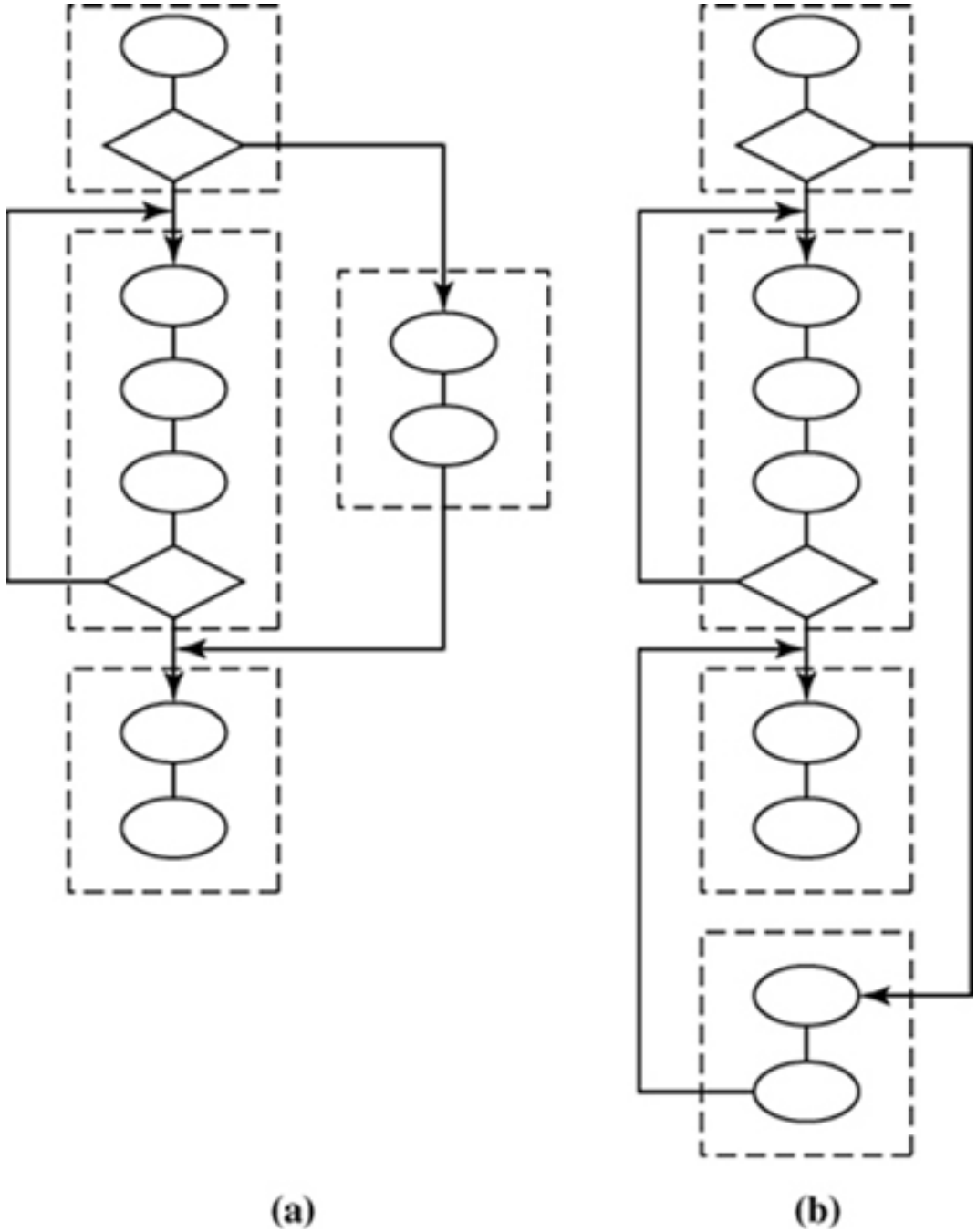


FIgure 5.1: Program Control Flow: (a) The Control Flow Graph (CFG); (b) Mapping the CFG to Sequential Memory Locations.

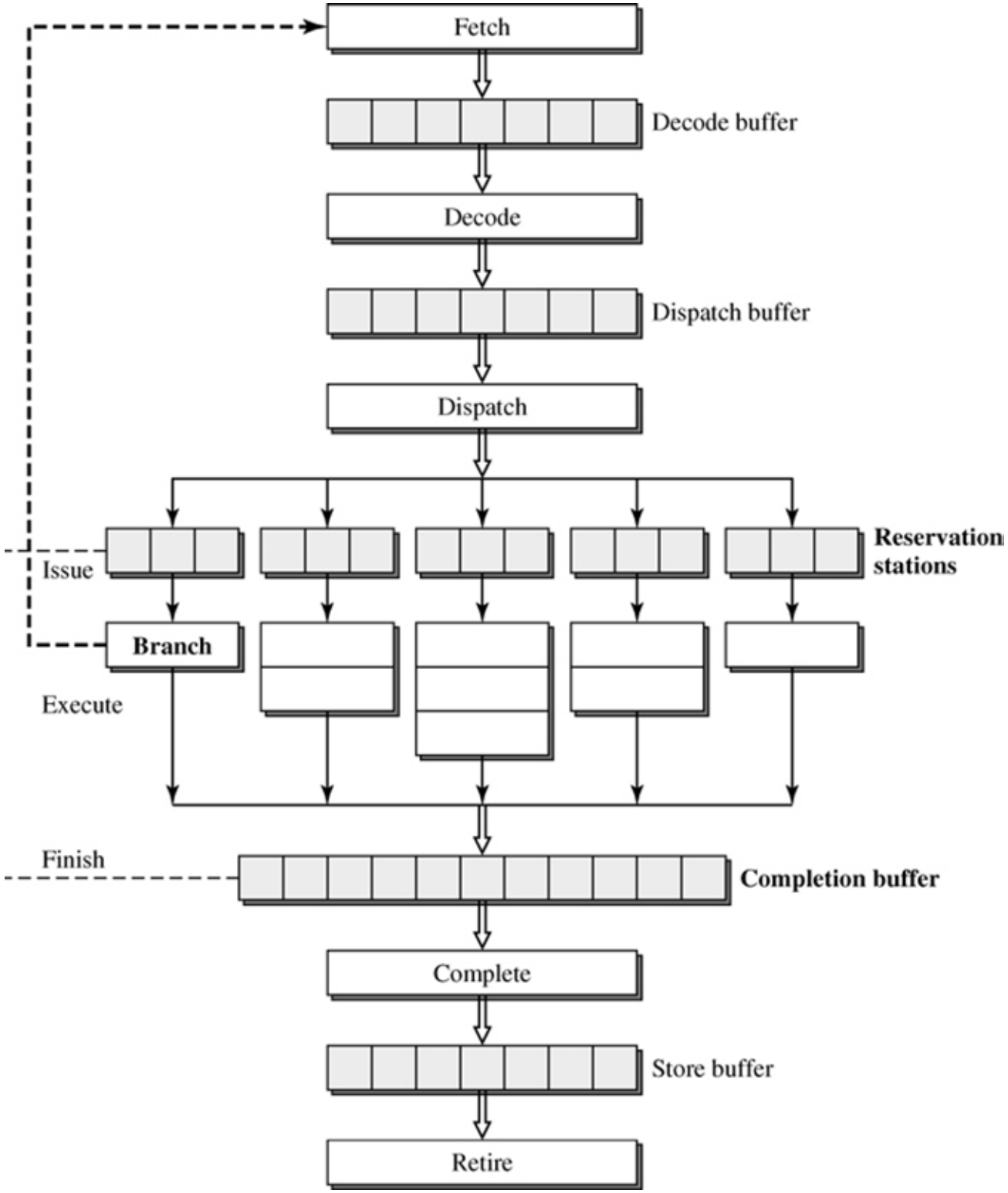


FIgure 5.2: Disruption of Sequential Control Flow by Branch Instructions.

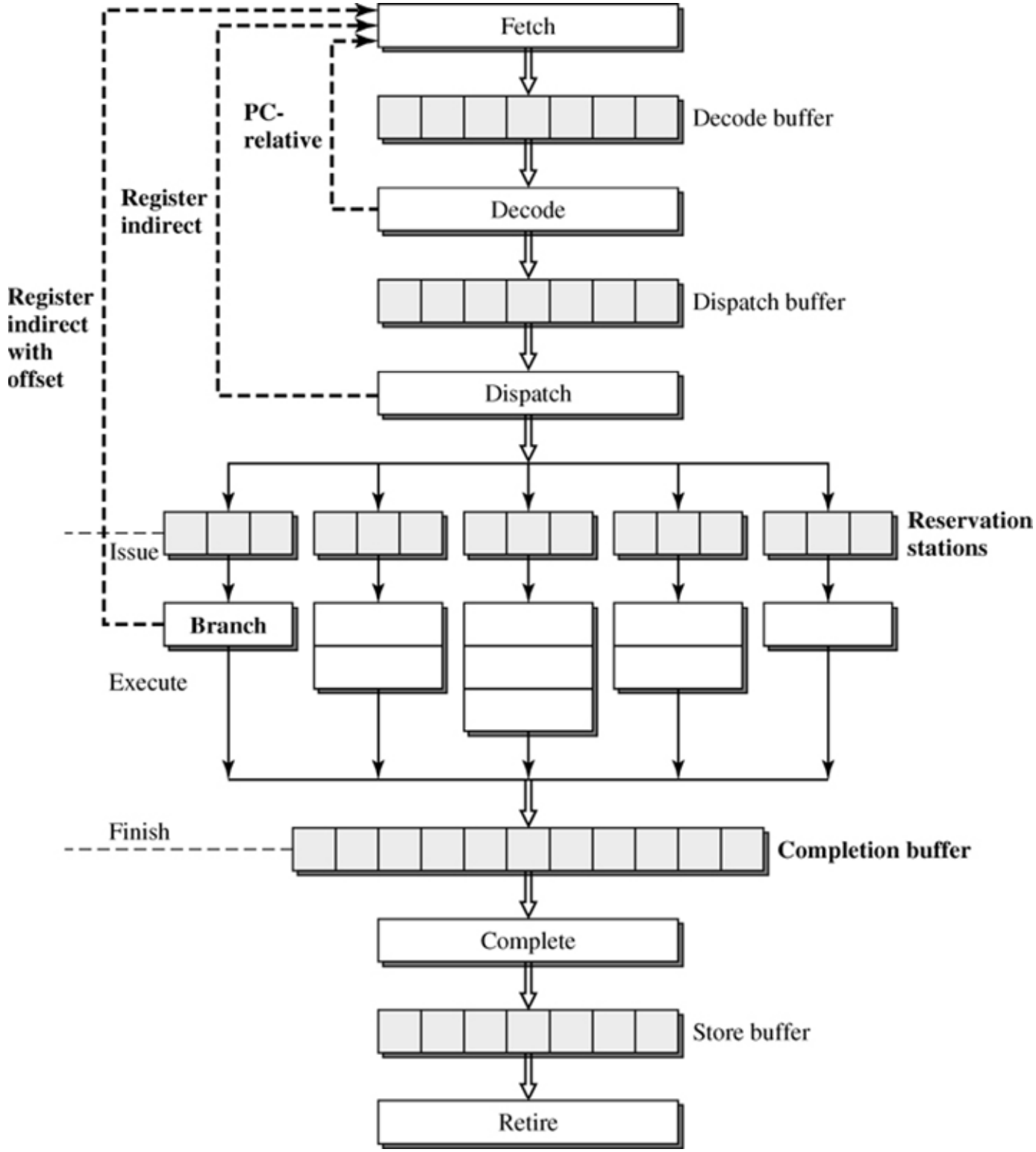


FIgure 5.3: Branch Target Address Generation Penalties.

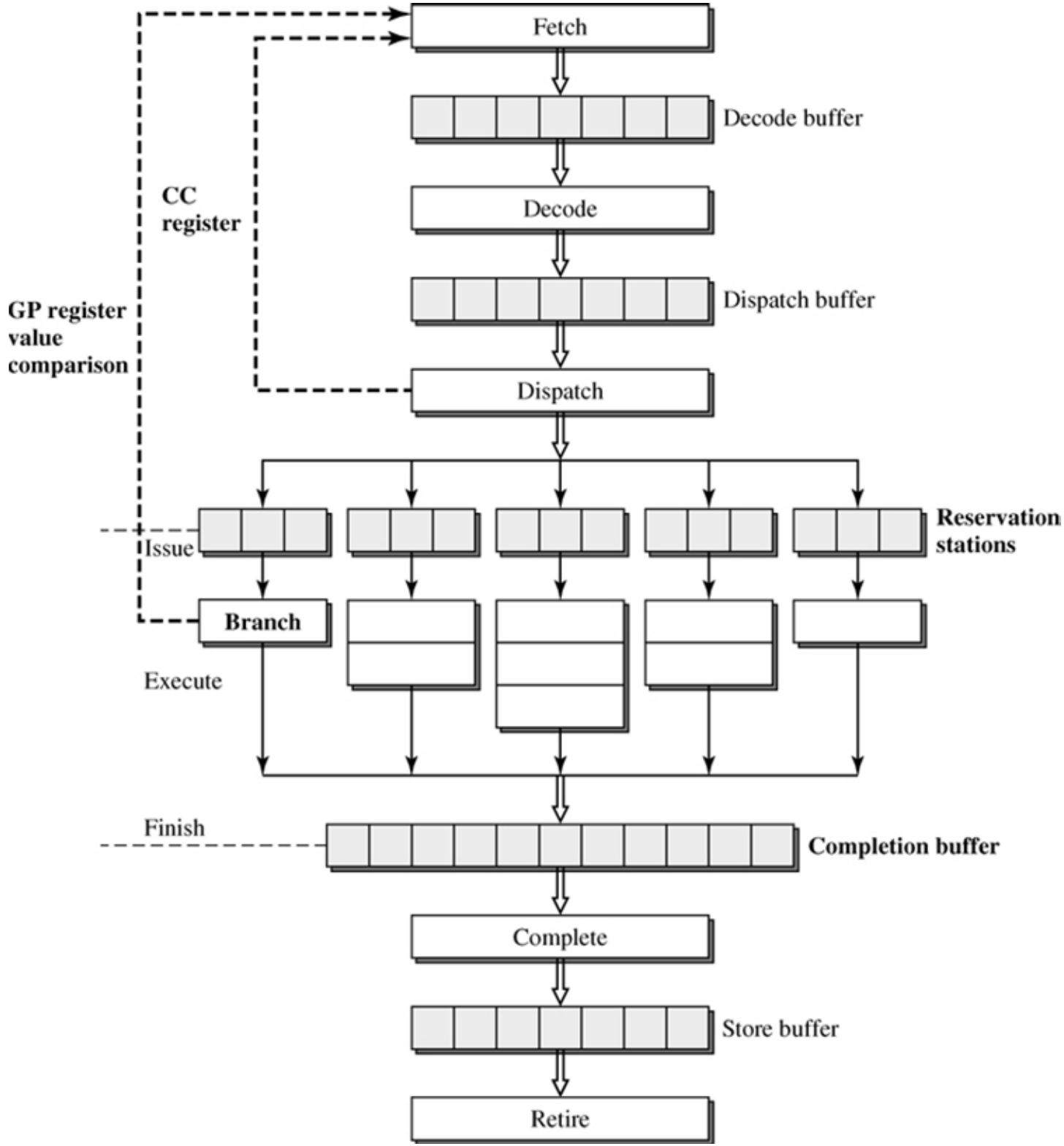


FIgure 5.4: Branch Condition Resolution Penalties.

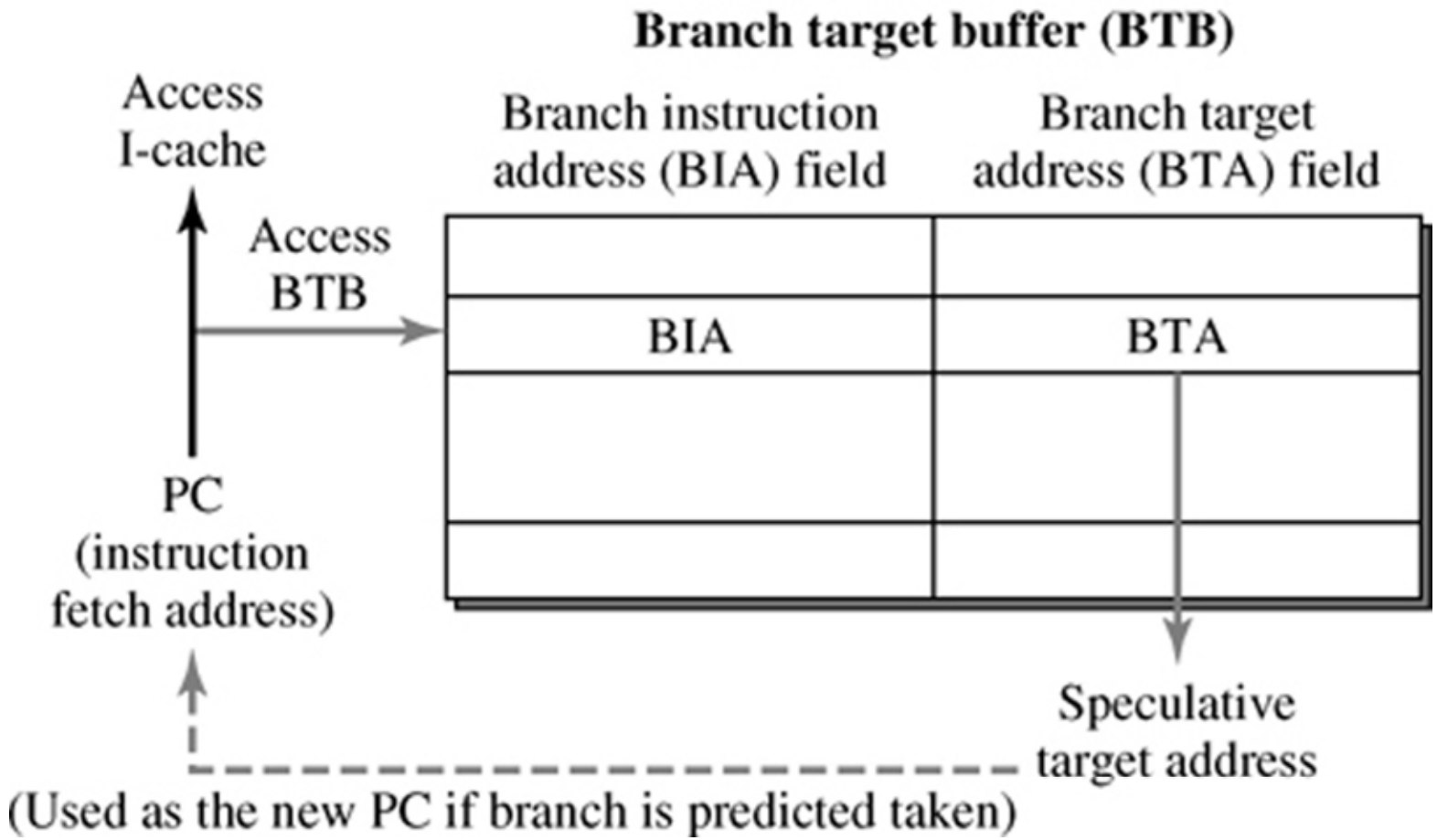


Figure 5.5: Branch Target Speculation Using a Branch Target Buffer.

