Pipeline Model Documentation

Sudhakar Yalamanchili

January 29, 2019

Table of Contents

1 Loading A Program

We do not yet have a file interface so you will have to encode your program and edit the instruction RAM that is implemented as an array of words. In the model this is defined as

```
TYPE INST_MEM IS ARRAY (0 to 7) of STD_LOGIC_VECTOR (31 DOWNTO 0);
  SIGNAL iram : INST_MEM := (
    X"8c070004",   --  lw $7, 4($0)
    X"8C080008",   --  lw $8, 8($0)
    X"01074820",   --  add $9, $8, $7
    X"ac09000c",   --  sw $9, 12($0)
    X"1000FFFB",   --  beq $0, $0, -5 (branch back 5 words/20 bytes)
    X"00000000",   --  nop
    X"00000000",   --  nop
    X"00000000"    --  nop
  );
```

Edit the array to encode a new sequence of instructions and recompile the model. You can similarly initialize the register and dram memory, both of which are also implemented as arrays. Easier to use QTSpim to encode the programs and then edit the iram array.

2 Model Organization

The pipeline model is more verbose than it needs to be for correctness. This is because we have defined local signals in each stage (as shown in Figure 4.46) rather that use the pipeline bits directly. The result is that the model is easier to read and therefore easier to debug. Local signals in each stage are prefaced with the stage name, i.e., id_, ex_, etc.

Each pipeline stage is structured into 3 sets of VHDL statements.

1. Local signals for the specific stage are extracted from the corresponding bits of the pipeline register (to the left) that drives the stage. For example, consider the ID stage. Code will first have statements such as the following

   ```
   id_instruction <= IF_ID(31 downto  0);
   id_PC          <= IF_ID(63 downto 32);
   ```

   These correspond to the signals in the **Figure 1** below.

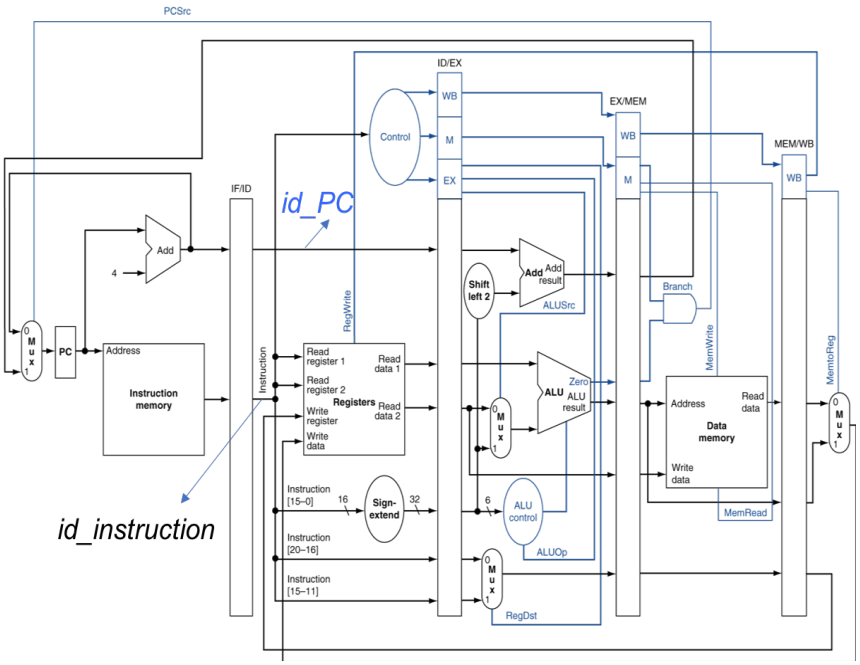

Figure 1 Reading the local signals in a pipeline stage

You will similarly see local signals defined for other signals extracted from the pipeline registers in each stage.

