Extending the Pipelined MIPS Verilog Model

ECE 472 Winter 2010

Ashtyn Moehlenhoff
Torben Rasmussen

Introduction

Pipelining the MIPS processor generates a speed-up by running multiple operations in paralell, instead of waiting for one operation to complete before starting the next. The basic model of the pipelined processor design (given to us) does not allow the jump instruction, nor does it perform data hazard detection or stalling. In order to fully exploit the benefits of the pipelined processor, three things must be added: 1) a jump instruction, 2) data hazard detection and forwarding, and 3) stalling.

Jump Instruction

The simple pipelined processor given to us does not allow jump instructions. Jump instructions are useful in order to move forward/backwards in the program. To add jump, we had to add a jump control signal and add to the control_pipeline.v. We also introduced a new MUX to select either pc+4 or the immediate jump location.

Because the processor is now pipelined, a nop must be inserted by the compiler after every jump instruction to ensure that when the jump is taken, no unwanted instructions are put into the pipeline.

Data Hazard Detection and Forwarding

Because the pipelined processor attempts to run multiple instructions in parallel, the processor must be able to detect dependencies between instructions and handle the dependency. The dependence of one instruction on an earlier instruction that is still in the pipeline is called a data hazard. An example of a data hazard is:

```
add $1, $0, $0;     IF    ID    EX    MEM   WB
add $2, $1, $1;           IF    ID    EX    MEM   WB
```

In this example you can see that the second instruction depends on the first. The register $1 will not be written back until after the fifth stage of the pipeline, but the second instruction requires the value of $1 for its EX stage. In order to solve this problem, we can forward the result of the EX stage from the first instruction to the input of the EX stage for the second instruction.

To detect data hazards, logic must be written to test whether an instruction depends on a previous instruction that is still in the pipeline:

```
always @(*)
begin
ForwardA = 2'b00;
ForwardB = 2'b00;

// EX Hazard
if(MEM_RegWrite && (MEM_RegRd != 0) && (MEM_RegRd == EX_rs))
       ForwardA = 2'b10;

if(MEM_RegWrite && (MEM_RegRd != 0) && (MEM_RegRd == EX_rt))
       ForwardB = 2'b10;

  //MEM Hazard
if(WB_RegWrite && (WB_RegRd != 0) && !(MEM_RegWrite && (MEM_RegRd !=0)
&& (MEM_RegRd != EX_rs) && (WB_RegRd == EX_rs)))
       ForwardA = 2'b01;

if(WB_RegWrite && (WB_RegRd != 0) && !(MEM_RegWrite && (MEM_RegRd !=0)
&& (MEM_RegRd != EX_rt) && (WB_RegRd == EX_rt)))
       ForwardB = 2'b01;
end
```

To forward the values, a MUX must be inserted before the EX stage to determine what to use for ALU_a and ALU_b. Here is a table of the control signal for the forwarding MUX and the output:

MUX control (ForwardA/ForwardB)	Output of MUX
00	From register file
01	From previous instruction's EX stage
10	From previous instruction's MEM stage

The two 4-to-1 MUX's we inserted are:

```
mux4 FWAMUX(ForwardA, EX_rd1, WB_wd, MEM_ALUOut, 32'd0, EX_fw_a_out);
mux4 FWBMUX(ForwardB, EX_rd2, WB_wd, MEM_ALUOut, 32'd0, EX_fw_b_out);
```

Stalling

If an instruction depends on a previous load word instruction, the instruction must wait for the word to be loaded from memory before it can continue:

```
lw   $1, 0($4)            IF    ID    EX    MEM   WB
add $2, $1, $1;           IF    ID    -     -     -
                                      EX    MEM   WB
```

Logic to detect load-use hazard:

```
if((EX_MemRead == 1'b1) && ((EX_rt == ID_rs) || (EX_rt == ID_rt))) Stall <= 1;
```