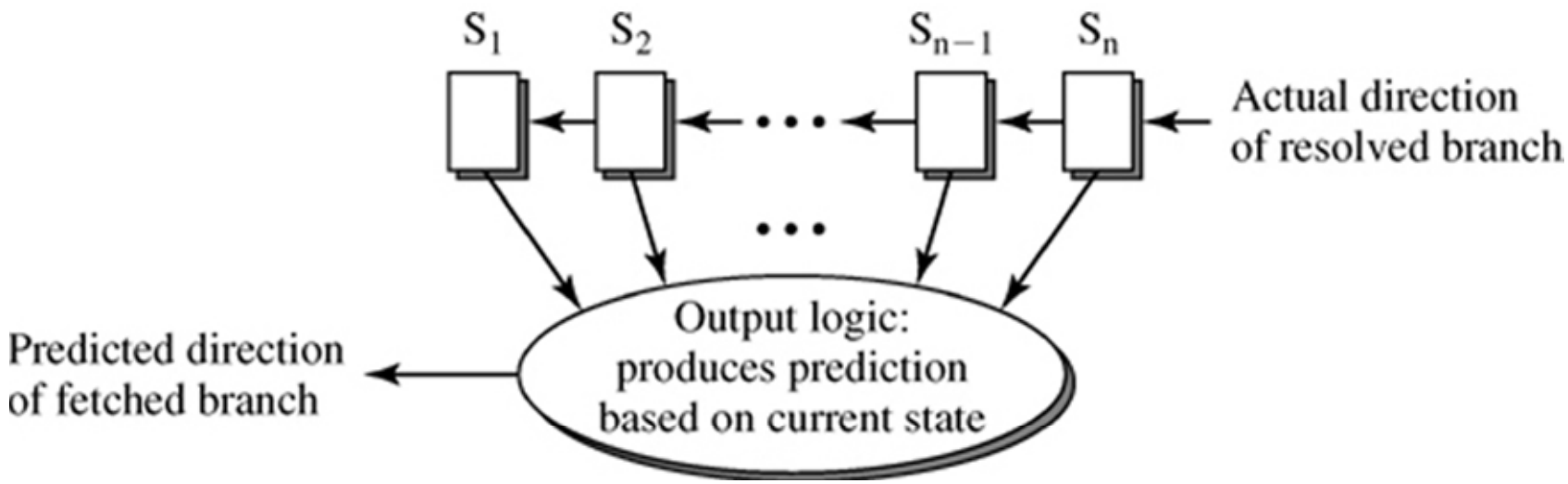
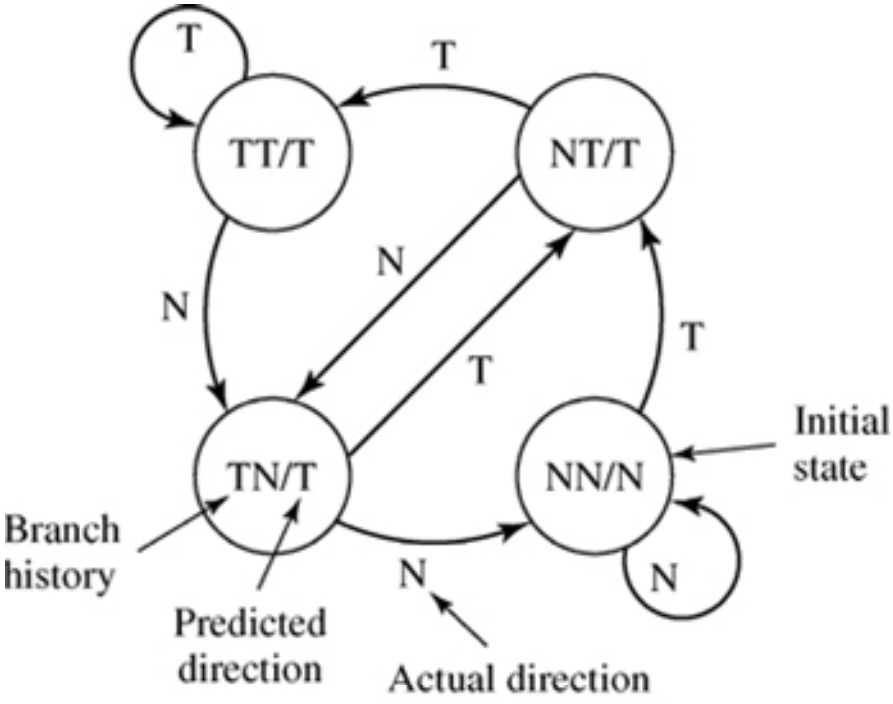


Figure 5.6: FSM Model for History-Based Direction Predictors.



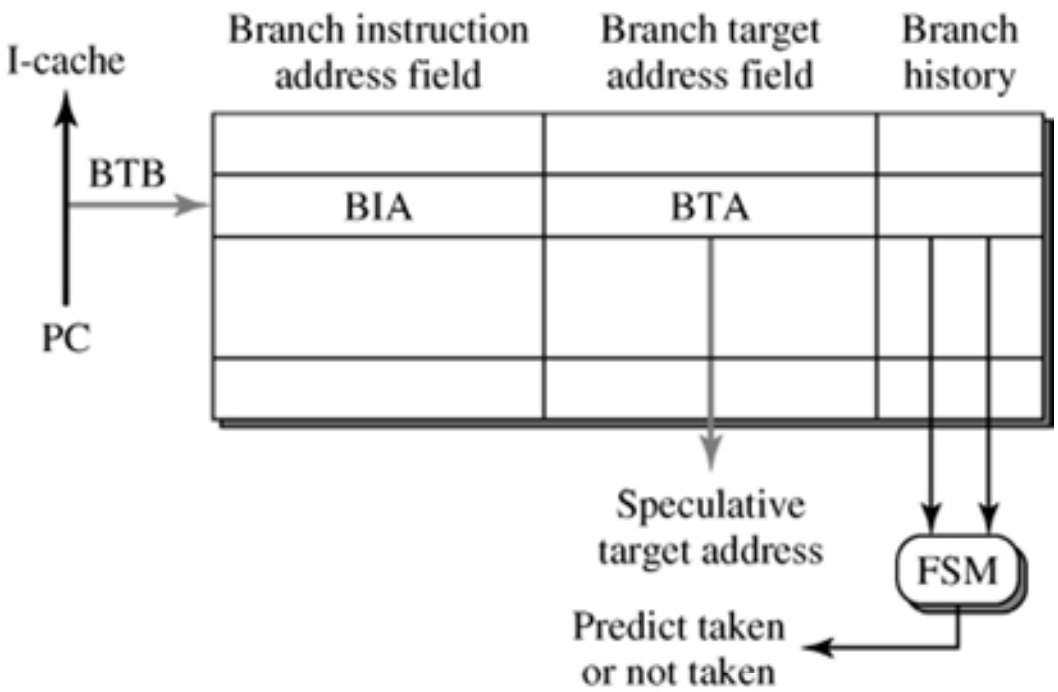


Figure 5.7: History-Based Branch Prediction: (a) A 2-Bit Branch Predictor Algorithm; (b) Branch Target Buffer with an Additional Field for Storing Branch History Bits.

Benchmark	Optimal	"Counter"
spice2g6	97.2	97.0
doduc	94.3	94.3
gcc	89.1	89.1
espresso	89.1	89.1
li	87.1	86.8
eqntott	87.9	87.2

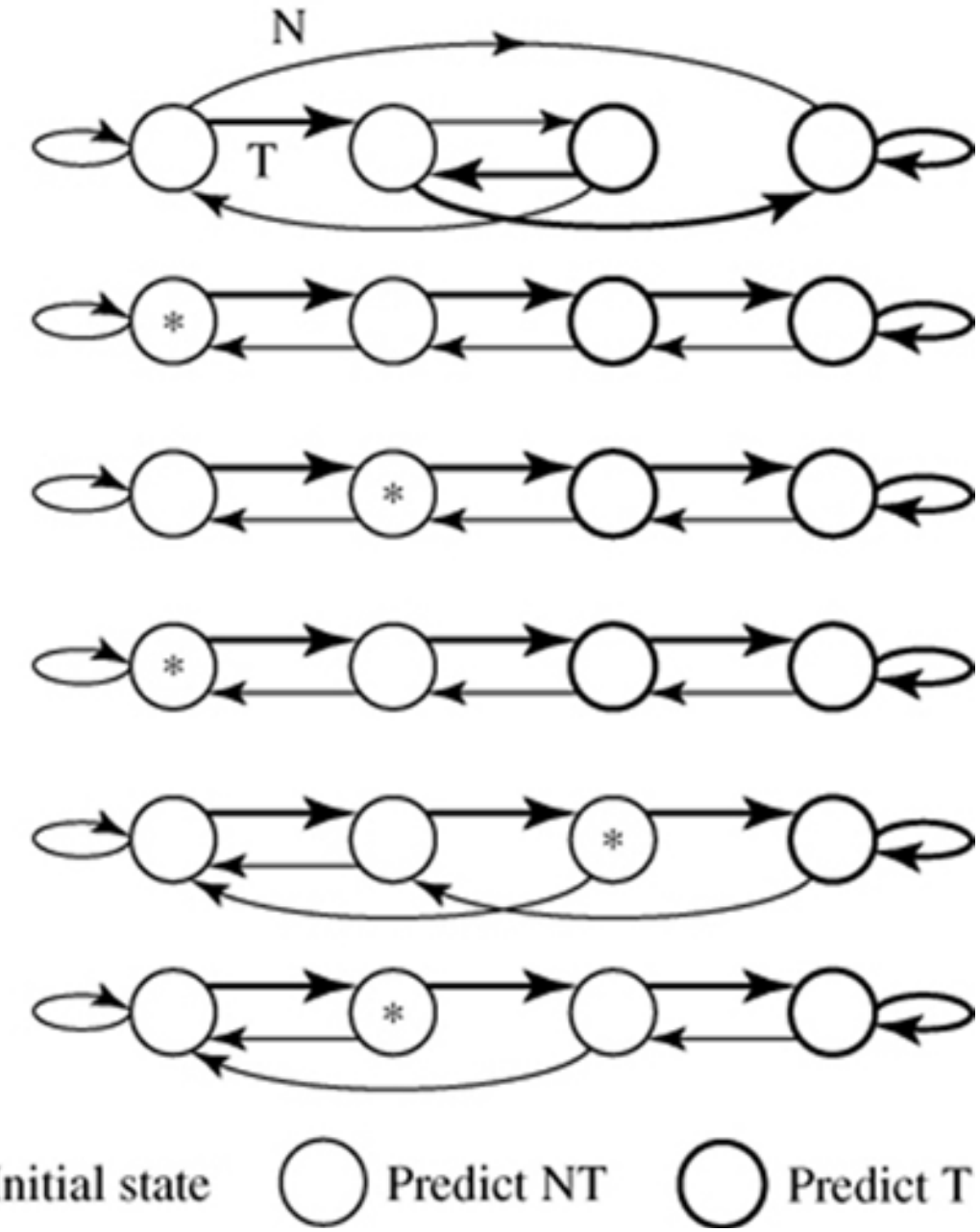


Figure 5.8: Optimal 2-Bit Branch Predictors for Six SPEC Benchmarks from the Nair Study.

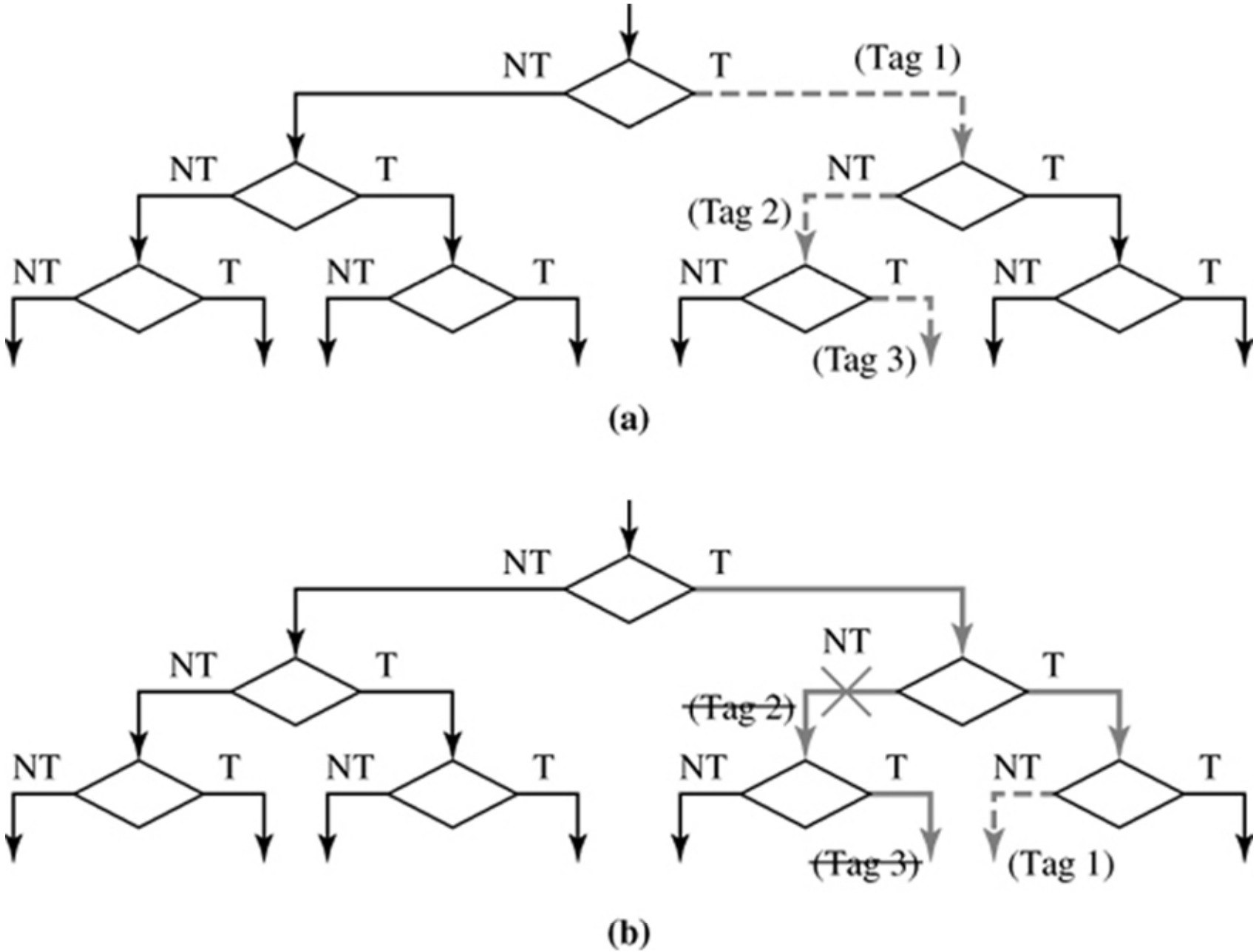


Figure 5.9: Two Aspects of Branch Prediction: (a) Branch Speculation; (b) Branch Validation/Recovery.