2. You will then see signal assignment statements for the computations for that stage. For example, see **Figure 2**. In ID you will see statements such as the following.

```
id_Sign_extend <=       X"0000" & id_instruction( 15 DOWNTO 0 ) WHEN id_instruction(15) = '0'
                    ELSE  X"FFFF"  & id_instruction( 15 DOWNTO 0 );

id_write_data   <=        wb_read_data WHEN (wb_MemToReg = '1' )  -- MemToReg signal
                    ELSE  wb_ALUOutput;
```

The corresponding signals are shown in **Figure 2**.

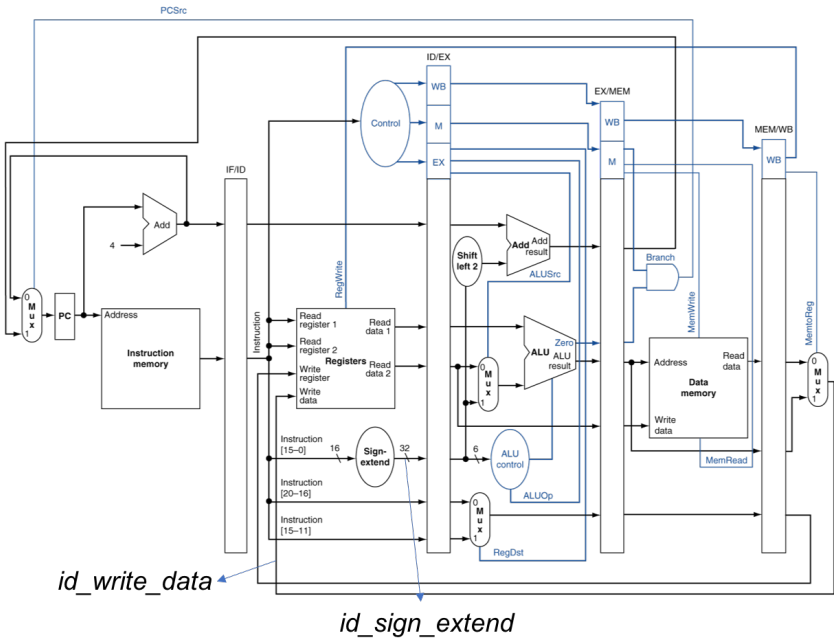

Figure 2 Computing new signal values in a stage

3. Finally, you will see a process that latches the signal values computed in this stage into the pipeline register that drives the next stage. For the EX stage the process is the following.

```
PROCESS
                BEGIN
                        WAIT UNTIL (rising_edge(clk));
                        IF (reset = '1') THEN
EX_MEM <= X"00000000000000000000000000" & "000";-- initialize
                        ELSE
--propagate control bits
EX_MEM (106 downto 102) <= ID_EX(146 downto 142);

-- propagate instruction related values
EX_MEM(101 downto 70) <= ex_branch_address;
EX_MEM(69)                <= ex_Zero;
EX_MEM(68 downto 37)  <= ex_ALU_result; -- addres or ALU result
EX_MEM(36 downto 5)    <= ex_register_rt; -- write data
EX_MEM(4 downto 0)     <= ex_wreg_addr;
                 end if;
END PROCESS;
```

