**CMPE 140 Lab 7: Enhanced Single-Cycle MIPS Processor**Group 1:

Albert Chen – 007037498  
Sanjay Maharaj – 007284355

April 18, 2014

**Introduction**

This lab was intended to assist us in learning how to extend the support for instructions for a processor. In the lab, we are required to enhance the sing-cycle MIPS processor’s functionality by extending its instruction support from the initial sub set {ADD, SUB, AND, OR, SLT, LW, SW, BEQ, J, ADDI} to cover the following additional instructions MULTU, MFHI, MFLO, JR, and JAL. The design must be test via both functional verification and FPGA validation. Machine code of the test program should be stored in the memory file named *memfile.dat* which should be placed in the project directory (Xilinx directory). This memory file should contain one 32-bit machine code (in hex) per line (unused memory space should be filled with zeroes). If any content of this memory file is changed, the entire design must be re-synthesized and implemented before programming the FPGA on the Nexys2 board.

**Task**

1. Extend the MIPS single-cycle processor design to support MULTU, MFHI, MFLO, JR, and JAL.
   1. Figure 1 and Figure 2 are provided – which are the design of the MIPS single-cycle processor and assembly code respectively.

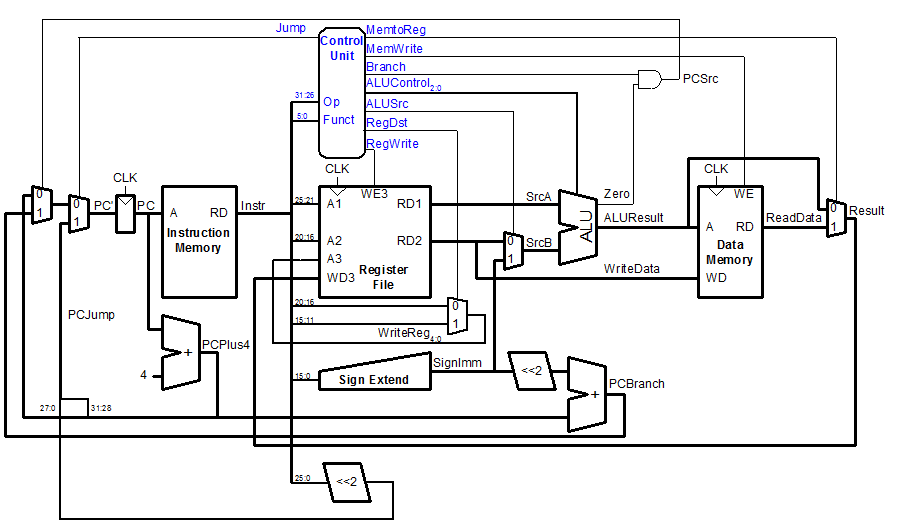


Figure 1. Provided design of MIPS Single-cycle processor.

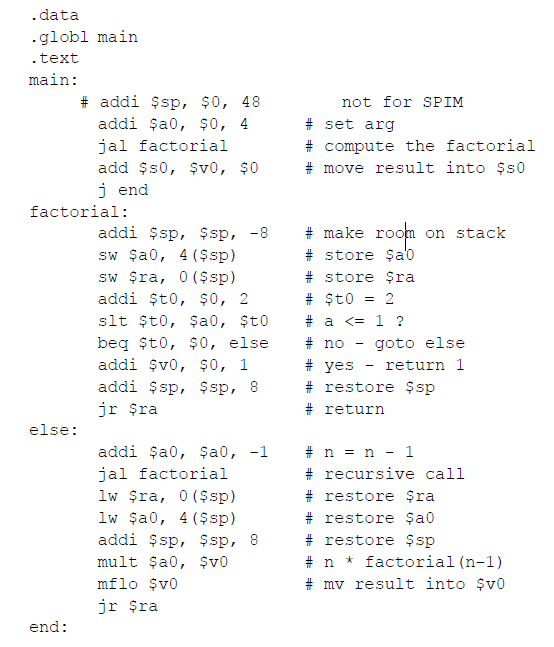


Figure 2. Provided test code for extended design.

It is required that the simulation is completed (on screen) before the implementation (hardware). The following are lists of provided files for simulation and implementation

The following source code was provided in the MIPS\_Single\_Simulation archive

* MIPS\_Single\_Simulation
  + memfile.dat
  + mips.v
  + mipsmem.v
  + mipsparts.v
  + mipstest.v
  + mipstop.v

The following source code was provided in the MIPS\_Single\_Implementation archive

* MIPS\_Source\_Code
  + disp\_hex.v
  + memfile\_s.dat
  + MIPS\_Inital.ucf
  + MIPS\_Inital.v
  + MIPS\_top.v

To come up with the design, we referenced the MIPS Reference Data Greencard to determine what kind of instructions MULTU, MFHI, MFLO, JR, and JAL were. After referring to the greencard, we came up with the following design (Figure 3).

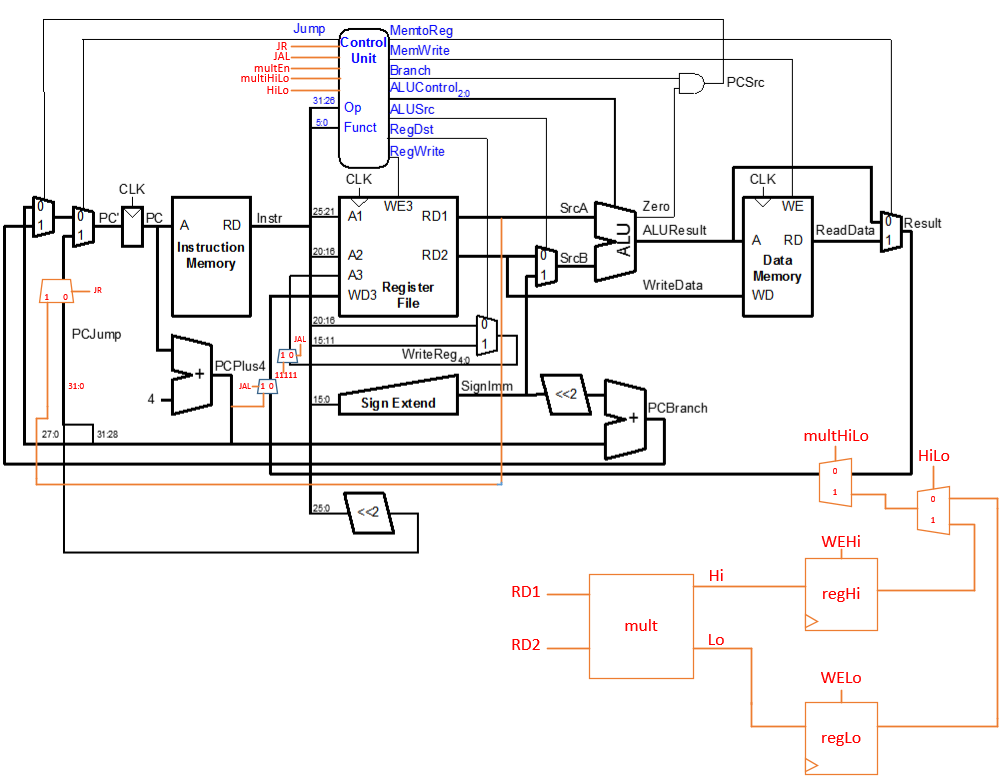


Figure 3. Extended MIPS single-cycle processor

To accommodate the instructions, both the main decoder (maindec module) and the ALU decoder (aludec module) within the control unit had to be extended. Maindec was extended to accommodate JAL as it was a J-type instruction, and aludec was renamed to auxdec and extended to accommodate the rest of the new instructions, which were all R-type instructions. In figure 4, these new extended signals and renamed modules can be seen.

In order to accommodate the MULT, MFHI, and MFLO signals, two new modules were designed: a multiplier which takes in two 32-bit signals and produces a 64-bit signal and a 32-bit register which will take in the results of either the upper ([64:32]) or lower ([31:0]) 32 bits of the output signal of the multiplier.

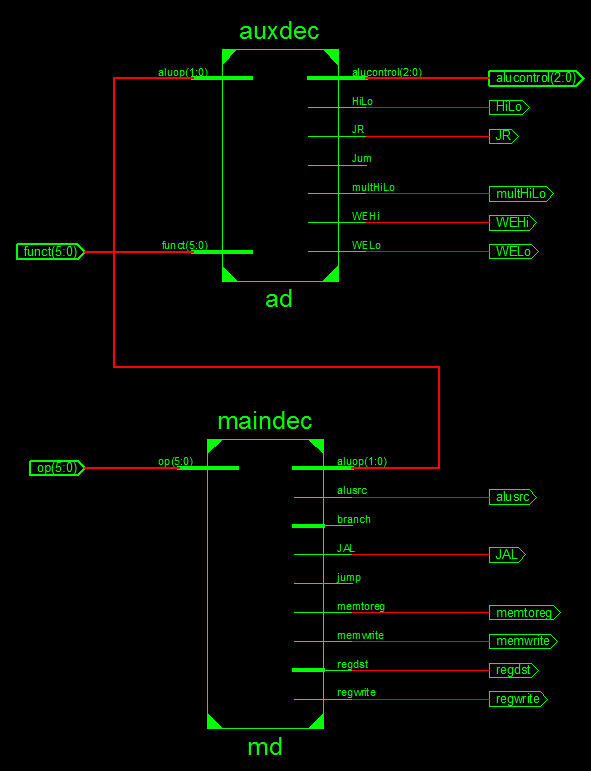


Figure 4. Control unit with extended signals.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op [5:0]** | **RegWrite** | **RegDest** | **AluSrc** | **Branch** | **MemWrite** | **MemtoReg** | **Jump** | **ALUOp[1:0]** | **JAL** |
| R-type | 000000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| lw | 100011 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 00 | 0 |
| sw | 101011 | 0 | x | 1 | 0 | 1 | x | 0 | 00 | 0 |
| beq | 000100 | 0 | x | 0 | 1 | 0 | x | 0 | 01 | 0 |
| j | 000100 | 0 | x | x | x | 0 | x | 1 | xx | 0 |
| **JAL** | **000011** | **0** | **x** | **x** | **x** | **0** | **x** | **1** | **xx** | **1** |

Table 1. Control unit main dec output table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **ALUOp [1:0]** | **Funct** | **ALUControl [2:0]** | **WEHi** | **WELo** | **HiLo** | **multHiLo** | **JAL** | **JR** |
| Add | 00 | X | 010 | x | x | x | x | x | x |
| Sub | X1 | X | 110 | x | x | x | x | x | x |
| Add | 1X | 100000 | 010 | x | x | x | x | x | x |
| Sub | 1X | 100010 | 110 | x | x | x | x | x | x |
| And | 1X | 100100 | 000 | x | x | x | x | x | x |
| Or | 1X | 100101 | 001 | x | x | x | x | x | x |
| SLT | 1X | 101010 | 111 | x | x | x | x | x | x |
| JR | 1X | 001000 | xxx | x | x | x | x | x | 1 |
| MULTU | 1X | 011001 | xxx | x | x | x | x | x | x |
| MFLO | 1X | 010010 | xxx | 0 | 1 | 0 | 1 | x | x |
| MFHI | 1X | 010000 | xxx | 1 | 0 | 1 | 1 | x | x |

Table 2. Control unit aux dec output table

**Design Verification**

The testbench from Lab 5, which was intended to demonstrate the simulation of a single-cycle MIPS processor on Xilinx iSim is sufficient for testing this design as well. This is due to the fact that when the complete design begins to perform its functions, the entire contents of the .dat file are read regardless of testbench directions. The expected result is the output of all signals, whose values can be compared to the output of the MIPS Assembler and Simulator software. Figure 5 and 6 show the waveforms of various signals. Initially, the waveform didn’t display what was expected so debugging was needed and it took a few hours to see the desired results on the waveform.