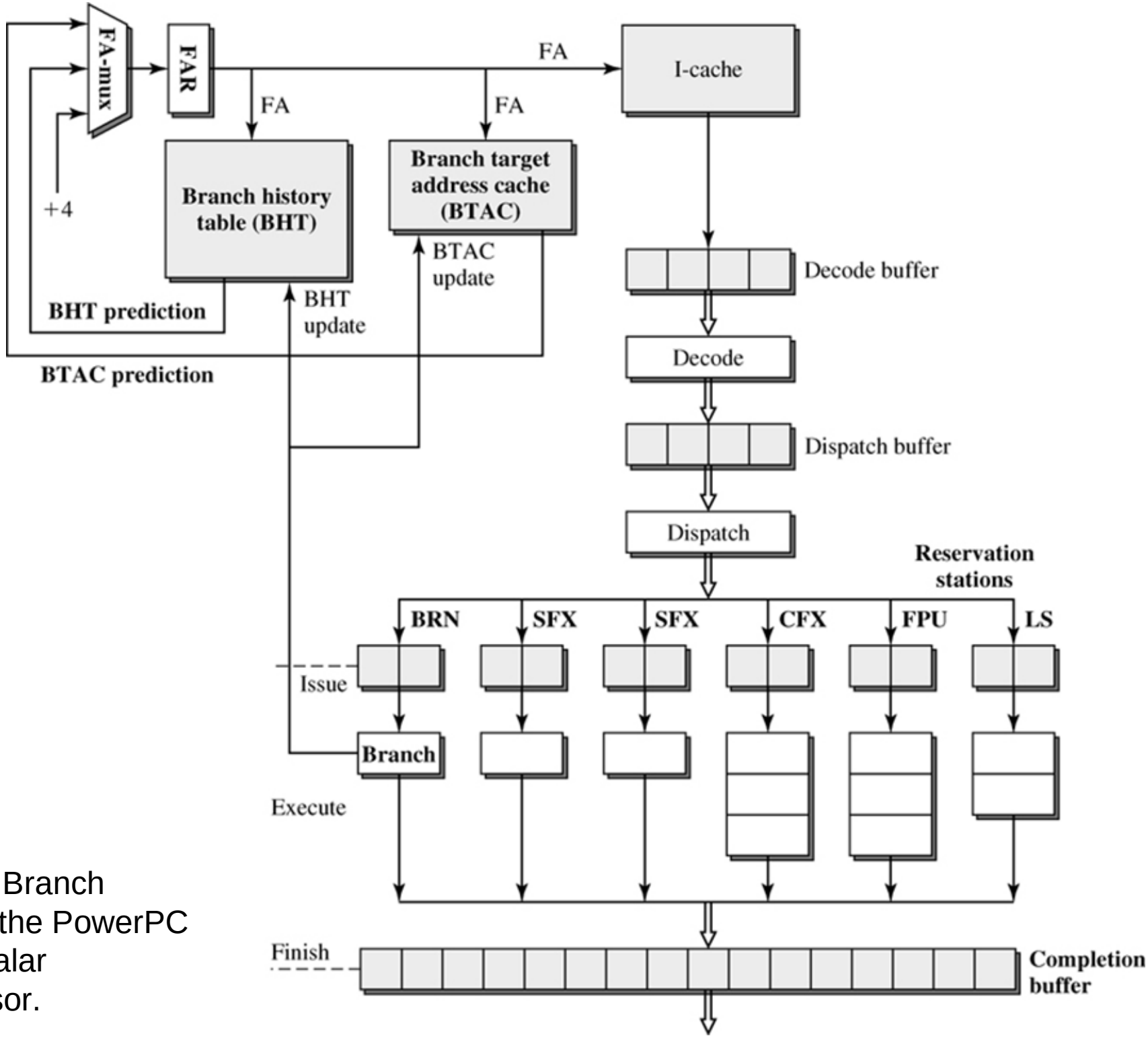


Figure 5.10: Branch Prediction in the PowerPC 604 Superscalar Microprocessor.

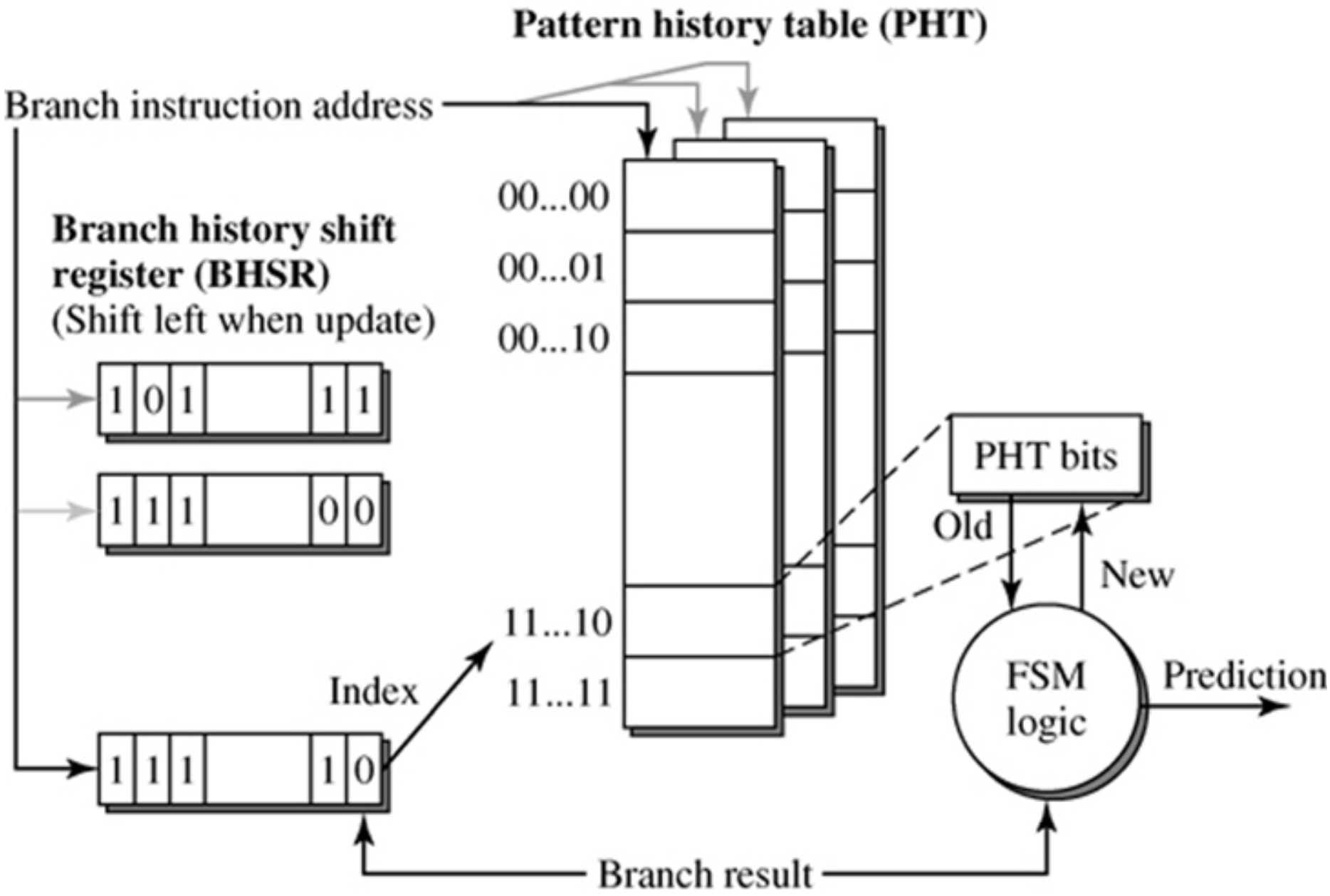


Figure 5.11: Two-Level Adaptive Branch Prediction of Yeh and Patt.
Source: Yeh and Patt, 1991.

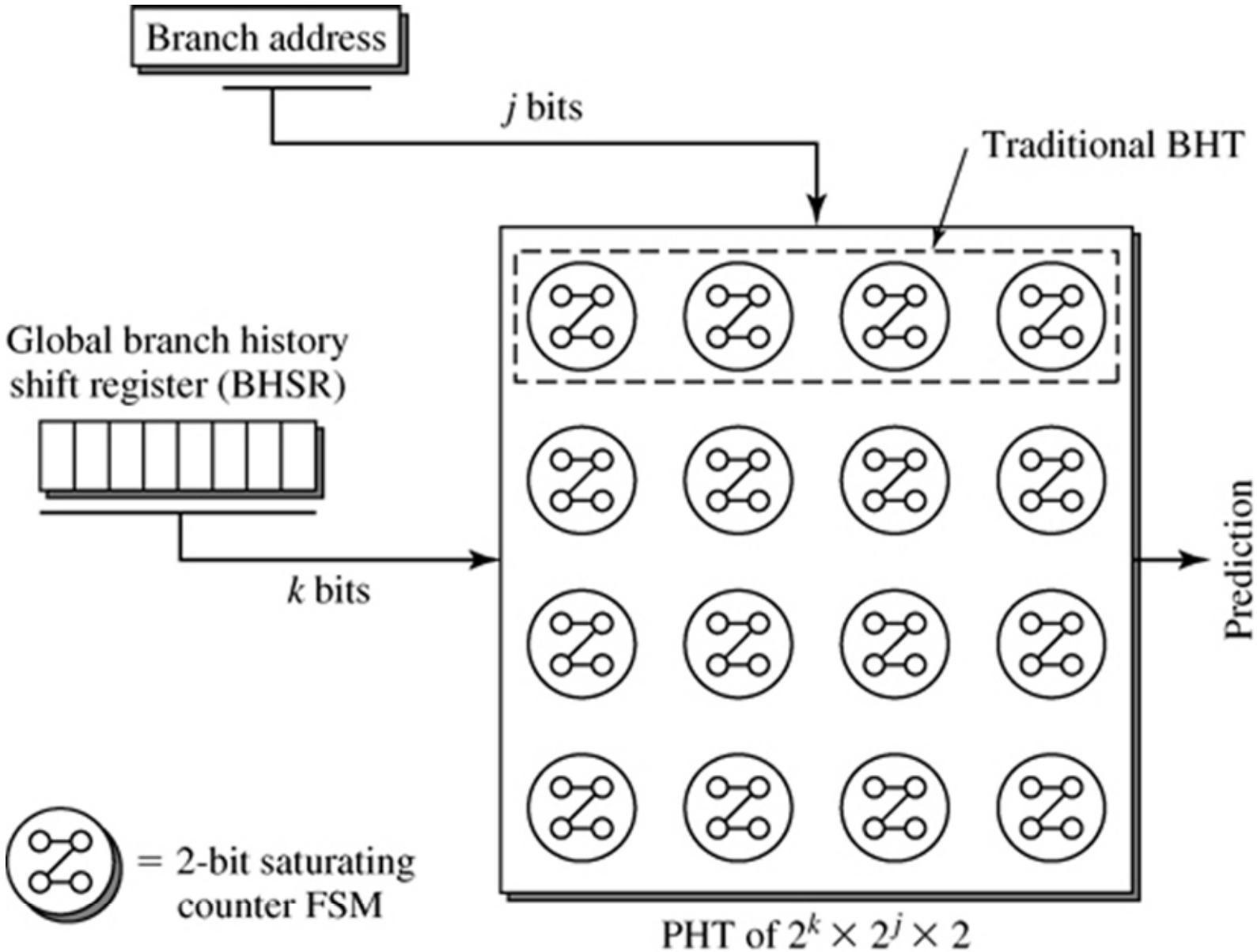


Figure 5.12: Correlated Branch Predictor with Global BHSR and Shared PHTs (GAs).

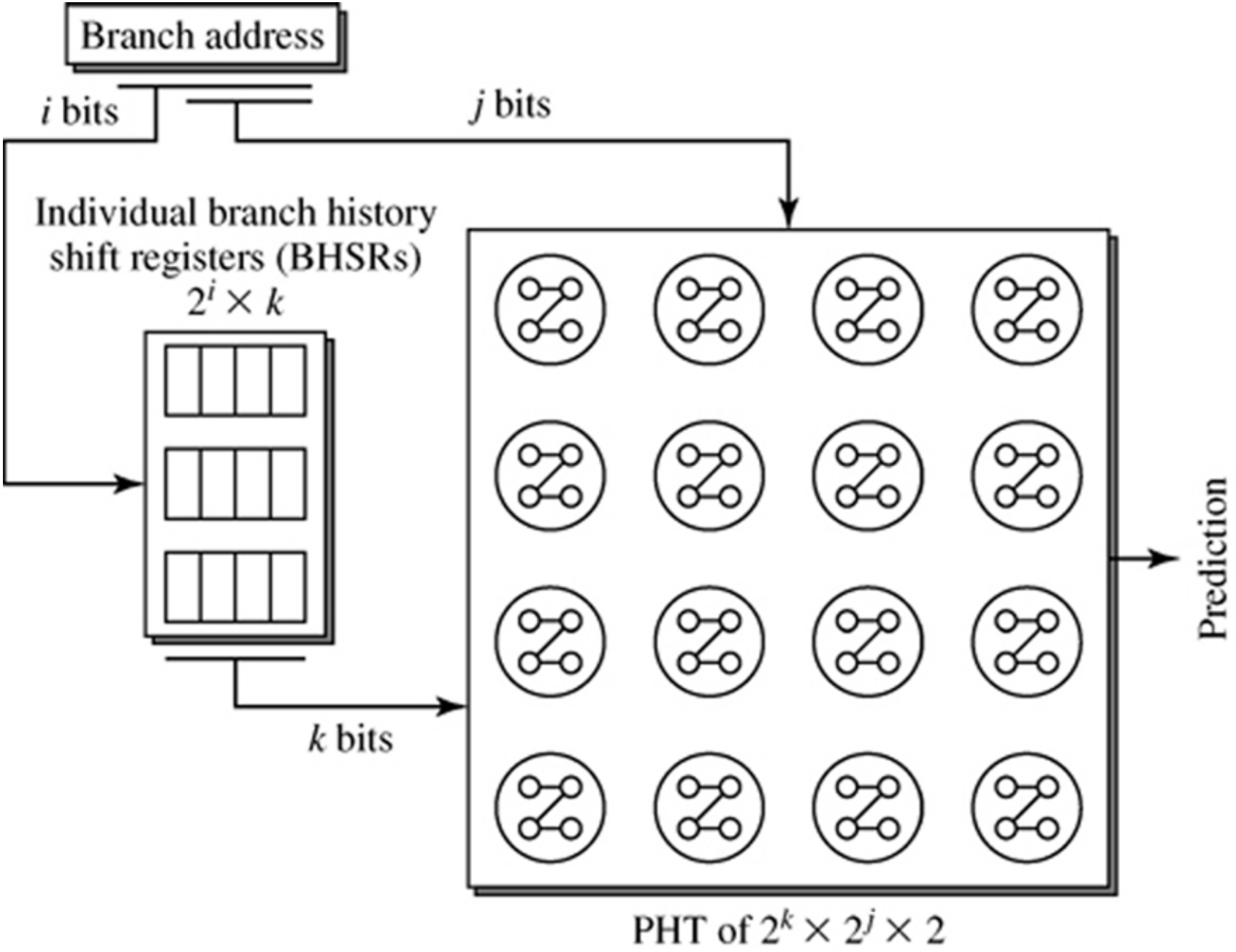


Figure 5.13: Correlated Branch Predictor with Individual BHSRs and Shared PHTs (PAs).

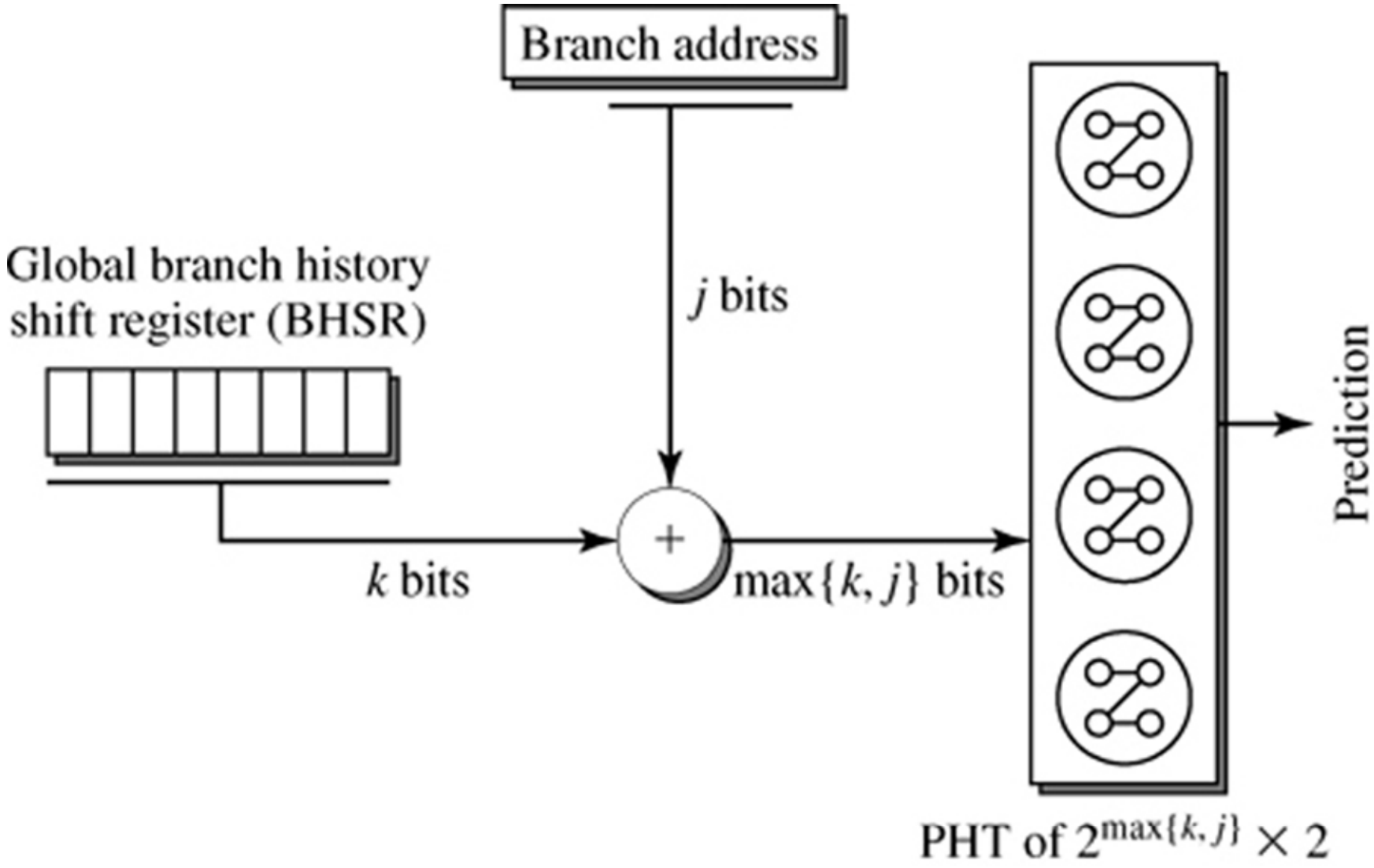


Figure 5.14: The gshare Correlated Branch Predictor of McFarling.
Source: McFarling, 1993.

