MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 120

Micro-Control for Parallel Asynchronous Computers

by

Suhas S. Patil

(A paper to be presented at the Euromicro Workshop, at Nice, France, 23-25 June 1975.

This work was supported in part by the National Science Foundation under research grant DCR 74-21822.

March 1975

MICRO-CONTROL FOR PARALLEL ASYNCHRONOUS COMPUTERS*

Suhas S. Patil
Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
U.S.A.

Conventional micro-controls employ some form of memory to store the control words and read the words sequentially to generate the desired control signals. Because of the single sequence operation, these control structures are neither efficient nor functionally suitable for the control of parallel asynchronous computers where many loosely coupled parallel activities may have to be controlled. This paper shows how an asynchronous logic array based on the Petri net model for representation of parallel systems can be used as a micro-control for parallel asynchronous computers. We explain the general approach and give examples to show how many essential control tasks such as sequencing, conditional operation, iteration, parallel process coordination, resource allocation and priority schemes can be easily implemented, and we examine how the logic array and the control programs can be structured for efficient utilization of the array. Since an asynchronous logic array is parallel in nature, the micro-control structures implemented with the array operate in parallel, allowing activities of parts which are independent to proceed in parallel without any interference.

INTRODUCTION

In this paper our primary concern will be to show how an asynchronous logic array capable of parallel realization of Petri nets can be used to implement the micro-controls for asynchronous parallel computers. We shall present Petri nets as a means for the representation of control, discuss the relation between Petri nets and the asynchronous array, briefly explain the nature of the array, and with examples show how various control tasks can be easily implemented.

Jump [9] has reported on an asynchronous array for realizing the subclass of Petri nets known as marked graphs [4], and the author has also found two different asynchronous arrays for the realization of Petri nets. One type of array, called asynchronous cellular array [16], can realize the subclass known as Simple nets, and the other type of array, called Kolte array [17], can realize the class of all safe Petri nets. Therefore in the following discussions we will assume that the full class of Petri nets can be realized by the arrays used in the implementation of the control. If the arrays are manufactured at an economical price, we will have a suitable means for realizing the micro-control of asynchronous parallel computers. In this regard the Kolte array, which consists of a diode array with terminating circuitry on each row and each column, is most promising because it is suitable for LSI technology.

* This work was supported in part by the National Science Foundation under research grant DCR 74-21822

REPRESENTATION OF ASYNCHRONOUS CONTROL

The Petri net model for the representation of parallel systems is an excellent language for the representation of asynchronous control structures, Dennis [5,6], Noe [10] and Patil [11,13, 15]. In the following paragraphs we shall describe Petri nets and their use in the specification of control structures.

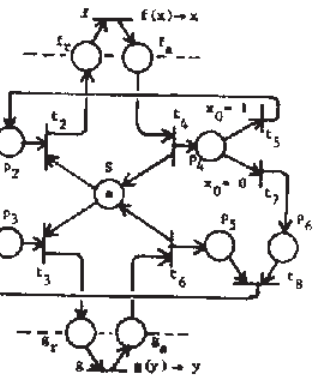
Petri nets consist of places and transitions which are connected by directed arcs; a directed arc connects either a place to a transition or a transition to a place [7, 8, 11] (there is no arc that directly connects a transition to another transition or a place to another place). The places from which there are arcs incident on a transition are called the input places of that transition; the output places of a transition are similarly defined to be those places which are connected to the transition by arcs which originate at the transition and terminate at the places. A place may have a token or it may be empty. When there is a token in each input place of a transition, that transition is said to be enabled because it is ready to fire. The act of firing involves removing a token from each input place and putting a token in each output place. We will only deal with nets in which no firing sequence for the net can lead to more than one token at a place. Such nets are called safe nets. The safe nets can represent all the finite state systems, since an unsafe net with a bound on the number of tokens can always be translated into a safe net. A fact to remember about transition firing in a safe net is that when two transitions share an input place, even if both are enabled, only one of them can actually fire because the single token at the shared place is consumed in the firing of one transition and the other transition is disabled for want of a token.