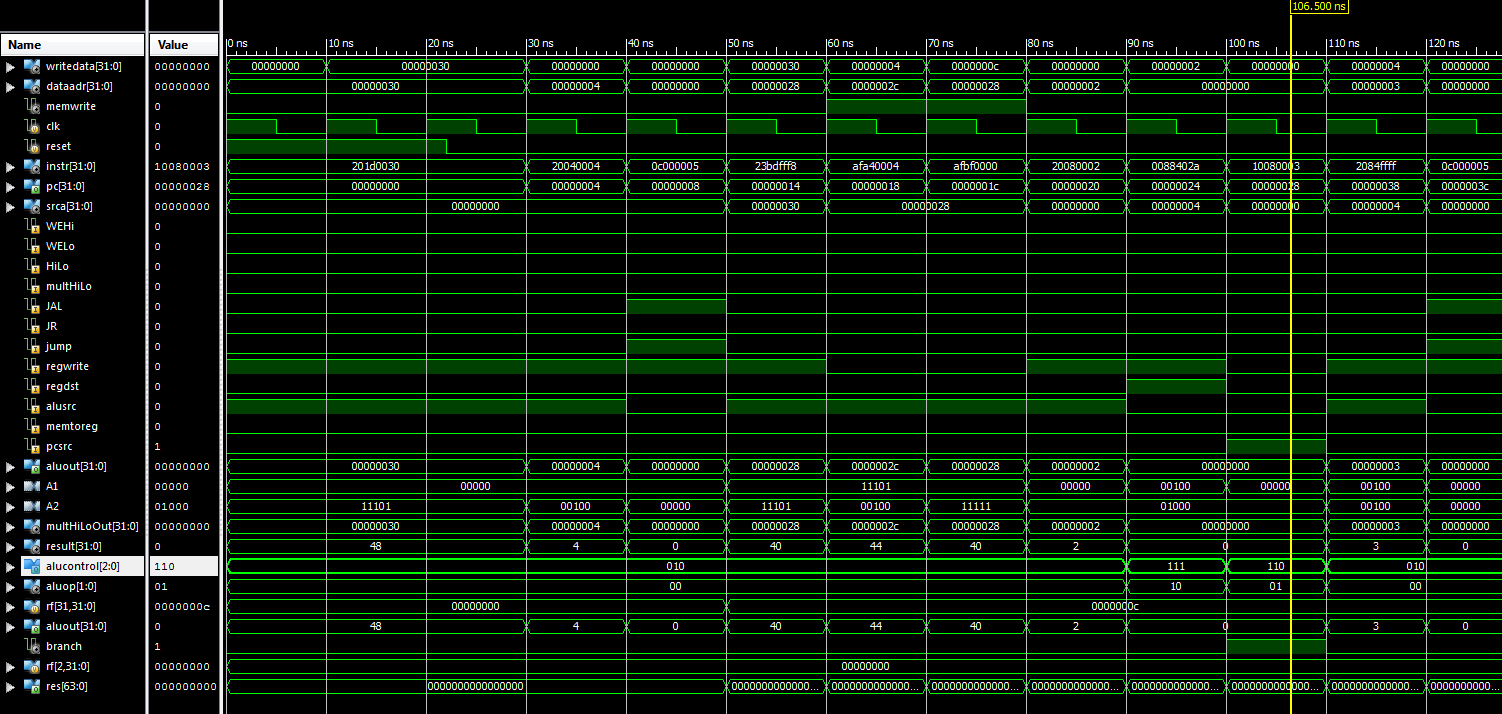


Figure 5. Screenshot of beginning waveform

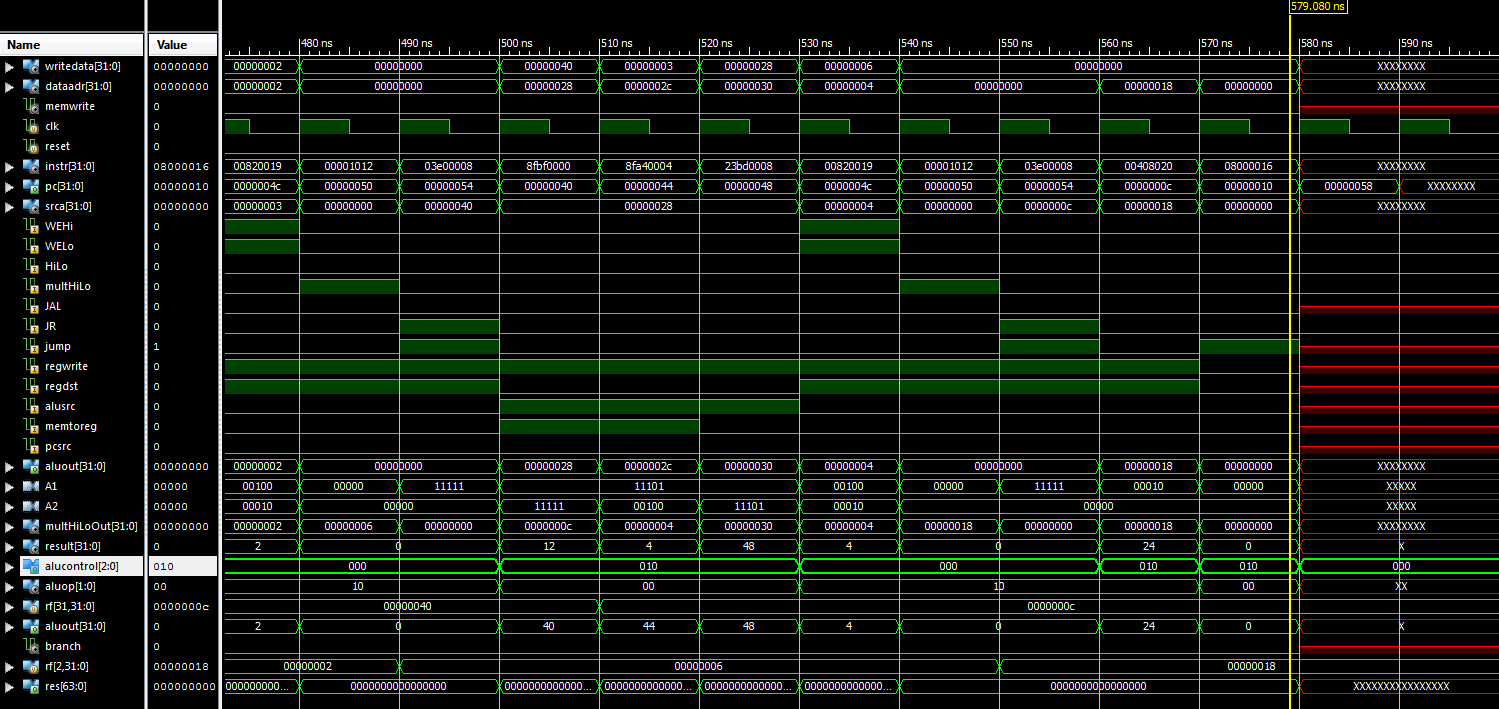


Figure 6. Screenshot of end of waveform

**Hardware Implementation**

To implement the design onto the Nexys2 FPGA Board, we first confirmed the successful operation of our design by simulation on Xilinx iSim. We then used the design provided for the hardware implementation of a single-cycle MIPS processor in lab assignment 6 as a basis for adding the necessary features. The *mips\_initial.v* file was modified so that each module was appended with the same contents appended to the modules in the simulation files, and so the pre-existing signals needed for display output on the seven segment lcd remained intact.

**Issues Encountered during Hardware Implementation**

Trimmed Modules

A synthesis of the design after it was modified produced approximately 20 warnings regarding trimming of signals and modules during the implementation stages. Trimming occurs when the synthesizer has determined that a module or signal will not be used in a design, so it can be removed, or "trimmed" in order to keep the design compact and concise. However, the warning message also suggests that signals were not connected correctly as a reason for trimming. After looking through the modified modules again, some signals were missing in some modules, and were added. The trimming warnings were no longer shown after this corrective action.

Pinout Difference on 1200k Nexys2

If using a Nexys2 with 1.2 million gates, the pinout location for sinkBit in the User Constraints File (mips\_initial.ucf) must be changed R4 to P4 to correctly complete the mapping stage of the design implementation process. Otherwise, the message "Error: pack 1107" is seen, as shown in Figure 7 below, in the console window and a failure occurs.

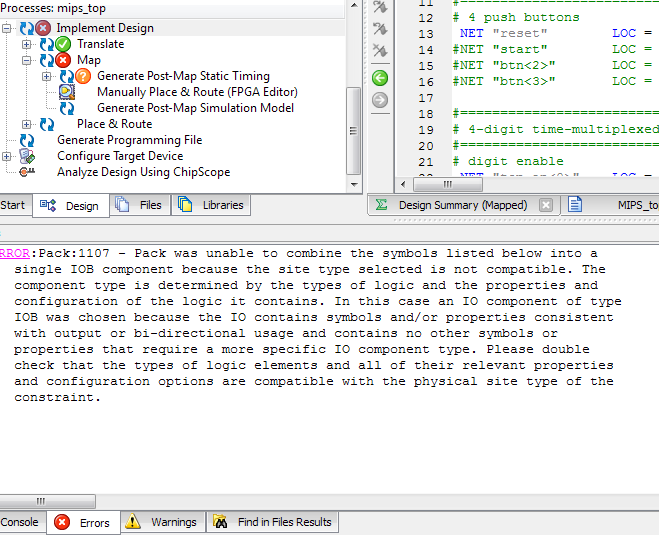


Figure 7. Error produced due to a difference in pin configurations

**Verification of Hardware Implementation**

After generating the bit file and uploading it, the output on the four positions of the seven-segment led display can be used to verify the correct operation of the design. The sliding switches on the board control what is to be displayed as output, as can be seen in table 3. The rightmost pushbutton on the board resets the program counter value to 0, so that the program may restart.

