Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Laboratorio de Organización y Arquitectura de Computadoras

Práctica 08:

Procesador RISC 68HC11

Alumnos:

- Murrieta Villegas Alfonso
- Reza Chavarria Sergio Gabriel
- Valdespino Mendieta Joaquin

Profesora: Ayesha Sagrario Román García

Grupo: 7

Fecha de entrega: 26 de noviembre de 2021

Práctica 07

Objetivo

Diseñar un microprocesador RISC de 8 bits, específicamente la versión de pipeline del microprocesador 68HC11 de Motorola.

Introducción

Con el avance continuo de la tecnología, el poder de procesamiento ha ido escalando para satisfacer nuestras necesidades, a través de realizar tareas con un nivel de complejidad mayor, por lo que resulta importante a nivel de componentes, los diferentes dispositivos que conforman la CPU, uno de ellos son los procesadores, aquellas entidades o circuitos integrados encargadas de la ejecución de instrucciones, mediante cálculos en lenguaje maquina o binario.

Como recordamos dentro de estos microprocesadores, existen elementos esenciales que permiten su funcionamiento, presentados a continuación:

-ALU: es el elemento que ejecuta los cálculos aritméticos y lógicos en una operación

-Unidad de control: elemento cuyo objetivo es mantener una secuencia y control de las operaciones de la unidad de procesamiento, buscarlas, decodificarlas y ejecutarlas,

-Registros: elementos cuyo fin es la de almacenar el código de operación y operandos de las instrucciones, leídas en memoria

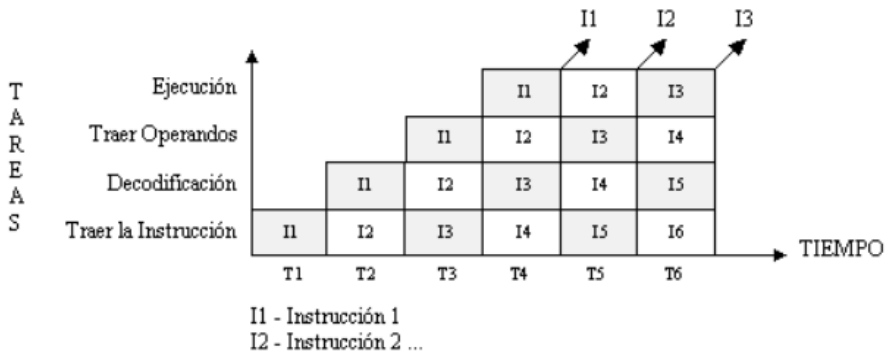
-Buses: son interconexiones entre los elementos, estos están divididos según el tipo de uso: buses de direcciones, bus de datos, bus de alimentación y buses de control

Por otro lado, con la constante optimización y necesidad de acelerar los procesos, hacer una secuencia de instrucciones, realizando los pasos del ciclo Fetch uno detrás de otro, termino siendo un proceso considerado relativamente lento, por lo que se implementó

una forma de realizar un procesamiento en conjunto, aprovechando los elementos y agregando otros, este se le denomino Pipeline.

El Pipeline es una técnica de implementación para ejecutar múltiples instrucciones simultáneamente, esta no reduce el tiempo de ejecución de una instrucción individual, pero puede reducir el tiempo de ejecución de un programa (conjunto de instrucciones)

Esta se le suele hacer una analogía con una línea de ensamble donde cada etapa o sector es un paso del ciclo Fetch, donde cada una de estas etapas completa una parte de la instrucción y desocupa el lugar, para dar entrada a la siguiente instrucción a completar la etapa, obteniendo una proporción como la observada en la siguiente figura, donde se observa el aprovechamiento de los tiempos.



(Figura 1: ejemplo de Pipeline)

En la presente practica se mostrará la implementación de estos elementos, basándose en la arquitectura de tipo CISC original 68HC11 de Motorola.

Desarrollo

1. La siguiente figura muestra el diagrama del procesador 68HC11 visto en teoría, instrumente la descripción de este hardware utilizando el lenguaje VHDL.

