Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Diseño de Procesadores

CPU. Básica

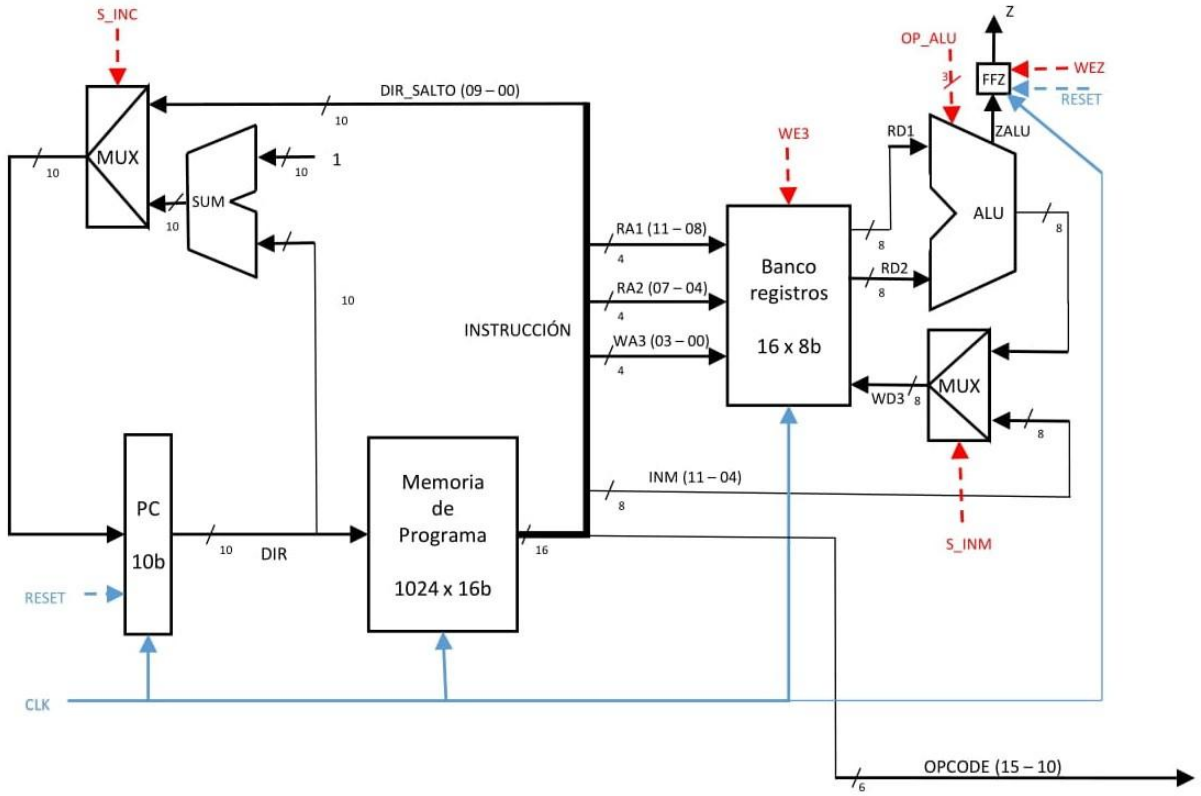
Cheuk Kelly Ng Pante
(alu0101364544@ull.edu.es)

Índice:

1. Introducción.

El objetivo de este proyecto es diseñar la unidad de control en Verilog de un procesador muy básico. Nos vamos a centrar en un procesador de un solo ciclo. Para que un procesador pueda ejecutar instrucciones en un solo ciclo sin recurrir al paralelismo en su implementación debemos separar las memorias de instrucciones y de datos de forma que se pueda realizar el acceso a ambas dentro del mismo ciclo.

