

COEN316 – Lab 3

Name: Christopher Dubuc-Pesetti

Student ID: 40037815

Date submitted: October 28, 2018

“I hereby certify that this submission is my original work and meets the
Faculty’s Expectations of Originality.”

Name: Christopher Dubuc-Pesetti

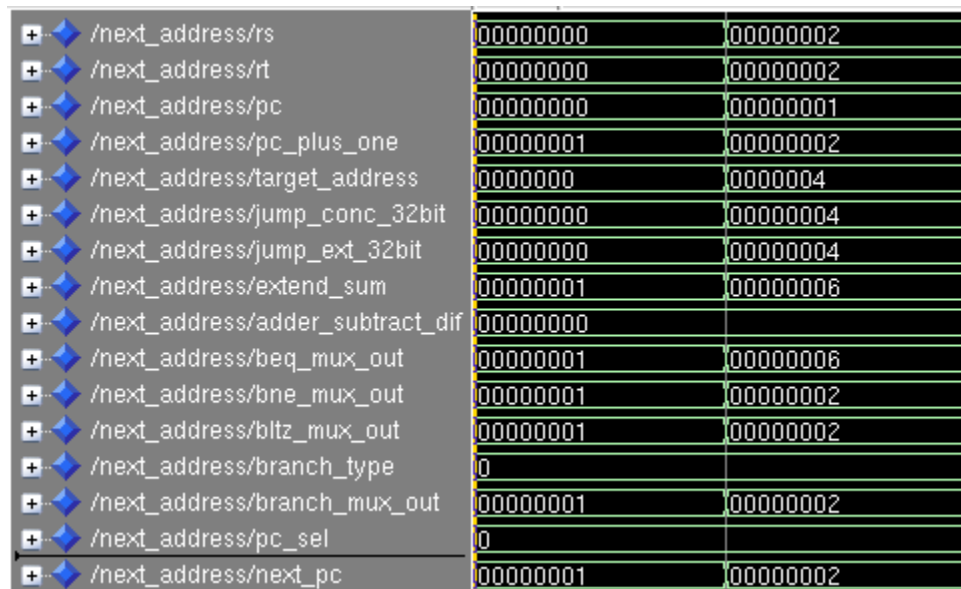
ID: 40037815

Objectives:

The objective of this lab was to design and simulate a Next-Address Unit to be used in a CPU. The NAU is the component responsible for generating which address will be used on the following clock cycle based on the opcode of the address being passed.

Results & Discussion:

The hand-drawn RTL schematic is attached at the front of the appendix which shows the general logic to how the NAU was designed. Two large multiplexers were used to route the appropriate address to the next_pc output. These muxes used the 2-bit branch_type and pc_sel inputs. The “jump” instruction was performed using a concatenation operation to the target address to get the new 32-bit value. The “jump register” instruction was sign extended by checking the 16th bit of the target_address input for ‘1’ or ‘0’; the upper 16 bits were then filled with whichever bit was read. For the branching instructions, 3 small muxes were used to determine whether to pass the PC+1 value or the branch offset value to the larger branch_type mux. The control signals used were based on what the branch condition operation required.



/next_address/rs	00000000	00000002
/next_address/rt	00000000	00000002
/next_address/pc	00000000	00000001
/next_address/pc_plus_one	00000001	00000002
/next_address/target_address	00000000	00000004
/next_address/jump_conc_32bit	00000000	00000004
/next_address/jump_ext_32bit	00000000	00000004
/next_address/extend_sum	00000001	00000006
/next_address/adder_subtract_dif	00000000	
/next_address/beq_mux_out	00000001	00000006
/next_address/bne_mux_out	00000001	00000002
/next_address/bltz_mux_out	00000001	00000002
/next_address/branch_type	0	
/next_address/branch_mux_out	00000001	00000002
/next_address/pc_sel	0	
/next_address/next_pc	00000001	00000002

Figure 1. PC increment testing

Some explanation of the signals may be required for clarification. “jump_conc_32bit” is the concatenated target_address extension from 26bit to 32bits using a 6bit string of zeroes (000000).

“jump_ext_32bit” is the sign extended target_address from 16bits to 32bits.

The “extend_sum” is the sign extended 32bit jump address plus PC+1.

“addersubtract_dif” is the subtraction difference of (rs – rt) used for the beq and bne branching.

The rest are self-explanatory based on the naming (mux outputs and the predetermined entity inputs and outputs).

For figure 1, a test of the pc incrementing was performed to see if the pc_plus_one signal was reaching the output through the proper mux paths (branch_type = 00, pc_sel = 00).

