REFILL KI - HARDWARE PAGE DONE
REFILL KL - MICRO HANDLES PAGE

DICK BAKER

CLK SP.

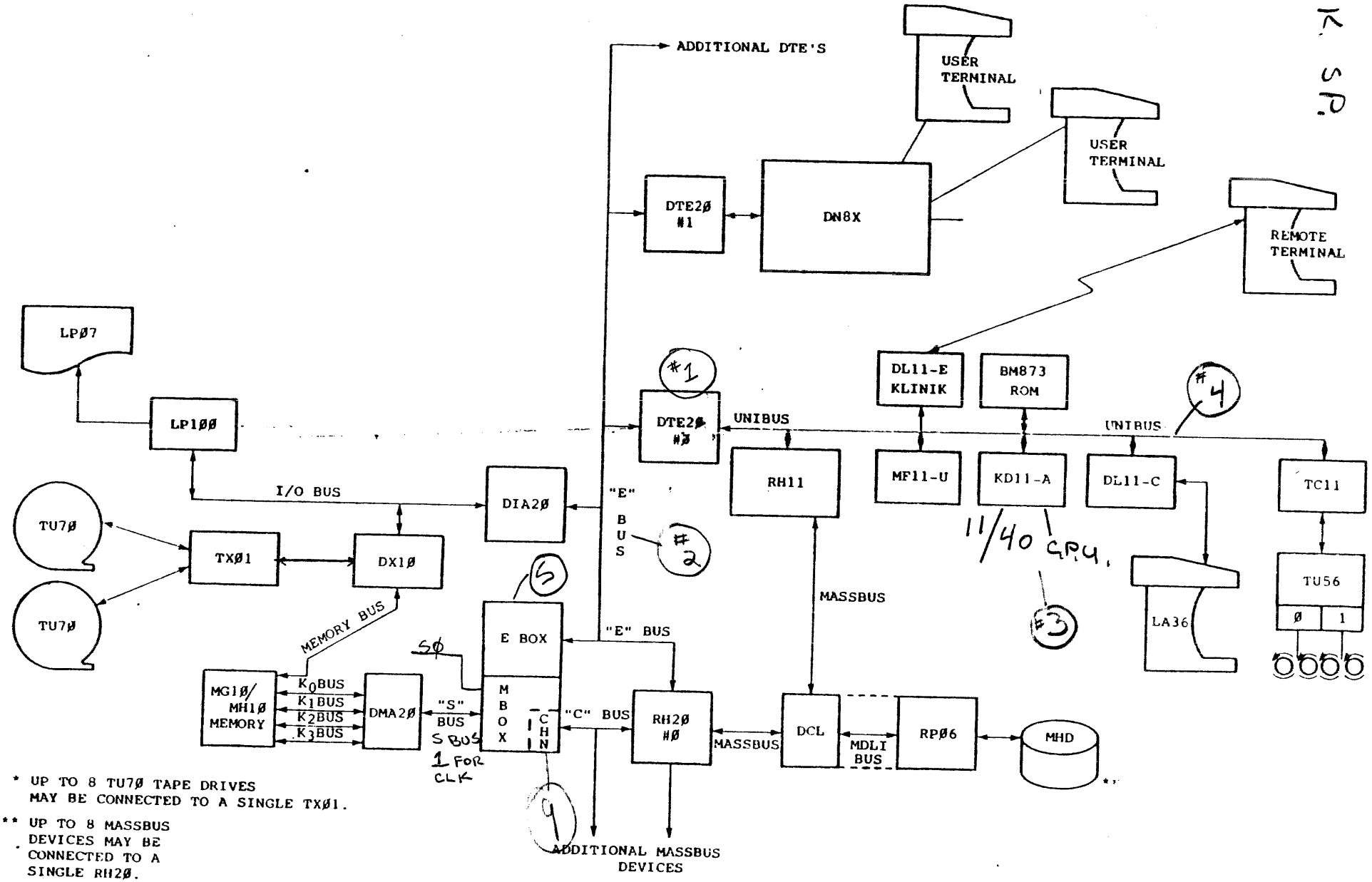


Figure 3. DECsystem-1Ø Block Diagram TOPS 10 SOFTWARE

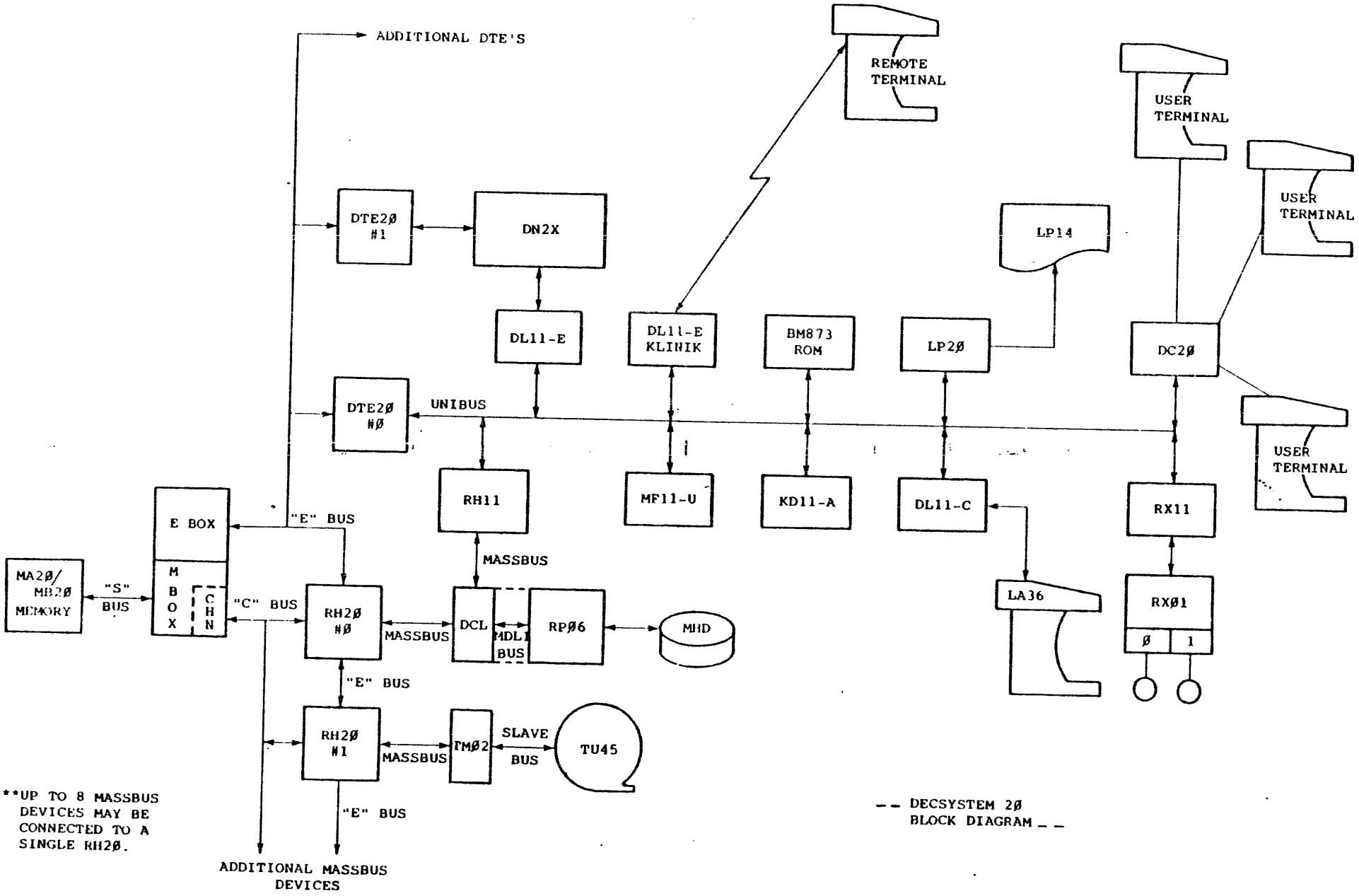


Figure 2. DECSYSTEM-2Ø Block Diagram

3. The accounting meter which counts EBox clock ticks and MBox references with two separate counters

4. The performance analysis counter which is used in evaluating the performance of the system.

4.2.15 External and Internal I/O Controllers and Devices (Typical)

Every device controller on the EBus and I/O bus has a 7-bit device selection network, a priority interrupt assignment, and at least two flags, busy and done, or some equivalent. The selection network decodes bits 3–9 of the instruction so that only the addressed device responds to signals sent by the KL10 Central Processor over the EBus. To use the device with the priority interrupt system, the program must assign a channel to it. Then whenever an appropriate event occurs in the device, it requests an interrupt on the assigned channel.

The busy and done flags together denote the basic state of the device. When both are clear the device is idle. To place the device in operation, a CONO or DATAO sets busy. If the device will be used for output, the program must give a DATAO that sends the first unit of data – a word or character depending on how the device handles information. When the device has processed a unit of data, it clears busy and sets done to indicate that it is ready to receive new data for output, or that it has data ready for input. In the former case, the program would respond with a DATAO to send more data; in the latter, with a DATAI to bring in the data that is ready. If an interrupt channel has been assigned to the device, the setting of done signals the program by requesting an interrupt; otherwise the program must keep testing done to determine when the device is ready.

4.2.16 Machine Instructions

Two basic types of instruction are implemented for the KL10 Central Processor. They are those that reference memory and execute a particular operation such as data transmission, logical operations, arithmetic operations, or program control operations and those that reference memory and perform I/O operations for internal devices and for devices connected to the EBus (refer to Section 3, System Features).

4.3 CONSOLE PROCESSOR SUBSYSTEM

The console processor is a unified hardware/software subsystem that replaces the KA10/KI10 based standard console I/O equipment (which is used for read-in operations), the computer operator console, and the three arithmetic processor indicator panels. To replace this hardware, the console processor provides an automatic bootstrap mechanism and a console command facility. The console processor which incorporates a PDP-11 minicomputer system interfaces with the central processor through the DTE20 Ten-Eleven Interface which is set up to operate in its privileged mode.

The bootstrap facility automatically initializes the dispatch and control RAMs in the EBox, places a small sequence of KL10 code into KL10 memory, and starts its execution to load the system. Several bootstrap options including an operator/software dialogue are available. The console command facilities are available through the console terminal (CTY) to permit the operator to examine or deposit KL10 memory locations and otherwise control and monitor the processor in the same way previously provided by the replaced operator console and indicator panels.

The PDP-11, used as the console processor, is a 16-bit, general purpose, microprogrammed minicomputer which uses single- and double-operand instructions and 2's complement arithmetic. Included in the console processor are the associated PDP-11 controllers and peripheral devices.

The instruction word format is such that the processor can directly address up to 32K words (64K bytes) of core memory. Only 28K are available for program storage. The remaining 4K are reserved for peripheral and register addresses. All communication among system components (including processor, core memory, and peripherals) is performed over the Unibus. Because of the Unibus concept, all peripherals are compatible, and device-to-device transfers can be accomplished at the rate of 2.5 million 16-bit words or 8-bit bytes per second. All system components and peripherals are linked by the Unibus and all peripherals are in the basic system address space. Most instructions applied to data in memory can also be applied to data in peripheral device registers, enabling peripheral device registers to be manipulated as flexibly as memory.

4.3.1 Devices

The following paragraphs provide a brief description of each component that is part of the console processor subsystem.

4.3.1.1 KD11-A Central Processor – This device is the basic component of the console processor. The KD11 is connected to the Unibus as a subsystem and controls time allocation of the Unibus for peripheral devices and performs arithmetic and logic operations through instruction decoding and execution. The instruction set is implemented through a group of hardware subroutines stored within the 256- by 56-bit read-only memory (ROM).

4.3.1.2 KY11-D Programmer's Console – This device is an integral part of PDP-11 system and provides the operator with a direct system interface through the control switches and display indicators. The control switches provide the operator with real-time control of normal program and diagnostic operations, thus allowing the operator to start, stop, load, modify, or continue a program.

4.3.1.3 KW11-L Line Clock Option – This device provides a method of referencing real-time intervals by generating a repetitious interrupt request to the KD11. The rate of interrupt is derived from the ac line frequency.

4.3.1.4 MF11-UP Memory – This device is a read/write, random access, coincident current, magnetic core type memory with a maximum cycle time of 980 ns and a maximum access time of 425 ns. It is organized in a 3D, 3-wire planar configuration. Storage capacity is 16,384 words (32,768 bytes), having an 18-bit word length (16 data bits plus two parity bits).

4.3.1.5 MM11-UP Memory – This device is a 16K word expansion memory having characteristics identical to the MF11-UP.

4.3.1.6 BM873-YD/YG ROM Loader Module – This module is an essential part of the console processor. It provides 256 words of read-only memory which are blasted to contain a number of routines to facilitate bootstrap, power fail, and dump operations. In addition to these routines, the ROM module contains four locations which serve as entry points for these routines. The routines in the ROM facilitate bootstrapping and dump operations to be initiated from the KL10 via the DTE20 10/11 Interface and by pressing a physical LOAD pushbutton on the margin check panel (refer to *Power System Description Manual*).

Either the PDP-11 based DECtape or the PDP-11 based RJP04/06 disk may be specified for the bootstrap or dump procedure.

NOTE

These storage devices are not supported as system devices.

4.3.1.7 DL11-C Asynchronous Line Interface – This device provides simultaneous 2-way transmission between the console terminal and the Unibus. The DL11-C is a character-buffered interface which controls and translates serial bit stream data from the terminal to parallel character data for transfer over the Unibus. The interface also provides parallel to serial translation from the Unibus to the terminal.

4.3.1.8 DL11-E Asynchronous Line Interface – This device has the same general characteristics as the DL11-C, and in addition provides control functions for a communication modem (e.g., Bell Model 103) that interfaces with the KLINIK diagnostic facility.

4.3.1.9 LA36 Keyboard Terminal – This device is a 30-character per second, serial upper/lowercase character printer with a 132-column format. The terminal interfaces to the Unibus through the DL11-C and is used as the main console terminal (CTY).

4.3.1.10 RJP04/06 Disk File System – This system consists of an RP04/06 disk drive and an RH11 device controller. The RP04/06 is a dual Massbus port, moveable head, disk pack drive used as the prime storage media for KL10 and PDP-11 diagnostic and bootstrap load routines. The RH11 provides the control and parallel data path interface between the Unibus and RP04/06 through the Massbus. In addition to communicating with the KD11, the controller has access to PDP-11 memory to fetch and store data.

NOTE

The drive is not supported as a system device from the PDP-11 side but is supported as a system storage device from the KL10 side. (Refer to Secondary Memory Subsystem Description.)

4.3.1.11 TC11-G Magnetic Tape System – This system consists of a TC11 control unit and TU56 tape transport. The TC11 controls transport selection and tape motion and direction. In addition it controls transport read and write operations, and buffers data transferred in either direction. The TU56 is a dual unit, bidirectional tape transport which handles 3/4-inch magnetic DECtape. The transport uses the Manchester-phase recording technique with redundant recording. The tape serves as an auxiliary storage media for KL10 and PDP-11 diagnostic and bootstrap load routines; it is not supported as a KL10 system device.

4.3.1.12 BC11-A Unibus – The Unibus is a 120-conductor ribbon cable connecting the console/front-end processor system components. The Unibus consists of 56 signal and 64 ground lines assembled alternately within the cable to minimize crosstalk.

4.3.1.13 DTE20 Ten-Eleven (Console/Front-End Processor) Interface – This device connects the KL10 central processor with the PDP-11 console/front-end processor to provide the system with interprocessor interrupt, examine, deposit, byte transfers, diagnostic, and boot facilities.

On the EBus, the DTE20 appears in some respect as a KL10-based DECsystem-10 device controller. On the Unibus it connects as a standard PDP-11 peripheral device, using the direct memory access and vector interrupt features of the PDP-11.

Introduction to Maintenance Software

GENERAL DESCRIPTION AND USE

Computer software is divided into three major categories: system software, application software, and maintenance software. Refer to Figure 1.

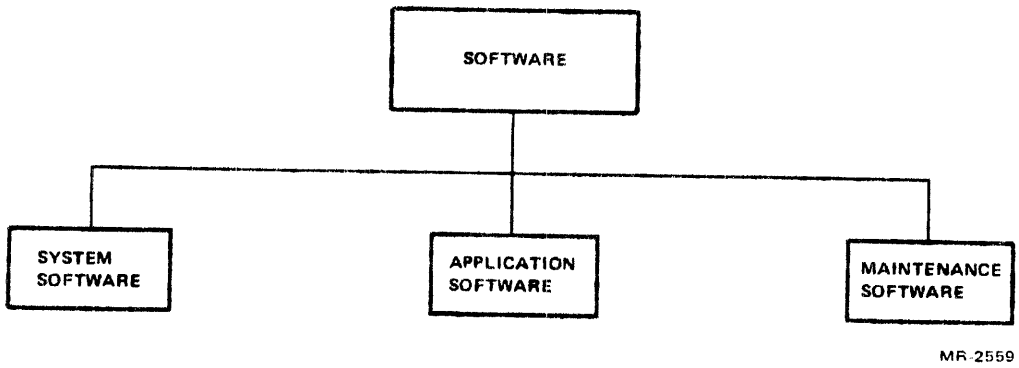


Figure 1. Three Major Categories of Software

The term maintenance software refers to a group of programs specifically designed to aid engineering, manufacturing, and field personnel in determining the operational status of system hardware. In this respect maintenance software is used:

1. During the prototype stages of hardware design to determine if the system, subsystem, or unit under development operates properly and as expected. During this time both the hardware and the maintenance software may undergo any number of changes or revisions.

2. During manufacturing and final system integration to assure that the product functions properly before shipment.

3. During installation to double check manufacturing and to correct any faults that may have occurred during shipment.

4. During customer acceptance to demonstrate to the customer that the hardware is operating properly.

5. During preventive maintenance to assure that the system is fully (100%) operational. Maintenance software is also used at this time to identify potential problems and problems that can be deferred to a later, more convenient, date for correction.

6. During corrective maintenance to detect and isolate the cause of the hardware fault.

7. Following corrective maintenance to verify that all problems have been identified and resolved and that the system is fully (100%) operational.

MAINTENANCE LIBRARIES

Maintenance software is organized into maintenance libraries. The libraries are identified by the base processor that executes the programs and by the type of system the programs are designed to maintain. This course, for example, describes three maintenance libraries. Refer to Figure 2.

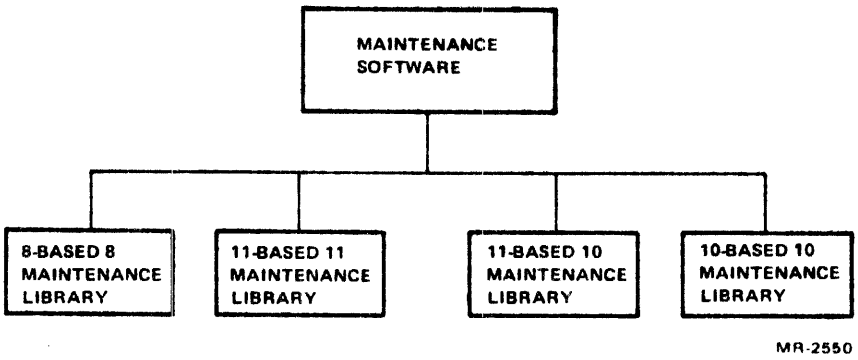


Figure 2. Maintenance Libraries

- 11-Based 11 Maintenance Library - The 11-Based 11 Maintenance Library is written in PDP-11 machine language and is used to diagnose faults in PDP-11 based systems and subsystems.

- 11-Based 10 Maintenance Library - The 11-Based 10 Maintenance Library is written in PDP-11 machine language and is executed by the KL10 console front-end subsystem. It is used to diagnose faults in the KL10 processor, main memory, and channels.

- 10-Based 10 Maintenance Library - The 10-Based 10 Maintenance Library is written in PDP-10 machine language and is used to diagnose faults in the PDP-10 processor, storage, and I/O subsystems.

Maintenance libraries have three component parts: utility programs, control files, and diagnostic programs. Utility programs and control files simplify loading, running, and maintaining the diagnostic programs and the diagnostic storage media. The diagnostic programs are used to detect and isolate hardware faults and to verify the operational status of ths system. Refer to Figure 3.

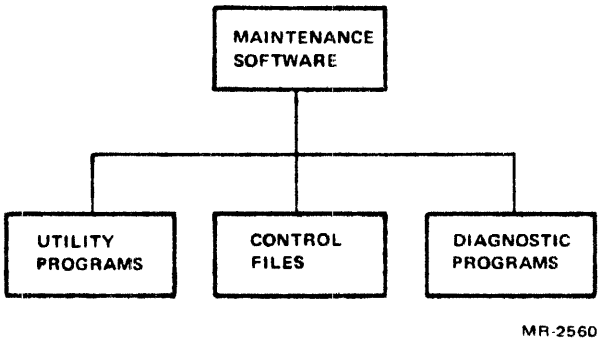


Figure 3. Component Parts of a Maintenance Library

Utility Programs

There are six basic types of utility programs.

1. Bootstrap Loaders - A bootstrap loader is a short, two-part program. The first part either resides permanently in a read-only memory (ROM) or is manually deposited into memory. The second part resides in the boot block (block 0) of the input

device. The first part consists of a few instructions which, when executed, will read in and start the second part. Once started, the bootstrap loader can be used to load and start a larger, more sophisticated program such as a diagnostic monitor.

2. **Downline Loaders** - A downline loader is a program that is used to transfer a utility or diagnostic program from the host system to another system or subsystem for execution.

3. **Diagnostic Monitors** - A diagnostic monitor is a special purpose loader that enables the user to:

   a. Directly load and run a single utility or diagnostic program

   b. Indirectly (via a control file) load and run a sequence or hierarchy of diagnostic programs.

4. Program Maintenance Utilities - This type of utility program is used to construct and modify (patch, update, etc.) control files and diagnostic programs.

5. File Maintenance Utilities - This type of utility program is used to duplicate and update diagnostic storage media (e.g., disk, tape, etc.).

6. **Hardware Utilities** - A hardware utility is a utility program that supports maintenance or diagnostic functions. The 11-Based 10 Maintenance Library, for example, uses utilities of this type to configure the main memory subsystem and to read and change the internal status of the KL10 processor, memory, and channels. Although this type of utility program performs maintenance and diagnostic operations, it is not a diagnostic in the true sense of the word. That is, it does not automatically detect and isolate hardware faults.

Utility programs are relatively easy to use because they are controlled by task-oriented commands. And, because the commands are task-oriented, they can easily be summarized in tables similar to those used in the KL10 Maintenance Guide. (See example below). The table lists the base commands in alphanumeric order, provides an example which illustrates the proper command format, and briefly describes the task

the command performs. A cross reference may be used when a more detailed description of the command is necessary. This example is from TRACON, an 11-based 10 utility program.

EXAMPLE:

Table 1 TRACON Control Function Summary

Command	Description	Cross Ref.
A	A<CR> Auto insert - automatically builds an internal command file as commands are typed.	1
E	E<CR> Edit or create a command buffer. Refer to Table 2.	N/A

TRACON COMMAND DESCRIPTION (Cross Reference)

This section describes in detail each of the commands summarized in Table 1.

1. A<CR> - The A command opens the control buffer for input. All commands typed following the A command are entered into the buffer until a KA command is typed. The commands in the buffer are executed via the X, L or M command. The buffer may be saved for future reference with the DC command.

NOTE

For further information about TRACON commands, refer to the 10-Based 10 section of the KL10 Maintenance Guide.

Control File

A control file (sometimes referred to as a chain or script) is an ASCII text file. It is generated using a standard editor program, and contains a list of commands to be executed by another program, usually a utility program. When directed to do so, the program will read the control file into a memory buffer and begin executing the commands it contains as though they were entered directly from the user's terminal.

The following is a typical example of a control file. The file is from the 11-Based 10 Maintenance Library. In the example, KLDCP, an 11-based 10 utility program, is directed to sequence through a series of diagnostic programs (i.e., run two passes of DGDTE, one pass of DGKAA, one pass of DGKAB, etc.). Note that lines preceded by a semicolon are considered comments and are ignored by KLDCP.

EXAMPLE:

```
;B.CMD, KL10 PROCESSOR DIAGNOSTIC BOOT, 9-JUN-78
;PROCESSOR HARDWARE

;DTE20 INTERFACE
P DGDTE.All
SED 2
;EBOX PART 1
P DGKAA.All
SED 1
;EBOX PART 2
P DGKAB.All
SED 1
;MBOX BASIC
etc.
```

Frequently during preventive and corrective maintenance, control files are used to automatically direct a utility program to sequence through a series (hierarchy) of diagnostic programs. Control files used in this manner assure that the proper programs are run in the correct order and for the prescribed period of time.

In most cases the standard control files shipped with a maintenance library are limited to testing the processor, main memory, and channels. Shipping standard control files for the rest of the system is usually not feasible because of the variety of storage and I/O subsystems that could be used in the final system configuration. Therefore, the control files for testing these subsystems should be generated locally, usually at the site during installation. Once the control files are generated, they can be transferred to the maintenance software storage media and used as necessary.

Diagnostic Programs

There are two major categories of diagnostics: operability tests, which include hardware (FRU) tests and functional tests, and exercisers, which include subsystem exercisers and system exercisers. Refer to Figure 4.

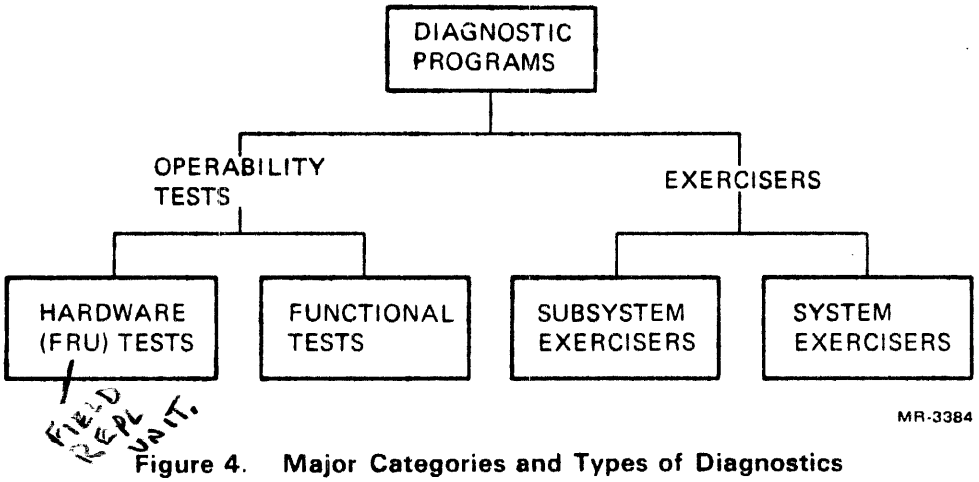


Figure 4. Major Categories and Types of Diagnostics

- Operability tests are intended to detect and isolate solid hardware malfunctions. That is, they are designed to test the basic ability of the hardware to operate.

- Exercisers are intended to detect and isolate intermittent hardware malfunctions; that is, they test for system and subsystem reliability, priority arbitration, and interaction faults that only occur when the system is operating at or near full capacity.

Both of these categories will be discussed in detail later.

Operational Control - There are two principal methods of controlling the operation of a diagnostic program. One is switch control and the other is operator dialogue. Many diagnostics use a combination of both. Refer to Figure 5.

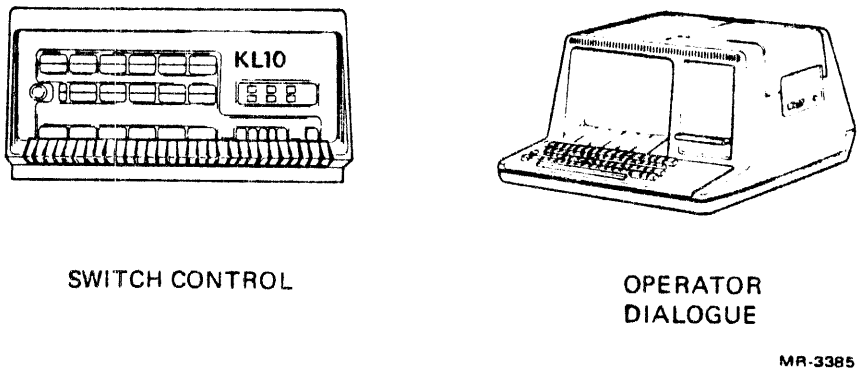


Figure 5. Two Methods of Controlling the Operation of Diagnostic Programs

Switch control is usually implemented through the console data switches. Each switch is assigned a fixed binary program control function (e.g., inhibit printouts, loop on test, halt on error, etc.). The switches are read at the start of the program and may be read periodically by the program while it is running. The program uses the state of the control switches to determine operating parameters and its reaction to various error conditions.

Operator dialogue takes place between the diagnostic and your terminal. It enables the diagnostic to query you and enables you to specify program parameters such as the subsystems and device units to be tested, the type of test to be run, and the data patterns to be used during testing.

The flexible control offered by operator dialogue complements the more rigid control offered by switches. Thus many diagnostics, particularly those in the exerciser category, use the switches to specify standard binary control functions and the operator dialogue to specify variable control functions.

Internal Structure - Although no rule states that all diagnostic programs must use a standard internal structure, most do. The structure consists of four parts: a program initialization and control routine, a main diagnostic segment, a set of service routines, and a storage area for fixed and variable data. Refer to Figure 6.

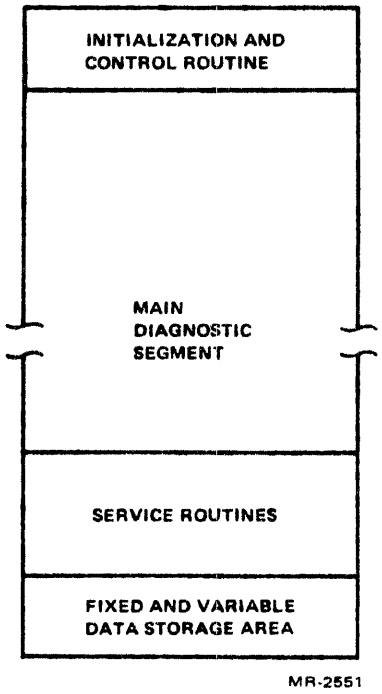


Figure 6. Internal Structure of Diagnostic Programs

- **Program Initialization and Control Routine** This routine initializes the system, reads and stores the state of the program control switches, performs operator dialogue, and dispatches to the main diagnostic segment of the program.

- **Main Diagnostic Segment** The main diagnostic segment consists of a set or series of exerciser or test routines. The set of exerciser routines that make up exercisers are inextricably intertwined (complex, to say the least). This is particularly true at the machine language level. On the other hand, the series of test routines that constitute operability tests are relatively easy to understand. This is true both at the conceptual level and at the machine language level. The internal organization and structure of operability tests will be discussed later.

- **Service Routines** Service routines are used by both the program initialization and control routine and the routines that make up the main diagnostic segment. Service routines are used to handle keyboard input and printer output, generate data patterns, load and unload memory buffers, format error messages, etc. Separating service routines from the main program and making them available to both the initialization and test routines has several advantages. For example, it simplifies programming, eliminates redundancies, helps standardize the diagnostics, and makes it significantly easier to read the program listing, should that become necessary.

- **Fixed and Variable Data Storage Area** The fixed storage area is used to store program constants such as fixed ASCII messages and test operands. The variable data storage area provides temporary buffer areas for storage of terminal input and output, program stack operations, various test data patterns, and so forth.

Operability Tests - Again, hardware (FRU) and functional operability tests are designed to detect and isolate solid hardware faults. That is, they are designed to test the basic ability of the hardware to operate. The major difference between these two types of diagnostics is their degree of fault resolution or isolation.

- **Hardware (FRU) Tests** Hardware (FRU) tests attempt, through a logical analysis of the symptoms, to isolate the faults to the specific field replaceable unit (FRU) causing the failure. Thus, in most cases, the need to analyze hardware (FRU) tests at the machine language level is eliminated. The exception, of course, is when the test detects a fault but is unable to identify the failing FRU. In this case an analysis of the test would be required.

- **Functional Tests** Functional tests, on the other hand, only isolate to the failing function (i.e., instruction or operation). From that point the technician must analyze the failing test or subtest at the machine language level and determine which component or field replaceable unit is causing the failure.

- **Main Diagnostic Segment** The main diagnostic segment of an operability diagnostic (as mentioned earlier) consists of a series of individual test routines. Each test routine is designed to check one specific hardware function. When a function can be checked in more than one way (e.g., an adder or a multiplexer with several data and gating inputs) then the test routine for that function is usually divided into subtests. Each subtest will in turn, test one aspect of the function until the function is completely tested. Refer to Figure 7.

