MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 101

The Design of a Highly Parallel Computer for Signal Processing Applications

by

Jack B. Dennis

David P. Misunas

The work reported here was supported by the National Science Foundation under research grants GJ-432 and GJ-34671.

August 1974

The Design of a Highly Parallel Computer for Signal Processing Applications

by

Jack B. Dennis and David P. Misunas

Abstract: A computer of unusual architecture is described that achieves highly parallel operation through use of a data-flow program representation. The machine is especially suited for signal processing computations such as waveform generation, modulation, and filtering, in which a group of operations must be performed once for each sample of the signals being processed. The difficulties of processor switching and memory/processor interconnection arising in attempts to adapt Von Neuman type computers for parallel operation are avoided by an organization in which sections of the machine communicate through transmission of information packets, and delays in packet transmission do not compromise effective utilization of the hardware. The major sections of the processor are specified as speed independent interconnections of a small set of asynchronous module types, and hence, the design concept is especially suited to implementation using asynchronous logic and large-scale integrated circuits.

Introduction

Highly parallel computers such as the Illiac IV [4] and the CDC Star [10] achieve their processing speed by imposing constraints on the structure of the data being processed. Both of these machines are organized to perform very well for data represented as vectors. To realize its potential, computation using the Illiac IV must be organized to exploit the machine's ability to execute the same operation simultaneously for many sets of operands. In the case of the CDC Star, the maximum processing rate of the pipelined processing unit is approached only if the computation is organized to make effective use of streaming operations on very long vectors. In both machines the programmer (or the compiler) is forced to use unusual and intricate data representations if highly parallel execution is to be achieved. Thus these machines are developments contrary to what is generally seen as one of the most important issues in contemporary computer practice -- the difficulty of developing correct programs. Even such an important notion as the use of subroutines is inadequately supported in these machines.

Other approaches to parallel processing in practical computer systems have not proved to be successful for highly parallel program execution: The classical multiprocessor computer system is limited in capability by the growth in complexity of the memory-processor switch as system size increases. Also, the problem of compiling user programs into concurrently executable parts of sufficient size that processor switching cost is acceptable is very difficult in the absence of unnatural constraints on the user language. Large processors like the CDC 6600 and the IBM 360/91 achieve parallel instruction execution by discovering absence of data dependency in instructions of a sequential program. The degree of parallelism achievable in this way has proved to be small.

We have been studying concepts of computer organization that can yield highly parallel program execution but with no sacrifice in the generality and ease of programming in the language supported. The ultimate goal of this work is a computer architecture able to achieve highly parallel execution of programs expressed in a user language such as CLU [14] which embodies linguistic features designed to support the development of well structured programs. An outline of the concepts expected to be employed in such a computer has been given in [5]. A central idea is the use of a machine language that permits a simple mechanism for identifying all instructions available for execution.

We have found that a program representation based on the concept of data flow is well suited for highly parallel program execution. In a data-flow representation, an instruction is enabled (that is, made available for execution) just when each required operand has been supplied by execution of predecessor instructions. Completion of instruction execution produces a result which is forwarded to each specified successor instruction for use as an operand. Data-flow representations for programs have been described by Karp and Miller [11], Rodriguez [21], Adams [1], Dennis and Fosseen [9], Bährs [3], Kosinski [12, 13], and Dennis [8].

In the present paper, we describe a machine capable of achieving highly parallel program execution for a special class of data-flow programs that correspond to the model of Karp and Miller [11]. These data-flow programs are well suited to representing signal processing computations such as waveform generation, modulation, and filtering, in which a group of operations is to be performed once for each sample (in time) of the signals being processed.

Our machine avoids the problems of processor switching and processor/memory interconnection present in attempts to adapt conventional Von Neuman type machines for parallel computation. In our desigr, processors in the usual sense do not exist. Sections of the machine communicate by the transmission of fixed size information packets, and the machine is organized so that the sections can tolerate delays in packet transmission without compromising effective utilization of the hardware. In future work we expect to further develop these ideas and investigate the feasibility of their application to the design of highly parallel computers using a generalized data-flow language such as described by Dennis [8], Kosinski [12, 13] and Bährs [3].

Part I: General Description

To illustrate the basic concepts of operation of the proposed processor, consider the data-flow program shown in Figure 1. This program represents the computation required for a second-order recursive digital filter

$$y(t) = A\,x(t) + B\,y(t-1) + C\,y(t-2)$$

where $x(t)$ and $y(t)$ denote input and output samples for time t. In this diagram, operators 2, 3 and 4 are unary operators that multiply by the fixed parameters A, B and C; operators 5 and 6 are binary operators that perform addition; and operator 7 is an identity operator that transmits its input values unchanged. Each small solid dot is a link that receives results from an operator and distributes them to other operators for use as operands. Input operator 1 represents a port through which an external stream of values that represent the input

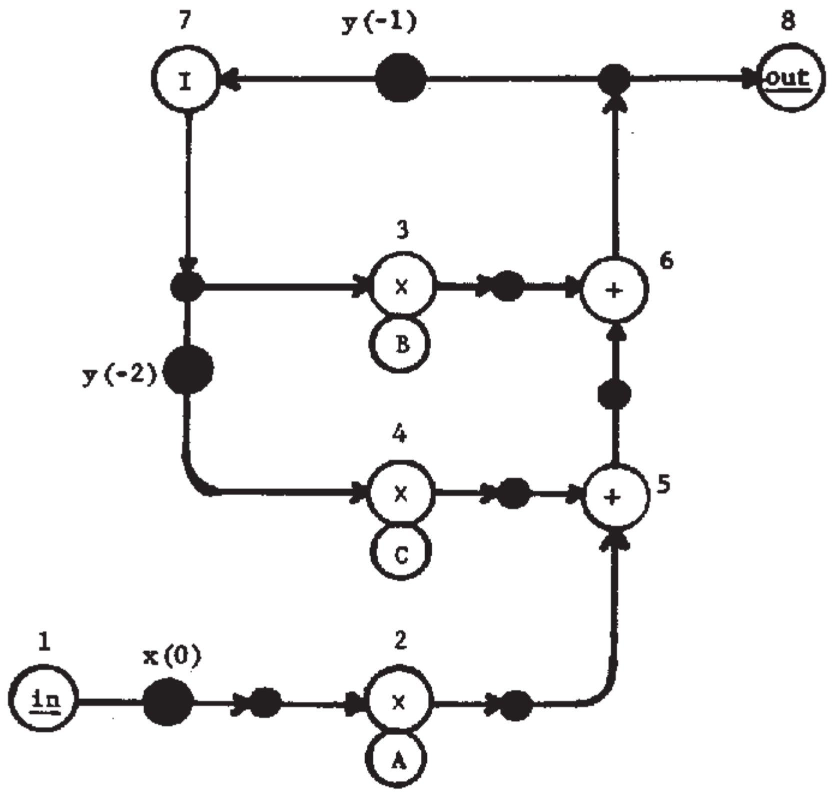


Figure 1. A data-flow program.

signal x(t) is presented to the program. Similarly, output operator 8 represents an output port at which the sequence of values representing y(t) is delivered during program execution.

The large solid dots (tokens) show the presence of values at certain input arcs of operators and define the initial configuration for program execution. An <u>operator</u> with tokens on each of its input arcs and no token on its output arc is <u>enabled</u>, and may <u>fire</u> by removing the tokens from its input arcs, computing a result using the values associated with the tokens, and associating the result with a token placed on the output arc of the operator. A <u>link</u> is <u>enabled</u> when a token is present on its input arc and no token is present on any of its output arcs. It fires by placing tokens on each of its output arcs and removing the token from its input arc. The new tokens distribute copies of the value associated with the input token over each output arc of the link.

The processor has seven major sections organized as shown in Figure 2:

Memory Section
Arbitration Network
Functional Units
Distribution Network
Controller
Command Network
Control Network

In addition, the block labeled "Host" represents a source of operating commands to he machine and could be either a manual console or a separate computer.

The design is conceived as using asynchronous communication of information packets between sections of the processor. Each connection is shown explicitly in Figure 2 and is independently coordinated using an acknowledge signal for each information packet transmitted. The arrowheads in the figure indicate direction of packet flow.

The information units transmitted through the Arbitration Network from the Memory Section to the Functional Units are <u>instruction packets</u>; each instruction packet specifies one unit of work for the Functional Unit to which it is directed. The information units sent through the Distribution Network from Functional Units to the Memory Section are <u>result packets</u>; each result packet delivers a newly computed value to a specific register in the Memory Section.

The Memory Section of the processor holds a representation of the program to be executed and holds computed values awaiting use. The Memory is a collection of Cells; one Cell must be associated with each operator of the program.

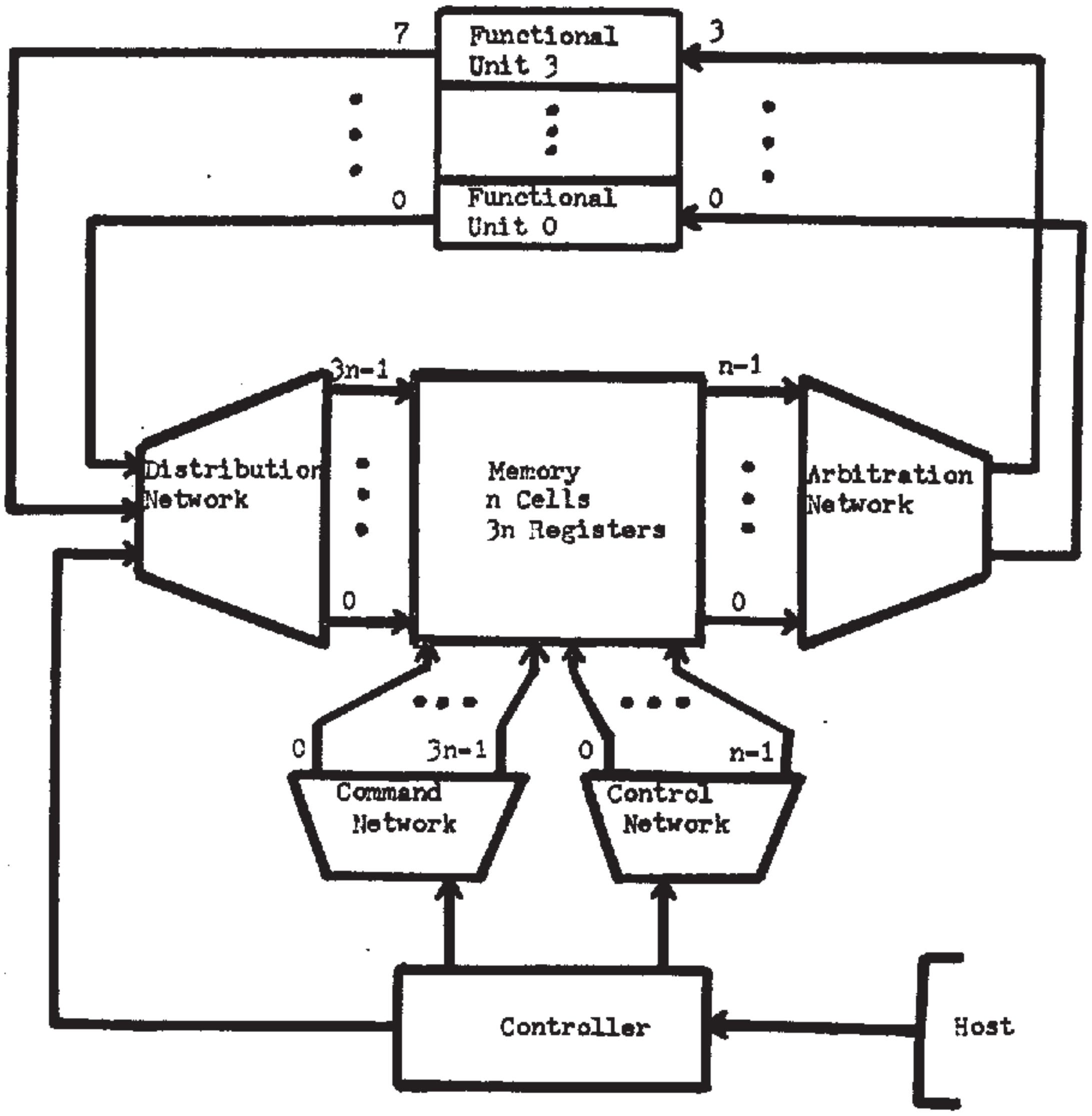


Figure 2. General organization of the processor.

Each Cell (Figure 3) contains three registers -- one register to hold an *instruction* which encodes the type of operator and its connections to other operators of the program, and two registers that receive operand values for use in the next execution of the instruction. Each register may be set to behave as a constant or as a variable. If set to act as a variable, a register expects to receive a result packet containing an operand value through the Distribution Network; if set to act as a constant, a register retains the value delivered to it when the program was loaded into the Memory. The instruction register of a Cell is normally set to hold a constant. When all three registers of a Cell are full, the Cell is said to be *enabled* and the contents of the three registers (instruction and two operand values) form an instruction packet which is presented to the Arbitration Network.

Figure 4 shows the instruction format. Assuming four Functional Units, the operation code has a field of two bits which specifies the Functional Unit required, and a field that indicates which specialized capability of the Functional Unit is to be used. Each destination field contains the address of a memory register which is to receive one copy of each result generated by execution of the instruction.

For digital filtering according to the data-flow program of Figure 1, eight Memory Cells may be set up as shown in Figure 5 to hold instructions and initial values for the computation; the Cells are numbered to correspond with the numbering of operators in Figure 1. Empty parentheses in an operand register indicate that the Cell is waiting for a value to be delivered to the register; a dash in a register means that the register is *idle* and does not have to be filled to enable the Cell. In Cells 1 and 8, which hold the input and output instructions, one operand register is shown holding a channel designator which may be regarded as an extension of the operation code. Let us trace the events that occur during execution of one instruction. In Figure 5, Cell 2 is enabled and contains the instruction

mult, 13, —

in which the dash indicates there is no second destination address. The Cell presents the contents of its three registers to the Arbitration Network as the instruction packet

$$\left\{\begin{array}{lll} \underline{\text{mult}}, & 13, & — \\ x(0) & & \\ A & & \end{array}\right\}$$

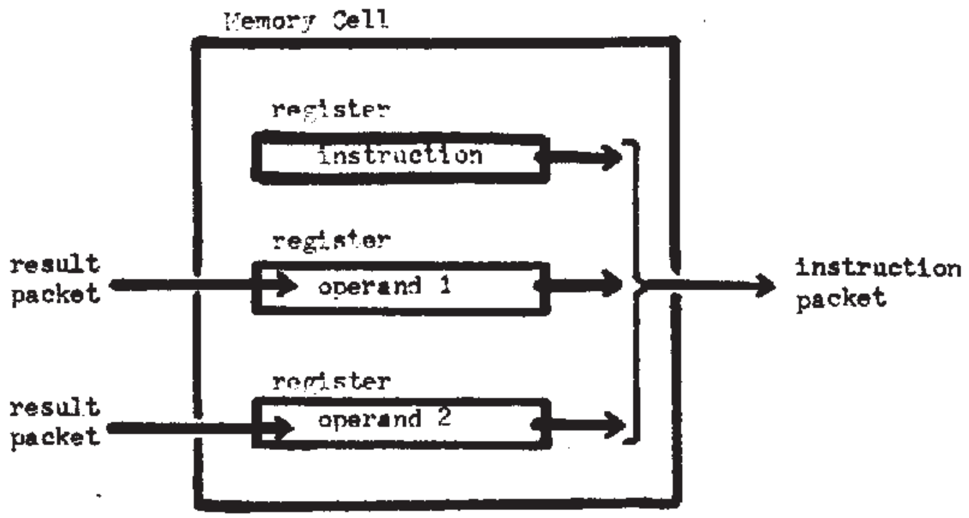


Figure 3. Operation of a Memory Cell.

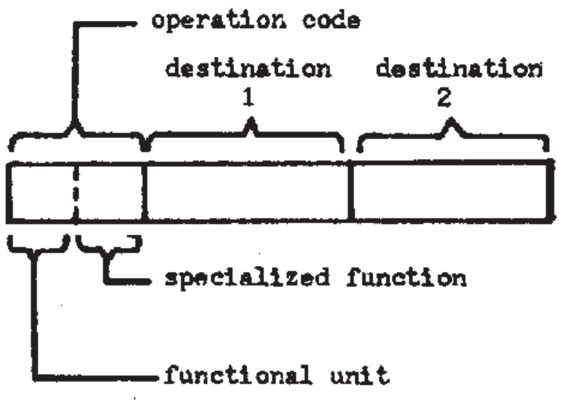


Figure 4. Instruction format.

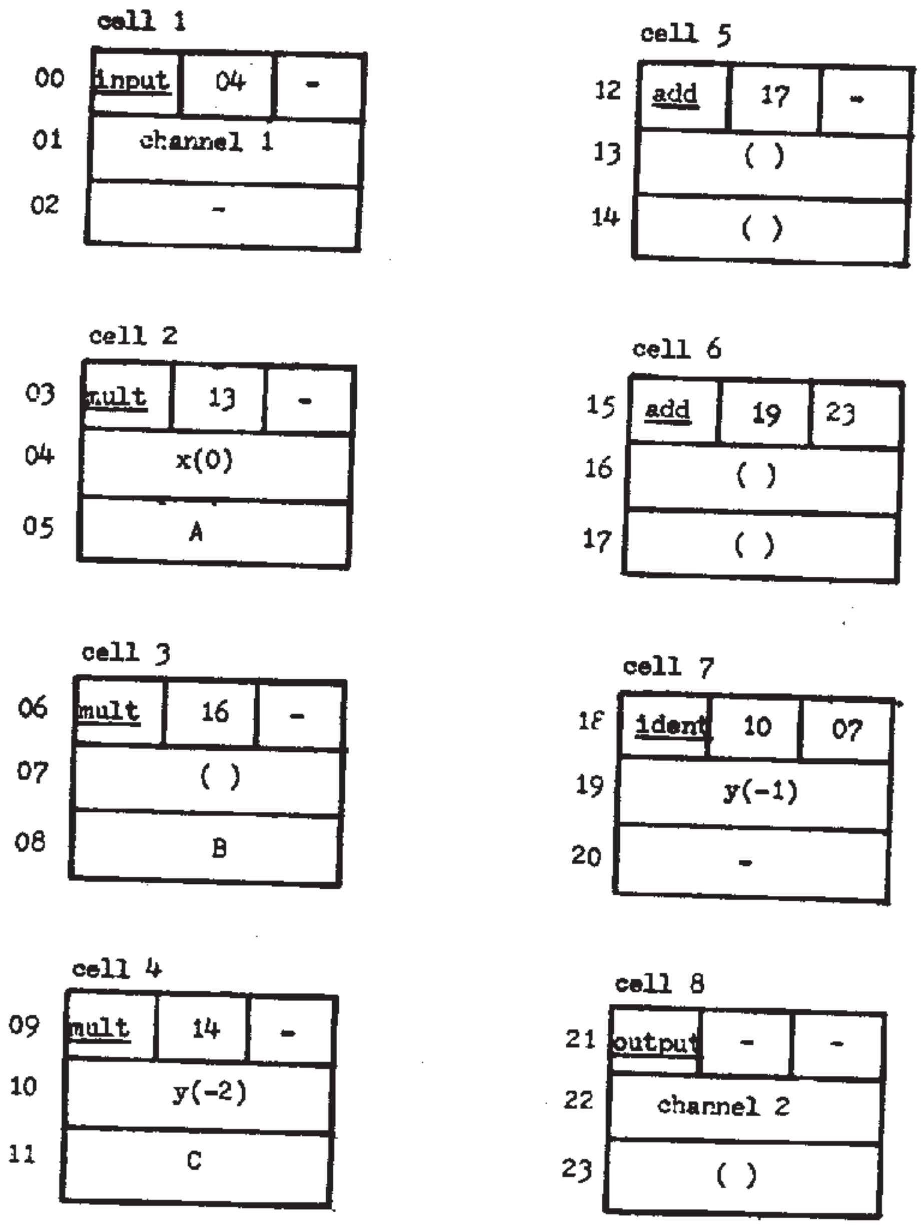


Figure 5. Initialization of Memory Cells for the digital filter computation.

The Arbitration Network routes the packet according to the Functional Unit bits of its operation code mult which direct the packet to a Functional Unit able to perform multiplication. The Functional Unit computes the product

$$z = A \times x(0)$$

forms the result packet

$$\begin{Bmatrix} 13 \\ 2 \end{Bmatrix}$$

by associating the computed value z with the destination address obtained from the instruction packet, and presents the result packet to the Distribution Network. The result value z is routed by the Distribution Network to Memory Register 13 where it becomes one operand for the instruction

cell 5: add, 17, —

Execution of an instruction with two destination addresses, for example,

cell 6: add, 19, 23

will present two result packets for independent transmission through the Distribution Network.

The Functional Units of the processor may be constructed to operate in pipeline fashion as illustrated in Figure 6 to realize the best compromise between throughput and complexity. Operands x and y from each instruction packet are fed into the computational pipeline together with the function specialization code c. At the same time the two destination addresses are entered into a pair of identity pipelines and are later associated with the result z as it emerges from the computational pipeline. Then each destination address and a copy of the computed result z form a result packet for delivery to the Distribution Network.

Since the data-flow form of a program exposes many possibilities for concurrent execution of instructions, we can expect that many Cells in the Memory Section of the processor will be enabled at once. As the Functional Units have high potential throughput, we must show how the Arbitration and Distribution Networks can be organized to handle many packets concurrently so all sections of the processor are effectively utilized. The Arbitration Network is designed so many instruction packets may flow into it concurrently from Cells of the Memory Section, and merge into four streams of packets -- one for each Functional Unit. The network is built of the four types of units shown in Figure 7.

