NASA Contractor Report 178391

DIAGNOSTIC EMULATION: IMPLEMENTATION AND USER'S GUIDE

Bernice Becher

PRC Kentron, Inc.

Hampton, Virginia 23666

Contract NAS1-18000

December 1987

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

List of Figures

Appendices

Acknowledgement

The author wishes to acknowledge Earle Migneault of NASA - Langley Research Center who dreamed up the concept and the detailed design of the Diagnostic Emulation process. The author also wishes to thank Robert Baker, Scott Mangum, and Charlotte Scheper of Research Triangle Institute for providing some of the examples used in this document.

Bernice Becher
September, 1987

1. Introduction

In the future, computer systems will be doing more of the tasks that are now performed by humans. The area of commercial avionics is no exception. Sophisticated computer systems will increase their share of the tasks involved in the control and flying of the aircraft. In order to make commercial aircraft of the 1990's more efficient and profitable, new and advanced technologies will be used in their design and construction. Ways must be found to reduce the risk caused by these new technologies and thus to speed their acceptance. The Systems Validation Methods Branch in the Information Systems Division, is doing research in order to develop methods for fully integrating guidance and control functions, to identify system architectural concepts, and to establish a creditable validation process for advanced digital system designs. The contractor, PRC Kentron, is involved in this effort by providing support in the development of software to accomplish the latter goal, namely the development of methods for the analysis of the reliability of highly reliable, fault tolerant digital avionics systems. These advanced digital systems must be significantly more reliable than the systems now in use. What is generally meant by stating that the system must be highly reliable is that the probability that a system containing no failed components at the start of operation will fail during the first ten hours of operation will be less than approximately 10^{-9}. It is clear that digital computer systems that are to be highly reliable must be fault tolerant. Fault tolerance is the characteristic of the hardware and software architecture which allows the system to continue operating correctly in spite of the occurrence of physical faults, i.e., the detection of faults and the recovery to normal operation is handled automatically by the hardware and software and does not require manual intervention. This detection and recovery must be carried out within a specified period of time and must be done concurrently with the controlling of the aircraft.

Fault tolerant digital systems are implemented by first identifying the reliability goals of the system, and then selecting and incorporating fault-detection and recovery algorithms into the original design of both hardware and software. The tolerance to faults is usually accomplished with redundancy of components and algorithms which can reconfigure components in case of failures.

Once a fault tolerant digital system has been constructed, an important problem is how to evaluate the reliability of the system. There are two approaches to this problem. One approach is the use of analytic modeling techniques. A second approach which can be used in conjunction with analytic methods, is the use of emulation techniques.

This latter approach is currently being studied at the Langley Research Center. The idea being studied is that rather than basing reliability analysis on manufacturer's supplied data, or on expected probability distributions of failures of components to determine the response of a system to faults, a gate-level representation of the system is emulated. An algorithm has been developed to emulate any network of logic gates, flip-flops and tri-state devices. The algorithm is independent of the particular piece of hardware being emulated. A description of the particular target digital system is fed to a translator which converts the description to a form which the emulator can process. The processing of this representation of the target hardware by the software-implemented algorithm consists of the gate-level emulation of the target hardware. During this emulation, faults can be injected, and their effects studied.

The particular algorithm was developed with a major objective being conservation of host time and memory. The speed is important because the target

system must be allowed to run for lengthy time periods, and the conservation of space is necessary because of the large number of gates, flip-flops, and tri-state devices in any modern digital system. The algorithm employs a general model for all types of gates, i.e., "AND", "OR", "NAND", "NOR", "NOT", "XOR", and a single generalized model for all types of flip-flops. These general models allow for efficient use of computer memory. Time is conserved by processing only those devices in a given cycle whose input(s) have changed during the previous cycle.

This algorithm allows for the insertion of faults into the system, and for the observation of the response of the system to these faults. This allows for controlled and accelerated testing of system reaction to hardware failures in the target machine.

As an initial experiment, a horizontally-microprogrammable computer, the Nanodata QM-1, was chosen as the host system. The emulation algorithm was coded at the microcode level to take advantage of the parallel capabilities of the host machine and to exploit the speed advantages of executing code at the most primitive level of the host computer. All preprocessing of the hardware description and fault-injection data, as well as all post-processing of fault data is performed on a Digital Equipment Corp. VAX 11 which is interfaced to the QM-1.

The emulation algorithm has been used to emulate a simplified model of a "Toy" computer, the central processing unit of the Bendix BDX930, and the communicator interstage unit of the Fault Tolerant Processor. Working emulators are resident in a QM-1 computer and a Vax computer in AIRLAB, the Avionics Integration Research Laboratory, at the Langley Research Center. These emulators will be used as general reliability analysis tools for highly reliable, fault tolerant avionics system. A complete and detailed discussion of the concepts inherent in the technique is given by Migneault[2]. The remainder of this document will describe in detail how the algorithm was implemented at NASA/LaRC and instructions on how one goes about using the system.

2. General Description

2.1 Overview: General Principles and Assumptions

The Diagnostic Emulation Technique is a general technique which allows for the emulation of a digital hardware system. The technique is general in the sense that it is completely independent of the particular target hardware which is being emulated. A description of the hardware to be emulated is presented to the emulation program in the form of input data.

The technique is a hybrid one in that parts of the system (the network) are described and emulated at the logic or gate level, while other parts of the system (the functional subsystem) are described and emulated at the functional level in order to save time and unnecessary complexity. It is up to the user of the emulation program as to which parts of his system are to be emulated at the gate level and which parts are to emulated at the functional level.

The network to be emulated at the gate level consists of a set of devices (gates, flip-flops, and tri-state devices), and a set of connections among these devices.

Each input and output to or from a device may assume one of two values, namely high (represented by 1) or low (represented by 0).

The basic unit of time(t) used by the emulator is the time it takes for the input signals on a logic device to be propagated through to the output of that device. It is assumed in this technique that the propagation time for all devices in the network is the same and remains constant throughout the emulation. This is not an inherent limitation of the diagnostic emulator, although unit delay is assumed for this implementation.

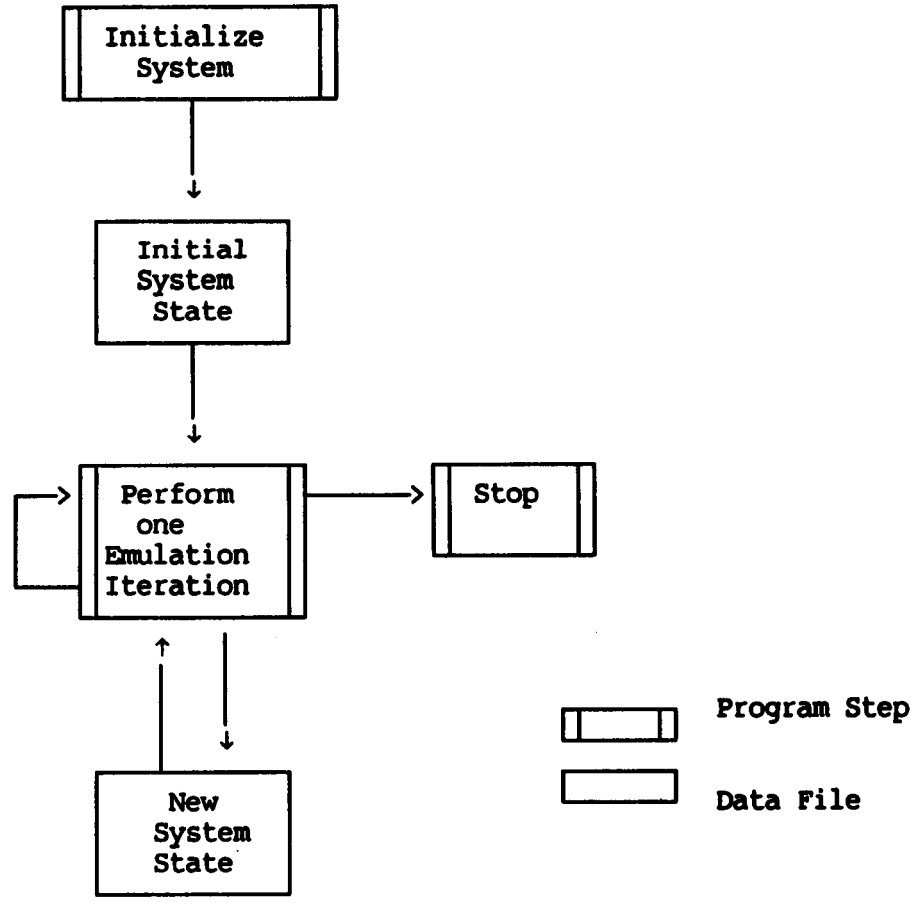
The technique allows for a very flexible method in which the gate-level network and the functional subsystems can communicate with each other. This method also allows the user to define any type of subsystem he wishes as long as he can describe it in terms of a data structure and a subprogram module that operates on the data structure and optionally also operates on the gate level network.

The state of the entire system at any given time consists of state descriptions of all logic devices in the network, state descriptions of all connections among devices, and a state description of the functional subsystem. The emulator must have given to it at t=1 the initial state of the entire system. The emulation then consists of a series of iterations, one for each time step. Given the current state of the system at time T, the emulator calculates the new state of the system at time T+1. It continues these iterations until it reaches the stop time specified by the user.

The emulation technique is event-driven in the sense that for a given iteration, only those logic devices are processed whose output values have changed during the previous iteration.

Important functional capabilities which have been incorporated into the NASA LaRC implementation are: the ability to insert and/or remove stuck-at faults at user-specified times into the logic gates and/or into the ROMS, the ability to input to the digital logic at user-specified times from sources external to

the simulation, and the ability to output from emulated logic to sources external to the emulation either at user-specified equally spaced time periods or at times controlled by the internal logic. The technique and its concepts are basically independent of any particular implementation. All of the characteristics which have been described above are general concepts of the technique and are independent of any particular implementation. The overall structure of the technique is depicted in Figure 1.



Overall Structure of the Technique

Figure 1

3. System Structure

The emulation system as it has been implemented at NASA/LaRC consists of two parts. The first part is the "Initialization" system which calculates a consistent initial state for the target hardware and generates this initial hardware state in the binary form required by the second part. Part 2 is the emulator. The emulator begins with the initial machine state and performs the emulation as per the user's specifications.

The initialization program requires as input a description of the gate-level network in the Diagnostic Emulator Netlist Format (DENF), and a list of the initial contents of any memories being emulated at the functional level, in the Diagnostic Emulator Memories Format (DEMF). The Initialization Program is described in detail in Section 5.4.1. It is the user's responsibility to provide these two required inputs to the initializer in the formats required. It is thus necessary for the user to provide some sort of "preprocessor" to generate the netlist in DENF format and the memories in DEMF format. To date, two preprocessors have been developed to provide these descriptions to the initializer in the required format. The first was developed at NASA for the CYBER computers in the Analysis and Computation Section and uses a NASA-developed netlist language for its input. The second preprocessor was developed by the Research Triangle Institute[1] and uses Futurenet as its input language.

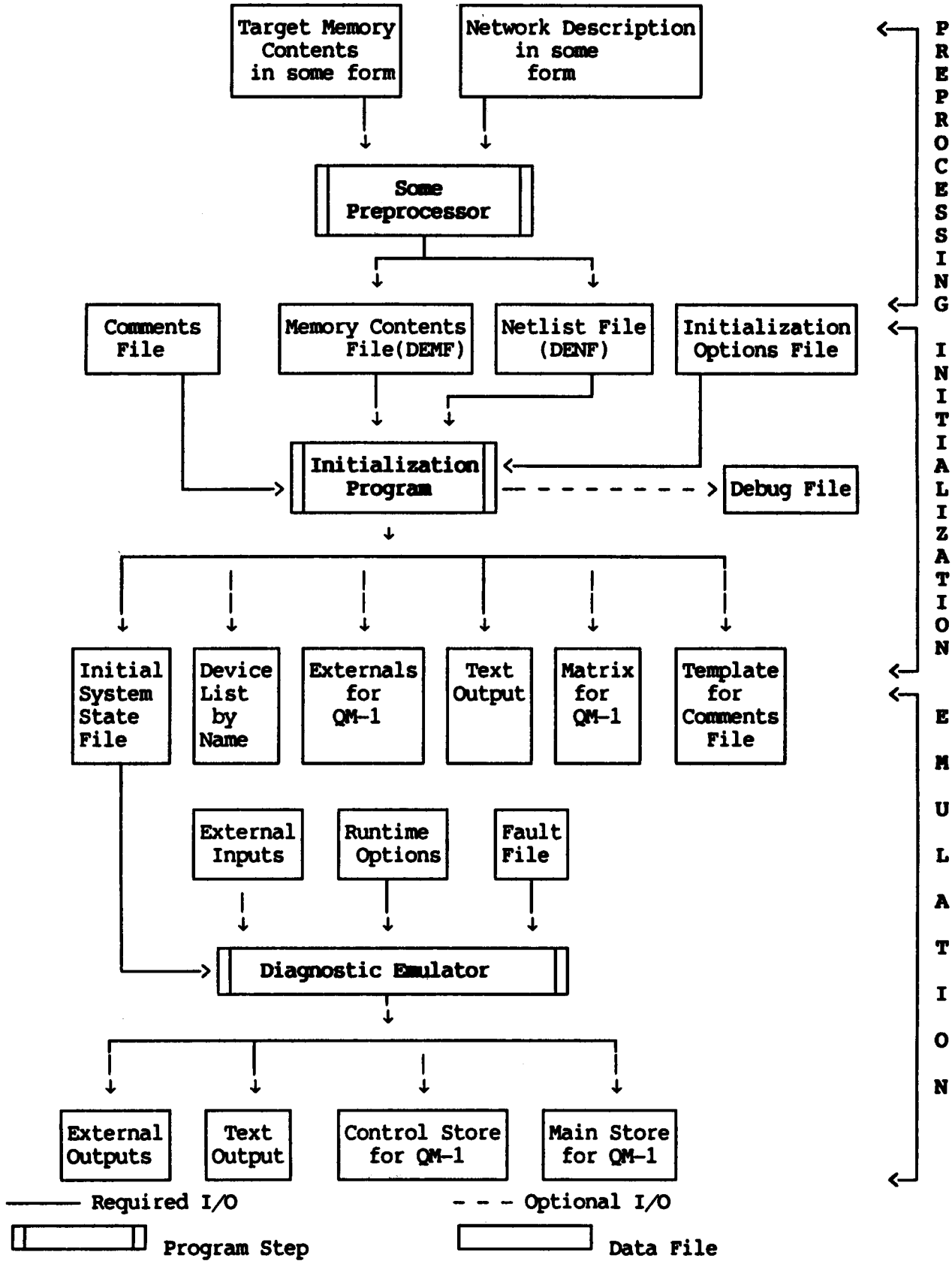
This document does not include any discussion of preprocessors. A complete description of the DENF and DEMF formats is given in Section 5.4.1.2. Thus a user can generate his own translator or preprocessor to translate from the language of his network description and from his memory format to the required formats.

The initialization program produces the complete network description and the memories' contents in the binary form required by the emulator. In addition, the initializer calculates (if possible) a consistent initial state for the entire system.

The emulator then uses the initial machine state generated by the initializer together with other user-supplied information to perform the emulation.

3.1 System Flow of Control

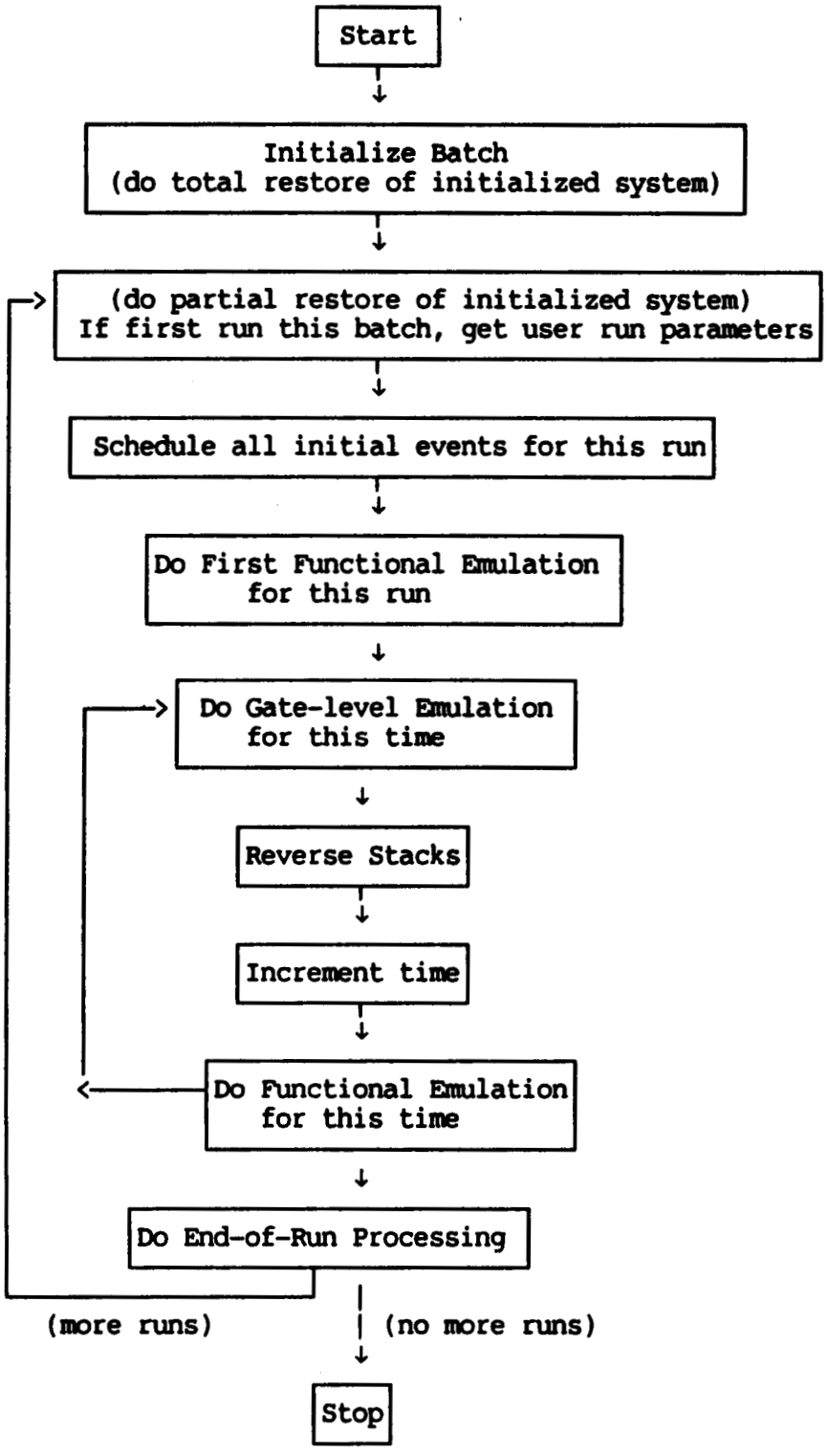
The overall program and data flow for the preprocessor, initializer, and emulator are shown in Figure 2. The general idea is that for a given target machine to be emulated, the preprocessor and initialization systems need be run once (or several times if errors or inconsistencies exist), i.e., until the system is successfully initialized. At that point the initial state of the system has been saved in a binary form for the emulator, and the emulator can be run as many times as desirable, varying its inputs, without having to rerun the preprocessor or initialization systems.



System Flow of Control
Figure 2

3.2 Emulation Flow of Control

One of the user-supplied inputs to the emulator is the "Fault File". The data in the fault file controls the emulator. All the data in one fault file is referred to as a "Batch". The fault file consists of any number of individual fault lists, each of which causes one "Run" to be executed. A "Run" is an emulation which begins at time t=1 and continues until a user-specified stop time. An individual fault list describes when and what kind of faults are to be inserted for the run, and when the run is to stop. Each run in the batch begins by re-initializing to the initial state as defined by the initialization program. Each run makes use of the same external inputs file. The differences from one run to another within the same batch are caused by the different fault list for each run. The fault file is described in detail in Section 5.4.2.2.3. The batch is completed when all fault lists have been processed, or in other words, when the last run has completed. In summary, a batch consists of many runs. For each run the gate-level network and the functional subsystems are the same. The initial state of the machine and the external inputs are the same. The faults injected and the stop time may vary for each run. The flow of control during which the emulator processes one batch is shown in Figure 3.



Emulation Flow of Control

Figure 3

4. Implementation of Diagnostic Emulation Technique

4.1 Overview of Implementation

The Nanodata QM-1 computer is a high-speed general-purpose digital computer. It was chosen for the first implementation of the Diagnostic Emulation Technique because at the lowest level it is horizontally microprogrammable. See the QM-1 Hardware Level User's Manual[3] for a detailed description of the QM-1 architecture. It should be noted that the QM-1 at NASA/LaRC contains three levels of memory. At the highest level is the main store memory which consists of 500K of 18-bit words. The control store memory consists of 40K of 18-bit words. At the lowest level is the nanostore which consists of 1K of 360-bit words. Microcode is stored in the control store, while nanocode is stored in the nanostore.

The emulator was implemented on the QM-1 as follows: The algorithm which emulates the gate-level logic was written in nanocode at which level the primitives of the QM-1 hardware are controlled in a parallel manner, i.e., during each t-step of the QM-1 many different nanoprimitives may be executed simultaneously. The QM-1 has a nanoassembler which was used to assemble the nanocode which implements the gate-level emulation algorithm. The algorithm which performs the functional parts of the emulation was written in microcode (on the QM-1, each microcode instruction is carried out by a sequence of nanocode instructions and is therefore one level higher than nanocode). The microcode language used was "Multi", and the functional algorithm or "Driver" was assembled with the Microcode Assembler. Two new microcode instructions, namely "Emul" and "Iemul" were defined as extensions to the "Multi" language, and when used in the Driver, cause the appropriate parts of the nanocoded gate-level algorithm to be executed. "Iemul" causes the initialization of the gate-level data structures to be carried out, and "Emul" causes one time period or one stack of the gate-level network to be processed. Both of these multi-extension codes then appear as instructions in the microcoded Driver. The nanocode and microcode are combined into an executable form as described in Section 5.1.2.

The front end program for the emulator, namely the Initialization program, was written for the Vax 11 in Fortran. Because of the need to check the results of the QM-1 emulation, an emulator was also written in Fortran to run on the Vax. The Vax emulator was naturally much simpler to write than the QM-1 emulator but runs about 36 times slower than the QM-1 implementation. The result is that the user now has two options as to how he will run the emulator. He must run the Initializer on the Vax, but then has the choice of whether to do the actual emulation on the Vax or on the QM-1. The user must weigh the disadvantage of the added complexity of using the QM-1 against the advantage of the gain in speed. It should be pointed out that the QM-1 emulator was implemented first, and that the Vax emulator was written to conform to the QM-1 18-bit word, and has basically emulated the control store and main store of the QM-1.

4.2 Models

4.2.1 Gate-level Network Model

Any network to be emulated at the gate level consists of a set of gates, flip-flops, and tri-state devices, and a set of the connections among these devices. Any input or output to or from a device may assume one of two values, namely high (represented by 1) or low (represented by 0).

4.2.1.1 Simple Gates

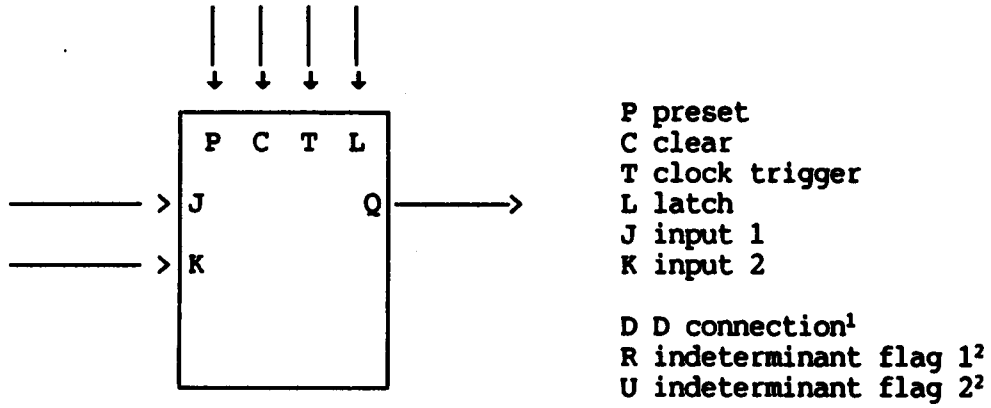
A gate may be any of the following types: AND, NAND, OR, NOR, NOT, XOR, NXOR. Normally, a simple gate is enabled; however, the faulting of a gate (output stuck at 1 or 0) is implemented by disabling the gate.

4.2.1.2 Tri-State Devices

A tri-state device is any of the simple gates listed above, but in addition has an enable/disable input. The internal value (namely the output value consistent with the inputs) of the tri-state device is always kept current, but if the device is disabled, its internal value will not be propagated to its output line, but rather its output line will be stuck at either 0 or 1 (the value chosen by the user in the netlist for that particular tri-state device) until the tri-state is enabled.

4.2.1.3 Flip-Flops

A general model for a flip-flop is used by the algorithm. Note that a flip-flop is not modeled at the gate level. The general model for the flip-flop is as shown in Figure 4.



[1] A "D" connection is merely one in which the K input is always the complement of the J input.
[2] See A-29, Legends for Internal Connectors, A-6, Flip-Flop Trigger Chart, and Migneault[2]

Flip-Flop Model

Figure 4

By using these lines appropriately, all of the useful edge-triggered types of flip-flops can be modeled, as described by Migneault[2]. Note that this model accomodates only one output, namely the Q output. If the QBAR output is desired, it can be obtained by adding an additional flip-flop with the inputs

reversed from those of the Q flip-flop. See A-42 for an example of a Q, QBAR flip-flop set. The preset and clear lines for all flip-flops in a network can be active high or active low, as defined by the user. Note, however, that all flip-flops in a network must be either active-low or active-high. The clock trigger for each flip-flop can be either upward edge-triggered or downward edge-triggered, again as specified by the user. In this case, however, the choice for each flip-flop is individually controlled. For each device in the network, a data structure exists which at all times reflects the state of that device.

4.2.1.4 Event-Driven Feature

The emulator technique is event driven; that is, during each time period a given device will be processed only if a specific event has occurred during the previous time period, namely that device's output value has changed. Any device whose output value did not change during the previous period need not be examined since it cannot affect any other device.

4.2.2 Functional Subsystem Model

Any subsystem which is to be emulated at the functional level is implemented with a data structure (called an action data structure) representing its state at any given time, and with an action subroutine module which performs the specified function. Some examples of functional emulations which have been implemented on this system are ROMS, RAMS, fault injection and removal, external inputs to the network, and external outputs from the network.

In order to implement functional emulation, event scheduling is used. While the gate-level network emulation is synchronous in the sense that at each time interval the devices are processed whose output values changed during the last time step, the functional emulation is asynchronous in that functional events do not necessarily occur at fixed time intervals and therefore must be scheduled. To provide for this, two data structures are used. An event list contains all events currently scheduled to be executed at specific times, and a free space list contains a list of memory slots currently available for use by the event list. Because the number of scheduled events grows and shrinks, there is dynamic allocation of space between the two lists, i.e., space is taken from and returned to the free space list according to the space requirements of the event list. Each event scheduled points to the head of an action list. Each action in that list is to be executed at the time specified in the event.

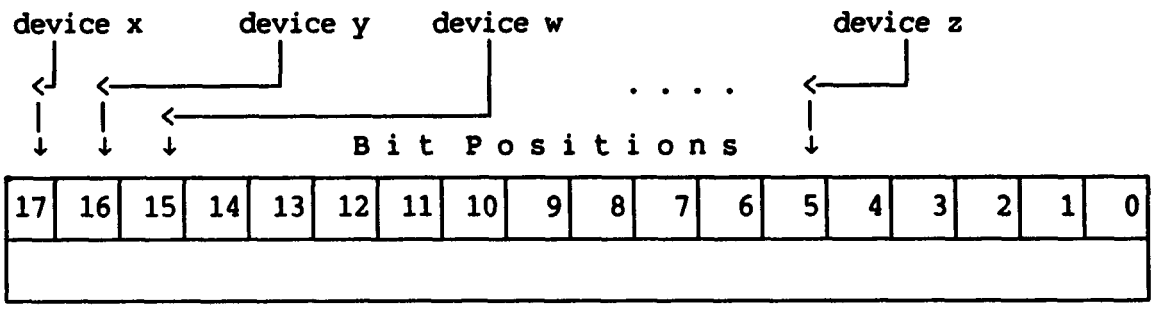
4.3 Data Structures

4.3.1 External Registers

In general, external registers are used when data is to be communicated between the gate-level emulation and the functional emulation. The user may direct that the emulator set up a block of contiguous external registers in control store and/or a block of external registers in the main store of the QM-1. Each external register is an eighteen-bit word in the QM-1 memory. Each block of external registers, no matter where it may be in the QM-1 memory, for the user's purposes, is labeled beginning with register number 1, and the rest of the block is numbered consecutively.

An external register can receive its value in two different ways. The user can specify in the netlist that the output of any particular device in the network feed into any bit(s) in one or more external registers. Thus during the emulation the bit in the external register at all times is a copy of the output line of the associated device. This is a technique for collecting in one contiguous group of bits in the QM-1 memory, the output values of any selected set of devices. This use is shown in Figure 5.

An external register can also receive its value from the functional subsystem emulation during the execution of an "action", and typically could then be used by any other action. An example of the use of external registers is the data and address registers used in the implementation of memory reads and writes. See Figures 9 and 10 for illustrations of these uses.



External Register

Figure 5

4.3.2 Network Connections

There are two types of connections within a gate-level network, namely internal connections and external("pseudo") connections. An internal connection is one which goes from the output of a network device to the input of a network device which may be the same device as the source device or a different one. In any case, both the source and destination of an internal connection are devices within the network. In the case of an external connection, the source is a device within the network, but the destination is

an external register in that it does not exist within the network being emulated, but is a register in the QM-1 created as a means for implementing the functional part of the emulation. An external connection is then one which goes from the output of some device in the network to a specified bit in some external register. Once this connection is set up in the netlist, then during the emulation the bit in the external register at all times is a copy of the output line of the associated device. Thus this is a technique for collecting in one contiguous group of bits in the QM-1 memory, the output values of any selected set of devices. External register connections are defined by the user in the netlist and may be used for any functional subsystem desired. To date, they have been used to implement memory reads and writes by emulating the data and address registers and for external inputs and outputs, again by serving as the data registers. They are also used in memory reads and writes and external outputs by holding the values on the control lines which then are used to trigger the particular action. Figure 6 shows diagrams of both types of connections.

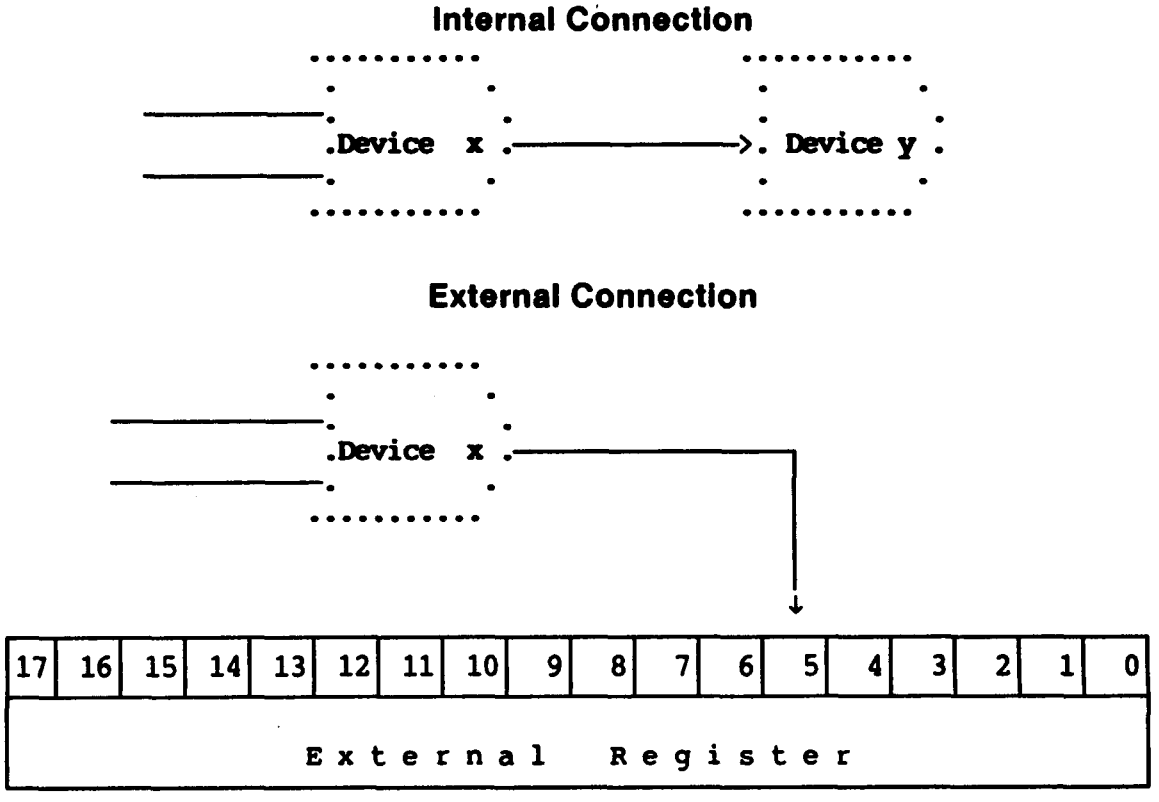
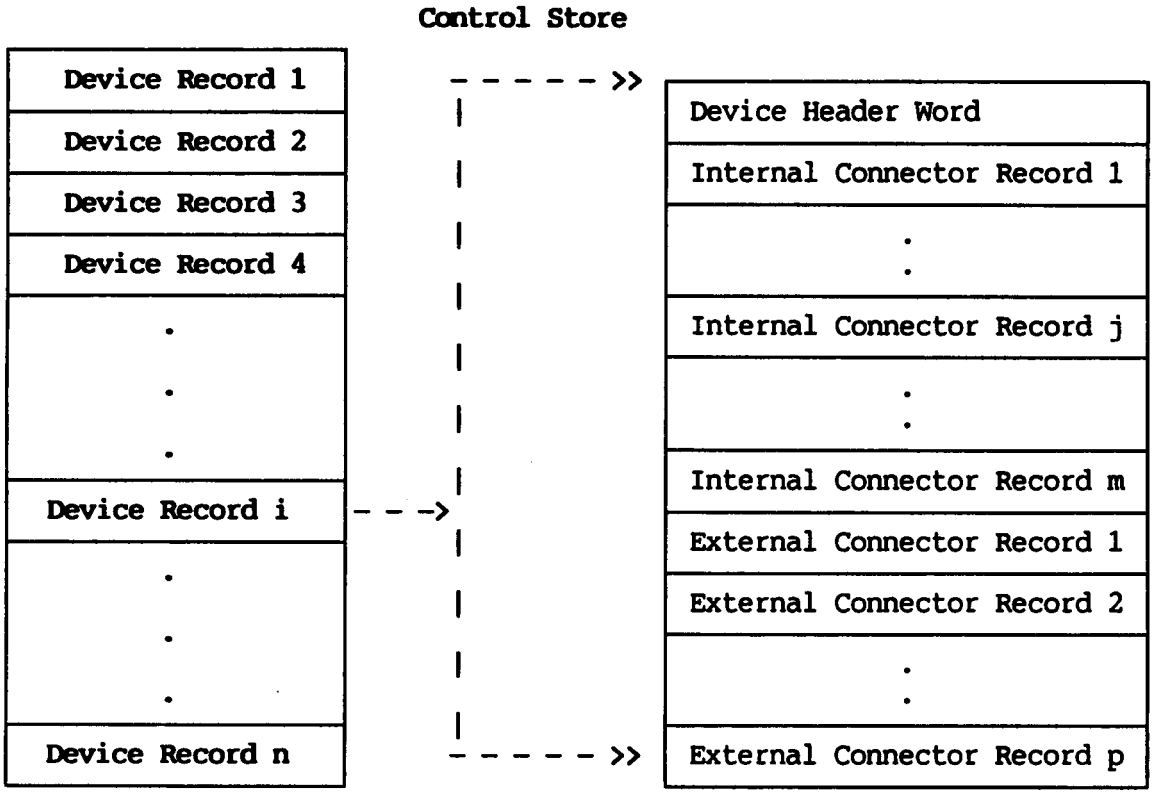


Figure 6

4.3.3 Hardware Description Matrix

The hardware description matrix is a binary representation of the entire network of devices and interconnections among the devices. This description of the network of gate-level logic is represented in the control store of the QM-1 by a set of Device Records, each record representing one device in the target network. A device can be a gate, flip-flop, or tri-state device. The device records need not be in any particular order.

Each device record is made up of exactly one Header Word, followed by one or more internal connector records, followed by zero or more external connector records. The Header Word fully describes the state of the device at any given time. The format for a particular Header Word varies depending on the type of device. The formats for the various types of devices is shown in A-25. An internal connector record describes a connection from the output of this device to the input of another device in the network. An external connector record describes a connection from the output of this device to an "external register" which is pseudo in that it does not exist in the real target network, but is used for implementation of functional emulations. The components of a device record must be in the order stated above, and they must be contiguous. The connector records for a particular device may be in any order, and the pseudo connector records may be in any order. The overall structure of the Hardware Description Matrix is shown in Figure 7.



(Layout for target with a total of n devices, device i has m internal connections and p external connections)

Hardware Description Layout

Figure 7

4.3.4 Stacks

The emulator gate-level technique is event driven; that is, during each time period a given device will be processed only if a specific event has occurred during the previous time period, namely that device's output value has changed. Any device whose output value did not change during the previous period need not be examined since it cannot affect any other devices. The method used to efficiently implement this event-driven capability is the maintenance of two stacks. At any given time, one stack is identified as the "c" stack, and the other is referred to as the "cbar" stack. During each time period, the list of devices which changed during the previous period is known as the c stack. The emulator takes one device at a time, namely the source device, off the c stack and processes it. For each source device, it examines each destination device into which this device feeds, as it is a possible candidate for a change in output value this time period. If the destination device does not have a change in output value, the emulator proceeds to examine the next device into which the source device feeds. If the output value of the destination device does change, it is added to the list on

the cbar stack. Thus it can be seen that at the end of this time period, the cbar stack contains a list of all devices whose output values have changed during this time period. A device placed on the cbar stack may have changed output value an even or odd number of times during this time period. If it changed an even number of times, it is not processed during the next time period. As part of the initialization for the next time period, the time is incremented by one, the cbar stack now becomes the c stack, and the previous c stack becomes the new cbar stack, and is cleared, to be built up again during the new time period. Thus it can be seen that the program is always reading the c stack and writing the cbar stack, and also that the identity of the two stacks reverses itself each time period. It has been observed that for any given time period, only a very small percentage of the devices in a network need be examined.

At the start of each emulation run, stack c must contain the device identifiers for all devices whose output values changed during the previous time period, namely t=0. At the end of processing for the first time period, stack cbar contains the device identifiers for all devices whose output values changed during this first period. The stacks are then reversed after each time period. The identifiers on the stack are not in any particular order. The program maintains a pointer to the base and top of each stack. Each stack grows upward to a higher control store location, i.e., as a device is added to the stack, it is pushed onto the top of the stack and the top of stack pointer is incremented. As each device is processed, it is popped off the top of the stack, and the top of stack pointer is decremented. At the beginning of each run, stack c must contain at least one device identifier, and stack cbar contains no identifiers. Figure 8 shows the general structure of a stack beginning at control store location x and containing n devices:

control store address	contents	
x	device identifier	<---- Base of stack
x+1	device identifier	
.	"	
.	"	
.	"	
x+n-1	device identifier	<---- Top of stack

S t a c k

Figure 8

4.3.5 Events

Events which are emulated at the functional level must be scheduled because they do not necessarily occur each time period. To implement this scheduling of events, two singly-linked lists are used, namely the event list and the free space list. Both lists are maintained in the control store of the QM-1. A pointer to the head of each list is also maintained in control store. Each element in the event list is a record consisting of three words. The first word contains the time at which the event is to be executed or

emulated. The second word contains a pointer to the next event in the event list which is to be executed at some time greater than the time for this event. The links in the event list are maintained so that the list is always in ascending time sequence. The last event in the event list contains a null (0) pointer in the second word. The third word in the event list is a pointer to the first action in control store which is to be executed at this time. The actions are also maintained in a singly-linked list, so that many different actions may be executed at one specified time. An action list is in the reverse order to that in which the actions were scheduled. The format for the event list data structure is shown in A-2, and a diagram showing the event list as it relates to the free space list and the action list is shown in A-1. Scheduling of events and actions is shown in A-4 and A-5 respectively.

