

HW4 Report

This report includes test runs of instructions in the processor. The simulation results are shown by giving the input files and output files.

Run 1:

addn: \$3 = \$3 + \$5 \$28 = 2

The screenshot shows the MARS MIPS32 simulator interface. The main window displays assembly code with columns for opcode, rs, rt, rd, and imm. The code includes instructions like `addn`, `subn`, `andn`, `orn`, `sw`, `lw`, and `beq`. A secondary window titled 'registers.txt - Not Defteri' shows the binary representation of the assembly code, with each instruction's binary value listed.

The screenshot displays the OpenCAPO environment with three windows open:

- someins.txt - Not Defteri:** Shows assembly code with mnemonics and their corresponding register values. The code includes instructions like `addn`, `subn`, `andn`, `orn`, `sw`, `lw`, and `beq`.
- registers.txt - Not Defteri:** Displays a list of registers (e.g., `Regs...Internal`, `Integer Internal`) and their values, which are mostly zeros.
- register_outp.mem - Not D...:** Shows the output of the simulation, including memory data file information and the format of the output.

The screenshot shows the Verilog IDE with the following components:

- Main Window (someins.txt - Not Deftari):** Displays the assembly code for the MIPS32 testbench. The code is organized into columns: opcode, rs\$, rt, rd/imm. The 'and' instruction is highlighted in blue.
- Registers Window (registers.txt - Not Deftari):** Shows the internal state of the registers. The 'i' register is highlighted in blue. The window also displays the output of the 'register_outp.mem' file, which contains the memory dump for the 'memory data file'.
- Objects Window:** Shows the internal state of the objects, including 'clk', 'i', and 'instruction_set'.
- Design Unit Window:** Shows the design unit type, visibility, and other properties.

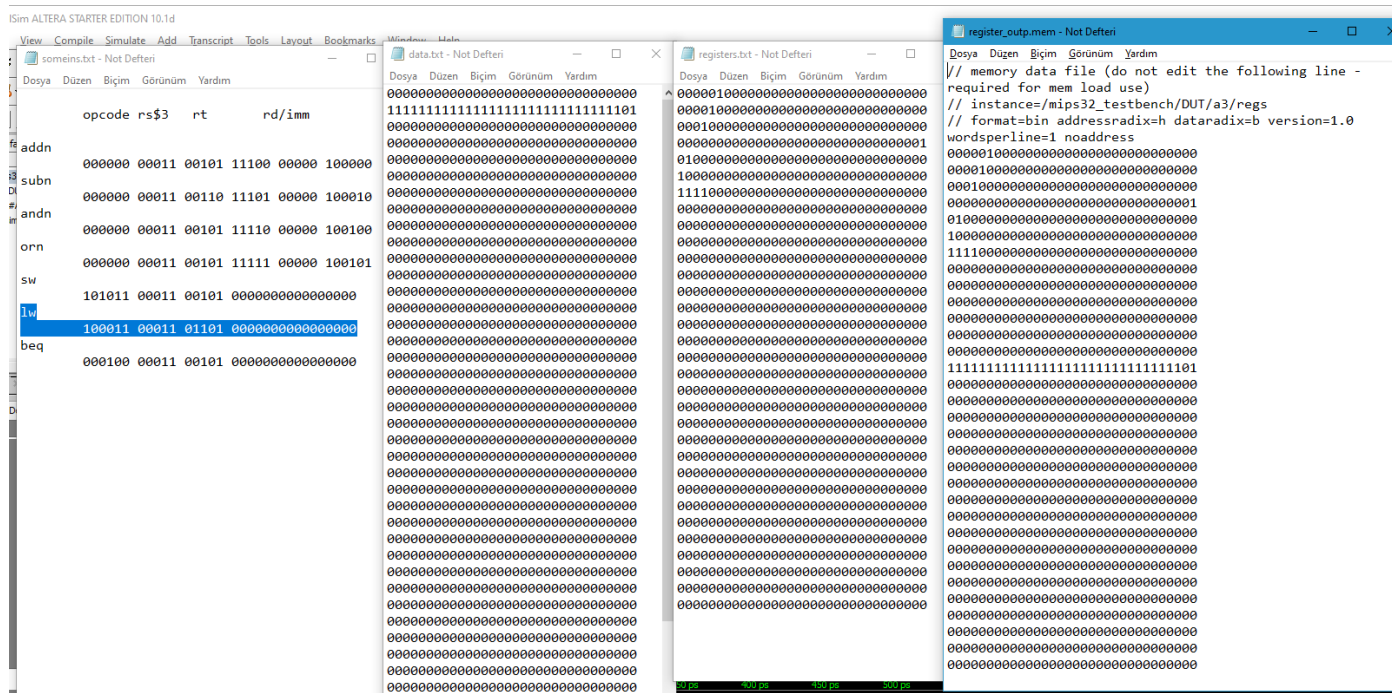
The screenshot shows the Verilog IDE with three windows open:

- Design Unit Hierarchy:** Shows the testbench structure with units like `mips32_test...`, `mips32`, `instruction...`, `control_unit`, and `control...`.
- Registers:** A table showing internal registers:

Name	Value	Kind	Mode
<code>clk</code>	0	Regis...	Internal
<code>i</code>	32	Integer	Internal
<code>instruction_set</code>	xxxxxxxxxxxxxxxx...Fixed...	Internal	
- Assembly Output:** Two windows showing the generated assembly code. The `somains.txt` window shows instructions like `opcode rs$3 rt rd/imm` and `addn 000000 00011 00101 11100 00000 100000`. The `registers.txt` window shows a list of registers with their values, mostly zeros.

The screenshot displays a multi-window IDE environment. The primary window, titled 'somens.txt - Not Defini...', contains MIPS assembly code. The code includes comments about memory data files and a series of instructions: 'addn', 'subn', 'andn', 'orn', 'slw' (highlighted in blue), 'lwl', and 'beq'. Each instruction is followed by its operands in hexadecimal. Other windows show the assembly output in binary format, with each instruction's binary representation aligned with the assembly code. The output for the 'slw' instruction is highlighted in blue, matching the instruction in the assembly code.

lw: \$13 = M[\$3 + SignExtImm(0)] (content of \$3 is 1'b1)



Run 2:

\$28 = 3

[illegible][illegible][illegible]

subn: \$3 = \$3 - \$6 \$29 = 2

The image displays three overlapping windows from the 'Not Defteri' application, which is used for analyzing MIPS assembly code and its execution results.

- Left Window: 'somsins.txt - Not Defteri'**
 - Columns: opcode, rs, rt, rd/imm
 - Assembly instructions visible:
 - addn: 000000 00011 00101 11100 00000 100000
 - subn: 000000 00011 00110 11101 00000 100010
 - andn: 000000 00011 00101 11110 00000 100100
 - orn: 000000 00011 00101 11111 00000 100101
 - sw: 101011 00011 00101 0000000000000000
 - lw: 100011 00011 01101 0000000000000000
 - beq: 000100 00011 00101 0000000000000000
- Middle Window: 'registers.txt - Not Defteri'**
 - Columns: registers, Kind, Mode
 - Content: A list of registers (e.g., \$0, \$1, \$2, etc.) with their corresponding values, kinds, and modes. The values are mostly 00000000000000000000000000000000.
- Right Window: 'register_out.mem - Not Defteri'**
 - Columns: Dosya, Düzen, Biçim, Görünüm, Yardım
 - Content: Memory dump data with comments:
 - // memory data file (do not edit the following line - required for mem load use)
 - // instance=/mips32_testbench/DUT/a3/regs
 - // format=bin addressradix=h
 - data radix=b version=1.0
 - wordspersline=1 noaddress

andn: \$3 = \$3 & \$5 \$30 = 3

The screenshot displays three windows from a MIPS assembly simulator:

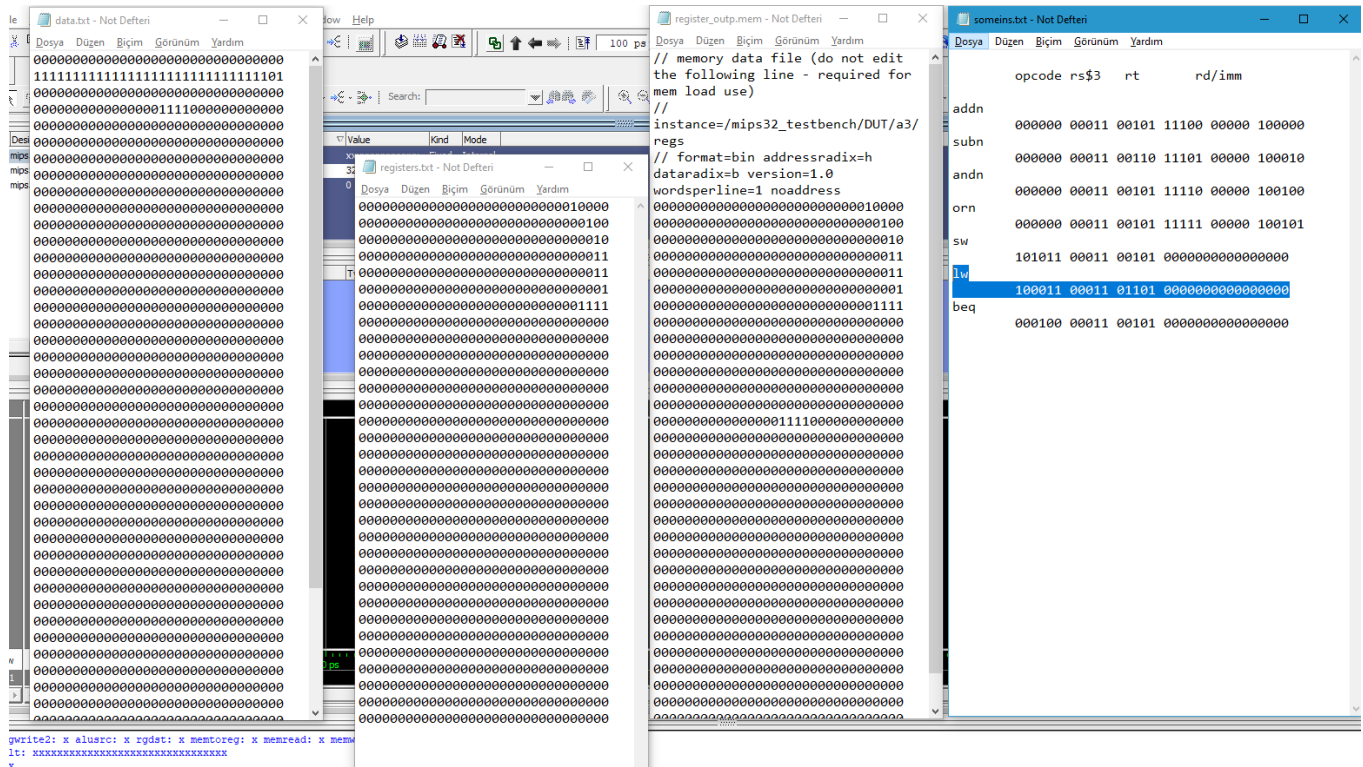
- someins.txt - Not Deferi:** Shows assembly instructions with their binary representations. The instruction `andn 000000 00011 00101 11110 00000 100100` is highlighted in blue. Below the instructions, the text `top mux result: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx` is visible.
- registers.txt - Not Deferi:** A window showing a list of registers, each followed by a long string of 32 zeros, representing the current state of the registers.
- register_outp.mem - Not Deferi:** A window showing memory data. It contains comments like `// memory data file (do not edit the following line - required for mem load use)` and `// instance=/mips32_testbench/DUT/a3/regs`, followed by a long string of 32 zeros.

\$31 = 3

The image shows a Windows desktop with three Notepad++ windows open. The leftmost window, titled "somsins.txt - Not Defteri", displays assembly code with columns for opcode, rs, rt, and rd/imm. The middle window, titled "registers.txt - Not Defteri", shows a list of registers with their current values. The rightmost window, titled "register_out.mem - Not Defteri", contains Verilog code for a memory module. The assembly code includes instructions like "addn", "subn", "andn", "orn", "sw", "lw", and "beq". The registers window shows values for registers like \$zero, \$at, \$v0, etc. The Verilog code defines a memory module with parameters for data file, radix, version, and word size.

sw: $M[\$3 + \text{SignExtImm}(0)] = \5 (content of \$3 is 2'b11)

lw: \$13 = M[\$3 + SignExtImm(0)] (content of \$3 is 2'b11)



How the new R-Type instructions were implemented:

(Sorry for the bad drawing)

