**ORGANIZACIÓN DE COMPUTADORES**

**PROYECTO FINAL:**

**PROCESADOR SACADA**

**PRESENTADO POR:**

**Santiago Burgos Martínez**

 (santiagoburgos@javeriana.edu.co)

**Dayanna Méndez Zamudio**

 (mendez.dayanna@javeriana.edu.co)

**Andrea Camila Moya Buitrago**

 (moya.andrea@javeriana.edu.co)

**PRESENTADO A:**

Francisco Fernando Viveros Moreno

02 de Diciembre de 2020

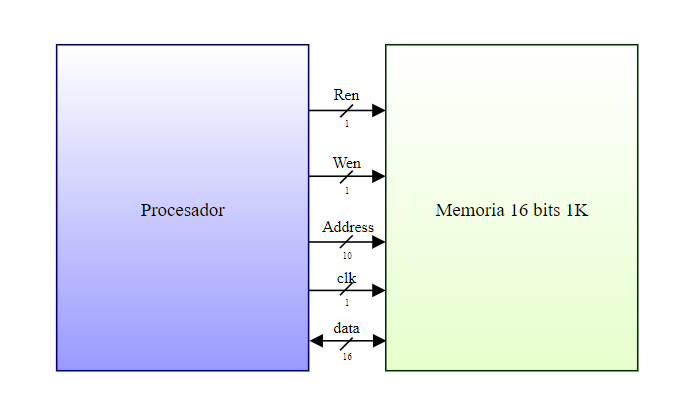


**INTRODUCCIÓN**

El presente documento tiene como objetivo principal plasmar el proceso de concepción, diseño e implementación de un procesador multipropósito capaz de realizar operaciones aritméticas básicas de suma y resta entre dos números de entre 1 y 16 bits leídos de una memoria RWM de 1024 posiciones, además tiene capacidad de interrumpir el programa debido a una señal externa dada por el usuario, y la posibilidad de moverse entre diferentes posiciones de la memoria para así ejecutar programas de mayor complejidad.

**DESCRIPCIÓN DE LA PROPUESTA**

En este proyecto se desea diseñar e implementar un procesador de 16 bits que ejecute operaciones entre datos leídos desde una memoria y almacene los resultados de la operación en la misma. Esta memoria debe ser una memoria de lectura y escritura RWM de 1024 posiciones por lo que, para seleccionar una posición de memoria se debe usar una dirección de 10bits. El diagrama de conexión entre el procesador y la memoria se puede apreciar en la figura 1.



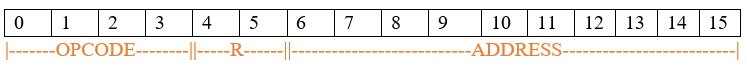
*Figura 1. Diagrama de entradas y salidas entre el procesador y la memoria.*

Las señales que se tienen en este diagrama son:

1. **Ren**: Como la memoria es de lectura-escritura es necesario una señal que indique si se desea leer el contenido de la memoria o escribir en la memoria, por lo que cuando esta señal está en 1, significa que el procesador leerá el contenido de la memoria.
2. **Wen**: Como ya se mencionó, para leer la memoria la señal Ren debe estar en 1, mientras que cuando se desea escribir datos en la memoria RWM la señal Wen debe estar en 1.
3. **Address**: Esta señal de 10 bits corresponde a la dirección de la memoria en la que se desea almacenar un dato o la posición en la que está el dato que el procesador requiere leer.
4. **Data**: La información almacenada en la posición establecida por la señal Address es la correspondiente a la señal data, cuyo tamaño es de 16 bits. El dato puede ser tanto enviado como recibido por el procesador, por esto es una señal bidireccional.
5. **Clk**: Finalmente, el procesador le da la señal de reloj clk a la memoria para que los datos sean leídos o escritos cada flanco de subida del clk.

**ORGANIZACIÓN Y DISTRIBUCIÓN DE LOS 16 del procesador**

Teniendo en cuenta que, el máximo de memoria a usar son 1KB, y que en sistema binario es , el número de bits para cada dirección es de 10 bits. Ahora, partiendo del tamaño de instrucciones se procede a diferenciar, caracterizar y seccionar que función o propósito tendrá cada una. El formato de instrucción se puede apreciar en la figura 2.



*Figura 2. Formato de instrucción.*

La descripción del formato de instrucciones se encuentra a continuación:

* 1. OPCODE: Será la parte de la instrucción (4 bits) que tiene la tarea de hacer saber e informar al procesador cual es la función/tarea que se quiere realizar con la información o el resto de los bits que hay en la instrucción.
  2. M: Estos dos bits están encargados de informar al procesador qué modo de direccionamiento tiene la posición de memoria que se está leyendo. Entre los modos de direccionamiento se tienen: Direccionamiento Inmediato, Directo e Indirecto.
  3. ADDRESS / DATA: Los 10 bits de Address corresponden a la dirección específica de la posición de memoria de la cual se obtienen los datos a operar y en donde se almacenan los resultados de las operaciones.
* Modo de direccionamiento

|  |  |  |  |
| --- | --- | --- | --- |
| Modos de Direccionamiento | | | |
|  | **Modo** | **Hexadecimal** | **Binario** |
| 0 | Inmediato | 1 | 01 |
| 1 | Directo | 2 | 10 |
| 2 | Indirecto | 3 | 11 |

**Tabla 1. Modos de direccionamiento.**

* **Instrucciones**

Con lo planteado anteriormente se pueden establecer algunas instrucciones que ejecutará el procesador, estas se pueden apreciar en la tabla 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instrucciones | | | | |
|  | **Comando** | **Hexadecimal** | **Binario** | **Modo** |
| 0 |  | 0 | 0000 |  |
| 1 | SUMA | 1 | 0001 | '01' ^ '10' ^ '11' |
| 2 | RESTA | 2 | 0010 | '01' ^ '10' ^ '11' |
| 3 | MOVER IN | 3 | 0011 | '01' ^ '10' ^ '11' |
| 4 | MOVER OUT | 4 | 0100 | '10' ^ '11' |
| 5 | COMPARACION | 5 | 0101 | '01' ^ '10' ^ '11' |
| 6 | SALTO > | 6 | 0110 | '01' ^ '10' ^ '11' |
| 7 | SALTO < | 7 | 0111 | '01' ^ '10' ^ '11' |
| 8 | SALTO SIN CONDICION | 8 | 1000 | '01' ^ '10' ^ '11' |
| 9 | SALTO SUBRUTINA | 9 | 1001 | '01' ^ '10' ^ '11' |
| 10 | RETORNO SUBRUTINA | A | 1010 | '01' ^ '10' ^ '11' |
| 11 |  | B | 1011 | - |
| 12 |  | C | 1100 | - |
| 13 | - | D | 1101 | - |
| 14 | - | E | 1110 | - |
| 15 | - | F | 1111 | - |

**Tabla 2. Instrucciones y su equivalente en binario y hexadecimal.**

* **SUMA:** Esta instrucción se encarga de realizar la operación aritmética de Suma entre dos números. Se tiene como condición previa que el primer número ya se encuentre en el registro dentro del procesador, mientras que esta instrucción lleva el segundo número de la operación. El segundo número puede ser llevado de tres formas, la primera forma corresponde a la forma de direccionamiento directa en donde en el espacio de Address se encuentra la posición de memoria donde se encuentra el otro valor. La segunda forma/modo de direccionamiento inmediato, es que el valor del segundo número se encuentre en el espacio de Address. Por otro lado, en el tercer modo de direccionamiento, el modo indirecto, en el espacio de Address se encuentra almacenada una dirección de memoria en la que se encuentra la dirección del número a Sumar.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 1 | x | X | x | x | x | x | x | x | x | x |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **RESTA:** Esta instrucción se encarga de realizar la operación aritmética de resta entre dos números. Se tiene como condición previa que el primer número ya se encuentre en el registro dentro del procesador (siendo este primer número el sustraendo), mientras que esta instrucción lleva el segundo número de la operación. Este segundo número puede ser llevado de tres formas, la primera forma corresponde a la forma de direccionamiento directa en donde en el espacio de Address se encuentra la posición de memoria donde se encuentra el otro valor. La segunda forma/modo de direccionamiento inmediato, es que el valor del segundo número se encuentre en el espacio de Address. Por otro lado, en el tercer modo de direccionamiento, el modo indirecto, en el espacio de Address se encuentra almacenada una dirección de memoria en la que se encuentra la dirección del número a restar.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 1 | x | X | x | x | x | x | x | x | x | x |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **MOVER IN:** Esta instrucción se encarga de llevar un dato/dirección de un espacio en la memoria a un registro en el procesador. Esta instrucción se puede realizar con los tres modos de direccionamiento anteriormente mencionados.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 1 | x | X | x | x | x | x | x | x | x | x |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **MOVER OUT**: Esta instrucción se encarga de llevar el contenido que se encuentra en el registro del procesador a un espacio/posición de memoria escogido. Esta instrucción se puede realizar con los tres modos de direccionamiento anteriormente mencionados.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | 0 | x | X | x | x | x | x | x | x | x | x |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **COMPARACIÓN:** Esta instrucción se encarga de realizar la comparación entre dos números. En la operación se compara el número que se encuentra en el procesador y el número que está guardado ya sea en la posición de memoria especificada en la instrucción, el número que se encuentra en la instrucción, o el número que se encuentra en la dirección dada por la dirección de la instrucción, según los modos de direccionamiento establecidos.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 1 | 0 | x | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **SALTO >=:** Esta instrucción se encarga de realizar un salto a otra posición de memoria dependiendo del resultado de la comparación, en este caso si es mayor o igual realizara el salto. (Modo Directo).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | x | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **SALTO <=:** Esta instrucción se encarga de realizar un salto a otra posición de memoria dependiendo del resultado de la comparación, en este caso si es menor o igual realizara el salto. (Modo Directo)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | 0 | X | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **SALTO SIN** **CONDICIÓN:** Esta instrucción permite los saltos a cualquier posición de memoria sin ninguna condición o comparación previa. (Modo Directo)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 1 | 0 | x | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **SALTO SUBRUTINA:** Esta instrucción se encarga de realizar un salto a una posición de memoria en donde se encuentra el inicio de una subrutina, teniendo en cuenta todos los cambios y modificaciones que tiene que hacer el programa para guardar la posición en donde debe continuar después de acabar la subrutina, además de inicializar el “Program Counter” en la posición deseada. En cuanto los modos de direccionamiento esta instrucción puede ser de direccionamiento directo e indirecto**.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 1 | 0 | x | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

