

PRÁCTICA 3

Diseño de un procesador con banco de registros



12 de abril de 2021

Álvaro fraidias y javier mengod

Contenido

[1. Introducción 2](#_Toc69402822)

[2. Trabajo previo 2](#_Toc69402823)

[2.1 Formatos de instrucciones 2](#_Toc69402824)

[2.2 Instrucción Mov 3](#_Toc69402825)

[2.3 Instrucción ADD 3](#_Toc69402826)

[3.Problema a resolver 3](#_Toc69402827)

[3.1 Instrucciones Mov 3](#_Toc69402828)

[3.2 Instrucciones ADD 4](#_Toc69402829)

[4. Procesador Monociclo 5](#_Toc69402830)

[4.1 Camino de datos 5](#_Toc69402831)

[4.2 Unidad de control 6](#_Toc69402832)

[4.3 Análisis temporal 6](#_Toc69402833)

[5. Procesador Multiciclo 7](#_Toc69402834)

[5.1 Camino de datos 7](#_Toc69402835)

[5.2 Unidad de control 8](#_Toc69402836)

[5.3 Análisis temporal 9](#_Toc69402837)

[6.Conclusión 10](#_Toc69402838)



# 1. Introducción

En esta práctica se pretende crear un procesador simple que ejecute en un único ciclo las instrucciones mov y add. Acto seguido se añadirán 3 registros intermedios para implementar el procesador multiciclo que más tarde explicaremos. Una vez tengamos los dos procesadores se analizarán sus tiempos y se compararán.

# 2. Trabajo previo

Antes de ponernos con logisim debemos diseñar sobre papel como debería ser nuestro procesador. Para ello dibujamos los siguientes componentes: un banco de registros BR32, una ROM, un contador, un sumador de 32 bits, un extensor de signo y un multiplexor.

Una vez dibujados los conectaremos de forma que:

* En cada dirección de la ROM se escribirá una instrucción.
* El contador se utilizará para direccionar la ROM.
* Con el sumador se calculará la suma de los dos registros leídos del BR32.
* El multiplexor seleccionará el dato adecuado a escribir en el BR32.

