UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

Simulador de un nanoprocesador segmentado

AUTOR: *Omar José Sanz Rodríguez*

TUTORES: *Félix García Carballeira*

Juan Antonio Ortega Ruiz

Leganés, 19 de Septiembre de 2015

Agradecimientos

Me gustaría agradecer a todos uno por uno todo lo que me han aportado durante esta carrera, pero en una sola cara me sería imposible, asique intentaré generalizar y espero que os deis por aludidos en algún párrafo todos los que lo leáis ☺.

Quiero agradecer a mi familia el que hayan creído en mí desde el primer día, ya no solo por apoyarme también por mostrarme como referencia para los más pequeños, me gustaría que leyeran esto y que vieran por ellos mismo como con esfuerzo y creyendo en uno mismo una persona puede lograr todo lo que se proponga.

De mi familia en especial me gustaría agradecer a mis padres José Luis y Neme por haber sabido guiarme durante todos estos años, haber confiado en mí, por realizar siempre ese esfuerzo extra que han hecho siempre para darme la mejor educación y permitirme estudiar lo que siempre he querido, estar a mi lado cuando lo he requerido y aguantarme cuando he estado de mal humor con el mundo.

Al pequeño de la casa, Marcos, que si no le dedico nada se enfada, agradecerle esos “fifitas” para relajarse entre horas de estudio y prácticas, sé que en el fondo perdía aposta para que no me fuese enfadado a estudiar.

A mis compañeros de clase que desde la primera semana hemos estado juntos y que tantos buenos momentos hemos pasado entre práctica y práctica en ese famoso laboratorio 1.0.F04 y que han hecho que esta etapa en lugar de convertirse en un calvario fuera inolvidable y épica.

Me gustaría destacar a mis tutores de proyecto, Félix y Juan Antonio por su esfuerzo, tiempo y dedicación que han tenido conmigo para que este proyecto existiese.

A Teo y Diego por esas largas noches insanas de entrega a última hora que finalmente merecieron la pena ya que todos estamos aquí gracias a ello, suena como mala experiencia pero la recordaré con mucho afecto.

A Fátima por estar siempre atenta, preocuparse y alegrarse por mí a pesar de la distancia. A su hija Jenifer, mi novia, por su cariño, apoyo y aguante cada día durante estos cuatro años, sobre todo en las épocas más insoportables de prácticas-exámenes. Gracias por enseñarme a creer en mí mismo, a valorarme y a enfrentarme sin miedo a mis temores. Por demostrarme el orgullo que sientes por haber logrado terminar la carrera y que solo tú sabes de verdad lo que ha costado alcanzar, por todos los detalles que has tenido conmigo para cada momento y por último, agradecerte (de parte de Teo y Diego también) habernos ofrecido esa segunda casa sin importar las horas y ruido sin ninguna queja.

Y no quiero terminar sin agradecer a aquellos que durante esta carrera dejaron de estar conmigo, sé que estén donde estén me han empujado para que siguiera adelante cuando he estado estancado y me han seguido enseñado a valorar lo que tenemos, disfrutar de cada momento y luchar por lo que queremos.

"No. No lo intentes. Hazlo, o no lo hagas, pero no lo intentes." ***—Maestro Yoda***

Tabla de contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

En este apartado se muestra una visión global del proyecto realizado y documentado en la presente memoria, incluyendo motivaciones y objetivos que llevaron a la elección de este Trabajo Final de Grado (TFG).

1.1 Motivación

El Trabajo Fin de Grado se propone estando como becario en la empresa Crisa Airbus Defence and Space como continuación de un proyecto ya comenzado por Juan Antonio Ortega jefe de departamento de microelectrónica. Este proyecto consiste en simular un nanoprocesador en lenguaje de programación C e importarlo a tcl. La fase desarrollada por Juan Antonio Ortega consiste en un programa en lenguaje C que permite transformar un fichero en lenguaje ensamblador .asm en varios ficheros con distintos formatos que serán utilizados para la carga inicial del nanoprocesador a simular. Los ficheros originados tras la ejecución del programa son los siguientes: fichero con formato .reg contiene los datos que se desean cargar inicialmente en el banco de registros del nanoprocesador, el fichero con formato .var contiene los datos que se desean cargar inicialmente en la memoria de datos y por último el fichero con formato .obj contiene los datos que se desean almacenar en la memoria de instrucciones.

Se pretende por tanto en este proyecto, simular el funcionamiento de un nanoprocesador en lenguaje C que permita ejecutar una serie de instrucciones en ensamblador facilitadas en el Anexo B y pueda ser importado a tcl para su posterior cosimulación con VHDL mediante el programa ModelSim, de tal forma, que permita cargar un programa en ensamblador a partir de los tres ficheros mencionados anteriormente (.reg, .var y .obj) sobre el simulador y que este permita ejecutar el programa y ver el camino que recorren los datos a través del nanoprocesador.

Este proyecto despertó mi interés ya que podía realizar un trabajo que necesitase investigar, incluía programar y me permitía la posibilidad de participar en un proyecto dentro de una empresa como Crisa.

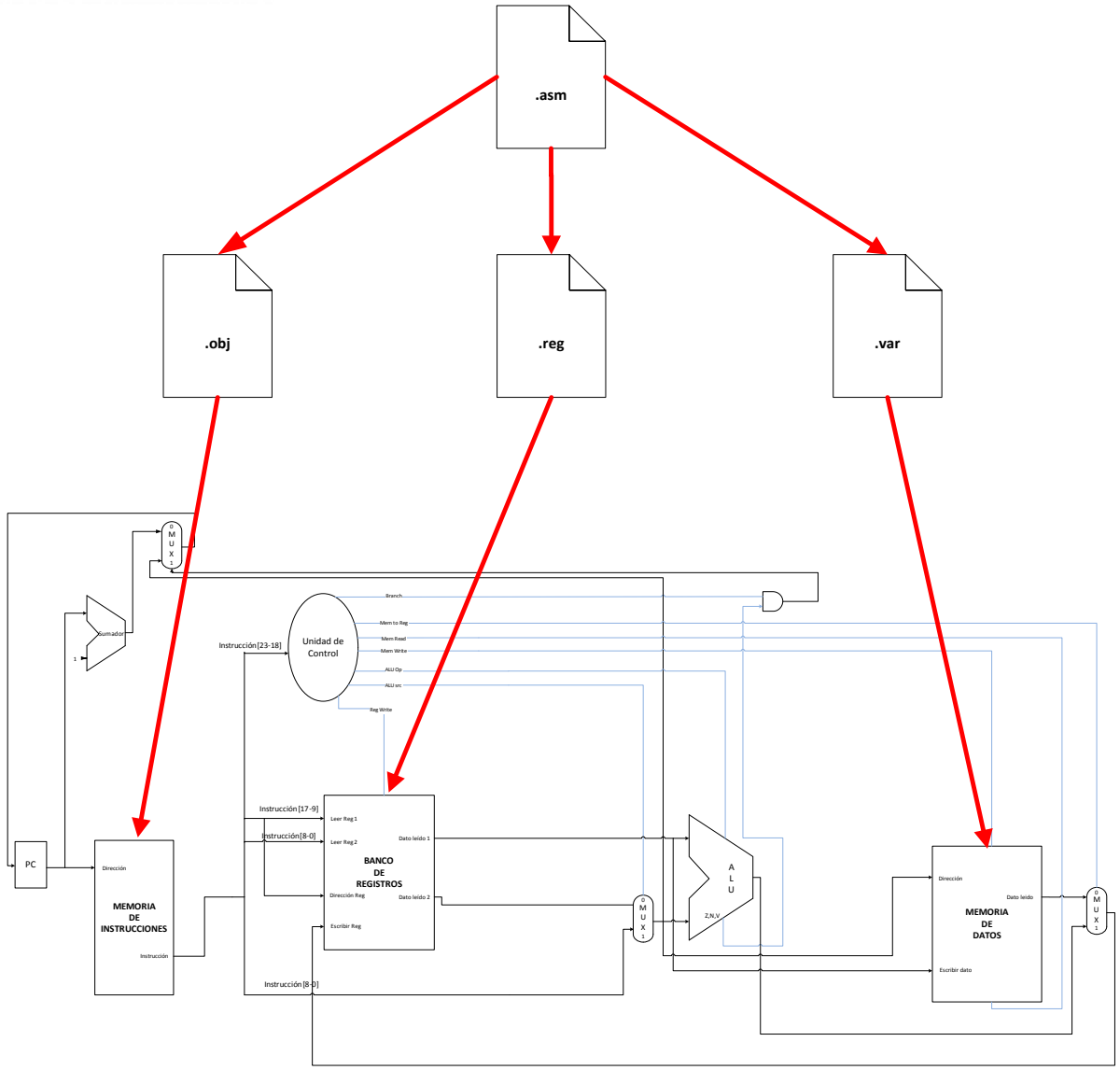


Ilustración 1. Esquema inicial proyecto nanosimulador

1.2 Objetivos

Una vez explicados los motivos con los que se inició la idea y el desarrollo del proyecto, en este apartado se lleva a cabo la descripción de los objetivos principales y específicos que el simulador que se generé debe cumplir.

1.2.1 Objetivos principales

El objetivo principal del proyecto es desarrollar una herramienta que simule el comportamiento de un nanoprocesador utilizando el lenguaje tcl que permita la cosimulación con otro programa vhdl ya existente en la herramienta ModelSim, con el fin de comprobar el correcto funcionamiento del programa vhdl que más tarde será implantado sobre una FPGA.

1.2.2 Objetivos específicos

A parte del objetivo principal, surge una serie de objetivos específicos con el fin de permitir el cumplimiento del objetivo principal. Los objetivos específicos a desarrollar en el proyecto son los siguientes:

- Permitir la carga de los ficheros .reg, .var y .obj sobre las estructuras del nanoprocesador.
- Permitir ejecutar el programa ensamblador por ciclos o de manera continua.
- Permitir visualizar el pipeline durante la ejecución del programa sobre el nanoprocesador.
- Visualizar las estructuras que componen el nanoprocesador.
- Modificar las estructuras del nanoprocesador en cualquier momento.

1.3 Estructura del documento

A continuación se procede a enumerar los apartados en los que se divide el documento de Trabajo Fin de Grado junto a una breve descripción del contenido de estos:

· Apartado 1: Introducción. Breve introducción sobre el Trabajo Fin de Grado realizado, se explica la motivación, sus objetivos, la estructura del documento y el entorno socio-económico.

· Apartado 2: Estado del arte. En este apartado se hace referencia al estado actual de la tecnología.

- Estado del arte de los procesadores. Se analiza la arquitectura MIPS posible base de estudio para el diseño del nanoprocesador.
- Estado del arte de los simuladores. Se hace referencia al estado actual de los simuladores de procesadores.

· Apartado 3: Diseño inicial nanoprocesador secuencial. En este apartado se muestra la primera idea de la arquitectura del nanoprocesador junto a sus principales componentes.

· Apartado 4: Análisis del sistema. En este apartado se define los casos de uso del sistema y los requisitos que debe cumplir, comprueba si es factible y estudia los posibles caminos a seguir.

· Apartado 5: Propuesta de diseño e implementación del nanoprocesador. En este apartado se define de forma adecuada y clara la arquitectura del sistema y la implementación del mismo incluyendo la descripción de las clases que lo forman.

· Apartado 6: Gestión del proyecto. En este apartado se realiza una estimación del tiempo necesario para la realización del proyecto.

· Apartado 7: Pruebas de verificación. En este apartado se comprueba el correcto funcionamiento del sistema implementado, realizando sus labores de forma adecuada.

· Apartado 8: Presupuesto. En este apartado se describe detalladamente del coste del proyecto.

· Apartado 9: Conclusiones y trabajo futuro. Tras la finalización del proyecto, se desarrollan una serie de conclusiones y líneas futuras a tener en cuenta.

· Apartado 10: Glosario de términos y referencias.

1.4 Entorno socio-económico

A día de hoy el uso de los simuladores con iteración es muy frecuente debido a que son modelos manipulables de un sistema real o teórico, lo que permite su uso para entrenar y aprender las reacciones del sistema en función de las acciones que se realicen sobre él. En una simulación, aunque suele ser una simplificación del mundo real, se pueden resolver problemas y situaciones, aprender procedimientos, llegar a entender las diferentes características de los fenómenos, aprender cómo controlarlos y qué acciones realizar en situaciones particulares. Con estos sistemas el usuario puede experimentar hasta agotar su posibilidad de variación del modelo en cuestión.

Como se ha comentado en el apartado 1.2, el proyecto a desarrollar surge de la necesidad por parte del departamento de microelectrónica de la empresa Crisa Airbus Defence and Space de crear un nanoprocesador simulado en lenguaje tcl para la cosimulación con un programa VHDL ya existente que posteriormente será implementado en las FPGA.

La tecnología FPGA (Field Programmable Gate Array) consiste en un dispositivo que contiene bloques de lógica programable y que por ello puede reproducir desde funciones sencillas como una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip, es decir, se puede trasladar el diseño hecho en papel al dispositivo físico para poder probarlo. Por ello, a día de hoy es una tecnología utilizada muy frecuentemente por su gran cantidad de utilidades.

Para la programación de este dispositivo, el diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL y uno de los más utilizados es el VHDL.

2. Estado del arte

En este apartado se muestra la investigación previa a la realización del proyecto sobre la arquitectura de microprocesadores y los simuladores actuales, se estudia la arquitectura de los microprocesadores ya que para crear el simulador primero hay que realizar un diseño de un nanoprocesador para que al simular el recorrido de los datos sea el más parecido posible al de uno real.

Como se ha comentado en la introducción, la arquitectura principal sobre la que me he basado para realizar el diseño de la arquitectura del nanoprocesador ha sido el procesador MIPS (Microprocessor without Interlocked Pipeline Stages) ya que este ha sido el más utilizado a lo largo de la carrera durante las asignaturas que tenían relación con la arquitectura de computadores.

MIPS es una arquitectura RISC desarrollado por MIPS Technologies, fue diseñado para optimizar la segmentación en unidades de control y para facilitar la generación automática de código máquina por parte de los compiladores y es utilizado por múltiples fabricantes de microprocesadores [7].

2.1 ¿Qué es un procesador RISC?

Es un procesador formado por un conjunto reducido de instrucciones, no por tener pocas, si no por la sencillez de estas.

Decimos que un procesador es RISC cuando la misma instrucción que carga datos de memoria no realiza ninguna otra acción sobre ellos, en caso de querer realizarla habría que esperar a otra instrucción para que realizase el tratamiento que se desea sobre esos datos. Gracias a esa sencillez en las instrucciones, los bloques lógicos que se encargan de traducir ocupan menos espacio y por lo tanto el sistema puede tener frecuencias de funcionamiento mayores.

Sin embargo, este tipo de procesador tiene sus desventajas:

- Tiene menos potencia que otros procesadores para la ejecución de operaciones como las de tratamiento de señales, fotos o videos, todas aquellas que sea mejor tener instrucciones que puedan tratar cientos de datos y escribirlos en memoria de una sola vez.
- Los programas tienden a ser mayores, al querer realizar ciertas operaciones, es necesario realizar varias operaciones sencillas, lo que hace que el programa incremente su tamaño, sin embargo, a día de hoy no supone un problema ya que el precio de los discos de almacenamiento ha bajado y su tamaño ha aumentado.

2.2 Diferencia entre arquitectura RISC Y CISC

CISC a diferencia de RISC es un conjunto de instrucciones complejo, es decir, CISC se caracteriza por tener un conjunto de instrucciones muy amplio y permitir operaciones complejas entre datos situados en memoria o en los bancos de registros internos.

Sin embargo, CISC tiene un tipo de arquitectura que dificulta el paralelismo entre instrucciones, por lo que a día de hoy la mayoría de estos procesadores implementan un sistema que convierte las instrucciones a otras más simples, como las que tiene la arquitectura RISC, llamadas microinstrucciones [8].

Esto último nos hace pensar que la tendencia futura es que ya no existan los CISC puros. Realmente la diferencia entre las dos arquitecturas RISC y CISC cada vez es más borrosa ya que las CPU combinan elementos de ambas.

2.3 Historia de MIPS

El primer diseño MIPS fue el R2000, este fue presentado en 1985 por la empresa MIPS Computer Systems. Este diseño añadía instrucciones multiciclo que permitían multiplicar y dividir en una unidad independiente integrada en el procesador. Asimismo se añadieron instrucciones para enviar los resultados de estas al núcleo, las cuales necesitaban de bloqueos. El R2000 permitía iniciarse como formato big-endian o como little-endian, estaba formado por 32 registro de tamaño 32 bit, pero no contaba con registro de estado, lo que generaba un cuello de botella y a diferencia de otros registros, el contador de programa no era directamente accesible. Este diseño soportaba hasta cuatro co-procesadores, uno de ellos integrado en la CPU principal para manejar las interrupciones y excepciones, mientras que el resto estaba destinado para otros usos. Existía una opción también de añadir la unidad de punto flotante en el diseño R2010, que al igual que R2000, contenía 32 registros de 32 bits que podían ser utilizados para manejar números de simple precisión, sin embargo con el nuevo diseño estos registros podían ser transformados en 16 con un tamaño de 64 bits en doble precisión.

Más tarde el diseño del R2000 fue mejorado con el R3000 de 1988. Este nuevo diseño añade una cache de 32kB, aumentada posteriormente a 64kB, para almacenar instrucciones y datos. Los primeros diseños tuvieron algunos defectos en el soporte del multiprocesador, sin embargo MIPS lo siguió incluyendo en varios diseños exitosos. También incluye una unidad de manejo de memoria (MMU) integrada, algo que tenían en común los procesadores en aquel momento. Este diseño fue el primer diseño más exitoso de MIPS en el mercado y se fabricaron más de 1 millón. Más adelante se crea el famoso R3000A, utilizado en la Sony PlayStation, fue una versión acelerada hasta los 40 MHz. Al igual que el R2000, el R3000 fue emparejado con la FPU R3010. Pacemips fabricó el R3400 e IDT el R3500, siendo ambos modelos procesadores R3000s con la FPU R3010 en un único chip. El Toshiba R3900 fue el primer sistema de chip único para los primeros ordenadores portátiles con Windows CE.

Esta familia de procesadores R2000 y R3000 fue la base de la empresa durante la década de los 80 siendo empleados en algunas series de workstations de SGI. Estos diseños se diferenciaban del resto creados por la nueva implementación de la mayoría de los bloqueos con hardware y proporcionar instrucciones completas.

El primer procesador con arquitectura autentica de 64 bits fue presentado por MIPS en 1991 y toma el nombre de R4000. Este procesador opera a una velocidad de reloj muy superior a los anteriores, inicialmente de 100MHz. Sin embargo, para alcanzar esa frecuencia se tuvo que reducir las caches a 8kB cada una (las anteriores eran de 64 kB), siendo necesario tres

ciclos de reloj para acceder a ellas. Estas altas frecuencias fueron alcanzadas gracias a la técnica de segmentación profunda o también llamada súper-segmentación.

MIPS realizó versiones más baratas de este modelo R4200 y R4300, está ultima consistía en un R4200 con bus externo de 32 bits. Uno de los usos que se le dio a este procesador fue para la Nintendo 64, la cual usaba una CPU NEC VR3400.

Los posteriores modelos como el R4600, R4700, R4650 y R5000 fueron diseñados por QED. Como se dijo anteriormente, R4000 aumentó la frecuencia a cambio de reducir la capacidad de la cache, sin embargo los diseños de QED contenían unas grandes cachés accesibles en solo dos ciclos de reloj por el uso eficiente del área de silicio. Los modelos R4600 y R4700 fueron utilizados para los primeros router Cisco basados en el MIPS, el modelo R4650 fue utilizado para WebTV y el R5000 gracias a su trabajo más eficiente y más flexible con números en simple precisión que el anterior modelo R4000 SGI los añadió a las viejas tarjetas gráficas para ofrecer mejor rendimiento gráfico. QUED posteriormente diseñó las familias RM7000 y RM9000 con el fin de ser usadas en sistemas embebidos como redes e impresoras láser.

El modelo R8000 diseñado en 1994 fue el primer diseño MIPS superescalar, capaz de ejecutar dos operaciones de ALU y otras dos de memoria en cada ciclo de reloj. El diseño se plasmó en seis chips: una unidad entera (con dos cachés de 16kB, una para instrucciones y otra L1 de datos), una unidad de punto flotante, tres de RAM de cache L2 personalizables (dos para acceso a caché secundaria y otra para bus), y un controlador de caché ASIC. El diseño tenía dos unidades segmentadas de suma-multiplicación en doble precisión, las cuales recibían el flujo de datos de la caché secundaria externa de 4 MB. El R8000 fue montado en los servidores SGI Power Challenge a mediados de los 90 y posteriormente en las estaciones de trabajo Power Indigo2. Su rendimiento limitado en operaciones enteras y su elevado coste lo hicieron impopular entre la mayoría de los usuarios, si bien el buen rendimiento de su FPU fue aprovechado por los usuarios científicos; el R8000 estuvo apenas un año en el mercado.

El modelo R10000 fue lanzado en 1995, este procesador está formado por un chip único con mayor velocidad de reloj que su antecesor R8000 y con cachés primarias con mayor tamaño, de 32 kB para instrucciones y datos, también era superescalar, pero su gran innovación fue ser “out of order”, aún con una fpu más simple, la mejora en las operaciones con enteros, su menor precio y la mayor densidad hicieron de este modelo el preferido de los clientes.

El modelo R12000 se basó en el R10000 y fue fabricado con tecnología mejorada para comprimir el chip y operar a mayor velocidad de reloj.

El modelo R14000 permite mayores frecuencias más un soporte adicional para DDR SRAM en el chip externo de caché y un FSB de 200 MHz para una mejor transferencia.

Los últimos modelos fueron llamados R16000 y R16000A, caracterizados por tener mayor velocidad de reloj, caché L1 adicional y un chip más pequeño.

A continuación se muestra una tabla con las especificaciones de todos los modelos MIPS anteriormente mencionados.

Modelo	Frecuencia (MHz)	Año	Proceso (µm)	Transistores (millones)	Tamaño del chip	Pines	Potencia (W)	Voltaje	Dcaché(k)	Icaché(k)	Scaché(k)
R2000	8-16.7	1985	2.0	0.11	--	--	--	--	32	64	--
R3000	12-40	1988	1.2	0.11	66.12	145	4	--	64	64	--
R4000	100	1991	0.8	1.35	213	179	15	5	8	8	1024
R4400	100-250	1992	0.6	2.3	186	179	15	5	16	16	1024
R4600	100-133	1994	0.64	2.2	77	179	4.6	5	16	16	512
R5000	150-200	1996	0.35	3.7	84	223	10	3.3	32	32	1024
R8000	75-90	1994	0.5	2.6	299	591	30	3.3	16	16	1024
R10000	150-250	1995	0.35	6.8	299	599	30	3.3	32	32	512
R12000	270-400	1998	0.18–0.25	6.9	204	600	20	4	32	32	1024
R14000	500-600	2001	0.13	7.2	204	527	17	--	32	32	2048
R16000	700-800	2002	0.11	--	--	--	20	--	64	64	4096

Tabla 1. Especificaciones MIPS

MIPS en el lanzamiento al mercado tuvo problemas financieros, sin embargo debido a la importancia del nuevo diseño, en 1992 SGI compró la empresa para evitar perder el diseño. Tras la compra, la compañía pasó a nombrarse MIPS Technologies.

A principios de la década de los 90, se logró simplificar el núcleo gracias a que MIPS comenzó a otorgar licencias de sus diseños a terceros, eso permitía que fuese utilizado para numerosas aplicaciones que anteriormente utilizaban la tecnología CISC. Durante esta época Sun Microsystems intentó otorgar licencias de su núcleo SPARC como MIPS sin el mismo éxito.

Terminando la década de los 90, MIPS consigue lograr desbancar con su CPU RISC a la famosa familia 68k de Motorola, este éxito hizo que SGI relanzara a MIPS Technologies en

1998, siendo más de la mitad de los ingresos procedentes de las concesiones de licencias y el resto procedente de contratos de diseño de núcleos para ser fabricados por terceros.

En 1999 MIPS asienta su sistema de licencias alrededor de dos diseños MIPS32 de 32 bits y el MIPS64 de 64 bits. Las primeras empresas en obtener las licencias fueron NEC, Toshiba, SiByte, Philips, LSI Logic e IDT. A medida que el tiempo paso se agrando su éxito y a día de hoy, los núcleos MIPS son uno de los pesos pesados del mercado de dispositivos como los portátiles o decodificadores y sintonizadoras de TV. Una de los indicios que nos hace ver su importancia es el ejemplo de Freescale que comienza a utilizar procesadores MIPS en sus aparatos en lugar de los suyos propios basados en el PowerPC.

Gracias al asentamiento del sistema de licencias comentado anteriormente MIPS atrae a numerosas nuevas compañías, una de ellas es Quantum Effects Devices (QED) fundada por antiguos ingenieros de MIPS.

Más adelante QED fue adquirida por el fabricante de semiconductores PMC-Sierra en 2000, siendo esta la última empresa en invertir en la arquitectura MIPS [7].

2.4 Simuladores

A continuación se describe las características principales de los simuladores utilizados como estudio para la creación de un simulador que compita como alternativa a los existentes.

2.4.1 SPIM

Es un simulador independiente que simula programas como si fuese un procesador MIPS32, este lee y ejecuta programas en lenguaje ensamblador escritos para este procesador. Spim también proporciona un depurador simple y un conjunto mínimo de servicios del sistema operativo, sin embargo, no ejecuta programas binarios (compilados). Implementa casi todo el conjunto de instrucciones de ensamblador extendida MIPS32 [9].

Es compatible con Microsoft Windows, Linux y Mac OS X.

El simulador SPIM funciona de tal forma que primero debes cargar un programa en ensamblador sobre el simulador y a partir de esta acción puedes realizar una serie de operaciones sobre él:

- Run, ejecuta el programa, se le puede añadir una dirección al comando para especificar que se desea ejecutar el programa a partir de esa dirección indicada.
- Step, ejecuta el programa un único ciclo por defecto, si se desea ejecutar más ciclos, se especifica en el comando la cantidad de ciclos a ejecutar.
- Continue, continúa con la ejecución sin realizar step.
- Print_all_regs, imprime por pantalla todos los registros, también los puede imprimir en hexadecimal con el comando print_all_regs_hex.
- Print addr, imprime el contenido de la dirección de memoria indicada.
- Print_sym, imprime los nombres y direcciones de las etiquetas globales que SPIM conoce.

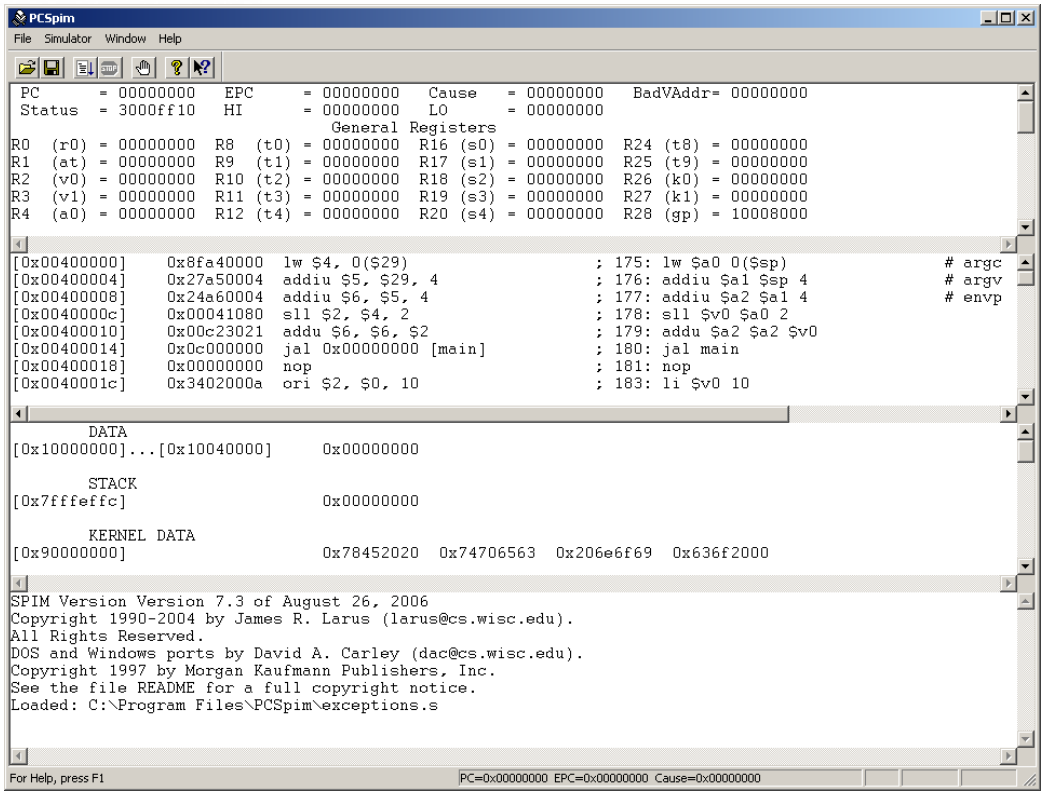


Ilustración 2. Simulador SPIM

2.4.2 WinDLX

Es un simulador del pipeline del procesador DLX, este es básicamente un MIPS revisado y simplificado con una arquitectura simple de almacenamiento de 32 bits.

Este simulador permite el procesamiento de programas escritos en ensamblador de DLX y muestra toda la información relevante de la CPU como el estado del pipeline, el banco de registros, las entradas/salidas, memoria, estadísticas, etc. Su versatilidad posibilita la modificación de la estructura y tiempos de latencia del pipeline de la CPU y del tamaño de la memoria, así como el contenido de otros de sus componentes mientras se desarrolla la ejecución de un programa [10].

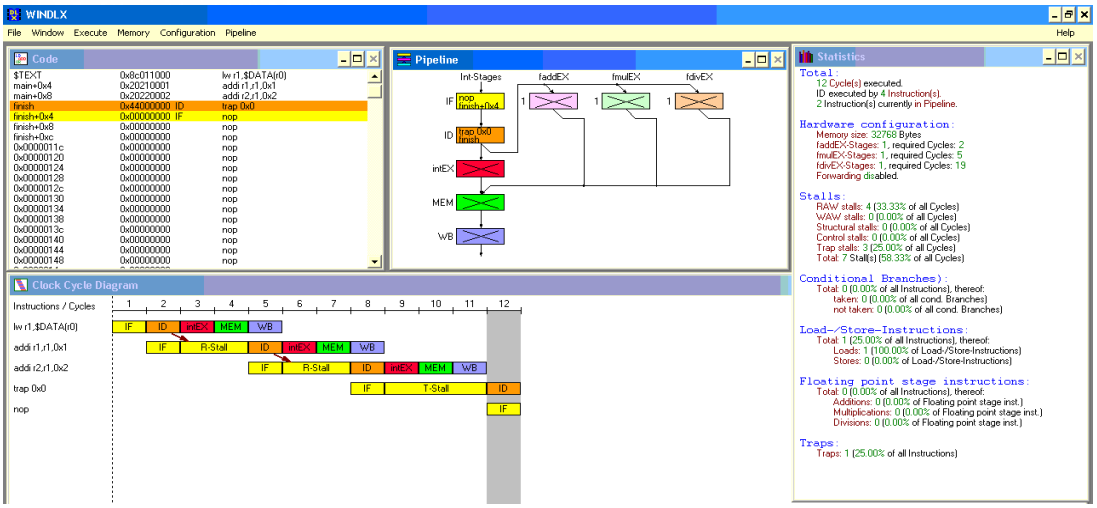


Ilustración 3. Simulador WinDLX

2.4.3 MIPSim

MIPSim es un simulador solo para Windows que muestra el recorrido de los datos a través de un pipeline dividido en cinco etapas. MIPSim incluye la unidad de seguimiento y la unidad de detección de riesgos. Además cuenta con una interfaz amigable. Este simulador contiene su propio subconjunto de instrucciones, lo que permite escribir pequeños programas que pueden ejecutarse sobre él, de esta manera el usuario puede ver el flujo de datos a través del pipeline, entender los riesgos de control y las soluciones que existen para ellos. Este simulador tiene una barra de herramientas que hace la interacción más fácil con el usuario y permite acceder a todas las funciones del simulador tales como crear código, abrir o guardar programas y ejecutar el programa de forma continua o ciclo a ciclo [11].

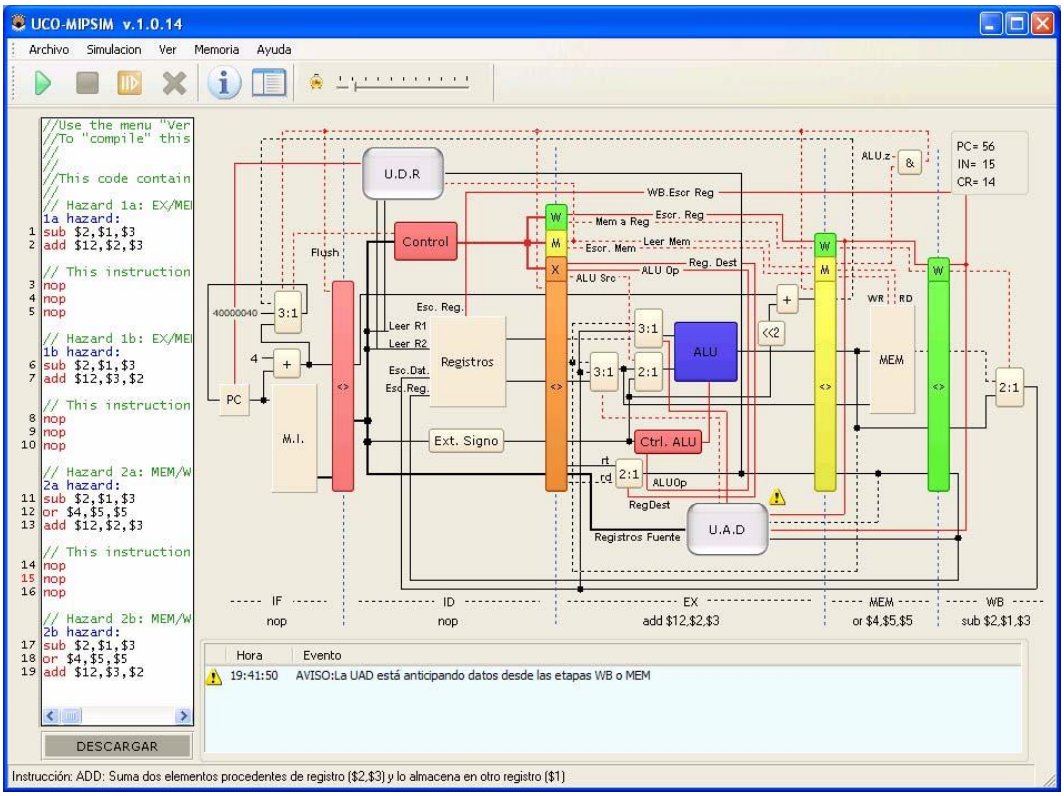


Ilustración 4. Simulador MIPSim

La instalación y uso de estos simuladores han ayudado a crear la idea inicial de cómo crear mi propio simulador de nanoprocasador para este proyecto intentando captar las mejores características de cada uno: la compatibilidad e instrucciones del SPIM, la posibilidad de mostrar toda la información relevante del estado de la CPU del WinDWLX y la posibilidad de mostrar el flujo de datos de forma gráfica mostrando el pipeline del simulador MIPSim.

3. Diseño inicial: nanoprocesador secuencial

A consecuencia del estudio previo expuesto en el apartado 2, en este apartado se muestra el diseño de la primera arquitectura que propongo para realizar el simulador del nanoprocesador de 24 bits y las estructuras que lo componen, dicho diseño será utilizado como base para la creación del simulador y para que el recorrido de los datos por él sea el más preciso posible.

3.1 Componentes

Para poder realizar el diseño inicial primero se estudian los diferentes componentes que formarán el nanoprocesador.

Como se podrá observar más adelante los componentes son muy parecidos a los de la arquitectura MIPS, sin embargo han sido adaptados para cumplir los requisitos que la empresa ha dictado y que se muestran en el apartado 4.2.

Memoria de instrucciones:

Elemento encargado de almacenar las instrucciones que conforman el programa, estas instrucciones son guardadas y suministradas en función de una dirección.

Las direcciones tienen un tamaño de 12 bits por lo que el nanoprocesador podrá almacenar programas de hasta 4096 instrucciones.

- Estructura:

  Para cada instrucción que se lee se necesita una entrada en la memoria de datos donde se especifique la dirección de memoria donde se encuentra. La memoria de instrucciones siempre devuelve a la salida la instrucción contenida en la dirección recibida como entrada.

  La entrada de la dirección de memoria *Dirección* es de 12 bits, como se ha explicado anteriormente, para direccionar a una de las 4096 instrucciones, mientras que el bus de salida de datos *Instrucción* es de 24 bits, esos datos de salida son instrucciones.

