UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Simulador web de la ruta de datos del procesador didáctico Easy 8

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Muñoz Navarro, Álvaro

Tutor/a: Martí Campoy, Antonio

CURSO ACADÉMICO: 2021/2022

Resumen

Este proyecto tiene como objetivo el desarrollo de una aplicación web para la simulación de una versión mejorada y más completa del computador Easy8.

El Easy8 es un computador didáctico que incluye una CPU simple de 8 bits que permite comprender el funcionamiento de la secuenciación de las instrucciones máquina.

El simulador está pensado para el uso de los alumnos de Fundamentos de Computadores del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación de la Universitat Politècnica de València para que puedan aprender cómo funciona la ruta de datos del procesador y puedan practicar el lenguaje ensamblador. Esto se ha tenido en cuenta a la hora del desarrollo e implementación del mismo.

Existen varios simuladores que han sido utilizados por los profesores de la asignatura durante los últimos cursos académicos. En este trabajo, se superan ciertas limitaciones de estos simuladores y además se utiliza una arquitectura ampliada del Easy8

Palabras clave: simulador, aplicación web, procesador, computador, Vue, ensamblador, registros, señales.

Contenido

Glosario

- **Ensamblador:** lenguaje de programación de bajo nivel.
- **Instrucción máquina:** operación elemental que puede ejecutar un procesador.
- **Secuenciación:** conjunto de señales de control de la ruta de datos que ejecutan una instrucción máquina determinada.
- **Circuito secuencial**: circuito cuyas salidas no solo dependen de sus entradas actuales, sino también de su estado actual, almacenada en elementos de memoria.
- **Circuito combinacional**: circuito en el que sus salidas son función exclusiva del valor de sus entradas en un momento dado.
- **CSS:** lenguaje de diseño gráfico.
- **HTML:** lenguaje de marcado para el desarrollo de páginas web.
- **JavaScript:** lenguaje de programación interpretado de alto nivel orientado a objetos para el desarrollo de páginas web interactivas.
- **Framework:** estructura conceptual y tecnológica que aporta la base y las herramientas necesarias para el desarrollo de software.
- **Node.js:** *framework* basado en JavaScript.
- **Vue.js:** *framework* de Vue centrado en *front-end.*
- **Front-end:** parte del software que interactúa con los usuarios.
- **HTTP:** protocolo de comunicación que permite la transferencia de información a través de archivos en la World Wide Web.
- **Heroku:** plataforma como servicio de computación en la nube que soporta distintos lenguajes de programación.
- **Git:** sistema de control de versiones distribuido.
- **XML:** lenguaje de marcado para codificar documentos.
- **SVG:** formato de imagen vectorial basado en XML.
- **JPG:** formato de archivo de imagen fotográfica

1. Introducción

El lenguaje ensamblador es una traducción casi completamente directa de las instrucciones máquina que un procesador puede ejecutar. La programación en lenguaje ensamblador permite acceder directamente y al más bajo nivel al hardware del procesador, necesario en el desarrollo de parte de algunas aplicaciones (sistemas operativos, manejadores, rutinas críticas, …). Pero también es necesario para comprender el funcionamiento del procesador. Por ello, el conocimiento de su funcionamiento por parte de los alumnos del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación de la Universitat Politècnica de València en la asignatura de Fundamentos de Computadores es imprescindible tanto para entender cómo funciona un sistema informático, así como para entender mejor las diferencias de un lenguaje de programación más simple y primitivo en contra de los lenguajes de alto nivel más modernos y por lo tanto comprender mejor estos últimos.

Para que el alumno pueda iniciarse en este tipo de máquinas sin enfrentar demasiadas complicaciones se ofrece el computador Easy8 capaz de trabajar con un conjunto de instrucciones máquina y registros sencillos y reducidos. La asignatura se imparte en el primer semestre del primer curso del GITST, y cuenta con 4,5 créditos, de los 0,6 son prácticas de laboratorio. Los objetivos incluyen la representación de la información en los computadores, y el estudio de las diferentes unidades funcionales que conforman un computador.
Uno de los objetivos de la asignatura es que los alumnos sean capaces de secuenciar instrucciones máquina sobre una ruta de datos de un procesador. Es decir, establecer los ciclos y las señales de control que deben activarse en cada ciclo para que la ruta de datos ejecute una instrucción máquina concreta. Con este objetivo se pretende que los estudiantes entiendan el funcionamiento interno de un computador.

Por lo tanto, este trabajo presenta como objetivo el desarrollo de una versión del simulador de la ruta de datos ampliada del computador Easy8 con una interfaz sencilla, donde el alumno pueda introducir las fases de una instrucción y verificar la ejecución de la instrucción. máquina paso a paso y ver cómo afecta a la ruta de datos y a la activación de los distintos componentes que la forman para poder usarlo como un refuerzo de la asignatura y a su vez poder realizar prácticas en él.

1.1 Motivación

El proyecto se ofertó por el DISCA para alumnos de la Rama de Ingeniería de Computadores y por los propios profesores de dicha rama para ser aprovechado como un recurso para sus clases.

Escogí este trabajo ya que me interesa tanto el desarrollo web como el lenguaje ensamblador. Aunque la rama que he estudiado es ingeniería de computadores he trabajado en desarrollo web por lo que me pareció interesante poder realizar un proyecto que trate sobre un computador en él que pueda utilizar y pulir conocimientos fuera de mi especialidad. Además, tiene un uso práctico real ya que será utilizado por los alumnos del grado y facilitará su formación.

1.2 Objetivos

El principal objetivo de este proyecto es el desarrollo de un simulador web del Easy8 que permita poder conocer la estructura básica de su ruta de datos, comprender la secuenciación de instrucciones mediante señales de control ciclo a ciclo sobre los distintos componentes de la ruta de datos, así como poder ejecutar y guardar programas mediante archivos de texto con los cuales se podrá explicar el funcionamiento de instrucciones máquina o se podrán plantear ejercicios al alumno.

Haciendo de este simulador una aplicación web conseguimos que el programa sea multiplataforma, sin necesidad de instalación, que sea accesible en cualquier momento y lugar y que en caso de que sea necesario pueda actualizarse inmediatamente.

En todo momento se tiene en cuenta que la aplicación sea lo más intuitiva posible para el alumno y tenga un uso fácil y directo.

1.3 Estructura

La memoria se estructura en 8 capítulos, siendo esta introducción el primero de ellos.

El capítulo 2, *Estado del Arte*, expondrá las aplicaciones similares a este proyecto ya existentes y lo que puede aportar este trabajo.

En el capítulo 3, *Análisis*, se especifican los requisitos con los que debe cumplir la aplicación.

El capítulo 4, *Diseño*, muestra la arquitectura de la aplicación y como afectará a la implementación de la misma.

En el capítulo 5, *Implementación*, se hará énfasis en las distintas tecnologías y herramientas utilizadas para llevar a cabo el proyecto y su influencia la estructura de la aplicación. Además, se explicarán los diferentes pasos por los que se ha desarrollado la aplicación y como las tecnologías utilizadas han influido en la toma de decisiones.

Por su parte, el capítulo 6, *Despliegue*, contará paso a paso como se realiza el despliegue de la aplicación.

El capítulo 7 *Pruebas*, consta, tal y como su nombre indica, de las pruebas realizadas para asegurar el correcto funcionamiento y visualización de la aplicación.

Por último, el capítulo 8, *Conclusión*, cerrará la memoria con las conclusiones generadas tras la finalización del proyecto.

2. Estado del arte

En este capítulo se va a realizar un breve análisis de las aplicaciones que se usan para simular el computador Easy8 y para simular la ruta de datos de dicho procesador respectivamente. El Easy8 es un computador didáctico que sigue la arquitectura von Neumann [1], es decir, cuenta con una unidad de procesamiento, una memoria principal, que es compartida por las instrucciones y los datos, y el sistema de entrada y salida. En la memoria del Trabajo de Fin de Grado de Rafael González Carrillo [2], donde se puede encontrar una descripción completa del mismo.

Para ayudar a los alumnos a comprender la estructura y arquitectura del procesador y computador Easy8, la asignatura dispone de dos simuladores. Un simulador del computador que puede programarse en lenguaje ensamblador del Easy8, y un simulador de la ruta de datos, que permite secuenciar las fases de ejecución de una instrucción máquina del procesador Easy8.

2.1 Simulador web del computador Easy8

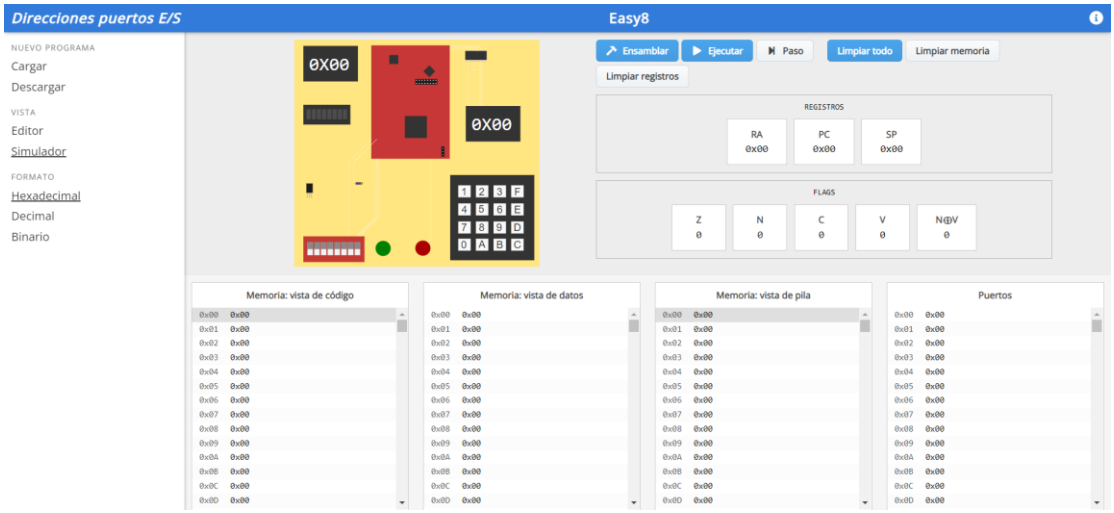

Figura 1: Simulador web del Easy8.

El Easy8 dispone de un simulador web[1], como se puede observar en la *Figura 1*, desarrollado por Rafael González Carrizo, citado y referenciado anteriormente, alumno del Grado en Ingeniería Informática, para su Trabajo de Fin de Grado. Este simulador es una mejora de otro anterior desarrollado por un profesor y es completamente funcional

[1] http://easy8.webs.upv.es/#/

pero no tiene opción a simular la ruta de datos, para ello existe el simulador que se comenta a continuación.

