FIGURE e4.14.1 A Verilog behavorial model for the MIPS five-stage pipeline, ignoring branch and data hazards. As in the design earlier in Chapter 4, we use separate instruction and data memories, which would be implemented using separate caches as we describe in Chapter 5.

FIGURE e4.14.2 A behavioral definition of the five-stage MIPS pipeline with bypassing to ALU operations and address calculations. The code added to Figure e4.14.1 to handle bypassing is highlighted. Because these bypasses only require changing where the ALU inputs come from, the only changes required are in the combinational logic responsible for selecting the ALU inputs.

FIGURE e4.14.3 A behavioral definition of the five-stage MIPS pipeline with stalls for loads when the destination is an ALU instruction or effective address calculation. The changes from Figure e4.14.2 are highlighted.

FIGURE e4.14.4 A behavioral definition of the five-stage MIPS pipeline with stalls for loads when the destination is an ALU instruction or effective address calculation. The changes from Figure e4.14.3 are highlighted.

FIGURE e4.14.5 A behavioral specification of the multicycle MIPS design. This has the same cycle behavior as the multicycle design, but is purely for simulation and specification. It cannot be used for synthesis.

FIGURE e4.14.6 A Verilog version of the multicycle MIPS datapath that is appropriate for synthesis. This datapath relies on several units from Appendix B. Initial statements do not synthesize, and a version used for synthesis would have to incorporate a reset signal that had this effect. Also note that resetting R0 to 0 on every clock is not the best way to ensure that R0 stays 0; instead, modifying the register file module to produce 0 whenever R0 is read and to ignore writes to R0 would be a more efficient solution.

FIGURE e4.14.7 The MIPS CPU using the datapath from Figure e4.14.6.

FIGURE e4.14.8  Single-cycle pipeline diagrams for clock cycles 1 (top diagram) and 2 (bottom diagram). This style of pipeline representation is a snapshot of every instruction executing during one clock cycle. Our example has but two instructions, so at most two stages are identified in each clock cycle; normally, all five stages are occupied. The highlighted portions of the datapath are active in that clock cycle. The load is fetched in clock cycle 1 and decoded in clock cycle 2, with the subtract fetched in the second clock cycle. To make the figures easier to understand, the other pipeline stages are empty, but normally there is an instruction in every pipeline stage.

FIGURE e4.14.9  Single-cycle pipeline diagrams for clock cycles 3 (top diagram) and 4 (bottom diagram). In the third clock cycle in the top diagram, lw enters the EX stage. At the same time, sub enters ID. In the fourth clock cycle (bottom datapath), lw moves into MEM stage, reading memory using the address found in EX/MEM at the beginning of clock cycle 4. At the same time, the ALU subtracts and then places the difference into EX/MEM at the end of the clock cycle.

FIGURE e4.14.10 Single-cycle pipeline diagrams for clock cycles 5 (top diagram) and 6 (bottom diagram). In clock cycle 5, lw completes by writing the data in MEM/WB into register 10, and sub sends the difference in EX/MEM to MEM/WB. In the next clock cycle, sub writes the value in MEM/WB to register 11.

FIGURE e4.14.11 Clock cycles 1 and 2. The phrase “*before <i>*” means the ith instruction before lw. The lw instruction in the top datapath is in the IF stage. At the end of the clock cycle, the lw instruction is in the IF/ID pipeline registers. In the second clock cycle, seen in the bottom datapath, the lw moves to the ID stage, and sub enters in the IF stage. Note that the values of the instruction fields and the selected source registers are shown in the ID stage. Hence register $1 and the constant 20, the operands of lw, are written into the ID/EX pipeline register. The number 10, representing the destination register number of lw, is also placed in ID/EX. Bits 15–11 are 0, but we use X to show that a field plays no role in a given instruction. The top of the ID/EX pipeline register shows the control values for lw to be used in the remaining stages. These control values can be read from the lw row of the table in Figure 4.18.

FIGURE e4.14.12  Clock cycles 3 and 4. In the top diagram, lw enters the EX stage in the third clock cycle, adding $1 and 20 to form the address in the EX/MEM pipeline register. (The lw instruction is written lw $10,. . . upon reaching EX, because the identity of instruction operands is not needed by EX or the subsequent stages. In this version of the pipeline, the actions of EX, MEM, and WB depend only on the instruction and its destination register or its target address.) At the same time, sub enters ID, reading registers $2 and $3, and the and instruction starts IF. In the fourth clock cycle (bottom datapath), lw moves into MEM stage, reading memory using the value in EX/MEM as the address. In the same clock cycle, the ALU subtracts $3 from $2 and places the difference into EX/MEM, reads registers $4 and $5 during ID, and the or instruction enters IF. The two diagrams show the control signals being created in the ID stage and peeled off as they are used in subsequent pipe stages.