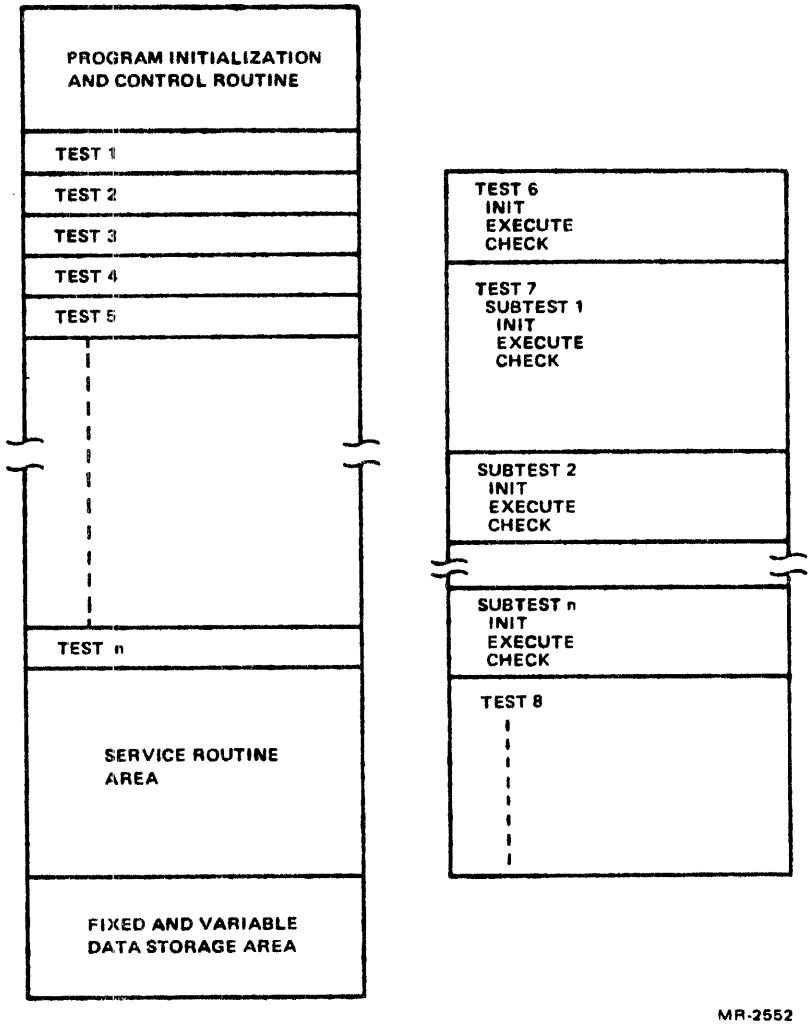


Figure 7. Internal Structure of Operability Tests

Each test or subtest in the main diagnostic segment of an operability diagnostic is composed of three major sections of machine language code.

1. **INIT** - The INIT or initialization code preconditions the hardware for testing. This may involve tasks such as resetting the hardware, checking to assure that no error conditions already exist, setting and clearing status bits, generating test operands, and building channel command lists and data buffers. Service routines are frequently used to accomplish many of the tasks involved in test initialization.

2. **EXECUTE** - The execute code usually consists of a machine language instruction that will cause the function or operation under test to activate. Occasionally two or more instructions may be required for this purpose.

3. **CHECK** - The checking code determines if the hardware responded properly. Most often this involves reading hardware status registers and checking buffers for correct data. Typically, an error checking service routine is used for this purpose. The expected results of the test are passed to the error checking service routine by the test INIT code. The actual test results are passed to the routine by the CHECK code. The error checking service routine compares the expected results with the actual results and determines if an error has been detected.

   ERROR - If an error is detected, the program may respond in a number of ways. For example, it may print an error message, ring the bell on the user's terminal, halt, go on to the next test, or loop on the failing test. How a program responds to an error depends largely upon the state of the program control switches and initial operator dialogue.

   NO ERROR - If no errors are detected, program control will generally proceed to the next test or subtest in the sequence. The program will continue in this manner until either an error is detected, or until each hardware function or FRU has been completely tested for solid faults.

In summary, operability tests are designed to systematically test each individual instruction, operation, or function that the hardware is capable of executing. Thus, they are best suited to detect solid hardware faults and isolate the cause to the failing function or field replaceable unit (FRU).

Exercisers - As mentioned earlier, subsystem exercisers and system exercisers are similar both in design and in use.

- **Subsystem Exercisers** Subsystem exercisers are designed to generate maximum subsystem interaction by operating the control unit, data channel (if present), and one or more storage or I/O devices at or near full capacity. They may be used to localize a subsystem failure to the control unit, channel, or device causing the failure. However, they are primarily intended to troubleshoot (intermittent) reliability, internal priority arbitration, and device and device bus interaction faults.

- **System Exercisers** System exercisers are designed to generate maximum system interaction by operating the processor, storage, and I/O subsystems at or near full capacity. They may be used to localize a system malfunction to the subsystem or device causing the failure. However, system exercisers are primarily intended to troubleshoot (intermittent) reliability, priority arbitration, and subsystem interaction faults.

In summary, when using exercisers keep these points in mind:

1. Exercisers are designed to troubleshoot the so-called intermittent class of failure. They should never be used to troubleshoot solid faults or faults that can be detected and isolated using an operability test.

2. Exercisers operate on the assumption that certain hardware error checking logic is functional (e.g., parity checking networks, time-out logic, etc.). In many cases exercisers use this logic as an indication that an error has occurred. Therefore, before using an exerciser for troubleshooting purposes, use the appropriate operability test to verify that no solid hardware malfunctions exist in the error checking logic.

3. Exercisers typically provide extensive operator dialogues and comprehensive error message reports. They do this because the internal structure and control of exercisers tends to be complex. When an exerciser must be used for troubleshooting, use the flexibility of the operator dialogue, the information provided by each error message, and deductive reasoning to determine the cause of the problem. In other words when possible, avoid using troubleshooting techniques that require analyzing the exerciser at the machine language level.

Figure 8 summarizes the overall external structure of maintenance software.

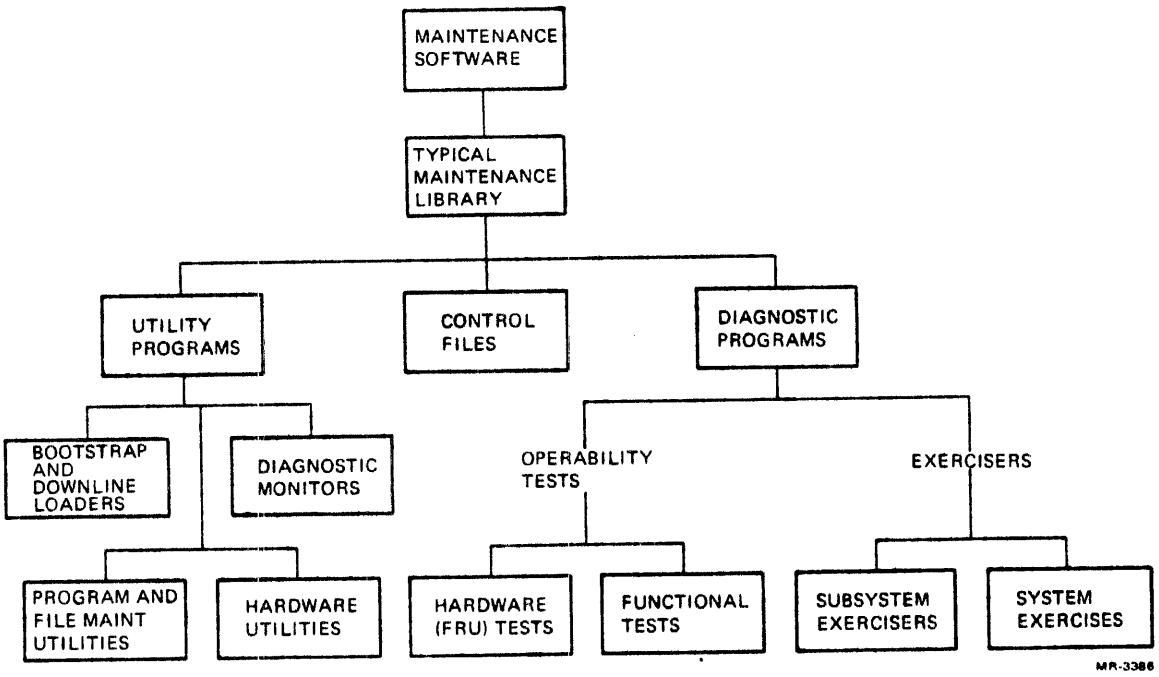


Figure 8. The Big Picture

- Product code
- Product name
- Version of the listing
- Date the program was released
- Name of the group responsible for maintaining the program
- Author(s) of the program.

2. Text Section - The text section consists of the:

   - Abstract (a brief description of the program)
   - System requirements and prerequisite programs
   - Switches used to control the operation of the program
   - Loading, starting, and operating procedures
   - Command or test descriptions.

   The text section may also include additional information that the author feels is necessary to use or maintain the program.

3. History Section - The history section describes each change or revision made to the program since its initial release.

4. Document Section - The document section outlines the internal organization of the program.

   The document section is generated by a program called DECDOC. DECDOC searches through the machine language listing and extracts the subtitle statements and pertinent comments pertaining to each subroutine or section of the program. The listing line number associated with each statement or comment is also extracted. Thus you can review the document file to grasp the big picture. Then, if you need more detail, the listing line number can be used to index into the machine language listing.

5. Flowcharts - Flowcharts may also be included to describe some of the more complex routines.

The Machine Language Listing - The machine language listing consists of the following six sections:

1. Program Title and Definition Section - The program title and definitions section defines the assembly switches, program address options, Unimplemented User Operations (UUOs), EMulator Traps (EMTs), accumulators (ACs), macro statements, device codes, I/O bit assignments, and registers.

2. Program Parameters (PARAM) Section - PARAM defines program operating parameters such as control switches, special subroutine linkages, and OPDEFs.

3. Fixed Control and Dispatch Storage (FIXED) - FIXED contains ASCII messages, dispatch tables, push down list control information, various flag definitions, etc.

4. Main Code Section - This is the main diagnostic program. It contains the program initialization routine, the dispatch routine, and the test or exerciser routines. The service routines are usually located at or near the end of this section.

5. Fixed and Variable Storage Area (STOR) - The STOR area is reserved for patches, channel lists, error messages, input and output buffers, data patterns, etc.

6. Cross Reference (CREF) - The CREF lists, in alphabetic order, each symbol or symbolic address used by the program. It lists every listing line number which references the symbol as well as the line number at which the symbol is defined.

In esense, the program description tends to be quite detailed because it must meet the needs of both the users (i.e., manufacturing and field maintenance personnel) and the diagnostic programmer responsible for maintaining the program after it is released. Typically, the text section fulfills the needs of the user, and the history, document, and flowcharts fulfill the needs of the program maintainer. The machine language listing fulfills the needs of anyone requiring a detailed understanding of the internal operation of the program.

NOTE

Do not confuse the organization of the diagnostic listing with the internal structure of the diagnostic program described earlier.

Diagnostic Hierarchies

The diagnostic programs in a maintenance library are organized into hierarchies. A hierarchy defines the order in which a set of related diagnostics must be run in order to assure that a unit or subsystem is thoroughly tested. There is a diagnostic hierarchy for the central processing unit as well as one for each of the storage and I/O subsystems. Refer to Figure 10.

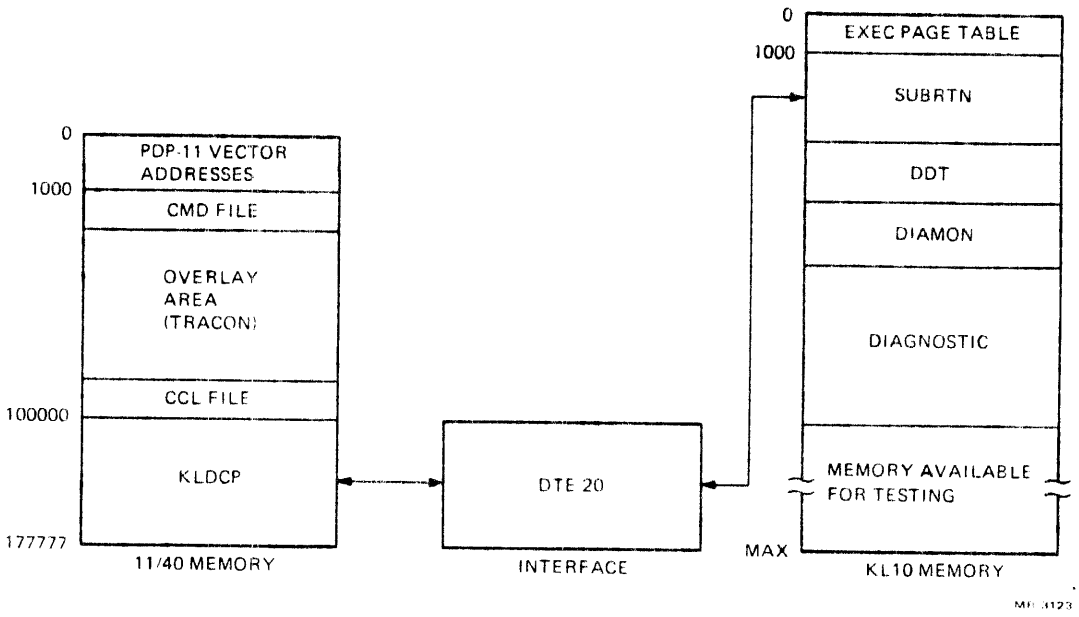


Figure 10. Typical Diagnostic Hierarchies

Ideally each diagnostic hierarchy consists of a:

- Hardware (FRU) test to verify that the FRUs (i.e., gates, storage elements, flip-flops, MOS memory, etc.) work independently. (There are no solid faults.)

- Functional test to verify that the individual gates and storage elements, etc., work collectively to perform their assigned functions. (Again, there are no solid faults.)

- Unit or subsystem exerciser to verify that the functions work together reliably to:

  a. Process data and control the overall operation of the system (CPU exerciser).

  b. Store and retrieve data (storage subsystem exercisers).

  c. Input and output information to the user (I/O subsystem exercisers).

At the top the system exerciser ties all the hierarchies together by verifying that CPU, storage, and I/O subsystems work together reliably as a computer.

This, of course, is an ideal diagnostic hierarchy. In practice, the hierarchy may vary from library to library. For example:

1. Some libraries do not support a full-fledged system exerciser. Instead, they rely on the operating system to perform this function. Each time the operating system detects an error, it captures the error information and records it in an error file. Later using a system library program, the error file can be printed out and the information analyzed. Thus, the operating system becomes, in part, an on-line system exerciser.

2. Sometimes the hardware (FRU) test and the functional test will be combined in one program. Occasionally, the exerciser will also be included in the program. Thus it may appear that there is only one diagnostic program for a particular subsystem. This combining technique is frequently used with small I/O

subsystems (i.e., line printers, card readers, terminals, etc.). Since these types of subsystems support a limited number of functions, it is generally more efficient to merge all testing into one program.

3. Frequently, large operability tests such as those for the central processing unit, disk, and tape subsystems, are divided into several parts. Sometimes this is done to accommodate systems with a limited amount of main memory. Other times the diagnostics are divided into functional areas of logic for convenience (For example, Part 1 of a processor diagnostic may test the internal word and byte transfers, Part 2 may test boolean instructions, Part 3 may test arithmetic instructions, etc.). In many cases, dividing diagnostics into parts accounts for the seemingly large number of individual diagnostic programs in many maintenance libraries.

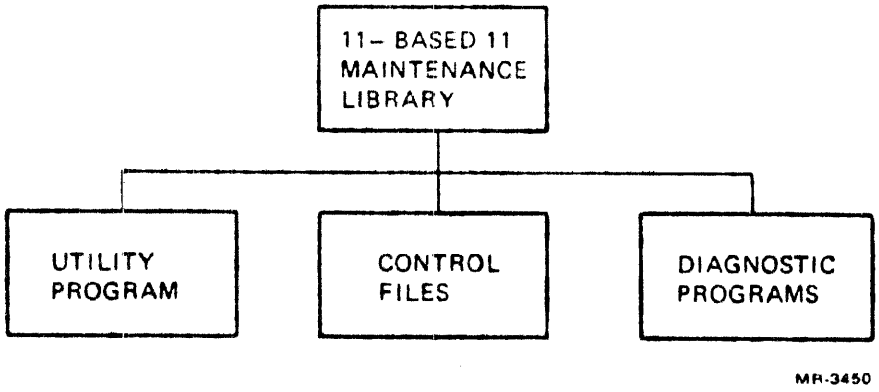
4. Most maintenance libraries do not support a full set of hardware (FRU) tests. There are two reasons for this. First, hardware (FRU) tests require a working processor to analyze symptoms and isolate the field replaceable unit causing the failure. Thus, hardware (FRU) tests for processors are generally restricted to those systems that have a separate maintenance console subsystem (i.e., KL10s, VAX11/780s, etc.). Secondly, because some storage and I/O subsystems are composed of a limited number of field replaceable units (three or four), their size does not warrant hardware (FRU) diagnostics.

5. Finally, it is worth mentioning that the title or name given a particular diagnostic does not necessarily conform to the four generic types of diagnostics mentioned earlier (i.e., system exerciser, subsystem exerciser, functional test, and hardware FRU test). The title given a diagnostic attempts instead to describe what the diagnostic does. For example, the following are some typical titles given diagnostics:

    Basic Instructions #1
    CPU/PI/Memory Reliability
    Memory Port Interaction
    DQ11 Basic Logic Test Part 2
    KG11 Communications CRC Test

Note that the title of a diagnostic does not explicitly state its relative position in a hierarchy. For this reason either the diagnostic listing or a table of diagnostic hierarchies must be consulted in order to assure that the proper tests are run in the proper order. Refer to the 10/10 STD module in the 10-Based 10 section of the KL10 Maintenance Guide for some examples of diagnostic hierarchy tables.

The 11-Based 11 Maintenance Library



GENERAL DESCRIPTION

The 11-Based 11 Maintenance Library is a standard maintenance software library. The library is written in PDP-11 machine language and is designed to detect and isolate faults in PDP-11 based systems and subsystems.

11-BASED 11 UTILITIES PROGRAMS

The library contains four major utility programs: XXDP, XTECO, UPD, and COPY. See Figure 1.

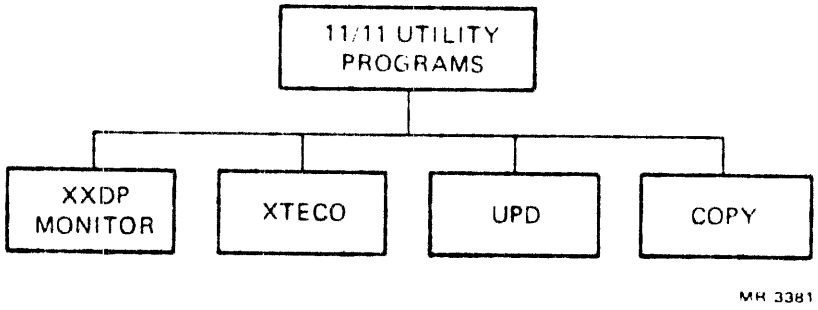


Figure 1. The Four Major 11-Based 11 Utility Programs

- **XXDP** - XXDP is the basic diagnostic monitor, where XX is the input media designator, (e.g., RPDP uses an RP11 and disk pack for input, RXDP uses an RX11 and floppy for input, TCDP uses a TC11 and DECtape

for input etc.). Basically the monitor allows you to load and run either a single diagnostic or utility program (via direct commands) or a sequence of programs (via a control or chain file).

- XTECO - XTECO is an editor program. It enables you to construct and modify ASCII text files. Primarily the editor is used to construct and modify control files and short test programs.

- UPD - UPD is a utility program used to update 11-based 11 utility and diagnostic programs.

- COPY - COPY is a utility program used to duplicate maintenance software storage media.

Like most utility programs, the 11-based 11 utility programs are relatively easy to use because they are controlled by task-oriented commands.

STOP

Before proceeding further, review the summary modules for XXDP, XTECO, UDP and COPY in the 11-Based 11 section of the KL10 Maintenance Guide.

11-BASED 11 DIAGNOSTIC PROGRAMS

There are very few hardware (FRU) tests in the 11/11 maintenance library. Most of the diagnostics are either functional tests or exercisers.

INTERNAL STRUCTURE AND CONTROL

The internal structure of the 11-Based 11 Maintenance Library is illustrated in Figure 2.

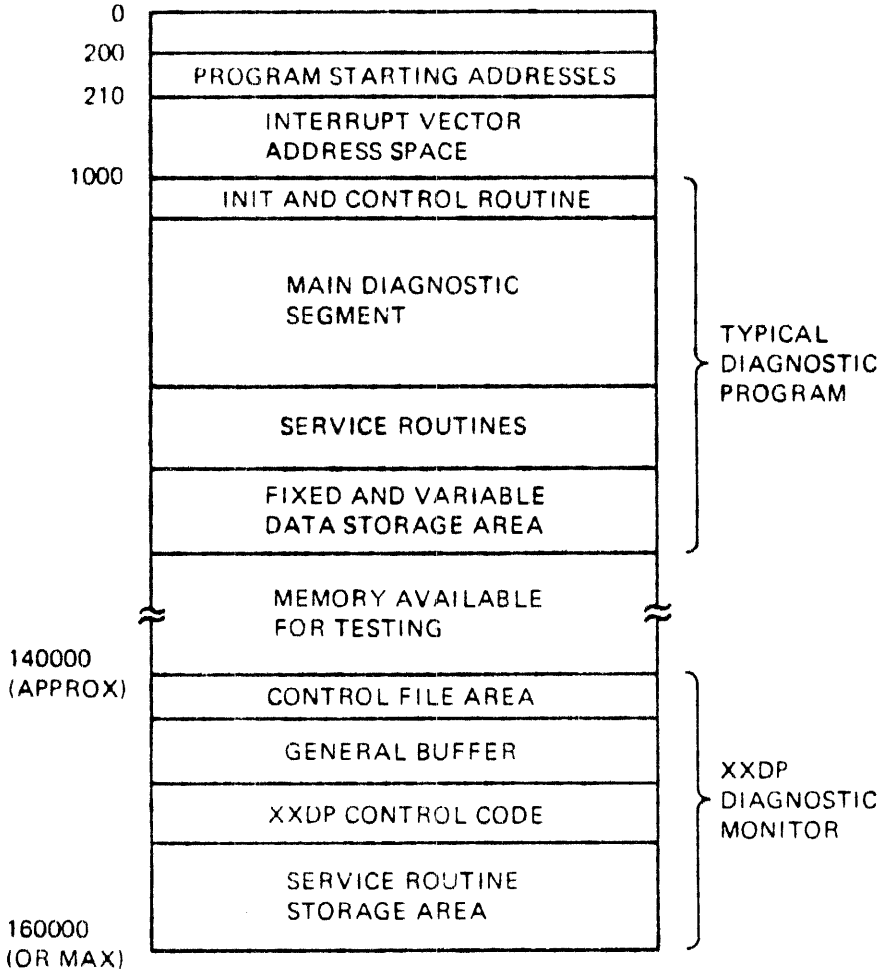


Figure 2. 11-Based 11 Memory Utilization Map

- LOC 0 through 776 - With the exception of locations 200 through 210, locations 0 through 776 are reserved for interrupt vector addresses. Location 200 contains a jump to the starting address of the diagnostic. Locations 201 through 210 are optional starting addresses.

- LOC 1000 - Most diagnostics begin around location 1000. Each diagnostic consists of the standard program initialization and control routine, the main diagnostic segment, the service routines and the fixed and variable data storage area.

- Monitor - The diagnostic monitor (XXDP) consists of approximately 2000 words and resides in the upper 2K of memory. The memory between the end of the diagnostic and the beginning of the diagnostic monitor is available for use by the diagnostic.

- Peripheral and Register Addresses - These locations address the standard PDP-11 registers and peripheral devices. This area is always in the uppermost 4K of PDP-11 addresses.

The standard control flow described earlier in the introductory module are applies to the 11-based 11 diagnostics. That is, the program control and initialization routine initializes the system, reads and stores the state of the program control switches, performs operator dialogue, and dispatches to the test routines. The test or exerciser routines perform the testing using the various service routines as necessary. Faults are reported as they are detected. When testing is complete, control is returned to the diagnostic monitor or calling program.

11-BASED 11 SYSTEM EXERCISER

Because of the wide range of options available on a PDP-11 system a single common system exerciser is not practical. Instead, the 11/11 Library uses a modular exerciser referred to as DECX/11. The exerciser may be assembled in the field to match any standard PDP-11 system configuration.

The exerciser consists of a monitor or control module and a set of individual exerciser modules. Refer to Figure 3 for a detailed view of the memory utilization map.

The exerciser is assembled using a utility program referred to as the Configurator. Via operator dialogue, the Configurator will request that you specify the version of the monitor module to use and the name of each exerciser module to be included in the final version of the DECX/11 exerciser. There are five basic monitors to choose from, each of which supports different features and internal CPU options (e.g., Unibus mapping, memory parity checking, memory management, and cache, etc.). In addition, there are approximately 95 individual exerciser modules, one for each major storage and I/O subsystem available on a PDP-11 system.

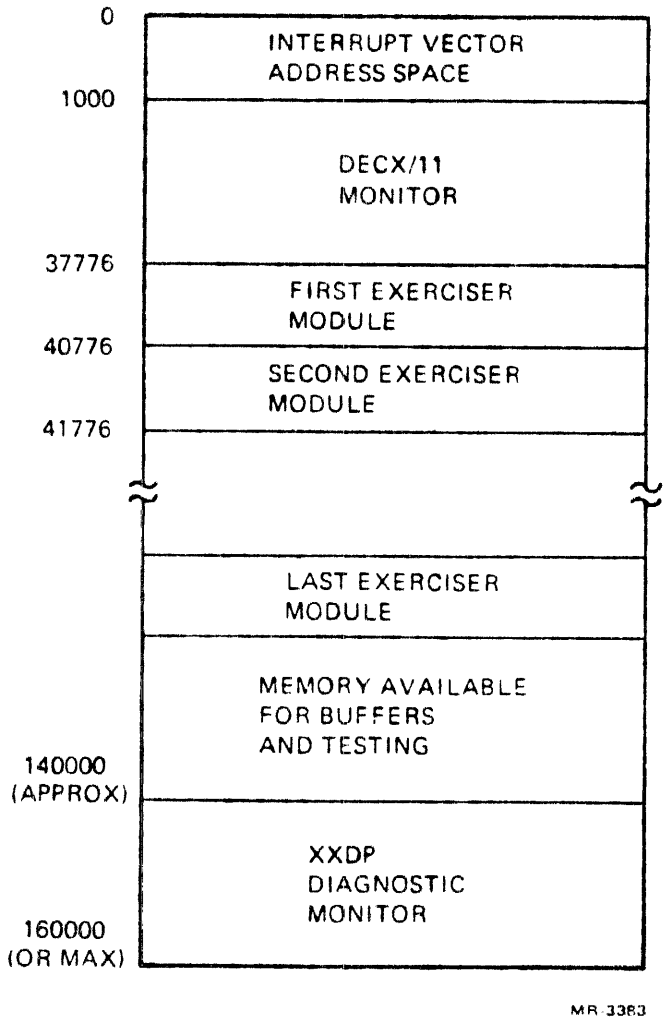
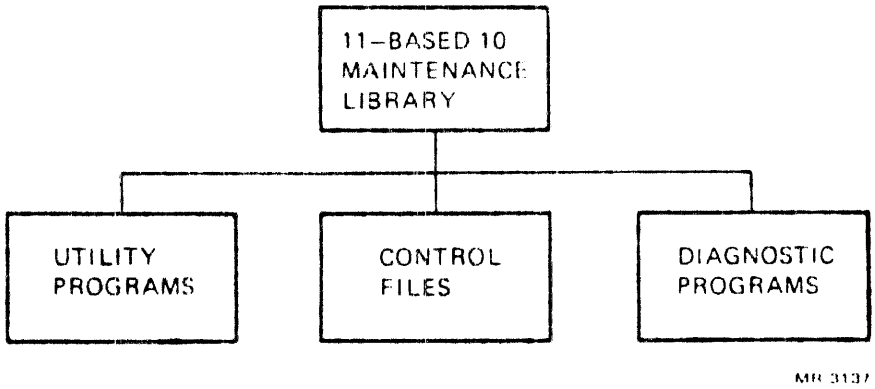


Figure 3. DECX/11 System Exerciser Memory Utilization Map

After all the modules are specified, the Configurator will link them together and automatically save the newly assembled exerciser. The exerciser can now be run under the XXDP diagnostic monitor any time the need arises. Keep in mind, however, that all the appropriate operability tests should be run before using an exerciser for fault isolations.

The 11-Based 10 Maintenance Library



GENERAL DESCRIPTION

The 11-Based 10 Maintenance Library is written in PDP-11 machine language and converts the KL10 operator console subsystem into a diagnostic console subsystem. The KL10 Diagnostic Console Program (KLDCP) performs the conversion. KLDCP is loaded from an 11/40 storage subsystem, DECtape, floppy, or disk (KLAD-10 or KLAD-20). The load operation is controlled by the BM873 ROM in the 11/40 processor. Refer to Figure 1, which illustrates the 11/40 maintenance console front-end subsystem.

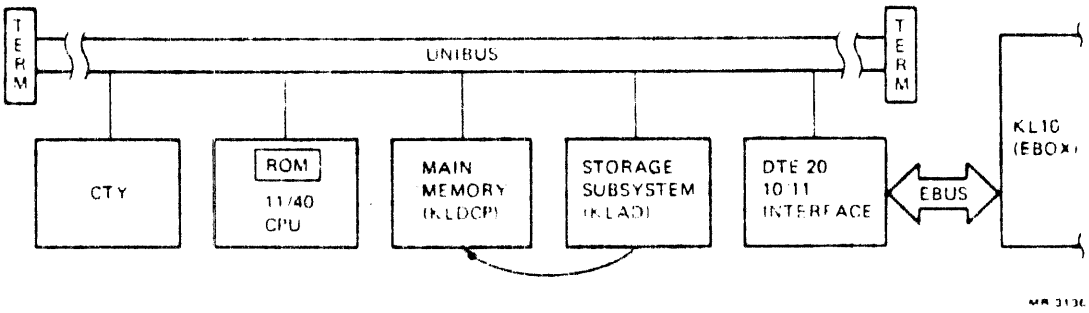


Figure 1. 11/40 Maintenance Console Front-End Subsystem

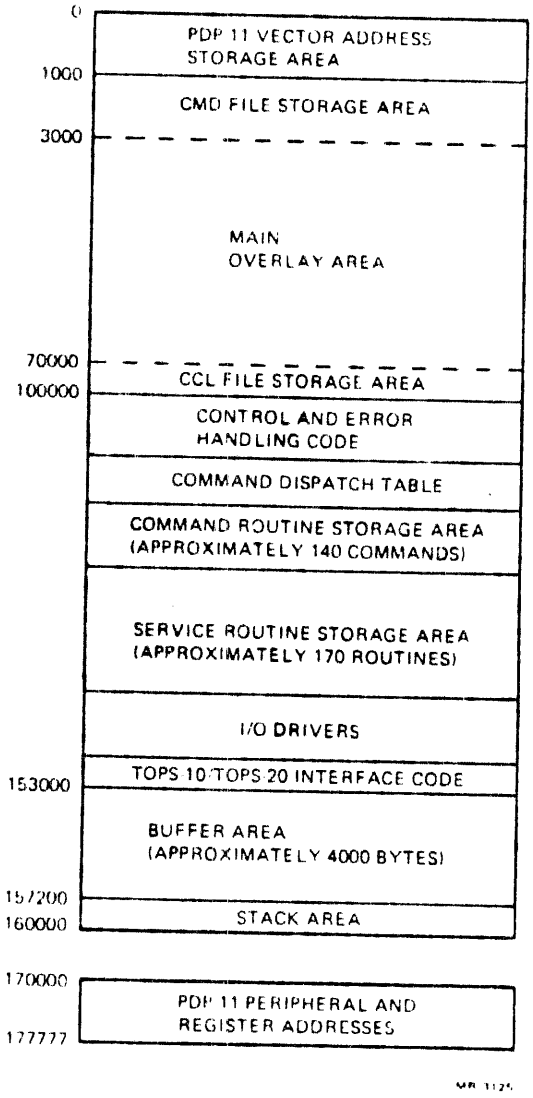


Figure 2. 11/40 KLDCP Memory Utilization Map

\
CONSOLE

- PDP-11 Vector Address Storage Area - This area is used to store the vector addresses associated with the DTE20 and other PDP-11 storage and I/O devices.

- CMD File Storage Area - This section of the overlay area is used to store and execute double-indirect control files. A double-indirect control file is unrestricted. It may be used to execute single 11/10 utility commands, call indirect control files (CCL files), and to load and run 11/10 or 10/10 utility and diagnostic programs.

- Main Overlay Area - This area is used for loading and running 11-based 10 utility and diagnostic programs.

- CCL File Storage Area - This section of the overlay area is used to store and execute indirect control files. Indirect control files are restricted to certain types of 11/10 utility commands. For example, they should not contain commands that will load and run other indirect control files, utility programs, or diagnostic programs. A double-indirect control file should be used for this purpose. This section of the overlay area is also used for storing and executing TIC files and isolation routines. This will be discussed later.

- Control and Error Handling Code - This code initializes the system, controls command processing, and handles errors.

- Command Dispatch Table - Base commands are compared to this table. The table provides the starting address of the command routine.

- Command Routine Storage Area - Approximately 140 command routines are stored here, one for each KLDCP command.

- Service Routine Storage Area - These routines are used by the command routines to perform various tasks (number conversions, checksum operations, message formatting and printing, etc.).

- I/O Drivers - This code handles the 11/40 storage and I/O subsystem input and output operations.

- TOPS10/20 Interface Code - This code interfaces the CTY with the operating system that is directing and monitoring the KL10 (TOPS-10 or TOPS-20). Nothing fancy goes on here. KLDCP simply passes the characters it receives from the CTY to operating system and, conversely, prints operating system responses on the CTY.

- Buffer Area - This is a general buffer area used for temporary storage of input and output information. It can handle approximately 4000 (octal) 8-bit bytes.

- Stack Area - This area is used to keep track of EMTs, TRAPs and PDP-11 priority interrupts.