2.2 Simulador de la ruta de datos del Easy8

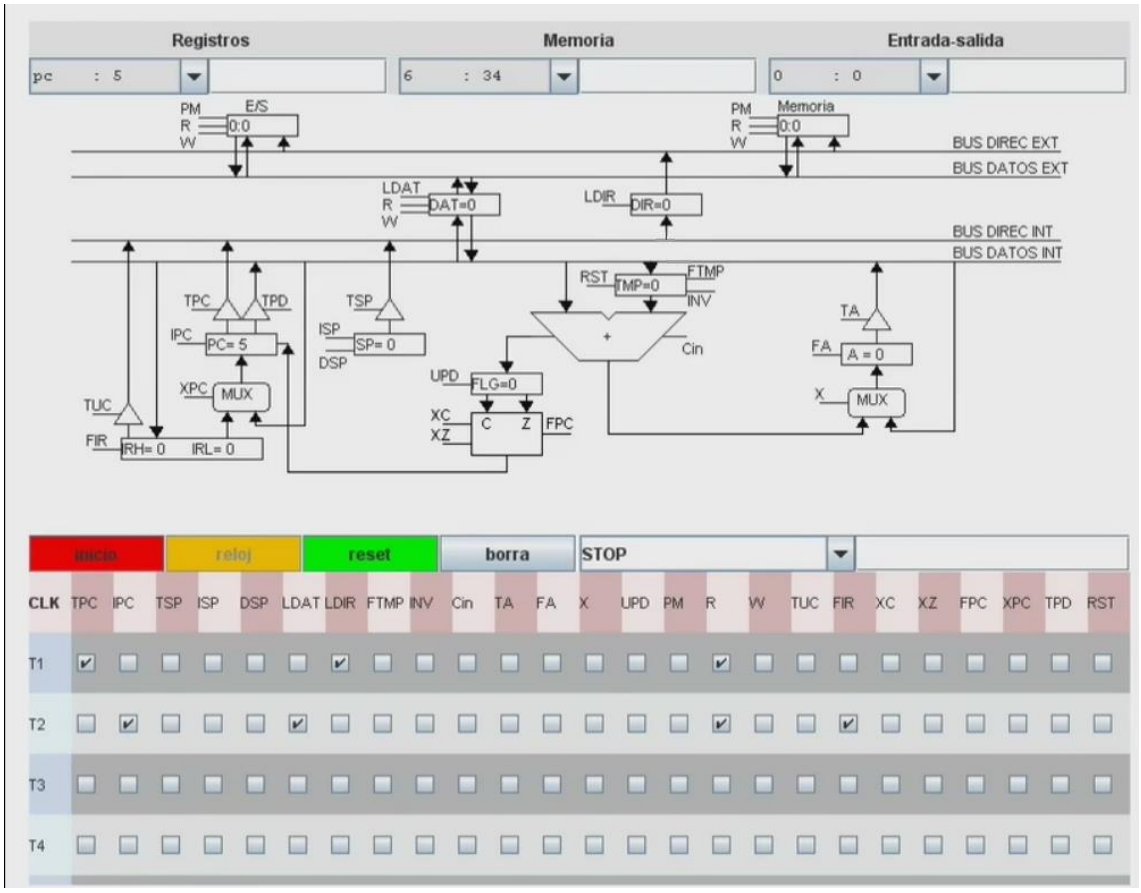

Figura 2: Simulador de la ruta de datos del Easy8.

El simulador actual de la ruta de datos del Easy8[2], tal y como puede apreciarse en la *Figura 2,* fue desarrollado por Alberto González Téllez, profesor del Departamento de Informática de Sistemas y Computadores, como un applet de Java. En él los estudiantes pueden establecer las señales de control que deben activarse en cada ciclo para ejecutar correctamente una instrucción máquina. Es decir, pueden diseñar la secuenciación de una instrucción. Este simulador tiene una serie de inconvenientes que se van a mejorar en este proyecto:

1. **No hay opción para cargar programas o guardarlos:** la aplicación solo proporciona la interfaz para usar manualmente el simulador, pero no es posible guardar la ejecución como un programa o cargar uno predeterminado o ejecutado anteriormente.
2. **Pocos ciclos de reloj:** la simulación solo permite 8 ciclos de reloj.

[2] https://labvirtual.webs.upv.es/e8cpu.htm

3. **Basado en Java:** es necesario tener Java instalado en el equipo y descargar el *applet*.
4. Su uso es poco intuitivo, y a veces presenta un funcionamiento no predecible.

2.3 Propuesta de proyecto

En la propuesta de este TFG se decidió no sólo mejorar el simulador de la ruta de datos, si no también mejorar el propio procesador. Es decir, añadir unidades funcionales para incrementar su juego de instrucciones y clarificar otras partes de la estructura.

Partiendo pues de las carencias citadas anteriormente y en las propuestas de los profesores, esté proyecto propone una versión web del simulador de la ruta de datos del Easy8 actualizada:

1. Se actualizará la ruta de datos para representar un procesador Easy8 con más unidades funcionales y más señales de control, así como se hará una simulación más clara e intuitiva.
2. Se permitirá una simulación con más ciclos de reloj para poder ejecutar instrucciones máquina más complejas.
3. La aplicación se usará desde un servidor web para evitar instalaciones y facilitar su acceso desde más dispositivos.
4. Se añadirá la opción de guardar un fichero con la secuenciación de la instrucción actual generada y la opción de cargar una simulación desde un fichero con la secuenciación de una instrucción escrita previamente.

Por lo tanto, la aplicación propuesta ofrece una simulación más útil y con más información sin hacerlo más complejo y haciendo una interfaz clara e intuitiva y también la posibilidad de simular secuenciaciones escritas de antemano y guardar secuenciaciones para futuras simulaciones; todo ello con el objetivo de facilitar y mejorar la formación de los alumnos de Fundamentos de Computadores.

2.4 La nueva ruta de datos del Easy8

La *Figura 3* muestra la nueva ruta de datos propuesta por los profesores de la asignatura y que se pretende utilizar cuando se dispongan de las herramientas de simulación.

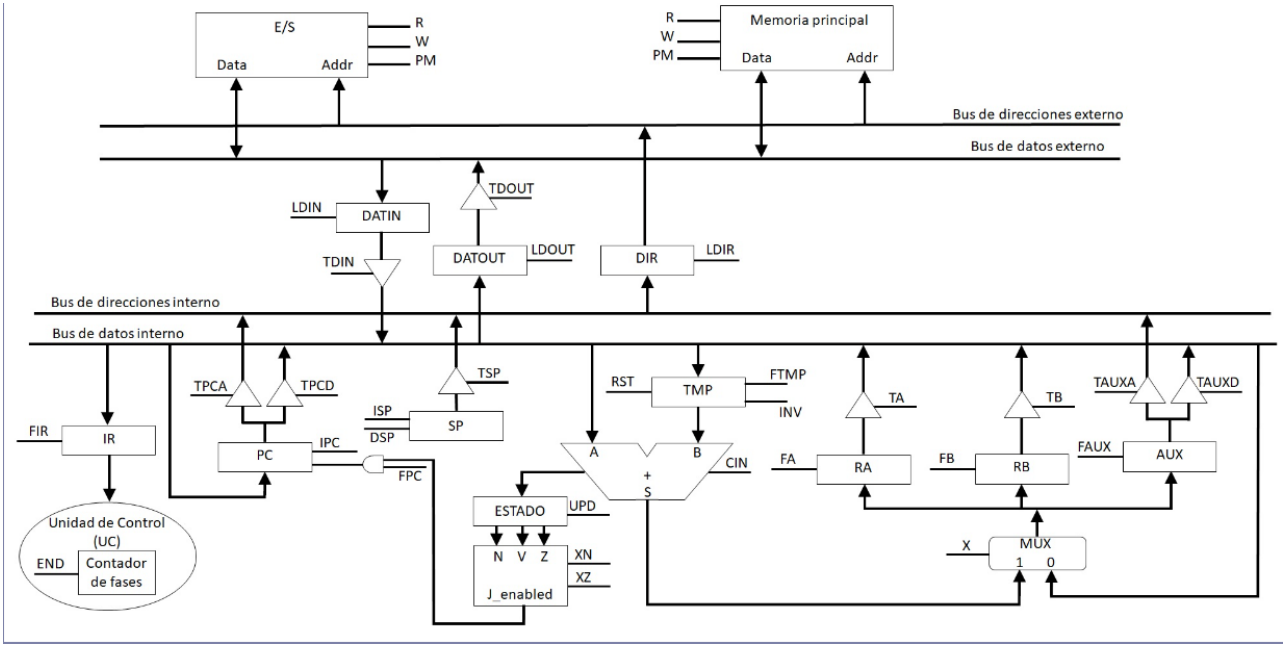

Figura 3: Nueva ruta de datos del Easy8

La figura 4 explica el funcionamiento de cada unidad funcional en relación a sus señales y a su comportamiento.

Unidad funcional	Descripción	Señales
Registro PC	Contador de programa	FPC: Escritura IPC: Incremento TPCA, TPCD: Lectura FPC: Indica instrucción de salto
Registro SP	Puntero de pila	DSP: Decremento ISP: Incremento TSP: Lectura
Registro A	Registro de propósito general activo por flanco	FA: Escritura TA: Lectura
Registro B	Registro de propósito general activo por flanco	FB: Escritura TB: Lectura
Registro AUX	Registro auxiliar no accesible por el programador activo por flanco.	FAUX: Escritura TAUXA, TAUXD: Lectura
Registro DATIN	Registro de interfaz con memoria y E/S. Activo por nivel.	LDIN: Escritura TDIN: Lectura

Registro DATOUT	Registro de interfaz con memoria y E/S. Activo por nivel.	LDOUT: Escritura LDIN: Lectura
Registro DIR	Registro de interfaz con memoria y E/S. Activo por nivel.	LDIR: Escritura
Registro TMP	Registro de almacenamiento de uno de los operandos en operaciones aritméticas	RST: Escribe un 0 FTMP: Escritura INV: Invierte la salida (Ca1)
Multiplexor	Selecciona la entrada entre la salida de la ALU o del bus de datos	X: Selección 1 o 0
Registro de estado	Contiene los indicadores de resultado de la ALU usados para los saltos condicionales.	UPD: habilitación de interrupciones XN: Indica instrucción JLESS XZ: Indica instrucción es JEQUAL
Registro IR	Registro de instrucción. Activo por flanco	FIR: Escritura
Unidad de control	Genera señales de control que gobiernan el funcionamiento del CPU	END: retorna al ciclo inicial de ejecución
Memoria principal y E/S	Memorias para el almacenamiento de instrucciones y datos, y puertos de entrada y salida	PM: selecciona entre memoria (0) o E/S (1) R: operación de lectura en dos ciclos W: operación de escritura en dos ciclos
ALU	Circuito que realiza operaciones matemáticas	CIN: carry inicial.

Figura 4: Unidades funcionales del Easy8

Dependiendo si el resultado de la operación de la ALU es negativo, cero o *overflow*, se activarán N, Z o V respectivamente. Teniendo en cuenta los tres *flags citados* y contando además con las señales XZ y XN se activará o desactivará la señal de salto *J_enabled* que controla la escritura en el registro PC. La lógica de dicha señal puede observarse en la *Figura 5*, donde se muestra parte de la tabla de verdad, en concreto los minitérminos.

XN	XZ	N xor V	Z	Salta (Out)
0	0	0	0	Si (1)
0	1	0	1	Si (1)
1	0	1	0	Si (1)
1	1	X	X	Si (1)

Figura 5: lógica de la señal de salto J_enabled. Se presenta parte de la tabla de verdad, mostrándose sólo las valoraciones para las cuáles la salida es uno.

3. Análisis

3.1 Introducción

En este capítulo se establecerán los requisitos del proyecto siguiendo y adaptando a este TFG el estándar IEE 830-1998.

3.1.1 Propósito

En este trabajo se va a implementar una aplicación web capaz de simular la ruta de datos del procesador Easy8 y de gestionar sus instrucciones máquina. Se espera que la aplicación ayude a los alumnos de Fundamentos de Computadores a comprender como funciona un computador y las instrucciones máquina.

3.1.2 Ámbito del sistema

El nombre de esta aplicación será Simulador de la ruta de datos del Easy8. El simulador será una aplicación web que no interactuará con otros sistemas informáticos

Constará de las siguientes funciones:

- **Simulación de la ruta de datos del Easy8:** ejecución ciclo a ciclo del circuito de unidades funcionales que forman el procesador Easy8.
- **Interfaz gráfica del simulador:** representación gráfica simbólica de los unidades funcionales del computador conectados entre sí que se irá actualizando ciclo a ciclo según se activen.
- **Gestión de ficheros:** gestor de archivos que permite al usuario guardar y cargar ficheros en una unidad de disco local.

La aplicación será completamente funcional en dispositivos cuya pantalla tenga una resolución de 1280 x720 o superior. Además, debe ser compatible con los navegadores más populares tales como: Chrome, Firefox, Opera o Microsoft Edge.