The number of tokens in a net is not necessarily conserved because a transition does not necessarily have an equal number of input and output places. In this paper we shall use an extended form of Petri nets in which a Boolean condition, expressed as a product of some Boolean variables, may be associated with a transition. In these nets a transition is enabled only when in addition to the presence of a token at each input place, the Boolean condition associated with the transition is satisfied.

In this paragraph we describe the operation of the Petri net of Figure 1a; those who are familiar with Petri nets and are able to read the control process specified by this net may safely proceed to the next paragraph. In the Petri net of Figure 1a, initially transition t_1 is the only transition which is enabled to fire. When this transition fires, place p_1 becomes empty and places p_2 and p_3 get tokens. At this point both transitions t_2 and t_3 are enabled but because of the shared place S, only one of them can be fired. Say transition t_2 fires, in which case place f_r gets a token and the operation $f(x) \rightarrow x$ which is controlled by transition f is initiated; because place S no longer has a token, transition t_3 will have to wait until transition t_4 fires after the firing of f (execution of $f(x) \rightarrow x$) is completed. Say t_4 has fired placing tokens in p_4 and S. At this point t_3 and, depending on the Boolean value of the variable x_0, either t_5 or t_7 will be enabled. Since t_3 and t_5 (or t_3 and t_7) do not have any common input places, their execution (firing) will proceed independently of each other. If x_0 is at level 1, then t_5 fires and places a token in p_2. If by this time t_3 has fired, then t_2 will not be enabled for firing until t_6 has fired after operation $g(y) \rightarrow y$ has been completed. Thus we see that this Petri net specifies a cyclic control process where in each cycle operation $g(y) \rightarrow y$ is performed once, and operation $f(x) \rightarrow x$ is performed iteratively until $x_0 = 0$, and, furthermore, even though the execution of f and g may take place in any sequence, they never overlap. In Figure 1b we have expressed the same process as the one specified by the Petri net of Figure 1a, but in a programming language with the fork, join and the semaphore primitives so that by comparison one can gain an appreciation of how various control primitives have been realized in the Petri net. For example, transition t_1 performs the fork operation and transition t_8, the join operation. Place S acts as a semaphore; transitions t_2 and t_3 as P[S] instructions; and transitions t_4 and t_6 as V[S] instructions. Transition f represents the execution of $f(x) \rightarrow x$, place f_r is used to initiate this operation and f_a serves as a point where the control (denoted by a token) is returned after the execution of the said function. Place p_4 together with transitions t_5 and t_7 implement the conditional instruction.

The physical control structure that the net of Figure 1a specifies has two output links, one to control operator f and another to control g (see Figure 1c). We are assuming that asynchronous operators are controlled by two-wire ready-acknowledge links; a signal on the ready link initiates the execution of the operation and a signal on the acknowledge link indicates the completion. More details on control structures may be found in other papers [1, 2, 3, 5, 6, 11, 13, 15]. Places f_a and f_r are considered input/output places because the operator f is considered to be outside of the control structure. These two places correspond to the two wires of the control link which connects operator f to the control structure, and the passage of tokens through these places corresponds to the passage of signals on the corresponding wires; the input-output places of the net represent the control wires and the tokens represent the control signals.

a) A Petri net



b) The control process

```
    begin                                        initial value
p1: fork p2;               p2: P[S];                S = 1
    P[S];                      f(x)→x;
    g(y)→y;                    V[S];
    V[S];                      if x0= 1 then goto p2
    join 2,p1                            else join 2,p1;
                               end
```