4.3.6 Actions

Each unique functional subsystem is implemented through the use of an "action". An action is composed of an action subprogram module, an action data structure, and optionally other data structures required for the particular action. In general, when the time period occurs for which the action has been scheduled, the specified action subprogram is given control, and it "executes" the action by making use of the corresponding action data structure(s).

There must be in the QM-1's control store memory one action record for each unique action be performed. The number of words in each action record varies according to the type of action; however, each action record contains at least three 18-bit words. The format of the first three words is the same for all actions. The remaining words, if any, vary according the action.

At any particular time, an individual action is scheduled to be executed, or not, as indicated by the "scheduled" switch in word 1 of the action. If it is not scheduled, it is not linked into any of the action lists. If it is scheduled, the appropriate pointers link it into the action list for the event scheduled for the time at which this action is to be executed.

The importance of the actions feature in the scheme of the diagnostic emulator cannot be overemphasized. Associated with each action data structure must be a subroutine module which is to be called when the time period is reached for which the action has been scheduled.

To date, six different action subprograms are available to the general user of the emulator. These actions are: write to memory, read from memory, stop run, "do operations", do external inputs, and do external outputs. Each of these actions is described in detail in Section 4.3.10. In addition to these supplied actions, the fact that the user can write as many of his own actions as desired is the feature which makes the emulator so flexible. The implication is that any functional emulation which can be written in subprogram form by the user can then be used in conjunction with the gate-level emulation. Thus it is possible for an action written by a user at the functional level to actually access and/or modify the state of the gate-level network. It should be noted that there is not necessarily a one-for-one mapping between the action data structures and the action subprograms. Typically, there may be many action data structures associated with one subprogram. For example, there is one read action subprogram, but there must be one memory control block, one emulated memory, and one action data structure for each ROM or RAM to be emulated. The subprogram performs the actual read action but the action data structure specifies the location of the memory to be read, the size of the target word, the locations of the data and address registers, etc. In other words, the subprogram is general for most reads, but the action data structure

is specific to the memory. It is usually possible for all the memories to make use of the same read memory subprogram. The same is true for the write memory subprogram.

4.3.7 Master Action Control Register

For each target emulation, one external register, namely the "action control register" must be designated to control the triggering of any actions associated with functional subsystem emulation. The high-order bit of this register is the master action control bit for all actions. Each device which controls the triggering of an action should have an external connection into the high-order bit of the action control register, in addition to having an external connection into some control bit in the action control block. For each time period, the emulator checks the high-order bit of the master action control register. If it is on, the emulator knows there is at least one action to be scheduled, and proceeds to check all the bits in control bit words of the action control block. For each bit in the control bit word which is on, the appropriate action is scheduled. On the other hand, if the high-order bit of the master action control register is off, the emulator knows that no actions are to be scheduled and need not check the individual control bit words of the action control block.

4.3.8 Action Control Block

For all functional subsystems (actions), a set of action control blocks is allocated in the control store of the QM-1. Each block will contain one or more action control records. There is one action control record for every eighteen action control lines. An action control record consists of one word, referred to as the "Control Bits" word, to represent the values of the eighteen control signals (this word is actually a "pseudo" register which is fed by appropriate devices in the netlist), and two additional words for each control line. The last action control record may not actually represent a full eighteen control lines, but the full amount of storage (37 words) is allocated in any case. The data structure for an action control block is illustrated in A-3. Note that words 1 through 37 will be repeated contiguously until all action control lines for which actions can be scheduled have been accounted for.

When the emulator has determined that the master action control bit is on, it proceeds to check each bit in the "control bits" word. When it finds a bit that is on, it accesses the appropriate two words in the action control record for the address of the corresponding action and the appropriate delta time. It then schedules the action whose address it has accessed to be executed at a time equal to the current time plus the delta time it has accessed.

4.3.9 Emulated Memories

A contiguous block of main store in the QM-1 is allocated for each ROM or RAM to be functionally emulated. Each QM-1 main store word contains eighteen bits. The number of QM-1 words necessary to represent one target memory word depends completely on the number of bits in the target word. If the target word has 18 or less bits, then only one QM-1 word is needed for each target word. In any case the target bits are stored in the QM-1 with the highest order target bits stored in the high order bits of the lowest QM-1 address used. The target word may be stored in the QM-1 either right or left

justified, as determined by the user. It is not necessary for two or more memories to be contiguous to each other in the QM-1, but within one memory, all QM-1 words are contiguous. See A-33 for a layout of the memory data structure and A-34 for the Emulated Memory Layout.

4.3.10 Action Descriptions

Following are descriptions of the actions which have been implemented to date:

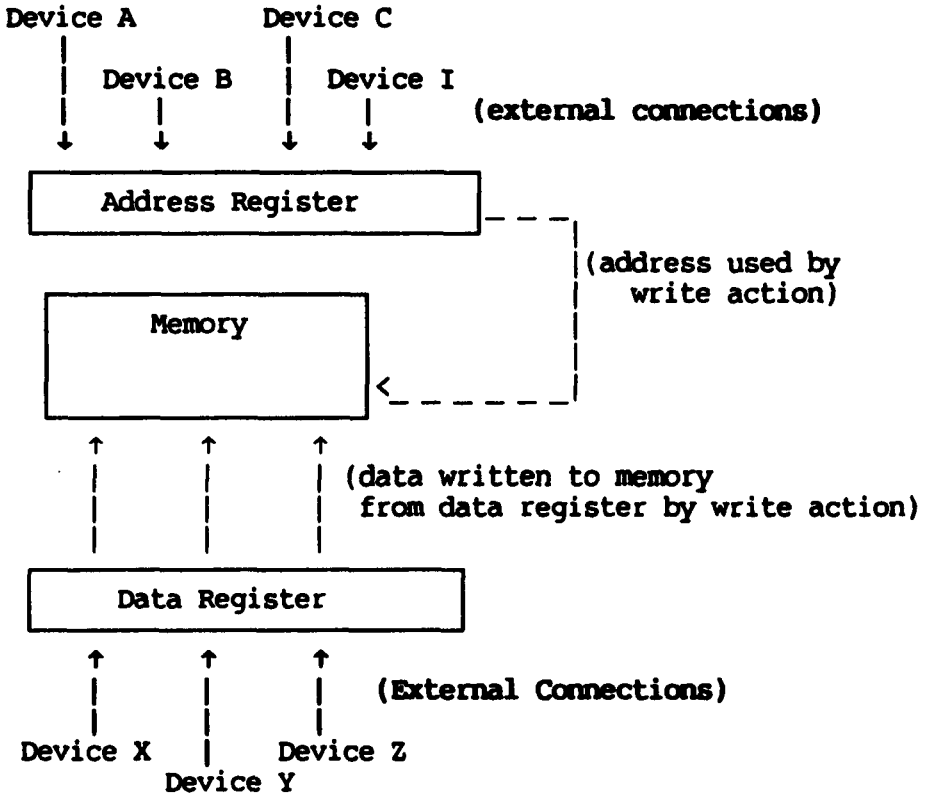
4.3.10.1 Write Memory Action

The write action is used to write a word from a data register to a target word in ROM or RAM. See A-36 for the Write Memory Action Data Structure Layout. The action is scheduled when the controlling device transitions from low to high. The action is scheduled at a time equal to the current time plus the delta time in the second word of the write action data structure.

When the current time reaches the scheduled time, the write action is executed. The emulator reads the emulated address register and shifts the bits the appropriate amount to right-justify the target address. Next it checks this address against the low and high valid target addresses in the seventh and eighth words of the action data structure. If the address is not within this valid range, a message is outputted, and the program aborts. If the address is valid, the actual QM-1 address for the target word is calculated as:

```
QM-1 address = relocation constant +
                  target address * number of QM-1 words per target word
```

(The number of QM-1 words per target word is obtained from the first word of the action data structure, and the relocation constant is obtained from the fourth word of the action data structure). The program then reads the data register as pointed to by the sixth word of the write action data structure. It then stores the data from the data register into the QM-1 address as calculated above. This procedure is repeated for all QM-1 words representing the one target word. Figure 9 is a diagram of a write memory action structure.



Write Memory Action Structure

Figure 9

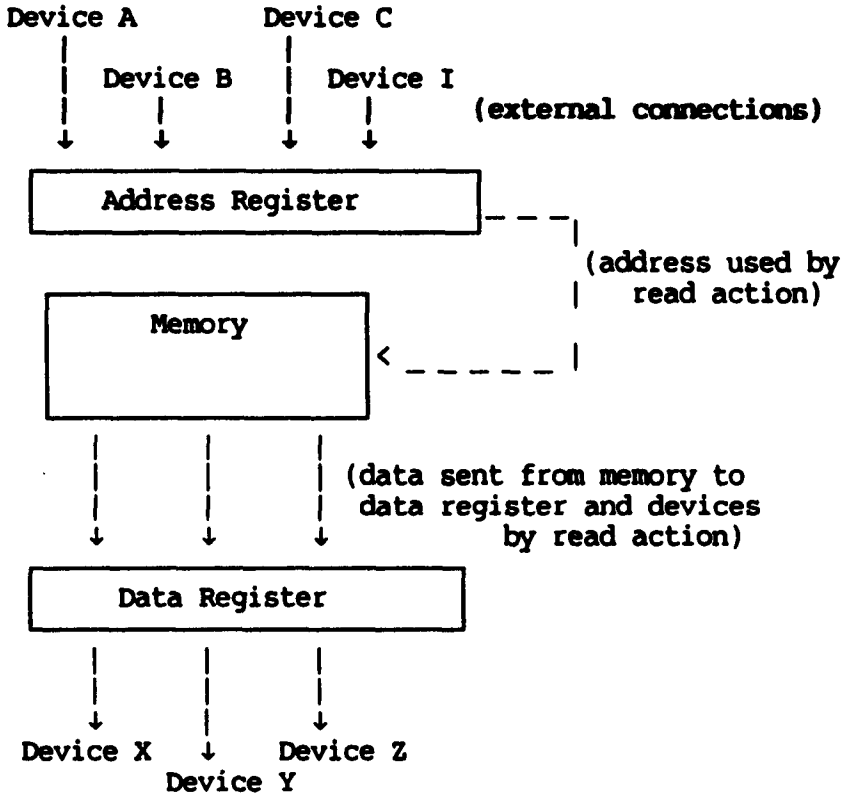
4.3.10.2 Read Memory Action

The read action is used to read a word from a target ROM or RAM. See A-37 for the Read Action Data Structure Layout. The action is scheduled when the controlling device transitions from low to high. The action is scheduled at a time equal to the current time plus the delta time in the second word of the read action data structure.

When the current time reaches the scheduled time, the read action is executed. The emulator reads the emulated address register and shifts the bits the appropriate amount to right-justify the target address. Next it checks this address against the low and high valid target addresses in the seventh and eighth words of the action data structure. If the address is not within this valid range, a message is outputted, and the program aborts. If the address is valid, the actual QM-1 address for the target word is calculated as:

```
QM-1 address = relocation constant +
                    target address * number of QM-1 words per target word
```

(The number of QM-1 words per target word is obtained from the first word of the action data structure, and the relocation constant is obtained from the fourth word). The program then reads the appropriate QM-1 address to get the new data. It then compares this new data, bit by bit, with the old data in the data register pointed to by word six of the action data structure. In each case, if the bit in the target word just read agrees with the bit in the data register, no action need be taken; however, if the bits are different, then the device in the network (as specified in the appropriate word in the action data structure) to which this bit feeds is enqueued on the stack. This procedure is repeated for each bit in this word, and then the entire procedure is repeated for all QM-1 words representing the one target word. Each word of the data register is then updated to represent the data just read from the target memory. It should be noted that for a read memory action, the devices into which the bits in the data register feed must be simple gates. Figure 10 is a diagram of the read memory action structure:



Read Memory Action Structure

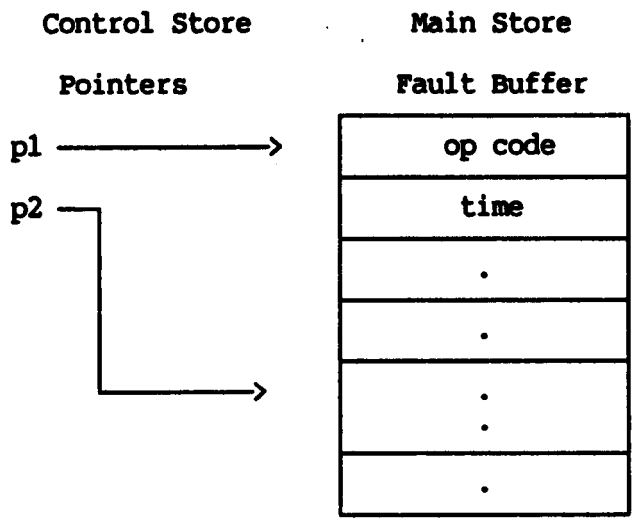
Figure 10

4.3.10.3 Operations Action

The "Operations Action" was created to allow the user to control, at run time, when and how certain functionally emulated "operations" are to be performed. See A-39 for the action data structure used for the operations action. Each valid operation has been assigned a particular operation code. To date, the valid operations codes are:

Code	Operation
2	stop run
3	stick gate at 0
4	stick gate at 1
5	lift gate fault
6	insert fault in ROM
7	lift fault from ROM

For each batch job, these operations are specified by the user in the Fault File. See Section 5.4.2.2.3 for a detailed discussion of the fault file. The "operations action" is the method used for implementing these valid operations at the properly scheduled times. The implementation works as follows: At run time, the fault file for the entire batch is read and converted to the "fault buffer" which is stored in the main store of the QM-1. For each op code in the fault file, the user has specified a time at which it is to be scheduled, and possibly other parameters, depending on the particular op code. In the control store of the QM-1, are maintained two pointers. The first(p1) points to the first word of the fault buffer and remains unchanged for the duration of the batch execution. The second pointer(p2) always points to the next entry in the fault list to be scheduled. For each run in the batch, the operations must be entered in the fault list in ascending time sequence. During initialization for each run, the emulator schedules the first operation for that run. When the emulator reaches the time period at which at least one operation has been scheduled, it executes all actions which have been scheduled for that time. It then adjusts pointer p2 appropriately and schedules the next operation. Since each run must have a "stop run" as its last operation, this is the manner in which multiple runs are carried out for each batch. Figure 11 shows the fault buffer format.



Fault Buffer Layout

Figure 11

Following are descriptions of the op codes which have been implemented to date.

4.3.10.3.1 Stop Run

When a "stop run" is executed, a switch is turned on which causes the main program to terminate processing for that run.

4.3.10.3.2 Stick Gate at 0/1

For the purposes of sticking and lifting stuck-at faults from gates, a dummy gate must be added by the user as the last gate in the network. See Section 5.2.3. When any gate, say gate X, is faulted, the program dynamically creates a temporary connection from the output of the dummy gate to the enable input of gate X, sets this line to "disabled", and simultaneously sets VDIS (see A-29) in the connector word to the value at which the gate is to be stuck. This causes the gate to be disabled, and its output equal to VDIS, in essence causing the gate to be stuck at the desired value, until a "lift gate fault" operation is scheduled for that gate.

4.3.10.3.3 Lift Gate Fault

In order to lift a gate fault, the connection that was established between the dummy device and device X when the gate was faulted, is removed, and the gate is thus enabled and its output will again reflect its inputs.

4.3.10.3.4 Insert Fault in ROM

The user specifies the number of the ROM, the address and the bit position to be faulted. The emulator merely complements the bit which is to be faulted.

4.3.10.3.5 Lift Fault from ROM

Again the user specifies the number of the ROM, the address and the bit position from which the fault is to be lifted. The emulator merely complements the bit, thus returning it to its correct value.

4.3.10.3.6 Stop Batch

This operation is unique in that it may not be specified by the user. The program automatically adds a "stop batch" code at the end of the fault buffer. When it is executed, a switch is set which causes the main program to terminate execution of the entire batch.

4.3.10.4 External Inputs Action

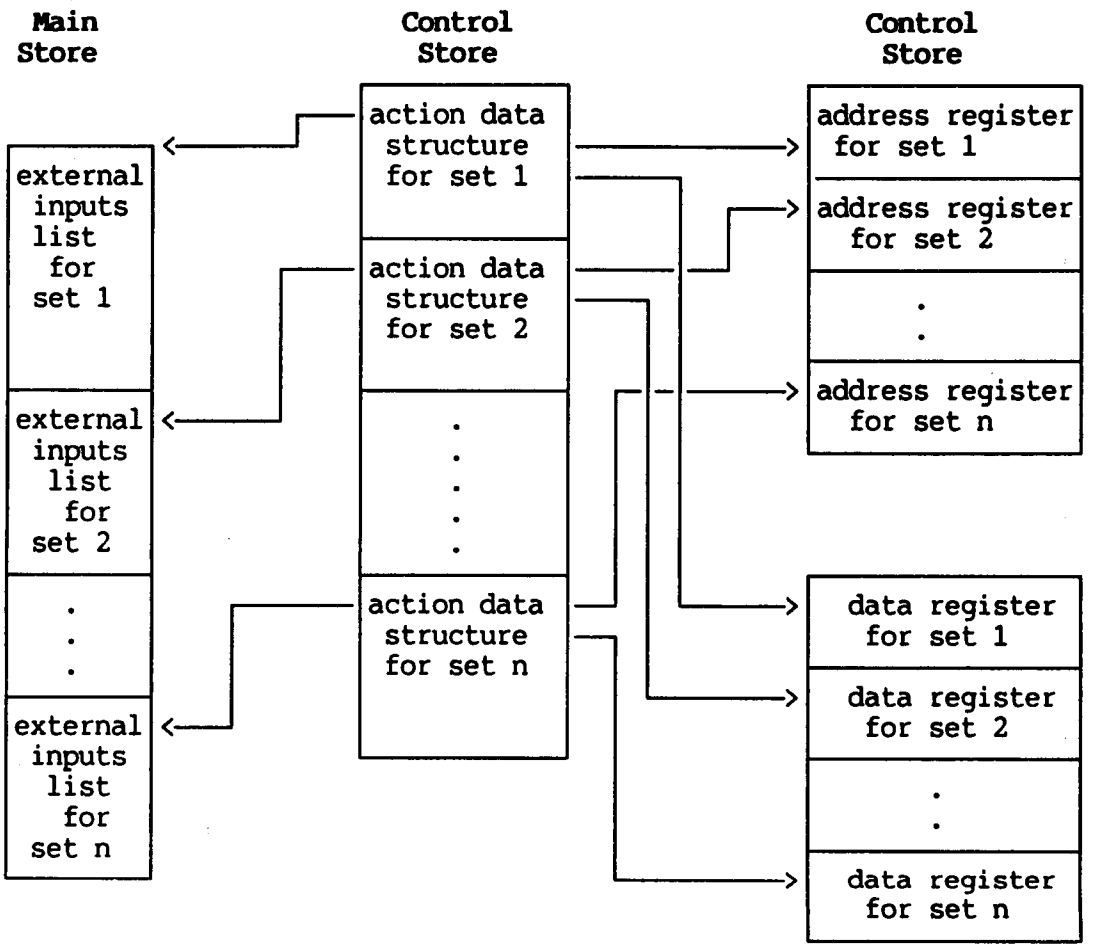
The emulator contains a feature which allows the user to request that inputs generated externally from the emulation be inserted into network devices internal to the emulation, at specified times. This feature is implemented with the external inputs action. For a given batch, a user may specify any number of external input sets, or he may request none. Each set corresponds to a particular set of devices in the network. Each set consists of a set of contiguous input data bits coming from an external source to be inserted into the set of specified devices in the internal logic network. The user decides how he wishes each set to be composed, i.e., for each external input set, he decides into which group of devices in the network and in which order the external inputs are to be fed. He also specifies at what times these input signals should be inserted. For each external input set that is specified, a separate external input file must be generated by the user before the emulation begins. This file contains a list of external input items. Each item consists of a time at which the data is to be inserted into the devices and the data (a contiguous set of 1's and 0's) which is to be inserted at the given time into the given devices. For a given batch, the same external input data is used for each run.

During batch initialization, the program reads the external input files and creates in the QM-1 main store a contiguous list of the data from these files, where this list consists of a sublist for each external input set. These sublists are re-used for each run, so that, for a given batch, the same external inputs are used for each run in the batch. The program also creates an external input action for each one of these sets. A pointer to this main store list is put into the appropriate action. The program also sets up a contiguous set of address and data registers for each external input set. For the purposes of setting up these structures, the user supplies to the initializer the address of the control store location for the first external inputs action data structure, the control store address for the first data register and the control store address for the first address register associated with the external inputs action.

The external inputs action is implemented in a manner similar to the read action, except that it is not triggered by a control line, but rather by the current time reaching the time specified in the external inputs file. Also, the external inputs action automatically increments the appropriate address register to point to the next data item and also schedules the next external inputs action for this set. The action data structure as well as the data registers and address registers needed are created by the program, and are basically transparent to the user, with the exception that he must specify where in the memory of the QM-1 these data structures will be placed. The first external inputs action for each external inputs set is scheduled at the

beginning of each run, and then immediately after any external input action is executed, the next one in time sequence for that run is scheduled.

For each set, the user supplies the name of the associated external inputs file, the number of bits in the data, and the names of the devices to which the data feeds. These devices must be single input gates or single-input tri-states, i.e., for a regular gate there must be no input to the gate other than this external one, and for a tri-state, there must be no input other than the enable/disable line. Figure 12 shows the structure for the external inputs action.



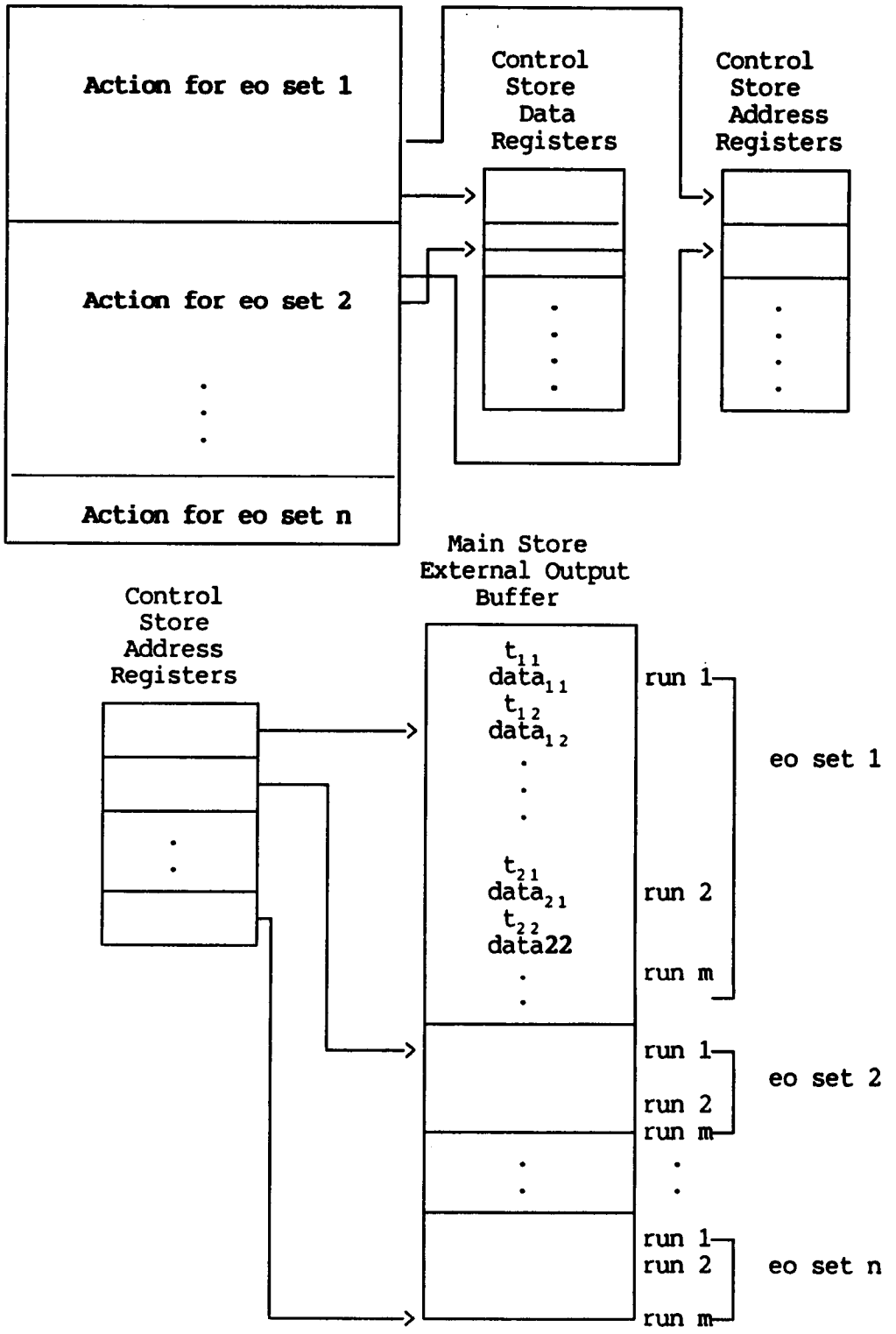
External Inputs Action Structure

Figure 12

4.3.10.5 External Outputs Action

The emulator contains a feature which allows the user to request that the output signals from specified devices in the network be recorded or "externally outputted" at specified times, and in specified groupings. This feature is implemented with the external outputs action. For a given batch, a user may request any number of external output sets, or he may request none. Each set corresponds to the output signals from a specified set of devices in the netlist. For each external output set that is requested, a separate external output file will be written at the completion of the batch. The user decides how he wishes each set to be composed, i.e., for each external output set, he sets up a group of external data register[s] into whose bits he feeds the signals he wishes to output in whatever order he wishes them to be arranged. He also specifies at what time periods these output signals should be captured. They can either be captured automatically at regular time intervals, or the capturing of data can be triggered by logic internal to the network. Thus when the batch is completed, each external output file will contain one record or entry for each time the external output action was triggered. Within this record will be the time at which the data was captured and the data itself. As an example, if the target hardware contains an accumulator whose contents the users wishes to track, he would feed the devices representing the bits of the accumulator into external register[s] and would use these external register[s] to create an external output file. By reading this external output file, he would see at specified times the contents of the accumulator. Note: for one external output set, all the bits in the external registers to be outputted must be contiguous and can occupy more than one QM-1 word; however the bits for one external output set (the data register[s]) do not need to be contiguous with the bits for a different external output set.

The implementation for an external output set is done as follows: During batch initialization, the program reads all the data necessary to create an external outputs action for each external output set. The emulator sets up the necessary action data structure for each external output set requested. The appropriate pointers are put into the appropriate action. The program also sets up a contiguous set of address registers, one for each external outputs set. The program automatically maintains the address registers, but it is the user's responsibility to maintain the data register for each external output set. The initializer reads the control store location for the first external outputs action, and the control store address for the first address register. For each set, the emulator reads the number of bits in the data, the name of the output file to be produced, the maximum number of data items in the buffer, the control address of the associated data register, the reschedule flag, the start time and the delta time. The external output generated can be triggered by a control bit or by automatic rescheduling, depending on how the reschedule flag is set. If the reschedule flag is 0, the scheduling is done by the internal logic using a control line(handling this in the same manner as a memory action). If the reschedule flag is 1, the program automatically schedules the action beginning at the designated start time, and automatically reschedules it from the start time to the end of the run in increments of delta t. Each time an external output is triggered for that set, either at regular time intervals or by the internal logic, the external output action is executed which saves the requested data in the QM-1 main store buffer. At the end of the batch, the entire memory buffer is written to disk file(s). Figure 13 shows a diagram of the external outputs action structure.



Note: eo = external output; assume n external output sets and m runs in batch
External Output Control Store Action Data Structures

External Outputs Action Structure
Figure 13

4.4 Algorithms

4.4.1 Initialization Algorithm

In the netlist, the user must specify the initial output value for anywhere from one to all devices. The algorithm works as follows: For each device, a record is kept of all input lines and the values on those input lines, as well as any initial user-defined output value for the device. In addition, a separate list is kept of all devices for which all input lines to that device have defined values. This list is then processed one device at a time. For each device whose input lines are all defined, its output value is calculated. If the predefined output value, if any, does not agree with the calculated value, then the user is notified, and the calculated value is used. In the case of a flip-flop, if the preset line is active, the output is set to one, whereas if the clear line is active, the output is set to zero. If neither preset nor clear is active, the output is set to the user's predefined output value if any is present. Next, the fanout from this device is examined, and the input lines to all devices to which it fans out are set accordingly. As these input lines are being set, the destination device is examined to see if after this input line is set, whether all of its inputs are then defined. If not, the program proceeds to the next destination device. If so, the program calculates an output value consistent with the input values, and then this destination device is added to the list of defined devices. Simultaneously, a check is made to see whether the predefined value, if any, agrees with the calculated value. If not, the user is notified, but the output value is set to the calculated value. This procedure continues for each device on the "defined" list, and hopefully the "defined" list grows as the procedure continues. After the program has processed the last device on the "defined" list, the initialization procedure has been completed. If at this time, all devices have defined output values, the initialization is considered successful; however, if not all the devices are on the "defined" list, the user is notified. He may choose to proceed with the emulation, but it would be a better idea to correct the netlist and do the initialization again before attempting an emulation.

4.4.2 Functional Emulation Algorithm

The functional algorithm schedules actions, executes actions at the times for which they have been scheduled, and implements the faulting of gates and memories. One iteration of the functional algorithm proceeds as follows:

Actions are scheduled as follows:
If the master action control switch is not on, no actions are to be scheduled. If the master action control switch is on, then each action control record is examined in turn. For each bit in an action control record which is on, the appropriate action (whose address is found in the corresponding word in the action control buffer) is scheduled.

Actions are then executed as follows:
The first event in the event list is examined. If its time is less than or equal to the current time, then all the actions to which it is linked, are executed, and that event is removed from the event list. If the time of the first event is greater than the current time, no actions are executed (because events are linked in order by ascending time).

Faulting of gates is carried out:
If there are any gates to be faulted or from which faults are to be lifted during this time period, the "faulter" gate is enqueued with connections to all gates which are to be faulted or from which faults are to be lifted. This enqueueing/dequeueing of the "faulter" gate insures that the fault will be inserted/lifted in the next time step.

4.4.3 Gate-level Algorithm

The gate-level algorithm examines only those devices whose output values changed during the previous period, and using this information, calculates which devices change output value during the current time period. Initially, there must be at least one device on the c stack. A device on the stack is one whose output has changed during the previous time period. One iteration consists of processing each device on the c stack and simultaneously building the cbar stack.

Processing of one device from the c stack proceeds as follows:

The device is removed from the top of the stack. It is then checked to see whether its output value has changed an even or odd number of times during the previous period. If the output value changed an even number of times, then this device need not be processed at all. If, however, it changed an odd number of times, then processing continues. Processing of a given source device from the stack consists of processing all internal connections from this device and then processing all external connections from this device.

The Internal Connections are processed as follows:
Each device into which this device feeds (destination device) is examined. The processing algorithm for the destination device depends on the type of internal connection:

If Connection is to a gate or tri-state(but not the enable input):
The count (see Section 4.4.3.1) of the destination header is appropriately updated. Next the current count and the initial count (before the updating took place), are examined. If neither is zero, then no more processing of this destination device is necessary; however, if either one is zero, then this destination device must be processed further. First the internal value is complemented. Next a check is made to see whether the gate is enabled. If not, no further processing is needed. If the gate is enabled, processing proceeds: the internal value is copied to the external value. Next a check is made to see whether this destination device is already on the cbar stack (as a result of its output value having changed because of a different source device which was already processed from the c stack). If it is on the stack, then all that is done is to update its header item which indicates whether it has changed an even or odd number of times. If it is not already on the cbar stack, then it is enqueued on that stack.

If Connection is to a flip-flop or to the enable line of a tri-state:
The processing carried out depends on the type of connection. In the case of a flip-flop, first the particular input in the header is

complemented, whether it be P, C, T, L, J, K, or D (J and K). The rest of the processing is particular to the type of connection. Again, the destination device is examined to see whether or not it should be enqueued on the cbar stack.

The External Connections are processed as follows:

The new output value from the source device is copied into all bits in external registers into which this device feeds. It should be noted that any time the high order bit in the master action control register is turned on (it is in an external register and is turned on if any device feeding it goes high), then the next time the functional algorithm is executed, some action(s) will be scheduled. The particular actions scheduled will be those corresponding to the one bits in the action control register(s).

Once this item from the c stack has been processed, the processing of the next source device from the c stack proceeds. This looping continues until the c stack is empty and the cbar stack represents the new stack. This consists of one iteration of the gate-level algorithm.

4.4.3.1 Description of Device "Count"

For each regular gate and tri-state device, a count is maintained within the header record. The purpose of this count is to enable the program to know (without explicitly calculating the output value as a function of the input values) when the output value of a simple gate has changed. The "count" for each device is initialized as shown in Figure 14.

Type of Gate	Initial Value of "count"
AND	$N_0 - M$
NAND	$N_0 - M$
OR	N_0
NOR	N_0
NOT	0
XOR	$N_0 - n$
NXOR	$N_0 - n$

M = total number of input lines to this device
N_0 = number of input lines that are high initially
n = number of input lines high which result in high output(XOR,NXOR only)

"Count" Initialization

Figure 14

Each gate is restricted to not more than 31(decimal) inputs.

Once the emulation has begun, the count is maintained as follows: Each time an input line transitions from zero to one, the count is incremented by one. Each time an input line transitions from one to zero, the count is decremented by one. Any time the count transitions into or out of zero, the output value of the device is complemented.

5. User's Guide

5.1 Installation of Programs

5.1.1 Installation of Emulator on Vax (Using Vax/VMS):

Note: This installation is necessary even if all production runs will be done on the QM-1, because the initialization and file transfers must be done from the Vax.

Notation Used:

user represents the name of the user's root directory (without the brackets). For example, if the user's root directory is [Smith], then in this document, **user** represents Smith.

Underlined items are those which the user types.

Installation Steps:

A tape has been created using the Vms Utility Backup. This tape has ID "bbemul" and Save Set Name "diagem.bck". This tape contains the following hierarchy of directories:

[bb.dem]

1.	[bb.dem.emulator]	source programs and command files for compiling and linking emulator
2.	[bb.dem.run]	command files for running emulator
3.	[bb.dem.transfers]	
	[bb.dem.transfers.qm1vax]	programs and command files for transfers from QM-1 to Vax
	[bb.dem.transfers.vaxqm1]	programs and command files for transfers from Vax to QM-1
4.	[bb.dem.templates]	Templates for data files
5.	[bb.dem.targets]	
	[bb.dem.targets.counter]	all data files for 3-bit counter circuit
	[bb.dem.targets.toy]	all data files for toy computer circuit
	[bb.dem.targets.test]	all data files for RTI test circuit
	[bb.dem.targets.comm]	all data files for RTI communicator interstage circuit

In all cases it is necessary to restore 1. and 2. If one wishes to do transfers, one must restore 3. If one wishes to use templates, one must restore 4, and if one wishes to use sample target circuits, one must restore any or all of the subdirectories of 5.

Assume the tape has been physically mounted on msa0:
Use the following commands to restore all directories and subdirectories from the tape:

```
$Mount/foreign msa0:

$Backup/verify msa0:diagem.bck/save/select=[bb.dem...]
 [user.dem...]          (restore from tape)

$set default [user.dem.emulator]

$@compandlinkemu                    (compile and link programs)
```

Example of Installation:

Assumptions: Name of user root directory is [Smith]

```
$Backup/verify msa0:diagem.bck/save/select=[bb.dem...]
 [smith.dem...]
                                        (Restore programs from tape)
$set default [smith.dem.emulator]
$@compandlinkemu                    (Compile and link programs)
```

To Make Modifications to Existing Programs

To make changes to existing initialization Program:

```
$set default [user.dem.emulator]
Edit appropriate Fortran module(s) in [user.dem.emulator]
Do Fortran compiles of appropriate module(s)
$@initlink      (links initialization programs)
```

To add new module(s) to existing initialization Program:

```
$set default [user.dem.emulator]
Create new Fortran modules in [user.dem.emulator] and compile
Add new module name(s) to init.opt file in [user.dem.emulator]
$@initlink      (links initialization programs)
```

To make changes to existing emulation Program:

```
$set default [user.dem.emulator]
Edit appropriate Fortran module(s) in [user.dem.emulator]
Do Fortran compiles of appropriate module(s)
$@emullink      (links emulation programs)
```

To add new module(s) to existing emulation Program:

$set default [user.dem.emulator]
Create new Fortran modules in [user.dem.emulator] and compile
Add new module name(s) to emul.opt file in [user.dem.emulator]
$@emullink (links emulation programs)

5.1.2 Installation of Emulator on QM-1

Note: This installation is not necessary if all production runs are to be done on the Vax. In order to proceed, one needs at the minimum a working knowledge of the Nova and Easy Operating Systems on the QM-1.

Notation Used:

Underlined characters are those which the user types into the QM-1 Operating System.

<CR> represents Carriage Return.
<ESC> represents "escape"
<Z> represents "control" key and Z key pressed simultaneously

The Diagnostic Emulation System Tape was created in Airlab at Langley Research Center using the DISK-SAVE function of the EASY operating system. The tape contains users 6 and 8 in that order. User 6 contains the diagnostic emulation programs, and user 8 contains the Vax-to-QM1 and the QM1-to-Vax transfer programs. Note that the tape files can be restored to users other than 6/8 by specifying the desired users in the USER-FORMAT,USER= command and in the DIRECTORY SEARCH command.

5.1.2.1 Restore Emulation System From Tape to Disk:

Mount User Disk (it is assumed for this document that the disk is mounted on drive 0, but it could be mounted on any drive)

Mount Emulation System Tape (it is assumed for this document that the tape is mounted on drive 0, but it could be mounted on any drive)

Press Master Clear, Start

???LDEASY

SET DATE AND TIME

!!DATE,XX/XX/XX

!!TIME,XX/XX/XX

```
!!@

!!DEADSTART

!!EASY-SPACE,BS=26,TPS=347777

!!<CR>



!!RESTORE-DISK,PASSWORD=HELP,MTUNIT=0,MTFILE=0,DSKUNIT=0
!!USER-FORMAT,USER=6
MOUNT TAPE ON DESIRED UNIT
HIT ANY KEY TO CONT., ESCAPE THROUGH 'ESC' KEY
(ANY KEY)
  SV-RES HEADER
  DATE=XX/XX/XX
  TIME=YY:YY:YY
  USER MODE
  ALL OF USER 6

TO ACTIVATE HIT RETURN
<CR>
HIT RETURN TO UNLOAD
<ESC>

(user 6 has now been restored from tape to disk)

!!USER-FORMAT,USER=8
MOUNT TAPE ON DESIRED UNIT
HIT ANY KEY TO CONT., ESCAPE THROUGH 'ESC' KEY
(ANY KEY)
  SV-RES HEADER
  DATE=XX/XX/XX
  TIME=YY:YY:YY
  USER MODE
  ALL OF USER 8

TO ACTIVATE HIT RETURN
<CR>
HIT RETURN TO UNLOAD

(user 8 has now been restored from tape to disk)

<CR>
<Z>
```

5.1.2.2 Compile & Link Easy Programs: Vax<-->QM-1 Transfers

```
!!DIRectory,Search 1st=06,2nd=,08
!!EXEC BBEX1          (compile Easy programs)
!!BIND.               (link Easy programs)
```

```
!!INCLUDE BBTEMP1
!!INCLUDE BBTEMP2
!!INCLUDE BBTEMP3
!!INCLUDE BBTEMP4
!!INCLUDE BBTEMP7
!!INCLUDE BBTEMP9
!!INCLUDE BBTEMP12
!!INCLUDE BBTEMP41
!!WRITE BBTEMP61
!!<Z>
!!BIND
!!INCLUDE BBTEMP55
!!INCLUDE BBTEMP61
!!INCLUDE BBTEMP56
!!INCLUDE BBTEMP57
!!INCLUDE BBTEMP58
!!WRITE BBVAXQM1
!!<Z>
!!BIND
!!INCLUDE BBTEMP9
!!INCLUDE BBTEMP7
!!INCLUDE ACWRTR:B
!!INCLUDE BBTEMP54
!!INCLUDE BBTEMP49
!!INCLUDE BBTEMP46
!!INCLUDE BBTEMP47
!!WRITE BBTEMP31
!!<Z>
!!BIND
!!INCLUDE BBTEMP66
!!INCLUDE BBTEMP67
!!INCLUDE BBTEMP68
!!INCLUDE BBTEMP41
!!INCLUDE BBSCR4
!!INCLUDE BBTEMP31
!!WRITE BBTEMP71
!!<Z>
!!BIND
!!INCLUDE BBMVMAIN:B
!!INCLUDE BBMVSEND:B
!!WRITE BBTEMP72
!!<Z>
!!BIND
!!INCLUDE BBTEMP71
!!INCLUDE BBTEMP72
!!WRITE BBMV
!!<Z>
```

5.1.2.3 Generation of program to write External Outputs to Disk

```
!!SIMPLQ SM WXMDISK:S WXMDISK:B $
```

5.1.2.4 Generation of Microcode Driver

```
PRESS Master Clear, Start
???LDNOV
!USER 6
!EX /BBECOMPILE
```

5.1.2.5 Generation of Nanocode Emulator

```
PRESS Master Clear, Start
???LDNOV
!USER 6
!LD*NASPC
!.S=1
!.S=2 INPT=/BBEMP1V1:S BIN=/BBBNBIN
!NP INPT=/BBEMP1V1:S BIN=/BBEMP1V1
!NP INPT=/BBEMP2V1:S BIN=/BBEMP2V1
!NP INPT=/BBEMP3V1:S BIN=/BBEMP3V1
!NP INPT=/MSNANON:S BIN=/MSNANO:B
!MAP INPT=/BBEMP1V1 BIN=W2
!MAP INPT=/BBEMP2V1 BIN=W3
!MAP INPT=/BBEMP3V1 BIN=W\3
!MAP INPT=/MSNANO:B BIN=/MSNANO:M
```

5.2 Data Preparation

5.2.1 Suggested QM-1 Template

In preparing to emulate a system, one of the preliminary steps for the user is to manually lay out the QM-1 memory to accomodate the various data structures. This step must be done whether or not the entire emulation will be done on the Vax or part on the Vax and part on the QM-1. The reason for this is that when the Vax emulator was written,it was assumed that the "production" runs would always be done on the QM-1 and that only "debugging" runs would be done on the Vax. Thus the Vax initializer always sets up the data as if it were to be run on the QM-1.

