Arithmetic Pipelining

Hwang and Briggs

Pipeline Fraction Adder or Floating Point Adder

To understand the operational principles of pipeline computation, we illustrate the design of a pipeline floating-point adder in Figure 3.2. This pipeline is linearly constructed with four functional stages. The inputs to this pipeline are two normalized floating-point numbers:

$$A = a \times 2^p$$
$$B = b \times 2^q \tag{3.3}$$

where a and b are two fractions and p and q are their exponents, respectively. For simplicity, base 2 is assumed. Our purpose is to compute the sum

$$C = A + B = c \times 2^r = d \times 2^s \tag{3.4}$$

where $r = \max(p, q)$ and $0.5 \leq d < 1$.

Operations performed in 4 pipelined stages are:

1. Compare the two exponents p and q to reveal the larger exponent $r = \max(p, q)$ and to determine their difference $t = |p - q|$.
2. Shift right the fraction associated with the smaller exponent by t bits to equalize the two exponents before fraction addition.
3. Add the preshifted fraction with the other fraction to produce the intermediate sum fraction c, where $0 \leq c < 2$.
4. Count the number of leading zeros, say u, in fraction c and shift left c by u bits to produce the normalized fraction sum $d = c \times 2^u$, with a leading bit 1. Update the large exponent s by subtracting $s = r - u$ to produce the output exponent.

Time Delays per Stage Speedup

The comparator, selector, shifters, adders, and counter in this pipeline can all be implemented with combinational logic circuits. Detailed logic design of these boxes can be found in the book by Hwang (1979). Suppose the time delays of the four stages are $\tau_1 = 60$ ns, $\tau_2 = 50$ ns, $\tau_3 = 90$ ns, and $\tau_4 = 80$ ns and the interface latch has a delay of $\tau_l = 10$ ns. The cycle time of this pipeline can be chosen to be at least $\tau = 90 + 10 = 100$ ns (Eq. 3.1). This means that the clock frequency of the pipeline can be set to $f = 1/\tau = 1/100 = 10$ MHz. If one uses a non-pipeline floating-point adder, the total time delay will be $\tau_1 + \tau_2 + \tau_3 + \tau_4 =$ 280 ns. In this case, the pipeline adder has a speedup of $280/100 = 2.8$ over the non-pipeline adder design. If uniform delays can be achieved in all four stages, say 70 ns per stage (including the latch delay), then the maximum speedup of $280/70 = 4$ can be achieved.

