Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica

IEE2753 - Diseño de Circuitos Integrados Digitales - 2017

Tarea 2: Procesador MIPS.

Juan Francisco Troncoso Z.

1. Procesador MIPS

El procesador diseñado tiene su base en la arquitectura MIPS. Su flujo de diseño consistió en la construcción de cada bloque logico abstrayendose de la implementación especifica que tendría en el procesador y preocupandose de que su construccion y funcionamiento sea el correcto para que su implementación tenga lugar sin problemas. Luego de tener la mayoría de elementos lógicos secuenciales y combinacionales construidos, se procedió a conectarlos mediante logica combinacional en los modulos "datapath" y "mips". El diagrama que representa su estructura, se puede visualizar en la Figura 1, donde se ha extendido la estructura combencional para adicionar instrucciones del tipo jump.

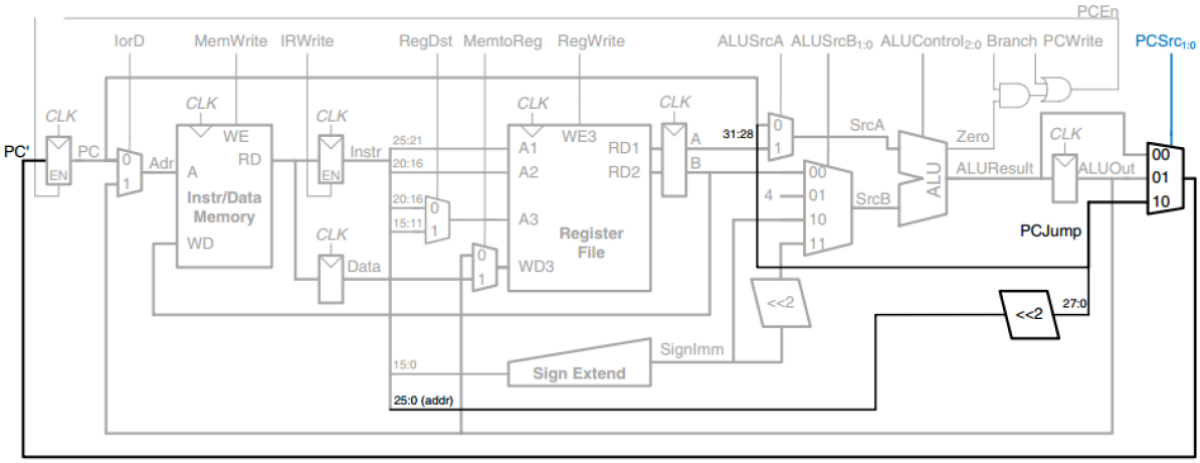

Figura 1: Estructura del procesador MIPS.

En forma resumida, el procesador operará en función de las instrucciones que recibirá el elemento lógico Instr Data Memory. En la arquitectura MIPS, se tiene 3 tipos de instrucciones. Las tipo R, las tipos Immediate y las tipo Jump. En nuestro caso, el procesador diseñado es de 32 bits, es decir, cada registro, instrucción y espacio en memoria tiene como base un espacio de 32 bits. Por lo anterior, cada instrucción de 32 bits, tienen ciertos bits dedicados para especificar los campos opcode, registros de fuente y destino, function, o immediate dependiendo de la instruccion. Estas instrucciones son codificadas por una unidad de control llamada "control mips" que comanda al resto de elementos lógicos para realizar lo que la instruccion quiera. A continuación se presentan 8 ejemplos que ejecutan instrucciones de los tres tipos en el procesador diseñado.

2. Elemento de Control del MIPS

El control del MIPS se baso en el siguiente diagrama de estados, el cual esta presente en las presentaciones del curso.