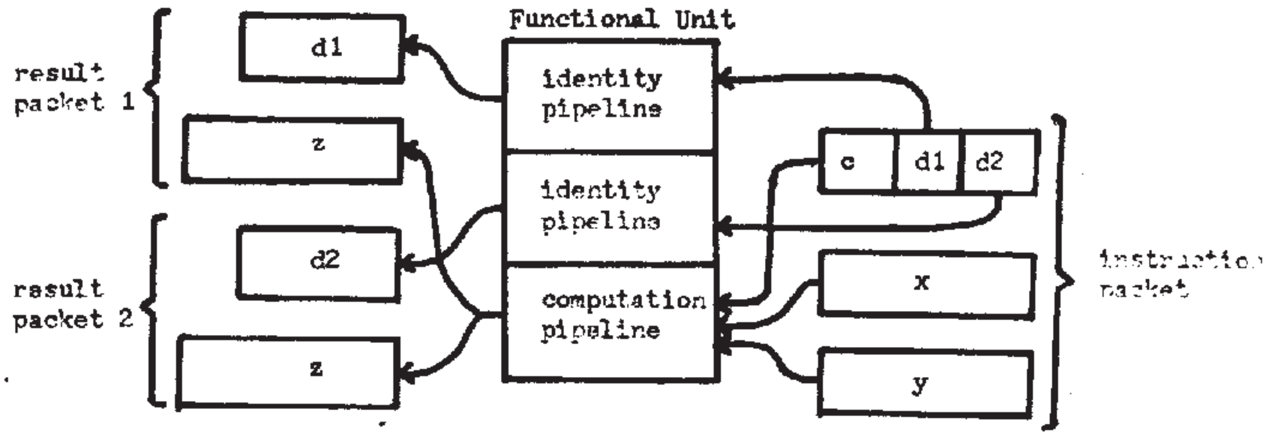
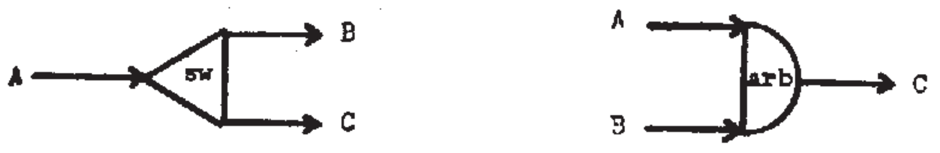
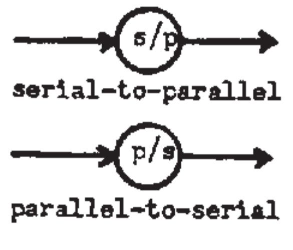


Figure 6. Pipeline operation of a Functional Unit.

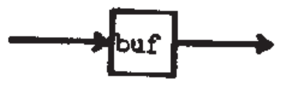


a. Switch Unit

b. Arbitration Unit



c. Conversion Units



d. Buffer Unit

Figure 7. Units for the Arbitration Network and Distribution Network.

The Arbitration Unit passes packets arriving at input ports A and B, one-at-a-time to output port C, using a round-robin discipline to resolve any ambiguity about which packet should be sent out next. The Switch Unit assigns packets arriving at port A to ports B or C according to some property of the packet. In the Arbitration Network, Switch Units separate instruction packets into four categories, one for each Functional Unit, by testing the operation codes of the instructions they contain. In the Distribution Network, Switch Units test successive bits of the destination address and direct each result packet to a specific register in the Memory.

Figure 8 shows how Arbitration Units and Switch Units might be arranged into an Arbitration Network. This network contains a unique path for instruction packets from each Memory Cell to each Functional Unit.

Since the Arbitration Network has many input ports and only four output ports, the rate of packet flow will be much greater at the output ports. Thus, a serial representation of packets is appropriate at the input ports to minimize the number of connections to the Memory Section, but a more parallel representation is required at the output ports so a high throughput may be achieved. Evidently, serial-to-parallel conversion is required within the Arbitration Network, and Conversion Units (Figure 7c) must be included. In addition, a packet emerging from a Conversion Unit must be prevented from engaging a subsequent Arbitration Unit until the serial packet has been completely absorbed by the Conversion Unit. Thus, Buffer Units are needed at the output of each converter. Figure 9 shows an improved Arbitration Network including Conversion Units and Buffer Units.

The Distribution Network is similarly organized. As shown in Figure 10, many Switch Units route result packets to the Memory Registers specified by their destination addresses. A few Arbitration Units are required so result packets with different destination codes may enter the Distribution Network concurrently and share use of the second rank of Switch Units.

The Host computer controls operation of the processor by giving commands to the Controller section of the processor. Four types of commands provide the means for setting up or changing the status and contents of Memory Registers, either at the beginning or at intermediate stages of a computation:

1. enter-constant a, v -- enter the value v in the Memory Register with address a, and set the Register to act as a constant.
2. enter-variable a, v -- enter the value v in the Memory Register with address a, and set the Register to act as a variable.
3. empty a -- empty the Memory Register with address a, and set the Register to act as a variable.

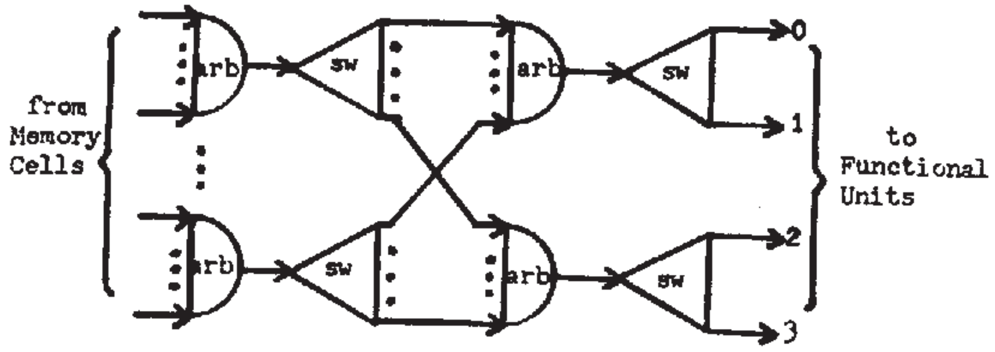


Figure 8. Primitive Arbitration Network.

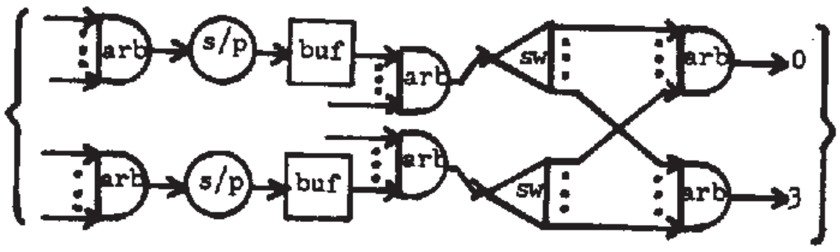


Figure 9. Improved Arbitration Network.

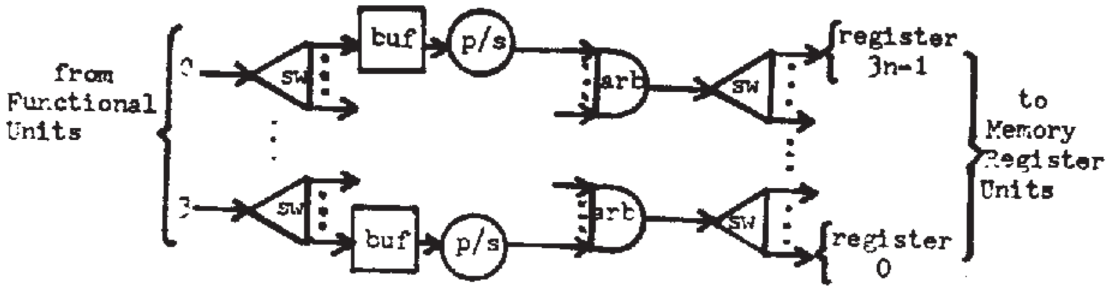


Figure 10. Distribution Network.

4. idle a -- set the status of the Memory Register with address a to idle.

For each command, the specified register is notified of the command type through the Command Network. In addition, the first two commands cause the value v to be sent through the Distribution Network as a result packet with address a.

A fifth command type provides control over execution of the data-flow program represented in the Memory Cells:

5. run v -- run the processor for v execution cycles of each instruction in an active (not idle) Memory Cell.

A run v command is distributed to the Memory Cells by sending v enable signals through the Control Network of the processor. The Controller waits for acknowledgement that all v execution cycles have completed before performing further commands.

The Host is responsible for requesting execution cycles only when the Memory contains a valid program representation, for otherwise program execution would lead to deadlock and no acknowledge signal to the Host.

Part II: Processor Specification in Terms of Speed Independent Modules

The concepts of processor organization just presented are very attractive for application of techniques for speed independent logic design [6, 7, 16, 19] that we have evolved from the early work of Muller [17]. The Appendix contains a complete specification for each unit of the processor as a speed independent interconnection of a small set of basic asynchronous module types. Here we explain the fundamental ideas on which the specifications rest and discuss their application to a key element of the processor -- the Memory Cell.

Asynchronous Circuits

An asynchronous circuit is an interconnection of switching elements. Each switching element has a unique output wire which is a node of the circuit and is said to be driven by the switching element. Each input wire of each switching element comes from some node of the circuit. At any instant during operation of the circuit each node has an associated value, which is either 0 or 1, and is determined by the switching element that drives the node. A set of values for all nodes of a circuit is a total state of the circuit. The total state changes with time through the firing of switching elements; firing an element changes the value of its output node (from 0 to 1, or from 1 to 0). The conditions for which a switching element may fire may be specified by a specialized form of Karnaugh map called a transition matrix. Transition matrices for three common switching elements are given in Figure 11. Each cell of a transition matrix corresponds to a particular set of values for the input and output nodes of the switching element. Conditions for which a switching element is able to fire are indicated by the presence of an asterisk in the appropriate cell; the condition that holds after firing is shown by an arrow. The initial condition of a circuit is with all nodes at 0.

A system is speed-independent if correctness of its operation is unaffected by the presence of delay. Two kinds of delay are considered: delay in the connections between system components; and internal delays in the components. A circuit built as an interconnection of component circuits is called a type 1 speed independent circuit if arbitrary delays on the interconnecting wires do not affect correctness of circuit operation. A circuit is a type 2 speed independent circuit if its correct operation is not affected by arbitrary delays inserted at the output nodes of its component circuits.

It is attractive from an engineering viewpoint to construct systems as type 1 interconnections of components, for timing consideration may be ignored in designing the physical placement of the components. However, the switching elements

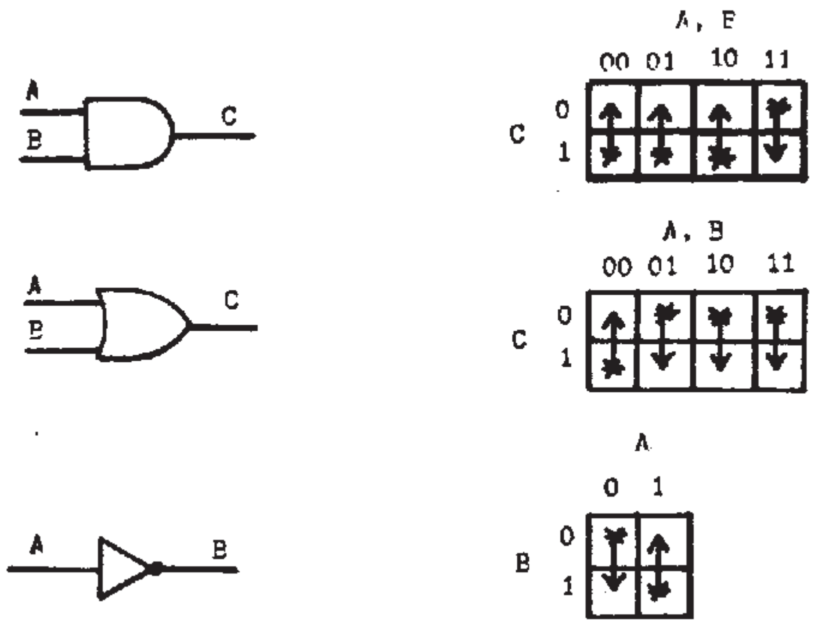
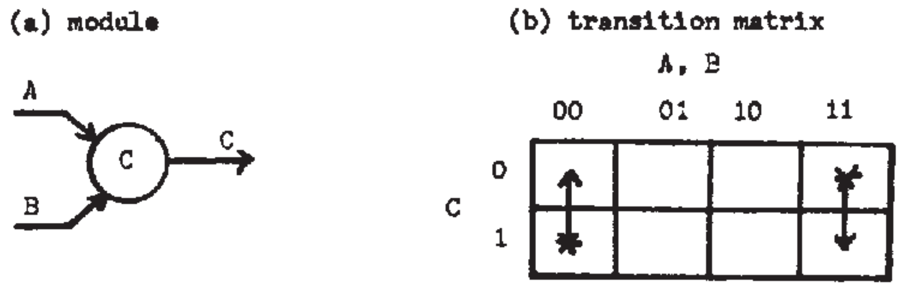


Figure 11. Transition matrices for elementary switching elements.



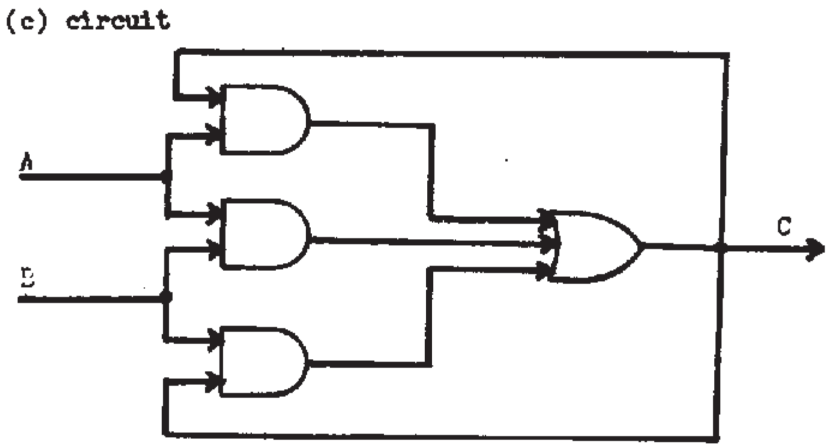


Figure 12. The C-module.

introduced so far do not provide an adequate basis for the description of systems as type 1 interconnections of components. In the following paragraphs we describe 14 kinds of basic modules which have proven to be useful in the formation of type 1 designs for the units of the processor.

In using interconnections of basic modules to specify units and sections of the processor, a communication discipline known as <u>reset signaling</u> is used: A 0 - to - 1 transition on a wire represents a meaningful event; we say that the wire has sent a <u>signal</u> from the module that drives the wire to the module to which the wire is connected. Before the wire can transmit a subsequent signal, a 1 - to - 0 transition must occur; this transition is called a <u>reset</u> of the wire. During the interval between a signal event and the following reset event, the wire is said to be <u>active</u>.

Basic Modules

An important switching element for speed independent systems is the C-module shown in Figure 12. The output node of a C-module becomes 1 when both inputs become 1, the output returns to 0 only when both inputs have become 0. The realization of the C-module shown in Figure 12 is neither a type 1 nor a type 2 connection of switching elements, yet its behavior will be consistent with the given transition matrix if delays within the circuit are properly controlled.

The use of the C-module as a synchronization element and as a buffer storage element is demonstrated in the design of the <u>data switch</u> (Figure 13). This module waits for signals to arrive on the enable wire E and one of input wires A1 and B1. It then sends a signal on the enable wire A2 or B2 corresponding to the active input wire. The output wire resets only when both the active input wire and the enable wire are reset. For speed independent operation, it is necessary to use the data switch module only where signals cannot arrive on both input wires A1 and B1 without an intervening reset.

The circuit for the data switch given in Figure 9 is a type 2 interconnection of switching elements because of the connection from the enable input to the two C-modules. Delay in the enable connection to the unused C-module may keep the C-module enable input from resetting in time to prevent a false output when an input signal arrives at that C-module. Nevertheless, if there is negligible delay in the wires, the circuit will exhibit the behavior specified by the transition diagram.

Data switch modules with more than two pairs of corresponding input and output wires are also used. A three-wire data switch is shown in Figure 14. For correct operation, only one of the three input wires A1, B1, C1 may be active at a time.

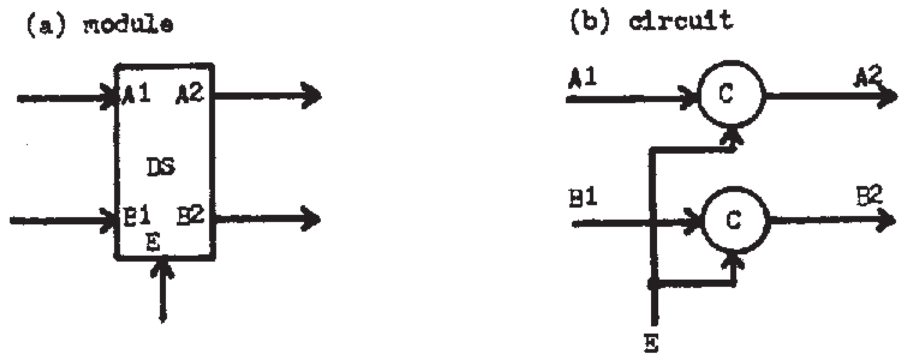


Figure 13. The data switch module.

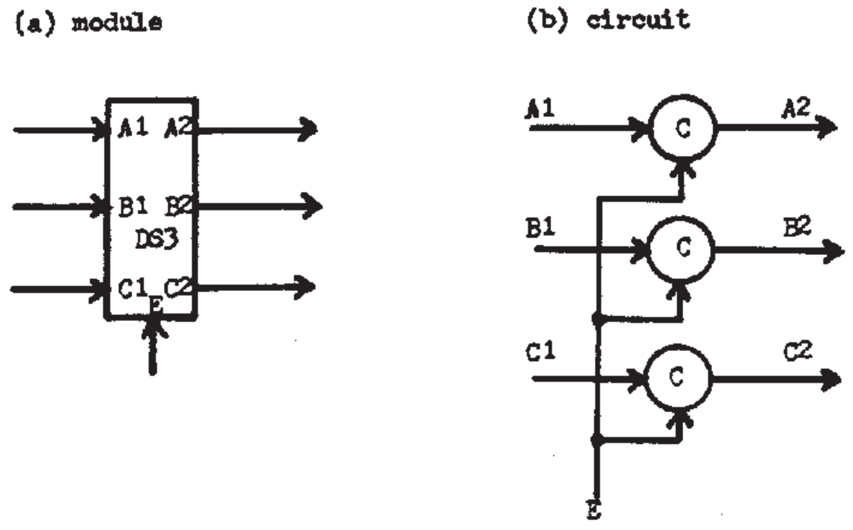


Figure 14. The three-wire data switch module

The multiple data switch shown in Figure 15 is three simple data switches combined into one module. In operation of the multiple data switch, the roles of the input wires and the enable wire of the data switch modules are reversed. A signal on input wires A0, B0, or C0 leads to a signal on one of the pair of corresponding output wires, according as an enable signal arrives on input E1 or E2. For correct operation, wires E1 and E2 must not be active at the same time.

The gate module shown in Figure 16 is a basic element for controlling parts of a system that perform cyclic activity. A channel for data flow from input D1 to output D2 is opened or closed by a signal at input S or input R, respectively. The data input wire D1 need not be 0 when input S is raised; data may be waiting at the input for the channel to be opened. However, data transmission must have stopped and the data input and output wires must reset to 0 before the channel is closed by the arrival of a signal at the reset input R. Also, inputs S and R must not be simultaneously active. The implementation shown uses the signals at inputs S and R to set and reset a flip-flop, which allows signal and reset events arriving at D1 to pass through to D2 when set, and allows no signals to pass when reset. The completion of opening or closing of the gate is acknowledged by a signal on wire AS or AR, respectively.

The select module (Figure 17) routes data events arriving at input D0 to output D1 or D2 according to whether the module was last set by a signal at enable input S1 or S2. Acknowledge signals at A1 and A2 confirm the setting of the module. Inputs S1 and S2 must not be simultaneously active, and input D0 and outputs D1, D2 must be 0 while the module is being set. The design is similar to that of the gate module.

Two conventional switching elements are important as modules in speed independent circuits: Figure 18 shows a convention used to represent the Boolean OR gate as a module. The OR module is drawn as a line on which lines representing input wires terminate in arrowheads. Only one input wire of an OR module may be active at a time; within this restriction, signal and reset events occurring at either input are transmitted at the output node. The NOT module performs the function of an inverter. Signal and reset events at the input node become reset and signal events, respectively, at the output node. The initial condition of a NOT module is with the input and output wires at 0. Thus, when a circuit is turned on each NOT module will be enabled, leading to activation of its output node; the NOT modules are the initiators of activity in a speed independent circuit.