* **RETORNO SUBRUTINA:** Esta instrucción se encarga de realizar los pasos necesarios para que él se siga ejecutando el programa desde la siguiente posición en donde se dejó luego de hacer el salto a subrutina.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 0 | x | X | x | x | x | x | x | x | x | X |

|-------OPCODE--------||-----M-----||---------------------------ADDRESS--------------------------|

**Funciones**

Las funciones que se tienen planteadas para que el procesador realice, ejecute o lleve a cabo son variadas y con el avance del proyecto estas pueden llegar a aumentar y/o cambiar, teniendo en cuenta lo siguiente.

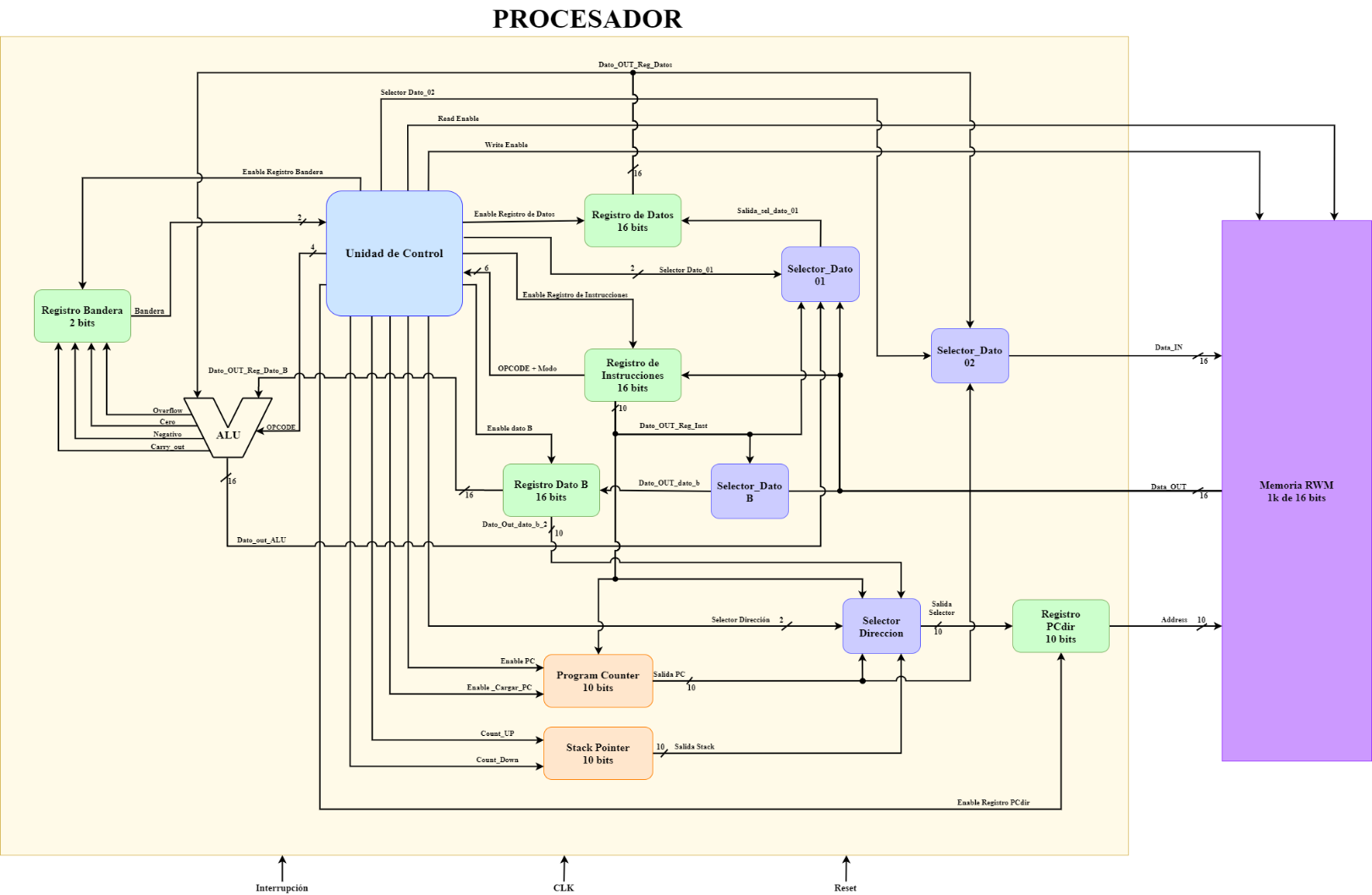
La primera función que se tiene es la de realizar operaciones aritméticas como la suma la resta y la multiplicación de los datos e información que el programa necesite realizar. Estas operaciones estarán condensadas o agrupadas dentro del procesador en una sección llamada ALU (unidad aritmética y lógica) la cual se encargará de realizar el procesamiento requerido y necesario por el programa.

La segunda función es el manejo de múltiples registros dentro del procesador para que de esta forma varias porciones de información y de instrucciones puedan ser procesadas, manipuladas y ejecutadas, creando así una forma más cómoda para el trabajo con los datos.

**DIAGRAMA DE BLOQUES**

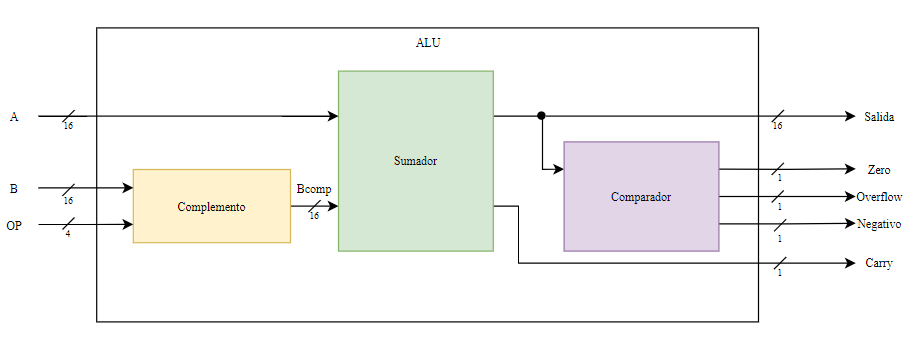
Descripción:

* **Unidad de Control:** Se encarga de interpretar, habilitar y deshabilitar, los distintos bloques del procesador para así llevar a cabo una variedad de tareas.
* **Registro de Datos:** Este registro de 16 bits se encarga de almacenar temporalmente los datos o valores que serán analizados por el procesador. También para cuando se estén realizando operaciones aritméticas este registro guardara el Dato A que ingresara a la ALU, posteriormente este mismo registro guardara el resultado de la ALU.
* **Registro Bandera:** Este registro de 2 bits se encarga de almacenar las banderas que puedan surgir en la ALU (Zero, Overflow, Carry, Comparación).
* **Registro de instrucciones:** Este registro de 16 bits es en donde llegan las instrucciones que están en una posición memoria, las cuales serán decodificadas e interpretadas para saber qué acción el programa guardado en la memoria quiere que el procesador realice.
* **Registro Dato B:** Este registro de 16 bits es usado para guardar el dato B o segundo dato que ingresaría a la ALU, además se usa para guardar posiciones de memoria cuando el programa envía una instrucción con direccionamiento indirecto.
* **Registro PCdir:** Este registro de 10 bits es en donde se guardará la información sobre la posición de memoria que se quiere leer.
* **Program Counter:** Este contador de 10 bits se encarga de recorrer el programa guardado en la memoria de forma ordenada y cíclica, avanzando de posición a medida que se procesa el valor de la posición de memoria previa. También tiene la función de recorrer subrutinas, es decir, puede adquirir una nueva posición de memoria (proveniente del “Registro de Instrucciones”) y comenzar a recorrer desde ese punto.
* **Stack Pointer:** Este contador como su nombre lo dice, de apuntar a una posición de memoria para así cuando se ingrese a una subrutina(s) se guarde la posición a la cual hay que regresar luego de acabar la subrutina(s).
* **Selector Dirección:** Este bloque se encarga de enviar la dirección de memoria correcta. Seleccionar entre la dirección proveniente del “Program Counter”, la dirección proveniente del “Registro de Instrucciones”, la dirección proveniente del “Stack Pointer2. La señal de selección del dato vendrá del bloque “Unidad de Control”.
* **Selector Dato 01:** Como en el modo de direccionamiento inmediato los datos se encuentran almacenados directamente en los 10 bits llamados “Address”, para evitar la lectura de la memoria dos veces sin necesidad se puede utilizar un selector llamado “Selector Dato” cuyas entradas son un habilitador llamado “Selector Dato” proveniente del control, el dato de salida de la memoria RWM , 10 bits de la salida del registro de instrucciones y por ultimo 16bits provenientes de la ALU ( unidad aritmética y lógica), cuando “Selector Dato” es 01, el dato que se almacena en el registro de datos es el dato del registro de instrucciones, en el caso de que sea 10, se almacena el dato de la memoria y por último en el caso de que sea 11, se almacena el dato proveniente de la ALU.
* **Selector Dato 02:** Este selector se encarga de escoger entre el valor guardado en el “Registro\_dato” o el “Program Counter”, el cual será guardado en una posición de memoria especificada por el “Registro PCdir”.
* **Selector Dato B:** Este selector se usa para escoger entre las salidas del registro de instrucciones (10 bits) o la salida de la memoria (16 bits), para luego enviar el dato correcto al Registro Dato B.



*Figura 3. Diagrama de bloques.*

Como se puede apreciar en la figura 3, el procesador cuenta con una unidad lógico-aritmética que permite realizar las operaciones aritméticas entre los números leídos de la memoria, en la figura 4 se tiene el diagrama de bloques de la ALU.

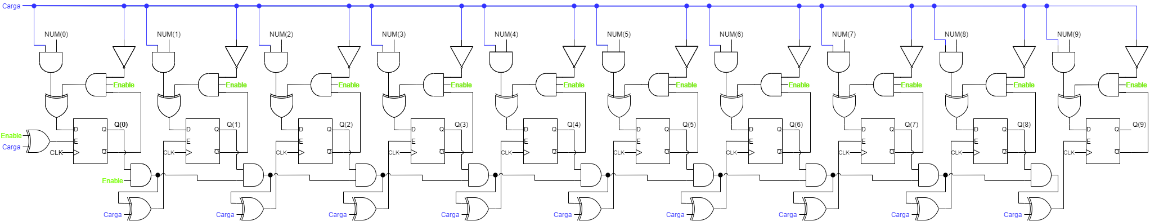


*Figura 4. Diagrama de bloques interno bloque ALU.*

**COMPUERTAS LOGICAS – Bloques procesasdor**

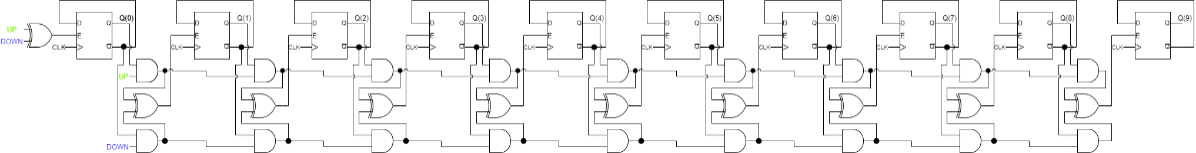


Para que el procesador ejecute subrutinas y retome la posición en la que se encontraba, se tiene el Stack Pointer y el Program Counter, el Stack Pointer apunta a una dirección de la memoria en la que no se encuentra el programa principal de modo que almacene la dirección de la memoria actual, el Program Counter es un contador que aumenta cada vez que se ejecuta una instrucción para ejecutar la siguiente. El Program Counter es un contador que debe estar en la capacidad de inicializarse en un valor específico de modo que cuenta con una señal de Carga proveniente del control y una señal de entrada llamada “NUM” que es el número con el que se inicializa el contador. En la figura 5 se puede apreciar el circuito lógico del bloque “Program Counter”.



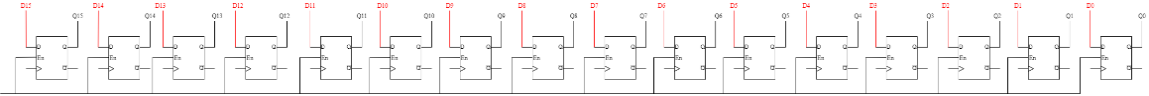
*Figura 5. Program Counter.*

Con respecto al “Stack Pointer” este es un bloque contador ascendente-descendente que posee dos habilitadores, una señal llamada “Cont\_up” que habilita el conteo ascendente y una señal “Cont\_down” que habilita el conteo descendente. En la figura 6 se puede apreciar el circuito lógico del Stack Pointer. Para este procesador, el espacio reservado de la memoria para la pila está en las últimas posiciones, de modo que el Stack Pointer inicialmente apunta a la posición 1023 y su valor va decrementando cada vez que se ejecuta la subrutina, mientras que incrementa su valor cada vez que se retorna de una subrutina.



*Figura 6. Stack Pointer.*

Como se puede ver en el diagrama de bloques, el procesador cuenta con 3 registros de 16bits, de modo que el circuito lógico de los registros “Registro\_Instrucciones”, “Registro\_Datos” y “Registro\_Dato\_B” que almacenan las instrucciones leídas de la memoria, el primer y el segundo dato a operar en la ALU respectivamente, de modo que su circuito lógico se puede apreciar en la figura 7. Las señales de entrada de este registro son el habilitador, el reloj, el reset y el dato de 16 bits, mientras que su salida es también un dato de 16 bits.



*Figura7. Registro de 16 bits.*

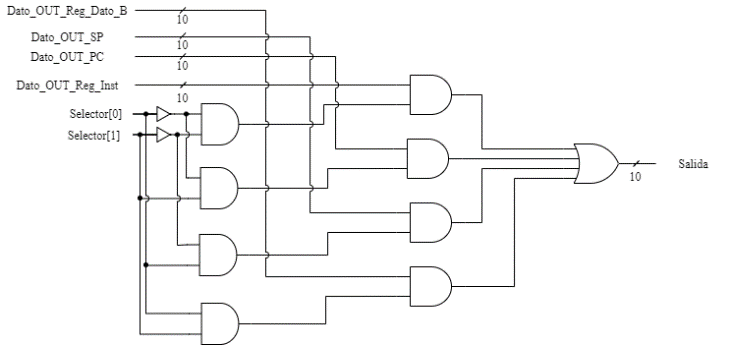
El registro de 10 bits “PC\_dir” es el registro que almacena la dirección de la memoria en la que se desea leer o escribir un dato, en la figura 8 se puede apreciar el circuito lógico interno de este registro. Como se puede ver en el diagrama de bloques, este registro posee un dato de entrada de 10 bits proveniente del bloque “Selector\_dirección”, y su salida se conecta directamente a la entrada de dirección de la memoria RWM.

Imagen que contiene objeto, reloj

Descripción generada automáticamente

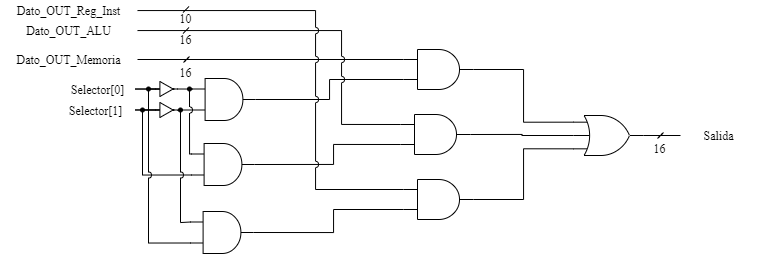
*Figura 8. Registro “PC\_dir” de 10 bits.*

Como se puede obtener la dirección de la memoria a partir de la instrucción cuando es un modo de direccionamiento directo y se puede obtener a partir del Stack pointer si se ejecuta una subrutina, es necesario un selector que cuente con 4 entradas provenientes del registro de datos “Registro\_Dato\_B”, el Stack Pointer, el program Counter y el registro de instrucciones "Registro\_Instruccciones”. La señal de selección proveniente del control es una señal de 2 bits de modo que para que el registro “PC\_dir” tome el valor del registro de instrucciones debe ser “00”, para que el registro tome el valor del Stack Pointer la señal debe ser “01”, para que el registro tome el valor del Program Counter debe ser “10” y para que tome el valor del registro de datos B debe ser “11”. En la figura 9 se puede apreciar la lógica combinacional para el selector de dirección.



