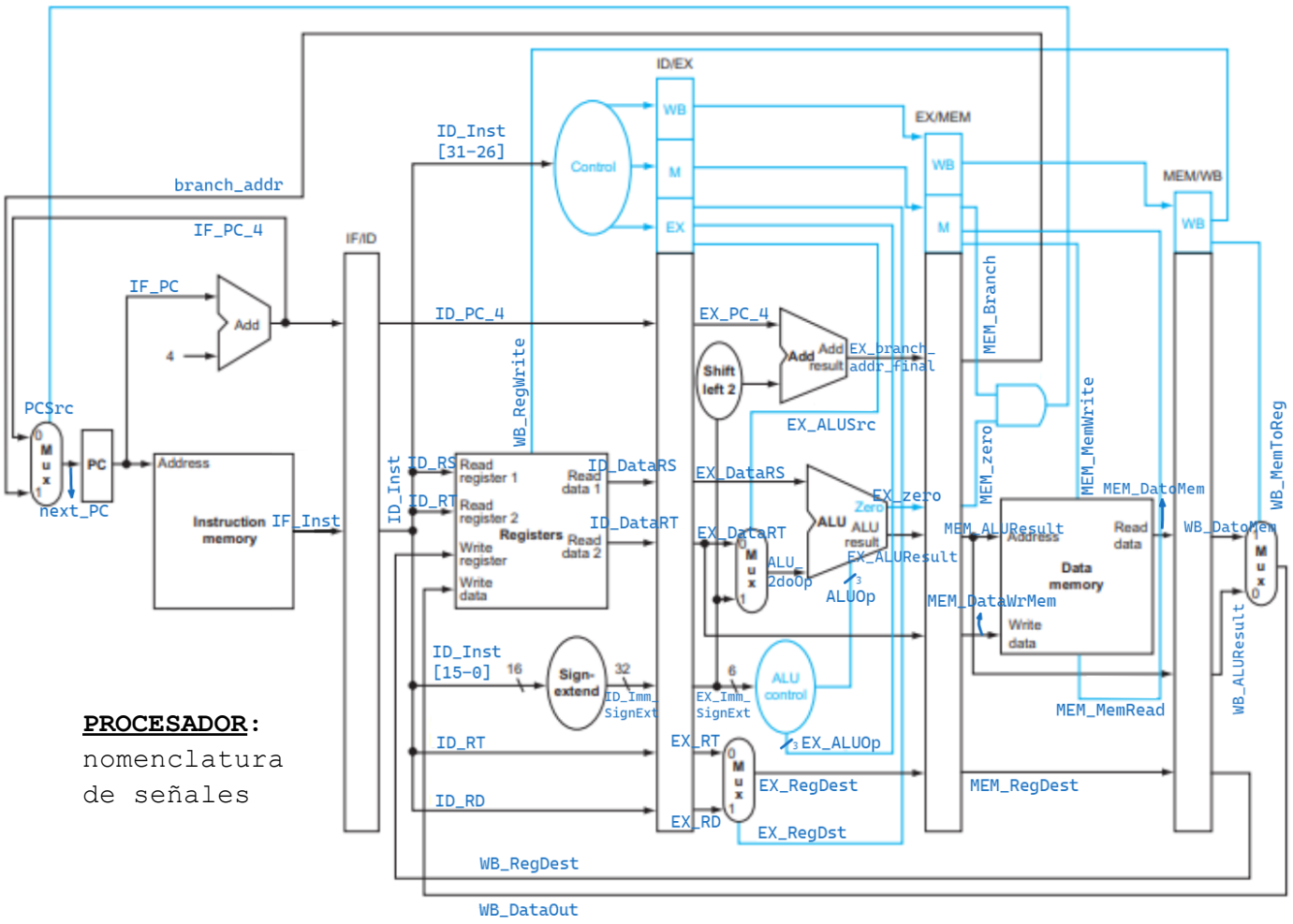
I N F O R M E

Arquitectura de Computadoras 1 - Laboratorio 2023

PROCESADOR MIPS SEGMENTADO

implementado en VHDL - EDAPlayground

COMISIÓN (2)	Martín Vázquez
Integrantes:	DELGADO, Lucia Ayelen FERRERI, Delfina Milagros
Enlace:	https://edaplayground.com/x/F8Yu
Descripción:	Comentarios sobre la realización del proyecto, haciendo énfasis sobre las problemáticas surgidas durante el proceso y las soluciones que pudimos encontrar para las mismas.