c) The control structure

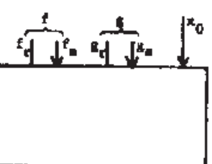


Figure 1. Representation of the control

Matrix Representation for a Petri Net†

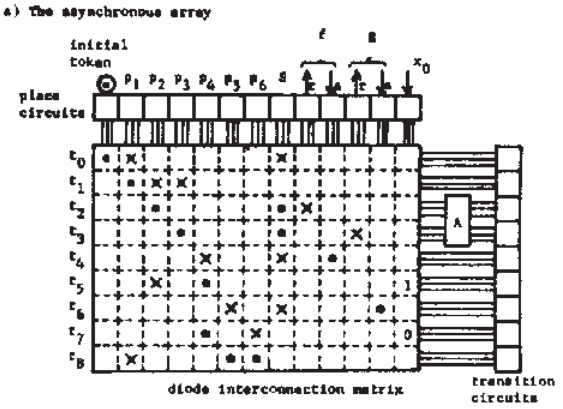
A Petri net can be represented by a matrix in which the rows represent the transitions and the columns represent the places and the test variables whose Boolean value control the firing of some transitions such as the transitions in the net implementing the conditional statement. The connectivity of the net, that is, how the places and transitions are connected to each other, is indicated by means of two symbols, the dot (•) and the cross (×). A dot in the cell at the interconnection of a column representing a place and a row representing a transition indicates an arc from the place to the transition, and a cross in the cell indicates an arc from the transition to the place. We shall assume that the nets do not have places which have both kinds of arcs to a transition so that a cell in the matrix may have either a dot or a cross but not both; this assumption does not restrict the class of systems under consideration because Petri nets which do not satisfy our assumption can be transformed into equivalent nets which do satisfy the assumption.

In our use of Petri nets we have allowed a Boolean product of Boolean variables to be associated with a transition. In the matrix notation, we shall therefore allow two additional symbols, a 0 and a 1, to express the Boolean condition associated with the transition. A 1 in a cell will indicate that the value of the Boolean variable associated with the column must be a 1 in order for the transition to be enabled; the meaning of cell 0 is similarly defined. These cell symbols may also be used in columns which represent places. The Boolean variable in this case corresponds to the predicate which tests whether there is a token in that place represented by the column; if the place has a token then its value is a 1, otherwise its value is a 0.

A matrix representation for the net of Figure 1a appears in the matrix of Figure 2a. Row t_0 indicates which places need the initial tokens and rows t_1 - t_8 indicate how the transitions relate to the places of the net and the Boolean value of input x_0. For example, row t_5 indicates that transition t_5 has an input from p_4 and output to p_2, and it can fire only when input x_0 is a 1.

†Our matrix notation is an extension of a matrix notation that Dr. Holt has used for many years to represent large nets on a piece of graph paper. Our notation reduces to that of Holt if we restrict the use of symbols to only the dot and the cross.

a) The asynchronous array



b) Cell configurations

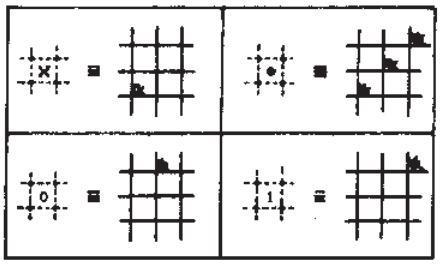


Figure 2. The asynchronous logic array

THE ASYNCHRONOUS LOGIC ARRAY

In this section we shall briefly describe an asynchronous logic array called Kolte array [16]. The bulk of this array consists of a diode array which is subdivided into rows and columns each made up of three wires (Figure 2a). The cells at the intersection of the rows and columns are provided with one of four diode configurations depending on the symbol in the corresponding cell of the matrix representation for the net being implemented (see Figure 2b); if the cell is unmarked then the intersection does not contain any diodes.

Along the top and on one side of the array the wires of the column and row are connected to appropriate terminating circuits. The circuits at the top, called place circuits, carry out the function of the places and the circuits at the side, called transition circuits, make the rows behave like transitions in the net; the input place circuit also serves as a means for connecting a wire representing a Boolean variable input to a column. If there are transitions in the net that might be in conflict, because they share common input places, an arbiter [14] is placed in between the array and the transition circuits connected to the rows representing those transitions (Figure 2a). The place circuits for an output place have a wire coming out of it and the place