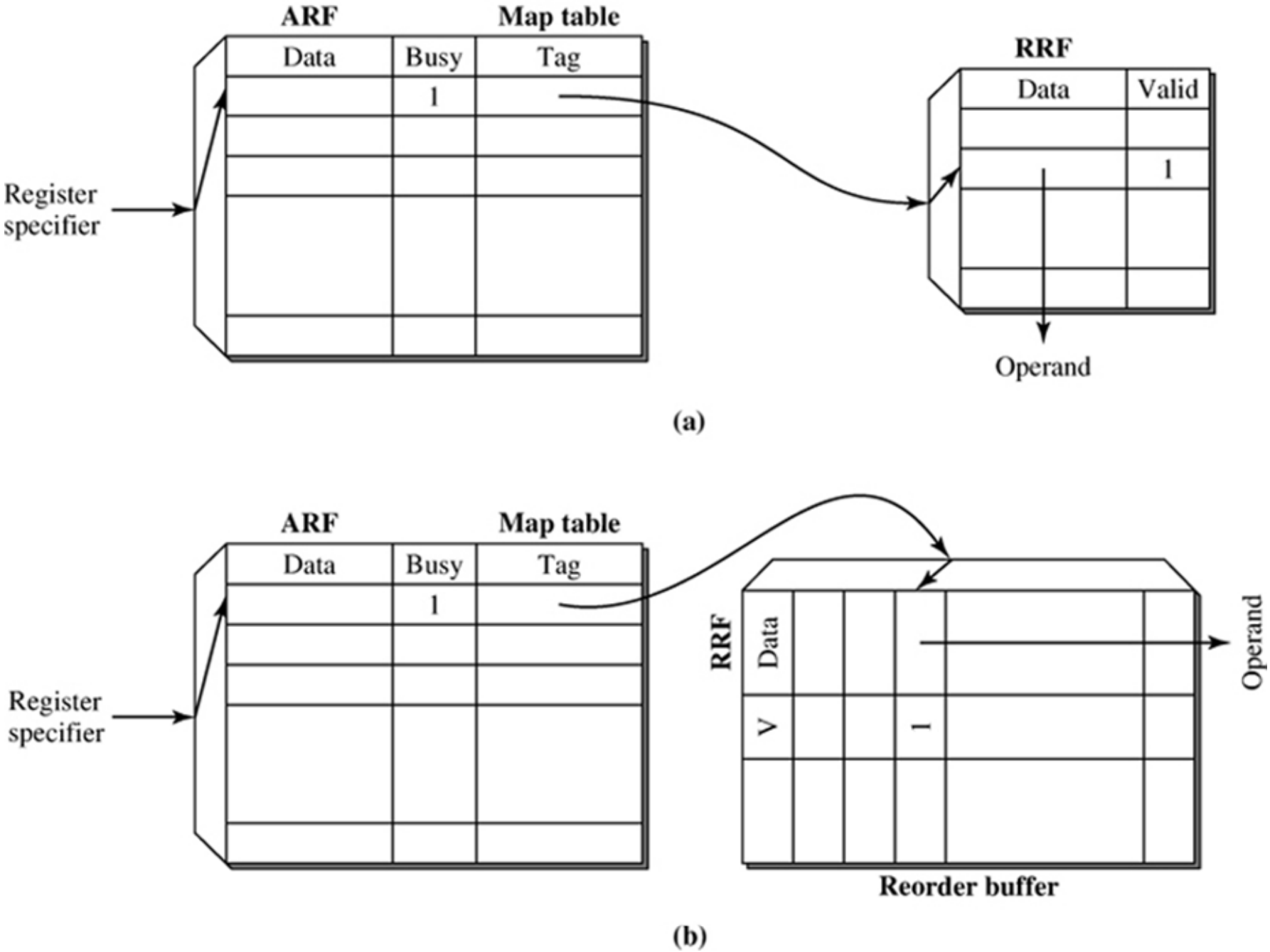


Figure 5.15: Rename Register File (RRF) Implementations: (a) Stand-Alone; (b) Attached to the Reorder Buffer

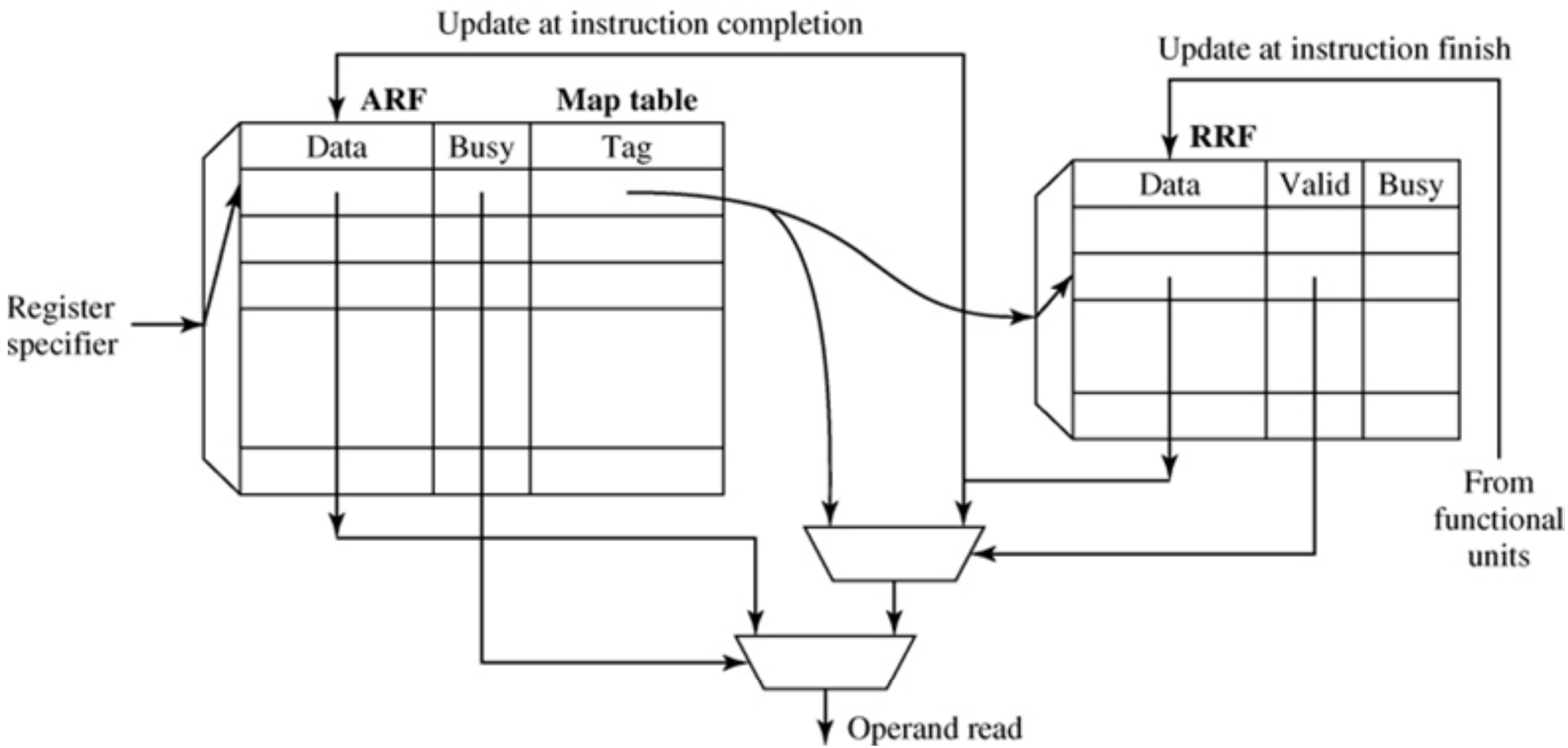


Figure 5.16: Register Renaming Tasks: Source Read, Destination Allocate, and Register Update.

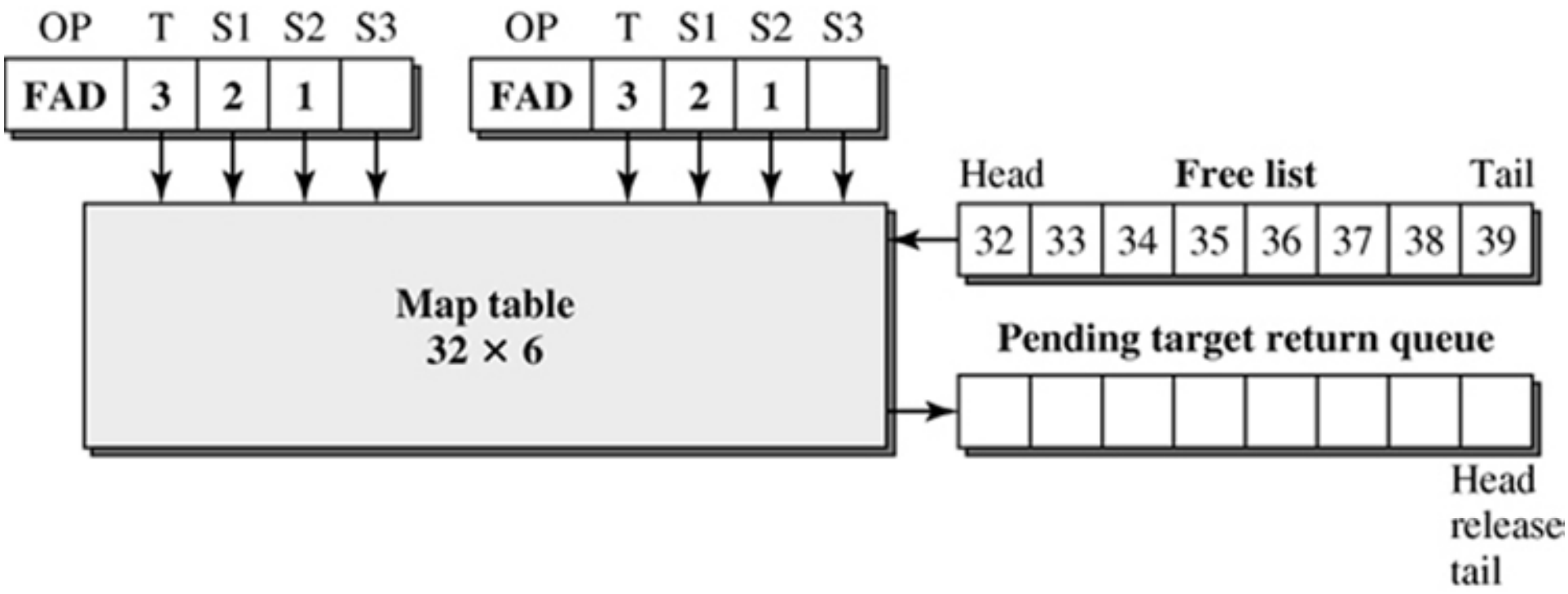


Figure 5.17: Floating-Point Unit (FPU) Register Renaming in the IBM RS/6000.

```
w[i+k].ip = z[i].rp + z[m+i].rp;
w[i+j].rp = e[k+1].rp * (z[i].rp - z[m+i].rp) - e[k+1].ip * (z[i].ip - z[m+i].ip);
```

(a)

```
i1:  f2 ← load,4(r2)
i2:  f0 ← load,4(r5)
i3:  f0 ← fadd,f2,f0
i4:  4(r6) ← store,f0
i5:  f14 ← laod,8(r7)
i6:  f6 ← load,0(r2)
i7:  f5 ← load,0(r3)
i8:  f5 ← fsub,f6,f5
i9:  f4 ← fmul,f14,f5
i10: f15 ← load,12(r7)
i11: f7 ← load,4(r2)
i12: f8 ← load,4(r3)
i13: f8 ← fsub,f7,f8
i14: f8 ← fmul,f15,f8
i15: f8 ← fsub,f4,f8
i16: 0(r8) ← store,f8
```

(b)

Figure 5.18: FFT Code Fragment: (a) Original Source Statements; (b) Compiled Assembly Instructions.

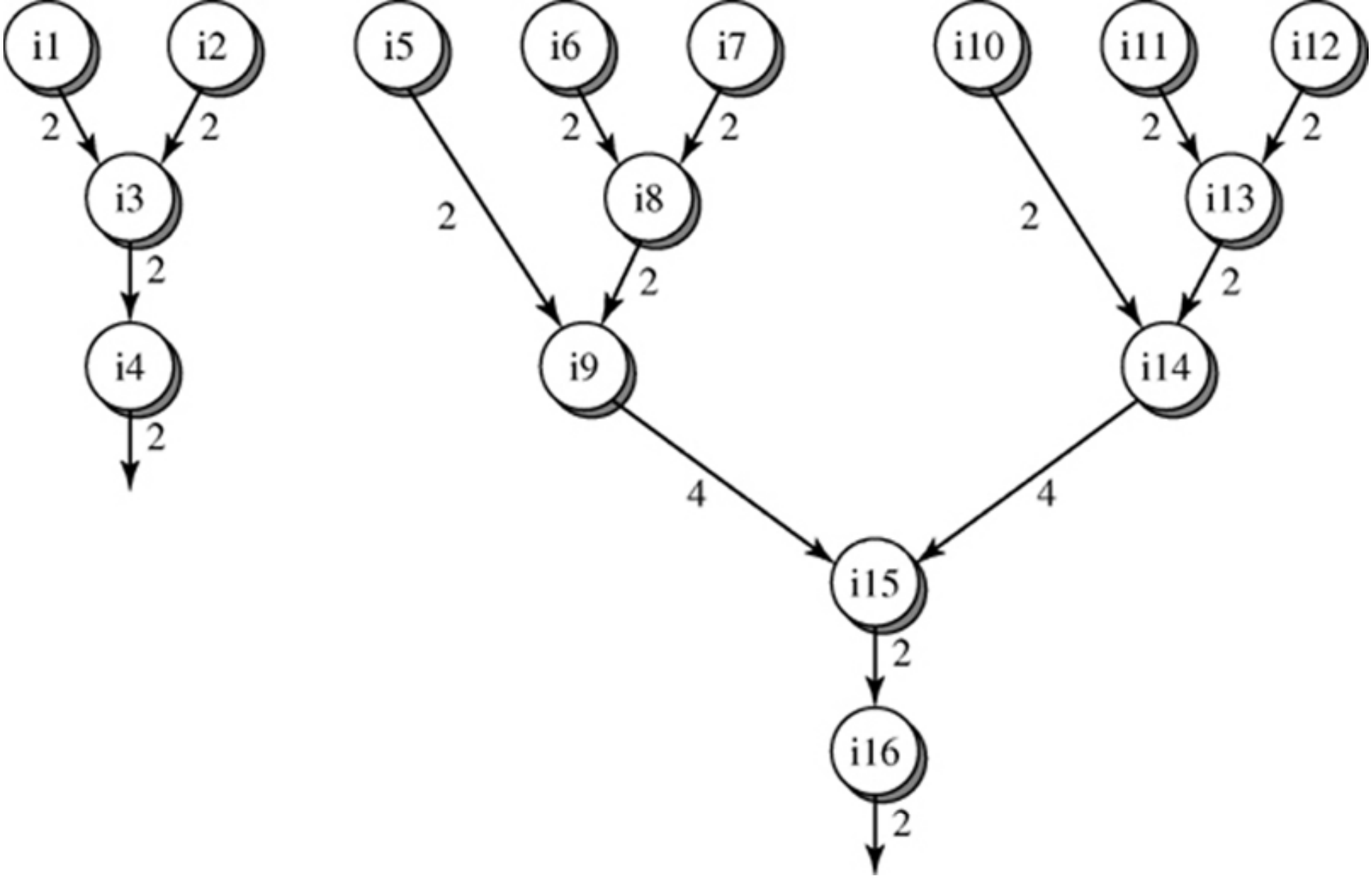


Figure 5.19: Data Flow Graph of the Code Fragment in Figure 5.18(b).