Ilustración 5. E/S Memoria de instrucciones

23 0	
000000000000001000000000	I-0000
000100000000000000100000	I-0001
000100100000000000000111	I-0002
000000011110000000000110	I-0003
…	…
000111111100000000011101	I-4093
100000111100010100001111	I-4094
000000000000000000000000	I-4095

Tabla 2. Estructura memoria de Instrucciones

- Instrucción:

A continuación se muestra el formato de instrucciones presente en esta arquitectura. En nuestro caso todas ellas tendrán un tamaño de 24 bits. Los campos de estas instrucciones se les atribuyen una seria de nombres para una fácil identificación:

- Opcode: Operación de la instrucción
- Rt: Registro destino donde se almacenará el resultado de la operación y primer registro operando fuente.
- Rs: Segundo registro operando fuente.
- Imm: Número constante inmediato.

Existen diferentes formatos de instrucción para las distintas clases de instrucciones:

· Instrucciones aritméticas y lógicas y de carga y almacenamiento:

-23-	**Formato(bits)**	**-0-**
Opcode(6)	RT(9)	RS(9)
23	17	8 0

Tabla 3. Formato instrucciones Aritméticas y Lógicas, de carga y almacenamiento

Ejemplos de uso:

add r7,r3

rt <- rt + rs

r7<-r7+r3

add 16	r7 7	r3 3
010000	000000111	000000011
23	17	8 0

Tabla 4. Ejemplo instrucción Aritmética

lw r5,r6

rt <- memory(rs)

r5<-memory(r6)

lw 8	r5 5	r6 6
001000	000000101	000000110
23	17	8 0

Tabla 5. Ejemplo instrucción de carga

· Instrucciones con operando inmediato:

-23-	**Formato(bits)**	**-0-**
Opcode(6)	RT(9)	IMM(9/4)
23	17	8 0

Tabla 6. Formato instrucciones con operando inmediato

Ejemplos de uso:

lwi r1,15

rt <- memory(imm)

r1<-memory(15)

lwi 9	r1 1	15
001001	000000001	00001111
23	17	8 0

Tabla 7. Ejemplo instrucción de carga con operando inmediato

addi r7,8

rt <- rt + imm

r7<-r7+8

addi 17	r7 7	8
010001	000000111	00000001000
23	17	8 0

Tabla 8. Ejemplo instrucción aritmética con operando inmediato

· Instrucciones de salto condicional:

-23-	Formato(bits)	-0-
Opcode(6)	IMM(12)	
23	17	0

Tabla 9. Formato de instrucciones de salto condicional

Ejemplos de uso:

be 8

Si Z=1 entonces PC=imm

Si Z=1 entonces PC= 8

be 3	8
000011	00000000000001000
23	17 0

Tabla 10. Ejemplo instrucción salto condicional

· Instrucciones de salto incondicional:

-23-	Formato(bits)	-0-
Opcode(6)	RS(9)	
23	17	0

Tabla 11. Formato instrucciones de salto incondicional

Ejemplos de uso:

br 8

PC=RS

PC= 8

br 1	8	
000001	00000000000001000	
23	17	0

Tabla 12. Ejemplo instrucción de salto incondicional

Contador de programa (PC):

Registro encargado de almacenar la dirección donde se encuentra el nanoprocesador en su secuencia de instrucciones.

El registro tiene una longitud igual al número de bits necesarios para representar la cantidad de direcciones de la memoria de instrucciones, en nuestro caso es de 12 bits.

Sumador:

Elemento encargado de incrementar el contador de programa para que éste apunte a la siguiente instrucción del programa. Este elemento está basado en una ALU el cuál es modificado para que sea un sumador permanente y no pueda realizar ninguna operación propia de una ALU. En el diseño del nanoprocesador es mostrado con una etiqueta “sumador” para ser diferenciado de la ALU.

Banco de registros:

Estructura contenedora de registros sobre los cuales se puede leer o escribir especificando su número identificador.

Los identificadores tienen un tamaño de 9 bits, por lo tanto el banco de registros del nanoprocesador puede albergar en él 512 registros.

- Estructura:

Para las instrucciones que posee el nanoprocesador es necesario al menos poder leer 2 registros y escribir en uno por cada instrucción. Para cada registro que se desea leer necesitamos una entrada dónde se especifique el identificador del registro, así como una salida a la que enviar el dato leído. En caso de querer escribir en un registro necesitamos dos entradas, una para especificar el identificador del registro donde queremos escribir y la segunda entrada para especificar el dato que se quiere escribir, esta escritura es controlada por la unidad de control, la cuál será explicada más adelante, la cual envía una señal de control de escritura.

No todas las entradas tienen el mismo tamaño, las estradas que sirven para especificar el identificador de un registro: *Leer Reg 1, Leer Reg2 y Dirección Reg*, tienen un tamaño de 9 bits, como se ha explicado anteriormente, para direccionar a uno de los 512 registros. Por otro lado, la entrada utilizada para escribir un dato en el bando de registros *Escribir Reg* tiene un tamaño de 24 bits, que es el tamaño de almacenamiento soportado por cada registro. En cuanto a las salidas, las dos sirven para enviar los datos contenidos en los registros leídos, por lo tanto su tamaño es de 24 bits, que como anteriormente se comentó es el tamaño de almacenamiento soportado por un registro.

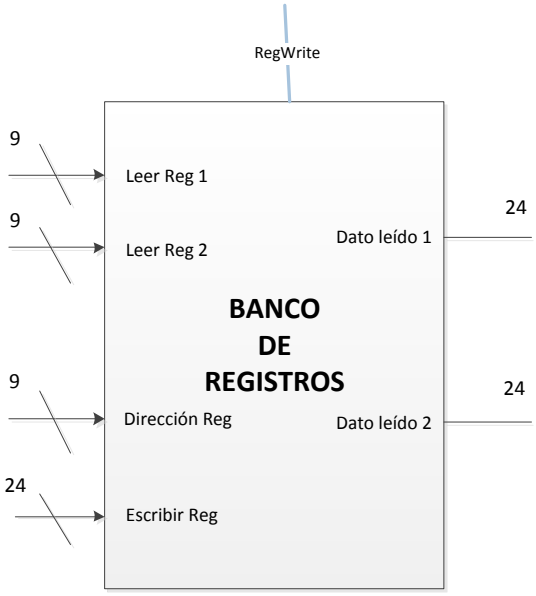


Ilustración 6. E/S Banco de registros

23 0	
00000000000000010000000000	R000
00010000000000000000100000	R001
00010010000000000000000111	R002
00000001111000000000000110	R003

…	…
000111111100000000011101	R509
100000111100010100001111	R510
000000000000000000000000	R511

Tabla 13. Estructura banco de registros

- Registro:

Memoria integrada en el nanoprocesador para almacenar información. Son utilizados para controlar las instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Para este nanoprocesador, su tamaño de almacenamiento es de 24 bits, coincidiendo a propósito con el tamaño de dirección de memoria. Los bits, por conveniencia se numeran de forma que el bit más significante se encuentra a la izquierda.

Unidad Aritmética Lógica (ALU):

Circuito digital capaz de calcular operaciones aritméticas y lógicas.

La ALU está compuesta por los siguientes componentes:

- Circuito operacional: Circuitos electrónicos necesarios para poder realizar las operaciones con los datos que la ALU recibe de los registros de entradas. Este circuito también tiene entradas de órdenes procedentes de la unidad de control las cuales envían que operación se debe realizar en ese momento.
- Registros de entrada: Registros donde se almacenan los datos que van a intervenir en la operación.
- Registro acumulador: Registro encargado de almacenar los valores temporales que se van produciendo en el circuito operacional.
- Registro de estado: Registro de memoria donde se almacenan los valores de Z, N y V de la última operación realizada. Estos registros son Zero(Z), cambiará su bit de 0 a 1 en caso de que el resultado sea 0; Negative(N), cambiará su bit de 0 a 1 en caso de que el resultado sea negativo y Overflow(V) cambiará su bit de 0 a 1 en caso de que el resultado supere el número de bits que la ALU puede manejar.

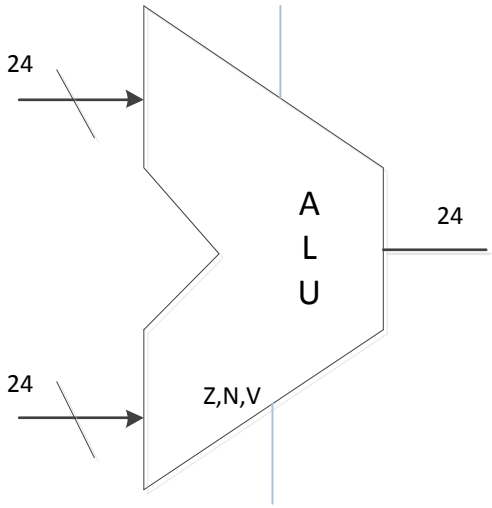


Ilustración 7. E/S Unidad Aritmético Lógica

Las operaciones de la ALU se pueden dividir en los siguientes tipos:

· Operaciones aritméticas:

- Suma
- Resta
- Comparación

· Operaciones lógicas:

- AND
- OR
- XOR

· Operaciones de desplazamiento de bits:

- Btest, modifica el valor del registro de estado Z en función del valor del bit de la primera entrada colocada en la i-esima posición indicada por la segunda entrada, en caso de ser 1 el valor del bit, el valor del registro de estado Z será 0, si es 0 el valor del bit el registro de estado Z será por tanto 1.
- Bset, esta operación consiste en modificar el dato recibido como primera entrada cambiando a 1 el valor del bit colocado en la i-esima posición indicada por la segunda entrada.
- Brst, esta operación consiste en modificar el dato recibido como primera entrada cambiando a 0 el valor del bit colocado en la i-esima posición indicada por la segunda entrada.

Memoria de datos

Elemento encargado de almacenar los datos que maneja el programa y que por tanto deben estar a disposición para su lectura y escritura.

Las direcciones tienen un tamaño de 16 bits por lo que el nanoprocesador podrá almacenar hasta 65535 datos.

- Estructura

La memoria de datos contiene como entradas la dirección de memoria a la que se desea acceder y el dato a escribir en ella y como salida el dato leído de la dirección recibida como entrada. Y recibe señales de control procedentes de la unidad de control que especifican si la instrucción es de lectura o escritura sobre la memoria de datos aunque solo una de las dos señales puede estar activa por cada acceso a memoria.

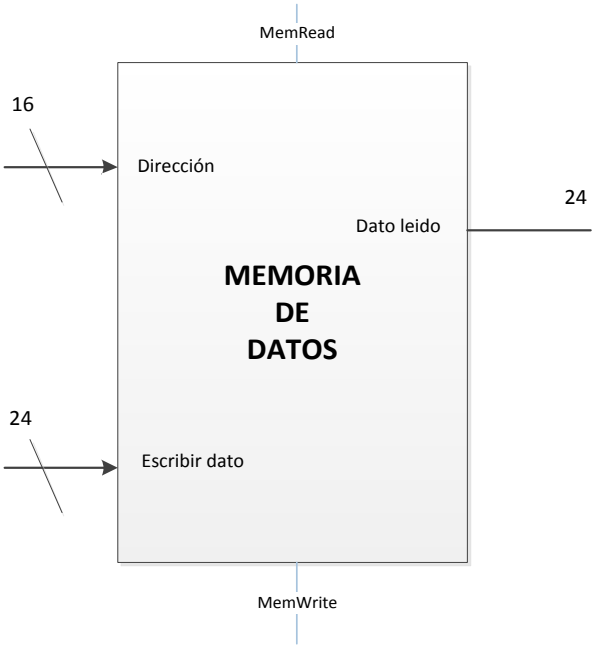


Ilustración 8. E/S memoria de datos

La entrada de la dirección de memoria *Dirección* es de 16 bits, como se ha explicado anteriormente, para direccionar a uno de las 65535 datos, mientras que la entrada de datos para su escritura en memoria *Escribir dato* es de 24 bits. El bus de salida de *Dato leído* es de 24 bits que es el tamaño en bits de un dato almacenado en la memoria de datos.

23 0	
000000000000001000000000	00000
000100000000000000100000	00001
000100100000000000000111	00002
000000011110000000000110	00003
...	...
000111111100000000011101	65533
100000111100010100001111	65534

00000000000000000000000000	65535

Tabla 14. Estructura memoria de datos

Multiplexor

Un multiplexor es un circuito combinacional con varias entradas pero una única salida de datos. La selección del dato de salida depende de la señal de control.

Los multiplexores utilizados para este diseño son los siguientes:

- Multiplexor que recibe como entrada los datos procedentes del banco de registros y valor inmediato de la instrucción y dependiendo de la señal procedente de la unidad de control si se trata de una instrucción inmediata devolverá a la salida el valor inmediato recibido a la entrada, de lo contrario si se trata de una instrucción de registro devolverá por la salida el valor procedente del banco de registros.
- Multiplexor que recibe como entrada el resultado de la ALU y un valor de la memoria de datos y dependiendo de la señal de control, devolverá por la salida el valor contenido en la memoria de datos en caso de que la señal sea de memoria a registro y en caso contrario devolverá por la salida el resultado de la ALU.
- Multiplexor que recibe como entrada la siguiente dirección de la memoria de instrucciones a tomar y la dirección que devuelve la instrucción de salto, la salida por lo tanto dependerá de la instrucción ejecutada si se cumple o no la condición, en caso de cumplir la condición, el multiplexor recibirá una señal para que devuelva a la salida la dirección del salto que la instrucción de salto indica y en caso contrario se devolverá la siguiente dirección que le correspondía tomar de la memoria de instrucciones.

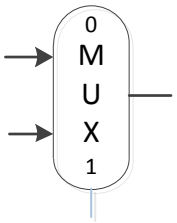


Ilustración 9. E/S Multiplexor

Unidad de Control (CU)

Núcleo del procesador, elemento encargado de dirigir y coordinar la mayoría las instrucciones. Para que esto sea posible envía las siguientes señales de control sobre los distintos elementos del sistema dependiendo del código de la operación de la instrucción que se esté ejecutando.

- Reg Write: Señal con destino al banco de registros la cual avisa de una escritura en el banco de registros.
- ALU src: Señal cuyo elemento destino es un multiplexor, al cual le indica si la instrucción en ejecución utiliza el valor de registro o un valor inmediato.
- ALU Op: Señal con destino al ALU la cual avisa del tipo de operación que la instrucción desea realizar.

- Mem Write: Señal con destino a la memoria de datos la cual avisa que se quiere realizar una escritura en la memoria de datos.
- Mem Read: Señal con destino a la memoria de datos la cual avisa que se quiere realizar una lectura en la memoria de datos
- Mem to Reg: Señal cuyo elemento destino es un multiplexor, al cúal le indica si la instrucción en ejecución utiliza el valor procedente de memoria o si toma el resultado procedente de la ALU.
- Branch: Señal con destino a una puerta lógica que en función de los registros de estado obtenidos de la ALU indica en su salida si debe tomar o no el salto.
-

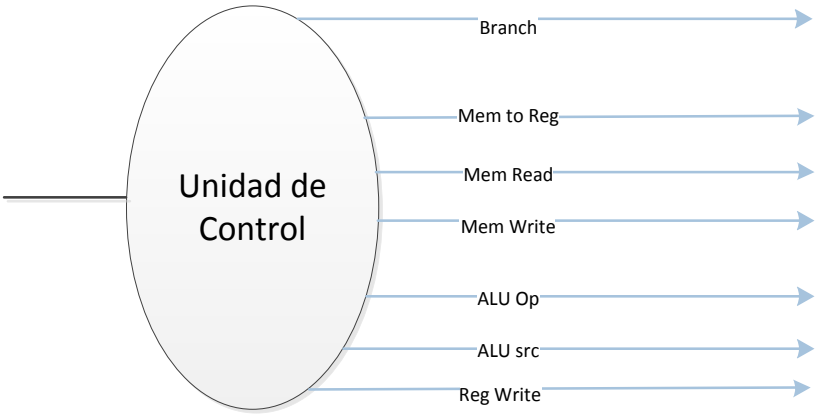


Ilustración 10. E/S Unidad de Control

A continuación se muestra el diseño inicial el cual, muestra la interconexión de los elementos principales anteriormente explicados necesarios para ejecutar cada instrucción del nanoprocesador.

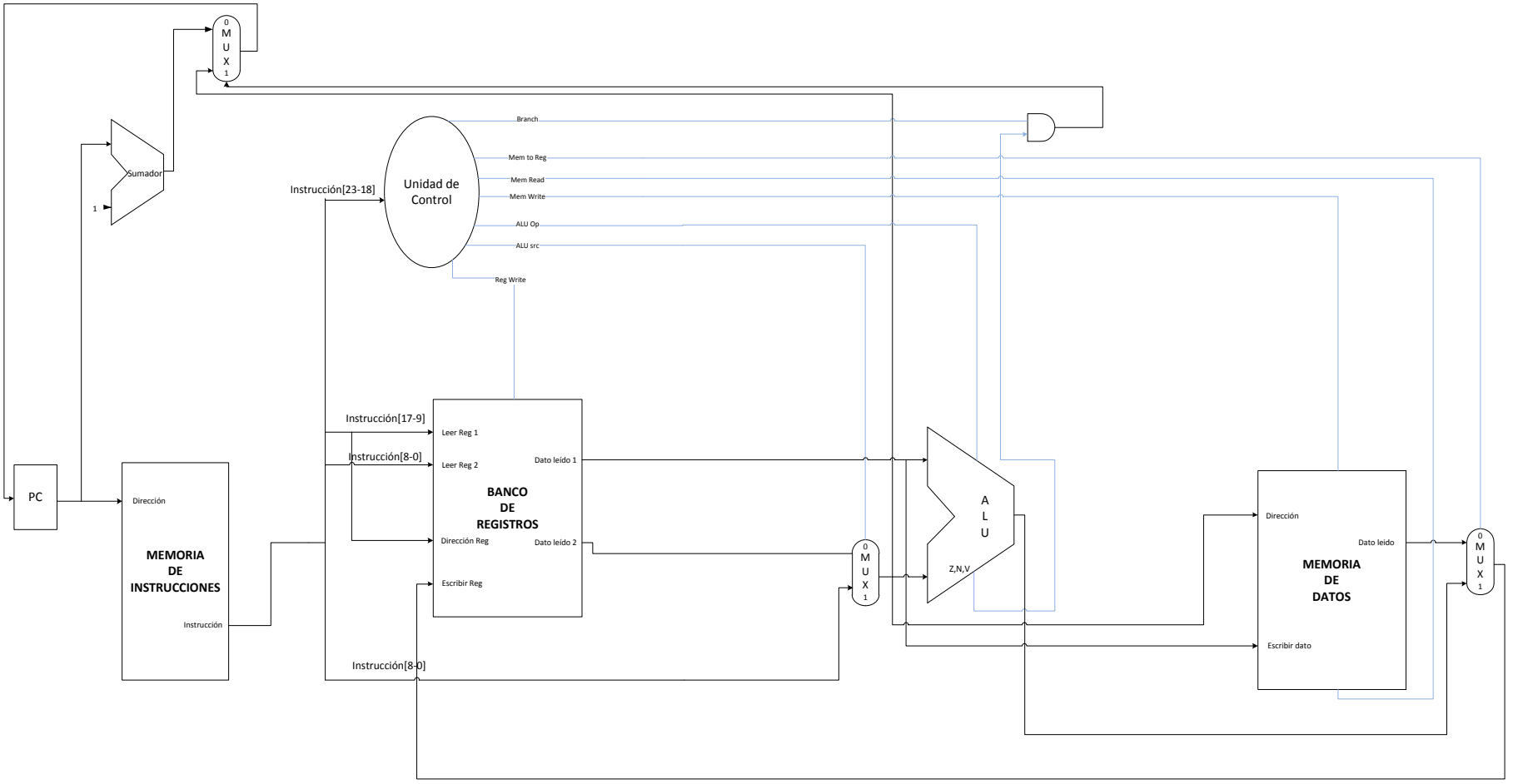


Ilustración 11. Nanoprocesador secuencial

4. Análisis del sistema

En este apartado se definen los casos de uso y los requisitos que el sistema a implementar debe cumplir.

4.1 Casos de uso

4.1.1 Diagrama de casos de uso

A continuación se muestra el diagrama de caso de uso que representará las necesidades que el sistema debe solucionar. Siendo el sistema el simulador del nanoprocesador y el actor el usuario final.

Ilustración 12. Diagrama Casos de Uso

4.1.2 Descripción textual de los casos de uso

El diagrama ofrece una información general pero no completa de la información necesaria que se quiere ofrecer. Antes de redactar los casos de uso, establecemos el formato que se seguirá para su especificación:

Identificador: CU-YYY	
Caso de uso	
Objetivo	
Precondiciones	
Postcondiciones	

Tabla 15. Formato casos de uso

· **Identificador:** Cada caso de uso incluirá una identificación para facilitar su traza por las fases subsiguientes, tal que CU-YYY, dónde YYY es el número correspondiente al caso de uso.

· **Caso de uso:** cómo un actor interactúa con el sistema y cuál es la respuesta que el sistema le ofrece.

· **Objetivo:** Finalidad del actor en el sistema.

· **Precondiciones:** Se definen las condiciones que deben cumplir para poder realizar una operación.

· **Postcondiciones**: Se define en qué estado queda el sistema tras realizar una operación.

A continuación se muestran todos los casos de uso descritos textualmente:

Identificador: CU-001	
Caso de uso	Carga de ficheros que componen el programa en ensamblador.
Objetivo	Cargar el programa en el simulador.
Precondiciones	Tener acceso al simulador.
Postcondiciones	Estructuras del simulador cargadas con los valores del programa.

Tabla 16. Caso de uso 1 – Carga de ficheros que componen el programa en ensamblador

Identificador: CU-002	
Caso de uso	Simular ciclo a ciclo el programa ensamblador.
Objetivo	Simular el programa en el nanoprocesador un único ciclo.

Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Valores en las estructuras modificadas en función del programa ensamblador.

Tabla 17. Caso de uso 2 – Simular ciclo a ciclo el programa ensamblador

Identificador: CU-003	
Caso de uso	Simular el programa ensamblador de forma continua.
Objetivo	Simular el programa en el nanoprocesador.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Valores en las estructuras modificadas en función del programa ensamblador.

Tabla 18. Caso de uso 3 – Simular el programa ensamblador de forma continua

Identificador: CU-004	
Caso de uso	Parar simulación en ejecución.
Objetivo	Poder parar la simulación cuando el usuario desee.
Precondiciones	Tener acceso al simulador, el programa cargado en el simulador y en ejecución.
Postcondiciones	Mostrar por terminal mensaje de advertencia si se desea parar la simulación, en caso de ser afirmativa la respuesta, la simulación debe parar.

Tabla 19. Caso de uso 4 – Parar simulación en ejecución

Identificador: CU-005	
Caso de uso	Reiniciar simulador.
Objetivo	Reiniciar los valores de las estructuras que componen el nanoprocesador antes de la carga de datos inicial.
Precondiciones	Tener acceso al simulador, las estructuras del nanoprocesador cargadas y el programa ejecutado.
Postcondiciones	Reiniciar el contenido de las estructuras que componen el nanoprocesador antes de la carga de datos inicial.

Tabla 20. Caso de uso 5 – Reiniciar simulador

Identificador: CU-006	
Caso de uso	Mostrar simulación.
Objetivo	Mostrar por terminal el pipeline de la simulación por ciclo del programa sobre el nanoprocesador.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Mostrar por terminal el pipeline a medida que la simulación avanza ciclo a ciclo. Valores en las estructuras son modificados en función del programa ensamblador.

Tabla 21. Caso de uso 6 – Mostrar simulación

Identificador: CU-007	
Caso de uso	Mostrar memoria de datos.
Objetivo	Mostrar el dato de una dirección especifica de la memoria de datos.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Mostrar por terminal el valor de una dirección especifica de la memoria de datos.

Tabla 22. Caso de uso 7 – Mostrar memoria de datos

Identificador: CU-008	
Caso de uso	Escribir en memoria de datos.
Objetivo	Escribir el dato en una dirección especifica de la memoria de datos.
Precondiciones	Tener acceso al simulador.
Postcondiciones	Valor del dato de la dirección introducida de la memoria de datos modificado.

Tabla 23. Caso de uso 8 – Escribir en memoria de datos

Identificador: CU-009	
Caso de uso	Leer memoria de datos.
Objetivo	Leer el dato de una dirección especifica de la memoria de datos.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Leer el valor de una dirección especifica de la memoria de datos.

Tabla 24. Caso de uso 9 – Leer memoria de datos

Identificador: CU-010	
Caso de uso	Mostrar banco de registros.
Objetivo	Mostrar el dato de un registro del banco de registros.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Mostrar por terminal el valor de un registro especifico del banco de registros.

Tabla 25. Caso de uso 10 – Mostrar banco de registros

Identificador: CU-011	
Caso de uso	Escribir en banco de registros.
Objetivo	Escribir el dato en registro especifico del banco de registros.
Precondiciones	Tener acceso al simulador.
Postcondiciones	Valor del dato del registro introducido del banco de registros modificado.

Tabla 26. Caso de uso 11 – Escribir banco de registros

Identificador: CU-012	
Caso de uso	Leer banco de registros.
Objetivo	Leer el dato de un registro especifico del banco de registros.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Leer el valor de un registro especifico del banco de registros.

Tabla 27. Caso de uso 12 – Leer banco de registros

Identificador: CU-013	
Caso de uso	Mostrar memoria de instrucciones.
Objetivo	Mostrar la instrucción de una dirección especifica de la memoria de instrucciones.
Precondiciones	Tener acceso al simulador y el programa cargado en el simulador.
Postcondiciones	Mostrar por terminal el valor de una dirección especifica de la memoria de instrucciones.

Tabla 28. Caso de uso 13 – Mostrar memoria de instrucciones

Identificador: CU-014	
Caso de uso	Mostrar registros de estado de ALU.
Objetivo	Mostrar el dato del registro de estado Z, N o V de la ALU.
Precondiciones	Tener acceso al simulador, el programa cargado en el simulador y programa ejecutado al menos 3 ciclos.
Postcondiciones	Mostrar por terminal el valor del registro de estado especificado de la ALU.

Tabla 29. Caso de uso 14 – Mostrar registros de estado de ALU

Identificador: CU-015	
Caso de uso	Leer registros de estado de ALU.
Objetivo	Leer el dato del registro de estado Z, N o V de la ALU que el usuario quiera.
Precondiciones	Tener acceso al simulador, el programa cargado en el simulador y programa ejecutado al menos 3 ciclos.
Postcondiciones	Leer por terminal el valor del registro de estado especificado de la ALU.

Tabla 30. Caso de uso 15 - Leer registros de estado de ALU

Identificador: CU-016	
Caso de uso	Mostrar información del simulador.
Objetivo	Mostrar lista de comandos del simulador e información de uso.
Precondiciones	Tener acceso al simulador.
Postcondiciones	Mostrar por pantalla listado de comandos o información del comando.

Tabla 31. Caso de uso 16 – Mostrar información del simulador

4.2 Especificación de requisitos

A continuación, se detallará todas las necesidades o condiciones que debe satisfacer el software. Estos requisitos son de necesidad obligatoria, dado que sobre ellos se va a cimentar el resto del proyecto, todos estos requisitos provienen del cliente, su prioridad es alta y su estabilidad ha de mantenerse durante todo el ciclo de vida de la aplicación. Estos requisitos es lo mínimo e indispensable que debe satisfacer la aplicación.

Antes de redactarlos, establecemos el formato que se seguirá para su especificación:

- **Identificador:** cada requisito software incluirá una identificación para facilitar su traza por las fases subsiguientes, tal que RX-YYY, dónde X es la primera inicial del tipo de requisito e YYY es el número correspondiente al requisito.
- **Necesidad:** Los requisitos esenciales de software se marcarán como tales. Los requisitos esenciales no son negociables. El resto pueden estar sujetos a negociación.
- **Prioridad:** Las medidas de prioridad serán alta, media o baja para que el desarrollador pueda decidir la planificación de la producción.
- **Fuente:** referencia al requisito o requisitos de usuario de los que parten los requisitos software, en mi caso alumno o tutores.
- **Estabilidad:** algunos requisitos se pueden saber fijos sobre la vida esperada del software, mientras que otros pueden depender de las decisiones de diseño o implementación que se tomen durante el desarrollo.
- **Descripción:** Descripción del requisito.
- **Claridad**: Medida en alta, media o baja según la claridad del requisito, si hay múltiples interpretaciones será baja, si hay dos media y si hay solo una alta.
- **Verificable:** Medido en alta, media o baja, debe ser posible que se pueda verificar que el requisito ha sido incorporado en el diseño y que se pueda demostrar que el software aplicar el requisito, por lo tanto, alta sería fácil de verificar, media no tan fácil y baja difícil de verificar.

Se hace distinción entre dos tipos de requisitos:

- De capacidad: su identificador será RC-XXX.
- De restricción: su identificador será RR-XXX.

Identificador:	
Prioridad: ☐ Alta ☐ Media ☐ Baja	**Fuente:** ☐ Tutores ☐ Alumno
Necesidad: ☐ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☐ Alta ☐ Media ☐ Baja	**Verificabilidad:** ☐ Alta ☐ Media ☐ Baja
Estabilidad:	
Descripción:	

Tabla 32. Formato requisitos

4.2.1 Requisitos de capacidad

A continuación se muestran los requisitos que representan que necesitan los usuarios para poder simular los programas en ensamblador.

Identificador:RC-001	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe poder ejecutarse tanto en tclsh como en programa ModelSim

Tabla 33. Requisito de capacidad 1 - Compatibilidad con tclsh y ModelSim

Identificador: RC-002	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe poder compilarse tanto en Linux como en Windows.

Tabla 34. Requisito de capacidad 2 - Compilación en Windows y Linux

Identificador: RC-003	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe simular el funcionamiento de un nanoprocesador en tres etapas: · Fetch · Decode · Exec

Tabla 35. Requisito de capacidad 3 - Simulador de un nanoprocesador con tres etapas

Identificador: RC-004	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	

Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El nanoprocesador a simular debe tener una memoria de datos.

Tabla 36. Requisito de capacidad 4 - El nanoprocesador debe contener una memoria de datos

Identificador: RC-005	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El nanoprocesador a simular debe tener un banco de registros.

Tabla 37. Requisito de capacidad 5 - El nanoprocesador debe contener un banco de registros

Identificador: RC-006	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El nanoprocesador a simular debe tener una memoria de instrucciones.

Tabla 38. Requisito de capacidad 6 - El nanoprocesador debe contener una memoria de instrucciones

Identificador: RC-007	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita leer un fichero .var y almacenar los datos leídos en la memoria de datos del nanoprocesador simulado.

Tabla 39. Requisito de capacidad 7 - Comando para lectura de ficheros .var

Identificador: RC-008	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita leer un fichero .reg y almacenar los datos leídos en el banco de registros del nanoprocesador simulado.

Tabla 40. Requisito de capacidad 8 - Comando para lectura de ficheros .reg

Identificador: RC-009	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita leer un fichero .obj y almacenar los datos leídos en la memoria de instrucciones del nanoprocesador simulado.

Tabla 41. Requisito de capacidad 9 - Comando para lectura de ficheros .obj

Identificador: RC-010	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El nanoprocesador a simular tendrá que poder ejecutar todas las instrucciones incluidas en el Anexo B de este documento.

Tabla 42. Requisito capacidad 10 - Nanoprocesador cumplirá con instrucciones de Anexo B

Identificador: RC-011	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	

Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador permitirá ejecutar la simulación por ciclos.

Tabla 43. Requisito capacidad 11 - Permitir simular por ciclos

Identificador: RC-012	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador permitirá ejecutar la simulación de manera continua.

Tabla 44. Requisito capacidad 12 - Permitir simular de forma continua

Identificador: RC-013	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El usuario podrá detener la simulación cuando desee.

Tabla 45. Requisito de capacidad 13 - Permitir detener la simulación

Identificador: RC-014	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que permitirá mostrar por pantalla la ejecución de la simulación.

Tabla 46. Requisito de capacidad 14 - Permitir mostrar por pantalla la ejecución

Identificador: RC-015	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que permitirá escribir sobre un registro del banco de registros en cualquier momento de la simulación.

Tabla 47. Requisito de capacidad 15 - Comando que permita escribir en el banco de registros

Identificador: RC-016	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que permitirá escribir un dato en la memoria de datos en cualquier momento de la simulación.

Tabla 48. Requisito de capacidad 16 - Comando que permita escribir en la memoria de datos

Identificador: RC-017	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que introduciendo un número de registro devolverá el valor de ese registro almacenado en el banco de registros en cualquier momento de la simulación.

Tabla 49. Requisito de capacidad 17 - Comando que permita devolver un dato del banco de registros

Identificador: RC-018	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja

Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que introduciendo una dirección devolverá el valor contenido en esa dirección en la memoria de datos en cualquier momento de la simulación.

Tabla 50. Requisito de capacidad 18 - Comando que permita devolver un dato de la memoria de datos

Identificador: RC-019	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que introduciendo un número de registro mostrará por pantalla el valor de ese registro almacenado en el banco de registros en cualquier momento de la simulación.

Tabla 51. Requisito de capacidad 19 - Comando que muestre por pantalla un dato del banco de registros

Identificador: RC-020	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que introduciendo una dirección permitirá mostrar por pantalla el valor en la memoria de datos de esa dirección en cualquier momento de la simulación.

Tabla 52. Requisito de capacidad 20 - Comando que muestre por pantalla un dato de la memoria de datos

Identificador: RC-021	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador tendrá un comando que introduciendo una dirección permitirá mostrar por pantalla la instrucción contenida en la memoria de instrucciones de esa dirección en cualquier momento de la simulación.

Tabla 53. Requisito de capacidad 21 - Comando que muestre por pantalla un dato de la memoria de instrucciones

Identificador: RC-022	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador cuando muestra por pantalla una instrucción imprime lo siguiente: · Valor en hexadecimal del dato · Instrucción desensamblada

Tabla 54. Requisito de capacidad 22 - Campos de una instrucción

Identificador: RC-023	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador permitirá introducir en los comandos que requieran escribir una dirección que estas estén en hexadecimal o en decimal.

Tabla 55. Requisito de capacidad 23 - Tipo de entrada de dirección

Identificador: RC-024	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador los valores de retorno de los comandos serán en decimal.

Tabla 56. Requisito de capacidad 24 - Tipo de dato de retorno

Identificador: RC-025	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	

Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando de ayuda que muestre por pantalla todos los comandos del simulador.

Tabla 57. Requisito de capacidad 25 - Comando ayuda listar comandos

Identificador: RC-026	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando de ayuda que muestre la información de cada comando.

Tabla 58. Requisito de capacidad 26 - Comando ayuda mostrar información de comando

Identificador: RC-027	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita reiniciar la simulación y devolver a 0 los datos de la memoria de datos, banco de registros y memoria de instrucciones.

Tabla 59. Requisito de capacidad 27 - Comando para reiniciar simulación

Identificador: RC-028	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador en la ejecución de la simulación debe mostrar lo siguiente: · Ciclo. · Etapas del procesador. · Instrucciones contenidas en las etapas en hexadecimal.

	· Instrucciones contenidas en las etapas desensambladas.

Tabla 60. Requisito de capacidad 28 - Mostrar campos al ejecutar

Identificador: RC-029	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita mostrar por pantalla los valores de los registros de estado del ALU (Z, N y V).

Tabla 61. Requisito de capacidad 29 – Comando mostrar registros de estado

Identificador: RC-030	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe tener un comando que permita devolver los valores de los registros de estado del ALU (Z, N y V).

Tabla 62. Requisito de capacidad 30 - Comando devolver registros de segmentación

4.2.2 Requisitos de restricción

A continuación se muestran las restricciones impuestas por la empresa sobre cómo se debe alcanzar el objetivo de realizar el simulador del nanoprocesador.

Identificador: RR-001	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe estar programado en lenguaje C y traducido a tcl.

Tabla 63. Requisito de restricción 1 - Simulador en lenguajes C y tcl

Identificador: RR-002	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	Para facilitar a los usuarios el uso del simulador, los comandos deben de ser intuitivos evitando añadir complejidad innecesaria.

Tabla 64. Requisito de restricción 2 – Comandos intuitivos

Identificador: RR-003	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El simulador debe responder al usuario por cada comando escrito si la ejecución se ha realizado con éxito.

Tabla 65. Requisito de restricción 3 - Feedback del simulador a usuario

Identificador: RR-004	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja
Estabilidad:	Durante toda la vida del sistema
Descripción:	El tiempo de ejecución de las distintas operaciones que se lleven a cabo en el simulador debe de mantenerse dentro de los límites aceptables.