The synchronization module (Figure 19) is used to coordinate action by two parts of a system so neither part continues before the other is ready. A signal

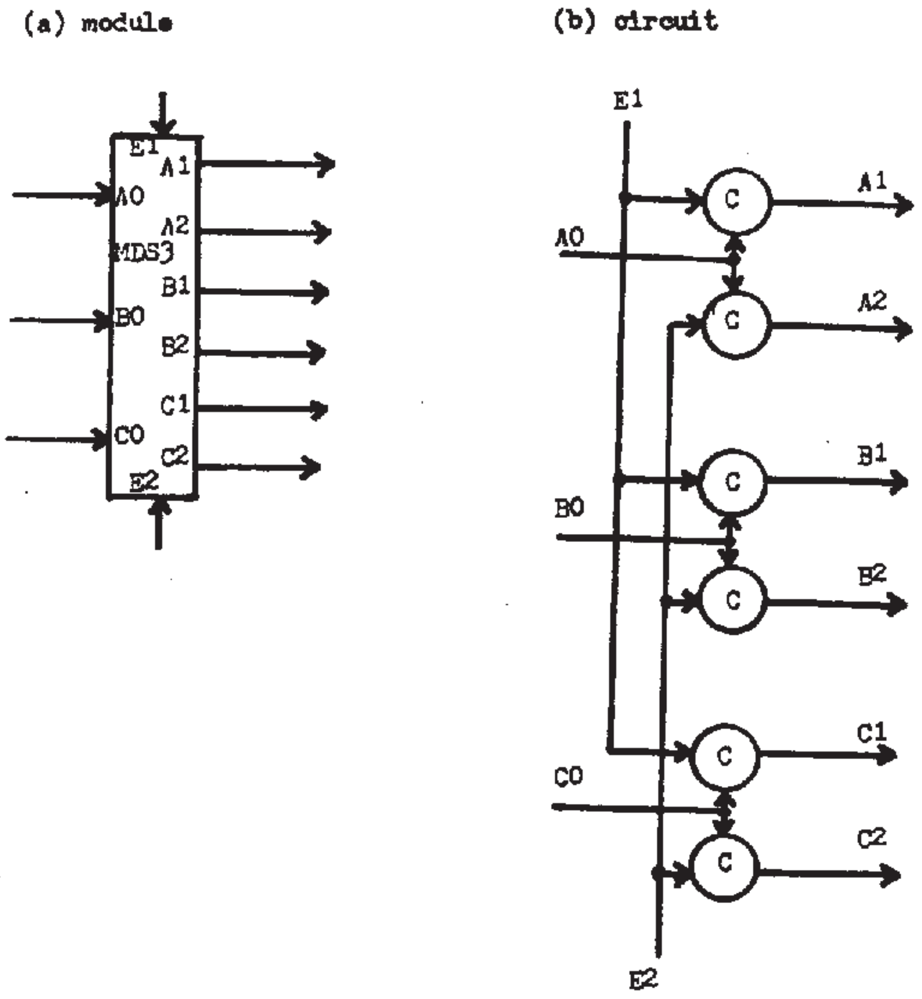


Figure 15. The multiple data switch module

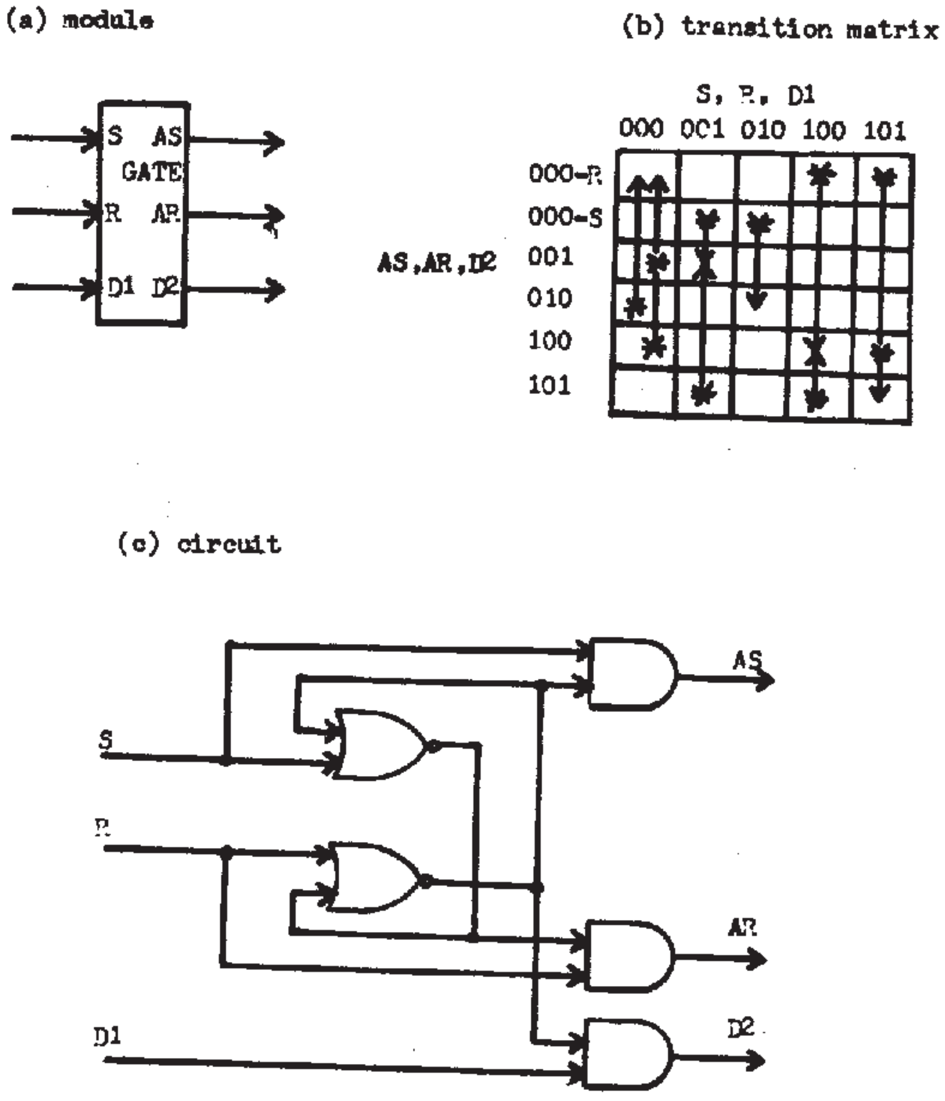


Figure 16. The gate module.

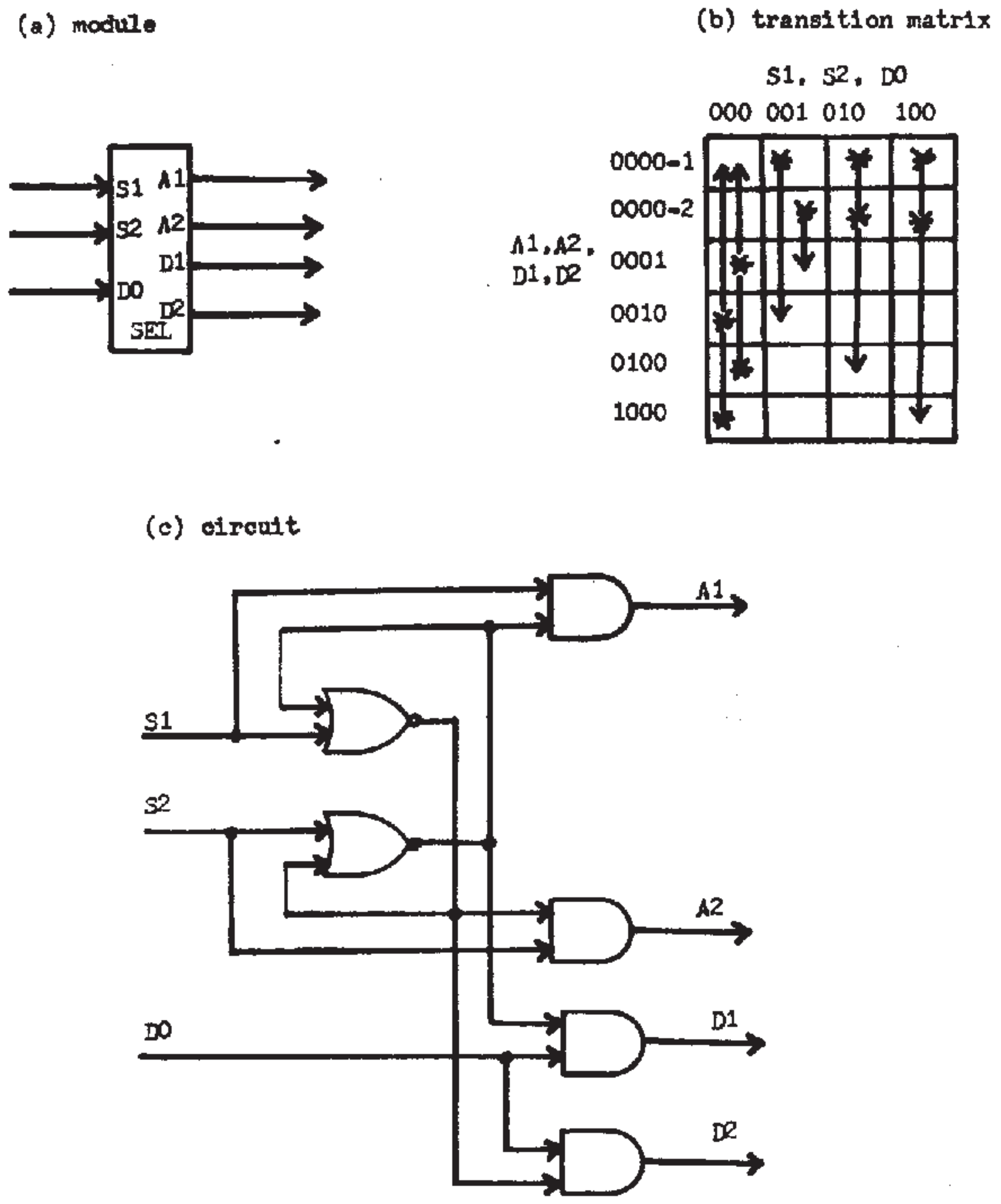


Figure 17. The select module.

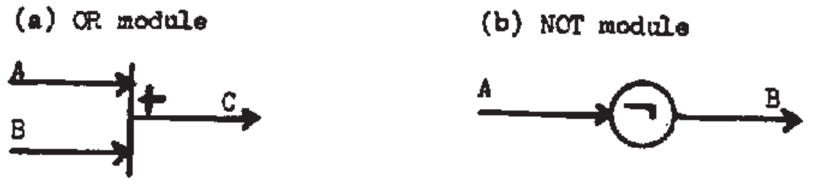


Figure 18. The OR and NOT modules.

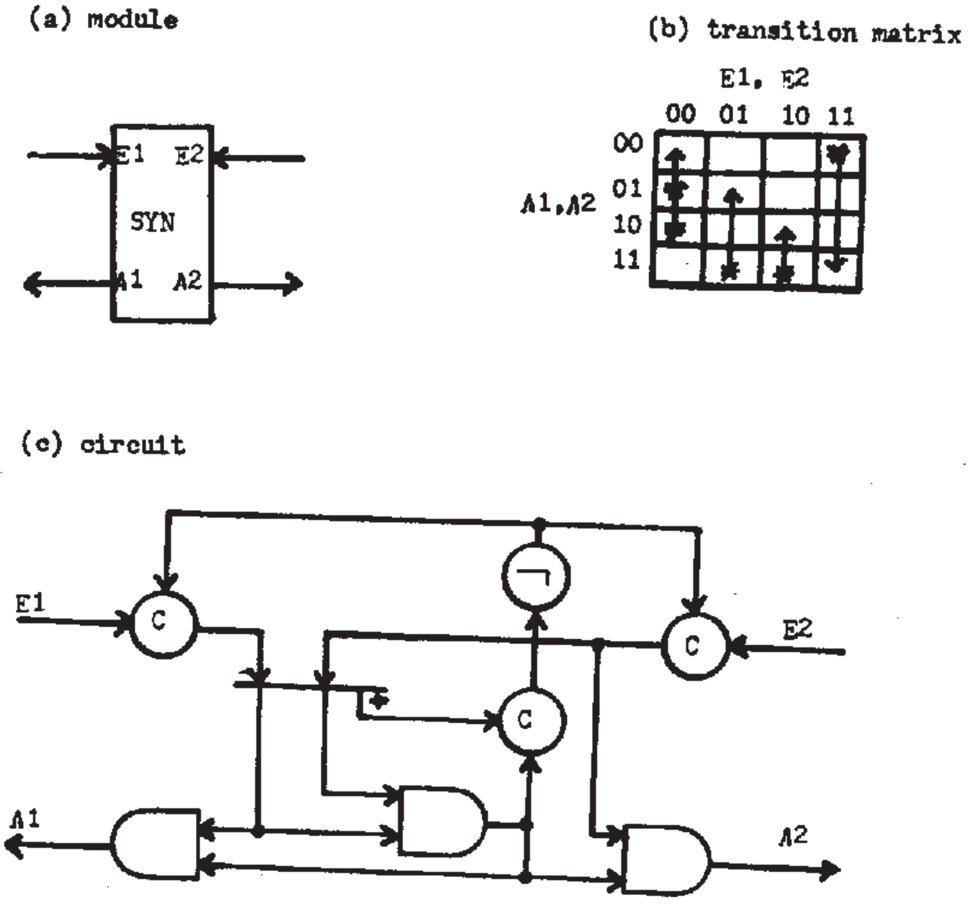


Figure 19. The synchronization module.

event must occur at input E1 and at input E2 before signals are transmitted at outputs A1 and A2. Each output resets only after its corresponding input has reset, and each input/output pair must reset before the two outputs can become 1 again. However, it is not necessary for both inputs to be 0 together. One can become 0 and then be set to 1 after its associated output has reset, yet the output will not become 1 again until the other input/output pair resets and the input becomes 1. When both inputs become 1, the three AND gates are enabled, and their outputs become 1. When either input returns to 0, the corresponding output will become 0, but the output node of the NOT module does not return to 1 until the two C-modules associated with inputs E1 and E2 have registered resets of their inputs. The synchronization module is a type 2 speed independent interconnection of its parts.

The signaling module shown in Figure 20 is used for controlling sequences of actions by parts of a system. The module acknowledges an enable signal on input S by a signal on AS only after a signal sent at output R is acknowledged by a signal on AR, and both R and AR have reset. Then a reset of S is followed by a reset of AS, with no activity at R and AR.

Initially, all wires are at 0, and the output of the C-module is 0, making one input of each AND gate active and making AS initially 0. A signal at S leads to a signal on R by firing the upper AND gate. An acknowledgement on AR enables the lower AND gate, and its firing allows the C-module and NOT module to fire, resetting both AND gates. The resetting of the upper AND gate causes a reset of R. The subsequent reset of AR completes an R/AR transaction and leads to an acknowledge signal on AS. Resetting S allows the C-module to reset its output leading to reset of AS, completing an S/AS transaction and returning the circuit to its original condition.

The signaling module is another example of a type 2 speed independent interconnection. The use of the AND gates within the circuit does not allow one to determine whether the reset of one or both inputs to the gate has occurred.

Activities by parts of a system may be sequenced by a cascade of signaling modules. The sequence module shown in Figure 21 is implemented in this way. The module operates by causing a cycle of signal and reset transitions of the R and AR wires of each SIG module before the next SIG module is enabled. When the final SIG module completes its cycle, an acknowledge signal occurs on AR. Wires R and AR reset without further transitions of the remaining input and output wires of the sequence module.

The signaling module also provides the basis for implementing counter modules (Figure 22). After receiving an enable signal on wire E, a CTR[n] module

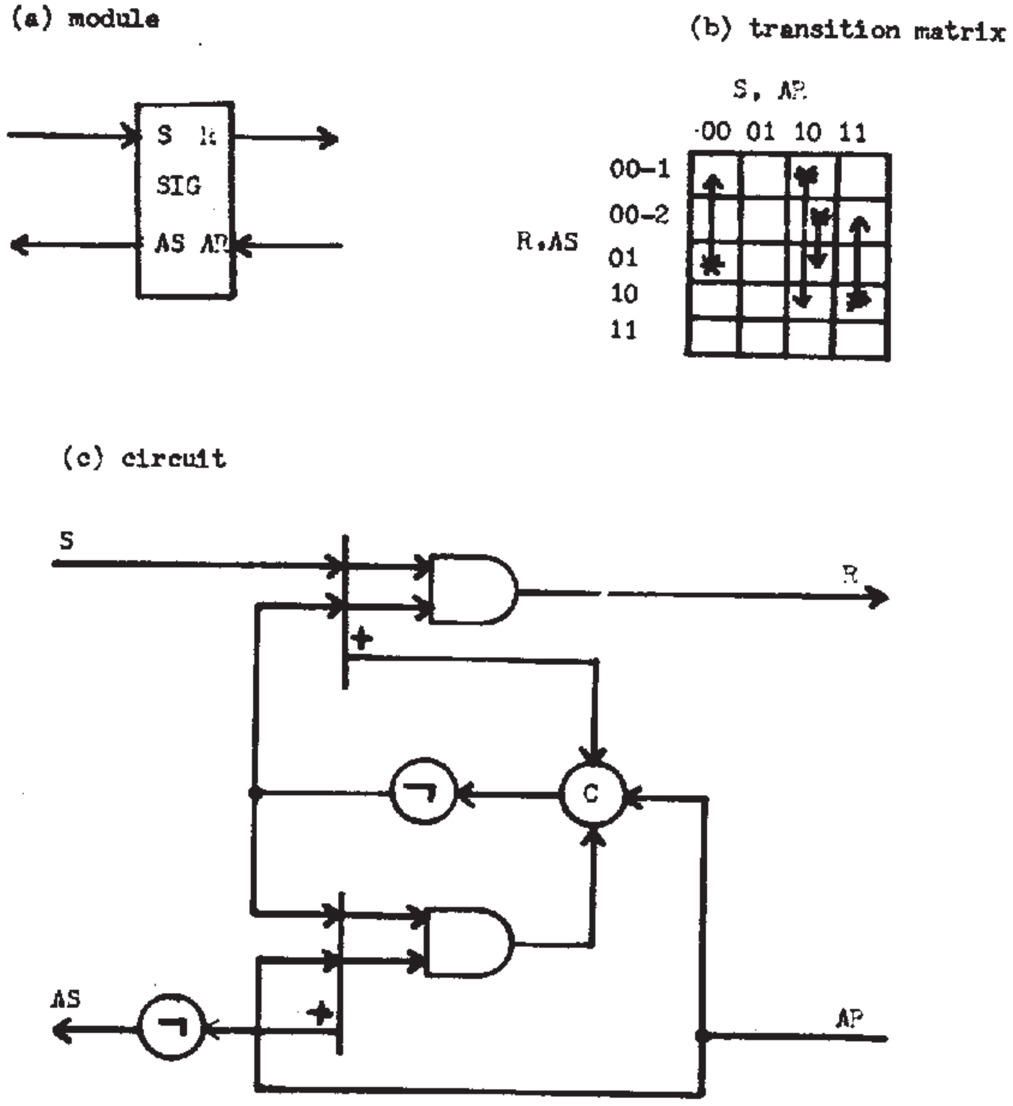


Figure 20. The signalling module.

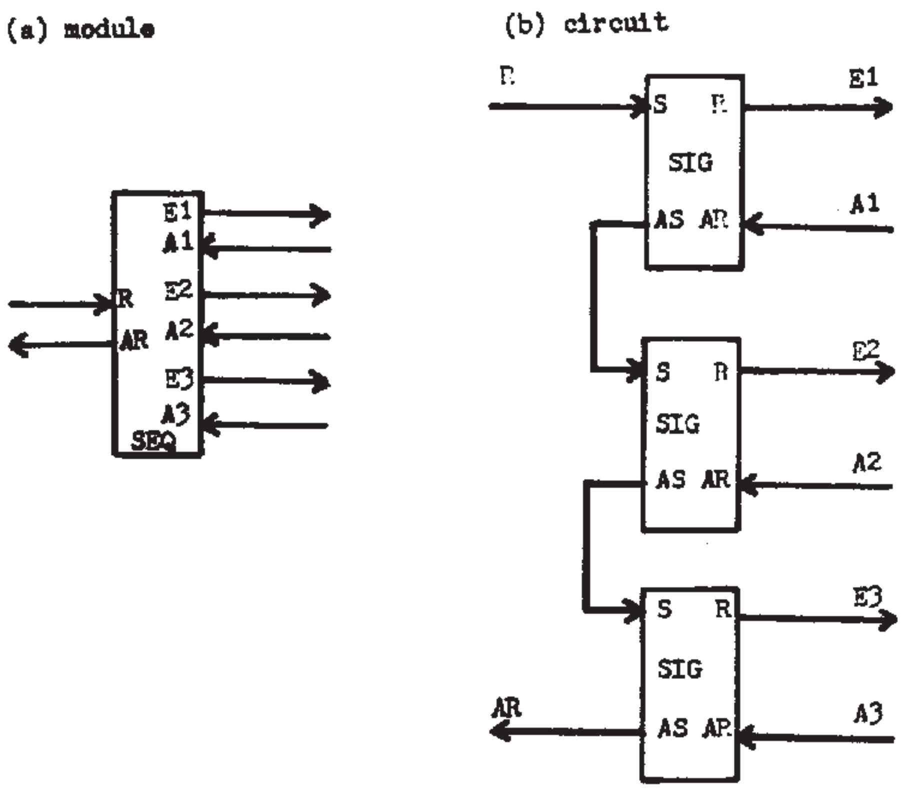


Figure 21. The sequence module.

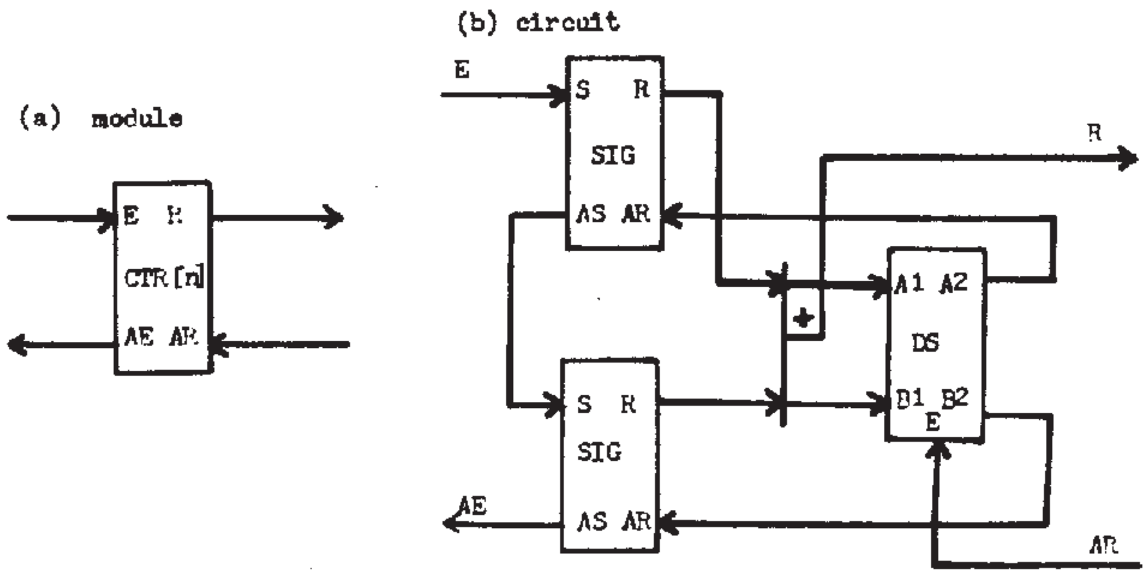


Figure 22. The counter module.

performs n complete cycles of signal and reset events on wires R and AR, and then returns an acknowledge signal on AE. Resetting E leads to resetting of AE without further transitions of wires R and AR.

The circuit shown is for one stage of a binary counter. The two SIG modules generate two enable signals on wire R in sequence through the OR module. Acknowledge signals on AR are directed to the appropriate SIG module by the data switch module. Since the upper SIG module must complete its cycle before the lower one is enabled, inputs A1 and B1 of the data switch will not be simultaneously active. Counter modules of n states may be constructed using various combinations of SIG, DS and OR modules.

The counter module and the sequence module are both type 1 speed independent interconnections of basic modules, so the correctness of their operation is unaffected by delays in the interconnections.

The Memory of a system can be constructed of asynchronous switching elements linked to other system components by a speed independent interconnection. One such memory element is the <u>bit pipeline module</u> shown in Figure 23. Signals on inputs E1 or E0 enable a 1 or 0 to be entered into the pipeline, after which an acknowledge signal is returned on wire A. Up to n bits may be entered into a pipeline module designated BP[2n] and having 2n sections. If the pipeline is not empty, a signal presented on wire E enables the pipeline to emit its longest held bit by a signal on output R1 or R0.

The pipeline module is constructed as a cascade of data switch modules with feedback connections using OR modules and NOT modules arranged so that bits advance through the data switches until all alternate stages hold information. As in the case of a data switch module, inputs E1 and E0 must not be simultaneously active.

Occasionally, a circuit is required that holds a queue of undistinguishable events. Such a circuit (Figure 24) is called an <u>event pipeline</u> and is similar in construction to the bit pipeline but uses half as many C-modules.

Often there is competition for resources between two units of a system. For example, two memory cells may wish to use the same functional unit at the same time. Such conflicts can be resolved by a switching element known as an elementary arbiter (Figure 25). The first signal to arrive at input A1 or B1 will set the latch and disable the other input. If both inputs arise simultaneously, one will succeed in disabling the other. The transition matrix of Figure 23b demonstrates the indeterminacy of this action; there are two possible transitions which can occur when both inputs go to 1.