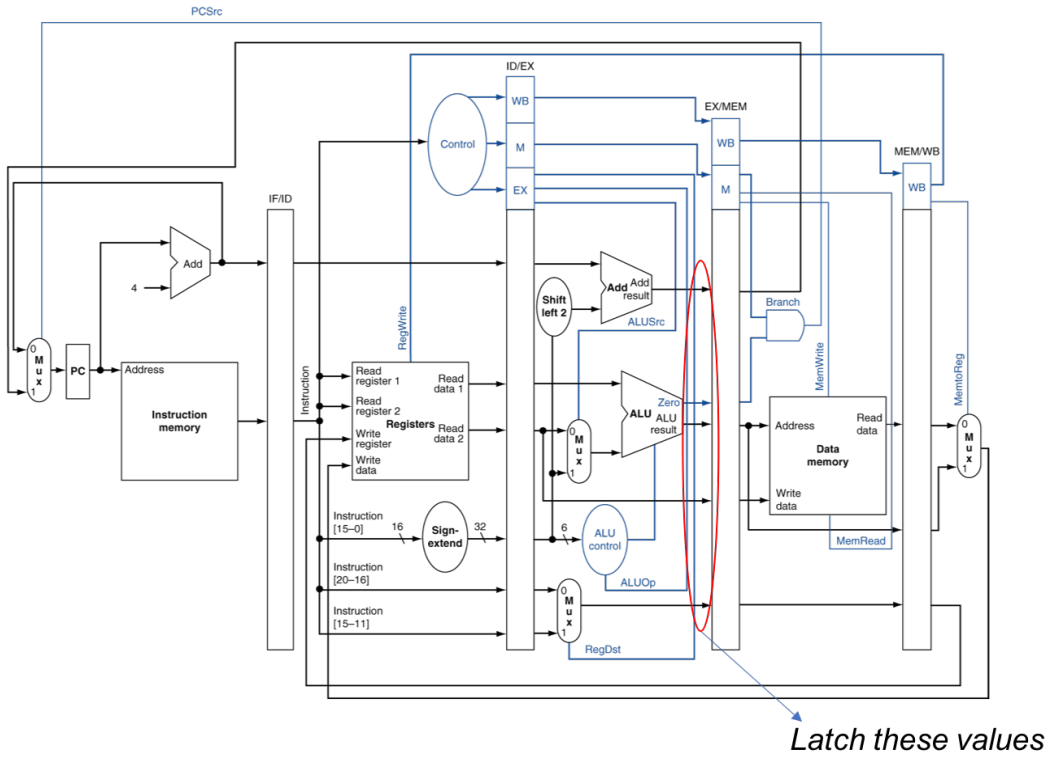

Figure 3 Latching stage signal values into the pipeline

4. Finally, to enhance readability, **Figure 4** documents the meaning of the bits in each pipeline register.

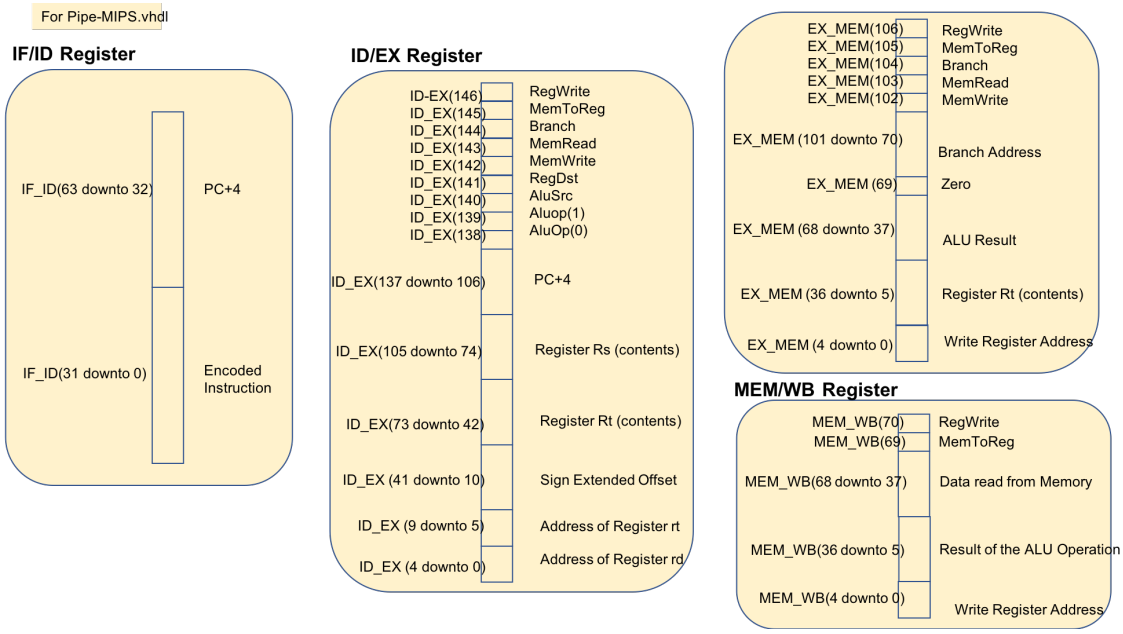

Figure 4 Meaning of pipeline register bits

3 Reading the Traces & Debugging

Compile and execute the base pipelined MIPS model in the same manner as described for the single cycle datapath. The following describes one approach to being able to systematically read the trace to debug your model.

1. First draw a time space diagram of the sequence of instructions being executed by the model. For example, for the program distributed in the model the diagram would appear as shown in **Figure 5**.

Cycle#/Pipe Stage	IF	ID	EX	MEM	WB
0	lw				
1	lw	lw			
2	add	lw	lw		
3	sw	add	lw	lw	
4	beq	sw	add	lw	lw
5	nop	beq	sw	add	lw
6	nop	nop	beq	sw	add
7	nop	nop	nop	beq	sw

Figure 5 Time space diagram of test code segment

2. Now place the yellow bar over a clock cycle. You can now read all of the signal values for that clock cycle. For example, consider clock cycle 0 and the signal trace shown in **Figure 6**.