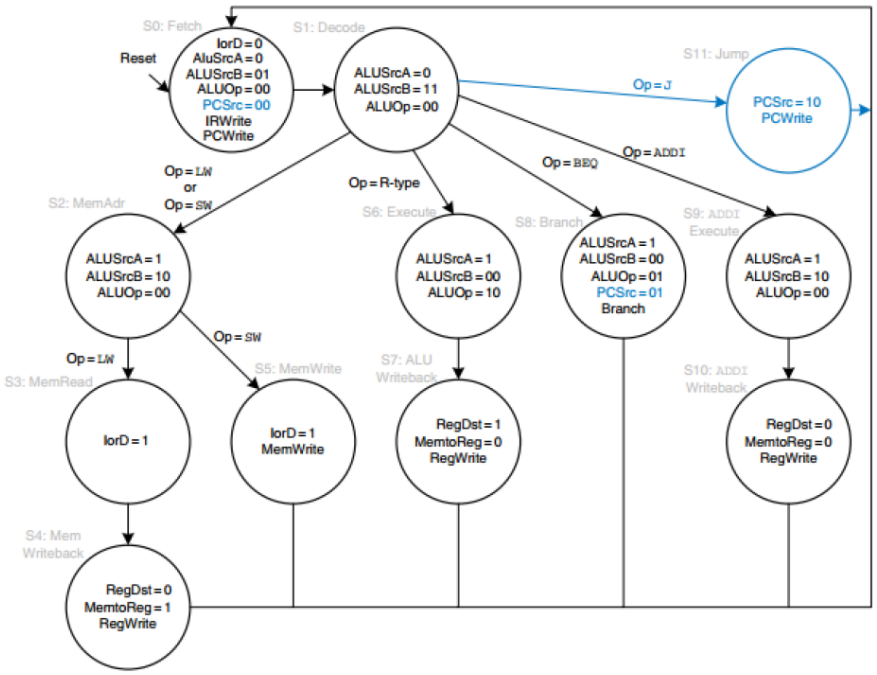

Figura 2: Diagrama de estados del MIPS.

Es importante mencionar que en el diagrama, no esta presente el estado de la instruccion “bnq”. En nuestro modulo se instacio de forma muy similar que el “beq” salvo que se definió un ALUOp=11 para especificar tal instrucción. Esta codifica una resta especifica que en caso de resultar cero, la variable Zero tomara valor 0, y en caso de no resultar cero, es decir ser distintos, Zero tomara valor 1.

3. Simulaciones del MIPS.

3.1. Instruccion tipo R: add

La primera instrucción a ejecutar es de tipo R. Especificamente se quiere realizar una suma entre dos registros que estan presentes en el "register file". En específico se quiere realizar una suma entre los registros s16 y s17 para almacenar el resultado en el registro s18. Para realizar las instrucciones, estas se deben almacenar en la memoria para poder ser ejecutadas, por lo que tenemos:

```
RAM[0] = 32'b000000_10000_10001_10010_00000_100000;
```

Considerando los campos que la instruccion de tipo R ofrece para la arquitectura MIPS, es facil ver que s18 = s16 + s17.

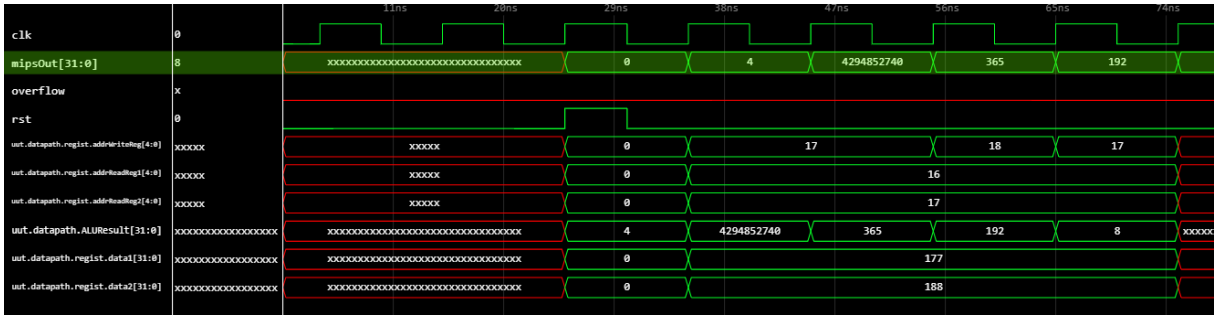

Figura 3: Simulacion tipo R, add.