If a load-use hazard is detected, we insert a nop by setting the control signals of the EX stage to 0 and make sure the PC isn't incremented:

```
always @(posedge clk)
begin
       if((EX_MemRead == 1'b1) && ((EX_rt == ID_rs) || (EX_rt == ID_rt)))
       begin
                      Stall          <= 1;
                      EX_ALUOp       <= 0;
                      EX_ALUSrc   <= 0;
                      EX_Branch   <= 0;
                      EX_MemRead  <= 0;
                      EX_MemWrite <= 0;
                      EX_RegWrite <= 0;
                      EX_MemtoReg <= 0;
                      add_increment = 32'd0;
       end
       else
       begin
                      Stall          <= 0;
                      add_increment = 32'd4;
       end
  end
```

To make sure the PC doesn't increment, so we used the variable "add_increment" and set it to 4 whenever stalling doesn't happen and to 0 whenever the pipeline needs stalled. The "add_increment" value was then used as the input to the PC add unit :

```
add32         IF_PCADD(IF_pc, add_increment, IF_pc4);
```

We added to the IF/ID pipeline register to stall the instruction and pc in the case of a load-use hazard.:

```
else if(Stall)
begin
       ID_instr <= ID_instr;
       ID_pc4   <= ID_pc4;
end
```

Testing

In order to test the code we simulated the following sequence of instructions (in rom32.v):

```
5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd4 };             // lw $1, 4($0)        r1=1
5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };             // lw $2, 4($0)        r2=1
5'd2 : data_out = { 6'd0,  5'd1, 5'd2, 5'd3, 5'd0, 6'd32 }; // add $3, $1, $2  r3=r1+r2
5'd3 : data_out = { 6'd0,  5'd3, 5'd3, 5'd4, 5'd0, 6'd32 }; // add $4, $3, $3  r3=r1+r2
5'd4 : data_out = { 6'd2,  26'd0 };                         // j 0; restart loop
5'd5 : data_out = { 32'd0 };                                // nop
```

The waveform (figure 1) shows a scenario where both types of hazards are encountered and a jump instruction is executed. The pipeline stalls on the first add instruction to wait for the word to be loaded into register 2. You can see in our waveform that the Stall control signal is set high and the entire pipeline is stalled for one cycle. In the next cycle the add can execute with the forwarded values from the EX and MEM stages. The second add is dependent on the results from the first add, so the values are forwarded from the output of the previous instruction's EX stage. Finally, the jump instruction is executed and the program restarts.

Synthesis and Place & Route

Synthesis and Place & Route are the next steps (after design) necessary to tapeout a chip. We used the program Design Vision to synthesize our MIPS pipelined processor Verilog code.

The program Encounter was used to route our design. The hardware design can be seen in Figure 2.

