INSITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

CIRCUITOS LÓGICOS: SDI-11322

PROYECTO FINAL

Computadora de un ciclo en FPGA

Alumnos:

Emiliano Sotomayor González - 155902

Jaime Limón Samperio - 162395

Profesor:

Edgar Alejandro Granados Osegueda

14 de diciembre de 2018

Resumen—Una computadora de un ciclo es aquella computadora que realiza una sola instrucción en un solo ciclo de reloj. Teniendo como objetivo el entendimiento del proceso de diseño y la implementación de dicha computadora, en el siguiente reporte se especifica su implementación en un FPGA (Field Programmable Gate Array). Además, se ejemplifica el uso de dicha computadora ejecutando un programa sencillo dentro de la misma.

1. Introducción

Los componentes que conforman a la computadora de un ciclo son los siguientes:

- Branch Control - controla la lógica sobre los saltos del programa
- Program Counter - lleva un control sobre el número de instrucción a realizar
- Instruction Memory - contiene una lista numerada de instrucciones a realizar
- Instruction Decoder - convierte una instrucción seleccionada en comandos para la comoputadora
- Zero Fill - toma un número de 3 bits y lo convierte a 16 bits
- Extend - toma un número de 6 bits y lo convierte a 16 bits
- Register File - funge como una memoria de registros
- Hexadecimal Display - controla los displays de 7 segmentos
- Function Unit - controla la lógica de operaciones lógicas, binarias o de corrimientos
- Data Memory - funge como una memoria RAM
- Mux - componente que funge como selección entre dos señales

La computadora de un ciclo puede realizar los programas que sean codificados dentro del *Intruction Memory*. Es dentro de dicho componente donde se codificará el programa a realizar.

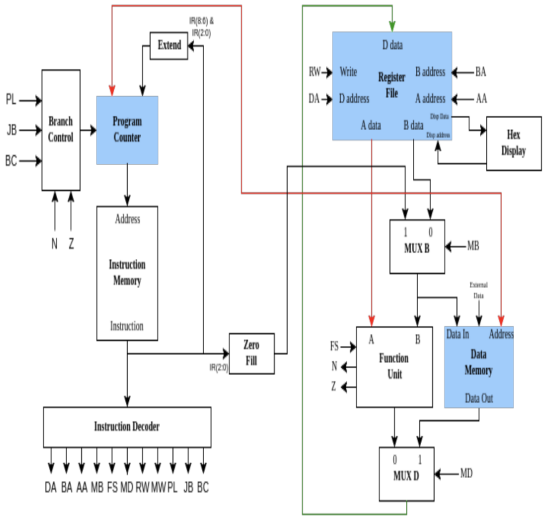

Figura 1. ESQUEMA DE LA COMPUTADORA

2. Objetivos

Se realizará un programa que resuelva la siguiente sumatoria

$$\sum_{i=1}^{n} \frac{i*(i+1)}{2} \quad ,\forall\ n\epsilon Z \wedge 1 <= n <= 7$$

3. Marco Teórico

Se puede descomponer un programa en instrucciones sencillas que se escribirán linea por linea en el *Instruction Memory*. Una vez dentro del *Instruction Memory* cada ciclo de reloj se ejecutará la instrucción correspondiente a el contador que reside dentro del *Program Counter*.

Los ciclos que se necesitan ejecutar (la sumatoria y la multiplicación) se pueden abstraer en determinadas lineas de código a partir de un indice que indique si se debería de continuar el ciclo o no. Para repetir un ciclo, teniendo en cuenta que el *Program Counter* es un contador que aumenta en 1 cada ciclo de reloj, agregamos condiciones de salto dado el indice elegido para indicar el fin del ciclo.

La división en el programa se puede ejecutar de una manera muy sencilla, ya que, al ser una división entre 2, está se puede programar realizando un corrimiento a la derecha del número que vayamos a dividir.

Utilizando el siguiente recuadro de instrucciones se pueden programar las distintas operaciones que se van a necesitar incluir en el *Instruction Memory* para poder realizar el programa deseado.