circuit for an input place has a wire incident on it; the place circuit for a column representing a test input is the same as the one for an input place. A place circuit can be in one of two states, ON or OFF. Except in the case of a place circuit for an output place, the ON condition represents a token and the OFF condition the absence of a token. Note that because we are only considering safe nets a place has only two possible states. In the case of an output place circuit only the change in the state has any meaning; the change takes place each time a transition puts a token into the output place. The level of the output wire of the output place circuit reflects the state of the circuit; a signal being defined as a change in level, each time a token is placed into the output place, a signal is sent out. Similarly a signal received on the input wire of the input place circuit changes the state of that circuit. If the input wire is the acknowledge wire of a control link then, since the net is safe, the circuit will be OFF before the signal arrives and it will turn ON when the signal arrives. If the input wire is a wire whose level is to be tested in the array then the state of the circuit will be the same as the state of the wire; when a signal (change in state of the input wire) is received the circuit will change its state to reflect the change in the state of the input wire.

Initialization of the array has the following effect: (i) a special internal place circuit which is to be the source of the initial token, is placed in ON state, (ii) all other internal place circuits and output place circuits are placed in the OFF state and (iii) all input place circuits are set to the same state as the input wires.

When we say the state of a column we mean the state of the place circuit connected to the column. A transition is ready to fire when all columns which have the marking • in it are ON and the states of the columns that have 0 and 1 markings correspond to these markings. A transition fires in two steps: first it turns OFF the place circuits of all the columns which have marking • in the row representing the transition and then it changes the state of the place circuits of all the columns with marking x in the row. The two step firing sequence ensures that the tokens from the input places of the transition are removed before tokens are placed in the output places. Firing a transition proceeds independently of other transitions except in the case when the row representing the transition is connected to an arbiter, in which case the transitions connected to the arbiter fire one at a time; the arbiter is biased so that it favors the transition closest to the top in making decisions about the next transition to be fired.

More details of the array including the details of circuits can be found in another paper [16]. It will suffice here to say that the array operates asynchronously with each enabled transition independently proceeding to fire as soon as possible. When the cell configurations of the array are chosen in accordance with the matrix representation of the Petri net, then the array produces control signals in accordance with the specification of that net.

MICRO-CONTROL

In an earlier section we have seen how sequencing of control, fork and join operations, conditional branch, iteration and semaphore operations can be easily expressed with Petri nets, and how the nets can be used in the specification of the control structures for digital systems. These are the basic control primitives one needs to express the control of a computer. Furthermore, we have briefly discussed an asynchronous array which can implement the Petri net in hardware. Thus we have the basic ability to realize the micro-control provided we have a large enough array to implement the Petri net specification of the control. In this section we shall examine the micro-control to see how some of the concepts of the conventional micro-control, such as organization of words into fields to save on the width of the words, are also valid for the micro-control based on the asynchronous array. We shall also present some typical examples of control structures one would encounter in realizing a micro-control for a computer. The examples also illustrate some general principles for efficient (economical) use of the array.

Organization of Columns into Fields

The concept of a field is useful when among a number of control points, only one is activated at a time. If one of eight registers of a system is transferred to the data bus at a time, then a three bit field could indicate which one of the transfers is to be performed, and a control link could indicate when the transfer is to be performed. In the illustration shown in Figure 3, the field consists of the columns representing places p_1, p_2 and p_3, and the columns representing the control link L. The token distribution in the places p_1, p_2, p_3 indicates the choice of the particular transfer. If the transfer $X \rightarrow A$ is to be performed, tokens are placed in p_1 and p_3 (and p_2 is left blank) so that the bit pattern 101, which corresponds to the choice of the desired transfer, is generated, and then a ready signal is sent on link L to initiate the transfer. When the transfer is completed, an acknowledge signal is received on link L. Then transition t_3 empties places p_1 and p_3, so that the field is free to be used again, and puts a token in place p_5 to signal that the transfer operation has been completed.

