**ARQUITECTURA DE COMPUTADORAS**

INTEGRADOR. PARTE I

**Alumno:** Küster Joaquín

**Profesor:** Claudio Biale

**Universidad:** UNaM - Apóstoles, Misiones

**Índice**

[Introducción 2](#_Toc130470427)

[ISA (Instruction Set Architecture) 3](#_Toc130470428)

[Formato de Instrucciones 3](#_Toc130470429)

[Instrucción NOP 4](#_Toc130470430)

[Instrucciones aritmético-lógicas 4](#_Toc130470431)

[Instrucciones de carga 5](#_Toc130470432)

[Instrucciones de almacenamiento 5](#_Toc130470433)

[Instrucciones de salto condicional 5](#_Toc130470434)

[Camino de datos 7](#_Toc130470435)

[Construcción del camino de datos 7](#_Toc130470436)

[Etapas del ciclo de ejecución 7](#_Toc130470437)

[Obtención 8](#_Toc130470438)

[Decodificación 10](#_Toc130470439)

[Ejecución 13](#_Toc130470440)

[Instrucciones 13](#_Toc130470441)

[Instrucción NOP 13](#_Toc130470442)

[Instrucciones aritmético-lógicas 13](#_Toc130470443)

[Instrucciones de carga-almacenamiento 16](#_Toc130470444)

[Instrucción de salto condicional 17](#_Toc130470445)

[Camino de datos completo 19](#_Toc130470446)

[Ejecución de instrucciones 20](#_Toc130470447)

[Desventajas 23](#_Toc130470448)

[Bibliografía 23](#_Toc130470449)

# **Introducción**

El presente informe detalla el diseño y construcción de un procesador monociclo de 16 bits.

Este proyecto tiene como objetivo principal desarrollar un sistema capaz de ejecutar instrucciones aritmético-lógicas, carga, almacenamiento y salto, en un solo ciclo de reloj.

Gran parte de lo necesario para construir el procesador es común a todas las instrucciones.

Para cada instrucción, los dos primeros pasos son idénticos:

1. Enviar el contenido del PC a la memoria, cargar la instrucción de memoria e incrementar el PC.
2. Leer uno o dos registros, dependiendo de la instrucción.

Los siguientes pasos dependen de cuál sea la instrucción.

Las **unidades funcionales** que se utilizan son de dos tipos:

* Combinacionales: son aquellas en las que la salida depende exclusivamente del valor de las entradas. Algunos ejemplos: puertas lógicas, decodificadores, multiplexores, etc.
* Secuenciales: son aquellas cuya salida depende no solo de las entradas, sino también del estado que almacenan. Algunos ejemplos: registros, contadores y temporizadores, etc.

# **ISA** (Instruction Set Architecture)

A continuación se define el conjunto de instrucciones que el procesador puede ejecutar y a su vez, se establece cómo se realiza la comunicación entre el software y el hardware.

## **Formato de Instrucciones**

Las instrucciones que se desean implementar pertenecen a uno de los siguientes formatos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato #1** (Tipo R) | | | | |
| OPCODE | RD | RF1 | RF2 | XXX |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 3 | 2 - 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato #2** (Tipo I) | | | |
| OPCODE | RD | RF1 | INM |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 0 |

Algunas características:

* En total, el procesador puede ejecutar 11 instrucciones
* Todos los formatos manejan un OPCODE de 4 bits.
* Generalmente, los campos para los registros que van a ser leídos o escritos están siempre en la misma ubicación para todas las instrucciones.
* Los campos inmediatos siempre son extendidos en signo.

### **Instrucción NOP**

No realiza nada

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato #1** | | | | |
| OPCODE (0000) | XXX | XXX | XXX | XXX |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 3 | 2 - 0 |

### **Instrucciones aritmético-lógicas**

ALU rd, rf1, rf2, ALUOP

* **BR[rd] ← BR[rf1] ALUOP BR[rf2]**

El código ALUOP funciona con las siguientes combinaciones binarias:

* (00) Suma (+)
* (01) Resta (-)
* (10) Y lógico bit a bit (&)
* (11) O lógico bit a bit (|)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato #1** | | | | |
| OPCODE (0001) | RD | RF1 | RF2 | ALUOP |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 3 | 2 - 0 |

SUMAI rd, rf1, inm

* **BR[rd] ← BR[rf1] + ExtSigno(inm, 16)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato #2** | | | |
| OPCODE (0010) | RD | RF1 | INM |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 0 |

### **Instrucciones de carga**

CARGA rd, rf1, rf2

* **BR[rd] ← MEMDATA[BR[rf1] + BR[rf2]]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato #1** | | | | |
| OPCODE (0011) | RD | RF1 | RF2 | XXX |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 3 | 2 - 0 |