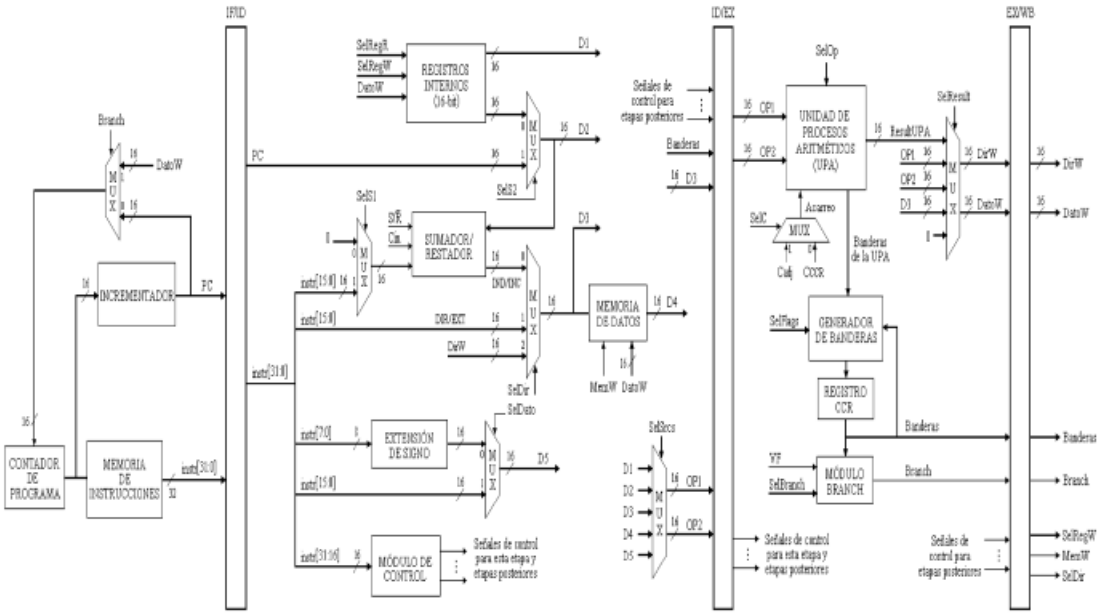


Figura 1: Arquitectura del procesaor 68HC11 en pipeline.

2. Pruebe la arquitectura con el siguiente código de lenguaje del 68HC11.

```
LDAB #$02
LDAA #$00
ABA
JMP #$0004
```

Realice los cambios en la arquitectura que crea convenientes.

Debido a la configuración con respecto a la arquitectura CISC, este ejercicio contemplo 4 etapas diferentes para el procesamiento de información y obtención de resultados de las operaciones a realizar.

A partir de la creación de los componentes como memorias, multiplexores, incrementador, la unidad de procesamiento aritmético, registros de banderas, etc, se obtuvieron las etapas de la arquitectura. La ventaja que tiene este tipo de arquitectura es que puede realizar varias acciones al mismo tiempo, en diferentes etapas.

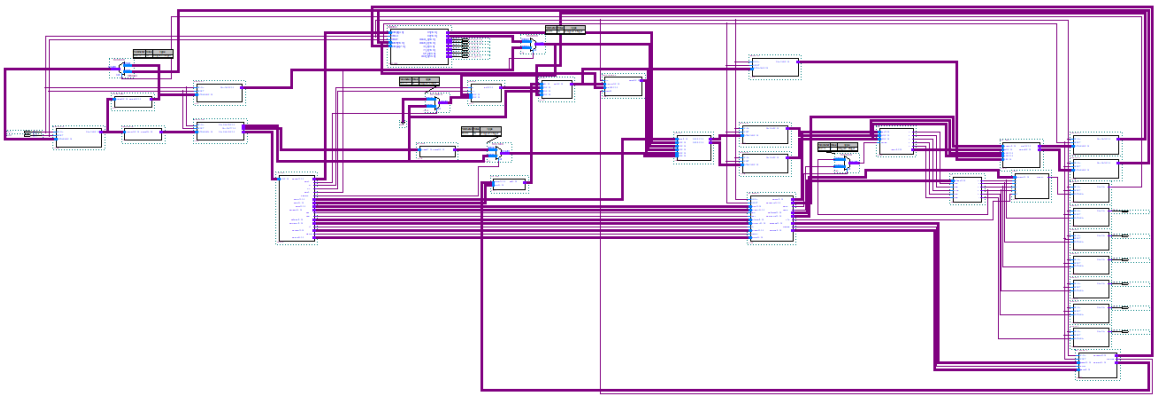


Ilustración 1: Arquitectura RISC

Las 4 etapas que corresponden a las etapas del ciclo fetch se consideran de la siguiente manera:

- Lectura de Instrucción: Lectura de memoria de la instrucción a ejecutar, obteniendo de una pasada toda la información que se necesitará
- Decodificación: La instrucción leída anteriormente es decodificada. El módulo de control genera señales de control que manejarán el hardware de esta etapa y posteriores.
- Ejecución: Opera los operandos obtenidos de la decodificación, actualiza los registros de estados o banderas y calcula la condición de salto.
- Post Escritura: Actualizar los resultados obtenidos en etapas anteriores.

Las ventajas de la arquitectura RISC es el hecho del manejo de tarea múltiples, así disminuyendo el tiempo de ejecución y en manejo de instrucciones básicas.

A continuación, se anexarán las 4 partes de la arquitectura RISC

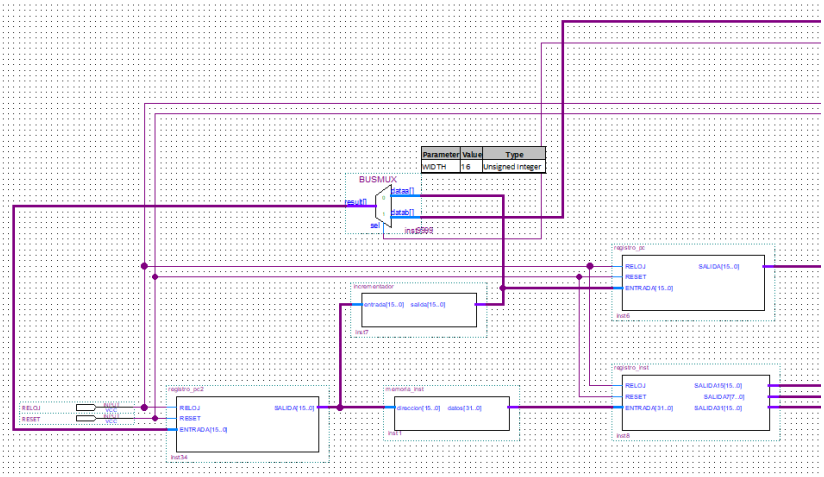


Ilustración 2: Etapa 1 (Lectura de Instrucción)

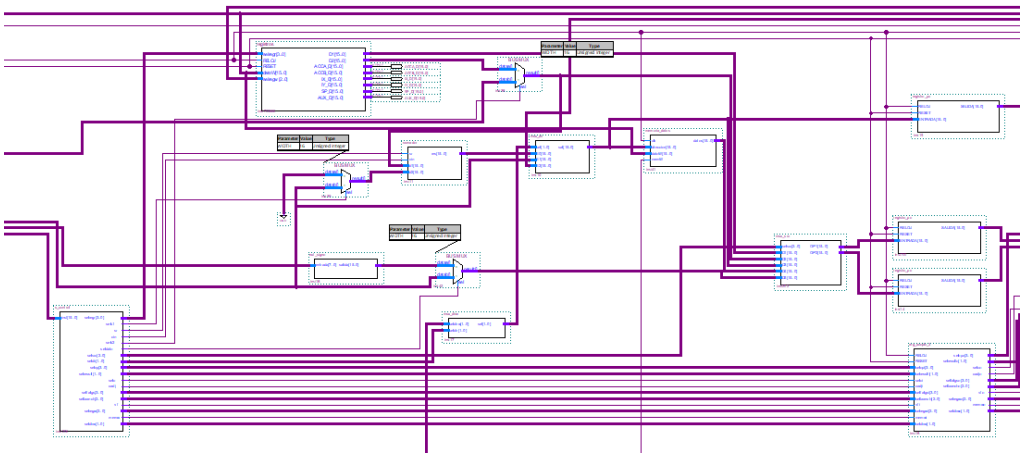


Ilustración 3: Etapa 2 (Decodificación)

