



**UNIVERSIDAD DE
GUADALAJARA**
Red Universitaria de Jalisco



ING. en Computación

**Seminario de Solución de Problemas de Arquitectura de
Computadoras "D12"**

Profesor: Jorge Ernesto López Arce Delgado



***PROYECTO FINAL: PROGRAMACIÓN EN VERILOG Y LENGUAJE
ENSAMBLADOR***

Equipo:

Estrada Huerta Félix Eduardo

Ortega Morales Juan José

González Ramírez Carlos Arturo

**01 - Junio - 2021.
Guadalajara, Jalisco**

Introducción

CARACTERISTICAS GENERALES DEL PROCESADOR MIPS DE 32 BITS

MIPS (Microprocessor without Interlocked Pipeline Stages), se conoce como MIPS a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

El origen de la arquitectura MIPS se remonta al año 1981. En la universidad de Stanford, donde un equipo liderado por John L. Hennessy comienza a trabajar con lo que sería el primer procesador MIPS. Con la idea de mejorar a gran escala el rendimiento de la máquina a través del uso de la segmentación .

Sintetiza las principales características de la arquitectura RISC, es una arquitectura simple y eficiente.

Un procesador MIPS es toda una familia de microprocesadores de arquitectura RISC. Es utilizado en Windows CE, routers Cisco, en la Nintendo 64, PlayStation, PlayStation 2 y PlayStation Portable.

Organización de un MIPS

- Unidad Aritmética y Lógica (ALU).
- Unidad Aritmética entera, operaciones de multiplicación y división.
- Unidad Punto Flotante.
- Coprocesador dedicado al manejo de memoria caché y virtual.

La arquitectura MIPS requiere que el software implemente algunas limitaciones en el diseño que está normalmente considerado parte de la implementación del hardware. Este papel presenta resultados experimentales en la efectividad de este procesador como un programa anfitrión.

Registros

Tiene 32 registros de 32 bits de propósito general (GPR)

32 registros de 32 bits de punto flotante (FPR)

Cuenta con instrucciones diferenciadas para GPR y FPR

Tipos de datos

Half Word (16 bits)

Word (32 bits)

Double Word (64 bits)

Simple precisión (32 bits)

Doble precisión (64 bits)

Los datos half Word y Word se cargan en GPRs y se completan con 0 o el signo

Las direcciones de memoria para cada instrucción son múltiplos de 4.

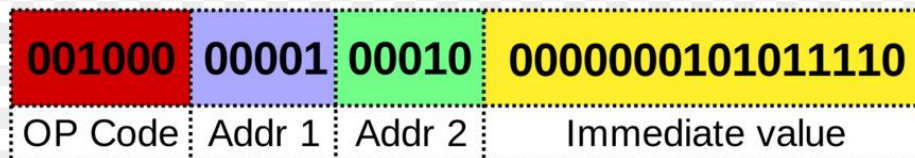
SET DE INSTRUCCIONES

Es una especificación que detalla las instrucciones que una CPU puede entender y ejecutar, o el conjunto de todos los comandos implementados por un diseño particular de una CPU. El término describe los aspectos del procesador generalmente visibles para un programador, incluidos los tipos de datos nativos, las instrucciones, los registros, la arquitectura de memoria y las interrupciones, entre otros aspectos.

Existen 3 principales tipos de set de instrucciones: CISC, RISC y SISC.

La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la microarquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Entre estos elementos se encuentran las microinstrucciones y los sistemas de caché.

MIPS32 Add Immediate Instruction



Equivalent mnemonic: **addi** \$r1, \$r2, 350

TIPOS DE INSTRUCCIONES MIPS

Formato R

Utilizado por las instrucciones aritméticas y lógicas

Tipo R (shamt: <i>shift amount</i> en instrucciones de desplazamiento)	Cód. Op.	Registro fuente 1	Registro fuente 2	Registro destino	Funct	
	xxxxxx	rs	rt	rd	shamt	funct
	6 31-26	5 25-21	5 20-16	5 15-11	5 10-6	6 5-0

Formato I

Utilizado por las instrucciones de transferencia, las de salto condicional y las instrucciones con operando inmediatos

Tipo I (carga o almacenamiento, ramificación condicional)	Cód. Op.	Registro base	Registro destino	Desplazamiento
	xxxxxx	rs	rt	Inmediato
	6 31-26	5 25-21	5 20-16	16 15-0

Formato J

Utilizado por las instrucciones de bifurcación

Tipo J (salto incondicional)	Cód. Op.	Dirección destino	
	xxxxxx	dirección	
	6 31-26	26 25-0	

TABLA DE INSTRUCCIONES MIPS

<i>Instrucción</i>	<i>Tipo</i>	<i>Sintaxis</i>	<i>Descripción</i>	<i>Tabla Info</i>
Add	R	Add \$rd, \$rs, \$rt	Suma Palabra (Add Word)	∞
Addi	I	Addi \$rd, \$ro1, \$inm	Suma Palabra Inmediata (Add Word Immediate)	
Sub	R	Sub \$rd, \$ro1, \$ro2	Resta palabra (Subtract Word)	ℓ
Mul	R	Mul \$rd, \$ro1, \$ro2	Multiplica palabra (Multiply Word)	Ω
Div	R	Div \$ro1, \$ro2	Divide Palabra (Divide Word)	∂
Or	R	Or \$rd, \$ro1, \$ro2	O (Or)	Δ
Ori	I	Ori \$rd, \$ro1, \$inm	O Inmediato (Or Immediate)	
And	R	And \$rd, \$ro1, \$ro2	Y (And), Pone el registro	Σ
Andi	I	Addi \$rd, \$ro1, \$inm	Y Inmediato (And immediate)	
slt	R	slt \$rd, \$ro1, \$ro2	Activa si menor (Set on Less Than)	∫
Slti	I	slti \$rd, \$ro, \$inm	Activa si menor inmediato (Set on Less Than Immediate)	
not	R	not \$rd, \$ro	No (Not)	∩
Lw	I	Lw \$rd, \$dir	Carga Palabra (Load Word)	℥
Sw	I	Sw \$ro, \$dir	Almacena Palabra (Store Word)	Ⓢ
beq	I	beq \$ro1, \$ro2, \$setiq	Bifurcación si igual (Branch on equal)	●
bne	I	bne \$ro1, \$ro2, \$setiq	Bifurcación si no igual (Branch on not equal)	
bgtz	I	bgtz \$ro, \$setiq	Bifurcación si mayor que cero (Branch on greater or equal than zero)	○
j	J	j \$setiq	Salto (Jump),	◇

