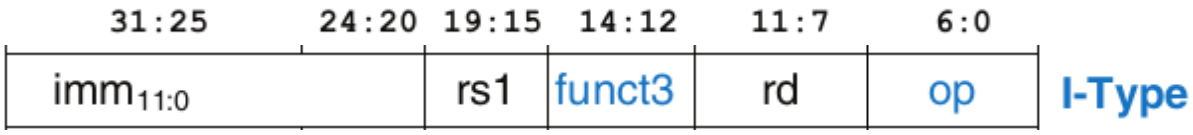
TASK: Verify that these 32 bits (0x0042a303) correspond to instruction `lw t1,4(t0)` in the RISC-V architecture.

0x0042a303 → 000000000100 00101 010 00110 0000011

$imm_{11:0}$ = 000000000100
rs1 = 00101 = x5 (t0)
funct3 = 010
rd = 00110 = x6 (t1)
op = 0000011

From Appendix B of DDCARV:



op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	010	–	I	lw rd, imm(rs1)	load word	rd = [Address]31:0

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary variables
s0/fp	x8	Saved variable / Frame pointer
s1	x9	Saved variable
a0-1	x10-11	Function arguments / Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved variables
t3-6	x28-31	Temporary variables

TASK: Perform a more detailed analysis of the `lw` instruction, similar to the one performed in Section 2.B of Lab 12 for an `add` instruction, or to the one performed in Section 2.B of Lab 13 of the RVfpga package for VeeR EL2.

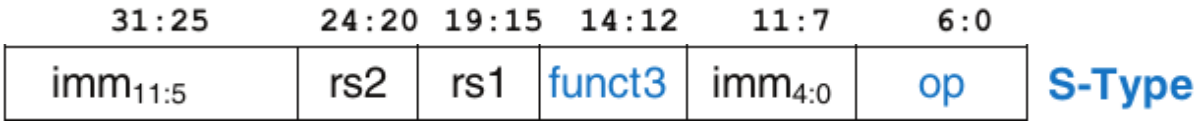
Solution not provided for this exercise.

TASK: Verify that these 32 bits (0x0062a023) correspond to instruction `sw t1,0(t0)` in the RISC-V architecture.

0x0062a023 → 0000000 00110 00101 010 00000 0100011

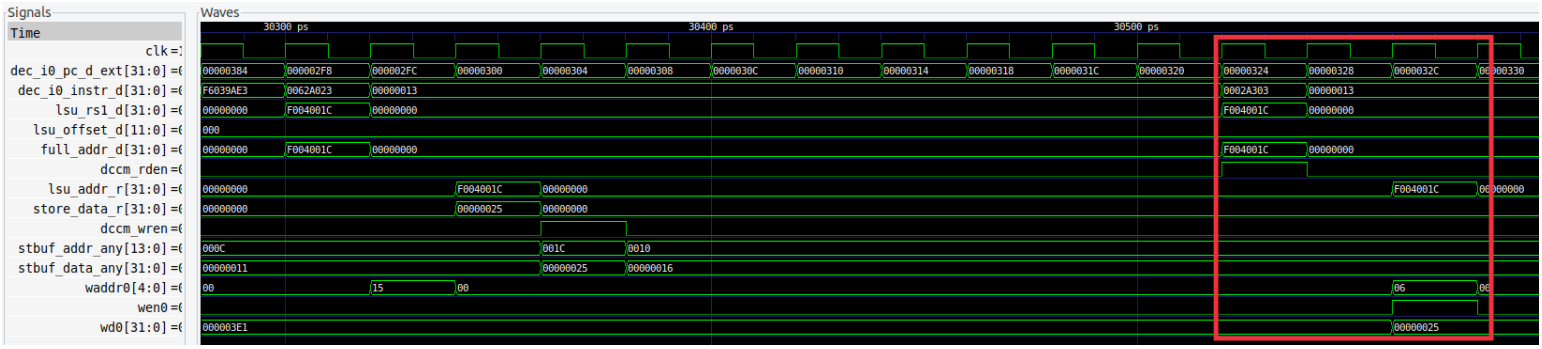
$imm_{11:0}$ = 000000000000
rs2 = 00110 = x6 (t1)
rs1 = 00101 = x5 (t0)
funct3 = 010
op = 0100011