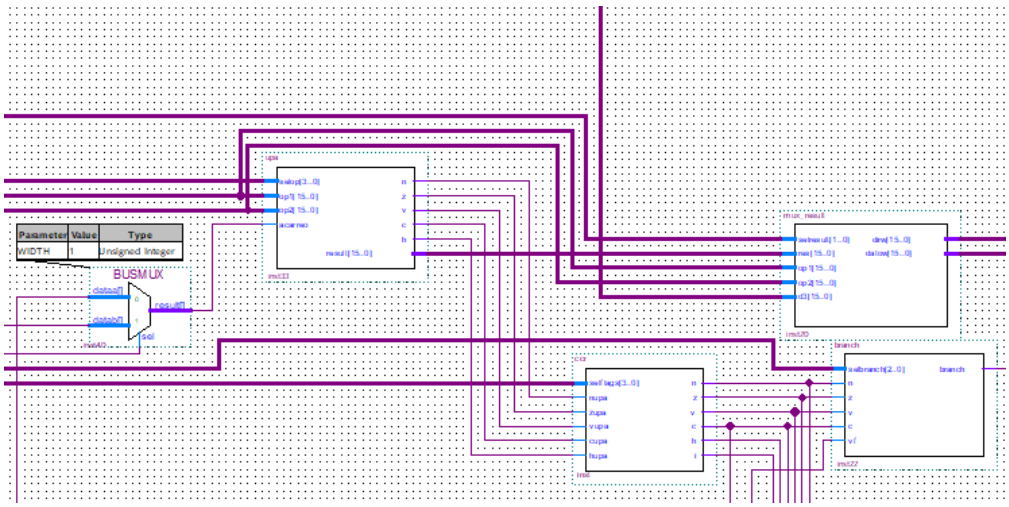


Ilustración 4: Etapa 3 (Ejecución)

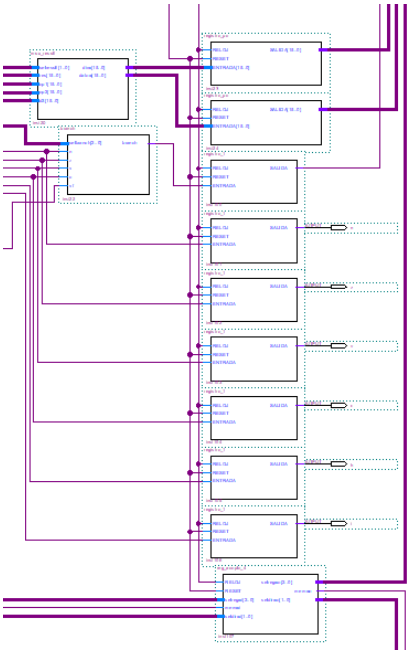


Ilustración 5: Etapa 4 (Post Escritura)

Ya con la arquitectura, es necesario escribir el código correspondiente a las operaciones. Estas instrucciones tienen el mismo significado que las utilizadas en la práctica anterior.

Para cada instrucción se necesitó tener la información y las señales necesarias para el complimiento de las siguientes etapas. La unidad de control se encargará de esto una vez que se le de lectura a las operaciones.

Es necesario tomar en cuenta la funcionalidad de cada señal, e cual cada instrucción se asignará para la operación.

Instrucción LDAB

Operación que carga en el registro ACCB un dato de 16 bits de manera inmediata. Para esto es necesario tener en cuenta las siguientes señales para la implementación de la operación.