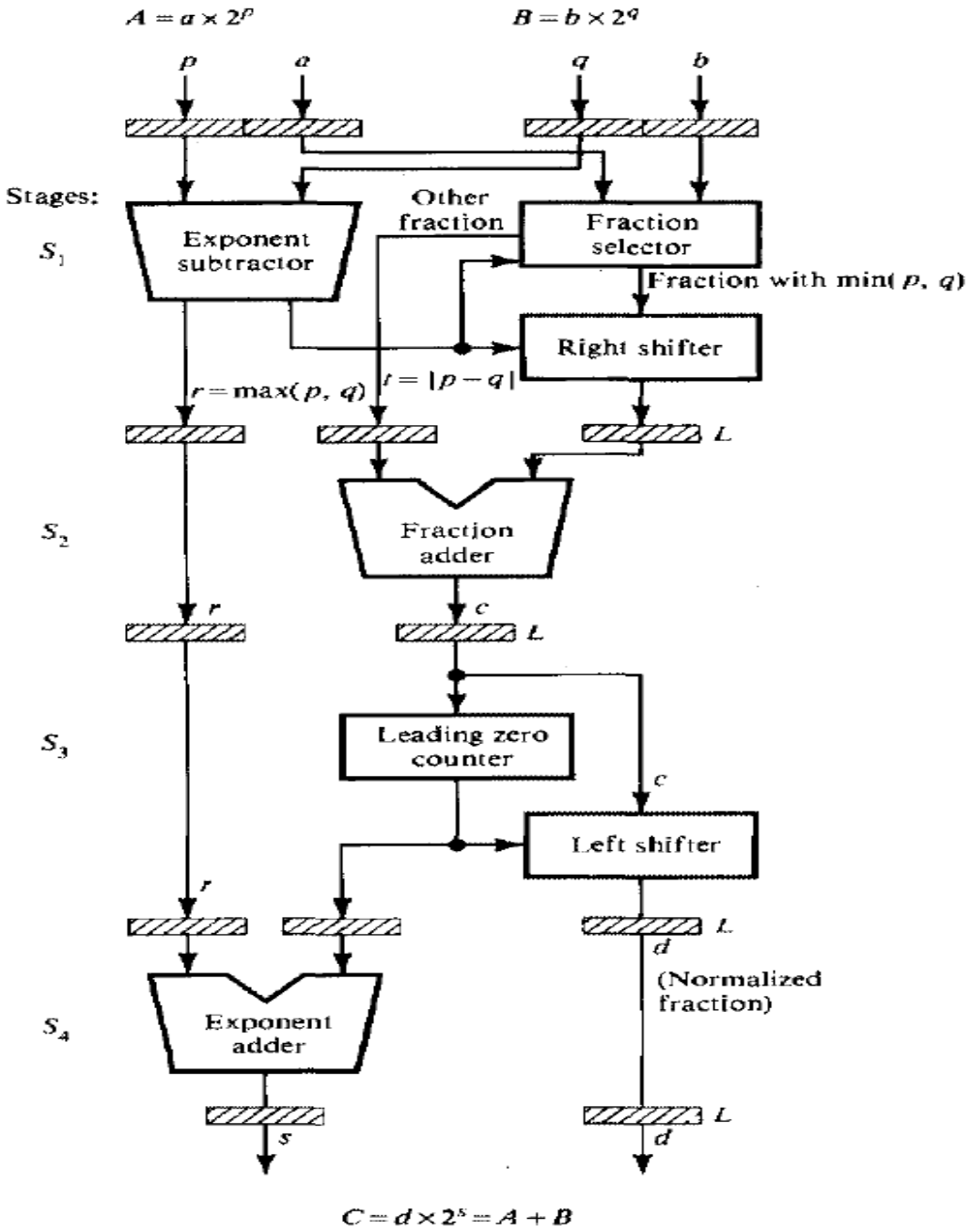


Figure 3.2 A pipelined floating-point adder with four processing stages.