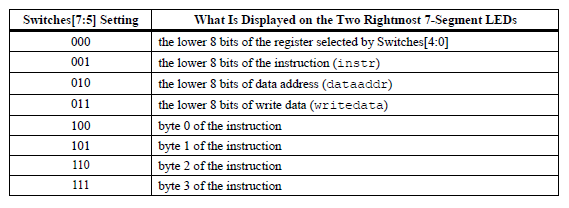


Table 3. Functions corresponding to switch combinations.

As can be seen in Figures 8, 9, and 10. above, the two leftmost places on the seven-segment display represent the value of the program counter (PC). When switch 1 is active (in the up position), the two rightmost places on the seven segment display represent the output of the second register within the register file. This contains the value of the computed factorial in hexadecimal format. A successful implementation of this code for 4! would be expected to display 1, 1\*2, 1\*2\*3, and finally 1\*2\*3\*4, as the algorithm travels recursively. In hexadecimal, this progression is displayed as 1, 2, 6, 18, and is successfully completed by our design.



Figure 8. PC value followed by hexadecimal notation of second register of the register file.



Figure 9. PC value followed by hexadecimal notation of second register of the register file.

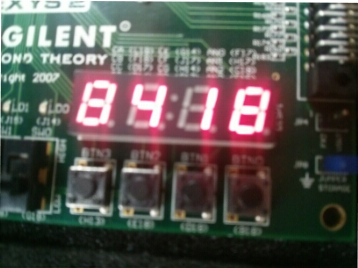
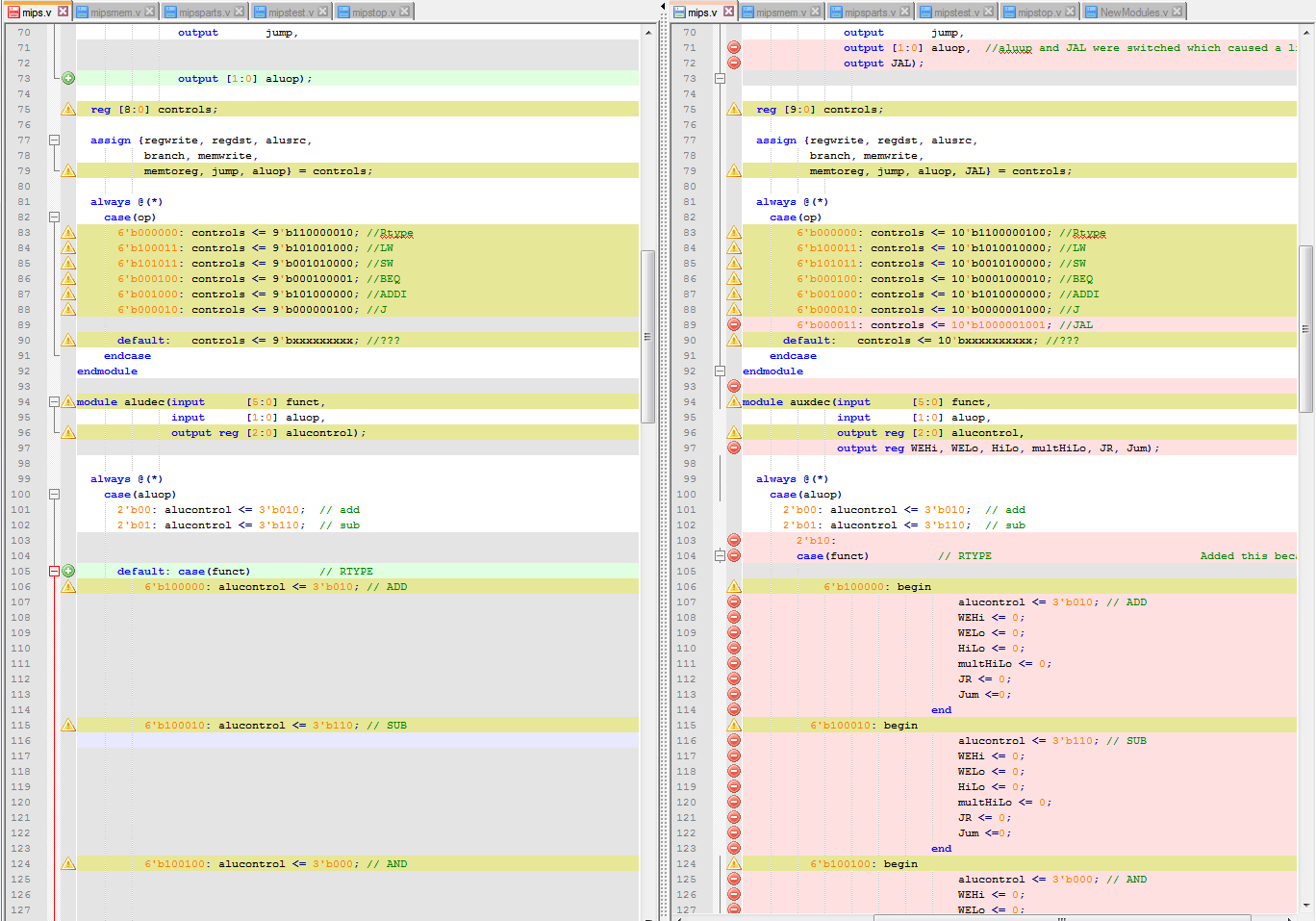
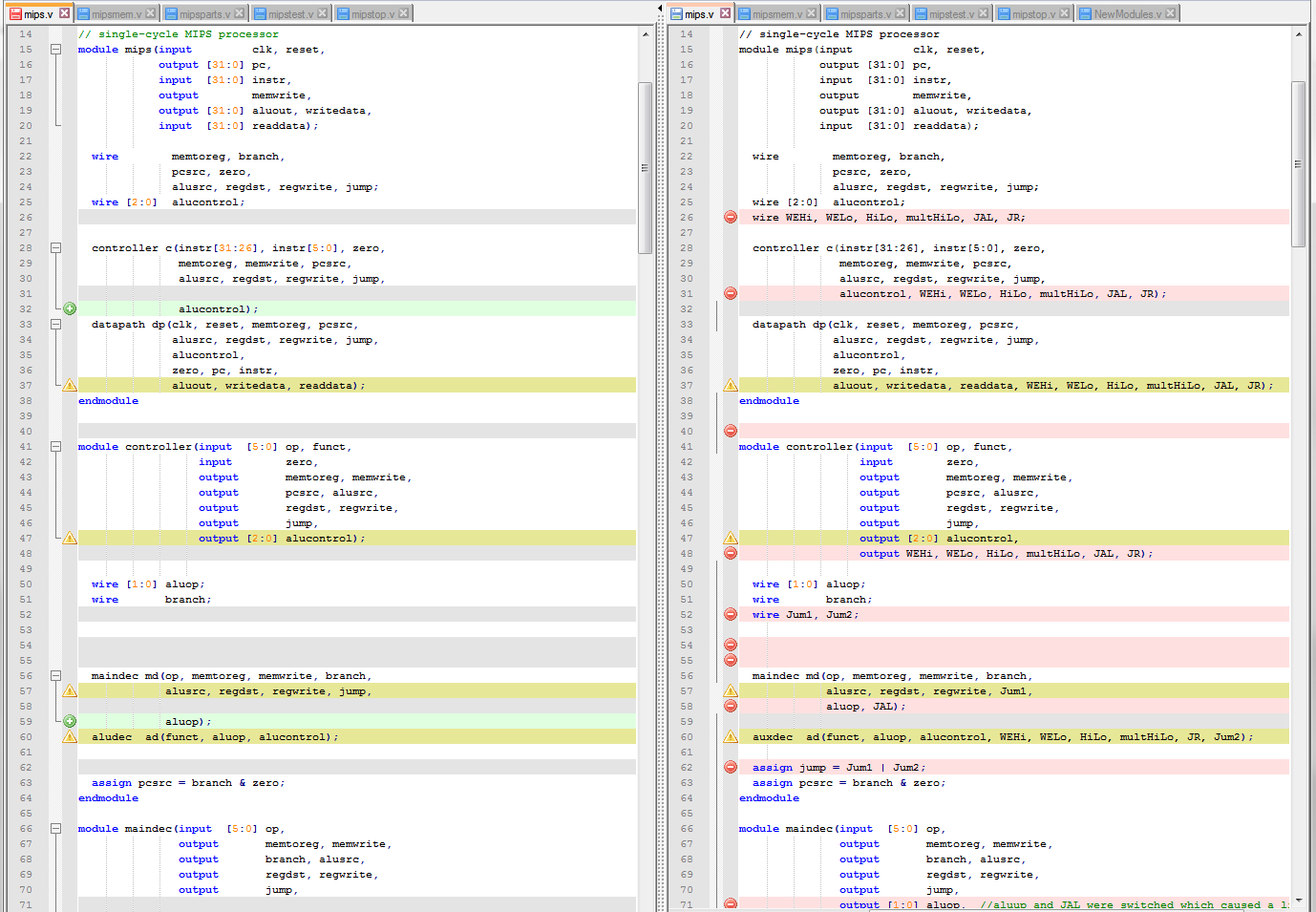