Tabla 66. Requisito de restricción 4 - Tiempos de ejecución en límites aceptables

Identificador: RR-005	
Prioridad: ☑ Alta ☐ Media ☐ Baja	**Fuente:** ☑ Tutores ☐ Alumno
Necesidad: ☑ Esencial ☐ Deseable ☐ Opcional	
Claridad: ☑ Alta ☐ Media ☐ Baja	Verificabilidad: ☑ Alta ☐ Media ☐ Baja

Estabilidad:	Durante toda la vida del sistema
Descripción:	Se debe definir una batería de pruebas completa para verificar el correcto y completo funcionamiento del simulador. Estas pruebas se especificarán mediante una plantilla que indique: · Identificador de la prueba. · Descripción de la prueba. · Objetivo de la prueba, qué se pretende realizando esa prueba. · Procedimiento seguido. · Requisito que se pretende corroborar con la prueba. · Resultado esperado. · Resultado de la prueba (éxito o fracaso).

Tabla 67. Requisito de restricción 5 - Definición de pruebas

4.3 Matrices de trazabilidad

	CU-001	CU-002	CU-003	CU-004	CU-005	CU-006	CU-007	CU-008	CU-009	CU-010	CU-011	CU-012	CU-013	CU-014	CU-015	CU-016
RC-003		X	X			X										
RC-004	X	X	X				X	X	X							
RC-005	X	X	X							X	X	X				
RC-006	X	X	X										X			
RC-007	X															
RC-008	X															
RC-009	X															
RC-010		X	X													
RC-011		X														
RC-			X													

012																
RC-013				X												
RC-014						X										
RC-015											X					
RC-016								X								
RC-017												X				
RC-018									X							
RC-019										X						
RC-020							X									
RC-021													X			
RC-022						X							X			
RC-023							X	X	X	X	X	X	X			
RC-024							X		X	X		X		X	X	
RC-025																X
RC-026																X
RC-027					X											
RC-028						X										

RC-029														X		
RC-030															X	

Tabla 68. Matriz de trazabilidad casos de uso - requisitos

5. Propuesta de diseño e implementación del nanosimulador

En este apartado se muestra el procedimiento seguido para la realización del diseño definitivo del nanoprocesador, para ello se hace una evaluación de riesgos del nanoprocesador secuencial estudiado en el apartado 3 y el estudio de las posibles mejoras que se le pueden aplicar. Una vez desarrollado el diseño final, en este apartado se describirá el recorrido que los datos realizan sobre el nanoprocesador y la implementación del mismo como simulador describiendo las clases que lo forman y su relación entre ellas.

5.1 Evaluación y resolución de riesgos.

Hasta ahora el diseño era secuencial, es decir, hasta que una instrucción no termina su ejecución, no puede comenzar otra. Sin embargo, se busca diseñar un procesador que permita solapar temporalmente varias instrucciones, para ello, se lleva a cabo la segmentación del procesador.

¿Qué es la segmentación?, la segmentación o pipeline consiste en dividir una función en subfunciones independientes que se pueden realizar simultáneamente, esto permite que se traten distintos procesos a la vez aunque en fases distintas.

Como podemos ver en la tabla, el diseño sin segmentación, al calcular los tiempos medios de las tres etapas, el tiempo de acceso a la memoria de instrucciones es de 2ns, el de acceso al banco de registros 2ns y el de operación ALU junto el acceso a memoria de datos y escritura en registro es de 5ns. Al sumar las etapas, conseguimos 9 ns por la ejecución de una instrucción, por lo tanto, en caso de tener tres instrucciones el nanoprocesador consume 36 nanosegundos como se demuestra en la siguiente tabla:

Instrucción/tiempo(ns)	2	4	9	11	13	18	20	22	27	29	31	36
0. add r1, r2	Instrucción 0											
1. add r7, r8				Instrucción 1								
2. add r3, r4							Instrucción 2					
3. sub r5, r6										Instrucción 3		

Tabla 69. Ejemplo ejecución nanoprocesador secuencial

Sin embargo, si utilizamos la segmentación, cada ciclo debe de ejecutarse en el mismo tiempo que la etapa más lenta sin segmentar, en este caso la tercera etapa consume 5 ns, como podemos ver en la tabla, las instrucciones recorres tres etapas debido al cumpliendo del requisito Tabla 35. Requisito de capacidad 3 - Simulador de un nanoprocesador con tres etapas, estas etapas son IF que corresponde a la etapa Fetch, ID que corresponde a la etapa Decode y EX que corresponde a la etapa Exec, el funcionamiento de estas etapas se explican más adelante:

Instrucción/tiempo(ns)	5	10	15	20	25	30
1. add r1, r2	IF	ID	EX			
2. add r7, r8		IF	ID	EX		
3. add r3, r4			IF	ID	EX	
4. sub r5, r6				IF	ID	EX

Tabla 70. Ejemplo ejecución nanoprocesador secuencial

Si observamos la última tabla ganamos 6 ns en la ejecución, utilizando un nanoprocesador segmentado por cada ejecución de 3 instrucciones, por lo tanto, a más instrucciones más se incrementará la diferencia de tiempo respecto al nanoprocesador sin segmentar.

Sin embargo no solo incrementamos la velocidad, también el rendimiento, ya que se aumenta la productividad al poder realizar más de una instrucción en un mismo ciclo.

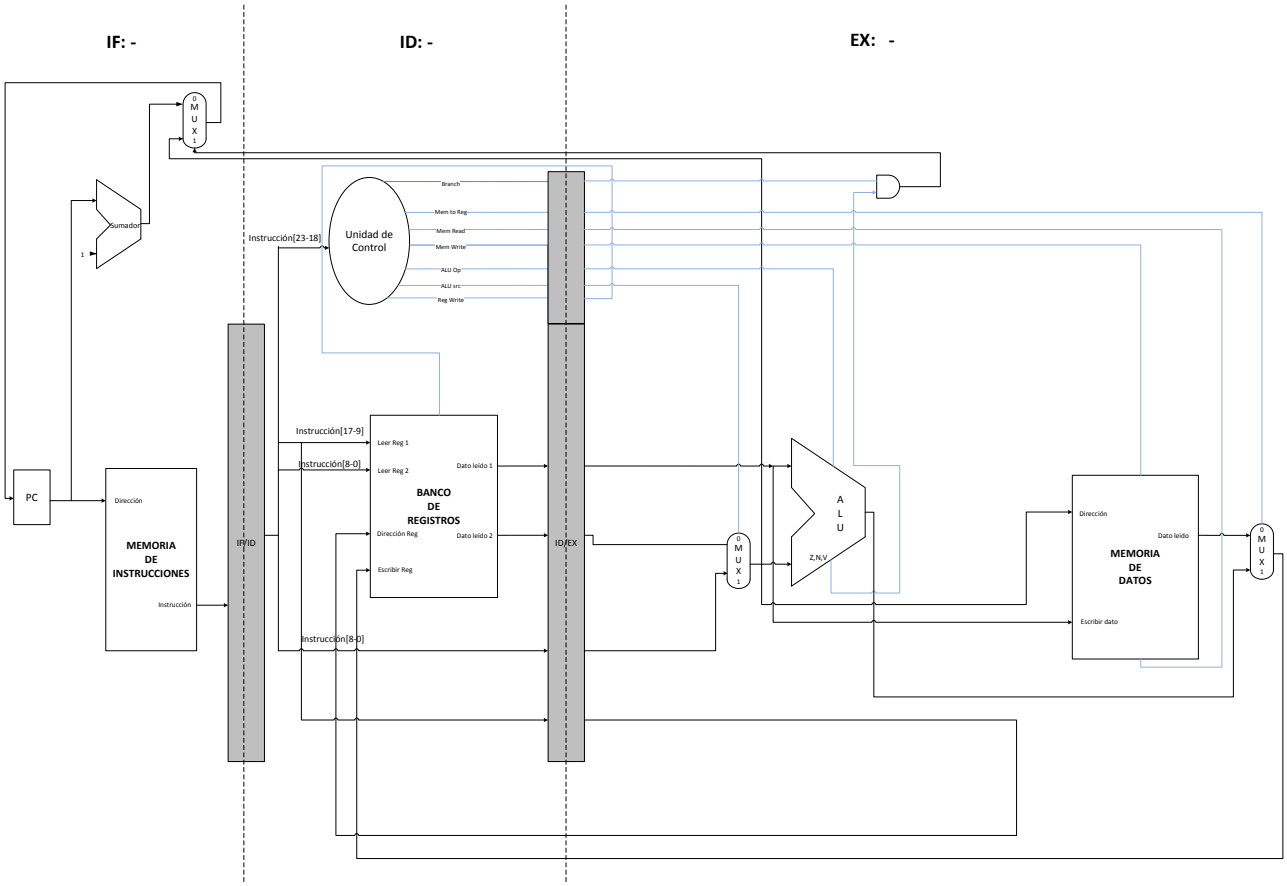


Ilustración 13. Nanoprocesador con registros de segmentación

Con la segmentación aparecen los registros de segmentación, esto son registros situados entre etapas que almacenan la información de todas las instrucciones que recorren las mismas, y a medida que avanzan por el pipeline se va propagando el control de las transferencias asociado a cada instrucción.

Sin embargo con la segmentación aparecen los siguientes riesgos en el pipeline:

Riesgos estructurales

Este tipo de riesgo ocurre cuando más de una instrucción intenta acceder a un mismo recurso simultáneamente.

Soluciones:

- Memoria de datos e instrucciones separadas para evitar conflictos en el acceso

Nuestro diseño al tratarse de tres etapas, podría darse que en el mismo ciclo una instrucción este accediendo a memoria para escribir un dato (etapa EX) y otra a la vez para tomar una instrucción(etapa IF), por ello se ha diferenciado y creado dos memorias, una para instrucciones y otra para datos.

- Lectura y escritura simultánea en el banco de registros

Es posible que en el mismo ciclo una instrucción en la etapa de ejecución se esté escribiendo un nuevo valor en el registro y que otra instrucción en la etapa de decodificación esté haciendo una lectura del mismo u otro registro del banco de registros.

Riesgos de datos

Este tipo de riesgo ocurre cuando una instrucción requiere un dato generado por la anterior instrucción en la etapa de ejecución.

Ejemplo:

0. add r1, r2 r1<- r1+r2
1. add r1,r 5 r1<- r1+r5

La instrucción 1 requiere el valor de r1 para sumar al valor de r5 y por lo tanto se produce el riesgo de tomar el valor antiguo de r1 para la suma.

Este riesgo puede clasificarse en tres categorías:

· RAW (Read After Write): Se produce cuando la segunda instrucción lee un dato antes de que la primera lo genere.

· WAR(Write After Read): Se produce cuando la segunda instrucción escribe antes de que la primera lo lea.

· WAW(Write After Write): Se produce cuando la segunda instrucción escribe un operando antes de ser escrito por la primera.

En nuestro diseño no encontramos los riesgos WAR Y WAW ya que su origen se encuentra en las ejecuciones fuera de orden y este nanoprocesador no tiene planificación dinámica(el hardware no reorganiza las instrucciones para reducir detenciones) por lo que no se producen este tipo de ejecuciones.

Solución para riesgo RAW:

- Crear un camino adicional de Hardware

Utilizar una unidad de anticipación, elemento que permite enviar el resultado obtenido en una instrucción a las instrucciones que lo necesitan como operando. Su funcionamiento es el siguiente:

Como entrada toma la dirección de los registros RT y RS utilizados en la etapa ID, la dirección del registro utilizado en la etapa EX como destino y el resultado que se quiere añadir al registro destino de EX.

En caso de coincidir la dirección de alguno de los registros en la etapa de decodificación con el registro modificado en ejecución modifica un flag de coincidencia a 1 o 2 distinguiendo con que registro es con el que coincide (RT=1, RS=2) y en caso de no coincidir deja el flag de coincidencia a 0.

De tal forma que cuando la instrucción que está en la etapa de decodificación avanza a la etapa de ejecución pregunta por la variable flag de coincidencia. En caso de ser 1 toma el resultado de la ejecución anterior que se ha escrito en el banco de registros como el valor de RT; En caso de ser 2 toma el resultado de la ejecución anterior que se ha escrito en el banco de registros como el valor de RS y por último, en caso de ser 0 toma el valor de ID/EX obtenido de las direcciones de registros correspondientes.

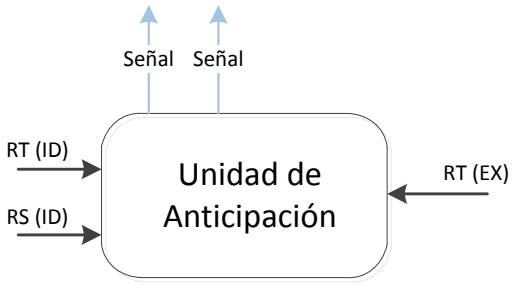


Ilustración 14. E/S Unidad de Anticipación

La introducción de este nuevo elemento en el diseño conlleva añadir dos multiplexores más en la etapa de ejecución antes de la ALU que controlen si el dato de salida de los multiplexores es el valor procedente del registro del banco de registros o el resultado de la anterior ejecución, esta salida es controlada a partir de la señal de control que envía la unidad de anticipación.

Riesgos de control

Este tipo de riesgo ocurre con las instrucciones de salto. Se produce cuando una instrucción que modifica el valor del PC todavía no lo ha hecho cuando se tiene que comenzar la ejecución de la siguiente instrucción.

Solución:

Flujos múltiples, durante la ejecución de la instrucción se siguen los dos caminos posibles, por lo tanto conllevaría la duplicación del hardware.

Salto retardado, modificación del ciclo de la instrucción, esto requiere reordenar el código por parte del compilador. En este caso la instrucción siguiente al salto siempre se ejecuta y en caso de cumplirse la situación de salto se añade una instrucción de nop.

Predicción de salto, consiste en predecir de antemano el si el salto será tomado o no, existen dos tipos de predicción:

Estática: Dentro de este campo existen varias posibilidades, la predicción de tomar todos los saltos, tomar saltos en función del código de operación y tomar saltos en función de su dirección (ejemplo: saltos hacia atrás tomarlos y los saltos hacia delante no).

Dinámica: Se asigna a cada instrucción de salto uno o dos bits de estado y en función del desarrollo del programa en la ejecución toma o no el salto.

Ejemplo con 1 bit:

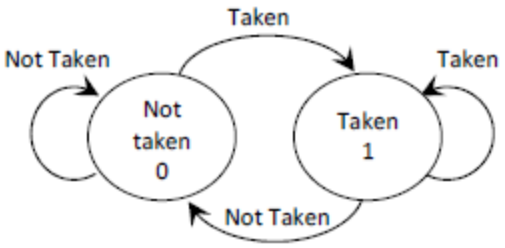


Ilustración 15. Predicción de salto con 1 bit

Ejemplo con 2 bits:

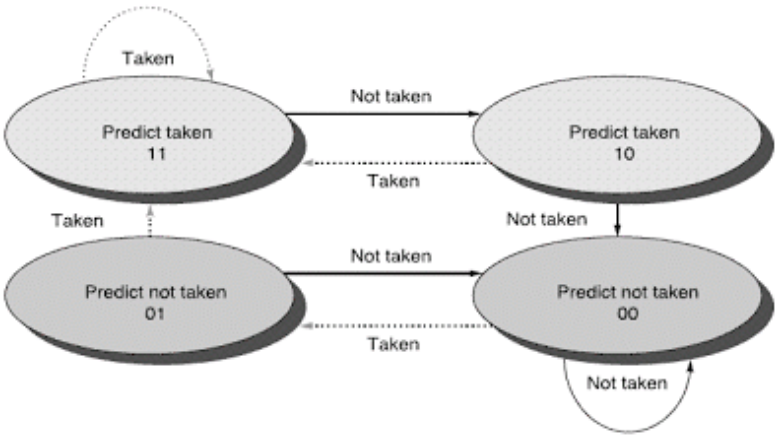


Ilustración 16. Predicción de salto con 2 bits.

Para este diseño se opta para resolver el riesgo predecir que no se realiza el salto y en caso de tomarse añadir un instrucción nop para que la etapa instrucción Fetch tome la dirección correcta tras la ejecución del salto.

Ejemplo de salto no tomado:

Instrucción/Ciclo	1	2	3	4	5	6	7
5. add r1, r2	IF	ID	EX				
6. beq 3		IF	ID	EX			
7. add r3, r4			IF	ID	EX		
8. sub r5, r6				IF	ID	EX	

Tabla 71. Ejemplo de salto no tomado

Ejemplo de salto tomado:

Instrucción/Ciclo	1	2	3	4	5	6	7
9. add r1, r2	IF	ID	EX				
10. beq 3		IF	ID	EX			
11. add r3, r4			IF	-			
12. sub r5, r6				-	IF	ID	EX

Tabla 72. Ejemplo de salto tomado

5.2 Diseño del nanoprocesador segmentado

Por consiguiente el diseño tras haber evaluado los riesgos y sus posibles soluciones será el mostrado a continuación:

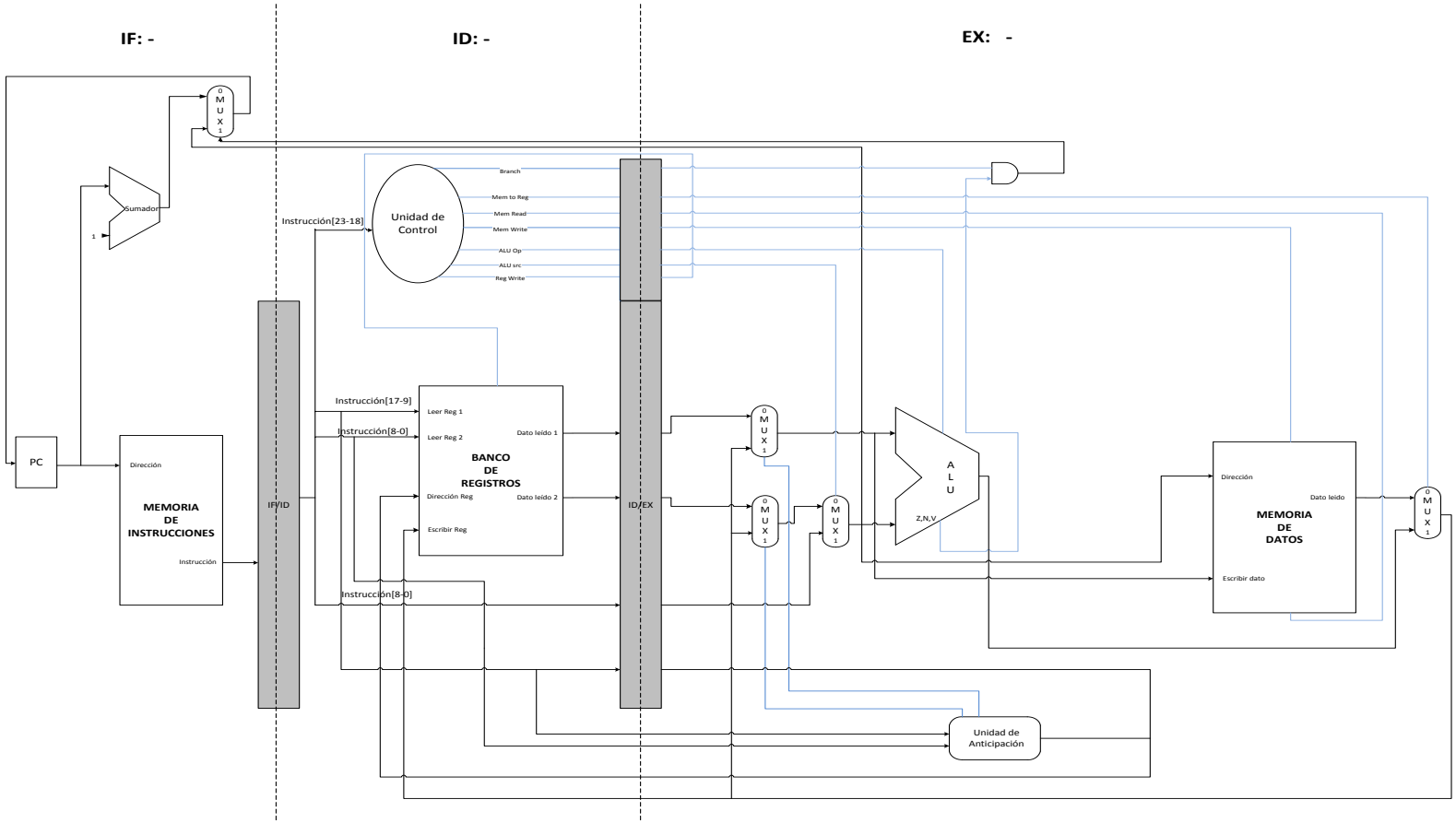


Ilustración 17. Nanoprocesador segmentado con unidad de anticipación

Como se puede observar en la Ilustración 17. Nanoprocesador segmentado con unidad de anticipación, la arquitectura varía muy poco respecto al diseño secuencial. La diferencia la encontramos en la presencia en este nuevo diseño de los registros de segmentación separando las tres etapas (Fetch, Decode y Exec) y en la inclusión de la unidad de anticipación.

5.2.1 Etapas

Como se ha comentado en el anterior apartado, el nanoprocesador queda dividido en tres etapas:

- Fetch: Etapa del procesador encargada de tomar las instrucciones que se encuentran en la dirección de la memoria de instrucciones que apunta el contador de programa e incrementa en uno el contador de programa.
- Decode: Etapa encargada de dividir la instrucción recibida de la anterior etapa en campos de bits distinguiendo entre el código de operación, número de registros y numero constante inmediato. El código de operación es enviado a la unidad de control la cual decidirá el camino de los datos en función del código de operación recibido. En caso de ser una instrucción que utilice registros envía el número de registro a la siguiente etapa y su valor obtenido del banco de registros. En caso de ser una operación con número constante inmediato envía su valor a la siguiente etapa.
- Exec: Etapa encargada de realizar distintas operaciones en función de los datos recibidos de la anterior etapa, estas operaciones pueden ser:
    - Almacenar un dato recibido del banco de registros en la memoria de datos.
    - Cargar un dato de la memoria de datos sobre un registro del banco de registros.
    - Realizar operaciones aritméticas y lógicas y almacenar su resultado en el banco de registros.
    - Realizar salto condicional modificando la dirección del contador de programa en función de cumplirse o no la condición.

En los siguientes apartados veremos en profundidad el camino que recorren los datos por cada etapa en función del tipo de instrucción.

5.2.2 Recorrido de datos sobre el nanoprocesador

A continuación se muestra el camino que siguen los datos por el nanoprocesador en función del tipo de instrucción ejecutada.

5.2.2.1 Instrucciones de carga

Las instrucciones Load son instrucciones encargadas de cargar un dato procedente de la memoria de datos sobre el registro indicado del banco de registros. Para un correcto funcionamiento, la instrucción debe recorrer las distintas etapas del pipeline a través del siguiente recorrido:

1. Búsqueda de Instrucción.

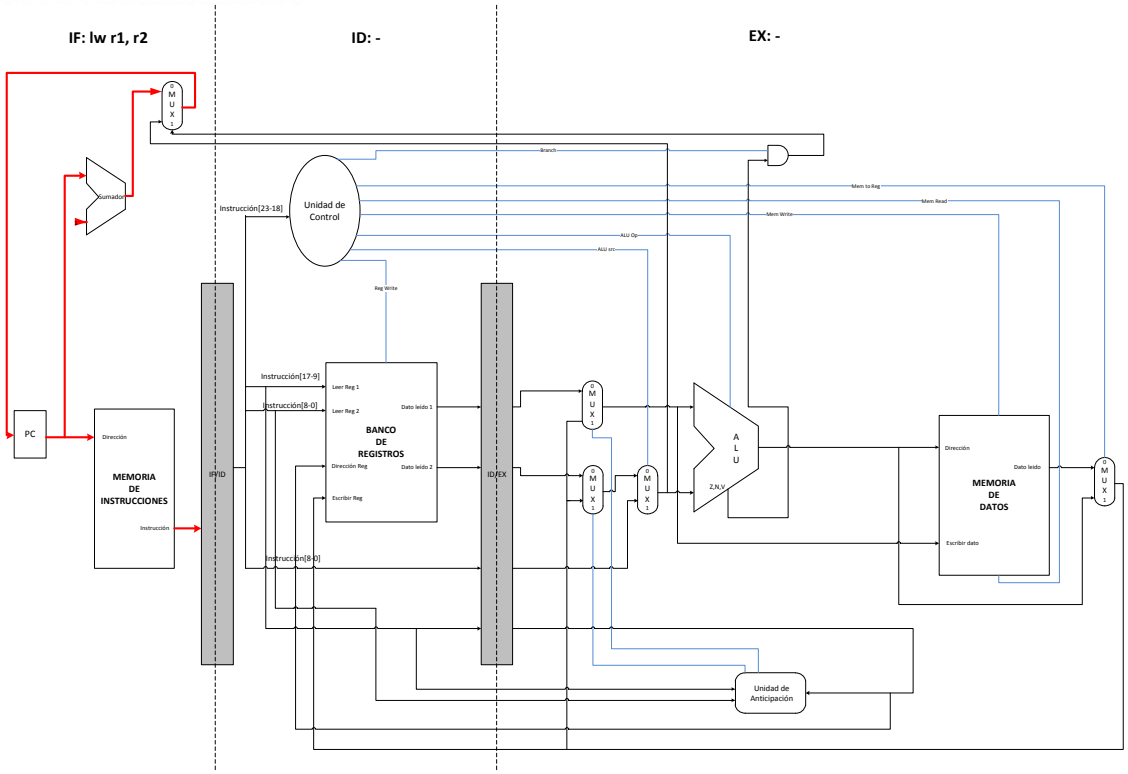


Ilustración 18. Etapa IF de Instrucción de carga

Como observamos en la Ilustración 18. Etapa IF de Instrucción de carga, esta etapa lee la instrucción contenida en la dirección de la memoria de instrucciones indicada por el PC y la coloca en el registro de segmentación IF/ID.

Como podemos ver, en esta etapa también se incrementa la dirección del PC y se guarda su resultado en el PC, salvo que se haya ejecutado una instrucción de salto y por lo tanto tenga que tomar la dirección proporcionada por el salto en lugar de la dirección incrementada, este valor es colocado en el registro de segmentación IF/ID.

2. Decodificación de la instrucción y lectura del banco de registros.

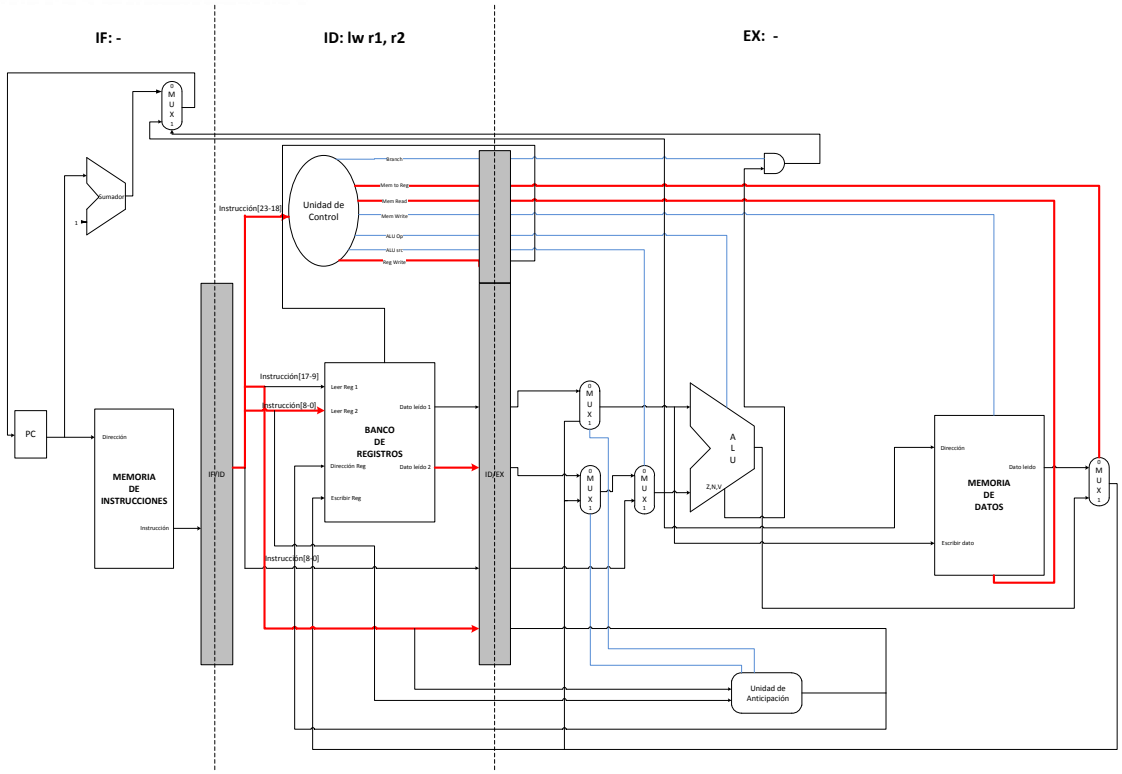


Ilustración 19. Etapa ID de Instrucción de carga

Como observamos en la Ilustración 19. Etapa ID de Instrucción de carga, esta etapa toma la instrucción del registro de segmentación IF/ID y la decodifica tomando los bits [23-18] para interpretar el tipo de operación que la instrucción realiza, estos bits son enviados a la unidad de control y la señal resultante se coloca en el registro de segmentación ID /EX, en este caso se tratan de señales que indican que habrá una operación de lectura de la memoria de datos y una escritura sobre el banco de registros; los bits [17-9] son utilizados para conocer cuál es el identificador del registro destino del banco de registros sobre el cuál será almacenado el valor obtenido de la memoria de datos, este identificador de registro es colocado en el registro de segmentación ID/EX, y los bits [8-0] son utilizados dependiendo de la instrucción para indicarnos cuál es el registro del cual hay que tomar el valor como dirección de memoria de datos, o bien como valor numérico que será utilizado como dirección de la memoria de datos, en ambos casos, el resultado es enviado al registro de segmentación ID/EX.

Como podemos ver, en esta etapa se coloca en el registro de segmentación ID/EX la dirección del PC que el registro de segmentación IF/ID contiene.

3. Acceso a memoria de datos y escritura en registro de banco de registros.

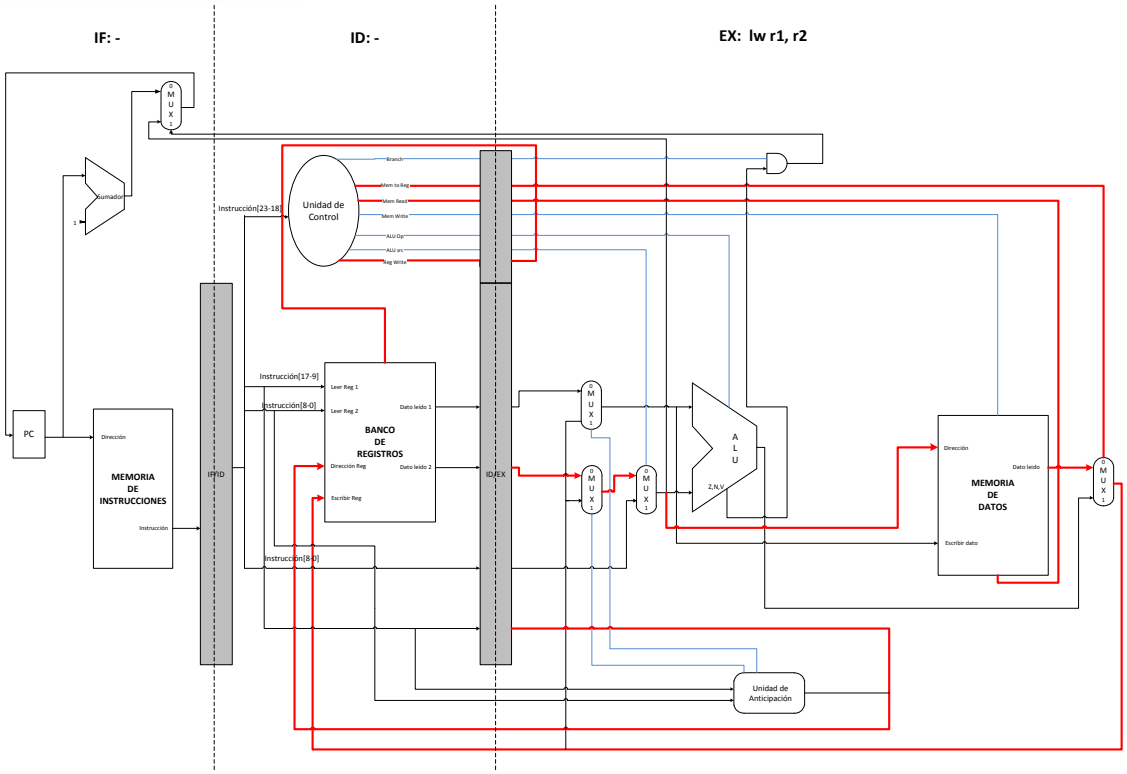


Ilustración 20. Etapa EX de Instrucción de carga

Como podemos observar en la Ilustración 20. Etapa EX de Instrucción de carga, disponemos de las siguientes entradas procedentes del registro de segmentación ID/EX:

- Registro origen o valor numérico.
- Dirección destino del banco de registros.
- Señales procedentes de la unidad de control.

Estas entradas tienen las siguientes finalidades en esta etapa:

El valor del registro origen o bien el valor numérico es utilizado como dirección de la memoria de datos para leer el valor de tal dirección. Para que esta lectura sea posible, utilizamos la señal procedente de la unidad de control Mem Read, que permite la lectura sobre la memoria de datos. Una vez leído el dato, se procede a guardarlo sobre la dirección destino del banco de registros. Para que esta escritura sea posible, utilizamos la señal procedente de la unidad de control Reg Write, la cual permite la escritura sobre el banco de registros.

5.2.2.2 Instrucciones de almacenamiento

Las instrucciones Store son instrucciones encargadas de almacenar en una dirección de la memoria de datos un valor contenido en un registro de origen o un valor numérico. Para un correcto funcionamiento, la instrucción debe recorrer las distintas etapas del pipeline a través del siguiente recorrido, muy similar al descrito anteriormente:

1. Búsqueda de Instrucción.

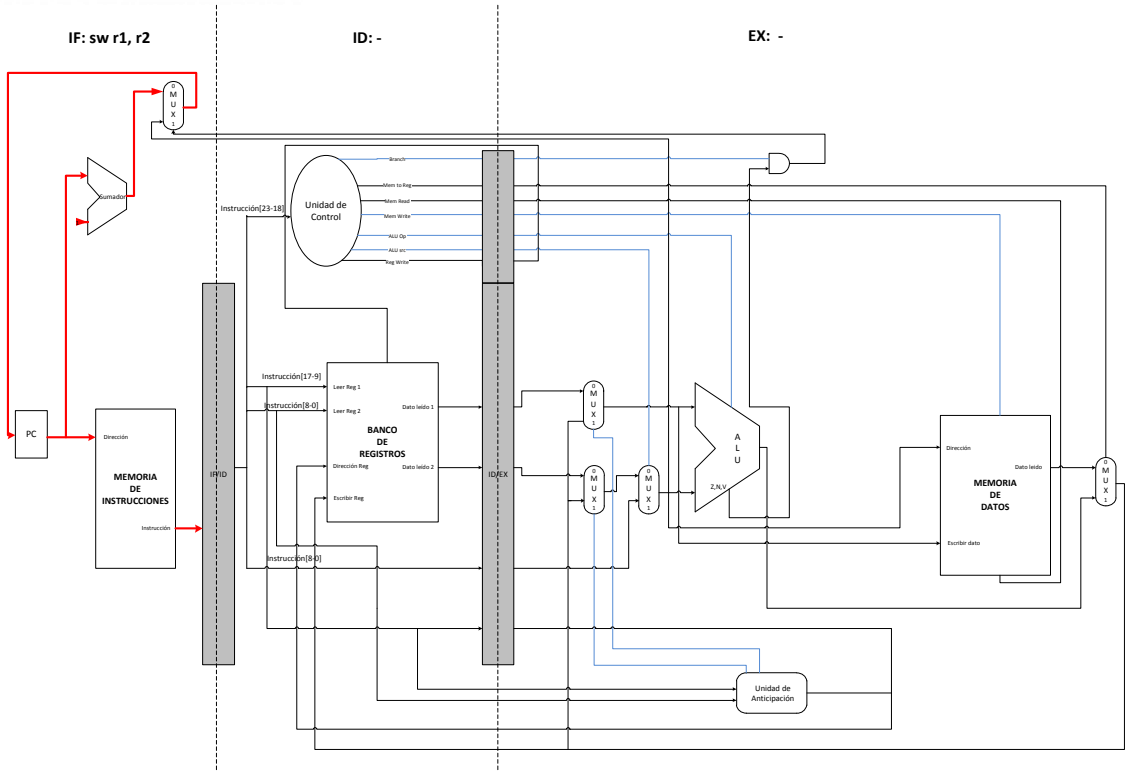


Ilustración 21. Etapa IF de Instrucción de almacenamiento

Como observamos en la Ilustración 21. Etapa IF de Instrucción de almacenamiento, esta etapa lee la instrucción contenida en la dirección de la memoria de instrucciones indicada por el PC y la coloca en el registro de segmentación IF/DEC.

