Learning Activity 2

By Alex Clunan

ECE 4435 Computer Architecture & Design

February 10, 2025

1 Problem Statement

Design and Implement a read data and write data component to process stores and loads to and from memory. The read component must be able to load 32 bit words, 16 bit half words with zero extend and sign extend parameters, and bytes with zero extend and sign extend parameters. The write component must be able to store words, half words, and bytes. In addition to this, assembly must be written that implements that stores values from registers in memory.

2 Analytical Design

The first stage of the read data component required selecting between four different byte positions, and two 16 bit half word positions. This is because data is read in 32 bit increments, and different bytes of the same increment need to be able to be selected. A 4-1 MUX is used to select the correct byte, and a 2-1 MUX is used to select the correct half word.

These outputs then enter the zero-extend and sign-extend circuits. Both circuits assign the LSBs of size WIDTH to the output, and fill the MSBs with zeros for zero-extend or all ones for sign-extend. Both components can be adjusted to any size and have a configurable number of zeros/ones and pass through width. The extend circuit outputs for the byte case are then connected to a 3-1 MUX with the zero-extend output, sign-extend output, and 32 bit word input. The half word case uses a 2-1 MUX that selects between the zero-extend and sign-extend outputs. The output of these two MUXs are then connected to a final 2-1 MUX which gives the output of the read data component. The full schematic of the read data component is shown in Fig. 1.

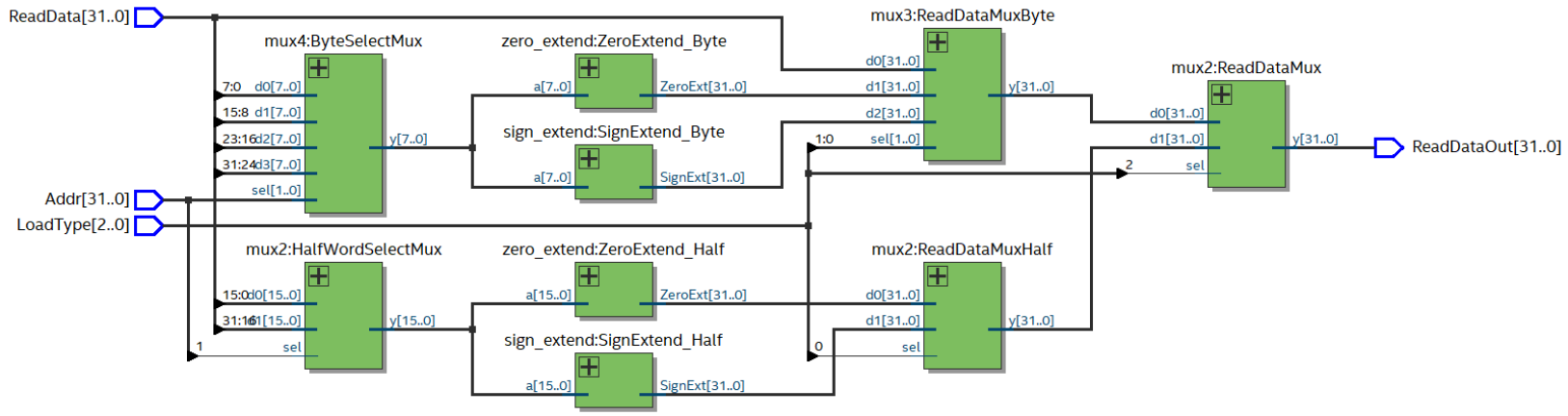

Fig 1. Read Data Schematic

Next, the write data component was designed. It writes a byte, half-word, and word based on the type of store, and the desired write address to the desired portion of memory. The circuit starts with wire components that route each byte to the correct address. For example, Byte 0 takes up positions [7:0] where Byte 2 takes up positions [23:16] in the 32 bit number. The

excess bits are filled in with a value from ReadData. The bytes are selected by a 4-1 MUX using the last two bits of the address as the select value. The output of this byte select MUX then outputs to another 2-1 MUX that chooses between outputting a byte or the full register based on the type of store requested.