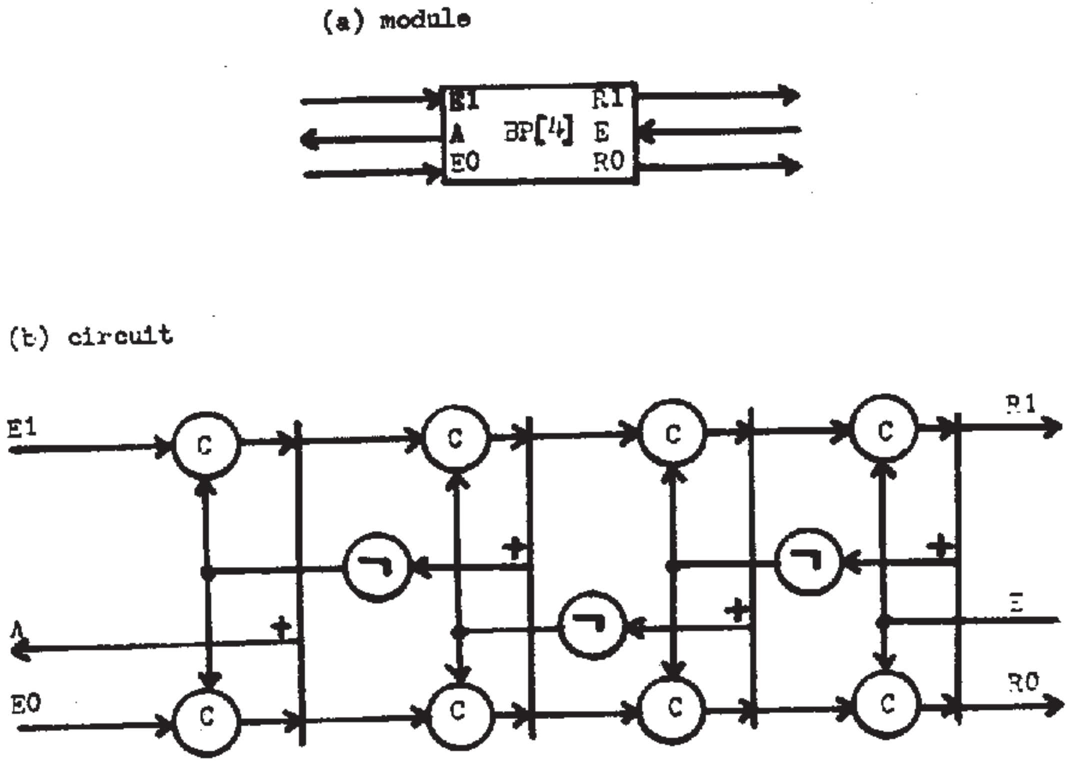


Figure 23. The bit pipeline module.

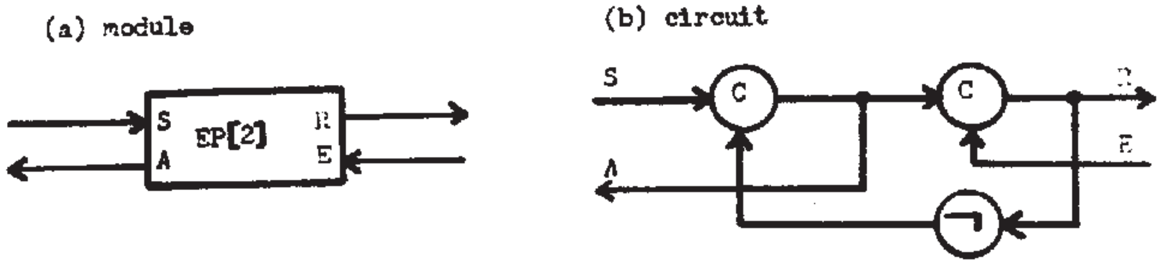


Figure 24. The event pipeline module.

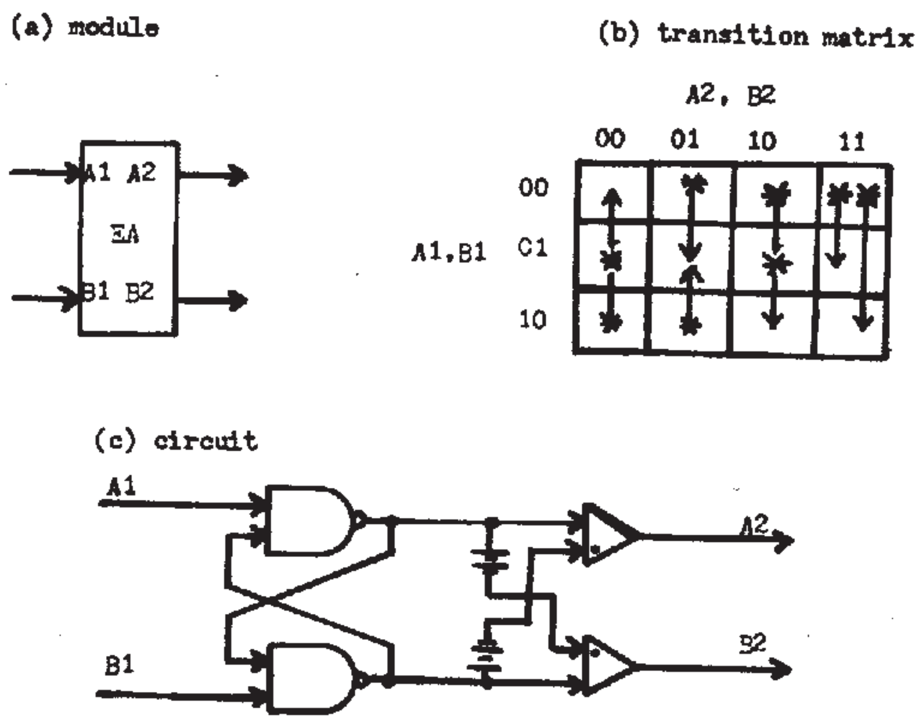


Figure 25. The elementary arbiter.

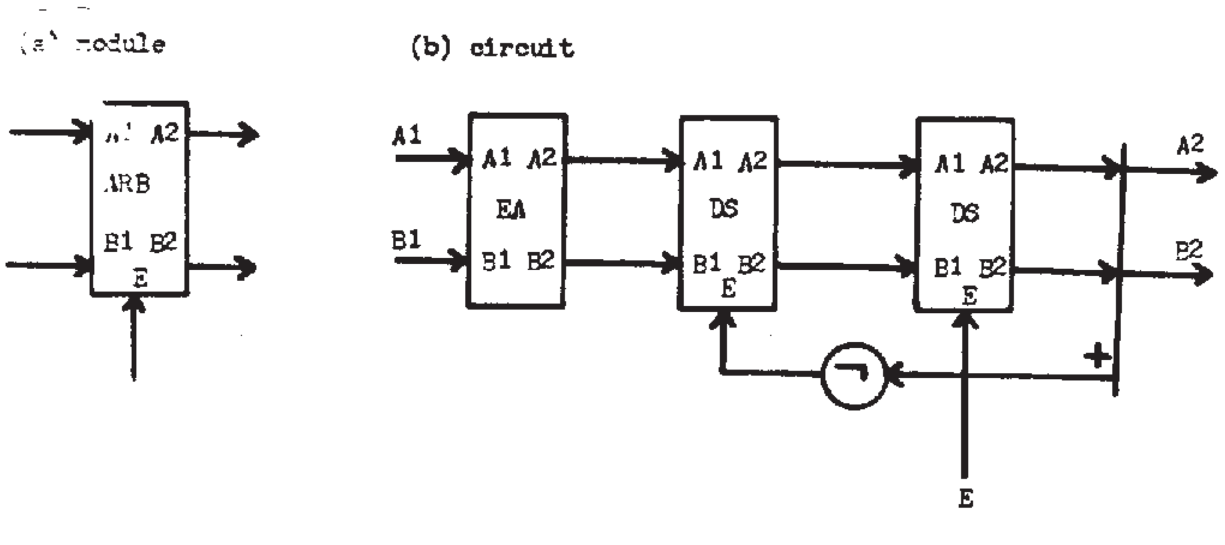


Figure 26. The arbitration module.

The difference amplifiers in the circuit produce an output signal when the two outputs of the latch differ by an amount greater than the fixed offset voltage. This prevents any false output signals that may result if the latch enters its metastable state. An analysis of the design of elementary arbiter circuits is presented by Patil [20].

The elementary arbiter of Figure 25 cannot be used to build larger circuits through speed independent interconnection because the reset of an active acknowledge wire is not forced to occur before the other acknowledge wire becomes active. This problem is resolved in the arbitration module shown in Figure 26, in which a signal at the enable input E is required before each transaction at either output, A2 or B2. A signal at input A1 or B1 is registered in the left hand data switch module upon arrival of an enable signal at E. If signals arrive at both A1 and B1, the elementary arbiter will allow only one at a time to be registered. Although the connections of the elementary arbiter to the left hand data switch are not speed independent, an arbitration module can be used as a component in type 1 speed independent circuits.

In the processor design, arbitration modules are used only in the Arbitration Units of the Arbitration Network and the Distribution Network.

Communication Between Units and Sections

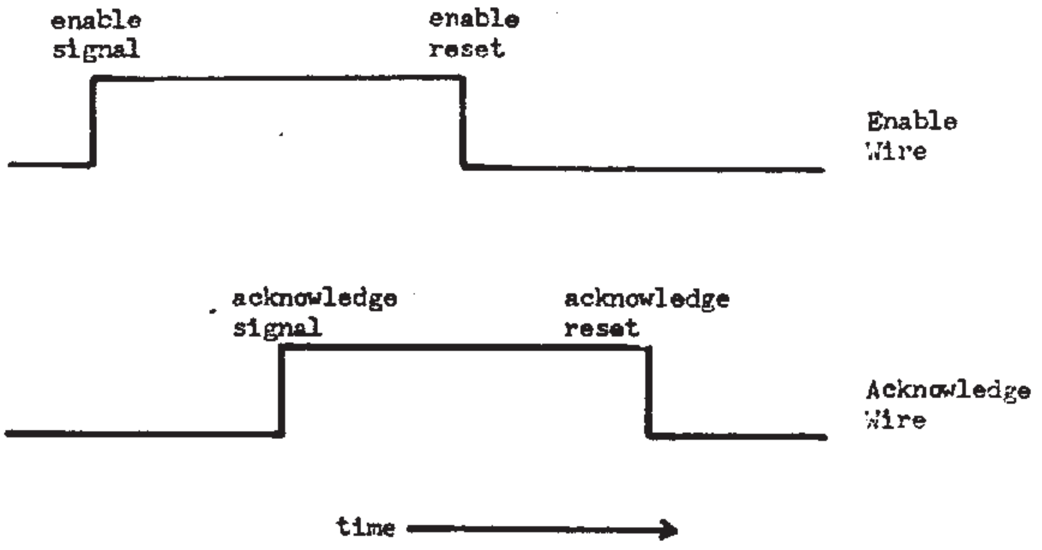
Communication between units and sections of the processor is, as we noted earlier, accomplished by transmission of packets of information. Packets are sent over paths called links, each link consists of one or more groups or wires within which a strict signaling discipline is observed.

In the quiescent condition of the processor (just turned on, or awaiting the next command from the Host), all wires are inactive. The wires of a group are divided into enable wires and acknowledge wires. The units of the processor are designed so that signaling in each group follows a repeated cycle of four steps:

1. Signals are sent over at least one enable wire.
2. Signals are returned over at least one acknowledge wire.
3. The enable wires used in step (1) are reset.
4. The acknowledge wires used in step (2) are reset.

All events of one step must occur before any event of the following step. A complete cycle of events on the wires of a group makes up a transaction. For example, the four events comprising a transaction on a two-wire group are as illustrated in Figure 27: enable signal, acknowledge signal, enable reset, acknowledge reset.

The structure of each type of link shown in Figure 28 is given in Figures 29 through 32, and the function of each type of link in the operation of the processor



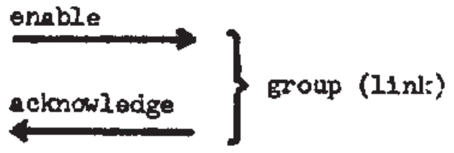


Figure 27. Link signalling discipline.

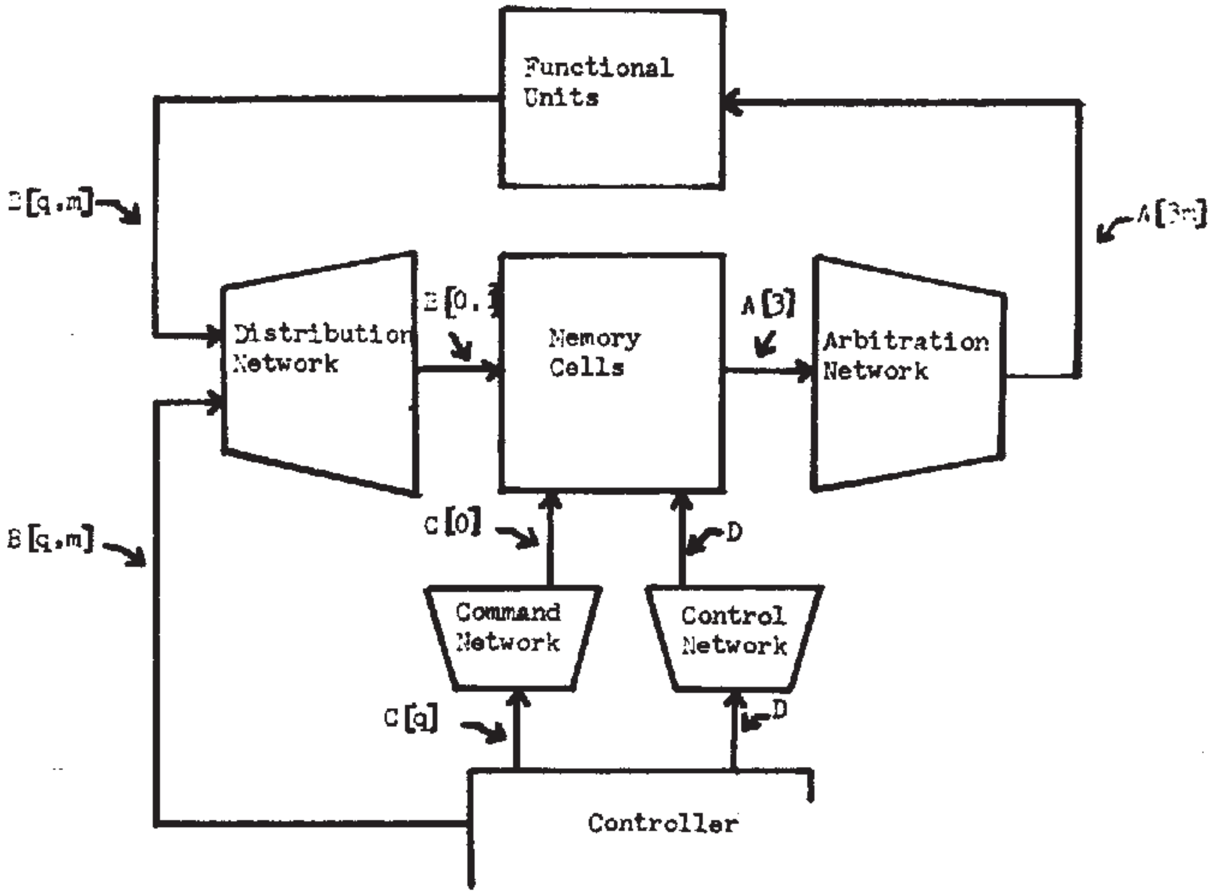


Figure 28. Link types for communication between sections of the processor.

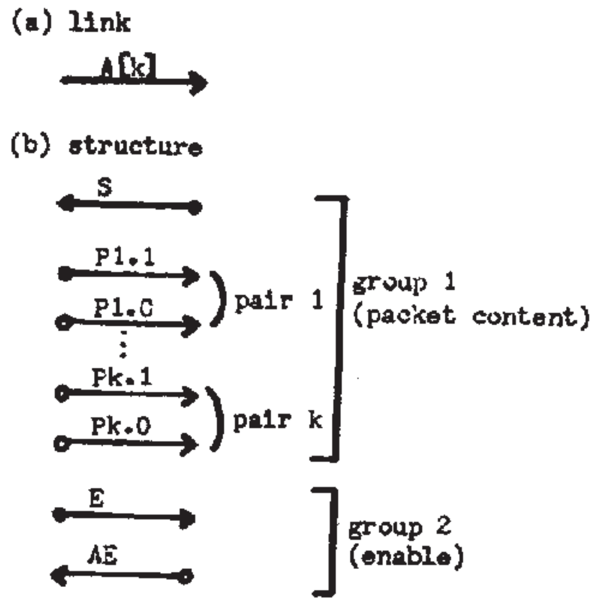


Figure 29. Structure of type A[k] link for instruction packet transmission.

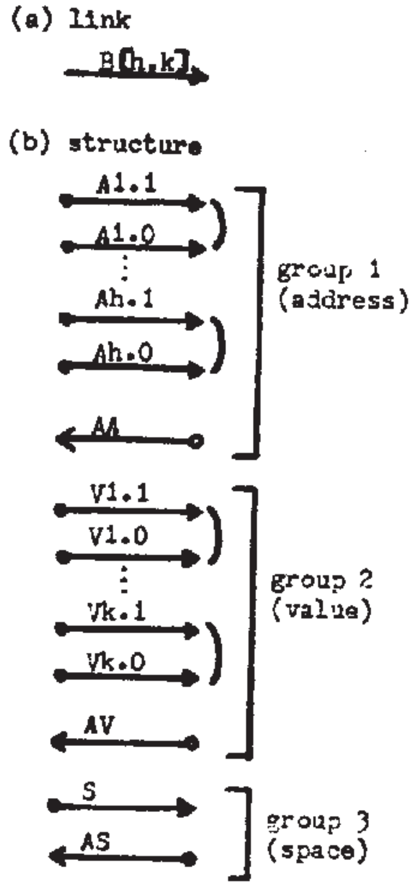


Figure 30. Structure of type B[h,k] link for result packet transmission.

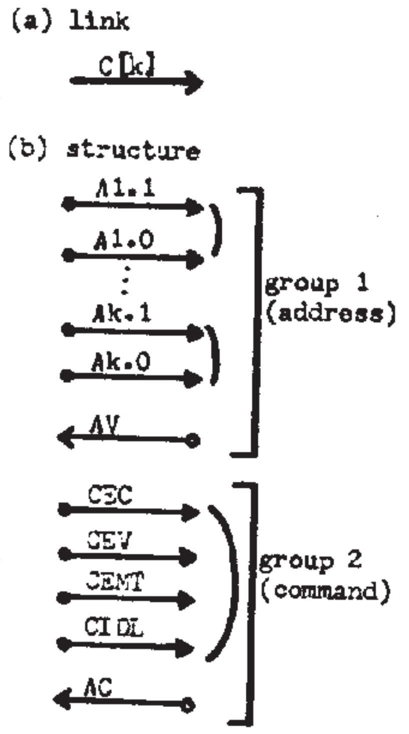


Figure 31. Structure of type C[k] link for command packet transmission.

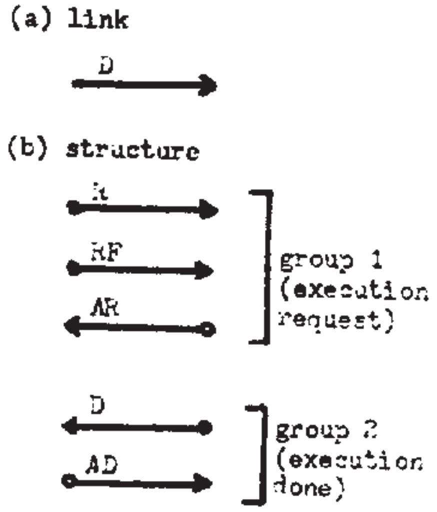


Figure 32. Structure of type D link for execution control.

is described below. Within each group the enable wires are distinguished by solid arrowheads. We suppose the Memory section consists of n Memory Cells, that each Register Unit holds an m-bit word, and that q-bit addresses are used to identify the Register Units of the Memory.

Instruction packets are transmitted between units of the Arbitration Network over type A[k] links having the structure shown in Figure 29, where k is the number of bits transmitted in parallel through the link. The links from Memory Cells to the Arbitration Network are of type A[3] (three bits at a time, one from each register), and the links from the Arbitration Network to each Functional Unit are of type A[3m].

Starting with all wires inactive, transmission of a k·p-bit instruction packet over an A[k]-link is requested by the producer of the packet by sending an enable signal on wire E of group 2. Then the contents of the packet are sent as p k-bit bytes by p transactions on the wires of group 1, where each transaction consists of: an enable signal on wire S; an acknowledge signal on one wire in each pair of acknowledge wires; followed by resets of S and then the active acknowledge wires. After completion of all p transactions, an acknowledge signal is returned on wire AE, and wires E and AE reset to complete a single transaction on the wires of group 2. The link is then ready for transmission of the next instruction packet.

Result packets are transmitted between units of the Distribution Network over type B[h, k] links shown in Figure 30, where h is the number of address bits in the result packet and k is the number of bits in each byte of the value part of the packet. The links to the Distribution Network from the Functional Units and the Controller are type B[q, m] having a complete set of address bits, and a fully parallel representation for the value. The links connecting the Distribution Network to Memory Registers are type B[0, 1] in which group 1 is absent and the value is in serial format.

For result packets containing h-bit addresses and values represented by p bytes of k bits each, transmission through a type B[h, k] link consists of one transaction on group 1 for the address and p transactions on group 2 for the value. When these transactions are completed, a transaction on group 3 signals that the entire packet has been transferred to the consuming unit.

Commands to set up the contents and status of Memory Registers are transmitted to the registers by command packets sent through the Command Network over type C[k] links having the structure shown in Figure 31. Transmission of a command packet consists of one transaction on group 1 for the address bits and one transaction on group 2, in which an enable signal is sent on one of wires CEC, CEV, CEMT, CIDL to indicate which of the four commands <u>enter-constant</u>, <u>enter-value</u>, <u>empty</u> or <u>idle</u> is to be effected at the specified register.

The type D links shown in Figure 32 are used to transmit execution requests from the Controller to the Memory Cells through the Control Network A command <u>run</u> n generates a sequence of n R/AR transactions followed by one RF/AR transaction on the group 1 wires; the enable signal on wire RF indicates the final execution request. A D/AD transaction signals that all Memory Cells served by the link have completed all n execution cycles, and are ready to accept further commands from the Controller.

