

Labeling Pipeline Diagrams with Control

6.15 [20] <\$6.3> To understand how pipeline control works, let's consider these five instructions going through the pipeline:

```
lw      $t0, 20($t1)
sub     $t1, $t2, $t3
and     $t2, $t4, $t5
or      $t3, $t6, $t7
add     $t4, $t8, $t9
```

Show the instructions in the pipeline that precede the `lw` as before <1>, before <2>, ..., and the instructions after the `add` as after <1>, after <2>, ... Figures 6.14.5 through 6.14.9 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. Reviewing these figures carefully will give you insight into how pipelines work. A few items you may notice:

- In Figure 6.14.7 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).
- In contrast to Chapter 5, where sequencing of control required special hardware, 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.

Using the same format as Figure 6.14.5, and starting with the blank pipelining diagram in Figure 6.14.10, draw the pipeline diagrams for the above sequence for a total of 4 clock cycles.

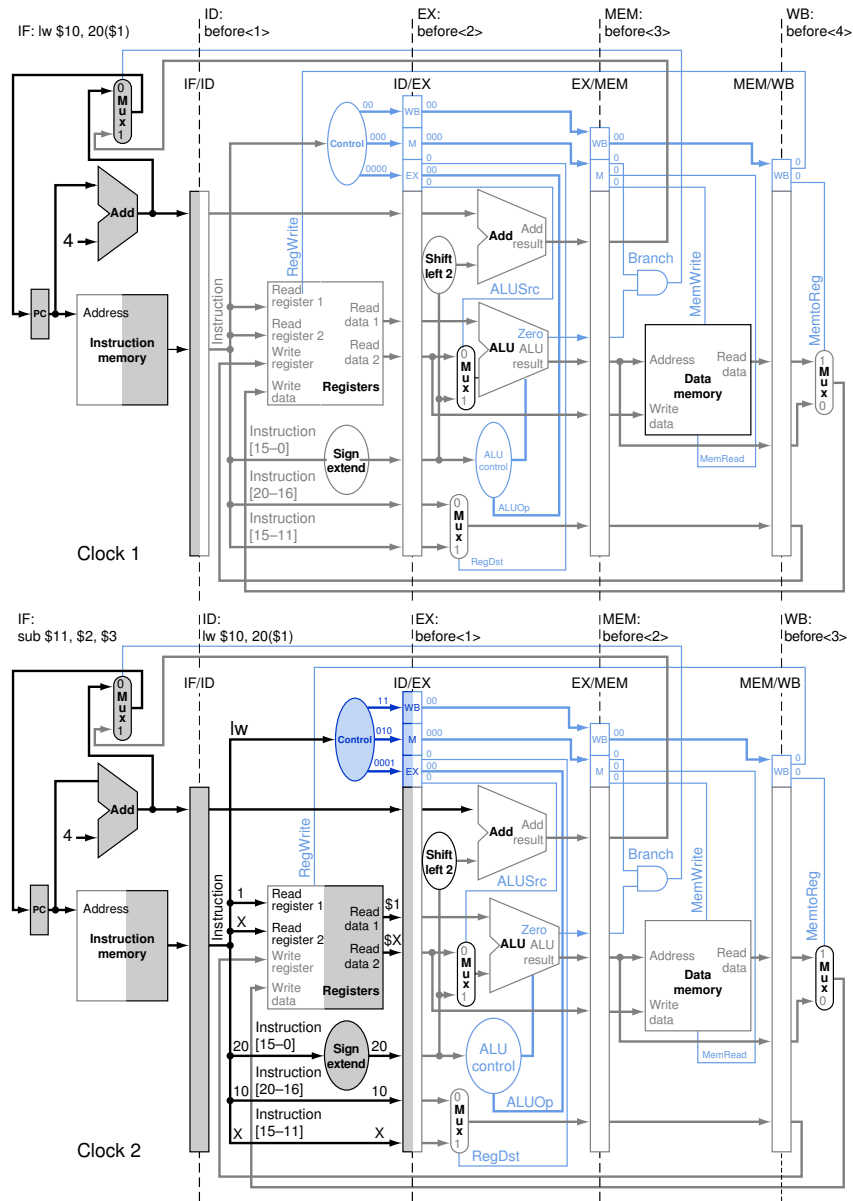


FIGURE 6.14.5 Clock cycles 1 and 2. The phrase “before i ” means the i th instruction before lw . The lw instruction in the top datapath is in the IF stage. At the end of the clock cycle, the lw instruction is in the IF/ID pipeline registers. In the second clock cycle, seen in the bottom datapath, the 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 $\$1$ 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 6.25 on page 401.

