

Lab Assignment #2

Teo Patrosio

EE-126/COMP-46: Computer Engineering w/lab

Professor: Mark Hempstead TA:Parnian Mokri

Tufts University, Fall 2021

- [illegible]

[illegible]

- [illegible]

[illegible][illegible]

- [illegible]

[illegible][illegible]

```
vcom -quiet -2008 add.vhd add_tb.vhd
vsim -quiet -c work.add_tb -do "log -r /*; run 500ns; exit" -wlf add.wlf
wlf2vcd add.wlf -o add.vcd
```

The ALU Control module as described in Figure 4.13 of the textbook (shown below). This module, based on 2-bit ALU Op and a 11-bit Opcode field, outputs one of six operation codes which can be interpreted by the ALU, implemented later in the lab.

| ALUOp              |                    | Opcode field |       |       |       |       |       |       |       |       |       |       | Operation |
|--------------------|--------------------|--------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----------|
| ALUOp <sub>1</sub> | ALUOp <sub>0</sub> | I[31]        | I[30] | I[29] | I[28] | I[27] | I[26] | I[25] | I[24] | I[23] | I[22] | I[21] |           |
| 0                  | 0                  | X            | X     | X     | X     | X     | X     | X     | X     | X     | X     | X     | 0010      |
| X                  | 1                  | X            | X     | X     | X     | X     | X     | X     | X     | X     | X     | X     | 0111      |
| 1                  | X                  | 1            | 0     | 0     | 0     | 1     | 0     | 1     | 1     | 0     | 0     | 0     | 0010      |
| 1                  | X                  | 1            | 1     | 0     | 0     | 1     | 0     | 1     | 1     | 0     | 0     | 0     | 0110      |
| 1                  | X                  | 1            | 0     | 0     | 0     | 1     | 0     | 1     | 0     | 0     | 0     | 0     | 0000      |
| 1                  | X                  | 1            | 0     | 1     | 0     | 1     | 0     | 1     | 0     | 0     | 0     | 0     | 0001      |

Figure 4.13 from the Computer Organization and Design ARM Edition

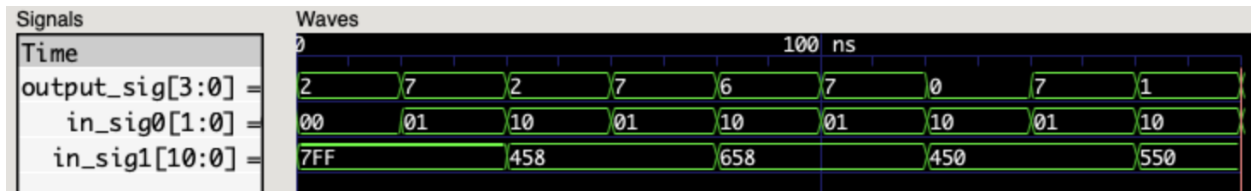


Figure 2: The waveforms from running `alucontrol_tb.vhdl`. The events of this waveform are described below.

The first input, the ALU Op code can be seen as the second waveform in Figure 2, the Opcode field as the bottom wave form, and the output operation code as the top signal.

In Figure 2, there are 10 cases tested (the 10th one isn't visible at the end) that output the six distinct operation codes. Each test case is 20 ns on the waveform.

- 0 - 20ns: Output of 0010, the first row in Figure 4.13.
- 20 - 40 ns: Output of 0111, the second row in Figure 4.13.
- 40 - 60 ns: Output again of 0010, the third row in Figure 4.13. This input was tested twice due to its two entries in Figure 4.13.
- 60 - 80 ns: Output again of 0111, the second row in Figure 4.13. This input was tested twice to verify that it was independent of the input opcode field, as depicted by the X's in Figure 4.13.
- 80 - 100 ns: Output of 0110, the fourth row in Figure 4.13.
- 100 - 120 ns: Output of 0111. This output was tested multiple times as a buffer between different 1-X inputs of ALU Opcodes, and verify more that its output is fully only dependent on the second digit of the ALU Opcode.
- 120 - 140 ns: Output of 0000, the fifth row in Figure 4.13.
- 140 - 160 ns: Output of 0111. This output was tested again multiple times as a buffer between different 1-X inputs of ALU Opcodes.
- 160 - 180 ns: Output of 0001, the last row of Figure 4.13.