Instruction Decoding

In a micro-control it is often necessary to decode the bit patterns of a field, such as the operation code of a computer instruction, so that the control sequence can be transferred to the appropriate part of the micro-control. While the ability to perform a conditional transfer based

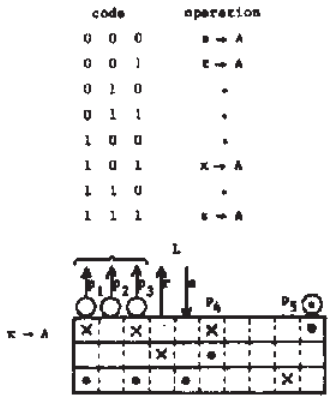


Figure 3 A micro-instruction field

on a single bit is adequate to perform this task, the ability to transfer the control to one of several locations based on a single test which simultaneously examines all the bits is very convenient. The 0 and 1 cell configurations of the array provide us with this capability. The illustration in Figure 4 shows a control structure which decodes an operation code field consisting of columns b_1, b_2 and b_3. If we examine transition t_5 we will notice that it has an input from place p_1 and has the cell configuration 101 in the columns of the operation code. Furthermore, no other transition which has the same input places has the same cell configuration. Therefore, if the operation field of the instruction register contains the bit pattern 101, t_5 is the only transition that is enabled when a token is placed in place p_1. It may be recalled that in order for a transition to be enabled, in addition to having tokens in all input places, the bit pattern applied to the test columns must match the cell configurations of that transition. Therefore, transition t_5 is the only transition which fires. In the process of firing, t_5 removes the token from p_1 and puts tokens in appropriate places to begin the execution of the instruction.

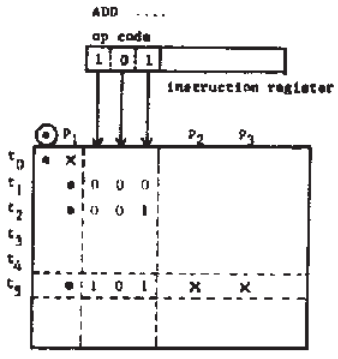


Figure 4. Instruction decoding

Efficient Utilization for Testing

In testing a field, there are times when we are only interested in finding out if the field contains a few of the many possible bit configurations. This situation can be handled more efficiently in a micro-control based on the asynchronous array than in the conventional micro-control as illustrated in Figure 5, which shows

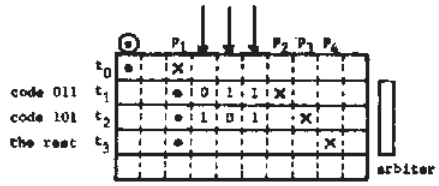


Figure 5. An efficient decoding structure

a decoding structure which sends the control to place p_2 if the code is 011, to place p_3 if the code is 101 and to place p_4 if the code is any of the six remaining possibilities. This structure makes use of the fact that the arbiter assigns higher priority to the transitions closer to the top. Transition t_3 is selected only when none of the transitions above it are enabled, thus t_3 takes care of all configurations that do not appear in the other transitions controlled by the arbiter. Thus we save on the number of rows needed to perform the decoding operation. In the conventional control one would have had to test the bits of the field one at a time or use a multiple-branch micro-instruction which would have required at least eight micro-words to account for all the possible locations to which the control could go as a result of the execution of that branch instruction.

Controlled Access to Shared Resources

Suppose there are six users who could place requests for a common resource. To control access to the resource, we will arrange the control in such a way that the token in a place, p_1, (Figure 6), will represent the resource. There

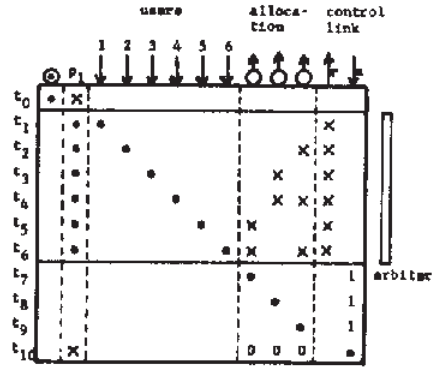


Figure 6. Allocation of shared resource

will be one input place for each user and the transition which allocates the resource to a user will have inputs from place p_1 and the input place of that user. Firing of any allocating transition will remove the token from place p_1 making the resource unavailable to the other users. The fact that the allocation has been made to a particular user will be conveyed to the outside by setting appropriate bits in a three-bit field and then sending a control signal on a control link. When the use of the resource is completed, an acknowledge signal will be returned on the control link. This acknowledge signal will lead to the firing of the necessary transitions to clear the field. When the field is cleared, transition t_{10} will put a token in place p_1 to indicate that the resource is free.