+ /next_address/rs	00000002	00000004	FFFFFFFF	
+ /next_address/rt	00000002			FFFFFFFF
+ /next_address/pc	00000002			
+ /next_address/pc_plus_one	00000003			
+ /next_address/target_address	00000004	00000005	00000006	
+ /next_address/jump_conc_32bit	00000004	00000005	00000006	
+ /next_address/jump_ext_32bit	00000004	00000005	00000006	
+ /next_address/extend_sum	00000007	00000008	00000009	
+ /next_address/addersubtract_dif	00000000	00000002	FFFFFFFFD	00000000
+ /next_address/beq_mux_out	00000007	00000003		00000009
+ /next_address/bne_mux_out	00000003	00000008	00000009	00000003
+ /next_address/bltz_mux_out	00000003		00000009	
+ /next_address/branch_type	1	2	3	
+ /next_address/branch_mux_out	00000007	00000008	00000009	
+ /next_address/pc_sel	0			
+ /next_address/next_pc	00000007	00000008	00000009	

Figure 2. Branch testing

For the figure 2 test, pc_sel was kept at “00” and the branch_type signal was flipped through the different possibilities of 01, 10 and 11 to test the beq, bne and bltz operations respectively. The last 2 tests were for the bltz operation, using different values of rs and rt to confirm functionality.

```

--zero detect
adder_subtract_dif <= unsigned(rs) - unsigned(rt);
process(adder_subtract_dif)
begin
    if (adder_subtract_dif = "00000000000000000000000000000000") then
        zero <= "1";
    else
        zero <= "0";
    end if;
end process;

-- less than zero detect
process(rs)
begin
    if (rs(31) = '1') then
        ltz <= "1";
    else
        ltz <= "0";
    end if;
end process;

```

Figure 3. Branch flags used for routing

Figure 3 shows the flags used to determine the branching functionality in the design. A “zero” and “less-than-zero” detection flag.

“target_address” was set to slightly different values (4, 5 then 6) to make sure the right values were passed. The extend_sum shows that the value of PC+1 (3 in this case) was added to the jump address correctly in each case and this was then passed correctly to the next_pc output.

+ /next_address/rs	FFFFFFFF	
+ /next_address/rt	FFFFFFFF	
+ /next_address/pc	00000002	
+ /next_address/pc_plus_one	00000003	
+ /next_address/target_address	00000009	0008009
+ /next_address/jump_conc_32bit	00000009	00008009
+ /next_address/jump_ext_32bit	00000009	FFFF8009
+ /next_address/extend_sum	0000000C	FFFF800C
+ /next_address/adder_subtract_dif	00000000	
+ /next_address/beq_mux_out	0000000C	FFFF800C
+ /next_address/bne_mux_out	00000003	
+ /next_address/bltz_mux_out	0000000C	FFFF800C
+ /next_address/branch_type	3	
+ /next_address/branch_mux_out	0000000C	FFFF800C
+ /next_address/pc_sel	0	
+ /next_address/next pc	0000000C	FFFF800C

Figure 4. Sign extend testing

Figure 4 shows a test case confirming the sign extension of the target_address signal (16bit to 32bit) using both positive and negative values.

+ /next_address/rs	FFFFFFFF			00000002
+ /next_address/rt	FFFFFFFF			
+ /next_address/pc	00000002			
+ /next_address/pc_plus_one	00000003			
+ /next_address/target_address	00000009	0F00009		
+ /next_address/jump_conc_32bit	00000009	00F00009		
+ /next_address/jump_ext_32bit	00000009			
+ /next_address/extend_sum	0000000C			
+ /next_address/adder_subtract_dif	00000000			00000003
+ /next_address/beq_mux_out	0000000C			00000003
+ /next_address/bne_mux_out	00000003			0000000C
+ /next_address/bltz_mux_out	0000000C			00000003
+ /next_address/branch_type	3			
+ /next_address/branch_mux_out	0000000C			00000003
+ /next_address/pc_sel	1		2	
+ /next_address/next_pc	00000009	00F00009	FFFFFFFF	00000002

Figure 5. Jump testing

Figure 5 show the test case for the jump instruction. The first two cases performed the pc_sel = 01 instruction of:

next_pc <= “000000” & target_address

using different magnitude values.

The last two cases show the value of “rs” being sent to the “next_pc” output, with different magnitude values.

Conclusion:

Compiling and simulating went smoothly, there were no errors or warnings in the precision_log file after synthesis.

Appendix