### **Instrucciones de almacenamiento**

ALMACEN rd, rf1

* **MEMDATA[BR[rd]] ← BR[rf1]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Formato #1** | | | | |
| OPCODE (0100) | RD | RF1 | XXX | XXX |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 3 | 2 - 0 |

### **Instrucciones de salto condicional**

IGUAL rd, rf1, inm

* **Si (rd == rf1) → PC ← PC + ExtSigno(inm, 8)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato #2** | | | |
| OPCODE (0101) | RD | RF1 | INM |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 0 |

MENOR rd, rf1, inm

* **Si (rd < rf1) → PC ← PC + ExtSigno(inm, 8)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato #2** | | | |
| OPCODE (0110) | RD | RF1 | INM |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 0 |

MENORIG rd, rf1, inm

* **Si (rd <= rf1) → PC ← PC + ExtSigno(inm, 8)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Formato #2** | | | |
| OPCODE (0111) | RD | RF1 | INM |
| 15 - 12 | 11 - 9 | 8 - 6 | 5 - 0 |

# **Camino de datos**

A continuación, se detallan los elementos de hardware necesarios para llevar a cabo las operaciones en el procesador.

## **Construcción del camino de datos**

La estrategia para construir el camino de datos es la siguiente:

* Examinar los elementos requeridos para la ejecución de cada una de las instrucciones.
* Construir un camino de datos para cada instrucción.
* Determinar las señales de control necesarias para cada una de ellas.
* Finalmente, unir las distintas partes para generar el camino de datos completo:
  + Qué rutas del camino de datos se utilizan en cada instrucción vendrá marcado por las correspondientes **señales de control**.

## **Etapas del ciclo de ejecución**

Se trata de una forma implementada para organizar el procesamiento de las instrucciones en el procesador. Las diferentes etapas del ciclo de ejecución de una instrucción son las siguientes:

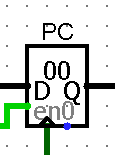
* Obtención
* Decodificación
* Ejecución

Cada una de estas etapas involucran varias operaciones.

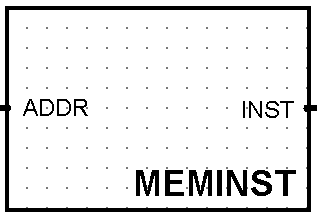
### **Obtención**

En esta etapa, el procesador obtiene la siguiente instrucción del programa desde la memoria a partir de la dirección indicada en el contador del programa para posteriormente decodificarla. Esta etapa necesita de los siguientes elementos:

* Registro PC
* Memoria de Instrucciones
* Sumador

**Registro PC**(Program Counter)**:**

* **clk** → entrada de sincronismo de escritura (flanco de subida)
* **en** → activación de escritura
* **D[15:0]** → Siguiente dirección de memoria a leer
* **Q[15:0]** → Dirección de memoria a leer

