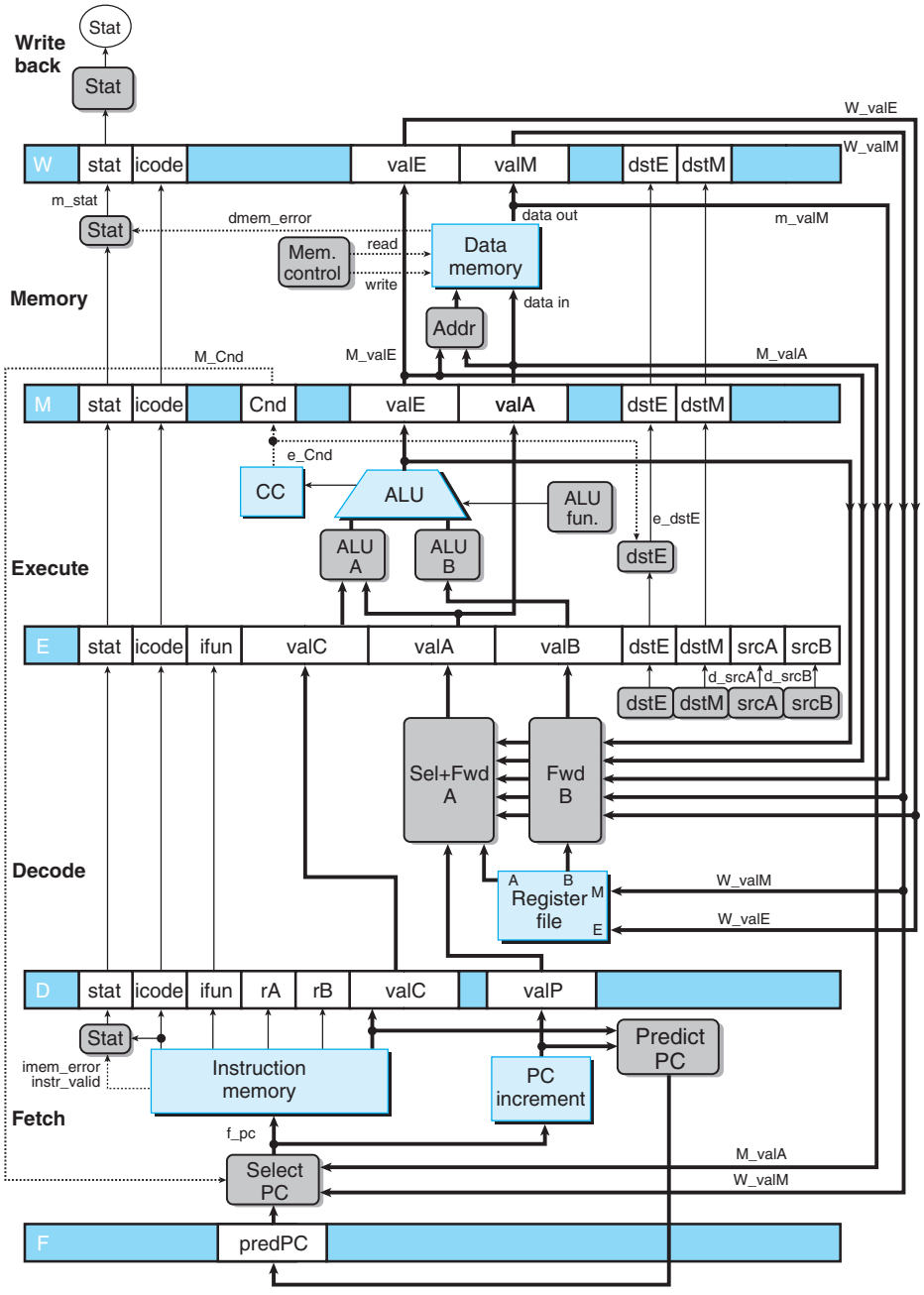

Figure 4.52 Hardware structure of PIPE, our final pipelined implementation. The additional bypassing paths enable forwarding the results from the three preceding instructions. This allows us to handle most forms of data hazards without stalling the pipeline.