Debido al diagrama de estados que se consideró para el elemento de control del MIPS, es facil ver que luego de 4 flancos de subida se obtiene el resultado. En nuestra simulación se observa que se ha instanciado al registro s16 = 177 y al registro s17= 188. Al cuarto flanco es posible observar en mipsOut el resultado de la suma el cual corresponde a 365. Tambien es posible observar que la variable "AddWriteReg" determina la direccion en la cual este resultado es escrito en los registros, el cual corresponde a 18 en el cuarto flanco de subida. Por lo anterior, la instruccion se esta ejecutando de buena manera.

3.2. Instruccion tipo R: sub

Considerando el analisis anterior, como es la instruccion tambien es de tipo R, el flujo de control y las variables son muy similares.

```
RAM[0] = 32'b000000_10001_10000_10010_00000_100010;
```

En este caso, como se realizará una resta se tiene que s18 = s17 - s16, donde el registro s17 = 188 y s16 = 177 por lo que la resta será 11. Nuevamente, luego de 4 flancos de subida se obtiene el resultado 11

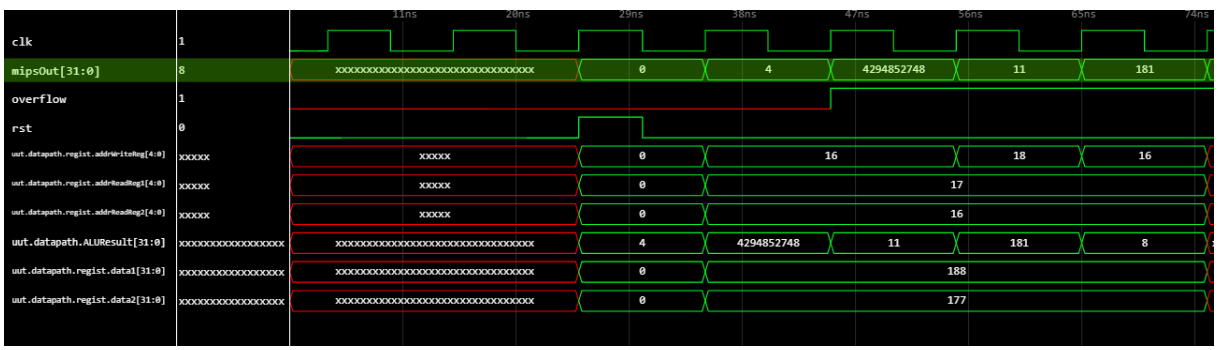

Figura 4: Simulacion tipo R, sub.

en “mipsOuts”, ademas se observa que “AddWriteReg” es 18 luego de cuatro flancos de subida del clock, por lo que la instruccion se realiza correctamente.

3.3. Instruccion tipo R: and

Adicionalmente, se agrega la instruccion de tipo R and. La instrucción corresponde a:

```
RAM[0] = 32'b000000_10001_10000_10010_00000_100100;
```

Considerando los campos de la instrucciond de tipo R, estamos realizando s18 = s17 && s16. Nuevamente,

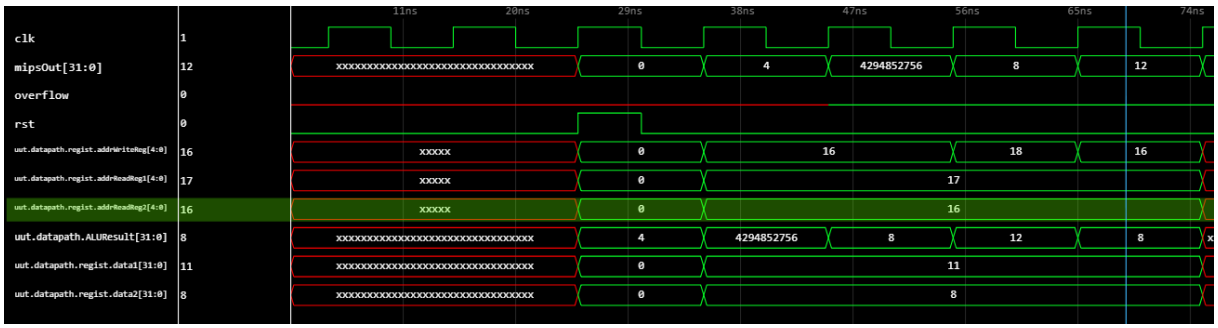

Figura 5: Simulacion tipo R, and.

como es de tipo R, el resultado se presenta luego de 4 flancos de subida del clock. Notemos que 11 se escribe en binario como 1011 y el 8 se escribe en binario como 1000 por lo tanto un and daria 1000, es decir, 8 en decimal, por lo tanto es correcto.