PROCESADOR:
nomenclatura de señales

PROBLEMÁTICAS Y DECISIONES

1. **Señal de control ALUOp: incrementamos la señal en un bit.**
   A diferencia del procesador MIPS real, solo contamos con un conjunto reducido de 12 instrucciones. Teniendo en cuenta esto, decidimos incrementar la señal de control **ALUOp** de 2 bits a 3 bits. Para las instrucciones de tipo-R, decidimos que **ALUOp** vale 111. Para las restantes, aquellas de tipo-I, **ALUOp** depende de la operación que debe efectuar la ALU (Por ejemplo: para LUI, ALUOp vale 100).
   Debido a que el conjunto es reducido, la implementación de la ALU Control es más sencilla. Funciona de la siguiente manera:
   - ALU Control recibe señal ALUOp.
     a. Si **ALUOp = 111**, la instrucción es de Tipo-R y ALU Control define la operación de la ALU según el campo **funct** de la instrucción.
     b. Si **ALUOp != 111**, la operación ya fue determinada previamente en la Unidad de Control de la etapa ID. ALU Control sencillamente deja pasar la señal a la ALU para que la ejecute.

2. **Errores y Warnings frecuentes al momento de compilar:**

```
#KERNEL: WARNING: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand,
the result will be 'X'(es).
#KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0.
```

   Inicialmente EDAPlayground parecía no compilar correctamente el procesador (no mostraba nunca la pestaña de EPWave a pesar de haber habilitado la opción). Adicionalmente tras 1 minuto de espera figuraba el siguiente error:

```
Execution interrupted or reached maximum runtime.
```

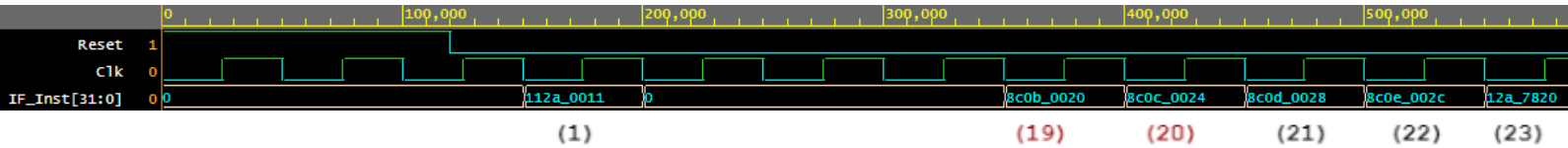
   Posteriormente nos dimos cuenta de que era un error de simulación: faltaba ingresar el parámetro del tiempo (run time). Lo seteamos en 2000ns.
   En el proceso de hallar el error investigamos qué significaban las advertencias anexadas arriba y descubrimos que no son necesariamente errores sino que advierten una inconsistencia, como por ejemplo querer convertir un metavalor (U,X,W,Z) a un entero utilizando la función to_integer(). Sin embargo, no siempre implican un riesgo real y es muy común que figuren antes de que el procesador concrete el reset total.

3. **Problema en el salto cuando se insertan instrucciones NOP.**
   Cuando logramos ejecutar la versión original del Program1 (que comienza con una instrucción BEQ seguida de tres NOPs) notamos una inconsistencia en la dirección de salto. Lo que debía ocurrir es que el salto sea efectivo ($t1=$t2=0 debido al reset) y por ende PC tenía que actualizarse a la dirección de la primera instrucción de lab1, ubicada en la dirección **80 (instrucción 21)** de la memoria de instrucciones.
   Sin embargo, saltaba a la línea **72 (instrucción 19)**:

   112a0011: beq $t1, $t2, lab1 **000100 01001 01010 0000000000010001**
   EX_branch_addr = 4 + 17*4 = 72 **-> salta a la instrucción 19**

   Concluímos en que el error se daba por la inserción de instrucciones NOP que (asumimos) es realizada por el compilador al detectar riesgos de control. En consecuencia, se produce un ligero corrimiento en las direcciones de las instrucciones posteriores en la memoria de instrucciones.