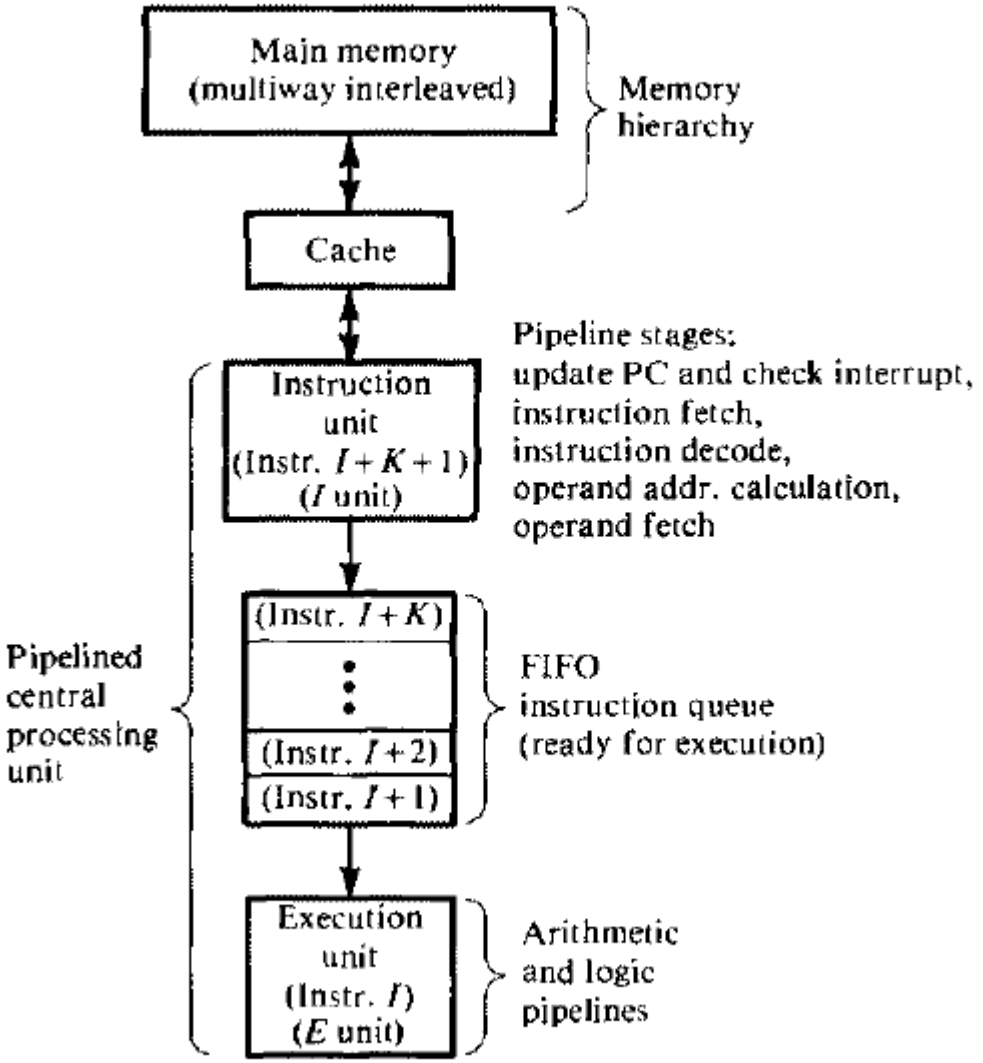


Figure 3.3 The pipelined structure of a typical central processing unit.

Description about the figure in previous slide

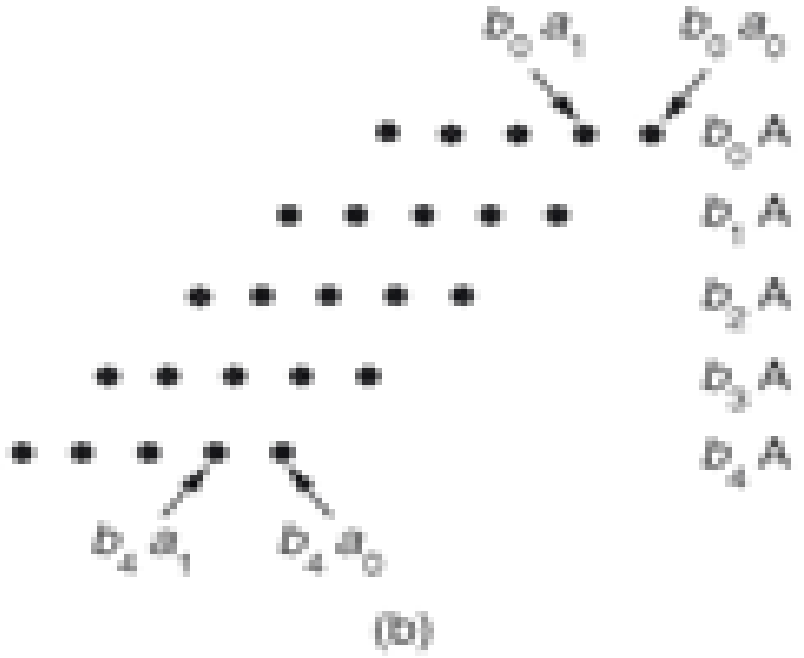
The central processing unit (CPU) of a modern digital computer can generally be partitioned into three sections: the *instruction unit*, the *instruction queue*, and the *execution unit*. From the operational point of view, all three units are pipelined, as illustrated in Figure 3.3. Programs and data reside in the main memory, which usually consists of interleaved memory modules. The cache is a faster storage of copies of programs and data which are ready for execution. The cache is used to close up the speed gap between main memory and the CPU.