The main store of the QM-1 must be used for the target memories, the fault buffer, the external input list, and the external output buffer. All other data structures, including the netlist and external registers, are stored in control store. A suggested layout for the QM-1 memories, which should accomodate most emulations, is shown in Figure 15 (note that control store locations 0 through 1777 cannot be used by the user.

CONTROL STORE

location (octal)	contents
0	reserved for nanocode implementation
1777	
2000	free space and events(3 words per event)
2377	
2400	memory actions (reads,writes)
2717	
2720	stop action(3 words)
2722	
2725	Stop action(3+n words where n=no. of memories)
.	
.	
.	
3777	
	Externals

4000	
4001	time
.	
.	
4427	master action control register
4430	action control bits
4431	pointers to actions
.	.
.	.
4474	
4475	more action control bits
4476	pointers to actions
.	.
.	.
4541	
.	
.	
.	
4777	
5000	external output address registers(1 word each)
5177	
5200	external inputs address registers(1 word each)
5277	
5300	external inputs data registers(size of each is determined by no. bits given in *eopts.dat)
5477	
5500	external outputs actions (8 words each)
5737	
5740	external inputs actions(each action is 10+n words where n is the no. of bits given in *eopts.dat)
7277	
.	
.	
.	
20000	netlist in binary form (hardware description matrix)

MAIN STORE

location (octal)	contents
0000	target memories
7777	
10000	externals (if any)
10777	
11000	fault buffer (The total size is determined by the fault list) no. words / code / function / data used 2 / 2 / stop run: / op,t 3 / 3 / stick gate 0 / op,t,gate no. 3 / 4 / stick gate 1 / op,t,gate no. 3 / 5 / lift gate fault / op,t,gate no. 5 / 6 / insert fault in rom / op,t,mem id,word id,bit id 5 / 7 / lift fault from rom / op,t,mem id,word id,bit id
12777	
	external inputs list (there is one ei list for each ei set. the size of each list is n*(m+1) where n is the no. of times an external input is inserted m is the no. of 18-bit words required to hold the no. of specified bits for this set) (the external inputs list is stored automatically by the program at the next 100_8 word boundary following the fault buffer)
	external output buffer (there is one buffer for each eo set the size of each buffer is n*(m+1) where n is the no. of times an external output is written m is the no. of 18-bits words required to hold the no. of specified bits for this set) (the external outputs buffer is stored automatically by the program at the next 100_8 word boundary following the external inputs list)

QM-1 Memory Template

Figure 15

5.2.2 Setup of Functional Memories

In most cases, target memories will be implemented at a functional level. Outlined below are the steps the user must take to set up for this functional emulation. These memories may be any combination of ROMS and RAMS. There are two types of actions associated with memories, namely read memory and write memory. Each ROM should have at least one read associated with it, and each ram should have at least one read and one write action associated with it. In order to implement a given memory, it is the user's responsibility to do the following (see Section 5.4.1.2):

1. In the netlist, there must be a single device of any kind whose output line controls when the read/write takes place. The appropriate action takes place only when this line transitions from low to high. The output of this device must feed the master bit in the master action control register, and it must also feed to a unique bit in a "control bit" word in the action control block.

2. The address of the action to be performed when the control line goes high, together with a delta time to be added to the current time for scheduling, must be placed in the appropriate words of the action control block, in the memories files.

3. For each read and write action, a separate data register(s) and an address register must be set up as externals in control store. The address register must be fed from the appropriate devices in the network for both reads and writes. The data register for a read has no explicit connections to it in the netlist. The identification numbers of the devices to which the memory data will be fed when the read is triggered must be designated within the read action. In the case of a write, the data register must have explicit connections from some devices in the netlist. For a RAM, the read and write actions must have different data registers. A given memory may have more than one read and/or write action associated with it. See A-36 and A-37 for read and write layouts.

4. It is the user's choice as to whether the address in the 18-bit address register is to be right or left-justified. This choice is determined by the bit positions in the address register into which the appropriate devices feed. The value of item W in *iopts.dat depends on the user's choice. Item W is the value by which the address in the address register must be divided to right-justify it in the 18-bit word. For example, if the user chooses to let the address be right-justified in the address register, then W=1; if the address is left-justified, and is represented by 6 bits, then W=2**12 or 4096.

5. The read and write action data structures must be provided by the user in the *mems.dat file.

6. The address of the action control block must be given in item D5, the address of the master action register must be given in item D7, and the number of memory control records must be given in item D6 of *iopts.dat. The number of memory control records is the number of 18-bit words needed to hold all control bits for the entire emulation.

7. The number of memories must be given in item V of *iopts.dat.

8. The initial contents of the target memories must be given in *mems.dat. These memories are implemented in the main store of the QM-1, hence these entries will begin with "M" in column 1.

   The contents of each word of the target memory may use one or more QM-1 18-bit words, depending on the number of bits in each target word. For a particular memory, let n represent the number of 18-bit words necessary to hold one target word. The user may decide whether the target word will be left-justified or right-justified over these n words. When the read action(s) for this memory are generated, it should be noted that word 9 of the action corresponds to bit 17 of the first of the n QM-1-words, word 10 corresponds to bit 16, etc.. Thus, if the memory contents are left-justified, word 9 contains the device identifier of the device into which the most significant bit of the data feeds, etc.; however, if the data is right-justified in the memory, an appropriate amount of zeros would appear in words 9 ff. to correspond to the leftmost data bits that are not used. In addition, item D in word 1 of the read/write action is affected by whether the data is right or left justified(see the description of the read/write action data structure).

9. If the number of memories is greater than zero, the relocation constant(s) for the memories must be given in items V1-Vn of *iopts.dat. The relocation constant is the amount by which the contents of the memory will be offset from absolute location 0 in the QM-1 main store. When the user lays out the QM-1 memory, he must determine at which QM-1 absolute location (for example, x) that each ROM or RAM will begin. Then he has a choice of two ways in which he can present the initial data for the target memories, in the memories file. Using the first method, he will specify a relocation constant of 0, and in the memories file, he will specify that the first word of memory begins in location x, etc. Using this method, if the actual memory begins at target location 0, he must manually add x to every location for this memory that he specifies in the memories file; however, if he wishes the program to do the relocation, then he would use the second method. In this case he decides into which absolute QM-1 location (say x) that the memory will begin; he gives x as the relocation constant in items V1-Vn of *iopts.dat, and in the memories file, he gives the contents beginning in location 0, and the program automatically adds x to each location.

10. The data registers associated with read actions must be initialized in the *mems.dat file to values consistent with the output values of the devices to which the data register feeds.

5.2.3 Setup of Faults

If one wishes to fault gates, it is necessary to include two extra dummy gates at the end of each netlist. A template for these is shown in Figure 16. It is also necessary to enter the name of the dummy faulting device (in this case ZZZFAULTER) in *iopts.dat, item E. See Section 5.4.1.2.3.2. The names of the two devices is arbitrary, but they must be the last devices in the netlist, and the names in the netlist must be in ascending order. A file containing this template is on directory [bb.dem.templates]. (see Section 5.1.1)

```
!               template for the two standard faulting devices
!               allows for 30 gate faults per time step.(can be increased)
!               These two devices should be included at the end of the netlist.

>ZZZFAULTER      1  CLASS=  1 TYPE=   1 VALUE=   0 NICON= 30 NECON=  1
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  8
 ZZZFAULTER     11  CSMSF=  0 REGNO= 12 BITNO=   0 REVER=  0
>ZZZZDUMMY       1  CLASS=  1 TYPE=   1 VALUE=   0 NICON=  1 NECON=  1
 ZZZZDUMMY      10  ZNAME=    ZZZZDUMMY            REVER=  0 CONNT=  0
 ZZZZDUMMY      11  CSMSF=  0 REGNO= 13 BITNO=   0 REVER=  0
```

Sample Template for Faulting Device

Figure 16

5.2.4 Setup of External Inputs

The user may create zero or more sets of external inputs for a given emulation batch. See Section 4.3.10.4 for a description of external inputs. It should be noted that the same external input files will be reused for each run in the batch. It is the user's responsibility to:

1. Set items in *iopts.dat (see Section 5.4.1.2.3):
   Set items X, Z, and AA to appropriate values.

2. Set items in *eopts.dat (see Section 5.4.2.2.2):
   Set item AA, which is the total number of external input files. Set items AA1, AA2, and AA3 for each external input set. If item AA is zero, then items AA1, AA2, and AA3 are omitted. For each external input set, the user must create an External Input File. He can use any valid VMS file name for this file. Each external input file must have a unique name. This user-chosen name is specified in item AA1. Item AA2 specifies the number of bits to be supplied to the netlist from this set. Item AA3 lists the devices into which the bits feed. Note that the external input sets can be listed in any order. For each external input set, the devices must be listed in the order corresponding to the data bits, where the first device listed corresponds to the most significant data bit.

3. Create External Input Files (see Section 5.4.2.2.4 for details):
   This step is omitted if item AA in *eopts.dat is zero.

5.2.5 Setup for Producing External Outputs

For a given batch, there may be zero or more external output files created. See Section 4.3.10.5 for a discussion of external outputs. An output file will be written for each external output set at the completion of each batch.

If the user has specified that the number of external output sets is one or more, then it is his responsibility to do the following:

1. In *iopts.dat, set items BB, and DD.
2. In *eopts.dat, set item BB.
3. In *eopts.dat, if BB is at least one, then set items BB1 through BB5.
4. In the netlist, create external register(s) to act as the external output data registers. The appropriate logic devices must feed into these registers.
5. For any external output set for which automatic rescheduling is not used:

   In the netlist the appropriate device must be fed into the master action control bit and into a bit in some "control bits" word (see Sections 4.3.7 and 4.3.8). This device is the one whose transition

from low to high will trigger the scheduling of the external output action.

Set the address of the external outputs action(s) together with a delta time to be added to the current time for rescheduling in the appropriate words of the action control block in the memories file. It should be noted in determining the action addresses that the first external outputs action is stored at the address specified by the user in item BB of *iopts.dat, and that each external outputs action following the first is displaced from the previous one by 10(octal).

5.3 Program Modifications

5.3.1 Implementation of User-Defined Action

Vax Version

In order to implement a new action, one must do the following:

1. Select an action code for use with this action. Each action must have a unique code. The codes 1 through 30 (decimal) are reserved for use by the emulator. The codes 50 through 55 are reserved for use by the University of Illinois. All other codes up to and including 127 (decimal) may be used.
2. Create the action data structure for this action according to the layout in A-35 and include it in *mems.dat.
3. Write a new Fortran subroutine module to perform the action. See the already existing action modules, ACT2, ACT3, ACT6, ACT7, or ACT8 to see how this is done. Compile the new Fortran action subroutine. Modify the Fortran Module EX1ACT (see A-9) to include a branch to the new action. Compile EX1ACT. Modify the emulation link file emu.opt to include linking of the new subroutine.
   $@emulink (see Section 5.1.1)
4. Include an external connection from some device in the netlist to the master action control bit and another to some bit in the action control buffer, in order to trigger the action. A transition from low to high on the output line of this device will then trigger the action.

QM-1 Version

All of the above would be done. One would modify the corresponding microcoded routine "Ex1act" and would create and assemble a new action routine written in the Multi language.

5.3.2 Instructions for Increasing Array Sizes

Vax Version

If one wishes to modify the array sizes for the components of the system, one should follow the steps below:

1. Modify the dimension parameter(s) in the module "emuparam.for". Listed below are the parameters which specify the array sizes. The comment to the right of each parameter explains what that parameter represents.

```
C EMUPARAM.FOR
c                          dimension parameters

      parameter (yndevi = 6000)        !maximum number of devices
      parameter (ynconn = 14000)       !maximum number of internal connections
      parameter (ynstac = 500)         !maximum no. items on stack at one time
      parameter (yncomm = 100000)      !maximum no. bytes for device comments
      parameter (ynmems = 30)          !maximum no. target memories
      parameter (ynei   = 20)          !maximum no. external input sets
      parameter (yneo   = 20)          !maximum no. external output sets
      parameter (ynupc  = 15)          !maximum no. user print choices
      parameter (ynstat = 500)         !maximum no. state information devices
      parameter (ynchid = 1000)        !max no. devices to change at one time
      parameter (pcslow = 0)           !control store low address
      parameter (pcsup  = 20000)       !control store high address
      parameter (pmslow = 0)           !main store low address
      parameter (pmsup  = 120000)      !main store high address
      parameter (plslow = 0)           !local store low address
      parameter (plsup  = 31)          !local store high address
```

2. Recompile and link all programs (initialization programs and emulation programs) as described in Section 5.1.1.

5.4 Running the System

It is assumed for the purposes of describing these files that the user is familiar with Fortran Formats. All of the formats listed here are in the Fortran language.

For all initialization and emulation runs executed on the Vax, all file names must be valid Vms file names. For a particular target hardware, all input and output files should be on the same subdirectory and must begin with the same user-defined prefix. The suffixes are predetermined and listed in A-7 and A-8. For the purposes of this document, the prefix is always denoted with "*". For example, assume the user specifies "counter" as his prefix. Then, in this document, *mems.dat would represent "countermems.dat".

5.4.1 Initialization of Target Hardware on Vax

5.4.1.1 General

Before a given network can be emulated, it must be initialized. The initialization process is one in which the inputs are a description of the netlist in DENF format, the initial contents of the target memories, the initialization run-time options, and the device descriptions to appear on the

emulation stack outputs; the principle output is a complete description of the netlist with initial output values defined, and a complete representation of the initial memories, both in the binary form required by the emulator.

Four input files are required for running the initialization program. Two output files are always produced, and another four output files are sometimes produced, depending upon the options the user has requested in *iopts.dat.

Input Files

Required Files

*net.dat	The target network description(netlist)in DENF format. See Section 5.4.1.2.1.
*mems.dat	The initial values to be resident in the host memory before the emulation begins. See Section 5.4.1.2.2.
*iopts.dat	The run-time initialization parameters. See Section 5.4.1.2.3.
*comm.dat	The comments or descriptions to appear alongside device names when they appear on the stack output. See Section 5.4.1.2.4.

Output Files

Mandatory Output Files

*sav.dat	Initialized System State File: Binary netlist and memories to be used as inputs to the Vax emulator.
*iout.dat	Text output which varies according to the options that the user has requested in *iopts.dat.

Optional Output Files

*mat.dat	Netlist in a form to be used by the QM-1 for emulation. This file is only produced if item O is turned on in *iopts.dat.
*extrn.dat	Initial contents of control store external registers in a form to be used by the QM-1 for emulation. This file is only produced if item O is turned on in *iopts.dat.
*alph.dat	A list in alphanumeric order by device name of all the devices in the netlist. Included with each device is the device name, device index number, device type, device class, and initial output value of the device, in the format (1X,A20,1X,I4,1X,2A10,1X,I1). This file is only produced if item R is turned on in *iopts.dat.

*nam.dat — A list in alphanumeric order by device name of all the devices in the netlist. Included in each record is the device number in decimal followed by the device number in octal followed by the device name in the format (1X,I4,1X,O6,1X,A20). This file is produced to aid the user in creating a meaningful *comm.dat.

Following is a detailed description of each file:

5.4.1.2 Input Files

5.4.1.2.1 Netlist File

The network description completely defines the target network of gate-level logic and the interconnections among the devices, all in the DENF format. Normally, this file would be generated from some preprocessor or translator. Each device and its fanout is described by a group of records, referred to as the "device definition". A device is defined as a regular gate (AND, NAND,OR, NOR, NOT, XOR, NXOR), a tri-state device or a flip-flop. Each record contains the name of the device being defined. **The device definitions must be in ascending order by device name, according to the ascii collating sequence.** Within each device definition, the records must be in the order as specified below. The group of records necessary to specify a particular device definition varies according the device type. The maximum number of devices at present is 6000. The maximum number of internal connections is 14000. The maximum number of external connections is 6000. These numbers can be increased should it become necessary by changing dimensions in Fortran programs (see Section 5.3.2). There are four different types of Device Definition corresponding to:

1. regular gate other than XOR or NXOR
   - record 1: <device description>
   - record 2 and following: <internal connections>
     <external connections>

2. XOR or NXOR gate
   - record 1: <device description>
   - record 2: <xor specification>
   - record 3 and following: <internal connections>
     <external connections>

3. tri-state device
   - record 1: <device description>
   - record 2: <tri-state specification>
   - record 3 and following: <internal connections>
     <external connections>

4. flip-flop
   - record 1: <device description>
   - record 2: <flip-flop specification 1>
   - record 3: <flip-flop specification 2>
   - record 4 and following: <internal connections>
     <external connections>

where:
<internal connections> := one or more <internal connection>
<external connections> := zero or more <external connection>

Record Types:

<device description> format: (1x,a20,i3,8x,i3,4(7x,i3))
contents: NAME,SEQUEN,CLASS,TYPE,VALUE,NICON,NECON

<xor specification> format : (1X,A20,I3,8X,I3)
contents: NAME,SEQUEN,XORNN

<tri-state specification> format : (1X,A20,I3,8X,I3)
contents: NAME,SEQUEN,VDIS

<flip-flop specification 1> format : (1X,A20,I3,8X,O3)
contents: NAME,SEQUEN,FFVALUE

<flip-flop specification 2> format :(1X,A20,I3,8X,I3,7X,I3)
contents: NAME,SEQUEN,R,U

<internal connection> format : (1x,a20,i3,12x,a20,6x,i3,7x,i3)
contents: NAME,SEQUEN,ZNAME,REVER,CONNT

<external connection> format : (1X,A20,I3,1X,4(7X,I3))
contents: NAME,SEQUEN,CSMSF,REGNO,BITNO,REVER

The symbols used for the "contents" above are as follows:

NAME Device Name: the unique device name. The name must be at least one but not more than 20 printable ascii characters. While the name may contain any valid ascii printable characters, it must be remembered that in the netlist, these names must be in ascending order according to the ascii collating sequence. It should also be noted that the two dummy devices used for faulting must be the last two devices in the netlist, and so must be named appropriately. Also note that upper case and/or lower case letters may be used in the name, but at any other point in an input file in which the name appears, the case must match, character by character, the case used in the netlist name.

SEQUEN Sequence Number-this number is not used by the emulator. It is included in order to keep the records for one device in order during any sort by device name. For purposes of the emulator, it may be left blank.

CLASS Device Class
1=Gate
2=Flip-flop
3=Tri-state

TYPE Gate Type

0=flip-flop
1=AND
2=NAND
3=OR
4=NOR
5=NOT
6=XOR
7=NXOR

VALUE Initial value on output line:
User-initialization (item B in Init. Options File = 0):
Value must be 0 or 1
Program will use the user-assigned value if there is no inconsistency. If there is an inconsistency, the user will be notified, and the calculated value will prevail.
Program-initialization (item B in Init. Options File = 1):
Value must be 9
Program will attempt to calculate the value. If it cannot, it will notify the user.)

NICON Number of internal connections (This number represents the number of devices in the network to which the output of this device fans out). This number must be greater than zero.

NECON Number of external connections (This number represents the number of "external" or "pseudo" connections to which the output of this device goes. These connections are not part of the internal network, but are used to hold output values of devices for the functional emulation.) This number can be zero or greater.

For XOR and NXOR gates only:

XORNN The exact number of input lines which must be high in order for the output line to be high. If the number of high input lines is less than or greater than this number, the output line will be low.

For Flip-Flops Only:

FFVAL Initial Flip-flop values(PCTLJK)
This is an octal value which represents the initial value for bit positions 0-5 of the flip-flop header word. Bit 5 is the value on the P connection, bit 4 on the C connection,etc.
See A-25, Device header Layout (Flip-Flop).

For the purposes of initialization:
P and C: the default is negative logic, i.e., the value of 1 is benign, and 0 is active on the preset and clear lines. This can be overridden at the time of initialization with a flag in the options file.
T: the default is negative edge-triggered. This can be overridden at the time of initialization by using a connection type of -3 rather than 3.

Note: During initialization, any value initialized by the user may be overridden if the program discovers an inconsistency.

R (Used for RS flip-flops)
Indeterminate Flag 1. See A-28 and A-29, Device Connector to Flip-Flop, word 1, bit 9. Also see Migneault[2]. The initial value should be 0 if this feature is not to be used.

U (Used for RS flip-flops) Indeterminate Flag 2. See A-28 and A-29, Device Connector to Flip-Flop, word 1, bit 8. Also see Migneault[2]. The initial value should be 0 if this feature is not to be used.

For Tri-States only:

VDIS The value the output line of the tri-state is to assume when it is disabled. This value may be 0 or 1.

For Each Internal Connection:

ZNAME Name of the destination device, i.e., the name of a device to which this device fans out.

REVER Reversal flag (reversal meaning same as inversion)
0=no reversal entering the destination device
1=reversal entering the destination device

CONNT Connection type
0 = connection to a gate, or connection to a tri-state but not the enable line of the tri-state.
1=P connection to flip flop
2=C connection to flip flop
3=T connection to flip flop (downward edge-triggered)
-3=T connection to flip flop (upward edge-triggered)
4=L connection to flip flop
5=J connection to flip flop
6=K connection to flip flop
7=D[1] connection to flip flop
8=Enable line to tri-state

[1] "D" connection is one in which the K input is always the complement of the J input.

For each External Connection:

CSMSF Control Store/Main Store
0=Control Store, 1=Main Store

REGNO Register Number
The number of the external register. These are numbered beginning with Register 1. The number is decimal.

BITNO Bit Number
The number of the bit within the register. These are numbered from 0 to 17. The least significant bit is numbered 0, and the most significant bit is 17. This number is decimal.

REVER Reversal flag (or inversion flag)
0=no reversal entering the external register
1=reversal entering the external register

See Appendix C for a sample of a network description file.

5.4.1.2.2 Memories File

Purpose

The memories file specifies the values which are to be resident in the control store and the main store of the QM-1 at the beginning of the emulation, but which are not generated by the initialization program and must therefore be supplied by the user. For most emulations, these initial memory values are:

In control store:
memory read and/or write actions
user-generated actions
action control block
initial memory data register contents

In main store:
actual contents of target memories

Format

The memories file may contain seven different record types. They are as follows:

Type 1: column 1 contains "!"
meaning: Remainder of record contains comments which are not used by emulator

Type 2: columns 1-3 contain "ROM" (must be upper case)
meaning: Remainder of this record is blank. Records following this record are of type 6 or 7, and they represent the contents of a target ROM.

Type 3: columns 1-3 contain "RAM" (must be upper case)
meaning: Remainder of this record is blank. Records following this record are of type 6 or 7, and they represent the contents of a target RAM.

type 4: column 1 contains "C" or "c"
meaning: The remainder of the record contains octal values separated by commas. Each octal value can occupy up to six columns and can have leading blanks. The first octal value represents the beginning control store location into which the remaining values will be consecutively placed.

type 5: column 1 contains "D" or "d"

meaning: the remainder of the record contains octal values separated by commas. Each octal value can occupy up to six columns and can have leading blanks. The first octal value represents the beginning control store location into which the remaining values will be consecutively placed. The only difference between type 5 and type 4 is that for type 5 the values to be placed into control store represent device index numbers. This type need only be used when preparing data for QM-1 emulation runs.

type 6: column 1 contains "M" or "m"
meaning: the remainder of the record contains octal values separated by commas. Each octal value can occupy up to six columns and can have leading blanks. The first octal value represents the beginning main store location into which the remaining values will be consecutively placed.

type 7: column 1 is blank
meaning: the remainder of the record contains octal values separated by commas. Each octal value can occupy up to six columns and can have leading blanks. The first octal value in this case is not a location but the value to be placed into the next consecutive location after the last location of the previous record. The remaining values will be consecutively placed.

Note regarding the order of the records in the memories file:

All records describing the contents of ROMS and/or RAMS should be at the end of the memories file. All the records for one ROM or RAM must be contiguous. Obviously, all records of type 7 are order-dependent, since the location comes from the previous record. All other records besides those just mentioned are independent of order.

Note regarding the relocation of ROMS and RAMS:

Immediately preceding the first record for each target memory, a record must be inserted which consists of the word "RAM" or "ROM" in columns 1-3. It is used to identify the beginning of each new target memory for purposes of relocating it in the QM-1 memory and for identifying the memory identification for memory fault insertions.

The user may, if he desires, request that the initializer relocate one or more ROMS and/or RAMS in the QM-1 memory. If he chooses to do this, he supplies the relocation constant to the program, and this relocation constant is automatically added to the location in the record. (see Section 5.4.1.2.3, items V1...Vn).

Note regarding in-record comments:

For any type listed above, if "!" appears in any column, then all columns after the "!" will be treated as comments.

5.4.1.2.2.1 Sample Memories File

Following are examples of records within a memories file, *mems.dat.!

```
!                       Memories File
! This file contains all values to be placed
!   during initialization into the QM-1
!   memory, both control store and main store.
!
! action #6, operations action
c002725,030000 !code=6
c002726,000000 !ptr to next action
c002727,000000 !reschedule time
c002730,002400 !action address table-bank 1
c002731,002440 ! memory bank 2
c002732,002532 ! memory bank 3
c002733,002600 ! memory bank 4
C002405,4406     !CS location of data register
C002406,0,37     !valid addresses for this memory
 13,35,4763
!
!           MEMORY #1, ROM8.32.1, SEQUENCE CONTROL ROM
ROM                                                          (type 2) (see note 2)
M005000,306,307,310,311,264,264,264,264,266,312              MAIN STORE CONTENTS(type 6)

M005020,153,154,155,156,0,0,0,0,264,265,266                  (see note 3)
!
!                       MEMORY #2, ROM8.512.1, MICROCODE START ADDRESS ROM
ROM
M   0,  41,  16,  45,  14,  27,  17,  43,  15
M  10,  50,  40,  44,  34,  51,  46,  42,  35
M  20, 262, 272, 276, 102, 264, 273, 277, 103
!
!
!                        MEMORY #3, Ram16.64.1
!
RAM                                                          (type 3) (see note 4)
M  0000,110001,   440,
   0400,  1401,  4001,  1401,  4002,  1401,  4003
   0406,  1401,  4004,  1401,  4005,  1401,  4006
   0420,  1401,  4011,  1401,  4012,  1401,  4013
   0432,  1401,  4016,177400, 11000, 77416, 41017
```

COMMENTS (type 1) (not used by emulator)

CONTROL STORE (type 4) CONTENTS (see note 1)

(type 7)

MAIN STORE CONTENTS (types 6,7) (see note 5)

Note 1: The first record causes 30000_8 to be placed into control store location 2725_8. The second record causes 0 to be placed in control store location 2726_8, etc.. The ninth record places 0 into location 2406_8 and 37_8 into location 2407_8. The tenth record places 13_8 into location 2410_8, 35_8 into location 2411_8 and 4763_8 into location 2412_8. Note that all of the text to the right of the "!" is merely comments.

Note 2: The records that follow (until the next type 2 or type 3 record) contain the contents for the next target ROM.

Note 3: These two records contain contents for a target ROM. The first record places the value 306_8 into main store location 5000_8, 307_8 into location 5001,..., and as the last value for this record, places 312_8 into location 5011_8. The second record causes 153_8 to be placed into main store location 5020_8, 154_8 into location 5021_8, etc..., and finally 266_8 into location 5032_8.

Note 4: The records that follow (until the next type 2 or type 3 record) contain the contents for the next target RAM.

Note 5: These records contain contents for a target RAM. The first record places the value 110001_8 into main store location 0, and the value 440_8 into location 1. The second record places the value 400_8 into location 2, 1401_8 into location 3,..., and finally 4003_8 into location 10_8, etc..

5.4.1.2.3 Initialization Run-Time Options File

The initialization options file *iopts.dat is an input file which contains parameters and user selections for the initialization run. The initialization options file is usually prepared manually with an editor. Listed below is a sample initialization options file. It can be used as a template for the user's preparation of his own file. Following the sample is a description of each of the records in an initialization options file. In order to facilitate the discussion, the individual records in the sample have been labeled on the far right with capital letters. Some of the items in this file are no longer used or are used only for debugging purposes. For that reason, only the items currently used that are relevant to the general user are so labeled. These capital letters are merely for documentation purposes. The general user need only be concerned with the labeled items. For all other items, they can be left at the values in the sample, but they must be present in the file in the order indicated. Items I through S control whether various option outputs will be produced. In each case, unless otherwise noted, the particular output will be produced as part of the *iout.dat file. If this is not the case, the name of the file produced is noted in the item description.

5.4.1.2.3.1 Sample Initialization Run-Time Options File

Following is a sample of an Initialization Options file, *iopts.dat. The label for each record is a capital letter appearing to the far right of the record. It is for documentation purposes only, and does not actually appear in the record.

The output options 1-50 (items I through S) are switches which control which outputs are produced. These options have no effect whatsoever on the initialization but are merely for the user's benefit if he wishes to see the initialization process in more detail (especially when the network has initialization problems). In each case, a 1 means the option is turned on and the corresponding output will be produced, while 0 means it will not. Unless otherwise noted, the particular output will be produced as part of the *iout.dat file. If this is not the case, the name of the file produced is noted. The records not labeled with a capital letter are not used (i.e., the values are "don't care", but must still be present).

Note: In each record, text following the "!" is comments

Abbreviations:
In what follows, the abbreviation ei is used for external inputs, and the abbreviation eo is used for external outputs. The abbreviation cs is used for control store, and ms is used for main store. An asterisk (*) preceeding the name of a file represents the user-supplied prefix.

Sample *iopts.dat File

```
Any Title       ! Title for hardware being emulated                              A
1               ! initialization flag                0=user, 1=computer          B
0               ! user val for no-input devices                                  C
1               ! preset-clear convention flag       0:1=benign 1:1=active       D
020000          ! cs address for netlist                                         D1
004000          ! cs address for external registers                              D2
000000          ! ms address for external registers                              D3
004001          ! cs address for time                                            D4
004430          ! cs address of action control block                             D5
000001          ! number of "control bit" words                                  D6
004427          ! cs address of master action control register                   D7
002720          ! cs address of stop action                                      D8
002000          ! cs address of free space list                                  D9
50              ! number of free space records                                   D10
003760          ! main store address of fault block                              D11
002725          ! cs address of operations action data structure                 D12
ZZZFAULTER         ! name of faulting device                                     E
004400,004407   ! cs lo,hi address for dump                                      F
000000,000000   ! ms   "        "       "                                       G
000000,000010   ! ls   "        "       "                                       H
1               !*1 initial device headers **first output option**               I
1               ! 2 not used
0               ! 3 not used
1               ! 4 not used
1               ! 5 not used
1               ! 6 not used
1               ! 7 not used
1               !*8  control store memory dump                                   K
0               !*9  main store memory dump                                      L
0               !*10 local store memory dump                                     M
1               !*11 not used
1               ! 12 not used
0               ! 13 not used
1               ! 14 netlist in QM-1 format                                      O
0               ! 15 not used
1               !*16 connections list                                            P
0               ! 17 not used
0               ! 18 not used
0               ! 19 not used
1               !*20 devices with undefined output values                        Q
1               ! 21 devices with defined output values                         QQ
0               ! 22 not used
1               ! 23 not used
```

```
1               ! 24 memory dumps at stop time                                  NN
0               ! 25 alphabetized list of devices                               R
0               ! 26 not used
0               ! 27 device name list                                           S
0               ! 28 not used
0               ! 29 not used
0               ! 30 not used
1               ! 31 not used
1               ! 32 not used
1               ! 31 not used
1               ! 32 not used
1               ! 33 not used
1               ! 34 not used
0               ! 35 not used
0               ! 36 not used
0               ! 37 not used
1               ! 38 not used
0               ! 39 not used
0               ! 40 not used
0               ! 41 not used
0               ! 42 not used
0               ! 43 cs initialized external registers in QM-1 format
T
0               ! 44 not used
0               ! 45 not used
0               ! 46 not used
0               ! 47 not used
0               ! 48 not used
0               ! 49 not used
0               ! 50 not used
1,1,1             !not used
2                 !not used
X1              ! stack items                                                   U
*****           !
*****           !
*****           !
*****           !
3               ! no. of target memories                                        V
001000          ! relocation constant for memory 1                              V1
005500          ! relocation constant for memory 2                              V2
006300          ! relocation constant for memory n                              Vn
1               ! divisor to right justify target address                       W
005000          ! address in cs for first ei action                             X
000200          ! not used
004500          ! address in cs for first ei address reg                        Z
004540          ! address in cs for first ei data reg                           AA
000160          ! address in cs for first eo action---beg of eo                 BB
000200          ! not used
000150          ! address in cs for first eo address reg-end eo                 DD
015000          ! highest loc in cs to go to save file                          EE
020000          ! highest loc in ms to go to save file                          FF
000031          ! highest loc in ls to go to save file                          GG
```

5.4.1.2.3.2 Record Descriptions for Init. Run-Time Options File

Formats:
In each item, the Fortran format follows in parentheses after the name of the item.

Descriptions:

A Title (10a4) : Any title which describes the target hardware.
This title will appear at the beginning of the initialization output file *iout.dat and at the beginning of the emulation output file *eout.dat, preceded by "TARGET MACHINE:".

B Initialization Flag (I1)
If set to 0, user must supply output values for all devices in *net.dat.

If set to 1, user will supply at least one device output value, but may supply more. Program will attempt to calculate any values not supplied.

C User-supplied value for devices with no inputs (I1)
For each device which does not have any inputs, and no predefined value, this value will be used as its output value.

D Preset-clear convention flag (I1)
If set to 0, then a value of 1 on either the P or C input to any flip-flop will be treated as benign, i.e., will not cause the output value to be set or cleared respectively.
If set to 1, then a value of 1 on either the P or C input to any flip-flop will be treated as active, i.e., will cause the output value to be set or cleared respectively.

D1 Control Store Address for Netlist (O6)
The starting address in control store for the binary netlist(O6)

D2 Control Store Address for External Registers (O6)
The starting address in control store for external registers. (the first register is referred to as register number 1)

D3 Main Store Address for External Registers (O6)
The starting address in main store for external registers.
(not generally used)

D4 Control Store Address for Time (O6)
The address of some cs external at which the current time will be stored at each clock cycle, to be available for output if so desired. It can be dumped in any format by using items Z1 and Z2 in the *eopts file.

D5 Control Store Address of Action Control Block (O6)
The starting address in control store of the action control block.

D6 Number of "Control Bit" Words in Action Control Block (*)
The number of QM-1 18-bit words needed to hold all the control bits for the emulation.

D7 Control Store Address of Master Action Control Register (O6)

The control store address of the master action control register which contains the master bit which goes high any time at least one control line goes high.

D8 Control Store Address of Stop Action (O6)
The starting address of the stop action in control store.

D9 Control Store Address of Free Space List (O6)
The starting address in control store of the free space and event lists.

D10 Number of free space records (*)
The maximum number of free space records to provide space for. Each record takes three QM-1 words.

D11 Address of fault block in main store (O6)
The starting address in main store where the fault list will reside.

D12 Control Store Address of op action data structure (O6)
The control store address of the op action (action 6).

E Name of Faulter Device (A20)
The name of the device to be used for faulting gates.

F QM-1 Control Store Dump Locations (O6,1X,O6)
The starting address(in octal) of the block of control store to be dumped, followed by the ending address(in octal) of the block of control store to be dumped. The dump takes place after initialization only if print option 8(item K) is on.

G QM-1 Main Store Dump Locations (O6,1X,O6)
Same as F, but for Main Store and print option 9(item L).

H QM-1 Local Store Dump Locations (O6,1X,O6)
Same as G, but for Local Store and print option 10(item M).

I Initial Device Headers (I1)
If this option is turned on, the initialization output will contain a list(in octal) of the initial header word for each device in the netlist along with its QM-1 control store address.

K Control Store Dump Option (I1)
If this option is on, the control store range specified in item F will be dumped after initialization.

L Main Store Dump Option (I1)
If this option is on, the main store range specified in item F will be dumped after initialization.

M Local Store Dump Option (I1)
If this option is on, the local store range specified in item F will be dumped after initialization.

O Netlist in QM-1 format (I1)

If this option is on, a file (*mat.dat) will be produced which can be sent to the QM-1 for emulation on that machine. This file contains the entire netlist in a matrix form to be used by the QM-1. This option would only be on if one is intending to do the emulation runs on the QM-1.

P Connections List (I1)
A complete list of the network, showing for each device, all the devices which feed into it, the device types, and the initialized output value for each device.

Q Devices with undefined output values (I1)
A list of all devices for which the program was not able to determine the output value. The user should analyze the netlist, correct the problem, and rerun the initialization. This should usually be turned on to see if there are any problems in the netlist description.

QQ Devices with Defined Output Values (I1)
A list of all devices for which the program was able to determine the output value.

R Alphabetic List of Devices (I1)
If this option is on, a file(*alph.dat) will be produced. This file contains the name,number, type, class, and initial output value of each device in the netlist, in alphanumeric order by name.

S Device Name List (I1)
If this option is on, a file(*nam.dat) will be produced. This file is to be used as a template for use with some editor to manually produce the file *comm.dat. It would normally only be necessary to produce this file once, and then to edit it as changes are made to the netlist. It is not necessary to produce this file at all if comments are not desired in the stack dumps produced during the emulation runs. See Section 5.4.1.2.4.

T Control Store Initialized External Registers in QM-1 Format (I1)
If this option is on, a file (*extrn.dat) will be produced which contains the control store initialized external registers in the format necessary to be sent to the QM-1 for emulation on that machine. This option would only be on if one is intending to do the emulation runs on the QM-1.

U Stack Item(s) (A20)
The names of all devices on the initial stack. There will be one record for each device on the initial stack. There must be at least one item in this list. The items can be in any order. The same initial stack will be used for each run in the batch.

V Number of target memories (*)
If this value is 0, then items V1 through Vn are not to be included. If this value is greater than zero, say n, then V1 through Vn must be included.

V1..Vn Memory Relocation Constants for memories 1 through n (O6)

The number of locations by which each target memory will be relocated in the QM-1. Each target memory is stored in the main store of the QM-1, and thus must have some relocation constant to map it into the memory of the QM-1. The contents of each target memory as specified in *mems.dat may either be manually relocated in the QM-1's memory by the user, or may be relocated by the program. If the user does the relocation, enter a 000000 here. If the program is to do the relocation, enter the relocation constant here. If the user does the relocation, then all memory addresses for main store in the *mems.dat file are absolute QM-1 addresses, i.e., the actual target address plus the QM-1 relocation constant. If the program is to do the relocation, then each main store address in the *mems.dat file is the target memory address. Regardless of whether or not the relocation constant is zero or greater than zero, the actual address register must contain the target address, i.e., the relocation constant is not included in the address in the address register.

The target memories must all appear at the end of the *mems.dat file in an order corresponding to the order of the relocation constants appearing here. The memories are identified starting with 1, and are numbered consecutively in the order in which they appear in *mems.dat. The relocation constants in *iopts.dat must correspond in number and order to the memory contents in *mems.dat.

W Divisor to right justify emulated address register (*)
The power of 2 by which the 18-bit emulated address register must be divided to right justify the address in an 18-bit word.

X The QM-1 control store address at which the first computer-generated ei action is to be stored (o6).
The rest will be stored contiguously.

Z The QM-1 control store address where the first ei address register will be stored (O6).
The rest will be stored contiguously.