   En síntesis: la dirección de lab1 en memoria de datos cambia y por ende, si no se hace nada, la instrucción beq funcionaría incorrectamente, como se observa en la imágen:



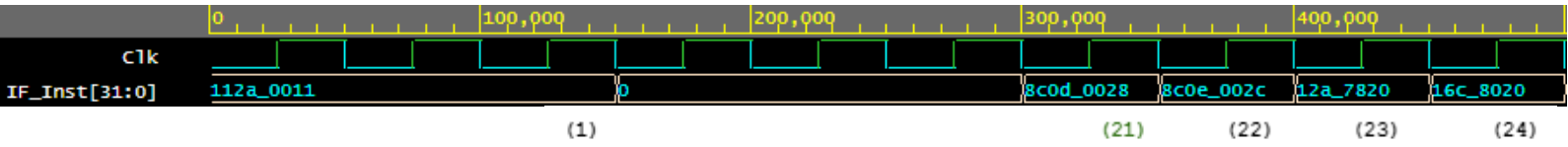
   Ante esta problemática se nos ocurrieron dos alternativas:
   1. Modificar la instrucción de BEQ teniendo en cuenta dicho corrimiento (NOPs insertadas por el programador o por un compilador que lo tenga en cuenta).
   2. Adaptar el procesador para que sea capaz de detectar estas situaciones y actuar en consecuencia.

   Optamos por la segunda opción.
   - Asumimos que el compilador nunca tiene en cuenta el corrimiento, entonces es responsabilidad del procesador.
   - Asumimos que siempre después de una BEQ se insertan 3 NOPs.

   MODIFICACIÓN: cuando se detecta una instrucción NOP en la etapa ID, la instrucción de la etapa EX suma 8 a la dirección de salto. Si es una instrucción BEQ, salta correctamente. Si no es una instrucción BEQ, no hay consecuencias por haber realizado la suma.

Consideramos que la ventaja que presenta esta solución es que si el compilador aplicara otra técnica para evitar riesgos de control, como por ejemplo *delayed decision*, no intervendría el mecanismo descripto.
La desventaja es que es una solución bastante estática: si se insertan más o menos NOPs tras una BEQ, no funcionaría. Lo mismo si el compilador o el programador de repente comenzaran a tomar en cuenta el corrimiento y modificar BEQ en consecuencia.

Tras aplicar la modificación, el salto es correcto:



4. **Incongruencia en Program1 - Program1.s**
   Al hacer el seguimiento, encontramos una incongruencia en las instrucciones 18, 19 y 20 con respecto al programa descrito en el archivo **program1.s:**

```
18. 8c0a001c:    lw $t6, 20($zero)        100011 00000 01010  0000000000011100
19. 8c0b0020:    lw $t2, 28($zero)        100011 00000 01011  0000000000100000
20. 8c0c0024:    lw $t3, 32($zero)        100011 00000 01100  0000000000100100
```

   18. Según **program1.s** escribe en $t6, pero en realidad escribe en $t2
   19. Según **program1.s** escribe en $t2, pero en realidad escribe en $t3
   20. Según **program1.s** escribe en $t3, pero en realidad escribe en $t4

   RESOLUCION: Modificamos **program1** para que corresponda con lo descrito en **program1.s**. A partir de esta resolución realizamos el seguimiento del programa.

5. **Eliminamos las primeras cuatro instrucciones para que se ejecuten las LW y SW de modo que el programa tenga más sentido.**