- PDP-11 Peripheral and Register Addresses - These locations address the standard PDP-11 registers and peripheral devices including the DTE20. This area is always the uppermost 4K of PDP-11 addresses.

Once loaded, KLDCP performs a checksum operation on its fixed code, initializes the system, prints its header and prompt, and waits for a command to be typed on the CTY or KLINIK line. Refer to Figure 3.

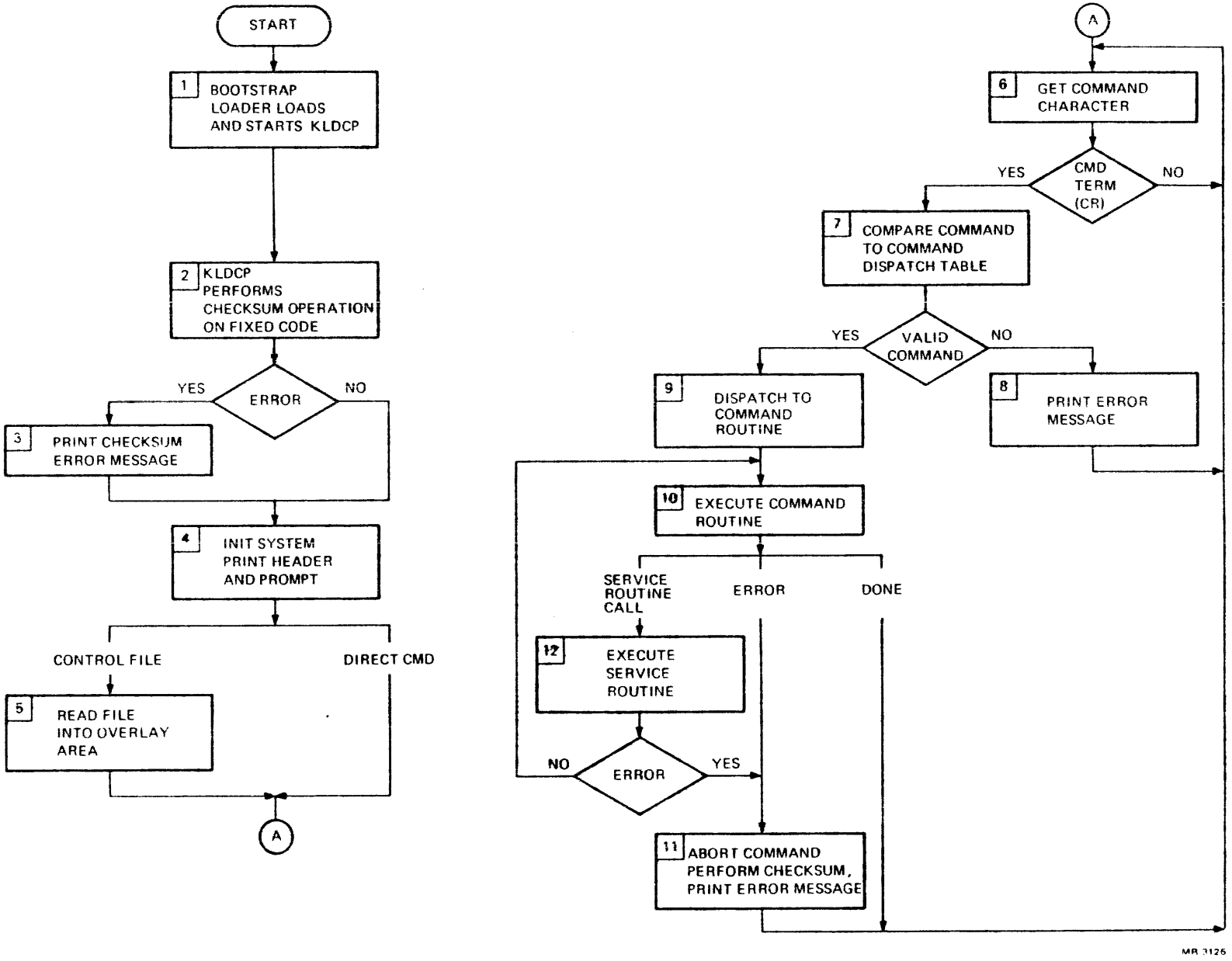


Figure 3. KLDCP Direct and Indirect Command Processing

The following text explains the basic flow illustrated in Figure 3. The numerals in the text correspond to the numerals in the figure.

1. You initiate a read-in operation. The bootstrap loader automatically loads and starts KLDCP.

2. KLDCP performs a checksum operation on its fixed code. The result is compared to a precalculated checksum value.

3. If there is a checksum discrepancy an error message is printed. The newly calculated checksum value replaces the original precalculated value, and KLDCP continues. Checksum errors are not considered fatal. Therefore, you may modify the code, and KLDCP will still run.

4. Independent of a checksum error, KLDCP will initialize the system, and print its header and prompt for a command.

5. If a control file is used, KLDCP will read it into the appropriate section of the overlay area. It will then get its commands (indirectly) from the control file.

6. KLDCP deposits each character of a command in the buffer area. When the command terminator (i.e., carriage return <CR>) is detected, KLDCP will begin processing the command. If a control file is being used, the commands are taken sequentially from the control file.

7. KLDCP compares the base command to each entry in the command dispatch table. The entry that matches the base command provides the starting address of the command routine.

8. If no match for the base command is found, an error message is printed and KLDCP returns to command input mode.

9. KLDCP dispatches to the appropriate command routine.

10. The command routine is executed. Unless an error occurs, the command is completed, and control is returned to KLDCP command input mode.

11. If an error is detected (i.e., KLDCP is unable to complete the command), the command is aborted. KLDCP then performs a checksum operation on itself to determine if the error caused any code to be modified. If so, a checksum error message will be printed before KLDCP returns to command input mode.

12. If the command routine uses service routines, they are called via EMTs and executed. Unless an error occurs, the service routine returns control to the command routine when it is finished. If an error does occur during the execution of a service routine, then the command is aborted, an error message is printed, a checksum is performed, and control is returned to KLDCP command input mode.

11-BASED 10 UTILITY PROGRAMS

In addition to KLDCP there are three other major utility programs in the 11-Based 10 Maintenance Library. Refer to Figure 4.

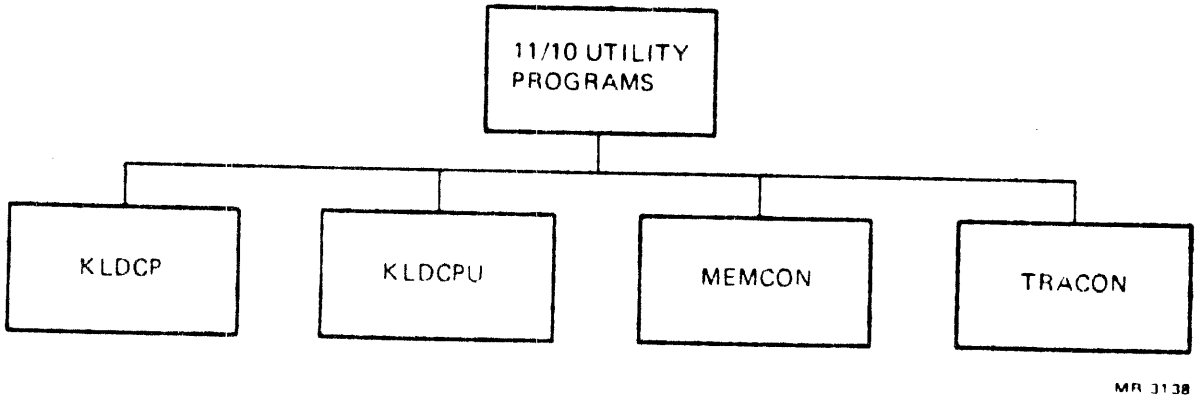


Figure 4. 11/10 Utility Programs

- KLDCPU - A utility program designed to simplify duplicating and maintaining the 11/10 storage media (DECtape, floppy and KLAD-10 packs).

- MEMCON - A utility program designed to simplify configuring KL10 internal and external main memory.

- TRACON - A hardware utility program designed to aid in troubleshooting KL10 processor, memory, and channel faults.

KLDCP loads these programs into the overlay area of 11/40 memory. Only one utility program may be co-resident with KLDCP at any given time.

STOP

Before proceeding further, review the summary modules for KLDCP, MEMCON and TRACON in the 11-Based 10 section of the KL10 Maintenance Guide.

Internal Structure and Control

The internal structure of the utility programs is similar to the internal structure of KLDCP. See Figure 5.

The method these utility programs use to process commands is also similar to the method used by KLDCP to process commands. That is:

1. Once loaded and started, the utility prints its header and prompt.

2. You type a command or submit a control file for execution.

3. The utility program compares the base command to its command dispatch table and dispatches to the appropriate command routine.

4. The command routine executes the command using service routines, if necessary, and returns to the utility program's command input mode when finished.

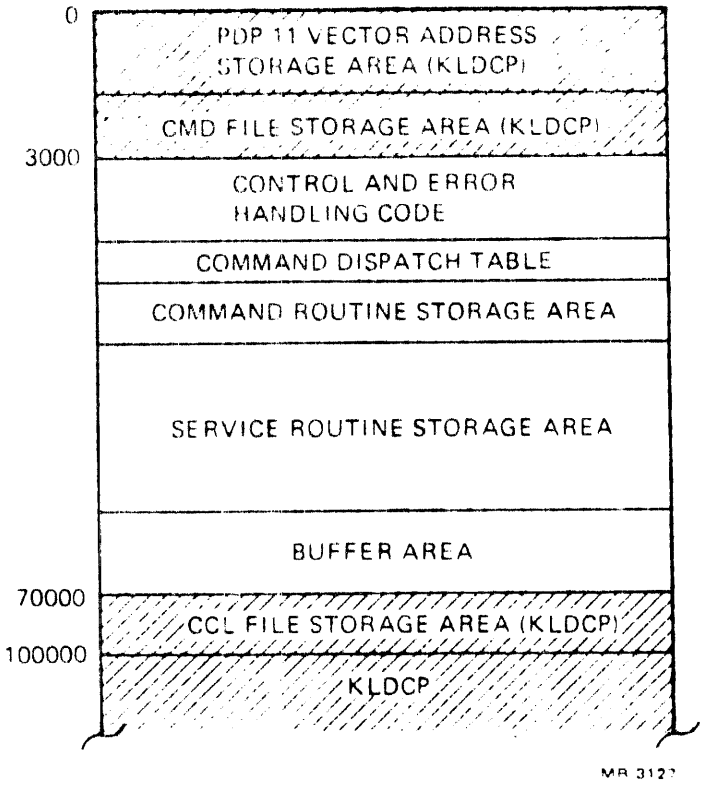


Figure 5. Internal Structure of 11/10 Utility Programs

If a utility program is unable to find the base command in its command dispatch table, control is returned to KLDCP. KLDCP will then attempt to execute the command and return control to the utility. If KLDCP is unable to execute the command, an error message will be printed. Then control will be returned to the utility.

11-BASED 10 DIAGNOSTIC PROGRAMS

The 11-based 10 diagnostics are predominantly hardware (FRU) tests. They load into the overlay area and are designed to detect faults in the DTE, EBox, cache, MBox, main memory and channels. They are also designed to isolate the cause of the fault to the smallest field replaceable unit (i.e., module, cable or backplane). In order to achieve this level of fault isolation the diagnostics frequently use TIC files and isolation routines.

STOP

Before proceeding further, review several of the diagnostic summary modules contained in the 11-Based 10 section of the KL10 Maintenance Guide.

TIC Files

TIC files (sometimes referred to as clock scan routines) are used by some 11-based 10 diagnostics to isolate a fault to the failing clock tick. TIC files are generated during the development of the diagnostic by single-stepping the execute phase of each test and subtest in the diagnostic. (Recall that tests and subtests have three parts: the init, the execute and the check.) The machine state change at each clock tick is read (via diagnostic read functions) and saved in a TIC file. Later, when the diagnostic is being used to isolate a fault, the appropriate TIC file is read into the CCL file storage area of memory. Then the execute phase of the test that detected the fault is single-stepped, and each machine state is compared to the machine state stored in the TIC file until a difference is found. Thus many 11-based 10 diagnostics are able to isolate a fault to a specific clock tick.

Isolation Routines

Isolation routines are also loaded into the CCL file storage area of memory, and they are used to isolate the cause of a hardware fault. They use the 11/40 processor to analyze the symptoms detected by a failing test in order to isolate the cause to the faulty field replaceable unit (FRU). Typically, an isolation routine will perform some additional testing using key symptoms and a look-up table to determine the most probable failing FRU. A typical isolation routine output is illustrated below.

EXAMPLE: A Typical Isolation Routine Output

```
CABLE  LOC:04-1.   M8519.  LOC:04-7.**    M8537.  LOC:04-20.
M8515. LOC:04-28.  M8538.  LOC:04-33.     M8525.  LOC:04-35.
M8512. LOC:04-43.  M8528.  LOC:04-44.     M8530.  LOC:04-47.

**HIGHEST PROBABILITY OF FAILURE

EBUS BIT 20. IS STUCK AT 1.

ANY ONE OF THE ABOVE LIST OF BOARDS MAY BE AT FAULT.  THIS
FAULT IS BEST ISOLATED BY LOOKING AT THE EBUS LINE AND
PULLING BOARDS ONE AT A TIME UNTIL THE BOARD WHICH IS
HANGING THE BUS IS FOUND.  IF THIS FAILS, THE EBUS WIRING IN
THE BAKPLANE SHOULD BE CHECKED.
HC TO CONTINUE CONVERGENCE OR LOOP ON ERROR.
SED TO RESTART

PRG HALT AT 006222
```

PRG HALT AT 006222

Internal Structure and Control

As mentioned earlier, 11-based 10 diagnostics are predominantly of the hardware (FRU) type. That is, they consist of a series of independent tests designed to isolate a fault to the failing FRU. Their internal structure, however, differs in two ways from the basic hardware (FRU) structure described earlier in the introductory module. First, the initialization section of each test is separate from the execute and check section. And secondly, each diagnostic is assembled with a common DIAgnostic CONtrol code, referred to as DIACON. See Figure 6.

DIACON provides both external and internal diagnostic control. Externally, DIACON (via operator dialogue) provides for test selection and test looping. Internally, DIACON handles:

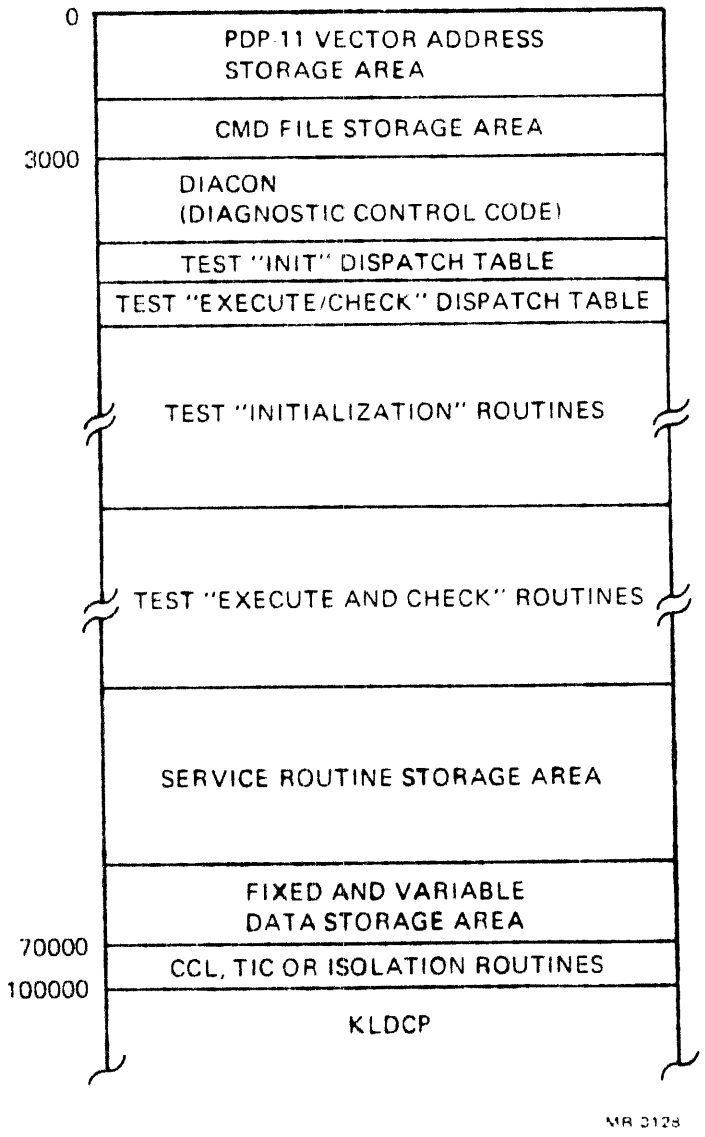


Figure 6. Internal Structure of 11/10 Diagnostics

- Program initialization
- Test dispatching
- Error reporting
- TIC file and isolation routine execution, if they are external to the main diagnostic
- Fault convergence
- Fault confidence calculation

STOP

Before proceeding further, review the DIACON summary module in the 11-Based 10 section of the KL10 Maintenance Guide.

Fault convergence and fault confidence calculation are concerned with intermittent hardware faults. Fault convergence is a method used to establish that the test reporting the fault is also the earliest test in the diagnostic that is able to detect the fault. Fault confidence calculation on the other hand is a method of establishing that the symptoms (which are intermittent) are the best indication of the fault. That is, fault confidence is the percentage of probability that the symptoms reported by the failing test are the same as they would have been, had the fault been solid. For further clarification refer to Case Studies 5 and 7, which follow Figure 7.

Figure 7 illustrates the major control functions performed by DIACON. The case studies following Figure 7 describe how DIACON handles various solid and intermittent fault conditions.

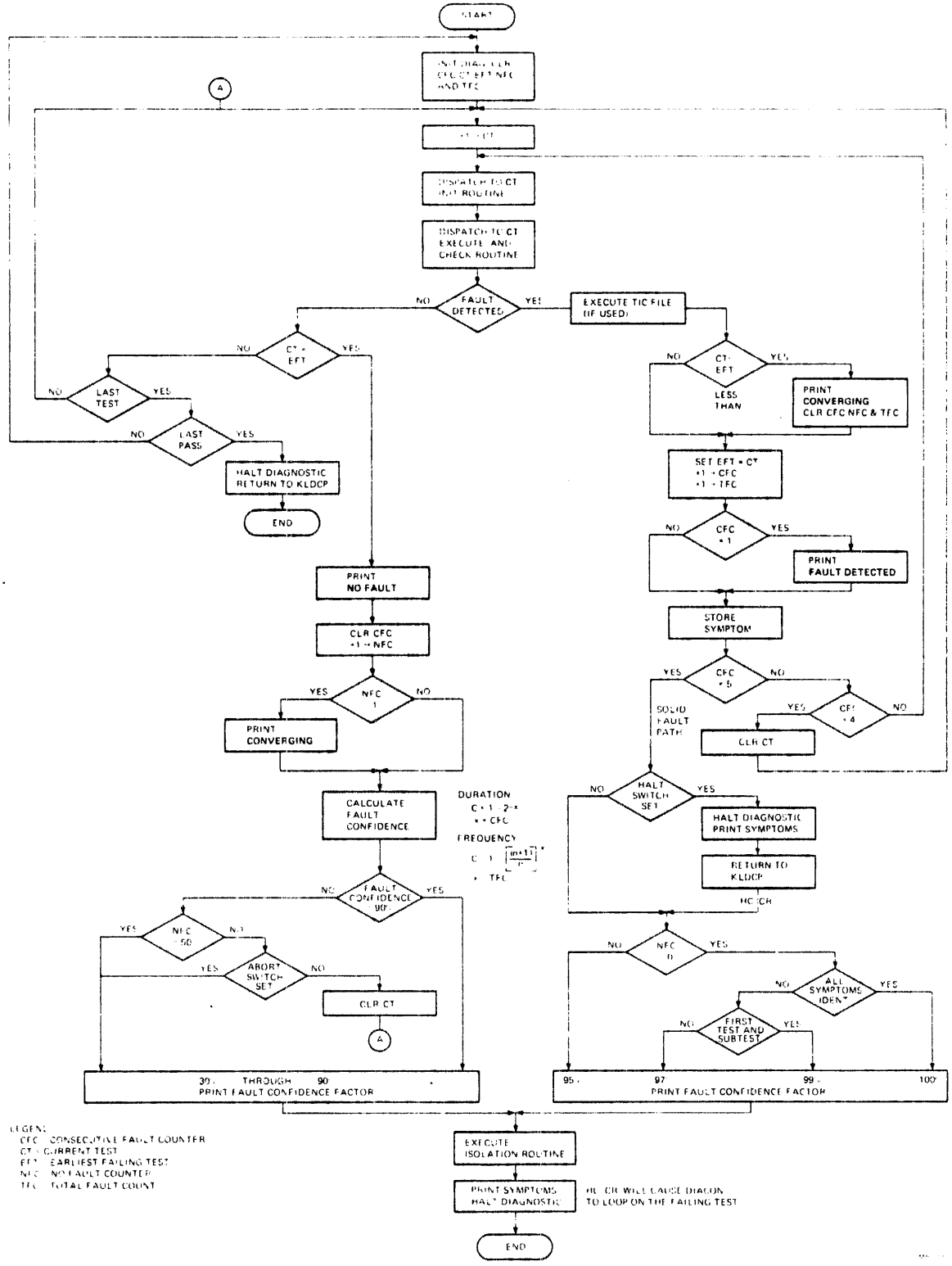


Figure 7. DIACON Control Flow

Case 1: No Faults - Assume a diagnostic runs error-free. No faults are detected.

1. DIACON initializes the diagnostic, clearing the major software registers and counters.

2. The current test (CT) is incremented by one. DIACON dispatches to the initialization routine for Test 1. Then DIACON dispatches to the execute and check routine for Test 1. No faults are detected.

3. The current test (CT) does not equal the earliest failing test (EFT) because EFT was initialized to zero and has not changed.

4. Is this the last test? No. Increment CT and repeat the processes.

5. Is this the last test? Yes. Last pass? No. Clear the major software registers and counters (CT is reset to 0). Rerun the entire diagnostic.

6. Is this the last test? Yes. Last pass? Yes. Halt the diagnostic and return control to KLDCP.

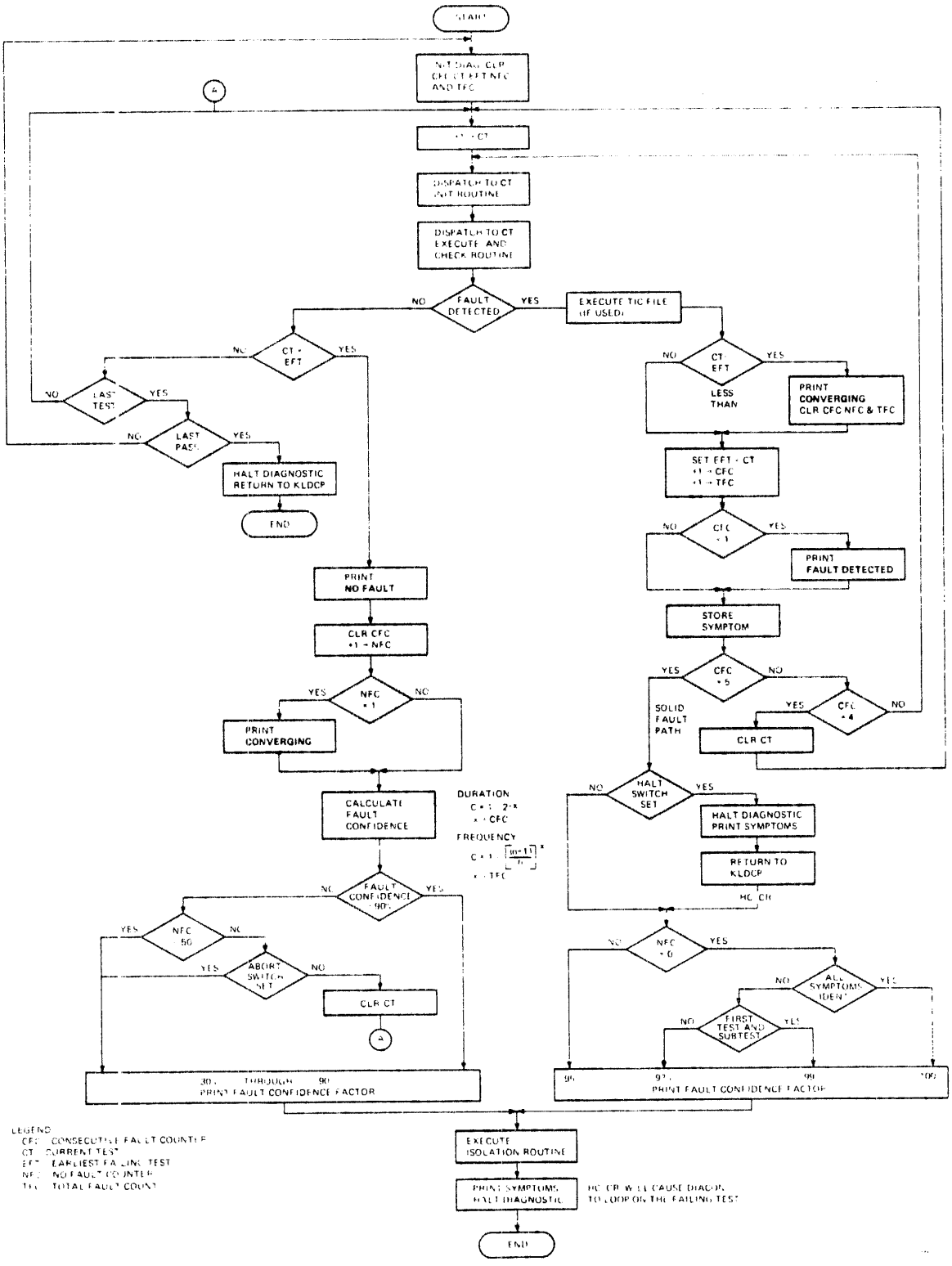


Figure 7. DIACON Control Flow

Case 2: 100% Confidence - Assume that a solid fault is detected by Test 8.

1. The diagnostic runs as described in Case 1 until Test 8 detects a fault. If a TIC file is used, it will be executed at this time.

2. The current test (CT) is not less than the earliest failing test (EFT) because CT = 8 and EFT was initialized to zero and has not changed.

4. EFT is now set equal to CT (both = 8), and the consecutive fault counter (CFC) and the total fault counter (TFC) are both incremented. CFT now equals one. FAULT DETECTED is printed, and the fault symptoms are stored.

5. Test 8 is executed three more times. Then CT is reset to zero and Tests 1 through 8 are run one more time. (CFC should now equal 5, one count for each time Test 8 detected the fault.)

6. Assume the HALT ON ERROR program control switch is set. The diagnostic is halted, and the fault symptoms are printed. Then control is returned to KLDCP. The KLDCP Halt Continue command (HC<CR>) will continue the diagnostic; fault confidence will be calculated and fault isolation performed. A second HC<CR> at this time will cause DIACON to loop on the failing test (TEST 8).

7. Assume that the HALT ON ERROR switch is reset, that the no fault counter (NFC) equals zero (from program initialization) and that all five consecutive fault symptoms associated with Test 8 are identical. The probability that Test 8 is the earliest test able to detect the fault, that the symptoms are indicative of a solid fault, and that the output of the isolation routine will be accurate is calculated to be 100%.

8. The fault confidence factor is printed and the isolation routine run. Then the results are printed, the diagnostic is halted, and control returned to KLDCP. The KLDCP HC<CR> command may be used at this time to cause DIACON to loop on the failing test (Test 8).

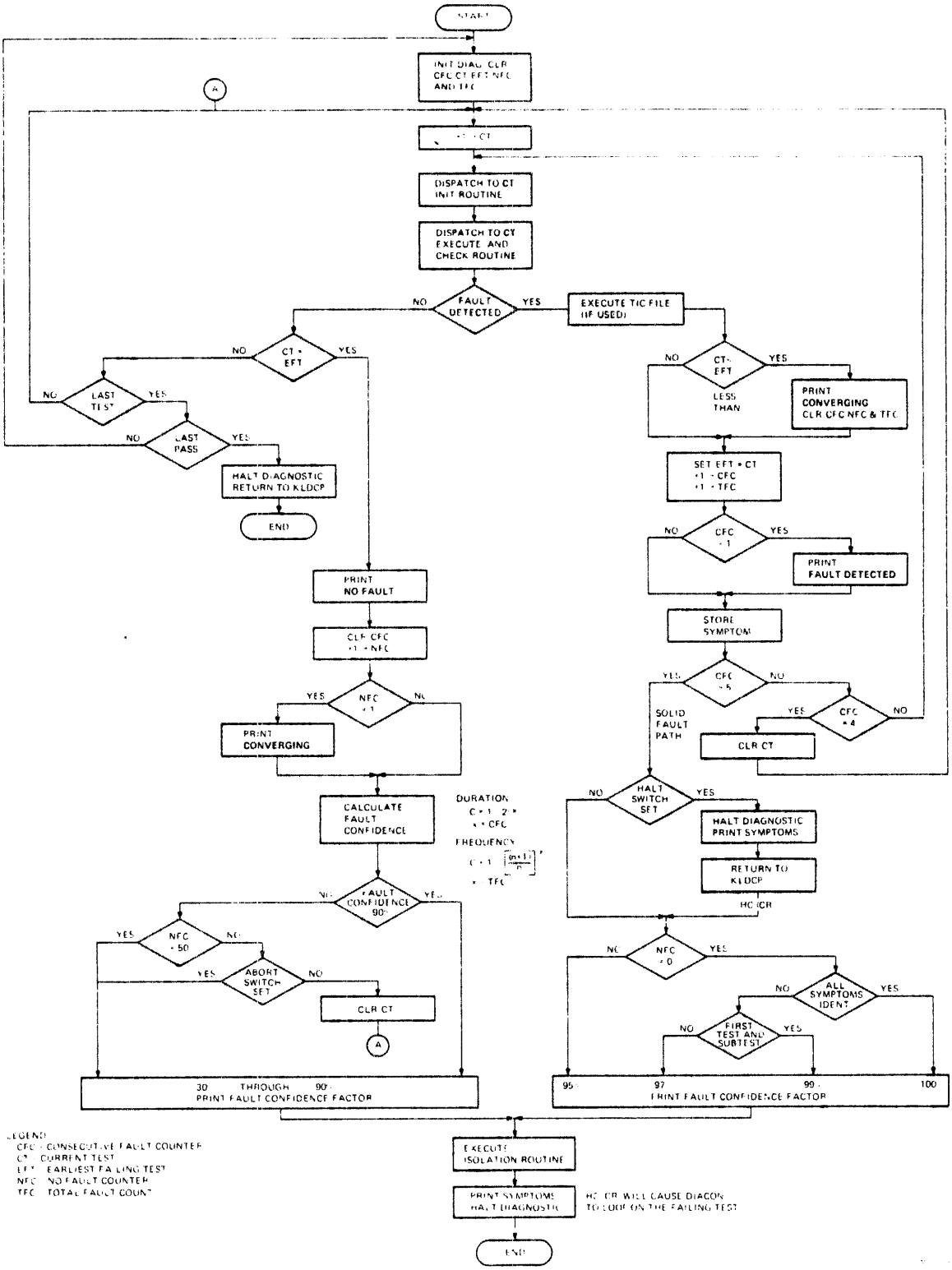


Figure 7. DIACON Control Flow

Case 3: 99% Confidence - Assume that the first test and subtest in the diagnostic detected five consecutive faults, but that in all five cases the fault symptoms were not identical. The probability that this (Test 1, Subtest 1) is the earliest test able to detect the fault, that symptoms are indicative of a solid fault, and that the output of the isolation routine will be accurate, is calculated to be 99%. A 1% probability of inaccuracy is introduced because all five symptoms were not identical.

Case 4: 97% Confidence - Assume that Test 8 detected five consecutive faults but that in all five cases the fault symptoms were not identical. The probability that this testis the earliest test able to detect the fault, that the symptoms are indicative of a solid fault, and that the output from the isolation routine will be accurate, is calculated to be 97%. A 3% probability of inaccuracy is introduced because Test 8 is not the first test in the diagnostic and because all five fault symptoms were not identical.

