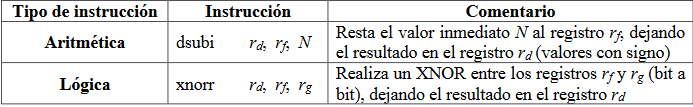
# Trabajo Práctico 10

Taciano Pacchialat

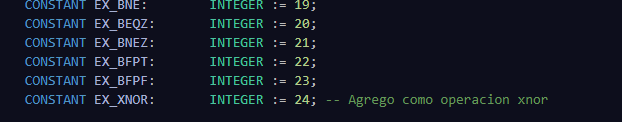
En este trabajo práctico se implementaron las siguientes instrucciones en el procesador TDA-1819, y se comprobó el correcto funcionamiento de las mismas mediante simulaciones.



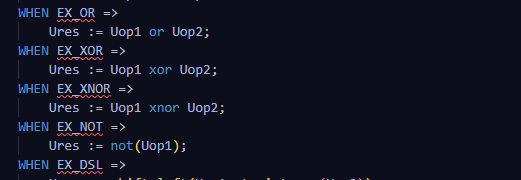
**Incorporación de Instrucciones**

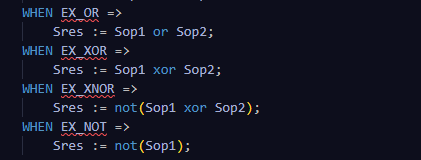
Luego de un extensivo análisis sobre el funcionamiento de la computadora y los módulos a modificar para la implementación de las dos instrucciones, se llevó a cabo el siguiente procedimiento:

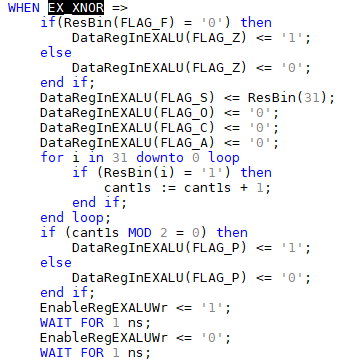
1. Incorporación de la constante de cpu “EX\_XNOR” con su respectivo código entero en el archivo *const\_cpu.vhd*.



1. Incorporación de la instrucción *EX\_XNOR* a la unidad aritmético-lógica en el archivo *excecute\_alu.vhd*, tanto para valores con signo como sin signo*:*





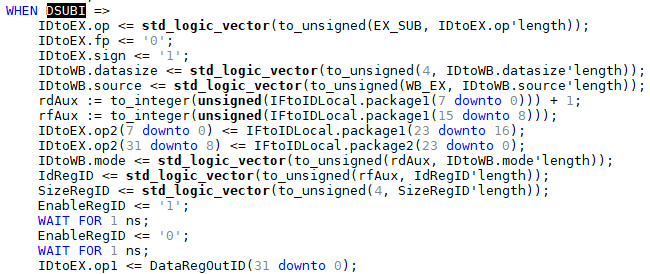
Y se implementó el manejo de flags

***Manejo de flags de EX\_XNOR***

1. Luego, se modificó la etapa *decode* para que la cpu reconozca las instrucciones nuevas (xnorr y dsubi) de la siguiente manera:



***Incorporación de XNORR en la etapa decode***



***Incorporación de DSUBI en la etapa decode***

1. Por último, se agregaron las instrucciones a las constantes del ensamblador (ver archivos adjuntos, la imagen es demasiado larga para la página del documento).

**Pruebas de Funcionalidad**

Una vez realizadas las modificaciones, se llevaron a cabo una serie de pruebas para verificar el funcionamiento de las nuevas instrucciones. Para ello, se desarrolló el programa *“testeo.asm”* adjuntado en la sección de archivos. En él, se realiza la operación aritmética de resta mediante *DSUBI* y la operación lógica xnor mediante *XNORR*, en conjunto con otras instrucciones de salto, carga y almacenamiento.

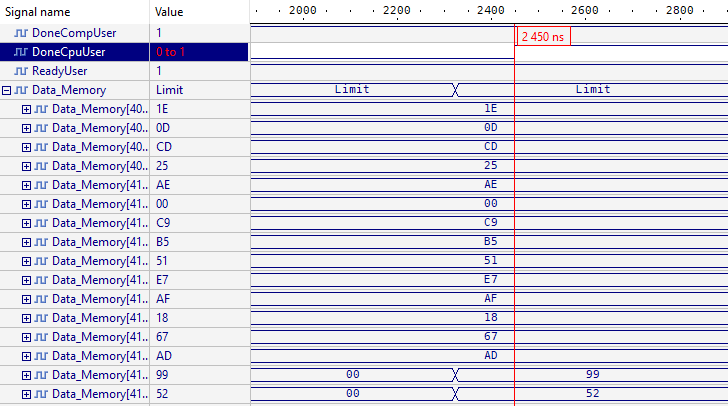
Los resultados esperados son los siguientes:

* 3358 - 9677 = -6319 (0xE751).
* 0xE751 xnor 0x00A7 = 0x1809.
* 9677 - 3358 = 6319 (0x18AF).
* 0x18AF xnor 0xB5C9 = 0x5299.

