```
  // Creates a RISC-V register file
    // Inputs are the rs1 and rs2 fields of the IR used to specify which
registers to read,
    // Writereg (the write register number), Writedata (the data to be
written),
    // RegWrite (indicates a write), the clock
    // Outputs are A and B, the registers read
    registerfile regs (IR[19:15], IR[24:20], IR[11:7], Writedata,
RegWrite, A, B, clock); // Register file

  // The clock-triggered actions of the datapath
  always @(posedge clock)
  begin
    if (MemWrite) Memory[ALUOut >> 2] <= B; // Write memory--must be a
store
    ALUOut <= ALUResultOut; // Save the ALU result for use on a later
clock cycle
    if (IRWrite) IR <= MemOut; // Write the IR if an instruction fetch
    MDR <= MemOut; // Always save the memory read value
    // The PC is written both conditionally (controlled by PCWrite) and
unconditionally

  end
endmodule
```

FIGURE e4.14.6 A Verilog version of the multicycle RISC-V datapath that is appropriate for synthesis. (*Continued*)

No Hazard Illustrations

On page 285, we gave the example code sequence:

```
lw    x10, 40(x1)
sub   x11, x2, x3
add   x12, x3, x4
lw    x13, 48(x1)
add   x14, x5, x6
```

Figures 4.59 and 4.60 showed the multiple-clock-cycle pipeline diagrams for this two-instruction sequence executing across six clock cycles. Figures e4.14.8 through e4.14.10 show the corresponding single-clock-cycle pipeline diagrams for these two instructions. Note that the order of the instructions differs between these two types of diagrams: the newest instruction is at the *bottom and to the right* of the multiple-clock-cycle pipeline diagram, and it is on the *left* in the single-clock-cycle pipeline diagram.

More Examples

To understand how pipeline control works, let's consider these five instructions going through the pipeline:

```
lw     x10, 40(x1)
sub    x11, x2, x3
and    x12, x4, x5
or     x13, x6, x7
add    x14, x8, x9
```

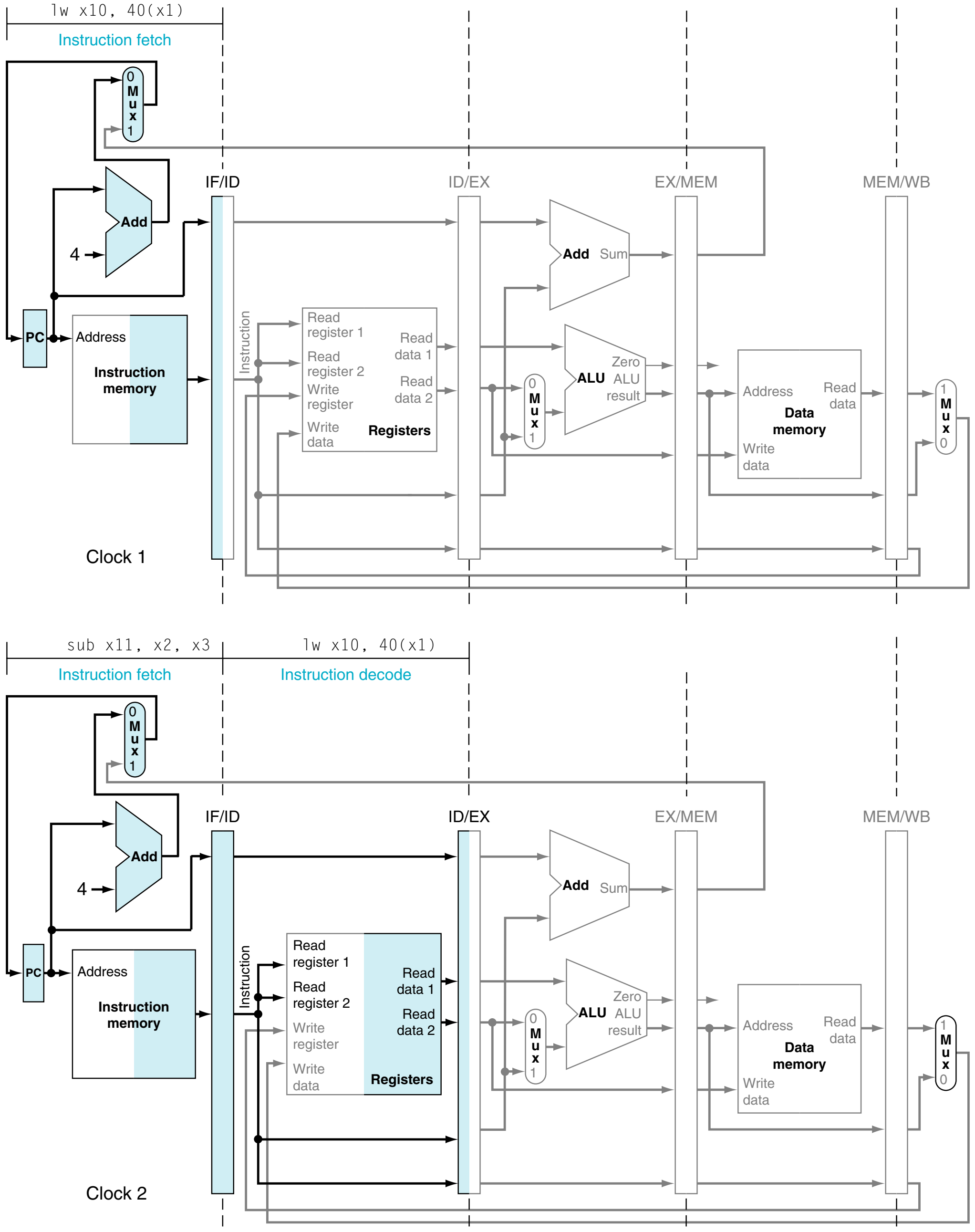


FIGURE e4.14.8 Single-cycle pipeline diagrams for clock cycles 1 (top diagram) and 2 (bottom diagram). This style of pipeline representation is a snapshot of every instruction executing during one clock cycle. Our example has but two instructions, so at most two stages are identified in each clock cycle; 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.