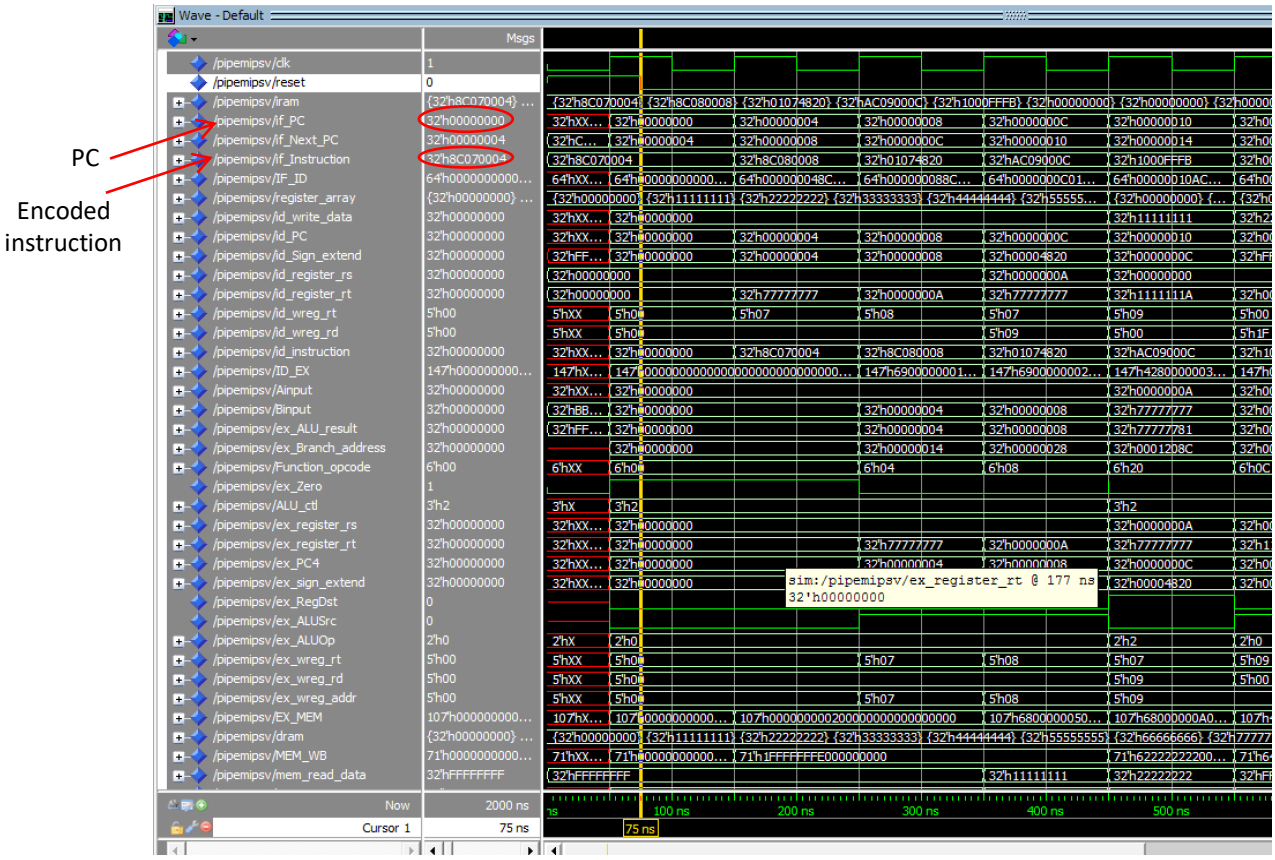

Figure 6 Examining the pipeline trace at cycle 0

According to the time space diagram the lw $7, 4($0) instruction is being fetched from memory. We can look at the IF stage signals as marked on the trace above. The PC is 0x00000000 and the encoded instruction is 0x8C070004 which is the encoding of the lw instruction. Note these are signals in the IF stage and are not yet latched into the IF/ID pipeline register.

3. Now consider a pipeline full of instructions as in clock cycle 4 as shown in the time space diagram in **Figure 5**. Place the bar over clock cycle 4 as shown in the trace in **Figure 7**. Now at the left end of the window you can read the value of every signal in each stage of the pipeline. Knowing the instruction in each stage of the pipeline, you what the values should be and can compare them to the actual values from the trace. For example, the next PC in the IF stage (1) is 0x14. This is the address of the next instruction after the beq instruction which is stored at 0x10. The signal id_write_data (2) is the value that is being written to the register file. This is the output of the MemToReg mux. The lw instruction in the WB stage is the source of the write data. This instruction loads data from location 0x4 in memory which on inspection of the default contents of DRAM can

be seen to be 0x11111111. The output of the ALU in the EX stage is should be the output of the `add` instruction (3). This the sum of the contents of register `$8` (0x0000000A) and register `$7` (0x77777777). This is 0x77777781 as shown. Also, in the EX stage there is the value of register `rt` read from the ID/EX pipeline register. This is `$7` from the add instruction and is shown to be 0x77777777 which is the default value stored in register `$7`. Finally, in the MEM stage the value read from memory is shown as 0x22222222 (5). The instruction in this stage is the second `lw` instruction which reads from location 0x8 in data memory which you can check is initialized to 0x22222222.

