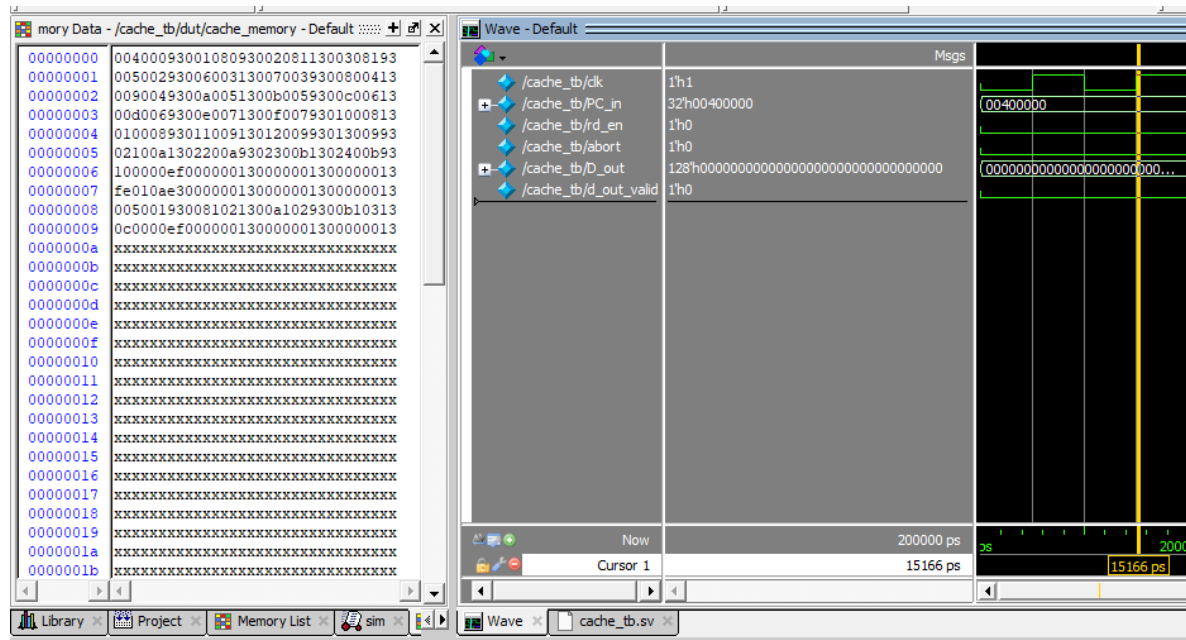


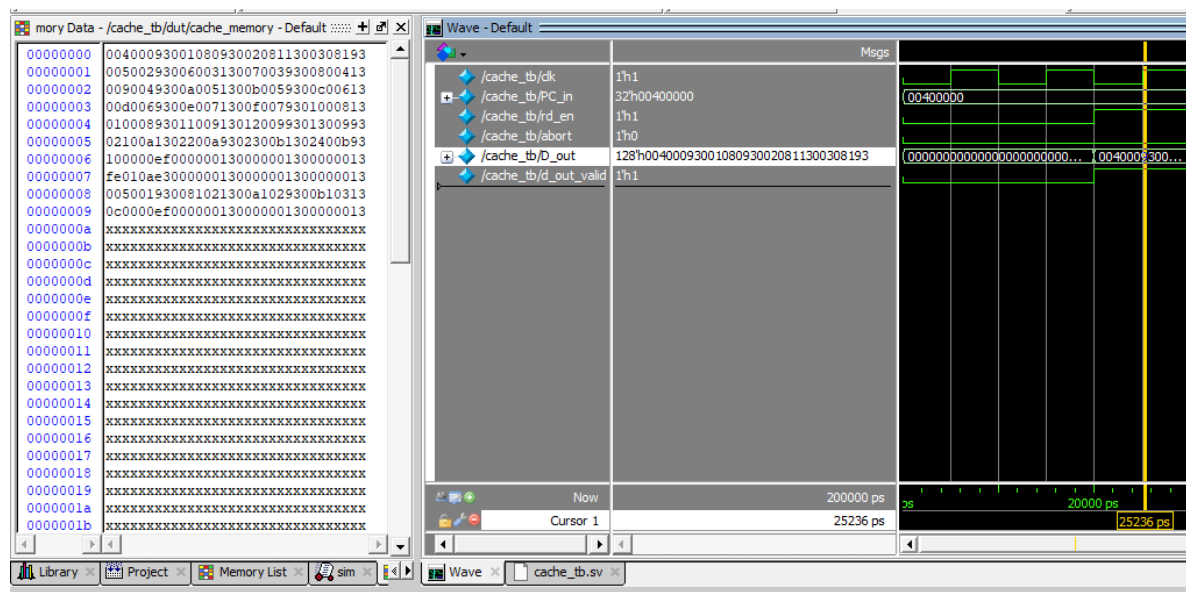
Realizado por: Luis Alberto Mena González

I-Cache

Primero se validó que la memoria cache no entregara datos cuando la señal de rd_en = 0.



