MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 86

Asynchronous Push-Down Stacks

by

Frederick C. Furtek

This research was supported in part by the National Science Foundation under research grant GJ-34671, and in part by the Advanced Research Projects Agency of the Department of Defense under ARPA Order No. 433 which was monitored by ONR Contract No. N00014-70-A-0362-0001.

August 1973

The Problem

Suppose that we wish to design a push-down stack that can receive and return data items at a constant rate. If the stack is to be of fixed size, this is not a serious problem as the many existing implementations attest. If, however, we wish the stack to be arbitrarily extendable while still maintaining a constant rate of operation, then we do indeed have a serious problem.

We make the problem more precise in the following way. A processor - human or mechanical - is located at some fixed position in three-dimensional Euclidean space. A scheme is desired whereby a push-down stack of arbitrary size may be implemented for use by this processor. The rate at which the processor may place items on the stack or remove items from the stack is to be totally unaffected by the stack's size and by the number of items on the stack. To insure that the implentation scheme is in fact realizable, we further require that it be consistent with the following physical constraints:

1. There exists a lower bound on the resolution of time.
2. There exists a lower bound on the resolution of space.
3. There exists a lower bound on the amount of energy required to change the state of a device so that the final state is distinguishable from the initial state.
4. There exists an upper bound on the power with which a device of fixed size may transmit signals.
5. There exists an upper bound on the density with which information may be stored
6. There exists an upper bound on the speed of signals.
7. The signal delay through a device changes in a way that is not totally predictable.

These constraints may be thought of as imposed by fundamental laws of physics

or as imposed by the limitations of available devices and instruments. We shall not go more deeply into this issue at present time, but the next section which discusses several faulty solutions to our problem should help clarify the significance of these constraints.

Some Faulty Solutions

The most common way to implement a push-down stack is by means of a random access memory. Typically, data items are placed in successive storage locations and a pointer is continually updated to point to the last item placed in memory (or to the next available location). The basic problem with this approach is that data items remain in fixed locations while in storage. Because of Constraints 5 and 6, as data items are placed in storage without bound the time required to retrieve them also grows without bound. This same problem arises in any scheme where the data items are stored in fixed locations.

We might try some form of tape device to implement the stack - something like a Turing machine with a finite, but arbitrarily extendable, tape. But if we interpret each tape square as a device, it follows from Constraints 3 and 4 (as well as from 5 and 6) that the time required to move the tape can grow without bound. Thus, if the entire tape is moved between successive reads and writes, there is no way to maintain a fixed rate of operation. It could be suggested that the tape remain stationary while the processor moved but this conflicts with our stipulation that the processor remain stationary.

Considering the failures of the two previous approaches we now turn to an implementation at the gate level. Figure 1 is a block diagram of a bidirectional shift register in which each stage operates in the manner of a master/slave flip-flop. The difficulty here is the unlimited fan-out on the

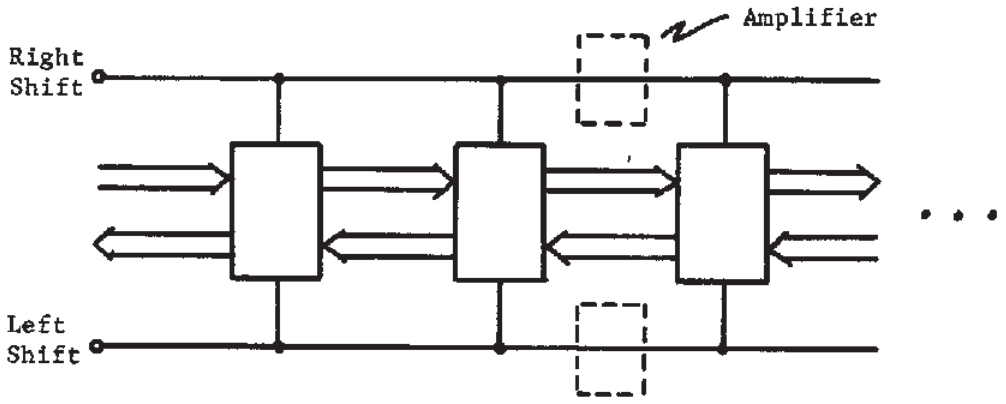


Figure 1 Bidirectional Shift Register

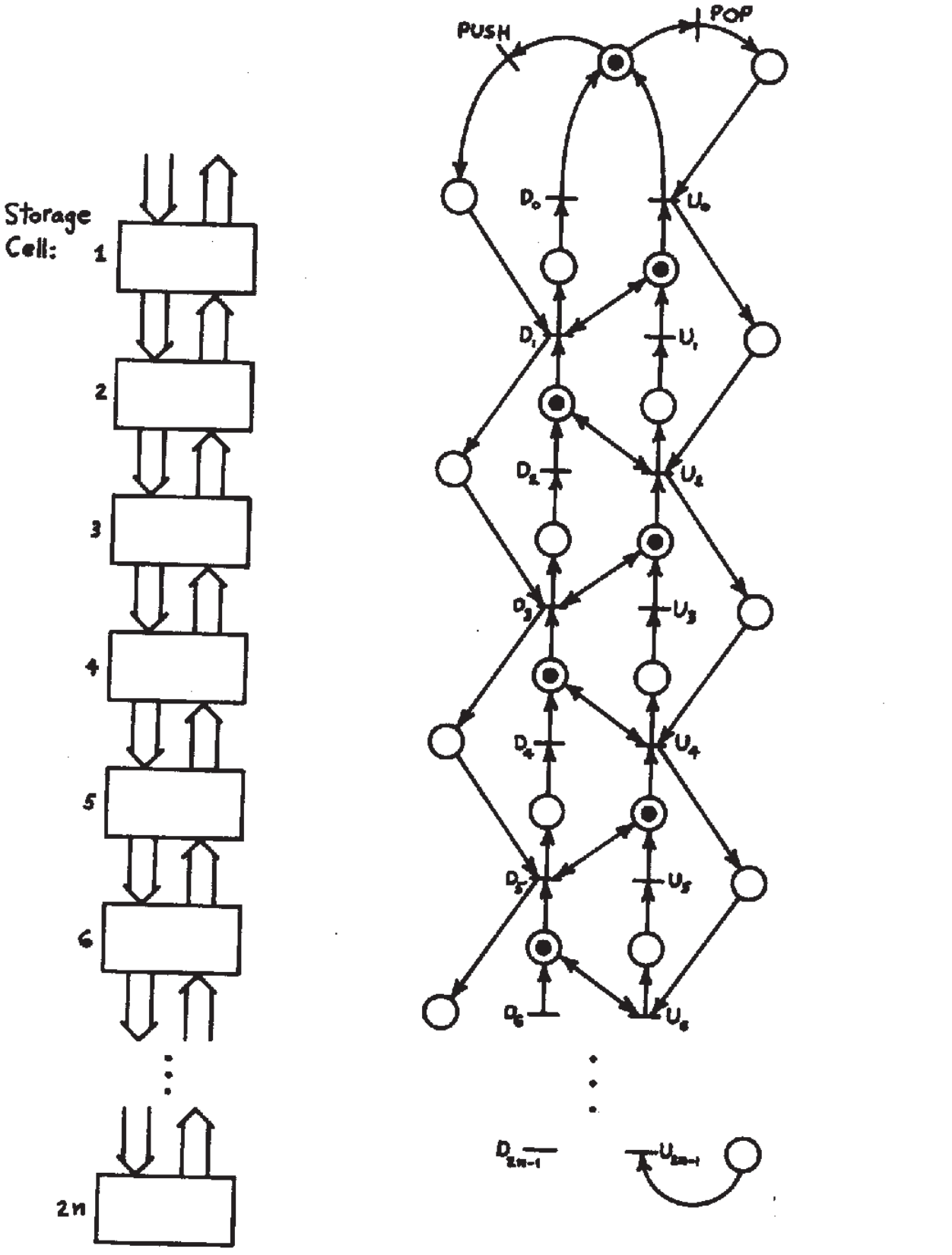
two shift lines, making a constant rate incompatible with Constraints 3 and 4. Of course, we can insert amplification devices at regular intervals on both shift lines, but we now run into trouble from Constraint 7. We cannot assume that the delay through an amplifier remains perfectly constant with time and so there is no guarantee that one shift pulse won't overtake another causing a malfunction. (Constraints 5 and 6 don't permit us to place a bound on the time needed for a pulse to reach the end of the shift register, so it is necessary to allow for concurrently propagating shift pulses.)