Con esta aplicación, se espera que alumnos con poca o nula experiencia en este campo tengan una primera toma de contacto con un procesador y sus componentes y puedan hacer todas las pruebas que necesiten para entender su funcionamiento. Por lo tanto, la aplicación tiene que ser lo más clara e intuitiva posible y debe proporcionar información útil en todo momento.

3.3 Requisitos específicos

3.3.1 Requisitos funcionales

Grupo funcional	Requisito funcional
Simulador estático	Visualización del hardware
	Visualización de flags
	Visualización del estado de los registros y memorias
	Visualización del ciclo actual
	Visualización de la ruta de datos
	Visualización del cronograma
Simulador activo	Destacar las unidades activas
	Calcular el estado de la máquina
	Creación del fichero de secuenciación.
	Lectura del fichero de secuenciación.
Usuario	Descarga del fichero de secuenciación.
	Carga del fichero de código
	Activar/desactivar las señales
	Modificar registros
	Limpieza de filas activas del cronograma
	Reinicio del ciclo
	Cambio al siguiente ciclo

Figura 6: tabla que agrupa los diferentes grupos funcionales

3.3.1.1 Requisitos del simulador estático

- RF01: Visualización del hardware: El sistema debe mostrar en todo momento de la simulación todos los componentes que forman la ruta de datos del procesador Easy8 y también la memoria principal y la entrada/salida del computador Easy8.

-RF02: Visualización de *flags:* El sistema debe mostrar en todo momento de la simulación si uno o varias *flags* están activos o no según una lógica determinada.

-RF03: Visualización del estado de los registros y memorias: El sistema debe mostrar en todo momento de la simulación el estado de cada uno de los registros, memorias y demás componentes se utilicen o no.

-RF04: Visualización de la unidad de control: El sistema debe mostrar en todo momento de la simulación en que ciclo se encuentra la máquina.

-RF05: Visualización del cronograma: El sistema debe mostrar el cronograma de las señales que controlan el simulador diferenciadas por ciclo.

3.3.1.2 Requisitos del simulador activo

- RF06: Destacar las unidades: El sistema de la aplicación debe mostrar en todo momento de la simulación las unidades lógicas que estén activas en el ciclo actual.

-**RF07: Calcular el estado de la máquina** El sistema debe calcular el estado de la máquina en cada uno de sus componentes para el ciclo actual. En concreto, se mostrará el estado de la máquina antes de ejecutar el ciclo actual.

-**RF08: Creación del fichero de secuenciación:** El sistema debe crear un fichero donde quede registrado la lista actual de señales activas seleccionadas por el usuario y almacenarlo en una unidad de disco.

-**RF09: Lectura del fichero de secuenciación:** El sistema debe ser capaz de leer un fichero con la lista de señales que deben activarse y mostrarlas en el cronograma de la aplicación.

3.3.1.3 Requisitos de usuario

- RF10: Descarga del fichero de secuenciación: El usuario debe ser capaz de descargar el fichero con la lista de señales activas en su máquina local.

-**RF11: Carga del fichero de secuenciación:** El usuario debe ser capaz de cargar un fichero de configuración desde su máquina local.

-**RF12: Activar/desactivar las señales:** El usuario debe ser capaz de activar o desactivar las señales de control de la ruta de datos en cada ciclo de reloj.

-**RF13: Modificar registros:** El usuario debe ser capaz de cargar un valor en los registros PC, SP y en cada una de las posiciones de la memoria principal y del sistema de entrada/salida.

-**RF14: Limpieza de filas activas del cronograma:** El usuario debe ser capaz de desactivar de una vez todas las señales activas de una fila (ciclo) del cronograma.

-**RF15: Avance de ciclo:** El usuario debe ser capaz de cambiar al siguiente ciclo de la simulación.

-**RF16: Reinicio del ciclo:** El usuario debe ser capaz de volver al ciclo inicial sin perder el estado de la máquina.

3.3.2 Requisitos no funcionales

- **Tiempo rápido de respuesta:**
  - En la carga inicial de la aplicación
  - En la carga de un fichero
  - En el ensamblaje del código de un fichero
  - En la creación y descarga de un fichero
  - En cada iteración del simulador
  - En la actualización del estado de los registros, memorias y *flags*
  - En la activación o desactivación de cada componente
- **Seguridad:**
  - El servidor evitará que se accedan a recursos que deberían ser inaccesibles.
  - La web será accesible a través de HTTPS
- **Facilidad de uso:**
  - No se requerirá ningún tipo de descarga o instalación
  - No se requerirá ningún tipo de registro de usuarios
- **Soporte**:
  - Compatible con versiones recientes de los navegadores más populares
  - Buena visualización en dispositivos con pantallas de al menos 1280x720
- **Desarrollo:**
  - Uso de software libre
  - Fácil de desplegar y mantener
  - Fácil de actualizar

4. Diseño

4.1 Arquitectura

La aplicación sigue una arquitectura en 2 niveles. Esta se utiliza para describir los sistemas cliente/servidor en donde el cliente solicita recursos y el servidor responde directamente a la solicitud, con sus propios recursos. Esto significa que el servidor no requiere otra aplicación para proporcionar parte del servicio.

Este tipo de arquitectura permite a la aplicación ser lanzada desde cualquier dispositivo con conexión a internet sin necesidad de instalar ningún complemento y con un tiempo rápido de respuesta.

Además, la aplicación cumple también con la arquitectura de 3 capas, como se puede ver en la *Figura 7*, siendo éstas presentación, negocio y datos [3]. En este caso, las tres capas se ejecutan en el cliente y el servidor se usa como repositorio para dar acceso a la aplicación. Como se ha expuesto en el capítulo de *Análisis* en el apartado de los requisitos, no hay registro de usuarios ni ningún tipo de almacenaje de la información para facilitar el uso y la accesibilidad de la aplicación.

Empezando por la capa de presentación, esta consta de la interfaz de la aplicación, todo lo que el usuario puede ver e interactuar en su navegador web. En lo que a la capa de datos se refiere, es el propio estado de la máquina en cada ciclo, todos sus componentes secuenciales (registros, señales, memorias). Por último, la capa de negocio se compone de todas las funciones, que, dado el estado actual de la máquina (procesador), calculan el estado siguiente, tales como las funciones de aritmética o de escritura en registros.

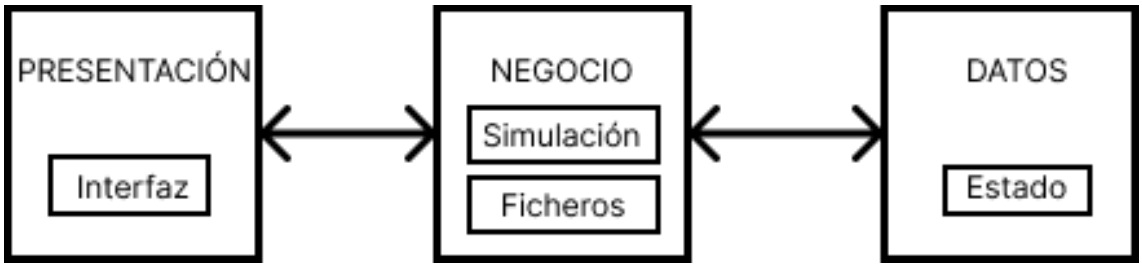

Figura 7: Arquitectura de 3 capas

4.2 Activación de las unidades funcionales

En lo que a la parte gráfica y visual del simulador se refiere, se han seguido unos códigos de colores para diferenciar cada parte del simulador. Para ello se han utilizado el color azul para los componentes que manejan los datos, verde para los que manejan las direcciones y rojo para componentes más específicos o que gestionan ambos.

La imagen se ha creado usando SVG, que está detallado más adelante en la memoria en la sección de *Implementación*, evitando usar imágenes mapas de bits, pues SVG aporta más flexibilidad.

4.3 Entrada del cronograma

Pese a que se plantearon varias opciones para que el estudiante pueda introducir las señales de control, como por ejemplo el uso de un botón por señal o de una lista de nombres, se ha considerado que la mejor opción era mantener la tabla tal y cómo se utiliza en el simulador anterior, pues permite un uso más intuitivo y que se pueda distinguir mejor cada ciclo de reloj.

4.4 Modificación de los elementos secuenciales

Los elementos secuenciales de la ruta pueden modificarse de dos formas diferentes.

La primera, usando un botón llamado ciclo, que ejecuta la activación de las señales de control seleccionadas en el cronograma. Simula la llegada de un flanco activo en la entrada de reloj de la unidad de control.

Por otro lado, para que el alumno pueda modificar de forma asíncrona algunos de los elementos secuenciales se dispone de una serie de *inputs* y botones con los que puede modificar el contenido de la memoria y la entrada/salida, cargar y generar ficheros y establecer el valor inicial de los registros PC y SP.

4.5 Formato de los ficheros de secuenciación

Los ficheros para almacenar y cargar las secuencias de señales están en formato *txt* y se estructuran de la siguiente manera: primero el ciclo en el que se activaran las señales seguido de dos puntos, y a continuación una lista de las señales que se quieren activar separadas por una coma. Cada ciclo se separa de su sucesor y su predecesor mediante un salto de línea. Se ha elegido este formato para que los estudiantes puedan abrir un fichero y entenderlo e incluso modificarlo.

Un ejemplo de este formato sería:

T1: TPCD, X, FA

T2: TA, LDIR

4.6 Optimización de la interfaz

Mediante el uso de *HTML* y estilos *CSS* se han ordenado los distintos elementos y se les ha cambiado el color y la forma para hacer una distribución y un diseño más atractivo e intuitivo de usar. La aplicación se distribuye en torno a la ruta de datos del procesador, puesto que es el elemento más importante, y alrededor de él se encuentran el resto de botones e *inputs* que permiten interactuar con él o con las demás funcionalidades. Por ello, se han agrupado los elementos en varios grupos funcionales.

4.6 Metodología

La metodología de desarrollo de software a seguir en este proyecto es la denominada desarrollo en cascada [4], llamada así porque separa el proceso del desarrollo del software en etapas de tal forma que una etapa no puede empezar sin haber finalizado la anterior.

En el caso de este proyecto, las etapas a distinguir, tal y como vemos en la *Figura 8,* son: análisis de requisitos, diseño del sistema, implementación, pruebas, despliegue y mantenimiento.

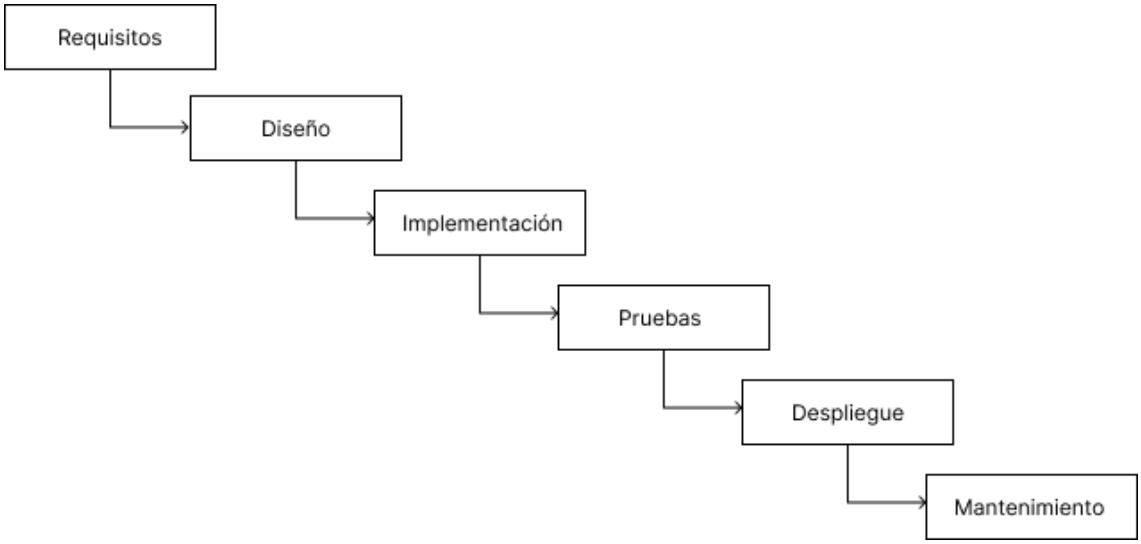

Figura 8: Metodología de desarrollo en cascada

Se ha decidido seguir este modelo dado que es fácil de interpretar y es y es adecuado cuando hay un único desarrollador en el proyecto, además de que el objetivo y requisitos del proyecto estaban claramente definidos desde el primer momento, por lo que se podía planificar sin riesgo a tener que cambiarlos a posteriori.

