# TASKS

**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**

**imm11:0 = 000000000100**

**rs1 = 00101 = x5 (t0)**

**funct3 = 010**

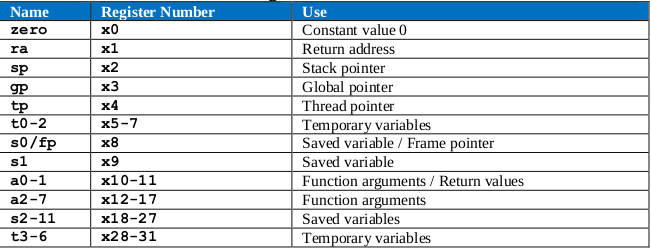
**rd = 00110 = x6 (t1)**

**op = 0000011**

From Appendix B of DDCARV:





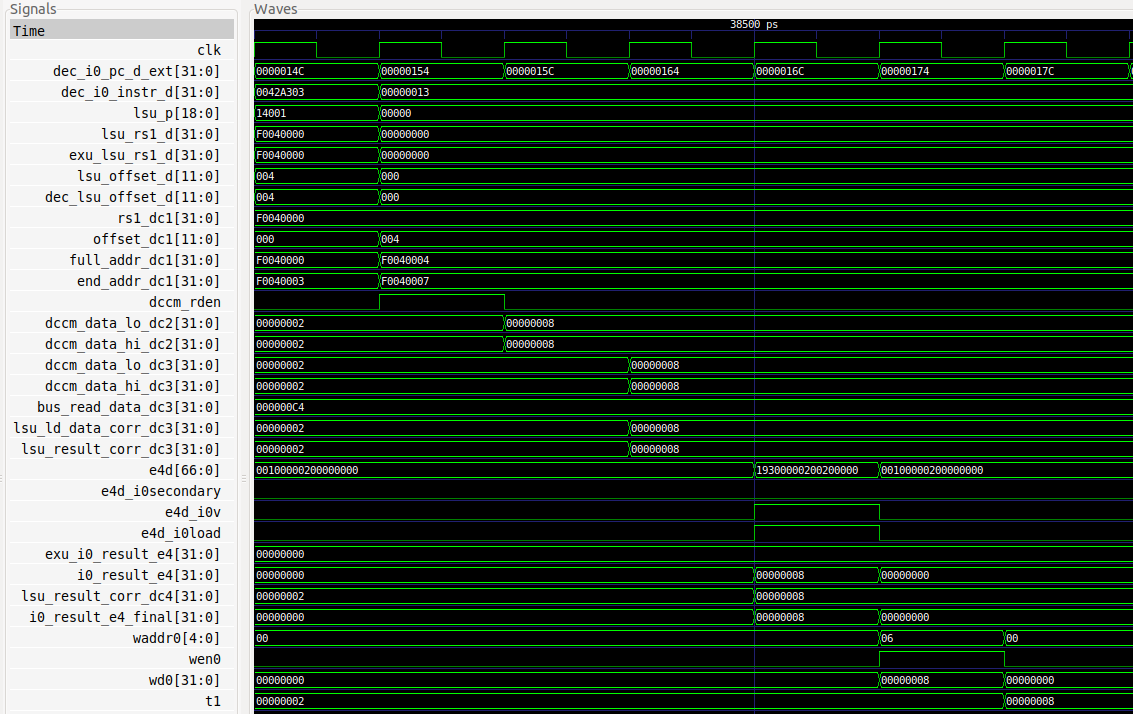


**TASK:** Replicate the simulation from Figure 4 on your own computer. Follow the next steps (as described in detail in Section 7 of the GSG):

* If necessary, generate the simulation binary (*Vrvfpgasim*).
* In PlatformIO, open the project provided at: *[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_DCCM*.
* Correct the path to the RVfpga simulation binary (*Vrvfpgasim*) in file *platformio.ini*.
* Generate the simulation trace with Verilator (Generate Trace).
* Open the trace using GTKWave.
* Use file *scriptLoad.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_DCCM/*) to open the same signals as the ones shown in Figure 4. For that purpose, on GTKWave, click on *File → Read Tcl Script File* and select the *scriptLoad.tcl* file.
* Click on *Zoom In* () several times and move to 18600ps.