<u>Structure and Operation of a Memory Cell</u>

Since the Memory Cell is the key element of the processor, we will use its design to illustrate the specification of behavior by type 1 interconnections of modules. A Memory Cell consists of three Register Units connected as shown in Figure 33, where each Register Unit is specified in terms of modules in Figure 34. Both diagrams are simplified by omitting the means for setting the mode and initial contents of the Register Units, including the type C links from the Command Network. (A complete design for the Memory Cell is included in the Appendix.)

In Figure 33 we see how a Memory Cell interacts with other sections of the processor. The Register Units of the Memory Cell are loaded with values received from the Distribution Network over the three type B[0, 1] links. Once enabled, the Cell groups the instruction and operands from its Register Units into an instruction packet, which it sends to the Arbitration Network over the type A[3] link. The Control Network requests one instruction execution cycle with each enable signal on the type D link. The request is distributed to the Register Units which respond individually when each has completed one cycle of operation. When all three Register Units have completed a cycle of operation, the two C-modules generate an acknowledge signal to the Control Network.

Each Register Unit generates an enable signal on wire E when it has been loaded from the Distribution Network and has received an execution request signal on wire R. The conjunction of enable signals from the three Register Units is detected by two C-modules, and produces an enable signal to the Arbitration Network. When it is able, the Arbitration Network accepts an instruction packet using the communication discipline described earlier for A-links, acknowledging completion with a signal on wire AE, which is distributed to the three Register Units.

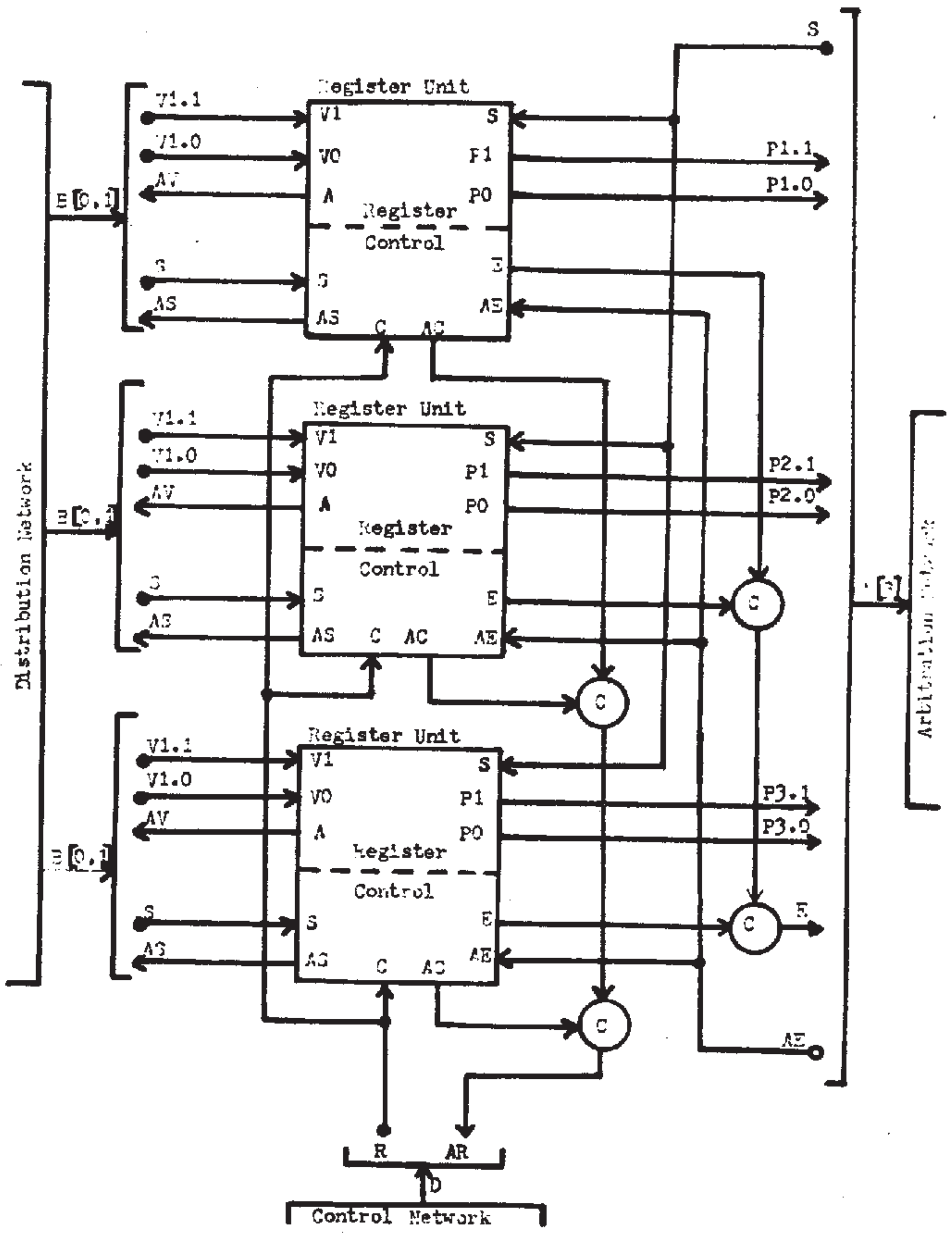
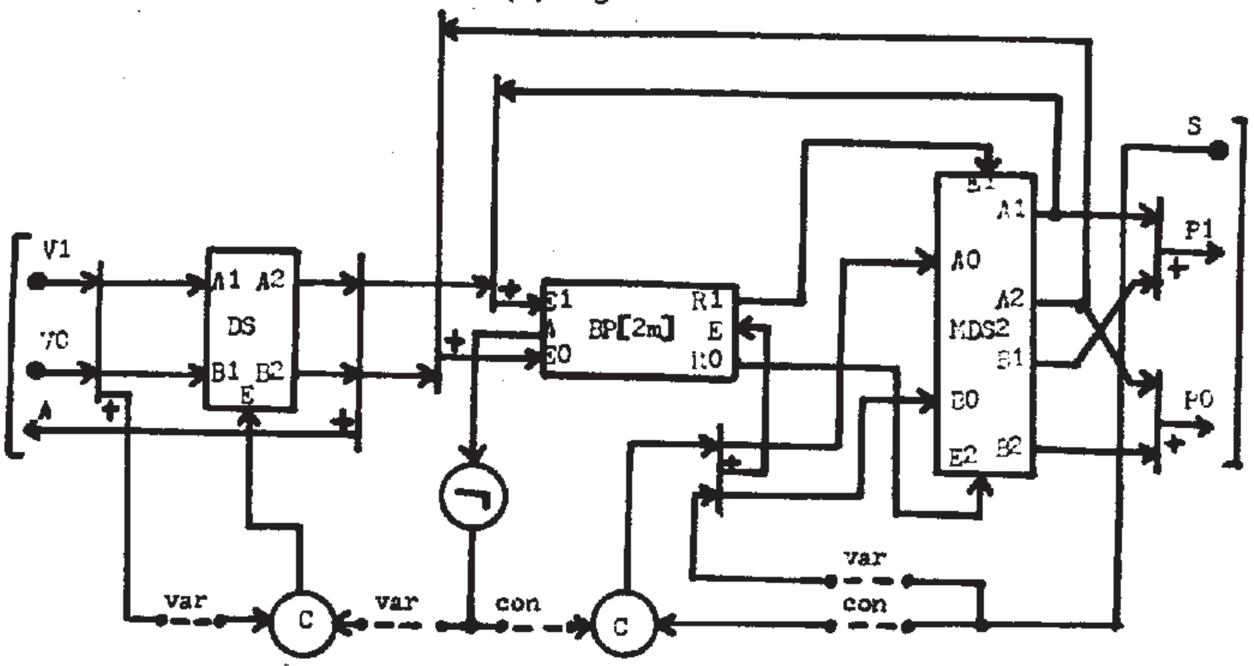


Figure 33. Specification of the Memory Cell.

(a) Register



(b) Control

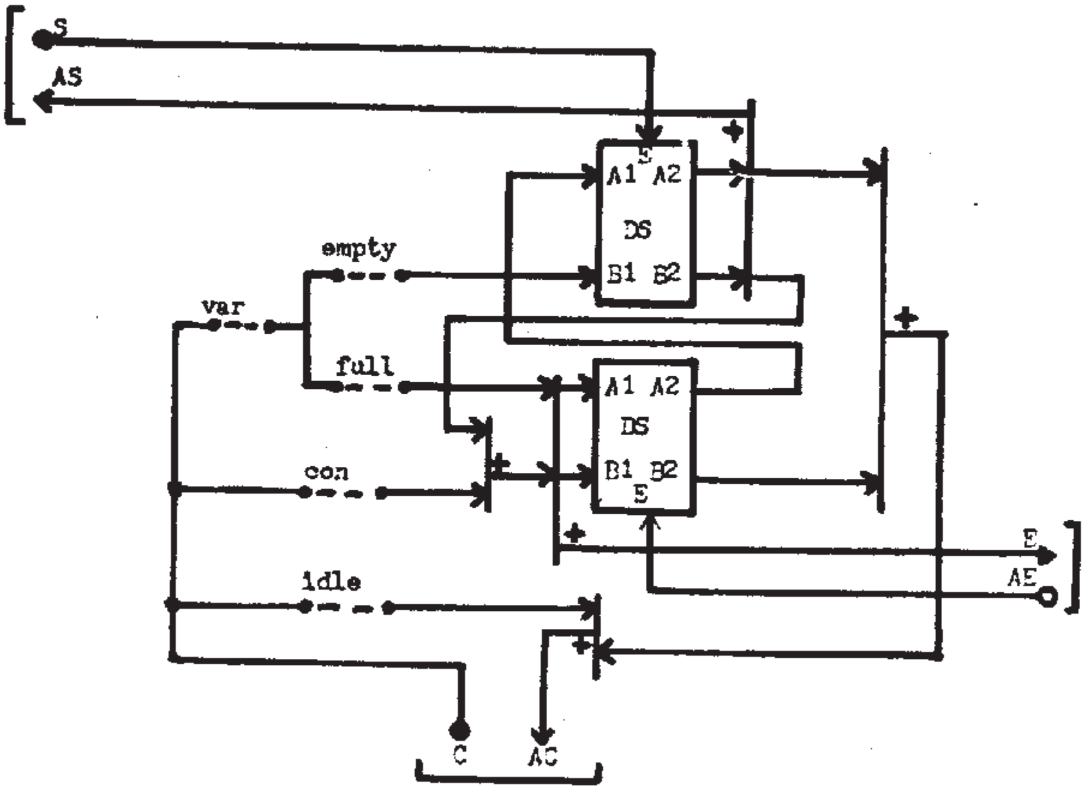


Figure 34. Specification of a Register Unit.

Before execution of a program, each Register Unit is set to one of four modes. The register may be set to <u>idle</u>, in which case it is not utilized in the computation, or it may be set to hold a constant (<u>con</u>) or a variable (<u>var</u>). In the case of a constant, the register is loaded with an initial value before program execution and will retain this value while transmitting it as part of an instruction packet. If the register contains a variable, it may be either empty or full. A full register holds an operand value in the initial configuration of the Memory, and must receive a new value through the Distribution Network after each packet transmission to complete one cycle of instruction execution. An empty register must receive a value in each cycle of instruction before packet transmission may begin.

In the diagram of the Register Unit (Figure 34), the mechanism for setting the mode of the register has been omitted for simplicity; the signal paths required for each mode are indicated by labeled gaps in the drawings. In the complete design, the mode of each register is set by transactions through the Control Network initiated by the Controller in response to commands from the Host.

The Register portion of a Register Unit is specified in terms of the basic speed independent modules in Figure 34a. Data presented at the input port enters the bit pipeline module through the data switch module if the first stage of the bit pipeline module is empty and the register is in variable mode. Each bit of data is acknowledged by a signal from one output of the data switch.

Data is requested from the pipeline by a signal from the Arbitration Network on the S wire. If the register contains a variable, the output of the bit pipeline passes through the lower section of the multiple data switch module and exits on the P1 or P0 output wires. If the register holds a constant, the upper section of the data switch passes the data to the output and also returns it to the pipeline input. The P0 and P1 output wires are reset when wire S resets to indicate that the Arbitration Network has absorbed the data, and, if the register is in constant mode, that the data has been reentered in the bit pipeline.

The Control part of a Register Unit is specified in Figure 33b. The response to the arrival of a signal on the C wire is determined by the mode of the unit. If the register is idle, an acknowledge signal is immediately returned to the Control Network. If the register contains a constant, an enable signal is immediately sent to the Arbitration Network on wire E. When an acknowledge signal returns on wire AE indicating that an instruction packet has been completely transmitted to the Arbitration Network, an acknowledge signal is sent to the Control Network.

If the register is in variable mode, the action depends on whether the register is empty or full. If it is full, the Control behaves initially as in constant mode -- it sends an enable signal on wire E. However, the acknowledge signal on wire AE causes the lower data switch to send a signal to the upper data switch, which waits until a space signal arrives from the Distribution Network indicating the the register has been reloaded. Then acknowledge signals are sent to the Distribution Network on wire AS and to the Control Network on wire AC.

If the register holds a variable, but is initially empty, an enable signal waits at the upper data switch for the register to be filled from the Distribution Network. Then action completes as if the register were in constant mode.

In specifying units of the processor as speed independent interconnections of basic modules, note that we have simply given precise statements of the intended behavior of the units -- the diagrams should not be regarded as wiring diagrams for their manufacture. The available logic elements do not permit economical fabrication of processor units directly from basic modules. Nevertheless, each unit has a reasonably efficient implementation, using conventional logic elements, that has behavior consistent with the specified behavior. Furthermore, since the processor design makes use of large numbers of identical units, fabrication by connecting several kinds of LSI chips is feasible, and a set of modules similar to those presented here could become basic cells for mask layout in an LSI technology.

Conclusion

The idea of organizing a computer so execution of instructions is triggered by the presence of their operands has been discussed by Seeber and Lindquist [22], Patil [18], Dennis [5], Shapiro, Saint and Presberg [23], and Miller and Cocke [15]. However, none of these authors has suggested a detailed and efficient scheme for communicating enabled instructions and operands to functional units for processing. We are hopeful that the architecture proposed here offers an attractive solution to this problem -- a solution that can be extended to the design of processors that support programming languages suitable for general purpose computation.

References

1. Adams, D. A. A Computation Model With Data Flow Sequencing. Technical Report CS 117, Computer Science Department, School of Humanities and Sciences, Stanford University, Stanford, Calif., December 1968.

2. Anderson, D. N., F. J. Sparacio, and R. M. Tomasulo. The IBM System/360 Model 91: machine philosophy and instruction handling. IBM Journal of Research and Development, Vol. 11, No. 1, January 1967, 8-24.

3. Bährs, A. Operation patterns (An extensible model of an extensible language). Symposium on Theoretical Programming, Novosibirsk, USSR, August 1972 (preprint).

4. Barnes, G. H., R. M. Brown, M. Kata, D. J. Kuck, D. L. Slotnick, and R. A. Stokes. The Illiac IV computer. IEEE Transactions on Computers, Vol. C-17, No. 8, August, 1968 746-757.

5. Dennis, J. B. Programming generality, parallelism and computer architecture. Information Processing 68, North-Holland Publishing Company, Amsterdam 1969, 484-492.

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. Dennis, J. B., and S. S. Patil. Speed independent asynchronous circuits. Proceedings of the Fourth Hawaii International Conference on System Sciences, Western Periodicals Co., North Hollywood, Calif., 1971, 55-58.

8. Dennis, J. B. First version of a data flow procedure language. Symposium on Programming, Institut de Programmation, University of Paris, Paris, France, April 1974, 241-271.

9. Dennis, J. B., and J. B. Fosseen. Introduction to Data Flow Schemas. (Submitted for publication), November 1973.

10. Hintz, R. G., and D. P. Tate. Control Data Star-100 processor design. Sixth Annual IEEE Computer Society International Conference, Digest of Papers 1972: Innovative Architecture, IEEE, New York., 1972, 1-4.

11. Karp, R. M., and R. E. Miller. Properties of a model for parallel computations: determinacy, termination, queueing. SIAM J. Appl. Math. 14 (November 1966), 1390-1411.

12. Kosinski, P. R. A Data Flow Programming Language. Report RC 4264, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., March 1973.

13. Kosinski, P. R. A Data Flow Language for Operating Systems Programming. Proceedings of ACM SIGPLAN-SIGOPS Interface Meeting, SIGPLAN Notices 8, 9 (September 1973), 89-94.

14. Liskov, B. H., and S. N. Zilles. Programming with abstract data types. Proceedings of a Symposium on Very High Level Languages, SIGPLAN Notices 9, 4 (April 1974), 50-59.

15. Miller, R. E., and J. Cocke, Configurable Computers: A New Class of General Purpose Machines. Report RC 3897, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., June 1972.

16. Misunas, D. P. Petri nets and speed independent design. Comm. of the ACM 16, 8 (August 1973), 474-481.

17. Muller, D. Asynchronous logics and application to information processing. In Switching Theory and Space Technology. Howard Aiken and William Mann (Eds.) Stanford University Press, Stanford, Calif., 1963.

18. Patil, S. S. An Abstract Parallel Processing System. S. M. Thesis, Department of Electrical Engineering, M.I.T., Cambridge, Mass., June 1967.

19. Patil, S. S., and J. B. Dennis. The description and realization of digital systems. Proceedings of the Sixth Annual IEEE Computer Society International Conference, IEEE, New York, N. Y. 1972, 223-226.

20. Patil, S. S. Synchronizers and Arbiters. Computation Structures Group Memo 91, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass., October 1973.

21. Rodriguez, J. E. A Graph Model for Parallel Computation. Report TR-64, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass., September 1969.

22. Seeber, R. R., and A. B. Lindquist. Associative logic for highly parallel systems. AFIPS Conference Proceedings 24, 1963, 489-493.

23. Shapiro, R. M., H. Saint, and D. L. Presberg. Representation of Algorithms as Cyclic Partial Orderings. Report CA-7112-2711, Vol. 1, Applied Data Research, Wakefield, Mass., December 1971.

24. Thornton, T. E. Parallel operation in the Control Data 6600. AFIPS Conference Proceedings: 1964 Fall Joint Computer Conference, Academic Press, London 1971, 575-588.

Appendix: Specifications of Processor Units

Each section of the Processor; Memory, Arbitration Network, Functional Units, Distribution Network, Controller, Command Network and Control Network, is built as an interconnection of simpler units. A specification for each unit type is given here as an interconnection of the basic modules presented in the paper.

In each specification, component modules of each figure are numbered and referred to by figure number and component number. For example, (A4#6) designates module 6 in Figure A4. Also, each port of each unit is labeled with an identifier specifying its type (input or output) and a number to differentiate among input or output ports. A specific wire of a link is referenced in the text by designating the port the link enters and the wire identifier within the link. For example, in1:P1.1 refers to wire P1.1 of input port in1.

1.0 Memory

The Memory consists of 3n Register Units organized into Cells as shown in Figure A1. Each Cell contains three Register Units having consecutive addresses and a control structure. As described previously, type B[0,1] links transmit values of result packets from the Distribution Network to the input ports of the Register Units, and links from output ports of the three Register Units and the control structure combine into a type A[3] link, delivering the contents of the Cell as an instruction packet to the Arbitration Network.

The Control Network enables the execution of instructions by loading the number of operations desired into the bit pipeline of the control structure over a type D link. The pipeline stores requests for data received on the R input and acknowledges them as AR. Each input allows one transferral of an instruction packet to the Arbitration Network by means of the SIG module. The final request from the control network (RF) is acknowledged on AR and, upon reaching the output of the pipeline, causes an acknowledge to be sent to the Control Network on wire D, signaling completion.

The Command Network controls the emptying of Register Units, the entering of values in Register Units, and the setting of Register Units to function as constants, variables or in idle mode, by a Type C link connected to the inputs of the Register Units.

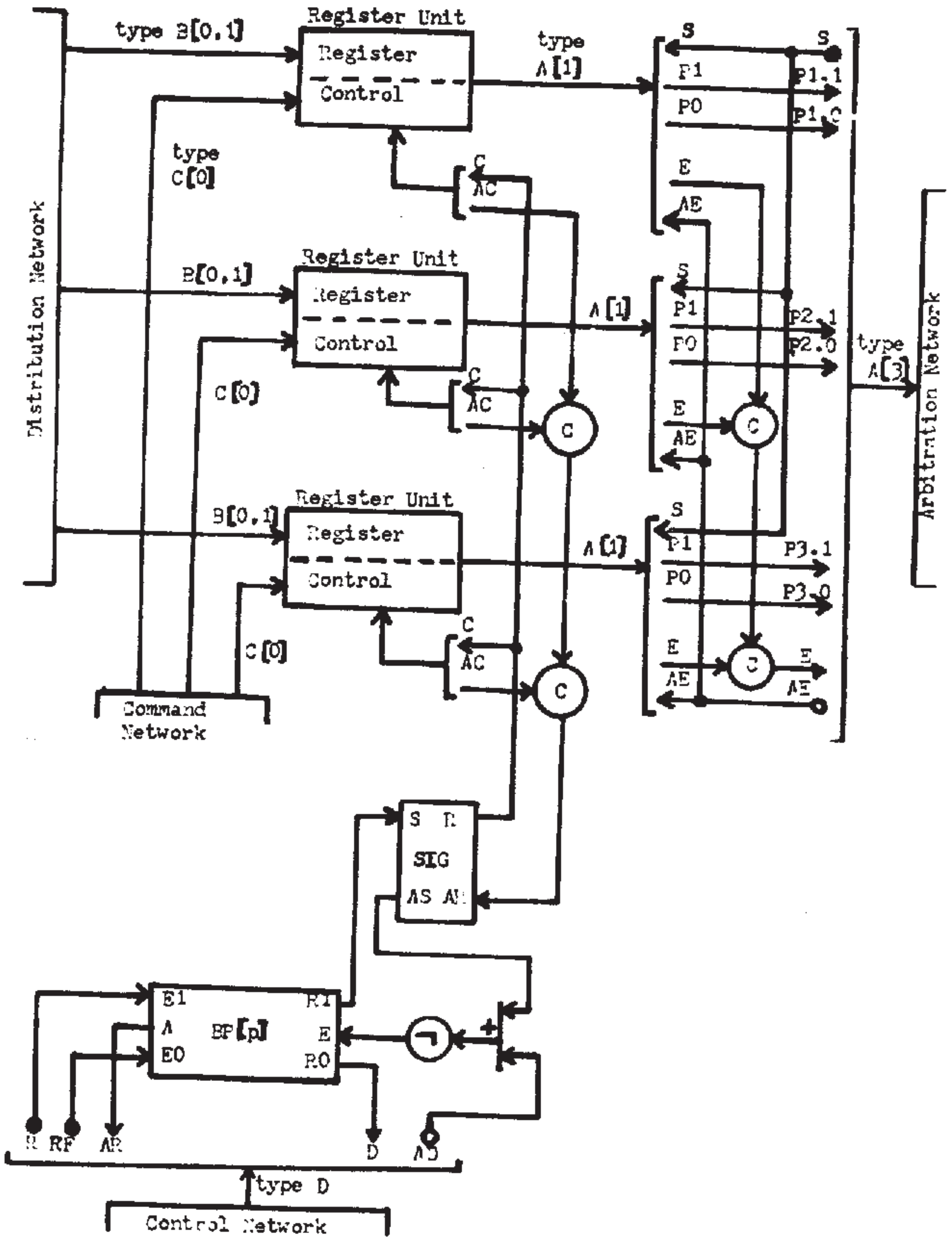


Figure A1. Specification of the Memory Cell.