Opcode Utilizado: C6

Etapa	Señal de control	Valor	Función
Etapa 2 (Decodificación)	SelRegR	0	Informa al módulo del registros internos que registro leer

	SelS1	0	Selecciona el primer dato que genera el módulo de registros. Si es 0 elige el valor de 0
	S/	1	Modulo sumador Restador que realice la suma entre los operando seleccionados.
	Cin	0	Acarreo de entrada
	SelS2	0	Selecciona el segundo dato que genera el módulo de registros
	SelDato	1	Selección del valor inmediato proveniente del formato de instrucción
	SelScrs	011	Guardado del dato inmediato
	SelDir	0	Selecciona el resultado obtenido por el modulo Sumador/Restador
Etapa 3 (Ejecución)	SelOp	100	Indica al UP la operación que se debe ejecutar
	SelResult	1	Toma el resultado de la operación
	SelC	1	Acarreo de entrada
	Cadj	0	Carreo de la etapa 2
	SelFlags	1	Banderas
	SelBranch	0	Revisión de la condición de salto
	VF	1	Valor comparado con 0, si es diferente, la señal Branch valdrá 0
Etapa 4 (Post Escritura)	SelRegW	100	Indica el módulo de registro interno en donde se guardará. Registro ACCA.
	MemW	0	Indica la operación a efectuar la memoria (Lectura o escritura)
	SelDir	0	Elección de bus de donde provendrá la dirección de memoria

Instrucción LDAA

Operación que carga en el registro ACCA un dato de 16 bits de manera inmediata. Para esto es necesario tener en cuenta las siguientes señales para la implementación de la operación.

Opcode: 86

Etapa	Señal de control	Valor	Función
Etapa 2 (Decodificación	SelRegR	0	Informa al módulo del registros internos que registro leer

	SelS1	0	Selecciona el primer dato que genera el módulo de registros. Si es 0 elige el valor de 0
	S/	1	Modulo sumador Restador que realice la suma entre los operando seleccionados.
	Cin	0	Acarreo de entrada
	SelS2	0	Selecciona el segundo dato que genera el módulo de registros
	SelDato	1	Selección del valor inmediato proveniente del formato de instrucción
	SelScrs	011	Guardado del dato inmediato
	SelDir	0	Selecciona el resultado obtenido por el modulo Sumador/Restador
Etapa 3 (Ejecución)	SelOp	100	Indica al UP la operación que se debe ejecutar
	SelResult	1	Toma el resultado de la operación
	SelC	1	Acarreo de entrada
	Cadj	0	Carreo de la etapa 2
	SelFlags	1	Banderas
	SelBranch	0	Revisión de la condición de salto
	VF	1	Valor comparado con 0, si es diferente, la señal Branch valdrá 0
Etapa 4 (Post Escritura)	SelRegW	1	Indica el módulo de registro interno en donde se guardará. Registro ACCA
	MemW	0	Indica la operación a efectuar la memoria (Lectura o escritura)
	SelDir	0	Elección de bus de donde provendrá la dirección de memoria

Instrucción ABA

Instrucción que realiza la suma del contenido del registro ACCA al contenido del registro ACCB, y el resultado lo guarda nuevamente en ACCA.

Opcode: 1B

Etapa	Señal de control	Valor	Función
Etapa 2 (Decodificación	SelRegR	1	Informa al módulo de los registros internos que registro leer. Registros A Y B
	SelS1	0	Selecciona el primer dato que genera el módulo de registros. Si es 0 elige el valor de 0
	S/	1	Modulo sumador Restador que realice la suma entre los operando seleccionados.
	Cin	0	Acarreo de entrada
	SelS2	0	Selecciona el segundo dato que genera el módulo de registros
	SelDato	1	Señal no utilizada
	SelScrs	1	Uso de buses D1 y D2.
	SelDir	0	Señal no utilizada
Etapa 3 (Ejecución)	SelOp	1	Indica al UP la operación que se debe ejecutar
	SelResult	1	Toma el resultado de la operación
	SelC	1	Acarreo de entrada
	Cadj	0	Carreo de la etapa 2
	SelFlags	10	Banderas
	SelBranch	0	Revisión de la condición de salto
	VF	1	Valor comparado con 0, si es diferente, la señal Branch valdrá 0
Etapa 4 (Post Escritura)	SelRegW	1	Indica el módulo de registro interno en donde se guardará. Registro ACCA
	MemW	0	Indica la operación a efectuar la memoria (Lectura o escritura). No se escribe nada en memoria
	SelDir	0	Elección de bus de donde provendrá la dirección de memoria. No se escribe nada en memoria

Instrucción JMP

Salto a la dirección indicada. En el caso del programa se realizará un salto a la dirección 2 para tener el comportamiento de la práctica pasada.