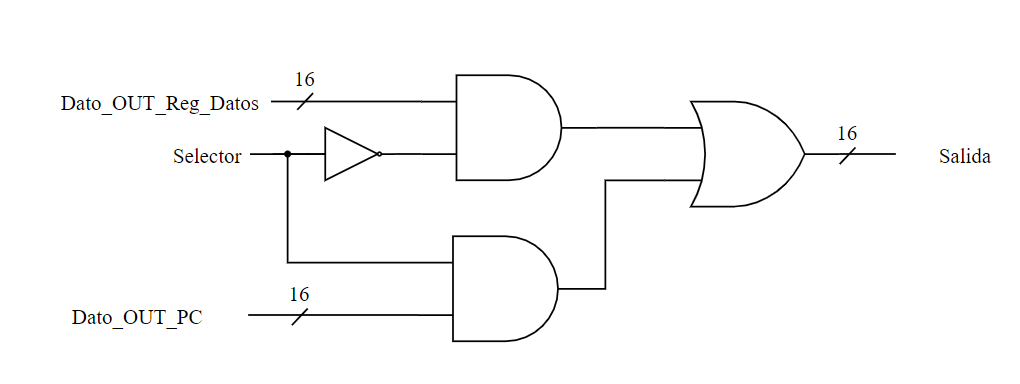
*Figura 9. Selector de dirección.*

Por otro lado, para seleccionar que dato se guardará en el registro de datos se tiene el selector llamado “Selector\_dato\_01” que cuenta con tres entradas. La primera entrada corresponde a la salida del registro de instrucciones, que se debe guardar en el registro de datos cuando el direccionamiento es inmediato, de modo que en este caso la entrada del registro es de 10 bits, la segunda entrada corresponde al resultado de la operación realizada por la ALU, la tercera entrada corresponde al dato leído de la memoria que es a su vez una entrada de la ALU. En la figura 10 se puede apreciar el circuito lógico de dicho selector. Como se puede ver, la salida únicamente tomará un valor según la señal de selección, cuando la señal de selección es “00” la salida de este selector es el dato de salida de la memoria, cuando la señal de selección es “01” la salida del selector son los 10 bits menos significativos del registro de instrucciones y cuando la señal de selección es “10” la salida del selector es el resultado de la operación realizada por la ALU.



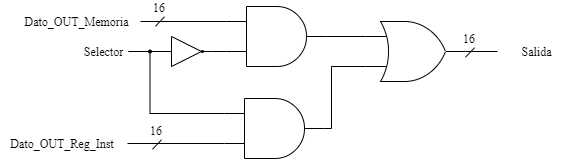
*Figura 10. “Selector\_dato\_01”.*

Como en la memoria se deben guardar tanto los datos del resultado de las operaciones realizadas por la ALU como el Program Counter cuando se ejecuta una subrutina, se tiene un selector con una señal de selección de 1 bit y dos entradas provenientes del registro de datos y el Program Counter. Cuando la señal de selección es 0, la salida del selector es el dato de salida del registro de datos, mientras que cuando la señal de selección es 1 la salida del selector es el Program Counter. En la figura 11 se tiene el circuito lógico del selector “Selector\_dato\_02”.



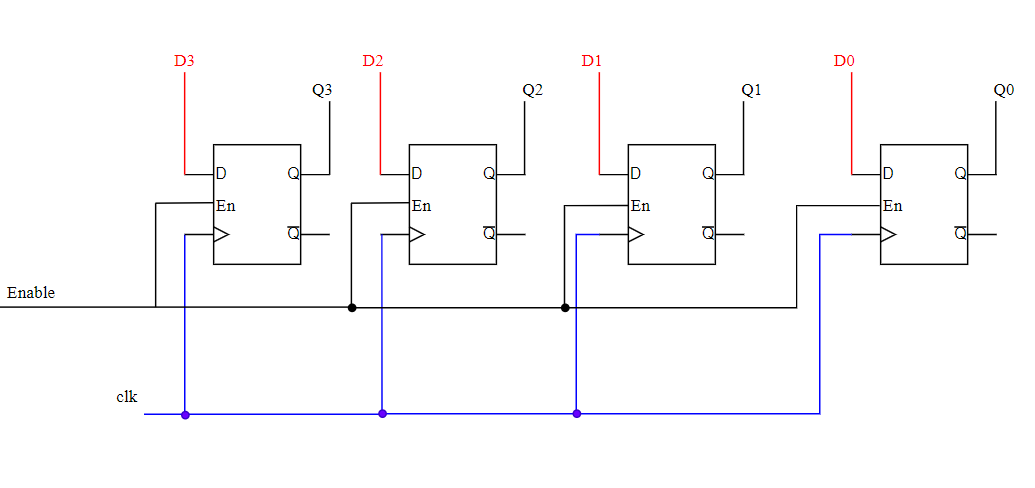
*Figura 11. “Selector\_dato\_02”.*

Como en el registro “Registro\_dato\_B” se desea guardar un dato leído de la memoria y un dato proveniente del registro de instrucciones, se tiene el selector de la figura 12. Al igual que el selector de la figura anterior, este posee una señal de selección de 1 bit, cuando esta señal es 0 la salida del selector es el dato leído de la memoria, cuando la señal de selección es 1, la salida del selector es el dato del registro de instrucciones.



*Figura 12. “Selector\_dato\_B”.*

Finalmente, en la figura 13 se tiene el circuito combinacional del registro de banderas, como se puede ver, este registro es de 4 bits de modo que cada bit guarda una bandera entre las banderas de carry, de overflow, de negativo y de zero. Al igual que en los registros anteriores, este circuito posee una entrada de habilitación y una de reloj. Cuando los flip flop se encuentran habilitados se almacenan las banderas de modo que el bit 0 del registro guarda el valor de la bandera de overflow, el bit 1 guarda el resultado la bandera que indica que ambos números son iguales, el bit 2 guarda la bandera “Negativo” que permite identificar si el resultado es un valor de signo negativo y finalmente en el bit 3 se almacena el valor del carry de salida de la operación ejecutada por la ALU.



*Figura 13. Registro “Registro\_Bandera”de 2 bits.*

**DESCRIPCIÓN ALU**

Se propone el diagrama de bloques en el anexo A, en el que se tienen tres bloques principales llamados Complemento, Sumador y Comparador. El primer bloque, “Complemento” tiene como función realizar el complemento a dos del número B que corresponde al sustraendo en la resta, este bloque tiene como entradas el número B y el código OP, ya que, este permite identificar si es necesario o no realizar el complemento a dos del número, en el caso de la suma, el número que entra al sumador debe ser el mismo que entra a la ALU, mientras que en la resta el número debe ser negativo. La salida de este bloque, llamada en la Figura 2 “Bcomp” corresponde a la segunda entrada del componente “Sumador”, que tiene la función de sumar el número “A” con el número “B” o el número “Bcomp”, este sumador tiene como salidas el número “Salida” de 16 bits y el “Carry” de la suma de 1 bit. Para comparar los números y evaluar si son iguales o si B es mayor que A se propone el bloque “Comparador” cuya entrada es la salida del sumador de 16 bits “Salida” y sus salidas son las banderas “Zero”, “Overflow”, “Carry” y “Negativo”.

**BLOQUE SUMADOR**

Para realizar la operación de suma entre dos números se hace uso de la lógica combinatoria denominada **FULL ADDER,** propuesta en la Figura 14. Este bloque de Full Adder está compuesto por 3 entradas - Bit del sumando A, Bit del sumando B y el Carry\_in - y 2 salidas – Bit de resultado y el Carry\_out -.

**Diagrama, Esquemático

Descripción generada automáticamente**

*Figura 14. Sumador*

Ahora, teniendo presente las entradas y salidas este bloque se puede configurar en cascada para así formar un sumador de la cantidad de bits que sean necesarios, en la Figura 15 se muestra la implementación de un sumador de 4 bits.

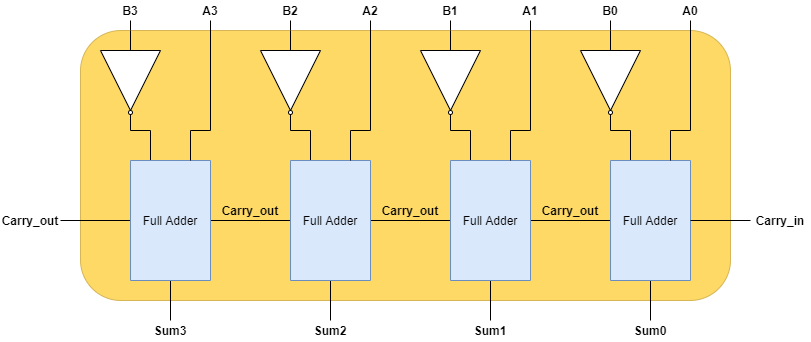
**Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente**

*Figura 15. Sumador de 4bits*

**COMPLEMENTO A 2**

Este bloque se usa para realizar la operación de resta sin tener la necesidad de usar el BLOQUE RESTADOR, este bloque hace uso del BLOQUE SUMADOR, haciendo que la cantidad de compuertas lógicas se reduzca.