1.1 Register

The construction of a Register from basic modules is shown in Figure A2. An m-bit instruction or operand is held in the bit pipeline module (A2#6). Whether the Register is to treat the word in the pipeline (A2#6) as a constant or variable is represented by the state of gate module (A2#13) and SEL module (A2#14). When holding a constant, the gate module (A2#13) is reset and SEL module (A2#14) is set for output D2. The opposite conditions hold when the Register holds a variable. The mode of the unit is set by the receipt of a command from the Control (in2:C, in2:V) specifying the type to be stored. This command appropriately sets or resets (A2#13), sets (A2#14) and is acknowledged (in2:AC, in2:AV).

The multiple data switch module (A2#8) directs the flow of bits leaving the pipeline module (A2#6). Outputs A1, A2 of (A2#8) are active during program execution if the Register is in constant mode; in this case bits from (A2#6) are directed to output port out1 and are also recirculated into (A2#6) via modules (A2#4, A2#5). Outputs B1, B2 of (A2#8) are active for program execution in variable mode. Outputs C1, C2 of (A2#8) are used to empty the pipeline module upon receipt of a command do so from the Control.

Data switch module (A2#2) gates bits into the pipeline module from port in1 either in response to an enter command or during program execution in variable mode. Data switch module (A2#12) directs input acknowledge events from the pipeline as appropriate for filling from port in1 (output A1), or recirculating a constant (output B1). Modules (A2#13, A2#14) set up signal paths for constant mode or variable mode operation.

1.2 Control

The Control provides the interface with the Command Network and the control structure of the Cell and maintains the status of the Register. The condition of a Register Unit when the processor is quiescent (not executing a command) is either empty or full, according as the pipeline module (A2#6) is empty or holds an m-bit value. This difference affects operation of the Register Unit and is recorded in SEL modules (A3#3, A3#4), the modules being set to their D1 output for the empty quiescent condition of the Register Unit. SEL module (A3#1), in a manner similar to the SEL modules in the Register (A2#13, A2#14), contains the mode of the Register (constant or variable). The state of the Register (idle or active) is stored in SEL module (A3#16). This module

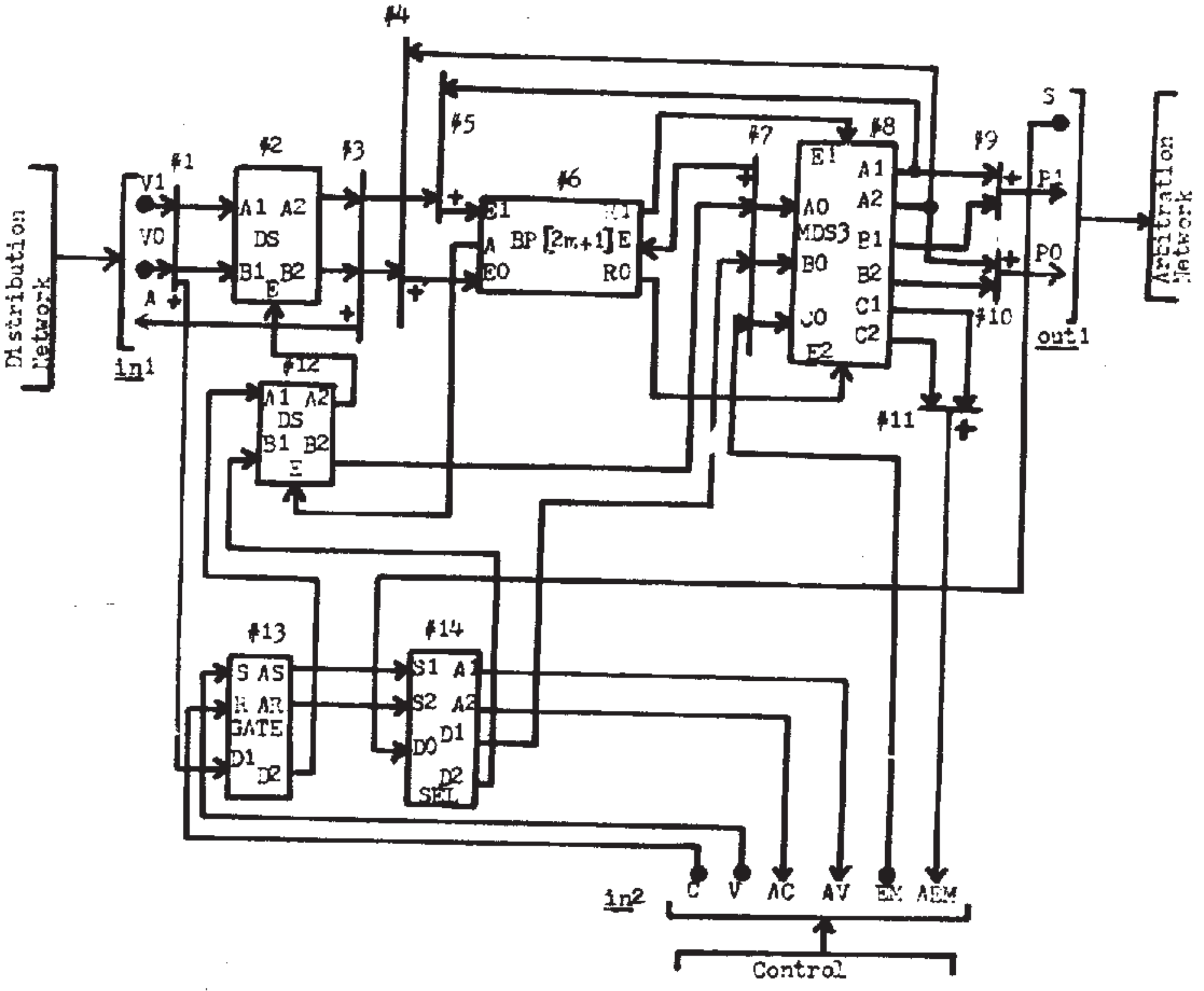


Figure A2. Specification of the Register.

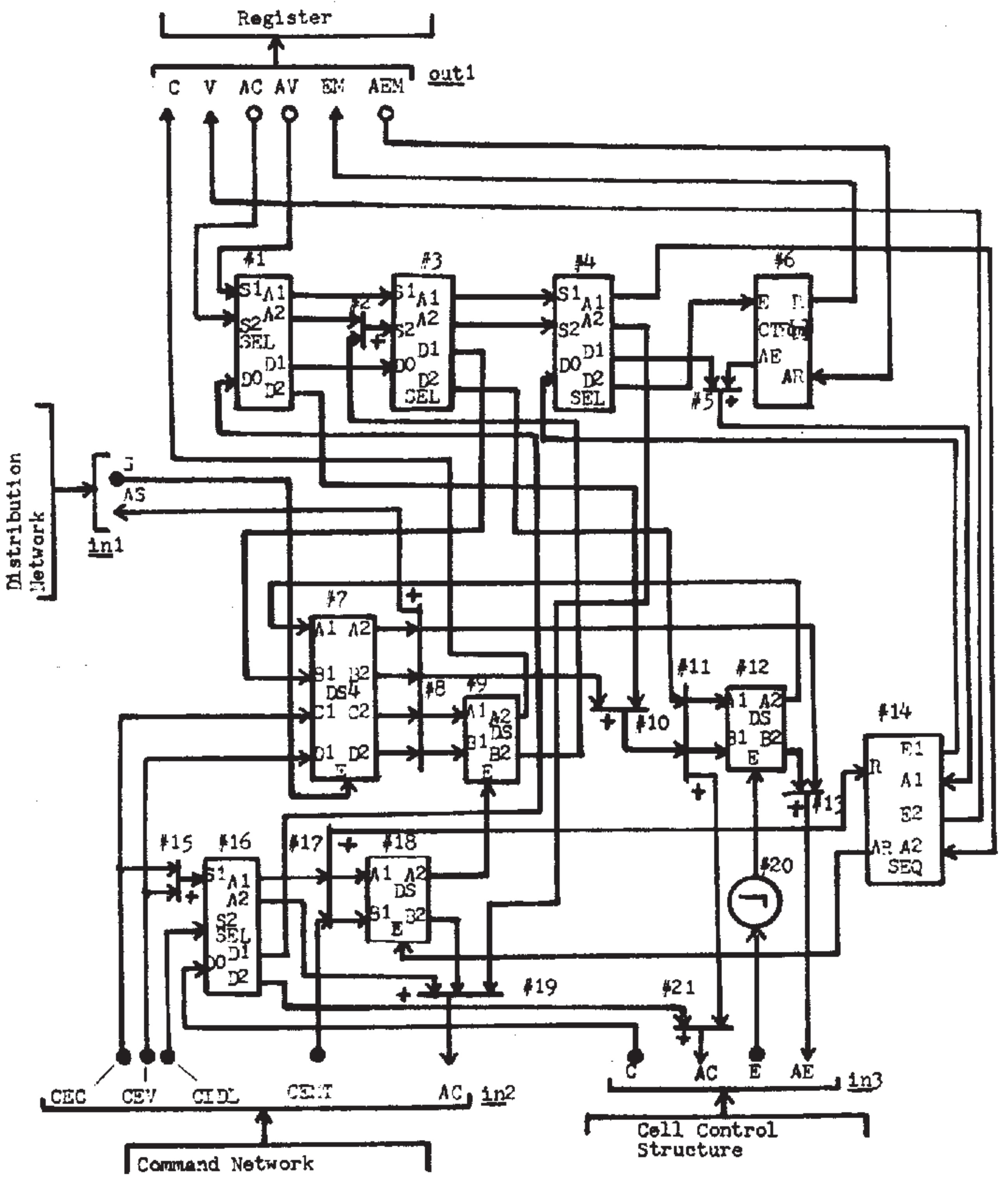


Figure A3. Structure of the Control.

is set to its D1 output by the receipt of a command to load a constant or a variable (in2:CEC, in2:CEV) and to its D2 output upon receipt of an idle command (in2:CIDL). The SEL modules change condition only in response to enter, empty, or idle commands.

One cycle of execution activity occurs in response to an enable signal from the control structure on wire in3:C. If the Register Unit has been set to idle, an acknowledge is immediately returned (A3#16, A3#21) on in3:AC. If the unit is in constant mode (A3#1), an acknowledge signal is generated via (A3#10, A3#11) on in3:AC, which informs the Cell control structure that the Register contains a value ready for transmission to the Arbitration Network. If the Register Unit is in variable mode (A3#1) and is empty (A3#3), generation of the acknowledge signal on in3:AC is delayed (A3#7, A3#10, A3#11) until an enable signal on in1:S indicates that the pipeline module (A2#6) has been filled by a value from the Distribution Network. An enable signal on in3:E signifies that the Arbitration Network has received an entire instruction packet. In the preceeding two cases, an acknowledge signal on in3:AE is generated immediately via (A3#12, A3#13). The initial condition of the Register Unit is reestablished by resets in sequence on in3:C in3:AC, in3:E and in3:AE. If the Register Unit is in variable mode and is full, an enable signal on in3:C immediately generates an acknowledge on in3:AC via modules (A3#1, A3#3, A3#11). In this case, module (A3#12) causes the acknowledge signal on in3:AE to be delayed via (A3#7) until an enable signal occurs on in3:E, indicating that the instruction packet has been entirely accepted by the Arbitration Network, and a space signal appears on in1:S indicating that the Register has been refilled by a result packet.

The commands enter-constant, enter-variable, empty and idle are signaled to the Control through port in2. For each of the first three commands, the initial step is to empty the pipeline if it is full (A3#14, A3#4, A3#6, in3:EM/AEM), to set the gate and SEL modules in both the Register and the Control (A2#13, A2#14, A3#1) to indicate variable mode, and to set SEL modules (A3#3, A3#4) to indicate empty. The setting of the modules in the Register is accomplished with either an out1:SC/AC or an out1:SV/AV transaction, controlled by the sequence module (A3#14). Data switch module (A3#18) causes an immediate acknowledge signal on in2:AC if the command is empty. In the case of the two enter commands, modules (A3#7, A3#9) cause a wait until the pipeline has been filled. Then modules (A2#13, A2#14, A3#1) are set for constant or variable mode, and, for both enter commands, modules (A3#3, A3#4) are set to indicate a filled pipeline. The receipt of an idle command (in2:CIDL) sets SEL module (A3#16) to indicate the idle state and is acknowledged (in2:AC).

2.0 Arbitration Network

The Arbitration Network provides transmission paths for instruction packets from each Cell of the Memory to each Functional Unit. One possible structure for the Arbitration Network was shown in Figure 9. An Arbitration Unit accepts instruction packets through several input ports on a first-come, first-served basis and delivers them at a common output port. Sufficient Arbitration Units are required to provide a path from each Cell to each Functional Unit. A Serial/Parallel Conversion Unit transforms each arriving instruction packet into a more parallel format. This conversion from serial to parallel format may be done in several stages. Buffer Units store complete instruction packets in readiness for quick transmission through a following Arbitration Unit, in order to prevent an instruction packet from engaging an Arbitration Unit before serial/parallel conversion is complete. Function Switch Units direct instruction packets to one of several output ports according to function bits of the instruction.

The Arbitration Units provide the fan-in whereby each Functional Unit may receive an instruction packet from any Cell. The Function Switch Units provide the fan-out whereby each Cell may send an instruction packet to any Functional Unit. For simplicity in the present description a fan-in and fan-out of two is assumed. Generalization of the designs is straightforward.

All connections to units of the Arbitration Network of Figure 9 are type A links. We suppose that each Serial/Parallel Converter multiplies the number of parallel data paths by b. Thus, input links to the Arbitration Network are type A[3]; the input links to the second rank of Arbitration Units are type A[3b]; and the input links to the third rank of Arbitration Units are type A[$3b^2$].

2.1 Arbitration Unit

A typical Arbitration Unit is shown in Figure A4. This design is for the first rank of Arbitration Units in Figure 9. The versions required for the second and third ranks are obtained merely by increasing the number of data switch modules (A4#3, A4#7, A4#11) to accommodate the larger number of data wires and by changing the modulus of counter (A4#14).

A signal on in1:E or in2:E indicates that an instruction packet is available at one of the input ports. The arbitration module (A4#15) gives priority to the first enable signal to reach it, and sets SEL module (A4#16) accordingly.

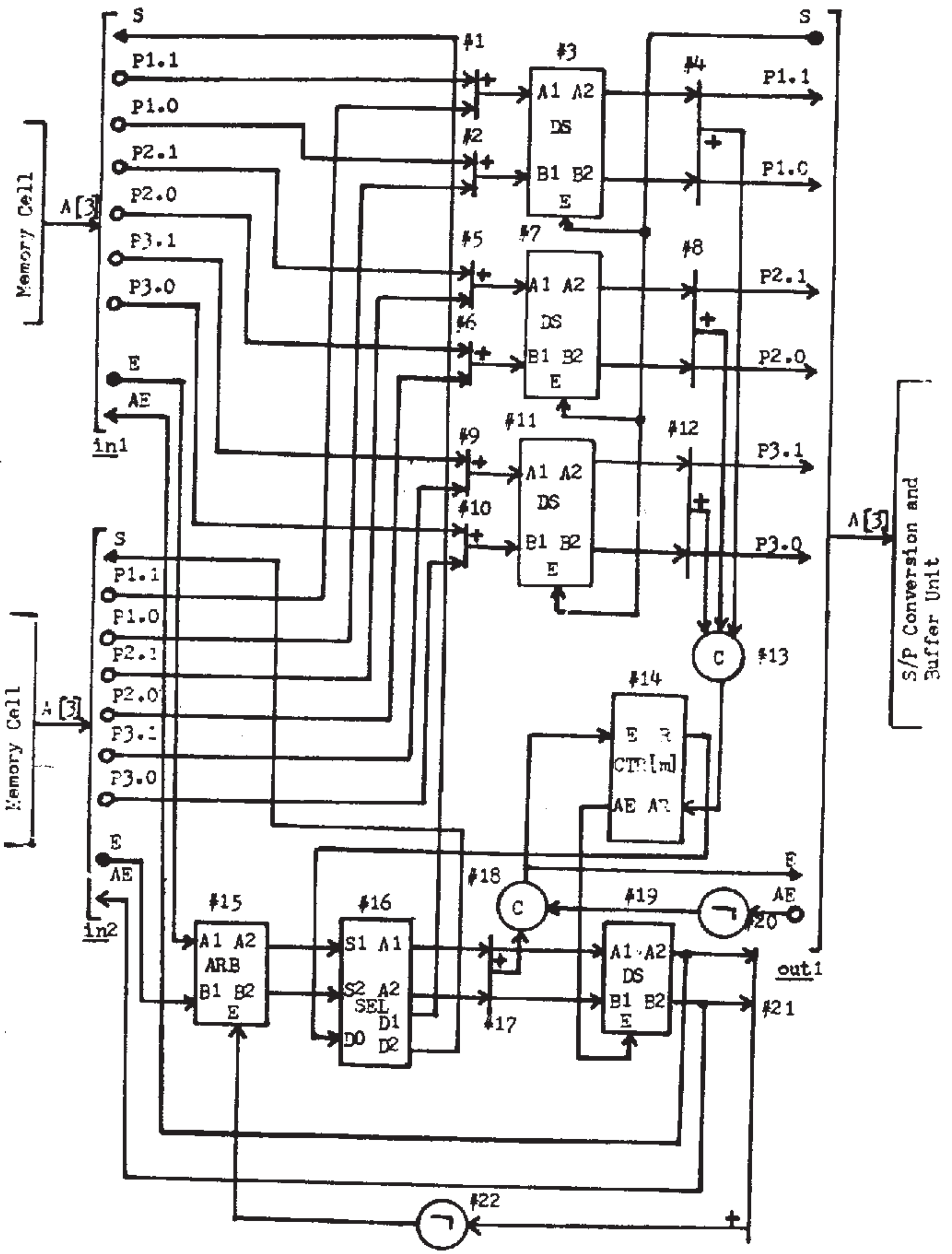


Figure A4. Specification of an Arbitration Unit.

Once (A4#16) is set, counter (A4#14) is enabled to perform m cycles of data transmission, and an enable signal is sent out on <u>out</u>1:E.

Each data transmission cycle is started by an enable signal from counter (A4#14) sent over wire S of the input port selected by (A4#16). When an acknowledge signal is received over one wire of each data pair, C-module (A4#13) sends an acknowledgement to the counter (A4#14). The signal path just traced must reset before a new transmission cycle may begin. When the count is complete, the acknowledge signal from (A4#14) is directed to wire AE of the appropriate input port by data switch (A4#19), and the arbitration module (A4#15) is reenabled through (A4#21).

2.2 <u>Serial/Parallel Conversion and Buffer Unit</u>

The functions of serial-to-parallel conversion and buffer storage are conveniently combined in one unit as shown in Figure A5. This unit has a type A[1] input link and a type A[3] output link. The generalization to the forms required in Figure 9 is easy.

Data arriving on wires <u>in</u>1:P1.1 and <u>in</u>1:P.0 are passed to the pipeline modules (A5#1, A5#2, A5#3) through a multiple data switch (A5#4). The sequence module (A5#6) controls distribution of successive bits cyclically to the three pipeline modules. The action of filling the pipeline modules is started by a signal on <u>in</u>1:E which enables counter (A5#7). When each pipeline is filled with w bits, the counter acknowledges, yielding through (A5#8) an enable signal on <u>out</u>1:E and an acknowledge signal on <u>in</u>1:AE. The pipeline buffer may then be emptied by w enable signals on <u>out</u>1:S.

2.3 <u>Function Switch Unit</u>

A typical Function Switch Unit is shown in Figure A6. For illustration, the instruction packet arriving at port <u>in</u>1 is assumed to have w bytes of 2 bits each. The Function Switch Unit uses bit P1 of the first byte to direct the packet to either output port <u>out</u>1 or <u>out</u>2.