AA The QM-1 control store address where the first ei data register will be stored (O6).
The rest will be stored contiguously.

BB The QM-1 control store address at which the first computer-generated eo action is to be stored (O6).
The rest will be stored contiguously.

DD The QM-1 control store address where the first eo address register will be stored (O6)
The rest will be stored contiguously.

EE The highest location in the control store save area[1] (to be saved in *save.dat) by the initialization program (O6)
The save area begins with location zero.

FF The highest location in the main store save area[1] (to be saved in *save.dat) by the initialization program (O6)

The save area begins with location zero.

GG The highest location in the local store save area[1] (to be saved in *save.dat) by the initialization program. The save area begins with location zero. (O6)

[1] Save Areas — At the beginning of an emulation batch, initialized data structures are read from disk and stored in the QM-1 memory. Certain of these data structures may change during a run, but some do not. Thus the ones which change are kept in the low portions of control store and main store so they can readily be restored before each run. The low word of a save area is always 0, but the highest word is specified by the user in *iopts.dat.

5.4.1.2.4 Device Comments File

The comments file specifies for each device listed in the file the descriptive comment that will appear to the right of the device name each time that device appears on the stack in the emulation text output file, during the emulation. The stack is only printed when the user requests it. This usually means that he is analyzing the results of the emulation at each clock step, or he is trying to follow the behavior of some device during the emulation. At such times, it has been found that with large number of devices in the netlist, seeing the device name on the stack is not sufficient to remind the user of the function of the device, and hence these descriptive comments are provided. Thus, when the device name appears on the stack, the comment reminds the user of the function of the device.

Because of the potentially large number of devices in a netlist, an optional aid was provided to enable the user to produce this comments file. When he runs the initialization the first time, he can provide an empty *comm.dat file, but turn on item S in the *iopts.dat file. By doing this a skeleton file will be produced containing all the device names in alphabetical order, and then all the user need do is edit the file, adding the descriptive comments. For any devices for which he does not desire any comments, he can merely delete that device record from the file or just leave the record with no comment. Then he must run the initialization again, this time using the newly edited file as the *comm.dat file.The file produced by turning on item S has the following format and contents:

```
Fortran Format for each Record: (1X,I4,1X,O6,1X,A20)
Contents of each Record:        Device Number in decimal
                                Device Number in octal
                                Device Name
```

The format required for the *comm.dat file is:

```
Fortran Format for each Record: (13X, A20,1X,A70)
Contents of each Record:        Device Name
                                Device Description or Comments
```

It can be seen that the device number in decimal and octal are not needed but that the user can leave them and merely add the description.

On the other hand, if the user desires, he can create the *comm.dat file independently of the emulator using whatever method he desires, merely using the format (13X,A20,1X,A70).

Following is an example of a *comm.dat file that was initially created by turning on item S and then editing the output file::

5.4.1.2.4.1 Sample Device Comments File

```
  2      2 FFA'CPUIC06          FOV single bit overflow flop
  3      3 FFA'CPUIC13          IND indirect storage flop
  5      5 FFA'CPUIC28          A* flop - repeat counter
  6      6 FFA'CPUIC71          FLAG1
  7      7 FFA0CPUIC39          bit 12 T register - 9407 mem addr processor
  8     10 FFA0CPUIC40          bit 8  T register - 9407 mem addr processor
  9     11 FFA0CPUIC42          bit 4  T register - 9407 mem addr processor
 10     12 FFA0CPUIC43          bit 0  T register - 9407 mem addr processor
 23     27 FFACPUIC06           FOV* single bit overflow flop
 24     30 FFACPUIC13           IND* indirect storage flop
 25     31 FFACPUIC21           IR04 - instruction register
 26     32 FFACPUIC28           A flop - repeat counter
 27     33 FFACPUIC71           NOT USED
 28     34 FFB'CPUIC06          PFEIN interrupt enable flop
 29     35 FFB'CPUIC13          LINK used by micro program
 30     36 FFB'CPUIC21          IR05* - instruction register chip
 31     37 FFB'CPUIC28          B* flop - repeat counter
 32     40 FFB'CPUIC71          FLAG2
 33     41 FFB0CPUIC39          bit 12 P register - 9407 mem addr processor
 34     42 FFB0CPUIC40          bit 8  P register - 9407 mem addr processor
 35     43 FFB0CPUIC42          bit 4  P register - 9407 mem addr processor
 36     44 FFB0CPUIC43          bit 0  P register - 9407 mem addr processor
 49     61 FFBCPUIC06           PFEIN* interrupt enable flop
495    757 GA0BCPUIC29          A0* RAM latch output* - 2901
496    760 GA0BCPUIC32          A0* RAM latch output* - 2901
497    761 GA0BCPUIC35          A0* RAM latch output* - 2901
498    762 GA0BCPUIC38          A0* RAM latch output* - 2901
499    763 GA0CPUIC01           UMA0 - micro memory prom address
500    764 GA0CPUIC08           UMA0 - micro memory prom address
501    765 GA0CPUIC15           UMA0 - micro memory prom address
507    773 GA0CPUIC45           addr input Y08 - START ADDR PROM
508    774 GA0CPUIC52           UMA0 - micro memory prom address
509    775 GA0CPUIC59           UMA0 - micro memory prom address
510    776 GA0CPUIC66           UMA0 - micro memory prom address
511    777 GA0CPUIC70           addr input - sequence control PROM
512   1000 GA0LCPUIC29          A0 RAM latch output - 2901
513   1001 GA0LCPUIC32          A0 RAM latch output - 2901
514   1002 GA0LCPUIC35          A0 RAM latch output - 2901
515   1003 GA0LCPUIC38          A0 RAM latch output - 2901
3168  6140 TSY3CPUIC39          D14 output - 9407 mem addr processor
3169  6141 TSY3CPUIC40          D10 output - 9407 mem addr processor
3170  6142 TSY3CPUIC42          D06 output - 9407 mem addr processor
3171  6143 TSY3CPUIC43          D02 output - 9407 mem addr processor
```

```
3172    6144 TSY3CPUIC62          UMA9
3177    6151 TSY4CPUIC62          SPARE
3178    6152 ZDUMMYCLOCK
3179    6153 ZGTSQ1CPUIC30        DAT15 - output register and
3180    6154 ZGTSQ1CPUIC31        DAT07 - output register and
3196    6174 ZMEM1CON2
3197    6175 ZSELECTCPUIC         MICRO MEMORY READ
```

For example, using the above as the *comm.dat file: if the device FFA'CPUIC06 were on the stack, the comment "FOV single bit overflow flop" would be printed to the right of the device name. If the device ADUMMYINPUT were to appear on the stack, no comment would appear, since that device does not appear in this file. Also, if the device ZDUMMYCLOCK were to appear on the stack, no comment would follow, since it appears in this file, but with no comment.

5.4.1.3 Initialization Output Files

5.4.1.3.1 Initialized System State File

The initialization program initializes the entire netlist, the external registers, and the target memories and captures this initial state of the entire system in a single binary file *save.dat. This file then becomes an input to the emulator. This file must be created each time any part of the netlist, target memories, device comments, or values in *iopts.dat changes. Once the system state file has been created to the user's satisfaction, the initialization need not be run again. The system state file is transparent to the user, other than the fact that he should be aware of its existence so that he does not inadvertently delete it.

5.4.1.3.2 Initialization Text Output File

The contents of the initialization text output file(*iout.dat) created by the initialization program are almost completely under the control of the user. In the input file, *iopts.dat, he specifies what outputs he wishes to appear in this file.

Mandatory Outputs

The first line of the file is the run date and time. The second line begins with the text "TARGET MACHINE:" and is followed by the text which the user inserted in item A of *iopts.dat.

If there are any gates in the netlist which have no inputs, then the third line consists of the text:

"ASSIGNMENT OF 0/1 TO FOLLOWING GATES WITH NO INPUTS:", and is followed by a list of devices for which no inputs were defined by the user. The initializer thus assigned the output value(0 or 1 as specified in item C of *iopts.dat) to all of these devices. If there were no such devices, this output does not appear.

Optional Outputs

All other outputs are optional and are controlled by the user in *iopts.dat. The optional outputs are selected by the user in items I through T in *iopts.dat. See Appendix B for an example of an Initialization Text Output File.

5.4.1.3.3 Initialization Matrix File

If the user is planning to run an emulation on the QM-1, he must do an initialization run in which he turns on item 14. This will cause the *mat.dat file to be generated. This is a text file which contains the initialized netlist in a form which can be processed by the QM-1.

5.4.1.3.4 Initialization External Registers File

If the user is planning to run an emulation on the QM-1, he must do an initialization run in which he turns on item 43. This will cause the *extrn.dat file to be generated. This is a text file which contains the initialized external registers in a form which can be processed by the QM-1.

5.4.2 Emulation on Vax

5.4.2.1 General

A given network can be emulated only after it has been initialized. The inputs to the emulation process are : the Initial System State contained in a binary file produced by the initialization, the Fault List, the Runtime Options file, and the optional External Inputs. The Text Output file is always produced, and its contents depend on the options the user has selected. The principle output is the optional External Outputs File(s). Other optional outputs are the control store and main store files for the QM-1.

Input Files

Required Input Files

*save.dat	The initialized system state, including the initial contents of the target memories, produced in binary form by the initialization program.
*eopts.dat	The run-time emulation parameters. See Section 5.4.2.2.2.
*fault.dat	The fault list.

Optional Input Files

**.dat	External Input files, named by the user

Output Files

Mandatory Output Files

*eout.dat	Text output which varies according to the options that the user has requested in *eopts.dat.

Optional Output Files

**.dat	External Output files, named by the user
*qmcs.dat	Control Store Initial Contents, for QM-1
*qmms.dat	Main Store Initial Contents, for QM-1
*summ.dat	Timing Summary

** User specifies entire Vax Vms file name rather than just a prefix.

5.4.2.2 Emulation Input Files

5.4.2.2.1 Initialized System State File

The initialization program initializes the entire netlist, the external registers, and the target memories and captures this initial state of the entire system in a single binary file *save.dat. This file then becomes an input to the emulator. This file must be created each time any part of the netlist, target memories, device comments, or values in *iopts.dat changes. Once the system state file has been created to the user's satisfaction, the initialization need not be run again. The system state file is transparent to the user, other than the fact that he should be aware of its existence so that he does not inadvertently delete it.

5.4.2.2.2 Emulation Run-Time Options File

The emulation options file *eopts.dat is the input file which contains parameters and user selections for the emulation run. This file allows the user to vary the external inputs and external outputs for each run and also to vary what outputs he wishes to have produced for each run, without having to redefine the target machine, that is without having to rerun the initialization. The emulation options file is usually prepared manually with an editor. Listed below is a sample emulation options file. It can be used as a template for the user's preparation of his own file. Following the sample is a description of each of the records in an emulation options file. In order to facilitate the discussion, the individual records in the sample have been labeled on the far right with capital letters which are then referred to as record identifiers in the descriptions of the records. Some of the items in this file are no longer used or are used only for debugging purposes. For that reason, only the items currently used that are relevant to the general user are so labeled. These capital letters are merely for documentation purposes. The general user need only be concerned with the labeled items. The records not labeled with a capital letter are not used (i.e., the values are "don't care", but must still be present).

All outputs requested in this file, except for external outputs, are produced as part of the *eout.dat file. Item Y controls the time(s) at which items K through Z6 will be produced. The external outputs are generated in user-named files (see item BB1).

All items from A through Z7 merely control the outputs which are to be produced to enable the user to analyze how the emulation is proceeding. These items in no way affect the emulation, and it is normal to produce none of them once the emulation is working properly. On the other hand, items AA through AA3 are the specifications for the external inputs and do affect the emulation.

See Appendix D for samples of the actual outputs produced.

5.4.2.2.2.1 Sample Emulation Run-Time Options File

Following is an example of an Emulation Options File, *eopts.dat. The label for each record is a capital letter appearing to the far right of the record. It is for documentation purposes only, and does not actually appear in the record.

The output options 1-50 are switches which control which outputs are produced. These options have no affect whatsoever on the emulation, but are merely for the user's benefit if he wishes to see the emulation process in more detail (especially when the emulation is not working as expected). In each case, a 1 means the option is turned on and the corresponding output will be produced, while 0 means it will not.

Abbreviations:
In what follows, the abbreviation ei is used for external inputs, and the abbreviation eo is used for external outputs. The abbreviation cs is used for control store, and ms is used for main store. An asterisk (*) preceding the name of a file represents the user-supplied prefix.

Sample *eopts.dat File

```
Any Title                          !Title to describe the Batch          A
004000,004017      ! cs low ,high address for dump                       F
000100,000117      ! ms low, high address for dump                       G
000000,000010      ! ls low, high address for dump                       H
0                  ! 1 not used                  *** First Output Option***
0                  ! 2 not used
0                  ! 3 not used
1                  ! 4 not used
1                  ! 5 not used
1                  ! 6 not used
0                  ! 7 not used
1                  ! 8  control store memory dump                        K
0                  ! 9  main store memory dump                           L
0                  ! 10 local store memory dump                          M
0                  ! 11 stack dump in full mode                          N
0                  ! 12 not used
0                  ! 13 not used
0                  ! 14 not used
0                  ! 15 not used
0                  ! 16 not used
0                  ! 17 not used
0                  ! 18 not used
0                  ! 19 not used
0                  ! 20 not used
0                  ! 21 not used
0                  ! 22 not used
0                  ! 23 not used
0                  ! 24 Memory Dumps at Stop Time                        NN
0                  ! 25 not used
0                  ! 26 not used
0                  ! 27 not used
```

```
1                ! 28 time line                                              T
0                ! 29 stack size                                            U
0                ! 30 stack dump in abbreviated mode                        V
1                ! 31 insertion and lifting of gate faults                  W
1                ! 32 not used
1                ! 33 insertion and lifting of memory faults               DD
1                ! 34 partial fault list                                  DD1
0                ! 35 scheduling and insertion of external inputs          EE
0                ! 36 not used
0                ! 37 not used
1                ! 38 scheduling and generation of external outputs        FF
0                ! 39 not used
0                ! 40 run numbers                                          GG
0                ! 41 fault file for QM-1                                  HH
0                ! 42 memory dumps at action-scheduling times              II
0                ! 43 not used
0                ! 44 external input list and ei registers for QM-1
JJ
0                ! 45 external output registers  for QM-1
KK
0                ! 46 abbreviated run to produce QM-1 data only
LL
0                ! 47 not used
0                ! 48 not used
0                ! 49 not used
0                ! 50 not used                          ***Last Output Option***
1,180,1          ! start,stop,delta times, for outputs                       Y
180              ! stop time (not used)
DEVICET            ! Device Name(s) for Trace                                Z
*****              ! Sentinel for trace devices                             Z1
time=               0 004001 004013 ! User dump specifications              Z2
(1x,a7,1x,i5,1x,o6,4x,10(i1))       ! Format for user-specified dump        Z3
*****                               ! Sentinel for User dump selections     Z4
DEVICEH                             ! Devices to have state info dumped     Z5
*****                               ! Sentinel for state info.devices       Z6
1                                   ! Number of external input lists        AA
[bb.edata.toy.ei]toyei1.dat         !name of file containing list           AA1
8                                   !no. of bits in each data item          AA2
TSY2U66                             !name of fanout device                  AA3
TSY1U66                                       "                             AA3
TSY3U66                                       "                             AA3
TSY4U66                                       "                             AA3
TSY2U65                                       "                             AA3
TSY1U65                                       "                             AA3
TSY3U65                                       "                             AA3
TSY4U65                                       "                             AA3
1                                  !number of external output sets          BB
eofile1.dat                        !output file name for this eo set        BB1
4                                  !no. of bits in each output this set     BB2
100                                !maximum number of items in eo buffer    BB3
004440                             !cs data register address this set       BB4
1,25,1      !reschedule flag, start time, delta time for rescheduling       BB5
```

5.4.2.2.2.2 Record Descriptions for Emul. Run-Time Options File

Description of Records in *eopts.dat File

Formats:
In each item, the Fortran format follows in parentheses after the name of the item.

A Title (10A4) : Any descriptive title for the Batch.
This title will appear at the beginning of the emulation output file *eout.dat, following the title for the target machine and preceded by "BATCH:" (One should be sure to begin any comments beyond column 40).

F QM-1 Control Store Dump Locations (O6,1X,O6)
The starting address(in octal) of the block of control store to be dumped, followed by the ending address(in octal) of the block of control store to be dumped. The dump takes place only if print option 8 (item K) is on, and occurs at the times specified in item Y. It also takes place at termination time if option 24 (NN) is on.

G QM-1 Main Store Dump Locations (O6,1X,O6)
Same as F, but for Main Store and print option 9(item L).
It also takes place at termination time if option 24 (NN) is on.

H QM-1 Local Store Dump Locations (O6,1X,O6)
Same as G, but for Local Store and print option 10(item M).

K Control Store Dump Option (I1)
If this option is on, the control store range specified in item F will be dumped at times specified in item Y.

L Main Store Dump Option (I1)
If this option is on, the main store range specified in item G will be dumped at times specified in item Y.

M Local Store Dump Option (I1)
If this option is on, the local store range specified in item H will be dumped at times specified in item Y.

N Stack Dump in Full Mode (I1)
If this option is on, the selected stack items (either the entire stack or a trace stack) will be dumped in the full format mode (see Section 5.4.2.3.2)

NN Memory Dumps at Stop Time(I1)
If this option is on, a control store memory dump and main store dump

will take place at the stop time for each run.

T Time Line (I1)
If this option is on, the time line will be dumped as a single line by itself. This option would only be used if item N is not on, and one wishes to see the time line. See Section 5.4.2.3.2.

U Stack Size (I1)
If this option is on, the number of items in the stack will

be dumped, but not the stack itself.

V Stack Dump in Abbreviated Mode (I1)
If this option is on, the selected stack items (either the complete stack or a trace stack) will be dumped in the abbreviated format mode. See Section 5.4.2.3.2.

W Insertion and Lifting of Gate Faults (I1)
If this option is on, each time a gate fault is inserted or lifted, the relevant information, namely the time, the name of the device, and the particular action will be dumped.

DD Insertion and Lifting of Memory Faults (I1)
If this option is on, each time a memory fault is inserted or lifted, the relevant information will be dumped.

DD1 Partial Fault Buffer Dump
If this option is on, the first 100(octal) locations and the last 27(octal) locations of the fault buffer will be dumped.

EE Scheduling and Insertion of External Inputs (I1)
If this option is on, each time an external input is scheduled and/or inserted, the relevant information will be dumped.

FF Scheduling and Generation of External Outputs (I1)
If this option is on, each time an external output is scheduled/generated, the relevant information will be dumped.

GG Run Numbers (I1)
If this option is on, numbers are assigned sequentially, starting at 1, to the runs in a batch, and are dumped at the beginning of each run.

HH Fault File for QM-1 (I1)
If this option is on, a file *qmms.dat containing the fault list will be produced which can be sent to the QM-1 for emulation on that machine.

II Memory Dumps at Action Scheduling Times (I1)
If this option is on, a control store memory dump and main store memory dump will take place each time an action is scheduled.

JJ External Input Data for QM-1 (I1)
If this option is turned on, then the external input list is produced in file *qmms.dat for the QM-1, and the external input data and address registers are produced in file *qmcs.dat for the QM-1.

KK External Output Data for QM-1 (I1)
If this option is turned on, then the external output registers are produced in file *qmcs.dat for the QM-1.

LL Abbreviated Run for QM-1 File Generation (I1)

If this option is on with any of options HH, JJ, and KK turned on, then the program will produce the output files for the QM-1 and stop without doing any emulation. Thus this should be turned on if one wishes to do the emulation runs on the QM-1 but not on the Vax. On the other hand, one can turn on items HH, JJ, and KK, and LL and perform emulation on both the QM-1 and the Vax.

Y Dump Option Time Window (*)
The start time, stop time, and time interval (in units of stacks) at which all the selected outputs K through Z6 will be produced.

Z Names of Devices to be Traced (A20)
The names of all devices which are to be traced, i.e., dumped when they appear on the stack. See Section 5.4.2.3.2. If one or more devices appear in this item, then the full stack will not be dumped, but only the devices listed here (when they appear on the stack). The names can appear in any order. If no names appear here, and items N and V are off, no stack will be dumped; however, the sentinel (item Z1) must be present in any case. If any comments are to be present in the record, one should be sure to begin the comment beyond column 20. There will be one item Z record for each device to be traced.

Z1 Sentinel for Trace Devices (A20)
This record signals that no more trace device names follow. This record must always be present, whether or not there are any trace devices listed. This record must have an asterisk in each of columns 1 through 5.

Z2 User-Defined Dump Specifications Part 1 (A20,1X,I1,1X,O6,1X,O6)
Items F through M allow the user to dump portions of control store, main store, and/or local store at times specified in item Y. The advantage of using items F through M is that the program produces the dump in a fixed format about which the user need not be concerned. The disadvantages of using items F through M are that only one contiguous section of control store, one section of main store, and one section of local store can be dumped, and this is always done in a fixed format. In order to overcome these disadvantages, one can use items Z2 and Z3. These items allow the user to define what he would like to dump and in what format he would like to see this dump. It is possible to define up to 15 (maxupch) different Dump Specifications. Each dump specification consists of two records, namely items Z2 and Z3. Item Z2 specifies what is to be dumped, and item Z3 specifies in what format the data is to be dumped. Thus it is possible to dump up to 15 different contiguous portions of control store, main store, and/or local store in user-defined valid Fortran 77 formats. If the user does not wish to have any user-defined dump specifications, there should be no Z2 or Z3 records, but there must always be one Z4 record.

Record Z2 contains:
1. The literal characters or title to be printed preceding the dump
2. The memory-type flag (0=control store, 1=main store, 2=local store)

3. The starting location to be dumped
4. The ending location to be dumped

Z3 User-defined Dump Specifications Part 2 (A80)
The second record, Z3, contains the Fortran format statement (enclosed in parentheses) in which the data which was defined in part 1 is to be dumped.

Z4 Sentinel for User-defined Dump Specifications (A20)
This record signals that no more user-defined dump specifications follow. This record must always be present, whether or not there are any user-defined dump specifications. This record must have an asterisk in each of columns 1 through 5.

Z5 Names of Devices for which State Information will be Dumped
The header word for each device contains all the state information for that device. The user would use item Z5 if he wishes to examine the state(s) of one or more particular devices at specified times during the emulation. There should be one Z5 record for each device for which state information is to be produced. It should be noted that it is possible to have header information of a given device change without having the device appear on the stack (e.g., an enabling or disabling of a tri-state). This item may have no devices in it; however, the sentinel, item Z6 must always be present.

Z6 Sentinel for State Information Devices (A20)
This record signals that no more state information device names follow. This record must always be present, whether or not there are any state information devices listed. This record must have an asterisk in each of columns 1 through 5.

AA Number of External Input Sets (*)
If this value is zero, then items AA1 through AA3 are left out. If this value is not zero, then the group of items AA1 through AA3 must appear once for each external input set.

AA1 Name of File containing the external input list (A40)
The file named here contains the actual data to be inputted from external sources during the run. See Section 5.4.2.2.4. for a complete description of this file.

AA2 Number of Bits in each data item (*)
This is the number of bits that must be supplied in the ei file each time the data is to be inserted into the network. The maximum number of bits is 32.

AA3 Name of fanout device (A20)
There must be as many devices listed as the number of bits specified in AA2 above. Each device will receive as input the bit specified in the data. The first device named will receive the most significant bit, and the last device the least significant bit. There will be one device named on each AA3 record.

BB Number of External Output Sets (*)

If this value is zero, then items BB1 through BB5 are left out. If this value is not zero, then the group of items BB1 through BB5 must appear once for each external output set.

BB1 Name of File to receive the output data (A40)
The file named here will be written to at the completion of the batch run and will contain the time-tagged data for this external output set for all runs in the batch.
See Section 5.4.2.3.3 for a complete description of this file.

BB2 Number of Bits in each data item (*)
This is the number of bits that will be dumped to the external output file each time the data is requested. The first bit dumped is the leftmost bit at the address specified in BB4, and bits are dumped rightward and from ascending locations. The largest acceptable value for this field is 126(decimal).

BB3 Maximum number of items in the buffer (*)
This is the largest number of items this data set is expected to generate during the entire batch run. It is used for storage allocation.

BB4 Control Store Address of External Output Data Register (O6)
The address in control store of the first data register to be dumped for the external output set.

BB5 Reschedule Flag, Start Time, Delta Time for External Outputs (*)

Reschedule flag: if this value is zero, then the scheduling of this external output set is controlled by internal logic, i.e., when a specified devices goes high, the output is produced, but otherwise the output is not produced. If this value is one, then the emulator does automatic rescheduling of this external output, starting at the specified start time, and at intervals of the specified delta time, until the end of the run.

Start time: The first time at which this output is to be automatically scheduled, if reschedule flag =1 (otherwise not used).

Delta time: The time increment between automatic rescheduling, if reschedule flag=1 (otherwise not used).

5.4.2.2.3 Fault List File

5.4.2.2.3.1 Contents of the File

In the fault list file, *fault.dat, the user specifies all "operations" to be performed for the batch. A batch consists of one or more "runs" for the same target machine. A run begins at time 1 and continues until the stop time designated in the fault list for that run. The parameters which the user must supply depend upon the particular operation.

The time given is in units of the basic clock ticks or numbers of stacks of the emulator. For each run in the batch, any number of operations may be

specified. There will be a maximum number of operations that can be accomodated for the entire batch, and if this number is exceeded, the user will be notified. Within each run, the operations must be in ascending time order. Valid operations, their corresponding op codes used in the fault list, and the parameters required for each are listed below:

Op Code	Operation	Parameters Required
1	Stop Batch	
2	Stop Run	Time
3	Stick Gate at 0	Time, Gate Name
4	Stick Gate at 1	Time, Gate Name
5	Lift Gate Fault	Time, Gate Name
6	Insert Fault in ROM	Time, Memory Id, Word Id, Bit Position
7	Lift Fault from ROM	Time, Memory Id, Word Id, Bit Position

Valid Op Codes

Figure 17

Stop Run

The user specifies the time at which the run is to terminate. There must be one "stop run" operation as the last operation for each run. It is possible that the "stop run" may be the only operation for the run.

Stick Gate at 0/1

The user may apply faults to simple gates. The faults that are applied are "stuck at" faults. The user specifies the gate name, whether the gate is to be stuck at 0 or 1 (by the op code) , and at what time the gate is to be stuck. For a gate to be stuck at 0 means that the output line of the gate will remain at 0 no matter what the input values happen to be; when a gate is stuck at 1, the output line will remain at 1 no matter what the input values happen to be. The gate remains stuck until a "lift gate fault" is applied to the gate.

Only simple gates may be faulted (AND, NAND, OR, NOR, XOR, NXOR). If one wishes to fault a flip-flop, then the flip-flop could be modeled as a set of gates, or a dummy gate could be inserted whose input is the output of the flip-flop, and the dummy gate could be faulted. If one wishes to fault a tri-state, the same is true as for flip-flops.

When a user specifies that a gate is to be stuck at time T, the fault actually becomes effective at time T+1. If one wishes to have a gate stuck from the very beginning of a run (T=1), then the time given with the op code should be 0.

Lift Gate Fault

When one wishes to remove a fault from a gate, he supplies the gate name and the time at which the fault is to be lifted. The user should not request that a fault be lifted from a gate unless a fault has previously been inserted and not yet lifted. Again, when a user specifies that a fault be lifted at time T, the lifting of the fault will be effective at

time T+1. When the fault is lifted, the output line of the gate will then again accurately reflect the values on the input lines.

It is possible in a particular run at present, to assert up to 30 gate fault insertions and/or lifts at the same time. This maximum can be increased if necessary. See Section 5.2.3.

Insert Fault in ROM

In order to insert a fault into a ROM, the user must specify the time at which the fault is to be inserted, the identification number of the particular rom, the address of the word to be faulted and the bit position of the bit to be faulted. Faulting a bit in a ROM is equivalent to complementing the correct value.

Lift Fault from ROM

One may request that a fault which has been previously inserted into a ROM be removed. Removing the fault is equivalent to complementing the value currently in the specified bit position, or in other words, returning it to its original value. Note that if one tries to lift a fault which has not previously been inserted, then one has effectively inserted a fault, since the existing bit is merely complemented. When a user specifies that a ROM fault be inserted or lifted at time T, the operation is actually effective at time T.

Stop Batch

This operation is unique in that it may not be specified by the user. The emulation automatically adds a "stop batch" code at the end of the fault buffer. Its execution causes the entire batch job to be terminated. This operation is basically transparent to the user.

5.4.2.2.3.2 Structure of the File

The first record of the file is a title which will be printed in the output file. Following the title is a list of "operations" to be executed for run 1, followed by operations for run 2, etc. There is no limit on the number of operations for each run. The minimum number of operations per run is one. There must be one "stop run" operation as the last operation for each run. In this "stop run" operation the user specifies at what time the run is to terminate. Each run may thus have a different stop time. It is possible that the "stop run" may be the only operation for the run. Thus every fault file must have at least two records, namely the title record and at least one "stop run" operation. Operations for any particular run consist of a sequence of operations which must be in ascending order by time. The structure of the file is show below (assuming n runs in the batch):

File Structure

Title Record
Operations for Run 1

Operations for Run 2
.
.
.
Operations for Run n

Record Structures

The number of records required for each operation is dependent on the particular operation; however, record 1 for each operation has the same format. The record contents and formats are:

Title Record
Format: (A40)
Contents: The first record of the file contains a title which will be printed at the beginning of the output file *eout.dat preceded by "Operations :"

Operations for Each Run

Valid operations, their corresponding op codes used in the fault list, and the parameters required for each are listed below:

Op Code	Operation	Parameters Required
1	Stop Batch	
2	Stop Run	Time
3	Stick Gate at 0	Time, Gate Name
4	Stick Gate at 1	Time, Gate Name
5	Lift Gate Fault	Time, Gate Name
6	Insert Fault in ROM	Time, Memory Id, Word Id, Bit Pos
7	Lift Fault from ROM	Time, Memory Id, Word Id, Bit Pos

Record Formats

Stop Run (op code = 2)
Record 1: op code, time format(*)

Stick Gate at 0 (op code = 3)
Record 1: op code, time format(*)
Record 2: device name format(a20)

Stick Gate at 1 (op code = 4)
Record 1: op code, time format(*)
Record 2: device name format(a20)

Lift Gate Fault (op code = 5)
Record 1: op code, time format(*)

```
  Record 2:   device name                          format(a20)

Insert Fault in Rom (op code = 6)
  Record 1:   op code, time                        format(*)
  Record 2:   Memory Id, Word Id, Bit Position     format(*)

Lift Fault from Rom (op code = 7)
  Record 1:   op code, time                        format(*)
  Record 2:   Memory Id, Word Id, Bit Position     format(*)
```

Following are descriptions of the individual items in the records:

Op Code: The one-digit code for the operation to be performed (see table above).

Time: The time at which the operation is to be performed, in units of emulator clocks or stacks. It should be noted that for op codes 3, 4, and 5, the sticking/lifting of the gate fault doesn't become effective until one clock after the time specified here.

Device name : the name of the device which is to be faulted or to have the fault lifted.

Memory Id :The memories are automatically numbered consecutively by the emulator, beginning with 1, in the order in which they appear in *mems.dat. This number is the Memory Id.

Bit Id:The bit id is the bit position in the target machine. The bits are numbered with bit position zero as the least significant position.

Word Id:The word id is the address containing the bit which is to be faulted. The word id is the actual target machine address if the emulator has performed the relocation to the QM-1 memory, but must be the absolute QM-1 address if the user did the relocation manually. See Section 5.4.1.2.3.2, items V1...Vn for a discussion of memory relocation.

Comments in Records:
Any record with * format can have a space after the last number and the rest of the record can contain comments. Any record with an A format can have comments after the last column specified for the character string.

5.4.2.2.3.3 Sample Fault List File

```
Insert and Lift Gate and Memory Faults    !Title
4,5                  !Run 1: stick gate named AND43 to 1 at time 5
AND43
2,40                 !       stop run 1 at time 40
3,12                 !Run 2: stick gate named AND44 to 0 at time 12
AND44
```

```
 4,12                !       stick gate named OR62 to 1 at time 12
OR62
5,50                 !       lift fault from gate named AND44 at time 50
AND44
2,100                !       stop run 2 at time 100
6,60                 !Run 3: insert fault in ROM at time 60
3,1000,13            !         stick bit 13 of word 1000 in memory 3
7,70                 !       lift fault from ROM at time 70
3,1000,13            !         lift from bit 13 of word 1000 in memory 3
2,150                !       stop run 3 at time 150
```

5.4.2.2.4 External Input Files

For each external input set that exists, the user must create one external input file for which he specifies the Vax Vms file name. No external input files are necessary if item AA in *eopts.dat is zero.

5.4.2.2.4.1 Contents and Structure of External Input Files

If item AA of file *eopts.dat is not zero, then one external input file must be created by the user in any manner he chooses for each external input set. The format for each such file is described below:

The file containing the actual external inputs list consists of one record for each insertion of an external input. Each record contains the time followed by the data bits to be inserted, in the following format:

(bn,i10,1x,o11)

The times for a given set must be in ascending order, and the data bits must be right justified. The maximum number of bits to be inputted in one data item is 32.

5.4.2.2.4.2 Sample External Input Files

Following are the entries in *eopts.dat which specify external input files:

```
4                                       !nexinp        no. of ei lists
combei1.dat                             !file name of first ei list
7                                       !no. of bits in first list
TS2G01                                  !names of devices feeding this list
TS2G02
TS2G03
TS2G05
TS2G06
TS2G07
TS2G08
combei2.dat                             !file name of second ei list
1
```

```
TS2G00
combei3.dat                                !file name of third ei list
1
TS1G00
combei4.dat                                !file name of fourth ei list
1
TS1G01
```

Following are contents of file COMBEI1.DAT

```
1                   000                    combei1.dat ba1-bd2
```

Following are contents of file COMBEI2.DAT

```
1                    0                     combei2.dat  ts2g00---ba1
18                   1
28                   0
```

Following are contents of file COMBEI3.DAT

```
1                    0                     combei3.dat ts1g00
```

Following are contents of file COMBEI4.DAT

```
1                    0                     combei4.dat ts1g01
40                   1
61                   0
180                  1
201                  0
```

5.4.2.3 Emulation Output Files

5.4.2.3.1 Text Output File

The contents of the emulation text output file(*eout.dat) created by the emulation program are almost completely under the control of the user. In the run options file, *eopts.dat, he specifies what outputs he wishes to appear in this file. See Appendix D for ten different samples of outputs produced by specific settings in *eopts.dat. Below is an explanation of these ten examples:

Outputs Which Appear in Every Run

Example 1:

1 Actual Date and time the run began.
2 Text which the user inserted in item A of file *iopts.dat.
3 Text which the user inserted in item A of file *eopts.dat.
4 Text which the user inserted as the first line in the file *fault.dat.
5 Emulation time at which the run completed.
6 Average stack size, minimum stack size, and maximum stack size over the entire run.
7 Actual Date and time the run ended.

Optional Outputs

All other outputs are optional and are controlled by the user in file *eopts.dat. The optional outputs are selected by the user in items F through Z6 in *eopts.dat. See Appendix D for examples of all of these outputs. Below are explanations for the examples.

Example 2:

Run Numbers (Item GG, Print Option 40)

1 The number of the run within the batch. (the runs are automatically numbered by the program in the order in which they occur in the fault file.

Stack Size (Item U, Print Option 29)

2 Stack size, i.e., the number of devices on the current stack, in octal.
3 Current time, in octal.
4 Stack size, in decimal.
5 Current time, in decimal.

Termination Dump (Item NN, Print Option 24)

6 Dump, in octal, of control store, local store, and main store at Termination Time.

Example 3:

Control Store Dump (Item K, Option 8)

1 Current time, in octal.
2 Current time, in decimal.
3 Address of first control store location dumped, in octal.
4 Contents, in octal, of successive control store locations, beginning with address in 3 above.

Main Store Dump (Item L, Option 9)

5 Current time, in octal.
6 Current time, in decimal.
7 Address of first main store location dumped, in octal.
8 Contents, in octal, of successive main store locations, beginning with address in 7 above.

Example 4:

Time Line (Item T, Option 28)

1 Current time, in octal
2 Current time, in decimal
3 The average size of the full stack as of the current time
4 The size of the smallest stack as of the current time
5 The size of the largest stack as of the current time

6 The static average fanout for the netlist, i.e., within the specified netlist, the average number of devices to which a device feeds.
7 The dynamic average number of destination devices examined for each source device on the stack, i.e., the average fanout for the devices which have been on the stack through the current time.
8 The dynamic average number of destination devices enqueued for each source device on the stack, i.e., the average number of devices whose output values have changed per each source device which has been on the stack through the current time.

Example 5:

Stack Dump in Abbreviated Mode (Item V, Option 30)

1 Current time, in decimal.
2 Name of Device on stack.
3 Value on output line of device named.

Example 6:

Insertion and Lifting of Gate Faults (Item W, Option 31)

1 Time at which fault was inserted, in octal.
2 Time at which fault was inserted, in decimal.
3 Value at which the output line of the gate was stuck.
4 Name of the device which was faulted.
5 Time at which fault was lifted, in octal.
6 Time at which fault was lifted, in decimal.
7 Name of the device whose fault was lifted.

Example 7:

Insertion and Lifting of Memory Faults (Item DD, Option 33)

1 Time at which fault was inserted, in octal.
2 Time at which fault was inserted, in decimal.
3 Memory Id into which fault was inserted.
4 Target Address into which fault was inserted.
5 Target Bit Number into which fault was inserted.
6 Absolute QM-1 address which holds faulted word.
7 Contents of QM-1 address prior to faulting.
8 Bit position of faulted bit, in QM-1 word.
9 Contents of QM-1 address after faulting.
10 Time at which fault was lifted, in octal.
11 Time at which fault was lifted, in decimal.
12 Memory Id from which fault was lifted.
13 Target Address from which fault was lifted.
14 Target Bit Number from which fault was lifted.
15 Absolute QM-1 address which holds fault to be lifted.
16 Contents of QM-1 address prior to lifting.
17 Bit position of faulted bit, in QM-1 word.
18 Contents of QM-1 address after lifting.

Example 8: Trace Stack (Items Z and Z1)

All items are the same as for example 10, except that item 9 will read "Trace Stack", and the only devices which will be outputted when they are on the stack are those whose names are listed in item Z of *eopts.dat.

Example 9: Device State Information (Items Z5 and Z6)

1 Current time, in octal.
2 Current time, in decimal.
3 Device Index Number.
4 Device Name.
5 Device Header Word, in octal (contains state information).
6 The QM-1 address of the header word for this device, in octal.

Example 10:

Stack Dump in Full Mode (Item N, Print Option 11)

1 Time of stack dump, in octal
2 Time of stack dump, in decimal
3 The average size of the full stack as of the current time
4 The size of the smallest stack as of the current time
5 The size of the largest stack as of the current time
6 The static average fanout for the netlist, i.e., within the specified netlist, the average number of devices to which a device feeds.
7 The dynamic average number of destination devices examined for each source device on the stack, i.e., the average fanout for the devices which have been on the stack through the current time.
8 The dynamic average number of destination devices enqueued for each source device on the stack, i.e., the average number of devices whose output values have changed per each source device which has been on the stack through the current time.
9 Description of what Selection Attribute the stack has, namely a "Complete" stack or a "Trace" stack
10 Sequential number representing the position of this item on the stack
11 The device index number of this device, in decimal.
12 The QM-1 address of the header word for this device, in octal.
13 The device name.
14 The value on the output line of the device.
15 The header word for this device, in octal.
16 The header word for this device, in binary.
17 The descriptive comment listed for this device in the Device Comments File. If no comment was given for this device, this field is blank.

5.4.2.3.2 Stack Outputs

A stack dump consists of a list of devices which are on the current stack. This dump has two attributes, namely the selection attribute and the format attribute. The selection attribute controls which devices will be selected for

printing, and the format attribute controls what information will be printed for each device that is selected. The attributes are selected by the user in the run options file *eopts.dat (see Section 5.4.2.2.2)

Selection Attribute:

Complete-Stack Mode:

In this mode, all devices that are currently on the stack are printed.

This mode is used if no device names are listed in item Z, and either item N or V is turned on.

Trace-Stack Mode:

In this mode, the user is attempting to trace the activity of specific devices and does not wish to see all the devices which are on the stack. He thus selects in item Z only the specific devices which he wishes to "trace", and when the stack is dumped, only those devices which he has selected will be dumped.

This mode is used if at least one device name is listed in item Z.

Format Attribute:

Full Mode:

In the full mode, the first line is always the Time Line which contains the current time in octal and in decimal,the average stack size, the minimum stack size, the maximum stack size, the average static fanout, average dynamic fanout examined during processing of stacks, and average dynamic fanout changing in value. Following the time line, every selected device from the current stack is dumped with its position on the stack, the device identification number, the header address in octal, the device name, the header contents in octal and in binary, and the user-supplied device description (if any) from *comm.dat.