Como podemos ver, en esta etapa también se incrementa la dirección del PC y se guarda su resultado en el PC, salvo que se haya ejecutado una instrucción de salto y por lo tanto tenga que tomar la dirección proporcionada por el salto en lugar de la dirección incrementada, este valor es colocado en el registro de segmentación IF/ID.

2. Decodificación de la instrucción y lectura del banco de registros.

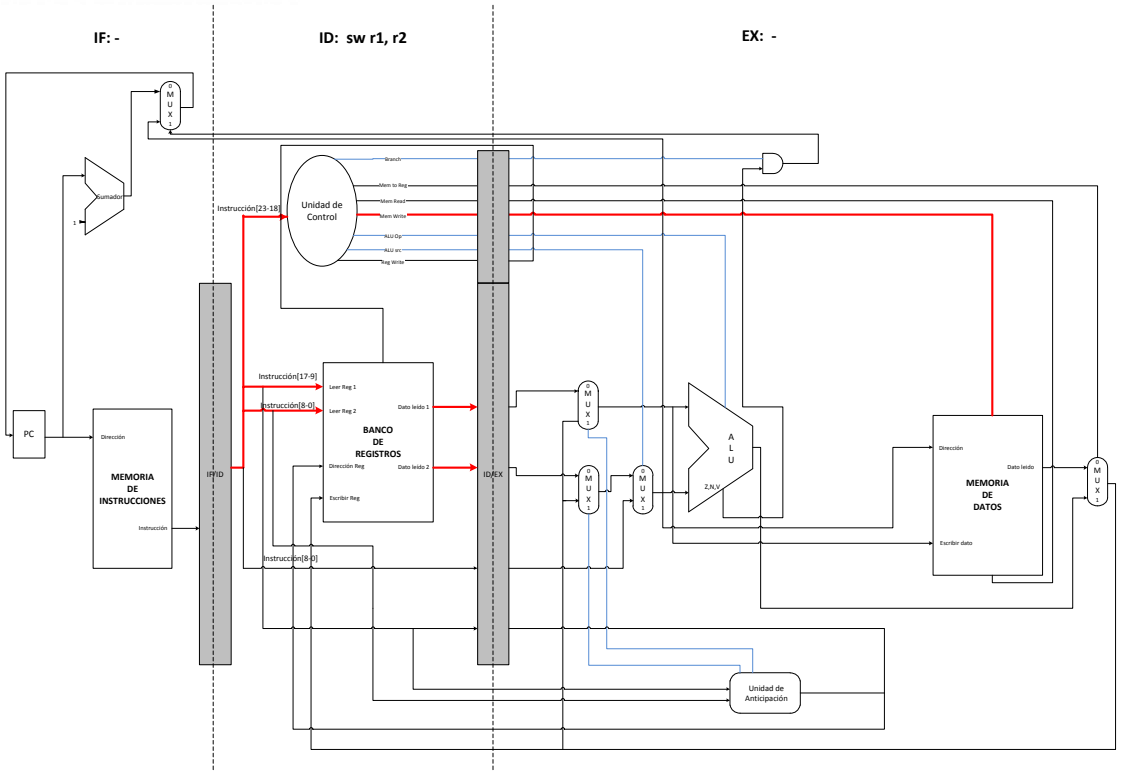


Ilustración 22. Etapa ID de Instrucción de almacenamiento

Como observamos en la Ilustración 22. Etapa ID de Instrucción de almacenamiento, esta etapa toma la instrucción del registro de segmentación IF/ID y la decodifica tomando los bits [23-18] para interpretar el tipo de operación que la instrucción realiza, estos bits son enviados a la unidad de control y el resultado se coloca en el registro de segmentación ID/EX en este caso se trata de una operación de escritura en la memoria de datos; los bits [17-9] son utilizados para conocer cuál es el registro origen del banco de registros sobre el cuál se quiere extraer el dato para su posterior almacenamiento en la memoria de datos, esta información obtenida del registro del banco de registros es colocada en el registro de segmentación ID/EX; y los bits [8-0] son utilizados dependiendo de la instrucción: para indicarnos cuál es el registro del cual hay que tomar el valor como dirección de memoria de datos, o bien como valor numérico que será utilizado como dirección de la memoria de datos sobre la que se desea almacenar una nueva información, en ambos casos, el resultado es enviado al registro de segmentación ID/EX.

Como podemos ver, en esta etapa se coloca en el registro de segmentación ID/EX la dirección del PC que el registro de segmentación IF/ID contiene.

3. Acceso a memoria de datos y escritura en registro de banco de registros.

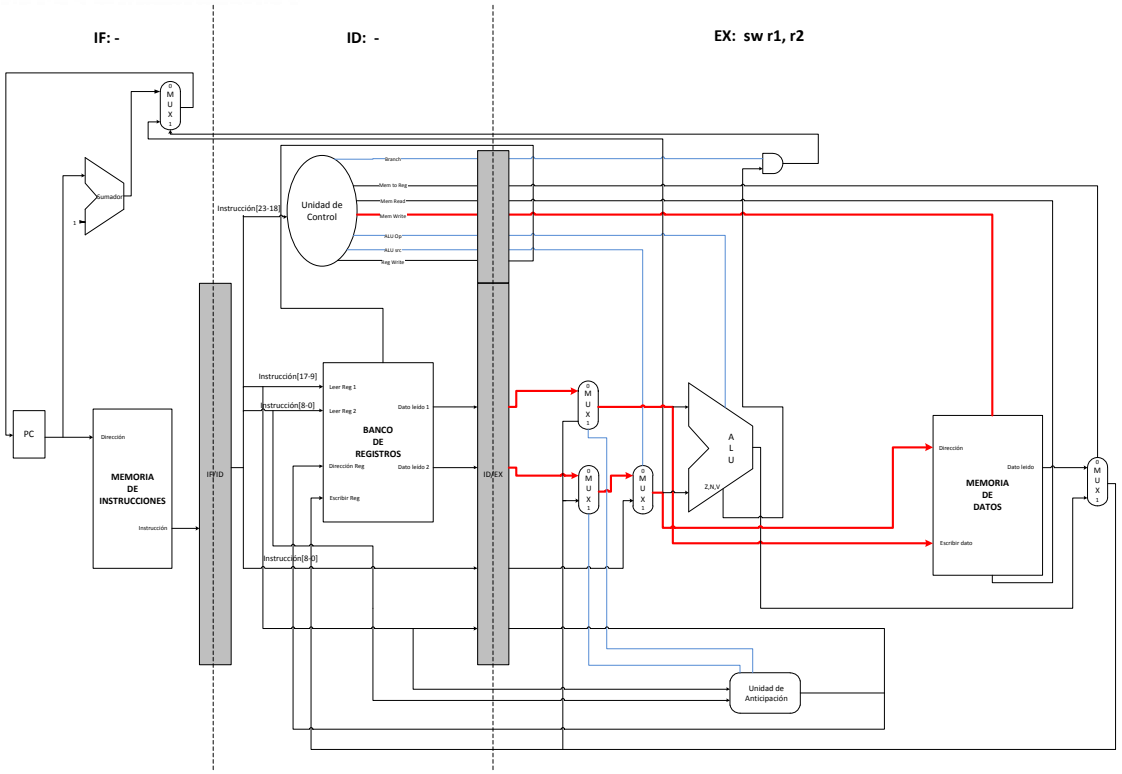


Ilustración 23. Etapa EX de Instrucción de almacenamiento

Como podemos observar en la Ilustración 23. Etapa EX de Instrucción de almacenamiento, esta etapa se posiciona en la dirección de la memoria de datos indicada por el valor procedente del registro origen almacenado en el registro de segmentación ID/EX y almacena el valor del registro que se pretende almacenar en la memoria de datos o bien el valor numérico contenido en el registro de segmentación DEC/EX.

5.2.2.3 Instrucciones aritméticas y lógicas

Las instrucciones aritméticas y lógicas realizan una operación ALU con uno o dos valores fuente para obtener un resultado destino. Para un correcto funcionamiento, la instrucción debe recorrer las distintas etapas del pipeline a través del siguiente recorrido, muy similar al descrito anteriormente:

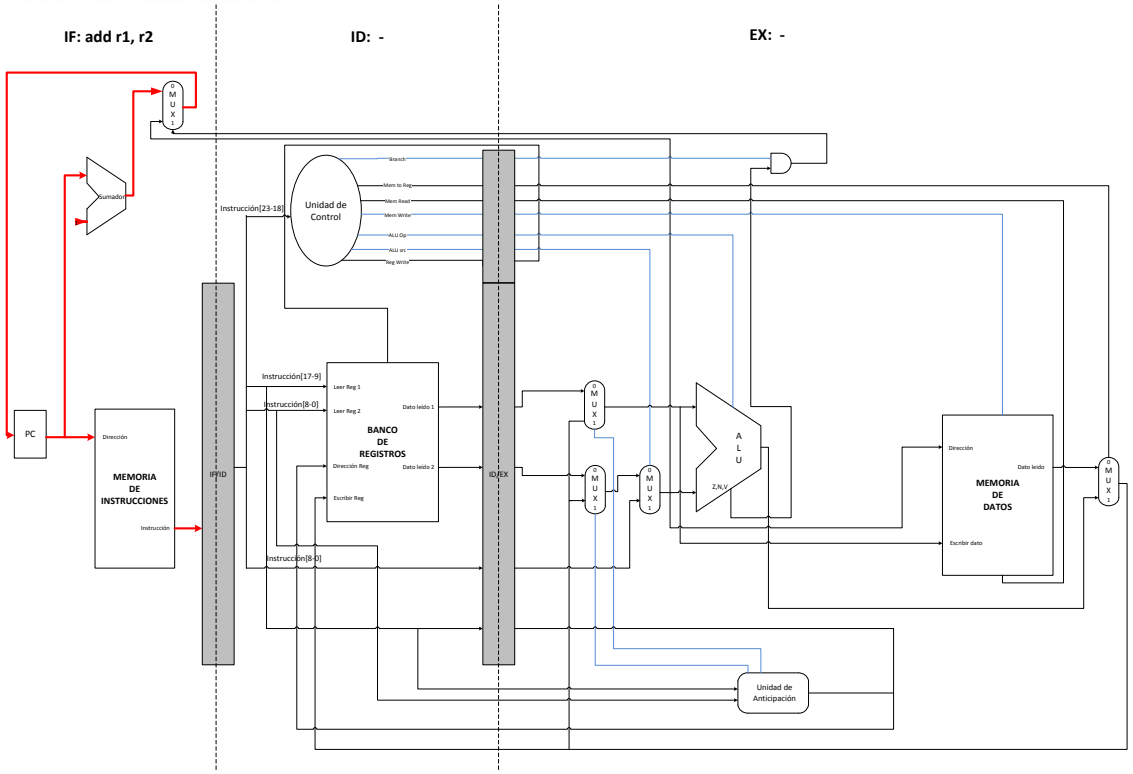


Ilustración 24. Etapa IF de Instrucción Aritmética – lógica

1. Búsqueda de la instrucción

Como observamos en la Ilustración 24. Etapa IF de Instrucción Aritmética – lógica, esta etapa lee la instrucción contenida en la dirección de la memoria de instrucciones que apunta el contador de programa(PC) y la coloca en el registro de segmentación IF/DEC.

Como podemos ver, en esta etapa también se incrementa la dirección del punteo de programa y se guarda su resultado, salvo que se haya ejecutado una instrucción de salto y por lo tanto tenga que tomar la dirección proporcionada por el salto en lugar de la dirección incrementada, este valor es colocado en el registro de segmentación IF/DEC.

2. Decodificación de la instrucción y lectura del banco de registros.

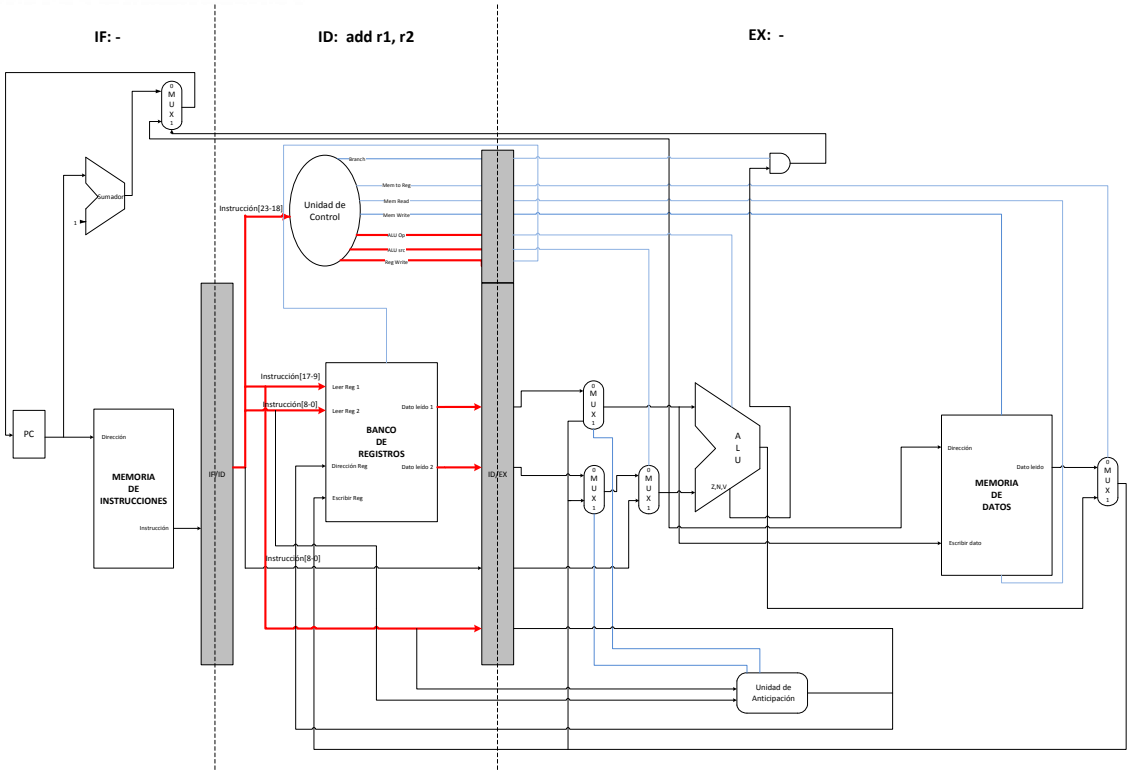


Ilustración 25. Etapa ID de Instrucción Aritmética – lógica

Como observamos en la Ilustración 25. Etapa ID de Instrucción Aritmética – lógica, esta etapa toma la instrucción del registro de segmentación IF/DEC y la decodifica tomando los bits [23-18] para interpretar el tipo de operación que la instrucción realiza, estos bits son enviados a la unidad de control y el resultado se coloca en el registro de segmentación DEC /EX; los bits [17-9] son utilizados para conocer cuál es el registro destino del banco de registros sobre el cuál se quiere almacenar el dato y tomar como primer operando, esta información es enviada al registro de segmentación DEC/EX, junto el dato contenido en ese registro; y los bits [8-0] son utilizados dependiendo de la instrucción: si se trata de una instrucción registro, son utilizados para conocer cuál es el registro del banco de registros utilizado como segundo operando y se envía junto al número de registro su contenido al registro de segmentación DEC/EX. En caso de ser una instrucción con número inmediato, su valor será utilizado como segundo operando y es enviado al registro de segmentación DEC/EX.

Como podemos ver, en esta etapa se coloca en el registro de segmentación DEC/EX la dirección del PC que el registro de segmentación IF/DEC contiene.

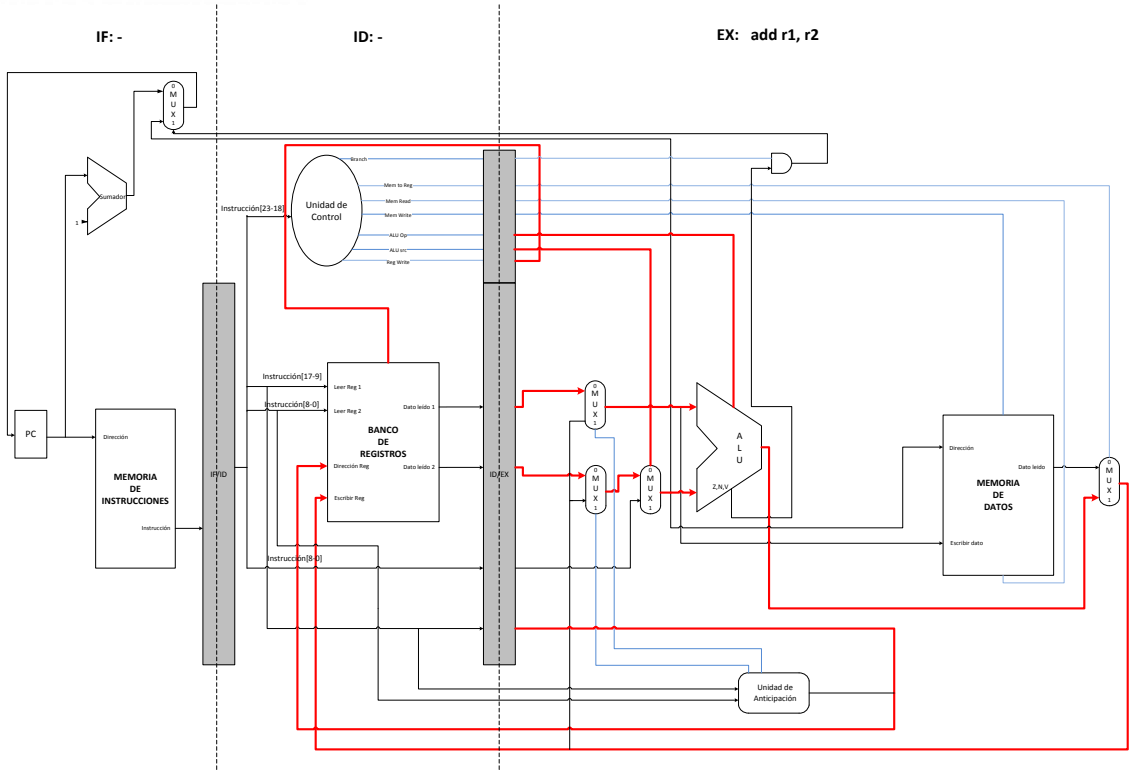


Ilustración 26. Etapa EX de Instrucción Aritmética – lógica

3. Operaciones ALU y escritura en registro de banco de registros.

Como observamos en la Ilustración 26. Etapa EX de Instrucción Aritmética – lógica, esta etapa toma del registro de segmentación DEC/EX el valor del registro destino y o bien, el valor del registro origen o bien el valor numérico inmediato en función de la señal de la unidad de control almacenada en el registro de segmentación DEC/EX. Ambos valores son enviados a las entradas de la ALU, que en función de la señal que reciba del registro de segmentación DEC/EX procedente anteriormente de la etapa anterior de la unidad de control realizará una operación u otra con ellos. Tras la operación en la ALU, los registros de estado que contiene son actualizados y a la salida envía el resultado de la operación. Tras esto, el resultado es enviado al banco de registros a la entrada “Escribir reg” y el número de registro destino contenido en el registro de segmentación DEC/EX a la entrada “Dirección Reg” para especificar sobre que registro almacenar el resultado. Esta escritura es permitida sobre el banco de registros gracias a la señal procedente del registro de segmentación DEC/EX que le envía la señal antes recibida de la unidad de control en la anterior etapa permitiendo en la etapa de ejecución la escritura sobre el banco de registros.

5.2.2.4 Instrucciones de salto condicional

Los saltos condicionales son instrucciones en las cuales primero realiza una comparación, en nuestro caso, con los registros de estado y en función del resultado se produce el salto o no a una determinada dirección de memoria de instrucciones. Para un correcto funcionamiento, la instrucción debe recorrer las distintas etapas del pipeline/procesador a través del siguiente recorrido, muy similar al descrito anteriormente:

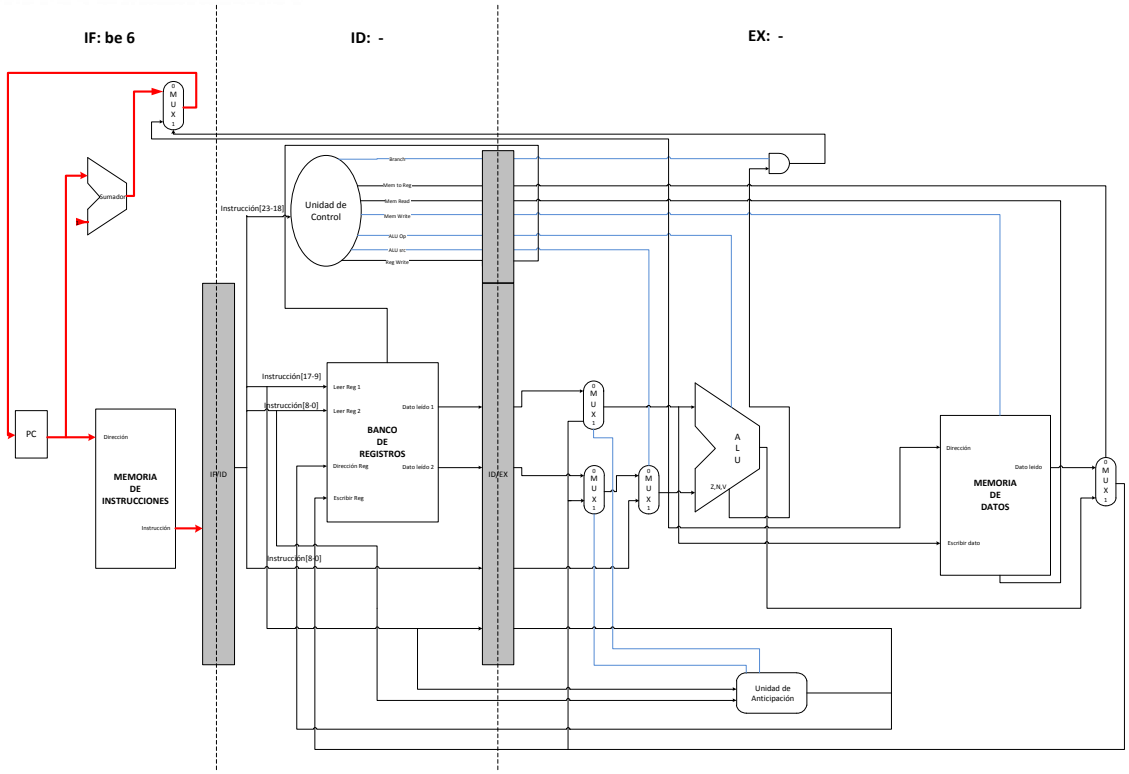


Ilustración 27. Etapa IF de Instrucción de Salto Condicional

1. Búsqueda de la instrucción

Como observamos en la Ilustración 27. Etapa IF de Instrucción de Salto Condicional, esta etapa lee la instrucción contenida en la dirección de la memoria de instrucciones que apunta el contador de programa y la coloca en el registro de segmentación IF/DEC.

Como podemos ver, en esta etapa también se incrementa la dirección del puntero de programa y se guarda su resultado en él, salvo que se haya ejecutado una instrucción de salto y por lo tanto tenga que tomar la dirección proporcionada por el salto en lugar de la dirección incrementada, este valor es colocado en el registro de segmentación IF/DEC.

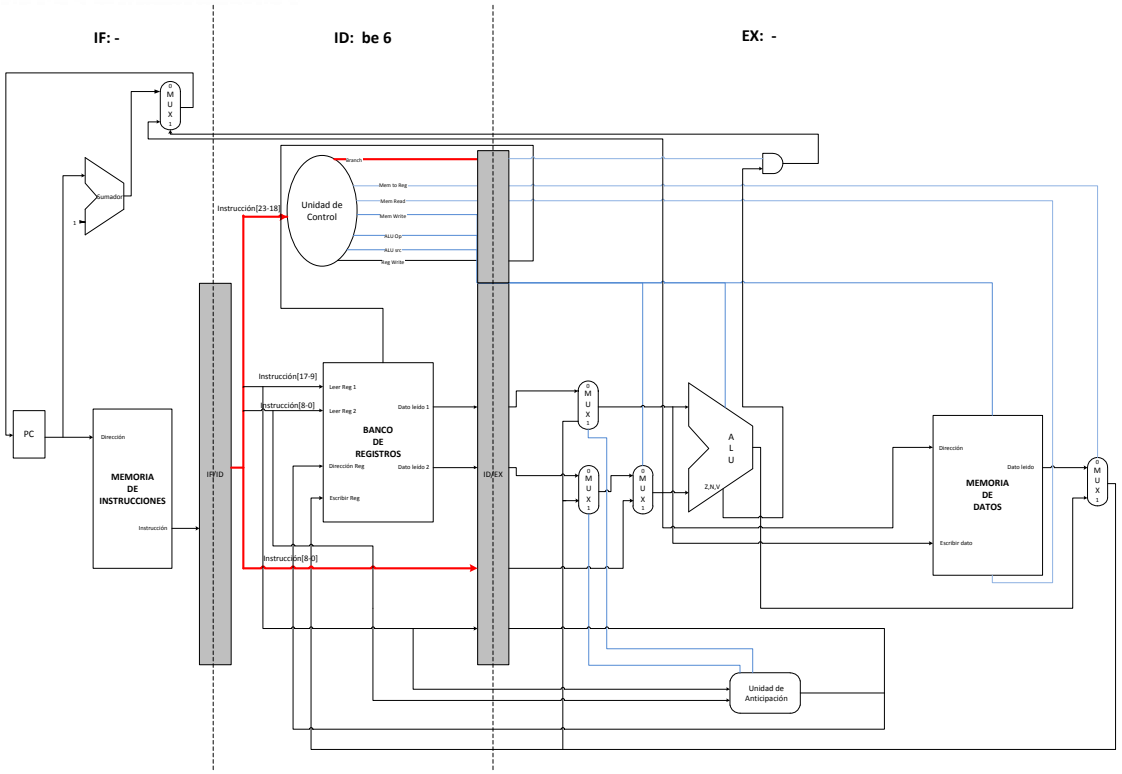


Ilustración 28. Etapa ID de Instrucción de Salto Condicional

2. Decodificación de la instrucción.

Como observamos en la Ilustración 28. Etapa ID de Instrucción de Salto Condicional, esta etapa toma la instrucción del registro de segmentación IF/DEC y la decodifica tomando los bits [23-18] para interpretar el tipo de operación que la instrucción realiza, estos bits son enviados a la unidad de control y el resultado se coloca en el registro de segmentación DEC/EX; los bits [11-0] son el número inmediato, su valor será utilizado como dirección de memoria a apuntar en caso de que el salto se produzca y es enviado al registro de segmentación DEC/EX.

Como podemos ver, en esta etapa se coloca en el registro de segmentación DEC/EX la dirección del PC que el registro de segmentación IF/DEC contiene.

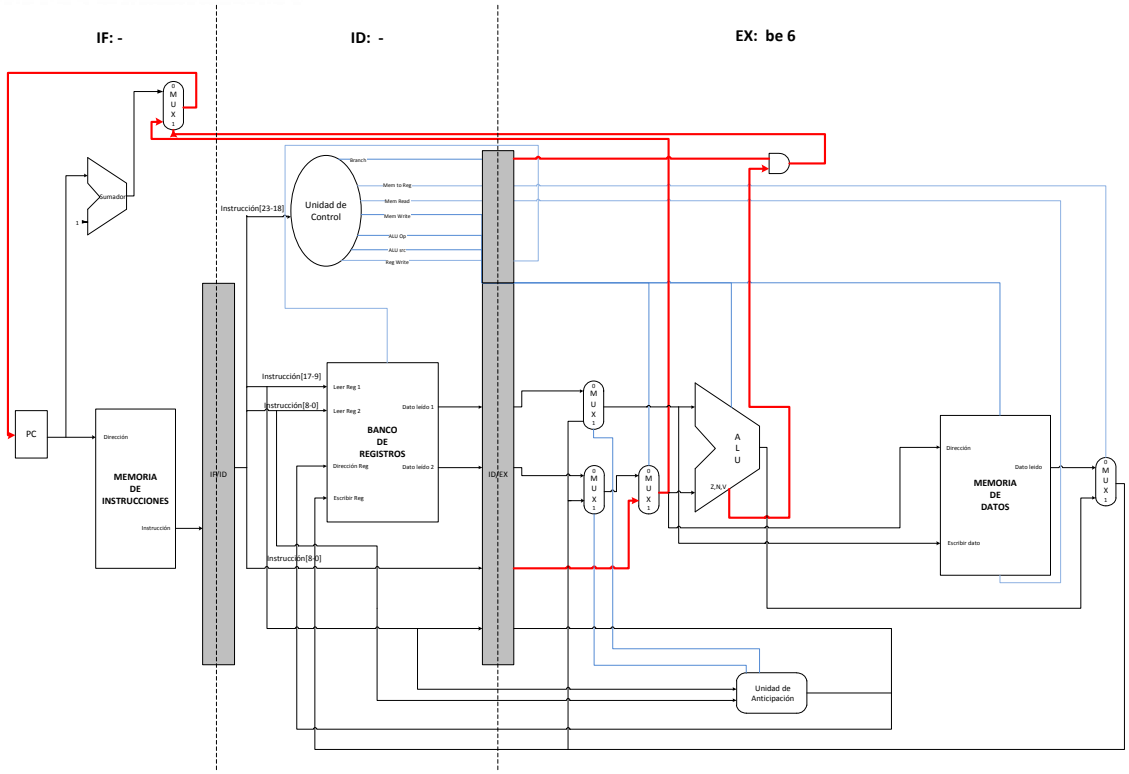


Ilustración 29. Etapa EX de Instrucción de Salto Condicional

3. Comparación y elección de salto.

Como observamos en la Ilustración 29. Etapa EX de Instrucción de Salto Condicional, durante esta etapa tomamos el valor de los registros de estado y se envían a la entrada de una puerta lógica junto a la señal que indica el tipo de salto procedente del registro de segmentación DEC/EX que en la anterior etapa provenía de la unidad de control, en función del tipo de salto y de los valores de los registros de estado enviará a la salida una señal de si se debe tomar o no el salto, esta señal de salida es enviada al multiplexor que en la etapa de Fetch recibe como entrada la siguiente instrucción a ejecutar.

El valor inmediato, procedente del registro de segmentación DEC/EX, se envía como segunda entrada del multiplexor anteriormente mencionado y en función de la señal recibida de la puerta lógica, la salida es la dirección de la siguiente instrucción a ejecutar o la dirección que coincide con el valor inmediato.

5.2.2.5 Instrucciones seguidas con unidad de anticipación

En este apartado se muestra el camino que los datos recorren cuando se trata de instrucciones que tienen acceso a un mismo registro cuando una se encuentra en la etapa de decodificación y la otra en la etapa de ejecución en el mismo ciclo. Se debe tener en cuenta que en la explicación de las anteriores instrucciones se ha despreciado el uso de la unidad de anticipación al tratarse de simulaciones de una única instrucción.

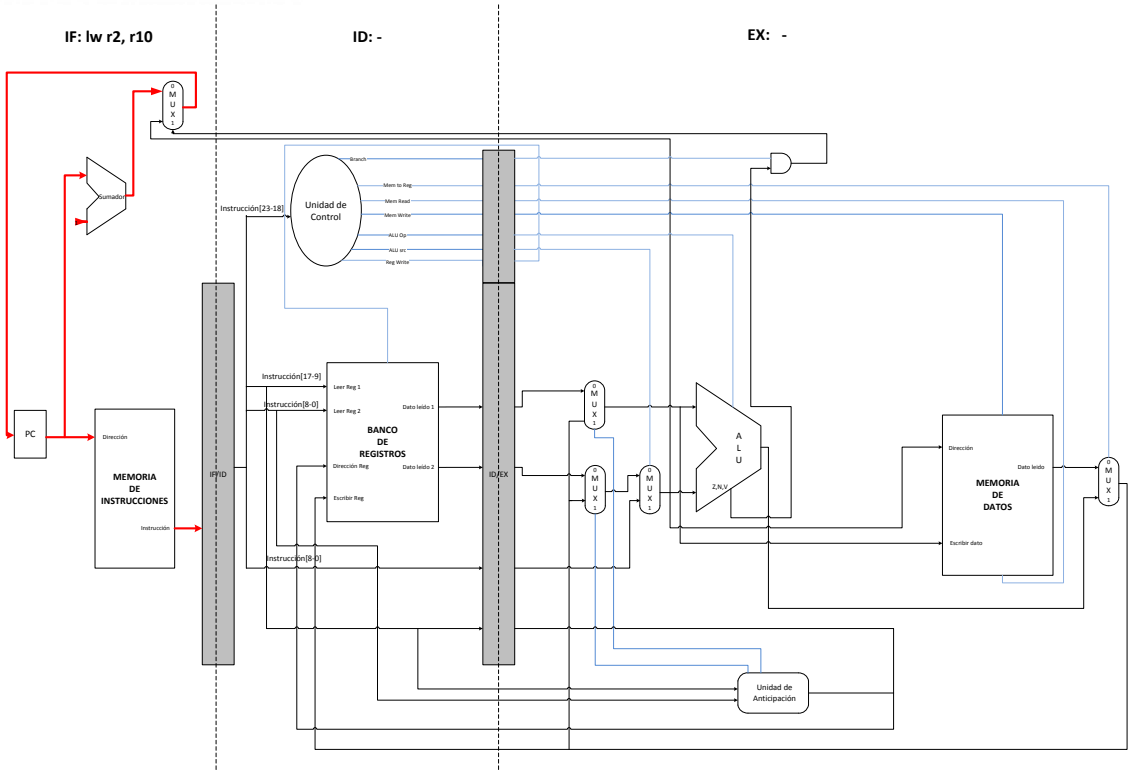


Ilustración 30. Ciclo 1 de Instrucciones que utilizan Unidad de Anticipación.

Ciclo 1

El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.1 – 1. Búsqueda de Instrucción.

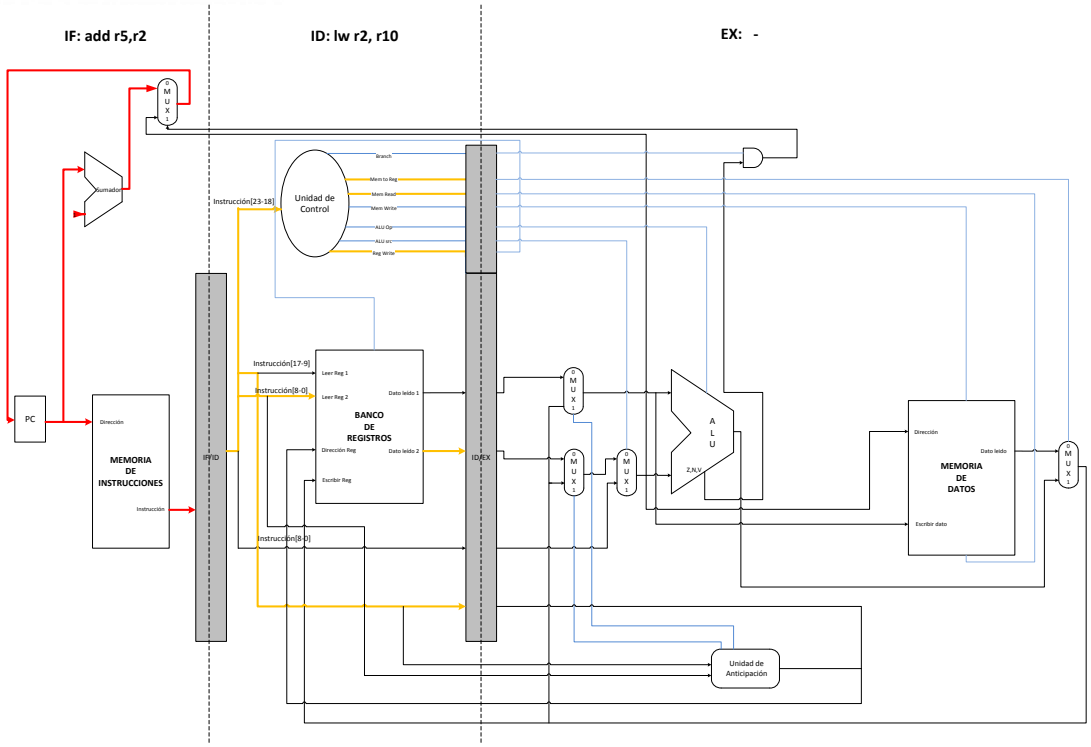


Ilustración 31. Ciclo 2 de Instrucciones que utilizan Unidad de Anticipación.

Ciclo2

- Decode
  El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.1 - 2. Decodificación de la instrucción y lectura del banco de registros.

- Fetch
  El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.3 - 1. Búsqueda de Instrucción.