3.4. Instruccion tipo Immediate: beq

La instrucción tiene la estructura:

```
RAM[0] = 32'b000100_10001_10000_0000000000000011;
```

Como es de tipo Immediate especificamente beq, se comparan los registros s17 = 11 y s16 = 11. En caso de ser iguales, se realiza que PC = PC + Imm2 + 4. Es decir, beq if(s17 == s16) imm=3.

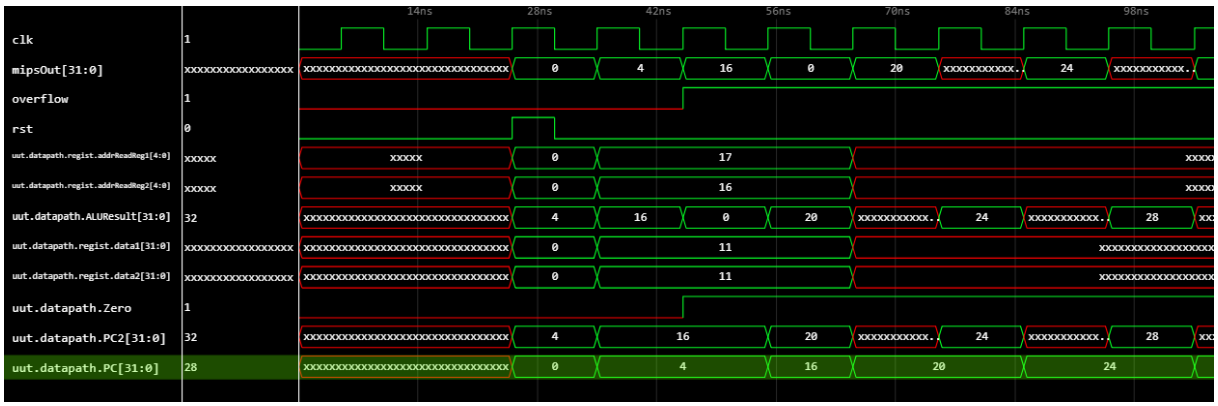

Figura 6: Simulacion tipo I, beq

Luego de 2 flancos de subida se obtiene el resultado dado que asi se definió en el diagrama de estados. Notemos que el immediato es 0011 cuyo shiftregister entrega 1100, es decir, a PC se le sumara 12+4 luego de transcurrir el beq, que demora 2 flancos de clock segun nuestra maquina de estados.

3.5. Instruccion tipo Immediate: bnq

La instrucción tiene la estructura:

RAM[0] = 32'b000101_10001_10000_0000000000000011;

La ntruccion es de tipo immediate, especificamente corresponde a un bnq, con lo cual se comparan los registros s17 y s16. En nuestro caso s17 = 12 y s16 = 11. En caso de ser distintos, se realiza que PC = PC + Imm2 + 4. Es decir, bnq if(s17 != s16) imm=3