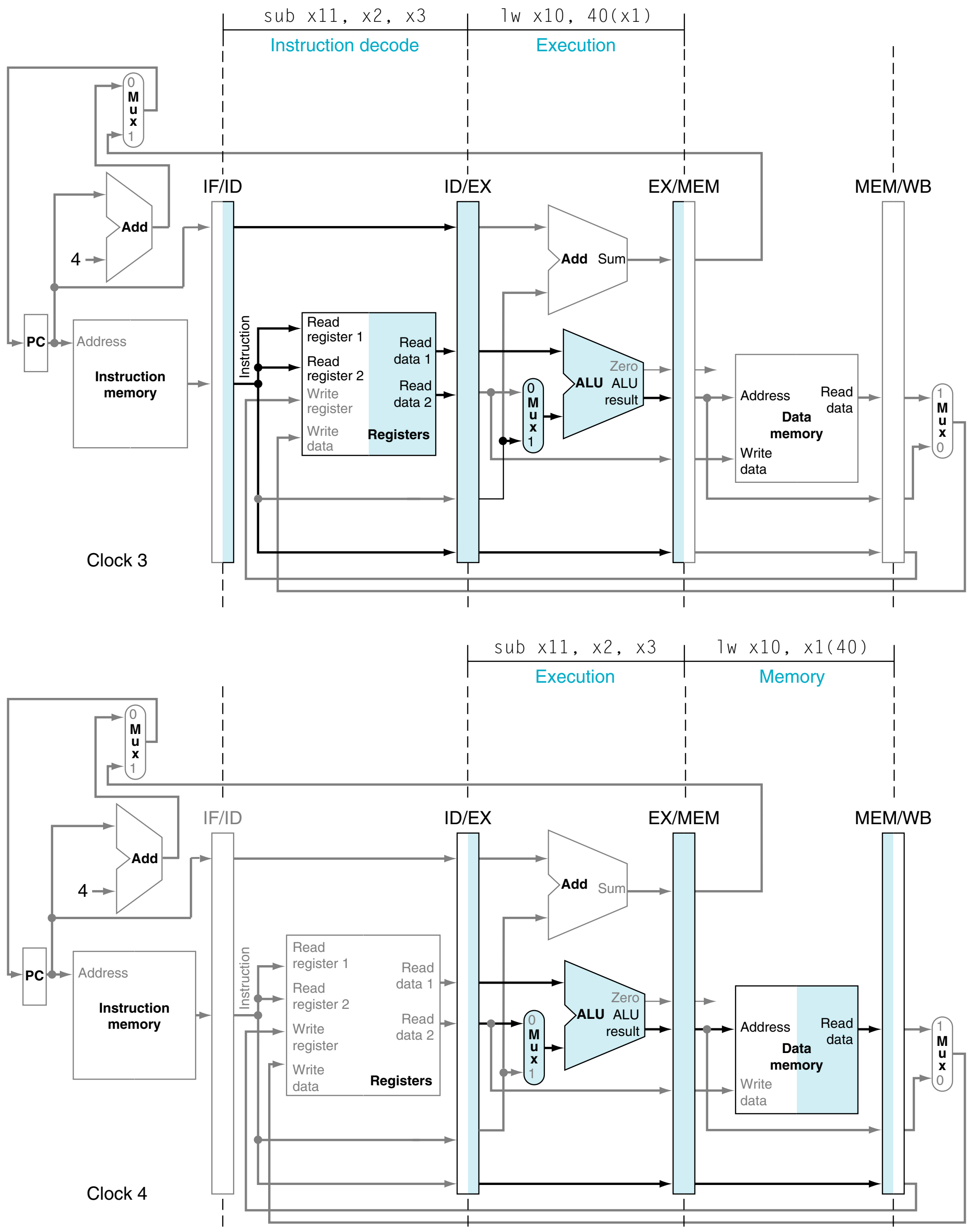
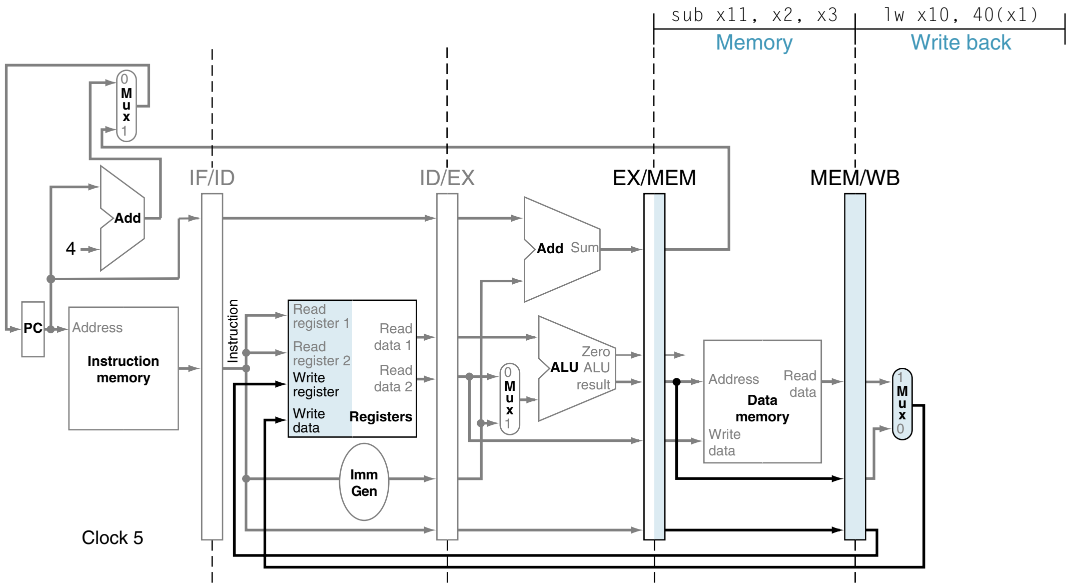


FIGURE e4.14.9 Single-cycle pipeline diagrams for clock cycles 3 (top diagram) and 4 (bottom diagram). In the third clock cycle in the top diagram, `lw` enters the EX stage. At the same time, `sub` enters ID. In the fourth clock cycle (bottom datapath), `lw` moves into MEM stage, reading memory using 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.



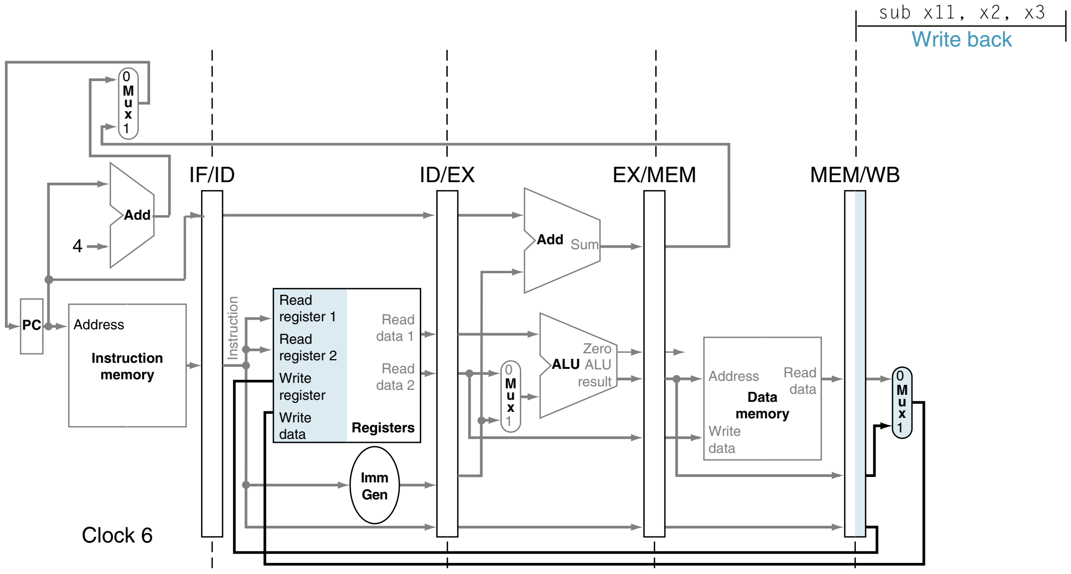


FIGURE e4.14.10 Single-cycle pipeline diagrams for clock cycles 5 (top diagram) and 6 (bottom diagram). In clock cycle 5, `lw` completes by writing the data in MEM/WB into register 10, and `sub` sends the difference in EX/MEM to MEM/WB. In the next clock cycle, `sub` writes the value in MEM/WB to register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

Figures e4.14.11 through e4.14.15 show these instructions proceeding through the nine clock cycles it takes them to complete execution, highlighting what is active in a stage and identifying the instruction associated with each stage during a clock cycle. If you examine them carefully, you may notice:

- In Figure e4.14.13 you can see the sequence of the destination register numbers from left to right at the bottom of the pipeline registers. The numbers advance to the right during each clock cycle, with the MEM/WB pipeline register supplying the number of the register written during the WB stage.
- When a stage is inactive, the values of control lines that are deasserted are shown as 0 or X (for don't care).
- Sequencing of control is embedded in the pipeline structure itself. First, all instructions take the same number of clock cycles, so there is no special control for instruction duration. Second, all control information is computed during instruction decode and then passed along by the pipeline registers.