Description about the figure
Continued from previous slide

The instruction unit consists of pipeline stages for instruction fetch, instruction decode, operand address calculation, and operand fetches (if needed). The instruction queue is a first-in, first-out (FIFO) storage area for decoded instructions and fetched operands. The execution unit may contain multiple functional pipelines for arithmetic logic functions. While the instruction unit is fetching instruction $I + K + 1$, the instruction queue holds instructions $I + 1$, $I + 2, \ldots, I + K$, and the execution unit executes instruction I. In this sense, the CPU is a good

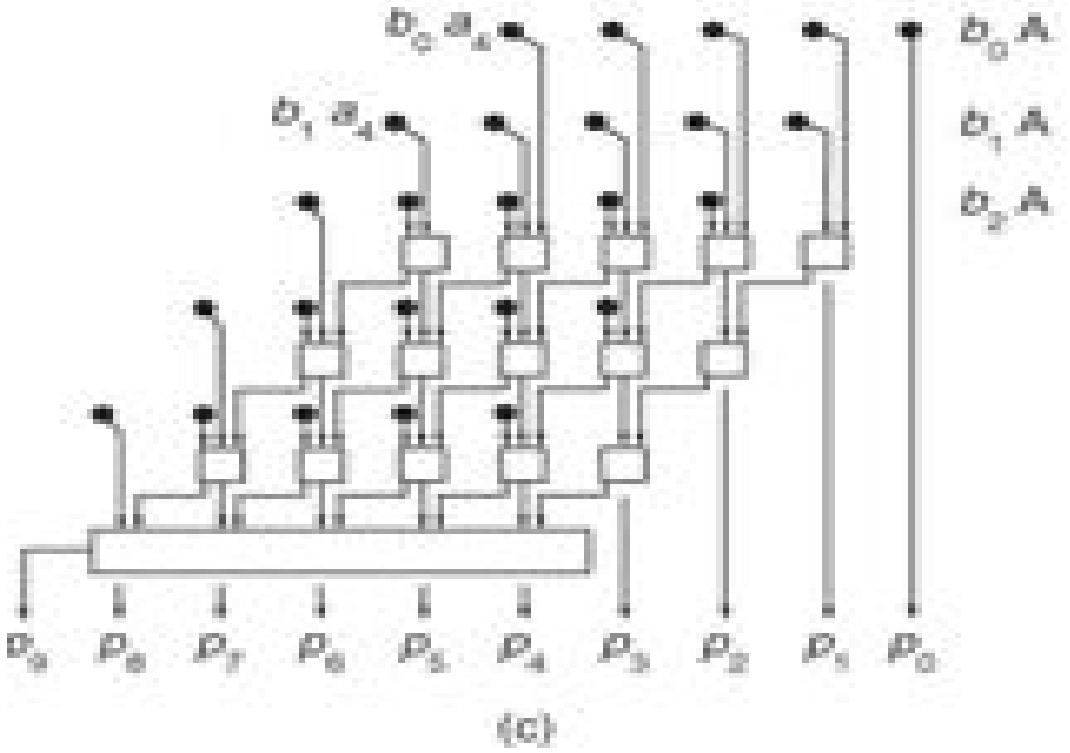
Pipeline Multiplier using Carry Save Adder(CSA)