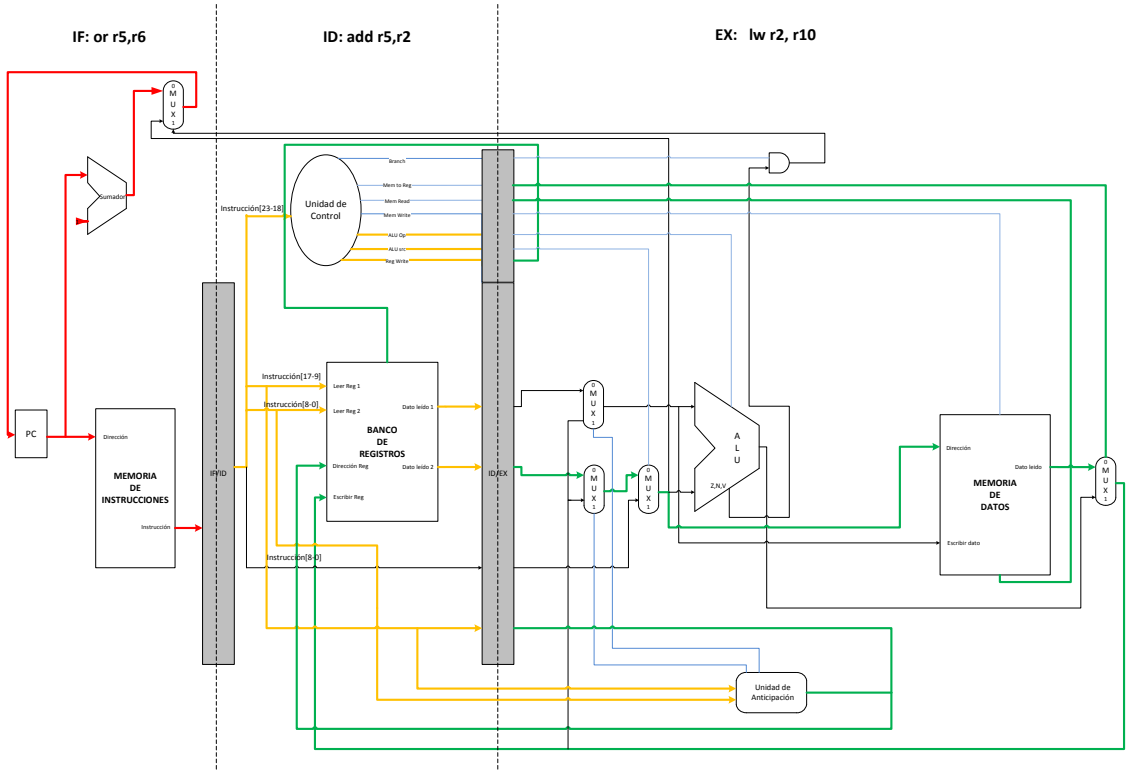


Ilustración 32. Ciclo 3 de Instrucciones que utilizan Unidad de Anticipación.

Ciclo 3

- Fetch
  El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.3 - 1. Búsqueda de Instrucción.
- Decode
  El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.3 - 2. Decodificación de la instrucción y lectura del banco de registros.
- Exec
  El funcionamiento es el descrito en el apartado 5.2.2.1 - 3. Acceso a memoria de datos y escritura en registro de banco de registros, sin embargo, hay una importante diferencia y es que en la etapa de decodificación tenemos una instrucción. Esta instrucción envía como entrada a la unidad de anticipación los registros que se dispone a utilizar en la etapa de ejecución en el siguiente ciclo, estos son comparados con el

registro destino de la instrucción que ahora se encuentra en ejecución. En caso de coincidir entre ellos se envía una señal al multiplexor de la etapa de ejecución para que en el siguiente ciclo la instrucción que se encuentre en la etapa de ejecución tome el valor de la unidad de anticipación en lugar del procedente del registro de segmentación.

En este caso coincide el registro destino r2 de la instrucción “lw r2, r10” en la etapa de ejecución con el registro origen r2 de la instrucción “add r5, r2” en la etapa de decodificación, por lo tanto, se envía la señal al multiplexor en el siguiente ciclo.

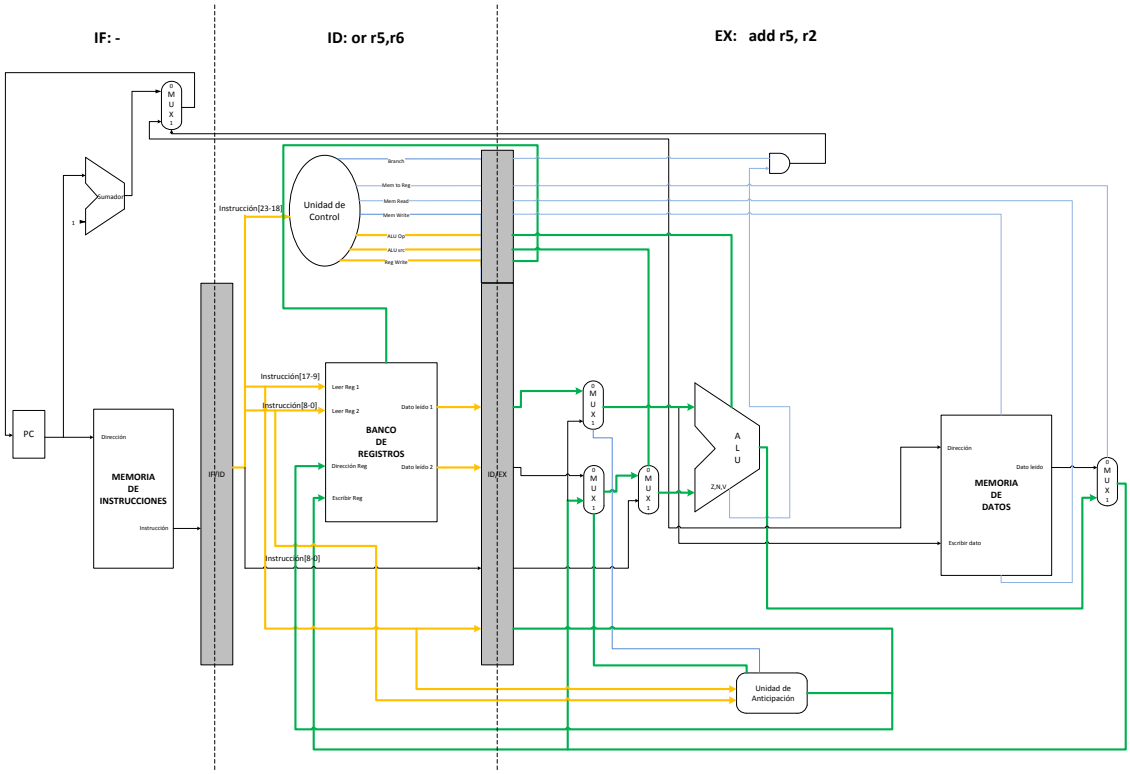


Ilustración 33. Ciclo 4 de Instrucciones que utilizan Unidad de Anticipación.

Ciclo 4

- Fetch

  En este ciclo ya no se encuentra ninguna instrucción en esta etapa.
- Decode

  El funcionamiento es exactamente como se ha descrito en el apartado 5.2.2.3 - 2. Decodificación de la instrucción y lectura del banco de registros.
- Exec

  El funcionamiento es el descrito en el apartado 5.2.2.3 - 3. Operaciones ALU y escritura en registro de banco de registros, sin embargo, en esta ocasión:

  · Toma el valor de la unidad de anticipación para r2(RS) al recibir la señal el multiplexor indicando que debe tomar el valor almacenado en la unidad de anticipación en lugar del procedente del registro de segmentación DEC/EX.

· También ocurre lo mismo que en el anterior ciclo, uno de los números de registro de la instrucción “or r5, r6” de la etapa de decodificación coincide con el registro destino de la instrucción “add r5, r2” de la etapa de ejecución, por lo tanto, se enviará una señal en el próximo ciclo para que se tome el valor de la unidad de anticipación en lugar del valor procedente del registro de segmentación.

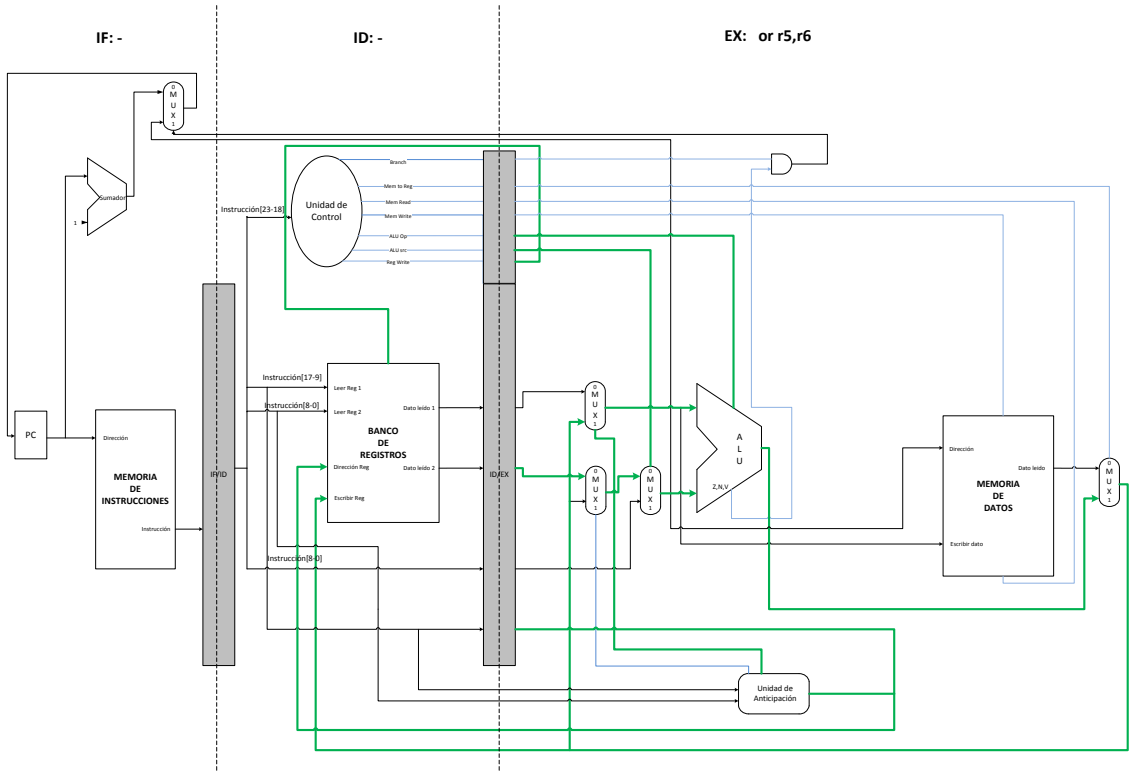


Ilustración 34. Ciclo 5 de Instrucciones que utilizan Unidad de Anticipación.

Ciclo 5

- Fetch
  En este ciclo ya no se encuentra ninguna instrucción en esta etapa.
- Decode
  En este ciclo ya no se encuentra ninguna instrucción en esta etapa.
- Exec
- El funcionamiento es el descrito en el apartado 5.2.2.3 - 3. Operaciones ALU y escritura en registro de banco de registros, sin embargo, en esta ocasión:
  · Toma el valor de la unidad de anticipación para r5(RS) al recibir la señal el multiplexor indicando que debe tomar el valor almacenado en la unidad de anticipación en lugar del procedente del registro de segmentación DEC/EX.

5.3 Lenguajes de programación

En esta sección se describen los lenguajes de programación utilizados para la elaboración del simulador. El principal motivo para elegir entre un lenguaje de programación u otro radica en las necesidades del sistema, y por lo tanto se eligen aquellos que se adecuen más al

cumplimiento de los requisitos del simulador a desarrollar. También se ha tenido en cuenta el conocimiento de los mismos o su dificultad de aprendizaje.

5.3.1 Lenguaje C

C es un lenguaje de programación originalmente desarrollado por Dennis M. Ritchie entre 1969 y 1972 en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Las características principales de este lenguaje son:

· Es un lenguaje de programación de nivel medio ya que combina los elementos del lenguaje de alto nivel con la funcionalidad del ensamblador.

· Su característica principal es ser portable, es decir, es posible adaptar los programas escritos para un tipo de computadora en otra.

· Otra de sus características principales es el ser estructurado, es decir, el programa se divide en funciones independientes entre sí.

· El lenguaje C inicialmente fue creado para la programación de sistemas operativos, intérpretes, editores, ensambladores y compiladores.

· Manipula bits, bytes y direcciones.

Como se ha comentado anteriormente, se utiliza este lenguaje para cumplir con uno de los requisitos especificados por la empresa Tabla 63. Requisito de restricción 1 - Simulador en lenguajes C y tcl.

5.3.2 Lenguaje tcl

Tcl es un tipo de lenguaje de programación perteneciente a la categoría de lenguajes script. Fue creado por John Ousterhout con la finalidad de crear un lenguaje con una sintaxis sencilla pero sin decaer en su funcionalidad. Es un lenguaje de comandos, cuyo intérprete recibe el nombre de tclsh, que tiene como una de sus principales características la gran facilidad con la que se pueden implementar funciones en C/C++ que pasan a ser nuevas instrucciones del intérprete. Las principales características de este lenguaje son:

· Es un lenguaje script.

· Es fácil de aprender.

· Trabaja en diferentes plataformas.

· Puede embeberse dentro de programas en C o C++.

· Se puede extender.

· Es libre.

Como se ha comentado anteriormente, se utiliza este lenguaje para cumplir con uno de los requisitos especificados por la empresa Tabla 63. Requisito de restricción 1 - Simulador en lenguajes C y tcl.

5.4 Implementación del simulador nanoprocesador

En este apartado se realiza un análisis de las distintas clases que componen el simulador junto a varios ejemplos de ejecución una vez implementado.

5.4.1 Análisis de clases

En este apartado se incluye una especificación de las clases que serán utilizadas por el sistema para su correcto funcionamiento.

A continuación se muestra un diagrama de clases que representa el sistema al completo, y posteriormente una breve descripción de cada clase junto a los métodos y atributos que aparecen en cada una de ellas.

5.4.1.1 Diagrama de clases

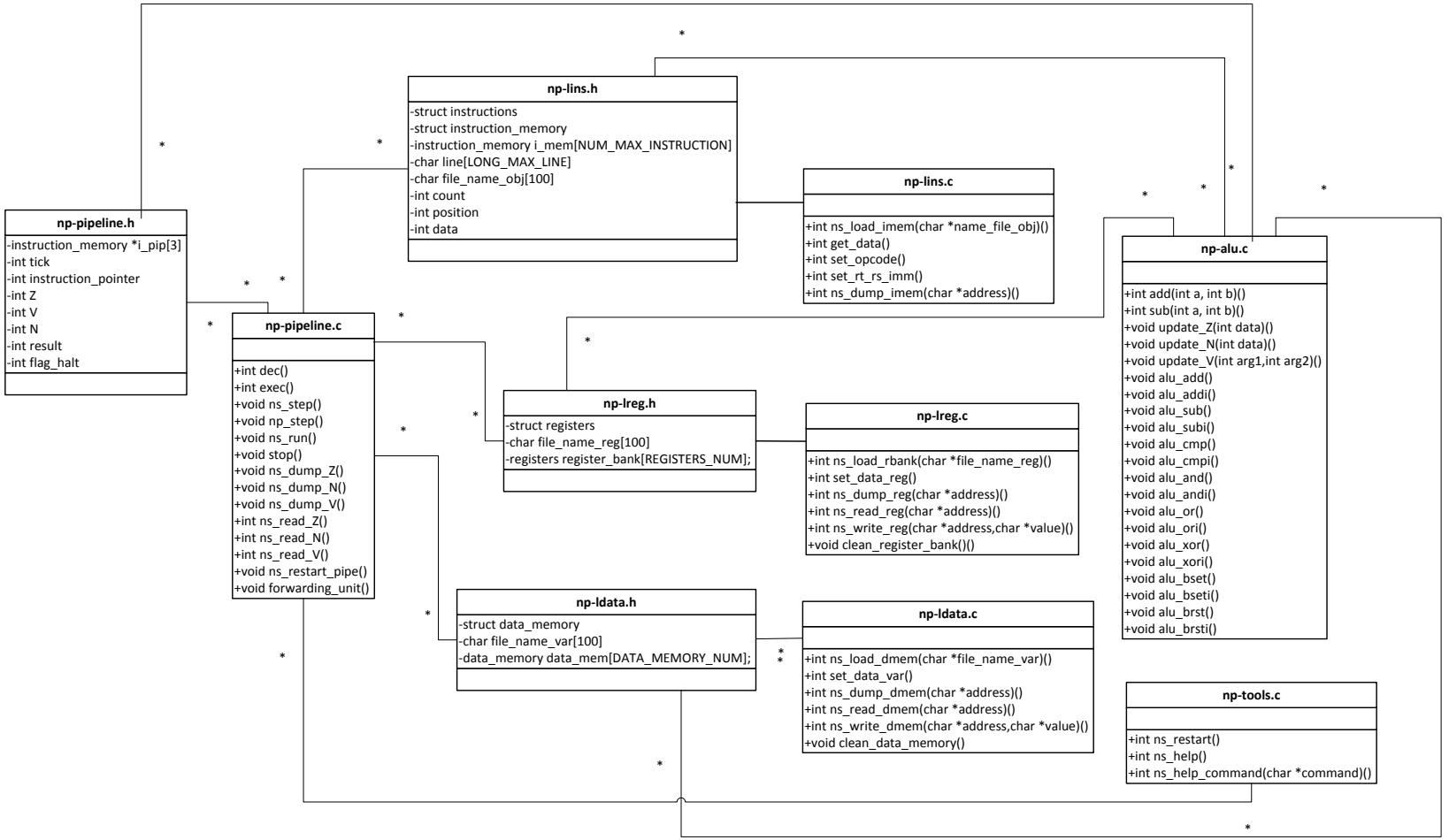


Ilustración 35. Diagrama de clases

5.4.1.2 Identificación de las clases

A continuación se exponen mediante una serie de tablas las clases del proyecto relativas al funcionamiento del simulador del nanoprocesador.

Nombre	NP-LINS.H
Descripción	Crea la estructura de las instrucciones. Crea la memoria de instrucciones.
Atributos	**LONG_MAX_LINE:** Numero de bytes máximo que el programa lee por línea de un fichero. **NUM_MAX_INSTRUCTION:** Número máximo de instrucciones permitidas en la memoria de instrucciones. **SIZE_POSITION_HEX**: Número de bits para describir la posición en hexadecimal del fichero .obj.. **SIZE_DATA_HEX_INS**: Número de bits para describir la instrucción en hexadecimal del fichero .obj. **NUM_SPACES_IN_LINE:** Número de espacios que hay entre la posición y la instrucción en una línea del fichero .obj. **num_bits:** Cantidad de bits que ocupa una instrucción representada en binario. **num_bits_nopecode**: Cantidad de bits que ocupa una instrucción sin los bits de operation code representada en binario. **num_bits_reg**: Cantidad de bits que ocupa el número de registro. **num_bit_noimm12**: Cantidad de bits que ocupa el valor inmediato imm12. **mask_RS**: Mascara de bits para extraer el número de registro RS de la instrucción. **mask_RT**: Mascara de bits para extraer el número de registro RT de la instrucción. **mask_RT6:** Mascara de bits para extraer el número de registro RT que ocupa 6 bits de la instrucción. **mask_IMM12:** Mascara de bits para extraer el valor inmediato imm12. **mask_IMM9:** Mascara de bits para extraer el valor inmediato imm9. **mask_IMM8:** Mascara de bits para extraer el valor inmediato imm8. **mask_IMM4:** Mascara de bits para extraer el valor inmediato imm4. **opcode_NOP:** Identificador de la operación nop. **opcode_BR:** Identificador de la operación br. **opcode_BI:** Identificador de la operación bi. **opcode_BE:** Identificador de la operación be. **opcode_BNE:** Identificador de la operación bne. **opcode_BGT:** Identificador de la operación bgt. **opcode_BGE:** Identificador de la operación bge. **opcode_LW:** Identificador de la operación lw. **opcode_LWI:** Identificador de la operación lwi. **opcode_SW:** Identificador de la operación sw. **opcode_SWI:** Identificador de la operación swi. **opcode_MV:** Identificador de la operación mv. **opcode_MVI:** Identificador de la operación mvi. **opcode_MVA:** Identificador de la operación mva. **opcode_ADD:** Identificador de la operación add. **opcode_ADDI:** Identificador de la operación addi.

opcode_SUB: Identificador de la operación sub.
opcode_SUBI: Identificador de la operación subi.
opcode_CMP: Identificador de la operación cmp.
opcode_CMPI: Identificador de la operación cmpi.
opcode_AND: Identificador de la operación and.
opcode_ANDI: Identificador de la operación andi.
opcode_OR: Identificador de la operación or.
opcode_ORI: Identificador de la operación ori.
opcode_XOR: Identificador de la operación xor.
opcode_XORI: Identificador de la operación xori.
opcode_BTEST: Identificador de la operación btest.
opcode_BTESTI: Identificador de la operación btesti.
opcode_BSET: Identificador de la operación bset.
opcode_BSETI: Identificador de la operación bseti.
opcode_BRST: Identificador de la operación brst.
opcode_BRSTI: Identificador de la operación brsti.
opcode_HALT: Identificador de la operación halt.
instructions: Contiene la estructura de las instrucciones del programa, está formado por los siguientes atributos:

- opcode: Identificador de la operación.
- union: Contiene en el mismo espacio de memoria los campos de o bien un registro, de dos registros o un registro y un valor inmediato, tal como en el anexo B:
  - rs: variable de tipo register que contendrá el registro de origen.

  o
  - rt: variable de tipo register que contendrá el registro de destino.

  o
  - imm: variable de int que contendrá el valor inmediato.

  o
  - rt e imm.
  - rt y rs.

instruction_memory: Contiene la estructura de la memoria de instrucciones del programa, está formada por los siguientes atributos:

- position: Dirección de la instrucción.
- hex[SIZE_POSITION_HEX]: array char que contiene la instrucción en hexadecimal.
- Ins: variable de tipo instructions que contendrá la instrucción.
- Disassembled[124]: array char que contiene la instrucción desensamblada.

imem[NUM_MAX_INSTRUCTION]: Array de tipo instruction_memory que simula la memoria de instrucciones.
line[LONG_MAX_LINE]: Array de char que contiene la línea leída del fichero .obj.
file_name_obj[100]: Array char que contiene el nombre del fichero .obj.
count: Cuenta el número de instrucciones que contiene el programa.

Tabla 73. Header np-lins.h

Nombre	NP-LINS.C

Descripción	Lee el fichero hexadecimal .obj que contiene las instrucciones del programa y las almacena en la memoria de instrucciones. Permite mostrar por pantalla el contenido de una dirección de la memoria de instrucciones.
Métodos	**int ns_load_imem(char *name_file_obj):** Lee el fichero .obj línea a línea, extrae la dirección en hexadecimal de memoria sobre la que se desea escribir la instrucción, la convierte en decimal y llama a los métodos get_data, set_rt_rs y set_opcode. En caso de que el método se haya ejecutado con éxito devuelve 0. **int get_data():**Lee la línea obtenida de ns_load_imem del fichero .obj, extrae la instrucción, la convierte a decimal y la almacena en la memoria de instrucciones en la dirección ya extraída en el método mencionado anteriormente. En caso de que el método se haya ejecutado con éxito devuelve 0. **int set_opcode():**Lee la instrucción extraída del método get_data y toma los 6 bits más significativos y los guarda en el campo opcode correspondiente a la instrucción leída en la memoria de instrucciones. **int set_rt_rs_imm():**Lee la instrucción extraída del método get_data y toma los bits correspondientes a rt, rs e imm en función de la instrucción como podemos ver en el anexo B y los almacena en los campos rt, rs o imm de la instrucción correspondiente en la memoria de instrucciones. **int ns_dump_imem(char *address):** Este método recibe como parámetro una dirección, distinguiendo si es en decimal o hexadecimal e imprime por pantalla la instrucción junto todos sus campos anteriormente mencionado que contiene la memoria de instrucciones en esa dirección. En caso de que el método se haya ejecutado con éxito devuelve 0.

Tabla 74. Clase np-lins.c

Nombre	NP-LREG.H
Descripción	Crea la estructura de los registros. Crea el banco de registros.
Atributos	**SIZE_DATA_HEX_REG_VAR**: Número de bits para describir el número de registro del banco de registros en hexadecimal del fichero .reg o bien, el número de bits para describir dirección de memoria de la memoria de datos en hexadecimal del fichero .var **REGISTERS_NUM**: Número máximo de registros permitidos en el banco de registros. **Registers**: Contiene la estructura de los registros del banco de registros, está formado por los siguientes atributos: • position: Número identificador del registro. • data: Datos que contiene el registro. **register_bank[REGISTERS_NUM]:** Array de tipo registers que simula el banco de registros.

	file_name_reg[100]: Array char que contiene el nombre del fichero .reg

Tabla 75. Header np-lreg.h

Nombre	NP-LREG.C
Descripción	Lee el fichero hexadecimal .reg que contiene los registros utilizados por el programa con valores iniciales y escribe los datos en el banco de registros. Permite mostrar por pantalla el contenido de un registro a partir de su dirección, leer el contenido de un registro a partir de su dirección y escribir directamente en el banco de registros enviando como parámetros una dirección y un dato.
Métodos	**int ns_load_rbank(char *file_name_reg):** Lee el fichero .reg línea a línea, extrae la dirección en hexadecimal del registro sobre el que se desea escribir en el banco de registros, la convierte en decimal y llama al método set_data_reg(). En caso de que el método se haya ejecutado con éxito devuelve 0. **int set_data_reg():**Lee la línea obtenida de ns_load_rbank del fichero .reg, extrae el dato, lo convierte a decimal y lo almacena en el registro del banco de registros en la dirección ya extraída en el método mencionado anteriormente. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_dump_reg(char *address):** Este método recibe como parámetro una dirección, distinguiendo si es en decimal o hexadecimal y muestra por pantalla el dato en decimal contenido en el registro indicado con la dirección del banco de registros. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_read_reg(char *address):** Este método recibe como parámetro una dirección, distinguiendo si es en decimal o hexadecimal y devuelve el dato en decimal contenido en el registro indicado con la dirección del banco de registros. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_write_reg(char *address,char *value):** Este método recibe como parámetros una dirección y un dato, distinguiendo si están en decimal o hexadecimal y escribe en el banco de registros el dato recibido en la dirección recibida. En caso de que el método se haya ejecutado con éxito devuelve 0. **void clean_register_bank():**Este método se encarga de reiniciar el banco de registros y por lo tanto devolver a cero todos los datos del banco de registros.

Tabla 76. Clase np-lreg.c

Nombre	NP-LDATA.H
Descripción	Crea la memoria de datos
Atributos	**SIZE_DATA_HEX_REG_VAR**: Número de bits para describir el número de registro del banco de registros en hexadecimal del fichero .reg o bien, el número de bits para describir dirección de memoria de la

	memoria de datos en hexadecimal del fichero .var **DATA_MEMORY_NUM**: Número máximo de direcciones permitidas en la memoria de datos. **Data_memory**: Contiene la estructura de la memoria de datos, está formada por los siguientes atributos: • position: Dirección de memoria del dato. • data: Datos. **data_mem[DATA_MEMORY_NUM]:** Array de tipo data_memory que simula la memoria de datos. **file_name_var[100]**: Array char que contiene el nombre del fichero .var

Tabla 77. Header np-ldata.h

Nombre	NP-LDATA.C
Descripción	Lee el fichero hexadecimal .var que contiene los datos de memoria utilizados por el programa con valores iniciales y escribe los datos en la memoria de datos. Permite mostrar por pantalla el contenido de una dirección de la memoria de datos, leer el contenido de una dirección de la memoria de datos y escribir directamente en la memoria de datos enviando como parámetros una dirección y un dato.
Métodos	**int ns_load_dmem(char *file_name_var):** Lee el fichero .var línea a línea extrae la dirección en hexadecimal de memoria sobre la que se desea escribir un dato la convierte en decimal y llama al método set_data_var(). En caso de que el método se haya ejecutado con éxito devuelve 0. **int set_data_var():** Lee el dato en hexadecimal del fichero .var y lo escribe en decimal en la memoria de datos en la dirección ya extraída por el método anterior explicado. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_dump_dmem(char *address):** Este método recibe como parámetro una dirección, distinguiendo si es en decimal o hexadecimal y muestra por pantalla el dato que contiene la memoria de datos en esa dirección. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_read_dmem(char *address):** Este método recibe como parámetro una dirección, distinguiendo si es en decimal o hexadecimal y devuelve el dato que contiene la memoria de datos en esa dirección. En caso de que el método se haya ejecutado con éxito devuelve 0. **int ns_write_dmem(char *address,char *value):** Este método recibe como parámetros una dirección y un dato, distinguiendo si están en decimal o hexadecimal y escribe en la memoria de datos el dato recibido en la dirección recibida. En caso de que el método se haya ejecutado con éxito devuelve 0. **void clean_data_memory():**Este método se encarga de reiniciar la memoria de datos y por lo tanto devolver a cero todos los datos de la

	memoria de datos.

Tabla 78. Clase np-ldata.c

Nombre	NP-PIPELINE.H
Descripción	Crea registros de segmentación.
Atributos	**instruction_memory *i_pip[3]:** Contiene los registros de segmentación que contendrá las instrucciones del programa correspondientes a cada etapa, es de tipo instruction_memory. **instruction_memory *i_pip_temp[3]:** Contiene los registros de segmentación temporales que contendrá las instrucciones del programa correspondientes a cada etapa antes de avanzar el ciclo de reloj, es de tipo instruction_memory. **instruction_pointer:** Contiene la dirección de la última instrucción tomada de la memoria de instrucciones. **Z:** Contiene el valor del registro de estado Zero de la ALU. **N:** Contiene el valor del registro de estado Negative de la ALU. **V:** Contiene el valor del registro de estado Overflow de la ALU. **flag_halt:** Bit que nos indica si la instrucción en ejecución es halt, en caso de ser así su valor será 1, en caso contrario será 0. **flag_jump:** Bit que nos indica si se va a producir un salto, en caso de ser así su valor será 1, en caso contrario será 0. **exec_position_rt_rbank:** Contiene el número de registro que es modificado en la etapa exec. **dec_position_rt_rbank:** Contiene el número de registro destino de la instrucción que se encuentra en la etapa decode. **dec_position_rs_rbank:** Contiene el número de registro origen de la instrucción que se encuentra en la etapa decode.

Tabla 79. Header np-pipeline.h

Nombre	NP-PIPELINE.C
Descripción	Simula el recorrido de los datos del nanoprocesador segmentado. Permite ejecutar la simulación ciclo a ciclo o bien de forma continua, mostrando por pantalla el pipeline por ciclo si el usuario lo desea.
Métodos	**int dec():** Simula la etapa de decodificación tomando los valores de los registros del banco de registros formulados en la instrucción y enviando a la etapa exec junto el código de operación y el valor inmediato si lo hubiese. **int exec():** Simula la etapa de ejecución, dependiendo del código de la instrucción realiza una operación de salto, de registro a registro, de memoria a registro, de registro a memoria o llama a la ALU para realizar una operación aritmético-lógica. **void ns_step():** Ejecuta la simulación un ciclo de reloj, mostrando por pantalla el pipeline. **void np_step():**Ejecuta la simulación un ciclo de reloj, sin mostrar por pantalla el pipeline. **void ns_run():**Ejecuta la simulación de forma continua mostrando por pantalla el pipeline. **void stop():** Para la simulación continua en caso de que reciba una señal de ctrl+C. **void ns_dump_Z():** Muestra por pantalla el valor del bit del registro de estado Zero de la ALU.

	int ns_read_Z(): Retorna el valor del bit del registro de estado Zero de la ALU. **void ns_dump_V():**Muestra por pantalla el valor del bit del registro de estado Overflow de la ALU. **int ns_read_V():** Retorna el valor del bit del registro de estado Overflow de la ALU. **void ns_dump_N():**Muestra por pantalla el valor del bit del registro de estado Negative de la ALU. **int ns_read_N():**Retorna el valor del bit del registro de estado Negative de la ALU. **void ns_restart_pipe():** Reinicia la simulación, devolviendo a 0 todos los datos de las estructuras del nanoprocesador: banco de registros, memoria de datos, memoria de instrucciones. **void forwarding_unit():** Simula el comportamiento de la Unidad de Anticipación comparando los numero de registros utilizados en la etapa decodificación con el número de registro que será modificado en la etapa de ejecución, en caso de coincidir el registro que coincide en la etapa de decodificación tomará el valor más nuevo.

Tabla 80. Clase np-pipeline.c

Nombre	NP-ALU.C
Descripción	Simula el comportamiento de la Unidad Aritmético Lógica.
Métodos	**int add(int a, int b):** Método encargado de la suma binaria entre dos número. **int sub(int a, int b):** Método encargado de la resta binaria entre dos número. **void update_Z(int data):** Actualiza el valor de Zero de la ALU. **void update_N(int data):** Actualiza el valor de Negative de la ALU. **void update_V(int arg1,int arg2):** Actualiza el valor de Overflow de la ALU. **void alu_add():** Método que realiza la instrucción add, suma bit a bit de dos valores procedentes del banco de registros y el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V. **void alu_addi():**Método que realiza la instrucción addi, suma bit a bit de dos valores procedentes uno del banco de registros y el otro el valor inmediato de la instrucción, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V. **void alu_sub():**Método que realiza la instrucción sub, suma bit a bit invirtiendo el segundo valor, los operandos son valores procedentes del banco de registros y el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V. **void alu_subi():**Método que realiza la instrucción sub, suma bit a bit invirtiendo el segundo valor, los operandos son un valor procedente del banco de registros y el otro el valor inmediato de la instrucción, el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado

Z, N y V.
void alu_cmp():Método que realiza la instrucción sub, suma bit a bit invirtiendo el segundo valor, los operandos son valores procedentes del banco de registros y el resultado a diferencia de los anteriores no lo almacena en el banco de. También actualiza los registros de estado Z, N y V.
void alu_cmpi():Método que realiza la instrucción sub, suma bit a bit invirtiendo el segundo valor, los operandos son un valor procedente del banco de registros y el otro el valor inmediato de la instrucción, el resultado a diferencia de los anteriores no lo almacena en el banco de registros. También actualiza los registros de estado Z, N y V.
void alu_and():Método que realiza la instrucción and, operación lógica and bit a bit de dos valores procedentes del banco de registros y el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_andi():Método que realiza la instrucción and, operación lógica and bit a bit de dos valores procedentes uno del banco de registros y el otro el valor inmediato de la instrucción, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_or():Método que realiza la instrucción or, operación lógica or bit a bit de dos valores procedentes del banco de registros y el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_ori():Método que realiza la instrucción ori, operación lógica ori bit a bit de dos valores procedentes uno del banco de registros y el otro el valor inmediato de la instrucción, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_xor():Método que realiza la instrucción xor, operación lógica xor bit a bit de dos valores procedentes del banco de registros y el resultado lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_xori():Método que realiza la instrucción xori, operación lógica xori bit a bit de dos valores procedentes uno del banco de registros y el otro el valor inmediato de la instrucción, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. También actualiza los registros de estado Z, N y V.
void alu_bset():Método que realiza la instrucción bset, operación encargada de sustituir en el registro destino el valor del bit de la posición igual al valor que contiene el registro de origen por 0, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica.
void alu_bseti():Método que realiza la instrucción bseti, operación encargada de sustituir en el registro destino el valor del bit de la

	posición igual al valor inmediato de la instrucción por 0, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. **void alu_brst():**Método que realiza la instrucción bset, operación encargada de sustituir en el registro destino el valor del bit de la posición igual al valor que contiene el registro de origen por 1, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica. **void alu_brsti():**Método que realiza la instrucción bseti, operación encargada de sustituir en el registro destino el valor del bit de la posición igual al valor inmediato de la instrucción por 1, el resultado de la operación lo almacena en el banco de registros en el registro destino que la instrucción especifica.

Tabla 81. Clase np-alu.c

Nombre	NP-TOOLS.C
Descripción	Contiene las instrucciones de ayuda del simulador y reinicio del mismo.
Métodos	**int ns_restart():** Método encargado de reiniciar las estructuras principales del nanoprocesador (memoria de instrucciones, banco de registros y memoria de datos). **int ns_help():** Método encargado de imprimir por pantalla la lista de comandos que el simulador posee. **int ns_help_command(char *command):** Se encarga de imprimir por pantalla la información del comando introducido como argumento.

Tabla 82. Clase np-tools.c

Cabe destacar que estas clases serán convertidas a código tcl mediante la herramienta SWIG [17] para cumplir con el requisito especificado por la empresa de ser un simulador en lenguaje tcl y poder ejecutarse sobre la herramienta tclsh Tabla 63. Requisito de restricción 1 - Simulador en lenguajes C y tcl y Tabla 33. Requisito de capacidad 1 - Compatibilidad con tclsh y ModelSim.