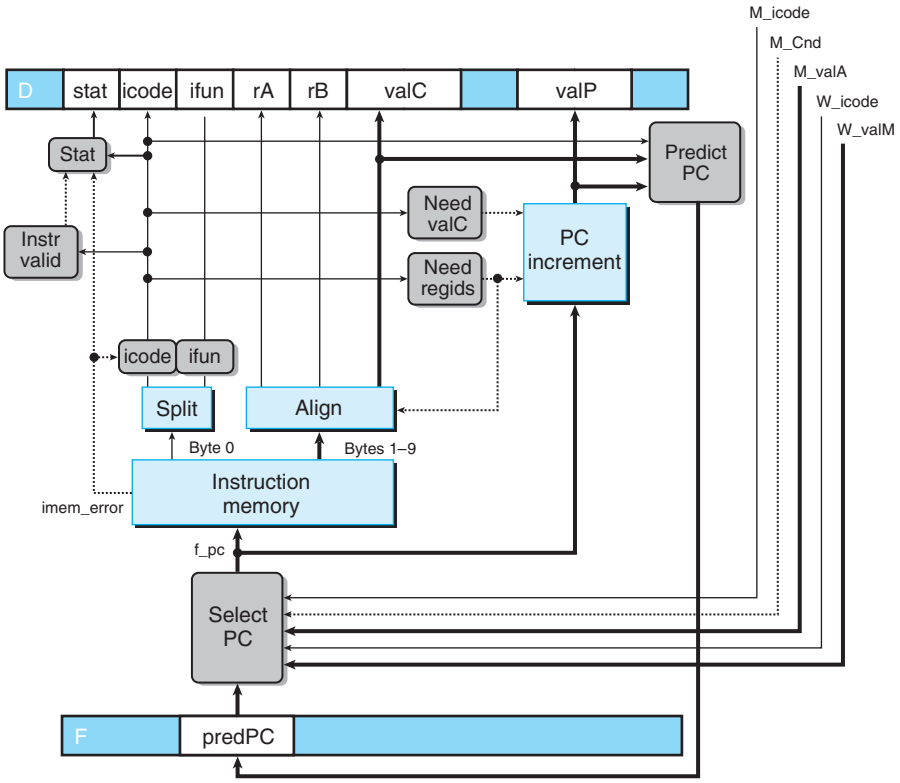

Figure 4.57 PIPE PC selection and fetch logic. Within the one cycle time limit, the processor can only predict the address of the next instruction.

memory and for extracting the different instruction fields are the same as those we considered for SEQ (see the fetch stage in Section 4.3.4).

The PC selection logic chooses between three program counter sources. As a mispredicted branch enters the memory stage, the value of valP for this instruction (indicating the address of the following instruction) is read from pipeline register M (signal M_valA). When a `ret` instruction enters the write-back stage, the return address is read from pipeline register W (signal W_valM). All other cases use the predicted value of the PC, stored in pipeline register F (signal F_predPC):

```
word f_pc = [
        # Mispredicted branch.  Fetch at incremented PC
        M_icode == IJXX && !M_Cnd : M_valA;
        # Completion of RET instruction
        W_icode == IRET : W_valM;
        # Default: Use predicted value of PC
        1 : F_predPC;
];
```