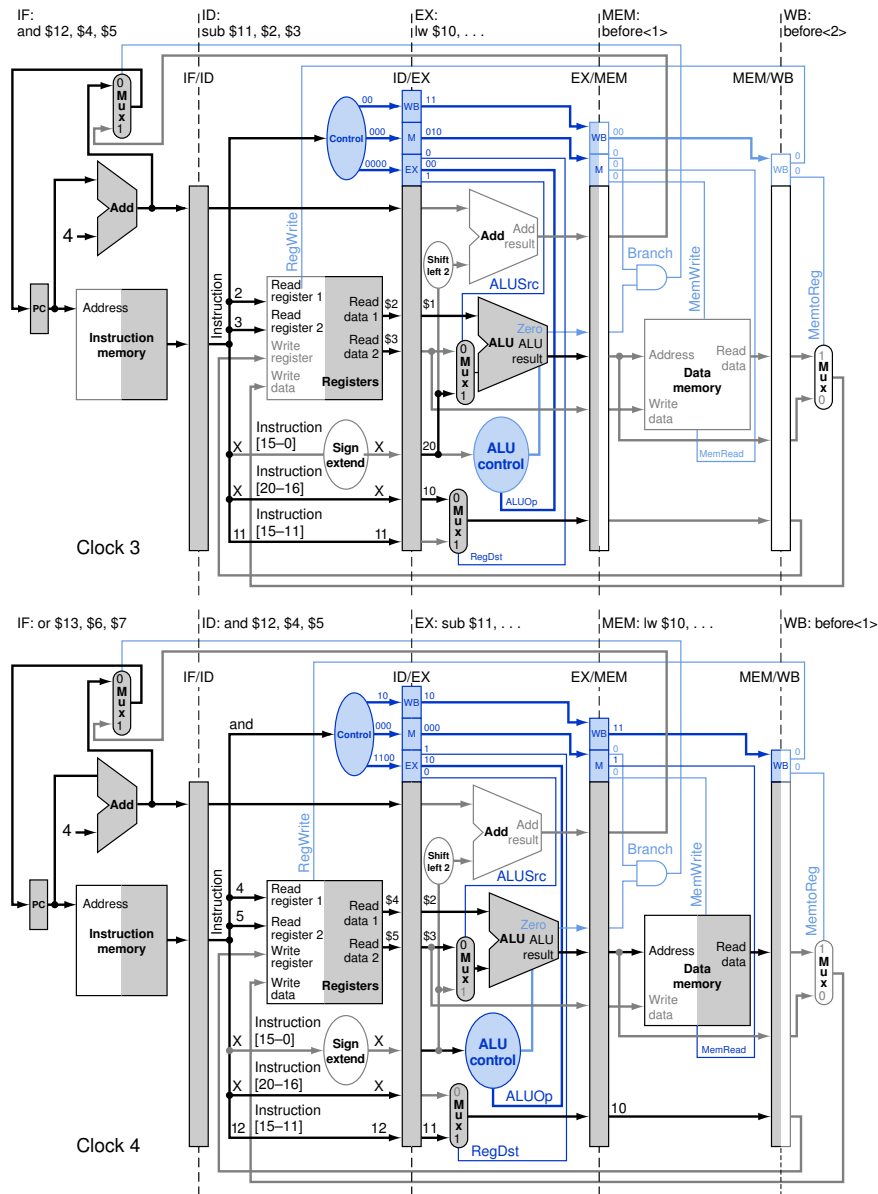


FIGURE 6.14.6 Clock cycles 3 and 4. In the top diagram, `lw` enters the EX stage in the third clock cycle, adding \$1 and 20 to form the address in the EX/MEM pipeline register. (The `lw` instruction is written `lw $10, . . .` upon reaching EX because the identity of instruction operands is not needed 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, and 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.

