característica, como su nombre indica, es empezar a ejecutar una instrucción y terminar de ejecutarla antes de terminar el ciclo.

El MIPS R2000 se diferencia del resto de los procesadores de su época justo en este enfoque que persigue, ejecutar todas las instrucciones en un solo ciclo. Además sus diseñadores tienen como objetivo conseguir que este ciclo sea más corto que el de sus rivales. Para ello, reducen el juego de instrucciones del MIPS R2000 solo a unas instrucciones más simples, aunque al mismo tiempo, las más utilizadas en los programas [2, 8].

El enfoque del equipo detrás del MIPS es bastante inusual. Sus competidores se centraban en crear instrucciones para satisfacer las necesidades de distintas aplicaciones específicas, por ejemplo, científicas. De este modo los diseñadores rivales preferían diseñar procesadores con instrucciones más complejas que requerían varios ciclos de ejecución cada una para completarse, obligando al mismo tiempo que sus unidades de control sean mucho más complejas y por tanto que ocupen más espacio en el dado de silicio.

Un ejemplo interesante de este enfoque es el hecho de que los procesadores Intel de la época al tener una unidad de control muy compleja no disponían de espacio suficiente para añadir una memoria cache en el propio chip [9].

En cambio los diseñadores del MIPS R2000 se percataron de que gran parte de los programas no hacían uso de las instrucciones complejas de las que disponían sus competidores, que al mismo tiempo obligaba a tener ciclos de reloj más largos, reduciendo la producción del procesador y además encarecía el producto para un usuario final que puede no obtener ningún beneficio por esta complejidad añadida.

Los creadores del MIPS decidieron utilizar únicamente instrucciones simples permitiendo de este modo reducir la complejidad de la unidad de control, disminuyendo el espacio ocupado por ella sobre el dado. Este último hecho es fundamental para el éxito de la arquitectura RISC, dado que al tener espacio sobrante en el dado de silicio tuvieron la oportunidad de añadir una memoria cache directamente en el procesador, entre otros beneficios.

Junto a todo esto el precio final del producto era más bajo que el de sus rivales. Se puede incidcar que hay aplicaciones para las que el MIPS R2000 no es apto, pero sus creadores se centraron en ofrecer un producto con unas características específicas a un público específico que sí se podía beneficiar de estas.

La versión final del MIPS R2000, la antes citada versión segmentada, tenía muchas características pioneras para la época. Una de ellas es, como se ha mencionado anteriormente, la inclusión de memoria cache en el propio procesador. Muchas de las otras características del procesador fueron posibles gracias a esta memoria cache.

La conocida memoria cache es una memoria de velocidad media alta, más lenta que los registros del propio procesador, pero mucho más rápida que la memoria principal del ordenador. Por tanto, su función es hacer de buffer entre el procesador y la memoria principal del equipo. Además podía estar separada en niveles, cada nivel más lento que el anterior pero de mayor tamaño que su antecesor.

Muy importante fue el planificador que estaba detrás de la memoria cache. Un planificador que ejecutaba una política de planificación, que dependiendo del tipo de esta, decide qué datos buscar en la memoria principal para posteriormente traerlos a la memoria cache, qué datos mantener en esta última y por último qué datos descender de la memoria cache de vuelta a la memoria principal. El empleo de un buen planificador y una política de planificación adecuada posibilita un uso eficiente de la memoria cache, acelerando la ejecución de las instrucciones que acceden a memoria y por consiguiente reduciendo el tiempo durante el cual el procesador está ocioso esperando la llegada de datos desde memoria, o la escritura de estas en memoria. De esta forma se consigue aumentar la productividad del procesador por unidad de tiempo, es decir, las instrucciones ejecutadas en un tiempo dado.

Otro aspecto importante que es decisivo para reducir el retardo ocasionado por los accesos a memoria es disponer de un compilador bien construido que optimice el código para juntar en un mismo bloque de memoria o bloques cercanos, datos que se utilizaran en un mismo bloque de código, permitiendo así reducir

los accesos a memoria, dado que de esta forma con un solo acceso a memoria o unos pocos accesos se obtendrán todos los datos necesarios para la ejecución de un bloque de código del programa.

El procesador MIPS R2000 segmentado, como su nombre indica dispone de una ruta de datos segmentada. Esto quiere decir que una instrucción no se ejecuta como un bloque aislado que ocupa todo el procesador, justo lo contrario. Una instrucción se ve como un conjunto de etapas. Teniendo en cuenta las funciones que un procesador lleva a cabo en cada instrucción, que hemos mencionado anteriormente durante este capítulo, podemos fácilmente encontrar varias etapas aisladas que existen durante la ejecución de una instrucción. Estas etapas básicas son [8]:

- Lectura de la instrucción desde memoria
- Decodificación de la instrucción en la unidad de control
- Ejecución de la instrucción
- Acceso a memoria
- Escritura en registros

De estas etapas se pueden derivar más, aunque para la finalidad de este texto, tomaremos que el MIPS R2000 segmentado reconoce estas cinco etapas separadas.

La versión monociclo del procesador MIPS R2000 ejecuta las instrucciones una a una, las ingiere una tras otra, al terminar completamente la ejecución de una instrucción comienza con la siguiente. En cambio, en el procesador segmentado al mismo tiempo en su ruta de datos puede haber varias instrucciones a la vez, cada una en una etapa separada. Cuando la primera instrucción termine de ejecutar una etapa y pasa a la siguiente, la siguiente instrucción pasa a la etapa que estaba la primera. La ruta de datos segmentada se puede observar en la Figura 3.3.

Para que al mismo tiempo en la ruta de datos pueda haber varias instrucciones se han introducido unos registros intermedios, que almacenan la instrucción y las señales de control que interesan.

Al ser las instrucciones del conjunto de instrucciones del MIPS R2000 todas del mismo tamaño, estas tienen 32 bits, se simplifica enormemente el segmentado del procesador. Al tener todas el mismo ancho de bits y ser todas instrucciones simples, el número de etapas se ha reducido al mínimo y por tanto la complejidad añadida por el segmentado es menor aunque los beneficios sean enormes. El segmentado de un procesador con arquitectura x86, en comparación, es mucho más complejo, ya que este tipo de procesadores tienen instrucciones de tamaño variable y por cual, precisan de más etapas por este mismo hecho [8].

Aunque la introducción de una ruta de datos segmentada aumentó la producción del procesador no vino sin problemas. Uno de los mayores problemas es la dependencia de datos entre una instrucción y otras posteriores. Es decir, una instrucción entra en la ruta pero los datos que necesitará todavía no están listos, debido a que una instrucción anterior que debe producir el dato necesario no ha terminado de ejecutarse aún.

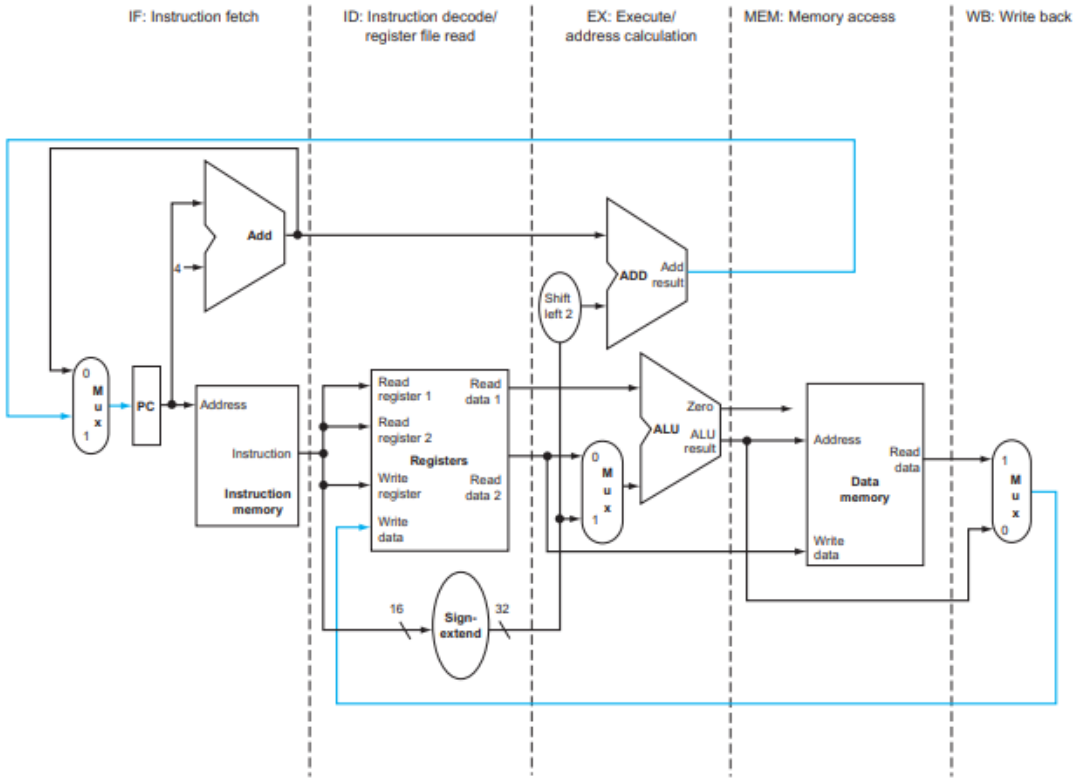

Figura 3.3: Ruta de datos segmentada del procesador MIPS R2000 donde se pueden apreciar las distintas etapas con sus nombres en inglés. Cabe destacar que en la figura no aparecen los registros intermedios que se encontrarían entre una etapa y la siguiente.
Figura obtenida del libro de *Computer Organization and Design: the hardware/software interface* de David A. Patterson y John L. Hennessy [8].

Para solucionar esta situación problemática existen varios caminos, no excluyentes unos a otros. El primero y más sencillo es hacer que la instrucción dependiente espere a que la instrucción que produce el dato que necesita termine. Una solución más avanzada es tener conexiones existentes entre la etapa de Decodificación de la instrucción y la etapa de Ejecución de la instrucción. De tal manera que nada más la instrucción productora de datos termine de ejecutarse en la etapa EX estos datos estén disponibles para la instrucción dependiente, sin necesidad de esperar que la instrucción anterior termine de recorrer completamente la ruta de datos.

Otra posible solución es utilizar un planificador para las instrucciones que detecte estas posibles situaciones de dependencia y analice si mediante una posible reordenación de las instrucciones próximas seria posible evitar esta situación de dependencia problemática.

Mediante el uso conjunto de estas tres técnicas, reordenar las instrucciones directamente en tiempo de ejecución, cortocircuitar las etapas necesarias para acelerar la obtención de los datos y por último si no hay más alternativa hacer que la instrucción dependiente espere a tener los datos listos, se ha conseguido solucionar los problemas de una ruta de datos segmentada manteniendo al mismo tiempo el aumento de producción que se consigue con esta ruta.

Conseguir estas soluciones viene con un coste, un aumento en la complejidad de la unidad de control, que debe ser capaz en primer lugar de detectar las dependencias y tras ello determinar la solución o soluciones más adecuadas en cada caso.

3.4 Conjunto de instrucciones del MIPS R2000

El conjunto de instrucciones de un procesador se puede entender como el lenguaje que este entiende o utiliza. Para que uno trabaje con una CPU necesita hablar su idioma o de otra forma utilizar un traductor, en este caso llamado compilador, que traduzca de un lenguaje externo al lenguaje hablado por el procesador [8].

El procesador directamente trabaja con conjuntos de bits únicamente. Aunque esto es cierto existe un lenguaje más cercano al humano, llamado lenguaje ensamblador o conjunto de instrucciones, donde cada instrucción se corresponde a un conjunto de bits que el procesador entiende [8].

Cuando se escribe un programa en un lenguaje de más alto nivel, como sería por ejemplo, el lenguaje de programación C. Este programa en C pasa por un compilador que traduce el programa desde C a instrucciones pertenecientes al lenguaje ensamblador del procesador en cuestión y finalmente este programa ensamblador pasa por otro compilador que lo traduce a lenguaje máquina, es decir, a ceros y unos.

Cada arquitectura de procesadores suele venir con su propio conjunto de instrucciones. Estos lenguajes pueden ser muy diferentes pero en general, suelen tener muchos puntos en común. Estas similitudes se deben a que los procesadores se construyen utilizando los mismos principios y todos comparten una serie de funciones que es necesario que todos los procesadores sean capaces de llevar a cabo [8].

Aunque estas similitudes sean ciertas podemos diferenciar claramente entre dos grupos de conjuntos de instrucciones. Unas más complejas, las llamadas CIS del inglés *Complex instruction set*, conjunto de instrucciones complejo. Los otros conjuntos de instrucciones, más simples o como son conocidos, reducidos, en inglés *Reduced Instruction Set*.

Como su nombre indica, los conjuntos de instrucciones complejos tienen un mayor número de instrucciones, además estas a su vez son más complejas. Es decir, con una instrucción compleja se hacen más acciones dentro del procesador. Los conjuntos de instrucciones complejos tienen instrucciones más especializadas para una tarea en concreto, por ejemplo, el conjunto de instrucciones x86 de Intel dispone de instrucciones preparadas para trabajar con vectores. Estas instrucciones permiten ejecutar la misma operación sobre varios datos mediante una sola instrucción.

Los conjuntos de instrucciones reducidos, como es el utilizado por el MIPS R2000 disponen de un menor número de instrucciones, estas suelen ser todas simples, es decir, se corresponden la mayoría con operaciones elementales dentro del procesador. Las instrucciones simples son a su vez más rápidas de ejecutar.

Además en un conjunto de instrucciones simple todas las instrucciones suelen tener el mismo tamaño.

El lenguaje ensamblador del MIPS R2000 se compone de varias categorías de instrucciones. Dispone de instrucciones de carga, aritméticas, de comparación, almacenamiento, lógicas, movimiento entre registros, desplazamiento, saltos condicionales y por último, saltos incondicionales.

Todas las instrucciones de una categoría tienen el mismo formato. Por ejemplo, todas las instrucciones aritméticas realizan una sola operación y utilizan tres variables, donde estas variables pueden ser tres registros o dos registros y un número inmediato. Un registro siempre será el destino donde se guardará el resultado de la operación, los otros dos registros o registro e inmediato serán los valores sobre los que se ejecutará la operación [8].

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Figura 3.4: Parte del conjunto de instrucciones del procesador MIPS R2000, donde podemos observar instrucciones aritméticas, lógicas, de transferencia de datos, es decir, de carga y almacenamiento, de salto condicional y de salto incondicional.
Figura obtenida del libro de *Computer Organization and Design: the hardware/software interface* de David A. Patterson y John L. Hennessy [8].

En la Figura 3.4 podemos observar parte del lenguaje ensamblador del MIPS R2000, fijándonos en cada categoría se percibe claramente como las instrucciones de cada una de estas tienen el mismo formato.

La totalidad de las instrucciones pertenecientes al lenguaje ensamblador del MIPS R2000 se pueden observar en el apéndice A.

Una gran diferencia entre un conjunto de instrucciones complejos y un conjunto de instrucciones reducido reside en los distintos tipos de direccionamiento que soporta. Los modos de direccionamiento tienen como objetivo especificar el lugar donde se encuentra el operando que se utilizará en la ejecución de la instrucción. Dependiendo de los modos de direccionamientos soportados estos permiten localizar un operando ubicado en en un registro, en la propia instrucción, es decir, un inmediato, o en memoria.

Los modos de direccionamiento soportados por el MIPS R2000 son:

- Direccionamiento a registro
- Direccionamiento inmediato
- Direccionamiento indexado

Los conjuntos de instrucciones complejos soportan mayor número de modos de direccionamiento porque se persigue permitir la ejecución eficiente de programas escritos en lenguajes de alto nivel [9]. Por ejemplo, muchos conjuntos de instrucciones complejos soportan direccionamiento indexado por factor de escala y desplazamiento, donde este tipo de desplazamiento es útil si se trabaja con *arrays*.

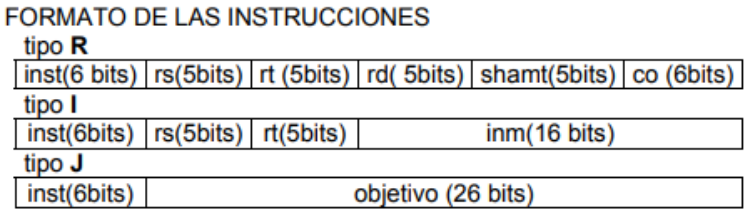

Figura 3.5: Los tres formatos de las instrucciones del MIPS R2000, uno por tipo de instrucción. Además de los diferentes campos aparece el número de bits de cada uno de estos.

En la Figura 3.5 se pueden observar los tres tipos de instrucciones que soporta el MIPS R2000. Estos son:

- Instrucciones tipo R, este tipo engloba todas las instrucciones aritmético-lógicas sin inmediato.
- Instrucciones tipo I, este conjunto de instrucciones abarca las instrucciones aritmético-lógicas con inmediato y las instrucciones de carga y almacenamiento, junto con las instrucciones de salto condicional.
- Instrucciones tipo J, comprende las instrucciones de salto incondicional.

En la tabla anterior se mencionan las categorías de instrucciones que abarca cada tipo, pero existen algunas excepciones, por ejemplo, la instrucción de salto

incondicional a registro, es de tipo R aunque el resto de instrucciones de salto incondicional son de tipo J.

Además de las instrucciones soportadas directamente por la unidad de control existen otras, llamadas pseudoinstrucciones, son varias instrucciones que se ejecutan una tras otra como si se tratase de una instrucción más compleja. Se han creado principalmente para simplificar la ejecución de una acción más compleja pero muy utilizada. Estas pseudoinstrucciones están marcadas en el apéndice A como instrucciones de tipo PS.

Para finalizar el capítulo actual hay que destacar de nuevo que el procesador de este proyecto, el MIPS R2000 es un procesador RISC con ruta de datos monociclo. Como un procesador RISC, al contrario que los procesadores CISC, tiene un conjunto de instrucciones reducido con instrucciones simples y todas con tamaño uniforme. En la versión monociclo de procesador, todas las instrucciones empiezan a ejecutarse y terminan en el mismo ciclo.

En conclusión, el procesador MIPS R2000 monociclo es un procesador relativamente simple aunque con toda la funcionalidad y características de un procesador real, muy apropiado para la educación, para introducir a los estudiantes en el diseño de un procesador.

CAPÍTULO 4
Diseño e implementación del MIPS R2000 en CircuitVerse

En el presente capítulo se expone el diseño y la implementación del MIPS R2000 en su versión monociclo mediante el simulador CircuitVerse. Se muestran los detalles de implementación tanto de la ruta de datos como de los distintos componentes individuales.

4.1 Diseño interno de los componentes que forman la ruta de datos

La ruta de datos del procesador está formada por distintos tipos de circuitos, tanto componentes más simples como multiplexores y registros, como elementos más complejos.

Para el diseño de estos elementos más complejos se han utilizado todas las herramientas que proporciona CircuitVerse para facilitar el trabajo. Para empezar, se han diseñado e implementado como circuitos separados en entornos de trabajo, o bancos de trabajo, separados para cada componente. De esta forma se consigue simplificar la tarea de diseñar un componente, dado que cada entorno de trabajo está totalmente aislado del resto. Este último hecho permite probar los elementos antes de introducirlos en la ruta de datos.

Antes de terminar la implementación de un componente CircuitVerse permite personalizar su diseño exterior, es decir, la manera en que se verá este componente cuando se introduce como subcircuito.

Una vez terminada la implementación de un componente este se introduce en la ruta de datos como un subcircuito.

4.1.1. Banco de Registros

El primer circuito diseñado es el banco de registros (ver Figura 4.1). Igual que en el MIPS R2000 auténtico el banco de registros tiene 32 registros de 32 bits cada uno, pudiendo ser leídos dos registros en cada ciclo de reloj y escrito uno.