Opcode: 7E

Etapa	Señal de control	Valor	Función

Etapa 2 (Decodificación	SelRegR	0	Informa al módulo de los registros internos que registro leer.
	SelS1	0	Selecciona el primer dato que genera el módulo de registros. Si es 0 elige el valor de 0
	S/	0	Modulo sumador Restador que realice la suma entre los operando seleccionados.
	Cin	0	Acarreo de entrada
	SelS2	0	Selecciona el segundo dato que genera el módulo de registros
	SelDato	1	Señal no utilizada
	SelScrs	11	Uso de buses D1 y D2.
	SelDir	0	Señal no utilizada
Etapa 3 (Ejecución)	SelOp	100	Indica al UPa la operación que se debe ejecutar. OR
	SelResult	1	Toma el resultado de la operación
	SelC	0	Acarreo de entrada
	Cadj	0	Carreo de la etapa 2
	SelFlags	0	Banderas
	SelBranch	0	Revisión de la condición de salto
	VF	0	Branch =1
Etapa 4 (Post Escritura)	SelRegW	0	Ningún registro modificado
	MemW	0	Indica la operación a efectuar la memoria (Lectura o escritura). No se escribe nada en memoria
	SelDir	0	Elección de bus de donde provendrá la dirección de memoria. No se escribe nada en memoria

La sección encargada de la asignación de señales está encontrada en la etapa 1 de la arquitectura, la unidad de control. A continuación, se mostrará la implementación.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity u_control is
7       Port( inst : in STD_LOGIC_VECTOR (15 downto 0);
8             selregr : out STD_LOGIC_VECTOR (3 downto 0);
9             sels1 : out STD_LOGIC;
10            sr : out STD_LOGIC;
11            cin : out STD_LOGIC;
12            sels2 : out STD_LOGIC;
13            seldato : out STD_LOGIC;
14            selsrc : out STD_LOGIC_VECTOR (2 downto 0);
15            seldir : out STD_LOGIC_VECTOR(1 downto 0);
16            selop : out STD_LOGIC_VECTOR (3 downto 0);
17            selresult : out STD_LOGIC_VECTOR (1 downto 0);
18            selc : out STD_LOGIC;
19            cadj : out STD_LOGIC;
20            selfalgs : out STD_LOGIC_VECTOR (3 downto 0);
21            selbranch : out STD_LOGIC_VECTOR (2 downto 0);
22            vf : out STD_LOGIC;
23            selregw : out STD_LOGIC_VECTOR (2 downto 0);
24            memw : out STD_LOGIC;
25            seldirw : out STD_LOGIC_VECTOR (1 downto 0));
26  end u_control;
```

```
architecture Behavioral of u_control is
begin
    process (inst)
    begin
        case inst is
            when X"00C6" => -- LDAB
                selregr <= "0000";
                sels1 <= '0';
                sr <= '1';
                cin <= '0';
                sels2 <= '0';
                seldato <= '1';
                selsrc <= "011";
                seldir <= "00";
                selop <= "0100";
                selresult <= "01";
                selc <= '1';
                cadj <= '0';
                selfalgs <= "0001";
                selbranch <= "000";
                vf <= '1';
                selregw <= "100";
                memw <= '0';
                seldirw  <= "00";
            when X"0086" => -- LDAA
                selregr <= "0000";
                sels1 <= '0';
                sr <= '1';
                cin <= '0';
                sels2 <= '0';
                seldato <= '1';
                selsrc <= "011";
                seldir <= "00";
                selop <= "0100";
                selresult <= "01";
                selc <= '1';
                cadj <= '0';
                selfalgs <= "0001";
                selbranch <= "000";
                vf <= '1';
                selregw <= "001";
                memw <= '0';
                seldirw  <= "00";
```