The area, power and timing reports are as follow:

```
****************************************
Report : area
Design : mips_core
Version: B-2008.09-SP4
Date   : Tue Dec  7 12:42:13 2010
****************************************

Library(s) Used:

    tt_1v8_25c (File: /nfs/spectre/u8/vlsi/UMC180/tt_1v8_25c.db)

Number of ports:              267
Number of nets:               827
Number of cells:               19
Number of references:          19

Combinational area:       120711.730838
Noncombinational area:    97137.533005
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          217849.263843
Total area:                 undefined

***** End Of Report *****
```

```
****************************************
Report : power
        -analysis_effort low
Design : mips_core
Version: B-2008.09-SP4
Date   : Tue Dec  7 12:40:38 2010
****************************************


Library(s) Used:

    tt_1v8_25c (File: /nfs/spectre/u8/vlsi/UMC180/tt_1v8_25c.db)


Operating Conditions: tt_1v8_25c   Library: tt_1v8_25c
Wire Load Model Mode: top


Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   9.4551 mW   (95%)
  Net Switching Power  = 451.2024 uW    (5%)
                         ---------
Total Dynamic Power    =   9.9063 mW  (100%)

Cell Leakage Power     = 726.4437 nW

***** End Of Report *****


****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
        -sort_by group
Design : mips_core
```

```
Version: B-2008.09-SP4
Date   : Tue Dec  7 12:46:37 2010
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: tt_1v8_25c   Library: tt_1v8_25c
Wire Load Model Mode: top

  Startpoint: rnd_pass1/r5_o_reg_3_
              (rising edge-triggered flip-flop)
  Endpoint: wr_en_o[2] (output port)
  Path Group: (none)
  Path Type: max

  Point                                                    Incr       Path
  -----------------------------------------------------------------------------
  rnd_pass1/r5_o_reg_3_/CK (DFFHQX1)                       0.00 #     0.00 r
  rnd_pass1/r5_o_reg_3_/Q (DFFHQX1)                        0.36       0.36 r
  rnd_pass1/r5_o[3] (r5_reg_2)                             0.00       0.36 r
  iforward/fw_alu_rn[3] (forward)                          0.00       0.36 r
  iforward/fw_alu_rt/alu_wr_rn[3] (forward_node_3)         0.00       0.36 r
  iforward/fw_alu_rt/U17/Y (XOR2X1)                        0.20       0.56 r
  iforward/fw_alu_rt/U15/Y (NOR3X1)                        0.05       0.61 f
  iforward/fw_alu_rt/U14/Y (AND4X2)                        0.20       0.81 f
  iforward/fw_alu_rt/U2/Y (NOR2X1)                         0.28       1.09 r
  iforward/fw_alu_rt/mux_fw[1] (forward_node_3)            0.00       1.09 r
  iforward/alu_rt_fw[1] (forward)                          0.00       1.09 r
  iexec_stage/muxb_fw_ctl[1] (exec_stage)                  0.00       1.09 r
  iexec_stage/i_alu_muxb/fw_ctl[1] (alu_muxb)              0.00       1.09 r
  iexec_stage/i_alu_muxb/U103/Y (XOR2X1)                   0.19       1.28 f
  iexec_stage/i_alu_muxb/U102/Y (NAND3BX1)                 0.06       1.34 r
  iexec_stage/i_alu_muxb/U101/Y (MX2X1)                    0.67       2.01 r
  iexec_stage/i_alu_muxb/U100/Y (AOI22X1)                  0.09       2.09 f
  iexec_stage/i_alu_muxb/U95/Y (NAND2X1)                   0.46       2.56 r
  iexec_stage/i_alu_muxb/b_o[0] (alu_muxb)                 0.00       2.56 r
  iexec_stage/MIPS_alu/b[0] (mips_alu)                     0.00       2.56 r
  iexec_stage/MIPS_alu/mips_alu/b[0] (alu)                 0.00       2.56 r
  iexec_stage/MIPS_alu/mips_alu/r92/B[0] (alu_DW01_addsub_0)
                                                           0.00       2.56 r
  iexec_stage/MIPS_alu/mips_alu/r92/U1/Y (XOR2XL)          0.25       2.80 f
  iexec_stage/MIPS_alu/mips_alu/r92/U1_0/CO (ADDFXL)       0.69       3.50 f
  iexec_stage/MIPS_alu/mips_alu/r92/U1_1/CO (ADDFX2)       0.20       3.70 f
  iexec_stage/MIPS_alu/mips_alu/r92/U1_2/CO (ADDFX2)       0.19       3.89 f
```

```