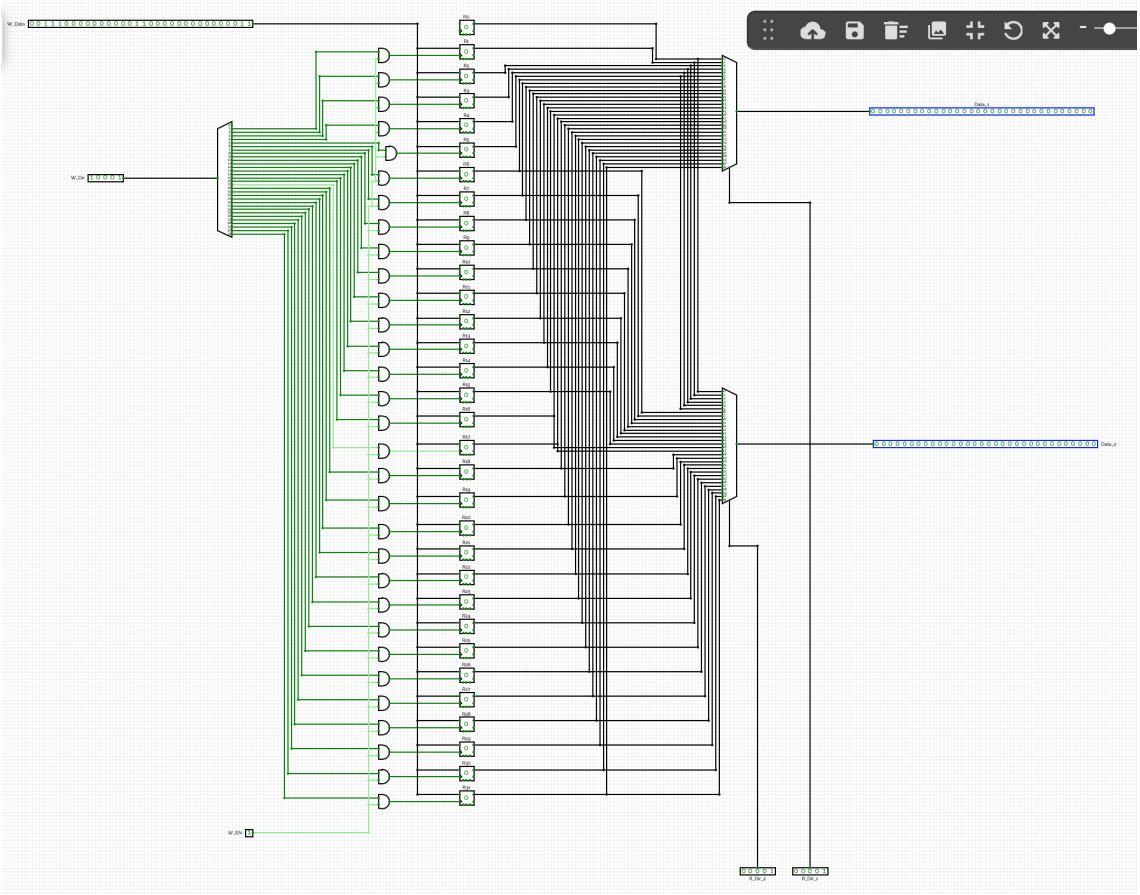

Figura 4.1: Vista completa del diseño del banco de registros del MIPS R2000 implementado en CircuitVerse. Se pueden observar sus entradas como rectángulos de color negro y las salidas como rectángulos de color azul.

Para los registros se han utilizado biestables de tipo D. CircuitVerse permite especificar los bits de cada biestable[1], por tanto, cada biestable de tipo D es de 32 bits de ancho, simplificando el diseño del banco de registros enormemente.

En el diseño actual se pueden observar claramente dos partes diferenciadas, estando en medio los propios registros. La parte que está a la izquierda de los registros se encarga de la escritura en los registros y la parte a la derecha de los registros se encarga de la lectura de los registros.

La parte encargada de la escritura en registro, véase la Figura 4.2, permite almacenar un dato de 32 bits en cualquiera de los 32 registros menos en el registro 0, el cual igual que en el diseño original siempre almacena un 0.

Para la operación de escritura en primer lugar se selecciona el registro en el cual se almacenará el dato mediante una entrada de cinco bits llamada *W_Dir*. Esta dirección llega a un decodificador que activa una de las 32 líneas según el valor que ha recibido. El bit activado por el decodificador llega a la puerta AND correspondiente aunque esta no se activará hasta que no se active además el bit de habilitación de escritura llamado *W_EN*. Antes de la activación del bit *W_EN*

[1] Aunque, por definición, un biestable tiene capacidad de un bit, CircuitVerse permite parametrizar el número de bits del elemento de memoria, por lo que cuando el número de bits es mayor que uno en realidad se trata de un registro.

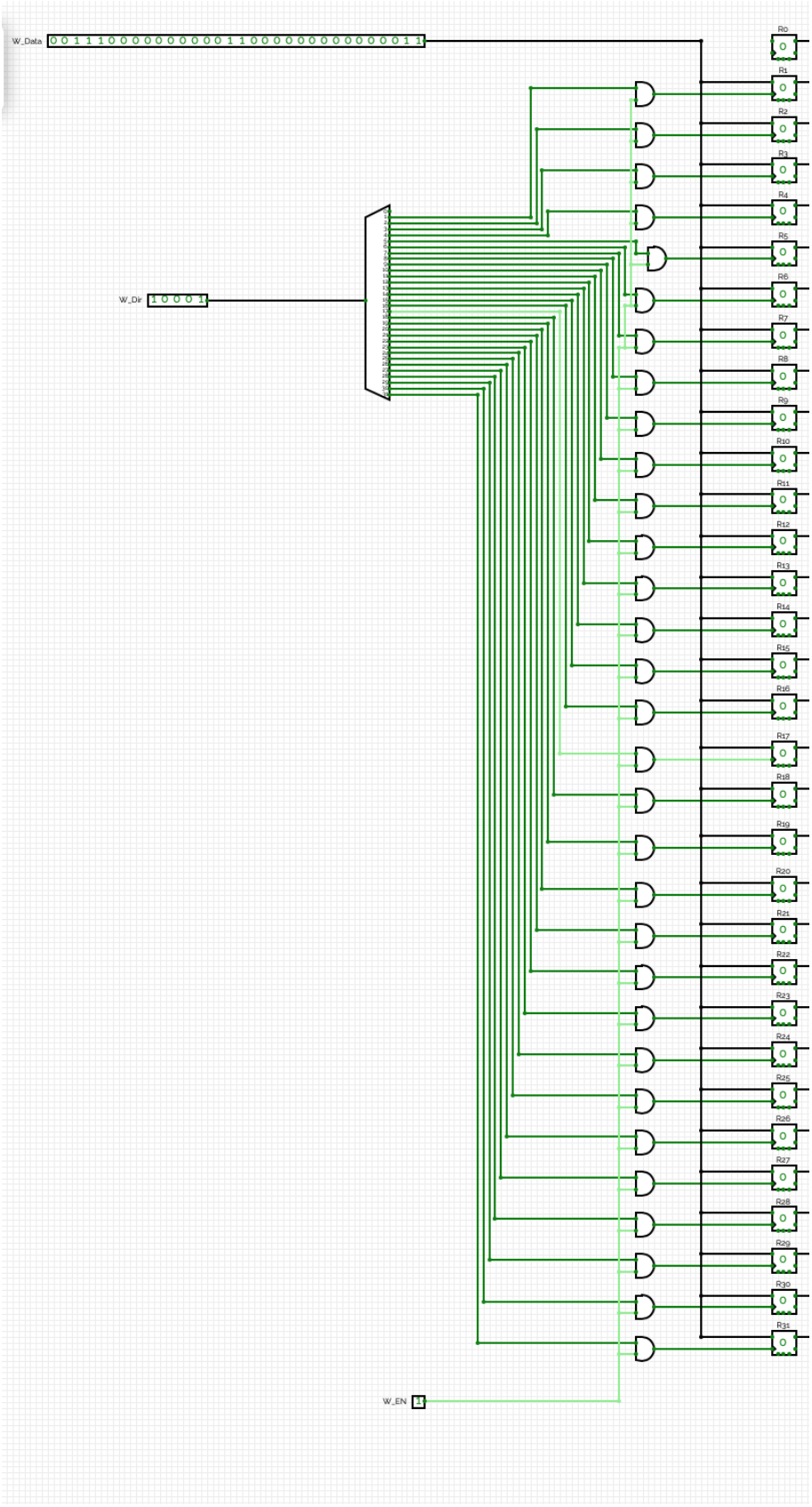

Figura 4.2: Parte del banco de registros encargada de la escritura en registro. Se observan tres entradas, por un lado una entrada para seleccionar el registro, por otro lado un bit de habilitación, *Write Enable*, y por último una entrada de 32 bits para el dato que se almacenará en registro.

a la entrada de todos los registros ha llegado el dato a almacenar proveniente de la entrada llamada *W_Data*, aunque en ninguno se ha almacenado este todavía dado que no se ha activado para ninguno su entrada de reloj que en este diseño se utilizará para habilitar la escritura. Teniendo la dirección del registro y el dato listo únicamente falta recibir la señal de habilitación de la escritura proveniente de la unidad de control, que llegará a todas las puertas AND, activando de esta manera la puerta AND que también recibe la señal del decodificador y por último mediante la activación de la puerta AND correspondiente le llega la señal de habilitación a la entrada de reloj del registro pertinente activando la escritura y almacenando el dato.

Teniendo en cuenta que los registros están formados por biestables de tipo D la operación de escritura será retardada permitiendo de esta manera la lectura y escritura de un mismo registro en un único ciclo de reloj.

La parte encargada de la lectura del valor almacenado en los registros permite leer al mismo tiempo dos registros independientes, véase la Figura 4.3. Para ello el circuito dispone de dos entradas de 5 bits cada una que indicarán los dos registros que se desean leer. Estas entradas están conectadas a las entradas de control de dos multiplexores, uno por registro leído. Estos multiplexores disponen de 32 líneas o buses cada uno de los cuales está conectado a la salida de un registro y por los cuales llegan a los multiplexores los valores almacenados en los registros. Dependiendo de los bits de control que recibirá desde la entrada a la que está conectado el multiplexor seleccionará uno de los buses provenientes y pasará el valor del registro correspondiente a la salida a la que está conectado dicho multiplexor.

4.1.2. Unidad extensora

El próximo circuito a considerar es la unidad extensora del bit de signo, utilizada cuando la palabra de bits se interpreta como un número entero con signo. Como su nombre indica este circuito extiende un número de 16 bits, en este caso, a un número de 32 bits. Esto se consigue extendiendo el bit de signo del número de 16 bits a los nuevos 16 bits, hasta alcanzar los 32 bits de ancho de palabra.

Para la implementación de la unidad extensora en CircuitVerse se ha utilizado una herramienta o componente llamado *Splitter*. Su función es muy sencilla aunque al mismo tiempo muy útil; permite separar un bus de varios bits de ancho en bits separados o la operación inversa, unir varios bits en un solo bus. Es interesante mencionar que este componente además de permitir separar un bus en bits separados también posibilita separar un bus en grupos de varios bits; por ejemplo, si existiera un bus de 32 bits de ancho, este se podría separar, por ejemplo, en un bus de 16 bits y dos buses de 8 bits cada uno.

Este último comportamiento ha sido muy conveniente para la implementación de la unidad extensora del bit de signo. En la Figura 4.4 se puede observar la implementación de este componente en CircuitVerse. Se contempla cómo se ha utilizado un *splitter* de 16 bits que separa el número de 16 bits de la entrada en dos grupos de bits, un grupo con los primeros 15 bits y por otro lado el bit 15, el bit de signo. Seguidamente se ha utilizado otro *splitter*, esta vez de 32 bits, donde los primeros 15 bits del número de 32 bits están conectados al grupo de los primeros

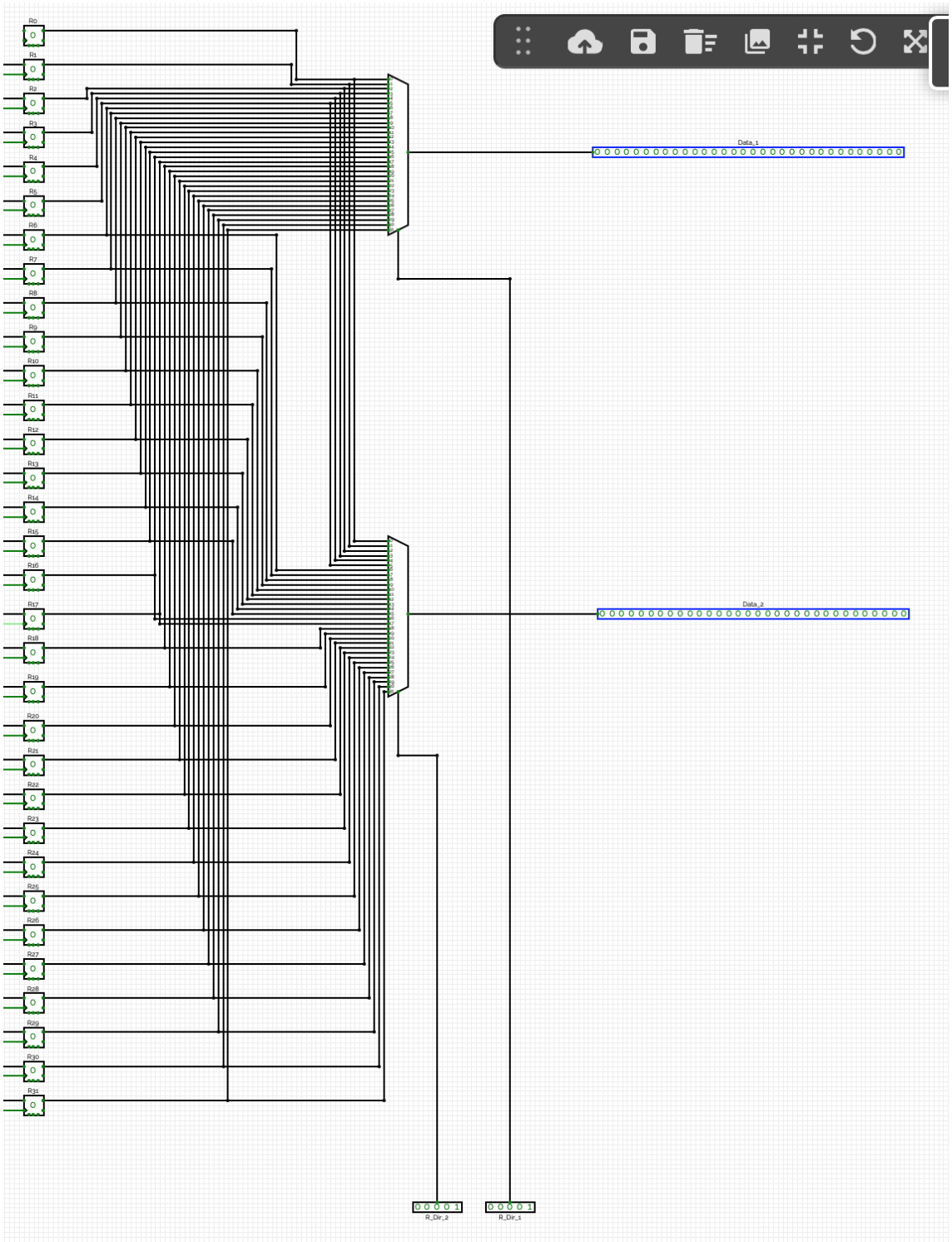

Figura 4.3: Parte del banco de registros encargada de la lectura desde registros. Se observan dos entradas correspondientes cada una a la dirección de un registros que se leerá. Además se contemplan dos salidas correspondientes a los datos provenientes de los dos registros leídos.

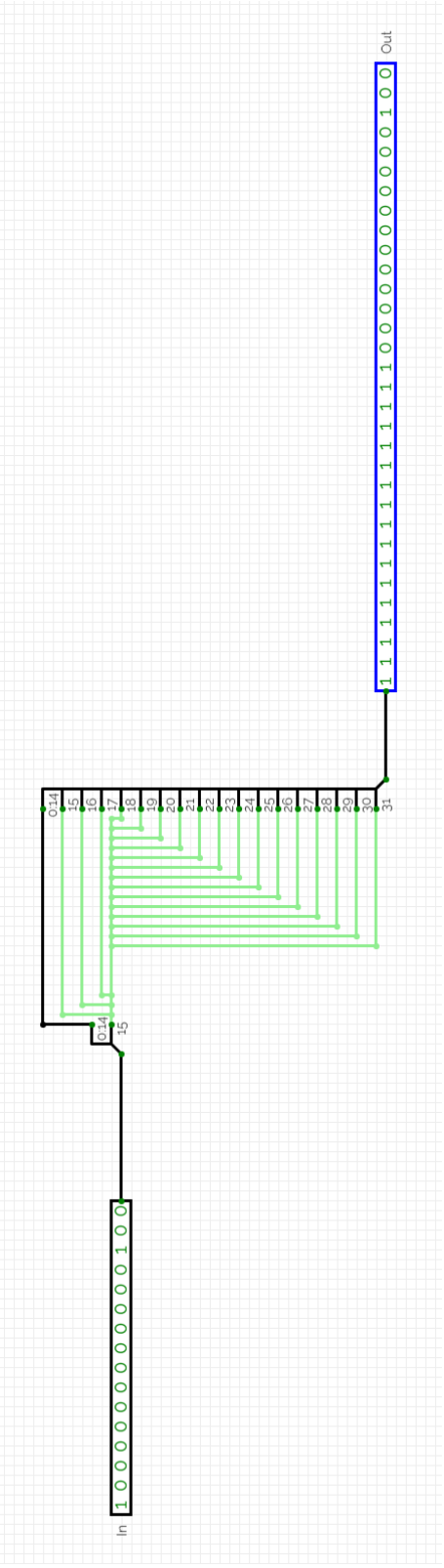

Figura 4.4: Implementación en CircuitVerse de la unidad extensora del MIPS R2000. Se puede observar que la entrada, rectángulo negro, tiene 16 bits y la salida, rectángulo azul, tiene 32 bits de ancho.

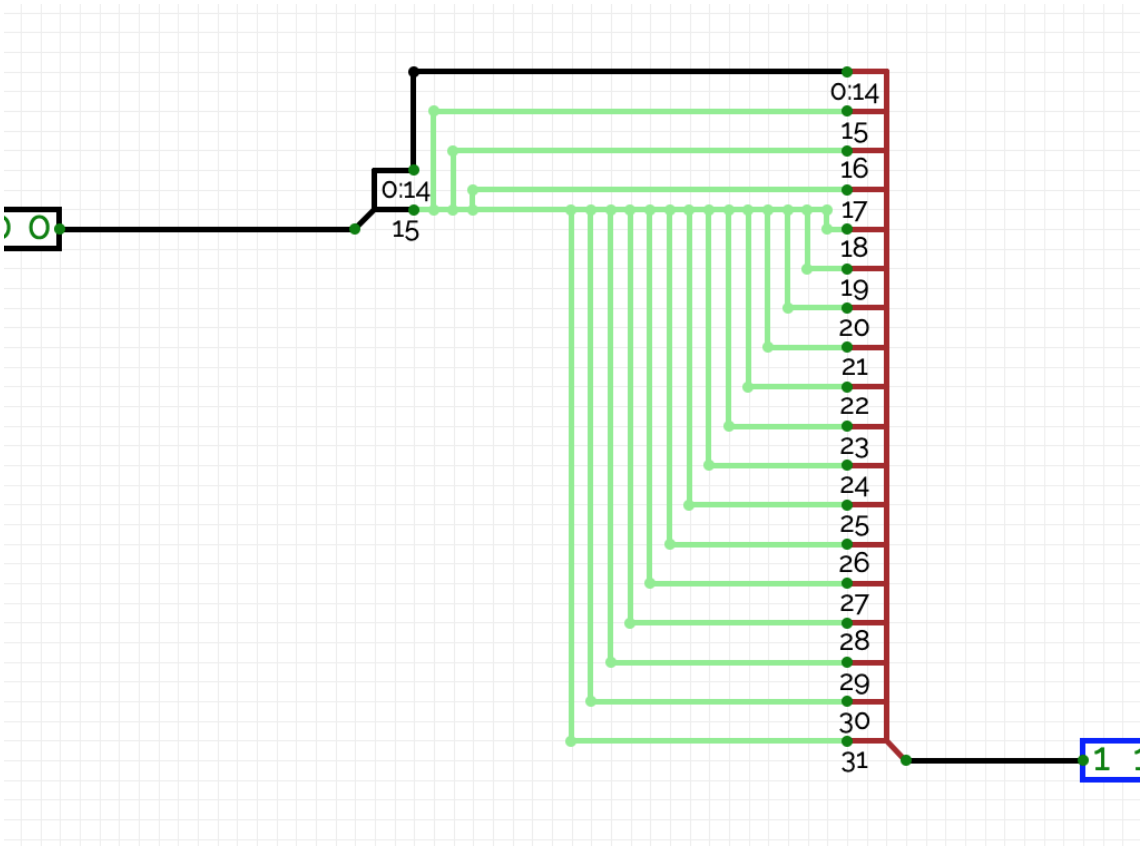

Figura 4.5: Interconexión interna de la unidad extensora del bit de signo. A la izquierda de la figura se encuentra la entrada, el número de 16 bits, a la derecha la salida, el número ya extendido a 32 bits.

15 bits del número de 16 bits. En cambio, los otros 17 bits de mayor peso del número de 32 bits todos están conectados al bit de signo del número sin extender, es decir, al bit 15. En la Figura 4.5 se puede observar mejor la interconexión indicada anteriormente.

4.1.3. Sumador del Contador de Programa

Para pasar de una instrucción a la siguiente y leer esta desde la memoria de instrucciones el contador de programa dispone de un sumador que en cada ciclo de reloj suma 4 al valor actual. El contador de programa es un registro que almacena la dirección de memoria de la instrucción actualmente en ejecución, una vez esta instrucción se lee desde memoria y empieza a ejecutarse, de manera paralela se actualiza el contenido del contador de programa sumando 4 al valor actual, de esta forma el *Program Counter* pasa a apuntar a la siguiente instrucción a ejecutar.

La funcionalidad del sumador es muy sencilla y su implementación en CircuitVerse es todavía más sencilla, véase la Figura 4.6. Se ha utilizado una unidad aritmético-lógica que ya está disponible entre los componentes ofrecidos por CircuitVerse, esta ALU se ha configurado para la operación suma mediante el uso de una constante que almacena el código de la operación suma. A las entradas de la ALU por un lado se ha conectado una constante de 32 bits con valor 4 y mediante

la otra entrada de la ALU esta recibirá el valor actual del contador de programa, que finalmente sumará y lo enviará a la salida.