FIGURE e4.14.13 Clock cycles 5 and 6. With add, the final instruction in this example, entering IF in the top datapath, all instructions are engaged. By writing the data in MEM/WB into register 10, lw completes; both the data and the register number are in MEM/WB. In the same clock cycle, sub sends the difference in EX/MEM to MEM/WB, and the rest of the instructions move forward. In the next clock cycle, sub selects the value in MEM/WB to write to register number 11, again found in MEM/WB. The remaining instructions play follow-the-leader: the ALU calculates the OR of $6 and $7 for the or instruction in the EX stage, and registers $8 and $9 are read in the ID stage for the add instruction. The instructions after add are shown as inactive just to emphasize what occurs for the five instructions in the example. The phrase “after,i.” means the ith instruction after add.

FIGURE e4.14.14  Clock cycles 7 and 8. In the top datapath, the add instruction brings up the rear, adding the values corresponding to registers $8 and $9 during the EX stage. The result of the or instruction is passed from EX/MEM to MEM/WB in the MEM stage, and the WB stage writes the result of the and instruction in MEM/WB to register $12. Note that the control signals are deasserted (set to 0) in the ID stage, since no instruction is being executed. In the following clock cycle (lower drawing), the WB stage writes the result to register $13, thereby completing or, and the MEM stage passes the sum from the add in EX/MEM to MEM/WB. The instructions after add are shown as inactive for pedagogical reasons.

FIGURE e4.14.15  Clock cycle 9. The WB stage writes the sum in MEM/WB into register $14, completing add and the five-instruction sequence. The instructions after add are shown as inactive for pedagogical reasons.

FIGURE e4.14.16 Clock cycles 3 and 4 of the instruction sequence on page 4.13-26. The bold lines are those active in a clock cycle, and the italicized register numbers in color indicate a hazard. The forwarding unit is highlighted by shading it when it is forwarding data to the ALU. The instructions before sub are shown as inactive just to emphasize what occurs for the four instructions in the example. Operand names are used in EX for control of forwarding; thus they are included in the instruction label for EX. Operand names are not needed in MEM or WB, so . . . is used. Compare this with Figures e4.14.12 through e4.14.15, which show the datapath without forwarding where ID is the last stage to need operand information.

FIGURE e4.14.17 Clock cycles 5 and 6 of the instruction sequence on page 4.13-26. The forwarding unit is highlighted when it is forwarding data to the ALU. The two instructions after add are shown as inactive just to emphasize what occurs for the four instructions in the example. The bold lines are those active in a clock cycle, and the italicized register numbers in color indicate a hazard.

FIGURE e4.14.18  Clock cycles 2 and 3 of the instruction sequence on page 4.13-26 with a load replacing sub. The bold lines are those active in a clock cycle, the italicized register numbers in color indicate a hazard, and the . . . in the place of operands means that their identity is information not needed by that stage. The values of the significant control lines, registers, and register numbers are labeled in the figures. The and instruction wants to read the value created by the lw instruction in clock cycle 3, so the hazard detection unit stalls the and and or instructions. Hence, the hazard detection unit is highlighted.

FIGURE e4.14.19  Clock cycles 4 and 5 of the instruction sequence on page 4.13-26 with a load replacing sub. The bubble is inserted in the pipeline in clock cycle 4, and then the and instruction is allowed to proceed in clock cycle 5. The forwarding unit is highlighted in clock cycle 5 because it is forwarding data from lw to the ALU. Note that in clock cycle 4, the forwarding unit forwards the address of the lw as if it were the contents of register $2; this is rendered harmless by the insertion of the bubble. The bold lines are those active in a clock cycle, and the italicized register numbers in color indicate a hazard.

FIGURE e4.14.20 Clock cycles 6 and 7 of the instruction sequence on page 4.13-26 with a load replacing sub. Note that unlike in Figure e4.14.17, the stall allows the lw to complete, and so there is no forwarding from MEM/WB in clock cycle 6. Register $4 for the add in the EX stage still depends on the result from or in EX/MEM, so the forwarding unit passes the result to the ALU. The bold lines show ALU input lines active in a clock cycle, and the italicized register numbers indicate a hazard. The instructions after add are shown as inactive for pedagogical reasons.