****

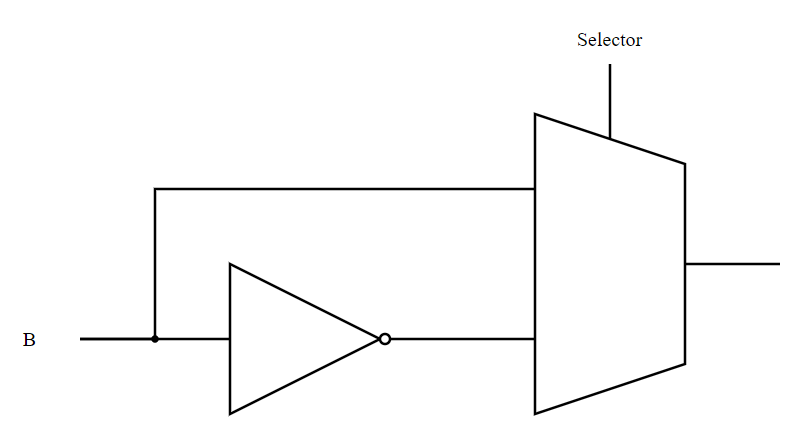
*Figura 16. Complemento a 2 de 4 bits.*

El bloque de complemento a 2 consiste en negar/Invertir el valor del número al que se le va a cambiar de signo, para el caso de la Figura#, este número hace referencia a los bits B. Luego de negar el número se procede a sumarle el valor de 1.

A continuación, se presentan las compuertas lógicas internas de cada uno de los bloques internos de la ALU.

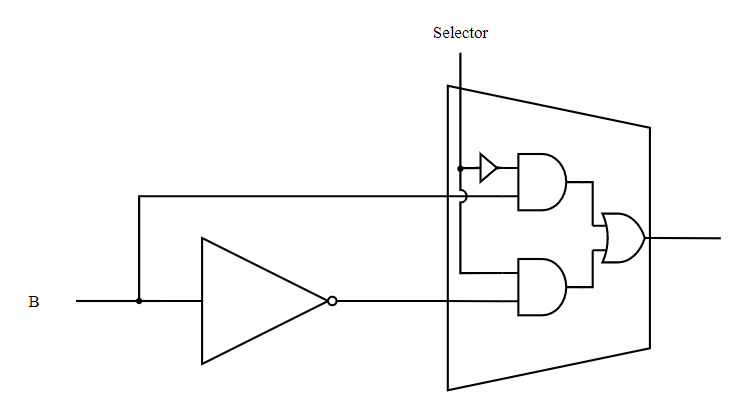
1. **Complemento**

Para realizar el complemento a dos de un número, este se debe negar y al número negado se le debe sumar 1, obteniendo su representación negativa, sin embargo, como no se debe realizar el complemento para todas las ecuaciones, se puede utilizar un selector que escoja si tomar el número B o el número B negado, como se puede ver en la Figura 17.



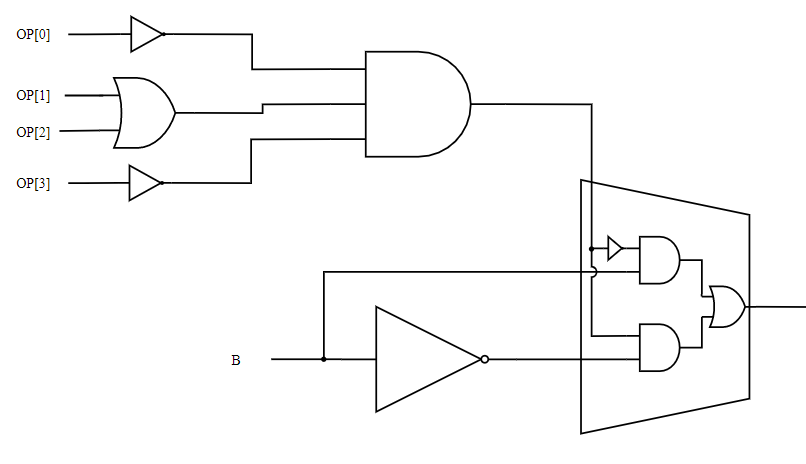
*Figura 17. Selector del bloque complemento.*

Como se desea que cuando el selector sea ‘1’se tome B negado, y en el caso contrario, B, internamente en el selector se tienen dos compuertas AND y una compuerta OR al final para el resultado, como se puede ver en la Figura 18. Cuando el selector es ‘1’, la AND entre B y el selector es 0, mientras que cuando el selector es ‘0’, la AND entre B y el selector da como resultado el valor de B.



*Figura 18. Circuito interno del selector del bloque Complemento.*

Con respecto a la señal de selección, para que únicamente tome B negado si la operación es de resta o de suma se tiene el circuito combinacional de la Figura 19, cuando el OP es distinto de “0110” o “0001”, el resultado de la compuerta AND es ‘0’ seleccionando B en lugar de B negado.



*Figura 19. Circuito para el selector del bloque Complemento.*

Por último, para realizar la suma de 1 bit se introducirá un ‘1’ en el Carry-in del bloque sumador para así ahorrar compuertas y no tener dos bloques sumadores sino solo uno.

En el anexo A se encuentra el diagrama en bloques completo de la ALU.

**AHPL**

1. Inicio
2. Registro\_PCdir ß Program Counter

à(Interrupt=1)/37, /2

1. Memoria ß Read Address (Registro\_PCdir)
2. Memoria ß Read Address (Registro\_PCdir)
3. Registro\_Instrucciones ß(Dato\_Out\_Memoria)
4. Control ß [0…5] Registro\_Instrucciones (Opcode\*Modo)

à (Modo=01 ˆ Opcode =0011) / 6 , ((Modo=10 + Modo=11) ^ (Opcode =0011 +Opcode=0100)) /7

, (Modo=01 ˆ (Opcode =0001 or Opcode =0010 or Opcode=0101) / 19,

, ((Opcode =0001 or Opcode =0010 or Opcode =0101) ^ (Modo=10 or Modo11))/20,

,(( Opcode =1001 ) ^ (Modo=10 or Modo=11)) /15,

, (Opcode =1010)/36,

(Opcode =1000 or Opcode =0110 or Opcode =0111)/42

MOVER IN ˆ MOVER OUT

1. Registro\_Dato ß [6…15] Registro\_Instrucciones

à /15

1. Registro\_PCdir ß [6…15] Registro\_Instrucciones



à (OPCODE="0100" and Modo="10")/17, /8

1. Memoria ß Read Address (Registro\_PCdir)
2. Memoria ß Read Address (Registro\_PCdir)

à (Opcode="0100")/16, ((Opcode ="1001" and Modo="11") or (Opcode ="1010") or Int\_end='1')/34, (Opcode ="0011" or ((Modo="11") and((Opcode ="1000") or (Opcode ="0110" and (Mayor='1')) or (Opcode ="0111" and (Menor='1')))))/10

1. Registro\_Dato\_B or Registro\_datosß Memoria

à (((Opcode ="0011") and (Modo="11")) or ((Modo="11") and((Opcode ="1000") or (Opcode ="0110" and (Mayor='1')) or (Opcode ="0111" and (Menor='1')))))/11, ((Modo="10" and Opcode ="0011"))/15, ((Modo="10" and Opcode ="0100"))/15

1. Registro\_PCdirßRegistro\_Dato\_B

à (Opcode ="0100")/17, /12

1. Memoriaß Registro\_Pcdir
2. Memoriaß Registro\_Pcdir

à (((Modo="11") and((Opcode ="1000") or (Opcode ="0110" and (Mayor='1')) or (Opcode ="0111" and (Menor='1')))))/ 44, /14

1. Registro\_DatosßMemoria
2. PCßIncrementa (PC)

à (Opcode ="1001")/30,/1

1. Registro\_Dato\_BßRead (Dato\_Memoria)

à/11

1. MemoriaßW\_enable (Registro\_PCdir)
2. MemoriaßW\_enable (Registro\_PCdir)

SUMA Y RESTA

1. Registro\_Dato\_Bß [6….15] ALU\_DatoB

à/28

1. Registro\_PCdirßRegistro\_Dato\_B

à ((Opcode ="0010" or Opcode ="0001") and (Modo="10"))/23, (Modo="11")/21

1. Memoria ß Read Address (Registro\_PCdir)
2. Memoria ß Read Address (Registro\_PCdir)
3. Registro\_Dato\_Bß[6…15] ALU\_DatoA

à (Modo="10")/28, /24

1. Registro\_PCdirßRegistro\_Dato\_B
2. MemoriaßREN
3. MemoriaßREN
4. Registro\_Dato\_BßALU\_DatoA
5. Enable\_Registro\_banderasßBanderas\_ALU

à (Opcode ="0001" or Opcode ="0010")/29, (Opcode ="0101")/15

1. Registro\_DatosßDato\_OUT\_ALU

à/15

SUBRUTINAS

1. Registro\_PCdirß Dato\_OUTSP

à( Opcode ="1010" or Int\_end='1')/8, /31

1. MemoriaßW\_enable (Registro\_PCdir)
2. MemoriaßW\_enable (Registro\_PCdir)
3. (SPßCount\_down ^ Registro\_PCdirßDato\_OUT\_Registro\_Instrucciones)