Simbología	Significado
∞	Suma el valor del registro ro1 y el registro ro2 (o el valor inmediato de 16 bits inm) y deja el resultado en el registro rd. Se consideran los operandos en complemento a 2, y cuando se produce un desbordamiento no se modifica el registro rd y se provoca una excepción del procesador
ℓ	Pone en el registro rd el resultado de calcular la resta entre los valores de los registros ro1 y ro2. Si la instrucción se consideran los operadores en complemento a 2, y cuando se produce desbordamiento no se modifica el registro rd y se provoca una excepción del procesador
Ω	Multiplica el valor de los registros ro1 y ro2 y deja los 32 bits menos significativos del resultado en el registro rd. Las instrucciones mul realizan multiplicación en complemento a 2.
∂	Divide el valor del registro ro1 entre el valor del registro ro2. Ambos valores se consideran cantidades enteras con signo. El cociente de la división se almacena en el registro L0 y el resto de la división en el registro HI. Esta instrucción nunca provoca una excepción
Δ	Pone en el registro rd el resultado de realizar una operación or a nivel de bit entre los registros ro1 y ro2. En el caso de la instrucción ori entre el registro ro y el valor de 16 bits inm extendido con ceros
Σ	Pone en el registro rd el resultado de realizar una operación and a nivel de bit entre los registros ro1 y ro2 (en caso de la instrucción and) o entre el registro ro y el valor de 16 bits inm extendido con ceros (Instrucción Andi)
\int	Pone el valor 1 en el registro rd, si los valores de los registros ro1 y ro2 cumplen la correspondiente condición. En otro caso pone el registro rd a cero. Si la comparación implica algún tipo de orden se considera que los números se interpretan en complemento a 2.
\cap	Pone el registro rd el resultado de realizar una operación not a nivel de bit sobre el registro ro
\mathcal{V}	Carga el valor de 4 bytes almacenado a partir de la posición de memoria dir en el registro rd. Si la dirección dir no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción
\mathcal{W}	Almacena a partir de la posición de memoria dir el valor de 4 bytes que se encuentra en el registro ro. Si la dirección dir no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepción.
●	Salta, si los valores de los registro ro1 y ro2 cumplen una relación, a la instrucción marcada con etiq
○	Salta, si el valor del registro ro cumple una relación con respecto al valor cero, a la instrucción marcada con etiq
◇	Salta a la dirección marcada con etiq

OBJETIVOS

El objetivo principal del proyecto es en general, la implementación de cada uno de los temas aprendidos a lo largo de la materia de Seminario de Arquitectura de Computadoras, por lo cual se deberá realizar un Data Path con los conocimientos adquiridos del mismo.

El resultado de la realización de este proyecto también lleva como objetivo la elaboración y utilización de un programa en lenguaje ensamblador, esto con el fin de comprender como es que la computadora funciona de una manera lógica a lo largo de cada uno de los circuitos del mismo, así como el uso de sus respectivas compuertas lógicas que esta misma utiliza. Dando así un conocimiento básico y general de las composiciones, funciones y procesos que lleva una parte de un computador a nivel básico en hardware.

DESARROLLO DE DATA PATH

Una Data Path es una colección de unidades funcionales como unidades lógicas aritméticas o multiplicadores que realizan operaciones de procesamiento de datos, registros y buses.

Junto con la unidad de control, compone la unidad central de procesamiento (CPU). Se puede crear una ruta de datos más grande uniendo más de una ruta de datos utilizando multiplexores. Una ruta de datos es la ALU, el conjunto de registros y los buses internos de la CPU que permiten que los datos fluyan entre ellos.

Una ruta de datos de microarquitectura organizada alrededor de un solo bus, el diseño más simple de una CPU utiliza un bus interno común. La utilización eficiente requiere una estructura de tres buses internos un poco más complicada. Muchas CPU relativamente simples tienen un archivo de registro de 2 lecturas y 1 escritura conectado a las 2 entradas y 1 salida de la ALU.

A fines de la década de 1990, hubo una creciente investigación en el área de las rutas de datos reconfigurables (rutas de datos que se pueden reutilizar en tiempo de ejecución utilizando una estructura programable), ya que dichos diseños pueden permitir un procesamiento más eficiente y ahorros de energía sustanciales.

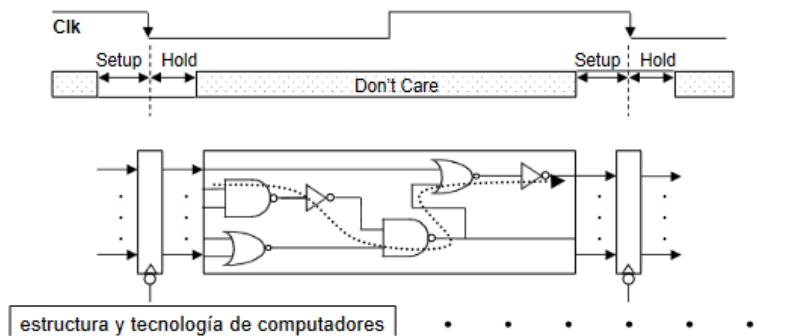
Temporización Monociclo

Ejecución típica (de una instrucción)

- Todos los registros se cargan simultáneamente (de modo selectivo)
- Todos los valores se propagan a través de las redes combinatoriales hasta estabilizarse en las entradas de los registros
- Se repite indefinidamente el proceso

Todos los elementos de almacenamiento están sincronizados al mismo flanco de reloj:

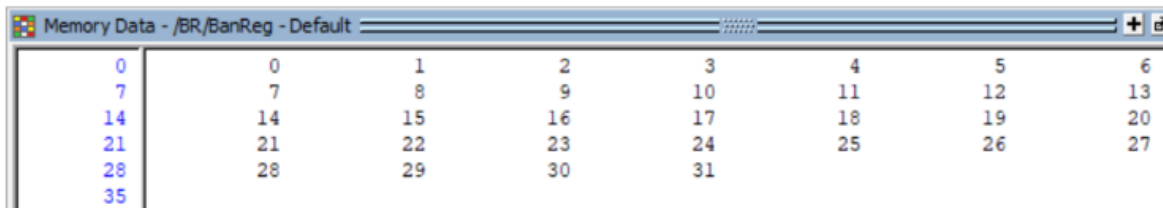
- Tiempo de ciclo = CLK-to-Q + Camino con retardo máximo + Setup + Clock Skew
- $(\text{CLK-to-Q} + \text{Camino con retardo mínimo} - \text{Clock skew}) > \text{Hold}$



REPORTE DE PROGRAMA EN VERILOG

FASE I:

Durante la fase 1 se pidió un módulo que fuera capaz de realizar instrucciones de tipo R, para comprobar su correcto funcionamiento, a continuación, se muestra la forma en que trabaja nuestro módulo.



0	0	1	2	3	4	5	6
7	7	8	9	10	11	12	13
14	14	15	16	17	18	19	20
21	21	22	23	24	25	26	27
28	28	29	30	31			
35							

Figura 1

En la figura 1 podemos ver los datos precargados del banco de registros, estos datos servirán como operadores para las operaciones de tipo R.

1	00000000	13	00000000
2	00000001	14	00000001
3	10000000	15	10010000
4	00100000	16	00100100
5	00000000	17	00000000
6	01000001	18	00000001
7	10001000	19	10011000
8	00100010	20	00100101
9	00000000	21	00000000
10	00000000	22	00000001
11	00000000	23	10100000
12	00000000	24	00101010

Figura 2

En la figura 2 tenemos las instrucciones que se van a realizar. La primera es una suma de los registros 0 y 1, el resultado se almacena en la dirección 16, la próxima instrucción es una resta de los registros 2 y 1, el resultado se almacena en la dirección 17, la próxima es Nop, se almacena un 0 en la dirección 0, luego sigue una instrucción AND entre los registros 0 y 1 y el resultado se almacena en la dirección 18, después una instrucción or entre los registros 0 y 1, el resultado se almacena en la dirección 19, por último tenemos una operación slt con los registros 0 y 1, el resultado se almacena en la dirección 20.