5. Implementación

En este capítulo de la memoria se van a explicar las diferentes herramientas utilizadas para implementar la aplicación y como está estructurada.

5.1 Herramientas

En este apartado se van a exponer las diferentes herramientas utilizadas en el desarrollo del proyecto. El sistema operativo utilizado ha sido Windows 10.

5.1.1 Front-end

Vue.js[3] es un *framework* de código abierto que utiliza las tecnologías comunes de desarrollo web tales como JavaScript, HTML y CSS [5]. La propia naturaleza del simulador nos obliga a actualizar frecuentemente la interfaz de la aplicación por lo que el uso de Vue.js facilita mucho la realización de este proyecto.

Vue presenta una serie de directivas propias, que se explican más adelante con más profundidad, que pueden usarse en la parte HTML del proyecto para utilizar la característica llamada *data binding*, entre otros usos. Esta característica permite vincular la vista de la aplicación con los datos del modelo, lo cual ahorra mucha cantidad de código y ayuda al correcto funcionamiento del simulador.

Además, este *framework* presenta otra gran característica propia que ha ayudado mucho a programar el simulador y es la definición de componentes *single-file* que están formados por tres partes: *template* (parte HTML), *script* (parte JavaScript) y *style (parte CSS)*. Este tipo de componentes pueden ser exportados y reutilizados por otros componentes del mismo tipo.

Un proyecto en Vue es compatible con el paquete de modulos *npm* de Node.js [6] por lo que es rápido y automático compilar y ejecutar el proyecto en un servidor local y empaquetarlo para su posterior despliegue, por lo que facilita significativamente el desarrollo del proyecto.

[3] https://vuejs.org/

5.1.2 SVG

SVG, del inglés: *Scalable Vector Graphics*, es un formato de gráficos vectoriales bidimensionales compuesto por código abierto en formato XML y que permite crear imágenes tanto estáticas como animadas. Como su propio nombre indica las imágenes se forman por vectores en vez de por mapa de bits por lo que son escalables de manera infinita sin perder resolución. Además, son compatibles con los navegadores más populares sin necesidad de extensiones y con código HTML, CSS y JavaScript. Gracias a estas compatibilidades se puede integrar en este proyecto sin ningún problema y se pueden alterar vía código según sea necesario sin tener que cambiar todo el diseño o crear uno nuevo.

Para crear este tipo de imágenes se ha usado la plataforma online *Figma*[4], que permite crear una imagen, separarla por elementos y exportarla a SVG.

5.1.3 Gestión de versiones del proyecto

Git[5] [7] es un sistema de control de versiones distribuido que permite tener un clon local de un proyecto. Se ha utilizado desde el inicio del desarrollo del proyecto para gestionar las distintas versiones que ha tenido la aplicación hasta llegar a la final. Su uso ayuda mucho también en caso de que se cometa un error y se tenga que volver a una versión anterior, por lo que permite probar varias opciones sin riesgo a comprometer todo el proyecto.

5.1.4 Servidor

Puesto que uno de los principales objetivos de este proyecto es que la aplicación se pueda usar en cualquier dispositivo sin necesidad de ninguna instalación la aplicación se ha desplegado en un servidor web para que los usuarios la ejecuten directamente desde un navegador.

Para ello, se ha elegido la plataforma online como servicio *Heroku*[6] [8] que soporta distintos lenguajes de programación y automatiza el despliegue de la aplicación. *Heroku*, que se explica en el capítulo de *Despliegue* con más detalle, utiliza *Git*, explicado en el apartado anterior, para recompilar el código y determinar que software necesita instalar en el servidor.

[4] https://www.figma.com/
[5] https://git-scm.com/
[6] https://www.heroku.com/

5.2 Estructura de directorios

Se va a explicar la estructura de directorios del proyecto, así como sus ficheros más importantes. Dicha estructura viene dada por Vue.js y puede consultarse en la documentación oficial proporcionada en su página web.[7]

5.1.1 Carpeta src

Esta es la carpeta más importante del proyecto pues es donde se encuentra todo el código de la aplicación. Podemos diferenciar los siguientes subdirectorios y ficheros:

- **App.vue:** Este fichero es el componente principal de la aplicación pues en él, aprovechando la estructura de componentes *single-file* de Vue, queda definida la parte HTML y CSS del proyecto; así como todos los métodos que se encargan del buen funcionamiento del simulador y del sistema de ficheros.
- **main.js**: Es el fichero encargado de importar Vue al proyecto y de permitir a este montar la aplicación para su ejecución usando el componente *App.vue*.
- **Carpeta components**: en ella se localizan otros componentes necesarios para el funcionamiento de la aplicación y que son importados por el componente principal *App.vue*. En este proyecto encontramos *Simulador.vue* que contiene la imagen SVG donde queda definido la parte visual del simulador así como la declaración de todos los componentes electrónicos (registros, memorias, buses, ALU, multiplexores) del mismo.
- **Carpeta assets**: En esta carpeta podemos encontrar recursos adicionales como imágenes en formato JPG para poder ser importadas en los componentes Vue.

5.1.2 Carpeta node_modules

Aquí se encuentran todos los módulos y librerías que permiten el funcionamiento de la aplicación. Entre ellos podemos encontrar por ejemplo módulos comunes como los de CSS, HTML, DNS, JSON, JavaScript; y también los definidos por Vue.js y Node.js.

5.1.3 Carpeta dist

Esta carpeta generada por Node.js contiene el proyecto ¿compilado y? optimizado y todo lo necesario para su despliegue.

5.1.4 Server.js

Contiene las directivas y los puertos necesarios para que *Heroku* sepa que tiene que utilizar la carpeta dist para montar la aplicación aprovechando los módulos disponibles en Node.js.

[7] https://vuejs.org/guide/essentials/application.html#app-configurations

5.1.5 Package.json

Este fichero contiene los scripts con las instrucciones que se deben ejecutar para desplegar la aplicación, tanto en el servidor como en local, así como los módulos o aplicaciones necesarias para su funcionamiento y en que versión se encuentran.

5.3. Desarrollo

A partir de lo establecido en los apartados anteriores, se va a explicar cómo se ha desarrollado la aplicación y cuál ha sido el resultado final.

5.3.1 Creación del proyecto

Puesto que el proyecto está planteado como una aplicación web el primer paso fue buscar un *framework* que nos permitiera hacerlo. La elección se decantó por Vue debido a las ventajas explicadas anteriormente y a la gran escalabilidad que éste posee en caso de que en un futuro los profesores quieran ampliarlo o se quede obsoleto. Tras instalarlo y crear un nuevo proyecto, con su *App.vue* y su *main.js*, se determinó que el objetivo a priorizar sería la simulación correcta del procesador y el apartado visual del simulador.

5.3.2 Creación del apartado visual del simulador

Se decidió que el apartado visual estaría formado por una imagen SVG que esté formada por la ruta de datos del procesador y por la representación de la activación de cada componente en rojo, de tal forma que estén separados por un grupo funcional. Usando la plataforma online *Figma*, explicada en apartados anteriores, se ha creado la imagen tal y como se muestra en la *Figura 9*.

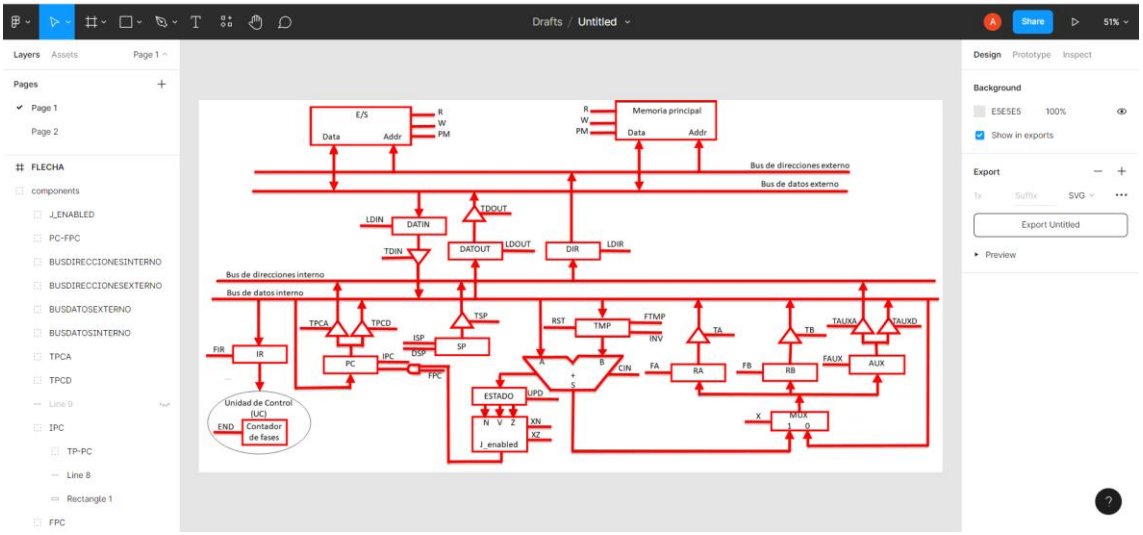

Figura 9: Diseño de la imagen SVG del simulador de la ruta de datos mediante Figma.

5.3.3 La clase Simulador.vue

Utilizando los componentes Single-File de Vue se creó esta clase donde se incluye la imagen SVG creada anteriormente. Para ello se integra el código SVG a la parte HTML de la clase y se define en la parte de JavaScript cada señal de control como un *Boolean*, con los que, gracias a las directivas Vue se puede usar un condicional que vuelva visible o no cada componente de la imagen SVG, así como importarlos a la clase principal, como se explica más adelante.

Gracias al código SVG se pudieron cambiar algunos atributos como el color para diferenciar entre datos, direcciones y otros componentes entre otras cosas que se fueron mejorando para el diseño final y para una interfaz más comprensible. Como se puede observar en la *Figura 6*, los buses de datos y sus unidades funcionales se representan en azul. Lo mismo ocurre para los buses de direcciones, pero en verde y, por último, las señales y demás componentes, como la ALU, en rojo.

Además, como se observa en la *Figura 10*, se establecen del mismo modo que las señales otras variables que contendrán el valor de cada registro del circuito y que son exportadas de la misma manera. El valor de estas variables se imprime en la parte HTML para que puedan visualizarse en el simulador.