à (Modo="10")/35, (Modo="11")/8

1. Registro\_Instrucciones ß Dato\_OUT\_Memoria

à (Int\_end='1')/41, ((Opcode ="1001" and Modo="11") or (Opcode ="1010"))/35

1. Program\_Counterß[6…15] Registro\_Instrucciones

à ((Modo="11") and((Opcode ="1000") or (Opcode ="0110" and (Mayor='1')) or (Opcode ="0111" and (Menor='1'))))/43, /1

1. Stack\_Pointer ß Incremento (Stack\_Pointer)

à/30

INTERRUPCIONES

1. Registro\_PCdir ß Stack\_Pointer

à (Int\_end='1')/8, /38

1. MemoriaßW\_enable (Registro\_PCdir)
2. MemoriaßW\_enable (Registro\_PCdir)
3. Interrupcion

à (Interrupt='1')/40, (Int\_end<='1’)/37

1. Registro\_PCßRegistro\_Instrucciones

à/1, Int\_end<='0'

SALTOS (>, <, SIN CONDICION)

1. Interpretación resultados registro bandera

à ((Modo="10" or Modo="11") and((OPCODE="1000") or (OPCODE="0110" and (Mayor='1')) or (OPCODE="0111" and (Menor='1'))))/35

1. Registro\_PCdirßProgram\_Counter

à/8

1. Program\_CounterßRegistro\_Intrucciones

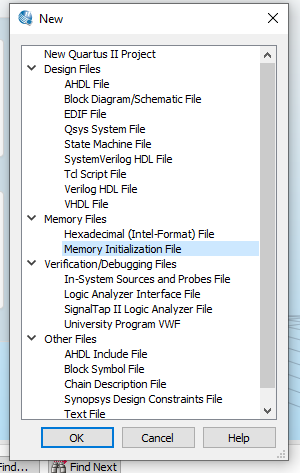
à/1

**PROTOCOLO DE PRUEBAS**

Para comprobar el funcionamiento del Procesador y de cada bloque se debe realizar el protocolo de pruebas para cada componente del procesador que se presenta a continuación.

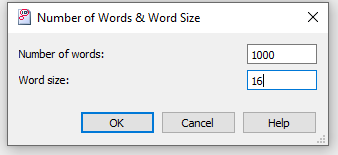
En primer lugar, se debe inicializar la memoria, para esto, se debe realizar el siguiente proceso.

1. Crear un nuevo archivo de memoria de tipo “Memory Initialization File”, como se muestra en la Figura 20.



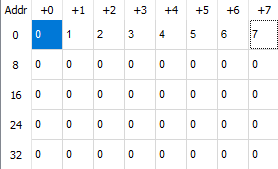
*Figura 20. Creación del archivo para inicializar la memoria.*

1. Establecer el tamaño de la memoria y el tamaño de los datos a guardar.



*Figura 21. Tamaño de la memoria y de los datos.*

1. Llenar el archivo con los datos.



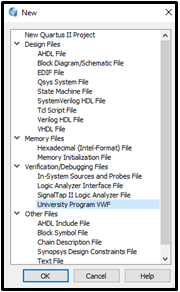
*Figura 22. Archivo en donde se guardan los datos.*

Para esta memoria, en el archivo de tipo mif se guardará el programa a ejecutar por el procesador.

**Bloques procesador**

El siguiente proceso aplica para cada uno de los bloques dentro del procesador.

Crear un archivo de tipo University Program VWF, que es un archivo de simulación.



*Figura 23. Archivo de tipo VWF.*

Figura X Insertar las señales de entrada y salida del bloque Complemento.

Después de haber creado el archivo VWF, se procede a escribir toda la lógica combinatoria de cada bloque respectivamente.

**PROTOCOLO DE SIMULACIÓN**

A continuación, se presenta el programa escrito en memoria en el cual se prueban varias instrucciones del procesador. Los valores tanto de las instrucciones como de los Opcode y los modos estarán escritos en decimal, pero el programa realizara la conversión de decimal a binario.







Fig 24. Proyecto procesador

En la siguiente figura se muestra el programa escrito en la memoria para comprobar la instrucción de Leer de la memoria. En este programa se tiene en cuenta los modos de direccionamiento que tiene la instrucción de leer y se usan posiciones al azar dentro de la memoria para probar la lectura de datos en posiciones de memoria diferentes.







Fig 25. Proyecto procesador leer dato de la memoria

En la siguiente figura se muestra el programa escrito en la memoria para comprobar la instrucción de Escribir en la memoria, guardando en la memoria los datos con distintos modos de direccionamiento.





Fig 26. Proyecto procesador escribir dato de la memoria

En la siguiente figura se muestra el programa a cargar en la memoria para probar el funcionamiento de la instrucción de suma y sus modos de direccionamiento







Fig 27. Proyecto procesador sumador

En la siguiente figura se muestra el programa escrito en la memoria para comprobar la instrucción de Resta junto a sus tres modos de direccionamiento







Fig 28. Proyecto procesador restador

En la siguiente figura se muestra el programa escrito en la memoria para comprobar la instrucción de comparación, realizando primero una operación de resta para así variar el dato a comparar.





Fig 29. Proyecto procesador instrucción comparación

En la siguiente figura se muestra el programa escrito en la memoria para comprobar las instrucciones de salto (Salto >, Salto <, Sato sin condición, Salto subrutina), probando también sus modos de direccionamiento.









Fig 30. Proyecto procesador instrucciones de salto.

Por último, en la Figura 31, se muestra el cambio de estado de la señal de interrupción que llega al procesador para así probar la respuesta que tiene le procesador frente a esta señal.

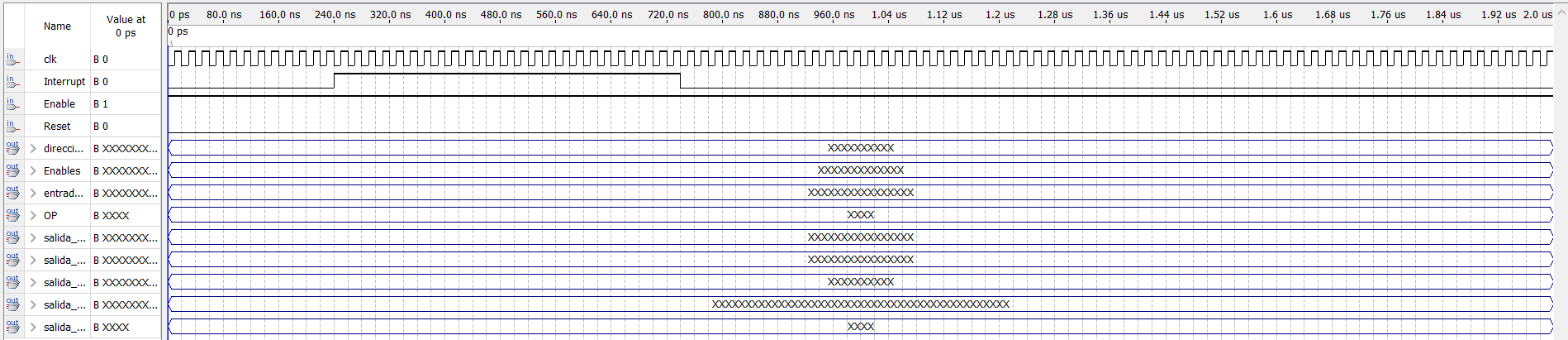


Fig 31. Comprobación interrupción.

**ANÁLISIS DE RESULTADOS - SIMULACIÓN**

En la Figura 32, se presenta los resultados de la simulación realizada comprobando el funcionamiento correcto del procesador cuando este lee un programa de la memoria RWM, realizando la lectura y ejecución de las distintas instrucciones que se crearon, se observa también que el procesador realiza y ejecuta las instrucciones de acuerdo con lo planteado en control y en los estados.

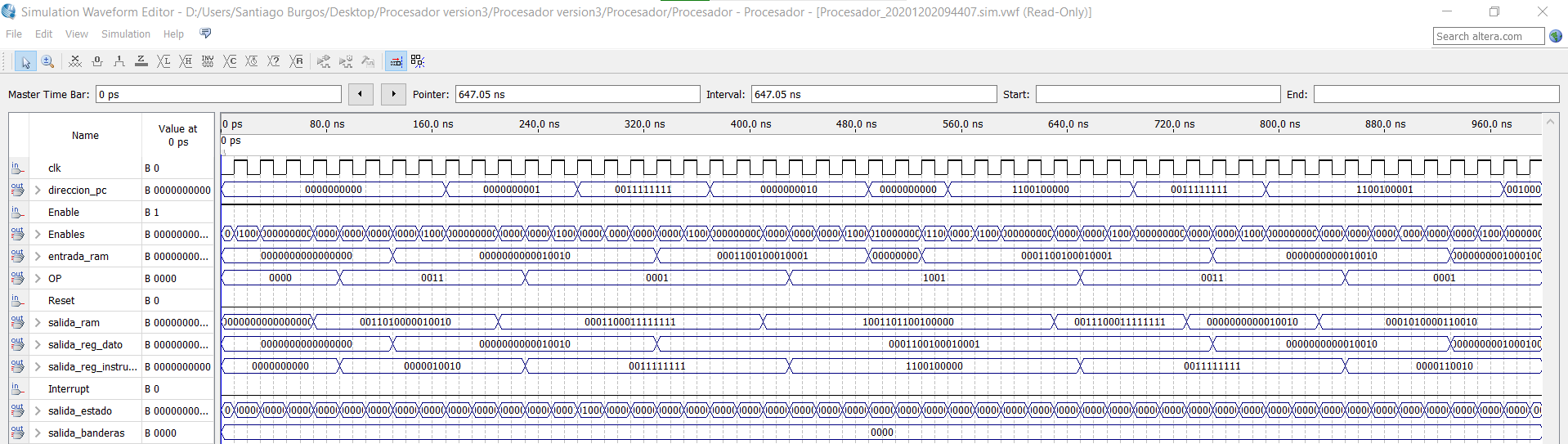


Figura 32. Simulación de distintas instrucciones.