The conflicts over the resource (the token at place p_1) are resolved by an arbiter which arbitrates among all of the allocating transitions. If more than one user is waiting for the resource, the one which has the highest priority, the one whose allocating transition is closest to the top will be granted the resource by the arbiter. Needless to say, this simple priority structure does not achieve fair allocation because a low priority user may be indefinitely discriminated against if some higher priority users are always waiting.

Fair Allocation

The fairness of allocation entirely depends on what is considered fair; to obtain a fair allocator one must specify the criteria of fairness and then devise an appropriate allocator with places, transitions and arbiters to realize the characteristics of the fair allocation. For example, an interesting round robin allocator is shown in Figure 7. In this allocator the allocation proceeds from user 1 to user 4. If the arbiter that resolves the conflict is of the kind that operates in parallel, then because of the associative nature of the array there is no sequential hunting of the users involved in determining the next user.

Places p_1, p_2, p_3 and p_4 indicate the last user who used the resource; p_1, for example, indicates that user 1 was the last user. A token in place p_5 represents the availability of the resource. Transitions t_1, ..., t_8 perform the allocation, and transitions t_8, ..., t_{15} update the record about who was the last user. The cell configurations of the cells at the intersections of transitions t_1, ..., t_7 and places p_1, p_2, p_3, p_4 are arranged such that if place p_i has a token indicating that user i was the last user then user $i+1$ modulo 4 has the highest priority in the next allocation.

An Efficient Program to Clear a Field

Transitions t_7, t_8, t_9 and t_{10} in Figure 6 perperform the task of clearing a field and that of putting a token in place p_1 to signal the completion of the task. This control program uses only $n+1$ rows where n is the number of bits in the field.

Multiple Resource Allocator

In Figure 8 we show a six-user two-server allocator. Tokens in places p_1 and p_2 represent the

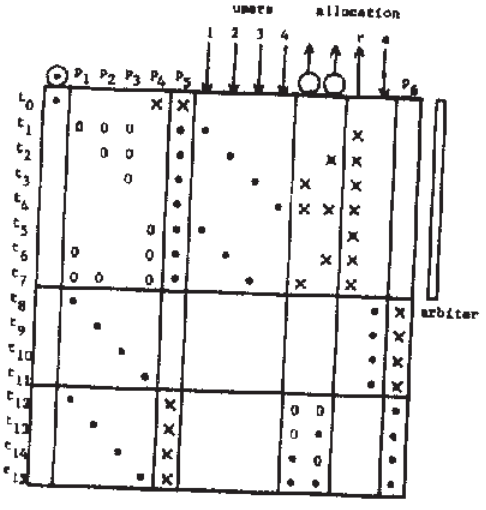


Figure 7. A round robin allocator

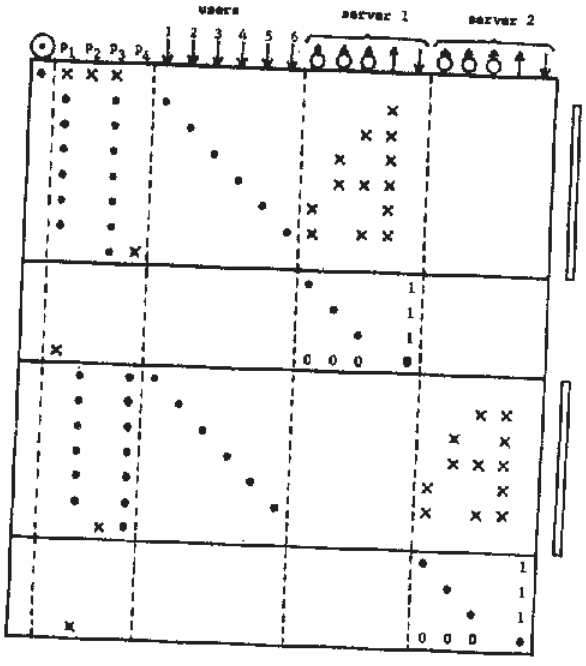


Figure 8. A simple 6-user 2-server allocator

availability of the two servers. The allocator actually consists of two parts, one which does the allocation for server 1 and another which does allocation for server 2. The control in the allocator oscillates between these two units; the presence of a token in place p_3 indicates that unit 1 is operating and a token in p_4 indicates that unit 2 is operating. After the operation of a unit, with or without allocation, the control passes to the other unit.