In the remainder of this paper we present several designs, each of which is a solution to our problem. They differ from the preceeding ones in that they are totally asynchronous with strictly local control.

A Solution

In this section we describe the most straightforward solution to our problem. For descriptive purposes it is convenient to view the design in terms of the separate data structure and control structure of Figure 2.

Each storage cell is capable of holding just a single data item, and



(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 2 PDS1

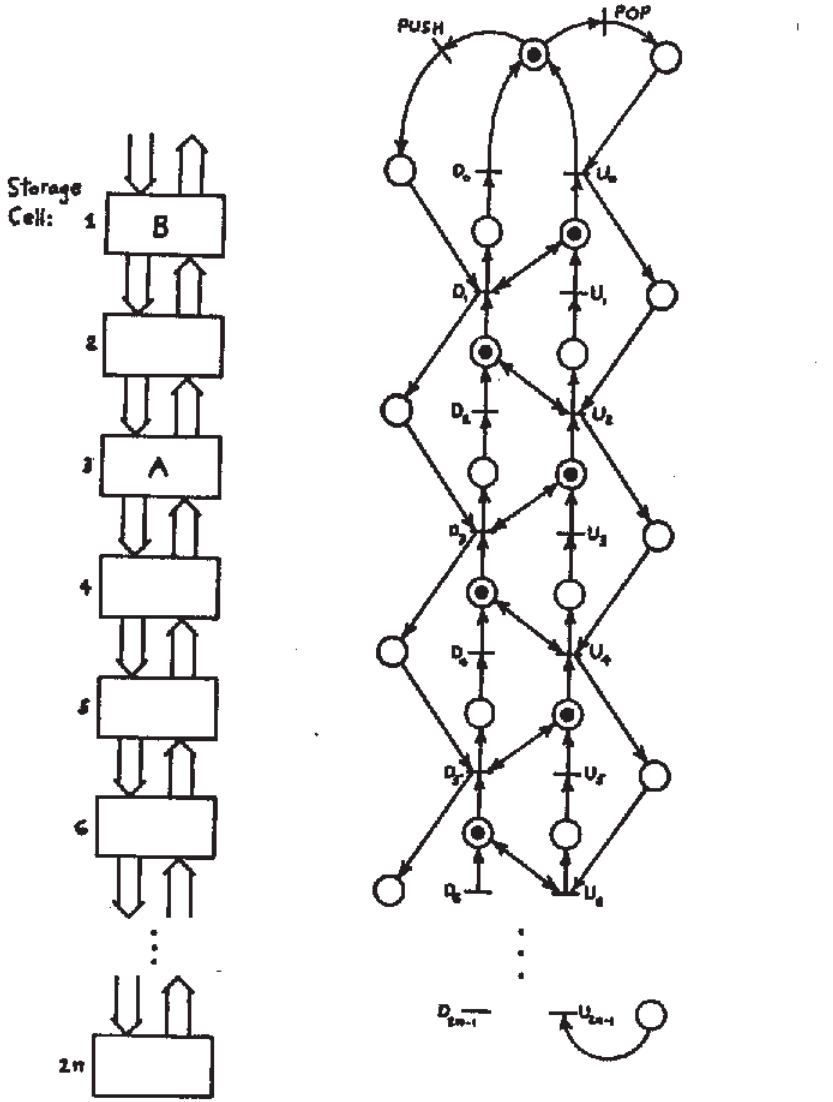
transfer of data items occurs only between adjacent cells. Coordinating these transfers is the task of the control structure which we've chosen to represent by a Petri net [2,4]. Actually, the net in Figure 2(b) represents the behavior of both the stack and the environment in which it is embedded.

The environment is very simple: it just makes push and pop requests of the stack. The free-choice structure involving the events labelled "PUSH" and "POP" represents the environment's complete freedom in the choice of requests. The stack is a little more complicated. Each occurrence of Event D_i represents the transfer of a data item from Cell i Down to Cell i+1. Similarly, each occurrence of U_i represents the transfer of a data item from Cell i+1 Up to Cell i.

The key to the stack's operation is it's tendency to reach an equilibrium condition in which alternate cells are empty. When the stack is pushed, a "compression effect" propagates down the stack and each data item moves down two locations. A similar phenomenon occurs when the stack is popped, but this time a "rarefaction effect" is propagated and data items move up two locations. In order to have fixed rate of operation the stack accommodates concurrently propagating effects, and this is accomplished without the need for conflict. To help clarify these ideas six "snapshots"[1] of the stack in operation are presented in Figures 3-8.

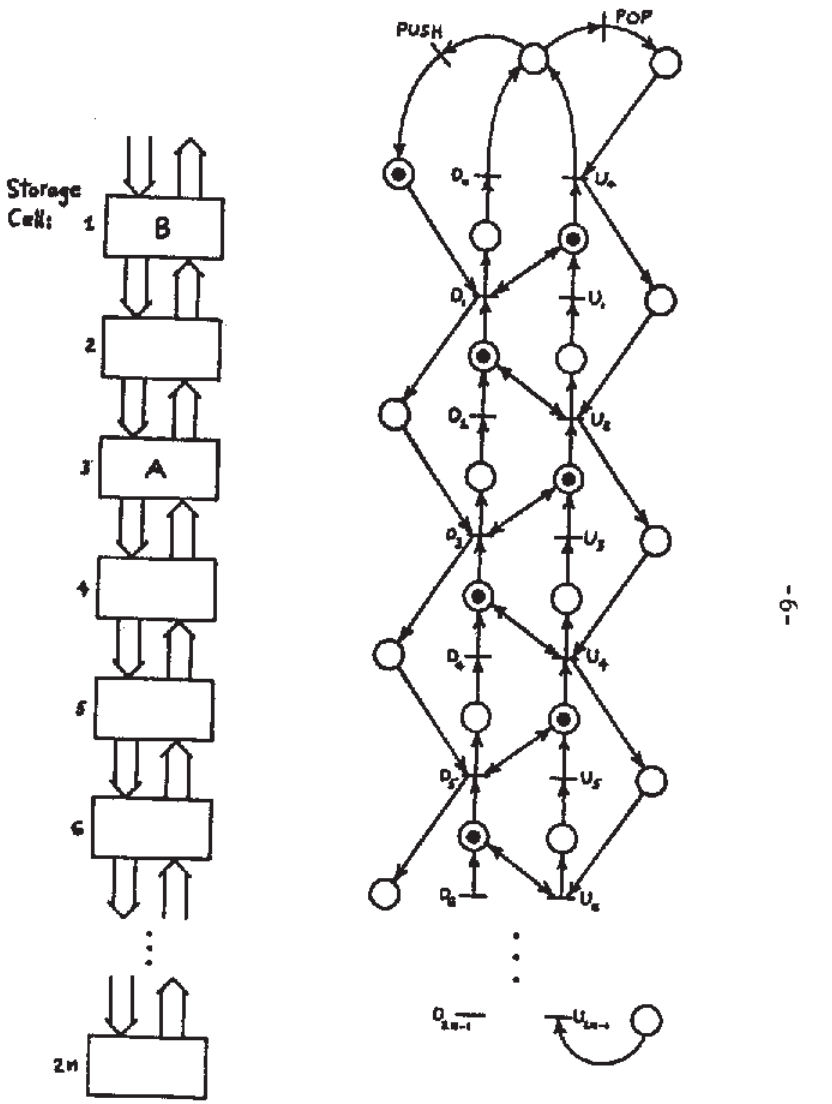
There is one peculiarity about PDS1 that these snapshots bring out. Notice that in Snapshot 4 Event D_5 is ready to occur even though Cell 5 is empty. It takes place nevertheless since this particular stack has no way

[1]There is an inherent problem in using a sequence of snapshots to describe concurrent activity. We are forced to make assumptions about the relative duration of concurrent events. In our case we assume that the duration of each transfer is the same, but, in fact, the correct operation of the stack is unaffected by the duration of any transfer.