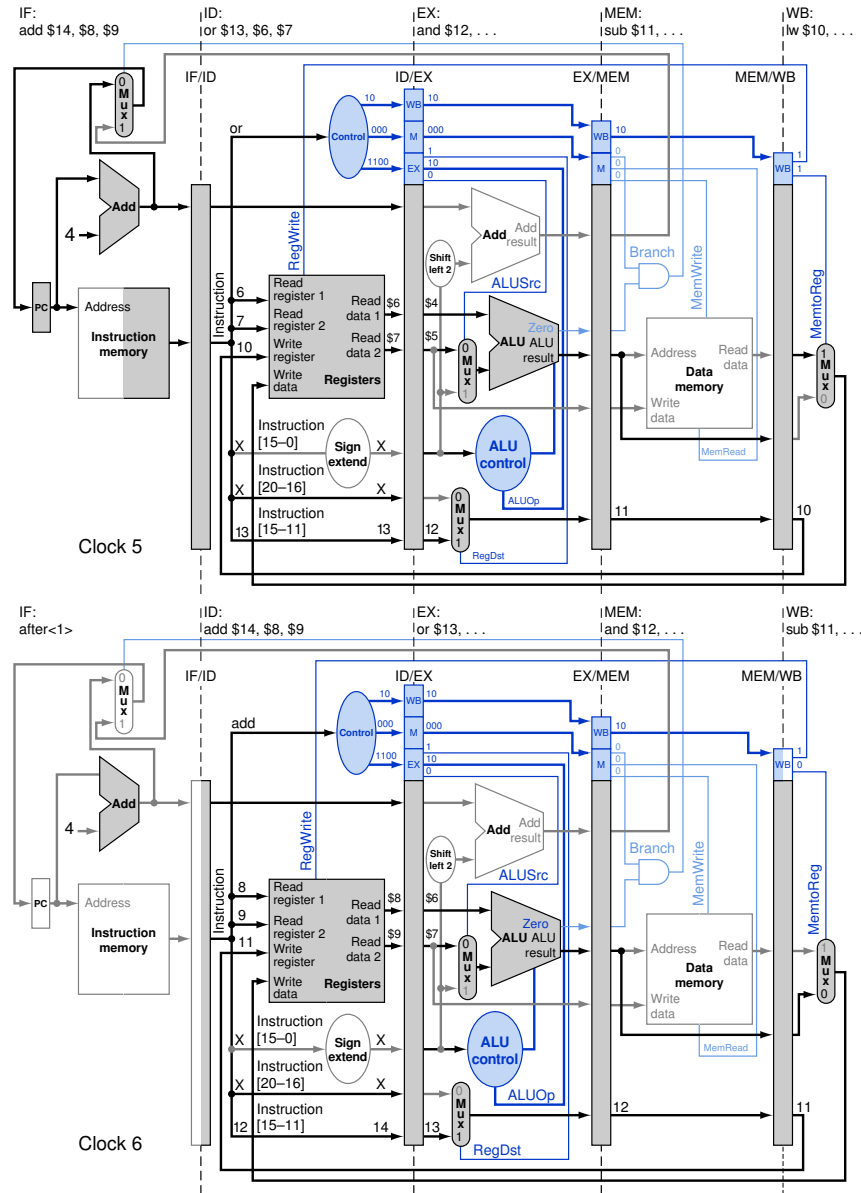


FIGURE 6.14.7 Clock cycles 5 and 6. With `add`, the final instruction in this example, entering IF in the top datapath, all instructions are engaged. By writing the data in MEM/WB into register 10, `lw` completes; both the data and the register number are in MEM/WB. In the same clock cycle, `sub` sends 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 *i*th instruction after `add`.