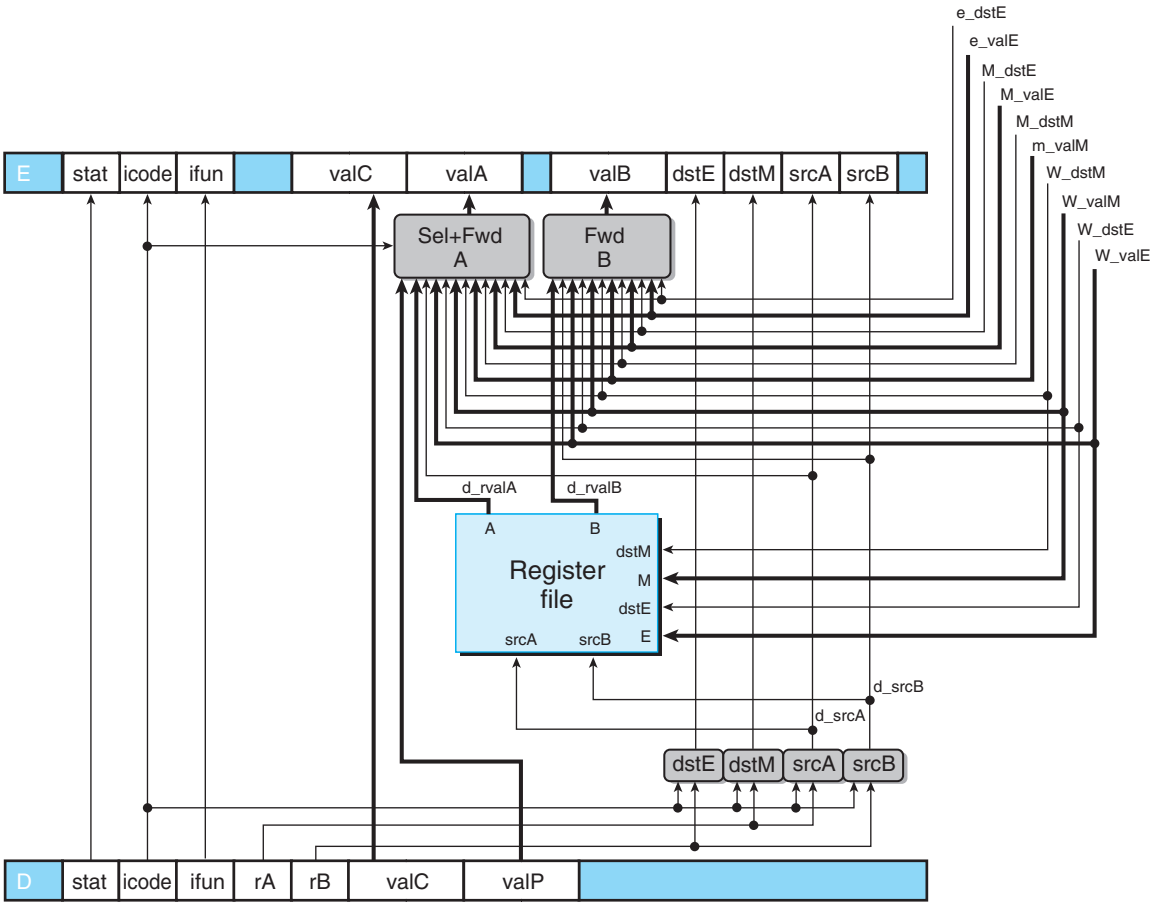

Figure 4.58 PIPE decode and write-back stage logic. No instruction requires both valP and the value read from register port A, and so these two can be merged to form the signal valA for later stages. The block labeled "Sel+Fwd A" performs this task and also implements the forwarding logic for source operand valA. The block labeled "Fwd B" implements the forwarding logic for source operand valB. The register write locations are specified by the dstE and dstM signals from the write-back stage rather than from the decode stage, since it is writing the results of the instruction currently in the write-back stage.

do not need the value read from the A port of the register file. This selection is controlled by the icode signal for this stage. When signal D_icode matches the instruction code for either `call` or `jXX`, this block should select D_valP as its output.

As mentioned in Section 4.5.5, there are five different forwarding sources, each with a data word and a destination register ID:

Practice Problem 4.34 (solution page 527)

Write HCL code for the signal d_valB, giving the value for source operand valB supplied to pipeline register E.

One small part of the write-back stage remains. As shown in Figure 4.52, the overall processor status Stat is computed by a block based on the status value in pipeline register W. Recall from Section 4.1.1 that the code should indicate either normal operation (AOK) or one of the three exception conditions. Since pipeline register W holds the state of the most recently completed instruction, it is natural to use this value as an indication of the overall processor status. The only special case to consider is when there is a bubble in the write-back stage. This is part of normal operation, and so we want the status code to be AOK for this case as well:

```
word Stat = [
        W_stat == SBUB : SAOK;
        1 : W_stat;
];
```

Execute Stage

Figure 4.60 shows the execute stage logic for PIPE. The hardware units and the logic blocks are identical to those in SEQ, with an appropriate renaming of signals. We can see the signals e_valE and e_dstE directed toward the decode stage as one of the forwarding sources. One difference is that the logic labeled "Set CC," which determines whether or not to update the condition codes, has signals m_stat and W_stat as inputs. These signals are used to detect cases where an instruction