Posteriormente se validó que se pudiera leer habilitando la señal de rd_en.



The screenshot displays the Xilinx ISE Waveform Editor interface. The top window, titled 'Memory Data - /cache_tb/dut/cache_memory - Default', shows a memory dump with addresses from 00000000 to 0000001b and their corresponding hexadecimal values. The bottom window, titled 'Wave - Default', shows a logic analyzer view with a table of signals and their values over time. The signals listed are /cache_tb/dk, /cache_tb/PC_in, /cache_tb/rd_en, /cache_tb/abort, /cache_tb/D_out, and /cache_tb/d_out_valid. The time scale is set to 200000 ps, and the cursor is positioned at 35063 ps. The waveform shows the signals changing over time, with some signals having a '00...' prefix, indicating a multi-bit signal.

Address	Value
00000000	00400093001080930020811300308193
00000001	00500293006003130070039300800413
00000002	0090049300a0051300b0059300c00613
00000003	00d0069300e0071300f0079301000813
00000004	01000893011009130120099301300993
00000005	02100a1302200a9302300b1302400b93
00000006	100000e0000000130000001300000013
00000007	fe010ae3000000130000001300000013
00000008	005001930081021300a1029300b10313
00000009	0c0000e0000000130000001300000013
0000000a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000011	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000012	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000015	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000017	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000018	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Signal	Value
/cache_tb/dk	1'h1
/cache_tb/PC_in	32'h00400004
/cache_tb/rd_en	1'h1
/cache_tb/abort	1'h1
/cache_tb/D_out	128'h00000000000000000000000000000000
/cache_tb/d_out_valid	1'h0

Time: 200000 ps
Cursor 1: 35063 ps

[illegible]

```

VSIM 25> run
# =====
# TESTBENCH: i_cache
# =====
# PC_in = 0x00400000 -> CACHE_LINE[0] = 00000000000000000000000000000000 (valid=0)
#
# --- Abort enabled ---
# PC_in = 0x00400004 -> CACHE_LINE[0] -> D_out = 00000000000000000000000000000000 valid=0 (esperado: invalidado)
#
# --- RD Disabled ---
# PC_in = 0x00400008 -> CACHE_LINE[0] -> D_out = 00000000000000000000000000000000 valid=0 (esperado: invalidado)
#
# --- Reading other line different from first one ---
# PC_in = 0x004000fc -> CACHE_LINE[15] = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (valid=1)
#
# --- Reading other line different from first one ---
# PC_in = 0x00400040 -> CACHE_LINE[4] = 01000893011009130120099301300993 (valid=1)
#
# =====
# END
# =====

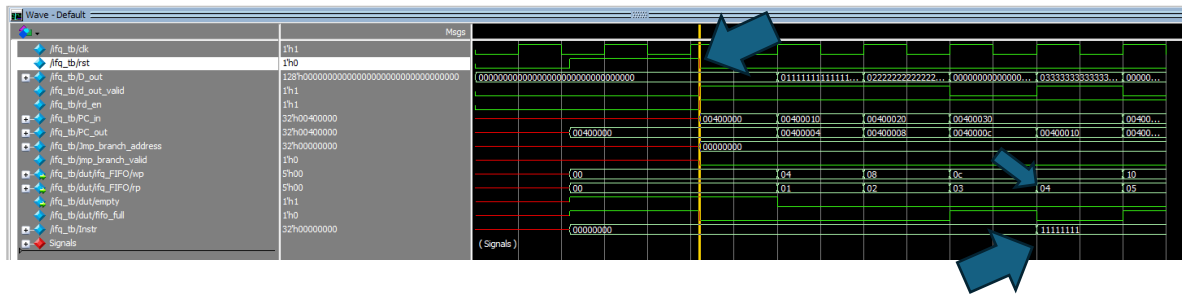
```

IFQ

El primer paso para ver el funcionamiento correcto de la IFQ, se inicializa la memoria cache de instrucciones:

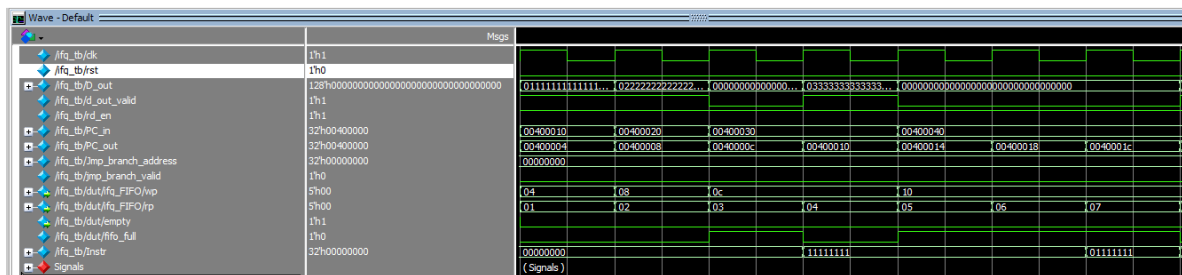
Address	Data
00000000	00000000000000000000000000000000
00000001	01111111111111111111111111111111
00000002	02222222222222222222222222222222
00000003	03333333333333333333333333333333
00000004	04444444444444444444444444444444
00000005	05555555555555555555555555555555
00000006	06666666666666666666666666666666
00000007	07777777777777777777777777777777
00000008	08888888888888888888888888888888
00000009	09999999999999999999999999999999
0000000a	0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
0000000b	0bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
0000000c	0cccccccccccccccccccccccccccccc
0000000d	0dddddddddddddddddddddddddddddd
0000000e	0eeeeeeeeeeeeeeeeeeeeeeeeeeeeee
0000000f	0fffffffffffffffffffffffffffffff
00000010	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000011	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000012	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000013	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000015	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000016	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000017	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000018	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001a	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001b	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001f	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000020	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000021	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000022	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000023	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000024	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000025	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000026	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000027	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000028	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Primero validaremos que después de un ciclo de reset no se pierdan ciclos y el mux de bypass de la primera instrucción funcione correctamente.

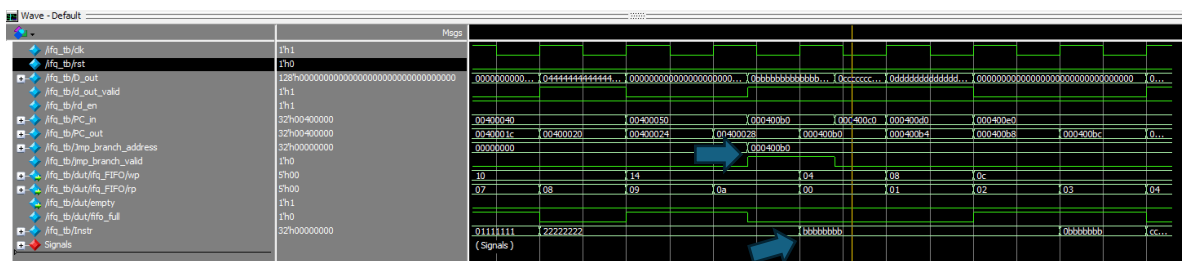


Comprobamos que justo después de un reset se tiene la instrucción (en este caso como inicialicé la primer línea de la memoria cache con 0s se debe ver durante rp 0 a 3 puros 0s y posteriormente cambiar a 1s que es la siguiente línea de la memoria cache).

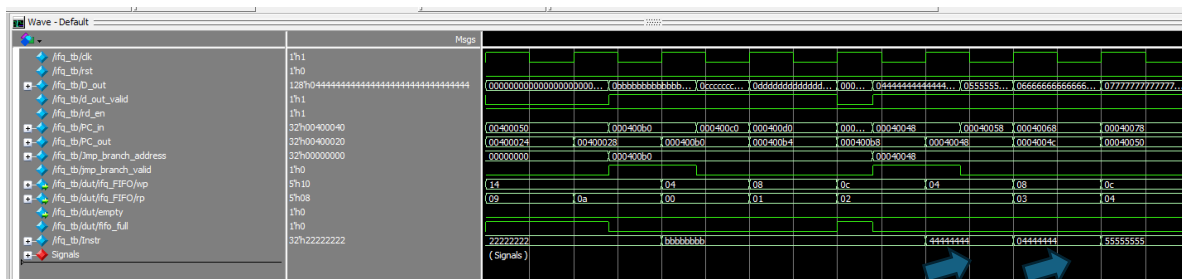
Se ve que se muestra la siguiente línea de la memoria cache en la instrucción:



Ahora realizaré un Branch valid y por medio del testbench inyectare un PC para ver el flush de la FIFO y la nueva instrucción salir de la IFQ. En este caso la dirección del program counter esta alineada con el inicio de una línea de memoria cache.



Otro caso de prueba será brincar no al principio de una línea de la cache si no a una dirección diferente (ejemplo 0x400048).



Debido a que no fue un brinco alineado con la memoria cache, se ve solamente la dirección 0x400048 y 0x40004C.