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{R[SA]}$ *	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

Figura 2. Recuadro de Instrucciones

4. DESARROLLO

Tomando en cuenta el siguiente formato de instrucciones, programaremos las siguientes instrucciones.

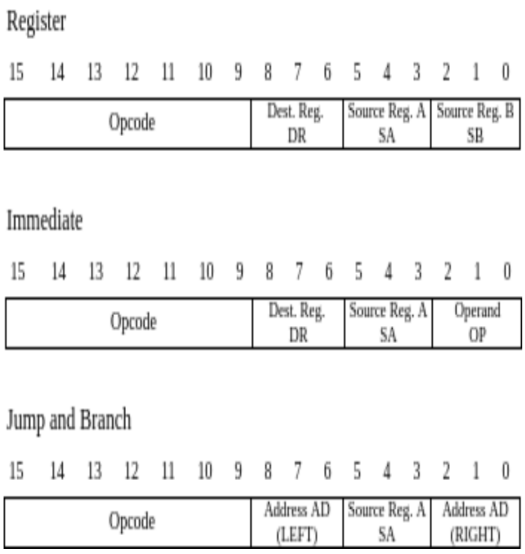

Figura 3. Formato de Instrucciones

CÓDIGO DEL PROGRAMA

1. LDI & O″0″& O″0″& O″6″when IE = X″0000″ else – R0 ← 6
2. LDI & O″1″& O″0″& O″6″when IE = X″0001″ else – R1 ← 6
3. ADD & O″0″& O″0″& O″1″when IE = X″0002″ else – R0 ← R0 + R1 = 12
4. ST & O″0″& O″7″& O″0″when IE = X″0003″ else – M[R7 = 0] ← R0
5. LDI & O″0″& O″0″& O″7″when IE = X″0004″ else – R0 ← 7
6. LDI & O″1″& O″0″& O″7″when IE = X″0005″ else – R1 ← 7
7. ADD & O″0″& O″0″& O″1″when IE = X″0006″ else – R0 ← R0 + R1
8. ADI & O″0″& O″0″& O″2″when IE = X″0007″ else – R0 ← R0 + 2 = 16
9. LDI & O″1″& O″0″& O″1″when IE = X″0008″ else – R1 ← 1
10. ST & O″0″& O″1″& O″0″when IE = X″0009″ else – M[R1 = 1] ← R0

11. LDI & O"0"& O"0"& O"7"when IE = X"000A" else – R0 ← N elegida (en este caso 7)
12. MOVA& O"5"& O"0"& O"0"when IE = X"000B" else – R5 ← i = N
13. INC & O"1"& O"5"& O"0"when IE = X"000C" else – R1 ← i + 1
14. BRZ & O"2"& O"5"& O"0"when IE = X"000D" else – if(i = 0){PC ← PC + 16}(Salta a Fin de Programa)
15. DEC & O"5"& O"5"& O"0"when IE = X"000E" else – R5 ← i - 1 (Inicia un ciclo de la sumatoria)
16. MOVA& O"3"& O"1"& O"0"when IE = X"000F" else – R3 ← j = i + 1
17. DEC & O"3"& O"3"& O"0"when IE = X"0010" else – R3 ← j - 1 (Inicia un ciclo de la multiplicación (i * (i+1)))
18. BRZ & O"0"& O"3"& O"5"when IE = X"0011" else – if(j = 0){PC ← PC + 5}(Termina de multiplicar, ve a dividir)
19. ADD & O"4"& O"4"& O"1"when IE = X"0012" else – R4 ← R4 + (i + 1) (Acumula un ciclo de la multiplicación)
20. LDI & O"6"& O"0"& O"1"when IE = X"0013" else – R6 ← 1
21. LD & O"6"& O"6"& O"0"when IE = X"0014" else – R6 ← M[R6 = 1] = 16
22. JMP & O"0"& O"6"& O"0"when IE = X"0015" else – PC ← R6 = 16 (Inicia otro ciclo de multiplicación)
23. SHR & O"4"& O"0"& O"4"when IE = X"0016" else – R4 ← R4sR = (R4/2)
24. ADD & O"7"& O"7"& O"4"when IE = X"0017" else – R7 ← R4 + R7 (Acumula un ciclo de la sumatoria)
25. ORX & O"4"& O"4"& O"4"when IE = X"0018" else – R4 ← (R4 $\oplus$ R4) = 0 (Limpia R4 para el sig. ciclo)
26. LDI & O"6"& O"0"& O"0"when IE = X"0019" else – R6 ← 0
27. LD & O"6"& O"6"& O"0"when IE = X"001A" else – R6 ← M[R6 = 0] = 12
28. JMP & O"0"& O"6"& O"0"when IE = X"001B" else – PC ← R6 = 12 (Inicia otro ciclo de la sumatoria)
29. "0101010"& "101"& "010"& "101"; – Operación nula. (Finaliza el Programa)

