UM-SJTU Joint Institute
Ve370 Introduction to Computer Organization

Yuanpeng
Jiang Yifeng
Li Bo

November 19th, 2014

Design of Single Cycle and Pipeline Processor
# Table of Contents

1 Objectives........................................................................................................... 4
2 Problem Definition .................................................................................................. 4
3 System partitioning ............................................................................................... 4
   3.1 Single cycle processor .................................................................................... 4
   3.2 Pipeline processor .......................................................................................... 4
4 Design entry .......................................................................................................... 6
   4.1 Single cycle processor .................................................................................... 6
      4.1.1 Data memory ........................................................................................... 6
      4.1.2 Instruction memory .................................................................................. 6
      4.1.3 Register file ............................................................................................. 6
      4.1.4 ALU ......................................................................................................... 7
      4.1.5 Control ...................................................................................................... 7
      4.1.6 ALU control ............................................................................................. 7
      4.1.7 PC ............................................................................................................. 7
   4.2 Pipeline processor ............................................................................................ 8
      4.2.1 D-memory, I-memory, Register file, ALU, Control, ALU control .......... 8
      4.2.2 PC ........................................................................................................... 8
      4.2.3 Comparator ............................................................................................... 8
      4.2.4 Forwarding unit1 ..................................................................................... 8
      4.2.5 Forwarding unit2 (for comparator) .......................................................... 9
      4.2.6 Hazard detection unit ............................................................................... 9
      4.2.7 Pipeline registers .................................................................................... 10
5 Test Plan ................................................................................................................ 11
   5.1 Test Plan for the Single-cycle processor ....................................................... 11
   5.2 Test Plan for the Pipeline processor ............................................................... 12
6 Simulation Results ................................................................................................. 13
   6.1 Simulation results for the Single-cycle processor ......................................... 14
   6.2 Simulation results for the Single-cycle processor ......................................... 16
7 Conclusions

8 Appendix

A-Verilog Source Files for the Single-cycle processor

B-Verilog Source Files for the Pipeline CPU (Duplicate Files Omitted)

C-RTL schematic for the Single-cycle processor

D-RTL schematic for the Pipeline processor
1 Objectives

Model both single cycle and pipeline implementation of MIPS computer in Verilog that support a subset of MIPS instruction set including:

- The memory-reference instructions load word (lw) and store word (sw)
- The arithmetic-logical instructions add, addi, sub, and, andi, or, and slt
- The jumping instructions branch equal (beq) and jump (j)

2 Problem Definition

The problem is to design a Verilog module that can do the MIPS instruction: lw, sw, add, addi, sub, andi, or, slt, beq and j and then store the result in the register file. The design should be based on the single cycle and pipeline processor.

3 System partitioning

3.1 Single cycle processor

The design of the single cycle processor is based on the Figure 1 (source: Computer Organization and Design, by Patterson and Hennessy, Morgan Kaufmann Publishers)

![Figure 1 Single cycle processor](image)

3.2 Pipeline processor

The design of the single cycle processor is based on the Figure 2.
Figure 2 Pipeline processor
4 Design entry

4.1 Single cycle processor

4.1.1 Data memory

The data memory is used to store the data. We do use the clock signal for the data memory. The write and read state of the data memory depends on the signal read, write and clk. When clk is 1 and write is 1, the write is enabled. When clk is 0 and read is 1, the read is enabled. When using the data memory, rnum and wnum are connected to the same wire.

![Figure 3 data memory](image)

4.1.2 Instruction memory

The instruction memory is used to store the instructions. We use the same data memory module for instruction memory. The only difference is that the write input for instruction memory is always 0 and read input is always 1.

4.1.3 Register file

Register file is similar to data memory, when clk is 1 and write is 1, the write is enabled. When clk is 0 and read is 1, the read is enabled. The only difference is that there are two rnum input and two rdata output for register file.