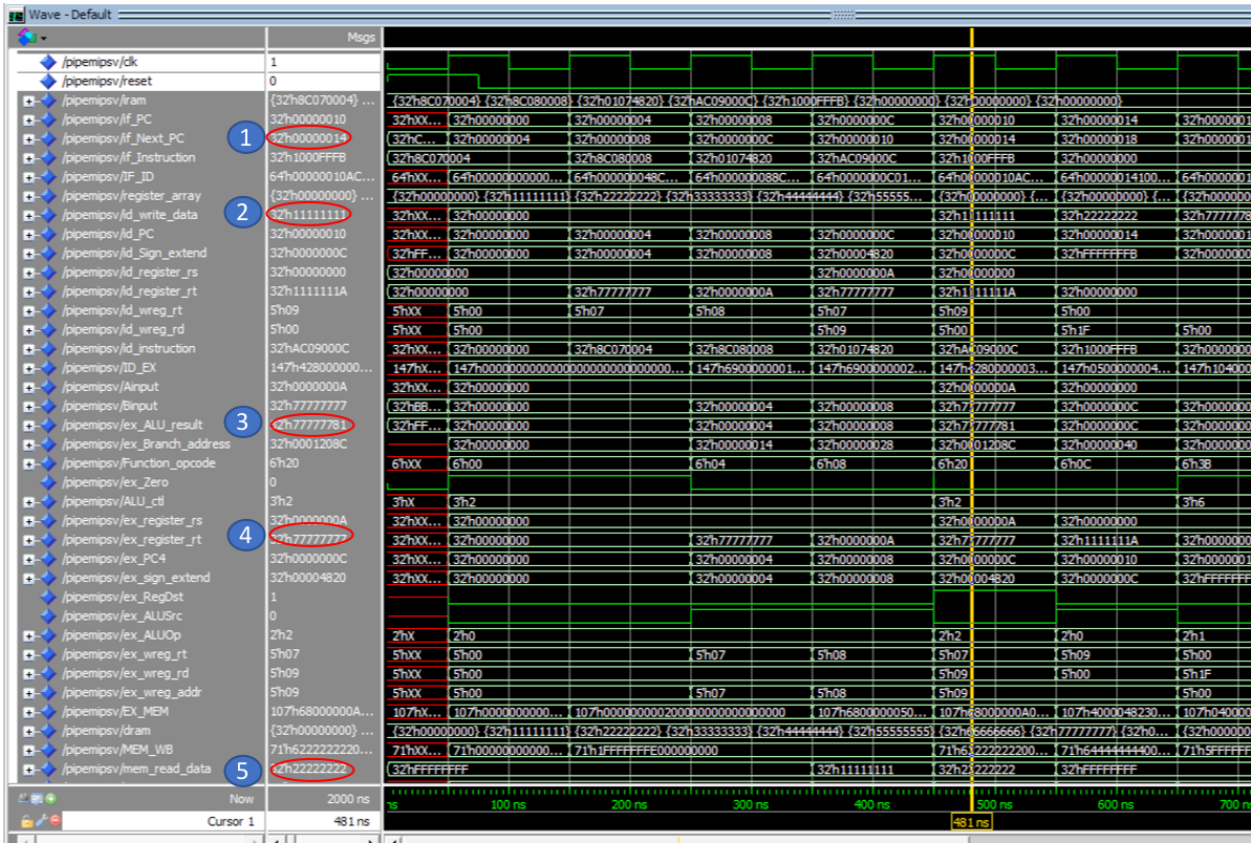

Figure 7 Examining the pipeline trace at cycle 4

From the preceding, hopefully you get the idea of how to read the trace to determine if the program is executing correctly. **Note that the base model does not support forwarding or data or control hazard detection**. Consequently, determining the correct values of the signals in various stages of the pipeline must take this into account, i.e., the program will not compute the values you think it might.

Exercise: What is the final value of memory location 0xC?

Note you can click on the + sign next to a signal to see the values of array elements such as DRAM or the register file.

4 Complementary YouTube Video

The following link is for a video tutorial for the VHDL MIPS pipeline model. The tutorial covers the exact same material as this document in video format.

- YouTube link: https://youtu.be/u0XvIJQLNWM

5 Some General Helpful Notes

Here are a few general comments that may prove helpful in writing correct code or recognizing problems.

1. In these models, you should have a signal being only assigned in one place in the code. If a signal is in the LHS of multiple signal assignment statements the value of the signal may not be what you expect and often will be 0xXXXXXXXX, where X represents the value Unknown. This happens when the simulator cannot determine the correct value of a signal with multiple drivers. Imagine a wire being driven by multiple gates. What should the value on the wire be? **Remember you are describing hardware!** This is not a programming language to describe algorithmic computation.
2. To add hardware functionality to the datapath the following steps are recommended.
    a. Study and conclude the dataflow that you need between the existing pipeline and the new hardware logic blocks.
    b. Modify the pipeline figure to place the new hardware bocks and walk through the execution.
    c. Connect the new hardware blocks with new signals as necessary. Now you should have a figure of complete hardware implementation.
    d. Declare any new signals that you have added.
    e. Add the new signal assignment statements/processes as necessary.
    f. Modify control.

    The most useful step is probably the first one – drawing the architecture diagram of the new datapath.