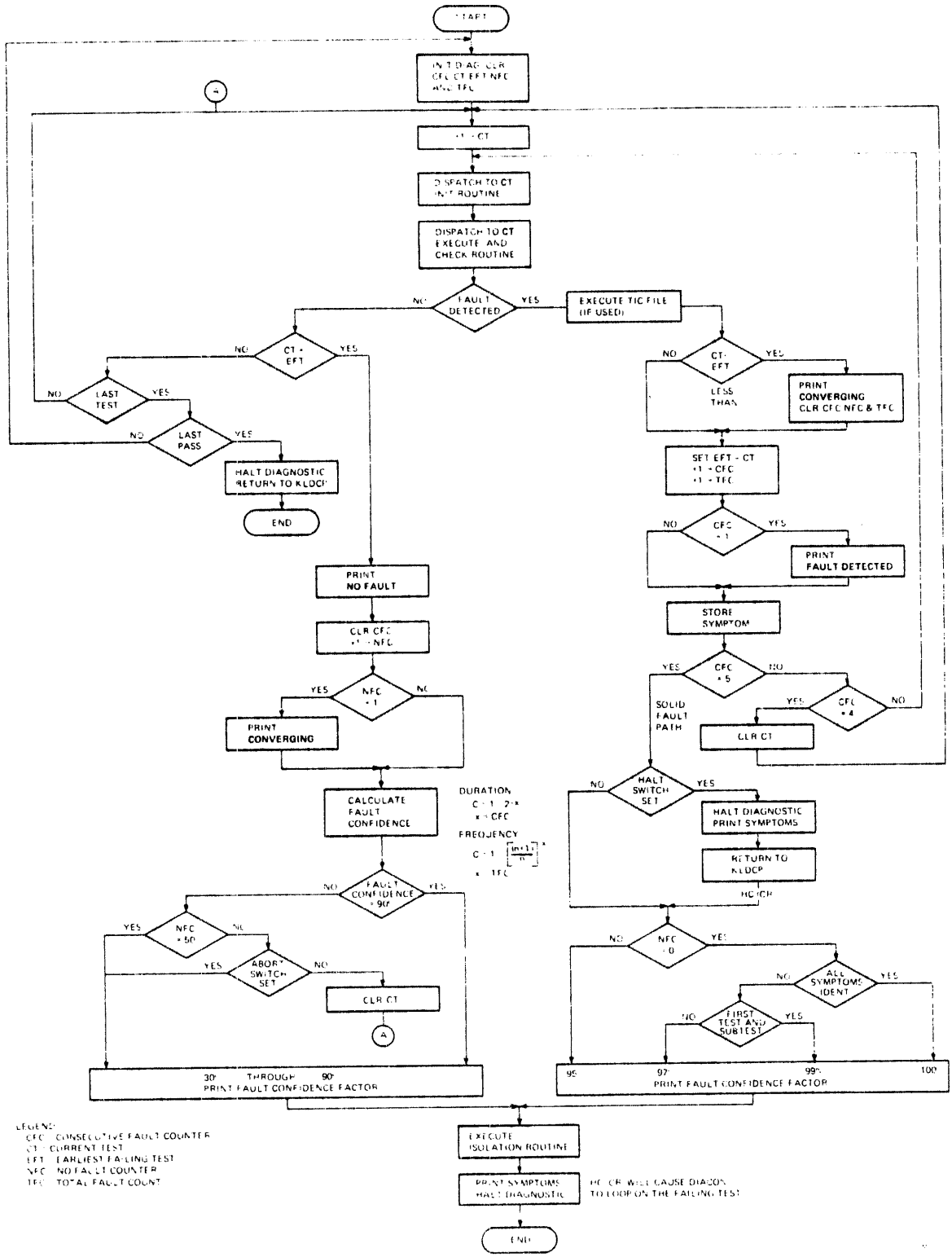


Figure 7. DIACON Control Flow

Case 5: Converging - An Intermittent Failure - Assume that Test 8 is the first test in the series to detect a fault. However, when it was being run for the fifth time, an earlier test, Test 4, detected the fault. This has to be an intermittent fault because Test 4 did not initially detect the fault. In this case, the following occurs:

1. The diagnostic runs as described in Case 2 until the consecutive fault counter (CFC) equals four. Then the current test (CT) is cleared, and the diagnostic is restarted at Test 1.

2. Tests 1 through 3 run, but Test 4 detects a fault. At this point the current test (CT) is less than the earliest failing test (EFT) (CT = 4 and EFT = 8). Therefore, CONVERGING is printed and the consecutive fault counter (CFC) is cleared (Formerly it contained a count of four from Test 8). In addition, the total fault count (TFC) is cleared because this is a new failing test. The no fault counter (NFC) is also cleared, although it was already clear as a result of program initialization.

3. The earliest failing test is then set equal to the current test (4), and the consecutive fault counter is incremented to one. FAULT DETECTED is printed and the new symptoms for Test 4 are stored.

4. Next Test 4 is run three more times. Then once again tests 1 through 4 are run. Assuming that test 4 fails solid and that no earlier test detects a failure, DIACON proceeds as described in Case 2, steps 6, 7, and 8.

5. The only apparent difference between the results produced in this case and the results produced in Case 2 is that the message CONVERGING is printed. This message alerts you to the fact that the fault which now appears to be solid was at one time intermittent.

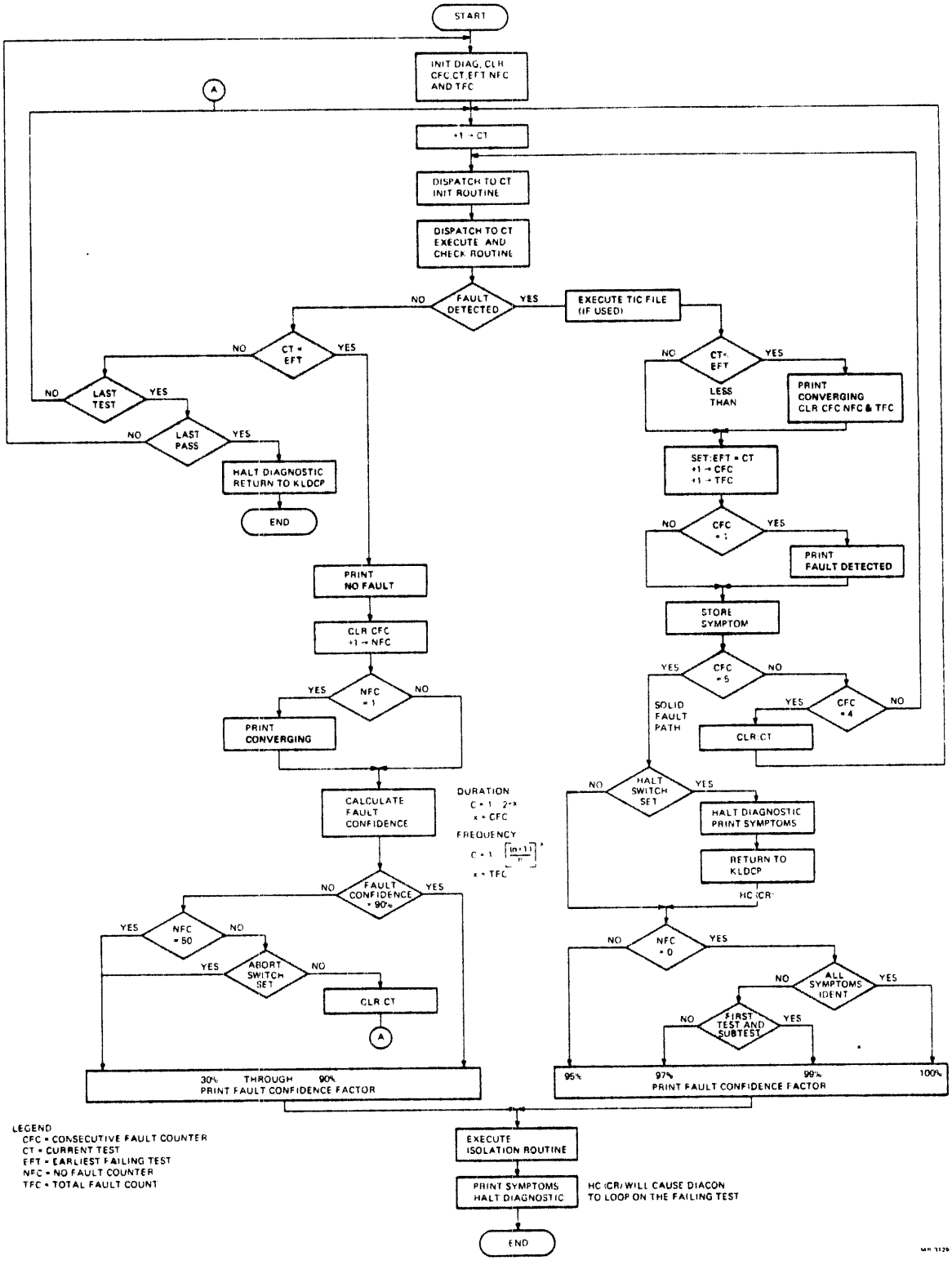


Figure 7. DIACON Control Flow

Case 6: 95% Confidence - An Intermittent Fault - Suppose Test 8 detected three consecutive faults but on the fourth try the fault went undetected. Then at the FAULT DETECTED decision block the path to the left would be taken.

1. At the decision block, current test (CT) equals the earliest failing test (EFT), the answer is Yes (both CT and EFT equal 8). Therefore, NO FAULT is printed, the consecutive fault counter (CFC) is cleared, and the no fault counter (NFC) incremented to one. Finally CONVERGING is printed and fault confidence is calculated.

   Note that the NFC is incremented each time the earliest failing test is unable to detect the original fault.

2. Assume that a 90% fault confidence has not been established. In addition, the no fault counter (NFC) has not reached a count of 50 and the ABORT TEST program control switch is not set. Then the current test (CT) is cleared and Tests 1 through 8 are repeated.

3. Now assume that Test 8 detects five consecutive faults. Then the answer to the decision block "no fault count (NFC) equals zero" (located in the lower right section of the flow) is No. In this case NFC equals 1.

4. Now the probability that this (Test 8) is the earliest test able to detect the fault, that the symptoms are indicative of a solid fault, and that the output of the isolation routine will be accurate is calculated to be 95%. A 5% probability of inaccuracy exists because Test 8 is not the first test in the diagnostic and because at least once Test 8 was unable to detect the fault.

5. With the fault confidence calculated, the fault isolation routine is run and the results printed. Then the diagnostic is halted, and control is returned to KLDCP. The KLDCP HC<CR> command may be used at this time to cause DIACON to loop on the failing test (Test 8).

Case 7: 30% Through 90% Confidence - An Intermittent Fault. DIACON will attempt to establish a 90% fault confidence factor for intermittent failures unless interrupted by:

1. The no fault counter (NFC) reaching a count of 50 indicating that the fault has gone away.

2. The operator setting the ABORT TEST program control switch.

In both cases DIACON will print the fault confidence factor calculated up to that point and cease further calculation. In the second case, however, the operator may type HC<CR> to continue calculation.

DIACON uses two methods to calculate the percentage of fault confidence. The first method is based upon fault duration. The following formula is used:

$$C = 1-2^{-x}$$

where x is the value of the consecutive fault counter (CFC).

Applying this formula, DIACON will calculate the following:

CFC	Percentage of Fault Confidence
2	50%
3	75%
4	88%
5	99%+

The second method used by DIACON to calculate fault confidence is based upon the frequency of the fault. The following formula is used:

$$C = 1 - \left[\frac{(n-1)}{n}\right]^{x}$$

where x is the value of the total fault count (TFC) (i.e., the total number of nonconsecutive faults), and n is the number of the earliest failing test (EFT).

Applying this formula, suppose Test 3 detected 2 nonconsecutive failures before fault confidence was interrupted. Then:

$$56\% = 1 - \left[\frac{2}{3}\right]^{2}$$

Had Test 3 detected four nonconsecutive failures, then:

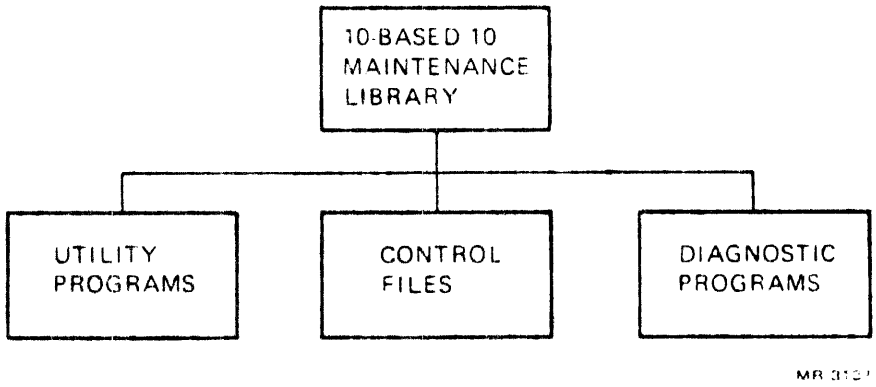
$$80\% = 1 - \left[\frac{2}{3}\right]^{4}$$

It should be clear that when DIACON prints CONVERGING, it is doing very useful work. That is, it is attempting to converge on an intermittent fault. Depending on both the length and complexity of a particular diagnostic and the nature of the fault, fault convergence may range from several minutes to nearly a half-hour.

STOP

Before taking the following module test, refer to the 11-Based 10 section of the KL10 Maintenance Guide. Review the 11/10 STD module and the summary modules for the 11/10 utility and diagnostic programs. Doing so will expand your current understanding and provide some practical insight into the overall organization and structure of the 11-Based 10 Maintenance Library.

The 10-Based 10 Maintenance Library



GENERAL DESCRIPTION

The 10-Based 10 Maintenance Library is written in PDP-10 machine language and is designed to detect and isolate faults in the PDP-10 processor, storage and I/O subsystems. The majority of these maintenance programs will run in either user mode (for on-line maintenance) or executive mode (for stand-alone maintenance). Both modes of operation will be discussed later.

10-BASED 10 UTILITY PROGRAMS

The 10-Based 10 Maintenance Library supports seven major utility programs. Their purpose is to simplify both using and maintaining the maintenance library. Refer to Figure 1.

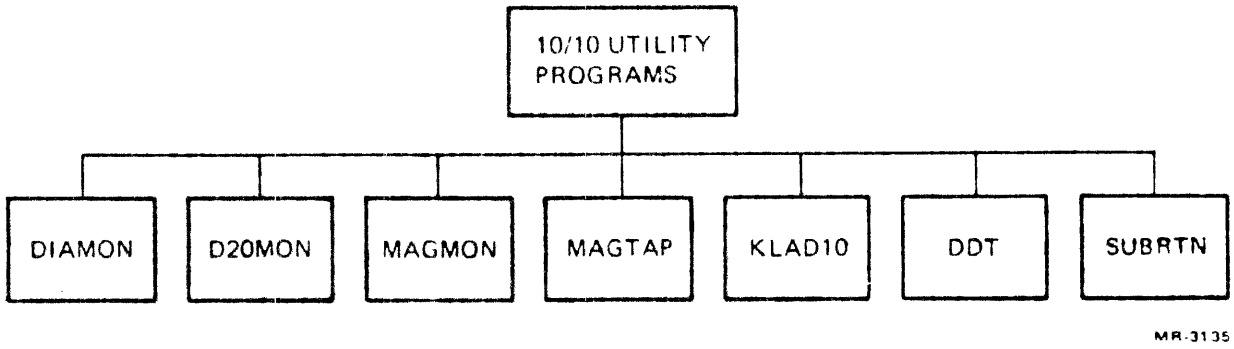


Figure 1. Major 10/10 Utility Programs

- DIAMON - DIAMON is the basic diagnostic monitor. It can handle three types of input media: paper tape, DECtape, and disks having a TOPS-10 file structure. The monitor allows you to load and run a single diagnostic program (via direct commands) or to load and run a series or hierarchy of diagnostic programs (via a control file).

- D20MON - D20MON is a variation of DIAMON. D20MON supports the same features as DIAMON; however, it is designed to handle disks having a TOPS-20 file structure.

- MAGMON - MAGMON is another variation of DIAMON. MAGMON also supports the same features as DIAMON; however, it is designed to handle magtape as an input medium.

- MAGTAP - MAGTAP is a utility program designed to generate diagnostic magtapes for use with MAGMON. MAGTAP also supports the command necessary to transfer files from magtape to disk.

- KLAD10 - KLAD10 is a program used to construct the 6,11 maintenance area on KLAD-10 disk packs. KLAD10 reads the files in the 6,10 maintenance area of the KLAD-10 pack, converts them to 16-bit PDP-11 format, and writes them into the 6,11 area.

- KLDDT - DDT is a modified version of the system software Dynamic Debugging Technique (DDT) program. The 10/10 version of DDT is a very powerful maintenance tool. It enables you to modify and dynamically control the operation of 10-based 10 programs while they are running.

- SUBRTN - SUBRTN is a collection of commonly used service routines. Technically, SUBRTN is not a utility program. It is a package of subroutines that are automatically loaded in conjunction with the 10-Based 10 Maintenance Library. Note that the use of SUBRTN is transparent to you as the user. You have no control of its use or operation.

SUBKL
SUBUSR

STOP

Before proceeding further, review the DIAMON, D20MON, MAGMON, MAGTAP, KLAD10, DDT and SUBRTN summary modules in the 10-Based 10 sections of the KL10 Maintenance Guide.

Normally, utility programs are also required to maintain the control files, diagnostic programs, and maintenance storage media. The 10-based 10 library, however, uses system software programs for this purpose in the following manner:

- Storage media (DECtape, floppies, disk, etc.) are maintained by system programs such as COPY and the Peripheral Interchange Program (PIP).
- Program maintenance is provided for by the system editor program. The editor also may be used to generate or modify maintenance control files.

System software can be used for storage media and program maintenance because both the 10-Based 10 Maintenance Library and the system software use the same file structure and program format.

LOADING AND STARTING METHODS

Systems prior to the KL10 use a conventional hardware read-in switch and a set of I/O device select switches to initially load the 10/10 Maintenance Library. The read-in switch causes the boot program on the selected I/O device to load into memory and start. Once started, the boot program can be directed to load and start either a single diagnostic program or one of the diagnostic monitors.

KL10-based systems, however, use the KLDCP BT command to boot the 10/10 Maintenance Library. KLDCP reads the necessary files from the PDP-11 load device in blocks containing 512 eight bit bytes. The blocks are buffered in eleven memory, then transferred (deposited) through the DTE20 into ten memory. Because the maintenance console (KLDCP) is, in effect, a programmable load device, it can also be directed to load and run (via a control file) a sequence or series of 10/10 programs. The KLDCP B command

is used for this purpose. The B command automatically loads a control file into the CMD overlay area in eleven memory. The control file first directs KLDCP to load and run the 11/10 diagnostic hierarchy. Then, after it has set up ten memory, it directs KLDCP to load and run the 10-based 10 processor and memory functional tests.

Internal Structure and Control

Regardless of the type of processor involved (KA10, KI10 or KL10), the internal structure and control of the 10-Based 10 Maintenance Library is basically the same. This structure is illustrated in Figure 2.

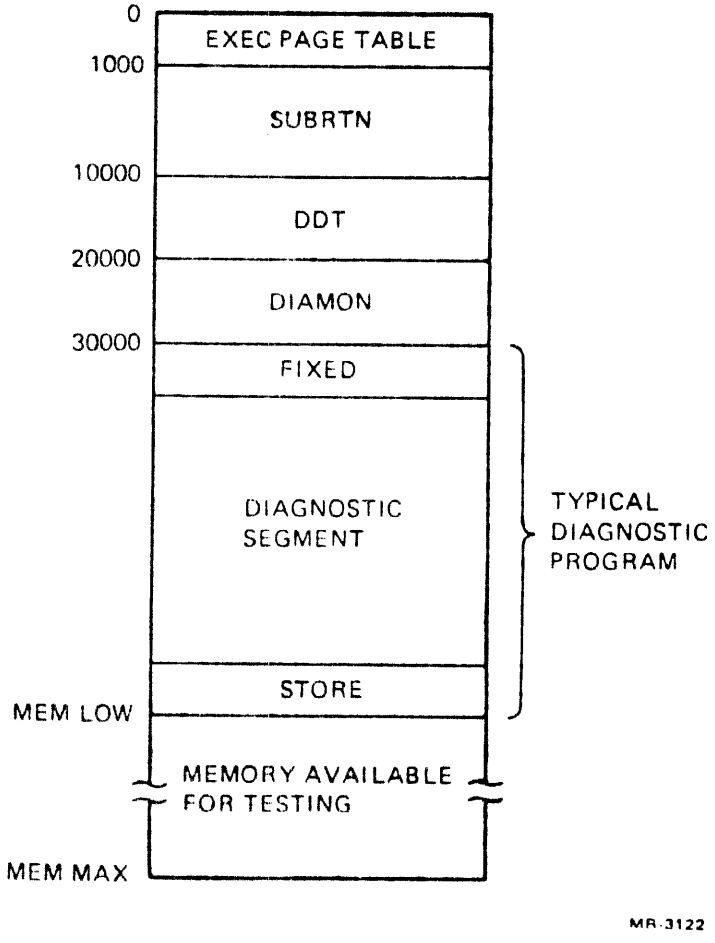


Figure 2. 10-Based 10 Memory Utilization Map

Assuming that a diagnostic monitor (DIAMON) is used, ten memory is structured as follows. First DIAMON is loaded and started. Then DIAMON loads the subroutine package (SUBRTN). If paging is enabled, DIAMON will call a routine in the subroutine package that will construct the executive page table. The page table consists of the necessary trap handlers (i.e., NXM, APR overflow, etc.) and a page map of all available memory.

After the page table is constructed, DIAMON will load DDT and enter command mode. When directed to do so, DIAMON will load and start a 10-based 10 diagnostic. The diagnostic includes the FIXED area, the diagnostic segment, and the STORE area.

The FIXED area contains addresses and other linkages necessary for the diagnostic to communicate with the other program components in ten memory (i.e., DIAMON, DDT, and SUBRTN).

The diagnostic segment and STORE area contain the program initialization and control routine, the test or exerciser routines, any special service routines not available in the subroutine package, and the fixed and variable data storage area.

The 10-based 10 diagnostics use the standard control flow described earlier in the introductory module. That is, the program initialization and control routine initializes the system, reads and stores the state of the program control switches, performs operator dialogue, and dispatches to the test routines. The test or exerciser routines perform the testing using the various service routines as necessary. Faults are reported as they are detected. When testing is complete, control is returned to the diagnostic monitor or calling program.

10-BASED 10 DIAGNOSTICS AND KLDCP

In order to run diagnostics in executive mode on a KL10 based system, KLDCP must be resident in the console front-end subsystem. Refer to Figure 3.

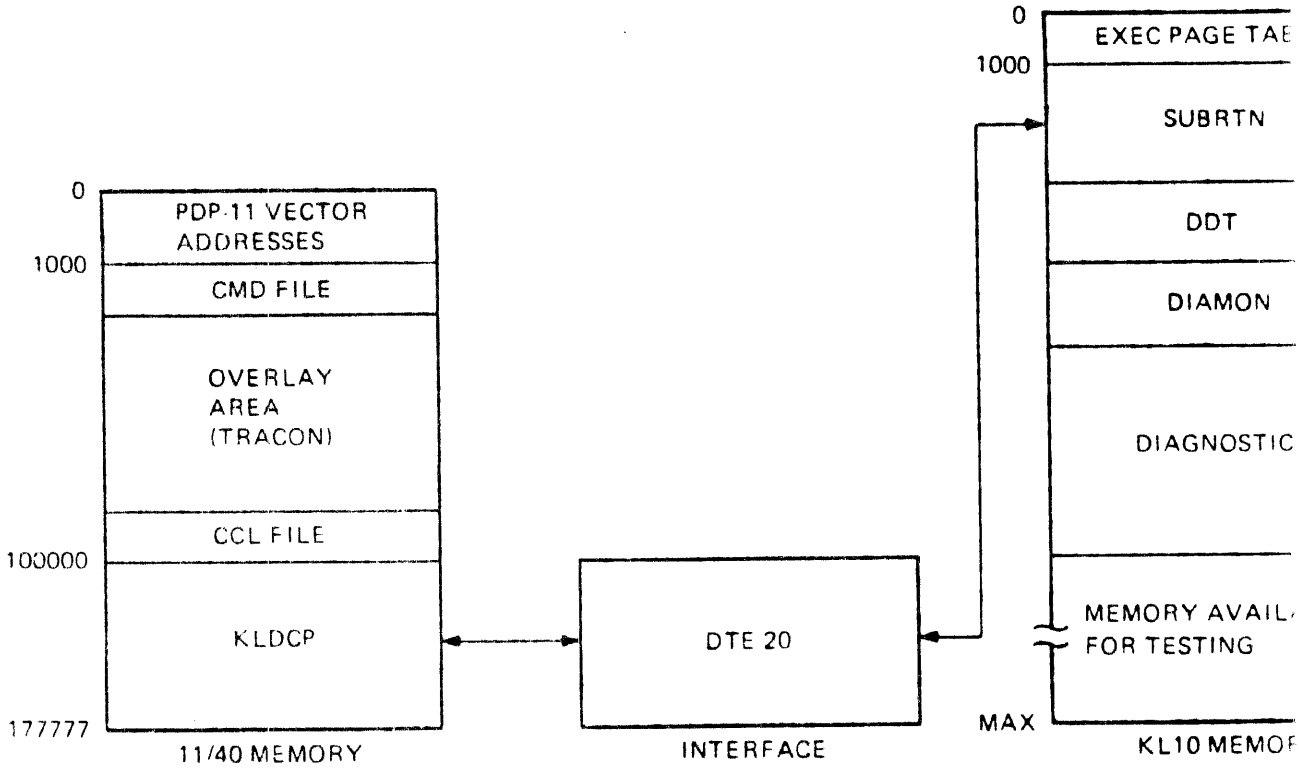


Figure 3. Relationship of KLDCP to the 10/10 Maintenance Library

Once KLDCP has structured ten memory, either the diagnostic monitor (DIAMON) or KLDCP can be used to load and run the 10-based 10 diagnostics. If DIAMON is chosen, the diagnostic will be loaded from the DIAMON load device, usually disk. If KLDCP is chosen, then the KLDCP load devices will be used.

In either case, KLDCP plays an important roll in running 10-based 10 diagnostics. First, KLDCP performs all input and output operations to the console terminal. For example, as you type a command or reply on the terminal, KLDCP builds a buffer in 11 memory. When the command or reply is complete, KLDCP transfers it to the appropriate software component in ten memory. Conversely, when a 10-based 10 program wants to output information to you, it builds a buffer in ten memory and notifies KLDCP. KLDCP handles printing the message on the console or KLINIK terminal.

Secondly, KLDCP allows the software components in ten memory to execute console operations (via calls to KLDCP). Thus the 10-based 10 diagnostics running on a KL10-based system have a significant advantage over those running on KA10 and KI10 based systems that use a traditional hardware console. That is, they do not require operator intervention to perform console operations. This greatly increases their ability to diagnose and isolate hardware malfunctions.

Finally, TRACON, the 11-based 10 hardware utility, may be co-resident with KLDCP. Thus when a 10-based 10 diagnostic detects a fault, TRACON can be used to examine the internal status of the processor, main memory, and channels.

10-BASED 10 MAINTENANCE SOFTWARE AND USER MODE OPERATION

10-based 10 maintenance software uses the same basic internal structure in user mode as it does in executive mode. This is true for both the TOPS-10 and the TOPS-20 operating systems. Refer to Figure 4.

Diagnostics that run in user mode, however, have one major advantage over diagnostics that run only in executive mode; that is, they do not require the system to run stand alone. Thus, system availability to the customer(s) is never seriously affected.

In fact, relative to the operating system (TOPS-10 or TOPS-20), a diagnostic running in user mode is, in most cases, no different than any other job (program) running in user mode. This, is however, both an advantage and a disadvantage to some user mode diagnostics.

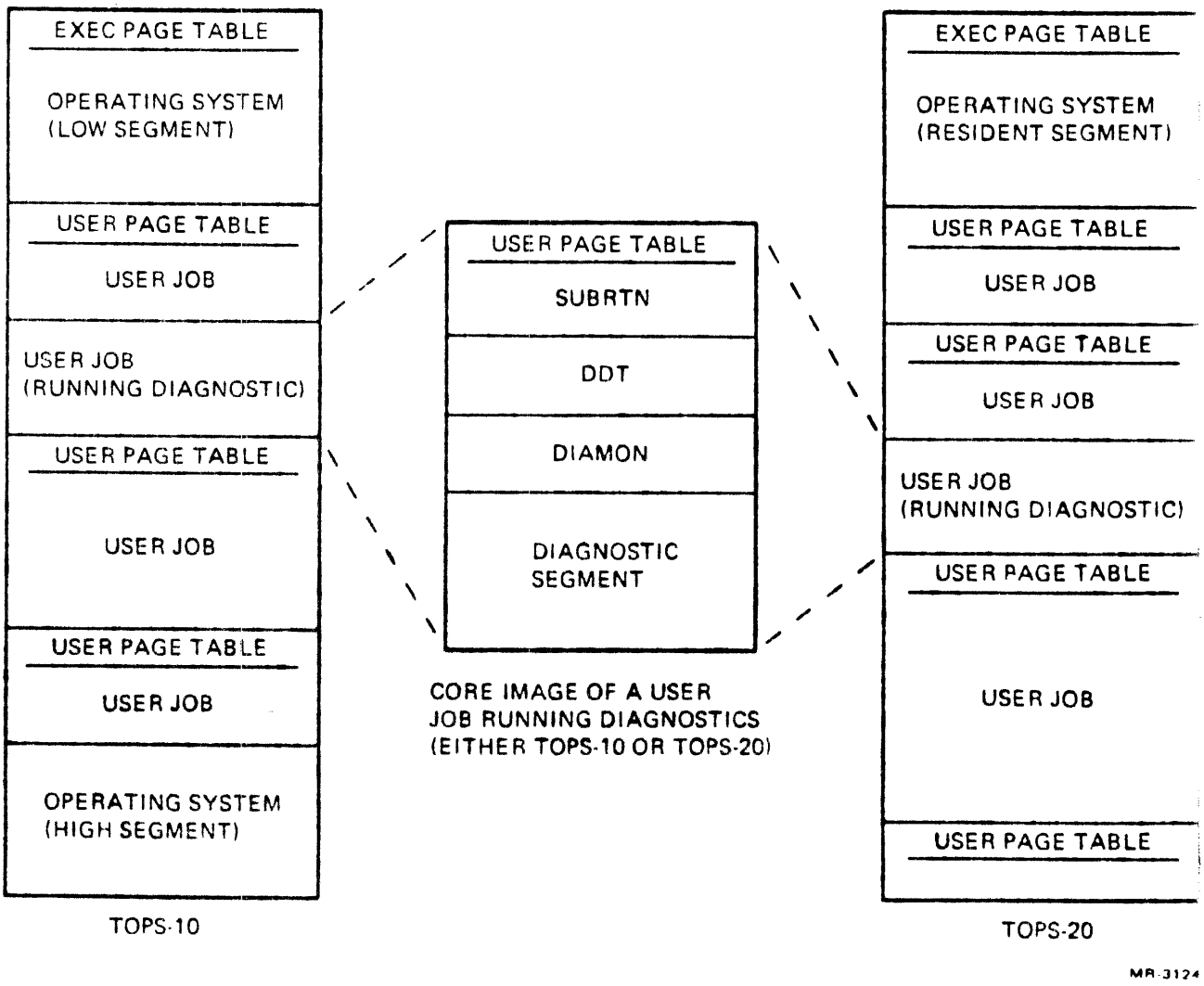


Figure 4. Running 10/10 Diagnostics in User Mode

The advantage is that normally the operating system performs all input and output operations for user mode jobs. This means that if a user mode diagnostic wants to read a file or output a message it need only make the appropriate call to the operating system. Thus, the complexity of writing a user mode diagnostic is reduced somewhat.

The disadvantage, however, is that some user mode diagnostics require direct control over certain I/O operations in order to complete testing. Usually this situation can be avoided by obtaining user mode IOT privileges and then assigning the device to the job. However, this solution is not satisfactory for shared system resources such as disk and tape subsystems since assigning an entire disk or tape subsystem to a single job would, in most cases, seriously affect the performance of the system.

To get around this problem, both the TOPS-10 and the TOPS-20 operating systems support a special diagnostic monitor call called DIAG. DIAG allows the diagnostic to request direct control over a physical channel and either one or all of the devices on that channel. While the DIAG is in effect (usually for less than a second) the operating system will not attempt to access or use the assigned channel. Instead all requests for access will be put in a queue until the diagnostic is finished. As soon as the diagnostic completes its I/O, including reading the status of the device registers, it returns the channel and device to the operating system. The check portion of the test is almost always done after the channel is returned. Thus DIAG makes certain that the user mode diagnostic causes minimal interference of normal system operation.

In summary, then, the major difference between a diagnostic running in executive (stand alone) mode and the same diagnostic running in user mode is that in exec mode the diagnostic must perform all I/O operations by itself but in user mode, the same diagnostic must request privileges in order to perform direct I/O operations. A diagnostic does not request privileges to swap. Swapping is characteristic of any job running in user mode. Its occurance has no effect on the actual execution of the diagnostic.

SYSTEM AND SUBSYSTEM EXERCISERS

Although the 10-Based 10 Maintenance Library does support some subsystem interaction tests, it does not support a standard system exerciser. Instead, the operating system (TOPS-10 or TOPS-20) performs this function. Essentially, each time the operating system detects an error it records the information in a system error file called ERROR.SYS. Later, during preventive or corrective maintenance, a system software program referred to as SYSERR is run. SYSERR reads, formats, and outputs the information in the ERROR.SYS file. The information can then be analyzed, and potential faults can be identified and corrected before the performance of the system is seriously affected.

Using an operating system in this way has two distinct advantages. First, no time is lost running a stand alone system exerciser. And secondly, the error symptoms and conditions reported are real as opposed to simulated. (Remember the objective of a maintenance software system exerciser is to simulate the environment experienced by the system during normal operation).

STOP

Before taking the following module test, review the 10/10 STD module and several of the diagnostic summary modules in the 10-Based 10 section of the KL10 Maintenance Guide. Doing so will increase your current understanding and provide some practical insight into the organization structure, and use of the library.

EMITTER-COUPLED-LOGIC (ECL)

The emitter-coupled-logic (ECL) microcircuit is, by design, nonsaturating, and therefore avoids transistor storage time and its attendant speed limitation, characteristic of transistor-transistor logic (TTL). The high speed of ECL microcircuits has either or both of two characteristics; switching rates of over 50 megahertz and/or gate propagation delays of less than 6 nanoseconds. In general, the gate propagation delays of ECL logic are approximately 2 nanoseconds.