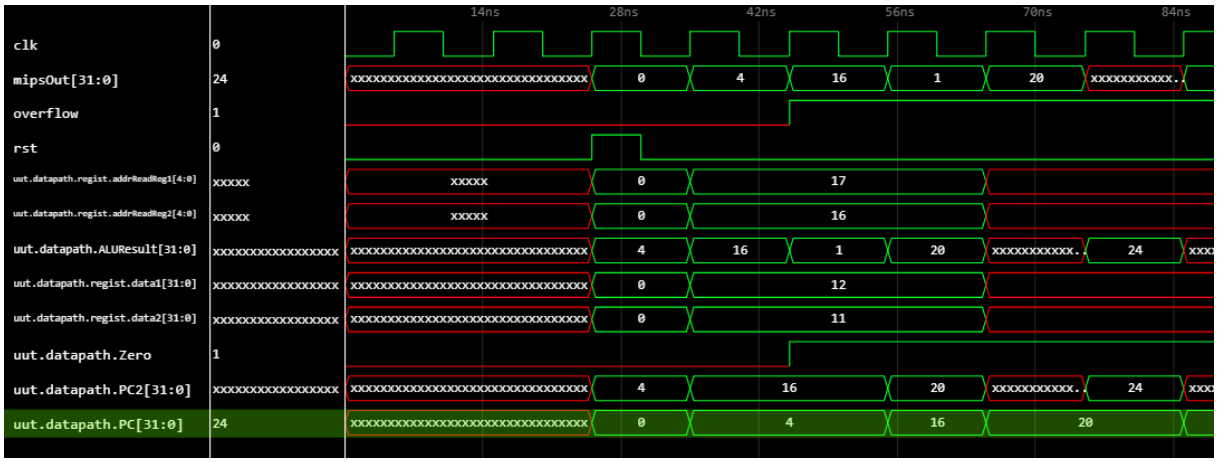

Figura 7: Simulacion tipo I, bnq

Luego de 2 flancos de subida se obtiene el resultado. Notemos que el immediato es 0011 cuyo shiftregister entrega 1100, es decir, a PC se le sumara 12+4 luego de transcurrir el beq, que demora 2 flancos de clock segun nuestra maquina de estados.

3.6. Instruccion tipo J: jump

La instruccion corresponde a un tipo J, y tiene la estructura:

RAM[0] = 32'b000010_00000_00000_0000000000001000;

Esta instrucción solo va hacia el valor en address shifteado en 2 hacia la izquierda. Es decir, jump address = 82.

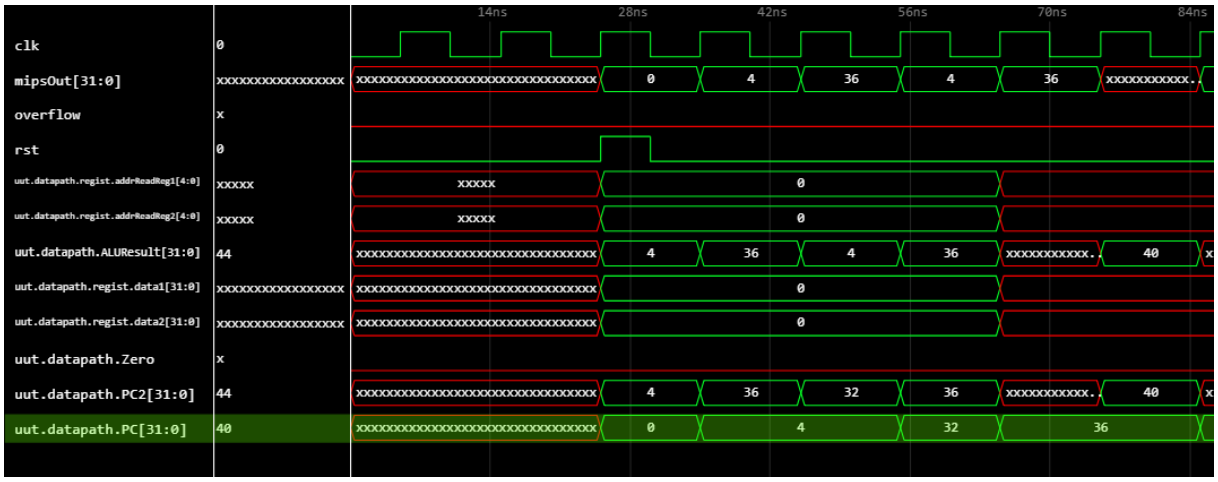

Figura 8: Simulacion tipo J, jump

Luego de 2 flancos de subida se obtiene el resultado. Notemos que el address es 1000 cuyo shiftregister entrega 100000, es decir, PC se direcciona hacia el 32 luego de dos flancos del clock.

3.7. Instruccion tipo I: save word

La instruccion tiene estructura:

```
RAM[0] = 32'b101011_00000_10000_0000000000001000;
```

Intruccion de tipo Immediate, donde el registro rt se guarda en la direccion en memoria segun el valor que este en el campo Immediate. En nuestro caso, se guarda en la direccion 8 de la memoria, lo que esta presente en el registro 16. En nuestro caso, rt = 16 y en el registro s16 tenemos el valor 11. Es decir, sw s16 8.