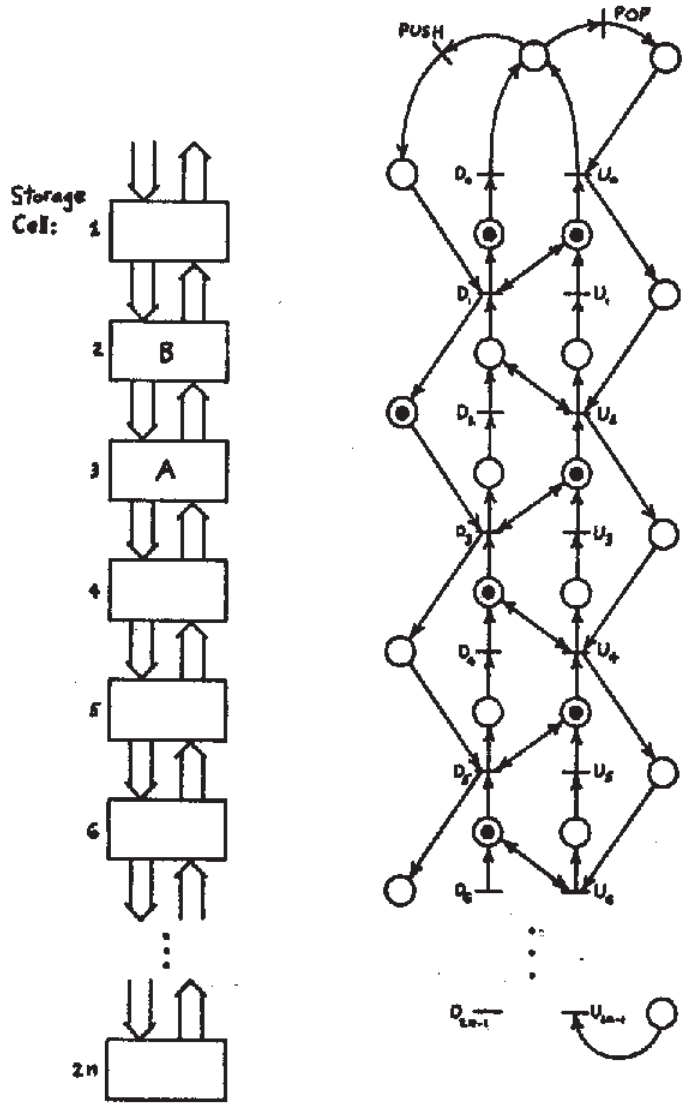
(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 3 Snapshot 1



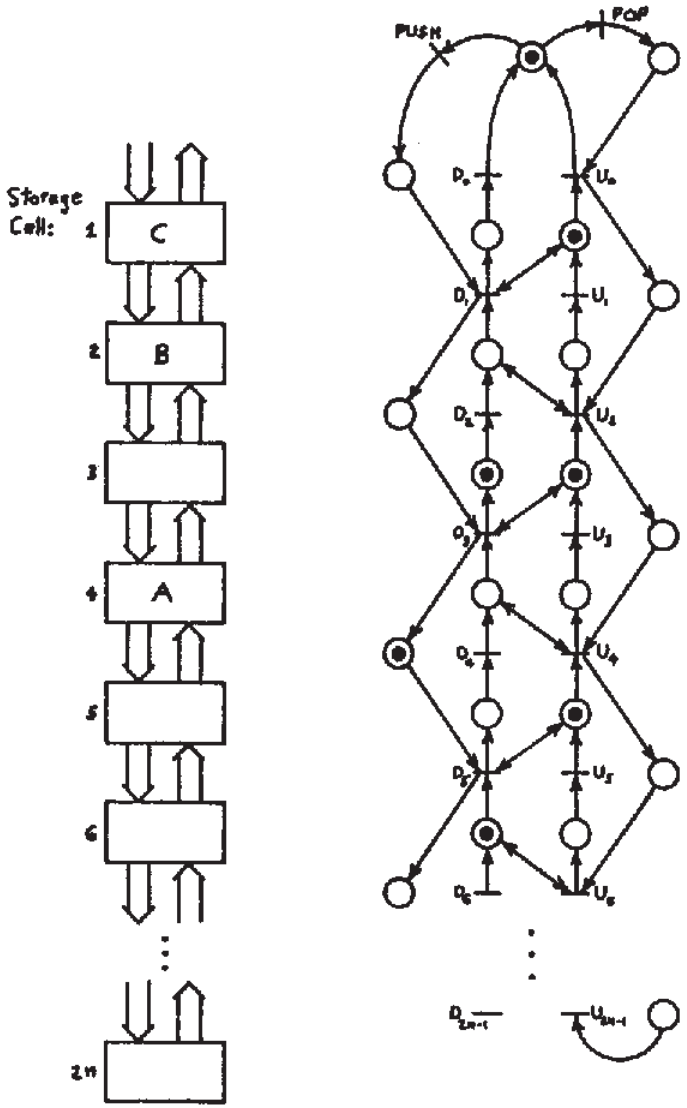
(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 4 Snapshot 2



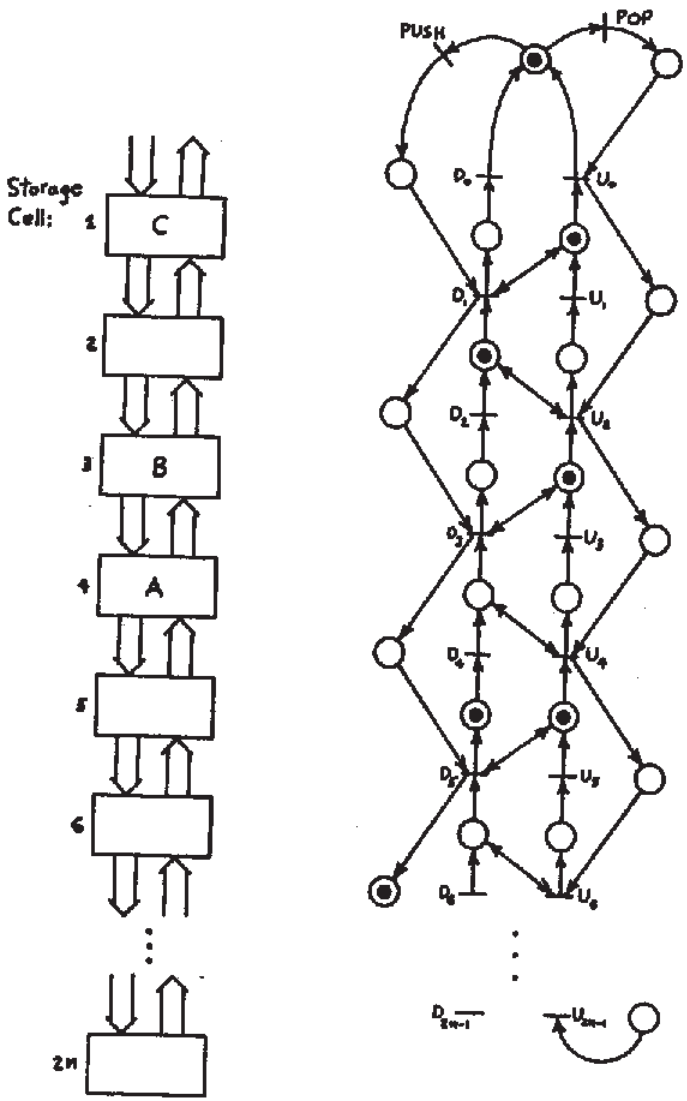
(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 5 Snapshot 3