Forwarding Illustrations

We can use the single-clock-cycle pipeline diagrams to show how forwarding operates, as well as how the control activates the forwarding paths. Consider the following code sequence in which the dependences have been highlighted:

```
sub  x2, x1, x3
and  x4, x2, x5
or   x4, x4, x2
add  x9, x4, x2
```

Figures e4.14.16 and e4.14.17 show the events in clock cycles 3–6 in the execution of these instructions.

Thus, in clock cycle 5, the forwarding unit selects the EX/MEM pipeline register for the upper input to the ALU and the MEM/WB pipeline register for the lower input to the ALU. The following `add` instruction reads both register `x4`, the target of the `and` instruction, and register `x2`, which the `sub` instruction has already written. Notice that the prior two instructions both write register `x4`, so the forwarding unit must pick the immediately preceding one (MEM stage).

In clock cycle 6, the forwarding unit thus selects the EX/MEM pipeline register, containing the result of the `or` instruction, for the upper ALU input but uses the non-forwarding register value for the lower input to the ALU.

Illustrating Pipelines with Stalls and Forwarding

We can use the single-clock-cycle pipeline diagrams to show how the control for stalls works. Figures e4.14.18 through e4.14.20 show the single-cycle diagram for clocks 2 through 7 for the following code sequence (dependences highlighted):

```
lw    x2, 40(x1)
and   x4, x2, x5
or    x4, x4, x2
add   x9, x4, x2
```

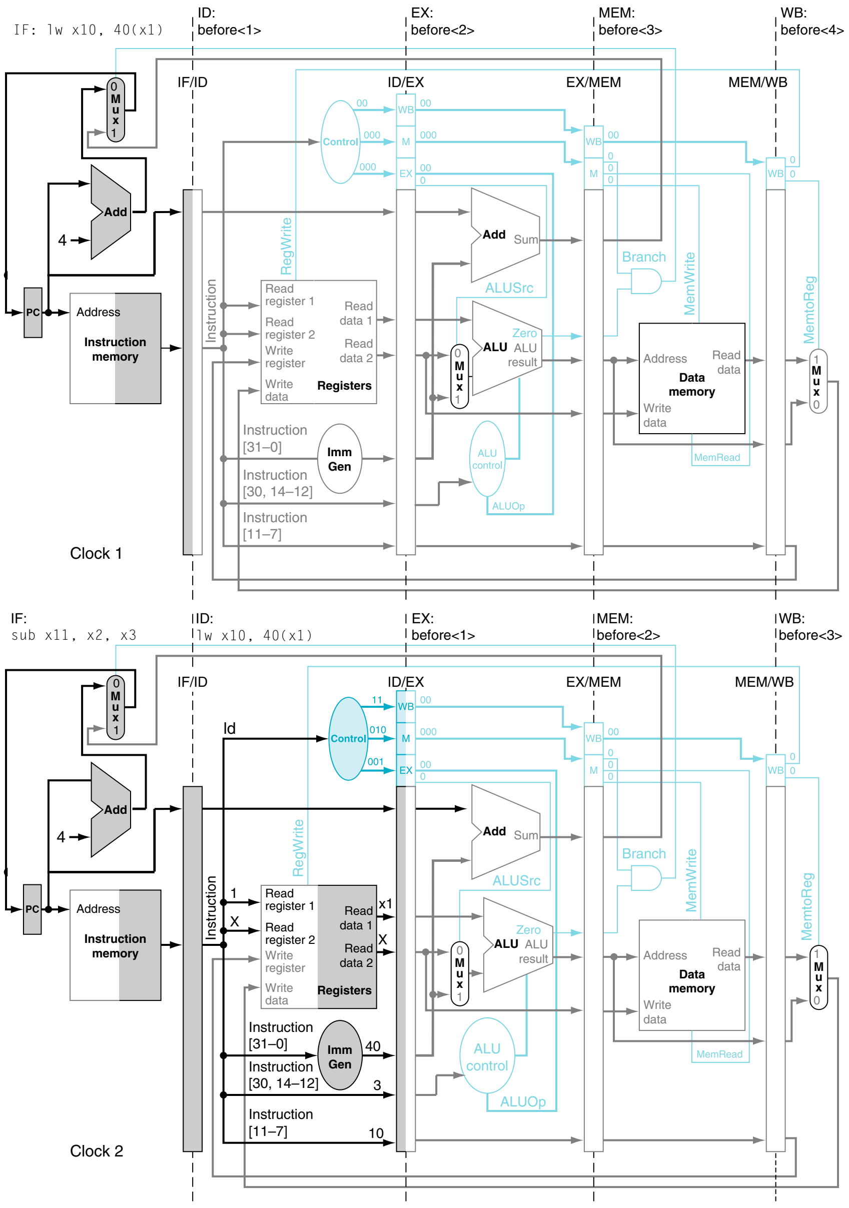


FIGURE e4.14.11 Clock cycles 1 and 2. The phrase "*before <i>*" means the *i*th instruction before `lw`. The `lw` instruction in the top datapath is in the IF stage. At the end of the clock cycle, the `lw` instruction is in the IF/ID pipeline registers. In the second clock cycle, seen in the bottom datapath, 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 `x1` and the constant 40, 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. The top of the ID/EX pipeline register shows the control values for `ld` to be used in the remaining stages. These control values can be read from the `lw` row of the table in Figure 4.22.