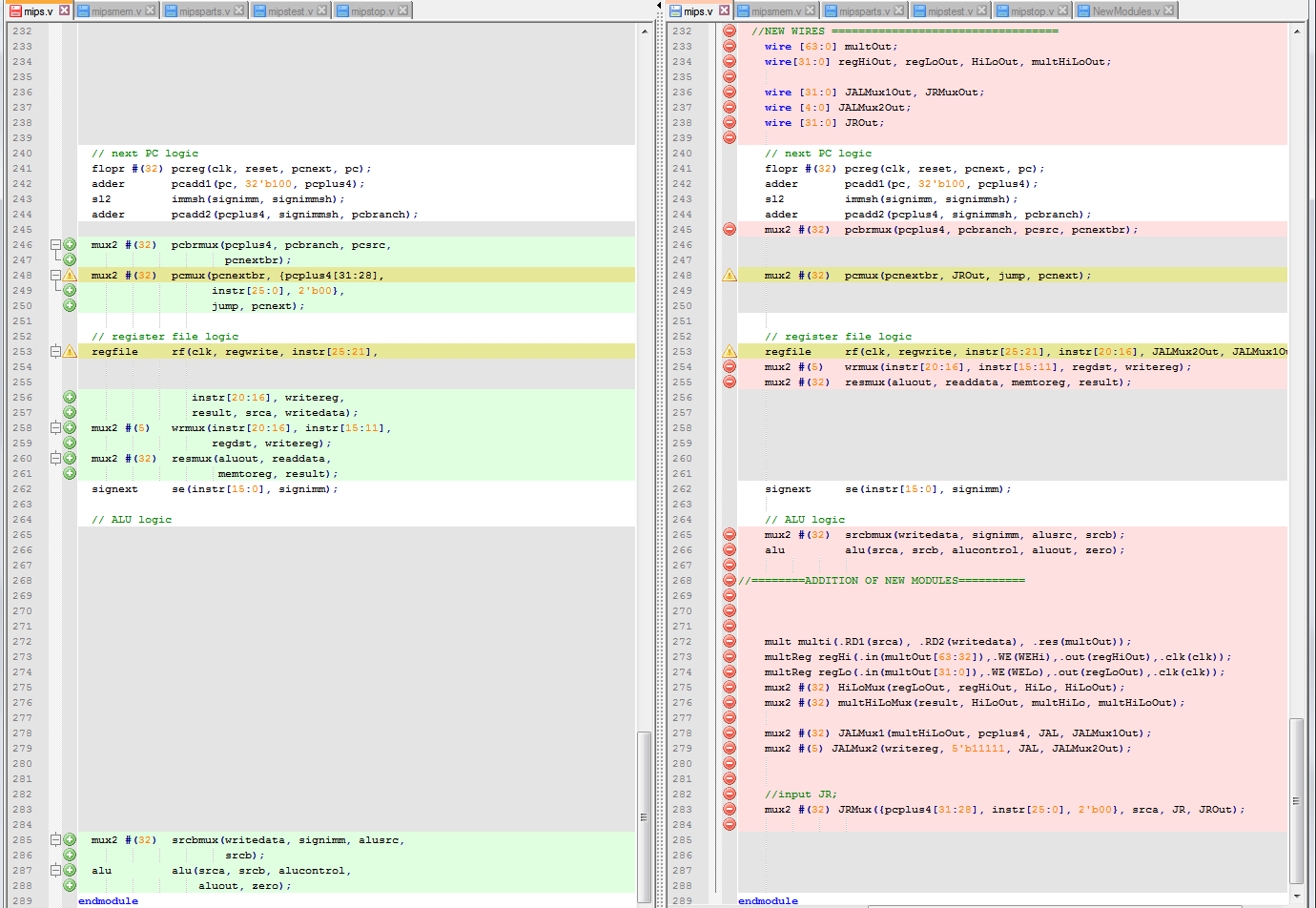
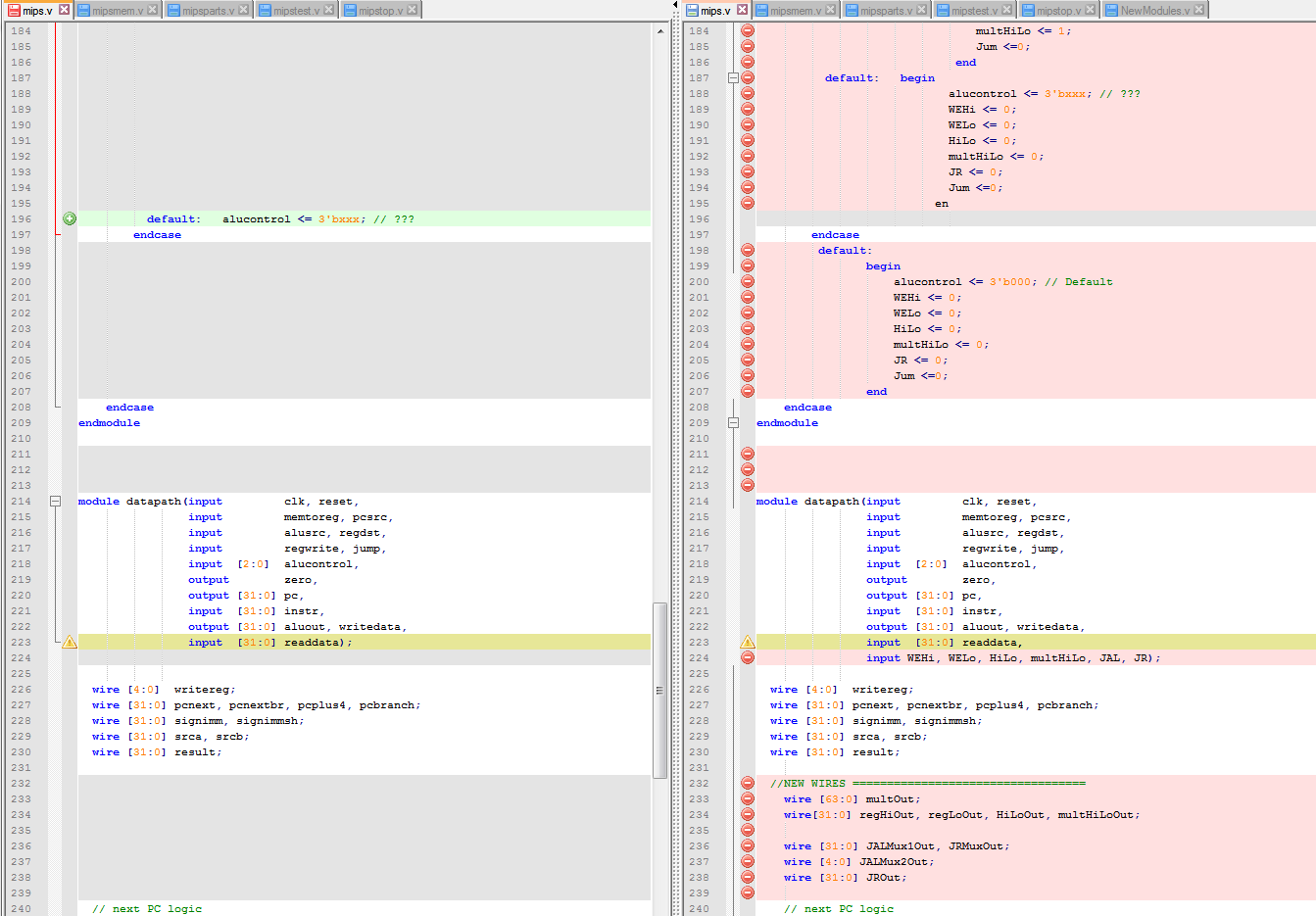
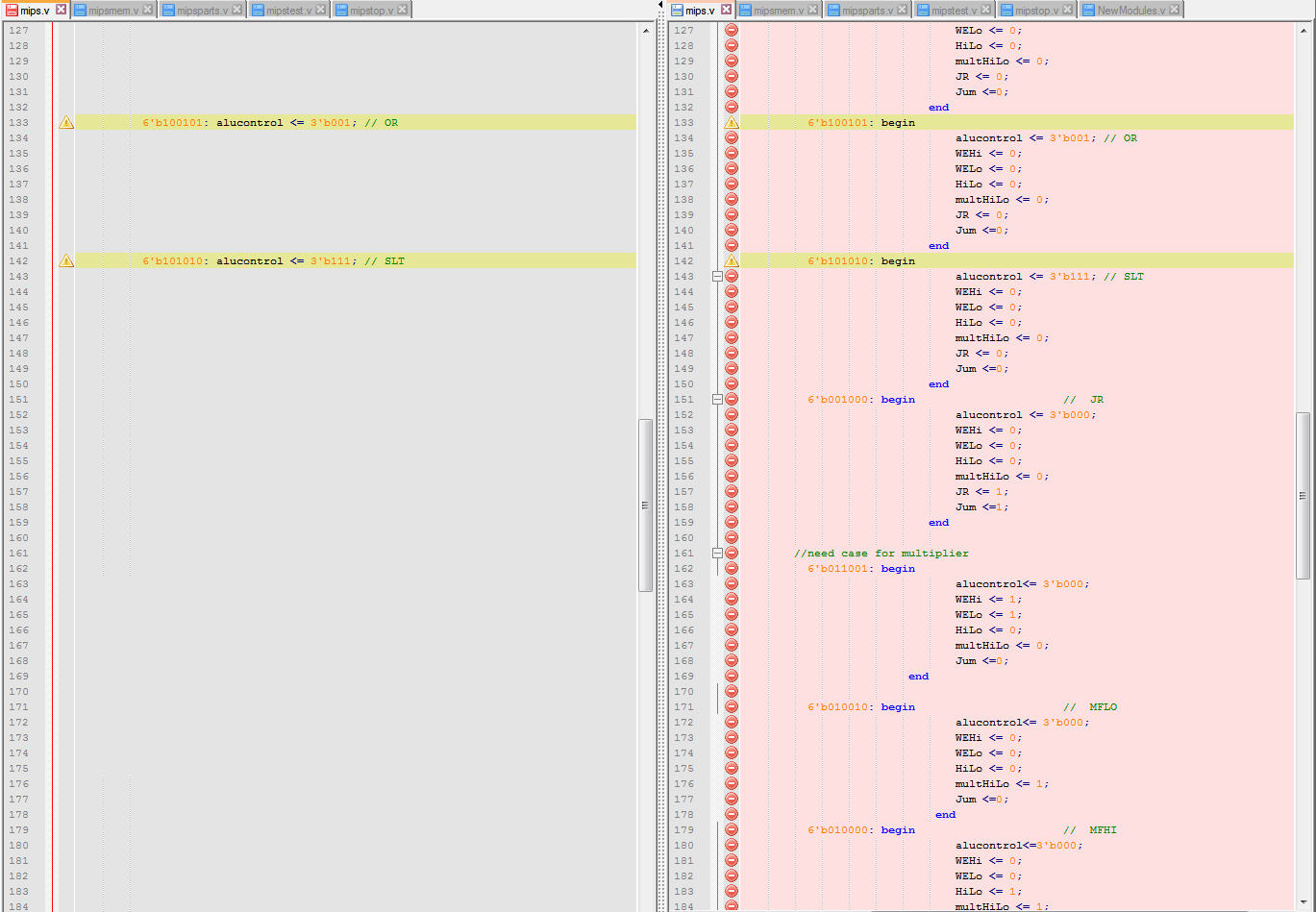
Figure 10. PC value followed by hexadecimal notation of second register of the register file.