4.1.4. Unidad Aritmético-lógica

CircuitVerse ofrece una unidad aritmético-lógica entre los componentes disponibles, aunque esta ALU tiene unas propiedades muy adecuadas para este proyecto faltaba una funcionalidad clave. Esta funcionalidad es el bit Z, es decir, indicador de resultado que se activa cuando el valor calculado es un 0. Que la ALU disponga de esta propiedad es un requisito para las instrucciones de salto condicional que se basan en ella. Debido a esto se ha tenido que diseñar una ALU nueva sobre la unidad aritmético-lógica disponible en CircuitVerse, que ofrezca esta funcionalidad adicional.

Para implementar esta nueva unidad aritmético-lógica se ha utilizado de base la ofrecida por CircuitVerse. La ALU implementada se puede observar en la Figura 4.7. Para obtener la nueva funcionalidad y determinar si el valor calculado es 0, se ha utilizado de nuevo un *splitter* que separa la salida obtenida desde la ALU en 32 bits sueltos.

Teniendo los bits independientes todos se conectan a unas puertas NOT y la salida de cada puerta NOT se conecta a una de las entradas de una puerta AND. Hay que mencionar que el diseño se podría simplificar si se utilizara una única puerta AND con 32 entradas, pero actualmente en CircuitVerse las puertas AND soportan únicamente 10 entradas. Las salidas de estas puertas AND se han conectado a una última puerta AND junto con los dos bits que faltaban, la salida de esta puerta será el bit Z.

Teniendo clara la implementación la funcionalidad es muy sencilla de entender, se niegan todos los bits para obtener un 1 si el bit es un 0, por tanto, la salida de una puerta AND será 1 únicamente si todas sus entradas son 1, esto es, si todos los bits conectados a sus entradas realmente son 0. La salida de la última puerta AND está conectada a la salida del circuito, llamada Z. Por tanto, si los bits del valor obtenido de la ALU son todos cero, la última puerta AND proporcionará un 1 en su salida y por tanto obtendremos directamente un 1 en la salida Z del circuito, en cambio, si algún bit del valor obtenido de la ALU es 1 la salida de la puerta AND será un 0 y por tanto la salida Z también.

4.1.5. Unidad de decodificación de función aritmética

Para simplificar el diseño de la unidad de control y abstraer funciones se ha tomado la decisión de crear un circuito independiente para identificar la función aritmético-lógica a ejecutar. Este circuito, a continuación, se introducirá en la unidad de control como un subcircuito, empleando al máximo las posibilidades que ofrece CircuitVerse para simplificar la implementación de circuitos.

La funcionalidad deseada para este circuito es sencilla, teniendo la instrucción como entrada, comprobar los 6 bits de menor peso e identificar la operación aritmético-lógica requerida, y ofrecer como salida el código de operación de la ALU para dicha operación aritmético-lógica.

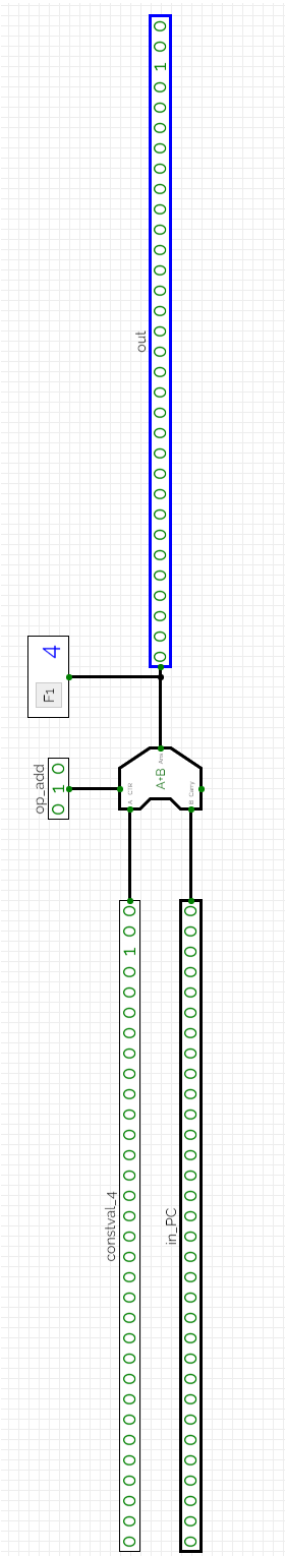

Figura 4.6: Implementación en CircuitVerse del sumador utilizado por el contador de programa para pasar a apuntar a la siguiente instrucción a leer. A la salida tiene conectado un *flag* utilizado para pruebas con la idea de mostrar el valor de la salida en decimal.

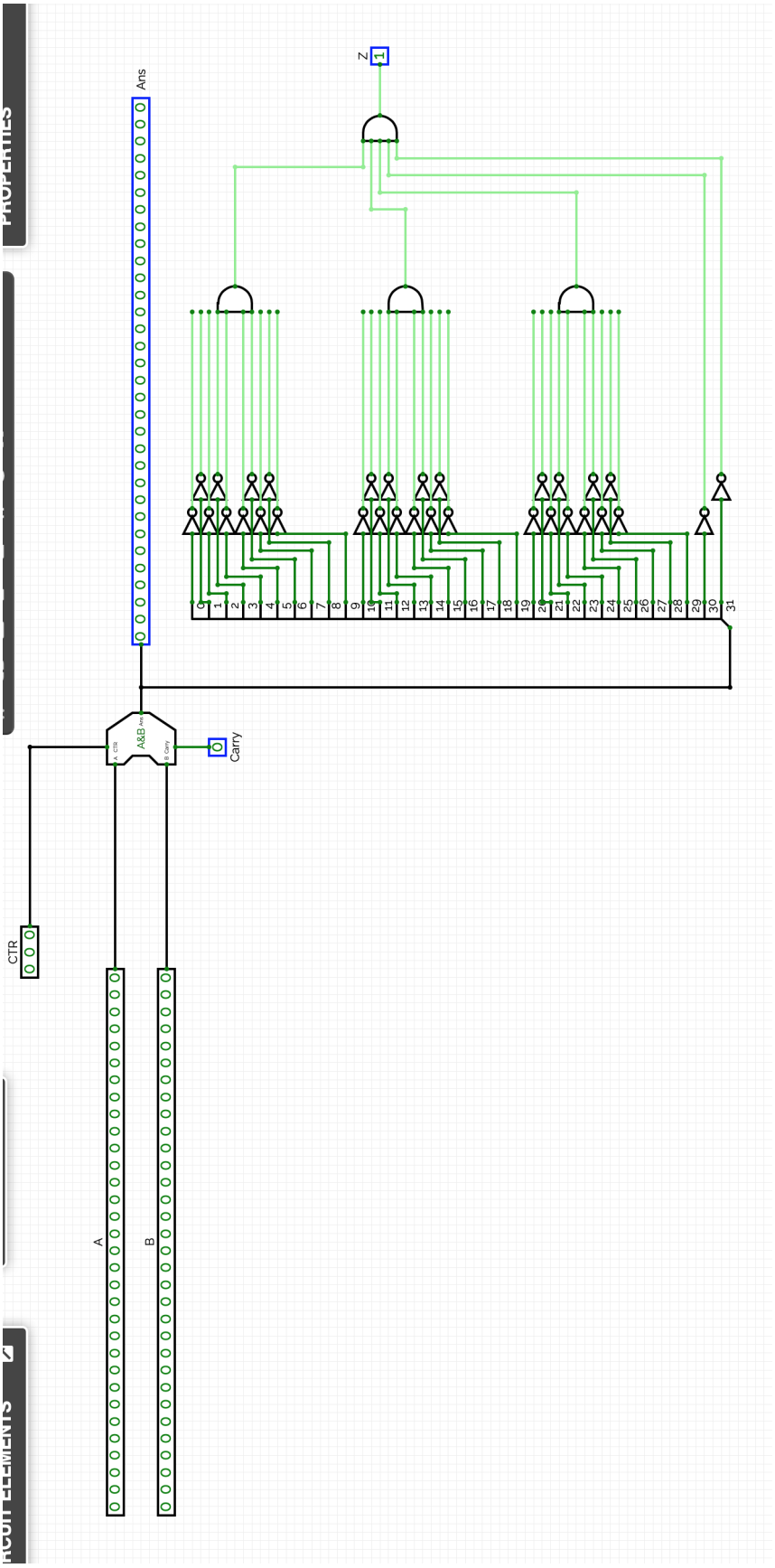

Figura 4.7: Implementación en CircuitVerse de la unidad aritmético-lógica. Para la nueva ALU se han mantenido todas las entradas y salidas que ofrece la unidad aritmético-lógica, únicamente se ha añadido una nueva salida, el así llamado bit Z.

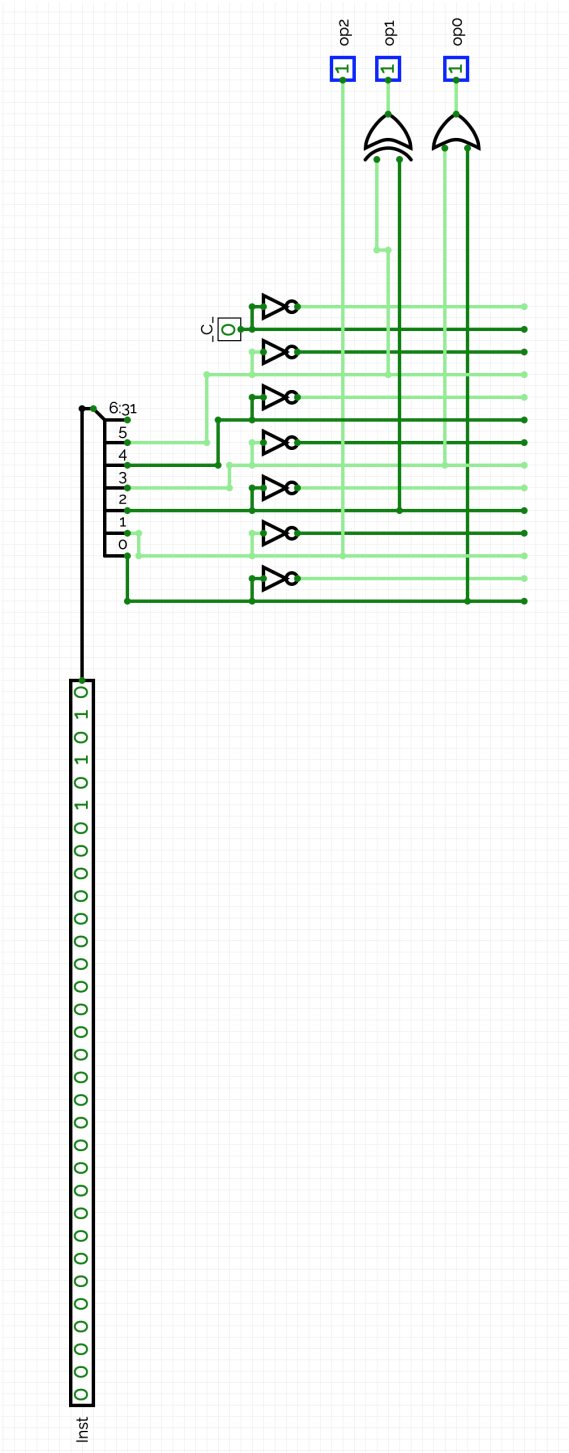

Figura 4.8: Implementación en CircuitVerse de la unidad de decodificación de función aritmética. Como entrada el circuito tiene la instrucción completa y como salidas ofrece los tres bits del código de operación de la ALU.

La implementación del circuito actual fue sencilla gracias a que para ello se ha aprovechado la posibilidad que ofrece CircuitVerse para crear circuitos utilizando una tabla de verdad. Se han introducido las entradas de la tabla de verdad y a continuación las salidas, posteriormente se ha establecido la relación de entradas y salidas para cada entrada de la tabla y el propio simulador a implementado un circuito que cumple con la tabla de verdad establecida.

Teniendo el circuito base ya creado mediante la tabla de verdad, únicamente se ha modificado ligeramente este, introduciendo como entrada del circuito la instrucción completa y posteriormente utilizando un *splitter* se han aislado y separado los bits que interesan para finalmente conectar estos con las entradas del circuito creado por las tablas de verdad. Pudiendo observar el circuito finalizado en la Figura 4.8.

4.1.6. Unidad de cálculo de la dirección de salto

Para las instrucciones de salto condicional anteriormente se ha implementado una ALU con bit Z, a continuación, se expondrá el diseño e implementación de la unidad que calcula la dirección a la que apunta la instrucción.

El cálculo de la dirección de salto ocurre en varios pasos. Como se ha mencionado anteriormente durante este trabajo, las instrucciones de salto condicional tienen reservados 16 bits para indicar la dirección de salto. Este valor de 16 bits en primer lugar se extiende a 32 bits utilizando la unidad de extensión del bit de signo mencionada con anterioridad. Posteriormente este valor llega como entrada al circuito actual y el siguiente paso ya llevado a cabo por este circuito es multiplicar por 4 el valor recibido en su entrada. El siguiente y último paso es sumar este valor al valor contenido en el registro del contador de programa.

En el circuito implementado en CircuitVerse, véase la Figura 4.9, una vez llega de la unidad extensora del bit de signo el valor de 32 bits, el circuito actual en primer lugar utilizando un *splitter* separa los 32 bits del valor de la entrada y todos los desplaza dos bits a la izquierda, introduciendo dos ceros en los dos bits de menor peso del valor desplazado, operación equivalente a multiplicar el valor de la entrada por 4. Este valor nuevo llega a la entrada de una ALU donde se sumará al valor contenido en el registro de contador de programa, recibiendo este por una segunda entrada de 32 bits. Teniendo el valor calculado este será la dirección a la que se saltará en caso de que la condición de salto se cumpla.

4.1.7. Unidad para componer la dirección de salto incondicional

Las instrucciones de salto incondicional o tipo J tienen un formato específico, como se ha visto anteriormente, la instrucción tiene dos campos. El campo del código de instrucción, los 6 bits de mayor peso, y el campo etiqueta, los otros 26 bits. El campo etiqueta apunta a la dirección a la que se desea saltar, pero con únicamente 26 bits no es posible acceder a los 4 GB de memoria soportada por el MIPS R2000, por tanto, se utilizaran los 26 bits del campo etiqueta además de varios bits del registro del contador de programa para componer la dirección a la que se saltará.

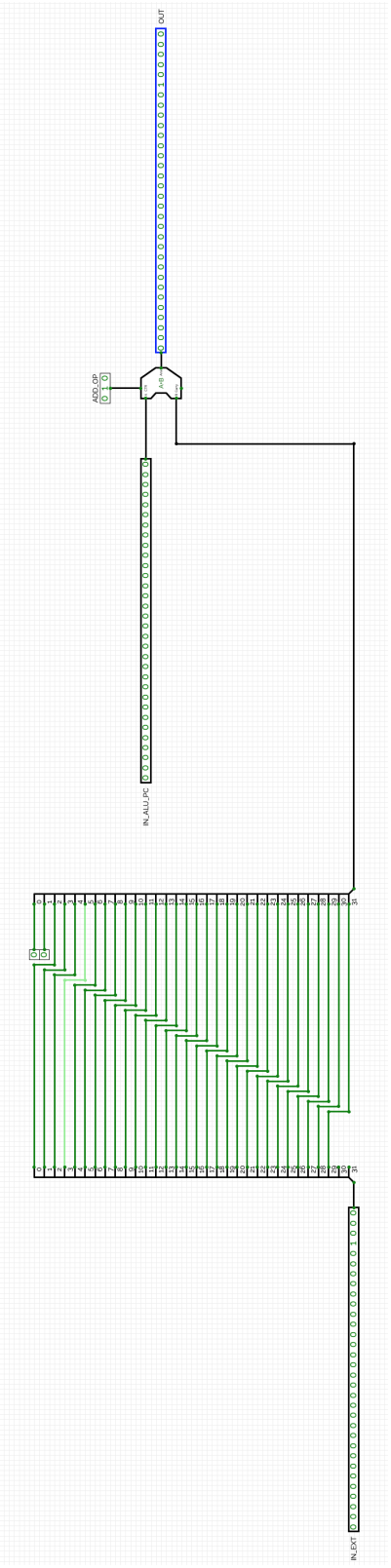

Figura 4.9: Implementación en CircuitVerse de la unidad de cálculo de la dirección de salto. En el circuito se pueden observar dos partes separadas, una que desplaza el valor de la entrada dos bits a la izquierda y la siguiente que suma este valor obtenido al valor proporcionado por el registro del contador de programa.

Este es el propósito de la unidad para componer la dirección de salto incondicional. Al mismo tiempo llevará a cabo varias acciones que conjuntamente compondrán la dirección a la que se va a saltar. En primer lugar es necesario desplazar los 26 bits de la etiqueta 2 bits a la izquierda e introducir dos ceros en los dos bits de menor peso. En segundo lugar en los 4 bits de mayor peso se introducirán los 4 bits de mayor peso del valor almacenado en el registro del contador de programa. Con todo ello se obtiene finalmente la dirección de 32 bits a la que se saltará.

La implementación del circuito actual en CircuitVerse se puede observar en la Figura 4.10. Para implementar este circuito se han utilizado principalmente tres *splitters* donde el primero separa los bits de la entrada correspondiente a la instrucción y utiliza únicamente los primeros 26 bits, el segundo *splitter* tiene la función de unir todas las componentes de la dirección final a la que saltar. Realmente en este *splitter* es donde se concentra toda la funcionalidad del circuito, en primer lugar los 26 bits del campo etiqueta se conectan dos bits desplazados hacia posiciones de mayor peso, en los dos bits de menor peso se introducen ceros y por último, en los cuatro bits de mayor peso del *splitter* se conectan los cuatro bits de mayor peso del tercer *splitter* que provienen del registro del contador de programa. El resultado de unir los bits mencionados anteriormente será la dirección a la que saltar.

4.1.8. Memoria de instrucciones

La memoria de instrucciones es uno de los componentes más importantes de un procesador. Este componente, cabe destacar, realmente está formado por varios circuitos que trabajan conjuntamente, siendo estos la memoria cache de varios niveles y la memoria principal. Es importante mencionar que algunos de estos componentes no se encuentran en el dado de silicio, aunque el nivel L1 de la memoria cache sí se encuentra dentro del propio procesador. El objetivo de la memoria de instrucciones es claro, almacenar las instrucciones que el procesador va a ejecutar.

Para este proyecto se ha decidido diseñar una memoria de instrucciones desde cero sin basarse en algún componente ya disponible en CircuitVerse, como por ejemplo, la memoria ROM existente entre los componentes utilizables, debido a que esta presenta una funcionalidad inadecuada para los propósitos del proyecto.

Se ha optado por diseñar una memoria pequeña de únicamente 32 posiciones de memoria, más que suficiente para los objetivos de este proyecto, aunque aumentar el número de posiciones no presentaría una dificultad extremada. La implementación de la memoria de instrucciones en CircuitVerse se puede apreciar en la Figura 4.11.

En el diseño actual las posiciones de memoria las conforman una serie de valores constantes, llamados en CircuitVerse *CONSTANTVAL*. Cada constante es una posición de memoria en la que se almacenará una instrucción. Al contrario que los biestables de tipo D utilizados para los registros los *CONSTANTVAL* no pierden el valor introducido permitiendo de esta forma crear un programa en código máquina y ejecutarlo las veces que se desee.

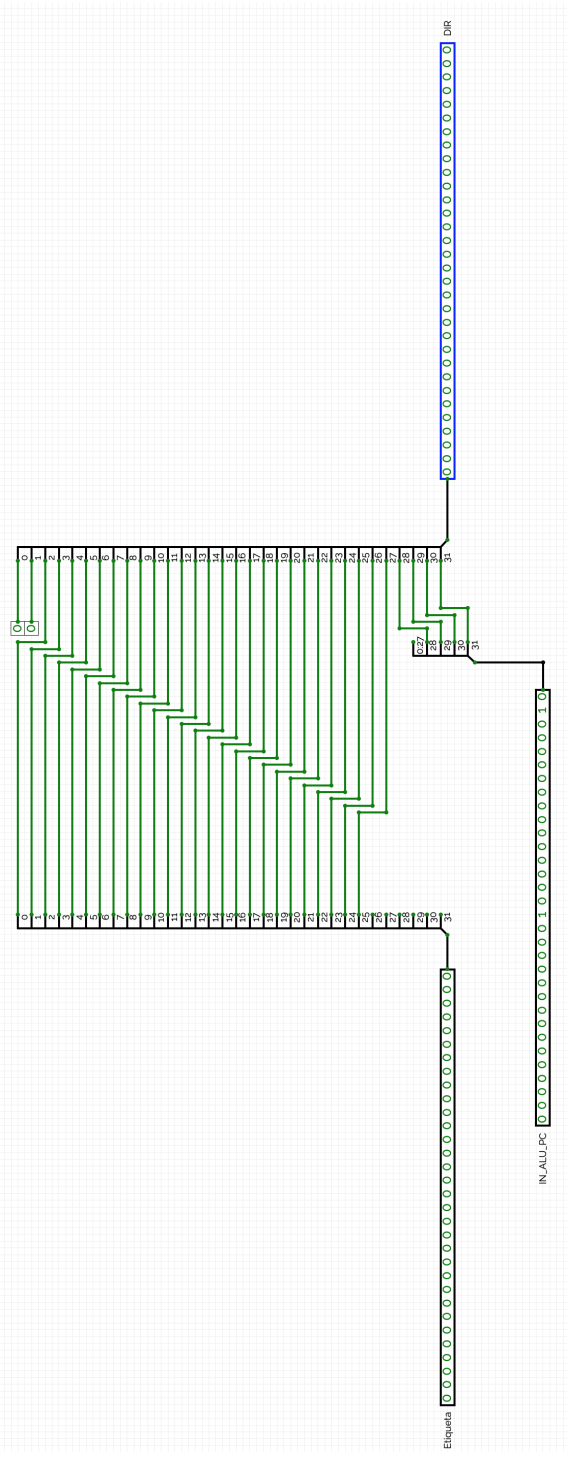

Figura 4.10: Implementación en CircuitVerse de la unidad para componer la dirección de salto incondicional. El circuito tiene dos entradas, una es la instrucción de salto que se está ejecutando, de los 32 bits de la instrucción únicamente se utilizarán los 26 bits del campo etiqueta. Como segunda entrada recibe el contenido del registro del contador de programa.