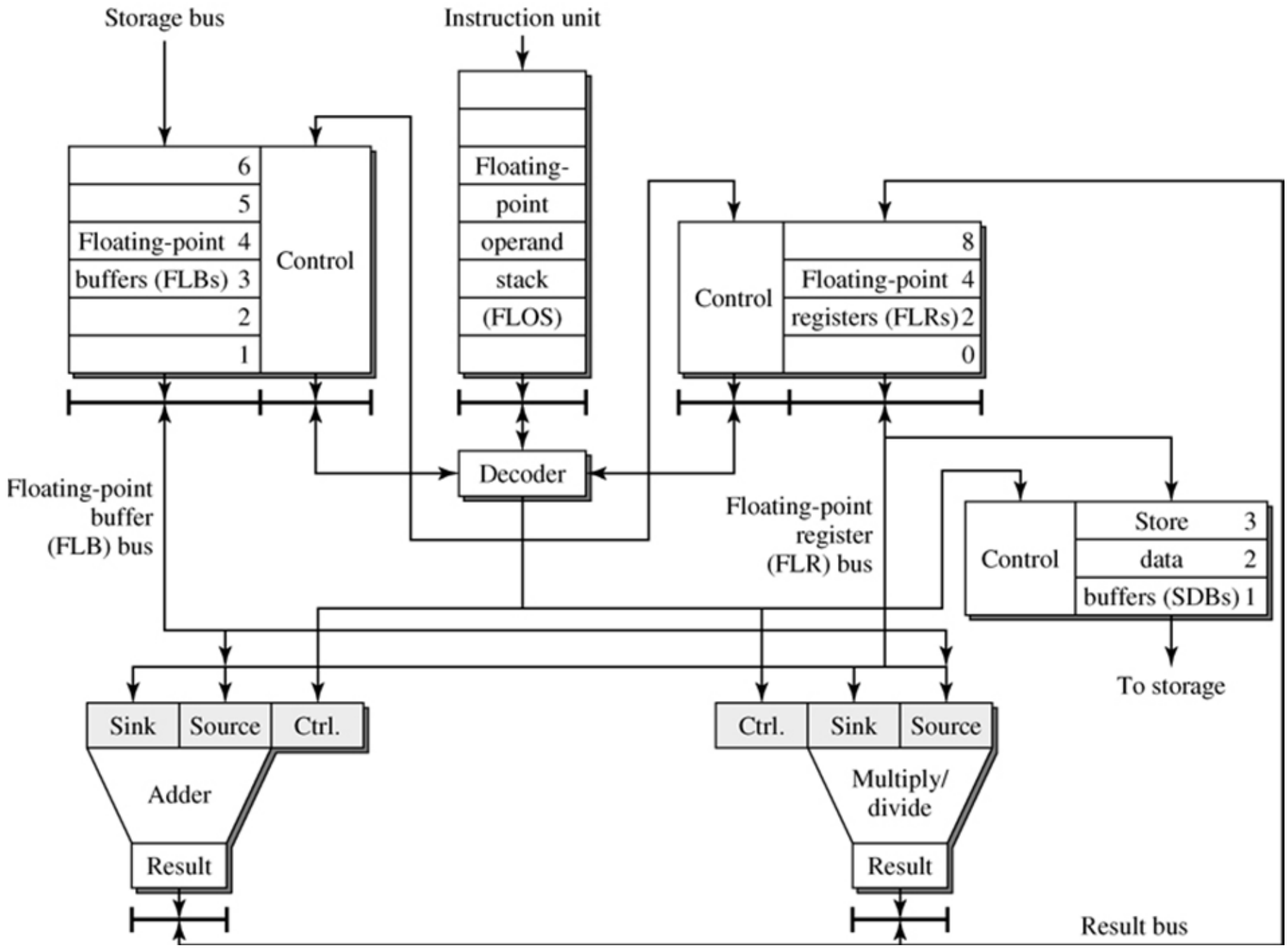


Figure 5.20: The Original Design of the IBM 360 Floating-Point Unit.

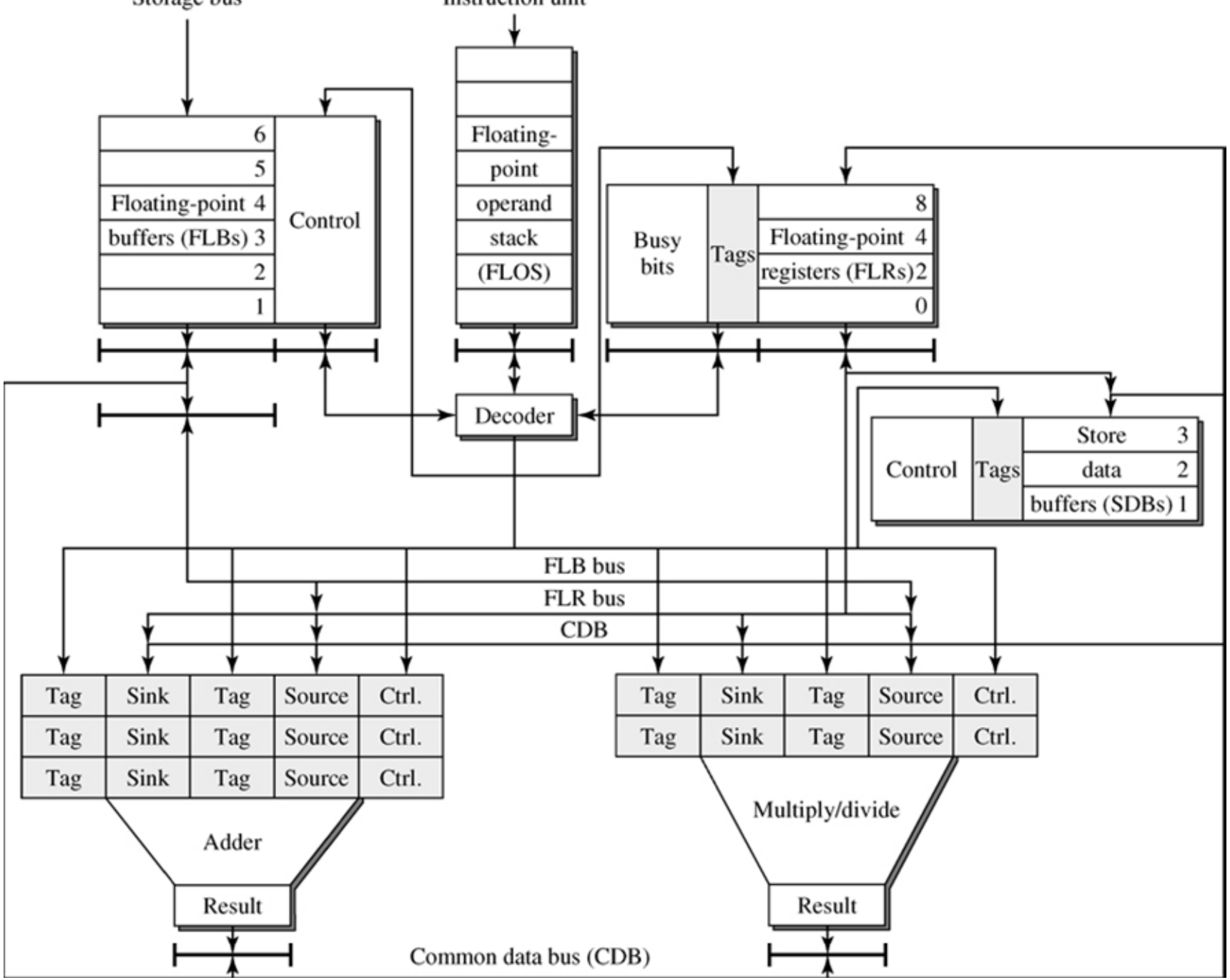


Figure 5.21: The Modified Design of the IBM 360/91 Floating-Point unit with Tomasulo's Algorithm.

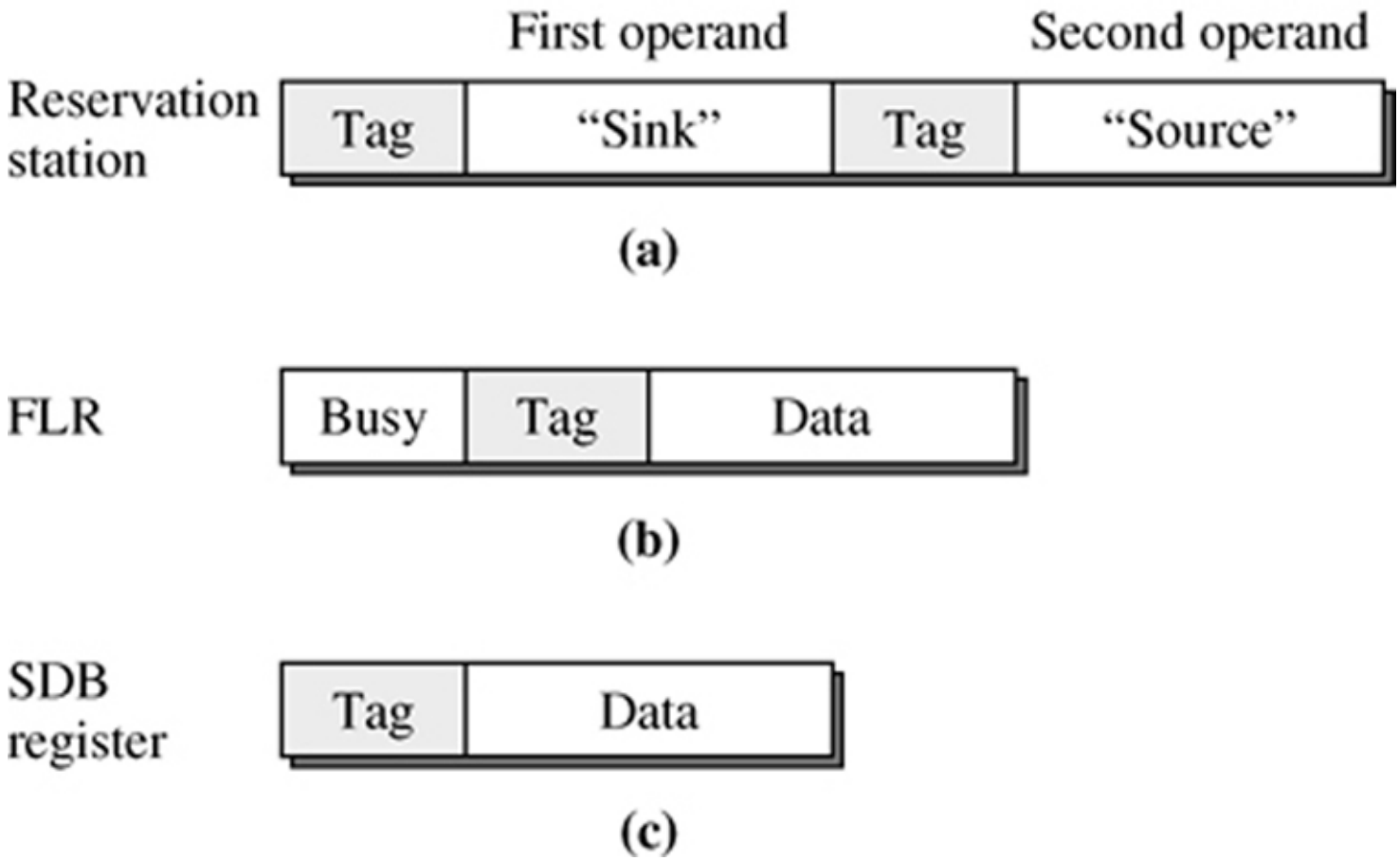


Figure 5.22: The Use of Tag Fields in (a) A Reservation Station, (b) A FLR, and (c) A SDB Register.

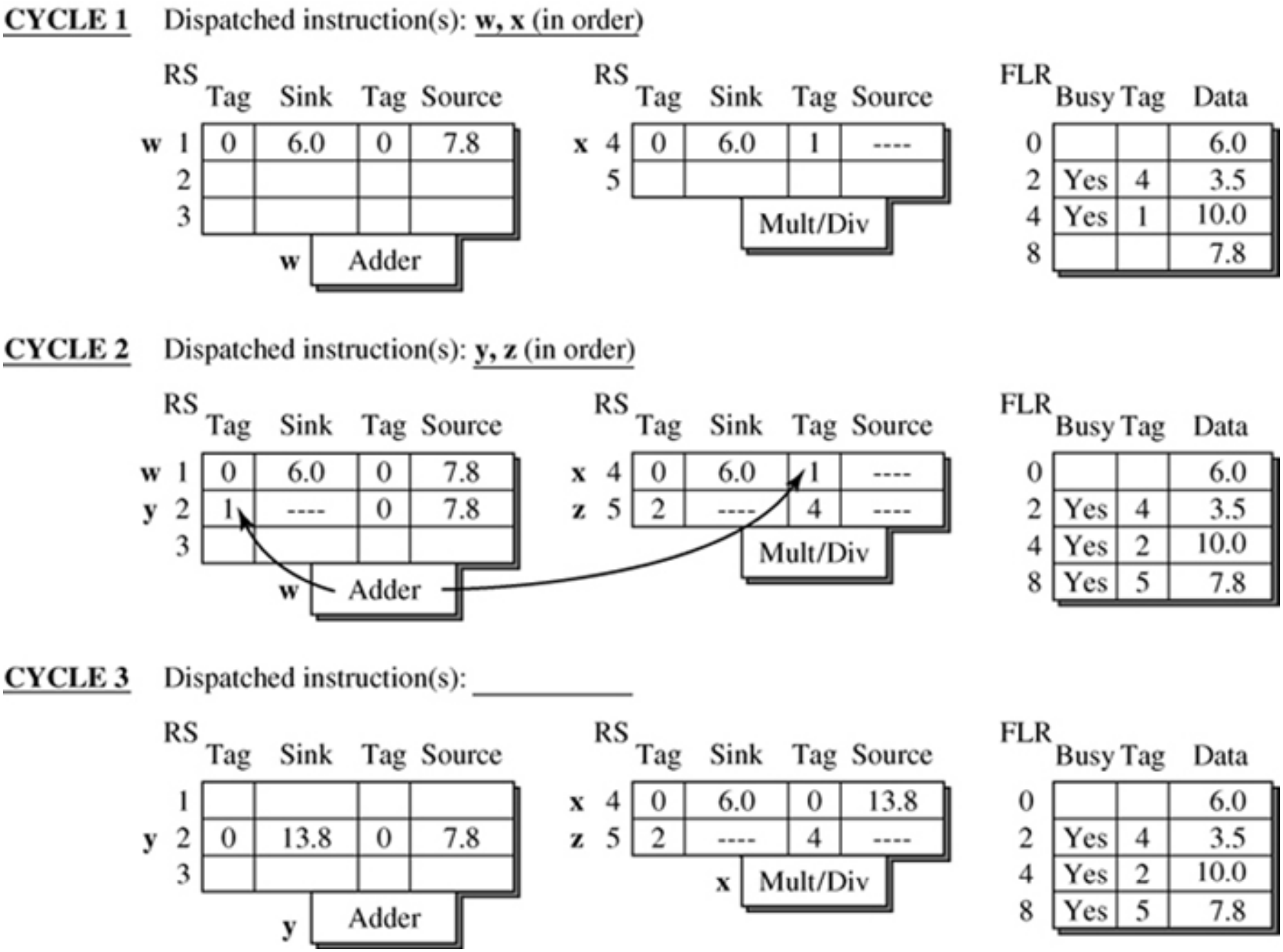
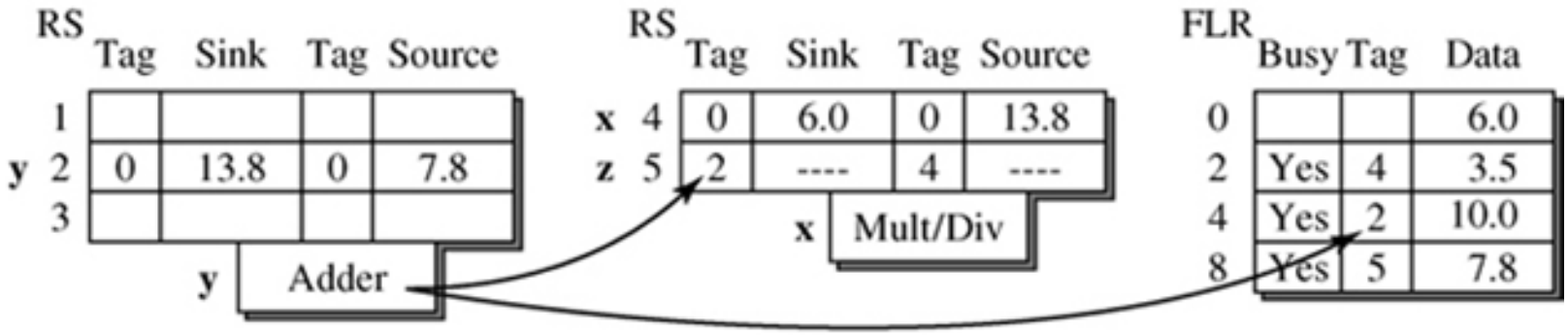
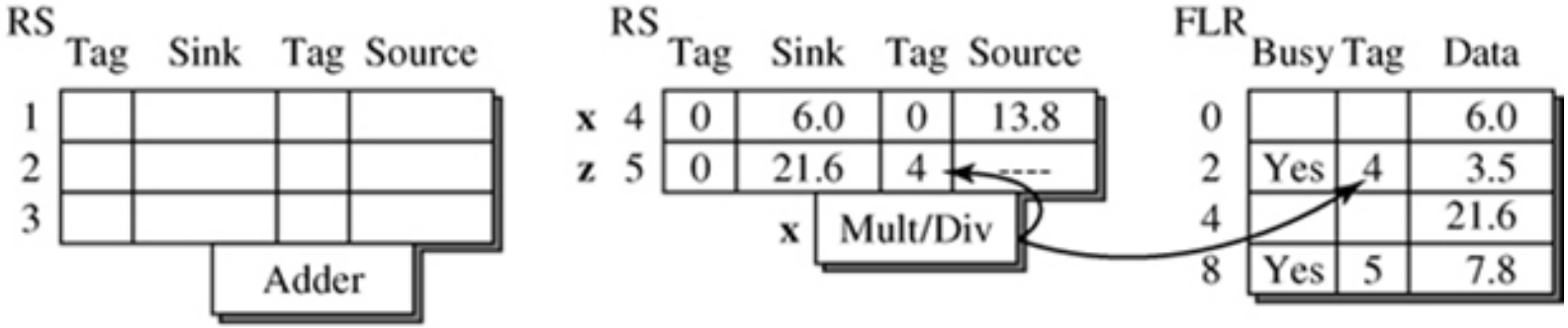


Figure 5.23: Illustration of Tomasulo's Algorithm on an Example Instruction Sequence (Part 1).