Full mode is selected by turning on option 11 (item N).

Abbreviated Mode:

In the abbreviated format mode, no time line is printed, and each device selected is printed in an abbreviated mode. For each device that has been selected, the only items printed are the current time, the device name, and the output value of the device.

Abbreviated mode is selected by turning on option 30 (Item V).

Note: If neither full mode nor abbreviated mode is selected, then full mode will be used. If both full mode and abbreviated mode are selected, then abbreviated mode will be used.

Following is a table showing the results of all combinations of input options:

Item N	Item V	Item Z	Result	
			Selection	Format
0	0	no device	no stack	
0	0	some device	Trace	Full
0	1	no device	Complete	Abbreviated
0	1	some device	Trace	Abbreviated
1	0	no device	Complete	Full
1	0	some device	Trace	Full
1	1	no device	Complete	Abbreviated
1	1	some device	Trace	Abbreviated

(See Section 5.4.2.3.1 and Appendix D for examples of stack outputs.)

5.4.2.3.3 External Output Files

For a given batch, there may be zero or more external output files created. See Section 4.3.10.5 for a discussion of external outputs and Section 5.2.5 for a discussion of setup of external outputs. An output file will be written for each external output set at the completion of each batch.

5.4.2.3.3.1 Contents and Structure of External Output Files

In *eopts.dat, the user specifies the Vax Vms name he has selected for each external output file. For a given batch, the user-specifications for a specific external output set are the same for each run, but the outputs produced will probably differ from run to run due to the differences in the fault list for each run. Within each output file in ascending time sequence will be one entry for each time the external output action was triggered. Within a given external output file, the first entry for run i+1 will immediately follow the last entry for run i. Each entry consists of the time the action was triggered followed by the data at that time. The format for one entry is:

From the Vax Emulator: (I12/(1007))

From the QM-1 Emulator (after being transferred to Vax): (1x,1007)

One could process the external output files directly in either of of these formats; however, if one wishes to convert the QM-1 format to the Vax format, see Section 5.4.2.5.

5.4.2.3.3.2 Sample External Output File

Following are items from *eopts.dat which specify external output sets:

```
3                                   !no. of external output sets
combeo1.dat                         !file name for first eo set
7                                   !number of bits in each entry
500                                 !max no. of items in eo buffer
004003                              !control store address of data register
0,1,1                               !reschedule flag,start time,delta time
```

```
combeo2.dat                          !file name for second eo set
6
500
004004
0,1,1
combeo3.dat                          !file name for third eo set
14
500
004006
0,1,1
```

Following is external output file COMBEO1.DAT:

```
          20
  40000
          34
  40000
          48
 240000
          62
  24000
          76
      0
          90
      0
         104
  40000
         118
  40000
         132
  40000
         146
  40000
         160
  40000
         174
  40000
         188
 240000
         202
  24000
         216
      0
```

5.4.2.4 Running Emulator on Vax

Notation:

user represents the name of the user's root directory (without the brackets). For example, if the user's root directory is [Smith], then in this document, **user** represents Smith.

Userdata represents the directory and prefix name of the user's data files. For example, if the directory holding the data is named [smith.data], and all input files begin with prefix "counter", i.e., they are named counternet.dat, countermems.dat, counteriopts.dat, countercomm.dat, countereopts.dat,and counterfault.dat, then in this case **Userdata** represents [smith.data]counter.

Underlining implies a command which the user inputs to Vax VMS.

Make addition to login.com file.

Insert a command into your login.com file which sets the symbol "demuser" to the name of your root directory (without the brackets). For example, if the name of your root directory is [Smith], then insert the following command into your login.com file:

$demuser:==Smith

To Run Initialization

1. Prepare input data files.
2. $@[**user**.dem.run]iemu **Userdata**

To Run Emulation

1. Prepare input data files.
2. $@[**user**.dem.run]emu **Userdata**

Example:

Assumptions: Command files will reside on directory [smith.dem.run]
Data will reside on directory [smith.data], and prefix for all data files is "counter".

1. Create input data files with prefix "counter" on directory [smith.data].
2. $@[smith.dem.run]iemu [smith.data]counter (Run initialization)
3. $@[smith.dem.run]emu [smith.data]counter (Run emulation)

5.4.2.5 External Outputs Postprocessing

Reason for External Outputs Conversion

When the emulation has been performed on the Vax computer, the external outputs file is generated with format (I12/(1007)) for each external output record.

External output files which have been produced as a result of running an emulation on the QM-1 and which have been transferred back to the Vax are in QM-1 format which is: (1x,1007)

One could choose to process, on the Vax, the external output file from the QM-1, as is, and then no conversion would be necessary; however, if one wishes the external output file from the QM-1 to be in the same format as the external output files produced by the Vax emulator, which is: i12/(1007), then one could use the external outputs conversion program.

It should be noted that the current form of the conversion program assumes there are four QM-1 words outputted for each external outputs triggering; one could modify the source code if this number is different from four.

EoQM1format represents the directory and file name of the external output file which was transferred from the QM-1 to the Vax after the QM-1 emulation run.

EoVaxformat represents the directory and file name of the external output file which has been converted to Vax format.

Make addition to login.com file:

Insert a command into your login.com file which sets the symbol "demuser" to the name of your root directory (without the brackets). For example, if the name of your root directory is [Smith], then insert the following command into your login.com file:

$demuser:==Smith

Note: (Underlining implies a command which the user inputs to Vax Vms.)

To make changes to existing conversion Program:

$set default [**user**.dem.emulator]
Edit appropriate fortran module (either conveoqv or tconveoqv[1]) in [**user**.dem.emulator]
Do Fortran compiles of appropriate module(s)
$@[**user**.dem.run]linkconveoqv (links conversion programs)

To Run Conversion

Transfer external output file from QM-1 to Vax on **Userdata**

$@[**user**.dem.run]conveoqv **EoQM1format EoVaxformat**
(to convert without setting of high-order time bit)[1], or
$@[**user**.dem.run]tconveoqv **EoQM1format EoVaxformat**
(to convert with setting of high-order time bit)[1]

Example:

Assumptions: Programs will reside on directory [smith.dem.emulator], data will reside on directory [smith.data], and prefix for all data files is "counter".

1. Transfer external output file from QM-1 to Vax on [smith.data]
   The external output file transferred from the QM-1 is counterqm1eo.dat, and the new file in Vax format is to be named counterVaxeo.dat:
3. $@[smith.dem.run]conveoqv [smith.data]counterqm1eo.dat [smith.data]counterVaxeo.dat

1 During the transfer from the QM-1 to the Vax, the two high order bits of eighteen are not transferred (i.e., only 16 bits are transferred). If these two high order bits are not needed, use conveoqv. If the high order bit is needed, use tconveoqv.

5.4.3 Emulation on QM-1

5.4.3.1 Creation of QM-1 Files:

A. Use Nova Files Utility to create the following files:
(assume * is the user-selected prefix for all the files)

```
*:E
*COMP
*CS
*CS:S
*EXT
*EXT:S
*MAT
*MAT:S
*MEMC
*MEMC:S
*MEMM
*MEMM:S
*MS
*MS:S
*PAR
*PAR:S
*TCOMP
```

When the Diagnostic Emulation System Tape was restored to disk, three sets of files beginning with the prefixes "ONEC", "GF01", and "GF02", were created on user 6 of the disk. If one wishes to use any of these prefixes, he can make use of these files and thereby not have to create his own. In any case, ONECPAR:S and ONEC:E should be copied to create *PAR:S and *E:S respectively.

B. Use Editor to customize *TCOMP AND *:E.
The references to all data files must be changed to contain the appropriate prefix.

C. Use Editor to customize *PAR:S and *:E.
The following control store locations must contain the specified values:

Location	Value
147	address of top of first stack) + 1
601	address of memory control block
602	number of memory control records
603	memory master control store address
605	free space address
613	address of main store fault block
614	control store address of faulting device
615	address of operations action data structure

5.4.3.2 Data Preparation

A. Preparation of Data for Target Computer
(theoretically this step only need be done once)

1. Conversion and transfer of Memories file, *mems.dat.
a)On Vax Side:
1)Be sure all references to devices in *mems.dat have a 'D' or 'd' in column 1 instead of 'C' or 'c' (see()).
2) $@[**user**.dem.run]convmems **Userdata**
This step produces a file *memsq.dat which is memories file in QM-1 format.
3) Use a Vax editor to split *memsq.dat into *memc.dat and *memm.dat, where *memc.dat is the control store part and *memm.dat is the main store part.
4) Use the Vax-to-QM-1 Transfer program to transfer *memc.dat from the Vax to the QM-1.
5) On the QM-1 side: !!COPYSN DESTFILE *MEMC:S
6) Use the Vax-to-QM-1 Transfer program to transfer *memm.dat from the Vax to the QM-1.
7) On the QM-1 side: !!COPYSN DESTFILE *MEMM:S

2. Transfer of Net List and External Registers.

a)Run initialization program on Vax with print option 14 and print option 43 turned on.

This produces a file *mat.dat, which is the netlist in QM-1 format, and a file *extrn.dat, which is the file of external registers in QM-1 format.

1) Use the Vax-to-QM-1 Transfer program to transfer *mat.dat from the Vax to the QM-1.
2) On the QM-1 side: !!COPYSN DESTFILE *MAT:S
3) Use the Vax-to-QM-1 Transfer program to transfer *extrn.dat from the Vax to the QM-1.

4) On the QM-1 side: !!COPYSN DESTFILE *EXT:S

3. Assemble Target Data on QM-1:

Press Master Clear, Start
???LDNOV
!USER 6
!EX /*TCOMP

B. Preparation of Data for Batch Run (do this step for each batch run)

1. Run emulation on Vax with the following options turned on:

Turn on print option 41 to produce fault list for QM-1.
Turn on print option 44 to produce external input registers and external input list for QM-1, if using external inputs.
Turn on print option 45 to produce external output registers for QM-1.
Turn on print option 46 if do not want emulation performed on Vax. (i.e., if only purpose of run is to produce QM-1 outputs)

This produces a file *qmms.dat. This file contains the fault list in QM-1 format, and the external inputs list, if option 44 was turned on.

This produces a file *qmcs.dat which contains external inputs data registers and address registers if option 44 was turned on, and/or external outputs data registers and address registers if option 45 was turned on.

2. Use the Vax-to-QM-1 Transfer program to transfer *qmms.dat from the Vax to the QM-1.
3. On the QM-1 side: !!COPYSN DESTFILE *MS:S
4. Use the Vax-to-QM-1 Transfer program to transfer *qmcs.dat from the Vax to the QM-1.
5. On the QM-1 side: !!COPYSN DESTFILE *CS:S

6. Assemble Batch Data on QM-1:

Press Master Clear, Start
???LDNOV
!USER 6
!EX /SETUP

5.4.3.3 To Run Emulation on QM-1:

Press Master Clear, Start
???LD6/R*

C-2

5.4.3.4 To Send QM-1 External Outputs to Vax

A. On QM-1 Side:

```
Press Master Clear, Start
???LDEASY
SET DATE AND TIME
!!DATE,XX/XX/XX
!!TIME,XX/XX/XX
!!@
!!DIRectory,Search 1st=06,2nd=,08
!!DEADSTART
!!EASY-SPACE,BS=26,TPS=347777
!!<CR>
!!EXEC EOTODISK
```

B. Use QM-1-to-Vax Transfer program to transfer QM-1 external output file from QM-1 to Vax.

On QM-1 side:

```
!!EXEC QM1VAXI
  QM-1 TO Vax PIO TRANSFER FROM MEMORY
  INTERMEDIATE PRINTOUTS? ENTER Y OR N
```

On Vax Side:

```
$tqm1vaxi
(type in Vax output file name when requested)
```

2. Convert external outputs if desired. (see Section 5.4.2.5)

5.4.4 Vax <--> Qm1 File Transfers

5.4.4.1 Vax to Qm1 Transfers

Underlined characters are those which the user types into the Operating System.

step 1: (QM-1 side)

Mount Application Pack on QM-1 Drive 0.
Disk should be write-enabled.

```
Master Clear, Start
???LDEASY
!!DATE,XX/XX/XX
!!TIME,XX/XX/XX
!!@
!!DIRECTORY,SEARCH 1ST=06,2ND=,08
```

!!EX(ec) TVAXQM1
you will then see on the screen:
!!test,entry=vaxqm1,file=bbvaxqm1.
vax to qm1 file transfer

step 2: (Vax side)

$@[user.dem.transfers.vaxqm1]tvqi FILENAME(where "FILENAME" is name of the vax file to be transferred)

When transfer completes:

On Vax side, file "translog.dat" contains the transmission log.
On QM-1 side, the new file is in DESTFILE.

optional step 3: (QM-1 side)
(do this step only if transferred file is to be used under Nova Operating System)

!!COPYSN DESTFILE NOVAFILE

(where "NOVAFILE" is the name of the Nova file)

5.4.4.2 Qm1 to Vax Transfers

Underlined characters are those which the user types into the Operating System.

step 1: (QM-1 side)

Mount Application Pack on QM-1 Drive 0.

Master Clear, Start
???LDEASY
!!DATE,XX/XX/XX
!!TIME,XX/XX/XX
!!@
!!DIRECTORY,SEARCH 1ST=06,2ND=,08
!!EX(ec) TQM1VAXI
you will then see on the screen:
!!test,entry=mv,file=bbmv.
qm1 to vax file transfer

step 2: (Vax side)

$@[user.dem.transfers.qm1vax]tqv FILENAME(where "FILENAME" is name of the Vax file to be created)

When transfer completes:

On Vax side, the new file is in "filename"
On QM-1 side, file "translog" contains transmission log.

6. Bibliography

6.1 References

1. Baker, R., Mangum, S., Scheper, C., A Fault Injection Experiment Using the AIRLAB Diagnostic Emulation Facility, NASA CR-178390, Research Triangle Institute, Research Triangle, North Carolina, December, 1987.

2. Migneault, G. E., On The Diagnostic Emulation Technique And Its Use In The AIRLAB, NASA TM-4027 (to be published 1988).

3. Nanodata Corporation, QM-1 Hardware Level Users Manual, Third Edition, Revision 3, Buffalo, New York, July, 1983.

4. Nanodata Corporation, QM Micro, Version 1.3, Second Edition, Williamsville, New York, 1976.

5. Nanodata Corporation, Multi Micromachine Description, Revision 2, December, 1976.

6. Nanodata Corporation, QM-1 Nanoassembler Programmer's Reference Manual, First Edition, February, 1980.

7. Nanodata Corporation, QM - NCS Operations Guide, Buffalo, New York, October, 1981.

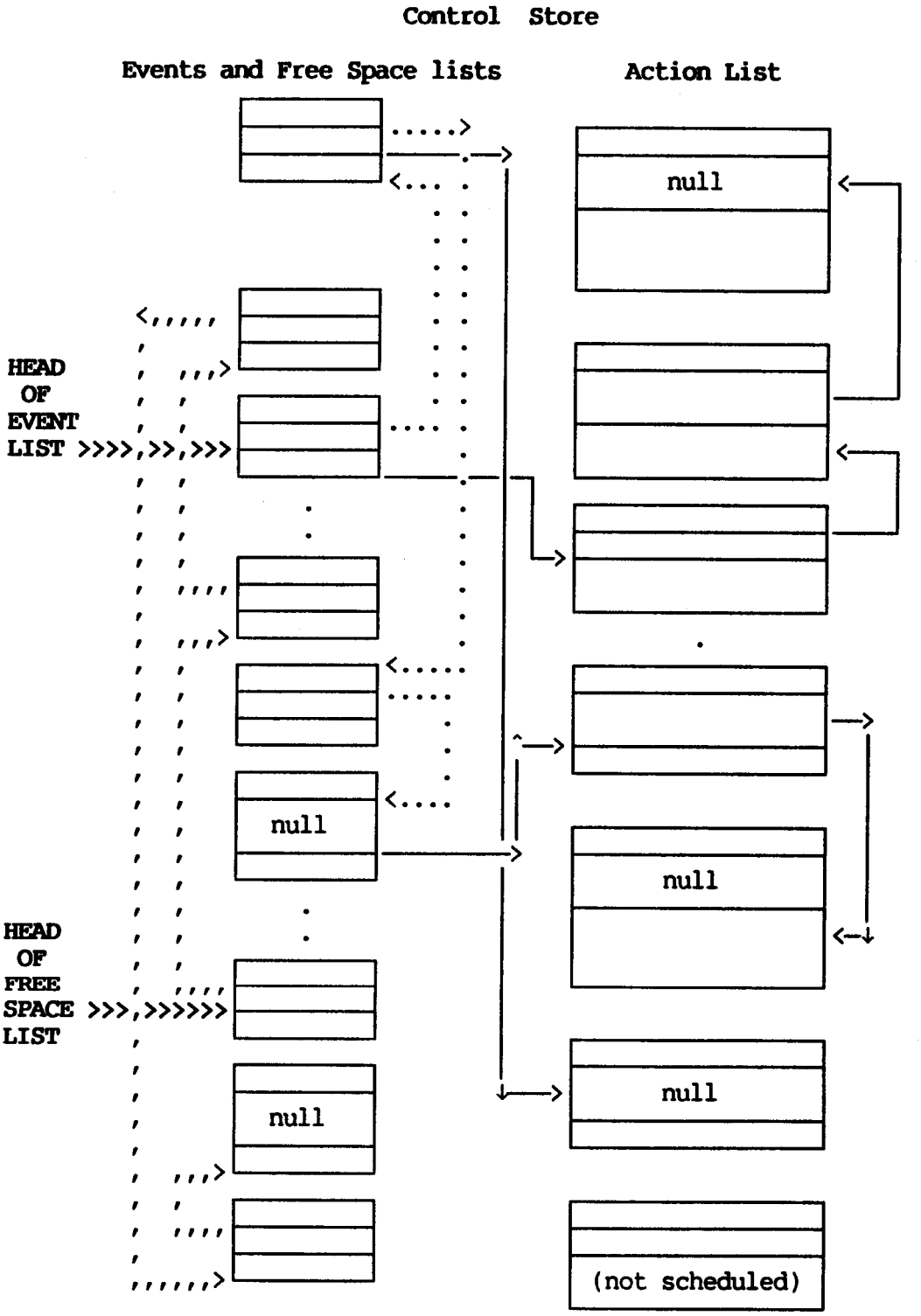
8. Naples, Charles, J., Emulation Aid System II (Easy II) System Programmer's Guide, Naval Surface Weapons Center, Dahlgren Laboratory Technical Report NSWC TR 81-98, Dahlgren, Virginia, March 1981.

9. Naples, Charles, J., Simpl-Q Reference Manual, Naval Surface Weapons Center, Dahlgren Laboratory Technical Report NSWC TR 81-262, Dahlgren, Virginia, May, 1981.

Appendix A

Additional Figures

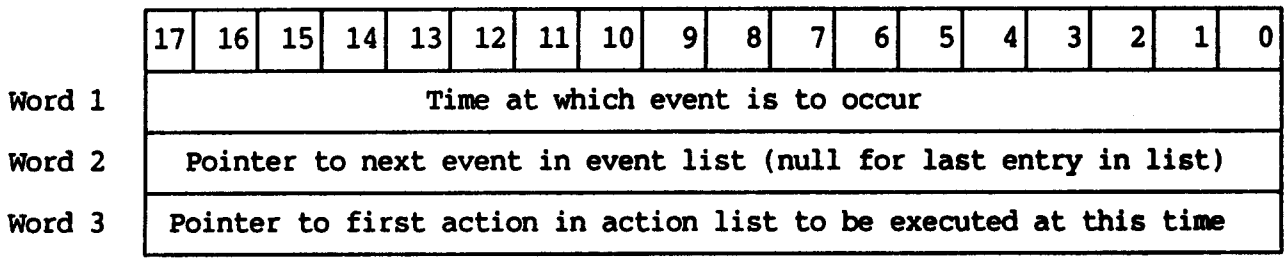
Event, Free Space, and Action List Layouts



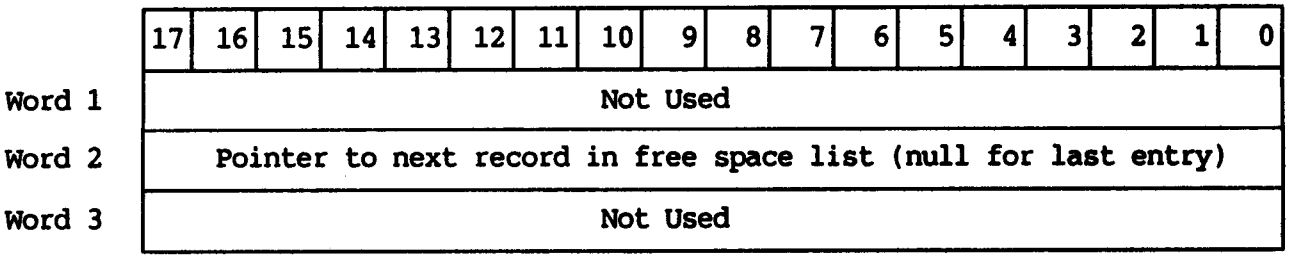
Event and Free Space Record Layouts

Control Store

Event Record Layout

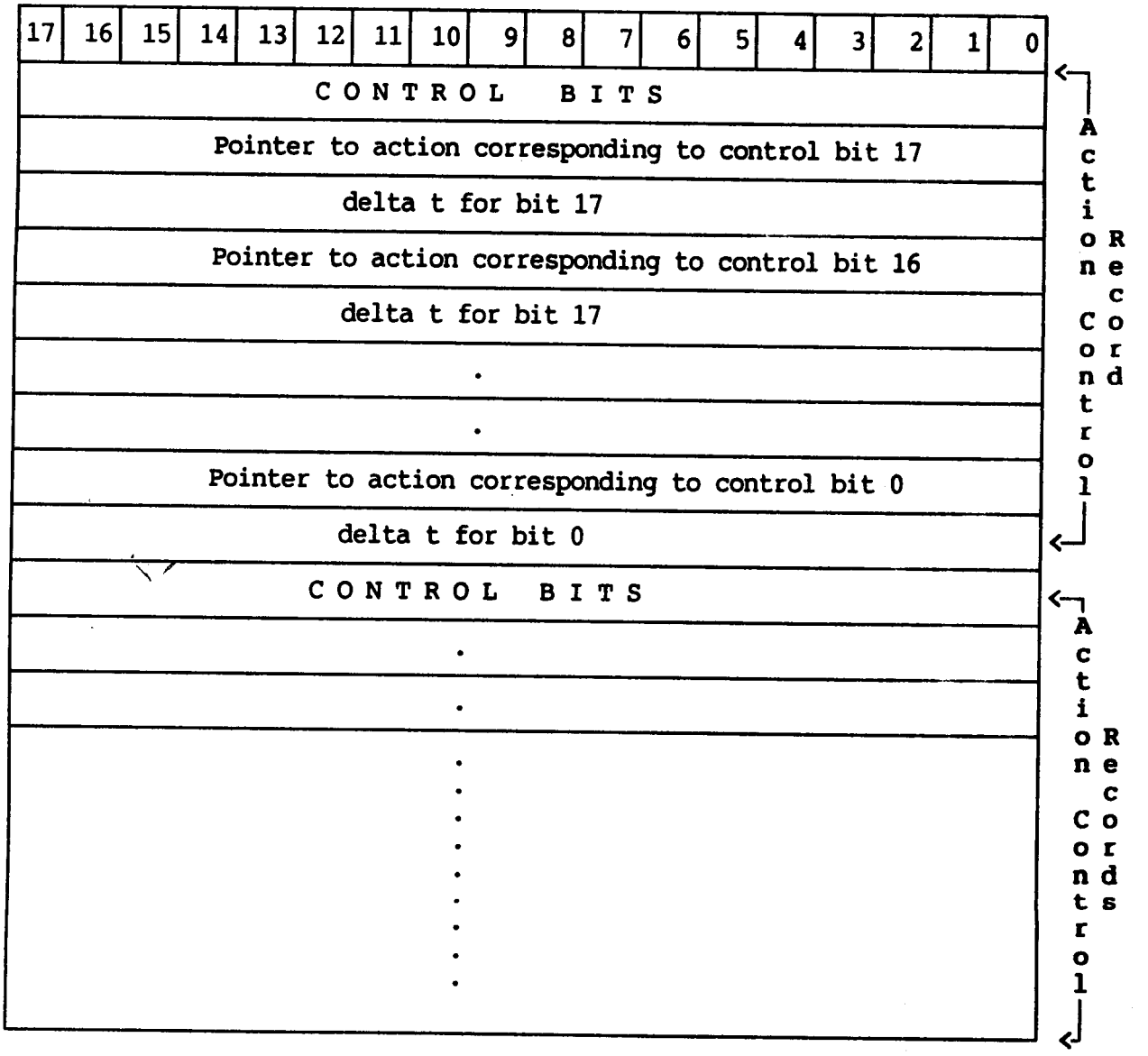


Free Space Record Layout

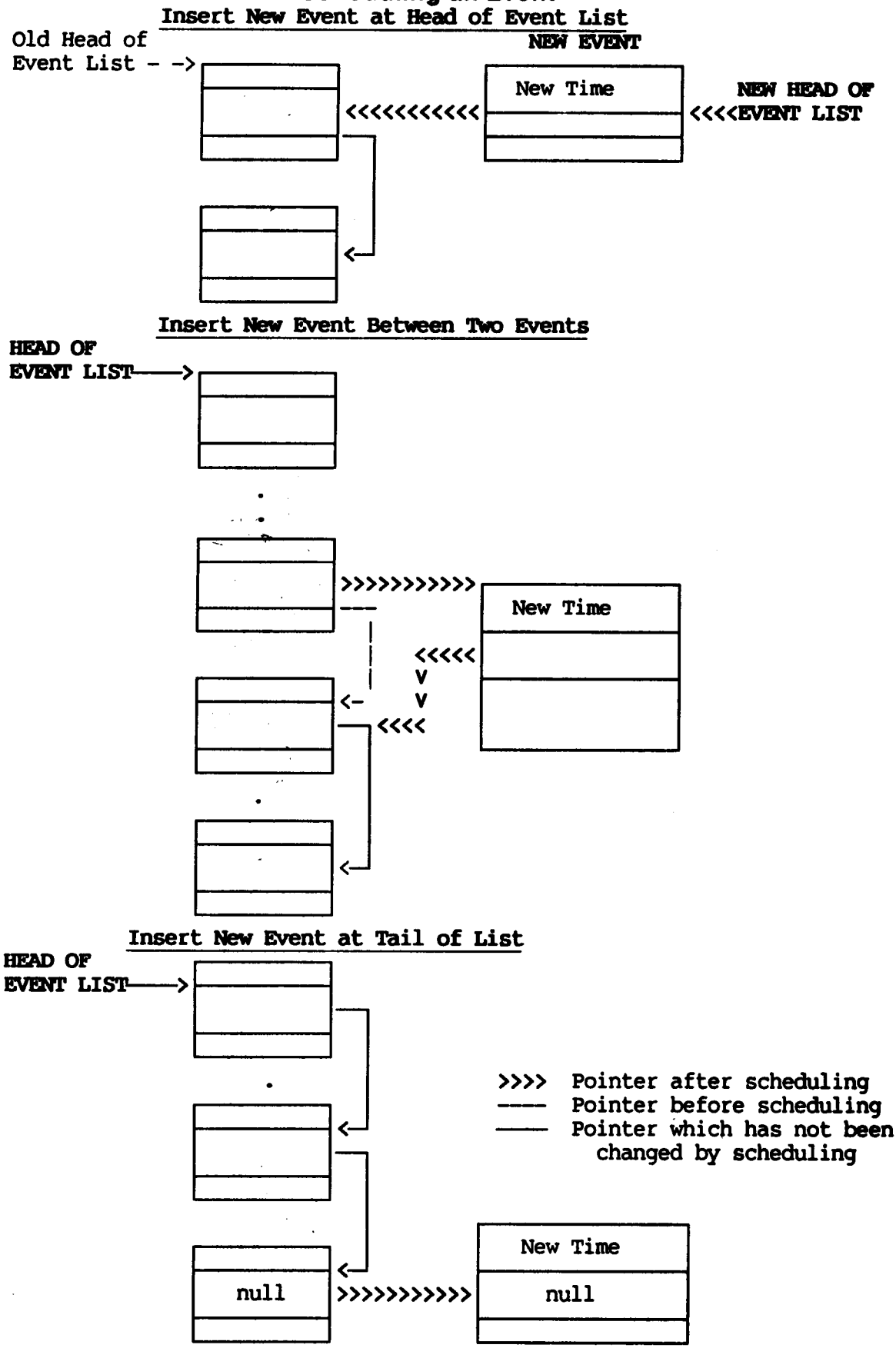


Action Control Block Layout

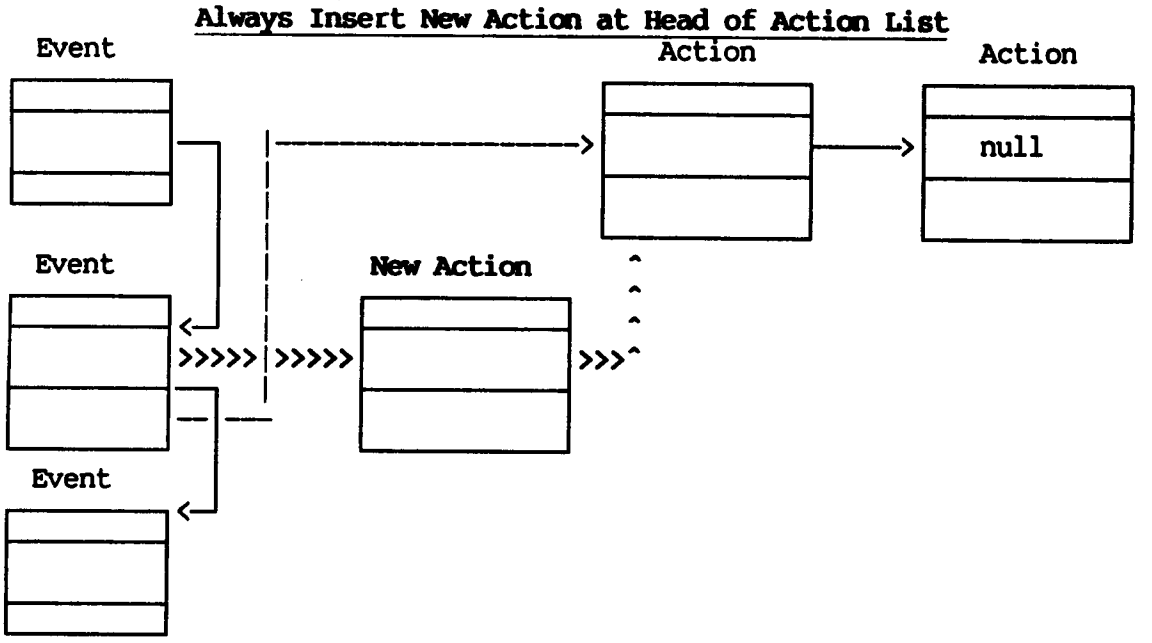
Control Store



Scheduling an Event



Scheduling an Action



>>>>> new pointer after scheduling
——— old pointer before scheduling, which has been replaced
—— pointer which has not been changed by scheduling

Flip-Flop Trigger Chart

(for downward edge-triggered flip-flop)

INPUTS						RESULTS			
						External		Internal	
P	C	T	L	J	K	Q_{n+1}	$\bar{Q}_{n+1}$	iQ_{n+1}	$i\bar{Q}_{n+1}$
1	1			–	–	(no change)			
0	0	–	–	–	–	1	1		
0	1	–	–	–	–	1	0		
1	0	–	–	–	–	0	1		
1	1	↑		–	–	(no change)			
1	1	↓		0	0	(no change)			
1	1	↓		0	1	0	1		
1	1	↓		1	0	1	0		
1	1	↓		1	1	$\bar{Q}_n$ (or indeterminant if R=1)	$\bar{Q}_n$		
1	1		↑	0	0	(no change)		(no change)	
1	1		↑	0	1	(no change)		0	1
1	1		↑	1	0	(no change)		1	0
1	1		↑	1	1	(no change)		$\overline{iQ_n}$ (or indeterminant if R=1)	$\overline{i\bar{Q}_n}$
1	1		↓	–	–	iQ_n	$i\bar{Q}_n$	(indeterminant if U=1 and J or K changed while L=1; no changes otherwise)	

Q_n Q external value at time n
Q_{n+1} Q external value at time n+1
iQ_n Q internal value at time n
iQ_{n+1} Q internal value at time n+1
↑ transition from 0 to 1
↓ transition from 1 to 0

Fortran Initialization I/O Units

Inputs

Fortran Variable Name	Logical File Name	Vax VMS File Name	Description
UIN	FOR008	*net.dat	the target network description(netlist) in DENF format
UIN0	FOR007	*comm.dat	the comments or descriptions to appear alongsode device names when they appear on the stack output
UIN1	FOR010	*mems.dat	the initial values to be resident in the host memory before the emulation begins
UIN2	FOR011	*iopts.dat	the user runtime initialization parameters

Outputs

Fortran Variable Name	Logical File Name	Vax VMS File Name	Description
UOUT	FOR014	*iout.dat	output text file from initialization
UOUT1	FOR012	*alph.dat	alphabetic list of devices: device name, device number, device type, device class, initial output value
UOUT2	FOR013	*nam.dat	template for creating *comm.dat device number, device number, device name
UOUT3	FOR019	*mat.dat	entire matrix in format to go to QM-1
UOUT4	FOR020	*check.dat	debugging information
UOUT5	FOR021	*save.dat	all initialized data structures in binary form
UOUT8	FOR028	*extrn.dat	control store externals to go to QM-1

Fortran Emulation I/O Units

Inputs

Fortran Variable Name	Logical File Name	Vax VMS File Name	Description
UIN2	FOR011	*eopts.dat	user runtime options for emulation
UIN3	FOR015	*fault.dat	fault list
UIN4	FOR016	(user name)	external input lists
UIN6	FOR009	*save.dat	all initialized data structures (in binary) created by initialization program

Outputs

Fortran Variable Name	Logical File Name	Vax VMS File Name	Description
UOUT	FOR014	*eout.dat	text output file from emulation
UOUT6	FOR026	(user name)	external outputs files
UOUT7	FOR027	*qmms.dat	main store contents to go to QM-1
UOUT9	FOR029	*qmcs.dat	control store contents to go to QM-1

User Modifications to Fortran Module to Execute One Action

```
c***********  Make changes where indicated by "++++++++++++++++++++++++++"


C$$$$$$ EX1ACT  EXECUTE ONE ACTION
C         INPUTS :GPA - PTR TO ACTION TO BE EXECUTED
C         OUTPUTS:EXECUTED ACTION
C                 IF INVALID ACTION CODE, PRINT ERROR & STOP
      SUBROUTINE EX1ACT
      IMPLICIT INTEGER (A-Z)
         INCLUDE 'COMM20.FOR/list'
         INCLUDE 'COMM21.FOR/list'
         INCLUDE 'COMM22.FOR/list'
         INCLUDE 'EMUPARAM.FOR/list'
C*************************************************************************
         LOGICAL*1 CFALSE
        DATA CFALSE/.FALSE./
        DATA LMCODE/'774000'O/
C       INCLUDE 'GETCS.FOR/list'
        INCLUDE 'CLEAR.FOR/list'

C     EXECUTE ACTION
        LACT=CS(GPA)
        LACODE=(LACT.AND.LMCODE)/DIVACT     !ACTION CODE RIGHT JUSTIFIED
        GO TO (10,20,30,40,50,60,70,80),LACODE          !AIRLAB ACTIONS
        IF((LACODE.GE.ILLAC1).AND.(LACODE.LE.ILLAC2))GO TO 500 !U.OF ILL.

c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c *** Insert "IF" here checking for new action code and branch to newly
inserted call to user-written action*** , for example:
c        IF (LACODE.EQ.NEWCODE)GO TO 600
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        WRITE(UOUT,1000)gpa,LACT,LACODE
        call termrn
        STOP
C       ACTION 1 - FILL BUFFER
 10       CALL ACT1
          GO TO 250
C       ACTION 2 - WRITE MEMORY
 20       CALL ACT2
          GO TO 250
C       ACTION 3 - READ MEMORY
 30       CALL ACT3
          GO TO 250
C       ACTION 4 - DUMP NON-EMPTY BUFFER TO DISK
 40       CALL ACT4
          GO TO 250
C       ACTION 5 - STOP RUN
 50       CALL ACT5
          GO TO 250
C        ACTION 6-EXECUTE OPERATIONS
 60        CALL ACT6
           GO TO 250
C        ACTION 7-EXTERNAL INPUTS
 70        CALL ACT7
```

```
          GO TO 250
C       ACTION 8-EXTERNAL OUTPUTS
 80       CALL ACT8(.FALSE.)          !NORMAL WRITE, NOT END OF RUN MARKER
          GO TO 250
C        ACTIONS FOR UNIV. OF ILLINOIS
 500      CALL ACTILL(LACODE)
          GO TO 250
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C*** Insert call to new module followed by GO TO 250   *** , for example:
C      Also compile and link NEWSUB as described in ().

 600      CALL NEWSUB
          GO TO 250
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

C     DO RESCHEDULINGdd
 250    IF((LACT.AND.CMASK(18)).EQ.0)THEN
          CALL PUTCS(GPA,CLEAR(LACT,CMASK(10)))
        ELSE
          CALL REACT
        ENDIF
 300  RETURN
 1000 FORMAT(' INVALID ACTION - address= ',o6,'word 1= ',
     x         O6,' action code= ',I10)
      END
```

Fortran Parameters & Common Variables, Sorted by Common Label

Name	Dimension	Common Label	Description
maxconn		parameter	-max no. of internal connections allowed
maxgate		parameter	-max no. of gates allowed
xname	(4000)	co6	character*20-device names, set by getdevn
prloc	(3,2)	co8	low & high address for cs,ms,ls for output
prsw	(50)	co8	user print option switched, 0=off, 1=on
prtime	(30)	co8	print window 1=start,2=stop,3=delta
prtisw	(10)	co8	1=print window flag(1=on)
nconnec		comm1	no. of connections, set by preproc
nextern		comm1	no. of external connections, set by preproc
ngates		comm1	no. of devices, set by initrn=neqn
runtitle	(10)	comm1	title for run, read in getparm from eopts file
title	(10)	comm1	i*4-title for output,read from opts file by initrn
xaddres	(4000)	comm1	qm1 control store address for header for device i
xconn	(10000)	comm1	full address for internal connection
xhdr	(4000)	comm1	header for each device i
xhigh	(4000)	comm1	index to connection list for last conn for device i
xlink	(10000)	comm1	first word of internal connector record
xlow	(4000)	comm1	index to connection list for first conn for device i
zptr	(10000)	comm1	the index of the dest device for this connection
xcount	(4000)	comm11	initial value of "count" for each device
xstack	(4000)	comm11	stack flag for device(0=not on,1=is on 1st stack)
datebuf		comm14	character*9-current date for output
timebuf		comm14	character*8-current time for output
dchigh	(4000)	comm15	high index for each device, into dcommen
dclow	(4000)	comm15	low index for each device, into dcommen
dcommen	(10000)	comm15	character*1-one string holding all device comments
csopact		comm16	ptr to op action structure in cs(calc from read-in)
cspflt		comm16	ptr to header in cs of faulter device(read in)
endbat		comm16	l*1 true if at end of batch(calc)
endrun		comm16	l*1 true if at end of run(calc)
ftitle		comm16	fault list title,read by colist,used act6 & schnop
infltr		comm16	index no. of faulter device(read in)
memadr	(30)	comm16	memory relocation constants
msfblk		comm16	ptr to ms fault blk(read in)
msnxfl		comm16	ptr to next op to be sched.,init by colist,inc in act6
ngfcon		comm16	no. of gate faults this stack(calc)
nomems		comm16	no. of rom and ram memories with relocation
nops		comm16	no. of ops in batch(calc)
opsize	(15)	comm16	no. of words for corresponding op
pfltcon		comm16	ptr to next fault connection(calc)
timesiz		comm16	no. of qm1 wds to hold time(read in)
cseiac	(21)	comm17	cs addr of 1st word of each ei action(read 1,calcrest)
cseial		comm17	last possible ei action entry(calculated)
cseiar		comm17	loc in cs of first ei address register(read)
cseidr		comm17	loc in cs of first ei data register(read)
mseile		comm17	last possible ei list entry(calculated)
mseili		comm17	loc in ms of first ei list(read)
nexinp		comm17	actual number of ei sets for this batch(read)
cseoac	(21)	comm18	cs addr of 1st word of each eo action(read 1,calcrest)
cseoal		comm18	not used
cseoar		comm18	loc in cs of first eo address register(read)
cseodr	(20)	comm18	loc in cs of data register