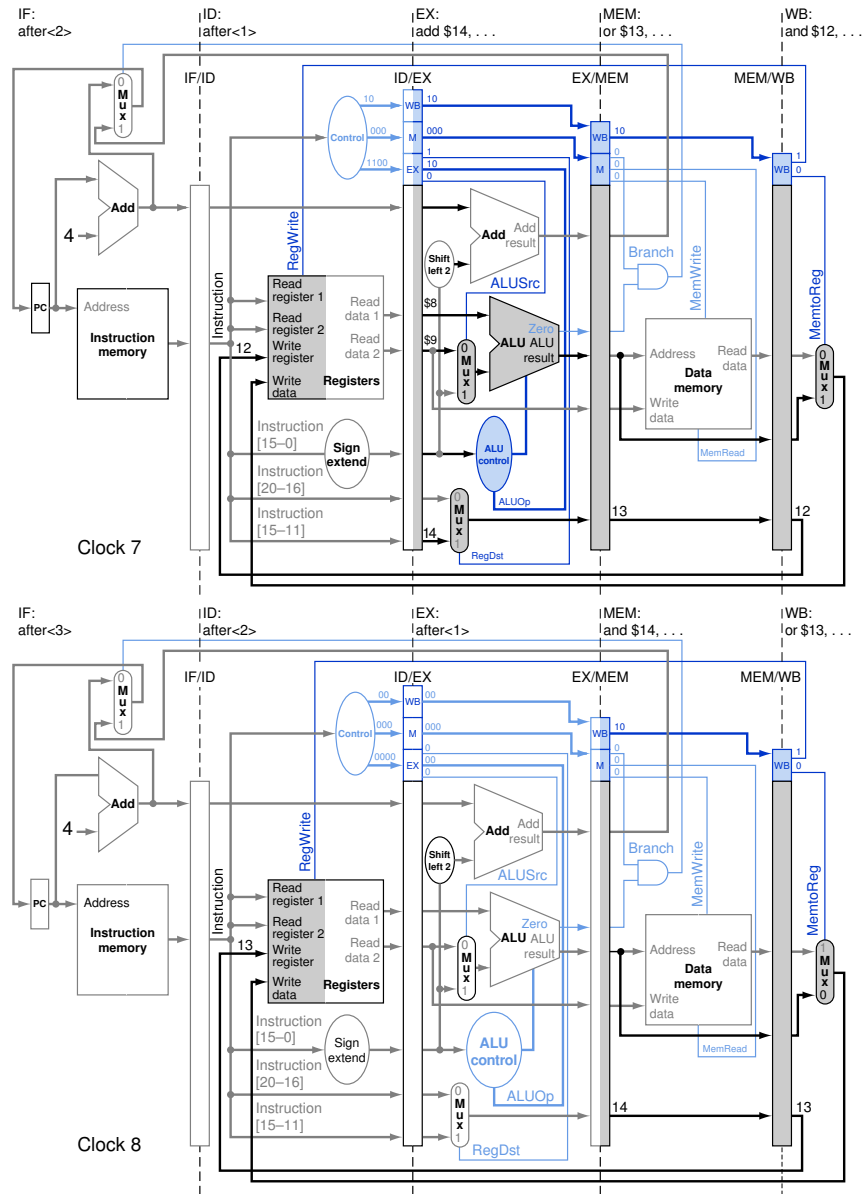


FIGURE 6.14.8 Clock cycles 7 and 8. In the top datapath, the `add` instruction brings up the rear, adding the values corresponding to registers \$8 and \$9 during the EX stage. The result of the `or` instruction is passed from EX/MEM to MEM/WB in the MEM stage, and the WB stage writes the result of the `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.