The half word selection follows the same principle as the byte selection. It uses a 2-1 MUX that selects between the upper and lower 16 bits of the input where the input is put over the Read Data. The final 2-1 MUX selects between the byte-word and half-word outputs are results in the outputted results. The schematic of the write data component is shown in Fig. 2.

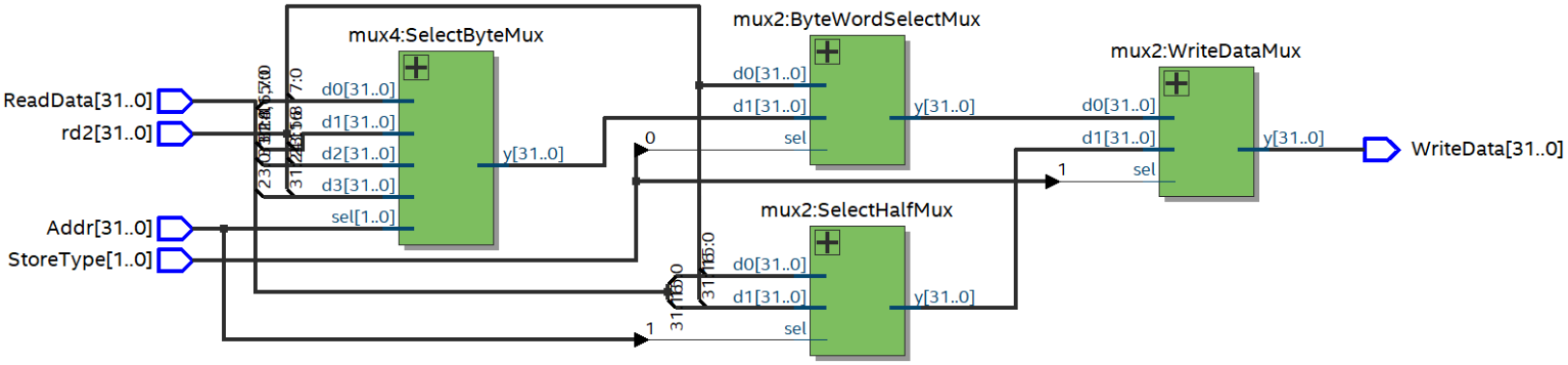

Fig. 2. Write Data Schematic

Finally, some assembly code was generated to test the load and store functions in the future. The code sets all usable registers to their respective values (i.e. r2 = 2), and then stores them in memory. The code was tested in the RARs 1.6 RISC-V simulator.

3 Numerical Verification

Both the read data and write data testbenches test all configurations of the circuit. The read data testbench tests all variations of selecting the correct byte, half-word, and word, as well as using the sign and zero extend circuits. The results of the read data testbench are shown in Figs. 3 & 4.