En la siguiente Figura 33, se presenta la simulación realizada comprobando el funcionamiento correcto de la instrucción de lectura y sus distintos modos de direccionamiento en donde se puede apreciar las acciones que ejecuta el procesador y que las hace de forma cíclica y no directa.

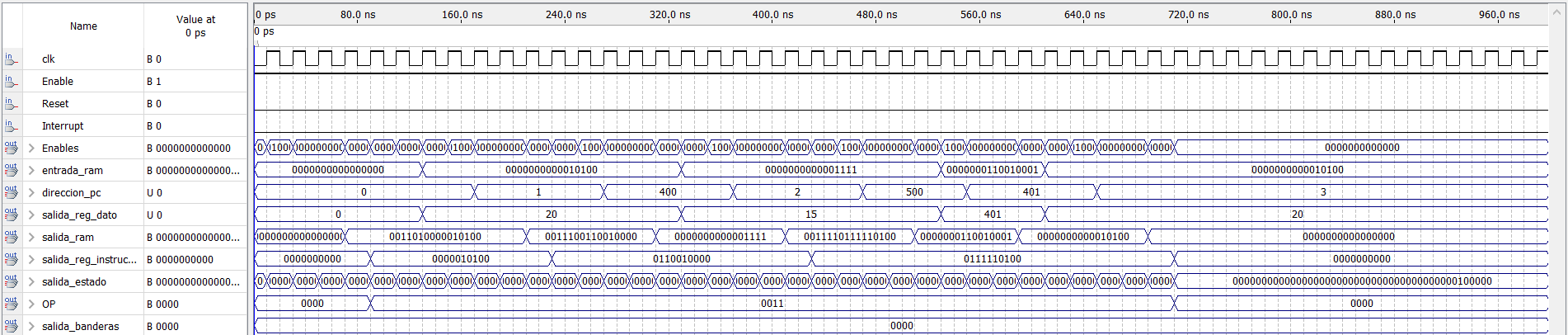


Figura 33. Simulación de la instrucción de Lectura (Move in).

En la Figura 34, se presentan los resultados obtenidos al simular la instrucción de lectura de memoria y sus múltiples direccionamientos. Observamos que se ejecutan las instrucciones de la forma esperada y en la secuencia deseada.

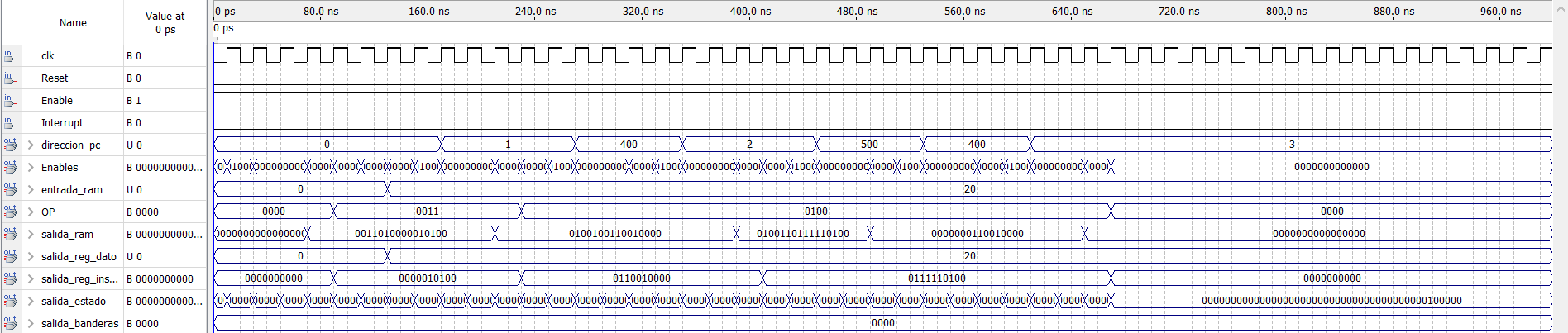


Figura 34. Simulación de la instrucción de Escritura (Move out).

En la Figura 35, se presenta la simulación realizada comprobando la instrucción de suma al igual que la simulación en sus tres diferentes modos de direccionamiento. Podemos observar que la instrucción de suma se ejecuta de acuerdo con los estados planteados y se observa el resultado obtenido guardado en el registro de datos.

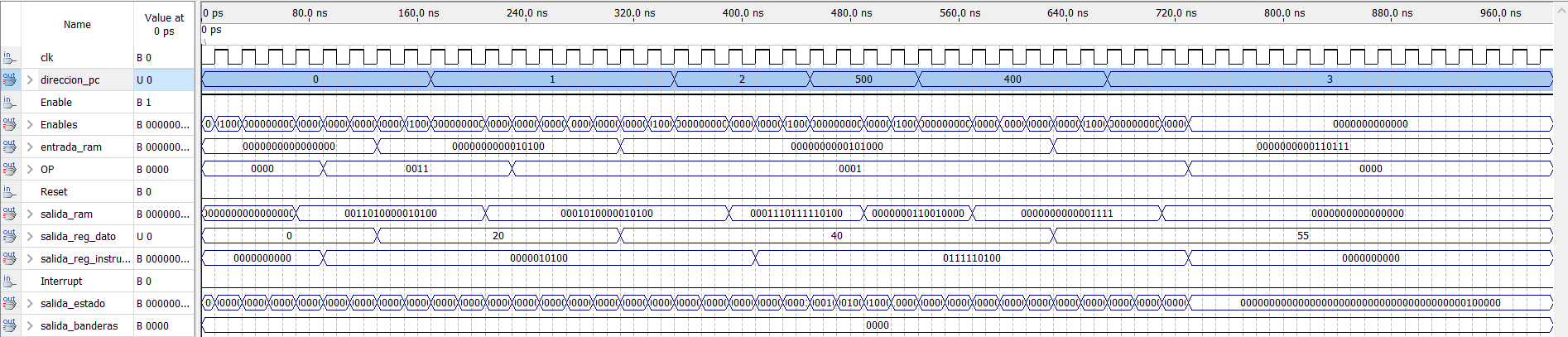


Figura 35. Simulación de la instrucción de Suma.

En la Figura 36, se muestra la simulación realizada del procesador ejecutando la instrucción de resta y sus tres modos de direccionamiento implementados. Al igual que en las instrucciones de suma se confirma un funcionamiento adecuado y una obtención del resultado en el registro de datos acertada.

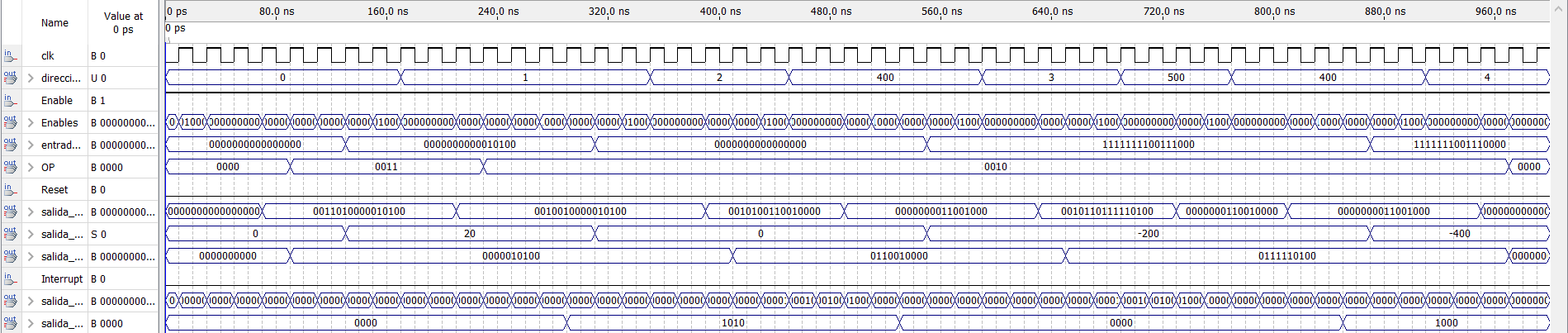


Figura 36. Simulación de la instrucción de Resta.

En la Figura 37, se presenta el funcionamiento del procesador ejecutando las instrucciones de comparación y sus modos de direccionamiento asi como del correcto funcionamiento del registro de banderas de la ALU. Como se mencionó en el protocolo de simulación, se observa que la comparación realizada y las banderas obtenidas del registro de banderas concuerdan con las instrucciones y lo escrito el programa de la memoria.