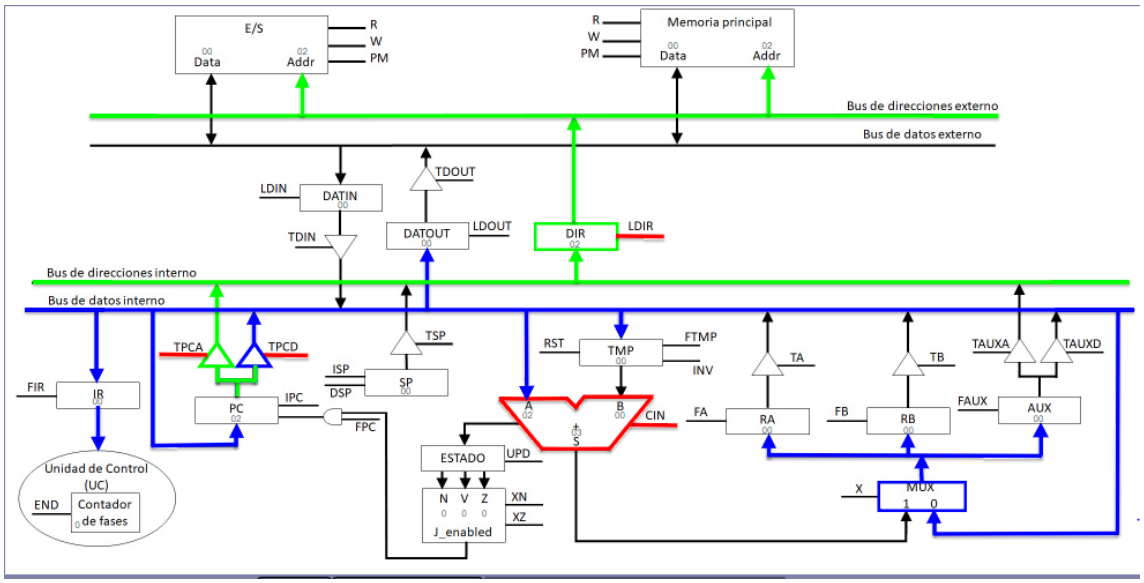

Figura 10, demostración de los colores del simulador y del valor de los registros.

5.3.4 La clase App.vue

Es la clase principal de la aplicación donde se encuentra el CSS, el HTML, los métodos que gestionan tanto el funcionamiento del simulador como de los ficheros y la importación del componente *Simulador.vue.* Como es un componente *single-file* está dividida en tres partes tal y como está explicado en el apartado de implementación de esta memoria.

Empezando por la parte *template* encontramos que está formada por una tabla donde se podrá activar y/o desactivar cada componente del simulador en cada ciclo de reloj. Cada *checkbox* de la tabla devolverá un *Boolean* que será asignado, mediante el *data binding* de Vue, a las señales definidas en la clase *Simulador.vue,* que habíamos exportado anteriormente. Con esto se consigue que se puedan hacer visibles o no en la imagen SVG del simulador, haciendo el efecto de activarse o desactivarse cada componente.

Además, cuenta con los botones e *inputs* necesarios para cambiar de ciclo, inicializar las señales de PC y SP y los contenidos de la memoria y la entrada/salida, y para generar o cargar un fichero de secuenciación.

En la parte *script* encontramos todos los métodos que hacen posible el cambio de valor, en hexadecimal, de cada registro y de la memoria, así como el funcionamiento de la ALU y de los *flags.* Estos métodos aprovechan los *Booleans* que la tabla genera para cada señal en cada ciclo. A partir de ellos, cada unidad funcional comprueba si hay alguna puerta lógica activa enviando un valor a sus respectivos buses y si su señal está activada para reescribir su registro. En el caso de la ALU se define también un método que calcula la suma de sus dos entradas y comprueba que *flags* deben activarse. La memoria y entrada/salida comprueban si tienen que realizar una operación de lectura o escritura con los valores proporcionados por los buses. El contenido de todos los registros se muestra en hexadecimal, y en el caso de la ALU, los operandos se consideran representados en complemento a dos, por lo que, en cada método donde sea necesario, se realizan conversiones a este tipo. Cómo este formato tiene un rango de -127 hasta 128 para números de 8 bits, si se obtiene un resultado en la ALU fuera del mismo se activará el *flag* de *overflow.*

El resultado retornado por estos métodos es asignado a las variables de registro que se han exportado de igual manera de la clase *Simulador.vue* mediante *data binding,* donde se cambiará el valor, si se debe, y se imprimirá en el apartado gráfico del simulador.

También encontramos otros métodos para el cambio de ciclo y para la gestión de ficheros. Estos últimos aprovechan los módulos proporcionados por JavaScript y interpretan si cada casilla de la tabla está seleccionada o no.

Se ha realizado el siguiente diagrama de flujo en base a lo que se acaba de explicar, *Figura 11*.

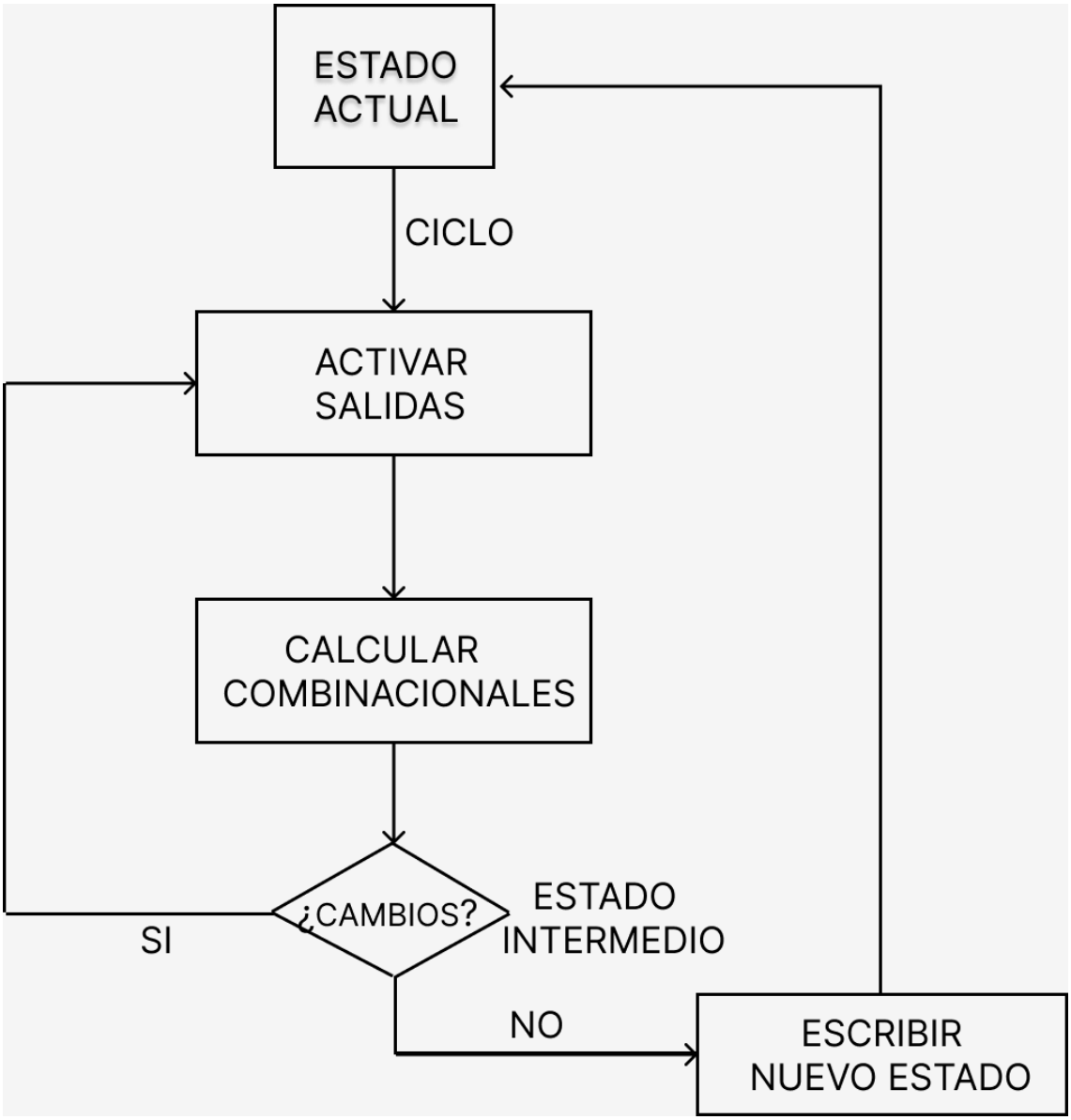

Figura 11, diagrama de flujo

5.3.5 Diseño gráfico final de la aplicación

Una vez se ha logrado el funcionamiento correcto de la aplicación, se utiliza la parte de *style* de la clase principal *App* para mejorar el diseño de la página. Se puede observar el diseño final en la *Figura 12.*

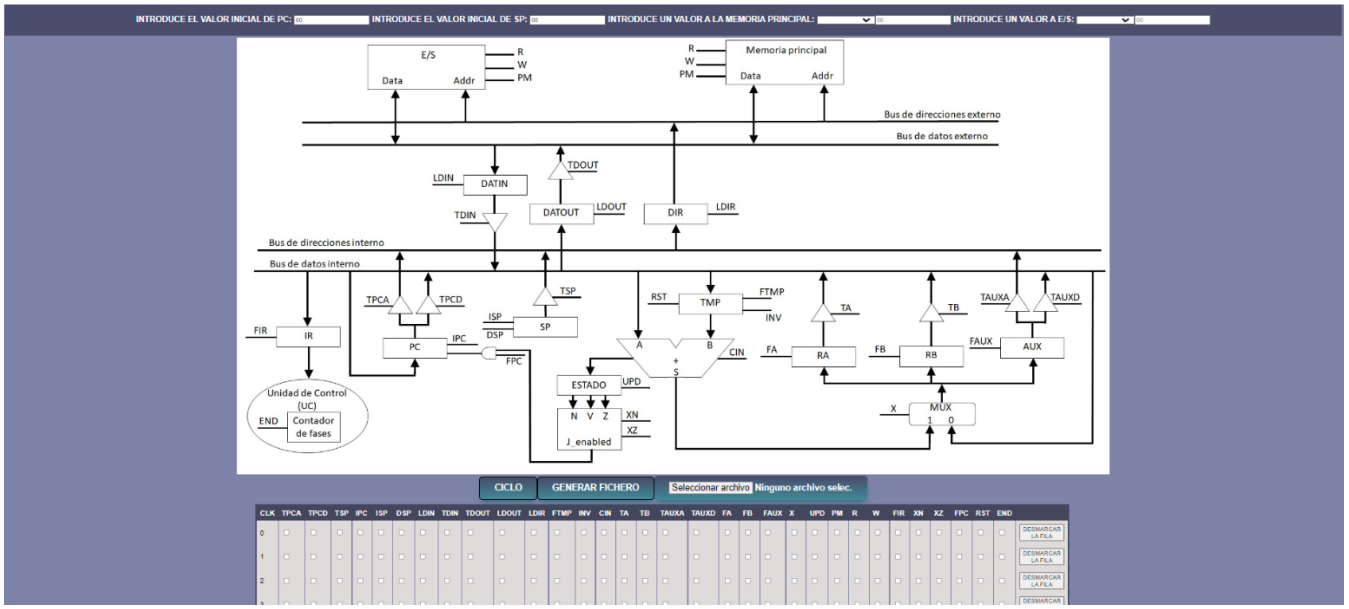

Figura 12: diseño final de la aplicación

Se puede observar que la aplicación se divide en tres partes: la inicialización de los registros, la visualización gráfica de la ruta de datos y la tabla o cronograma para elegir que señales se activan por ciclo, así como los botones para pasar al siguiente y para generar o subir un fichero. Estos grupos de elementos se pueden diferenciar a primera vista gracias al color de cada uno. Se ha decidido que la gama de colores quede entre azules oscuros y grises para que no sobresature la vista y resaltar más los colores del simulador previamente expuestos.

6. Despliegue

En este capítulo se explica cómo se ha realizado el despliegue de la aplicación en un servidor web que, como se ha mencionado en capítulos anteriores, ha sido *Heroku*, para conseguir que la aplicación sea accesible desde cualquier dispositivo compatible sin necesidad de instalación alguna.

Para poder realizar el proceso del despliegue es necesario tener cuenta en *Heroku*, la gratuita ofrece funciones suficientes para ello, y las herramientas *Heroku CLI y Git* instaladas en nuestro ordenador.