From Appendix B of DDCARV:



op	funct3	funct7	Type	Instruction	Description	Operation
0100011 (35)	010	–	S	sw rs2, imm(rs1)	store word	[Address]$_{31:0}$ = rs2

TASK: Analyse in the simulation the load instruction that follows the store to verify that the value has been correctly written to the DCCM.



TASK: Extend the basic analysis performed in this section for the `sw` instruction in a similar way as the advanced analysis performed for the `lw` instruction in Section 2.B.

Solution not provided for this exercise.

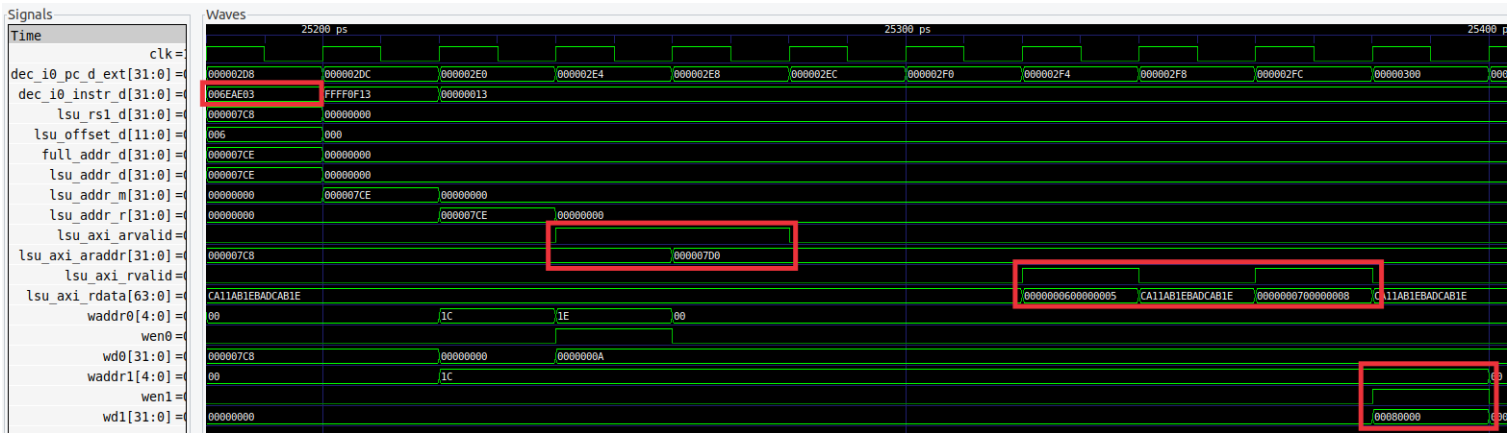
TASK: Analyse unaligned stores to the DCCM, as well as sub-word stores: store byte (`sb`) or *store half-word* (`sh`).

Solution not provided for this exercise.

TASK: Modify the program from **Error! Reference source not found.** in order to analyse an unaligned load access that needs to send two addresses to the Main Memory through the AXI Bus and then combine the 64-bit data.

We simulate the following modified program:

```
REPEAT:
lw t3, 6(t4)
add t5, t5, -1
INSERT_NOPS_10
add t6, t3, t6
add t4, t4, 4
INSERT_NOPS_9
bne t5, zero, REPEAT # Repeat the loop
INSERT_NOPS_4
```



TASK: It can be interesting to analyse the AXI Bus implementation for accessing the DRAM Controller, for which you can inspect the **lsu_bus_intf** module.

Solution not provided for this exercise.