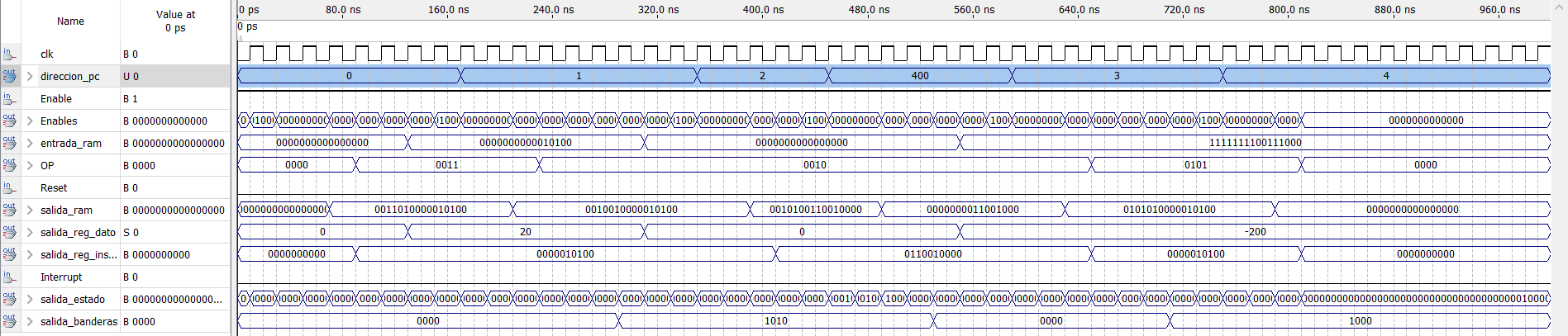


Figura 37. Simulación de la instrucción de Comparación.

En la Figura 38, se presenta la simulación realizada para comprobar el funcionamiento correcto de las instrucciones de salto que tiene el procesador, siendo estas: salto <, salto>, salto sin condición y salto a subrutina. Como se puede aprecia en la señal “dirección Pc” la cual es la que envía la dirección a la memoria se observan los saltos a posiciones de memoria realizados de forma correcta.

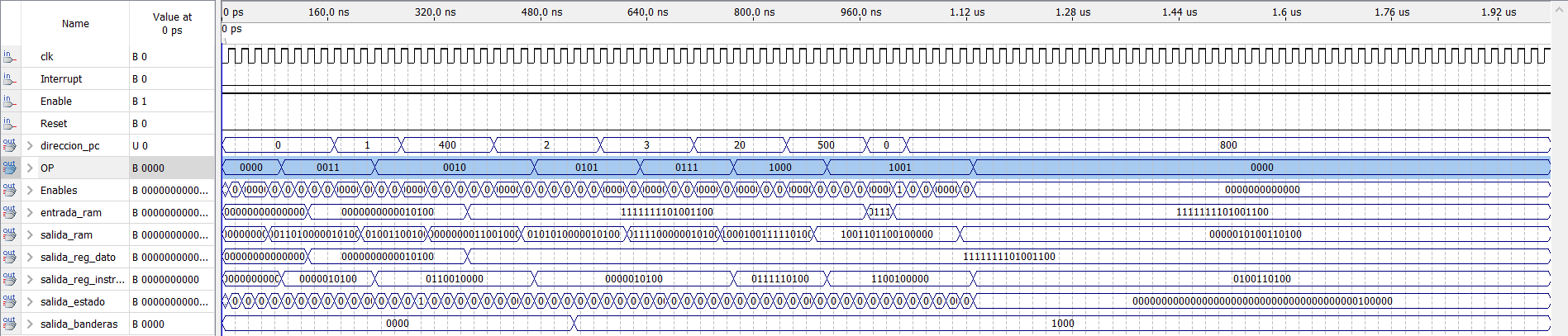


Figura 38. Simulación de las instrucciones de saltos.

En la figura 39, se presenta el funcionamiento de la señal de interrupción del sistema en donde se aprecian los cambios que se realizan en las señales de “dirección pc” , “entrada\_ram”, y el paso por los estados que se ejecutan cuando llega una interrupción.

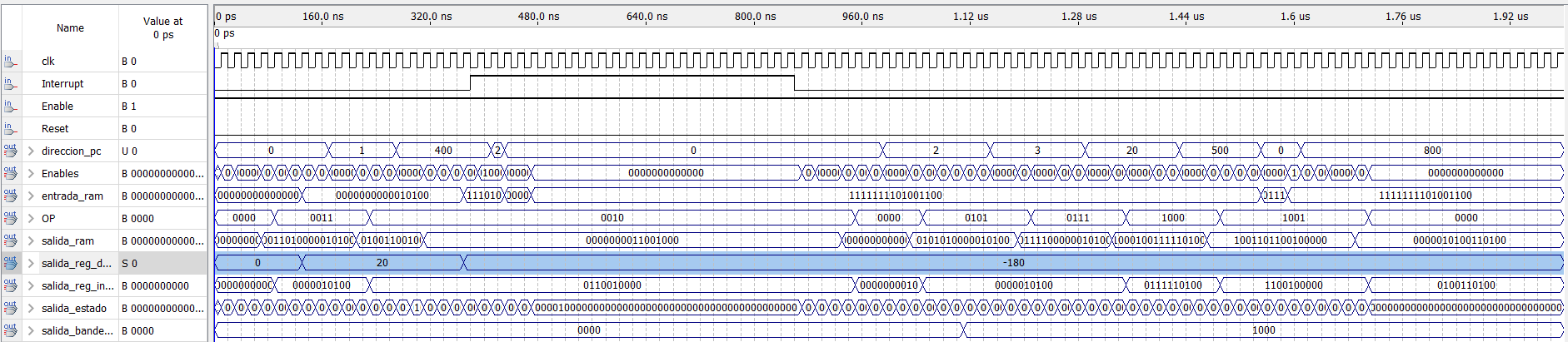


Figura 39. Simulación de la señal de Interrupción.

**REFLEXIÓN**

**Andrea**: En este proyecto se presentaron varios retos tanto en la construcción como en el proceso de realizar lo que se quería. Sin embargo, pienso que todos y cada uno de los integrantes ayudaron y cooperaron en la construcción de este proyecto, fue un muy buen trabajo en equipo, aunque. Por mi parte, sentí mucho apoyo por parte de mis dos compañeros debido a los horarios en los cuales yo estaba disponible y siempre intentamos hacer todo juntos para lograr comprender todo o que se estaba haciendo en el trabajo. Personalmente, encontré problemas sobre todo en la parte del control del procesador. Debido a que son tantos estados, uno logra confundirse bastante. Sin embargo, con ayuda de mis dos compañeros se logró un entendimiento y finalmente se llegaron a acuerdos para poder ensamblarlo dentro de nuestro proyecto. Es un proyecto que requiere de bastante tiempo, sin embargo, logramos con mucha dedicación hacer lo necesario para su funcionamiento. Me gustó mucho el trabajo en equipo y la dinámica que se llevó, uno de los trabajos en mi carrera que me he disfrutado no solo por el conocimiento obtenido sino por el trabajo en equipo que se logro por medio de este.

**Santiago:** Después de haber pasado por todos los trabajos y tareas realizadas durante el semestre y de haber podido realizar este proyecto final, siento quelos conceptos e ideas aprendidos se pudieron aplicar satisfactoriamente en cada una de las entregas realizadas. También he podido presenciar el avance tanto grupal como personal en cuanto al entendimiento de los temas propuestos por la materia. Siento que lo que puedo aprender es mucha más y me gustaría poder seguir aprendiendo sobre los procesadores, sus inicios su futuro y si es posible hacer parte de proyectos en el futuro. En cuanto al trabajo y “convivencia” grupal siento que ha sido de lo mejor, tanto porque nos hemos entendido a lo largo de toda la materia como por el apoyo que nos damos mutuamente cuando uno de los integrantes necesita ayuda con algún tema que no entiende o con el apoyo en momento en los que reunirnos es complicado o no fuera muy factible.

**Dayanna**: Como autoevaluación creo que el rendimiento ha sido excelente. Creo que el trabajo de todos los integrantes ha sido excelente, todos hemos realizado los trabajos asignados y cumplimos con los requerimientos y las fechas establecidas para la entrega. Con respecto a la elaboración del proyecto, se logró diseñar todos y cada uno de los componentes necesarios para el funcionamiento del procesador. El proyecto ayuda a aclarar aun mas conceptos de los que de pronto al principio de la materia no se tenían totalmente claros. El avance personal y grupal es notorio y logre disfrutar realizar los componentes de este proyecto. Una de las dificultades que se presentó en la elaboración del proyecto fue evidentemente el control, debido a que este contenía muchos estados y debíamos tener mucho cuidado fijándonos bien en que compuertas utilizar. Además, nuestro diagrama de bloques tuvo varios cambios en torno al entendimiento tanto del procesador como del control.

Diagrama

Descripción generada automáticamenteANEXO A -Diagrama en bloques ALU