Solution provided in main document of Lab 13.

**TASK:** Extend the simulation from Figure 4 to include the signals shown in Figure 6, which are explained below.



**TASK:** Locate the structures and signals from Figure 6 in the Verilog files of the SweRV EH1 processor.

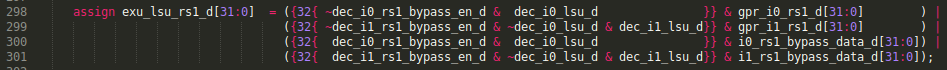
Solution not provided.

**TASK:** Include signal *lsu\_p* in the simulation from Figure 4 and analyse its bits according to this description.

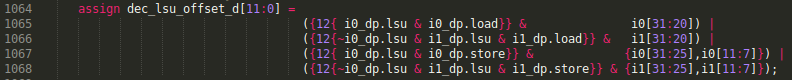
See the simulation above. We can see that when the load is decoded lsu\_p = 0x14001:

* valid = 1. The instruction is valid.
* load = 1. It is a load.
* word = 1. The size of the access is word.

**TASK:** Analyse in the Verilog code the path followed by the two inputs to the LSU (exu\_lsu\_rs1\_d and dec\_lsu\_offset\_d) from the sources where they are obtained. Several modules are involved in this process: **dec**, **exu**, **lsu**. Analyse the behaviour of these signals for other instructions.



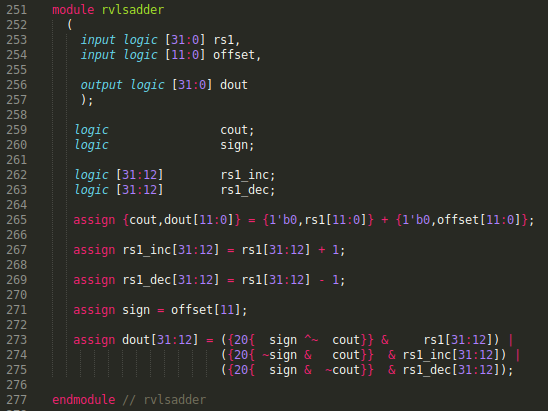
The base address can come from the Register File or from the Bypass, either from Way-0 or Way-1.



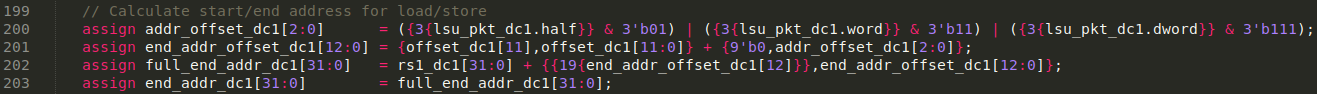
The offset comes from the 32 bits of the instruction at Way-0 or Way-1.

**TASK:** Analyse the implementation of the two adders from the DC1 stage, which are instantiated in module **lsu\_lsc\_ctl**. We provide guidance in Figure 7 below by showing the implementation of these adders.