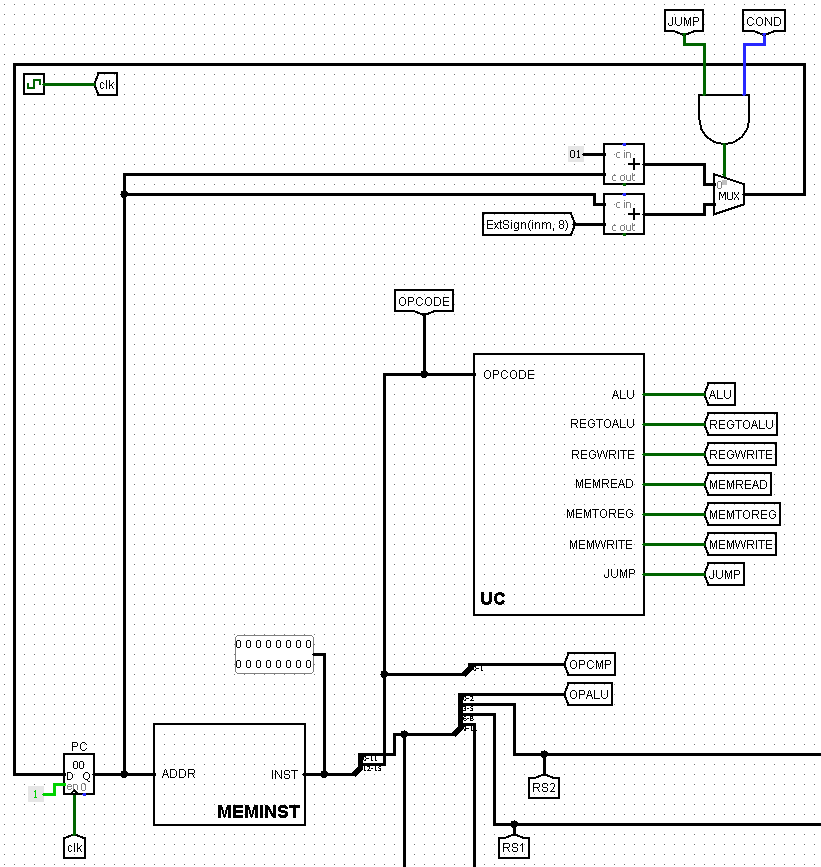
**Memoria de instrucciones**(ROM)**:**

* Bus de direcciones de 8 bits
* Bus de datos de 16 bits

Posee los siguientes puertos de entrada y salida:

* **ADDR[7:0]** → Dirección de memoria a leer
* **INST[15:0]** → Instrucción leída de la dirección ADDR

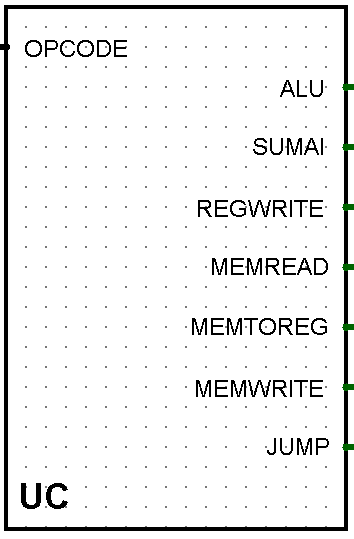
Cuenta con la siguiente parte del camino de datos:



### **Decodificación**

En esta etapa, el procesador decodifica la instrucción y determina la operación que debe realizar. Esta etapa necesita de los siguientes elementos:

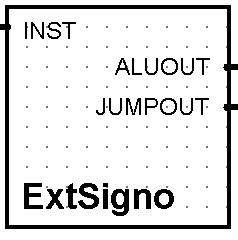
* Unidad de control
* Extensor de signo

 **Unidad de Control** (UC)**:** Emite un total de 7 señales de control (1 bit c/u).

* **OPCODE[3:0]** → Código de operación
* **ALU [0:0]** → Señal de control para indicar que se está ejecutando una instrucción aritmética-lógica
* **SUMAI[0:0]** → Señal de control para indicar que se está ejecutando una instrucción SUMAI
* **REGWRITE[0:0]** → Señal de control para activar la escritura en el banco de registros
* **MEMREAD[0:0]** → Señal de control para activar la lectura en la memoria de datos
* **MEMTOREG[0:0]** → Señal de control para ir desde memoria hacia el banco de registros
* **MEMWRITE[0:0]** → Señal de control para activar la escritura en la memoria de datos
* **JUMP[0:0]** → Señal de control para indicar que se está ejecutando una instrucción de salto condicional

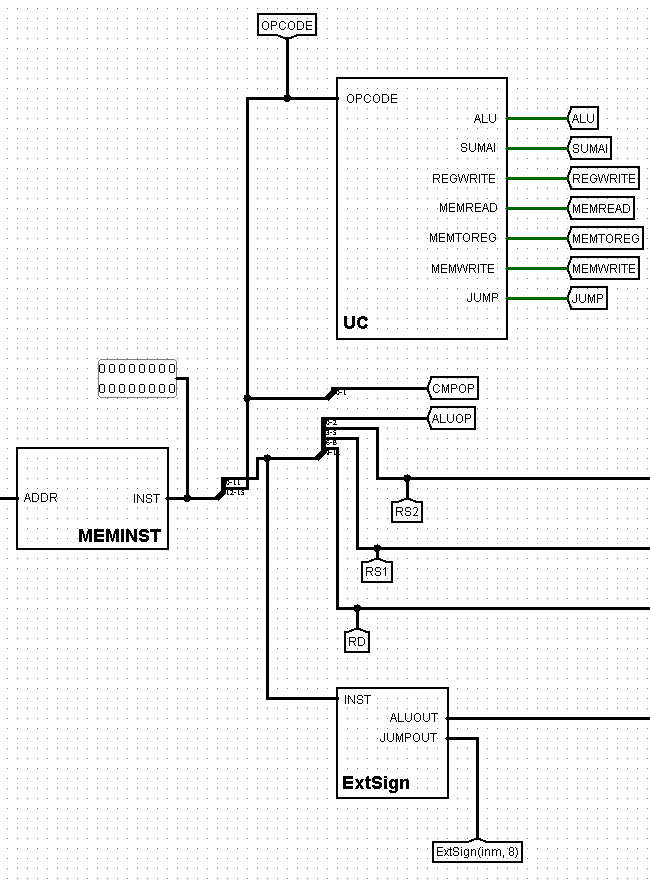
El valor de las señales de control está determinado por el código de operación de la instrucción en curso. El funcionamiento de la unidad de control está definido por la siguiente tabla:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **INST** | **JUMP** | **MEMREAD** | **MEMTOREG** | **MEMWRITE** | **ALU** | **SUMAI** | **REGWRITE** |
| **NOP** | X | 0 | X | 0 | X | X | 0 |
| **CARGA** | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| **ALMACEN** | 0 | 0 | X | 1 | X | X | 0 |
| **ALU** | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| **SUMAI** | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **IGUAL** | 1 | 0 | X | 0 | X | 0 | 0 |
| **MENOR** | 1 | 0 | X | 0 | X | 0 | 0 |
| **MENORIG** | 1 | 0 | X | 0 | X | 0 | 0 |