![Figure 4 register file](image)
Ve370 project 2 report

4.1.4 ALU

The ALU can do addition, bitwise and, bitwise or, subtraction, set less than and bitwise compare. We write the carry-lookahead adder for the ALU to improve the efficiency of the component. The ALU control signal input is ALUctl.

![Figure 5 ALU](image)

4.1.5 Control

The control is totally designed in combinational circuit to improve the efficiency of the component. The control for signal single processor cycle is that same as for pipeline processor. The only difference is that when using the control in signal single processor, IF_flush will not be connected.

![Figure 6 Control](image)

4.1.6 ALU control

The ALU control is used to control the ALU. It is totally designed in combinational circuit to improve the efficiency of the component. The ALUop is the EX[2:1] from the control.

![Figure 7 ALU control](image)

4.1.7 PC

PC is a positive edge triggered 32 bits register.
4.2 Pipeline processor

4.2.1 D-memory, I-memory, Register file, ALU, Control, ALU control

These components are the same as single cycle processor. IF_Flush of Control will be connected to the IF/ID register to solve the control hazard.

4.2.2 PC

PC is a positive edge triggered 32 bits register, with a control input ctr_hold. When signal ctr_hold is 1, the data in register PC will not be changed.

4.2.3 Comparator

To improve the efficiency of the circuit, we put the branch operation to the ID stage. The 32 bit comparator will provide a signal to control to determine if the program will take branch.

4.2.4 Forwarding unit1

The forwarding unit1 is used to solve the data hazard for include EX data hazard and MEM data hazard. We check if the register number rs and rt for the EX stage is the same as the write back destination of MEM and WB stage. If so, the forward unit will forward the data from MEM or WB stage to the EX stage. The priority of EX data hazard is higher than the MEM data hazard which means that when double data hazard happens, the forwarding unit1 will forward the EX/MEM register data to the EX stage.
4.2.5 Forwarding unit2 (for comparator)

In order to solve the data hazard for comparator, we design the forwarding unit2. The forwarding unit2 have two functions. The first one is to forwarding, if the data that is needed to determine branch is still in the MEM stage or WB stage, the forwarding unit2 will forward the data to the ID stage. The second function is to insert nop. When the data that is needed is from a ALU instruction in EX stage or from a load word instruction in MEM stage (load word in EX stage will be solved by hazard detection unit), the output flush will be 1 and be given to hazard detection unit to insert a nop.

4.2.6 Hazard detection unit

Hazard detection unit have two functions. The first one is when the instruction in EX stage is load word and the instruction in ID stage need the write back data of load word, the hazard detection unit will insert a nop by flush the control signal of ID stage and hold the PC and IF/ID register. The second function is that when the forwarding unit2 give a signal flush is 1, the hazard detection unit will also insert a nop by the same method.
4.2.7 Pipeline registers

The pipeline registers are positive edge triggered registers. The special one is IF/ID register. The IF/ID register has a ctr_hold signal and a ctr_flush signal. When ctr_hold is 1, the data in IF/ID register will not change. When ctr_flush is 1, the data in IF/ID register will change to zero. The priority of ctr_hold is higher than ctr_flush. When ctr_hold and ctr_flush are both 1, the data in register will not change.
5 Test Plan

5.1 Test Plan for the Single-cycle processor

We need to test both the functionality of our single-cycle processor and pipeline processor. For the single-cycle processor, since there is no hazard, we only need to check all ten kinds of instructions (lw, sw, add, addi, or, sub, and, and, slt, jw, beq) implemented in our design give expected results. The detailed test plan is given below. Note that for all tests, we will only use $s0-$s6 to make the textual simulation simpler and clearer.

Test Case 1. lw & sw & addi:

```
addi $s0, $zero, 7
addi $s1, $zero, 16
sw $s0, -4($s1)  # with offset
addi $s1, $s1, -4
lw $s2, 0($s1)   # with zero offset, result should be $s2 = $s0 = 7
```

The corresponding machine code initiating into the Instruction Memory and the simulation results are given in next section.