**File *beh\_lib.sv*:**

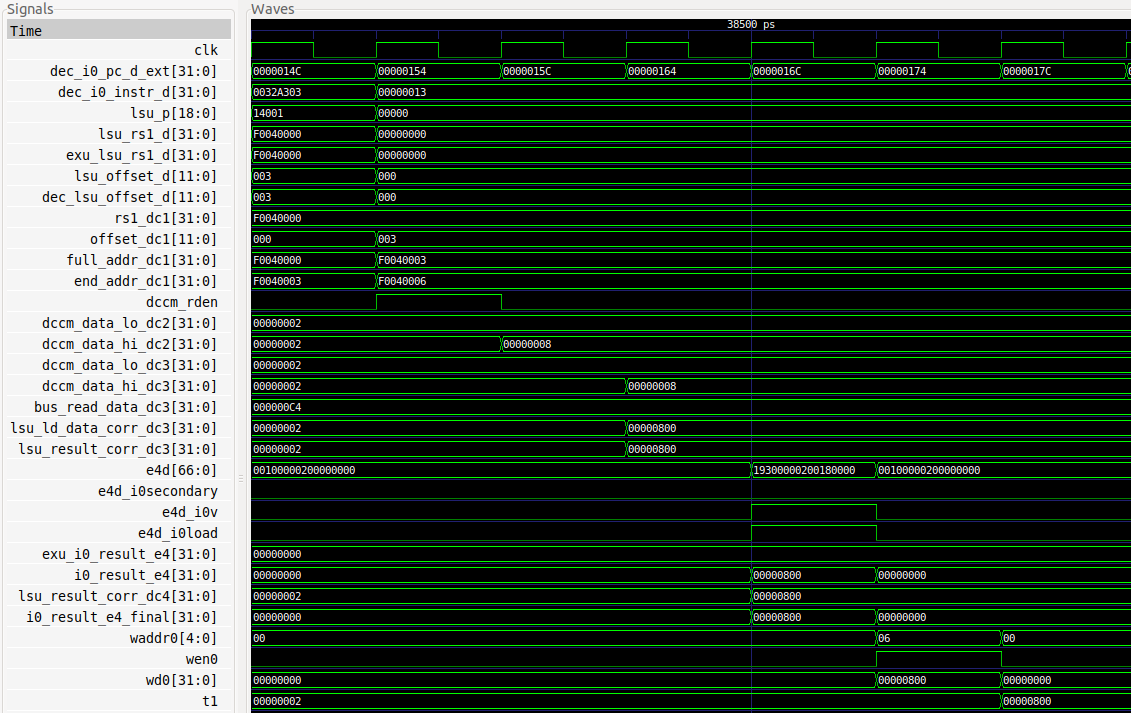


**File *lsu\_lsc\_ctl.sv*:**



**TASK:** In the program from Figure 2, try different access sizes (byte, half-word) and unaligned accesses. To do so, change the offset or the access type from lw to lb (*load byte*) or lh (*load half-word*). For example, if you change the offset from 4 to 3, the load word instruction performs an unaligned access to the 32-bits starting at address 0xF0040003, as shown in Figure 8. Analyse the value of signals lsu\_addr\_dc1[31:0] (or full\_addr\_dc1[31:0]) and end\_addr\_dc1[31:0] under these different situations.

In Lab 20 we analyse this situation from the internals of the DCCM.



The values of signals lsu\_addr\_dc1[31:0] and end\_addr\_dc1[31:0] communicate to Memory the starting and final address of the access: 0xF0040003 and 0xF0040007. Two words are read (0x00000002 and 0x00000008) and the final word is extracted in the Aligner (0x00000800).

**TASK:** In the program from Figure 2, compare the value of signals dccm\_data\_lo\_dc2[31:0] and dccm\_data\_hi\_dc2[31:0] when doing a lw to address 0xF0040004 and to address 0xF0040003.

Above you can see the two simulations.

* lw to address 0xF0040004

dccm\_data\_lo\_dc2[31:0]: 0x00000008

dccm\_data\_hi\_dc2[31:0]: 0x00000008

Both signals contain the value read from the requested address.

* lw to address 0xF0040003

dccm\_data\_lo\_dc2[31:0]: 0x00000002 (value from address 0xF0040000)

dccm\_data\_hi\_dc2[31:0]: 0x00000008 (value from address 0xF0040004)

**TASK:** Analyse the Align, Merge, and Error Check logic used in the Verilog code in modules **lsu\_dccm\_ctl** and **lsu\_ecc**.

Solution not provided.

**TASK:** In the program from Figure 2, compare the value of signal lsu\_result\_corr\_dc3[31:0] when doing a lw to address 0xF0040004 and to address 0xF0040003.

Above you can see the two simulations.