- The 5-bit number in A is multiplied by $b_4b_3b_2b_1b_0$



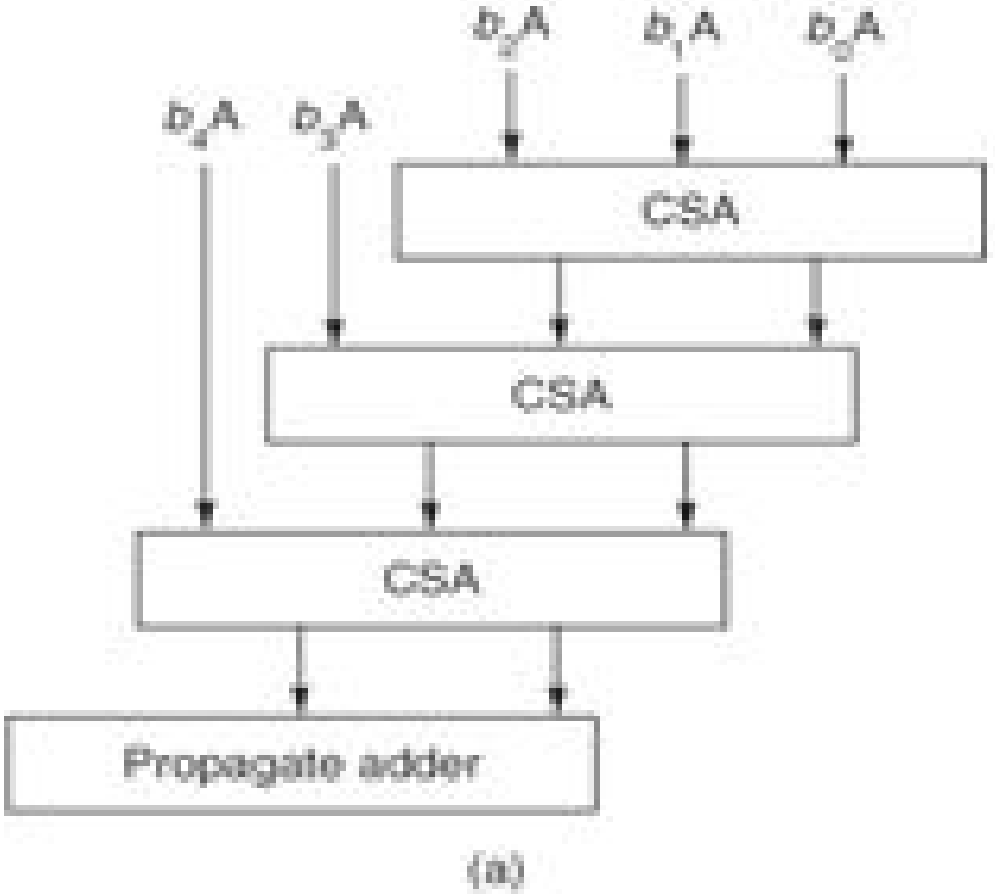
(b) shows the inputs to the adders to do partial summation in regular multiplication

Pipeline Multiplier using Carry Save Adder(CSA) contd…..



Showing all the full adders. 3 inputs taken by full adders and 2 outputs given (reduces carry propagation delay in every stage of partial addition).Last stage is a carry-look-ahead adder(/CPA)(5 bit numbers being added at each stage with 5 full adders**-as in previous slide of Array Multiplication)**

Pipeline Multiplier using Carry Save Adder(CSA) contd….



An array multiplier. The 5-bit number in A is multiplied by $b_4b_3b_2b_1b_0$. (a) shows the block diagram of the previous slide.