Operación	Rs y dato	Rt y dato	Rd	Resultado
100000(suma)	00000 (0)	00001 (1)	10000 = 16	1
100010(resta)	00010 (2)	00001 (1)	10001 = 17	1
000000(Nop)	00000 (0)	00001 (1)	00000 = 0	0
100100(and)	00000(0)	00001 (1)	10010 = 18	0
100101(or)	00000(0)	00001 (1)	10011 = 19	1
101010(slt)	00000(0)	00001 (1)	10100 = 20	1

Tabla 1

En la tabla 1 podemos observar cuales son los resultados esperados de todas las operaciones que describí anteriormente.

Register	Value
0	0
7	7
14	14
21	21
28	28
35	28
1	1
8	8
15	15
22	22
29	29
2	2
9	9
16	1
23	23
30	30
3	3
10	10
17	1
24	24
31	31
4	4
11	11
18	0
25	25
5	5
12	12
19	1
26	26
6	6
13	13
20	1
27	27

Figura 3

En la figura 3 podemos ver los nuevos datos almacenados en el banco de registros, los cuales coinciden con los resultados esperados para cada una de las operaciones.

Banco de Registros:

Registro	Valor	Instrucción
20	3	SUM
21	1	SUB
24	0	AND
25	10	OR
26	1	SLT
0	0	NOP

Existe un error en una instrucción del archivo "TestF1_MemInst.mem" debe identificarlo, anotar en el reporte y modificarlo para su correcto funcionamiento

FASE II:

En la fase 2 del proyecto se implemento el uso de instrucciones de tipo I, instrucciones de sw, lw y beq en nuestro módulo creado en la fase 1, se agregó cuatro buffers para el uso de algunas señales (con las que así se requiere) lleguen al mismo tiempo.

La creación de estos módulos no fue de gran dificultad, la implementación de otro tipo de operaciones en el código en opinión personal, fue uno de los retos más complicados de realizar, al igual que cada una de las interconexiones de los módulos y el asegurar el correcto funcionamiento y traslado de los bits, así evitando una gran cantidad de errores y perdidas de datos.

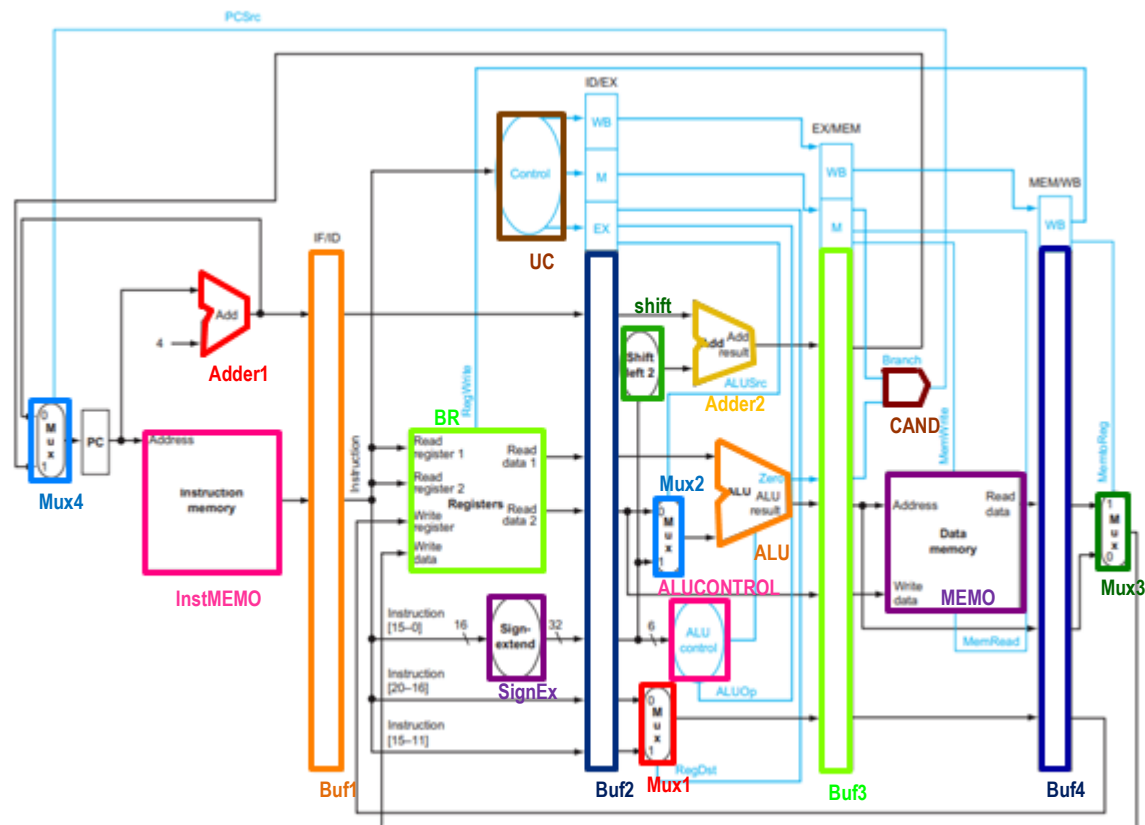
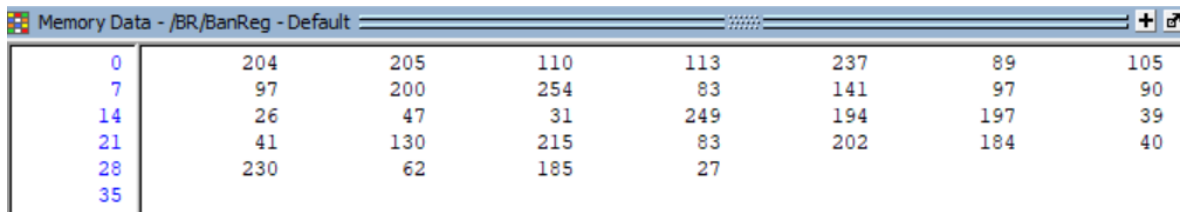


Figura 2

En la figura 1 podemos ver el módulo completo y los nombres de cada módulo que usamos en nuestro proyecto, sólo para que sea más fácil entender el código al momento de verlo. El módulo que contiene todos estos módulos mostrados anteriormente tiene por nombre PF1 y el testbench se llama TB_Prueba.

Para esta fase del proyecto, primero se necesito terminar el código de la fase 1 que no se termino por completo, cuando se terminó la implementación comprobé su funcionalidad con instrucciones de tipo R, se inicio agregando la instrucción Beq en el código, cuando se logro hacerlo funcionar esta instrucción en el módulo se añadio las instrucciones lw y sw, después de eso implementó las instrucciones de tipo I (addi, andi, slti, ori) y se volvió a probar todas estas instrucciones una por una en el código, posteriormente se añadieron los buffers y reconectaron todos los módulos de la forma que se muestra en la figura 1.