* lw to address 0xF0040004

lsu\_result\_corr\_dc3[31:0]: 0x00000008

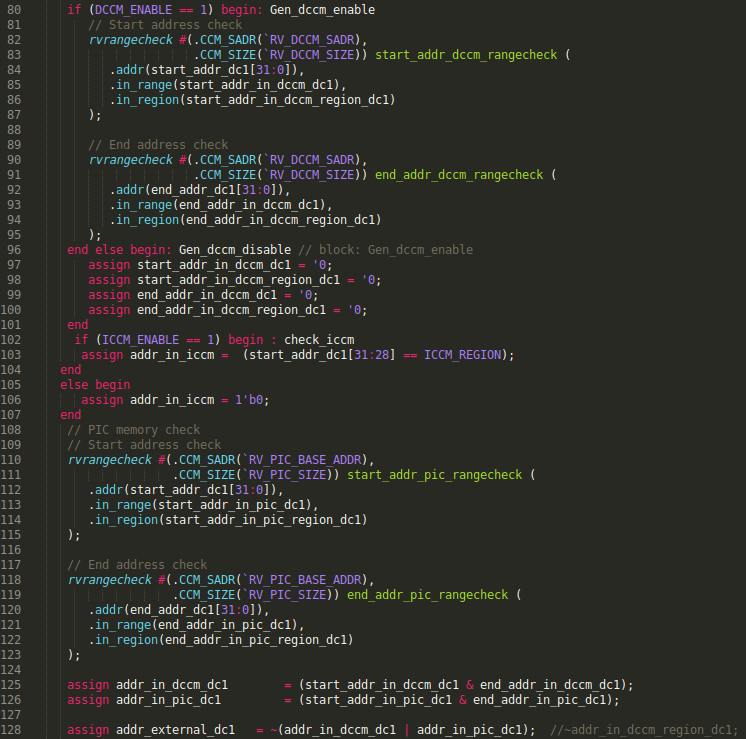
It contains the value read from the requested address.

* lw to address 0xF0040003

lsu\_result\_corr\_dc3[31:0]: 0x00000800

It contains the value read from the requested address. Take into account that RISC-V is little-endian.

**TASK:** Analyse in the Verilog code how signal addr\_external\_dc1 was computed in the DC1 stage in module **lsu\_addrcheck**.



Module **rvrangecheck** is used to check the requested address:

* If it is within the DCCM/ICCM address range (lines 80-107), in which case signal addr\_in\_dccm\_dc1 = 1
* If it is within the PIC address range (lines 108-123), in which case signal addr\_in\_pic\_dc1 = 1
* If it is not in any of these address ranges, then it is at the DDR External Memory, in which case: addr\_external\_dc1 = 1

**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**

**imm11:0 = 000000000000**

**rs2 = 00110 = x6 (t1)**

**rs1 = 00101 = x5 (t0)**

**funct3 = 010**

**op = 0100011**

From Appendix B of DDCARV:

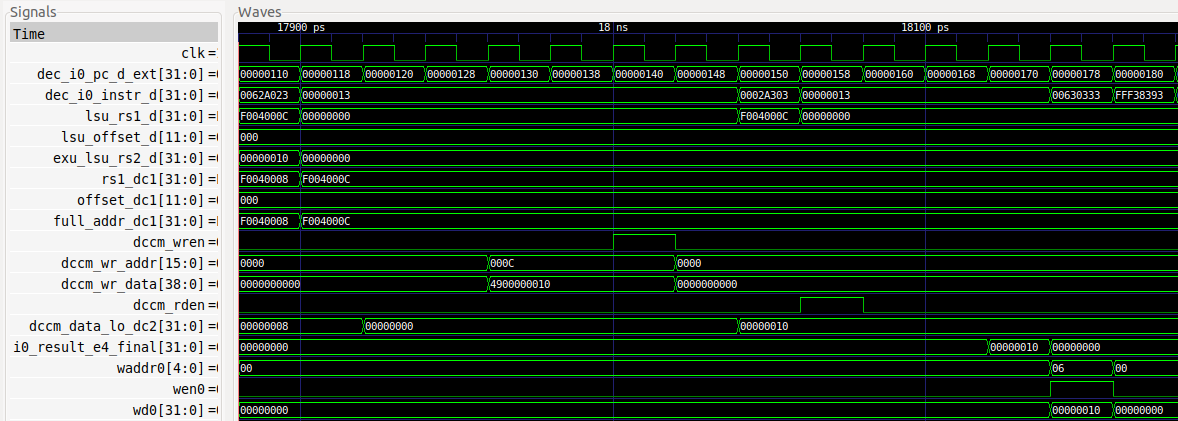


**TASK:** Replicate the simulation from Figure 12 on your own computer. Follow the next steps (as described in detail in Section 7 of the GSG):

* If necessary, generate the simulation binary (*Vrvfpgasim*).
* Open in PlatformIO the project provided at: *[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM*.
* Update the path to the RVfpga simulation binary (*Vrvfpgasim*) in file *platformio.ini*.
* Generate the simulation trace with Verilator (Generate Trace).
* Open the trace on GTKWave.
* Use file *scriptStore.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM/*) to display the same signals as the ones shown in Figure 4. For that purpose, in GTKWave, click on *File → Read Tcl Script File* and select the *scriptStore.tcl* file.
* Click on *Zoom In* () several times and move to 17900ps.

Solution provided in the main document of Lab 13.

**TASK:** Analyse in the simulation the load instruction that follows the store to verify that the value has been correctly written to the DCCM. You will need to add some of the signals from Figure 4 and Figure 6 to analyse the load.



**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.

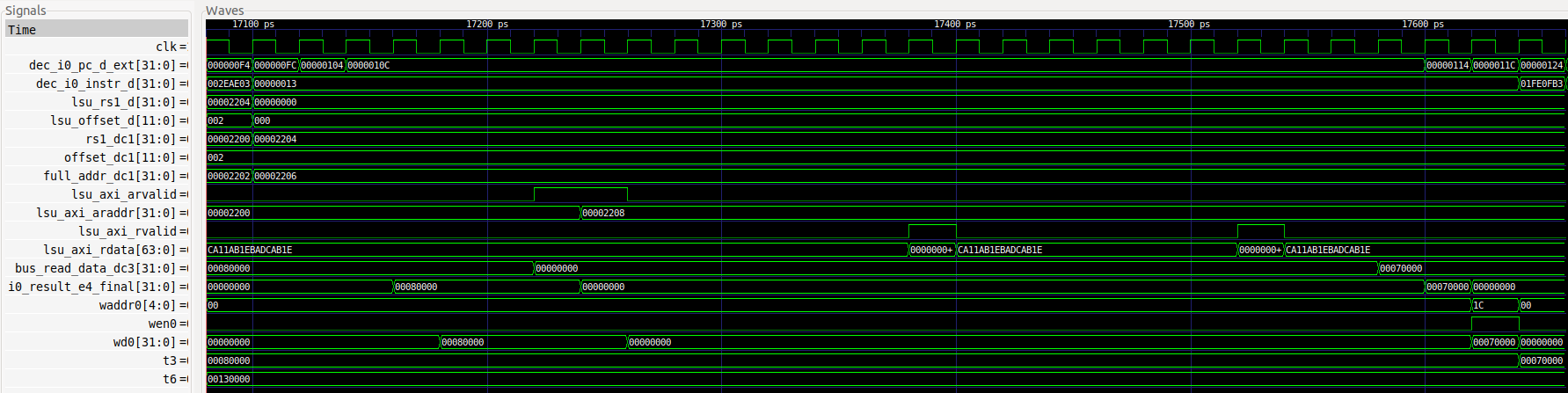
**TASK:** Analyse unaligned stores to the DCCM, as well as sub-word stores: store byte (sb) or *store half-word* (sh).

Solution not provided.

**TASK:** Replicate the simulation from Figure 17 on your own computer. Use file *test\_Blocking.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_ExtMemory*). *Zoom In* () several times and move to 16940ps.