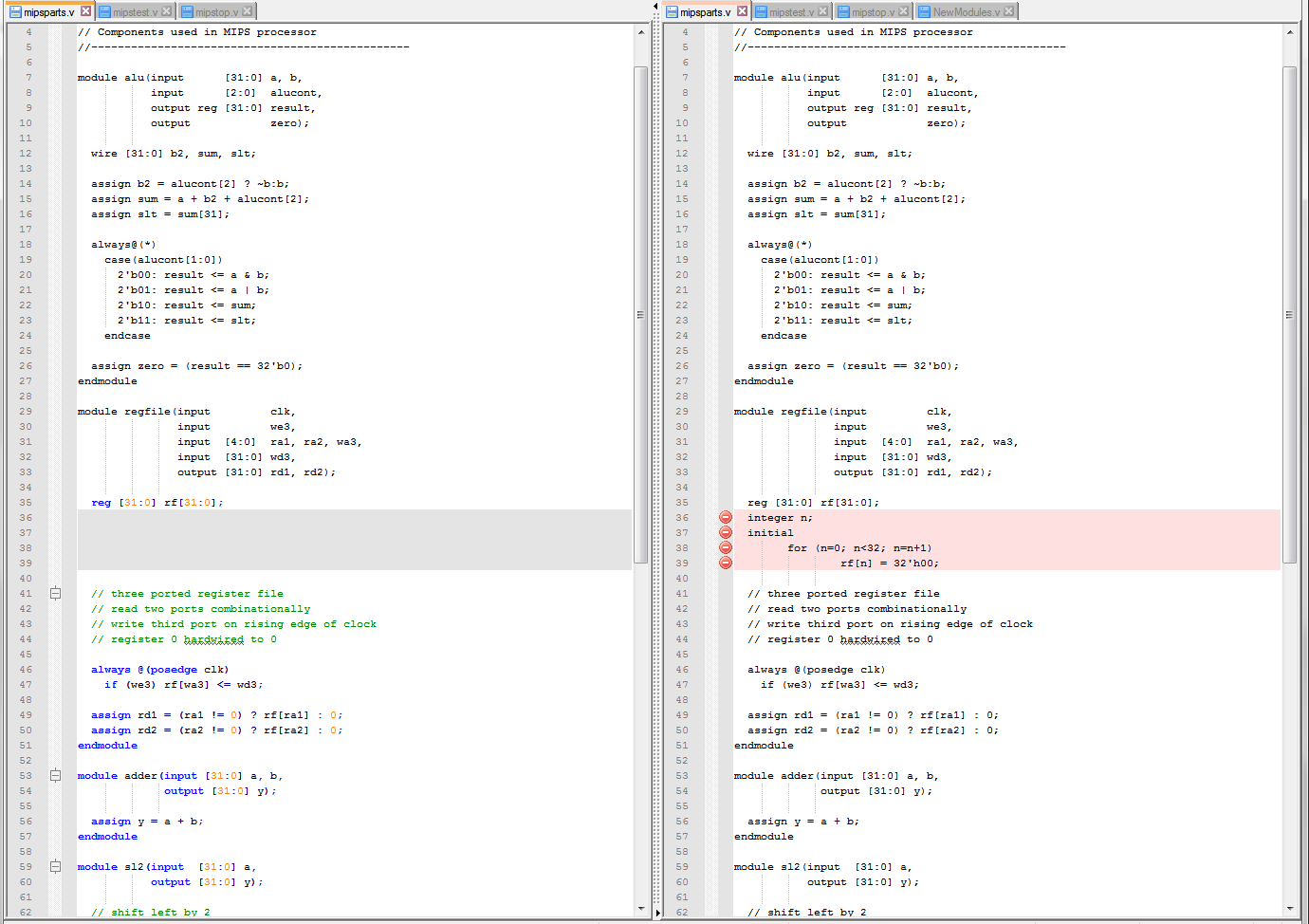
**Appendix A: Verilog Code: Original on Left, Modified on Right.**