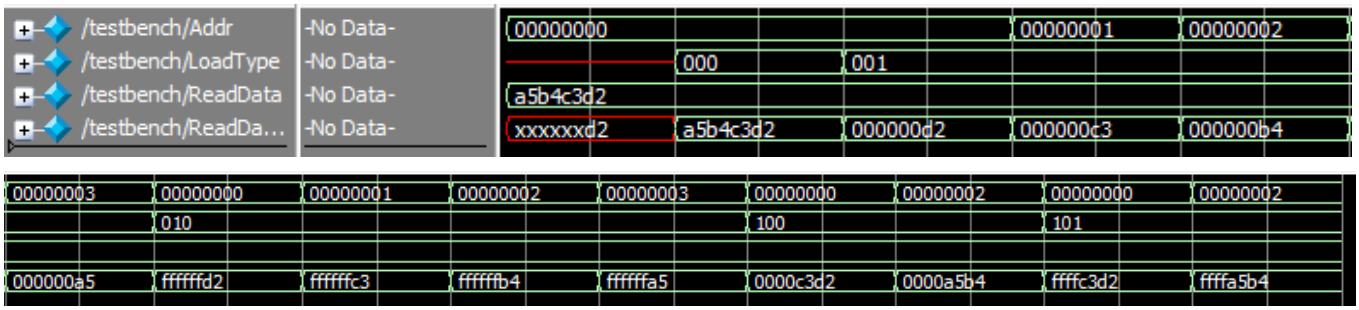

Fig. 3. Waveform Output of Read Data Testbench

```
#
# PASSED
# ACTUAL a5b4c3d2
# EXPECTED a5b4c3d2
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read the 32-bit word
# TIME 20
#
# PASSED
# ACTUAL 000000d2
# EXPECTED 000000d2
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 0, zero extend
# TIME 30
#
# PASSED
# ACTUAL 000000c3
# EXPECTED 000000c3
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 1, zero extend
# TIME 40
#
# PASSED
# ACTUAL 000000b4
# EXPECTED 000000b4
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 2, zero extend
# TIME 50
#
# PASSED
# ACTUAL 000000a5
# EXPECTED 000000a5
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 3, zero extend
# TIME 60
#
# PASSED
# ACTUAL ffffffd2
# EXPECTED ffffffd2
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 0, sign extend
# TIME 70
#
# PASSED
# ACTUAL ffffffc3
# EXPECTED ffffffc3
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 1, sign extend
# TIME 80
#
# PASSED
# ACTUAL ffffffb4
# EXPECTED ffffffb4
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 2, sign extend
# TIME 90
#
# PASSED
# ACTUAL ffffffa5
# EXPECTED ffffffa5
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read byte 3, sign extend
# TIME 100
#
# PASSED
# ACTUAL 0000c3d2
# EXPECTED 0000c3d2
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read half word 0, zero extend
# TIME 110
#
# PASSED
# ACTUAL 0000a5b4
# EXPECTED 0000a5b4
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read half word 1, zero extend
# TIME 120
#
# PASSED
# ACTUAL ffffc3d2
# EXPECTED ffffc3d2
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read half word 0, sign extend
# TIME 130
#
# PASSED
# ACTUAL ffffa5b4
# EXPECTED ffffa5b4
# TESTED SIGNAL NAME ReadDataOut
# INPUTS ReadData = 32'ha5b4c3d2
# DESCRIPTION Read half word 1, sign extend
# TIME 140
# INFO End of tests
# TIME 140
```

Fig. 4. Numerical Verification of the Read Data Testbench

These figures show that the read data component passes all verification tests and works as intended.

Next, the write data component was tested with all byte, half-word, and word scenarios tested. The results of the testbench verification for the write data component are shown in Figs. 5 & 6.