```
            when X"001B" => -- ABA
                selregr <= "0001";
                sels1 <= '0';
                sr <= '1';
                cin <= '0';
                sels2 <= '0';
                seldato <= '1';
                selsrc <= "001";
                seldir <= "00";
                selop <= "0001";
                selresult <= "01";
                selc <= '1';
                cadj <= '0';
                selfalgs <= "0010";
                selbranch <= "000";
                vf <= '1';
                selregw <= "001";
                memw <= '0';
                seldirw  <= "00";
            when X"007E" => -- JMP
                selregr <= "0000";
                sels1 <= '0';
                sr <= '0';
                cin <= '0';
                sels2 <= '0';
                seldato <= '1';
                selsrc <= "011";
                seldir <= "00";
                selop <= "0100";
                selresult <= "01";
                selc <= '0';
                cadj <= '0';
                selfalgs <= "0000";
                selbranch <= "000";
                vf <= '0';
                selregw <= "000";
                memw <= '0';
                seldirw  <= "00";
```

```
109             WHEN OTHERS =>
110                selregr <= "0000";
111                sels1 <= '0';
112                sr <= '0';
113                cin <= '0';
114                sels2 <= '0';
115                seldato <= '0';
116                selsrc <= "000";
117                seldir <= "00";
118                selop <= "0000";
119                selresult <= "00";
120                selc <= '0';
121                cadj <= '0';
122                selfalgs <= "0000";
123                selbranch <= "000";
124                vf <= '1';
125                selregw <= "000";
126                memw <= '0';
127                seldirw  <= "00";
128         end case;
129      end process;
130   end Behavioral;
```

Código 1: Código de la Unidad de Control

Para generar la simulación, se necesita integrar el código de las instrucciones a realizar.

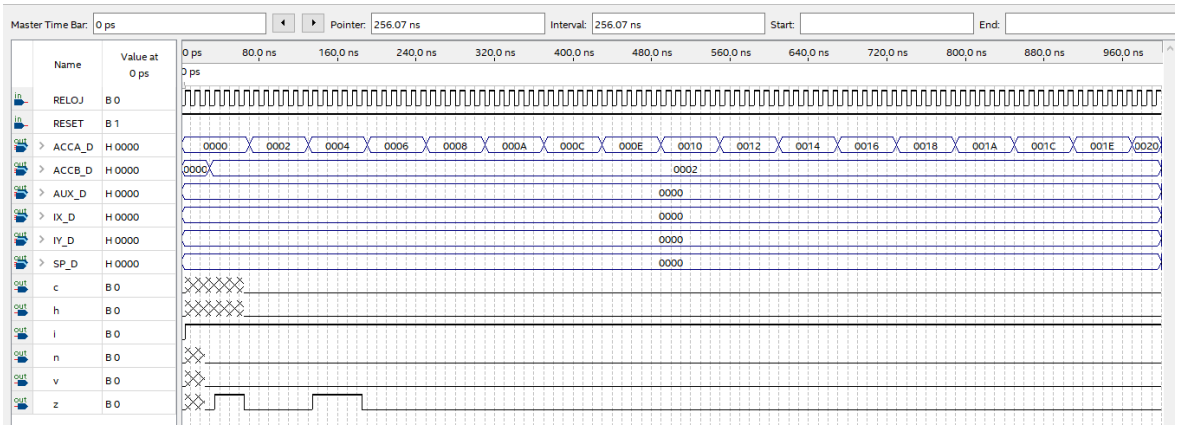
Nota: El caso por defecto se asignará a la operación NOP, esto es debido a que como cada instrucción pasa por las 4 etapas, es necesario tomar en cuenta que se necesita terminar la anterior operación para tomar en cuenta la operación. En el código en la instrucción 3 y 4 solo se hace tiempo para que la operación 1 y 2 (LDAB Y LDAA) puedan terminar su ciclo. El código, en formato *.MIF, será guardado en la memoria ram con archivo externo utilizado.

mem_prog.mif