Some of the salient features of ECL microcircuits are as follows:

- High input impedance/low output impedance properties enable large fanout and versatile drive characteristics.
- Minimal power supply noise generation due to differential amplifier design.
- Nearly constant power supply current drain.
- Minimal crosstalk due to low-current switching on signal path.
- Low on-chip power consumption (e.g., less than 8 milliwatts in some complex function chips)
- No line drivers needed due to open emitter outputs of ECL
- Capability of driving twisted pair transmission lines of up to 1000 feet in length.
- Simultaneous complementary outputs available at logic element output without using external inverters.

LOGIC AND POWER LEVELS

	Nom	Min	Max
Supply voltage (V_{EE})	-5.2		
Noise immunity	*	*	*
High output voltage	-0.924	-0.96	-0.81
Low output voltage	-1.75	-1.65	-1.85
High input voltage		-1.105	
Low input voltage		-1.85	-1.475

*Noise immunity of a system involves line impedances, circuit output impedances, propagation delays, and noise margin specifications. Noise margin is a dc parameter calculated from specified points tabulated on an ECL data sheet for the particular microcircuit in question.

CIRCUIT THEORY

A typical ECL gate circuit is shown in figure 1-9 and consists of a differential amplifier input circuit, a temperature and voltage compensated bias network, and emitter-follower buffering transistors for transmission line driving. To explain operation of the gate, each of the major sections is discussed in the following paragraphs.

The differential amplifier is an emitter-coupled current switch consisting of transistors Q1 through Q5. The multiple gate inputs provide an OR function (Q1 through Q4) which is amplified by current switch Q5. To understand the gate's operation, assume that all gate inputs are low (-1.75 V and Q1 through Q4 therefore cutoff). Under this condition, Q5 is forward biased, the base being held at -1.29 volts by the bias supply voltage (V_{BB}), and its emitter is at one diode voltage drop (0.8 V) more negative than its base (-2.09 volts total). The base-to-emitter differential then becomes the difference between the low logic level (-1.75 V) and V_{BB} (-1.29 V), or 0.34 volt. Since this voltage is less than the threshold voltage to turn Q1 through Q4 on, they remain in the cutoff state. The current through Q5 will be about 4 milliamperes with a voltage of -2.09 volts at the emitter nodes of Q1 through Q4 using an emitter resistor R_E of about 780 ohms.

The emitter-follower outputs buffer the current switch from loading and restore output voltages to proper ECL levels. The OR output is obtained through Q8 producing the low-level logic signal of -1.75 volts. Similarly, the NOR output is obtained through Q7 producing the high-level logic signal of -0.924 volt.

When any or all of the logic inputs is switched to the high logic level, the appropriate input transistor turns on, a current flows through resistor RC1 in the differential amplifier, and transistor Q5 cannot sustain conduction due to forward biasing, so is cut off. After translating through the emitter followers, the NOR output is a nominal -1.75 volts and the OR output a nominal -0.924 volt.

The differential action of the switching transistors (one section off when the other is on) produces simultaneous complementary signals at the output.

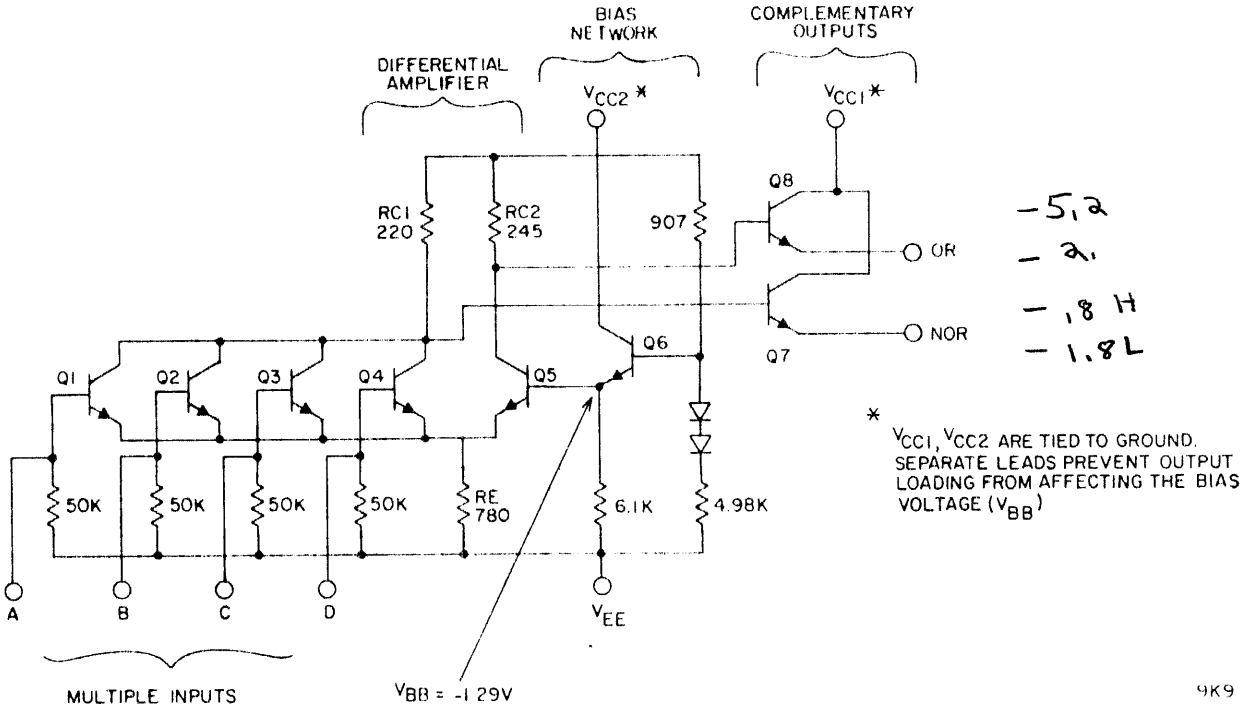


Figure 1-9. Typical ECL Gate

The bias network provides a reference voltage (VBB) of -1.29 volts. This network compensates for variations in power supply voltage and temperature changes to ensure that the bias voltage threshold point remains in the proper operating region.

LOADING CHARACTERISTICS

The differential input to ECL circuits offers several advantages. Its common mode rejection feature offers immunity against power supply noise, and its relatively high input impedance enables any circuit to drive a large number of gate inputs without deterioration of noise margins. The dc loading factor (the number of gate inputs of the same family that can be driven by a circuit output) for ECL circuits is 90. While dc loading causes a change in output voltage levels, thereby tending to affect noise margins, ac loading increases the capacitances associated with the circuit, and therefore affects circuit speed. For ECL circuits, best performance at fanouts of greater than 10 will occur with the use of transmission lines. The propagation delay and rise time of a driving gate are affected very little by capacitance loading along a matched parallel-terminated transmission line. However, the delay and characteristic impedance (Z_O) of the transmission line itself are affected by the distributed capacitance and loading due to stubs off the line. Maximum allowable stub lengths for loading of an ECL transmission line vary with line impedance. For example, a transmission line with a Z_O of 50 ohms will accept stubs 4.5 inches (114.3 mm) long. When the Z_O of the line is changed to 100 ohms, stubs may be only 2.8 inches (71.1 mm) long.

The input loading capacitance of an ECL 10109 is 2.9 picofarads. Therefore, fanouts in a non-transmission line environment should be limited to a maximum of 10 loads due to line delay increases which in turn limit speed.

UNUSED INPUTS

The input impedance of the differential amplifier used in the typical ECL circuit is high when the applied signal level is low. Under low signal conditions, any leakage to the input capacitance of the gate may cause a gradual buildup of voltage on the input lead, thereby adversely affecting switching characteristics at low repetition rates. All but a few of the ECL circuits contain input pull-down resistors between the input transistor bases and the -5.2 volt power supply (V_{EE}). Therefore, unused inputs may be left unconnected because leakage current is dissipated by the resistor, keeping inputs sufficiently negative so that circuits will not trigger due to noise coupled into those inputs.

Input pull-down resistors must not be used as pull-down resistors for preceding open-emitter outputs. If an ECL circuit (such as the 10116) does not contain input pull-down resistors, one input of a circuit must be connected to the reference bias supply voltage (V_{BB}) and the other input to V_{EE}.

WIRED LOGIC

Wired-OR gates can be produced in ECL microcircuits by wiring output emitters together (to a maximum of 10) outside their respective packages. Wired-OR gates can be connected directly to a bus, also. Propagation delay is increased by approximately 50 picoseconds per wired-OR gate. To economize on power dissipation, a single output pull-down resistor is used per wired-OR gate. Normally, wired-OR gates are connected between gates on the same logic board.

ECL PACKAGING

ECL microcircuits are manufactured in a variety of physical configurations. Two of the more common ones used for the ECL microcircuits described in this manual are the ceramic dual in-line and the plastic dual in-line cases having both 16 and 24 pins, depending upon the size of the package. Figure 1-10 illustrates these packages.

ECL/TTL WAVEFORMS AND LOGIC SYMBOLOGY

Student Guide

CONTENTS

CONTENTS (cont.)

PREFACE

As the computer industry continues to grow and expand, so does the role of the field service engineer. Announcements of new technology and new applications appear almost daily. It is a tremendous challenge to the field engineer not only to be aware of these advances but to incorporate them effectively into his troubleshooting techniques.

One such advance is the growing use of emitter coupled logic (ECL) devices. High speed and low propagation delay of ECL lends it readily to mainframe applications. Although Transistor-Transistor Logic (TTL) has been and will continue to be used for many years, especially in peripheral equipment, ECL has begun to take over in CPU applications and in other areas of high speed usage. Along with the speed of ECL, however, come unique problems and waveforms that may be unfamiliar to a TTL technician. The purpose of this book is to acquaint the reader with some of the more common problems encountered in ECL devices and with typical ECL waveforms. This text is not intended to create ECL design engineers nor to provide a comprehensive study of ECL. It is intended to supply basic ECL and TTL references and logic symbology for the comparative circuits of both logic types, in addition to making the field engineer aware of problems common to both kinds.

BASIC TRANSMISSION LINE THEORY

As the preface pointed out, the biggest advantage of ECL circuits is their speed. It is this, however, which also causes some of the common problems of ECL because at higher frequencies, signal runs no longer act as straight wires with lumped characteristics but rather as transmission lines with distributed impedence.

According to Ohm's Law, as a signal travels down a line, a voltage to current ratio is established that is dependent upon the line's impedence. When the ratio reaches the line's end or an area with a different characteristic impedence, a rapid adjustment must be made to maintain the proper ratio. This rapid change in the voltage to current ratio causes a reflected voltage wave to travel back down the line towards the source. The amplitude and sign (whether it will add or subtract from the initial voltage level) of the wave is dependent upon the reflection coefficient which is determined by the ratio of the load resistance minus the line impedence to the load resistance plus the line impedence $\frac{(RT - Z0)}{RT + Z0}$.
It can range from a -1 for a shorted line ($R_T = 0$) to a +1 for an open line (R_T = infinity). As a general rule of thumb, if the load resistance is greater than the line impedence, the amplitude of the reflected wave will be positive and add to the initial voltage. If the load resistance is less than the line impedence, the amplitude of the reflected wave will be negative and subtract from the initial voltage.

If when the reflected wave returns down the line to the source, the source resistance is also unequal to the line impedence, another adjustment must be made in the voltage to current ratio. Thus another wave will be generated but this time travels towards the load. These reflections will continue back and forth between source and load with the amplitude of each wave becoming less than the previous one until they settle at the desired voltage level (about -0.8V for a logical "1" or about -1.75V for a logical "0". Refer to Figure 1-1). These waves are one cause of the phenomenon called ringing.

In general, one of two methods is used to reduce or eliminate these reflections. The first method, called series termination (Refer to Figure 1-2), involves placing a resistor in series with the output of the source gate. The value of this resistor is such that it matches the source resistance to the line impedence. Thus when the first reflection is received back at the source, there is no impedence mismatch; therefore, since no adjustment has to be made in the voltage to current ratio, no further reflections occur.

TYPICAL ECL WAVEFORMS

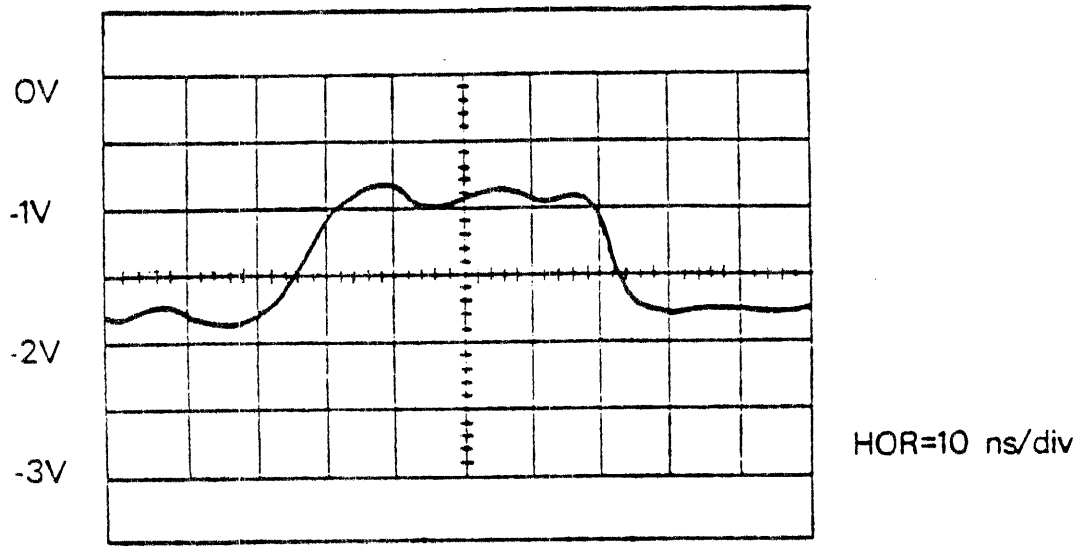
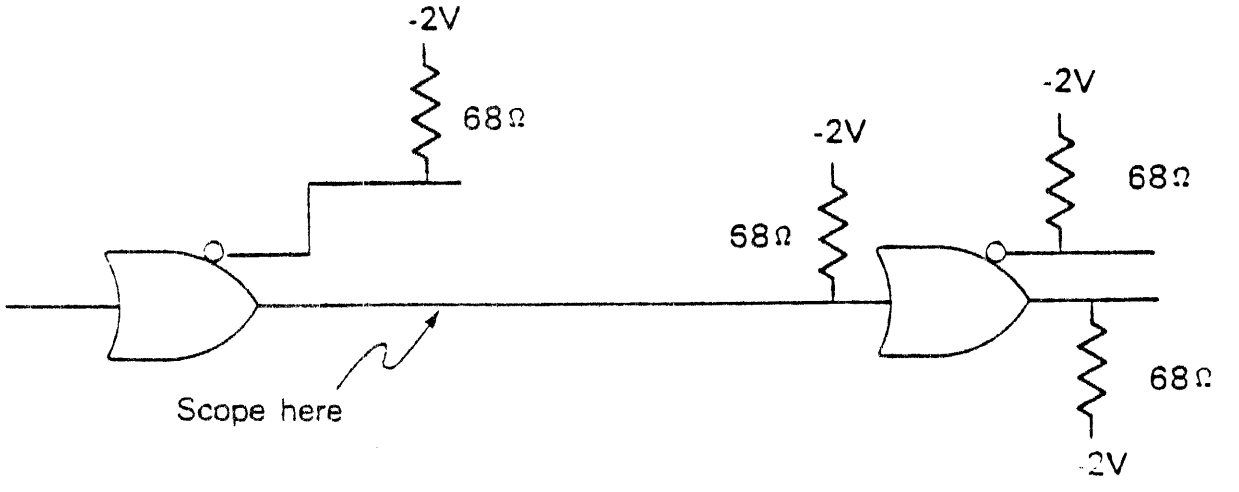


Figure 1-1. A Typical ECL Signal

Parallel Termination : 68Ω to -2V



D8 0033

TERMINATION, SERIES AND PARALLEL

One disadvantage to the series termination method is that the receiving gate must be placed at the end of the line since the signal does not reach full amplitude until the reflection from the load begins to travel towards the source.

The second method is the one most used in DEC circuits. It is called parallel termination (Refer to Figure 1-3). It involves hanging a terminating resistor between the receiving end of the line and having a voltage that is more negative than the lowest logic voltage used. DEC equipment utilizes a separate -2 volt supply. If, however, the terminating resistor is cracked, missing or of the wrong value, unusual waveforms will occur (Refer to the trouble shooting guide for typical waveforms). Mose DEC circuits use a 68 Ω terminating resistor. In certain applications such as delay lines and some bus drivers, however, a different value may be used. Check the schematic of the circuit test to be certain of the correct value.

The major disadvantage of the parallel termination method is that more power is dissipated in various ways. In addition, besides the possible mismatch between line and load, other situations must be considered which can also cause unusual reflections. Several mismatches may occur as the signal travels from a PC board etch to backplane wiring and then to another PC board. DEC uses twisted pair backplane wiring with one wire connected to ground to dampen reflections here and to minimize cross talk. In general, however, the impedence mismatch between PC board and backplane is not sufficient to cause problems except in special situations which can seriously degrade the signal. If, for example, the backplane wire is nicked, sharply bent, or if the fingers of the PC board are filmed with dirt, the mismatch becomes significant, and reflections may occur. Figure 1-4 illustrates the effect of a serious mismatch between the PC board and backplane wiring. It is essential that an ECL technician be aware of these potential problems to avoid replacing costly ICs that are functioning properly but appear to be bad.

For more information on reflections, impedence mismatch, and transmission line theory, refer to the reference section at the end of this book.

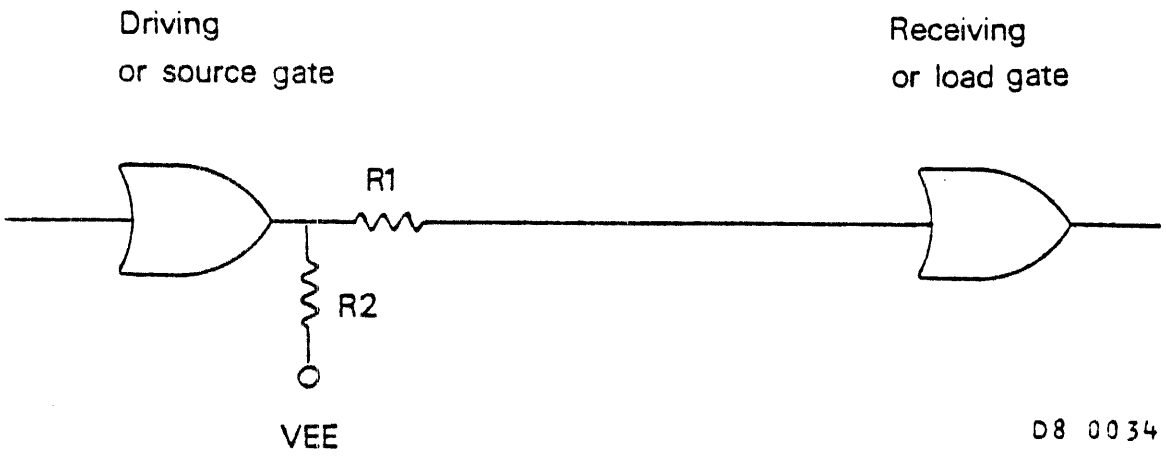


Figure 1—2. Series Termination

The value of R1 is selected such that the source resistance is equal to the line impedence. R2 provides a return to VEE. Recieving gates must be placed at the end of the line.

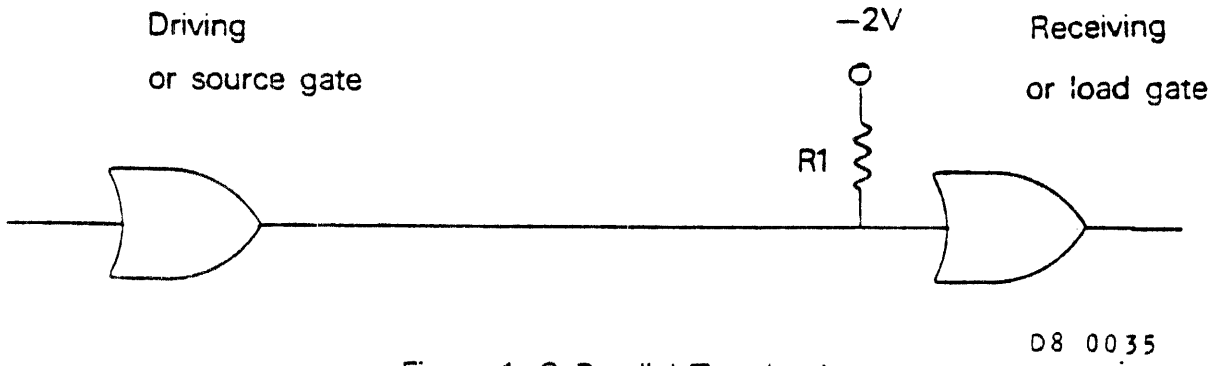


Figure 1—3. Parallel Termination

The value of R1 is matched to the line impedence. Most DEC circuits use a 68 ohm resistor. Loads may be placed anywhere along the line.

IMPEDENCE MISMATCH

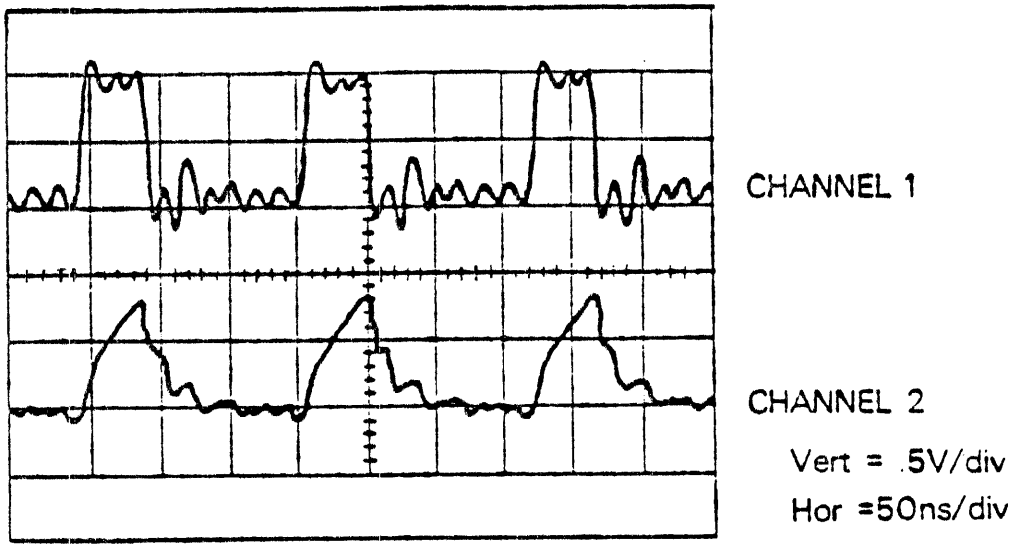
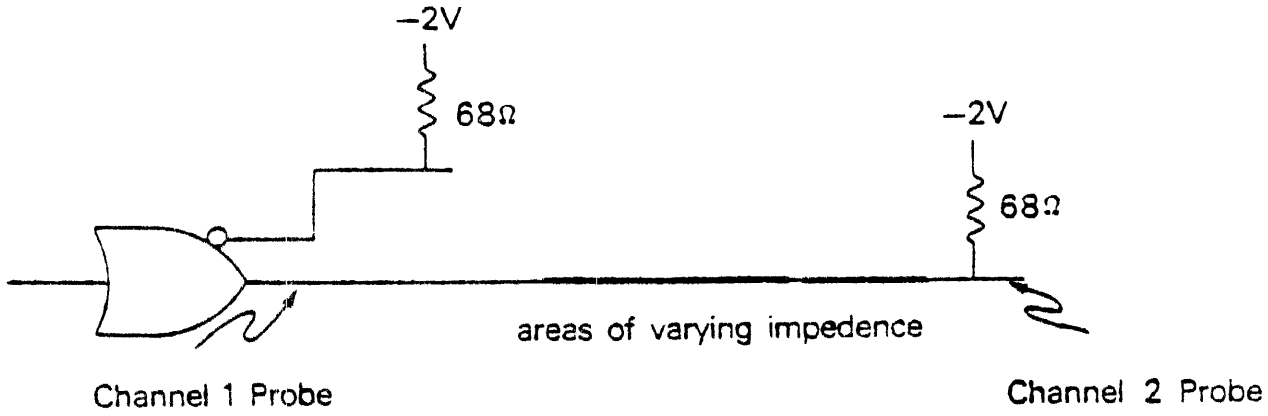


Figure 1-4. Impedence Mismatch

Channel 1 is the output of an ECL gate. Channel 2 shows the same signal after it has travelled through wires of varying impedence. Although most technicans would never encounter as dramatic a mismatch as is illustrated here, dirty fingers, nocked etches or backplane wiring, or kinked backplane wires can all contribute to impedence mismatch.

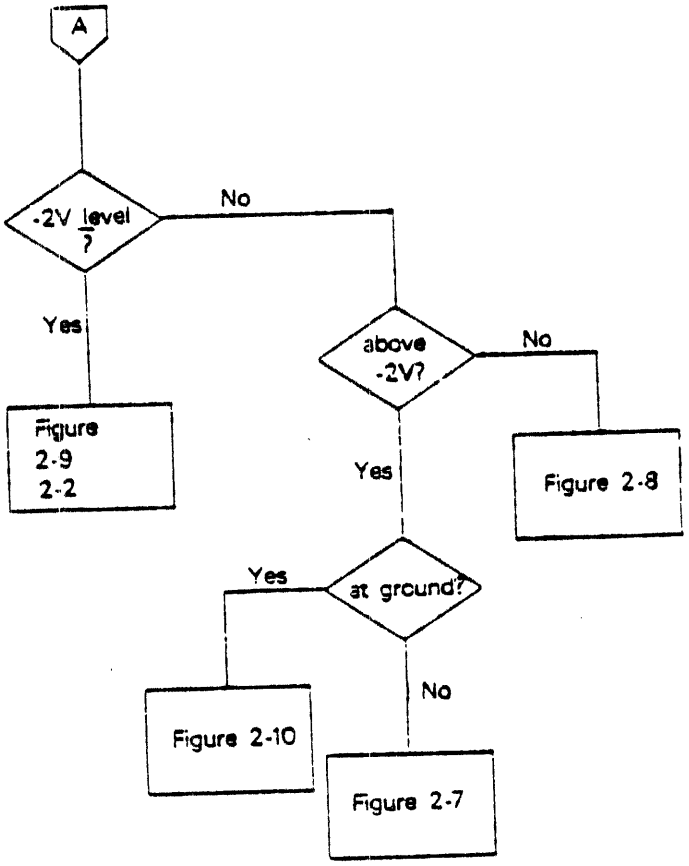


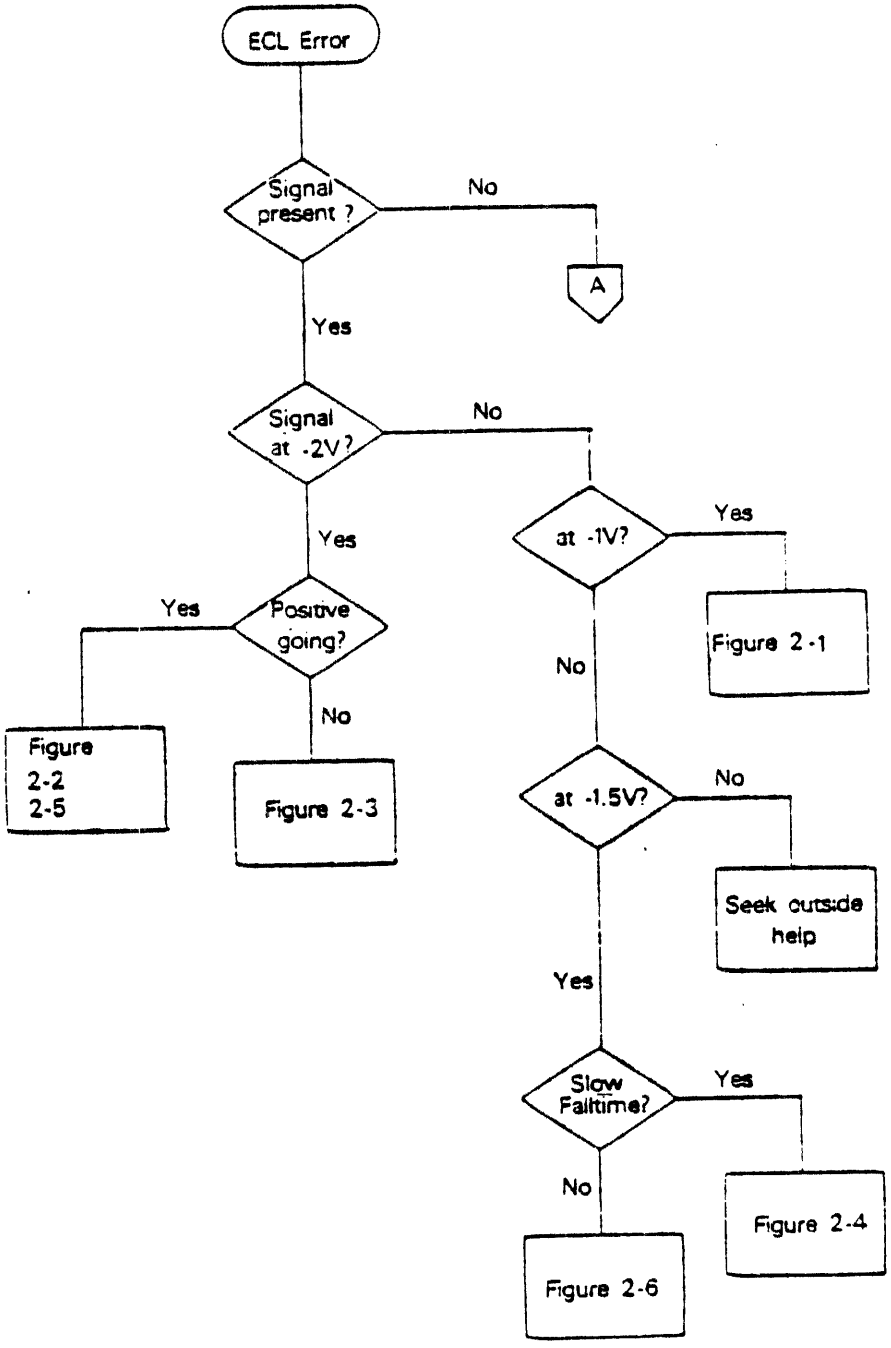
D8 0036

SECTION 2

ECL TROUBLESHOOTING GUIDE

FAULT ISOLATION FLOW




ECL Error
Signal present ?
No
A
Yes
Signal at -2V?
No
Yes
at -1V?
Yes
Figure 2-1
Yes
Positive going?
No
Figure 2-2 2-5
Figure 2-3
No
at -1.5V?
No
Seek outside help
Yes
Slow Falltime?
Yes
Figure 2-4
No
Figure 2-6
DS 0052