CYCLE 4 Dispatched instruction(s): _____



CYCLE 5 Dispatched instruction(s): _____



CYCLE 6 Dispatched instruction(s): _____

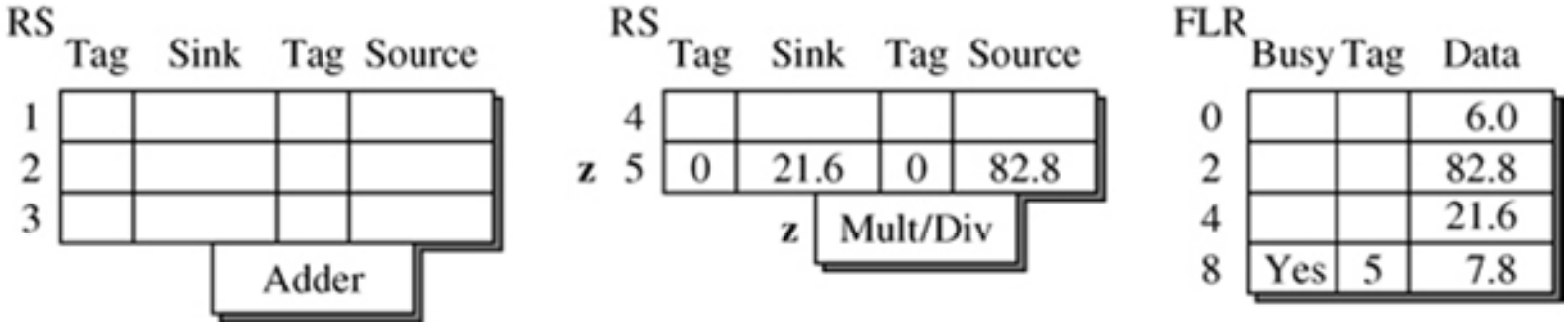


Figure 5.24: Illustration of Tomasulo's Algorithm on an Example Instruction Sequence (Part 2).

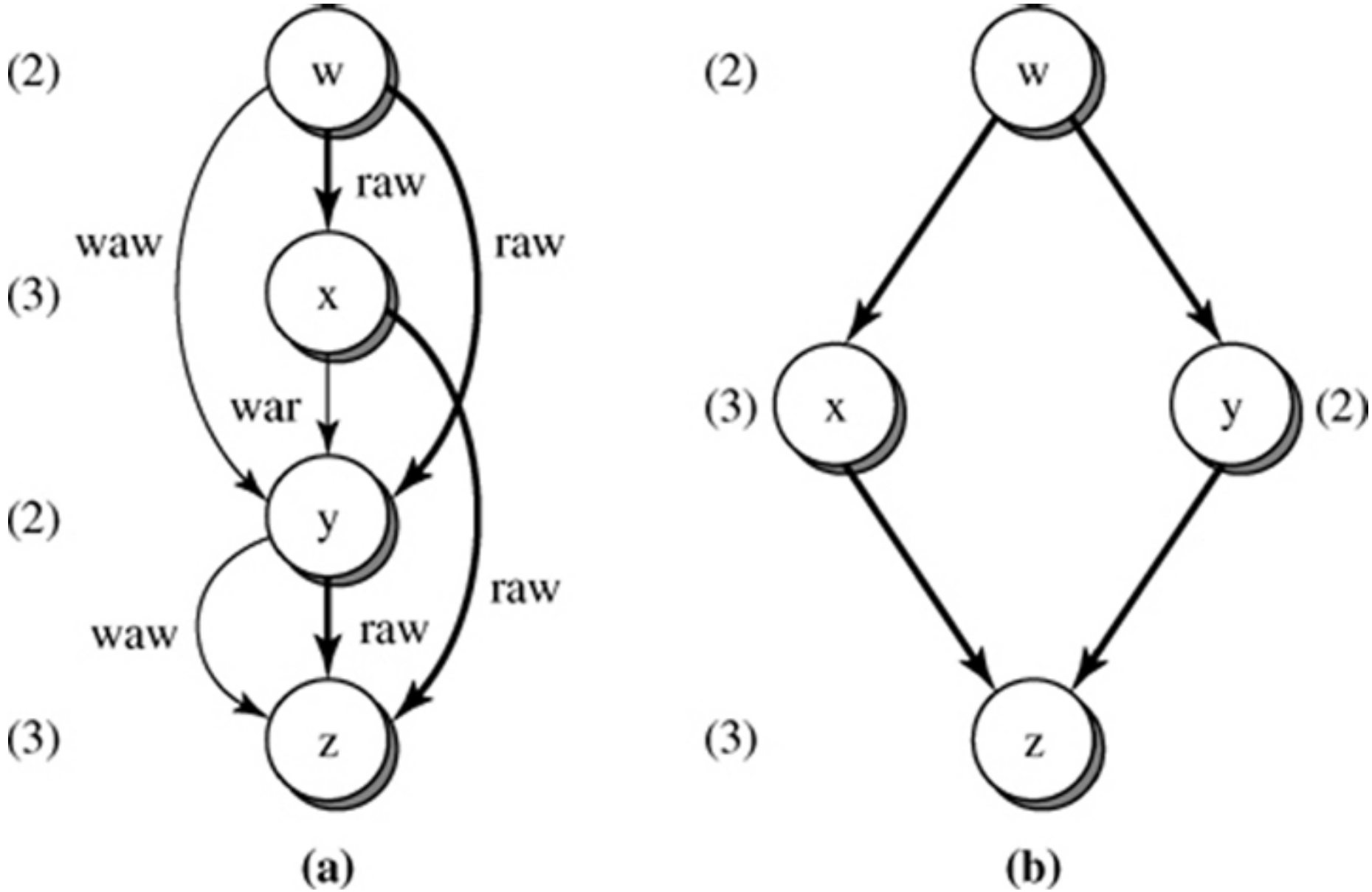


Figure 5.25: Data Flow Graphs of the Example Instruction Sequence: (a) All Data Dependences; (b) True Data Dependences.

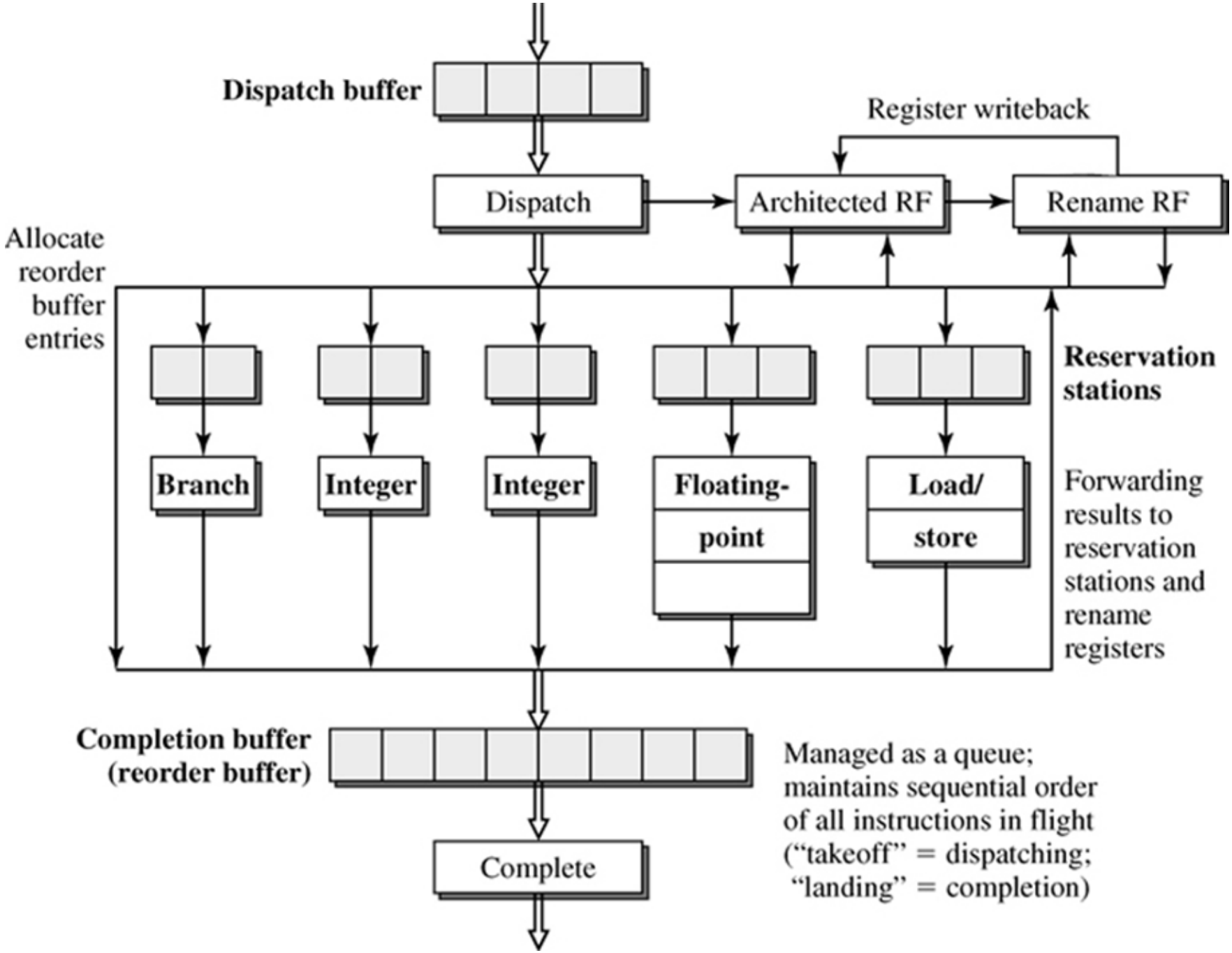


Figure 5.26: Micro-Dataflow Engine for Dynamic Execution.

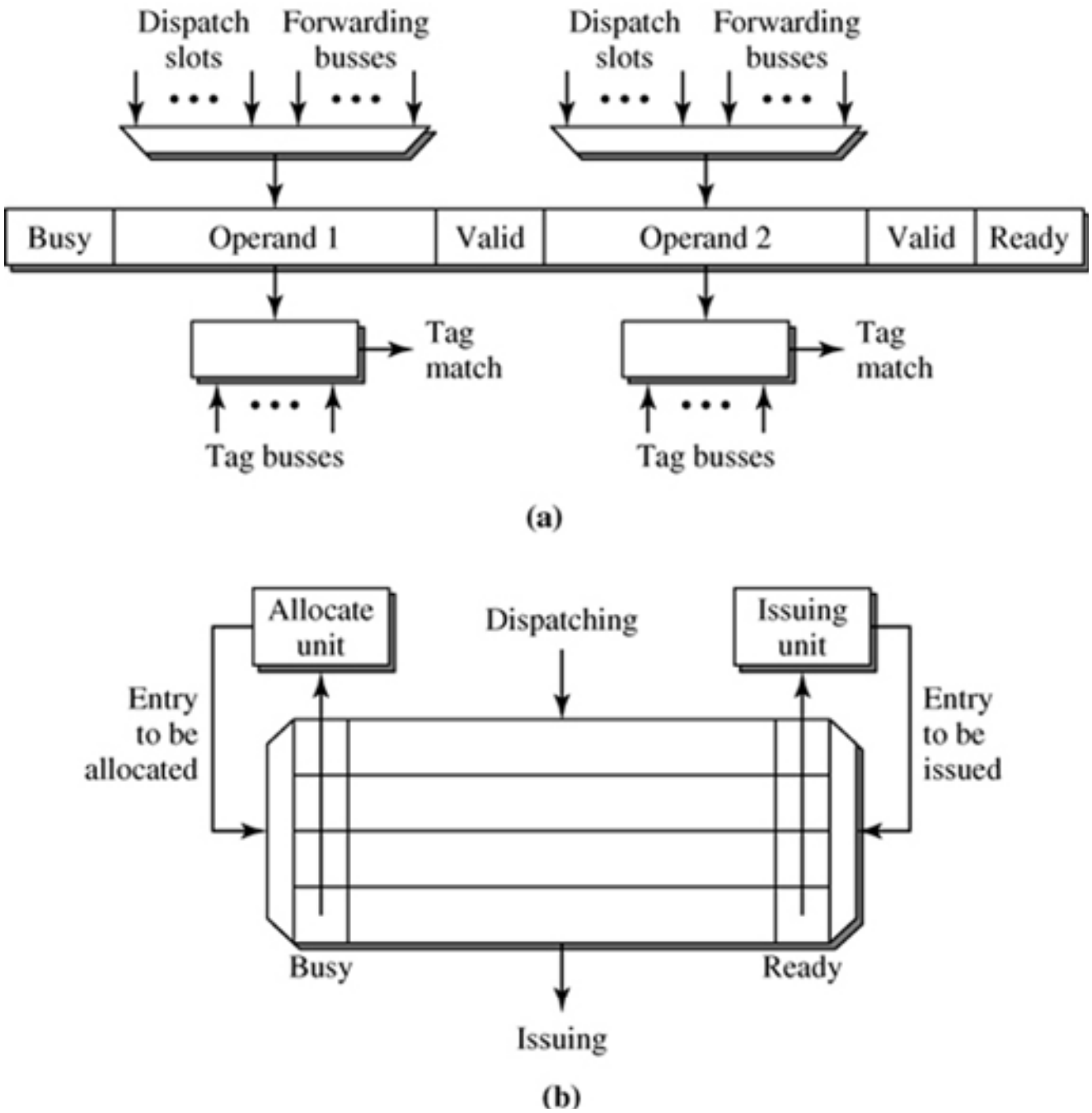
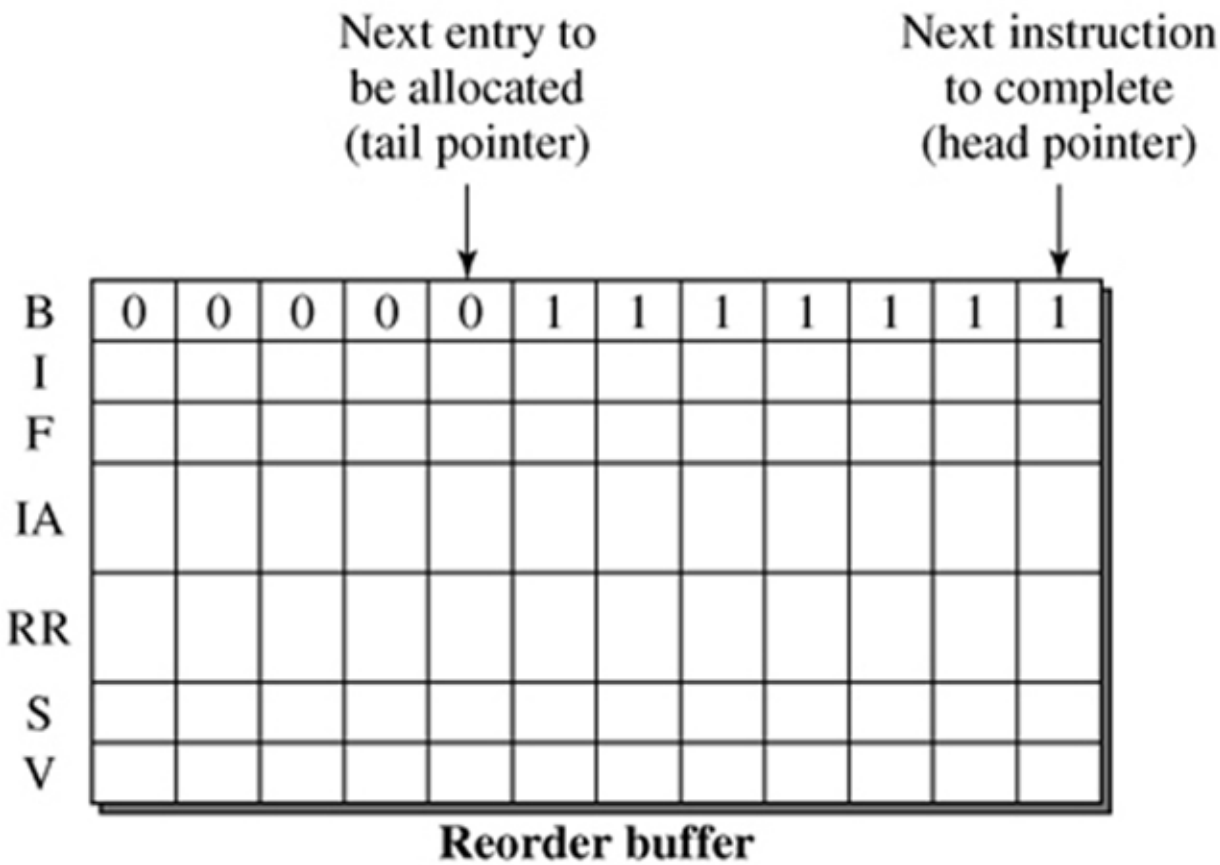


Figure 5.27: Reservation Station Mechanisms: (a) A Reservation Station Entry; (b) Dispatching into and Issuing from a Reservation Station.

Busy	Issued	Finished	Instruction address	Rename register	Speculative	Valid

(a)



(b)

Figure 5.28: (a) Reorder Buffer Entry; (b) Reorder Buffer Organization.