Figura 4.11: Implementación en CircuitVerse de la memoria de instrucciones. Teniendo esta 32 posiciones de memoria estando cada posición de memoria en una dirección múltiplo de cuatro. La memoria de instrucciones en la figura contiene varias instrucciones *addi* utilizadas para una prueba de la ruta de datos.

Otro aspecto importante de la memoria es el direccionamiento. Para ello en la implementación actual se recibe como entrada la dirección completa de 32 bits, de estos 32 bits se utilizarán cinco bits que permitan direccionar 32 posiciones de memoria. Para mantener el diseño original del MIPS R2000 y también del resto de procesadores se han utilizado los bits del 2 al 6 ambos incluidos para el direccionamiento, dejando los dos bits de menor peso sin examinar, de esta forma se consigue que las posiciones de memoria estén en direcciones múltiplos de cuatro.

Como último aspecto interesante de la implementación de la memoria de instrucciones cabe destacar el uso de un multiplexor de 32 entradas, donde cada entrada estará conectada a una posición de memoria o *CONSTANTVAL*. El multiplexor recibirá como señal de control los cinco bits antes mencionados de la dirección de 32 bits recibida como entrada. Dependiendo del valor de estos cinco bits se leerá una posición de memoria u otra.

4.1.9. Memoria de datos

Igual que la memoria de instrucciones, parte de la memoria de datos es un circuito externo al dado de silicio, pero el nivel L1 de la memoria cache sí se encuentra dentro del procesador, y de la misma forma que la anterior es un elemento intrínseco de la ruta de datos de la CPU.

Su función principal, como su nombre indica, es almacenar los datos del programa o programas en ejecución. Como se ha mencionado en el capítulo anterior, el procesador MIPS R2000 únicamente puede utilizar la información almacenada en memoria si esta primero se carga en un registro. Por tanto, realmente las únicas instrucciones que acceden a la memoria de datos son las instrucciones de carga y almacenamiento, ambas de tipo I, que mueven información entre los registros y la memoria de datos.

La memoria de datos para este proyecto se ha tenido que diseñar desde cero, dado que CircuitVerse actualmente no ofrece una solución adecuada para este componente. Igual que en la memoria de instrucciones se ha optado por una memoria de datos pequeña, de 32 posiciones, suficiente para los objetivos del proyecto.

La implementación de la memoria de datos se puede observar en la Figura 4.12. Su diseño es similar al diseño del banco de registros, dado que ambos componentes comparten funciones similares.

Para las posiciones de memoria se utilizan biestables de tipo D, muy adecuados para el cometido de la memoria de datos. Para el direccionamiento se reciben en la entrada correspondiente 32 bits, de los cuales se tendrán en cuenta cinco, para direccionar 32 posiciones de memoria. De la misma manera que en la memoria de instrucciones se saltan los dos bits de menor peso y directamente se examinan los bits del 2 al 6, ambos incluidos, de esta manera se logra que las posiciones de memoria se encuentren en direcciones múltiplos de cuatro. Véase la Figura 4.13.

Se puede separar el circuito en dos partes, la parte de la izquierda es la encargada de la operación de escritura, ver Figura 4.14. Esta parte esta formada por un decodificador que recibe la dirección desde el *splitter* antes mencionado y por el otro lado está conectado a una serie de puertas AND que como segunda entrada

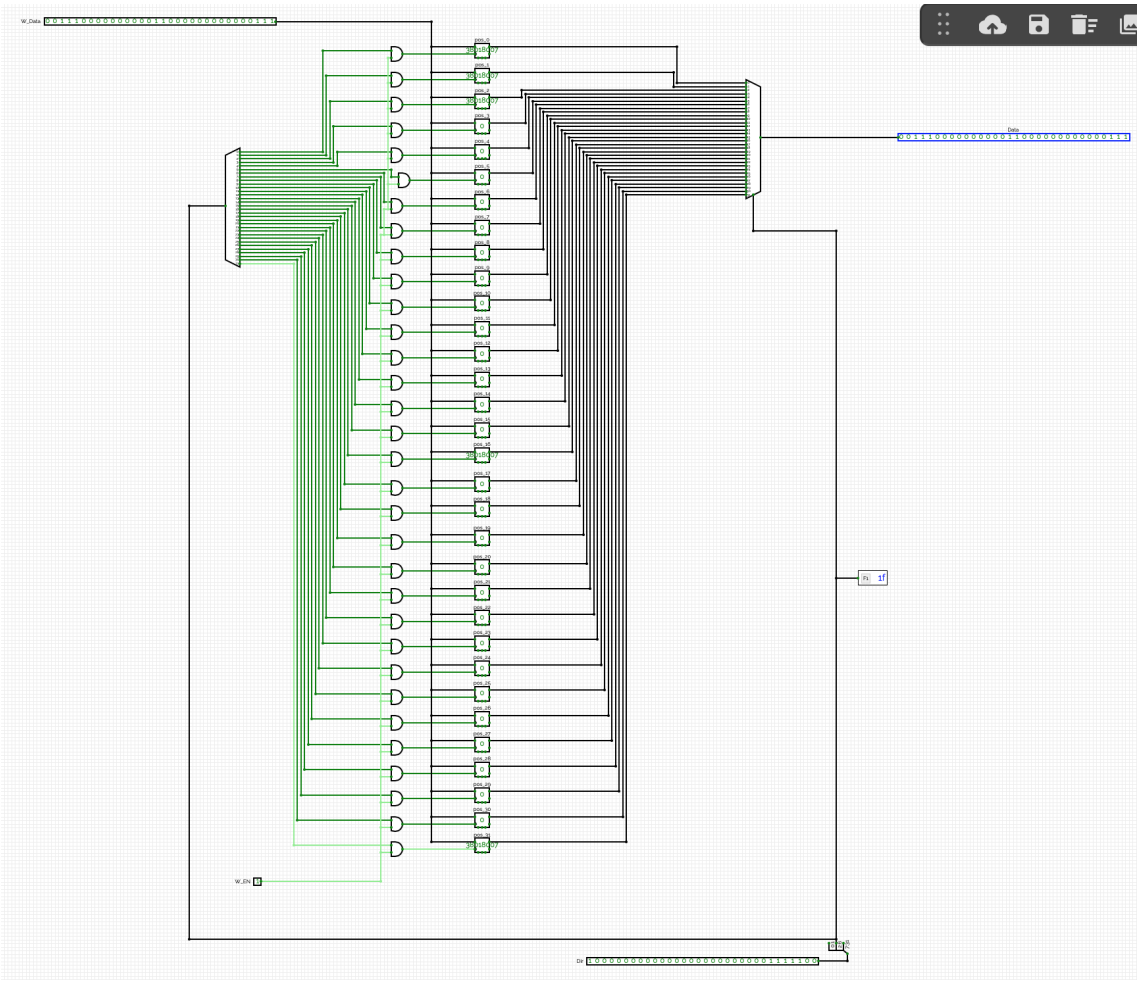

Figura 4.12: Implementación en CircuitVerse de la memoria de datos. El circuito tiene dos partes separadas, la parte a la izquierda de los biestables de tipo D, dedicada a la operación de escritura y la parte a la derecha de los *DFlipFlops* dedicada a la lectura de la memoria de datos.

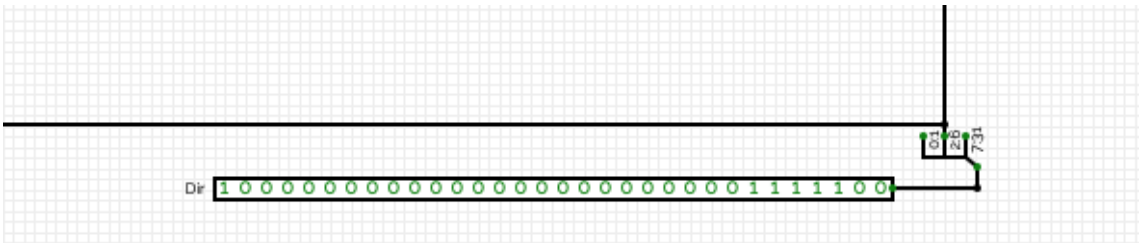

Figura 4.13: Direccionamiento de la memoria de datos, mediante una entrada de 32 bits, los cuales a continuación se separan en los bloques de bits que interesan mediante un *splitter*. El grupo de bits que corresponde, del bit 2 al bit 6, se transfieren al decodificador y al multiplexor correspondientes.

tienen el bit de habilitación de escritura y sus salidas están conectadas a los puertos de habilitación de los biestables. El dato a escribir llega a todos los biestables mediante la entrada de 32 bits correspondiente. Según la señal que recibe el decodificador, es decir, la dirección, activará una de las líneas que estará conectada a la puerta AND correspondiente, si se activa el bit de habilitación de escritura a su vez, se guardará el dato de la entrada en el biestable que se ha habilitado.

En la parte encargada de la escritura del circuito de la memoria de datos, el componente central es un multiplexor de 32 entradas, cada una conectada a la salida de un biestable o posición de memoria, véase Figura 4.15. Este componente es el encargado de elegir la información contenida de qué biestable transferir a la salida, dependiendo de los 5 bits de dirección que recibe en su entrada de control desde el *splitter* de la dirección.

4.1.10. Unidad de Control

La unidad de control es uno de los circuitos más complejos de cualquier procesador y uno de los más importantes. En la mayoría de los procesadores suele ocupar gran parte del espacio disponible sobre el dado de silicio debido a su complejidad.

Para facilitar la tarea compleja de diseñar e implementar la unidad de control del MIPS R2000 se han utilizado todas las herramientas ofrecidas por CircuitVerse. Se ha comenzado por utilizar una tabla de verdad para crear automáticamente una primera versión del circuito, que posteriormente se vería ampliado a medida que se da soporte a más instrucciones.

La primera versión de la unidad de control soportaba únicamente las instrucciones aritmético-lógicas de tipo R y solamente ofrecía las señales necesarias para estas instrucciones. Para ello la tabla de verdad tenía seis entradas, una por cada bit del campo de operación. Con esta información, el simulador CircuitVerse creó un circuito sencillo que tenía una línea negada y una línea no negada por cada entrada junto con una linea constantemente a 1 y otra constantemente a 0. Este circuito aunque sencillo fue muy útil para estructurar toda la unidad de control y se ha utilizado como base para las siguientes modificaciones.

La primera modificación de este circuito construido automáticamente fue introducir como entrada la instrucción completa y posteriormente separar los bits que interesan mediante *splitters*. Las salidas de este *splitter* se conectarán a las lineas que ha creado automaticamente CircuitVerse sustituyendo las seis entradas anteriores. El circuito queda como se observa en la Figura 4.16.

En este punto se introduce el primer subcircuito de todo el proyecto, se corresponde a la unidad de decodificación de función aritmética, véase la Figura 4.17. Su función en la unidad de control es, mediante los bits pertinentes de la instrucción recibida en la entrada, determinar la función aritmético-lógica y retornar en su salida el código de operación aritmética de la ALU correspondiente.

La unidad de decodificación de función aritmética se utiliza únicamente en las instrucciones de tipo R. Para ejecutar instrucciones de tipo I se ha modificado la unidad de control introduciendo nuevos elementos. Las instrucciones de tipo I necesitan que la propia unidad de control pueda determinar la operación

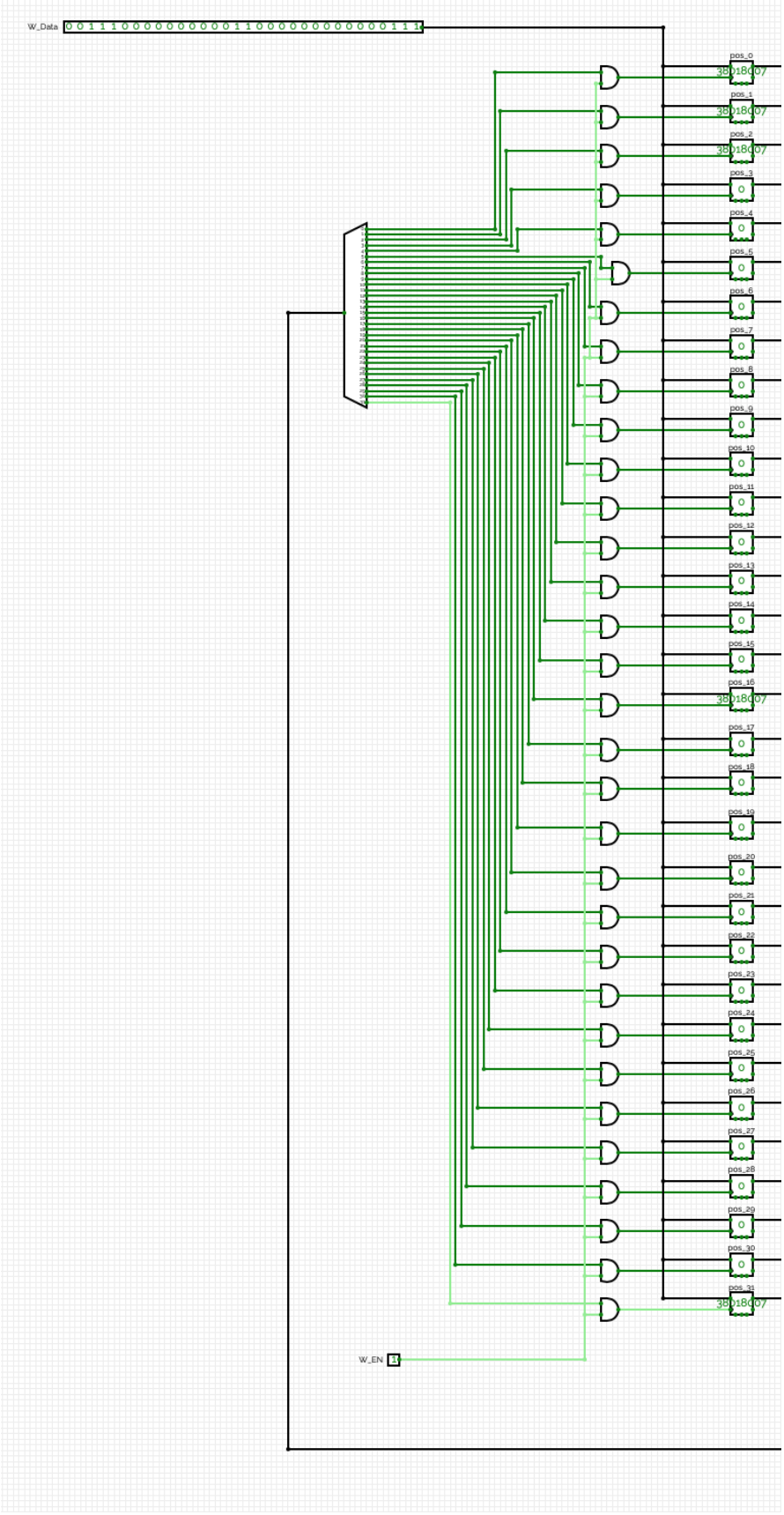

Figura 4.14: Parte encargada de la escritura del circuito que implementa la memoria de datos, se utilizan las tres entradas del circuito, una entrada de 32 bits para la dirección sobre la que escribir, una entrada de 32 bits para el dato a escribir y un bit de habilitación de la escritura.

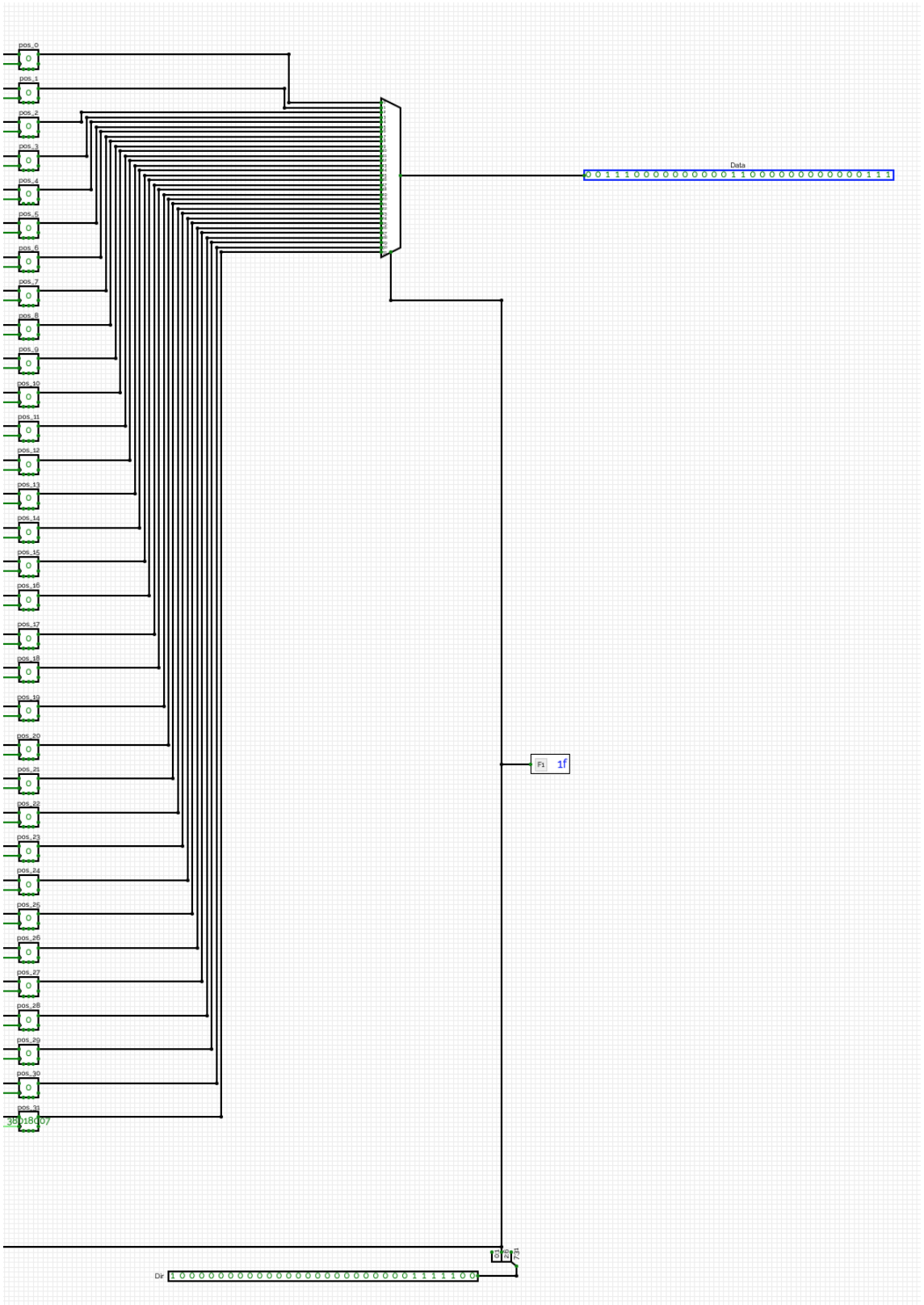

Figura 4.15: Parte encargada de la lectura del circuito que implementa la memoria de datos. Entre el *splitter* de la dirección y el multiplexor se ha introducido un *flag* para mostrar la dirección en decimal, para facilitar las pruebas con el circuito.

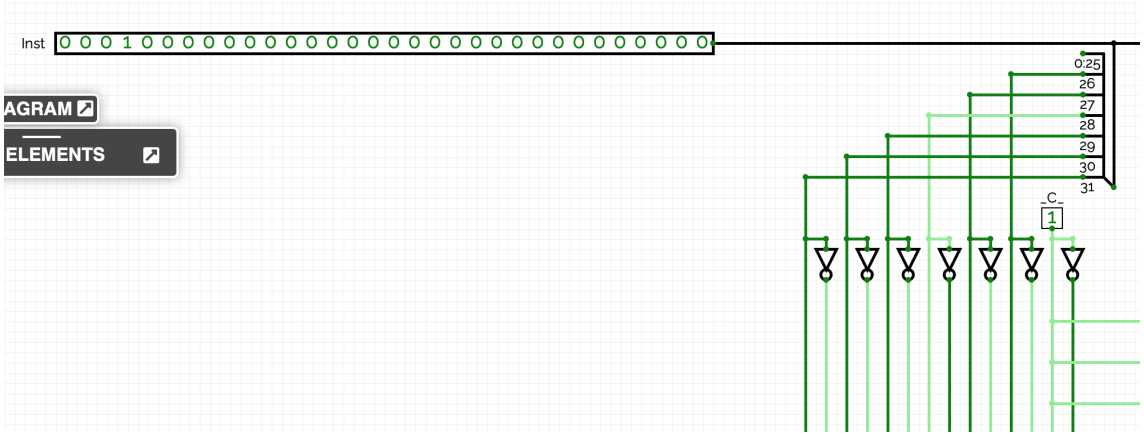

Figura 4.16: Entrada del circuito que implementa la unidad de control, se recibe la instrucción completa como entrada del circuito. En la parte derecha de la figura se observan las líneas negadas mediante una puerta NOT y las no negadas. Las últimas líneas a la derecha son constantes, una es siempre 1 y la negada es siempre 0.