**Note: Only Verilog files that have changed are included in this listing.**

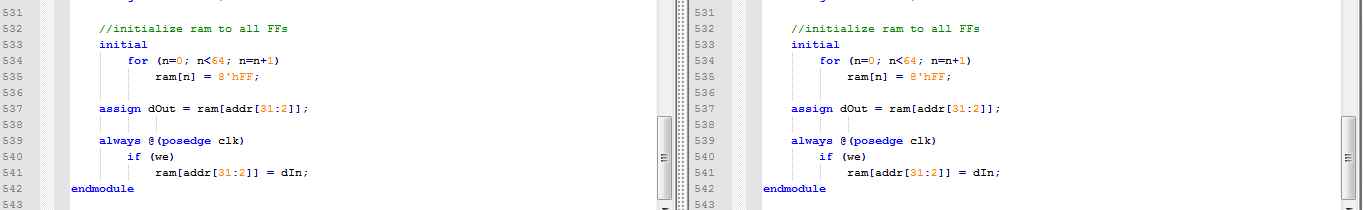
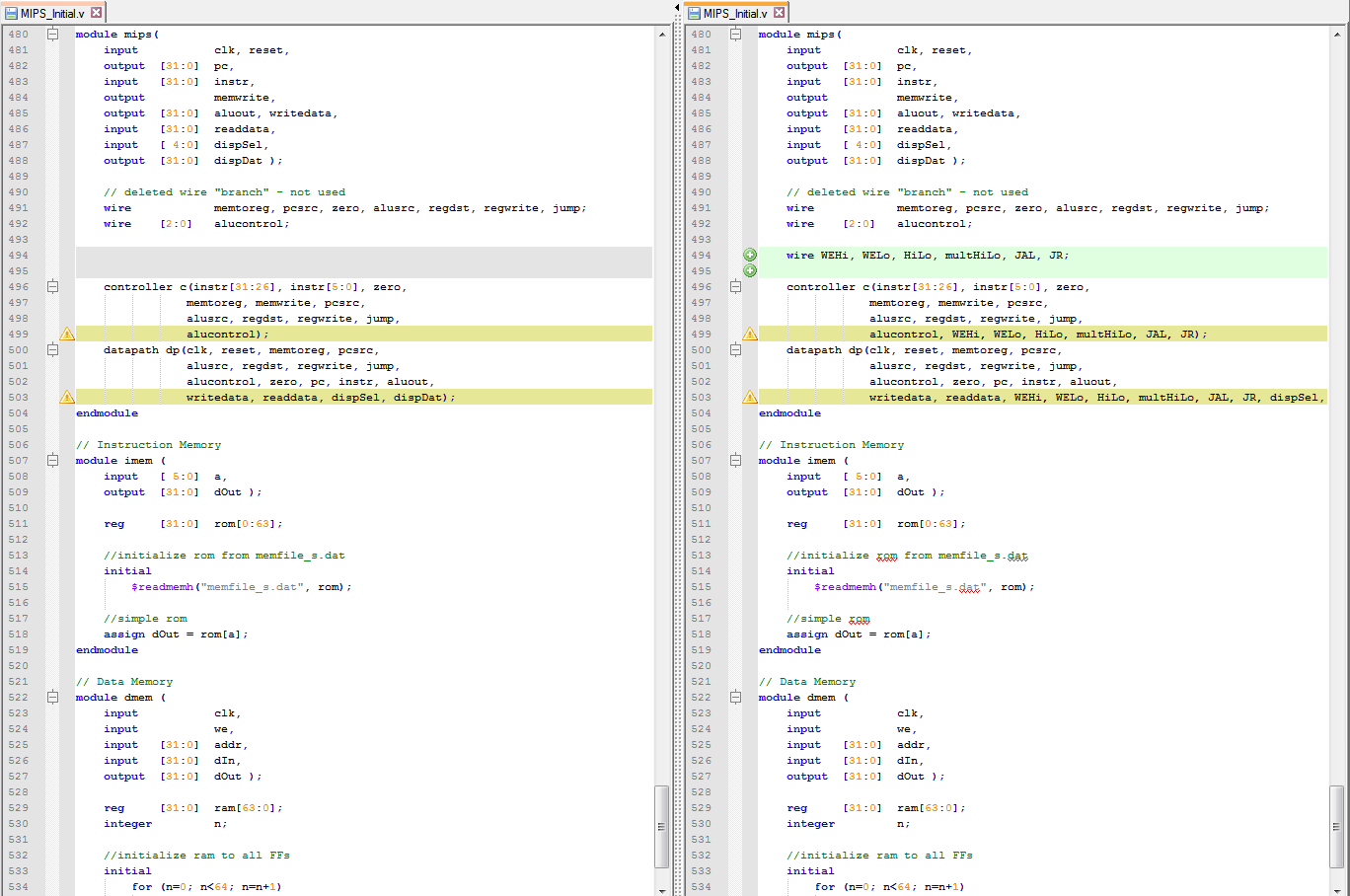
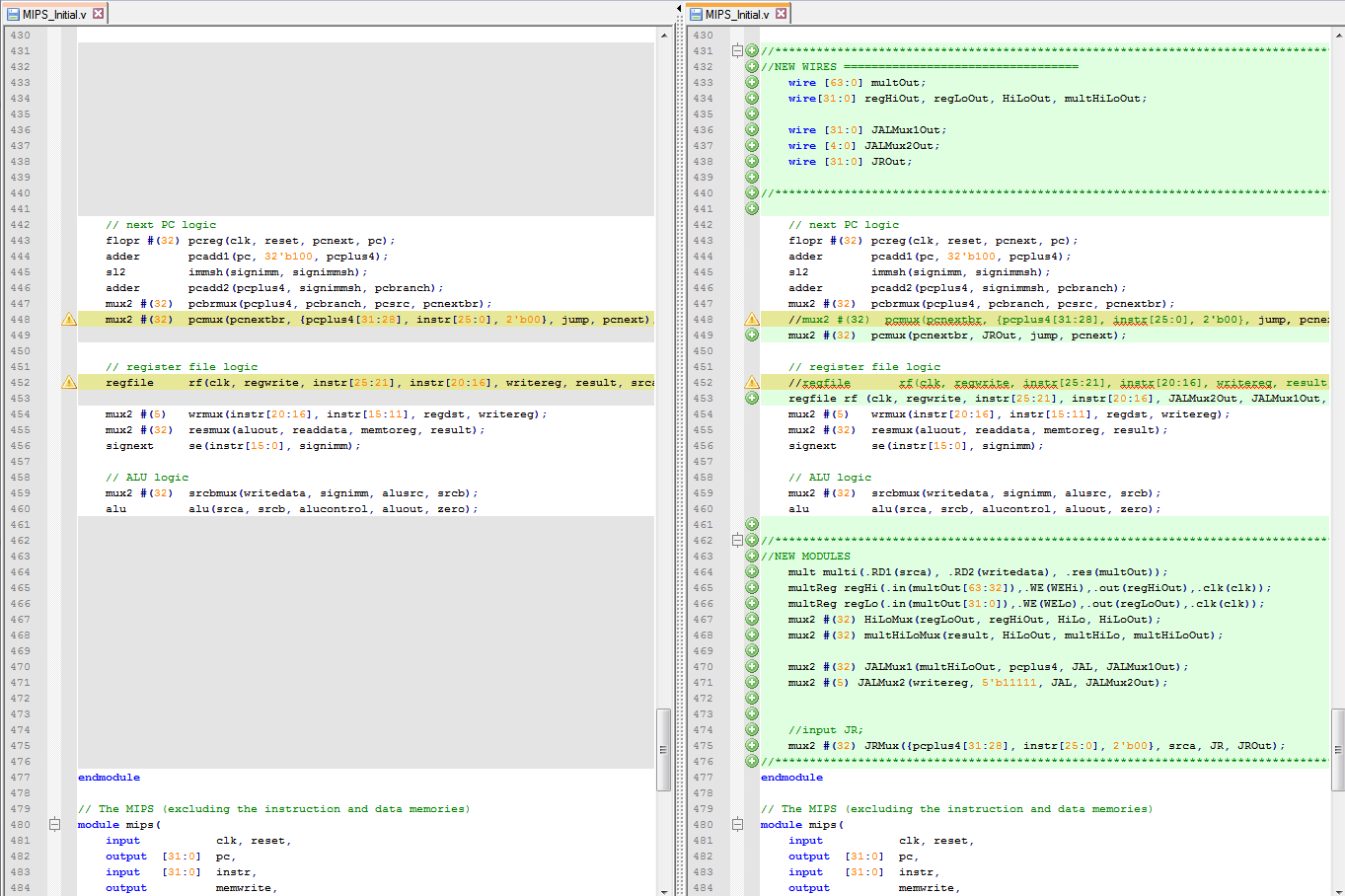
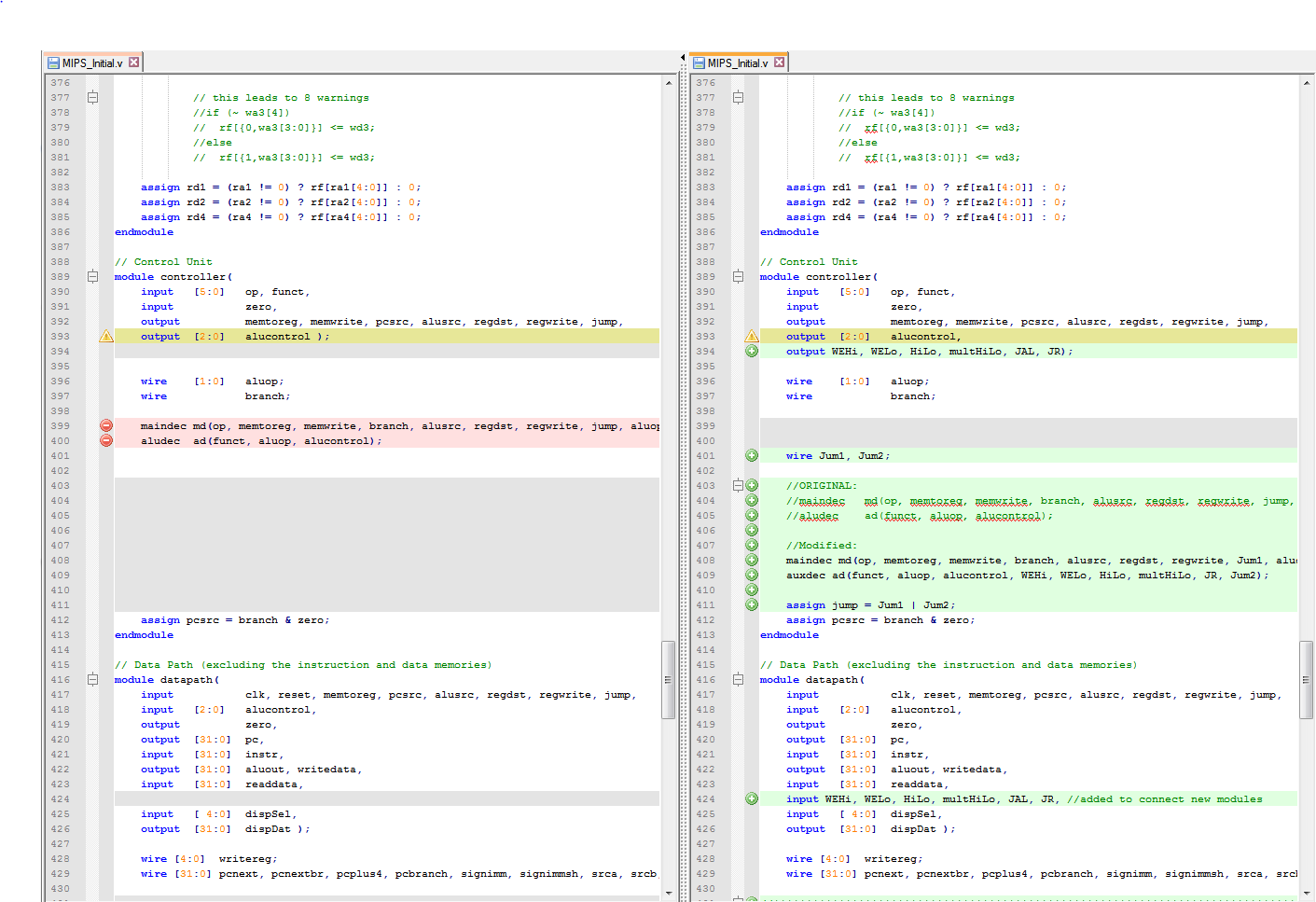