6.1 Crear una nueva aplicación en Heroku

Tras iniciar sesión, se hace click en *Create new App (Figura 13).* Tras ello, se nombra la aplicación creada y se selecciona la región donde se ubicará el servidor, siendo Europa nuestro caso.

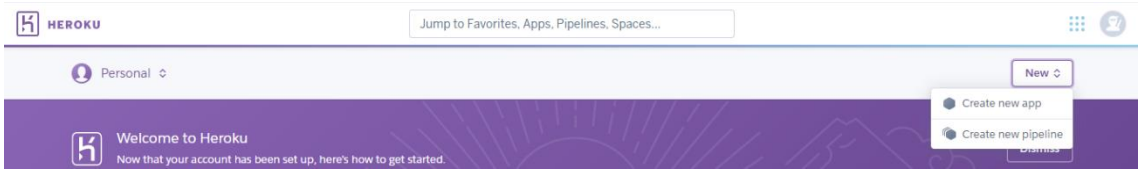

Figura 13, creación de un nuevo proyecto en Heroku

6.2 Clonar el repositorio mediante Git

En el segundo paso es necesario vincular el proyecto local al proyecto que se acaba de crear en *Heroku*. Para ello tendremos que utilizar *Git*, por lo que mediante una consola hay que situarse en la ubicación del proyecto local y acceder a *Heroku* mediante `heroku login` y a *Git* mediante `git init`. Una vez tengamos acceso a ambas herramientas utilizaremos el comando `heroku git:remote -a simulador-easy8` con el cual clonaremos el repositorio local al remoto de *Heroku*. En este caso usamos simulador-easy8 en el comando, pero se reemplazará por el nombre que se haya dado a la aplicación remota en el paso anterior.

6.3 Adaptar *Heroku* a nuestra arquitectura

Quizás en otras aplicaciones se podría subir ya a *Heroku* el código de la aplicación y dejarle que la monte y la ejecute. Por defecto, la ejecución se realiza mediante `npm start`, comando que no es compatible la arquitectura de esta aplicación, por lo tanto, hay que cambiar la forma en la que se ejecuta la aplicación. Hay varias alternativas, pero se ha optado por la que más se adapta a esta arquitectura.

Para ello, se utiliza la clase *Servidor.js*, que funcionará como un *script*, tal y como se ha explicado en el apartado de *Implementación*, para que se ejecute el directorio dist. Tras esto, en package.json se añade que cuando *Heroku* ejecute el comando start, se ejecute el script *Servidor.js*, por lo que cuando *Heroku* lea este JSON para instalar dependencias cambiará la forma de ejecutar la aplicación por la que se acaba de definir.

6.4 Subir el proyecto local al remoto y ejecutar la aplicación

Mediante el comando git commit -am se prepara el repositorio para ser subido al servidor remoto de *Heroku*, incluyendo los cambios del apartado anterior. Por último, mediante git push heroku master, se sube el proyecto local. Cuando *Heroku* lo recibe, automáticamente lo compila y lo despliega, utilizando el script previamente creado. Cuando termina este proceso devuelve el link para acceder a la aplicación, como se observa en la *Figura 14.*

```
remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> web
remote:
remote: -----> Compressing...
remote:        Done: 37.7M
remote: -----> Launching...
remote:        Released v17
remote:        https://simulador-easy8.herokuapp.com/ deployed to Heroku
remote:
remote: Starting November 28th, 2022, free Heroku Dynos, free Heroku Postgres, and fr
onger be available.
remote:
remote: If you have apps using any of these resources, you must upgrade to paid plans
ntinue to run and to retain your data. For students, we will announce a new program
 at https://blog.heroku.com/next-chapter
remote:
remote: Verifying deploy.... done.
To https://git.heroku.com/simulador-easy8.git
   503af3f..5597330  master -> master
```

Figura 14: final de la compilación y ejecución de la aplicación en Heroku y link para acceder.

En caso de que en un futuro se requiera hacer algún cambio a la aplicación, con tan solo repetir estos últimos pasos *Heroku* se encarga automáticamente de actualizarlos.

7. Pruebas

En este capítulo se realizarán una serie de pruebas para verificar el correcto funcionamiento de la aplicación.

7.1 Pruebas de uso

En este apartado se va a realizar un recorrido completo de todas las funcionalidades de la aplicación, así como una prueba de un uso real del simulador, para comprobar su correcto funcionamiento.

7.1.1 Simulador

Se han realizado dos tipos de pruebas.

El primer tipo ha consistido en verificar el funcionamiento de cada unidad funcional de forma individual, verificando que la activación de sus señales de control produce el resultado esperado. Para estas pruebas se han realizado transferencias registro-registro, registro-memoria y entrada/salida, y viceversa. Algunas transferencias se han utilizado también para verificar el funcionamiento de las unidades combinacionales, como la ALU y el multiplexor. Todas las pruebas han resultado satisfactorias.

El segundo tipo de prueba ha consistido en simular la ejecución de una instrucción máquina. El principal objetivo de estas pruebas es verificar la usabilidad del simulador.

Como ejemplo, se describe la ejecución de una instrucción de incremento en una unidad de un valor inmediato y su escritura en el registro A. Se omite la fase de búsqueda de la instrucción.

Se supone que el valor inmediato se encuentra en la posición de memoria 2, y que su valor es 7.

Para ello, el primer paso será guardar en la posición de memoria principal 2 el valor que queremos leer, tal y como se enseña en la *Figura 15*, así como guardar un 2 en el registro PC, *Figura 16*, que será la dirección que leerá la memoria principal.

Figura 15, inicialización de PC

Figura 16, inicialización de la memoria principal

Una vez se han inicializado los valores anteriores, se seleccionarán las señales que se tienen que activar en cada ciclo para lograr el objetivo planteado, es decir, ejecutar la instrucción máquina. Estas señales pueden seleccionarse antes de empezar la simulación o durante esta, siempre y cuando no se quiera activar una señal en el ciclo que se esté ejecutando o anterior. Para este ejemplo se deben activar las señales que se muestran en la *Figura 17.*

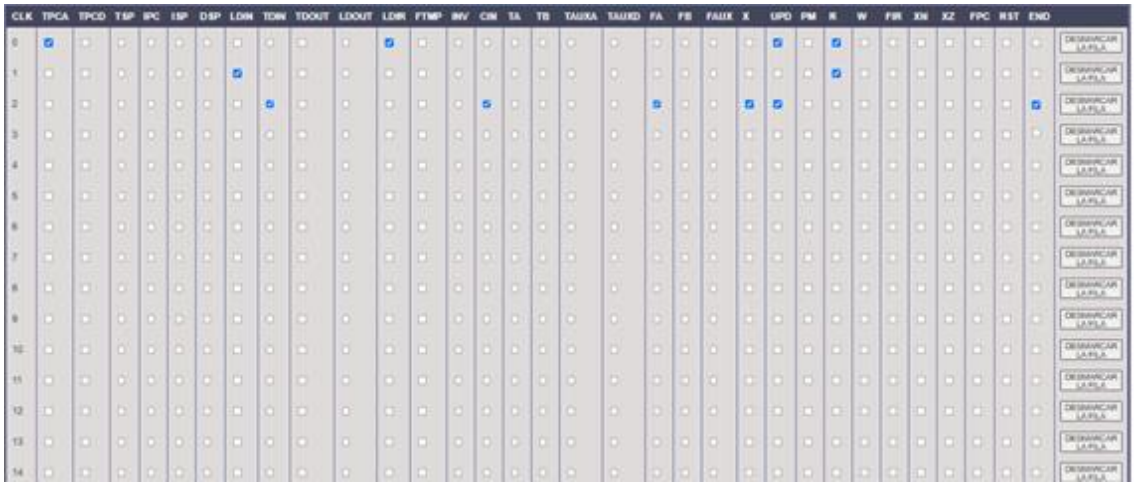

Figura 17, selección de señales activas en cada ciclo

Al hacer clic en el botón de ciclo, comenzará la simulación en el ciclo 0, *Figura 18.* En este ciclo se inicializan todos los registros y demás valores a 0, excepto aquellos a los que se les haya asignado a priori un valor, que tomarán éste. Además, se podrá saber en qué ciclo se encuentra la unidad de control mirando el contador de fases y mirando que fila de la tabla esta sombreada. Al activar TPCA, se permitirá la salida del valor 2 almacenado en PC al bus de direcciones interno. Al estar activo LDIR, DIR almacenará este valor y lo mandará al bus de direcciones externo. Puesto que no está activada la señal PM, se activará la memoria principal en lugar de la entrada/salida. Como la lectura se realiza en dos ciclos, en este se activa R para que la memoria se prepare para leer. Además, para observar el comportamiento de las *flags*, aunque no sea necesario se ha activado UPD para observar que puesto que el resultado de la ALU es cero se ha activado Z.

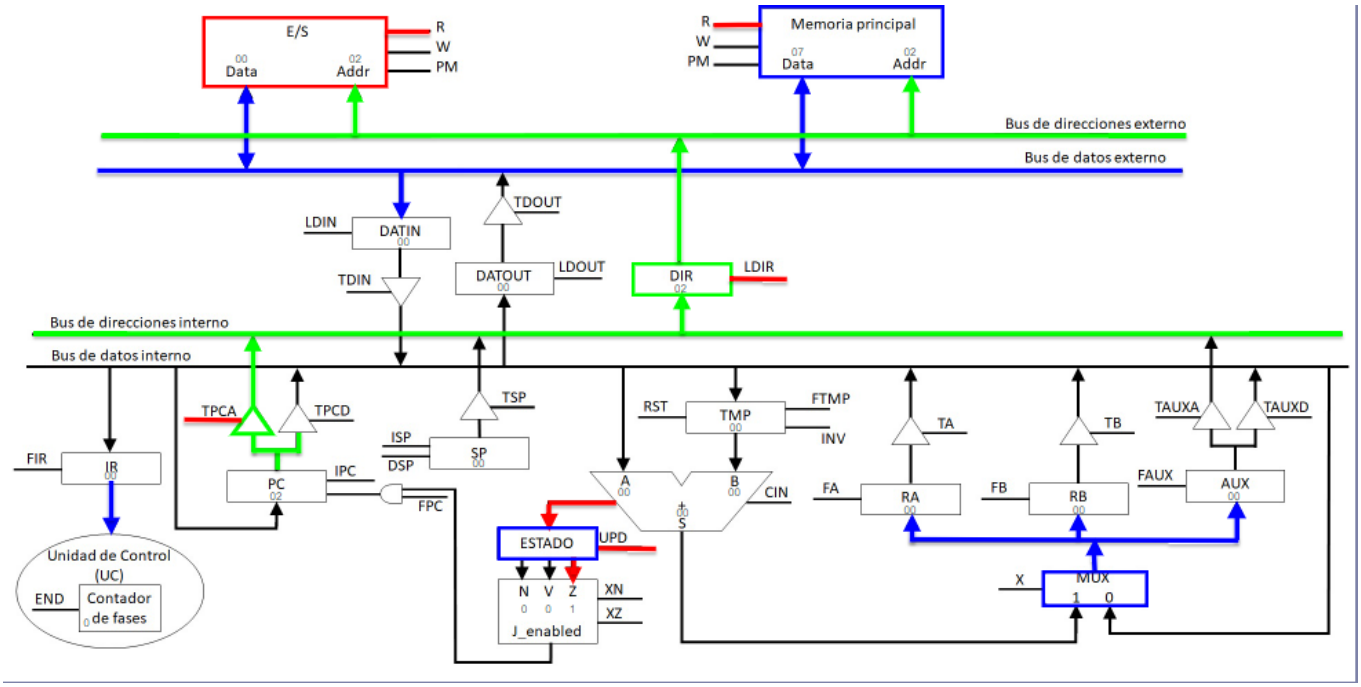

Figura 18, ciclo 0 de la simulación

En el siguiente ciclo, el ciclo 1 como podemos observar en la *Figura 19,* se vuelve a activar R para terminar el proceso de lectura de la memoria, por lo que se devolverá el dato, 7 en este caso, que se había grabado antes de empezar la simulación. Teniendo ya el dato deseado se ha activado LDIN para que quede registrado en DATIN.