Modules (A6#12, A6#13, A6#14, A6#15) make up a w-state counter whose first state is distinguished by an enable signal on output R of (A6#12) rather than on output R of (A6#15). Hence, when enabled by a signal on <u>in</u>1:E, the SIG module (A6#12) causes bit P1 of the first byte received to be stored in data switch (A6#7). An enable signal is then sent on <u>out</u>1:E or <u>out</u>2:E, and the entire packet is sent out port <u>out</u>1 or <u>out</u>2 through multiple data switch (A6#4) by a sequence of w enable signals on <u>out</u>1:S or <u>out</u>2:S. An acknowledge signal occurs on

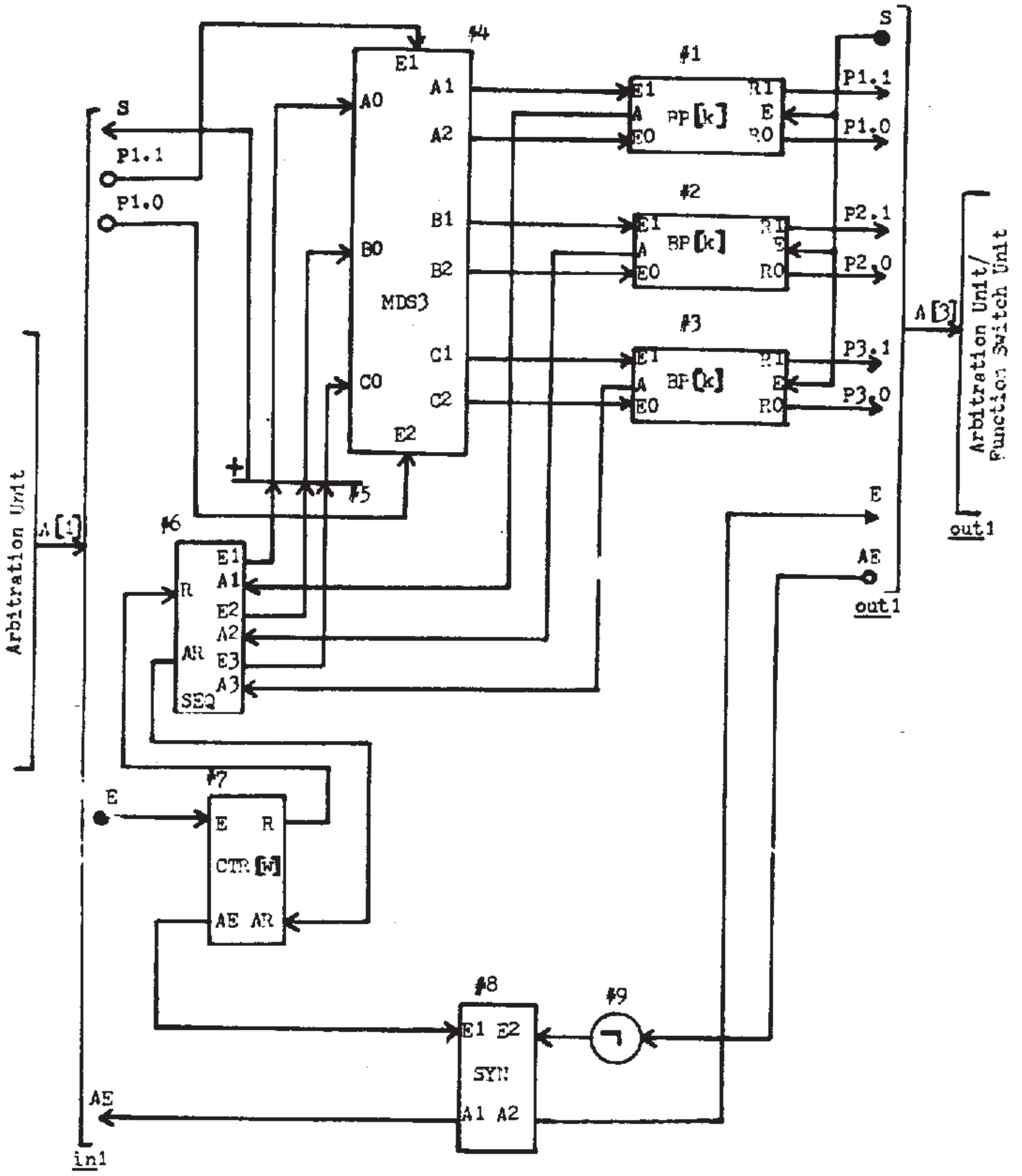


Figure A5. Specification of a Serial-to-Parallel Conversion and Buffer Unit.

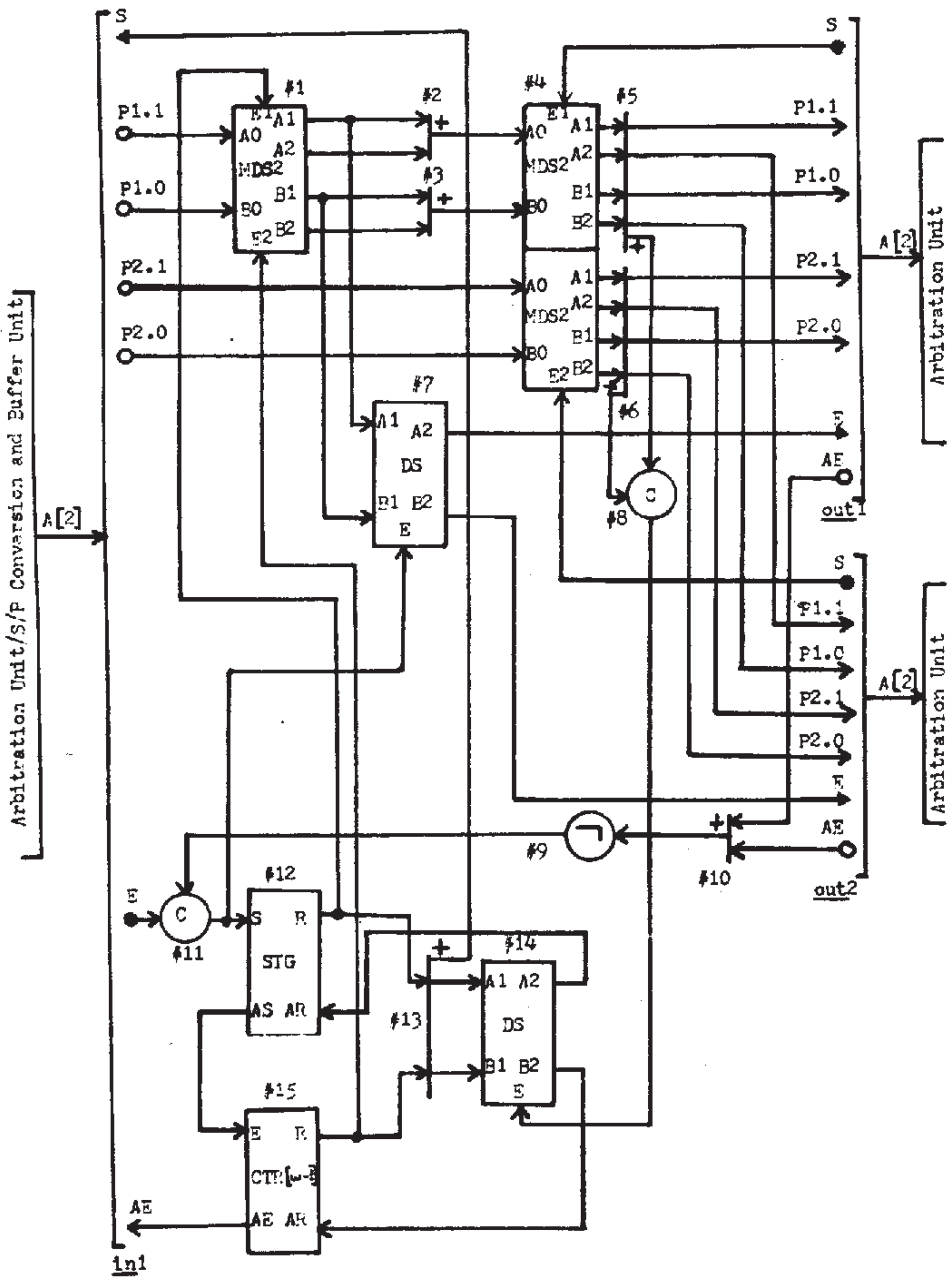


Figure A6. Specification of the Function Switch Unit.

in1:AE when packet transmission is complete and counter (A6#15) acknowledges.

3.0 Functional Units

Each Functional Unit has the structure shown in Figure A7. Instruction packets enter through port in1 in parallel format, and consist of two addresses, a specification and two values. The Operation Unit (A7#3), which might be a pipeline multiplier, for example, receives the two values which are used as operands, and the specification, which indicates a specialization of the operation to be performed. The two addresses are entered into two Address Pipeline Units (A7#1, A7#2). Results of operations and their associated addresses are formed into two result packets and sent separately to the Distribution Network through ports out1 and out2.

Modules (A7#4, A7#5) implement the signaling discipline required of type A (port in1) and type B (ports out1 and out2) links.

4.0 Distribution Network

The Distribution Network provides transmission paths for result packets from each Functional Unit and from the Controller to each Register Unit of the Memory. One possible structure for the Distribution Network was shown in Figure 10 of the paper. A few Arbitration Units are required in the network because result packets from several Functional Units may compete for access to the Distribution Network. Each Switch Unit directs result packets over one of two paths in the Distribution Network according to the most significant remaining bit in the address part of the packets. The address bit tested by the Switch is deleted from the packet. The Parallel/Serial Conversion Units handle only the value part of the result packet since there is insufficient advantage to serializing the address part. The purpose of the Buffer Unit is to avoid causing the preceding Value Switch Unit to wait for the parallel/serial conversion to complete before passing other result packets.

4.1 Switch Unit

The structure of a typical Switch Unit is shown in Figure A8. The most significant address bit of the result packet is received on wires in1:A1.1, in1:A1.0 and is held in data switch (A8#1). The output signal from (A8#1) enables multiple data switch module (A8#9) for transmitting a value and multiple data switch module (A8#2) for transmitting the remainder of the address from port in1 to port out1 or out2.

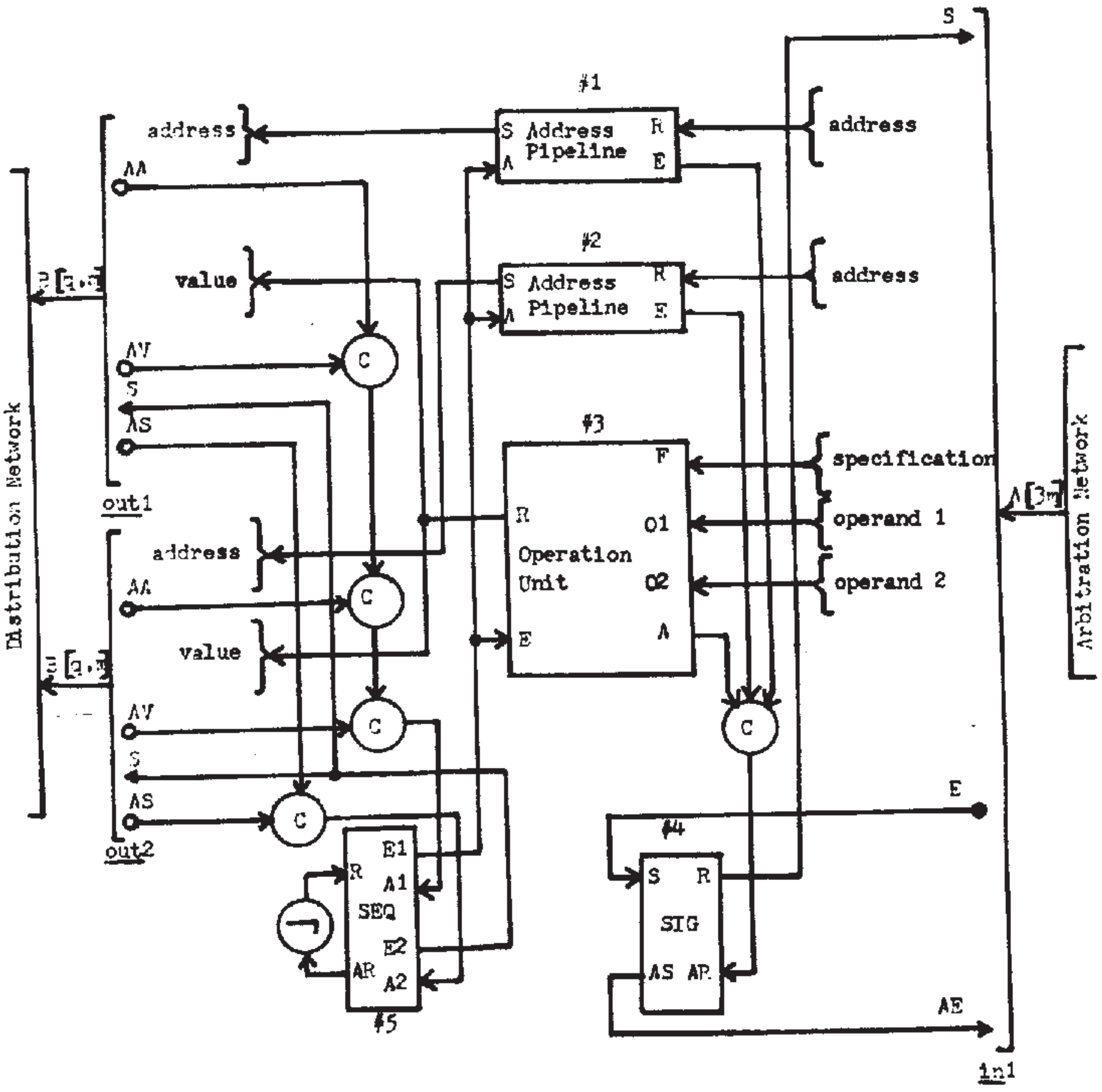


Figure A7. Specification of a Functional Unit.

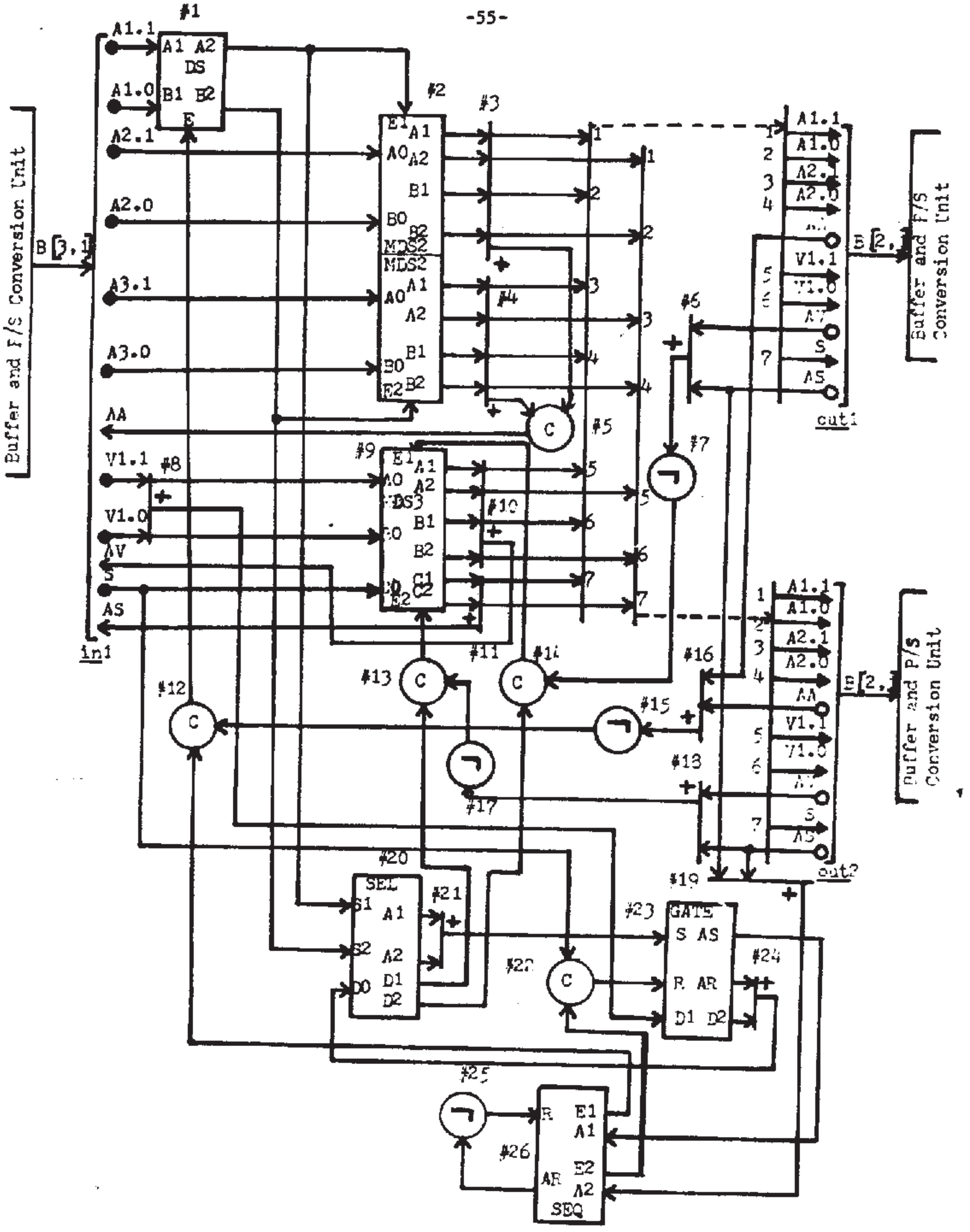


Figure A8. Specification of the Switch Unit.

This enable signal sets the select module (A8#20) to indicate the output port to be used, and then sets the gate module (A8#23) which permits passage of value bits through the multiple data switch modules. Sequence module (A8#26), which starts operation with a signal at output E1, permits a signal on E2 only after the data switch (A8#1) has reset. Then a space signal on in1:S resets gate module (A8#23), blocking further data transmission, and the space signal is transmitted by an out1:S/AS or an out2:S/AS transaction. When this transaction is complete, sequence module (A8#26) returns to its initial condition.

4.2 Buffer and Parallel/Serial Conversion Unit

The Buffer and Parallel/Serial Conversion Unit (Figure A9) uses the same principle of operation as the Parallel/Serial Conversion and Buffer Unit (Figure A5). The buffer storage is provided for the address by pipeline unit (A9#1) and for the value by three pipeline modules (A9#2, A9#3, A9#4) which are filled by w transactions on port in1. Sequence module (A9#10) and data switch (A9#11) control the transmission of bits through port out1 from the three pipeline modules cyclically. Counter (A9#12) generates the space signal on out1:SP when all bits of a result packet have been transmitted and the counter has reset. The event pipeline (A9#13) ensures that an S/AS transaction is completed on port in1 before a new packet is transmitted.

5.0 Controller

The Controller accepts commands from the Host, which may be a supervisory computer or an operator console. The five types of commands are the following:

enter-constant(a, v)
enter-variable(a, v)
empty(a, -)
idle(a, -)
run(-, v)

The two types of enter commands cause the value v to be entered in the Register Unit with address a. An empty command causes the Register Unit with address a to be emptied. An idle command specifies that the Register Unit with address a is not to be used in the computation and should be set to idle. A run command is a request that the program represented in the Memory be run for v execution cycles, so that each instruction is executed v times.

The structure of the Controller is shown in Figure A10. The Command Interpreter Unit (A10#1) accepts commands from the Host (A10#3), transmits result

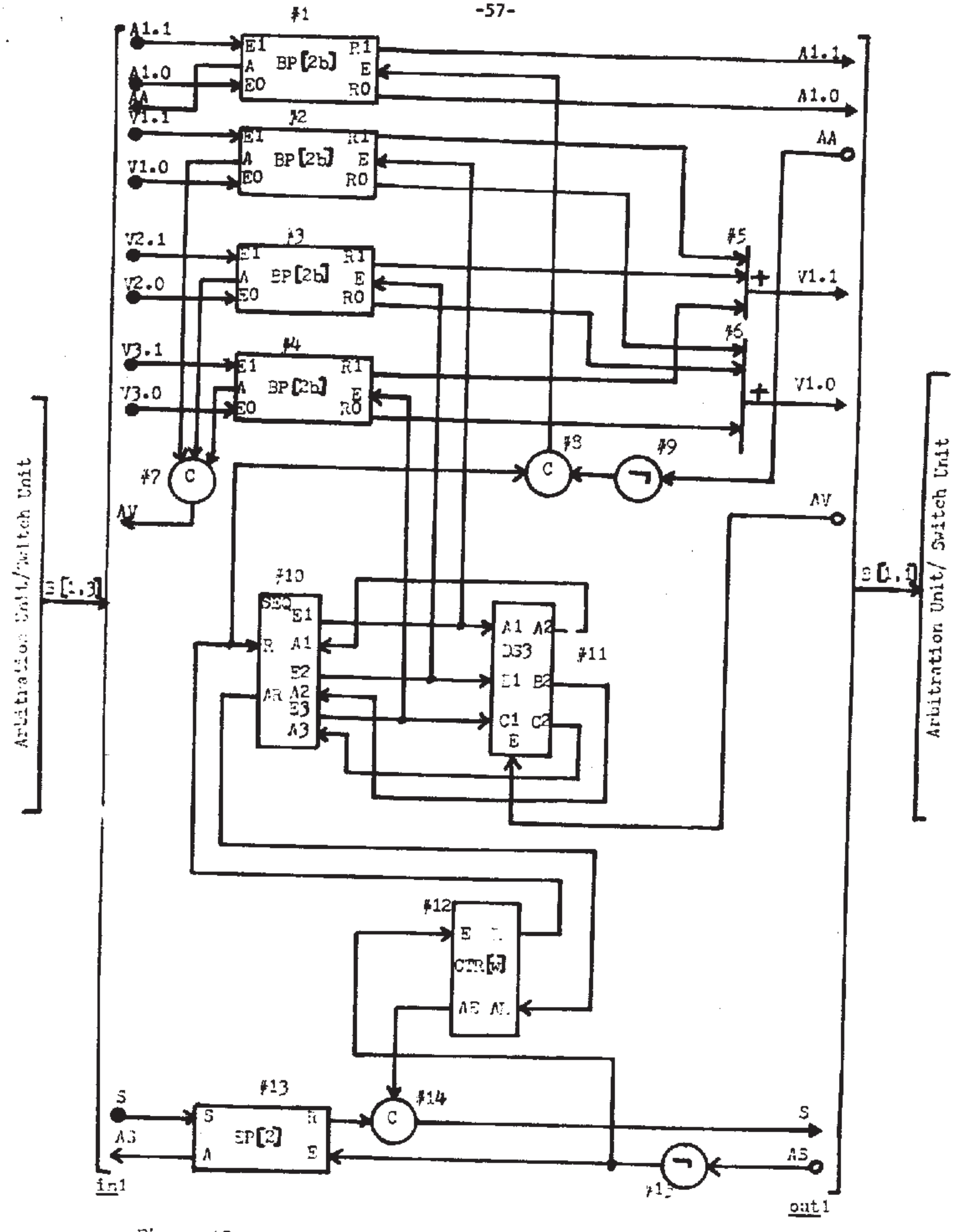


Figure A9. Specification of a Buffer and Parallel-to-Serial Conversion Unit.

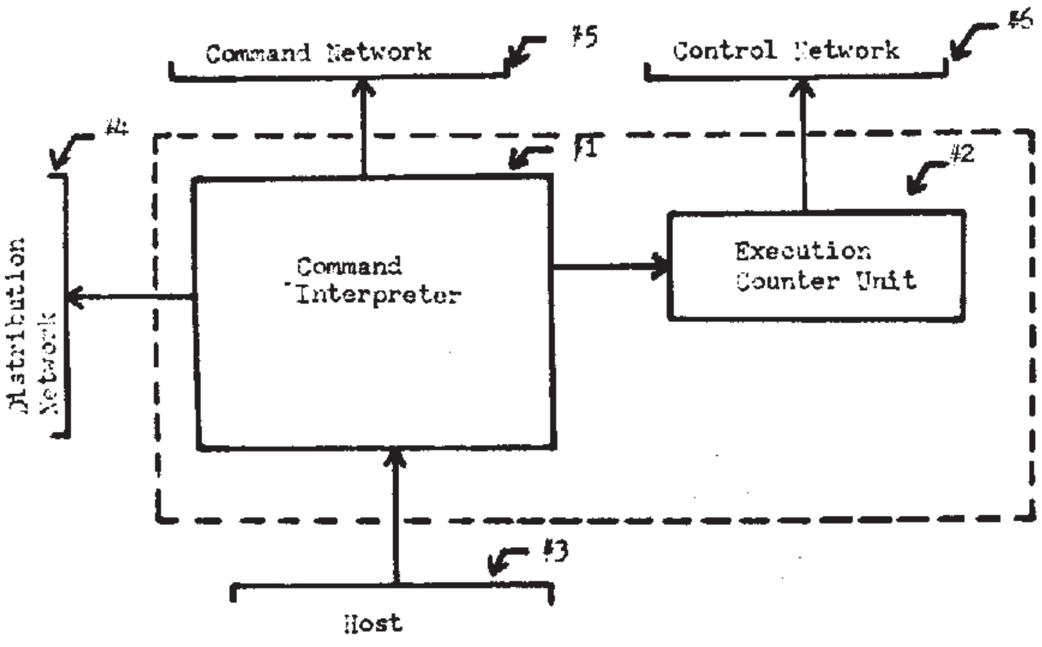


Figure A10. Links and Specification of the Controller.

packets through the Distribution Network (A10#4) for enter commands, requests Register Units to fill or empty themselves or set themselves to idle through the Command Network (A10#5), and delivers run commands to the Execution Counter Unit (A10#2). The Execution Counter Unit transmits the specified number of execution cycle request signals to the Cells of the Memory through the Control Network (A10#6).