**Unidad de Extensión de Signo****:**

* **INST[11:0]** → Instrucción sin el código de operación
* **ALUOUT[15:0]** → Inmediato de 6 bits extendido a 16 bits utilizado por la ALU
* **JUMPOUT[0:0]** → Inmediato de 6 bits extendido a 8 bits utilizado por las instrucciones de salto condicional

Cuenta con la siguiente parte del camino de datos:



### **Ejecución**

En esta etapa, el procesador realiza la operación indicada por la instrucción, utilizando la ALU u otros componentes del procesador.

## **Instrucciones**

A continuación, se define el funcionamiento de las 11 instrucciones disponibles en el procesador.

### **Instrucción NOP**

La instrucción NOP no realiza ninguna operación. Por lo tanto, no cuenta con ningún camino de datos.

### **Instrucciones aritmético-lógicas**

Las instrucciones aritmético-lógicas realizan operaciones aritméticas y lógicas. En este caso, se pueden realizar las siguientes operaciones:

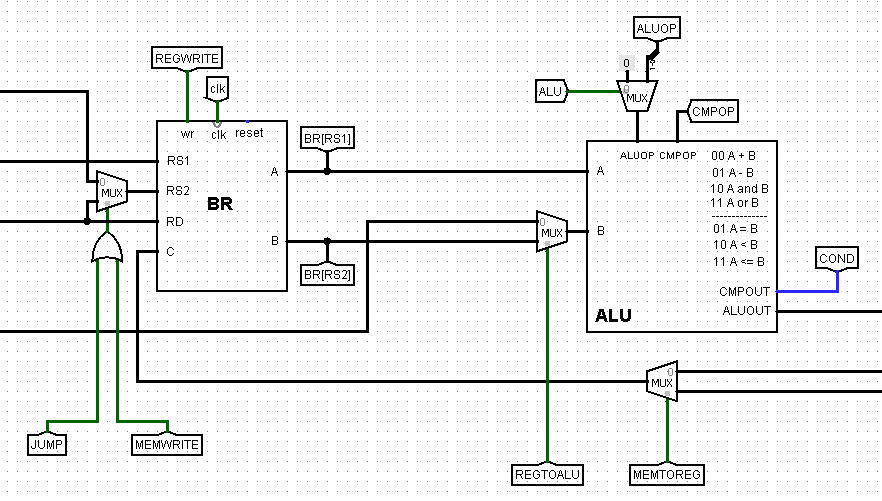
* Suma
* Resta
* AND lógico (Y)
* OR lógico (O)

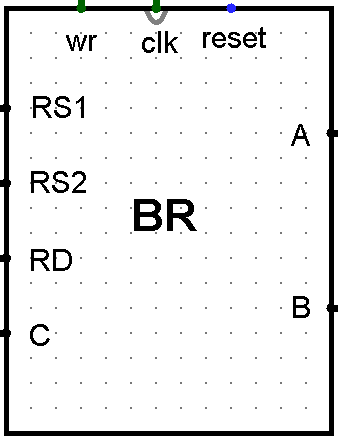
Estas instrucciones necesitan de los siguientes elementos:

* Banco de registros
* ALU

Adicionalmente, se dispone de la instrucción **SUMAI** que realiza una suma entre el registro rf1 y un inmediato de 6 bits para posteriormente almacenar el resultado en el registro rd.