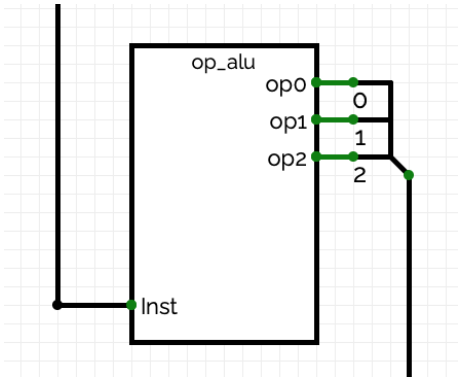

Figura 4.17: La unidad de decodificación de función aritmética introducida como subcircuito en la unidad de control. En su entrada recibe la instrucción completa proveniente de la entrada de la unidad de control y en sus salidas ofrece los bits de la operación aritmético-lógica a ejecutar en la ALU.

aritmético-lógica a ejecutar y transferir el código de operación correspondiente a la ALU.

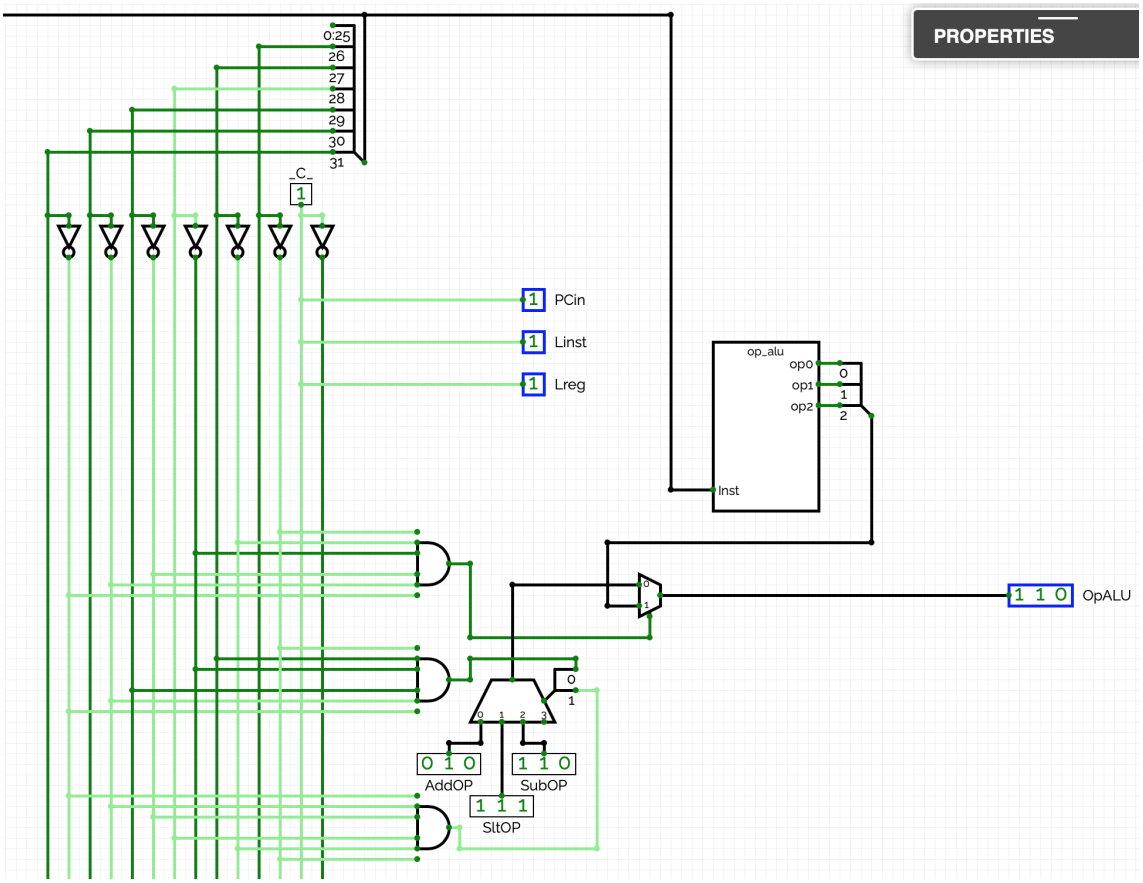

Figura 4.18: Parte de la unidad de control encargada de determinar el código de operación aritmética. Un multiplexor determina si el código de operación de la ALU lo establece la unidad de decodificación de función aritmética o si lo resuelve la propia unidad de control.

En la Figura 4.18 se puede observar el circuito capaz de determinar el código de operación de la ALU a ejecutar. La unidad de decodificación de función aritmética proporcionará la función aritmética cuando la instrucción a ejecutar sea de tipo R, esto depende del código de control que recibe el multiplexor al que está conectada la salida del subcircuito antes mencionado. Esta unidad está conectada a la entrada 1 del multiplexor, por tanto, su código de control debe ser 1 para activar esta entrada. Para ello, este código de control proviene de una puerta AND de seis entradas conectadas todas a las lineas negadas de los bits del código de operación de la instrucción. Es decir, la puerta AND está activa cuando el código de operación de la instrucción es 000000, identificando una instrucción tipo R.

En caso de no tratarse de una instrucción de tipo R, la entrada del multiplexor activa será la 0, a esta entrada está conectado un sencillo circuito capaz de determinar la función aritmética a ejecutar dependiendo del código de operación de la instrucción. Este circuito consiste en un multiplexor de 4 entradas a las cuales hay conectados valores constantes, en CircuitVerse *CONSTANTVAL*, cada uno correspondiente a una función aritmética. El código de control del multiplexor de cuatro entradas proviene de un *splitter* que junta las salidas de dos puertas AND. La primera puerta AND se activa si se trata de una instrucción *slti* y hace

llegar al multiplexor la señal de control 01 activando la entrada correspondiente. La siguiente puerta AND se activa si la instrucción es *beq*, haciendo llegar al multiplexor el código de control 10 actvando la salida correspondiente a la operación *sub* de la ALU. En caso de no tratarse de ninguna de estas dos instrucciones estará activa la entrada 0 del multiplexor, correspondiente al código de operación *add*.

Teniendo expuesta la parte de la unidad de control encargada de determinar la función aritmética a ejecutar en la ALU, es momento de estudiar la implementación del resto de señales de control.

Las tres primeras señales a examinar son *PCin*, *Linst* y *Lreg*. Estas tres señales están siempre activas, dado que siempre que se ejecute una instrucción estas acciones son necesarias, véase la Figura 4.18. Debido a ello se han conectado a la linea que constantemente es 1. Es importante mencionar que la señal *PCin* sí se utiliza pero las otras dos señales antes mencionadas se han incluido solo de manera simbólica sin llegar a utilizarse en la ruta de datos ya que la lectura del registro de la instrucción y la lectura del banco de registro se han decidido dejar siempre activas.

La implementación de las señales restantes se puede observar en la Figura 4.19. Se ha optado por un diseño sencillo que gira en torno al uso de puertas lógicas. Concretamente las puertas AND se utilizan para determinar la instrucción en ejecución. Se establecen las instrucciones que pueden activar cada señal y se han implementado puertas AND que se activan únicamente si el código de operación de la instrucción correspondiente coincide con el código de operación de la instrucción en ejecución. Adicionalmente si hay varias instrucciones que pueden modificar una señal y establecer su valor a 1, las puertas AND correspondientes están conectadas a una puerta OR que permite que la señal se active mediante distintas instrucciones.

Un caso excepcional es la señal *MxPC*, que es la única señal de control de dos bits. Además su activación no depende únicamente del código de operación de la instrucción actual, sino que, también depende de una señal externa, el bit Z concretamente. La señal *MxPC* controla el multiplexor de la entrada del registro del contador de programa, dado que se trata de un multiplexor de cuatro entradas su señal de control debe tener dos bits. Para establecer los dos bits de la señal en cuestión se ha utilizado un *splitter* que junta los bits provenientes de una y otra puerta AND, como se puede observar en la Figura 4.19. En este caso, la entrada 00 del multiplexor será la entrada activa siempre, exceptuando si la instrucción en ejecución es una instrucción *beq* o *j*. En caso de que se trate de una instrucción *j* se activará la puerta AND correspondiente que a su vez establecerá el bit de menor peso de la señal *MxPC* a 1. En cambio, si la instrucción a ejecutar es *beq* se activará la puerta AND correspondiente igual que antes, pero en esta ocasión, la salida de esta puerta AND está conectada a otra puerta AND que como segunda entrada tiene el bit Z, en caso de que se trate de una instrucción *beq* y el bit Z es 1 se establecerá el bit de mayor peso de la señal *MxPC* a 1, seleccionando posteriormente en la ruta de datos la entrada adecuada del multiplexor de la entrada del registro del contador de programa.

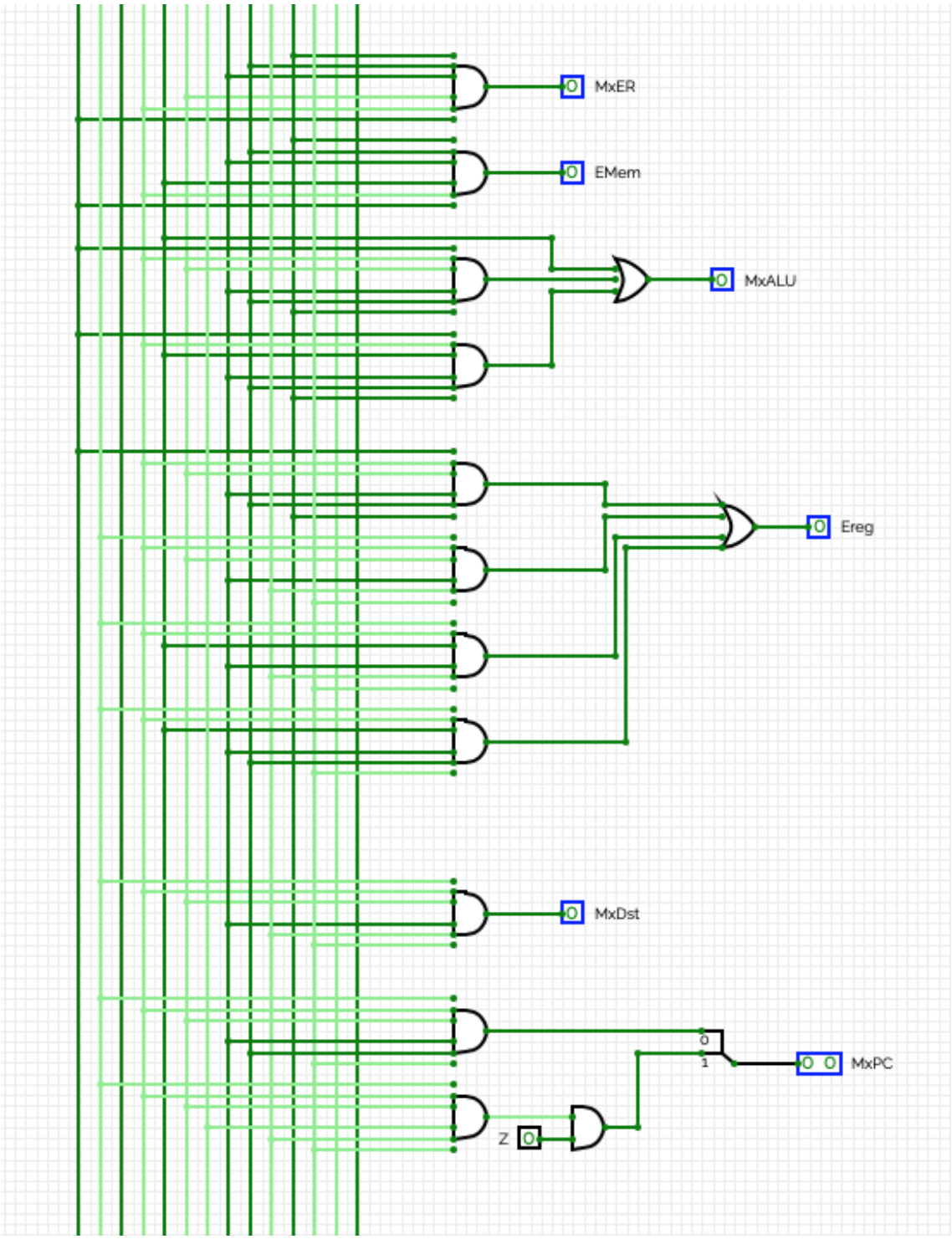

Figura 4.19: Implementación de la parte de la unidad de control donde se determinan gran parte de las señales de control. En la parte baja del circuito se puede observar la única otra entrada, correspondiente al bit Z, a parte de la instrucción completa.

4.2 Diseño e implementación de la ruta de datos monociclo

Como se ha mencionado anteriormente la ruta de datos es un complejo sistema de diversos componentes interconectados entre sí: Todo el sistema se controla desde la unidad de control mediante señales de control. La ruta de datos monociclo, hay que resaltar de nuevo, busca una instrucción en memoria y la ejecuta completamente en un solo ciclo de reloj.

La ejecución de una instrucción comienza con su búsqueda en la memoria de instrucciones. El contador del programa es una parte de la ruta de datos, un circuito formado por varios componentes, la función del cual es mantener en un registro la dirección de la instrucción a buscar; además en cada ciclo, una vez buscada la instrucción, el contador del programa es el encargado de pasar a la siguiente instrucción, o en el caso de ejecutar una instrucción de salto, incondicional o condicional, el contador del programa debe sustituir el contenido con la dirección a la que apunta la dirección de salto, modificando de esta forma el flujo de ejecución.

La implementación del circuito que conforma el contador de programa se puede observar en la Figura 4.20. Se ha seguido muy de cerca el diseño original del MIPS R2000 a la hora de implementar el contador del programa en CircuitVerse. Para el registro del PC y el registro de instrucción se han utilizado biestables de tipo D de 32 bits, igual que el resto de registros implementados hasta el momento en el proyecto.

A la entrada del registro del *program counter* hay conectado un multiplexor de cuatro entradas. Este multiplexor es el encargado de seleccionar la fuente de la que proviene la dirección de la próxima instrucción a buscar en memoria. Estas fuentes pueden ser tres, dependiendo de la última instrucción ejecutada:

1. **Caso 0:** La última instrucción ejecutada no es una instrucción de salto y se debe seguir con la ejecución normal del programa, por tanto; la fuente de la dirección a acceder en memoria es la ALU del contador del programa, que suma 4 al valor actual del registro del PC pasando a la siguiente posición en memoria. La salida de esta unidad aritmético-lógica especial está conectada a la entrada 0 del multiplexor antes mencionado.

2. **Caso 1:** La instrucción anterior en ejecución es una instrucción de salto incondicional. Debido a ello, el flujo de ejecución se verá modificado, estando la próxima instrucción a ejecutar en una posición de memoria no contigua a la actual. La fuente de la dirección de memoria en este caso es la unidad para componer la dirección de salto incondicional. Esta unidad se puede observar insertada como subcircuito en la ruta de datos en la Figura 4.20, bajo el nombre abreviado de *Calculo_DIR*.

   La salida del subcircuito anterior está conectada a la entrada 1 del multiplexor del contador de programa, siendo seleccionada esta fuente en el caso que la última instrucción ejecutada sea una instrucción de salto incondicional.

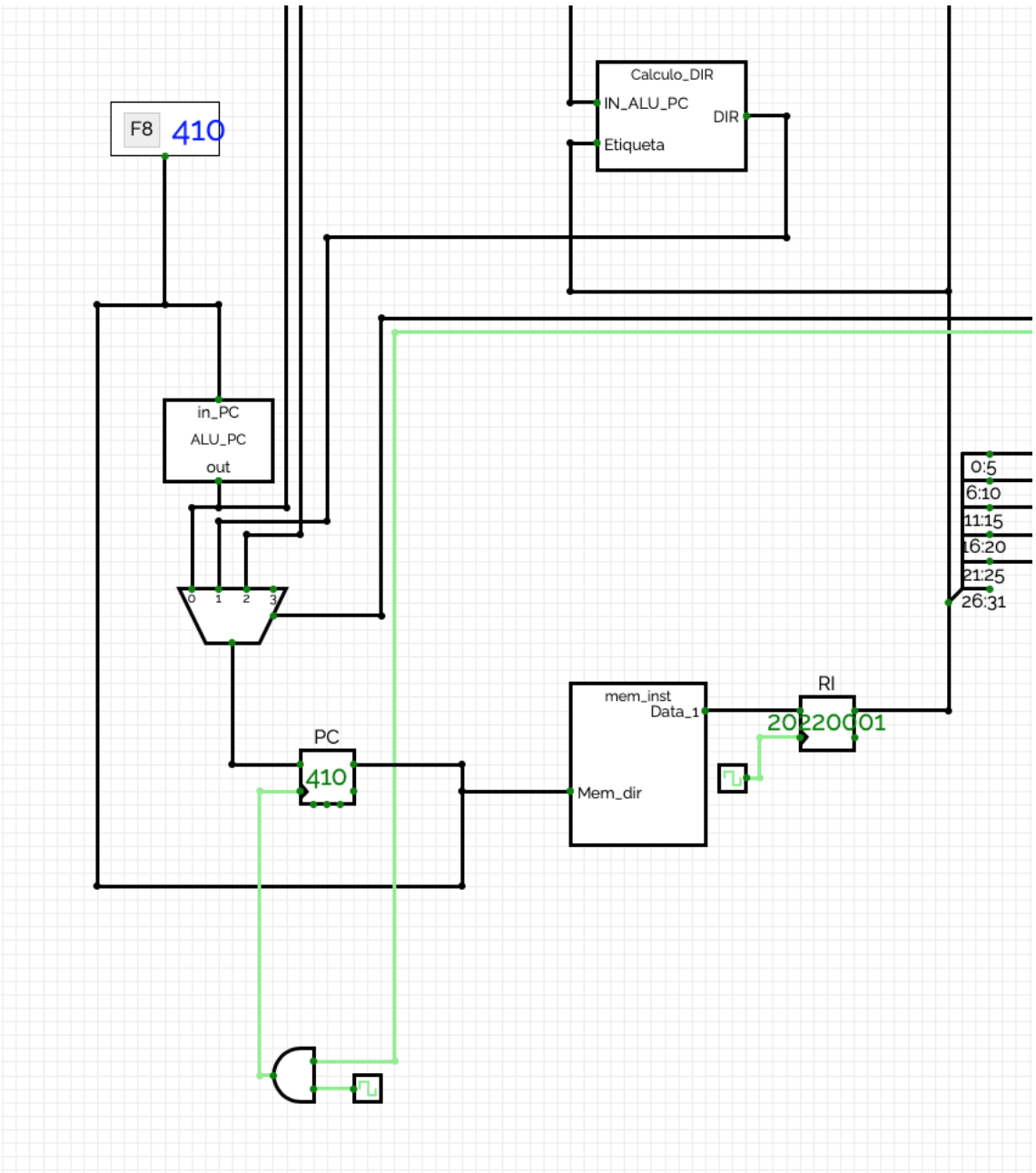

Figura 4.20: Implementación en CircuitVerse de la parte de la ruta de datos que conforma el contador del programa. La salida del registro de instrucción, *RI*, está conectada al bus general de la ruta de datos que lleva los 32 bits de la instrucción al resto de componentes del procesador.

3. **Caso 2:** La última instrucción ejecutada es una instrucción de salto condicional. En este caso, igual que en el anterior, el flujo de ejecución se verá modificado, pero únicamente si llega a cumplirse la condición indicada en la instrucción de salto condicional ejecutada.

   La fuente en este último caso es la unidad de cálculo de la dirección de salto, que se puede observar en la Figura 4.21 bajo el nombre abreviado *ALU_Saltos*. La salida de esta unidad está conectada a la entrada 2 del multiplexor del contador de programa.

Hay que acentuar de nuevo, que el multiplexor selecciona la fuente de la dirección dependiendo de la señal de dos bits que recibe de la unidad de control, siendo la ALU del contador de programa la fuente seleccionada por defecto, es decir, el caso 0 es el caso normal.

Estando seleccionada la entrada adecuada del multiplexor, los bits de la dirección atraviesan el multiplexor y mediante su salida llegan a la entrada del registro PC, aunque este no los almacenará hasta que no se active su señal de habilitación, que por su parte depende de la señal de control *PCin* y de la señal de reloj del procesador, mediante una puerta AND. En el momento que ambas estén a nivel alto, la puerta AND envía un 1 a la entrada de habilitación al registro del *program counter*, que a su vez sustituirá el valor almacenado por el de su entrada, recibido desde el multiplexor.