5.1 Command Interpreter Unit

The structure of the Command Interpreter Unit is shown in Figure A11. However, addresses and values are illustrated as 2-bit values for simplicity. The generalization to q-bit addresses and m-bit values is straightforward. The commands enter-constant, enter-variable, empty, idle and run are performed in response to enable signals on wires in1:CEC, in1:CEV, in1:CEMT, in1:CIDL, in1:CRN, respectively. An acknowledge signal occurs on in1:AC when execution of a command is completed. Addresses are presented to the Controller on wires in1:A1.1 through in1:A2.0 and acknowledged on in1:AA for enter, empty and idle commands. Values are presented on wires in1:V1.1 through in1:V2.0 and acknowledged on in1:AV, for enter and run commands.

For an enter-constant or enter-variable command, a result packet is presented to the Distribution Network (port out1) through a type B[2, 2] link. The result packet is formed by data switch modules (A11#1, A11#4) controlled by modules (A11*8, A11#6, A11#7). The sequence module (A11#6) performs a transaction in1:S/in1:AS after the result packet is acknowledged, and the packet wires are reset, as required for type B links. At the same time the address is sent to the Command Network (port out2) through a type C[2] link, and one of the transactions out2:CEC/AC or out2:CEV/AC is performed [Modules (A11#12, A11#10)]. This action directs the specified Register Unit to await delivery of a value from the Distribution Network and return an acknowledge signal when the value is completely received.

An empty or idle command is executed by sending the address to the Command Network (port out2) and performing an out2:CEMT/AC or out2:CIDL/AC transaction. [Modules (A11#12, A11#10)]. This directs the specified Register Unit to empty its pipeline module or set its state to idle.

A run command is executed by sending the value to the Execution Counter (port out3) and performing an out3:D/AD transaction [Module (A11#15)].

Modules (A11#3, A11#2) handle acknowledgement of addresses and permit the address wires to be reset as soon as the address has been absorbed by the Distribution Network or Command Network. Module (A11#5) provides the same function

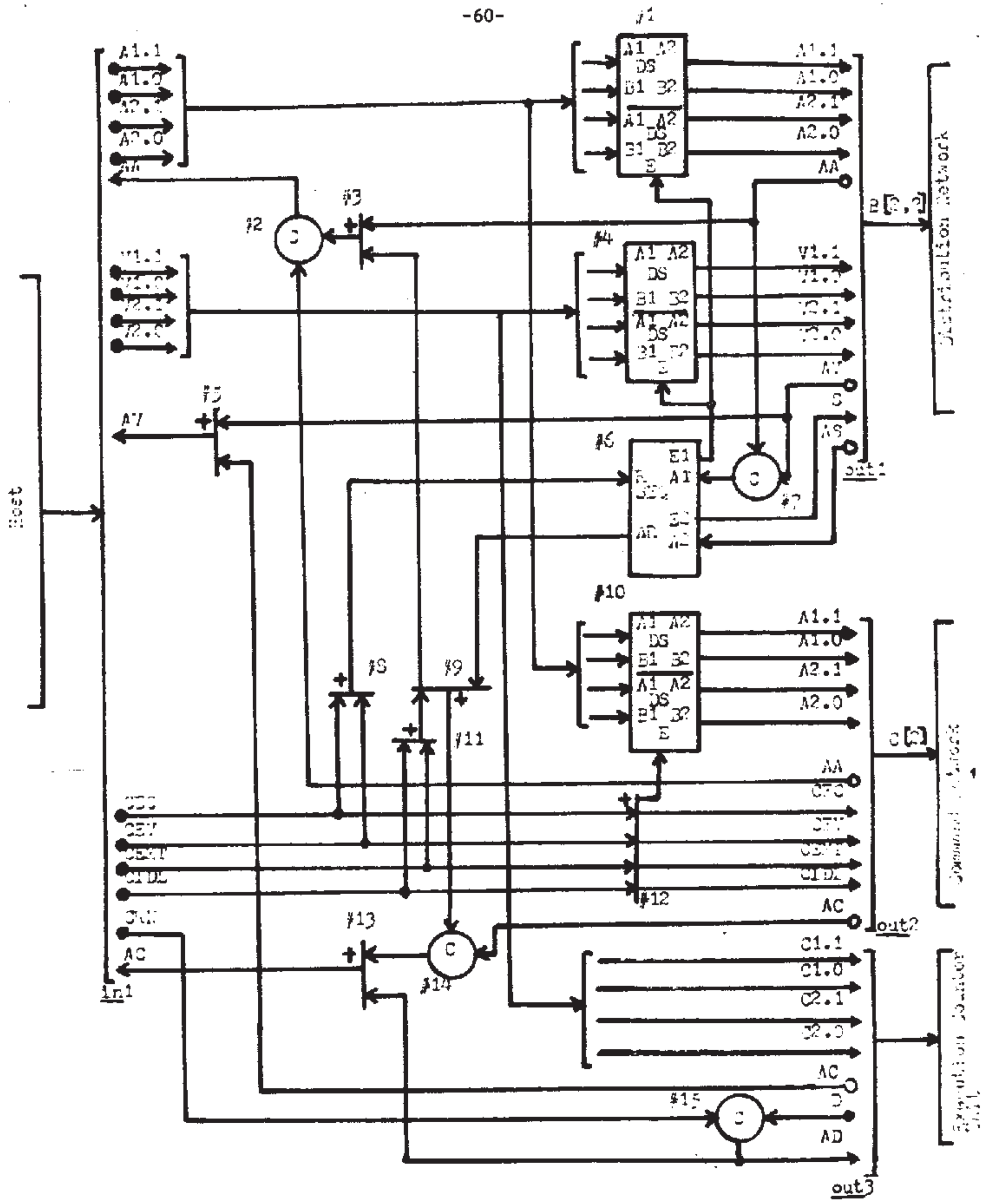


Figure A11. Specification of the Command Interpreter Unit.

for values. Modules (A11#9, A11#14, A11#13) generate the command acknowledge signal on <u>in</u>1:AC when command execution is complete.

5.2 <u>Execution Counter Unit</u>

The Execution Counter Unit is shown in Figure A12. Again values are assumed to have 2 bits for simplicity. The value supplied at port <u>in</u>1 by a <u>run</u> command is entered in the two bit pipelines (A12#5, A12#6). The decrement module (A12#7) has the property that the three-bit value represented on output pairs X, Y, Z is always one less than the two-bit value on input pairs A, B. Output pair X of (A12#7) represents 1 if a borrow is required to form the decrement. If no borrow is required, the value is discarded and the pipelines are left empty to receive the next <u>run</u> command.

This action is controlled by sequence module (A12#7) and the associated gate (A12#19) and data switch (A12#6) modules. Output E1 of (A12#17) is initially 1, and when data enters the pipeline (A12#5, A12#6) from port <u>in</u>1, the acknowledge is sent out <u>in</u>1:AC by data switch module (A12#16). The sequence module is then enabled on its E2 output and will set gate module (A12#19). This permits a signal on the input of (A12#19) to pass through the unit and enable the pipeline outputs. An <u>out</u>1:R/AR transaction requests one cycle of program execution through the Control Network (port <u>out</u>1). The <u>out</u>1:R signal, in conjunction with the acknowledge (<u>out</u>3:AR), resets the pipeline outputs if the decremented value has entered the pipeline (A12#15, A12#16). The pipeline outputs are then reenabled.

When a borrow is required by decrement module (A1#7), an <u>out</u>1:RF/AR transaction is sent through the Control Network to direct all Memory Cells to acknowledge that all requested execution cycles are complete. The acknowledge of this (<u>out</u>1:AR) causes the sequence module (A12#17) to enable its E3 output and reset gate module (A12#19). Module (A12#17) is then enabled on its E1 output to await another numerical input. The acknowledge of the <u>out</u>1:RF signal leads to an <u>out</u>1:D/AD transaction, which, by module (A12#27) signals completion of command execution by an <u>in</u>1:D/AD transaction.

6.0 <u>Command Network</u>

The Command Network (Figure A13) is a tree of Register Select Units (A13#1, A13#2, A13#3) that direct command transactions from the Controller (A13#4) to the Register Unit (A13#6, A13#7) specified by a q-bit address. For simplicity in this description, we suppose that one address bit is decoded at each level of the

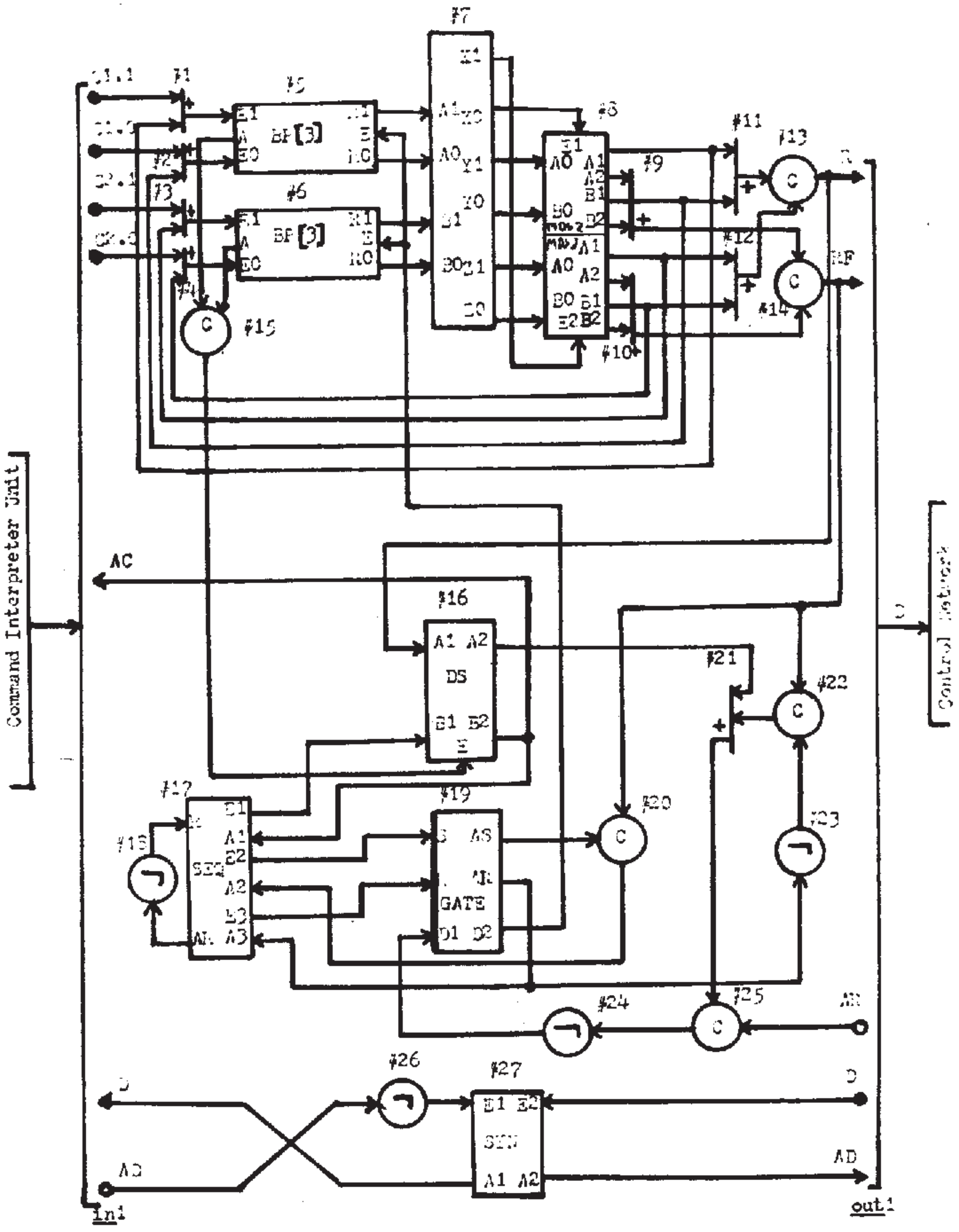


Figure A12. Specification of the Execution Counter Unit.

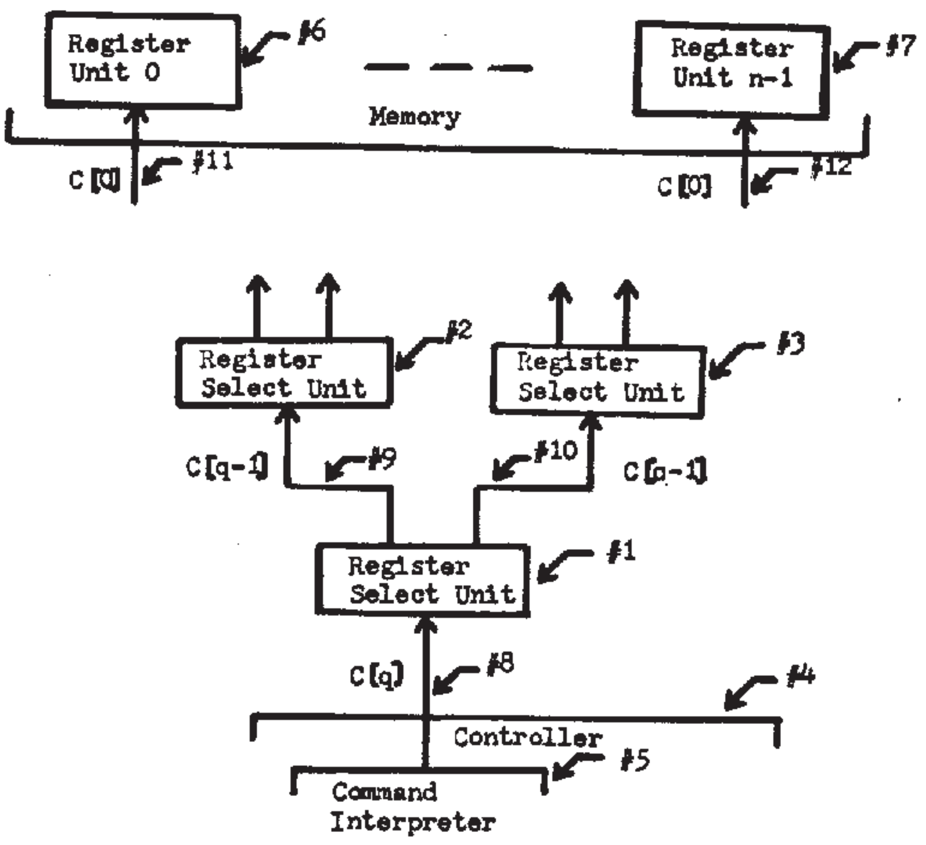


Figure A13. Specification of the Command Network.

tree. Thus the link (A13#8) connecting the Command Interpreter (A13#5) of the Controller to the First Register Select Unit (A13#1) is a type C[q] link and includes a pair of enable wires for each of the q address bits. The links (A13#9, A13#10) connecting the first Register Select Unit to the second level units (A12#2, A12#3) are of type C[q-1], and the links (A13#11, A13#12) connecting to Register Units are of type C[0].

Figure A14 shows the structure of a typical Register Select Unit having a type C[4] input link and two type C[3] output links. Multiple data switch modules (A14#1, A14#2) direct address and command signals to port out1 or out2 according to the most significant address bit. Acknowledge signals are returned through modules (A14#3, A14#4).

7.0 Control Network

The Control Network (Figure A15) is a tree of Run Enable Units (A15#1, A15#2, A15#3) that communicate request and completion transactions between the Execution Counter Unit (A15#4) of the Controller and Cell control structures (A15#5, A15#6) of the Memory. All links shown in Figure A15 are type D.

The specification of a Run Enable Unit is shown in Figure A16. Modules (A16#1, A16#2, A16#3, A16#4) forward request signals and provide immediate acknowledgement so many request transactions may propagate through the Run Network at once. Modules (A16#5, A16#6, A16#7) perform the same function for completion signals.

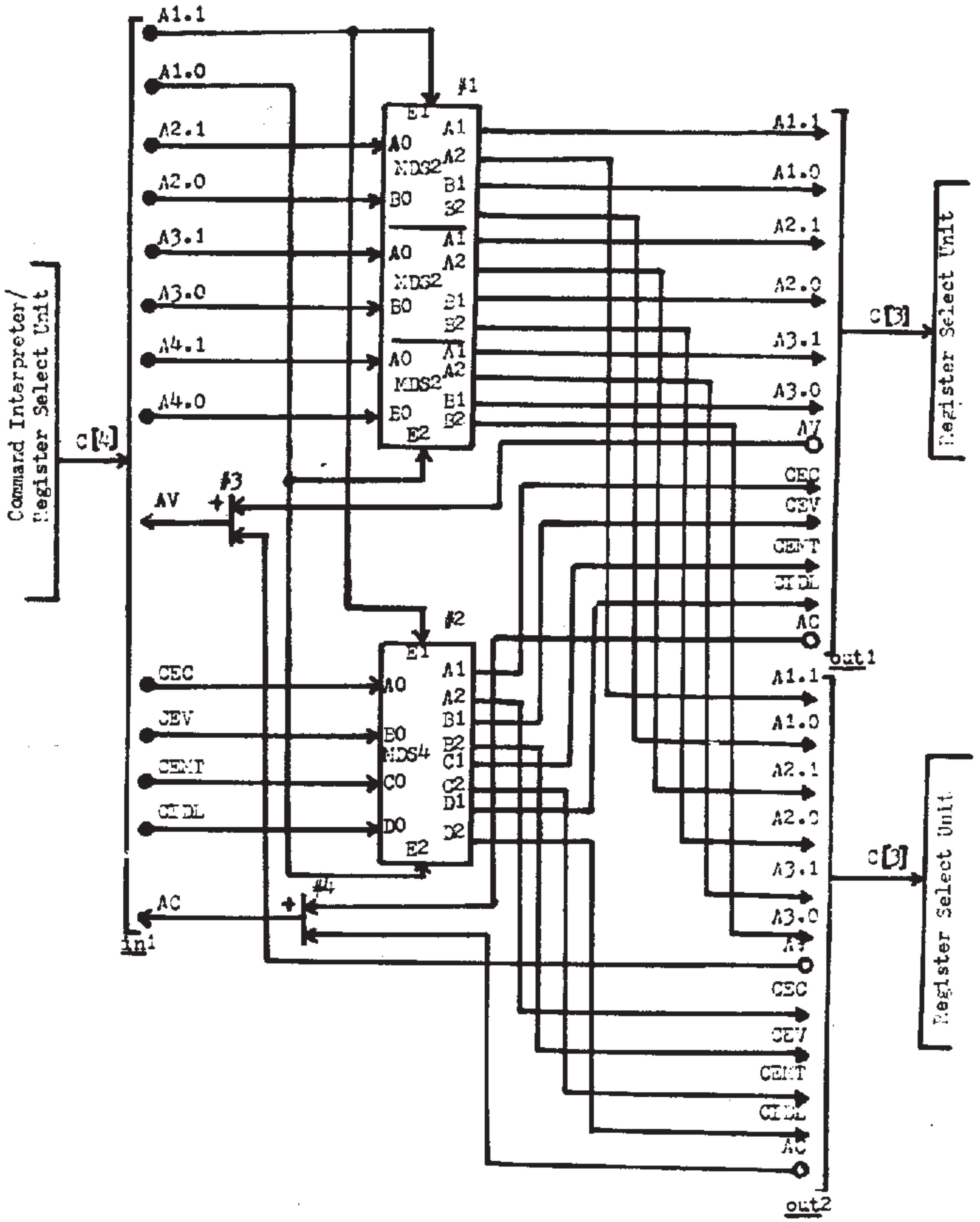


Figure A14. Specification of a Register Select Unit.

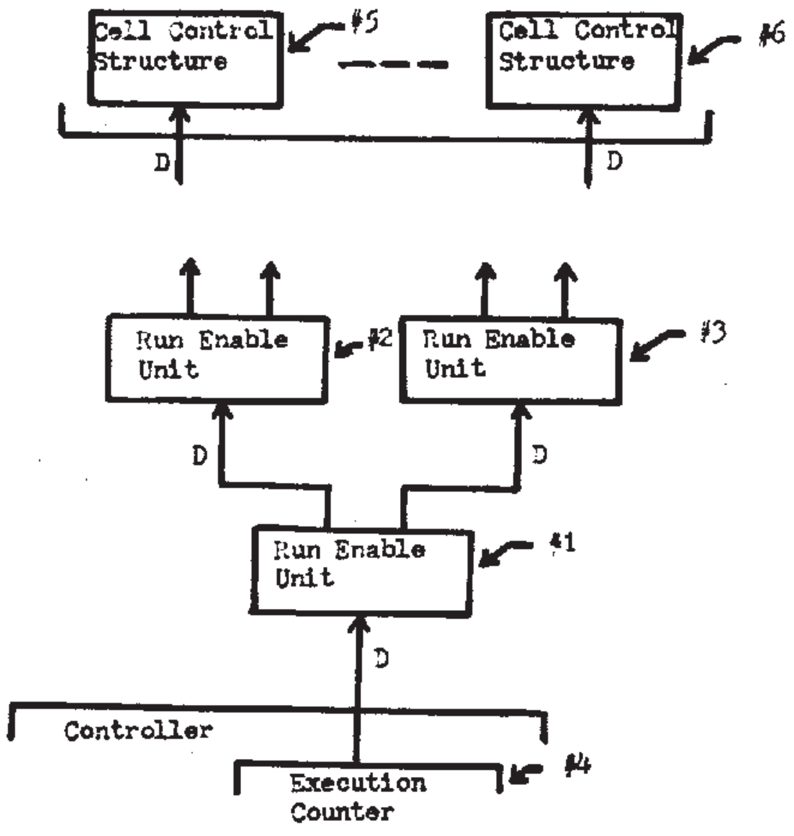


Figure A15. Specification of the Control Network.