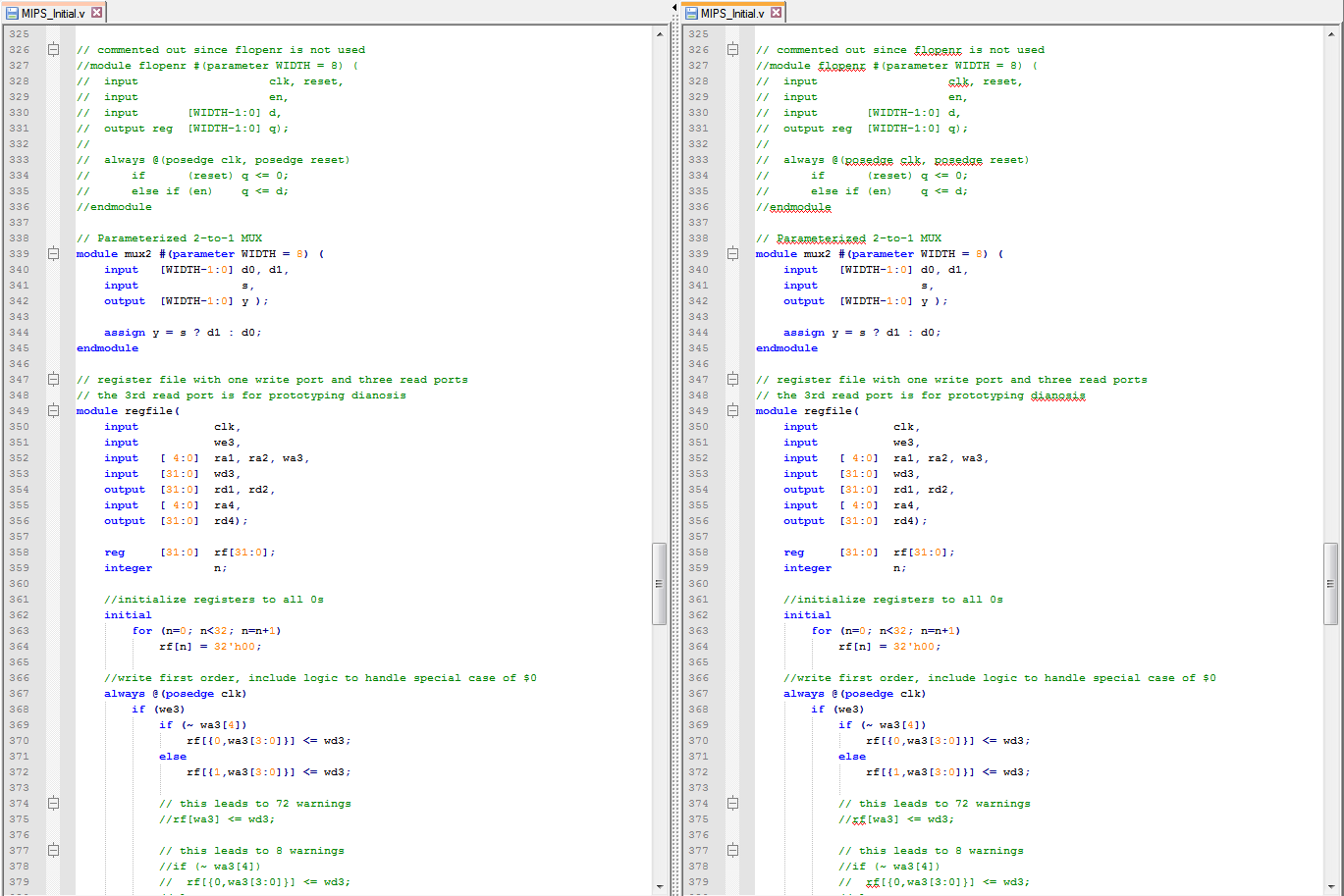
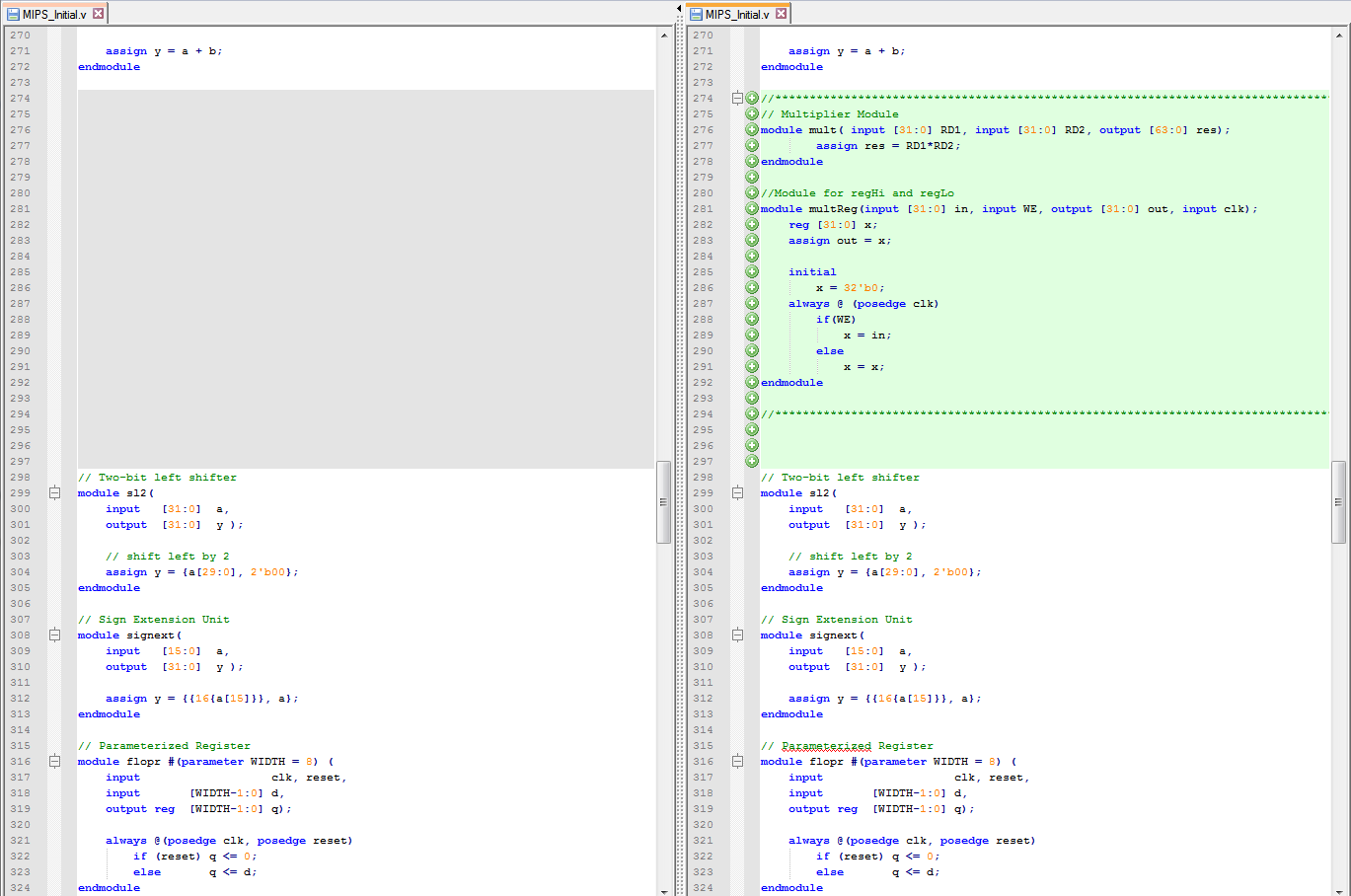
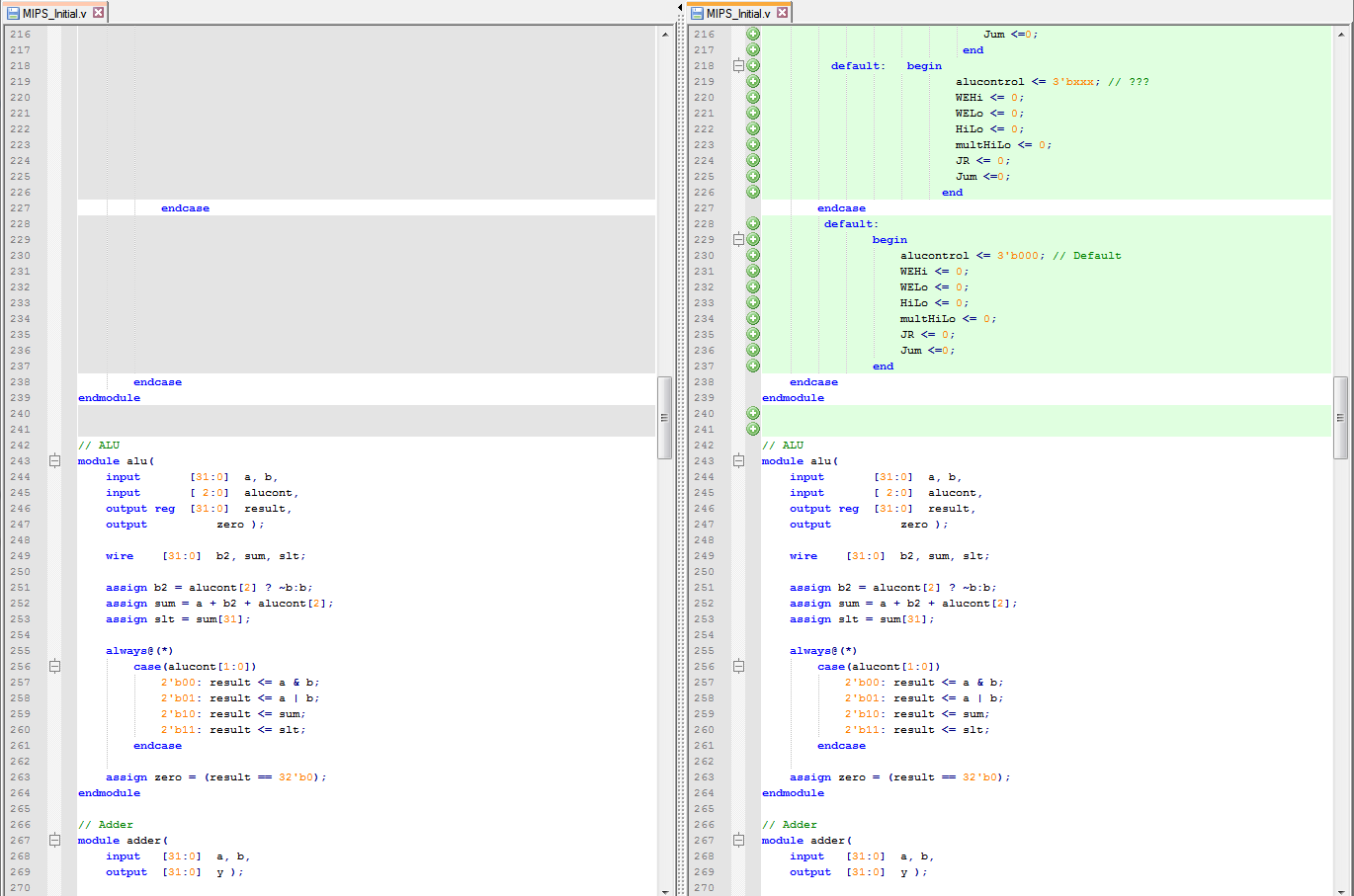
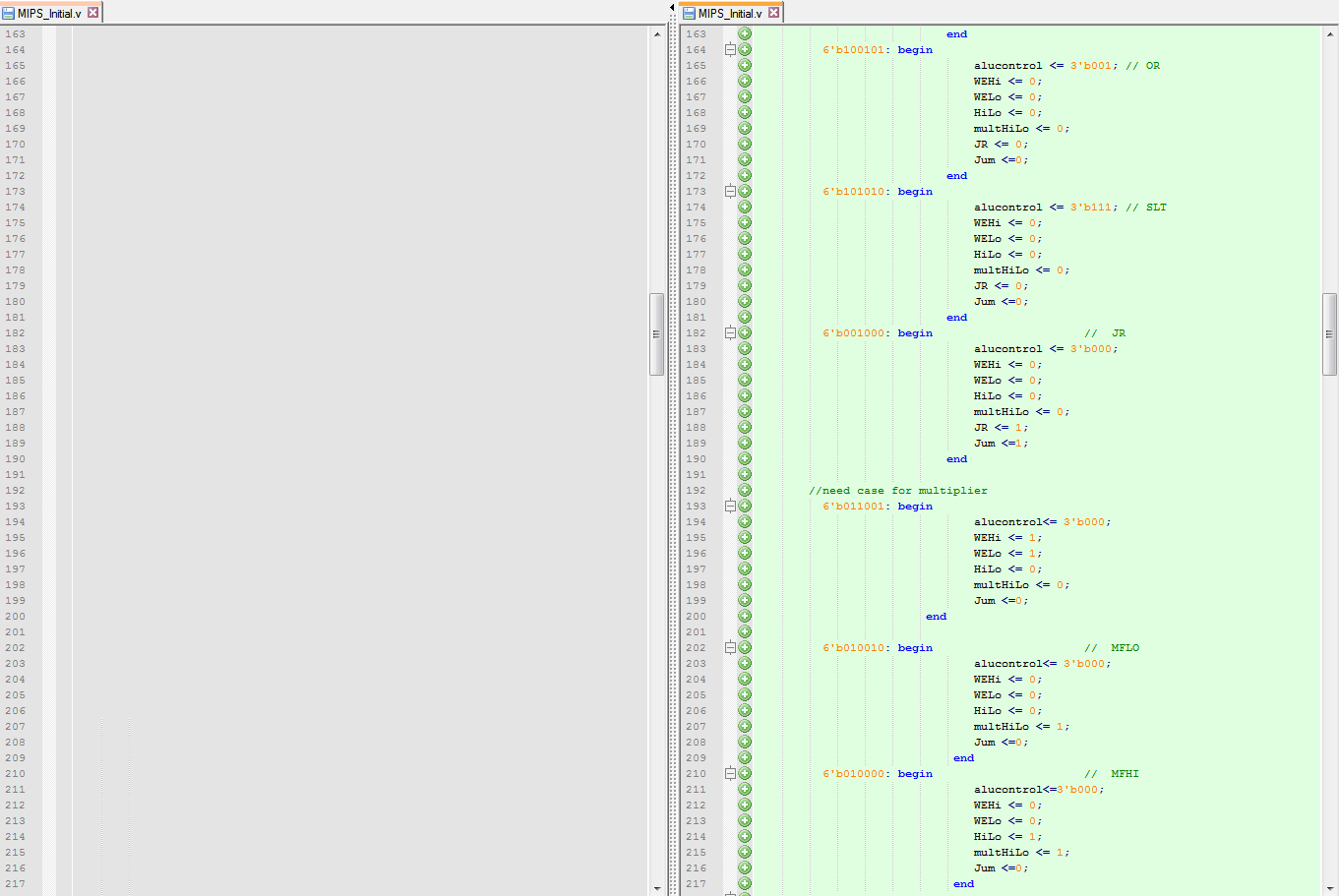
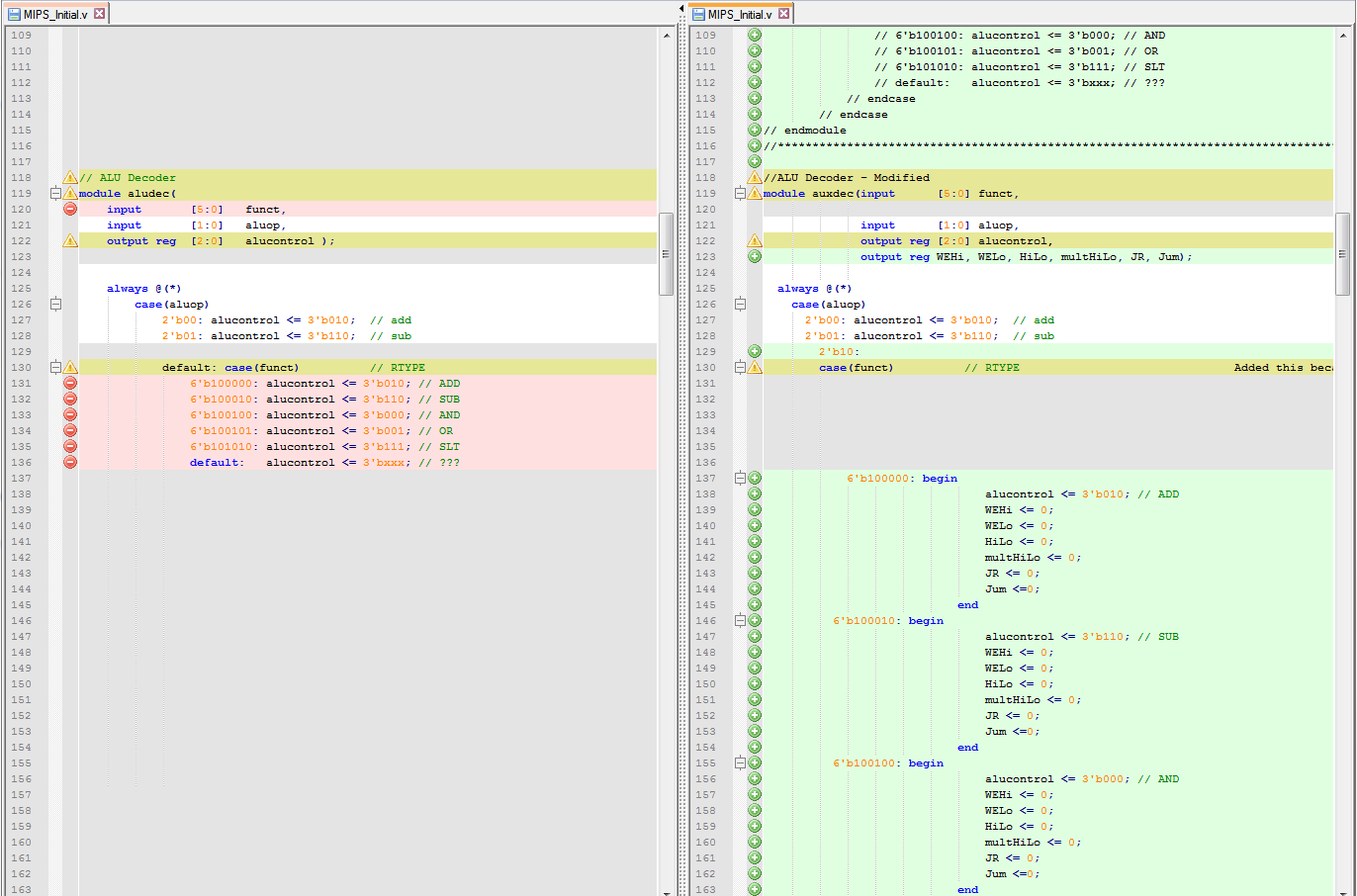