The commands used to compile, simulate, and generate the VCD files for the Add module are below.

```
vcom -quiet -2008 alucontrol.vhd alucontrol_tb.vhd
vsim -quiet -c work.alucontrol_tb -do "log -r /*; run 500ns; exit" -wlf
alucontrol.wlf
wlf2vcd alucontrol.wlf -o alucontrol.vcd
```

## CPU Control

The APU Control module as described in Figure 4.22 of the textbook (shown below). This module has an 11-bit input with 9 output flags.

| Input or output | Signal name | R-format | LDUR | STUR | CBZ |
|-----------------|-------------|----------|------|------|-----|
| Inputs          | I[31]       | 1        | 1    | 1    | 1   |
|                 | I[30]       | X        | 1    | 1    | 0   |
|                 | I[29]       | X        | 1    | 1    | 1   |
|                 | I[28]       | 0        | 1    | 1    | 1   |
|                 | I[27]       | 1        | 1    | 1    | 0   |
|                 | I[26]       | 0        | 0    | 0    | 1   |
|                 | I[25]       | 1        | 0    | 0    | 0   |
|                 | I[24]       | X        | 0    | 0    | 0   |
|                 | I[23]       | 0        | 0    | 0    | X   |
|                 | I[22]       | 0        | 1    | 0    | X   |
|                 | I[21]       | 0        | 0    | 0    | X   |
| Outputs         | Reg2Loc     | 0        | X    | 1    | 1   |
|                 | ALUSrc      | 0        | 1    | 1    | 0   |
|                 | MemtoReg    | 0        | 1    | X    | X   |
|                 | RegWrite    | 1        | 1    | 0    | 0   |
|                 | MemRead     | 0        | 1    | 0    | 0   |
|                 | MemWrite    | 0        | 0    | 1    | 0   |
|                 | Branch      | 0        | 0    | 0    | 1   |
|                 | ALUOp1      | 1        | 0    | 0    | 0   |
|                 | ALUOp0      | 0        | 0    | 0    | 1   |

Figure 4.22 from the Computer Organization and Design ARM Edition used to implement CPUControl