Una vez los bits de la dirección son almacenados por el registro PC, estos mediante su salida llegan, por una parte de vuelta a la ALU del contador del programa, que le sumará 4 al valor actual para repetir el proceso antes explicado, el próximo ciclo de reloj. Por otra parte, los bits de la dirección además llegan a la entrada del subcircuito que implementa la memoria de instrucciones, explicada en el anterior apartado, que a su vez los lee y proporciona en su salida el valor contenido en la posición de memoria indicada por el registro PC.

La salida de la memoria de instrucciones está conectada al registro de instrucción que almacenará los 32 bits de la nueva instrucción, sustituyendo la anterior instrucción almacenada. A la salida de este registro se encuentra el bus general de la ruta de datos, que lleva los bits de la instrucción al resto de componentes.

A este bus general en primera instancia tenemos conectados dos componentes, la unidad de control, que como se ha visto en el apartado anterior, recibe la instrucción entera y posteriormente separa los bits que necesita, y por otra parte está conectado un *splitter* que separa los bits de la instrucción en los grupos de bits correspondientes a una instrucción de tipo R.

Desde este *splitter* los bits correspondientes a los campos RS, RT y RD llegan a las entradas del banco de registros (véase la Figura 4.22 para una mejor apreciación). En el caso de la entrada tocante a la dirección de escritura, antes de esta entrada existe un multiplexor, que selecciona el campo correspondiente a la dirección de escritura, esto es, selecciona el campo RD o RT como dirección de escritura, dependiendo de si se trata de una instrucción de tipo R o de tipo I, respectivamente. La señal de control asociada al multiplexor mencionado es *MxDst*.

Por otro lado, desde el *splitter* que separa los bits del bus general, los 16 bits de menor peso de la instrucción también llegan a la unidad extensora del bit de

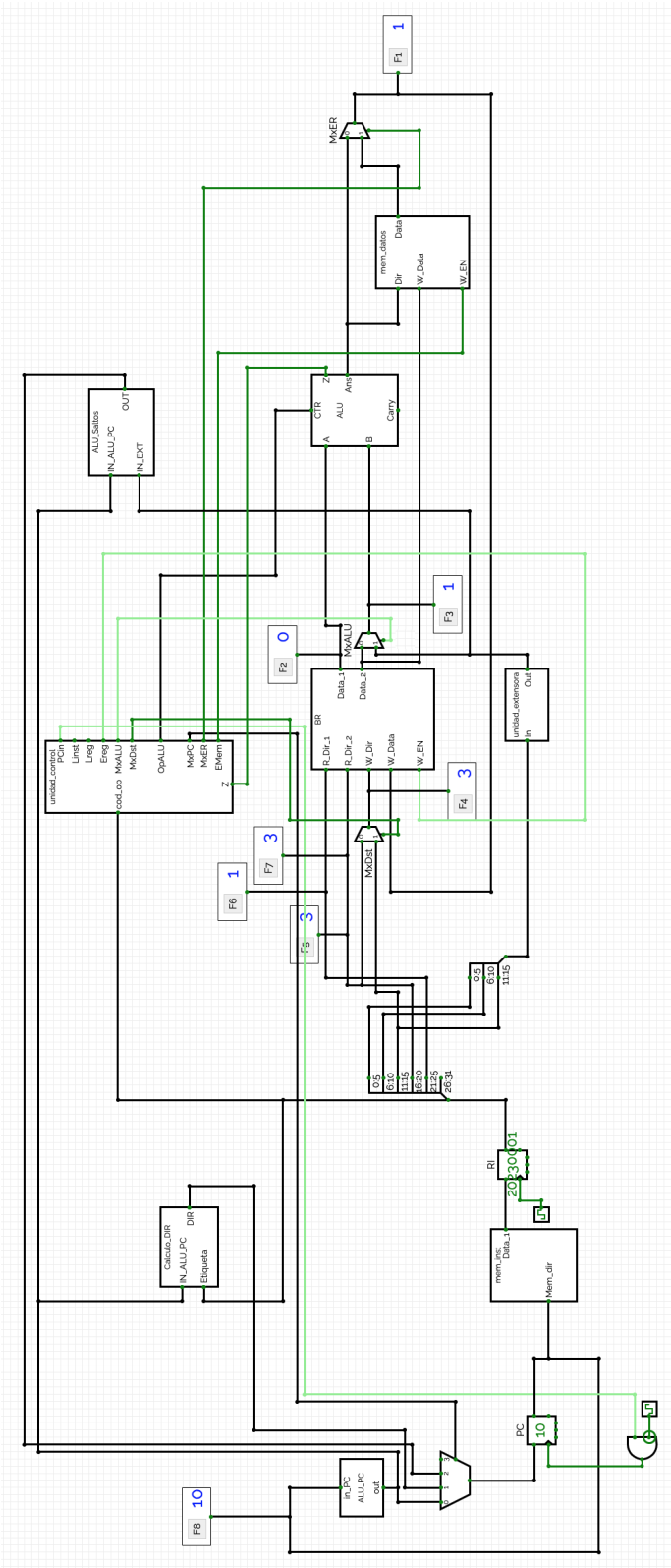

Figura 4.21: Implementación en CircuitVerse de la ruta de datos monociclo del procesador MIPS R2000. En la figura se pueden observar todas las señales de control en verde, claro y oscuro, dependiendo de si están activas o no, exceptuando las señales de control de la ALU y del multiplexor del contador de programa que utilizan señales de control de varios bits de ancho y están en color negro como el resto de buses de la ruta de datos..

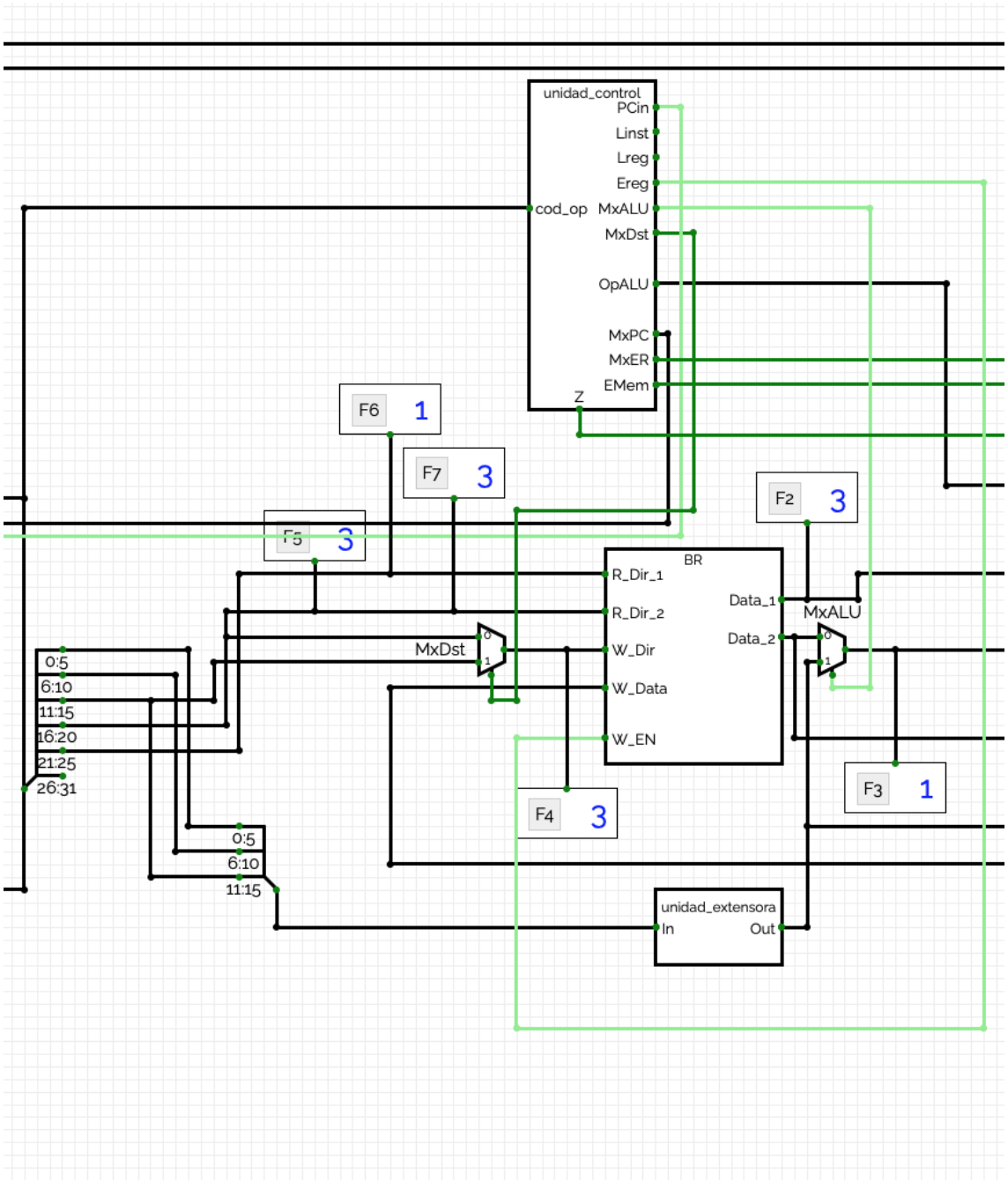

Figura 4.22: Implementación en CircuitVerse de la parte central de la ruta de datos. Se pueden observar insertados varios *flags* en lugares estratégicos para analizar el contenido de las distintas líneas existentes en la ruta de datos, permitiendo apreciar la ejecución de cada instrucción y los cambios ocasionados por ello, además de facilitar enormemente la depuración del circuito durante su implementación.

signo. No obstante, antes de la entrada de la unidad extensora se ha introducido un *splitter* que tiene como objetivo unificar los tres campos que se reciben desde el *splitter* anterior.

Tras exponer las conexiones entre el bus general y el banco de registros y la unidad extensora del bit de signo, el siguiente paso en la ejecución de una instrucción es la unidad aritmético-lógica. La ALU del procesador recibe dos operandos y el código de operación desde la unidad de control. El primero de los dos operandos siempre se recibe desde el banco de registros, siendo indicado por el campo RS de la instrucción; debido a ello están conectados directamente la salida del banco de registros referente al primer registro leído y la entrada de la ALU correspondiente al primer operando. Véase la Figura 4.23 para una mejor comprensión.

En cambio, el segundo operando puede provenir de distintas fuentes. En caso de ser una instrucción de tipo R, proviene igual que el primer operando, del banco de registros, correspondiente al contenido del registro indicado por el campo RT de la instrucción.

En caso de tratarse de una instrucción de tipo I, el operando procede de la unidad extensora del bit de signo. Dado que hay dos posibles fuentes se ha introducido un multiplexor antes de la entrada del segundo operando de la ALU, para seleccionar una u otra fuente (véase la Figura 4.23). Este multiplexor de dos entradas se gobierna mediante la señal de control *MxALU*.

Como se ha mencionado anteriormente durante este proyecto, las instrucciones de salto condicional también son instrucciones de tipo I, que emplean el campo de desplazamiento para indicar la dirección de salto. Por tanto, estas instrucciones igualmente utilizan la unidad extensora del bit de signo para extender los 16 bits del campo desplazamiento a 32 bits, para posteriormente traspasar estos 32 bits a la unidad de cálculo de la dirección de salto. Para ello existe una conexión entre la salida de la unidad extensora y la entrada correspondiente de la unidad de cálculo de la dirección de salto, introducida en la implementación de la ruta de datos como subcircuito bajo el nombre abreviado de *ALU_Saltos* (véase la Figura 4.23).

La unidad de cálculo de la dirección de salto en sus entradas tiene por una parte conectada la unidad extensora, como se ha indicado anteriormente, y en la otra entrada está conectada la ALU especial del contador de programa, por ello hay un bus de 32 bits que llega desde la salida de esta unidad aritmético-lógica especial hasta la entrada de la unidad de cálculo de la dirección de salto. La salida de esta unidad, por su parte, está conectada a la entrada 2 del multiplexor del contador de programa, tal como se aprecia en la Figura 4.21.

El subcircuito de la ALU por un lado recibe los dos operandos, aunque además de estos tiene una tercera entrada dedicada al código de operación que ha de ejecutar. Esta entrada está conectada a un bus de control de 3 bits llamado *OpALU*, proveniente de la unidad de control.

Una vez la ALU tiene los dos operandos y el código de operación en sus entradas ejecuta la operación correspondiente y el resultado de esta se presenta en la primera salida del subcircuito, llamada para abreviar *Ans*. Junto a esta salida, como se ha mencionado anteriormente durante el capítulo, la unidad aritmético-

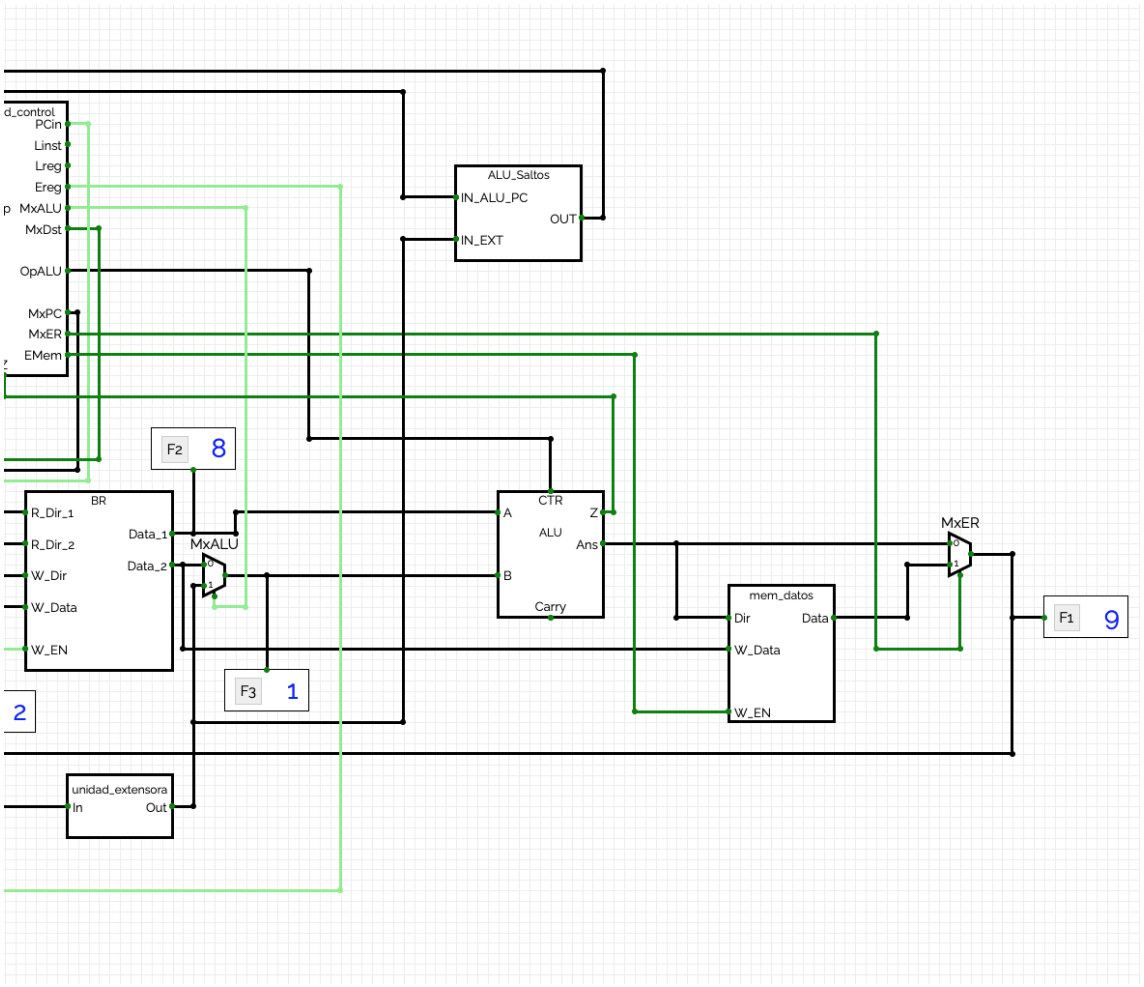

Figura 4.23: Implementación en CircuitVerse de la parte derecha de la ruta de datos. En esta parte de la ruta de datos se encuentran insertados la unidad aritmético-lógica y la memoria de datos, entre otros componentes. Además se pueden observar varios *flags* conectados a los buses enlazados a las entradas de la ALU.

lógica dispone de una salida auxiliar *Z* de un bit, que indica mediante un 1 si el resultado de la operación es 0.

La salida *Z* vuelve a la unidad de control (véase la Figura 4.21), y se utiliza en caso de tratarse de una instrucción de salto condicional, para resolver si se cumple la condición y en última instancia si llevar a cabo el salto o no.

En cambio, el resultado de la operación, es decir el valor expuesto en la salida *Ans*, llega a dos componentes distintos (Figura 4.23). Por un lado la salida de la ALU llega a un multiplexor de dos entradas cuyo objetivo es seleccionar la fuente del dato que se guardará en registro. Por otro lado llega al subcircuito correspondiente a la memoria de datos, siendo el valor calculado en la ALU la dirección que se leerá desde la memoria de datos.

La memoria de datos recibe en su entrada de dirección, llamada para abreviar *Dir*, el valor calculado por la ALU, busca la posición indicada por este y recupera el dato o valor almacenado en esta posición de memoria mediante su salida. En caso de tratarse de una instrucción de almacenamiento, por ejemplo, instrucción *sw*, ocurre el mismo proceso pero en este caso la dirección calculada por la ALU indica la posición de memoria en la que se guardará el dato proveniente del banco de registros, siendo recibido este dato en la entrada llamada *W_Data* de la memoria de datos. Es importante indicar que el subcircuito de la memoria de datos en cada ciclo recibe en sus entradas una posible dirección y un posible dato a guardar, pero almacenar o no este valor, depende de la entrada de habilitación de escritura, llamada para abreviar *W_EN*, que a su vez está controlada por la señal de control *EMem* proveniente de la unidad de control (véase la Figura 4.23).

Hay que mencionar que la lectura de la posición de la memoria de datos indicada por la dirección recibida siempre está activa, pero utilizar el valor leído o no depende del multiplexor *MxER*, el mismo al que está conectada directamente la salida de la ALU. Es decir, seleccionar la fuente del dato escrito en el banco de registros depende del multiplexor antes mencionado y además hay dos posibles fuentes: por un lado la unidad aritmético-lógica y por otro lado la memoria de datos. Este multiplexor se rige por la señal de control *MxER* proveniente de la unidad de control, y únicamente se pondrá a 1 si se está ejecutando una instrucción de carga, por ejemplo, instrucción *lw*. Es decir, el multiplexor seleccionará como fuente la memoria de datos, conectada a la entrada 1 del multiplexor, si se trata solamente de una instrucción de carga.

Cabe destacar que, además de seleccionar la fuente del dato a escribir en el banco de registros, es necesario que esté activa la entrada de habilitación de escritura del banco de registros, llamada en esta implementación *W_EN*. Esta entrada está conectada a la señal de control *Ereg* proveniente de la unidad de control (véase la Figura 4.22).

Otro aspecto importante de la implementación de la ruta de datos es la conexión existente entre la salida del registro de instrucción y la unidad para componer la dirección de salto incondicional. Esta unidad, como se ha visto en el apartado anterior, dispone de dos entradas, en una de ellas recibe la instrucción desde el registro de instrucción, para posteriormente utilizar los bits del campo etiqueta. Junto con esta entrada existe una segunda, mediante la cual la unidad recibe la dirección calculada por la ALU del contador de programa. Para ello existe una conexión de 32 bits entre la salida de la ALU del contador de programa y esta uni-

dad para componer la dirección de salto. En la salida de esta unidad se obtendrá la dirección a la que saltar, esta salida está conectada al multiplexor del contador de programa que selecciona la fuente de la dirección de la próxima instrucción y en este caso dependiendo de la señal que reciba de la unidad de control determinará si utilizar la dirección calculada en la unidad mencionada anteriormente. Obsérvense las Figuras 4.21 y 4.20 para un mejor entendimiento de la exposición anterior.

4.3 Camino seguido por cada tipo de instrucción

En el juego de instrucciones del procesador MIPS R2000 existen tres tipos de instrucciones: tipo R, tipo I y tipo J. Cada tipo de instrucción sigue un camino un tanto diferente dentro de la ruta de datos y es interesante destacar el recorrido que sigue cada tipo de instrucción y las diferencias entre estos caminos. Es importante mencionar que dentro de un mismo tipo de instrucción, pueden haber instrucciones que siguen recorridos distintos y utilicen componentes de la ruta de datos diferentes, por ejemplo, una instrucción *addi* y una instrucción *lw* son ambas de tipo I, pero tienen un recorrido diferente dentro de la ruta de datos y utilizan distintos componentes.

4.3.1. Instrucciones de tipo R

Comenzamos por las instrucciones de tipo R. Una instrucción de este tipo tiene el formato visto en la Figura 4.24. Para cumplimentar la exposición que sigue a continuación, observar la Figura 4.21. Al comienzo del ciclo de reloj, como ocurre con todas las instrucciones se lee, la dirección desde el registro del contador de programa y se accede a la posición de la memoria de instrucciones indicada por esta, recuperando la instrucción guardada y almacenándola en el registro de instrucción.