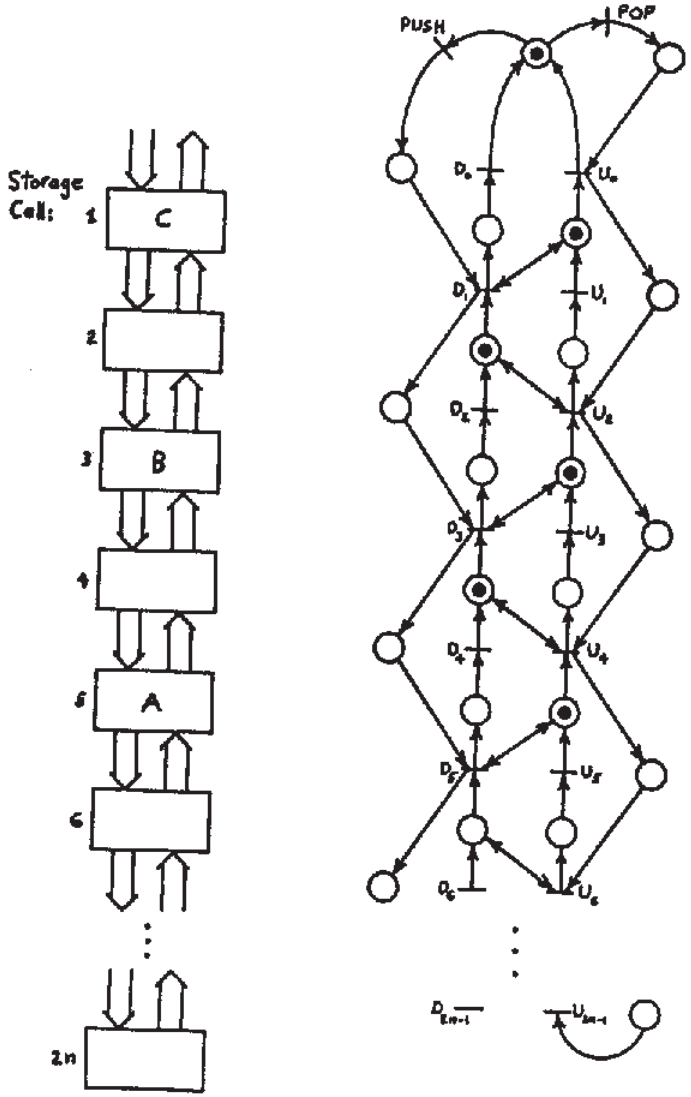
(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 6 Snapshot 4



(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 7 Snapshot 5



(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 8 Snapshot 6

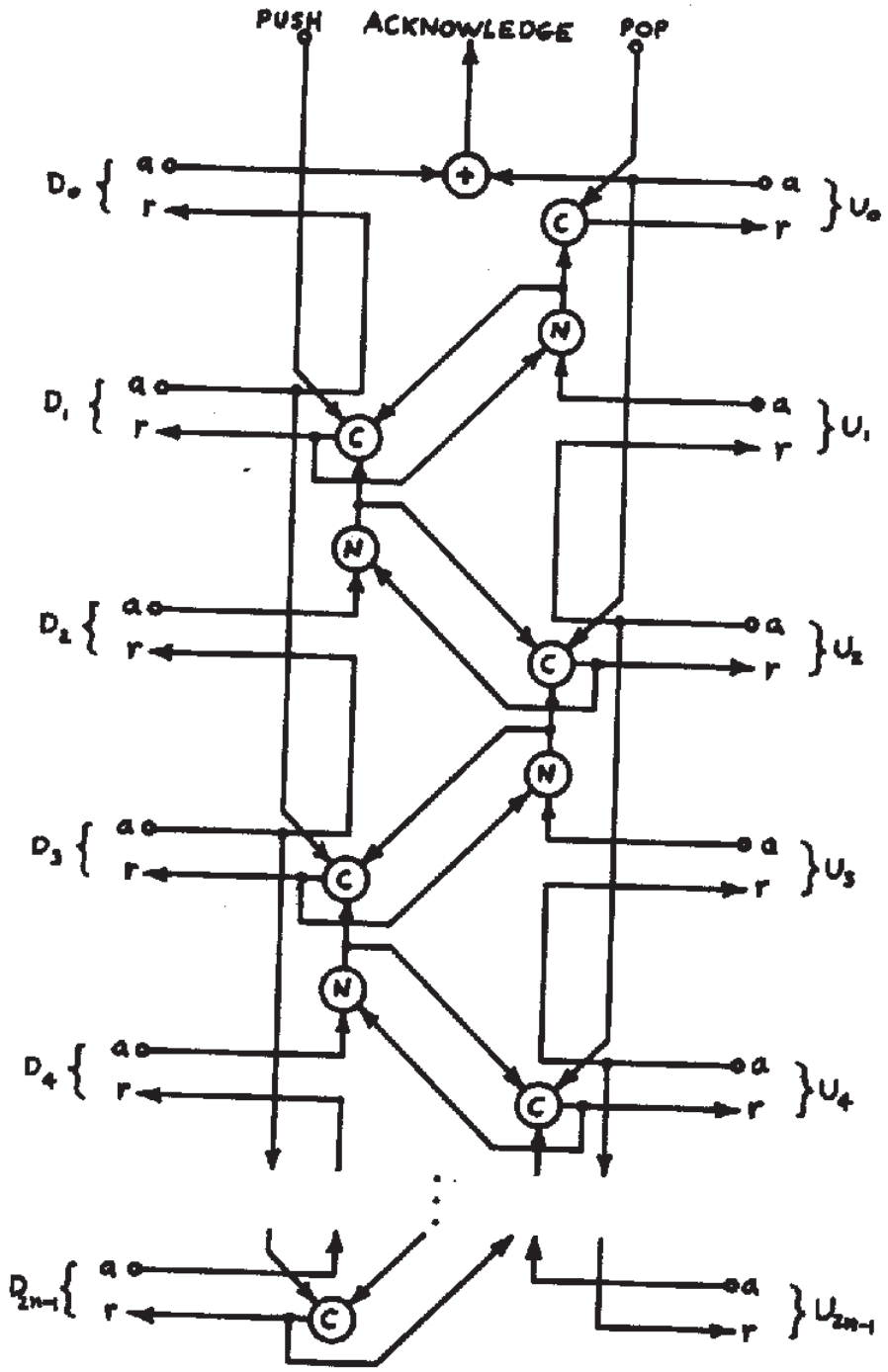


Figure 9 Circuit Realisation of PDS1

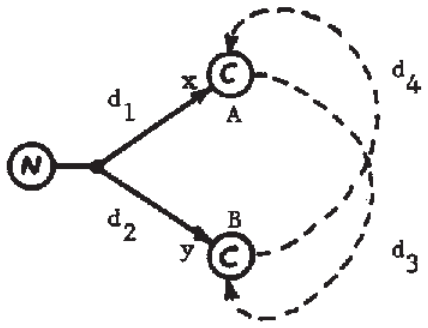


Figure 11 A Critical Configuration

$$\left.\begin{array}{l} d_1 < d_2 + d_4 \\ d_2 < d_1 + d_3 \end{array}\right\} \text{ for each critical configuration}$$

and it is consistent with the seven physical constraints listed above.

To convince ourselves that the operation rate of PDS1 is independent of stack size and the number of data items stored, we first note that PDS1 operates on the same principal as a bucket brigade. Both consist of a linear sequence of identical stages, each of which has a communication protocol in which it communicates alternately with its two adjacent neighbors. In one case, buckets get passed from stage to stage and in the other, push requests and pop requests.

Each stage of PDS1 is associated with the state-machine pair of Figure 12. The enabling of Event D_{x+1} by the next higher stage represents a push request of the stage, and, likewise, the enabling of U_x represents a pop request. Clearly, a single stage is capable of servicing requests at the rate of 1/3T, where T is the maximum duration of any data transfer. An induction argument shows that a stack of arbitrary size is also able to service requests at the rate of 1/3T. This then completes our solution to the problem stated at the beginning of this paper.