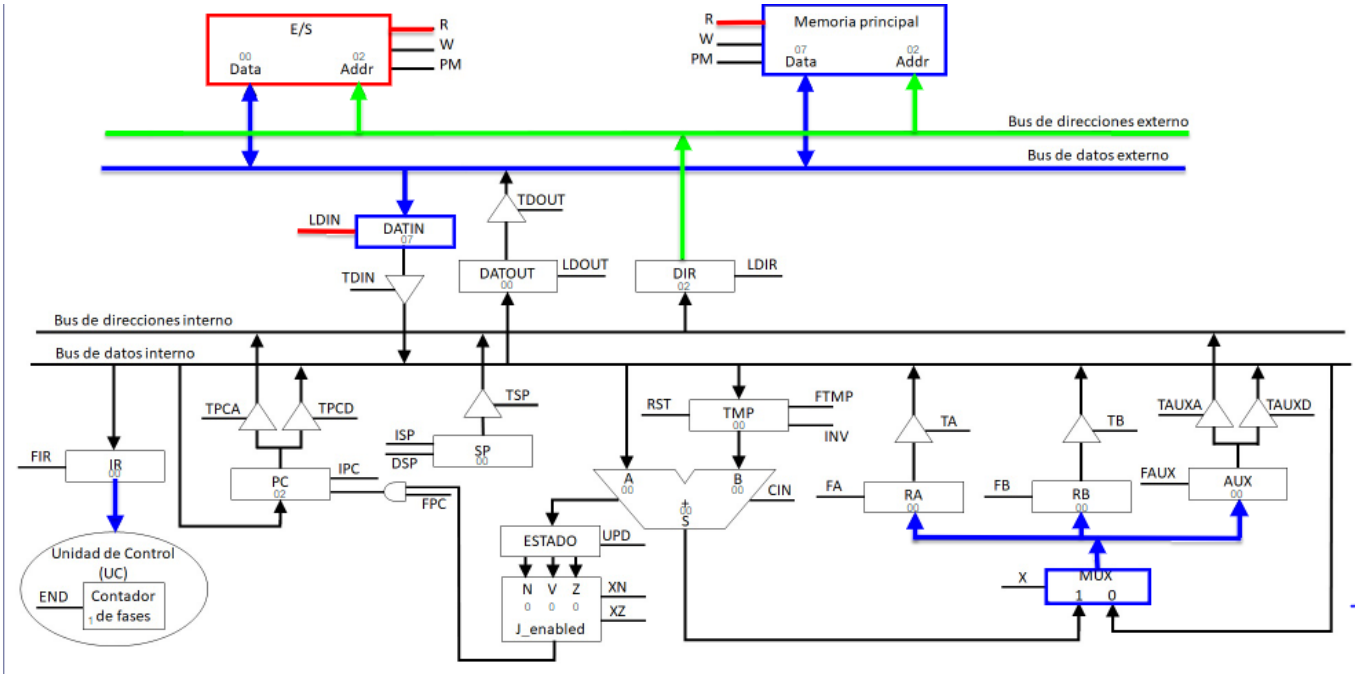

Figura 19, ciclo 1 de la simulación

En el ciclo 2, *Figura 20*, teniendo ya el dato deseado en el registro DATIN se activan TDIN para dejar salir el dato al bus, por lo que llega a la ALU. En esta, se activa CIN para sumarle uno y se activa el multiplexor con la señal X para que RA, con FA activo, almacene el valor que sale de la ALU y no el del bus. En este caso, dado que se ha realizado una operación aritmética, se activa UPD para actualizar el registro de flags. Podemos ver que se activa ningún flag pues el resultado no cumple con ninguno de los requisitos explicados en el apartado de *Implementación.* Tal como se indica en la figura 7, la combinación de los flags y las señales XN y XZ hace que se active la salida *J_enabled.* Si se activara también la señal FPC la puerta AND permitiría la escritura del registro PC, por lo que tomaría el valor del bus de datos.

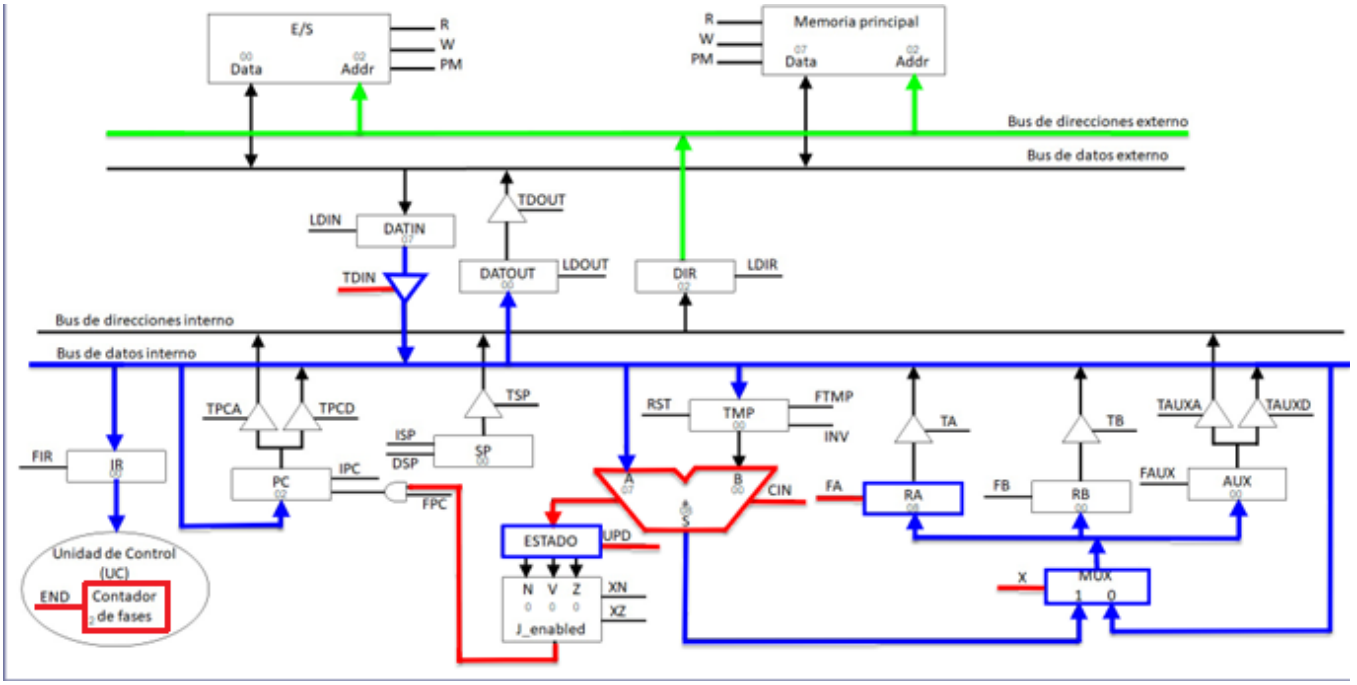

Figura 20: ciclo 2 de la simulación

Por último, el ciclo 2 también activa la señal END por lo que, al ciclo siguiente, *Figura 21,* se retornará al ciclo 0 pero manteniendo los valores que se han conseguido en la simulación de los ciclos anteriores.

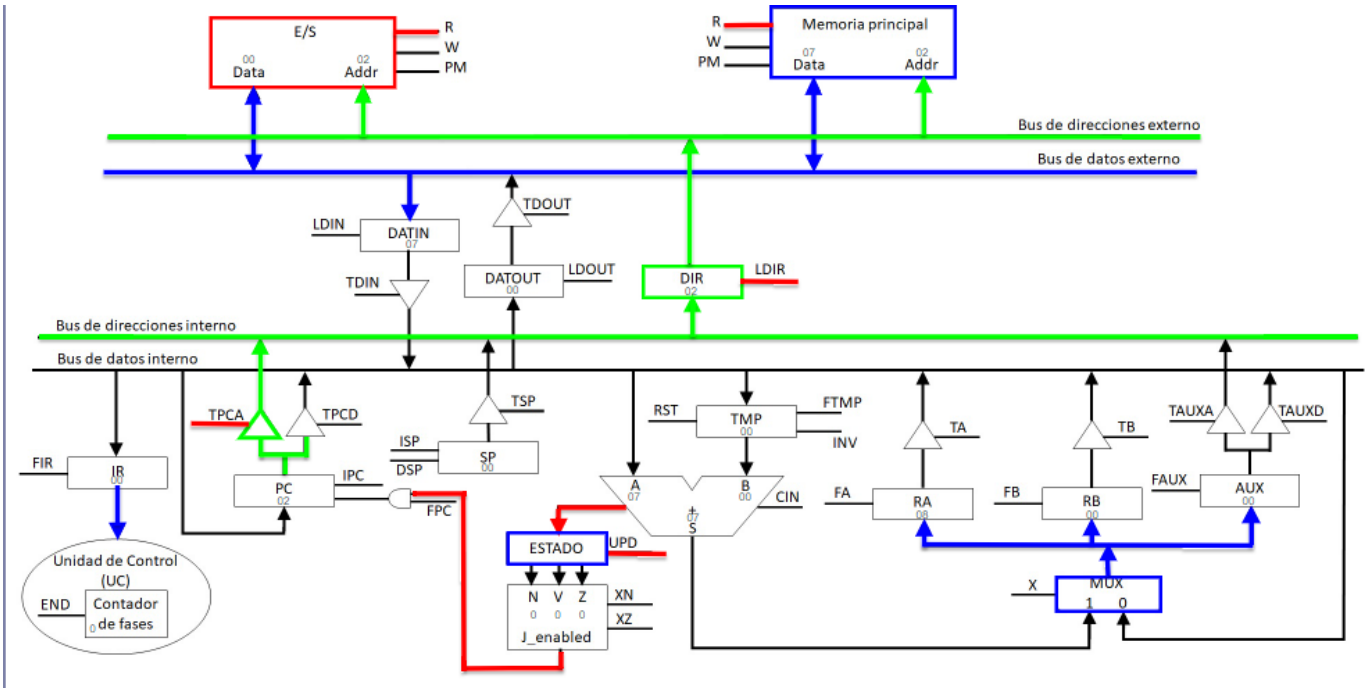

Figura 21: ciclo 0 tras la activación de la señal END.

7.1.2 Sistema de ficheros

El sistema de ficheros se basa en dos funciones, cargar un fichero o generarlo, siguiendo el formato planteado en el capítulo de *Diseño*.

Al pulsar el botón de generar fichero, se descargará mediante el explorador dónde este abierta la aplicación un archivo en formato *txt*, para esta prueba usaremos el generado de la simulación anterior tal y como se enseña en la *Figura 22*.

```
F0: TPCA, LDIR, UPD, R
F1: LDIN, R
F2: TDIN, CIN, FA, X, UPD,END
```

Figura 22: fichero de texto generado de la simulación anterior

Al clicar en el botón de seleccionar archivo, se abrirá el explorador de archivos del sistema operativo para seleccionar un fichero de texto, tal y como se enseña en la *Figura 23*.