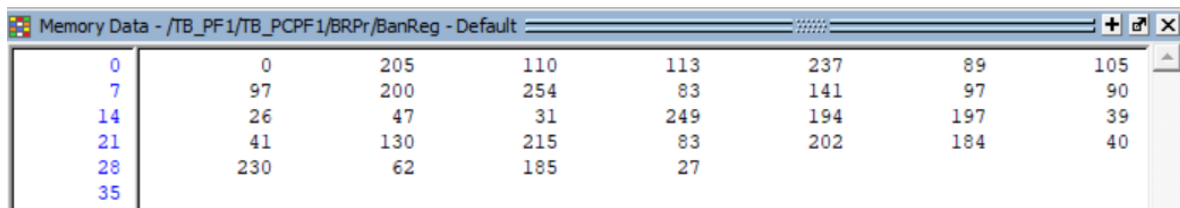
Cuando el módulo estaba listo nuevamente se realizaron las pruebas de las instrucciones individualmente, estos fueron los resultados:



Register	0	7	14	21	28	35
0	204	205	110	113	237	89
7	97	200	254	83	141	97
14	26	47	31	249	194	197
21	41	130	215	83	202	184
28	230	62	185	27		
35						

Figura 3

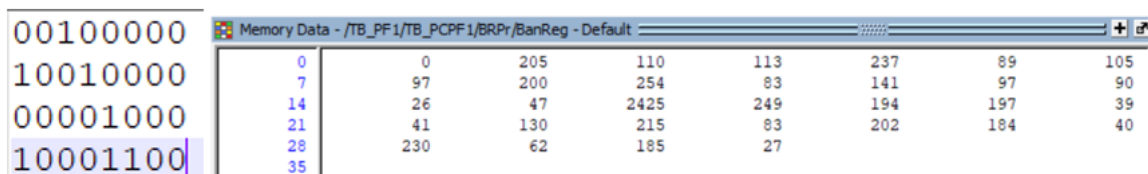
Primero, en la figura 2 podemos ver los datos precargados en el banco de registros.



Register	0	7	14	21	28	35
0	0	205	110	113	237	89
7	97	200	254	83	141	97
14	26	47	31	249	194	197
21	41	130	215	83	202	184
28	230	62	185	27		
35						

Figura 4

La primera instrucción que probé fue Nop, en la figura 3 podemos ver que no se realiza operación alguna y se almacena un 0 en la dirección 0.



Register	0	7	14	21	28	35
0	0	205	110	113	237	89
7	97	200	254	83	141	97
14	26	47	2425	249	194	197
21	41	130	215	83	202	184
28	230	62	185	27		
35						

Figura 5

La siguiente instrucción fue addi, en la figura 4 se muestra la instrucción y el banco de registros con el nuevo dato. En la instrucción se indica primero la operación a realizar (010000), luego el registro operando (00100), luego el registro destino (10000) y por último 0000100010001100 que es el valor inmediato que se va a sumar.

0	0	205	110	113	237	89	105
7	97	200	254	83	141	97	90
14	26	47	31	1	194	197	39
21	41	130	215	83	202	184	40
28	230	62	185	27			
35							

Figura 6

La siguiente fue SLti, en la figura 5 podemos ver que el resultado de la operación se almacena en la dirección 17, el código es el siguiente: 001010 (para la operación), 00101 (para operando 1), 10001 (dirección destino) y 0011100111100000 como valor inmediato, al ser menor el operando 1 que el valor inmediato, se almacena un 1.

0	0	205	110	113	237	89	105
7	97	200	254	83	141	97	90
14	26	47	31	249	240	197	39
21	41	130	215	83	202	184	40
28	230	62	185	27			
35							

Figura 7

[illegible]

Memory Data - /TB_PF1/TB_PCPF1/BRPr/BanReg - Default							
0	0	205	110	113	237	89	105
7	97	200	254	83	141	97	90
14	26	47	31	249	194	14719	39
21	41	130	215	83	202	184	40
28	230	62	185	27			
35							

Figura 8

La próxima instrucción es ori, el resultado se almacena en la dirección 19 (ver figura 7), el operando 1 es el contenido de la dirección 15 del banco de registros (000000000000000000000000101111) y el otro operando es el inmediato 0011100101010011.

Memory Data - /TB_PF1/TB_PCPF1/MEMOPr/mem							
0	220	115	157	67	249	213	64
7	49	183	99	150	46	251	69
14	78	94	163	227	114	209	6
21	196	172	48	248	19	217	191
28	85	212	16	207			
35							

Figura 9

Como vamos a ver las instrucciones lw y sw, en la figura 8 podemos ver los datos que hay precargados en nuestra memoria de datos.

0	0	205	110	113	64	89	105
7	97	200	254	83	141	97	90
14	26	47	31	249	194	197	39
21	41	130	215	83	202	184	40
28	230	62	185	27			
35							

Figura 10

En la figura 9 podemos ver la operación lw, la dirección base es 0, la dirección destino es la 4 y offset es igual a 6, entonces, el dato que se almacena en el banco de registros en la dirección 4 es el dato que se encuentra en la dirección 6 de la memoria de datos.

0	220	200	157	67	249	213	64
7	49	183	99	150	46	251	69
14	78	94	163	227	114	209	6
21	196	172	48	248	19	217	191
28	85	212	16	207			
35							

Figura 11

La siguiente operación es sw, se muestra en la figura 10, la dirección base también es la 0, la dirección del banco de registros cuyo valor se va a guardar en la memoria de datos es la 8, el valor se va a guardar en la dirección 1.

0	0	205	110	113	237	89	105
7	97	200	254	83	141	97	90
14	26	47	2425	1	240	14719	39
21	41	130	215	83	202	184	40
28	230	62	185	27			
35							

Figura 12

0	220	200	157	67	249	213	64
7	49	183	99	150	46	251	69
14	78	94	163	227	114	209	6
21	196	172	48	248	19	217	191
28	85	212	16	207			
35							

Figura 13

La última operación es Beq, en la figura 11 podemos ver que se realizaron todas las instrucciones que expliqué de forma secuencial, la primera instrucción que agregué en esta simulación fue un beq que va a comparar el registro 0 con el 0 y como son iguales se va a saltar dos instrucciones, estas dos instrucciones que se va a saltar tendrían que almacenarse en las direcciones 1 y 2 (en caso de que no se saltaran) y como se puede ver en la figura, los datos en estas dos direcciones permanecen iguales que al principio. La figura 12 muestra la memoria de datos para comprobar que se realizó también la operación sw.

CONCLUSIÓN DE FASE II

La realización de esta tarea fue algo compleja y complicada, en especial por el hecho de tener el tiempo contado al tener el atraso de la fase anterior, ya que se necesitaba complementar y lograr funcionar el código al 100%, no fue sencillo ni complejo, ya que todos tenemos una idea base de lo que se necesita realizar en cada uno de los módulos, pero al mismo tiempo, el plasmar las ideas es un tema complicado.

Al terminar la fase 1, no se perdió el tiempo y directamente se comenzó a realizar la implementación de la fase 2, la cual en opinión personal fue de las más complicadas, ya que a diferencia de la fase 1 se necesitaba realizar la implementación de mayor cantidad de módulos, la mayoría de las de los errores que se encontraron en el desarrollo de este proyecto fueron al momento de realizar las interconexiones de los módulos, fue un gran reto, pero al final se logró terminar con buenos resultados. Dejando mucho aprendizaje del mismo.