Solution provided in the main document of Lab 13.

**TASK:** Modify the program from Figure 15 in order to analyse an unaligned load access that needs to send two addresses to the External Memory through the AXI Bus.

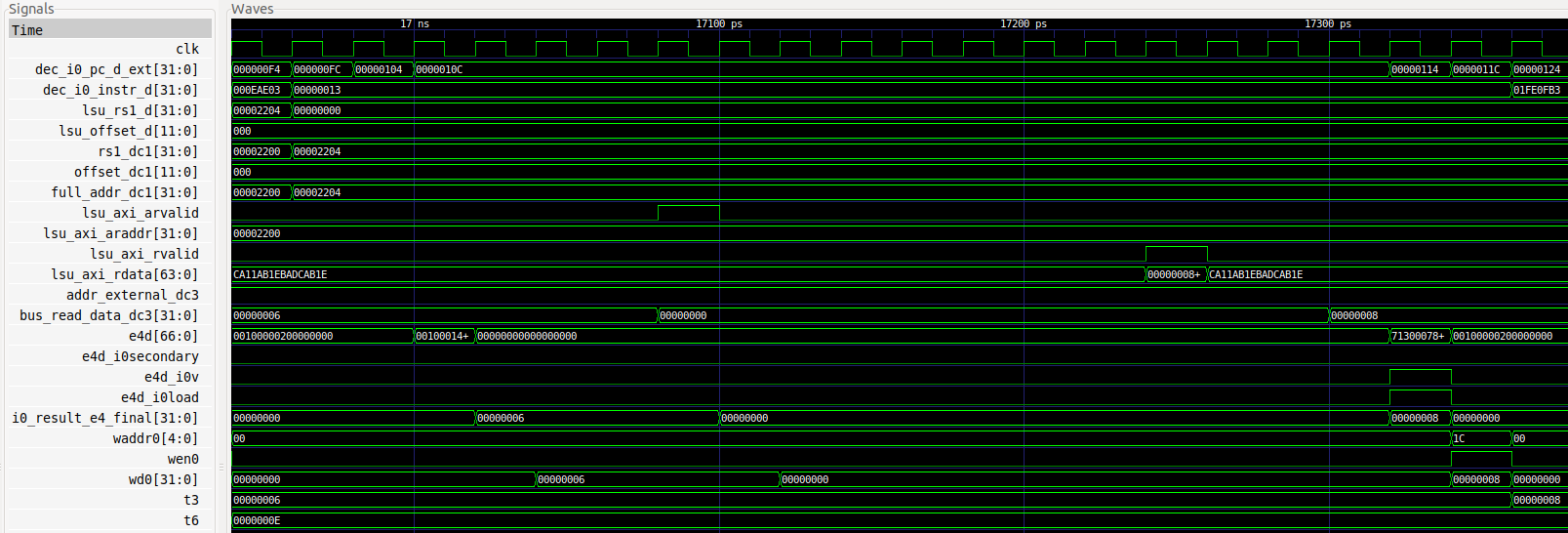


**TASK**: Add to the simulation the signals that control the multiplexers (in the DC3 and Commit stages in Figure 16) that select the data provided by the DDR External Memory. You can find these multiplexers at the following lines of the Verilog code:

- 2:1 Multiplexer: Line 264 of module **lsu\_lsc\_ctl**.

- 3:1 Multiplexer: Line 2277 of module **dec\_decode\_ctl**.

A *.tcl* file that you can use is provided at: *[RVfpgaPath]/RVfpga/Labs/Lab13/LW\_Instruction\_ExtMemory/test\_Blocking\_Extended.tcl*



**TASK:** It can also 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.

**TASK:** Replicate the simulation from Figure 18 on your own computer. Use file *scriptStoreBuffer.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab13/SW\_Instruction\_DCCM*). *Zoom In* () several times and move to 17900ps.

Solution provided in the main document of Lab 13.

**TASK:** Modify the program from Figure 11 in order to have two outstanding stores and perform a similar analysis to the one from Figure 18.

Solution not provided.