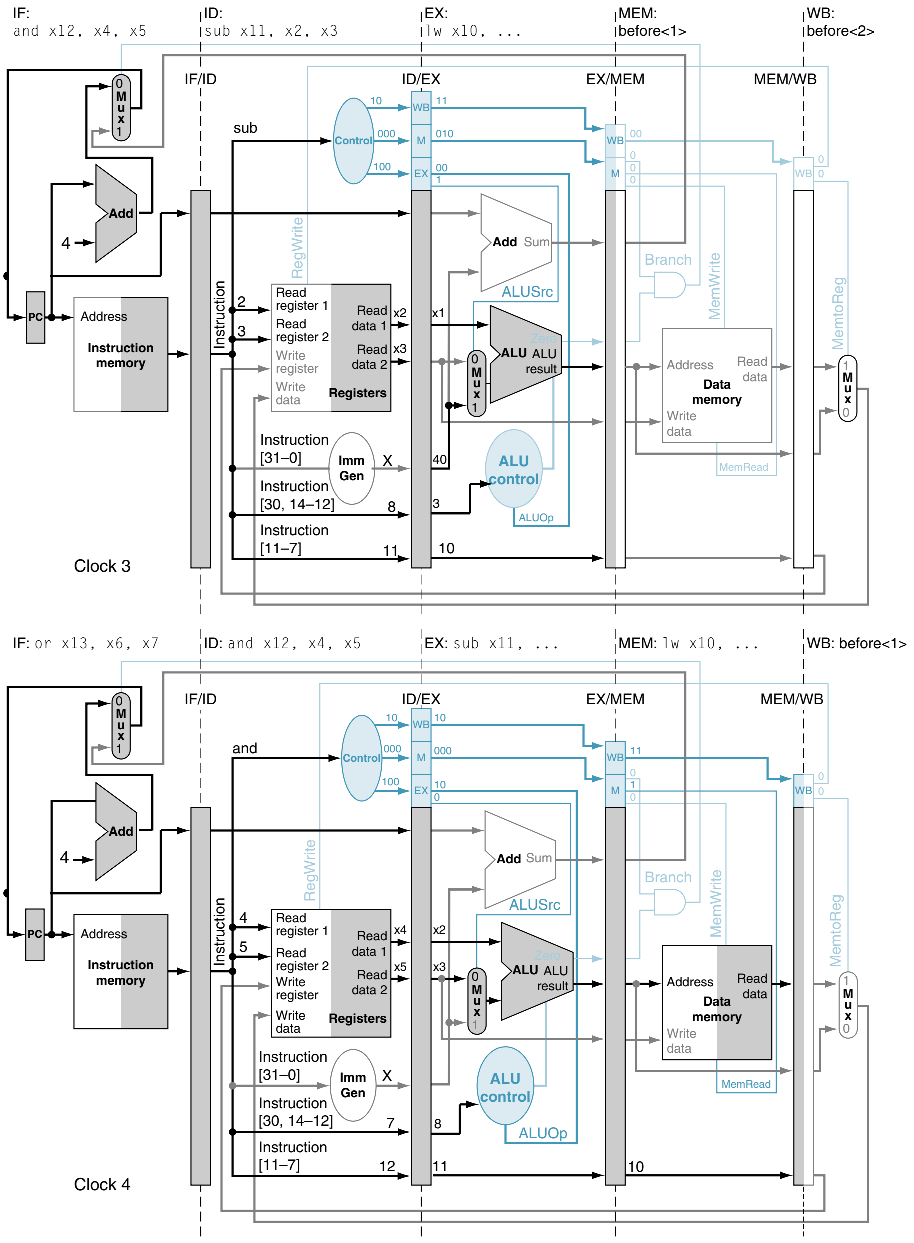


FIGURE e4.14.12 Clock cycles 3 and 4. In the top diagram, `lw` enters the EX stage in the third clock cycle, adding `x1` and 40 to form the address in the EX/MEM pipeline register. (The `lw` instruction is written `lw x10, …` upon reaching EX, because the identity of instruction operands is not needed 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 `x2` and `x3`, 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 `x3` from `x2` and places the difference into EX/MEM, reads registers `x4` and `x5` 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.

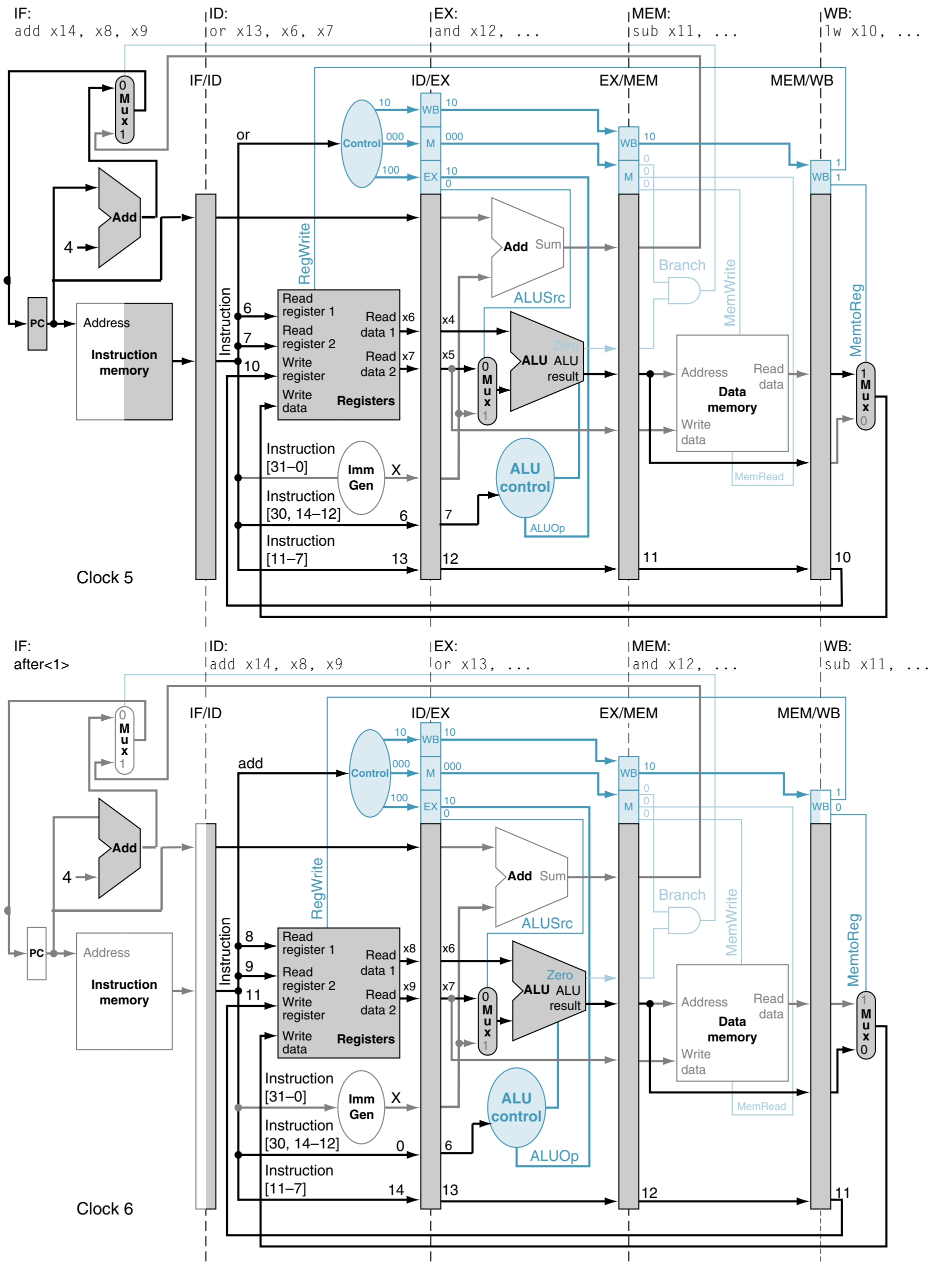


FIGURE e4.14.13 Clock cycles 5 and 6. With `add`, the final instruction in this example, entering IF in the top datapath, all instructions are engaged. By writing the data in MEM/WB into register 10, `lw` completes; both the data and the register number are in MEM/WB. In the same clock cycle, `sub` sends 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 `x6` and `x7` for the `or` instruction in the EX stage, and registers `x8` and `x9` 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 *i*th instruction after `add`.