Wallace Tree for multiplying 2 numbers of 8 bits each

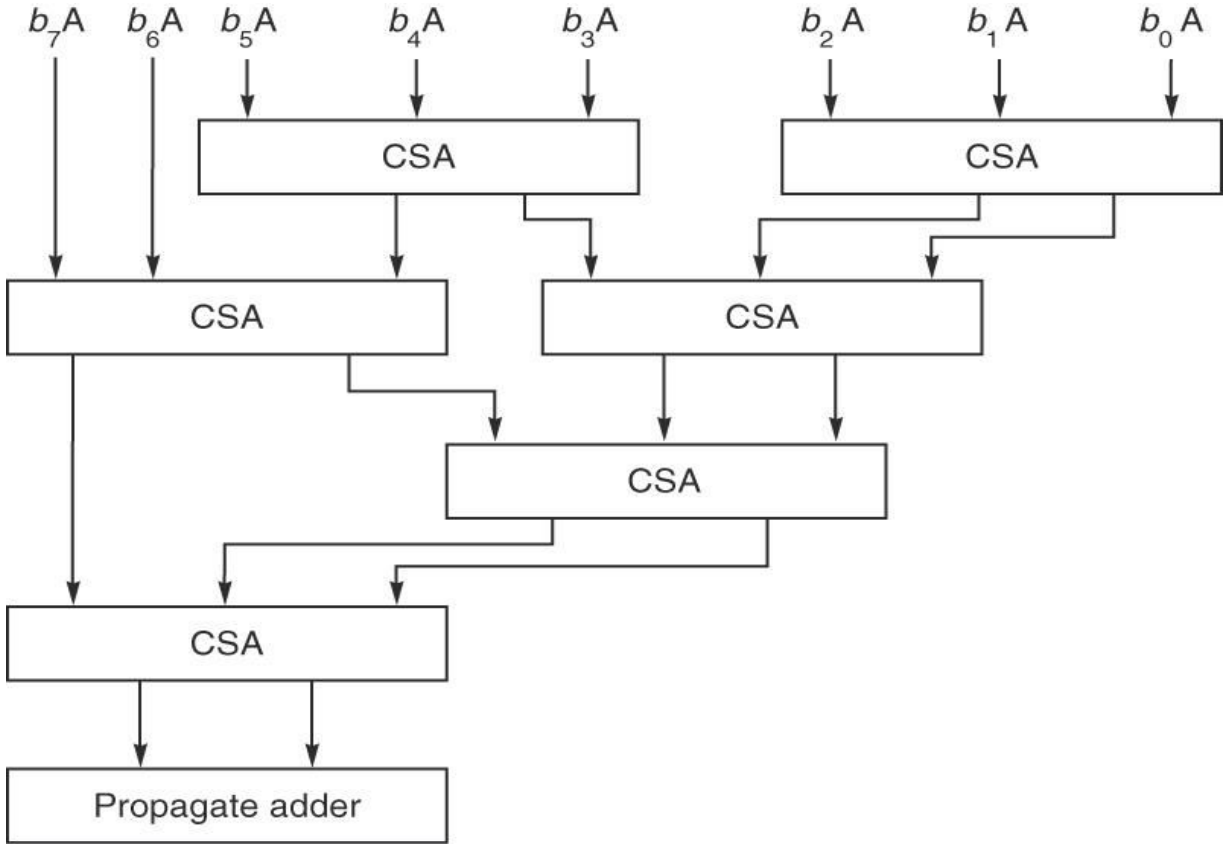


Figure J.30 **Wallace tree multiplier** for multiplying 8bit , 2 numbers A and B. An example of a multiply tree that computes a product in **0(log *n*) steps**.

Practice Problem: Draw Wallace Tree for multiplying i) 5 bit numbers. ii) 6 bit numbers iii) 7 bit numbers

•In Wallace tree architecture

- all the bits of all of the partial products in each column are added together by a set of counters in parallel without propagating any carries.

- Another set of counters then reduces this new matrix and so on, until a two-row matrix is generated.
    - counter used is the 3:2 counter which is a Full Adder..
- The final results are added using usually carry propagate adder.

•The advantage of Wallace tree is speed because the addition of partial products is now O (logN).