UNTERMINATED WAVEFORM

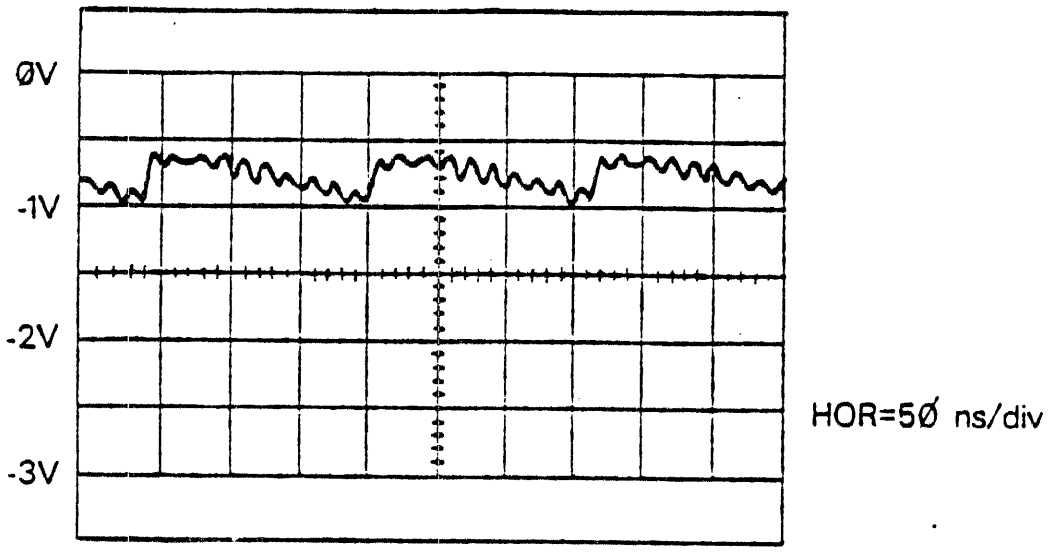
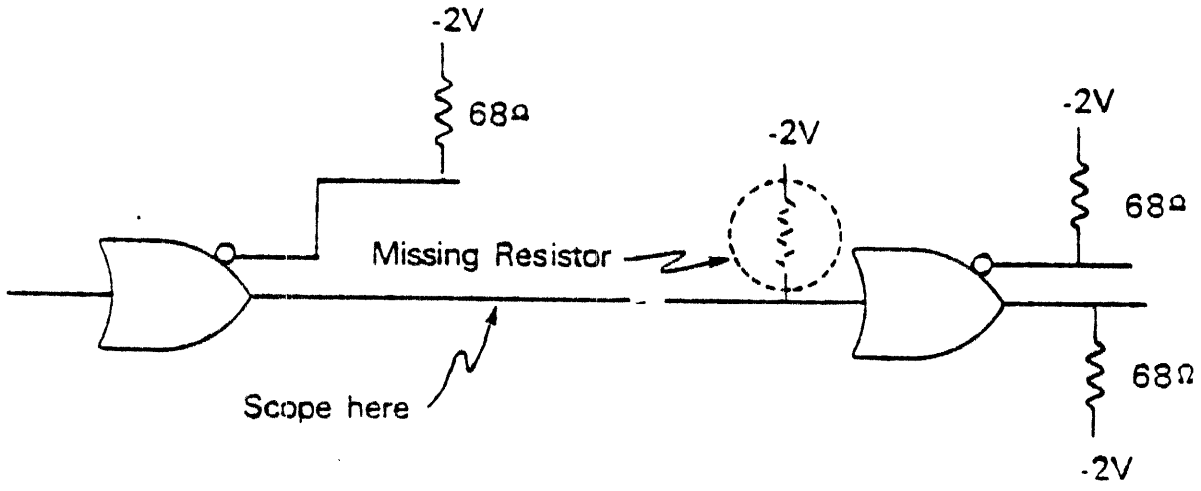
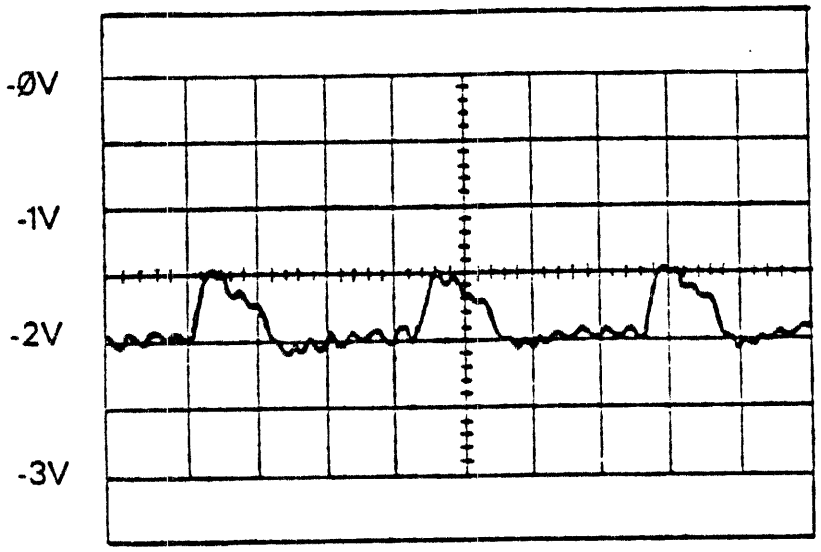


Figure 2-1. Unterminated Wave Form: 68Ω Resistor Missing



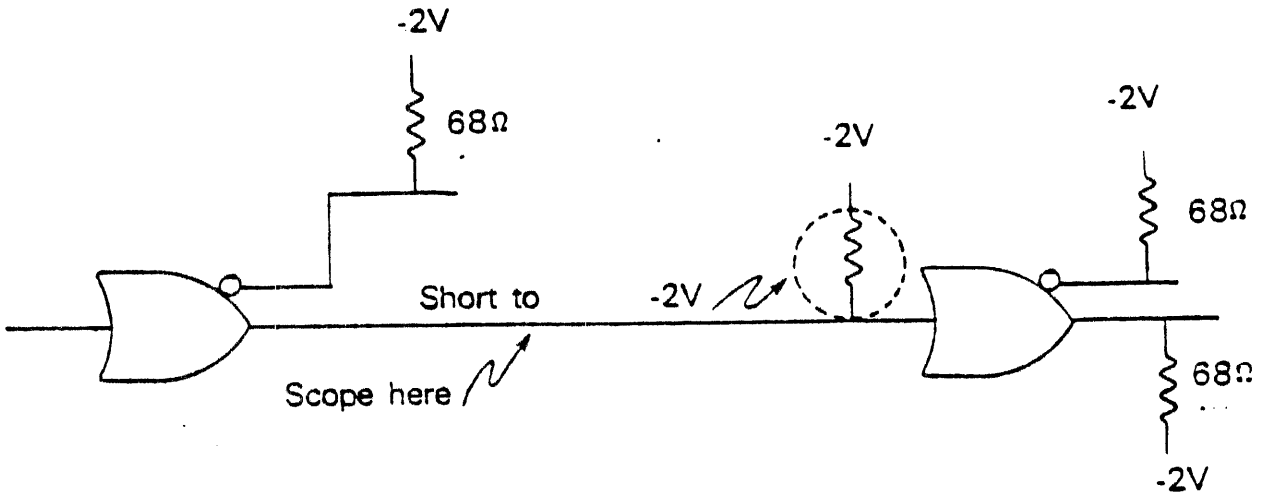
D8 0037

CIRCUIT SHORT TO -2V



HOR = 5Øns/div

Figure 2-2. Positive Going Signal Shorted to -2V



D8 0038

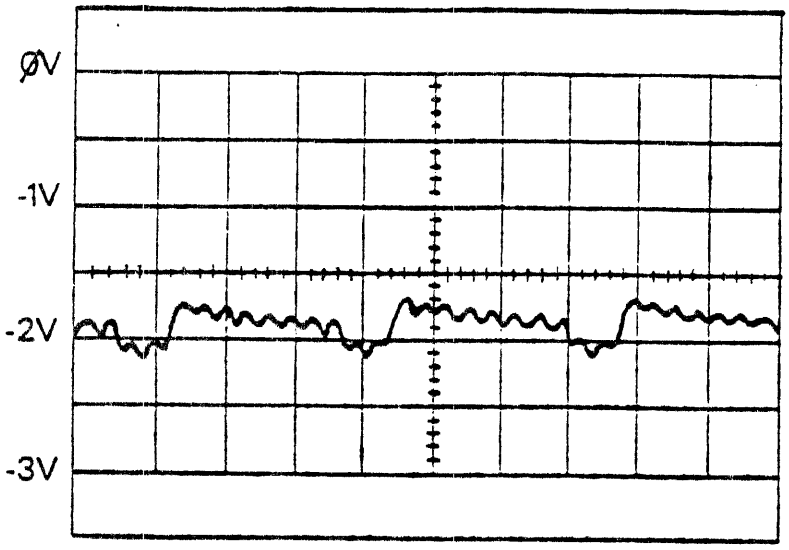
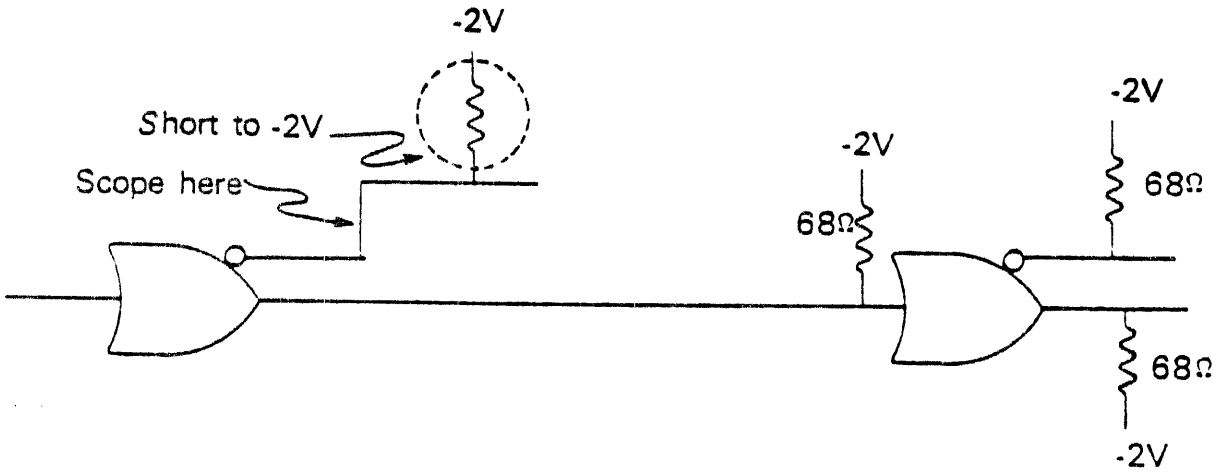


Figure 2-3. Negative Going Signal Shorted to -2V



IMPROPER TERMINATION

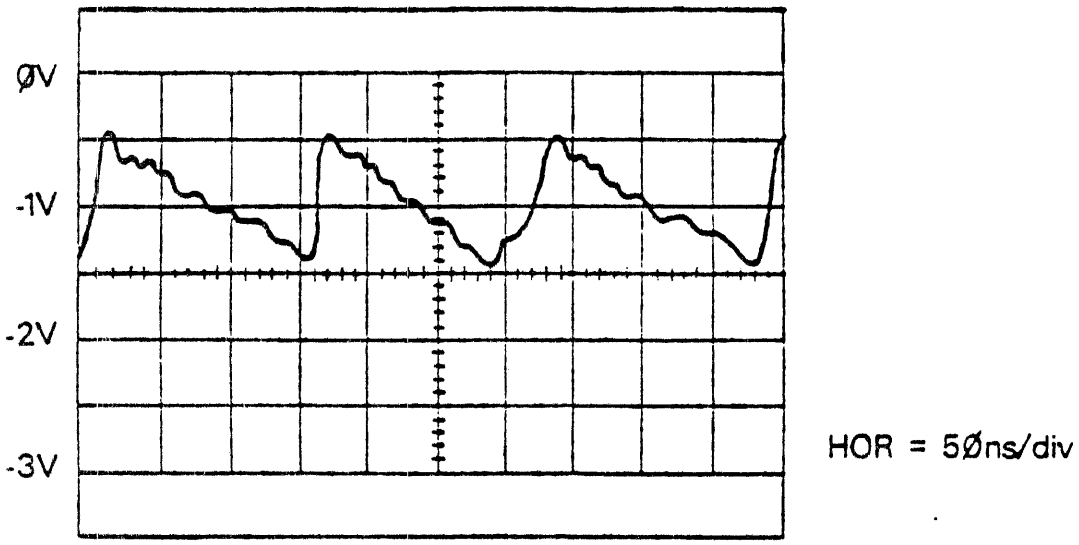
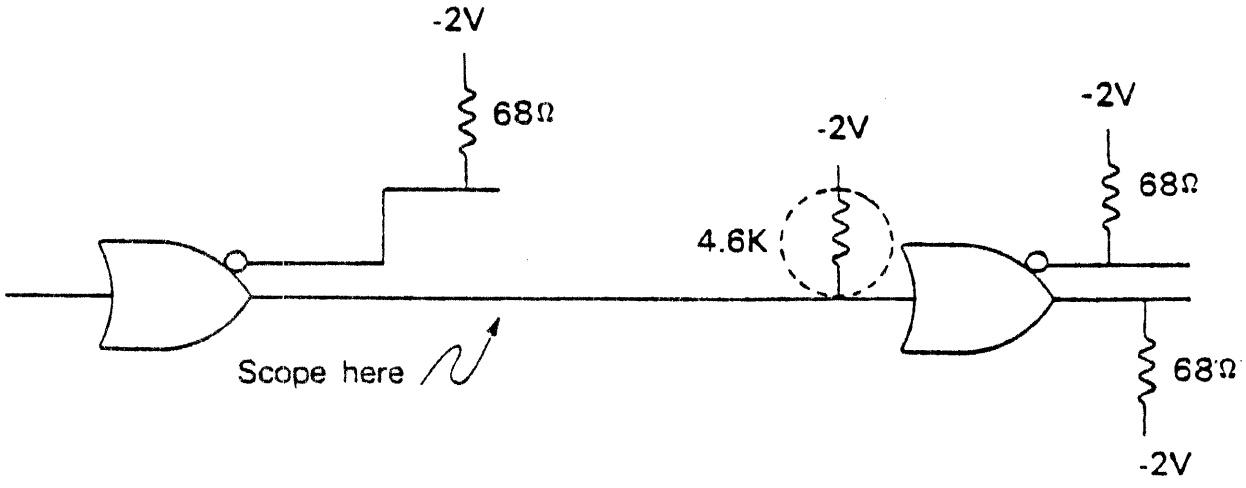


Figure 2-4. Improper Termination: 4.6K Should Be 68Ω

Note that as the value of the resistor increases the signal takes on more characteristics of an unterminated line (Figure 2-1).



08 0040

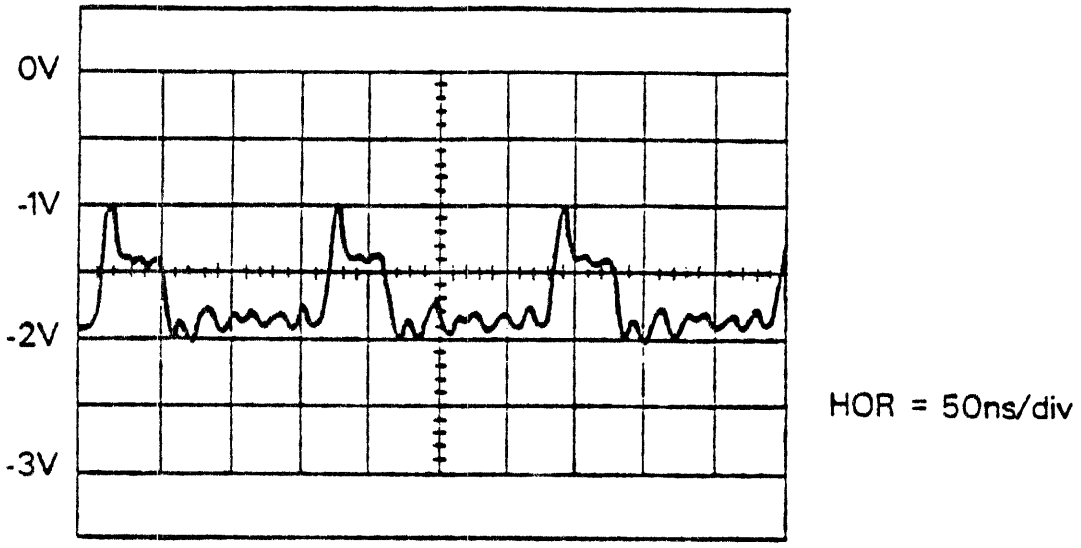
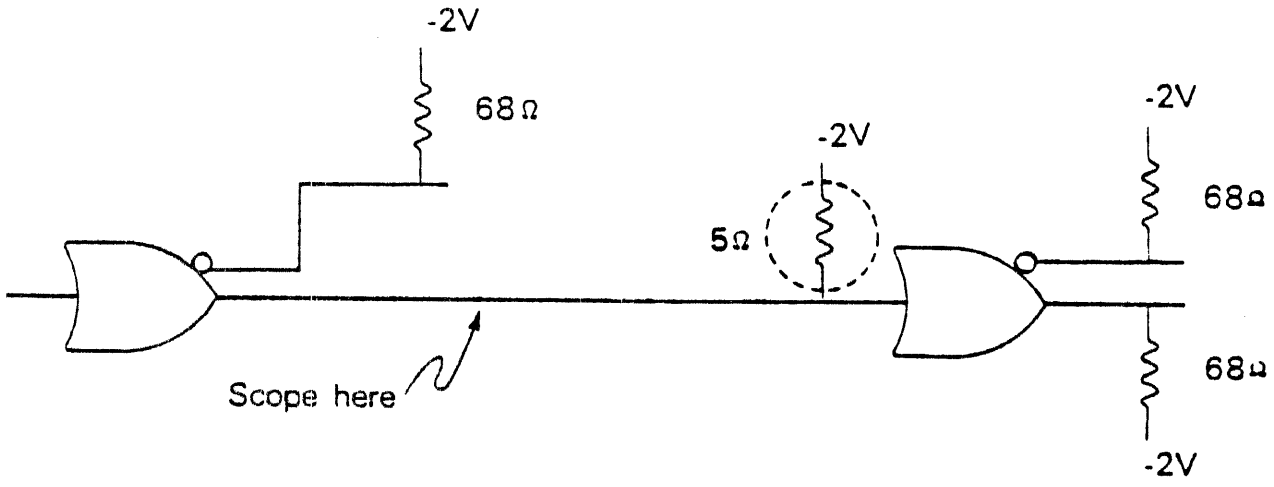


Figure 2-5. Improper Termination: 5Ω Should Be 68

Note that as the value of the resistor decreases, the signal takes on more characteristics of a line shorted to -2V. (figures 2-3 and 2-2)



D8 0041

CAPACITOR IN SIGNAL RUN

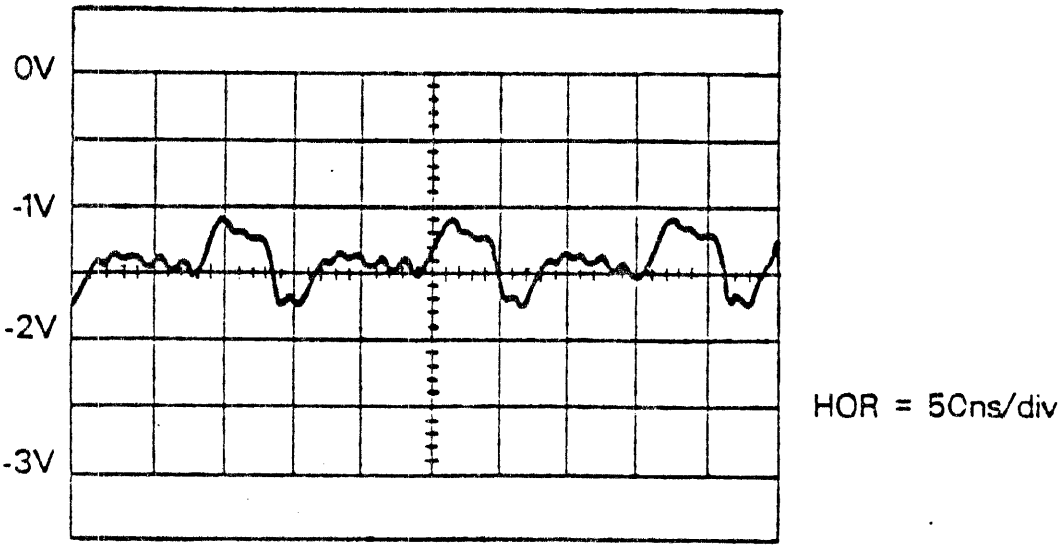
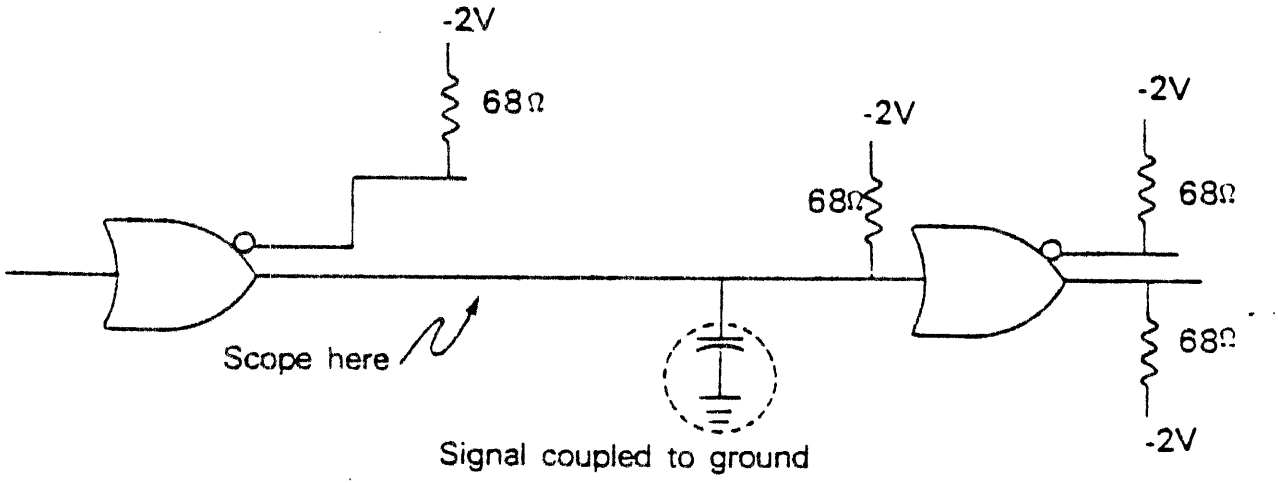


Figure 2-6. A Filter Cap Placed Between a Signal Run and a Voltage Plane

On multiplayer boards it is quite common for a capacitor to be improperly inserted, coupling the signal to a plane. This particular photograph shows a cap between the waveform and ground. The result is nearly identical for other voltage planes as well.



D8 0042

COMPLEMENTARY OUTPUTS SHORTED TOGETHER

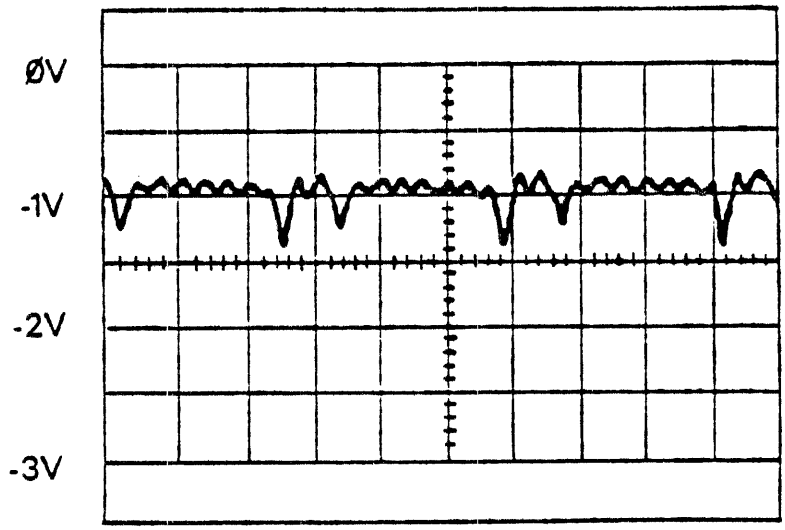
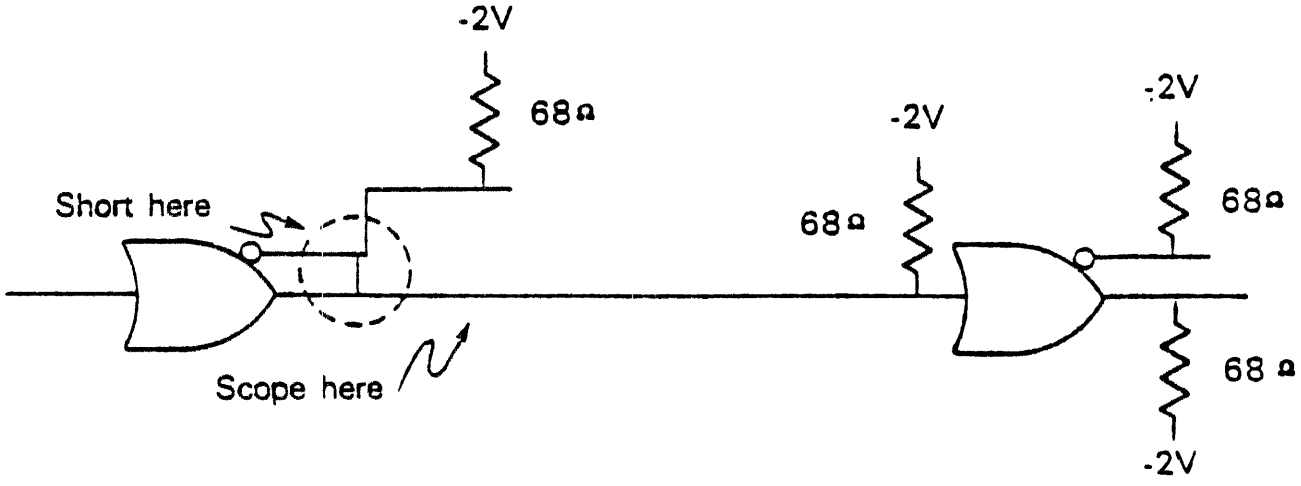


Figure 2-7. Complementary Outputs Shorted Together



OPEN INPUT ETCH

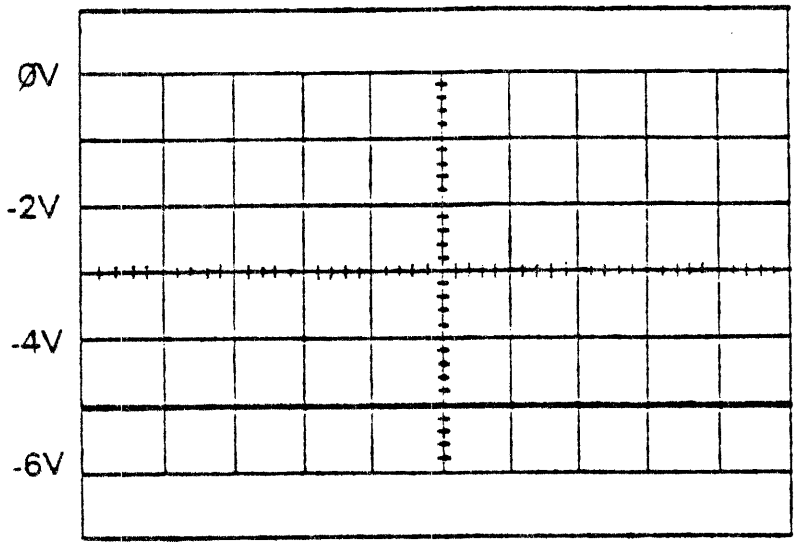
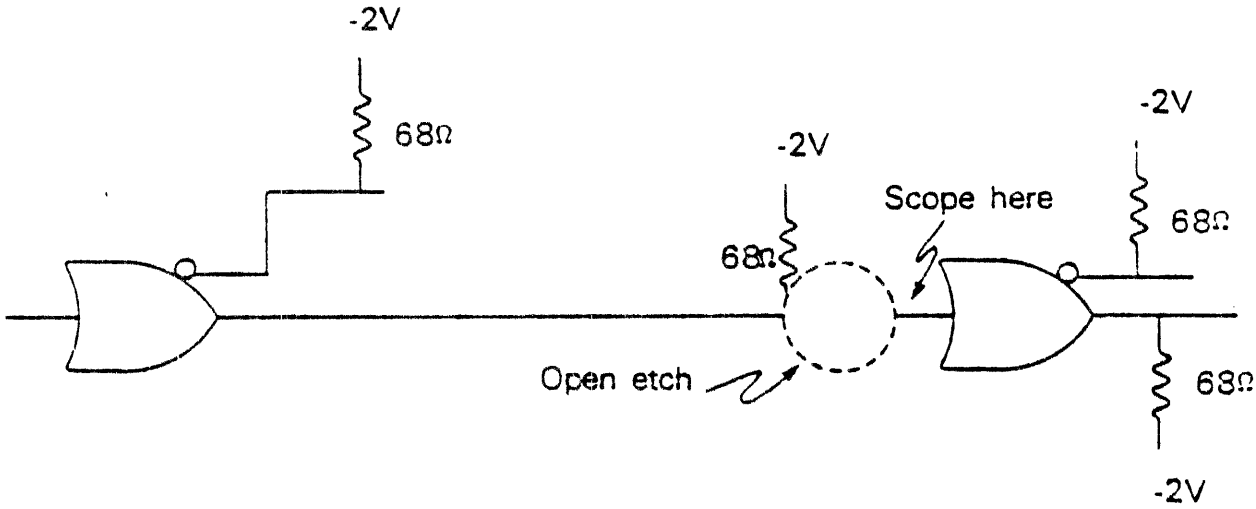


Figure 2-8. Open Input Etch With No Connection to -2V



D8 0043

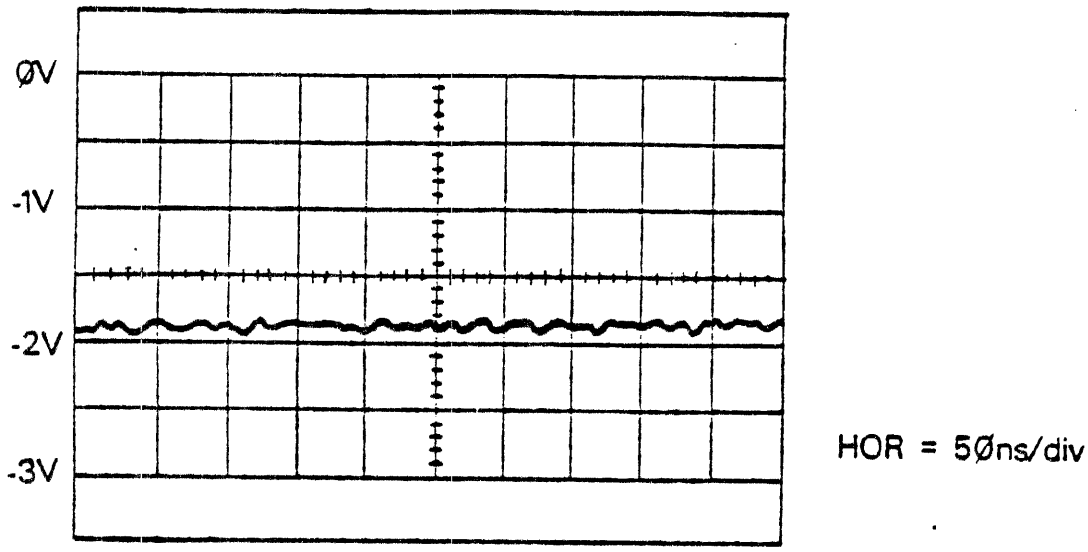
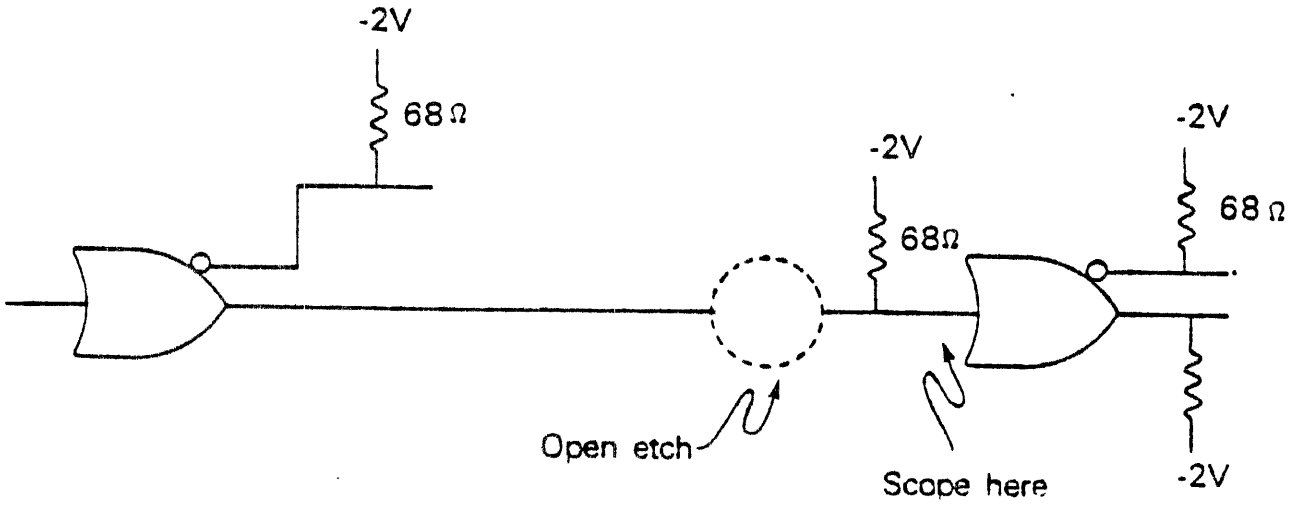


Figure 2-9. Open Input Etch with 68Ω to -2V



D8 0044

SIGNAL SHORTED TO GROUND

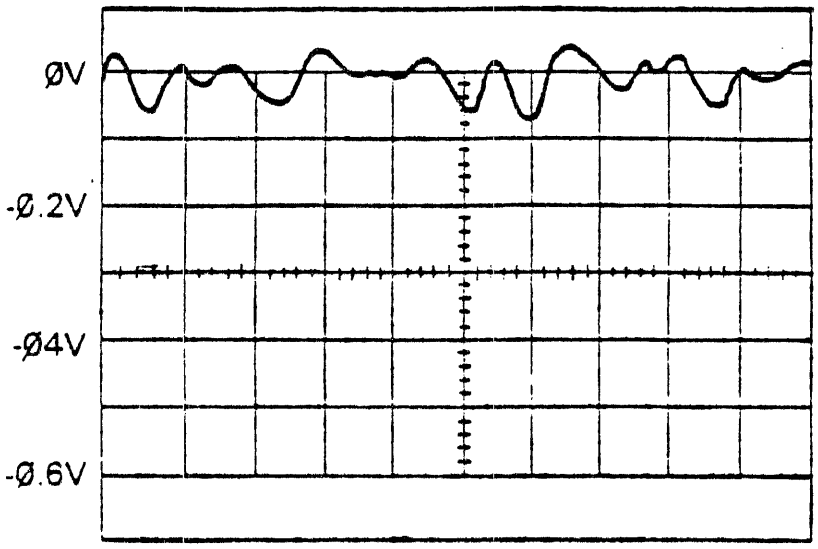
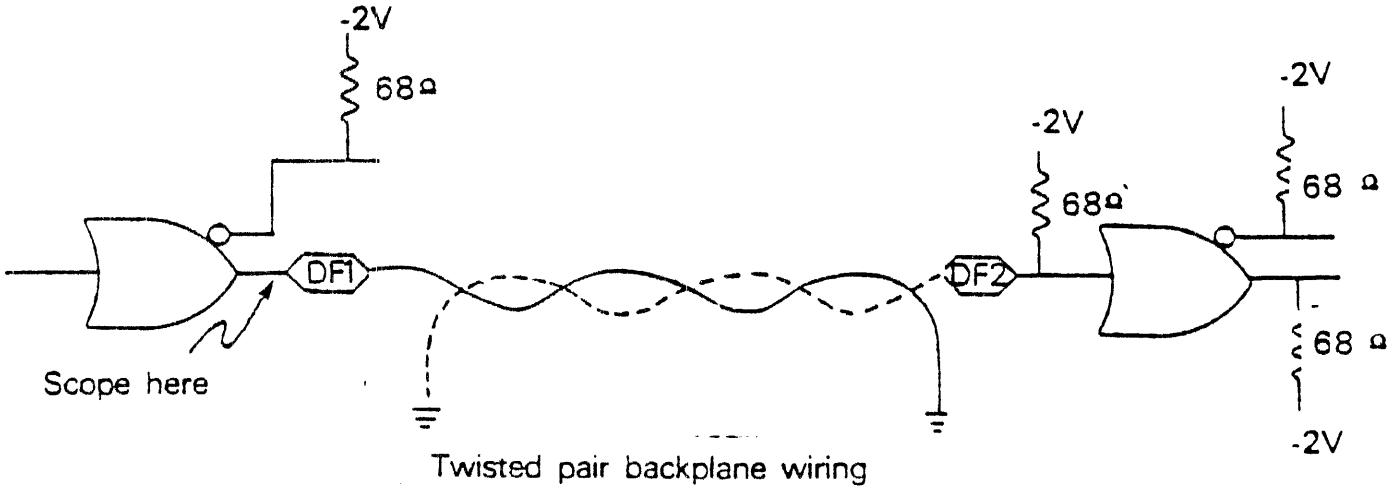


Figure 2-1Ø. Signal Shorted to Ground

Since the backplane wiring is twisted pair with one wire going to ground, it is not unusual to find the wrong wire (ie. the signal) attached to ground.