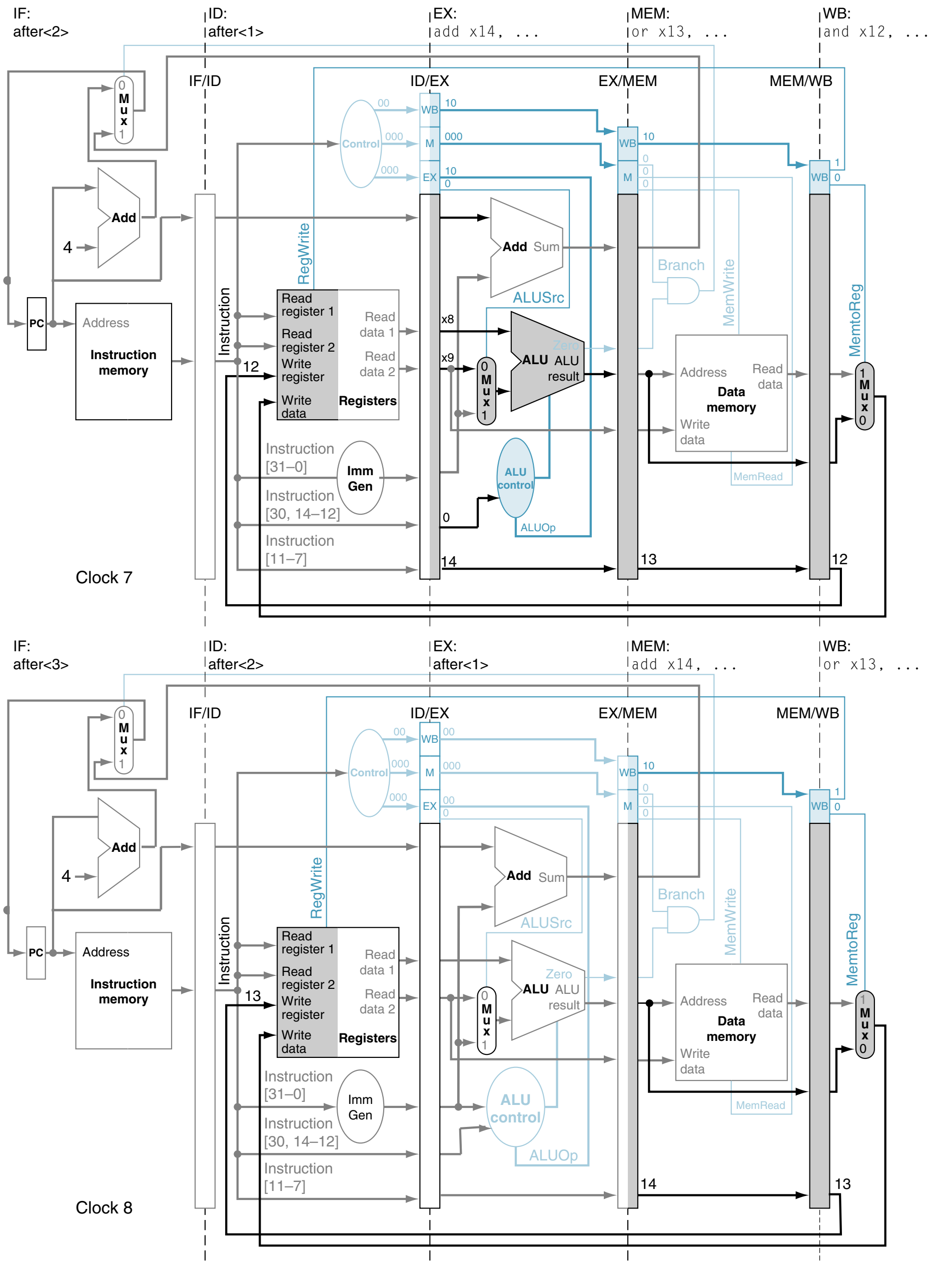


FIGURE e4.14.14 Clock cycles 7 and 8. In the top datapath, the add instruction brings up the rear, adding the values corresponding to registers x8 and x9 during the EX stage. The result of the or instruction is passed from EX/MEM to MEM/WB in the MEM stage, and the WB stage writes the result of the and instruction in MEM/WB to register x12. 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 x13, 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.

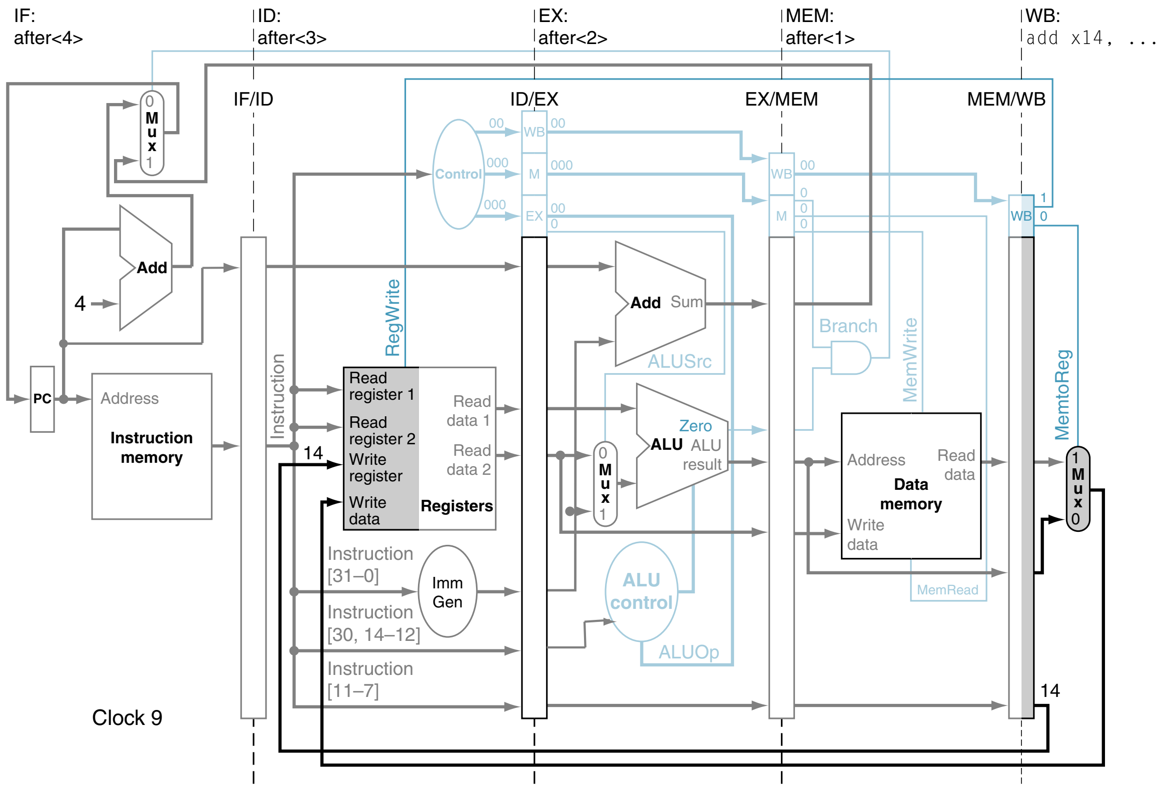


FIGURE e4.14.15 Clock cycle 9. The WB stage writes the ALU result in MEM/WB into register `x14`, completing `add` and the five-instruction sequence. The instructions after `add` are shown as inactive for pedagogical reasons.