5.4.2 Ejemplos de simulaciones

En este apartado se muestran una serie de ejemplos de programas en ensamblador ejecutados sobre nuestro simulador.

5.4.2.1 Programa con instrucciones aritméticas y lógicas "alu_test.asm"

El programa ensamblador es el siguiente:

```
        .header
r1:     .reg %r1 100
r2:     .reg %r2 5

        .text
main:
        add r1, r2
        addi r2, 50
        and r1,r2
        halt
        .end
```

Este programa realiza las siguientes acciones:
· Almacena inicialmente en el banco de registros los siguientes datos:
- En el registro r1 almacena el dato 100.
- En el registro r2 almacena el dato 5.

· Ejecuta las operaciones:
- Suma entre registros r1 y r2.
- Suma entre el registro r2 y el valor inmediato 50.
- Realiza operación lógica AND entre los registros r1 y r2.
- Para la ejecución.

Tras la ejecución del programa "alu_test.asm" con el programa np-asm creado por Juan Antonio explicado en el apartado 1.1 se generan los siguientes ficheros:

· alu_test.obj, contiene las instrucciones del programa en hexadecimal a introducir en la memoria de instrucciones.
0x000 0x400202
0x001 0x440432
0x002 0xfc0000

· alu_test.reg, contiene los datos en hexadecimal a introducir en el banco de registros.
0x001 0x0064
0x002 0x0005

· alu_test.var, contiene los datos en hexadecimal a introducir en la memoria de datos.
Vacío al no tener ninguna referencia a la memoria de datos.

Una vez creados los ficheros se procede a la ejecución del programa en el simulador del nanoprocesador y comprobación del mismo:

```
[root@localhost np_sim]# tclsh
% load ./nanosim.so package
% cd ..
% ns_load_imem alu_test.obj
0
% ns_load_rbank alu_test.reg
0
% ns_run
t0 0x000  0x400202[00 10]   IF add    r%1, r%2
t0                        DE
t0                        EX

t1 0x001  0x440432[00 11]   IF addi    r%2, 50
t1 0x000  0x400202[00 10]   DE add    r%1, r%2
t1                        EX

t2 0x002  0xfc0000?    IF halt
t2 0x001  0x440432[00 11]   DE addi    r%2, 50
t2 0x000  0x400202[00 10]   EX add    r%1, r%2

t3 0x002      IF
t3 0x002  0xfc0000?    DE halt
t3 0x001  0x440432[00 11]   EX addi    r%2, 50

t4 0x000      IF
t4 0x002      DE
t4 0x002  0xfc0000?    EX halt

% ns_dump_reg 1
105
0
% ns_dump_reg 2
55
```

Ilustración 36. Simulación programa “alu_test.asm”

Como se puede observar en la Ilustración 36. Simulación programa “alu_test.asm”:
· La carga de los ficheros sobre el simulador es correcta al mostrar por pantalla el número 0.
· La ejecución de la simulación es correcta, muestra la instrucción que se encuentra en cada etapa por cada ciclo.
· Las operaciones se han realizado correctamente, para comprobarlo mostramos por pantalla los registros destino de las operaciones:

- El registro r1 tiene como valor 105, como resultado de la operación add r1,r2 donde el registro r1 tiene el valor 100 y el registro r2 tiene el valor 5.
- El registro r2 tiene como valor 55, como resultado de la operación addi r2,50 donde el registro r2 tiene el valor 5 y se le suma el valor del número inmediato 50.

5.4.2.2 Programa con instrucciones de carga y almacenamiento en memoria de datos “load_store_test.asm”

El programa en ensamblador es el siguiente:

```
        .header
r3:     .reg %r3 1

        .data
        .loc 0x0
var1:   .word 10
var2:   .word 20

        .text
main:
```

```
        lwi %r1, 0
        lw %r2, r3
        swi r3, 3
        halt
    .end
```

Este programa realiza las siguientes acciones:
· Almacena inicialmente en el banco de registros los siguientes datos:
- En el registro r3 almacena el dato 1.

· Almacena inicialmente en la memoria de datos los siguientes datos:
- En la dirección 0 almacena el dato 10.
- En la dirección 1 almacena el dato 20.

· Ejecuta las operaciones:
- Carga el dato contenido en la dirección de memoria 0 sobre el registro r1.
- Carga el dato contenido en la dirección de memoria indicada por el registro r3 sobre el registro r2.
- Almacena en la dirección de memoria 3 el dato contenido en el registro r3.
- Para la ejecución.

Tras la ejecución del programa "load_store_test.asm" con el programa np-asm creado por Juan Antonio explicado en la sección 1.1 se generan los siguientes ficheros:

· load_store_test.obj, contiene las instrucciones del programa en hexadecimal a introducir en la memoria de instrucciones.
0x000 0x240200
0x001 0x200403
0x002 0x2c0603
0x003 0xfc0000

· load_store_test.reg, contiene los datos en hexadecimal a introducir en el banco de registros.
0x003 0x0001

· load_store_test.var, contiene los datos en hexadecimal a introducir en la memoria de datos.
0x000 0x000a
0x001 0x0014

Una vez creados los ficheros se procede a la ejecución del programa en el simulador del nanoprocesador y comprobación del mismo:

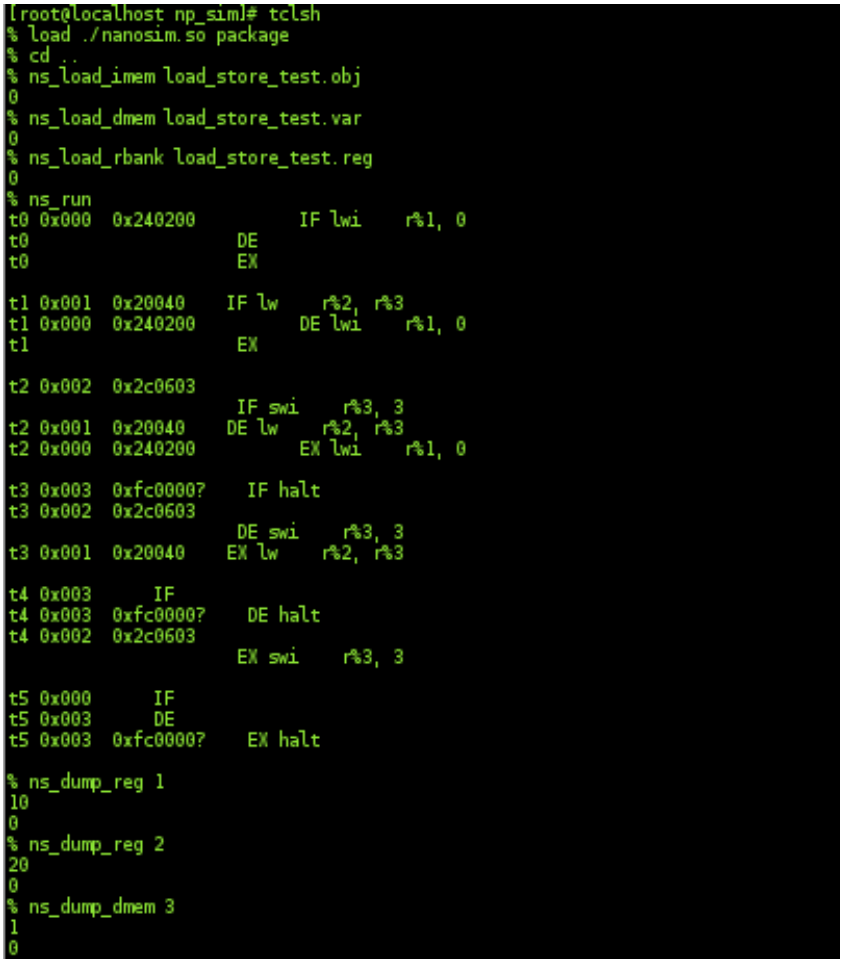


Ilustración 37. Simulación programa "load_store_test.asm"

Como se puede observar en la Ilustración 37. Simulación programa "load_store_test.asm":
· La carga de los ficheros sobre el simulador es correcta al mostrar por pantalla el número 0.
· La ejecución de la simulación es correcta, muestra la instrucción que se encuentra en cada etapa por cada ciclo.
· Las operaciones se han realizado correctamente, para comprobarlo mostramos por pantalla los registros y direcciones destino de las operaciones:

- El registro r1 tiene como valor 10, como resultado de la carga del dato procedente de la dirección 0 de la memoria de datos.
- El registro r2 tiene como valor 20, como resultado de la carga del datp procedente de la dirección 1 de la memoria de datos.
- La dirección 3 de la memoria de datos tiene como valor 1 al ser almacenado sobre ella el dato contenido en el registro r3.

5.4.2.3 Programa con instrucción de salto condicional "loop_test.asm"

El programa en ensamblador es el siguiente:

```
        .header
r1:   .reg %r1 5
r2:   .reg %r2 1

        .data
```

```
        .text
main:
        sub r1, r2
        bne 0
        halt
        .end
```

Este programa realiza las siguientes acciones:
· Almacena inicialmente en el banco de registros los siguientes datos:

- En el registro r1 almacena el dato 5.
- En el registro r2 almacena el dato 1.

· Ejecuta las operaciones:

- Resta al valor del registro r1 el valor del registro r2 y almacena el resultado en el registro r1.
- Compara si la anterior instrucción tuvo como resultado un valor distinto de 0, en caso de cumplirse la condición salta a la instrucción 0, si no la cumple no se produce el salto.
- Para la ejecución.

Tras la ejecución del programa "loop_test.asm" con el programa np-asm creado por Juan Antonio explicado en la sección 1.1 se generan los siguientes ficheros:

· loop_test.obj, contiene las instrucciones del programa en hexadecimal a introducir en la memoria de instrucciones.

```
0x000  0x480202
0x001  0x140000
0x002  0xfc0000
```

· loop_test.reg, contiene los datos en hexadecimal a introducir en el banco de registros.

```
0x001  0x0005
0x002  0x0001
```

· loop_test.var, contiene los datos en hexadecimal a introducir en la memoria de datos.
Vacío al no tener ninguna referencia a la memoria de datos.

Una vez creados los ficheros se procede a la ejecución del programa en el simulador del nanoprocesador y comprobación del mismo:

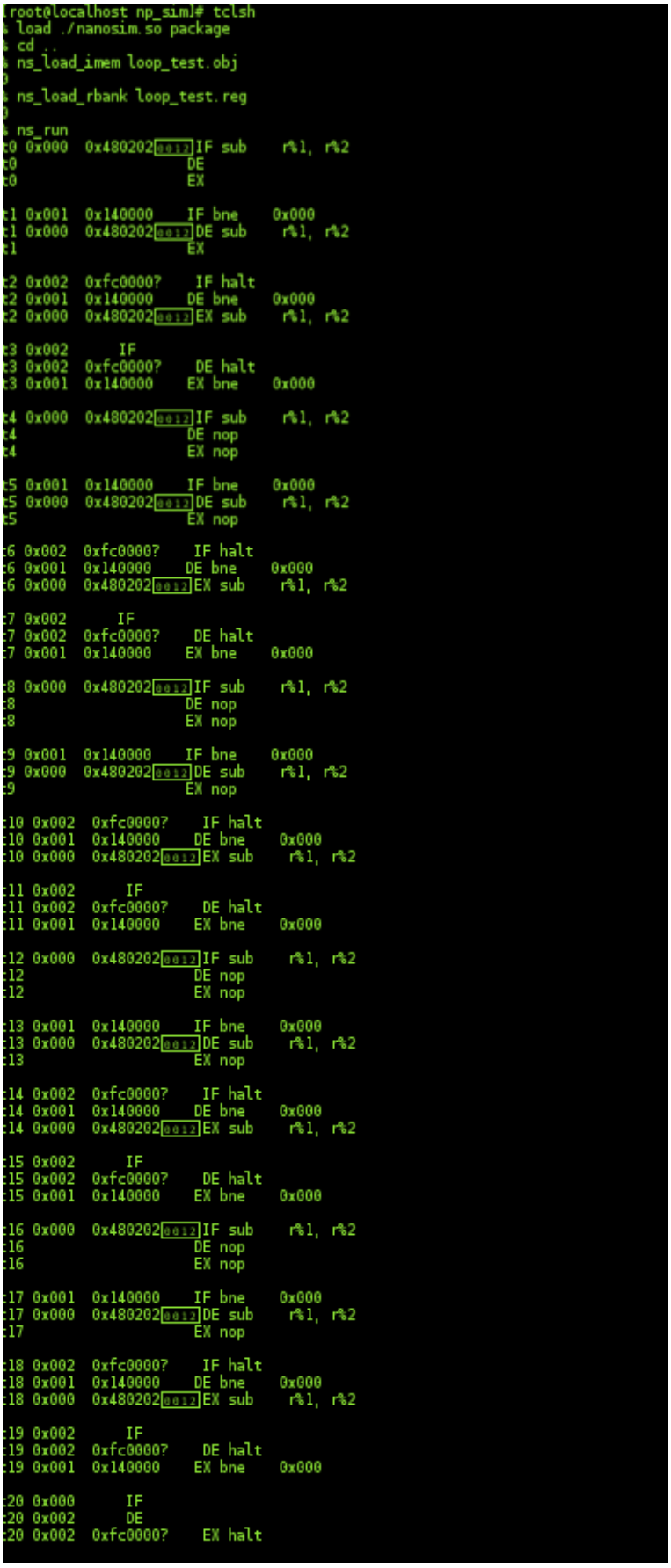


Ilustración 38. Simulación programa "loop_test.asm"

Como se puede observar en la Ilustración 38. Simulación programa “loop_test.asm”:

· La carga de los ficheros sobre el simulador es correcta al mostrar por pantalla el número 0.
· La ejecución de la simulación es correcta, muestra la instrucción que se encuentra en cada etapa por cada ciclo.
· Las operaciones se han realizado correctamente, para comprobarlo observamos la ejecución y que el bucle se ha ejecutado 4 veces.

6. Gestión del proyecto

En este apartado se muestra la planificación, el seguimiento y control de las actividades y de los recursos humanos que intervienen en el proyecto. Como consecuencia de este control es posible conocer en todo momento qué problemas se producen y resolverlos o paliarlos de manera inmediata.

6.1 Metodológica del proyecto

La metodología utilizada para este proyecto ha sido la MÉTRICA Versión 3, es una metodología de planificación, desarrollo y mantenimiento de sistemas de información, promovida por el Ministerio de Hacienda y Administraciones Públicas del Gobierno de España para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas. Esta metodología propia está basada en el modelo de procesos del ciclo de vida de desarrollo ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) así como en la norma ISO/IEC 15504 SPICE (Software Process Improvement And Assurance Standards Capability Determination) [18].

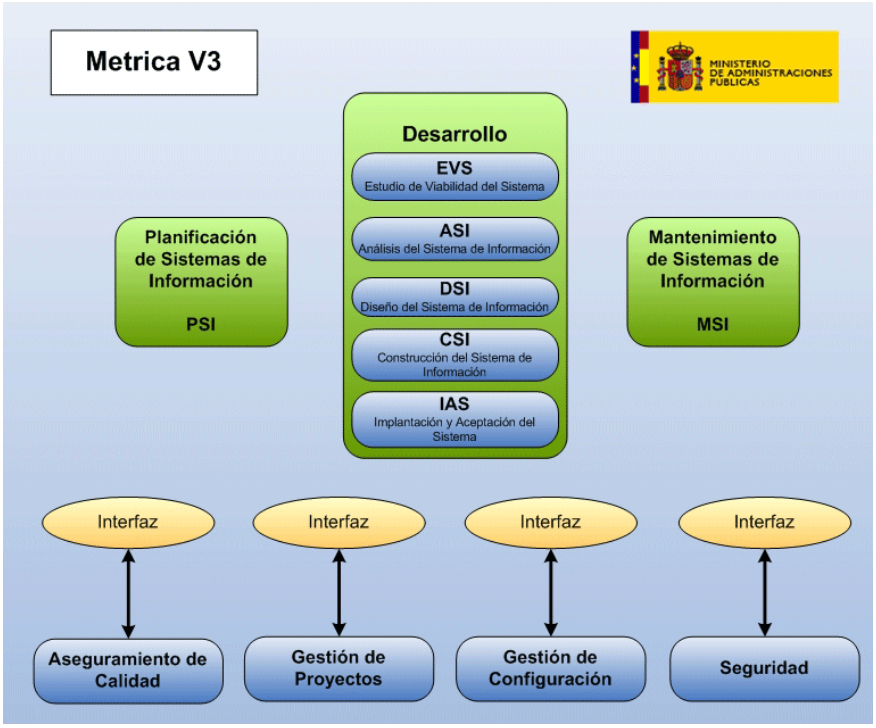


Ilustración 39. Metodología Métrica v3

Los procesos principales de la métrica son los siguientes:
· Planificación de Sistemas de Información (PSI), en este proceso se detalla el alcance del plan, se organiza el equipo de personas que lo va a llevar a cabo y se elabora un calendario de ejecución.

· Desarrollo de Sistemas de Información (DSI), esta debido a su complejidad se divide en cinco procesos:

- Estudio de Viabilidad del Sistema (EVS): En esta actividad se estudia el alcance de la necesidad planteada por el cliente o usuario, o como consecuencia de la realización de

un PSI, realizando una descripción general de la misma. Se determinan los objetivos, se inicia el estudio de los requisitos y se identifican las unidades organizativas afectadas estableciendo su estructura.

- Análisis del Sistema de Información (ASI): Esta actividad tiene como objetivo efectuar una descripción del sistema, delimitando su alcance, estableciendo las interfaces con otros sistemas e identificando a los usuarios representativos.
- Diseño del Sistema de Información (DSI): El objetivo del proceso es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información.
- Construcción del Sistema de Información (CSI): El objetivo de esta actividad es asegurar la disponibilidad de todos los medios y facilidades para que se pueda llevar a cabo la construcción del sistema de información.
- Implantación y aceptación del sistema (IAS):

· Mantenimiento de Sistemas de Información (MSI): En este proceso se realiza el registro de las peticiones de mantenimiento recibidas, con el fin de llevar el control de las mismas y de proporcionar datos estadísticos de peticiones recibidas o atendidas en un determinado periodo, sistemas que se han visto afectados por los cambios, en qué medida y el tiempo empleado en la resolución de dichos cambio. Este último proceso se desecha, debido a que el proyecto está orientado al diseño y desarrollo.

6.2 Ciclo de vida

Para ordenar las etapas que componen el desarrollo del proyecto, se ha utilizado un modelo de cascada retroalimentada. Este modelo está diseñado para esperar a que una etapa finalice para comenzar la siguiente y tras finalizar la etapa llevar a cabo una revisión final, que se encarga de determinar si el proyecto está listo o no para avanzar a la siguiente fase, sin embargo al ser retroalimentada permite volver a etapas anteriores para revisarlas y modificarlas si es necesario [19]. Este ciclo de vida consta de las siguientes etapas:

- Análisis de requisitos: Primera etapa del ciclo de vida, trata de capturar y describir detalladamente los requerimientos de funcionalidad y de calidad de servicio del simulador que se desarrolla.
- Diseño: En esta etapa se evalúa las soluciones alternativas y especifica la solución definitiva, características del sistema y herramientas que serán utilizadas para su desarrollo.
- Codificación: En esta etapa se desarrolla el simulador. Aquí surgen ya los primeros prototipos de pruebas para ensayos utilizados para corregir errores.
- Pruebas: Etapa en la cual se realiza una evaluación dinámica del simulador, esta evaluación se especifica previamente y se registran los resultados.
- Integración: En esta etapa se desarrolla la puesta en funcionamiento del simulador sobre el sistema.

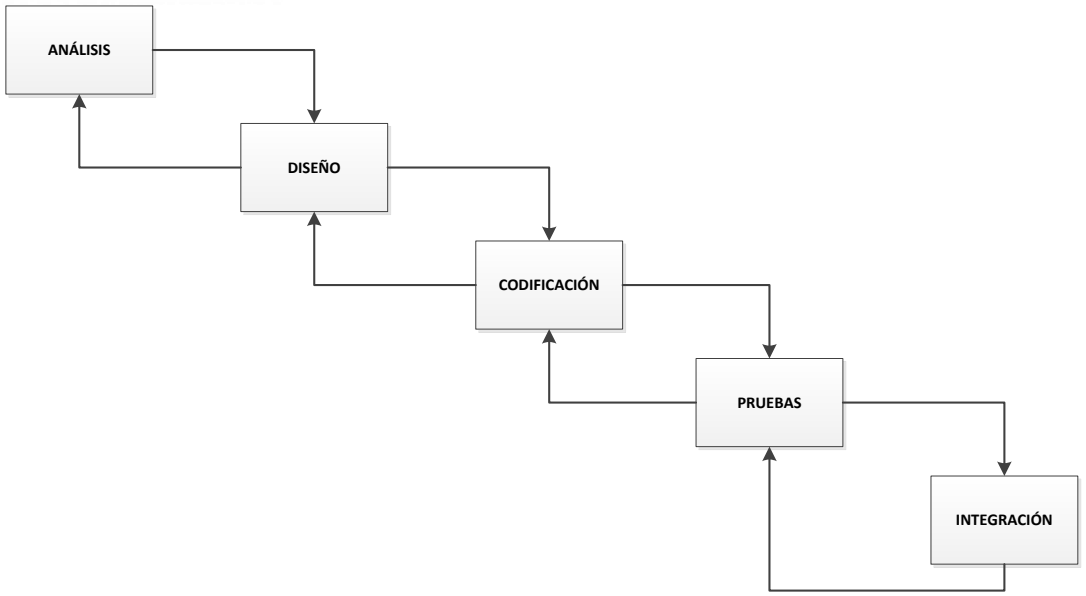


Ilustración 40. Método cascada retroalimentada

6.3 Planificación temporal del proyecto

En este apartado se detalla la planificación del proyecto, estimando las actividades, hitos y tareas realizadas durante el desarrollo del proyecto. A continuación se muestran una seria de ilustraciones con los diagramas de Gantt correspondientes a las tareas que se consideran necesarias para finalizar con éxito el proyecto.

6.3.1 Diagrama general

Este proyecto comenzará el día 19 de Enero y finalizará el día 18 de Septiembre de 2015 con 242 días de duración de los cuales 181 días son de trabajo. El proyecto constará de 7 fases.

- Planificación (8 días).
- Gestión de configuración (13 días).
- Estudio de viabilidad del sistema (24 días).
- Análisis del sistema de información (23 días).
- Diseño del sistema de información (83 días).
- Pruebas (22 días).
- Implantación y aceptación del sistema (8 días).

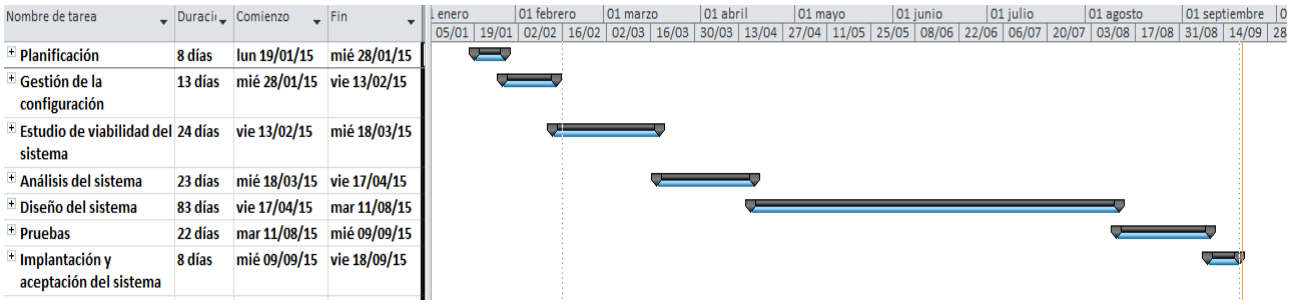


Ilustración 41. Diagrama Gantt general

6.3.2 Diagrama de Planificación

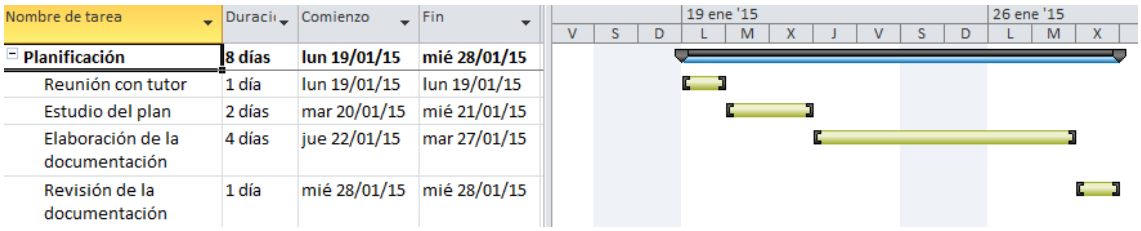
Esta es la primera fase del proyecto y tiene una duración de 8 días, esta fase consiste en la planificación de las actividades y tareas a realizar para cumplir con los objetivos del proyecto. Consta de las siguientes partes:

· Reunión con tutores: 1 día.

· Estudio del plan: 2 días.

· Elaboración de la documentación: 4 días.

· Revisión de la documentación: 1 día.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	**8 días**	**lun 19/01/15**	**mié 28/01/15**
Reunión con tutor	1 día	lun 19/01/15	lun 19/01/15
Estudio del plan	2 días	mar 20/01/15	mié 21/01/15
Elaboración de la documentación	4 días	jue 22/01/15	mar 27/01/15
Revisión de la documentación	1 día	mié 28/01/15	mié 28/01/15

Ilustración 42. Diagrama Gantt planificación

6.3.3 Diagrama de Gestión de la configuración

Esta es la segunda fase del proyecto y tiene una duración de 13 días, esta fase consiste en la estimación de la gestión de versiones, proceso de control de cambios y formato de la documentación. Consta de las siguientes partes:

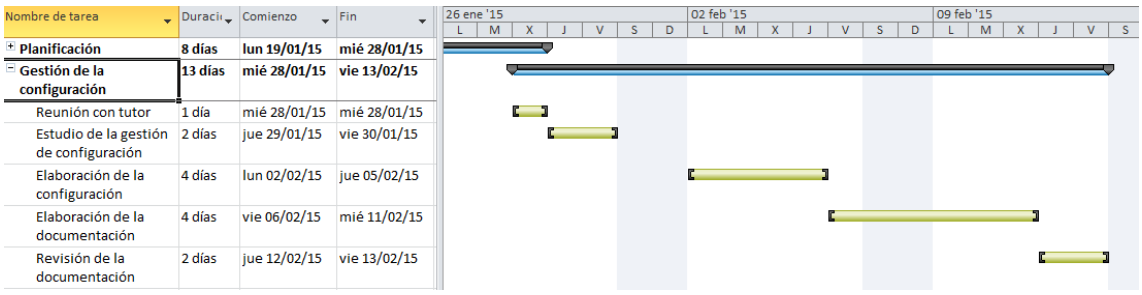
· Reunión con tutor: 1 día.

· Estudio de la gestión de configuración: 2 días.

· Elaboración de la configuración: 4 días.

· Elaboración de la documentación: 4 días.

· Revisión de la documentación: 2 días.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	**8 días**	**lun 19/01/15**	**mié 28/01/15**
Gestión de la configuración	**13 días**	**mié 28/01/15**	**vie 13/02/15**
Reunión con tutor	1 día	mié 28/01/15	mié 28/01/15
Estudio de la gestión de configuración	2 días	jue 29/01/15	vie 30/01/15
Elaboración de la configuración	4 días	lun 02/02/15	jue 05/02/15
Elaboración de la documentación	4 días	vie 06/02/15	mié 11/02/15
Revisión de la documentación	2 días	jue 12/02/15	vie 13/02/15

Ilustración 43. Diagrama Gantt Gestión de la configuración

6.3.4 Diagrama de Estudio de viabilidad del sistema

Esta es la tercera fase del proyecto y tiene una duración de 24 días, esta fase consiste en el análisis de un conjunto concreto de necesidades para proponer una solución a corto plazo, que

tenga en cuenta restricciones económicas, técnicas, legales y operativas. Consta de las siguientes partes:

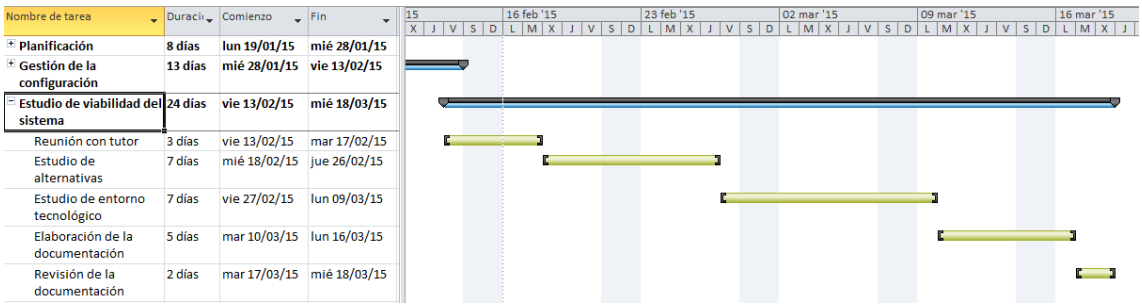
· Reunión con el tutor: 3 días.

· Estudio de las alternativas tecnológicas: 7 días.

· Estudio del entorno tecnológico: 7 días.

· Elaboración de la documentación: 5 días.

· Revisión de la documentación: 2 días.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	**8 días**	**lun 19/01/15**	**mié 28/01/15**
Gestión de la configuración	**13 días**	**mié 28/01/15**	**vie 13/02/15**
Estudio de viabilidad del sistema	**24 días**	**vie 13/02/15**	**mié 18/03/15**
Reunión con tutor	3 días	vie 13/02/15	mar 17/02/15
Estudio de alternativas	7 días	mié 18/02/15	jue 26/02/15
Estudio de entorno tecnológico	7 días	vie 27/02/15	lun 09/03/15
Elaboración de la documentación	5 días	mar 10/03/15	lun 16/03/15
Revisión de la documentación	2 días	mar 17/03/15	mié 18/03/15

Ilustración 44. Diagrama Gantt Estudio de viabilidad del sistema

6.3.5 Diagrama de Análisis del sistema de información

Esta es la cuarta fase del proyecto y tiene una duración de 23 días, esta fase consiste en determinar los objetivos y límites del proyecto, caracterizar su estructura y funcionamiento, marcar las directrices que permitan alcanzar los objetivos propuestos y evaluar sus consecuencias. Consta de las siguientes partes:

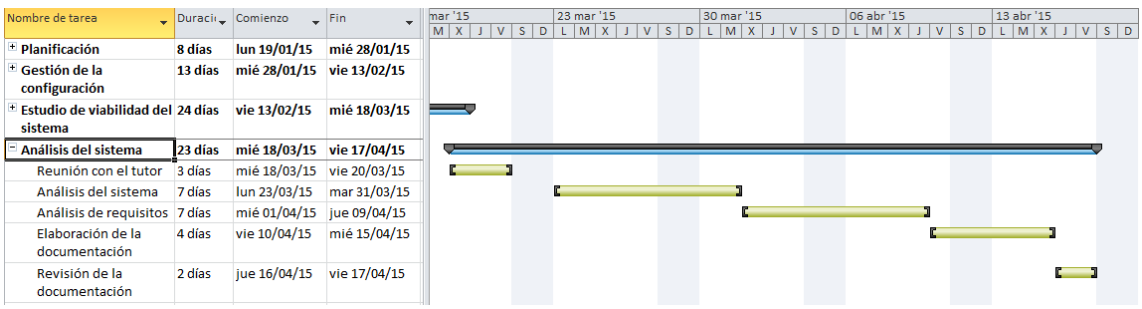
· Reunión con el tutor: 3 días.

· Análisis del sistema: 7 días.

· Análisis de requisitos: 7 días.

· Elaboración de la documentación: 4 días.

· Revisión de la documentación: 2 días.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	**8 días**	**lun 19/01/15**	**mié 28/01/15**
Gestión de la configuración	**13 días**	**mié 28/01/15**	**vie 13/02/15**
Estudio de viabilidad del sistema	**24 días**	**vie 13/02/15**	**mié 18/03/15**
Análisis del sistema	**23 días**	**mié 18/03/15**	**vie 17/04/15**
Reunión con el tutor	3 días	mié 18/03/15	vie 20/03/15
Análisis del sistema	7 días	lun 23/03/15	mar 31/03/15
Análisis de requisitos	7 días	mié 01/04/15	jue 09/04/15
Elaboración de la documentación	4 días	vie 10/04/15	mié 15/04/15
Revisión de la documentación	2 días	jue 16/04/15	vie 17/04/15

Ilustración 45. Diagrama Gantt Análisis del sistema

6.3.6 Diagrama de Diseño del sistema de información

Esta es la quinta fase del proyecto y tiene una duración de 83 días, esta fase consiste en definir la arquitectura hardware y software, componentes, módulos y datos para satisfacer los requisitos originados en la fase anterior. Consta de las siguientes partes:

· Reunión con el tutor: 4 días.

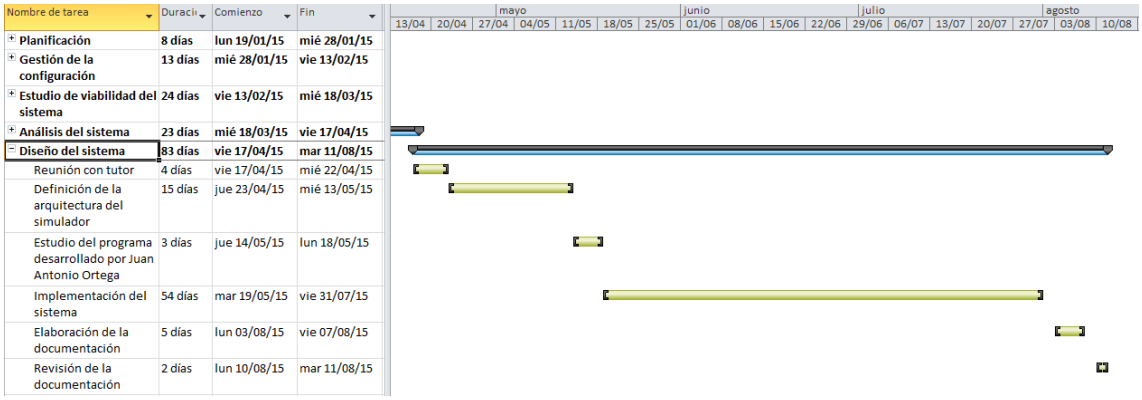
· Definición de la arquitectura del sistema: 15 días.

· Estudio del programa desarrollado por Juan Antonio: 3 días.

· Implementación del sistema: 54 días.

· Elaboración de la documentación: 5 días.

· Revisión de la documentación: 2 días.



Nombre de tarea	Duraci	Comienzo	Fin
Planificación	**8 días**	**lun 19/01/15**	**mié 28/01/15**
Gestión de la configuración	**13 días**	**mié 28/01/15**	**vie 13/02/15**
Estudio de viabilidad del sistema	**24 días**	**vie 13/02/15**	**mié 18/03/15**
Análisis del sistema	**23 días**	**mié 18/03/15**	**vie 17/04/15**
Diseño del sistema	**83 días**	**vie 17/04/15**	**mar 11/08/15**
Reunión con tutor	4 días	vie 17/04/15	mié 22/04/15
Definición de la arquitectura del simulador	15 días	jue 23/04/15	mié 13/05/15
Estudio del programa desarrollado por Juan Antonio Ortega	3 días	jue 14/05/15	lun 18/05/15
Implementación del sistema	54 días	mar 19/05/15	vie 31/07/15
Elaboración de la documentación	5 días	lun 03/08/15	vie 07/08/15
Revisión de la documentación	2 días	lun 10/08/15	mar 11/08/15

Ilustración 46. Diagrama Gantt diseño del sistema

6.3.7 Diagrama de Pruebas

Esta es la sexta fase del proyecto y tiene una duración de 22 días, esta fase consiste en la investigación empírica y técnica cuyo objetivo es proporcionar información sobre la calidad del producto. Consta de las siguientes partes:

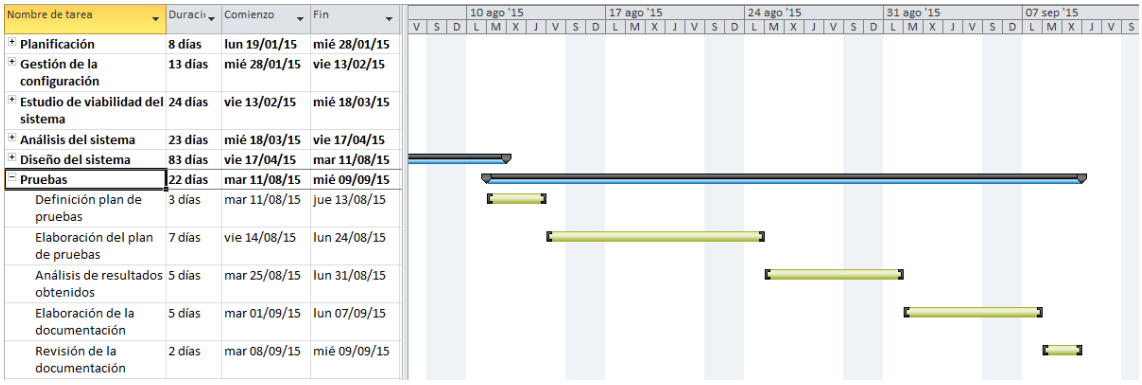
· Definición del plan de pruebas: 3 días.