*** Las lineas del programa expuesto serían distintas dentro del programa. La línea 1 del programa expuesto sería la línea 0 dentro de la FPGA. Se utiliza la anterior notación dentro del código expuesto por cuestiones de entendimiento general para el lector.**

5. Resultados

El código expuesto en resultados encuentra la solución a el problema dado,

$$\sum_{i=1}^{n} \frac{i*(i+1)}{2} \quad ,\forall\ n\epsilon Z \wedge 1 <= n <= 7$$

Lo único necesario a hacer antes de correr el programa dentro de la FPGA es: cambiar el número 7 por la N deseada en la línea 11 del código expuesto.

Antes de que el programa fuera un éxito nos encontramos con distintos problemas:

1. Habíamos escrito números incorrectos en las memorias M0 y M1 y por lo tanto los saltos realizados dentro del programa erran erróneos.
2. Al agregar la condición para iniciar el ciclo de multiplicación (línea 16), agregábamos una j errónea y realizabamos la multiplicación de (n* (n+1)).
3. Al agregar la numeración dentro del programa, agregamos números decimales en lugar de números hexadecimales por lo que el programa terminaba incorrectamente en la línea 10.

6. Conclusiones

- **Emiliano Sotomayor González:** El proceso de desarrollo del proyecto se puede complicar si no se conoce a profundidad los componentes que lo conforman. Es necesario tener un conocimiento previo que satisfaga las necesidades técnicas del proyecto.
  El previo desarrollo del programa y las pruebas necesarias para conocer su buen funcionamiento deberían de ser necesarias antes de empezar a codificarlo en la FPGA, sino se dan las complicaciones que nosotros tuvimos durante el desarrollo del proyecto.
  Por último, siempre es bueno tener una buena referencia gráfica o escrita sobre lo previamente construido o sobre los componentes a utilizar. Cualquier herramienta que simplifique el proceso de desarrollo

del proyecto debería ser considerada con alta importancia.

- **Jaime Limón Samperio:** Durante el desarrollo del proyecto final (ensamblado de prácticas anteriores) se deben tener varias consideraciones.
  En ocasiones, para simplificar algunos procesos en prácticas anteriores, se debía cambiar alguna parte de los programas para realizar pruebas de componentes específicos. Por lo anterior, al ensamblar todos los componentes se podían tener problemas en la ejecución.
  Es importante siempre tener un registro de los cambios para no tener complicaciones. Otro punto a tomar en cuenta es saber explotar las herramientas disponibles. Al momento de utilizar el simulador existen funciones que pueden hacer que la detección de errores y/o seguimiento del programa sea mucho más sencillo.
  Por último, un punto a favor es investigar a fondo los componentes que se están utilizando. Saber el funcionamiento a fondo de un componente no solo es enriquecedor, sino ahorra problemas a la hora de la simulación.

7. ROLES

- **Emiliano Sotomayor González:** escritura y desarrollo en LaTeX del reporte, así como desarrollo del código en Quartus.
- **Jaime Limón Samperio:** desarrollo del código en Quartus y desarrolo del reporte.

8. FUENTES CONSULTADAS

[1] Logic & Computer Design Fundamentals de M. Morris R. Mano y Charles R. Kime.