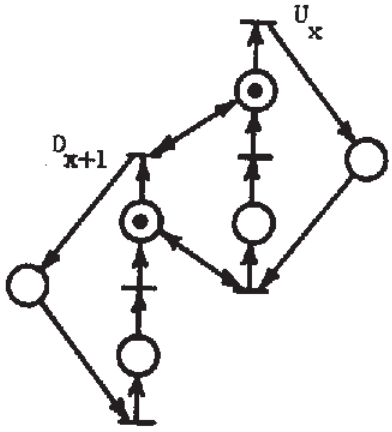


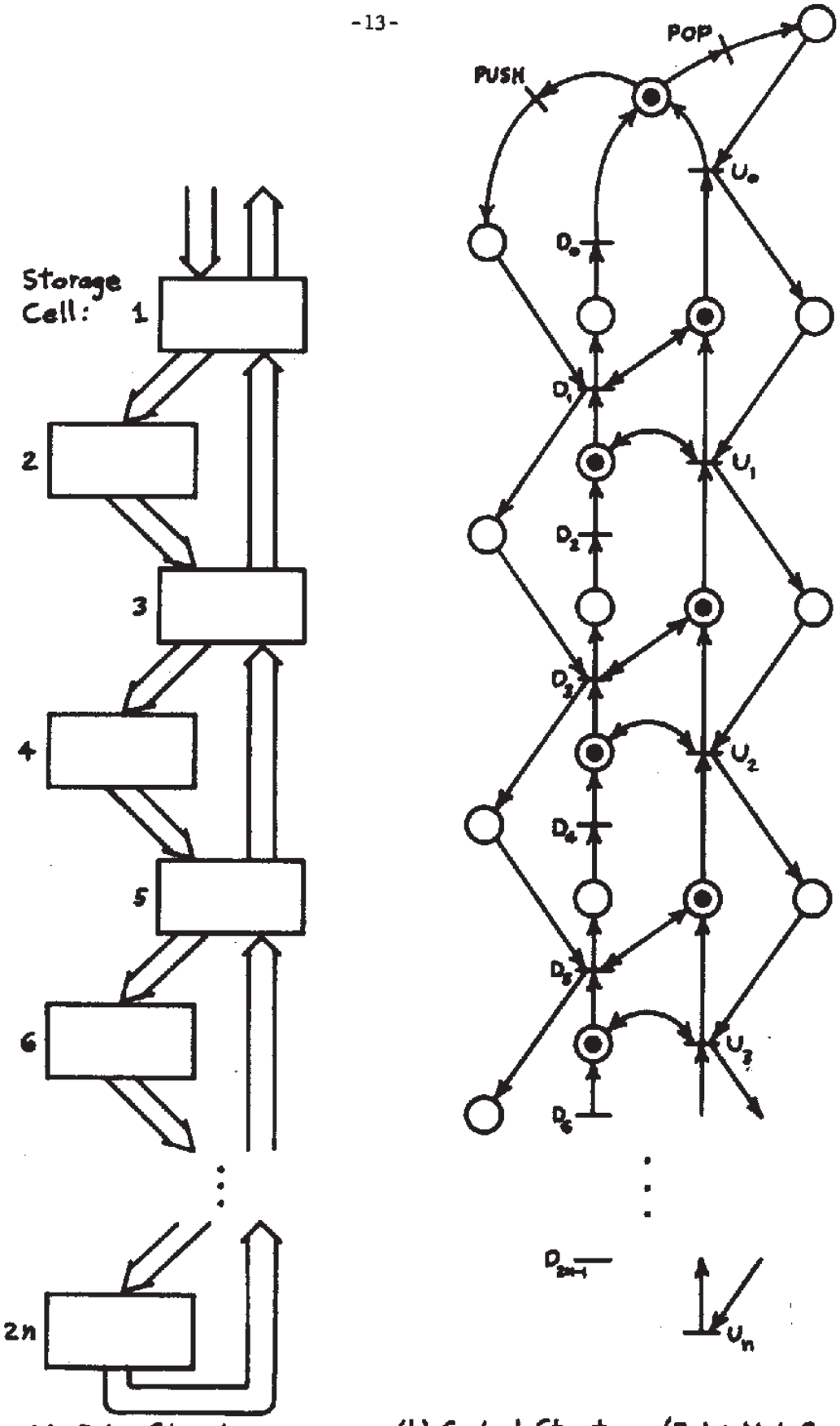
Figure 12 A Single Stage of PDS1

Three Variations

PDS1 can be simplified by combining two transfers in each stage into one. Specifically, the transfers U_{2i} and U_{2i-1}, for $1 \le i \le n-1$, are combined to form a new transfer, U_i, operating between consecutive odd-numbered cells. The simplified stack is illustrated in Figure 13.

This modification works because when a data item in PDS1 moves upward from an odd-numbered cell, that transfer is always followed immediately by another transfer of that same data item to the next higher cell. We've merely combined these two transfers into one. Unfortunately, the same approach cannot be used for downward movement of data items since a data item must, in general, wait for a cell to be cleared before entering it from above. (In this case, it is actually "empties" that move up two locations.)

PDS2 has the interesting property that it may be popped at the rate of 1/2T while it can be pushed at only 1/3T.



(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 13 PDS2

Carl Adam Petri in his doctoral dissertation [5] describes an asynchronous push-down stack, the first to be arbitrarily extendable. We've updated the notation and generalized the design, the result being the stack of Figure 14. It shares with the two previous stacks the property that in steady state only half the storage cells are filled.

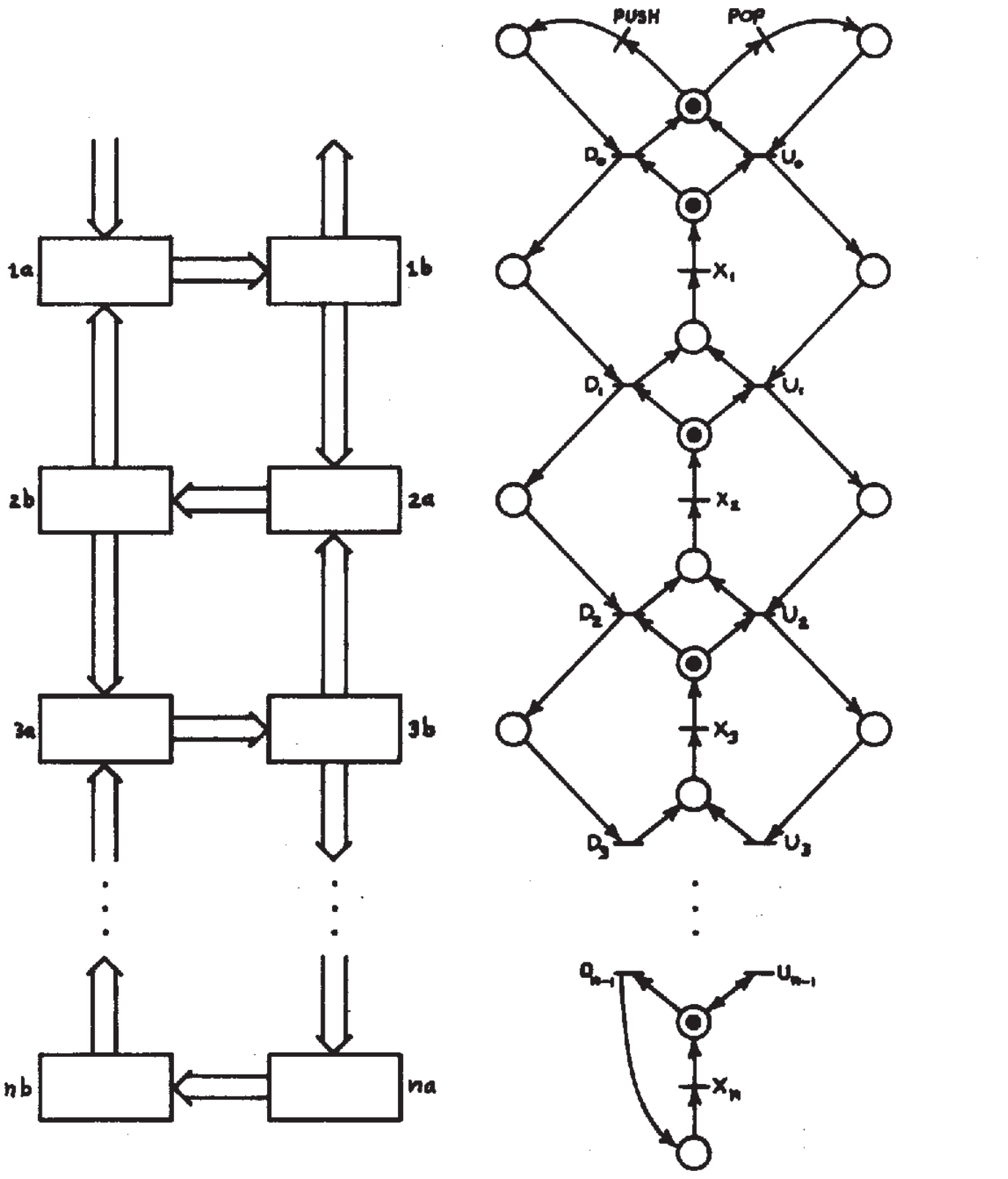
Each stage consists of two storage cells, ia and ib, where $1 \le i \le n$. In steady state, data resides in the "b" cells while the "a" cells are empty. Data enters a stage, from above or below, through the "a" cell and leaves, either up or down, through the "b" cell.