La siguiente figura representa el camino de datos del procesador. En rojo y con línea discontinua son las señales que provienen de la Unidad de Control:



2. Codificación.

2.1. Codificación de las instrucciones

Codificación	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Saltos (J, JZ, JNZ)	OP	OP	OP	OP	OP	OP	D	D	D	D	D	D	D	D	D	D
Carga Inm. (LI)	OP	OP	OP	OP	C	C	C	C	C	C	C	C	Rd	Rd	Rd	Rd
Oper. ALU	OP	OP	OP	OP	R1	R1	R1	R1	R2	R2	R2	R2	Rd	Rd	Rd	Rd

2.2. Codificación elegida para cada instrucción.

Instrucción	OPCODE	Descripción	
LOAD	0000??	Carga un determinado valor en un registro	Instrucción de carga
J	001000	Salto condicional	Instrucción de salto
JZ	001001	Salto si el flag de 0 está activo	Instrucción de salto
JNZ	001010	Salto si el flag de 0 no está activo	Instrucción de salto
A	1000??	Salida de ALU = A	Instrucción Aritmético-Lógica
~A	1001??	Salida de ALU = A negada	Instrucción Aritmético-Lógica
ADD	1010??	Salida de ALU = suma	Instrucción Aritmético-Lógica
SUB	1011??	Salida de ALU = Resta	Instrucción Aritmético-Lógica
AND	1100??	Salida de ALU = Operación AND entre bits	Instrucción Aritmético-Lógica

OR	1101??	Salida de ALU = Operación OR entre bits	Instrucción Aritmético-Lógica
-A	1110??	Salida de ALU = Negar operando A	Instrucción Aritmético-Lógica
-B	1111??	Salida de ALU = Negar operando B	Instrucción Aritmético-Lógica

3. Instrucciones verificadas.

Para el correcto funcionamiento e implementación vamos a crear un pequeño programa en ensamblador. Se ha creados dos programas para comprobar las distintas instrucciones:

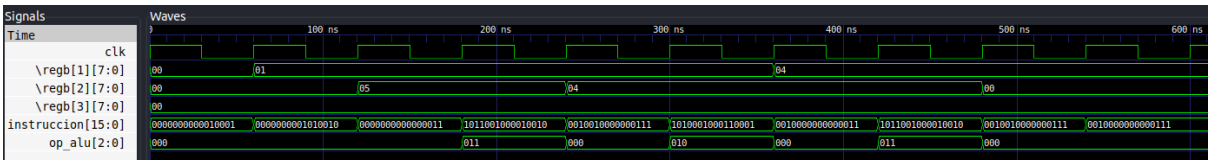
- Ejemplo 1:

```
program: li R1, 1
 li R2 5
 li R3 0
bucle: sub R2, R1, R2
 jz fin
 add R1, R2, R3
 j bucle
fin: j fin
```

Este primer ejemplo está codificado en el fichero `progfile.dat`. Para hacer la codificación hay que fijarse en la tabla de codificación de instrucciones, en este caso la traducción del código ensamblador a binario quedaría de la siguiente manera:

```
0000_0000_0001_0001  //Instrucción 0 li R1 1
0000_0000_0101_0010  //Instrucción 1 li R2 5
0000_0000_0000_0011  //Instrucción 2 li R3 0
1011_0010_0001_0010  //Instrucción 3 sub R2 R1 R2
0010_0100_0000_0111  //Instrucción 4 jz fin (#7)
1010_0010_0011_0001  //Instrucción 5 add R1 R2 R3
0010_0000_0000_0011  //Instrucción 6 J bucle (#3)
0010_0000_0000_0111  //Instrucción 7 J fin (#7)
```

Y el resultado en GtkWave es el siguiente:



- Ejemplo 2:

Para el ejemplo 2 la codificación se encuentra en el fichero progfile1.dat. El programa en ensamblador es el siguiente:

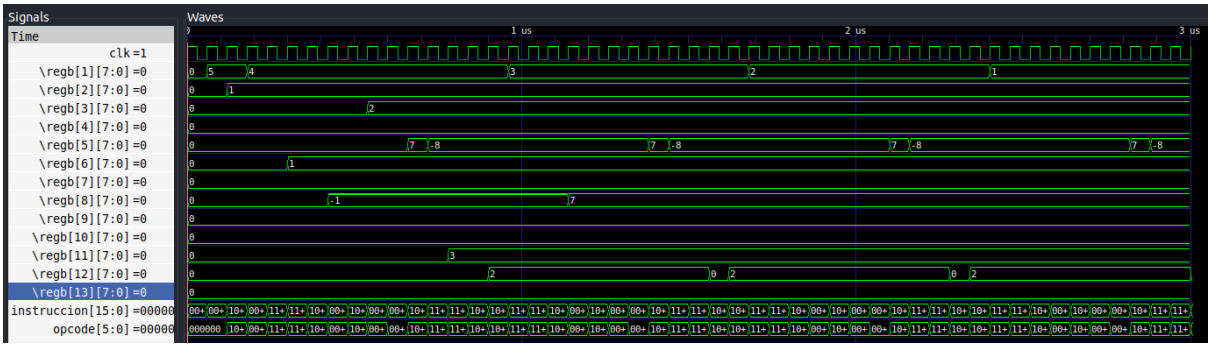
```
li R1 5                     ; R1 = 5
li R2 1                     ; R2 = 1
loop: sub R1 R1 R2          ; R1 = R1 - R2 = 5 - 1 = 4
     j endloop              ; salta a endloop
otraetiqueta: add R3 R2 R2  ; R3 = R2 + R2 = 1 + 1 = 2
     jnz main               ; si R3 != 0, salta a main = 2 != 0
main: li R5 7               ; R5 = 7
     not R5 R5              ; R5 = ~R5 = ~7
     or R11 R3 R2           ; R11 = R3 | R2 = 2 | 1 = 3
     and R12 R3 R2          ; R12 = R3 & R2 = 2 & 1 = 0
     add R12 R2 R2          ; R12 = R2 + R2 = 1 + 1 = 2
endloop: sub R1 R1 R2       ; R1 = R1 - R2 = 0 - 1 = -1
     and R6 R2 R2           ; R6 = R2 & R2 = 1 & 1 = 1
     or R7 R0 R0            ; R7 = R0 | R0 = 0 | 0 = 0
     not R8 R5              ; R8 = ~R5 = ~7
     jnz otraetiqueta       ; si R1 != 0, salta a otraetiqueta = -1 != 0
```

Su equivalencia en su codificación binaria es la siguiente:

```
0000_0000_0101_0001   // Instrucción 0 LI R1 5
0000_0000_0001_0010   // Instrucción 1 LI R2 1
1011_0001_0010_0001   // Instrucción 2 SUB R1 R1 R2
0010_0000_0000_1100   // Instrucción 4 j loop
1010_0010_0010_0011   // Instrucción 5 ADD R3 R2 R2
0010_1000_0000_0110   // Instrucción 6 JNZ main
0000_0000_0111_0101   // Instrucción 7 LI R5 7
1001_0101_0000_0101   // Instrucción 8 NOT R5 R5
1101_0011_0010_1011   // Instrucción 9 OR R11 R3 R2
1100_0011_0010_1100   // Instrucción 10 AND R12 R3 R2
1010_0010_0010_1100   // Instrucción 11 ADD R12 R2 R2
1011_0001_0010_0001   // Instrucción 12 SUB R1 R1 R2
1100_0010_0010_0110   // Instrucción 13 AND R6 R2 R2
1101_0000_0000_0111   // Instrucción 14 OR R7 R0 R0
```

```
1001_0101_0000_1000   // Instrucción 15 NOT R8 R5
0010_1000_0000_0100   // Instrucción 16 JNZ otraetiqueta (#5)
```

El resultado en GtkWave es la siguiente:



4. Dificultades encontradas.

Las dificultades encontradas han sido la codificación de la unidad de control, saber que señales de la unidad de control iban a estar activas para cada instrucción. Además, realizar la codificación de las instrucciones y comprobar el correcto funcionamiento en el programa GtkWave me resultó algo dificultoso.