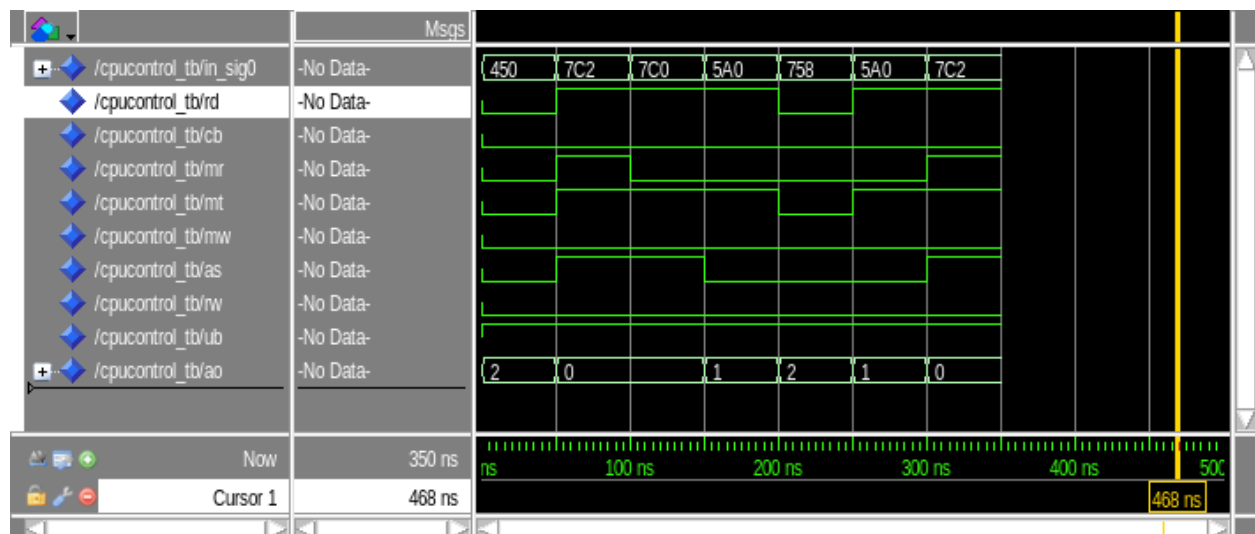


Figure 3: The waveforms from running *cpucontrol\_tb.vhdl*. The events of this waveform are described below.

In *cpucontrol\_tb.vhdl*, there are 4 distinct input test opcodes which match the formats in Figure 4.22, with 6 tests completed in 50 ns increments.

- 0 - 50 ns: The given input is 10001010000, which would fall under the first output “R-format”, where the floating inputs are pulled to 0.

- 50 - 100 ns: The given input is 11111000010, which falls under the last input “LDUR”.
- 100 - 150 ns: The given input is 11111000000, which is the third column “STUR”.
- 150 - 200 ns: The given input is 10110100000, which is a test for the fourth column “CBZ” where the floating inputs are pulled to 1.
- 200 - 2150 ns: The given input is 11101011000, which is again the second column in Figure 4.22 “R-format”, where the floating inputs are pulled to 1.
- 200 - 250 ns: The given input is 10110100000, which tests an alternative signal name for “CBZ” where the floating inputs are pulled to 1.

The commands used to compile, simulate, and generate the VCD files for the Add module are below.

```
vcom -quiet -2008 cpucontrol.vhd cpucontrol_tb.vhd
vsim -quiet -c work.cpucontrol_tb -do "log -r /*; run 500ns; exit" -wlf
cpucontrol.wlf
wlf2vcd cpucontrol.wlf -o cpucontrol.vcd
```

## ALU

The ALU module implements six different operations: and, or, add, subtract, pass b, and nor as described in Section 4.4 of [Computer Organization and Design ARM Edition](#). This module can then apply any of the six operations based on a four bit op code to two 64 bit operands.

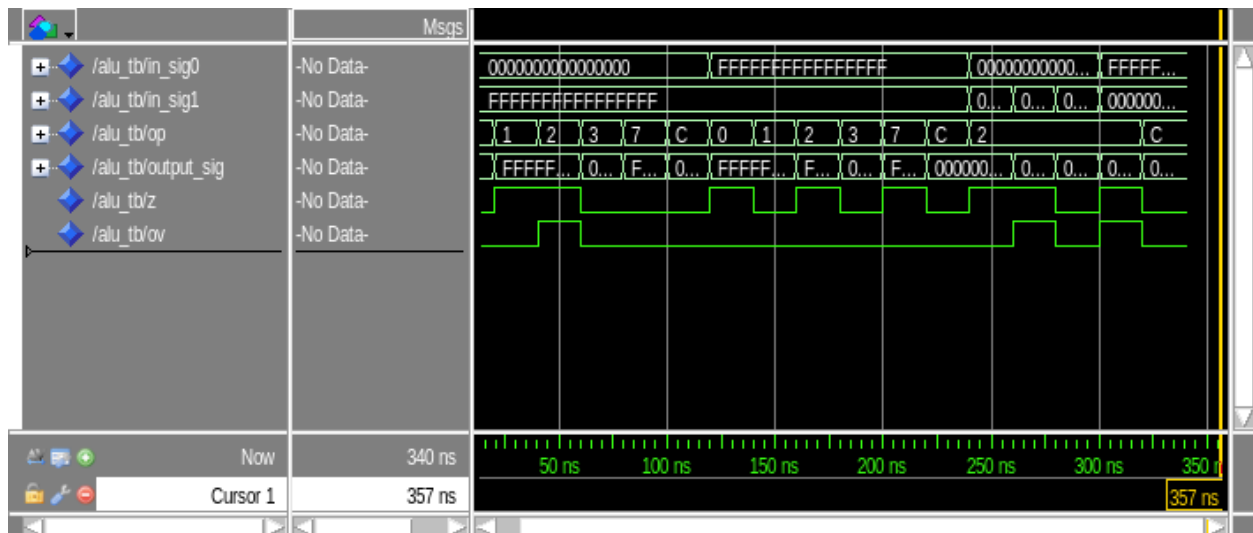


Figure 4: The waveforms from running `alu_tb.vhdl`. The events of this waveform are described below.