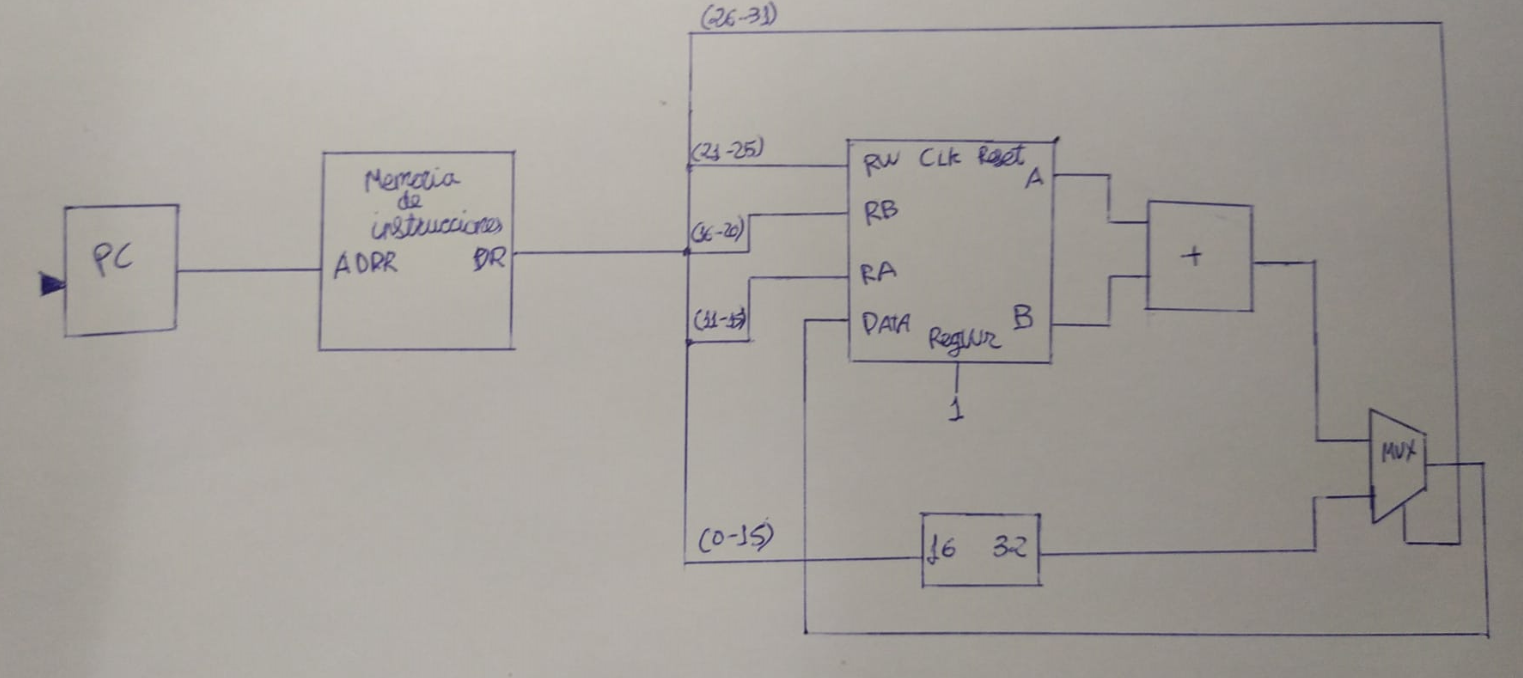


Ilustración :Diseño procesador monociclo

## 2.1 Formatos de instrucciones

Para ejecutar las dos instrucciones hemos utilizado los siguientes formatos de instrucción con 32 bits cada uno:

RA= s

RB = s

RW = s

INMEDIATO= ,

Código operación = ,

## 2.2 Instrucción Mov

Para la instrucción Mov el formato será el siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Código operación | RW | NADA | INMEDIATO |

31 26 25 21 20 16 15 0

* Del bit 0 al 15 (16 bits) irá codificado el número que se pretende cargar en el registro.
* Del bit 16 al 20 no dará igual los bits que haya.
* Del bit 21 al 25 (5 bits) estará codificado el registro destino donde se guardará el número inmediato.
* Del bit 26 al 31 irá codificado el código de operación que al pasar por la UC determinará el valor de las señales de control. En este caso solo nos será útil el bit 26 ya que solo tenemos dos instrucciones que se puede codificar con un único bit.

## 2.3 Instrucción ADD

Para la instrucción ADD el formato será el siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código operacion | RW | RB | RA | NADA |

31 26 25 21 20 16 15 11 0

* Del bit 0 al 11 (12 bits) la instrucción no nos dará ninguna información.
* Del bit 11 al 15 (5 bits) vendrá codificado el primer operando del sumador proveniente de los 32 registros del banco de registros.
* Del bit 16 al 20 (5 bits) irá codificado el segundo operando del sumador proveniente de los 32 registros del banco de registros.
* Del bit 21 al 25 (5 bits) estará codificado el registro destino del banco de registros donde se guardará la suma.
* Del bit 26 al 31 (5 bits) vendrá dado el código de operación. Igual que en el caso de la instrucción Mov, solo nos interesaría el bit 26.

# 3.Problema a resolver

Vamos a escribir un programa que almacene los 6 dígitos de nuestro NIA en los registros r0 – r5, para acto seguido calcular la suma de todos los dígitos y la guardarla en r6. Para ello necesitaremos 6 instrucciones Mov y 6 instrucciones ADD. Dichas instrucciones se codifican de la siguiente manera:

## 3.1 Instrucciones Mov

**Instrucción:** lw r0, 7

**Binario:**00000100000000000000000000000111

**Hexadecimal:** 04000007

**Instrucción:** lw r1, 8

**Binario:**00000100001000000000000000001000

**Hexadecimal:**04200008

**Instrucción:** lw r2, 0

**Binario:**00000100010000000000000000000000

**Hexadecimal:**04400000

**Instrucción:** lw r3, 3

**Binario:**00000100011000000000000000000011

**Hexadecimal:**04600003

**Instrucción:** lw r4, 3

**Binario:**00000100100000000000000000000011

**Hexadecimal:**04800003

**Instrucción:** lw r5, 6

**Binario:**00000100101000000000000000000110

**Hexadecimal:**04A00006

## 3.2 Instrucciones ADD

**Instrucción:** add r6, r6, r0

**Binario:**00000000110000000011000000000000

**Hexadecimal:**00C03000

**Instrucción:** add r6, r6, r1

**Binario:**00000000110000010011000000000000

**Hexadecimal:**00C13000

**Instrucción:** add r6, r6, r2

**Binario:**00000000110000100011000000000000

**Hexadecimal:**00C23000

**Instrucción:** add r6, r6, r3

**Binario:**00000000110000110011000000000000

**Hexadecimal:**00C33000

**Instrucción:** add r6, r6, r4

**Binario:**00000000110001000011000000000000

**Hexadecimal:**00C43000

**Instrucción:** add r6, r6, r5

**Binario:**00000000110001010011000000000000

**Hexadecimal:**00C53000

# 4. Procesador Monociclo

## 4.1 Camino de datos

Este procesador será muy simple: se usará un sumador en lugar de una ALU y no tendrá memoria de datos. El hardware necesario será el siguiente:

Sea cual sea la instrucción que se desea implementar en el procesador, deberá haber un contador que incrementalmente aumente para acceder a todas las instrucciones que debe almacenar la memoria de instrucciones. Una ROM de 32 bits que almacene todas las instrucciones y un banco de registros de 32 bits para almacenar los datos.

Para implementar las instrucciones que nos pide el enunciado debemos conectar adicionalmente los siguientes dispositivos:

* Para la instrucción Mov necesariamente debe conectarse un extensor de signo ya que el dato que queremos mover de la instrucción a un registro del BR32 estará codificado con 16 bits y necesitamos 32 bits.
* Para la instrucción add debe conectarse un sumador (en un procesador más complejo se usaría la ALU)

Como se puede observar, el BR32 puede almacenar datos provenientes del extensor de signo o del sumador y por ello debemos conectar un multiplexor que decida. Para decidir si toma un valor u otro se usa una señal de control proveniente de la unidad de control de este procesador.

Al ser un procesador tan simple, el propio bit que decide la instrucción (1= MOV, 0= ADD) debe estar conectado a dicho multiplexor ya que se encargará él de decidir.

El procesador quedaría de la siguiente manera:

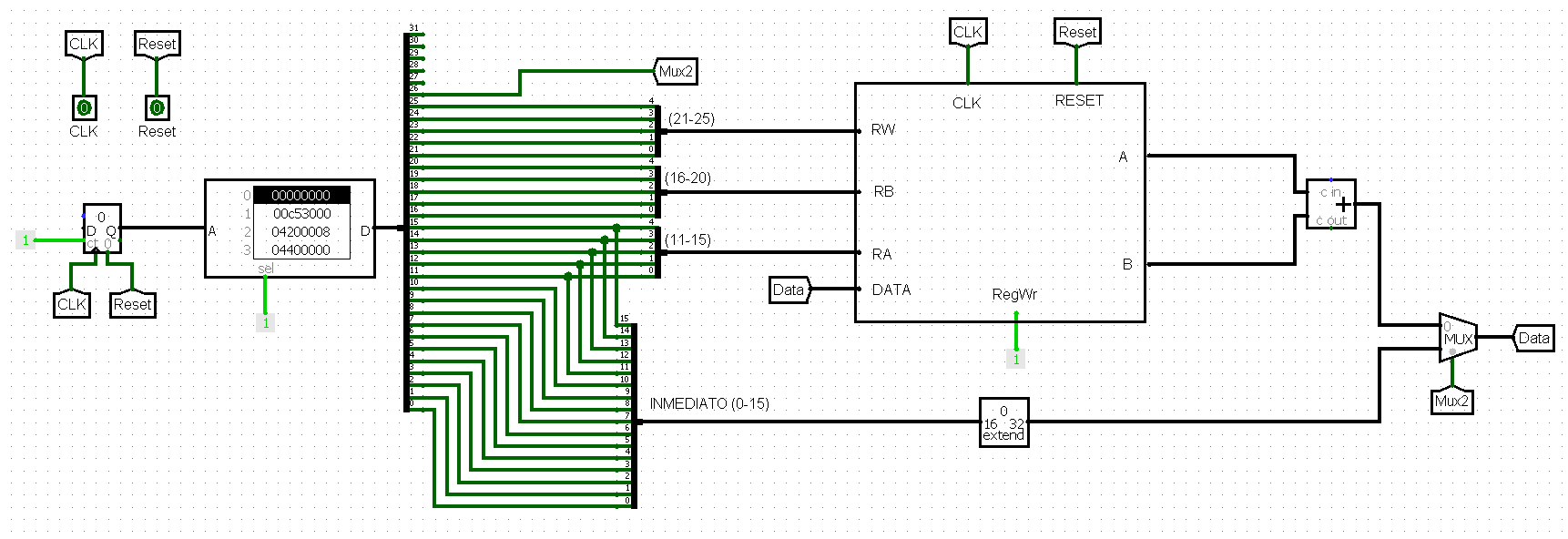


Ilustración 2: Ruta de datos procesador monociclo

## 4.2 Unidad de control

Como ya hemos dicho antes, al ser un procesador tan simple que solo ejecuta dos instrucciones hemos visto que no es necesario crear una unidad de control ya que el bit que decide si se puede escribir en el BR32 debe estar siempre a 1 ya que las dos instrucciones escriben en el BR32. Por consiguiente, solo nos queda la señal de control del multiplexor 2:1.

De esta señal se encarga el propio código: Si la operación es una Mov, el bit del código se pondrá a 1 y el dato guardado en el BR32 provendrá del extensor de signo. Si la operación es una ADD, el bit del código se pondrá a 0 y el dato guardado en el BR32 provendrá del sumador.

## 4.3 Análisis temporal

Para realizar el análisis temporal debemos tener en cuenta los dos caminos que toma el procesador: La instrucción Mov y la instrucción ADD:

* **Mov**: DelayPC + DelayROM + DelayExtensorSigno + DelayMux2:1 + TiempoSetupBR32

= 75ps + 95ps + 10ps + 60ps + 101ps = 341ps

* **ADD**: DelayPC + DelayROM + DelayBR32 + DelaySumador + DelayMux2:1 + TiempoSetupBR32

= 75ps + 95ps + 184ps + 180ps + 60ps + 101ps = 695ps

Para hacer un cálculo exhaustivo del procesador hemos tenido que diseñar un multiplexor 2:1 para saber su temporización:

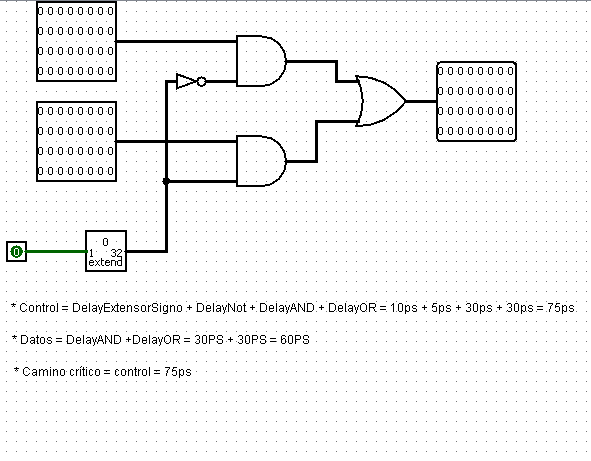


Ilustración 3: Multiplexor 2:1

Como podemos comprobar la instrucción ADD tarda más que la instrucción Mov ya que pasa por más componentes. El camino crítico del procesador monociclo es 695 picosegundos por lo que el tiempo de ciclo será de:

En cuanto al tiempo tardado por el procesador monociclo en ejecutar el programa es:

Tiempo ciclo \* número de ciclos \* número instrucciones = 695ps \* 1 \* 12 instrucciones

= 8340 ps

# 5. Procesador Multiciclo

Hemos podido comprobar que la instrucción ADD ocupa mucho tiempo y no es práctico diseñar un procesador que ejecute estas dos instrucciones y que a la hora de usarlo haya muy pocas o ninguna instrucción add. Por ello vamos a diseñar un procesador multiciclo para dividir las instrucciones de la siguiente manera:

La instrucción Mov se realizará en un ciclo y la add en 3 ciclos

## 5.1 Camino de datos

Para implementar un procesador multiciclo solo debemos conectar 3 registros adicionales que servirán como zona auxiliar para guardar los datos entre diferentes ciclos:

Un registro para cada salida del banco de registros y un registro conectado a la salida de la ALU (en este caso el sumador). Visualmente quedaría de la siguiente manera:

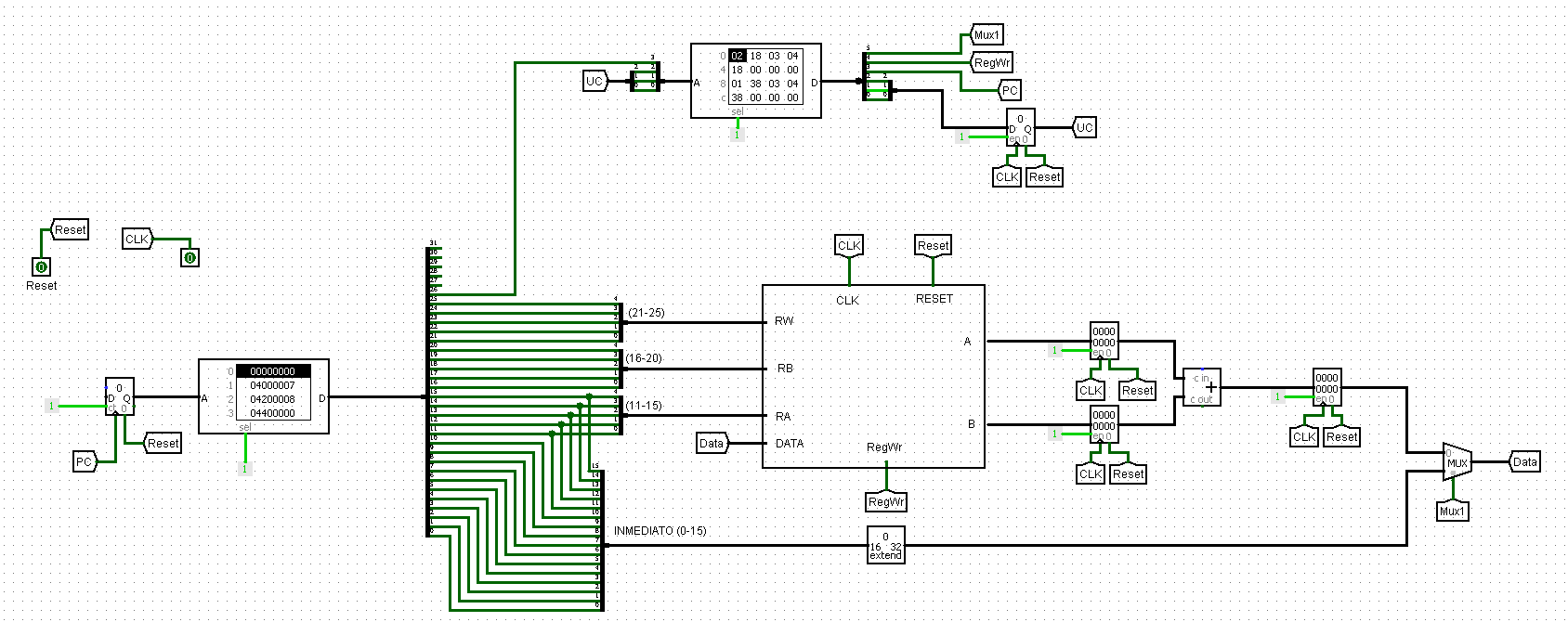


Ilustración 4: Procesador multiciclo

## 5.2 Unidad de control

Esta unidad de control es más compleja que en un procesador monociclo ya que habrá que tener en cuenta las diferentes señales de control necesarias en cada ciclo de la instrucción y el estado en el que nos encontramos. Para hacer un diagrama de estados hemos usado la herramienta Qsfm y quedaría de la siguiente manera:

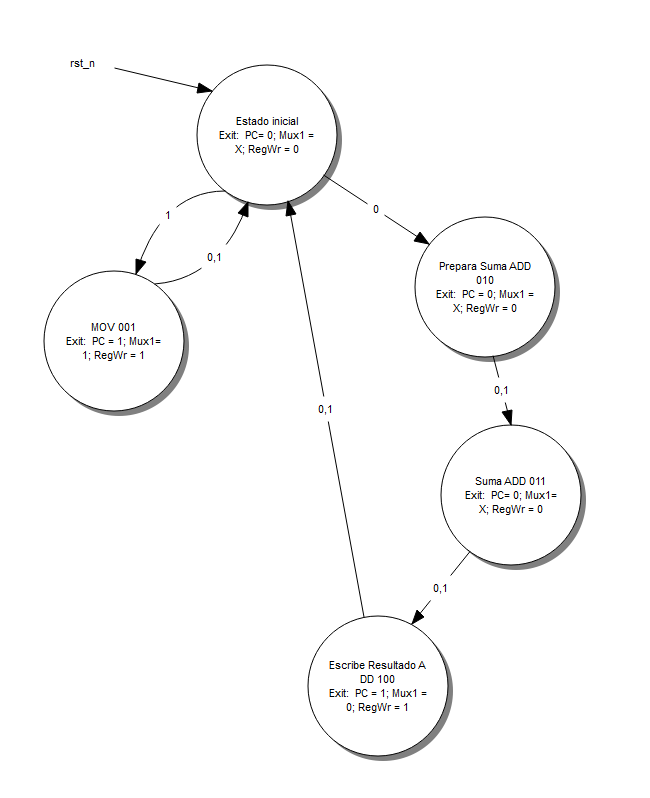


Ilustración 5: Diagrama estados Unidad control multiciclo

Una vez definido el diagrama nos disponemos a crear una tabla de verdad cuyas entradas son el estado donde nos encontramos y el código de operación. Como salida tenemos el siguiente estado y las señales de control necesarias en el ciclo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Entradas | | Salidas | | | |
| Código op | Estado Inicial | Mux1 | RegWr | Pc | Sig. Estado |
| 0 | 0 0 0 | X | 0 | 0 | 0 1 0 |
| 0 | 0 0 1 | 1 | 1 | 1 | 0 0 0 |
| 0 | 0 1 0 | X | 0 | 0 | 0 1 1 |
| 0 | 0 1 1 | X | 0 | 0 | 1 0 0 |
| 0 | 1 0 0 | 0 | 1 | 1 | 0 0 0 |
| 0 | 1 0 1 | X | X | X | X |
| 0 | 1 1 0 | X | X | X | X |
| 0 | 1 1 1 | X | X | X | X |
| 1 | 0 0 0 | X | 0 | 0 | 0 0 1 |
| 1 | 0 0 1 | 1 | 1 | 1 | 0 0 0 |
| 1 | 0 1 0 | X | 0 | 0 | 0 1 1 |
| 1 | 0 1 1 | X | 0 | 0 | 1 0 0 |
| 1 | 1 0 0 | 0 | 1 | 1 | 0 0 0 |
| 1 | 1 0 1 | X | X | X | X |
| 1 | 1 1 0 | X | X | X | X |
| 1 | 1 1 1 | X | X | X | X |

## 5.3 Análisis temporal

Para analizar correctamente este procesador habrá que calcular el tiempo que tarda cada estado en ejecutarse y coger el que tarde más:

* Mov:
* 1ºCiclo: Delay contador + Delay ROM + Ext.Sig + Mux2:1crtl + TsetupBR32

+ UC + Mux2:1crtl + TsetupBR32

Tiempo primer camino = 75ps + 95ps + 10ps + 75ps + 101ps = 356ps

Tiempo segundo camino = 75ps + 95ps + 95ps + 75ps + 101ps = 441ps

* ADD:
* 1ºCiclo: Delay contador + Delay ROM + Delay BR32 + Tsetup Registro Temporal

75ps + 95ps + 184ps + 30ps = 384ps

* 2ºCiclo: Delay Registro Temporal + Delay Sumador + Tsetup Registro Temporal

70ps + 180ps + 30ps = 280ps

* 3ºCiclo: UC + Mux2:1ctrl + Tsetup BR32

Delay Registro Temporal + Mux2:1ctrl + Tsetup BR32

Primer camino = 95ps + 75ps + 101ps = 271ps

Segundo camino = 70ps + 75ps + 101ps = 246ps

Una vez hemos calculado todos los estados nos damos cuenta que el camino crítico es el segundo camino de la instrucción Mov. Por lo tanto, el camino crítico es: 441ps

El tiempo de ciclo de esta máquina es: 2.267.573.696 =

En cuanto al tiempo tardado por el procesador multiciclo en ejecutar el programa es:

Tiempo ciclo \* número de ciclos \* número instrucciones =

6 instrucciones Mov y 6 instrucciones ADD =

(441 \*6\*1) + (441\*6\*3) = 7.938ps

Ahora vamos a comparar los dos procesadores:

P(monociclo) =

P(multiciclo) =

N =

El procesador multiciclo es un 5,076% más rápido que el procesador monociclo.

# 6.Conclusión

Un procesador multiciclo recorta el tiempo de ciclo pero aumenta el número de ciclos por instrucción por lo cual en algunos casos puede ser mejor que el procesador monociclo y en otros casos peor.