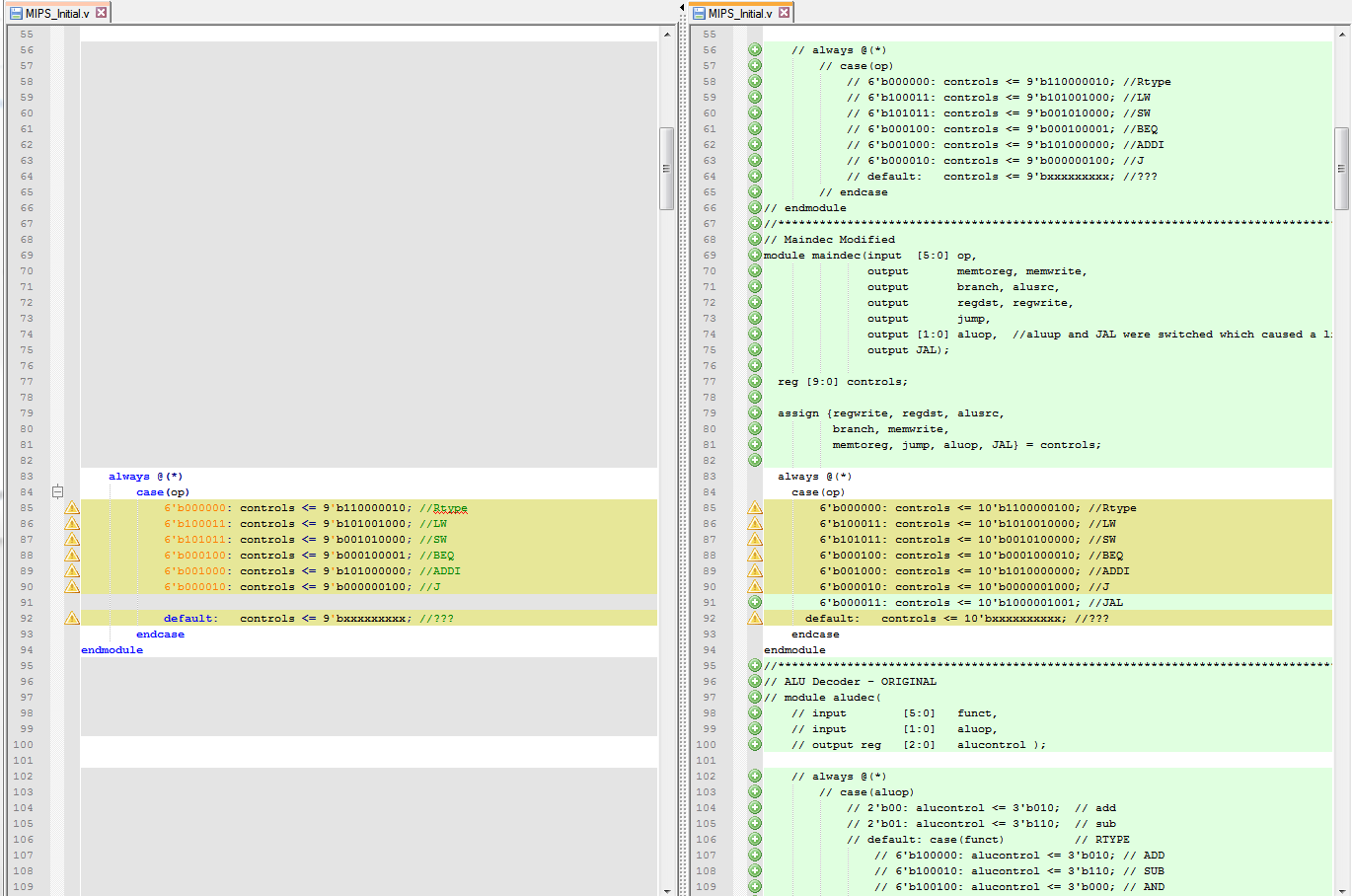
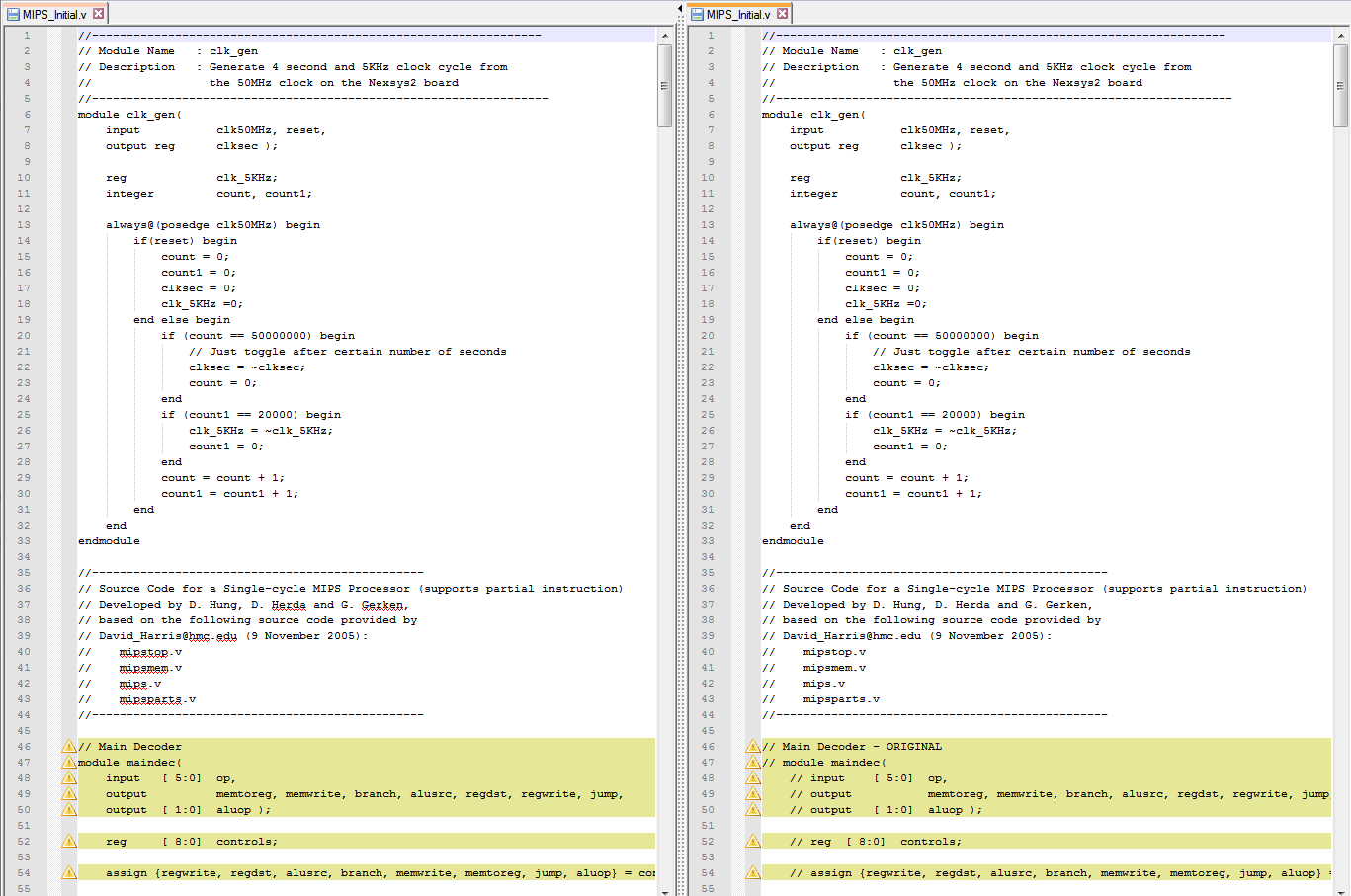
**Mips.v Lines 14 to 247:**

****

****

**Mipsparts.v (Only change)**

**Mipsinitial.v (for hardware implementation only) lines 1 to 543:**

****