FASE III:

Para la fase 3 de nuestro proyecto se nos pidió que implementáramos los módulos correspondientes para poder realizar instrucciones de tipo jump. Las modificaciones que tuvimos que hacer en el código de nuestro módulo completo fue agregar un multiplexor, al que llamé multiplexor 5, otro módulo de shift left y las salidas correspondientes de la unidad de control, que fue sólo una salida llamada jump.

```
//jump
6'b000010:
begin
    RegDst = 1'b0;
    Branch = 1'b0;
    MemRead = 1'b0;
    MemToReg = 1'b0;
    ALUOp = 3'b000;
    MemToWrite = 1'b0;
    ALUSrc = 1'b0;
    RegWrite = 1'b0;
    jump = 1'b1;
end
```

Figura 14

En la figura 1 podemos ver el fragmento de código que se le agregó al módulo unidad de control para que se lleven a cabo las instrucciones de tipo jump, se agregó un nuevo caso para la variable opcode, esto quiere decir que al recibir el opcode "000010", la unidad de control enviará las señales que vemos en la figura 1 y se realizará el jump. Cabe aclarar que aunque aquí no se vea, se añadió un output llamado jump al módulo.

El nuevo módulo de shift left fue agregado después del buffer 1, recibe 26 bits que son los primeros bits de la instrucción (del bit 25 al bit 0), y la salida del shift left es de 28 bits que entran al buffer 2, en el buffer 2 se agregó una entrada para estos 28 bits, otra entrada para la señal de jump y una salida de 32 bits, esta salida esta dada por los 28 bits del shift left concatenados con 4 bits (del bit 31 al bit 28) de la salida del sumador 1. Esta salida de 32 bits es la dirección a la que se hará el jump. La señal de jump entra desde la unidad de control y sale hacia el próximo buffer.

En el buffer 3 también se agregaron dos entradas, una de 32 bits y otra para la señal de jump, también dos pares de salidas para la dirección de jump y la señal de jump, que son de 32 bits y 1 bit respectivamente. Estas dos salidas ya no pasan al próximo buffer, las dos se van al multiplexor 5.

En el módulo multiplexor 5 entra la dirección de jump y entra también el valor de salida del multiplexor 4, es decir, el multiplexor 4 ya no está conectado directamente al PC, ahora está conectado el multiplexor 5, entonces si la señal de jump está “encendida”, sale la dirección del jump (y entra al PC) y si está en 0, sale la dirección de la próxima instrucción (que viene de mux 4).

Esto es básicamente, una breve descripción de los cambios que se tuvieron que hacer en el módulo de la fase 2 para que se pudieran realizar instrucciones de tipo jump.

Para comprobar que funciona, probamos el módulo con una serie de instrucciones que son las instrucciones que se requieren para realizar la sucesión de Fibonacci.

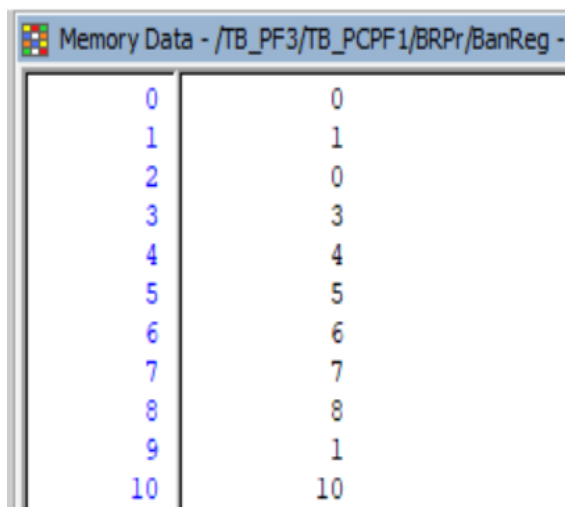
1	00000000	En la figura 2 podemos ver las instrucciones que se probaron, primero tenemos una suma de los registros 0 y 1 que se almacena en el registro 2, la próxima instrucción es un addi, pero lo único que hace es guardar el valor del registro 1 en el registro 0, la próxima también es un addi que guarda el valor del registro 2 en el registro 1, la próxima instrucción aumenta en 1 el registro que funciona como contador, la próxima es un beq que compara el registro contador (que es el 9) con el registro 10, cuando el valor del registro 9 se haya aumentado 9 veces se saltará una instrucción y por último tenemos la operación de jump, que regresa a la dirección 0.
2	00000001	
3	00010000	
4	00100000	
5	00100000	
6	00100000	
7	00000000	
8	00000000	
9	00100000	
10	01000001	
11	00000000	
12	00000000	
13	00100001	
14	00101001	
15	00000000	
16	00000001	
17	00010001	
18	00101010	
19	00000000	
20	00000001	
21	00001000	
22	00000000	
23	00000000	
24	00000000	

Figura 15

Entonces los resultados esperados son los siguientes, que podemos ver en nuestra tabla 1.

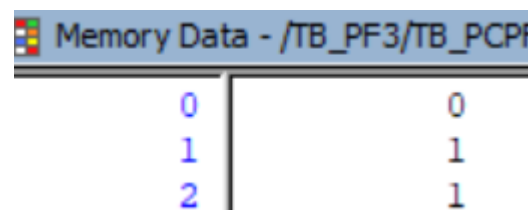
<i>Vuelta</i>	<i>add \$0, \$1, \$2</i>	<i>addi \$1, \$0, #0</i>	<i>addi \$2, \$1, #0</i>	<i>addi \$9, \$9, #1</i>	<i>beq \$9, \$10, #1</i>	<i>Jump #0</i>
1	\$2 = 1	\$0 = 1	\$1 = 1	\$9 = 2	No	Sí
2	\$2 = 2	\$0 = 1	\$1 = 2	\$9 = 3	No	Sí
3	\$2 = 3	\$0 = 2	\$1 = 3	\$9 = 4	No	Sí
4	\$2 = 5	\$0 = 3	\$1 = 5	\$9 = 5	No	Sí
5	\$2 = 8	\$0 = 5	\$1 = 8	\$9 = 6	no	Sí
6	\$2 = 13	\$0 = 8	\$1 = 13	\$9 = 7	No	Sí
7	\$2 = 21	\$0 = 13	\$1 = 21	\$9 = 8	No	Sí
8	\$2 = 34	\$0 = 21	\$1 = 34	\$9 = 9	No	Sí
9	\$2 = 55	\$0 = 34	\$1 = 55	\$9 = 10	Sí	No

Tabla 1



0	0
1	1
2	0
3	3
4	4
5	5
6	6
7	7
8	8
9	1
10	10

Figura 16

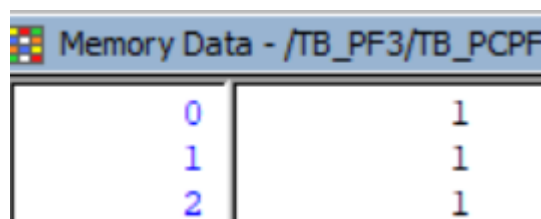


0	0
1	1
2	1

Figura 4

Al correrlo por primera vez sólo 400 nanosegundos, podemos ver los valores iniciales del banco de registros en la figura 3. Como los valores que nos interesan se encuentran entre los registros 0 y 10, sólo muestro los registros del 0 al 10.