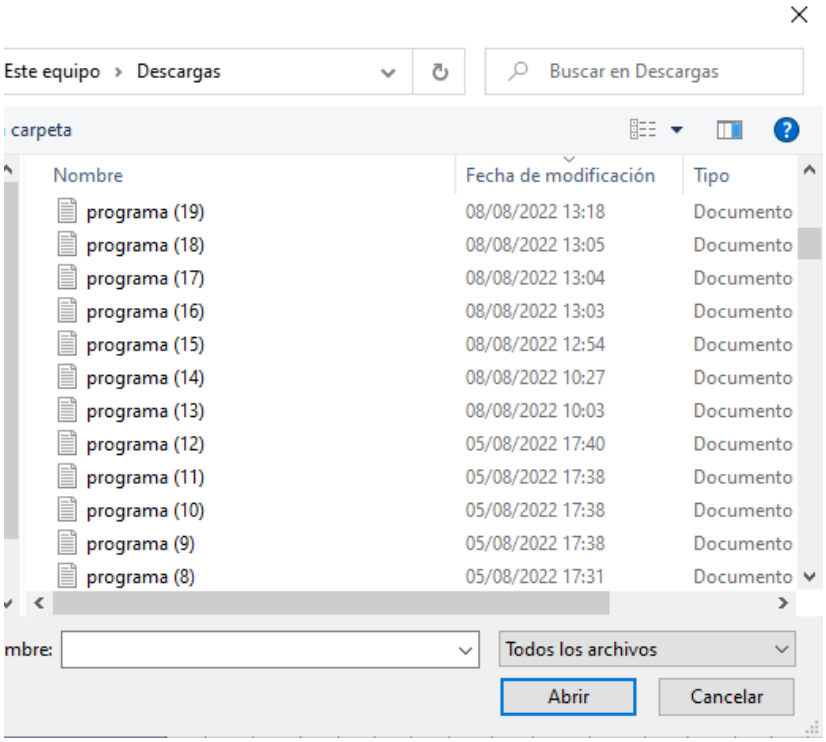

Figura 23: explorador de archivos para escoger un archivo de texto a subir.

Una vez se hay seleccionado el archivo, el programa comprueba que tenga el formato adecuado y si es así, carga la secuenciación escrita en este, como se puede observar en la *Figura 24*. La simulación se realizará de igual manera que si se hubieran seleccionado manualmente, eso sí, se tendrán que inicializar los registros y la memoria manualmente.

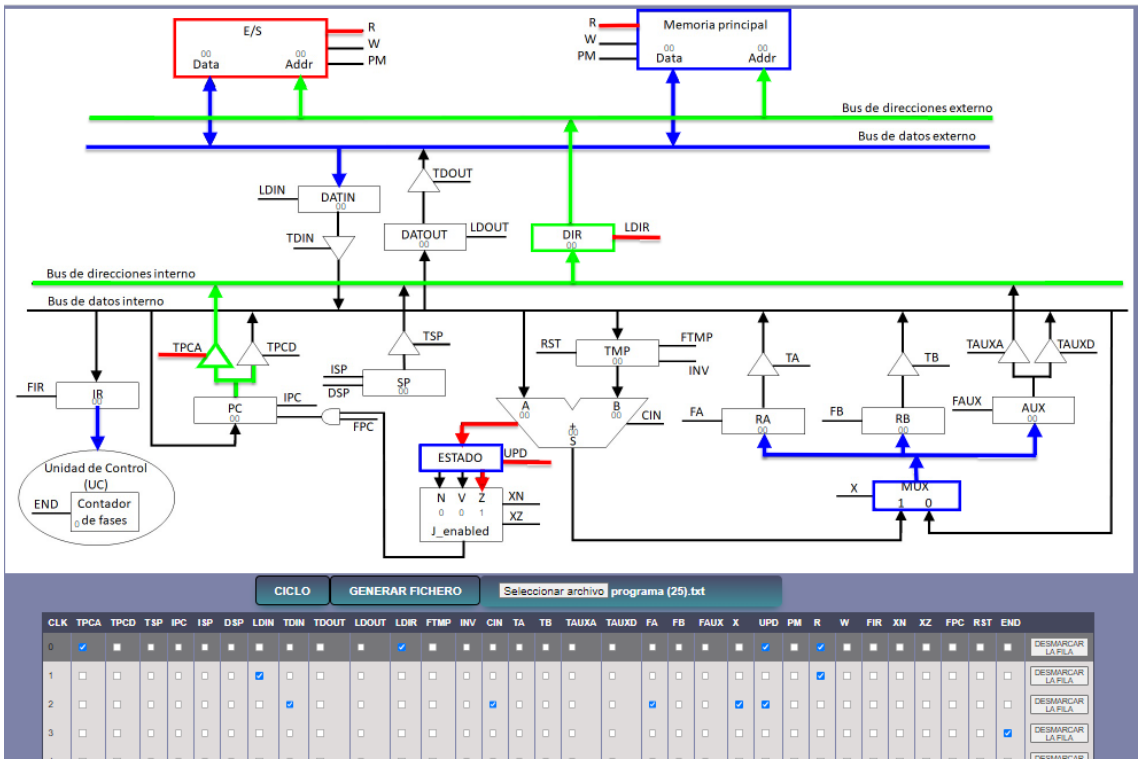

Figura 24, comprobación de una ejecución del simulador cargada por fichero.

8.Conclusiones

.

Este trabajo tenía como objetivo desarrollar una aplicación web para que los estudiantes dispongan de un simulador de la ruta de datos del Easy8, para que puedan entender cómo funciona internamente un procesador y tener un primer acercamiento no solo de este computador si no de cada componente específico, así como de las instrucciones máquina que se pueden ejecutar. Siendo estos alumnos de la asignatura de *Fundamentos de Computadores*, se entiende que no están experimentados en esta materia, por lo que durante todo el desarrollo del proyecto se ha tenido en cuenta que se tenía que mejorar el simulador anterior para que sea lo más completo posible y tenga a su vez tanto un uso como una interpretación fácil.

Llegado a este punto, se puede decir que se ha logrado este objetivo, así como los demás requisitos planteados en el análisis, como se ha ido explicando a lo largo de la memoria. También ha favorecido mucho a que se cumplan dichos objetivos subir la aplicación a un servidor web, para que se pueda acceder fácilmente a ella y empezar a usarla desde el primer momento. Espero que esta aplicación pueda ayudar a la formación de nuevos alumnos.

Desde el punto de vista personal, aunque nunca había creado un proyecto en Vue ni tampoco había desplegado ninguno, sí que había usado herramientas parecidas por lo que he podido comprender la documentación disponible y asimilándolo con lo que ya había aprendido en la carrera, por lo que me ha ayudado a comprender mejor y a expandir mis conocimientos.

La mayor dificultad de este proyecto ha sido gestionar la interacción de cada uno de los componentes de la aplicación entre sí, pues siempre acababa uno sobreescribiendo a otro o iban saliendo excepciones a tratar; aunque también ha sido la parte más satisfactoria del mismo, ver como al probar la simulación cada componente hacía su función y ver cómo funcionaba la interacción entre cada uno de ellos. Es este "trasteo", casi como jugar con el simulador, lo que espero que despierte la curiosidad de los alumnos, el: "¿Qué pasa si activo este registro después de sumar estos dos?" o "¿Y si leo este registro de la memoria y luego lo invierto?"; y que poco a poco acabe ayudando a sentar las bases de esta ingeniería.

Para terminar, añadiré las asignaturas del Grado que más me han ayudado en la realización de éste proyecto, siendo estas las siguientes:

- Las asignaturas de computadores **Fundamentos de Computadores, Estructura de Computadores y Arquitectura e Ingeniería de Computadores**, donde adquirí los conocimientos sobre computadores y lenguaje ensamblador necesarios.
- **Interfaces persona computador,** para que a la hora de realizar la interfaz pensar en todo momento cómo reaccionaría el usuario y que elementos tendría que añadir para mejorar el diseño.

- **Estructura de datos y algoritmos,** para estructurar bien el código de la aplicación y saber cuál utilizar en cada momento; y **Lenguajes de programación y procesadores de lenguajes** para saber procesar bien dicho código.
- Y por último, **Redes de computadores,** donde aprendí los protocolos web necesarios para el despliegue de la aplicación.

Como trabajos futuros, y continuación de este TFG, se podría integrar este simulador con un simulador de computador, donde los alumnos introduzcan programas ensamblador y puedan ejecutar dicho programa, y en determinados momentos o instrucciones, acceder a la vista de la ruta de datos para introducir u observar, la secuenciación de una instrucción máquina. También, incorporar una verificación automática de la corrección de una secuencia de activación de señales de control. No es infrecuente que los alumnos simulen una instrucción y duden de si el resultado es correcto.

9.Bibliografía

[1] Hamacher, V. Carl, Organización de computadores, 2003

[2] Rafael González Carrizo, Simulación y almacenamiento de programas del Easy8 en Web, TFG, 2019

[3] I. Somerville, Ingeniería del software, Pearson, 2005

[4] S. Pressman, Roger, Ingeniería del software: Un enfoque práctico, 2010

[5] Sergio Luján Mora, Programación en Internet: Clientes Web, 2010

[6] E. M. Hahn, Express in Action: Writing, building, and testing Node.js applications, Manning, 2016

[7] Santacroce, Ferdinando, create, merge, and distribute code with Git, the most powerful and flexible versioning system available, 2015

[8] Heroku, NoSQL, Heroku, and You, 2010

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**			**x**	
ODS 2. **Hambre cero.**				**x**
ODS 3. **Salud y bienestar.**			**x**	
ODS 4. **Educación de calidad.**	**x**			
ODS 5. **Igualdad de género.**				**x**
ODS 6. **Agua limpia y saneamiento.**				**x**
ODS 7. **Energía asequible y no contaminante.**			**x**	
ODS 8. **Trabajo decente y crecimiento económico.**		**x**		
ODS 9. **Industria, innovación e infraestructuras.**		**x**		
ODS 10. **Reducción de las desigualdades**.			**x**	
ODS 11. **Ciudades y comunidades sostenibles**.			**x**	
ODS 12. **Producción y consumo responsables.**			**x**	
ODS 13. **Acción por el clima.**				**x**
ODS 14. **Vida submarina.**				**x**
ODS 15. **Vida de ecosistemas terrestres.**				**x**
ODS 16. **Paz, justicia e instituciones sólidas.**				**x**
ODS 17. **Alianzas para lograr objetivos.**				**x**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En este anexo, se va a realizar una reflexión sobre la relación de este proyecto con los ODS.

Los Objetivos de Desarrollo Sostenible, establecidos en 2015 por la Asamblea General de las Naciones Unidas, son 17 objetivos globales conectados entre sí y diseñados como un plan para lograr un futuro mejor y más sostenible para todo el mundo. Estos objetivos pretenden acabar con la pobreza, proteger el planeta y mejorar las vidas de todas las personas del mundo, entre otros. Por lo tanto, se espera que todos los países, sean miembros o no de la ONU e independientemente de su nivel de riqueza participen y se comprometan con ellos. Se pretende haber alcanzado estos objetivos para 2030. En la siguiente ilustración, *Figura 25*, se pueden observar los 17 ODS.

Figura 25: los 17 ODS

Por supuesto, las nuevas tecnologías tienen un gran peso dentro de estos objetivos, pues gracias a ella se puede mejorar la vida de las personas y, de una manera u otra, acaban influyendo en todos los ODS.

En lo referente a la relación de este trabajo con dichos objetivos, este trabajo está muy ligado con el objetivo número cuatro: **Educación de calidad**.

Este objetivo tiene como foco garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos.

Aunque este proyecto tenga como usuario objetivo a los alumnos del grado y no a cualquier persona, facilita el aprendizaje y el grado de inicialización de estos en una competencia técnica y profesional que les abrirá paso al acceso de un empleo decente y/o al emprendimiento. Por ello, se puede afirmar que cumple con la meta 4.4 del objetivo ayudando así al cumplimiento del mismo.

Siendo una buena educación la base del desarrollo, y con lo ligado que está con la tecnología, especialmente en el mundo de la informática, se podría decir que mejorando la educación de este sector se puede tener un impacto positivo en prácticamente todos los objetivos, pues con todas las salidas que existen, y que siempre pueden aparecer más en un futuro, un alumno puede acabar trabajando en proyectos que fomenten el desarrollo de la sociedad.