For the ALU, there 16 total test cases each run at 20 ns intervals:

The first six cases run through the six operations of the two inputs 0x00000000 and 0xFFFFFFFF:

- 0 - 20 ns:  $0x00000000 \text{ AND } 0xFFFFFFFF = 0x00000000$
- 20 - 40 ns:  $0x00000000 \text{ OR } 0xFFFFFFFF = 0xFFFFFFFF$
- 40 - 60 ns:  $0x00000000 + 0xFFFFFFFF = 0xFFFFFFFF$
- 60 - 80 ns:  $0x00000000 - 0xFFFFFFFF = -0xFFFFFFFF$ ;  $V = 1$
- 80 - 100 ns:  $0x00000000$ , pass B,  $0xFFFFFFFF = 0xFFFFFFFF$
- 100 - 120 ns:  $0x00000000 \text{ NOR } 0xFFFFFFFF = 0x00000000$

The second six cases run through the six operations of the two inputs  $0xFFFFFFFF$  and  $0xFFFFFFFF$ :

- 120 - 140 ns:  $0xFFFFFFFF \text{ AND } 0xFFFFFFFF = 0xFFFFFFFF$
- 140 - 160 ns:  $0xFFFFFFFF \text{ OR } 0xFFFFFFFF = 0xFFFFFFFF$
- 160 - 180 ns:  $0xFFFFFFFF + 0xFFFFFFFF = 0xFFFFFFF$ ;  $V = 1$
- 180 - 200 ns:  $0xFFFFFFFF - 0xFFFFFFFF = 0x00000000$ ;  $Z = 1$
- 200 - 220 ns:  $0xFFFFFFFF$ , pass B,  $0xFFFFFFFF = 0xFFFFFFFF$
- 220 - 240 ns:  $0xFFFFFFFF \text{ NOR } 0xFFFFFFFF = 0x00000000$

The last four cases explicitly test the overflow and zero flags

- 240 - 260 ns:  $0x00000000 + 0x00000000$ ;  $Z = 1$
- 260 - 280 ns:  $0x00000000 + 0x00000001$ ;  $Z = 0$
- 280 - 300 ns:  $0x00000000 + 0x00000000$ ;  $V = 0$
- 300 - 320 ns:  $0xFFFFFFFF + 0x00000002$ ;  $V = 1$

The commands used to compile, simulate, and generate the VCD files for the ALU module are below.

```
vcom -quiet -2008 alucontrol.vhd alucontrol_tb.vhd
vsim -quiet -c work.alucontrol_tb -do "log -r /*; run 500ns; exit" -wlf
alucontrol.wlf
wlf2vcd alucontrol.wlf -o alucontrol.vcd
```