Test Case 2. add & andi & or & sub:

```
addi $s0, $zero, 63  # 6'b111111
andi $s1, $s0, 25    # $s1 = 25 (6'b011001)
add $s2, $s0, $s1    # $s2 = 25+63 = 88 (7'b1011000)
or $s3, $s2, $s1     # $s3 = 89 (7'b1011001)
sub $s0, $s2, $s3    # $s0 = 88-89 = -1
```

Test Case 3. slt & beq & j:

```
add $s0, $zero, $zero # $s0 = 0
add $s4, $zero, $zero # $s4 = 0
addi $s1, $zero, 4 # $s1 = 4
Loop: slt $s2, $s0, $s1 # $s2 = ($s0<$s1)?1:0
beq $s2, $zero, 3  # Exit if $s0 >= $s1
addi $s0, $s0, 1
add $s4, $s4, $s0   # $s4 += $s1
j 3 # jump back to Loop
Exit: add $s5, $s4, $zero # if successfully jump out of the loop, we should be able to see that $s5 = $s4 = 10
```
5.2 Test Plan for the Pipeline processor

Firstly we will again use case 1~3 above to show our pipeline processor can realize the same basic functions as our Single-cycle processor. Still, the more important part of testing pipeline processor is cases with data dependences and/or control flow since Pipeline processor may have data and control hazards. The test cases concerning different kinds of hazards is given below.

Test Case 4. EX data hazard, MEM data hazard, double data hazard (structure hazard)

```
addi $s0, $zero, 17 #$s0 = 17
addi $s1, $s0, 25 #$s1 = 17+25 = 42, EX hazard for I type
add $s2, $s0, $s1 # $s2 = 42+17 = 59, EX/double hazard for R type
sub $s3, $s0, $s1 # $s3 = 17-42 = -25, MEM hazard for R type, (exactly) no hazard for $s0 (structure hazard)
andi $s4, $s2, 63 #$s4 = $s2 = 59 (63 = 6'b111111)
```

Test Case 5. Load-use data hazard for R type (I type) and for branch beq

```
addi $s0, $zero, 7
addi $s1, $zero, 16
addi $s3, $zero, 3 #Initiate $s3 to 3
sw $s0, -4($s1) # EX/MEM hazard for I type 'sw'
addi $s1, $s1, -4
lw $s2, 0($s1) # $s2 = $s0 = 7, EX hazard for 'lw'
beq $s2, $s0, 1 # load-use hazard for beq, since the beq comparator is in ID stage, the processor should hold for two clock cycles and then 'take' the branch
lw $s3, 0($s1) # $s3 should still be 3 because this instruction is skipped by beq (control hazard)
lw $s4, 0($s1) # $s4 = $s2 = $s0 = 7
add $s5, $s3, $s4 # $s5 = 7+3 = 10, load-use hazard for R type (rt), computer stalled for one clock cycle
lw $s1, 0($s1) # $s1 = 7
addi $s2, $s1, 12 # $s2 = 7+12 = 19, load-use hazard for I type (rs), computer should stall for one clock cycle
```

Test Case 6. Data hazard for branch beq
addi $s1, $zero, 7 # $s1 = 7
addi $s0, $zero, 7 # $s0 = $s1 = 7
add $s1, $s1, $s0 # change $s1 = 14
beq $s0, $s1, 1 # beq not taken, data hazards for both $s0 and $s1. Note that beq is supposed to hold for one clock cycle and then compare the new $s1(14) with $s0. Wrong use of old $s1(7) will flush the following instruction and make the results totally no sense.
addi $s3, $zero, 3 # $s3 = 3, should not be skipped by beq
addi $s4, $zero, 4 # $s4 = 4

Remark: Control hazard for branch beq is already covered in test case 5 & 6 above (taken & not taken). For control hazard of jump j, we just use case 3 since we will run cases 1~3 for both single-cycle and pipeline as we stated before.