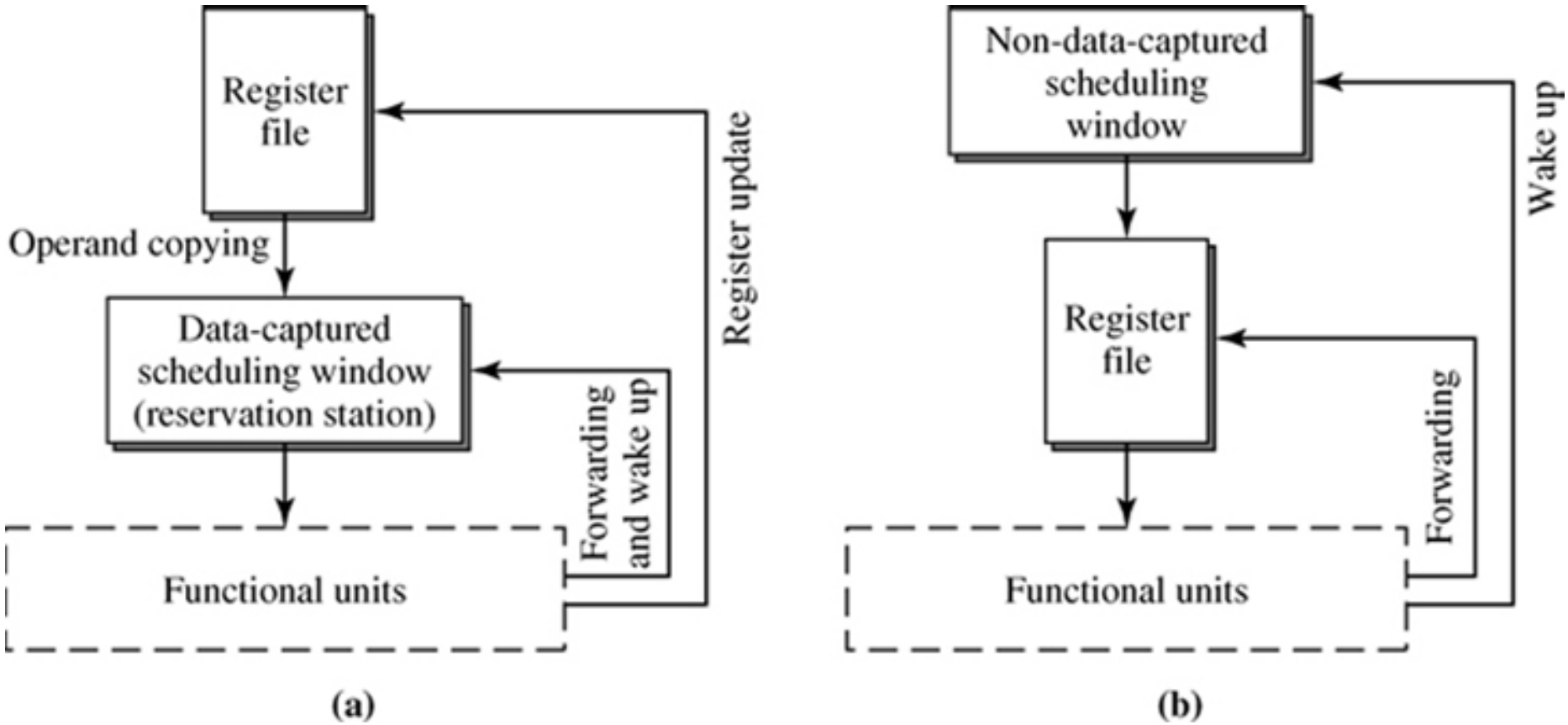


Figure 5.29: Dynamic Instruction Scheduler Design: (a) With Data Capture; (b) Without Data Capture.

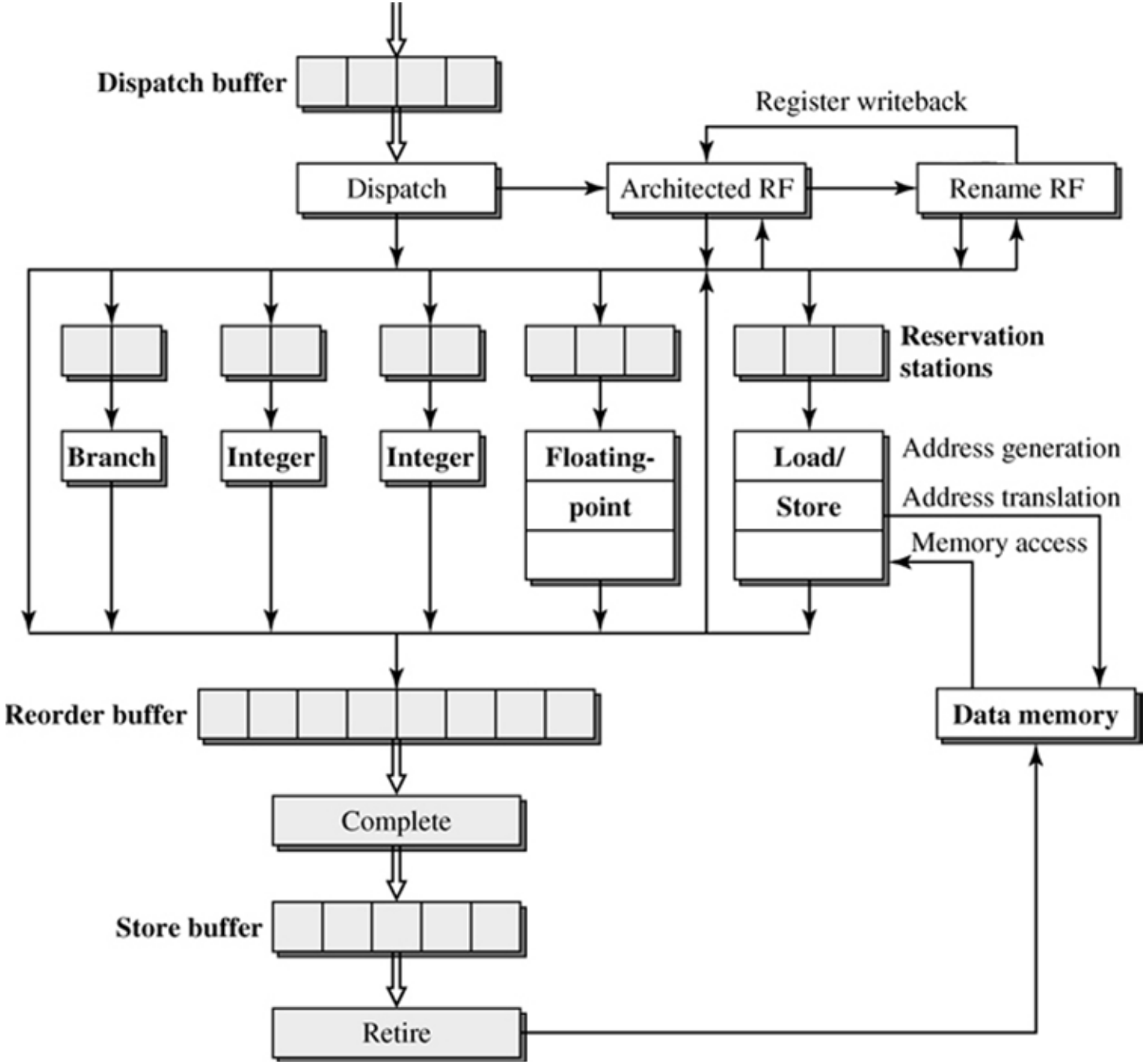


Figure 5.30: Processing of Load/Store Instructions.

```
Y(i) = A * X(i) + Y(i)

      F0 ← LD,a
      R4 ← ADDI,Rx,#512        ;last address

Loop:
      F2 ← LD,0(Rx)            ;load X(i)
      F2 ← MULTD,F0,F2         ;A*X(i)
      F4 ← LD,0(Ry)            ;load Y(i)
      F4 ← ADDD,F2,F4          ;A*X(i)+Y(i)
      0(Ry) ← SD,F4            ;store into Y(i)
      Rx ← ADDI,Rx,#8          ;inc. index to X
      Ry ← ADDI,Ry,#8          ;inc. index to Y
      R20 ← SUB,R4,Rx          ;compute bound
      BNZ,R20,Loop             ;check if done
```

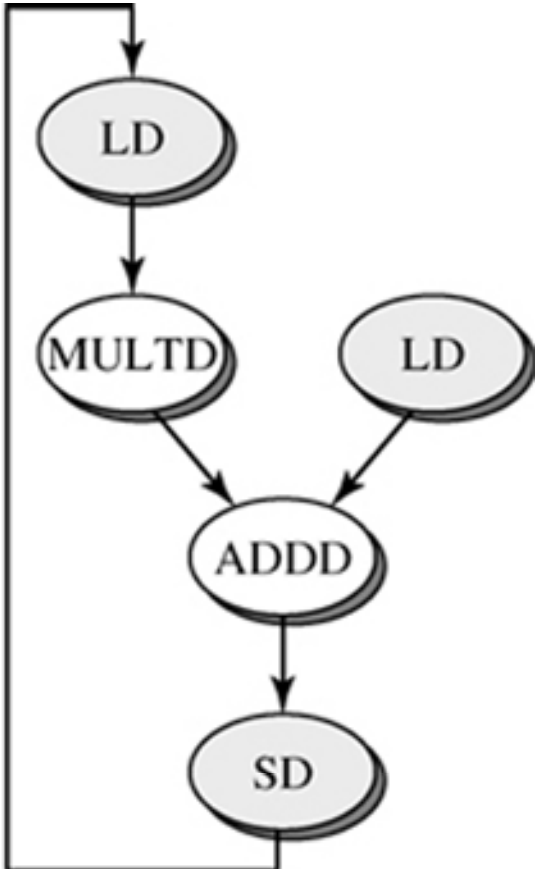


Figure 5.31: The DAXPY Example.

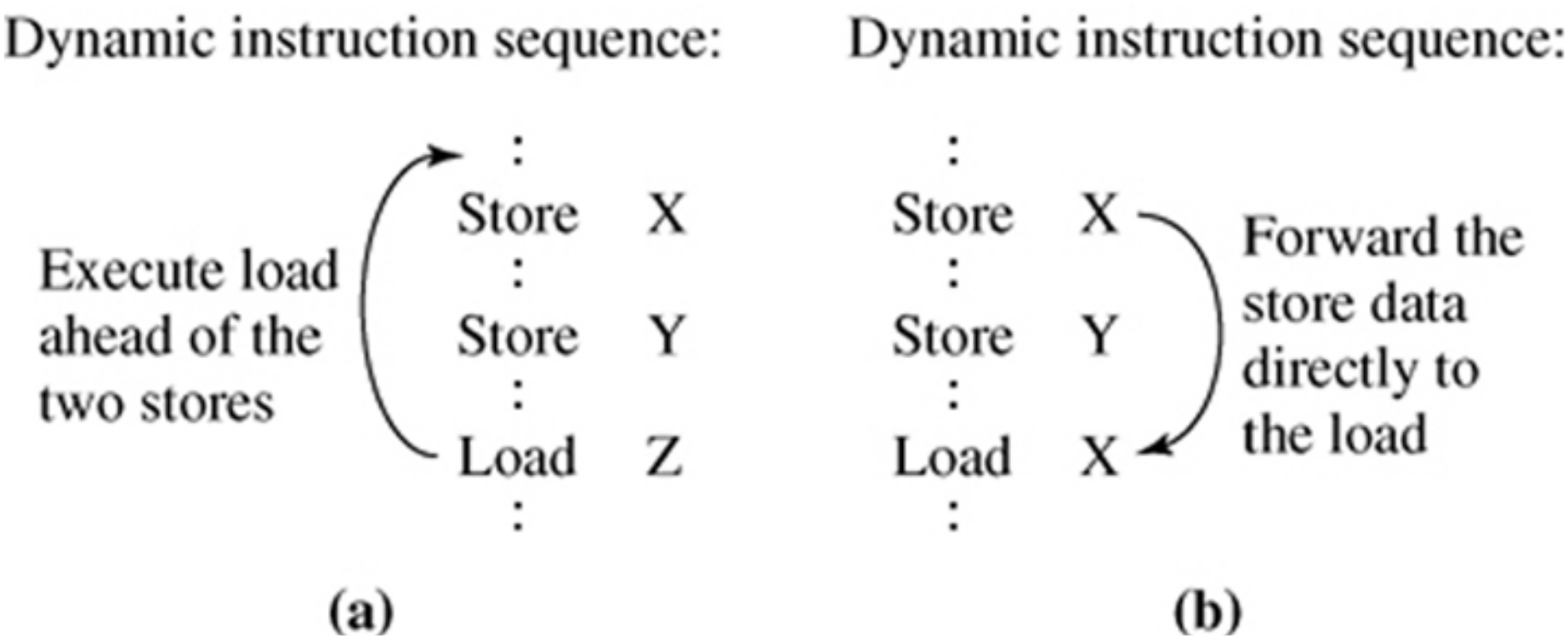


Figure 5.32: Early Execution of Load Instructions: (a) Load Bypassing; (b) Load Forwarding.

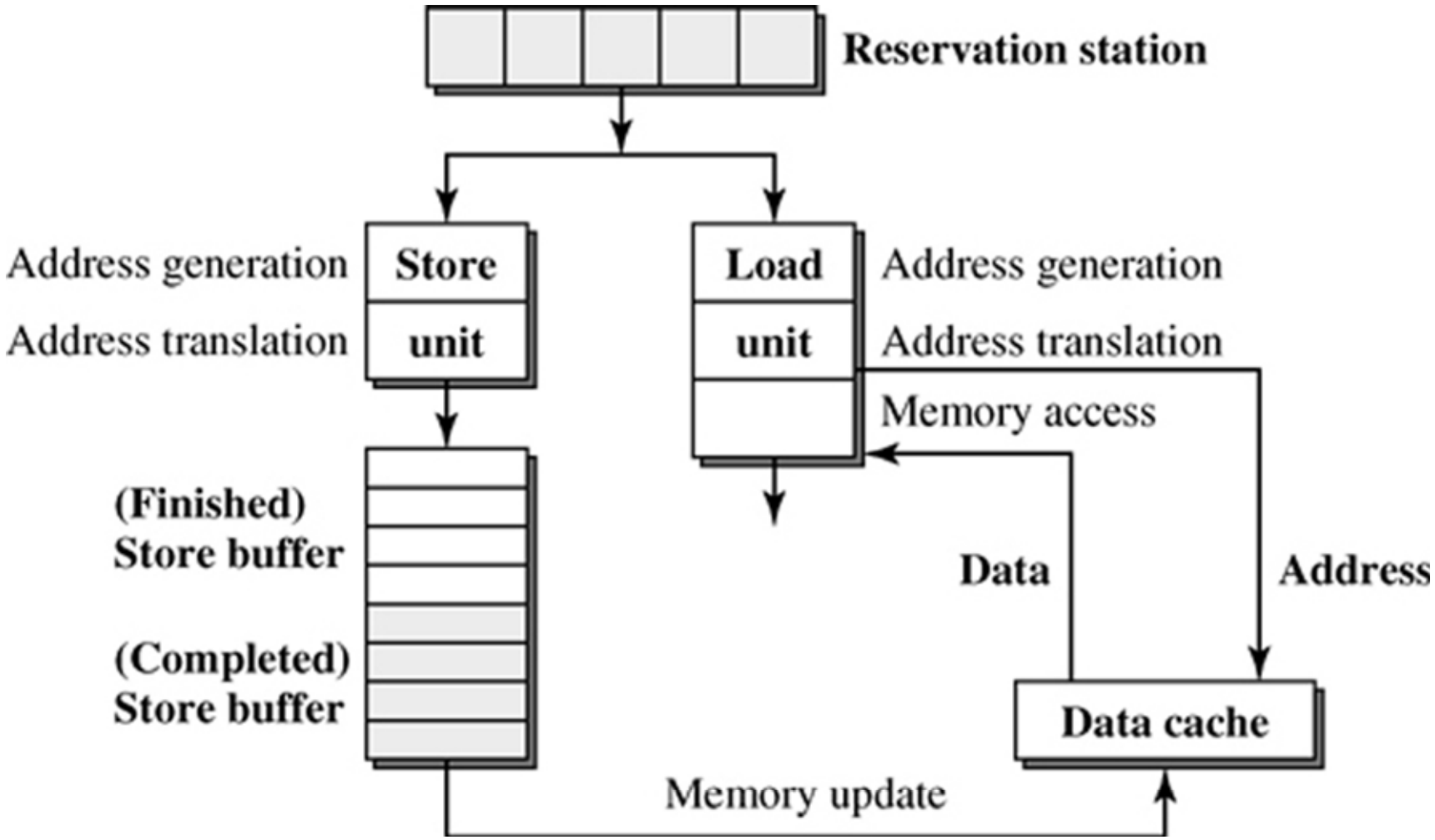


Figure 5.33: Mechanisms for Load/Store Processing: Separate Load and Store Units with In-Order Issuing from a Common Reservation Station.