PROGRAM1	assembly language	machine language
main:		
1. 112a0011:	beq $t1, $t2, lab1	000100 01001 01010 0000000000010001
2. 00000000:	nop	000000 00000 00000 00000 00000 000000
3. 00000000:	nop	000000 00000 00000 00000 00000 000000
4. 00000000:	nop	000000 00000 00000 00000 00000 000000
5. 8c090000:	lw $t1, 0($zero)	100011 00000 01001 0000000000000000
6. 8c0a0004:	lw $t2, 4($zero)	100011 00000 01010 0000000000000100
7. 8c0b0008:	lw $t3, 8($zero)	100011 00000 01011 0000000000001000
8. 8c0c000c:	lw $t4, 12($zero)	100011 00000 01100 0000000000001100
9. 8c0d0010:	lw $t5, 16($zero)	100011 00000 01101 0000000000010000
10. 8c0e0014:	lw $t6, 20($zero)	100011 00000 01110 0000000000010100
11. ac090018:	sw $t1, 24($zero)	101011 00000 01001 0000000000011000
12. ac0a001c:	sw $t2, 28($zero)	101011 00000 01010 0000000000011100
13. ac0b0020:	sw $t3, 32($zero)	101011 00000 01011 0000000000100000
14. ac0c0024:	sw $t4, 36($zero)	101011 00000 01100 0000000000100100
15. ac0d0028:	sw $t5, 40($zero)	101011 00000 01101 0000000000101000
16. ac0e002c:	sw $t6, 44($zero)	101011 00000 01110 0000000000101100
17. 8c090018:	lw $t1, 24($zero)	100011 00000 01001 0000000000011000
18. 8c0e001c:	lw $t6, 20($zero)	100011 00000 01110 0000000000011100
19. 8c0a0020:	lw $t2, 28($zero)	100011 00000 01010 0000000000100000
20. 8c0b0024:	lw $t3, 32($zero)	100011 00000 01011 0000000000100100
lab1:		
21. 8c0d0028:	lw $t5, 40($zero)	100011 00000 01101 0000000000101000
22. 8c0e002c:	lw $t6, 44($zero)	100011 00000 01110 0000000000101100
23. 012a7820:	add $t7, $t1, $t2	000000 01001 01010 01111 00000 100000
24. 016c8020:	add $s0, $t3, $t4	000000 01011 01100 10000 00000 100000
25. 01a98822:	sub $s1, $t5, $t1	000000 01110 01010 10010 00000 100010
26. 01ca9022:	sub $s2, $t6, $t2	000000 01110 01010 10011 00000 100010
27. 012a9824:	and $s3, $t1, $t2	000000 01001 01010 10011 00000 100100
28. 01eaa024:	and $s4, $t7, $t2	000000 01110 10101 01010 00000 100100
29. 012aa825:	or $s5, $t1, $t2	000000 01001 01010 10101 00000 100101
30. 020ab025:	or $s6, $s0, $t2	000000 10000 01010 10110 00000 100101
31. 012ab82a:	slt $s7, $t1, $t2	000000 01001 01010 10111 00000 101010
32. 020ac02a:	slt $t8, $s0, $t2	000000 10000 01010 11000 00000 101010
33. 3c090001:	lui $t1, 1	001111 00000 01001 0000000000000001
34. 3c0a0002:	lui $t2, 2	001111 00000 01010 0000000000000010
35. 216b0006:	addi $t3, $t3, 6	001000 01011 01011 0000000000000110
36. 358c0070:	ori $t4, $t4, 112	001101 01100 01100 0000000000000111
37. 31ef0002:	andi $t7, $t7, 2	001100 01111 01111 0000000000000010

SEGUIMIENTO: BANCO DE REGISTROS

#	0	9	a	b	c	d	e	f	18	10	11	12	13	14	15	16	17
i	$0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$t8	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	2	4	8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	1	2	4	8	10	0	0	0	0	0	0	0	0	0	0	0
10	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
11	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
12	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
13	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
14	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
15	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
16	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
17	0	1	2	4	8	10	20	0	0	0	0	0	0	0	0	0	0
18	0	1	2	4	8	10	2	0	0	0	0	0	0	0	0	0	0
19	0	1	4	4	8	10	2	0	0	0	0	0	0	0	0	0	0
20	0	1	4	8	8	10	2	0	0	0	0	0	0	0	0	0	0
21	0	1	4	8	8	10	2	0	0	0	0	0	0	0	0	0	0
22	0	1	4	8	8	10	20	0	0	0	0	0	0	0	0	0	0
23	0	1	4	8	8	10	20	5	0	0	0	0	0	0	0	0	0
24	0	1	4	8	8	10	20	5	0	10	0	0	0	0	0	0	0
25	0	1	4	8	8	10	20	5	0	10	f	0	0	0	0	0	0
26	0	1	4	8	8	10	20	5	0	10	f	1c	0	0	0	0	0
27	0	1	4	8	8	10	20	5	0	10	f	1c	0	0	0	0	0
28	0	1	4	8	8	10	20	5	0	10	f	1c	0	4	0	0	0
29	0	1	4	8	8	10	20	5	0	10	f	1c	0	4	5	0	0
30	0	1	4	8	8	10	20	5	0	10	f	1c	0	4	5	14	0
31	0	1	4	8	8	10	20	5	0	10	f	1c	0	4	5	14	1
32	0	1	4	8	8	10	20	5	0	10	f	1c	0	4	5	14	1
33	0	10	4	8	8	10	20	5	0	10	f	1c	0	4	5	14	1
34	0	10	20	8	8	10	20	5	0	10	f	1c	0	4	5	14	1
35	0	10	20	e	8	10	20	5	0	10	f	1c	0	4	5	14	1
36	0	10	20	e	78	10	20	5	0	10	f	1c	0	4	5	14	1
37	0	10	20	e	78	10	20	0	0	10	f	1c	0	4	5	14	1
EF	0	10	20	e	78	10	20	0	0	10	f	1c	0	4	5	14	1
i	$0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$t8	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
#	0	9	a	b	c	d	e	f	18	10	11	12	13	14	15	16	17