6 Simulation Results

We simulate these MIPS assembly test cases by giving Program Counter and Instruction Memories initial values (the corresponding machine codes). We put "monitors" (Monitor function in Verilog) in the register file so that we could see the registers' values. The Verilog segments relating to simulation are displayed below in Figure 14-17.

```
initial begin
  for(k=0;k<64;k=k+1) begin
    Imemory[k] = 32'b0;
  end
  Imemory[0] = 32'h20100007;
  Imemory[1] = 32'h20110010;
  Imemory[2] = 32'hae30fff0;
end
```

Figure 14  The initialization of Imemory in the Instruction Memory module. (Corresponding machine codes (32-bit binary) from different test cases will be generated from PCSim separately.)

```
initial begin
  $display("******************************************");
  $display("The textual simulation results:");
  $display("******************************************");
  $monitor("Stim," ,clk, Reg[16], Reg[17], Reg[18], Reg[19], Reg[20], Reg[21]);
end
```

Figure 15  The monitors in the Register File module. (Only $s0~$s5 is used for simplicity.)

```
initial begin
  pcdat0= 32'h00400000;
end
```

Figure 16  The initialization of Program Counter in the PC module.
module test1;
  parameter half_period = 50;
  reg clk;
  initial begin
    #0 clk = 0;
  end
  always #half_period clk = ~clk;
  initial #1600 $stop;
  single_cycle single(AN, CN, clk, reg_num, innerClk);
endmodule

Figure 17  The top-level test module.

The simulation results from Test Cases 1~6 (for pipeline, Case 1~3 for single-cycle) are given below in Figure 18~26.

6.1 Simulation results for the Single-cycle processor

![Simulation results for the Single-cycle processor](image-url)

Figure 18  The simulation results of test case 1. We can see that $s_2 = s_1 = 7$ as expected. And $s_2$ gets its value at the middle of the fifth CC.

![Simulation results for the Single-cycle processor](image-url)

Figure 19  The simulation results of test case 2. $s_0$ changes from 63 to -1, $s_1 = 25$, $s_2 = 88$, $s_3 = 89$ as expected. The active simulation lasts for 5 CC since there are in total 5 instructions.
Figure 20  The simulation results of test case 3. The loop in the MIPS program executes 4 times as expected. $s5 = s4 = 1+2+3+4 = 10.$
6.2 Simulation results for the Single-cycle processor

Figure 21 The simulation result of test case 1 (for pipeline). The results are the same as in single-cycle simulations.

Figure 22 The simulation result of test case 2 (for pipeline). The results are the same as in single-cycle simulations.
Figure 23  The simulation result of test case 3 (for pipeline). The results are the same as in single-cycle simulations. (Process from 1000s~3000s are skipped to save space.)

Figure 24  The simulation result of test case 4 (Pipeline). $s_0 = 17, s_1 = 42, s_2 = 59, s_3 = -25$ and $s_4 = 59$ as expected. The processor is not stalled through the process.
Figure 25  The simulation result of test case 5 (Pipeline). $s0 = 7$, $s1$ changes from 16 to 12 to 7, $s2$ changes from 7 to 19, $s3 = 3$, $s4 = 7$, $s5 = 10$ as expected. Between $s2$ and then $s4$ is set to 7, there are 4 CCs (850~1250). That is because the processor is stalled for 2 CC + the execution of beq 1CC + the penalty for take the branch 1CC (control hazard), which conforms to our design. The load-use hazards of R type and I type also stall the processor at (1350~1450) & (1650~1750) respectively.

Figure 26  The simulation result of test case 6 (Pipeline). $s0 = 7$, $s1 = 14$, $s2 = 3$, $s3 = 4$ as expected. The Pipeline processor is stalled for one CC since there is no control penalty (branch not taken) and EX data hazard for beq hold the processor for one CC.
7 Conclusions

In this project, we modeled a simplified single-cycle processor and a pipeline processor using Verilog. We simulated our modules using Xilinx ISE and verified that our two models did behave as expected. The two processors can perform basic R, I, J-type MIPS instructions correctly and the pipeline processor is able to properly handle data and control hazards.