DR 0045

TWO SHORTED SIGNALS

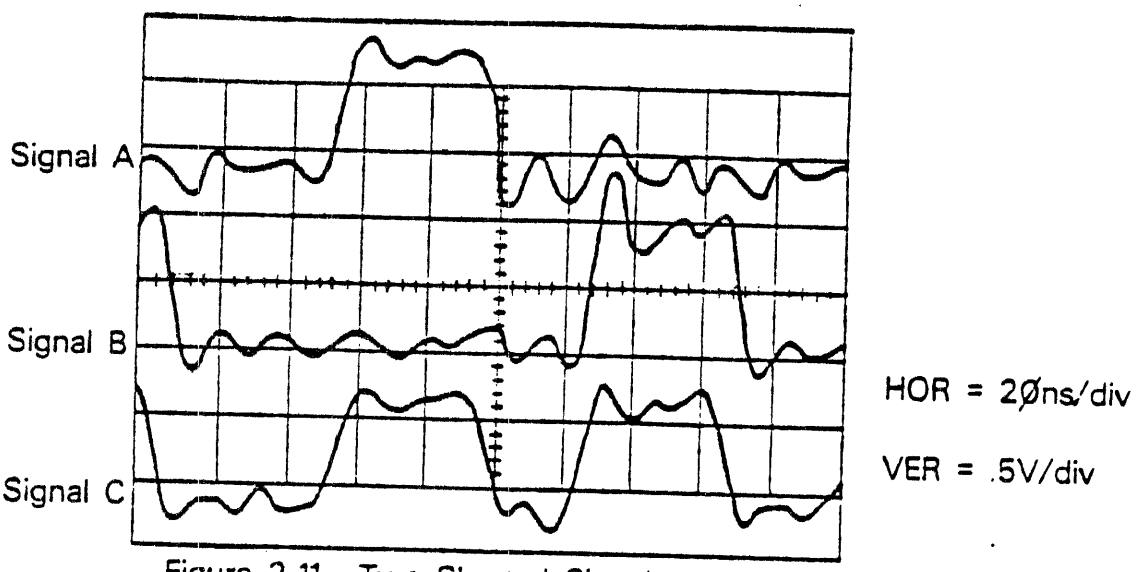


Figure 2-11. Two Shorted Signals 08 0046

One advantage of ECL is that the outputs can be connected in a "wired OR" configuration. This allows designers to reduce the number of packages used in some circuits. However, if two outputs should accidentally short together, a "wired OR" function may result, which can cause confusing waveforms. At first glance, the output of a gate may appear to be good when in fact it is shorted to another signal.

In the illustration, when signal A and signal B are shorted together, the result is signal C. Note that anytime either A or B goes high, the resultant signal also goes high.

The easiest way to determine if a short exists is to simply measure the resistance between the suspected etch and -2 volts. If a short exists, the resistance reading will be less than the reading of a comparable circuit.

SECTION 3

TTL TROUBLESHOOTING GUIDE

GENERAL

Even the most experienced TTL technicians will occasionally encounter problems or waveforms that are foreign to them. "Open collector" chips are, to some extent, responsible for these problems because, unlike more conventional TTL, they require a pull-up to Vcc (+5 volts). Like ECL outputs, the outputs of these gates may be connected in a "wired OR" configuration and thus contain the possibility of shorted outputs. Since a low takes precedent over a high, if the outputs of two open collector gates are shorted together, any time either output is supposed to go low, both will go low.

Problems will also occur if the pull-up resistor is cracked, missing or of the wrong value. Figure 3-2 shows the output of a gate missing this resistor. Since the value of the resistor varies from circuit to circuit, the technician must refer to the schematic of the circuit under test to verify the value of the pull-up resistor. Typically, the resistor will be in the 330 Ohm to 1K range, though extreme variations from these values will cause unusual waveforms. As the value of the resistor increases from the nominal value, the waveform will begin to deviate from the typical waveform (Figure 3-1) and approach the "missing resistor" waveform. Conversely, as the value decreases, the amplitude of the signal approaches Vcc and the base may begin to ride above 0 volts.

The following illustrations indicate some problems common to both conventional and open collector TTL gates. Note that the circuits showing a pull-up apply only to open collector gates, while those with none may be applied to both.

TYPICAL TTL WAVEFORM

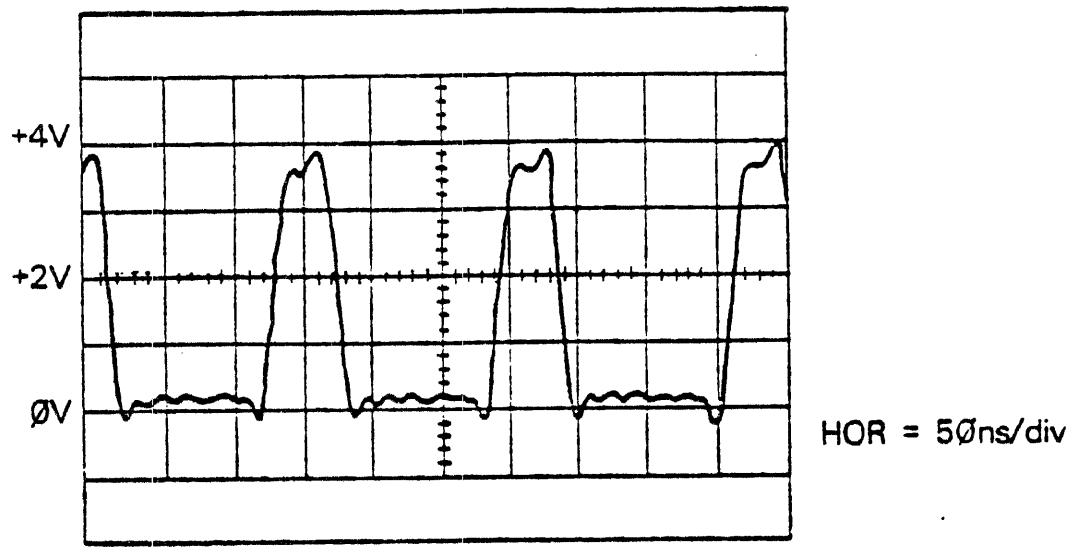
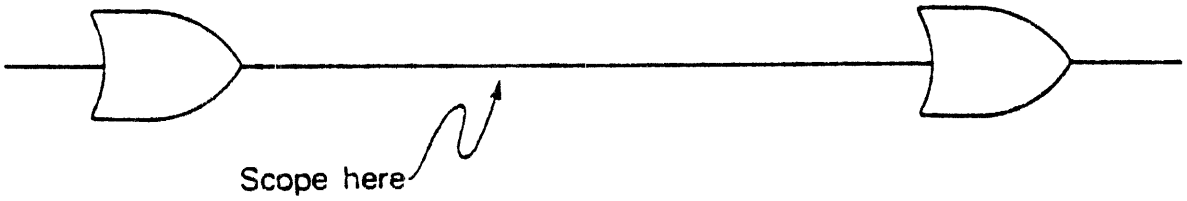


Figure 3-1. Typical TTL Signal



D8 0047

OPEN COLLECTOR OUTPUT WITH NO TERMINATION

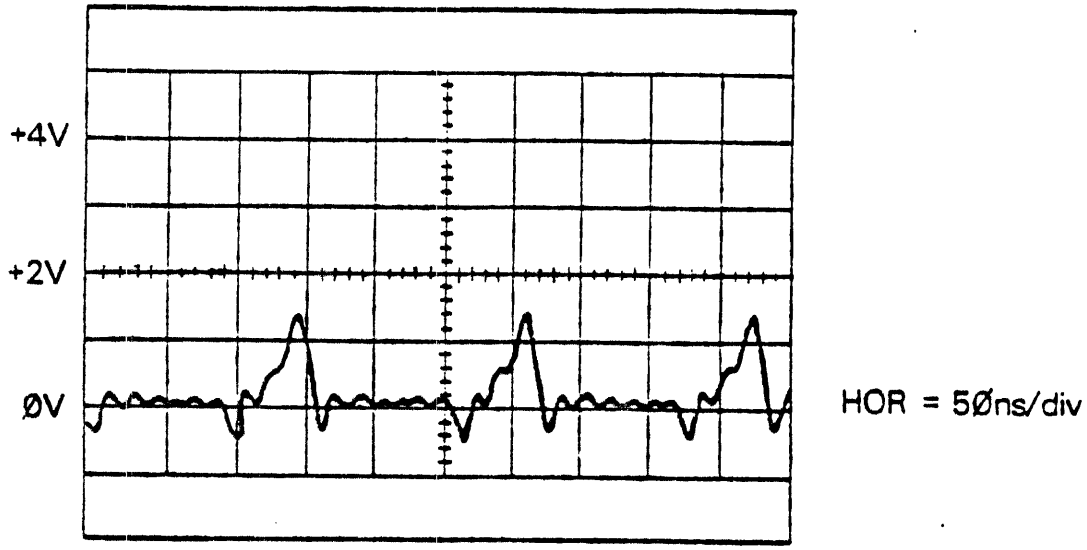
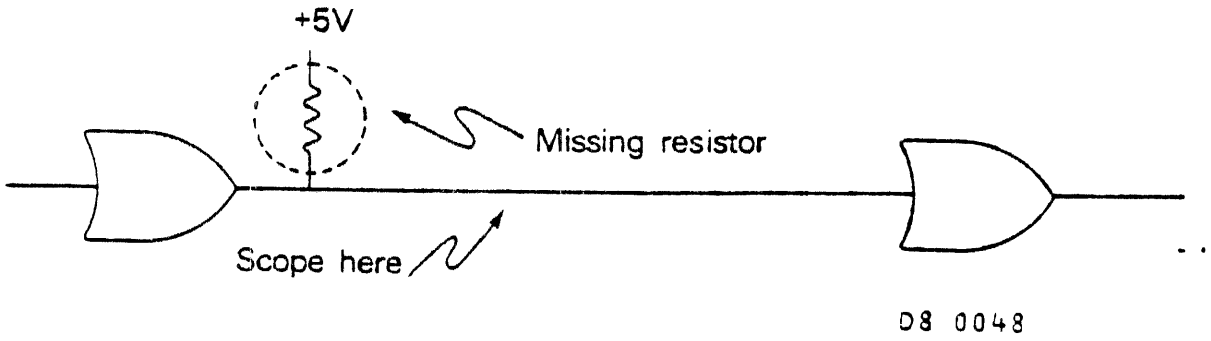


Figure 3-2. Open Collector Output with no Termination



OPEN INPUT ETCH

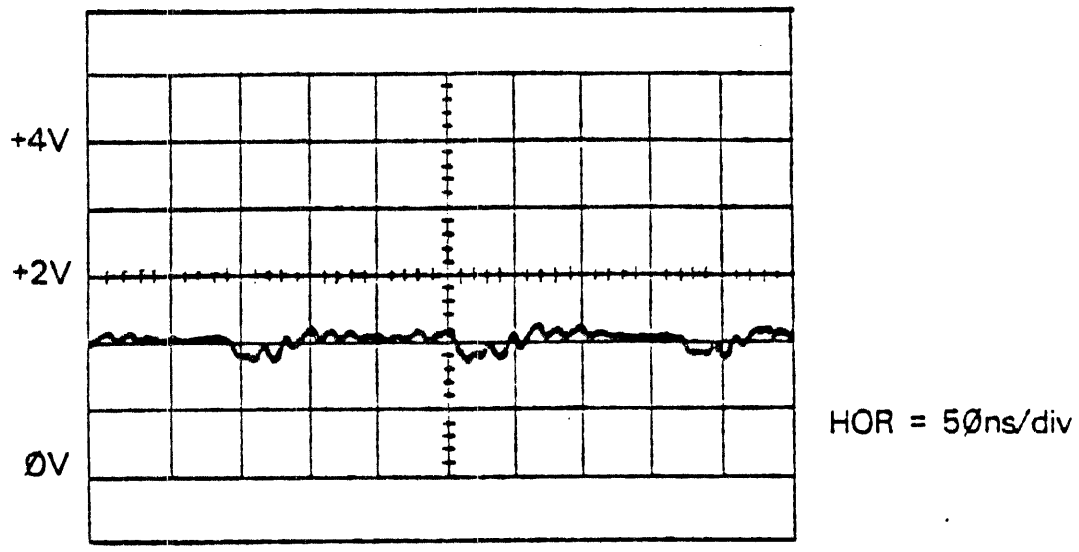
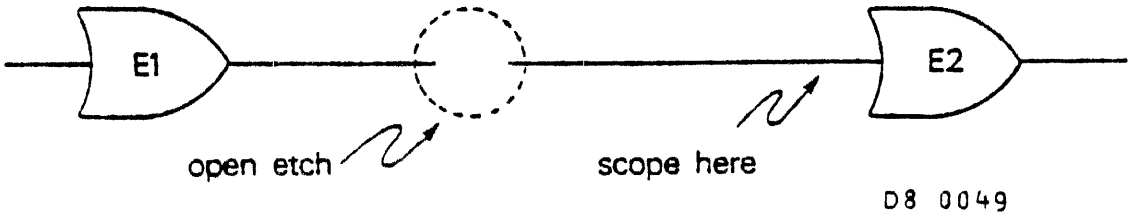


Figure 3-3. Open Input Etch

This signal, commonly called a "float" is a sure sign of an open input etch. The output of E1 would still appear normal. As a technician, you cannot assume that since E1 has a signal on the output, E2 will have a signal on the input.



TWO SHORTED SIGNALS

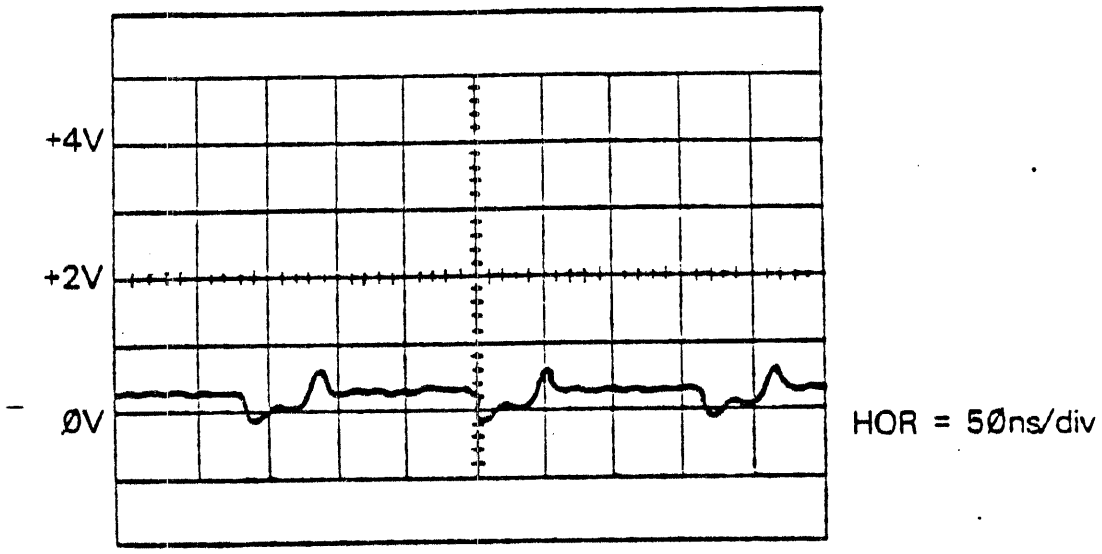
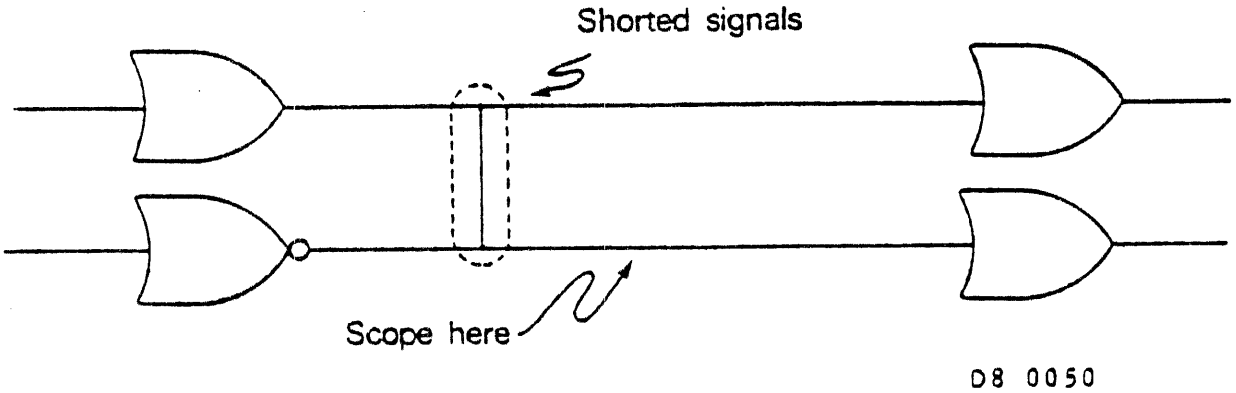


Figure 3-4. Two Signals Shorted Together



CAPACITOR IN SIGNAL RUN

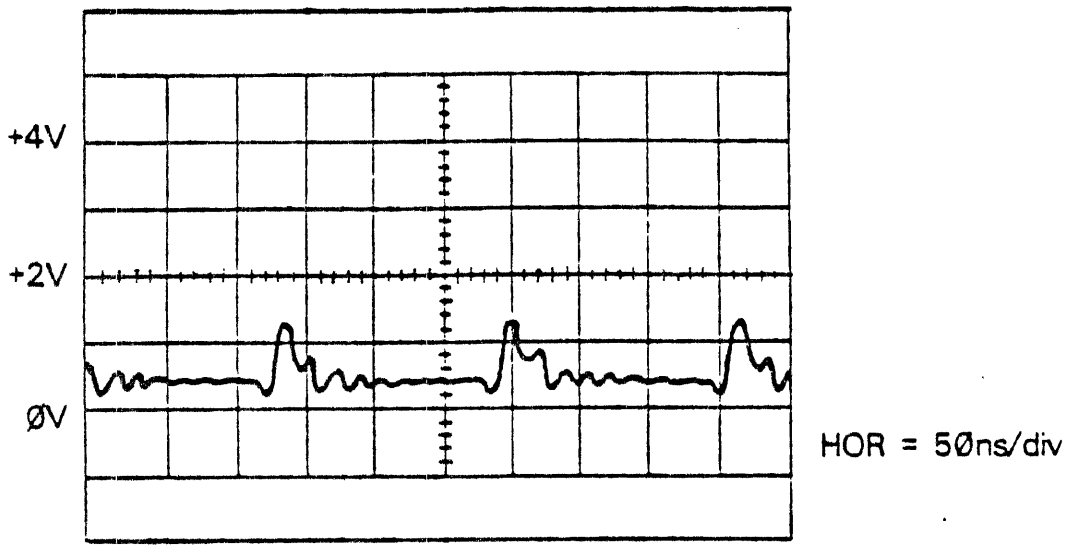
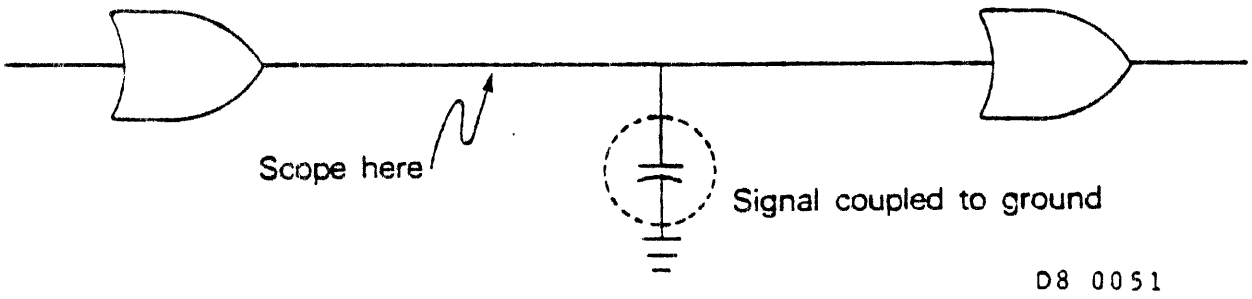


Figure 3-5. A Capacitor Inserted in Signal Run

Occasionally when a board is built,a capacitor will be inserted improperly coupling a signal to ground.Refer to the parts placement page of the schematic of the circuit under test to verify proper placement.



SHORTS TO:

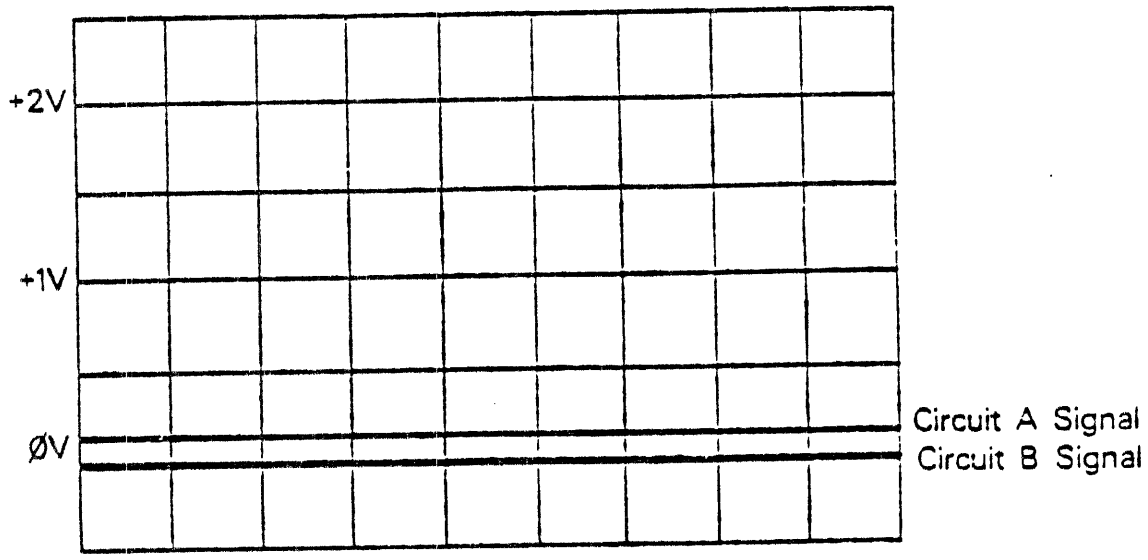
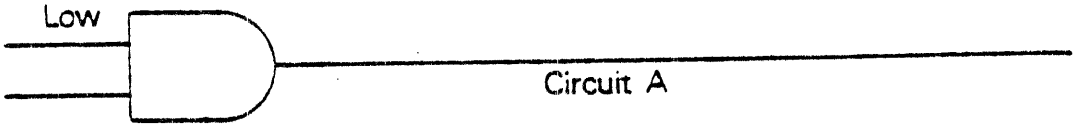
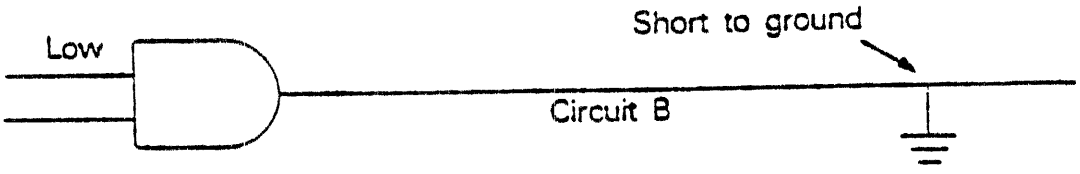


Figure 3.6.
A Signal Shorted to Ground Versus a Steady Logical Low

A steady logical low will generally ride about a tenth of a bolt above ground.



A steady low into an AND gate produces a level slightly above ground.



At first the output of this circuit would appear good. An astute technician would notice the level at a solid ground and suspect a problem.

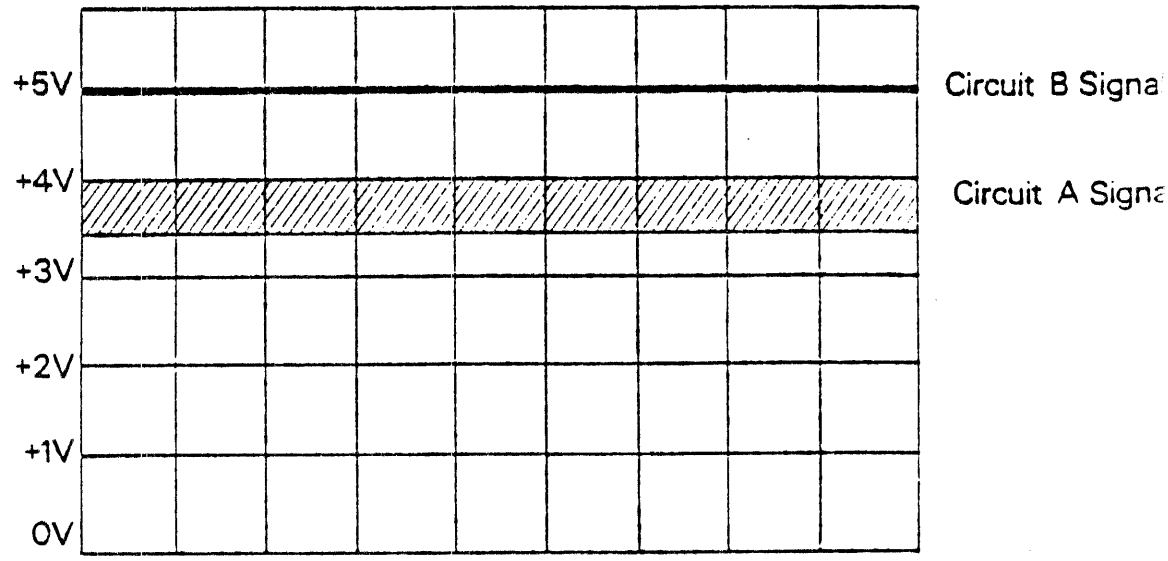
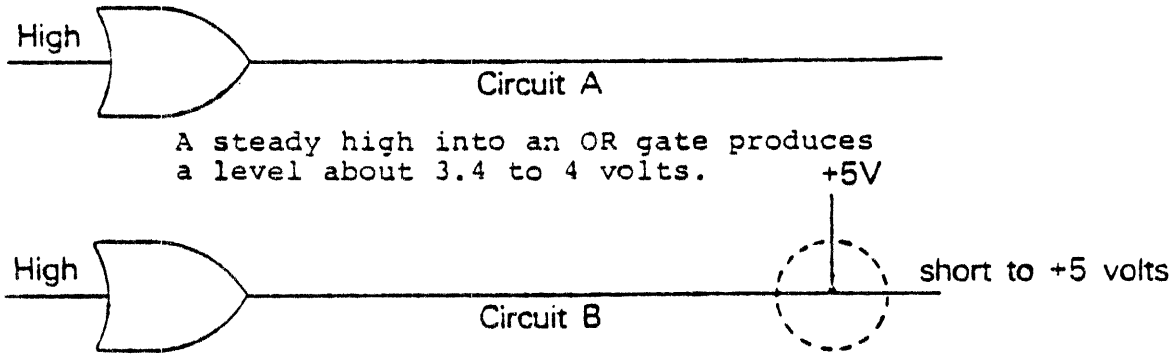


Figure 3-7

A Signal Shorted to +5 Volts Versus a Steady Logical High

A logical high will generally be in the range of 3.4 to 4 volts.

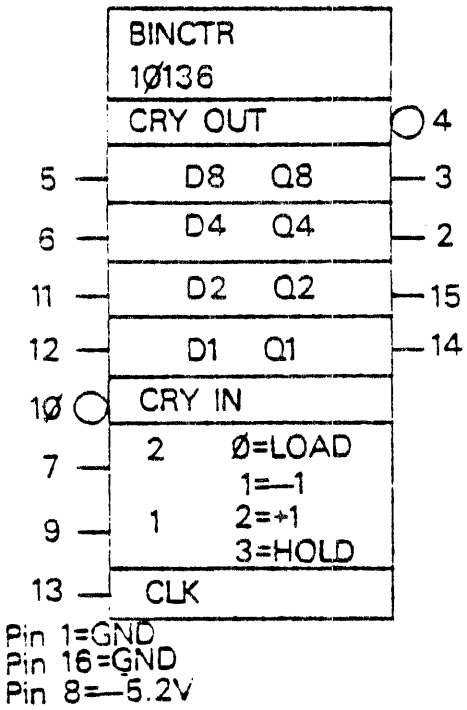


A steady high into an OR gate produces a level about 3.4 to 4 volts.

At first, the output of this circuit would appear good. An astute technician would notice the level at a solid +5 volts and suspect a problem.

SECTION 4

ECL LOGIC SYMBOLOGY



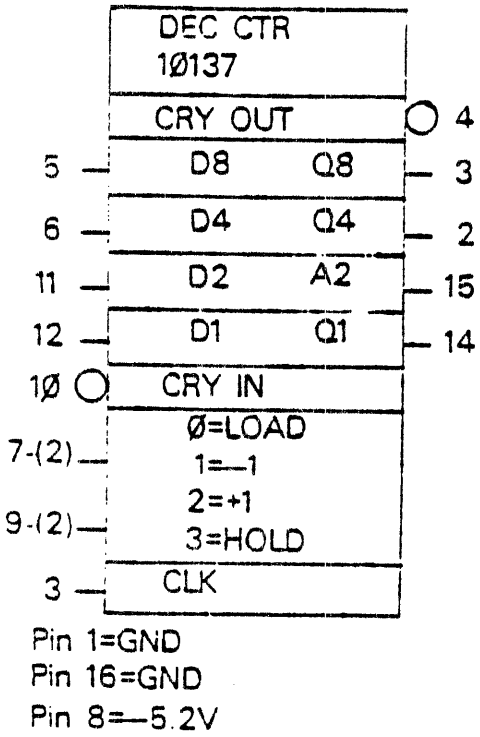
S1	S2	Function Select Table Operation Mode
L	L	Preset
L	H	Increment (count up)
H	L	Decrement (count down)
H	H	Hold (stop count)

1. 1Ø136 UNIVERSAL HEXADECIMAL COUNTER

The 1Ø136 is a high speed synchronous counter that can count up, count down, preset, or stop count. The control lines determine the operation mode of the count. They are S2 (pin 7), S1 (pin 9), and CARRY IN (pin 1Ø).

Note that in the preset mode a clock pulse is necessary to load the counter, and that the information present on the data inputs (D1, D2, D4, D8) will be entered into the counter. CARRY OUT goes low both on the terminal count and when the counter is being preset.

The counter changes state only on the positive going of the clock. Any other input may change at any time except during the positive transition of the clock.



Function Select Table

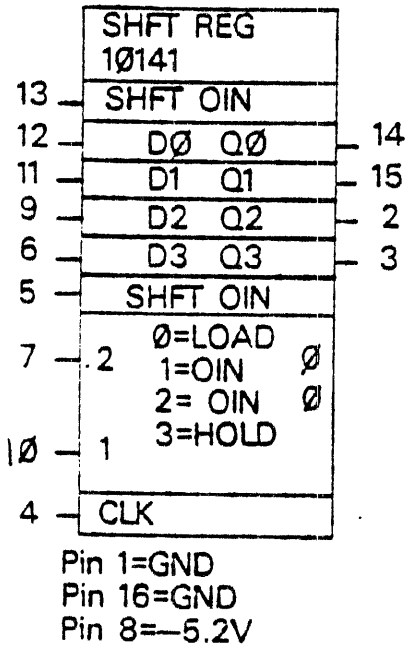
S1	S2	Operating Mode
L	L	Present
L	H	Increment (count up)
H	L	Decrement (count down)
H	H	Hold (stop count)

2. 1Ø137 UNIVERSAL DECADE COUNTER

The 1Ø137 is a high speed synchronous counter that can count up, count down, preset, or stop count. Three control lines determine the operation mode of the counter. They are S1 (pin 9), S2 (pin 7), and CARRY IN (pin 1Ø).