Push and pop requests are handled in a manner similar to that for PDS1. The essential difference is that in PDS3 the path followed by data items on the way up is not the reverse of the path followed on the way down. In particular, a falling data item passes through locations in the order: 1a, 1b, 2a, 2b, 3a, 3b, ···. On the other hand, a data item rising from Stage 3 follows the path: 3b, 2a, 2b, 1a, 1b.

Until now we've described each stack in terms of a separate data structure and control structure. However, it is not always fruitful to partition a design in such a way. A case in point is the stack, with data structure and control structure merged, whose net representation is given in Figure 15.

PDS4 is patterned after PDS2[1] and accepts binary data items. Each token in the left-hand column of states represents a "0" and, likewise, each token in the right-hand column represents a "1". Each pair of states (from these two outside columns) at the same level corresponds to a storage cell. As

[1] In fact, we can define a homomorphism from the net of PDS4 to a net nearly identical to the one for PDS2. This homomorphism preserves both structure and behavior. In effect, PDS4 is just a more detailed version of its image under the homomorphism.



(a) Data Structure (b) Control Structure (Petri Net Representation)

Figure 14 PDS3 (Petri's Stack)

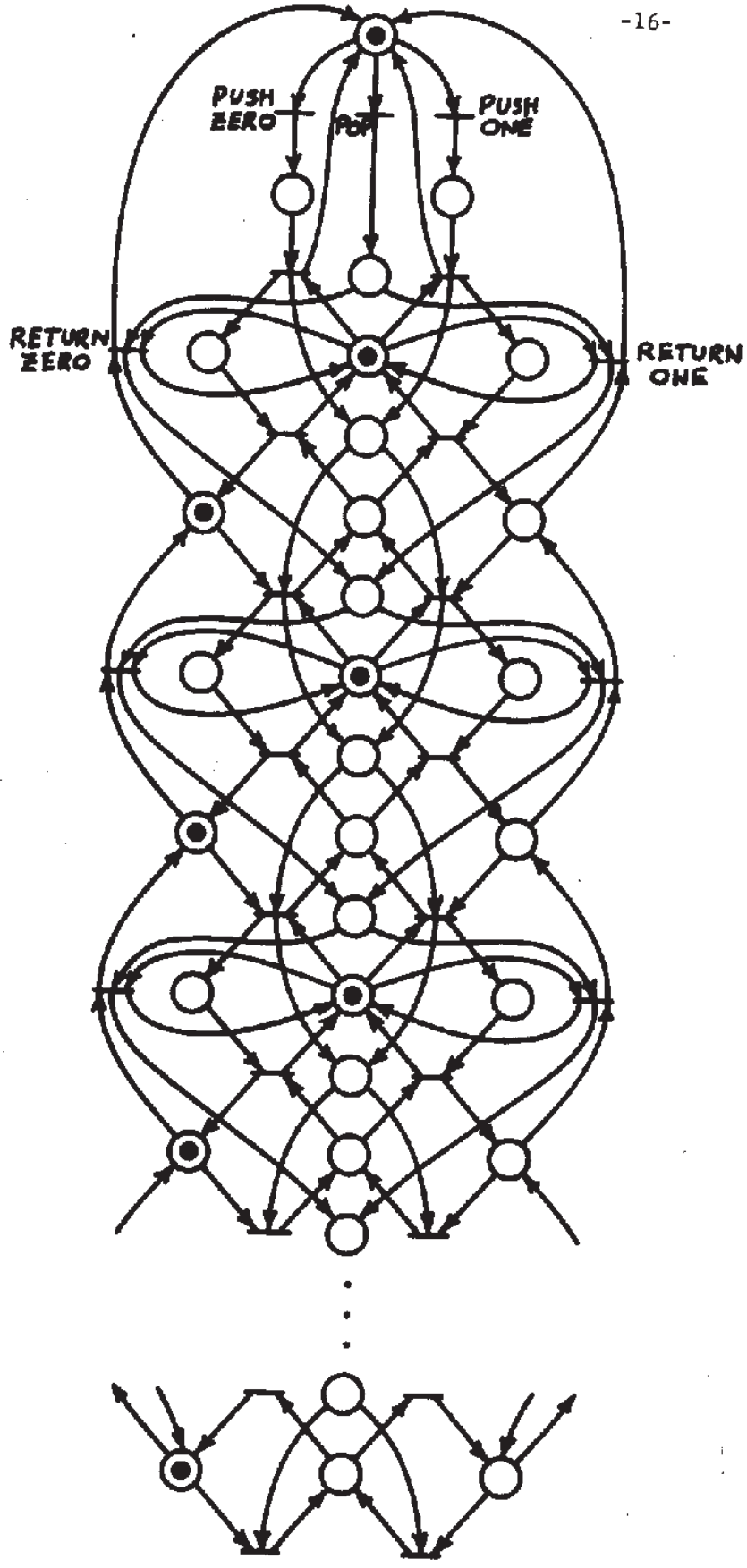


Figure 15 PDS4

can be seen, alternate cells are initially empty and the remaining cells contain 0's. The movement of data items is exactly as specified by the net for PDS2.

It is worth noting that PDS4 makes it possible to store variable-length operands without all the encumbrances found in more conventional designs.