If we restrict the CSA tree to adding only multiple single-bit numbers, we have the well-known bit-slice Wallace trees. In general, a v-level CSA tree can add up to $N(v)$ input numbers, where $N(v)$ is evaluated by the following recursive formula:

$$N(v) = \left\lfloor \frac{N(v-1)}{2} \right\rfloor \times 3 + N(v-1) \bmod 2 \qquad \text{with } N(1) = 3 \qquad (3.9)$$

For example, one needs a 10-level CSA tree to add 64 to 94 numbers in one pass through the tree. In other words, a pipeline with 10 stages on the CSA tree is needed to multiply two 64-bit fixed-point numbers in one pass. The floor notation $\lfloor x \rfloor$ refers to the largest integer not greater than x.

3.1.2 Classification of Pipeline Processors

According to the levels of processing, Händler (1977) has proposed the following classification scheme for pipeline processors, as illustrated in Figure 3.4.

Arithmetic pipelining The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats (Figure 3.4*a*). Well-known arithmetic pipeline examples are the four-stage pipes used in Star-100, the eight-stage pipes used in the TI-ASC, the up to 14 pipeline stages used in the Cray-1, and the up to 26 stages per pipe in the Cyber-205. These arithmetic logic pipeline designs will be studied subsequently.

Instruction pipelining The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode, and operand fetch of subsequent instructions (Figure 3.4*b*). This technique is also known as *instruction lookahead*. Almost all high-performance computers are now equipped with instruction-execution pipelines.

Processor pipelining This refers to the pipeline processing of the same data stream by a cascade of processors (Figure 3.4*c*), each of which processes a specific task. The data stream passes the first processor with results stored in a memory block which is also accessible by the second processor. The second processor then passes the refined results to the third, and so on. The pipelining of multiple processors is not yet well accepted as a common practice.

According to pipeline configurations and control strategies, Ramamoorthy and Li (1977) have proposed the following three pipeline classification schemes:

Unifunction vs. multifunction pipelines A pipeline unit with a fixed and dedicated function, such as the floating-point adder in Figure 3.2, is called *unifunctional*. The Cray-1 has 12 unifunctional pipeline units for various scalar, vector, fixed-point, and floating-point operations. A *multifunction* pipe may perform different functions, either at different times or at the same time, by interconnecting different subsets of stages in the pipeline. The TI-ASC has four multifunction pipeline processors, each of which is reconfigurable for a variety of arithmetic logic operations at different times.

Static vs. dynamic pipelines A *static pipeline* may assume only one functional configuration at a time. Static pipelines can be either unifunctional or multifunctional. Pipelining is made possible in static pipes only if instructions of the same type are to be executed continuously. The function performed by a static pipeline should not change frequently. Otherwise, its performance may be very low. A *dynamic pipeline* processor permits several functional configurations to exist simultaneously. In this sense, a dynamic pipeline must be multifunctional. On the other hand, a unifunctional pipe must be static. The dynamic configuration needs much more elaborate control and sequencing mechanisms than those for static pipelines. Most existing computers are equipped with static pipes, either unifunctional or multifunctional.

Scalar vs. vector pipelines Depending on the instruction or data types, pipeline processors can be also classified as scalar pipelines and vector pipelines. A *scalar pipeline* processes a sequence of scalar operands under the control of a DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands. The IBM System/360 Model 91 is a typical example of a machine equipped with scalar pipelines. However, the Model 91 does not have a cache.

Vector pipelines are specially designed to handle vector instructions over vector operands. Computers having vector instructions are often called *vector processors*. The design of a vector pipeline is expanded from that of a scalar pipeline. The handling of vector operands in vector pipelines is under firmware and hardware controls (rather than under software control as in scalar pipelines). Pipeline vector processors to be studied in Chapter 4 include Texas Instruments' ASC, Control Data's STAR-100 and Cyber-205, Cray Research's Cray-1, Fujitsu's VP-200, AP-120B (FPS-164), IBM's 3838, and Datawest's MATP.