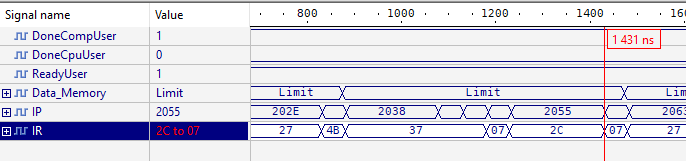
Poniendo el foco en los tiempos de ejecución de cada instrucción, los identificadores de las instrucciones implementadas son:

* DSUBI: 0x2C.
* XNORR: 0x37.

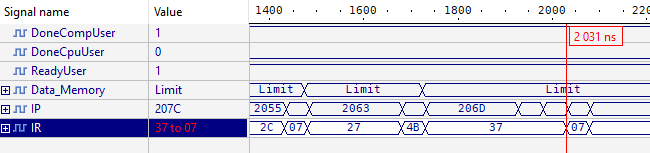
A continuación se exponen imágenes de la simulación realizada:

****

En esta simulación, el **cauce segmentado estaba activado** y como se observa, el tiempo de ejecución fué de 2450 ns. Aquí se puede notar cómo la mayoría de los resultados son los correctos.



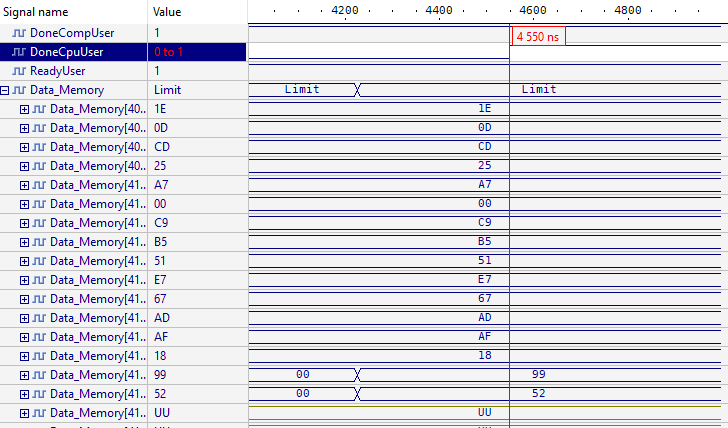
Aquí se aprecia la instrucción DSUBI, con un tiempo de 1431 ns - 1231 ns = 200 ns. Ahora se muestra la instrucción XNORR:



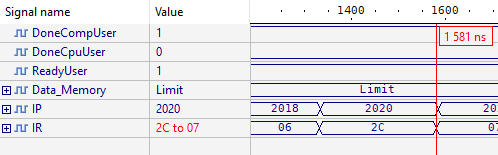
Con un tiempo de 2031 ns - 1731 ns = 300 ns.

**Nota 1:** no logré identificar el origen del error por el cual la primera operación de XNOR no tiene el resultado esperado, lo cual me resulta extraño ya que el segundo XNOR sí arroja el resultado correcto.

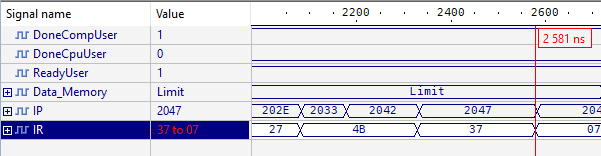
Ahora, se muestran los resultados con el **cauce segmentado desactivado**:



Ahora se aprecia que el tiempo de ejecución es de 4550 ns.



Aquí podemos observar mediante el IR (Instruction Register) la duración de la instrucción DSUBI, que se ejecuta desde los 1331 ns hasta los 1581 ns, dándonos un tiempo de ejecución total de 1581 - 1331 = 250 ns.



Ahora vemos la instrucción XNORR, que ocupa el IR desde los 2331 ns hasta los 2581 ns, y en total 2581 - 2331 = 250 ns de ejecución. Sorprendentemente, la instrucción XNORR es más lenta con el cauce activado.

**Relative Paths de los Archivos Entregados**

A continuación se detallan los paths relativos desde la carpeta “src” del workspace:

* **testeo.asm:** src/Assembler/testeo.asm
* **const\_cpu.vhd:** src/Packages/const\_cpu.vhd
* **reprert\_cpu.vhd:** src/Packages/repert\_cpu.vhd
* **const\_ensamblador.vhd:** src/Packages/const\_ensamblador.vhd
* **decode.vhd:** src/Usuario/PC/2. CPU/2. Etapas/2. Decode/decode.vhd
* **execute\_alu.vhd:** src/Usuario/PC/2. CPU/2. Etapas/3. Execute/1. ALU/execute\_alu.vhd
* **usuario.vhd:** src/Usuario/usuario.vhd