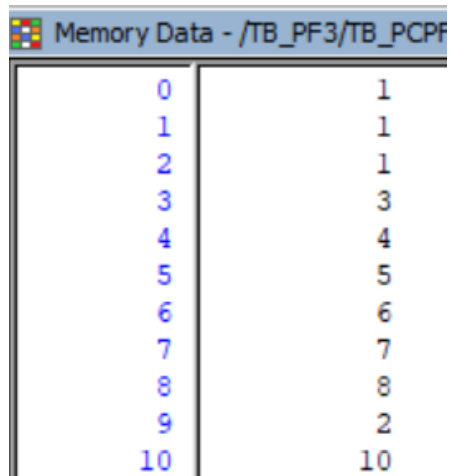
En la figura 4 podemos ver que después de correrlo una vez más (400 nanosegundos), se guarda la suma de los registros 0 y 1 en el registro 2.



0	1
1	1
2	1

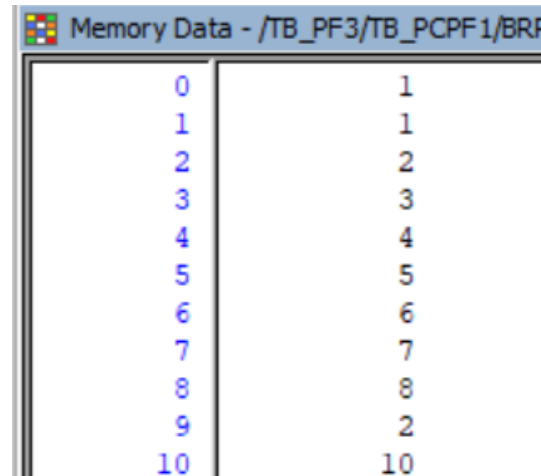
Figura 5

En la figura 5 podemos ver que también ya se cambiaron los valores de los registros 0 y 1, el registro 0 tomó el valor que tenía el registro 1 y el registro 1 tomó el valor del registro 2, esto sucede al correrlo otras veces más.



0	1
1	1
1	1
2	1
3	3
4	4
5	5
6	6
7	7
8	8
9	2
10	10

Figura 17

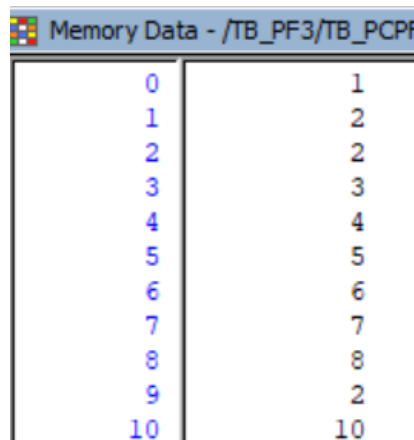


0	1
1	1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	2
10	10

Figura 18

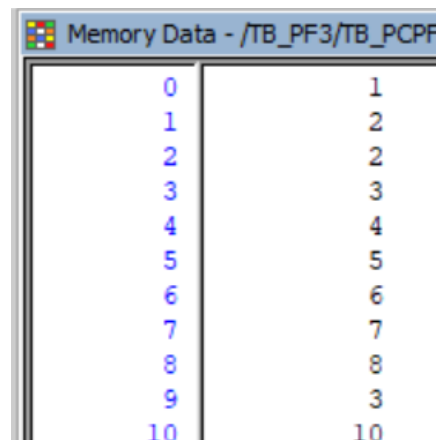
En la figura 6 podemos ver que al seguir corriendo el código se aumenta en 1 el valor del registro 9, hasta este momento no se ha realizado ninguna operación de jump o beq, en las próximas figuras podremos ver que la instrucción de jump se lleva a cabo exitosamente y se vuelven a sumar los valores de los registros 0 y 1.

En la figura 7 podemos ver que ya se realizó el jump y se volvieron a sumar los registros 0 y 1.



0	1
2	2
1	2
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	2
10	10

Figura 19



0	1
2	2
1	2
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	3
10	10

Figura 9

En la figura 8 ya tenemos los nuevos valores de los registros 0 y 1.

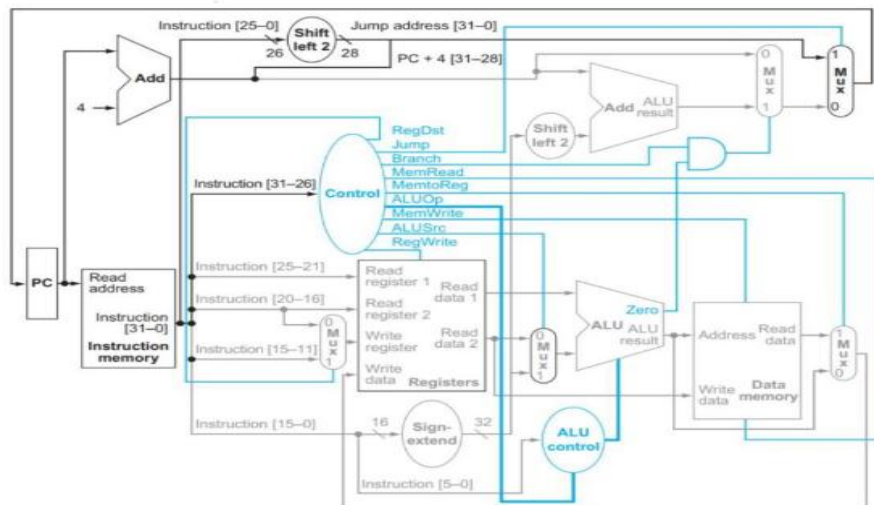
En la figura 9 se vuelve a aumentar el valor del registro 9 que nos va a servir como contador.

Memory Data - /TB_Pf3/TB_PCPF1/	
0	1
1	2
2	3
3	3
4	4
5	5
6	6
7	7
8	8
9	3
10	10

Figura 20

En la figura 10 se vuelve a saltar hasta la instrucción 0 y se vuelven a sumar los registros 0 y 1. Voy a dejar hasta aquí la demostración para no extenderme tanto en el espacio y en el tiempo.

Diagrama Final de Data Path:



CONCLUSIÓN DE FASE III

Fue una fase fácil y sencilla a comparación de la fase 2 donde tuvimos que añadir 4 módulos e interconectar todo otra vez. Esta fase fue sencilla porque solo fueron dos módulos y no hubo necesidad de añadir mucho código, así que fue fácil. Con estas nuevas instrucciones que añadimos podremos realizar nuestro algoritmo planeado.

La implementación del algoritmo de la sucesión de Fibonacci no tuvo complicación alguna, ya que al tener el código ensamblador disponible fue más fácil realizar la conversión para que funcionará ya que las instrucciones implementadas ya estaban en utilización en el programa de verilog.

PROGRAMA ENSAMBLADOR

El programa ensamblador que se llevará a cabo para la entrega final de este proyecto será la sucesión de Fibonacci, esto propuesto por el profesor ya que las propuestas del equipo eran muy complejas o complicadas por lo que se tardaría o atrasaría mucho para la realización del mismo, por lo cual se agradece al profesor por su atención hacia nosotros.