iexec_stage/MIPS_alu/mips_alu/r92/U1_3/CO (ADDFX2)      0.19       4.08 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_4/CO (ADDFX2)      0.19       4.27 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_5/CO (ADDFX2)      0.19       4.46 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_6/CO (ADDFX2)      0.19       4.65 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_7/CO (ADDFX2)      0.19       4.84 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_8/CO (ADDFX2)      0.19       5.02 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_9/CO (ADDFX2)      0.19       5.21 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_10/CO (ADDFX2)     0.19       5.40 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_11/CO (ADDFX2)     0.19       5.59 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_12/CO (ADDFX2)     0.19       5.78 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_13/CO (ADDFX2)     0.19       5.97 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_14/CO (ADDFX2)     0.19       6.16 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_15/CO (ADDFX2)     0.19       6.35 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_16/CO (ADDFX2)     0.19       6.54 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_17/CO (ADDFX2)     0.19       6.73 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_18/CO (ADDFX2)     0.19       6.92 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_19/CO (ADDFX2)     0.19       7.11 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_20/CO (ADDFX2)     0.19       7.30 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_21/CO (ADDFX2)     0.19       7.49 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_22/CO (ADDFX2)     0.19       7.68 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_23/CO (ADDFX2)     0.19       7.87 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_24/CO (ADDFX2)     0.19       8.06 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_25/CO (ADDFX2)     0.19       8.24 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_26/CO (ADDFX2)     0.19       8.43 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_27/CO (ADDFX2)     0.19       8.62 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_28/CO (ADDFX2)     0.19       8.81 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_29/CO (ADDFX2)     0.19       9.00 f
iexec_stage/MIPS_alu/mips_alu/r92/U1_30/S (ADDFX2)      0.21       9.21 r
iexec_stage/MIPS_alu/mips_alu/r92/SUM[30] (alu_DW01_addsub_0)
                                                        0.00       9.21 r
iexec_stage/MIPS_alu/mips_alu/U310/Y (CLKINVX1)         0.06       9.27 f
iexec_stage/MIPS_alu/mips_alu/U309/Y (OAI22X1)          0.14       9.41 r
iexec_stage/MIPS_alu/mips_alu/r86/B[30] (alu_DW01_add_0)
                                                        0.00       9.41 r
iexec_stage/MIPS_alu/mips_alu/r86/U1_30/CO (ADDFX2)     0.35       9.76 r
iexec_stage/MIPS_alu/mips_alu/r86/U1_31/CO (ADDFX2)     0.18       9.94 r
iexec_stage/MIPS_alu/mips_alu/r86/U1_32/Y (XOR3X2)      0.12      10.06 f
iexec_stage/MIPS_alu/mips_alu/r86/SUM[32] (alu_DW01_add_0)
                                                        0.00      10.06 f
iexec_stage/MIPS_alu/mips_alu/U286/Y (AOI221XL)         0.19      10.25 r
iexec_stage/MIPS_alu/mips_alu/U197/Y (NAND2X1)          0.04      10.29 f
iexec_stage/MIPS_alu/mips_alu/alu_out[0] (alu)          0.00      10.29 f
iexec_stage/MIPS_alu/U32/Y (OR2X1)                      0.19      10.48 f
iexec_stage/MIPS_alu/c[0] (mips_alu)                    0.00      10.48 f
```

```
  iexec_stage/alu_ur_o[0] (exec_stage)                    0.00      10.48 f
  MEM_CTL/dmem_addr_i[0] (mem_module)                     0.00      10.48 f
  MEM_CTL/i_mem_addr_ctl/addr_i[0] (mem_addr_ctl)         0.00      10.48 f
  MEM_CTL/i_mem_addr_ctl/U10/Y (MXI2X1)                   0.11      10.59 f
  MEM_CTL/i_mem_addr_ctl/U4/Y (OAI2BB1X1)                 0.12      10.71 f
  MEM_CTL/i_mem_addr_ctl/wr_en[2] (mem_addr_ctl)          0.00      10.71 f
  MEM_CTL/Zz_wr_en[2] (mem_module)                        0.00      10.71 f
  wr_en_o[2] (out)                                        0.00      10.71 f
  data arrival time                                                 10.71
  ------------------------------------------------------------------------------
  (Path is unconstrained)

***** End Of Report *****
```