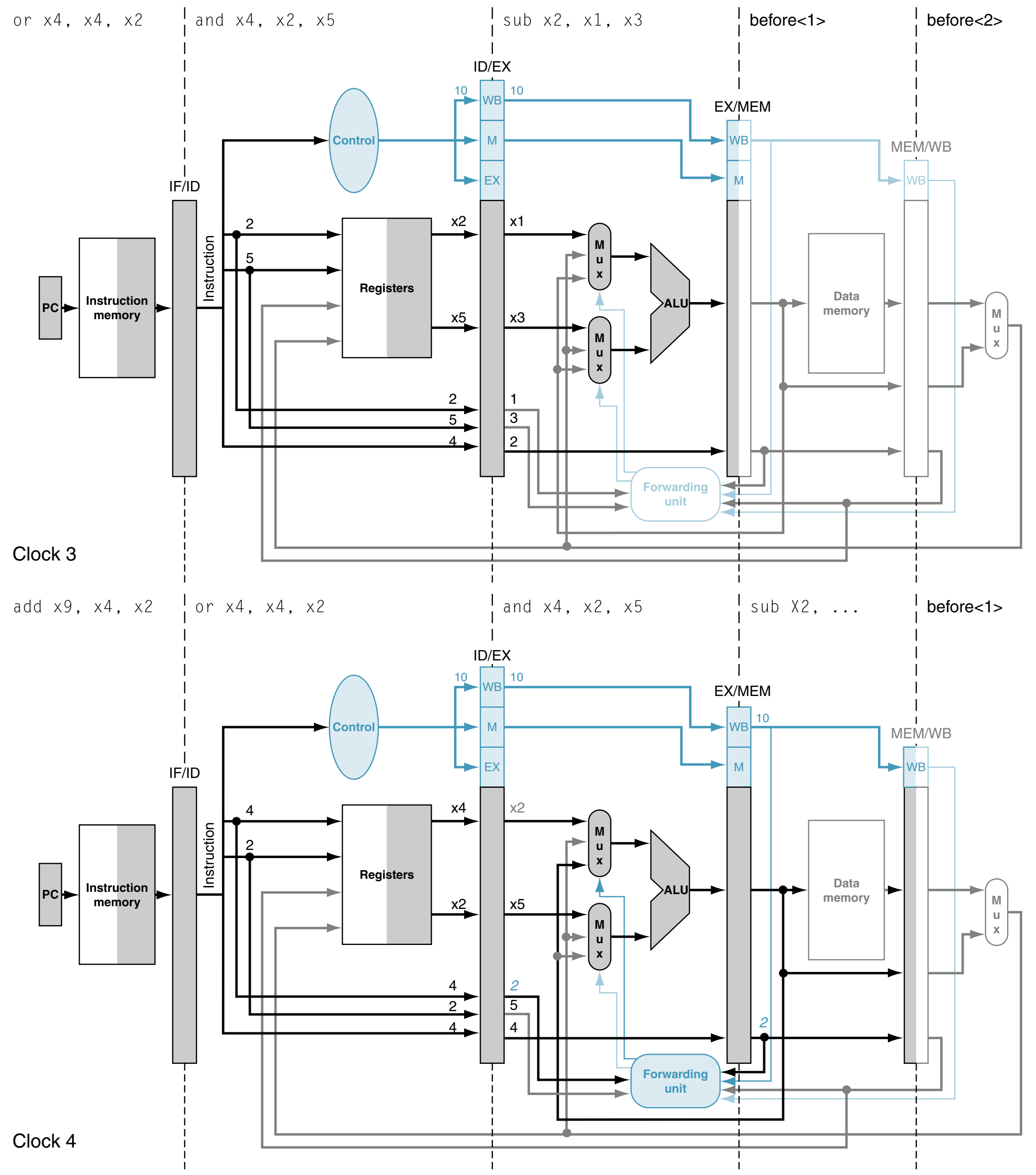


FIGURE e4.14.16 Clock cycles 3 and 4 of the instruction sequence on page 366.e26. The bold lines are those active in a clock cycle, and the italicized register numbers in color indicate a hazard. The forwarding unit is highlighted by shading it when it is forwarding data to the ALU. The instructions 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.