The allocator discussed above is a single sequence allocator. It is possible to realize a parallel allocator but that calls for a much larger control program. Some ideas on parallel allocators can be found in a paper on a multiple server arbiter (Patil [12]).

Sharing Micro-code

If a certain arrangement of places and transitions repeats in the control structure many times, then it can be set aside as a sub-control structure to be used as a subroutine if we know that at any one time the sub-control structure is needed only at one place. The illustration in Figure 9 shows how a group of places and transitions which account for the indirect addressing feature in a computer are used from two locations in the control, once to fetch operand A and then to fetch operand B. A 1 in the highest order bit of an address field indicates that this address is to be replaced by the address in the word that is addressed; the true address of the operand is found when the highest order bit is a 0. In the Petri net specification of the control, a place which appears split into two halves represents a pair of input-output places that correspond to the control link whose name appears besides the place. This notation is an abbreviation which makes the diagram more readable; it does not add any new capability to the Petri nets.

Structuring the Array for Efficient Utilization

We shall now turn our attention to a problem that arises in constructing a large micro-control if every column in the array realizes only a single place. As the control program becomes larger, more internal places are needed in the control, which calls for a wider array. Thus given a maximum width of the array, there is a limit to the size of the control program one can realize, unless a way is found for a column to be used for more than one place. For example, the columns could be broken into segments which realize distinct places. This idea is very much like the reuse of variable names in a block structured program, and not too difficult to attain if the array is provided with extra hardware so that a column can be terminated with a place circuit at a row in the array. It may be too expensive to provide this facility at every row in the array, but the ability to terminate a column at rows at selected intervals could prove quite satisfactory. Furthermore, the columns which are used to control the links, that is, the columns representing the input and output places, do not need this facility because for a fixed number of control points, the number of these columns remains fixed regardless of the size of the control program.

In a micro-control it is often necessary to sequence a number of instructions, which calls for the transitions to be connected by a chain of places. This specialized requirement can be very efficiently met by three columns that are divided into overlapping segments which in effect realizes a structure, which those familiar with Electrical Engineering will recognize as a three phase structure (Figure 10b). This three phase structure