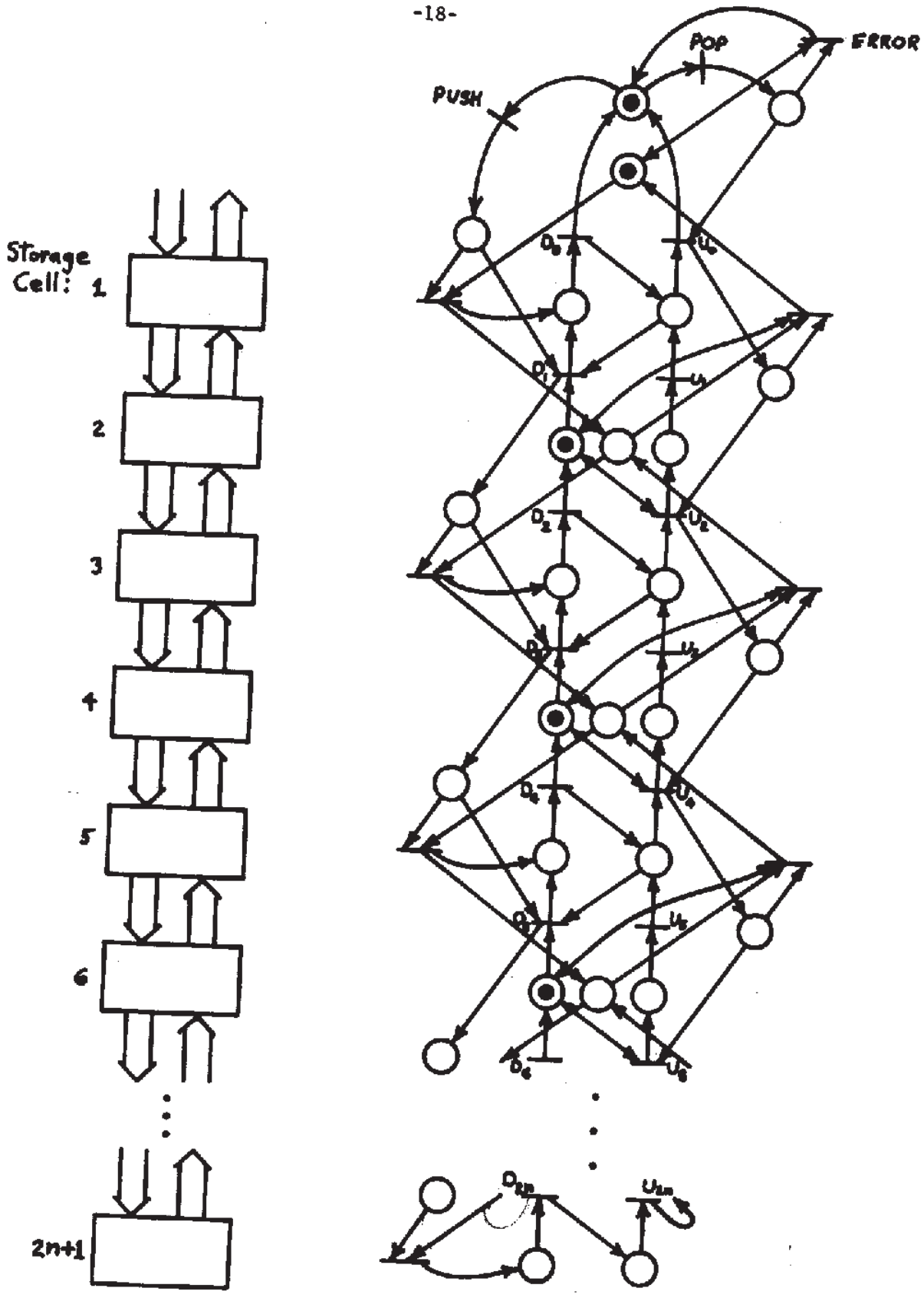
Stacks with Terminating Effects

The four stacks presented so far have a defect: they have no way of knowing where the bottommost data item resides. Thus, the possibility exists that a stack will overflow and it will go undetected. In a practical situation the possibility of losing data cannot be tolerated.

To correct the problem it is necessary for a stack to keep track of the bottommost data item. In PDS5, PDS6, PDS7, and PDS8 (Figures 16-19) the required modifications have been made to correct the deficiencies of PDS1, PDS2, PDS3, and PDS4, respectively.

The principal is the same in all four cases. Each of the four net representations has imbedded within it a state machine resembling a string of diamonds (Figure 20). The token in this state machine indicates the location of the bottommost data item, moving up and down with that data item. When a push effect or pop effect reaches the token's locus, the effect is terminated and there are no dummy transfers as previously.

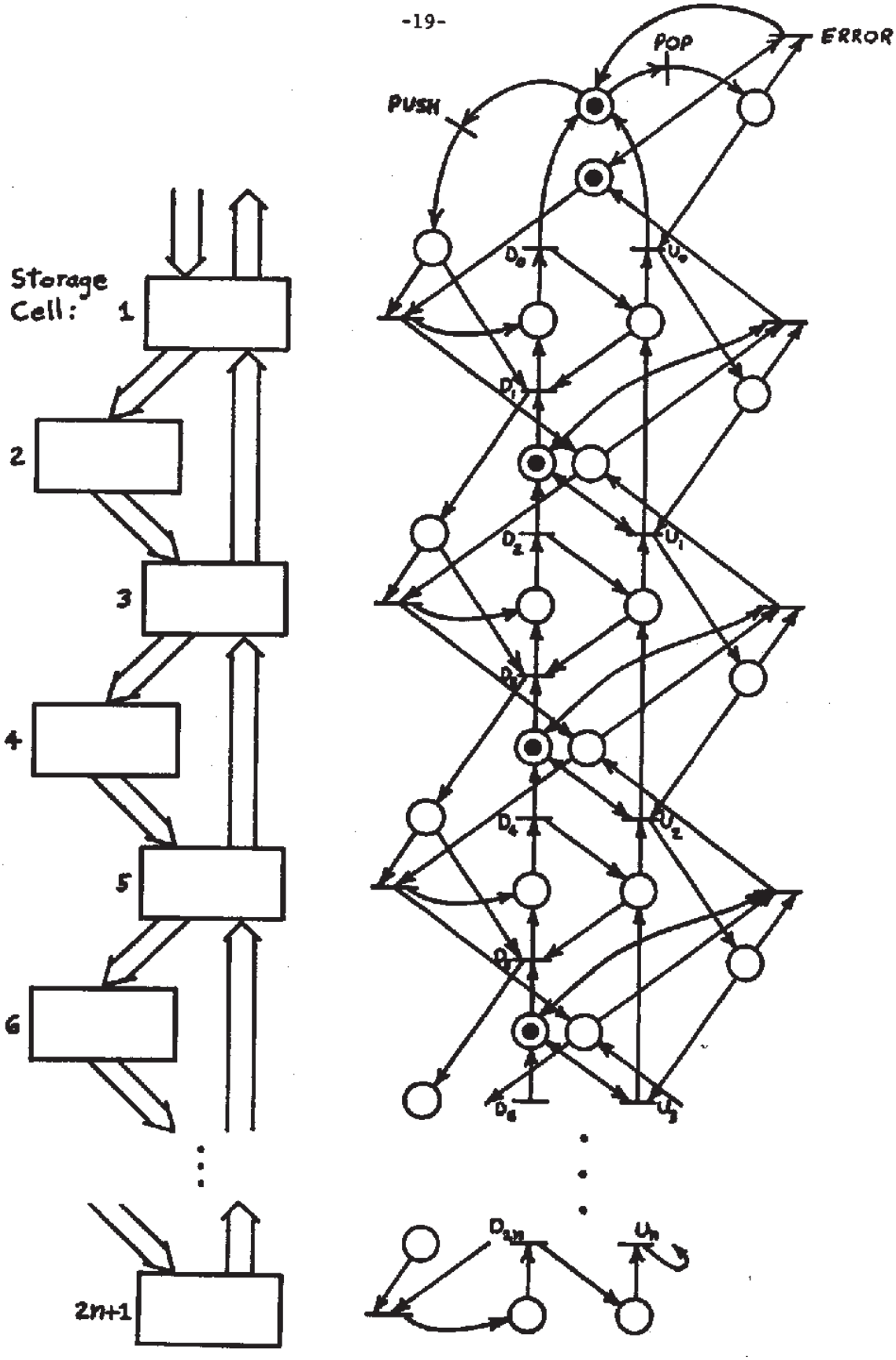
Two cases are of special interest. When the stack is empty - token in the topmost state - a pop request results in an error, as it should. When the capacity of the stack is exceeded the net eventually dies (ceases activity). This happens because effects "back up" when the last stage is unable to make room for a new data item. As a practical matter, two actions are then



(a) Data Structure

(b) Control Structure (Petri Net Representation)