Sucesión De Fibonacci:

La sucesión o serie de Fibonacci hace referencia a la secuencia ordenada de números descrita por Leonardo de Pisa, matemático italiano del siglo XIII: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...**

A cada uno de los elementos de la serie se le conoce con el nombre de número de Fibonacci.

Esta sucesión fue descrita por Fibonacci como la solución a un problema de cría de conejos: *“Cierta hombre tiene una pareja de conejos juntos en un lugar cerrado y desea saber cuántos son creados a partir de este par en un año cuando, de acuerdo a su naturaleza, cada pareja necesita un mes para envejecer y cada mes posterior procrea otra pareja”* (Laurence Sigler, Fibonacci's Liber Abaci, página 404).

La secuencia de esta resolución anterior es la que sigue:

- Partimos de una pareja de conejos el primer mes.
- El segundo mes la pareja envejece pero no procrea.
- El tercer mes la pareja procrea otra pareja (es decir, ya tenemos dos parejas).
- El cuarto mes, la primera pareja vuelve a procrear y la pareja nueva envejece sin procrear (luego tenemos tres parejas).
- El quinto mes, las dos parejas más viejas vuelven a procrear mientras que la nueva pareja no procrea (cinco parejas en total)
- ...

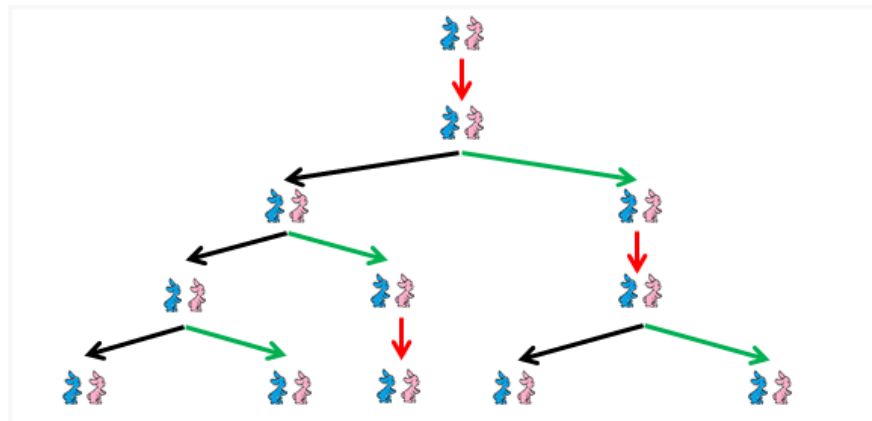


Figura 1

Simbología:

—> La pareja de conejos envejece.

—> La pareja de conejos envejece por primera vez (es por ello por lo que no puede procrear).

—> Procreación de la pareja de conejos.

Para realizar el cálculo de un elemento de la sucesión se realiza sumando el número anterior al número actual. representado a través de la siguiente función: $f_n = f_{n-1} + f_{n-2}$

Programa en Ensamblador:

Sucesión de Fibonacci	
Instrucciones en Ensamblador	Características y proceso de Instrucción
<i>addi \$9, \$9, #1</i>	Variable i del ciclo for del programa en c (i++)
<i>add \$4, \$0, \$1</i>	RESULT = n1 + n2
<i>addi \$0, \$1, #0</i>	n1 = n2
<i>addi \$1, \$4, #0</i>	n2 = RESULT
<i>beq \$9, \$10, #1</i>	Si \$9 == \$10
<i>j #17</i>	Saltar inicio de programa
<i>sw \$1, \$4, #0</i>	Guardar en la Memoria

br		
Dirección	Dato	Característica
\$0	0	n1
\$1	1	n2
\$2	345	-
\$3	8	-
\$4	8	Result
\$5	-200	-
\$6	745	-
\$7	654	-
\$8	5	-
\$9	0	temporal
\$10	12	comparativo

Memoria		
Dirección	Dato	Característica
\$0	645	
\$1	1	Guardado Final en Memoria
\$2	345	
\$3	8	
\$4	8	
\$5	-200	
\$6	745	
\$7	654	
\$8	5	
\$9	0	

Descripción de Código:

1. **addi \$9, \$9, #1:**

Realizamos una Suma inmediata de los valores que se encuentran en la dirección establecida (\$9), Sumarle 1 al Valor y guardarlo en la dirección asignada (\$9), este valor será utilizado como bandera comparativa para finalizar la operación de la sucesión, siendo la última posición obtenida de la sucesión.

2. **add \$4, \$0, \$1**

Se suman los Valores encontrados en las direcciones \$0(n1) y \$1(n2) y se Guardaran en la dirección asignada para el resultado (\$4).

3. **addi \$0, \$1, #0**

Se realizan el cambio y traslación de los valores a las posiciones de secuencia correspondiente (n1 = n2) para así evitar errores de sucesión

4. **addi \$1, \$4, #0**

Se realizan el cambio y traslación de los valores a las posiciones de secuencia correspondiente (n2 = resultado) para así evitar errores de sucesión

5. **beq \$9, \$10, #1**

Se realiza una comparación de los valores que se encuentran en las direcciones \$9 y \$10, que serian nuestras banderas de finalización de sucesión, siendo \$9 la bandera de aumento y \$10 la bandera de posición final de algoritmo. Si estos valores son iguales se saltará la instrucción siguiente. De lo contrario seguirá su secuencia.