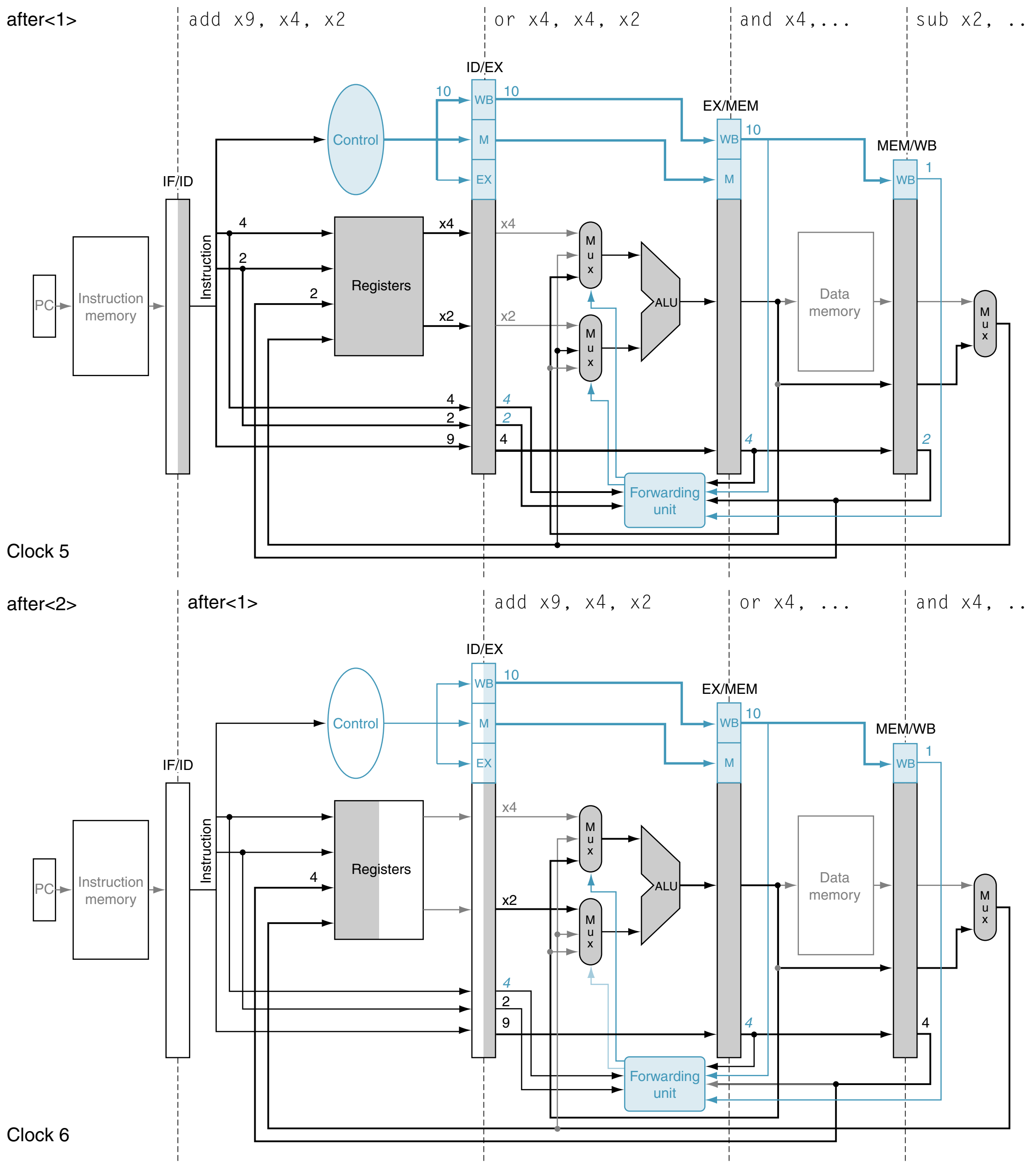


FIGURE e4.14.17 Clock cycles 5 and 6 of the instruction sequence on page 366.e26. The forwarding unit is highlighted when it is forwarding data to the ALU. The two instructions after `add` are shown as inactive just to emphasize what occurs for the four instructions in the example. The bold lines are those active in a clock cycle, and the italicized register numbers in color indicate a hazard.

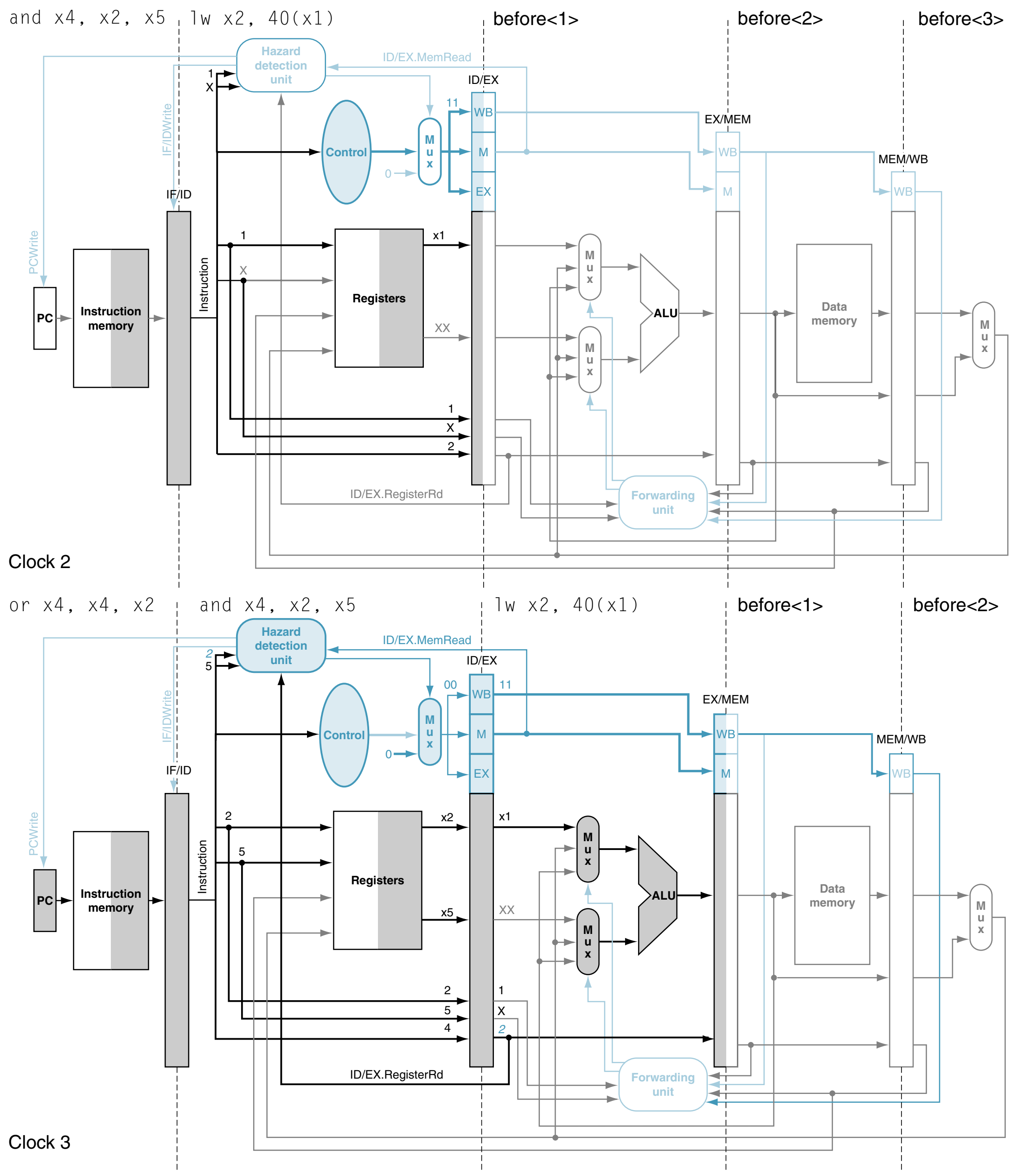


FIGURE e4.14.18 Clock cycles 2 and 3 of the instruction sequence on page 366.e26 with a load replacing `sub`**.** The bold lines are those active in a clock cycle, the italicized register numbers in color indicate a hazard, and the … in the place of operands means that their identity is information not 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.