Fortran Parameters & Common Variables, Sorted by Common Label

Name	Dimension	Common Label	Description
eofile	(20)	comm18	char*40-name for external output file(read)
eonwrd	(20)	comm18	no. qm1 wrds per datum in eo action-use getparm,termrn
eorfl	(20)	comm18	byte-external output reschedule flag(1=on)
eorstr	(20)	comm18	external output start time for rescheduling
mseobu		comm18	loc in ms of first eo buffer
mseole		comm18	not used
nexoup		comm18	no. of external output sets
dmask	(0:17)	comm19	mask for bit 0,0-1,0-2,...0-17
csaddr		comm2	qm1 control store address for matrix
csexter		comm2	qm1 control store address for first external register
cstime		comm2	qm1 control store address for storing time for outputs
msexter		comm2	qm1 main store address for first external register
xehigh	(4000)	comm2	index to last external data structure for each device
xei	(4000)	comm2	1=external complemented,0=not (not needed after init)
xelink	(4000)	comm2	external link word
xelow	(4000)	comm2	index to first external data structure for each device
xew	(4000)	comm2	qm1 cs or ms address of external
cmask	(0:19)	comm20	mask for bit 0,1,2...17,mask for bits 8&9,0(not used)
cstopa		comm21	cs address of stop action
gnewt		comm21	time for new event to be scheduled
gnmcon		comm21	number of action control records
gpa		comm21	general purpose pointer to action
gpe		comm21	general purpose pointer to event
gpevhd		comm21	ptr to head of event list, init by initfe
gpfrhd		comm21	ptr to head of free space list, init by initfe
gpmcon		comm21	pointer to action control block
gpmmas		comm21	pointer to master action control register
gpnewa		comm21	pointer to new action
gpnewe		comm21	pointer to newly allocated event
gsflag		comm21	stop flag(1=stop)
gstime		comm21	user-defined stop time
gtime		comm21	current time
cs	(0:20000)	comm22	qm1 control store
cssup		comm22	highest cs loc to save on save file
ls	(0:31)	comm22	qm1 local store
lssup		comm22	highest ls loc to save on save file
ms	(0:70000)	comm22	qm1 main store
mssup		comm22	highest ms loc to save on save file
pcslow		comm22	parameter-low dimension for control store (0)
pcsup		comm22	parameter-high dimension for control store (20000)
plslow		comm22	parameter-low dimension for local store (0)
plsup		comm22	parameter-high dimension for local store (37)
pmslow		comm22	parameter-low dimension for main store (0)
pmsup		comm22	parameter-high dimension for main store (70000)
ntrace		comm24	no. of devices to be traced
xtrace	(4000)	comm24	byte-trace flag(0=dont print output changes,1=do)
nupcho		comm25	no. of user print choices(output formats)
upcsms	(15)	comm25	user print choice memory type(0=cs,1=ms,2=ls)
upform	(15)	comm25	character*80-user print choice format incl. ()
uploc1	(15)	comm25	user print choice low mem address to output
uploc2	(15)	comm25	user print choice high mem address to output
uptitle	(15)	comm25	character*20-user print choice title to output

Fortran Parameters & Common Variables, Sorted by Common Label

Name	Dimension	Common Label	Description
zfullw	(10000)	comm26	byte
nheads		comm27	no. of devices to have headers printed
xheadt	(500)	comm27	indexes of devices to have headers printed
nchange		comm28	no. of headers that changed this stack
xchange	(4000)	comm28	byte- 0 if x didn't change this stack, 1 if did
xchid	(1000)	comm28	index nos. of the headers that changed this stack
checkon		comm29	equivalence(sw1,sw(1),checkon)
sw	(20)	comm29	
sw1		comm29	logical-true if prsw(3) and prsw(4) on(check hdrs)
sw2-sw20		comm29	logical-switches(not used)
xebit	(4000)	comm3	bit no. for external (not needed after init)
xecsms	(4000)	comm3	type of external(0=cs,1=ms,2=ls)(not needed afterinit)
xereg	(4000)	comm3	external register no. (cs,msls) (not needed afterinit)
divear		comm30	divisor for emulator address reg. to right justify
emask	(1:18)	comm31	masks for bits 17,17-16,17-15...17-0
adfand		comm4	real-denom,avg dyn fanout,calc in pstack,used
adfcn		comm4	real-numerator,avg dyn fanout change,i.e., enequeued
adfen		comm4	real-numerator,avg dyn fanout examined
asfan		comm4	real-average static fanout, set by getdevn
nstack	(2)	comm4	number of items in stack i
s		comm4	current stack number (1 or 2)
savg		comm4	real-average number of items on stack
sbar		comm4	non-current stack number (1 if s=2, 2 if s=1)
smax		comm4	maximum number of items on stack
smin		comm4	minimum number of items on stack
stack	(2,500)	comm4	current & new stacks holding indices of stack devices
ingnin		comm5	value to assign to output for devices with no inputs
initfl		comm5	initialization flag (0=user,1=computer)
iprclr		comm5	print-clear convention flag: 0(1=benign) 1(1=active)
ntri		comm5	number of devices with defined output values
triang	(4000)	comm5	indices of all devices with output value defined
xeval	(4000)	comm5	external value for device
xffval	(4000)	comm5	"PCTLJK" values for flip-flop
xhead	(4000)	comm5	pts to 1st entry in conn list(this device is destin.)
xival	(4000)	comm5	internal value for device
xnudef	(4000)	comm5	no. of undefined inputs for this device
xpval	(4000)	comm5	predefined output value for device
dconnt	(10000)	comm6	connection type for internal connection
dinnum	(10000)	comm6	index no. of the source device for connectioni
dinval	(10000)	comm6	value on input line coming into device
drflag	(10000)	comm6	reversal flag for connection
dxnext	(10000)	comm6	ptr to next item in connection list w.same dest device
xclass	(4000)	comm8	device class(gate,flip-flop,or tri-state)
xdis	(4000)	comm8	disconnected output value for tri-states
xr	(4000)	comm8	R value
xtype	(4000)	comm8	device type(fiip-flop,and,nand,or,etc.)
xu	(4000)	comm8	U value
xvalue	(4000)	comm8	output value for device
c100o			parameter-constant of 100(octal)
c10o			parameter-constant of 10(octal)
clff			parameter-device class for ff (2)
clgate			parameter-device class for gate (1)

Fortran Parameters & Common Variables, Sorted by Common Label

Name	Dimension	Common Label	Description
clts			parameter-device class for tri-state (3)
cnqbit			parameter-number of bits in qm1 word (18)
cnull			parameter-(0)
connhi			parameter-highest valud value for gate types(7)
connlo			parameter-lowest valid value for gates types(1)
cpdval			parameter-user output value for computer calculated(9)
csentl			parameter-sentinel of -1 for action 8
ctyc			parameter-connection type to flip-flop input c (2)
ctyd			parameter-connection type to flip-flop input d (7)
ctyen			parameter-connection type to enable line of tri-state
ctygts			parameter-connection type to regular gate (0)
ctyj			parameter-connection type to flip-flop input j (5)
ctyk			parameter-connection type to flip-flop input k (6)
ctyl			parameter-connection type to flip-flop input l (4)
ctyp			parameter-connection type to flip-flop input p (1)
ctyt			parameter-connection type to flip-flop input t (3)
divact			parameter-divisor to righ-justify action code in action(2048)
dumtime			parameter-dummy time to insert into stop action
eiasize			parameter-max size in cs fo all ei actions(1000 o)
eilsize			parameter-max size in ms for all ei lists
fbsize			parameter-no. of qm1 words in ms fault buffer
illac1			parameter-constant for U. of Ill. lowest action code(50)
illac2			parameter-constant for U. of Ill. highest action code(52)
illin1			parameter-input unit for045-for U. of Illinois use only
illout1			parameter-output unit for040-for U. of Illinois use only
infin			parameter-infinity(2147483647=max no. for i*4)
maxgfl			parameter-max no. of gate faults per single time(30)
maxmem			parameter-max. no. of target memories(30)
maxnei			parameter-max no. ei sets (20)
maxupch			parameter-maximum no. of user print choices (output formats)
mbit0			parameter-mask for rightmost bit 0 (1)
mnlbit			parameter-mask for no. leftover bits in action
mnword			parameter-mask f. #qm1 wds/targer wd in 1st wd action('340'o)
mul1			parameter-(17)
mul2			parameter-(1)
mulnbi			parameter-divisor to right-justify mnlbit (1)
mulnwo			parameter-divisor to right-justify mnword (32)
nthead			parameter-max no. of devices which can have headers printed
numvops			parameter-number of valid op codes(8)
opgtype			parameter-gate fault op type(1)
oplifg			parameter-op code for lift gate fault(5)
oplifm			parameter-op code for lift memory fault(7)
opmtype			parameter-memory fault op type(2)
opsbat			parameter-op code for stop batch(1)
opsrun			parameter-op code for stop run(2)
opstg0			parameter-op code for stick gate at 0(3)
opstg1			parameter-op code for stick gate at 1(4)
opstm			parameter-op code for fault memory(6)
tema71			parameter-mask template for action 7, word 1('034000'o)
tema81			parameter-template, action 8, word 1, no rescheduling
temb81			parameter-template, action 8, word 1, resched on
tyand			parameter-device type for and gate (1)

Fortran Parameters & Common Variables, Sorted by Common Label

Name	Dimension	Common Label	Description
tyff			parameter-device type for ff (0)
tynand			parameter-device type for nand gate (2)
tynor			parameter-device type for nor gate (4)
tynot			parameter-device type for not gate (5)
tynxor			parameter-device type for nxor gate (7)
tyor			parameter-device type for or gate (3)
tyxor			parameter-device type for xor gate (6)
uin			parameter-input unit for008-matrix(bdxhd2s.dat)
uin0			parameter-input unit for007-device comments(bdxcomm.dat)
uin1			parameter-input unit for010-target memories(bdxmems.dat)
uin10			parameter-input unit for025-
uin2			parameter-input unit for011-user options(bdxopts.dat)
uin3			parameter-input unit for015-
uin4			parameter-input unit for016-
uin5			parameter-input unit for017-
uin6			parameter-input unit for009-
uin7			parameter-input unit for022-
uin8			parameter-input unit for023-
uin9			parameter-input unit for024-
uout			parameter-output unit for014-output file(bdxout.dat)
uout0			parameter-output unit for018-
uout1			parameter-output unit for012-alpha device list(bdsalph.dat)
uout10			parameter-output unit for030-
uout2			parameter-output unit for013-device name list(bdxnam.dat)
uout3			parameter-output unit for019-matrix for qm1(bdxmat.dat)
uout4			parameter-output unit for020-binary checkingfile(bdxcheck.dat)
uout5			parameter-output unit for021-
uout6			parameter-output unit for026-
uout7			parameter-output unit for027-
uout8			parameter-output unit for028-
uout9			parameter-output unit for029-
vophigh			parameter-highest valid user op code(7)
voplow			parameter-lowest valid user op code(1)

Fortran Parameters & Common Variables, Sorted by Variable Name

Name	Dimension	Common Label	Description
adfand		comm4	real-denom,avg dyn fanout,calc in pstack
adfcn		comm4	real-numerator,avg dyn fanout change,i.e., enequeued
adfen		comm4	real-numerator,avg dyn fanout examined
asfan		comm4	real-average static fanout
c100o			parameter-constant of 100(octal)
c10o			parameter-constant of 10(octal)
checkon		comm29	equivalence(sw1,sw(1),checkon)
clff			parameter-device class for ff (2)
clgate			parameter-device class for gate (1)
clts			parameter-device class for tri-state (3)
cmask	(0:19)	comm20	mask for bit 0,1,2...17,mask for bits 8&9,0(not used)
cnqbit			parameter-number of bits in qm1 word (18)
cnull			parameter-(0)
connhi			parameter-highest valud value for gate types(7)
connlo			parameter-lowest valid value for gates types(1)
cpdval			parameter-user output value for computer calculated(9)
cs	(0:20000)	comm22	qm1 control store
csaddr		comm2	qm1 control store address for matrix
cseiac	(21)	comm17	cs addr of 1st word of each ei action(read 1,calcrest)
cseial		comm17	last possible ei action entry(calculated)
cseiar		comm17	loc in cs of first ei address register(read)
cseidr		comm17	loc in cs of first ei data register(read)
csentl			parameter-sentinel of -1 for action 8
cseoac	(21)	comm18	cs addr of 1st word of each eo action(read 1,calcrest)
cseoal		comm18	not used
cseoar		comm18	loc in cs of first eo address register(read)
cseodr	(20)	comm18	loc in cs of data register
csexter		comm2	qm1 control store address for first external register
csopact		comm16	ptr to op action structure in cs(calc from read-in)
cspflt		comm16	ptr to header in cs of faulter device(read in)
cssup		comm22	highest cs loc to save on save file
cstime		comm2	qm1 control store address for storing time for outputs
cstopa		comm21	cs address of stop action
ctyc			parameter-connection type to flip-flop input c (2)
ctyd			parameter-connection type to flip-flop input d (7)
ctyen			parameter-connection type to enable line of tri-state
ctygts			parameter-connection type to regular gate (0)
ctyj			parameter-connection type to flip-flop input j (5)
ctyk			parameter-connection type to flip-flop input k (6)
ctyl			parameter-connection type to flip-flop input l (4)
ctyp			parameter-connection type to flip-flop input p (1)
ctyt			parameter-connection type to flip-flop input t (3)
datebuf		comm14	character*9-current date for output
dchigh	(4000)	comm15	high index for each device, into dcommen
dclow	(4000)	comm15	low index for each device, into dcommen
dcommen	(10000)	comm15	character*1-one string holding all device comments
dconnt	(10000)	comm6	connection type for internal connection
dinnum	(10000)	comm6	index no. of the source device for connectioni
dinval	(10000)	comm6	value on input line coming into device
divact			parameter-divisor to righ-justify action code in action(2048)
divear		comm30	divisor for emulator address reg. to right justify
dmask	(0:17)	comm19	mask for bit 0,0-1,0-2,...0-17

Fortran Parameters & Common Variables, Sorted by Variable Name

Name	Dimension	Common Label	Description
drflag	(10000)	comm6	reversal flag for connection
dumtime		parameter-dummy time to insert into stop action	
dxnext	(10000)	comm6	ptr to next item in connection list w.same dest device
eiasize		parameter-max size in cs fo all ei actions(1000 o)	
eilsize		parameter-max size in ms for all ei lists	
emask	(1:18)	comm31	masks for bits 17,17-16,17-15...17-0
endbat		comm16	1*1 true if at end of batch(calc)
endrun		comm16	1*1 true if at end of run(calc)
eofile	(20)	comm18	char*40-name for external output file(read)
eonwrd	(20)	comm18	no. qm1 wrds per datum in eo action-use getparm,termrn
eorfl	(20)	comm18	byte-external output reschedule flag(1=on)
eorstr	(20)	comm18	external output start time for rescheduling
fbsize		parameter-no. of qm1 words in ms fault buffer	
ftitle		comm16	fault list title,read by colist,used act6 & schnop
gnewt		comm21	time for new event to be scheduled
gnmcon		comm21	number of action control records
gpa		comm21	general purpose pointer to action
gpe		comm21	general purpose pointer to event
gpevhd		comm21	ptr to head of event list, init by initfe
gpfrhd		comm21	ptr to head of free space list, init by initfe
gpmcon		comm21	pointer to action control block
gpmmas		comm21	pointer to master action control register
gpnewa		comm21	pointer to new action
gpnewe		comm21	pointer to newly allocated event
gsflag		comm21	stop flag(1=stop)
gstime		comm21	user-defined stop time
gtime		comm21	current time
illac1		parameter-constant for U. of Ill. lowest action code(50)	
illac2		parameter-constant for U. of Ill. highest action code(52)	
illin1		parameter-input unit for045-for U. of Illinois use only	
illout1		parameter-output unit for040-for U. of Illinois use only	
infin		parameter-infinity(2147483647=max no. for i*4)	
infltr		comm16	index no. of faulter device(read in)
ingnin		comm5	value to assign to output for devices with no inputs
initfl		comm5	initialization flag (0=user,1=computer)
iprclr		comm5	print-clear convention flag: 0(1=benign) 1(1=active)
ls	(0:31)	comm22	qm1 local store
lssup		comm22	highest ls loc to save on save file
maxconn		parameter-max no. of internal connections allowed	
maxgate		parameter-max no. of gates allowed	
maxgfl		parameter-max no. of gate faults per single time(30)	
maxmem		parameter-max. no. of target memories(30)	
maxnei		parameter-max no. ei sets (20)	
maxupch		parameter-maximum no. of user print choices (output formats)	
mbit0		parameter-mask for rightmost bit 0 (1)	
memadr	(30)	comm16	memory relocation constants
mnlbit		parameter-mask for no. leftover bits in action	
mnword		parameter-mask f. #qm1 wds/targer wd in 1st wd action('340'o)	
ms	(0:70000)	comm22	qm1 main store
mseile		comm17	last possible ei list entry(calculated)
mseili		comm17	loc in ms of first ei list(read)
mseobu		comm18	loc in ms of first eo buffer

Fortran Parameters & Common Variables, Sorted by Variable Name

Name	Dimension	Common Label	Description
mseole		comm18	not used
msexter		comm2	qm1 main store address for first external register
msfblk		comm16	ptr to ms fault blk(read in)
msnxfl		comm16	ptr to next op to be sched.,init by colist,inc in act6
mssup		comm22	highest ms loc to save on save file
mul1		parameter-	(17)
mul2		parameter-	(1)
mulnbi		parameter-	divisor to right-justify mnlbit (1)
mulnwo		parameter-	divisor to right-justify mnword (32)
nchange		comm28	no. of headers that changed this stack
nconnec		comm1	no. of connections, set by preproc
nexinp		comm17	actual number of ei sets for this batch(read)
nexoup		comm18	no. of external output sets
nextern		comm1	no. of external connections, set by preproc
ngates		comm1	no. of devices, set by initrn=neqn
ngfcon		comm16	no. of gate faults this stack(calc)
nheads		comm27	no. of devices to have headers printed
nomems		comm16	no. of rom and ram memories with relocation
nops		comm16	no. of ops in batch(calc)
nstack	(2)	comm4	number of items in stack i
nthead		parameter-	max no. of devices which can have headers printed
ntrace		comm24	no. of devices to be traced
ntri		comm5	number of devices with defined output values
numvops		parameter-	number of valid op codes(8)
nupcho		comm25	no. of user print choices(output formats)
opgtype		parameter-	gate fault op type(1)
oplifg		parameter-	op code for lift gate fault(5)
oplifm		parameter-	op code for lift memory fault(7)
opmtype		parameter-	memory fault op type(2)
opsbat		parameter-	op code for stop batch(1)
opsize	(15)	comm16	no. of words for corresponding op
opsrun		parameter-	op code for stop run(2)
opstg0		parameter-	op code for stick gate at 0(3)
opstg1		parameter-	op code for stick gate at 1(4)
opstm		parameter-	op code for fault memory(6)
pcslow		comm22	parameter-low dimension for control store (0)
pcsup		comm22	parameter-high dimension for control store (20000)
pfltcon		comm16	ptr to next fault connection(calc)
plslow		comm22	parameter-low dimension for local store (0)
plsup		comm22	parameter-high dimension for local store (37)
pmslow		comm22	parameter-low dimension for main store (0)
pmsup		comm22	parameter-high dimension for main store (70000)
prloc	(3,2)	co8	low & high address for cs,ms,ls for output
prsw	(50)	co8	user print option switched, 0=off, 1=on
prtime	(30)	co8	print window 1=start,2=stop,3=delta
prtisw	(10)	co8	1=print window flag(1=on)
runtitle	(10)	comm1	title for run, read in getparm from eopts file
s		comm4	current stack number (1 or 2)
savg		comm4	real-average number of items on stack
sbar		comm4	non-current stack number (1 if s=2, 2 if s=1)
smax		comm4	maximum number of items on stack
smin		comm4	minimum number of items on stack

Fortran Parameters & Common Variables, Sorted by Variable Name

Name	Dimension	Common Label	Description
stack	(2,500)	comm4	current & new stacks holding indices of stack devices
sw	(20)	comm29	
sw1		comm29	logical-true if prsw(3) and prsw(4) on(check hdrs)
sw2-sw20		comm29	logical-switches(not used)
tema71			parameter-mask template for action 7, word 1('034000'o)
tema81			parameter-template, action 8, word 1, no rescheduling
temb81			parameter-template, action 8, word 1, resched on
timebuf		comm14	character*8-current time for output
timesiz		comm16	no. of qm1 wds to hold time(read in)
title	(10)	comm1	i*4-title for output,read from opts file by initrn
triang	(4000)	comm5	indices of all devices with output value defined
tyand			parameter-device type for and gate (1)
tyff			parameter-device type for ff (0)
tynand			parameter-device type for nand gate (2)
tynor			parameter-device type for nor gate (4)
tynot			parameter-device type for not gate (5)
tynxor			parameter-device type for nxor gate (7)
tyor			parameter-device type for or gate (3)
tyxor			parameter-device type for xor gate (6)
uin			parameter-input unit for008-matrix(bdxhd2s.dat)
uin0			parameter-input unit for007-device comments(bdxcomm.dat)
uin1			parameter-input unit for010-target memories(bdxmems.dat)
uin10			parameter-input unit for025-
uin2			parameter-input unit for011-user options(bdxopts.dat)
uin3			parameter-input unit for015-
uin4			parameter-input unit for016-
uin5			parameter-input unit for017-
uin6			parameter-input unit for009-
uin7			parameter-input unit for022-
uin8			parameter-input unit for023-
uin9			parameter-input unit for024-
uout			parameter-output unit for014-output file(bdxout.dat)
uout0			parameter-output unit for018-
uout1			parameter-output unit for012-alpha device list(bdsalph.dat)
uout10			parameter-output unit for030-
uout2			parameter-output unit for013-device name list(bdxnam.dat)
uout3			parameter-output unit for019-matrix for qm1(bdxmat.dat)
uout4			parameter-output unit for020-binary checkingfile(bdxcheck.dat)
uout5			parameter-output unit for021-
uout6			parameter-output unit for026-
uout7			parameter-output unit for027-
uout8			parameter-output unit for028-
uout9			parameter-output unit for029-
upcsms	(15)	comm25	user print choice memory type(0=cs,1=ms,2=ls)
upform	(15)	comm25	character*80-user print choice format incl. ()
uploc1	(15)	comm25	user print choice low mem address to output
uploc2	(15)	comm25	user print choice high mem address to output
uptitle	(15)	comm25	character*20-user print choice title to output
vophigh			parameter-highest valid user op code(7)
voplow			parameter-lowest valid user op code(1)
xaddres	(4000)	comm1	qm1 control store address for header for device i
xchange	(4000)	comm28	byte- 0 if x didn't change this stack, 1 if did

Fortran Parameters & Common Variables, Sorted by Variable Name

Name	Dimension	Common Label	Description
xchid	(1000)	comm28	index nos. of the headers that changed this stack
xclass	(4000)	comm8	device class(gate,flip-flop,or tri-state)
xconn	(10000)	comm1	full address for internal connection
xcount	(4000)	comm11	initial value of "count" for each device
xdis	(4000)	comm8	disconnected output value for tri-states
xebit	(4000)	comm3	bit no. for external (not needed after init)
xecsms	(4000)	comm3	type of external(0=cs,1=ms,2=ls)(not needed afterinit)
xehigh	(4000)	comm2	index to last external data structure for each device
xei	(4000)	comm2	1=external complemented,0=not (not needed after init)
xelink	(4000)	comm2	external link word
xelow	(4000)	comm2	index to first external data structure for each device
xereg	(4000)	comm3	external register no. (cs,msls) (not needed afterinit)
xeval	(4000)	comm5	external value for device
xew	(4000)	comm2	qm1 cs or ms address of external
xffval	(4000)	comm5	"PCTLJK" values for flip-flop
xhdr	(4000)	comm1	header for each device i
xhead	(4000)	comm5	pts to 1st entry in conn list(this device is destin.)
xheadt	(500)	comm27	indexes of devices to have headers printed
xhigh	(4000)	comm1	index to connection list for last conn for device i
xival	(4000)	comm5	internal value for device
xlink	(10000)	comm1	first word of internal connector record
xlow	(4000)	comm1	index to connection list for first conn for device i
xname	(4000)	co6	character*20-device names, set by getdevn
xnudef	(4000)	comm5	no. of undefined inputs for this device
xpval	(4000)	comm5	predefined output value for device
xr	(4000)	comm8	R value
xstack	(4000)	comm11	stack flag for device(0=not on,1=is on 1st stack)
xtrace	(4000)	comm24	byte-trace flag(0=dont print output changes,1=do)
xtype	(4000)	comm8	device type(fiip-flop,and,nand,or,etc.)
xu	(4000)	comm8	U value
xvalue	(4000)	comm8	output value for device
zfullw	(10000)	comm26	byte
zptr	(10000)	comm1	the index of the dest device for this connection

Flip-Flop Decision Table for QM-1 Version

I N P U T S														O U T P U T S			
Input	* Value before Complementing									*				Action Taken		Branch to:	
	P	C	T	L	J	K	V^c	VE	R	U	e	V_{DIS}	V_z	Δ_3	V	Δ_3	Check[1]	Skip[2]
ΔP	0	0						0							0	1		x
	0	0						1							0	1	x	
	1	0						0							1	1	x	
	1	0						1							1	1		x
	1	1						0							1	1	x	
	1	1						1							1	1		x
ΔC	0	0						0							1	1	x	
	0	0						1							1	1		x
	0	1						0							1	1	x	
	0	1						1							1	1		x
	1	1						0							0	1		x
	1	1						1							0	1	x	
ΔT	1	1	1		0	1		1									x	
	1	1	1		1	0		0									x	
	1	1	1		1	1			0								x	
	1	1	1		1	1	0	0	1								x	
	1	1	1		1	1	1	1	1								x	
ΔL	1	1		1				0					1	1			x	
	1	1		1				1					0	1			x	
	1	1		1			0	0						0	0	1		x
	1	1		1			1	1						0	1	1		x
	1	1		1			0	1						0	0	1	x	
	1	1		1			1	0						0	1	1	x	
	1	1		0	0	1									0			x
	1	1		0	1	0									1			x
	1	1		0	1	1							0		1			x
	1	1		0	1	1							1		0			x
	1	1		0	1	1	1		1						1			x
	1	1		0	1	1	0		1						0			x
ΔJ or ΔK or (ΔJ & ΔK)	1	1		1						1						0		x
Δen-able							1	0			0						x	
							0	1			0						x	
								0			1	1					x	
								1			1	0					x	

Note: Any case not in table represents no action taken, and branch to "Skip".
For input columns, blanks are "don't care" conditions.
[1] Check whether device should be enqueued, and continue processing.
[2] Skip enqueueing check, and just continue processing.

QM-1 Emulator Files

File Name	Operating System	File Type	File Description
Nanocode Emulator:			
BBEMP1V1:S	Nova	Source	Emulator Nanocode, Part 1
BBEMP1V1	Nova	Binary	"
BBBNBIN	Nova	Definition	"
BBEMP2V1:S	Nova	Source	Emulator Nanocode, Part 2
BBEMP2V1	Nova	Binary	"
BBEMP3V1:S	Nova	Source	Emulator Nanocode, Part 3
BBEMP3V1	Nova	Binary	"
MSNANON:S	Nova	Source	Main Store Extend. Address. Nanoword
BBNANOE:S	Easy	Source	"
MSNANON:B	Nova	Binary	"
MSNANON:M	Nova	Mapped	"
Microcode Driver:			
BBDSNOVA:S	Nova	Source	Symbol Definitions
BBDSEASY:S	Easy	Source	"
BBGDNOVA:S	Nova	Source	Global Data Definitions
BBGDEASY:S	Easy	Source	"
BBGD:B	Nova	Binary	"
BBDRNOVA:S	Nova	Source	Driver Module
BBDREASY:S	Easy	Source	"
BBDR:B	Nova	Binary	"
BBA1NOVA:S	Nova	Source	Action Modules Set 1
BBA1EASY:S	Easy	Source	"
BBA1:B	Nova	Binary	"
BBA2EASY:S	Easy	Source	"
BBA2:B	Nova	Binary	"
BBESNOVA:S	Nova	Source	Subroutine Modules Set 1
BBESEASY:S	Easy	Source	"
BBES:B	Nova	Binary	"
BBEMNOVA:S	Nova	Source	Subroutine Modules Set 2
BBEMEASY:S	Easy	Source	"
BBEM:B	Nova	Binary	"
BBEENOVA:S	Nova	Source	Subroutine Modules Set 3
BBEE:B	Nova	Binary	"
BBUTNOVA:S	Nova	Source	Utility Modules for Driver
BBUTEASY:S	Easy	Source	"
BBUT:B	Nova	Binary	"
BBIONOVA:S	Nova	Source	I/O Modules for Driver
BBIOEASY:S	Easy	Source	"
BBIO:B	Nova	Binary	"
BBECOMPILE	Nova	Execute	Assemble all Driver Microcode Programs
BBECONVERT	Easy	Execute	Convert all Driver Source Microcode Programs from Easy to Nova
BBCGD	Nova	Execute	Assemble BBGDNOVA:S
BBCDR	Nova	Execute	Assemble BBDRNOVA:S
BBCA1	Nova	Execute	Assemble BBA1NOVA:S
BBCA2	Nova	Execute	Assemble BBA2NOVA:S
BBCES	Nova	Execute	Assemble BBESNOVA:S
BBCEM	Nova	Execute	Assemble BBEMNOVA:S
BBCEE	Nova	Execute	Assemble BBEENOVA:S
BBCUT	Nova	Execute	Assemble BBUTNOVA:S
BBCIO	Nova	Execute	Assemble BBIONOVA:S

QM-1 Utility Files

File Name	Operating System	File Type	File Description
EOTODISK	Easy	Execute	Write External Outputs from Memory to Disk
TVAXQM1	Easy	Execute	Transfer Disk File from Vax to QM1 Test VAXQM1 BBVAXQM1
TQM1VAXI	Easy	Execute	Transfer Disk File from QM1 to Vax Test MV BBMV
*:E	Nova	Execute	Generate Executable File for Target Machine *
R*	Nova	Loadable	Executable file for Target Machine *
WXMDISK:S	Easy	Source	Write External Outputs from Memory to Disk
WXMDISK:B	Easy	Binary	"

QM-1 Files for Transfers With Vax

File Name	Operating System	File Type	File Description
		Vax to QM1 and QM1 to Vax	
BBGETPIO	Easy	Source	Get next record from pio interface i.e., wait till ready, and determine no. of records read
BBPUTPIO	Easy	Source	Put next record out over pio interface
BBRACKNAK	Easy	Source	Receive acknowledge/no ack. code
BBCACKNAK	Easy	Source	Calculate whether ack or nak code recvd
BBVAXIN	Easy	Source	Get next record from pio(lowest level)
BBREST	Easy	Source	Miscellaneous low-level modules for interface
BBVAXOUT	Easy	Source	Put next record to pio(lowest level) from a character array
BBSACKNAK	Easy	Source	Send ack/nak code over pio
		QM1 to Vax	
BBIVAXIN	Easy	Source	Get next record from pio(lowest level) (as an integer array)
BBACWRTRAN	Easy	Source	Write activity on transaction log file
BBRDATAF	Easy	Source	Read next record from QM1 disk file
BBEXITARUN	Easy	Source	Close disk files and exit the run
BBIVAXOUT	Easy	Source	Put next record to pio(lowest level) from an integer array
BBACCOMPAR	Easy	Source	Compare record sent to record received
BBIPUTPIO	Easy	Source	Put next record out over pio, from an integer array
BBIGETPIO	Easy	Source	Get next record from pio interface (as an integer array)
BBRINTF	Easy	Source	Read next integer value from QM1 disk file
BBMVMAIN:S	Easy	Source	Main Program
BBMV	Easy	Executable	Main Program
BBMVSEND:S	Easy	Source	Send record from QM1 to Vax
TQM1VAXI	Easy	Execute	Transfer disk file from QM1 to Vax
		Vax to QM1	
BBPVAXQM1	Easy	Source	Main Program
BBVAXQM1	Easy	Executable	Main Program
BBWRDATAF	Easy	Source	Write record received to QM1 disk file
BBWRTRANF	Easy	Source	Write activity on transaction log file
BBEXITRUN	Easy	Source	Close disk files and exit the run
TVAXQM1	Easy	Execute	Transfer disk file from Vax to QM1

Device Header Layouts

Gate or Tri-State

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	V_E	V^C	$V^{\overline{C}}$	e	Δ_3	F	$\Delta_1{}^C$	$\Delta_2{}^C$	E	$\Delta_1{}^{\overline{C}}$	$\Delta_2{}^{\overline{C}}$	C	O	U	N	T	

Flip-Flop

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	V_E	V^C	$V^{\overline{C}}$	e	Δ_3	F	$\Delta_1{}^C$	$\Delta_2{}^C$	E	$\Delta_1{}^{\overline{C}}$	$\Delta_2{}^{\overline{C}}$	P	C	T	L	J	K

Legends for Header Words

Name	Description	Coding	Initial Value
V	Internal Value	0 = low, 1 = high	
V_E	External Value	0 = low, 1 = high	
V^c	External Value (updated only while processing cbar stack)	0 = low, 1 = high	
$V^{\bar{c}}$	External Value (updated only while processing c stack)	0 = low, 1 = high	
e	Enabled/Disabled Flag	0 = disabled, 1 = enabled	1
Δ_3	Special Indeterminant Status for Flip-flops (used internally by algorithm)	0=input to flip-flop changed while clock high 1=input to flip-flop did not change while clock high	1
F	Failure Flag (not implemented to date)	0=not failed 1=failed	
$\Delta_1{}^c$	Stack Bypass Flag - used internally by algorithm to indicate whether external value of stack item has changed an even or odd number of times during previous time period. This flag is examined during 1st, 3rd, 5th, etc. time periods and set during 2nd, 4th, 6th etc. time periods.	0=value has changed an odd number of times 1=value has changed an even number of times; thus this device can be bypassed on this processing of c stack	1
$\Delta_2{}^c$	Stack Flag - indicates whether device has been enqueued on stack. This flag is examined during 1st, 3rd, etc. time periods and is set during 2nd, 4th, etc. time periods.	0=has been enqueued 1=has not been enqueued	0=this device is on first stack 1=this device is not on first stack
	External Connections Flag	0=the output of this device does not feed into any external registers 1=the output of this device feeds into one or more	

Legends for Header Words

Name	Description	Coding	Initial Value
		external registers	
Δ_1^{τ}	Stack Bypass Flag - used internally by algorithm to indicate whether external value of stack item has changed an even or odd number of times during previous time period. This flag is examined during 2nd, 4th, 6th, etc. time periods and set during 1st, 3rd, 5th, etc. time periods.	0=value has changed an odd number of times 1=value has changed an even number of times;	1
Δ_2^{τ}	Stack Flag - indicates whether device has already been enqueued on stack. This flag is examined during 2nd, 4th, etc., time periods and set during 1st, 3rd, etc. time periods.	0=has been enqueued 1=has not been enqueued	1
Count	A six-bit count which is used to determine when the output value of a gate changes. See Section 4.4.3.1	2's complement notation	see Figure 14
P	The current value on the P input to the flip-flop	0 = low, 1 = high	
C	The current value on the C input to the flip-flop	0 = low, 1 = high	
T	The current value on the T input to the flip-flop	0 = low, 1 = high	
L	The current value on the L input to the flip-flop	0 = low, 1 = high	
J	The current value on the J input to the flip-flop	0 = low, 1 = high	
K	The current value on the K input to the flip-flop	0 = low, 1 = high	

Internal Connector Layouts

Connector to Gate, Type 1

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
← DISPLACEMENT →															i	t	s = 0

Connector to Gate, Type 2

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
← NOT USED →														n = 0	i	t	s = 1
← FULL ADDRESS →																	

Connector to Flip-Flop

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Connection Type								R	U					n = 1	i	t	s = 1
← FULL ADDRESS →																	

Connector to Tri-State Enable

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Connection Type							V_{DIS}							n = 1	i	t	s = 1
← FULL ADDRESS →																	

Legends for Internal Connectors

Name	Description	Coding	Initial Value
Displacement	Displacement of destination device from source device	Let x=address of source header word Let z=address of destination header word. Then Displacement = z - x + 2^{14} (in 2's complement notation)	
i	Inversion Flag	0 = the value on this connection is inverted going into the destination device 1 = the value on this connection is not inverted going into the destination device	
t	Sentinel for internal connections	0 = this is not the last internal connection to a destination device from this source device 1 = this is the last internal connection to a destination device from this source device	
s	Type of representation of destination address	0 = displacement of destination from source 1 = full 18-bit address of destination device(an additional word must follow)	0 for gate connector type 1 1 for all others
n	General description of the connection into the destination device	not used for connection into a gate or tri-state input*(type 1) 0 = connection into a gate or tri-state input*(type 2) 1 = connection into a flip-flop or tri-state enable**	0 for connection into gate or tri-state input*(type 2) 1 for connection into flip-flop or tri-state enable**
Full Address	Full 18-bit absolute control store address of header word of destination device		
Connection Type	Specific description of the connection into the destination device	041_8 = enable line of tri-state 021_8 = P connection into flip-flop	

Legends for Internal Connectors

Name	Description	Coding	Initial Value
		033_8 = C connection into flip-flop 061_8 = T connection into flip-flop 113_8 = L connection into flip-flop 141_8 = J connection into flip-flop 161_8 = K connection into flip-flop 121_8 = D(J&K)*** connection into flip-flop note: in each case, the first octal digit is one bit, the second and third octal digits are three bits each.	
R	Indeterminant Flag 1	1 = Either a) this is a T connection and clock's going low while J and K are high causes indeterminant output, or b) this is an L connection and clock's going high while J and K are high causes an indeterminant output 0 = All other cases	
U	Indeterminant Flag 2	1 = This is a J or K or D connection and J and/or K changing while L, P, and C are high causes an indeterminant output. 0 = All other cases	
V_{DIS}	Disabled Output Value	0 = When device is disabled, its output value will remain at 0 1 = When device is disabled, its output value will remain at 1	

* tri-state input is any input line to a tri-state except the enable line
** tri-state enable is the line going into a tri-state which controls whether it is enabled or disabled
*** D input is one where the K input is the complement of the J input

External Connector Layouts

External Connector to Local Store Register (not implemented)

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rflag = 0	i	t						Register Number					Bit Position				

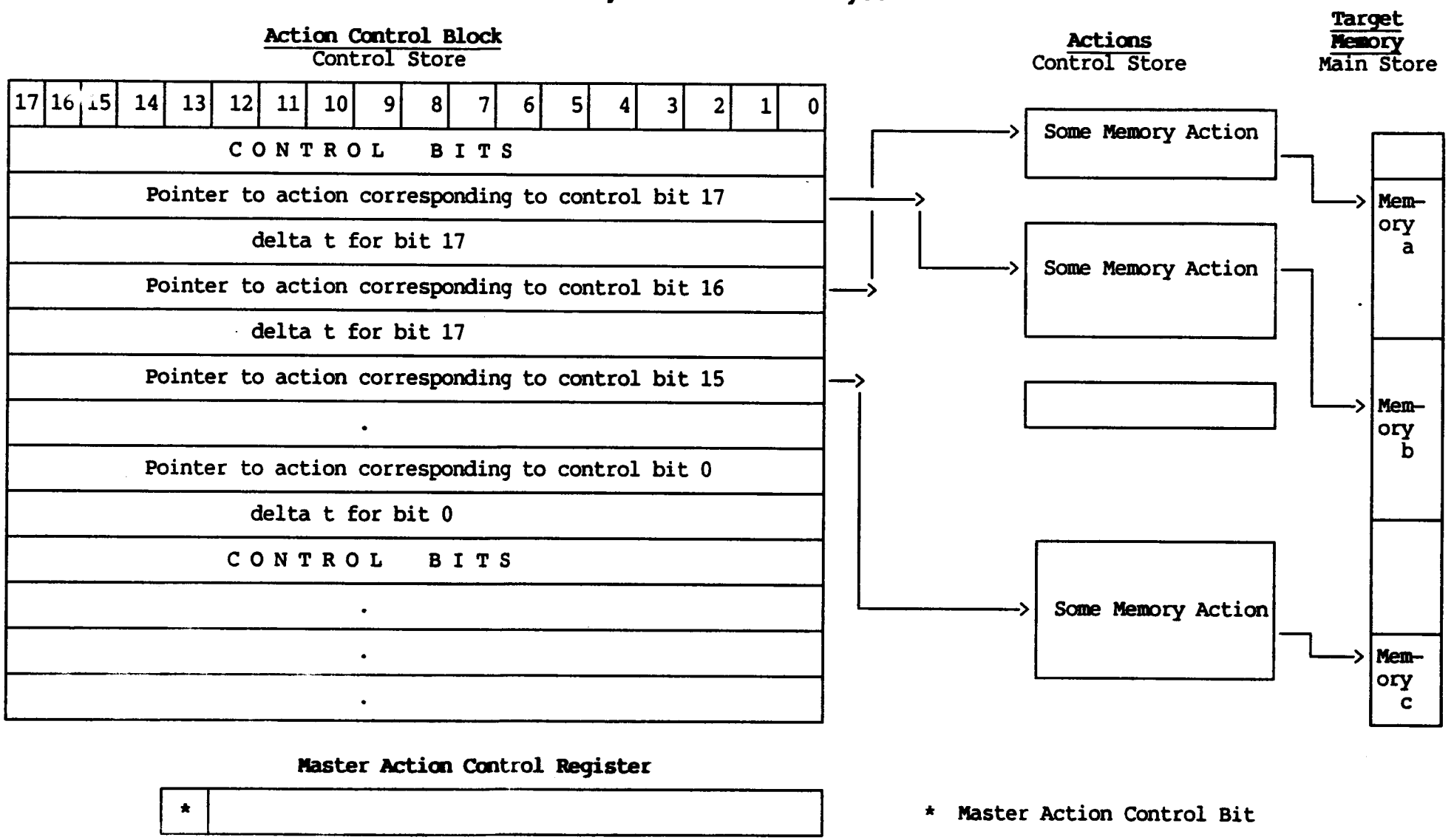
External Connector to Control Store or Main Store Register