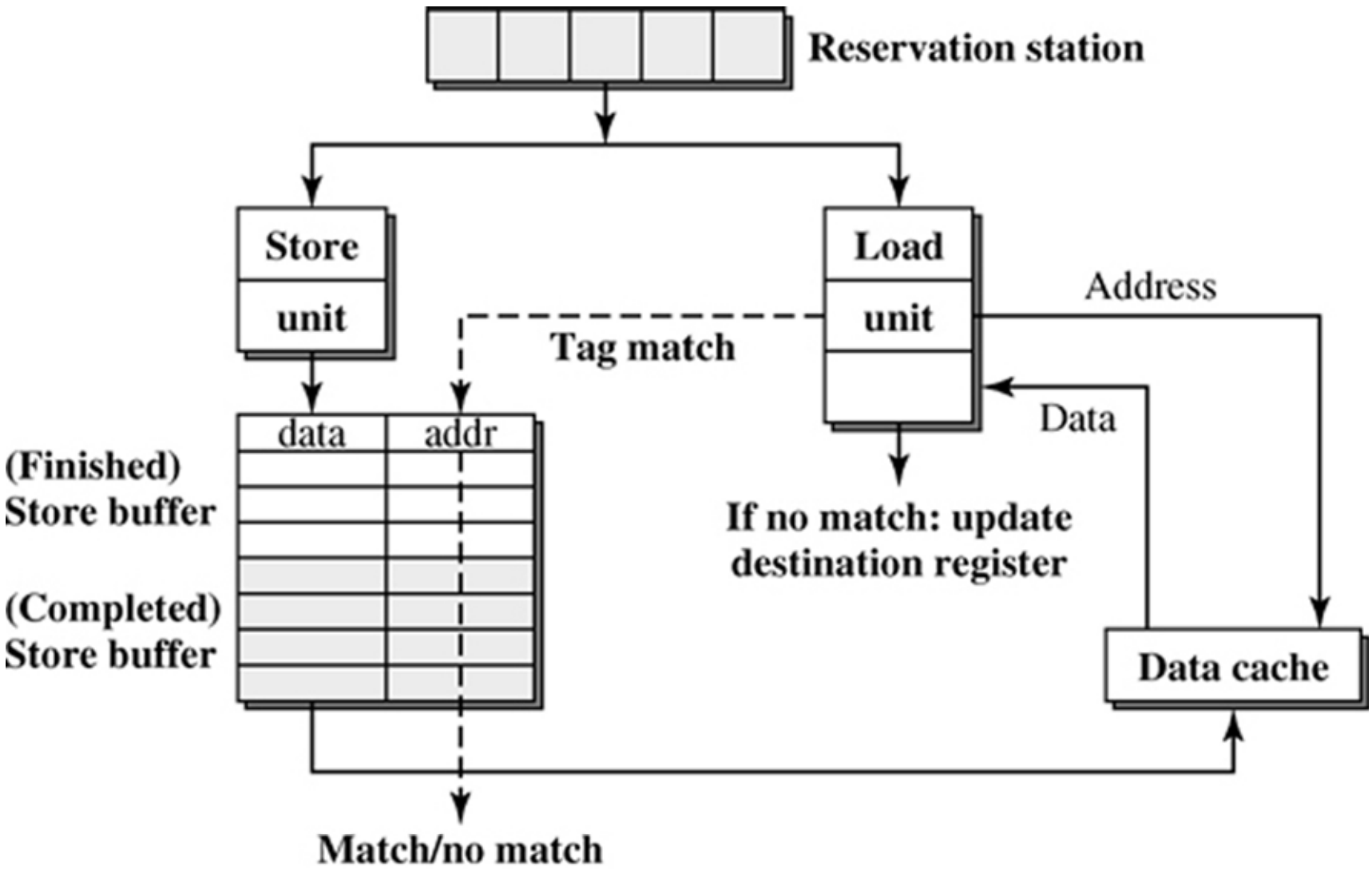


Figure 5.34: Illustration of Load Bypassing.

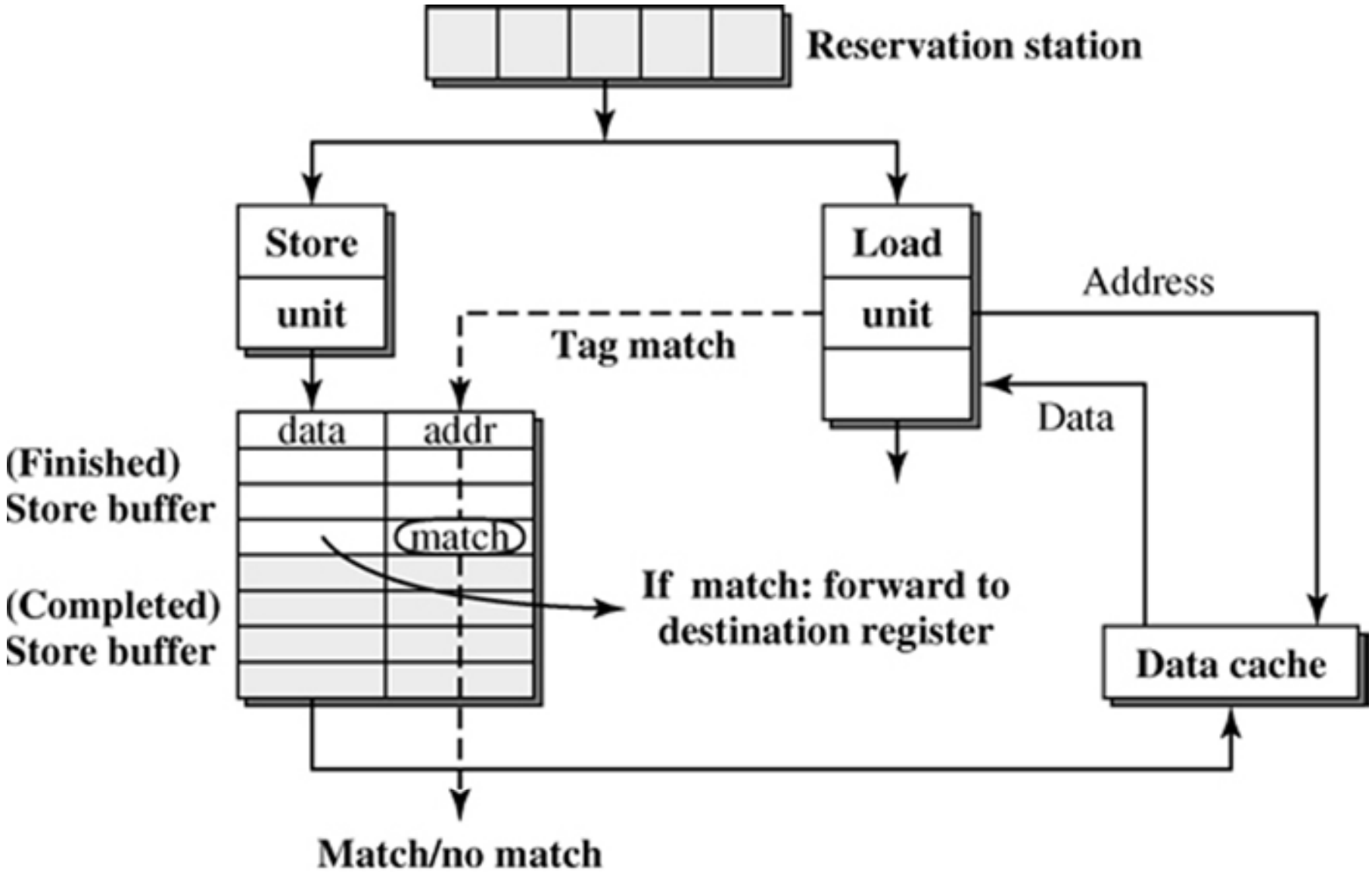


Figure 5.35: Illustration of Load Forwarding.

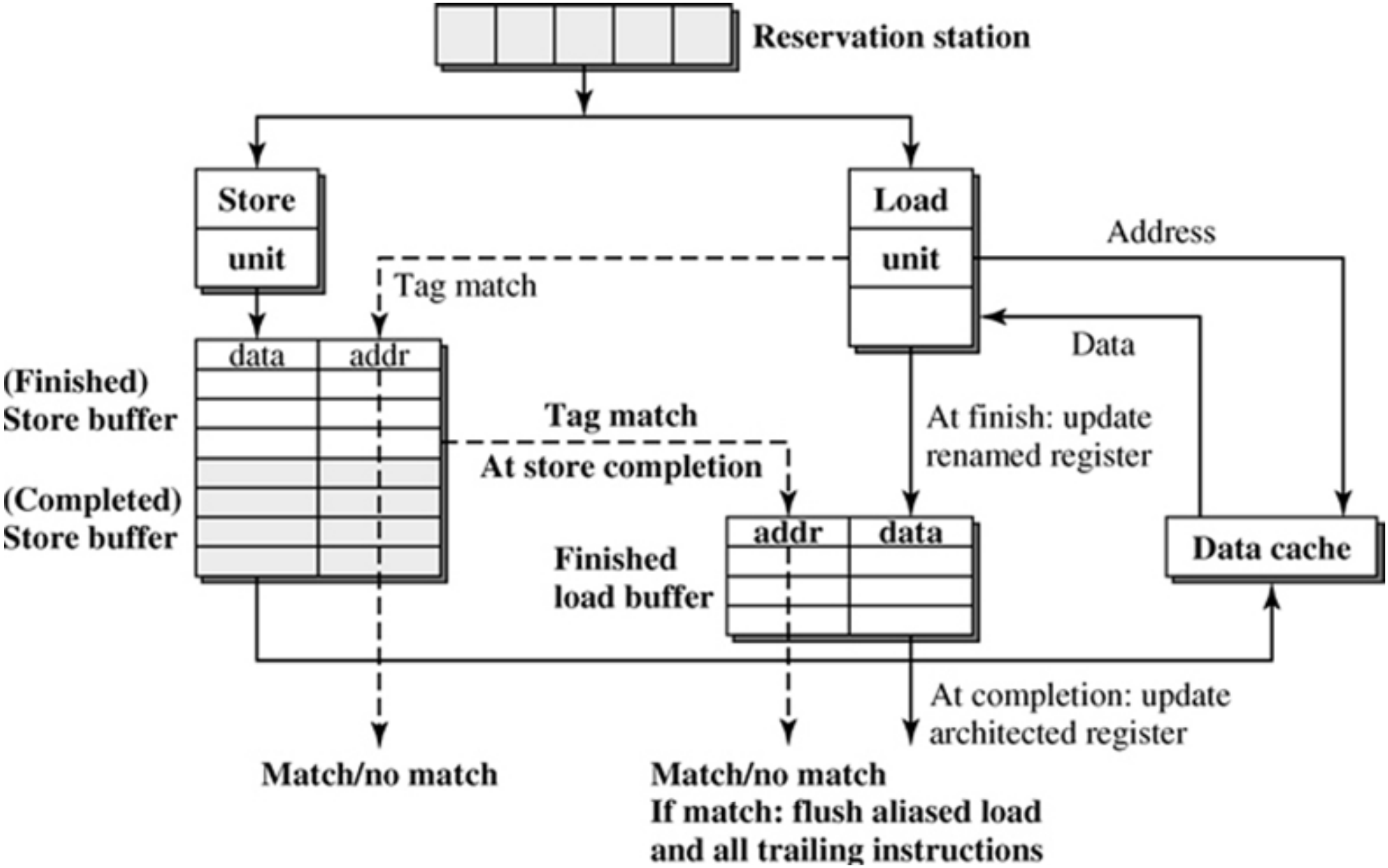


Figure 5.36: Fully Out-of-Order Issuing and Execution of Load and Store Instructions.

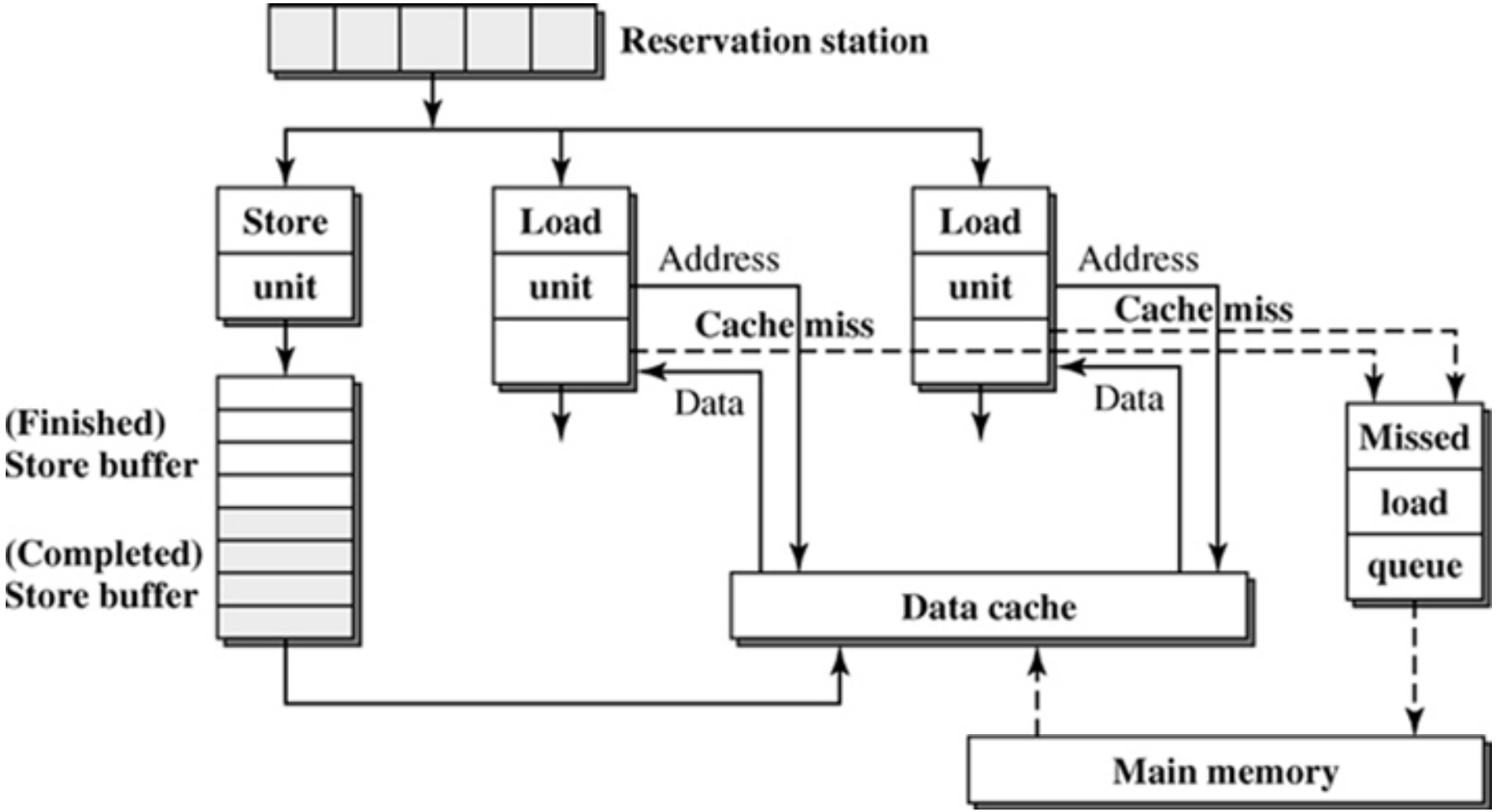


Figure 5.37: Dual-Ported and Nonblocking Data Cache.

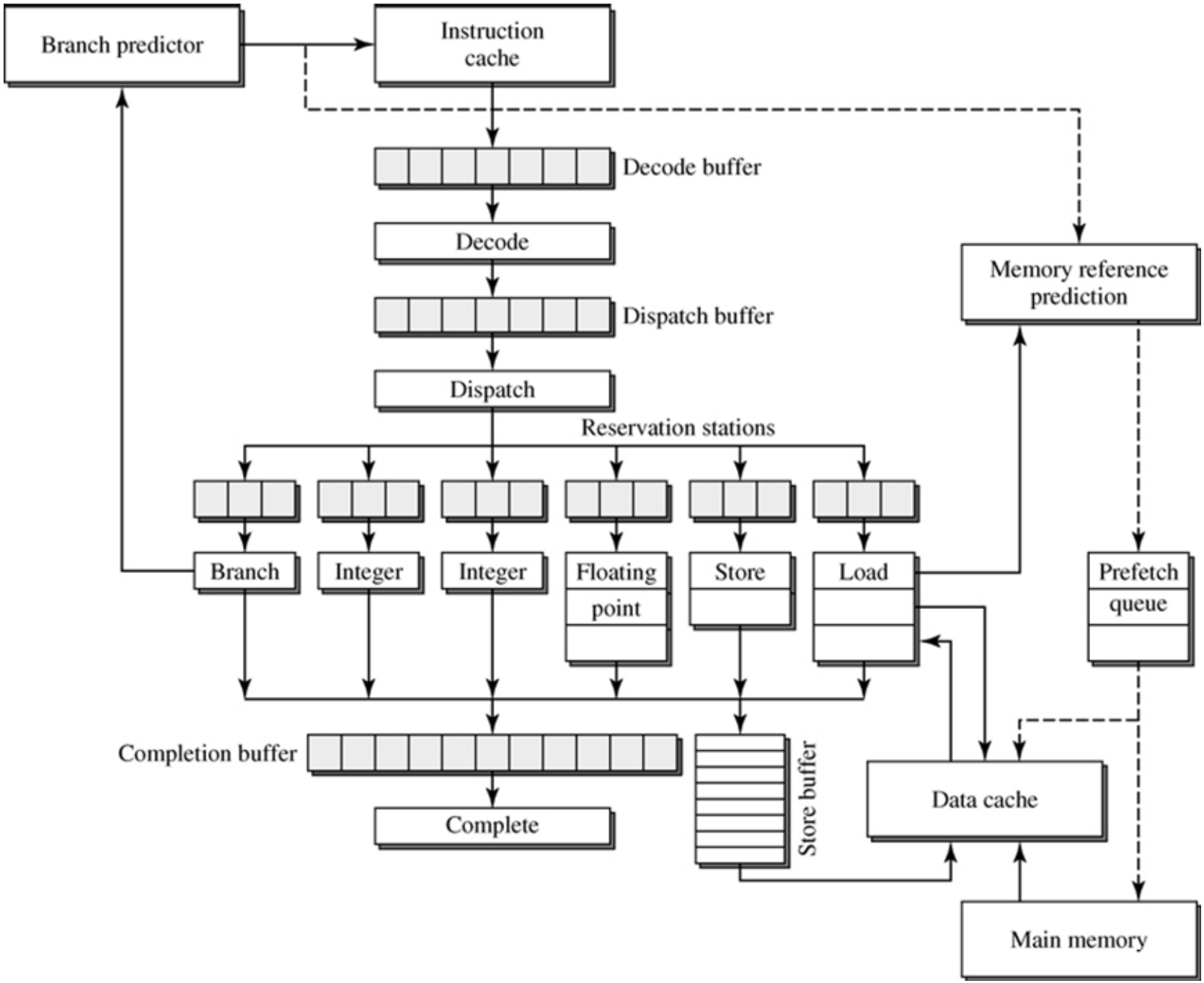


Figure 5.38: Prefetching Data Cache.