A continuación, varios componentes reciben los bits de esta, siendo dos de ellos la unidad de control y el banco de registros. La unidad de control descifra el código de operación determinando que es una instrucción aritmética, manda las señales de control necesarias y al mismo tiempo comprueba el código de función resolviendo la operación a ejecutar en la ALU, enviando el código de operación de la ALU correspondiente a esta.

El banco de registros al mismo tiempo lee los bits de la instrucción correspondientes a los campos *RS*, *RT* y *RD* accediendo a los valores almacenados en los registros indicados por los dos primeros mencionados y llevando estos valores a sus salidas. Desde las dos salidas del banco de registros los valores de los registros indicados llegan, el primero directamente a la entrada del primer operando de la ALU y el valor de la segunda salida llega a la entrada 0 del multiplexor llamado *MxALU*, que ha recibido la señal de control habilitando esta entrada y a continuación permite el paso de los bits que llegan a la entrada del segundo operando de la ALU.

La unidad aritmético-lógica en este momento tiene preparados los dos operandos que necesita y recibe el código de operación proveniente de la unidad de

control, ejecuta la operación aritmética indicada por este y presenta en su salida *Ans* el valor calculado tras la operación, además en su salida *Z* indica si el valor obtenido es equivalente al 0.

El valor calculado llega al multiplexor *MxER* en su entrada 0, que ha recibido la señal de control y habilita el paso de los bits de esta entrada que mediante esta acción llegan al banco de registros en su entrada correspondiente al dato a escribir, llamada *W_Data*. A continuación, se manda la señal de habilitación al registro indicado por el campo *RD*, dado que el banco de registros ha recibido en su entrada *W_EN* la señal de habilitación de escritura, almacenando finalmente en este registro el valor calculado.

Durante los pasos anteriores, la ALU del contador de programa ha sumado 4 al valor almacenado en el registro PC, es decir, ha aumentado en 4 la dirección de la instrucción ejecutada, pasando a apuntar a la siguiente posición de la memoria de instrucciones.

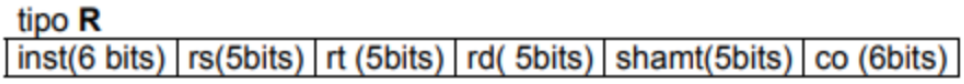

Figura 4.24: Formato de las instrucciones de tipo R, existen 5 bits, campo *shampt*, que no se utilizan en este tipo de instrucciones.

Con esto termina el ciclo de reloj y con ello la ejecución de una instrucción de tipo R. Los componentes que no se han mencionado no se utilizan durante la ejecución de una instrucción de este tipo, y aunque es cierto que algunos proporcionan valores según los bits de la instrucción que les corresponden, estos valores no se llegan a utilizar y son descartados en los multiplexores siendo seleccionados los valores proporcionados por los componentes adecuados a la ejecución de una instrucción de tipo R.

4.3.2. Instrucciones de tipo I

Las instrucciones de tipo I integran varias categorías de instrucciones que comparten el mismo formato de instrucción, visto en la Figura 4.25. Algunas categorías son: instrucciones aritméticas con inmediato, de carga y almacenamiento, instrucciones de salto condicional, entre otras. Aunque es cierto que todas estas instrucciones comparten el mismo formato, los caminos que siguen dentro de la ruta de datos y los componentes de los que hacen uso se diferencian entre una categoría y otra.

Figura 4.25: Formato de las instrucciones de tipo I

La ejecución de una instrucción sea del tipo que sea o de la categoría que sea, comienza de la misma manera, como se ha mencionado antes, con su búsqueda en la memoria de instrucciones. Se accede a la posición de memoria a la que apunta la dirección almacenada en el registro del contador de programa y se recu-

peran los 32 bits de la instrucción almacenando estos a continuación en el registro de instrucción.

Instrucciones aritméticas con inmediato

Esta categoría abarca todas las instrucciones aritméticas que utilizan como segundo operando un número almacenado directamente en la propia instrucción y no como es de costumbre, en un registro. Igual que su función, la ruta que siguen las instrucciones aritméticas con inmediato es ciertamente muy parecida al camino utilizado por las instrucciones de tipo R, aunque con algunas diferencias importantes.

En la Figura 4.26 se puede observar el camino recorrido por una instrucción aritmética con inmediato durante su ejecución. Claramente es muy parecido al recorrido por una instrucción aritmética de tipo R. Aunque existen ciertas diferencias importantes: una de ellas es que el registro destino lo marca el campo RT y no el campo RD como ocurría en las instrucciones de tipo R. Esto depende del multiplexor *MxDst* que en este caso tiene su señal de control a 0 y por tanto, selecciona su entrada 0, que se corresponde con el campo RT.

Otra desemejanza que es importante señalar es la fuente del segundo operando de la ALU, que en este caso es la unidad extensora del bit de signo. Esto se debe a que, como se ha mencionado antes, el segundo operando está directamente almacenado en los 16 bits de menor peso de la instrucción, correspondientes al campo *Inmediato*. Para seleccionar como fuente del segundo operando la unidad extensora se utiliza el multiplexor *MxALU*, que en este caso tiene su señal de control puesta a 1 y por tanto, selecciona su entrada 1 correspondiente a esta unidad.

El resto del camino que recorre una instrucción aritmético-lógica con inmediato es equivalente al camino recorrido por una instrucción de tipo R.

Instrucciones de carga

Las instrucciones de carga, como es bien sabido, tienen como función recuperar un dato desde la memoria de datos y almacenar este en un registro. Para ello utilizan por una parte el campo RS, para seleccionar el registro en el cual hay almacenada una dirección de memoria y sobre está dirección se aplica un desplazamiento indicado por el campo *Inmediato*, formado por los 16 bits de menor peso de una instrucción de tipo I. Por otra parte, el campo RT selecciona el registro en el cual se almacenará el dato recuperado desde memoria.

Observando la Figura 4.27 se puede apreciar claramente que el camino seguido por una instrucción de carga es muy similar a la ruta seguida por una instrucción aritmética con inmediato. La única diferencia importante consiste en que tras la ejecución de la operación en la ALU, el valor calculado por esta se corresponde a la dirección que se accederá en memoria de datos, por tanto, tras llegar este valor a la memoria de datos, se recupera el valor almacenado que llega al multiplexor *MxER* en su entrada 1. Este multiplexor es el encargado de seleccionar la fuente del dato que se almacenará en registro, que en este caso, es la memoria de

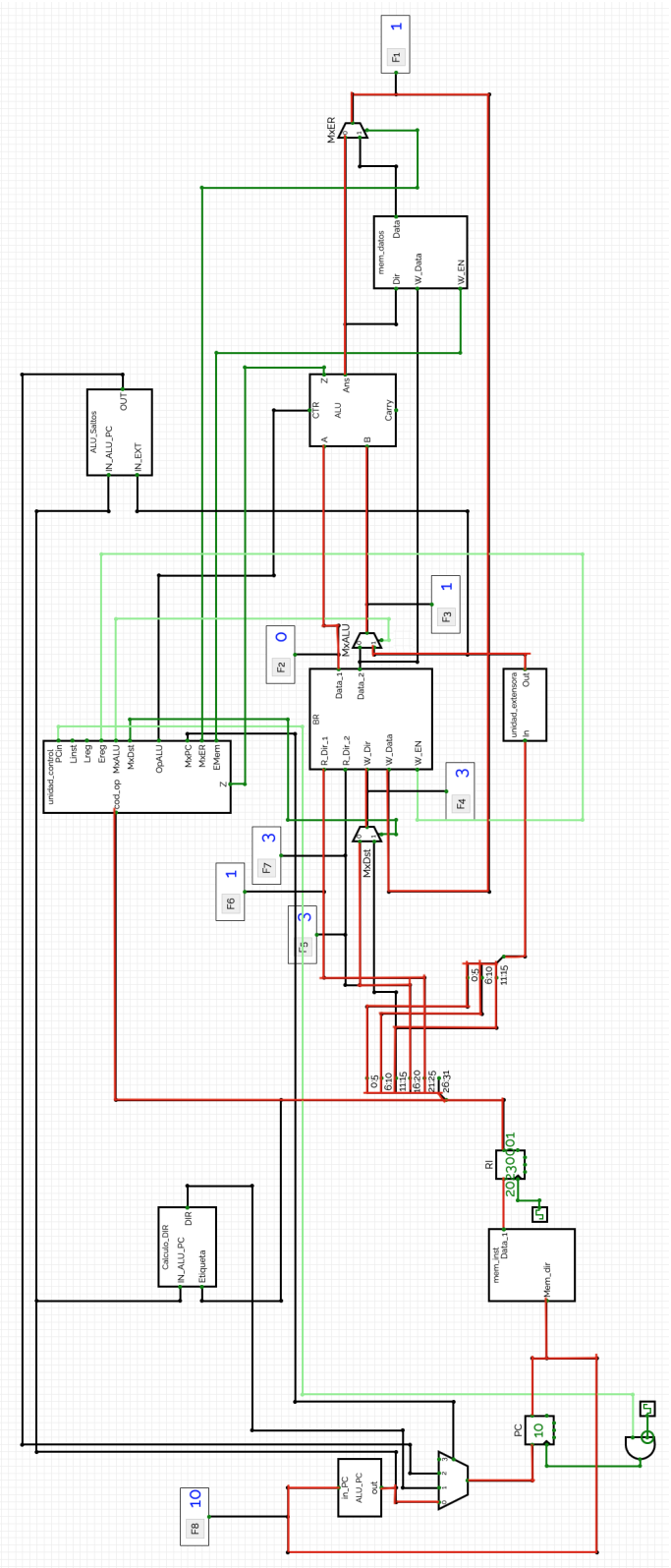

Figura 4.26: Ruta seguida por las instrucciones aritméticas con inmediato. En rojo aparecen marcados los buses utilizados durante la ejecución de la instrucción. Las señales de control no están resaltadas dado que todas las señales de control se tienen en cuenta durante la ejecución de una instrucción, estando estos a 1 o a 0.

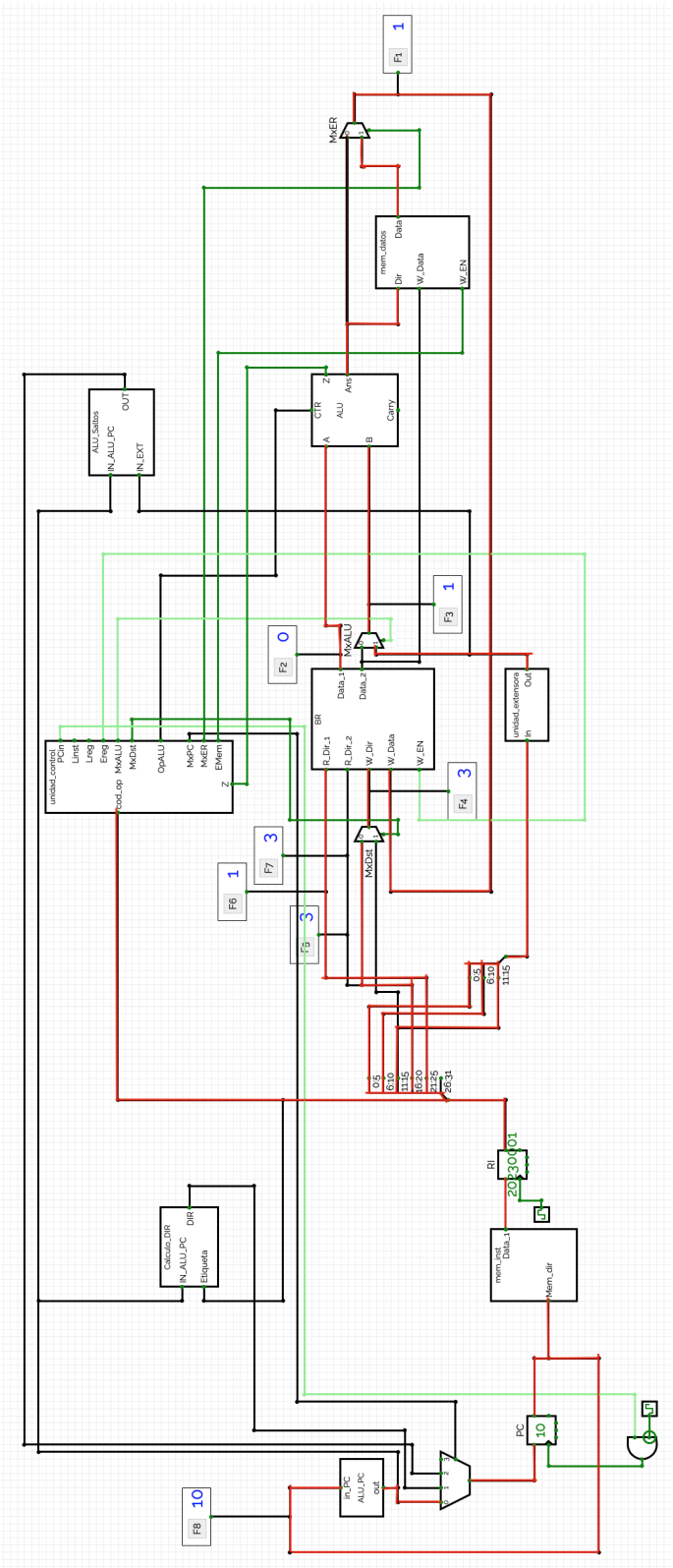

Figura 4.27: Ruta seguida por las instrucciones de carga. En rojo aparecen marcados los buses utilizados durante la ejecución de la instrucción. Las señales de control no están resaltadas dado que todas las señales de control se tienen en cuenta durante la ejecución de una instrucción, estando estos a 1 o a 0.

datos. Por tanto, en su entrada de control recibirá un 1, seleccionando la entrada número 1.

Es importante señalar que para el cálculo de la dirección a la que acceder en memoria, la unidad aritmético-lógica ejecuta una operación de suma, en inglés *add*. Agregando a la dirección recibida como primer operando el inmediato de 16 bits de la instrucción que ha sido extendido a 32 bits por la unidad extensora, recibido como segundo operando.

Instrucciones de almacenamiento

Las instrucciones de almacenamiento, como su nombre indica, tienen como objetivo almacenar en la memoria de datos un valor almacenado en un registro. Las instrucciones de almacenamiento utilizan, igual que las instrucciones de carga, el campo RS para indicar el registro que almacena la dirección y el campo del inmediato de 16 bits para indicar el desplazamiento que se aplicará sobre esta dirección. En cambio, a diferencia de la instrucciones de carga, las instrucciones de almacenamiento utilizan el campo RT para indicar el registro que almacena el dato a guardar en la memoria.

En la Figura 4.28 se puede observar la ruta seguida por una instrucción de almacenamiento. La ejecución de esta categoría de instrucciones difiere del resto de instrucciones vista hasta el momento en cuanto a que no se escribe ningún dato en el banco de registros tras la ejecución. Por tanto, las posiciones de los multiplexores *MxDst* y *MxER* no importan dado que no llegará a activarse la señal de escritura del banco de registros.

Es interesante mencionar que durante la ejecución de esta categoría de instrucciones se utilizan el registro leído correspondiente al campo RS, el registro leído correspondiente al campo RT y además el valor inmediato almacenado en la propia instrucción.

Instrucciones de salto condicional

Las instrucciones de esta categoría son las primeras vistas hasta el momento en este apartado que pueden cambiar el flujo de ejecución. Como se indica en su nombre, las instrucciones de salto condicional, permiten saltar a la dirección apuntada por la instrucción en caso de que la condición indicada sea cierta.

Este proyecto implementa como instrucción de salto condicional la instrucción *beq* que ejecuta el salto si los dos operandos, almacenados en los registros indicados por los campos RS y RT, son iguales. es decir si su resta equivale da como resultado un 0.

En estas instrucciones, igual que en las instrucciones de almacenamiento no se llega a almacenar nada en el banco de registros tras la ejecución de la instrucción. Por tanto, de la misma manera que antes, la selección de los multiplexores *MxDst* y *MxER* es irrelevante para la ejecución de estas instrucciones.

Estas son las primeras instrucciones que llegan a interactuar en el flujo de ejecución del programa, pudiendo cambiar el contenido del registro del contador de programa según si se cumple la condición establecida por la instrucción.

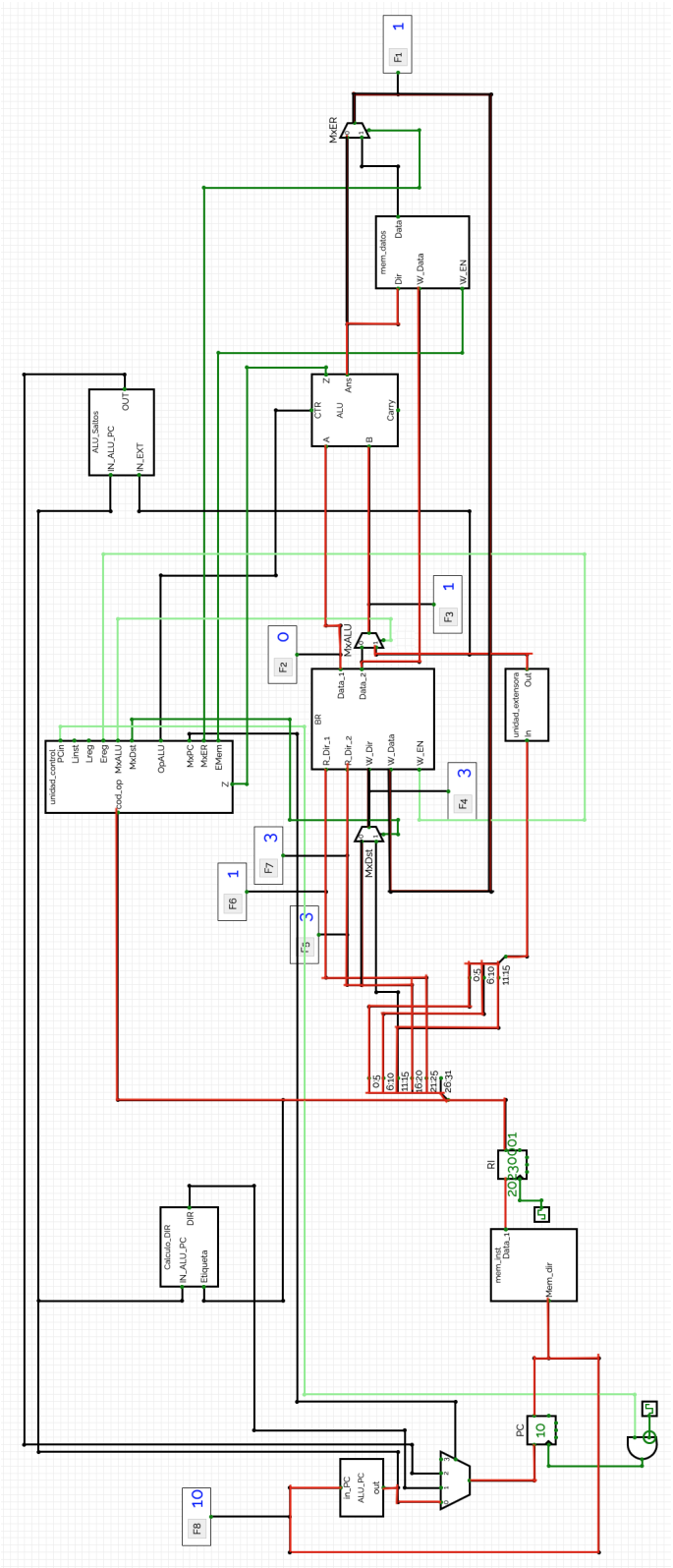

Figura 4.28: Ruta seguida por las instrucciones de almacenamiento. En rojo aparecen marcados los buses utilizados durante la ejecución de la instrucción. Las señales de control no están resaltadas dado que todas las señales de control se tienen en cuenta durante la ejecución de una instrucción, estando estos a 1 o a 0.