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rflag = 1	i	t	Mem-Type	AC Cntrl									Bit Position				
← FULL ADDRESS →																	

Legends for External Connectors

Name	Description	Coding
RFLAG	External Register Flag	0 = the external register is represented by a qm1 local store register 1 = the external register is represented by a qm1 control store or main store memory word.
i	Inversion Flag	0 = the value on this connection is inverted going into the destination device 1 = the value on this connection is not inverted going into the destination device
t	Sentinel for external connections	0 = this is not the last external connection from this source device 1 = this is the last external connection from this source device
Register Number	The number of the external register into which the connection feeds	The register number relative to the first register in the block of control store or main store, where the first register is numbered 1.
Bit Position	The number of the external register into which the connection feeds	The bits are numbered from least significant to most significant in the order 0 to 17.
Mem-Type	Type of memory for external register	0 = control store 1 = main store
AC Cntrl	Action Control Register Flag	0 = this external register is not an action control register 1 = this external register is an action control register.
Full Address	Full 18-bit absolute qm1 address of external register	

Memory Data Structures Layout



Emulated Memory Layout

(Two options for memory layout in QM-1 of target word with n bits)

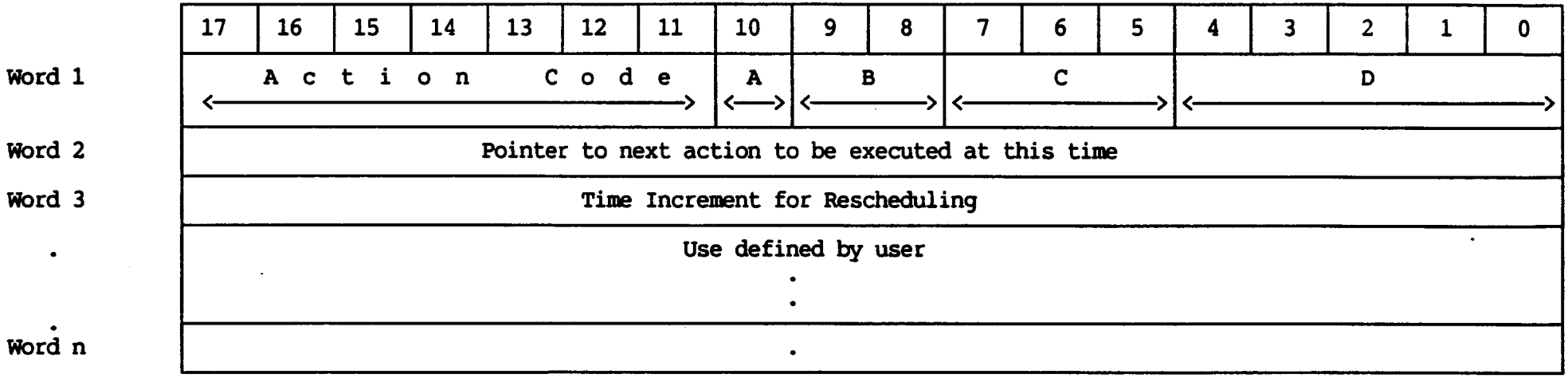
Right Justified

17	16		2	1	0
← N O T U S E D →			target bit n	target bit n-1	target bit n-2
target bit n-3	target bit n-4	. . . . .	.	.	.
		:			
		. . . . .	target bit 2	target bit 1	target bit 0

Left Justified

17	16		2	1	0
target bit n	target bit n-1		target bit n-15	target bit n-16	target bit n-17
target bit n-18	target bit n-19	. . . . .	.	.	.
		:			
	. . . .	target bit 2 / target bit 1 / target bit 0	← N O T U S E D →		

General Action Layout



A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qml 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

Write Memory Action (code = 2)

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	Action Code 0	0	0	0	0	1	0	A <→>		B <→>			C <→>			D <→>		
Word 2	Pointer to next action to be executed at this time																	
Word 3	Time Increment for Rescheduling																	
Word 4	Absolute QM-1 Main store address of first word of target memory (i.e., relocation constant)																	
Word 5	Absolute QM-1 Control store address of emulated address register																	
Word 6	Absolute QM-1 Control store address of emulated data register																	
Word 7	Valid address of first word word of target memory																	
Word 8	Valid address of last word of target memory																	

A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qm1 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

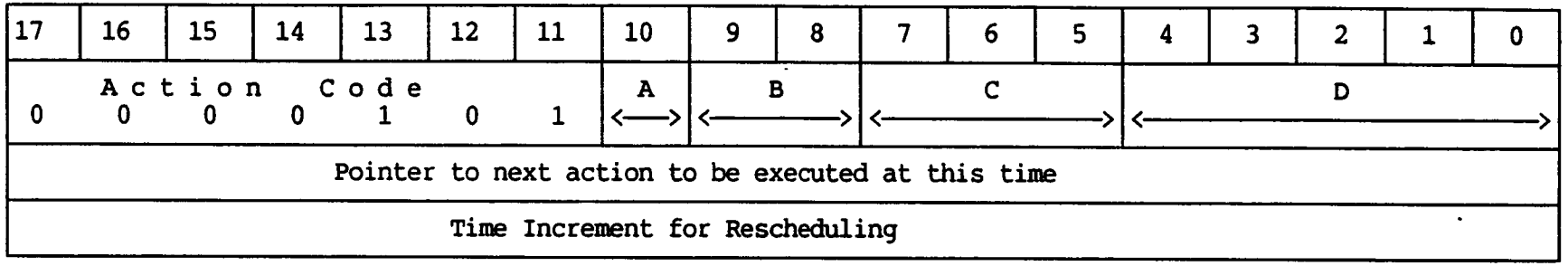
Read Memory Action (code = 3)

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	Action Code 0 0 0 0 0 1 1							A <→>	B <→>		C <→>			D <→>				
Word 2	Pointer to next action to be executed at this time																	
Word 3	Time Increment for Rescheduling																	
Word 4	Absolute QM-1 Main store address of first word of target memory (i.e., relocation constant)																	
Word 5	Absolute QM-1 Control store address of emulated address register																	
Word 6	Absolute QM-1 Control store address of emulated data register																	
Word 7	Valid address of first word of target memory																	
Word 8	Valid address of last word of target memory																	
Words 9-26	Device Identifier of device into which the corresponding memory bit feeds																	

Note: Word 9 corresponds to bit 17 of word 1; Word 10 corresponds to bit 16 of word 1, etc..

A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qm1 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

Stop Action (code = 5)



A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qm1 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

Operations Action (code = 6)

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	Action Code							A	B		C			D				
	0	0	0	0	1	1	0	<⟷>	<⟷>		<⟷>			<⟷>				
Word 2	Pointer to next action to be executed at this time																	
Word 3	Time Increment for Rescheduling																	
Word 4	Control store address of any read or write action for memory 1																	
Word 5	Control store address of any read or write action for memory 2																	
	.																	
	.																	
Word 3 + M*	Control store address of any read or write action for memory M*																	

* M = the number of memories being emulated at the functional level
One word is required for each target memory. If there are no target memories, then word 3 is the last word of the action.

A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qm1 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

External Input Action (code = 7)

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	ACTION CODE							A	B		C			D				
	0	0	0	0	1	1	1	<—>	<——>		<——>			<——>				
Word 2	Pointer to next action to be executed at this time																	
Word 3	Time Increment for Rescheduling																	
Word 4	Main store address of first word of external input list																	
Word 5	Control store address of address register																	
Word 6	Control store address of data register																	
Word 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Word 8	(Total number of external input list items for this set) - 1																	
Word 9	Main store address of next external input to be scheduled																	
Word 10	Main store address of last external input to be scheduled																	
Word 11	address of device into which most significant bit of external input feeds																	
Word 12	address of device into which next bit feeds																	
Word 13	"																	
Word 8+N*	address of device into which least significant bit of external input feeds																	

* N = number of bits in this external input set

A = Scheduled Switch: (0=not scheduled; 1=scheduled)

B = Reschedule Flag: (00=no rescheduling; 01=reschedule onece; 10=reschedule every t period)

C = Number of qm1 18-bit words needed to represent one target word

D = 18(decimal), if target word is right justified

= number of bits used in last word if target word is left justified

External Output Action (code = 8)

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Word 1	Action Code							A	B		C			D				
	0	0	0	1	0	0	0	⟷	⟷		⟷			⟷				
Word 2	Pointer to next action to be executed at this time																	
Word 3	Time Increment for Rescheduling																	
Word 4	Main store address of first word of external output buffer																	
Word 5	Control store address of address register																	
Word 6	Control store address of data register																	
Word 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Word 8	(Maximum number of items which can be outputted for this set) - 1																	

A = Scheduled Switch: (0=not scheduled; 1=scheduled)
B = Reschedule Flag : (00=no rescheduling; 01=reschedule once; 10=reschedule every t period)
C = Number of qm1 18-bit words needed to represent one target word
D = 18(decimal), if target word is right justified
= number of bits used in last word, if target word is left justified

Example of Q Flip-Flop and Qbar Flip-Flop Pair

Following are the netlist entries and printout in the connections list for the pair, FFQ1'U33 and FFQ1U33:

Netlist Entries

```
>FFQ1'U33          1  CLASS=  2 TYPE =  0 VALUE=  1 NICON=  3 NECON=  1
 FFQ1'U33          2  FFVAL= 72
 FFQ1'U33          3  R    =  0 U    =  0
 FFQ1'U33         10  ZNAME=    GO1U7              REVER=  0 CONNT=  0
 FFQ1'U33         11  ZNAME=    GO2U7              REVER=  0 CONNT=  0
 FFQ1'U33         12  ZNAME=    GO1U9              REVER=  0 CONNT=  0
 FFQ1'U33         13  CSMSF=  0 REGNO= 19 BITNO= 10 REVER=  0
>FFQ1U33           1  CLASS=  2 TYPE =  0 VALUE=  0 NICON=  5 NECON=  2
 FFQ1U33           2  FFVAL= 72
 FFQ1U33           3  R    =  0 U    =  0
 FFQ1U33          10  ZNAME=    GO3U7              REVER=  0 CONNT=  0
 FFQ1U33          11  ZNAME=    GO2U8              REVER=  0 CONNT=  0
 FFQ1U33          12  ZNAME=    GO2U9              REVER=  0 CONNT=  0
 FFQ1U33          13  ZNAME=    GO3U9              REVER=  0 CONNT=  0
 FFQ1U33          14  ZNAME=    TSY1U35            REVER=  0 CONNT=  0
 FFQ1U33          15  CSMSF=  0 REGNO= 19 BITNO=  9 REVER=  0
 FFQ1U33          16  CSMSF=  0 REGNO= 23 BITNO= 12 REVER=  0
>GCLOCKU33         1  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  8 NECON=  1
 GCLOCKU33        10  ZNAME=    FFQ1U33            REVER=  1 CONNT= -3
 GCLOCKU33        11  ZNAME=    FFQ1'U33           REVER=  1 CONNT= -3
 GCLOCKU33        12  ZNAME=    FFQ2U33            REVER=  1 CONNT= -3
 GCLOCKU33        13  ZNAME=    FFQ2'U33           REVER=  1 CONNT= -3
 GCLOCKU33        14  ZNAME=    FFQ3U33            REVER=  1 CONNT= -3
 GCLOCKU33        15  ZNAME=    FFQ3'U33           REVER=  1 CONNT= -3
 GCLOCKU33        16  ZNAME=    FFQ4U33            REVER=  1 CONNT= -3
 GCLOCKU33        17  ZNAME=    FFQ4'U33           REVER=  1 CONNT= -3
 GCLOCKU33        18  CSMSF=  0 REGNO= 19 BITNO=  7 REVER=  0
>GCLRU33           1  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  8 NECON=  1
 GCLRU33          10  ZNAME=    FFQ1U33            REVER=  1 CONNT=  2
 GCLRU33          11  ZNAME=    FFQ1'U33           REVER=  1 CONNT=  1
 GCLRU33          12  ZNAME=    FFQ2U33            REVER=  1 CONNT=  2
 GCLRU33          13  ZNAME=    FFQ2'U33           REVER=  1 CONNT=  1
 GCLRU33          14  ZNAME=    FFQ3U33            REVER=  1 CONNT=  2
 GCLRU33          15  ZNAME=    FFQ3'U33           REVER=  1 CONNT=  1
```

Example of Q Flip-Flop and Qbar Flip-Flop Pair

```
 GCLRU33          16  ZNAME=    FFQ4U33              REVER=  1 CONNT=  2
 GCLRU33          17  ZNAME=    FFQ4'U33             REVER=  1 CONNT=  1
 GCLRU33          18  CSMSF=  0 REGNO= 19 BITNO=  8 REVER=  0
>GO4U32            1  CLASS=  1 TYPE =  3 VALUE=  9 NICON=  2 NECON=  1
 GO4U32           10  ZNAME=    FFQ1U33              REVER=  0 CONNT=  7
 GO4U32           11  ZNAME=    FFQ1'U33             REVER=  1 CONNT=  7
 GO4U32           12  CSMSF=  0 REGNO= 14 BITNO= 16 REVER=  0
```

Connections List Printout

```
 9 FFQ1'U33          FLIP FLOP  1   246 GO4U32          0 OR    GATE  D  REVERSED
                                    58 GCLRU33          1 NOT   GATE  P  REVERSED
                                    54 GCLOCKU33        0 NOT   GATE  T  REVERSED  UP-EDGE
16 FFQ1U33           FLIP FLOP  0   246 GO4U32          1 OR    GATE  D
                                    58 GCLRU33          1 NOT   GATE  C  REVERSED
                                    54 GCLOCKU33        0 NOT   GATE  T  REVERSED  UP-EDGE
```

Appendix B

Sample Initialization Text Output File

Sample Initialization Text Output File

ORIGINAL PAGE IS
OF POOR QUALITY

*** DIAGNOSTIC EMULATOR INITIALIZATION ***   DATE: 8-JUN-87   TIME:11:10:38

TARGET MACHINE: 3-bit counter with not,xor,or,and gates

DEVICES WITH DEFINED OUTPUT VALUES

<--------D e v i c e------> #(decimal)	Name	Internal Value	External Value	PCTLJK	Predefined Output	Calculated Output
1	X1	1	1	0	1	1
2	X2	0	0	0	0	0
3	X3	0	0	0	*********	0
4	X4	0	0	0	*********	0
5	X5	0	0	0	0	0
6	X6	0	0	0	*********	0
7	X7	0	0	0	*********	0
8	X8	0	0	0	0	0
9	ZZZFAULTER	0	0	0	0	0
10	ZZZZDUMMY	0	0	0	0	0

NUMBER OF DEFINED DEVICES = 10

C O N N E C T I O N S   L I S T

<------D e s t i n a t i o n---------->					<-------------S o u r c e------------->					Connect	Reversal	Edge
D e v i c e					D e v i c e					Type	Flag	Trigger
#	Device Name	Type	Class	Output	#	Device Name	Output	Type	Class			
1	X1	NOT	GATE	0	2	X2	0	OR	GATE			
2	X2	OR	GATE	0	1	X1	0	NOT	GATE			
3	X3	OR	GATE	0	2	X2	0	OR	GATE			
4	X4	OR	GATE	0	5	X5	0	XOR	GATE			
5	X5	XOR	GATE	0	4	X4	0	OR	GATE			
					3	X3	0	OR	GATE			
6	X6	AND	GATE	0	5	X5	0	XOR	GATE			
					2	X2	0	OR	GATE			
7	X7	OR	GATE	0	8	X8	0	XOR	GATE			
8	X8	XOR	GATE	0	7	X7	0	OR	GATE			
					6	X6	0	AND	GATE			
9	ZZZFAULTER	AND	GATE	0								
10	ZZZZDUMMY	AND	GATE	0	10	ZZZZDUMMY	0	AND	GATE			
					9	ZZZFAULTER	0	AND	GATE	TS ENABLE		

```
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE
9 ZZZFAULTER     0 AND     GATE     TS ENABLE

          I N I T I A L    D E V I C E    H E A D E R S (at time    0)
<---------------D e v i c e---------------> Address(octal)  Header(octal)
#(decimal) #(octal)    Name

     1          1     X1                 20000       771700
     2          2     X2                 20004        33700
     3          3     X3                 20014        33700
     4          4     X4                 20020        33700
     5          5     X5                 20024        33777
     6          6     X6                 20033        33776
     7          7     X7                 20037        33700
     8          8     X8                 20043        33777
     9          9     ZZZFAULTER         20051        33777
    10         10     ZZZZDUMMY          20150        33741

Address                   ----Control Store Dump----
  4400     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  4420     0     0     0     0     0     0     0     0     0  2532     0  2500     0  2570     0  2400

Address                   ----Main Store Dump----
   100     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
   120     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

Address                   ----Local Store Dump----
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
    20     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

Appendix C

Sample Netlist File

Sample Netlist File

```
>FPTS1F00    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS1F00    0  ZNAME=    TS1G20              REVER=  0 CONNT=  0
>FPTS1F01    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  2 NECON=  0
 FPTS1F01    0  ZNAME=    TS1G03              REVER=  0 CONNT=  0
 FPTS1F01    0  ZNAME=    TS1G06              REVER=  0 CONNT=  0
>FPTS1F02    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS1F02    0  ZNAME=    TS1G02              REVER=  0 CONNT=  0
>FPTS1F03    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  2 NECON=  0
 FPTS1F03    0  ZNAME=    TS2G43              REVER=  1 CONNT=  0
 FPTS1F03    0  ZNAME=    TS1G20              REVER=  0 CONNT=  0
>FPTS1F04    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  2 NECON=  0
 FPTS1F04    0  ZNAME=    TS2G45              REVER=  1 CONNT=  0
 FPTS1F04    0  ZNAME=    TS1G20              REVER=  0 CONNT=  0
>FPTS1F05    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  2 NECON=  0
 FPTS1F05    0  ZNAME=    TS2G04              REVER=  1 CONNT=  0
 FPTS1F05    0  ZNAME=    TS1F00              REVER=  0 CONNT= -3
>FPTS1F06    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS1F06    0  ZNAME=    TS1G20              REVER=  0 CONNT=  0
>FPTS1F07    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS1F07    0  ZNAME=    TS1G20              REVER=  0 CONNT=  0
>FPTS2F00    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F00    0  ZNAME=    TS2G09              REVER=  0 CONNT=  0
 FPTS2F00    0  ZNAME=    TS2G10              REVER=  0 CONNT=  0
 FPTS2F00    0  ZNAME=    TS2G11              REVER=  0 CONNT=  0
>FPTS2F01    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F01    0  ZNAME=    TS2G14              REVER=  0 CONNT=  0
 FPTS2F01    0  ZNAME=    TS2G10              REVER=  0 CONNT=  0
 FPTS2F01    0  ZNAME=    TS2G12              REVER=  0 CONNT=  0
>FPTS2F02    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F02    0  ZNAME=    TS2G13              REVER=  0 CONNT=  0
 FPTS2F02    0  ZNAME=    TS2G09              REVER=  0 CONNT=  0
 FPTS2F02    0  ZNAME=    TS2G12              REVER=  0 CONNT=  0
>FPTS2F03    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F03    0  ZNAME=    TS2G14              REVER=  0 CONNT=  0
 FPTS2F03    0  ZNAME=    TS2G13              REVER=  0 CONNT=  0
 FPTS2F03    0  ZNAME=    TS2G11              REVER=  0 CONNT=  0
>FPTS2F04    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F04    0  ZNAME=    TS2G19              REVER=  0 CONNT=  0
 FPTS2F04    0  ZNAME=    TS2G18              REVER=  0 CONNT=  0
 FPTS2F04    0  ZNAME=    TS2G17              REVER=  0 CONNT=  0
>FPTS2F05    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F05    0  ZNAME=    TS2G22              REVER=  0 CONNT=  0
 FPTS2F05    0  ZNAME=    TS2G20              REVER=  0 CONNT=  0
 FPTS2F05    0  ZNAME=    TS2G18              REVER=  0 CONNT=  0
>FPTS2F06    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F06    0  ZNAME=    TS2G21              REVER=  0 CONNT=  0
 FPTS2F06    0  ZNAME=    TS2G20              REVER=  0 CONNT=  0
 FPTS2F06    0  ZNAME=    TS2G17              REVER=  0 CONNT=  0
>FPTS2F07    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 FPTS2F07    0  ZNAME=    TS2G22              REVER=  0 CONNT=  0
 FPTS2F07    0  ZNAME=    TS2G21              REVER=  0 CONNT=  0
 FPTS2F07    0  ZNAME=    TS2G19              REVER=  0 CONNT=  0
>FPTS2F08    0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
```

```
 FPTS2F08          0  ZNAME=     TS2G44                REVER=  0 CONNT=  0
>FPTS2F09          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS2F09          0  ZNAME=     TS2G44                REVER=  0 CONNT=  0
>FPTS2F10          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 FPTS2F10          0  ZNAME=     TS2G44                REVER=  0 CONNT=  0
>TS1F00            0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  2
 TS1F00            0  FFVAL= 72
 TS1F00            0  R    =  0 U    =  0
 TS1F00            0  ZNAME=     FPTS1F00              REVER=  1 CONNT=  0
 TS1F00            0  CSMSF=  0 REGNO=  4 BITNO= 12 REVER=  0
 TS1F00            0  CSMSF=  0 REGNO=  6 BITNO=  2 REVER=  0
>TS1F01            0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  1
 TS1F01            0  FFVAL= 72
 TS1F01            0  R    =  0 U    =  0
 TS1F01            0  ZNAME=     FPTS1F01              REVER=  1 CONNT=  0
 TS1F01            0  CSMSF=  0 REGNO=  4 BITNO= 11 REVER=  0
>TS1F02            0  CLASS=  2 TYPE =  0 VALUE=  1 NICON=  1 NECON=  1
 TS1F02            0  FFVAL= 72
 TS1F02            0  R    =  0 U    =  0
 TS1F02            0  ZNAME=     FPTS1F02              REVER=  1 CONNT=  0
 TS1F02            0  CSMSF=  0 REGNO=  5 BITNO= 17 REVER=  0
>TS1F03            0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  1
 TS1F03            0  FFVAL= 72
 TS1F03            0  R    =  0 U    =  0
 TS1F03            0  ZNAME=     FPTS1F03              REVER=  1 CONNT=  0
 TS1F03            0  CSMSF=  0 REGNO=  5 BITNO= 16 REVER=  0
>TS1F04            0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  1
 TS1F04            0  FFVAL= 72
 TS1F04            0  R    =  0 U    =  0
 TS1F04            0  ZNAME=     FPTS1F04              REVER=  1 CONNT=  0
 TS1F04            0  CSMSF=  0 REGNO=  5 BITNO= 15 REVER=  0
>TS1F05            0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  1
 TS1F05            0  FFVAL= 72
 TS1F05            0  R    =  0 U    =  0
 TS1F05            0  ZNAME=     FPTS1F05              REVER=  1 CONNT=  0
 TS1F05            0  CSMSF=  0 REGNO=  5 BITNO= 14 REVER=  0
>TS1F06            0  CLASS=  2 TYPE =  0 VALUE=  1 NICON=  1 NECON=  2
 TS1F06            0  FFVAL= 72
 TS1F06            0  R    =  0 U    =  0
 TS1F06            0  ZNAME=     FPTS1F06              REVER=  1 CONNT=  0
 TS1F06            0  CSMSF=  0 REGNO=  5 BITNO= 13 REVER=  0
 TS1F06            0  CSMSF=  0 REGNO=  6 BITNO=  1 REVER=  0
>TS1F07            0  CLASS=  2 TYPE =  0 VALUE=  1 NICON=  1 NECON=  2
 TS1F07            0  FFVAL= 72
 TS1F07            0  R    =  0 U    =  0
 TS1F07            0  ZNAME=     FPTS1F07              REVER=  1 CONNT=  0
 TS1F07            0  CSMSF=  0 REGNO=  5 BITNO= 12 REVER=  0
 TS1F07            0  CSMSF=  0 REGNO=  6 BITNO=  0 REVER=  0
>TS1G00            0  CLASS=  1 TYPE =  3 VALUE=  9 NICON=  1 NECON=  1
 TS1G00            0  ZNAME=     TS1F00                REVER=  0 CONNT=  7
 TS1G00            0  CSMSF=  0 REGNO=  4 BITNO= 17 REVER=  0
>TS1G01            0  CLASS=  1 TYPE =  3 VALUE=  9 NICON=  1 NECON=  1
 TS1G01            0  ZNAME=     TS1G02                REVER=  0 CONNT=  0
 TS1G01            0  CSMSF=  0 REGNO=  4 BITNO= 16 REVER=  0
>TS1G02            0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
```

```
 TS1G02          0  ZNAME=    TS1F01                REVER=  0 CONNT=  7
>TS1G03          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G03          0  ZNAME=    TS1G04                REVER=  0 CONNT=  0
>TS1G04          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G04          0  ZNAME=    TS1G05                REVER=  0 CONNT=  0
>TS1G05          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G05          0  ZNAME=    TS1G06                REVER=  0 CONNT=  0
>TS1G06          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  6 NECON=  1
 TS1G06          0  ZNAME=    TS1F07                REVER=  0 CONNT=  2
 TS1G06          0  ZNAME=    TS1F06                REVER=  0 CONNT=  2
 TS1G06          0  ZNAME=    TS1F05                REVER=  0 CONNT=  2
 TS1G06          0  ZNAME=    TS1F04                REVER=  0 CONNT=  2
 TS1G06          0  ZNAME=    TS1F03                REVER=  0 CONNT=  2
 TS1G06          0  ZNAME=    TS1F02                REVER=  0 CONNT=  2
 TS1G06          0  CSMSF=  0 REGNO=  4 BITNO= 15 REVER=  0
>TS1G07          0  CLASS=  1 TYPE =  5 VALUE=  1 NICON=  1 NECON=  0
 TS1G07          0  ZNAME=    TS1G08                REVER=  0 CONNT=  0
>TS1G08          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G08          0  ZNAME=    TS1G09                REVER=  0 CONNT=  0
>TS1G09          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G09          0  ZNAME=    TS1G10                REVER=  0 CONNT=  0
>TS1G10          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G10          0  ZNAME=    TS1G11                REVER=  0 CONNT=  0
>TS1G11          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G11          0  ZNAME=    TS1G12                REVER=  0 CONNT=  0
>TS1G12          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G12          0  ZNAME=    TS1G13                REVER=  0 CONNT=  0
>TS1G13          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  9 NECON=  0
 TS1G13          0  ZNAME=    TS1G07                REVER=  0 CONNT=  0
 TS1G13          0  ZNAME=    TS1F01                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F07                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F06                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F05                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F04                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F03                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1F02                REVER=  0 CONNT= -3
 TS1G13          0  ZNAME=    TS1G21                REVER=  0 CONNT=  0
>TS1G14          0  CLASS=  1 TYPE =  3 VALUE=  1 NICON=  1 NECON=  1
 TS1G14          0  ZNAME=    TS1F02                REVER=  0 CONNT=  7
 TS1G14          0  CSMSF=  0 REGNO=  4 BITNO= 14 REVER=  0
>TS1G15          0  CLASS=  1 TYPE =  3 VALUE=  0 NICON=  1 NECON=  0
 TS1G15          0  ZNAME=    TS1F03                REVER=  0 CONNT=  7
>TS1G16          0  CLASS=  1 TYPE =  3 VALUE=  0 NICON=  1 NECON=  0
 TS1G16          0  ZNAME=    TS1F04                REVER=  0 CONNT=  7
>TS1G17          0  CLASS=  1 TYPE =  3 VALUE=  0 NICON=  1 NECON=  1
 TS1G17          0  ZNAME=    TS1F05                REVER=  0 CONNT=  7
 TS1G17          0  CSMSF=  0 REGNO=  4 BITNO= 13 REVER=  0
>TS1G18          0  CLASS=  1 TYPE =  3 VALUE=  1 NICON=  1 NECON=  0
 TS1G18          0  ZNAME=    TS1F06                REVER=  0 CONNT=  7
>TS1G19          0  CLASS=  1 TYPE =  3 VALUE=  1 NICON=  1 NECON=  0
 TS1G19          0  ZNAME=    TS1F07                REVER=  0 CONNT=  7
>TS1G20          0  CLASS=  1 TYPE =  3 VALUE=  9 NICON=  1 NECON=  0
 TS1G20          0  ZNAME=    TS1G20                REVER=  0 CONNT=  0
>TS1G21          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G21          0  ZNAME=    TS1G22                REVER=  0 CONNT=  0
```

```
>TS1G22      0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G22      0  ZNAME=    TS1G23              REVER=  0 CONNT=  0
>TS1G23      0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  2 NECON=  2
 TS1G23      0  ZNAME=    TS1G24              REVER=  0 CONNT=  0
 TS1G23      0  ZNAME=    TS1G25              REVER=  0 CONNT=  0
 TS1G23      0  CSMSF=  0 REGNO=280 BITNO= 17 REVER=  0
 TS1G23      0  CSMSF=  0 REGNO=281 BITNO= 15 REVER=  0
>TS1G24      0  CLASS=  1 TYPE =  1 VALUE=  1 NICON=  1 NECON=  0
 TS1G24      0  ZNAME=    TS1G24              REVER=  0 CONNT=  0
>TS1G25      0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS1G25      0  ZNAME=    TS1G26              REVER=  0 CONNT=  0
>TS1G26      0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  3
 TS1G26      0  ZNAME=    TS1G27              REVER=  0 CONNT=  0
 TS1G26      0  CSMSF=  0 REGNO=280 BITNO= 17 REVER=  0
 TS1G26      0  CSMSF=  0 REGNO=281 BITNO= 17 REVER=  0
 TS1G26      0  CSMSF=  0 REGNO=281 BITNO= 16 REVER=  0
>TS1G27      0  CLASS=  1 TYPE =  1 VALUE=  1 NICON=  1 NECON=  0
 TS1G27      0  ZNAME=    TS1G27              REVER=  0 CONNT=  0
>TS2F00      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F00      0  FFVAL= 72
 TS2F00      0  R    =  0 U    =  0
 TS2F00      0  ZNAME=    FPTS2F00            REVER=  1 CONNT=  0
>TS2F01      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F01      0  FFVAL= 72
 TS2F01      0  R    =  0 U    =  0
 TS2F01      0  ZNAME=    FPTS2F01            REVER=  1 CONNT=  0
>TS2F02      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F02      0  FFVAL= 72
 TS2F02      0  R    =  0 U    =  0
 TS2F02      0  ZNAME=    FPTS2F02            REVER=  1 CONNT=  0
>TS2F03      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F03      0  FFVAL= 72
 TS2F03      0  R    =  0 U    =  0
 TS2F03      0  ZNAME=    FPTS2F03            REVER=  1 CONNT=  0
>TS2F04      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F04      0  FFVAL= 72
 TS2F04      0  R    =  0 U    =  0
 TS2F04      0  ZNAME=    FPTS2F04            REVER=  1 CONNT=  0
>TS2F05      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F05      0  FFVAL= 72
 TS2F05      0  R    =  0 U    =  0
 TS2F05      0  ZNAME=    FPTS2F05            REVER=  1 CONNT=  0
>TS2F06      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F06      0  FFVAL= 72
 TS2F06      0  R    =  0 U    =  0
 TS2F06      0  ZNAME=    FPTS2F06            REVER=  1 CONNT=  0
>TS2F07      0  CLASS=  2 TYPE =  0 VALUE=  9 NICON=  1 NECON=  0
 TS2F07      0  FFVAL= 72
 TS2F07      0  R    =  0 U    =  0
 TS2F07      0  ZNAME=    FPTS2F07            REVER=  1 CONNT=  0
>TS2F08      0  CLASS=  2 TYPE =  0 VALUE=  0 NICON=  1 NECON=  1
 TS2F08      0  FFVAL= 72
 TS2F08      0  R    =  0 U    =  0
 TS2F08      0  ZNAME=    FPTS2F08            REVER=  1 CONNT=  0
 TS2F08      0  CSMSF=  0 REGNO=  7 BITNO=  4 REVER=  0
```

```
>TS2F09          0  CLASS=  2 TYPE =  0 VALUE=  0 NICON=  1 NECON=  1
 TS2F09          0  FFVAL= 72
 TS2F09          0  R     =  0 U     =  0
 TS2F09          0  ZNAME=    FPTS2F09            REVER=  1 CONNT=  0
 TS2F09          0  CSMSF=  0 REGNO=  7 BITNO=  5 REVER=  0
>TS2F10          0  CLASS=  2 TYPE =  0 VALUE=  0 NICON=  1 NECON=  1
 TS2F10          0  FFVAL= 72
 TS2F10          0  R     =  0 U     =  0
 TS2F10          0  ZNAME=    FPTS2F10            REVER=  1 CONNT=  0
 TS2F10          0  CSMSF=  0 REGNO=  7 BITNO=  6 REVER=  0
>TS2G00          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  4 NECON=  1
 TS2G00          0  ZNAME=    TS2G27              REVER=  0 CONNT=  0
 TS2G00          0  ZNAME=    TS2G35              REVER=  0 CONNT=  0
 TS2G00          0  ZNAME=    TS2G39              REVER=  0 CONNT=  0
 TS2G00          0  ZNAME=    TS2F00              REVER=  0 CONNT=  7
 TS2G00          0  CSMSF=  0 REGNO=  7 BITNO=  7 REVER=  0
>TS2G01          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G01          0  ZNAME=    TS2F01              REVER=  0 CONNT=  7
 TS2G01          0  CSMSF=  0 REGNO=  7 BITNO=  8 REVER=  0
>TS2G02          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G02          0  ZNAME=    TS2F02              REVER=  0 CONNT=  7
 TS2G02          0  CSMSF=  0 REGNO=  7 BITNO=  9 REVER=  0
>TS2G03          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G03          0  ZNAME=    TS2F03              REVER=  0 CONNT=  7
 TS2G03          0  CSMSF=  0 REGNO=  7 BITNO= 10 REVER=  0
>TS2G04          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  8 NECON=  1
 TS2G04          0  ZNAME=    TS2F00              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F01              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F02              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F03              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F04              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F05              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F06              REVER=  0 CONNT= -3
 TS2G04          0  ZNAME=    TS2F07              REVER=  0 CONNT= -3
 TS2G04          0  CSMSF=  0 REGNO=  7 BITNO= 11 REVER=  0
>TS2G05          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  4 NECON=  1
 TS2G05          0  ZNAME=    TS2G31              REVER=  0 CONNT=  0
 TS2G05          0  ZNAME=    TS2G33              REVER=  0 CONNT=  0
 TS2G05          0  ZNAME=    TS2G37              REVER=  0 CONNT=  0
 TS2G05          0  ZNAME=    TS2F04              REVER=  0 CONNT=  7
 TS2G05          0  CSMSF=  0 REGNO=  7 BITNO= 12 REVER=  0
>TS2G06          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G06          0  ZNAME=    TS2F05              REVER=  0 CONNT=  7
 TS2G06          0  CSMSF=  0 REGNO=  7 BITNO= 13 REVER=  0
>TS2G07          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G07          0  ZNAME=    TS2F06              REVER=  0 CONNT=  7
 TS2G07          0  CSMSF=  0 REGNO=  7 BITNO= 14 REVER=  0
>TS2G08          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G08          0  ZNAME=    TS2F07              REVER=  0 CONNT=  7
 TS2G08          0  CSMSF=  0 REGNO=  7 BITNO= 15 REVER=  0
>TS2G09          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G09          0  ZNAME=    TS2G15              REVER=  0 CONNT=  0
>TS2G10          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G10          0  ZNAME=    TS2G15              REVER=  0 CONNT=  0
>TS2G11          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
 TS2G11    0  ZNAME=     TS2G15                  REVER=  0 CONNT=  0
>TS2G12    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G12    0  ZNAME=     TS2G15                  REVER=  0 CONNT=  0
>TS2G13    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G13    0  ZNAME=     TS2G15                  REVER=  0 CONNT=  0
>TS2G14    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G14    0  ZNAME=     TS2G15                  REVER=  0 CONNT=  0
>TS2G15    0  CLASS=  1  TYPE =  3 VALUE=  9 NICON=  1 NECON=  0
 TS2G15    0  ZNAME=     TS2G16                  REVER=  0 CONNT=  0
>TS2G16    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 TS2G16    0  ZNAME=     TS2G26                  REVER=  0 CONNT=  0
 TS2G16    0  ZNAME=     TS2G36                  REVER=  0 CONNT=  0
 TS2G16    0  ZNAME=     TS2G39                  REVER=  0 CONNT=  0
>TS2G17    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G17    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G18    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G18    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G19    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G19    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G20    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G20    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G21    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G21    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G22    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G22    0  ZNAME=     TS2G23                  REVER=  0 CONNT=  0
>TS2G23    0  CLASS=  1  TYPE =  3 VALUE=  9 NICON=  1 NECON=  0
 TS2G23    0  ZNAME=     TS2G24                  REVER=  0 CONNT=  0
>TS2G24    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  3 NECON=  0
 TS2G24    0  ZNAME=     TS2G30                  REVER=  0 CONNT=  0
 TS2G24    0  ZNAME=     TS2G34                  REVER=  0 CONNT=  0
 TS2G24    0  ZNAME=     TS2G37                  REVER=  0 CONNT=  0
>TS2G25    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  1
 TS2G25    0  ZNAME=     TS2G26                  REVER=  0 CONNT=  0
 TS2G25    0  CSMSF=  0  REGNO=  7 BITNO= 16 REVER=  0
>TS2G26    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G26    0  ZNAME=     TS2G28                  REVER=  0 CONNT=  0
>TS2G27    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G27    0  ZNAME=     TS2G28                  REVER=  0 CONNT=  0
>TS2G28    0  CLASS=  1  TYPE =  3 VALUE=  9 NICON=  1 NECON=  0
 TS2G28    0  ZNAME=     TS2F08                  REVER=  0 CONNT=  7
>TS2G29    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS2G29    0  ZNAME=     TS2G30                  REVER=  0 CONNT=  0
>TS2G30    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G30    0  ZNAME=     TS2G32                  REVER=  0 CONNT=  0
>TS2G31    0  CLASS=  1  TYPE =  1 VALUE=  9 NICON=  1 NECON=  0
 TS2G31    0  ZNAME=     TS2G32                  REVER=  0 CONNT=  0
>TS2G32    0  CLASS=  1  TYPE =  3 VALUE=  9 NICON=  1 NECON=  0
 TS2G32    0  ZNAME=     TS2F09                  REVER=  0 CONNT=  7
>TS2G33    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS2G33    0  ZNAME=     TS2G38                  REVER=  0 CONNT=  0
>TS2G34    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS2G34    0  ZNAME=     TS2G38                  REVER=  0 CONNT=  0
>TS2G35    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
 TS2G35    0  ZNAME=     TS2G40                  REVER=  0 CONNT=  0
>TS2G36    0  CLASS=  1  TYPE =  5 VALUE=  9 NICON=  1 NECON=  0
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
 TS2G36          0  ZNAME=    TS2G40                   REVER=  0 CONNT=   0
>TS2G37          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G37          0  ZNAME=    TS2G41                   REVER=  0 CONNT=   0
>TS2G38          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G38          0  ZNAME=    TS2G41                   REVER=  0 CONNT=   0
>TS2G39          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G39          0  ZNAME=    TS2G41                   REVER=  0 CONNT=   0
>TS2G40          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G40          0  ZNAME=    TS2G41                   REVER=  0 CONNT=   0
>TS2G41          0  CLASS=  1 TYPE =  3 VALUE=  9 NICON=  1 NECON=   0
 TS2G41          0  ZNAME=    TS2G42                   REVER=  0 CONNT=   0
>TS2G42          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=   0
 TS2G42          0  ZNAME=    TS2F10                   REVER=  0 CONNT=   7
>TS2G43          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  4 NECON=   1
 TS2G43          0  ZNAME=    TS2F08                   REVER=  0 CONNT=  -3
 TS2G43          0  ZNAME=    TS2F09                   REVER=  0 CONNT=  -3
 TS2G43          0  ZNAME=    TS2F10                   REVER=  0 CONNT=  -3
 TS2G43          0  ZNAME=    TS2G46                   REVER=  0 CONNT=   0
 TS2G43          0  CSMSF=  0 REGNO=  7 BITNO= 17 REVER=  0
>TS2G44          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G44          0  ZNAME=    TS2G44                   REVER=  0 CONNT=   0
>TS2G45          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  4 NECON=   0
 TS2G45          0  ZNAME=    TS2G27                   REVER=  0 CONNT=   0
 TS2G45          0  ZNAME=    TS2G25                   REVER=  0 CONNT=   0
 TS2G45          0  ZNAME=    TS2G29                   REVER=  0 CONNT=   0
 TS2G45          0  ZNAME=    TS2G31                   REVER=  0 CONNT=   0
>TS2G46          0  CLASS=  1 TYPE =  5 VALUE=  9 NICON=  1 NECON=   0
 TS2G46          0  ZNAME=    TS2G47                   REVER=  0 CONNT=   0
>TS2G47          0  CLASS=  1 TYPE =  5 VALUE=  0 NICON=  1 NECON=   2
 TS2G47          0  ZNAME=    TS2G48                   REVER=  0 CONNT=   0
 TS2G47          0  CSMSF=  0 REGNO=280 BITNO= 17 REVER=  0
 TS2G47          0  CSMSF=  0 REGNO=281 BITNO= 14 REVER=  0
>TS2G48          0  CLASS=  1 TYPE =  1 VALUE=  9 NICON=  1 NECON=   0
 TS2G48          0  ZNAME=    TS2G48                   REVER=  0 CONNT=   0
>ZZZFAULTER      1  CLASS=  1 TYPE=   1 VALUE=  0 NICON= 30 NECON=   1
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTEP     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
 ZZZFAULTER     10  ZNAME=    ZZZZDUMMY                REVER=  0 CONNT=   8
```

ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	8
ZZZFAULTER	11	CSMSF=	0	REGNO=	12	BITNO=	0	REVER=	0		
>ZZZZDUMMY	1	CLASS=	1	TYPE=	1	VALUE=	0	NICON=	1	NECON=	1
ZZZZDUMMY	10	ZNAME=		ZZZZDUMMY				REVER=	0	CONNT=	0
ZZZZDUMMY	11	CSMSF=	0	REGNO=	13	BITNO=	0	REVER=	0		

Appendix D

Sample Emulation Text Outputs