Cuenta con la siguiente parte del camino de datos:

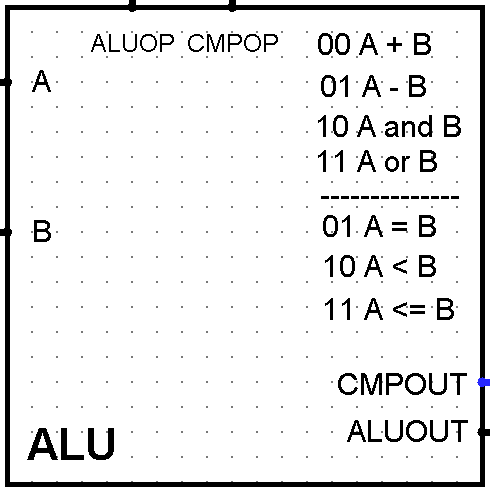


­**Banco de Registros** (BR)**:**

* 8 registros de propósito general de 16 bits (r0 cableado a 0)
* 2 puertos de lectura (A, B)
* 1 puerto de escritura (C)

Posee los siguientes puertos de entrada y salida:

* **clk** → entrada de sincronismo de escritura (flanco de subida)
* **wr** → activación de escritura en el banco de registros
* **reset** → señal de reset de los registros
* **RS1[2:0]** → número de registro a leer por puerto A
* **RS2[2:0]** → número de registro a leer por puerto B
* **RD[2:0]** → número de registro a escribir por puerto C
* **A[15:0]** → puerto de salida A (dato leído registro RS1)
* **B[15:0]** → puerto de salida B (dato leído registro RS2)
* **C[15:0]** → puerto de entrada C (dato escritura registro RD)

**ALU** (Arithmetic Logic Unit)**:**

* **ALUOP[1:0]** → Código de operación ALU
  + [00: A + B, 01: A - B, 10: A and B, 11: A or B]
* **CMPOP[1:0]** → Código de operación comparativa
  + [00: --, 01: A = B, 10: A < B, 11: A <= B]
* **A[15:0]** → Entrada datos A
* **B[15:0]** → Entrada datos B
* **ALUOUT[15:0]** → Salida de la operación ALU (resultado)
* **CMPOUT[0:0]** → Salida de la operación comparativa (condición)

El código ALUOP determina la operación de la ALU que se va a realizar de acuerdo con la siguiente tabla de verdad:

|  |  |
| --- | --- |
| ALUOP | Función |
| 00 | Suma |
| 01 | Resta |
| 10 | AND lógico |
| 11 | OR lógico |

Luego, el código CMPOP determina la operación comparativa que se va a realizar:

|  |  |
| --- | --- |
| CMPOP | Función |
| 00 | -- |
| 01 | Igual |
| 10 | Menor |
| 11 | Menor o igual |

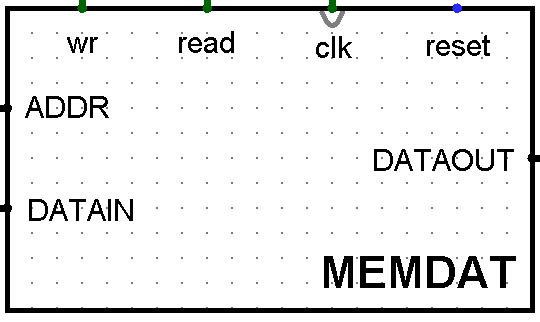
### **Instrucciones de carga-almacenamiento**

Las instrucciones de carga y almacenamiento realizan transferencias de datos entre la memoria y los registros del procesador. Estas instrucciones necesitan de los siguientes elementos:

* Banco de registros
* ALU
* Memoria de datos

El funcionamiento de estas instrucciones se define de la siguiente manera:

* La instrucción **CARGA** guarda en el registro rd el valor almacenado en la dirección de memoria que surge de la suma entre los registros rf1 y rf2.
* La instrucción **ALMACEN** guarda en la dirección de memoria especificada en el registro rd el valor almacenado en el registro rd de la instrucción de carga anterior.