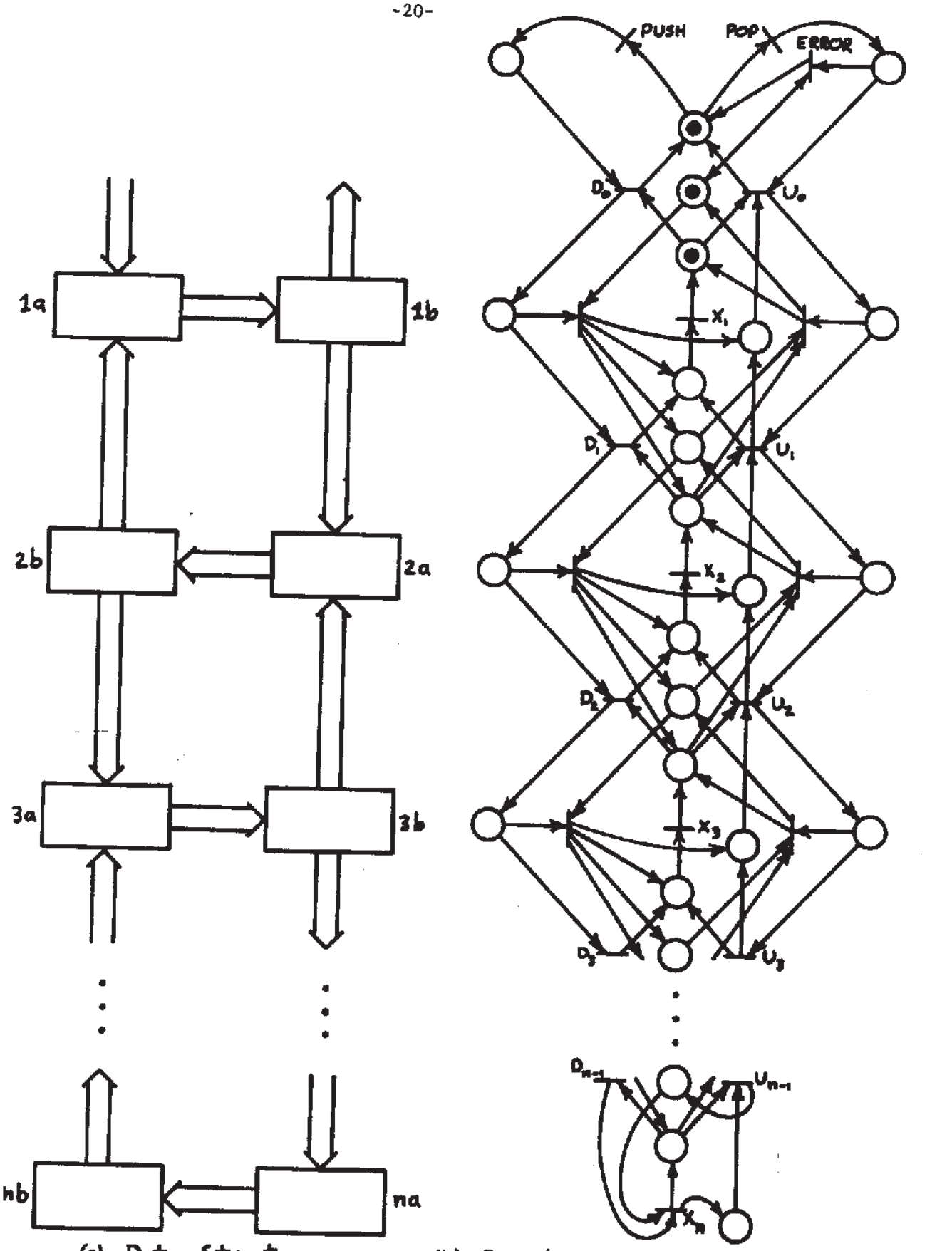
Figure 16 PDSS



(a) Data Structure

(b) Control Structure (Petri Net Representation)

Figure 17 PDS6



(a) Data Structure

(b) Control Structure (Petri Net Representation)

Figure 18 PDS7

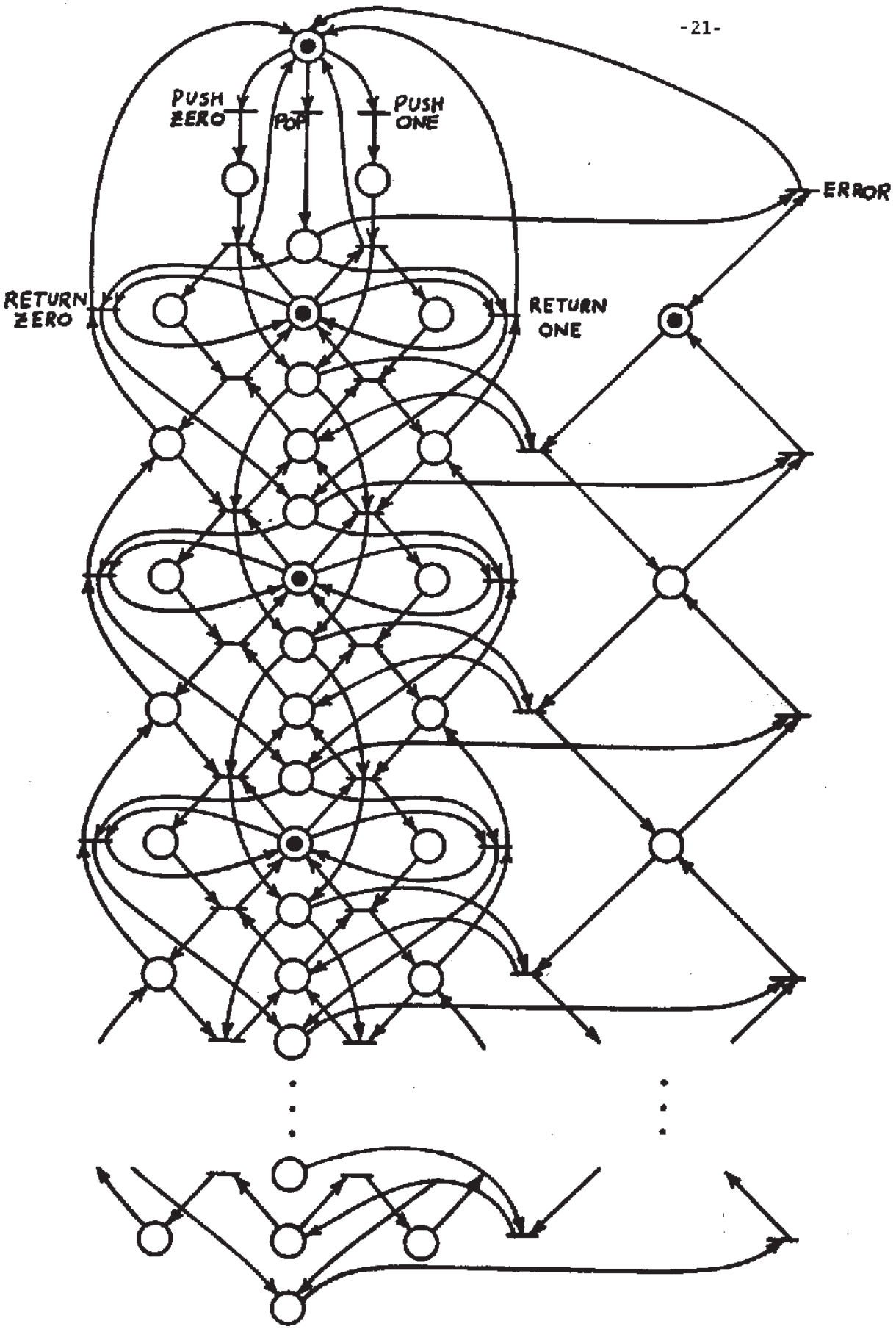


Figure 19 PDS8

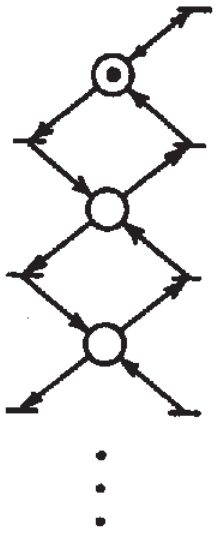


Figure 20 Indicator State-Machine

possible: the task can be aborted and a new one initiated, or more stages can be added to accomodate the demand. The latter alternative is especially intriguing since the stack then becomes arbitrarily extendable and is no longer a finite-state device (Petri makes this point in his thesis).

Conclusions

The significance of the foregoing results is twofold.

First, we've presented several solutions to the problem we initially set for ourselves, namely, to design a push-down stack whose rate of operation is unaffected by stack size or by the number of data items stored on the stack. A stack implemented according to one of these schemes is asynchronous, conflict-free, and arbitrarily extendable. This result is of particular interest to those looking at the general problem of designing computational device that can be extended indefinitely without impairing efficiency.

Second, we've demonstrated the appropriateness of Petri nets for describ-

ing complex communication structures. The use of nets has allowed us to express our intentions exactly. As a consequence we can make the following claim:

> Each stack described in this paper has the property that a data transfer occurs as soon as it is logically possible. There are absolutely no artificial or unnecessary constraints placed on the movement of data.

References

1. Dennis, J. B. and S. S. Patil, "Speed Independent Asynchronous Circuits", Proceedings of the Fourth Hawaii International Conference on System Sciences, January 1971, pp 55-58

2. Holt, A. W. and F. Commoner, Events and Conditions (In three parts), Applied Data Research, Wakefield, Mass., 1970 [Chapters I, II, IV, and VI appear in Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York, 1970]

3. Muller, D. E., "Asynchronous Logics and Application to Information Processing", Switching Theory in Space Technology, Stanford University Press, 1963, pp 289-297

4. Patil, S. S. and J. B. Dennis, "The Description and Realization of Digital Systems", Digest of Papers 1972: Innovative Architecture, IEEE, New York, pp 223-226

5. Petri, C. A., Communication with Automata, Supplement 1 to Technical Report RADC-TR-65-377, Vol. 1, Griffiss Air Force Base, New York, 1966 [Originally published in German: Kommunikation mit Automaten, University of Bonn, 1962]