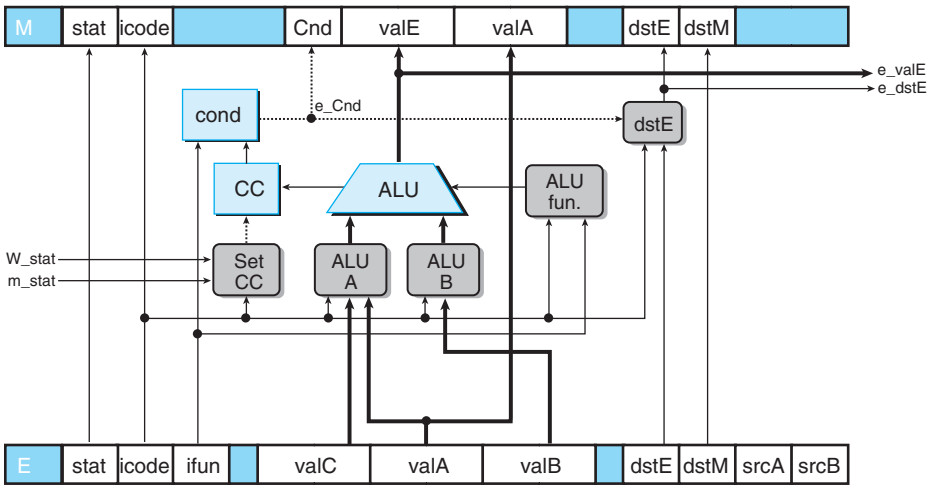

Figure 4.60 PIPE execute stage logic. This part of the design is very similar to the logic in the SEQ implementation.

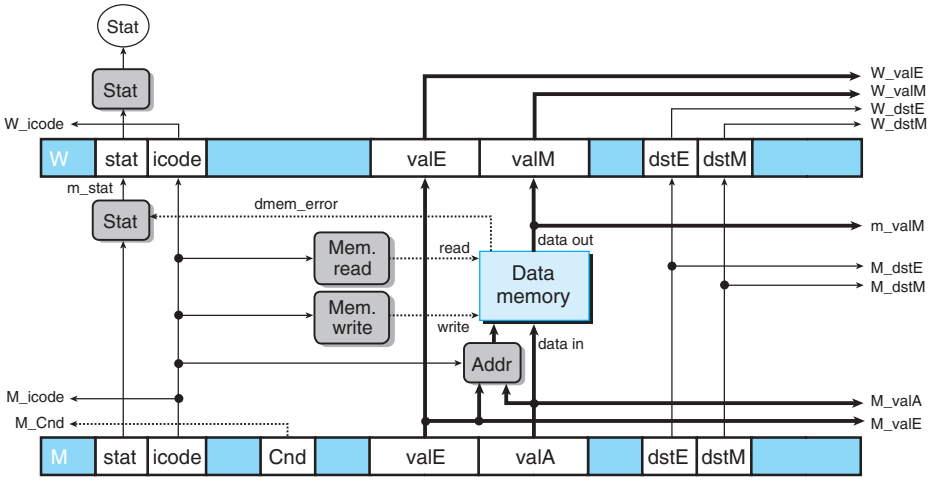

Figure 4.61 PIPE memory stage logic. Many of the signals from pipeline registers M and W are passed down to earlier stages to provide write-back results, instruction addresses, and forwarded results.

causing an exception is passing through later pipeline stages, and therefore any updating of the condition codes should be suppressed. This aspect of the design is discussed in Section 4.5.8.

Practice Problem 4.35 (solution page 527)

Our second case in the HCL code for d_valA uses signal e_dstE to see whether to select the ALU output e_valE as the forwarding source. Suppose instead that we use signal E_dstE, the destination register ID in pipeline register E for this selection. Write a Y86-64 program that would give an incorrect result with this modified forwarding logic.

Memory Stage

Figure 4.61 shows the memory stage logic for PIPE. Comparing this to the memory stage for SEQ (Figure 4.30), we see that, as noted before, the block labeled "Mem. data" in SEQ is not present in PIPE. This block served to select between data sources valP (for `call` instructions) and valA, but this selection is now performed by the block labeled "Sel+Fwd A" in the decode stage. Most other blocks in this stage are identical to their counterparts in SEQ, with an appropriate renaming of the signals. In this figure, you can also see that many of the values in pipeline registers and M and W are supplied to other parts of the circuit as part of the forwarding and pipeline control logic.