6. **j #17**

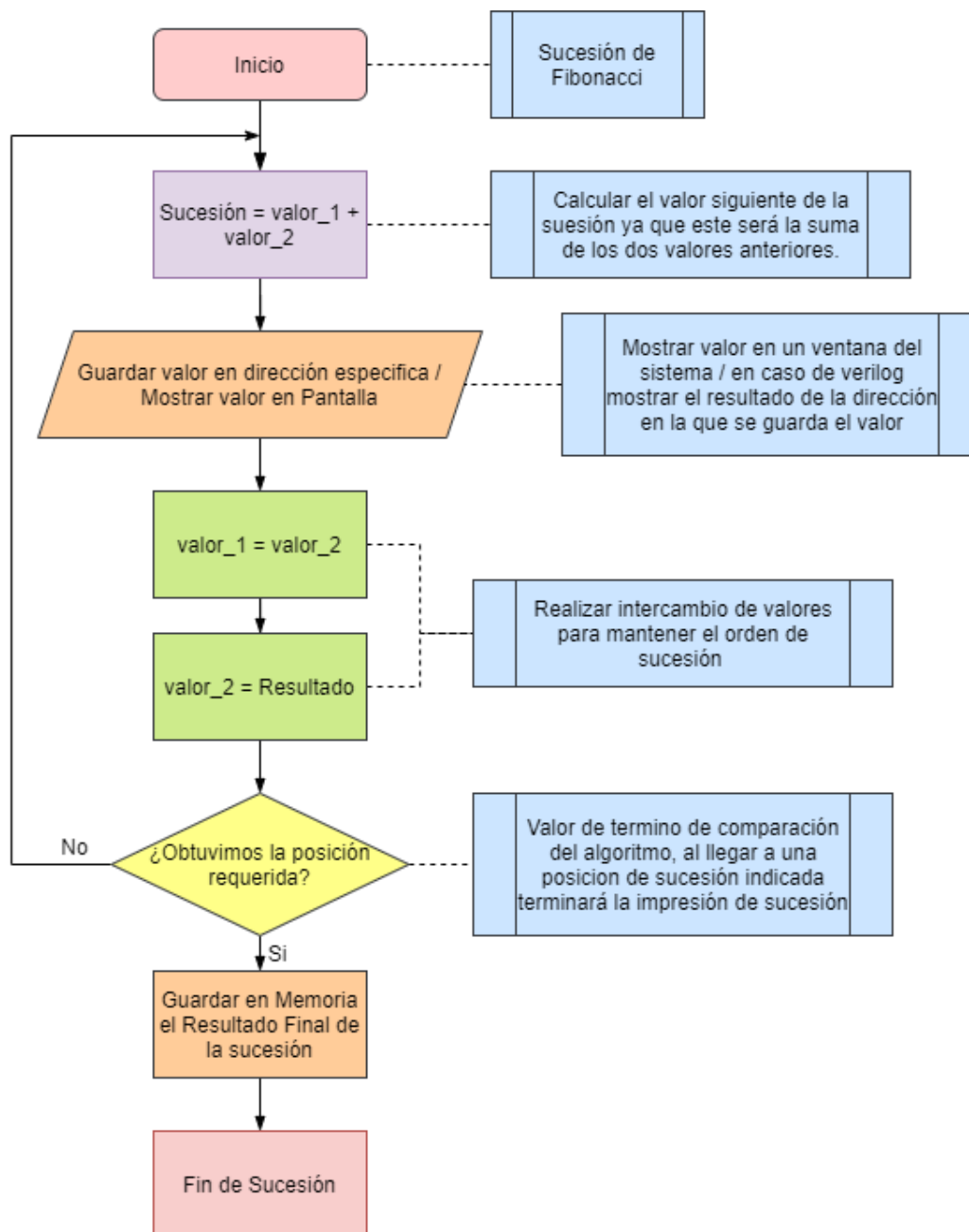
Instrucción de repetición o salto de líneas, esta instrucción nos regresará al comienzo del algoritmo para proseguir con la obtención de los valores siguientes de la sucesión de Fibonacci, Esta instrucción no se ejecutará solamente cuando la instrucción anterior sea verdadera, ya que la instrucción anterior la saltará, dando por terminado el Algoritmo

7. **sw \$1, \$4, #0**

Guardado del resultado final en Memoria en su dirección \$1, El resultado del algoritmo final se ubica en la dirección del branch \$4.

<i>Vuelta</i>	<i>add \$4, \$0, \$1</i>	<i>addi \$0, \$1, #0</i>	<i>addi \$1, \$0, #0</i>	<i>addi \$9, \$9, #1</i>	<i>beq \$9, \$10, #1</i>	<i>Jump #0</i>
1	\$4 = 1	\$0 = 1	\$1 = 1	\$9 = 2	No	Sí
2	\$4 = 2	\$0 = 1	\$1 = 2	\$9 = 3	No	Sí
3	\$4 = 3	\$0 = 2	\$1 = 3	\$9 = 4	No	Sí
4	\$4 = 5	\$0 = 3	\$1 = 5	\$9 = 5	No	Sí
5	\$4 = 8	\$0 = 5	\$1 = 8	\$9 = 6	no	Sí
6	\$4 = 13	\$0 = 8	\$1 = 13	\$9 = 7	No	Sí
7	\$4 = 21	\$0 = 13	\$1 = 21	\$9 = 8	No	Sí
8	\$4 = 34	\$0 = 21	\$1 = 34	\$9 = 9	No	Sí
9	\$4 = 55	\$0 = 34	\$1 = 55	\$9 = 10	Sí	No

Diagrama de Flujo de Algoritmo de la Sucesión de Fibonacci:



DECODIFICACIÓN

Para realizar la decodificación de las instrucciones en ensamblador, se realizó de la siguiente forma, cada una de las instrucciones en ensamblador se buscó su codificación en binario, realizando así el cambio de sistema de numeración.

Instrucciones MIPS Utilizados	
Binario	Instrucción
000000	ADD
001000	ADDI
000100	BEQ
000010	J
101011	SW

Instrucción en Ensamblador:

1. add \$0, \$1, \$2
2. addi \$1, \$0, #0
3. addi \$2, \$1, #0
4. addi \$9, \$9, #1
5. beq \$9, \$10, #1
6. jump #0
7. sw \$4, \$2, #0

Decodificación:

1. 00000000000000010001000000100000
2. 00100000001000000000000000000000
3. 00100000010000010000000000000000
4. 001000010010100100000000000000001
5. 000100010010101000000000000000001
6. 00001000000000000000000000000000
7. 10101100100000100000000000000000

CONCLUSIÓN DE PROYECTO

La implementación de cada uno de las fases para la correcta realización y funcionamiento del Data Path fue muy interesante y a su vez compleja, ya que en lo personal al momento de realizar la programación y codificación de cada uno de los sistemas, llegaba en un punto en el cual no tenía en claro los siguientes pasos a realizar, a su vez la información que existe de este tipo de programas era un poco escasa, pero el libro de texto proporcionado por el profesor fue de gran utilidad para la realización del mismo.

Al momento de llegar al programa ensamblador la implementación fue sencilla, ya que una vez teniendo el algoritmo a implementar, solo falta realizar la conversión correspondiente de valores para que el Data Path lo ejecutará sin problema alguno.

Para finalizar, la realización de este proyecto fue una gran experiencia y reto a superar, gracias a esto ahora conozco un poco más el funcionamiento de un computador y como maneja los datos de entrada, sin más por agregar doy por finalizado este Reporte.

REFERENCIAS

- José Manuel Mendías Cuadros. (2018). Diseño de la ruta de datos y la unidad de control unidad de control. 05 - Jun - 2021, de Dpto. Arquitectura de Computadores y Automática. Universidad Complutense de Madrid Sitio web: <http://www.fdi.ucm.es/profesor/mendias/512/docs/tema16.pdf>
- Robert B. Anderson. (2021). MIPS (procesador). 02 - Jun - 2021, de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/MIPS_\(procesador\)](https://es.wikipedia.org/wiki/MIPS_(procesador))
- Richard A. Smith. (2019). Datapath. 06 - Jun - 2021, de Wikipedia Sitio web: <https://en.wikipedia.org/wiki/Datapath>
- Cristian Tejedor García. (2020). ARQUITECTURA MIPS. 06 - Jun - 2021, de Universidad de Valladolid Sitio web: https://www.infor.uva.es/~bastida/OC/TRABAJO2_MIPS.pdf
- John R. Espinoza. (2018). Arquitectura MIPS: Formato de la instrucción máquina Ar. 04 - Jun - 2021, de Universidad Computense de Madrid Sitio web: <http://www.fdi.ucm.es/profesor/jjruiz/ec-is/temas/Tema%205%20-%20Repaso%20ruta%20de%20datos.pdf>
- fjrodriguez2. (2014). Números de Fibonacci. 13 de Junio de 2021, de QuantDare Sitio web: <https://quantdare.com/numeros-de-fibonacci/>