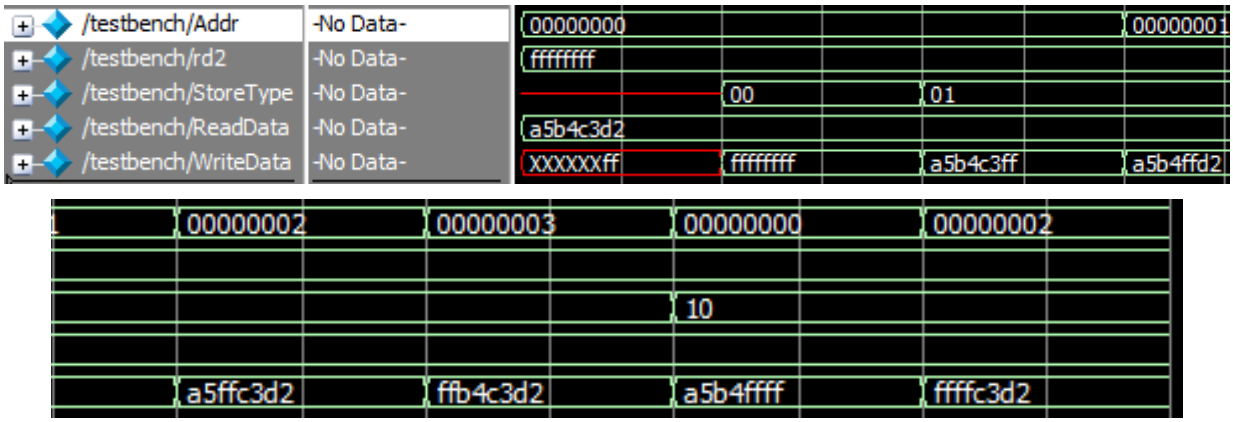

Fig. 5. Waveform Verification of the Write Data Component

```
#
# PASSED
# ACTUAL ffffffff
# EXPECTED ffffffff
# TESTED SIGNAL NAME WriteData
# INPUTS rd2 = 32'hffffffff
# DESCRIPTION Write 32-bit word in rd2
# TIME 20
#
# PASSED
# ACTUAL a5b4c3ff
# EXPECTED a5b4c3ff
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write byte 0 of rd2
# TIME 30
#
# PASSED
# ACTUAL a5b4ffd2
# EXPECTED a5b4ffd2
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write byte 1 of rd2
# TIME 40
#
# PASSED
# ACTUAL a5ffc3d2
# EXPECTED a5ffc3d2
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write byte 2 of rd2
# TIME 50
#
# PASSED
# ACTUAL ffb4c3d2
# EXPECTED ffb4c3d2
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write byte 3 of rd2
# TIME 60
#
# PASSED
# ACTUAL a5b4ffff
# EXPECTED a5b4ffff
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write half word 0 of rd2
# TIME 70
#
# PASSED
# ACTUAL ffffc3d2
# EXPECTED ffffc3d2
# TESTED SIGNAL NAME WriteData
# INPUTS ReadData = 32'ha5b4c3d2, rd2 = 32'hffffffff
# DESCRIPTION Write half word 1 of rd2
# TIME 80
```

Fig. 6. Numerical Verification of the Write Data Component

The tests shown in Figs. 5 & 6 show that the write data component passes all tests and works as intended.

Next, the assembly code was verified in RARs 1.6. First, all the registers corresponding values were loaded as shown in Fig. 7.

Name	Number	Value
zero	0	0x00000000
ra	1	0x000001f8
sp	2	0x00000002
gp	3	0x00000003
tp	4	0x00000004
t0	5	0x00000005
t1	6	0x00000006
t2	7	0x00000007
s0	8	0x00000008
s1	9	0x00000009
a0	10	0x0000000a
a1	11	0x0000000b
a2	12	0x0000000c
a3	13	0x0000000d
a4	14	0x0000000e
a5	15	0x0000000f
a6	16	0x00000010
a7	17	0x00000011
s2	18	0x00000012
s3	19	0x00000013
s4	20	0x00000014
s5	21	0x00000015
s6	22	0x00000016
s7	23	0x00000017
s8	24	0x00000018
s9	25	0x00000019
s10	26	0x0000001a
s11	27	0x0000001b
t3	28	0x0000001c
t4	29	0x0000001d
t5	30	0x0000001e
t6	31	0x0000001f
pc		0x000000e8

Fig. 7. Registers x0 - x31 Loaded with their Respective Register Numbers

Each Register contains its corresponding number. Next, the register values were loaded into memory at their corresponding address from the memory base pointer. Fig 8 shows the register values loaded up into memory.

Address	Value (+0)	Value (+4)	Value (+8)	Value
0x00002000	0x00000000	0x00000000	0x00000002	
0x00002020	0x00000008	0x00000009	0x0000000a	
0x00002040	0x00000010	0x00000011	0x00000012	
0x00002060	0x00000018	0x00000019	0x0000001a	

(+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f
0x00000013	0x00000014	0x00000015	0x00000016	0x00000017
0x0000001b	0x0000001c	0x0000001d	0x0000001e	0x00002000

Fig 8. Register Values Loaded Into Memory

This shows that the register values were correctly loaded into memory as all values are in ascending order and are the correct value. Finally, the registers were reset to 0 as shown in Fig. 9.

Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000200
sp	2	0x00000000
gp	3	0x00000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00002000
pc		0x000001e8

Fig. 9 Registers Reset to 0 After Memory Write

This verifies the assembly code.

4 Summary

The problem requirement for this assignment was to design a 32 bit read data circuit, 32 bit write data circuit, and assembly code that changes values of registers and stores values to memory. The read and write circuits support byte, half-word, and word sizes. The read data uses zero and sign extension circuits to pad byte and half-word types so that the data fits the 32 bit bus width. These circuits were implemented in Verilog using the previously designed mux2, mux3, and mux4 modules and the newly created sign-extend and zero-extend modules. “The design was verified numerically to confirm that the specified digital circuit design solution satisfies the problem requirements.” (from Example Submission for learning activity 0)