```
1   -- Copyright (C) 2019  Intel Corporation. All rights reserved.
2   -- Your use of Intel Corporation's design tools, logic functions
3   -- and other software and tools, and any partner logic
4   -- functions, and any output files from any of the foregoing
5   -- (including device programming or simulation files), and any
6   -- associated documentation or information are expressly subject
7   -- to the terms and conditions of the Intel Program License
8   -- Subscription Agreement, the Intel Quartus Prime License Agreement,
9   -- the Intel FPGA IP License Agreement, or other applicable license
10  -- agreement, including, without limitation, that your use is for
11  -- the sole purpose of programming logic devices manufactured by
12  -- Intel and sold by Intel or its authorized distributors.  Please
13  -- refer to the applicable agreement for further details, at
14  -- https://fpgasoftware.intel.com/eula.
15
16  -- Quartus Prime generated Memory Initialization File (.mif)
17
18  WIDTH=32;
19  DEPTH=50;
20
21  ADDRESS_RADIX=HEX;
22  DATA_RADIX=HEX;
23
24  CONTENT BEGIN
25      00  :   00C60002;
26      01  :   00860000;
27      [02..03]  :   00010000;
28      04  :   001B0000;
29      05  :   007E0002;
30      [06..08]  :   00010000;
31      [09..31]  :   00000000;
32  END;
```

Simulación

A continuación, se mostrarán los registros y el comportamiento de la arquitectura.



Simulación 1: Resultado de código implementado en arquitectura RISC

Como se aprecia se realiza la suma entre el dato del acumulador A y el acumulador B, el cual tiene un valor de 2, y el resultado se guarda en el acumulador A. A comparación de la práctica anterior toma una cantidad de ciclos mayor con respecto a las instrucciones. Es necesario tomar en cuenta la finalización del ciclo con respecto a las instrucciones.

Conclusiones

Murrieta Villegas Alfonso

Sin duda la presente práctica es el resultado claro de un progreso sucesivo respecto a cada uno de los temas y elementos asociados a un procesador, como se puede observar a lo largo del desarrollo del procesador fuimos integrando elementos clave como memoria o incluso el diseñar un pipeline de procesamiento de manera implícita.

Además, y como parte complementaria, aprendimos la diferencia entre la arquitectura CISC y RISC donde, mediante el desarrollo de este microprocesador de tipo RISC observamos las ventajas que ofrece respecto a la otra arquitectura.

Por último, cabe destacar que sin duda, VHDL y sobre todo el trabajar con FPGA's nos da una alta versatilidad en el desarrollo de estos componentes que si bien tal vez no tengan la complejidad de lo que hoy en día hacen empresas como AMD o Intel, nos dan

una aproximación en cómo es que están compuestos los procesadores independientemente de la arquitectura, además de la forma en que estos funcionan.

Reza Chavarria Sergio Gabriel

Tomar en cuenta el tipo de arquitectura a utilizar es necesario para la integración en proyecto. La arquitectura RISC nos permite realizar operaciones o instrucciones de manera simultánea, en forma de pipeline, lo cual nos da una ejecución más rápida. Otro punto a considerar es si necesitamos utilizar operaciones complejas o simples, las cuales el RISC nos permite utilizar. Las 2 arquitecturas vistas tienen sus puntos a favor y en contra, así que es necesario tomar en cuenta el propósito y los requisitos de un proyecto para poder implementar alguno de los 2.

Valdespino Mendieta Joaquin

En la presente practica se pudo observar el comportamiento y posteriormente el manejo de los elementos esenciales que conforman un microprocesador de tipo RISC, elementos como los registros, la ALU, el secuenciador, o la implementación del pipeline que permite este tipo de arquitectura, entre otros, que pudieron ser implementados como emulador (clon) en Quartus, permitiendo ejecutar instrucciones básicas descritas y con ayuda del pipeline ejecutarlas de forma simultánea, implementadas gracias a un análisis previo y conjunción de conocimientos, como los modos de direccionamiento para el manejo de microinstrucciones, el desarrollo y armados de máquinas de estado, además de la interpretación de cartas ASM, permitiendo realizar las operaciones con la mayor calidad posible.

Bibliografía

- Savage J., Vázquez G & Chávez N. (2015). Diseño de Microprocesadores, Capítulo III Construcción De Máquinas de Estado usando Memorias. Encontrado el 20 de noviembre del 2021. Sitio Web: https://biorobotics.fi-p.unam.mx/wp-content/uploads/Courses/arquitectura_de_computadoras/material_de_apoyo/diseo_de_procesadores.pdf