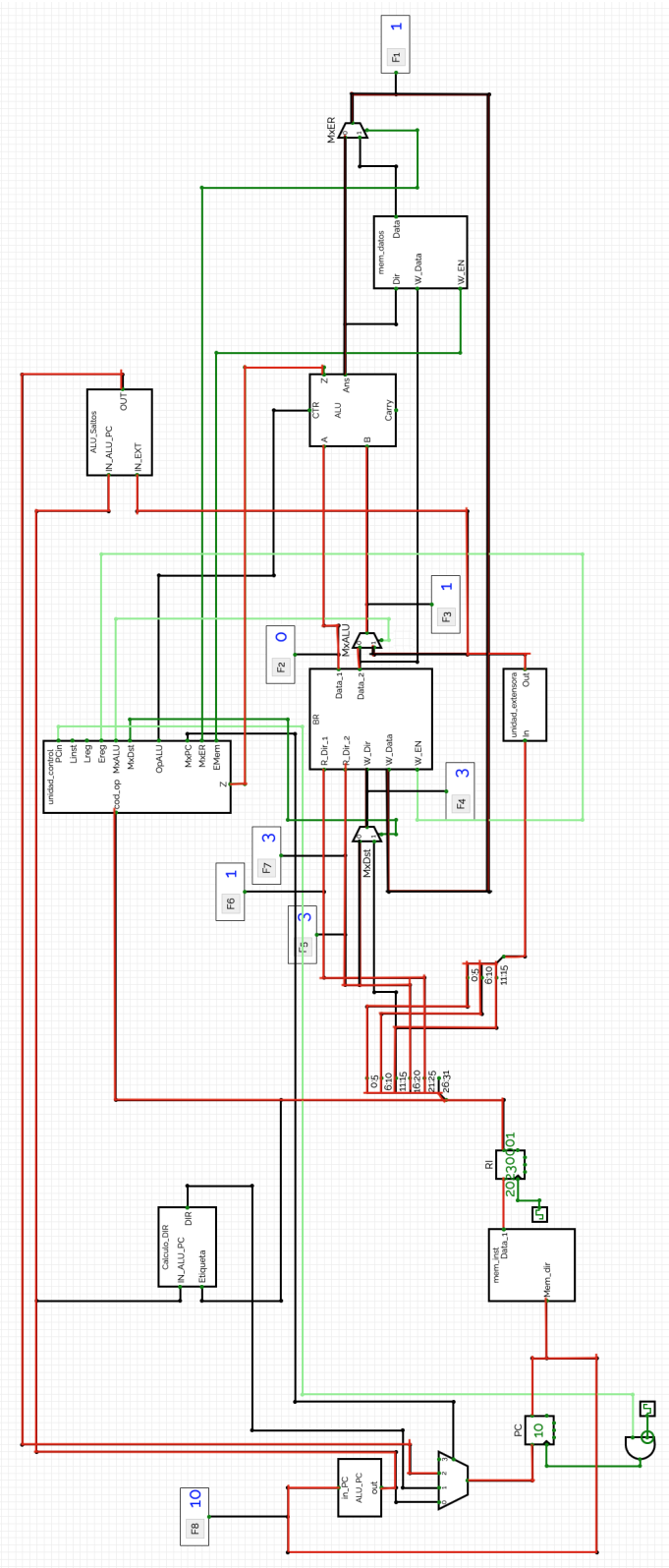

Figura 4.29: Ruta seguida por las instrucciones de salto condicional. En rojo aparecen marcados los buses utilizados durante la ejecución de la instrucción. Las señales de control no están resaltadas dado que todas las señales de control se tienen en cuenta durante la ejecución de una instrucción, estando estos a 1 o a 0.

En la Figura 4.29 se puede observar la ruta que sigue la instrucción y los elementos de la ruta de datos con los que llega a interactuar.

Para el cálculo de la dirección a la que se saltará se utilizan el valor calculado por la ALU del contador de programa, que posteriormente se envía a la unidad de cálculo de la dirección de salto, llamada para abreviar *ALU_Saltos*. Junto a este valor, esta unidad recibe desde la unidad extensora el desplazamiento que estaba almacenado en los 16 bits de menor peso de la propia instrucción. Utilizando estos dos elementos llega a calcular la dirección a la que saltar, que tras ello llega al multiplexor del contador de programa en su entrada 2.

Únicamente falta establecer si se cumple la condición. Para ello se leen los dos registros indicados por los campos RS y RT y se pasan a la unidad aritmético-lógica que los resta y establece si el resultado es equivalente o no a 0. Si esto es cierto se activa el bit Z que llega a la unidad de control y esta envía la señal correspondiente al multiplexor del contador de programa seleccionando la entrada 2 de este y almacenando la dirección calculada en el registro del contador de programa, para acceder a esta en el próximo ciclo de reloj.

4.3.3. Instrucciones tipo J

Las instrucciones de tipo J abarcan únicamente las instrucciones de salto incondicional. Estas instrucciones cumplen la misma función que las instrucciones de salto incondicional vistas anteriormente, pero en este caso, cambian el flujo de ejecución sin necesidad de cumplirse ninguna condición.

Aunque la función es la misma de las instrucciones de salto condicionales e incondicionales, estas últimas utilizan un método distinto para obtener la dirección de salto a partir de la información proporcionada en la instrucción. Por esto mismo utilizan un formato de instrucción distinto, observado en la Figura 4.30. El campo *objetivo* de la instrucción contiene los 26 bits de menor peso de la dirección a la que se saltará, los 4 bits de mayor peso que faltan se obtendrán de la dirección calculada en la ALU del contador de programa.

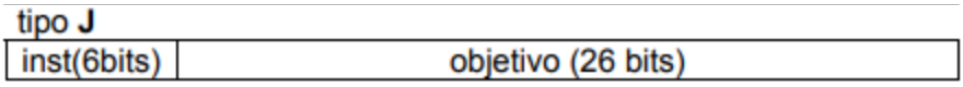

Figura 4.30: Formato de las instrucciones tipo J. En este formato de instrucción únicamente existen dos campos, el código de operación de 6 bits y otro campo, llamado *objetivo*, de 26 bits.

En la Figura 4.31 se puede observar el camino seguido por una instrucción de salto incondicional, es decir, de tipo J. Al contemplar la figura es claramente visible que gran parte de los componentes de la ruta de datos no se utilizan. Obviando los elementos utilizados durante la carga de la instrucción, como el contador de programa y la memoria de instrucciones, únicamente entran realmente en uso la unidad de control y la unidad para componer la dirección de salto incondicional, llamada en esta implementación para abreviar *Calculo_DIR*.

Tras la búsqueda y recuperación de la instrucción desde la memoria de instrucciones los bits de esta llegan entre otros componentes a la unidad para componer la dirección de salto incondicional, que por su parte calcula la dirección

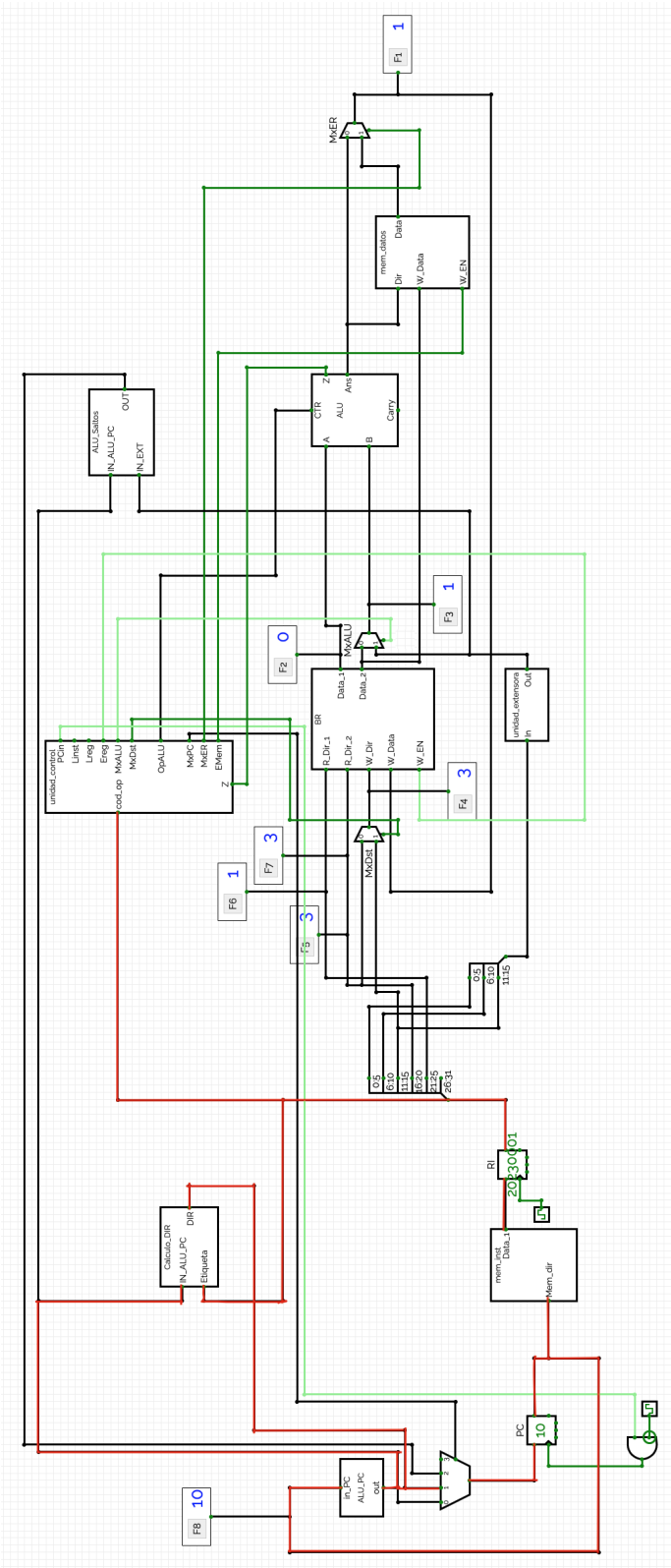

Figura 4.31: Ruta seguida por las instrucciones de tipo J, estas son, las instrucciones de salto incondicional. En rojo aparecen marcados los buses utilizados durante la ejecución de la instrucción. Las señales de control no están resaltadas dado que todas las señales de control se tienen en cuenta durante la ejecución de una instrucción, estando estos a 1 o a 0.

a la que se saltará y envía esta a la entrada 1 del multiplexor del contador de programa. Mientras tanto, la unidad de control decodifica la instrucción y tras determinar que se trata de una instrucción de salto incondicional envía la señal de control pertinente al multiplexor antes citado, seleccionando la entrada 1 de este, permitiendo que la dirección calculada se almacene en el registro del contador de programa. Terminando con esto la ejecución de la instrucción de salto incondicional.

Para finalizar el capítulo se deben destacar las instrucciones actualmente soportadas por la actual implementación del MIPS R2000 en CircuitVerse, estas se pueden observar en la Tabla 4.1. Las instrucciones soportadas abarcan los tres tipos de instrucciones existentes. Gracias a ello, existen todos los elementos necesarios para implementar el resto de instrucciones del lenguaje ensamblador del MIPS R2000 y únicamente se necesitaría modificar la unidad de control. Merced a las facilidades que ofrece CircuitVerse estas modificaciones únicamente consistirían en añadir las puertas AND y OR necesarias a las señales de control que son utilizadas para cada instrucción.

En esta implementación no se ha considerado el conjunto de instrucciones completo del MIPS R2000 porque se intenta fomentar la simplicidad de diseño y de esta manera impulsar la facilidad de comprensión de los componentes internos del procesador y su funcionamiento, por alumnos y personas interesadas en la ingeniería de computadores. La implementación del lenguaje ensamblador completo únicamente añadiría más elementos en la unidad de control, dificultando sobremanera su diseño y posterior comprensión por personas interesadas.

Instrucciones de tipo R
add
sub
and
or
slt
Instrucciones de tipo I
addi
slti
lw
sw
beq
Instrucciones de tipo J
j

Tabla 4.1: Instrucciones soportadas por la actual implementación del MIPS R2000 en CircuitVerse, separadas en grupos según el formato de instrucción que utilizan.

CAPÍTULO 5
Conclusiones

En este capítulo final se realiza una mirada recapituladora al trabajo desarrollado y se evalúan los objetivos que fueron establecidos al comienzo de este. Junto a esto, se plantean posibles expansiones y mejoras al proyecto realizado hasta el momento.

5.1 Observaciones finales

El objetivo principal del trabajo actual es presentar el simulador de circuitos lógicos CircuitVerse y demostrar su valía para la educación mediante el diseño e implementación en CircuitVerse del procesador más utilizado en la enseñanza de la ingeniería de computadores, el MIPS R2000, en su variante monociclo.Además como objetivo secundario, se pretende familiarizar a los lectores con la estructura interna de un procesador y su funcionamiento.

El segundo capítulo del documento está completamente dedicado a presentar el simulador CircuitVerse. Se presentan sus características generales y el entorno de trabajo que ofrece al usuario. Asimismo, se mencionan otros simuladores disponibles en el mercado y se hace una comparación del simulador utilizado en este trabajo a estos otros simuladores, destacando sus ventajas y desventajas respecto a estos durante el proceso.

Siendo el procesador MIPS R2000 la base de este proyecto se ha intentado familiarizar el lector con este procesador tanto en su variante monociclo como en su variante segmentada. Junto con la exposición sobre el propio procesador se introducen comentarios sobre el estado del arte existente en el mercado de los procesadores en la época de este y también del estado del mismo mercado en la actualidad. Además de esto se dan pequeñas pinceladas sobre las arquitecturas RISC y las arquitecturas CISC, conceptos importantes para entender la importancia del MIPS R2000 y su utilidad en la enseñanza.

Tras exponer tanto el procesador MIPS R2000 como el simulador CircuitVerse, es necesario pasar al diseño y posterior implementación del procesador en CircuitVerse. Se ha pretendido llevar a cabo esta tarea utilizando todas las facilidades ofrecidas por el simulador para evidenciar de esta forma las comodidades que ofrece a sus usuarios y con ello evaluar la facilidad con la que lo utilizaría un usuario novel, dado que una de las principales características de CircuitVerse es

sus sencillez y comodidad. Además, ya que se pretende utilizar el simulador en la educación como herramienta introductoria a la ingeniería de computadores , este debe ser adecuado para usuarios poco familiarizados con la circuitería digital. Se han utilizado los componentes de circuitos lógicos disponibles en CircuitVerse lo máximo posible, para examinar las funcionalidades ofrecidas por ellos y su utilidad. Asimismo, se han utilizado subcircuitos para abstraer dificultad de los circuitos y en algunas ocasiones se han construido de manera automática partes de los circuitos empleando tablas de verdad como base y siendo el propio simulador el que implemente un circuito que cumpla con la tabla de verdad establecida.

En todo momento se han utilizado todas las herramientas ofrecidas por CircuitVerse para depurar los circuitos de posibles errores y comprobar que ofrecen la funcionalidad deseada. Donde hay que destacar que, aunque las herramientas ofrecida sean sencillas y no muy numerosas, son todas muy útiles y fáciles de utilizar, permitiendo solucionar posibles errores muy pronto en la implementación.

Una vez implementada la ruta de datos en su totalidad, se ha diseñado un pequeño programa en código binario, que fue introducido en la memoria de instrucciones, que permita mostrar y comprobar el correcto funcionamiento del procesador implementado y simulado en CircuitVerse.

Mediante la implementación y simulación de un circuito complejo, como es la ruta de datos de un procesador, se ha pretendido demostrar que el simulador CircuitVerse es una herramienta potente a la vez que fácil de utilizar y aprender, que puede ser un instrumento muy útil para la enseñanza de ingeniería de computadores.

5.2 Trabajo futuro

El trabajo actual es posible expandirlo en varias líneas independientes. Por un lado, se puede ampliar el tamaño de las memorias de datos e instrucciones permitiendo almacenar y ejecutar programas más largos y más complejos. Por otro lado, se puede ampliar el juego de instrucciones soportado actualmente por el procesador simulado, llegando incluso a implementar el juego de instrucciones completo del procesador original.

Otra posible vía de expansión del trabajo es la implementación del procesador MIPS R2000 en su versión segmentada, donde la temporización de los elementos sería todavía más crítica.

Bibliografía

[1] Hamacher, V. Carl, Zaky, Safwat G., Vranesic, Zvonko G. *Computer Organization*. McGraw-Hill, quinta edición, 2002.

[2] Hennessy, John L., Patterson, David A. *Computer architecture: a quantitative approach*. Morgan Kaufmann, quinta edición, 2012.

[3] Malik, Norbert R. *Circuitos electrónicos: Análisis, simulación y diseño*. Prentice Hall, 1996.

[4] Manual del simulador CircuitVerse Consultado en https://docs.circuitverse.org/#/.

[5] Mateos, A. Calderón, Carballeira, F. García , Cepeda, J. Prieto . WepSIM: Simulador modular e interactivo de un procesador elemental para facilitar una visión integrada de la microprogramación y la programación en ensamblador. *Enseñanza y Aprendizaje de Ingeniería de Computadores*, Número 6, 2016.

[6] Ortega, J., Anguita M., Prieto, A. *Arquitectura de Computadores*. Thomson, 2005.

[7] Parhami, B. *Arquitectura de Computadores*. McGraw-Hill, 2007.

[8] Patterson, David A., Hennessy, John L. *Computer Organization and Design: the hardware/software interface*. Morgan Kaufmann, 2014.

[9] Stallings, William. *Computer Organization and Architecture: Designing for Performance*. Pearson, décima edición, 2016.

[10] Tanenbaum, Andrew S. *Organización de computadoras: un enfoque estructurado*. Pearson Educación, cuarta edición, 2000.

APÉNDICE A
Lenguaje ensamblador del MIPS R2000

En el apéndice actual se expone el conjunto de instrucciones del MIPS R2000 mediante dos figuras que conforman el cuadro resumen del lenguaje ensamblador del MIPS R2000, Figura A.1 contiene la primera parte del lenguaje ensamblador y la Figura A.2 recoge la segunda parte del lenguaje ensamblador. Junto a cada instrucción aparece una breve explicación de su funcionamiento, se expone su formato y su tipo. Además aparece un ejemplo de su uso para cada instrucción.

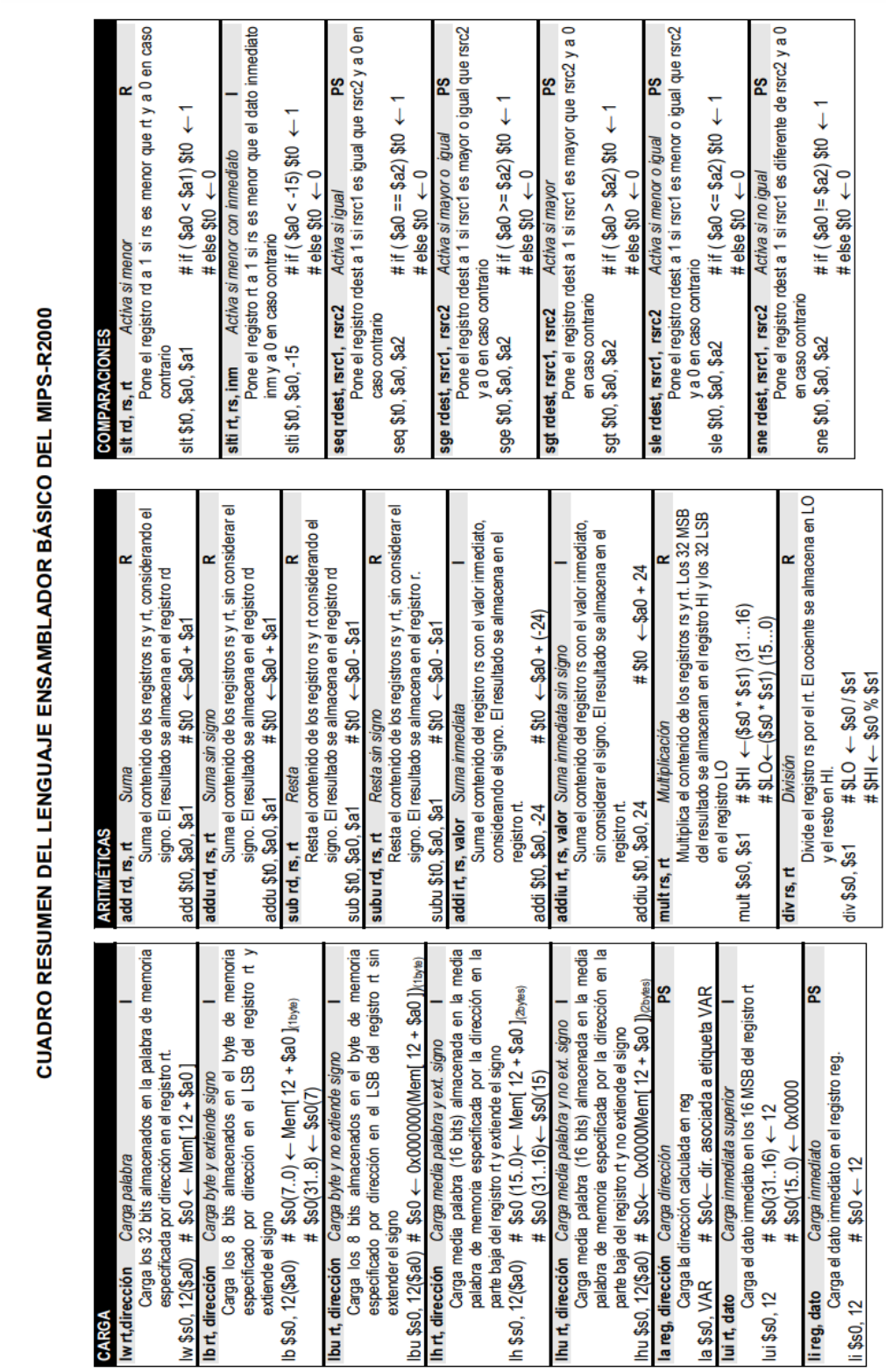

Figura A.1: Primera parte del conjunto de instrucciones del procesador MIPS R2000, donde podemos ver el conjunto de instrucciones aritméticas, de carga y de comparación. Figura obtenida del material de la asignatura Estructura de Computadores

Figura A.2: Segunda y última parte del conjunto de instrucciones del procesador MIPS R2000, donde podemos observar el conjunto de instrucciones de almacenamiento, lógicas, de movimiento entre registros, de desplazamiento, saltos incondicionales y por último el conjunto de instrucciones de saltos condicionales.

Figura obtenida del material de la asignatura Estructura de Computadores