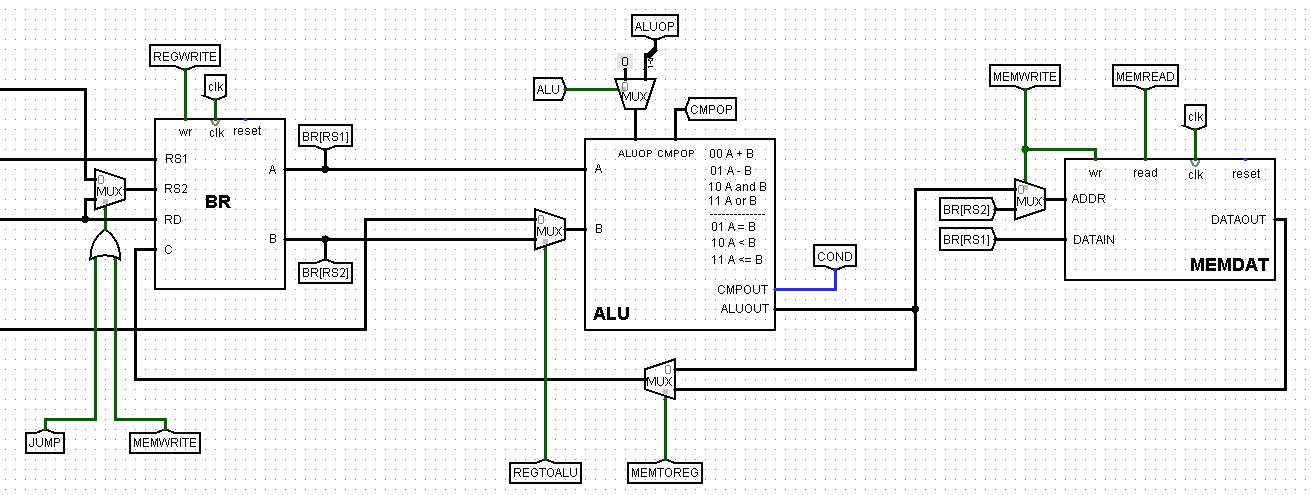
**Memoria de Datos** (RAM)**:**

* Bus de direcciones de 16 bits
* Bus de datos de 16 bits (entrada / salida)

Posee los siguientes puertos de entrada y salida:

* **clk** → entrada de sincronismo de escritura (flanco de subida)
* **wr** → señal activación de escritura en memoria
* **read** → señal activación de lectura en memoria
* **reset** → señal de reset de la memoria
* **ADDR[15:0]** → Dirección de memoria a leer / escribir
* **DATAIN[15:0]** → Dato a escribir en la dirección ADDR
* **DATAOUT[15:0]** → Dato leído de la dirección ADDR

Cuenta con la siguiente parte del camino de datos:

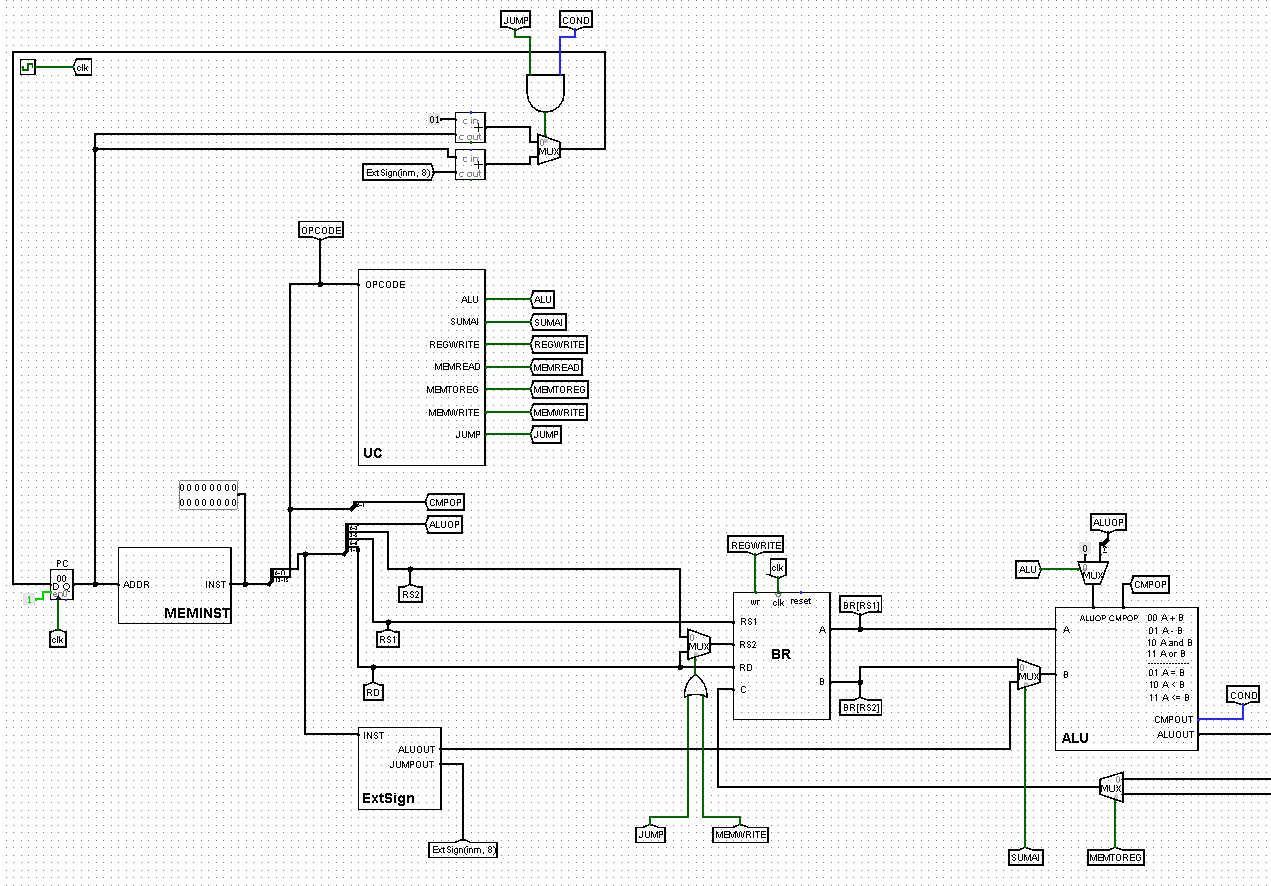


### **Instrucción de salto condicional**

Las instrucciones de salto condicional funcionan mediante la evaluación de una condición y, en función del resultado de esta evaluación, cambian el flujo de ejecución del programa a una dirección de memoria específica. Estas instrucciones necesitan de los siguientes elementos:

* Banco de registros
* ALU
* Extensor de signo
* Sumador

Cuenta con la siguiente parte del camino de datos:



Las instrucciones de salto condicional disponibles en el procesador son las siguientes:

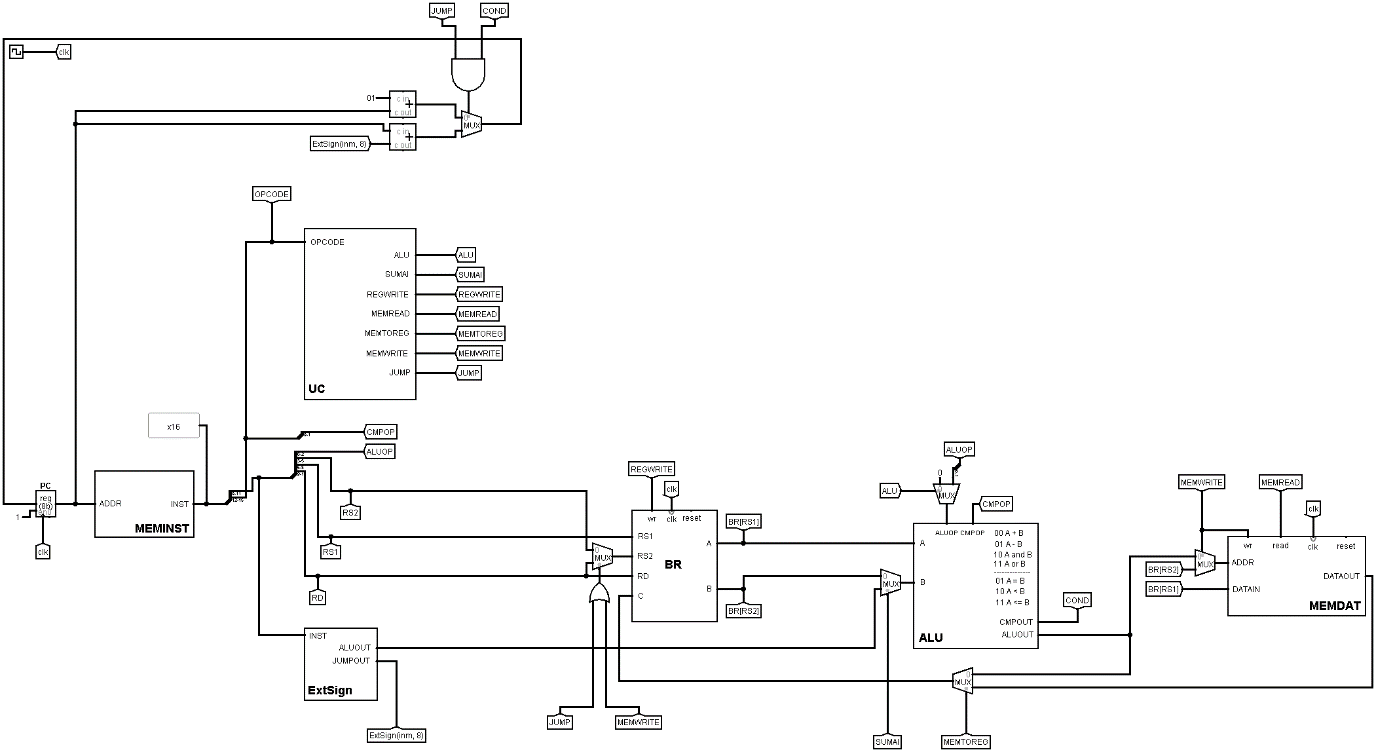
* La instrucción **IGUAL** compara el contenido de los registros rf1 y rf2, y si son iguales se realiza un salto a PC + inmediato de 6 bits.
* La instrucción **MENOR** compara el contenido de los registros rf1 y rf2, y si rf1 es menor que rf2 se realiza un salto a PC + inmediato de 6 bits.
* La instrucción **MENORIG** compara el contenido de los registros rf1 y rf2, y si rf1 es menor o igual que rf2 se realiza un salto a PC + inmediato de 6 bits.