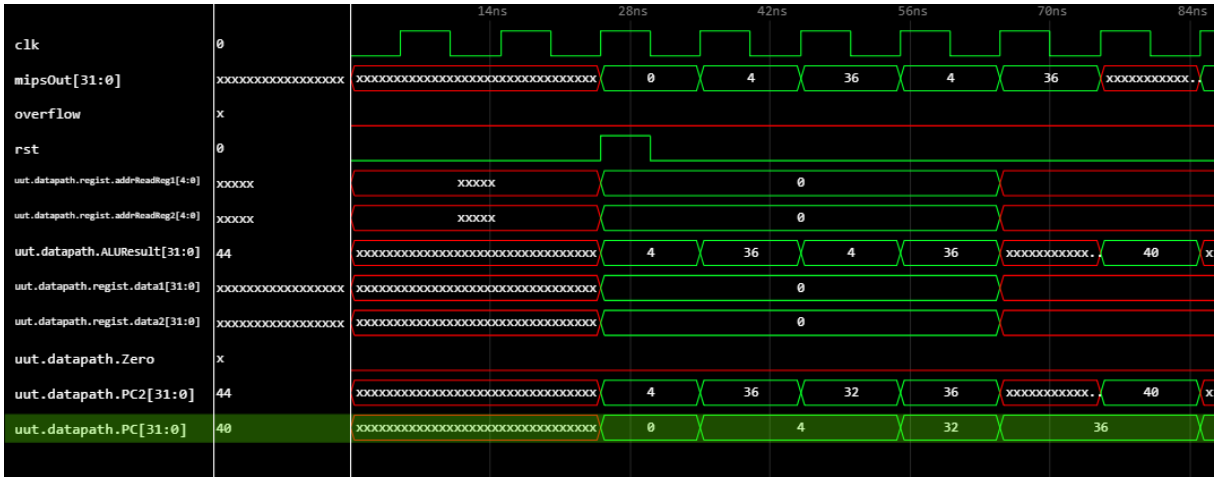

Figura 9: Simulacion tipo J, jump

Luego de 4 flancos de subida se obtiene el resultado dada por la maquina de estados realizada.

3.8. Instruccion tipo I: load word

La instruccion tiene la estructura:

```
RAM[0] = 32'b100011_10001_10000_0000000000010000;
RAM[9] = 32'd100;
```

Intruccion de tipo Immediate, donde se resume en lw $s16 offset($s17). En nuestro caso, el registro s17 almacena el valor 20, cuyo valor es sumado al offset escrito en el campo immediate. El resultado de esto es 20+16 = 36. Esta sera la dirección en memoria RAM que será inspeccionada para obtener el valor que se almacena en esa dirección. Como se direcciona por byte, la dirección 36 corresponde a la palabra RAM[9]. Esta almacena 32 bits cuyo valor en decimal es 100. Este valor 100 será el que se guardará en el registro s16.
Luego de 5 flancos de subida se obtiene el resultado dada por la maquina de estados realizada.

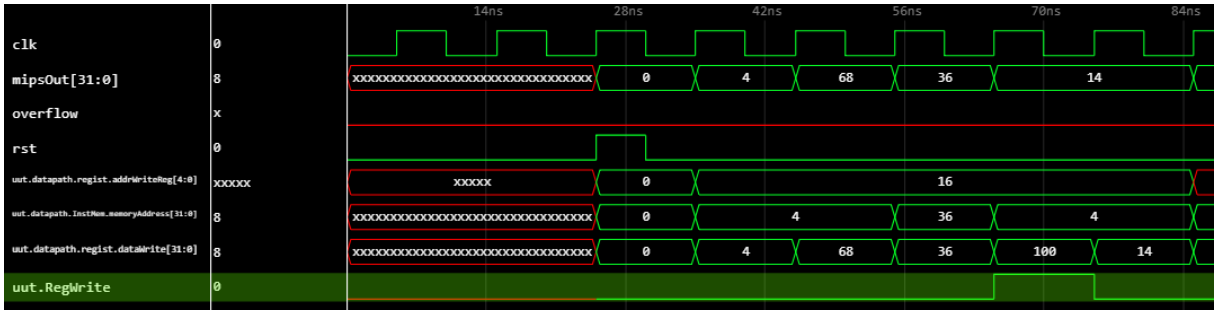

Figura 10: Simulacion tipo I, load word.