Figure e4.5.1 The high-level view of the multicycle datapath. This picture shows the key elements of the datapath: a shared memory unit, a single ALU shared among instructions, and the connections among these shared units. The use of shared functional units requires the addition or widening of multiplexors as well as new temporary registers that hold data between clock cycles of the same instruction. The additional registers are the Instruction register (IR), the Memory data register (MDR), A, B, and ALUOut.

Figure e4.5.2 Multicycle datapath for MIPS handles the basic instructions. Although this datapath supports normal incrementing of the PC, a few more connections and a multiplexor will be needed for branches and jumps; we will add these shortly. The additions versus the single-clock datapath include several registers (IR, MDR, A, B, ALUOut), a multiplexor for the memory address, a multiplexor for the top ALU input, and expanding the multiplexor on the bottom ALU input into a four-way selector. These small additions allow us to remove two adders and a memory unit.

Figure e4.5.3 The multicycle datapath from Figure e4.5.2 with the control lines shown. The signals ALUOp and ALUSrcB are 2-bit control signals, while all the other control lines are 1-bit signals. Neither register A nor B requires a write signal, since their contents are only read on the cycle immediately after it is written. The memory data register has been added to hold the data from a load when the data returns from memory. Data from a load returning from memory cannot be written directly into the register file since the clock cycle cannot accommodate the time required for both the memory access and the register file write. The MemRead signal has been moved to the top of the memory unit to simplify the figures. The full set of datapaths and control lines for branches will be added shortly.

Figure e4.5.4 The complete datapath for the multicycle implementation together with the necessary control lines. The control lines of Figure e4.5.3 are attached to the control unit, and the control and datapath elements needed to effect changes to the PC are included. The major additions from Figure e4.5.3 include the multiplexor used to select the source of a new PC value; gates used to combine the PC write signals; and the control signals PCSource, PCWrite, and PCWriteCond. The PCWriteCond signal is used to decide whether a conditional branch should be taken. Support for jumps is included.

Figure e4.5.5 The action caused by the setting of each control signal in Figure e4.5.4. The top table describes the 1-bit control signals, while the bottom table describes the 2-bit signals. Only those control lines that affect multiplexors have an action when they are deasserted. This information is similar to that in Figure 4.16 on page 264 for the single-cycle datapath, but adds several new control lines (IRWrite, PCWrite, PCWriteCond, ALUSrcB, and PCSource) and removes control lines that are no longer used or have been replaced (PCSrc, Branch, and Jump).

Figure e4.5.6 Summary of the steps taken to execute any instruction class. Instructions take from three to five execution steps. The first two steps are independent of the instruction class. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction class. The empty entries for the Memory access step or the Memory read completion step indicate that the particular instruction class takes fewer cycles. In a multicycle implementation, a new instruction will be started as soon as the current instruction completes, so these cycles are not idle or wasted. As mentioned earlier, the register file actually reads every cycle, but as long as the IR does not change, the values read from the register file are identical. In particular, the value read into register B during the Instruction decode stage, for a branch or R-type instruction, is the same as the value stored into B during the Execution stage and then used in the Memory access stage for a store word instruction.

Figure e4.5.7 The high-level view of the finite-state machine control. The first steps are independent of the instruction class; then a series of sequences that depend on the instruction opcode are used to complete each instruction class. After completing the actions needed for that instruction class, the control returns to fetch a new instruction. Each box in this figure may represent one to several states. The arc labeled Start marks the state in which to begin when the first instruction is to be fetched.

Figure e4.5.8 The instruction fetch and decode portion of every instruction is identical. These states correspond to the top box in the abstract finite-state machine in Figure e4.5.7. In the first state we assert two signals to cause the memory to read an instruction and write it into the Instruction register (MemRead and IRWrite), and we set IorD to 0 to choose the PC as the address source. The signals ALUSrcA, ALUSrcB, ALUOp, PCWrite, and PCSource are set to compute PC  4 and store it into the PC. (It will also be stored into ALUOut, but never used from there.) In the next state, we compute the branch target address by setting ALUSrcB to 11 (causing the shifted and sign-extended lower 16 bits of the IR to be sent to the ALU), setting ALUSrcA to 0 and ALUOp to 00; we store the result in the ALUOut register, which is written on every cycle. There are four next states that depend on the class of the instruction, which is known during this state. The control unit input, called Op, is used to determine which of these arcs to follow. Remember that all signals not explicitly asserted are deasserted; this is particularly important for signals that control writes. For multiplexor controls, lack of a specific setting indicates that we do not care about the setting of the multiplexor.

Figure e4.5.9 The finite-state machine for controlling memory reference instructions has four states. These states correspond to the box labeled “Memory access instructions” in Figure e4.5.7. After performing a memory address calculation, a separate sequence is needed for load and for store. The setting of the control signals ALUSrcA, ALUSrcB, and ALUOp is used to cause the memory address computation in state 2. Loads require an extra state to write the result from the MDR (where the result is written in state 3) into the register file.

Figure e4.5.10 R-type instructions can be implemented with a simple two-state finite- state machine. These states correspond to the box labeled “R-type instructions” in Figure e4.5.7. The first state causes the ALU operation to occur, while the second state causes the ALU result (which is in ALUOut) to be written in the register file. The three signals asserted during state 7 cause the contents of ALUOut to be written into the register file in the entry specified by the rd field of the Instruction register.

Figure e4.5.11 The branch instruction requires a single state. The first three outputs that are asserted cause the ALU to compare the registers (ALUSrcA, ALUSrcB, and ALUOp), while the signals PCSource and PCWriteCond perform the conditional write if the branch condition is true. Notice that we do not use the value written into ALUOut; instead, we use only the Zero output of the ALU. The branch target address is read from ALUOut, where it was saved at the end of state 1.

Figure e4.5.12 The jump instruction requires a single state. that asserts two control signals to write the PC with the lower 26 bits of the Instruction register shifted left 2 bits and concatenated to the upper 4 bits of the PC of this instruction.

Figure e4.5.13 The complete finite-state machine control for the datapath shown in Figure e4.5.4. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output signals asserted during that state; we always specify the setting of a multiplexor control signal if the correct operation requires it. Hence, in some states a multiplexor control will be set to 0.

Figure e4.5.14 Finite-state machine controllers are typically implemented using a block of combinational logic and a register to hold the current state. The outputs of the combinational logic are the next-state number and the control signals to be asserted for the current state. The inputs to the combinational logic are the current state and any inputs used to determine the next state. In this case, the inputs are the instruction register opcode bits. Notice that in the finite-state machine used in this chapter, the outputs depend only on the current state, not on the inputs. The elaboration below explains this in more detail.