## **Camino de datos completo**

Para completar el diseño de un procesador basta con:

* Combinar las partes vistas anteriormente en un único camino de datos.

A continuación, se muestra el camino de datos completo:



# **Ejecución de instrucciones**

A modo de ejemplo, se realizó un programa sencillo cuya única finalidad es comprobar que todas las instrucciones definidas en el lenguaje ensamblador cumplen su correcto funcionamiento dentro del procesador.

A tener cuenta, para cargar las instrucciones del programa se debe acceder a la memoria de instrucciones dentro del archivo “.circ” del procesador completo, donde se encuentra una memoria ROM. En cada celda de esa memoria se almacena una instrucción de 16 bits c/u, dado que la misma trabaja con valores hexadecimales, antes de cargar las instrucciones, previamente escritas en binario, se debe convertirlas a hexadecimal.

A continuación, se presenta el programa en cuestión en lenguaje ensamblador, binario y por último, en hexadecimal.

**En ensamblador →**

**NOP** // No realiza nada

**SUMAI ax, ax, 4** // Realiza una suma entre el valor numérico 4 y el valor almacenado en la dirección de registro especificada en ax, para almacenar el resultado en la misma dirección de registro → ax = ax + 4

**SUMAI bx, bx, 2** // Mismo procedimiento con el registro bx y el valor numérico 2 → bx = bx + 2

**ALU cx, ax, bx, SUMA** // Realiza una suma entre los valores almacenados en las direcciones de registro especificadas en ax y bx, para almacenar el resultado en la dirección de registro especificada en cx → cx = ax + bx

**ALU dx, ax, bx, RESTA** // Realiza una resta entre los valores almacenados en las direcciones de registro especificadas en ax y bx, para almacenar el resultado en la dirección de registro especificada en dx → dx = ax – bx

**IGUAL bx, dx, 2** // Compara la igualdad entre los valores almacenados en las direcciones de registro especificadas en bx y dx, si es verdadero, realiza un salto sumando al PC + 2 → Si (bx == dx) → PC ← PC + 2

**NOP** // No realiza nada

**CARGA ex, bx, dx** // Realiza una suma entre los valores almacenados en las direcciones de registro especificadas en bx y dx, para construir una nueva dirección de memoria de la cual se lee y almacena un valor en la dirección de registro especificada en ex → ex = MEMDATA[bx + dx]

**ALMACEN bx, ex** // Obtiene una dirección de memoria del valor almacenado en bx y almacena el contenido de ex en esa dirección → MEMDATA[bx] = ex

**En binario →**

0

0010 001 001 000100

0010 010 010 000010

0001 011 001 010 000

0001 100 001 010 001

0101 010 100 000010

0

0011 101 010 100 000

0100 010 101 000 000

**En hexadecimal →**

0000

2244

2482

1650

1851

5502

0000

3AA0

4540

Para finalizar se realizó un video sencillo y rápido donde se realiza una ejecución del programa en el procesador monociclo, instrucción por instrucción.

**Link del video:** <https://youtu.be/s9WikaYVWPw>

# **Desventajas**

Es posible realizar un procesador utilizando esta técnica. Sin embargo, no se utiliza debido a que es ineficiente:

* El tiempo de ciclo viene determinado por la instrucción más larga.
* Las unidades funcionales solo pueden utilizarse una vez por ciclo.

# **Bibliografía**

<https://www.fceqyn.unam.edu.ar/moodle/pluginfile.php/202193/mod_resource/content/0/procesador%20monociclo.pdf>

<https://lorca.act.uji.es/curso/latex/ejemplo/el_procesador_p.pdf>

<https://www.fceqyn.unam.edu.ar/moodle/pluginfile.php/202191/mod_resource/content/0/isas_introduccion.pdf>

<https://www.academia.edu/20204917/DISE%C3%91O_DE_UN_MICRO_PROCESADOR_DE_16_BITS>

<https://www.youtube.com/watch?v=oCCKGZDW2hI>

<http://www.hpca.ual.es/~vruiz/docencia/laboratorio_arquitectura/proyectos/00-01/RoceroBlanes/exe/Doc_HTML/docu.html>

<http://pearsonespana.blob.core.windows.net/books/9788483226506.pdf>

<http://quegrande.org/apuntes/EI/2/ECm1/teoria/07-08/tema_4.pdf>

<https://www.studypool.com/documents/3568473/arquitectura-de-computadores-procesador-monociclo-risc-v>