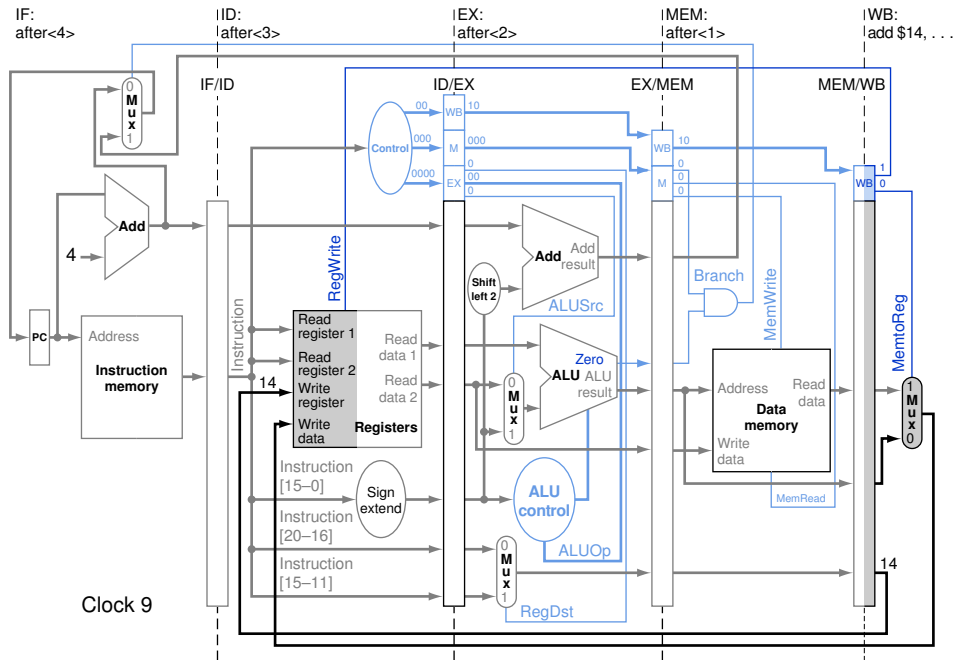


FIGURE 6.14.9 Clock cycle 9. The WB stage writes the sum in MEM/WB into register \$14, completing add and the five-instruction sequence. The instructions after add are shown as inactive for pedagogical reasons.

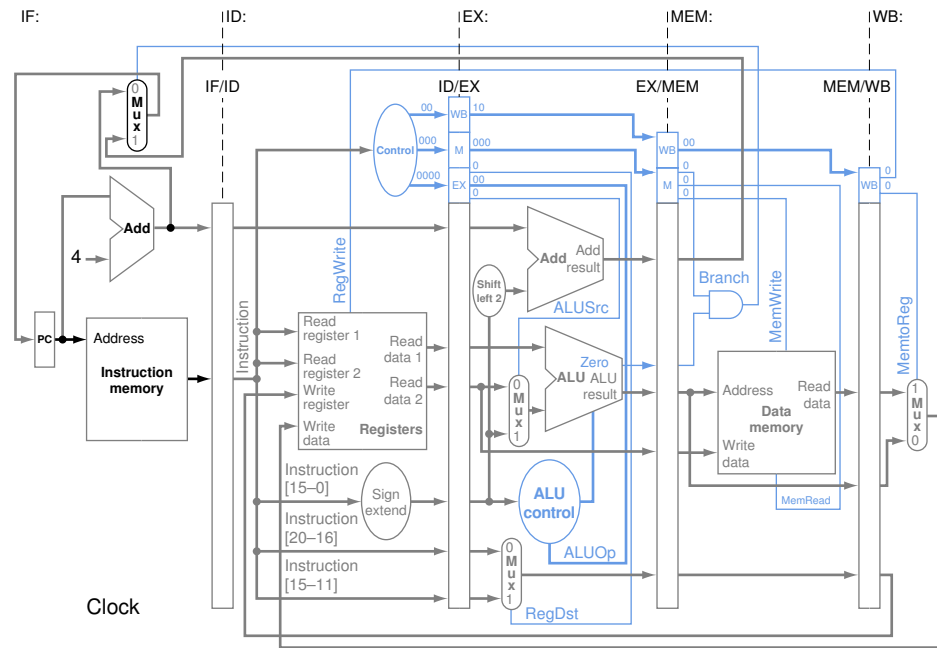


FIGURE 6.14.10 A blank single-clock-cycle pipeline diagram with control.