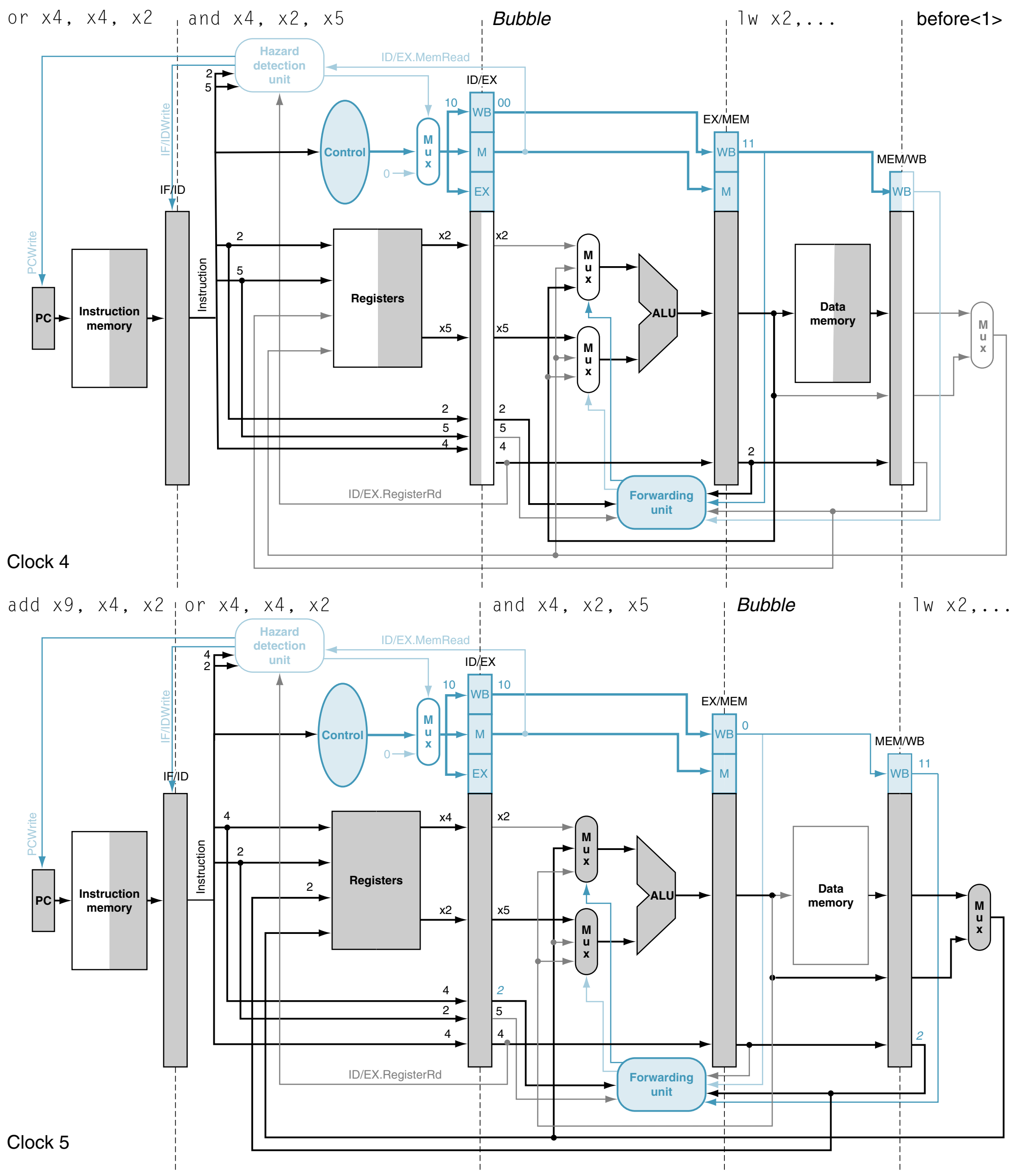


FIGURE e4.14.19 Clock cycles 4 and 5 of the instruction sequence on page 366.e26 with a load replacing `sub`**.** The bubble is inserted in the pipeline in clock cycle 4, and then the `and` instruction is allowed to proceed in clock cycle 5. The forwarding unit is highlighted in clock cycle 5 because 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 `x2`; 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.

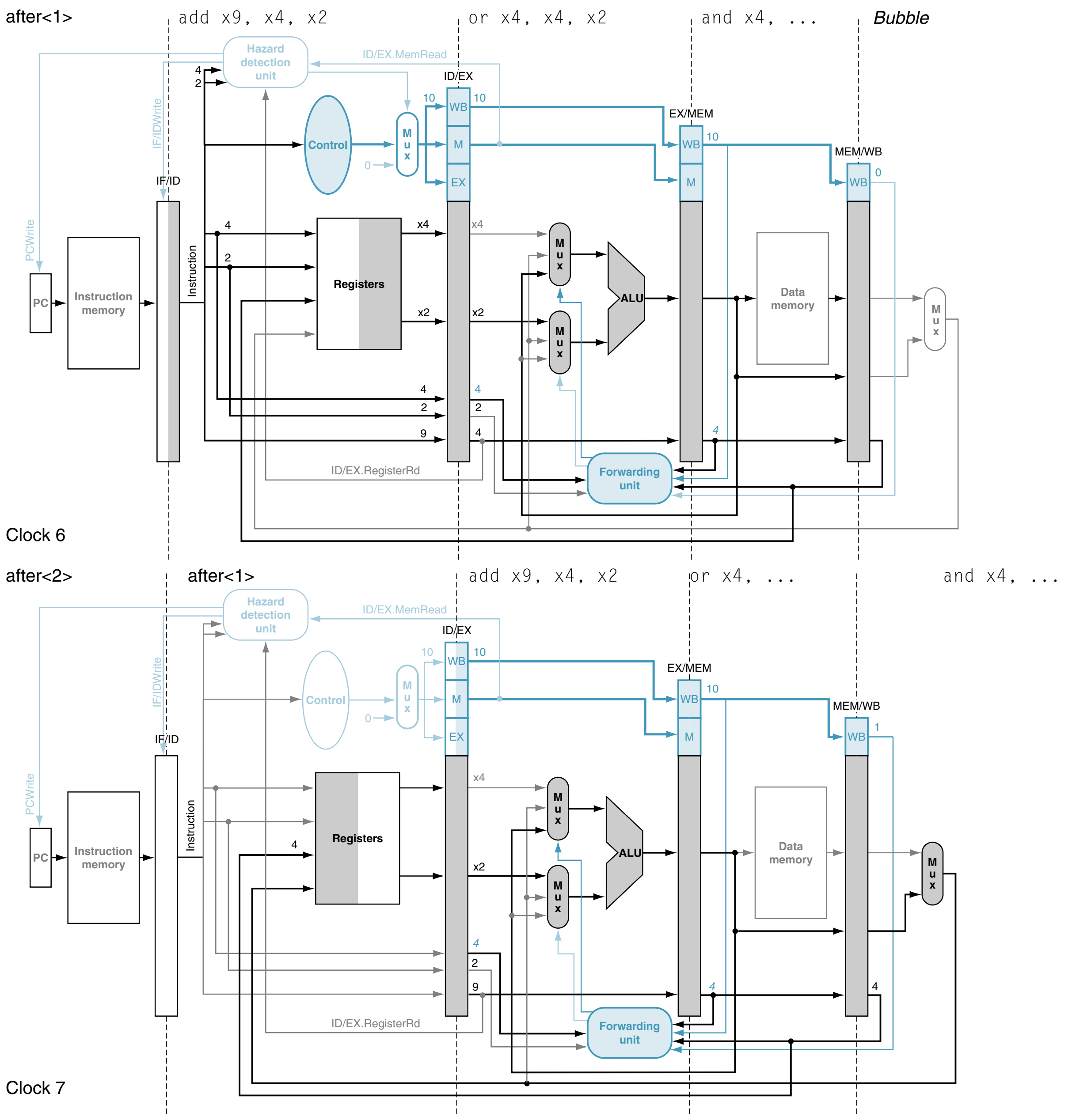


FIGURE e4.14.20 Clock cycles 6 and 7 of the instruction sequence on page 366.e26 with a load replacing `sub`**.** Note that unlike in Figure e4.14.17, the stall allows the `lw` to complete, and so there is no forwarding from MEM/WB in clock cycle 6. Register `x4` for the `add` in the EX stage still depends 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.