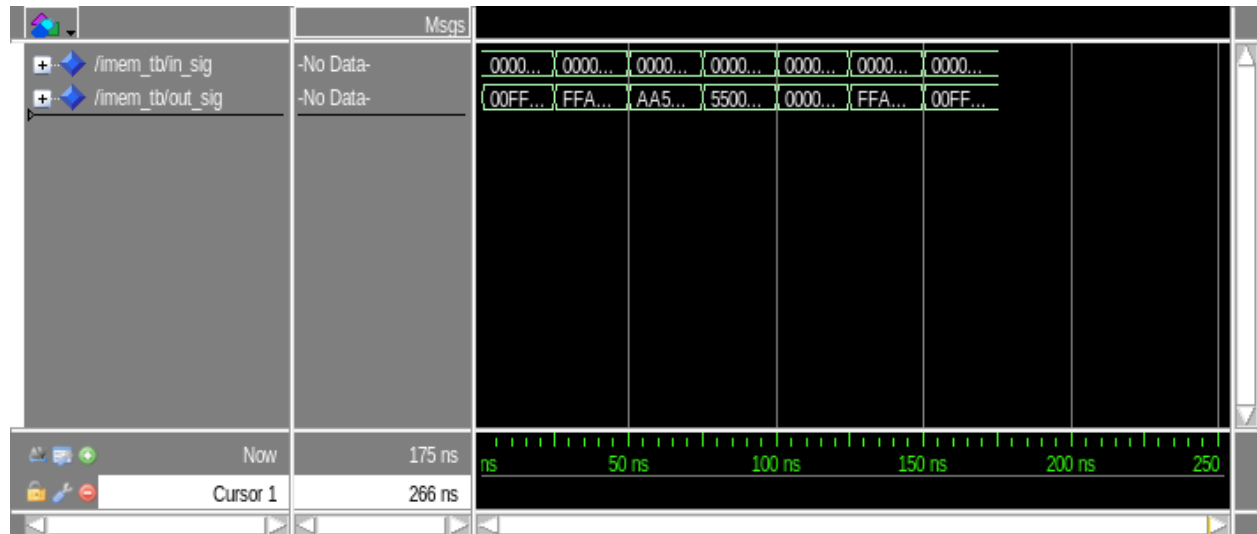
## Registers

\_\_\_\_\_ In my implementation of registers, based on the description in Section 4.4 of Computer Organization and Design ARM Edition, I used a length-32 array where each element was a 64-bit register. In computer architecture, registers are used to temporary hold data and temporary values.





\_\_\_\_\_ In systems with instruction memory and data memory separated, Harvard architectures-- a separate memory module is required. In my implementation of the module, I use an array of 8 bits. Initially I set the default values to 0 => "00000000", 1 => "11111111", 2 => "10101010", 3 => "01010101",



*Figure 6: The waveforms from running imem\_tb.vhdl. The events of this waveform are described below.*

Seven tests in 25 ns increments were done:

- 0 - 25 ns:  
Read Address:  
"00",  
with an expected output of "00000000"
- 25 - 50 ns:  
Read Address:  
"0001",  
with an expected output of "11111111"
- 50 - 75 ns:  
Read Address:  
"0010",  
with an expected output of "10101010"
- 75 - 100 ns:  
Read Address:  
"0011",  
with an expected output of "01010101"

- The commands used to compile, simulate, and generate the VCD files for the IMEM module are below.

DMEM

The screenshot displays the Logic Analyzer interface. The left pane lists signals: `/dmem_tb/wd`, `/dmem_tb/ad`, `/dmem_tb/mr`, `/dmem_tb/mw`, `/dmem_tb/clck`, `/dmem_tb/rd`, and `/dmem_tb/debug`. The middle pane shows 'Msgs' with '-No Data-' for all signals. The right pane shows a timing diagram with green waveforms. Two data values are highlighted in red boxes: `98003311XXXXXXXXXX` and `XXXXXXXXXX...`. The bottom pane shows a time scale from 0 to 250 ns, with a cursor at 278 ns.

Figure 7: The waveforms from running `imem_tb.vhdl`. The events of this waveform are described below.

Note the waveform is not being directly annotated because of how it is generated so a clear description is below. For this simulation, the clock cycle is a total of 20 ns.

- For the first clock cycle (0 to 20 ns), an address is given and the mr is set to 1.
- The second clock cycle (20 to 40 ns), mr is disabled.
- The third cycle (40 to 60 ns), the mr is re-enabled.
- The fourth cycle (60 to 80 ns), the memory write is enabled and it is clear that the data in wd sets rd. At this step, I wondered why there were no default values for the data memory before setting.

The commands used to compile, simulate, and generate the VCD files for the DMEM module are below.

```
vcom -quiet -2008 dmem.vhd dmem_tb.vhd
vsim -quiet -c work.dmem_tb -do "log -r /*; run 500ns; exit" -wlf dmem.wlf
wlf2vcd dmem.wlf -o dmem.vcd
```