Conclusion

The group spent approximately 20 hours total on this project. The most valuable part was learning the intricacies of hazard detection on a pipelined MIPS processor. The least valuable part was creating the hardware design for the part, simple because it was pre-scripted for us.

In the future it may be valuable to limit the amount of starter code that teams are given. It would promote creativity, and would waste less time learning how to interpret others' code.

Appendix

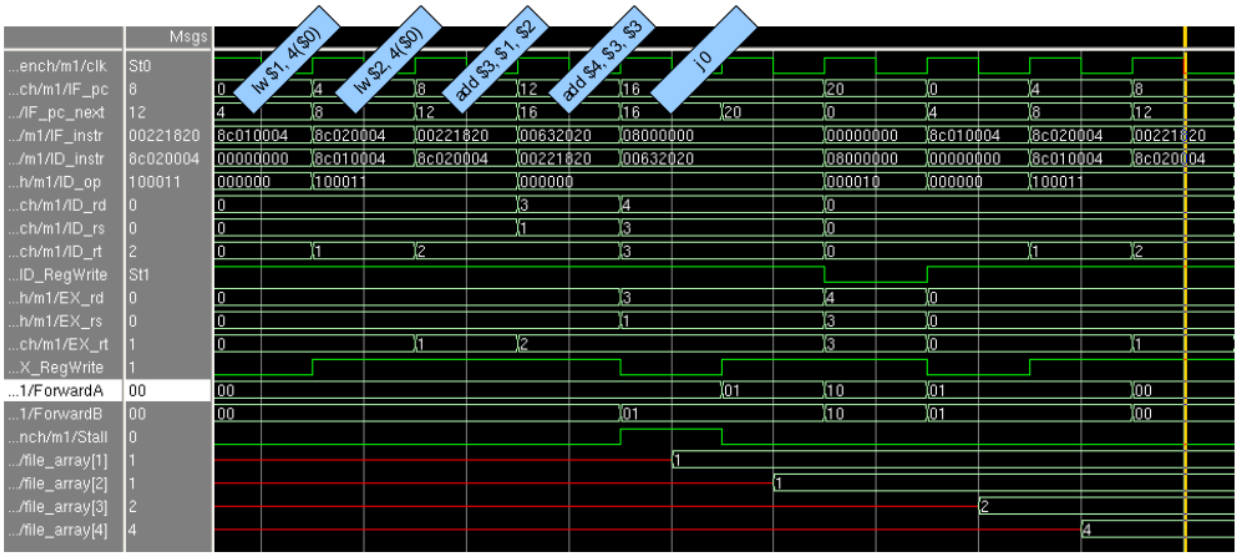

Figure 1

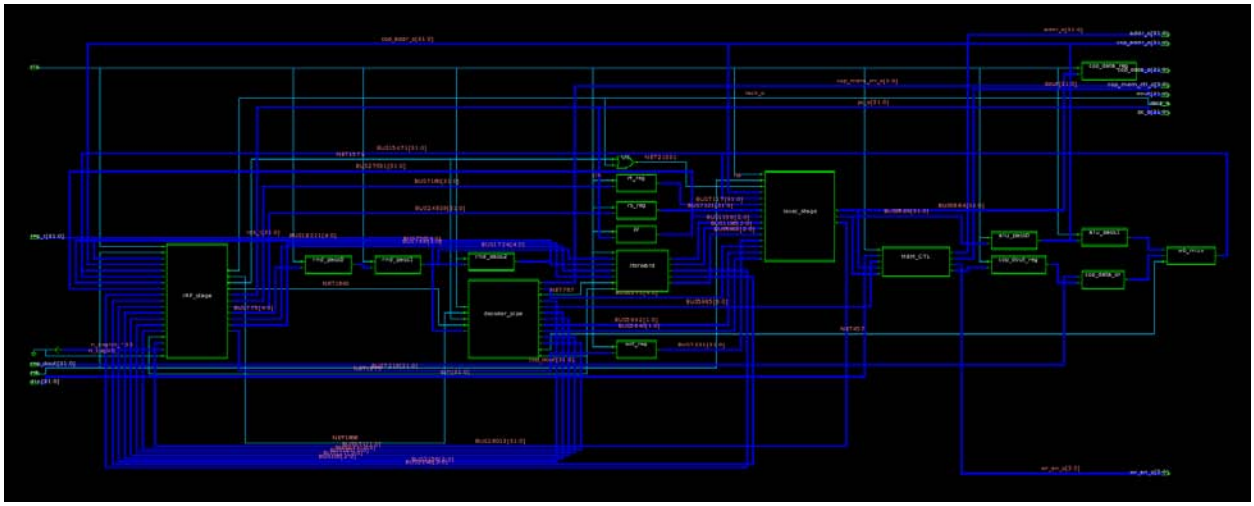

Figure 2