Note that in the preset mode a clock pulse is necessary to load the counter and that the information present on the data inputs (D1, D2, D4, D8) will be entered into the counter. CARRY OUT goes low on the terminal count. The CARRY OUT is partially decoded from Q4 and Q2 directly, so in the preset mode the condition of the CARRY OUT after the clock's positive excursion will depend on the condition of Q1 and/or Q2.

The counter changes state only on the positive going edge of the clock. Any other input may change at any other time except during the positive transition of the clock.

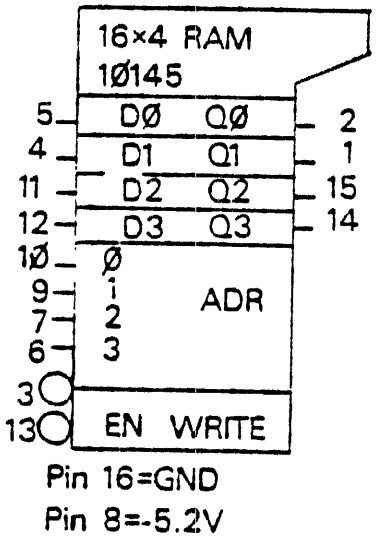


TRUTH TABLE

Select		Operating Mode	Outputs
S1	S2		QØ Q1 Q2 Q3 n+1 n+1 n+1 n+1
L	L	Parallel Entry	DØ D1 D2 D3
L	H	Shift Right	$Q1_n$ $Q2_n$ $Q3_n$ DR_n
H	L	Shift Left	DL $QØ_n$ $Q1_n$ $Q2_n$
H	H	Stop Shift	$QØ_n$ $Q1_n$ $Q2_n$ $Q3_n$

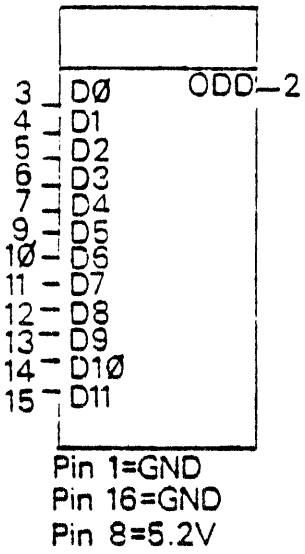
3. 10141 FOUR-BIT UNIVERSAL SHIFT REGISTER

The 10141 is a four-bit universal shift register which performs shift left, or shift right, serial/parallel in, and serial/parallel out operations with no external gating. Inputs S1 (pin 10) and S2 (pin 7) control the four possible operations of the register without external gating of the clock. The flip flops shift information on the positive edge of the clock. The four operations are stop shift, shift left, shift right, and parallel entry of data. The pin 5 input (DR) allows shifting in from the right: pin 13 (DL) allows shifting in from the left.



4. 1Ø145 64-BIT REGISTER FILE (RAM)

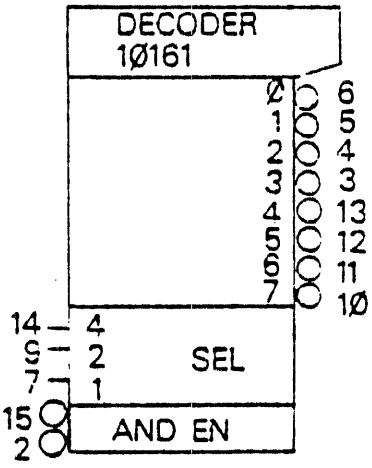
The 1Ø145 is a 64 bit RAM organized as a 16x4 array. The WRITE input when low allows data to be entered when high, disables the data inputs. The CHIP ENABLE input (pin 3), when low, allows full functional operation of the device. When high, it permits all outputs to go to a low level state.



INPUT	OUTPUT
Sum of high Level inputs	Pin 2
EVEN	LOW
ODD	HIGH

5. 10160 12-BIT PARITY GENERATOR - CHECKER

The 10160 consists of nine EXCLUSIVE - OR gates in a single package, internally connected to provide odd parity checking or generation. Output goes high when an odd number of inputs are high. Unconnected inputs are pulled to low logic levels allowing parity detection and generation for less than 12 bits.



Pin 1=GND

Pin 16=GND

Pin 8=5.2V

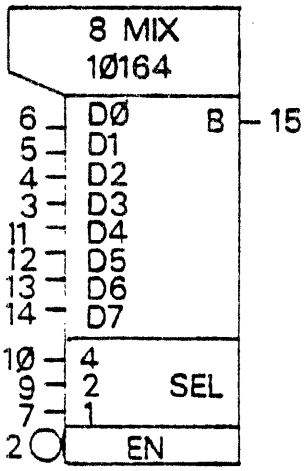
TRUTH TABLE

Enable Input		Inputs(sel)			Outputs						
Pin 15	Pin 2	4	2	1	Q0	Q1	Q2	Q3	Q4	Q5	Q6
L	L	L	L	L	L	H	H	H	H	H	H
L	L	L	L	H	H	L	H	H	H	H	H
L	L	L	H	L	H	H	L	H	H	H	H
L	L	L	H	H	H	H	H	L	H	H	H
L	L	H	L	L	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	L	H
L	L	H	H	L	H	H	H	H	H	H	L
L	L	H	H	H	H	H	H	H	H	H	H
H	Ø	Ø	Ø	Ø	H	H	H	H	H	H	H
Ø	H	Ø	Ø	Ø	H	H	H	H	H	H	H

Ø = Don't Care

6. 1Ø161 BINARY TO 1-8 DECODER (LOW)

The 1Ø161 is designed to decode a 3-bit input to a one of eight line output. The selected output will be low while all other outputs will be high. The enable inputs, when either or both are high, all output high.



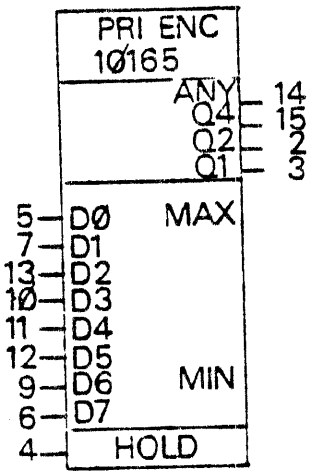
TRUTH TABLE

ENABLE	ADDRESS INPUTS (SEL)			B
	4	2	1	
L	L	L	L	DØ
L	L	L	H	D1
L	L	H	L	D2
L	L	H	H	D3
L	H	L	L	D4
L	H	L	H	D5
L	H	H	L	D6
L	H	H	H	D7
H	Ø	Ø	Ø	L

Ø = Don't Care

7. 1Ø164 8 - LINE MULTIPLEXER

The 1Ø164 is an eight-channel data selector which routes data present at one-of-eight inputs to the output. The data is routed according to the three bit code present on the address (select) inputs.



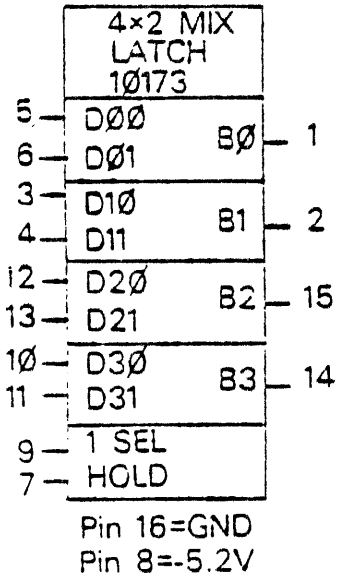
TRUTH TABLE

DATA INPUTS								OUTPUTS			
D0	D1	D2	D3	D4	D5	D6	D7	ANY	Q4	Q2	Q1
H	Ø	Ø	Ø	Ø	Ø	Ø	Ø	H	L	L	L
L	H	Ø	Ø	Ø	Ø	Ø	Ø	H	L	L	H
L	L	H	Ø	Ø	Ø	Ø	Ø	H	L	H	L
L	L	L	H	Ø	Ø	Ø	Ø	H	L	H	H
L	L	L	L	H	Ø	Ø	Ø	H	H	L	L
L	L	L	L	L	H	Ø	Ø	H	H	L	H
L	L	L	L	L	L	H	Ø	H	H	H	L
L	L	L	L	L	L	L	H	H	H	H	H
L	L	L	L	L	L	L	L	L	L	L	L

8. 10165 8 - INPUT PRIORITY ENCODER

The 10165 is a device designed to encode eight inputs to a binary coded output. The output code is that of the highest order input. Any input of lower priority is ignored. Each output incorporates a latch allowing synchronous operation. When the HOLD is low, the outputs follow the inputs and latch when the clock goes high.

The input is active when high, (e.g. the three binary outputs are low when input D0 is high). the ANY output is high when any input is high.



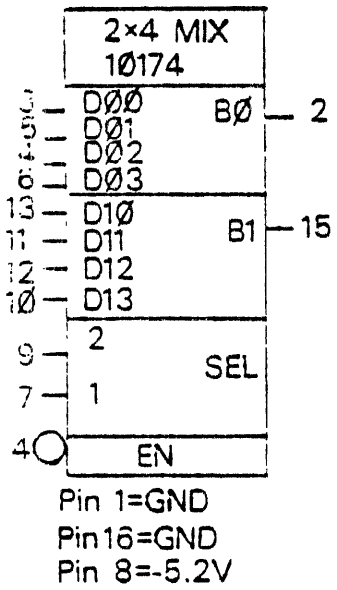
TRUTH TABLE

SELECT	HOLD	$B\emptyset_{n+1}$
H	L	DØØ
L	L	DØ1
Ø	H	$B\emptyset_n$

Ø = Don't Care

9. 1Ø173 QUAD 2 - INPUT MULTIPLEXER/LATCH

The 1Ø173 is a quad two-channel multiplexer with latch. It incorporates common HOLD and common data select inputs. The SELECT input determines which data input is enabled. A high level enables data inputs D1Ø, D2Ø, D3Ø, and a low level enables data inputs DØ1, D11, D21, D31. Any change on the data input will be reflected at the outputs while the HOLD is low. The outputs are latched on the positive transition of the HOLD. While the HOLD is in the high state, a change in the information present at the data inputs will not affect the output information.

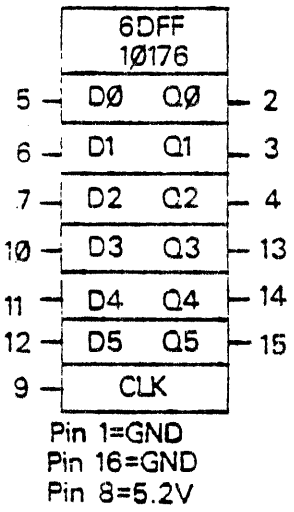


TRUTH TABLE

EN	SELECT		OUTPUTS	
	2	1	BØ	B1
H	Ø	Ø	L	L
L	L	L	DØØ	D1Ø
L	L	H	DØ1	D11
L	H	L	DØ2	D12
L	H	H	DØ3	D13

10. 10174 Dual 4 to 1 Multiplexer

The 10174 is a high speed dual channel multiplexer with output enable capability. The SEL input determines one of four active data inputs for each multiplexer. The output enable (EN) forces both outputs low when in the high state.



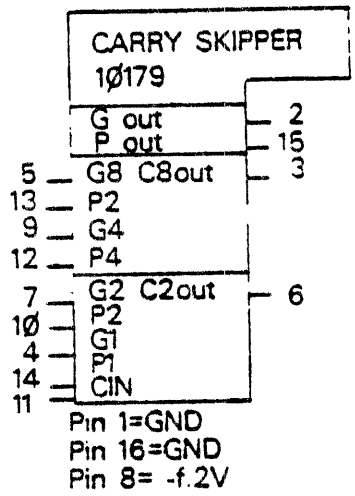
CLOCKED TRUTH TABLE

CLK	Ø	Qn+1
L	Ø	Qn
H*	L*	L
H*	H	H

Ø = Don't Care
* A clock H is a clock transition from a low to a high state.

11. 1Ø176 HEX "D" MASTER-SLAVE FLIP-FLOP

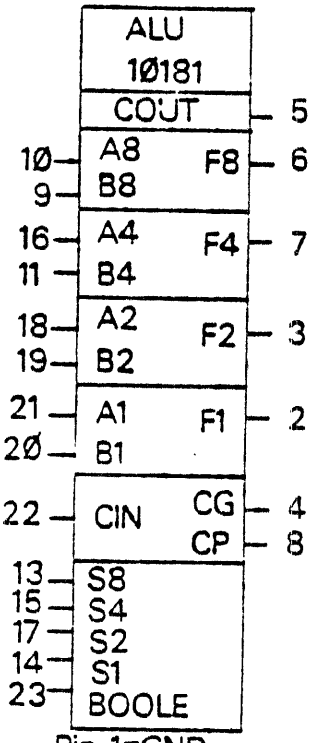
The 1Ø176 contains six high-speed, master-slave type "D" flip-flops. Clocking is common to all six flip-flops. Data is entered into the master when the clock is low. Master to slave data transfer takes place on the positive-going CLK transition. Thus, outputs may change only a positive-going clock transition. A change in the information present at the data (D) input will not affect the output information any other time due to the master-slave construction of this device.



12. 1Ø179 LOOK-AHEAD CARRY BLOCK

The 1Ø179 device has 12 low power gates internally connected to perform the look-ahead carry function.

P out = P1+P2+P4+P8
G out = (G1+P2+P4+P8) (G2+P4+P8) (G4+P8) G8
C2out = (CIN+P1+P2) (G1+P2) G2
C8out = (CIN+P1+P2+P4+P8) (G1+P2+P4+P8) (G2+P4+P8) (G4+P8)G8



13. 10181 4-BIT ARITHMETIC LOGIC UNIT/ FUNCTION GENERATOR

The 10181 is a high speed arithmetic logic unit capable of performing 16 logic operations and 16 arithmetic operations on two four-bit words. Full internal carry is incorporated for ripple through operation.

Arithmetic logic operations are selected by applying the appropriate binary word to the select inputs (S1 through S8) as indicated in the tables of arithmetic/logic functions. Group carry propagate (P_G) and carry generate (G_G) as provided allow fast operations on very long words using a second order look ahead. The internal carry is enabled by applying a low level voltage to the mode control input (BOOLE).

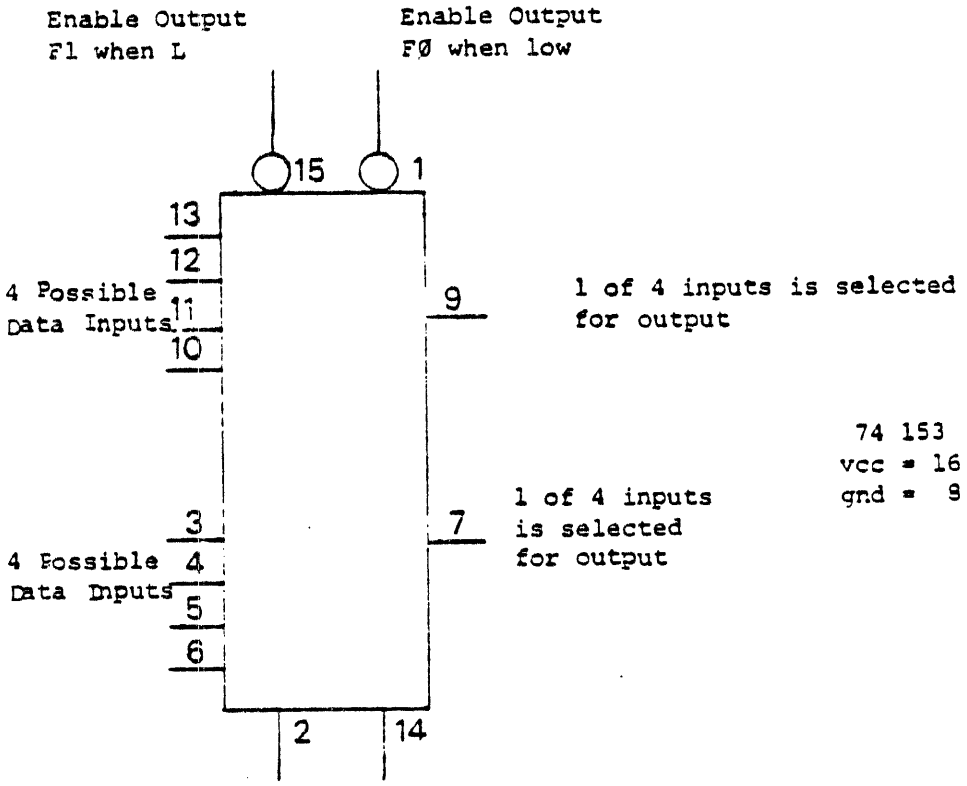
TABLE OF ARITHMETIC/LOGIC FUNCTIONS

FUNCTION SELECT S8	S4	S2	S1	LOGIC FUNCTIONS BOOLE IS HIGH	ARITHMETIC OPERATION. IF BOOLE IS LOW, CIN OF LSB MUST BE.
L	L	L	L	F = A	F = A minus 1
L	L	L	H	F = A + B	F = A plus (A+B)
L	L	H	L	F = A . B	F = A plus (A+B)
L	L	H	H	F = logical "0"	F = A times 2
L	H	L	L	F = A . B	F = (A.B) minus 1
L	H	L	H	F = B	F = (A.B) plus (A+B)
L	H	H	L	F = A + B	F = A plus B
L	H	H	H	F = A . B	F = A plus (A.B)
H	L	L	L	F = A + B	F = (A.B) minus 1
H	L	L	H	F = A . B	F = A minus B minus 1
H	L	H	L	F = B	F = (A . B) plus (A+B)
H	L	H	H	F = Logical "1"	F = minus 1 (2's complement)
H	H	L	H	F = A + B	F = (A+B) plus 0
H	H	H	L	F = A + B	F = (A+B) plus 0
H	H	H	H	F = A	F = A plus 0

SECTION 5

TTL LOGIC SYMBOLOGY

DUAL 4-LINE-TO-1-LINE DATA SELECTOR/MULTIPLEXER



TRUTH TABLE

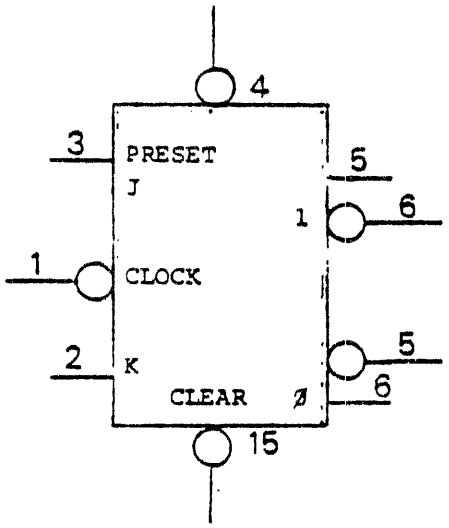
S1	SØ	F1	FØ
Ø	Ø	A1	AØ
Ø	1	B1	BØ
1	Ø	C1	CØ
1	1	D1	DØ

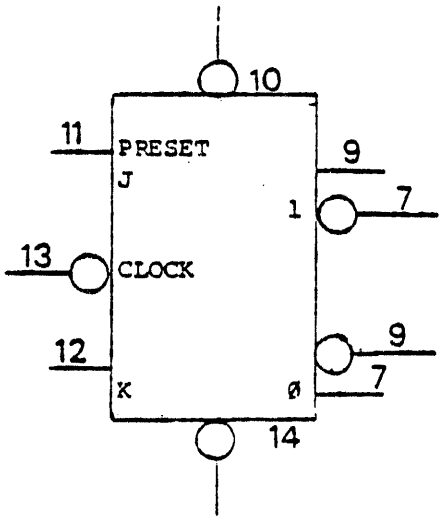
If STB input is false = (H) then the output (F) of the multiplexer will be low (L).

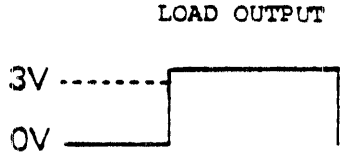
JK Master - Slave Flip Flop

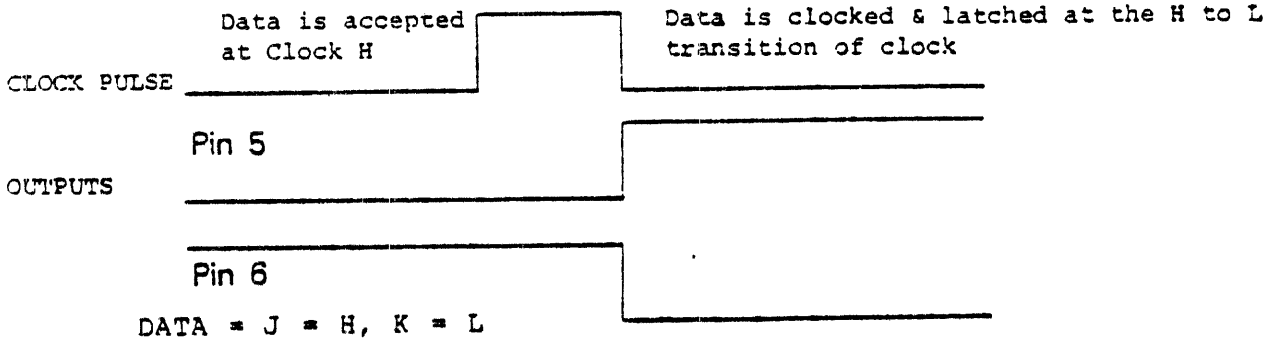
74 76
74 112

vcc = 16
gnd = 8









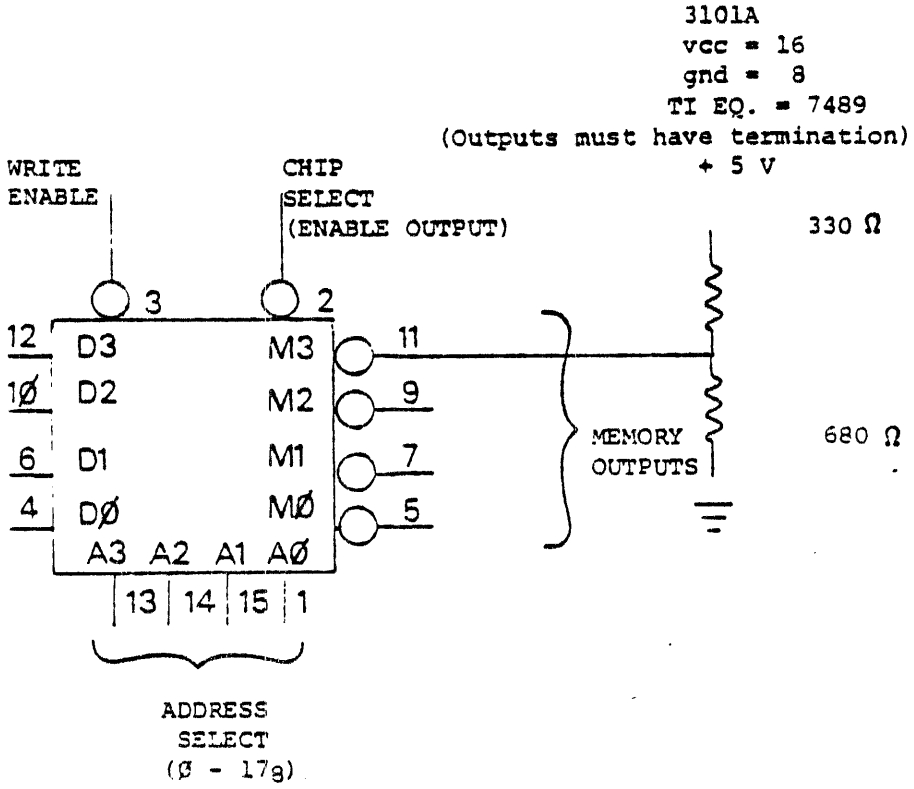
1. If both J and K are +3v and the clock keeps running, each pulse of the clock complements the outputs.

2. If J = H K = L at clock time, the flop sets pin 5 = H pin 6 = L.

3. If J = L K = H at clock time, the flop clears pin 5 = L pin 6 = H.

The J-K master-slave flip-flop is so called because it takes a full clock cycle to change the outputs. At the L H transistion of the clock, the data is loaded into the flop (master), but the outputs have not changed. At the H L transistion the output (slave) is loaded and will now reflect the data that was clocked in the master.

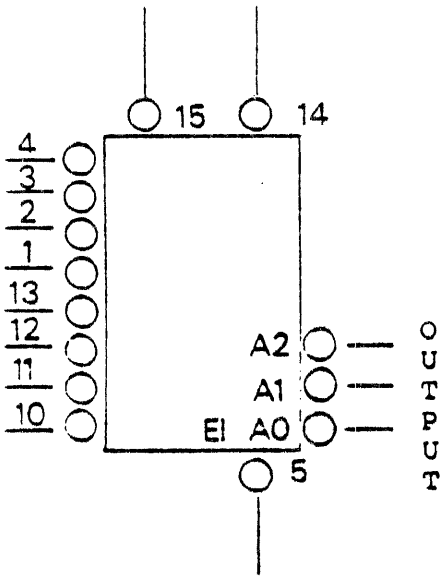
4 BIT X 16 ADDR. RANDOM ACCESS MEMORY (RAM)



Memory Enable	Write Enable	Operation	Outputs
Ø	Ø	Write	Logical '1' States Compliment of Stored Data
Ø	Ø	Read	
1			Logical '1' States

Outputs not used
on 11/45

8 LINE -TO- 3 LINE PRIORITY ENCODER



EI	0	1	2	3	4	5	6	7	GS	A_0	A_1	A_2	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	X	L	H	H	L	L	H	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	L	L	L	H	H
L	X	X	L	H	H	H	H	H	L	H	L	H	H
L	X	L	H	H	H	H	H	H	L	L	H	H	H
L	L	H	H	H	H	H	H	H	L	H	H	H	H

H = High Voltages level

L = Low Voltages level

X = Don't Care

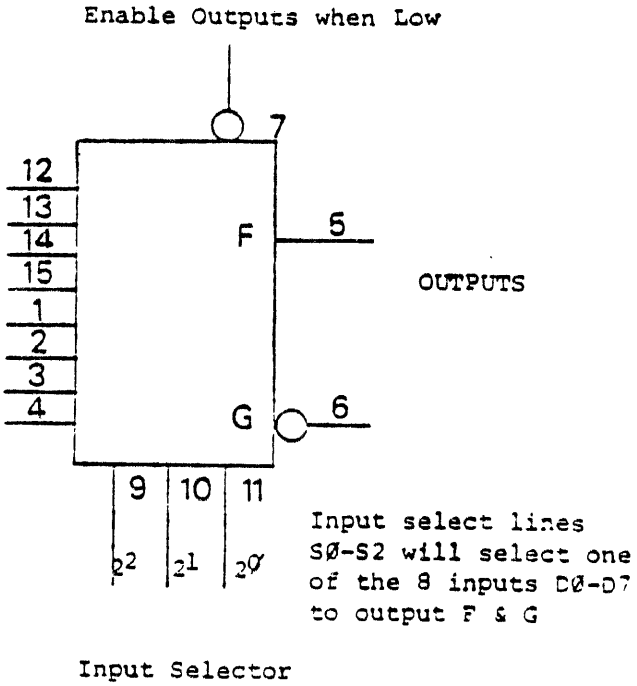
9318
VCC = 16
gnd = 8

Any one line at input will cause the corresponding binary to appear only when EI is low.

Example:

INPUT DECIMAL	BIN. OUTPUT A2	A1	AØ
6 pin 3	1	1	Ø
3 pin 13	Ø	1	1

8 TO 1 DATA SELECTOR MULTIPLEXER



74 151
vcc = 16
gnc = 8

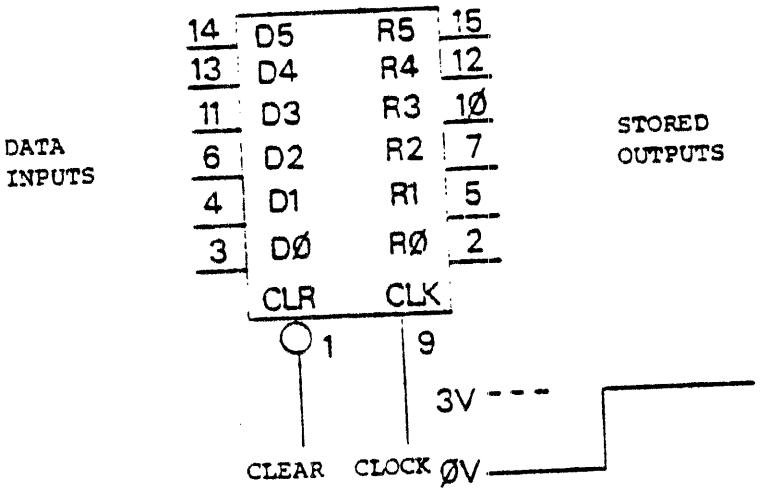
CONTROL INPUTS			STROBE	OUTPUT
SØ	S1	S2	G	F
LOW	LOW	LOW	LOW	DØ
HIGH	LOW	LOW	LOW	D1
LOW	HIGH	LOW	LOW	D2
HIGH	HIGH	LOW	LOW	D3
LOW	LOW	HIGH	LOW	D4
HIGH	LOW	HIGH	LOW	D5
LOW	HIGH	HIGH	LOW	D6
HIGH	HIGH	HIGH	LOW	D7
DON'T CARE			HIGH	HIGH

TRUTH TABLE

With the SØ-S2 line set the chip will select one of eight lines to be output to F when STB is low.

H D-TYPE FLOPS WITH CLEAR

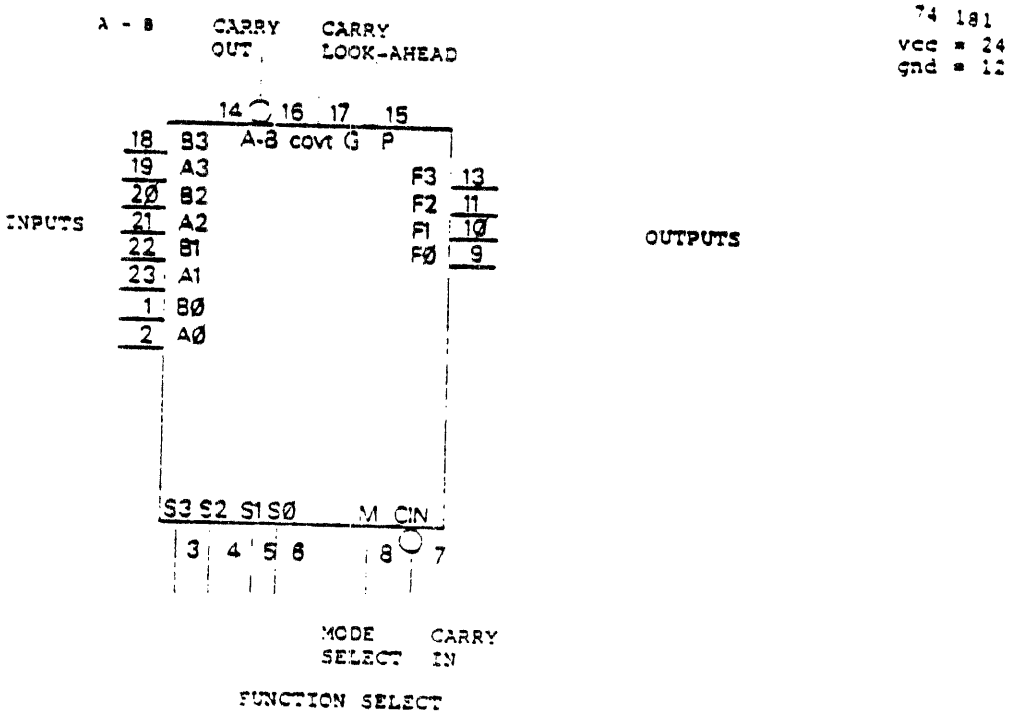
74 174
VCC = 16
gnd = 8



Low to high transition causes output to reflect input and stores input to hold output constant when the input changes.

(Commonly used as temporary)
(Storage registers)

ARITHMETIC LOGIC UNIT/FUNCTION GENERATOR (ALU)



74S181-ALU

S3	S2	S1	S0	M = H LOGIC FUNCTIONS	M = L ARITHMETIC OPERATIONS C_N = 0 = H	C_N = 1 = L
L	L	L	L	1 F = A	F = A	F = A PLUS
L	L	H	H	2 F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	H	L	3 F = A+B	F = A MINUS B MINUS 1	F = MINUS B
L	H	H	H	4 F = AB	F = AB MINUS 1	F = AB
H	L	L	H	5 F = A+B	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	6 F = B	F = (A+B) PLUS AB	F = A+B PLUS AB PLUS 1
H	L	H	H	7 F = AB	F = AB MINUS 1	F = AB
H	H	L	L	8 F = 1	F = A PLUS A	F = A PLUS A PLUS 1
H	H	H	L	9 F = A+B	F = (A+B) PLUS A	F = A+B PLUS A PLUS 1
H	H	H	H	10 F = A	F = A MINUS 1	F = A

The 74 181 is a chip that can look at two sets of 4-bit data and perform any one of twenty-four different functions (12 arithmetic and 12 logic). The following table lists the functions that one commonly used in the 11/45.