```
*** DIAGNOSTIC EMULATION ***     DATE:  9-MAR-87     TIME: 10:06:41
TARGET MACHINE: 3-bit counter with not,xor,or,and gates
BATCH:          Demonstrate Fixed Output
FAULT LIST:     No faults

________________________________ Run Number    1 ________________________________

T=      1          1 AVG STACK SIZE=     1.0 MIN SIZE=     1 MAX SIZE-     1 AVG FAN-4.00 AVG FAN FX-0.00 AVG FAN FN-0.00  Complete Stack
    1       1      20000  X1                   1 771700    1111110011110000000 to x2

T=      2          2 AVG STACK SIZE=     1.0 MIN SIZE=     1 MAX SIZE=     1 AVG FAN-4.00 AVG FAN FX-1.00 AVG FAN FN-1.00  Complete Stack
    1       2      20004  X2                   1 673401    1101110111000000001 lsb of counter:to x1,x3,x6

T=      3          3 AVG STACK SIZE=     1.3 MIN SIZE-     1 MAX SIZE-     2 AVG FAN-4.00 AVG FAN FX-2.00 AVG FAN FN-1.50  Complete Stack
    1       1      20000  X1                   0  70701    0001110001110000001 to x2
    2       3      20014  X3                   1 730701    1110110001110000001 to x5

T=      4          4 AVG STACK SIZE=     1.5 MIN SIZE-     1 MAX SIZE=     2 AVG FAN-4.00 AVG FAN FX-1.50 AVG FAN FN-1.25  Complete Stack
    1       5      20024  X5                   1 673400    1101110111000000000 middle bit of counter:to x4,x6
    2       2      20004  X2                   0  33400    0000110111000000000 lsb of counter:to x1,x3,x6

T=      5          5 AVG STACK SIZE=     1.8 MIN SIZE=     1 MAX SIZE-     3 AVG FAN-4.00 AVG FAN FX-1.33 AVG FAN FN-1.33  Complete Stack
    1       1      20000  X1                   1 770700    1111110001110000000 to x2
    2       3      20014  X3                   0  30700    0000110001110000000 to x5
    3       4      20020  X4                   1 730701    1110110001110000001 to x5

T=      6          6 AVG STACK SIZE=     1.8 MIN SIZE=     1 MAX SIZE-     3 AVG FAN-4.00 AVG FAN FX-1.56 AVG FAN FN-1.11  Complete Stack
    1       5      20024  X5                   1 673600    1101110111100000000 middle bit of counter:to x4,x6
    2       2      20004  X2                   1 673401    1101110111000000001 lsb of counter:to x1,x3,x6

T=      7          7 AVG STACK SIZE=     2.0 MIN SIZE=     1 MAX SIZE=     3 AVG FAN-4.00 AVG FAN FX-1.70 AVG FAN FN-1.30  Complete Stack
    1       1      20000  X1                   0  70701    0001110001110000001 to x2
    2       3      20014  X3                   1 730701    1110110001110000001 to x5
    3       6      20033  X6                   1 730700    1110110001110000000 to x8

T=     10          8 AVG STACK SIZE=     2.1 MIN SIZE=     1 MAX SIZE-     3 AVG FAN-4.00 AVG FAN FX-1.54 AVG FAN FN-1.23  Complete Stack
    1       8      20043  X8                   1 673400    1101110111000000000 msb of counter:to x7
    2       5      20024  X5                   0  33401    0000110111000000001 middle bit of counter:to x4,x6
    3       2      20004  X2                   0  33400    0000110111000000000 lsb of counter:to x1,x3,x6

T=     11          9 AVG STACK SIZE=     2.4 MIN SIZE-     1 MAX SIZE-     5 AVG FAN-4.00 AVG FAN FX-1.63 AVG FAN FN-1.31  Complete Stack
    1       1      20000  X1                   1 770700    1111110001110000000 to x2
    2       3      20014  X3                   0  30700    0000110001110000000 to x5
    3       6      20033  X6                   0  30776    0000110001111111110 to x8
    4       4      20020  X4                   0  30700    0000110001110000000 to x5
    5       7      20037  X7                   1 730701    1110110001110000001 to x8
__________________________________________________

----EMULATION RUN IS COMPLETE AT TIME=    10 (decimal)        12 (octal)
    AVERAGE STACK SIZE=   2.4 MIN SIZE=     1 MAX SIZE=     5
    DATE:  9-MAR-87 TIME: 10:06:42
```

Example 1

ORIGINAL PAGE IS
OF POOR QUALITY

*** DIAGNOSTIC EMULATION *** DATE: 9-MAR-87 TIME: 10:07:38

MACHINE: 3-bit counter with not,xor,or,and gates

BATCH: Show Run #, Stack Size, Term, Data

FAULT LIST: No faults

			Run Number		
1	1	STACK SIZE & TIME(OCTAL)	1	1	STACK SIZE & TIME(DECIMAL)
1	2	STACK SIZE & TIME(OCTAL)	1	2	STACK SIZE & TIME(DECIMAL)
2	3	STACK SIZE & TIME(OCTAL)	2	3	STACK SIZE & TIME(DECIMAL)
2	4	STACK SIZE & TIME(OCTAL)	2	4	STACK SIZE & TIME(DECIMAL)
3	5	STACK SIZE & TIME(OCTAL)	3	5	STACK SIZE & TIME(DECIMAL)
2	6	STACK SIZE & TIME(OCTAL)	2	6	STACK SIZE & TIME(DECIMAL)
3	7	STACK SIZE & TIME(OCTAL)	3	7	STACK SIZE & TIME(DECIMAL)
3	10	STACK SIZE & TIME(OCTAL)	3	8	STACK SIZE & TIME(DECIMAL)
5	11	STACK SIZE & TIME(OCTAL)	5	9	STACK SIZE & TIME(DECIMAL)

1 2 3 4 5 6

----EMULATION RUN IS COMPLETE AT TIME- 10 (decimal) 12 (octal)

Address T= 12 10 ----Control Store Dump----
4000 0 11 4 1 0 0 0 0 0 1 1 0 0 0 0 0

Address T= 12 10 ----Main Store Dump----
100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Address T= 12 10 ----Local Store Dump----
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

AVERAGE STACK SIZE= 2.4 MIN SIZE= 1 MAX SIZE= 5

DATE: 9-MAR-87 TIME: 10:07:38

Example 2

*** DIAGNOSTIC EMULATION *** DATE: 9-MAR-87 TIME: 10:07:57

MACHINE: 3-bit counter with not,xor,or,and gates

BATCH: Show Memory Dumps

FAULT LIST: No faults

Run Number 1

Address	T=			----Control Store Dump----												
4000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	1	1	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	2	2	----Control Store Dump----												
4000	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	2	2	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	3	3	----Control Store Dump----												
4000	0	3	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Address	T=	3	3	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	4	4	----Control Store Dump----												
4000	0	4	1	0	1	1	0	0	0	0	0	0	0	0	0	0
Address	T=	4	4	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	5	5	----Control Store Dump----												
4000	0	5	2	0	0	1	0	1	0	0	0	0	0	0	0	0
Address	T=	5	5	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	6	6	----Control Store Dump----												
4000	0	6	2	1	0	0	1	1	0	0	0	0	0	0	0	0
Address	T=	6	6	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	7	7	----Control Store Dump----												
4000	0	7	3	1	1	0	1	1	0	0	0	0	0	0	0	0
Address	T=	7	7	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	10	8	----Control Store Dump----												
4000	0	10	3	0	1	1	1	1	1	0	0	0	0	0	0	0
Address	T=	10	8	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address	T=	11	9	----Control Store Dump----												
4000	0	11	4	0	0	1	1	0	1	0	1	0	0	0	0	0
Address	T=	11	9	----Main Store Dump----												
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Example 3

```
*** DIAGNOSTIC EMULATION ***      DATE:  9-MAR-87     TIME: 1Ø:Ø8:21
[redacted]CHINE: 3-bit counter with not,xor,or,and gates
BATCH:          Show Time Line
FAULT LIST:     No faults
                                     Run Number
T=   1    1 AVG STACK SIZE=   1.Ø MIN SIZE=   1 MAX SIZE=   1 AVG FANOUT=4.2Ø AVG FAN EX=Ø.ØØ AVG FAN ENO=Ø.ØØ
T=   2    2 AVG STACK SIZE=   1.Ø MIN SIZE=   1 MAX SIZE=   1 AVG FANOUT=4.2Ø AVG FAN EX=1.ØØ AVG FAN ENO=1.ØØ
T=   3    3 AVG STACK SIZE=   1.3 MIN SIZE=   1 MAX SIZE=   2 AVG FANOUT=4.2Ø AVG FAN EX=2.ØØ AVG FAN ENO=1.5Ø
T=   4    4 AVG STACK SIZE=   1.5 MIN SIZE=   1 MAX SIZE=   2 AVG FANOUT=4.2Ø AVG FAN EX=1.5Ø AVG FAN ENO=1.25
T=   5    5 AVG STACK SIZE=   1.8 MIN SIZE=   1 MAX SIZE=   3 AVG FANOUT=4.2Ø AVG FAN EX=1.83 AVG FAN ENO=1.33
T=   6    6 AVG STACK SIZE=   1.8 MIN SIZE=   1 MAX SIZE=   3 AVG FANOUT=4.2Ø AVG FAN EX=1.56 AVG FAN ENO=1.11
T=   7    7 AVG STACK SIZE=   2.Ø MIN SIZE=   1 MAX SIZE=   3 AVG FANOUT=4.2Ø AVG FAN EX=1.7Ø AVG FAN ENO=1.3Ø
T=   1Ø   8 AVG STACK SIZE=   2.1 MIN SIZE=   1 MAX SIZE=   3 AVG FANOUT=4.2Ø AVG FAN EX=1.54 AVG FAN ENO=1.23
T=   11   9 AVG STACK SIZE=   2.4 MIN SIZE=   1 MAX SIZE=   5 AVG FANOUT=4.2Ø AVG FAN EX=1.63 AVG FAN ENO=1.31

----EMULATION RUN IS COMPLETE AT TIME=    1Ø (decimal)     12 (octal)
    AVERAGE STACK SIZE=    2.4 MIN SIZE=    1 MAX SIZE=    5
    DATE:  9-MAR-87 TIME: 1Ø:Ø8:22
```


ORIGINAL PAGE IS
OF POOR QUALITY

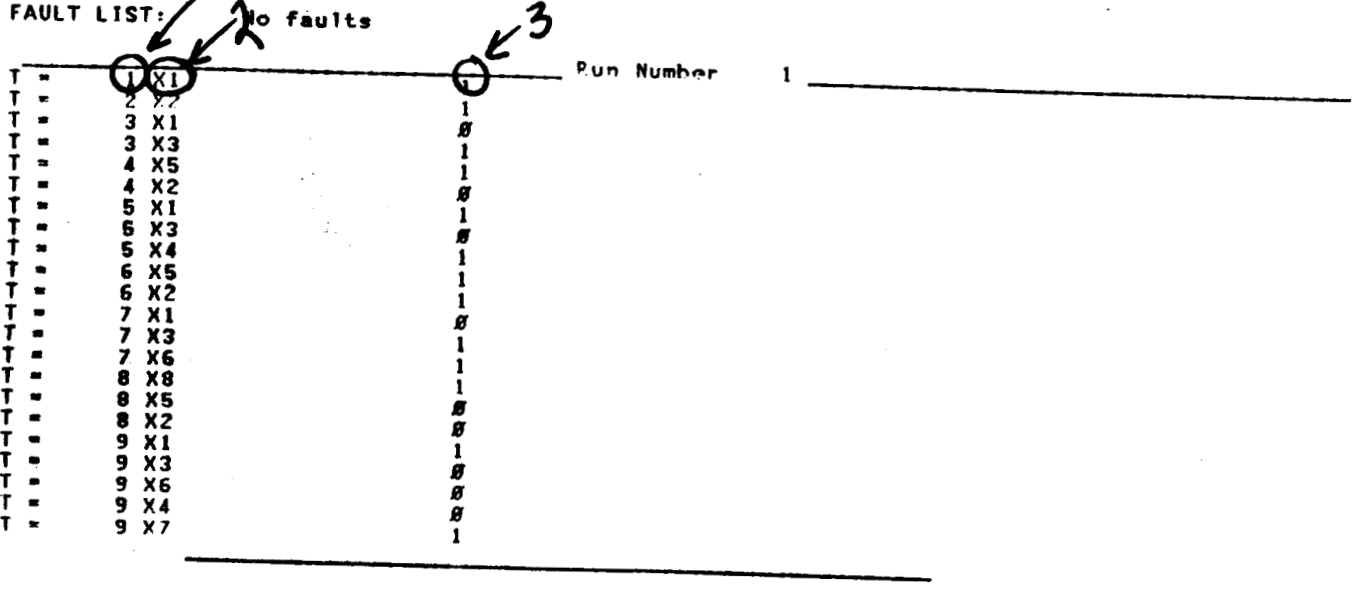
Example 4

*** DIAGNOSTIC EMULATION *** DATE: 9-MAR-87 TIME: 10:32:13

[redacted]CHINE: 3-bit counter with not,xor,or,and gates

BATCH: Show Stack in Abbreviated Mode

FAULT LIST: No faults



T =	1	X1	1
T =	2	X2	Ø
T =	3	X1	1
T =	3	X3	Ø
T =	4	X5	1
T =	4	X2	Ø
T =	5	X1	1
T =	5	X3	1
T =	5	X4	1
T =	6	X5	Ø
T =	6	X2	1
T =	7	X1	1
T =	7	X3	Ø
T =	7	X6	Ø
T =	8	X8	1
T =	8	X5	Ø
T =	8	X2	Ø
T =	9	X1	Ø
T =	9	X3	Ø
T =	9	X6	Ø
T =	9	X4	Ø
T =	9	X7	1

----EMULATION RUN IS COMPLETE AT TIME= 10 (decimal) 12 (octal)

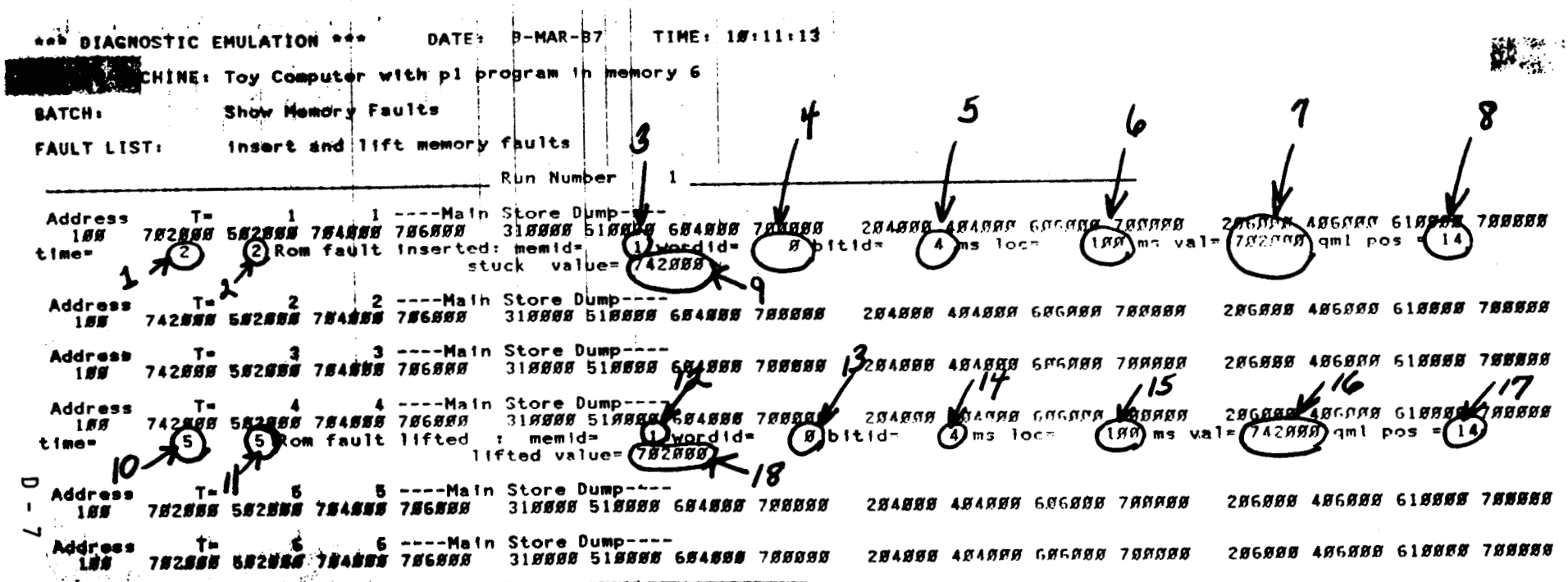
AVERAGE STACK SIZE= 2.4 MIN SIZE= 1 MAX SIZE= 5

DATE: 9-MAR-87 TIME: 10:32:13

Example 5

```
*** DIAGNOSTIC EMULATION ***     DATE:  9-MAR-87    TIME: 1Ø:Ø9:4Ø
[illegible]CHINE: 3-bit counter with not,xor,or,and gates
BATCH:          Show Gate Faults
FAULT LIST:     Stick X1 to Ø at t=2 and lift t=8
___________________________________ Run Number   1 ____________________________
T=    1        1 AVG STACK SIZE=     1.Ø MIN SIZE=     1 MAX SIZE=     1 AVG FAN=4.2Ø AVG FAN EX=Ø.ØØ AVG FAN EN=Ø.ØØ  Complete Stack
   1      1    2ØØØØ   X1                    1 7717ØØ  1111110011110000000 to x2
time=     2    2 Stick at Ø       X1
T=    2        2 AVG STACK SIZE=     1.5 MIN SIZE=     1 MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.ØØ AVG FAN EN=1.ØØ  Complete Stack
   1      2    2ØØØ4   X2                    1 6734Ø1  1101110111000000001 lsb of counter:to x1,x3,x6
   2      9    2ØØ51   ZZZFAULTER            Ø  33477  0000110111001111111
T=    3        3 AVG STACK SIZE=     1.7 MIN SIZE=     1 MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.67 AVG FAN EN=1.ØØ  Complete Stack
   1      1    2ØØØØ   X1                    Ø  5Ø7Ø1  0001010001110000001 to x2
   2      3    2ØØ14   X3                    1 73Ø7Ø1  1110110001110000001 to x5
T=    4        4 AVG STACK SIZE=     1.8 MIN SIZE=     1 MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.4Ø AVG FAN EN=1.ØØ  Complete Stack
   1      5    2ØØ24   X6                    1 6734ØØ  1101110111000000000 middle bit of counter:to x4,x6
   2      2    2ØØØ4   X2                    Ø  334ØØ  0000110111000000000 lsb of counter:to x1,x3,x6
T=    5        5 AVG STACK SIZE=     1.8 MIN SIZE=     1 MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.71 AVG FAN EN=1.ØØ  Complete Stack
   1      3    2ØØ14   X3                    Ø  3Ø7ØØ  0000110001110000000 to x5
   2      4    2ØØ2Ø   X4                    1 73Ø7Ø1  1110110001110000001 to x5
T=    6        6 AVG STACK SIZE=     1.7 MIN SIZE=     1 MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.56 AVG FAN EN=Ø.89  Complete Stack
   1      5    2ØØ24   X5                    1 6736ØØ  1101110111100000000 middle bit of counter:to x4,x6
time=    1Ø    8 Lift fault:      X1
T=   1Ø        8 AVG STACK SIZE=     1.4 MIN SIZE=     Ø MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.56 AVG FAN EN=Ø.89  Complete Stack
   1      9    2ØØ51   ZZZFAULTER            Ø  33477  0000110111001111111
T=   11        9 AVG STACK SIZE=     1.3 MIN SIZE=     Ø MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.5Ø AVG FAN EN=Ø.9Ø  Complete Stack
   1      1    2ØØØØ   X1                    1 77Ø7ØØ  1111110001110000000 to x2
T=   12       1Ø AVG STACK SIZE=     1.3 MIN SIZE=     Ø MAX SIZE=     2 AVG FAN=4.2Ø AVG FAN EX=1.45 AVG FAN EN=Ø.91  Complete Stack
   1      2    2ØØØ4   X2                    1 6734Ø1  1101110111000000001 lsb of counter:to x1,x3,x6
               ______________________________________________________
----EMULATION RUN IS COMPLETE AT TIME=     11 (decimal)      13 (octal)
    AVERAGE STACK SIZE=    1.3 MIN SIZE=    Ø MAX SIZE=    2
    DATE:  9-MAR-87 TIME: 1Ø:Ø9:41
```

Example 6



Example 7

```
*** DIAGNOSTIC EMULATION ***     DATE:  9-MAR-87     TIME: 1Ø:Ø9:59
[redacted]CHINE: 3-bit counter with not,xor,or,and gates
BATCH:          Show trace devices
FAULT LIST:     No faults
____________________________________ Run Number   1 ______________________________
T=    1        1 AVG STACK SIZE=   1.Ø MIN SIZE=    1 MAX SIZE=    1 AVG FAN=4.2Ø AVG FAN EX=Ø.ØØ AVG FAN EN=Ø.ØØ  Trace Stack
   1      1    2ØØØØ  X1                  1 77177ØØ  1111110Ø1111ØØØØØØ to x2
T=    3        3 AVG STACK SIZE=   1.3 MIN SIZE=    1 MAX SIZE=    2 AVG FAN=4.2Ø AVG FAN EX=2.ØØ AVG FAN EN=1.5Ø  Trace Stack
   1      1    2ØØØØ  X1                  Ø  7Ø7Ø1  ØØØ11ØØØ111ØØØØØ1 to x2
T=    4        4 AVG STACK SIZE=   1.5 MIN SIZE=    1 MAX SIZE=    2 AVG FAN=4.2Ø AVG FAN EX=1.5Ø AVG FAN EN=1.25  Trace Stack
   1      5    2ØØ24  X5                  1 6734ØØ  11Ø111Ø111ØØØØØØØØ middle bit of counter:to x4,x6
T=    5        5 AVG STACK SIZE=   1.8 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.83 AVG FAN EN=1.33  Trace Stack
   1      1    2ØØØØ  X1                  1 777Ø7ØØ  111111ØØØ111ØØØØØØ to x2
T=    6        6 AVG STACK SIZE=   1.8 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.56 AVG FAN EN=1.11  Trace Stack
   1      5    2ØØ24  X5                  1 6736ØØ  11Ø111Ø1111ØØØØØØØ middle bit of counter:to x4,x6
T=    7        7 AVG STACK SIZE=   2.Ø MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.7Ø AVG FAN EN=1.3Ø  Trace Stack
   1      1    2ØØØØ  X1                  Ø  7Ø7Ø1  ØØØ111ØØØ111ØØØØØ1 to x2
T=   1Ø        8 AVG STACK SIZE=   2.1 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.54 AVG FAN EN=1.23  Trace Stack
   2      5    2ØØ24  X5                  Ø  334Ø1  ØØØØ11Ø111ØØØØØØØ1 middle bit of counter:to x4,x6
T=   11        9 AVG STACK SIZE=   2.4 MIN SIZE=    1 MAX SIZE=    5 AVG FAN=4.2Ø AVG FAN EX=1.63 AVG FAN EN=1.31  Trace Stack
   1      1    2ØØØØ  X1                  1 777Ø7ØØ  111111ØØØ111ØØØØØØ to x2

----EMULATION RUN IS COMPLETE AT TIME=    1Ø (decimal)      12 (octal)
    AVERAGE STACK SIZE=    2.4 MIN SIZE=    1 MAX SIZE=    5
    DATE:  9-MAR-87 TIME: 1Ø:1Ø:ØØ
```

ORIGINAL PAGE IS
OF POOR QUALITY

Example 8

```
Show trace devices
ØØ4ØØØ,ØØ4Ø17     ! cs lo,hi addresses, printing (o6,1x,o6) ØØ24ØØ,4437
ØØØ1ØØ,ØØØ117     ! ms     "          "           "
ØØØØØØ,ØØØØ1Ø     ! ls     "          "           "
Ø                 !*1 initial headers for each device (i1)    ***print options
Ø                 ! 2 octal input to qml
Ø                 ! 3 checking file
1                 ! 4   checking device headers
1                 ! 5   checking stacks
1                 ! 6   checking control store
Ø                 ! 7   checking main store
Ø                 !*8  control store memory
Ø                 !*9  main store memory
Ø                 !*1Ø local store memory
Ø                 !*11 stack
Ø                 ! 12   checking local store
Ø                 ! 13 read action data
Ø                 ! 14 punch input for qml
Ø                 ! 15 initialization debug data
Ø                 !*16 connections list
Ø                 ! 17 debug data
Ø                 ! 18 initialized arrays (pinit)
Ø                 ! 19      "         "
Ø                 !*2Ø devices with undefined output values
Ø                 ! 21 devices with defined output values
Ø                 ! 22 write action data
Ø                 ! 23 stop action data
Ø                 ! 24 termination data
Ø                 ! 25 alphabetized list of devices
Ø                 ! 26 debug output from tryval
Ø                 ! 27 device name list
Ø                 ! 28 time line
Ø                 ! 29 stack size(stack must also be turned on)
Ø                 ! 3Ø stack dump in abbreviated mode
1                 ! 31 insertion and lifting of gate faults
Ø                 ! 32 enqueing of faulting device
1                 ! 33 insertion and lifting of memory faults
Ø                 ! 34 partial fault list to regular output file
Ø                 ! 35 external inputs (act 7 and schnel)
Ø                 ! 36 intermediate time, level 1
Ø                 ! 37 intermediate time, level 2
Ø                 ! 38 external outputs
Ø                 ! 39 debugging of faultor link words (pstack)
1                 ! 4Ø run numbers
Ø                 ! 41 fault file for qml
Ø                 ! 42 scheduling actions in schact
Ø                 ! 43 cs external registers for qml
Ø                 ! 44 external input data for qml
Ø                 ! 45 external output data for qml
Ø                 ! 46 abbreviated run to produce qml data only
Ø                 ! 47
Ø                 ! 48
Ø                 ! 49
Ø                 ! 5Ø
1,1ØØ,1             ! print option start,stop,interval times(*)
1ØØ                 ! stop time                           (*)
X1
X5
*****               ! end of traced devices-user print choices follow
*****
*****               !end of devices to have headers printed(TRACE ON TOO)
1                                  !timesiz
Ø                                  !nexinp(*)       no. of ei lists
Ø                                  !nexoup          no. of eo lists
```

Names of Trace Devices

Example 8

*** DIAGNOSTIC EMULATION *** DATE: 9-MAR-87 TIME: 10:10:24

TARGET MACHINE: 3-bit counter with not,xor,or,and gates

BATCH: Show specific device headers

FAULT LIST: No faults

						Run Number 1
TIME=	1	1	2	20004	X2	33700
TIME=	1	1	6	20033	X6	33776
TIME=	2	2	2	20004	X2	673401
TIME=	2	2	6	20033	X6	33776
TIME=	3	3	2	20004	X2	673701
TIME=	3	3	6	20033	X6	33777
TIME=	4	4	2	20004	X2	33400
TIME=	4	4	6	20033	X6	33777
TIME=	5	5	2	20004	X2	33700
TIME=	5	5	6	20033	X6	33777
TIME=	6	6	2	20004	X2	673401
TIME=	6	6	6	20033	X6	33777
TIME=	7	7	2	20004	X2	673701
TIME=	7	7	6	20033	X6	733700
TIME=	10	8	2	20004	X2	33400
TIME=	10	8	6	20033	X6	733700
TIME=	11	9	2	20004	X2	33700
TIME=	11	9	6	20033	X6	30776

----EMULATION RUN IS COMPLETE AT TIME= 10 (decimal) 12 (octal)

AVERAGE STACK SIZE= 2.4 MIN SIZE= 1 MAX SIZE= 5

DATE: 9-MAR-87 TIME: 10:10:25

Example 9

```
Show specific device headers
[illegible]4017    ! cs lo,hi addresses. printing  (o6.1x,o6) 007400,4437
[illegible]117     ! ms    "       "       "       "
000000,000010      ! ls    "       "       "       "
0                  !*1 initial headers for each device (11)    ***print options
0                  ! 2 octal input to qmi
0                  ! 3 checking file
1                  ! 4   checking device headers
1                  ! 5   checking stacks
1                  ! 6   checking control store
0                  ! 7   checking main store
0                  !*8  control store memory
0                  !*9  main store memory
0                  !*10 local store memory
0                  !*11 stack
0                  ! 12   checking local store
0                  ! 13 read action data
0                  ! 14 punch input for qmi
0                  ! 15 initialization debug data
0                  !*16 connections list
0                  ! 17 debug data
0                  ! 18 initialized arrays (pinit)
0                  ! 19    "       "
0                  !*20 devices with undefined output values
0                  ! 21 devices with defined output values
0                  ! 22 write action data
0                  ! 23 stop action data
0                  ! 24 termination data
0                  ! 25 alphabetized list of devices
0                  ! 26 debug output from tryval
0                  ! 27 device name list
0                  ! 28 time line
0                  ! 29 stack size(stack must also be turned on)
0                  ! 30 stack dump in abbreviated mode
1                  ! 31 insertion and lifting of gate faults
0                  ! 32 enqueing of faulting device
1                  ! 33 insertion and lifting of memory faults
0                  ! 34 partial fault list to regular output file
0                  ! 35 external inputs (act 7 and schnet)
0                  ! 36 intermediate time, level 1
0                  ! 37 intermediate time, level 2
0                  ! 38 external outputs
0                  ! 39 debugging of faulter link words (pstack)
1                  ! 40 run numbers
0                  ! 41 fault file for qmi
0                  ! 42 scheduling actions in schact
0                  ! 43 ee external registers for qmi
0                  ! 44 external input data for qmi
0                  ! 45 external output data for qmi
0                  ! 46 abbreviated run to produce qmi data only
0                  ! 47
0                  ! 48
0                  ! 49
0                  ! 50
1,10,1                ! print option start,stop,interval times(*)
100                ! stop time                                  (*)
*****              ! end of traced devices-user print choices follow
*****
X2
X6
*****              !end of devices to have headers printed
1                                 !timesiz
0                                 !nexinp(*)     no. of ei lists
0                                 !nexoup        no. of eo lists
```

Names of Devices for which state information will be printed.

Example 9

```
*** DIAGNOSTIC EMULATION ***      DATE:  9-MAR-87     TIME: 1Ø:1Ø:41
TARGET MACHINE: 3-bit counter with not,xor,or,and gates
BATCH:           Show Stack in Full Mode
FAULT LIST:      No faults
                                        Run Number     1
T=     1   1 AVG STACK SIZE=    1.Ø MIN SIZE=    1 MAX SIZE=    1 AVG FAN=4.2Ø AVG FAN EX=Ø.ØØ AVG FAN EN=Ø.ØØ  Complete Stack
   1    1   2ØØØØ  X1                   1 7717ØØ 111110Ø111110ØØØØØ to x2
T=     2   2 AVG STACK SIZE=    1.Ø MIN SIZE=    1 MAX SIZE=    1 AVG FAN=4.2Ø AVG FAN EX=1.ØØ AVG FAN EN=1.ØØ  Complete Stack
   1    2   2ØØØ4  X2                   1 6734Ø1 11Ø111Ø111ØØØØØØØ1 lsb of counter:to x1,x3,x6
T=     3   3 AVG STACK SIZE=    1.3 MIN SIZE=    1 MAX SIZE=    2 AVG FAN=4.2Ø AVG FAN EX=1.ØØ AVG FAN EN=1.5Ø  Complete Stack
   1    1   2ØØØØ  X1                   Ø  7Ø7Ø1 ØØØ111ØØØ111ØØØØØ1 to x2
   2    3   2ØØ14  X3                   1 73Ø7Ø1 111Ø11ØØØ111ØØØØØ1 to x5
T=     4   4 AVG STACK SIZE=    1.5 MIN SIZE=    1 MAX SIZE=    2 AVG FAN=4.2Ø AVG FAN EX=1.5Ø AVG FAN EN=1.25  Complete Stack
   1    5   2ØØ24  X5                   1 6734ØØ 11Ø111Ø111ØØØØØØØØ middle bit of counter:to x4,x6
   2    2   2ØØØ4  X2                   Ø  334ØØ ØØØØ11Ø111ØØØØØØØØ lsb of counter:to x1,x3,x6
T=     5   5 AVG STACK SIZE=    1.8 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.83 AVG FAN EN=1.33  Complete Stack
   1    1   2ØØØØ  X1                   1 77Ø7ØØ 111111ØØØ111ØØØØØØ to x2
   2    3   2ØØ14  X3                   Ø  3Ø7ØØ ØØØØ11ØØØ111ØØØØØØ to x5
   3    4   2ØØ2Ø  X4                   1 73Ø7Ø1 111Ø11ØØØ111ØØØØØ1 to x5
T=     6   6 AVG STACK SIZE=    1.8 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.56 AVG FAN EN=1.11  Complete Stack
   1    5   2ØØ24  X5                   1 6736ØØ 11Ø111Ø111ØØØØØØØØ middle bit of counter:to x4,x6
   2    2   2ØØØ4  X2                   1 6734Ø1 11Ø111Ø111ØØØØØØØ1 lsb of counter:to x1,x3,x6
T=     7   7 AVG STACK SIZE=    2.Ø MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.7Ø AVG FAN EN=1.3Ø  Complete Stack
   1    1   2ØØØØ  X1                   Ø  7Ø7Ø1 ØØØ111ØØØ111ØØØØØ1 to x2
   2    3   2ØØ14  X3                   1 73Ø7Ø1 111Ø11ØØØ111ØØØØØ1 to x5
   3    6   2ØØ33  X6                   1 73Ø7ØØ 111Ø11ØØØ111ØØØØØØ to x8
T=    1Ø   8 AVG STACK SIZE=    2.1 MIN SIZE=    1 MAX SIZE=    3 AVG FAN=4.2Ø AVG FAN EX=1.54 AVG FAN EN=1.23  Complete Stack
   1    8   2ØØ43  X8                   1 6734ØØ 11Ø111Ø111ØØØØØØØØ msb of counter:to x7
   2    5   2ØØ24  X5                   Ø  334Ø1 ØØØØ11Ø111ØØØØØØØ1 middle bit of counter:to x4,x6
   3    2   2ØØØ4  X2                   Ø  334ØØ ØØØØ11Ø111ØØØØØØØØ lsb of counter:to x1,x3,x6
T=    11   9 AVG STACK SIZE=    2.4 MIN SIZE=    1 MAX SIZE=    5 AVG FAN=4.2Ø AVG FAN EX=1.63 AVG FAN EN=1.31  Complete Stack
   1    1   2ØØØØ  X1                   1 77Ø7ØØ 111111ØØØ111ØØØØØØ to x2
   2    3   2ØØ14  X3                   Ø  3Ø7ØØ ØØØØ11ØØØ111ØØØØØØ to x5
   3    6   2ØØ33  X6                   Ø  3Ø776 ØØØØ11ØØØ11111111Ø to x8
   4    4   2ØØ2Ø  X4                   Ø  3Ø7ØØ ØØØØ11ØØØ111ØØØØØØ to x5
   5    7   2ØØ37  X7                   1 73Ø7Ø1 111Ø11ØØØ111ØØØØØ1 to x8
----EMULATION RUN IS COMPLETE AT TIME=     1Ø (decimal)     12 (octal)
    AVERAGE STACK SIZE=   2.4 MIN SIZE=    1 MAX SIZE=    5
    DATE:  9-MAR-87 TIME: 1Ø:1Ø:42
```

ORIGINAL PAGE IS
OF POOR QUALITY

Example 10

Appendix E

Terms and Abbreviations

Terms and Abbreviations

Terms

device	A gate, flip-flop or tri-state.
simple gate, or regular gate	An AND,OR,NAND,NOR,XOR, or NXOR gate.
device name	A 20-character name assigned by the user to the device
device index number	An integer assigned to each device by the initialization program. The first device in the netlist is assigned the number 1, and integers are then assigned sequentially to the remaining devices in the order in which they appear in the netlist.
device address	The beginning QM-1 control store location assigned by the initialization program to hold the state description or "header word" (see ()) for the device.
device identifier	For the Vax: the device index number For the QM-1: the device address.
stack	A list of device index numbers(for vax emulator) or device addresses(for QM-1 emulator) of those devices whose output values changed during the previous time step.

Abbreviations

ei	external inputs
eo	external outputs
cs	control store
ms	main store
*filename	User Prefix followed by rest of file name

General Notes

Device Names	The user must enter a device name in Upper Case in every instance in which it appears in any input file.
Fortran Formats	It is assumed in the descriptions of the input files to the emulator programs that the user is familiar with Fortran Format statements.
Radix Notation	All numbers in this document are assumed to be decimal, unless the radix is specifically noted, for example, "octal".

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-178391		

4. Title and Subtitle
Diagnostic Emulation:
Implementation and User's Guide

5. Report Date
December 1987

6. Performing Organization Code

7. Author(s)
Bernice Becher

8. Performing Organization Report No.

10. Work Unit No.
505-66-21-03

9. Performing Organization Name and Address
PRC Kentron, Inc.
Hampton, VA 23666

11. Contract or Grant No.
NAS1-18000

13. Type of Report and Period Covered
Contractor Report

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, DC 20546-0001

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The Diagnostic Emulation Technique was developed within the System Validation Methods Branch as a part of the development of methods for the analysis of the reliability of highly reliable, fault tolerant digital avionics systems. This is a general technique which allows for the emulation of a digital hardware system. The technique is general in the sense that it is completely independent of the particular target hardware which is being emulated. Parts of the system are described and emulated at the logic or gate level, while other parts of the system are described and emulated at the functional level. This algorithm allows for the insertion of faults into the system, and for the observation of the response of the system to these faults. This allows for controlled and accelerated testing of system reaction to hardware failures in the target machine. This document describes in detail how the algorithm was implemented at NASA Langley Research Center and gives instructions for using the system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Fault tolerance fault simulation logic simulation	Unclassified - Unlimited Subject Catetory 66

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	172	A08