· Elaboración del plan de pruebas: 7 días.

· Análisis de resultados obtenidos: 5 días.

· Elaboración de la documentación: 5 días.

· Revisión de la documentación: 2 días.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	8 días	lun 19/01/15	mié 28/01/15
Gestión de la configuración	13 días	mié 28/01/15	vie 13/02/15
Estudio de viabilidad del sistema	24 días	vie 13/02/15	mié 18/03/15
Análisis del sistema	23 días	mié 18/03/15	vie 17/04/15
Diseño del sistema	83 días	vie 17/04/15	mar 11/08/15
Pruebas	22 días	mar 11/08/15	mié 09/09/15
Definición plan de pruebas	3 días	mar 11/08/15	jue 13/08/15
Elaboración del plan de pruebas	7 días	vie 14/08/15	lun 24/08/15
Análisis de resultados obtenidos	5 días	mar 25/08/15	lun 31/08/15
Elaboración de la documentación	5 días	mar 01/09/15	lun 07/09/15
Revisión de la documentación	2 días	mar 08/09/15	mié 09/09/15

Ilustración 47. Diagrama Gantt Pruebas

6.3.8 Diagrama de Implantación y aceptación del sistema

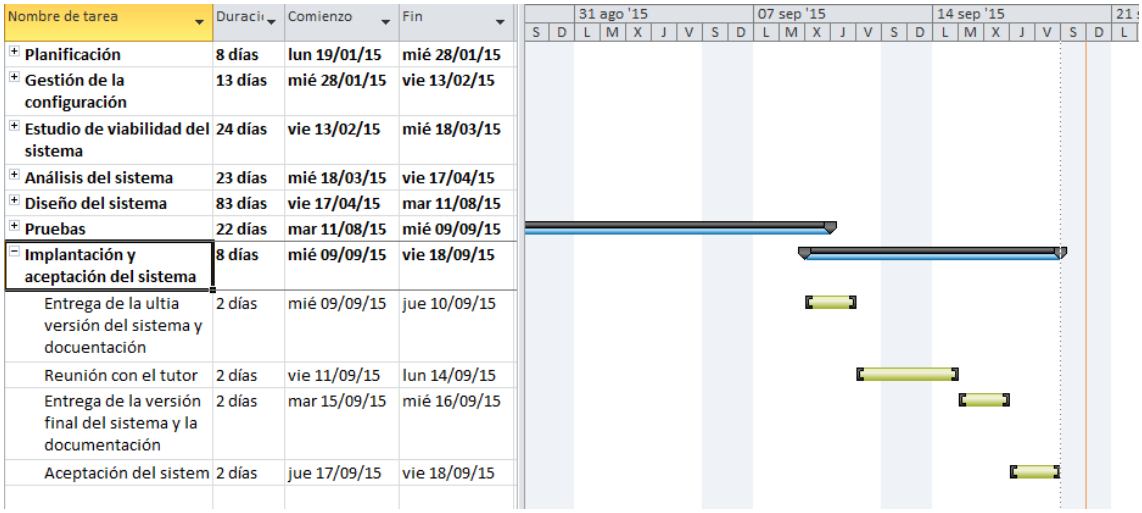
Esta es la última fase del proyecto y tiene una duración de 8 días, esta fase contiene la entrega y aceptación del sistema en su totalidad. Consta de las siguientes partes:

· Entrega de la última versión del sistema y la documentación: 2 días.

· Reunión con el tutor: 2 días.

· Entrega de la versión final del sistema y la documentación: 2 días.

· Aceptación del sistema: 2 días.



Nombre de tarea	Duración	Comienzo	Fin
Planificación	8 días	lun 19/01/15	mié 28/01/15
Gestión de la configuración	13 días	mié 28/01/15	vie 13/02/15
Estudio de viabilidad del sistema	24 días	vie 13/02/15	mié 18/03/15
Análisis del sistema	23 días	mié 18/03/15	vie 17/04/15
Diseño del sistema	83 días	vie 17/04/15	mar 11/08/15
Pruebas	22 días	mar 11/08/15	mié 09/09/15
Implantación y aceptación del sistema	8 días	mié 09/09/15	vie 18/09/15
Entrega de la ultia versión del sistema y docuentación	2 días	mié 09/09/15	jue 10/09/15
Reunión con el tutor	2 días	vie 11/09/15	lun 14/09/15
Entrega de la versión final del sistema y la documentación	2 días	mar 15/09/15	mié 16/09/15
Aceptación del sistem	2 días	jue 17/09/15	vie 18/09/15

Ilustración 48. Diagrama Gantt Implantación y aceptación del sistema

7. Pruebas de verificación

7.1 Diseño de las pruebas

En este apartado se especifican las pruebas realizadas que el simulador debe superar para que sea utilizable, esto significa que cumple con las necesidades descritas en este documento.

7.1.1 Plantilla

A continuación se detalla el formato que seguirán las plantillas de las pruebas:

· **Identificador:** cada prueba incluirá una identificación para facilitar su traza por las fases subsiguientes, tal que PRS-YYY, dónde PRS indica que es una prueba del sistema e YYY es el número correspondiente a la prueba.

· **Descripción:** Exposición de la prueba.

· **Objetivo:** Propósito de realizar la prueba.

· **Requisitos relacionados:** Requisitos a los que hace referencia la prueba.

· **Resultado deseado:** Resultado que se planea obtener tras ejecutar la prueba.

· **Resultado obtenido:** Resultado tras la ejecución de la prueba.

PRX-YYY
Descripción:
Objetivo:
Procedimiento:
Requisitos relacionados:
Resultado esperado:
Resultado obtenido:

Tabla 83. Plantilla de pruebas

7.1.2 Pruebas del sistema

A continuación se muestran de manera detallada las pruebas que se han realizado sobre el simulador.

PRS-001	
Descripción:	Compilar el simulador del nanoprocesador en Windows.
Objetivo:	Comprobar que el simulador puede ser compilado en Windows.
Procedimiento:	Desde el cmd de Windows situarse en el directorio donde se encuentre el simulador y escribir make.
Requisitos relacionados:	RC-002
Resultado esperado:	Debe aparecer en pantalla los comandos de compilación ejecutados con el make y debe haberse creado los siguientes archivos en el directorio donde nos encontramos: · nanosim_wrap.c · nanosim_wrap.o · nanosim.so
Resultado obtenido:	Éxito. No existen problemas en la compilación del simulador y se han creado los archivos nanosim_wrap.c, nanosim_wrap.o y nanosim.so.

Tabla 84. Prueba del sistema 1 - Compilar el simulador del nanoprocesador en Windows

PRS-002	
Descripción:	Compilar el simulador del nanoprocesador en Linux.
Objetivo:	Comprobar que el simulador puede ser compilado en Linux.
Procedimiento:	Desde el Terminal de Linux situarse en el directorio donde se encuentre el simulador y escribir make.
Requisitos relacionados:	RC-002
Resultado esperado:	Debe aparecer en pantalla los comandos de compilación ejecutados con el make y debe haberse creado los siguientes archivos en el directorio donde nos encontramos: · nanosim_wrap.c · nanosim_wrap.o · nanosim.so
Resultado obtenido:	Éxito. No existen problemas en la compilación del simulador y se han creado los archivos nanosim_wrap.c, nanosim_wrap.o y nanosim.so.

Tabla 85. Prueba del sistema 2 - Compilar el simulador del nanoprocesador en Linux

PRS-003	
Descripción:	Cargar el paquete del simulador en la herramienta tclsh.
Objetivo:	Comprobar que se carga correctamente el paquete del simulador en tclsh.

Procedimiento:	Abrir tclsh desde cualquier terminal, y ejecutar el siguiente comando: load path/nanosim.so package donde path es la ruta donde se encuentra el archivo nanosim.so.
Requisitos relacionados:	RC-001, RR-003
Resultado esperado:	La herramienta tclsh deberá mostrar por pantalla un 0 lo que significará que el paquete se ha cargado correctamente.
Resultado obtenido:	**Éxito**. No existen problemas en la carga del paquete y muestra por pantalla un 0.

Tabla 86. Prueba del sistema 3 - Cargar el paquete del simulador en la herramienta tclsh

PRS-004	
Descripción:	Cargar con el simulador el fichero file.var
Objetivo:	Comprobar que el simulador lee el fichero y almacena los datos leídos en la memoria de datos del nanoprocesador.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_load_dmem file.var**
Requisitos relacionados:	RC-004, RC-007, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un 0 para demostrar al usuario que la ejecución ha sido correcta y en la memoria de datos del nanoprocesador deben aparecer los datos que el fichero file.var contiene.
Resultado obtenido:	**Éxito**. No existen problemas en la carga de los datos en la memoria de datos del nanoprocesador y muestra por pantalla un 0.

Tabla 87. Prueba del sistema 4 - Cargar con el simulador el fichero "file.var"

PRS-005	
Descripción:	Cargar con el simulador el fichero file.reg
Objetivo:	Comprobar que el simulador lee el fichero y almacena los datos leídos en el banco de registros del nanoprocesador.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_load_rbank file.reg**
Requisitos relacionados:	RC-005, RC-008, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un 0 para demostrar al usuario que la ejecución ha sido correcta y en el banco de registros del nanoprocesador deben aparecer los datos que el fichero file.reg contiene.

Resultado obtenido:	**Éxito**. No existen problemas en la carga de los datos en el banco de registros del nanoprocesador y muestra por pantalla un 0.

Tabla 88. Prueba del sistema 5 - Cargar con el simulador el fichero "file.reg"

PRS-006	
Descripción:	Cargar con el simulador el fichero file.obj
Objetivo:	Comprobar que el simulador lee el fichero y almacena los datos leídos en la memoria de instrucciones del nanoprocesador.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_load_imem file.obj**
Requisitos relacionados:	RC-006, RC-009, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un 0 para demostrar al usuario que la ejecución ha sido correcta y en la memoria de instrucciones del nanoprocesador deben aparecer los datos que el fichero file.obj contiene.
Resultado obtenido:	**Éxito**. No existen problemas en la carga de los datos en la memoria de instrucciones del nanoprocesador y muestra por pantalla un 0.

Tabla 89. Prueba del sistema 6 - Cargar con el simulador el fichero "file.obj"

PRS-007	
Descripción:	Mostrar por pantalla el valor de una dirección de la memoria de datos del nanoprocesador introduciendo la dirección en hexadecimal.
Objetivo:	Comprobar que el simulador imprime el valor en decimal corresponde al contenido en la dirección de memoria escrita en hexadecimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_dmem 0x00F**
Requisitos relacionados:	RC-004, RC-007, RC-020, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un valor que está contenido en la dirección 15 de la memoria de datos.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 5 que es el valor que tras cargar file.var la memoria de datos tiene en la dirección 15.

Tabla 90. Prueba del sistema 7 - Mostrar por pantalla dato de memoria introduciendo dirección hexadecimal

PRS-008	
Descripción:	Mostrar por pantalla el valor de una dirección de la memoria de datos del nanoprocesador introduciendo la dirección en decimal.

Objetivo:	Comprobar que el simulador imprime el valor en decimal corresponde al contenido en la dirección de memoria escrita en decimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_dmem 15**
Requisitos relacionados:	RC-004, RC-007, RC-020, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un valor que está contenido en la dirección 15 de la memoria de datos.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 5 que es el valor que tras cargar file.var la memoria de datos tiene en la dirección 15.

Tabla 91. Prueba del sistema 8 - Mostrar por pantalla dato de memoria introduciendo dirección decimal

PRS-009	
Descripción:	Mostrar por pantalla la instrucción de una dirección de la memoria de instrucciones del nanoprocesador introduciendo la dirección en hexadecimal.
Objetivo:	Comprobar que el simulador imprime la instrucción con sus pertinentes campos que corresponde al contenido en la dirección de memoria escrita en hexadecimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_imem 0x00A**
Requisitos relacionados:	RC-006, RC-009, RC-021, RC-022, RC-023, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla una instrucción que está contenido en la dirección 10 de la memoria de instrucciones.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 0xA80202 bset r%1, r%2 que es el valor que tras cargar file.obj la memoria de instrucciones tiene en la dirección 10.

Tabla 92. Prueba del sistema 9 - Mostrar por pantalla instrucción de memoria de instrucciones introduciendo dirección hexadecimal

PRS-010	
Descripción:	Mostrar por pantalla la instrucción de una dirección de la memoria de instrucciones del nanoprocesador introduciendo la dirección en decimal.
Objetivo:	Comprobar que el simulador imprime la instrucción con sus pertinentes campos que corresponde al contenido en la dirección de memoria escrita en decimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_imem 10**
Requisitos relacionados:	RC-006, RC-009, RC-021, RC-022, RC-023, RR-003

Resultado esperado:	El simulador debe mostrar por pantalla una instrucción que está contenido en la dirección 10 de la memoria de instrucciones.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 0xA80202 bset r%1, r%2 que es el valor que tras cargar file.obj la memoria de instrucciones tiene en la dirección 10.

Tabla 93. Prueba del sistema 10 - Mostrar por pantalla instrucción de memoria de instrucciones introduciendo dirección decimal

PRS-011	
Descripción:	Mostrar por pantalla el valor de registro del banco de registros del nanoprocesador introduciendo el número de registro en hexadecimal.
Objetivo:	Comprobar que el simulador imprime el valor en decimal corresponde al contenido en el registro del banco de registros escrito en hexadecimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_reg 0x008**
Requisitos relacionados:	RC-005, RC-008, RC-019, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un valor que está contenido en la dirección 8 del banco de registros.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 32 que es el valor que tras cargar file.reg el banco de registros tiene en el registro r8.

Tabla 94. Prueba del sistema 11 - Mostrar por pantalla dato de banco de registros introduciendo registro en hexadecimal

PRS-012	
Descripción:	Mostrar por pantalla el valor de registro del banco de registros del nanoprocesador introduciendo el número de registro en decimal.
Objetivo:	Comprobar que el simulador imprime el valor en decimal corresponde al contenido en el registro del banco de registros escrito en decimal.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_dump_reg 8**
Requisitos relacionados:	RC-005, RC-008, RC-019, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla un valor que está contenido en la dirección 8 del banco de registros.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 32 que es el valor que tras cargar file.reg el banco de registros tiene en el registro8.

Tabla 95. Prueba del sistema 12 - Mostrar por pantalla dato de banco de registros introduciendo registro en decimal

PRS-013

Descripción:	Recibir el valor de una dirección de la memoria de datos del nanoprocesador introduciendo la dirección.
Objetivo:	Comprobar que el simulador devuelve el valor en decimal corresponde al contenido en la dirección de memoria escrita.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_read_dmem 15**
Requisitos relacionados:	RC-004, RC-007, RC-018, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe devolver un valor que está contenido en la dirección 15 de la memoria de datos.
Resultado obtenido:	**Éxito**. El simulador devuelve el número 5 que es el valor que tras cargar file.var la memoria de datos tiene en la dirección 15.

Tabla 96. Prueba del sistema 13 - Recibir dato de dirección de memoria de datos

PRS-014	
Descripción:	Recibir el valor de registro del banco de registros del nanoprocesador introduciendo el número de registros.
Objetivo:	Comprobar que el simulador recibe el valor en decimal y que corresponde al contenido en el registro del banco de registros escrito.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_read_reg 0x008**
Requisitos relacionados:	RC-005, RC-008, RC-017, RC-023, RC-024, RR-003
Resultado esperado:	El simulador debe devolver un valor que está contenido en la dirección 8 del banco de registros.
Resultado obtenido:	**Éxito**. El simulador devuelve el número 32 que es el valor que tras cargar file.reg el banco de registros tiene en el registro8.

Tabla 97. Prueba del sistema 14 - Recibir dato de registro de banco de registros

PRS-015	
Descripción:	Escribir un valor en una dirección de la memoria de datos del nanoprocesador introduciendo la dirección y un número.
Objetivo:	Comprobar que el simulador almacena el valor correcto en la dirección correcta de la memoria de datos.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_write_dmem 15 3** y después **ns_dump_dmem 15.**
Requisitos relacionados:	RC-004, RC-016, RC-023, RR-003

Resultado esperado:	El simulador debe mostrar por pantalla 0 al ejecutar el comando **ns_write_dmem 15 3** y al ejecutar el comando **ns_dump_dmem 15** debe mostrar por pantalla 3.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 0 al ejecutar el primer comando y 3 al ejecutar el segundo comando.

Tabla 98. Prueba del sistema 15 - Escribir dato en dirección de memoria de datos

PRS-016	
Descripción:	Escribir un valor en un registro del banco de registros del nanoprocesador introduciendo el número de registro y un número.
Objetivo:	Comprobar que el simulador almacena el valor correcto en el registro correcto del banco de registros.
Procedimiento:	Tras cargar el paquete del simulador en tclsh, ejecutar el comando **ns_write_reg 9 3** y después **ns_dump_reg 9.**
Requisitos relacionados:	RC-005, RC-015, RC-023, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla 0 al ejecutar el comando **ns_write_reg 9 3** y al ejecutar el comando **ns_dump_reg 9** debe mostrar por pantalla 3.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla 0 al ejecutar el primer comando y 3 al ejecutar el segundo comando.

Tabla 99. Prueba del sistema 16 - Escribir dato en registro de banco de registros

PRS-017	
Descripción:	Ejecutar un programa ensamblador en el simulador y que la simulación se muestre por pantalla.
Objetivo:	Comprobar que el programa ensamblador se ejecuta correctamente en el simulador y que se muestra por pantalla.
Procedimiento:	Cargar simulador en tclsh: load ./nanosim.so package Cargar las instrucciones del programa: ns_load_imem file.obj, este fichero contiene dos instrucciones, addi r1, 5 y halt, donde r1 tiene el valor 0 al no cargar ningún fichero .reg. Ejecutar el programa: ns_run Comprobar que el registro del banco de registros ha sido modificado: ns_dump_reg 1
Requisitos relacionados:	RC-003, RC-004, RC-005, RC-006, RC-010, RC-012, RC-014, RC-022, RC-028, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla el pipeline por cada ciclo, según los cálculos previos realizados debe ejecutarse en 4 ciclos y haber modificado el registro 1 del banco de registros.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla el pipeline por cada ciclo al ejecutar ns_run, el número de ciclos mostrado coincide con 4 y muestra por pantalla el numero 5 al ejecutar el comando ns_dump_reg 1.

Tabla 100. Prueba del sistema 17 - Ejecutar programa ensamblador y mostrar por pantalla

PRS-018	
Descripción:	Parar la ejecución de un programa ensamblador en el simulador.
Objetivo:	Comprobar que el programa en simulación se detiene al presionar ctrl+C y permite volver a ejecutar correctamente el simulador desde el ciclo donde lo paramos.
Procedimiento:	Cargar simulador en tclsh: load ./nanosim.so package Cargar las instrucciones del programa: ns_load_imem file.obj, este fichero contiene dos instrucciones, addi r1, 5 y bnq 0, donde r1 tiene el valor 0 al no cargar ningún fichero .reg y la segunda compara si la anterior instrucción dio como resultado 0, al tratarse de una suma nunca será 0 y por lo tanto tenemos un ciclo que se ejecutará infinitas veces permitiéndonos parar el programa y volver a ejecutarlo las veces que queramos para comprobar el correcto funcionamiento. Ejecutar el programa: ns_run Detener el programa presionando ctrl+C Ejecutar de nuevo el programa: ns_run
Requisitos relacionados:	RC-003, RC-004, RC-005, RC-006, RC-010, RC-012, RC-013, RC-014, RC-022, RC-028, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla el pipeline por cada ciclo y detenerse al presionar ctrl+C y al ejecutar de nuevo el programa debe ejecutarse desde el último ciclo mostrado en la anterior ejecución.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla el pipeline por cada ciclo al ejecutar ns_run, y al pararlo muestra de nuevo el terminal a la espera de que el usuario introduzca un comando y de nuevo al ejecutar de nuevo el programa muestra el pipeline por ciclo comenzando donde lo detuvimos.

Tabla 101. Prueba del sistema 18 - Parar simulación en ejecución

PRS-019	
Descripción:	Ejecutar un programa ensamblador en el simulador ciclo a ciclo y que se muestre por pantalla y reiniciar el simulador tras la ejecución.
Objetivo:	Comprobar que el programa ensamblador se ejecuta ciclo a ciclo correctamente en el simulador y que es posible reiniciar la simulación después de ejecutar un programa.
Procedimiento:	Cargar simulador en tclsh: load ./nanosim.so package Cargar las instrucciones del programa: ns_load_imem file.obj, este fichero contiene dos instrucciones, addi r1, 5 y halt, donde r1 tiene el valor 0 al no cargar ningún fichero .reg. Ejecutar el programa ciclo a ciclo repitiendo 4 veces el siguiente comando: ns_step Comprobar que el registro del banco de registros ha sido modificado: ns_dump_reg 1 Ejecutar el reinicio del simulador: ns_restart Comprobar el reinicio: ns_dump_reg 1

Requisitos relacionados:	RC-003, RC-004, RC-005, RC-006, RC-010, RC-011, RC-014, RC-028, RC-022, RR-003
Resultado esperado:	El simulador debe mostrar por pantalla el pipeline de un único ciclo cada vez que ejecutamos el comando ns_step, al ejecutarlo 4 veces debe haber modificado el registro 1 del banco de registros y tras el reinicio el registro 1 debe tener el valor 0.
Resultado obtenido:	**Éxito**. El simulador muestra por pantalla el pipeline por cada ciclo al ejecutar ns_run y muestra por pantalla el numero 5 al ejecutar el comando ns_dump_reg 1 por primera vez, sin embargo tras la ejecución de ns_restart su valor es 0.

Tabla 102. Prueba del sistema 19 - Ejecutar simulación ciclo a ciclo , mostrar por pantalla y reiniciar simulación

PRS-020	
Descripción:	Mostrar por pantalla el valor de los registros de estado de la ALU del nanoprocesador simulado.
Objetivo:	Comprobar que el simulador imprime el bit de cada registro de estado y que coincide con la operación anterior realizada.
Procedimiento:	a) Para mostrar por pantalla 1 cuando imprimimos el registro de estado Z: · Cargamos el fichero de instrucciones file2.obj que contiene la instrucción subi r1, 5 y halt. ns_load_imem file2.obj · Cargamos el fichero de valores de registro file2.reg que contiene el valor 5 para el registro 1. ns_load_rbank file2.reg ·Ejecutamos la simulación. np_run · Comprobamos el valor de Z. ns_dump_Z b) Para mostrar por pantalla 0 cuando imprimimos el registro de estado Z: · Mismo procedimiento, cambiando la instrucción subi r1,5 por r1, 8 c) Para mostrar por pantalla 1 cuando imprimimos el registro de estado N: Mismo procedimiento que b) d) Para mostrar por pantalla 0 cuando imprimimos el registro de estado N: Mismo procedimiento que a) e) Para mostrar por pantalla 1 cuando imprimimos el registro de estado V: · Cargamos el fichero de instrucciones file3.obj que contiene una instrucción que provoca overflow.

	· Comprobamos el valor de V. ns_dump_V f) Para mostrar por pantalla 0 cuando imprimimos el registro de estado V: · Cargamos el fichero de instrucciones file4.obj que contiene una instrucción que no provoca overflow. · Comprobamos el valor de V. ns_dump_V
Requisitos relacionados:	RC-024, RC-029, RR-003
Resultado esperado:	Según el caso: a) Mostrar por pantalla 1. b) Mostrar por pantalla 0. c) Mostrar por pantalla 1. d) Mostrar por pantalla 0. e) Mostrar por pantalla 1. f) Mostrar por pantalla 0.
Resultado obtenido:	Éxito. El simulador muestra por pantalla el valor esperado para cada caso.

Tabla 103. Prueba del sistema 20 - Mostrar por pantalla valor de registros de estado ALU

PRS-021	
Descripción:	Mostrar por pantalla la lista de comandos que el simulador contiene e información de uno de ellos
Objetivo:	Comprobar que el simulador muestra todos los comandos existentes en el simulador y que la descripción que ofrece de cada comando es correcta y corresponde al comando.
Procedimiento:	Cargar simulador en tclsh: load ./nanosim.so package Ejecutar comando ns_help y ns_help_info ns_load_imem
Requisitos relacionados:	RC-025, RC-026
Resultado esperado:	El simulador debe mostrar por pantalla la lista de comandos que ofrece el simulador y la información correspondiente a la instrucción ns_load_imem
Resultado obtenido:	Éxito. El simulador muestra por pantalla la lista de comandos y la información del comando ns_load_imem.

Tabla 104. Prueba del sistema 21 - Mostrar por pantalla lista de comandos del simulador e información

7.1.3 Análisis de consistencia

Después de realizar todas las pruebas descritas en el anterior apartado, se muestra la matriz de trazabilidad entre las pruebas realizadas y los requisitos que verifica. Como se puede observar las pruebas realizadas cubren todos los requisitos.

	RC-001	RC-002	RC-003	RC-004	RC-005	RC-006	RC-007	RC-008	RC-009	RC-010	RC-011	RC-012	RC-013	RC-014	RC-015	RC-016	RC-017	RC-018	RC-019	RC-020	RC-021	RC-022	RC-023	RC-024	RC-025	RC-026	RC-027	RC-028	RC-029	RC-030
PRS-001		X																												
PRS-002		X																												
PRS-003	X																													
PRS-004				X			X																							
PRS-005					X			X																						
PRS-006						X			X																					
PRS-007				X			X													X			X	X						
PRS-008				X			X													X			X	X						
PRS-009						X			X												X		X	X						

PRS-010						X			X												X		X	X					
PRS-011					X			X											X				X	X					
PRS-012					X			X											X				X	X					
PRS-013				X			X											X					X	X					
PRS-014					X			X									X						X	X					
PRS-015				X												X							X						
PRS-016					X										X								X						
PRS-017			X	X	X	X				X		X		X								X					X		
PRS-018			X	X	X	X				X		X	X									X					X		
PRS-019			X	X	X	X				X	X			X								X				X	X		
PRS-020																								X				X	X
PRS-021																									X	X			

Tabla 105. Matriz de trazabilidad entre requisitos - pruebas

Los requisitos no introducidos en la tabla están relacionados con la metodología, herramientas y lenguaje a utilizar, por lo tanto, para comprobar estos requisitos bastaría con estudiar el documento y el simulador.

8. Presupuesto

En este apartado se muestra el cálculo de los costes de la realización del proyecto y el presupuesto final del mismo.

8.1 Costes de Software

En este apartado se desglosa las herramientas software utilizadas para este proyecto y su coste en función del tiempo utilizado.

Software	Precio	Utilización	Periodo desamortización	Coste aplicable
Microsoft Windows 7 Professional	143€	6 meses	36 meses	23,83€
Microsoft Windows 8	94,75€	6 meses	36 meses	15,79€
Linux Centos 7	-	6 meses	36 meses	-
VMware Player	71,95€	6 meses	36 meses	11,99€
Microsoft Office 2010	96,80€	6 meses	36 meses	16,13€
Microsoft Visio 2010	579,60€	6 meses	36 meses	96,60€
Microsoft Project 2010	539,66€	6 meses	36 meses	89,94€
NotePad ++	-	6 meses	36 meses	-
SWIG	-	6 meses	36 meses	-
Tclsh	-	6 meses	36 meses	-
ModelSim	930€	6 meses	36 meses	155€
Total coste Software				**409,28€**

Tabla 106. Costes de Software

8.2 Costes de Hardware

En este apartado se desglosa el hardware utilizado para este proyecto y su coste sobre el proyecto en función del tiempo utilizado.

Hardware	Precio	Utilización	Periodo desamortización	Coste aplicable
HP ENVY 15	999€	6 meses	36 meses	166,5€
ASUS Essentio	799€	6 meses	36 meses	133,16€

CG8250				
Impresora HP Deskjet 3059ª	53,72€	6 meses	36 meses	8,95€
Router BHS-RTA	65€	6 meses	36 meses	10,83€
Pen Drive SanDisk 32GB	13,50€	6 meses	36 meses	2,25€
Total coste Hardware				**321,69€**

Tabla 107. Costes de Hardware

8.3 Costes de personal

En este apartado se muestra el desglose de personal, considerando en el coste el gasto por IRFP y Seguridad Social. Para definir el coste individual se ha tenido en cuenta la experiencia que posee el alumno. Se incluye el desglose por tareas del alumno encargado de realizar el proyecto y el de los tutores en relación a las reuniones, la revisión de la documentación, la aceptación del proyecto, etc.

Fase	Coste/día	Total días	Total coste
Planificación y configuración	34€	10	340€
Análisis	44€	34	1.496€
Diseño e implementación	44€	76	3.344€
Pruebas	24€	15	360€
Documentación	37€	46	1.702€
Total			**7.242€**

Tabla 108. Coste individual

Integrantes	Coste/día	Total días	Total coste
Félix García Carballera	83€	5	415€
Juan Antonio Ortega Ruiz	83€	9	747€
Omar José Sanz Rodríguez			7.242€
Total coste de personal			**8.404€**

Tabla 109. Coste de personal

8.4 Costes de material fungible

En este apartado se desglosan los materiales que solo tienen uso para este proyecto y su coste.

Material	Coste
Papel	6,70€
Material de oficina (bolígrafos, tinta…)	50,49€
Otros gastos	30€
Total coste gasto fungible	**87,19€**

Tabla 110. Coste de material fungible

8.5 Presupuesto final

En este apartado se toman los costes totales de los puntos anteriores que forman el apartado *Presupuestos* y se calcula el presupuesto final teniendo en cuenta el riesgo, IVA y beneficios.

Presupuesto de costes totales	
Recursos Humanos	8.404€
Software	409,28€
Hardware	321,69€
Material fungible	87,19€
Subtotal	9.222,16€

Tabla 111. Costes totales

Riesgo		
Base imponible	Cuota riesgo	Coste riesgo
9.222,16€	20%	1.844,43€

Tabla 112. Coste riesgo

Beneficio		
Base imponible	Cuota beneficio	Total beneficio
11.066,59 €	10%	1.106,65€

Tabla 113. Beneficio

Impuesto de valor añadido		
Base imponible	Cuota beneficio	Impuesto
12.173,25€	21%	2.556,38€

Tabla 114. Impuesto

El coste total del proyecto asciende a: **14.729,63€ (I.V.A incluido)**

El precio total del proyecto es de 14.729,63€ (Catorce mil setecientos veintinueve euros con sesenta y tres céntimos). En dicho precio se incluyen los siguientes conceptos:

- Documentación relativa al proyecto.
- Código fuente.
- Derechos de propiedad intelectual.
- Derecho de explotación.
- Derecho de distribución a terceros.

En ningún caso se incluyen:

- Mantenimiento del simulador.
- Futuras actualizaciones o incremento de funcionalidades.

9. Conclusiones y trabajos futuros

En este apartado se valoran las conclusiones que se han obtenido durante el desarrollo de este proyecto. Además, se analizan las posibles características que podrían aplicarse al sistema en un futuro.

9.1 Conclusiones

Este proyecto para mí ha sido como un paso por meta que hace cuatro años cuando empecé la carrera me parecía inalcanzable, estoy muy orgulloso del trabajo realizado y el resultado que este ha mostrado.

El objetivo de crear un simulador de un nanoprocesador y que permitiera la ejecución de programas en ensamblador y su cosimulación con otro programa VHDL se ha alcanzado con éxito. Aunque en el desarrollo de este han aparecido dificultades como:

- Aprendizaje de nuevos lenguajes como es tcl.
- Uso de nuevas herramientas como ModelSim para ejecutar el simulador.
- Uso de Git para controlar las versiones, compartir y coordinar con el resto de grupo de trabajo el proyecto.

Sin embargo, ha merecido la pena ya que gracias a este proyecto he podido:

- Adquirir nuevos conocimientos en el ámbito de la programación.
- Demostrar parte de los conocimientos adquiridos durante el grado.
- Desarrollar un proyecto en una gran empresa como es Crisa junto a un excelente grupo de trabajo.

Además este proyecto puede ser aplicable como práctica docente debido a su similitud respecto instrucciones y funcionamiento con otros simuladores de procesadores conocidos y su fácil aprendizaje y manejo.

9.2 Trabajos futuros

Por último me gustaría resaltar en un futuro me gustaría finalizar el simulador permitiendo la posibilidad al usuario de acceder a una interfaz gráfica para el manejo de la simulación permitiendo las mismas operaciones que en el terminal, pero de una forma más gráfica y visible, mostrando en ella el nanoprocesador simulado y el recorrido que los datos hacen sobre él.

10. Glosario de términos y referencias

A continuación, se definen los acrónimos, tecnicismos y referencias que se han utilizado a lo largo de este proyecto.

10.1 Acrónimos

· FPGA: Field Programmable Gate Array.

· IF: Instruction Fetch.

· ID: Instruction Decode.

· EX: Instruction Exec.

· RAW: Read After Write.
· WAR: Write After Read.

· WAW: Write After Write.

· RT: Registro fuente y destino.

· RS: Registro fuente.

· CU: Casos de Uso.

· RC: Requisito de Capacidad.

· RR: Requisito de Restricción.

· PRS: Pruebas de Sistema.

· UC: Unidad de Control.

· PC: Controlador de programa.

· ALU: Unidad Aritmético-Lógica.

· NS: Nanosegundos.

· UC3M: Universidad Carlos III de Madrid.

· TFG: Trabajo Fin de Grado.

10.2 Definiciones

· Microelectrónica: Conjunto de reglas, normas, requisitos, materiales y procesos que aplicados en una secuencia determinada, permite obtener como producto final un circuito integrado, que son dispositivos electrónicos miniaturizados.

· Lenguaje ensamblador: Lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la

representación más directa del código máquina específico para cada arquitectura legible por un programador.

· Pipeline: Arquitectura que transforma un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior. Esta arquitectura es muy común en el desarrollo de programas para el intérprete de comandos, ya que se pueden concatenar comandos fácilmente con tuberías (pipe).

· ModelSim: Herramienta simuladora de código HDL.

· Header: Fichero de cabecera, contiene normalmente, una declaración directa de clases, subrutinas, variables, u otros identificadores.

· Clase: Abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener.

· cmd: Es el intérprete de comandos en OS/2 y sistemas basados en Windows NT (incluyendo Windows 2000, Windows XP, Windows Server 2003, Windows Vista , Windows 7 y Windows 8). Es el equivalente de command.com en MS-DOS y sistemas de la familia Windows 9x.

·Terminal:Es un programa informático donde interactúa el usuario con el sistema operativo mediante una ventana que espera ordenes escritas por el usuario desde el teclado.

10.3 Referencias bibliográficas

[1] David A. Patterson, John L. Hennessy. (2004). *Estructura y diseño de computadores VOL 1.Ed:* Reverte.

[2] David A. Patterson, John L. Hennessy. (2004). *Estructura y diseño de computadores VOL 2.Ed:* Reverte.

[3] David A. Patterson, John L. Hennessy. (2004). *Estructura y diseño de computadores VOL 3.Ed:* Reverte.

[4] Montero Montes, G. (2012). Desarrollo de un simulador de tráfico ferroviario sobre cartografía descargada dinámicamente. Trabajo de Fin de Grado. Madrid: Universidad Carlos III. <http://e-archivo.uc3m.es/bitstream/handle/10016/16782/PFG_Gabriel_Montero_Montes.pdf?sequence=1>

[5] Matías Hernández, D. (2006). Implementación VHDL del microprocesador MIPS64 Un enfoque docente. Proyecto Fin de Carrera.

[6] González de las Peñas Rodríguez, Rafael. (2011). Diseño del núcleo de un procesador MIPS. Trabajo de diploma para optar por el título de Ingeniero en Automática. La Habana: Instituto Superior Politécnico José Antonio Echeverría.