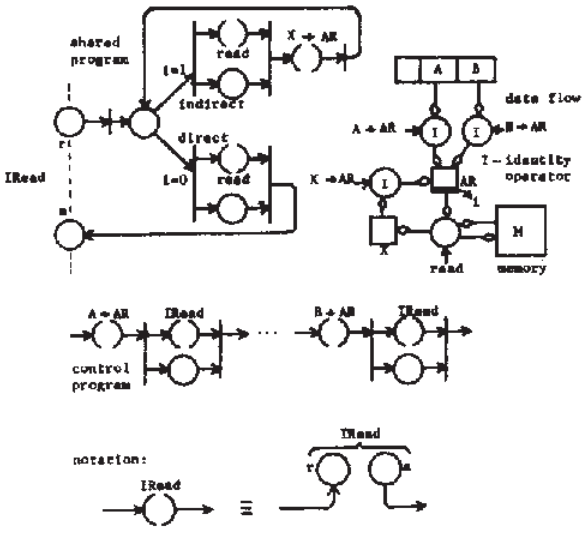


Figure 9. Sharing program

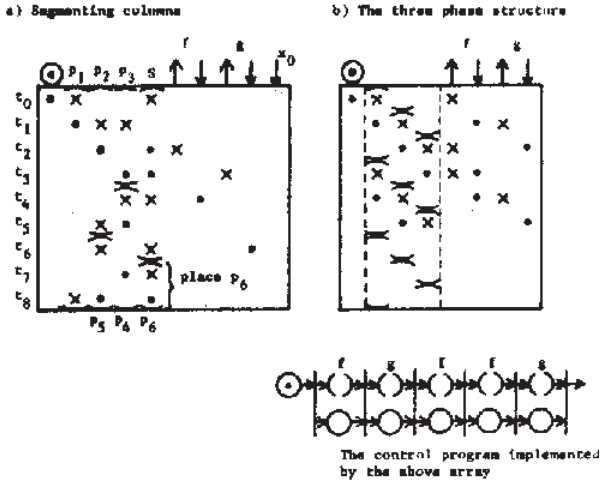


Figure 10. Structuring the array for efficient use

provides places which in addition to sequencing the instructions perform the task of remembering the locations to which the control from a subroutine is to be returned.

Concluding Remarks

Through the numerous examples that we have presented in this paper we have tried to convey our feeling that Petri nets are well suited for representing micro-control of parallel computers, especially when the computers are asynchronous computers. An important reason why this topic deserves attention is that now we have a systematic means of realizing the Petri-net specification of a control using a logic array based on LSI technology. The most important properties of a control implemented in this manner is that it is truly asynchronous and parallel in operation and is, therefore, well-matched to the needs of parallel computers. A demonstration model of a micro-control based on these ideas has been realized using discrete components.

We would like to express our gratitude to the National Science Foundation for their support of the research reported in this paper.

REFERENCES

[1] Bell, G. G., Grason, J., Newell, A., Designing Computers and Digital Systems, Digital Press, Maynard, Mass. (1972).

[2] Bruno, J., Altman, S. M., Asynchronous Control Networks, IEEE Conference Record, Tenth Annual Symposium on Switching and Automata Theory, 1969, 61-73.

[3] Clark, W. A., Macromodular Computer Systems, AFIPS Conference Proceedings 30, 335-336.

[4] Commoner, F. A., Holt, A. W., Even, S., Pnueli, A., Marked Directed Graphs, J. of Computer and System Sciences 5 (May 1971), 511-523.

[5] Dennis, J. B., Patil, S. S., Computation Structures. Notes for Subject 6.232, Department of Electrical Engineering, M.I.T., Cambridge, Mass., 1971. Most of the relevant concepts were included in the edition of the notes used for a Summer Conference at Princeton University 1968.

[6] Dennis, J. B., Modular, Asynchronous Control Structures for a High Performance Processor, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, 55-80.

[7] Holt, A. W., Introduction to Occurrence Systems, Associative Information Techniques (edited by E. L. Jacks), American Elsevier Publishing Co., 1971.

[8] Holt, A. W., Commoner, F., Events and Conditions, Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, 3-52.

[9] Jump, J. R., Asynchronous Control Arrays, IEEE Trans. on Computers C-23 (October 1974), 1020-1029.

[10] Noe, J. E., Nutt, G. J., Macro E-Nets for Representation of Parallel Systems, IEEE Trans. on Computers C-22 (August 1973), 718-727.

[11] Patil, S. S., Coordination of Asynchronous Events, Technical Report MAC-TR-72, Project MAC, M.I.T., Cambridge, Mass., June 1970.

[12] Patil, S. S., Forward Acting n x m Arbiter, Computation Structures Group Memo 67, Project MAC, M.I.T., Cambridge, Mass., June 1972.

[13] Patil, S. S., Dennis, J. B., Description and Realization of Digital Systems, Proceedings of the Sixth Annual IEEE Computer Society International Conference, September 1972.

[14] Patil, S. S., Synchronizers and Arbiters, Submitted to the IEEE Trans. on Computers, April 1974.

[15] Patil, S. S., Dennis, J. B., The Description and Realization of Digital Systems, Revue Française d'Automatique, Informatique et de Recherche Opérationnelle, February 1973, 55-59.

[16] Patil, S. S., Cellular Arrays for Asynchronous Control, Proceedings of ACM Micro-7, September 1974.

[17] Patil, S. S., An Asynchronous Logic Array, Computation Structures Group Memo 111-1, Project MAC, M.I.T., Cambridge, Mass., March 1975.