LECTURA - ESCRITURA - LECTURA Y ESCRITURA - ESTADO FINAL

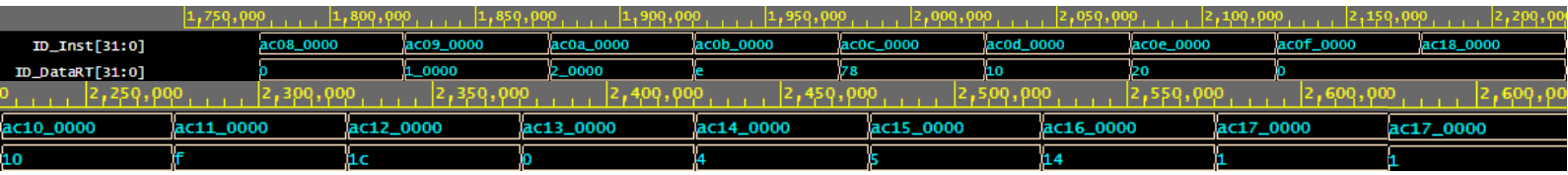
SEGUIMIENTO: MEMORIA DE DATOS

dir	00	04	08	c	10	14	18	1c	20	24	28	2c
-	1	2	4	8	10	20	0	0	0	0	0	0
5	1	2	4	8	10	20	0	0	0	0	0	0
6	1	2	4	8	10	20	0	0	0	0	0	0
7	1	2	4	8	10	20	0	0	0	0	0	0
8	1	2	4	8	10	20	0	0	0	0	0	0
9	1	2	4	8	10	20	0	0	0	0	0	0
10	1	2	4	8	10	20	0	0	0	0	0	0
11	1	2	4	8	10	20	1	0	0	0	0	0
12	1	2	4	8	10	20	1	2	0	0	0	0
13	1	2	4	8	10	20	1	2	4	0	0	0
14	1	2	4	8	10	20	1	2	4	8	0	0
15	1	2	4	8	10	20	1	2	4	8	10	0
16	1	2	4	8	10	20	1	2	4	8	10	20
17	1	2	4	8	10	20	1	2	4	8	10	20
18	1	2	4	8	10	20	1	2	4	8	10	20
19	1	2	4	8	10	20	1	2	4	8	10	20
20	1	2	4	8	10	20	1	2	4	8	10	20
21	1	2	4	8	10	20	1	2	4	8	10	20
22	1	2	4	8	10	20	1	2	4	8	10	20
EF	1	2	4	8	10	20	1	2	4	8	10	20
dir	00	04	08	0c	10	14	18	1c	20	24	28	2c

LECTURA - ESCRITURA - ESTADO FINAL

VERIFICACIÓN: BANCO DE REGISTROS

Agregamos al final del programa original 17 instrucciones **SW** para conocer el estado final del banco de registros. En la imágen siguiente se observa el valor contenido en cada uno de los registros en el siguiente orden:
$0, $t1, $t2, $t3, $t4, $t5, $t6, $t7, $t8, $s0, $s1, $s2, $s3, $s4, $s5, $s6, $s7.



EF	0	10	20	e	78	10	20	0	0	10	f	1c	0	4	5	14	1
i	$0	$t1	$t2	$t3	$t4	$t5	$t6	$t7	$t8	$s0	$s1	$s2	$s3	$s4	$s5	$s6	$s7
#	0	9	a	b	c	d	e	f	18	10	11	12	13	14	15	16	17

Los valores coinciden con el estado final del seguimiento.

VERIFICACIÓN: MEMORIA DE DATOS

De igual manera, agregamos al final 12 instrucciones **LW** para conocer el estado final de la memoria de datos. En la imágen siguiente se observa el valor contenido en cada una de las direcciones de memoria de datos:

EF	1	2	4	8	10	20	1	2	4	8	10	20
dir	00	04	08	c	10	14	18	1c	20	24	28	2c

Los valores coinciden con el estado final del seguimiento.