[7] Wikipedia. Procesador MIPS <https://es.wikipedia.org/wiki/MIPS_(procesador)> [Consulta: 18 de Febrero del 2015]

[8] Universidad Autónoma Metropolitana. Arquitectura RISC vs CISC. <http://www.azc.uam.mx/publicaciones/enlinea2/num1/1-2.htm> [Consulta: 19 de Febrero del 2015]

[9] Apache 2 Test Page powered by CentOS. Parte I. El simulador SPIM. <http://www2.dis.ulpgc.es/~ii-fc/web-practicas/parte1.html> [Consulta: 27 de Febrero del 2015]

[10]Universidad Nacional de La Plata. WINDLX. <http://electro.fisica.unlp.edu.ar/arq/downloads/Software/WinDLX/windlx.html> [Consulta: 2 de Marzo del 2015]

[11]MIPSIM. <http://www.mipsim.com/> [Consulta: 5 de Marzo del 2015]

[12]Programiz. C Programming Structure.<http://www.programiz.com/c-programming/c-structures> [Consulta: 19 de Mayo del 2015]

[13]CQuestions. Write C program to convert binary. <http://cquestionbank.blogspot.com.es/2010/06/write-c-program-to-convert-binary.html> [Consulta: 25 de Mayo del 2015]

[14]Crasseux. Mask. <http://crasseux.com/books/ctutorial/Masks.html> [Consulta: 28 de Mayo del 2015]

[15]Exploring Binary. Binary calculator. <http://www.exploringbinary.com/binary-calculator/> [Consulta: 9 de Junio del 2015]

[16]"The nacoos. Como usar Visio de Microsoft Office para crear Graficas, Diagramas u Organigramas". *Youtube.* <https://www.youtube.com/watch?v=izGrMIPWpoM> [Consulta: 3 de Agosto del 2015]

[17]SIWG. Tcl and SWIG as a C/C++ Development Tool. <http://www.swig.org/papers/Tcl98/TclChap.html> [Consulta: 5 de Julio del 2015]

[18]Página oficial Gobierno de España. Métrica v.3. <http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#.VgPHjsvtmko> [Consulta: 21 de Enero del 2015]

[19]CCM. Ciclo de vida Software. <http://es.ccm.net/contents/223-ciclo-de-vida-del-software> [Consulta: 21 de Enero del 2015]

[20]“ CityIngenieria MS project 2010 – 1 tutorial en español”. *Youtube.* <https://www.youtube.com/watch?v=ni1dE-78SPg> [Consulta: 23 de Enero del 2015]

Anexo A: Manual de usuario

1. Objetivo

En este anexo se detalla la explicación de cómo funciona el simulador desarrollado como trabajo fin de grado. El manual servirá para guiar al usuario dentro de la aplicación para que este vea que es un simulador con gran funcionalidad y fácil manejo.

2. Manual de referencia

A continuación se muestra una lista con toda la funcionalidad que presenta el simulador:

· El simulador permitirá al usuario cargar el programa ensamblador.

· El usuario podrá ejecutar el programa en el simulador eligiendo este si hacerlo ciclo a ciclo o de forma continua, pudiendo pararla cuando desee.

· El usuario podrá ver, recibir o escribir los datos contenidos en las estructuras del nanoprocesador tales como banco de registros, memoria de datos, memoria de instrucciones (solo ver) o registros de estado de la ALU(solo ver y recibir) cuando desee.

· El usuario podrá visualizar una lista de los comandos para facilitar el uso del simulador.

3. Manual de usuario

3.1 Primer contacto con el sistema

El primer contacto que tendrán los usuarios del simulador será el terminal de tclsh. Con esto, el usuario tendrá que cargar el paquete del nanosimulador escribiendo el siguiente comando "**load path_nanosim.so package"** a partir de este momento, podrá escribir el comando que desee realizar sobre el nanosimulador, este será el punto de partida.

```
[root@localhost pruebas]# tclsh
% load /root/Desktop/pruebas/np_sim/nanosim.so package
```

Ilustración 49. Carga de simulador sobre herramienta tclsh

3.2 Ayuda

A continuación se muestra como el usuario puede pedir ayuda al simulador para mostrar información sobre que comandos tiene, para ello se debe introducir el comando **"ns_help":**

```
Command LIST:

TOOLS
        - ns_help
        - ns_help_command [command]
        - ns_restart

LOAD
        - ns_load_rbank [file]
        - ns_load_dmem [file]
        - ns_load_imem [file]

READ
        - ns_read_reg [address]
        - ns_read_dmem [address]
        - ns_read_imem [address]
        - ns_read_Z
        - ns_read_C
        - ns_read_V
        - ns_read_N

WRITE
        - ns_write_reg [address] [data]
        - ns_write_dmem [address] [data]

DUMP
        - ns_dump_reg [address]
        - ns_dump_dmem [address]
        - ns_dump_imem [address]
        - ns_dump_pipe
        - ns_dump_Z
        - ns_dump_C
        - ns_dump_V
        - ns_dump_N

EXECUTION
        - ns_step
        - np_step
        - ns_run

0
```

Ilustración 50. Pantalla información lista de comandos simulador

En esta imagen se muestra como el comando lista todos los comandos que el simulador ofrece al usuario, si el usuario desea una mayor información de un comando en específico lo tendrá al escribir el siguiente comando "**ns_help_command 'nombre del comando'**":

```
% ns_help_command ns_load_rbank
ns_load_rbank(1)                        User Commands                        ns_load_rbank(1)

NAME
       ns_load_rbank - Load data to the register bank

SYNOPSIS
       ns_load_rbank [file.reg]

DESCRIPTION
       Read the file .reg and write in the register bank the data in the specific address.
0
```

Ilustración 51. Pantalla información de comando especifico

Como se puede ver en la imagen el comando muestra una breve descripción, funcionalidad y entradas y salidas.

3.3 Carga de programa ensamblador

A continuación se muestra como el usuario puede cargar el programa sobre el simulador, para ello, debemos cargar las siguientes estructuras con los datos del programa.

Mediante el comando "**ns_load_imem path_fichero.obj**" cargamos la memoria de instrucciones con las instrucciones del programa en ensamblador:

```
% ns_load_imem test.obj
0
```

Ilustración 52. Carga de fichero .obj

Mediante el comando "**ns_load_dmem path_fichero.var**" cargamos la memoria de datos con los valores del programa en ensamblador:

```
% ns_load_dmem test.var
0
```

Ilustración 53. Carga de fichero .var

Mediante el comando "**ns_load_reg path_fichero.reg**" cargamos el banco de registros con los valores del programa en ensamblador:

```
% ns_load_rbank test.reg
0
```

Ilustración 54. Carga de fichero .reg

En todas las instrucciones como se puede ver el simulador devuelve un 0, lo que significa que la carga sobre el simulador ha sido correcta, en caso contrario mostraría 1.

3.4 Ejecución del programa sobre el simulador

A continuación se muestra como el usuario puede ejecutar el programa en ensamblador sobre el simulador, en nuestro caso puede hacerse de dos maneras:

· Ejecutando la simulación ciclo a ciclo mediante el comando "**ns_step**" si queremos ver por pantalla la ejecución del pipeline.

```
% ns_step
t0 0x000  0x20000    IF lw    r%0, r%3
t0                    DE
t0                    EX
```

Ilustración 55. Pantalla ejecución de un ciclo

Y en caso de querer ejecutar y no ver la ejecución del pipeline "**np_step**".

· Ejecutando la simulación de forma continua mediante el comando "**ns_run**" si queremos ver por pantalla la ejecución del pipeline.

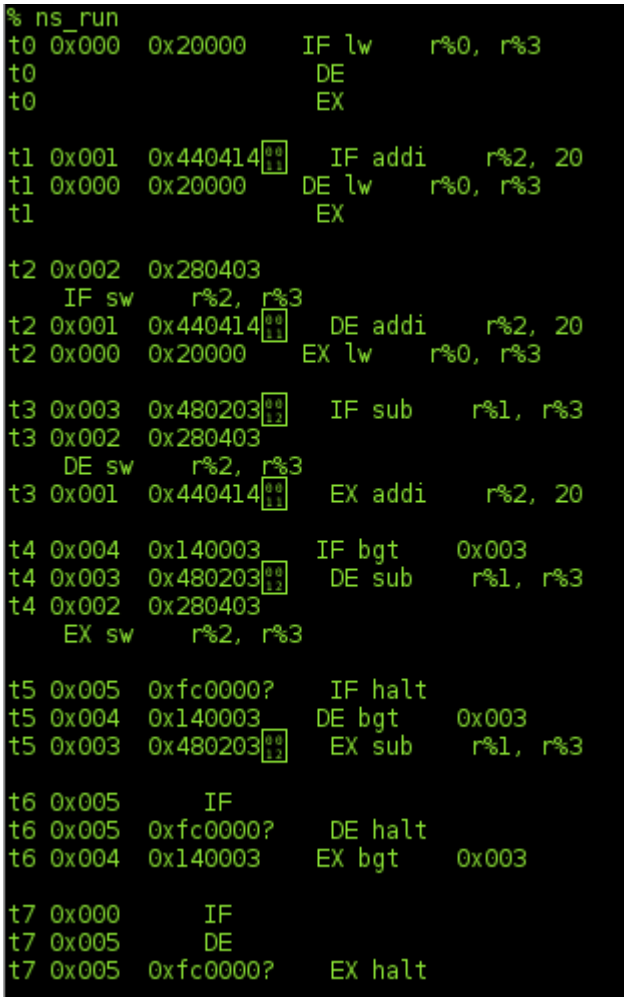


Ilustración 56. Pantalla ejecución continua

Si queremos parar la ejecución antes de finalizar el programa podemos utilizar la siguiente señal "**Ctrl+C**".

Por último, si queremos reiniciar el simulador dejando a 0 tanto las estructuras, como los ciclos y el pipeline el usuario debe escribir el comando "**ns_restart**"

Ilustración 57. Pantalla reiniciar simulador

3.5 Mostrar datos de las estructuras

A continuación se muestra como el usuario puede ver por pantalla los valores de las distintas estructuras que forman el nanoprocesador simulado.

Para poder ver un dato de una dirección de memoria de datos, el usuario debe escribir "**ns_dump_dmem 'dirección'**" dónde la dirección la puede escribir tanto en decimal como en hexadecimal.

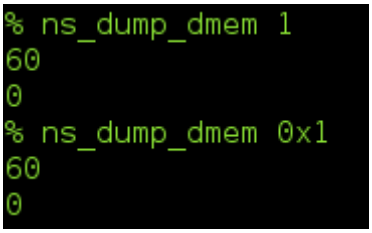


Ilustración 58. Pantalla mostrar dato de memoria de datos

Como se puede ver en las imágenes, el simulador muestra el dato en decimal que guardaba la dirección que hemos escrito como argumento de entrada.

Para poder ver un dato de un registro del banco de registros, el usuario debe escribir "**ns_dump_reg 'direccion'**" donde dirección es el número de registro y puede escribirse tanto en decimal como en hexadecimal.

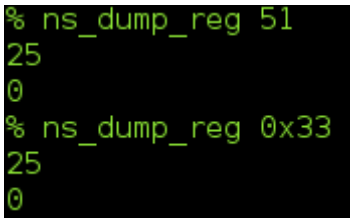


Ilustración 59. Pantalla mostrar dato de banco de registros

Como se puede ver en las imágenes, el simulador muestra el dato en decimal que guardaba la dirección que hemos escrito como argumento de entrada.

Para poder ver una instrucción de la memoria de instrucciones, el usuario debe escribir "ns_dump_imem 'dirección'" donde la dirección la puede escribir tanto en decimal como en hexadecimal.

Ilustración 60. Pantalla mostrar dato de memoria de instrucciones

Como se puede ver en las imágenes, el simulador muestra el dato en hexadecimal y la instrucción desensamblada que guardaba la dirección que hemos escrito como argumento de entrada.

Para poder ver el valor del registro de estado Zero de la ALU, el usuario debe introducir el comando "ns_dump_Z".

Ilustración 61. Pantalla mostrar valor de registro de estado Z

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de Z.

Para poder ver el valor del registro de estado Negative de la ALU, el usuario debe introducir el comando “ns_dump_N”.

Ilustración 62. Pantalla mostrar valor de registro de estado N

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de N.

Para poder ver el valor del registro de estado Overflow de la ALU, el usuario debe introducir el comando “ns_dump_V”.

Ilustración 63. Pantalla mostrar valor de registro de estado V

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de V.

3.6 Devolver datos de las estructuras

A continuación se muestra como el usuario puede recibir por pantalla los valores de las distintas estructuras que forman el nanoprocesador simulado.

Para poder recibir un dato de una dirección de memoria de datos, el usuario debe escribir “**ns_read_dmem ‘dirección’**” dónde la dirección la puede escribir tanto en decimal como en hexadecimal.

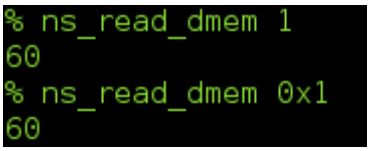


Ilustración 64. Pantalla recibir dato de memoria de datos

Como se puede ver en las imágenes, el simulador muestra el dato en decimal que guardaba la dirección que hemos escrito como argumento de entrada.

Para poder recibir un dato de un registro del banco de registros, el usuario debe escribir “**ns_read_reg ‘direccion’**” donde dirección es el número de registro y se puede escribir tanto en decimal como en hexadecimal.

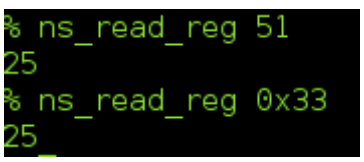


Ilustración 65. Pantalla recibir dato de banco de registros

Como se puede ver en las imágenes, el simulador muestra el dato en decimal que guardaba la dirección que hemos escrito como argumento de entrada.

Para poder recibir el valor del registro de estado Zero de la ALU, el usuario debe introducir el comando “ns_read_Z”.

Ilustración 66. Pantalla recibir dato de registro de estado Z

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de Z.

Para poder recibir el valor del registro de estado Negative de la ALU, el usuario debe introducir el comando “ns_read_N”.

Ilustración 67. Pantalla recibir dato de registro de estado N

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de N.

Para poder reacibir el valor del registro de estado Overflow de la ALU, el usuario debe introducir el comando “ns_dump_V”.

Ilustración 68. Pantalla recibir dato de registro de estado V

Como se puede ver el simulador muestra un bit cuyo valor es 0 o 1 en función del valor de V.

3.7 Modificar datos de las estructuras

A continuación se muestra como un usuario puede escribir sobre las estructuras de memoria de datos y banco de registros.

Para poder escribir sobre una dirección de la memoria de datos se debe escribir el comando “**ns_write_dmem** “dirección” “dato”” dónde la dirección puede ser en hexadecimal o decimal como antes y el dato como decimal o hexadecimal.

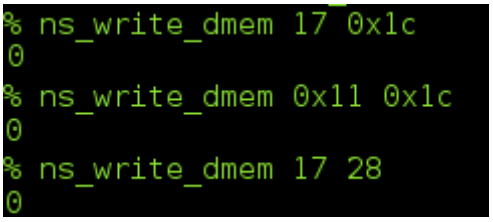


Ilustración 69. Pantalla escribir dato sobre memoria de datos

Como se puede ver en la imagen, el simulador muestra 0, lo que significa que la acción se ha realizado con éxito, en caso contrario, mostraría 1.

Para poder escribir sobre un registro del banco de registros se debe escribir el comando "**ns_write_reg** "dirección" "dato"" dónde la dirección puede ser en hexadecimal o decimal como antes y el dato como decimal o hexadecimal.

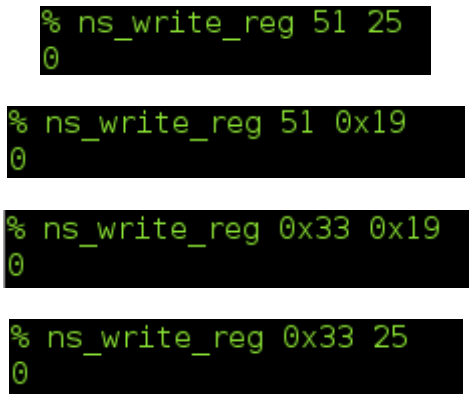


Ilustración 70. Pantalla escribir dato sobre banco de registros

Como se puede ver en la imagen, el simulador muestra 0, lo que significa que la acción se ha realizado con éxito, en caso contrario, mostraría 1.

Anexo B: Lista de operaciones nanoprocesador

	Instrucción	Código de operación						Número de registro destino									Dirección o número de registro									Instrucción completa
		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
		5	4	3	2	1	0	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0	
1	Nop	0	0	0	0	0	0																			nop
1	Br	0	0	0	0	0	1										RS									br RS
1	Bi	0	0	0	0	1	0										imm12									bi imm12
1	Be	0	0	0	0	1	1										imm12									be imm12
1	Bne	0	0	0	1	0	0										imm12									bne imm12
1	Bgt	0	0	0	1	0	1										imm12									bgt imm12
1	Bge	0	0	0	1	1	0										imm12									bge imm12
	Reserved	0	0	0	1	1	1																			

1	Lw	0	0	1	0	0	0	RT	RS	lw RT, (RS)
1	Lwi	0	0	1	0	0	1	RT	imm9	lwi RT, (imm9)
1	Sw	0	0	1	0	1	0	RT	RS	sw RT, (RS)
1	Swi	0	0	1	0	1	1	RT	imm9	swi RT, (imm9)
1	Mv	0	0	1	1	0	0	RT	RS	mv RT, RS
1	Mvi	0	0	1	1	0	1	RT	imm9	mvi RT, imm9
1	Mva	0	0	1	1	1	0	RT	imm12	mva RT, imm12
	Reserved	0	0	1	1	1	1			

1	Add	0	1	0	0	0	0	RT	RS	add RT, RS
1	Addi	0	1	0	0	0	1	RT	imm9	addi RT, imm9
1	Sub	0	1	0	0	1	0	RT	RS	sub RT, RS
1	Subi	0	1	0	0	1	1	RT	imm9	subi RT, imm9
1	Cmp	0	1	0	1	0	0	RT	RS	cmp RT, RS
1	Cmpi	0	1	0	1	0	1	RT	imm9	cmpi RT, imm9

	Reserved	0	1	0	1	1	x		
1	And	0	1	1	0	0	0	RT	RS	and RT, RS
1	Andi	0	1	1	0	0	1	RT	imm9	andi RT, imm9
1	Or	0	1	1	0	1	0	RT	RS	or RT, RS
1	Ori	0	1	1	0	1	1	RT	imm9	ori RT, imm9
1	Xor	0	1	1	1	0	0	RT	RS	xor RT, RS
1	Xori	0	1	1	1	0	1	RT	imm9	xori RT, imm9
	Reserved	0	1	1	1	1	x		

	Reserved	1	0	0	x	x	x		
1	Btest	1	0	1	0	0	0	RT	RS	btest RT, RS
1	Btesti	1	0	1	0	0	1	RT	imm4	btesti RT, imm4
1	Bset	1	0	1	0	1	0	RT	RS	bset RT, RS
1	Bseti	1	0	1	0	1	1	RT	imm4	bseti RT, imm4

1	Brst	1	0	1	1	0	0	RT	RS	brst RT, RS
1	Brsti	1	0	1	1	0	1	RT	imm4	brsti RT, imm4
	Reserved	1	0	1	1	1	x			
	Halt	1	1	1	1	1	1			halt

Tabla 115. Lista operaciones nanoprocesador

ANEXO C: Resumen Ingles

1. Introduction

This section provides an overview of the project performed and documented, this include motivations and objectives that led to the choice of this Final Career Project.

1.1 Motivation

The Final Career Project is proposed to be an intern in the company Crisa Airbus Defence and Space as a continuation of a project begun by Juan Antonio Ortega microelectronics head department. This project is to simulate a nanoprocessor in C programming language and import it into tcl. The phase developed by Juan Antonio Ortega is a C language program that transforms a file .asm assembly language files in several different formats to be used for the initial loading of nanoprocessor simulated. Files originated after the execution of the program are:

- .reg file format contains the data to be initially stored into the bank records nanoprocessor.
- .var file format contains the data to be initially stored into data memory.
- .obj file format contains the data to be stored in the instruction memory.

It is intended, therefore, in this project, to simulate the operation of a nanoprocessor in C language to execute a series of assembly instructions provided in Annex B and can be imported into tcl for later cosimulation with VHDL by ModelSim program so that allows loading a program in assembly from the three files mentioned above (reg, .var and .obj) on the simulator and this allows run the program and see the road that run through the data nanoprocessor.

This project sparked my interest because I could do a job that needed research, including program and allowed me the opportunity to participate in a project within a company as Crisa.

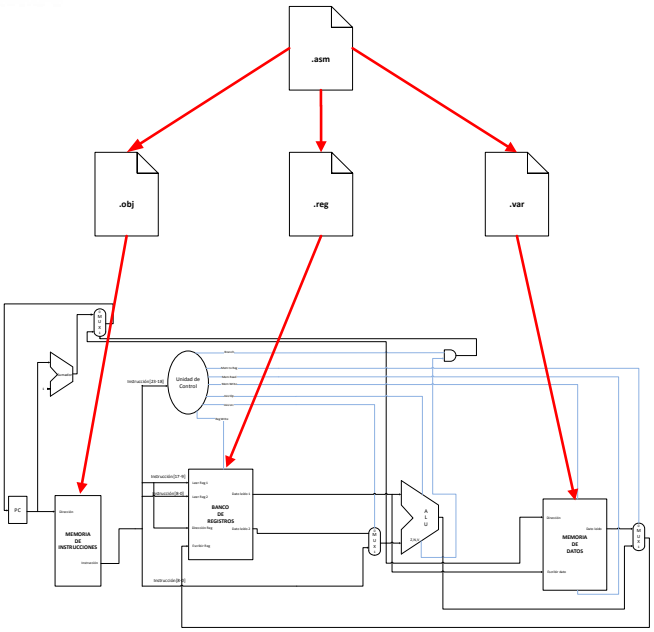


Ilustración 71. Esquema inicial proyecto nanosimulador

1.2 Objetives

After explaining the reasons with which the idea and the project started, this section is carried out the description of the main and specific objectives that the simulator must comply.

1.2.1 Main objetives

The main objective of the project is develop a tool that permit simulate the actions of a nanoprocessor using the TCL languaje which permit cosimulation with an existing programa in VHDL languaje on the ModelSim tool, in order to check the right functioning of the VHDL program which will implemented on an FPGA.

1.2.2 Especific objetives

Apart from the main objective, emerges a series of specific objectives in order to allow compliance with the main objective. The specific objectives to be developed in the project are:

- Allow the loading of the .reg, .var and .obj files on nanoprocessor structures.
- Allow the assembly program run per cycles or continuous.

• You can choose view the pipeline during execution of the program on nanoprocessor.

• You can view the structures that make up the nanoprocessor.

• Modify nanoprocessor structures at any time.

1.3 Document structure

Here it proceeds to list the sections of the Final Career Project and a brief description of the contents of these:

· Section 1: Introduction. Brief introduction to the Final Career Project, its objectives are explained, the structure of the document and the socio-economic environment.

Section 2: State of the art. In this section it refers to the current state of technology.

- State of the art processors. The study MIPS possible basis for the design of our processor architecture is analyzed.

- State of the art simulators. Referring the current state of the processor simulators.

· Section 3: Initial Design sequential nanoprocessor. In this section shown the first idea of implementing nanoprocessor with its main components.

Section 4: Analysis System. In this section is defined the system use cases and requirements that must be met, check whether it is feasible and study the possible ways forward.

· Section 5: Proposed design and implementation of nanoprocessor. This section is defined properly and clear the system architecture and its implementation including the description of the classes that form.

· Section 6: Project management. This section provides an estimate of the time required for the project.

· Section 7: Verification Tests. In this section, the proper functioning of the implemented system is checked, performing its duties properly.

· Section 8: Budget. This section describes in detail the cost of the project.

· Section 9: Conclusions and future work. After completion of the project, a series of conclusions and future lines to consider are developed.

· Section 10: Glossary of terms and references.

1.4 Socio-Economic Environment

Today the use of simulators is very common because they are manipulated models of a real or theoretical system, allowing its use to train and learn the system reacts according to the actions taken on it. In a simulation, although is usually a simplification of the real world, they can solve problems and situations, learning procedures, come to understand the different characteristics of the phenomena, learn how to control

and actions to apply in particular situations. With these systems, the user can experience until exhaust their possibility of change the model in question.

As mentioned in paragraph 1.2 Objectives, the project to develop arises from the need for the department microelectronics company Crisa Airbus Defence and Space to create a simulated nanoprocessor with tcl language for cosimulation with an existing VHDL program thereafter it will be implemented in the FPGA.

FPGA (Field Programmable Gate Array) technology is a device containing blocks of programmable logic and therefore can play from simple functions such as a logic gate or a combinational logic to complex systems on a chip, that is, you can move the design made on paper to the physical device to test it. Therefore, today is a technology used most often by its many utilities.

To program this device, the designer with the help of development environments specialized in designing systems to be implemented in an FPGA. A design can be caught either as schematic, or by using a special programming language. These special programming languages are known as HDL and one of the most used is the VHDL.

2. Project summary

2.1 State of the art

This section describes the investigation done previous to the design and implementation of the simulator.

This section is divided into two parts, the investigation done on the processor and the investigation on the simulators on assembler programs.

To investigate processors for this project I based on the MIPS technology because it is the most used for teaching at the university. MIPS processors are RISC technology, RISC is a processor made up of a reduced instruction set and not allowed access to the same component in the same cycle in two different instructions and differs from the CISC technology to have this more complex instructions and not allow parallelism between instructions.

For the investigation of the simulators i have been installed: SPIM, WinDLX and MIPSim to see its operations and to extract the best features and to implement them in our simulator.

- The SPIM simulator is an independent simulator that simulates programs as if it were a MIPS32 processor, it reads and executes programs written assembly language for this processor and offers load an assembly program on the simulator and since this action you can make a series of operations as run, step, print data records and print data memory.
- The WinDLX simulator is a simulator of the pipeline processor DLX is essentially a revised MIPS processing and allows programs written in assembler DLX and shows all relevant information such as the status of the CPU pipeline, bank records, I / O, memory and statistics.
- The MIPSim simulator is a simulator that shows the path of data across a pipeline divided into five stages.

2.2 System analysis

This section describes the system use cases and requirements that must be met before starting with the design of nanoprocessor:

The use cases are:

The user should be able to:

- Load an assembler program on the simulator.
- Simulate the program loaded in cycles or continuously.
- Stop running or restart simulation.
- Show or hide the simulation.
- View / receive / write data on memory.

- View / receive / write about the bank records.
- See the instruction memory.
- View status registers of the ALU.
- View information operations simulator.

The system must be able to:

- Simulate the nanoprocessor architecture.
- Driving simulation data path by nanoprocessor.

Having described the use cases, the main requirements to be met are:

- The program should be able to compile both Windows and Linux.
- The simulator must operate from the tclsh and ModelSim tools.
- The simulator must allow the loading of the .reg ,.obj and .var files.
- The simulator should be able to simulate the path of the data assembler program through nanoprocessor.
- The user must be able to stop an execution.
- The user should be able to restart an execution.
- The simulator must allow show, reading and writing nanoprocessor structures such as bank records and data memory.
- The simulator should allow show the data in the memory instruction and status registers of the ALU.
- The user must have a command that allows the user showing help or existing commands in the simulator or information from a user-specified command.

2.3 Design simulator

This section shows the steps followed for the design of nanoprocessor. As it commented on the state of the art we MIPS processor architecture and modify according to our needs and meeting the requirements desired by the company. Being the following components:

· Data memory

· Report instruction

· Bank records

· Control unit

· ALU

· Program Counter

· Multiplexers

We can see the initial sequential design in section *Diseño inicial: nanoprocesador secuencial* in "Ilustración 11. Nanoprocesador secuencial".

Our initial design is a sequential nanoprocessor, but we soon realize that it is possible to improve performance by segmenting the nanoprocessor, however, the following risks and solutions appear:

- Structural Risks: This type of risk occurs when more than one instruction tries to access the same resource simultaneously. Solutions: Data memory and instruction memory separate to avoid conflicts in access and simultaneous reading and writing records bank instructions.
- Risk Data: This type of risk occurs when an instruction requires a data generated by the previous instruction in the execution stage. It can be classified into three categories:

  - RAW (Read After Write): Occurs when the second instruction reads a data before the first generates it.

  - WAR (Write After Read): Occurs when the second instruction writes before the first read it.

  - WAW (Write After Write): Occurs when the second instruction writes operand before being written by the former.

  Solution: Create an additional way of hardware, use a forwarding unit, element that sends the result in a statement to the instructions as they need to operate.

- Control Risks: This type of risk occurs with jump instructions. It occurs when an instruction that changes the value of the PC have not already done when you have to start execution of the next instruction. Solution: For this design i opt to resolve the risk predicting that the jump is performed and if taken to add a nop instruction to the instruction fetch take the right direction after the execution of the jump.

2.4 Implementation

This section shows how developed the simulator nanoprocessor explained above. To do this, we proceed to show the class diagram and explain the purpose of the same.

The image of the class diagram is in the paragraph *Propuesta de Diseño e implementación* in "ilustración Diagrama de clases", we can see the relationship between the classes and headers.

- np-ldata.c: Read .var hexadecimal file which contains the data memory used by the program with initial values and writes the data in the data memory. Allows to show on the screen the contents of a memory address, reading the contents

of a memory address and write data directly into memory data sending as parameters an address and data.

- np-ldata.h: Create the data memory.
- np-lins.c: Read hexadecimal .obj file containing program instructions and stored in the instruction memory. Allows to show on the screen the contents of a memory address instruction.
- np-lins.h: Create the structure of instructions and instruction memory.
- np-lreg.c: Read hexadecimal .reg file containing the registers used by the program with initial values and writes the data in the register bank. Allows to show on the screen the contents of a register from its address, read the contents of a register from its address and write directly to the register bank as parameters by sending an address and data.
- np-lreg.h: Create the structure of the registers and register bank.
- • np-pipeline.c: Simulates path of the data on nanoprocessor segmented. Lets you run the cycle to cycle either continuously or simulation showing on screen the pipeline per cycle if the user desires.
- np-pipeline.h: Create segmentation registers.
- np-tools.c: Contains help instructions simulator and restart it.
- np-alu.c: Simulates the behavior of the Arithmetic Logic Unit.

Here one example of simulation performed, being the assembly program:

```
        .header
r1:     .reg %r1 100
r2:     .reg %r2 5

        .text
main:
        add r1, r2
        addi r2, 50
        and r1,r2
        halt
        .end
```

This program performs the following actions:

· Initially stored in the bank registers the following data:

- At r1 register stores the data 100.

- In the register R2 stores the data 5.

· Execute operations:

- Addition between registers r1 and r2.

- Addition between register r2 and the immediate value 50.

- Performs logical AND operation between r1 and r2 .

- Stop the execution.

After the execution of "alu_test.asm" program with the np-asm program created by Juan Antonio explained in the *Introducción* the following files are generated:

- Alu_test.obj contains program instructions in hexadecimal to put in the memory instruction.
  0x000 0x400202
  0x001 0x440432
  0x002 0xfc0000
- Alu_test.reg contains hexadecimal data to be introduced in the bank.
  0x001 0x0064
  0x002 0x0005
- Alu_test.var contains hexadecimal data to enter in the data memory.
  Empty, it do not have references to the data memory.

Once created the file, proceeds to execution of the program in the simulator and testing nanoprocessor, see "Ilustración 35. Simulación de programa" alu_test.asm "".

As can be seen in the image:

- The load files on the simulator is correct to show on screen the number 0.
- The execution of the simulation is correct, shows the instruction in each stage per cycle.
- The operations were successful, to check registers shown on screen target of operations:
- The value r1 register is 105, as a result of operation add r1, r2 where r1 is the value register 100 and the register r2 has the value 5.
- The register r2 has a value of 55, as a result of the operation addi r2, 50 where the register r2 has the value 5 and adds the immediate value 50.

2.5 Testing

The tests are:

- Compile nanoprocessor simulator in Windows.
- Compile nanoprocessor simulator in Linux.
- Load Simulator package in the tclsh tool.
- Load with the simulator the file .var
- Load with the simulator the file .reg
- Load with the simulator the file .obj
- Show to screen the value of a data memory address nanoprocessor entering the address in hexadecimal.
- Show to screen the value of a data memory address nanoprocessor entering the address in decimal.

- Show to screen the value of an instruction memory address nanoprocessor entering the address in hexadecimal.
- Show to screen the value of an instruction memory address nanoprocessor entering the address in decimal.
- Show to screen the value of a register bank number nanoprocessor entering the address in hexadecimal.
- Show to screen the value of a register bank number nanoprocessor entering the address in decimal.
- Receive the value of a data memory address nanoprocessor entering the address.
- Receive registry value of a register bank nanoprocessor entering the number of register.
- Write a value in a data memory address nanoprocessor entering the address and number.
- Write a value in a register bank nanoprocessor entering the register number and a number.
- Run an assembler program on the simulator and that the simulation is displayed on screen.
- Stop the execution of an assembler program on the simulator.
- Run an assembler program on the simulator cycle to cycle and is displayed on the screen and restart the simulator after execution.
- Show to screen the value of the status registers of the ALU nanoprocessor simulated.
- Show to screen the list of commands and the simulator contains information from one of them.

All tests described above were passed **successfully** allowing the system implementation in the company. We can see the fulfillment of the requirements with the traceability matrix “Tabla 104. Matriz de trazabilidad entre requisitos-pruebas.”.

2.6 Planning

This section describes the planning, monitoring and control of activities and human resources involved in the project. As a result of this control is possible to know at all times where problems are occurring and solve or palliate them immediately.

The methodology used is the metric version 3 is a methodology for planning, development and maintenance of information systems, promoted by the Ministry of Finance and Public Administration of the Government of Spain for the systematization of activities lifecycle of software projects scope of government. The main processes of this metric are:

- Planning information systems.

- Development of information systems, is divided into five processes:
  - System Viability study
  - Analysis of information system
  - Design of the information system
  - Construction of the information system
- Maintenance Information System.

To order stages that make up the project, we used a model of cascade fed back. This model is designed to wait for a stage finishes to start the next and after finishing the stage to perform a final review, which is responsible for determining if the project is ready or not to advance to the next phase, however to be fed back can return to previous stages for review and modify them if necessary. This life cycle consists of the following stages:

- Analysis of requirements
- Design
- Coding
- Evidence
- Integration

Project planning is shown with a Gantt diagram, this diagram shows the activities, milestones and tasks performed during the project is estimated. Then the general diagram with tasks that are considered necessary for successful completion of the project is described:

This project begins on January 19 and ends on September 18, 2015 with 242-day of which 181 days are working. The project consists of 7 phases.

- Planning (8 days).
- Configuration management (13 days).
- System Viability study (24 days).
- Analysis of the information system (23 days).
- Design of the information system (83 days).
- Tests (22 days).
- Implementation and acceptance of the system (8 days).

The Gantt chart can be seen in “Ilustracion 40 – Diagrama de Gantt general”.

2.7 Budget

In this section the calculation of the costs of the project and the final budget thereof is shown. To do this we estimate the following costs:

- Software Costs: This section lists the software tools used for this project and its cost in function of time used. We can see the software project cost in “Tabla 105. Costes de Software”.
- Hardware Costs: This section lists the hardware used for this project and its cost in function of time used. We can see the hardware project cost in “Tabla 107. Costes de Hardware”.
- Staff costs: This section provides the list of staff costs, considering the cost of the IRFP and Social Security. To define the individual cost the experience of the student have been taken into account. It includes a breakdown by student work responsible for carrying out the project and tutors in relation to meetings, review of documentation, the project acceptance, etc. We can see the result in “Tabla 108. Coste individual” and “Tabla 109. Coste personal”.
- Costs of consumable items: This section list the items are only used for this project and its cost. We can see the result in “Tabla 110. Coste de material fungible”.
- Final budget: In this section I take the total costs of the above sections forming the budget and the final budget is calculated taking into account the risk, IVA and benefits. We can see the calculation of the final budget in “Tabla 111. Costes totales”, “Tabla 112. Coste riesgo”, “Tabla 113. Beneficio” and “Table 114. Impuesto”.

Finally the total cost of the project is: **14.729,63€ (I.V.A incluided)**

The price included:

- Documentation relating to the project.

- Source code.

- Copyright.

- The right of exploitation.

- Distribution rights to third parties.

In no event the price include:

- Maintenance of the simulator.

- Future updates or increased functionality.

3. Conclusions and future work

In this section the conclusions that have been obtained during the development of this project are valued. In addition, the possible features that could be applied to the system in the future are discussed.

3.1 Conclusions

The goal of creating a simulator of a nanoprocessor and to allow the implementation of programs in Assembler and cosimulation with another program VHDL has been successfully achieved. Although this development there have been difficulties such as:

- Learning new languages such as TCL.

- Use of new tools such as ModelSim to run the simulator.

- Using Git for version control, share and coordinate with the rest of the project working group.

However, it was worth it because thanks to this project I have:

- Acquire new knowledge in the field of programming.

- Demonstrate part of the knowledge acquired during the degree.

- Develop a project in a great company like Crisa with an excellent workgroup.

Furthermore, this project can be applied as teaching practice because of its similarity regarding operating and instructions with other simulators of know processors and its easy to learn and use.

3.2 Future work

Finally I would like to emphasize in the future I would like to end the simulator allowing the user the possibility to access a graphical interface for managing simulation allowing the same operations in the terminal, but in a more graphical and visibly showing in it simulated the nanoprocessor and travel data.